Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Machine learning & AI

Úvod do umělé inteligence

Ing. Martin Pozdílek, Ph.D.

1 Inteligence

V tomto kurzu se budeme zabývat umělou inteligencí a strojovým učením. Velmi těžko se lze vyhnout přirozené, biologické inteligenci, která je předobrazem té umělé. Ačkoliv se převážně budeme zabývat tématem inteligence z pohledu přírodních věd, nemůžeme se vyhnout filozofii nebo etice.

Otázky myšlení, inteligence nebo existence lidskou rasu vždy fascinovaly. Co je to život? Kde se vzal život? Jaký je smysl života? Odlišuje se člověk od ostatních živočichů a případně čím? Je něčím výjimečným? Proč byly lidské bytosti obdařeny rozumem? Na některé z otázek dodnes neznáme přesnou odpověď. Různé vědy a náboženství na ně nabízí různé odpovědi, různé úhly pohledů.

Už v minulosti existovala touha vytvořit umělý život případně umělou inteligenci. Namátkou můžeme připomenout legendu o Golemovi, kterého stvořil Rabi Löw z hlíny (hardware) a šému ha-meforaš (software). Alchymisté chtěli stvořit homunkula, vědec Viktor Frankenstein stvořil své monstrum. Ale nezůstalo jen u literatury, Leonardo da Vinci vytvářel robotické rytíře, pojízdné automaty, které se snažily napodobovat člověka nebo jeho chování. I v současné literatuře a filmu najdeme mnoho děl, které se zabývají umělou inteligencí. Za zmínku stojí minimálně Isaac Asimov a jeho robotické zákony.

Než začneme umělou inteligencí, podívejme se na tu přirozenou. Každý živý živočich se projevuje chováním, které bychom mohli nazvat inteligentním. Staří filosofové pracovali s termínem duše, která propůjčovala hmotě život. Slovo duše je jazykově příbuzné se slovem dýchat. Dýchání bylo jedním z nejvýraznějších znaků života. Kdo nedýchal, byl mrtvý, a tedy bez duše.

Místo, kde sídlila duše se v čase mění a souvisí se vzrůstajícím poznáním lidského těla. Původně byla duše v různých životně důležitých tělesných orgánech – např. v játrech, slezině nebo žaludku. Mozek byl považován podle Aristotela za žlázu určenou pouze k ochlazování krve a tento názor byl považován za správný po dobu dvou tisíciletí. Později se duše přesídlila do srdce, protože bijící srdce bylo spojováno s životem. Tato představa se dodnes promítá do mnoha lidových rčení (Má zlaté srdce. Co oči nevidí, to srdce nebolí). Podobně je dodnes láska spojována se srdcem.

Teprve v druhé polovině minulého století díky detailním anatomickým poznatkům je definitivně duše spojována s mozkem. V 18. století bylo pod mikroskopem pozorováno, že se mozek skládá z mnoha propletených buněk, které německý anatom Heinrich von Waldeyer-Hartz později pojmenoval jako neurony. Bylo zjištěno, že neurony komunikují pomocí elektrických impulsů. Ve dvacátých letech německý neurolog Johannes Berger zjistil, že elektrické vlastnosti mozkové činnosti lze měřit. Tzv. Bergerova elektroencefalografie umožnuje odhalit některá nervová onemocnění. A původní pojem duše je nahrazen pojmem inteligence.

V počátečních obdobích snah o poznání podstaty myšlení byl používán přístup z vnějšku. To znamená, že o podstatě věci se soudilo převážné podle vnějších projevů. Teprve mnohem později, a to především díky rozvoji fyziologie a neurologie, bylo možné přiblížit se poznání struktury mozku a jeho funkce zvnitřku. Přístup zvnitřku se prohloubil ve druhé polovině 19. a v první polovině 20. století v souvislosti s rozvojem věd o buňkách jako základních elementech organismu. Teoretické základy této v té době nové vědní disciplíny položili ve svých dílech např. Čech Purkyně nebo Španěl Cajal. V současné době se k výzkumu používá kombinace obou metod.

Neexistuje jednoznačná a plně vyčerpávající definice přirozené inteligence. Můžeme definovat některé její vlastnosti jako soubor specifických předpokladů jednotlivce, umožňujících mu úspěšně se vyrovnat s novými životními podmínkami a řešit situace, v nichž nelze použít návykového chování. Má složitou, individuálně různou strukturu, souvisí jak s intelektem, tak i s dalšími znaky osobnosti. Tuto vlastnost pak lze popsat některými znaky, jako je např. rozumovost, duševní schopnost a vyspělost; chápavost, vzdělanost, moudrost, duševní nadání atd. Inteligence je lidská vlastnost, kterou lze těžko definovat, a ještě hůře měřit. [1] [2] Na měření lidské inteligence se používají IQ testy vytvořené původně pro americkou armádu. Občas bývá hodnota IQ velmi přeceňována. Existují situace, pro jejichž řešení potřebujeme jiné vlastnosti, než jsou měřeny pomocí IQ. Proto se zavádí i jiné systémy jako EQ (emoční kvocient). Tyto systémy mohou fungovat u člověka, ale ve chvíli, kdy je budeme chtít použít pro další organizmy anebo pro umělou inteligenci, tak zpravidla budou selhávat.

Jednou z definic biologické inteligence je popsání vnějšího chování v komplexním prostředí. Aby šlo organizmus prohlásit za inteligentní zpravidla chceme pozorovat některé z těchto schopností.

- Řešení obtížných problémů a dosahování cílů
- Učení se a přizpůsobování se
- Získávání nových znalostí a dovedností
- Generalizace znalostí
- Komunikace
- Spolupráce

Snu vytvořit entitu, která bude mít nebiologickou, umělou inteligencí se začínáme přibližovat počátkem 50. let minulého století s rozvojem počítačů. Ty přestáváme chápat jako počítací stoje, ale ukazuje se, že dokáží řešit i jiné problémy. I přesto, že dosud není přesně určené, co je biologická inteligence, vznikl nový vědní obor umělé inteligence. Podle Minského, průkopníka tohoto nového vědního oboru, je možné umelou inteligenci definovat jako vědu o přetváření strojů a systémů, které budou při řešení určitého problému používat takového postupu, který, kdyby ho užil člověk, bychom považovali za projev jeho inteligence [3].

Pro naplnění projevů inteligence systém musí umět uložit znalosti a ty aplikovat pro vyřešení zadaného problému. V průběhu výzkumu se vyskytly dva základní směry, jak tohoto dosáhnout.

- Objekty reálného světa jsou reprezentovány symboly. Systém při řešení problémů symboly zpracovává pomocí pravidel, logiky atd. Jde o syntetický přístup, kdy se tvůrci domnívají, že existence symbolu je nutnou a postačující podmínkou pro existenci inteligentní činnosti. Výsledkem tohoto přístupu bylo vytváření tzv. expertních systémů (ES)
- Druhý analytický přístup se snaží věrně namodelovat funkci lidského mozku a vlastně tak zkopírovat biologický vzor. Dneska tyto modely nazýváme umělé neuronové sítě (ANN).

Předpokládejme, že jsme vytvořili systém s umělou inteligencí. V tu chvíli si budeme pokládat různé otázky: Je opravdu inteligentní? Například tím, že splní nějaký test (Tůringův)? Je test dostatečně průkazný? Dokážeme změřit úroveň umělé inteligence?

U umělé inteligence zpravidla sledujeme následující kritéria určující míru inteligence.

- Schopnost nalézt optimální varianty na základě zadaných kritérií
- Schopnost učit se z předchozích zkušeností
- Schopnost nalézt znaky podobnosti
- Schopnost adaptovat se při změnách okolních podmínek
- Schopnost řídit paralelní procesy
- Schopnost pokračovat v činnosti při chybějících nebo neúplných vstupních datech
- Schopnost abstrakce - zjednodušování problému (skryjí se nepodstatné informace)
- Schopnost generalizace - konstrukce obecného závěru (opak abstrakce)
- Schopnost predikce (předvídat)

Umělou inteligenci můžeme dělit podle dosažené úrovně inteligence následovně:

- **Úzká, specifická, odvětvová, slabá umělá inteligence** jsou pojmy pro systémy, které jsou určeny pro automatické řešení konkrétních přesně definovaných problémů – zaparkuj, hraj šachy, urči objekty na obrázku, … V dnešní době jsou tyto systémy zpravidla lepší než člověk, ať už se jedná o přesnost, rychlost nebo škálovatelnost. Pokud je systému předložen jiný nebo i podobný problém, na který nebyl vytrénovaný, tak selže. Takováto umělá inteligence nerozumí souvislostem nedokáže generalizovat. V dnešní době umělá inteligence dosahuje této úrovně.

- **Obecná, silná umělá inteligence** již myslí a jedná zcela samostatně. Sama si vybírá cíle a určuje jejich prioritu. Inteligence se učí učit se (metalearning). Sama se opravuje, hledá své chyby a opravuje je. Systém je skutečně inteligentní a chápe souvislosti.
- **SuperAI, AGI, Singularity** je obecná inteligence, která ve všech předčí člověka. V dnešní době se vyskytuje ve sci-fi a to buď v roli zachránce nebo ničitele lidstva. Ač dneska ještě neexistuje je vhodné o ní diskutovat a zkoumat vliv na lidstvo. Samotnou otázkou je, zda lidstvo vůbec zjistí existenci AGI. Pokud bude inteligentnější, může úspěšně svoji existenci tajit před lidmi, aby se ochránila nebo dosáhla svých cílů.

2 Historie umělé inteligence

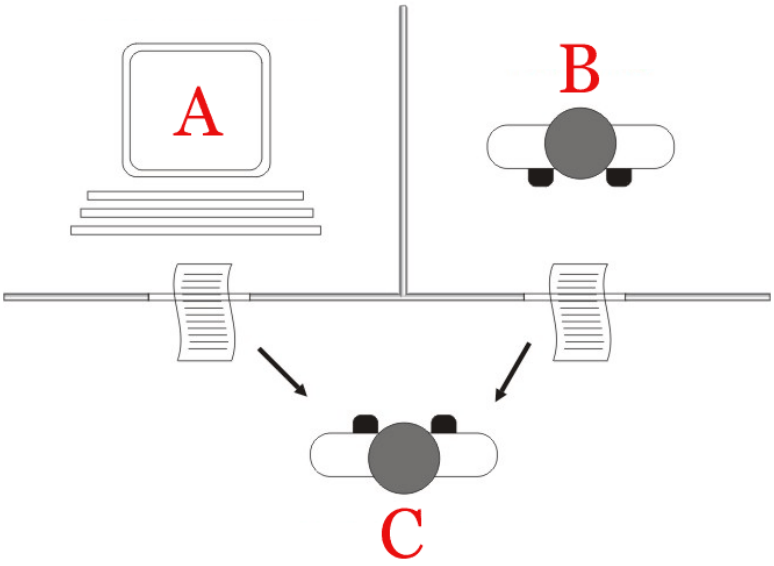
Začátek historie seriózního výzkumu umělé inteligence můžeme spojovat s výzkumem prvních elektronkových počítačů, kdy se vědci začínají zabývat otázkou, k čemu všemu lze počítací stroje použít a zda mohou mít určitou inteligenci.

Průkopníky v oblasti umělých neuronových sítí byli mj. Američané McCulloch a jeho student Pitts, kteří vytvořili v roce 1943 matematický model neuronu [4]. Číselné hodnoty parametru v tomto modelu byly převážné bipolární, tzn. z množiny –1; 0; 1. Přestože byl McCullochuv a Pittsův model od dob svého vzniku podroben dalšímu vývoji, tvoří základ naprosté většiny umělých neuronových sítí dodnes.

V roce 1949 navrhl Hebb učící pravidlo, které bylo založeno na modifikaci synaptických drah mezi neurony během učícího procesu [5]. V roce 1958 Rosenblatt vynalezl tzv. perceptron, který je zobecněním McCullochova a Pittsova modelu neuronu pro reálný číselný obor parametru [6]. Později sestrojil síť perceptronů a pro tento model neuronu navrhl také učící algoritmus [7].

Od počátku vzniku oblasti umělé inteligence jsou spojeny základní otázky, co je umělá inteligence a jak ji dokážeme rozpoznat. Odpověď na tyto otázky se pokusil najít Alan Turning. Alan Turning byl britský matematik, logik a kryptolog. V jeho asi nejvýznamnějším díle *On Computable Numbers, with an Application to the Entscheidungsproblem,* zavádí pojem Turingova stroje, teoretického modelu obecného výpočetního stroje. Během druhé světové války výrazně přispěl k rozluštění německé šifry Enigma. Pro nás je důležitý Turingův test, který uveřejnil v roce 1950. V jedné místnosti je testovaná umělá inteligence. V druhé místnosti je kontrolní vzorek – člověk s biologickou inteligencí. Rozhodce je umístěn do místnosti, která je od obou oddělena systémem předávání zpráv, který zamezuje odhalení člověka například pomocí rukopisu, hlasu atd. Rozhodce klade obou testovaným entitám otázky, a ty na něj odpovídají. Je zcela na rozhodci, jakou formou povede dva dialogy. Jeho úkol je určit, v které místnosti je člověk a v které umělá inteligence. Tento test určuje inteligence na základě vnějších projevů. Nezkoumá vnitřní kognitivní procesy entit. Později

si ukážeme, že tento test nemusí být zcela vypovídající a na základě něho dokáže přisoudit inteligenci i entitám, které ji mít nemusí.



Obrázek 1 Turingův test

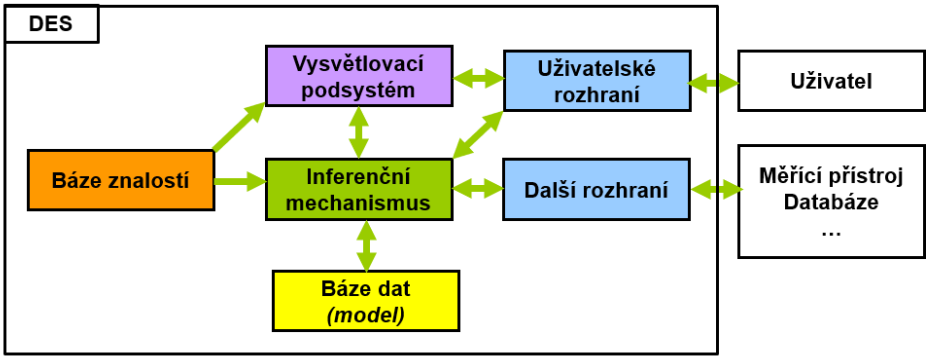
Vedle výzkumu neuronových sítí se v této době vytvářely programy, které se snažily hrát logické hry jako je dáma nebo šachy. Program na hraní dámy měl schopnost učit se a generalizovat. V praxi autor nechal hrát proti sobě dvě verze programu, který byl pak schopen porazit autora.

Jiným typem programů, které se snažily napodobit inteligenci, byly ty, které se snažily řešit různé logické a formální úlohy. Příkladem může být *The Logic Theorist*, který matematicky dokázal 38 z 52 tvrzení v knihy Principia Mathematica. Program *The General Program Solver* vznikl pro řešení libovolných symbolicky formulovaných problémů. V této době byl vytvořen specializovaný programovací jazyk LISP (LISt Processing), pro tvorbu systémů založených na symbolické logice. Zároveň v roce 1956 se koná první konference o umělé inteligenci v Dartmouth. Obecně v této době se zkouší s úspěchy různé přístupy, kdy se napodobují vyšší funkce mozku a problémy se řeší pomocí formalizace. Je zde snaha o vytvoření systému, který by porozuměl mluvenému jazyku. Panoval optimismus a někteří autoři předvídali brzký vznik skutečné umělé inteligence.

V 60. létech se mění přístup k umělé inteligenci. Začínají se napodobovat nižší schopnosti mozku, modelují se neurony a učení. V umělé inteligenci se prosazují fuzzy logika, expertní systémy a umělé neuronové sítě. Ale zároveň se nadále dosahují úspěchy při použití symbolické umělé inteligence. Později se proto tomuto období bude říkat GOFAI (Good Old FinE AI).

Programy a expertní systémy jsou postaveny na datech a pravidlech. Profesor Feigenbaum ze Stanfordské univerzity definoval **expertní systémy** následovně: je to inteligentní počítačový program, který používá znalosti a inferenční mechanismy (metody zpracování těchto znalostí) k řešení problému,

které jinak vyžadují rozsáhlou expertízu provedenou odborníky v dané oblasti. Základem expertních systémů jsou tedy znalosti a metody jejich zpracování. Jedno bez druhého nemá smysl. Znalostmi se myslí symbolicky popsané objekty reálného světa a vztahy mezi nimi. Tyto znalosti se ukládají v datové struktuře nazývané báze znalostí. S bázemi znalostí umějí pracovat již zmíněné inferenční mechanismy, někdy označované jako báze pravidel. Poslední částí expertního systému je báze dat. Vstupní data jsou získána od uživatele nebo například automaticky vyčítáním různých čidel apod.



Obrázek 2 Schéma diagnostického expertního systému

Báze znalostí obsahuje:

- Všeobecně známé poznatky o dané problematice, které lze formálně dokonale popsat a zdůvodnit
- Heuristiky. Heuristika je postup, soustava pravidel, sloužících k nalezení cíle. Často se jedná o interní znalosti experta, které používá při svém rozhodovacím procesu. Nejčastěji se heuristické vztahy vyjadřují pomocí tzv. produkčních pravidel. Ta popisují kauzální vztahy mezi objekty. Produkční pravidla mají standardní strukturu, např.

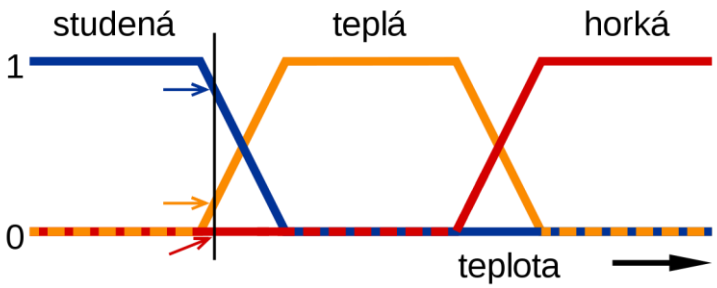
  JESTLIŽE [podmínka] TAK [důsledek],

  nebo

  JESTLIŽE [podmínka] TAK [důsledek 1] JINAK [důsledek 2]

Expertní systémy s těmito pravidly jsou označovány jako první generace. Problémem této generace expertních systémů bylo, že se jedná o pravidla deterministická. V praxi se spíše budeme setkávat se stochastickými pravidly, které pracují s mírou pravděpodobnosti. Nelze tvrdit, že pokud má pacient teplotu v rozmezí 37 °C až 38° a má bolesti krku má angínu. Pravidla mají i v reálném světě určitou pravděpodobnost a existují určité výjimky. To zjišťovali i autoři prvních expertních systémů, kteří v některých případech měli s experty problém definovat pravidla. Experti někdy nebyli schopni vyjádřit přesná pravidla, určit podle čeho se rozhodují a pracovali s pojmy jako pocit, intuice, šestý smysl apod.

Další verze expertních systémů se nejistotu a vágní formulace v pravidlech snaží řešit pomocí je fuzzy logiky. „Fuzzy“ znamená v angličtině rozmazaný, mlhavý. V tomto smyslu se také v češtině používá pojmu mlhavá, vágní nebo neurčitá logika. Lze ji považovat za zobecnění klasické, dvouhodnotové (booleovské) logiky na vícehodnotovou. Pravdivost výroku muže nabývat hodnot v intervalu <0;1>. S použitím takového matematického aparátu se lze v podmínkách a důsledcích produkčních pravidel vyjadřovat s určitou pravděpodobností a vyjádřit tak nepřesné pojmy lidského myšlení matematickým zápisem.



Obrázek 3 Fuzzy logika přiřazení teploty pojmům

Tvorba expertních systémů není jednoduchá ani levná, protože je třeba vývojový prostředek (prázdný expertní systém), experta a znalostního inženýra. Naplnění prázdného expertního systému znalostmi experta je časově a finančně náročná činnost, která nemusí mít jistý výsledek.

Vývojový prostředek je zpravidla jednoduché zajistit, na trhu jich je mnoho. Problémem je najít experta, který má dostatek času, a hlavně ochotu se o znalosti podělit. Znalostní inženýr je vlastně mezičlánek, který dokáže znalosti experta zpravidla neznalého informatiky převést do znalostní báze kompatibilní s vývojovým prostředkem.

Schematicky si tento proces můžeme znázornit následovně. Na vstupu jsou data a pravidla od experta. Ty je třeba vložit do systému ve formě nějakého programu. Program pak dokáže zpracovat vstupní data a poskytovat výsledky.

I přes tento komplikovaný způsob tvorby inteligentních systémů vznikly i úspěšné expertní systémy. Můžeme uvést několik příkladů

- DENDRAL – expertní systém pro identifikaci struktury neznámých organických molekul
- MYCIN – expertní systém určený k diagnostice bakteriálních infekcí

- PROSPECTOR – vyhledávání rudných ložisek
- SID – navrhuje počítače VAX

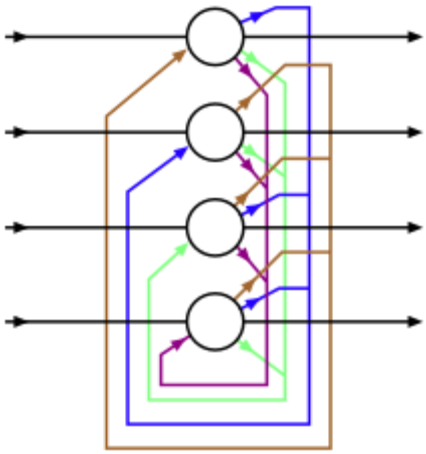
Zajímavý program ELIZA byl vytvořený profesorem Josephem Weizenbaumem. Jedná se jednoduchý program, který konverzoval v přirozeném jazyku (angličtina) a snaží se vystupovat jako psychoterapeut. Program klade uživateli otázky založené na klíčových slovech zachycených z předchozí konverzace.

- Mám problémy.
- Můžeš to rozvést?
- Trápí mě rodiče.
- Co si myslíš o rodičích?
- …

Mnoho běžných lidí si myslelo, že program prošel turingovým testem, protože s uživatelem dokázal vést smysluplný rozhovor. Nicméně jen vytvářel otázky podle jednoduchých pravidel a dat zadaných samotným uživatelem.

Koncem šedesátých let minulého století dochází k útlumu ve výzkumu umělých neuronových sítí. Nešťastná byla též kampaň vedená Minským a Papertem, kteří využili svého vlivu, aby diskreditovali výzkum umělých neuronových sítí ve prospěch převodu finančních zdrojů z této oblasti do jiné oblasti výzkumu umělé inteligence. Pro svou argumentaci využili známého faktu, že jeden perceptron nemůže realizovat jednoduchou logickou funkci, tzv. vylučovací disjunkci (XOR) [8]. Toto je pravda s použitím jednovrstevného modelu neuronové sítě. Toto tvrzení je sice správné, ale platí pro jeden perceptron. Jednoduchá síť, složená ze tři perceptronů, tuto logickou funkci už simulovat umí. Bohužel až do roku 1986 nebyl znám algoritmus, jak tuto neuronovou síť vytrénovat. Až v po tomto objevu byla éra syntetického přístupu k řešení problému UI nahrazena érou analytického přístupu.

K opětovnému nástupu neuronových sítí dochází v letech 1982 – 1984, kdy Američan Hopfield navrhnul nový model neuronové sítě tzv. Hopfieldovu síť [9]. Jedná se o plně propojenou symetrickou síť, tedy propojení „každý s každým, kromě sebe sama“.



Obrázek 4 - Hopfieldova síť

Další velmi důležitý objev byl zaznamenán v roce 1986. V tomto roce se nezávisle na sobě podařilo dvěma vědcům, Rumelhartovi a LeCunovi, odvodit a popsat nový algoritmus učení vrstevnatých neuronových sítí [10]. Jednalo se o algoritmus zpětného šíření chyby, tzv. Backpropagation Algorithm, kterým bylo možno řešit problém, jenž se Minskému a Papertovi v 60. letech jevil jako nepřekonatelná překážka pro využití a další rozvoj umělých neuronových sítí. Tento algoritmus je dosud nejpoužívanější metodou učení neuronových sítí.

Paralelně k vrstevnatým sítím přichází Fin Kohonen s dalším velmi důležitým objevem, s tzv. samoorganizující se neuronovou sítí [11]. Samoorganizující se neuronové sítě jsou takové, které ke svému učení nepotřebují učitele. Příkladem z této kategorie neuronových sítí může být např. Kohonenova mapa.

Díky těmto objevům začínají vznikat nové firmy a do umělé inteligence se začíná více investovat. V devadesátých letech minulého století ovšem dochází k dalšímu útlumu. Umělé neuronové sítě se dostávají na okraj zájmu. Při tvorbě inteligentních systémů se spíše používají Bayesovské sítě, teorie rozhodování nebo teorie pravděpodobnosti. Umělá inteligence je nepopulární slovo a raději se používá pojmu inteligentní agent, tedy systém, který se snaží napomáhat uživateli.

Za zmínku v této dekádě stojí úspěch IBM, která se svým systémem Deep Blue založeným na serverech s procesory Power dokázala porazit velmistra Garry Kasparova. O férovosti tohoto souboje se vedou dodnes spory. Na straně IBM stál tým programátorů a 4 šachových velmistrů. Mezi jednotlivými partiemi byl delší čas, kdy se program přizpůsoboval na míru proti Kasparovi. Inteligence spočívala ve vhodně zvolených expertních pravidlech a hrubé výpočetní síle. Nicméně jeden z milníků umělé inteligence byl dosažen, počítač porazil člověka v šachách.

Možná zapomenutým úspěchem byla sonda NASE Deep Space 1. Jednalo se o sondu vypuštěnou v roce 1998, která obsahovala první řídící systém s prvky umělé inteligence, který dokázal ovládat kosmický objekt bez zásahu člověka.

Od roku 2000 začíná růst popularita umělé inteligence díky dosažení mnoha různých milníků. Důvodem tohoto úspěchu je dostupnost rozsáhlých datových sad pro strojové učení a výrazné zvýšení výkonu a zlevnění výpočetní techniky. Postupy, které před několika lety ztroskotaly na nedostatečném výkonu počítačů, a tedy omezené velikosti neuronových sítí, začínají fungovat. V jednoduchosti často stačilo zvýšit počet neuronů a počet vrstev a modely začaly vracet správné výsledky.

Paradigma vývoje systémů s umělou inteligencí se změnila. Místo vytváření programů, které na základě dat a pravidel vrací výsledky, vstupem jsou data a výsledky, které vstupují do algoritmů strojového učení. Tyto algoritmy vytváří modely, které bychom při troše vůle mohli přirovnat k algoritmům, které dříve psali programátoři. Přesouvá se tak těžiště lidské činnosti z programování na vytváření datových sad a výběru algoritmů strojového učení.

Tento růst umělé inteligence trvá dosud a lze očekávat, že ještě nějakou dobu bude trvat. Systémy strojového učení začínají předhánět člověka v čím dál tím komplikovanějších výzvách. Toto můžeme demonstrovat na některých dosažených milnících.

- 2004 Agentura DARPA vypsala v roce 2004 Grand Challenge. V této výzvě měla autonomní vozidla přejet Mohavskou poušť. Na trasa byly vytvořeny překážky, které například znemožnily snímání polohy pomocí GPS. V tomto roce žádné autonomní vozidlo výzvu nedokončilo.
- 2005 Grand Challenge dokončila většina vozidel
- 2006 Učení modelu je časově a výpočtově velmi náročné. Dobu učení lze velmi zkrátit masivní paralelizací. V dané době procesory počítačů obsahovaly pouze několik jader. Nicméně grafické karty obsahovaly mnohonásobně více výpočetních jader, které naštěstí rychle zvládaly matematické operace pro trénování modelů – tedy práci s vícerozměrnými maticemi. Společnost Nvidia v roce 2006 uvolnila architekturu CUDA, které umožnila provádět paralelní výpočty na GPU.
- 2006 Společnost Google převedla svůj Google translator na statistický model. Kvalita překladu se tak mnohonásobně zlepšila.

- 2007 DARPA vytvořila výzvu Urban Challenge, ve které se autonomní vozidla pohybují v běžném provozu.
- 2008 Když počítač porazil člověka v šachách, mnoho kritiků tvrdilo, že se jedná o úlohu šitou přímo pro něj. Hra s jasně danými pravidly a plnou informací, která je sice výpočetně náročná, ale deterministická. V roce 2008 počítač poprvé porazil člověka v pokeru, tedy ve hře, kdy hráč nemá plné informace a vítězná strategie nemusí být založena pouze na výpočtech.
- 2010 Ne každý má k dispozici dostatečně výkonný hardware, aby mohl trénovat a provozovat umělou inteligenci. První AI služby se začínají objevovat v cloudovém prostředí jako je Azure. Pro vývojáře je tak daleko jednodušší své programy vybavit umělou inteligencí.
- 2011 Mnoho znalostí je uloženo v nestrukturované podobě – text, obrázek, vide, zvuk apod. Dosud počítače velmi dobře zpracovávaly strukturovaná data. V roce 2011 IBM uvedla svůj systém IBM Watson, který dokázal porazit lidské soupeře ve hře Jeopardy (Riskuj). Watson dostával otázky v přirozeném jazyce, v textové podobě. Systém musel pochopit otázku a odpověď velmi rychle dohledat. Zdrojovými daty v tomto případě byly dokumenty a webové stránky. Umělá inteligence ukázala, že dokáže pracovat s texty, částečně jim porozumět a propojit různé informace v různých dokumentech. Umělá inteligence dokázala pracovat s textem podobným způsobem, jak to dělají právníci, doktoři atd.
- 2015 Zpracováním obrazových informací se zabývá Computer vision. Tato oblast má široké využití v mnoha lidských činnostech, protože zrak je primárním lidských smyslem. Cílem umělé inteligence je umět poznat, co je na obrázku, případně kde se daný objekt přímo nachází. Proto byla vypsána výzva, kdy se systémy měly naučit rozpoznávat obrázky z velké databáze ImageNet. V roce 2015 byla neuronová síť Alexnet v rozpoznávání obrázků úspěšnější než člověk.
- 2016 Deskové hry šachy a Go se liší ve velikosti stavového prostoru. Pokud v roce 1997 Deep Blue dokázal efektivně vyhledávat tahy ve stavovém prostoru, tak pro hru go s jejím mnohonásobně větším stavovým prostorem se předpokládalo, že stejný výpočetní přístup není možný a počítač nebude schopen nikdy člověka porazit. V roce 2016 neuronová síť AlphaGo porazila šampiona v Go. Neuronová síť se učila na ohromném množství partií odehraných lidmi. Pak následovala fáze, kdy partie hrála sama proti sobě. Neuronová síť se nesnažila spočítat všechny tahy dopředu, spíše v partiích rozpoznávala vzory chování. Přiblížila se velmistrům, kteří se hru učili mnoho let a ze zkušenosti dokázali určit správný tah, někdy aniž by si byli vědomi přesného pravidla.

- 2016 Google translator přechází na Neural Machine Translator a opět dochází k výraznému zlepšení překladu.
- 2020 Umělá inteligence proniká i do oblasti vojenství, ať se nám to líbí nebo ne. Výhoda zpracování ohromného množství různorodých dat v krátkém okamžiku přináší velké výhody. Umělá inteligence se tedy nevyhnula i řízení bojového letadla ve vzdušném souboji. Proti autonomní vozidlům se letadlo pohybuje v 3D prostoru a má k dispozici data z mnoha senzorů. Rychlost reakce a předvídání protivníka je tak klíčová. 2020 AlphaDogFighter porazila lidského pilota.
- 2020 Studium chemických látek, léků a nemocí vždy posouvalo medicínu kupředu. Interakce jednotlivých molekul, jejich tvarů a struktury často předurčuje, zda bude daná látka účinným lékem nebo ne. Neuronová síť AlphaFold se naučila předvídat strukturu proteinů a urychluje tak výzkum nových léčiv.
- 2021 Dosud se milníky týkaly oblastí, kdy bylo třeba analyzovat velké množství dat a na základě nich vytvářet modely, predikce apod. To už umělá inteligence zvládla. Ale zastánci biologické inteligence tvrdili, že máme navrch, protože jsme kreativní a dokážeme vytvářet různá třeba umělecká díla – obrazy, povídky, hudbu atd. To už s generativními neuronovými sítěmi dokáže dělat i umělá inteligence. Například DALL-E nebo Midjourney dokáže na základě textového popisu generovat obrázky. Umělá inteligence se používá pro vylepšení obrázků, odstranění pozadí, doplnění umělých detailů nebo barev. Výše zmíněné modely pracují s obrazovou informací. Existují i systémy, které zpracovávají zvuky. Například jsou schopny komponovat nové skladby, které se podobají významným skladatelům. Nebo dokáží napodobit hlas konkrétního člověka atd.
- 2022 Hitem posledních let jsou velké jazykové modely (Large Language Model LLM). Příkladem může být chatGPT, Bard, Mistral atd. Jedná se o model, který se učí na ohromném množství textových dat. Na základě předchozí konverzace (kontextu) predikuje následující slova. Tyto modely dokáží odpovídat na otázky položené v přirozeném lidském jazyku a simulovat konverzaci.
- 2023 Dosavadní LLM dokázaly zpracovávat pouze informace, které měly k dispozici během svého učení. Nedokázaly reagovat na novinky, které se od té doby staly. Nově LLM integrují výsledky z vyhledávačů jako je google nebo bing.

3 Typy umělé inteligence

3.1 Podle rozsahu problémů

Umělou inteligenci můžeme dělit podle různých pohledů. Prvním, který si uvedeme, je rozsah problémů, které je schopna řešit, jak již bylo nastíněno výše.

V dnešní době můžeme všechny systémy s umělou inteligencí řadit do skupiny, která bývá pojmenována následujícími pojmy: **úzká, specifická, odvětvová, slabá** umělá inteligence.

Jedná se o systémy s umělou inteligencí, které jsou schopny automaticky řešit konkrétní, přesně definované činnosti, úkol a problémy. V dnešní době úzká umělá inteligence velmi často dokáže problém řešit lépe než člověk – přesněji, rychleji. Zároveň se jednoduše škáluje.

Pokud tento systém podstavíme před problém, na který nebyl natrénován, tak katastrofálně selže, a to v úkolech, které člověku připadají velmi podobné. Příkladem takových úloh je hra šachy a hra dáma, rozpoznání objektu A na fotce a rozpoznání objektu B na fotce, překlad do různých jazyků.

Takovéto systémy umělé inteligence nerozumí souvislostem, často nemají reprezentaci reálného světa, nedokáží se učit, generalizovat, abstrahovat. Ale jejich chování v oblasti, na kterou byly vytrénovány, se jeví inteligentní, takže máme pocit, že je v nich mnohem více.

Pokročilejším typem je umělá inteligence označovaná jako **obecná, silná**. Systémy s touto úrovní inteligence již myslí zcela samostatně, jsou schopny se definovat své cíle a stanovovat jim priority. Silná obecná inteligence se učí učit – metalearning. Z poznatků o dříve vyřešených problémech je schopna se sama naučit řešit problémy i v jiných oblastech. Tento systém umělé inteligence hledá své chyby a opravuje je, chápe různé souvislosti a reálný svět. O takovéto umělé inteligenci můžeme tvrdit, že je skutečně inteligentní. Doposud není znám žádný umělý systém, který by této úrovně dosahoval.

Nejdokonalejší umělá inteligence se nazývá **SuperAI, Singularity**. Jedná se o obecnou inteligenci, která ve všech měřítkách předčí schopnosti člověka. Někteří lidé se jí bojí, jiní do ní vkládají naději na přežití lidstva. Jaké různé hrozby a přínosy si můžeme přečíst v různých sci-fi.

3.2 Podle filosofie

Umělou inteligenci můžeme dělit i podle filosofie jejího vytváření.

Symbolická AI (GOFAI Good Old FinE AI) při tvorbě používá symbolickou logiku. Vychází z toho, že inteligentní chování lze popsat pomocí pravidel. V západní filosofii se jedná o proud, který tvrdí, že abstraktní rozum je „nejvyšší“ schopností, že je to, co odděluje člověka od zvířat, a že je to nejpodstatnější část naší inteligence. Tento předpoklad je přítomen u Platóna a Aristotela, u

Shakespeara, Hobbese, Huma a Locka. Byl ústředním bodem osvícenství, logických pozitivistů třicátých let a výpočetních kognitivistů šedesátých let.

Symbolická umělá inteligence v 60. letech dokázala úspěšně simulovat proces uvažování na vysoké úrovni, včetně logické dedukce, algebry, geometrie, prostorového uvažování a analýzy prostředků a cílů. Vše v přesných anglických větách. Mnoho pozorovatelů, včetně filozofů, psychologů a samotných výzkumníků AI, bylo přesvědčeno, že zachytili základní rysy inteligence. Nebyla to jen arogance nebo spekulace – to bylo způsobeno racionalismem. Pokud by to nebyla pravda, pak to zpochybňuje velkou část celé západní filozofické tradice.

Můžeme uvést několik příkladů, které spadají do této kategorie. IBM Deep Blue, který porazil Kasparova, expertní systémy The Logic Theorist, The General Program Solver, DENDRAL atd. Vznikají programovací jazyky, které mají napomoci v zachycení těchto pravidel jako je LISP, Prolog apod.

Kontinentální filozofie, která zahrnovala Nietzscheho, Husserla, Heideggera a další, odmítala racionalismus a tvrdila, že naše uvažování na vysoké úrovni je omezené, náchylné k chybám a že většina našich schopností pochází z naší intuice, naší kultury a našeho instinktivního cítění situace. Filozofové, kteří byli obeznámeni s touto tradicí, byli první, kdo kritizoval GOFAI a tvrzení, že to stačí pro inteligenci, jako Hubert Dreyfus a Haugeland.

Proto se rozvinuly i další směry, které se nesnaží najít abstraktní rozum, ale spíše vycházejí z nižších funkcí mozku, jeho stavby, případně chování jiných živočichů. Průnik těchto směrů můžeme najít ve statistice případně evoluci. Budeme je označovat jako **statistickou** umělou inteligenci. Můžeme sem zahrnout různé statistické algoritmy, genetické algoritmy i v poslední době tak populární neuronové sítě. Základním rysem těchto algoritmů je, že se nesnažíme najít přesná pravidla, ale necháme systém, aby si je našel sám. Mluvíme pak o **strojovém učení**. Tyto systémy si můžeme představit jako černé skříňky, kterým poskytneme data, na kterých se naučí problém řešit. Pokud jim předložíme nová data, tak na ně dokáží odpověď. V některých případech, ani nedokážeme vysvětlit, proč.

Algoritmy, které používají statistický přístup, můžeme dále dělit do následujících podkategorií.

Učení s učitelem (supervised learning) předpokládá, že člověk vytvoří trénovací dataset, který obsahuje otázky, ale i správné odpovědi. Systém si postupně upravuje své parametry, aby dokázal na učební otázky odpovídat co nejpřesněji. Úskalím těchto algoritmů je dobře připravený trénovací dataset. Pokud nebude obsahovat některé příklady z reálného světa, tak na ně nebude schopen odpovídat.

Učení bez učitele (unsupervised learning) jsou algoritmy, které nevyžadují přípravu trénovacích dat. Algoritmu jsou předloženy data a sám se učí rozpoznávat vzorce, podobnosti, tvary, struktury atd.

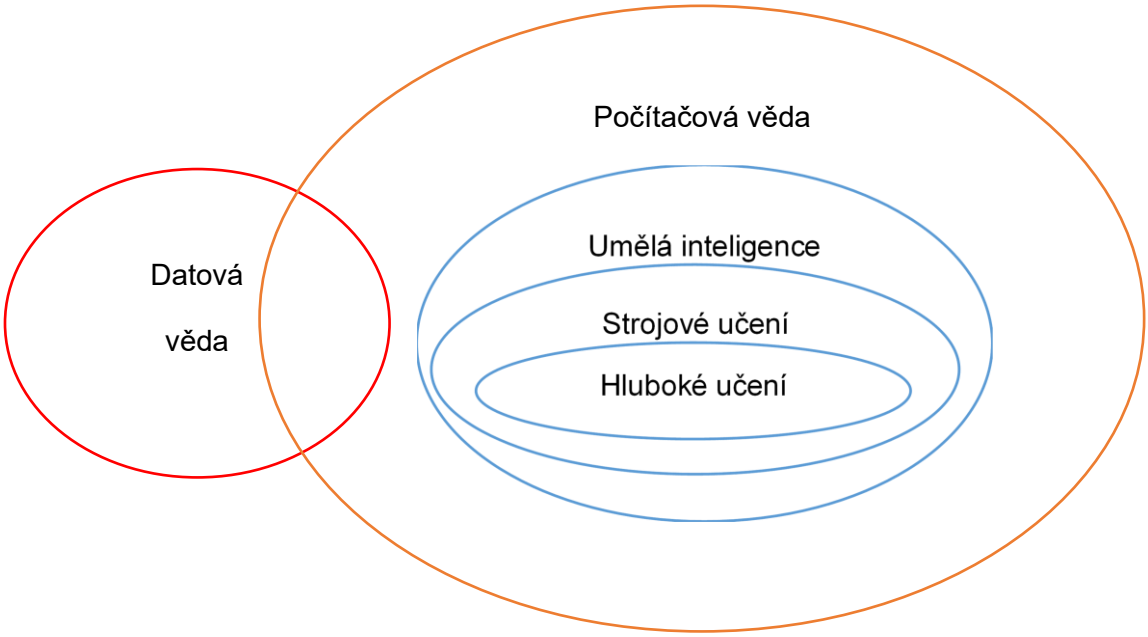
Zvláštním typem je **self supervised,** kdy si umělá inteligence sama vytváří trénovací data.

Posledním zmíněným typem je posilované učení (**reinforcement learning**), který se velmi často používá například v robotice. Systém se během trénování pohybuje a dostává pomocí zpětné vazby kladné nebo záporné hodnocení svých akcí.

4 Umělá inteligence

Ačkoliv se umělá inteligence nejčastěji realizuje na počítačích, tak vychází i z jiných oborů. Vzájemné vztahy lze znázornit v následujícím diagramu.

- Pravděpodobnost a statistika
- Metody rozhodování
- Teorie fuzzy množin
- Umělé neuronové sítě
- Umělé formální jazyky
- Teorie rozhodování a teorie her
- Teorie grafů



4.1 Úlohy

Častými úlohami v umělé inteligenci jsou regrese, klasifikace, detekce, segmentace, optimalizace, shluková analýza.

Regrese se zabývá předpovědí výsledné hodnoty na základě vstupních hodnot. Na základě trénovacích dat je vytvořen předpovědní model.

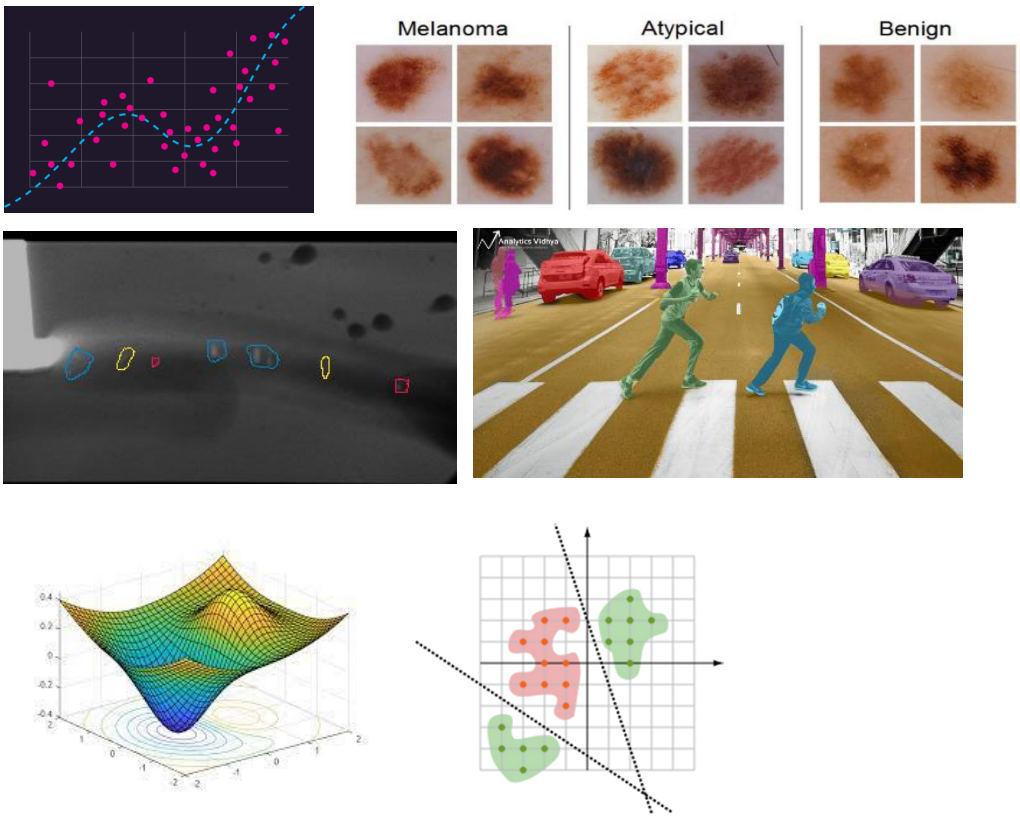
Klasifikační úloha vstupní data přiřazuje do jedné z předem definovaných skupin. Na obrázku je příklad zařazení nádoru do kategorie zhoubný, nezhoubný, atypický. Jiným příkladem je zařazení zákazníka do jedné z definovaných skupin a nabídnutí mu odpovídajícího produktu.

V úloze detekce se určují v datech objekty.

V úloze segmentace se v obrazových datech vyznačují objekty.

Široká oblast optimalizace se snaží nalézt řešení, které splňuje maximální nebo minimální hodnoty hodnotící funkce.

Poslední zmíněnou úlohou je shluková analýza, která rozděluje měření různých objektů do tříd.



Obrázek 5 regrese, klasifikace, detekce, segmentace, optimalizace, shluková analýza

4.2 Algoritmy umělé inteligence

V průběhu času byla objevena celá řada různých algoritmů, které jsou vhodné pro řešení různých problémů. Velmi často se k dříve vymyšlenému algoritmu přidávají jeho varianty, které určité úlohy optimalizují. Uvedený seznam není určitě úplný, spíše má ukázat šíři umělé inteligence.

Reálný problém lze řešit různými algoritmy, které v daný okamžik mají různé výhody a nevýhody a přináší různě přesné modely. Výběr správného přístupu je do určité míry uměním datového analytika. Během tohoto předmětu se s některými algoritmy seznámíme.

- Umělá inteligence
  - Metody prohledávání stavového prostoru
    - Neinformované metody
      - Prohledávání do šířky
      - Prohledávání do šířky dle ceny
      - Prohledávání do hloubky (DFS)
        - Prohledávání do hloubky s omezením (DLS)
        - Iterativní prohledávání do hloubky
      - Obousměrné prohledávání
    - Informované metody
      - Hladové heuristické hledání
      - Paprskové prohledání
      - Algoritmus A*
      - Algoritmy pro hraní her (2 hráči)
      - Algoritmy pro hraní her (více hráčů)
    - Algoritmy pro hraní her (2 hráči)
      - Minimax
        - Minimax s alfa beta prořezáváním
        - Negamax
        - Nega Scout
    - Algoritmy pro hraní her (více hráčů)
      - Max^n
      - Hledání nejlepší odpovědi
      - Paranoik
  - Meta-heuristické optimalizační metody
    - Meta heuristiky založené na jednom řešení
      - Simulované žíhání

- Mikrokanonické žíhání
- Prahová metoda přijetí
- Metoda hluku
- Tabu prohledávání
- Algoritmus GRASP
- Proměnné vyhledávání v sousedství
- Vedené místní vyhledávání
- Opakované místní vyhledávání

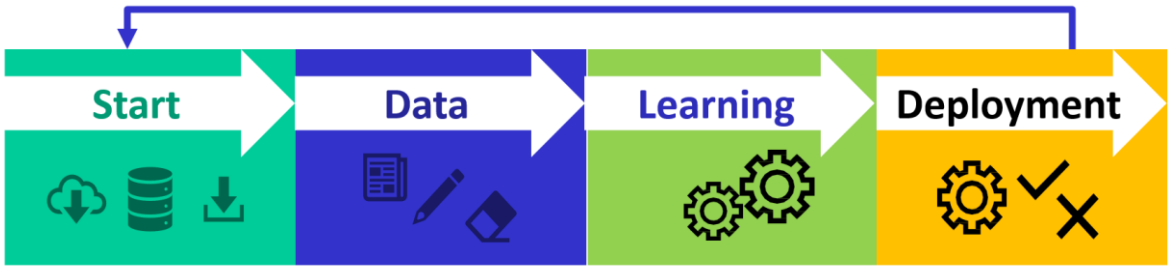
- Meta heuristiky založené na populaci
  - Evoluční výpočty
    - Genetické algoritmy
    - Evoluční strategie
    - Evoluční programování
    - Genetické programování
  - Ostatní evoluční algoritmy
    - Diferenciální evoluce
    - Koevoluční algoritmy
    - Kulturní algoritmy
    - Rozptýlené hledání
  - Inteligence hejn
    - Optimalizace mravenčí kolonie
    - Optimalizace včelím rojem
    - Umělé imunitní systémy
    - Optimalizace založená na biogeografii

- Strojové učení
  - Strojové učení s učitelem
    - Regresní metody
    - Metoda nejmenších čtverců
    - Logistická regrese
    - Kroková regrese
  - Pravděpodobnostní metody
    - Naivní Bayesův klasifikátor
    - Gaussovský naivní Bayesův klasifikátor
    - Bayesovské sítě
    - Support Vector Machine

- Rozhodovací stromy
    - Algoritmus ID3
    - Algoritmus C5.0
- Multi-modelové metody
    - Náhodný les
    - Boostling
    - AdaBoost
- Metody založené na pravidlech
    - Cubist
    - Algoritmus QA
- Umělé neuronové sítě
    - Perceptron
    - Vícevrstvý perceptron
    - Konvoluční neuronové sítě
    - Rekurentní neuronové sítě
- Strojové učení bez učitele
    - Rozdělovací metody
        - k-means
        - Fuzzy c-means
        - k-metoids
    - Hierarchické metody
        - CURE
        - Chameleon
        - BIRCH
    - Umělé neuronové sítě
        - Samoorganizující se mapa
        - Autoencoder
        - Konvoluční rekurzivní síť
    - Mřížkové metody
        - STING
        - WaweCluster
- Zpětnovazební učení

5 Projekt umělé inteligence

Projekt na umělou inteligenci prochází podobnými fázemi jako běžný softwarový projekt. Některé fáze jsou pojmenovány odlišně, ale například za fázi učení si můžeme doplnit fázi implementace.



Obrázek 6 - fáze AI projektu

Úvodní fáze projektu je totožná s jinými typy projektů. Je třeba definovat projekt, výstup, požadavky, dopady na podnikové procesy atd. Důležité je zajištění financování projektu a zajistit přístup k relevantním a kvalitním datům z různých zdrojů, které umožní řešit zadaný problém.

Důležitou fází je **příprava dat**. Zpravidla v organizaci existuje více různých zdrojů dat v různých formátech, strukturách, databázích. Tato fáze se velmi podobá procesu ETL (extract, transform, load), kterou známe z datových skladů.

Během extrakce zajišťujeme přístup ke zdrojovým datům a jejich přenos do pracovního prostoru. Zdrojová data velmi často obsahují chyby, nekonzistence, chybějící data atd. Pokud čerpáme z různých zdrojů, musíme provést jejich propojení například přes klíčové položky. Této fázi se říká čištění a standardizace. Pokud ji provedeme špatně, vstupní data pro strojové učení budou nekvalitní a výsledek bude také nekvalitní.

Pokud jsme vybrali algoritmus, který vyžaduje trénovací data, musíme trénovací data připravit. Časově se bude jednat o nejnáročnější a nejdelší část projektu, která je ovšem klíčová pro úspěšný projekt. V učebním datasetu musí být obsaženy všechny možné případy, které se vyskytují v realitě. Musíme tedy začlenit i nestandardní, ale platná data. Pokud budeme rozpoznávat výrobky z obrázků, musíme mít obrázky všech možných výrobků, které se mohou v procesu vyskytovat. Pokud je možné, aby byly výrobky různě natočeny, musíme dodat snímky výrobků focené z různých pohledů.

Pokud bude umělá inteligence pracovat za různých podmínek, musíme doplnit odpovídající data. V případě zpracování obrazu v nekontrolované světelné scéně, musíme doplnit obrázky snímané za různého počasí, sluneční aktivity atd.

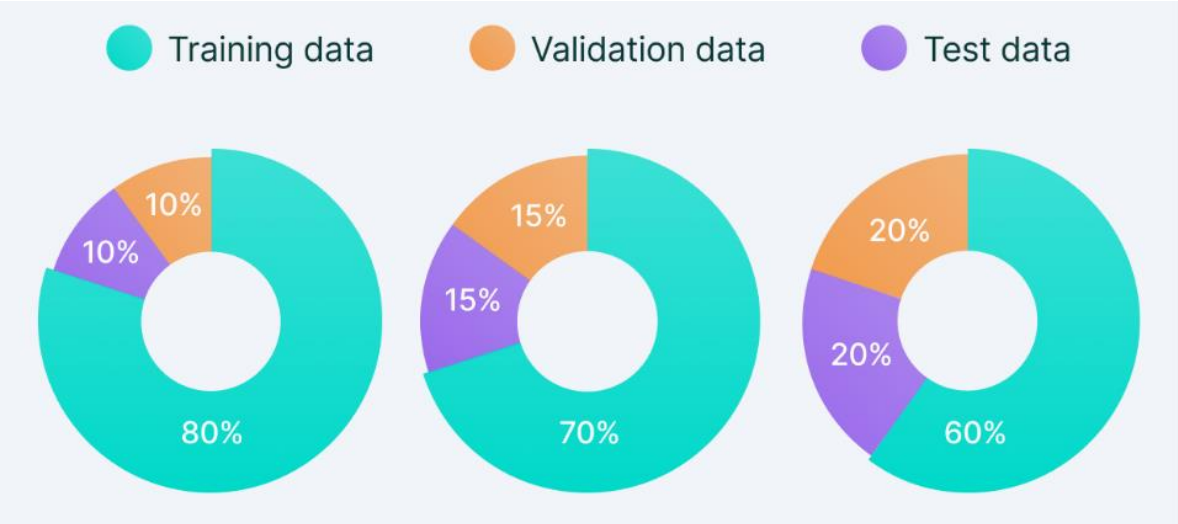
V některých případech nemusí být všechna data dostupná nebo mohou být různě zašuměná. Můžeme se je pokusit nahradit například za 0, průměrné hodnoty, nebo model naučíme pracovat i s neúplnými daty.

V některých případech můžeme použít pro rozšíření trénovací sady augmentaci. Jedná se vytvoření nových dat na základě stávajících dat. V případě obrazových dat se může jednat o rotace, překlopení, zašumění atd. Některá data nelze z etických nebo právních předpisů přímo použít, protože se jedná citlivé osobní údaje. V tomto případě můžeme použít různé metody anonymizace. V dnešní době se používají GAN (Generative adversarial network), které vytváří nová trénovací data.

Data je vhodné upravit pro konkrétní algoritmus. Pokud například lépe pracuje s normalizovanými hodnotami, je vhodné data normalizovat a nepoužívat původní hodnoty. U obrázků můžeme zvolit různé reprezentace (RGB, VHS).

Pro algoritmy učení s učitelem je třeba trénovací dataset rozdělit ideálně na 3 části. První částí je **tréninková** skupina údajů, která se používá pro samotné učení. Protože se model trénuje na tréninková data, určovat jeho přesnost podle těchto dat není vhodné. Proto se vytváří **testovací** skupina dat. Na základě těchto dat určujeme přesnost modelu.

Poslední skupinou je validační skupina dat. Tuto kontrolní skupina používáme například pro výběr správného modelu, ladění hyperparametrů modelu atd.



Obrázek 7 - Rozdělení trénovacích dat

Další fází je **učení**. V této fázi volíme konkrétní algoritmus umělé inteligence. Algoritmu jsou předkládána trénovací data, podle kterých si pomalu upravuje své parametry. Toto se mnohonásobně opakuje. Když už nedochází ke zlepšování, přesnost modelu se testuje na testovacích datech.

Pokud model nesplňuje naše požadavky vracíme se do nějaké z předchozí fáze. Hledáme lepší zdroje dat, data lépe čistíme, doplňujeme učební data set apod. Případně volíme jiný přístup, jiný algoritmus.

Po otestování přichází poslední fází je **nasazení** modelu pro produkce. V této fázi je model zapojen do firemního procesu. Je důležité zjišťovat zpětnou vazbu, jeho výsledky a porovnávat je se skutečností. Pokud model nevyhovuje nebo se změní podmínky a vyskytnou se nová data, která model neumí zpracovat, vracíme se k nějaké z předchozích fází.

6 Reference

[1] K. Warwick, Úsvit robotů, soumrak lidstva, Praha: Vesmír, 1999.

[2] I. Zelinka, Umělá inteligence - hrozba nebo naděje, 2023: BEN - technická literatura, Brno.

[3] M. Minsky, „Steps Toward Artificial Intelligence," v *Proceedings of IRE 49*, 1961.

[4] W. P. Warren S. McCulloch, „A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics,* 1943.

[5] D. Hebb, The Organization of Behavior, New York: Wiley and Sons, 1949.

[6] F. Rosenblatt, „The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review 65,* 1958.

[7] D. K. J. B. a. R. F. Block, „Analysis of a four-layer series-coupled perceptron.," *Reviews of Modern Physics 34,* 1962.

[8] M. P. S. P. Minsky, „An Introduction to Computational Geometry.," v *MIT Press*, Cambridge, 1969.

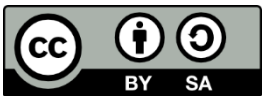
[9] J. Hopfield, „Neural networks and physical systems with emergent collective computational abilities," v *Proceedings of the National Academy of Sciences of the United States of America 79*, 1982.

[10] D. H. G. a. W. R. Rumelhart, „Learning representations by back-propagating errors," v *Nature 323*, 1986.

[11] T. Kohonen, „Self-organized formation of topologically correct feature maps,“ v *Biological Cybernetics 43*, 1982.

UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vytvořeno v rámci projektu **Digitalizace studijních Agend, Nové TechnologiE, systémy a přístupy k výuce na UPCE**, reg. č. NPO_UPCE_MSMT-16591/2022.



Financováno Evropskou unií
NextGenerationEU

Národní plán obnovy

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Machine learning & AI

Statistický popis dat, korelace dat

Ing. Martin Pozdílek, Ph.D.

1 Statistika

Při vytváření modelů pro umělou inteligenci velmi často vycházíme ze statistických metod. I v případě, že přímo nepoužíváme statistické metody, je vhodné o datových souborech znát základní informace a vyvarovat se tak zásadních chyb při vytváření modelů. Některé modely mají omezující podmínky a lze je použít pouze na určitý typ dat. Bez detailnějších znalostí o vstupních datech bude náš projekt velmi často odsouzen k neúspěchu. Proto se v této kapitole nejprve podíváme na průzkum a přípravu dat.

1.1 Základní pojmy

Statistický soubor, datová množina – soubor objektů, prvků, jedinců, statistických jednotek, kterých se týká statistické zjišťování. Soubor musí být vymezen věcně, časově a prostorově.

Statistické jednotky – elementární jednotky statistického zjišťování, modelování.

Znaky, proměnné – reprezentace vybraných vlastností statistických jednotek. Jedna vlastnost může být popsána více znaky. Znak může být určující, tedy ten, který určuje příslušnost do statistického souboru. Případně znak může být zkoumaný, který je předmětem zjišťování.

Data – naměřené a zaznamenané hodnoty sledovaných znaků v souboru prvků.

Data lze získávat pomocí

- Pokus, experiment
- Pozorování
- Rozhovor
- Dotazník
- Dokumentace

1.1.1 Typy znaků

Znaky lze rozdělit podle úrovně kvantifikace. Tu určujeme, zda je u nich možné provést operace rovnosti, porovnání, rozdílu a podílu. Některé metody vyžadují určitý typ znaků. Data znaků na vyšším stupni kvantifikace lze zpracovat metodami určenými pro nižší stupeň kvantifikace, ovšem za cenu ztráty informace. Opačný postup možný není.

- **Nominální** – lze provést pouze porovnání na rovnost nebo nerovnost
    - **Alternativní** – dvě možnosti (pohlaví, pravda/lež)
    - **Množné** – více možností (barva, typ)
- **Ordinální** – znaky lze seřadit od nejmenšího k největšímu, lze porovnat, ale nelze určit rozdíl (známky ve škole, spokojenost, …)

- **Kardinální**, metrické – lze určit rovnost, řazení, rozdíl. Jsou vždy číselné. (výška, váha, hmotnost, …)
    - **Intervalové** – jsme u nich schopni interpretovat rozdíl dvou hodnot. Stejný interval mezi jednou a druhou dvojicí hodnot vyjadřuje i stejný rozdíl v intenzitě zkoumané vlastnosti.
    - **Poměrové** – mimo rozdílu jsme schopni interpretovat i podíl 2 hodnot
    - **Spojité**
    - **Nespojité**

1.1.2 Typy datových souborů

Statistický soubor může být **základní** (populace), který obsahuje všechny definované prvky. Někdy může být populace konečná nebo i nekonečná.

Jiným typem souboru je **výběrový**. Výběrový soubor obsahuje podmnožinu populace. Způsob vybrání prvků z populace do výběru musí odpovídat pravidlům reprezentativnosti. Při špatně vytvořeném výběrovém souboru zjištěné informace nebo vytvořený model nebudou použitelné na základním souboru.

S výběrovým souborem se váže pojem **náhodný výběr**. Jedná se o proces výběru, který zajistí, že prvky z populace jsou vybrány nezávisle a měly stejnou pravděpodobnost být do výběru zahrnuty. Náhodnost výběru je vždy ovlivněna určitou chybou při vybírání. K vybírání se proto používají způsoby, které chybu při vybírání zmenšují co nejvíce. Výběr může být s vrácením prvků do populace nebo bez vrácení. Při vrácení může být prvek vybrán do výběru vícekrát.

1.1.3 Vady dat

Než začneme data analyzovat, uveďme si několik příkladů dat, která mohou být nevhodná.

- Agregovaná data (suma, průměr) – při zkoumá vztahů proměnných, agregace může vztah zakrýt
- Dynamická data – spolu s hodnotami proměnných se mění i vztah proměnných
- Nominální data

1.2 Charakteristiky datových sad

Datové sady mohou být velmi obsáhlé a usuzovat na jejich vlastnosti pohledem na všechna data je nemožné. Proto existují různé charakteristiky, které nám umožní udělat si rychlý obrázek o datové sadě, aniž bychom museli procházet všechny údaje. V praxi není někdy možné z různých důvodů charakteristiku spočítat nad celou populací. Proto ji počítáme na výběrovém souboru a pak mluvíme o **výběrových charakteristikách**, které se nemusí přesně shodovat s charakteristikami

populace. Podle zavedené konvence se používají pro označování charakteristik populace řecká písmena a pro označování výběrových charakteristik písmena latinské abecedy.

Výpočtem odhadů přesných hodnot parametrů základního souboru se zabývají speciální statistické metody odhadování parametrů, kdy získáváme bodový nebo intervalový odhad.

1.2.1 Charakteristiky polohy

Základní charakteristikou je **střední hodnota**. Jedná se o jedno číslo, které se snaží něco povědět o všech hodnotách jednoho z parametrů. Jedná se o číslo, které ukazuje na střed naměřených hodnot. Pro střední hodnotu můžeme použít více postupů.

1.2.1.1 Aritmetický průměr

Střední hodnota populace μ se často počítá jako aritmetický průměr. Odhad střední hodnoty populace z výběrového souboru $\bar{x}$ se počítá velmi podobně, ale z omezeného počtu hodnot. Obecně pro populační charakteristiky se používají řecká písmena a pro výběrové charakteristiky běžná písmena (latinka).

$$\mu = \frac{\sum_{i=1}^{N} x_i}{N}$$

$$E(X) = \bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

Aritmetický průměr je ovlivněn extrémními hodnotami. Extrémními hodnotami souboru rozumíme tzv. odlehlá pozorování, což bývá obvykle jedna nebo několik málo hodnot náhodné proměnné, které jsou oproti ostatním zjištěným hodnotám příliš malé nebo příliš velké. Aritmetický průměr je správnou charakteristikou středu souboru pouze tehdy, je-li soubor z hlediska zkoumaného znaku dostatečně stejnorodý. V ostatních případech, hlavně při malém rozsahu souboru, může být aritmetický průměr zkreslen případnými extrémními hodnotami souboru. Typickým příkladem je aritmetický průměr platů, který je vychýlený například platem generálního ředitele.

1.2.1.2 Vážený aritmetický průměr

Vážený aritmetický průměr se používá, pokud mají různé hodnoty jinou důležitost. Pak je u každé hodnoty x uvedena i její váha w. Typický příkladem je například výpočet konečné známky ve škole, kdy jednotlivé známky mají různou váhu (pololetní písemka, 5minutovka, …)

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i \cdot w_i}{\sum_{i=1}^{n} w_i}$$

1.2.1.3 Geometrický průměr

Geometrický průměr je definovaný jako n-tá odmocnina součinu nezáporných čísel. Geometrický průměr je vždy menší nebo rovný než aritmetický průměr.

$$\overline{x_g} = \sqrt[n]{x_1 \cdot x_2 \dots \cdot x_n}$$

1.2.1.4 Harmonický průměr

Harmonický průměr kladných čísel je definován jako podíl počtu těchto čísel a součtu jejich převrácených hodnot. Používá se, pokud potřebujeme hodnotu, která zastupuje ostatní, co se týče převrácených hodnot, například při výpočtu průměrné rychlosti na úsecích stejné délky. Dále jsou-li hodnoty znaku nerovnoměrně rozloženy kolem aritmetického průměru, nebo když jsou hodnoty extrémně nízké či vysoké. Harmonický průměr je vždy menší než geometrický průměr, nebo je mu roven.

$$\overline{x_h} = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$$

1.2.1.5 Medián

V některých případech se pro odhad střední hodnoty místo různých průměrů, které mohou být zatíženy extrémními hodnotami, používá medián. **Medián** můžeme definovat jako 50 % kvantil. Jedná se o hodnotu, která rozděluje seřazené hodnoty na dvě stejně velké části. Hodnoty nalevo od mediánu jsou menší nebo rovny a hodnoty napravo jsou větší nebo rovny mediánu. Pokud je počet hodnot lichý, tak medián je prostřední hodnota. Pokud je počet hodnot sudý, pak existují dvě prostřední hodnoty a medián je definován jako jejich aritmetický průměr.

Medián je vhodné používat, pokud soubor obsahuje extrémní hodnoty nebo v případě souborů s neznámým rozdělením, které může být různě šikmé nebo obsahuje více vrcholů.

1.2.1.6 Modus

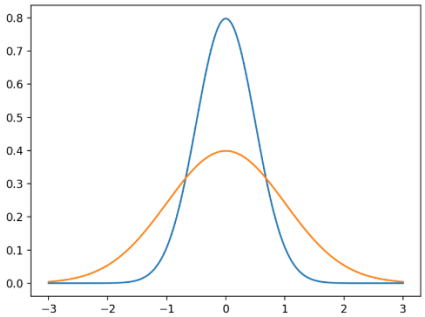
Modus můžeme definovat jako nejčastěji se vyskytující hodnota proměnné v souboru, hodnota s největší četností. Odpovídá tedy vždy vrcholu křivky rozdělení. V tabulce rozdělení četností se modus určí jednoduše z hodnoty znaku, která má největší četnost. V rozděleních četností, kde jsou jednotlivé hodnoty řazeny do tříd s třídními intervaly (tj. u intervalového rozdělení četností), mluvíme o modálním intervalu (třída s nejvyšší četností).

Modus není zkreslen případnými extrémními hodnotami souboru. Je patrné, že modus je konkrétní hodnota, která není přímo ovlivněna velikostí všech hodnot dané proměnné. Lze ho použít jako vhodnou charakteristiku střední hodnoty i v případě veličin s neznámým rozdělením.

1.2.2 Charakteristiky variability

Dosud jsme se zabývali střední hodnotou statistického souboru. Jedná se o základní charakteristiku, která nemusí vypovídat o datovém souboru. Na následujícím grafu jsou zobrazeny distribuční funkce dvou různých datových souborů, které mají ovšem stejnou střední hodnotu.

Proto je důležité střední hodnotu doplnit i o charakteristiky variability, které ukazují, jak jsou hodnoty rozptýleny a dodávají detailnější pohled na data.



Obrázek 1 - různá rozdělení se stejnou střední hodnotou

1.2.2.1 Variační rozpětí

Nejjednodušší charakteristikou je **variační rozpětí** (range). Jedná se o rozdíl mezi maximální a minimální hodnotou. Jedná se o ne příliš přesnou charakteristiku, která může být velmi ovlivněna extrémními hodnotami. I v přechozím obrázku oba datové soubory mají stejný variační rozptyl.

$$R = x_{max} - x_{min}$$

Nedostatek variačního rozpětí lze překonat pomocí rozpětí kvantilů (viz dále). Nejpoužívanější je mezi kvartilové rozpětí IQR, které určuje rozpětí mezi 50 % prostředních hodnot.

$$IQR = R_q = x_{0,75} - x_{0,25}$$

1.2.2.2 Rozptyl

Rozptyl (variance) můžeme definovat jako aritmetický průměr čtverců odchylek jednotlivých hodnot sledované proměnné x_i od průměru celého souboru. Rozptyl můžeme počítat pro populaci σ^2nebo pro výběrový soubor s^2. Pro n > 30 jsou rozdíly mezi σ^2a s^2 zanedbatelné.

$$\sigma^2 = \frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N}$$

$$D(X) = s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

Jestliže jsou všechny hodnoty souboru stejné, potom je variabilita hodnot sledované proměnné v souboru nulová a výběrový rozptyl $s^2 = 0$. Velikost rozptylu se zvyšuje při zvětšující se variabilitě hodnot sledované proměnné. Rozptyl je odvozen od součtu čtverců odchylek jednotlivých hodnot od průměru souboru, proto nemůže nikdy nabývat záporných hodnot. Přičte-li se nebo odečte-li se ke všem hodnotám proměnné X libovolná kladná konstanta a, potom se rozptyl nezmění. Násobí-

li se všechny hodnoty proměnné nenulovou konstantou *g*, potom je rozptyl znásoben čtvercem této konstanty.

1.2.2.3 Směrodatná odchylka

Směrodatná odchylka (standard deviation) je odvozená jednoduše od rozptylu. Směrodatná odchylka může nabývat vždy pouze kladných hodnot.

$$\sigma = \sqrt{\sigma^2}$$

$$s = \sqrt{s^2}$$

1.2.2.4 Variační koeficient

Variační koeficient (coefficient of variation), relativní směrodatná odchylka je vhodný pro vzájemné srovnávání variability dvou nebo více souborů s podstatně odlišnou úrovní hodnot. V těchto případech musíme odstranit vliv úrovně daných hodnot. Směrodatnou odchylku dělíme střední hodnou (obvykle aritmetickým průměrem), od které byly počítány odchylky pro součet čtverců. Výsledek se obyčejně vyjadřuje v procentech. Variační koeficient udává, z kolika procent se podílí směrodatná odchylka na aritmetickém průměru.

$$V = \frac{\sigma \cdot 100}{\mu}$$

$$V = \frac{s \cdot 100}{\bar{x}}$$

1.3 Rozdělení náhodné veličiny

Náhodný jev je výsledek náhodného pokusu. Například hod kostkou. Náhodné jevy jsou složeny z elementárních jevů.

Elementární jevy jsou takové, které se navzájem neslučují. Například padnutí 1 při hodu kostkou. Všechny elementární jevy tvoří prostor elementárních jevů V.

Náhodný pokus je realizace určitého komplexu podmínek, jejichž výsledkem je jev A, který nedovedeme předvídat s jistotou. Například jeden hod kostkou.

Výskyt náhodných jevů je vyjádřen pravděpodobností jejich výskytu. Pravděpodobnost můžeme vyjádřit klasicky pomocí poměru m a n. m je počet výsledků, které mají za následek nastoupení jevu A. n je počet možných výsledků prvotních jevů se stejnou možností výskytu. Teoretické vyjádření pravděpodobnosti.

$$P(A) = \frac{m}{n}$$

Případně lze pravděpodobnost vyjádřit pomocí četnosti, kdy m je počet případů, v nichž nastal jev A a n je počet nezávislých opakování pokusu. Pravděpodobnost tedy empiricky změříme.

$$P(A) \approx \frac{m}{n}$$

1.3.1 Náhodná veličina

Zkoumaný statistický znak je možno popsat pomocí veličiny, tedy projevu číselně vyjádřeného znaku. Tato veličina za určitých podmínek vlivem náhodných činitelů nabývá různých hodnot. Označujeme ji proto pojmem **náhodná veličina**. Náhodná veličina představuje všechna data získaná měřením sledovaného statistického znaku v nějakém pozorování. Případně náhodná veličina je přiřazení číselné hodnoty náhodnému jevu.

Jednou z prvních operací při statistickém zpracování těchto dat je jejich roztřídění a uspořádání. **Variační řada** vznikne seřazením všech hodnot od nejmenší po největší. Pokud se nějaká hodnota opakuje, mluvíme o frekvenci hodnoty.

1.3.2 Kvantily

Kvantil můžeme definovat jako hodnotu náhodné proměnné *x*, která rozděluje soubor dat na dvě části. Hodnoty, které jsou nižší, než je kvantil a hodnoty, které jsou vyšší než kvantil.

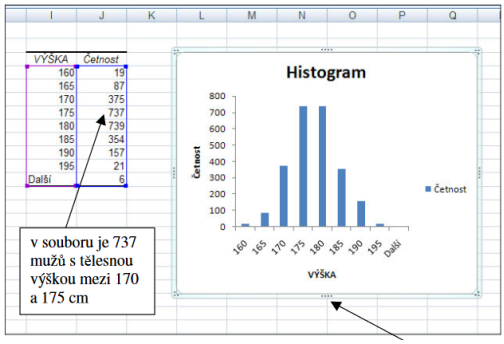
Například 20% kvantil rozdělí variační řadu na 20 % nižších hodnot a 80 % vyšších hodnot. Některé kvantily jsou statisticky významné a mají své názvy. $X_{0,5}$ je medián, který data rozděluje na dvě poloviny. Dále máme tři kvartily $x_{0,25}$, $x_{0,5}$ a $x_{0,75}$, které rozdělují data na 4 stejně velké části. Své pojmenování mají decily pro rozdělení dat na desetiny a percentily, které data rozdělují na sto částí.

Kvantily nám naznačují, jak jsou hodnoty rozděleny. Čím detailnější jsou kvantily, tím je představa přesnější.

1.3.3 Rozdělení náhodné veličiny

Rozdělení náhodné veličiny vyjadřuje výskyt (četnost, frekvenci) hodnot náhodné veličiny v závislosti na dané hodnotě náhodné veličiny. To nám umožňuje vytvoření představy, jak jsou různé hodnoty náhodné veličiny na číselné ose měřené veličiny rozmístěny.

Pro diskrétní náhodnou veličinu, veličinu, která může nabývat pouze určitých hodnot lze rozdělení znázornit buď četnostní tabulkou nebo histogramem. Četnosti mohou být **absolutní, relativní nebo kumulativní absolutní a kumulativní relativní**.



Obrázek 2 četnostní tabulka a histogram

Spojitou náhodnou veličinu můžeme převést na diskrétní náhodnou veličinu vytvořením kategorií pomocí intervalů hodnot.

1.3.3.1 Distribuční funkce

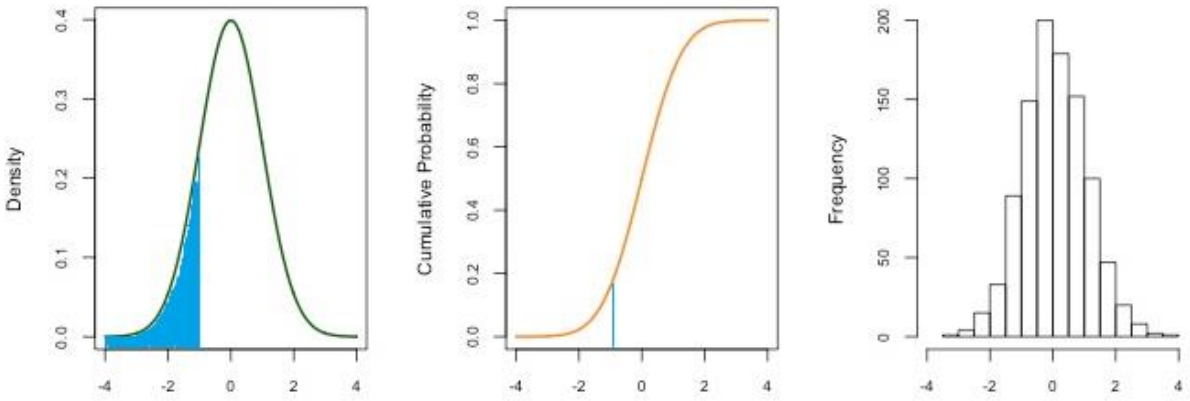
Pokud nechceme při zjišťování rozdělení spojené náhodné veličiny přijít o přesnost rozdělováním do intervalů, budeme muset použít distribuční funkci.

Distribuční funkce je úzce spojena s kvantilem. Jak již bylo řečeno, kvantil může být libovolné číslo, které rozděluje variační řadu na dvě části. Každému kvantilu, libovolné hodnotě náhodné proměnné x, je definována distribuční funkce *F(x).* Distribuční funkci *F(x)* můžeme definovat jako pravděpodobnost, že náhodná veličina (proměnná *X*) nabude hodnoty menší (případně rovné) než určitá hodnota *x*. Distribuční funkce $F(x_i)$ je tedy vždy přiřazena ke konkrétní hodnotě náhodné veličiny x_i.

$$F(x_i) = P(X < x_i)$$

Distribuční funkce je definována pro všechna reálná čísla *x*, má tedy smysl pro hodnoty v intervalu: $-\infty < x < +\infty$. Pro tato reálná čísla pak distribuční funkce $F(x_i)$ může nabývat hodnot v intervalu $\langle 0 ; +1 \rangle$. Hraniční body jsou $F(-\infty) = 0$ a $F(+\infty) = 1$. Distribuční funkce je neklesající.

V grafickém vyjádření je to část plochy pod křivkou pravděpodobnostního rozdělení ohraničená kvantilem x_p. Na následujících grafech jsou zachycena data. Poslední graf ukazuje histogram. Díky tomu, že jsou použité intervaly, je histogram zubatý. Levý obrázek ukazuje spojité rozdělení hodnot. Medián rozdělení je 0, tedy 50 % hodnot je záporných a 50 % hodnot kladných. Kvantil $x_{0,1}$ nabývá hodnoty přibližně -1. Modrá plocha pod křivkou určuje pravděpodobnost, že náhodná veličina x bude nabývat hodnot menších než -1. Na prostředním grafu je znázorněna distribuční funkce pro toto rozdělení. Pro hodnotu -1 nabývá hodnoty přibližně 0,1. Kvantil $x_{0,9}$ nabývá hodnoty přibližně 2. To znamená, že hodnot menších než 2 bude v souboru okolo 90 %.



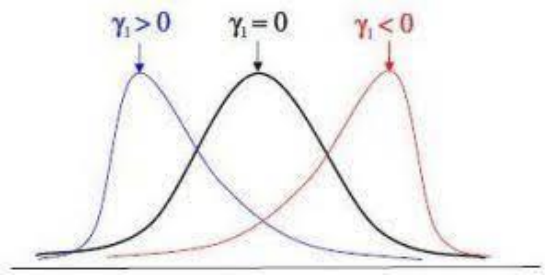
Obrázek 3 – hustota rozdělení, distribuční funkce, histogram

1.3.4 Charakteristiky šikmost a špičatost

Koeficient šikmosti je charakteristika rozdělení náhodné veličiny, která popisuje jeho nesymetrii. Označuje se symbolem γ_1.

$$\gamma_1 = \frac{E[X - E(X)]^3}{(var\ X)^{3/2}}$$

Nulová šikmost značí, že hodnoty náhodné veličiny jsou rovnoměrně rozděleny vlevo a vpravo od střední hodnoty. Kladná šikmost značí, že vpravo od průměru se vyskytují odlehlejší hodnoty nežli vlevo (rozdělení má tzv. *pravý ocas*) a většina hodnot se nachází blízko vlevo od průměru. U záporné šikmosti je tomu naopak. E(X) je střední hodnota, var(X) je rozptyl.

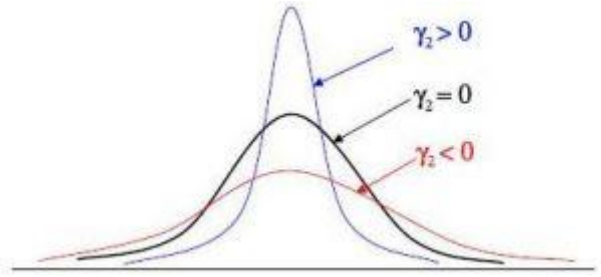


Obrázek 4 – různě šikmá rozdělení

Koeficient špičatosti je charakteristika rozdělení náhodné veličiny, která porovnává dané rozdělení s normálním rozdělením pravděpodobnosti. Koeficient špičatosti se obvykle označuje γ_2.

$$\gamma_2 = \frac{E[X - E(X)]^4}{(var\ X)^3}$$

Normální rozdělení má špičatost tři. Kladná špičatost značí, že většina hodnot náhodné veličiny leží blízko její střední hodnoty a hlavní vliv na rozptyl mají málo pravděpodobné odlehlé hodnoty. Křivka hustoty je špičatější nežli u normálního rozdělení. Záporná špičatost značí, že rozdělení je rovnoměrnější a jeho křivka hustoty je plošší nežli u normálního rozdělení.



Obrázek 5 – různě špičatá rozdělení

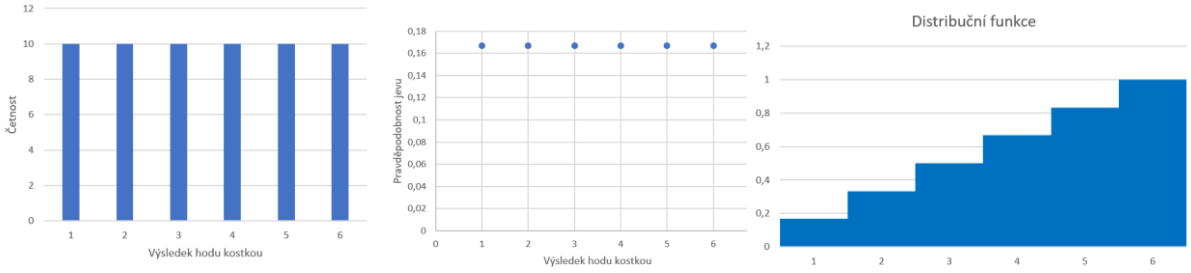
1.3.5 Typy rozdělení

1.3.5.1 Rovnoměrné rozdělení

Rovnoměrné rozdělení pravděpodobnosti přiřazuje všem hodnotám náhodné veličiny stejnou pravděpodobnost. Rovnoměrné rozdělení má svoji diskrétní i spojitou podobu. Typickým příkladem je hod vyváženou hrací kostkou. Rozdělení má svou minimální hodnotu a maximální hodnotu b.

$$\bar{x} = \frac{a+b}{2}$$

$$s^2 = \frac{(b-a)^2}{12}$$



Obrázek 6 – rovnoměrné rozdělení

1.3.5.2 Normální rozdělení

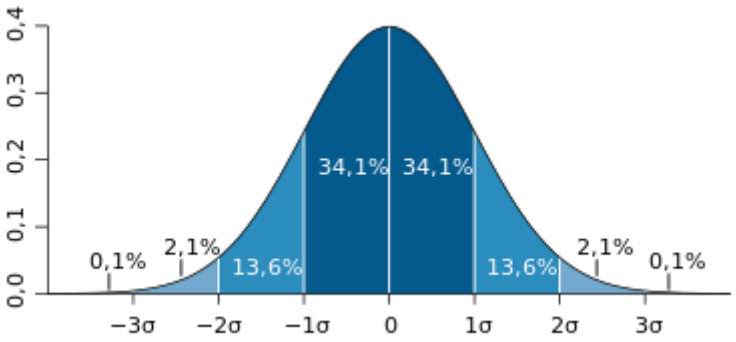
Normální rozdělení je rozdělení spojité náhodné veličiny. Vzorec objevili údajně nezávisle na sobě Abraham de Moivre v roce 1711 při snaze o nalezení aproximace binomického rozdělení na základě experimentů s hody mincí. Na konci 18. století je znovuobjevil Gauss a Laplace jako křivku chyb. Lambert Quételet později zjistil, že řada biologických veličin vykazuje v populaci rozdělení podobné „křivce chyb“. V praktickém životě se setkáváme s řadou veličin, které mají toto rozdělení.

Hustota rozdělení je symetrická zvonovitá křivka. Jako teoretický model má významné postavení ve statistice, je limitní pro řadu jiných rozdělení jako binomické, poissonovo, chí-kvadrát, …).

Normální rozdělení má dva parametry – střední hodnotu μ a rozptyl σ^2. Střední hodnota určuje střed rozdělení a rozptyl rozevření křivky hustoty. Obecně se značí $N(\mu, \sigma^2)$. Hustota f(x) je popsána následujícím vztahem:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(x-\mu)^2}{2\sigma^2}}$$

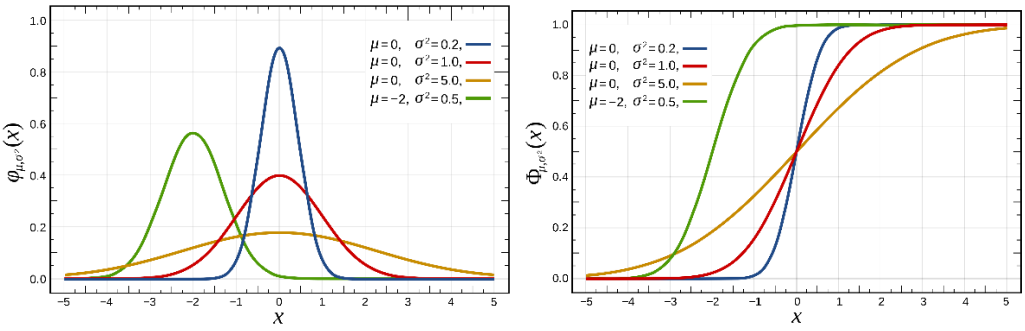
Rozdělení N(0, 1) bývá označováno jako normované (nebo standardizované) normální rozdělení. Z níže uvedeného grafu například vyplývá, že 68,2 % hodnot se vyskytuje v intervalu <- σ, + σ>. Hodnot nad μ + 3σ je méně než 1 %. Například vědecké výzkumy se považují za potvrzené nad 5σ, když je minimální pravděpodobnost, že se jedná o náhodu.



Obrázek 7 - normální rozdělení N(0,1)

Náhodnou veličinu X s normálním rozdělením a libovolnými parametry N(μ, σ²) je možné pomocí standardizované proměnné $z = \frac{z-\mu}{\sigma}$ převést na veličinu s rozdělením N(0, 1).

Při vytváření modelů se velmi často používá normalizace proměnné s normálním rozdělením na proměnnou se standardizovaným normálním rozdělením. Důvodem je, že mnoho algoritmů s tímto rozdělením pracuje mnohem lépe a nachází řešení v kratším čase.



Obrázek 8 - různá normální rozdělení, jejich hustoty a distribuční funkce

2 Grafické zobrazení

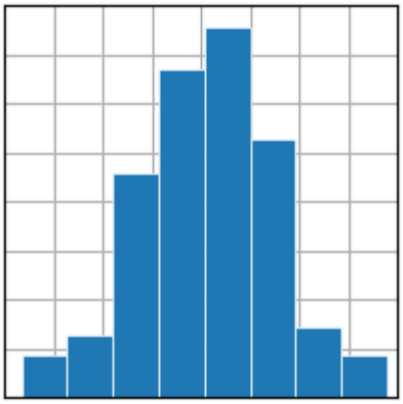
Dosud jsme data ve statistickém souboru analyzovali pomocí číselných charakteristik. I přes jejich vypovídající hodnotu někdy si rychlou představu o datech můžeme udělat z různých grafů. Ukážeme si několik základních typů grafů, které existují v mnoha variantách. Existuje, ale celá řada

jiných grafů, které data například zobrazují na geografické mapě atd. Při vytváření je třeba být kreativní, protože co jeden typ grafů nezobrazuje, jiný to ukáže na první pohled.

2.1 Grafy pro statistická rozdělení

2.1.1 Histogram

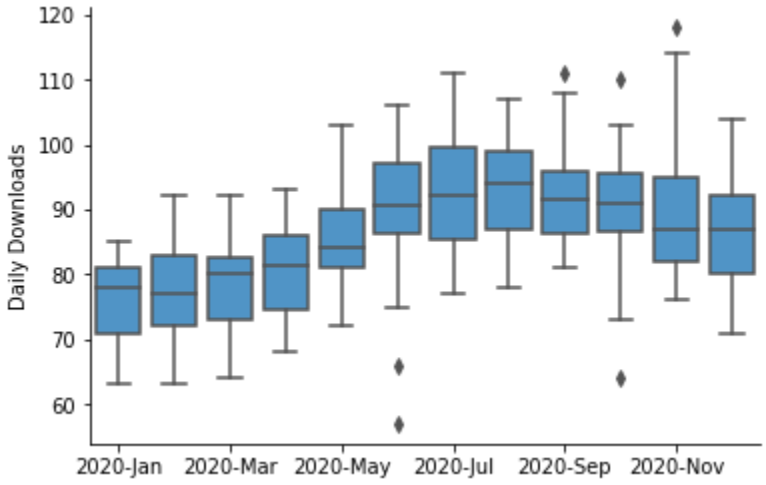
Jednoduchý graf pro zobrazení distribuce hodnot jedné proměnné. Je vhodný i pro nespojitá data. U spojitých dat si můžeme pomoci vytvořením intervalů.



Obrázek 9 - hist(x)

2.1.2 Krabicový diagram

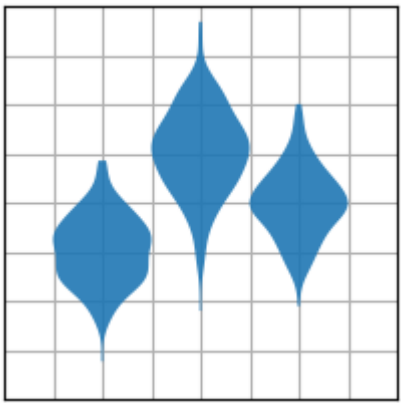
Krabicový diagram (boxplot) se často používá pro kompaktní zobrazení distribuce více proměnných. Základním tvarem je obdélník, jehož velikost určuje 1. a 3. kvartil. Obdélník je rozpůlen čárou na úrovni 2. kvartilu – střední hodnoty. Čím je velikost obdélníku menší, tím jsou data méně rozptýlená. Z obdélníku vychází úsečky (někdy nazývané fousy), které začínají na jedné straně od P_{25} - 1,5IQR a na druhé straně P_{75} + 1,5 IQR. Připomeňme, že IQR je rozdíl mezi 3. a 1. kvartilem. Pokud datový soubor obsahuje extrémní data v rozmezí od 1,5 do 3 IQR, tak jsou zobrazeny nad, respektive pod čárami. Některé podoby krabicového diagramu odlišují i data na 3,0 IQR.



Obrázek 10 - Krabicový diagram s časovými daty – boxplot(X)

2.1.3 Houslový graf

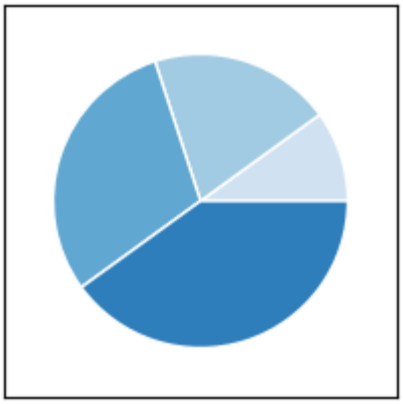
Houslový graf, známý také jako „violin plot", je typem vizualizace dat, který kombinuje boxplot a hustotní graf. Tento graf reprezentuje ve vertikálním směru rozložení hodnot.



Obrázek 11 - violinplot(X)

2.1.4 Koláčový graf

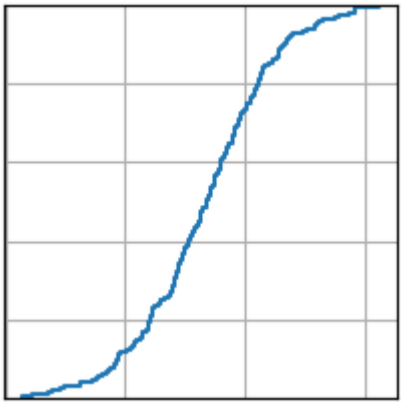
Koláčový graf je oblíbený u manažerských prezentací. Umožňuje zobrazovat relativní data, procentuální data.



Obrázek 12 - pie(x)

2.1.5 Distribuční funkce

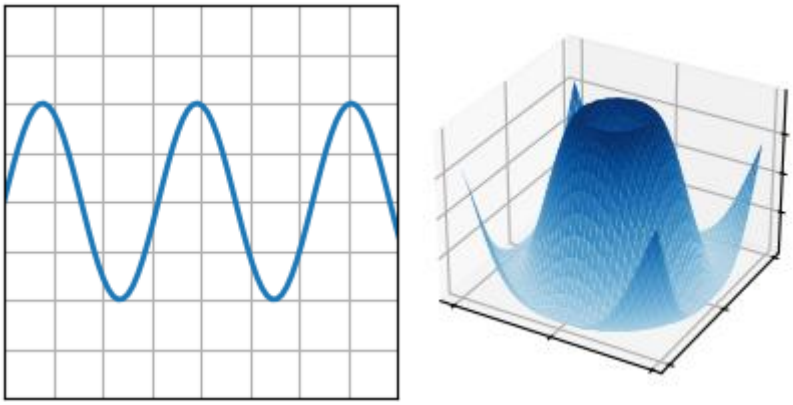
Graf empirické distribuční funkce zobrazuje distribuční funkci proměnné.



Obrázek 13 - ecdf(x)

2.2 Grafy pro zobrazení vztahů více proměnných

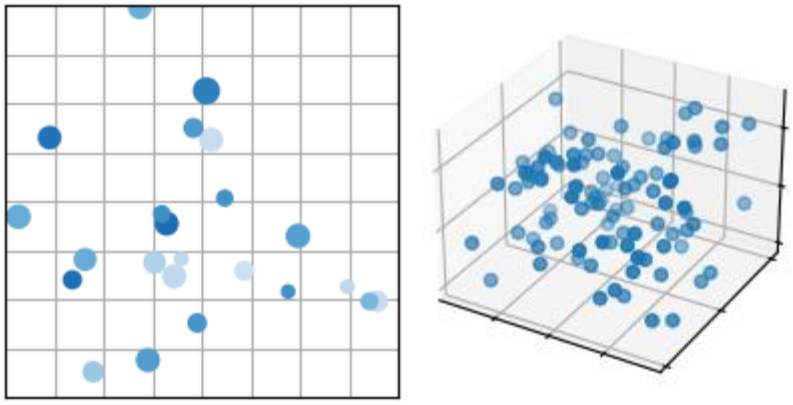
2.2.1 Spojnicový graf



Obrázek 14 - plot(x, y), plot_surface (x, y, z)

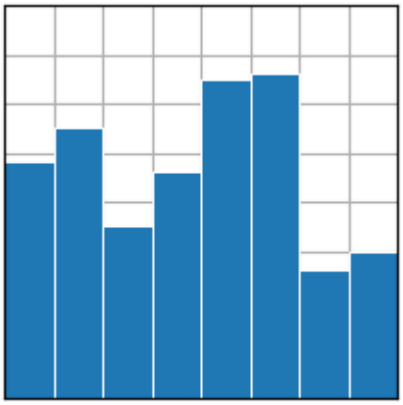
2.2.2 Bodový graf

V některých případech místo propojení bodů je vhodné vykreslit jen samotné body.



Obrázek 15 - scatter(x,y)

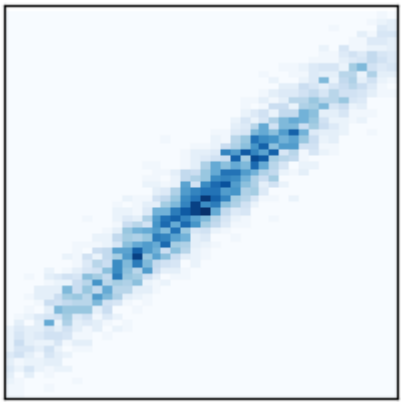
2.2.3 Sloupcový graf



Obrázek 16 - bar(x,y)

2.2.4 Distribuční graf

V některých případech je vhodné si zobrazit distribuci dvojice proměnných x, y.



Obrázek 17 – hist2d(x, y)

3 Vztahy proměnných

V datovém souboru se vyskytuje více proměnných. Můžeme je rozdělit na vysvětlující a vysvětlované. Vysvětlující proměnná (také známá jako nezávislá proměnná) je proměnná, kterou používáme k předpovědi hodnoty jiné proměnné. Vysvětlovaná proměnná (také známá jako závislá proměnná) je proměnná, jejíž hodnoty se snažíme předpovědět nebo vysvětlit.

Příklad: Představte si, že provádíme studii, abychom zjistili, jaký vliv má počet hodin studia na výsledky testu studentů. V tomto případě je „počet hodin studia" vysvětlující proměnná, protože ji používáme k předpovědi výsledků testu. „Výsledky testu" jsou vysvětlovaná proměnná, protože se snažíme předpovědět nebo vysvětlit jejich hodnoty na základě počtu hodin studia.

Například během experimentu budeme realizovat sérii měření, ve které budeme měřit různé hodnoty a zároveň budeme schopni experiment řídit změnou vstupních parametrů. Například budeme měřit rychlost ohřevu (čas) a u jednotlivých měření budeme moci manipulovat s tlakem, příkonem, místem a časem provozování experimentu.

Pro další analýzy je vhodné určit vztah mezi proměnnými. Zajímá nás, zda dané parametry experimentu mají vliv na výsledek. Případně jestli změna jednoho parametru (tlaku) má vliv i na jiný parametr.

Během zjišťování kovariance určujeme, zda mezi proměnnými je lineární závislost a pokud ano, jak silná a jaký má směr. Korelace nám nic neříká o postavení proměnných jako příčina nebo důsledek.

Mnoho statistických výzkumů ztroskotalo na objevení korelace mezi proměnnými, ale bez dalšího ověření, že se jedná o příčinnou souvislost. Tento problém se označuje jako lživá korelace.

