Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Predikce ceny automobilu

Diplomová práce

Autor: Bc. Zdenko Brandejs
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. RNDr. Kamila Štekerová, Ph.D.

Hradec Králové srpen 2020

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 19.8.2020 Zdenko Brandejs

Poděkování:

Děkuji vedoucímu diplomové práce doc. RNDr. Kamile Štekerové, Ph.D. za vedení práce a Ing. Michalovi Dobrovolnému za poznámky k praktické části práce.

Anotace

Cílem práce je popsat principy strojového a hlubokého učení, vysvětlit problematiku predikce cen, porovnat přístupy v algoritmizaci predikce, vysvětlit možnosti implementace v Pythonu a v neposlední řadě realizovat ukázkovou aplikaci zaměřenou na predikování vývoje ceny automobilu od okamžiku jeho uvedení na trh. Během praktické části je použito vícero přístupů v experimentech, jejichž výsledky jsou analyzovány a rozebrány. Práce má praktické využití v odvětvích jako půjčování a prodej automobilů, bankovnictví, pojišťovnictví, automobilový průmysl atp.

Annotation

Title: Prediction of car prices

The aim of this work is to describe the principles of machine and deep learning, explain the issue of price prediction, compare approaches in predicting algorithms, explain implementation options in Python and last but not least implement a sample application aimed at predicting car price from the moment it is launched. During the practical part, several approaches are used in experiments, the results of which are analyzed and decomposed. The work has practical use in sectors such as rental and car sales, banking, insurance, automotive industry, etc.

Obsah

Seznam obrázků

Seznam tabulek

Seznam diagramů

1 Úvod

V dnešní době se hodně mluví o umělé inteligenci, která se uplatňuje v mnoha odvětvích, aniž bychom si to někdy plně uvědomovali. Lidský čas je pro nás velice cenný a je i dražší ve srovnání se strojovým. Kromě finančních úspor však práce strojů přináší mnohé další výhody, ale bohužel i úskalí. Jednou z odvětví, kde se hojně využívá umělé inteligence, je ekonomie. Tento obor se často snaží předpovídat chování určitých indikátorů v průběhu času.

Toužil jsem vytvořit práci, která by měla přesah do vícero odvětví v praxi a nebyla úzce spjata pouze s jedním konkrétním využitím na míru. Proto jsem se rozhodnul vytvořit práci, jejímž jádrem je predikování ceny automobilu. Vhodně optimalizovaný nástroj může sloužit nejen širokému portfoliu podniků, ale také koncovým zákazníkům.

Mým motivačním motorem bylo také nahlédnout do oboru umělé inteligence, s kterým jsem jako programátor ještě neměl možnost se seznámit jinak než jako pouhý uživatel. Velkou oblibou mi je také využití Python programovacího jazyka a s ním související technologie.

1.1 Cíl

Cílem této diplomové práce je vytvořit ucelený soubor potřebných teoretických poznatků přesahující samotnou podstatu informačních technologií a umělé inteligence směrem k ekonomii. Teorii jsem se rozhodl převést do praktického výsledku v podobě spustitelného programu, který uživateli poskytne hmatatelné výsledky. Výsledky jsou adekvátně interpretovány pro vyvození závěrů v podobě srovnání, grafů a tabulek.

1.2 Metody

Při přípravě diplomové práce jsem provedl rešerši literatury, na základě které jsem jako vhodnou technologii zvolil programovací jazyk Python s frameworkem Keras, umožňující snadnou implementaci umělé neuronové sítě.

Pro získávání dat jsem vytvořil automatizovaný nástroj taktéž v jazyce Python, který pomocí web scrape techniky postupně získává potřebná data o automobilech z internetového portálu www.sauto [1].

2 Teoretická část

V této kapitole je rozebrán veškerý teoretický základ potřebný pro další postup v práci. Vysvětlení postupuje z obecnějšího okruhu umělé inteligence přes strojové učení až po konkrétnější vysvětlení neuronových sítí spadajících do hlubokého učení. Dále je potřeba vysvětlit ekonomický pohled na problematiku predikce ceny automobilu. Posledním potřebným oborem k seznámení se s jeho teorií jest matematika. V matematickém popisu teorie je probráno několik vhodných přístupů k transformaci dat, které se následně implementují. Výstupy z implementací budou sloužit k porovnání a vybrání nejpřesnějšího modelu pro samotnou predikci. Závěrečná podkapitola vysvětlí možnosti Pythonu a jeho dostupných technologií, které jsou vhodné pro použití této práce.

2.1 Umělá inteligence

Umělá inteligence (dále jen AI = Artificial Intelligence) je v dnešní době všude kolem nás. Mnozí lidé se jí bojí, jelikož představa, že vlak je řízen naprogramovaným strojem a v kabině strojvedoucího není žádný člověk, se nemusí líbit každému. Některým lidem tenhle trend přijde naopak zajímavý až fascinující, jaký potenciál s sebou přináší do této moderní doby. Pravdou je, že naši předci snili o stvoření myslících strojů, již od starověkého Řecka (převážně v literárních dílech). Práce Ady Lovelace [2] definovala v polovině 19. století, jak programovat různé bloky kódu, aniž by fyzicky v té době existoval počítač, který by byl schopen program spustit. Svým dílem předstihla první počítače přibližně o 100 let.

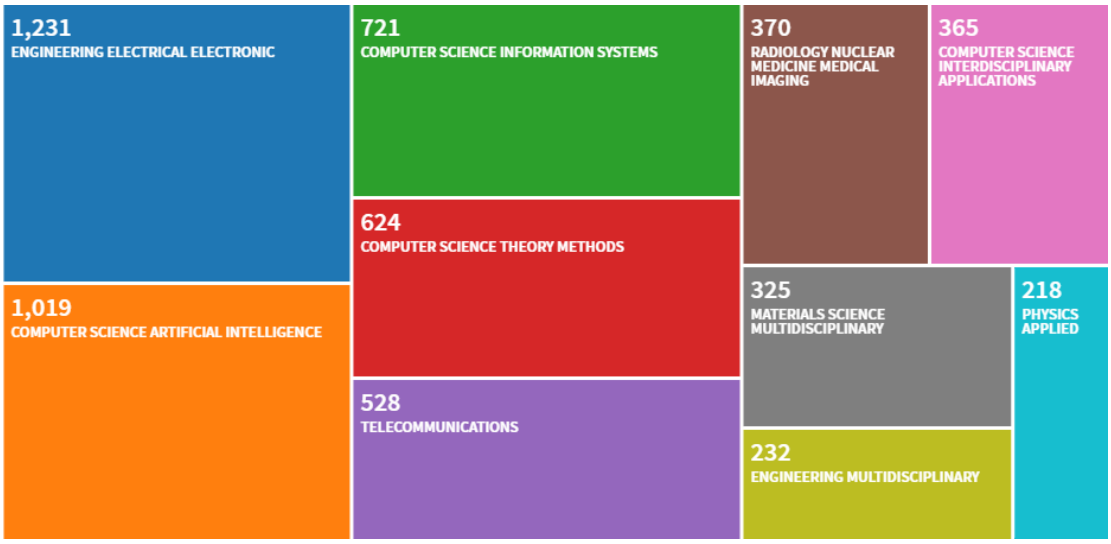
Velmi významným milníkem ranné historie umělé inteligence byl rok 1997, kdy tehdejšího světového šachového velmistra Garry Kasparova porazil počítač. Tehdejší stroj firmy IBM byl naprogramovaný striktně na bázi předpočítání možných tahů a vyhodnocení možných strategií dopředu. Program tehdy ještě nemusel mít ponětí o širším světě, protože ten šachový je omezen pouze na 64 polí s předem definovanými pohyby postaviček, které lze snadno kompletně definovat formálními pravidly.

Abstraktní a formální úkoly jsou paradoxně pro lidi jedny z nejnáročnějších vůbec, přitom pro počítače jedny z nejlehčích. Důkazem je právě vítězství počítače

nad Kasparovem v šachu, v opačném kontrastu se však teprve před nedávnými lety počítače naučily rozpoznávat mluvenou řeč, což odpovídá průměrným schopnostem člověka. Každodenní život člověka vyžaduje velkou spoustu znalostí o světě kolem nás. Většina těchto znalostí jsou subjektivní a intuitivní, proto je velice obtížné všechny vazby kolem nás formálně vyjádřit. Stejně jako lidi, i počítače musí postupně zachytit tyto znalosti k tomu, aby se chovaly inteligentně. Jednou z klíčových výzev oboru AI je odpověď na otázku – jak získat neformální znalosti do počítače.

V dnešní době bereme umělou inteligenci jako prosperující oblast s mnoha praktickými využitími a aktivními vědeckými tématy. Typicky využíváme AI k automatizování rutin, porozumění či rozpoznání řeči a obrázků, tvorbě diagnostik v medicíně, podpoře vědeckých výzkumů atd.

Oblast AI je aktuálně velice bádané. Například na portálu Web of Science je již 8004 příspěvků za celý rok 2019 s přesahem do 13. 1. 2020, více či méně spjatých právě s AI oborem.



Obr. 1: Výsledky vyhledání prací s tematikou Artificial Intelligence
Zdroj: Web of Science

2.2 Strojové učení

Autoři různých AI projektů se pokoušeli pevně naprogramovat znalosti o světě formálními jazyky. Počítač následně pomocí pravidel, logických operací a odvozování mohl argumentovat o prohlášeních. Takový přístup AI je založen na znalostní bázi (anglicky knowledge base). Problémy, kterým čelí systémy postavené na pevně naprogramovaných znalostech, naznačují, že systémy AI potřebují umět získat znalosti extrahováním vzorů ze surových dat. Tato dovednost je známa jako strojové učení (dále jen ML = Machine Learning).

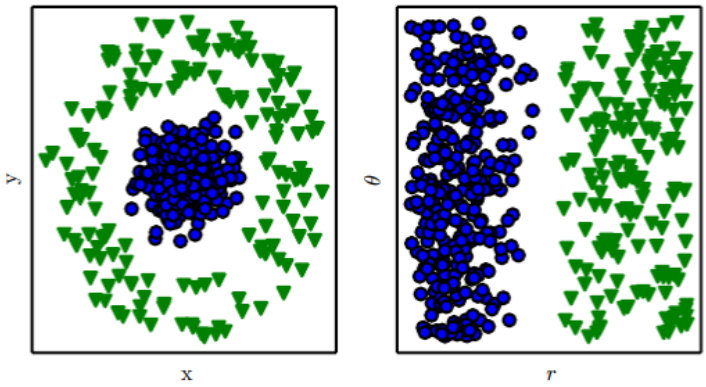
ML umožnilo počítačům řešit problémy skutečného světa, které jsou pro lidstvo značně subjektivní. Subjektivní problém znamená, že každý člověk má na daný problém jiný pohled či názor, na základě svých dojmů a zkušeností. Od podobné subjektivity je stroj odstíněn a umí tedy problém řešit objektivně bez předsudků a zaujetí. Mezi takové algoritmy patří např. logistická regrese, která umí doporučit provedení císařského řezu v porodnictví [3]. Dalším příkladem je využití jednoduchého algoritmu – naivní Bayes, který se běžně používá pro rozlišování e-mailů, zda se jedná o spam, nebo vyžádanou pošta.

Metody ML jsou založeny na trénovacích algoritmech s cílem identifikovat v datech vztahy, tvořit predikční modely a dělat rozhodnutí. Čím „lepší“ vstupní trénovací data jsou, tím jsou lepší výsledky algoritmů. Klasickým příkladem je klasifikace obrázků, kdy člověk roztřídí datovou sadu do kategorií (např. pes, kočka) a algoritmus se naučí rozpoznávat obsah obrázků. Následně algoritmus zařazuje do kategorií testovací obrázky [4].

Velký vliv na výkonnost AI algoritmů má **reprezentace** vstupních dat a dodatečných informací. Výše zmíněný algoritmus pro doporučení použití císařského řezu by bez dodatečných informací vložených lékařem, nemohl nikdy správně určit postup. Každá informace doplňující reprezentaci dat je charakteristická **vlastnost** (anglicky feature). Následně algoritmus na základě dat např. vložených snímků z magnetické rezonance a lékařského reportu, umí korelovat ke správné predikci s možnými komplikacemi.

Další dopad na výkonnost může mít výběr vhodné datové struktury s operacemi nad nimi. Typicky se jedná o prohledávání v datových kolekcích, které mají vhodnou

strukturu a jsou inteligentně indexovány. Tak jako pro lidi je snazší zpravidla počítat s arabskými číslicemi než s římskými, určité algoritmy zase upřednostňují polární soustavu souřadnic oproti kartézské (Obr. 2). Polární soustava umožní vhodnější kategorizaci dat vstupního vzorku, např. jednoznačnost pohlaví.



Obr. 2: Rozdíl datové reprezentace. Vlevo kartézská, vpravo polární soustava souřadnic
Zdroj: kniha Deep Learning [2]

Často je velice obtížné znát veškeré klíčové vlastnosti, které by měly být extrahovány. Běžným úkolem AI bývá rozpoznávání předmětu v obrazu, např. detekce automobilu. Lze postupovat od identifikování kol a následně zbytku vozidla. Kolo má jednoduchý geometrický tvar, ale zároveň se v obraze může objevit artefakt (např. vržený stín jiného předmětu, sluneční odraz, překážející předmět v popředí atd.), který může pohled stroje na tento tvar mystifikovat, což povede ke špatné identifikaci. Dalším příkladem chyby detekce může být rozpoznání auta oproti slonovi v obraze. V případě fotografie hořícího auta může plamen pokrývající auto rozšířit tvar subjektu právě do podoby slona a algoritmus mylně detekuje oheň jako součást vozidla a určí výsledný předmět jako slona [5].

Algoritmy, které jsou závislé na kategorizování vzorků jsou náchylné i kvůli samotné kategorizaci. Tuhle problematiku zmínila Kris Howard[1] [6] během své přednášky, kde prezentovala svoji práci s detekcí pletacího stehu z fotografií hotových výtvorů. K naprogramovanému výtvoru využila širokou veřejnost z celého

[1] Konference Build Stuff, listopad 2019, Vilnius, Litva. Účastnil jsem se osobně. (Záznam: https://youtu.be/0lqyYJl9vuM)

světa. Lidé ji v první fázi poslali fotografie pletených děl a v druhé fázi měli identifikovat použitou pletací techniku. Problémy nastaly v obou fázích. Veřejnost zásobovala Kris fotografiemi, které neobsahovaly vůbec žádné upletené subjekty. V druhé fázi se objevily identifikační chyby, pramenící z terminologie pletení, která se lišila napříč zeměmi a kontinenty, např. stejnému stehu byl přiřazen jiný název v USA a jiný v Austrálii.

Podobné chyby v detekci obrazu se dají řešit rozšířením algoritmů, tak aby se učily rozpoznávat vstupní data, a nejen pouze mapovaly vstup na výstup. Takový přístup se označuje jako **učení reprezentace** (anglicky representation learning). Metodika typicky používá mechanismus tzv. autoencoder, který je schopný vstupní data zakódovat do své interní reprezentace, které člověk nemusí rozumět (např. značky automobilek přeloží na celá čísla). Následně autoencoder umí reprezentaci dekódovat opět do podoby vstupních dat, tak aby byl výsledek použitelný. Autoencoder je běžně stavěný tak, aby během transformací s sebou nesl všechny informace a o nic nepřišel. Je jich několik druhů a každý cílí na jiné použití a využitelnost.

Pro další eliminaci chyb a správné navržení algoritmů je nutné oddělit **variační faktory**, které charakterizují pozorovaná data. Jsou to právě vlastnosti, které nejsou ihned zřejmé z pohledu stroje. Při rozpoznávání mluveného slova to je například věk, pohlaví, nebo přízvuk řečníka. U detekce vozidla to může být poloha auta, jeho barva, nebo úhel a jasnost slunečního svitu. Nejčastější chybou v mnoha AI projektech z reálného světa je právě kombinace faktorů, jelikož každá taková variace ovlivňuje jiný úsek dat, které jsme schopni sledovat. Ve vztahu k problematice automobilů – při detekci ve špatných světelných podmínkách, se červená bude jevit více jako barva černá; tvar siluety vozidla bude záležet na úhlu pohledu. Proto je tedy nutné oddělit variace faktorů a vynechat ty, které potřebné nejsou. Někdy se na první pohled může zdát, že učení reprezentace nám moc nepomáhá, protože některá data se jeví příliš abstraktní, např. určení přízvuku z řeči, může být někdy skoro tak obtížné, jako původní problém určit řečníka. Právě na podobně složité problémy už nám pomáhá hluboké učení, které je schopné tento problém rozložit do více potřebných učících reprezentací skrze tzv. neuronovou síť.