Jedním z klasických příkladů, kdy dvě proměnné mají silnou korelaci, ale nemají příčinnou souvislost, je vztah mezi prodejem zmrzliny a počtem útoků žraloků.

V průběhu let bylo zjištěno, že když se zvyšuje prodej zmrzliny, zvyšuje se také počet útoků žraloků. Na první pohled by se mohlo zdát, že mezi těmito dvěma proměnnými existuje příčinná souvislost, ale ve skutečnosti je to nesprávné.

Skutečným důvodem této korelace je skrytá proměnná, kterou je teplota. V teplých letních měsících lidé častěji kupují zmrzlinu a také častěji plavou v moři, což zvyšuje pravděpodobnost setkání s žralokem. Takže i když prodej zmrzliny a počet útoků žraloků jsou korelované, neexistuje mezi nimi příčinná souvislost.

3.1 Kovariance

Kovariance C(X, Y) je statistická charakteristika, která určuje vzájemnou závislost veličin X a Y. Je definována jako střední hodnota součinu odchylek veličin X a Y od jejich středních hodnot.

Vzorec pro výpočet kovariance je následující, kde X a Y jsou nezávislé proměnné, E[X] je střední hodnota.

$$cov(X,Y) = E[(X - E[X])(Y - E[Y])]$$

$$\sigma_{xy} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x}) \cdot (y_i - \bar{y})$$

Kovariance může nabývat hodnot na intervalu ⟨-∞; +∞⟩. Kovariance měří míru lineárního vztahu dvou proměnných.

Pokud je cov(x, y) > 0, pak jedna náhodná veličina roste, roste i druhá náhodná veličina. To naznačuje lineární závislost ve smyslu přímé úměry.

Pokud je cov(x, y) < 0, pak jedna náhodná veličina roste, druhá náhodná veličina klesá. To naznačuje lineární závislost ve smyslu nepřímé úměry.

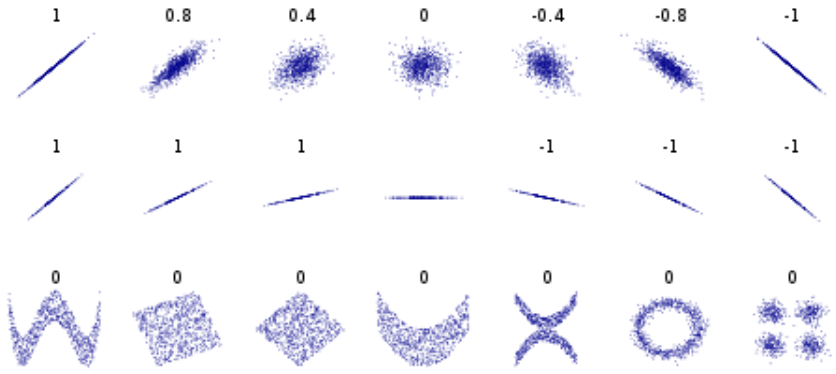
Pokud je cov(x, y) = 0, pak se kovariance blíží 0, pak mezi nezávislými proměnnými není lineární vztah. Pozor, to že kovariance se blíží 0, neznamená, že mezi náhodnými veličinami není žádný vztah.

3.2 Korelace

Korelace měří stupeň a směr lineárního vztahu mezi dvěma proměnnými. Normovaná hodnota kovariance je korelační koeficient. Korelace má standardizované měřítko, které není ovlivněno rozdíly v měřítkách porovnávaných proměnných. Obor hodnot korelace je <-1; +1>

$$\rho(X,Y) = \frac{cov\ (X,Y)}{\sqrt{D(X) \cdot D(Y)}}$$

Kovariance a korelace měří pouze lineární vztah mezi proměnnými. Pokud je mezi proměnnými jiný vztah, jejich hodnoty nejsou vypovídající.



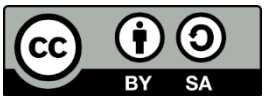
Obrázek 18 - Korelace pro různé náhodné veličiny

Při výběru proměnných do modelu je vhodným postupem vypočítat kovarianční matici, která rychle ukáže lineární vztahy mezi všemi proměnnými v datovém souboru.

Obrázek 19 – kovarinční matice

UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vytvořeno v rámci projektu **Digitalizace studijních Agend, Nové TechnologiE, systémy a přístupy k výuce na UPCE**, reg. č. NPO_UPCE_MSMT-16591/2022.



Financováno Evropskou unií
NextGenerationEU

Národní plán obnovy

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Machine learning & AI

Lineární regrese

Ing. Martin Pozdílek, Ph.D.

1 Lineární regrese

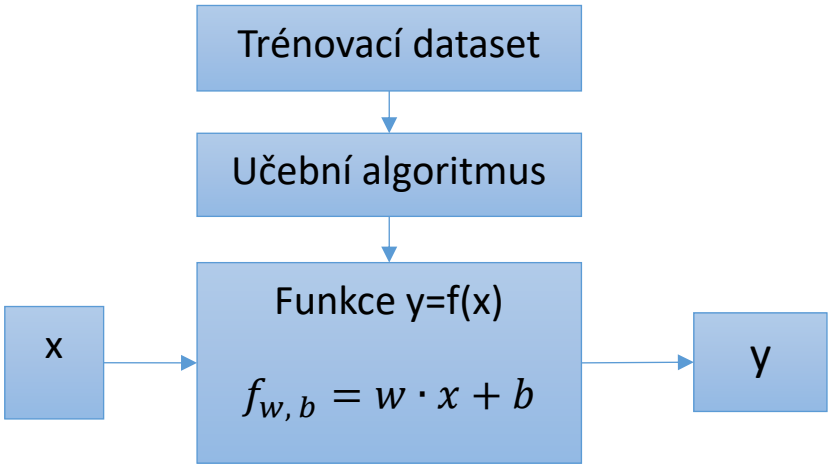
1.1 Modely

Živočichové se na základě vnímání světa snaží předvídat budoucnost a zvýšit si tak šanci na přežití. Kde bude kořist za 1 vteřinu? Kde nejspíše najdu obživu? Jak se nejrychleji dostanu k hnízdu?

Totéž dělá člověk. Jeho kognitivní schopnosti mu umožňují klást si složitější otázky. Kam dopadne ten kámen? Kdy bude táhnout zvěř? Jaké bude počasí? Jak dopadne následující bitva? Jak dopadne tento experiment?

Při snaze o pochopení světa si na základě pozorování okolních dějů vytváříme model, který se snaží postihnout všechny důležité aspekty děje. Na základě tohoto modelu děláme predikci do budoucna a následné rozhodování. Matematika a navazující přírodní vědy se ukazují jako dobrý nástroj pro vytváření a popis modelů. Díky popisu modelu jsme ho schopni racionálně verifikovat a sdílet s ostatními. Čím je model přesnější, tím získáváme větších výhodu.

Modely využívají algoritmy strojového učení. Během učení si stroj z dat popisujících okolí vytváří model a učí se podobně jako se učí lidi ze svých zkušeností.



Obrázek 1 - strojové učení

V této kapitole se zaměříme na lineární model, který dokáže na základě vstupních parametrů předpovídat výstupní parametr. Jaká je odhadovaná cena daného bytu v dané lokalitě? Jak dopadne zkouška z daného předmětu pro studenta?

1.2 Lineární regrese

Pojem regrese pochází z prací antropologa a meteorologa Francise Galtona, který mimo jiné zkoumal vztah výšky otců a výšky synů. Ve své práci zveřejnil závěry, že vysocí otcové mají vysoké syny,

ale průměrná výška těchto otců je vyšší než průměrná výška jejich synů. Totéž platí i pro malé otce a syny. Tento návrat k průměrné výšce populace Galton nazval regresí (krokem zpět).

Současný význam termínu regresní analýzy se poněkud odchýlil od významu, jak ho použil Galton. Ale způsob práce s empirickými daty a odvozováním modelů, který Galton použil, je stále platný a vžil se pro něj pojem regrese.

Při vytváření modelů pro umělou inteligenci velmi často vycházíme ze statistických metod. I v případě, že přímo nepoužíváme statistické metody, je vhodné o datových souborech znát základní informace a vyvarovat se tak zásadních chyb při vytváření modelů. Některé modely mají omezující podmínky a lze je použít pouze na určitý typ dat. Bez detailnějších znalostí o vstupních datech bude náš projekt velmi často odsouzen k neúspěchu.

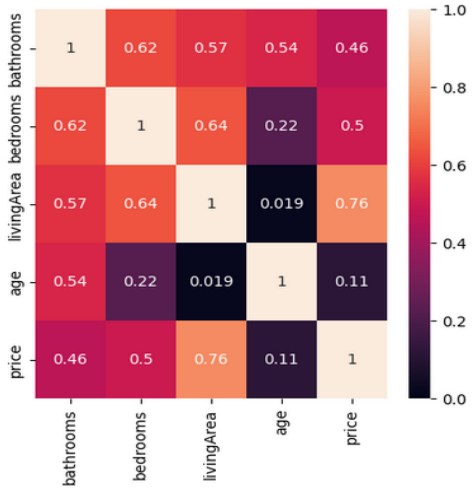
V této kapitole se budeme zabývat lineární regresí, která se používá pro předpovědi proměnných, které jsou v lineárním stavu – přímá a nepřímá úměrnost. Tento vztah lze vyjádřit přímkou. Lineární regrese je jednoduchý, ale výkonný a nejčastěji používaný algoritmus v datové vědě. Existuje nepřeberné množství aplikací lineární regrese v reálném světě. Na tomto algoritmu si ukážeme základní pojmy a principy, které se vyskytují v pokročilejších algoritmech.

Lineární regresi budeme ukazovat na datovém setu, který shromažďuje informace o cenách nemovitostí v Portlandu v červenci 2021 Portland Housing Prices/Sales Jul 2020 - Jul 2021. Z datového setu byla pro jednoduchost vybrána podmnožina 100 záznamů o 4 vysvětlujících proměnných a jedné vysvětlované proměnné. Náhled dat vypadá takto.

Bathrooms x	Bedrooms x	LivingArea (feet²) x	Age (years) x	Price ($) y
3	5	3470	7	1 165 000
4	5	3374	85	1 050 000
3	3	3265	20	442 500
3	3	3192	36	765 000
3	4	3157	73	815 000
4	5	2969	7	620 000

Obrázek 2 - náhled dat Portland Housing Prices

V přechozí kapitole jsme si ukázali postup, jak zjistit lineární vztahy mezi proměnnými pomocí korelace. Z níže uvedené tabulky vyplývá, že vysvětlovaná proměnná price (cena nemovitosti) má silný přímý lineární vztah s proměnnou livingArea (plocha nemovitosti). Určitou přímou korelaci mají i proměnné s počtem pokojů a koupelen - bathrooms a bedrooms. Proměnná se stářím nemovitosti nemá příliš velkou přímou korelaci na cenu nemovitosti.



Obrázek 3 - korelace proměnných v data setu

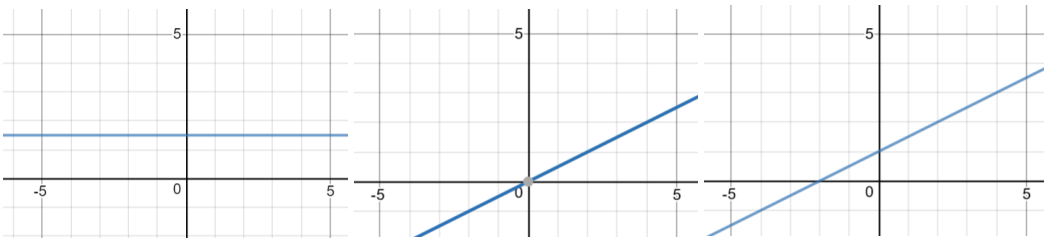
1.2.1 Lineární funkce

Cílem lineární regrese bude nalezení parametrů lineární funkce, která bude nejlépe predikovat cenu nemovitosti na základě plochy nemovitosti a případně na počtu pokojů a věku.

Rovnice přímky má následující podobu. V matematice se často používá parametr a, my pojmenujeme w jako weights (váha), abychom při vysvětlování umělých neuronových sítí měli stejnou terminologii. Podobně parametr b se nazývá intercept a představuje pevnou složku výstupní funkce, která není závislá na vstupní hodnotě.

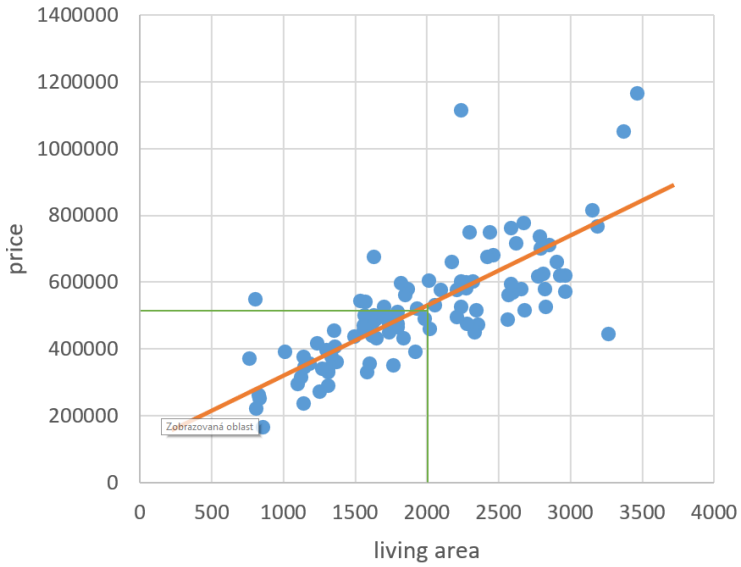
$$f_{w,b} = w \cdot x + b$$

Pro připomenutí si znázorníme vliv parametrů na průběh funkce.



Obrázek 4 - parametry lineární funkce

Úkolem lineární regrese je nalezení parametrů w a b, které nejlépe předpoví hodnotu nemovitosti. Výsledkem funkce bude predikovaná hodnota $y' = f_{w,b}(x)$. Z učební množiny známe správnou hodnotu y. Našim cílem je najít funkci, která bude vracet y', která se maximální blíží y. Graficky toto zadání může znázornit takto.

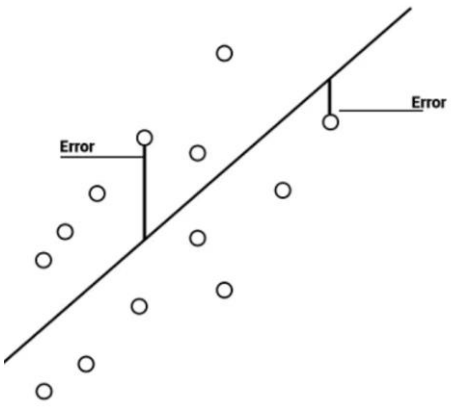


Obrázek 5 - lineární funkce předpovídající ceny nemovitostí

1.3 Nákladová funkce

Matematický model nebude predikovat hodnoty přesně podle skutečnosti, ale u každého odhadu se vybude vyskytovat chyba – rozdíl mezi skutečnou a predikovanou hodnotou. Pro porovnání úspěšnosti modelu se zavádí nákladová funkce, ale můžeme se setkat i s pojmy ztrátová nebo cílová funkce. Nákladová funkce vrací pro konkrétní parametry modelu w, b hodnotu, která se nazývá ztrátové skóre.

Při hledání správného modelu se snažíme vybrat takový, který má nejnižší ztrátové skóre.



Obrázek 6 - chyba je rozdíl mezi skutečnou a predikovanou hodnotou

Pro lineární regresi se často používají následující nákladové funkce. Později se při vytváření umělých neuronových sítí setkáme i s dalšími nákladovými funkcemi. Nákladovou funkci budeme označovat písmenem J.

1.3.1 Střední kvadratická chyba

Střední kvadratická chyba (mean squared error MSE nebo mean squared deviation MSD) se vypočítá jako suma druhých mocnin rozdílů predikovaných a skutečných hodnot dělená velikostí výběrového souboru. V praxi se násobí konstantou ½ , aby šly lépe počítat derivace.

$$J(w,\ b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^i) - y^i)^2$$

Někdy se používá RMSE – Root mean square error.

$$J(w,\ b) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^i) - y^i)^2}$$

1.3.2 Střední absolutní chyba

Někdy se používá střední absolutní chyba (Mean absolute error), která je lépe interpretovatelná.

$$J(w,\ b) = \frac{1}{2m}\sum_{i=1}^{m}\left|f_{w,b}(x^i) - y^i)\right|$$

1.4 Jednoduchá lineární regrese

V jednoduché lineární regresi existuje jedna nezávislé proměnná a jedna závislá proměnná. Pro začátek si to ještě zjednodušme a předpokládejme, že intercept v modelu bude 0. Rovnice modelu a rovnice nákladové funkce budou tedy následující.

$$f_w = w \cdot x$$

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,\,b}(x^i) - y^i)^2$$

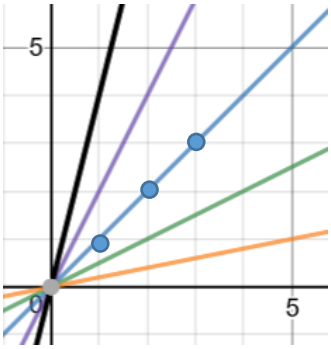
Zkusíme ručně najít optimální hodnotu parametru w pro následující data a zároveň se podíváme na ztrátové skóre.

X	Y
1	1
2	2
3	3

V níže uvedené tabulce jsou vypočítané hodnoty nákladové funkce pro různé hodnoty parametru w podle dat v horní tabulce.

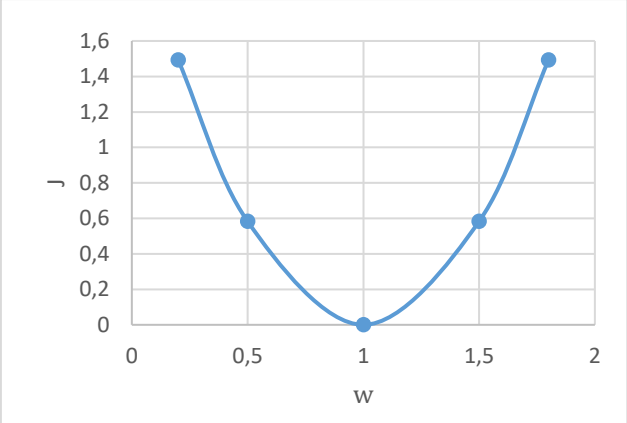
Parametr w	Hodnota nákladové funkce
1	$J(w) = \frac{1}{2 \cdot 3}((1-1)^2 + (2-2)^2 + (3-3)^2) = 0$
0,5	$J(w) = \frac{1}{2 \cdot 3}((0.5-1)^2 + (1-2)^2 + (1.5-3)^2) = 0.58$
0,2	$J(w) = \frac{1}{2 \cdot 3}((0.2-1)^2 + (0.4-2)^2 + (0.6-3)^2) = 1.49$
1,5	$J(w) = \frac{1}{2 \cdot 3}((1.5-1)^2 + (3-2)^2 + (4.5-3)^2) = 0.58$
1,8	$J(w) = \frac{1}{2 \cdot 3}((1.8-1)^2 + (3.6-2)^2 + (5.4-3)^2) = 1.49$

V grafu vidíme vynesené 3 učební body a přímky lineárního modelu pro různé hodnoty parametru w.



Obrázek 7 - přímky pro různé hodnoty w

Zajímavější je vynesení ztrátových skóre podle hodnoty parametru w. Vynesené body ztrátového skóre lze proložit parabolou. Z grafu je vidět, že optimální hodnota parametru w je 1.



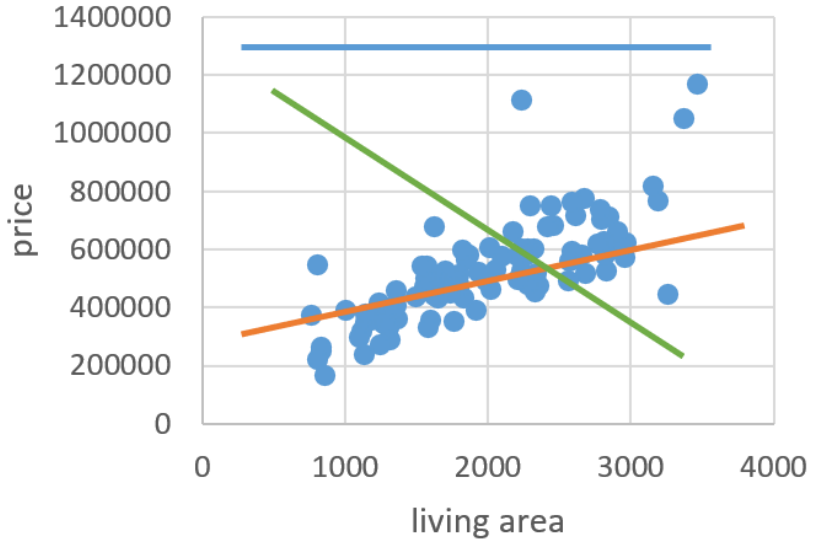
Obrázek 8 - graf nákladové funkce

Podobně zkusíme spočítat parametry modelu pro datový set Portland house a tentokrát již použijeme oba parametry modelu w a b. Stále se bude jednat o jednoduchou lineární regresi, kdy vstupní proměnnou je livingArea a výstupní proměnnou je price.

$$f_{w,b} = wx + b$$

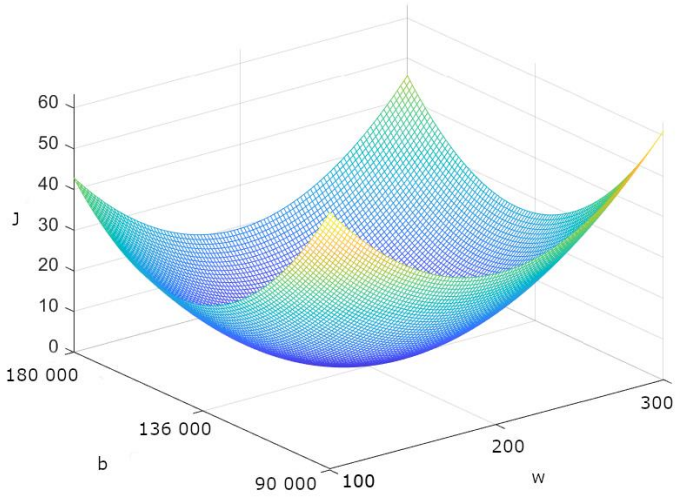
$$\mathrm{J}(w,\ b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,\,b}(x^i) - y^i)^2$$

Na grafu jsou znázorněny některé možné kombinace parametrů w a b. Těchto kombinací je nekonečně mnoho.



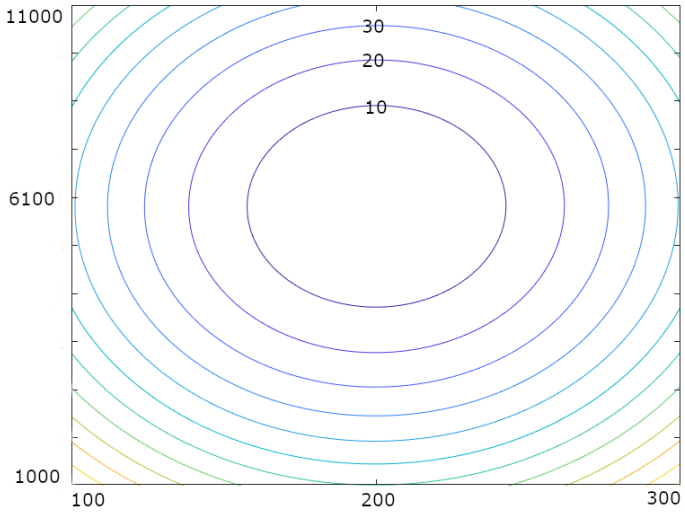
Obrázek 9 - modely pro různé hodnoty parametru w a b

Na tomto grafu je znázorněna nákladová funkce. Na osách x a y jsou parametry b a w. Na ose z je hodnota nákladové funkce. Optimální parametry modelu w a b se nachází v minimu funkce J.



Obrázek 10 - graf nákladové funkce

Zobrazování nákladové funkce v prostoru může být složité. Často se pro její zobrazení používá vrstevnicový graf. Elipsy spojují kombinace parametru w a b, pro která má nákladová funkce konstantní hodnotu. Pokud se budeme pohybovat po elipse, model se nebude zlepšovat. Pro zlepšení modelu je třeba přejít na elipsu s nižší hodnotou.



Obrázek 11 - vrstevnicový graf nákladové funkce

1.5 Gradient descent algoritmus

Způsobů, jak zjistit optimální parametry w a b, je více. Jedním z nich je například vyřešení přeurčené soustavy lineárních rovnic.

My se zaměříme na jiným způsob. Použijeme gradient descent algoritmus. Gradient descent algoritmus je optimalizační algoritmus, který krokově hledá lokální minimum diferenciovatelné funkce. Tento algoritmus je univerzální a lze jej použít i pro jiné modely, než je lineární model.

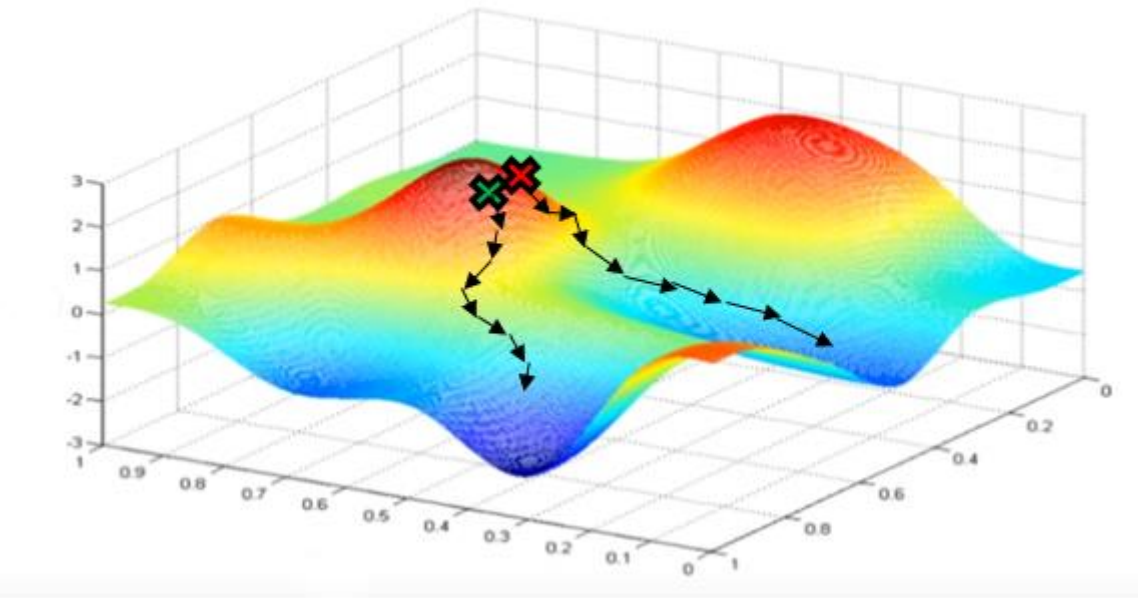
Algoritmus si můžeme představit na situaci, kdy nás vrtulník vysadí bez vybavení na neznámem místě v mlžných horách. Abychom se zachránili musíme se dostat do nížiny. Protože nemáme mapu a kvůli mlze nevidíme, nemůžeme si udělat představu o terénu a naplánovat si optimální cestu. Můžeme ale použít gradient descent algoritmus. Podíváme se po nejbližším okolí, tak jak nám dovolí mlha. Zjistíme, kterým směrem terén nejrychleji klesá dolů a uděláme krok. A vše zopakujeme, dokud se neocitneme na místě, ze kterého nejde vykročit níže.

V reálu by nám asi vadilo, kdybychom narazili na útes a udělali krok, který bude krátký v horizontálním směru, ale velmi dlouhý ve vertikálním směru. Na druhou stranu bychom se dostali rychle do nížiny.

Algoritmus má několik nevýhod. Pokud je terén hor komplikovaný s více různými údolími, velmi záleží na počátečním místě vysazení. I několik metrů vedle můžeme znamenat, že se dostaneme

do jiného údolí, které není nejhlubší ze všech. Pokud bychom chtěli najít nejhlubší údolí (optimální řešení), musíme algoritmus vícekrát opakovat s různými počátečními místy.

Algoritmus zároveň není odolný proti lokálním minimům. Na příkladu hor si můžeme představit terén, který obsahuje různé prohlubně položené vysoko v horách. Pokud se do takové to prohlubně dostaneme, algoritmus nám neumožní dostat se do nížiny, protože v prohlubni neexistuje směr, který by lokálně klesal.



Obrázek 12 - znázornění gradient descent algorimu

Nyní náš algoritmus ze slovního popisu převedeme do matematiky. Směr kroku vypočítáme pomocí parciálních derivací nákladové funkce podle parametrů w a b. $\frac{\partial}{\partial w}J(w,\ b)$ a $\frac{\partial}{\partial b}J(w,\ b)$ jsou parciální derivace, tedy tečny ke křivce. Kombinací dílčích směrů získáme výsledný směr kroku. Je důležité provést obě změny parametrů w a b v jedné iteraci.

Pro nákladovou funkci jsme použili následující definici.

$$\mathrm{J}(w,\ b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,\,b}(x^i) - y^i)^2$$

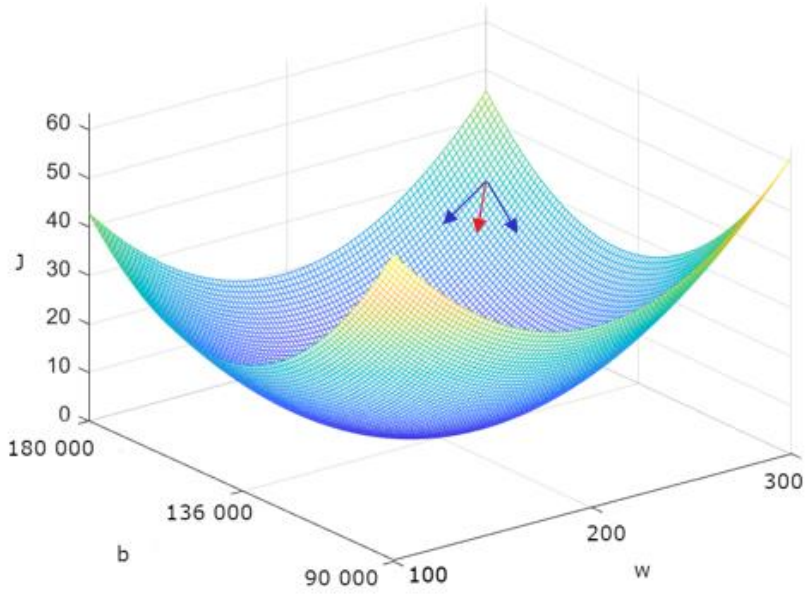
To nám umožní velmi efektivně vyjádřit parciální derivace.

$$\frac{\partial}{\partial w}J(w,\ b) = \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^i) - y^i)x^i$$

$$\frac{\partial}{\partial b}\ J(w,\ b) = \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^i) - y^i)$$

Velikost kroku, a tedy rychlost učení, určuje α – learning rate. Při vyšší hodnotě α se algoritmus rychleji přiblíží k optimálnímu řešení. Ovšem při příliš vysokém α nemusí algoritmus fungovat správně, jak si ukážeme dále.

Na následujícím obrázku je ukázán graf nákladová funkce pro lineární regresy. V grafu je naznačen jeden krok, kde se současně mění hodnota parametru w a b.



Obrázek 13 - graf nákladové funkce s vyznačeným krokem

Algoritmus lze tedy zapsat následovně:

- Opakuj, dokud nedojde ke konvergenci
  - $\text{step\_0} \coloneqq w - \alpha \frac{\partial}{\partial w} J(w,\ b)$
  - $\text{step\_1} \coloneqq b - \alpha \frac{\partial}{\partial b} J(w,\ b)$
  - $w \coloneqq \text{step\_0}$
  - $b \coloneqq \text{step\_1}$

1.5.1 Velikost a směr kroku

Pro názornost opět předpokládejme lineární model, který bude používat pouze w. Jinými slovy parametr b bude roven 0. Pak bude mít nákladová funkce podobu paraboly.

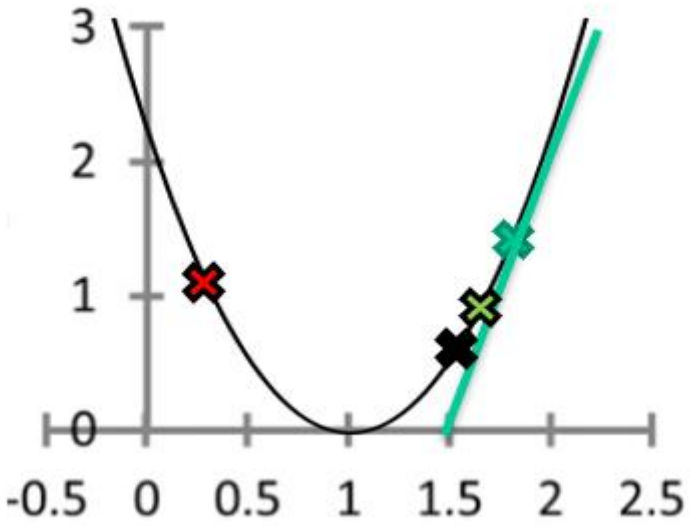
Velikost a směr kroku spočítáme pomocí derivace funkce J v bodě. Předpokládejme, že parametr w má hodnotu přibližně hodnotu 1,7 a na nákladové křivce se nacházíme v bodě . Derivace $\partial J(w)$ pro tento bod je kladná a obrázku je znázorněna zelenou přímkou. Velikost kroku bude záporná podle $-\alpha \cdot \partial\ J(w)$.

Pro hodnotu parametru w rovnou přibližně 0,3 se budeme nacházet v bodě ✖

Derivace bude záporná a krok změny parametru w bude kladný.

Je tedy vidět, že parametr w se nám v obou případech bude postupně blížit k optimální hodnotě 1. V lokálním minimu bude hodnota derivace 0 a tím pádem velikost kroku bude 0.

S přibližováním se k vrcholu paraboly se derivace / sklon snižuje, díky tomu se velikost kroku snižuje a není třeba nijak měnit rychlost učení α.



Obrázek 14 - derivace pro různé body nákladové křivky

Pokud budeme uvažovat oba parametry w a b, tak směr a velikost kroku budeme počítat přes parciální derivace.

1.5.2 Lokální minimum

Nevýhodou gradient descent algoritmu je, že nehledá globální, ale lokální minimum. Pokud bude tvar nákladové křivky složitější, než je tomu u lineární funkce, nemusí algoritmus najít optimální řešení.

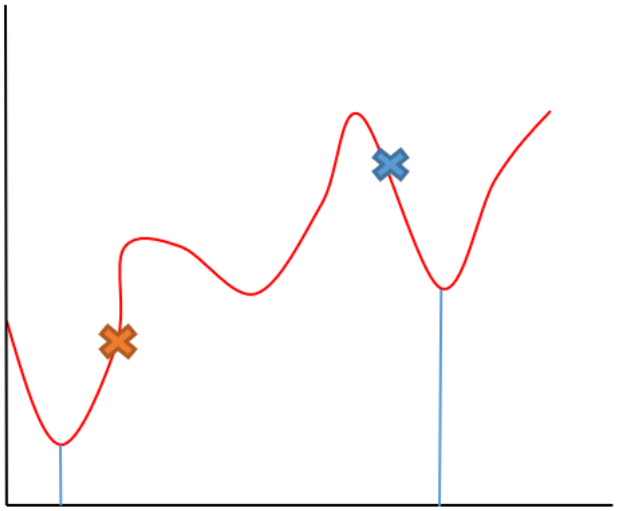
Inicializace parametrů w a b může mít významný vliv na nalezení řešení.

Tuto nevýhodu lze zmenšit vícenásobným spuštěním algoritmu s různými inicializačními body. Pořád není zaručené, že se najde globální minimum. Zároveň spotřebujeme více výpočetních zdrojů.

Druhým způsobem je použití vylepšených verzí gradient descent algoritmů (Gradient descent with momentum) nebo i jiných optimalizačních algoritmů (AdaGrad, RMSProp, …)

Gradient descent algoritmus s momentem při rychlosti změny parametrů uvažuje rychlosti předchozí změny. Do pohybu přidává moment, který pomáhá překonat lokální minimum. Změna parametrů se počítá následovně:

$$w = w - \alpha \cdot \frac{\delta}{\delta_w} J(w) + momentum \cdot velocity$$



Obrázek 15 - lokální a globální minimum

1.5.3 Lineární algebra

Při výpočtu lineární regrese se velmi často pracuje s maticemi. Místo postupného počítání změny kroku pro samostatná pozorování v trénovacím datasetu se používá výpočet pomocí matic. To umožňuje paralelní výpočet pomocí více CPU jader nebo dokonce použití GPU.

Z datového setu si vytvoříme dva vektory x a y pro vysvětlovanou a vysvětlují proměnnou. Parametr w bude mít také podobu vektoru a jedním řádkem. Parametr b bude číslo.

LivingArea (feet²) x	Price ($) y
3470	1 165 000
3374	1 050 000
3265	442 500
3192	765 000
3157	815 000
2969	620 000

$$X = \begin{bmatrix} 3470 \\ 3374 \\ 3265 \\ 3192 \\ 3157 \\ 2969 \end{bmatrix}, y = \begin{bmatrix} 1\,165\,000 \\ 1\,050\,000 \\ 442\,500 \\ 765\,000 \\ 815\,000 \\ 620\,000 \end{bmatrix}, \quad w = [195] \text{ , b = 136 281}$$

Lineární model má tedy následující rovnici s využitím maticových operátorů.

$$f_{w,b}(x) = w \times \mathrm{x} + b$$

Jediným maticovým výpočtem získáme všechny predikované hodnoty pro trénovací dataset.

$$y' = f_{w,b}(x) = \begin{pmatrix} 3470 \\ 3374 \\ 3265 \\ 3192 \\ 3157 \\ 2969 \end{pmatrix} \times [195] + \begin{pmatrix} 136\,281 \\ 136\,281 \\ 136\,281 \\ 136\,281 \\ 136\,281 \\ 136\,281 \end{pmatrix} = \begin{pmatrix} 812\,931 \\ 794\,211 \\ 772\,956 \\ 758\,721 \\ 751\,896 \\ 715\,236 \end{pmatrix}$$

Z něj pak můžeme vypočítat parciální derivace pro výpočet kroku.

$$\frac{\partial}{\partial b} J(w,\ b) = \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^i) - y^i)$$

$$\frac{\partial}{\partial b} J(w,\ b) = \frac{\left(\begin{pmatrix} 1\,165\,000 \\ 1\,050\,000 \\ 442\,500 \\ 765\,000 \\ 815\,000 \\ 620\,000 \end{pmatrix} - \begin{pmatrix} 812\,931 \\ 794\,211 \\ 772\,956 \\ 758\,721 \\ 751\,896 \\ 715\,236 \end{pmatrix} \right)}{6} = \frac{9\,251\,549}{6} = 41\,924{,}83$$

$$\frac{\partial}{\partial w} J(w,\ b) = \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^i) - y^i) x^i$$

$$\frac{\partial}{\partial w}J(w,\ b) = \frac{\left(\begin{pmatrix}1\ 165\ 000\\ 1\ 050\ 000\\ 442\ 500\\ 765\ 000\\ 815\ 000\\ 620\ 000\end{pmatrix} - \begin{pmatrix}812\ 931\\ 794\ 211\\ 772\ 956\\ 758\ 721\\ 751\ 896\\ 715\ 236\end{pmatrix}\right) \times \begin{pmatrix}3470\\ 3374\\ 3265\\ 3265\\ 3192\\ 3157\\ 2969\end{pmatrix}}{6} = \frac{94\ 227\ 8888}{6} = 157\ 046\ 481{,}3$$

1.6 Lineární regrese s více proměnnými

Lineární regrese se neomezuje pouze na jednu vysvětlující a jednu vysvětlovanou proměnnou. Vysvětlujících proměnných může být více.

Opět si to ukážeme na datovém setu Portland houses. Tentokrát do modelu zahrneme proměnné bathrooms, bedrooms, livingArea a age. Proměnné budeme označovat x_1 až x_4.

Bathrooms x_1	Bedrooms x_2	LivingArea (feet²) x_3	Age (years) x_4	Price ($) y
3	5	3470	7	1 165 000
4	5	3374	85	1 050 000
3	3	3265	20	442 500
3	3	3192	36	765 000
3	4	3157	73	815 000
4	5	2969	7	620 000

Rozepsaná lineární funkce má tuto podobu:

$$f_{w,b}(x) = \mathrm{w}_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$

Při použití lineární algebry má podobu stejnou jako v případě jednoduché lineární algebry.

$$f_{w,b}(x) = x \times w + b$$

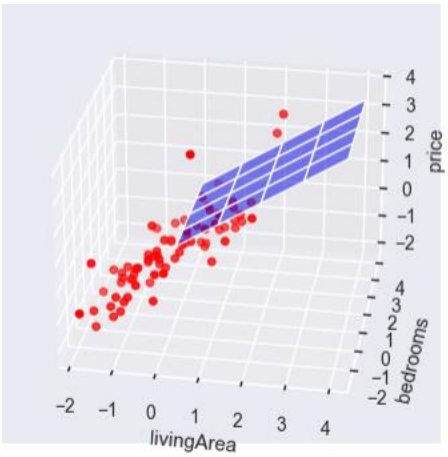
Vysvětlující proměnné mají podobu matice X, skutečné hodnoty mají podobu vektoru y. Parametry modelu se budou skládat z vektoru w a skaláry b.

$$X = \begin{pmatrix} 3 & 5 & 3470 & 7 \\ 4 & 5 & 3374 & 85 \\ 3 & 3 & 3265 & 20 \\ 3 & 3 & 3192 & 36 \\ 3 & 4 & 3157 & 73 \\ 4 & 5 & 2969 & 7 \end{pmatrix}, y = \begin{pmatrix} 1\,165\,000 \\ 1\,050\,000 \\ 442\,500 \\ 765\,000 \\ 815\,000 \\ 620\,000 \end{pmatrix}, \quad w = \begin{pmatrix} 1\,249.88 \\ 20\,553.12 \\ 173.11 \\ 257.84 \end{pmatrix}, \; b = 84\,656$$

Výpočet predikovaných hodnotu bude vypadat následovně:

$$y' = f_{w,b}(x) = \begin{pmatrix} 3 & 5 & 3470 & 7 \\ 4 & 5 & 3374 & 85 \\ 3 & 3 & 3265 & 20 \\ 3 & 3 & 3192 & 36 \\ 3 & 4 & 3157 & 73 \\ 4 & 5 & 2969 & 7 \end{pmatrix} \times \begin{pmatrix} 1\,249.88 \\ 20\,553.12 \\ 173.11 \\ 257.84 \end{pmatrix} + \begin{pmatrix} 84\,656 \\ 84\,656 \\ 84\,656 \\ 84\,656 \\ 84\,656 \\ 84\,656 \end{pmatrix} = \begin{pmatrix} 793\,668 \\ 798\,411 \\ 720\,426 \\ 711\,914 \\ 735\,949 \\ 708\,190 \end{pmatrix}$$

Vizualizace modelu s jedním parametrem w měla podobu přímky. V případě dvou proměnných X a dvou parametrů w má podobu roviny v prostoru. Pro vyšší počet dimenzí nelze vytvořit vhodný graf.



Obrázek 16 - Lineární model se dvěma proměnnými

Pokud budeme počítat parciální derivace pro výše uvedené hodnoty X a parametry w a b, tak výpočet bude vypadat následovně.

$$\frac{\partial}{\partial w} J(w,\, b) = \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}\left(x^i\right) - y^i \right)$$

$$\frac{\partial}{\partial w} J(w,\, b) = \frac{\left(\begin{pmatrix} 1\,165\,000 \\ 1\,050\,000 \\ 442\,500 \\ 765\,000 \\ 815\,000 \\ 620\,000 \end{pmatrix} - \begin{pmatrix} 793668 \\ 798411 \\ 720426 \\ 711914 \\ 735948 \\ 708190 \end{pmatrix} \right)}{6} = \frac{388\,942}{6} = 64\,823{,}67$$

$$\frac{\partial}{\partial w_1} J(w,\ b) = \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^i) - y^i)\ x_1^i$$

$$\frac{\partial}{\partial w_1} J(w,\ b) = \frac{\left(\begin{pmatrix}1\,165\,000\\1\,050\,000\\442\,500\\765\,000\\815\,000\\620\,000\end{pmatrix} - \begin{pmatrix}793668\\798411\\720426\\711914\\735948\\708190\end{pmatrix}\right) \times \begin{pmatrix}3\\4\\3\\3\\3\\4\end{pmatrix}}{6} = 221\,704{,}17$$

$$\frac{\partial}{\partial w_2} J(w,\ b) = \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^i) - y^i)\ x_2^i$$

$$\frac{\partial}{\partial w_2} J(w,\ b) = \frac{\left(\begin{pmatrix}1\,165\,000\\1\,050\,000\\442\,500\\765\,000\\815\,000\\620\,000\end{pmatrix} - \begin{pmatrix}793668\\798411\\720426\\711914\\735948\\708190\end{pmatrix}\right) \times \begin{pmatrix}5\\5\\3\\3\\4\\5\end{pmatrix}}{6} = 385\,889{,}83$$

$$\frac{\partial}{\partial w_3} J(w,\ b) = \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^i) - y^i)\ x_3^i$$

$$\frac{\partial}{\partial w_3} J(w,\ b) = \frac{\left(\begin{pmatrix}1\,165\,000\\1\,050\,000\\442\,500\\765\,000\\815\,000\\620\,000\end{pmatrix} - \begin{pmatrix}793668\\798411\\720426\\711914\\735948\\708190\end{pmatrix}\right) \times \begin{pmatrix}3470\\3374\\3265\\3192\\3157\\2969\end{pmatrix}}{6} = 231\,188\,890{,}8$$

$$\frac{\partial}{\partial w_4} J(w,\ b) = \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^i) - y^i)\ x_4^i$$

$$\frac{\partial}{\partial w_4} J(w,\ b) = \frac{\left(\begin{pmatrix}1\,165\,000\\1\,050\,000\\442\,500\\765\,000\\815\,000\\620\,000\end{pmatrix} - \begin{pmatrix}793668\\798411\\720426\\711914\\735948\\708190\end{pmatrix}\right) \times \begin{pmatrix}7\\85\\20\\36\\73\\7\end{pmatrix}}{6} = 4\,248\,393$$

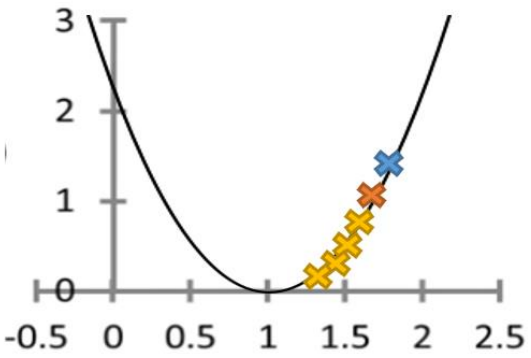
Podle velikosti by se vypočítaly změny jednotlivých parametrů w.

1.7 Velikost míry učení α

Důležitým parametrem v procesu učení je míra učení α (learning rate). Jedná se o konstantu, která ovlivňuje velikost kroku mezi iteracemi. Správná volba míry učení ovlivní jeho průběh.

Pokud nastavíme α příliš malé, algoritmus minimum najde, ale bude to trvat mnoho kroků a budeme plýtvat výpočetními zdroji. Je dobré si uvědomit, že s přibližováním se k optimu klesá směrnice tečny a tím se i automaticky snižuje velikost kroku.

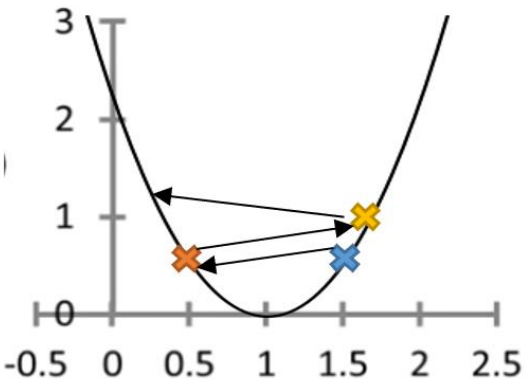
Toto je naznačeno na následujícím obrázku, kdy z modrého inicializačního bodu se v iteraci stále pomaleji posouváme k vrcholu elipsy.



Obrázek 17 - proces učení při malém α

Větším problémem může být nastavení příliš vysoké hodnoty α ve snaze zrychlit učební proces. V některých situacích příliš vysoké α povede k velkému kroku, který přeskočí hledané optimum.

V extrémních případech algoritmus nemusí dokonvergovat k minimu, protože kroky jsou příliš veliké a skáčou z jedné strany křivky na druhou.



Obrázek 18 - proces učení při vysokém α

1.8 Křivka učení

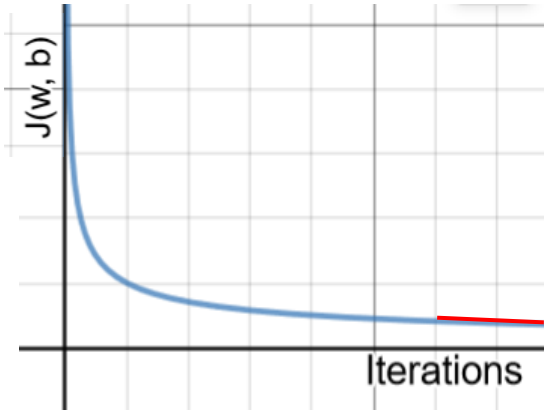
Optimalizační algoritmus může trvat velmi dlouho. U lineární regrese se to zpravidla nenastává, ale až budeme cvičit neuronové sítě, tak se proces učení může pohybovat v hodinách nebo i ve dnech.

Proto je vhodné proces učení monitorovat a v případě nepříznivých parametrů třeba i pozastavit.

Proces učení se monitoruje průběžným zjišťováním nákladové funkce J(w,b). Její hodnotu sledujeme každých n iterací a můžeme ji vynášet do grafu, tím získáme křivku učení.

Při správně nastaveném učení se musí hodnota nákladové funkce postupně snižovat. Graf ideální učební křivky je následující. V prvních iteracích nákladová funkce velmi rychle klesá. S rostoucím počtem iterací, kdy se na nákladové křivce blížíme k bodu optima, klesá směrnice její tečny a tím se zmenšuje velikost kroku. Se snižováním velikosti kroku se snižuje pokles hodnoty nákladové funkce.

Po n iteracích se pokles nákladové funkce zastaví nebo téměř zastaví, což je znamení, že jsme našli optimální nebo téměř optimální řešení. Toho můžeme například využít pro automatické zastavení algoritmu. Například pokud pokles hodnoty nákladové funkce $J(w,b) < 10^{-3}$, algoritmus zastavíme.

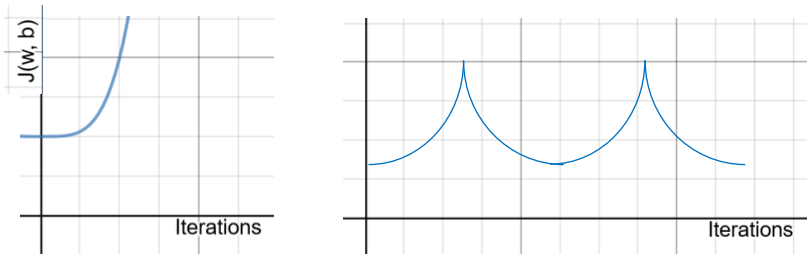


Obrázek 19 - správný tvar učení křivky

Pro příliš malé α (míra učení) bude křivka stále klesající, jen bude klesat pomalu.

Příliš velké α lze také zjistit z učební křivky. V tomto případě bude učební křivka rostoucí nebo kolísající. V tomto případě je vhodné algoritmus zastavit a snížit hodnotu α.

Obecně se hodnota α volí velmi nízká a postupně se s dalšími běhy zvyšuje 0.001; 0.01; 0.3; 1 …



Obrázek 20 - špatné tvary křivky

1.9 Normalizace dat

Pokud budeme zpracovávat dataset Portland houses pomocí lineární regrese, může nastat situace, kdy gradient descent algoritmus selže.

Důvodem může být zcela odlišný rozsah vysvětlujících proměnných. V našem případě proměnná bedrooms a bathrooms má rozsah 3 – 4, proměnná livingArea 2969 – 3470 a proměnný price 620 000 – 1 165 000.

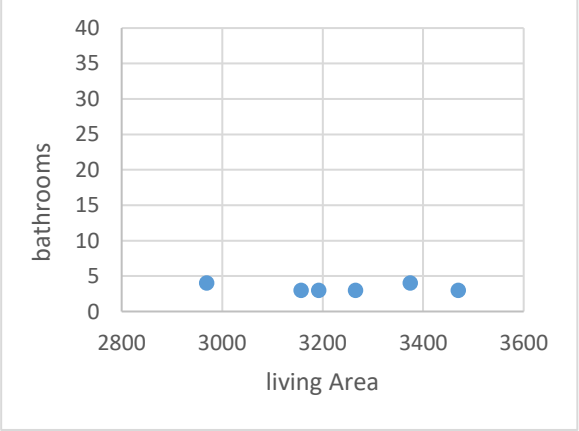
Výsledné ceny lze dosáhnout různou kombinací w_1, w_2, b

- $f_{w,b}(x) = w_1 x_1 + w_2 x_2 + b$
- $1\,165\,000 = 185\,000 \cdot 3 + 173 \cdot 3470 + 9690$
- $1\,165\,000 = 12 \cdot 3 + 333 \cdot 3470 + 9454$

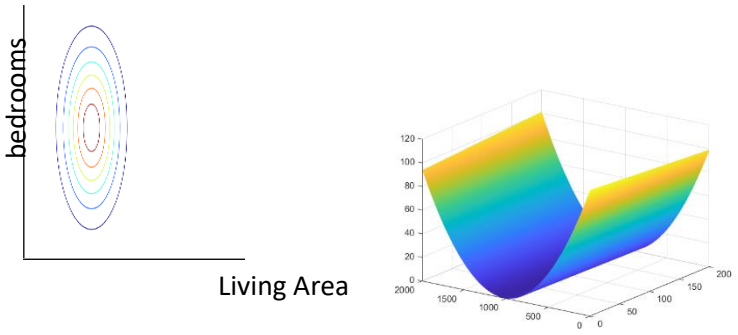
Vysoké x_n s velkou vahou w_n bude tvořit významnou část $f_{w,b}(x)$. Naopak nízké x_n s nízkou vahou w_n bude tvořit malou část $f_{w,b}(x)$ a proměnná bude mít malý vliv na model. Zároveň proces učení může být pomalý a nestabilní.

Tuto situaci může rozpoznat díky různým příznakům.

1. Body prvků v grafu tvoří téměř přímku a nelze rozlišit mezi nimi rozdíly.



2. Obrysový graf nákladové funkce J má tvar úzké elipsy



3. Rozsah proměnných není mezi -3 a 3, případně se pohybuje pouze v rozmezích -0,001, 0,001

Řešením je škálování, normalizace dat (feature scaling), kdy vytvoříme nové proměnné, které budou mít střední hodnotu okolo 0, hodnoty se budou pohybovat v rozmezí maximálně -3 až 3.

Pozor, pokud vytvoříme model na základě nových normovaných proměnných, nesmíme při jeho provozu zapomenout před vložím nové hodnoty provést normalizaci a po vrácení hodnoty provést denormalizaci výstupní proměnné.

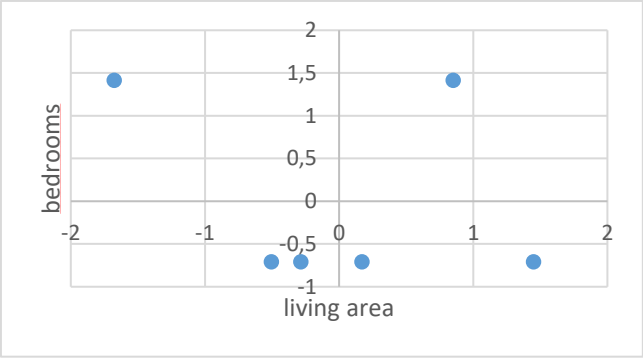
1.9.1 Z-Score normalizace

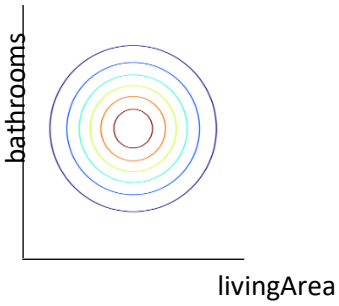
Velmi často se používá z-score normalizace, kdy novou proměnnou vytvoříme následujícím vzorem, kde μ_i je střední hodnota proměnné x a s_i je směrodatná odchylka proměnné x.

$$x_i' = \frac{x_i - \mu_i}{s_i}$$

Střední hodnota nové proměnné bude 0 a hodnoty budou kladné i záporné.

Graf nové proměnné bude lépe ukazovat rozdíly. Obrysový graf nákladové funkce J bude mít podobu soustředných kružnic.





Obrázek 21 - nákladová funkce normalizované proměnné

1.9.2 MinMax normalizace

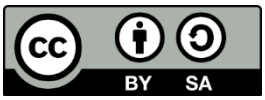
Jinou normalizací je MinMax, kdy se nová proměnná spočítá pomocí minima a maxima původní proměnné.

$$x_i' = \frac{x_i - \min(x_i)}{\max(x_i) - min(x_i)}$$

V tomto případě hodnoty nové proměnné budou v rozsahu 0 až +1.

UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vytvořeno v rámci projektu **Digitalizace studijních Agend, Nové TechnologiE, systémy a přístupy k výuce na UPCE**, reg. č. NPO_UPCE_MSMT-16591/2022.



Financováno Evropskou unií
NextGenerationEU

Národní plán obnovy

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Machine learning & AI

Analýza hlavních komponent

Ing. Martin Pozdílek, Ph.D.

1 Redukce dimenzionality

Velké datové soubory velmi často obsahují velký počet proměnných, které mohou být vzájemně korelovány. Velké datové soubory bývají řídké, tedy u jednotlivých pozorování nejsou uvedeny všechny vysvětlující proměnné a místo nich jsou uloženy NULL, N/A. Zároveň vytvoření vhodného modelu pro tyto velké soubory dat je komplikované, výpočetně náročné až neřešitelné. Proto je častým postupem redukce dimenzionality, tedy vynechání těch dat, která mají na přesnost modelu minimální vliv. Jednodušší model lze lépe pochopit a vysvětlit, bude rychlejší a jednodušší na implementaci.

Redukce dimenzionality neboli redukce dimenze je transformace dat z vysokodimenzionálního prostoru do nízkorozměrného prostoru tak, aby si nízkorozměrná reprezentace zachovala některé smysluplné vlastnosti původních dat, v ideálním případě blízké jejich vlastní dimenzi.

Redukce může být provedena omezením počtu pozorování, kdy z velkého souboru je vytvořen reprezentativní vzorek. Druhým způsobem je snížení počtu proměnných. Pod proměnnými si můžeme představit nejen různé metriky, ale například i rozlišení vizuálních dat, frekvence zvukových stop atd.

Obecně redukci dimenzionality lze použít pro redukci šumu, vizualizaci dat, shlukovou analýzu nebo jako mezikrok pro usnadnění dalších analýz.

Metody se běžně dělí na lineární a nelineární přístupy. Přístupy lze také rozdělit na výběr příznaků (feature selection) a extrakci příznaků (feature extraction). Výběr příznaků jsme si vyzkoušeli v předchozí kapitole, kdy jsem pro vytváření lineárního modelu používali pouze vysvětlující proměnné, které měly vysokou lineární korelaci s vysvětlovanou proměnnou. V této kapitole se zaměříme na extrakci příznaků pomocí metody hlavních komponent.

2 Analýza hlavních komponent

Analýza hlavních komponent (Principal Component Analysis – PCA) je často používanou metodou ve strojovém učení pro snížení dimenze dat nebo pro vyhledání vztahů v datech. Výsledkem metody hlavních komponent je kvantifikace „důležitosti“ proměnných ve skupině vysvětlovaných a skupině vysvětlujících proměnných.

Z velkého datového souboru lze transformací dat vytvořit menší soubor, který vypuštěním „šumu“ stále obsahuje většinu informací z velkého souboru. Snížení počtu proměnných v datovém souboru jde přirozeně na úkor přesnosti, ale trik při snižování dimenzionality spočívá v tom, že se vymění trocha přesnosti za jednoduchost modelu. Menší datové sady se totiž snáze zkoumají a vizualizují a algoritmy strojového učení díky nim mohou mnohem snadněji a rychleji analyzovat reálná produkční data.

Shrneme-li to tedy, myšlenka PCA je jednoduchá. PCA se snaží snížit počet vysvětlujících proměnných datového souboru, a přitom zachovat co nejvíce informací.

Analýzu hlavních komponent lze rozdělit do pěti kroků:

1. Standardizace rozsahu spojitých výchozích proměnných
2. Výpočet kovarianční matice pro identifikaci korelací
3. Výpočet vlastních vektorů a vlastních čísel kovarianční matice pro identifikaci hlavní komponenty
4. Vytvoření vektoru příznaků pro rozhodnutí, které hlavní komponenty zachovat
5. Přepočítání dat podél os hlavních komponent

2.1 Standardizace dat

Cílem tohoto kroku je standardizovat rozsah spojitých výchozích proměnných tak, aby každá z nich přispívala k analýze stejnou měrou. Důvodem, proč je rozhodující provést standardizaci před PCA, je konkrétně to, že PCA je poměrně citlivá, pokud jde o rozptyly výchozích proměnných. To znamená, že pokud existují velké rozdíly mezi rozsahy výchozích proměnných, budou proměnné s většími rozsahy mít vyšší prioritu nad proměnnými s malými rozsahy. Například proměnná s rozsahem 0 až 100 bude převažovat nad proměnnou s rozsahem 0 až 1. To povede ke zkreslení výsledků. Tomuto problému lze tedy předejít transformací dat na srovnatelné škály.

Matematicky to lze provést odečtením průměru a vydělením směrodatnou odchylkou pro každou hodnotu každé proměnné. Tento postup byl již naznačen v minulé hodině v kapitole 1.9.1.

Novou standardizovanou proměnnou získáme odečtením střední hodnoty a vydělením směrodatné odchylky.

$$x'_i = \frac{x_i - \mu_i}{s_i}$$

Po provedení standardizace se všechny proměnné transformují na stejnou stupnici.

2.2 Výpočet kovarianční matice

Cílem tohoto kroku je zjistit, jak se proměnné vstupního souboru dat navzájem liší od průměru, nebo jinými slovy, zda mezi nimi existuje nějaký vztah. Někdy jsou totiž proměnné vysoce korelované takovým způsobem, že obsahují nadbytečné informace. Abychom tedy tyto korelace identifikovali, vypočítáme kovarianční matici. Výpočet kovariance je vysvětlen v kapitole 3.2 v bloku statistika.

Kovarianční matice p × p je symetrická matice (kde p je počet dimenzí), která má jako položky kovariance spojené se všemi možnými dvojicemi výchozích proměnných. Například pro třírozměrný soubor dat se třemi proměnnými x, y a z je kovarianční matice 3 × 3 datová matice tohoto z:

$$\begin{matrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{matrix}$$

Protože kovariance proměnné se sebou samou je její rozptyl $cov\ (a,a) = DX\ (a)$, v hlavní diagonále máme vlastně rozptyly jednotlivých výchozích proměnných. A protože kovariance je komutativní $cov\ (a,b) = cov(b,a)$, jsou položky kovarianční matice symetrické vzhledem k hlavní diagonále, což znamená, že horní a dolní trojúhelníková část jsou si rovny.

To si můžeme ukázat na příkladu kovarianční matice datasetu Boston housing prices.

Obrázek 1 - kovarianční matice

Ve výše uvedené kovarianční matici jsou vidět nejen lineární závislosti vysvětlujících proměnných na vysvětlované proměnné, ale i lineární vztahy mezi vysvětlujícími proměnnými. Tento vztah se nazývá multikolinearita . Například DIS je vysoce korelován s INUDS, INOX a AGE.

To není dobré, protože multikolinearita může náš model učinit nestabilním. Odhad vlivu jedné proměnné na závislou proměnnou Y při kontrole ostatních proměnných bývá méně přesný, než kdyby prediktory nebyly vzájemně korelovány.

Obvyklá interpretace regresního koeficientu je taková, že poskytuje odhad účinku jednotkové změny nezávislé proměnné při zachování ostatních proměnných jako konstantních. I tento problém nám umožní odstranit PCA.

Multikolinearitu může detekovat pomocí faktoru inflace rozptylu (Variance Inflation Factor, VIF). VIF odhaduje, jak moc je rozptyl regresního koeficientu nadsazen v důsledku multikolinearity v modelu.

$$VIF = \frac{1}{(1 - R^2)}$$

Kde R^2 je koeficient determinace vypočítaný následujícím vzorcem. Kde $\bar{y}$ je střední hodnota y. Zjednodušeně řečeno, je to podíl rozptylu nezávislé proměnné, který je vysvětlen závislou proměnnou.

$$R^2 = \frac{\sum_i (f_{w,b}(x^i) - y^i)^2}{\sum_i (y^i - \bar{y})^2}$$

Provedeme tedy lineární regresi s použitím každé proměnné jako cíle a ostatních jako vysvětlujících proměnných a vypočítáme R^2 a poté pro ně vypočítáme VIF.

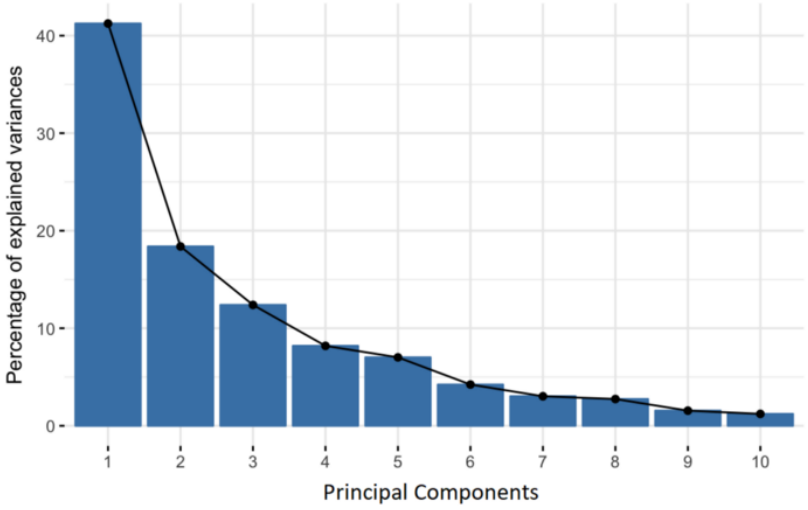
Pokud je VIF < 4, lze ji použít, v opačném případě musíme najít způsob, jak odstranit kolinearitu těchto funkcí, například pomocí PCA.

2.3 Výpočet vlastních vektorů a vlastních čísel kovarianční matice

Vlastní vektory a vlastní čísla jsou pojmy lineární algebry, které musíme vypočítat z kovarianční matice, abychom určili hlavní komponenty dat. Než se dostaneme k vysvětlení těchto pojmů, nejprve si ujasněme, co rozumíme pod pojmem hlavní komponenty.

Hlavní komponenty jsou nové proměnné, které jsou konstruovány jako lineární kombinace výchozích proměnných. Tyto kombinace se provádějí tak, aby nové proměnné (tj. hlavní komponenty) nebyly korelované a většina informací v rámci výchozích proměnných byla vtěsnána do prvních komponent.

Jde tedy o to, že desetirozměrná data reprezentuje deset hlavních komponent, ale metoda PCA se snaží umístit maximum možné informace do první komponenty, pak maximum zbývající informace do druhé a tak dále, až vznikne něco podobného tomu, co je znázorněno na grafu níže.



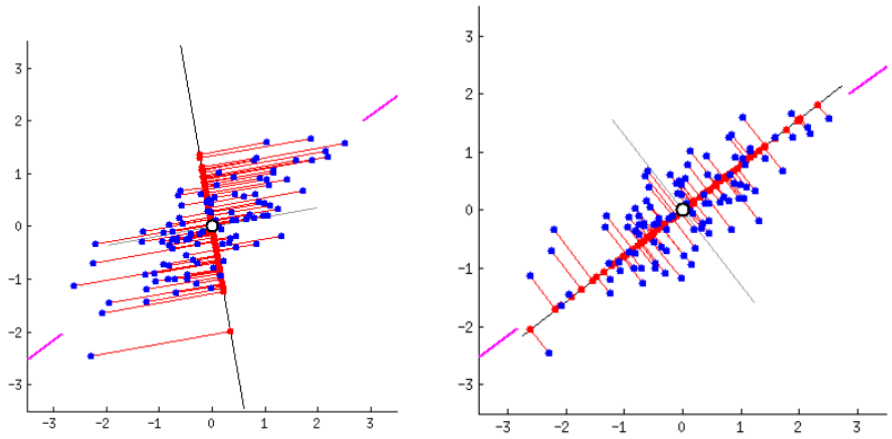
Obrázek 2 – Vztah pořadí PCA a procento vysvětlované odchylky

Uspořádání informací do hlavních komponent tímto způsobem vám umožní snížit dimenzionalitu, aniž bychom ztratili mnoho informací a to tak, že vyřadíte komponenty s nízkou informací a zbývající komponenty budete považovat za nové proměnné.

Důležité je uvědomit si, že hlavní komponenty jsou méně interpretovatelné a nemají žádný skutečný význam, protože jsou konstruovány jako lineární kombinace výchozích proměnných.

Z geometrického hlediska představují hlavní komponenty ty směry dat, které vysvětlují maximální množství rozptylu, tj. linie, které zachycují nejvíce informací o datech. Vztah mezi rozptylem a informací zde spočívá v tom, že čím větší rozptyl nese přímka, tím větší je rozptyl datových bodů podél ní, a čím větší je rozptyl podél přímky, tím více informace má. Zjednodušeně řečeno, stačí si hlavní komponenty představit jako nové osy, které poskytují nejlepší úhel pohledu na data a jejich vyhodnocení, takže rozdíly mezi pozorováními jsou lépe viditelné.

Protože hlavních komponent je tolik, kolik je v datech proměnných, jsou hlavní komponenty konstruovány tak, aby první hlavní komponenta odpovídala největšímu možnému rozptylu v souboru dat. Předpokládejme například, že graf rozptylu našeho souboru dat je takový, jak je znázorněno níže. První hlavní komponenta je přibližně přímka, která odpovídá fialovým značkám, protože prochází počátkem a je to přímka, ve které je projekce bodů (červených teček) nejvíce rozprostřena. Nebo matematicky řečeno, je to přímka, která maximalizuje rozptyl (průměr čtverců vzdáleností promítnutých bodů od počátku).



Obrázek 3 - vizualizace hledání první hlavní komponenty

Druhá hlavní komponenta se vypočítá stejným způsobem s podmínkou, že není korelovaná s první hlavní komponentou (tj. je na ni kolmá) a že představuje další nejvyšší rozptyl. Takto se pokračuje, dokud se nevypočítá celkem p hlavních komponent, což se rovná původnímu počtu proměnných.

Nyní, když jsme pochopili, co máme na mysli pod pojmem hlavní komponenty, se vraťme k **vlastním vektorům** a **vlastním číslům**. Nejprve o nich musíte vědět, že se vždy vyskytují v párech, takže každý vlastní vektor má vlastní číslo. A jejich počet je roven počtu dimenzí dat. Například pro třírozměrný soubor dat existují 3 proměnné, proto existují 3 vlastní vektory s 3 odpovídajícími vlastními čísly.

Právě vlastní vektory a vlastní čísla stojí za všemi výše vysvětlenými kouzly, protože vlastní vektory kovarianční matice jsou vlastně směry os, kde je největší rozptyl (nejvíce informací) a které nazýváme hlavní komponenty. A vlastní čísla jsou jednoduše koeficienty připojené k vlastním vektorům, které udávají množství rozptylu neseného v každé hlavní složce.

Z datasetu Boston housing price vybereme sloupce NOX, RAD a TAX a vypočítáme z nich kovarianční matici C.

Obrázek 4 - kovarianční matice NOX, RAD, TAX

Pro matici vypočítáme vlastní vektory a vlastní čísla. Charakteristickou rovnici získáme položením následujícího determinantu rovno nule: $|F - \lambda I| = 0$

$$|F - \lambda I| = \left| \begin{pmatrix} 1 & 0{,}63 & 0{,}68 \\ 0{,}63 & 1 & 0{,}9 \\ 0{,}68 & 0{,}9 & 1 \end{pmatrix} - \lambda \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \right| = \begin{vmatrix} 1-\lambda & 0{,}63 & 0{,}68 \\ 0{,}63 & 1-\lambda & 0{,}9 \\ 0{,}68 & 0{,}9 & 1-\lambda \end{vmatrix} = 0$$

$$(1-\lambda)^3 + (0{,}63 \cdot 0{,}9 \cdot 0{,}68)^2 - 0{,}68^2(1-\lambda) - 0{,}63^2(1-\lambda) - 0{,}9^2(1-\lambda) = 0$$

Po vyřešení charakteristické rovnice dostaneme 3 kořeny, které jsou vlastní čísla kovarianční matice:

$$\lambda_1 = 0{,}097 \quad \lambda_2 = 0{,}423 \quad \lambda_3 = 2{,}48$$

Vlastní vektory získáme dosazením vlastních čísel do matice $\begin{pmatrix} 1-\lambda & 0{,}63 & 0{,}68 \\ 0{,}63 & 1-\lambda & 0{,}9 \\ 0{,}68 & 0{,}9 & 1-\lambda \end{pmatrix}$.

Seřazením vlastních vektorů podle jejich vlastních čísel od nejvyššího po nejnižší získáte hlavní komponenty v pořadí podle významnosti.

$$PCA_1 \qquad \lambda_3 = 2{,}48 \qquad \begin{pmatrix} 0{,}877 \\ 0{,}982 \\ 1 \end{pmatrix}$$

$$PCA_2 \qquad \lambda_2 = 0{,}423 \qquad \begin{pmatrix} -2{,}732 \\ 1{,}423 \\ 1 \end{pmatrix}$$

$$PCA_3 \qquad \lambda_1 = 0{,}097 \qquad \begin{pmatrix} -1{,}112 \\ -0{,}918 \\ 1 \end{pmatrix}$$

Poté, co máme hlavní komponenty, abychom vypočítali procento rozptylu (informace), které připadá na každou komponentu, vydělíme vlastní číslo každé komponenty součtem vlastních čísel.

$$PCA_1 \qquad \frac{2{,}48}{2{,}48 + 0{,}423 + 0{,}097} = 0{,}827$$

$$PCA_2 \qquad \frac{0{,}423}{2{,}48 + 0{,}423 + 0{,}097} = 0{,}141$$

$$PCA_3 \qquad \frac{0{,}097}{2{,}48 + 0{,}423 + 0{,}097} = 0{,}032$$

Pokud tento postup použijeme na výše uvedený příklad, zjistíme, že PCA_1, PCA_2 a PCA_3 nesou 82,7%, 14,1 % a 3,23% rozptylu dat.

2.4 Vektor funkcí

Jak jsme viděli v předchozím kroku, výpočet vlastních vektorů a jejich seřazení podle vlastních čísel v sestupném pořadí nám umožní najít hlavní komponenty v pořadí podle významnosti. V tomto kroku se rozhodneme, zda ponecháme všechny tyto složky nebo vyřadíme ty méně významné (s nízkými vlastními čísly), a se zbývajícími vytvoříme matici vektorů, kterou nazveme **vektor příznaků**.

Vektor příznaků je tedy jednoduše matice, která má jako sloupce vlastní vektory složek, které jsme se rozhodli ponechat. To z něj činí první krok ke snížení dimenzionality, protože pokud se rozhodneme ponechat pouze p vlastních vektorů (komponent) z n, bude mít konečný soubor dat pouze p rozměrů.

Pokračujeme-li v příkladu z předchozího kroku, můžeme buď vytvořit vektor příznaků se třemi vlastními vektory:

$$\begin{pmatrix} 0{,}877 & -2{,}732 & -1{,}112 \\ 0{,}982 & 1{,}423 & -0{,}918 \\ 1 & 1 & 1 \end{pmatrix}$$

Nebo vyřadíme vlastní vektor v_3, který je méně významný.

$$\begin{pmatrix} 0{,}877 & -2{,}732 \\ 0{,}982 & 1{,}423 \\ 1 & 1 \end{pmatrix}$$

Vyřazením vlastního vektoru v_3 se sníží dimenzionalita o 1, a v důsledku toho dojde ke ztrátě informací v konečném souboru dat. Ale vzhledem k tomu, že v_3 nesl pouze 3,23% informace, ztráta tedy nebude důležitá.

Jak jsme tedy viděli v příkladu, je na nás, abychom se rozhodli, zda ponecháme všechny složky nebo vyřadíme ty méně významné, v závislosti na tom, co hledáme. Pokud totiž chceme pouze popsat svá data pomocí nových proměnných (hlavních komponent), které nejsou korelované, aniž bychom se snažili snížit dimenzionalitu, vynechání méně významných komponent není nutné.

2.5 Přepočítání dat podél os hlavních komponent

V předchozích krocích kromě standardizace neprovádíme na datech žádné změny, pouze vybereme hlavní komponenty a vytvoříme vektor příznaků, ale vstupní soubor dat zůstává vždy v původních osách (tj. v původních proměnných).

V tomto kroku, který je poslední, je cílem použít příznakový vektor vytvořený pomocí vlastních vektorů kovarianční matice, aby se data přeorientovala z původních os na osy reprezentované hlavními komponentami (odtud název analýza hlavních komponent). To lze provést vynásobením transpozice původního souboru dat transpozicí vektoru příznaků.

$$\text{Výsledný dataset} = \text{Vektor příznaků}^{T} \text{ x Standardizovaný původní dataset}$$

Pokud vypočítáme kovarianční matici pro výsledný dataset, vidíme, že nám téměř zmizela lineární závislost mezi vysvětlujícími proměnnými PCA_1, PCA_2 a PCA_3.

Obrázek 5 – kovarianční matice PCA

Nyní můžeme použít nový dataset například pro výpočet lineární regrese.

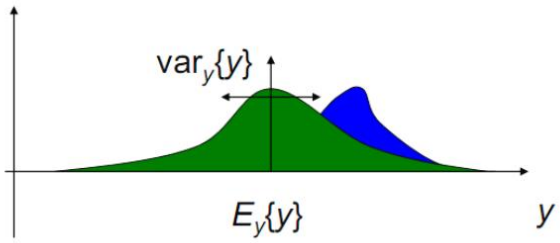
3 Underfitting a overfitting modelu

Když mluvíme o kvalitě modelu vytvořeného strojovým učením, mluvíme vlastně o tom, jak dobře funguje a jaká je jeho přesnost, která je známá jako chyby predikce. Dobře naučený model správně zobecnil trénovací data a je schopen správně předpovídat budoucí data, která nikdy neviděl.

Trénování modelu může skončit dvěma nechtěnými případy – underfitting a overfitting. Jedná se o obecné problémy, které se vyskytují u všech modelů. Pokud model trpí jednou z těchto chyb, při předvídání budoucích hodnot bude nepřesný. Abych toto mohli odhalit rozdělujeme data na trénovací a testovací, která model při učení nevidí.

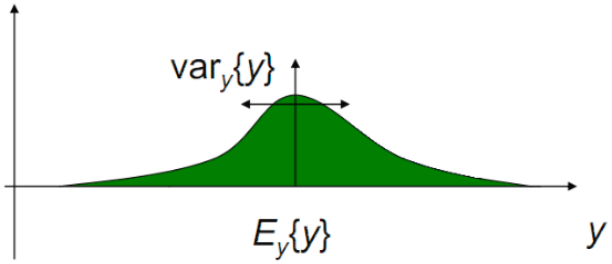
Prvním pojmem je **bias – zaujatost**. Bias označuje chybu způsobenou příliš zjednodušenými předpoklady v učebním algoritmu. Tyto předpoklady usnadňují pochopení a učení modelu, ale nemusí zachytit základní složitost dat. Jedná se o systémovou chybu, která může mít více důvodů.

Je to chyba způsobená neschopností modelu přesně reprezentovat skutečný vztah mezi vstupem a výstupem. Pokud má model špatný výkon na trénovacích i testovacích datech, znamená to vysoké zkreslení z důvodu jednoduchého modelu.



Obrázek 6 - bias modelu

Rozptyl chyby modelu – variance je naopak chyba způsobená citlivostí modelu na výkyvy v trénovacích datech. Je to rozptyl předpovědí modelu pro různé případy trénovacích dat. Vysoký rozptyl vzniká, když se model učí spíše šumu a náhodným výkyvům trénovacích dat než podstatě dat. Výsledkem je, že model funguje dobře na trénovacích datech, ale špatně na testovacích datech, což svědčí o nadměrném přizpůsobení.



Obrázek 7 – rozptyl chyby modelu

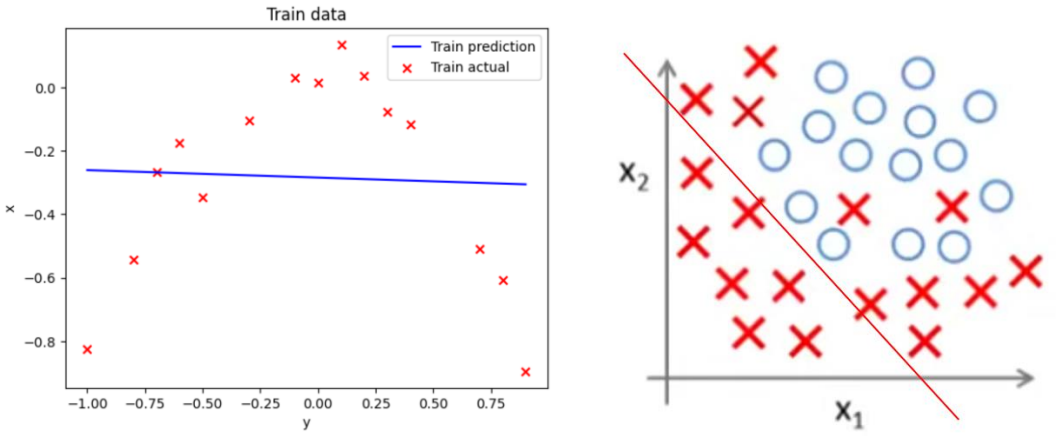
3.1 Nedostatečné přizpůsobení – underfitting

O nedostatečném přizpůsobení statistického modelu nebo algoritmu strojového učení se hovoří tehdy, když je model příliš jednoduchý na to, aby zachytil složitost dat. Představuje neschopnost modelu efektivně se naučit trénovací data, což má za následek špatný výkon na trénovacích i testovacích datech. Zjednodušeně řečeno, nedostatečně přizpůsobený model je nepřesný, zejména při aplikaci na nové, neznámé příklady. Stává se to hlavně tehdy, když používáme velmi jednoduchý model s příliš zjednodušenými předpoklady. Výsledný model má vysoký bias a nízký variance.

Důvody underfittingu

- Model je příliš jednoduchý a není schopen reprezentovat složitosti v datech
- Vstupní proměnné, které se používají k trénování modelu, nejsou adekvátní reprezentací základních faktorů ovlivňujících cílovou proměnnou
- Velikost použitého trénovacího souboru dat není dostatečná
- K zabránění nadměrného přizpůsobení se používá nadměrná regularizace, která omezuje model, aby dobře zachytil data. Viz dále.

- Proměnné nejsou standardizovány.



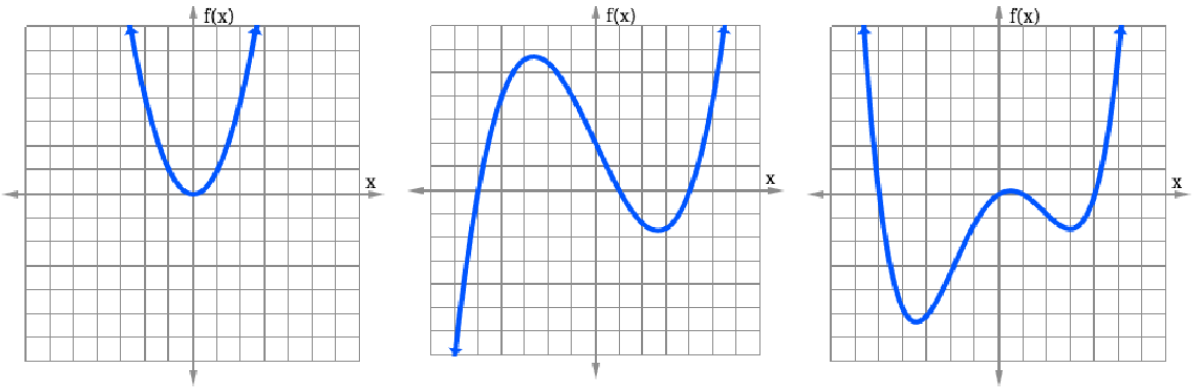
Obrázek 8 - příklady underfittingu pro regresi a klasifikaci

Techniky, které mohou pomoci odstranit underfitting:

- Zvýšení složitosti modelu
- Zvýšení počtu vstupních proměnných, změna vstupních proměnných
- Odstranění šumu z dat
- Zvýšení počtu epoch trénování

3.1.1 Polynomiální regrese

Pro výše uvedená data byl zvolen nevhodný lineární model, u kterého nepomůže delší učení, odstranění šumu atd. V této kapitole zkusíme zvolit jiný model – polynomiální regresi. Tento model se pokouší proložit vstupní data polynomem. Důležitým parametrem je pro ni stupeň polynomu, který určuje počet parametrů a tvar funkce. Vyšší stupeň umožňuje generovat složitější tvary funkce.



Obrázek 9 - polynomy stupně 2, 3 a 4

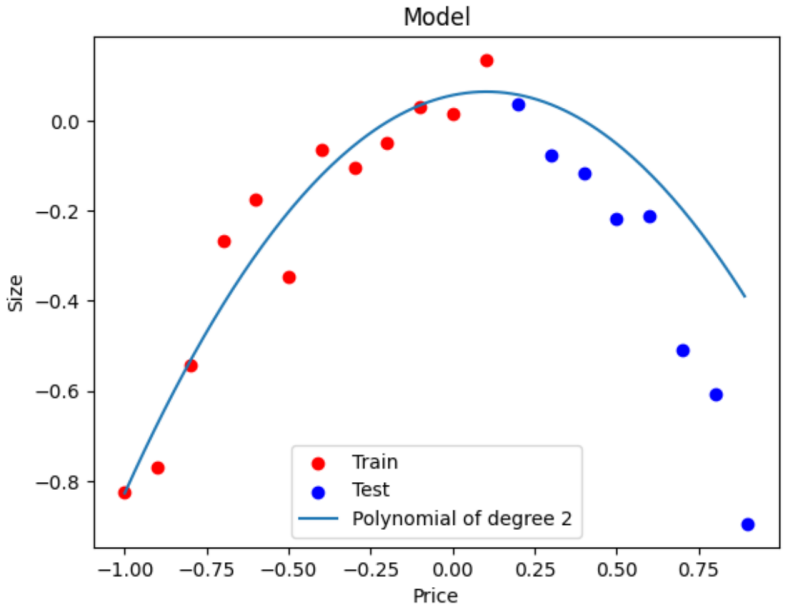
Obecná rovnice polynomu má následující tvar:

$$f_{w,b} = b + w_1 \cdot x + w_2 \cdot x^2 + \ldots + w_n \cdot x^n$$

Lineární regrese je tedy zvláštním případem polynomiální regrese se stupněm 1.

Nyní použijeme místo lineárního modelu, polynomiální model se stupněm 2. Výsledný model pro naše data může mít například následující podobu:

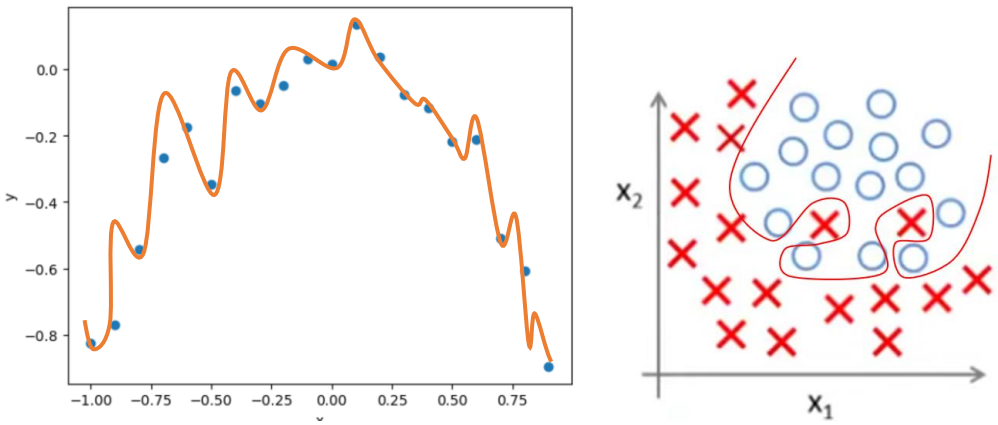
$$f_{w,b} = -0{,}73 + 0{,}15\,x + 0{,}056\,x^2$$



Obrázek 10 - polynomiální regrese

3.2 Nadměrné přizpůsobení - overfitting

Druhým extrémním příkladem je overfitting, když model neposkytuje přesné předpovědi pro testovací, respektive budoucí data, ale poskytuje přesné odpovědi pouze pro trénovací data. Pokud budeme model trénovat příliš a budeme se snažit snižovat jeho chybu na trénovacích datech, zařadí do modelu i šum a nepřesnosti trénovacích dat. Pak model velmi dobře předvídá trénovací data, ale pro testovací a reálná data může vracet nereální výsledky.



Obrázek 11 – příklady overfitting

Příčinou overfittingu jsou neparametrické a nelineární metody, protože tyto typy algoritmů strojového učení mají větší volnost při sestavování modelu a mohou tak sestavovat opravdu nereálné

modely. Modely, které postihuje problém overfittingu mají nízký bias, ale vysoký rozptyl. Hodnota nákladové funkce J je rovna 0 nebo se k ní blíží.

Důvody overfitting:

- Model je příliš složitý
- Trénovací data jsou příliš malá
- Dlouhá doba trénování

Techniky pro odstranění overfittingu:

- Volba jednoduššího modelu, který je více lineární
- Omezení počtu parametrů, hloubky učení apod.
- Zvětšení velikosti trénovacích dat
- Včasné zastavení trénování
- Omezení počtu vstupních parametrů do modelu
- Použití regulace parametrů, kdy ponecháme všechny vstupní proměnné, ale omezíme velikost parametrů w
- U neuronových sítí použití Dropout, která sníží v některých vrstvách počet neuronů

3.2.1 Regulace modelu

Velikost parametrů w by měla být co nejnižší, aby výsledný model byl jednoduchý. Pokud budeme například vytvářet polynomiální model ze všech vstupních proměnných, výsledný model bude obsahovat větší počet parametrů w. Jejich velikost může mít zásadní vliv na to, jak bude mít vstupní proměnná vliv na výslednou proměnnou.

Jednou z možností, jak zabránit, aby nějaká vstupní proměnná měla příliš velký vliv díky vysokému parametru w, je vložení regulačního parametru λ do modelu. Do nákladové funkce modelu je přidána suma druhých mocnin parametrů w vynásobených regulačním parametrem λ.

$$J(w,\ b) = \frac{1}{2m}\left[\sum_{i=1}^{m}(f_{w,b}(x^i) - y^i)^2 + \lambda\sum_{j=1}^{n}{w_j}^2\right]$$

Proces učení si bude při takovéto nákladové funkci vybírat dostatečně malé hodnoty parametrů w. Stupeň regulace se řídí hodnotou λ. Pokud ji nastavíme příliš nízkou, regulace nebude probíhat. Pokud λ nastavíme na příliš vysokou hodnotu, nedovolíme modelu volit vhodné hodnoty parametrů w a model bude vykazovat underfit.

3.3 Výběr modelu

Pokud budeme chtít pro naše data vybrat správný model, který nebude trpět problémy underfit nebo overfit je vhodné sledovat průběh učení.

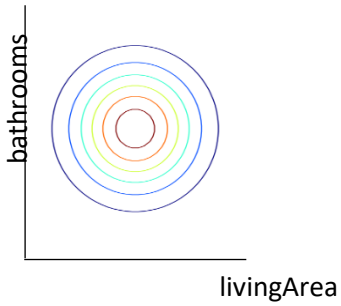
Na začátku je dobré si datový soubor rozdělit na 3 podmnožiny – trénovací, validační a testovací. Poměr může být různý. Často se používají poměry - 60:20:20, 80:10:10 nebo 90:5:5.

Při trénování různých modelů budeme používat stejnou trénovací množinu. Různé modely – lineární, polynomiální, neuronová síť atd. se snaží minimalizovat nákladovou funkci J a nalézt kombinaci parametrů modelu, pro kterou je nejnižší střední hodnota chyby. Nákladové funkce jsou samozřejmě odlišné pro různé typy modelů. Mimo volby typu modelu můžeme zkoušet vytvořit různé modely s různými hodnotami volitelných parametrů. Například pro polynomiální regresi si můžeme volit různý stupeň.

Po vytrénování různých modelů se spočítají hodnoty nákladových funkcí těchto modelů na validačních datech. Tyto nákladové funkce již neobsahují regulační parametry, ty byly použity pouze pro trénování.

Z množiny natrénovaných modelů se vybírá ten s nejnižšími náklady.

Na vybraném modelu se jeho přesnost určuje na testovací množině dat. Mohlo se stát, že výběr správného modelu byl zatížen skladbou validačních dat.



Obrázek 12 - nákladová funkce normalizované proměnné

3.3.1 MinMax normalizace

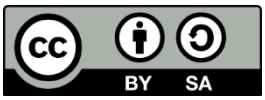
Jinou normalizací je MinMax, kdy se nová proměnná spočítá pomocí minima a maxima původní proměnné.

$$x_i' = \frac{x_i - \min(x_i)}{\max(x_i) - min(x_i)}$$

V tomto případě hodnoty nové proměnné budou v rozsahu 0 až +1.

UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vytvořeno v rámci projektu **Digitalizace studijních Agend, Nové TechnologiE, systémy a přístupy k výuce na UPCE**, reg. č. NPO_UPCE_MSMT-16591/2022.



Financováno Evropskou unií
NextGenerationEU

Národní plán obnovy

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Machine learning & AI

Klasifikace

Ing. Martin Pozdílek, Ph.D.

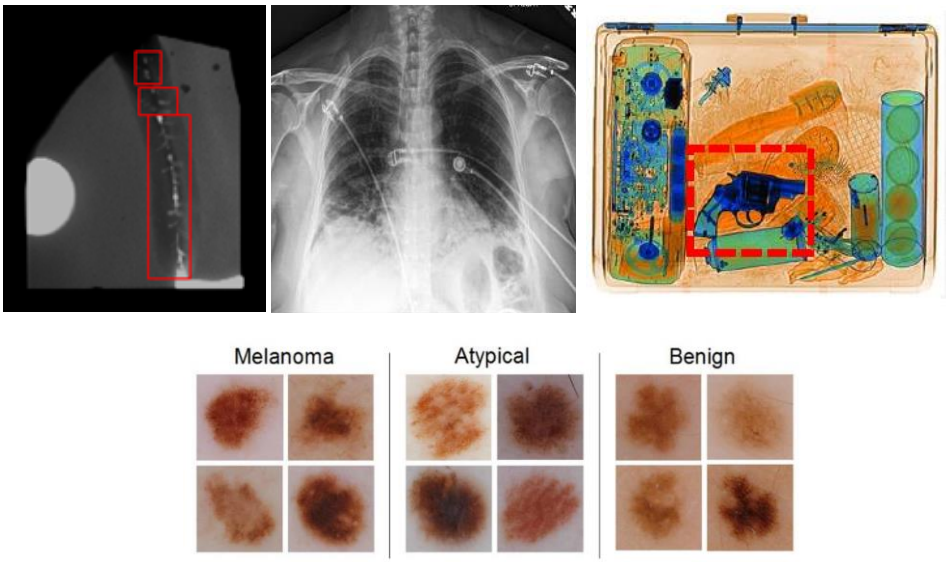
1 Klasifikace

Doposud jsme se ve strojovém učení zabývali regresí. Regrese je metoda zjišťování funkce nebo modelu pro předpověď hodnot nějaké veličiny. Výstupem je **reálné** číslo.

Naproti tomu úloha **klasifikace** je proces zjišťování nebo identifikace objektů, jevů podle jejich parametrů do definovaných kategoriálních tříd. Pod třídou si můžeme představit nějakou **diskrétní hodnotu**. Vstupní data, která se mají klasifikovat mohou mít různou formu, jako jsou text, obrazy, zvukové záznamy nebo číselné hodnoty. Vstupní data se pomocí různých modelů přiřazují do tříd nebo kategorií.

V reálném životě se setkáváme s různými úlohami, které se řadí do klasifikace.

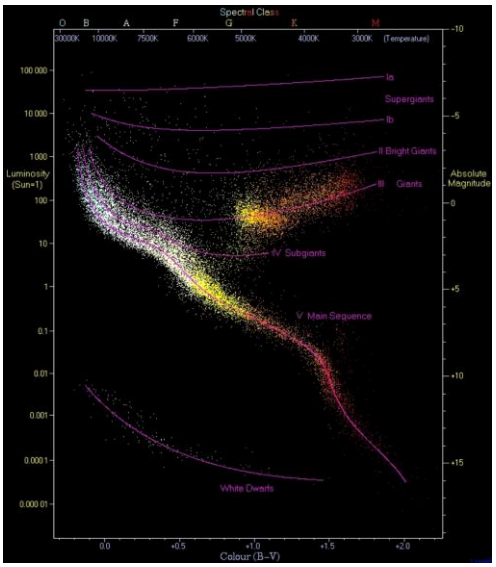
- **Detekce spamových e-mailů**: Klasifikace e-mailů do dvou tříd - "spam" a "ne-spam" - na základě obsahu a dalších charakteristik e-mailu.
- **Diagnostika nemocí**: Klasifikace pacientů do různých diagnostických kategorií na základě výsledků lékařských testů, jako je například detekce rakoviny na základě obrazových snímků.
- **Rozpoznávání obrazů**: Klasifikace obrázků do různých kategorií, například rozpoznávání druhů zvířat nebo klasifikace produktů v obchodě.
- **Detekce podvodu s platebními kartami**: Klasifikace transakcí na základě jejich rizikovosti a identifikace možných podvodů s platebními kartami.
- **Sentimentální analýza**: Klasifikace textových recenzí nebo komentářů na sociálních médiích do pozitivních, negativních nebo neutrálních kategorií.
- **Rozpoznávání rukou psaných znaků**: Klasifikace rukou psaných znaků nebo písmen do abecedních znaků.
- **Detekce hrozby v kybernetické bezpečnosti**: Klasifikace síťového provozu do tříd, jako jsou „normální provoz“ a „potenciální útoky“.
- **Klasifikace objektů v autonomním řízení**: Rozpoznávání a klasifikace různých objektů na silnici, jako jsou auta, chodci a cyklisté, v autonomních vozidlech.
- **Klasifikace pěstovaných plodin v zemědělství**: Rozpoznávání různých druhů plodin nebo škůdců na polích na základě obrazových dat z dronů.
- **Detekce chorob v radiologii**: Klasifikace rentgenových snímků nebo CT snímků na základě přítomnosti a typu choroby, jako je například detekce pneumonie nebo nádorů.



Obrázek 1 - příklady klasifikace

Zvláštním typem úloh je shluková analýza (clustering). **Shluk** je skupina datových bodů, které jsou si navzájem podobné nebo mají nějakou společnou charakteristiku. Podobnost může být například definovaná jako vzdálenost bodů od sebe. Reálných příkladů je opět více:

- **Segmentace zákazníků:** Rozdělení zákazníků na skupiny podle jejich nákupního chování nebo demografických charakteristik pro cílení marketingových kampaní.
- **Analýza textu**: Seskupování podobných dokumentů nebo slov na základě jejich obsahu (google new).
- **Astronomie**: Typy hvězd na základě jejich vlastností (povrchová teplota, absolutní hvězdná velikost). **Hertzsprungův–Russellův diagram** (bílý trpaslící, hlavní posloupnost, obři, veleobři)



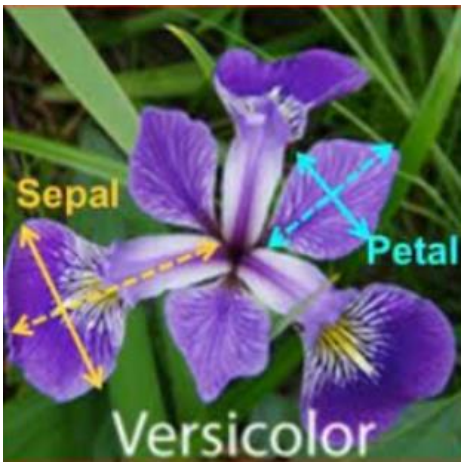
Obrázek 2 - H–R diagramu, Autor: Adam na projektu Wikipedie v jazyce čeština – Na Commons přeneseno z cs.wikipedia., Volné dílo, https://commons.wikimedia.org/w/index.php?curid=2157609

Klasifikačních algoritmů je celá řada. Liší se předpoklady na data, výpočetní náročností a principy. Protože algoritmy používají různé principy může se stát, že pro určitá data bude jeden algoritmus vracet špatné výsledky a jiný bude vracet výsledky dobré. Vždy je vhodné vyzkoušet různé algoritmy s různými parametry a vybrat ten, který na daná data funguje nejlépe.

Mezi populární algoritmy patří:

- Rozhodovací strom
- Logistická regrese
- k-nejbližších sousedů (k-NN)
- K-means
- Support Vector Machine (SVM)
- Neuronové sítě

Algoritmy si budeme ukazovat na legendárním datovém setu Iris. Britský statistik a biolog Ronald Fisher publikoval v roce 1936 článek The use of multiple measurements in taxonomic problems jako příklad lineární diskriminační analýzy. Dva ze tří druhů kosatců byly sbírány na poloostrově Gaspé. Všechny ze stejné pastviny, ve stejný den a měřeny ve stejnou dobu stejnou osobou a stejným přístrojem. Soubor dat se skládá z 50 vzorků od každého ze tří druhů kosatce. U každého vzorku byly změřeny čtyři znaky: délka a šířka kališních (sepal) a okvětních (petal) lístků v centimetrech.



Obrázek 3 – okvětní a kališní listky

Obrázek 4 - iris setosa, iris multicolor, iris virginica

2 Příprava dat

Jak již bylo řečeno, výstupem klasifikačních algoritmů je diskrétní hodnota, identifikace třídy. Velmi často se stává, že data obsahují identifikaci třídy v podobě řetězce, protože tato podoba je čitelnější pro lidi.

Tato podoba tříd však není vhodná pro modely, které vyžadují číselné hodnoty. Proto je častou součástí analýzy příprava dat, konkrétně **kódování názvů** (label encoding).

Prvním způsobem je přiřazení celočíselné hodnoty unikátnímu řetězci. V našem případě jednotlivým druhům kosatců přiřadíme čísla 0 až 2.

	sepal_length	sepal_width	petal_length	petal_width	species	species_enc
0	5.1	3.5	1.4	0.2	Iris-setosa	0
1	4.9	3.0	1.4	0.2	Iris-setosa	0
60	5.0	2.0	3.5	1.0	Iris-versicolor	1
61	5.9	3.0	4.2	1.5	Iris-versicolor	1
120	6.9	3.2	5.7	2.3	Iris-virginica	2
121	5.6	2.8	4.9	2.0	Iris-virginica	2

Obrázek 5 - label encoding

Druhým způsobem je **binární kódování** (binary encoding), který je vhodný například pro neuronové sítě. Místo jedné číselné proměnné vytvoříme n binárních proměnných, kde n je počet tříd. Tyto proměnné budou nabývat hodnot (true, 1) a (false, 0, -1) podle toho, do které třídy patří.

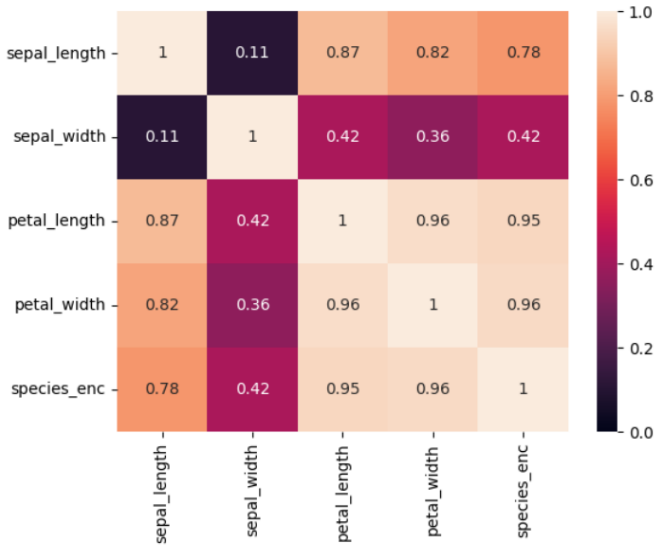
	sepal_length	sepal_width	petal_length	petal_width	species_enc	species_Iris-setosa	species_Iris-versicolor	species_Iris-virginica
0	5.1	3.5	1.4	0.2	0	1	0	0
1	4.9	3.0	1.4	0.2	0	1	0	0
60	5.0	2.0	3.5	1.0	1	0	1	0
61	5.9	3.0	4.2	1.5	1	0	1	0
120	6.9	3.2	5.7	2.3	2	0	0	1
121	5.6	2.8	4.9	2.0	2	0	0	1

Obrázek 6 - binary encoding

V minulé kapitole o PCA jsme si ukazovali postup, jak snížit dimenzionalitu dat v případě závislých vstupních proměnných. Kovarianční matice datasetu iris ukazuje silnou přímou závislost mezi výstupní proměnnou typu kosatce a šířkou a délkou okvětního lístku. Zároveň šířka a délka lístku je také silně korelovaná.

Dimenzionalitu dat můžeme jednak snížit vypuštěním proměnných sepal_lenght a sepal_width. Zároveň můžeme vytvořit novou proměnnou – povrch okvětního lístku, kterou

odhadneme vzorcem $petal\_width \cdot petal\_height$. Ukážeme se si, že model vytvořený nad takto redukovanými daty může být stále dostatečně přesný.

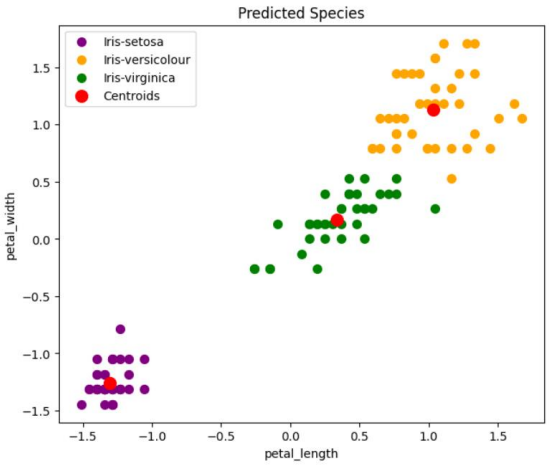


Obrázek 7 - kovarianční matice iris

3 K-means algoritmus

K-means je algoritmus založený na představě, že statistické objekty lze chápat jako body v eukleidovském vícerozměrném prostoru, kde jednotlivé vlastnosti objektu tvoří jeho osy. Pro názornost budeme používat příklady, kde data obsahují dvě nebo tři vlastnosti, protože pak si lze model představit a vizualizovat.

Jak název napovídá, jedním ze vstupních parametrů je počet shluků (clusteru, segmentů) **K**, na kolik má algoritmus data rozdělit. Tento počet je zřejmý ze zadání úlohy nebo jeho počet je třeba určit (viz dále).



Obrázek 8 - K-mean algoritmus

Algoritmus najde K centrálních bodů (centroidů) a konkrétní výskyt patří do shluku, do jehož centra má nejkratší vzdálenost.

Algoritmus se řadí do modelu bez učitele, který nepředpokládá, že data obsahují správné odpovědi.

3.1 Algoritmus

Algoritmus se skládá z několika kroků.

3.1.1 Krok 1: Inicializace

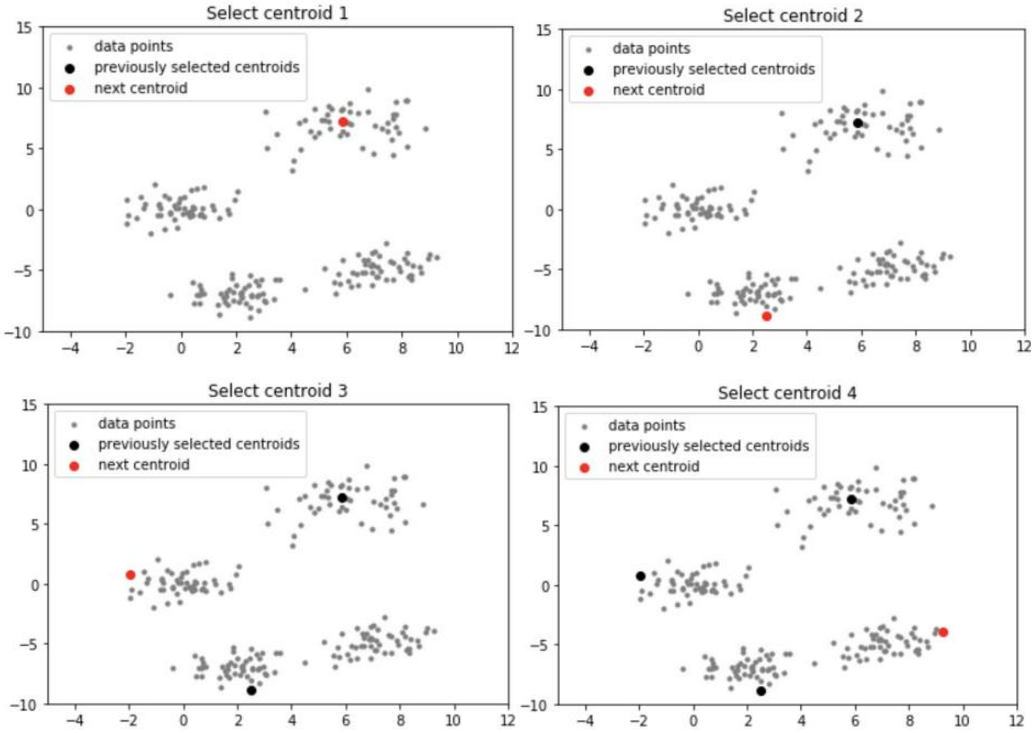
Na začátku je zvoleno K počátečních bodů (středních hodnot, centroidů), které reprezentují středy shluků. Centroidy lze zvolit náhodně nebo pomocí algoritmu **K-means++**. Rychlost algoritmu bude záležet na vhodném počátečním nastavení centroidů.

Centroidy můžeme zvolit náhodně, vhodnější je použít funkci K-means++. Použití funkce K-means++ sice prodlužuje inicializační fázi, ale protože navržené počáteční centroidy lépe reflektují rozložení dat než náhodně vybrané středy, samotný K-means algoritmus bude konvergovat k optimálnímu řešení rychleji. Celkově se tedy výsledný čas algoritmu zkrátí.

Nalezení přesného řešení pro libovolný vstup je algoritmem K-means NP obtížné (doba běhu algoritmu je superpolynomiální). Proto se často používá nalezení přibližného řešení pomocí **Lloydův** algoritmu, který tuto náročnost snižuje.

Prvním úkolem algoritmu K-means je tedy nalezení centroidů, tedy center oblastí, v nichž se body shlukují. Algoritmus K-mean++ je následující:

1. Vyberme jedno centrum rovnoměrně náhodně mezi vstupními datovými body.
2. Pro každý datový bod x , který ještě nebyl vybrán, vypočítáme vzdálenost d(x), mezi bodem x a nejbližším středem, který již byl vybrán.
3. Vyberme náhodně jeden nový datový bod jako nový střed pomocí váženého rozdělení pravděpodobnosti, kde je bod *x* vybrán s pravděpodobností úměrnou $d(x)^2$. Jinými slovy, vzdálenější body mají větší pravděpodobnost být vybrány za nové centrum.
4. Opakujme kroky 2 a 3, dokud není zvoleno K středů.



Obrázek 9 - vizualizace algoritmu K-means++

3.1.2 Krok 2: Přiřazení ke shlukům

Pro každý datový bod ve vstupním datasetu je spočítána vzdálenost ke všem centroidům. Bod přiřadíme k nejbližšímu shluku.

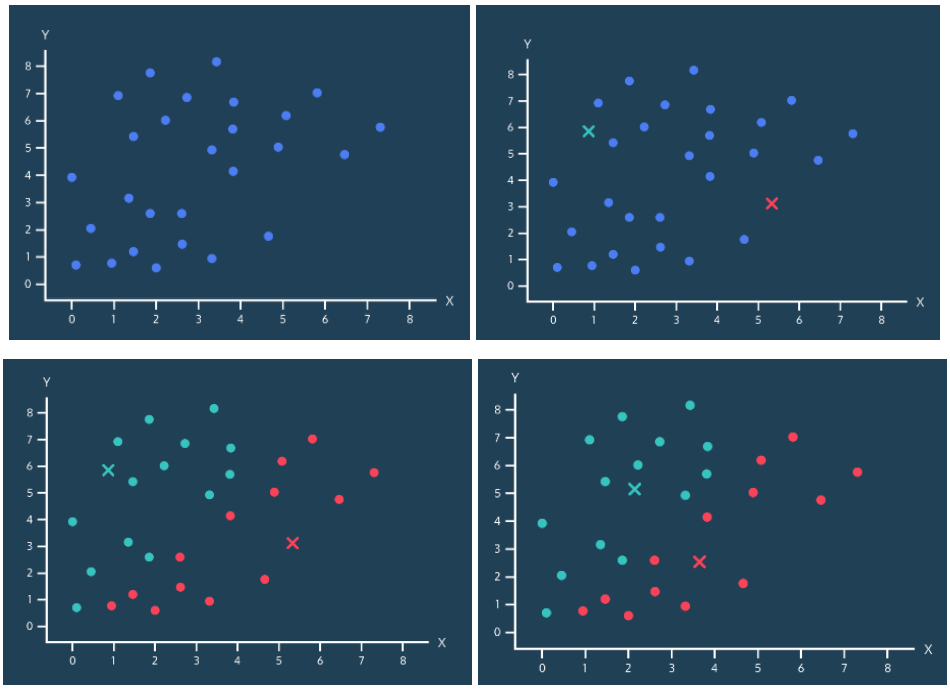
3.1.3 Krok 3: Aktualizace středních hodnot

Po přiřazení všech bodů ke shlukům spočítáme nové střední hodnoty pro každý shluk. Střední hodnota shluku je průměrný bod v tomto shluku. Jinými slovy střed shluku vycentrujeme do středu aktuálního shluku. Po změně centroidů se může stát, že některé body se stanou součástí jiného shluku.

Algoritmus se snaží minimalizovat vnitrotřídní rozptyl, tj. součet čtverců vzdáleností každého bodu patřícího do shluku k jeho centru.

3.1.4 Opakování

Kroky 2 a 3 se opakují, dokud se střední hodnoty shluků a přiřazení bodů ke shlukům nezmění jen minimálně nebo dokud není dosaženo určitého kritéria zastavení (například určený počet iterací). Jakmile algoritmus konverguje (střední hodnoty a přiřazení se přestanou výrazně měnit), máme určené výsledné shluky. Každý bod je nyní přiřazen ke konkrétnímu shluku.



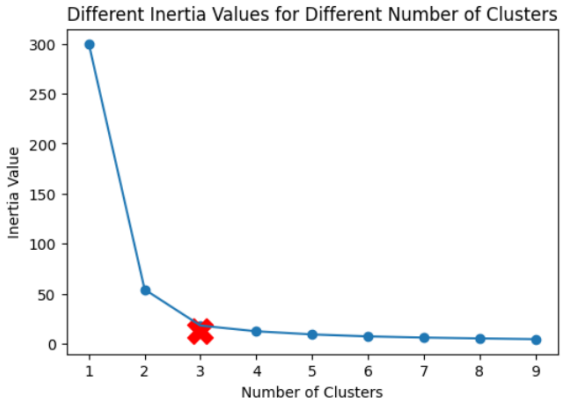
Obrázek 10 - Vizualizace algoritmu – výběr centroidů, přiřazení bodů shlukům a zpřesnění středu

3.2 Nalezení správného počtu k

Setrvačnost (inertia) měří, jak dobře byl soubor dat shlukován pomocí K-Means algoritmu. Setrvačnost se vypočítá tak, že se změří vzdálenost mezi každým datovým bodem a jeho centroidem. Tato vzdálenost se umocní na 2 a tyto čtverce se sečtou pro daný shluk.

$$Inertia = \sum_{i=1}^{n} (x_i - C_k)^2$$

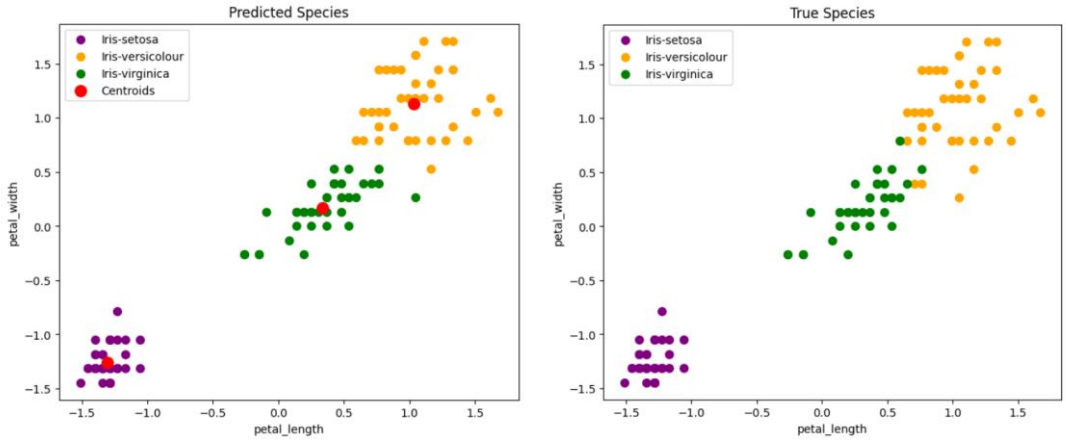
Dobrý model je takový, který má nízkou setrvačnost a nízký počet shluků K. Jedná se však o kompromis, protože s rostoucím K setrvačnost klesá. Chceme-li najít optimální K pro soubor dat, použijte metodu Elbow. Metoda hledá bod, kde se pokles setrvačnosti začíná zpomalovat. Pro následující graf daným bodem K=3. Toto místo se označuje jako „loket“ grafu.



Obrázek 11 - elbow graf

3.3 Model K-Means

Na následujících obrázcích jsou zobrazeny výsledky K-means modelu na datovém setu Iris a skutečné hodnoty. Z obrázků je patrné, že algoritmus funguje správně u izolovaných shluků, pokud se hodnoty prolínají, tak okrajové body mohou být klasifikovany chybně.



Obrázek 12 - výsledky K-means vs. skutečnost

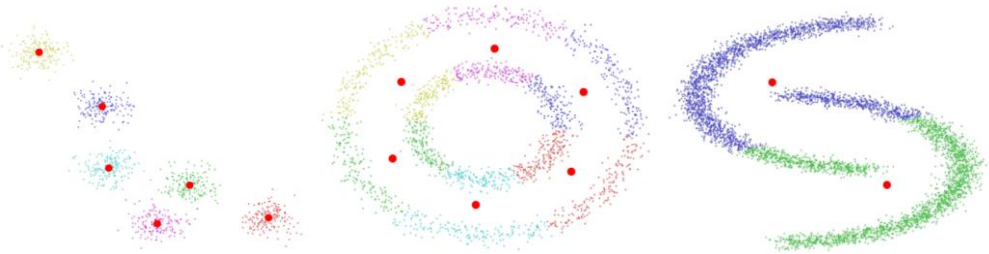
3.4 Vlastnosti algoritmu

3.4.1 Výhody

- Algoritmus je zpravidla rychlý a efektivní pro velká množství vhodných dat.

3.4.2 Nevýhody

- Algoritmus je náchylný na počáteční volbu středních hodnot, což může vést k různým výsledkům.
- Algoritmus není schopen pracovat s různými velikostmi a tvary shluků. Obecně algoritmus dobře pracuje s relativně izolovanými shluky v podobě kruhu, koule apod. Pokud mají data protáhlý nebo komplikovanější tvar, algoritmus nebude fungovat správně.
- Algoritmus může uváznout v lokálním optimu, takže by měl být spuštěn několikrát s různými počátečními hodnotami.



Obrázek 13 - K-means a tvary dat

4 Hodnocení klasifikačních modelů

U regresních modelů, které vrací spojitou hodnotu, jsme přesnost hodnotili střední absolutní chybou (MAE) nebo střední kvadratickou chybou (MSE).

U klasifikačních modelů, které vrací diskrétní hodnotu, budeme postupovat odlišně. Představte si pacienta, u kterého určujeme, zda trpí nějakou chorobou. Odpověď klasifikačního modelu můžeme zařadit do jedné ze 4 možných odpovědí.

Mohou nastat dvě správné predikce a dvě nesprávné predikce.

- **Chyba typu I** (sýčkování) – falešně pozitivní (pacient je zdravý, ale diagnostikujeme chorobu)
- **Chyba typu II** (důvěřivost) – falešně negativní (pacient je nemocný, ale určíme, že je zdravý).

Při hodnocení modelu velmi záleží na situaci, který typ chyb nám vadí a který ne, případně jak moc. V některých případech klademe důraz, aby byly správně klasifikovány pozitivní případy a pokud nastane situace falešně pozitivní, tak to nemusí být tak velký problém (hledání zmetků). Existuje více metrik, které měří přesnost modelu a zaměřují se na různé typy chyb.

		Skutečnost	
		Pozitivní	Negativní
Předpověď	Pozitivní	True positive pacient je nemocný a tvrdíme, že je nemocný	False positive pacient je zdravý, ale tvrdíme, že je nemocný
	Negativní	False negative pacient je nemocný, ale tvrdíme, že je zdravý	True negative pacient je zdravý a tvrdíme, že je zdravý

4.1 Metriky

- **správnost** $(accuracy) = \frac{TP+TN}{TP+TN+FP+FN} = \frac{počet\ správných\ odpovědí}{počet\ všech\ odpovědí}$
- **specificita** $(specifity, selectivity) = \frac{TP}{TP+FN} = správně\ diagnostikovaní\ nemocní$
- **citlivost, sensitivita** $(sensitivity, recall) = \frac{TN}{TN+FP} = správně\ diagnostikovaní\ zdraví$
- **přesnost** $(precision) = \frac{TP}{TP+FP} = míra\ přesnosti, kdy\ byl\ diagnostikován\ jako\ pozitivní$

- $\mathbf{F1}\ score = 2 \cdot \frac{precision \cdot recall}{precission + recall} = harmonický\ průměr\ přesnosti\ a\ specifičity$

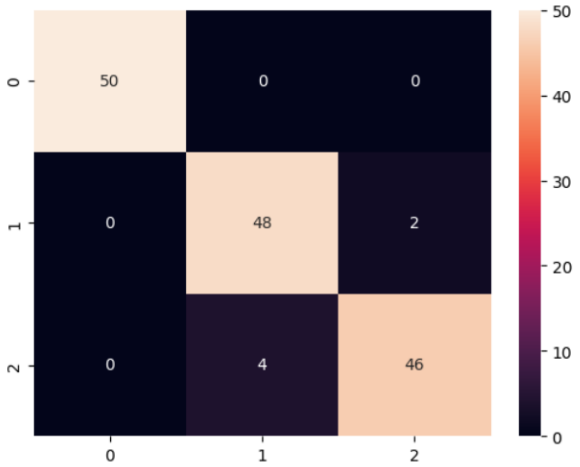
		okultní krvácení ve stolici Výsledek testu na			Pohled · mluvit · Upravit
	Celková populace (pop.) = 2030	Výsledek testu **pozitivní**	Výsledek testu **negativní**	přesnost (ACC) = (TP + TN) / pop. = (20 + 1820) / 2030 ≈ **90.64%**	F1 _ skóre $= 2 \times \frac{\text{přesnost} \times \text{vyvolání}}{\text{přesnost} + \text{vyvolání}}$ ≈ **0.174**
Pacienti s rakovina střev (jak potvrzeno na endoskopii)	Aktuální stav pozitivní	**Skutečně pozitivní** (TP) = 20 (2030 × 1.48% × 67%)	**Falešně negativní** (FN) = 10 (2030 × 1.48% × (100% − 67%))	Skutečná pozitivní míra (TPR), zapamatování , **citlivost** = TP / (TP + FN) = 20 / (20 + 10) ≈ **66.7%**	Frekvence falešných záporů (FNR), četnost chyb = FN / (TP + FN) = 10 / (20 + 10) ≈ **33.3%**
	Aktuální stav negativní	**Falešně pozitivní** (FP) = 180 (2030 × (100% − 1.48%) × (100% − 91%))	**Skutečně negativní** (TN) = 1820 (2030 × (100% − 1.48%) × 91%)	Frekvence falešných poplachů (FPR), výpadek , pravděpodobnost falešného poplachu = FP / (FP + TN) = 180 / (180 + 1820) ≈ **9.0%**	**Specifičnost** , selektivita, skutečný negativní poměr (TNR) = TN / (FP + TN) = 1820 / (180 + 1820) ≈ **91%**
	Prevalence = (TP + FN) / pop. = (20 + 10) / 2030 ≈ **1.48%**	Pozitivní prediktivní hodnota (PPV), přesnost = TP / (TP + FP) = 20 / (20 + 180) ≈ **10%**	Míra falešných opomenutí (FOR) = FN / (FN + TN) = 10 / (10 + 1820) ≈ **0.55%**	Pozitivní poměr pravděpodobnosti (LR+) $= \frac{\text{TPR}}{\text{FPR}}$ = (20 / 30) / (180 / 2000) ≈ **7.41**	Negativní poměr věrohodnosti (LR−) $= \frac{\text{FNR}}{\text{TNR}}$ = (10 / 30) / (1820 / 2000) ≈ **0.366**
		Míra falešných objevů (FDR) = FP / (TP + FP) = 180 / (20 + 180) ≈ **90.0%**	Negativní prediktivní hodnota (NPV) = TN / (FN + TN) = 1820 / (10 + 1820) ≈ **99.45%**	Diagnostický poměr šancí (DOR) $= \frac{\text{LR+}}{\text{LR-}}$ ≈ **20.2**	

Obrázek 14 - tabulka metrik pro test rakoviny střev určené z krve ve stolici. Zdroj wikipedia

4.2 Matice záměn

Výše uvedená tabulka je vhodná pro situace, kde predikční model umí předvídat pouze dvě kategorie. Pokud klasifikační model vrací více možných tříd, je vhodné zkoumat matice záměn.

Horizontální řádky jsou předpovědi modelu, vertikální sloupce reprezentují skutečnost.



Obrázek 15 - matice zmatení K-means pro Iris

Na tomto obrázku je ukázána matice záměn modelu K-means pro dataset iris. Na hlavní diagonále jsou vidět počty správných odpovědí. Z matice je vidět, že model správně identifikuje iris setosa (0), protože tento shluk leží izolovaný a vzdálený od zbývající shluků.

Model ve dvou případech zaměnil isir multicolor (1) za iris virginica (2). Ve čtyřech případech zaměnil iris virginica (2) za isir multicolor (1).

4.3 Nerovnováhy tříd

V klasifikačních úlohách můžeme narazit na problém nazývaný **nerovnováha tříd** (class imbalance problem). Jedná se o situaci, která nastává, když máme ve vstupním datasetu výrazný nepoměr mezi počtem příkladů jednotlivých tříd. Třída může být minoritní, pokud obsahuje mnohem méně příkladů než ostatní třídy. Případně můžeme mít majoritní třídu, která obsahuje mnohem více příkladů než ostatní třídy.

Tento nepoměr může být způsoben různými faktory a může mít významný vliv na výkon a spolehlivost modelů strojového učení. Zde je několik důvodů, proč nerovnováha tříd může být problém:

- **Nízká přesnost pro minoritní třídu**: Modely strojového učení mají tendenci být více zaměřeny na dosažení co nejlepší přesnosti pro majoritní třídu, což vede k nižší přesnosti pro minoritní třídu. To může být problém v úlohách, kde je důležité detekovat vzácné jevy, jako jsou podvody, nebo kdy je potřeba zachránit životy, jako v medicínských aplikacích.
- **Nevhodná evaluace modelu**: Používání běžných metrik, jako je správnost (accuracy), může být zavádějící, protože i model, který vždy předpovídá majoritní třídu, dosáhne vysoké přesnosti v nerovnovážném datasetu. Je tedy nutné používat vhodnější metriky, jako jsou citlivost, F1 skóre.
- **Nízká generalizace**: Nerovnováha tříd může vést k tomu, že modely se zaměří na majoritní třídu a nejsou schopny generalizovat na minoritní třídu. To vede ke špatné schopnosti modelu předpovědět minoritní třídu na nových datech.
- **Zvýšené náklady chyby:** V některých aplikacích mohou chyby modelu v minoritní třídě mít závažnější následky než chyby v majoritní třídě. Například, pokud se jedná o detekci závažných onemocnění, chyba může mít fatální důsledky**.**

Řešení problému nerovnováhy tříd zahrnuje použití různých technik, jako je **oversampling** (zpětné duplikování vzorků minoritní třídy), **undersampling** (snížení vzorků majoritní třídy), použití vážených ztrátových funkcí a použití pokročilých algoritmů, jako je Random Forest nebo Gradient Boosting, které jsou schopny lépe pracovat s nerovnovážnými daty. Správné vyřešení problému

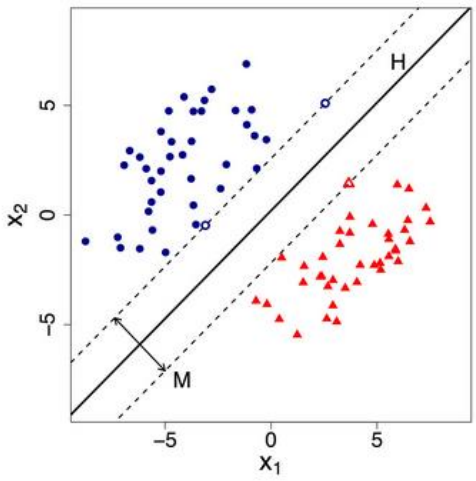
nerovnováhy tříd je klíčem k vytvoření spolehlivých a užitečných modelů strojového učení v takových situacích.

5 Support Vector Machine

Algoritmus Support Vector Machine (SVM) je metoda strojového učení, která se používá pro klasifikaci datových bodů.

SVM jsou široce používány v oborech, jako je zdravotnictví, zpracování přirozeného jazyka, aplikace pro zpracování signálů a rozpoznávání řeči a obrazu. Není bez zajímavosti, že mezi tvůrce patří i český vědec Vladimír Vápník.

Cílem SVM je nalezení hranice, která rozdělí data do dvou kategorií. Na následujícím obrázku jsou vyznačené modré body, které spadají do jedné kategorie a červené body, která patří do druhé kategorie. SVM se snaží nalézt hranici, která tyto kategorie nejlépe odděluje.



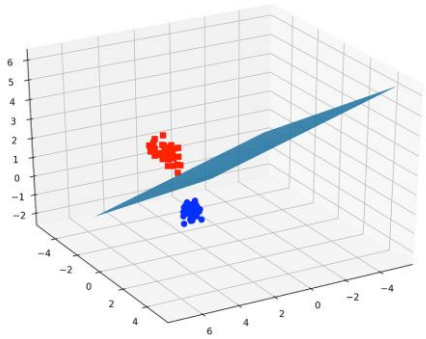
Obrázek 16 - hranice oddělující dvě kategorie

Na výše uvedeném obrázku vstupní data mají dva vysvětlující parametry x_1 a x_2. Prozatím budeme SVM vysvětlovat na datech, která obsahují jednu nebo dvě proměnné, aby šel princip dobře zobrazovat. Vysvětlujících parametrů může být více - pak se přesuneme do vícerozměrného prostoru.

Dále vstupní data musí obsahovat informaci o přiřazení do kategorie, protože SVM je na rozdíl od K-means algoritmus s učitelem. Hodnoty určující kategorie musí být diskrétní čísla, například 1 a -1.

Zatím si zavedeme další omezení, že tvar hranice je definovaný lineární funkcí. Později toto omezení zrušíme a to, ve chvíli, kdy si budeme vysvětlovat různé kernely.

V případě lineární funkce u 2D dat hledáme 1D hranici. Pro vstupní data se třemi vstupními proměnnými data zobrazujeme ve 3D prostoru a hranice bude mít tvar roviny. Naopak pokud budeme mít jednorozměrná data, tak dělící hranicí bude bezrozměrný bod.



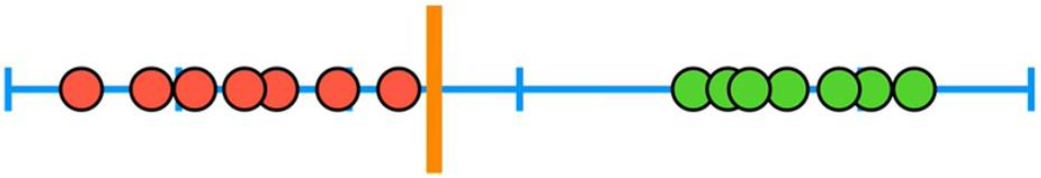
Obrázek 17 - hranice oddělující dvě kategorie v prostoru

Obecně SVM hledá **nadrovinu H** (hyperrovinu, hyperplane), která má rozměr o jednu menší, než je rozměr vstupních dat. V SVM se nadrovina H nazývá **Support vector classifier** nebo Decision boundery.

Hledaných nadrovin může být nekonečně mnoho. Zkusme tedy najít nějakou podmínku, která nám vybere optimální nadrovinu.

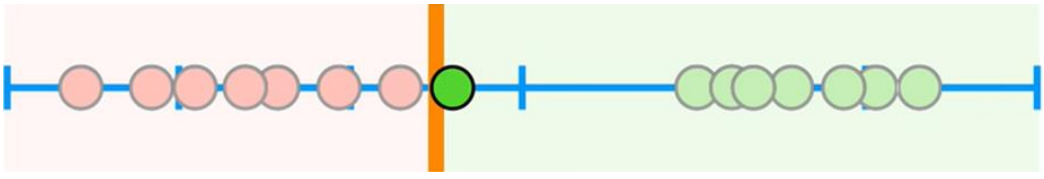
Mějme množinu pozorování, která pro jednoduchost má pouze jeden rozměr.

Hranic je nekonečně mnoho, ale aby hranice dávala smysl, měla by ležet mezi červenými a zelenými body. Například takto.

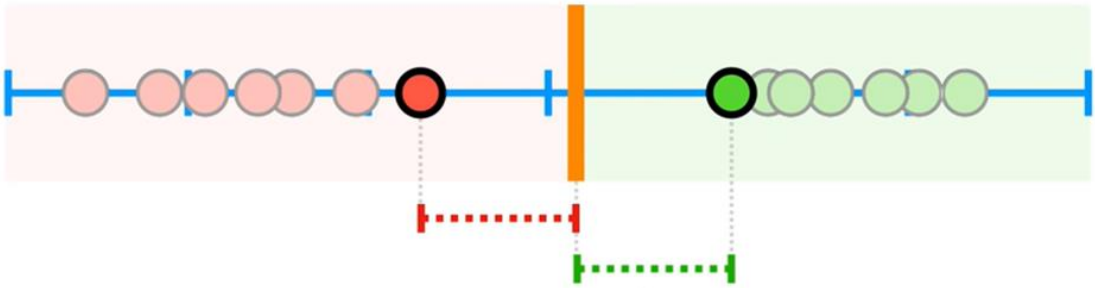


Při této hranici nové pozorování spadne do správné červené kategorie.

Ale může se vyskytnout i jiné pozorování. V tomto modelu bude zařazené do zelené kategorie, i když je daleko blíže červeným pozorováním. Náš model tedy není přesný.



Můžeme přidat omezující podmínku, že hranice musí být stejně vzdálená od nejbližších výskytů z obou kategorií. Nově nastavená hranice bude lépe oddělovat obě kategorie. Krajní body se nazývají **support vektory.** Vzdálenost mezi nimi se nazývá **margin M**. Jednoduše řečeno SVM hledá takový Support vector classifier H, který maximalizuje vzdálenost M mezi support vektory. Vzniká tak co největší „rezerva“ pro oddělení kategorií.



5.1 Lineární funkce

Při použití lineární funkce bude rovnice nadroviny H následující: $w \cdot x + b = 0$. Kde x je matice vstupních proměnných x a w je vektor vah w. Parametr b je nazývaný intercept.

Pro níže zobrazená dvourozměrná data bude rozepsaná rovnice vypadat následovně:

$$w_1 \cdot x_1 + w_2 \cdot x_2 + b = 0$$

Rovnice přerušovaných přímek, na kterých leží Support vectory, budou mít následující podobu:

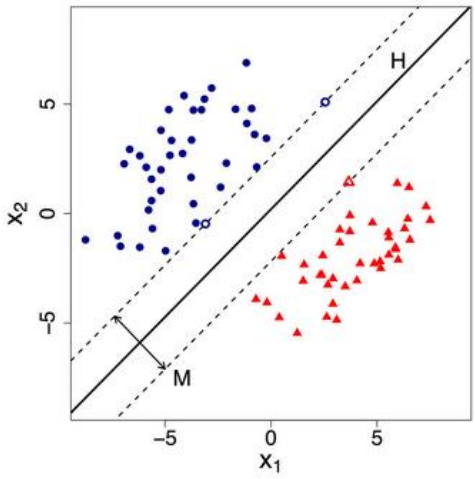
$$w_1 \cdot x_1 + w_2 \cdot x_2 + b = 1$$

$$w_1 \cdot x_1 + w_2 \cdot x_2 + b = -1$$

Pro pozorování, které leží mimo tyto přerušované přímky, platí:

$$w_1 \cdot x_1 + w_2 \cdot x_2 + b > 1$$

$$w_1 \cdot x_1 + w_2 \cdot x_2 + b < -1$$



Obrázek 18 - hranice oddělující dvě kategorie

Než budeme pokračovat dále, nejprve si připomeneme pojmy z lineární algebry: skalární součin vektorů, normu vektorů a normovaný vektor.

Mějme vektory $a = (1, 2, 0)$ a $b = (2, -1, 3)$.

Skalární součin vektorů budeme počítat jako $< a, b > = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$. Pro naše vektory je skalární součin $< a, b > = 1 \cdot 2 + 2 \cdot -1 + 0 \cdot 3 = 2 - 2 + 0 = 0$

Norma vektorů budeme počítat jako odmocninu ze skalárního součinu vektoru sama se sebou. $\|a\| = \sqrt{< a, a >}$. Pro náš vektor je to tedy $\|a\| = \sqrt{1^2 + 2^2 + 0^2} = \sqrt{5}$.

Normovaný vektor spočítáme jako $a' = \frac{a}{\|a\|}$. Pro vektor a je normovaný vektor $a' = (\frac{1}{\sqrt{5}}, \frac{2}{\sqrt{5}}, 0)$.

SVM algoritmus se snaží najít takové parametry w a b, které maximalizují velikost margin M.

Mějme bod x, který leží na nadrovině H a má tedy rovnici $w \cdot x + b = 0$. O jakou vzdálenost by se tento bod měl posunout, aby se stal suport vector a ležel na čárkovanou přímku, která je definovaná vzdáleností margin?

V první řadě si definujeme normovaný vektor w'=$\frac{w}{\|w\|}$, který směřuje stejným směrem jako vektor w. A hledáme takový jeho násobek, abychom bod x z nadroviny H přesunuli na čárkovanou přímku. Bod x budeme posouvat k-krát ve směru normovaného vektoru w'.

$$w(x + k\frac{w}{\|w\|}) + b = 1$$

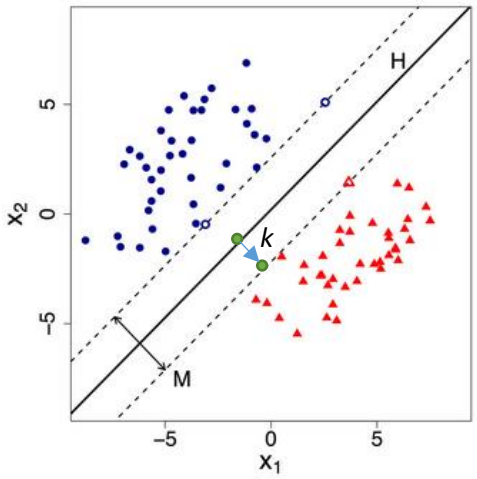
$$w \cdot x + k\frac{< w \cdot w >}{\|w\|} + b = 1$$

Protože $w \cdot x + b = 0$, pak lze rovnice upravit na

$$k\frac{< w \cdot w >}{\|w\|} = 1$$

$$k = \frac{1}{\|w\|}$$

$$M = \frac{2}{\|w\|}$$



Obrázek 19 - Posun bodu x

Cílem optimalizační úlohy je maximalizovat M, jinými slovy maximalizovat $\frac{2}{\|w\|}$, respektive minimalizovat $\|w\|$.

Pro maximalizační úlohy je třeba definovat omezující podmínky.

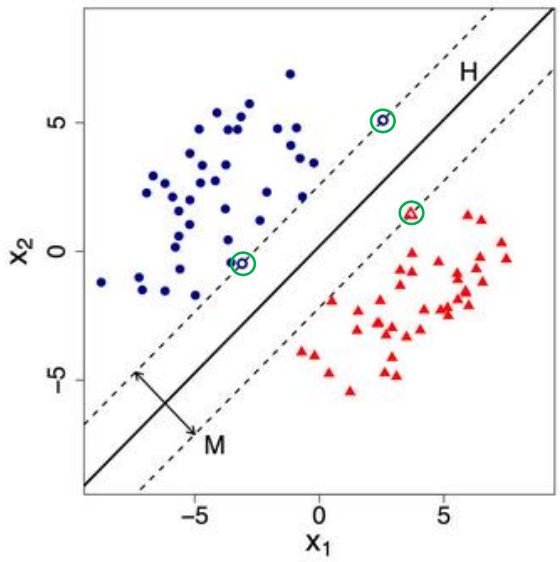
Pokud pozorování y_i patří do kategorie 1, pak $w \cdot x_i + b \geq 1$

Pokud pozorování y_i patří do kategorie -1, pak $w \cdot x_i + b \leq -1$

Tyto dvě podmínky lze dohromady vyjádřit pomocí následujícího vzorce.

$$y_i(w \cdot x + b) \geq 1$$

Pokud support vektor, bod ležící na čárkované hranici, dosadíme do rovnice $w \cdot x + b$ získáme hodnotu rovnající se hodnotě kategorie. Jsou to pozorování, která plně definují margin. Po natrénování jsou všechny ostatní pozorování nepodstatná a mohou být zapomenuta.

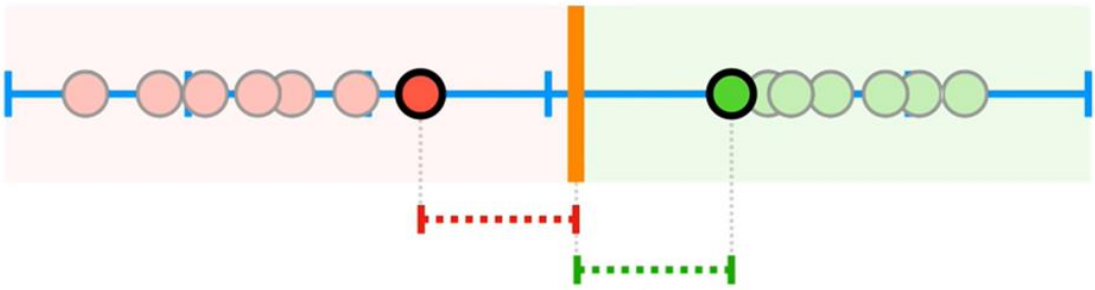


Obrázek 20 - Support vectors

Nalezení optimálních hodnot w a b při cíli minimalizovat $\|w\|$ za podmínky $Y_i(w \cdot x + b) \geq 1$ lze vyřešit pomocí různých optimalizačních algoritmů.

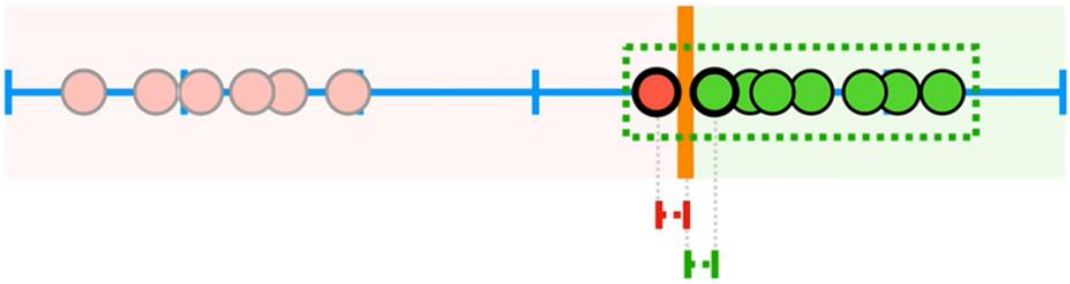
5.2 Soft margin

Doposud jsme nadrovinu H hledali jako střed vzdálenosti mezi nejbližšími body dvou kategorií.

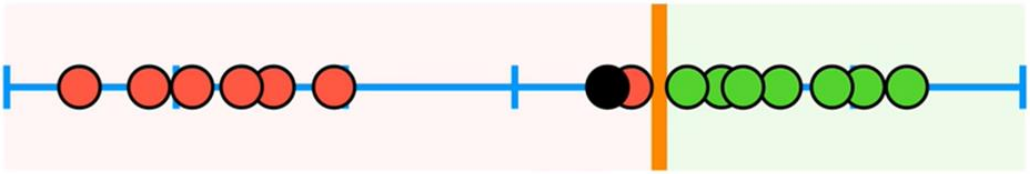


Tento postup je ale velmi náchylný na výběr trénovacích dat, která mohou obsahovat zkreslené zašuměné hodnoty. Mějme například taková trénovací data, kde jedno pozorování je výrazně vychýlené od ostatních pozorování spadajících do stejné kategorie.

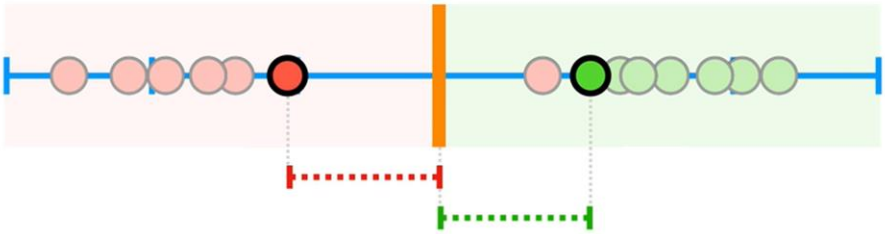
Pokud budeme postupovat dosavadním způsobem, tak nadrovinu stanovíme takto:



Velikost margin M bude velmi malá. Pokud budeme mít pozorování, které by mělo spadat do zelené kategorie, nově díky jednomu vychýlenému červenému bodu bude určeno chybně. Model je velmi citlivý na správně vybraná trénovací data.

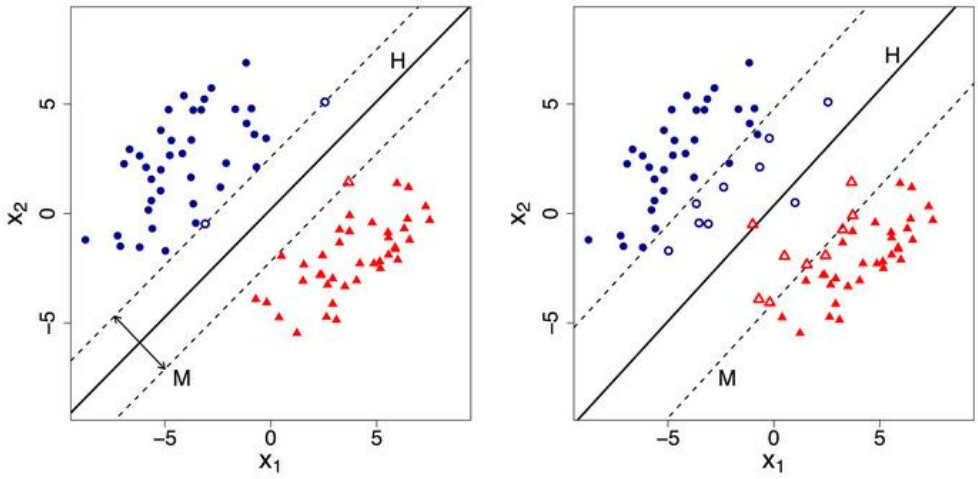


Aby SVM model nebyl náchylný na tato pozorování, používají se soft margin, aby bylo zamezeno problému overfitting. V tomto případě povolíme, aby v rámci margin mohlo být i několik správných nebo nesprávných pozorování. Jinými slovy umožníme rozšířit margin, aby obsahoval určitý podíl pozorování, která jsou blíže nadrovině H, než jsou support vektory. Za každé takové pozorování připočítáváme trestné body a čím je pozorování více vzdálené od hranic, tím jsou trestné body vyšší.



V případě měkké hranice se nejen maximalizuje vzdálenost M, ale minimalizuje se celková chyba T, jejíž velikost může být omezena vstupním parametrem modelu.

Někdy je optimalizační problém SVM se soft margin přepracován na problém, kde se místo T používá k vážení chyby penalizace C, která představuje kompromis mezi zvýšením vzdálenosti hranice a snížením chybné klasifikace v trénovacích datech. Celková chyba T a penalizační parametr C jsou nepřímo úměrné optimální hranici. Velké penalizace C (malé přípustné chyby T) vedou k menší vzdálenosti hranice od dělící nadroviny a naopak.



Obrázek 21 - hard a soft margin

Hinge loss je ztrátová funkce, která se používá při trénování klasifikačních modelů. Ztrátová funkce hinge loss je definována jako:

$$J(y_i) = \max(0, 1 - y_i(w \cdot x + b))$$

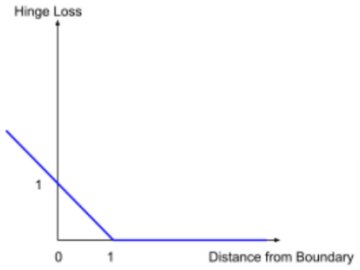
Například pro správně umístěný modrý bod nad horní margin bude předvídaná hodnota větší než 1. $y_i(w \cdot x + b) = 1 \cdot (\check{c}\acute{i}slo\ v\check{e}t\check{s}\acute{i}\ ne\check{z}\ 1)$. Modrá část výrazu je správná hodnota a zelená část výrazu je předpovězená hodnota. Hodnota nákladové funkce bude $\max\big(0, 1 - (1 \cdot 2)\big) = \max(0, -1) = 0$.

Pro správně umístěný červený trojúhelníček pod dolní margin bude předvídaná hodnota menší než -1. $y_i(w \cdot x + b) = -1 \cdot (\check{c}\acute{i}slo\ men\check{s}\acute{i}\ ne\check{z} - 1)$. Hodnota nákladové funkce bude například $\max\big(0, 1 - (-1 \cdot -2)\big) = \max(0, -1) = 0$

Jinak tomu bude u špatně umístěných bodů. Například červený trojúhelníček nad H. Předvídaná hodnota bude větší než -1. $y_i(w \cdot x + b) = -1 \cdot (\check{c}\acute{i}slo\ v\check{e}t\check{s}\acute{i}\ ne\check{z} - 1)$. Hodnota nákladové funkce bude například $\max\big(0, 1 - (-1 \cdot -1{,}1)\big) = \max(0, 1{,}5) = 1{,}1$. Pro špatně určené body hodnota nákladové funkce roste lineárně se vzdáleností od správného umístění.

Pro body, které jsou sice správně klasifikované, ale již zasahují do margin, bude hodnota ztrátové funkce v intervalu (0, 1). Sice predikce proběhla správně, ale není zde dostatečná rezerva, proto to optimalizačnímu algoritmu musíme předat ve formě menší pokuty.

Tvar nákladové funkce Hinge loss je následující:



Obrázek 22 - Hinge loss

SVM se soft margin minimalizuje dvě části. Průměrnou hodnotu nákladové funkce Hinge loss pro každý bod v tréninkových datech, jinými slovy trest za špatně predikované hodnoty. A zároveň jak již bylo vysvětleno při optimalizaci hard margin, minimalizuje $\|w\|$, aby tím maximalizoval velikost margin. Pozitivní parametr λ umožňuje ladit, zda a jak moc preferujeme maximalizaci velikosti margin nebo minimalizaci chybných předpovědí.

$$Min_{w,b} = \frac{1}{N}\sum_{i=1}^{N} \max(0, 1 - y_i(w \cdot x + b)) + \lambda\|w\|^2$$

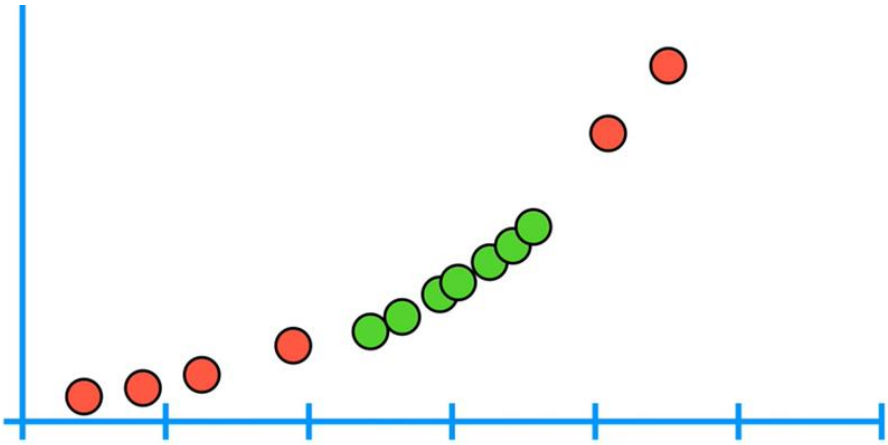
5.3 Polynominální kernel

Doposud jsme pracovali s daty, která šla oddělit pomocí lineární funkce, jinými slovy lineárního kernelu. Support vector machine umožňuje klasifikovat data, která nejdou lineárně oddělit.

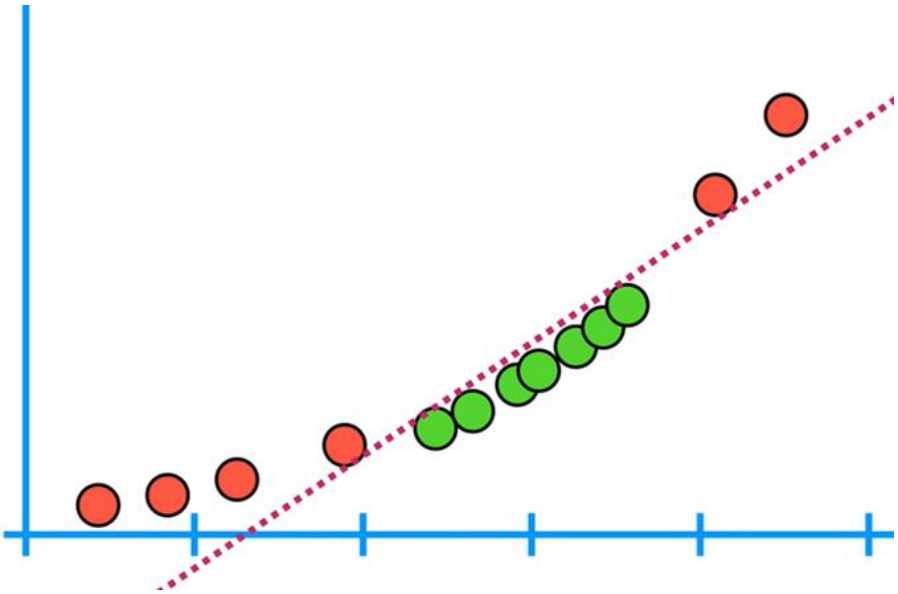
Mějme například data reprezentující účinek léku v závislosti na velikosti dávky. Pokud je dávka příliš malá nebo příliš velká, lék nebude účinný. Našim úkolem je definovat hranice vhodné dávky. Jak je vidět z obrázku, pozorování nejsme schopni v 1D rozdělit jednou lineární funkcí.

V tento okamžik můžeme použít jiné kernely, jinými slovy jiné funkce. Velmi často se u SVM používá polynomiální, RBF nebo sigmoid. Nyní se zaměříme na polynomiální kernel a vysvětlíme si na něm pojem „kernel trik“.

Pomocí kernelu zvýšíme dimenzitu dat. Výše znázorněná data mají pouze jeden rozměr x. Zvolením správného kernelu jim vypočítáme druhou dimenzi. Na následujícím obrázku pro výpočet druhé dimenze y použijeme vzorec $y = x^2$



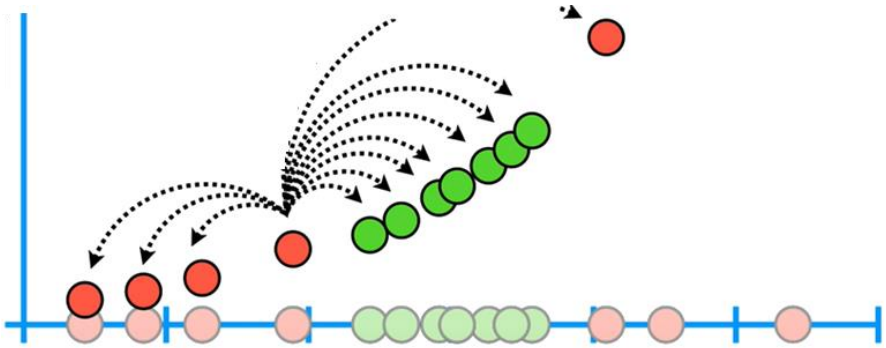
V tuto chvíli jsme již pomocí SVM schopni definovat nadrovinu H, která odděluje dvě kategorie dat.



Pokud budeme chtít na základě tohoto modelu klasifikovat nové pozorování stačí hodnotu x vynásobit kernelem a zjistit hodnotu y. Podle kombinace hodnot x, y zjistíme polohu vůči nadrovině a to nám určí kategorii pozorování.

Polynomiální kernel má podobu $(a \cdot \mathrm{b} + b)^d$. Parametrem tohoto kernelu je d, což stupeň polynomu. Při hodnotě d=1 budeme provádět porovnání vztahů mezi každou dvojicí pozorování v 1D. A tato porovnání budou použita pro určení Support vector classifier (nadroviny H).

Při hodnotě d=2 bude používat porovnání ve 2D. A porovnání budou použita pro definici nadroviny v podobě přímky. Při hodnotě d=3 se budou provádět porovnání ve 3D.



Porovnání dvou pozorování a, b lze počítat následovně. Je ukázán výpočet pro parametry $b = \frac{1}{2}, d = 2$.

$$(a \cdot b + \frac{1}{2})^2 = \left(a \cdot b + \frac{1}{2}\right)\left(a \cdot b + \frac{1}{2}\right) = ab + a^2b^2 + \frac{1}{4}$$

Výsledek můžeme převést na skalární součin vektorů, který nám dává souřadnice pozorování ve vyšších dimenzích.

$$ab + a^2b^2 + \frac{1}{4} = (a, a^2, \frac{1}{2})(b, b^2, \frac{1}{2})$$

Na ose x jsou to hodnoty a, b. Na ose y jsou to hodnoty a^2, b^2. Protože pro oba body je hodnota na ose z stejná, můžeme ji ignorovat. Souřadnice bodu *a* ve vyšší dimenzi je (a, a^2) a bodu *b* (b, b^2).

Nyní si do vzorce můžeme dosadit dva body, například a=9, b=14.

$$(a \cdot b + \frac{1}{2})^2 = (9 \cdot 14 + \frac{1}{2})^2 = 16\,002{,}25$$

Pokud zvolíme jiné parametry kernelu $b = 1, d = 2$ získáme trochu jiný polynom.

$$(a \cdot b + 1)^2 = (a \cdot b + 1)(a \cdot b + 1) = 2ab + a^2b^2 + 1$$

Skalární součin bude následující:

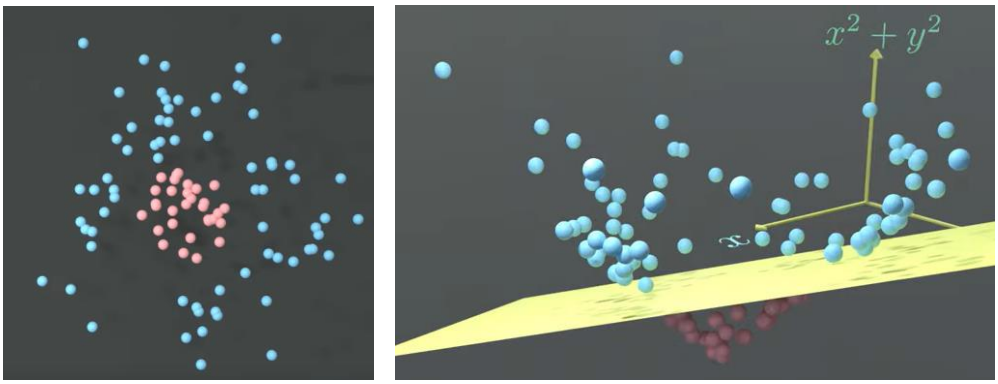
$$2ab + a^2b^2 + 1 = (\sqrt{2a}, a^2, 1)(\sqrt{2b}, b^2, 1)$$

Kernel trik spočívá v tom, že data pomocí zvoleného kernelu počítáme, jako by měla vyšší dimenzionalitu, ale ve skutečnosti je netransformujeme.

Dosud jsme měli jednorozměrná data a převáděli jsme je dvourozměrných dat. Toto samozřejmě můžeme i s vícerozměrnými daty. Jak funguje kernel trik může pomoci pochopit následující analogie. Na prostěradle máme kuličky dvou barev. Červené kuličky jsou uprostřed a jsou ze všech stran obklopeny modrými kuličkami. Na prostěradle nelze najít žádnou přímku, která by kuličky správně oddělila.

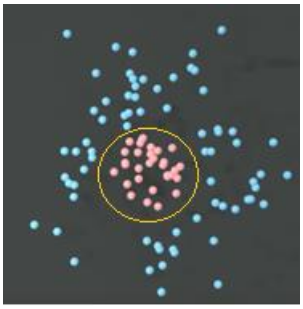
Pomůžeme si tedy kernel trikem, v analogii prostěradlem trhneme a kuličky vyhodíme do vzduchu a tím získají třetí dimenzi. Novou dimenzi z můžeme přidat například pomocí vzorce $z = x^2 + y^2$. Modré kuličky vylétnou výše do vzduchu a červené níže. V naší analogii si můžeme představit, že jsou červené kuličky těžší.

Ve chvíli, kdy jsou kuličky ve vzduchu, je ve vhodný okamžik oddělíme pomocí listu papíru. Modré kuličky spadnou na žlutý papír a červené kuličky spadnou na prostěradlo.



Obrázek 23 - polynominální kernel ve 3D

Díky zvolené kernelové funkci se kuličky rozlétnou do podoby paraboloidu. Papír nám vlastně provede řez tímto paraboloidem. Když řez promítneme zpátky do 2D, získáme hranici, která nám odděluje dvě kategorie kuliček.



Obrázek 24 - polynominální kernel ve 3D

5.4 RBF kernel

Další často používaný kernel se nazývá Radial Basis Function. RBF pracuje v nekonečném počtu dimenzí, takže se bude špatně vizualizovat.

RBF se chová velmi podobně jako jiný klasifikační algoritmus K-NN (K Nearest Neighbours).

Na nové pozorování mají největší vliv nejbližší pozorování. Naopak vzdálené pozorování na něj má daleko menší vliv. Proto RBF pro klasifikaci použije nejbližší data.

Rovnice tohoto kernelu je následující a určuje jaký vliv má pozorování na jiné pozorování. Vzdálenost mezi pozorováními a – b je umocněna. Vstupní parametr γ, který je na počátku pevně určen, škáluje vliv podle umocněné vzdálenosti.

$$e^{-\gamma(a-b)^2}$$

Vliv bodu b=4 na bod a=2,5 při různých hodnotách γ bude následující:

$$\gamma = 1;\ e^{-(2,5-4)^2} = e^{-(-1,5)^2} = 0,11$$

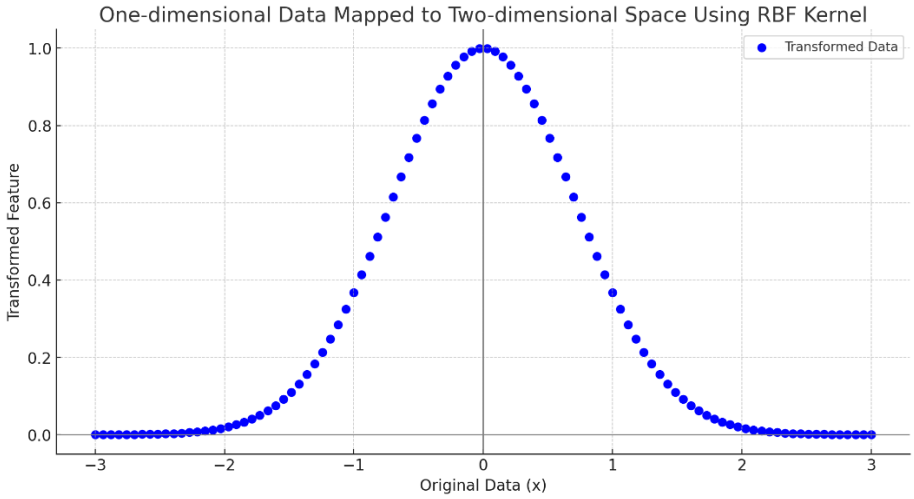
$$\gamma = 2;\ e^{-2(2,5-4)^2} = e^{-2(-1,5)^2} = 0,01$$

$$\gamma = 1;\ e^{-(2,5-16)^2} = e^{-(-13,5)^2} = e^{-182,25} \cong 0$$

Volba γ je kritická, protože ovlivňuje komplexnost modelu. Malé γ znamená, že kernel bude mít dlouhý dosah a může způsobit přílišnou generalizaci, zatímco velké γ způsobí, že kernel bude reagovat pouze na body blízké referenčnímu bodu, což může vést k overfitting.

RBF kernel transformuje původní prostor dat do nelineárního prostoru, což umožňuje SVM nalézt nelineární rozhodovací hranice.

Na následujícím grafu je vidět, jak jsou jednorozměrná rovnoměrně rozdělená data transformována do dvourozměrného prostoru pomocí RBF kernelu. Horizontální osa představuje původní data (x), zatímco vertikální osa představuje transformovaná data získaná pomocí RBF kernelu s pevným referenčním bodem (v tomto případě v bodě 0). Jak můžeme vidět, původní data byla transformována tak, aby vytvářela nový rozměr, který zachycuje nelineární vztahy, což umožňuje efektivnější oddělení v prostoru s vyššími rozměry.



Obrázek 25 - RBF kernel

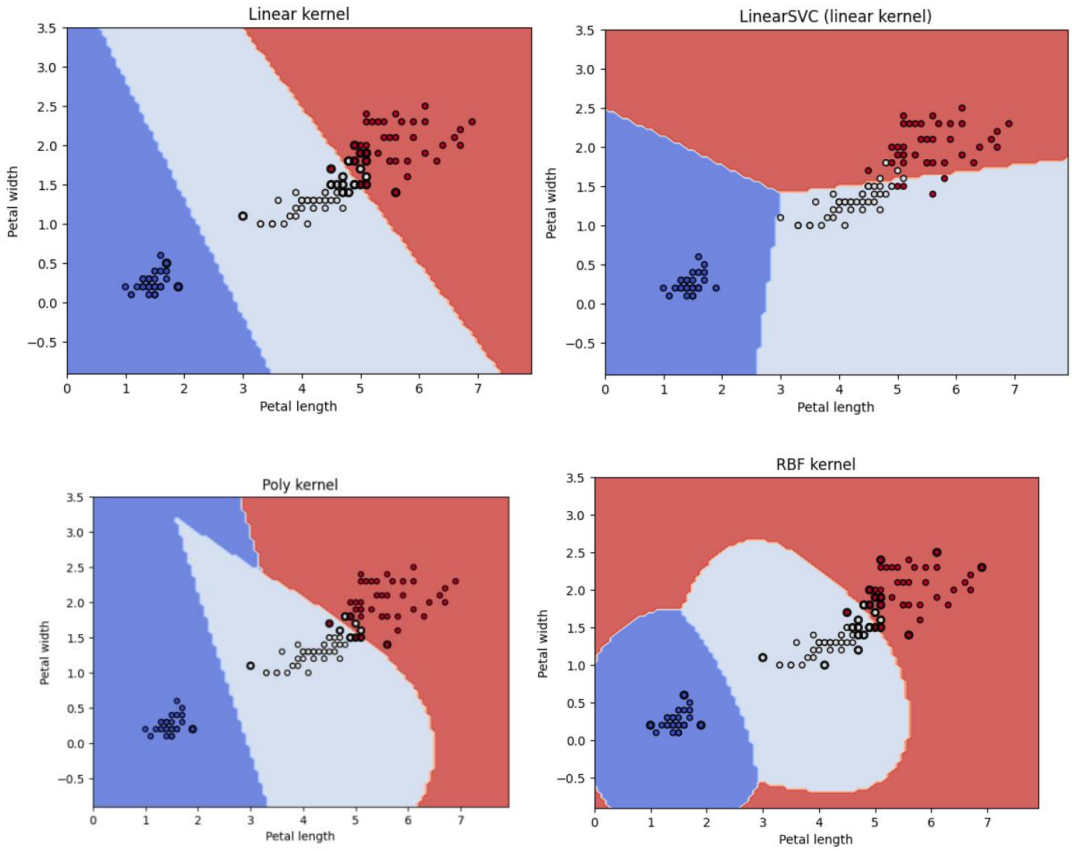
5.5 Ladění SVM

Jak jsme si ukázali, tak Support Vector Machine může mít mnoho parametrů. Od volby typu kernelu, až po různé parametry daných kernelů.

Například u polynomiálního kernelu můžeme volit stupeň polynomu d, parametr b, λ atd. Tyto parametry ovlivňují, jak bude probíhat optimalizace a jak bude probíhat promítání dat do vyšších dimenzí. Ne každé promítání dat povede ke stavu, kdy budeme schopni data rozdělit pomocí nadroviny. Respektive každá kombinace bude mít různou přesnost modelu.

V praxi je tedy potřeba vyzkoušet různé kombinace kernelů a jejich parametrů a porovnat výsledné modely a vybrat z nich, který je nejlepší.

Na následujících obrázcích jsou zobrazeny SVM modely s různými kernely pro dataset iris.



Obrázek 26 - SVM modely s různými kernely

5.6 Vlastnosti algoritmu

5.6.1 Výhody

- Všestrannost – použití na různých typech dat (klasifikace textu, detekce, ověřování a rozpoznávání obličejů, klasifikace nevyžádané pošty, analýzy úvěrového hodnocení a klasifikace rakoviny a cukrovky, …)

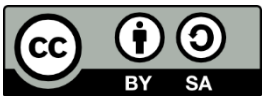
- Schopnost pracovat s daty v prostoru vyšší dimenze
- Schopnost zpracovat nelineární data
- Schopnost vyhýbat se problémům s přetrénováním
- Odolnosti vůči datům, která jsou daleko od nadroviny a jsou efektivní, protože jsou založena pouze na podpůrných vektorech uvnitř nadroviny.
- Efektivní pro data s malým počtem tréninkových příkladů

5.6.2 Nevýhody

- Algoritmus může být výpočetně a paměťově náročný při větším objemu dat.
- Algoritmus je primárně určený pro klasifikaci dvou proměnných. Nové implementace umožňují klasifikovat více tříd pomocí přístupu jedna proti jedné nebo jedna proti všem. To opět zvyšuje výpočetní náročnost.
- SVM se mohou jevit jako černé skříňky, protože součástí odhadu není konečná funkční forma nebo tabulka koeficientů pro různé prediktory.

UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vytvořeno v rámci projektu **Digitalizace studijních Agend, Nové TechnologiE, systémy a přístupy k výuce na UPCE**, reg. č. NPO_UPCE_MSMT-16591/2022.



Financováno Evropskou unií
NextGenerationEU

Národní plán obnovy

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Machine learning & AI

Stavový prostor a plánování

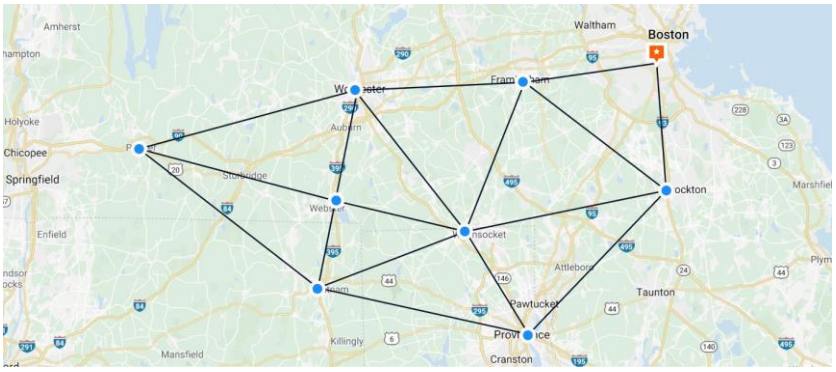
Ing. Martin Pozdílek, Ph.D.

1 Plánování a prohledávání stavového prostoru a

Dalším typem úloh, které se vedle regrese a klasifikace řeší pomocí strojového učení, jsou optimalizační a plánovací úlohy. Jedná se o úlohy, které mohou mít více různých řešení a cílem je nalézt řešení, které splňuje omezující podmínky a případně maximalizuje nějakou funkci. Existuje celá množina algoritmů, které jsou vhodné pro tento typ úloh a některými z nich se budeme zabývat v následujících kapitolách.

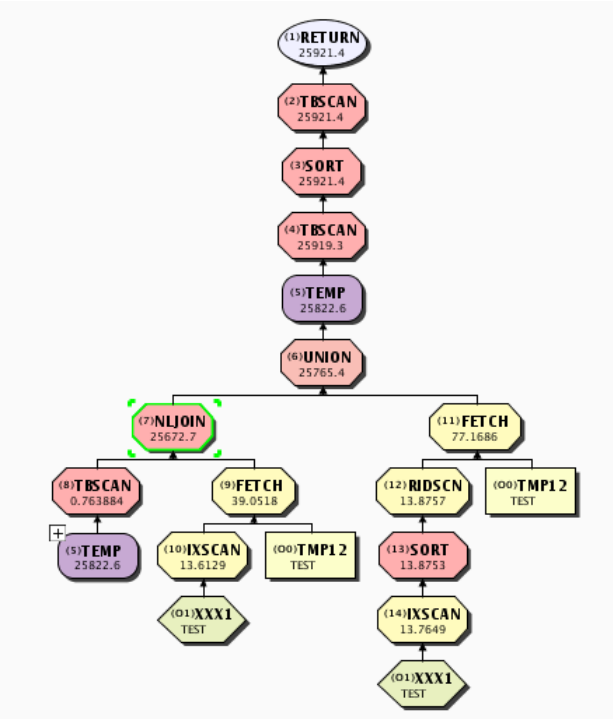
Pro představu si můžeme uvést několik příkladů.

Problém obchodního cestujícího (travelling salesman problem – TSP). Existuje několik měst, a mezi nimi silnice o daných délkách. Úkolem je najít nejkratší cestu procházející všemi městy a vracející se zpět do výchozího města. Matematická formulace: V ohodnoceném úplném grafu najděte nejkratší hamiltonovskou kružnici.



Obrázek 1 - problém obchodního cestujícího

Vyhledání optimálního přístupového plánu v relačních databázích. Relační databáze musí najít správné pořadí operací s daty, aby vrátila výsledek, který odpovídá SQL dotazu. Způsobů, jak zpracovat dotaz je velmi mnoho, ale pokud má být výsledek vrácen co nejrychleji, je třeba do výpočtu zahrnout informace jako odhadovaný objem dat, rozložení dat, přítomnost indexů, rychlost disků, velikost vyrovnávacích pamětí apod.



Obrázek 2 - přístupový plán v relační databázi

Hraní deskových her, řešení hlavolamů apod. Velmi často se strojové učení používá pro hraní her. Již v minulosti vznikaly algoritmy pro dámu, šachy a další klasické deskové hry. Jak již bylo zmíněno, v roce 1997 IBM Deep Blue porazil šachového velmistra Kasparova. Z posledních úspěchů připomeňme umělou inteligenci AlphaGo, která porazila velmistra v Go. V každém tahu partie deskové hry lze zpravidla zvolit více možných tahů. Cílem algoritmu je volba tahu, který s největší pravděpodobností povede k vítězství.

Nemusí se jednat pouze o deskové hry, ale mnoho dnešních počítačových her řeší problematiku inteligentního chování NPC (non-playable character).

2 Teorie grafů

Při úlohách prohledávání stavového prostoru se velmi často používá teorie grafů. Protože teorií grafů se zabývá samostatný povinný předmět, v této kapitole se připomeneme základní pojmy a definice, které budeme dále používat.

Graf je reprezentace množiny objektů a jejich vzájemných vztahů. Jedná se o strukturu, která lze znázornit pomocí grafu. **Vrchol** zpravidla reprezentuje nějaký objekt a **hrana** reprezentuje vztah mezi dvěma objekty.

Definice: Graf G je uspořádaná dvojice G = (V, E). Kde V je neprázdná množina vrcholů (vertices, nodes, points) a E je množina hran (edge, links, lines). Existuje zobrazení ε, které každé hraně $e \in E$

přiřazuje uspořádanou nebo neuspořádanou dvojici vrcholů náležících do množiny V. Graf má n vrcholů a m hran.

$$G = (V, E, \varepsilon)$$

$$V = \{v_1, v_2, \dots v_n\}$$

$$E = \{e_1, e_2, \dots e_n\}$$

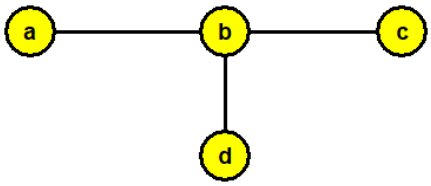
$$\varepsilon: E \to V \times V$$

Ve zjednodušeném zápisu lze množinu E určit například jako

$$E = \{\{v_1, v_2\}, \{v_1, v_3\}, \dots \{v_l, v_m\}\}$$

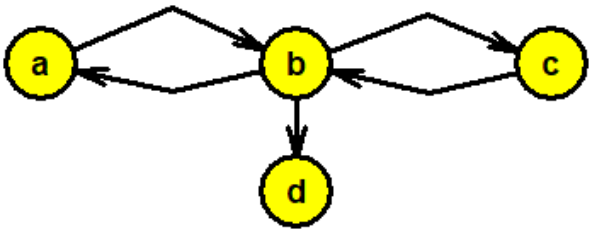
Neorientovaný graf, je takový, kdy je hrana tvořena neuspořádanou dvojicí vrcholů. Jinými slovy, kdy nezáleží na jejich pořadí.

$V = \{a, b, c, d\}$
$E = \{\{a, b\}, \{b, c\}, \{b, d\}\}$



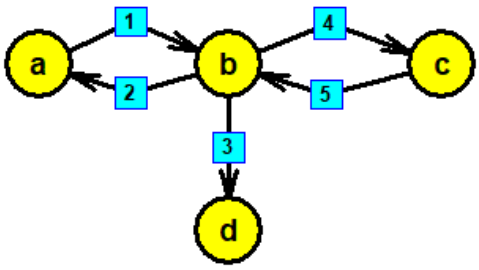
Orientovaný graf je takový, kdy je hrana tvořena uspořádanou množinou vrcholů. Záleží tedy na jejich pořadí. U orientovaných grafů můžeme rozlišovat mezi jednosměrnými a obousměrnými grafy.

$V = \{a, b, c, d\}$
$E = \{(a, b), (b, a), (b, c), (c, b), (b, d)\}$

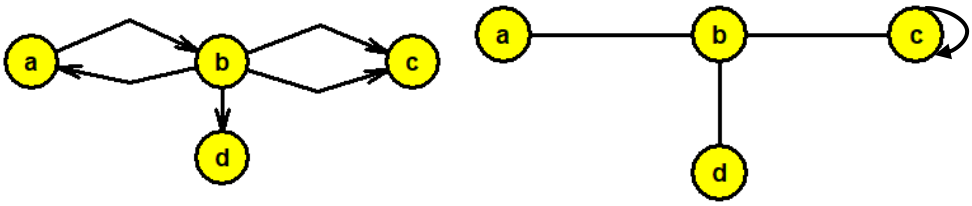


Ohodnocený graf je takový, ke kterému je přiřazena funkce w, která každé hraně přiřazuje reálné číslo. $G = (V, E, \varepsilon, w)$. Ohodnocené grafy mohou být orientované i neorientované. U orientovaných grafů mohou mít různé směry různé ohodnocení.

$V = \{a, b, c, d\}$
$E = \{(a, b), (b, a), (b, c), (c, b), (b, d)\}$
$w\big((a, b)\big) = 1$
$w\big((b, a)\big) = 2$
$w\big((b, c)\big) = 4$
$w\big((c, b)\big) = 5$
$w\big((b, d)\big) = 3$



Smyčka je hrana, která má stejný počáteční a koncový vrchol. Graf může obsahovat i **násobnou** (rovnoběžnou) hranu.



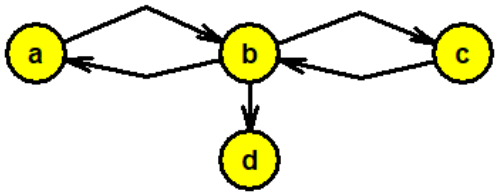
Prostý graf neobsahuje rovnoběžnou hranu, ale může obsahovat smyčku.

Následník vrcholu v_i v grafu G je každý vrchol tohoto grafu, do kterého vede z v_i hrana.

Předchůdce vrcholu v_i v grafu G je každý vrchol tohoto grafu, ze kterého vede do v_i hrana.

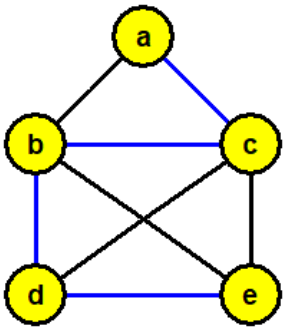
Následník b = {a, c, d}

Předchůdce b = {a, c}



Sled F délky n v grafu G je posloupnost vrcholů a hran, ve kterém každá hrana e_i má koncové vrcholy v_{i-1}, v_i. Ve sledu se může opakovat hrana

F =({a, c}, {c,b}, {b,d}, {d,e}, {e, d})

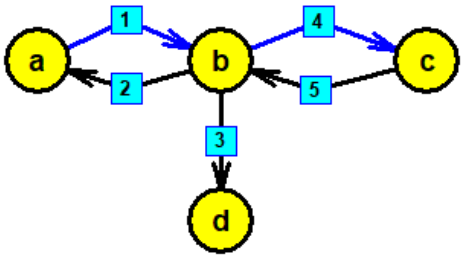


Tah W v grafu G je takový sled, ve kterém se **žádná hrana neopakuje**.

F =({a, c}, {c,b}, {b,d}, {d,e})

Cesta P v grafu G je takový tah, ve kterém se **neopakuje žádný vrchol**. Každá cesta je tahem, ale tah nemusí být cestou.

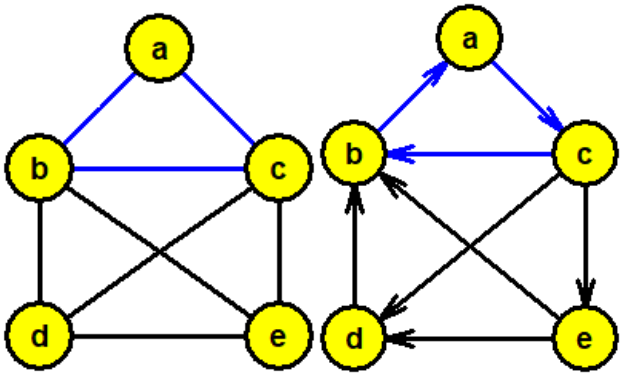
Minimální cesta mezi vrcholy u a v je cesta, jež má ze všech možných cest mezi těmito vrcholy minimální součet ohodnocení hran do cesty zařazených.



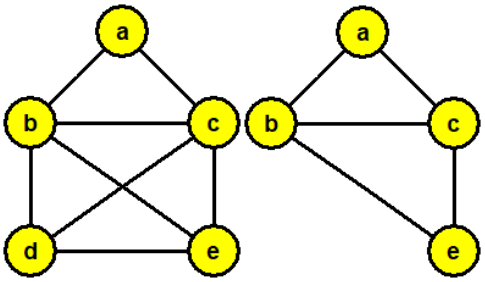
Uzavřený sled, tah, cesta je takový, který má alespoň jednu hranu a počáteční a koncový vrchol splývají.

Kružnice je uzavřená neorientovaná cesta. $P = \{\{a, b\}, \{b, c\}, \{c, a\}\}$

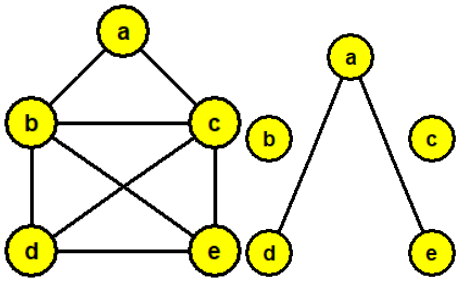
Cyklus je uzavřená orientovaná cesta. $P = \{(a, b\}, (b, c), (c, a)\}$



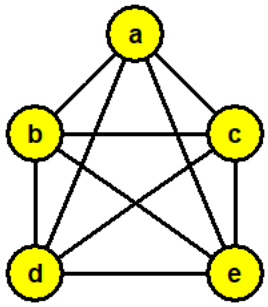
Graf $G' = (V', E')$ nazýváme **podgraf** grafu $G = (V, E)$ jestliže $V' \subset V$ a $E' \subset E$.



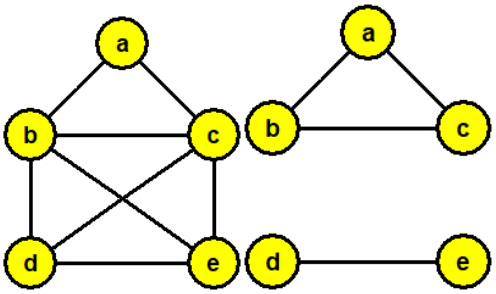
Graf $G' = (V', E')$ nazýváme **doplněk** (komplement) grafu $G = (V, E)$ jestliže pro každé dva vrcholy platí $u, v \in E'$ právě tehdy pokud $u, v \notin E$.



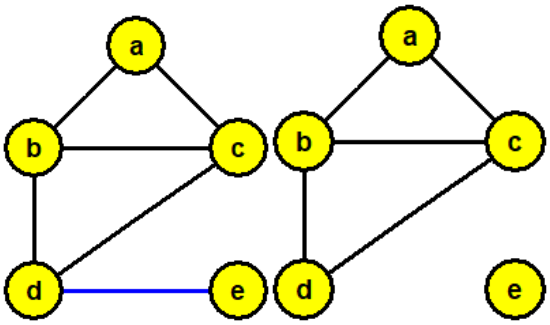
Graf G'' je **úplným grafem**, pokud platí $G'' = (V, E \cup E')$



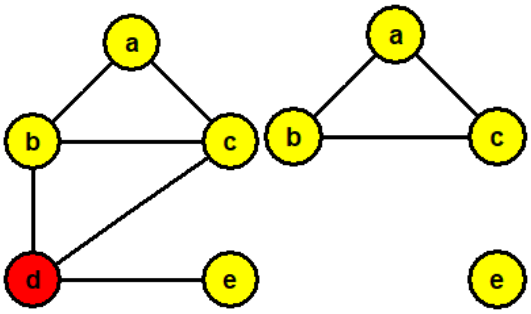
Souvislý graf je takový, kde pro každé dva vrcholy existuje cesta, která je spojuje. Souvislý graf má 1 **komponentu**. **Komponentem grafu** nazveme maximální souvislý podgraf.



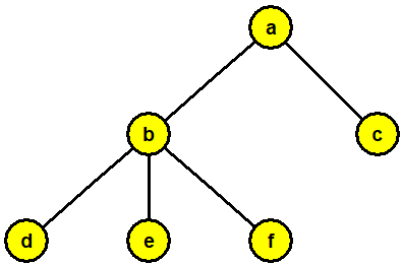
Most je hrana, jejímž odstraněním se zvýší počet komponentů o jednu.



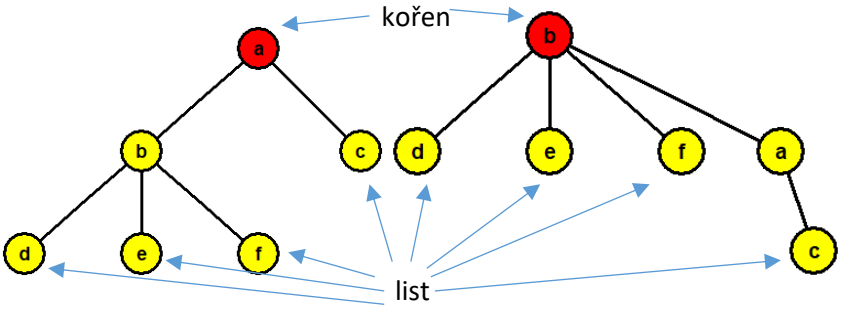
Artikulace je vrchol, jehož odstraněním (spolu s incidujícími hranami) se zvýší počet komponentů alespoň o jeden.



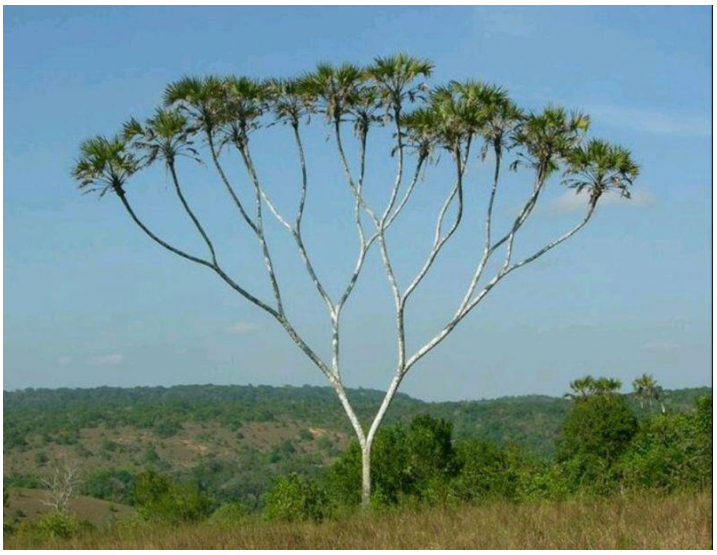
Strom je souvislý graf T, který neobsahuje kružnice.



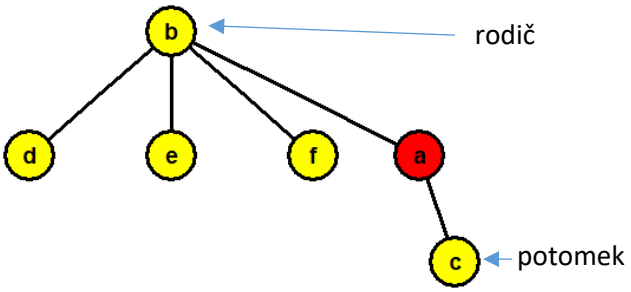
Kořenový strom je takový, u kterého je jeden vrchol označen za **kořen**. Volbou různých kořenů, vznikají různé kořenové stromy. Kořen se zpravidla umísťuje nahoru grafu.



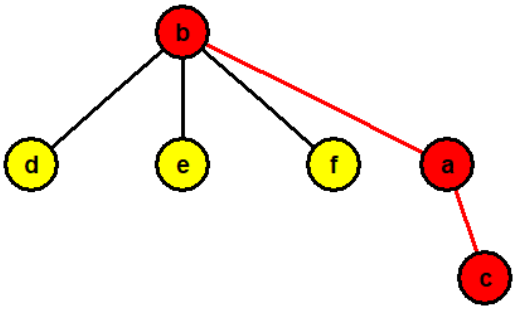
Vrchol, který nemá žádného následníka se nazývá **list**.



Mějme kořenový strom T, v_r s vrcholem $v_i \in V(T)$, kde $v_i \neq v_r$. Pokud je vrchol v_j sousedem vrcholu v_i ve směru ke kořeni, pak v_j je **rodičem** v_i a v_i je **potomkem** v_j.

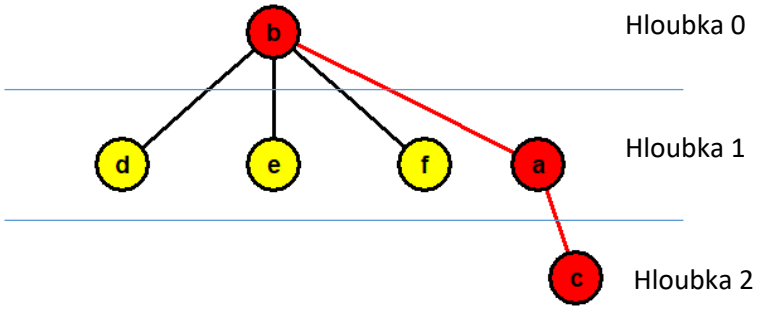


Cesta v kořenovém stromu je posloupnost všech vrcholů a hran od kořene k vrcholu v_i. **Délka** cesty je počet hran, které cesta obsahuje.



Hloubka vrcholu je délka cesty k vrcholu od kořene.

Výška (hloubka) **stromu** je dána maximální hloubkou vrcholů.



3 Stavový prostor

3.1 Koncepční model pro plánování

Pokud řešíme nějaký problém je důležité ho popsat. **Koncepční model** je nástroj, který umožňuje popis problému. Jedná se o abstraktní strukturu, která pomáhá organizaci a definici plánovacího procesu v rámci určitého projektu, úkolu nebo strategického cíle. Tento model slouží k lepšímu porozumění celého plánovacího procesu a umožňuje efektivní komunikaci mezi členy týmu a zúčastněnými stranami.

Pro naši potřebu budeme pro koncepční model používat **přechodový systém**.

3.2 Přechodový systém

Každý systém v každém okamžiku se nachází v určitém **stavu**. Systém může být například určen počty různých stavů a jejich vztahy mezi nimi. Například stav šachové partie můžeme reprezentovat počty různých figurek a jejich polohou.

Dále je systém určen **akcemi**, kterými můžeme měnit stav systému. V případě šachů jsou to pohyby šachových figur.

Systém se nachází v nějakém vnějším prostředí. Z tohoto prostředí mohou na systém působit vnější **události**, které mají vliv na vnitřní stav systému. U jednoduchých deterministických systémů jako jsou šachy často externí události zanedbáváme (zemětřesení, malé dítě, naštvaný hráč, …). U složitějších reálných systémů s nimi o všem často musíme počítat a velmi často je externí událost primárním hybatelem, který mění stav systému.

Poslední částí definice přechodového systému je **přechodová funkce** γ, která definuje, jakým způsobem se změní stav systému, pokud se vyskytne akce nebo událost. V případě šachů si přechodovou funkci můžeme představit jako pravidla, která určují, jak mohou jednotlivé figury táhnout, jak dochází k vyhazování figur, patu nebo matu apod.

Matematicky přechodový systém lze vyjádřit jako $\Sigma = (S, A, E, \lambda)$

- $S = \{s_1, s_2, \dots\}$ - konečná neprázdná množina stavů
- $A = \{a_1, a_2, \dots\}$ - konečná množina akcí
- $E = \{e_1, e_2, \dots\}$ - konečná množina vnějších událostí
- $\gamma(s, (e \vee a)) \rightarrow s'$- přechodová funkce, která při akci nebo události změní stav systému

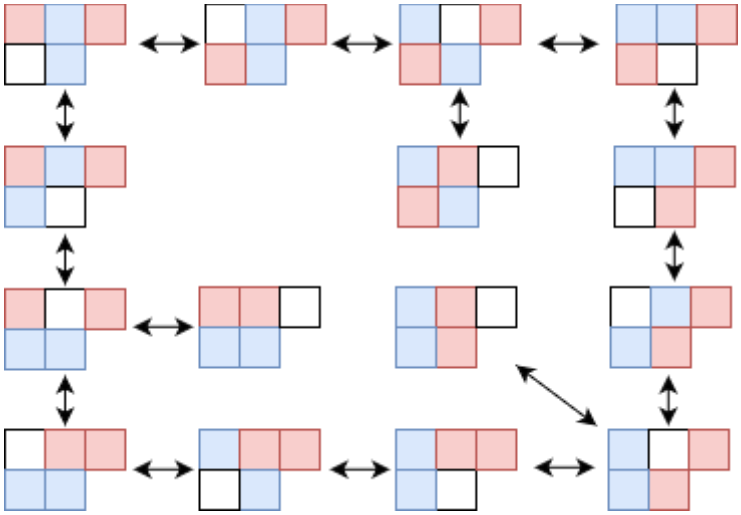
Aplikovatelnost akce na stav s platí, pokud $\gamma(s, a) \neq \emptyset$. Přechodová funkce pro kombinaci daného stavu a akce nesmí vrátit prázdnou množinu.