Ucelené souvislosti podoblastí AI lze znázornit za pomoci diagramu.

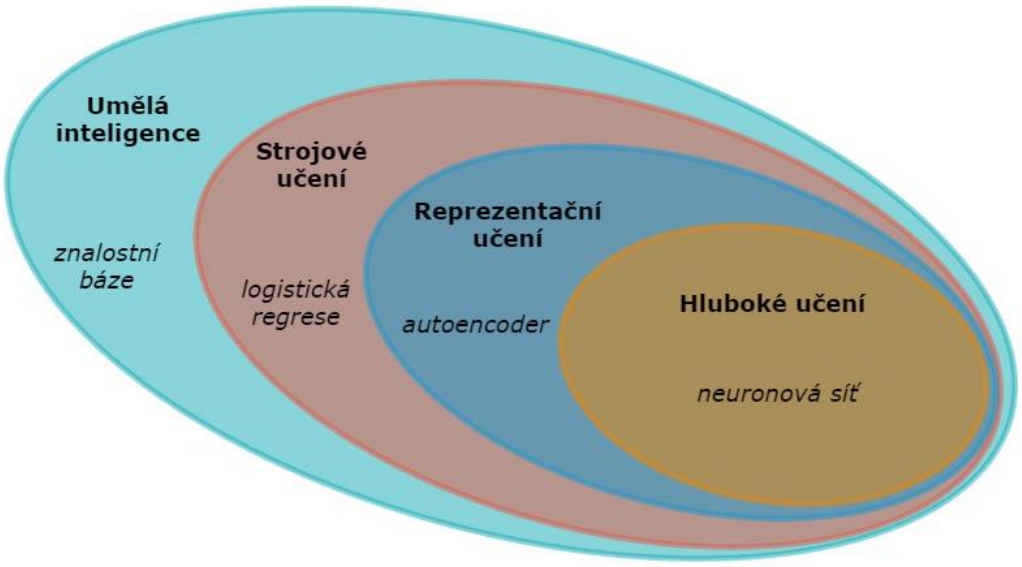


Diagram 1: Diagram znázorňující zmíněné podoblasti AI jako její podmnožiny s typickým využitím.
Zdroj: vlastní zpracování

2.3 Hluboké učení

Druhou podoblastí AI, kterou zmíním, je hluboké učení (dále jen DL = Deep Learning). DL je vhodné zejména pro intuitivnější problémy. Takové řešení umožňuje počítači učit se ze zkušeností a porozumět světu v rámci hierarchické sítě, přičemž každý prvek je definován dále ve vztahu k dalšímu prvku sítě. Shromažďováním znalostí ze získaných zkušeností se tento přístup vyhýbá nutnosti specifikování znalostí člověkem. Hierarchie umožňuje počítači pochopit komplikované vazby skládáním vrstev od jednodušších po složitější celky. Graf popisující celou strukturu, má plno vrstev a bude hluboký. Odtud tedy vznikl název oboru hlubokého učení.

Stejně jako předchozí ML je potřeba použít trénovací data, avšak s tím rozdílem, že člověk nemusí extrahovat data na dílčí kategorie. Místo toho jsou data vložena přímo do algoritmu, který sám určí, co za objekt se vyskytuje v obrázku. [7]

Myšlenka DL přináší více perspektiv, jak můžeme na řešenou problematiku pohlížet. Jednou z nich je osvojení správné reprezentace dat, tou další je umožnění naučení počítače vícekrokovým programem. O správné reprezentaci dat již bylo řečeno dost výše, a proto se zaměřím na druhou perspektivu. V podstatě si lze představit každou vrstvu učení, jako stav paměti počítače po provedení nějaké sady paralelních operací. Sítě s větší hloubkou mohou provést více operací v sekvenci. Sekvenční operace nabízí velkou sílu, protože pozdější operace mohou odkazovat zpět k výsledkům z předešlých operací. Navzdory tomuto tvrzení, ne vždy je každá informace v aktivaci vrstvy nezbytně zakódována z variace faktorů, která by vysvětlovala vstup. Reprezentace také uchovává stav informace, který napomáhá v běhu programu, tak aby vstup dával smysl. Zmíněný stav je analogicky podobný mechanismům běžně používaným v počítačových programech, jako čítače, nebo ukazatele. Nemá vliv na samotný obsah vstupu, ale napomáhá modelu v organizaci zpracování.

Pro přehlednost je vhodné u větší sítě umět změřit její hloubku. Hloubku DL modelu můžeme měřit dvěma hlavními způsoby. Prvním je pohled zaměřený na počtu sekvenčních operací, které musí být spuštěny k vyhodnocení architektury. Jednoduše řečeno se jedná o délku nejdelší možné cesty napříč vývojovým

diagramem (anglicky známé jako flowchart), který popisuje, jak daný model ze vstupů dosáhne patřičných výstupů. Druhým používaným způsobem zejména v pravděpodobnostních modelech, je hloubka grafu popisující vzájemné relace mezi prvky modelu. Hloubka výpočtového grafu není směrodatná, protože délka vývojového diagramu výpočtů může být mnohem větší než vhodnější graf relací entit modelu. Důvodem je samotný princip DL, kdy porozumění jednodušší struktury vede k pochopení složitější, proto tedy výpočet vstupní reprezentace může být zbytečně větší, než tomu ve skutečnosti může být v případě zaměření se na samotné vazby v modelu. Např. detekce uší v obrázku, kde jedno ucho může být ve stínu a může se jevit, že objekt má pouze jedno ucho, avšak pokud je detekován obličej, pravděpodobně bude mít i dvě uši. Daný model by měl tedy dvě vrstvy, jedna pro uši a druhá pro obličej. Kdežto graf výpočtů by měl hloubku 2n, pokud zpřesníme náš odhad každého prvku vzhledem k ostatním n-krát.

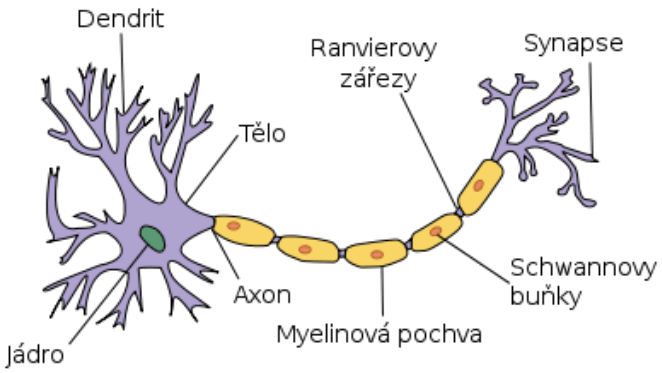
Neexistuje jednotné stanovisko pro určení správné hloubky modelu, protože pro každý model se hodí jiný přístup a jiné určení nejmenších prvků modelu z kterých se vytvoří výsledný graf. Stejně tak neexistuje hranice, od které by se určilo, zda se jedná o hluboký model spadající do DL oblasti. Obecně se však vychází z toho, že DL zahrnuje větší množství učících vrstev a funkcí než tradiční ML.

2.4 Umělá neuronová síť

V této podkapitole vysvětlím základy neuronových sítí (dále jen NS), které spadají do hlubokého učení. Zejména uvedu popis fyziologického neuronu, umělého neuronu, učení sítě a druhů NS.

Mnoho algoritmů a postupů v informatice je inspirováno přírodou. Lidé se odjakživa zajímaly o funkce organismů a jedním z nejsložitějších systémů, je mozek. Hluboké učení se zrovna tak inspirovalo u nervové tkáně. Neuron je základní funkční a stavební jednotkou živočišné nervové soustavy. Soustava neuronů se vyznačuje zvýšenou mírou dráždivostí a vodivostí, je tedy schopná vést smyslové vzruchy. Neurony samotné se skládají z těla (soma), dvou druhů výběžků (dendrity, axony) a nervového zakončení. Dendrity jsou typem zakončení, které do neuronu přijímají informace z receptorů či ostatních neuronů. Informace je v těle neuronu dále uložena a transformována. Axon je naopak výběžkem zodpovědným za přenos

informace ven z neuronu do dalšího, případně k efektoru. Počet výběžků má vliv na klasifikaci neuronu, dělí se pak na unipolární, bipolární a multipolární. Informace mezi neurony je předávána přes místo, kde se neurony potkávají, takové místo se označuje jako synapse. [8]



Obr. 3: Lidský neuron
Zdroj: Anatomie neuronu **[9]**

Neuron si přijatou informaci uchovává a dalšími signály se informace sčítá a probíhá tak transformace. Tato vlastnost neuronu je velice důležitá a vypichuje tak funkčnost celé nervové soustavy. Člověk dík takové soustavě je schopen vnímat, učit se, uchovat si informaci a v případě potřeby si na ni vzpomenout. Umělá neuronová síť využívá všech vlastností nervové soustavy.

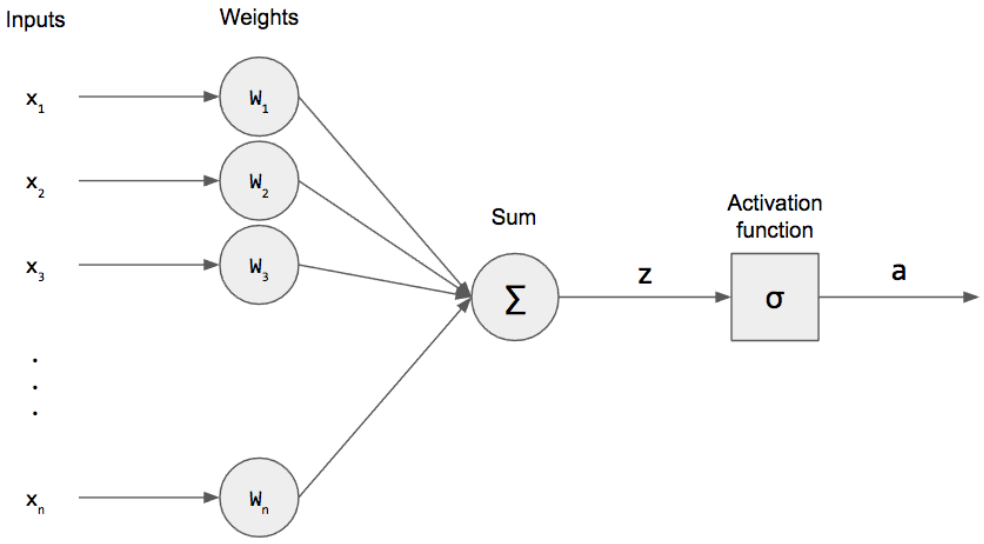
NS mají jisté výhody i nevýhody. Mezi výhody lze brát fakt, že není potřeba hledat vztahy mezi proměnnými [10]. Data jsou jednoznačná a tím se eliminuje časová náročnost s objevováním relací ve vzorku dat. Kromě toho, další analýza dat může vést k nárustu chybovosti. NS je nelineární aproximátor, který je vhodný k řešení velice komplexních problémů, jako např. chování cestujících a ekonomický vývoj. Další výhodou je četnost propojení neuronů a jejich paralelní zapojení, které umožňuje šíření informace různými směry. Vše uvedené dohromady umožňuje spoustu kombinací šíření informace v rámci NS, to vede k robustnosti celého systému. Pokud dojde ke zkreslení informace v jednom úseku, tak to na výpočet jako takový, nemá velký dopad. Nejběžněji zmiňovanou nevýhodou NS je dojem, že se jedná o tzv. „black box“ (česky černá skříňka). Často totiž není patrné, k jakým výpočetním operacím dochází v transformačních vrstvách. Také je někdy náročné

pochopit interpretaci parametrů, které do vrstev vstupují a vystupují, zejména v hlubokých sítích se snadno ztrácí přehled.

2.4.1 Perceptron

Základním stavebním prvkem neuronových sítí je perceptron. Název vznikl z matematického označení modelu neuronu známého z biologie. Analogicky pak vícevrstvá perceptronová síť je veřejně známá neuronová síť, stejně tak jako perceptron je umělý neuron.

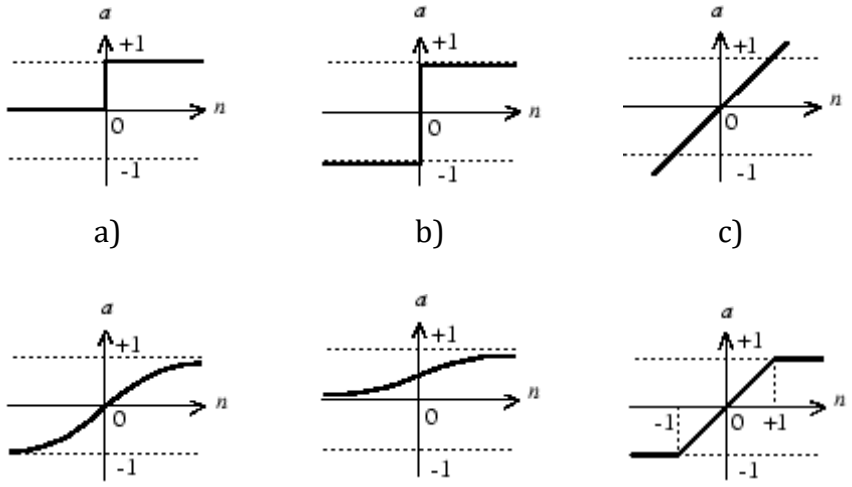
Perceptron má jeden až N vstupů (obdoba dendritů) a jeden výstup (obdoba axonu). Každý přechod mezi neurony (synapse) je doplněn o váhu vstupu tzv. synaptická váha. Samotný perceptron obsahuje tzv. potenciál neuronu (někdy uváděno jako prahová hodnota, nebo vnitřní potenciál). Při překonání potenciálu dojde k nabuzení perceptronu a následné indikaci signálu dál svým výstupem v podobě přenosové funkce (v některých literaturách známé jako funkce přechodu, nebo aktivační funkce).[11]



Obr. 4: Složení umělého neuronu
Zdroj: článek „Perceptrons - The First Neural Networks“ **[12]**

Váha vstupu určuje důležitost konkrétního vstupu. Neuron je v tomhle místě schopný uložit si naučenou informaci a zreprodukovat v případě potřeby. Během učení sítě se váhy mohou adaptovat a měnit důležitost vstupu.

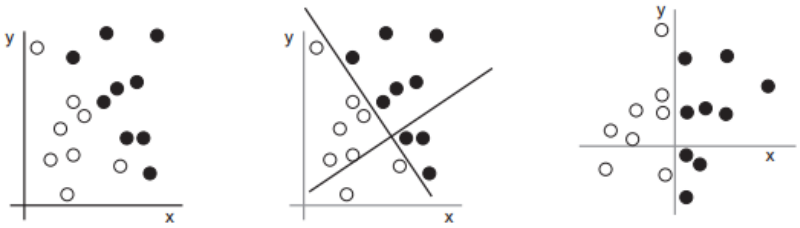
V neuronových sítích se dá využít široká škála přenosových funkcí. Vybrat tu správnou pro NS řešící určitý problém není vždy snadné. Avšak nejčastěji používané funkce jsou např. jednotkového skoku, signum, sigmoidální, lineární, saturační a hyperbolický tangent. Výběr přenosové funkce má zásadní vliv na konvergenci výpočtu naučení NS.



Obr. 5: Nejpoužívanější přenosové funkce využívané v hlubokém učení.
a) jednotkový skok, b) signum, c) lineární, c) hyperbolický tangent, d) sigmoidální, e) saturační

Zdroj: MATLAB dokumentace Deep Learningu **[13]**

Klasickým příkladem, kde lze využít samotný perceptron bez nutnosti komplexní NS, je klasifikace, nebo jiné jednoduché rozhodování. Viz Obr. 6, kde jsou zaznamenány prvky do dvou rozměrných souřadnic. Pro vhodnější reprezentaci a snadnější vyhodnocení by souřadnice měly být transformovány před vstupem perceptronu. Perceptron má nastavené rozlišení bodů na základě přenosové funkce. Pak může perceptron snadno rozlišit prvky pomocí signum funkce, která určí bílé prvky v případě kdy $x < 0$ a analogicky černým prvkem se stane s $x > 0$.



Obr. 6: Demonstrace využití perceptronu v klasifikaci 2D prvků na černé a bílé.