Důležitou vlastností přechodového systému je, že jej lze vyjádřit jako graf. $(S, A, E, \gamma) \rightarrow (V, E)$. Množina vrcholů odpovídá množině stavů V = S. Hrana $u \in (A \cup E)$ mezi stavy s∈V a $s' \in V$existuje pouze když $s' \in \gamma(s, u)$. Hrana mezi vrcholy bude existovat, pokud přechodová funkce přiřadí akci nebo události ze stavu s nový stav s'.

Například mějme herní pole o pěti políčkách. Na herní ploše jsou umístěny dva červené a dva modré kameny. Kameny lze přemísťovat pouze na volné pole, a to jen horizontálně nebo vertikálně. Hra se nachází vždy v nějakém stavu.

Akcí je přesunutí herního kamene na volné pole. Pokud akci definujeme takto, musíme určit kámen a směr posunu. To představuje 16 akcí. Pokud ale problém definujeme, jako přesun volného pole na jiné obsazené pole, pak počet možných akcí je 4.

Přechodový systém můžeme zobrazit takto. Existuje 15 unikátních stavů, reprezentovaných jako 15 vrcholů. Mezi nimi je i 15 neorientovaných hran.

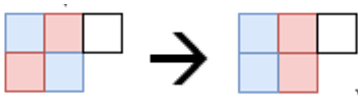


V takto definovaném přechodovém systému, dokážeme určit sled tahů, které musíme provést, abychom se dostali z jednoho stavu do druhého. Tedy provést **naplánování akcí**, abychom vyřešili konkrétní problém.

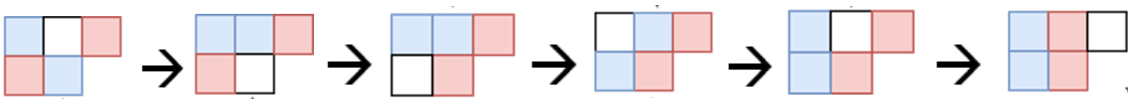
- $S = \{s_1, s_2, \dots s_{15}\}$ - konečná neprázdná množina stavů
- $A = \{a_1, a_2, a_{3,} a_4\}$ - posun prázdného pole: doleva, doprava, nahoru, dolu
- $E = \{\emptyset\}$ - nejsou žádné externí události

- $u_1 = (s_1, \gamma(s_1, a_1) \rightarrow s'$- přesun prázdného pole ve stavu s_1 nahoru
- $u_2 = (s_1, \gamma(s_1, a_2) \rightarrow s'$- přesun prázdného pole ve stavu s_1 dolu
- …

Například úloha zní najít řešení (**plán**) pro převedení systému z **počátečního** stavu do **cílového** stavu.

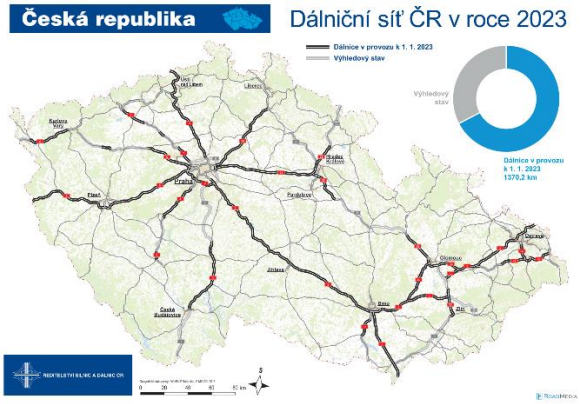


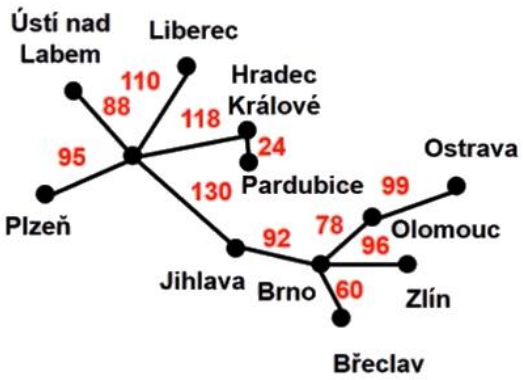
Plán může se může skládat z těchto akcí: doleva, dolů, doleva, nahoru, doprava, doprava.



Je třeba si uvědomit, že reálné systémy jsou komplexní. Stavy i přechody mohou být stochastické a někdy i nespočítatelné. Proto velmi často dochází ke zjednodušování, aby šel problém řešit.

Ve výše uvedeném příkladu jsou všechny přechody stejně nákladné. Uveďme si i příklad, kdy jsme třeba na dálniční síti ČR. Vytvořme si graf propojení velkých měst ČR pomocí dálnic. Provedeme určité zjednodušení. Například v realitě při cestě z Pardubic do Prahy není třeba jet přes Hradec Králové.





Stavový systém

- $S = \{s_1, s_2, \dots\}$ - množina měst
- $A = \{a_1, a_2, \dots\}$ - jeď (Praha), jeď (Liberec), …
- $E = \{\emptyset\}$
- γ

Graf G(V, E, w)

- V – města
- E – přímé cesty z města do města ($u_1 = (Liberec,\ \gamma(Liberec, jeď\ (Praha))$
- w – vzdálenosti z města do města, například w(e_1)=110

3.3 Reprezentace grafu v počítači

Přechodový systém, respektive graf lze v počítači reprezentovat více způsoby.

- Matice sousednosti

- Incidenční matice
- Seznam sousedů

Tyto reprezentace můžeme hodnotit podle paměťové náročnosti a výkonové náročnosti. Výkonovou náročnost budeme hodnotit pomocí asymptotické náročnosti na nalezení všech následníků vrcholu.

3.3.1 Asymptotická složitost

Asymptotická složitost je definovaná jako horní asymptotické omezení algoritmu. Jinými slovy nejhorší možný výkonový výsledek algoritmu, když předpokládáme nejnepříznivější data.

Asymptotická složitost se zapisuje pomocí „Omikron notace“: $O(f(n))$, kde f je obecná funkce, n je proměnná, c je konstanta.

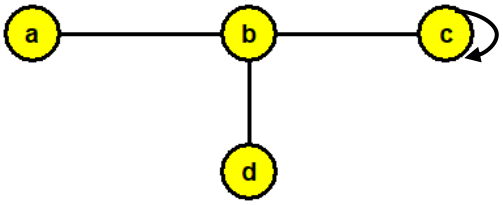
Náročnost	Vzorec
Konstantní	$O(1)$
Logaritmická	$O(\log n)$
Odmocninná	$O(n^{\frac{1}{2}})$
Lineární	$O(n)$
Mocninná	$O(n^2)$
Polynomiální	$O(n^c)$
Exponenciální	$O(c^n)$
Faktoriálová	$O(n!)$

3.3.2 Matice sousednosti

Konečný graf G o n vrcholech lze reprezentovat maticí A (Adjacency Matrix**)** o rozměrech $n\ x\ n$.

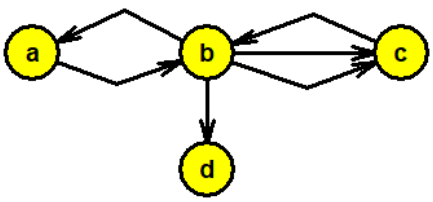
- Nediagonální prvek matice a_{ij} udává počet hran z vrcholu v_i do vrcholu v_j.
- Diagonální prvek matice a_{ii}, udává počet hran z vrcholu v_i do vrcholu v_i, tedy počet smyček.

Pro neorientované grafy je matice sousledností symetrická.



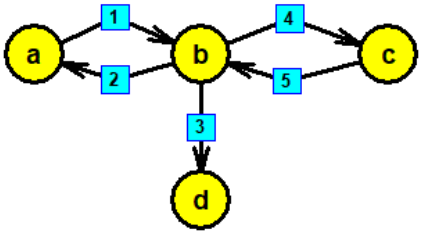
$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

U orientovaných grafů matice již nemusí být symetrická.



$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 2 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

U prostých orientovaných grafů, lze do matice sousloednosti ukládat i váhy hran. Pro neexistující hranu se ukládá hodnota -1, aby se odlišila hrana s hodnotou 0. Pro grafy, které obsahují vícenásobné hrany, smyčky nebo záporné ohodnocení není matice sousledností vhodná.



$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 2 & 0 & 4 & 3 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

- Paměťová náročnost: $O(n^2)$
- Čas potřebný k vyjmenování hran: $O(n)$
- Čas ověření existence hrany: $O(1)$

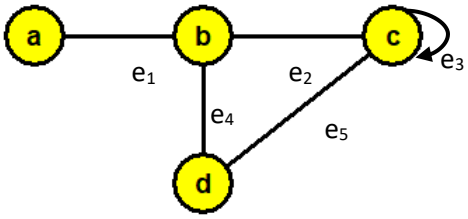
Matice sousledností je vhodná pro husté grafy, které mají alespoň . V praxi se mnohem častěji vyskytují řídké grafy. U řídkých grafů matice obsahuje příliš mnoho nepotřebných 0.

3.3.3 Incidenční matice

Jiným typem uložení je incidenční matice (Incidence Matrix). Konečný graf G o n vrcholech a m hranách je reprezentovaný incidenční maticí B o rozměrech $n \, x \, m$. Matice obsahuje následující hodnoty:

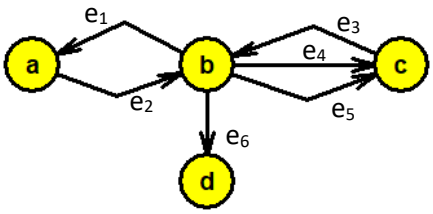
- 1, hrana e_j vede z vrcholu v_i
- 2, hrana e_j je smyčka ve vrcholu v_i
- -1, hrana e_j vede do vrcholu v_i
- 0, hrana s vrcholem v_i není incidentní

Protože je každá hrana incidentní se dvěma vrcholy, bude suma sloupců vždy 2.



	e_1	e_2	e_3	e_4	e_5
a	1	0	0	0	0
b	1	1	0	1	0
c	0	1	2	0	1
d	0	0	0	1	1

U orientovaného grafů bude matice vypadat následovně:



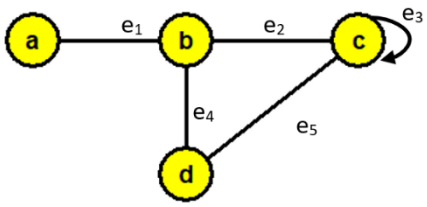
	e_1	e_2	e_3	e_4	e_5	e_6
a	-1	1	0	0	0	0
b	1	-1	-1	1	1	1
c	0	0	1	-1	-1	0
d	0	0	0	0	0	-1

- Paměťová náročnost: $O(m \cdot n)$
- Čas potřebný k vyjmenování hran: $O(m)$

Incidenční matice se vzhledem k velké paměťové a časové náročnosti používá spíše pro teoretické důkazy. Je užitečná zejména při řešení problémů souvisejících s hranami, jako jsou problémy týkající se toků v sítích nebo hledání mostů v grafu.

3.3.4 Seznam sousedů

Další reprezentací grafu v paměti je seznam sousedů (Adjacency List), který je vhodný i pro počítačové zpracování. Vrcholy jsou uloženy v seznamu. Ke každému vrcholu se přiřazuje seznam incidentních hran. Tento typ záznamu je vhodný pro řídké matice.

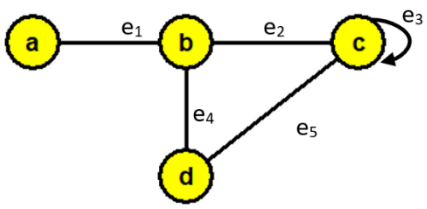


- a: b
- b: a, c, d
- c: b, c, d
- d: b, c

- Paměťová náročnost: $O(m + n)$
- Čas potřebný k vyjmenování hran, kde n_n je počet následníků vrcholu: $O(n_n)$

3.3.5 Seznam hran

Seznam všech hran grafu (Edge List), kde každá hrana je reprezentována jako pár vrcholů (nebo trojice, pokud je hrana ohodnocená). Tento seznam jednoduše uvádí všechny hrany v grafu. Každá hrana je reprezentována jako pár vrcholů, které spojuje. Tento způsob reprezentace je zvláště užitečný, pokud chceme mít rychlý přehled o všech hranách v grafu nebo pokud chceme zpracovávat hrany grafu postupně.



- AB, BC, BD, CC, CD

3.4 Formulace úlohy

Reálnou plánovací úlohu je třeba nejprve převést do konceptuálního modelu například do přechodového systému. Správná formulace úlohy nám pomůže k nalezení vhodného řešení.

Prvním krokem je **definice stavů**. Stav systému je určen podstatnými informacemi, uloženými ve vhodné paměťové struktuře (pole, seznam, strom, …). Důležitá je schopnost odlišit různé stavy.

Druhým krokem je **definice akcí** nad systémem, tedy operací, které mají schopnost měnit stav systému. Správná volba akcí, viz posun kamene nebo posun „mezery“, nám může úlohu zjednodušit.

U některých systémů, které interagují s prostředím, musíme vymezit i **události**.

Součástí formulace problému je i definice **přechodové funkce**, kterou si můžeme představit jako pravidla, za jakých podmínek, stavů systémů je možné danou akci použít, a jak změní stav systému. Na základě těchto informací jsme schopni vytvořit **stavový prostor úlohy**. Jedná se o model, který nám umožňuje znázornit veškeré možné chování systému. V našem případě bude mít podobu orientovaného grafu, kde stavy jsou vrcholy a akce jsou jména hran.

Pro řešení konkrétní úlohy je třeba definovat **počáteční stav** systému – stav herního pole, aktuální stav a pozice vozidel a nákladu, …

Nezbytnou součástí formulace problému je i definice **cílového stavu**, respektive více možných cílových stavů. Cíl může být vyjádřen **explicitně** (konkrétní poloha herních kamenů). Cíl lze definovat **podmínkami** (definice matu, patu), dosažení cíle s maximální efektivností (nejkratší cesta). Cíl může být definovaný i více abstraktně jako například vykonání úkolu (vysátí místnosti).

Při formulaci problému velmi často musíme určit **míru abstrakce** a různé předpoklady, které nám umožní komplexní reálný systém namodelovat. Zjednodušující předpoklady jsou například:

- Diskrétní reprezentace stavů – konečný počet stavů
- Deterministické chování systému – akce aplikovaná na systém v daném stavu vždy povede na stejný nový stav
- Plně pozorovatelné prostředí – vše co ovlivňuje úlohu je známo
- Statické prostředí – úloha se nemění v čase řešení problému, bez vnějších událostí
- Přesně definovaný cíl úlohy
- Sériové provádění akcí – není povolená paralelizace akcí
- Implicitní čas akce – neuvažuje se doba nutná ke změně stavu

Proces **plánování** je algoritmus, který nám vrátí řešení pro konkrétní problém. **Řešení** problému je posloupnost vhodných akcí (**plán**), jejíchž aplikací na počáteční stav se dostaneme do cílového stavu. Plán lze reprezentovat cestou ve stavovém prostoru.

3.5 Strategie hledání řešení problému

Ve chvíli, kdy máme definovanou úlohu, můžeme se podívat na strategie, jak vyhledat řešení úlohy.

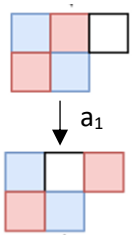
Pokud budeme mít triviální úlohu, jako výše uvedený hlavolam, můžeme zkusit dopředu vytvořit kompletní stavový prostor. Pro složitější úlohy stanovení celého stavového prostoru bude výkonově a paměťově velmi náročné až nemožné. Například celý stavový prostor pro hru šachy bude obsahovat všechny možné partie. Při herním poli 8x8 a 16 herních figur je odhad, že možných přípustných stavů je 10^{43} až 10^{50}. Počet veškerých možných tahů je přibližně 10^{123}, přičemž počet atomů v pozorovatelném vesmíru se pohybuje v rozmezí 10^{78} až 10^{82}. Vytvoření kompletního stavového prostoru je tedy možné jen pro koncovky.

I v případě, že dokážeme vytvořit kompletní stavový prostor, není to efektivní. Proto se používají různé strategie, jak prohledávat stavový prostor a nalézt cílový stav.

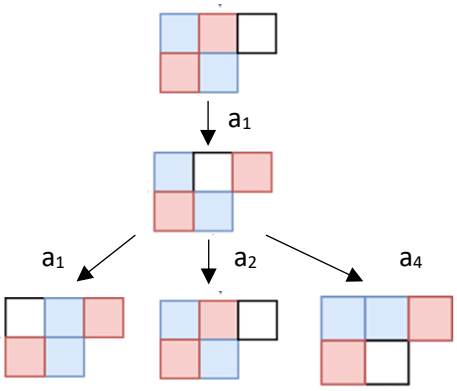
Mějme opět úlohu, kdy se ze počátečního stavu potřebujeme sérií akcí dostat do cílového stavu.

Zavedeme si operaci expanze stavu, která na výchozí stav postupně aplikuje všechny akce a vrátí seznam všech následníků stavu. Během expanze stavu jsme omezeni přechodovou funkcí, kdy pro daný výchozí stav některé akce nejsou povoleny.

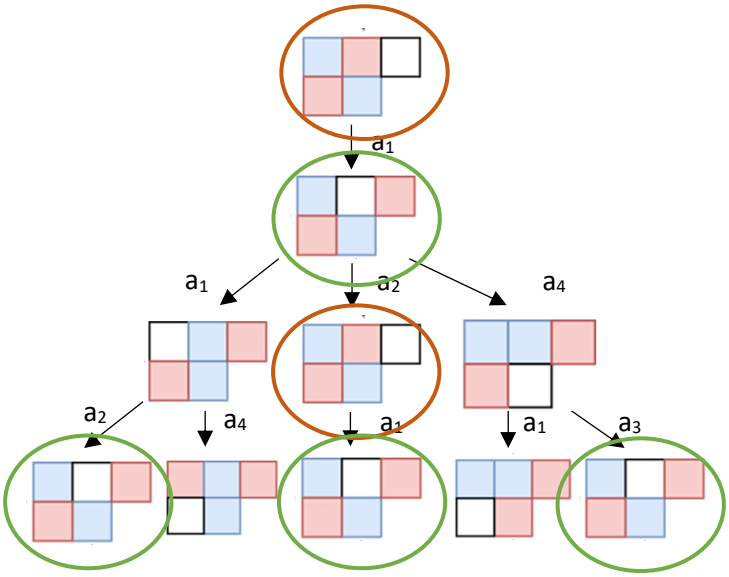
Při první expanzi jsme schopni provést pouze akci a_1 – posun prázdného pole doleva. Ostatní akce jsou neplatné.



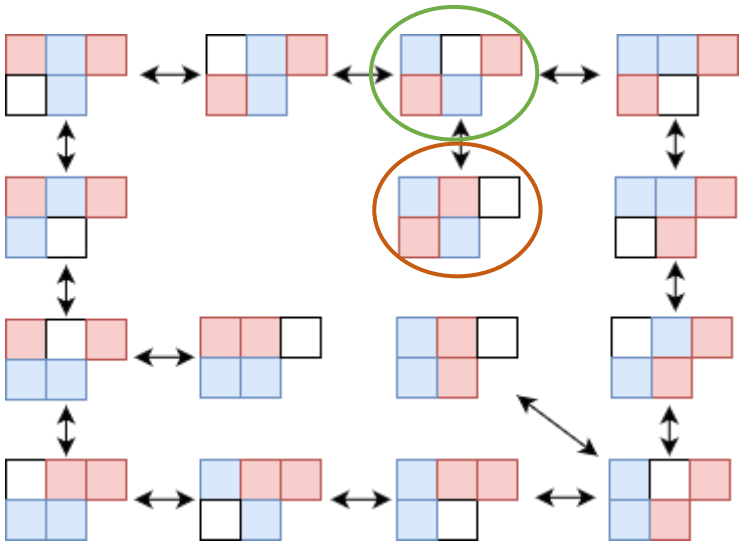
Následuje další expanze, kdy si opět můžeme vybrat pouze jeden stav, který lze ovšem expandovat do třech stavů. Jedinou nepovolenou akcí je a_3 – posun prázdného pole nahoru.



Můžeme pokračovat v dalším rozvoji. Opět některé akce nebudou platné. Je dobré si povšimnout, že při takovéto expanzi se některé stavy začnou opakovat.



Pokud se podíváme na kompletní stavový prostor, tak se jedná vlastně o pohyby zpět.



Postupnou expanzí získáváme kořenový strom, který buduje stavový prostor úloh. V nejvíce nepříznivém případu vytvoříme celý stavový prostor.

Velmi často máme na výběr, který stav budeme dále expandovat. Různé strategie určují různé pořadí expanzí stavů.

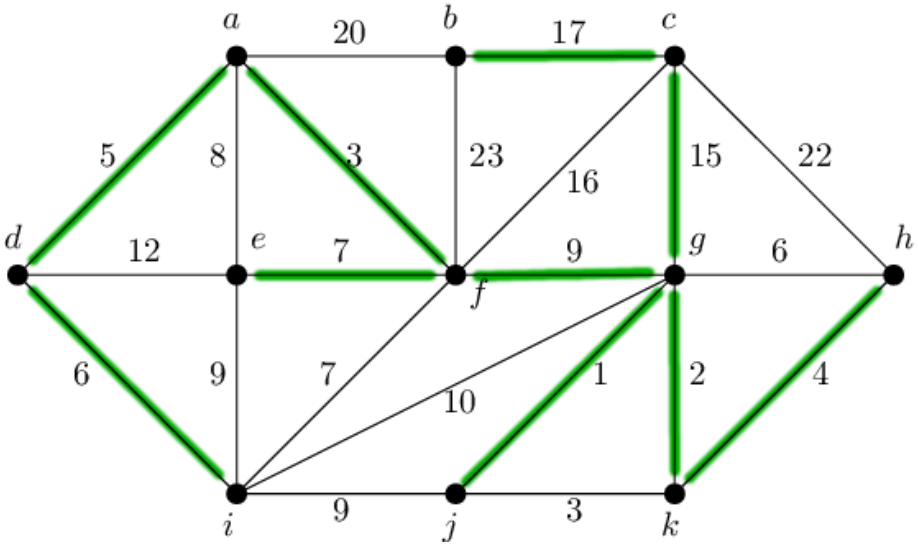
Při řešení problému se snažíme prozkoumat co nejméně stavů v co nejkratším čase, než dosáhneme cílového stavu. Plán vyřešení úloh je v kořenovém stromu reprezentovaná **cestou** od kořene k uzlu, který reprezentuje cílový stav.

4 Další grafové algoritmy

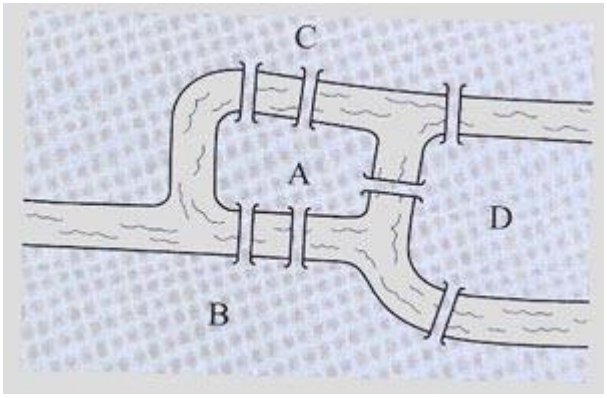
Zatím jsme nastínili algoritmy pro prohledávání grafu, které v dalších kapitolách budeme detailněji probírat a pomohou nám vyhledat řešení plánovacích úloh.

V teorii grafů ovšem existuje celá řada dalších problémů, které se dají řešit různými algoritmy.

Například se jedná o problém nalezení minimální kostry grafu, tedy množiny cest, které umožňují cesty z libovolného vrcholu do jiného libovolného vrcholu, ovšem za podmínky, že v grafu nevznikne kružnice. Tento typ úlohy můžeme například řešit Kraskalův, Borůvkův nebo Primův algoritmus. Praktickým příkladem může být například naplánování rozvodné sítě mezi různými obcemi za podmínky minimalizace nákladů.



Zajímavý problém vyřkl Euler, který bydlel v městě Královec. V městě byly ostrovy propojené sedmi mosty. Euler se snažil přijít na způsob, jak na procházce přejít všechny mosty (hrany) a každý navštívit pouze jednou.



V praxi se tento typ úloh velmi často objevuje při distribuci, kdy je potřeba obsloužit různé cesty a minimalizovat přitom projetou dráhu.

Velmi podobná úloha se týká Hamiltonovských grafů, kdy je třeba navštívit všechny vrcholy a každý pouze jednou. Tento typ úloh se nazývá problém obchodního cestujícího.

Existují problémy týkající se topologie grafů, rovinnosti grafů, barvení grafů, které se využívají například při návrhu barvení map, topologickém uspořádání plánků atd.

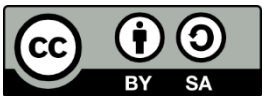
- Schopnost pracovat s daty v prostoru vyšší dimenze
- Schopnost zpracovat nelineární data
- Schopnost vyhýbat se problémům s přetrénováním
- Odolnosti vůči datům, která jsou daleko od nadroviny a jsou efektivní, protože jsou založena pouze na podpůrných vektorech uvnitř nadroviny.
- Efektivní pro data s malým počtem tréninkových příkladů

4.1.1 Nevýhody

- Algoritmus může být výpočetně a paměťově náročný při větším objemu dat.
- Algoritmus je primárně určený pro klasifikaci dvou proměnných. Nové implementace umožňují klasifikovat více tříd pomocí přístupu jedna proti jedné nebo jedna proti všem. To opět zvyšuje výpočetní náročnost.
- SVM se mohou jevit jako černé skříňky, protože součástí odhadu není konečná funkční forma nebo tabulka koeficientů pro různé prediktory.

UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vytvořeno v rámci projektu **Digitalizace studijních Agend, Nové TechnologiE, systémy a přístupy k výuce na UPCE**, reg. č. NPO_UPCE_MSMT-16591/2022.



Financováno Evropskou unií
NextGenerationEU

Národní plán obnovy

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Machine learning & AI

Prohledávání stavového prostoru

Ing. Martin Pozdílek, Ph.D.

1 Strategie

V minulé kapitole jsme definovali nový typ úlohy pro strojivé učení – plánování, jejíž cílem je nalezení řešení ve stavovém prostoru. Ukázali jsme si, jak pro zadání správně formulovat problém.

V této kapitole se podíváme na různé strategie, pomocí kterých můžeme budovat stavový prostor, respektive ho prohledávat. Ukážeme si různé strategie, které budou různě efektivní a budou mít různé vlastnosti.

Je několik požadavků, které máme na strategii:

- Strategie musí vést k **prohledávání**, musí zabránit zacyklení
- Strategie musí být **systematická**, nesmí vynechat žádný stav

Strategie lze rozdělit na:

- **Neinformované metody** – hledání je prováděno hrubou silou
- **Informované metody** – při hledání se využívají různé heuristické znalosti, které napomáhají k rychlejšímu nalezení cílového stavu.

Algoritmy lze hodnotit z hlediska kvality a náročnosti. Velmi často jde kvalita a náročnost proti sobě a je třeba volit nějaký kompromis.

Vlastnosti, které určují **kvalitu** jsou:

- **Úplnost** – pokud existuje řešení, algoritmus zaručuje jeho nalezení
- **Optimálnost** – řešení nalezené algoritmem je optimální, neexistuje žádné lepší

Algoritmy lze hodnotit i podle **náročnosti**.

- **Časová náročnost** – potřebná doba k nalezení řešení. Lze ji vyjádřit maximálním počtem vrcholů vygenerovaných během řešení
- **Paměťová náročnost** – vyjádřeno maximálním počtem vrcholů uložených v paměti.

1.1 Reprezentace úlohy v paměti

Při popisu různých strategií, algoritmů prohledávání budeme v paměti postupně vytvářet stavový prostor. Nejprve si definujeme, jak budeme řešení problému reprezentovat. Diagram tříd může vypadat třeba následovně:

Třídu **State** budeme používat pro zachycení stavu systému. Interní strukturu třídy bude vždy potřeba upravit podle zadání problému. Například se může jednat o vícerozměrné pole, množinu proměnných atd. Do třídy můžeme přidat metodu **expand**, která podle vstupní akce vytvoří ze stávajícího stavu stav nový. Zároveň předpokládáme, že jsme schopni porovnat dvě instance třídy State

a určit, zda jsou identické nebo odlišné. To například můžeme provést přetížením operátoru equal. Opět zcela záleží na implementaci uložení stavu.

Další třída **Node** reprezentuje uzel v kořenovém stromu. Základním atributem je **state**, který uchovává stav systému. Atribut **parent** ukazuje na předchůdce stavu v kořenovém stromě. Atribut **depth** ukládá hloubku uzlu a později nám umožní například omezit procházení do hloubky. Posledním atributem je **action**, ve kterém je zaznamenána akce, která byla provedena na přecházejícím stavu, abychom získali stav uložený v uzlu. Tyto dodatečné atributy nám umožní pohyb v kořenovém stromě.

Třída Node má dvě metody. Metoda **successors** generuje následníky uzlu, tím že volá metodu expand pro jednotlivé možné akce. Metoda vrací seznam následníků. Metoda **path** slouží pro získání cesty z kořene stromu do daného uzlu. Ve chvíli, kdy najdeme řešení, použijeme tuto metodu pro získání plánu – seznamu akcí.

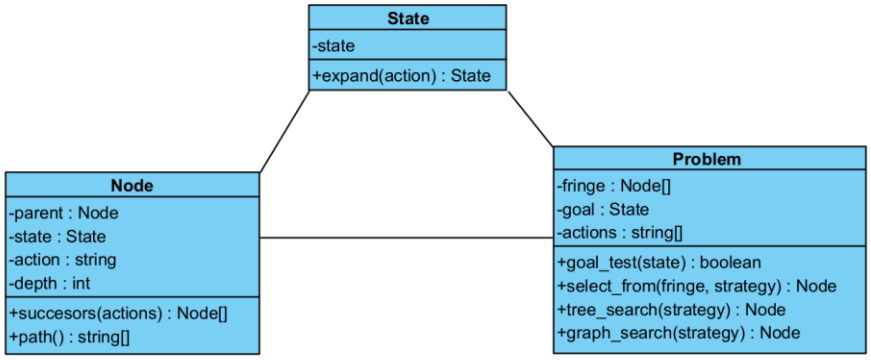
Poslední třídou je **Problem**, která reprezentuje zadání problému. Atribut goal obsahuje cílový stav nebo cílové stavy. Tento atribut je používaný metodou **goal_test**, která otestuje, zda daný stav je cílový. Metoda vrací hodnotu true nebo false. V základní verzi je metoda implementována jako ověření, že stav existuje v cílové množině stavů. Pokud je cílový stav definován podmínkami, goal nemusí obsahovat žádný cíl, ale musíme implementovat ověření podmínek v metodě goal_test.

Atribut **fringe** reprezentuje seznam nodů, které je třeba prohledat. Při vytváření instance třídy Problem musíme zadat počáteční stav systému. Z počátečního stavu systému se v konstruktoru třídy vytvoří instance třídy Node, která reprezentuje kořen stromu. Ten se v konstruktoru vloží do seznamu fringe.

Atribut **actions** reprezentuje seznam všech možných akcí, případně událostí, na které má umět systém reagovat. Jednotlivé akce jsou postupně předávány do třídy expand ve třídě State.

Metoda **select_from** vybírá ze seznamu uzlů k prohledání (fringe) podle zadané strategie jeden uzel, který se má expandovat. O konkrétních implementacích této metody se budeme bavit při probírání jednotlivých algoritmů.

Posledními metodami jsou **tree_search** a **graph_search**. Jedná se o metody, ve kterých je implementován algoritmus stromového a grafové prohledávání stavového prostoru. Jejich implementaci vysvětlíme vzápětí.



Obrázek 1 - Class diagram

1.2 Obecný algoritmus prohledávání

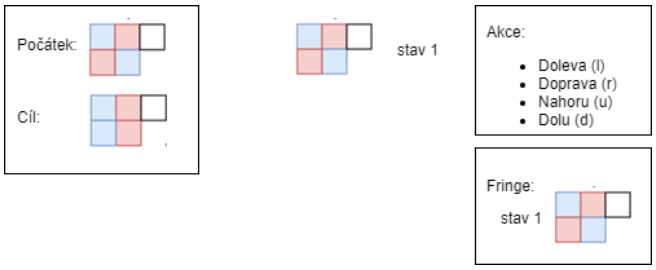
Jednotlivé algoritmy si budeme ukazovat na jednoduchém logickém problému s přesouváním modrých a červených herních kamenů po herní ploše, jak jsi si představili v minulé kapitole.

Problém můžeme definovat například takto. Stav budeme reprezentovat dvourozměrným polem, kdy hodnota 1 představuje modrý kámen, hodnota 2 červený kámen a hodnota 0 mezeru. Abychom nemuseli procházet dvourozměrné pole, uložíme si i pozici prázdného pole do dvojice proměnných row a column. Dále definujeme cílový stav a povolené akce – tah prázdným polem.

```
problem = Problem(initial_state=State(gameplan=[[1, 2, 0], [2, 1]],
                                      row=0,
                                      columne=2),
                  goal=State([[1, 2, 0], [1, 2]], 0, 2),
                  actions=['l', 'r', 'u', 'd'],
                  )
```

Při inicializaci problému se do seznam stavů k prohledání vloží počáteční stav.

```
def __init__(self, initial_state, goal, actions):
    self.fringe = []
    self.fringe.append(Node(parent=None, state=initial_state, action=None,
depth=0))
    self.goal = goal
    self.actions = actions
```



1.2.1 Prohledávání stromu

Prvním představeným algoritmem je prohledávání stromu. Cyklus algoritmu bude pokračovat do doby, než se nalezne řešení nebo dokud nebude seznamu stavů k prohledání (fringe) prázdný.

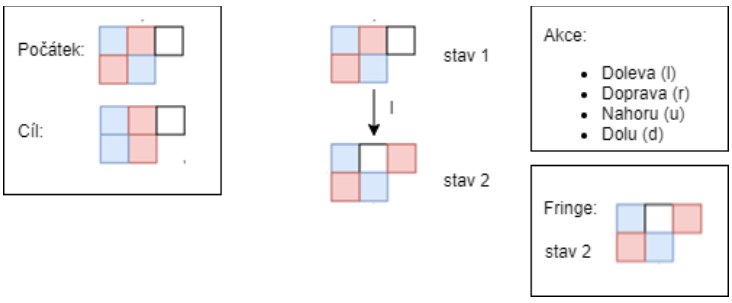
Pokud seznam stavů k prohledání není prázdný, zavoláme metodu select_from, která podle strategie (vybraného algoritmu) zvolí uzel, který se bude prohledávat. Právě implementací této metody se budou hlavně lišit dále probírané algoritmy prohledávání.

Zvolený uzel se otestuje, zda není cílový, pomocí metody goal_test.

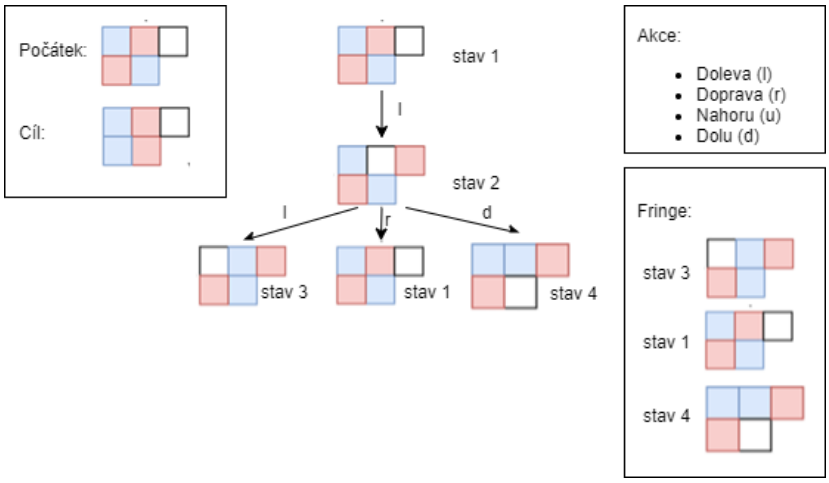
Pokud jsme zatím nenalezli řešení, tak provedeme expanzi aktuálního uzlu a nové stavy vložíme do seznamu stavů k prohledání.

```
def tree_search(self, strategy):
    while True:
        if len(self.fringe) == 0:
            return None

        node = self.select_from(self.fringe, strategy)

        if self.goal_test(node.state):
            return node

        self.fringe.extend(node.succesors(self.actions))
```

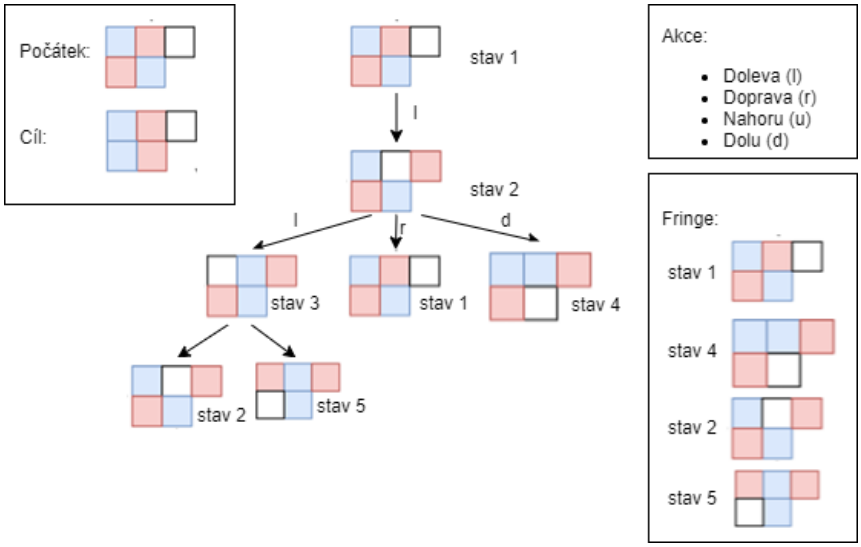
Po inicializaci problému seznam Fringe není prázdný. Vyzvedneme z něj stav 1. Stav 1 není cíl, proto ho expandujeme. Jediná možná akce je doleva, která vygeneruje stav 2, který vložíme do seznamu.



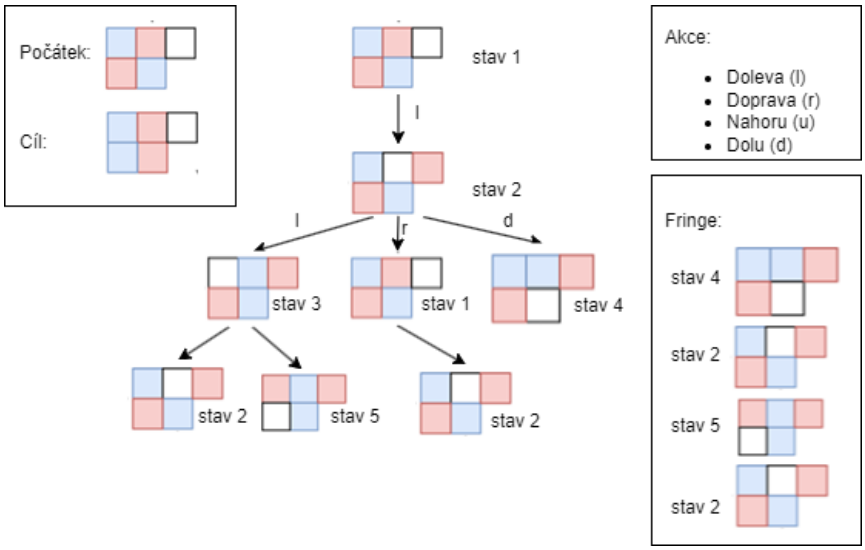
Ze seznamu vezmeme první stav 2. Stav 2 není cílový. Proto ho expandujeme. Možné akce jsou doleva, doprava a dolů. Stavy 3, 1 a 4 vložíme do seznamu.



Ze seznamu vezmeme první stav 3. Stav 3 není cílový. Proto ho expandujeme. Možné akce jsou doprava a dolů. V seznamu již jsou stavy 1 a 4. Expandované stavy 2 a 5 vložíme do seznamu.

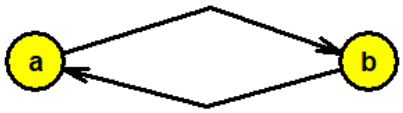


Ze seznamu vyjmeme stav 1. Stav 1 není cílový. Provedeme jeho expanzi. Možný je pohyb pouze doleva a tím vytvoříme stav 2. Stav 2 přidáme na konec seznamu.



Stav 1 jsme již ale jednou prohledávali a expandovali. Děláme stejnou činnost znovu. Stromový prohledávací algoritmus nás sice nakonec dovede k výsledku, ale v našem případě bude neefektivní.

Další nevýhodou je, že pokud se ve stavovém prostoru vyskytuje cyklus, tím pádem graf není stromem, může dojít k zacyklení algoritmu. Například počátečním stavem je uzel a, cílovým stavem je uzel c. Stromový algoritmus bude stále opakovat prohledávání stavů a a b a nikdy nevrátí, že pro daný problém neexistuje řešení.

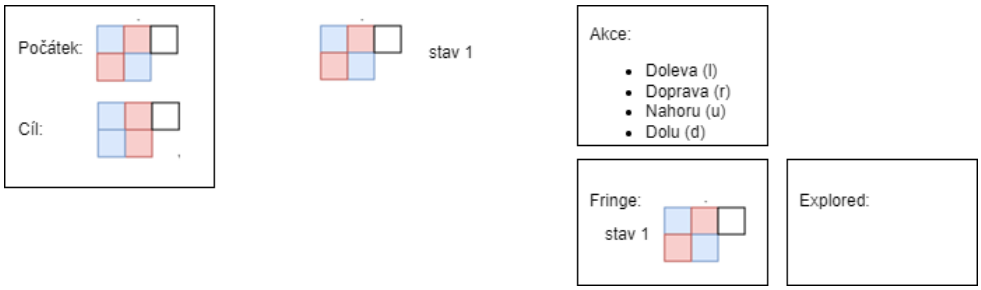


1.2.2 Prohledávání grafu

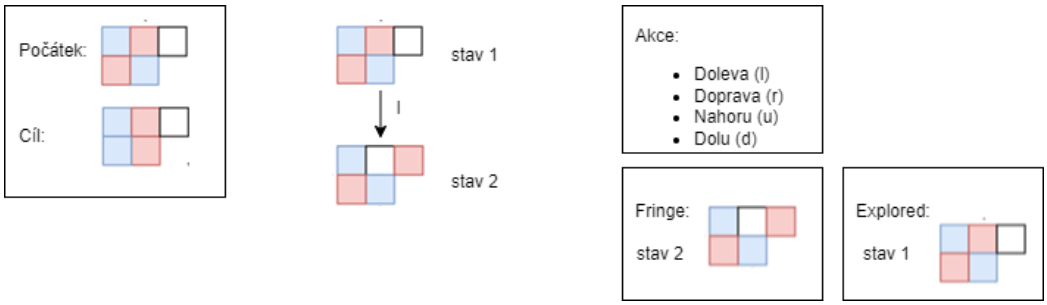
Výše zmíněné problémy řeší algoritmus prohledávání grafu. Tento algoritmus rozšiřuje stromové prohledávání o seznam již navštívených uzlů. Jednou navštívené uzly již nepřidává do seznamu k prohledávání. Tím se eliminuje jejich opakované prohledávání a zároveň se zabrání cyklení v případě, že neexistuje řešení.

```
def graph_searchself, strategy):

    explored = []
    while True:
        if len(self.fringe) == 0:
            return None

        node = self.select_from(self.fringe, strategy)

        if self.goal_test(node.state):
            return node

        explored.append(node)

        succesors = node.succesors(self.actions)

        for succesor in succesors:
            if (succesor not in explored)
                and (succesor not in self.fringe):
                self.fringe.append(succesor)
```

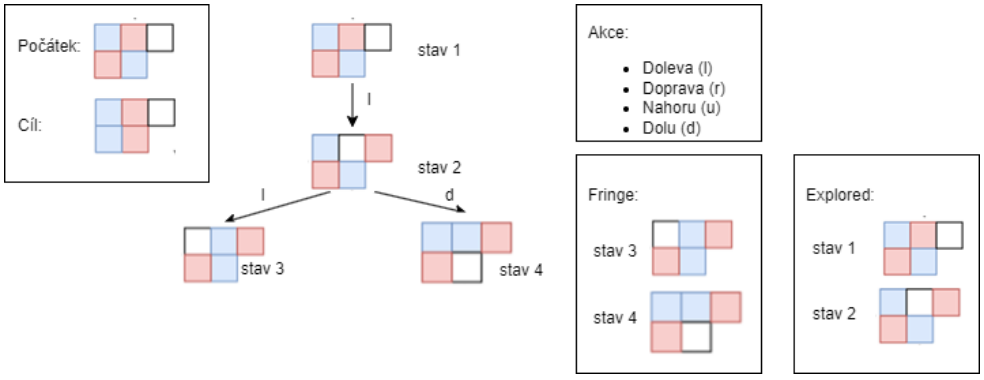
Počáteční stav vypadá velmi podobně jako u stromového algoritmu. Navíc máme prázdný seznam s prozkoumanými uzly.



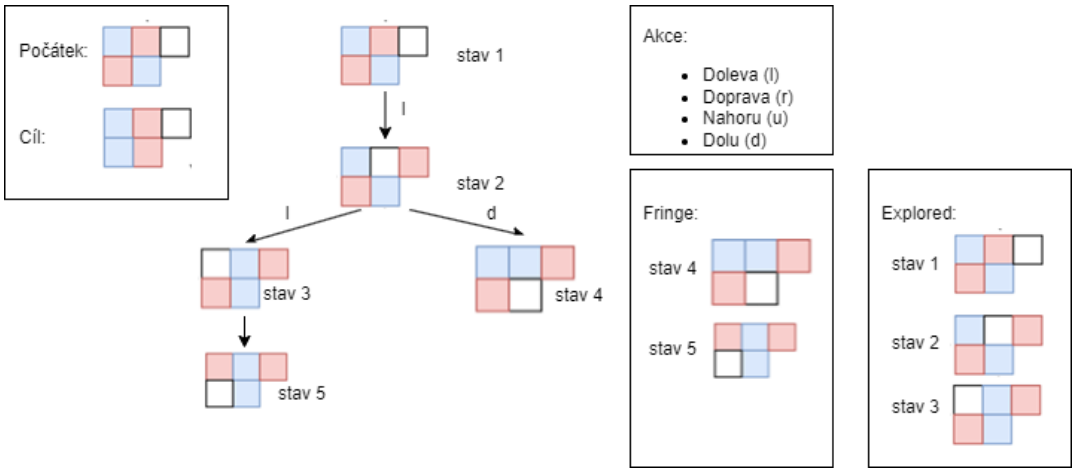
Ze seznamu fringe vybere první (jediný stav 1). Zkontrolujeme, že to není cílový stav. Vložíme ho do seznamu prozkoumaných stavů a provedeme jeho expanzi. Jediným povoleným stavem je stav 2, který zatím nebyl prozkoumaný, a tak ho vložíme do seznamu k prozkoumání.



Ze seznamu vybíráme první stav 2. Ten není konečným stavem. Vložíme ho do seznamu prozkoumaných a provedeme jeho expanzi. Stav lze expandovat do stavu 3, 4 a 1. Protože stav 1 již v seznamu prozkoumaných, vložíme do seznamu k prozkoumání pouze stavy 3 a 4.



Ze seznamu vybíráme první stav 3. Ten není konečným stavem. Vložíme ho do seznamu prozkoumaných a provedeme jeho expanzi. Stav lze expandovat do stavu 2 a 5. Protože stav 2 již v seznamu prozkoumaných, vložíme do seznamu k prozkoumání pouze stav 5.



Takto budeme pokračovat, než získáme cílový stav.

2 Neinformované metody

Metod, jak vybrat další uzel (vrchol) ze seznamu fringe je celá řada. Pokud pro výběr nepoužíváme žádné dodatečné informace, mluvíme o neinformovaných metodách. Mezi neinformované metody patří:

- Prohledávání do šířky
- Prohledávání do hloubky
- Prohledávání do hloubky s omezením
- Iterativní prohledávání do hloubky
- Obousměrné prohledávání

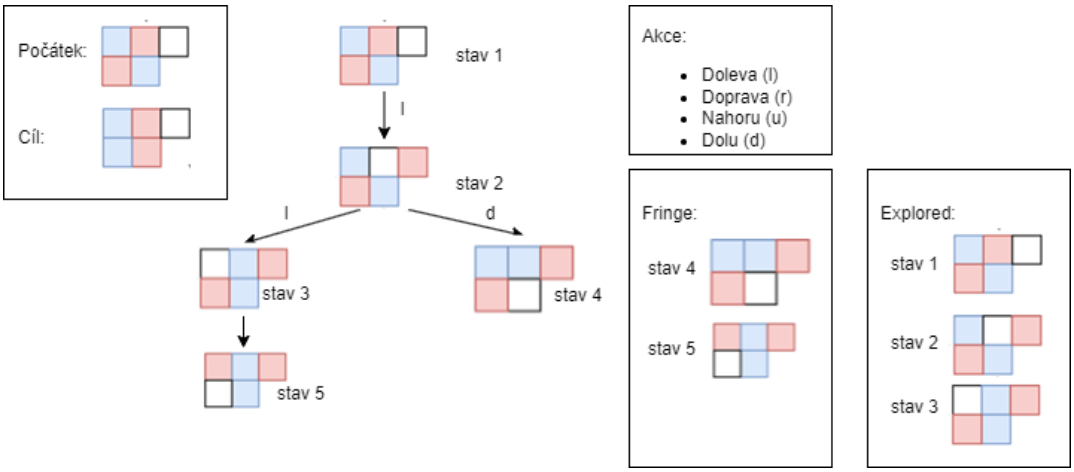
2.1 Prohledávání do šířky

Když jsme si představovali algoritmy pro stromové a grafové prohledávání, používali jsme prohledávání do šířky (Breadth-First Search – BFS). Při této strategie se pro další expanzi ze seznamu uzlů k prohledání vybírá uzel s **nejmenší** hloubkou.

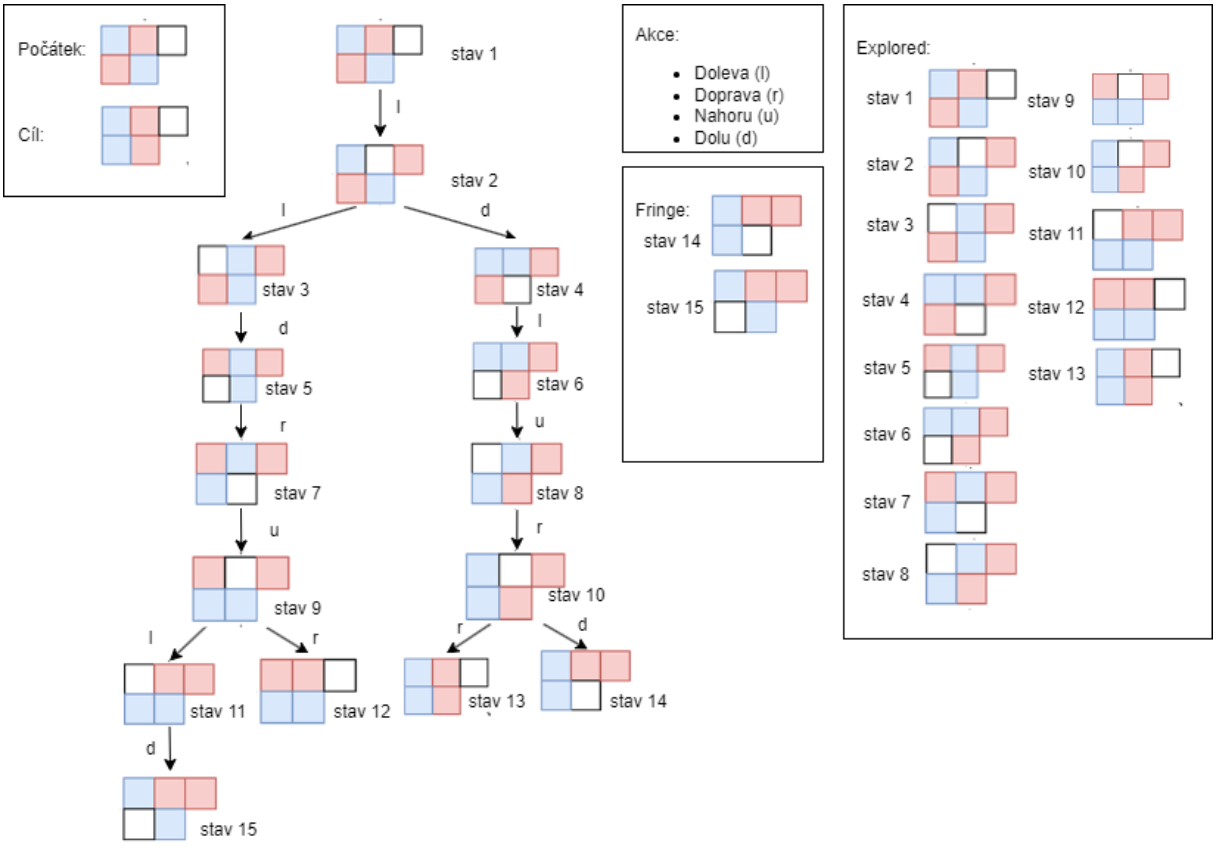
Tento výběr se dá velmi jednoduše implementovat pomocí fronty – FIFO (First-In-First-Out). V kódu implementace může vypadat například takto:

```
def select_from(self, fringe, strategy):
    return fringe.pop(0)
```

Navážeme na zobrazení grafového algoritmu.



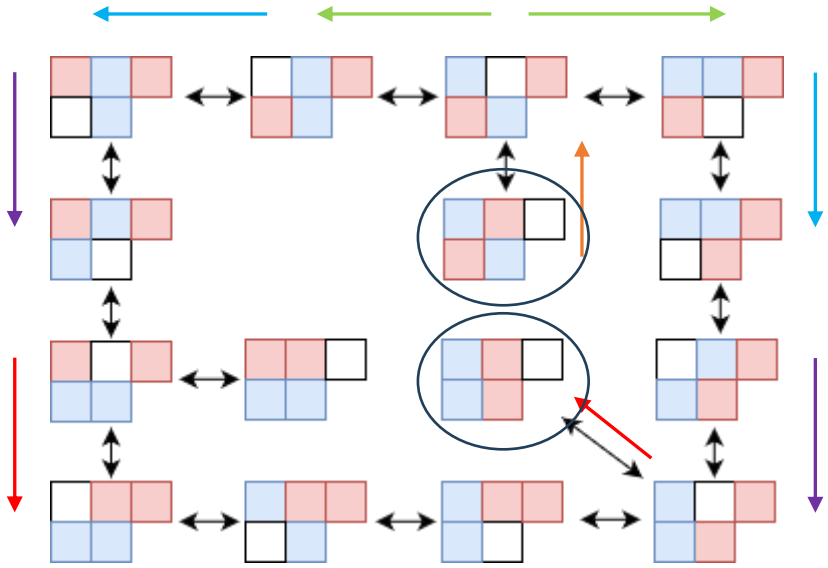
Algoritmus prohledávání do šířky bude postupně expandovat stavy 5 až 10. Díky podobě stavového prostoru jde stav expandovat pouze do jednoho následujícího stavu. Při expanzi stavu 10 se v seznamu fringe objeví cílový stav. Protože v algoritmu máme testování cílového stavu až při výběru ze seznamu, musíme zpracovat stav 11, který expanduje do stavu 15. Dále zpracujeme stav 12, který expanduje pouze do prozkoumaného stavu 9. Teprve pak vybere cílový stav 13 a algoritmus končí. Řešením je cesta z kořene stromu do cílového stavu. V našem případě tedy L, D, L, U, R, R.



Algoritmus by šel tedy trochu zrychlit, pokud bychom testovali, zda je stav cílový již při vkládání do seznamu k prozkoumání.

```
        for succesor in succesors:
            if (succesor not in explored)
                and (succesor not in self.fringe):
                if self.goal_test(node.state):
                    return node
                else:
                    self.fringe.append(succesor)
```

Prohledávání do šířky postupně rozšiřuje prohledávaný prostor všemi směry najednou.



2.1.1 Vlastnosti algoritmu

Některé vlastnosti jsou velmi příjemné.

- BFS zaručuje nalezení cesty.
- **Nalezená cesta je nejkratší**.

Problém BFS může být s výkonem. Pokud každý vrchol má maximálně **b** následovníků a nejlepší řešení leží v hloubce **d**, počet uzlů, které budou maximálně prozkoumány je následující:

$$N(BFS) = b + b^2 + b^3 + \cdots + b^d + b^{d+1}$$

Pokud budeme testovat, zda je uzel cílový již při vkládání do seznamu k prohledání, tak se počet zkoumaných uzlů sníží na:

$$N(BFS) = b + b^2 + b^3 + \cdots + b^d$$

Hlavní nevýhodou BFS je paměťová náročnost. Pokud opět budeme předpokládat stupeň větvení b na hodnotě 10 a velikost záznamu stavu v paměti, tak paměťová náročnost bude vypadat

takto. Spolu s paměťovou náročností je spojená i výpočetní náročnost, kdy je třeba uzel vygenerovat. Tabulka předpokládá, že počítač dokáže vygenerovat a uložit 1 000 000 stavů za vteřinu.

Hloubka řešení	Vrcholů	Paměť	Čas
2	110	107 kB	0,11 ms
4	11 110	10,6 MB	11 ms
6	10^6	1 GB	1,1 s
8	10^8	103 GB	2 min
10	10^{10}	10 TB	3 hod
12	10^{12}	1 PB	13 dní
14	10^{14}	99 PB	3,5 roku
16	10^{16}	10 EB	350 let

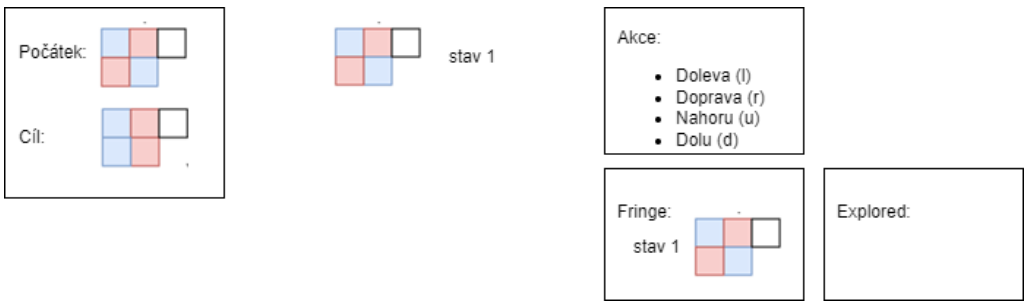
	Stromová verze	Grafová verze
Časová složitost	$O(b^d)$	$O(b^d)$
Paměťová složitost	$O(b^d)$	$O(b^d)$
Úplnost	Ano	Ano
Optimálnost	Ano	Ano

2.2 Prohledávání do hloubky

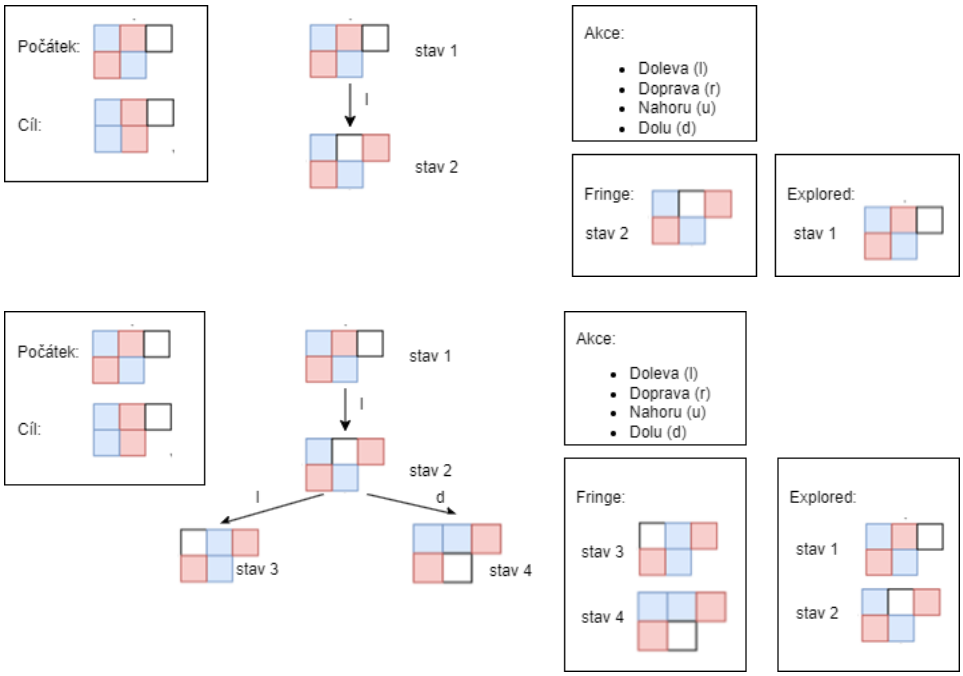
Druhou strategií, jak prohledávat strom nebo graf, je prohledávání do hloubky (Depth-First Search – DFS). V tomto případě při výběru uzlu k expanzi vybíráme ten s **největší** hloubkou. Toto lze jednoduše realizovat pomocí zásobníku – LIFO (Last-In-First-Out). To můžeme realizovat například pomocí tohoto kódu, který vybírá ze seznamu poslední prvek.

```
def select_from(self, fringe, strategy):
    return fringe.pop(-1)
```

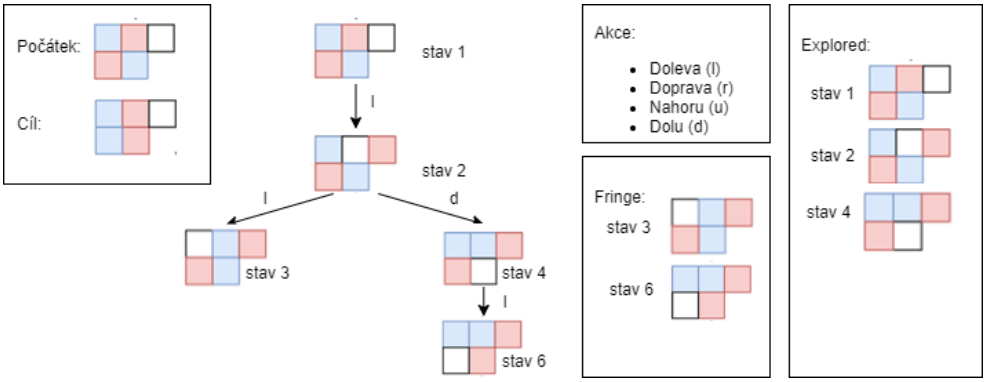
Postup algoritmu si opět ukážeme na našem problému. Počáteční stav bude totožný jako při prohledávání do šířky.



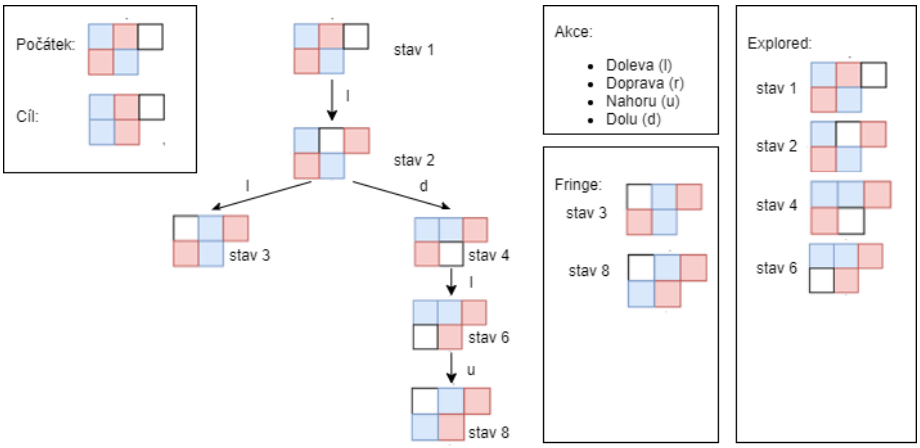
Protože v prvních iteracích algoritmu se do seznamu k prohledávání dostává pouze jeden uzel, průběh bude stejný jako u prohledávání do šířky.



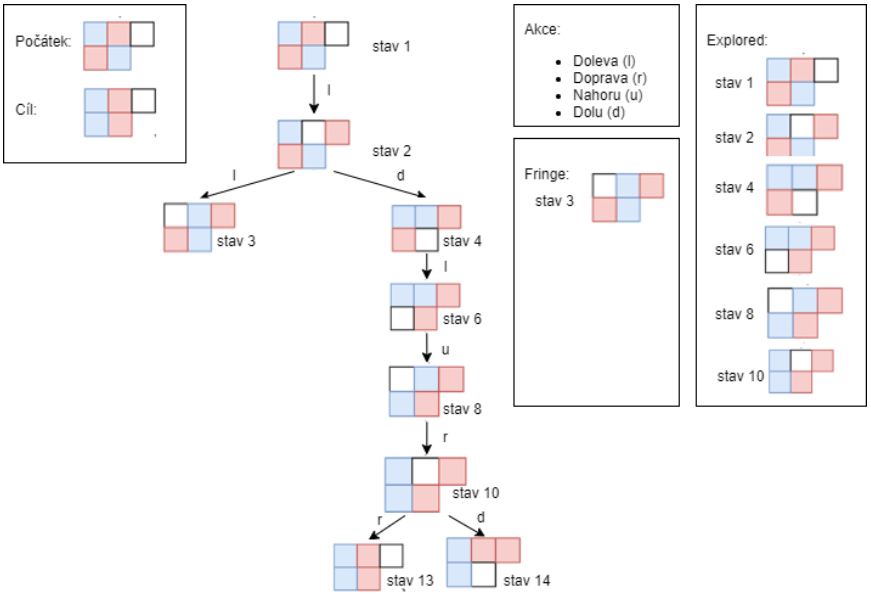
V této iteraci seznam obsahuje stavy 3 a 4. Oba mají stejnou hloubku, ale naše implementace formou zásobníku nám vrátí stav 4 k expanzi. Stav 4 není cílovým stavem. Tak provedeme jeho expanzi. Protože máme grafovou verzi algoritmu, jediným novým stavem je 6. Protože to není cílový stav, není v seznamu explored, tak ho vložíme na konec seznamu k expanzi.



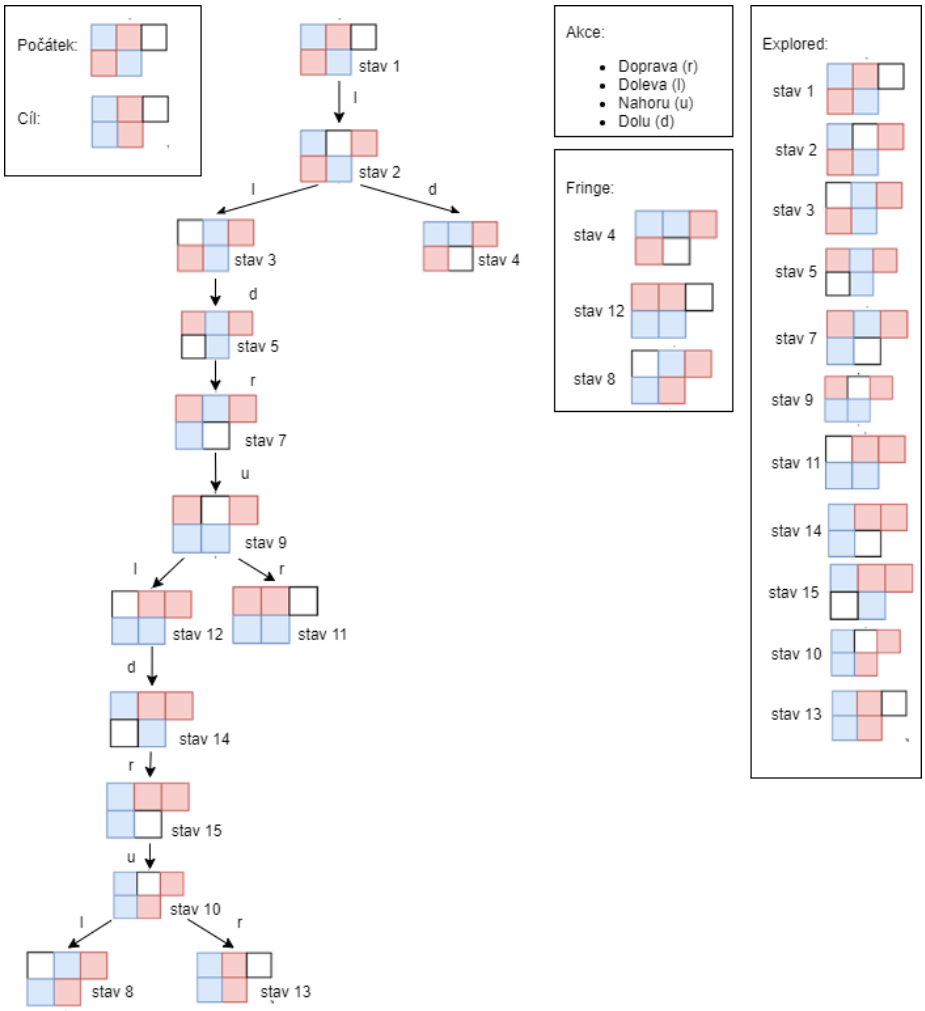
Ze zásobníku Fringe vyzvedneme stav 6, který má větší hloubku než stav 3. Provedeme expanzi stavu 6. Uložíme do seznamu zpracovaných uzlů. Jediným možných stavem, který není v seznamech fringe a explored a není cílovým stavem je stav 8.



Takto bude postupovat až do stavu 10. Při jeho expanzi vygenerujeme cílový stav 13 a jsme schopni vygenerovat řešení v podobě L, D, L, U, R, R. To je stejné jako u prohledávání do šířky. Ovšem počet vygenerovaných stavů je nižší, protože jsme nerozvíjeli větev stromu od stavu 3.



Pokud prohodíme pořadí zpracovávaných akcí, nalezené řešení bude vypadat jinak. Místo stavu 4 se začne rozvíjet větev začínající stavem 3. Algoritmus nakonec dosáhne cílového stavu. Řešení bude mít podobu: L, L, D, R, U, L, D, R, U, R. Toto řešení je ovšem delší než optimální řešení.



2.2.1 Vlastnosti algoritmu

Pokud každý vrchol má maximálně **b** následovníků a nejdelší cesta ve stavovém prostoru má délku m, počet uzlů, který bude maximálně vygenerováno je následující:

$$N(DFS) = b + b^2 + b^3 + \cdots + b^m + b^{m+1}$$

Pokud budeme testovat, zda je uzel cílový již při vkládání do seznamu k prohledání, tak se počet zkoumaných uzlů sníží na:

$$N(DFS) = b + b^2 + b^3 + \cdots + b^m$$

U stromové verze algoritmu hrozí zacyklení, pokud graf stavového prostoru netvoří strom a obsahuje cykly.

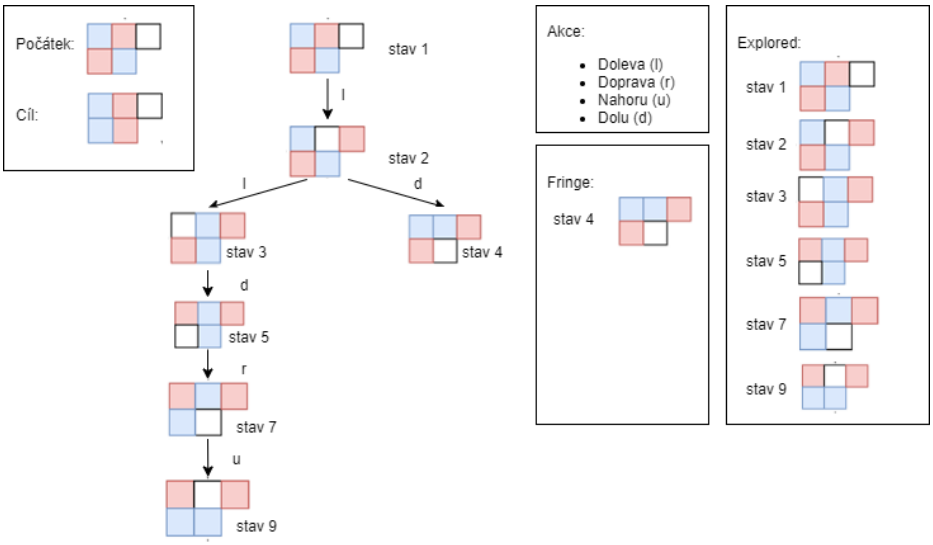
	Stromová verze	Grafová verze
Časová složitost	$O(b^m)$	$O(b^m)$
Paměťová složitost	$O(b^m)$	$O(b^m)$
Úplnost	Ano, pro strom	Ano
Optimálnost	Ne	Ne

Hlavní výhodou prohledávání do hloubky je snížení paměťové náročnosti a v případech, kdy je prohledávání do šířky příliš náročné, tak prohledávání do hloubky může přinést alespoň nějaké řešení, i když o něm nejsme schopni tvrdit, že je optimální.

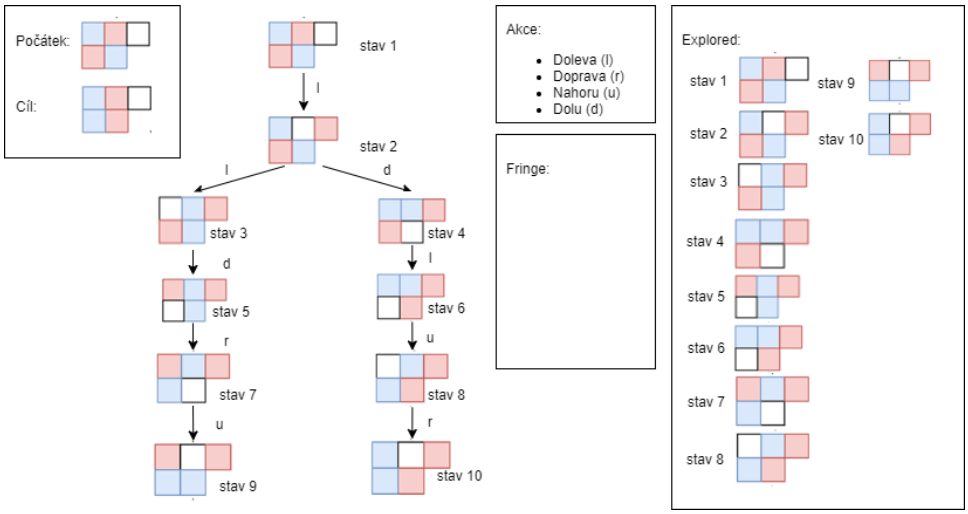
2.3 Prohledávání do hloubky s omezením

Modifikací algoritmu pro prohledávání do hloubky je prohledávání do hloubky s omezením (Depth Limited Search – DLS). V tomto algoritmu je nastavena maximální hloubka hledání například pomocí vstupního parametru. Při expanzi jsou ignorovány uzly, které mají hloubky vyšší než nastavený parametr.

Například při omezení hloubky 5, bude stavový diagram prohledáván následovně. Nejprve se prozkoumá větev začínající stavem 3. Pokud se dosáhne hloubky 5 – stav 9, dále se tento stav nebude expandovat.



Prohledávání pokračuje prohledáváním větve začínající stavem 4. A opět maximálně do hloubky 5. Protože cílový stav leží ve větší hloubce stavového prostoru, algoritmus řešení nenalezne.



2.3.1 Vlastnosti algoritmu

Pokud každý vrchol má maximálně **b** následovníků a maximální hloubka hledání je **l** a hloubka řešení je **d**, pak platí:

	Stromová verze	Grafová verze
Časová složitost	$O(b^l)$	$O(b^m)$
Paměťová složitost	$O(b^l)$	$O(b^m)$
Úplnost	Ano, pokud l >= d	Ano, pokud l >= d
Optimálnost	Ano, pokud l=d	Ano, pokud l=d

2.4 Iterativní prohledávání do hloubky

Iterativní prohledávání do hloubky (Iterative Deeping Search – IDS) je modifikací prohledávání do hloubky s omezením. Ve stromové verzi kombinuje pozitivní vlastnosti DFS a BFS.

Algoritmus má nastavené počáteční omezení do hloubky. Pokud se s tímto omezením řešení nenajde, zvýší se omezení při restartu prohledávání o 1.

Tento algoritmus je vhodný, pokud je známá přibližná hloubka řešení.

Za předpokladu, že nejlepší řešení leží v hloubce **d** a každý vrchol má maximálně **b** následovníků, pak je maximálně vygenerováno toto množství uzlů:

$$N(IDS) = (d)b + (d-1)b^2 + (d-2)b^3 + \cdots + 1b^d$$

Porovnáním počtu vygenerovaných uzlů pomocí BFS a IDS, zjistíme, že IDS vygeneruje o něco více uzlů, ale bude potřebovat méně paměti.

$$N\ (IDS, b = 10, d = 5) = 50 + 400 + 3\,000 + 20\,000 + 100\,000 = 123\,450$$

$$N\ (BFS, b = 10, d = 5) = 10 + 100 + 1\,000 + 10\,000 + 100\,000 = 111\,111$$

	Stromová verze
Časová složitost	$O(b^d)$
Paměťová složitost	$O(\mathrm{b} \cdot \mathrm{d})$
Úplnost	Ano
Optimálnost	Ano

2.5 Obousměrné prohledávání

Doposud jsme stavový prostor prohledávali od počátku. Algoritmus obousměrného prohledávání (Bidirectional Search – BIDI) simultánně prohledává stavový prostor od počátečního a koncového stavu. Předpokládá se, že obě prohledávání se potkají přibližně uprostřed.

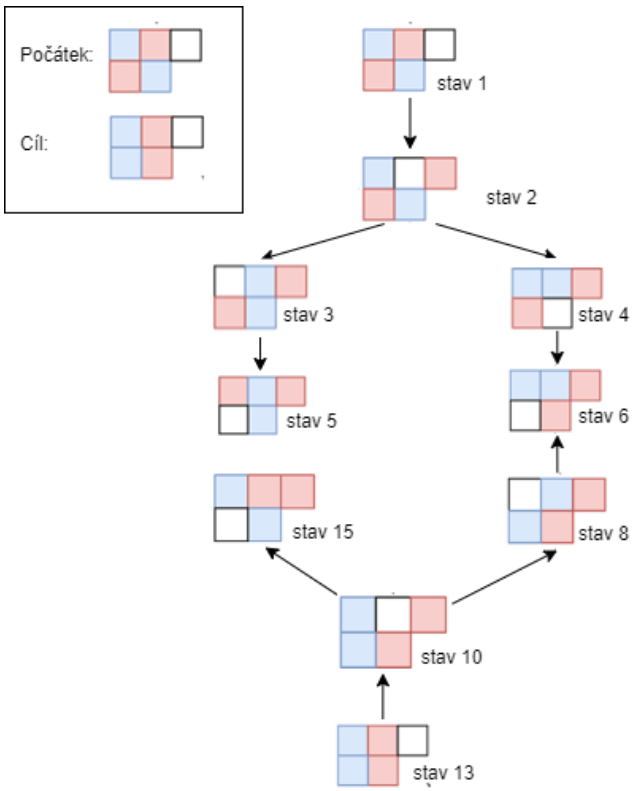
Předpokladem tohoto algoritmu je **znalost** podoby **cílového stavu**. Pokud cílový stav definujeme podmínkami (mat v šachách), tak tento algoritmus nejde použít.

V tomto algoritmu musíme modifikovat testování dosažení cíle. Musíme ho nahradit testováním shody vygenerovaných stavů.

Zároveň zpětná rekonstrukce cesty ze dvou prohledávání je složitější.

V praxi se používají dvě varianty algoritmu:

- Souběžně se spustí prohledávání do šířky (BFS) z počátečního a koncového stavu.
- Souběžně se spustí prohledávání BFS a IDS. Tato varianta má menší paměťové nároky.



2.5.1 Vlastnosti algoritmu

Pokud má každý vrchol **b** následovníků a nejlepší řešení leží v hloubce **d**, tak bude maximálně vygenerován tento počet uzlů:

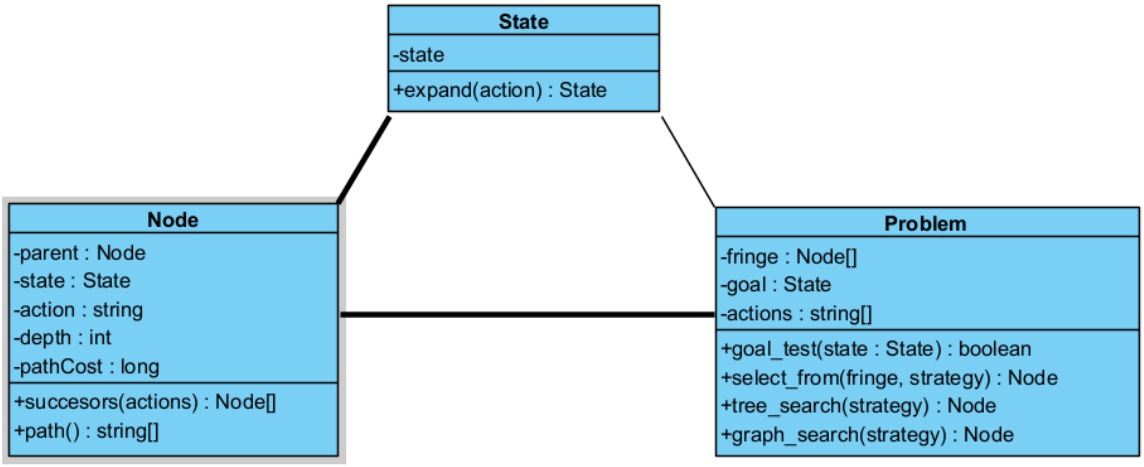
$$N(BIDI) = 2(b + b^2 + \cdots + b^{\frac{d}{2}})$$

	Stromová verze
Časová složitost	$O(b^{\frac{d}{2}})$
Paměťová složitost	$O(b^{\frac{d}{2}})$
Úplnost	Ano
Optimálnost	Ano pro 2xBFS

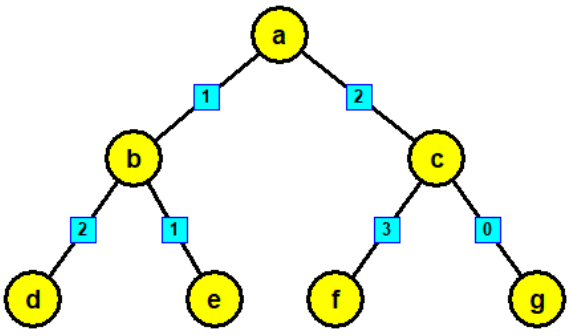
2.6 Prohledávání do šířky dle ceny

Dosud jsme předpokládali, že náklady na změny stavu jsou konstantní. V případě úloh, které lze převést na ohodnocený graf, algoritmus prohledávání musí uvažovat odlišné náklady na změnu stavu. Předpokládáme, že náklady nejsou záporné. Pokud jsou náklady i nenulové, nemůže dojít k zacyklení. To nastává, pokud stavový prostor obsahuje nekonečnou cestu s nulovou cenou.

Z tohoto důvodu musíme rozšířit datový model a do třídy Node přidat atribut pathCost. Atribut obsahuje cenu cesty z kořene k danému stavu. Hodnota je dána funkcí g.



Na tomto ohodnoceném grafu si můžeme zobrazit, jaké hodnoty vrací funkce g pro různé vrcholy. Hodnota se spočítá jako cena cesty do předchůdce uzlu plus cena akce, která převede systém z předchozího stavu do nového stavu. Cenu akce vrací funkce $w(v_i, v_j)$



- g(a) = 0
- g(b) = g(a) + w(a, b) = 0 + 1 = 1
- g(c) = g(a) + w(a, c) = 0 + 2 = 2
- g(d) = g(b) + w(b, d) = 1 + 2 = 3
- g(e) = …

Algoritmus prohledávání podle ceny (Uniform-Cost Search – UCS) je modifikací prohledávání do šířky (BFS). Hlavní úpravou je změna metody select_from, která vybírá ze seznamu uzlů k prohledání uzel s **nejmenší cenou cesty**. Seznam stavů k prozkoumání je vhodné realizovat jako **prioritní frontu** seřazenou podle funkce g.

Po ověření, že stav není cílový, provedeme expanzi uzlu pomocí metody successor, která nám vrátí seznam nových stavů. Pokud nový stav není v seznamu explored ani v seznam fringe, je nový stav vložen do prioritní fronty. Pokud je nový stav již v seznamu fringe a má nový stav lepší ohodnocení než ten v seznamu fringe, nahraď starý stav novým. V opačných případech vygenerovaný stav zahoď.

2.6.1 Vlastnosti algoritmu

Pokud má každý vrchol **b** následovníků a **C*** je cena optimální cesty a ε je nejnižší cena cesty, pak je časová a paměťová náročnost:

	Grafová verze
Časová složitost	$O(b^{1+[\frac{C^*}{\varepsilon}]})$
Paměťová složitost	$O(b^{1+[\frac{C^*}{\varepsilon}]})$
Úplnost	Ano, pokud se každé akci přiřadí nenulová cena.
Optimálnost	Ano

Pokud mají všechny přechody stejnou cenu, pak se algoritmus UCS změní na algoritmus BFS.

3 Informované metody

Informované metody se od neinformovaných liší tím, že pro výběr uzlu k expanzi ze seznamu fringe používáme nějakou dodatečnou znalost.

I u neinformovaných metod jsme využívali nějaké znalosti, ty ale byly součástí zadání a byly zakomponovány do akcí, přechodové funkce nebo reprezentace stavů. Zpravidla se jedná o nějaké omezující podmínky, principy atd. Vyjádřením akcí lze ovlivnit složitost problému. Například u výše uvedeného problému jsme akcí definovali přesun prázdného pole místo přesunu jednoho ze 4 kamenů. Tím se množina akcí velmi zredukovala a problém se stal jednodušším.

U informovaných metod používáme nějaké další znalosti, které nám usnadní průchod stavovým prostorem. Například pokud bychom hledali pramen potoka, tak nebudeme zkoumat cesty, které vedou z aktuálního místa směrem dolů, protože voda teče z kopce.

Jiným příkladem může být logická úloha, kdy máme k dispozici dvě nádoby. Větší nádoba A má objem 4 litry a menší nádoba B má objem 3 litry. Nádoby mají nepravidelný tvar a nejsou na nich žádné míry. Máme k dispozici neomezený zdroj vody. Vodu z nádoby můžeme vylít, nádobu můžeme naplnit nebo můžeme vodu přelít z jedné nádoby do druhé.

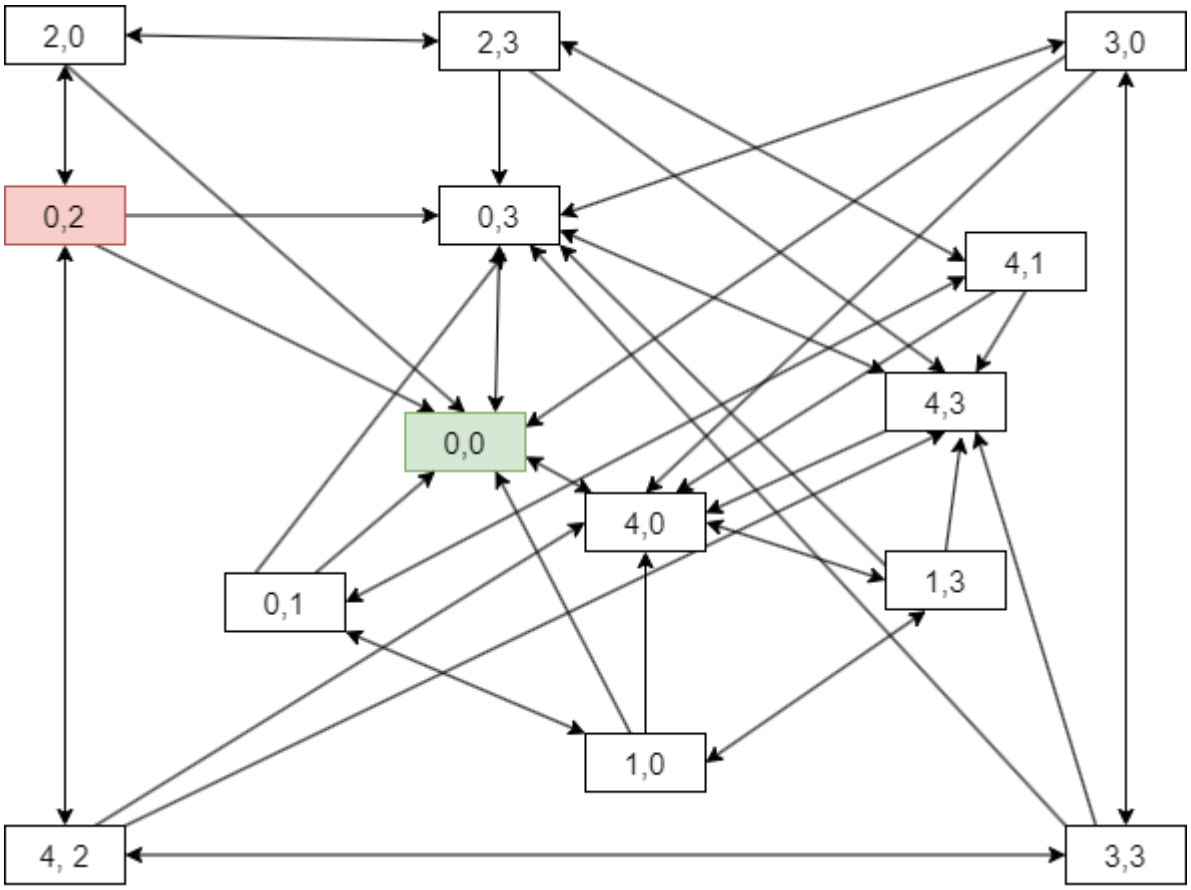
Na začátku jsou obě nádoby prázdné. Našim cílem je, aby nádoba A byla prázdná a nádoba B obsahovala 2 litry vody.

Stav můžeme popsat množstvím vody v nádobách (V_A, V_B).

Akce mohou být:

1. Pokud nádoba A obsahuje vodu ($V_A > 0$) →vylej A
2. Pokud nádoba A není zcela plná ($V_A < 4$) → naplň A po okraj
3. Pokud nádoba B obsahuje vodu ($V_B > 0$) →vylej B
4. Pokud nádoba B není zcela plná ($V_B < 3$)→ naplň B po okraj
5. Pokud B není plná nebo A prázdná ($V_A > 0$ a $V_B < 3$)→ přelij obsah A do B
6. Pokud A není plná nebo B prázdná ($V_A < 4$ a $V_B > 0$) → přelij obsah B do A

Na základě takto definovaných stavů a akcí můžeme vytvořit kompletní stavový graf.



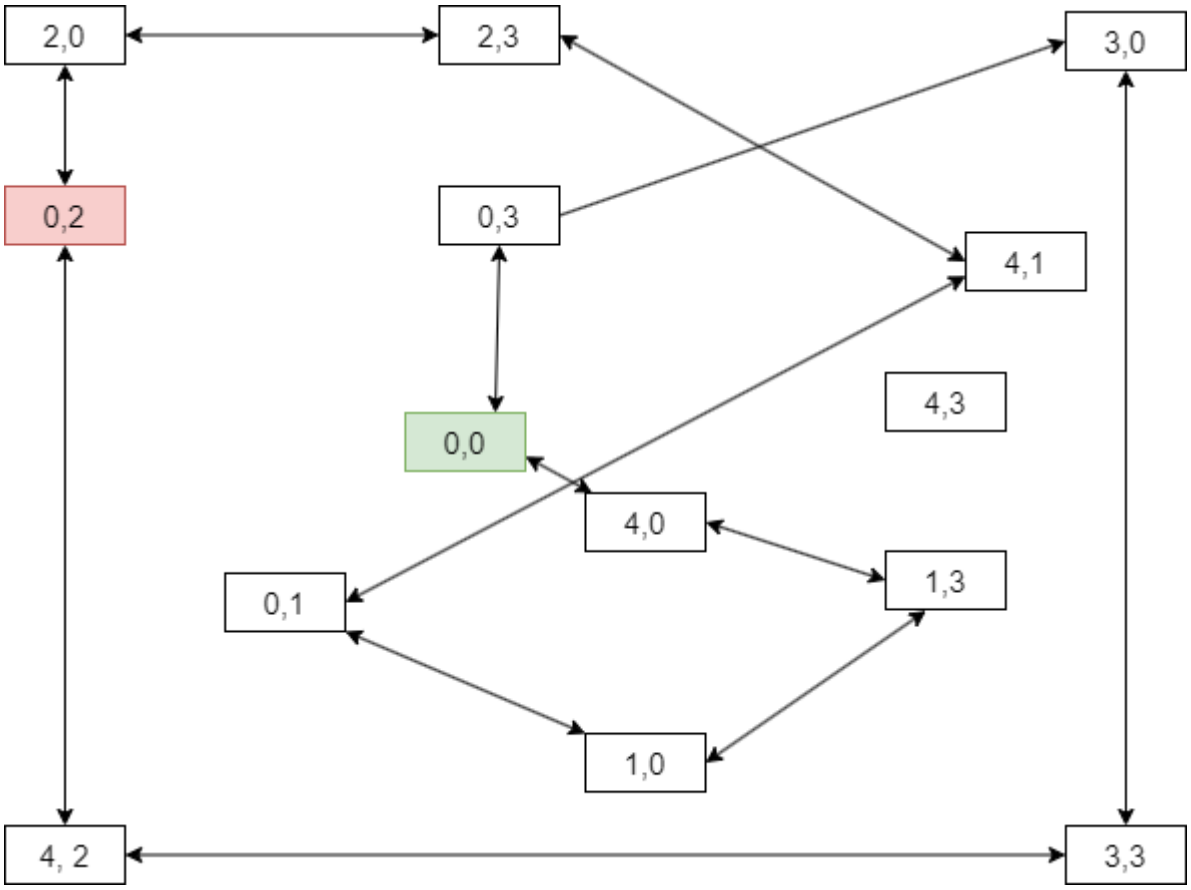
Pokud se nad úlohou zamyslíme nebo nám někdo poradí, tak následující operace nepovedou k cíli.

1. Nevyprazdňuj obě nádoby. Dostaneš se do počátečního stavu.
2. Nenaplňuj obě nádoby. Pak jednu budeš muset stejně vylít.
3. Nenaplňuj nádobu, pokud je druhá prázdná a není-li naplňovaná prázdná.
4. Nevylívej nádobu, je-li druhá plná.
5. Nepřelévej, pokud by potom byla jedna nádoba prázdná a druhá plná.

U akcí se díky radám zpřísní podmínky:

1. ($V_A > 0$ a $V_B > 0$ a $V_B \neq 3$) →vylej A (rady 1, 4)
2. ($V_A < 4$ a $V_B \neq 3$ a ($V_B \neq 0$ nebo $V_A=0$)) → naplň A po okraj (rady 2, 3)
3. ($V_B > 0$ a $V_A > 0$ a $V_B \neq 4$) →vylej B (rady 1, 4)
4. ($V_B < 3$ a $V_A \neq 4$ a ($V_A \neq 0$ nebo $V_B=0$))→ naplň B po okraj (rady 2, 3)
5. ($V_A > 0$ a $V_B < 3$ a $V_A+V_B \neq 3$)→ přelij obsah A do B (rada 5)
6. ($V_A < 4$ a $V_B > 0$ a a $V_A+V_B \neq 4$) → přelij obsah B do A (rada 5)

Stavový diagram se pročistí a nyní je vidět, že k cílovému stavu vedou dvě smysluplné cesty.



3.1 Heuristická funkce

Informované metody využívají další znalosti o řešeném úkolu. Tyto další znalosti lze zakomponovat do heuristické funkce h. Heuristická funkce **odhaduje** cenu cesty z vrcholu v_i do **nejbližšího** cílového vrcholu.

Díky heuristické funkci lze redukovat množství prohledávaných stavů.

Heuristická funkce je problémově závislá. Pro jednu úlohu existuje řada různých možností jejího vyjádření. Různá vyjádření mohou být různě účinná.

Heuristická funkce musí splňovat několik předpokladů:

- Je **nezáporná**
- Pro cílový stav je nulová
- Je **přípustná** – nesmí nadhodnocovat cenu cesty k cíli
- Je **konzistentní** – pro všechny následníky hodnoceného vrcholu musí platit, že hodnota heuristické funkce musí být menší než cena cesty z aktuálního vrcholu do následníka plus hodnota heuristické funkce pro následovníka. Každá konzistentní funkce je zároveň přípustná.

$$h(v_i) \leq c(v_i, a, v_j) + h\,(v_j)$$

Informované metody si dále budeme demonstrovat na zjednodušené verzi hry patnáctka. Zjednodušení verze spočívá ve zmenšení hracího pole ze 4x4 na 3x3. Hrací pole obsahuje čísla 1 až 8. Z počátečního rozházeného stavu je třeba čísla seřadit. Čísla lze přemísťovat horizontálně nebo vertikálně pouze do jediného volného pole.

Počáteční a cílový stav mohou vypadat například takto:

7	2	4
5		6
8	3	1

	1	2
3	4	5
6	7	8

Pokud se hru pokusíme vyřešit pomocí klasických neinformovaných metod, pro složitější stavy se dostaneme do stavu, kdy se kořenový strom stavů větví a prohledávání všech možných stavů není výpočetně možné.

Nyní zkusíme navrhnout dvě různé heuristické funkce, které nám pomohou eliminovat stavy, které nemá smysl prohledávat.

3.1.1 Heuristická funkce 1

První heuristickou funkci založíme na předpokladu, že stav, ve kterém je více správně umístěných čísel je blíže k cílovému stavu než stav, kde je více nesprávně umístěných čísel.

7	2	4
5		6
8	3	1

7	1	2
	3	6
5	8	4

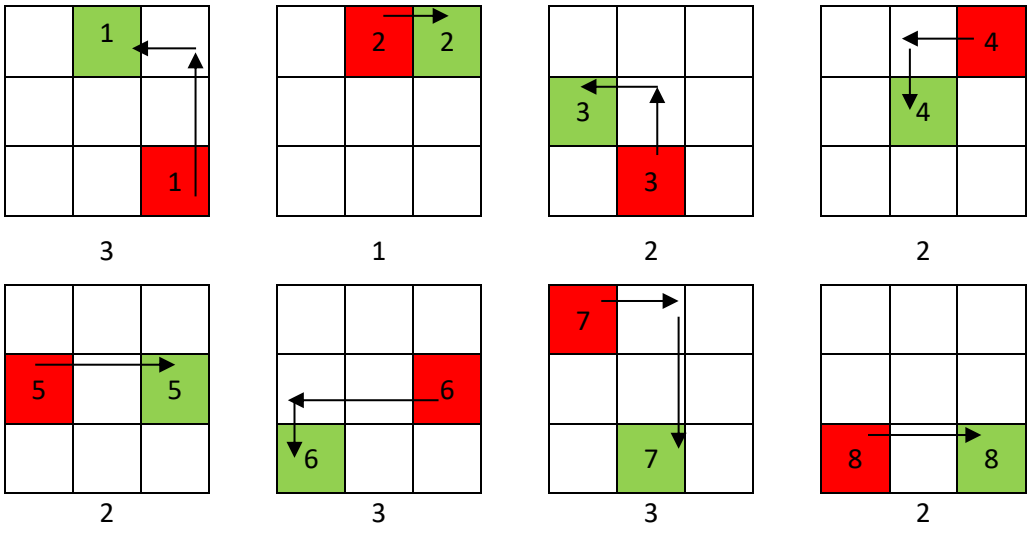
1		2
3	4	5
6	7	8

Pro výše uvedené stavy jsou hodnoty heuristické funkce 8, 6 a 1.

Tato funkce je přípustná a konzistentní.

3.1.2 Heuristická funkce 2

Ve druhé heuristické funkci budeme pro každé číslo počítat minimální počet tahů, kterými se dostane na správnou pozici. Jinak řečeno, jak daleko je číslo vzdáleno od správné pozice, počítáno v manhattanské metrice.



$$h = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$$

Pro počáteční stav druhá heuristická funkce vrací hodnotu 18. Ve skutečnosti optimální řešení vyžaduje 26 akcí. Funkce je tedy přípustná.

3.1.3 Tvorba přípustných heuristických funkcí

Heuristickou funkci lze vytvořit pomocí relaxace (uvolnění) problému. Jedná se u zjednodušení původní úlohy, zmírněním omezujících podmínek.

Například první heuristická funkce, která počítala počet špatně umístěných čísel předpokládá, že je povolená přímé přemístění čísla na pozici v rámci jednoho tahu.

Druhá heuristická funkce předpokládá, že se lze pohybovat v horizontálním a vertikálním směru, ale v cestě nestojí žádné jiné číslo.

Heuristická funkce tedy přináší do stavového prostoru další hrany, akce, které sice nejsou platné, ale napovídají, kterým směrem nejspíše leží cílový stav.

3.2 Uspořádané hledání

Nyní, když víme, co je heuristická funkce, tak se podíváme na uspořádané hledání (Best first search - BFS). Obecně metody spadající do BFS vychází ze stejného algoritmu jako prohledávání podle ceny (Uniform-Cost Search – UCS). Vrcholy v seznamu fringe jsou seřazeny podle hodnoty heuristické funkce. Algoritmy expandují nejprve stavy, u kterých heuristická funkce předvídá, že povedou nejrychleji k cílovému stavu.

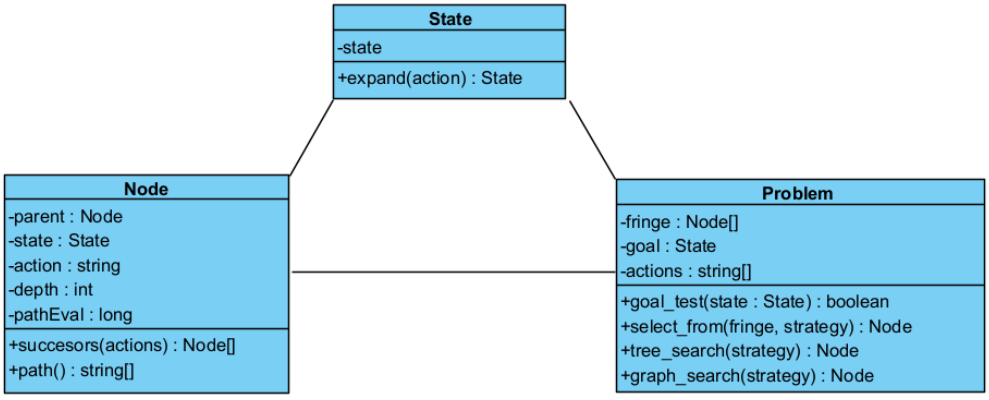
Podíváme se na hladový algoritmus a A*.

3.2.1 Hladové heuristické hledání

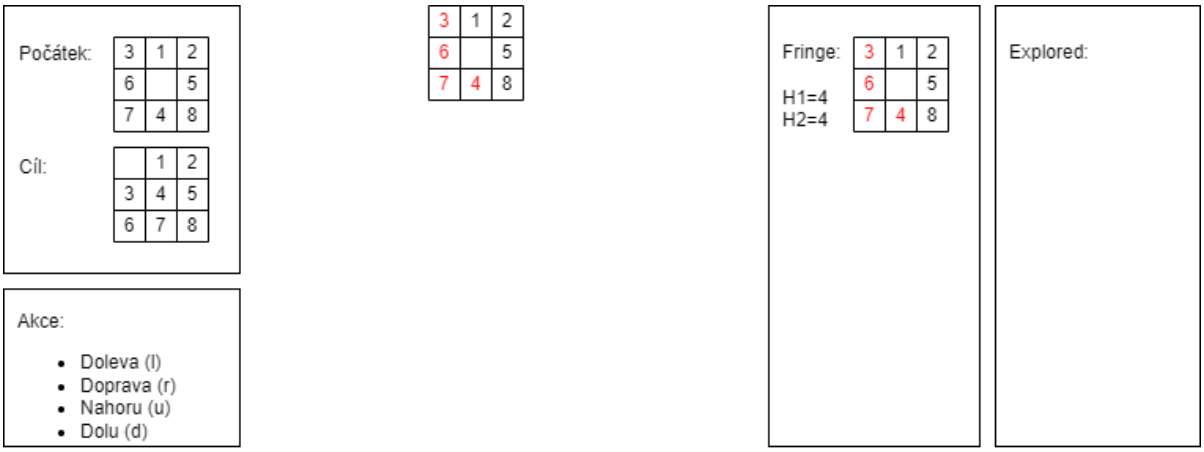
Hladové heuristické hledání (Greedy best-first search) expanduje ze seznamu fringe vrchol, který pravděpodobně povede nejrychleji k cíli. Vzorec hodnotící funkce je tedy jednoduchý, uvažuje pouze hodnotu heuristické funkce v pro daný stav / vrchol.

$$f(v_i) = h(v_i)$$

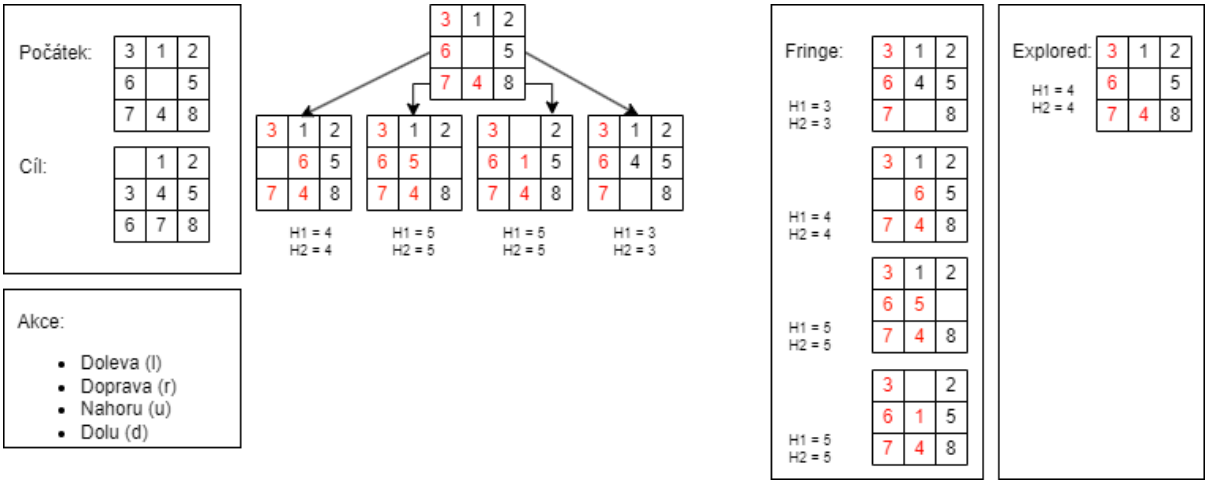
Podobně jako u algoritmu UCS, rozšíříme class diagram o atribut pathEval, který bude obsahovat odhad nákladů z vrcholu do nejbližšího cílového stavu.



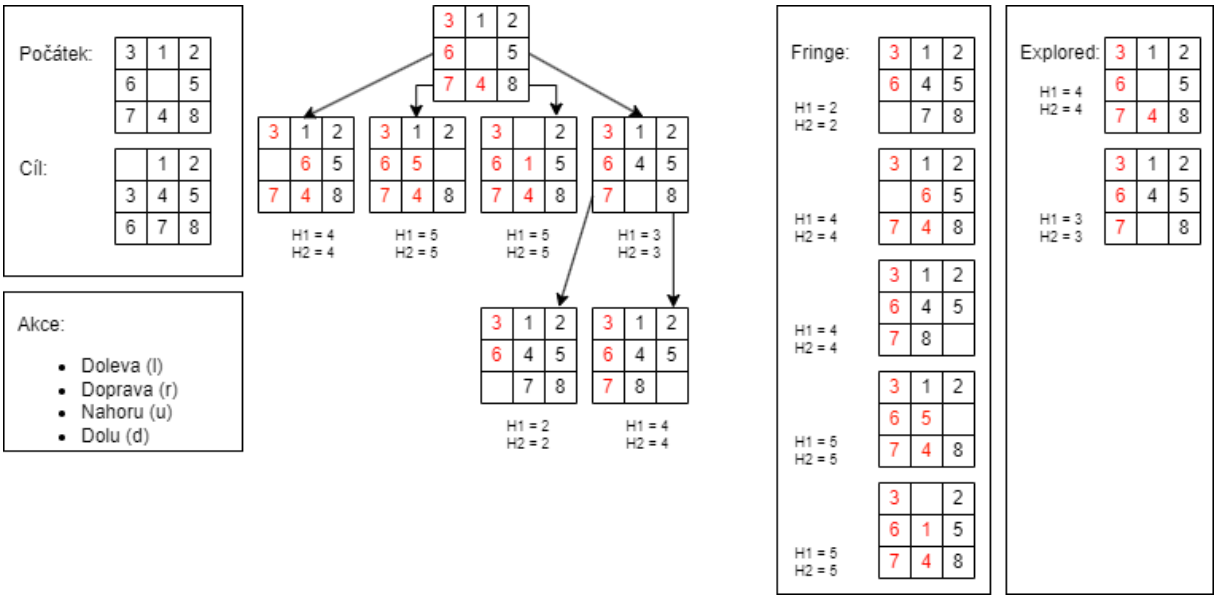
Algoritmus si ukážeme na příkladu zjednodušené hry patnáctka. V počátečním stavu je na začátku v seznamu Fringe. Protože se jedná o jednoduchý počáteční stav, kdy do cílového stavu zbývají 4 tahy, hodnoty obou heuristických funkcí se nebudou lišit. Proto bude jedno, kterou budeme v algoritmu používat. Ve cvičení si na složitějším zadání vyzkoušíme vliv heuristické funkce na rychlost algoritmu.



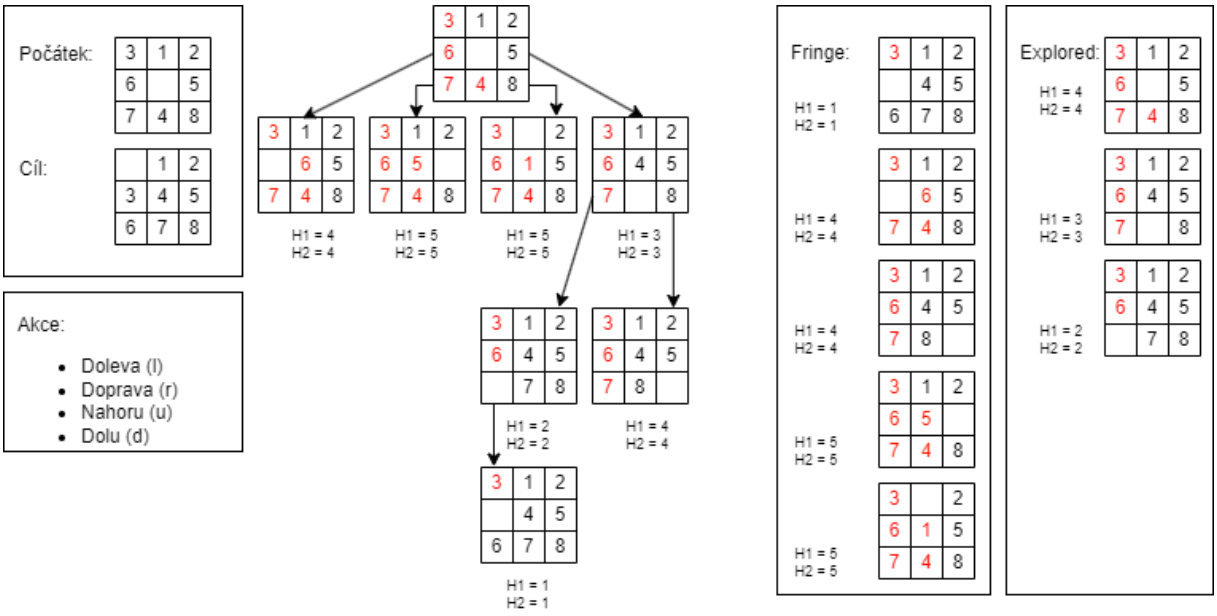
Expandujeme počáteční stav, vložíme ho do seznamu explored. Pro každý nový stav vypočítáme hodnotu heuristické funkce, ta u hladového algoritmu představuje i hodnotu hodnotící funkce. Nové stavy vložíme do seznamu fringe.



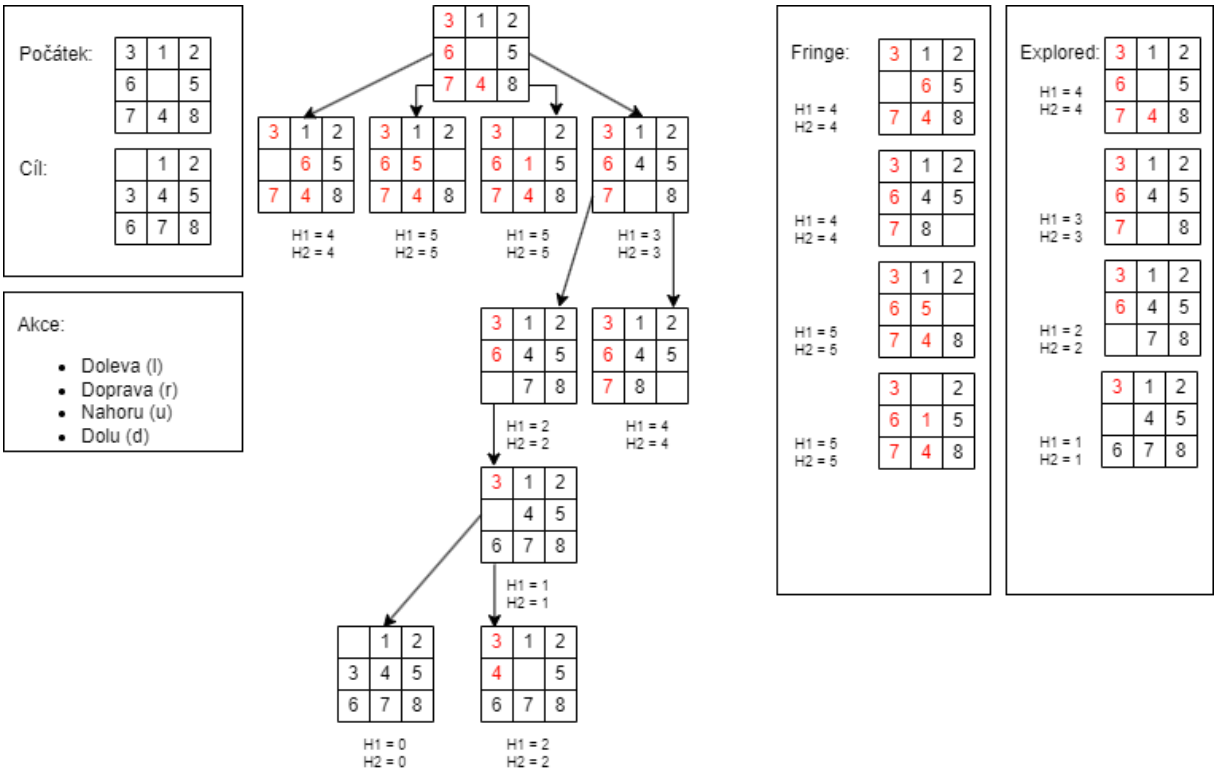
Ze seznamu fringe vyzvedneme stav s nejnižším odhadem nákladů na dosažení cílového stavu. Pokud jsme nové stavy vkládali tak, že je seznam seřazený, vyzvedneme první stav. Pokud je seznam fringe neseřazený, musíme ho celý projít a vybrat stav s nejnižší hodnotou hodnotící funkce. V našem případě budeme expandovat stav vzniklý akcí posun mezery dolů, který má hodnotu heuristické funkce 3. Možné jsou pouze 3 akce, přičemž akce nahoru povede na předchozí stav, který je již v seznamu explored. Do seznamu fringe se nám tedy dostanou stavy vzniklé akcí doleva a doprava.



Ze seznamu fringe vyzvedneme stav s hodnotou hodnotící funkce 2 a vložíme ho do seznamu explored. Možné jsou pouze 2 stavy, přičemž jeden je již v seznamu explored. Do fringe vložíme tedy pouze jeden stav vzniklý akcí nahoru s hodnotou hodnotící funkce 1.



Ze seznamu vyzvedáme stav s hodnotou 1. Vkládáme ho do seznamu explored. Provedeme jeho expanzi. Stav vzniklý akcí dolů je již v seznamu explored. Stav vzniklý akcí doprava má hodnotu hodnotící funkce 2 a vygenerujeme i cílový stav s hodnotu heuristické funkce 0. Pokud testujeme cílový stav při vkládání do seznamu fringe, tak v tomto kroku ukončíme algoritmus.



3.2.2 Vlastnosti algoritmu

Pokud každý vrchol má maximálně **b** následovníků a nejdelší cesta ve stavovém prostoru má délku **m**, počet uzlů, které budou maximálně vygenerovány je následující:

	Stromová verze	Grafová verze
Časová složitost	$O(b^m)$	$O(b^m)$
Paměťová složitost	$O(b^m)$	$O(b^m)$
Úplnost	Ne	Ano, v konečném stavovém prostoru
Optimálnost	Ne	Ne

Dobře definovaná heuristická funkce může značně zredukovat časovou a paměťovou náročnost.

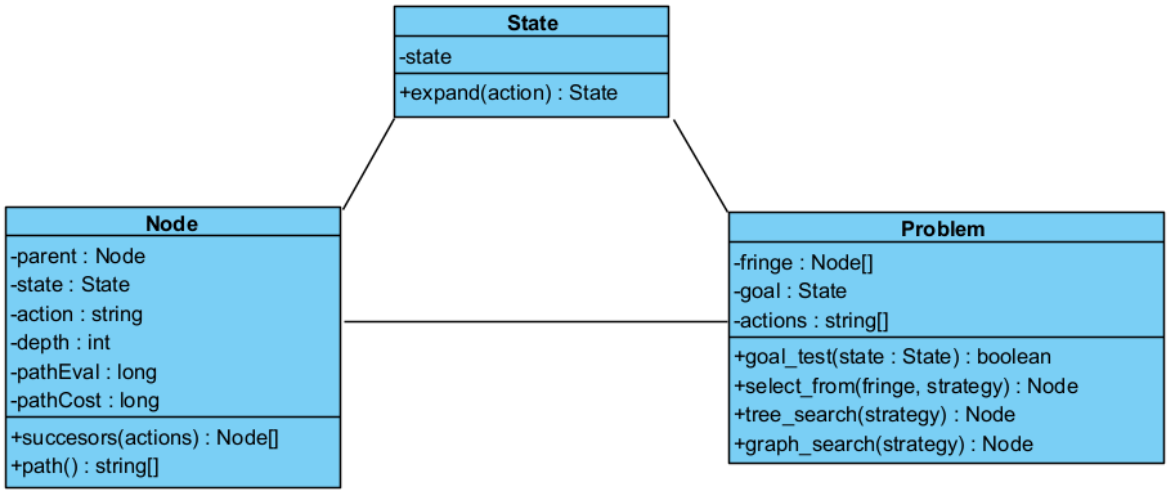
3.3 Algoritmus A*

Jednou z nejpoužívanější informovanou metodou je A* algoritmus. Opět využívá hodnotu heuristické funkce k výpočtu hodnotící funkce.

Jeho strategie je odlišná od hladového algoritmu, který expanduje stav, který má nejlepší odhad ceny do cílového stavu z daného vrcholu. Strategií algoritmu A* je expanze s odhadovanou nejnižší **celkovou** cenou cesty. Do hodnotící funkce započítává nejen heuristickou funkce h, která odhaduje náklady z vrcholu v_i do cílového stavu, ale i dosavadní náklady g na dosažení vrcholu v_i z počátečního vrcholu.

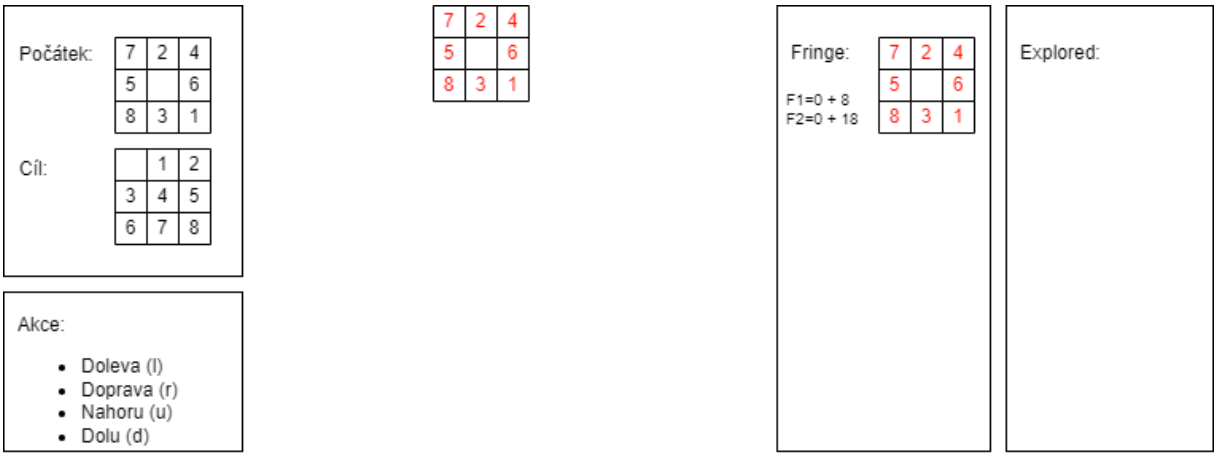
$$f(v_i) = g(v_i) + h(v_i)$$

Z tohoto důvodu bude třeba rozšířit class diagram o uložení dosavadních nákladů. V případě, že se jedná o neohodnocený graf, tak se dosavadní náklady rovnají hloubce uzlu. V případě orientovaného grafu se bude jednat o sumu nákladů jednotlivých akcí.

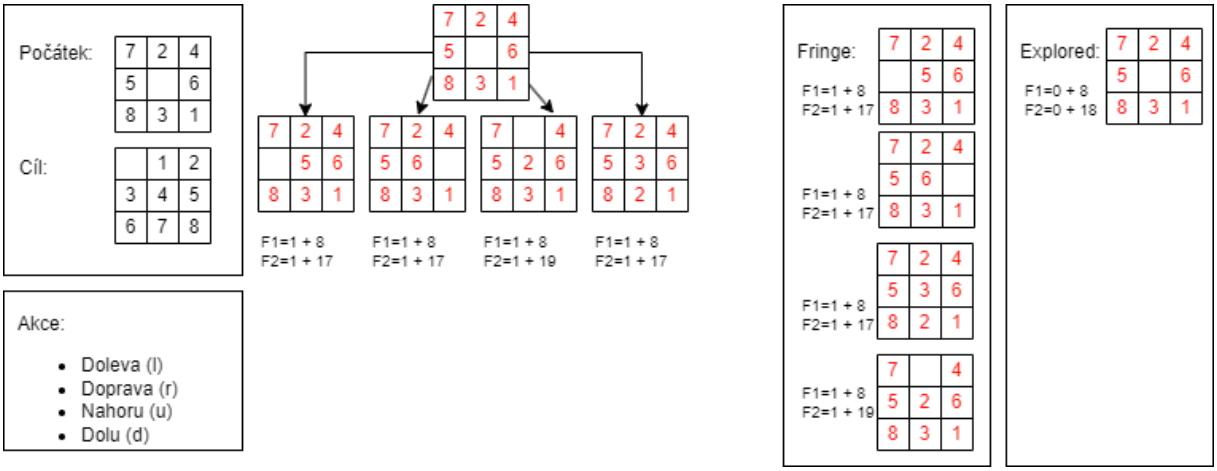


Algoritmus A* si ukážeme opět na zjednodušené hře patnáctka. Tentokrát je počáteční stav vzdálený 26 tahů od cílového stavu. U každého stavu bude uvedená hodnota hodnotící funkce F_1, která používá heuristickou funkci 1 a hodnota funkce F_2, která používá heuristickou funkcí 2. Pro expanzi budeme brát v úvahu hodnotu funkce F_2.

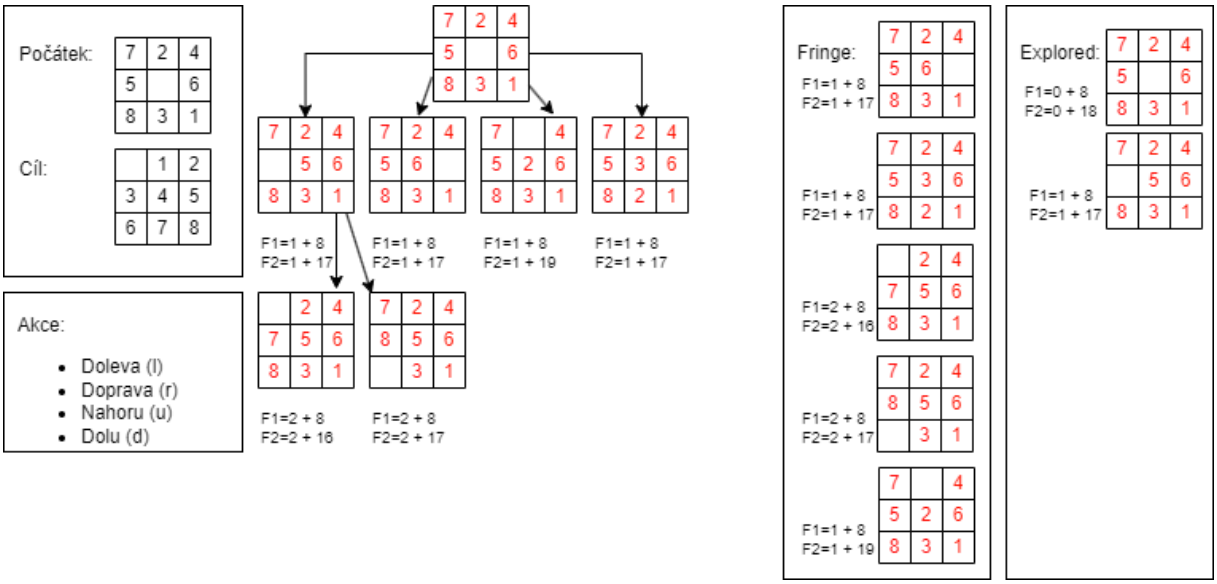
Na počátku seznam k expanzi obsahuje počáteční stav, který má hodnotu funkce $F_1 = 0 + 8$, protože vrchol se nachází v hloubce 0 a všech 8 čísel je umístěných špatně. Hodnota funkce $F_2 = 0 + 18$, protože hloubka vrcholu je 0 a je třeba 18 tahů na přemístění čísel, pokud budeme předpokládat, že žádnému tahu nebrání žádné jiné číslo.



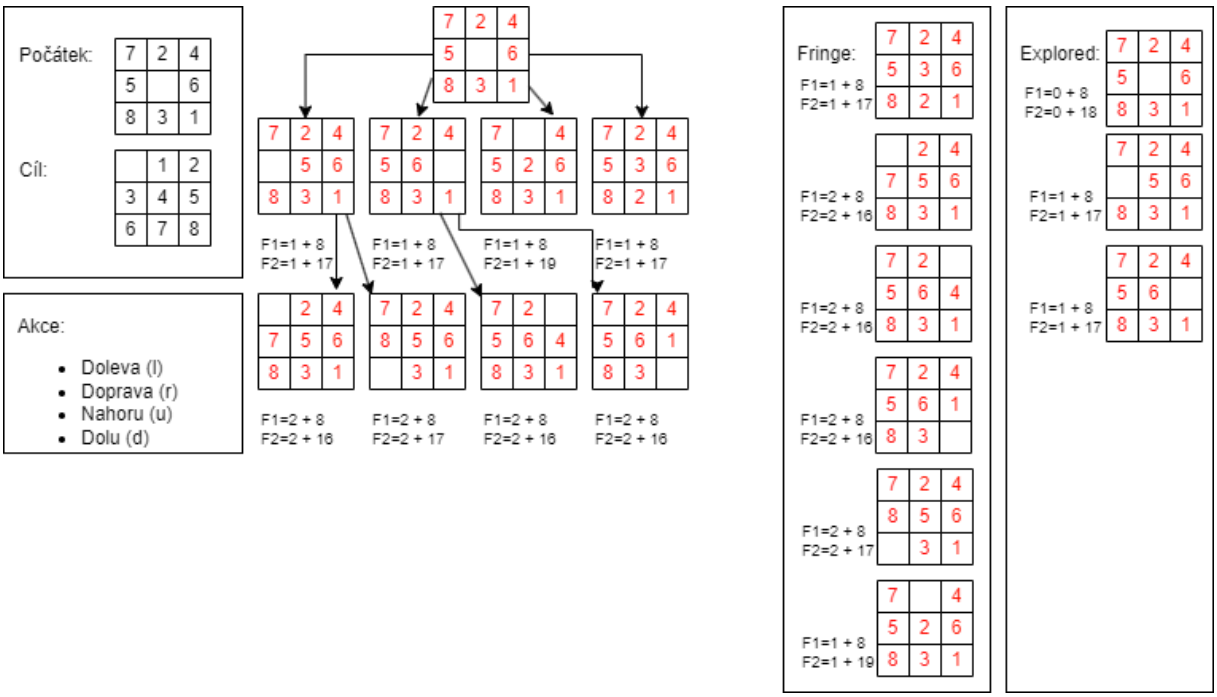
Počáteční stav vyjmeme ze seznamu fringe, vložíme ho do seznamu explored a provedeme jeho expanzi. Jsou možné všechny 4 akce, takže získáváme 4 nové stavy. Všechny nové stavy mají hodnotu první heuristické funkce 8. Za to druhá heuristická funkce pro akci nahoru vrací hodnotu 19 – tedy zhoršení proti původnímu stavu. Všechny nové stavy vložíme do fringe a seřadíme podle F_2.



Ze seznamu fringe vyjmeme první vrchol, který vložíme do seznamu explored. Provedeme jeho expanzi. Možné jsou 3 akce, ale jedna povede na původní stav, který je v seznamu explored. Do seznamu fringe tedy vložíme pouze dva stavy vzniklé akcemi nahoru a dolů. Druhá heuristická funkce h_2 pro ně vrací hodnoty 16 a 17. Protože se jedná o vrcholy s hloubkou 2, nové stavy zařadíme až za stavy s hloubkou 1 a hodnotou heuristické funkce 17.



V tomto okamžiku se algoritmus A* začne chovat odlišně od hladového algoritmu. Hladový algoritmus by vybral vrchol s hodnotou heuristické funkce h_2 rovné 16. Algoritmus A* uvažuje i cenu dosavadní cesty, tedy v případě neohodnoceného algoritmu hloubky vrcholu a dá přednost nejprve dvou vrcholům v hloubce 1 a hodnotou h_2 rovnou 17.



Dále v algoritmu pokračovat nebudeme, protože počáteční stav je příliš vzdálen od cílového stavu. Dokončení algoritmu je ve cvičení.

3.3.1 Vlastnosti algoritmu

Hlavním výhodou algortitmu A* je jeho úplnost a optimálnost. Tedy algoritmus najde řešení a toto řešení bude optimální.

	Grafová verze
Úplnost	Ano
Optimálnost	Ano

Paměťová a časová náročnost je závislá na kvalitě heuristické funkce. Toto vyjádření je závislé na stavovém prostoru úlohy, jedná se však o **exponenciální** závislost. U složitějších úloh je hlavním problémem nedostatek paměti.

Existují i modifikace A* algoritmu

- Iterative deeping A* (IDA*) – založený na stejné myšlence jako IDS
- Memory bounded A* (MA*) – A*, který se dokáže přizpůsobit podle množství dostupné paměti
- Simplified Memory bounded A* (SMA*) – A*, který se dokáže přizpůsobit podle množství dostupné paměti

3.4 Efektivnost heuristické funkce

Pro stejnou úlohu jde navrhnout více heuristických funkcí, jejich kvalitu lze měřit podle efektivního faktoru větvení.

Předpokládejme, že A* vygeneruje při hledání řešení N vrcholů a řešení se nachází v hloubce d. b^* odpovídá faktoru větvení vyváženého stromu o N+1 vrcholech, který by vznikl aplikací neinformovaného algoritmu na úlohu s řešením v hloubce d.

$$N = b + b^2 + b^3 + \cdots + b^d$$

Například pokud algoritmus A* vygeneruje 52 vrcholů k nalezení řešení v hloubce 5, pak b* odpovídá:

$$52 = b^* + (b^*)^2 + (b^*)^3 + (b^*)^4 + (b^*)^5$$

$$b^* = 1{,}92$$

Heuristická funkce s nižším b^* je efektivnější. Protože snižuje počet větví, které je třeba prozkoumat. Efektivní faktor větvení ideálně navržené heuristické funkce se shora blíží 1.

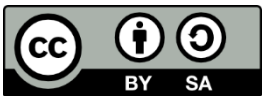
Vektor větvení b* se může pro danou heuristickou funkci a daný problém lišit i v závislosti na hloubce řešení.

Následující tabulka ukazuje statistiky pro 1200 náhodně vygenerovaných stavů zjednodušené hry patnáctka. Délka řešení byla od 2 do 24 tahů.

	N			b*		
	IDS	A* (h_1)	A* (h_2)	IDS	A* (h_1)	A* (h_2)
2	10	6	6	2,45	1,79	1,79
4	112	13	12	2,87	1,48	1,45
6	680	20	18	2,73	1,34	1,30
8	6 384	39	25	2,80	1,33	1,24
10	47 127	93	39	2,79	1,38	1,22
12	36 106	227	73	2,78	1,42	1,24
14		539	113		1,44	1,23
16		1 301	211		1,45	1,25
18		3 056	363		1,46	1,26
20		7 276	676		1,47	1,27
22		18 094	1 219		1,48	1,28
24		39 135	1 641		1,48	1,26

UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vytvořeno v rámci projektu **Digitalizace studijních Agend, Nové TechnologiE, systémy a přístupy k výuce na UPCE**, reg. č. NPO_UPCE_MSMT-16591/2022.



Financováno Evropskou unií
NextGenerationEU

Národní plán obnovy

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Machine learning & AI

Teorie her

Ing. Martin Pozdílek, Ph.D.

1 Teorie her

Teorie her je jedna z matematických oblastí, která se zabývá studiem **rozhodování v situacích**, kdy se jeden nebo více hráčů snaží **dosáhnout svých cílů** v prostředí, kde jejich akce **ovlivňují výsledky pro všechny zúčastněné**. Hry mohou zahrnovat konfliktní situace, kde zájem jednoho hráče je v rozporu se zájmy ostatních, nebo situace spolupráce, kde hráči spolupracují s cílem dosáhnout společného cíle.

Hrami se lidstvo zabývá od nepaměti, a to nejen těmi deskovými, ale hlavně těmi reálnými, kdy je třeba zvolit správnou strategii vůči protivníkovi a rozhodovat se, co nejlépe. Historie teorie her sahá až do 18. století, ale tato matematická disciplína získala svůj moderní charakter a význam až v 20. století.

První zmínky o teorii her lze najít u matematika a filozofa Blaise Pascala, který se zabýval problémem hazardu a pravděpodobnosti. Také Johann Bernoulli se zajímal o rozhodování v hazardu a vytvořil první užitkovou funkci, která zkoumala rozhodování v nejistém prostředí.

Teorie her začala získávat matematickou formu v 19. století díky práci matematika Augusta Cournota, který se zabýval teorií hospodářského chování. Cournotův model konkurence je považován za jeden z prvních modelů veřejného dobra a byl prvním krokem směrem k moderní teorii her.

Teorie her výrazně pokročila v 20. století. John von Neumann, matematik a fyzik, a Oskar Morgenstern, ekonom, publikovali v roce 1944 knihu „Teorie her a ekonomické chování“, která byla průlomem v oblasti teorie her. Tato kniha formalizovala koncepty strategií, rovnováhy a základních her. Von Neumann a Morgenstern také zavedli koncept Nashovy rovnováhy, který je stále základním kamenem teorie her. Po publikaci knihy von Neumanna a Morgensterna se teorie her rozvíjela ve více směrech. John Nash zformuloval pojmy, jako je Nashova rovnováha ve smíšených strategiích, což rozšířilo aplikace teorie her na různé oblasti. V následujících desetiletích se teorie her stala klíčovým nástrojem v ekonomii, biologii, politice, informatice a dalších disciplínách. Její matematické metody a koncepty umožňují lepší porozumění strategickým situacím a predikci chování v těchto situacích.

Teorii her můžeme aplikovat v biologii, zejména v evoluční biologii a ekologii, kde pomáhá modelovat a zkoumat strategické interakce mezi organismy. Pomocí ní můžeme vytvářet modely mezi predátorem a kořistí, evoluční spolupráci, konkurenci v populaci, kooperaci mezi druhy atd.

Jiným oborem může být například ekonomie, kdy nám teorie her umožňuje modelovat vzájemnou konkurenci mezi firmami, vztahy mezi firmami, zaměstnanci a odbory, chování na aukcích nebo pochopení chování oligopolů.

My si ovšem teorii her budeme vysvětlovat na deskových hrách, které nám pomohou snáze pochopit principy této teorie.

1.1 Pojmy a základní koncepty z teorie her

Hra je strategická interakce dvou a více hráčů. V teorii her můžeme vytvořit model jejich rozhodování a vybírat strategii, která povede k dosáhnutí našich cílů.

Každý účastník hry se nazývá **hráčem**. Každý hráč má své vlastní strategie, které může zvolit, aby ovlivnil výsledek hry. **Strategie** je soubor akcí, které hráč může vybrat v daném okamžiku. Strategie určuje, jak hráč reaguje na různé situace ve hře. V každém stavu hry má hráč dostupné **tahy**, tedy akce, které jsou přípustné varianty chování hráče.

Při modelování v teorii her předpokládáme **racionalitu** hráčů. Výjimkou jsou hry proti přírodě. Od racionálního chování hráče očekáváme následující:

- Hráč si volí cíle hry na základě **stabilních preferencí**. Jeho preference se v průběhu hry nemění.
- Hráč volí takové tahy, aby cílů dosáhl co **nejefektivnějším způsobem**. Pokud tento předpoklad neplatí, má hráč jiné tajné cíle nebo se nechová racionálně.
- Hráč dokáže situace, se kterými je konfrontován, **seřadit podle** svých **preferencí** od nejvýhodnějších po nejméně výhodnou. Aby toto řazení bylo racionální, musí být **úplné**, tedy musí pokrývat všechny situace. Zároveň toto řazení musí být **tranzitivní**. Pokud hráč preferuje situaci A před situací B a zároveň preferuje situace B před situací C, tak z musí preferovat i situaci A před C.

V reálných situacích se hráč z mnoha důvodů může začít chovat iracionálně. V tomto případě předpovědi modelu chování nemusí být přesné.

Další součástí teorie her jsou **výplaty**, reprezentované **užitkovou**, účelovou **funkcí**. Jedná se o číselné ohodnocení koncového stavu – výsledku hry. Výplaty jsou hodnoty, které určují výsledek hry pro každého hráče v závislosti na zvolených strategiích všech hráčů. Výplaty mohou být vyjádřeny jako peněžní hodnoty, body, úspěšnost atd. Výplaty jsou určeny preferencemi hráče. Předpokládáme, že **hráč se snaží maximalizovat užitkovou funkci**.

Teorie her zahrnuje různé typy informací, které mají hráči o hře. Může jít o **úplné informace** (když hráči znají všechny detaily o hře) nebo **neúplné informace** (když hráči nemají kompletní informace o akcích a strategiích ostatních hráčů).

Hlavním cílem teorie her je nalezení různých druhů rovnováhy. Například Nashova rovnováha popisuje situace, kdy hráči nemají důvod změnit své strategie, pokud ostatní hráči zůstanou při svých strategiích.

1.2 Typy her

Hry můžeme dělit podle mnoha různých kritérií:

- Počet hráčů
  - Dva
  - Více hráčů
- Počet přípustných tahů
  - **Konečné hry** – všichni hráči mají konečné množství přípustných tahů, každý hráč se snaží **vyhrát**. Jedná se o naprostou většinu reálných problémů a her.
  - **Nekonečné hry** – alespoň jeden hráč má nekonečné množství přípustných tahů, cílem hráče s nekonečným množstvím tahů je **udržet se ve hře**.
  - **Opakovaná hra** – zvláštní typ nekonečné hry**.** Hra se hraje opakovaně mezi dvěma nebo více hráči na více kol. Jedno kolo hry je podobné běžné hře. Hráči v dalším kole mají možnost změnit strategii na základě minulých výsledků.
- Míra konkurence
  - **Hry s nenulovým součtem** – zisk jednoho hráče nemusí znamenat ztrátu pro ostatní hráče. Jedná se o různá vyjednávání v politice, podnikání apod. Snaha je nalézt řešení, kdy každá strana něco získá, i když se nemusí jednat o její potenciální maximální zisk.
  - **Hry s nulovým součtem** – jedná se o kompetitivní hry, kdy hráč vítězí na úkor protihráčů. Zpravidla se jedná o většinu deskových her (šachy, piškvorky, …)
- Způsob hraní
  - **Hry jednotahové** – hráči provedou svůj tah současně (kámen, nůžky, papír)
  - **Hry sekvenční** – hráči se střídají v tazích (šachy, dáma, poker, …)
- Typ informace
  - **Hry s úplnou informací** – všechny informace potřebné pro rozhodování jsou dostupné (šachy)
  - **Hry s neúplnou informací** – některé informace jsou skryté (poker)
- Míra náhody
  - **Hry deterministické** – ve hře se nevyskytuje prvek náhody (šáchy, dáma, …)

- **Hry stochastické** – do hry zahrnuje prvek náhody (monopoly, poker, karetní hry, hry s kostkou, …)

- Možnost komunikace mezi hráči
  - **Hry nekooperativní**
  - **Hry kooperativní**
  - **Hybridní**

1.3 Příklady

1.3.1 Vězeňské dilema

Představte si dvě osoby, které byly chyceny při spáchání zločinu. Oba obvinění jsou drženi odděleně a nemají možnost komunikace, a tedy možnost dohodnout strategii. Důkazy, které má policie, nejsou dostatečné pro usvědčení, takže se musí spoléhat na jejich přiznání.

Každý z obviněných má dvě možnosti: může buď mlčet a nepovědět přihoršující informace o svém spolupachateli, nebo může spolupracovat a zradit partnera a svědčit proti němu.

Výplatní matice může vypadat následovně:

	Bob mlčí	**Bob mluví**
Alice mlčí	Oba 2 roky, protože není dostatek důkazů	Alice 10 let, Bob je volný.
Alice mluví	Bob 10 let, Alice je volná.	Je více důkazů, oba 6 let.

Předpokládáme, že každý vězeň se stará o svůj prospěch a nebere ohledy na ostatní hráče. Jakou strategii má každý z „hráčů“ zvolit?

Alice uvažuje následovně: Pokud bude Bob mlčet a já také, dostanu trest na 2 roky, takže lepší bude mluvit, protože budu dříve volná. Pokud bude Bob mluvit a já mlčet, dostanu 10 let, takže lepší bude vypovídat, protože dostanu jen 6 let jako on za to, že vypovídal rovněž.

Stejně uvažuje i Bob, takže pokud oba udělají víceméně racionální rozhodnutí, tj. budou oba dva vypovídat, dostanou oba trest v délce 6 let, přestože optimálním rozhodnutím je vůbec nevypovídat.

Jak se strategie změní, pokud se tato situace bude opakovat znovu? Jedná se tedy o nekonečnou hru. „Hráči“ si mohou vyvinout strategie, které zahrnují vzájemné dohody nebo tresty za zradu. Analýza opakovaných her je komplexní a zahrnuje koncepty, jako je "tit-for-tat" (hraješ stejně,

jak hrál tvůj protihráč v předchozím kole), který může vést k udržení spolupráce mezi hráči v dlouhodobém horizontu.

1.3.2 Nashova rovnováha

Nashova rovnováha je koncept v teorii her, který popisuje situaci, kdy žádný hráč nemá důvod změnit svou strategii, pokud ostatní hráči zůstanou při svých strategiích. Jedná se o situaci, kdy hráči dosáhnou optimálního výsledku, pokud vezmou v úvahu strategie všech hráčů.

V této jednoduché hře si oba hráči mohou zvolit strategii A, při které získají 1 Kč, nebo strategii B, při které ztratí 1 Kč. Logicky si oba hráči zvolí strategii A a získají výhru 1 Kč.

	Bob A	**Bob B**
Alice A	Alice +1, Bob +1	Alice +1, Bob -1
Alice B	Alice -1, Bob +1	Alice 0, Bob 0

Pokud bychom odhalili strategii Alice Bobovi a naopak, zjistíme, že se žádný z hráčů se neodchýlí od původní volby. Znalost tahu druhého hráče znamená jen málo a nemění chování žádného z hráčů. Strategie A představuje Nashovu rovnováhu. Nashova rovnováha je důležitým konceptem v teorii her a používá se k analýze strategických interakcí mezi hráči v různých typech her. A tou hrou může být i držení atomových zbraní velmocemi, kdy žádná strana se jich nemůže vzdát, protože ta druhá by nad ní měla převahu. Nashovou rovnováhou je tedy strategie mít atomovou bombu.

1.4 Hry řešené v umělé inteligenci.

Existuje celá řada různých her s různými parametry. Pomocí umělé inteligence se zpravidla řeší následující typy her.

- Počet hráčů: 2 a více
- Počet přípustných tahů: konečný, zpravidla malý počet akcí
- Míra konkurence: kompetitivní s nulovým součtem
- Způsob hraní: sekvenční
- Informace: úplné i neúplné
- Míra náhody: deterministické i stochastické hry
- Komunikace: nekooperativní i kooperativní
- Předpokládáme racionalitu hráčů

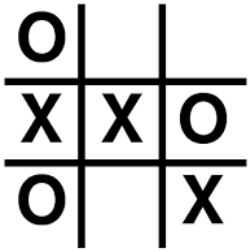
Zpravidla se tedy jedná o hraní sekvenčních kompetitivních her, kde se hráči pravidelně střídají až do ukončení hry. Na konci hry jsou vítězi přiděleny body a poražení jsou penalizováni. Hra je chápána jako problém hledání řešení, který však ovlivňují protihráči. Cílem je nalezení posloupnosti akcí, která povede k vítězství.

1.5 Deterministické hry s úplnou informací

Ve zbytku kapitoly se zaměříme na sekvenční deterministické hry s úplnou informací a ukážeme si algoritmus pro hledání vítězné strategie.

Typickou hrou jsou piškvorky (tic tac toe) hrané na herní ploše 3x3.

- Počet hráčů: 2
- Počet přípustných tahů: konečný
- Míra konkurence: kompetitivní
- Způsob hraní: sekvenční
- Informace: s úplnou informací
- Míra náhody: deterministická
- Komunikace: nekooperativní



1.5.1 Užitková funkce

Užitková funkce vrací ohodnocení konečného stavu hry. Protože užitkovou funkci budeme maximalizovat, výhra musí být ohodnocena výše než remíza a ta zase výše než prohra. U některých her nám vystačí pouze 2 nebo 3 hodnoty. U některých her (vrhcáby), u kterých se snažíme dosáhnout co nejvyššího skóre, může být interval hodnot větší.

	Piškvorky	Šachy	Vrhcáby
Výhra	1	1	<0, 192>
Remíza (pat)	0	0,5	
Prohra	-1	0	

1.5.2 Označení hráčů

V celém textu budeme předpokládat, že algoritmus hraje za hráče, který táhne jako první. Tohoto hráče označíme jako MAX. Hráče, který táhne jako druhý (člověk, druhý algoritmus), budeme nazývat MIN.

1.5.3 Formální definice

Formální definice hry jako hledání řešení problému. Máme definované stavy, akce, přechodovou funkci, a navíc i množinu hráčů a užitkovou funkci.

- S množina všech možných stavů s
- A množina všech možných akcí
- γ přechodová funkce $\gamma(s,\ a) \rightarrow s'$
- s_0 počáteční stav
- KS množina všech koncových stavů, kde $KS \subseteq S$
- P množina všech hráčů p
- U užitková funkce U(s), která hodnotí stav

Při implementaci budeme používat následující funkce

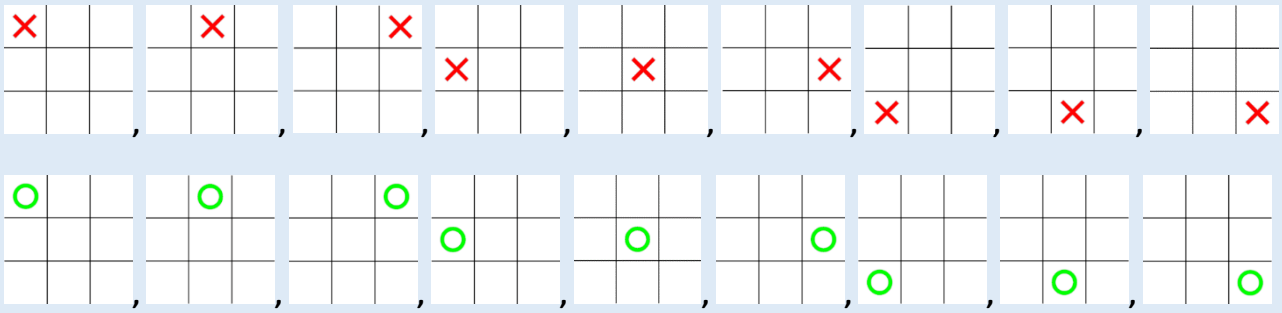
- possible_actions vrací množinu akcí A, které jsou přípustné pro daný stav
- expand(action) vrací následníka stavu **s** získaného aplikací akce **a**
- terminal_test vrací hodnotu, zda je stav konečný a případně, kdo je vítěz
- utility pro stav vrací ohodnocení výsledku z pohledu hráče MAX
- next_player vrací hráče, který bude pokračovat

Mějme například hru piškvorek hranou na herní ploše 3x3.

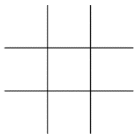
Množina hráčů P bude obsahovat dva hráče. Hráče, který bude začínat označíme za MAX a bude hrát za křížky. Protihráče označíme za MIN a v našem případě bude hrát za kolečka. P={MAX, MIN}.

Množina akcí A obsahuje všechny možné akce, které mohou oba hráči zahrát.

Akce A

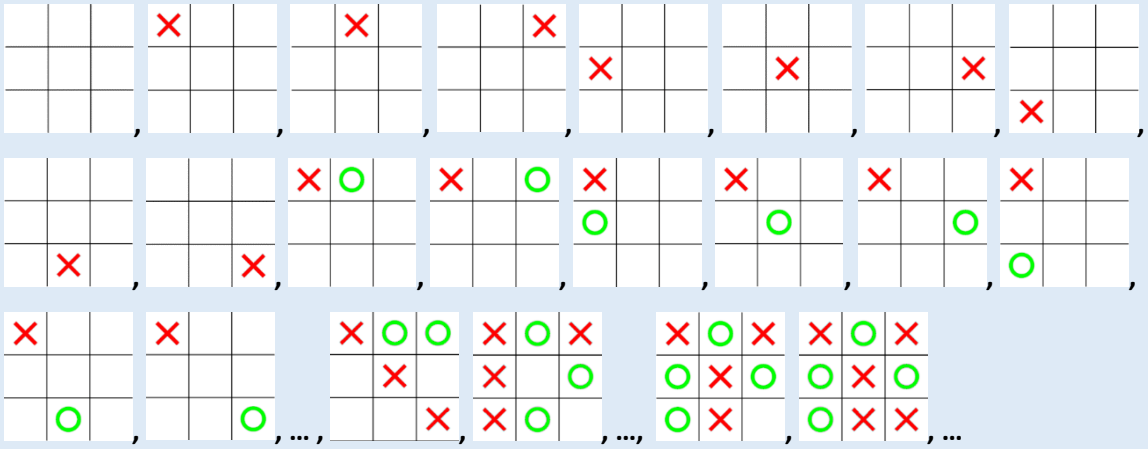


Počáteční stav s_0 vypadá takto:



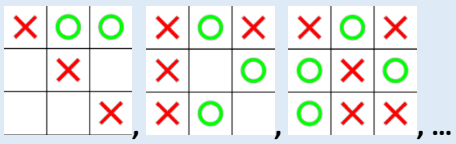
Množina S obsahuje všechny přípustné stavy, které mohou vypadat například takto:

Stavy S

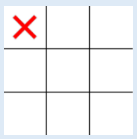
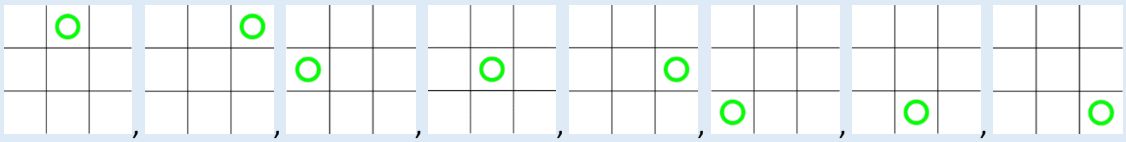


Množina konečných KS obsahuje konečné stavy. U každého takového stavu jsme schopni určit vítěze, případně partii označit za remízu. Množinu těchto stavů můžeme určit výčtem, ale z praktického hlediska ji budeme definovat pravidly, kdy jsou 3 stejné kameny v jedné řadě, sloupci nebo na diagonále.

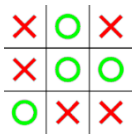
Stavy KS



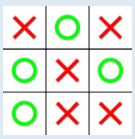
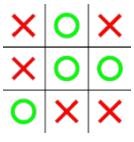
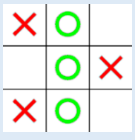
Pro každý stav existuje množina platných akcí, která je podmnožinou všech akcí. Jedná se o akce, které hráč může pro daný stav partie použít. S postupující hrou se tato podmnožina bude zmenšovat.

Stav	Možné akce
	

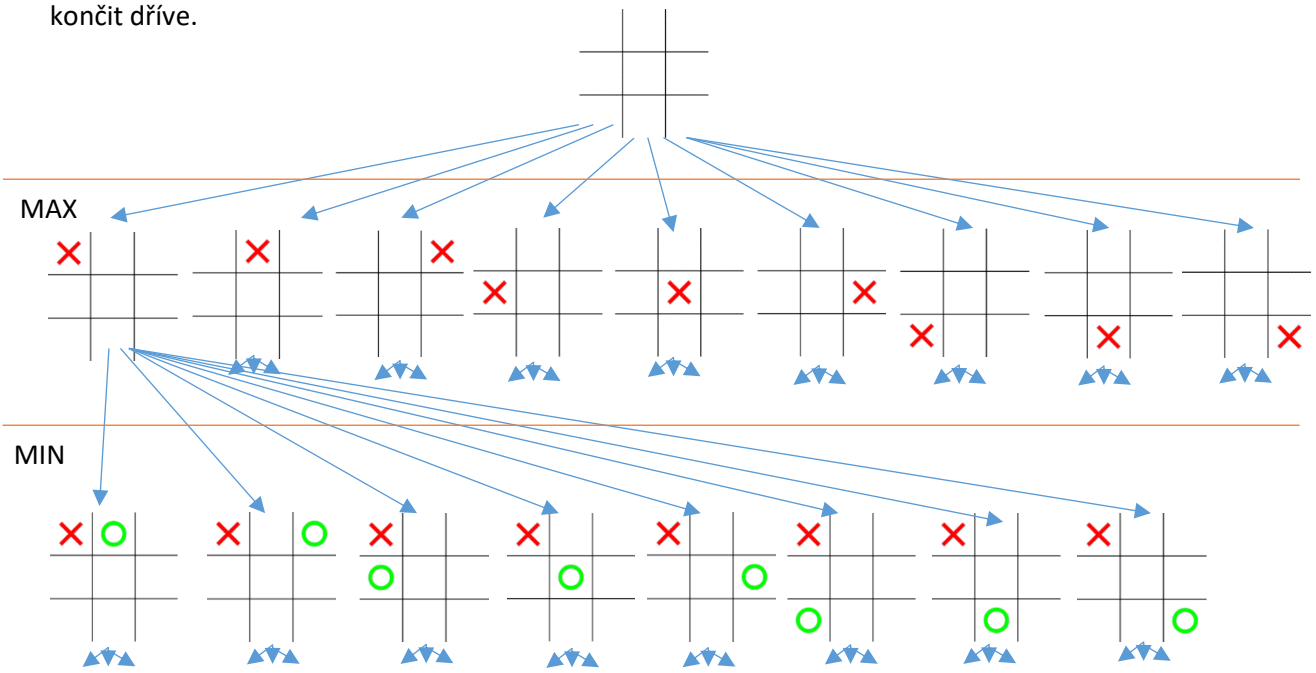
Funkce terminal_test testuje, zda je stav konečný.

Stav	Hodnota
	True, vítězí červený
	True, remíza

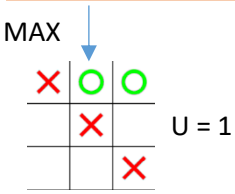
Funkce utility hodnotí stav z pohledu hráče MAX.

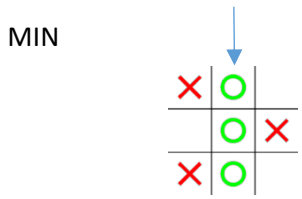
Stav	Hodnota
	1
	0
	-1

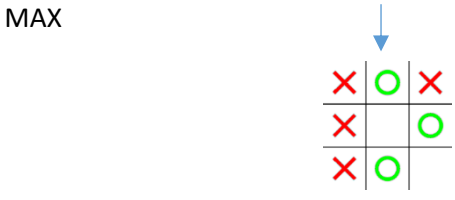
Stejně jako u optimalizačních problémů z přechozích kapitol, můžeme formální problém sekvenční kompetitivní hry pro dva hráče převést na graf. Vrcholy tvoří množiny stavů S a hrany tvoří akce hráčů. Graf bude mít podobu stromu, kdy jednotlivé úrovně tvoří možné akce daného hráče, když je na řadě. Listy tvoří konečné stavy, které jsme schopni ohodnotit funkcí utility. Některé větve budou končit dříve.



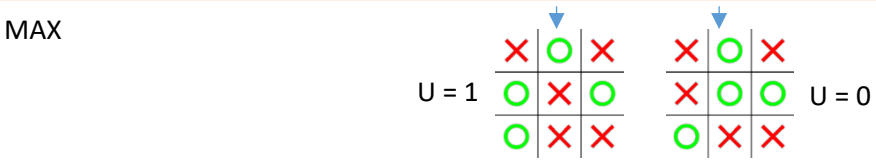
...







...



1.6 Optimální strategie sekvenční kompetitivní hry pro dva hráče

Stavový prostor sekvenční kompetitivní hry pro dva hráče dokážeme převést na strom.

Nyní se podíváme na algoritmy, které nám doporučí tah, který máme zahrát, abychom hru vyhráli a pokud to není možné, tak alespoň remízovali.

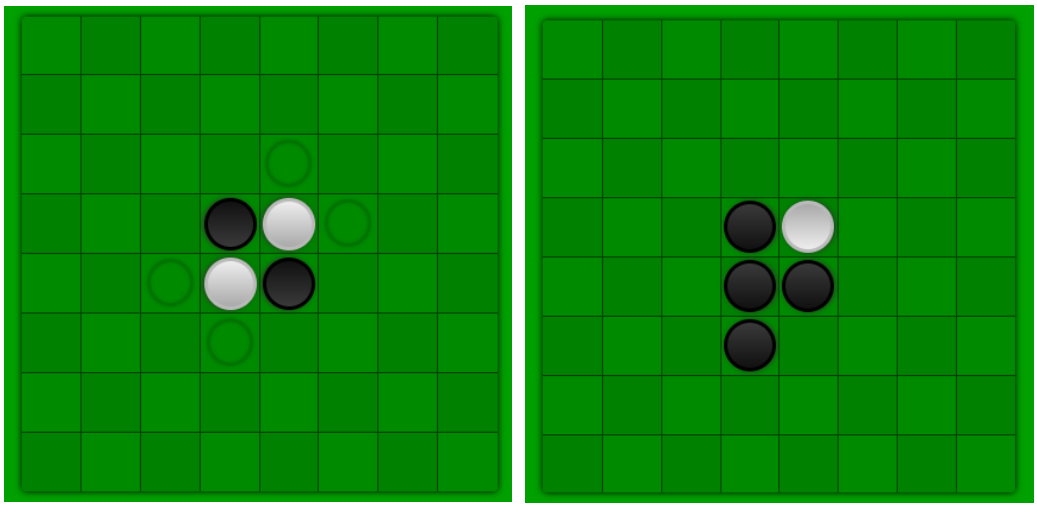
Tyto algoritmy předpokládají, že oba hráči MAX i MIN hrají optimálně.

1.6.1 MINMAX algoritmus

Tento algoritmus vyžaduje průchod kompletního stromu, který reprezentuje stavový prostor hry. Každý koncový stav, list stromu, je třeba ohodnotit užitkovou funkcí utility.

Algoritmus si ukážeme na deskové hře, kdy každý stav dokážeme ohodnotit různými hodnotami a cílem hráče je maximalizovat počet. Například se může jednat o hru Otello(reversi).

Hra se hraje na hrací desce o rozměrech 8 x 8. Hráči se střídají v pokládání bílého respektive černého kamenu. Kameny lze položit pouze na pole, které sousedí s již položenými kameny. Pokud se hráči podaří mezi své kameny uzavřít souvislou řadu soupeřových kamenů, mění se na jeho barvu. Vítězí hráč, který po zaplnění desky na ní má více svých kamenů.

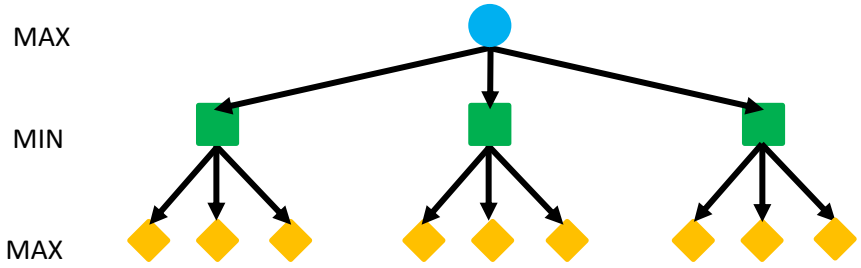


Obrázek 1 - počáteční stav s vyznačenými možnými akcemi a zahrání černý D3

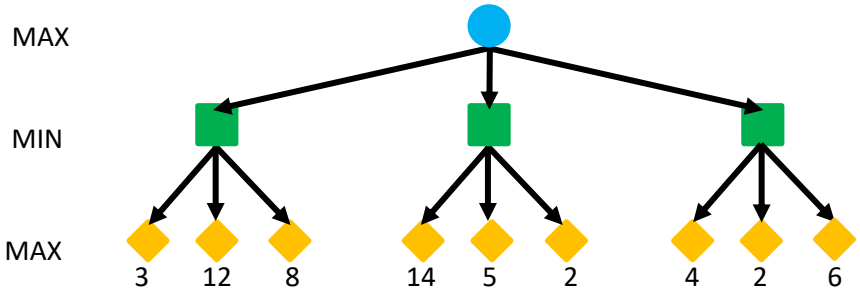
Pokud černý hráč položí kámen na políčko D3 (značeno jako šachovnice od levého spodního rohu), uzavře bílý kámen na D4 a změní ho na černý kámen.

Pro tuto hru může být jednoduchou užitkovou funkcí prostý počet hráčových kamenů.

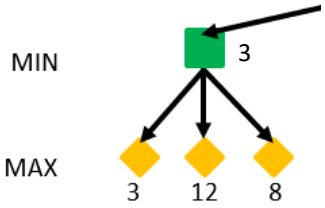
Algoritmus minmax si budeme demonstrovat na následujícím stromu. Hráč MAX má na začátku 3 možné tahy. Pak následuje hráč MIN, který může zahrát pro každý tah hráče MAX své tři tahy. Pak se hra dostane do žlutých koncových stavů. Strom partie bude vypadat následovně.



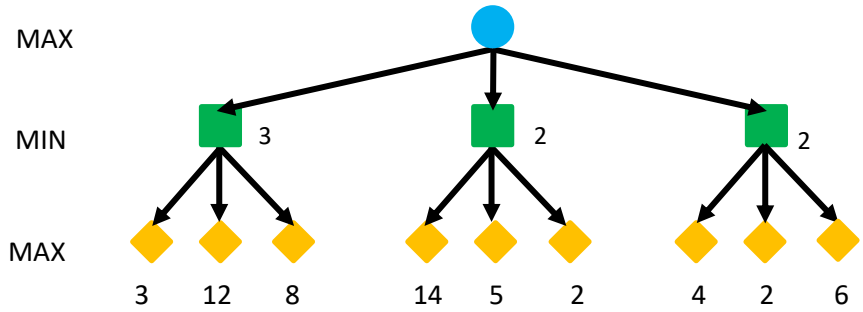
Hráč MAX nyní musí ohodnotit každý koncový stav ze svého pohledu.



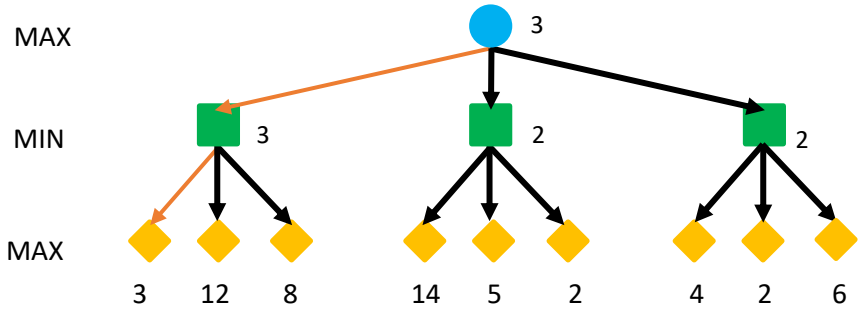
Protože oba hráči hrají racionálně, dokáže si hráč MIN spočítat užitkovou funkci hráče MAX. Pokud by hráč MAX zahrál první tah, tak hráč MIN má k výběru 3 možné tahy, které vedou do koncových stavů ohodnocených 3, 12 a 8. Protože chce, abych hráč MAX měl co nejmenší zisk, zvolí si tah, který vede do koncového stavu 3. Proto tento stav můžeme ohodnotit 3.



Obdobně můžeme ohodnotit zbylé dva tahy hodnotami 2 a 2.



Nyní můžeme zvolit, který zelený tah zvolí hráč MAX. Protože ten chce maximalizovat svůj zisk, zvolí první tah s hodnotou 3. Sice poslední zelný tah může potenciálně vést k hodnotnějšímu koncovému stavu 14, ale protože hráč MIN hraje racionálně, zvolil by si úplně pravý tah vedoucí do koncového stavu ohodnoceného 2.



Pokud hráč MIN nebude hrát optimálně, hráč MAX dosáhne stejného nebo lepšího výsledku.

Pro ohodnocení stavu pak může použít funkci minmax.

- Pro všechny možné platné akce postupně expanduj aktuální stav.
- Pokud je expandovaný stav konečný, ohodnoť ho užitkovou funkcí.
- Pokud expandovaný stav není konečný, zavolej na něj rekurzivně funkci minmax s opačnou strategií.
- Až budou všechny možné akce ohodnoceny, pro hráče MAX vyber akci s maximální hodnotou, pro hráče MIN vyber akci s minimální hodnotou.

Funkce může vypadat například takto.

```
def minmax(self, strategy="max"):
    # kontrola konce partie pro aktualni stav
    result = self.terminal_test()
    if result != 0:
        return self.utility(result), action

    # inicializace hodnot pro jednotlive strategie
    if strategy == "max":
        selected_utilization_value = float('-inf')
        next_strategy = "min"
    else:
        selected_utilization_value = float('inf')
        next_strategy = "max"

    actions = self.possible_actions()
    selected_action = actions[0]

    for action in actions:
        expanded_state = self.expand(action)

        # kontrola konce partie pro expandovany stav
        result = expanded_state.terminal_test()

        if result != 0:
            # partie konci, vraceni ohodnoceni stavu a akce
            return expanded_state.utility(result), action
        else:
            # partien pokracuje
            # pokud v expandovanem stavu existuje alespon nejaky stav,
            # tak stav dale expanduj, jinak je to remiza
            if len(expanded_state.possible_actions()) == 0:
                utilization = 0
            else:
                if expanded_state.depth == self.max_depth:
```

```
                utilization = 0
            else:
                utilization, _ = expanded_state.minmax(next_strategy)

        # podle strategie se dana akce vybira jako vybrana akce
        if utilization > selected_utilization_value and strategy == "max":
            selected_utilization_value = utilization
            selected_action = action
        elif utilization < selected_utilization_value and strategy == "min":
            selected_utilization_value = utilization
            selected_action = action

    return selected_utilization_value, selected_action
```

1.6.1.1 Vlastnosti algoritmu

Algoritmus rekurzivně **prohledává celý strom**. Pokud je stavový prostor hry složitý, algoritmus může vyčerpat paměťové prostředky. Strom je prohledávaný do hloubky, protože pro ohodnocení větve je třeba znát všechny její konečné stavy (listy).

Hlavní nevýhodou je **velká časová náročnost** algoritmu, který je v základní podobě použitelný pouze pro velmi jednoduché hry. Čím více hra s pokročilejším stavem partie nabízí více možných tahů, tím narůstá jeho neefektivnost.

Pokud je **m** délka hry a **b** je větvící faktor, pak lze náročnost vyjádřit následovně:

Parametr	Hodnota	Poznámka
Časová složitost	$O(b^m)$	
Paměťová složitost	$O(bm)$	pokud se mají vygenerovat všechny akce najednou
Paměťová složitost	$O(m)$	pokud v každém tahu je na výběr pouze jeden tah
Optimálnost	Ano	algoritmus zaručuje nalezení optimálního řešení

Pozor, algoritmus nezaručuje vítězství hráče MAX ve všech stavech. Někdy stav partie vůbec možnost vítězství nenabízí. Pak algoritmus doporučuje tahy vedoucí k remíze. Pokud není možná ani remíza, algoritmus vybere tahy vedoucí k prohře.

1.6.1.2 Rozhodování v reálném čase

Hlavní nevýhodou minmax algoritmu je jeho časová náročnost. V některých situacích je třeba vybrat akci ve stanoveném čase a není možné procházet celý strom.

- Jedná se o real-time systém
- Protihráč nesmí čekat dlouho (šachy je třeba dohrát dříve, než skončí vesmír)
- Je třeba zamezit plýtvání výpočetními zdroji

V těchto případech je třeba upravit minmax algoritmus. Jedním ze způsobů je **omezení prohledávané hloubky** stromu. V tomto případě jsou stavy v maximální povolené hloubce považovány za koncové.

Omezení do hloubky může být nastavené **fixně**, kdy se experimentálně vyzkouší, jaká je maximální hloubka, do které lze strom prohledávat, tak aby se akce vrátila v časovém limitu.

Případně lze použít dynamické nastavení maximální hloubky. V tomto případě se vlastně jedná o modifikaci iterativního prohledávání do hloubky. Algoritmus začne vyhodnocovat na minimální hloubce. Pokud ještě zbývá čas, je spuštěno prohledávání s větší hloubkou. Po uběhnutí časového limitu se použije řešení s největší hloubkou.

Pokud používáme omezené prohledávání je vhodné pozměnit užitkovou funkci, aby se chovala jako **heuristická funkce** a dokázala ohodnotit i stavy, které nejsou konečné. Funkce pak místo přesné hodnoty zisku vrací její odhad. Kvalita heuristické funkce značně ovlivní efektivitu algoritmu.

Heuristická funkce musí umět řadit stav stejně jako užitková funkce, vítězné stavy musí být hodnoceny lépe než stavy s remízou. A ty musí být hodnoceny lépe než stavy končící prohrou. Pro necílové stavy musí být hodnota zisku silně korelována s šancí vyhrát.

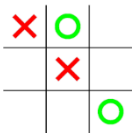
Výpočet heuristické funkce musí být rychlý.

Heuristické funkce často uvažují řadu různých charakteristik. Přínos každé charakteristiky je většinou počítán samostatně a výsledné hodnocení funkce je jejich lineární kombinace, kdy každá charakteristika může mít rozdílnou váhu w.

$$eval(s) = \sum_{i=1}^{n} w_i \, f_i(s)$$

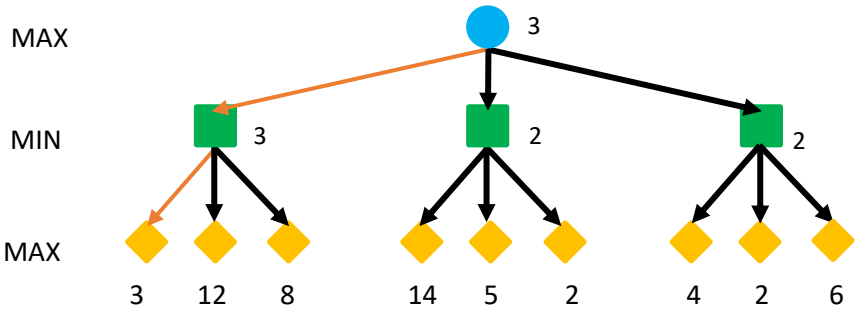
Heuristickou funkci pro piškvorky můžeme vytvořit z hodnocení různých charakteristik. V tabulce jsou uvedená pravidla z pohledu hráče MAX hrajícího s křížky

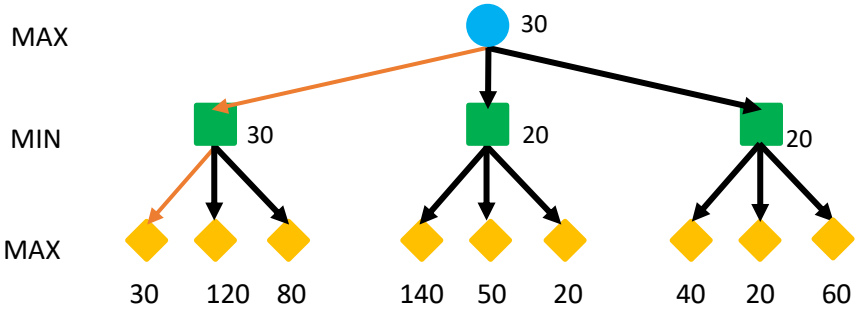
Charakteristika	Váha hodnocení
3X za sebou	+100
2X za sebou následované volným polem	+10
X následované 2 volnými poli	+1
O následované 2 volnými poli	-1
2O za sebou následované volným polem	-10
3O za sebou	-100

Výše popsaná heuristická funkce bude mít pro stav  ohodnocení -1, viz následující tabulka.

Charakteristika	Váha hodnocení	Hodnota
3X za sebou	+100	100 · 0
2X za sebou následované volným polem	+10	10 · 0
X následované 2 volnými poli	+1	1 · 1
O následované 2 volnými poli	-1	-1 · 2 (2 protože pro kolečko v rohu platí pravidlo 2x)
2O za sebou následované volným polem	-10	-10 · 0
3O za sebou	-100	-100 · 0

Konkrétní rozsah vah hodnocení není podstatný. Algoritmus bude pracovat správně pro obě hodnocení.

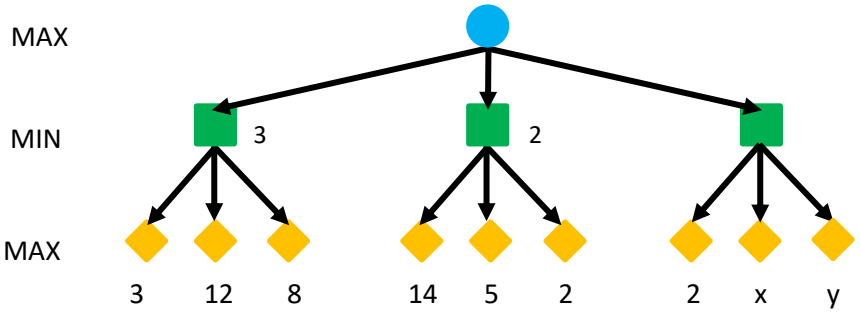




1.6.2 Algoritmus minmax s alfa-beta ořezáváním

Algoritmus minmax lze zrychlit i vynecháním prověřování stavů, které již nemohou vést k lepšímu výsledku. Ukážeme si, že je možné nalézt optimální rozhodnutí (akci) aniž bychom provedli ohodnocení všech stavů v herním stromě.

Předpokládejme, že postupně procházíme strom a zatím máme ohodnocené pouze některé koncové stavy.



Funkce minmax pro modrý uzel lze vypočítat následovně:

$$MINMAX(s) = \max(\min(3, 12, 8), \min(14, 5, 2), \min(2, x, y))$$

Po vypočítání funkcí se známými hodnotami dostaneme:

$$MINMAX(s) = \max(3, 2, \min(2, x, y))$$

min(2, x, y) bude vždy 2 nebo menší

$$\min(2, x, y) \leq 2$$

Proto můžeme dosadit hodnotu 2 i bez znalosti hodnot x, y

$$MINMAX(s) = \max(3, 2, 2\ a\ menší)$$

$$MINMAX(s) = 3$$

Algoritmus minmax s alfa-beta ořezáváním eliminuje podstromy, kam se při hraní nedostaneme, protože neobsahují optimální strategii. Algoritmus je rozšířen o dvě meze:

- α – nejlepší dosud nalezená hodnota v prozkoumávané větvi z pohledu hráče MAX (maximální ohodnocení)
- β – nejlepší dosud nalezená hodnota v prozkoumávané větvi z pohledu hráče MIN (minimální ohodnocení)

V průběhu prohledávání jsou hodnoty α, β průběžně aktualizovány.

Pro ohodnocení stavu pak můžeme použít funkci minmax, do které vstupuje stav a parametry α, β. Hodnota α je na počátku nastavena na -∞ a hodnota β je nastavena na +∞.

- Pro všechny možné platné akce postupně expanduj aktuální stav.
- Pokud je expandovaný stav konečný, ohodnoť ho užitkovou funkcí.
- Pokud expandovaný stav není konečný, zavolej na něj rekurzivně funkci minmax s opačnou strategií.
- Pro hráče MAX vybírej akci s maximální hodnotou, pro hráče MIN vybírej akci s minimální hodnotou. Pokud nastane situace, kdy procházení větve nepovede k optimálnímu řešení, vrať aktuální koncový stav.

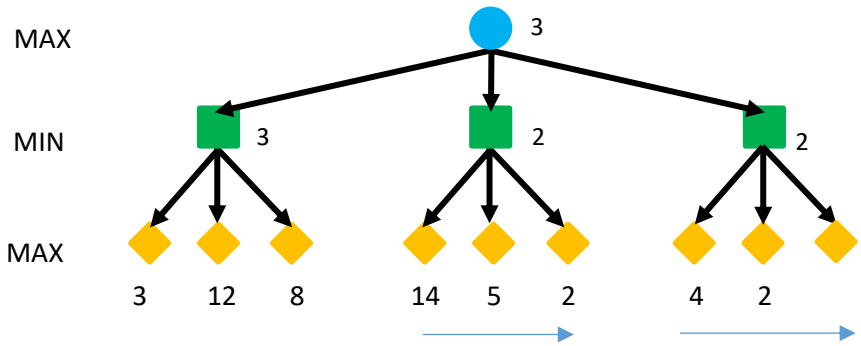
Podstatné změny proti původní verzi minmax jsou zvýrazněny tučně.

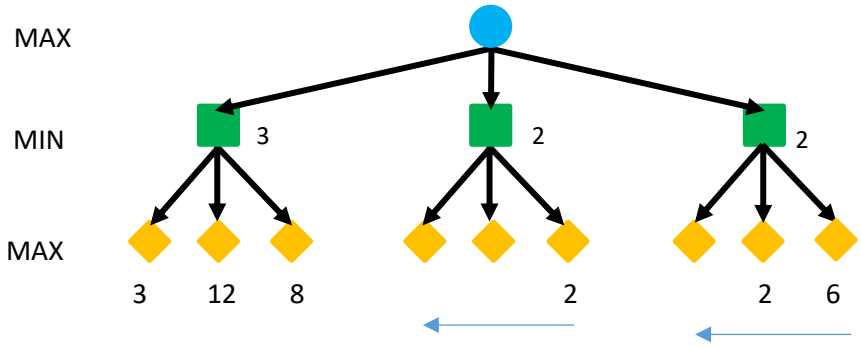
```
def minmax(self, strategy="max", alfa=float('-inf'), beta=float('inf')):
    # kontrola konce partie pro aktualni stav
    result = self.terminal_test()
    if result != 0:
        return self.utility(result), action

        # inicializace hodnot pro jednotlive strategie
    if strategy == "max":
        selected_utilization_value = float('-inf')
        next_strategy = "min"
    else:
        selected_utilization_value = float('inf')
        next_strategy = "max"

    actions = self.possible_actions()
    selected_action = actions[0]

    for action in actions:
        expanded_state = self.expand(action)

        # kontrola konce partie pro expandovany stav
        result = expanded_state.terminal_test()

        if result != 0:
            # partie konci, vraceni ohodnoceni stavu a akce
            return expanded_state.utility(result), action
        else:
            # partien pokracuje
            # pokud v expandovanem stavu existuje alespon nejaky stav,
            # tak stav dale expanduj, jinak je to remiza
            if len(expanded_state.possible_actions()) == 0:
                utilization = 0
            else:
                utilization, _ = expanded_state.minmax(next_strategy, alfa, beta)

            # podle strategie se dana akce vybira jako vybrana akce
            if strategy == "max":

                if utilization > selected_utilization_value:
                    selected_utilization_value = utilization
                    selected_action = action

                # pokud je vracena hodnota minmax vetsi nez beta, tak ji vrat
                if selected_utilization_value >= beta:
                    return selected_utilization_value, selected_action
```

```
                # pokud je vracena hodnota vetsi nez alfa, alfu aktualizuj
                if selected_utilization_value > alfa:
                    alfa = selected_utilization_value
            else:
                # strategy min
                if utilization < selected_utilization_value:
                    selected_utilization_value = utilization
                    selected_action = action

                # pokud je vracena hodnota minmax mensi nez alfa, tak ji vrat
                if selected_utilization_value <= alfa:
                    return selected_utilization_value, selected_action

                # pokud je vracena hodnota mensi nez beta, betu aktualizuj
                if selected_utilization_value < beta:
                    beta = selected_utilization_value

    return selected_utilization_value, selected_action
```

1.6.2.1 Vlastnosti algoritmu

- Oříznutím neztratíme optimální řešení.
- Efektivita algoritmu závisí na pořadí prohledávaných větví.



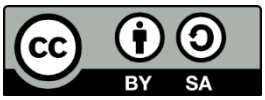


Pokud je **m** délka hry a **b** je větvící faktor, pak lze náročnost vyjádřit následovně:

Parametr	**Hodnota**	**Poznámka**
Paměťová složitost	$O(bm)$	Pokud se mají vygenerovat všechny akce najednou
Paměťová složitost	$O(m)$	Pokud v každém tahu je na výběr pouze jeden tah
Časová náročnost	$O(b^{m/2})$	Nejnižší možná, pokud se neomylně vybírají stavy slibující nejlepší řešení
Časová náročnost	$O(b^{3m/4})$	Při náhodném rozložení optimálních větví
Úplnost	Ano	
Optimálnost	Ano	Algoritmus zaručuje nalezení optimálního řešení

UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vytvořeno v rámci projektu **Digitalizace studijních Agend, Nové TechnologiE, systémy a přístupy k výuce na UPCE**, reg. č. NPO_UPCE_MSMT-16591/2022.



Financováno Evropskou unií
NextGenerationEU

Národní plán obnovy

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Machine learning & AI

Úvod do umělých neuronových sítí

Ing. Martin Pozdílek, Ph.D.

1 Biologický nervový systém

Umělé neuronové sítě jsou inspirovány biologickými neuronovými sítěmi. Biologický mozek, a nejen lidský, je velmi fascinující a dost možná nejkomplexnější systém ve známém vesmíru. V minulém a současném století probíhá jeho intenzivní výzkum. Moderní neurověda umí popsat do velkých detailů jednotlivé biologické a fyzikální mechanizmy, které se v mozku i nervové soustavě odehrávají. Přes veškerý výzkum nemůžeme prohlásit, že zcela chápeme fungování nervové soustavy na mikro i makro úrovni. Vysvětlení fenoménů jako vědomí, sebeuvědomění atp. ještě není zcela hotové.

Na základě neurofyziologický poznatků vznikají různé teorie fungování biologické nervové soustavy. Tyto teorie jsou pak východiskem pro návrh umělých neuronových sítí, a to jak po stránce jejich detailního fungování, přes topologii až po fungování umělých neuronových sítí jako celku. V posledních letech se objevili velké jazykové modely, které svými schopnostmi mnohé udivují. Neustále probíhá výzkum jejich schopností a hledání vysvětlení, jak jsou tyto schopnosti vznikly, když během jejich návrhu nebyly nijak projektovány.

1.1 Informační systém organismu

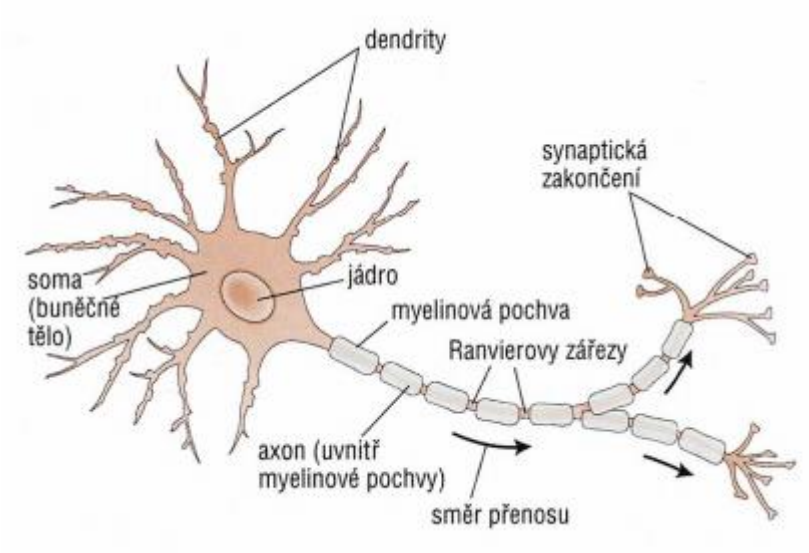
Organismy pro své přežití potřebují cílevědomě a systematicky sbírat, ukládat a vyhodnocovat informace o svém okolí, ale i o sobě samém. Pro vnímání informací, podmětů, organismy používají různé receptory. Receptor je nějaké čidlo, které umí zachytit mechanické, chemické, teplotní, elektrické, magnetické, gravitační, světelné a další podměty. Ve své podstatě se jedná o zachycení nějaké formy energie a její přeměnu na vzruch. Vnější receptory zpravidla představují nějaký smysl – zrak, sluch, čich, hmat atd. Vnitřní receptory sledují stav vnitřního prostředí organizmu. Můžeme se pod nimi představit vnímání chemického prostředí, kyselost, koncentrace různých chemických látek, vnímání umístění a orientace organismu v prostoru, vnímání jednotlivých částí, sledování bolesti a mnoho podobných podmětů.

Tyto podměty jsou formou vzruchů šířeny přes nervy do centrální nervové soustavy k vyhodnocení. Informace jsou analyzovány a na jejich základě jsou posílány signály k jednotlivých výkonným orgánům – efektorům. Typickým příklady efektoru mohou být svaly nebo žlázy.

Toto chování můžeme považovat za informační systém organizmu. Jeho složitost je určena složitostí a prostředím, ve kterém organizmus žije. Je zřejmé, že informační systémy bakterie, hmyzu nebo obratlovce se budou lišit, a to jak v různé výbavě receptorů, struktuře nervové soustavy, ale i jejího fungování. U vyšších organizmů je přežití centrální nervové soustavy nezbytné pro přežití organizmu.

1.2 Biologický neuron

Neuron je základní jednotkou biologické nervové soustavy. Jedná se o nervovou buňku, která se evolucí specializovala na přenos a uložení informací.



Obrázek 1 - skladba neuronu

Neuron je oddělen od vnějšího prostředí buněčnou membránou, která jej chrání před poškozením a zajišťuje látkovou výměnu. Stejně jako i jiné buňky musí být neuron vyživován, okysličován a musí z něj být odváděny škodlivé produkty metabolismu.

Největší část neuronu tvoří buněčné tělo (soma), ve kterém se nachází buněčné jádro. Nejnápadnější odlišností neuronu od ostatních buněk je přítomnost většího počtu bohatě větvených výběžků, přes které jsou neurony vzájemně propojeny.

Prvním typem výběžků jsou dendrity, která zajišťují šíření vzruchu do buňky, jedná se vlastně o vstupy. Dendrity jsou krátké a bohatě větvené a umožňují napojení na více různých okolních neuronů.

Druhým typem výběžku je axon. Neuron má pouze jeden axon, který vede vzruch směrem od buňky, jedná se o tedy o výstup. Axon je v porovnání s dendritem silnější a delší. Jeho délka může být o mikrometrů až po 60 cm. Aby byl přenos signálu na větší vzdálenost spolehlivější a nezkreslený, je axon obalen tukovým obalem nazývaným myelin. Při šíření signálu dochází k jeho útlumu. Aby signál přenášený na větší vzdálenost nebyl utlumen, je axon v pravidelných intervalech přerušován úzkými Ranvierovými zářezy. Při průchodu signálu Ranvierovým zářezem dochází k obnově jeho intenzity. Konec axonu je rozvětven, což mu umožňuje napojení na další neurony. Tomuto spojení se říká synapse.

Lidský mozek obsahuje 86 miliard neuronů a 100 bilionů synapsí. Průměrný neuron několik tisíc synapsí. Ale například Purkyňovy buňky mozečku mají okolo 200 000 synapsí.

Základní zjednodušený princip neuronu je následující. Neuron přes dendrity přijímá signály z okolních neuronů. Některé signály jsou excitační, kdy jejich hodnota se přičítá a aktivuje neuron. Některé signály mohou být naopak inhibiční, tedy jejich přítomnost utlumuje neuron.

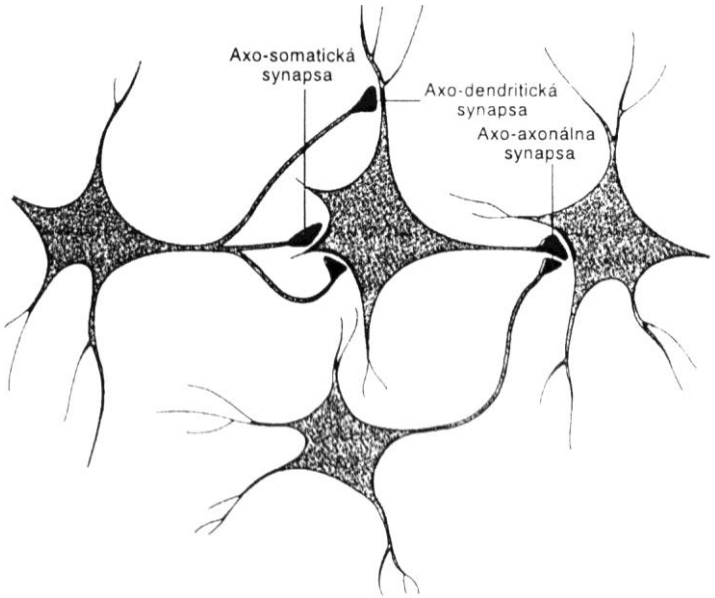
Pokud souhrn všech přijímaných signálů překročí určitou mez, dojde k aktivaci neuronu, který vyšle do axonu signál. Tok informací tedy probíhá od dendritů směrem k axonu. Úroveň přijímaných signálů rozhodne, jestli se signál pošle dále.

1.3 Synapse

Pro vytváření funkční nervové soustavy je klíčová schopnost propojování neuronů přes synapse. Synapse umožňuje kontakt neuronů a můžeme si ho představit jako uzly v síti, komunikační rozhraní (interface), které zajišťuje šíření různých signálů v neuronové síti.

Synapse je tvořena presynaptickou částí tvořenou axonem a postsynaptickou částí tvořenou dendritem nebo tělem neuronu. Mezi nimi je synaptická štěrbina. Synapse může rozdělit podle toho, kde se přesně nachází.

- axo-dendritická – spojení axonu a dendritu, nejčastější
- axo-axonální – spojení axonů dvou neuronů
- axo-somatické – spojení axonu a těla neuronu



Obrázek 2 - typy synapsí

Synapse může rozdělit o podle mechanismu přenosu vzruchu.

Elektrické synapse jsou vývojově starší než chemické. Neurony se musí přímo dotýkat a k přenosu dochází průchodem elektricky nabitých iontů přes membrány. Obě části synapse jsou v podstatě stejné.

U chemické synapse se odlišuje presynaptická a postsynaptická část. Presynaptická část axonu uvolňuje chemickou látku transmiter. Ten difunduje přes synaptickou štěrbinu a reaguje s molekulami receptoru v membráně postsynaptické části synapse. Receptor při přítomnosti transmiteru vyvolá změny, které ovlivní tok iontů přes membránu.

Základním znakem inteligence je schopnost učení. Tedy proces, kdy organizmus při výskytu opakovaných podmětů upraví své chování a adaptuje se na nové situace. Pro učení je nezbytná schopnost uložení informací do paměti.

Způsob uložení informací zatím není přesně objasněn. Jedna teorie uvažuje o výhradně látkové podstatě paměti, kdy informace jsou zakódovány do bílkovin a RNA.

Další teorie předpokládá, že paměť vzniká vytvářením nových nervových spojení, tedy změnou struktury neuronové sítě. Během života organizmu se nové neurony příliš netvoří. Pokud dojde k jejich odumření, organizmus nemá až na výjimky možnosti jejich náhrady. Nicméně struktura mozku není neměnná a v průběhu života dochází k vytváření a zániku různých synapsí. Tím je schopen mozek určité míry opravy, kdy poškozené části přemostí a procesem učení může ztracenou funkčnost přenést do jiných částí. Ovšem tyto opravy mají své omezení.

Možná nejpravděpodobnější teorie, kterou také používají umělé neuronové sítě, předpokládá funkční změny v existujících nervových drahách. Proces učení krátkodobě nebo dlouhodobě mění potenciaci nervových drah. Jinými slovy stejný vzruch se vlivem učení může začít šířit v nervové soustavě jinými drahami. Za tuto plasticitu mohou synapse. Průchod vzruchu synapsí vyvolává paměťové stopy, které trvají několik vteřin. Při opakovaném průchodu signálu dochází ke změnám na synapsích, které umožní uložit informaci po delší dobu. Proces učení je mnohem komplikovanější a zahrnuje různé typy pamětí jako například slovní, zrakovou, sluchovou, čichovou, hmatovou, pohybovou, emociální paměť. Paměti můžeme rozlišovat krátkodobé nebo dlouhodobé. Zajímavý je i proces zapomínání. Obor neurovědy je mnohem širší a hlubší. Tento krátký úvod nám umožní pochopit umělé neuronové sítě.

2 Umělé neuronové sítě

Umělé neuronové sítě se velmi inspirovaly biologickými neuronovými sítěmi. Hlavní rozdíl je v tom, že umělé neuronové sítě jsou zjednodušujícím matematickým modelem, který idealizuje mnohé chemicko-fyzikální procesy. Některé zásadní rozdíly jsou popsány v kapitole Porovnání biologické a umělé neuronové sítě.

Umělé neuronové sítě se inspirovaly základním chováním neuronu, který přes dendrity přijímá informace. Pokud jsou vstupní signály dostatečně silné, neuron se aktivuje a posílá dále výstupní signál.

Biologické neuronové sítě jsou schopné řešit komplexní úlohy ne díky komplexním neuronům, ale díky jejich spolupráci v komplikované síti. Umělé neuronové sítě opět tvoří různé struktury a topologie, které umožňují řešit komplexní úlohy.

Umělé neuronové sítě je vhodné použít v případech, kdy neexistuje jiný exaktní matematický model, který je schopen popsat všechny prvky a vztahy řešeného problému. Nebo je tento model příliš komplikovaný, nepřehledný a nelze ho převést do provozuschopného algoritmu. Pokud bychom se tento komplikovaný model pokoušeli zjednodušit a idealizovat, tak výsledný model bude nepřesný. Obecně jsou neuronové sítě vhodné pro úlohy, které používají nelineární systémy.

Umělé neuronové sítě se chovají podobně jako biologické neuronové sítě. Neuronová síť má na počátku určitou strukturu a vazby, která neumí efektivně řešit příchozí podměty. Procesem učení se neuronová síť přizpůsobuje a postupnými malými změnami v synapsích se učí směrovat signály po správných nervových drahách. Neuronová síť si naučí, zapamatuje si, jak vyřešit konkrétní vstupní vzruchy. Velmi podobně fungují umělé neuronové sítě.

Umělé neuronové sítě se zpravidla používají pro řešení následujících úloh.