Zdroj: Deep learning with Python **[14]**

2.4.2 Učení neuronové sítě

Tahle kapitola popisuje nejcennější vlastnost neuronových sítí, kterou je učení problému pouze na základě předkládaných vzorů. Učení NS má tedy za cíl nastavit váhy modelu tak, aby vytvářely správnou odezvu výstupního signálu na dané vstupní signály. Po celém procesu učení lze nahlížet na vytvořený model jako na „black box“ a řešení aplikovat na danou problematiku.

Vstupní nasbíraná data se rozdělí do dvou množin mezi trénovací a testovací. V některých případech lze použít také třetí množinu tzv. validační. Neurony sítě se učí na trénovacích datech. Bývá zvykem, že tato sada je do počtu vzorků nejpočetnější. Trénovací vzorky sady obsahují mezi sebou vzory, které NS rozpoznává a kontinuálně se učí, v důsledku učení se nastavují váhy a hledá nejvhodnější konfigurace. Během trénování je vhodné urychlit experimentování monitoringem chyb, tím odhalíme ideální počet kroků trénování sítě. Velký počet kroků totiž zdánlivě nemusí konvergovat ke správné konfiguraci. To může vézt k nedostatečné obecnosti modelu a jiným chybám z přeučenosti. Validační množina oproti učení slouží ke kontrole tréninku a v případě, že trénink směřuje špatným směrem, trénink zastaví. Je to vhodný mechanismus k zabránění přeučení. Poslední testovací množina slouží k ověření již naučené sítě. Vzorky z této množiny nebyly součástí učení, a právě proto jsou vhodné k testování již naučené nakonfigurované sítě.

Učení NS může probíhat podle dvou odlišných strategií. První strategie využívá učení s učitelem a druhá bez učitele. V prvním případě s učitelem, se NS snaží přenastavit váhy porovnáním aktuálního výstupu s požadovaným, tak aby se konkrétním vstupem snížil rozdíl mezi skutečným a požadovaným výstupem. Oproti

tomu učení bez učitele podobnou dynamiku postrádá a zaměřuje se na konzistenci výstupu. Jinými slovy síť se snaží poskytovat stejnou odezvu na podobné, nebo stejné vstupní vektory.

Váhy neuronů bývají nejčastěji inicializovány na náhodná čísla v předem stanoveném intervalu, anebo jsou všechny váhy nastaveny na stejnou hodnotu. Navrženým modelem NS se snažíme minimalizovat chybovost sítě. Chybovost můžeme vyhodnotit pomocí chybové funkce a v optimálně učící síti by tato funkce měla každou iterací klesat. Algoritmus učení je odlišný s každým modelem odlišným NS.

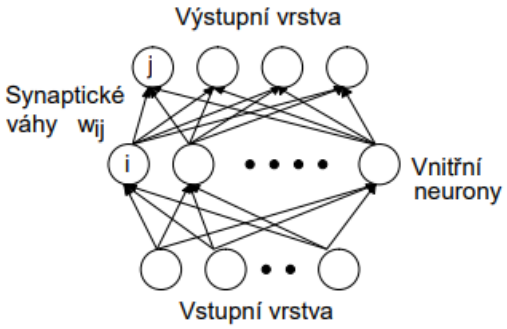
2.5 Druhy neuronových sítí

Neuronové sítě se dají rozlišit podle různých aspektů do mnoha skupin, např. dle počtu neuronů, uspořádání, šíření informací, typu využitých neuronů, nebo algoritmu učení.

Ohledně uspořádání sítě a šíření informací se používá přístup cyklický a acyklický (neboli dopředný). Cyklické uspořádání si lze představit jako zapojení neuronů v kruhu, tedy informace se do daného neuronu může vrátit i poté co z něj již odešla. Oproti tomu dopředné acyklické uspořádání takovou strukturu vylučuje a informace postupně prochází systémem neuronu, aniž by se měla možnost vrátit. Druhý zmíněný přístup je využit v této práci, jelikož je vhodnější pro samotnou predikci.

Pro úplnost zde uvedu opak „jednoduché sítě“ v podobě perceptronu a tím je právě vícevrstvá NS (v některých literaturách Multi Layer Perceptron). Model NS (viz Obr. 7) v takovém případě musí mít minimálně 3 vrstvy – vstupní, alespoň jednu vnitřní (v některých literaturách se popisují jako skryté vrstvy, což nepřispívá k pochopení „black boxu“) a výstupní (někdy se uvádějí vrstvy bez vstupní, v tom případě je minimální počet vrstev 2).

Vícevrstvá NS se dále může lišit algoritmem učení. Nejčastěji se volí Back Propagation Errors, Hebovské učení, sdružené gradienty, učení s učitelem atd.



Obr. 7: Schéma vícevrstvé neuronové sítě.
Zdroj: Skripta Umělá inteligence a neuronové sítě [15]

2.5.1 Dopředná NS

V kapitole 2.4.2 už byl nastíněn princip dopředné neuronové sítě, což je zároveň i český výraz FNN. Patří mezi nejjednodušší ANN (inteligentní NS), kde data nebo vstup putuje jedním směrem, informace se tedy nijak necyklí. Data prochází skrz vstupní uzly a opouštějí síť výstupními uzly. Dopředná síť může, ale nemusí mít více skrytých vrstev.

Ve své podstatě síť může vypadat jako na Obr. 7. Spočítá se suma vstupů s váhou uzlu a přivede na výstup, kde se rozhodne, zda je výsledná hodnota nad tzv. prahovou hodnotou (obvykle hodnota 0). Neuron pošle signál dál s hodnotou 1 v případě překročení prahové hodnoty, pokud hodnota bude pod tou prahovou odešle hodnotu -1 (přenosová funkce signum).

Tento typ NS se nejčastěji trénuje pomocí metody Backpropagation [16] (česky lze přeložit jako zpětné šíření chyb). Vzhledem k umělé NS a chybové funkci, vypočítává metoda gradient chybové funkce s ohledem na váhy sítě. Představuje zobecnění delta pravidla pro perceptrony k vícevrstvým dopředným NS. Slovo dopředná v názvu vychází ze skutečnosti, že výpočet gradientu postupuje zpět přes síť, přičemž gradient konečné vrstvy vah se vypočítává jako první a gradient první vrstvy vah se počítá jako poslední. Částečné výpočty gradientu z jedné vrstvy se znovu použijí při výpočtu předchozí vrstvy. Tento zpětný tok chybových informací umožňuje efektivní výpočet gradientu v každé vrstvě oproti naivnímu přístupu výpočtu gradientu každé vrstvy zvlášť. Popularita algoritmu v posledních letech vzrostla právě díky velkému využití v neuronových sítích pro rozpoznávání obrazu

a řeči. Považuje se za efektivní algoritmus a díky inovativním implementacím využívá výhod specializovaných GPU pro výkonnostní zlepšení.

2.5.2 Radial Basis Function NS

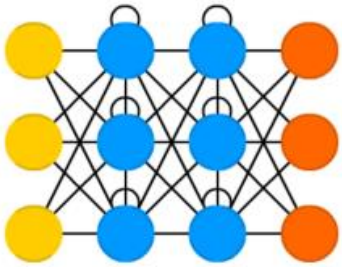
Zkráceně RBF je síť radiálních jednotek s pevným počtem vrstev a využívá stejnojmennou aktivační funkci. Ve své podstatě se jedná o jediný rozdíl oproti dopředné síti FNN. Algoritmus pracuje se vzdáleností bodu ke středu. RBF sítě mají dvě vrstvy, v první vnitřní vrstvě se vstupy kombinují s RBF funkcí a poté je výstup těchto funkcí zohledněn při výpočtu stejného výstupu v dalším časovém kroku, což je v podstatě paměť. Neurony mohou být tedy dvojího druhu, radiální a perceptronové. Na začátku učení jsou nastaveny váhy první vrstvy a u druhé se nastavují stejně jako u jiných vícevrstvých NS, avšak oproti nim je u RBF sítě rychlejší učení.

Praktickým příkladem využití RBF sítě může být systém obnovy energie. Systémy dodávky energie vzrůstají do velikosti, tak i do komplexity a oba tyto faktory zvyšují riziko výpadku proudu [17]. Po výpadku proudu je nutné dodávku obnovit co nejrychleji a stabilně. RBF síť by tedy prvně měla obnovit dodávku do míst, která jsou kritická pro potřebu a fungování dalších lidí (zdravotnické zařízení, policie, hasiči, obecní infrastruktura atp.). Poté obnovit dodávku vedení a rozvoden, které slouží pro širší základnu lidí. Měla by dát vyšší prioritu opravám, které co nejrychleji zvýší provozuschopnost co největšímu počtu lidí a v poslední řadě obnovit dodávku do menších čtvrtí, jednotlivých domovů a podniků.

2.5.3 Rekurentní NS

Rekurentní NS funguje na principu uložení výstupu vrstvy a jejího zpětného vložení na vstup, což napomáhá předpovídat výsledek vrstvy. První vrstva funguje stejně jako u dopředné NS se součtem vah a uzlů. Proces rekurentní NS začíná, jakmile je výpočet z první vrstvy k dispozici, to znamená, že každý neuron si bude pamatovat některé informace, které měl v předchozím kroku. Takové chování umožňuje každému neuronu sloužit jako paměťová buňka. Důležité je myslet na to, jaké informace budeme potřebovat pro pozdější použití a nechat síť pracovat v předních vrstvách. Pokud je predikce špatná, použijeme rychlost učení, nebo korekci

chyb k malým změnám, abychom postupně zlepšovali samotnou predikci během zpětného šíření chyb. Schéma RNN může vypadat např. jako na Obr. 8, kde jsou dvě vnitřní vrstvy, což z ní dělá hlubokou NS. Dle schématu by mohlo být milně vyvozováno, že se jedná hlavně o reflexivní relaci, ale cyklické uspořádání může probíhat i mezi vrstvami, a ne pouze v rámci té samé.



Obr. 8: Hluboká rekurentní NS
Zdroj: The mostly complete chart of Neural Networks [18]

Typicky se tato síť používá v analýze textu, protože pro potřebné rozhodnutí analýzy je potřeba znát širší kontext slov, případně vět. Umožňuje to právě cyklické proudění informace a vliv iterací předchozích uzlů na následující. V kalifornské laboratoři Baidu Artificial Intelligence vyvinuli nástroj Deep Voice [19] inspirovaný tradiční strukturou převodu textu na řeč, který byl nahrazen rekurentní NS. Text je nejprve převeden na foném a následně jej model audio syntézy převede na řeč.

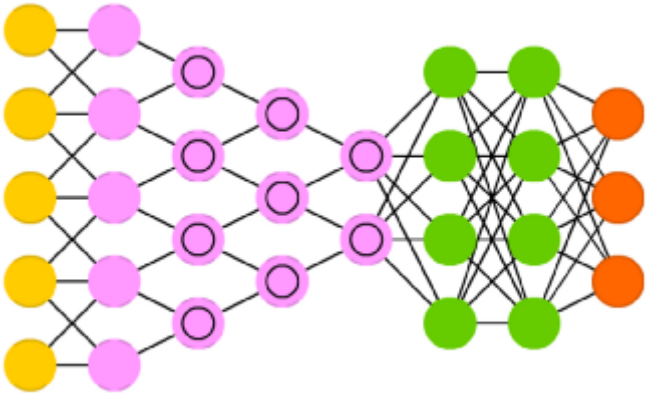
2.5.4 Konvoluční NS

Konvoluční NS je opět podobná dopředné v tom smyslu, že i zde jsou použity váhy a odchylky (anglicky bias). Odlišují se od ostatních druhů NS použitými konvolučními buňkami (nebo sdružující vrstvy) a jádra. Každá část slouží jinému účelu. Jádra ve skutečnosti zpracovávají vstupní data a sdružující vrstvy je zjednodušují, což má za cíl redukci přebytečných dat, která pro síť a výpočty nejsou potřeba.

Přední využití nachází ve zpracování signálů a obrazu, kde přebírá pozici po proslulém frameworku OpenCV. V praxi to následně funguje tak, že se určí velikost podmnožiny obrazu (např. 3x3 pixelu) a tato oblast začne být od levého horního rohu detekována (např. hrany objektu). Postupně se toto okno výběru posouvá po celém obrazu, výstupem takové vrstvy bude např. zvýrazněný obrys tělesa

a v dalších vrstvách může proběhnout třeba klasifikace objektu. Konvoluční NS je hojně využita v zemědělství, nebo při předpovědi počasí, kde probíhá analýza satelitních snímků. V zemědělství díky tomu můžeme predikovat růst a výnos konkrétních území.

Na Obr. 9 je příklad hluboké konvoluční NS, kde růžové uzly bez kroužku jsou typu jádra a s kroužkem konvoluční (sdružující), zelené jsou skryté vnitřní uzly.



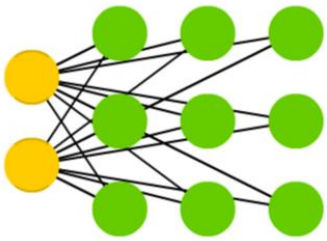
Obr. 9: Hluboká konvoluční NS
Zdroj: The mostly complete chart of Neural Networks [18]

2.5.5 Kohonenova NS

Cílem Kohonenovy NS je vložit vektory libovolného rozměru do diskrétní mapy složené z neuronů. Mapa potřebuje být dále trénována, aby vytvořila vlastní organizaci trénovacích dat. Během trénování mapy pozice neuronů zůstává konstantní, ale mění se váhy v závislosti na hodnotách. Tento samo organizující se proces má několik fází a přináší nový termín „vzdálenost k buňce“.

První je inicializace vah na nízkou hodnotu a vložení vektorů. V další fázi se hledá vítězný neuron, který je určen jako nejbližší k danému bodu. Neurony spojené s vítězným se posunou také směrem k bodu. Vzdálenost mezi bodem a neurony se počítá pomocí euklidovské vzdálenosti, přičemž neuron s nejmenší vzdáleností vyhrává. Prostřednictvím iterací jsou body seskupovány a každý neuron pak představuje druh shluku.

Kohonenovy NS jsou vhodné k rozpoznávání vzorců v datech. Použití nachází v medicinských analýzách pro kategorizaci dat, kde s vysokou přesností umí klasifikovat pacienty s onemocněním ledvin [20].



Obr. 10: Kohonenova NS
Zdroj: The mostly complete chart of Neural Networks [18]

2.6 Matematický základ

Jak již bylo dříve v této práci nastíněno, NS využívají některé základní matematické operace. Pokusím se v této kapitole alespoň povrchně nastínit jejich princip, tak abych se vyhnul dojmu „blackboxu“. Hlubší vysvětlení matematických operací by bylo nad rámec cíle této práce a k dosažení výsledku to není potřeba znát dopodrobna. Díky existujícím technologiím, jsme od těchto nízkoúrovňových základů odstíněni. Je však vhodné mít alespoň elementární ponětí.

Strojové učení využívá základní datovou strukturu tenzor. Tenzor lze chápat jako kontejner pro číselná data. Tenzory mohou mít různý počet dimenzí (v některých zdrojích se uvádí jako osy). Počet os lze zobrazit pomocí knihovny NumPy atributem ndim.

Skalár je tenzor bez dimenzionálního rozměru (0D) a má tedy 0 os. V Python knihovně NumPy lze skalár reprezentovat pomocí float. Počet os nad float hodnotou lze zjistit atributem *ndim*, kde *ndim* je rovno 0.

Jednorozměrným (1D) tenzorem je vektor a má tedy přesně jednu osu. Hodnota *ndim* je rovna 1.

2.6.1 Matice

Matice je datová struktura s dvourozměrnou dimenzí. Tento typ tenzoru má tedy 2 osy, kde první osa se typicky označuje jako řádky a druhá jako sloupce. V knihovně

NumPy se matice typicky zobrazuje jako dvourozměrné pole, kde každé vnořené pole označuje řádek a konkrétní hodnoty pole (podle indexu) jsou sloupce.

Pokud matice nabývá více rozměrů, tak se zvyšuje i dimenze samotného tenzoru. Např. pokud v knihovně NumPy matici A duplikuji na B a C, které vložím do nového pole, vznikne 3D matice, a tedy 3D tenzor. Dimenze mohou analogicky podobně narůstat dále.

$$A, B, C = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad \rightarrow \quad \begin{array}{l} \text{array([[[1, 2], [3, 4]],} \\ \text{[[1, 2], [3, 4]],} \\ \text{[[1, 2], [3, 4]]]} \end{array}$$

2.6.2 Tenzorové operace

Jelikož tenzor je nejčastěji matice, nebo n rozměrné pole, pracujeme s tenzory obdobně a využíváme běžné operace nad těmito strukturami. Často je potřeba oříznout data (např. zahození posledních sloupců matice). Data, která potom tečou do modelu je mnohdy nutné rozporcovat do dávek (anglicky batches), a ty poté postupně servírovat do sítě.