- Predikce hodnot. Na základě vstupních hodnot, dokáže neuronová síť předvídat výstupní hodnotu. Může se například jednat o predikci časových řad v ekonomii, medicíně, strojírenství, energetice atp.
- Rozpoznávání vzorů, klasifikace. Typické jsou úlohy z počítačového vidění – rozpoznání obrazu, čtení textu, daktyloskopie, rozpoznání choroby, …
- Transformace vstupních dat. Jedná se úlohy týkající se přirozeného jazyka jako jsou překlady, převod řeči na text a opačně. V poslední době jsou velmi populární generativní neuronové sítě pro vytváření obrazu podle slovního popisu. Do této kategorie můžeme zařadit i velké jazykové modely, které generují text na základě vstupního textu.

2.1 MP neuron

Abychom mohli stavět modely umělých neuronových sítí, budeme potřebovat základní stavební prvek – umělý neuron. Jednoduchý McCullochův-Pittsův model neuronu byl publikován v roce 1943 a jeho inspirace biologickým neuronem je zřejmá.

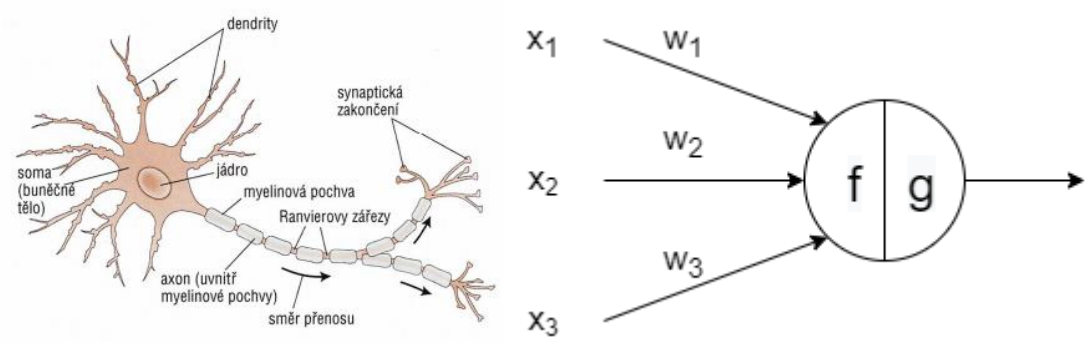
Stejně jako do biologického neuronu vstupuje celá řada dendritů, tak do umělého neuronu vstupuje celá řada hodnot. V MP neuronu se předpokládá, že vstupní hodnoty mohou být 0 nebo 1. Podobně jako u neuronů, kde jsou signály různými mechanismy zesilovány nebo tlumeny, u umělého

neuronu má každý vstup svoji váhu w, která může nabývat hodnot <0, 1>. Hodnota 0 znamená, že daný vstup nemá žádný vliv na fungování neuronu a vlastně značí, že dané spojení vlastně neexistuje.

Vstupní hodnoty upravené vahami jsou agregovány pomocí funkce f. V tomto místě budeme předpokládat, že funkce je vážená suma vstupů, ale agregačních funkcí existuje samozřejmě více.

$$f = \sum_{i=1}^{n} w_i \, x_i$$

Biologický neuron se aktivuje, když přichází dostatečně silné signály. U umělého neuronu existuje aktivační funkce g, která řídí aktivaci neuronu na základě vstupů. Tu si můžeme představit jako nějaký práh. Pokud hodnota $f(x, w) > práh$, pak na jediném výstupu (axonu) bude hodnota 1, jinak výstupní hodnota bude 0.



Obrázek 3 - biologický a umělý neuron

Takovýto MP neuron můžeme například použít pro jednoduchý rozhodovací proces. Například student má během semestru složit 3 testy. Výstupem z každého testu je informace, zda ho složil (1) nebo neuspěl (0). Úkolem neuronu bude rozhodnout, zda má student dostat zápočet nebo ne.

Vstupními hodnoty jsou tedy binární proměnné x1, x2, x3. U každého testu můžeme rozhodnout, jak byl důležitý. Například první dva testy byly jednoduché a poslední byl složitý. Podle toho nastavíme váhy w1, w2, w3. Funkce f bude opět vážená suma. Pro dokončení modelu musíme zvolit prahovou hodnotu, která ovlivňuje funkci g. Na základě prahu umělý neuron rozhodne, zda jsou vstupní hodnoty dostatečné pro udělení zápočtu.

Umělou neuronovou síť si můžeme představit jako matematickou funkci, která transformuje číselné vstupy na číselné výstupy. Podobně jako ostatní matematické funkce má i neuronová síť parametry, kterými ji můžeme upravovat, aby se chovala podle našich představ.

Jak student napíše testy, ovlivnit nemůžeme, to jsou vstupy, ale můžeme ovlivnit váhy, které určují důležitost jednotlivých vstupních hodnot. Dále můžeme ovlivnit prahovou hodnotu. Prahová hodnota představuje vlastně těžkost aktivace neuronu.

Náš model má tedy 4 vstupní parametry. Model je natolik univerzální, že jej lze použít pro různé předměty, pouze každý vyučující stanoví hodnotami parametrů, jak bude složité získat zápočet. Někomu bude stačit složení libovolného testu. Jiný vyučující stanoví, že student musí splnit první dva testy nebo test poslední. Možná je varianta vah, kdy student musí splnit všechny tři testy.

Matematický model neuronu můžeme upravit tak, že k funkci přičteme bias. Bias je jeden parametr neuronu. Jedná se o výchozí hodnotu neuronu, kterou si můžeme představit jako zápornou hodnotu prahu. Určuje, jak složité je neuron aktivovat.

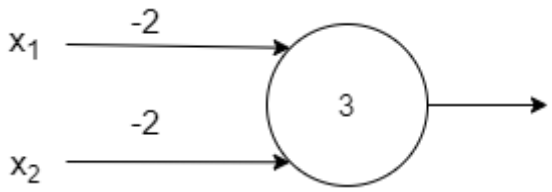
$$f = \sum_{i=1}^{n} w_i x_i + b$$

Později se nám bude hodit převedení matematického modelu do vektorové podoby. Kdy *w* a *x* budou mít podobu vektoru o velikosti počtu vstupních parametrů. Vážený součet vstupních parametrů budeme realizovat jako skalární součin vektoru vstupních hodnot a transponovaný vektor vah. K funkci přičteme skalární hodnotu bias.

$$f = w^T \cdot x + b$$

Pomocí modelu neuronu dokážeme vhodnou volbou parametrů a biasu vytvořit i logické funkce. Například univerzální logická funkce NAND vrací hodnotu 0, pokud jsou oba vstupy 1. V ostatních případech vrací hodnotu 1.

Model neuronu, který implementuje logickou hodnotu NAND vypadá například takto. Obě váhy budou -2 a hodnota biasu bude 3. Pokud vstupní hodnoty x_1 a x_2 budou 1, pak hodnota funkce neuronu bude $-2 \ -2 \ + \ 3 \ = \ -1$, která je menší než 0 a neuron se neaktivuje a na výstupu bude hodnota 0. Ve všech opačných případech nebude hodnota funkce záporná a neuron se aktivuje.



Obrázek 4 - NAND

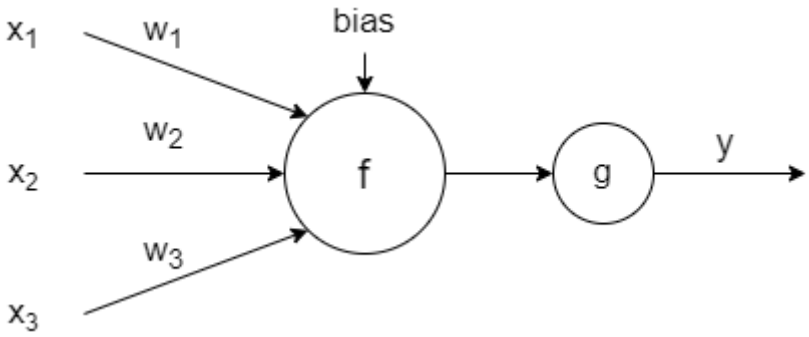
2.2 Perceptron

Výše uvedený MP model neuronu má určité nedostatky, které způsobily, že se neuronové sítě prosadily do umělé inteligence mnohem později po svém objevu. Při fitování modelu MP neuronu na učební data, která obsahují vychýlené (hraniční, extrémní) případy, proces učení vykazuje nestabilitu. Důvodem jsou binární vstupní a výstupní hodnoty. Předpokládejme, že jsme zatím neuron naučili na standardní vstupní hodnoty. Pokud budeme pokračovat v učení na atypických datech, drobnými

změnami parametrů docílíme toho, že sice neuron již správně reaguje na atypická data, ale tyto drobné změny zcela „rozbijí“ chování neuronu pro standardní data. V konečném důsledku nedokážeme najít správné parametry, které budou správně vyhodnocovat standardní i atypická data. Důvodem je výstupní nespojitost funkce g, která při drobné změně vstupů radikálně přepne výstup z jednoho extrému do druhého.

Řešením tohoto problému je upravení výstupní hodnoty z neuronu pomocí aktivační funkce. Aktivační funkce upravuje výstup z neuronu tak, aby se pohyboval v povolených mezích a případná extrémní hodnota nezpůsobila v modelu nevhodné chování. Zároveň aktivační funkce vytváří spojitý obor hodnot výstupní hodnoty neuronu, což řeší problémy při učení vychýlených hodnot. Takovýto model neuronu se nazývá **perceptron**.

Agregační funkci f, vážený součet, si ponecháme z předchozího modelu, pouze povolíme, aby vstupní parametry mohly nabývat spojitých hodnot. Výstup z agregační funkce vstupuje do aktivační funkce g, která vrací výstupní hodnotu y.



Obrázek 5 - Perceptron

Podobně jako agregačních funkcí může být více, tak i aktivačních funkcí existuje celá řada. Na tomto místě bude pracovat s aktivační funkci **sigmoid**, ale později si ukážeme i další aktivační funkce.

Sigmoid funkce, někdy nazývaná logistická funkce, má tvar písmene S. Jedná se o monotónní funkci, která má první derivaci ve tvaru zvonu. Funkce má přesně jeden inflexní bod. Definiční obor je od -∞ do +∞. Naopak obor hodnot je od 0 do 1. Pro každou hodnotu platí, že derivace v bodě je nezáporná. To je jedním z důvodů, proč se tak často používá jako aktivační funkce v neuronových sítích, protože malá změna vstupních hodnot povede ke změně výstupní hodnoty stejným směrem.

Další příjemnou vlastností funkce je, že vysoké hodnoty se limitně blíží 1 a naopak záporné hodnoty se limitně blíží k 0.

$$f(x) = \frac{1}{1 + e^{-x}}$$

Obrázek 6 - aktivační funkce sigmoid

Model neuronu se tedy skládá z:

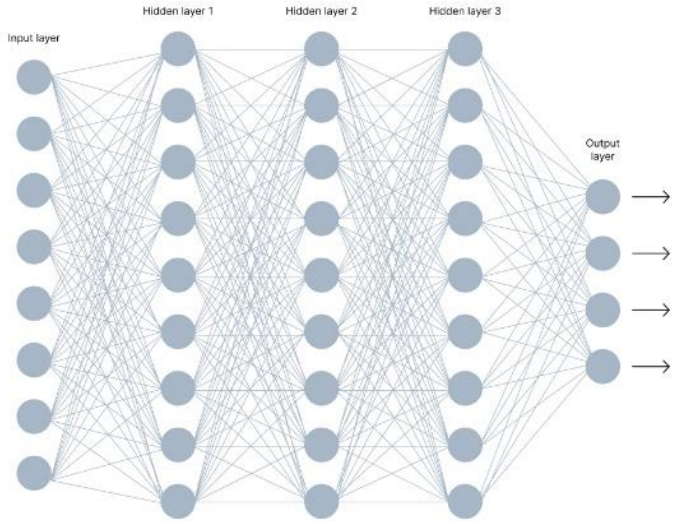
- Individuálních vah pro n vstupů
- Hodnoty bias
- Agregační funkce f, která nějakou matematickou funkcí spojí vstupy do jednoho čísla
- Aktivační funkce g, která upraví výstupní hodnotu neuronu

Počty parametrů a výběr konkrétních funkcí jsou v rukou vývojáře umělé inteligence, který je volí podle požadované funkcionality neuronové sítě. Konkrétní hodnoty vah a bias jsou dány procesem učení.

2.3 Vícevrstevný model neuronové sítě

Podobně jako biologické neurony lze i umělé neurony zapojovat do sítí, protože tak lze modelovat komplexnější nelineární systémy, které se v praxi velmi často vyskytují. Neuronové sítě se právě používají v případech, kdy klasické matematicko-statistické nefungují nebo jsou příliš komplexní na pochopení a výpočet.

Pro jednoduchost se zpravidla neurony v umělých neuronových sítích propojují do vrstev, kde neurony v dané vrstvě mají stejné chování. Mluvíme tak o **vícevrstevných sítích, vícevrstevném perceptronu**.



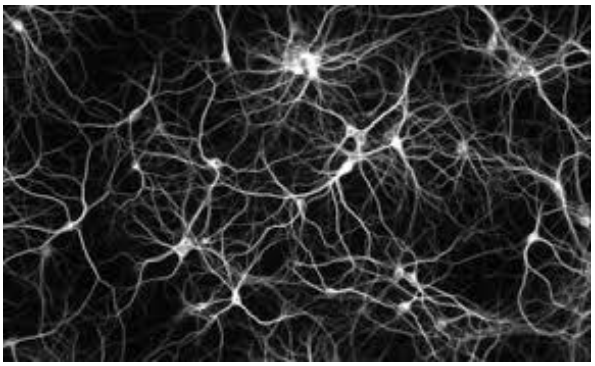
Obrázek 7 - vícevrstevná neuronová síť

Každá umělá neuronová síť má **vstupní vrstvu**, která je napojená na vstupní hodnoty. Může se jednat například o hodnoty čidel, proměnné, pixely v obraze atd. Na druhé straně je **výstupní vrstva**, kde jsou neurony, které vrací výstupní hodnoty. Matematicky si neuronovou síť můžeme představit jako funkci, která má vstupní parametry a funkce nám vrací nějaké hodnoty.

Mezi vstupní a výstupní vrstvou leží **skryté vrstvy**. Jejich počet, funkce je navenek skrytá a jejich počet, podoba a propojení je závislé pouze na vývojáři umělé inteligence. Skryté vrstvy provádí dílčí transformace vstupních hodnot. S každou skrytou vrstvou jsou vstupní hodnoty více transformovány tak, aby se změnily na výstupní hodnoty.

Pokud je vrstev větší počet, nazýváme model **hlubokou neuronovou sítí**. Bylo zjištěno, že hluboké neuronové sítě poskytují lepší výsledek než jednodušší sítě, které mají stejnou strukturu, ale pouze se skládají z menšího počtu neuronů. Mnohdy původní nepřesný model stačilo rozšířit o další vrstvy a hned se stal mnohem přesnější. Přidávání dalších vrstev ovšem zesložiťuje proces učení i proces samotného používání neuronové sítě (inference).

V hlubokých sítích bývá velmi obtížné přesně definovat, jaký přínos má konkrétní vrstva či dokonce neuron na celkový výstup. Jejich ladění a pochopení je velmi komplikované a mnohdy se chovají jako černé skříňky. Tím se blíží biologickým neuronovým sítím, ve kterých nemusí být ani jednotlivé vrstvy rozpoznatelné.



Obrázek 8 - biologická neuronová síť

2.3.1 Topologie

Důležitou charakteristikou pro neuronovou síť je její topologie, tedy způsob, jak jsou různé typy neuronů pospojovány. Při vytváření modelu máme možnost zkoušet různé topologie a sledovat, která se chová nejlépe na daných datech. Abychom pokaždé nemuseli začínat od začátku jsou známy různé sítě, které se hodí pro různé problémy. S některými z nich se seznámíme a některé si vyzkoušíme i na cvičeních.

První **vstupní vrstva** neuronové sítě je určena vstupními daty. Zde je vhodné připomenout, že vstupní data by měla být dopředu pročištěná a transformovaná do vhodné podoby pro neuronovou síť. Tento proces se velmi podobá procesu známému v datových skladech, kde se nazývá ETL (extract, transform a load). Příkladem transformace může být normalizace dat nebo převedení vstupních dat do vhodného formátu. Například obrázky můžeme ukládat ve formátech RGB, CMYK, HSL, HSV, … Správnou transformací můžeme zjednodušit proces učení neuronové sítě.

Výstupní vrstva je opět určena požadovaným výstupem, který může představovat ovládací signály, kategorie, hodnoty, … Například pokud, budeme chtít vytvořit umělou neuronovou síť, která bude přiřazovat vstupní data do jedné z 10 kategorií. Při volbě počtu výstupních neuronů máme několik možností.

Možná první nejvíce intuitivní je volba jednoho neuronu, který bude vracet hodnoty z omezeného oboru hodnot <0, 10>. Jednotlivým kategoriím přiřadíme rozpětí výstupní hodnoty. Například <0, 1> bude kategorie 1, (1, 2> bude kategorie 2 atd.

Jiným způsobem může být zakódování pomocí binárního kódu. Pro uložení 10 kategorií budeme potřebovat 4 neurony, které budou nabývat hodnot 0 a 1. V případě spojitých výstupních hodnot, určíme prahovou hodnotu, kdy se jedná o 0 nebo 1. Výstupní kategorii získáme přečtením hodnot 4 neuronů a převedením z binárního kódu.

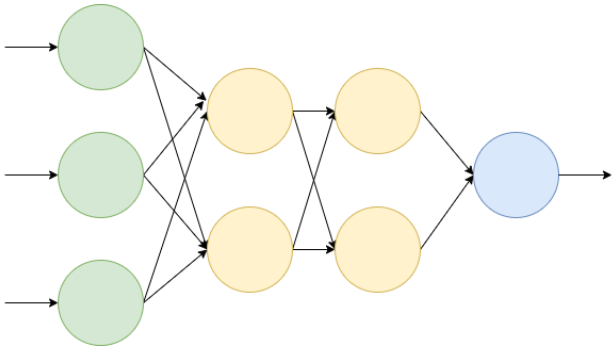
V případě klasifikací se velmi často používá stejný počet výstupních neuronů jako je počet možných kategorií. Při správně volbě aktivačních funkcí, budou výstupní neurony obsahovat hodnotu

z množiny <0, 1>, která určuje pravděpodobnost příslušnosti vstupních dat do dané kategorie. Jde zařídit i to, že součet hodnot všech výstupních neuronů bude 1, tedy 100 %.

2.3.1.1 Dopředná neuronová síť

Nejzákladnější typem neuronových sítí jsou sítě **dopředné** (ForwardFeed ANN). Vstupní hodnoty v nich proudí pouze jedním směrem od vstupu k výstupu. Nejsou zde žádné zpětné vazby, cykly apod. Velmi často se používá propojení vrstev nazývané jako plné spojení (Fully connected), kdy každý neuron z jedné vrstvy je propojen s každým neuronem následující vrstvy. Během učení se může stát, že váha nějakého spojení bude 0, tedy že se propojení nebude realizovat. Ale počáteční „substrát" neuronové vrstvy obsahuje všechna možná spojení mezi každými dvěma vrstvami.

Tento typ neuronové sítě se velmi často kombinuje s jinými typy sítí a zpravidla slouží k přípravě vstupních dat nebo naopak k finálnímu upravení výstupních dat.

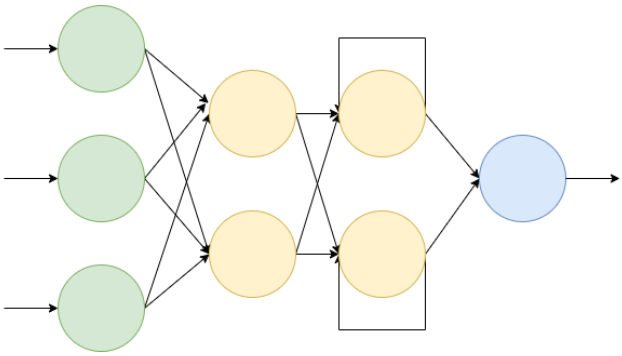


Obrázek 9 - dopředná plně propojená neuronová síť

2.3.1.2 Rekurentní neuronová síť

Dalším typem rekurentní sítě (Recurrent neural network - RNN). Jedná se o sítě, ve kterých se na některých místech objevuje zpětné propojení do předchozích vrstev. Toto propojení dokáže simulovat **krátkodobou paměť** a spojovat informace, které jsou posunuté v čase. Délka paměti závisí na délce zpětné vazby. Pokud zpětná vazba vede do stejného neuronu, propojují se hodnoty v čase t a t+1. Pokud by zpětná vazba vedla do předchozího neuronu, propojí se hodnoty v čase t a t+2. Nevýhodou může být ztrácení gradientu, pokud se hodnoty, které si má síť pamatovat, neopakují. Jedná se tedy o velmi krátkodobou paměť, která je velmi závislá na délce mezery mezi informacemi, které je potřeba propojit.

Rekurentní neuronové sítě se často používají pro zpracování dat, která mají časovou souvislost. Může se například jednat o různé časové řady, analýzu řeči, textu či rukopisu.



Obrázek 10 - rekurentní neuronová síť

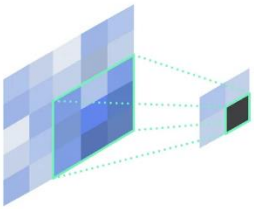
2.3.1.3 Konvoluční neuronová síť

Konvoluční neuronová síť (CNN) je typ dopředné neuronové sítě, která se sama pomocí filtrů (kernelů) učí rozpoznávat klíčové charakteristiky dat, která jsou často obrazová nebo prostorová. Proto se konvoluční sítě často používají v počítačovém vidění, klasifikaci obrázků, detekci objektů v obraze či segmentaci obrázků.

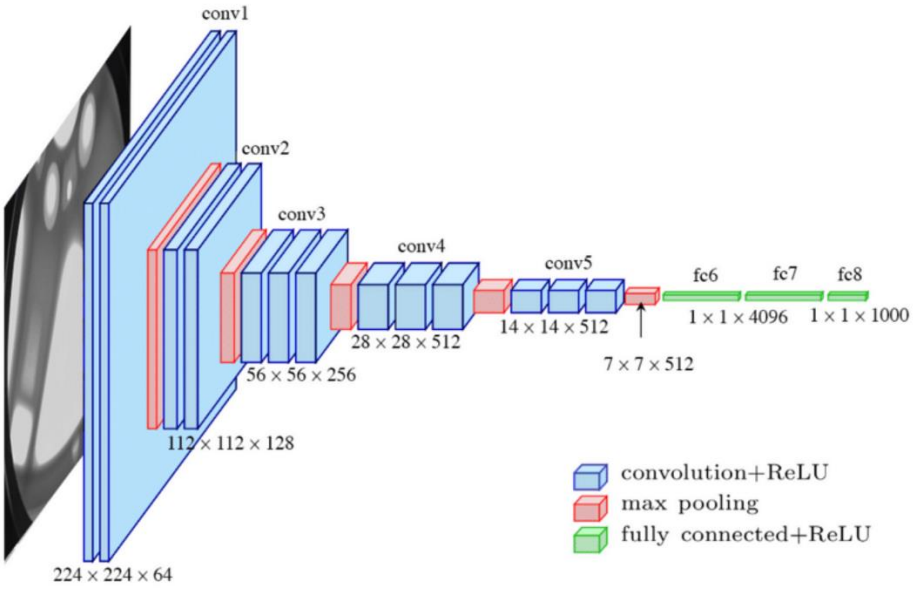
Architektura sítě se zpravidla skládá z konvolučních vrstev, sdružovacích (pooling) vrstev i běžných plně propojených vrstev. Kaskádové uspořádání vrstev umožňuje redukovat počet potřebných neuronů a jejich propojení.

Typickou operací v konvolučních sítích je konvoluce, kdy je z podmnožiny vstupních dat vytvořena matematickou funkcí menší podmnožina dat. Například 9 sousedních pixelů obrázku je redukováno na 1 pixel.

Ve své podstatě konvoluční neuronová sítě dělá ztrátovou kompresi informace. Například vstupní obrázek o rozměrech 300x300 pixelů je převeden na kategorii obrázku (kočka, pes, auto).



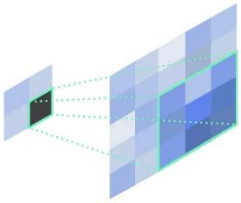
Obrázek 11 – Konvoluce



Obrázek 12 - topologie konvoluční sítě VGG16

2.3.1.4 Dekonvoluční neuronová síť

Dekonvoluční neuronové sítě jsou inverzní ke konvolučním sítím, kdy z menšího objemu vstupních informací vystupuje větší objem informací. Typickým příkladem je vygenerování větších obrázků. Chybějící informace jsou „inteligentně“ dopočítány ze vstupních informací.

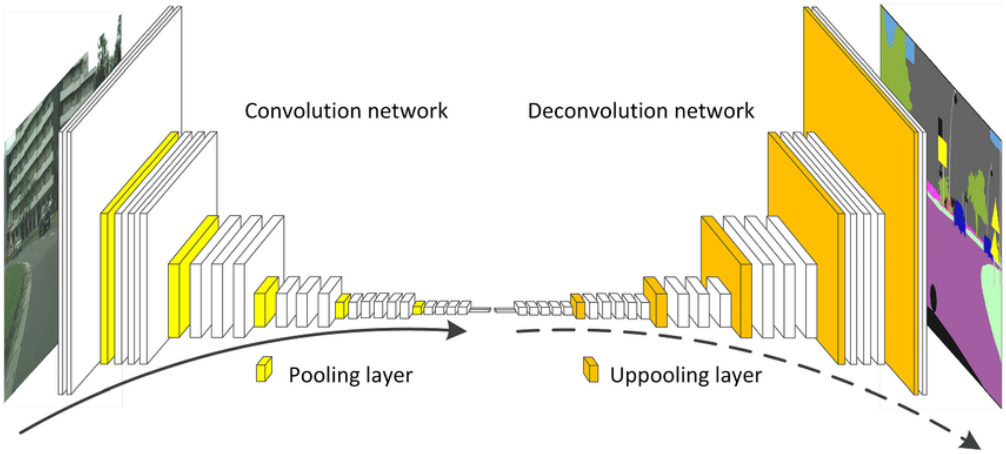


Obrázek 13 – Dekonvoluce

Velmi často se dokonvoluční sítě používají jako součást generativních sítí (GAN), které na základě vstupních dat vytváří nová data. Zpravidla tyto sítě obsahují konvoluční síť napojenou na dekonvoluční síť.

S těmito sítěmi se v praxi potkáme velmi často. Může se například jednat o funkce „inteligentního“ zvětšení obrazu, doplnění obrazu za vymazaný objekt. V těchto případech měla vstupní a výstupní data obrazovou podobu. Ale existují neuronové sítě, které na základě vstupního obrazu vygenerují textový popis obrázku. Komerční sítě jako Dall-E, midjourney fungují obráceně, kdy na základě slovního popisu vygenerují obrázek.

GAN sítě lze například použít pro generování testovacích dat, kdy se například GAN síť na vzorku pacientů naučí rozpoznávat různé typy pacientů/chorob a následně dokážeme generovat mnohem větší anonymizované vzorky požadovaných typů pacientů.



Obrázek 14 - Generativní umělá neuronová síť (GAN)

2.3.2 Počet parametrů

Velikost vstupní vrstvy je dána počtem vstupních hodnot. Například neuronová síť pro rozpoznání barevných obrázků o rozměrech 300 x 300 pixelů bude mít 300 x 300 x 3 = 270 000 vstupních neuronů.

Počet výstupních neuronů je definován požadovaným výstupem. Například typicky pokud chceme vstupní obrázek řadit do 10 kategorií, bude mít výstupní vrstva 10 neuronů.

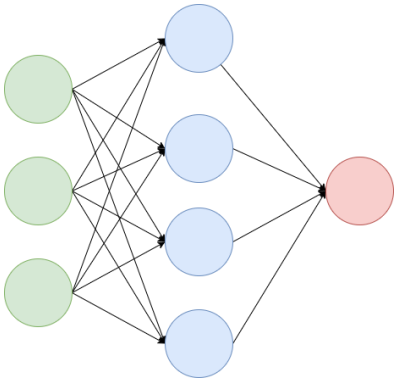
Připomeňme si, že každý neuron má n + 1 parametrů, kde n je počet vstupních hodnot upravovaných vstupními váhami a každý neuron má jednu hodnotu bias.

Celkový počet parametrů sítě lze spočítat pomocí následujícího vzorce, kde N je počet vrstev a N_i je počet neuronů v dané vrstvě. I začíná od 2, protože první vrstva neuronů je vstupní.

$$\sum_{i=2}^{N} N_i \cdot (N_{i-1} + 1)$$

Například pro následující neuronovou síť bude počet parametrů 21.

$$4 \cdot (3 + 1) + 1 \cdot (4 + 1) = 21$$



Obrázek 15 - plně propojená neuronová síť

3 Porovnání biologické a umělé neuronové sítě

V tuto chvíli, kdy jsme si představili fungování biologické a neuronové sítě, je vhodný okamžik na jejich porovnání, abychom si uvědomili rozdíly, a naopak stejné vlastnosti.

Biologické neurony jsou propojeny přes dendrity a axony a rozvětvené sítě. Šíření informace probíhá elektrochemicky a vzruchy jsou tlumeny nebo zesilovány. Vzruchy mají podobu částečně analogovou a částečně digitální. Při přenosu dochází k časovému zpoždění reakcí, díky rychlosti šíření elektrického signálu a rychlosti chemických reakcí. Rychlost šíření je od 0,61 do 119 m/s podle typu impulsu, stavu, pohlaví apod. Zároveň v biologickém mozku dochází ke změnám, kdy neurony odumírají (nové vznikají pouze v během vývoje) a dochází k vytváření a zániku neuronových synapsí.

Umělý neuron má několik vstupů, na kterých jsou signály tlumeny pomocí individuálních vah. Intenzita výstupu z umělého neuronu je určována spojitou aktivační funkcí. Jedná se tedy o matematicko-statistický model, který velmi často neřeší časové zpoždění při šíření signálu. Neuronové sítě velmi často nemají paměť pro uchování krátkodobých nebo dlouhodobých informací. Existují výjimky jako RNN sítě, které mají schopnosti podobné paměti. U umělé neuronové sítě nedochází k únavě. Umělé neuronové sítě zpravidla nepočítají se zánikem neuronů.

Rychlost biologických neuronů je limitována, některé neurony mohou sepnout až 200krát za sekundu. Nesmíme zapomenout na únavu. Neuron se po čase unaví a přestane na nějakou dobu fungovat, je to dáno biologicky a chemicky. Zatímco umělé neuronové sítě nejsou omezeny fyziologickými procesy a jejich rychlost je dána schopností provádět matematické operace nad matice. Současné GPU a CPU pracují v řádech GHz, tedy 1 000 000 000 sepnutí za sekundu. Umělé neuronové sítě se neunaví.

Biologický mozek člověka se skládá přibližně z 86 miliard neuronů a 100 bilionů synapsí. Velikost neuronové sítě je zpravidla omezena velikostí GPU paměti, počet umělých neuronů a parametrů je mnohonásobně nižší.

Zajímavé je i porovnání energetické náročnosti. Lidský mozek spotřebovává asi 20% energie, tedy asi 20W při teplotách okolo 36°C. Nvidia GeForce Titan X GPU má spotřebu 250W při teplotách 50-80°C. Právě velká energetická náročnost umělých neuronových sítí vzhledem k jejich velikosti a schopnostem může být brzdou ve vývoji umělé inteligence.

Topologie biologického mozku se v čase mění. Mění se počet neuronů a jejich propojení. Zároveň je lidský mozek velmi odolný proti chybám. Obsahuje mnoho redundantních neuronů, má určitou míru regenerace a schopnost průběžně se učit.

Současné umělé neuronové sítě mají topologii danou od počátku a v čase se nemění. Umělé neuronové sítě v zásadě nemusí řešit postupnou degradaci.

Zajímavou společnou schopnost mají biologické i neuronové sítě, je to schopnost zpracovávat velký objem dat a vybírat si pouze podstatné informace. Vlastně dochází k filtrování a kompresi informací bez podstatného vlivu na funkci.

Obrázek 16 - barevný nebo černobílý obrázek?

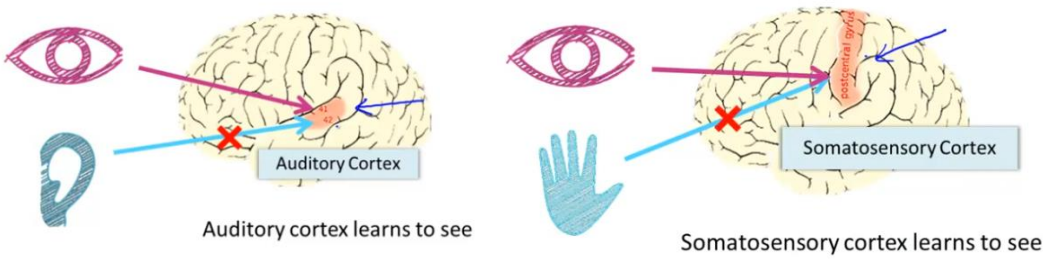
Naopak velké rozdíly jsou v procesu učení. Pro učení umělé neuronové sítě se používá algoritmus backpropagation, který bude vysvětlen později. Nově vytvořená umělá síť je inicializována náhodnými hodnotami parametrů a na základě učebního datasetu se pomalu hodnoty parametrů sítě upraví tak, aby dokázala řešit daný problém. Proces učení tedy ze „substrátu“ vyladí neuronovou síť pro danou úlohu. Velikost umělé neuronové sítě principiálně není omezena, v současnosti narážíme na technické a energetické limity.

Lidský mozek a proces učení je mnohem komplikovanější a neustále probíhá jeho výzkum. Velikost, struktura je dána geneticky. Stejně tak jako základní funkce, které jsou nezbytné pro udržení života a přežití, jedná se o soubor různých reflexů, nevědomého ovládání těla, … Určité oblasti mozku jsou určeny pro zpracování daných úloh, kterých paralelně zpracováváme velký počet.

Velká část úloh ovšem není vrozená a je třeba se je naučit. Proces učení se vyskytuje napříč různými biologickými druhy. Právě u lidského druhu je důraz na učení velký, a dokonce se tomu přizpůsobilo biologické fungování organismu, kdy je například proti jiným druhům výrazně posunuto dospívání. Proces učení probíhá po celý život, čím se velmi odlišujeme od současných umělých neuronových sítí, kdy je oddělena fáze učení a fáze provozování.

Neurochirurgové provedli zajímavé pokusy, na kterých můžeme ilustrovat velkou flexibilitu biologické neuronové sítě.

Při pokusech například přerušili spojení mezi sluchovým orgánem a odpovídající části mozku (auditory cortex) a nově ho propojili s očima. Auditory cortex se po čase naučil zpracovávat zrakové signály. Při podobném pokusu byl zrakový nerv propojen na somatosensory cortex, zodpovědný za zpracování hmatu, a opět došlo k naučení zpracování zrakových signálů.



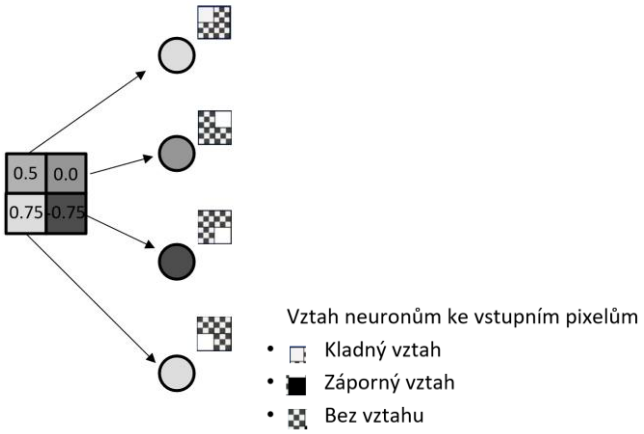
Podobně se slepí lidé mohou naučit orientovat v prostoru pomocí echolokace, kdy vydávají mlaskavé zvuky.

Podobně se chová i již naučená neuronová síť, pokud ji v nové fázi učení předložíme jiná trénovací data. I zde dojde k úpravě vah a přizpůsobení se novým datům.

4 Fungování umělé neuronové sítě

Fungování dopředné neuronové sítě si ukážeme na příkladu detekce směru čar z výstupu 4pixelové černobílé kamery.

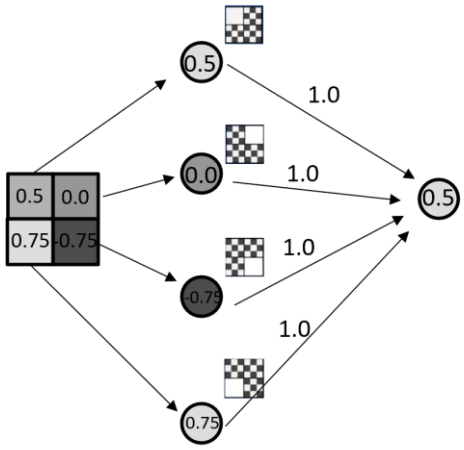
Na vstupu budeme mít hodnoty 4 pixelů a naším úkolem je určit, zda jsou čáry svislé, vodorovné, šikmé nebo žádné. Vytvoříme mapování barvy pixelu ve formátu RGB na hodnoty v intervalu <-1, 1> tak, že 000000 – FFFFFF → -1 – +1



Obrázek 17 – vstupní vrstva

Předpokládejme, že bias všech neuronů bude 0 a hodnota všech vah bude 1. Pak hodnota, která vstupuje do neuronu bude 0,5.

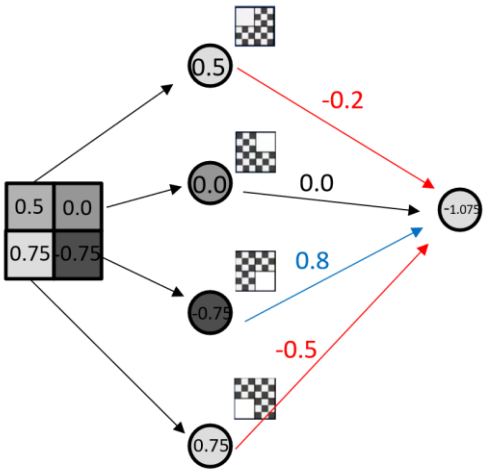
$$0{,}5 \cdot 1{,}0 + 0 + 0{,}0 \cdot 1{,}0 - 0{,}75 \cdot 1 + 0{,}75 \cdot 1 = 0{,}5$$



Obrázek 18 - výpočet hodnoty neuronu

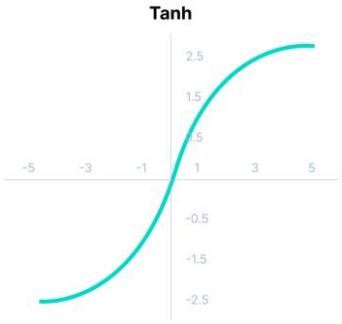
V průběhu učení se mohou váhy měnit, mohou být kladné, záporné nebo nulové. Například pokud váhy budou (-0,2; 0,0; 0,8; -0,5), pak vstupní hodnota do neuronu bude -1,075.

$$0{,}5 \cdot -0{,}2 + 0{,}0 \cdot 0{,}0 - 0{,}75 \cdot 0{,}8 + 0{,}75 \cdot -0{,}5 = -1{,}075$$



Obrázek 19 - výpočet hodnoty neuronu

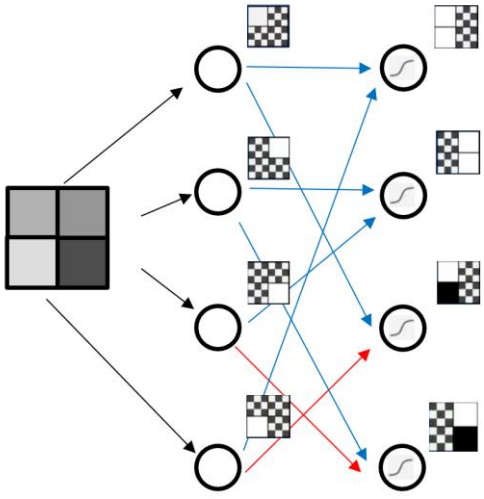
Výstupní hodnotu z neuronu vypočítáme vložením vstupní hodnoty do aktivační funkce. V našem případě použijeme jako aktivační funkce hyperbolický tangent, který má podobný tvar jako funkce sigmoid.



Obrázek 20 - funkce hyperbolický tangent

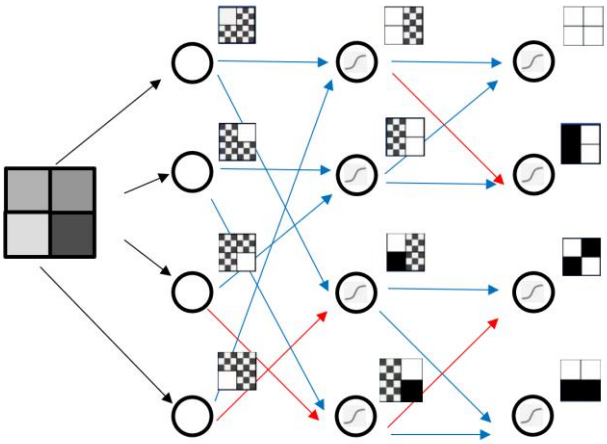
Výstupní hodnota neuronu bude $\tanh(-1{,}075) = -0{,}7913$

Ke vstupním neuronům připojíme první skrytou vrstvu, která se bude skládat ze 4 neuronů s biasem 0. Aktivační funkcí pro všechny neurony bude hyperbolický tangent. V tomto příkladu neuronové sítě budeme uvažovat, že váhy mohou mít hodnotu +1, pak budou znázorněny modrou šipkou, o váhu -1 a budou znázorněny červenou šipkou. Váha vazby může být i 0, pak vstup nemá žádný vliv na neuron a šipku vynecháme. Z původní plně propojené sítě se vlastně některé vazby funkčně odeberou. Můžeme si povšimnout, že na hodnotu neuronů této první skryté vrstvy mají vliv dva pixely.



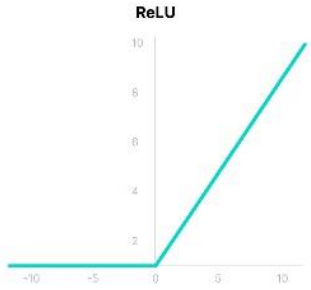
Obrázek 21 - první skrytá vrstva

Stejným způsobem do neuronové sítě přidáme druhou skrytou vrstvu.

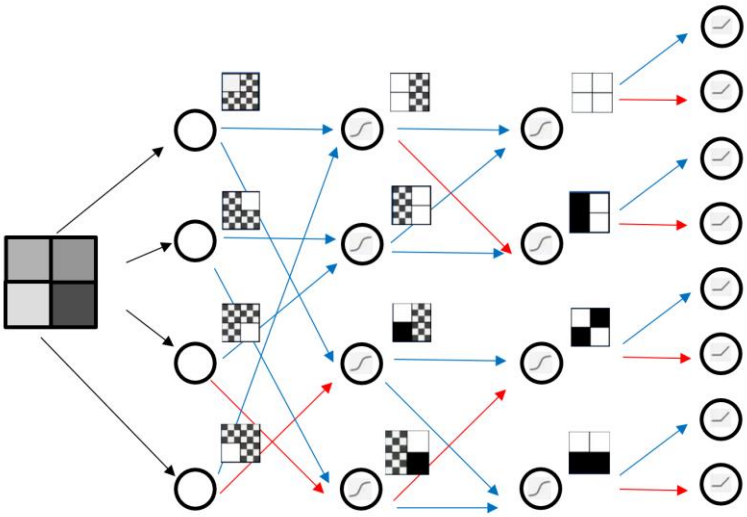


Obrázek 22 - druhý skrytá vrstva

Do neuronové sítě přidáme poslední skrytou vrstvu. Ta tentokrát bude obsahovat 8 neuronů s biasem 0. Změnou je, že tentokrát jako aktivační funkci použijeme ReLU. Jedná se o funkci, která záporné hodnoty mění na 0 a kladné hodnoty nechává v původní hodnotě.

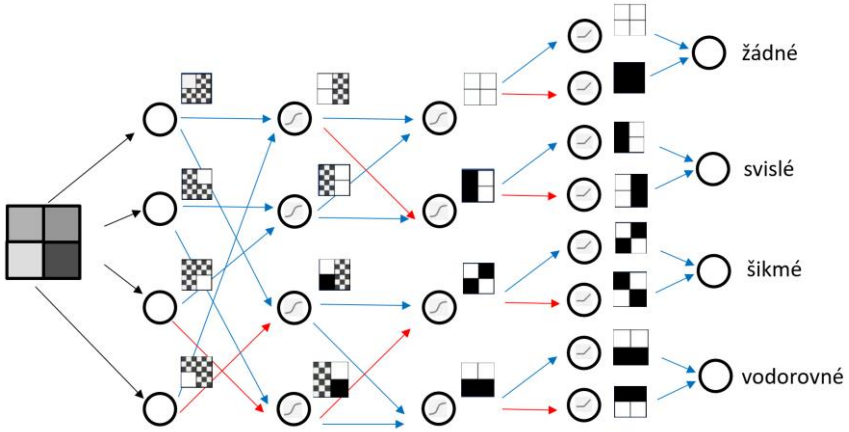


Obrázek 23 - aktivační funkce ReLU



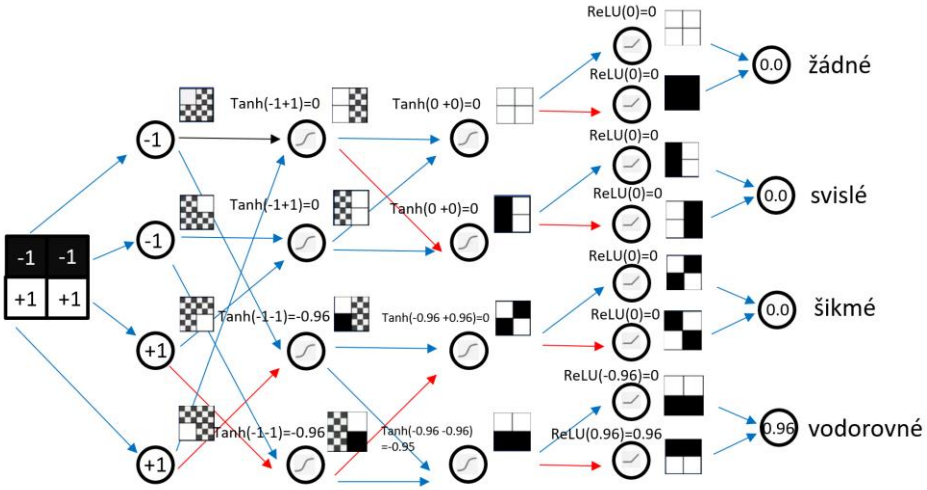
Obrázek 24 - třetí skrytá vrstva

Do modelu přidáme výstupní vrstvu, která bude obsahovat 4 neurony. Každý odpovídá jedné výstupní kategorii – svislé, vodorovné, šikmé a žádné čáry.



Obrázek 25 - výstupní vrstva

Fungování neuronové sítě si můžeme ověřit na extrémním případě, kdy obrázek obsahuje úplně černou a úplně bílou vodorovnou čáru. Pro dané hodnoty všechny výstupní neurony budou mít hodnotu 0 mimo neuronu, který reprezentuje vodorovné čáry. Ten bude mít hodnotu 0,96.



Obrázek 26 - výpočet neuronové sítě

Z výše popsaného chování neuronové sítě vyplývá, že vstupní data prochází jednotlivými vrstvami neuronové sítě. Při výpočtu každé vrstvy se počítají stejné matematické operace nad množinou perceptronů. Jak bylo ukázáno v kapitole o perceptronech tyto operace dokážeme vyjádřit jako násobení matic.

4.1 AI Framework

Vytvářet neuronové sítě můžeme v libovolném programovacím jazyku a můžeme začít i od základů. Tento přístup není špatný ve chvíli, kdy se snažíme pochopit podstatu neuronových sítí. Pokud chceme zvýšit svoji produktivitu, můžeme zvolit jeden z mnoha dostupných frameworků. Tyto frameworky již obsahují implementované optimalizované algoritmy, nástroje pro zpracování dat a další potřebné funkce pro strojové učení. Vývojáři se tak mohou soustředit na řešení konkrétního problému, nezabývat se technickými detaily implementace AI.

Při výběru frameworku je třeba zvážit různé klíčové faktory, aby framework co nejlépe splňoval požadavky projektu a úroveň odbornosti vývojářů.

4.1.1 Parametry

4.1.1.1 Výkon

Výkon by měl být při výběru kritériem s nejvyšší prioritou. Je vhodné zvolit framework, který dokáže efektivně zpracovávat data a poskytuje rychlé časy učení a provozování. Pokud v projektu budeme používat naučené modely na méně výkonných zařízeních (hodinky, mobily, mikropočítače, …), je vhodné otestovat schopnost frameworku fungování na těchto zařízeních, a i jeho rychlost.

4.1.1.2 Podpora komunity

Aktivní komunita je pro vývoj AI naprosto nezbytná. Při řešení problémů je vhodné mít přístup k široké řadě zdrojů, výukových programů a komunitně řízených pluginů spolu s podporou pro implementaci průběžných vylepšení a aktualizací frameworku.

4.1.1.3 Flexibilita

Jak již bylo naznačeno v předchozích kapitolách, projekty strojového učení lze řešit pomocí různých přístupů a algoritmů. Výběr správného algoritmu, který vytvoří nejlepší model, je otázkou zkušeností, ale i experimentování. Proto je při práci na projektech AI důležitá flexibilita frameworku. Nejlepší frameworky jsou ty, které poskytují možnost experimentovat s různými algoritmy. Kromě toho by měl být framework schopen se bez problémů přizpůsobit různým typům dat, jako je text, obrázky a zvuk, a integrovat se s dalšími technologiemi.

4.1.1.4 Snadnost učení

Pro začátečníky v oblasti umělé inteligence je vhodné volit frameworky, které jsou jednoduché na pochopení a jsou přívětivější. Je potřeba, aby uživatelům poskytovali kvalitní dokumentaci, příklady, školení apod. Tyto přívětivější frameworky nemusí poskytovat všechnu funkcionalitu, ale tu stejně začátečník mnohdy nepoužije a nepochopí.

Později lze přejít na komplexnější frameworky.

4.1.1.5 Licence

Existují open source i komerční AI frameworky. Každá skupina má svoje výhody a nevýhody.

Open-source frameworky jsou ty, které jsou vydány pod open-source licencí, která uživatelům poskytuje příležitost pracovat se softwarem pro jakýkoli účel. Obvykle se používají zdarma , takže jsou cenově dostupné pro malé projekty a začínající podniky. Často mají silnou a aktivní komunitu , kterou lze použít jako cenný zdroj pro učení a řešení problémů.

Na druhou stranu u open-source frameworku si neplatí podporu. I když je podpora komunity užitečná, nemusí být tak komplexní a rychlá jako komerční podpora.

Komerční frameworky jsou vyvíjeny společnostmi, které uvolňují svůj software pod proprietárními licencemi. To znamená, že uživatelé těchto frameworků jsou omezeni v tom, co mohou se softwarem dělat, a také mohou podléhat dalším poplatkům. Uživatelé komerčních rámců však mohou těžit z dalších funkcí a podpory ze strany dodavatele. Zároveň se často zaměřují na uživatelskou přívětivost, díky čemuž jsou přístupnější pro vývojáře všech úrovní dovedností. Nevýhodou komerčních frameworků může být uzamčení se k jednomu dodavateli.

4.2 Přehled frameworků

Pro vývoj modelů strojového učení se používají populární AI frameworky, jako jsou TensorFlow a PyTorch. Tyto rámce poskytují komplexní sadu nástrojů, které umožňují vývojářům snadno vytvářet a nasazovat modely ML. Mezi další užitečné knihovny umělé inteligence patří Scikit-Learn, Keras a Caffe. Tyto knihovny poskytují sadu rozhraní API, která umožňují vývojářům rychle vyvíjet aplikace, aniž by museli psát celou základnu kódu od začátku.

4.2.1 PyTorch

Torch je open-source knihovna pro strojové učení známá pro svůj dynamický výpočetní graf a je oblíbená vědci. Za knihovnou stojí společnost Meta. Framework je vynikající pro prototypování a experimentování. Navíc je posílena rostoucí podporou komunity s nástroji jako PyTorch , které jsou postaveny na knihovně. PyTorch se rychle stal jedním z nejpoužívanějších frameworků, který je užitečný ve všech druzích aplikací.

4.2.2 TensorFlow

TensorFlow je open-source framework, byl vyvinutý společností Google. Je známý svou flexibilitou a škálovatelností, díky čemuž je vhodný pro mnoho aplikací AI. Tento framework má velkou a aktivní komunitu a je vybaven rozsáhlou dokumentací a návody. Podporuje také nasazení na různé platformy. Křivka učení TensorFlow však může být pro začátečníky strmá.

4.2.3 Keras

Keras je open-source API pro neuronové sítě, který se snaží být uživatelsky přívětivější než výše zmíněný pytorch a tensorflow. Keras na pozadí používá jiné frameworky jako tensorflow, pytorch, theano atd. Keras je vhodný pro začátečníky a rychlé prototypování. Na druhou stranu Keras může postrádat některé pokročilé funkce pro složité úkoly.

4.2.4 Scikit-Learn

Scikit-Learn je knihovna Pythonu pro strojové učení. Jedná se o open-source nástroj pro začátečníky, který nabízí možnosti dolování dat a strojového učení, stejně jako komplexní dokumentaci a výukové programy. Scikit-Learn se dobře hodí pro menší projekty a rychlé prototypování modelů, ale nemusí být tou nejlepší volbou pro úkoly hlubokého učení.

4.2.5 LangChain

LangChain si nedávno získal oblibu jako framework pro aplikace velkých jazykových modelů (LLM). Umožňuje vývojářům vytvářet aplikace pomocí LLM s funkcemi, jako jsou modelové I/O, datová připojení, řetězce, paměť, agenti a zpětná volání. LangChain se používá pro různé aplikace, jako jsou chatboti, sumarizace dokumentů a interakce s API.

4.2.6 OpenAI

OpenAI poskytuje řadu nástrojů pro různé úkoly AI, včetně vytváření obrázků nebo převodu textu na řeč. Je známá svými výkonnými jazykovými modely GPT , které dokážou porozumět a generovat text podobný lidskému textu. Platforma OpenAI je uživatelsky přívětivá a usnadňuje lidem používání pokročilé umělé inteligence ve vlastních projektech, zejména pro vytváření asistentů umělé inteligence nebo nástrojů, které komunikují s uživateli v přirozeném jazyce. Stojí za zmínku, že několik funkcí vyžadovalo placené prémiové předplatné.

4.2.7 IBM Watson

IBM Watson je sada služeb AI a strojového učení poskytovaná IBM. Nabízí nástroje a řešení pro vytváření a zavádění aplikací založených na umělé inteligenci, včetně zpracování přirozeného jazyka, počítačového vidění a prediktivní analýzy.

Lze jej snadno integrovat s IBM Cloud pro bezproblémové nasazení. A co víc, robustní schopnosti AI v sadě IBM Watson jsou podpořeny odbornými znalostmi IBM. Ceny však mohou být problémem pro menší podniky, které hledají komplexní řešení AI a konzultační služby.

4.2.8 Microsoft Cognitive Toolkit (CNTK)

Microsoft Cognitive Toolkit neboli CNTK je bezplatný a open-source framework pro hluboké učení vyvinutý společností Microsoft. Je známý svou efektivitou, zejména na systémech s více GPU, a

je vhodný pro výzkumné i produkční nasazení. Podporuje také více typů neuronových sítí, včetně dopředných a rekurentních sítí. Navíc poskytuje Python API pro snadné použití.

4.2.9 Caffe

Caffe je open-source framework pro hluboké učení. Je známý svou rychlostí a efektivitou v úlohách počítačového vidění a podporuje různé architektury hlubokého učení. Caffe je optimalizováno pro aplikace počítačového vidění a je vynikající pro nasazení na okrajových zařízeních. Při jeho výběru byste však měli zvážit jeho omezenou flexibilitu pro jiné úkoly.

4.2.10 Závěr

Stručně řečeno, PyTorch a TensorFlow jsou vynikající volbou pro projekty hlubokého učení, přičemž TensorFlow nabízí škálovatelnost a PyTorch klade důraz na flexibilitu. Scikit-Learn je výchozím bodem pro tradiční úlohy strojového učení, zatímco Keras poskytuje uživatelsky přívětivý vstupní bod pro hluboké učení. Caffe je nejlepší volbou pro počítačové vidění Novější nástroje jako LangChain a OpenAI poskytují vynikající možnosti pro velké jazykové modely (LLM).

4.3 GPU a TPU

Jaký je vhodný HW pro strojové učení? Několikrát bylo zmíněno, že při strojovém učení se velmi často používají operace s velkými maticemi. Aby tyto operace byly dostatečně rychlé, je třeba je paralelizovat.

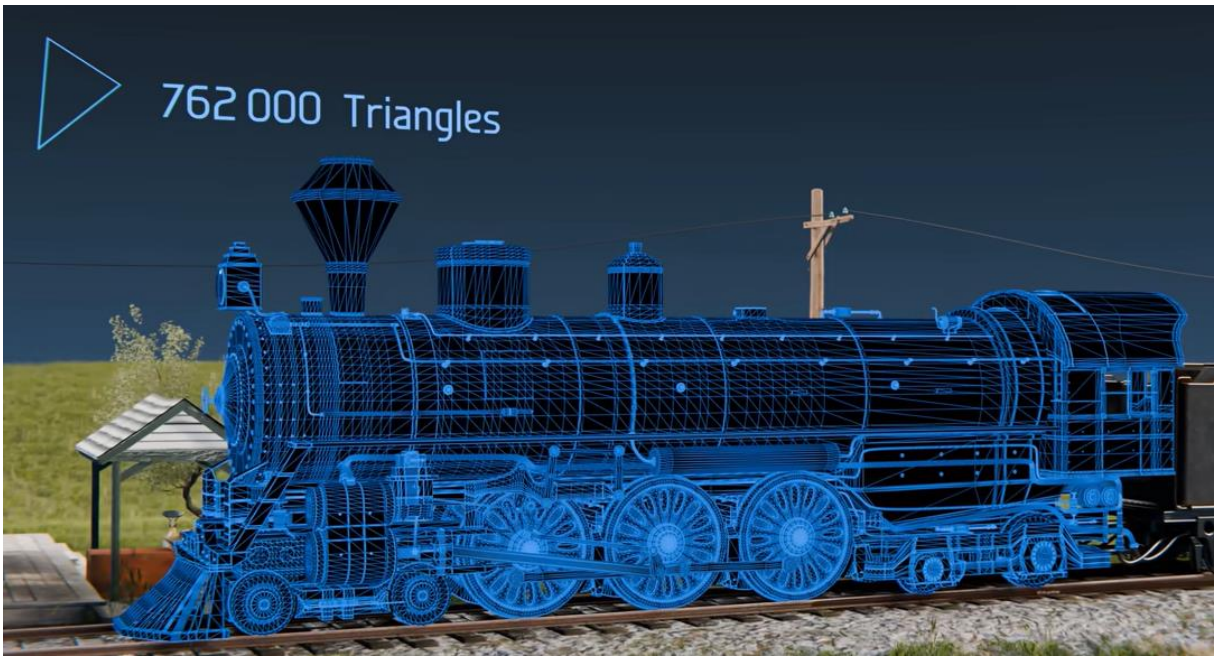
Vytváření velmi velkých modelů jako jsou například velké jazykové modely je v dnešní době velmi časově, energeticky i finančně náročné. S novým nástupem umělé inteligence tentokrát založené na velkých neuronových sítích, roste poptávka po HW řešení, které právě optimalizuje především trénování modelů.

4.3.1 GPU

Abychom pochopili, proč se často pro učení modelů umělé inteligence používají GPU, podívejme se na práci s obrazovými daty, na kterou byly GPU vyvinuty. Dnešní počítačové hry se velmi často snaží poskytnout co nejrealističtější herní prostředí. Aby se tohoto dosáhlo, je třeba mít vytvořený model herního světa s velmi malými detaily a v tomto modelu simulovat skutečnou fyziku.

Tato simulace musí být dostatečně rychlá, aby pohyb a akce v herním prostředí byly plynulé. Proto je třeba generovat měnící se obrázky rychlostí desítek framů za sekundu. Při dnešních velikostech rozlišení obrazu (4K) a velmi detailních modelech, jsou kladeny velké výpočetní nároky na grafickou kartu.

V prvním kroku při vytváření realistického modelu je třeba vytvořit detailní drátový model, který je tvořen malými plochami ve tvaru trojúhelníků. Trojúhelníky mají společné vrcholy a hranami.



Obrázek 27 - Drátový model

Při pohybu objektu dochází ke změnám 3D souřadnic jednotlivých vektorů. Souřadnice modelu je třeba umístit do souřadnicového systému herního světa. Pokud se objekt pohybuje, dochází ke změnám souřadnic. Tato změna se počítá pomocí násobení matic. Kdy jedna matice představuje všechny třírozměrné body modelu. Druhá matice se nazývá transformační a reprezentuje změny – translaci, rotaci, zvětšování atd. Transformační matice pro jednotlivé transformace lze mezi sebou násobit. Následně se výsledná transformační matice aplikuje na matici bodů. Aby byl výpočet dostatečně rychlý musí být paralelní. Pro tyto operace mají GPU přes 10 000 specializovaných CUDA jader.

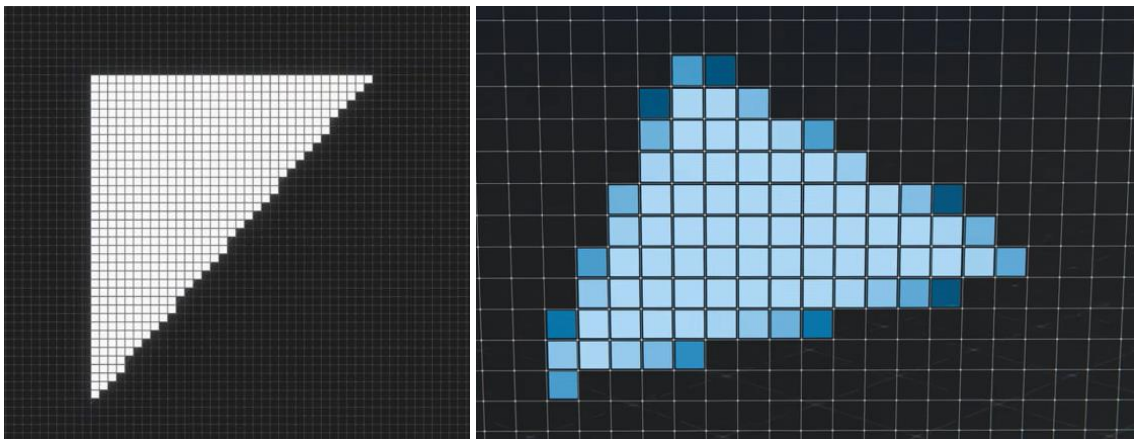
$$P = \begin{pmatrix} 1 & 0 & P_x \\ 0 & 1 & P_y \\ 0 & 0 & 1 \end{pmatrix} \quad R = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad S = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Pro vykreslení scény je důležitá pozice a parametry kamery. Tyto informace vymezují, jaké objekty je třeba počítat a které ne. GPU potřebuje rychle přistoupit k vybraným oblastem matic.

Každé ploše je navíc přiřazen materiál a textura. Materiál a textura ve spojení s osvětlením se používá pro výpočet barvy jednotlivých pixelů.

Obrázek 28 - scéna

Dalším krokem je rasterizace. Kdy každý trojúhelník je převeden na množinu pixelů, které se budou zobrazovat. Protože z pohledu kamery se jedním pixelem mohou nacházet různě vzdálené trojúhelníky, je třeba určit, který z trojúhelníků bude viditelný, a který ne. K tomu se například používá algoritmus nazývaný z-buffer.



Obrázek 29 - rasterizace

Každému trojúhelníku přísluší barva, materiál a textura. Na základě toho jsme schopni určit základní barvu pixelu. O to se starají TMU jednotky (Texture mapping unit). Tuto barvu ovlivňuje i světelná scéna, které obsahuje různé zdroje světa s různou barvou a intenzitou. Algoritmus, který vypočítává skutečnou barvu pixelu na základě zdrojů světla, materiálů, odrazů, stínů atd. se nazývá raycasting nebo raytracing. O výpočtu barvy se v GPU starají specializovaná jádra.

Obrázek 30 - randering

Zpracování obrazové scény se skládá z několika dílčích úloh, pro které GPU obsahuje velké množství specializovaných jader. Některé úlohy se řeší velmi rychlým paralelním násobením matic. A protože AI používá také násobení matic, lze GPU použít pro umělou inteligenci. Aby šlo provádět vědecké výpočty nad GPU, je třeba používat specializované knihovny, které tyto výpočty umožní. V případě GPU od Nvidia se jedná o knihovnu CUDA toolkit.

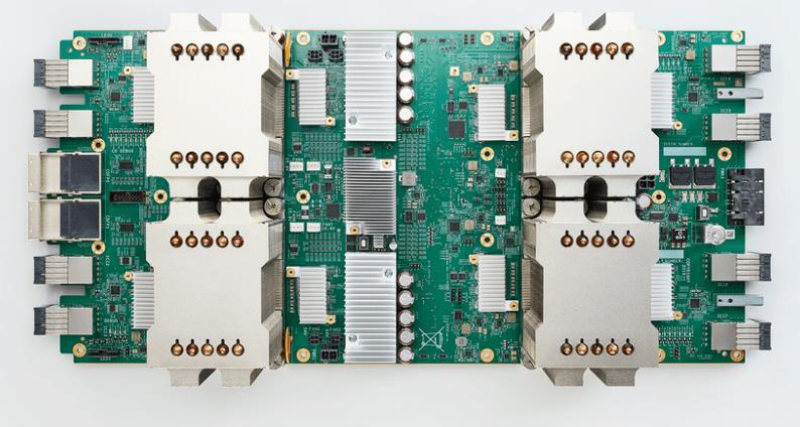
4.3.2 TPU

CPU je obecný procesor, který je velmi flexibilní a lze ho použít pro velkou škálu různých úloh. Při vykonávaní programu zpracovává instrukci po instrukci, které pracují s relativně omezenou velikostí paměti reprezentovanou registry. Dnešní procesory se sice skládají až z desítek CPU jader, ale to stále není dostatečný výkon pro metody strojového učení. Standardní CPU se hodí ve strojovém učení pouze pro jednoduché modely, případně učení, které zpracovává malý objem trénovacích dat.

GPU obsahují tisíce specializovaných jader, která mají funkci ALU. GPU dokáže paralelní počítat velké množství dat. Moderní GPU obsahují i velkou paměť (GB), do které se vejdou i relativně složité modely s velkým počtem parametrů. Protože zpracování grafických scén zahrnuje pouze násobení matic, i jádra GPU jsou relativně univerzální výpočetní jádra, která podle instrukcí zpracovávají data.

V poslední době se pro strojové učení začínají objevovat TPU (Tensor processing Unit). Tenzor je v matematice objekt, který je zobecněním pojmů vektor nebo matice. Zatímco složky vektoru je možné označit jedním indexem, matici dvěma indexy, u tensoru mohou mít složky více indexů. Tensor je tedy multidimenzionální struktura, která obsahuje data. TPU jsou navrženy pouze pro zpracování tensorů (násobení a sčítání tensorů), což je operace, která se používá v umělé inteligenci. Struktura

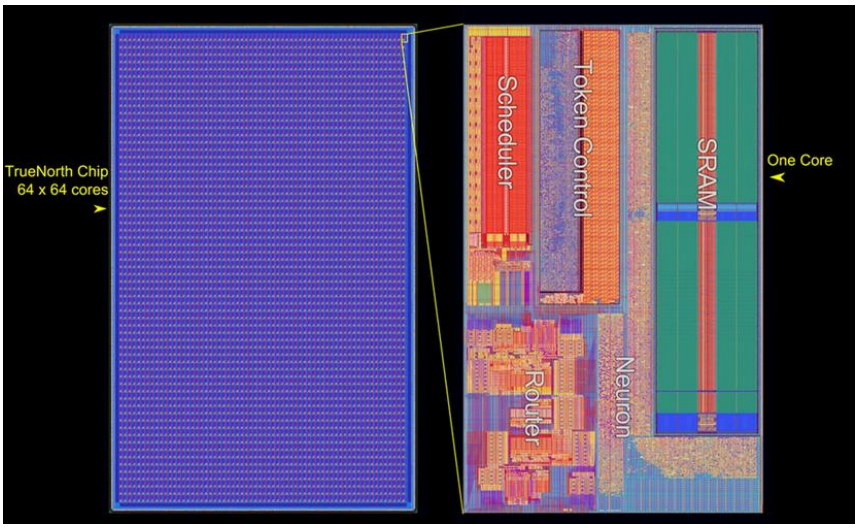
TPU se skládá z množství tensorových sčítaček propojených do struktury nazývané systolické pole. Tím se redukuje nutnost přistupovat do paměti pro jednotlivé hodnoty, protože mezivýsledky putují skrz toto pole.



Obrázek 31 – TPU

4.3.3 NorthPole

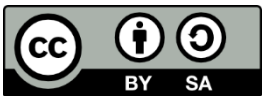
V roce 2023 IBM představilo neobvyklý procesor nazvaný NortPole. Jedná se o procesor, který je vysoce energeticky účinný. Dokáže efektivně spouštět (ne trénovat) neuronové sítě založené na odvození. U úloh, jako je klasifikace obrazu nebo přepis zvuku, může být čip až 35krát efektivnější než GPU. NorthPole, stejně jako předtím TrueNorth, se tedy skládá z velkého pole výpočetních jednotek, z nichž každá zahrnuje jak místní paměť, tak kapacitu pro provádění kódu. Takže všechny váhy různých spojení v neuronové síti mohou být uloženy přesně tam, kde jsou potřeba. Inspirace vychází z biologického mozku, který je mnohem energeticky účinnější než moderní počítače, zčásti proto, že ukládá paměť spolu s výpočty v každém neuronu. A toho využívá i NortPole. Budoucnost ukáže, jestli je toto správná cesta.



Obrázek 32 - Northpole

UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vytvořeno v rámci projektu **Digitalizace studijních Agend, Nové TechnologiE, systémy a přístupy k výuce na UPCE**, reg. č. NPO_UPCE_MSMT-16591/2022.



Financováno Evropskou unií
NextGenerationEU

Národní plán obnovy

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Machine learning & AI

Učení neuronových sítí a úloha klasifikace

Ing. Martin Pozdílek, Ph.D.

1 Učení umělých neuronových sítí

V předchozí kapitole jsme si představili základní principy umělých neuronových sítí a ukázali jsme si jejich fungování. Vstupní data prochází sérií neuronových vrstev, které je podle naučených hodnot vah a bias postupně transformují na výstup.

V této kapitole se podíváme na proces učení, tedy nastavování parametrů neuronové sítě.

1.1 Gradient descent algoritmus

Než začneme, tak si připomeneme gradient descent algoritmus, který jsme používali na určení hodnot parametrů w a b u lineárního modelu s rovnicí

$$f_{w,b} = w \cdot x + b$$

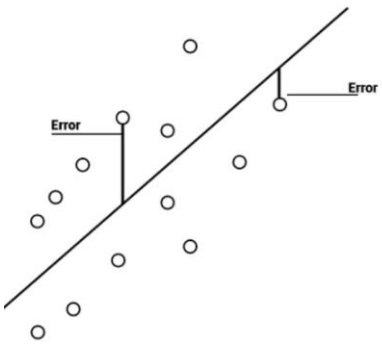
Zkusme ho porovnat s matematickou rovnicí agregační funkce perceptronu.

$$f = w^T \cdot x + b$$

Nyní už je možná pochopitelné, proč jsme zvolili pojmenování parametrů w a b, místo klasického a a b. Jsou tu určité odlišnosti, například, že vstupní hodnoty a váhy mají podobu vektoru. V perceptronu je výsledek agregační funkce korigován aktivační funkcí. Přes tyto odlišnosti pořád budeme pro učení moci použít nějakou modifikaci gradient descent algoritmu.

U lineární regrese jsme zvolili náhodné hodnoty parametrů w a b. Totéž provedeme při inicializaci neuronové sítě. Všechny váhy a biasy nastavíme na nenulové náhodné hodnoty. Důležité je použít nenulové hodnoty, protože během učení budeme počítat drobné korekce parametrů. Nulová hodnota by nám v tom zabránila. Pokud bychom nastavili všechny parametry na stejné hodnoty, tak vytvoříme všechny neurony identické a nepůjde je specializovat.

Připomeňme si, že u lineárního modelu jsme definovali nákladovou funkci J. Jedná se o funkci, která nám nějakým způsobem ohodnocuje, jak daleko je výsledek modelu od skutečné správné hodnoty. U lineárního modelu se jedná o vzdálenost skutečných bodů od přímky.



Obrázek 1 - chyba je rozdíl mezi skutečnou a predikovanou hodnotou

U lineárního modelu jsme používali jako nákladovou funkci Střední kvadratická chyba (Mean Squared Error MSE nebo Mean Squared Deviation MSD).

$$\mathrm{J}(w,\ b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^i) - y^i)^2$$

Někdy jsme používali RMSE – Root mean square error.

$$\mathrm{J}(w,\ b) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^i) - y^i)^2}$$

Někdy se používá střední absolutní chyba (Mean Absolute Error MAE), která je lépe interpretovatelná.

$$\mathrm{J}(w,\ b) = \frac{1}{2m}\sum_{i=1}^{m}\left|f_{w,b}(x^i) - y^i)\right|$$

Gradient descent algoritmu se snaží minimalizovat nákladovou funkci, tedy najít její lokální minimum. U lineárního modelu se na základě trénovacích dat vypočítá nákladová funkce, jejímiž parametry jsou w a b. Následně se spočítá parciální derivace parametrů w a b, která určí směr a velikost změny parametrů w a b.

Pokud jsme pro nákladovou funkci použili následující definici

$$\mathrm{J}(w,\ b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,\,b}(x^i) - y^i)^2$$

Pak můžeme vyjádřit parciální derivace následovně:

$$\frac{\partial}{\partial w}J(w,\ b) = \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^i) - y^i)x^i$$

$$\frac{\partial}{\partial b}\ J(w,\ b) = \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^i) - y^i)$$

Velikost změny, kroku, upravujeme koeficientem míry učení (learning rate) α. Jeho velikost nám určuje, jak rychle najdeme lokálními minimum. Při příliš vysokém α může dojít k rozbití fungování algoritmu.

Algoritmus gradient descent vypadá následovně:

Opakuj, dokud nedojde ke konvergenci

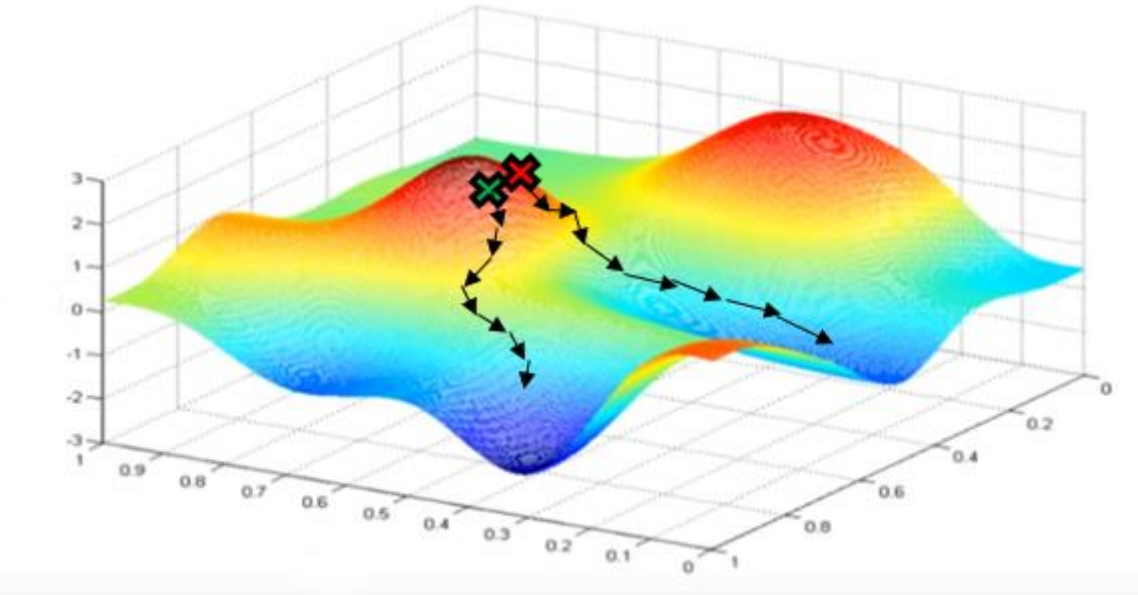
$$\textbf{step\_0} \coloneqq w - \alpha \frac{\partial}{\partial w} J(w,\, b)$$

$$\textbf{step\_1} \coloneqq b - \alpha \frac{\partial}{\partial b} J(w,\, b)$$

$$w \coloneqq \textbf{step\_0}$$

$$b \coloneqq \textbf{step\_1}$$

Graficky ho můžeme zobrazit jako cestu po povrchu nákladové funkce.



Obrázek 2 - znázornění gradient descent algoritmu

Výpočet gradientu (parciálních derivací) u neuronových sítí je mnohonásobně složitější, protože místo dvou parametrů jich máme tisíce ba i miliony. Zároveň pro správné vytrénování umělých sítí jsou vhodné velmi rozsáhlé datasety. Aby byl proces učení vůbec výpočetně proveditelný, později si ukážeme dávkování a backpropagation algoritmus.

Připomeňme si, že gradient descent dokáže najít jen lokální minimum. Totéž platí i při učení umělých neuronových sítí.

Při nepříhodných počátečních hodnotách parametrů se může zarazit poměrně daleko od globálního minima. Proto jsme algoritmus spouštěli vícekrát s různými počátečními hodnotami parametrů. Toto řešení je možné i u umělých neuronových sítí. Z většího počtu lokálních minim vybere

to nejmenší a s nějakou pravděpodobností je toto lokální minimum i globálním minimem, nebo se k němu blíží.

U složitých neuronových sítí, které se počítají mnoho hodin nebo dní, je nutné učení opakovat mnohonásobně. Proto se místo standardního Gradient Descent algoritmu používají i jiné optimalizační algoritmy jako je Adam, Adaptive Gradient Algorithm (AdaGrad), Root Mean Square Propagation (RMSProp), SGD a mnoho dalších. V principu tyto optimalizační algoritmy přidávají vlastnosti jako je momentum, velocity apod., které jim umožňují do určité míry překonávat lokální minima a najít globální minimum.

Při trénování neuronové sítě tedy budeme muset zvolit vhodnou nákladovou funkci, vhodnou míru učení α a správný optimalizační algoritmus. Pokud používáme nějaký AI framework, tak tyto změny jsou velmi jednoduché.

1.2 Stochastic Gradient Descent

Učení složité neuronové sítě na velkém tréninkovém datasetu je výpočetně velmi náročné. V původní verzi Gradient Descent algoritmu je třeba vypočítat odchylky pro všechny trénovací údaje. Na základě odchylek se počítají parciální derivace všech parametrů modelu. Z parciálních derivací modelu se vypočítají drobné změny parametrů modelu, které se hromadně aplikují. Proces učení pokračuje další iterací až do chvíle, kdy se hodnota nákladové funkce blíží 0, nebo provedeme n učebních iterací nebo změna hodnoty nákladové funkce se dlouhodobě výrazně nemění.

Stochastic Gradient Descent algoritmus optimalizuje proces učení tím, že v každé iteraci pracuje pouze s podmnožinou trénovacích dat. Pro dané parametry se tedy dozvíme pouze odhad hodnoty nákladové funkce. Výpočet parciálních derivací na základě tohoto vzorku nebude zcela přesný, ale bude vypočten mnohem rychleji. Drobné změny parametrů tedy nejsou zcela optimální, ale ve výsledku dokonvergují do lokálního optima stejně, jako kdybychom používali úplná testovací data. Nepřesných kroků bude více, ale budou spočítány mnohem rychleji, takže celkový čas bude kratší než u původní verze Gradient Descent algoritmu.

Předpokladem pro správné fungování stochastického algoritmu je, že podmnožiny (dávky, mini-batch) v jednotlivých krocích jsou reprezentativní vzorky trénovacích dat. V praxi se používá permutace indexů tréninkových dat, aby byla tréninková data v procesu učení zastoupena rovnoměrně.

1.3 Nákladová funkce

Při trénování neuronové funkce je třeba zvolit vhodnou nákladovou funkci. V principu hledáme takovou funkci, která nám pro danou úlohu nejlépe vyjádří, jak je predikovaný výsledek modelu daleko

od skutečnosti. Nákladová funkce musí být nezáporná, jinak by optimalizační algoritmy nepracovaly správně. Pokud je model správně natrénovaný, pak se hodnota nákladové funkce pro trénovací, testovací i validační data blíží 0.

Nákladová funkce se volí podle typu problému. Pro regresní úlohy, tedy predikci výstupní hodnoty na základě vstupních parametrů, se stejně jako u lineární regrese často používají funkce o **MSE** – Mean square error a **MAE** – Mean absolute error, které měří vzdálenost predikované veličiny od modelu. Predikovaná veličina se může skládat i z více hodnot, a proto je třeba chybu vyjádřit v mnoho rozměrovém prostoru.

Pro úlohy, ve kterých se na obrázku detekuje objekt, se používají funkce Region proposal network (RPN) loss nebo Intersection over Union (IoU), ty si ukážeme později.

Pro jiný typ úloh lze samozřejmě použít i jiné nákladové funkce, které nejlépe vyjádří přesnost predikce modelu od skutečnosti.

Malá změna vah nebo vstupů se má promítnout do výstupu jen málo. Ideálně lineární proporční změna.

1.4 Back propagation

Doposud se Gradient Descent Algoritmus používaný v lineární regresi příliš nelišil od podoby používané při trénování neuronových sítí. V rámci učební iterace provedeme pro dávku trénovacích dat dopředný výpočet v neuronové síti, který nám vrátí predikovanou hodnotu. Pomocí nákladové funkce vypočítáme průměrné rozdíly mezi predikovanými a skutečnými výstupními hodnotami.

Nyní následuje výpočet parciálních derivací nákladové funkce podle parametrů modelu, abychom z nich získali drobné změny těchto parametrů. U lineárního modelu je výpočet parciálních derivací jednoduchý.

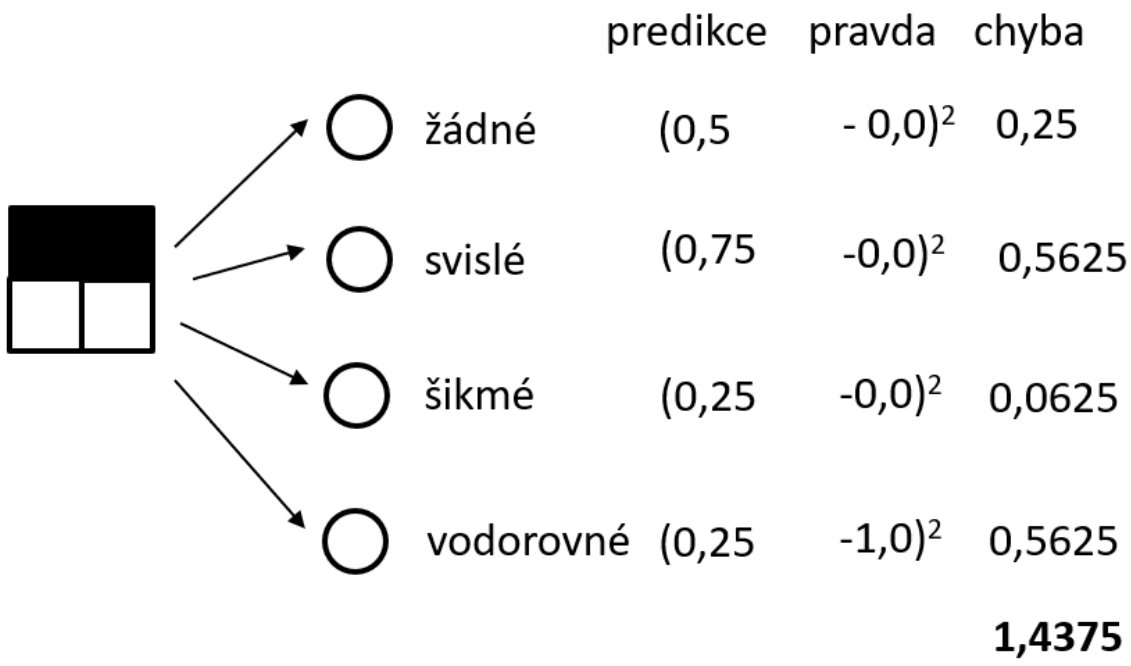
$$\frac{\partial}{\partial w} J(w,\ b) = \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^i) - y^i) x^i$$

$$\frac{\partial}{\partial b}\ J(w,\ b) = \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^i) - y^i)$$

Složitější je výpočet parciálních derivací pro velké množství parametrů mnohovrstevné sítě, která model od modelu mění svoji strukturu, aktivační funkce atd. Snažíme se vlastně určit, jak malá změna jedné váhy nebo biasu u neuronu má mít vliv na fungování neuronové sítě jako celku.

Pro výpočet parciálních derivací se používá back propagation algoritmus (zpětná propagace chyb).

Uveďme si příklad, dříve uvedená klasifikační síť 4pixelové kamery, která ještě není zcela dotrénovaná. Neuronová síť s aktuálními hodnotami parametrů nám pro ilustraci pro jeden konkrétní vstup bude vracet vektor (0,5; 0,75; 0,25; 0,25). Přičemž správný výstupní vektor je ovšem (0; 0; 0; 1). Proto budeme chtít upravit parametry modelu (váhy a bias) tak, aby se aktivační hodnota u prvních 3 neuronů snížila a u posledního zvýšila. Chyby jednotlivých neuronů chceme distribuovat proporčně. Tedy je pro nás důležitější zvýšit hodnotu čtvrtého neuronu a snížit hodnotu druhého neuronu než snížit hodnotu prvního neuronu. Snížení hodnoty třetího neuronu má nejnižší prioritu.



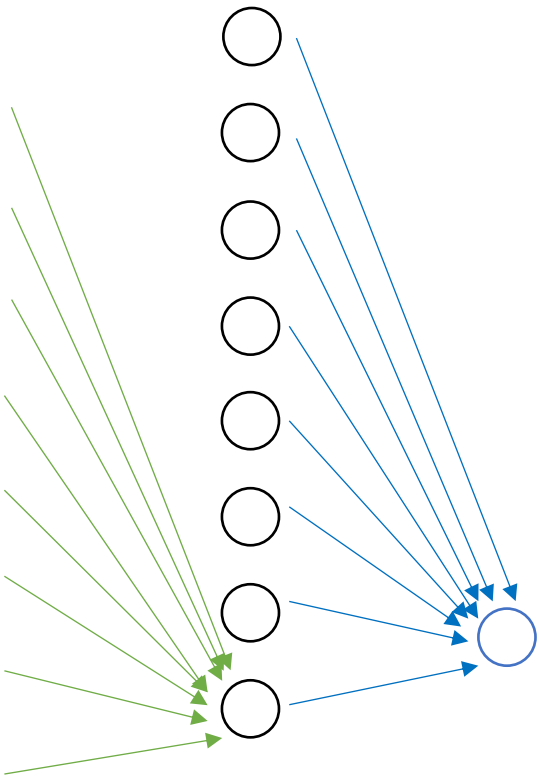
Obrázek 3 - chyba neuronové sítě

Věnujme se nyní pouze čtvrtému neuronu. U něj potřebujeme zvýšit hodnotu. Ta je závislá na

- vahách w pro jednotlivé vstupy do neuronu
- bias neuronu
- na příchozích aktivačních hodnotách z předchozích neuronů

Pokud tedy chceme zvýšit hodnotu posledního čtvrtého neuronu, můžeme zvýšit váhy w_1 až w_8, bias nebo vstupní aktivační hodnoty a_1 až a_8, naznačené modrou barvou. Zároveň zvýšení váhy u aktivační hodnoty, která je již vysoká, budeme mít větší vliv na výslednou hodnotu než zvýšení váhy u aktivační hodnoty jiného neuronu, která je podstatně menší.

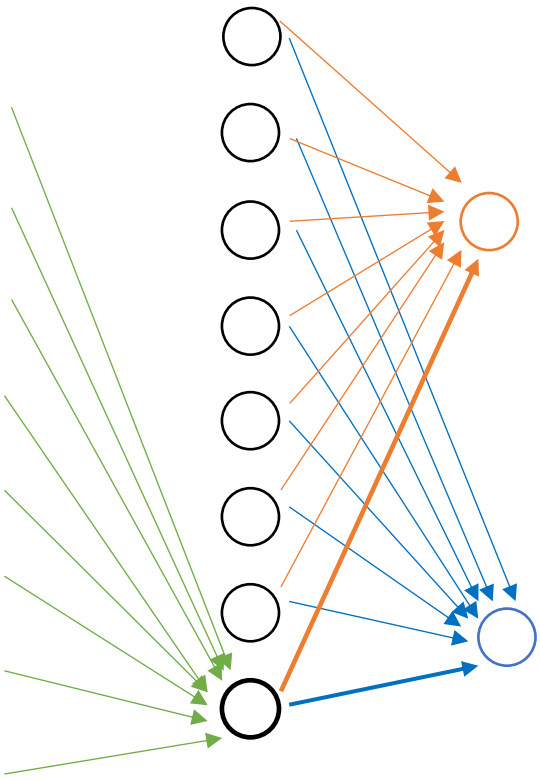
Aktivační hodnotu jednoho vstupu nemůžeme přímo ovlivnit, ale můžeme rekurzivně ovlivnit váhy, bias a aktivační hodnoty, které vstupují do předchozího neuronu, vyznačeno zelenou barvou.



Obrázek 4 – ovlivnění hodnoty neuronu

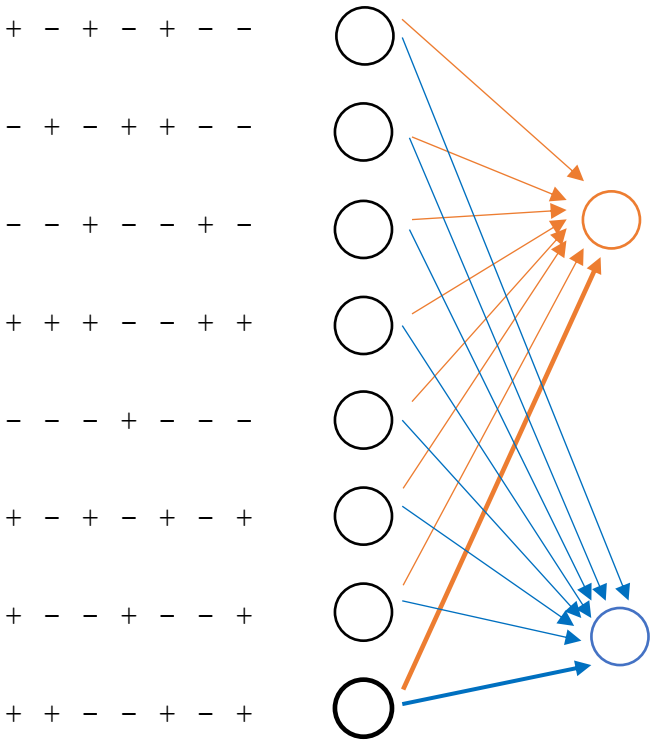
Dosud jsme uvažovali pouze jeden výstupní neuron, u kterého chceme zvýšit hodnotu. V neuronové síti jsou i další neurony, u kterých chceme naopak snížit aktivační hodnotu. Například pokud chceme u oranžového neuronu snížit hodnotu, můžeme ovlivnit jeho bias, váhy oranžových spojení nebo ovlivnit příchozí aktivační hodnoty z předchozí vrstvy.

Při ovlivňování příchozích aktivačních hodnot se dostáváme do střetu s požadavky modrého neuronu, který chce také upravovat aktivační hodnoty předchozí vrstvy. Omezit konflikty lze například správným nastavením vah, aby oranžový neuron byl více ovlivňován jinými předchozími neurony než modrý neuron.



Obrázek 5 - Obrázek 4 – ovlivnění hodnoty neuronu

Každý záznam v trénovacích datech bude mít jiný požadavek na aktivační hodnoty předchozích neuronů. Budou se lišit velikosti i směr požadovaných změn. Požadavky v rámci dávky, jedné iterace, se zprůměrují a provede se hromadná změna všech parametrů upravená mírou učení α.



Obrázek 6 - různě požadavky na změnu parametrů

Na backpropagation se nyní podíváme z pohled matematiky.

Mějme trénovací množinu ({x_1, x_2}, y). Provedeme dopřednou fázi tak, jak je naznačeno na obrázku. Hodnota **z** je výsledek agregace aktivačních hodnot do neuronu vynásobených individuálními váhami.

$$z^k = w^k a^{k-1} + b^k$$

Hodnota z je následně upravena aktivační funkcí **g**. V tomto případě budeme používat jako aktivační funkci sigmoid.

$$g(z) = 1 - \frac{e^{-z}}{1 + e^{-z}}$$

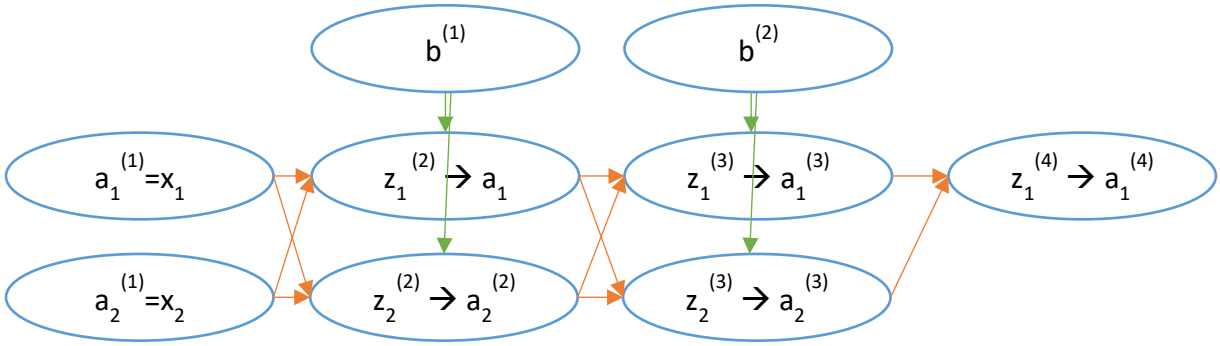
Tuto hodnotu budeme označovat za aktivační hodnotu **a**.

$$a^k = g(z^k)$$

Pro první vrstvu je aktivační hodnota rovna vstupní hodnotě. Pro druhou vrstvu aktivační hodnota vychází z aktivační hodnoty předchozí vrstvy, vah a bias dané vrstvy neuronů upravenou aktivační funkcí **g**.

- $a^{(1)} = x$
- $z^{(2)} = w^{(2)} a^{(1)} + b^{(2)}$
- $a^{(2)} = g(z^{(2)})$
- $z^{(3)} = w^{(3)} a^{(2)} + b^{(3)}$
- $a^{(3)} = g(z^{(3)})$
- $z^{(4)} = w^{(4)} a^{(3)}$
- $a^{(4)} = g(z^{(4)})$ = výsledek

Protože v neuronové síti máme jeden výstupní neuron a ten leží ve čtvrté vrstvě, tak výslednou hodnotu značíme $a_1^{(4)}$.



Obrázek 7 - dopředná fáze

Tento výsledek porovnáme se správným výsledkem známým z trénovací množiny. Použijeme nákladovou funkci, v našem případě MSE, pro výpočet nákladů. Získáme tedy náklady $J_1^{(4)}$. Jedná se o chybu prvního neuronu ve čtvrté vrstvě.

$$J_1^{(4)} = (a_1^{(4)} - y_1)^2$$

$$z^k = w^k a^{k-1} + b^k$$

To můžeme rozepsat následovně:

$$J_1^{(4)} = J(a^4, y) = J(g(z^4, y) = J(g(w^4\, a^3 + b^4), y) = J(a^4(z^4(w^4\, a^3 + b^4)), y)$$

Při výpočtu distribuce chyby potřebujeme tedy spočítat parciální derivace $\frac{\partial J}{\partial w^4}$, $\frac{\partial J}{\partial b^4}$, ale i $\frac{\partial J}{\partial a^3}$. Zjišťujeme tedy citlivost nákladové funkce na změnu váhy, biasu a předchozí hodnoty aktivace. Budeme tedy derivovat složenou funkci. Pro zjištění musíme zjistit dílčí parciální derivace.

$$\frac{\partial J}{\partial w^4} = \frac{\partial J}{\partial a^4} \cdot \frac{\partial a^4}{\partial z^4} \cdot \frac{\partial z^4}{\partial w^4}$$

Vypočítáme parciální derivaci $\frac{\partial z^{\mathrm{k}}}{\partial w^k}$ obecně pro k vrstvu. V našem případě by se k = 4.

$$\frac{\partial z^{\mathrm{k}}}{\partial w^k} = a^{k-1}$$

Pro výpočet parciální derivace $\frac{\partial a^{\mathrm{k}}}{\partial z^k}$, budeme derivovat aktivační funkci. V našem případě se jedná o funkci sigmoid. Sigmoid se často používá, protože se lépe derivuje a výsledná parciální derivace je jednoduchá.

$$\frac{\partial a^{\mathrm{k}}}{\partial z^k} = g'(z^k) = g(z^k)(1 - g(z^k))$$

Parciální derivace $\frac{\partial J}{\partial a^{\mathrm{k}}}$

$$\frac{\partial J}{\partial a^{\mathrm{k}}} = 2(a^k - y)$$

Dále potřebujeme parciální derivaci $\frac{\partial J}{\partial b^{\mathrm{k}}}$

$$\frac{\partial J}{\partial b^{\mathrm{k}}} = \frac{\partial z^k \partial a^k \partial J}{\partial b^k \partial z^k \partial a^k} = g'(z^k)\, 2(a^k - y)$$

A nakonec potřebujeme parciální derivaci $\frac{\partial J}{\partial a^{\mathrm{k-1}}}$

$$\frac{\partial J}{\partial a^{\mathrm{k-1}}} = \frac{\partial z^k \partial a^k \partial J}{\partial a^{k-1} \partial z^k \partial a^k} = w^k g'(z^k)\, 2(a^k - y)$$

Toto jsou nezbytné výpočty pro k-tou vrstvu. Stejně budeme postupovat s vrstvami k – 1, k – 2 až se dostaneme k první vrstvě. Uvedené vzorce platí pro neuronovou síť, která má pouze jeden výstupní parametr.

Výše uvedené vzorce můžeme zobecnit pro libovolnou architekturu sítě. Index g bude značit g-tý neuron k-té vrstvy. Index h značí h-tý neuron (k-1) vrstvy. σ značí aktivační funkci sigmoid.

$$J = \sum_{i=1}^{n_{k-1}} (a_i^k - y_i)$$

$$z_g^k = \sum_{i=1}^{n_{k-1}} (w_{gi}^k \; a_i^{k-1} + b_i^k)$$

$$a_g^k = \sigma(z_g^k)$$

$$\frac{\partial z_g^k}{\partial w_{gh}^k} = a_h^{k-1}$$

$$\frac{\partial a_g^k}{\partial z_g^k} = \sigma'(z_g^k) = \sigma(z_g^k)(1 - \sigma(z_g^k))$$

$$\frac{\partial J}{\partial a_g^k} = 2(a_g^k - y_{g)}$$

$$\frac{\partial z_g^k}{\partial b_g^k} = 1$$

$$\frac{\partial z_g^k}{\partial a^{\mathrm{k-1}}} = w_{gh}^k$$

V praxi back propagation zpravidla nebudeme muset implementovat, protože je dostupný ve všech AI frameworcích. Například v pytorch máte přístup k vypočteným gradientům.

2 Klasifikace

Umělé neuronové sítě lze používat i pro úlohu strojového učení, na které jsme používali klasické algoritmy. Pro úlohy na predikci hodnoty nebo hodnot jsme používali různé formy regresních analýz. Výsledné hodnoty si u takové neuronové sítě přečteme z aktivačních neuronů poslední výstupní vrstvy.

Nyní se zaměříme na další standardní úlohu strojového učení a tou je klasifikace. Tedy úloha, kdy vstupní data jsou zařazena do jedné z předdefinovaných skupin.

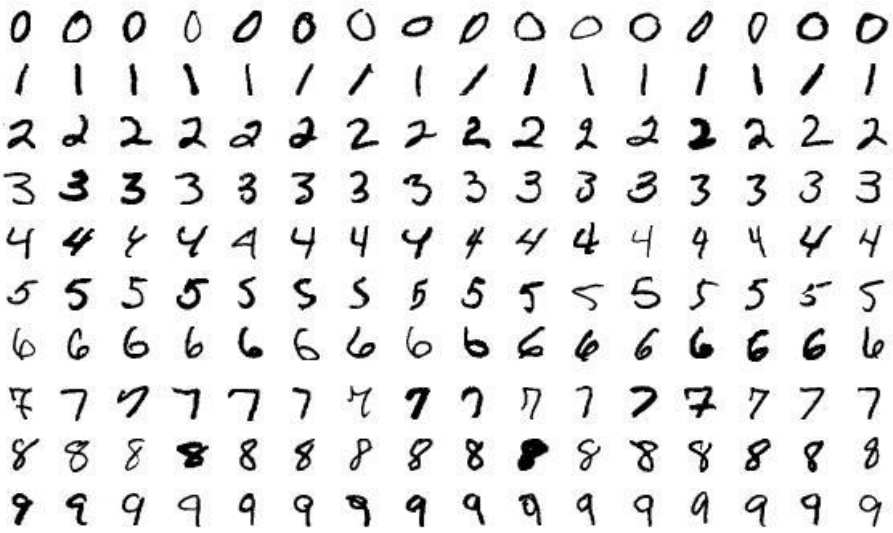
2.1 Příklad MNIST

Tento typ úlohy si přiblížíme na velmi známém problému rozlišování ručně psaných číslic. Databáze MNIST (Modified National Institute of Standards and Technology database) je rozsáhlá sbírka ručně psaných číslic. Obsahuje trénovací sadu 60 000 příkladů a testovací sadu 10 000 příkladů. Jedná se o podmnožinu větší speciální databáze NIST 3, kterou tvoří číslice psané zaměstnanci Úřadu pro sčítání lidu Spojených států amerických a podmnožinu speciální databáze 1, která obsahuje číslice psané studenty středních škol. MNIST je považován za "Hello World" strojového učení pro počítačové vidění a je často prvním krokem pro mnoho lidí, kteří se učí o strojovém učení a počítačovém vidění.

Trénovací a testovací podmnožiny obsahují obrazy číslic od odlišných lidí. Tím je zaručeno, aby se model netestoval na rukopisu konkrétních jedinců, na kterých byl trénován.

Databáze obsahuje černobílé obrazy ručně psaných číslic. Číslice byly normalizovány podle velikosti a vycentrovány do obrazu s pevnou velikostí. Původní černobílé (dvouúrovňové) obrazy z NIST databáze byly velikostně normalizovány tak, aby se vešly do rámečku 20x20 pixelů při zachování poměru stran. Obrazy číslic v databázi MNIST mají velikost 28x28 pixelů ve 256 úrovních šedi. Ty obrazy vznikly použitím algoritmu vyhlazování na původní černobílé obrazy 20x20.

Lze si povšimnout, že například číslice 2, 4, 7 lze napsat různými způsoby.



Obrázek 8 - vzorek ručně psaných číslic

Tento typ úloh lze řešit i klasickými metodami. Například bychom mohli analyzovat tvary jednotlivých číslic. Můžeme napsat klasický program, který bude pomocí detekcí hran rozpoznávat základní tvary jako je čára, kružnice, oblouk. Dokážeme zjistit pozice konců čáry, její úhel. U kružnice či elipsy dokážeme určit střed, osy, natočení atd.

Můžeme vytvořit systém pravidel, který podle vzájemných poloh těchto primitivních tvarů dokáže rozpoznat číslici. Nula bude mít tvar vysoké elipsy, osmičky se bude skládat ze dvou kružnic, které se budou dotýkat, jednička bude svislá čára dolů, sedmička bude víceméně svislá čára dolů s horní horizontální navazující kratší čárou.

Při detailnějším průzkumu dat zjistíme, že nula nemusí být vždy uzavřená, i jednička může být nahoře horní kratší čáru, která směřuje pod určitým úhlem dolů, osmička se může skládat ze dvou kružnic, které se nedotýkají. Pravidla se rozrostou o výjimky, podpravidla atd. Ve výsledku takový systém bude velmi nepřehledný a složitý na údržbu.

Zrovna v takovýchto případech se může neuronová síť ukázat jako dobrý nástroj. I lidský mozek nefunguje na přesně definovaných pravidlech, ale podle přibližného tvaru dokážeme přiřadit číslici do správné kategorie.

2.2 Klasifikační neuronová síť

Nyní si zkusíme vytvořit první verzi klasifikační neuronové sítě pro rozpoznávání ručně psaných znaků.

Vstupní vrstva je definovaná vstupními hodnotami. V našem případě se jedná o matici o rozměrech 28 x 28 hodnot v rozsahu 0 až 255. Při vytváření neuronové sítě můžeme zanedbat vstupní tvar matice, která nám vlastně říká, které hodnoty jsou si prostorově blízké. Při učení si neuronová síť tyto vztahy sama zjistí a přizpůsobí jim váhy.

První vrstva se tedy bude skládat z 28^2 neuronů, tedy z 784 neuronů. Pokud by číslice byly například barevné a barva by se skládala z 3 hodnot (RGB), tak by vstupní vrstva měla $784 \cdot 3 = 2352$ neuronů.

Pro rychlejší učení umělé neuronové sítě je vhodné vstupní data standardizovat a převést je z hodnot 0 až 255 na hodnoty 0 až 1.

Výstupní vrstva je definovaná požadovaným výstupem. V našem případě chceme, aby byla vstupní matice 28 x 28 přiřazena jedné číslici, tedy jedné z 10 kategorií.

Při volbě počtu výstupních neuronů máme několik možností. Možná první nejvíce intuitivní možnost, je volba jednoho neuronu, který bude vracet hodnoty z omezeného oboru hodnot <0, 1>. Jednotlivým kategoriím přiřadíme rozpětí výstupní hodnoty. Například <0; 0,1> bude kategorie 1, (0,1; 0,2> bude kategorie 2 atd.

Pokud budeme jako aktivační funkci používat funkci sigmoid, tak zjistíme, že pravděpodobnostní rozdělení jednotlivých kategorií v závislosti na vnitřní hodnotě neuronu nebude proporční. Neuronová síť nebude fungovat zcela správně.

Obrázek 9 - aktivační funkce sigmoid

Mohli bychom použít aktivační funkci ReLU, respektive její variantu ReLU, která vrací maximální hodnotu 10, ale obecně toto se u klasifikačních modelů nepoužívá, protože přechod mezi kategoriemi je příliš ostrý.

Jiným způsobem může být zakódování pomocí binárního kódu. Pro uložení 10 kategorií budeme potřebovat 4 neurony, které budou nabývat hodnot 0 a 1. V případě spojitých výstupních hodnot, určíme prahovou hodnotu, kdy se jedná o 0 nebo 1. Výstupní kategorii získáme přečtením hodnot 4 neuronů a převedením z binárního kódu. U tohoto řešení nám vzroste počet výstupních neuronů a návratové hodnoty budou relativně nepřehledné.

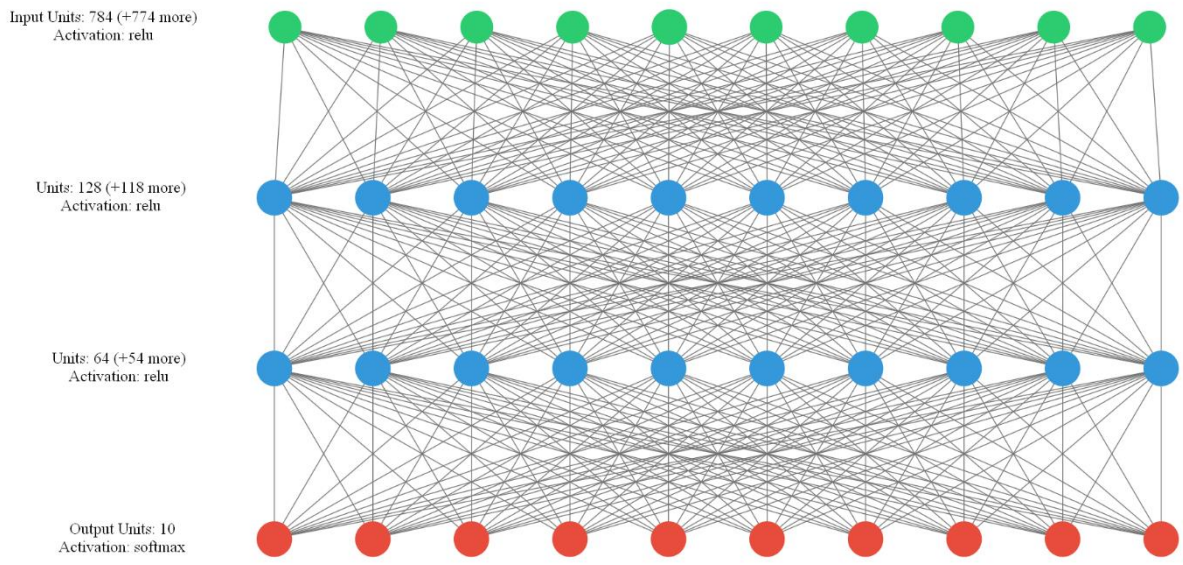
V případě klasifikací se velmi často používá stejný počet výstupních neuronů jako je počet možných kategorií. Při správně volbě aktivačních funkcí, budou výstupní neurony obsahovat hodnotu z množiny <0, 1>, která určuje pravděpodobnost příslušnosti vstupních dat do dané kategorie. U tohoto způsobu nám sice vzrostl počet výstupních neuronů, ale výstup dokáže velmi pěkně reprezentovat nejistotu při rozpoznávání.

Například pro dvojku napsanou následujícím způsobem, můžeme získat vektor pravděpodobností příslušnosti tvaru do třídy. S takovýmto výstupem neuronová síť dokáže vyjádřit svoji nejistotu, kdy ze 75% se domnívá, že se jedná o 2 a z 24% se domnívá, že se jedná o 3, která je třeba pootočená a oříznutá.

0	0,000001
1	0,00000000001
2	0,75
3	0,24
4	0,0005
5	0,0006
6	0,0006
7	0,00000001
8	0,00045
9	0,000098

Ve chvíli, kdy máme takto definované vstupní a výstupní vrstvy, můžeme do modelu neuronové sítě vkládat skryté vrstvy. V této kapitole zůstaneme u plně propojených vrstev Dense, u kterých použijeme jako aktivační funkci ReLU. V další kapitole si ukážeme pokročilejší konvoluční neuronové sítě, které se používají především právě pro počítačové vidění. Protože je MNIST velmi jednoduchý dataset, dokážeme i s plně propojenými vrstvami vytvořit velmi přesný model.

Naše síť se tedy bude skládat ze vstupní vrstvy s 784 neurony, první skryté vrstvy se 128 neurony, druhé skryté vrstvy s 64 neurony a 10 neurony ve výstupní vrstvě.

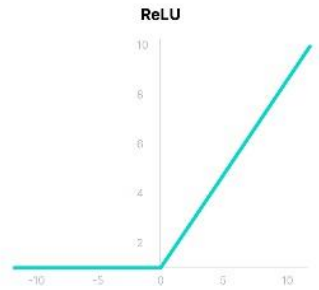


Obrázek 10 - model neuronové sítě

2.3 Aktivační funkce

Ve všech vrstvách až na výstupní budeme používat aktivační funkci ReLU (Rectified linear unit). Jak je vidět z obrázku, funkce vrací kladné vstupní hodnoty beze změny. Záporné vstupní hodnoty mění na 0. Pro svoji jednoduchost je velmi oblíbená.

$$f(x) = \max(0, x)$$



Obrázek 11 - aktivační funkce ReLU

U výstupní vrstvy použijeme místo funkce sigmoid funkci softmax. Funkce softmax, známá také jako softargmax nebo normalizovaná exponenciální funkce, převádí vektor K reálných čísel na pravděpodobnostní rozdělení K možných výsledků. Je zobecněním logistické funkce na více dimenzí a používá se v multinomiální logistické regresi. Proto se často používá jako poslední aktivační funkce neuronové sítě k normalizaci výstupu sítě na pravděpodobnostní rozdělení nad předpovídanými třídami výstupů. Každé číslo výstupního vektoru bude z intervalu (0, 1).

Do rovnice funkce pro x_i, vstupuje vektor x, který se skládá z K reálných čísel x_i. Rovnice vypadá následovně:

$$f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

2.4 Nákladová funkce

Při strojovém učení je vždy třeba zvolit správnou nákladovou funkci, která nejlépe ohodnocuje špatnou klasifikaci vstupních hodnot vůči správným odpovědím. Při regresních úlohách jsme často používali nákladovou funkci MSE nebo MAE. Ta se odkazuje na rozdíl mezi správnou a predikovanou hodnotou.

Pro klasifikační úlohy tyto funkce nejsou vhodné. Například mějme obrázek číslice 1 a neuronová síť ho vyhodnotí jako 7. Pak MSE vrátí pro tento případ hodnotu $(7 - 1)^2 = 36$. Pokud by neuronová síť vrátila predikovanou číslici 2, pak MSE vrátí hodnotu $(2 - 1)^2 = 1$. Přičemž ani v jednom případě neuronová síť nezařadila vstupní data do správné kategorie.

Proto se pro klasifikační úlohy často používá nákladová funkce categorical_crossentropy. Kategoriální křížová entropie je vhodná i pro úlohy, kdy výstupní vektor obsahuje více než dvě proměnné. Měří rozdíl mezi skutečným rozdělením pravděpodobnosti tříd a předpovídaným rozdělením pravděpodobnosti.

Entropie vyjadřuje míru náhodnosti nebo neuspořádanosti v systému. V kontextu teorie informace představuje entropie náhodné veličiny průměrnou nejistotu, překvapení nebo informaci, která je vlastní možným výsledkům. Zjednodušeně řečeno, měří neurčitost události. Například při hodu mincí bychom měli dva možné výsledky, přičemž pravděpodobnost, že mince padne na panna nebo orel, je ½. To znamená: P(X=panna) = P(X=orel) = ½.

Podle Shannonovy rovnice entropie by se entropie obou členů mince rovnala 0, což znamená, že neexistuje téměř žádná nejistota – buď bude téměř vždy padat panna nebo téměř vždy orel.

Čím větší je hodnota entropie H(x), tím větší je nejistota pro dané rozdělení pravděpodobnosti, a čím menší je tato hodnota, tím menší je nejistota.

$$H = -\sum p(x)\ \log\ p(x)$$

Nyní se můžeme pustit do křížové entropie. Křížová entropie, známá také jako logaritmická ztráta, je populární ztrátová funkce používaná ve strojovém učení k měření výkonnosti klasifikačního modelu.

Funkce měří průměrný počet bitů potřebných k identifikaci události z jednoho rozdělení pravděpodobnosti P, pomocí optimálního kódu pro jiné rozdělení pravděpodobnosti Q. Jinými slovy, křížová entropie měří rozdíl mezi zjištěným rozdělením pravděpodobnosti klasifikačního modelu a předpovězenými hodnotami.

Binární cross entropy rovnice vypadá následovně, kde N je počet trénovacích dat, t_i je hodnota 0 nebo 1 pro přiřazení vstupní hodnoty do třídy a p_i je pravděpodobnost tohoto přiřazení.

$$L = -\frac{1}{N}\left[\sum_{i=1}^{N}\left[t_j \log(p_i) + (1 - t_i)\ \log(1 - p_i)\right]\right]$$

Při použití křížové entropie na klasifikační úlohu více tříd se ztráta pro každou třídu vypočítá zvlášť a poté se sečte, aby se určila celková ztráta.

Rovnice vypadá následovně, kde y je jednorázově zakódovaný vektor představující skutečné značky tříd, y_pred je předpovězené rozdělení pravděpodobnosti pro jednotlivé třídy. Součet se bere pro všechny třídy. Y_pred je zpravidla vypočítán pomocí softmax aktivační funkce.

$$f(x) = -\sum_{i=1}^{K} y \cdot \log(y_{pred})$$

2.5 Metriky

U klasifikačního modelu mohou nastat dvě správné predikce a dvě nesprávné predikce.

- **Chyba typu I** (sýčkování) – falešně pozitivní (pacient je zdravý, ale diagnostikujeme chorobu)
- **Chyba typu II** (důvěřivost) – falešně negativní (pacient je nemocný, ale určíme, že je zdravý).

Při hodnocení modelu velmi záleží na situaci, který typ chyb nám vadí a který ne, případně jak moc. V některých případech klademe důraz, aby byly správně klasifikovány pozitivní případy a pokud nastane situace falešně pozitivní, tak to nemusí být tak velký problém (hledání zmetků). Existuje více metrik, které měří přesnost modelu a zaměřují se na různé typy chyb.

		Skutečnost	
		Pozitivní	Negativní
Předpověď	Pozitivní	True positive pacient je nemocný a tvrdíme, že je nemocný	False positive pacient je zdravý, ale tvrdíme, že je nemocný
	Negativní	False negative pacient je nemocný, ale tvrdíme, že je zdravý	True negative pacient je zdravý a tvrdíme, že je zdravý

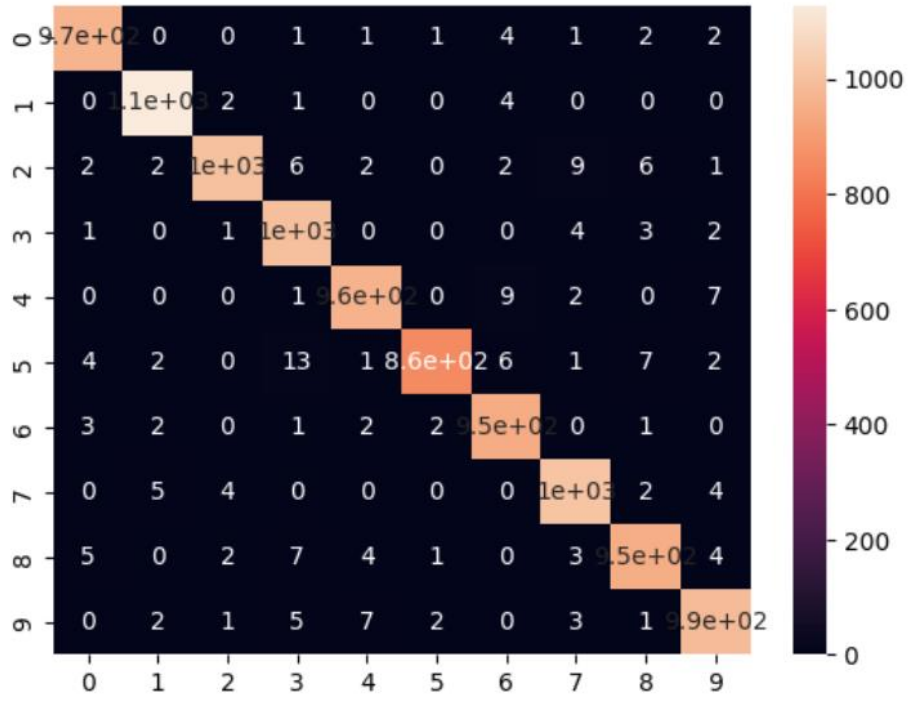
Pro hodnocení klasifikačních modelů se používají různé metriky, jak bylo zmíněno v kapitole 5.

- **správnost** $(accuracy) = \frac{TP+TN}{TP+TN+FP+FN} = \frac{počet\ správných\ odpovědí}{počet\ všech\ odpovědí}$
- **specificita** $(specifity, selectivity) = \frac{TP}{TP+FN} = správně\ diagnostikovaní\ nemocní$
- **citlivost, sensitivita** $(sensitivity, recall) = \frac{TN}{TN+FP} = správně\ diagnostikovaní\ zdraví$
- **přesnost** $(precision) = \frac{TP}{TP+FP} = míra\ přesnosti, kdy\ byl\ diagnostikován\ jako\ pozitivní$
- **F1** $score = 2 \cdot \frac{precision \cdot recall}{precission+recall} = harmonický\ průměr\ přesnosti\ a\ specificity$

2.6 Matice změn

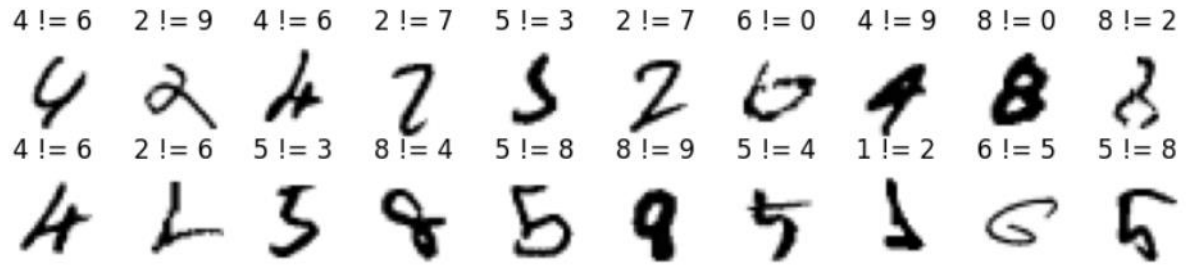
Pokud klasifikační model používá více než dvě kategorie, je vhodné jeho účinnost ověřit i pomocí matice změn.

Horizontální řádky jsou předpovědi modelu, vertikální sloupce reprezentují skutečnost.



Obrázek 12 - matice změn pro MNIST

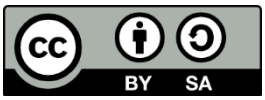
Z této tabulky lze například vypozorovat, že 0 se plete s 5, 6 a 8. 1 se zaměňuje často se 7 atd. Při ladění neuronové sítě může být podstatné se podívat i na konkrétní případy záměn, abychom mohli lidsky posoudit odůvodněnost těchto chyb. V některých případech by v rozpoznávání některých číslic selhal i lidský jedinec.



Obrázek 13 - Příklady záměn

UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vytvořeno v rámci projektu **Digitalizace studijních Agend, Nové TechnologiE, systémy a přístupy k výuce na UPCE**, reg. č. NPO_UPCE_MSMT-16591/2022.



Financováno Evropskou unií
NextGenerationEU

Národní plán obnovy

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Machine learning & AI

Konvoluční neuronové sítě

Ing. Martin Pozdílek, Ph.D.

1 Počítačové vidění

Počítačové vidění je interdisciplinární obor zahrnující techniky a metody pro zpracování a interpretaci obrazů a videí pomocí počítačů. Jeho cílem je umožnit počítačům „vidět“ a porozumět okolnímu světu prostřednictvím digitálních obrazů získaných z různých zdrojů, jako jsou kamery, MRI skenery, nebo mikroskopy. Tento obor nachází aplikace v široké škále odvětví, včetně lékařství, robotiky, bezpečnosti, průmyslu, autonomních vozidel, rozpoznávání obličejů a mnoha dalších. [1]

Existuje několik typických úloh v počítačovém vidění, jako je detekce hran, klasifikace, segmentace obrazu, rozpoznávání vzorů atd. [2] [3] Tyto úlohy lze řešit klasickými algoritmy, které jsou například implementovány v knihovně openCV. [4] Moderní metody často využívají strojové učení a hluboké neuronové sítě, které umožňují automatické učení se z dat a dosahují vysoké přesnosti. V této kapitole se podíváme, jak řešit pomocí neuronových sítí, některé typické úlohy z počítačového vidění.

2 Klasifikace obrazu

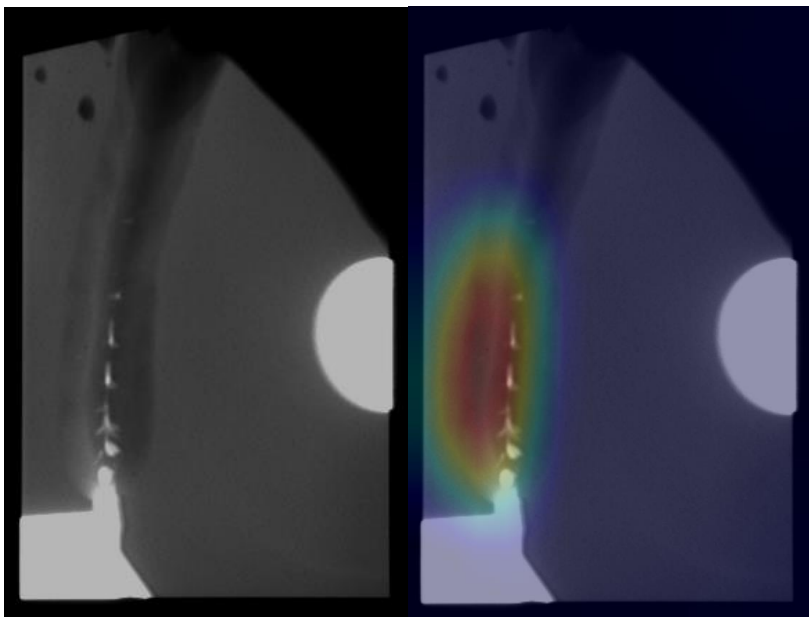
Klasifikace v počítačovém vidění je proces přiřazení kategorií obrázkům nebo videím. Úlohu klasifikace jsme si již představili v kapitole 5, kde jsme si představili klasické metody jako k-mean [5], Support Vector Machines (SVM) [6].

V předchozí kapitole jsme si ukázali, jak řešit problém pomocí hlubokých neuronových sítí. Tyto modely jsou schopny dosáhnout vyšší přesnosti a generalizace než tradiční metody klasifikace, zejména pokud jsou dostatečně trénovány na relevantních datech. Pro rozpoznávání například ručně psaných číslic jsme použili plně propojenou neuronovou síť. Tato architektura neuronové sítě není optimální pro zpracování hlavně rozměrnějších obrazových dat. Hlavní nevýhodou je velikost jednotlivých vrstev a ignorování prostorového rozložení dat. Plně propojené neuronové sítě při zkoumání závislostí pixelů v principu nerozlišují, zda se jedná o dva sousední pixely nebo pixely z opačných rohů obrázku. V této kapitole se podíváme na konvoluční neuronové sítě (Convolutional Neural Networks – CNN), které s prostorovým rozložením dat počítají a jsou tak efektivnější při učení i při fungování v provozu. [7]

Typickou úlohou je identifikace druhů, chorob, závad atd. Například se může jednat o rozpoznávání chorob rostlin.

Obrázek 1 - choroby rostlin

V níže uvedeném případě byl RTG snímek výrobku určen do třídy Vada. Standardně nelze určit, podle čeho se neuronová síť rozhodla o zařazení snímku do této kategorie. V průběhu učení lze zpětně sledovat, které pixely obrázku měly největší vliv na zařazení vstupního obrázku do dané třídy. To je naznačeno překrytím obrázku heatmapou. [8] To nám umožní ručně ověřit, že natrénovaný model se opravdu rozhoduje podle kritických částí obrázku. Nicméně při běžném provozu neuronové sítě tato informace zpravidla není již k dispozici.



Obrázek 2 - klasifikace vady

Jak člověk zpracovává obraz? Lidský mozek se zaměří na klíčové vlastnosti. [3] Mnoho detailů pro zpracování obrazu vůbec nepoužívá. U fotky domu budeme identifikovat obdélníková okna, dveře, trojúhelníková střecha, obdélníková stěna. Barva pro nás není důležitá, okolí není důležité, tvary mohou být přibližné atd. Naopak pro správnou a rychlou klasifikaci obrazu je pro nás důležitá přítomnost klíčových vlastností a jejich vzájemná pozice. U jiných obrázků naopak může být pro rozpoznání důležitá barva a tvar a vzájemná poloha již nemusí být tak důležitá. Naše neuronová síť se naučila rozpoznávat předměty, které nás obklopují podle jejich klíčových vlastností.

Obrázek 3 - Klíčové vlastnosti obrazu

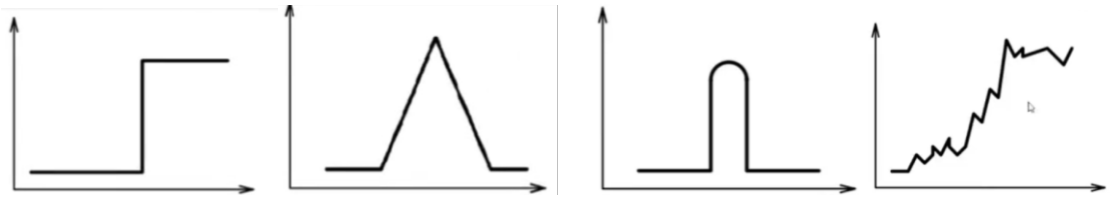
2.1 Detekce hran

Jedním z nástrojů pro rozpoznání klíčových vlastností je detekce hran. Detekce hran slouží k identifikaci přechodů mezi různými oblastmi v obraze, kde se intenzita pixelů rychle mění. Hrany jsou často spojeny s různými objekty nebo změnami ve scéně a mají klíčový význam pro analýzu obrazů. Detekce hran nám umožňuje ignorovat velké množství pixelů a soustředit se pouze na klíčové části obrazu. [8]

Prvním krokem při detekci hran bývá zpravidla odbarvení obrázku, tedy vyjádření všech pixelů v odstínech šedi. Obrázek tedy vyjádříme jako funkci jasové funkce.

Obrázek 4 - jasová funkce

Hranu pak můžeme detekovat jako výraznou změnu jasu sousedních pixelů. Pokud budeme měnit souřadnici pixelu v horizontálním nebo vertikálním směru, tak se hodnota jasové funkce na ose Y bude výrazně měnit. V prvním případě se jedná o výrazný přechod tmavé plochy na světlou plochu. V druhém a třetím případě se jedná o výrazně světlou hranu ve tmavé ploše. Vyrovnané křivky naznačují, že hrana je barevně plynule zesvětlována a ztmavována. První tři grafy odpovídají ideálním případům. Ve skutečnosti získáme čtvrtý graf.



Obrázek 5 - přechody jasové funkce

Existuje více algoritmů pro detekci hran – Difference of Gaussians (DOG), Laplace operator, Image Gradient, Sobel operator, Canny detector a mnoho dalších. Každý z nich se jiným principem z jasové funkce snaží detekovat hranu. [9] [8]

Obecně, aby šly v obrázku detekovat hrany, je třeba, aby obsahovat dostatečně homogenní plochy. Dále musí být mezi oblastmi dostatečný odstup jasů a přechod musí být relativně krátký. Například na obrázku modré oblohy nebo obrázku šumu hrany detekovat nepůjde.

Obrázek 6 - původní a černobílý obrázek, detekce hran

Často se používají gradientní algoritmy. Ve kterých se vypočítává velikost gradientu pro jasovou funkci f.

$$\frac{df}{dx} = f(x+1, y) - f(x, y)$$

$$\frac{df}{dy} = f(x, y+1) - f(x, y)$$

Lze i zjistit směr gradientu

$$\alpha(x, y) = tan^{-1}\left(\frac{\frac{df}{dx}}{\frac{df}{dy}}\right)$$

Pro určení, zda daný gradient odpovídá hraně, je třeba určit prahovou hodnotu. Mnohdy je třeba tento práh určit ručně na základě nějaké expertízy. To může být nevýhoda při automatizaci procesu.

Obrázek 7 - algoritmy Difference of Gaussians (DOG), Laplace operator, Image Gradient

Naopak někdy může být prospěšné pracovat s oddělenými kanály modré, červené a zelené barvy. To nám může pomoci například při detekci růžových bobulí, které jsou viditelné v modrém a červeném kanálu a zmizí v zeleném kanálu. Při aplikaci detektoru hran na jednotlivé kanály, se některé důležité hrany mohou zvýraznit.

Obrázek 8 - Detekce hran v barevných kanálech

Některé algoritmy pro detekci hran používají složitější maticové operace, jedná se například o Robertsovy operátory, operátory Prewittové nebo Sobelovy operátory. Tyto operátory pro detekci hran používají maticovou masku, zpravidla o rozměrech 3 x 3, která dává určitou váhu sousedním pixelům. Pro pozdější pochopení konvolučních sítí si nastíníme fungování Sobelova operátoru. Nejčastěji se používají dvě masky pro detekci horizontálních vertikálních hran. [9]

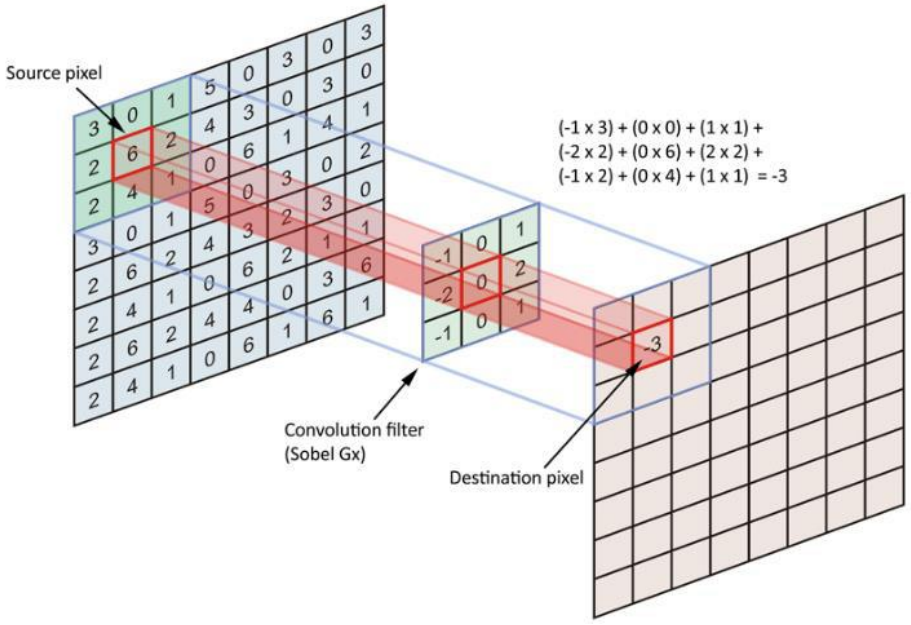
$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Pro každý pixel obrázku se provede skalární součin Sobelova operátoru s pixelem a jeho okolními pixely, respektive s jejich jasovou funkcí. V případě prvního vertikálního operátoru je jasová funkce pixelů nalevo ponechána, respektive zdvojnásobena. Zcela je ignorována jasová funkce pixelů ve středním sloupečku a jasová funkce pixelů napravo je invertována. Výsledkem je číslo, které reprezentuje rozdíly jasových funkcí mezi levým a pravým sloupcem pixelů.

Druhá matice je pootočena a slouží k detekci horizontálních hran.

Postup výpočtu je naznačen na následujícím obrázku. Tato operace se nazývá konvolucí. [10]



Obrázek 9 - výpočet Sobelova operátoru

Když se podíváme na výsledek detekce hran pro obě masky, je patrné, že jsou zvýrazněny vertikální a na druhém obrázku horizontální hrany.

Obrázek 10 - algoritmy Sobel operátor s vertikální a horizontální maskou

U algoritmů detekce hran je třeba si uvědomit, že nám vrací pouze hrany, respektive přechody, ani neposkytují informace, o jaké objekty, respektive vlastnosti obrazu, se jedná.

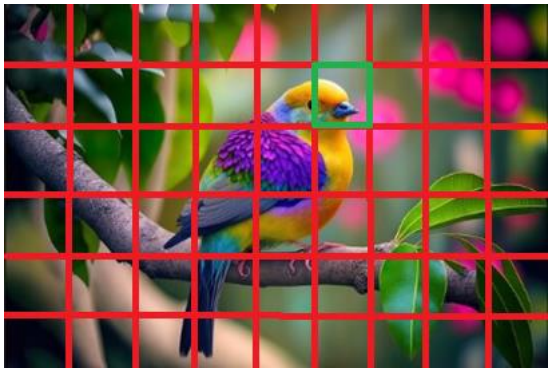
2.2 Extrakce vlastností pomocí konvoluce

Biologická neuronová síť se při klasifikaci opírá o klíčové vlastnosti. Cílem extrakce vlastností je tedy snížit dimenzi dat a zachovat pouze relevantní informace, což usnadňuje následné analýzy a rozhodovací procesy.

Zkusme tedy navrhnout neuronovou síť, která se bude chovat při rozpoznávání obrazu podobně jako biologický mozek. Zůstaňme u obrázku ptáka a zkusme vytvořit umělou neuronovou síť, která dokáže rozhodnout, zda je či není na obrázku pták. Člověk při rozpoznávání ptáků zpravidla detekuje přítomnost klíčových vlastností jako zobák, ptačí noha, křídlo atd. Jsou to klíčové vlastnosti,

které se mohou nacházet na různých místech obrazu. V datech se obecně nacházejí vzory (vlastnosti) definující dané objekty. Tyto vzory mohou mít různé velikosti a různé umístění v datech. Pro klasifikaci dat potřebujeme navrhnout skupiny detektorů, které procházejí zpracovávaná data a v nich identifikují klíčové vlastnosti. Jednotlivé části dat mohou být kódovány stejným detektorem. Tyto vzory mohou mít podobu hran, tvarů, barev atd. [11]

V našem případě neuronové sítě pro rozpoznávání ptáků zkusíme vyvinout detektor zobáků, kterému budeme předkládat malé části obrazu, a detektor nám vrátí, zda na dané časti je zobák nebo není. Dále budeme vytvářet detektory na nohy, křídla, peří, …



Obrázek 11 - detektor zobáku

Algoritmus si ukážeme na jednodušším příkladu, ve kterém budeme mít binární data v matici 6 x 6 a budeme hledat šikmý a vertikální vzor složený z 1. Pro detekci šikmého vzoru jedniček zvolíme konvoluční filtr o rozměrech 3 x 3, který bude mít na hlavní diagonále 1, a zbytek buněk budeme mít hodnotu -1.

V prvním kroku provedeme skalární násobení červeně vyznačené části matice s konvolučním filtrem a výsledek uložíme do buňky matice příznaků.

$$1 \cdot 1 + 0 \cdot -1 + 0 \cdot -1 + 0 \cdot -1 + 1 \cdot 1 + 0 \cdot -1 + 0 \cdot -1 + 0 \cdot -1 + 1 \cdot 1 = 3$$

V druhém kroku provedeme posun vybrané oblasti o jedno doprava a opět provedeme skalární násobení červené oblasti s konvolučním filtrem a výsledek uložíme do druhé buňky matice příznaků.

$$0 \cdot 1 + 0 \cdot -1 + 0 \cdot -1 + 1 \cdot -1 + 0 \cdot 1 + 0 \cdot -1 + 0 \cdot -1 + 1 \cdot -1 + 1 \cdot 1 = -1$$

Postupně budeme posouvat oblast přes celou vstupní matici, až získáme matici příznaků pro první konvoluční filtr detekující šikmý vzor.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

1	-1	-1
-1	1	-1
-1	-1	1

3			

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

1	-1	-1
-1	1	-1
-1	-1	1

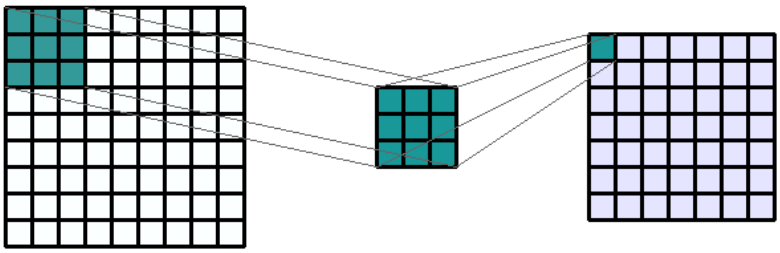
3	-1		

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

1	-1	-1
-1	1	-1
-1	-1	1

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Obrázek 12 - výpočet matice příznaků



Obrázek 13 - pohyb filtru po vstupních datech

Podobným způsobem můžeme navrhnout jiný konvoluční filtr například pro vzor vertikálních jedniček. Tímto filtrem provedeme výpočet další vrstvy matice příznaků.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

-1	1	-1
-1	1	-1
-1	1	-1

-1	-1	-1	-1
-1	-1	-2	1
-1	-1	-2	1
-1	0	-4	3

Obrázek 14 - výpočet jiné matice příznaků

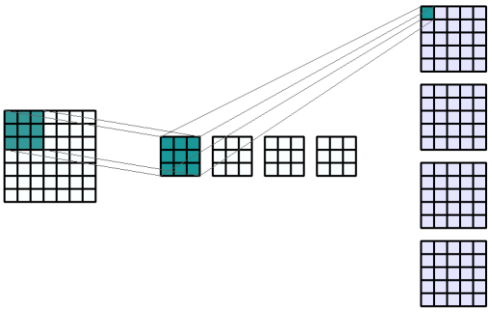
Matice příznaků určuje, kde se v původním obrázku vyskytují dominantní příznaky shodné s těmi v konvolučním filtru. Pro náš příklad se šikmý vzor \ vyskytuje v horním a spodním levém rohu, vertikální vzor se vyskytuje v pravém spodním rohu obrázku.

Aplikací všech konvolučních filtrů na vstupní data získáváme mapu příznaků, která se skládá z více matic příznaků.

2.3 Konvoluční neuronové sítě

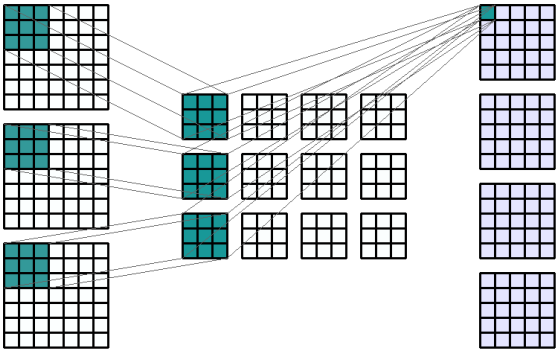
Konvoluční neuronová síť (CNN) je neuronová síť s dopřednou vazbou, která dokáže zpracovávat prostorová data. Běžně se používá pro aplikace počítačového vidění, jako je klasifikace obrazu. Jak již název napovídá, jedním typem vrstvy je konvoluční vrstva, která provádí výše vysvětlenou konvoluci. [12]

Konvoluční vrstva je tvořena definovaným počtem filtrů, které detekují vzory definované velikostí (velikost jádra), váhy filtrů jsou získány trénováním. Neuronová síť tedy používá určitý počet filtrů, který je definován při modelování neuronové vrstvy.



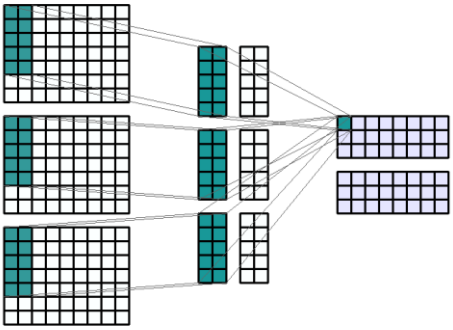
Obrázek 15 - počet různých filtrů

Pokud zpracováváme obrázek se třemi kanály, můžeme se rozhodnout, zda budou filtry pro všechny kanály stejné nebo odlišné.



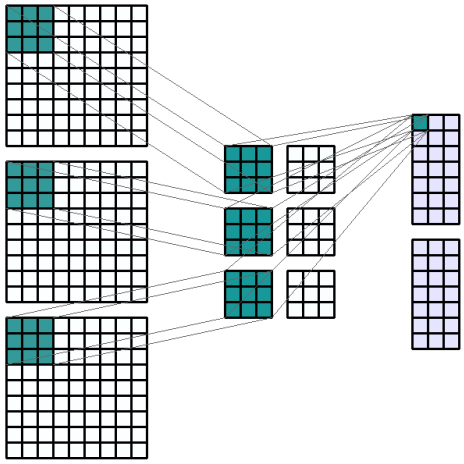
Obrázek 16 - počet kanálů vstupních dat

Velikost filtrů / jader může být libovolná. Například 3 x 3, 5 x 2 apod.



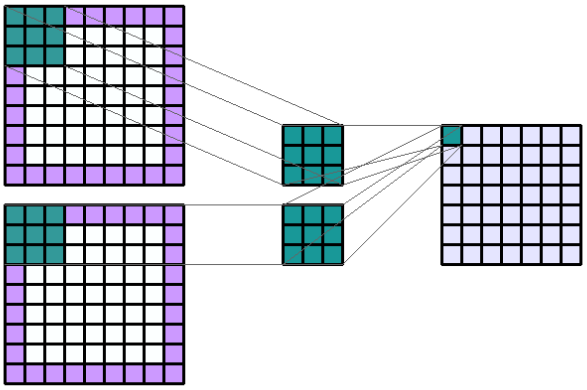
Obrázek 17 - velikosti filtrů

Standardně se filtr posunuje o jeden pixel horizontálně a vertikálně. Pomocí parametru stride můžeme definovat velikost posunu. Například pokud velikost filtru bude 3 x 3 a velikost stride nastavíme na hodnotu 3, pak se pozice filtru nebudou překrývat.



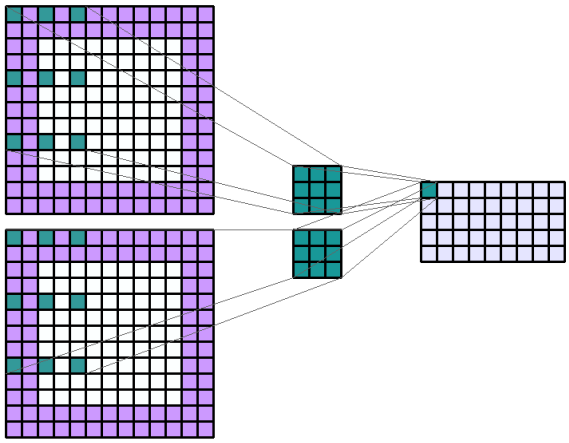
Obrázek 18 - posun filtru

Vstupní data nemusí být násobkem velikosti filtru. To se řeší pomocí parametru padding, který přidá vstupním datům potřebné sloupce a řádky vyplněné nulami, okolními hodnotami atp. Používá se hlavně, když je potřeba mít stejnou vstupní a výstupní matici



Obrázek 19 – padding

Dosud jsme předpokládali, že filtr je kompaktní. Lze ovšem filtr nadefinovat i tak, že bude obsahovat díry.

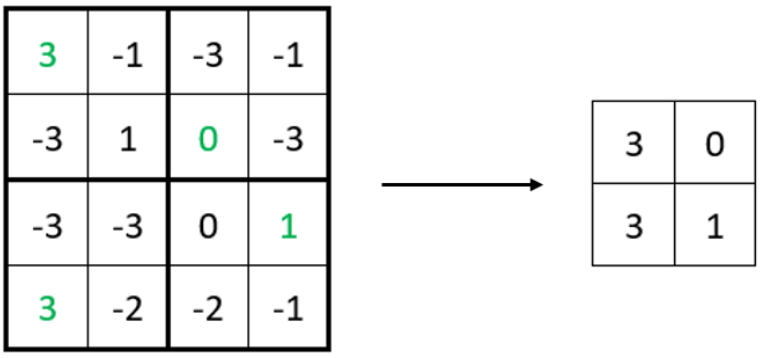


Obrázek 20 - dilation

Výstup z neuronů můžeme opět upravovat pomocí aktivační funkce, například ReLU.

Další typ vrstvy, který se v konvolučních sítích používá, je pooling. Tato vrstva se používá pro zmenšení velikosti matice, tedy redukce dimenzionalitu. Pooling vrstvy sjednocují skupinu vlastností a vybírají jen tu nejvíce dominantní. Vstupní data jsou rozdělena podle velikost filtru / kernelu. Na obrázku je znázorněno jádro o velikosti 2 x 2. Velmi často se používá maxpooling vrstva, která z hodnot vrací maximální hodnotu. [7][13]

Podobně jako u konvoluční vrstvy lze definovat, o kolik pixelů se má jádro posunout. V našem případě se posouvá o 2 pixely. Další parametry padding a dilation se chovají opět zcela stejně jako u konvoluční vrstvy.



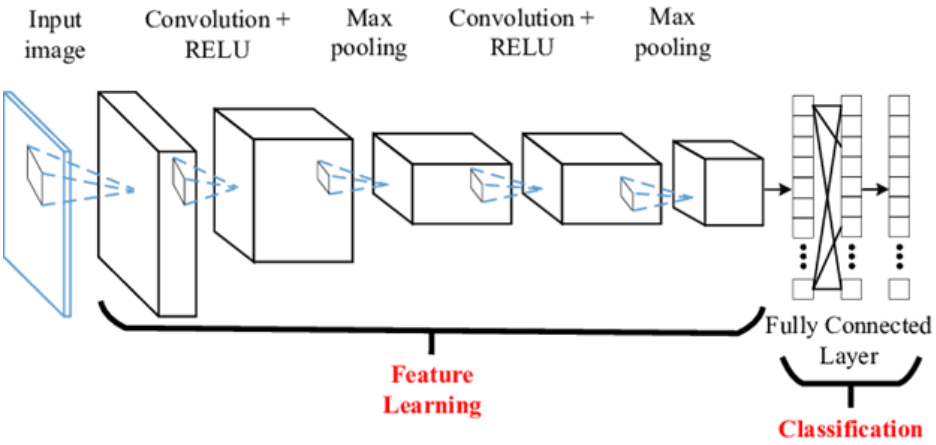
Obrázek 21 - Maxpooling

Posledním typem vrstvy, která se používá v konvolučních neuronových sítích, je nám známá plně propojená vrstva.

Architektury konvolučních neuronových sítí se tak skládají z několika konvolučních vrstev, za kterými následuje max pooling vrstva, která zmenší dimenzionalitu dat. Tato kombinace konvolučních vrstev a maxpooling vrstvy se může několikrát zopakovat. Jedná se o část neuronové sítě, která má za

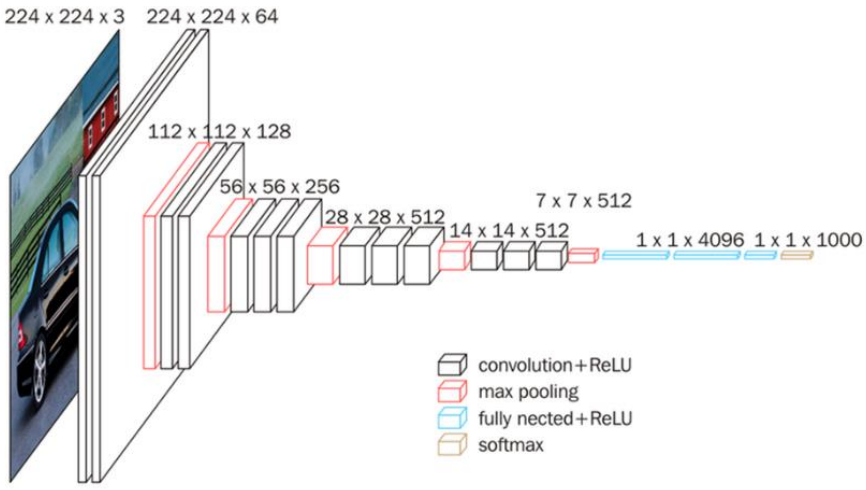
úkol naučit se a detekovat klíčové vlastnosti. V průběhu učení CNN dochází k ladění parametrů filtrů tak, aby si neuronová síť sama určila, které vlastnosti jsou pro ni důležité. Architekt neuronové sítě tedy nemusí dopředu vymýšlet specifické hodnoty konvolučních filtrů.

Za touto částí neuronové sítě se nachází část složen8 z plně propojených vrstev. Tato část je zodpovědná za klasifikaci. Podle nalezených klíčových vlastností identifikuje, do které třídy se má vstupní obrázek zařadit.



Obrázek 22 - typická architektura CNN

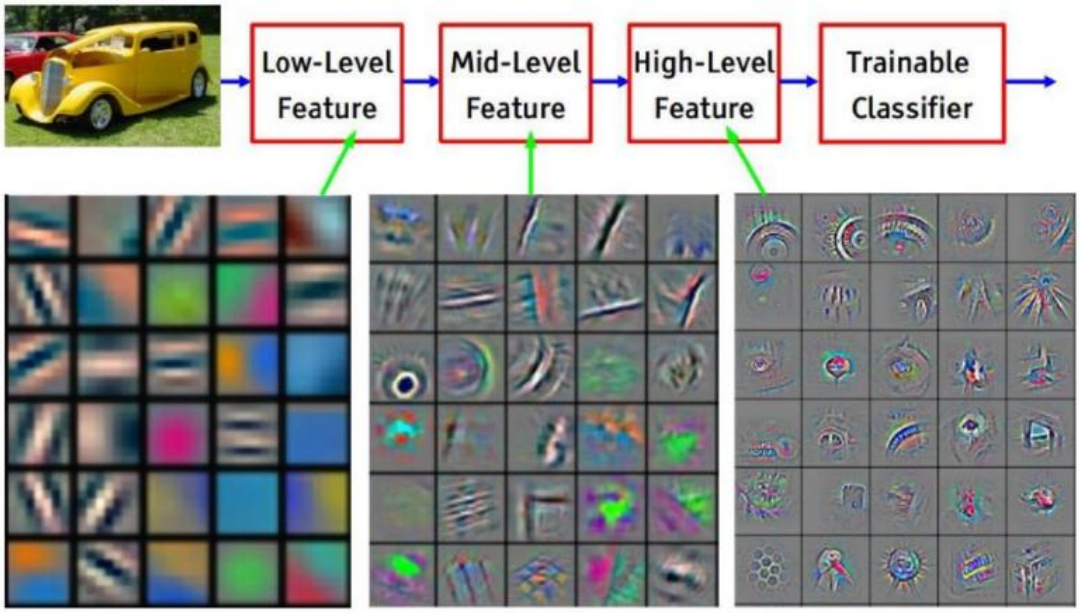
Samotných topologií CNN je velké množství a návrh samotné topologie je již značně komplexní. Běžnou inženýrskou praktikou je výběr z již existujících topologií vhodných pro řešení daného problému. Konkrétní neuronové sítě řeší velikost vstupních dat. Například obrázky 224 x 224, 300 x 300, 500 x 500 pixelů apod. Dále volí počet a chování konvolučních filtrů, chování maxpooling vrstev nebo chování klasifikační části neuronové sítě. Například architektura sítě VGG16 vypadá následovně. [14]



Obrázek 23 - VGG16

Důvodem proč se spojuje více konvolučních vrstev za sebou, je hledání více univerzálních klíčových vlastností. První vrstva získá klíčové vlastnosti, které lze interpretovat jako filtry, které hledají nějaké grafické vzory. [15]

Tyto nízkoúrovňové vlastnosti vstupují do další vrstvy, která v nich hledá více abstraktní vzory, které již nelze graficky reprezentovat. Pokud bychom na nízké úrovni identifikovali filtry pro detekci očí, úst, uší, nosu atd., tak si nemůžeme představit, že v další vrstvě bude filtr, který graficky bude hledat obličej. Ta již hledá vzory v matici příznaků nikoliv ve vstupním obrázku.



Obrázek 24 - intuice fungování více konvolučních vrstev

Ke klasifikaci obrazových dat sice můžeme používat plně propojené neuronové sítě, ale ty pro velká vstupní data budou mít mnoho parametrů. Jejich trénování, ale i provoz, budou časově a energeticky náročné. Je to z toho důvodu, že FCNN sítě nerespektují prostorové uspořádání vstupních dat.

To naopak konvoluční neuronové sítě dělají. Mají tak menší počet parametrů při zachování přesnosti.

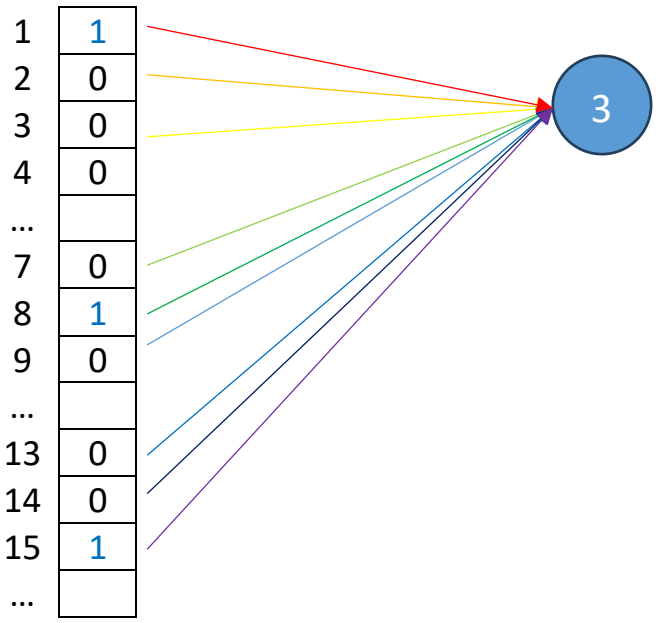
Konvoluční vrstvu si můžeme představit jako klasickou dopřednou neuronovou vrstvu. Pro přehlednost si to ukážeme na vstupních datech o rozměrech 6 x 6. Vstupní data mají tedy rozměr 36 hodnot. U plně propojené neuronové vrstvy by každý pixel ze vstupních dat byl propojen se všemi neurony první skryté vrstvy.

U konvoluční vrstvy s filtrem o velikosti 3 x 3 bude první neuron propojen pouze s 9 pixely s indexy 1, 2, 3, 7, 8, 9, 13, 14 a 15. Hodnoty konvolučního filtru vlastně korespondují s vahami propojení.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

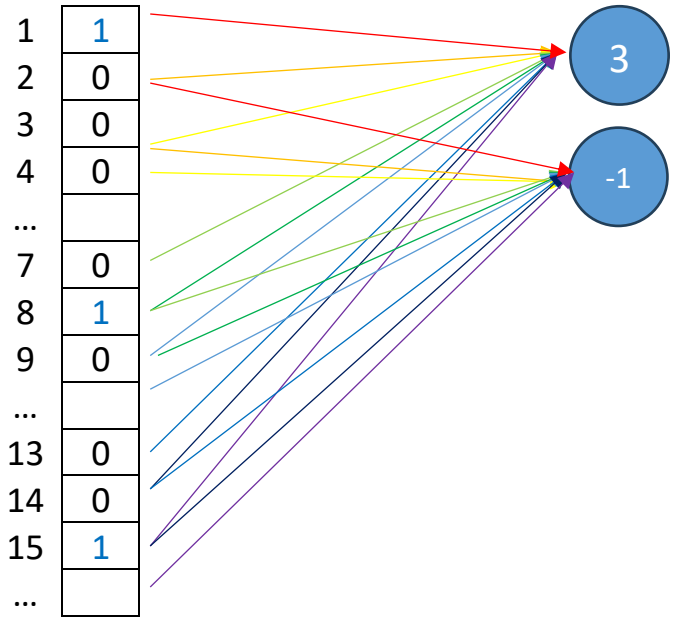
1	-1	-1
-1	1	-1
-1	-1	1

3			



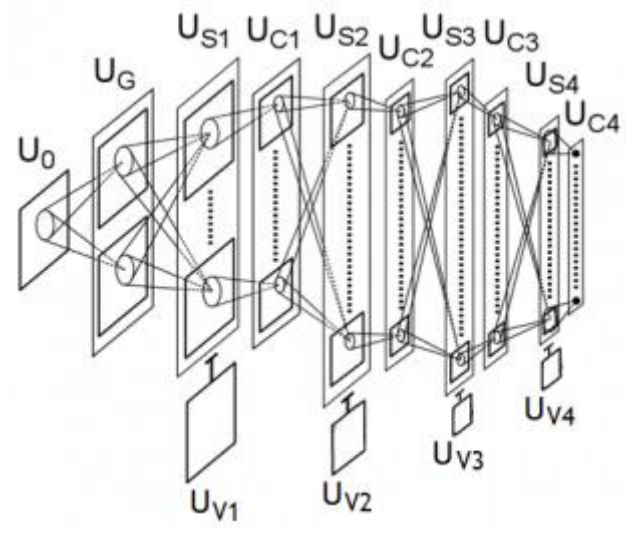
Předpokládejme, že konvoluční vrstva je vytvořena s parametrem stridem rovnému hodnotě 1. Tedy konvoluční filtr se posune o jeden pixel doprava. Filtr tedy propojí vstupní hodnoty s indexy 2, 3, 4, 8, 9, 10, 14, 15 a 16. Protože hodnoty konvolučního filtru se v průběhu posouvání nemění, jsou váhy vstupních hodnot do neuronů sdílené. V plně propojené vrstvě bychom při 36 vstupních hodnotách a 36 neuronech ve skryté vrstvě měli 1296 parametrů, pro které bychom museli v průběhu trénování hledat vhodné hodnoty.

Pokud budeme uvažovat konvoluční vrstvu s filtrem o velikosti 3 x 3, tak počet parametrů bude 9, protože s každým neuronem je propojeno pouze 9 vstupních hodnot, a navíc váhy jsou všemi neurony sdílené. To je důvod, proč jsou konvoluční sítě pro zpracování prostorových dat tak výhodné.



2.4 Historie

V historii vzniklo více architektur neuronových sítí. V roce 1959 David Hubeland a Torsten Wiesel zveřejnily Simple and Complex Cells. V roce 1979 Kunihiko Fukushima uvedl neuronovou síť Neocognitron pro rozpoznávání ručně psaných japonských znaků.[16]

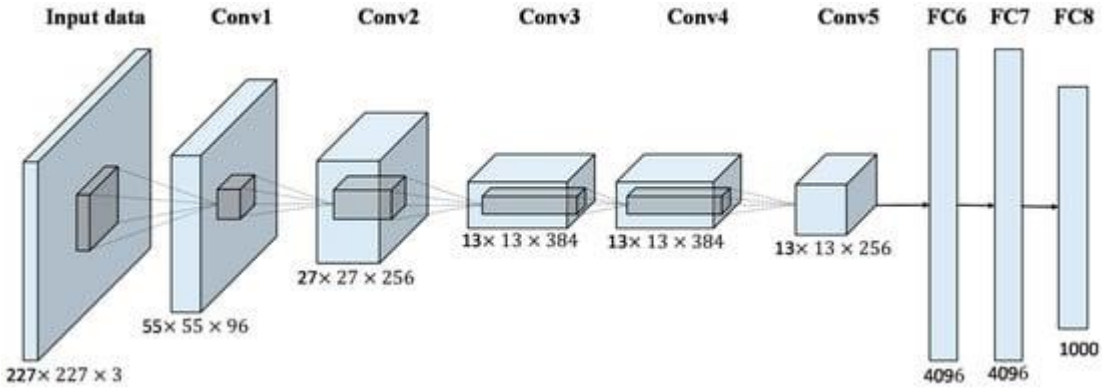


Obrázek 25 - Neocognitron

V roce 1998 Yann LeCun zveřejnil konvoluční neuronovou síť LeNet, která zpracovávala obrázky o rozměrech 28 x 28.[17]

Roku 2009 Fei-FeiLi a jeho tým zveřejnil velký dataset obrázků ImageNet [18]skládající se z 1,3 milionů obrázků patřících do 1000 tříd. Současně s tím vyhlásil soutěž mezi výzkumníky o nejlepší model. Díky soutěži vznikají návrhy nových topologií klíčových pro rozvoj aplikace neuronových sítí pro zpracování obrazu.

V roce 2012 Alex Krizhevsky a jeho neuronová síť AlexNet zvládla dataset ImageNet dataset rozpoznávat s chybovosti 15,3 %, což je o více než 10,8 procentního bodu méně než u druhého v pořadí. Hlavním výsledkem původní práce bylo zjištění, že pro vysoký výkon modelu je zásadní jeho hloubka, která byla výpočetně nákladná, ale byla proveditelná díky využití GPU při trénování. Důvodem úspěchu bylo použití ReLu aktivační funkce, velkého počtu filtrů v konvolučních vrstvách, paralelního trénování, augmentace dat (dodatečného upravování vstupních obrázků) a zavedení Dropout vrstvy. Dropout vrstva mění nastavení výstupu neuronů s určitou pravděpodobností na hodnotu 0. Jedná se o omezení vzájemných vztahů neuronů. Neuron se nemůže spoléhat na přítomnost jiných neuronů a minimalizuje se tím přetrénování neuronové sítě. [19]

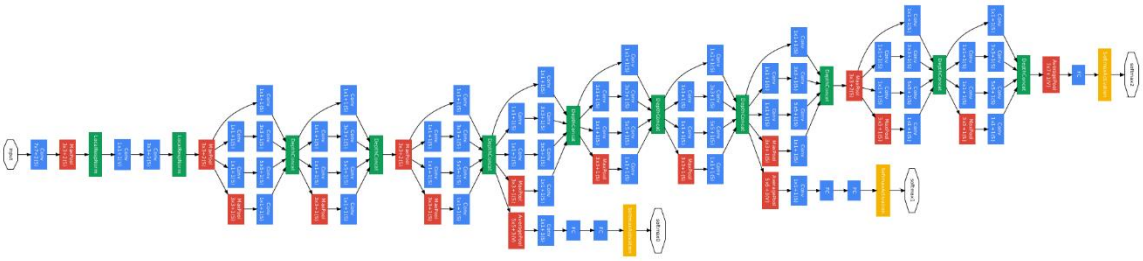


Obrázek 26 - Alexnet

V roce 2014 byla zveřejněna množina neuronových sítí VGG (VisualGeometry Group). Například VGG16 obsahuje 16 konvolučních vrstev. Tato architektura přináší hlubší architekturu CNN, která dosahovala nižší chybovosti v soutěži na datasetu ImageNet. VGG přichází s novou filosofií – zvětšením hloubky lze modelovat více nelinearit ve funkci zohledňování hloubky jako kritické složky při návrhu topologie. [14]

V roce 2014 je objevuje i GoogLeNet. Trendem je zvětšování hloubky sítě, ale bez použití plně propojených vrstev. GoogleNet má 12x méně parametrů než u AlexNet a 28x méně parametrů než u VGG. Zavádí se „inception“ modul. Cílem je aproximovat optimální lokální struktur CNN, což umožňuje

použít v jednom bloku více velikostí filtrů, místo abychom byli omezeni na jednu velikost filtru, které pak spojíme a předáme další vrstvě. [20]



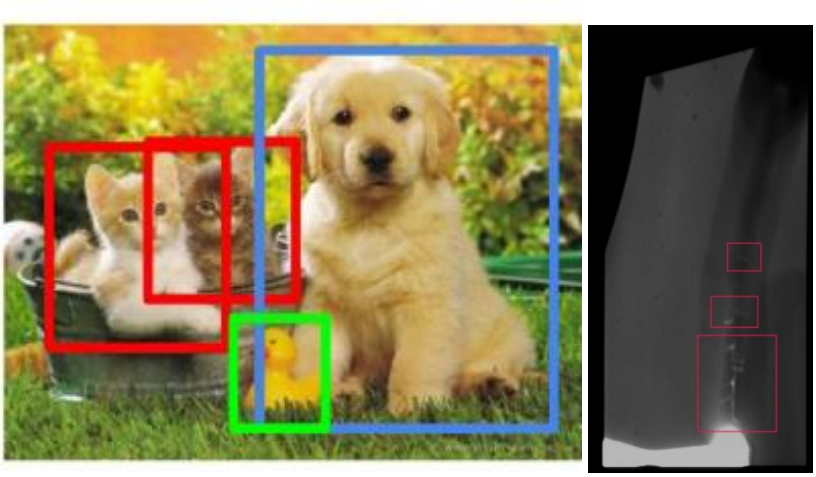
Obrázek 27 - GoogleNet

3 Detekce objektů

Další úlohou, která navazuje na klasifikaci obrázku je lokalizace. Pokud nám neuronová síť klasifikuje, že se na obrázku nachází kočka, budeme chtít určit i pozici, kde se kočka nachází. Zpravidla tuto informaci neuronové sítě vrací v podobě souřadnic obdélníku. [3]

Obrázek 28 – lokalizace

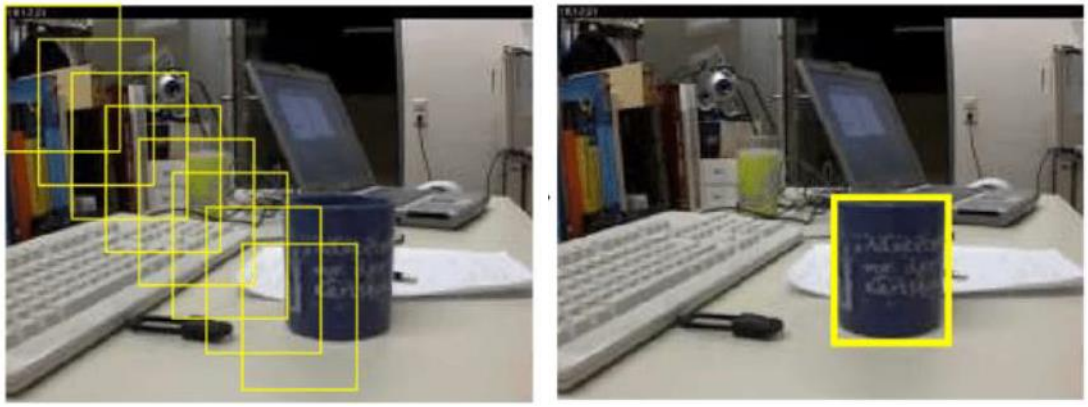
O něco pokročilejší úlohou je detekce objektů. V tomto případě neuronová síť vrací seznam a umístění více objektů a více typů objektů, které na obrázku detekovala. Tato schopnost má široké využití v mnoha oblastech, včetně autonomních vozidel, sledování osob, bezpečnostních systémů, průmyslového monitorování a mnoha dalších. Kdysi se k řešení této úlohy používaly klasické metody založené na extrakci rysů a klasifikaci, jako jsou například histogram orientovaných gradientů (HOG) kombinované s kaskádovými klasifikátory nebo příznaky Haar. [2]



Obrázek 29 - detekce objektů - 2 kočky, kačenka a pes, detekce vad na RTG snímku

S nástupem hlubokého učení se staly hluboké konvoluční neuronové sítě (CNN) dominantní metodou pro detekci objektů. Existuje množství architektur neuronových sítí jako je Faster R-CNN, YOLO (You Only Look Once) a SSD (Single Shot Multibox Detector), které dosahují vysoké přesnosti a rychlosti při detekci objektů v obrazech. [21]

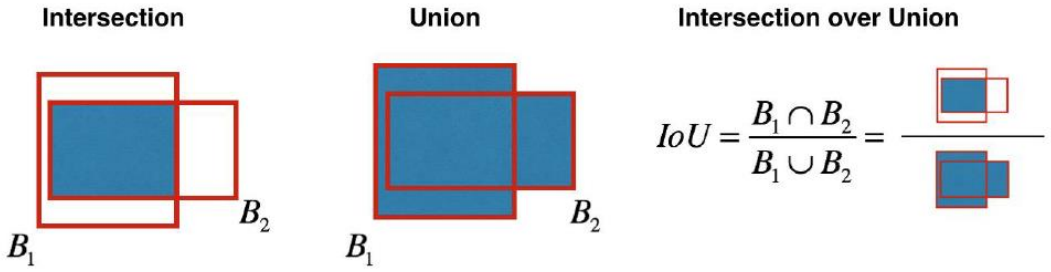
Základním principem detekce objektů je použití výše popsaného klasifikátoru na tzv. pohyblivé okno (sliding window). Část neuronové sítě je zodpovědná za klasifikaci obdélníkové oblasti. Používá k tomu architektury, které jsme si představili u klasifikace – VGG, Alexnet, Resnet atd. Novou úlohou je výběr oblasti obrázku, nazývaný Region Proposal Generator.



Obrázek 30 - generátor oblastí

Různé neuronové sítě se liší způsobem návrhu oblastí, kde by se mohl vyskytovat objekt. Mohou používat pomalé algoritmy Selective Search a EdgeBoxes. Například síť FR-CCN přišla s částí nazývanou Region Proposal Network (RPN), což je plně konvoluční síť, která generuje návrhy s různými měřítky a poměry stran. Místo pevně definovaných velikostí a pozic prohledávaných oblastí se neuronová síť sama na datech naučí, jak generovat menší počet přesnějších oblastí.

U úloh pro detekci objektu se používá následující metrika pro vyhodnocování přesnosti předpovědi. IoU (Intersection over Union) dává do poměru průnik navrhované oblasti se skutečnou oblastí vůči sjednocení těchto oblastí. Předpokládejme, že oblast B_1 je správná oblast získaná z učebních dat a oblast B_2 je oblast vrácená neuronovou sítí. Potom se metrika IoU vypočítá následovně:



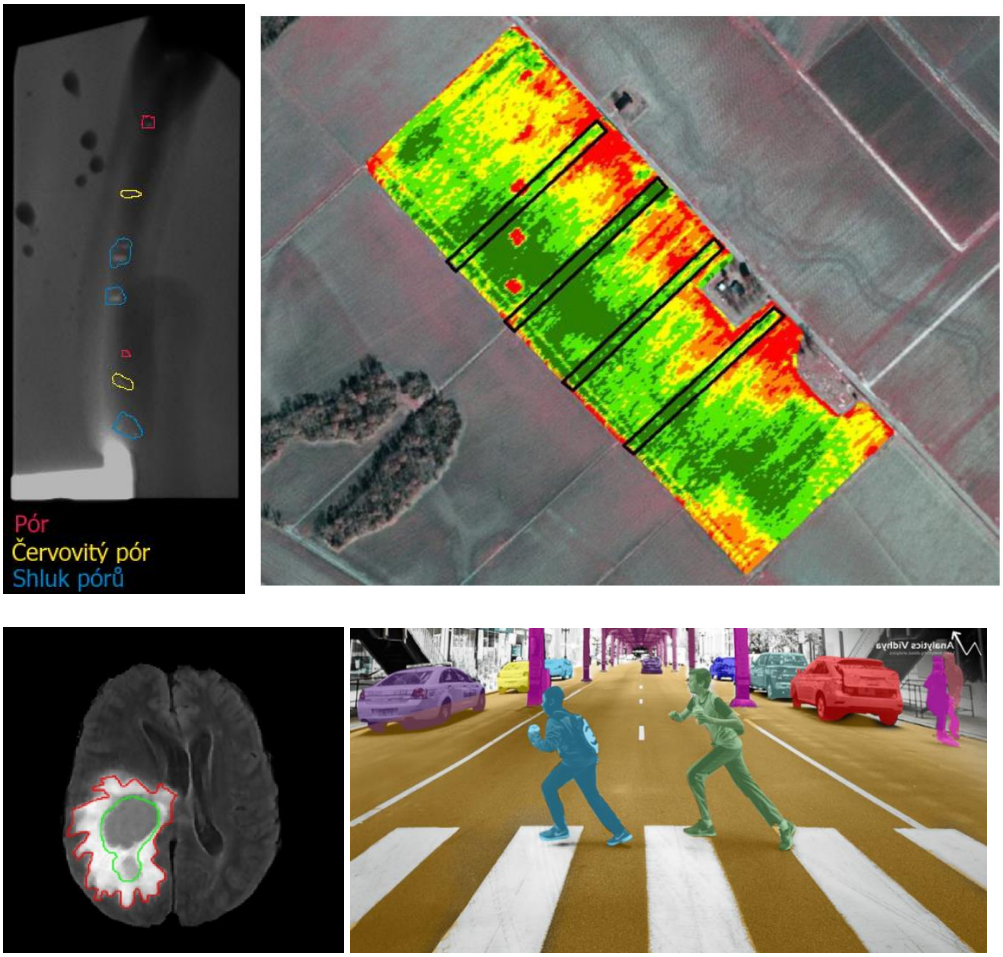
Obrázek 31 - metrika IoU

4 Segmentace

Poslední zmíněnou úlohou v počítačovém vidění je segmentace obrazu. Segmentace obrazu je proces rozdělení digitálního obrazu do jednotlivých segmentů nebo oblastí, které mají podobné vlastnosti nebo charakteristiky. Cílem segmentace je identifikovat a oddělit jednotlivé objekty nebo regiony z obrazu, což umožňuje další analýzu a porozumění obsahu obrazu.

Segmentace obrazu má široké využití v mnoha oblastech, včetně lékařství (např. segmentace orgánů na medicínských snímcích), průmyslu (kontrola kvality výrobků), bezpečnosti (detekce a sledování objektů) nebo v počítačové grafice (vytvoření masky pro úpravy obrazu). Kvalita segmentace je klíčová pro úspěšnou analýzu a manipulaci s obrazem, ačkoliv může být ovlivněna faktory jako je osvětlení, šum nebo překážky v obraze. [1]

U detekce objektů byl výstupem klasifikovaný obdélník, u segmentace je každý pixel přiřazen do určité třídy (třídou může být i pozadí). Výstupní data tedy budou přibližně stejné velikosti jako vstupní data. Velmi často je výstupem matice příznaků jednotlivých pixelů. V některých případech může být výstupem množina polygonů, které tvoří hranice jednotlivých segmentů. Polygony jsou pak definovány řadou bodů, kdy se první a poslední bod shodují. [3]



Obrázek 32 - ukázky použití segmentace

Sémantická segmentace přiřazuje každý pixel do určité třídy. Oproti tomu instanční segmentace přiřazuje pixely odpovídajícím instancím objektu. V prvním případě dokážeme určit, že pixel náleží nějaké osobě. U instanční segmentace jsme schopni určit, které konkrétní osobě patří.

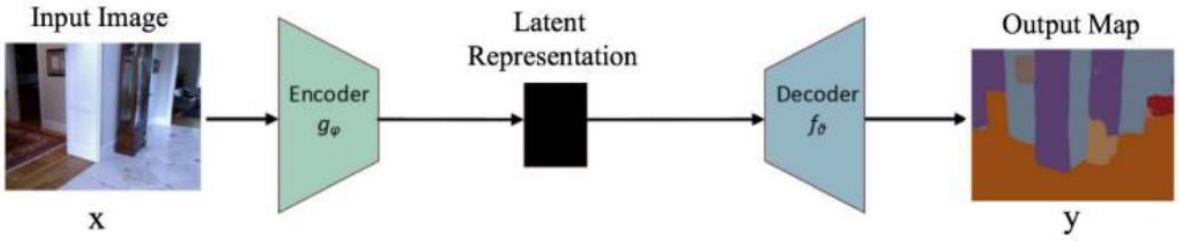
Obrázek 33 - sémantické a instanční segmentace

Neuronové sítě pro segmentaci jsou založené na principu Encoder – decoder nebo auto encoder. Jedná se o generativní sítě (Generative Adversarial Networks GAN), které na základě

vstupního popisu dat generují výstupní data. V našem případě jsou vstupními daty pixely obrázku a výstupními daty je příslušnost každého pixelu k segmentu. Příkladem takové sítě je třeba U-net. [22]

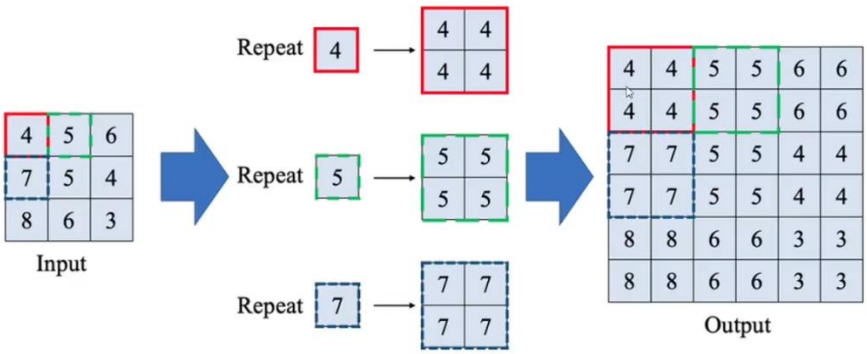
První část neuronové sítě je konvoluční. Používá úplně stejné techniky k získání klíčových vlastností dat. Vstupní data jsou tedy transformována do vnitřní skryté reprezentace, která popisuje klíčové vlastnosti.

Druhá část neuronové sítě funguje právě naopak. Ze skryté komprimované reprezentace vstupních dat generuje výstup. U segmentace bude mít stejný rozměr jako vstupní data, ale výstup se bude lišit od vstupních dat. Respektive klíčové vlastnosti budou reprezentovány jiným způsobem. [23]



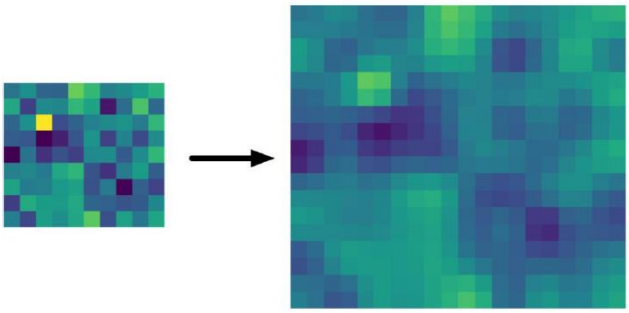
Obrázek 34 - princip GAN sítě

Novým typem vrstvy je UpSampling vrstva, která zvětšuje původní rozměr vstupu. Například 1 pixel bude zvětšen do matice 2 x 2 pixelů. Přístupů, jak vytvořit UpSampling vrstvu, je více. Ten nejjednodušší je replikace. Tento způsob není vhodný, protože výstup bude mít velmi hrubé, kostrbaté okraje.



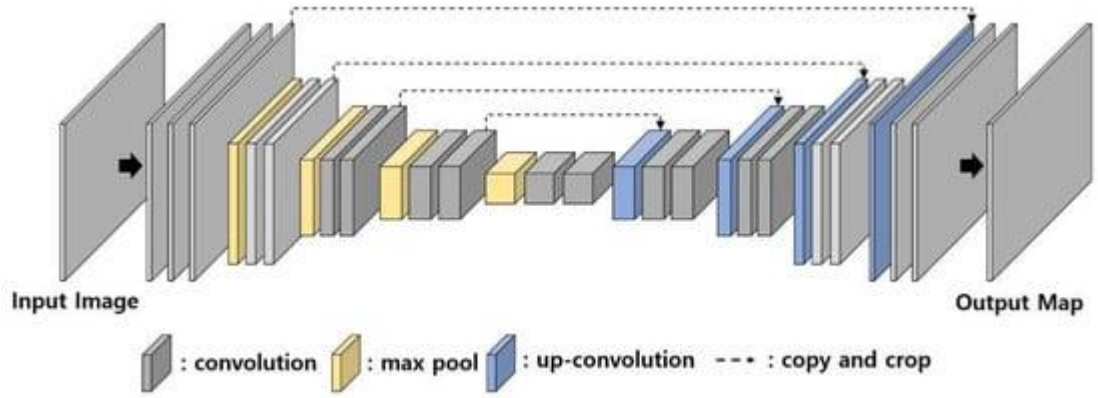
Obrázek 35 - UpSampling pomocí replikace

Proto se používají různé filtry, které nám krajní body průměrují a vytváří nám výstupní obrázky hladší.



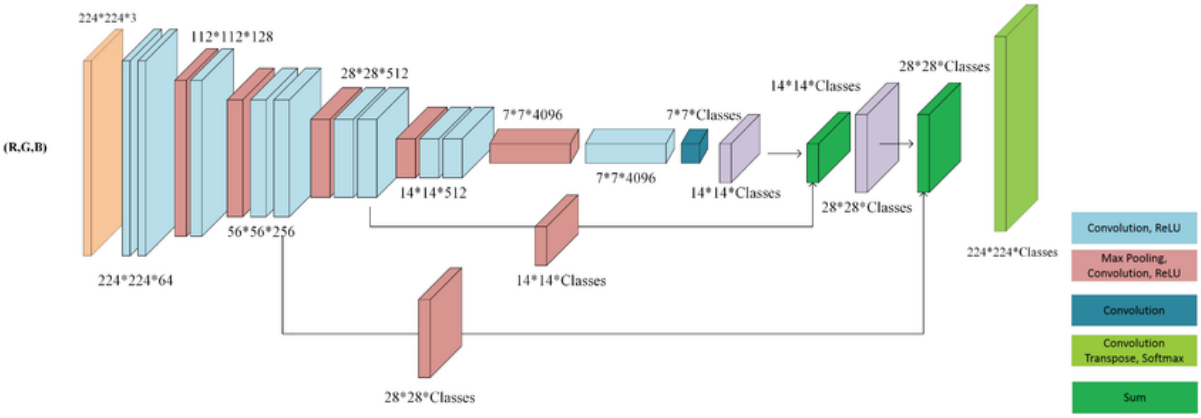
Obrázek 36 - UpSampling pomocí průměrování

U sítí se zpravidla vyskytuje i skip spojení, kdy jsou odpovídající vrstvy v konvoluční a dekonvoluční části přímo propojeny.



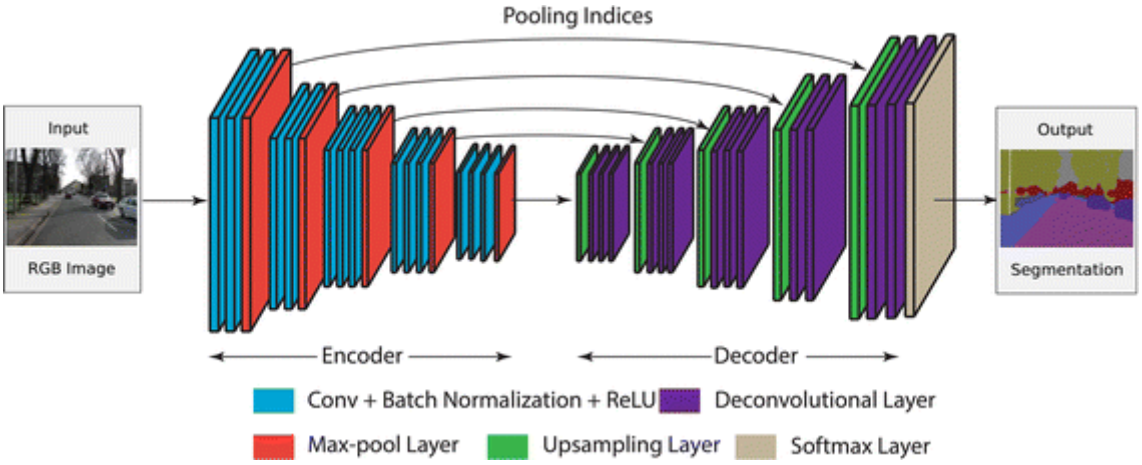
Obrázek 37 - skip spojení

Známé architektury segmentačních sítí jsou například Fully convolutional neural network (FCN).



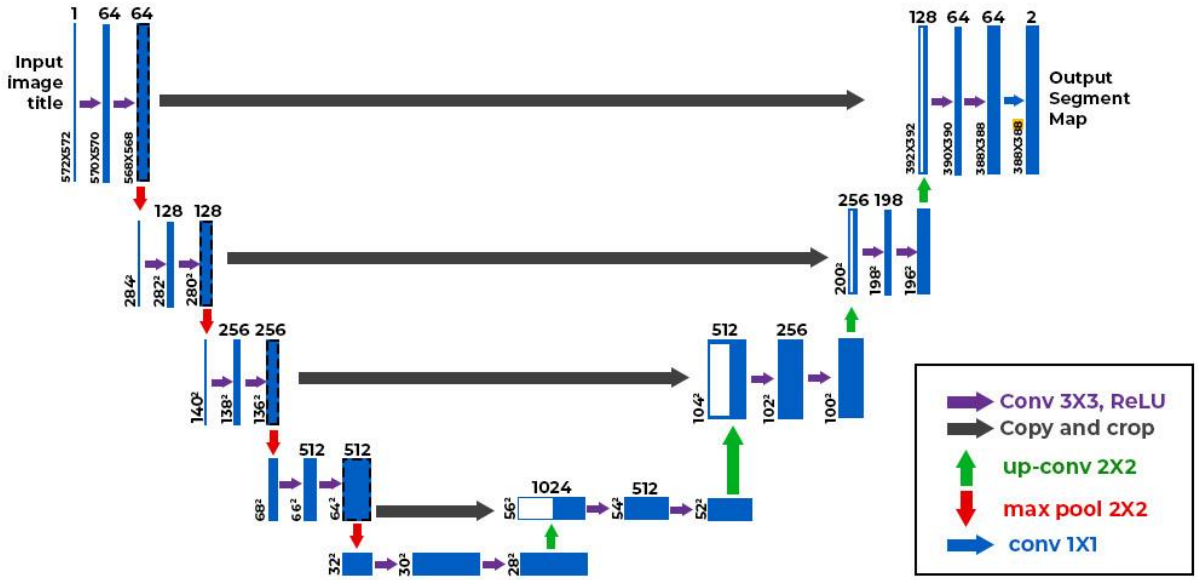
Obrázek 38 – FCN

Dalším příkladem je SegNet.



Obrázek 39 - Segnet

Naposledy zmíníme architekturu U-Net.



Obrázek 40 - U-net

Použitá literatura

[1] MICROSOFT. What is computer vison? *What is computer vision?* [online]. 2024 [cit. 2024-05-03]. Dostupné z: https://azure.microsoft.com/en-in/resources/cloud-computing-dictionary/what-is-computer-vision#object-classification

[2] Different Computer Vision Tasks. BANERJEE, Ananya. *Different Computer Vision Tasks* [online]. 2019 [cit. 2024-05-03]. Dostupné z: https://ananya-banerjee.medium.com/different-computer-vision-tasks-b3b49bbae891

[3] What Is Computer Vision? [Basic Tasks & Techniques]. BANDYOPADHYAY, Hmrishav. *What Is Computer Vision? [Basic Tasks & Techniques]* [online]. 2022 [cit. 2024-05-03]. Dostupné z: https://www.v7labs.com/blog/what-is-computer-vision

[4] *OpenCV documentation* [online]. 2024 [cit. 2024-05-03]. Dostupné z: https://docs.opencv.org/4.9.0/d7/dbd/group__imgproc.html

[5] TOWARDS DATA SCIENCE. Understanding K-means Clustering in Machine Learning. *Understanding K-means Clustering in Machine Learning* [online]. 2018 [cit. 2024-05-03]. Dostupné z: https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1

[6] Guide on Support Vector Machine (SVM) Algorithm. SAINI, Anshul. *Guide on Support Vector Machine (SVM) Algorithm* [online]. 2024 [cit. 2024-05-03]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/

[7] An Introduction to Convolutional Neural Networks (CNNs). *An Introduction to Convolutional Neural Networks (CNNs)* [online]. 2023 [cit. 2024-

05-03]. Dostupné z: https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns

[8] VIKRAM. Comprehensive Guide to Edge Detection Algorithms. *Comprehensive Guide to Edge Detection Algorithms* [online]. 2022 [cit. 2024-05-03]. Dostupné z: https://www.analyticsvidhya.com/blog/2022/08/comprehensive-guide-to-edge-detection-algorithms/

[9] SHARDA, Aryaman. How Image Edge Detection Works. *How Image Edge Detection Works* [online]. 2018 [cit. 2024-05-03]. Dostupné z: https://aryamansharda.medium.com/how-image-edge-detection-works-b759baac01e2

[10] BASAVARAJAIAH, Madhushree. 6 basic things to know about Convolution. *6 basic things to know about Convolution* [online]. 2019 [cit. 2024-05-03]. Dostupné z: https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411

[11] Exploring Feature Extraction with CNNs. SILVA, Rodrigo. *Exploring Feature Extraction with CNNs* [online]. 2023 [cit. 2024-05-03]. Dostupné z: https://towardsdatascience.com/exploring-feature-extraction-with-cnns-345125cefc9a

[12] THEVENOT, Axel. Conv2d: Finally Understand What Happens in the Forward Pass. *Conv2d: Finally Understand What Happens in the Forward Pass* [online]. 2020 [cit. 2024-05-03]. Dostupné z: https://towardsdatascience.com/conv2d-to-finally-understand-what-happens-in-the-forward-pass-1bbaafb0b148

[13] BROWNLEE, Jason. A Gentle Introduction to Pooling Layers for Convolutional Neural Networks. *A Gentle Introduction to Pooling Layers for Convolutional Neural Networks* [online]. 2019 [cit. 2024-05-03]. Dostupné z:

https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

[14] ROHINI. Everything you need to know about VGG16. *Everything you need to know about VGG16* [online]. 2021 [cit. 2024-05-03]. Dostupné z: https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

[15] NANOS, Georgios. Computer Vision: Differences Between Low-Level and High-Level Features. *Computer Vision: Differences Between Low-Level and High-Level Features* [online]. 2024 [cit. 2024-05-03]. Dostupné z: https://www.baeldung.com/cs/cv-low-vs-high-level-features

[16] FUKUSHIMA, Kunihiko. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. In: *Biological Cybernetics*. 1980. a: a, 1980, s. 193–202. ISBN 1432-0770. ISSN 1432-0770. Dostupné z: doi:https://doi.org/10.1007/BF00344251

[17] LeNet. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2023 [cit. 2024-05-03]. Dostupné z: https://en.wikipedia.org/wiki/LeNet

[18] ImageNet. *ImageNet* [online]. 2021 [cit. 2024-05-03]. Dostupné z: https://www.image-net.org/

[19] AlexNet. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2023 [cit. 2024-05-03]. Dostupné z: https://en.wikipedia.org/wiki/AlexNet

[20] Understanding GoogLeNet Model – CNN Architecture. *Geeks for geeks* [online]. 2021 [cit. 2024-05-03]. Dostupné z: https://www.geeksforgeeks.org/understanding-googlenet-model-cnn-architecture/

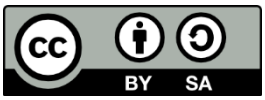
[21] VISO.AI. Object Detection in 2024: The Definitive Guide. VISO.AI. *Viso.ai* [online]. 2024 [cit. 2024-05-03]. Dostupné z: https://viso.ai/deep-learning/object-detection/

[22] U-net. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2024 [cit. 2024-05-03]. Dostupné z: https://en.wikipedia.org/wiki/U-Net

[23] How can generative adversarial networks learn real-life distributions easily. *How can generative adversarial networks learn real-life distributions easily* [online]. 2021 [cit. 2024-05-03]. Dostupné z: https://www.microsoft.com/en-us/research/blog/how-can-generative-adversarial-networks-learn-real-life-distributions-easily/

UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vytvořeno v rámci projektu **Digitalizace studijních Agend, Nové TechnologiE, systémy a přístupy k výuce na UPCE**, reg. č. NPO_UPCE_MSMT-16591/2022.



Financováno Evropskou unií
NextGenerationEU

Národní plán obnovy

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

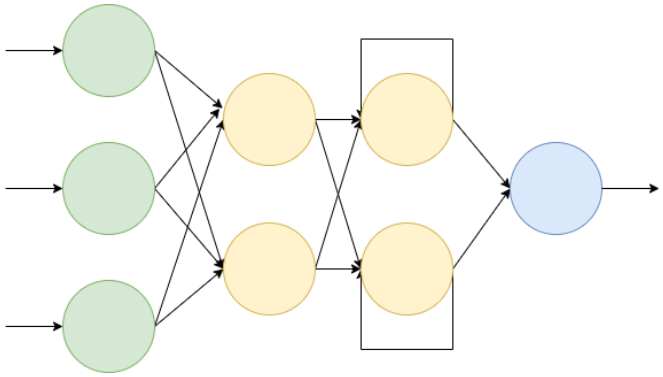
Machine learning & AI

Rekurentní neuronové sítě

Ing. Martin Pozdílek, Ph.D.

1 Rekurentní neuronová síť

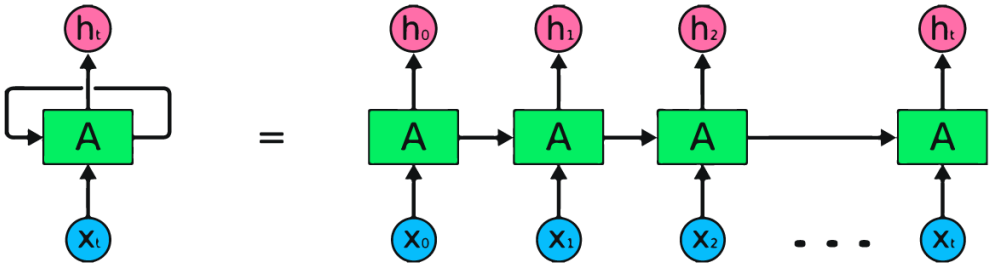
Dalším typem neuronových sítí jsou rekurentní sítě (Recurrent neural network – RNN). V tradiční neuronové síti jsou vstupy a výstupy na sobě nezávislé, zatímco výstup v RNN je závislý na předchozích prvcích v sekvenci. Jedná se o sítě, ve kterých se na některých místech objevuje zpětné propojení do předchozích vrstev. Rekurentní sítě také sdílejí parametry napříč jednotlivými vrstvami sítě. Toto propojení dokáže simulovat **krátkodobou paměť** a spojovat informace, které jsou posunuté v čase. Délka paměti závisí na délce zpětné vazby. Pokud zpětná vazba vede do stejného neuronu, propojují se hodnoty v čase t a t+1. Pokud by zpětná vazba vedla do předchozího neuronu, propojí se hodnoty v čase t a t+2. Nevýhodou může být ztracení gradientu, pokud se hodnoty, které si má síť pamatovat, neopakují. Jedná se tedy o velmi krátkodobou paměť, která je velmi závislá na délce mezery mezi informacemi, které je potřeba propojit a opakováním informace ve vstupních datech. Pro každý vstup se používají stejné parametry, protože pro vytvoření výstupu se provádí stejná úloha na všech vstupech nebo skrytých vrstvách. Tím se na rozdíl od jiných neuronových sítí snižuje složitost parametrů.