Kromě výše zmíněných úprav se na tenzory aplikují i čistě matematické operace. Mezi elementární patří sčítání, násobení, odčítání atp. V NS se často používají tzv. element-wise operace, tedy operace po prvcích. Mezi takové operace patří i funkce *relu*, která je obdobou hledání maxima mezi vybranými prvky. Výhodou podobných operací je jednoduchost implementace, není totiž nutné procházet v zanořených cyklech mezi prvky tenzorů.

Další transformační operací může být změna tvaru tenzoru, kde dochází ke změně v seskupení řádků a sloupců, tak aby byl splněn cílový tvar. Upravený tenzor musí mít samozřejmě stejný počet prvků jako počáteční.

2.6.3 Gradient

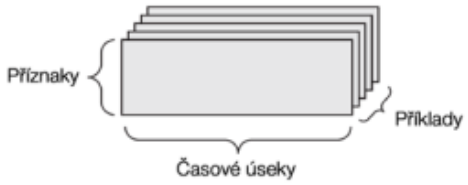
Jak již bylo dříve zmíněno, NS využívá vah k rozhodování natrénovaných dat. Síť své váhy neboli koeficienty umí optimalizovat. Dělá tak díky skutečnosti, že všechny operace použité v síti jsou diferencovatelné a lze vypočítat gradient ztrát s ohledem na koeficienty sítě. Následně díky tomu je síť schopná posunout koeficienty opačným směrem gradientu a tím se ztráta sníží.

Je-li funkce (operace) diferencovatelná, lze pak říct, že funkce má v každém bodě derivaci, typicky jsou to hladké spojité funkce. Derivací tenzorové operace je gradient. Můžeme o gradientu smýšlet jako zobecněném konceptu derivace, funkci s parametry (např. funkce s tenzorem jako jejím parametrem).

2.6.4 Tenzorová data

Praktická část se věnuje predikci cen automobilu, kde lze využít časové řady, a proto se omezím jen na popis těchto dat. Jen ve zkratce je však dobré zmínit, že vektorová data jsou uložena ve 2D tenzoru (příklady, funkce), časové řady a sekvenční data ve 3D tenzoru (příklady, časové údaje, funkce), obrázky ve 4D tenzoru (příklady, výška, šířka, kanály) a videa v 5D tenzoru (příklady, rámy, výška, šířka, kanály).

3D tenzor je vhodné použít vždy, když záleží na čase nebo pořadí hodnot. Data mohou být zakódována jako posloupnost vektorů, a proto bude dávka dat kódována jako 3D tenzor. Časové ose dle konvence patří osa s indexem 1.



Obr. 11: 3D tenzor časové řady NS
Zdroj: Deep Learning with Python **[14]**

2.7 Předpovídání a predikce cen

V této části bych rád zmínil několik pohledů na samotnou predikci ceny automobilu. Jsou to aspekty, které nelze přejít a mají vliv na samotný proces. Po této kapitole bude patrné, které atributy vozidel je potřeba znát, jaký výpočetní mechanismus se používá k určení ceny a nastínění problematiky časových řad využité v ekonomické predikci.

2.7.1 Cena automobilu

Abychom mohli určit cenu automobilu, je vhodné znát ekonomicko-právní pohled na pozorovaný subjekt. Automobil patří mezi majetek movitý.

Charakterizující vlastnosti jsou pohyblivost a přemístitelnost. Z hlediska účetnictví je však v rámci zákonných norem brán movitý majetek podobně jako nemovitý s menšími rozdílnostmi [21]. Oceňování majetku se věnuje celý samostatný obor, který zde rozebírat více nehodlám. Můžu alespoň nastínit metody využívané experty. Vycházejí totiž ze základních modelů a jako takové prezentují zjednodušení skutečností. Cílem těchto metod je určení ceny majetku a srozumitelné popsání její tvorby. V procesu oceňování hraje však roli i faktor samotného experta, jeho subjektivní pohled, zkušenosti, vnímání skutečností a celkový um. V návaznosti na kapitolu DL je patřičné, že i expert užívá zkušeností širšího světa, a ne pouze skutečnosti založené na získaných datech.

Základní přístupy vycházejí z teorie hodnoty [22]. Např. tyto tři nejpoužívanější – principy porovnávací, výnosový a princip majetkové substance.

Prvním zmíněným, je **porovnávací princip**, který je postaven na porovnání oceňovaného majetku s cenami podobného majetku, vyhodnocenými v nedávné době. Model založený na tomto principu lze matematicky popsat:

$$V = \frac{1}{n} \sum_{n}^{1} p_i \cdot k_i$$

Kde **V** je výsledná tržní hodnota majetku, $\mathbf{p_i}$ realizovaná cena podobného majetku, $\mathbf{k_i}$ koeficient vzájemné podobnosti porovnávaných majetků a **n** počet porovnávání. Obecným problémem tohoto přístupu, je určení rozsahu podobnosti porovnávaných majetků. V případě automobilů to takový problém není, protože porovnat totožné modely s podobnými atributy bude snazší než porovnat například nemovitost s odlišnými atributy navíc v odlišných lokalitách.

Druhým rozšířeným přístupem je **výnosový princip**, kde vlastník očekává generování zisku v budoucnu, typicky pronájmem. Matematicky lze tento přístup popsat následovně:

$$C_v = \frac{N}{p} \cdot 100$$

Kde $\mathbf{C_v}$ je očekávaný diskontovaný výnos, **N** předpokládaný budoucí výnos z majetku a **p** zvolená kapitalizační míra v procentech. Tahle metoda však úměrně závisí na zkušenostech oceňujícího experta, protože hodnoty N a p se v době oceňování vztahují k přítomnosti. Je potřeba aby odhadce vyhodnotil známé hodnoty

z minulosti do přítomnosti a na základě známých dat odhadnul vývoj veličin do budoucna, avšak s maximálním horizontem tří let vzhledem k tržnímu prostředí, nad tuto mez je výsledek zkreslen. Navíc tento přístup má větší zpoždění vůči nynější tržní situaci než použití porovnávacího principu.

Posledním zmíněným je **princip majetkové substance**, vhodný zejména na trhy s charakteristicky pomalým vývojem a neměnnou cenou. Matematicky se jedná pouze o součet hodnoty hmotných částí s hodnotou nehmotných částí majetku. Tato metoda tedy není vhodná pro oceňování automobilů v zemích s rozvinutou ekonomikou, použitelná je možná tak v zemích třetího světa.

Veškeré termíny, postupy a nutné výpočty k ocenění automobilu jsou definovány v doporučené technické normě Znalecký standart č.1/2005 vydaný Ústavem soudního inženýrství v Brně.

2.7.2 Časové řady

Většina statistických metod je zaměřena na použití v nezávislých experimentech nebo ve výsledcích průzkumů, kde uspořádání pozorování nemá žádný zvláštní význam (např. biologie, agronomie, sociologie). V ekonomii mají data často podobu sekvencí pozorování jedné nebo více proměnných pořízených k určitému datu, avšak takové pozorování nemůže být považováno za nezávislé. Základní definice časové řady říká, že se jedná o jakoukoliv konečnou, nebo nekonečnou sekvenci pozorování seřazenou v čase $(X_t : t \in T)$ [23].

Časové řady lze dělit podle různých kritérií. Mezi ně patří např. doba měření a náhodnost. Krátkodobé časové řady jsou z pravidla kratší úsek do jednoho roku. Analogicky ty dlouhodobé zaznamenávají úsek delší než rok. Kromě časového intervalu pozorování, se často dělí řady podle „detailnosti“. Tedy jestli je řada spojitá nebo diskrétní.

Spojitá časová řada obsahuje hodnotu proměnné X_t nepřetržitě (např. naměřené fyzikální hodnoty). Časový index t může nabývat veškerých hodnot v daném intervalu a oboru reálných čísel. Ve světě ekonomie jsou však takové časové řady poměrně vzácné.

Diskrétní časové řady jsou oproti spojitým běžněji používané. Zejména v ekonomii. Prvky časové řady jsou diskrétní tehdy, když soubor možných hodnot t je

diskrétní množinou, tedy T je podmnožinou celých čísel. Dále se diskrétní časové řady dají rozlišit v závislosti na reprezentaci pozorování. Prvním typem může být sledování okamžité úrovně (cena, objem skladu). Druhým typem mohou být kumulované toky za určité období (příjem, spotřeba).

2.8 Existující práce

Predikce cen automobilů za pomocí AI je docela rozšířené téma, jemuž se věnovalo mnoho výzkumů. Už v roce 2009 zveřejnila v Hamburgu svou diplomovou práci Mariana Listiani [24]. Ve svém výzkumu se věnovala využitím regresního modelu vytvořeného pomocí SVM (Support Vector Machines), který dovedl predikovat cenu automobilu, která byla pronajímána s větší přesností, než umožnila vícerozměrná regrese, nebo jednoduchá vícenásobná regrese. Důvodem je, že využití SVM je vhodnější při řešení datových sad s více rozměry a je méně náchylné k chybám vedoucím k underfitting či overfitting.

Další práce [25] z roku 2011 byla postavena na modelu BP NN (Back Propagation Neural Network) pro predikci ceny použitého automobilu. Model bral v potaz několik atributů: ujeté kilometry, odhadovanou životnost a značku automobilu. Navržený model byl postaven tak, aby se mohl vypořádat s nelineárními vztahy v datech, což nebyl případ předchozí práce s regresním modelem a SVM. Nelineární model dokázal lépe předpovídat ceny automobilů s větší přesností než jiné lineární modely.

V další práci [26] byly použity různé algoritmy z oblasti ML. Jmenovitě k-nejbližší sousedé, vícenásobná lineární regresní analýza, rozhodovací stromy a naivní Bayes pro predikci cen automobilů na Mauriciu. Soubor dat použitý pro predikční modely byl manuálně získán z tamních novin pouze z rozmezí jednoho měsíce, poněvadž časová prodleva může mít dopad na celý výsledek. Použity byly následující atributy – značka, model, objem motoru, počet najetých km, rok výroby, barva exteriéru, typ řazení a cena. Výsledek práce odhalil, že metody rozhodovací strom a naivní Bayes nebyly schopny predikovat a klasifikovat numerické hodnoty. Navíc, omezená velikost testovací sady nemohla mít vysokou klasifikační výkonnost (údajná přesnost menší než 70 %).

Poslední práce [27] opět využívá model postavený na vícenásobné lineární regresi pro predikci cen aut. Sada dat byla sesbírána během dvouměsíčního intervalu a obsahoval následující atributy – cena, objem motoru, barva exteriéru, datum publikování inzerátu, počet zobrazení reklamy, posilovač řízení, počet najetých km, typ ráfků, druh převodovky, typ motoru, město prodeje, město registrace, model, značka, verze a rok výroby. Se zvoleným nastavením autoři dosáhli vysoké přesnosti 98 %.

Ve všech výše zmíněných pracích autoři použili pro predikční model jediný algoritmus ML založený na regresi, nebo neuronovou síť se zpětnou propagací. Modely se svými výsledky odlišují v přesnosti, což je důsledek nevhodně zvolených modelů, algoritmů a v neposlední řadě datových sad pro trénování AI. Proto bych rád zvolil model ANN (Artificial NN), který by měl lépe reflektovat nelineární vztahy mezi trénovacími daty. Trénovací data pro predikční model budou sesbíraná z českého prodejního portálu www.sauto.cz.

2.9 Možnosti implementace v Pythonu

TensorFlow je end-to-end open source platforma pro machine learning. Disponuje komplexním flexibilním ekosystémem nástrojů, knihoven a komunitních zdrojů, který umožňuje výzkumníkům použít nejmodernější technologie v ML a vývojářům snadno vytvářet a vydávat aplikace poháněné ML. [28]

Další možností je použít framework **Keras**. Jeho API rozhraní je určené pro deep learning psaný v Pythonu, který funguje na vrchní části vrstvy ML platformy TensorFlow. Byl vyvinut se záměrem umožnit rychlé experimentování a odstínit uživatele od zbytečné nízko úrovňové realizace. Vizí frameworku je myšlenka, že klíčem k dobrému výzkumu je schopnost být co nejrychleji schopný přejít od nápadu k výsledku [29].

V praktické části této práce se věnuji implementaci pomocí frameworku Keras. Jak již bylo nastíněno dříve, jsou potřeba i různé knihovny. Pro matematické metody je vhodné využít **NumPy**. Kromě potřebných operací poskytuje i potřebné datové struktury. Knihovna je vysoce optimalizovaná pro maximální využití výpočetního výkonu a jeho jádro je psáno v jazyce C [30].

Velice žádanou knihovnou je **Pandas**. Poskytuje rychlý, výkonný, flexibilní a snadno použitelný open source nástroj pro analýzu a manipulaci s daty v Pythonu.

Dále existují další rozsáhlejší projekty a sady nástrojů. Jedním z nich je SciPy, což je ucelený ekosystém potřebných nástrojů. Mimo jiné obsahuje zmíněné NumPy a Pandas, dále však **Matplotlib** pro vykreslování grafů, interaktivní python konzoli, SymPy a další [31]. Alternativou SciPy je do jisté míry podobný **Scikit-learn**, který přímo obsahuje SciPy a další vhodné nástroje a knihovny [32].

Modely umělých neuronových sítí mohou být implementovány a spouštěny v prostředí Jupyter Notebook [33]. Jedná se o open source webovou aplikaci, která umožňuje vytvářet a sdílet dokumenty obsahující živý kód, rovnice, vizualizace atd. Výhodou je také průběžný minimalizovaný náhled do datových struktur, během transformací.

3 Praktická část

Cílem této práce je využít hluboké učení k predikování ceny automobilu. Je tedy potřeba nastavit model NS pomocí Keras frameworku, model nechat trénovat na známých datech a pak následně nechat predikovat cenu konkrétního vozu. Výstupem je predikovaná aktuální cena daného vozu. Např. uvažujme scénář, kde chci prodat vůz Škoda Octavia o určité motorizaci a kondici. Jako prodejce mě bude zajímat cena, za kterou budu moci automobil inzerovat k prodeji. Protože znám parametry prodávaného vozu i postup, jak je zakódovat, tak aby model s takovým vstupem uměl pracovat a predikovat cenu, data daného vozu nachystám a pošlu na vstup sítě.

Aby však algoritmus mohl začít trénovat, potřebuje sadu dat. Pro tento účel jsem vytvořil web scrape nástroj, jehož cílem je stáhnout v zeštíhlené podobě veškeré záznamy z prodejního portálu www.sauto.cz.

V poslední fázi jsou data použita v python skriptu, kde se transformují skrze neuronovou síť. Jediným výstupem sítě je aktuální cena daného vozidla, kterou mohu použít k prodeji daného vozidla (viz příklad ze začátku kapitoly).

Nové řešení spočívá v ucelené sadě programů, které jsou vhodné pro predikci cen automobilů na českém trhu. Případ užití je znázorněn na následujícím diagramu 2.

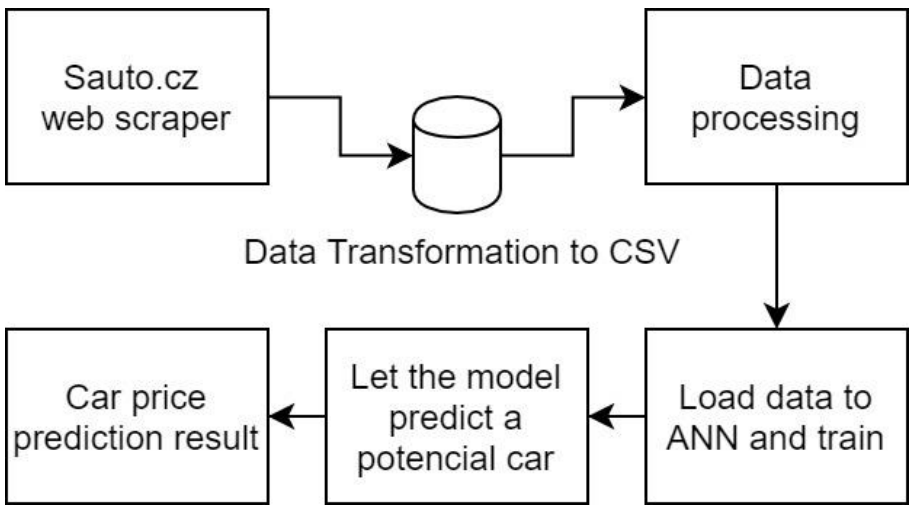


Diagram 2: Vysoko úrovňový pohled na celý proces
Zdroj: vlastní zpracování

Jak již bylo výše zmíněno, nástroje Carscraper a Carpricer, jsem naimplementoval v Pythonu (verze 3.6). Jsou multiplatformní a kdokoliv s přístupem na GitLab [34], si může zdrojové kódy naklonovat a spustit. Repositáře Carscraper jsou dostupné zde:

https://gitlab.com/zedbe/carscraper

https://gitlab.com/zedbe/carpricer

3.1 Příprava dat

Scraping [35] nástroj napsaný v Pythonu jsem pojmenoval Carscraper. Jeho úkolem je projít postupně všechny stránky s inzercí osobních automobilů, které jsou buď nové, předváděcí, nebo ojeté. Sada dat by neměla obsahovat tedy např. nabouraná auta, nebo vozy prodávané na náhradní díly.

Každá strana obsahuje maximálně 15 automobilů. Z každé strany tak o každém automobilu Carscraper získá následující atributy: **značka**, **model**, **rok výroby**, **nájezd** (kilometry), **cena** (česká koruna), **typ motoru** (podle paliva) a vygenerovanou **url** pro případnou manuální kontrolu, nebo budoucí detailnější scrapování.

značka	model	rok výroby	nájezd	cena	palivo	url
Alfa Romeo	145	1997	208700	39000	Benzín	http://...
Alfa Romeo	146	1997	245940	12800	CNG+benzín	http://...
Alfa Romeo	146	2000	324258	14000	Nafta	http://...
Alfa Romeo	146	1998	200000	39000	Benzín	http://...
…	…	…	…	…	…	…

Tabulka 1: Vizuální podoba vzorku scrapovaných dat
Zdroj: vlastní zpracování

Carscraper šetří spoustu času, jelikož projít manuálně přibližně 90.000 inzerátů by zabralo výrazně delší čas než nechat puštěný tento nástroj, kterému podobné kvantum vzorků trvá sesbírat bez paralelního zpracování kolem 30 min. Nástroj během svého procesu zobrazuje do terminálu ukazatel úspěchu (anglicky progress bar) s procenty i poměrem zpracovaných značek vůči celku. Až Carscraper dokončí svou práci, uloží shromážděná data do *.csv souboru.

Během získávání dat může uživatel vidět „progress bar“ vystihující v jaké fázi se nachází.

Obr. 12: Progress bar sloužící jako zpětná vazba programu uživateli
Zdroj: vlastní zpracování

3.1.1 Carscraper implementace

Carscraper je zároveň submodulem [36] Carpricer repositáře. Je tedy zbytečné klonovat repositáře separovaně.

Pro spuštění scrapovacího nástroje je potřeba mít pro python interpreta následující knihovny:

```
import requests
import pandas as pd
from datetime import datetime, timedelta
import os
from carscraper.create_logger import create_logger
```

Poslední importovaná funkce `create_logger` je pomocná ve vedlejším stejnojmenném souboru. Vrací vytvořenou instanci loggeru z knihovny `logging` [37], která je k dispozici, aby mohla být využita. `Logger` má v sobě nastavený formát i úroveň zaznamenávání údajů. Kromě toho má nastavený i `FileHandler` pro zápis do souboru na určité místo. Výstupem Carscraperu jsou 2 textové soubory. Jeden s příponou *.log a druhý s příponou *.csv. Z místa, kde se spustí, vytvoří adresář outputs. V něm zanořený další adresář s označením data spuštění a uvnitř konečně 2 zmíněné soubory. Soubory mají stejný název jako adresář, v němž se

nacházejí. Pouze s přidaným sufixem znázorňující čas spuštění, tak aby bylo možné rozpoznat vydolovaná data, pokud se nástroj spustí vícekrát během dne. V *.csv souboru jsou data potřebná k predikci a v *.log souboru se nachází údaje z běhu nástroje (ID procesu / vlákna, časové razítko, úroveň logování, modul, čas spuštění a dokončení, celkový čas dolování dat). Samozřejmě pokud adresářová cesta již existuje, soubory se do ní jen uloží.

Jakmile je otázka souborového systému vyřešena, spustí se dolování zavoláním:

```
brands = list_dict_2_list_list(extract_brands())
```

Metoda `extract_brands` obsahuje první request na server. Response serveru se převede do json formátu, následně jsou dostupné informace o značce automobilky. Pokud nejsou žádné inzeráty dané značky, tak se do výsledných dat značka ani nedostane a pokračuje další značkou. Celé zpracování značek probíhá objektově v následující hierarchii (viz Diagram 3).

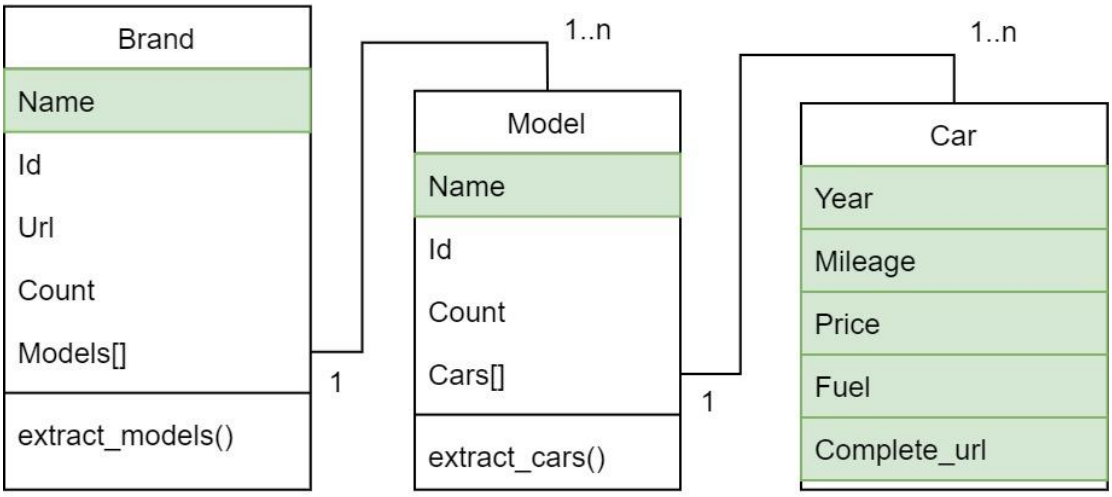


Diagram 3: Diagram tříd popisující strukturu i dolované atributy
Zdroj: vlastní zpracování

Jakmile program získá od serveru potřebné údaje o značce, zavolá konstruktor třídy **`Brand`** a vytvoří její instanci k dané značce. V konstruktoru se zároveň volá metoda `extract_models` (v cyklu vytváří instance třídy **`Model`** a přidává do pole `Models`). Princip se opakuje a v konstruktoru se volá metoda `extract_cars`, která

volá konstruktor třídy **Car**. Vzniknou tedy instance aut odpovídající danému inzerátu a spadající do modelového označení dané značky a předají se do pole `Cars`.

Poslední zmíněná extrakční metoda prochází se zadaným filtrem stránku po stránce, kde je maximálně zobrazeno 15 inzerátů (počet nelze navýšit). Zjistil jsem, že webová stránka je schopná zobrazit maximálně 667 stránek, a na www.sauto.cz není možné tedy postupně dojít až na stranu 668, i když další inzeráty v daném filtru nejsou zobrazeny všechny. Je to snadno zjistitelné např. při procházení webu bez filtrování značky a modelu, jelikož portál obsahuje přibližně 90.000 inzerátů osobních automobilů, ale bez užšího filtrování postupným stránkováním je schopná zobrazit pouze 15*667 = 10.005 inzerátů. Proto jsem tedy nucen posílat více requestů až na úroveň s filtrováním konkrétních modelů. Nejprodávanější model Škoda Octavia má přibližně 8.000 inzerátů a nehrozí tedy u tohoto modelu, že by nástroj nevydoloval veškeré inzeráty z důvodu překročení možného limitu zobrazení.

Diagram 3 navíc vizuálně zobrazuje cílové atributy zelenou barvou. Když se Carscraper vyskytuje v daném stavu, kdy má k dispozici pole kompletních objektů s daty, převádí jej metodou `list_dict_2_list_list` v trojrozměrném cyklu z pole objektů na dvourozměrné pole. Každé pole v poli vystihuje právě jeden inzerát (viz Tabulka 1, odpovídá 1 řádek).

```
brands = list_dict_2_list_list(extract_brands())
columns = ["brand", "model", "year", "mileage", "price", "fuel",
"url"]
df = pd.DataFrame(brands, columns=columns)
df.to_csv('./{0}/{1}/{2}.csv'.format(output_root,
                                     output_dir, output_file))
```

Tuto strukturu v proměnné `brands` již knihovna `pandas` umí přetvořit do dataframu [38] s pomocí proměnné `columns`, která v poli definuje označení sloupců. Vytvoření dataframe snadno uloží do výstupního souboru *.csv.

3.2 Návrh prototypu predikování ceny

Druhým implementovaným nástrojem a zároveň klíčovým v této práci, je Carpricer. Napsaný je též v Pythonu a lze jej spustit přímo v jupyter notebooku. Jeho úkolem je načíst sesbíraná data (jako dataframe) z cesty, ve které je již uložený *.csv soubor. Následně z dataframe odstraní nepotřebné sloupce (číslování a url). V této fázi má model nachystaná veškerá data, ale ještě se všemi nebude moct pracovat v nečíselné podobě. Carpricer tedy rovnou rozdělí data určené pro osu X, což jsou veškeré údaje kromě cen, které připadnou ose Y. Následně přetransformuje sloupec označující značku, model a typ motoru (kategoricky rozlišené, např. veškeré vozy Škoda budou mít číselnou hodnotu 84). Nyní jsou data nachystaná k trénování a implementovaný model s nimi bude moci pracovat a hledat vztahy mezi jejich hodnotami. Dále je potřeba vytvořit sekvenční model [39], nad daným modelem vytvořit vrstvy a nakonfigurovat.

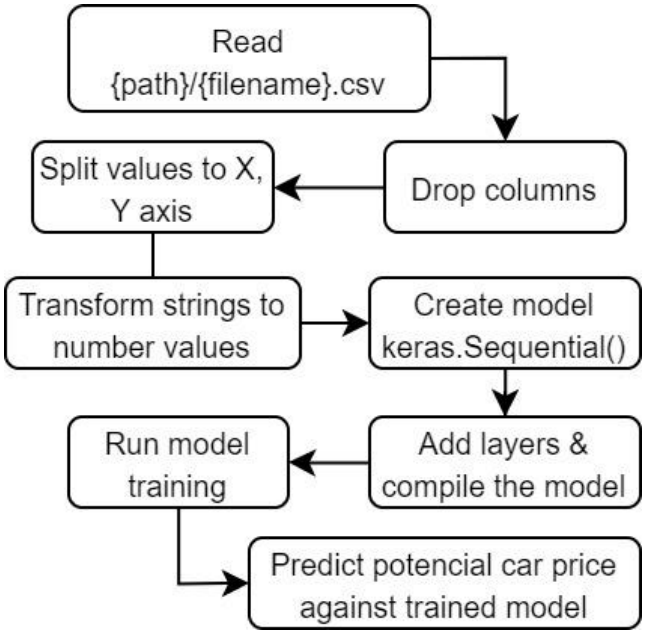


Diagram 4: Flowchart definující nástroj Carpricer
Zdroj: vlastní zpracování

Data i model jsou k dispozici, proto se může spustit trénování modelu nad nachystanou sadou dat. Po určité době (závislé na nastavení modelu a trénování) je model schopný přijmout nová data, v našem případě znázorňující potenciální auto v dané podobě a predikovat jeho aktuální hodnotu.

Po skončení predikce nového automobilu modelem získáváme konečný výsledek. Tedy odhad ceny s určitou chybou.

3.2.1 Carpricer implementace

Carpricer pokračuje tam, kde Carscraper skončil. Pomocí pandas otevře *.csv s vydolovanými daty:

```
df = pd.read_csv(path)
```

Jak již bylo zmíněno v předchozí kapitole, používám sekvenční model frameworku Keras, kde postupně přidávám vrstvy od vstupní až po výstupní. U první vstupní vrstvy definuji dimenzi výstupu z vrstvy, aktivační funkci typu RELU [40] a vstupní dimenzi. U druhé vrstvy už není potřeba definovat vstupní dimenzi, jelikož to rozpozná sama. U poslední výstupní vrstvy platí to, co u druhé, navíc není potřeba aktivační funkce, protože je ryze výstupní, tedy definuji jako jednodimenzionální. Pro nakonfigurování sítě použiji metodu `Compile`, s optimalizační funkcí `adam` [41] a ztrátovou funkcí `mean_squared_error` [42].

```
model.add(keras.layers.Dense(5, activation='relu',
                             input_shape=(5, )))
model.add(keras.layers.Dense(5, activation='relu'))
model.add(keras.layers.Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')
```

Poté stačí model natrénovat nad nasbíranými daty. K tomu slouží metoda fit. První dva argumenty přijmou data z X a Y os. Třetím argumentem nastavuji počet iterací trénování a posledním čtvrtým callback s předčasným zastavením, pokud se 5 iterací za sebou výsledky mění nevýrazně.

```
model.fit(X1, Y1, epochs=30,
          callbacks=[keras.callbacks.EarlyStopping(patience=5)])
```

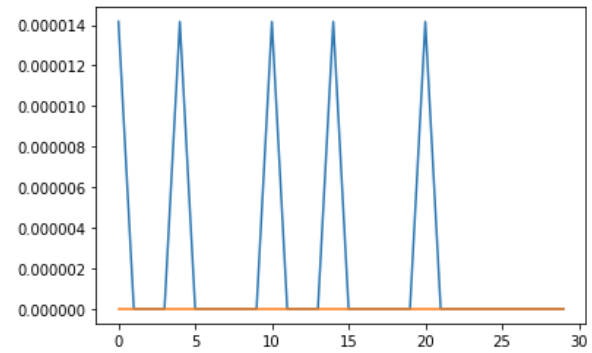
Posledním krokem je nachystání testovaného vozu, kterému chceme predikovat jeho cenu. Pomocí knihovny `numpy` [43] nachystám nová vstupní data pro predikci a předám metodě `predict`, kde druhý argument `batch_size` označuje velikost vzorku (1).

```
import numpy as np

test_car = np.array([84, 552, 2016, 58233, 6])
print(model.predict(test_car.reshape(1, 5), batch_size=1))
```

3.2.2 Validace prototypu

Pro celkovou analýzu, jak se model úspěšně trénuje nad daty jsem použil metodu `train_test_split` [44] na rozdělení nasbírané sady dat z knihovny `sklearn` s poměrem rozdělení dat do 80% trénovacích vůči 20% testovacích. Bohužel výsledek přesnosti vůči ztrátovosti se mi nepodařilo změřit s vypovídající hodnotou (viz Obr. 13). Osa X odpovídá iteracím epoch a osa Y údajné přesnosti modelu, která by byla 0.




```
history.history['val_acc'][-1]
```

```
0.0
```

Obr. 13: Měření metriky přesnosti nad validačními daty
Zdroj: vlastní zpracování

3.2.3 Predikce prototypu

Pro predikci nového vozu je potřeba znát jeho údaje [**značka**, **model**, **rok výroby**, **počet najetých km**, **typ motoru**] a „zakódovat“ tak, aby první, druhá a pátá

hodnota odpovídala kategoriím použitých v trénování modelu. Viz závěr předchozí kapitoly. V tomto případě se tedy jedná o Škoda (84) Octavia (552) rok výroby 2016 s nájezdem 58233 km a naftovým motorem (6).

Výsledkem mi je: [[**365646.5**]]. Protože se jedná o inzerát, který jsem umístil mezi testovanými daty (poslední řádky polí), měl mít hodnotu 235.000,- CZK.

```
array([['Alfa Romeo', '145', 1997, 208700, 'benzín'],
       ['Alfa Romeo', '146', 1997, 245940, 'CNG + benzín'],
       ['Alfa Romeo', '146', 2000, 324258, 'nafta'],
       ...,
       ['Wartburg', '353', 1989, 9000, 'benzín'],
       ['ZhiDou', 'D2', 2018, 10, 'elektro'],
       ['Škoda', 'Octavia', 2016, 58233, 'nafta']], dtype=object)
array([[ 39000],
       [ 12800],
       [ 14000],
       ...,
       [ 55000],
       [449900],
       [235000]], dtype=int64)
```

Obr. 14: Zaznamenaný inzerát před „zakódováním“
Zdroj: vlastní zpracování

3.3 Úprava prototypu

Z předešlých kapitol je patrné, že výsledky nejsou optimální a je potřeba zlepšit prediktivní model a lépe vyhodnotit chybovost výsledků.

Auto	palivo	…
Škoda Octavia	benzín	…
Škoda Superb	nafta	…
Suzuki Vitara	hybrid	…
…	…	…

Tabulka 2: Kategorizace automobilů podle druhu paliva
Zdroj: vlastní zpracování

Dalším problémem se jeví tzv. **fiktivní proměnná** (anglicky dummy variable) [45]. Obecně lze proměnné, tedy sledované hodnoty ze sady dat rozdělit mezi kvantitativní a kvalitativní (někdy uváděné jako kategorické). **Kvantitativní**

proměnné jsou typicky vyčíslitelné hodnoty. V případě této práce se jedná o najeté kilometry, ceny, a stáří vozu. Naopak **kvalitativní** proměnné jsou ty, které vystihují nějakou kategorii. Opět se v této práci jedná o určení značky automobilu, modelu a druh paliva (viz Tabulka 2). Fiktivní proměnná je termín, který se používá v souvislosti s kvalitativními hodnotami. Fiktivní je proto, že místo označení kategorie ve formě řetězce, získá číselnou hodnotu, se kterou může model sítě daleko snáz pracovat. Transformace by následně vypadala viz Tabulka 3.

auto	benzín	nafta	hybrid	…
Škoda Octavia	1	0	0	…
Škoda Superb	0	1	0	…
Suzuki Vitara	0	0	1	…
…	…	…	…	…

Tabulka 3: Převod kategorizovaných informací na fiktivní proměnné
Zdroj: vlastní zpracování

V Carpricer prototypu již takový mechanismus byl částečně implementován pomocí `LabelEncoder` [46]. Avšak je vhodné následně použít také `OneHotEncoder` [47], který v prototypu chyběl. Díky kombinaci těchto encoderů, lze předejít situaci tzv. pasti fiktivní proměnné (anglicky dummy variable trap). Jedná se o scénář, kde jsou nezávislé proměnné multikolineární. Pokud jsou proměnné multikolineární, znamená to, že jsou vysoce korelovány a lze tedy jednu proměnnou predikovat na základě ostatních. Řešení spočívá v tom, že můžeme jednu fiktivní proměnnou zrušit, protože jsme schopni zrušenou hodnotu zjistit na základě hodnot ostatních. Tedy např. z tabulky 3 bychom mohli zrušit sloupec s hodnotou hybrid (pokud by nebyli další kategorie) a informaci, zda se jedná o hybridní automobil, zjistíme z hodnot benzín = 0 a nafta = 0. Lze odstranit jakoukoliv z fiktivních proměnných. Vždy musíme pamatovat na základní pravidlo, že počet fiktivních proměnných nezbytných k reprezentaci jediné kategorie, je rovný počtu fiktivních proměnných - 1. Např. pokud tedy budu mít v kategorii paliva pouze 3 druhy (benzín, nafta, hybrid), celkový počet mohu zredukovat na 2.

Zmíněné řešení vede tedy ke snížení redundance dat a snížení standartní chyby. Samotná kolinearita proměnných je velice důležitá ve fázi verifikace modelu. Pokud jsou proměnné modelu multikolineární mohou vzniknout pochybnosti o specifikaci modelu a relevantnosti proměnných.

Další úprava se týká samotné vstupní sady dat. Prvotní sada postrádá údaj o převodovce. Tato informace (`advert_gearbox_cb`) je snadno dosažitelná z odpovědí serverů portálu www.sauto.cz bez potřeby posílání dodatečných požadavků. Je to natolik důležitá a relevantní informace ovlivňující cenu automobilu, že je nutné doplnit její získání do Carpricer a spustit dodatečné scrapování.

3.3.1 Implementace úprav

Na základě dříve navržených úprav predikčního modelu je potřeba začít úpravou Carscraper. V již zanořené struktuře požadavku na server stačí přidat vybranou proměnnou (`advert_gearbox_cb`), která může podle interního výčtu nabývat jednu z hodnot 0, 1, 2, 3. Každá z těchto číselných hodnot symbolizuje kategorii dle www.sauto.cz slovníku (codebook) viz následující obrázek. Převodovka s označením 0 nemá u inzerátu vyplněný údaj (proto undefined hodnota v posledním sloupci).

```
gearbox_types=df.gearbox.unique()
gearbox_types
```

```
array([1, 0, 3, 2], dtype=int64)
```

```
counts=df.groupby('gearbox').size().reset_index(name='counts')
counts['sauto.cz codebook'] = ['undefined', 'manual', 'semi automatic', 'automatic']
counts
```

	gearbox	counts	sauto.cz codebook
0	0	2545	undefined
1	1	50495	manual
2	2	38	semi automatic
3	3	24111	automatic

Obr. 15: Výstup se zjištěnými kategoriemi, dekódovaným značením a počty aut dle typu převodovky
Zdroj: vlastní zpracování

Vzhledem ke zjištěným faktům, je patrné, že nezanedbatelné množství inzerátů nemá vyplněný druh převodovky (2545 vozidel) a poloautomatickou převodovku má zanedbatelný počet inzerátů (38 vozidel). Manuální převodovku má většina vozidel (50495 vozidel) a automatickou necelá polovina (24111 vozidel). Poloautomatické převodovky jsou v dnešní době spíše historickým přežitkem a samotný počet by mohl zbytečně ovlivnit výsledný model, proto této množině vozidel zaměním typ převodovky na manuální. U vozidel s nedefinovanou převodovkou měním typ také na manuální převodovku z prostého důvodu, kterým je přidaná hodnota vozidla v případě, že má automatickou převodovku. Za vozidla s automatickou převodovkou se běžně připlácí a inzerenti by byli sami proti sobě, pokud by opomněli tento typ vyplnit. Předpokládám tedy, že jim na navýšení hodnoty inzerovaného vozu nesejde a výchozím typem řazení zůstane manuální. Transformace vypadá následovně pomocí metody `replace` z knihovny Pandas [48].

```
df.gearbox = df.gearbox.replace(to_replace=[0, 2], value=1)
df.gearbox = df.gearbox.replace(to_replace=3, value=0)

counts=df.groupby('gearbox').size().reset_index(name='counts')
counts['Carpricer codebook'] = ['automatic', 'manual']
counts
```

	gearbox	counts	Carpricer codebook
0	0	24111	automatic
1	1	53078	manual

Obr. 16: Transformace typů převodovek se změnou značení a následným ověřením počtu vozidel
Zdroj: vlastní zpracování

Z provedené transformace je patrné, že jsem využil uvolnění kategorie s hodnotou 0 pro vozidla s automatickou převodovkou. Udělal jsem tak z důvodu nastínění „dummy variable trap“ mechanismu. Jelikož jsem zredukoval 4 kategorie označující druh převodovky na 2, dále vím, že po aplikaci mechanismu získám jedinou fiktivní proměnnou (počet = 2-1), která bude nabývat binární hodnoty (např. pro automatiku = 0 a manuál = 1).

Knihovna sklearn od verze 0.22 změnila svou funkcionalitu a je pro daný záměr (vedle již zmíněných enkodérů) nutné použít třídu `ColumnTransformer` [49], kde jedním z parametrů je `OneHotEncoder`. Díky této aktualizaci již není nutné, aby této transformaci předcházelo využití `LabelEncoder`, který umí hodnoty převést na číselné hodnoty. Implementace vypadá následovně.

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
ct = ColumnTransformer([("gearbox", OneHotEncoder(drop='first'), [5])],
                       remainder='passthrough')
X1 = ct.fit_transform(X1)
X1
```

```
array([[1.0, 'Alfa Romeo', '145', 2000, 173335, 'benzín'],
       [1.0, 'Alfa Romeo', '145', 1997, 208700, 'benzín'],
       [1.0, 'Alfa Romeo', '146', 1997, 245940, 'CNG + benzín'],
       ...,
       [1.0, 'UAZ', 'Ostatní', 1977, 2157, 'benzín'],
       [1.0, 'Wartburg', '353', 1974, 33946, 'benzín'],
       [0.0, 'ZhiDou', 'D2', 2018, 10, 'elektro']], dtype=object)
```

Obr. 17: „Dummy variable trap“ mechanismus aplikován na typ řazení
Zdroj: vlastní zpracování

Po transformaci sloupce označující druh řazení, je sloupec přesunut na první pozici. Navíc díky parametru `drop='first'` se zahodila první fiktivní proměnná označující automatickou převodovku. Z toho vyplývá, že manuální převodovku lze najít pod hodnotou 1 a automatickou pod 0. Past aplikuji stejně i na sloupec označující druh paliva. Druhů paliva a možných kombinací je celkem 7, jak je vidět nad výčtem mojí datové sady.

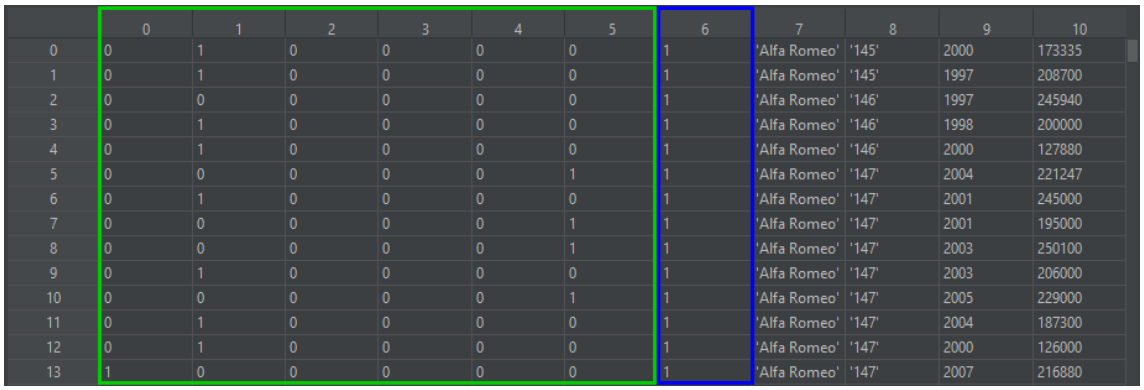
```
fuel_types=df.fuel.unique()
fuel_types
```

```
array(['benzín', 'CNG + benzín', 'nafta', 'LPG + benzín', 'hybridní',
       'elektro', 'ethanol'], dtype=object)
```

Obr. 18: Výčet druhů paliva z datové sady
Zdroj: vlastní zpracování

Po samotné transformaci na fiktivní proměnné a následné ztrátě jedné z nich, vypadají data ve vědeckém režimu PyCharm jako na Obr. 19. Sloupec s indexem 6

obklopený modrou barvou znázorňuje již známý druh převodovky. Nově však vznikly sloupce ohraničené zelené barvou s indexy 0-5 reprezentující druh paliva inzerovaných vozidel. Po opětovném aplikování mechanismu pasti jsem zredukoval počet druhů paliva ze 7 na 6.



	0	1	2	3	4	5	6	7	8	9	10
0	0	1	0	0	0	0	1	'Alfa Romeo'	'145'	2000	173335
1	0	1	0	0	0	0	1	'Alfa Romeo'	'145'	1997	208700
2	0	0	0	0	0	0	1	'Alfa Romeo'	'146'	1997	245940
3	0	1	0	0	0	0	1	'Alfa Romeo'	'146'	1998	200000
4	0	1	0	0	0	0	1	'Alfa Romeo'	'146'	2000	127880
5	0	0	0	0	0	1	1	'Alfa Romeo'	'147'	2004	221247
6	0	1	0	0	0	0	1	'Alfa Romeo'	'147'	2001	245000
7	0	0	0	0	0	1	1	'Alfa Romeo'	'147'	2001	195000
8	0	0	0	0	0	1	1	'Alfa Romeo'	'147'	2003	250100
9	0	1	0	0	0	0	1	'Alfa Romeo'	'147'	2003	206000
10	0	0	0	0	0	1	1	'Alfa Romeo'	'147'	2005	229000
11	0	1	0	0	0	0	1	'Alfa Romeo'	'147'	2004	187300
12	0	1	0	0	0	0	1	'Alfa Romeo'	'147'	2000	126000
13	1	0	0	0	0	0	1	'Alfa Romeo'	'147'	2007	216880

Obr. 19: Náhled na transformovanou datovou sadu se 13 vzorky ve vědeckém režimu PyCharm
Zdroj: vlastní zpracování

Pro lepší přehlednost datové struktury a podobnost s původní datovou sadou, přesouvám obě kategorie na pozice posledních sloupců. N rozměrné pole pod proměnnou X1, kterou jsem do teď měnil, si převádím zpět do dataframe, jelikož se s touto datovou strukturou knihovny Pandas pracuje lépe, pokud potřebuji měnit pořadí sloupců. Nejprve potřebuji vrátit stávající označení sloupců (proměnná `old_cols`), abych nemusel pracovat s indexy, což by velice pravděpodobně vedlo k chybě. Následně stačí pohodlně předat novou sekvenci názvů sloupců (proměnná `new_cols`) a obsah datové struktury se přeskládá.

```
from pandas import DataFrame
old_cols=["fuel_type1", "fuel_type2", "fuel_type3",
          "fuel_type4", "fuel_type5", "fuel_type6",
          "gearbox", "brand", "model", "year", "mileage"]
new_cols=["brand", "model", "year", "mileage", "fuel_type1",
          "fuel_type2", "fuel_type3", "fuel_type4",
          "fuel_type5", "fuel_type6", "gearbox"]
df_arranged = DataFrame(data=X1, columns=old_cols)
df_arranged = df_arranged[new_cols]
```

Přeskládaný dataframe znovu převedu na n rozměrné pole se ztrátou popisu sloupců. Sloupce nesoucí informaci o značce a modelu automobilu (typ řetězce) transformuji pomocí třídy `LabelEncoder` na číselné hodnoty.

```
from sklearn.preprocessing import LabelEncoder
X = df_arranged.iloc[:, 0:].values
leX = LabelEncoder()
X[:, 0] = leX.fit_transform(X[:, 0])
X[:, 1] = leX.fit_transform(X[:, 1])
```

Po všech úpravách a optimalizaci dat získávám následující strukturu (viz Obr. 20). Sloupce s označením značky a modelu jsou převedeny do číselné kategorie. Stejně tak i sloupce s označením paliva a druhu řazení, kde navíc proběhla aplikace „dummy variable trap“ s redukcí n-1 fiktivních proměnných.

brand	model	year	mileage	fuel_type1	fuel_type2	fuel_type3	fuel_type4	fuel_type5	fuel_type6	gearbox
3	16	2000	173335	0.0	1.0	0.0	0.0	0.0	0.0	1.0
3	16	1997	208700	0.0	1.0	0.0	0.0	0.0	0.0	1.0
3	17	1997	245940	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	17	1998	200000	0.0	1.0	0.0	0.0	0.0	0.0	1.0
3	17	2000	127880	0.0	1.0	0.0	0.0	0.0	0.0	1.0
3	18	2004	221247	0.0	0.0	0.0	0.0	0.0	1.0	1.0
3	18	2001	245000	0.0	1.0	0.0	0.0	0.0	0.0	1.0
3	18	2001	195000	0.0	0.0	0.0	0.0	0.0	1.0	1.0
3	18	2003	250100	0.0	0.0	0.0	0.0	0.0	1.0	1.0
3	18	2003	206000	0.0	1.0	0.0	0.0	0.0	0.0	1.0
3	18	2005	229000	0.0	0.0	0.0	0.0	0.0	1.0	1.0
3	18	2004	187300	0.0	1.0	0.0	0.0	0.0	0.0	1.0
3	18	2000	126000	0.0	1.0	0.0	0.0	0.0	0.0	1.0
3	18	2007	216880	1.0	0.0	0.0	0.0	0.0	0.0	1.0

Obr. 20: Reprezentace 13 vzorků datové struktury po transformacích ve vědeckém režimu PyCharm
Zdroj: vlastní zpracování

Síť by tuto naformátovanou datovou sadu v podobě tenzorů už mohla absorbovat a začít se nad ní trénovat. Pořád ale data nejsou tolik optimální a rozdělená na trénovací a testovací. Jak již bylo řečeno u prototypu, k rozdělení dat použiji metodu `train_test_split`. Trénovací data vůči testovacím jsou rozdělena v poměru 2:1 díky parametru `test_size`, který určuje, kolik prvků z datové sady připadne testovací podmnožině a zbytek trénovací. Např. mám-li sadu s 90 000 prvky podle poměru 2:1, trénovací podmnožině by připadlo 60 000 prvků a trénovací 30 000. Metoda se zároveň stará o náhodné zamíchání prvků v sadě před tím, než provede

samotné rozdělení. Pro připomenutí, pod proměnnou X jsou veškeré klíčové vlastnosti vozidel a pod Y jsou ceny inzerovaných vozidel.

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(
                        X, Y, test_size=0.33, random_state=0)
```

Hodnoty tenzorů by měly být obvykle upraveny na malé hodnoty například v intervalech [-1, 1] nebo [0, 1]. Protože jsou data heterogenní a hodnoty příznaků nabývají různých intervalů, musím provést normalizaci hodnot. Používám k téhle operaci třídu `StandardScaler` [50]. Samozřejmostí je potřeba normalizovat data jak trénovací, tak testovací.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

Nyní stačí nastavit síť pomocí sekvenčního modelu frameworku Keras. Oproti konfiguraci prototypu jsem změnil počet jednotek „neuronů“ v prvních dvou vrstvách (10), lépe je pojmenoval (parametr `name`) a velikost vstupu jednotek jsem určil dynamicky díky vlastnosti `shape`. Aktivační funkce vrstev zůstala stejná `relu`, avšak až na výstupní vrstvu (`layer3`), kde jsem ji smazal, a tak se z ní stala lineární vrstva. Tato výstupní vrstva je typická pro použití v regresních úlohách, kde se pokoušíme predikovat jednu spojitou hodnotu.

```
import keras

model = keras.Sequential()

model.add(keras.layers.Dense(10, activation='relu',
          name="layer1", input_shape=(X_train.shape[1], )))
model.add(keras.layers.Dense(10, activation='relu',
          name="layer2"))
model.add(keras.layers.Dense(1, name="layer3"))
```

```
model.compile(optimizer='rmsprop',
              loss='mean_squared_error',
              metrics=['mean_squared_error', 'mean_absolute_error',
               'mean_absolute_percentage_error','cosine_proximity'])
```

Kromě umíněných úprav vrstev, jsem při kompilaci modelu změnil optimalizátor z `adam` na `rmsptop`. Pro odhalení, zda je model nakonfigurován správně a konverguje k optimálnímu poměru mezi podučením a přeučením sítě, jsem využil regrese specifické metriky: `mean_squared_error`, `mean_absolute_error`, `mean_absolute_percentage_error` a `cosine_proximity`.

3.3.2 Validace

V prvotním návrhu modelu jsem použil ztrátovou funkci střední kvadratické chyby (anglicky **mean squared error**, zkratka **MSE**). Běžně se používá na tento typ regresních úloh, takže zde by problém být neměl. Jedním z důležitých důvodů, proč samotná validace původního navrženého modelu byla zkreslená a nepoužitelná, je použití nevhodné metriky přesnosti (anglicky **accuracy**, zkratka **ACC**). Pro vyhodnocení střední kvadratické chyby můžu využít metriku střední absolutní chyby (anglicky **mean absolute error**, zkratka **MAE**). Znamená absolutní hodnotu rozdílu mezi predikcí a cílem. Cílové (hodnoty proměnné Y) a predikované hodnoty jsou v jednotkách českých korun, např. odchylka s hodnotou 1000 znamená, že predikce se v průměru liší o 1000 korun.

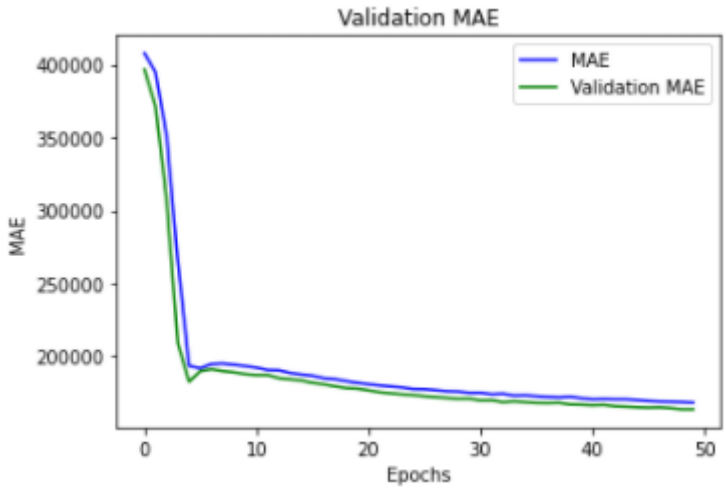
Nyní lze spustit samotné trénování sítě nad připravenými daty, které jsou rozděleny do trénovacích a validačních neboli testovacích. Počet epoch jsem navýšil na 50, protože nikdy nevíme, kolik epoch bude stačit síti k nalezení optimální hranice před prvotním natrénováním. Modelu je servírována datová sada, rozporcovaná do menších dávek o velikosti 16 vzorků.

```
history = model.fit(X_train, Y_train, validation_data=(X_test,
                    Y_test), epochs=50, batch_size=16)
```

Díky knihovně `matplotlib`, si mohu opět hodnoty metrik hlídající validaci modelu reprezentovat na grafu (viz Obr. 21).

```
import matplotlib.pyplot as plt
plt.plot(history.history['mean_absolute_error'], 'b',
         label='MAE')
plt.plot(history.history['val_mean_absolute_error'], 'g',
         label='Validation MAE')
```

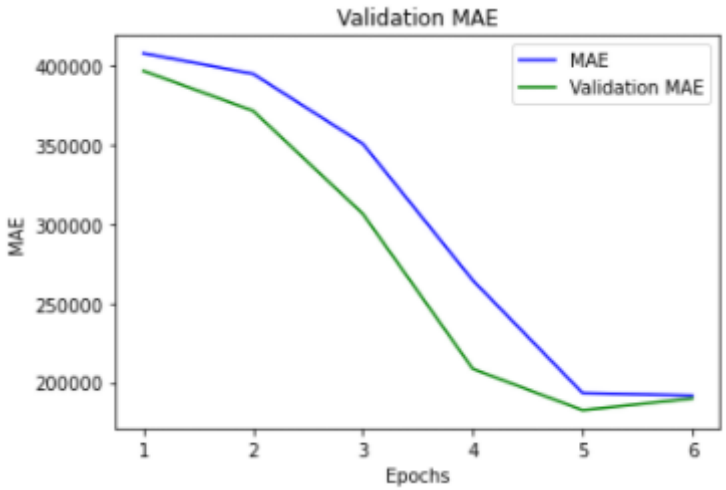
Křivka se zelenou barvou označuje hodnoty metriky MAE nad validačními daty a s modrou nad trénovacími daty. Na grafu je vidět, že kolem páté epochy nastává zlom. Do zlomu se model výrazně učí (podučení) a chybovost se snižuje. Po zlomu dochází k přeučení. Model pak začíná být sice více optimalizován, na druhou stranu ztrácí generalizaci, což vede k tomu, že model přestane být schopný správné predikce nad daty, které neměl k dispozici při trénování. Takový jev je nežádoucí.



Obr. 21: Graf metriky MAE nad trénovacími i validačními daty během 50 epoch
Zdroj: vlastní zpracování

Řešením je snížení počtu epoch trénování modelu. Pro ušetření času přetrénování, lze nad dostupnými metrikami zaměřit graf do šesté epochy a odhalím tak klíčovou zlomovou epochu.

Z grafu (Obr. 22) je patrné, že zlom přichází v páté epoše. Díky `verbose` režimu při trénování modelu se stačí podívat na detail dané epochy a je vidět, že odchylka nad validačními daty (`val_mean_absolute_error`) je přibližně 182 510,- CZK.



Obr. 22: Graf metriky MAE s výřezem na prvních 6 epoch
Zdroj: vlastní zpracování

Po opětovném přetrénování modelu během pěti epoch získávám nad validačními daty hodnotu MAE (`val_mean_absolute_error`): 162 951,- CZK. Chybovost se tedy ještě snížila.

3.3.3 Predikce

Predikci ceny automobilu nad upraveným modelem provádím obdobně, jako u prvotního prototypu. Zásadním rozdílem je, že predikovaný vůz musí projít stejnými transformacemi nad daty (změna kódování a „dummy variable trap“). Je tedy nutné automobil přidat do vstupní datové struktury `dataframe` (nejpohodlněji nakonec).

```
predicted_car = pd.DataFrame({"brand": ["Škoda"],
                              "model": ["Octavia"],
                              "year": [2016],
                              "mileage": [58233],
                              "fuel": ["nafta"],
                              "gearbox": [1]})
X = X.append(predicted_car)
```

Poté co data vozu projdou optimalizací, je nutné vůz vyjmout z datové sady před rozdělením a promícháním. Pokud se tak nestane, už se k zakódovaným datům automobilu těžko dostaneme, a navíc nepůjde spustit trénování modelu, protože bude jeden záznam u osy X (vlastnosti vozidla) přebývat oproti ose Y (cena vozidla).

Odebrání tedy provedu vyříznutím vzorku ze sady a uložím do stejné proměnné `predicted_car`.

```
predicted_car = X[-1]
X = X[:-1, :]
```

Posledním krokem je samotná predikce automobilu. Jak je z ukázky kódu výše patrné, zvolil jsem Škodu Octavia se stejnými parametry jako u prototypu. Před předáním dat o predikovaném vozidle modelu provádím dodatečné škálování velkého rozptylu hodnot, použit je již existující objekt `scaler`, který škáloval před trénováním trénovací i testovací sady, proto mám jistotu, že i vybrané vozidlo bude škálováno podle stejného měřítka.

```
prediction = model.predict(scaler.fit_transform([predicted_car]))
print(prediction)
```

```
[[228704.98]]
```

Obr. 23: Ukázka kódu a výsledek predikce ceny automobilu Škoda Octavia
Zdroj: vlastní zpracování

Výsledná predikovaná cena pro daný vůz Škoda Octavia je přibližně 228 704,-.

3.4 Uživatelské rozhraní

Cílem této práce není vytvořit grafické uživatelské rozhraní. Aplikace jsou spustitelné v terminálu, případně v některém vývojovém prostředí. V této kapitole bych rád shrnul, jak zdrojový kód zprovoznit, spustit a případně upravit.

Jak již bylo zmíněno v kapitole 2.9 , kromě potřebné verze Python verze 3.6, je nutné doinstalovat knihovny Keras, NumPy, Pandas, Scikit-learn a Matplotlib. Ze své zkušenosti mohu doporučit instalaci knihoven do virtuálního prostředí (systémový `venv`, `conda` atp.). Má to čistě preventivní důvod. Ne jednou se mi stalo, že verze knihoven byly mezi sebou nekompatibilní a pokud se knihovny instalují přímo do systémového prostředí, změna na kompatibilní verzi nemusí být vždy snadná. V takovém případě je snadnější virtuální prostředí se špatnými verzemi knihoven jednoduše smazat a vytvořit novou. Dává to také svobodu mít nastavených více

proměnných mezi různými projekty a jednoduše prostředí přepínat (např. projekty postavené na starších či novějších verzích Pythonu se sdruženými knihovnami). Osobně mohu doporučit prostředí Anaconda [51]. Má příjemné grafické rozhraní i schopný terminál. Nejvíce jsem však ocenil zaměření na technologie s datovou vědou, což přináší komfort v podobě kompatibility verzí knihoven (zejména Keras běžící nad TensorFlow).

Je mi blízké vývojové prostředí od společnosti JetBrains [52] a jako programovací jazyk upřednostňuji jejich produkt PyCharm [53], který nabízí vše potřebné na jednom místě, lze si v něm nakonfigurovat zmíněné `conda` prostředí, potřebné prerekvizity stáhnout z repositářů, naklonovat z GitLab projekt Carpricer (díky vnořenému submodulu Carscraper se nemusí klonovat více projektů) a spustit.

Pro lepší pochopení jednotlivých kroků je kód strukturován do bloků Jupyter Notebook. Lze tak snadno postupně bloky spouštět klávesovou kombinací (`Ctrl + Enter`). PyCharm umožňuje kromě selektivního spouštění bloků spustit všechny dohromady sekvenčně. Vedle spuštěných bloků s výsledky je možné využít vědecký mód, který funguje interaktivně s debug režimem. Vědecký režim se s každou změnou v datových strukturách obnovuje a lze vidět změny např. v tabulkách či grafech bez potřeby využití knihoven třetích stran.

Zdrojový kód s příponou *.ipynb indikuje právě použití s jupyter notebook. Není však problém tento kód např. uložit jako soubor *.py a spustit přímo skrz terminál a Python interpreta. Výsledek je pak stejný jako když se v PyCharm spustí kód s příkazem „Run All“ a na konci běhu se zobrazí predikovaná cena automobilu.

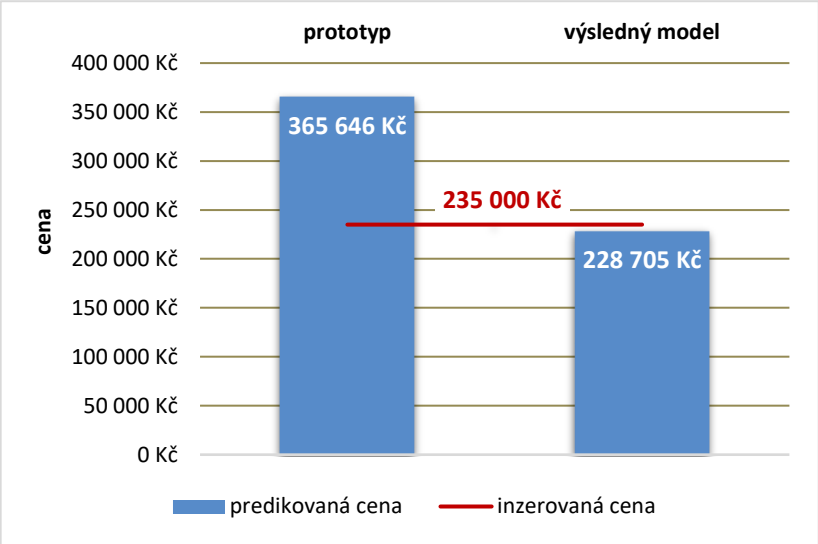
Pro predikci automobilu vlastního zadání je potřeba v kódu Carpricer upravit proměnnou `predicted_car`, jak je zmíněno v kapitole 3.3.3.

4 Shrnutí výsledků

Vytvořená sada programů dokázala, že je schopna získat datovou sadu automobilů prodávaných v České republice na portále www.sauto.cz a predikovat cenu vozidla pomocí modelu umělé neuronové sítě.

Prototyp modelu NS byl navržen jako sekvenční se třemi vrstvami, kde první dvě vrstvy měly každá 5 jednotek a poslední výstupní jedinou jednotku. Optimalizační funkce při kompilaci modelu byla zvolena „adam“. Dále u prototypu byla vybrána špatná metrika ke zhodnocení validace trénování nad daty. Nakonec i samotná predikce vozidla Škoda Octavia byla značně nepřesná. Model predikoval cenu **365 646,- CZK**, ale hodnota inzerovaného vozidla byla **235 000,- CZK**. Predikce se tedy minula o 130 646,- CZK, což je nárust ceny o **55,6 %.**

Na základě nepřesné predikce prototypu, jsem se rozhodl optimalizovat model i samotnou datovou sadu. Nástroj Carscraper jsem upravil o získání údaje druhu převodovky vozidel. Vlastnosti vozidel jsem vhodněji převedl na fiktivní proměnné a aplikoval na patřičné proměnné „dummy variable trap“, čímž jsem snížil multikolinearitu a redundanci dat. Po této fázi jsem získal v datové sadě pouze číselné údaje, které však měly často hodnoty širokého rozptylu. Proto jsem potřeboval hodnoty škálovat na daleko menší interval.



Obr. 24: Graf predikce ceny automobilu prototypem a výsledným modelem
Zdroj: vlastní zpracování

Výslednou sadu vzorků jsem rozdělil podle poměru 2:1 na trénovací a testovací (validační) data. Se sekvenčním modelem jsem různě experimentoval. Protože jsem neměl tušení, jak bude model konvergovat k optimálnímu zlomu mezi podučením a přeučením, začal jsem model trénovat na 500 epochách. Postupně jsem počet epoch snižoval, až jsem zjistil, že optimální mez je 5 epoch. Zároveň jsem zdvojnásobil počet jednotek v prvních dvou vrstvách modelu na 10, použil jsem optimalizační funkci „rmsprop“ a změnil validační metriky na nejběžněji používané pro regresní typ úloh. Model jsem poté nechal predikovat stejný automobil (opět Škoda Octavia se stejnými parametry) a získal jsem výslednou cenu **228 705,- CZK**, což byl pokles o necelé **3 %**.

5 Závěry a doporučení

Výsledkem práce je teoretické shrnutí problematiky hlubokého učení a praktická ukázka využití umělé neuronové sítě k predikci ceny automobilu na českém trhu. Pro implementaci řešení bylo následováno známých doporučení z publikované oborové literatury a technických dokumentací. Výsledný program je schopný predikovat cenu automobilu s odchylkou ceny 3 % oproti inzerované ceně. S použitím Python jazyka je možné skripty spustit na každé běžné platformě. Implementace predikce ceny automobilu popsaná v této práci za použití frameworku Keras je originální a není mi známá žádná alternativní či konkurenční práce.

Kromě programu k predikci, jsem vyvinul nástroj, který dokáže získat data z prodejního portálu www.sauto.cz pomocí metody web scrape. Nástroj je schopný získat přibližně 70 000 záznamů za 20 minut a běží plně automatizovaně.

Výsledek je použitelný pro osoby prodávající a kupující automobily, webové služby (např. jako rozšíření recenzí o nových automobilech), pojišťovny (vyhodnocení pojistek pokrývající automobily), bankovnictví (poskytování úvěrů na vozidla) a další podnikatelské subjekty. Kromě komerčního využití se nabízí také studijní, obzvlášť pro zájemce z oblasti predikování pomocí hlubokého učení.

Během implementace a experimentování s modelem jsem zjistil, že úspěšnost modelu neuronové sítě je tak vysoká, jak jsou kvalitní nasbíraná data a jejich následné optimalizování. S následnou úpravou dat a modelu umělé neuronové sítě, jsem získal přesnější predikci samotného automobilu.

Řešení ale odhalilo také problémy, které by stály za další bádání. Přesnost predikce cen méně rozšířených značek a modelů se značně zkreslovala. Spektrum inzerovaných automobilů bylo veliké. Srovnáme-li vzorky podle ceny zjistíme, že nejlevnější vozidla mohou být za směšnou cenu jednotek tisíc korun českých a nejdražší vozidla mají hodnotu desítek milionu korun českých. Kromě velkého cenového rozdílu hraje roli rozložení počtu inzerátů mezi značkami a modely. Např. vozidel značky Škoda je minimálně čtvrtina všech záznamů (přibližně 20 000 z 80 000 inzerátů), kdežto vozidel Aston Martin je jen několik jednotek (3). To zanáší do modelu další zkreslení, zejména pro značky s nedostatkem vzorků v datové sadě.

Chybějící vozidla by bylo nutné doplnit z jiných zdrojů, uměle vygenerovat, nebo odstranit.

Kromě zmíněného problému s daty, který spadá převážně do datového inženýrství, by bylo vhodné dále zkoumat vliv zvýšení počtu vrstev modelu s odlišnými aktivačními funkcemi a změnou počtu jednotek v každá vrstvě.

6 Seznam použité literatury

[1] *Sauto.cz - prodej aut, inzerce automobilů* [online]. [vid. 2020-07-26]. Dostupné z: https://www.sauto.cz/

[2] GOODFELLOW, I., Y. BENGIO a A. COURVILLE. *Deep Learning*. Cambridge: Mit Press, 2016. ISBN 978-0-262-03561-3.

[3] MOR-YOSEF, S., A. SAMUELOFF, B. MODAN, D. NAVOT a J. G. SCHENKER. Ranking the risk factors for cesarean: logistic regression analysis of a nationwide study. *Obstetrics and Gynecology*. 1990, **75**(6), 944–947. ISSN 0029-7844.

[4] ALZUBI, Jafar, Anand NAYYAR a Akshi KUMAR. Machine Learning from Theory to Algorithms: An Overview. In: A. GEORGE, A. S. PANDIAN, V. VINOTHINA a A. C. DONALD, ed. *Second National Conference on Computational Intelligence (ncci 2018)*. Bristol: Iop Publishing Ltd, 2018, s. UNSP 012012.

[5] MIND, Data Science-Data. *Hluboké učení pro každého? Neuronové sítě a spol. | Data Science - Data Mind* [online]. [vid. 2020-04-04]. Dostupné z: https://www.datamind.cz/cz/blog/hluboke-uceni-neuronove-site

[6] *Kris Howard - Knitted Disruption* [online]. 15. březen 2020 [vid. 2020-07-26]. Dostupné z: https://www.youtube.com/watch?v=0lqyYJl9vuM&feature=youtu.be

[7] LIVINGSTON, Steven a Mathias RISSE. The Future Impact of Artificial Intelligence on Humans and Human Rights. *Ethics & International Affairs* [online]. 2019, **33**(2), 141–158. ISSN 0892-6794. Dostupné z: doi:10.1017/S089267941900011X

[8] ROSYPAL, Stanislav. *Nový přehled biologie*. Praha: Scientia, 2003. 1. vydání. ISBN 978-80-86960-23-4.

[9] *Neuron a jeho stavba | Mentem.cz* [online]. [vid. 2020-05-31]. Dostupné z: https://www.mentem.cz/blog/neuron/

[10] TSAI, Tsung-Hsien, Chien-Hung WEI a Chi-Kang LEE. DESIGN OF DYNAMIC NEURAL NETWORKS TO FORECAST SHORT-TERM RAILWAY PASSENGER DEMAND [online]. 2005, **6**, 1651–1666. Dostupné z: doi:10.11175/easts.6.1651

[11] MAŘÍK, Vladimír, Olga ŠTĚPÁNKOVÁ a Jiří LAŽANSKÝ. *Umělá inteligence*. Praha: Academia, 2003. ISBN 80-200-1044-0.

[12] DESHPANDE, Mohit. Perceptrons: The First Neural Networks. *Python Machine Learning* [online]. 12. září 2017 [vid. 2020-06-03]. Dostupné z: https://pythonmachinelearning.pro/perceptrons-the-first-neural-networks/

[13] *Deep Learning Toolbox Documentation - MathWorks United Kingdom* [online]. [vid. 2020-06-04]. Dostupné z: https://uk.mathworks.com/help/deeplearning/index.html

[14] CHOLLET, François. *Deep learning with Python*. Shelter Island, NY: Manning, 2018. ISBN 978-1-61729-443-3.

[15] VONDRÁK, Ivo. *Umělá inteligence a neuronové sítě: Určeno pro posl. 4. roč. FEI*. Ostrava: VŠB-Technická univerzita, 1994. ISBN 80-7078-259-5.

[16] *Backpropagation | Brilliant Math & Science Wiki* [online]. [vid. 2020-07-04]. Dostupné z: https://brilliant.org/wiki/backpropagation/

[17] MALADKAR, Kishan. 6 Types of Artificial Neural Networks Currently Being Used in ML. *Analytics India Magazine* [online]. 15. leden 2018 [vid. 2020-07-04]. Dostupné z: https://analyticsindiamag.com/6-types-of-artificial-neural-networks-currently-being-used-in-todays-technology/

[18] TCH, Andrew. The mostly complete chart of Neural Networks, explained. *Medium* [online]. 4. srpen 2017 [vid. 2020-07-07]. Dostupné z: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

[19] ARIK, Sercan Ö, Mike CHRZANOWSKI, Adam COATES, Gregory Frederick DIAMOS, Andrew GIBIANSKY, Yongguo KANG, Xiongmin LI, John MILLER, Andrew NG, Jonathan RAIMAN, Shubho SENGUPTA a Mohammad SHOEYBI. Deep Voice: Real-time Neural Text-to-Speech. *undefined* [online]. 2017 [vid. 2020-07-04]. Dostupné z: /paper/Deep-Voice%3A-Real-time-Neural-Text-to-Speech-Arik-Chrzanowski/63880b57b95de8afd73036e55b9c4bccb7a528b9

[20] VAN BIESEN, W., G. SIEBEN, N. LAMEIRE a R. VANHOLDER. Application of Kohonen neural networks for the non-morphological distinction between glomerular and tubular renal disease. *Nephrology, Dialysis, Transplantation: Official Publication of the European Dialysis and Transplant Association - European Renal Association* [online]. 1998, **13**(1), 59–66. ISSN 0931-0509. Dostupné z: doi:10.1093/ndt/13.1.59

[21] INFO@AION.CZ, AION CS-. 563/1991 Sb. Zákon o účetnictví. *Zákony pro lidi* [online]. [vid. 2020-04-19]. Dostupné z: https://www.zakonyprolidi.cz/cs/1991-563

[22] HEŘMAN, Jan. *Oceňování majetku* [online]. B.m.: nakl. Oeconomica, VŠE, 2005 [vid. 2020-04-19]. ISBN 978-80-245-0967-9. Dostupné z: https://is.ambis.cz/publication/3/en/Ocenovani-majetku/Herman

[23] DUFOUR, Jean-Marie. *Introduction to time series analysis* [online]. 2003. Dostupné

z: https://mail.cirano.qc.ca/~dufourj/Web_Site/ResE/Dufour_1998_C_TS_IntroductionTS.pdf

[24] LISTIANI, Mariana, Ralf MÖLLER, Michael MORLOCK, Stefan LESSMANN a Gert HAMBURG. Support Vector Regression Analysis for Price Prediction in a Car Leasing Application. In: . 2009.

[25] GONGQI, Shen, Wang YANSONG a Zhu QIANG. New Model for Residual Value Prediction of the Used Car Based on BP Neural Network and Nonlinear Curve Fit. In: *2011 Third International Conference on Measuring Technology and Mechatronics Automation*: *2011 Third International Conference on Measuring Technology and Mechatronics Automation* [online]. 2011, s. 682–685. ISSN 2157-1481. Dostupné z: doi:10.1109/ICMTMA.2011.455

[26] PUDARUTH, Sameerchand. Predicting the Price of Used Cars using Machine Learning Techniques. *International Journal of Information & Computation Technology*. 2014, **4**, 753–764.

[27] NOOR, Kanwal a Sadaqat JAN. Vehicle Price Prediction System using Machine Learning Techniques. In: [online]. 2017. Dostupné z: doi:10.5120/ijca2017914373

[28] *TensorFlow* [online]. [vid. 2020-07-21]. Dostupné z: https://www.tensorflow.org/

[29] TEAM, Keras. *Keras documentation: About Keras* [online]. [vid. 2020-07-22]. Dostupné z: https://keras.io/about/

[30] *NumPy* [online]. [vid. 2020-07-22]. Dostupné z: https://numpy.org/

[31] *SciPy.org — SciPy.org* [online]. [vid. 2020-07-22]. Dostupné z: https://www.scipy.org/

[32] *scikit-learn: machine learning in Python — scikit-learn 0.23.1 documentation* [online]. [vid. 2020-07-22]. Dostupné z: https://scikit-learn.org/stable/index.html

[33] *Project Jupyter* [online]. [vid. 2020-08-17]. Dostupné z: https://www.jupyter.org

[34] The first single application for the entire DevOps lifecycle. *GitLab* [online]. [vid. 2020-08-17]. Dostupné z: https://about.gitlab.com/

[35] BATRINCA, Bogdan a Philip C. TRELEAVEN. Social media analytics: a survey of techniques, tools and platforms. *Ai & Society* [online]. 2015, **30**(1), 89–116. ISSN 0951-5666. Dostupné z: doi:10.1007/s00146-014-0549-4

[36] *Git - Submodules* [online]. [vid. 2020-01-14]. Dostupné z: https://git-scm.com/book/en/v2/Git-Tools-Submodules

[37] *16.6. logging — Logging facility for Python — Python 3.6.10 documentation* [online]. [vid. 2020-01-14]. Dostupné z: https://docs.python.org/3.6/library/logging.html

[38] *pandas.DataFrame — pandas 0.25.3 documentation* [online]. [vid. 2020-01-15]. Dostupné z: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

[39] *Sequential - Keras Documentation* [online]. [vid. 2020-01-14]. Dostupné z: https://keras.io/models/sequential/

[40] *Activations - Keras Documentation* [online]. [vid. 2020-01-15]. Dostupné z: https://keras.io/activations/

[41] *Optimizers - Keras Documentation* [online]. [vid. 2020-01-15]. Dostupné z: https://keras.io/optimizers/

[42] *Losses - Keras Documentation* [online]. [vid. 2020-01-15]. Dostupné z: https://keras.io/losses/

[43] *numpy.array — NumPy v1.17 Manual* [online]. [vid. 2020-01-15]. Dostupné z: https://docs.scipy.org/doc/numpy/reference/generated/numpy.array.html

[44] *sklearn.model_selection.train_test_split — scikit-learn 0.22.1 documentation* [online]. [vid. 2020-01-15]. Dostupné z: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[45] ANAND, Saurav. Dummy Variable Trap. *Medium* [online]. 2. únor 2020 [vid. 2020-08-09]. Dostupné z: https://medium.com/datadriveninvestor/dummy-variable-trap-c6d4a387f10a

[46] *sklearn.preprocessing.LabelEncoder — scikit-learn 0.23.2 documentation* [online]. [vid. 2020-08-09]. Dostupné z: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html?highlight=labelencoder#sklearn.preprocessing.LabelEncoder

[47] *sklearn.preprocessing.OneHotEncoder — scikit-learn 0.23.2 documentation* [online]. [vid. 2020-08-09]. Dostupné z: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

[48] *pandas.DataFrame.replace — pandas 1.1.0 documentation* [online]. [vid. 2020-08-10]. Dostupné z: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.replace.html

[49] *sklearn.compose.ColumnTransformer — scikit-learn 0.23.2 documentation* [online]. [vid. 2020-08-11]. Dostupné z: https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html

[50] *sklearn.preprocessing.StandardScaler — scikit-learn 0.23.2 documentation* [online]. [vid. 2020-08-15]. Dostupné z: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[51] Anaconda | The World's Most Popular Data Science Platform. *Anaconda* [online]. [vid. 2020-08-17]. Dostupné z: https://www.anaconda.com/

[52] JetBrains: Developer Tools for Professionals and Teams. *JetBrains* [online]. [vid. 2020-08-17]. Dostupné z: https://www.jetbrains.com/

[53] PyCharm: the Python IDE for Professional Developers by JetBrains. *JetBrains* [online]. [vid. 2020-08-17]. Dostupné z: https://www.jetbrains.com/pycharm/

7 Přílohy

1) CD s implementací a pomocnými soubory

Příloha č. 1

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor:	**Bc. Zdenko Brandejs**
Studium:	I1800634
Studijní program:	N1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název diplomové práce:	**Predikce ceny automobilu**
Název diplomové práce AJ:	Prediction of car prices

Cíl, metody, literatura, předpoklady:

Cílem práce je popsat principy tzv. hlubokého učení, vysvětlit možnosti implementace v Pythonu a realizovat ukázkovu aplikaci zaměřenou na predikování vývoje ceny automobilu od okamžiku jeho uvedení na trh.

Osnova:

1. Úvod

1.1 Motivace

1.2 Cíl

1.3 Metody

2. Teoretická část

2.1 Strojové učení

2.2 Hluboké učení

2.3 Předpovídání a predikce cen

2.4 Algoritmizace

2.5 Možnosti implementace v Pythonu

3. Praktická část

3.1. Vývoj ceny automobilu

3.1 Návrh prototypu algoritmu

3.2 Implementace

3.3 Testování

4. Výsledky

5. Závěr

- Goodfellow, I., Bengio, J., Courville, A. (2016) Deep Learning.
- Kelleher, J.D. et al. (2015) Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms. Worked Examples, and Case Studies.
- Buduma, N. (2017) Fundamentals of Deep Learning: Designing Next-Generation Machine Learning Algorithms.
- Muller, A.C., Guido, S. (2016) Introduction to Machine Learning with Python: A Guide for Data Scientists.
- Raschka, S., Mirjalili, V. (2017) Python Machine Learning.
- Vanderplas, J.T. (2017) Python data science handbook: essential tools for working with data.
- Summerfield, M. (2010) Python 3 : výukový kurz.

Garantující pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	doc. RNDr. Kamila Štekerová, Ph.D.
Oponent:	Ing. Karel Mls, Ph.D.
Datum zadání závěrečné práce:	21.10.2014