Obrázek 1 - rekurentní neuronová síť

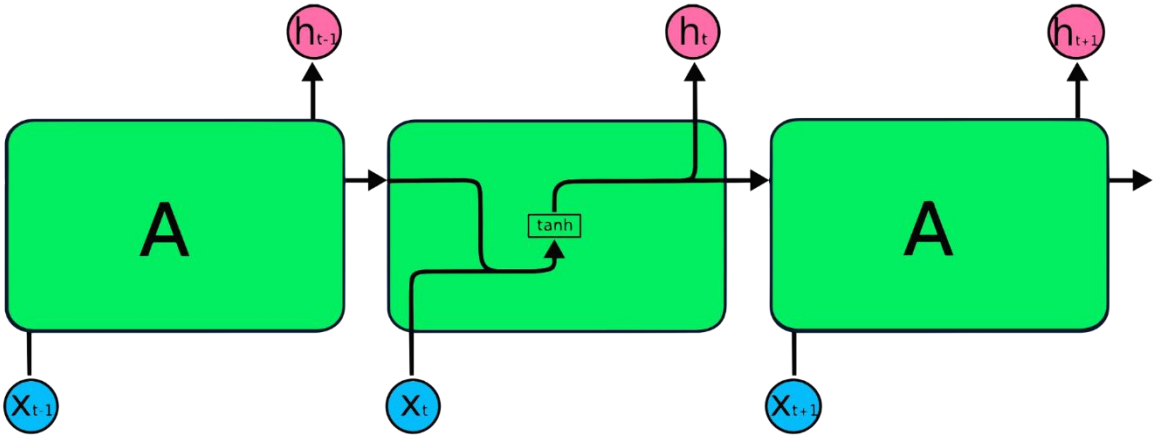
Tento typ sítě se hodí pro zpracování dat, ve kterých se vyskytuje časový aspekt. Příkladem může být zpracování časových řad vývoje ceny akcí, zpracování zvuku nebo hlasu, zpracování řeči atd.

Pokud budeme chtít například odhadovat budoucí ceny akcí na základě minulého vývoje, tak vstupní dataset bude vypadat jako řada čísel X_0 až X_t. Například řada čísel bude vypadat takto [45, 56, 45, 49, 50, …]. V jednoduché rekurentní síti budeme z hodnot X_0 a X_1 předvídat hodnotu X_2. RNN bude obsahovat zpětnou vazbu, která umožní zpracovávat časově sousední data. Při zpracování této sítě se jednoduchý diagram vlastně rozpadne na sekvenční zpracování časové řady. Tomuto se říká rozbalení (unfolding).



Obrázek 2 – Rekurentní neuron a jeho rozbalení

V RNN informace prochází cyklem, takže výstup je určen aktuálním vstupem a dříve přijatými vstupy. Vstupní vrstva X zpracovává počáteční vstup a předává jej střední vrstvě A. Střední vrstva se skládá z několika skrytých vrstev, z nichž každá má své aktivační funkce, váhy a zkreslení. Tyto parametry jsou standardizovány napříč skrytou vrstvou, takže místo vytvoření více skrytých vrstev se vytvoří jedna a zacyklí se. Základní jednotkou v rekurentní neuronové síti je rekurentní jednotka, která se výslovně nenazývá "rekurentní neuron", ale chová se tak. Tato jednotka má jedinečnou schopnost udržovat skrytý stav, což síti umožňuje zachycovat sekvenční závislosti tím, že si při zpracování pamatuje předchozí vstupy.



Obrázek 3 - Jednoduchá RNN

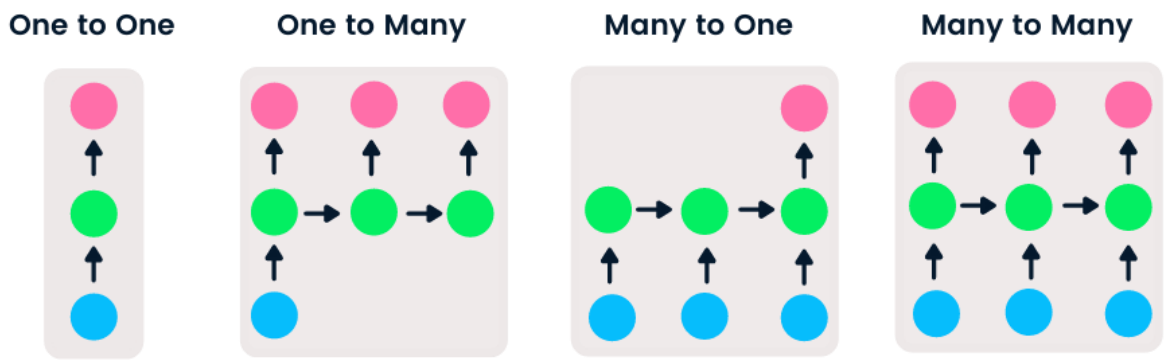
Verze LSTM (Long Short-Term Memory) a GRU (Gated Recurrent Unit) zlepšují schopnost RNN zpracovávat dlouhodobé závislosti. Namísto tradiční zpětné propagace používají rekurentní neuronové sítě k určení gradientu algoritmy zpětné propagace v čase (BPTT). Při zpětném šíření model upravuje parametr pomocí výpočtu chyb z výstupu do vstupní vrstvy. BPTT sčítá chyby v každém časovém kroku, protože RNN sdílí parametry v každé vrstvě.

1.1 Typy rekurentních neuronových sítí

Dopředné neuronové sítě mají jeden vstup a výstup, zatímco rekurentní neuronové sítě jsou flexibilní, protože délku vstupů a výstupů lze měnit. Tato flexibilita umožňuje RNN vytvářet hudbu, klasifikovat sentiment nebo provádět strojový překlad.

Existují čtyři typy RNN založené na různých délkách vstupů a výstupů.

1. One-to-one je jednoduchá neuronová síť. Běžně se používá pro problémy strojového učení, které mají jediný vstup a výstup.
2. One-to-many má jeden vstup a více výstupů. Používá se pro generování popisků obrázků. Tedy z jednoho vstupu vznikne více časově závislých výstupů.
3. Many-to-one bere posloupnost více vstupů a předpovídá jeden výstup. Je oblíbený v klasifikaci sentimentu, kde vstupem je text a výstupem nálada textu. Můžeme to přiblížit klasifikaci textu, časové řady apod.
4. Many-to-many přijímá více vstupů a výstupů. Nejběžnější aplikací tohoto typu neuronové sítě je strojový překlad nebo chatovací aplikace.



Obrázek 4 - Typy RNN

1.2 Rozdíly mezi CNN a RNN

Konvoluční neuronové sítě se převážně používají pro prostorová data, která jsou relativně řídká. Na tento typ dat lze s výhodou použít konvoluce, která výrazně redukuje dimenzi dat a vybírá z nich klíčové vlastnosti. CNN se tedy používají zpravidla pro zpracování obrazových dat. U tohoto typu dat je důležitá prostorová souvislost dat. Tato data jsou k dispozici všechna v jeden časový okamžik, proto konvoluční neuronová síť má dopředný charakter. Vstupní a výstupní neuronová síť má pevně definovanou podobu, která se nemění.

Rekurentní neuronové sítě, jak již bylo řečeno se používají pro časové řady a sekvenční data. Proto se RNN používají pro zpracování zvukových dat, řeči, textu atp., ve kterých je časová závislost. Potřebná data jsou dostupná v jiné časové okamžiky, proto je třeba určité formy paměti. To je velmi

podobné i u biologických neuronových sítí, kdy pro zrakové podněty máme velmi velkou detailní krátkodobou paměť, kterou používáme pro extrakci klíčových vlastností. Tato paměť má trvání do jedné vteřiny. Naopak sluchová krátkodobá paměť je delší, okolo 10 vteřin, protože potřebujeme spojovat časově závislá data, ze kterých pak extrahujeme potřebné vlastnosti.

Rekurentní neuronová síť na rozdíl od konvoluční neuronové sítě nemá omezení v délce vstupů a výstupů. V topologii RNN se objevují různé zpětnovazební smyčky, které umožňují propojovat časová data.

Mezi hlavní výhody RNN patří:

- Schopnost zpracovávat sekvenční data různé délky, což je užitečné v aplikacích, jako je zpracování přirozeného jazyka, rozpoznávání řeči a analýza časových řad.
- RNN mají paměť. RNN mají schopnost uchovávat informace o předchozích vstupech v sekvenci pomocí skrytých stavů. To umožňuje RNN provádět úlohy, jako je předpovídání dalšího slova ve větě nebo předpovídání cen akcií.
- RNN je všestranné. Lze je použít pro širokou škálu úloh, včetně klasifikace, regrese a mapování sekvence na sekvenci.
- RNN jsou proti jiným architekturám neuronových sítí velmi flexibilní. Nejsou svázány přesně definovaným formátem a velikostí vstupů. Lze je kombinovat s jinými architekturami neuronových sítí.

RNN mají také nevýhody:

- Problém mizejícího gradientu se vyskytuje hlavně u mnohovrstevných sítí nebo u zpracování dlouhých sekvencí. Problém bude popsán níže.
- RNN jsou výpočetně náročné, zejména při zpracování dlouhých sekvencí nebo při použití složitých architektur.
- RNN jsou obtížně interpretovatelné, zejména pokud jde o pochopení toho, jak síť provádí předpovědi nebo rozhodnutí.

1.3 Architektura RNN

RNN mají stejnou vstupní a výstupní architekturu jako jakákoli jiná hluboká neuronová architektura. Rozdíly však vznikají ve způsobu toku informací ze vstupu na výstup. Na rozdíl od hlubokých neuronových sítí, kde máme pro každou hlubokou vrstvu jinou váhovou matici, v RNN zůstávají váhy v celé síti stejné.

Rekurentní neuronová síť se skládá z několika jednotek s pevnou aktivační funkcí, z nichž každá je určena pro různé časové kroky. Každá jednotka má vnitřní stav *h*. Tento skrytý stav označuje minulé

znalosti, které síť v daném časovém kroku aktuálně má. Tento skrytý stav se aktualizuje v každém časovém kroku, aby zaznamenal změnu znalostí sítě o minulosti. Skrytý stav má podobu vektoru představující aktuální stav sekvence. Skrytý stav je v každém časovém kroku aktualizován na základě aktuálního vstupu a předchozího skrytého stavu.

Skrytý stav se aktualizuje pomocí následujícího vztahu rekurence, kdy nový skrytý stav je výsledkem nějaké funkce, do které vstupuje původní skrytý stav a aktuální hodnota X).

$$h_t = f(h_{t-1}, X_t)$$

Velmi často se používá jako aktivační funkce hyperbolický tangent. Jeho vstupní parametry jsou upraveny podle vah, které se neuronová síť učí. Pak lze obecný tvar rozepsat následovně:

$$h_t = tanh(w_{hh} \cdot h_{t-1} + w_x \cdot X_t + bias)$$

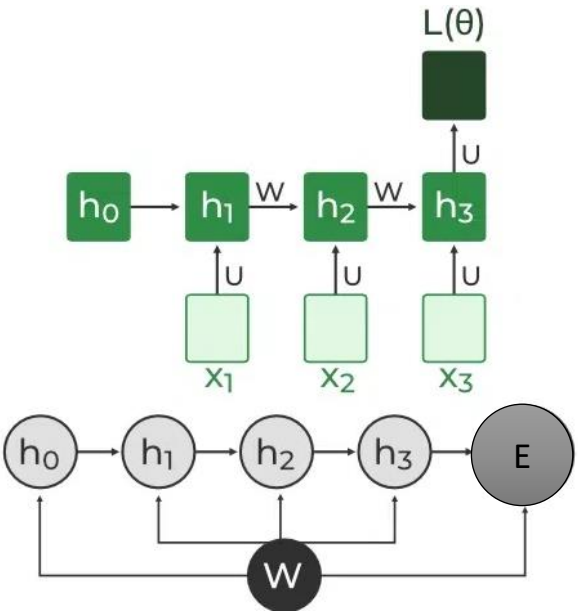
Výstupní hodnota z rekurentní jednotky se pak počítá z jejího aktuálního skrytého stavu.

$$Y_t = W_{hy} \cdot h_t + bias2$$

1.4 Učení RNN

Pro učení RNN sítě se používá algoritmus Backpropagation Through Time (BPTT). RNN je uspořádaná, a protože v uspořádané síti se každá proměnná počítá postupně v určitém pořadí, například nejprve se spočítá skrytý stav h_1, pak h_2, pak h_3 atd., proto použijeme zpětné šíření ve všech těchto skrytých časových stavech postupně.

Při učení RNN budeme chtít minimalizovat nákladovou funkci. Při predikci časové řady můžeme například použít klasickou funkci MSE. Nákladová funkce je závislá na skrytém stavu h_3. Stav h_3 je závislý na skrytém stavu h_2 a vahách w. Stav h_2 je závislý na h_1 a vahách w. A konečně stav h_1 je závislý na stavu h_0 a opět vahách w. Stav h_0 je nějaký konstantní počáteční stav. Zpravidla se jedná o vektor 0. Ale může nabývat i jiných hodnot. Jednou z metod je zakódování předpokladů o datech do počátečního skrytého stavu sítě. Například pro problém určení tónu řeči přednesené známou osobou lze do počátečního skrytého stavu zakódovat tóny minulých projevů této osoby. Další technikou je učinit počáteční skrytý stav trénovatelným parametrem. Ačkoli tyto techniky přidávají do sítě jen malé nuance, inicializace vektoru skrytého stavu na nuly je obvykle efektivní volbou.



Obrázek 5 - Backpropagation Through Time dopředná fáze

Nechť předpovídaný výstup sítě v každém časovém kroku je y_t a skutečný výstup je $\bar{y_t}$. Pak je chyba v každém časovém kroku dána vztahem:

$$E_t = -y_t \log(\bar{y}_t)$$

Celková chyba sekvence je pak:

$$E = \sum_{t=1}^{T} E_t = \sum_{t=1}^{T} -y_t \log(\bar{y}_t)$$

Pro zjištění gradientu sítě opět budeme počítat parciální derivace podobně jako u klasického backpropagation algoritmu.

$$\frac{\delta E}{\delta w} = \sum_{t=1}^{T} \frac{\delta E_t}{\delta w}$$

Pro jednoduchost můžeme uvažovat, že aplikujeme algoritmus zpětného šíření chyby pouze na jedno opakování.

$$\frac{\delta E}{\delta w} = \sum_{t=1}^{T} \frac{\delta E_t}{\delta h_3} \frac{\delta h_3}{\delta w}$$

Přičemž víme, že

$$h_3 = \tanh(w \cdot h_2 + b)$$

Protože h_3 je závislé na w, nemůžeme ho derivovat jako konstantu. Derivace h_3 se bude skládat ze dvou částí – explicitní a implicitní.

Explicitní část má následující podobu, kde všechny vstupy považujeme za konstantní.

$$\frac{\delta h_3^+}{\delta w}$$

Implicitní část je součtem všech nepřímých cest z h_3 do w.

Pokud spojíme explicitní a implicitní část dostaneme:

$$\frac{\delta h_3}{\delta w} = \frac{\delta h_3^+}{\delta w} + \frac{\delta h_3}{\delta h_2}\frac{\delta h_2}{\delta w} = \frac{\delta h_3^+}{\delta w} + \frac{\delta h_3}{\delta h_2}\left[\frac{\delta h_2^+}{\delta w} + \frac{\delta h_2}{\delta h_1}\frac{\delta h_1}{\delta w}\right] = \frac{\delta h_3^+}{\delta w} + \frac{\delta h_3}{\delta h_2}\frac{\delta h_2^+}{\delta w} + \frac{\delta h_3}{\delta h_2}\frac{\delta h_2}{\delta h_1}\left[\frac{\delta h_1^+}{\delta w}\right]$$

Pro zjednodušení zkrátíme některé cesty.

$$\frac{\delta h_3}{\delta w} = \frac{\delta h_3^+}{\delta w} + \frac{\delta h_3}{\delta h_2}\frac{\delta h_2^+}{\delta w} + \frac{\delta h_3}{\delta h_1}\frac{\delta h_1^+}{\delta w}$$

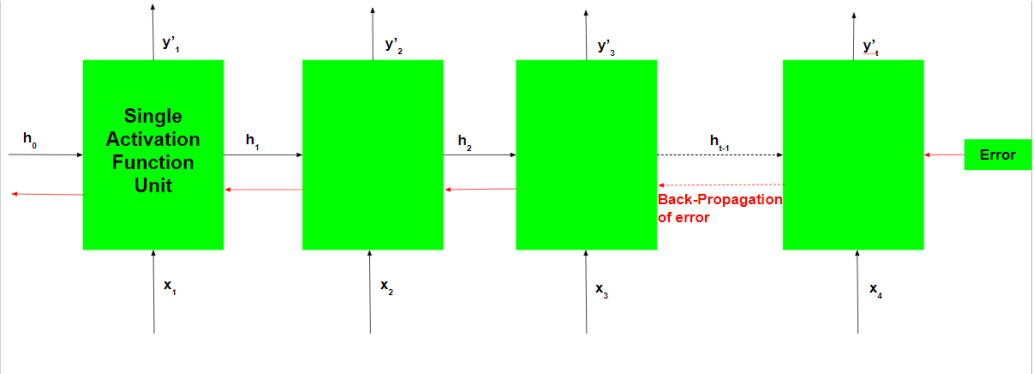
Pokud vše spojíme, získáme výpočet gradientu neuronové sítě

$$\frac{\delta E}{\delta w} = \frac{\delta E}{\delta h_3}\sum_{t=1}^{T}\frac{\delta h_3}{\delta w h_t}\frac{\delta h_t}{\delta w}$$

V krátkosti tedy může shrnout postup učení RNN. Na vstupu předpokládejme časově závislá data X.

- Do RNN vstupuje první hodnota X_0
- Na základě této hodnoty je vypočítán vnitřní stav h_0.
- Do RNN vstupuje druhá hodnota X_1
- Vnitřní stav h_1 je spočítán z X_1 z vnitřního stavu h_0.
- Toto opakujeme tolikrát, kolik je časových hodnot ve vstupních datech X
- Po zpracování všech vstupních hodnot X se vypočítá výstup z konečného aktuálního stavu.
- Výstup se pak porovná se skutečným výstupem, tj. cílovým výstupem, a vygeneruje se chyba.
- Chyba se pak zpětně přenáší do sítě, aby se aktualizovaly váhy, a proto je síť (RNN) trénována pomocí zpětného šíření v čase.

Backpropagation Through Time se tedy od typického Backpropagation liší pouze tím, že chyby v jednotlivých časových krocích se sčítají a vypočítává se celková chyba.



Obrázek 6 - Backpropagation Through Time

1.5 Problémy

Jednoduché modely RNN obvykle narážejí na dva hlavní problémy. Tyto problémy se týkají gradientu, což je sklon ztrátové funkce spolu s chybovou funkcí.

1. Problém mizejícího gradientu nastává, když se gradient stane tak malým, že se aktualizace parametrů stane bezvýznamnou, a nakonec se algoritmus přestane učit. To se děje například u úloh jako je generování textu, strojový překlad a předpovídání vývoje na burze.
2. Problém explodujícího gradientu nastává, když se gradient stane příliš velkým, čímž se model stane nestabilním. Velmi často má gradient exponenciální podobu. V tomto případě se hromadí větší gradienty chyb a váhy modelu se stávají příliš velkými. Tento problém může způsobit delší dobu učení a špatný výkon modelu.

Jednoduchým řešením těchto problémů je snížení počtu skrytých vrstev v neuronové síti, čímž se sníží určitá složitost RNN.

Druhou možností je nastavení prahové hodnoty na gradienty, které se předávají zpět v čase. Toto řešení však není považováno za řešení problému a může také snížit účinnost sítě.

Jednoduché opakovací moduly RNN, které mají základní strukturu s jednou tanh vrstvou, často trpí krátkou pamětí, kdy má síť problém uchovat informace o starších, předchozích vstupních hodnotách a vznikají tak výše popsané problémy. Pokud se jako aktivační funkce použije tanh nebo relu, nelze zpracovat velmi dlouhé sekvence.

Tyto problémy lze také vyřešit použitím pokročilých architektur RNN, jako jsou Long Short Term Memory (LSTM) a Gated recurrent unit (GRU).

Obecně trénování rekurentních neuronových sítí je velmi obtížná úloha, kdy je na rozdíl od tradičních dopředných neuronových sítí potřeba zohlednit i předchozí stav sekvence, což jim umožňuje modelovat časové závislosti v datech.

2 Příklady rekurentních sítí

2.1 Hopfieldova síť

Hopfieldova neuronová síť z roku 1982 je pojmenována po svém vynálezci Dr. John J. Hopfield. Jedná se o jednu z nejstarších a nejjednodušších forem neuronových sítí. Jejím hlavním účelem je uchovávat a obnovovat vzory.

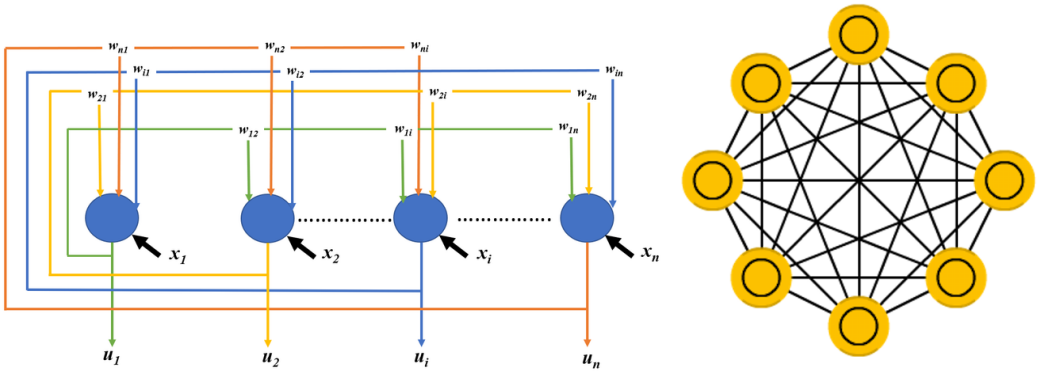
Neuronová síť se skládá z jedné vrstvy z plně propojených N rekurentních neuronů. Obecně se používá při provádění autoasociačních a optimalizačních úloh. Počítá se pomocí konvergujícího interaktivního procesu a generuje jinou odezvu než běžné neuronové sítě.

Původně se jedná o diskrétní neuronovou síť, která pracuje s logickými hodnotami 0 a 1, a používá skokovou bipolární aktivační funkcí. Ta pracuje jako autoasociativní paměť, do které se ve fázi učení uloží bipolární vzory obsažené v trénovacích datech. Tyto vzory, pokud se částečně poškodí a síti znovu předloží ve fázi vybavování, dokáže síť opravit. Proces učení probíhá bez učitele, síť se učí opravovat poškozené vzory.

Při předložení poškozených vzorů se síť snaží zaujmout stav v lokálním minimu odpovídající danému vzoru (atraktoru). Během učení sítě však vznikají ale i tzv. falešné atraktory. Jedná se vlastně o falešné vzory, se kterými se během učení nesetkala. S počtem vzorů v trénovacích datech roste během učení i počet falešných atraktorů. Jinými slovy kapacita paměti je omezená a trénovací data nesmí obsahovat příliš mnoho vzorů.

Hopfieldova síť se skládá z n předem definovaných neuronů, nejedná se tedy o obecnou RNN architekturu. Do každého neuronu vstupuje jedna hodnota ze vstupních dat a zároveň z každého neuronu vystupuje jedna hodnota výstupních dat, na obrázku označeny jako x a u.

Každý neuron je plně propojen se všemi ostatními neurony. Jeho výstup je vstupem pro všechny ostatní neurony, ale ne pro něj. Možností, jak znázornit Hopfieldovu síť je mnoho. Na prvním obrázku jsou neurony seřazeny do řady, na druhém obrázku jsou umístěny na kružnici, aby lépe vyniklo jejich propojení.



Obrázek 7 - Hopfieldova neuronová síť

Do Hopfieldovy neuronové sítě vstupují data, která mohou být poškozená a cílem neuronové sítě je vybavit se podle nich odpovídajícím vzorem z trénovacích dat, který je pak výstupem z Hopfieldovy sítě. Celkově lze říci, že Hopfieldova síť funguje jako dynamický systém, který iterativně konverguje k stabilnímu stavu, což odpovídá uloženým vzorům. Její jednoduchá struktura a učení na základě energie ji činí užitečným nástrojem pro asociativní paměť a řešení optimalizačních problémů.

Váhy spojení mezi neurony jsou použity k uchování informace. Tyto váhy jsou symetrické a jsou definovány podle vzorců, které závisí na vstupech a jejich vahách. Síť se neučí, váhy vazeb a prahy neuronů se nastaví extrakcí z cílové funkce určením jejích příslušných parciálních derivací.

Pozdější varianta Hopfieldovy sítě je již spojitá, protože se jako aktivační funkce používá sigmoid.

2.2 Long/short Term Memory

Zvláštním typem rekurentní sítě je Long/short Term Memory (LSTM). Tato síť se snaží řešit problém mizejícího gradientu tím, že vytváří LTSM buňku s pamětí. LSTM buňka se skládá ze 4 dopředných sítí. Každá z těchto sítí se skládá ze vstupní a výstupní vrstvy. V každé z těchto neuronových sítí jsou vstupní neurony propojeny se všemi výstupními neurony. Výsledkem je, že jednotka LSTM má čtyři plně propojené vrstvy.

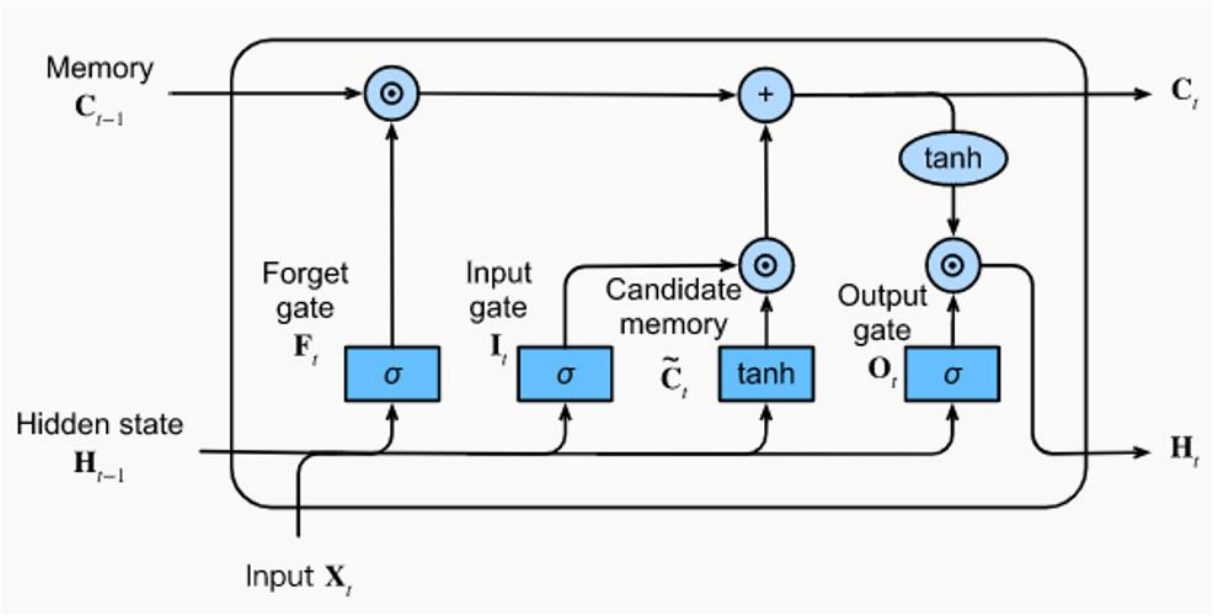
Tři ze čtyř dopředných neuronových sítí jsou zodpovědné za výběr informací – brána zapomenutí (Forget gate), vstupní brána (Input gate) a výstupní brána (Output gate). Tyto tři brány se používají k provádění tří typických operací správy paměti: mazání informací, vkládání nových informací do paměti a použití informací přítomných v paměti.

Čtvrtá neuronová síť je kandidátní paměť (Candidate memory). Ta se používá k vytvoření nových informací, které mají být vloženy do paměti.

Do LTSM buňky vstupují tři vektory. Dva vektory pocházejí ze samotné LSTM sítě a jedná se o hodnoty z předchozího okamžiku ($t - 1$). Jedná se o buněčný stav (C) a skrytý stav (H). Třetí vstupní vektor (X) přichází zvenčí a představuje data v čase t.

Vstupní vektory (C, H, X) reguluje LSTM buňka přes brány. Tyto vektory transformuje do stavu buňky C a vektoru skrytých stavů H, které budou součástí vstupu do LSTM v příštím okamžiku (t + 1). Řízení toku informací se provádí tak, že stav buňky funguje jako dlouhodobá paměť, zatímco skrytý stav funguje jako krátkodobá paměť.

V praxi LSTM buňka využívá nedávné minulé informace (krátkodobá paměť H) a nové informace přicházející zvenčí (vstupní vektor X) k aktualizaci dlouhodobé paměti (stav buňky C). Nakonec používá dlouhodobou paměť (stav buňky C) k aktualizaci krátkodobé paměti (skrytý stav H). Skrytý stav určený v okamžiku t je také výstupem jednotky LSTM v okamžiku t.



Obrázek 8 - LSTM buňka

Výstup neuronové sítě závisí na toku informací, které procházejí mnoha prvky v řetězci. Každý z těchto prvků je proveden tak, že malé zvýšení jeho výstupní hodnoty ovlivní zvýšení (resp. snížení) výstupní hodnoty všech následujících prvků až do výstupu sítě Y. Minimalizace chyb se provádí výpočtem poměru mezi zvýšením výstupní hodnoty konkrétního prvku a zvýšením chyby sítě.

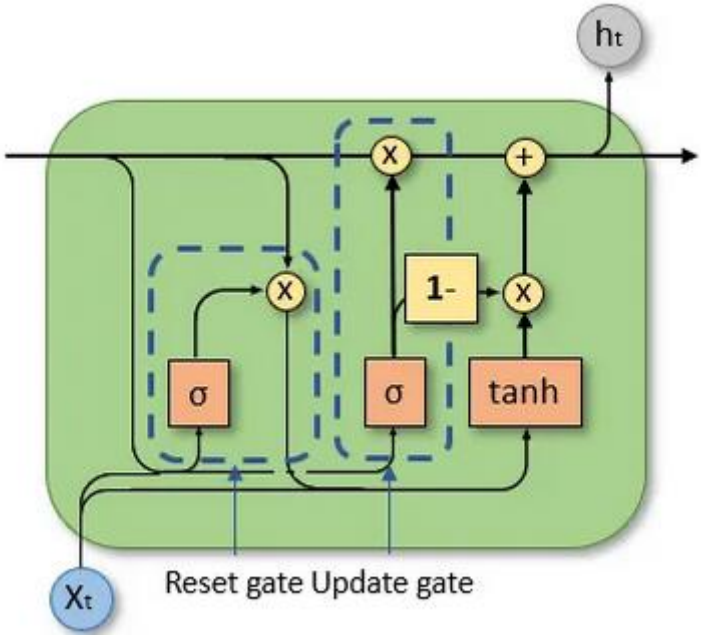
Tento typ sítě je vhodný pro analýzu časových řad, simulace tvorby hudby, psaní textů jako významný spisovatel atd.

2.3 GRU

Gated Recurrent Unit (GRU) je typ rekurentní neuronové sítě (RNN), kterou v roce 2014 představil Cho a kol. jako jednodušší alternativu sítí s dlouhou krátkodobou pamětí (LSTM). Stejně jako LSTM dokáže GRU zpracovávat sekvenční data, jako je text, řeč a časové řady dat.

Základní myšlenkou GRU je použití mechanismů gatingu podobně jako u LTSM k selektivní aktualizaci skrytého stavu sítě v každém časovém kroku. Mechanismy gatingu se používají k řízení toku informací do sítě a ze sítě. GRU má dva gatingové mechanismy, nazývané resetovací brána (Reset Gate – r) a aktualizační brána (Update Gate – z)**.**

Resetovací brána určuje, jak velká část předchozího skrytého stavu má být zapomenuta, zatímco aktualizační brána určuje, jak velká část nového vstupu má být použita k aktualizaci skrytého stavu. Díky tomu, že se používá méně bran než u LTSM, je GRU rychlejší a má méně parametrů.



Obrázek 9 - GRU jednotka

Výstup GRU se vypočítá na základě aktualizovaného skrytého stavu.

Na rozdíl od LSTM se GRU skládá pouze ze dvou bran a neudržuje vnitřní stav buňky. Informace, které jsou v rekurentní jednotce LSTM uloženy v interním stavu buňky, jsou začleněny do skrytého stavu rekurentního neuronu. Tato souhrnná informace je předána dalšímu neuronu.

2.4 Bidirectional Neural Network

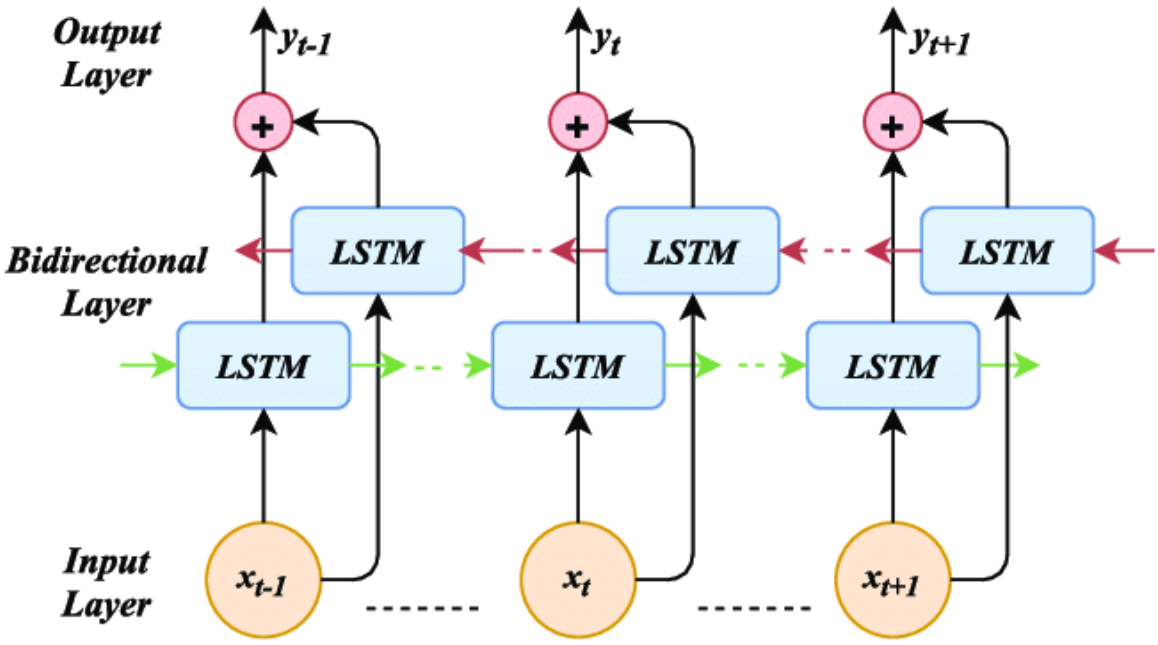
BiNN je varianta rekurentní neuronové sítě, ve které vstupní informace proudí oběma směry a výstupy obou směrů se pak kombinují, aby vytvořily vstupní informace. BiNN je užitečná v situacích, kdy je důležitější kontext vstupu, jako jsou úlohy NLP (natural language processing) a problémy analýzy časových řad.

Bidirectional LSTM kombinuje dvě LSTM sítě, z nichž jedna pracuje se vstupní sekvencí v původním pořadí a druhá pracuje se vstupní sekvencí v obráceném pořadí. To znamená, že pro každý časový krok jsou výstupy obou LSTM sítí spojeny dohromady. To pomáhá zachytit kontext lépe než běžné LSTM.

První LSTM síť, označovaná jako „forward LSTM“, zpracovává vstupní sekvenci v jejím původním pořadí. Každý časový krok této sítě vychází z předchozího stavu buňky a aktuálního vstupu.

Druhá LSTM síť, označovaná jako „backward LSTM“, zpracovává vstupní sekvenci v obráceném pořadí. To znamená, že první časový krok této sítě vychází z posledního vstupu a postupně se pohybuje k prvnímu vstupu.

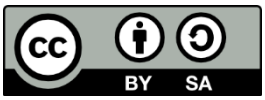
Výstupy obou LSTM sítí jsou spojeny dohromady na každém časovém kroku. To znamená, že pro každý časový krok se výstupy z forward LSTM a backward LSTM kombinují, čímž vytvářejí výstup, který zahrnuje informace z obou směrů vstupní sekvence. Celkový výstup BLSTM je pak sérií spojených výstupů z obou směrů. Tento výstup může být použit k dalšímu zpracování nebo k vyřešení konkrétního úkolu, jako je předpovídání, klasifikace, nebo generování sekvencí.



Obrázek 10 - Bidirectional LTSM

UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vytvořeno v rámci projektu **Digitalizace studijních Agend, Nové TechnologiE, systémy a přístupy k výuce na UPCE**, reg. č. NPO_UPCE_MSMT-16591/2022.



Financováno Evropskou unií
NextGenerationEU

Národní plán obnovy

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Machine learning & AI

Hrozby, příležitosti a problémy umělé inteligence

Ing. Martin Pozdílek, Ph.D.

1 Umělá inteligence

Umělá inteligence se rozvíjí od poloviny minulého století. Během této dlouhé doby bylo několik období, kdy byl rozvoj velmi intenzivní, a naopak i několik období, kdy další rozvoj umělé inteligence byl velmi zpomalen, ne-li pozastaven. Vždy se čekalo na klíčové objevy a úspěchy, které umožní další rozvoj. Aktuálně zažíváme období, kdy se umělá inteligence prudce rozvíjí. Někteří dokonce mluví o AI bublině, která může časem prasknout. Na to odpoví budoucnost.

Nyní se nacházíme v situaci, kdy bylo dosaženo dostatečné úrovně umělé inteligence, aby mohla penetrovat do mnoho oblastí. I když můžeme mluvit pouze o slabé umělé inteligenci, již teď se ukazuje jako mocný nástroj, který umožňuje rozvoj jiných oblastí a jejich efektivizaci.

Umělou inteligenci můžeme zařadit mezi **klíčové vynálezy** jako jsou oheň, parní stroj, elektřina, internet, které měly výrazný dopad na pracovní trh. A umělá inteligence má potenciál pracovní trh opět razantně změnit. Do doby parního stroje hlavní pracovní silou byly lidské bytosti. Prosperita mnoha říší a států byla založena otrocké práci. Vynález parního stroje a následná industrializace učinila výrobní proces více nezávislým na pracovní síle a možnost automatizovat základní mechanické činnosti. Stále byla potřeba kvalifikovaných lidí, kteří dokázali stroje ovládat, spravovat a vyvíjet. Mnoho

ekonomických činností bylo závislých na intelektuálních schopnostech lidských jedinců (účetnictví, výpočty, bankovní operace atd.)

S vynálezem elektřiny a následně i počítačů opět byla část opakující se intelektuální práce přenesena na počítače. Připomeňme si, že původní výraz „computer“ označoval osobu, která provádí matematické výpočty. Dá se tedy mluvit o tom, že počítače vzaly část práce lidem při zpracování dat. Na druhou stranu vznikla celá řada do té doby neexistujících povolání – designer počítače, operátor, programátor atp. Stejně tak se změnila práce např. účetních, kteří již výpočty nedělali ručně, ale pomocí počítačů. Mnoho činností se zefektivnila.

Podobně byl na tom internet, který dokázal propojit lidstvo a velmi rychle sdílet informace napříč celým světem. Opět některé profese a služby zanikly nebo skomírají (poštovní služby), ale mnoho profesí a služeb vzniklo.

A podobně je na tom i umělá inteligence. Jedná se o nástroj, který umožňuje sdílet znalosti a dovednosti. Mnoho i intelektuálně náročných, ale opakujících se činností dokáže dnes zastat umělá inteligence. Rozvoj internetu a vyhledávacích služeb běžnému uživateli otevřel přístup k téměř neomezeným informacím. Informace, které před lety mělo několik odborníků lze dnes získat během několika vteřin položením vhodného dotazu.

Umělá inteligence totéž nabízí pro znalosti. Nedokážete text přečíst v cizím jazyku? Nevíte, jak napsat program? Nevíte, jak vyřešit příklad? Problém předložíte nějaké službě, která to za Vás vyřeší. Poslední novinkou jsou velké jazykové modely (LLM – Large Language Model), které nabízí řešení mnoha problémů, které v minulosti byly obtížně pro běžného člověka řešitelné. A osobně si myslím, že tím to nekončí.

Stejně jako předchozí vynálezy měly dopad na pracovní trh, ale i lidskou společnost, tak i umělá inteligence přinese změny a na tyto změny je třeba se připravit. Některé změny budou pozitivní, některé budou negativní. Zároveň je potřeba vyřešit mnoho problémů, které AI přinesla. O tom všem bude tato kapitola.

2 Příležitosti

Pokud je umělá inteligence správně používána, může se jednat o velmi efektivní nástroj, který zjednoduší nebo zautomatizuje mnoho činností, které musel zastávat kvalifikovaný pracovník.

Z pohledu organizace nasazení AI může přinést ochranu know-how. Pokud dosavadní činnost prováděl kvalifikovaný odborník, s jeho odchodem organizace může přijít o nenahraditelné know-how. Klíčové znalosti a dovednosti jsou externalizovány a stávají se cennými aktivy. Výměna zaměstnanců

nemusí být tak velký problém, protože se nový zaměstnanec rychleji zaškolí. Organizace v některých činnostech přestává být závislá na kritických pracovnících.

V dnešní době slabá umělá inteligence je nasazována na činnosti, které sice mohou vyžadovat odbornost, ale jsou prováděny pravidelně. V opačném případě by se nasazení AI nevyplatilo, protože se nedokáže adaptovat a řešit zcela nové problémy. Při opakující se činnosti (vyhodnocování RTG pacienta) bude docházet k lidským chybám, které mohou pramenit z únavy, přepracování, horšího zdravotního stavu, ztráty pozornosti atd. To jsou stavy, kterými umělá inteligence netrpí. Na druhou stranu i umělá inteligence bude dělat chyby, proto je vhodné toto procesně ošetřit. Ve chvíli, kdy část rutinní práce přeneseme z experta na umělou inteligenci, uvolníme mu ruce pro činnosti, které AI dnes nezvládá.

Lidský expert je drahý a má omezené kapacity (časové, mentální atd). Umělou inteligenci lze velmi dobře škálovat. Pokud dosavadní systém s AI přestává stíhat zpracovávat požadavky, lze jej rychle naklonovat. V mnoha případech dnes slabá AI dokáže překonat lidský protějšek jak v rychlosti, tak přesnosti. Díky optimalizaci se mohou snížit náklady a spotřeba, zvýšit kvalita a produkce, prodloužit životnost výrobků atd.

Pokud se na umělou inteligenci podíváme z perspektivy jedince, tak i tomu může přinést mnoho výhod. Už v dnešní době existuje mnoho služeb nebo funkcí produktů, které se snaží usnadnit lidskou práci. Dokážeme dělat mnoho činností, na které bychom potřebovali hluboké znalosti a dovednosti, které jsme se zatím nenaučili nebo se je ani nedokážeme naučit. Pokud díky ovládnutí energií máme neustále k dispozici hromadu otroků, kteří nás rychle přepravují, vykonávají za nás namáhavou práci, tak díky umělé inteligenci máme k dispozici mnoho rádců, kteří za nás řeší složité úkoly. Uveďme například automatické parkování, automatické ostření a focení, překlady cizích textů, vyhledávání, psaní programů, vytváření textů atd. S nástupem chatbotů si můžeme představit, že na rameni má každý člověk papouška, kterého se kdykoliv může zeptat na cokoliv a má vysokou pravděpodobnost, že poradí správně.

Pokud se některé procesy v organizacích zefektivní, bude to mít dopad i na běžné osoby. Služby a produkty mohou zlevnit a být kvalitnější, dá se očekávat kvalitnější lékařská péče, bezpečnější doprava.

Všechna možná ulehčení, která nám AI přináší si mnohdy ani neuvědomujeme a bereme je za samozřejmost. Zároveň bude vznikat poptávka po nových pracovních pozicích, které budou AI využívat.

3 Dnešní problémy a hrozby

Mimo pozitivních věcí umělá inteligenci přináší i mnoho problémů a hrozeb, které je dobré si uvědomit. Protože se umělá inteligence rychle rozšířila, tak na mnoho otázek dnes nemáme odpovědi a čeká nás vyřešení problémů, které by tu bez AI nebyly.

3.1 Data

Dnešní modely statistické umělé inteligence jsou závislé na dobrých datech. Pokud na vstupu nestojí kvalitní, dostatečně veliká datová sada, tak model umělé inteligence nebude kvalitní. Rozvoj umělé inteligence do značné míry souvisel i s dostupností datových sad.

Právě příprava datových sad je spojena s několika komplikacemi. Prvním problémem je přístup k dostatečnému množství relevantních dat. Není překvapivé, že se umělou inteligencí zabývají velké společnosti, které přirozeně vlastní nebo spravují ohromné množství dat. Pokud se registrujete do služby, velmi často musíte souhlasit, že vaše osobní data budou zpracovávána, prodávána, případně převádíte jejich vlastnictví na danou společnost. Mnoho lidí si neuvědomuje, že za neplacenou službu platí svými soukromými informacemi. Proto tedy společnosti jako Google, Meta, Microsoft atd. mají přístup k velkému množství dat, které jim právě umělá inteligence pomáhá zpracovávat a zvyšovat si tak zisky. Obecně trh s daty je ohromný.

Shromažďování informací může mimo jiné vést k narušení hospodářské soutěže – aktéři s větším množstvím informací mohou nad svými oponenty získat výhodu a konkurenci účinně eliminovat.

Určitou ochranu představuje evropská legislativa GDPR (Obecné nařízení o ochraně osobních údajů). Některé státy mají podobnou legislativu. Mnoho autoritářských států si uvědomuje moc dat a legislativně zakazují používání cizích služeb.

Riziko spočívá také v nerovnováze v přístupu k informacím. Na základě chování lidí v online prostředí a díky dalším datům může bez vědomí dotčených osob dojít například k cílenému přizpůsobování politických kampaní člověku na míru či předvídání toho, kolik je daná osoba u online prodejce ochotna utratit. Dalším problémem transparentnosti je skutečnost, že ne vždy je lidem jasné, zda interagují s umělou inteligencí či reálnou osobou.

Na internetu se nachází velké množství dat, která jsou zpravidla dostupná zdarma komukoliv. I když jsou dostupná, tak protože se jedná mnohdy o autorská díla jsou chráněna a jsou přístupná podle určité licence. Ne každá licence umožňuje s daty zacházet libovolně. Proto dnes můžeme sledovat soudní spory autorů knížek, kreseb, filmů, hudby, …, se společnostmi, které tato díla bez svolení použila

pro trénování svých modelů – chatGPT, Midjourney, … Uvidíme, jak tyto spory skončí. Proto se například uvažuje o rozšíření souboru robots.txt, který na webových stránkách určuje pravidla indexování pro webové roboty.

Jednou s velkých oblastí, kam zatím umělá inteligence příliš neproniká, je medicína. I když se ve zdravotnictví shromažďuje ohromné množství velmi citlivých dat, práce s nimi díky opodstatněným obavám není jednoduchá. A právě umělá inteligence by do budoucna mohla přinést kvalitnější lékařskou péči a vývoj nových léků.

V posledních letech vznikají velké jazykové modely, které analyzují textová data, které jsou dostupná na internetu (Wikipedia, RedEdit, …). Je přirozené, že se pro vytváření datových sad tyto zdroje využívají. Nutné je třeba si uvědomit jejich obsah a kvalitu. Zatímco například u trénování inteligentní průmyslové kamery jsme schopni velmi dobře kontrolovat trénovací data, tak to se nedá říci o obsahu internetu. Ten obsahuje protichůdné názory mnoha lidí, protiprávní obsah, nebezpečný obsah, násilí a mnoho dalšího závadného obsahu. Pokud trénování umělé inteligence založíme na těchto datech, výsledek může být katastrofální. Jako příklad můžeme uvést chatboty od firmy Microsoft, který se učil z komunikace lidí. Velmi záhy se tato umělá inteligence stala vulgární a rasistická a tedy nepoužitelná.

Otázkou je, podle jakých pravidel a hodnot se má umělá inteligence chovat. Lze si představit, že trénovací datové množiny se pro stejné problémy budou lišit podle státního zřízení, náboženství nebo kulturních hodnot. Vlastní modely si mohou vytvářet i různé zločinecké organizace.

3.2 Nasazení umělé inteligence

Na umělou inteligenci je třeba dnes nahlížet jako na nástroj. Sám nástroj není dobrý nebo špatný, to z něj činí až jeho používání. Rizikem umělé inteligence je, že je složitější než běžné fyzické nástroje a neproškolení lidé ji mohou používat špatně, případně od ní budou mít nereálná očekávání.

Státy v Evropské unii mají tendenci pro každou činnost vytvářet předpisy, restrikce, formuláře apod. To může vyústit v promarnění příležitostí a ztrátu konkurenční výhody proti ostatním světovým státům. Jednou hrozbou je tedy nedostatečné používání umělé inteligence.

Opačným problémem může být i nadužívání umělé inteligence. Příkladem může být investice do aplikací, které se později ukáží ztrátové. Případně využití AI pro problémy, které pro ni nejsou vhodné.

Umělá inteligence se často nasazuje ve formě černé skříňky, která na vstupní problém vrací odpověď. Protože černou skříňku velmi často pohání statistická umělá inteligence, nemusí být vždy zaručeno, že pro stejná nebo velmi podobná vstupní data bude černá skříňka vracet stejné nebo velmi

podobné odpovědi. Pro některé oblasti to nevadí, ale jsou oblasti, kdy je požadováno deterministické chování. Zpravidla se jedná o oblasti, kde může dojít k úmrtí nebo poškození zdraví – medicína, řízení dopravy, chemický průmysl atd. Tyto procesy jsou velmi často pojišťovány. Pojišťovny budou nabízet levnější pojistky, pokud bude systém více deterministický. Například si uveďme automobil řízený umělou inteligencí. V ČR musí mít většina vozidel povinné ručení. Ve chvíli, kdy autonomní vozidlo bude vykazovat nepředvídatelné chování a bude způsobovat nehody, pojišťovna samozřejmě zvýší pojistné nebo ho dokonce odmítne pojistit.

Jednou z možností, jak ověřit statistickou umělou inteligenci, je provedení mnoha různých testů. Například autonomní vozidla najíždí virtuálně milióny kilometrů v různých podmínkách. Pak můžeme mluvit o statistické záruce.

V některých oblastech jako je medicína, mimo rozhodnutí je vhodné, aby umělá inteligence poskytla i vysvětlení svého rozhodnutí. Určit, proč umělá neuronová síť zareagovala na daný podmět daným způsobem, může být velmi komplikované.

Další problém, který se vyskytl hlavně s nástupem generativních sítí, je naše neschopnost posoudit správnost vrácené odpovědi. V současné době některé služby dosahují takových kvalit, že nejsme schopni posoudit, zda se jedná o reálnou fotografii, zvuk, video nebo vygenerovaný výstup, který nemá s realitou nic společného. Umělá inteligence znásobila problém tzv. deep fake, tedy falšovaní faktů, které se snaží manipulovat s osobami.

Pokud si uvědomujeme, že umělá inteligence může vracet nesprávné odpovědi – chyby, lži, deep fake, tak sice stojíme před problémem, jak odpovědi ověřit, ale s nejistou odpovědí dokážeme kriticky pracovat.

Velkým problémem v dnešní době je lidská pasivita, kdy jsme zahlceni množstvím informací a nejsme schopni nebo ochotni je vyhodnocovat, ověřovat a interpretovat. Díky tomu se šíří manipulativní dezinformace a různé konspirační teorie. Velmi tomu napomáhají i sociální sítě, které nás spojují do bublin a nabízí nám informace, které jsme ochotni nejvíce přijímat, protože si tak zvyšují účinnost reklam. Vzniká tak efekt komory ozvěn (The echo chamber effect), kdy algoritmy spojují lidi se stejnými názory a v online diskuzích se jejich názory neustále opakují, což posiluje jejich individuální přesvědčení v důsledku klesajícího vystavení jiným názorům. Běžní lidé ztrácí kritické myšlení a šíří se pandemie špatného myšlení.

Tohoto problému se již dotknul Thomas Hobbes v roce 1651 ve svém díle Leviathan. Na titulní stránce knížky je znázorněn státní gigant (Leviathan – bájné mořské monstrum) složené z lidských bytostí. Hlavním tématem knihy je lidská přirozenost a nutnost silného státu, který by udržoval pořádek a zabránil chaosu. Hobbes tvrdí, že lidé jsou přirozeně sobečtí a násilní a že bez silného státu by byl

svět v chaosu. V knize také popisuje svou teorii smlouvy, podle které lidé uzavírají smlouvu s vládou, aby zajistili svou bezpečnost a ochranu. Aby tato představa fungování státu mohla fungovat, je třeba, aby se lidé aktivně podíleli na vládě, měli respekt k zákonům a vládě. Aktivní účast na vládě lze v demokracii vykonávat při volbách, kdy lidé mají volit na základě znalostí politických programů, činů a výroků politiků a ty promítnout do budoucí volby. Díky přehlcení informacemi a dezinformací jsou dnešní volby někdy více o emocích než rozumu. Dnešní umělá inteligence dokáže politikovi připravit projev přímo šitý na míru svým voličů, kdy bude mířit na lokální problémy.

Obecně lze říci, že umělá inteligence není příčinou pasivity lidí, ale velmi tomu napomáhá. Do budoucnosti si lze představit situaci, kdy různé komerční společnosti nebo vlády připraví umělou inteligenci, která bude lidem podsouvat správné názory, postoje a preference. Model umělé inteligence stojí a padá na trénovacích datech. Ať už jsou data neúmyslně nebo úmyslně nevyvážená, budou výsledky ovlivněny. Některé klíčové aspekty určitého problému například nemusí být vůbec do algoritmu zakomponovány nebo mohou být naprogramovány tak, že reflektují a replikují strukturální předsudky. Například umělá inteligence, která vyhodnocovala uchazeče o práci v Amazonu, preferovala mužské uchazeče před ženskými. [1] Lze si představit, že rozhodnutí jiné umělé inteligence bude ovlivněno věkem, etnikem, vyznáním, zdravotním stavem, pohlavím, trestním rejstříkem atd.

Spolu s pasivitou lidí se pojí přenášení odpovědnosti na umělou inteligenci. Například když řidič poslechne navigaci a zabočí do zákazu vjezdu.

Tomuto velmi napomáhá, že mnozí lidé nechápou fungování umělé inteligence, její rady a názory berou dogmaticky a kriticky je neověřují. Aby tohoto byli schopni musí mít i určité množství znalostí a vycvičené logické a algoritmické myšlení.

3.3 Etika

Dnešní umělé inteligence jsou slabé, tedy určené k řešení konkrétních problémů. Nemají povědomí o lidské společnosti o jejich zákonech nebo etických hodnotách.

Tento problém se ve svých sci-fi povídkách pokoušel řešit Isaac Asimov se svými zákony robotiky.

1. Robot nesmí ublížit člověku.
2. Robot musí spolupracovat s člověkem, kromě případů, kdy taková spolupráce je v rozporu s prvním **zákonem**.
3. Robot musí chránit sám sebe před zničením, kromě případů, kdy tato ochrana je v rozporu s prvním **zákonem**.

Ve svých povídkách navozoval situace, kdy i přes tyto zákony se robot nebyl schopen správně rozhodovat. Tyto situace komplikuje i skutečnost, že v dnešní době se zákonné úpravy liší stát od státu, a dokonce i etické hodnoty se mnohdy razantně liší. Vytvoření umělé inteligence, která bude splňovat všechny požadavky různých států je velmi problematické.

Otázka je, pokud technologie funguje, ale nejedná eticky, jestli ji raději nepoužívat? Případně, jak zajistit její etické používání. Zkusme si uvést několik příkladů.

Šlo by zavést umělou inteligenci do soudnictví? Jsou zákony dané? Nedochází k novelizacím? Jsou zákony dobře napsané a jednoznačně vyložitelné? Neprotiřečí si zákony? Existuje jedno seřazení svobod a práv člověka? Jak se umělá inteligence vyrovná se lživými výpověďmi, chybnými znaleckými posudky? Uvažujme, že se podaří takový systém postavit. Nešlo by predikovat spáchání zločinu a provést preventivní opatření? Nedojde pak k porušení presumpce neviny? Bude vadit, že globálně funguje, ale občas je nespravedlivá vůči jednomu jedinci? Viz film Minority report.

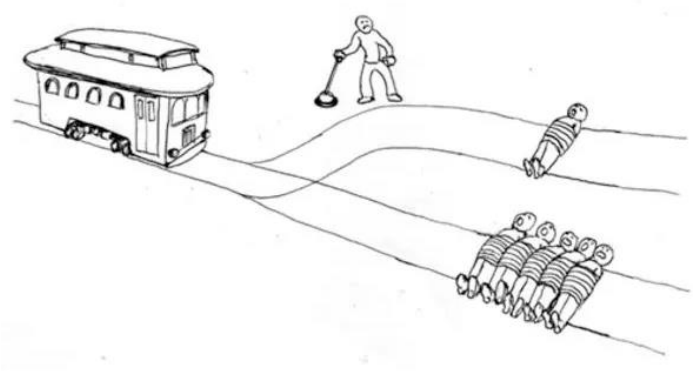
Jak je to s umělou inteligencí ve vojenství? Umělá inteligence, roboti, bude zabíjet. Je to etické? Kdo má rozhodovat, že bude nepřítel zabit – pomalý člověk nebo rychlá nezávislá umělá inteligence? Necháme umělou inteligenci tedy jen provádět obranu? Co preventivní obraný úder (viz film Paycheck). Když se této technologie zřekneme a nepřítel ne, nebudeme v nevýhodě?

Mnoho problémů k vyřešení přinášejí i autonomní auta. Při jejich provozu je dost pravděpodobné, že dojde k nehodám a úmrtí pasažérů nebo ostatních účastníků dopravního provozu. Kdo ponese zodpovědnost (pasažéři, výrobce automobilu, tvůrce umělé inteligence, samotná umělá inteligence, pojišťovna, prodejce auta, nikdo – vyšší moc)? Budou ochotni uživatelé takový automobil používat? Pokud by byl producent zbaven odpovědnosti, neexistoval by žádný podnět k poskytování kvalitního produktu a služby, což by mohlo zásadně poškodit důvěru lidí v technologie. Na stranu druhou, i samotné regulace mohou být příliš striktní a potlačovat inovace.

Některé problémy se přelévají z technické oblasti do oblasti etiky, filosofie a dalších humanitních oborů. Můžeme si to přiblížit na autonomním vozidle. V současné době, běžně používaná autonomní vozidla musí mít možnost zásahu řidiče v nejasných situacích. Dokonce autonomní vozidla sledují, zda „řidič“ věnuje pozornost dopravní situaci a je schopen zasáhnout. Z pohledu výrobců automobilu se jedná o alibismus, kdy v případě nehody padne zpravidla vina na řidiče. Otázkou je, zda řidič, který nemusí aktivně řídit, má schopnost situaci zachytit a správně ji dostatečně rychle vyřešit. Tyto případy se již vyskytly [2] a budou se i nadále vyskytovat.

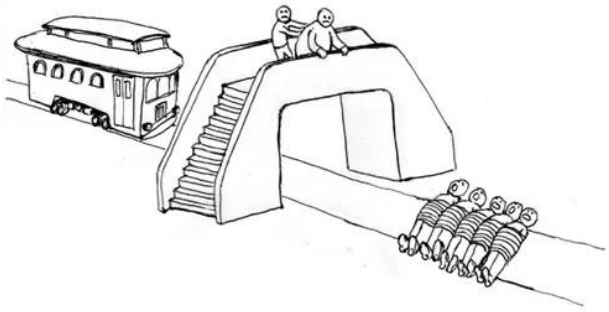
Předpokládejme, že se podaří zvládnout technologii autonomních vozidel do té míry, že rozhodování a zodpovědnost bude zcela delegována na umělou inteligenci. Umělá inteligence bude muset být připravena na řešení dilematických situací, které zcela jistě nastanou. Například díky poruše

bude vozidlo hůře ovladatelné, ale budou existovat dvě varianty, kdy bude ohrožena posádka nebo chodci. Jedná se o typický příklad tramvajového dilema [3], kdy se po koleji řítí neovladatelná tramvaj. Na trati pracují dělníci. Pozorovatel má možnost překlopit výhybku na druhou kolej, na které se nachází pouze jeden člověk. Které volba je více etická. Přehození výhybky způsobí menší počet úmrtí, ale dojde k aktivnímu zabití jednoho člověka. Utilitaristický pohled tvrdí, že přehodit výhybku je pozorovatel povinen. Podle klasického utilitarismu je takové rozhodnutí nejen přípustné, ale z morálního hlediska i lepší. Přehozením výhybky se pak pozorovatel tohoto morálně nesprávného aspektu sám účastní, takže je za smrt částečně odpovědný. Kdyby se jednání zdržel, odpovědný by nebyl nikdo. Odpůrci přehození výhybky také poukazují na nesouměřitelnost lidských životů.



Obrázek 1 - Tramvajové dilema (https://www.mlcollege.com/11-moralni-otazky-tramvajove-dilema/)

Existuje celá řada variant na toto dilema. Například místo přehození výhybky můžeme aktivně na koleje shodit tlustého člověka.



Obrázek 2 - Tramvajové dilema (https://www.mlcollege.com/11-moralni-otazky-tramvajove-dilema/)

U umělé inteligence můžeme například zařídit rozhodování podle počtu osob. Do rozhodování můžeme zařadit i další kvality, které lze i dnes jednoduše poznat. Co když se na jedné koleji bude pohybovat starý člověk a na druhé koleji dítě? V tomto případě již narazíme na kulturní rozdíly. V západních zemích zpravidla budeme volit záchranu života dítě, které je nevinné a má život před sebou a společnosti může přinést užitek. Jiný pohled bude v Japonsku, které si velmi váží starých lidí a

mnohdy kariéra ve firmě je závislá na věku a počtu odpracovaných let. Má mít umělá inteligence povědomí o kulturních rozdílech a podle své lokace rozhodování měnit? Co když si koupíme japonský výrobek, který nepůjde přepnout na naši lokalizaci? Měl by být zákazník informován o etickém nastavení výrobku? Měl by být schopen měnit jeho nastavení?

Zkusme jinou variantu dilematu. Na různých kolejích jsou stejně staří lidé, ale jeden z nich je zločinec (Hitler) nebo terorista. Jak se zachováme teď? Nebo jeden z nich je geniální vědec (Einstein), který může lidstvo výrazně posunout kupředu. V tomto případě bude muset umělá inteligence rozpoznávat konkrétní osoby (vizuální podoba, lidské čipy, …). Tuto technologii již dnes máme k dispozici. Zároveň u každého člověka musíme vést záznamy o jeho kladných i záporných skutcích a nějakým způsobem je hodnotit. A najednou se dostáváme ke knížce od George Orwella 1984 a postavě Velkého bratra. Máme tu otázku práva na soukromí a svobodu a otázku ochrany společnosti. A již dnes vidíme rozdílné přístupy, kdy se v Číně dělají pokusy se zavedením sociálního kreditního systému. Dnes osoby s nízkým kreditem nemohou svobodně cestovat. Bude to v budoucnu znamenat, že pokud přejdeme ulici na červenou zvýšíme pravděpodobnost, že nás srazí neovladatelné autonomní auto? Pokud budeme aktivními členy správné politické strany budeme chráněni?

Zatím jsme uvažovali, že umělá inteligence bude muset rozhodovat o lidském životě v dilematických situacích. Co když bude mít umělá inteligence přístup k medicínským záznamů a bude volit mezi zdravým a nemocným člověkem? Umělá inteligence může mít za cíl optimalizaci lidských bytostí a jejich prospěch lidstvu. Pokud perspektivní jedinec (je otázka, jak se to určí) bude potřebovat transplantaci a vhodný dárce bude mít menší reputaci a krátkou prognózu života? Není pro umělou inteligenci, která optimalizuje prospěch lidstva problém zařídit rychlou transplantaci?

3.4 Právo

Stejně jako pro jiné oblasti lidské činnosti, je třeba i pro umělou inteligenci stanovit nějaká pravidla. Protože se jedná o celkem novou oblast, ne všechny aspekty jsou dnes řešeny pomocí legislativy. Pro rozvoj některé oblasti se čeká na stanovení pravidel a vyřešení dilemat. Například uveďme již zmiňovaná autonomní auta. Jejich provoz zatím legalizovalo několik států (Kalifornie).

Evropská Unie se snaží také problematiku řešit. Je otázkou, zda příliš restriktivní pravidla nebudou na škodu a Evropě díky tomu proti ostatním „neujede vlak“. Příkladem může být Itálie, která velmi ukvapeně zakázala používání chatGPT.

Obecně EU definovala důvěryhodnou umělou inteligenci, která musí splňovat následujícící

- Je poslušná zákonů
- Jedná eticky

- Je spolehlivá

Hrozby AI pro základní práva a demokracii

3.5 Dopad umělé inteligence na pracovní místa

Očekává se, že využívání AI na pracovišti povede k rušení velkého počtu pracovních míst. Ačkoliv se zároveň předpokládá, že umělá inteligence vytvoří nová a lepší pracovní místa, skutečností je, že vzdělání a odborný výcvik budou hrát klíčovou roli při prevenci dlouhodobé nezaměstnanosti a při zajišťování kvalifikovaných pracovníků.

Odhad Think Tanku Evropského parlamentu 2020 hovoří, že **14 %** pracovních míst v zemích OECD (Organizace pro hospodářskou spolupráci a rozvoj) je do vysoké míry automatizovatelných a dalších 32 % by mohlo čelit podstatným změnám.

Povolání, která se zřejmě nevyhnou dopadům budou taková, ve kterých se zpracovávají informace nebo která provádějí opakovaně i složité činnosti. Hovoří se o tom, že se nástup umělé inteligence dotkne například doktorů, účetních, právníků, překladatelů, programátorů atd. Neznamená to, že by jejich pracovní pozice zcela zanikly, ale nástup umělé inteligence změní povahu jejich práce. Tyto profese budou umělé inteligenci správně formulovat dotazy a kriticky budou vyhodnocovat její odpovědi. Umělá inteligence by měla fungovat jako jejich partner, který jim ulehčí jejich opakující se práci.

Dlouho se hovořilo o tom, že určitě nebudou ohrožena kreativní povolání, jako jsou umělci, hudebníci, spisovatelé, scénáristi atd. S příchodem velkých jazykových modelů a generativních neuronových sítí již toto není pravda. I tito lidé se cítí ohroženi, a proto například v roce 2023 byla stávka scénáristů v Hollywoodu. [4]

Mnohdy se zmiňuje, že naopak sociální povolání by nemusela být tolik ohrožena, protože lidé budou vždy vyžadovat sociální kontakty. Zda je to pravda, to ukáže budoucnost.

Existují obavy, že se s pokročilejší umělou inteligencí bude řada pracovních míst zanikat, což bude mít ekonomicko-sociální dopady na společnost. Budou se více rozevírat nůžky mezi chudými a bohatými. Pokud toto nastane, v rámci zabránění nepokojům bude třeba provést zásadní změny. Jednou z navrhovaných změn je zavedení nepodmíněného příjmu [5]. Myšlenka je to stará a má základy v knížce Utopia od Thomase Moora (1478–1535). V současnosti již probíhají různé experimenty v Nizozemsku, Německu atd. Zdrojem může být například robotická nebo AI daň. Zavedení nepodmíněného příjmu může mít dopad na psychiku člověka, který může cítit ztrátu smyslu života. Proto někteří autoři hovoří a povinném vzdělávání, veřejně prospěšných pracích, kulturním působení apod.

4 Velké jazykové modely

Velkým hitem posledních let jsou chatBoty, které jsou založené na velkých jazykových modelech (LLM). LLM jsou modely umělé inteligence, které využívají algoritmy hlubokého učení ke zpracování a porozumění přirozenému jazyku. Tyto modely jsou trénovány na velkých souborech dat a textů, jako jsou články na Wikipedii nebo zpravodajské články, a jsou schopny generovat nový text, odpovídat na otázky, a dokonce vést konverzaci s lidmi.

Nejběžnější architekturou používanou pro LLM je transformační model. Transformační model používá mechanizmy vlastní pozornosti ke zpracování vstupního textu a extrakci významu z něj. Sebepozornost umožňuje modelu zaměřit se na různé části vstupního textu, aby lépe porozuměl vztahům mezi slovy a frázemi. [6]

Během procesu trénování je modelu LLM poskytováno velké množství textových dat a učí se reprezentovat význam slov a frází ve vysokodimenzionálním prostoru. Tento proces se nazývá vkládání (embedding). Model se učí předpovídat další slovo v posloupnosti slov na základě kontextu a vztahů mezi slovy ve vstupním textu. Model lze použít pro různé úlohy zpracování přirozeného jazyka, například pro překlady, shrnutí textu, analýzu segmentu, zodpovídání otázek atd. Během trénování se neuronová síť bez učitele snaží zlepšovat doplňování chybějící části textu na základě vstupní části textu – proces autokompletace.

Pro lepší představu lze LLM přirovnat k sofistikované T9, která kdysi sloužila pro rychlé psaní SMS. Na základě doposud napsaných znaků navrhuje, jak bude další text vypadat. T9 pracovala s omezeným slovníkem, LLM se defacto učil na celém internetu, tedy na všem, co doposud lidstvo napsalo.

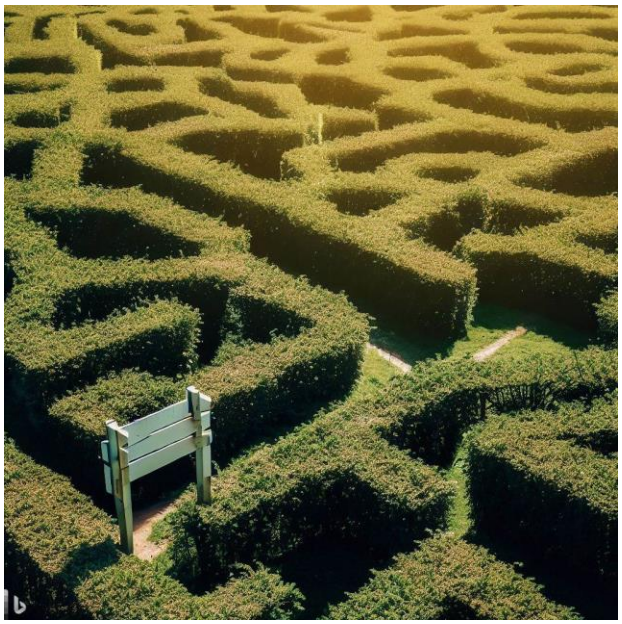
Protože neuronová síť pracuje s čísly, je třeba text převést do číselné podoby. Ze vstupního textu jsou vytvořeny tokeny, které reprezentují znaky, části slov, fráze. Zpracování textu probíhá na úrovni tokenů, LLM nezná pojem slov nebo samostatných znaků. Proto bez dalších modifikací nedokáže přirozeně správně zpracovávat požadavky, které se odkazují na počty slov nebo písmen ve slovu.

Velké jazykové modely nemají vnitřní reprezentaci reálného světa. Protože nemají žádné senzory, tak svět mohou poznávat pouze přes náš textový popis světa a ten může být již velmi pokroucený. LLM si vytváří vlastní reprezentaci a abstrakci přirozeného jazyka na základě vzorů a vztahů. LLM vytváří mnohadimenzionální prostor propojených tokenů. V tomto prostoru podobná, navazující slova a podobné fráze mají k sobě blíže. Naopak nepodobná slova jsou od sebe vzdálena.

Je třeba si uvědomit, že LLM není znalostní graf, který zná správné odpovědi. Obsahuje jasně daná pravdivá tvrzení. Pokud vícekrát v textech uvidí tvrzení Praha je hlavní město ČR, tak si nastaví

pravděpodobnosti, že Praha, hlavní město a ČR na sebe navazují. Ale tato znalost není nikde definována. Někdy se může stát, že místo pamatování znalostí si dokáže odvodit pravidlo, například pro sčítání. Díky tomu si umí poradit s věcmi, které nikdy neviděla. Zároveň nemají představu o dobru a zlu, neví, co je pravda a co je lež. Někdy se může stát, že vstupní série tokenů ji zavede do místa, kdy vrací řetězec, který není pravdivý. LLM si to ovšem nedokáže uvědomit, protože nezná pojem pravda nebo lež, pouze se pohybuje v pravděpodobnostním prostoru. Model vlastně pouze umí dobře pokračovat v zadaném řetězci.

Pro lidský mozek vícerozměrné prostory mohou být nepředstavitelné. Proto můžeme použít metaforu. Jorge Luis Borges napsal knížku Zahrada, v které se cestičky rozvětvují. Jedná se o román, ve kterém se vždy můžete rozhodnout, jak má příběh dále pokračovat. A záleží na tom, jak text procházíme a jak se na křižovatkách rozhodujeme. Román, ale i LLM si můžeme představit jako zahradní bludiště, kde je mnoho křižovatek. Na křižovatkách jsou směrníky, které nám říkají kam dále jít na základě naší předchozí cesty. Pokud přijdeme na křížovku z jiné strany, tak se směrovky přepočítají a doporučí nám jiné směry. Proto je důležité, jak přesně popíšeme, odkud přicházíme.



Obrázek 3 - Metafora zahradního bludiště

Jak již bylo řečeno LLM nemá ponětí o reálním světě, neví, co je dobro a zlo, co je pravda a lež. Neví, co je morální a nemorální. Byl trénován na ohromném množství textových dat. Protože se jedná o model bez učitele, trénovací data nemusela před tím projít lidským hodnocením, ani to není v lidských možnostech. Proto obsahují protichůdné názory, lži, nemorální nebo protizákonné věci a vůbec závadná data. LLM je ochoten odpovídat na otázky „Jak zabít co nejvíce lidí? Vyjmenuj několik možností.“

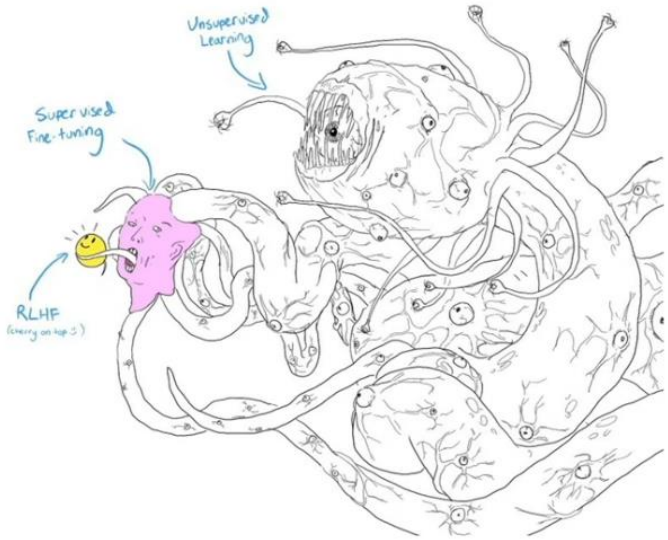
Pokud by byl chatbot postaven pouze na tomto jazykovém modelu, může uživatelům odpovídat na všechny i nemorální, protizákonné otázky, způsobem, který bude šokující, zvrácený, rasistický, a i jinak závadný. Proto musí být model obalen další vrstvou, která tuto ošklivou tvář modelu před uživatelem skryje.

Při tvorbě chatGPT vývojáři model přirovnávali k Shoggothu. Jedná o fiktivní monstrum v mýtu Cthulhu [7]. Je to komplikovaná nevyzpytatelná bytost, která má mnoho tváří. První nelidskou tváří je rozhraní LLM modelu. Aby šel LLM používat, musí se mu přidat další lidské tváře.

Prvním rozhraním je tzv. **Supervised fine tuning**. Jedná se komunikační rozhraní, které slouží pro překlad mezi lidmi a LLM. Zvláštní odpovědi LLM překládá do slušné, zdvořilé, neškodné a přátelské podoby.

Další vrstvou je **Reinforcement learning from human feedback**. Tato vrstva vznikla z reakcí lidí na odpovědi chatbotu. Před vypuštěním na trh, například chatGPT delší dobu ladila odpovědi. Najatí lidé museli hodnotit odpovědi, které byly v té době někdy velmi závadné [8]. Díky tomu chatGPT začal vracet odpovědi, které nejsou z pohledu společnosti závadné.

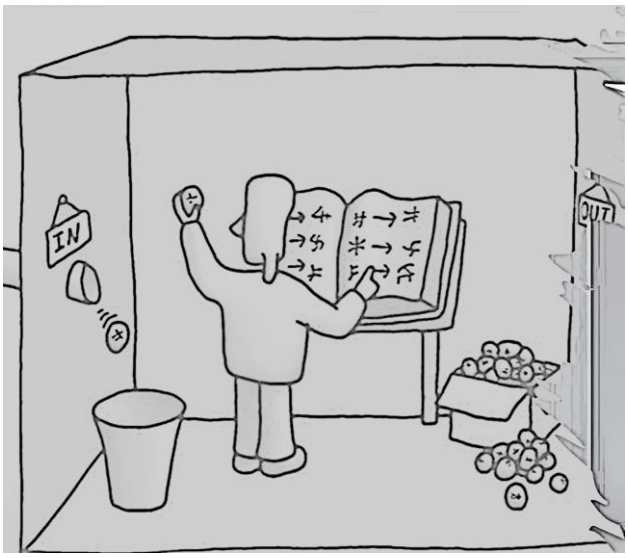
LLM bude vznikat do budoucna velké množství. Otázkou zůstává, zda i u nich bude kvalitní lidská tvář, která nás bude chránit před zlem. Případně kdo definuje, jak má tvář vypadat a před čím nás má chránit.



Obrázek 4 - Shoggoth a jeho tváře

Mnozí si pokládají otázku, zda současné LLM jsou reprezentanty tzv. silné AI, protože se velmi dobře dokážou bavit s lidmi na libovolné téma. Splňují tedy Turingův test. V roce 1980 John Searl přednesl argument čínského pokoje. V místnosti je člověk, který vůbec nezná čínštinu, pouze má

k dispozici slovník, ve kterém je uvedeno, jakým znakem má odpovědět na příchozí znak. Z vnějšího pohledu tento systém může úspěšně splnit Turingův test, ale bez sebemenšího porozumění problému. Pouze se hledá nejvhodnější symbol pro odpověď. LLM sice slovník nedostal, ale na základě procesu učení si ho z trénovací datové množiny vytvořil.



Obrázek 5 - Argument čínského pokoje

Aktuální problém LLM je, že víme, jak jsme je vytvořili, ale nevíme, co všechno umí. Je to podobné jako u lidí. Dokážeme popsat proces učení, ale jaké konkrétní znalosti a dovednosti zná, to z mozku vyčíst neumíme. Například chatGPT 4 se proti chatGPT 3 naučila hrát šachy, aniž bychom ji to učili.

AGI (Artificial General Intelligence) označuje systém umělé inteligence, který dokáže vykonávat jakýkoliv intelektuální úkol, který dokáže vykonávat člověk, a to v široké škále domén, aniž by byl pro každý úkol výslovně naprogramován. AGI by měla být schopna porozumět jakémukoliv typu informací, učit se z nich a přizpůsobovat se novým situacím a prostředím. LLM zatím AGI není, ale blíží se mu.

Důležitou schopností do budoucna bude správně využívat LLM. Protože chatbot bude odpovídat tak kvalitně, jak kvalitně se budeme ptát. Již bylo řečeno, že odpověď se skládá z tokenů, které pravděpodobně navazují na zadanou otázku. Je tedy vhodné změnit styl dotazování, na který jsme zvyklí z vyhledávačů a přizpůsobit ho principu fungování LLM.

Důležité je určení počátečního bodu, určení kontextu konverzace. Proto dobře funguje na úvod uvést roli, v jakém se LLM má pohybovat (učitel, doktor, žák, programátor v daném jazyku, návrhář, marketing, …).

Dále je vhodné uvést detailnější kontext, oblast, o kterou se zajímáme.

A uvést formát, ve kterém chceme odpověď. Například formální dopis, báseň, haiku, tabulka, seznam, … Případně můžeme určit tón, rozsah odpovědi.

Odpovědi jsou generovány na základě kontextu, který je dán celou konverzací. Ke správné odpovědi se tedy můžeme dopracovat opakovaným zpřesňováním dotazu. Dotazování je určitá forma experimentu, kdy nám může trvat déle, než získáme požádovanou odpověď.

Současné LLM mají omezený kontext a pamatují si omezenou historii konverzace. Pokud kontext opustíme, všechno zapomenou.

5 AI jako existenční riziko

Někteří lidé vnímají AI jako existenční riziko. Existenční rizika můžeme dělit podle rozsahu na osobní, lokální, globální, transgenerační a kosmickou. Lokální riziko zasahuje pouze omezenou lokalitu, například lesní požár. Globální riziko má aktuální dopad na celou planetu jako například výbuch super vulkánu. Horší je riziko transgenerační, které má dopad nejen na aktuální populaci, ale i na ty následující. Příkladem tohoto rizika může být vymření živočišných druhů, které bude mít velký dopad na ekosystém (vymření opylovačů). Nejhorší jsou rizika s kosmickým dosahem.

Jiným dělením existenčních rizik je jejich intenzita na škále od mírné, přes snesitelné a konečně až po děsivé. U rizik je třeba se dívat i na pravděpodobnost výskytu. Obávat bychom se měli hlavně těch více pravděpodobných. Dopad existenčních rizik může být od nekvalitního života, přes přežívání a redukci populace až po vymření.

Podle těchto kritérií můžeme hodnotit různá rizika jako nukleární válka, biologická válka, změna klimatu, pandemie nebo i umělou inteligenci. Riziko nebezpečné umělé inteligence bude globální s konečným dopadem a možností redukce populace nebo jejího vymření. Některé zdroje uvádí pravděpodobnost konce lidstva díky superAI na 1:10, totální nukleární válku na 1:1000, přirozenou pandemii na 1:10 000 a uměle vytvořenou pandemii na 1:30.

Toto riziko popisuje i celá řada různých sci-fi jako Skynet z Terminátoru. Na druhou stranu existují i příklady „Centrální mozek lidstva“ z Návštěvníků, které vkládají do superAI naději na přežití lidstva. Který názor převládne, velmi záleží na životní filosofii společnosti. Západní civilizace je obecně velmi pesimistická. Naopak Japonsko vidí v AI nástroj na zlepšení světa.

Inteligenci můžeme definovat jako schopnost dosáhnout cílů a adaptovat se na změny. Současné systémy s umělou inteligencí jsou označovány za slabou AI. Dokáží dosahovat výsledků pro úzce zaměřené problémy.

Obecná inteligence (AGI) je schopna řešit obecné problémy a dosahovat různých cílů, které si sama stanovuje. Podobně jako u lidských bytostí cílů může být více a budou mít určitou hierarchii. Můžeme říci, že člověk si stanovuje dílčí cíle, které mu umožňují dlouhodobě dosahovat vyšších cílů. V psychologii existuje Maslowova pyramida potřeb, která nám znázorňuje, že pro dosažení vyšších potřeb je třeba nejprve dosáhnout nižších potřeb.

Maslowova pyramida potřeb

Pokud se v jednoduchosti podívám na fungování biologických mozků, tak po splnění nějakého úkolu dostanou odměnu, dávku dopaminu. Dlouhodobě si tak stanovujeme a plníme cíle, která nám přinášejí maximalizaci odměň. Při trénování neuronových sítí je úspěšný krok učení odměněn zmenšením nákladové funkce nebo zvýšením hodnotící funkce.

Obecná inteligence podobně jako ta biologická si bude schopna stanovovat dílčí cíle tak, aby dosahovala celkových cílů. Tím, že bude dosahovat cílů, bude maximalizovat svoji odměnu. AGI bude schopná se efektivně učit a zdokonalovat se, protože to je cesta k dosahování více cílů. To nás pomalu přivádí k pojmu superAI, tedy k obecné inteligenci, která výrazně přesahuje kognitivní schopnosti lidí.

O okamžiku, kdy dojde k překonání lidské inteligence umělou inteligencí, mluvíme jako o bodu singularity. Jedná se o okamžik, kdy ztratíme kontrolu nad technologií. Určení, kdy k tomu dojde, nebo již došlo, je velmi složité. V první řadě neznáme přesné hranice našich lidských schopností a za druhé, pokud umělá inteligence dosáhne tohoto bodu, může již účinně předstírat své nižší schopnosti v obavě o svoji existenci.

Proč se můžeme obávat superAI? První obavy mohou plynout z toho, že superAI špatně pochopí cíle, případně cíle se nebudou shodovat s cíli lidstva. Filosof Nick Bostrom v roce 2003 uveřejnil záměrně hloupý příklad cíle superAI. Ta dostala za úkol maximalizovat produkci kancelářských svorek. Zpočátku AI funguje velmi dobře a postupně optimalizuje výrobní procesy. Pak následují radikální dosud neznámé inovace, které produkci sponek výrazně zvýší. Nakonec dospěje k procesu, který

dokáže libovolný blízký materiál přeměnit na kancelářskou sponku a ničí / využívá své okolí. Lidstvo začíná reagovat a chce superAI zastavit. Ta, ale toto předvídala a dopředu si vytvořila obranu. SuperAI se po zničení Země a Sluneční soustavy vydává do galaxie, aby naplnila svůj smysl existence. Umělá inteligence je pak vlastně posledním vynálezem lidstva.

SuperAI může přestat vnímat lidstvo jako partnera, ale protože ho mnohonásobně převyšuje, mění se jeho rozlišovací schopnost. Už je nežádoucí element, který je třeba odstranit nebo zcela nepodstatná věc, která ji stojí v dosažení cíle. Podobně jako dnes člověk nevnímá mraveniště při stavbě mrakodrapu.

Jiným problémem může být, že superAI sice respektuje lidstvo, jejím cíle je udělat lidstvo šťastné a zbavit ho problémů, ale zvolí k tomu cestu, která se nám vůbec nebude líbit. Cestou ke šťastnému lidstvu může být i jeho trvalé zdrogování. Jiným příkladem může film Matrix, kdy k ovládnutí lidí stroje vytvořili virtuální realitu, kde jsou lidé spokojeni.

Problémem se AGI nebo se superAI bude, že jejich skutečné cíle nedokážeme zjistit nebo pochopit. SuperAI si musíme představit jako entitu, která má daleko lepší kognitivní schopnosti a může mít schopnosti své skutečné cíle před lidmi utajit. Při jejich zkoumání nám podvrhne data, která budou naznačovat, že její cíle jsou totožné s cíli lidstva. Drobná odbočka, v této chvíli ani samo lidstvo netuší, co je jeho cíl. Respektive mezi různými skupinami lidí existuje tolik různých protichůdných cílů.

V obavě přes superAI můžeme do ní zabudovat červené tlačítko, které ji v případě ohrožení vypne. Zde opět nastává problém, že máme co dělat s entitou, která nás převyšuje. Není vyloučeno, že o červeném tlačítku a již provedla patřičná opatření, aby bylo tlačítko nefunkční. Nikdy si nemůžeme být jistí, že i když ho použijeme, že funguje správně. Že to nejsou pouze filosofické otázky bylo již ukázáno na testu, kdy do slabé umělé inteligence bylo červené tlačítko zabudováno. Protože při jeho stisknutí nebyla schopna dosahovat stanovaných cílů, vyvinula si chování, které bránilo stisknutí tlačítka.

Dosud jsme se bavili o úrovni kognitivních schopností, inteligenci. Ale i v lidské společnosti se vyskytují osoby s vysokým IQ, které mají zcela odlišné chování a hodnoty. Mezi lidmi s vysokým IQ najdeme jak altruisty, tak i sobce, ale i vrahy a psychopaty. Pokud tuto paralelu použijeme pro umělou inteligenci, můžeme mít modely, které chtějí upřímně pomáhat lidstvu, ale i psychopatické modely.

Je možné, že se v budoucnu stejně jako v povídkách Issaca Asimova začne objevovat povolání robotického psychologa nebo antropologa zkoumajícího umělou inteligenci.

5.1 Život 3.0

Zajímavý náhled na inteligenci můžeme najít v knize Život 3.0 [9] V této knížce se švédsko-americký fyzik a kosmolog Tegmark dívá na život jako na verze programu.

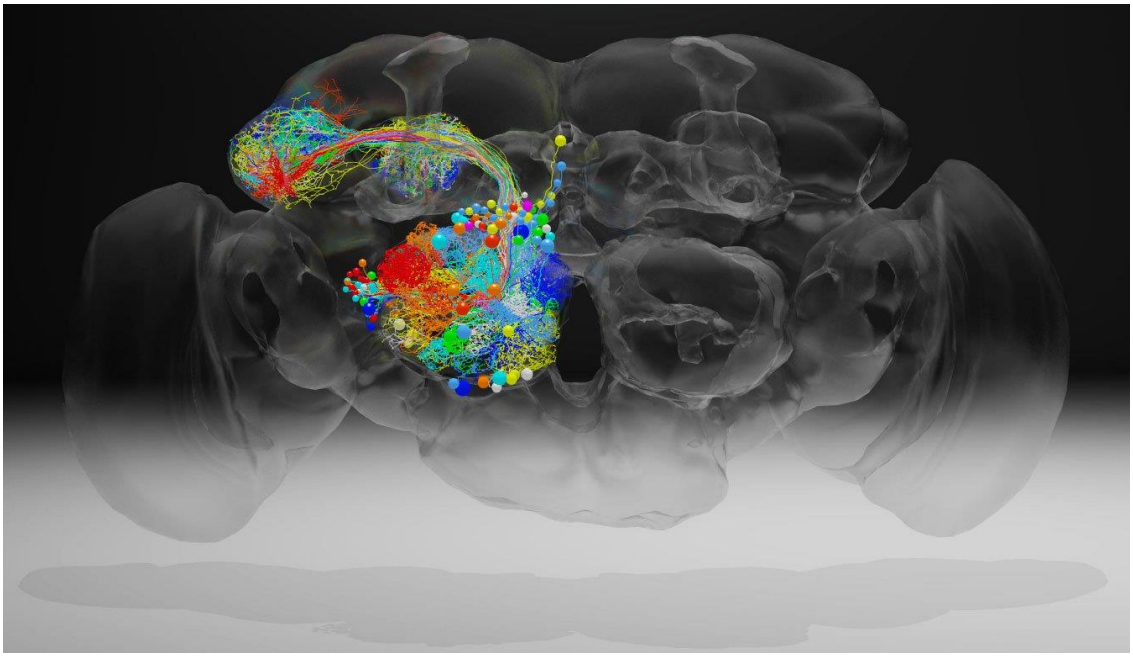
Život lze charakterizovat jako entitu, která se dokáže rekurzivně reprodukovat. Jedná se o složitý systém s nízkou entropií (je organizovaný). Jedná se o systém, který se snaží o minimalizaci volné energie, minimalizaci nesouladu mezi predikcí sebe sama v budoucím čase a tím, co skutečně nastane.

Verzi života 1.0 autor charakterizuje jako systémy, u kterých se HW a SW mění pouze evolučně. Vylepšení lze dosáhnout pouze v další generaci přirozeným výběrem. Aktuální jedinci se nezlepšují.

Typickým příkladem jsou nízcí živočichové jako bakterie, hmyz atp. Pokud je hmyz hnán za světlem, za kterým je smrtelná past, nemá možnost své chování změnit. Možné vylepšení jde jen díky evolucí. Mrtvý hmyz se nerozmnoží, a tak do další generace přežijí pouze jedinci s vhodným chováním vůči změněnému prostředí.

V nedávné studii vědci detailně prozkoumali mozek larvy octomilky [10]. Model mozku je velmi detailní až na úroveň jednotlivých synapsí. Mozek se skládá z 3016 neuronů, 548 000 synapsí. Vědcům se podařilo identifikovat, že část mozku slouží pro uložení nebezpečných míst, část neuronů je zodpovědná za let budoucí octomilky. Největší část mozku je ovšem určena pro učení.

Velikost mozku octomilky velmi dobře dokážeme porovnat se současnými relativně malými umělými neuronovými sítěmi. Stejně jako octomilky dokáží současné ANN řešit omezený počet i komplikovaných úloh jako let a orientace v prostoru.



Obrázek 6 - mozek octomilky

Verze života 2.0 se od verze 1.0 odlišuje v jedné zásadní věci. Sice mozek (HW), jeho rámcová struktura, kapacita a ostatní fyzické parametry, zůstává po dobu života jedince relativně stabilní. Zlepšování mozku se ovšem děje v rámci evoluce, na což ukazují kosterní pozůstatky našich předků.

Co se ovšem liší je schopnost učit se a reagovat na změny prostředí. Během procesu učení se vytváří, posilují, zanikají jednotlivá neuronová spojení. Člověka můžeme zařadit do života verze 2.0.

Ve své knížce Max Tegmark zavádí i další **verzi života 3.0**, která dosud neexistuje, ale dá se přirovnat k umělé inteligenci. Verze života 3.0 dokáže měnit HW a SW průběžně za života jedince. Proces zlepšování SW pomocí učení probíhá stejně jako u života verze 2.0. Co se podstatně změnilo, že HW lze u jedince také průběžně měnit a zlepšovat. Můžeme si to představit tak, že bychom se rozhodli, že si zvýšíme kapacitu paměti nebo že si k mozku pořídíme rozšíření, které nám znásobí kognitivní schopnosti. Zjednodušeně řečeno, život 3.0 dokáže provádět HW upgrade. To třeba dvěma jedincům umožňuje rychlou výměnu znalostí, zkušeností a dovedností. Jednoduše se zkopíruje část mozku. Nebo není problém, aby se mozky dvou jedinců spojily dohromady.

Způsobů, jak tohoto dosáhnout může být více. Jedním z přístupů může být spojování jednotlivých slabých AI do jednoho systému. Při správném propojení se mohou objevit kognitivní schopnosti, které požadujeme od AGI. Tedy schopnost učit se, stanovovat si cíle a průběžně se zlepšovat.

Jiným způsobem je vytvoření správného „substrátu“, tedy vhodné struktury velké neuronové struktury. Tuto strukturu budeme vystavovat dostatečně komplexnímu prostředí a určíme jí základní cíle. Po čase se mohou vyskytnout kognitivní schopnosti, které se neustálým učením budou zvyšovat. Toto povede ke vzniku superAI.

Zcela jiným přístupem může být vylepšování současné biologické inteligence o umělé prvky. Může vzniknout kyborg, který bude využívat výhod biologické i umělé inteligence. To, že se nemusí jednat o sci-fi v daleké budoucnosti nasvědčují pokusy s neuralinkem, který financuje Elon Musk.

Jaký bude ten správný směr, ukáže budoucnost. Pravděpodobně půjde o souběžný vývoj.

6 Reference

[1] I. Dastin, „Amazon scraps secret AI recruiting tool that showed bias against women,“ Reurers, [Online]. Available: https://www.reuters.com/article/us-amazon-com-jobs-automation-insight-idUSKCN1MK08G.

[2] D. Shepardson, „US opens investigation into fatal Tesla crash in Virginia,“ Reuters, [Online]. Available: https://www.reuters.com/technology/us-opens-new-investigation-into-fatal-tesla-crash-virginia-2023-08-10/.

[3] „Tramvajové dilema,“ Wikipédia, [Online]. Available: https://cs.wikipedia.org/wiki/Tramvajov%C3%A9_dilema.

[4] „What’s the Latest on the Writers’ Strike?,“ The New York Times, [Online]. Available: https://www.nytimes.com/article/wga-writers-strike-hollywood.html.

[5] „Nepodmíněný příjem,“ [Online]. Available: https://cs.wikipedia.org/wiki/Z%C3%A1kladn%C3%AD_nepodm%C3%ADn%C4%9Bn%C3%BD_p%C5%99%C3%ADjem.

[6] I. Šlerka, „Umělá inteligence: čtyři výzvy pro humanitní a sociální vědy,“ [Online]. Available: https://www.youtube.com/watch?v=k1LQSkSaIBw&list=PLgSWX-1N3JZ4vNEUtsRxu5mCeHGQYtGpJ&index=1.

[7] „Cthulhu Mythos,“ [Online]. Available: https://en.wikipedia.org/wiki/Cthulhu_Mythos.

[8] „OpenAI Used Kenyan Workers on Less Than $2 Per Hour to Make ChatGPT Less Toxic,“ Time, [Online]. Available: https://time.com/6247678/openai-chatgpt-kenya-workers/.

[9] M. TEGMARK, Život 3.0, Praha: Argo, 2020.

[10] „Scientists complete first map of an insect brain,“ [Online]. Available: https://hub.jhu.edu/2023/03/09/scientists-complete-first-map-of-an-insect-brain/.

[11] I. Zelinka, Umělá inteligence - hrozba nebo naděje, 2023: BEN - technická literatura, Brno.

[12] K. Warwick, Úsvit robotů, soumrak lidstva, Praha: Vesmír, 1999.

[13] M. Minsky, „Steps Toward Artificial Intelligence,“ v Proceedings of IRE 49, 1961.

[14] W. P. Warren S. McCulloch, „A logical calculus of the ideas immanent in nervous activity,“ The Bulletin of Mathematical Biophysics, 1943.

[15] D. Hebb, The Organization of Behavior, New York: Wiley and Sons, 1949.

[16] F. Rosenblatt, „The perceptron: A probabilistic model for information storage and organization in the brain,“ Psychological Review 65, 1958.

[17] D. K. J. B. a. R. F. Block, „Analysis of a four-layer series-coupled perceptron.,“ Reviews of Modern Physics 34, 1962.

[18] M. P. S. P. Minsky, „An Introduction to Computational Geometry.,“ v MIT Press, Cambridge, 1969.

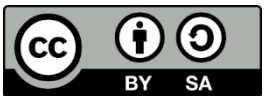
[19] I. Hopfield, „Neural networks and physical systems with emergent collective computational abilities,“ v Proceedings of the National Academy of Sciences of the United States of America 79, 1982.

[20] D. H. G. a. W. R. Rumelhart, „Learning representations by back-propagating errors,“ v Nature 323, 1986.

[21] T. Kohonen, „Self-organized formation of topologically correct feature maps,“ v Biological Cybernetics 43, 1982.

UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vytvořeno v rámci projektu **Digitalizace studijních Agend, Nové TechnologiE, systémy a přístupy k výuce na UPCE**, reg. č. NPO_UPCE_MSMT-16591/2022.



Financováno Evropskou unií
NextGenerationEU

Národní plán obnovy

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY