Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky
Katedra znalostních a webových technologií

Machine learning a moderní JavaScript

Autor diplomové práce: Bc. Pavel Nepomucký
Vedoucí diplomové práce: Ing. David Chudán, Ph.D.
Rok obhajoby: 2019

Čestné prohlášení

Prohlašuji, že diplomovou práci na téma

„Machine learning a moderní JavaScript“

jsem vypracoval samostatně a veškerou použitou literaturu a další prameny jsem řádně označil a uvedl v přiloženém seznamu.

V Praze dne 29. dubna 2019

.....

podpis

Název diplomové práce:

Machine learning a moderní JavaScript

Abstrakt:

Předmětem této diplomové práce je využití programovacího jazyka JavaScript v oblasti machine learning. Součástí práce je implementace webové aplikace. Webová aplikace obsahuje implementaci tří různých úloh pomocí javascriptové knihovny Tensorflow.js. V rámci práce je popsána oblast machine learning, včetně základní terminologie neuronových sítí. Praktická část obsahuje ukázky implementace vlastní neuronové sítě pro úlohu lineární regrese, klasifikaci čísel a klasifikací obrázků. Na úlohách s klasifikací obrázků je popsáno, jak lze využít již předtrénované modely neuronových sítí a pomocí rozhraní Keras je využít ve webové aplikaci. Implementované úlohy jsou vyhodnoceny včetně jejich použitelnosti do budoucna. Zároveň jsou popsány i problémy, se kterými se autor během implementace setkal.

Klíčová slova:

machine learning, JavaScript, neuronové sítě, Tensorflow.js, webová aplikace

Title of diploma thesis:

Machine learning and modern JavaScript

Abstract:

The subject of this thesis is usage JavaScript programming language in the field of machine learning. Part of the thesis is the implementation of the web application. The web application includes the implementation of three different tasks using the Tensorflow.js JavaScript library. Within this thesis is described the area of machine learning, including the basic terminology of neural networks. The practical part contains examples of implementation of own neural network for the linear regression, number classification and image classification. Image classification tasks describe how to use pre-trained neural network models and apply them in a web application with usage of Keras interface. Implemented tasks are evaluated including their future applicability. The problems encountered during the implementation are described by the author as well.

Keywords:

machine learning, JavaScript, neural networks, Tensorflow.js, web application

Poděkování: Rád bych poděkoval panu Ing. Davidu Chudánovi, Ph.D., CSc., za cenné rady a pomoc při vypracování této diplomové práce.

Seznam obrázků

Seznam tabulek

Úvod

Jsme obklopeni nezměrným množstvím dat, které neustále roste. Máme dispozici čím dál větší výpočetní výkon. Je přirozené, že chceme pomocí dostupných prostředků vytěžit z dostupných dat maximum informací.

Není to dlouho, kdy jsme viděli ve filmech autonomní vozidla, roboty v továrnách nahrazující kompletně lidskou práci nebo program vyhodnocující, zda má pacient rakovinu, a nevěřícně kroutili hlavami. Vše uvedené začíná být již běžnou realitou. Dnešní počítače umožňují vytvářet vysoce sofistikované modely neuronových sítí, které dokáží vyřešit i velice složité úlohy.

Samotní uživatelé v dnešní době očekávají alespoň základní autonomii webových aplikací, která jim usnadní využití aplikace. Nejtypičtějším příkladem jsou internetové našeptávače, které už nenapovídají pouze doplněním slova, ale snaží se také o rozšíření dotazu podle daného kontextu.

Machine learning v internetovém prohlížeči by byl před pár lety nemyslitelný pojem. Pro trénování komplexních modelů a jejich aplikaci se dříve používaly clustery výkonných počítačů a myšlenka, že by se stejný proces mohl provést i na běžném počítači v internetovém prohlížeči pomocí jazyka JavaScript byla čistě utopie. První pokusy využít jazyk JavaScript pro machine learning se objevují okolo roku 2012 s příchodem JavaScriptové knihovny Natural. Knihovna byla zaměřená pouze na zpracování přirozeného jazyka. Až o pár let později začínají vznikat první JavaScriptové knihovny soustředěné na neuronové sítě. JavaScript je velice oblíbený a rozšířený jazyk, proto popularita těchto knihoven časem výrazně sílila. Společnost Google si rostoucího trendu všimla a rozhodla se vytvořit knihovnu Tensorflow.js, která způsobila prakticky revoluci v oblasti machine learningu.

V diplomové práci je cílem seznámit čtenáře se základními termíny pro využití machine learningu v prohlížeči pomocí jazyka JavaScript. Práce také obsahuje popis oblíbených frameworků zaměřených na machine learning v jazyku JavaScript a jejich srovnání a konečně implementaci tří různých úloh pomocí webové aplikace a knihovny Tensorflow.js. Na první úloze je ukázáno, jak lze získat parametry lineární regrese pomocí knihovny, včetně srovnání výsledků modelu lineární regrese s neuronovou sítí. Druhý příklad obsahuje klasifikaci čísel. Pomocí implementované komponenty lze kreslit čísla, která jsou poté klasifikována modelem. Třetí příklad obsahuje klasifikaci obrázků. Po nahrání obrázku do aplikace je obrázek klasifikován do jednotlivých kategorií dle pravděpodobnostního rozdělení.

První kapitola vysvětluje pojem machine learning včetně základních termínů, se kterými se v tomto oboru setkáme, větší pozornost je věnována neuronovým sítím.

Druhá kapitola se zabývá pojmem big data, vysvětluje vztah big data a machine learningu a to, jakou úlohu mají big data jako zdroj dat pro machine learning.

Třetí kapitola se věnuje jazyku JavaScript. Jazyk za poslední léta prošel mnoha inovacemi včetně standardizace dle ECMAScript 2015, kterou již drtivá většina prohlížečů implementuje. V kapitole je vysvětleno, jak funguje mechanismus event loop, jsou zde představeny problematické prvky jazyka, s nimiž se často setkáme, a je popsáno, jak jim lze čelit. Velký prostor je také věnován asynchronním prvkům jazyka promises.

Čtvrtá kapitola obsahuje výběr a popis JavaScriptových knihoven zaměřených na machine learning. Kapitola se podrobně zabývá také knihovnou Tensorflow.js.

Poslední kapitola je věnována popisu implementace webové aplikace postavené pomocí frameworku Angular a knihovny Tensorflow.js, obsahuje popis tří implementačních úloh lineární regrese, klasifikace čísel a klasifikace obrázků. Konec kapitoly podává vyhodnocení aplikace včetně návrhu na zlepšení.

V závěru diplomové práce najdeme její vyhodnocení, možné návrhy na budoucí rozšíření a také posouzení, zda se všechny deklarované cíle diplomové práce podařilo splnit.

1 Machine learning

S pojmy machine learning a data science se setkáváme v posledním desetiletí čím dál tím častěji. Odpověď na otázku, proč tomu tak je, není složitá. Máme čím dál tím více dat, čím dál tím více výpočetního výkonu a také máme potřebu získávat nové informace, které nám usnadňují život. Díky značnému posunu výpočetního výkonu jsme schopni použít algoritmy na stále větším objemu dat. Dříve bylo možné využít omezený objem dat pro algoritmy, protože pro jejich reálné využití nestačil hardware. Omezení platí i dnes, ale objem dat, která jsme schopni zpracovat, každým rokem roste.

Pokud bychom chtěli vyjmenovat oblasti, kde se machine learning využívá, v první řadě bychom nejspíše vzpomněli marketing, například pro predikci chování zákazníka. Machine learning však nachází využití také v dalších oblastech, které mají dopad na naše životy, jako ve zdravotnictví, energetice, médiích, ve službách nebo na sociálních sítích. Např. ve zdravotnictví umělá inteligence využívající machine learning poráží při diagnostice zkušené doktory. Na internetových stránkách magazínu IEEE Spectrum[1] lze najít aktuální výsledky s úspěšností správné diagnózy pro konkrétní oblast diagnostiky. Dle dosavadních výsledků lze říci, že umělá inteligence si nevede vůbec špatně.

Současné stolní počítače typicky disponují vícejádrovými procesory s vysokou frekvencí, některé typy procesorů dosahují až 5 GHz, proto lze i běžný počítač efektivně využít pro zpracování většího objemu dat. Zpracovávání dat je přitom nejtypičtějším příkladem, jak lze vyžít vícejádrový procesor; na každém jádru lze zpracovávat určitou část dat a data na sobě jsou ve většině případů nezávislá.

Občas dochází k zaměňování pojmů machine learning a data mining. Data mining si lze představit jako hledání informací, například o lidech a jejich chování v různých zdrojích. Tyto informace se většinou ukládají v datových skladech. Předpokládá se, že se v těchto datech nacházejí cenné odpovědi na vybrané otázky. Nástroje data miningu hledají v datech právě tyto cenné odpovědi, které nejsou na první pohled zřejmé. [1]

Machine learning a data mining jsou dva různé obory, které jsou spolu v harmonii. Data mining je proces vyhledávání cenných informací a vzorů z dat. Často je se data mining považuje za proces získávání dat, ale to není správně. Data mining je proces,

[1] Přehled výsledků je k dispozici na internetových stránkách https://spectrum.ieee.org/static/ai-vs-doctors

který odhaluje na již získaných datech nové vzory. Organizace používají data mining pro identifikaci vazeb mezi daty. [2]

Proces machine learningu můžeme rozdělit do několika kroků. Prvním krokem je získávání dat. Na první pohled se nám může zdát, že je tento krok nezajímavý. Nesmíme ho však podcenit, správná příprava vhodných dat a jejich čištění jsou velice důležité, pokud chceme, aby náš model byl přesný.

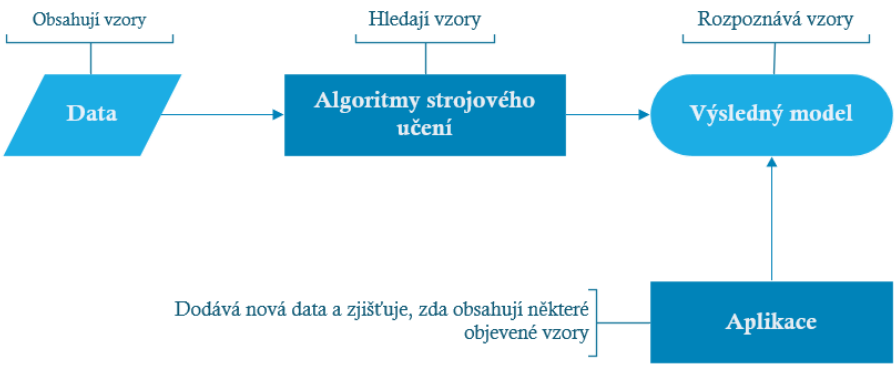
Druhým krokem je zvolení vhodného algoritmu. Algoritmů máme na výběr mnoho a je nutné mít při výběru toho správného již nějakou zkušenost a vědět, jaký algoritmus je pro daný problém vhodný. Nezkušenost s různými typy algoritmů lze substituovat vysokým výpočetním výkonem, který využijeme pro aplikaci všech dostupných algoritmů, nicméně tuto metodu nelze použít vždy.

Ačkoliv se celý proces nazývá machine learning, ty nejdůležitější kroky, které vedou k efektivnímu výsledku, vykonává stále člověk. Společnosti rády zaplatí jmění za člověka, který dokáže vytěžit relevantní data a vybrat vhodný algoritmus. Takový odborník se nazývá data scientist a jeho schopnosti jsou velmi ceněné, jelikož získat zkušenosti s daty, algoritmy a problematikou dané domény není vůbec jednoduché.

Cílem machine learningu je aplikace umělé inteligence, tak aby umožnila systémům schopnost automatického učení a zlepšování ze zkušeností, aniž by tak byl systém výslovně naprogramován. [3] To znamená vytvořit v počítači schopnost učit se s pomocí nashromážděných dat a již známých informací bez toho, aby počítače byly předem explicitně naprogramovány k řešení určitého úkolu. Machine learning lze definovat jako využívání dat k hledání odpovědí či vzorů.

> *„A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.“* – ***Tom Mitchell***

Obrázek 1 – Machine learning schéma



zdroj: autor

Machine learning se využívá v aplikacích rozeznávající obrázky, u chatovacích robotů nebo pro výběr doporučeného videa na serveru YouTube. Nejvýraznějším příkladem využití je vyhledávač Google. Ten pracuje mnohem sofistikovaněji, než tomu bylo před lety. K vyhledávácím frázím si prohlížeč snaží doplnit správný kontext a historii dřívějšího vyhledávání. Aniž bychom si to uvědomovali, tento důmyslný proces vyhledávání nám ulehčuje a ve většině případů i urychluje vyhledávání.

Každá větší společnost by měla přemýšlet, zda nemůže využít machine learning a svoje data pro zlepšení kvality svých produktů. Machine learning začíná být pro některé produkty již předpokládanou vlastností. Stejně jako již mnoho lidí vyžaduje responzivní zobrazení stránek ve svém telefonu, brzy přijde doba, kdy se bude od aplikace očekávat personalizace a schopnost hlubšího porozumění požadavku. Machine learning využíváme, abychom rychleji a jednodušeji řešili úkoly.

Existuje několik technik učení, které lze uplatnit v počítačovém řešení určitého problému. Každá technika je vhodná pro specifický typ úloh. V zásadě můžeme říci, že se během procesu machine learningu nejčastěji setkáme s technikami supervised learning, unsupervised learning a reinforcement learning. Uvedené techniky jsou popsány v kapitole 1.3.

1.1 Základní pojmy v oblasti machine learningu

Machine learning je velice obsáhlý obor užívající matematiky, statistiky a různých nově vznikajících pojmů. Čím dál tím více protíná i ostatní obory, začíná být jejich součástí a razí si cestu z univerzit do širšího odborného povědomí.

Autoři nástrojů, frameworků a knihoven se snaží psát dokumentace tak, aby jejich výtvory byly co nejvíce používané, ale i přesto se zřídkakdy obejdeme bez znalosti těch nejzákladnějších pojmů, které si nyní uvedeme.

1.1.1 Dataset

Data, ze kterýho se snažíme vytvořit model, se nazývá dataset. Dataset se skládá z data pointů, které lze interpretovat jako řádky v tabulce. Data point je reprezentován jako sada atributů. Každý z těchto atributů je určitého datového typu. Nejběžnější datové typy obsažené v datasetech jsou numerické hodnoty a textové řetězce (nominální data). Nominální data lze také kategorizovat vytvořením kategorií. Například atribut plnoletost bude obsahovat kategoriální data ano/ne. Kategoriální data můžeme nazývat také factors.

1.1.2 Feature

Atribut, který se pokoušíme v daném datasetu vysvětlit a vytvořit pro něj vhodný model, se nazývá feature nebo též vysvětlovaná proměnná. Někdy je třeba feature vytvořit ručně ve fázi předzpracování dat, například kategorizací atributu.

Pro správné určení feature atributu je nutná doménová znalost, díky které lze filtrovat relevantní data pro daný úkol. Feature ovlivňuje následný výběr algoritmů, které se použijí pro vytvoření vhodného modelu. Nejčastěji jsou hodnoty atributu numerické, ale mohou to být i textové řetězce. V tabulce č. 1 je znázorněn velice zjednodušený dataset se 6 atributy (sloupci), poslední sloupec obsahuje feature atribut neboli proměnnou, kterou chceme předpovídat.

Tabulka 1 - Ukázka datasetu

Jméno	Věk	Zůstatek	Místo použití	Čas použití	Podvod
V. Novák	37	183399	New York	6:15	**ANO**
P. Novotný	56	19520	Praha	15:17	**NE**
L. Nezkusil	61	690005	Brno	13:22	**ANO**
C. Nový	25	117530	Olomouc	5:59	**ANO**

1.1.3 Model

Často se setkáváme s potřebou popsat určitou část našeho světa. Model je jednoduchá reprezentace reality, která nám pomáhá pochopit, jak daná část funguje, nejčastěji je model definován pomocí matematiky a statistiky. Modely nám však také pomáhají rozeznat neznámé nebo předpovídat, co může nastat.

Realita je velice komplexní a model nikdy nebude schopen kompletně nasimulovat zkoumanou část reality, přesto bychom se však měli pokusit vytvořit takový model, který bude schopen zachovat co nejširší možné chování skutečnosti.

George Box, slavný statistik, jednou řekl:

"All models are wrong, but some are useful."

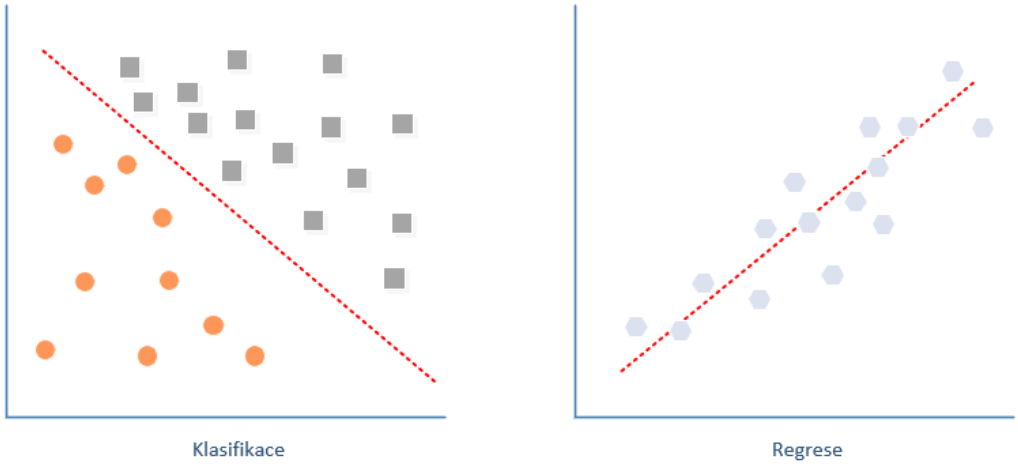
1.1.4 Algoritmus

Algoritmy můžeme považovat za návod, jak řešit určitý problém. Definují jednotlivé kroky, které nám pomáhají řešit specifický úkol. Algoritmů je velké množství, ty nejzákladnější jsou popsány níže.

Klasifikace se používá, pokud je nutné hodnotu zařadit do některé kategorie. Příkladem klasifikace může být rozpoznání zvířete. [4]

Pokud je nutné predikovat reálnou hodnotu z nějakého vzorku dat, můžeme se setkat s regresí. Příkladem regrese může být predikce budoucích srážek v daném regionu.

Obrázek 2 – Klasifikace vs. regrese



zdroj: autor

1.1.5 Gradient descent

Gradient descent můžeme chápat jako jeden z velice důležitých pojmů v oblasti machine learningu. Gradient descent je nejpoužívanější optimalizační algoritmus pro hledání optimálního řešení, existuje mnoho jeho variant, ale v základu se moc neliší.

Algoritmus se stará o optimalizaci modelu během trénování minimalizací hodnoty cost funkce, která říká, jak moc nepřesná byla predikce modelu. [5] Optimalizace probíhá iterativně pomocí změn hodnot u parametrů, tak aby abychom získali lepší výsledek při ověřování správnosti modelu.

"A gradient measures how much the output of a function changes if you change the inputs a little bit."—Lex Fridman (MIT)

Dle uvedené definice můžeme říci, že výsledná hodnota algoritmu říká, o kolik se změní hodnota funkce, pokud nepatrně změníme vstupní parametry modelu. Během tréninku to znamená, že pomocí gradientu se snažíme měnit hodnoty modelu tak, aby model měl co nejvyšší šanci provést správnou predikci výsledné hodnoty.

Úseku, kdy se snažíme minimalizovat hodnotu cost funkce, říkáme fáze učení modelu. Gradient descent je efektivní optimalizační algoritmus, který se pokouší

najít minimum funkce. [6] Velikost kroků, pomocí kterých se pohybujeme po gradientu a hledáme bod, ve kterém má cost funkce nejmenší hodnotu, říkáme rychlost učení (learning rate). [7] Pokud použijeme příliš velkou míru učení, tak bude konvergence rychlejší, ale může se stát, že nenajdeme absolutní minimum. Menší míra učení je pomalejší, ale spolehlivější, po gradientu děláme malé krůčky a zmenšíme riziko, že minimum přejdeme.

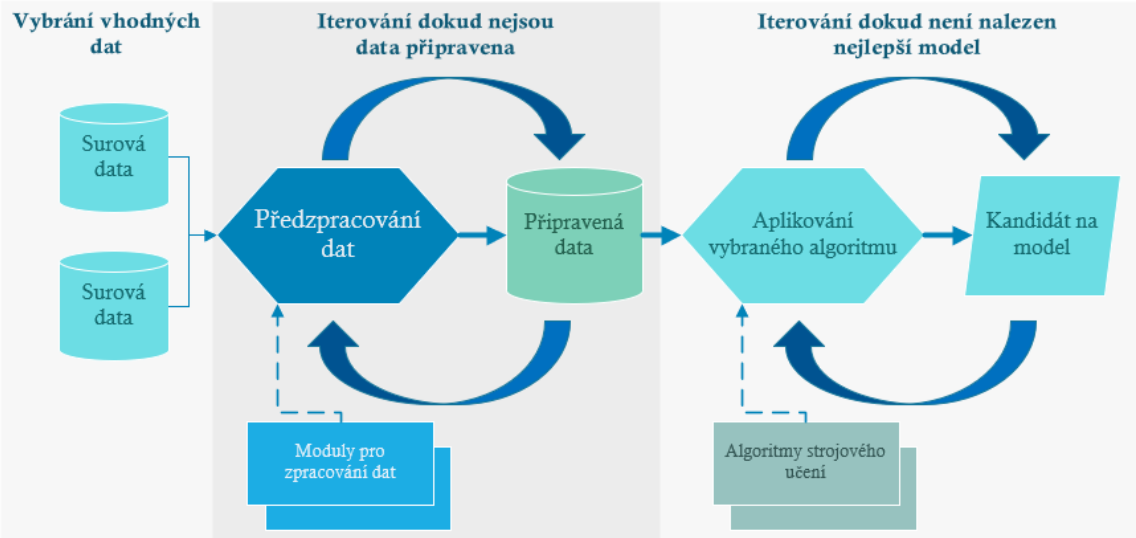
1.2 Machine learning proces

Celý proces machine learningu by měl začít položením správné otázky. Poté bychom měli zjistit, zda máme pro její zodpovězení k dispozici dostatek správných dat. Během získávání dat je často třeba spolupracovat s experty pro danou doménu.

Data nemusíme mít vždy zcela kompletní, a proto je nutné provést jejich předzpracování, kdy odfiltrujeme špatné, duplicitní nebo nepřesné hodnoty. Nejvíce času strávíme během machine learningu právě prací s daty. [8]

Během předzpracování dat můžeme vytvořit další atributy ze stávajících například pomocí kategorizace nebo rozdělení hodnot do intervalů. Jakmile máme data připravená, začneme aplikovat algoritmy a vyhodnocovat výsledky.

Obrázek 3 – Machine learning proces



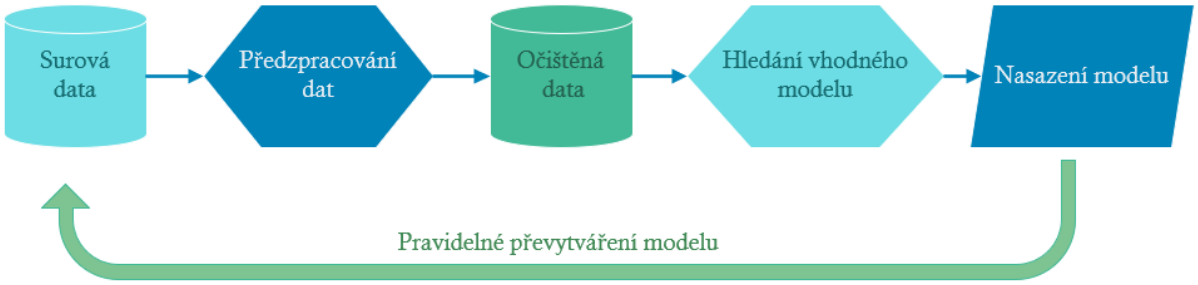
zdroj: autor

Po nalezení vhodného kandidáta proces machine learningu nekončí, pouze se pozastaví, dokud nenajdeme nový a lepší vzorek dat. Nový vzorek dat použijeme pro trénování nového modelu, tím vlastně celý proces na obrázcích č. 3 a 4 zopakujeme. Dat je k dispozici čím dál tím víc, proto je nutné modely stále přetrénovávat a

zkoušet, zda právě v těchto nových datech není nový vzor, který nám umožní vytvořit nový a lepší model.

Machine learning je stále se opakující iterace hledání vhodných dat a jejich zpracování, trénování modelu a jeho aplikace. Proces se neustále dokola opakuje a právě díky iteracím dochází k učení.

Obrázek 4 – Proces aktualizace modelu



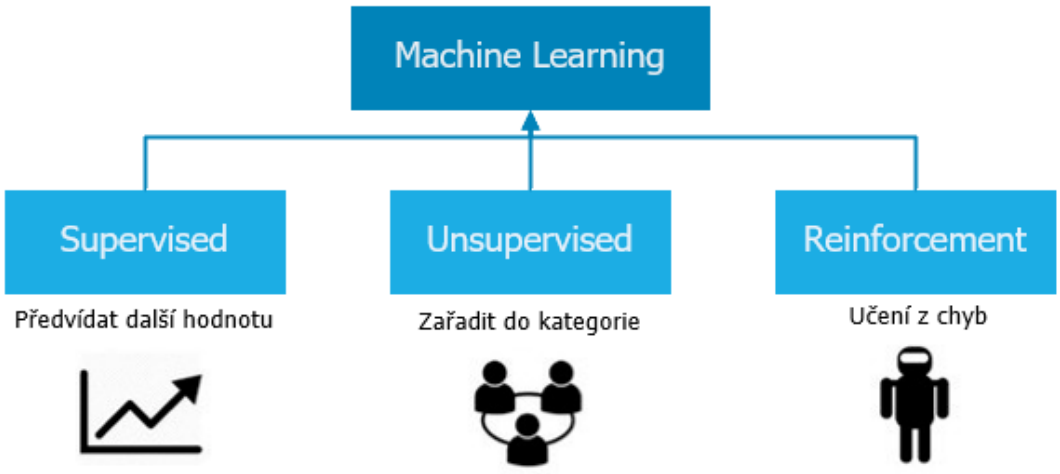
zdroj: autor

1.3 Typy učení

Každá metoda je navržena k řešení specifického problému. Je tedy velice důležité určit, s jakým typem problému se potýkáme.

Algoritmy machine learningu lze kategorizovat na supervised learning, unsupervised learning a reinforcement learning. [9] Je zásadní znát všechny typy machine learningu, tak abychom je uměli použít pro správný typ úlohy.

Obrázek 5 – Typy ML



zdroj: https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f

Během výběru lze také zohlednit čas nutný k trénování modelu, který ve většině případech závisí na požadované přesnosti a objemu trénovacích dat. Dále lze zohlednit počet parametrů modelu, které ovlivňují počet iterací a v případě velkého počtu parametrů komplikují také aplikaci modelu. [10]

V příloze diplomové práce je na obrázku č. 44 přiložená pomůcka pro vybrání vhodného algoritmu.

1.3.1 Supervised learning

Během metody supervised learning začínáme s trénovacími daty, která jsme si připravili a transformovali do podoby relevantní pro náš daný problém. Data musí být čistá a smysluplná. Cílová proměnná, kterou se budeme snažit predikovat při použití techniky učení s učitelem, musí být součástí našich dat. Například pokud bychom chtěli, aby model uměl predikovat, zda byla platební transakce pomocí karty podvodná, trénovací množina dat musí obsahovat údaj, zda byla, či nikoliv.

Prvním krokem při použití techniky supervised learning je vybrat z trénovací množiny dat atributy, tento krok se nazývá attribute selection. V případě, že v množině trénovacích dat je velké množství atributů, vybereme pouze ty, které podle nás mají vyšší váhu pro předpovídání vybrané cílové proměnné. Velikost trénovacích dat je jedním z faktorů, které nám mohou pomoci během výběru vhodného algoritmu.

Ve zmíněném příkladu s kreditní kartou bychom si mohli zvolit třeba hodnoty věk a národnost držitele karty, dále by to mohla být i země, kde byla karta použita. Někdy je lepší vybrat z trénovací množiny dat méně atributů než použít všechny dostupné.

Množství trénovacích dat bývá často pouhým zlomkem dostupných dat, a proto je třeba kontrolovat, zda vytvořený model funguje správně nejen na trénovacích datech, ale zároveň umí pracovat i s daty novými. Proto se během validace modelu používají metody rozdělení dat na sety a cross-validation.

Rozdělení dat na trénovací, validační a holdout množiny

Při aplikaci metody rozdělíme data na tři množiny. První množina obsahuje trénovací data, zpravidla okolo 65 % celkových dat. Dalších 15 % dat bude sloužit jako data validační. Zbylých 20 % dat se nazývá holdout. Trénovací množina slouží pro trénování modelu. Validační množina se používá pro odhalení přesnosti modelu na známých datech, které již obsahují výsledky pro cílenou proměnnou. Holdout množina se označuje jako testovací a poskytuje finální odhad výkonnosti modelu, poté co

byl model natrénován a validován. Holdout množiny by se nikdy neměly brát v úvahu při ladění algoritmů. [11]

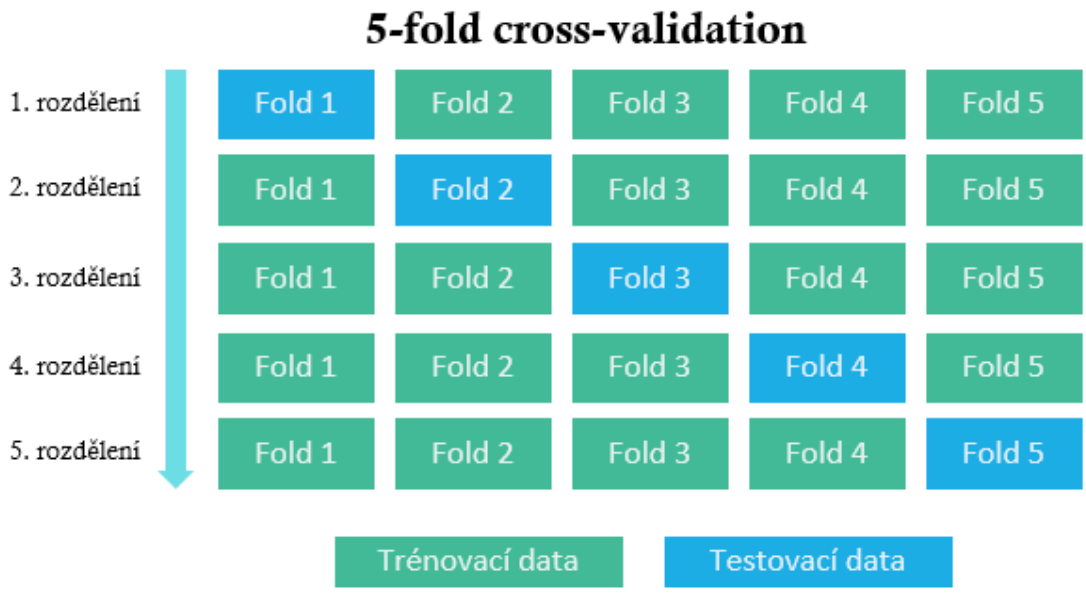
Rozdělení dat na jednotlivé sety je dobré použít, pokud pracujeme s velkým objemem dat nebo potřebujeme rychlou validaci modelu. Nevýhodou metody je náchylnost na výběr dat určených pro kontrolu modelu. Může se stát, že množina dat pro validaci bude obsahovat extrémní nebo okrajové hodnoty. [12]

Cross-validation

Metoda se často nazývá také *k-fold cross-validation*. Při aplikaci cross-validation se trénovací data rozdělí na *k* skupin. Jedna skupina je určená pro testování a ostatní skupiny jsou určené pro trénování. Poté se vybere další skupina, která se ještě nepoužila jako testovací, a proces se opakuje, dokud se neprostřídají všechny skupiny. Postup je znázorněn na obrázku č. 6.

Cross-validation je běžně preferovaná metoda, protože umožňuje modelu trénovat na více datových rozděleních. Díky tomu lze lépe usuzovat, jak dobře bude model vyhodnocovat na neznámých datech. [12]

Obrázek 6 – Ilustrace k-fold cross-validace



zdroj: https://www.datacamp.com/

Pokud je přesnost výsledného modelu příliš nízká, vrátíme se o krok zpět a pokusíme se vybrat jiné atributy, zvolit jiný algoritmus nebo změnit vstupní parametry modelu.

Tento proces stále opakujeme, dokud nezískáme model, se kterým jsme spokojeni. Může však nastat situace, kdy se k požadované přesnosti modelu nedostaneme, poté je nutné změnit vstupní data nebo algoritmus generující model. Výhodou techniky

učení s učitelem je informace o přesnosti modelu, protože si můžeme vždy ověřit výsledek na datech, která jsou vyhrazena pro kontrolu.

S metodou supervised learning se běžně setkáme u [9]:

- Rozpoznávání obličejů
- Aplikování správných reklamních sestav
- Rozpoznávání spamu

V dnešní době máme k dispozici mnoho algoritmů pro sestavování modelu a vybrat ten správný pro danou problematiku není vůbec jednoduché, nicméně se příprava vhodných dat nesmí nikdy podcenit. Žádný algoritmus není schopen vytvořit správný model nad špatnými daty. Výpočetní výkon počítačů je čím dál tím vyšší, a proto i trénovací data, která máme k dispozici, můžou obsahovat klidně 50 i více atributů, to ovšem neznamená, že díky tomu budeme mít přesnější modely.

Vybrat správné atributy a správný algoritmus není úplně jednoduché a nikde neexistuje příručka, která by nám s rozhodováním pomohla. Právě z těchto důvodů se můžeme setkat s rolí data scientist.

Data scientist je člověk, který má zkušenosti s algoritmy a také disponuje doménovými znalostmi pro oblast, ze které data pocházejí. Najít takového člověka je velice obtížné, protože jsou to odborníci vzácní a také velice dobře placení.

1.3.2 Unsupervised learning

Na rozdíl od techniky supervised learning se při technice unsupervised learning nesnažíme předvídat již známý atribut, ale snažíme se najít v datech vzory nebo struktury. U unsupervised learning začínáme většinou bez hlubších znalostí problematiky, které se zadaný úkol týká.

Metoda unsupervised learning je mnohem bližší učení takovému, jak ho známe my lidé. Při aplikaci této metody pracujeme pouze s daty, nesnažíme se předvídat již známý atribut, cílem metody je najít neznámé vztahy a vzory v datech, se kterými pracujeme. Oproti učení s učitelem nemůžeme určit přesnost modelu, jelikož nemáme s čím porovnat jeho výsledné hodnoty. Při aplikaci metody učení bez učitele se nejčastěji setkáme s algoritmy neuronových sítí a shlukování.

Co však metodu unsupervised learning činí v současnosti velice zajímavou, je skutečnost, že okolo nás je obrovské množství dat, které lze touto metodou zpracovat.

Unsupervised learning se většinou používá pro problémy, které můžou být vyřešeny pouze pohledem na dostupná data, v případě malé datové množiny. Pokud by ovšem datový set byl velice rozsáhlý, je třeba pro získání informací využít

rychlosti počítačů. V zásadě se s touto technikou setkáváme u problémů, které lze řešit intuitivně. Větší množství dat však člověk dokáže velmi obtížně manuálně zpracovat, proto je vhodné využít rychlost počítačů.

Příkladem, jak metoda v praxi funguje, může být aplikace, která na základě našeho chování nabízí relevantní zdroje. Pokud bychom psali odbornou publikaci, tak by nám aplikace mohla nabízet relevantní zdroje na téma, o kterém zrovna píšeme, na základě textu, který jsme již napsali, nebo také poznámek či dotazů, pomocí kterých jsme dříve hledali informace k danému tématu. S pomocí takové aplikace je produktivita výrazně vyšší. [9]

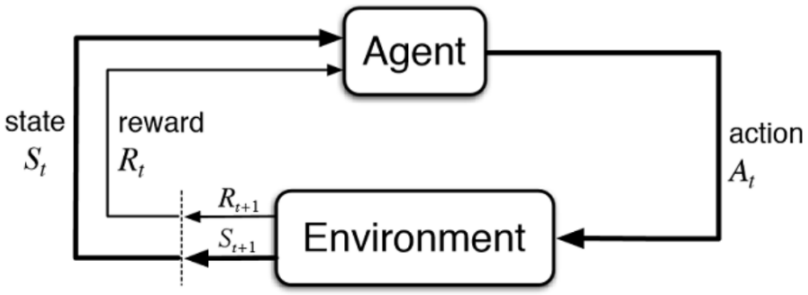
Metoda unsupervised learning je založená na datech a jejich vlastnostech, a proto se o ní říká, že je řízena pomocí dat (data-driven). [9] S metodou se nejčastěji setkáme u doporučovacích systémů (systémy, které uživateli nabízejí produkty dle jeho chování, např. oblíbených filmů).

1.3.3 Reinforcement learning

Během metody reinforcement learning se snažíme naučit jednat agenta v prostředí tak, aby maximalizoval užitek. Pro usnadnění rozhodování se mu poskytuje pouze omezené množství informací. Agent se pokouší hledat způsob, jak získat to, co hledá. Učí se reagovat na situace reakcemi, tak aby byl odměněn. Obrázek č. 7. ilustruje metodu.

Agentovi nejsou poskytnuty žádné informace o tom, jak se má zachovat, pouze je mu poskytnuta zpětná vazba o tom, zda něco svým jednáním získal. On sám musí zjistit, jak se má v dané chvíli zachovat, na základě zpětné vazby. Zpětná vazba se agentovi může předat po každém tahu nebo až na konci sezení. [9]

Obrázek 7 - Reinforcement learning



Zdroj: www.altexsoft.com[2]

.....

[2] Článek je dostupný na adrese https://www.altexsoft.com/blog/datascience/reinforcement-learning-explained-overview-comparisons-and-applications-in-business/

Cílem metody reinforcement learning je definovat nejlepší sekvenci rozhodnutí agenta, tak aby agent byl schopný řešit problém a maximalizovat při tom užitek.

Seznam správných akcí agent získá interakcí s prostředím a pozorováním odměn. [13]

Reinforcement learning se převážně využívá v těchto oblastech:

- Videohry (naučit počítač hrát hry)
- Roboti na výrobních linkách
- Správa zdrojů (vyhodnocování nákladů oproti užitku)

1.4 Možnosti nasazení natrénovaného modelu

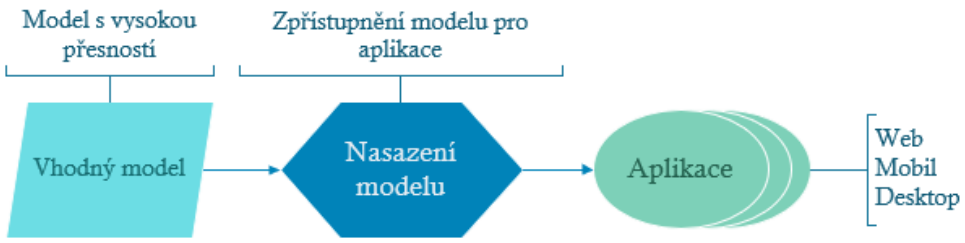
Způsobů, jak natrénovaný model reálně uplatnit, je mnoho, podobně jako technologií, které k tomu lze využít. V zásadě lze všechny způsoby rozdělit na dvě kategorie.

Do první kategorie řadíme všechny metody, kde se model nachází na straně serveru. Server přijímá požadavky klienta a provádí výpočty pomocí dostupného modelu, výsledky poté vrací klientovi.

Druhá kategorie zahrnuje způsoby, kdy model zapouzdříme přímo do klientské aplikace, výpočty se poté provádí na straně klienta, díky tomu odpadá nutnost být připojen na internetu a aplikace zůstává nezávislá na prostředí.

Každý typ aplikace modelu má svá pro i proti. Existuje však ještě kombinace obou dvou uvedených způsobů, a to využití serveru pouze pro získání nejvhodnějšího modelu, načež je model vrácen do klientské aplikace, kde je využíván. V případě, že dojde k nutnosti aktualizovat model nebo ke změně v trénovacích datech, klient pošle požadavek pro aktualizaci modelu serverem.

Obrázek 8 – Nasazování vhodného modelu



Zdroj: autor

Obrázek č. 8 popisuje proces nasazování modelu do reálné aplikace. V prvním kroku získáme model. Můžeme použít již předtrénovaný model, kterému však musíme transformovat data, tak aby odpovídala vstupnímu formátu modelu. Model si lze

natrénovat na vlastních datech, ale získat kvalitní a přesný model je velice časově náročné, a proto je někdy lepší použít již předtrénovaný.

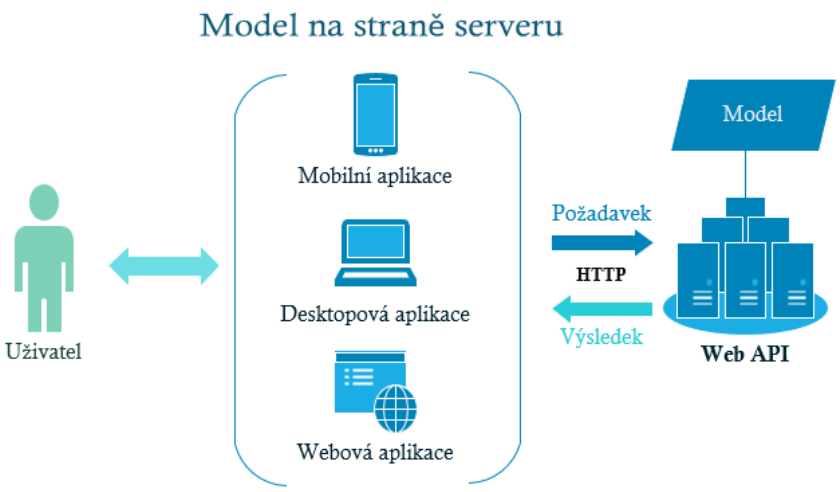
Nasazení modelu může být realizováno pomocí odkazu URL na vystavený model. Nebo si model můžeme přiložit přímo do aplikace. Pro načítání modelu z URL je nutné si rozmyslet, jak velký model bude použit, protože velké modely mohou nabýt velikosti až několika stovek megabajtů.

1.4.1 Model na straně serveru

Donedávna bylo nejčastějším způsobem, jak využít model v aplikaci, nasazení na straně serveru. To znamená, že všechny výpočty probíhají na straně serveru. Klient pouze zasílá serveru vstupy do modelu a server zasílá zpět výsledky. Tento scénář je ideální pro společnosti, které mají v modelu ukryté svoje know-how a nechtějí, aby bylo odhaleno konkurenční společnosti.

Model může být například zpřístupněn na internetu pomocí REST API a protokolu HTTP. Model lze přímo integrovat a zapouzdřit přímo v dané aplikaci. Pomocí vystaveného API lze model využít pro dotazování z různých desktopových, mobilních nebo webových aplikací. Podmínkou pro využití je nutnost být připojen k internetu pro přístup k endpointu, kde je API s modelem nasazeno.

Obrázek 9 – Model na straně serveru



zdroj: autor

Využívání modelu na straně serveru s sebou přináší i nevýhody, k nimž lze zařadit vysoké náklady na vybudování infrastruktury, nutnost připojení aplikace k serveru a delší odezva aplikace než v případě nasazení modelu na straně klienta.

Na druhou stranu nám server umožní využívat rozsáhlé a komplexní modely, pro které je třeba velká výpočetní síla. Uživatel je také ušetřen instalace technologií, které jsou nutné pro využívání modelu.

Model lze trénovat na mnohem rozsáhlejším datasetu a i model samotný může být několikanásobně složitější než na klientské straně. Dalším benefitem, který je pro společnosti velice důležitý, je fakt, že trénovací data a model jsou uloženy na serveru a tím jsou více chráněny před zneužitím jiné společnosti. Na obrázku č. 9 je znázorněno schéma při využití serverové architektury.

1.4.2 Model na straně klienta

Internetové prohlížeče se neustále zdokonalují a implementují nové standardy (Web API, HTTP2, WebAssembly), které umožňují využívat nové funkce a díky tomu vytvářet čím dál komplexnější aplikace. Pro mnoho populárních desktopových aplikací existuje alternativa ve formě webové aplikace.

Webové aplikace jsou multiplatformní, není nutné vyvíjet pro každý operační systém jinou verzi a navíc není nutné webové aplikace instalovat. Pro jejich použití stačí prohlížeč a připojení k internetu. Technologie implementované v prohlížečích, které nejvíce ovlivnily využití machine learningu v internetovém prohlížeči, jsou WebGL a WebGPU, protože umožnily využívání grafického akcelerátoru a tím otevřely bránu k využití grafické karty pro maticové operace.

Pomocí JavaScriptového rozhraní lze z prohlížeče využít grafický adaptér a tím zrychlit výpočet specifických úloh, pro které je grafický adaptér optimalizovaný. Mezi tyto úlohy můžeme zařadit grafické výpočty, v případě machine learningu práci s maticemi. Při využití grafického adaptéru výpočet až několikanásobně rychlejší.

Výkon mobilních zařízení jde neustále kupředu, a pokud porovnáme například první model chytrého telefonu Samsung Galaxy S, který byl uveden na trh v červnu 2010, s modelem z roku 2019, tak si můžeme všimnout, že první model telefonu disponoval jednojádrovým procesorem s označením Exynos 3110 o frekvenci 1,0 GHz a model z roku 2019 disponuje osmijádrovým procesorem s frekvencí 2,4 GHz s označením Exynos 9820.

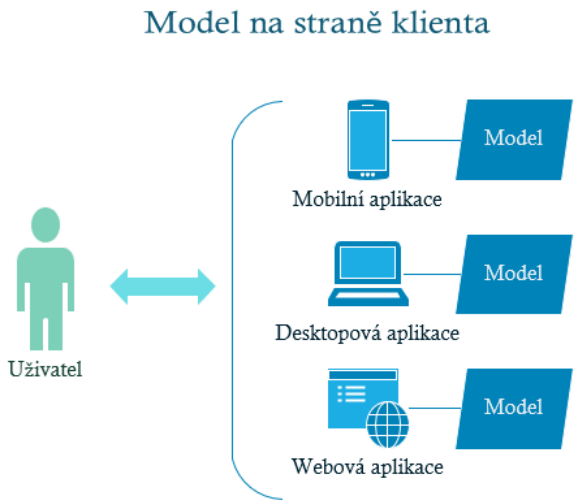
Během ani ne dekády došlo k více než 20násobnému zvýšení výkonu procesoru. Podobný výkonnostní trend lze sledovat i u ostatních mobilních zařízeních, jako jsou tablety a netbooky. Lze tedy říci, že výkon mobilních zařízení se neustále zlepšuje a limituje v náročnějších aplikací čím dál tím méně.

Každá aplikace je zodpovědná za stažení modelu a jeho použití. Pokud porovnáme, zda má využití klientské architektury dopad na přesnost modelu, je nutné si

uvědomit hardwarovou limitaci klienta. Při výběru vhodného modelu musíme zohlednit hardware klienta, a proto i modely, které lze využít, budou logicky méně komplexní, protože ve většině případů telefony, tablety a osobní počítače nejsou tak výkonné jako servery.

Méně komplexní modely nedosahují takové přesnosti, nicméně ztráta vykoupená menší složitostí bývá většinou v jednotkách procent.

Obrázek 10 – Model na straně klienta



zdroj: autor

Tabulka níže obsahuje porovnání využití serverové a klientské architektury s natrénovaným modelem.

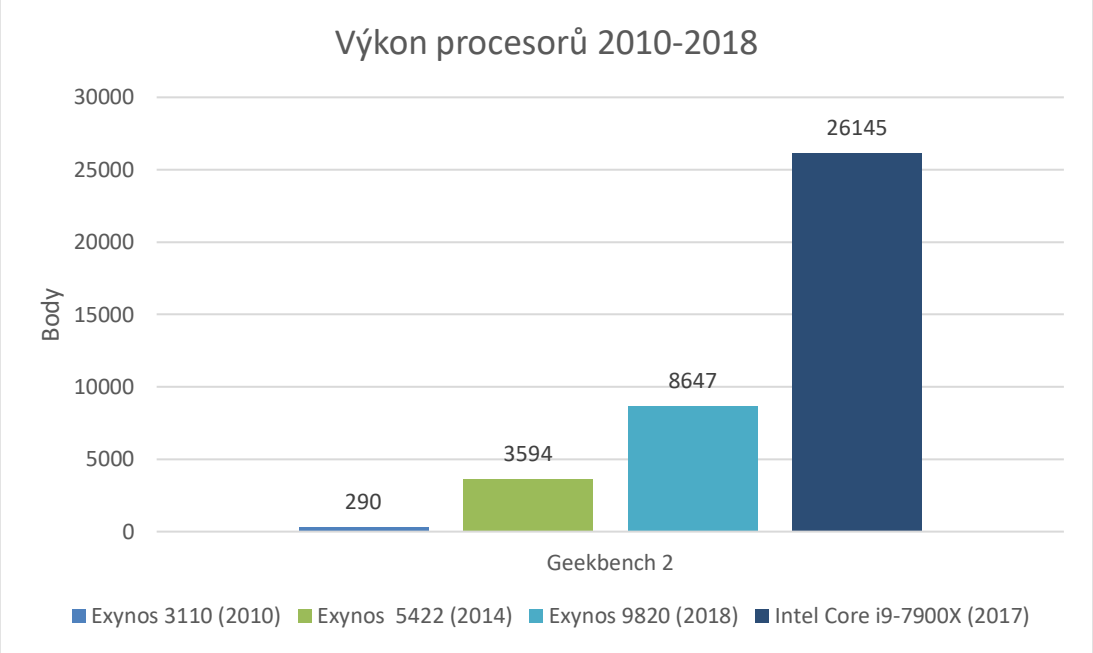
Tabulka 2 – Srovnání klientské a serverové architektury

Klient	**Server**
Pouze jednoduší a méně komplexní modely	Komplexní a rozsáhlé modely
Možnost využití na široké škále zařízení	Data, zdrojový kód a model jsou chráněny na straně serveru
Data, model a zdrojový kód jsou pouze u klienta	Vysoký výkon, možnost využití cloudu
Levnější a rychlejší vývoj	Aplikace musí být připojená k serveru
Složitější distribuce při aktualizaci modelu	Vysoká cena infrastruktury

zdroj: autor

Graf na obrázku č. 11 zobrazuje výkonnostní vývoj mobilních procesorů (procesorů určených pro tablety a mobilní telefony) v letech 2010–2018. Na grafu si můžeme všimnout, že výkon dnešních nejlepších mobilních procesorů dosahuje téměř 35 % výkonu nejlepších procesorů určených pro osobní počítače.

Obrázek 11 – Vývoj procesorů



zdroj: autor (data: https://www.notebookcheck.net/Mobile-Processors-Benchmark-List.2436.0.html)

1.5 Neuronové sítě

Machine learning a neuronové sítě jsou mocné nástroje, které již teď hrají roli v našich životech a usnadňují nám práci. Moderní grafické karty usnadňují trénování nerunové sítě a na internetu je velké množství dat, převážně díky sociálním sítím, které lze využít k učení. Tyto důvody vedou k využívání neuronových sítí i na osobních počítačích, a právě proto jsou stále běžnější a oblíbenější. [14] Neuronové sítě jsou specifickou metodou, kterou lze využít jak u supervised learningu, tak i při unsupervised a reinforcement learningu. V praktické části diplomové práce bude popsána implementace neuronové sítě, proto je to jediná metoda, která je v této kapitole popsána.

Během návrhu neuronových sítí se jejich autoři nechali, jak už z názvu vyplývá, inspirovat lidským mozkem. Neuronová síť se skládá z neuronů, které jsou propojeny mezi sebou podobně jako lidský mozek. Neuron v neuronové síti v sobě drží číslo v rozmezí od 0 do 1, které se nazývá aktivace. Neurony jsou organizovány

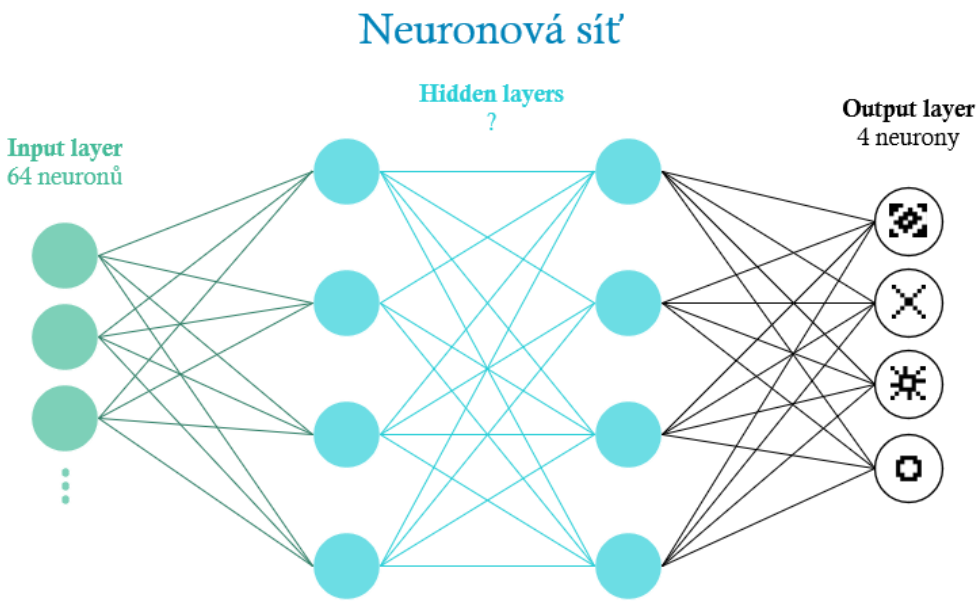
ve vrstvách a vrstvy, které se nacházejí mezi první a poslední vrstvou, se nazývají hidden.

Existuje mnoho typů neuronových sítí, některé se hodí pro specifický typ úloh. Feedforward neuronové sítě umí správně klasifikovat jednoduché věci, ale nemají možnost pamatovat si výsledky předchozích operací a mají nekonečnou variabilitu výsledků. Time step recurrent neuronové sítě si dokáží pamatovat předchozí operace a umožňují predikci budoucích hodnot. Recurrent neuronové sítě si také dokáží zapamatovat stavy předchozích operací a mají konečný set výsledků. Convolutional neural network se používá pro rozpoznávání obrázků, long short-term memory network je vhodný pro rozpoznávání řeči. Původním a nejtypičtější příkladem neuronové sítě je multiplayer perceptron.

Počet hidden vrstev bývá definován nejčastěji člověkem, a to včetně počtu neuronů, které obsahují. Neurony v hidden vrstvách slouží pro uložení vzorů během trénovací části.

Na obrázku č. 12 je zjednodušený příklad neuronové sítě pro rozpoznávání obrázků. Pro zjednodušení ilustrace se rozpoznávají obrázky o velikosti 8×8 pixelů. Počet neuronů, které potřebujeme ve vstupní vrstvě, je 64.

Obrázek 12 – Neuronová síť



zdroj: autor

Každý neuron ve vstupní vrstvě v sobě uchovává hodnotu, zda je vyplněn (0–1). Neurony ve skryté vrstvě v sobě uchovávají vzory rozpoznané na trénovacích obrázkách. Například v sobě může jeden neuron ve skryté vrstvě obsahovat informaci, zda má

obrázek vyplněn diagonály nebo zda obsahuje část nějakého tvaru, například část kruhu. Pokud je hodnota neuronu vyšší než aktivační mez, dojde k aktivaci tohoto neuronu a propojení s další vrstvou, na další vrstvě opět dojde k porovnání, zda hodnota přesahuje aktivační mez dostupných neuronů.

Celý proces pokračuje podle toho, kolik je k dispozici vrstev, dokud nedojdeme do poslední output vrstvy, ve které již máme výslednou hodnotu. Během přechodu z jedné vrstvy na další se hodnota aktivace vynásobí příslušnou vahou daného spojení. Může se stát, že výsledek bude nesprávný, a v případě, že jsem v procesu trénování neuronové sítě, se provede backpropagation.

Backpropagation je proces, při kterém se postupuje opačným směrem od výsledku ke vstupu, a během jednotlivých průchodů se upravují váhy tak, aby příští predikce byla přesnější.

Mezi hlavní přednosti neuronových sítí lze zařadit [15]:

- Extrakce vzorů z komplikovaných dat
- Detekce trendů, které jsou příliš komplexní na identifikaci člověkem
- Učení se pomocí příkladů
- Rychlost

1.5.1 Aktivační funkce

O tom, zda se má daný neuron aktivovat, nebo zůstat nečinný, rozhoduje aktivační funkce. Bez aktivační funkce by neuronové sítě byly reprezentovány pouze pomocí modelu lineární regrese, takový model je však velice limitovaný a nedokáže podávat dobré výsledky ve většině případů. Navíc by se neuronové sítě bez aktivační funkce nebyly schopny učit a modelovat komplikované druhy dat, jako například obrázky, video, zvukový záznam. [16]

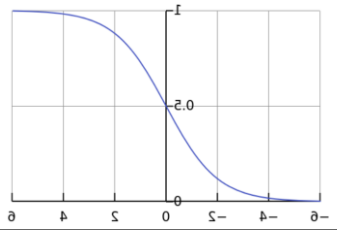
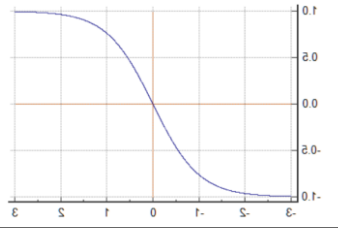
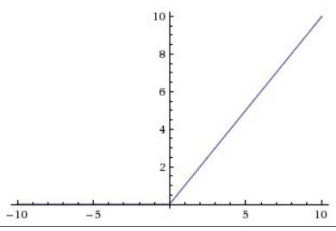
Neuronové sítě se považují za universal function approximators, to znamená, že dokáží napodobit a počítat jakoukoliv funkci. [17] Aktivačních funkcí existuje několik druhů, dříve se často používala funkce sigmoid, protože dobře reprezentovala chování neuronu. Tato funkce vezme jakoukoliv hodnotu a vtěsní ji mezi hodnoty od -1 do 1. Nedávno však byly zjištěny dva problémy, vanishing and exploding gradient, které zapříčinily ustoupení od používání této funkce. Oba zmíněné problémy souvisí s učením neuronové sítě během procesu backpropagation, kdy v případě velkých chyb dochází ke značné úpravě vah a velké úpravy můžou vést ke špatnému průběhu gradientu směrem k prvním vrstvám neuronové sítě. [18] Druhým problémem je, že funkce není centrovaná na nulu (zero-centered) a v případě aktivace jsou její hodnoty mezi 0 a 1, to znamená, že hodnoty aktivační funkce budou kladné a díky tomu budou hodnoty v optimalizačním algoritmu vždy buď záporné, nebo kladné a

budou daleko od sebe, což způsobí náročnější optimalizaci. Stále populárnější aktivační funkcí se stává funkce ReLU (Rectified Linear Unit), která se začala používat pro hidden vrstvy a odstraňuje nedostatky funkce sigmoid.

$$Y = \sum(weight * input) + bias$$

Bias je kontanta, kterou v neuronu lze nastavit, tak aby model podával lepší výsledky. Slouží jako prostředek, jak lze model ještě více přizpůsobit.

Tabulka 3 – Příklady aktivačních funkcí

Sigmoid	Hyperbolický tangens	ReLu
$\boldsymbol{A = \frac{1}{1 + e^{-x}}}$	$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$A(x) = \max(0, x)$
		

zdroj: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-network

1.5.2 Backpropagation

Backpropagation je důležitý mechanismus v neuronových sítích. Říká neuronové síti, zda udělala chybu během své predikce. Algoritmus byl poprvé představen v sedmdesátých letech 20. století, ale oceněn byl až později, v roce 1986 ve slavné publikaci Learning Representations by Back-Propagating Errors. [19]

Algoritmus se stará a propagování informace o chybě v neuronové síti. Netrénované neuronové sítě jsou během svých prvních kroků velice chybové. Z každé chyby se však neuronová síť snaží poučit pro další predikci.

Učení v neuronové síti znamená, že se v neuronové síti upravují parametry a bias u aktivační funkce, tak aby neuronová síť minimalizovala počet svých chyb. [20]

1.5.3 Feedforward sítě

Uzly neuronových sítí typu feedforward nevytvářejí mezi sebou cykly, patří proto k jednodušším neuronovým sítím. Informace proudí v sítích pouze jedním směrem

dopředu, nelze se vracet k předchozím uzlům. V těchto sítích nedochází k vytváření zacyklení. [21]

Nejjednodušším příkladem neuronové sítě typu feedforward je single-layer perceptron. Síť neobsahuje žádné skryté vrstvy a vstupní uzly jsou přímo propojené k výstupu, pomocí vah se definuje výstupní hodnota. Další ukázkou neuronové sítě typu feedforward je multi-layer perceptron. Ta může obsahovat jednu a více skrytých vrstev, díky kterým lze zachytit vzory umožňující získání přesnějšího výsledku.

Convolutional neuronové sítě (CNN) se používají pro klasifikaci obrázků a rozeznávání jednotlivých objektů. CNN jsou regulované verze neuronových sítí typu multi-layer perceptron. [22] Používají se pro rozpoznávání obličejů, osob na ulici, dopravních značek a pro rozpoznávání vizuálních dat. S CNN se setkáme u autonomních vozidel, dronů, lékařské diagnostiky a v oblasti bezpečnosti. [23]

1.5.4 Recurrent sítě

Recurrent Neural Network (RNN) jsou sítě, které dokáží vytvořit na vybraných uzlech cyklus. Výstup vybraného uzlu je opět jeho vstupem a proces se opakuje. RNN sítě jsou velice užitečné, pokud se zpracovávají sekvenční data. U sekvenčních dat se předpokládá, že současná sekvence dat závisí na minulé sekvenci. RNN sítě nachází uplatnění například na trhu s akciemi, kde se předpokládá, že současná hodnota akcií je závislá na jejich předešlé hodnotě. Dalším využitím je práce s textem, kde lze pomocí RNN sítě předpokládat, jaké další slovo nebo písmeno má být vygenerováno po dalším stisknutí klávesy.

Long Short Term Memory (LSTM) sítě jsou speciálním typem RNN sítí, jsou schopné učit se z dlouhodobých závislostí. [24] Poprvé byly představeny v práci Long Short-Term Memory (Hochreiter & Schmidhuber, 1997).

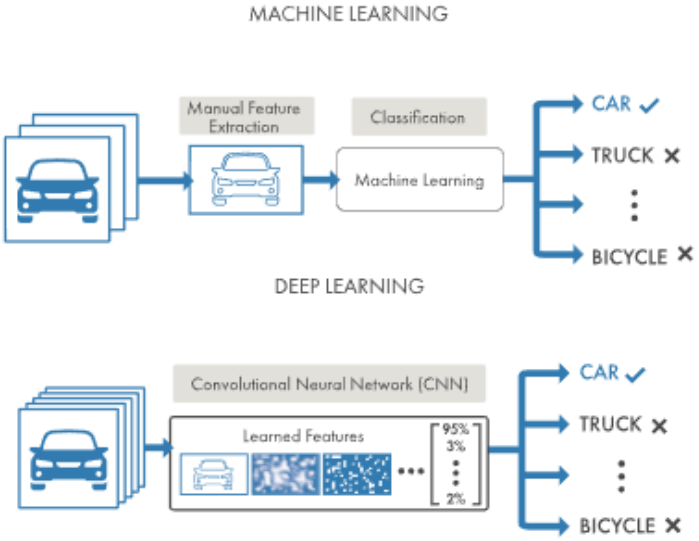
1.5.5 Deep Learning

Machine learning můžeme označit jako podmnožinu umělé inteligence. Stejně tak můžeme říct, že deep learning je podmnožina machine learningu. Deep learning využívá složité algoritmy, neuronové sítě a velké množství dat. Pro použití deep learning metody je zapotřebí vysoký výpočetní výkon, a to pro zpracování velkého objemu dat, které lze označit již uvedeným pojmem big data.

Při porovnání machine learningu a deep lerningu (Obr. 13) zjistíme, že atributy během machine learningu vybírá většinou data scientist, zatímco při použití deep learningu jsou vybrány pomocí neuronové sítě. Při aplikaci deep learningu se snažíme co nejvíce napodobit způsob, jak funguje lidský mozek. Deep learning můžeme považovat za propojení mnoha neuronových sítí do jednoho velkého celku.

Většina metod deep learningu využívá architektury neuronových sítí, proto jsou modely hlubokého učení často označovány jako deep neural networks. Termín deep obvykle označuje počet skrytých vrstev v neuronové síti. Tradiční neuronové sítě obsahují pouze 2–3 skryté vrstvy, zatímco deep neural networks mohou mít až 150 vrstev. Modely jsou trénovány pomocí obrovského objemu dat a neuronových sítí obsahujících mnoho vrstev. Hluboké učení vyžaduje velké množství označených dat. Například vývoj automobilů bez řidiče vyžaduje miliony obrázků a tisíce hodin videa. [25]

Obrázek 13 – Porovnání machine learning a deep learning



zdroj: https://www.mathworks.com/discovery/deep-learning.html

1.6 Artificial Intelligence

John McCarty se považuje za jednoho ze zakladatelů pojmu AI (Artificial Intelligence), pojem AI definoval jako:

> *"the science and engineering of making intelligent machines."* [26 str. 2].

Existuje mnoho způsobů, jak lze simulovat lidskou inteligenci, některé metody jsou více inteligentní než ostatní. Umělou inteligenci můžeme vnímat jako sadu podmínek nebo jako komplexní matematický model mapující data do kategorií. Uvedené způsoby jsou však explicitně naprogramovány člověkem, nedochází k učení a zdokonalování bez zásahu člověka. Schopnost programu modifikovat sám sebe, za účelem zlepšení, lze již považovat za jistou formu umělé inteligence.

Společnosti Google Brain, DeepMind a OpenAI nám v posledních letech ukazují svoje úspěchy v aplikaci umělé inteligence pro řešení čím dál tím komplikovanějších úloh.

Několik známých osobnosti se však nenechá ukolébat benefity umělé inteligence, ale naopak v nich vidí hrozbu. Mezi nejznámější osobnosti upozorňující na možnou hrozbu umělé inteligence můžeme zařadit vizionáře Elona Muska a známého vědce Stephena Hawkinga.

„The genie is out of the bottle. We need to move forward on artificial intelligence development but we also need to be mindful of its very real dangers. I fear that AI may replace humans altogether. If people design computer viruses, someone will design AI that replicates itself. This will be a new form of life that will outperform humans.“ – Stephen Hawking (Wired)

„I think we should be very careful about artificial intelligence. If I were to guess like what our biggest existential threat is, it's probably that…. Increasingly scientists think there should be some regulatory oversight maybe at the national and international level, just to make sure that we don't do something very foolish. With artificial intelligence we are summoning the demon. In all those stories where there's the guy with the pentagram and the holy water, it's like yeah he's sure he can control the demon. Didn't work out. “ – Elon Musk (MIT)

1.7 Artificial intelligence, machine learning a deep learning

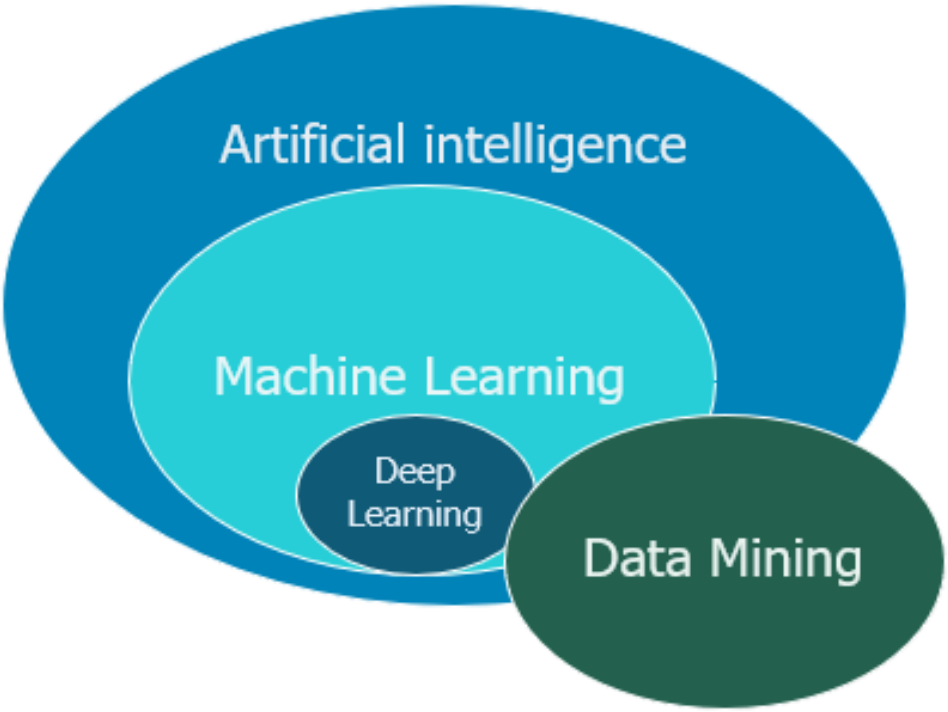
Pojmy machine learning a artificial intelligence oba spadají pod obor computer science a často bývají zaměňovány, přitom se jedná o dvě různé věci, které mají odlišné cíle a postupy. Na obrázku č. 14 je zobrazeno, jak spolu termíny souvisí.

Umělá inteligence je snaha vytvořit agenta, který v mnoha směrech uvažuje a pracuje jako člověk. To znamená, že robot je schopen se učit z chyb, napravovat je a získávat zkušenosti, které vedou k učení. Umělá inteligence je věda a přístup k vývoji technologie v oborech zpracování obrázků, zpracování signálů a jazyka, databází atd.

Algoritmy, které se mohou učit, se nazývají učící se algoritmy a sada těchto algoritmů se označuje jako machine learning. Machine learning je podmnožinou umělé inteligence. [15]

Deep learning je podmnožinou machine learningu. Pro řešení úloh pomocí metody deep learning se využívají hluboké neuronové sítě. Neuronové sítě využívané v metodě deep learning se musejí trénovat na velkém objemu dat a celá neuronová síť se učí rozpoznávat sama, nepotřebuje k tomu být explicitně naprogramovaná. Příkladem může být síť Google Brain, která se naučila rozpoznávat kočky poté, co zpracovala 10 milionů obrázků. [15]

Obrázek 14 – Rozdíl mezi AI, ML, DL a DM



zdroj: https://softwareengineering.stackexchange.com

2 Big Data

Pojem big data lze považovat v oblasti IT již za fenomén, zvlášť u expertů představujících se jako data scientists. Důvodem, proč jsou big data populární, je fakt, že efektivní využití těchto dat často končí velice slibnými výsledky. Přitom je ironií, že nikdo vlastně neví, co pojem big data znamená. Termín se začíná objevovat začátkem devadesátých let dvacátého století, jeho autorem je John Mashey. [27]

Definice pojmu big data se neustále vyvíjí s možností zpracovat stále větší objem dat, pro rok 2018 je definice pojmu big data následující:

„Big data je takové množství dat, pro které je zapotřebí nástrojů pro paralelní zpracování.“ [27]

Neustále se vyvíjející hardware umožňuje ukládat stále větší množství dat a tato skutečnost komplikuje definování slova ‘big’, přesto ale existují 4 charakteristiky, které lze použít pro definování a měření. Společnost IBM nazývá tyto ukazatele čtyři V (volume, velocity, variety a veracity). [28]

Ukazatel volume říká, s jakým objemem dat pracujeme, v současné době se jako měrná jednotka pro big data používá jednotka zettabyte, přičemž 1 zettabyte = 1×10^{21}, nebo se můžeme setkat i s jednotkou brontobyte, 1 brontobyte = 1×10^{27}. Množství dat, které produkujeme a jsme schopni uchovávat, se každým rokem zvětšuje, proto je velice pravděpodobné, že se v dalším desetiletí opět posune měrná jednotka o další řád, včetně změny definice. V roce 2013 eBay publikoval zprávu, ve které uvedl, že datové sklady tohoto obchodu uchovávají 90 PB (petabyte = 1×10^{15}) dat [29]. Pro analýzu takového objemu dat je nutné využít distribuovaný systém, ve kterém jsou data uložena a analyzována skrz síť databází propojených po celém světě. Čím více dat je schopný systém absorbovat, tím více informací z nich lze získat.

Big data velocity ukazuje, jakou rychlostí proudí nová data, ze všech možných zdrojů, do systému v reálném čase. Zdroje dat můžou například být business procesy, mobilní zařízení, různé senzory nebo sociální média. Tok těchto dat je nejen masivní, ale také kontinuální, což znamená, že data do systému stále proudí. To umožňuje uživatelům v reálném čase přístup k analýze aktuálních dat. Aktuální data umožňují společnostem a výzkumným pracovníkům dělat cenné a rychlé rozhodnutí a mít také strategickou výhodu před konkurencí.

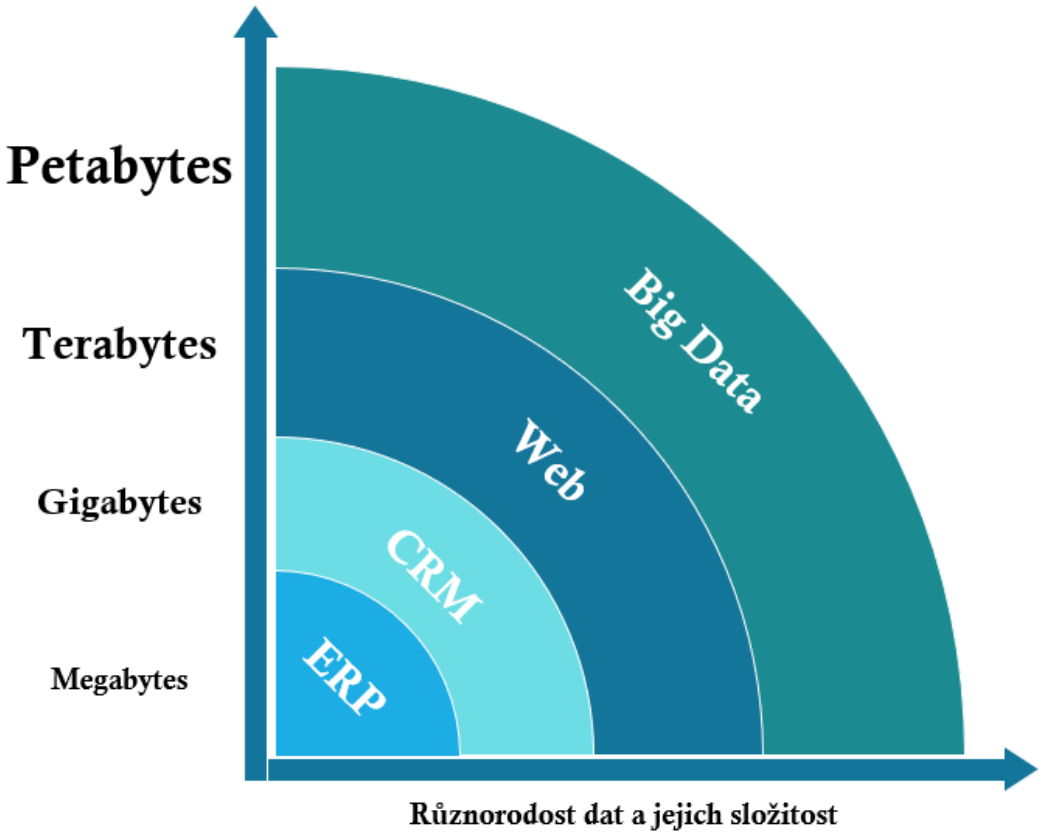
Dalším ukazatelem je variety. Tento ukazatel říká, jaká je rozmanitost získávaných dat. Data do systému proudí v různých podobách, ve strukturované, nestrukturované nebo v polostrukturované formě. Různorodost dat je způsobená různými zdroji. Strukturovaná data jsou nejčastěji uložená v databázích nebo v tabulkách, data jsou

organizovaná a obsažená v přesně definovaných polích a díky tomu jsou lehce dohledatelná. Pro dotazování strukturovaných dat byl společností IBM vyvinut v sedmdesátých letech jazyk SQL (Structured Query Language). Nestrukturovaná data nejsou organizovaná do předem definovaných polí, jsou těžko interpretovatelná modelem. Příkladem nestrukturovatelných dat jsou fotografie, internetové stránky, PDF soubory, videa, e-maily a příspěvky na sociálních médiích.

Posledním ukazatelem, který se používá v oblasti big data, je veracity neboli přesnost či pravdivost. Veracity popisuje proces eliminace jakýkoliv abnormalit, které mohou být v datech obsaženy. Jako abnormality předpokládáme nepřesné či nesprávné hodnoty, které byly systémem zachyceny. Velká nepřesnost dat komplikuje analýzu a považuje se za větší výzvu pro využití než práce s menším a přesnějším objemem dat. [30]

Nejtypičtějším zdrojem dat, který spadá pod big data, jsou v současné době sociální sítě. Ty generují každý den obrovské množství dat v různé formě (audio, video, textové zprávy). [31]

Obrázek 15 – Big Data



Zdroj: https://hortonworks.com/blog/big-data-defined-part-deux-value-definition/

3 Moderní JavaScript

Programovací jazyk JavaScript se v posledních letech těší velké oblibě. Je vhodné rozvést důvody, proč tomu tak je. JavaScript, lze využít pro mnoho různých aplikací, v jazyku lze programovat klientskou a zároveň serverovou část. Jazyk je standardizovaný a často aktualizovaný novými verzemi, je flexibilní a poměrně lehce naučitelný. Jazyk jako takový má i své slabší stránky, které mohou přinášet méně zkušeným vývojářům nečekané problémy během vývoje. Některé specifické prvky jazyka je značně obtížné na první pohled pochopit, přestože se jazyk řadí do skupiny jednodušších jazyků. Mezi složitější prvky jazyka řadíme scope, callbacks, klíčové slovo this, neexistence typu integer (jazyk jako takový patří do netypových, tzn. interpret jazyka se sám rozhoduje o typu hodnoty, se kterou pracuje); pojmy budou vysvětleny v následujících kapitolách.

Dříve se JavaScript používal spíše pro interakci uživatele s webovými prvky. S příchodem knihovny JQuery roku 2006 se pomalu začal měnit způsob jeho používání. Autor knihovny JQuery John Resig v ní implementoval zjednodušenou práci s dotazy AJAX a s manipulací DOM (Document Object Model). John Resig se také snažil odstranit různé chování prohlížečů, které v té době bylo noční můrou webových vývojářů. Knihovna JQuery ulehčila práci programátorům s tvorbou náročnějších webových aplikací. Weboví programátoři si knihovnu rychle oblíbili a popularita jazyka začala stoupat. S rostoucí oblibou programovacího jazyka JavaScript se na internetu objevily spousty dalších zajímavých knihoven, včetně nejrůznějších frameworků, které usnadňovaly vývojářům každodenní starosti a urychlovaly vývoj aplikací.

Další nepříjemností je skutečnost, že každý prohlížeč implementuje vlastní interpret jazyka JavaScript, proto se může stát, že v jednom prohlížeči bude kód fungovat, a v jiném ne. Příkladem může být funkce find, která vyhledá v poli prvek pomocí daného predikátu. Funkci find stále nenajdeme naimplementovanou v prohlížeči Internet Explorer, proto je vhodné během programování občas zkontrolovat, zda je použitá funkcionalita podporovaná v prostředí, pro které je aplikace cílená.

Existuje mnoho internetových stránek, kde lze ověřit, zda je použitá funkcionalita podporována v určitém prostředí. Celkově lze však říci, že případy chybějící funkcionality bývají v dnešní době vzácné. Občas se s nimi setkáme u prohlížeče Internet Explorer, který si však drží malé procento zastoupení na trhu, v ostatních prohlížečích chybějící funkcionalitu většinou nenajdeme.

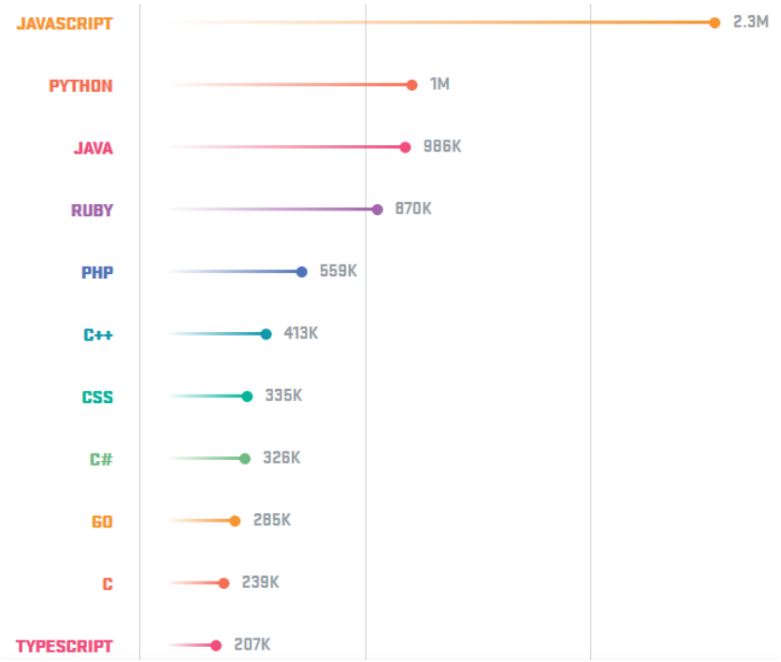
Přestože je JavaScript stále používanějším jazykem a jeho obliba stoupá[3], v oblasti machine learningu je nováčkem a soupeří s jazykem Python, který se v oboru machine learningu těší podpoře rozsáhlé komunity. Jazyk Python je také rychlejší a existují pro něj spousty knihoven speciálně pro machine learning. Právě díky tomu je jazyk Python stále nejpoužívanějším jazykem v oblasti machine learningu.

Server GitHub v roce 2018 zaznamenal rostoucí popularitu jazyka JavaScript v oblasti machine learningu a ve své zprávě také uvedl, že komunita Javascriptových vývojářů byla v oblasti machine learningu nejaktivnější s repositářem tensorflow. [32] Přesto se JavaScript umístil až na třetí příčce. Nejpopulárnějším jazykem je stále Python s repositářem Scikit-learn a poté C++.

Pomocí JavaScriptu můžeme naprogramovat webové stránky a aplikace, webové servery, prezentace, hry, umění, aplikace do chytrých hodinek, mobilní aplikace a létající roboty. Jeff Atwood, slavný americký vývojář a bloger jednou řekl, že:

„Cokoliv, co lze naprogramovat pomocí JavaScriptu, bude nakonec v JavaScriptu naprogramováno.“

Obrázek 16 – Počet commits dle jazyka na serveru GitHub za rok 2017



zdroj: https://octoverse.github.com/

[3] Server Stackoverflow vydal výsledky ankety za rok 2018, jazyk JavaScript se stal nejpopulárnějším jazykem za rok 2018, již šestým rokem v řadě (https://insights.stackoverflow.com/survey/2018#most-popular-technologies).

3.1 Scopes

Jedním ze základních kamenů každého programovacího jazyka je schopnost uchovávat hodnoty v proměnných. Bez možnosti ukládat, načítat a měnit proměnné bychom byli velice limitováni a programy by byly velice jednoduché a nezajímavé.

Otázkou zůstává, kde by se proměnné měly ukládat a jak k hodnotám přistupovat. Proto je nutné v každém programovacím jazyku definovat pravidla pro ukládání proměnných do paměti, včetně způsobu, jak k proměnným později přistoupit.

JavaScipt je založen jako function-based scope, to znamená, že každá definovaná funkce pro sebe vytváří vlastní scope. Zpravidla smýšlíme o funkcích tak, že nejprve definujeme funkci a poté do ní přidáme kód. Opačný postup je však srovnatelně užitečný. Obalit jakoukoliv již existující část kódu funkcí a tím skrýt tento kód uvnitř scope obalující funkce. Jinými slovy lze pomocí funkce schovat proměnné a další vnořené funkce. Technika schovávání proměnných a funkcí se na první pohled nemusí zdát vůbec užitečná, existují však důvody, které motivují tuto techniku používat. Například princip Principle of Least Privilege, někdy též nazývaný Least Authority nebo Least Exposure, říká, že bychom měli vystavovat v modulu či objektu jen to nezbytně nutné API a ostatní skrýt. [33] Moduly a knihovny se právě proto často celé obalují do funkce, nejčastěji můžeme u modulů vidět zápis pomocí immediately invoked function expression (IIFE, obrázek č. 18). Výhoda tohoto zápisu je v zapouzdření proměnných uvnitř funkce, nedochází proto k nechtěnému přepisováváni již deklarovaných funkcí a proměnných. [34] Na obrázku č. 17 je vidět, že se funkce nejprve zabalí do prvních žlutých závorek, které udělají z funkce výraz. Druhé žluté závorky slouží k okamžitému vykonání funkce.

Častým problémem programátorů je, že při implementaci funkcionality ztratí referenci na původní context[4] pomocí klíčového slova this, které se vztahuje k současné scope a jejímu contextu. Odpovědí na tento problém bývá často nepoužívat klíčové slovo this. Existují však metody, díky kterým se těmto problémům lze vyhnout. This se používá u literálů objektů, konstruktorů funkcí a tříd.

Jedním z mnoha způsobů, jak ztratit referenci na context při použití klíčového slova this, je zanořováním funkcí, viz Obr. 17. Dalším případem, kdy dojde ke ztrátě contextu, je využívání funkcí jako callbacků.

.....

[4] Context lze v JavaScriptu chápat jako referenci na objekt, kterému funkce patří.

Obrázek 17 – Ukázka zanořování funkcí

```
class Logger {
  constructor() {
    this.numbers = [1, 2, 3];
    this.message = "Cislo je:";
  }

  showNumberAfterDelay() {
    setTimeout(function doAnotherThing() {
      this.numbers.forEach(function log(number) {
        // this ztraci context => this.message je undefined
        console.log(this.message, number)
      });
    }, 500);
  }
}
```

zdroj: autor

K dispozici máme několik způsobů, jak tento problém řešit. Setkáme se s použitím funkce bind(), která vytvoří novou verzi s již přiřazeným contextem. Funkci bind je nutné použít pro každou zanořenou funkci. [35]

Další možností je uchovávat referenci na context v extra proměnné[5]. Stále populárnějším způsobem, jak řešit ztrátu reference na context, je využití arrow funkce. Arrow funkce je funkce, které není definovaný název.

V JavaScriptu se setkáme s funkcemi, kterým není definován identifikátor, tzn. nejsou pojmenovány; takové funkce se nazývají anonymní. Anonymní funkce se těžko debugují, protože neobsahují jméno a díky tomu jsou těžko dohledatelné ve stack trace. Stack trace je seznam všech volaných funkcí v danou chvíli. Anonymní funkce nevytváří vlastní context, context je zděděn od rodiče. Nevýhodou arrow funkce je ztráta názvu funkce, kód je tím méně čitelný.

Obrázek 18 – IIFE zápis

```
(function () {
    // Kód knihovny nebo modulu
})();
```

zdroj: autor

.....

[5] Typická forma zápisu je **let that = this;** nebo **let self = this;**

3.2 Closures

Pokročilým konstruktem jazyka je koncept closure. S closure se nejčastěji setkáme u funkcionálního programování, které je jedním z mnoha programátorských paradigmat. Koncept closure často vidíme u JavaScriptových knihoven a je stále více používán u zkušených programátorů.

Ve zkratce je closure scope vytvořený při deklaraci funkce a umožní funkci manipulovat a přistupovat k proměnným, které jsou definovány mimo block funkce. Closures se často používají pro implementaci timers, callbacks a privátních proměnných. [36]

3.3 Event Loop

Programovací jazyk JavaScript patří do kategorie programovacích jazyků označovaných jako single-threaded. To znamená, že program napsaný pomocí jazyka JavaScript nemůže vykonávat více instrukcí najednou. Na první pohled se tato skutečnost může zdát limitující v případě psaní asynchronního programu. Přesto však lze v JavaScriptu napsat asynchronní program s využitím jednoho vlákna, a to pomocí callbacks nebo promises. [37]

Jakýkoliv JavaScriptový kód, který se právě vykonává, blokuje celý event loop, brzdí vykreslování a zpomaluje interakci se stránkou. Pokud je event loop blokovaný, tak přestává reagovat celá stránka, včetně UI, a uživatel nemůže na nic kliknout, scrollovat atd. Proto se vývojáři webových aplikací snaží optimalizovat své weby, tak aby web působil responzivně. Responzivní web by měl dosahovat dnes již standardních 60 FPS[6] (60 Hz). Pro dosažení 60 FPS je nutné vykreslit okno prohlížeče každých 16,7 ms, takovou dobu má v ideálním případě event loop na zpracování JavaScriptového kódu. Pokud zpracování kódu trvá delší dobu, je vykreslení dalšího okna prohlížeče o tuto dobu zbrzděno. [38]

Ideální scénář se každých 16,7 ms opakuje a je následující [39]:

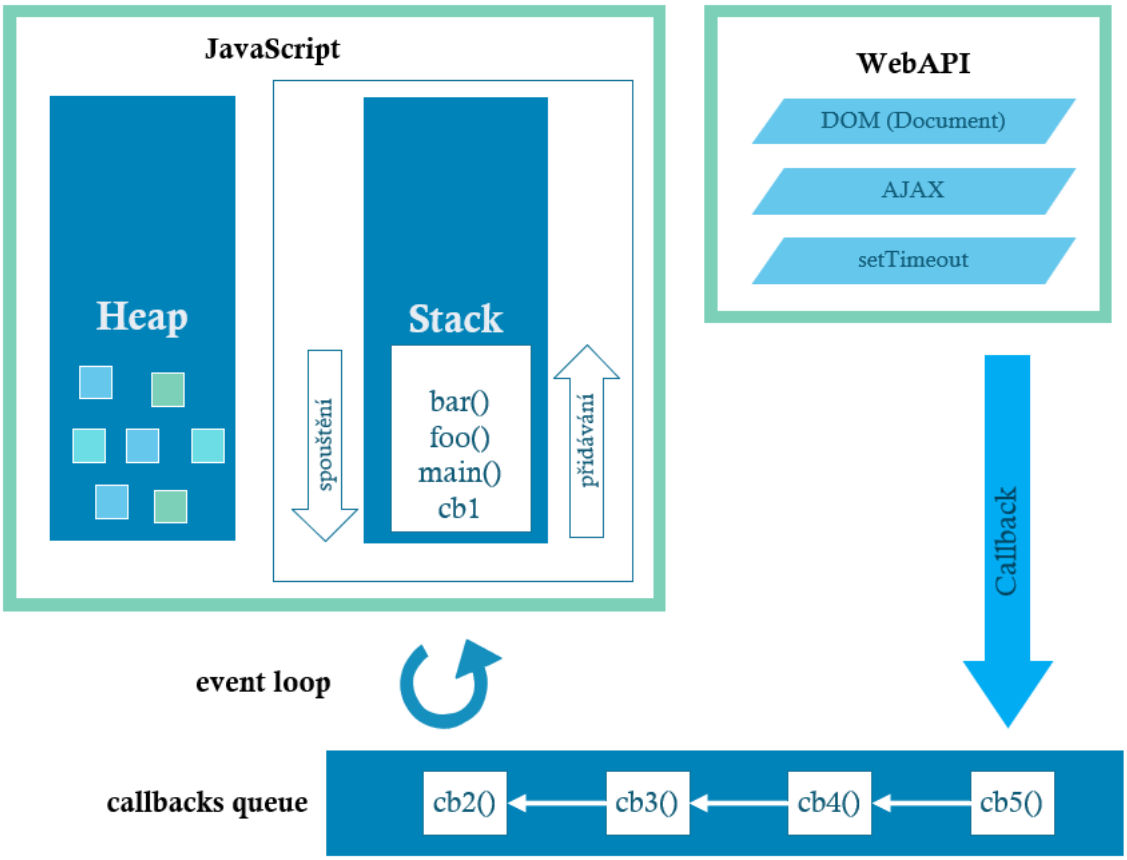
- Zkontroluje se, zda Stack obsahuje nějaký JavaScriptový kód, který je nutné obsloužit, pokud ano, kód se spustí. V případě, že se jedná o volání funkcí z WebAPI (DOM manipulace, AJAX, setTimout), dojde k přesunutí těchto funkcí do fronty k pozdějšímu zpracování.
- Jestliže se již nenachází žádný zdrojový kód ve Stacku, dojde ke zpracování všech funkcí z WebAPI. Tyto funkce většinou v sobě obsahují v parametru callback,

[6] FPS – Frames per second (počet vykreslených obrázků z vteřinu).

který se předá do fronty s callbacky k pozdějšímu zpracování. Po zpracování funkce dojde k předání callbacku do fronty, proces se opakuje pro každou funkci.

- Callbacky ze zásobníku se vkládají opět do Stacku, kde dochází k jejich spouštění.

Obrázek 19 – Event loop v JavaScriptu



zdroj: https://medium.com/front-end-weekly/javascript-event-loop-explained-4cd26af121d4

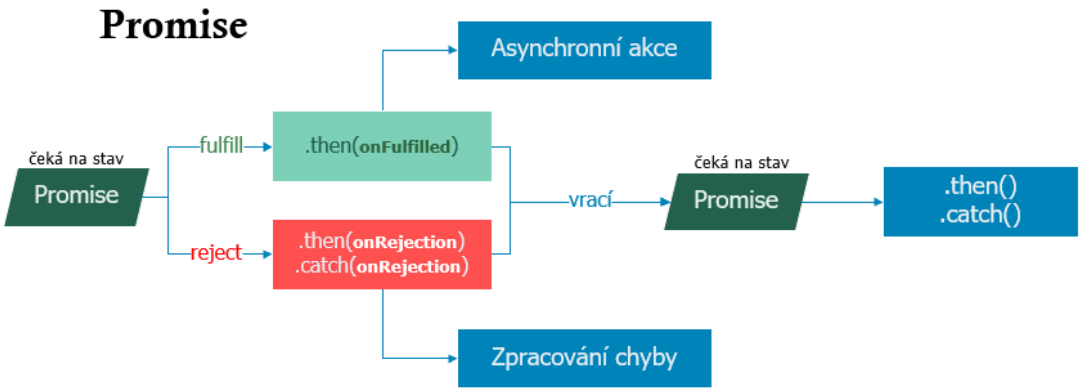
3.4 Promises

Program napsaný pomocí jazyka JavaScript nemůže vykonávat více instrukcí najednou. Promises si lze představit jako budoucí hodnotu nějaké operace. Promises lze řetězit dohromady do jednotlivých asynchronních kroků. Promise může nabývat jednu ze dvou možných hodnot, fulfilled, nebo rejected. Po získání finální hodnoty je již objekt promise uzavřen, jakékoliv pokusy modifikovat objekt budou ignorovány. [40]

Pro konstrukci objektu promise je nutné definovat dvě funkce. První parametr je funkce, která se typicky nazývá resolve, a druhý je funkce reject. Pokud uvnitř

objektu promise zavoláme funkci resolve, tak se stav objektu promise změní na fulfilled. V případě, že zavoláme funkci reject, stav se změní na rejected. Obě funkce resolve a reject lze zavolat s parametrem, který nese v případě volání resolve očekávanou návratovou hodnotu nebo v případě volání funkce reject důvod, proč se nepodařilo vrátit očekávanou hodnotu. Promise obsahuje funkci then, která také obsahuje dva parametry. První je funkce, která se zavolá v případě, že stav promise se změnil na fulfilled. Druhá funkce se zavolá, pokud se stav změnil na rejected. Do funkcí se předává návratová hodnota z funkcí resolve a reject.

Obrázek 20 – Promise objekt



zdroj: https://developer.mozilla.org

3.5 ECMAScript 2015 (ES6)

Vývojáři internetových prohlížečů se zavázali implementovat interpret jazyka JavaScript dle standardu vydaného společností Ecma International.

Ecma International je společnost vytvářející standardy pro technologie. V souvislosti s JavaScriptem nás zajímá standard pod označením ECMA-262, který obsahuje obecné specifikace pro skriptovací jazyk.

ECMAScript je definice skriptovacího jazyka obsažená ve standardu ECMA-262. Definice obsahuje pravidla, která by skriptovací jazyk měl splňovat, aby mohl být označen jako standardizovaný. Bohužel prohlížeče implementují různé verze standardu ECMAScript. Standard definuje chování JavaScriptu v prohlížečích. Záleží také na tom, jak rychle jsou schopni vývojáři prohlížečů reagovat na změny s nově vydanou definicí. Velmi progresivní v implementaci nejnovější vydané verze standardu jsou vývojáři společnosti Mozilla, kteří se snaží držet krok s poslední verzí.

Poslední vydanou verzí je verze označovaná jako ES9 (ES2018). Na první pohled se může zdát, že ECMAScript neřeší problém chování JavaScriptu, protože dle uvedeného popisu lze říci, že v jednom prohlížeči může být implementovaný

ECMAScript 2015 a v druhém může být implementace ECMAScript 2017. Většina kriticky nevyhovujících prvků jazyka však již byla ve větší míře odstraněna s vydáním ECMAScript 2009, který již je plně implementován ve všech běžných prohlížečích.

Novější verze ECMAScript přidávají spíše zjednodušení syntaxe a zabudovaní některých populárních konstruktů jiných jazyků. Skutečnost, že většina prohlížečů ještě plně nepodporuje syntaxi jazyka definovanou nejnovějším standardem, nám nezakazuje tuto syntaxi již nyní během vývoje webové aplikace použít. Musíme však použít JavaScriptový kompilátor, např. Babel. Tento kompilátor převede náš zdrojový kód na syntaxi, které již prohlížeče rozumí, a my můžeme použít teprve připravované prvky ECMAScriptu, které ještě nejsou prohlížeči plně podporovány.

V JavaScriptu existovala dříve pouze jedna možnost, jak deklarovat proměnnou, a to pomocí klíčového slova var jako variable. JavaScript však má trošku jiný způsob, jak deklaruje a inicializuje proměnné, v jazyku není platnost proměnných bloková, jako to bývá zvykem u jiných programovacích jazyků, a proto často v JavaScriptu docházelo k nechtěným přepisováním proměnných. Standard ES6 přinesl nový způsob deklarování proměnných pomocí klíčových slov const a let. Const se používá pro deklaraci proměnných, které se nedají po inicializaci již změnit. Klíčové slovo let nahrazuje var a přináší do jazyka blokové působení proměnné.

Velice vítaná změna ve standardu ES6 byla možnost vytvářet třídy a možnost dědit z ostatních tříd, tak jak je to běžné u ostatních objektově orientovaných jazyků. Dříve se OOP v JavaScriptu dědění programovalo pomocí objektu Object.prototype.

3.6 Typescript

JavaScript je řadí do skupiny dynamických jazyků, to znamená, že během programování není třeba definovat typy proměnných a návratové hodnoty funkcí. Interpret si vše odvozuje automaticky, dle daného kontextu. Dynamické přetypování má však jednu nehezkou vlastnost, a to tu, že znepříjemňuje práci programátorům, protože je velice obtížné naprogramovat chytré IDE (Integrated Development Environment), které by pomohlo předcházet běžným chybám, které lze odhalit u striktně typových jazyků, jež se kompilují. Nakonec pro JavaScript vznikl nástroj ulehčující vývojářům práci a tím je TypeScript, programovací jazyk, který se používá jako nadstavba nad JavaScriptem. Tato nadstavba do jazyka přidává typovou kontrolu a mnoho dalších atributů, převážně z objektově orientovaného programování.

Typová kontrola v jazyce hlídá, zda programátor pracuje se správnými typy proměnných, zda posílá správný počet argumentů do funkce a například zda objekt, se

kterým se pracuje, je úplný, tzn. zda jsou všechny povinné vlastnosti daného objektu správně vyplněné.

TypeScript je vyvíjen společností Microsoft. Díky TypeScriptu se programování v JavaScriptu stalo přívětivější činností a do jisté míry jednodušší, hlavně díky uvedené typové kontrole. TypeScript je označován jako transpiler. Transpilování je proces, během kterého je pozměněn původní kód. Pro využití výhod TypeScriptu je třeba psát více řádků kódu. Tyto řádky navíc hlídají programátora. Po transpilování se navíc smažou a zbyde pouze čistý JavaScript.

3.7 Node.js

Před vznikem platformy Node.js se JavaScriptový kód mohl spouštět v prohlížeči nebo ve specializovaném softwaru, který ho uměl interpretovat.

Ryan Dahl přišel v roce 2008 s myšlenkou vytvořit prostředí, ve kterém lze JavaScriptový kód spustit a tím nebýt závislý na prohlížeči. Lze říct, že tato myšlenka v té době byla revoluční, způsobila obrovský rozmach JavaScriptových aplikací v následujících letech a velice změnila styl vývoje webových aplikací. Během vývoje platformy Node.js Ryan Dahl využil volně přístupný engine V8 implementovaný společností Google v prohlížeči Chrome. Engine V8 je aplikace společnosti Google naprogramovaná v jazyku C++ sloužící k interpretování jazyka JavaScript. Společnost Google uvádí, že engine V8 je vysoce výkonný a rychlý. Jeho architektura je optimalizovaná k rychlým operacím.

Během vývoje platformy Node.js bylo JavaScriptové API rozšířeno o mnoho užitečných funkcionalit. Mezi ty nejzásadnější patří možnost pracovat v JavaScriptu se soubory, s databází a možnost dotázat se na prostředí, ve kterém aplikace Node.js běží. API pro práci se soubory v prohlížečích z bezpečnostních důvodů k dispozici není, v Node.js lze najít spoustu dalších rozšíření funkcionalit pro JavaScript.

Aplikace Node.js běží pouze na jednom vlákně oproti ostatním serverovým aplikacím, vytvořeným na technologiích PHP, ASP.NET, Ruby nebo Java, které jsou mnohovláknové. Všechny multivláknové aplikace pro každý požadavek vytváří novou instanci na novém vlákně nebo nový proces. Výhodou Node.js je fakt, že aplikace napsané v prostředí Node.js běží na jednom vláknu, jsou založené na událostech a nejsou blokované I/O operacemi. Platforma je k dispozici zdarma, což oceňuje široká a stále rostoucí komunita.

Počet výhod, které s sebou platforma Node.js přináší, je mnoho, ale mezi ty nejvýznamnější lze zařadit:

- Jednoduchá instalace, během níž není třeba složitě nastavovat prostředí, stačí pouze projít jednoduchým instalátorem a poté je platforma připravena k použití. Node.js je také multiplatformní, aplikaci lze nainstalovat na většinu operačních systémů (Windows, macOS, Linux).
- V prostředí Node.js je k dispozici mnoho balíčků, které lze využít pro rychlý vývoj aplikací. Tyto balíčky je možno stáhnout pomocí NPM (Node Package Manager) nebo YARN, který je součástí instalace.

Pokud porovnáme rozdíly mezi použitím platformy Node.js a některého z běžně dostupných prohlížečů, zjistíme, že ve většině případů je využití prohlížeče pro trénování modelu lepší variantou, protože prohlížeče obsahují již implementované rozhraní pro WebGL. Díky uvedenému rozhraní se výpočty během trénování modelu mohou přesunout na GPU, které je v těchto výpočtech rychlejší než běžné CPU; autoři knihovny Tensorflow.js uvádí, že v některých případech je rozdíl až patnáctinásobný.[7]

.....

[7] Autoři knihovny se zmiňují o rychlosti v úvodním videu dostupném na: https://www.youtube.com/watch?v=WYvgP9LfvTg&t

4 Machine learning a JavaScriptové knihovny

Machine learning zažívá v posledních letech obrovský vzestup, s tím ruku v ruce roste počet dostupných knihoven, včetně knihoven určených užšímu publiku, a to webovým vývojářům. V následující kapitole budou popsány knihovny, které lze zařadit mezi více používané nebo které jsou v některých oblastech unikátní.

Převážná část knihoven je lincencovaná jako MIT, což znamená, že zdrojové kódy může uživatel bez omezení používat a šířit i jako součást proprietárního software. Povinností uživatele je přiložení kopie dané licence a uvedení jména původního autora. Licence Apache 2.0 vyšla v roce 2004 a umožňuje uživateli svobodné užívání softwaru (distribuce, upravování). Oproti MIT nepožaduje, aby byl typ následné distribuce zachovaný, je však nutné zachovat všechny nemodifikované části projektu.

Tabulka 4 – Srovnání JavaScriptových ML knihoven

Knihovna	Licence	Verze	Velikost[8]	První verze	Ve vývoji[9]
Brain.js	MIT	1.6.1	880 KB	Leden 2018	Ano
Synaptic	MIT	1.1.4	106 KB	Červenec 2015	Ano
Ml.js	MIT	3.5.1	597 KB	Říjen 2014	Ano
Neataptic	MIT	1.2.14	99 KB	Březen 2017	Ne
Convent.js	MIT	1.1.0	92 KB	Říjen 2014	Ne
Webdnn	MIT	1.2.7	133 KB	Květen 2017	Ano
TensorFlow	Apache 2.0	1.0.0	40 MB	Březen 2018	Ano
Compromise	MIT	11.12.5	418 KB	Květen 2014	Ano
Neuro.js	MIT	-	89 KB	Říjen 2016	Ano
Mind	MIT	1.0.1	6 KB	Květen 2017	Ne
Natural	MIT	0.5.1	4.7 MB	Březen 2012	Ano
Keras.js	MIT	0.3.0	586 KB	Květen 2017	Ne

[8] Velikost nezminifikované verze (zdrojový kód je čitelný a odřádkovaný). Minifikace znamená odstranění bílých znaků a odřádkování zdrojového kódu. Minifikace způsobí zmenšení souboru, díky tomu se knihovna rychleji nahraje do prohlížeče.

[9] Pokud v posledních 12 měsících došlo k publikování nové verze.

Počet knihoven, jež jsou nyní vývojářům k dispozici, je obrovský, a jak už to bývá ve světě JavaScriptu, počet těchto knihoven neustále roste, díky velice aktivní komunitě, jednoduché správě a instalaci balíčků pomocí programů YARN a NPM.

4.1 Tensorflow.js

Vzrůstající popularita knihovny TensorFlow[10] upoutala pozornost odborníků nejen z oblasti umělé inteligence a machine learningu, ale také z řad mnoha vývojářů, programátorů a zapálených nadšenců pro umělou inteligenci. Mnoho odborníků z IT se začalo zajímat o tuto knihovnu a zkoumat, proč je tak populární.

Autoři knihovny TensorFlow ji navrhli tak, aby všechny složité věci, které děsí a odradí mnoho lidí, abstrahovali a co nejvíce zjednodušili použití knihovny. Tím umožnili jednoduší vstup lidem se zájem o machine learning.

Původní knihovna TensorFlow je volně přístupná a naprogramovaná pomocí jazyků Python, C++ a Cuda. Jádro knihovny je napsané v programovacím jazyku C++, stará se o interakci s hardwarem a obsahuje mnoho funkcionalit pro machine learning. Nadstavbou nad tímto jádrem je další API napsané v jazyku Python. Toto rozhraní má ulehčit používání celé knihovny. Autorem knihovny je tým Google Brain společnosti Google. První stabilní verze knihovny byla vydána v listopadu 2015. V současnosti je knihovna k dispozici ve verzi 1.12 a její zdrojové kódy jsou dostupné na serveru Github. Knihovna podporuje výpočet pomocí procesoru (CPU), grafické karty (GPU) a čipu TPU (tensor processing unit) vyvinutého přímo společností Google za účelem rychlejšího výpočtu při využití neuronových sítí. [41]

V počítačové historii se před mnoha lety většina operací vykonávala pouze pomocí počítačového procesoru (CPU). Počet těchto operací však rok od roku neustále přibýval a bylo nutné vymyslet nějaké řešení, které by zkrátilo čas potřebný k vykonání všech operací. Řešením bylo vytvoření grafického procesoru (GPU), který byl navržen a optimalizován pro práci s počítačovou grafikou. Obrázky jsou v počítači reprezentovány pomocí matic pixelů a právě pro počítání matic byly grafické procesory optimalizovány.

Název knihovny v sobě nese dvě důležitá slova, tensor a flow, každé z těchto slov má hlubší význam. Tensor je matematická struktura, která obsahuje čísla. Typy hodnot, které může obsahovat tensor, jsou skaláry, vektory nebo matice. Všechny uvedené typy hodnot může tensor obsahovat v n-dimenzionální podobě. Vývojáři a programátoři si mohou představit pole, které obsahuje numerické hodnoty.

[10] Internetová adresa knihovny https://www.tensorflow.org/js

Stavebním kamenem pro většinu algoritmů jsou matice čísel, které algoritmy přijímají, upravují a přeposílají, proto v sobě název knihovny nese slovo flow, které zdůrazňuje neustálou práci s maticemi.

Jediným nedostatkem knihovny TensorFlow byl fakt, že její API, přes které probíhá interakce s knihovnou, bylo naprogramované pomocí jazyka Python. Jazyk Python se nejčastěji používá pro řešení matematických problémů nebo ve statistice, lze ho však využít pro téměř jakýkoliv problém. Pokud budeme hledat nejpoužívanější programovací jazyk v oblasti machine learningu a v oboru data science, tak se s velkou pravděpodobností dopátráme právě k jazyku Python. Pro běžné vývojáře a širší veřejnost je programovací jazyk Python většinou až příliš specializovaný a většinou s ním lidé nemají zkušenost.

Autoři knihovny si uvědomili, že programovací jazyk Python může být důvodem k tomu, že někteří lidé, přestože TensorFlow usnadňuje machine learning, tuto knihovnu nevyužijí, protože neovládají programovací jazyk Python. Zaměstnancům ze společnosti Google také neunikl rostoucí trend jazyka JavaScript a přemýšleli, jestli by nedokázali udělat také rozhraní pro knihovnu TensorFlow v jazyku JavaScript, aby tím ještě více zpopularizovali knihovnu a dostali jí do širšího povědomí. Nakonec se jejich myšlenka stala skutečností a vznikla knihovna TensorFlow.js.

Během vývoje knihovny TensorFlow.js autoři narazili na problém, jak pomocí JavaScriptu ovládat jádro knihovny TensorFlow naprogramované v jazyku C++. JavaScript je interpretovaný jazyk, a proto je závislý na systému, který bude jeho příkazy interpretovat. Pro využití funkcionality jádra TensorFlow knihovny v prohlížeči bychom museli donutit vývojáře internetových prohlížečů, aby do prohlížečů implementovali TensorFlow knihovnu a udělali nad ní wrapper, který by umožnil JavaScriptu knihovnu použít. Toto řešení je však nereálné, protože existuje mnoho internetových prohlížečů a určitě by si vývojáři těchto prohlížečů nechtěli zbytečně zanášet a udržovat cizí knihovnu. Autoři Tensorflow.js se proto rozhodli přepsat většinu funkcionality z jádra TensorFlow do JavaScriptu a využít standard WebGL, umožňující využití grafického procesoru v prohlížeči. Standard WebGL většina prohlížečů podporuje, a je tedy možné využívat pro výpočty grafický adaptér přímo z prohlížeče. [42]

4.1.1 Keras

V minulé kapitole bylo zmíněno, že jádro knihovny TensorFlow je napsané v jazyce C++. Ten je úzce spjat s nízkoúrovňovým (low-level) programováním. Nízkoúrovňové programování si můžeme představit jako programování, při kterém se musíme starat o hardwarové prostředky, například alokace pamětí RAM. Většinou při nízkoúrovňovém programovaní nepracujeme s abstrakcí, ale ovládáme přímo daný

hardware. Psaní ovládacích programů pro specifický hardware lze také nazvat nízkoúrovňovým programováním.

Keras je vysokoúrovňové API, které slouží jako nadstavba nad knihovnou TensorFlow. Cílem projektu Keras mělo být sjednocení API knihoven pro práci s neuronovými sítěmi, tak aby jednotlivé knihovny byly lehce zaměnitelné. Keras je naprogramovaný v jazyku Python. Zároveň díky sjednocení API mělo dojít ke zjednodušení používání dalších knihoven. Všechny knihovny šlo používat pomocí stejného rozhraní, díky tomu bylo možné velice rychle vyzkoušet jinou knihovnu a zjistit, zda nám neposkytne lepší výsledky. Další knihovny, které lze pomocí Keras využít, jsou CNTK (Microsoft Cognitive Toolkit) nebo Theano. [43]

Keras se nedávno stal součástí přímo TensorFlow. Lehce modifikované Keras API existuje také nad knihovnou Tensorflow.js a obsahuje v sobě definici pro práci s vrstvami a jádro původního Keras API. V Tensorflow.js můžeme brát vrstvy přímo jako celý Keras.

4.1.2 Tensor

Základním stavebním prvkem Tensorflow.js je tensor. Pro práci s knihovnou je vždy důležité, s jakým typem hodnot má knihovna pracovat. Tvar (shape) těchto hodnot se specifikuje právě při vytváření prvku tensor. Pokud by se pracovalo s obrázkem, který bude definován 128×128 pixelů, tvar tensor objektu bude [128, 128]. Při využití více obrázků, například při zpracování 250 obrázků, se tvar definovaného tensor objektu změní na [250, 128, 128]. Tensor je ve skutečnosti komplexní objekt, který v sobě nese mnoho dalších informací, například jaké má id či jakého je datového typu.

Základní API nám během vytváření objektu tensor umožňuje specifikovat 3 parametry. Prvním a jediným povinným parametrem jsou hodnoty, se kterými budeme pracovat. Druhým parametrem je, jaký tvar má tensor mít, zpravidla specifikujeme matici, do které budeme vstupní hodnoty transformovat. Třetím parametrem je datový typ vstupních hodnot. Možné hodnoty jsou float32, int32, bool, complex64 nebo string. Během vytváření objektu je třeba dávat pozor, zda máme správný počet vstupních hodnot v případě, že specifikujeme tvar tensoru. Například pokud je specifikovaný tvar 2×2, počet vstupních hodnotu musí být 4. Pro vytváření objektu tensor můžeme využít generickou funkci a specifikovat všechny uvedené hodnoty nebo můžeme využít specifické funkce pro vytváření hodnot. Obě varianty vytvoří stejný objekt. Využití specifických funkcí je doporučeno, jelikož se tím usnadňuje čtení kódu.

4.1.3 Operace a proměnné

Tensory mají jednu důležitou vlastnost, která ještě nebyla zmíněna, a tou je neměnnost. Tensory, které pomocí Tensorflow.js vytvoříme, jsou neměnné, po definování již nelze měnit jejich hodnoty. Vždy když definujeme nový tensor objekt, tak se celý tento objekt zkopíruje do paměti grafického procesoru. Pokud bychom chtěli získat hodnotu definovaného tensor objektu včetně jeho dat, musíme použít funkci, která nám tato data vytáhne z grafické paměti. K dispozici jsou v knihovně Tensorflow.js dvě metody. Asynchronní a synchronní způsob. Jestliže pracujeme s velkým množstvím dat, je lepší přistupovat k nim asynchronně pomocí JavaScriptových promisů. Při asynchronním přístupu nedochází k blokování pracovního vlákna čekáním na data. Synchronní způsob můžeme využít, pokud pracujeme s menším objemem dat, v tomto případě blokace primárního vlákna není postřehnutelná.

Velice často je nutné provádět s maticemi různé transformace, nejčastěji násobení a sčítání. Tensoru však nelze měnit přímo jeho hodnoty, pro změnu hodnot je třeba využít příslušné funkce Variable. Funkce vytvoří novou proměnnou pro příslušný tensor, kterému chceme měnit data.

Pro většinu matematických operací pro tensory existuje již v knihovně implementace. Při aplikování těchto operací je třeba dávat pozor, zda je daná operace přípustná pro daný tvar tensoru.

Většina moderních programovacích jazyků má již v sobě implementovaný Garage Collector, což je funkcionalita, která automaticky spravuje využívání paměti počítače. Pokud je v programu nevyužitá proměnná, která byla deklarována, ale již se nevyužívá, Garbage Collector tuto proměnnou automaticky smaže z paměti a uvolní místo. Účel Garbage Collectoru je hlídat již nevyužité objekty a proměnné programu a uvolňovat je z paměti. V jazycích, kde Garbage Collector není k dispozici, si musí nevyužité objekty a proměnné hlídat programátor sám, aby nedocházelo k únikům paměti (Memory Leak). Memory Leak je situace, kdy se v paměti hromadí nevyužité objekty a postupně zaplňují celou paměť počítače, většinou je tato chyba způsobena programátorem, protože zapomíná odstraňovat nevyužité proměnné z paměti. JavaScript patří k jazykům, které garbage collector implementovaný mají, tudíž automaticky čistí paměť od nevyužitých objektů. Při využívání knihovny Tensorflow.js je na programátorovi, aby se staral o využití grafické paměti, protože, jak již bylo zmíněno výše, při vytvoření objektu tensor je jeho hodnota zkopírována do grafické paměti. Grafickou paměť je třeba čistit od již nevyužívaných tensorů, aby nedošlo k jejímu přetečení. Pro manipulaci grafické paměti je v knihovně k dispozici několik funkcí. [44]

4.1.4 Model a vrstvy

Vrstvy jsou primárním stavebním blokem při definici modelů. Každá vrstva má svoje vstupy a zároveň svoje výstupy, ve většině případů se ve vrstvě provádějí nějaké výpočty. Definice vrstev je součástí Tensorflow.js CoreAPI.

Vrstvy modelu se starají automaticky o inicializaci většiny potřebných proměnných, včetně nastavení vah. Tím se vytváření modelu zjednodušuje. [45]

4.1.5 Jednoduchá neuronová síť

Během implementace neuronové sítě nejdříve začínáme s definicí tréninkových dat pomocí tensorů. Většinou definujeme set dvou tensorů, které reprezentují vstupy a výstupy. Při jejich definici je třeba brát ohled na to, aby si dimenze tensorů byly rovny. Příkladem tréninkových dat pro neuronovou síť učící se XOR[11] operaci by tréninková data vypadala jako na obrázku č. 21, kde vycházejí z XOR pravdivostní tabulky. [46]

Obrázek 21 – Tréninková data pro operaci XOR

```
const xs = tf.tensor2d([[0, 0], [0, 1], [1, 0], [1, 1]]); // Vstup
const ys = tf.tensor2d([[0], [1], [1], [0]]); // Výstup
```

zdroj: autor

Po definici tréninkových dat se vytváří definice modelu. Model se skládá z vrstev a v každé vrstvě lze definovat její typ. Pro jednoduché příklady lze vystačit s typem vrstvy **dense**, jedná se o plně propojenou vrstvu. K dispozici jsou i další typy vrstev, například convolutional.

Při konfiguraci dense vrsty je nutné nastavit počet vstupů a to, jakou aktivační funkci použít. Vrstvu lze konfigurovat mnohem jemněji volitelnými parametry, pro první vrstvu lze definovat počet vstupních parametrů. Příkladem definice modelu neuronové sítě pro XOR operaci může být obrázek č. 22. Nejdříve vytvoříme sekvenční model (sequential model), který později konfigurujeme. Sekvenční model je model, kde výstupy vrstvy jsou zároveň vstupy pro nadcházející vrstvu, podobně jako funguje zásobník. Žádná vrstva se nepřeskakuje a ani se nikam nevětví. [47] Další operací, kterou je nutné definovat před použitím modelu, je **compile**.

Funkce compile připraví model pro trénování a vyhodnocování vstupů. Funkce má na vstupu tři parametry, typ optimalizační funkce, která v sobě obsahuje definici

[11] XOR (Exclusive OR) – Je logická operace, ve které je výstup pravdivý, pokud se vstupy liší

rychlosti učení (learning rate) a loss funkce, a poslední nepovinný parametr určuje, jaké metriky se mají použít během trénování a testování. Knihovna Tensorflow.js se neustále vyvíjí a počet možných typů u optimalizačních nebo aktivačních funkcí nebo jejich názvů se může v budoucnosti změnit.

Obrázek 22 – Definice modelu neuronové sítě

```
createModel() {
  const model = tf.sequential();
  model.add(tf.layers.dense({units: 8, inputShape: [2], activation: 'tanh'}));
  model.add(tf.layers.dense({units: 1,  activation: 'sigmoid'}));
  model.compile({optimizer: 'sgd', loss: 'binaryCrossentropy', });

  return model;
}
```

zdroj: https://medium.com/tensorflow/a-gentle-introduction-to-tensorflow-js-dba2e5257702

Po konfiguraci modelu je vhodné model začít učit, k tomu slouží funkce **fit**. Vstup do fit funkce jsou vstupní a výstupní tréninková data (xs, ys) a poté konfigurační objekt, ve kterém lze definovat počet vzorků, než proběhne aktualizace gradientu pomocí parametru **batchSize**. Lze také konfigurovat počet iterací za tréninkovými daty, které se nazývají **epochs**. Lze také nastavit parametr **verbose**, který ukazuje stav učení modelu. [47]

Obrázek 23 – Tensorflow.js fit funkce

```
await model.fit(xs, ys, {
  batchSize: 1,
  epochs: 1000
});
```

Posledním krokem je využití natrénovaného modelu pro predikci nových hodnot. Natrénovaný model pro předpovězení hodnoty použijeme pomocí funkce **predict**. Parametr funkce jsou hodnoty, pro které chceme získat pomocí modelu výsledek. Na příkladu s XOR operací lze ověřit model pomocí vstupních hodnot, viz obrázek č. 21 a tabulky č. 5.

Obrázek 24 – Tensorflow.js prediction funkce

```
const predictions = model.predict(xs);
```

Tabulka 5 – XOR pravdivostní tabulka

Input		Output
A	**B**	
0	**0**	0
0	**1**	1
1	**0**	1
1	**1**	0

4.2 Brain.js

Velice populární knihovnou pro tvorbu vlastních neuronových sítí pomocí JavaScriptu je Brain.js[12]. Knihovna je oblíbená jak mezi studenty, tak mezi vývojáři. Dokumentace knihovny je velice přehledná a vše je vysvětleno, tak aby knihovnu dokázal použít uživatel i bez hlubších znalostí neuronových sítí.

Knihovnu lze použít buď přímo v prohlížeči, nebo též na platformě Node.js. Instalace v prostředí Node se realizuje skrze NPM nebo YARN. Pro použití v prohlížeči stačí stáhnout knihovnu a připojit ji pomocí skriptu do HTML souboru, u prohlížeče se však setkáme s výkonnostním omezením, proto se doporučuje použití technologie Web Worker, která umožňuje spustit JavaScriptový kód na pozadí na jiném vláknu.

API knihovny je intuitivní, pro trénování modelu je k dispozici funkce train(), které na základě vybraného algoritmu pro tvorbu neuronové sítě poskytneme vstupní data. Trénink neuronové sítě lze parametrizovat pomocí objektu, ve kterém můžeme například specifikovat počet iterací, logování, akceptovatelnou chybu z trénovacích dat.

Tabulka 6 – Typy neuronových sítí v knihovně Brain.js

Feedforward Neural CPU	Time Step Gated Recurrent Unit
Feedforward Neural GPU	Recurrent Neural Network
Time Step Recurrent Neural Network	Long Short Term Memory Neural Network
Time Step Long Short Term Memory	Gated Recurrent Unit

[12] Internetová adresa knihovny https://github.com/BrainJS/brain.js

V knihovně ve verzi 1.6.1 lze nalézt 8 typů neuronových sítí, každý typ neuronové sítě se hodí pro jiný typ úlohy. Knihovna obsahuje také podpůrné API pro vizualizaci topologie neuronové sítě.

4.3 Ml5.js

Většina dostupných knihoven již předpokládá alespoň základní znalost programování a machine learningu. U knihovny Ml5.js [13]se autoři snažili usnadnit její použití tak, aby byla přístupná široké veřejnosti, umělcům, kodérům a studentům. Vývoj knihovny probíhá na New York University a první verze knihovny byla veřejnosti dostupná v červenci 2018. Autoři se při vývoji inspirovali knihovnami Processing a P5.js[14]. Knihovna funguje jako wrapper[15] nad knihovnou Tensorflow.js a je bez jakýchkoliv dalších externích závislostí. Wrapper slouží k ulehčení práce s knihovnou pro uživatele, kteří nemají prakticky žádné zkušenosti v oblasti machine learningu. Knihovna umí pracovat s běžnými typy dat, jako jsou obrázky, text, zvuk a hudba.

Použití knihovny se nijak neliší od použití ostatních knihoven, opět stačí nareferencovat knihovnu pomocí skript tagu v HTML souboru. Model lze využít již natrénovaný nebo si vytrénovat svůj vlastní. Pro klasifikaci obrázků se nejčastěji setkáme s modelem MobileNet[16], který byl optimalizován pro využití v malých mobilních nebo webových aplikacích. MobileNet je kompromisem mezi velikostí, přesností a odezvou. Většina funkcí obsažených v knihovně je asynchronních. Důvodem jsou delší výpočty, které zpracovávají dotazy a generují výsledek. Během volání funkcí můžeme využívat návrhový vzor **error-first callback**, který je velice blízký vývojářům z Node.js platformy, nebo klasicky pomocí **promises.**

Nejběžnější využití knihovny najdeme v jednoduchých mobilních a webových aplikacích sloužících jako nástroj pro zaučení méně zkušených uživatelů do oblasti machine learning.

[13] Internetová adresa knihovny https://ml5js.org/

[14] Knihovny jsou dostupné na https://processing.org/ a https://p5js.org/

[15] Jako wrapper lze označit obal nad již existující funkcionalitou. Wrapper se používá převážně pro zjednodušení komplikovaného API. Wrappery umožnují sjednocovat funkcionalitu různých knihoven.

[16] Model MobileNet je dostupný na internetových stránkách https://github.com/tensorflow/tfjs-models/tree/master/mobilenet

4.4 Synaptic

Knihovnu Synaptic[17] lze využít k sestavení vlastní neuronové sítě, lze pomocí ní definovat vlastní neurony, vrstvy a poté celé neuronové sítě. Vše lze skládat dohromady a tvořit různé kombinace. Dokumentace knihovny obsahuje návod pro uživatele bez zkušeností, ale také podrobné informace k parametrizovaní neuronů, vrstev i sítí. Vlastním skládáním neuronů lze sestavit komplexní a flexibilní architekturu celé neuronové sítě.

V knihovně jsou k dispozici již předpřipravené typy architektur, například multiplayer perceptrons, long short-term memory sítě, liquid state machines a Hopfieldovy sítě. Knihovnu lze použít v prohlížeči nebo na platformě Node.js. Knihovna je vybavena také API Trainerem pro trénování vlastní sítě, který obsahuje též funkcionalitu pro testování výkonu trénované sítě.

Pro definování neuronu lze definovat, s jakým dalším neuronem je definovaný neuron propojen. Neuron lze propojit i sám se sebou. Dále lze definovat hodnotu aktivace, která musí být v rozmezí od 0 do 1. Rovněž lze definovat funkci propagate, díky které lze definovat, jaká je cílová hodnota neuronu. Pro definování funkce propagate je nutné poskytnout dva parametry. První parametr je rychlost učení a druhý cílová hodnota, která opět musí být v rozmezí od 0 do 1. Přednastavenou aktivační funkcí neuronů je sigmoid funkce, k dispozici jsou však i další; hodnotu bias lze rovněž nastavit.

Běžně se v neuronových sítích nepracuje pouze s neurony, ale s vrstvami. Vrstvy jsou pole neuronů a lze pro ně definovat prakticky stejné náležitosti jako u neuronů. Neuronové sítě se definují pomocí vrstev a opět se definují stejnými parametry jako vrstvy a neurony. Neuronové sítě lze propojovat mezi sebou a lze je exportovat do formátu JSON nebo se dají exportovat do jedné JavaScriptové funkce.

Knihovna Synaptic umožní náhled do jednotlivých stavebních bloků celé neuronové sítě, díky tomu si uživatel může udělat praktickou představu, jak neuronové sítě fungují a z jakých částí se skládají.

.....

[17] Internetová adresa knihovny http://caza.la/synaptic/#/

4.5 ConvNetJS

Další knihovnou vyvíjenou na stanfordské univerzitě je ConvNetJS[18], podle oficiálních stránek je knihovna určena pro trénování neuronových sítí v prohlížeči. Knihovna nevyužívá žádné další externí závislosti a během trénování nevyužívá GPU.

Knihovna se v zásadě nijak neliší od ostatních knihoven při definování neuronové sítě, využívá se pouze jiné syntaxe při její definici. Knihovna obsahuje experimentální modul pro reinforcement learning založený na metodě deep Q learning. Vývoj knihovny je již pozastaven, poslední aktualizace repositáře proběhla v listopadu 2016.

4.6 WebDNN

WebDNN[19] (Web Deep Neural Network) knihovna se již odlišuje od ostatních knihoven. Ostatní uvedené knihovny sloužily jak k definování vlastních sítí, tak k jejich trénování. WebDNN se soustředí pouze na práci s již existujícími modely s cílem poskytovat výsledky na již definovaných modelech co nejrychleji.

WebDNN umí pracovat s modely, které jsou definované různými frameworky. Autoři knihovny si jsou vědomi, že práce s rozsáhlými modely vyžaduje velký výpočetní výkon, kterého ne každý uživatel internetu dosahuje. Díky optimalizaci natrénovaného modelu dochází ke zrychlení vyhodnocování vstupních hodnot modelem, tato optimalizace se děje skrze technologie WebAssembly a WebGPU. Knihovnu lze nainstalovat pomocí programu Node Package Manager a využívat ji v prostředí Node.js, autoři knihovny také definovali v balíčku definiční soubory pro TypeScript.

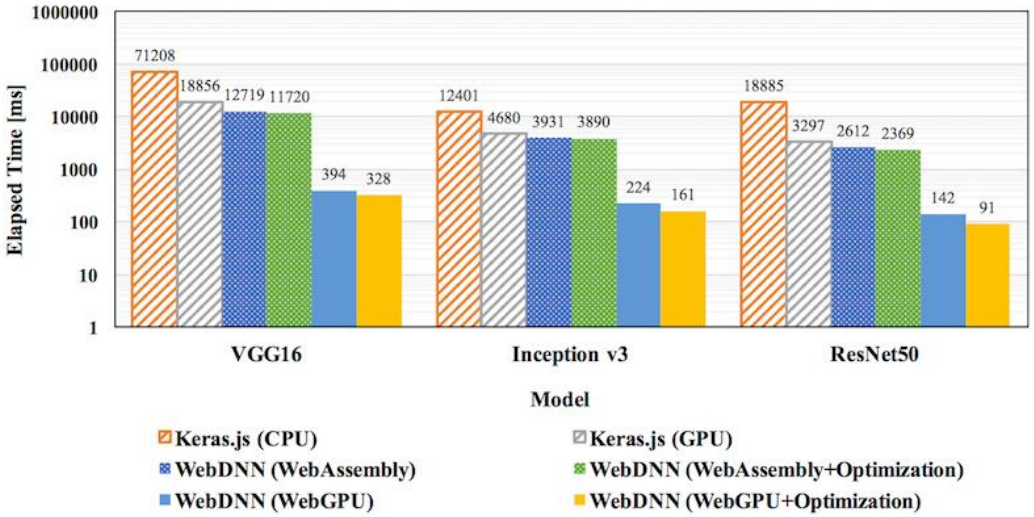
Na obrázku č. 25 je vidět porovnání rychlosti modelů spuštěných v různých prostředích a s využitím různých technologií (WebGL, WebAssembly). Test byl proveden na běžném laptopu (procesor i5 2,7 GHz, 16 GB Ram, GPU Intel Iris Graphics 6100).

.....

[18] Knihovna je dostupná na adrese https://cs.stanford.edu/people/karpathy/convnetjs/

[19] Internetová adresa knihovny https://mil-tokyo.github.io/webdnn/

Obrázek 25 – Porovnání rychlosti modelů v různých prostředích



zdroj: https://github.com/mil-tokyo/webdnn

Tabulka 7 – Vlastnosti natrénovaných modelů

Model	Velikost	Top-1[20]	Top-5[21]	Parametry
VGG16	528 MB	0.713	0.901	138,357,544
Inception v3	92 MB	0.749	0.921	23,851,784
ResNet50	98 MB	0.779	0.937	25,636,712
MobileNet	16 MB	0.704	0.895	4,253,864
NASNetMobile	23 MB	0.744	0.919	5,326,716

zdroj: https://keras.io/applications/#models-for-image-classification-with-weights-trained-on-imagenet

Z obrázku je patrné, že po optimalizaci dochází k výraznému zrychlení. Nicméně lze konstatovat, že v případě využití technologie WebGPU jsou si hodnoty jednotlivých prostředí blízké. API pro práci s knihovnou je velice jednoduché, prakticky je k dispozici pouze funkce pro nahrání modelu, poté již máme k dispozici buffer, do kterého vkládáme vstupní hodnoty obrázků, a výstupní buffer, v němž máme k dispozici výsledky. Operace se opět provádí v knihovně asynchronně, aby neblokovaly vlákno prohlížeči.

.....

[20] Přesnost, s níž bude obrázek klasifikován do správné kategorie.

[21] Přesnost, s níž bude obrázek patřit alespoň do jedné z prvních 5 kategorií.

5 Praktická část

V následujících kapitolách praktické části diplomové práce bude popsáno využití JavaScriptové knihovny Tensorflow.js pro tři různé úlohy včetně implementace základní webové aplikace.

Praktická část obsahuje také kapitolu věnovanou výběru technologií pro implementaci aplikace včetně krátkého popisu. Velké množství obrázků aplikace je umístěno v příloze včetně popisu uživatelského prostředí.

První ukázka knihovny proběhne na jednoduchém příkladu lineární regrese, kde vstupem pro trénování modelu budou body definované uživatelem pomocí HTML elementu canvas nebo náhodně generovaný vzorek dat. Pomocí knihovny se pokusíme natrénovat parametry *b* a *m* pro definici přímky pomocí proměnných knihovny Tensorflow.js. Součástí příkladu využití knihovny pro lineární regresi bude natrénování jednoduché neuronové sítě pomocí sekvenčního modelu a jedné vrstvy. V teoretické části je zmíněno, že neuronové sítě lze využít místo jakékoliv funkce a aproximovat její průběh. Poté porovnáme hodnoty pomocí předpisu přímky a hodnoty neuronové sítě.

Stránka s úlohou lineární regrese obsahuje několik boxů, pomocí kterých lze aplikaci ovládat. Na obrázku č. 26 jsou jednotlivé boxy očíslované. První box obsahuje HTML canvas element, pomocí kterého lze naklikat do plátna trénovací body. Pod plátnem se nachází seznam všech bodů, které budou použity pro trénování.

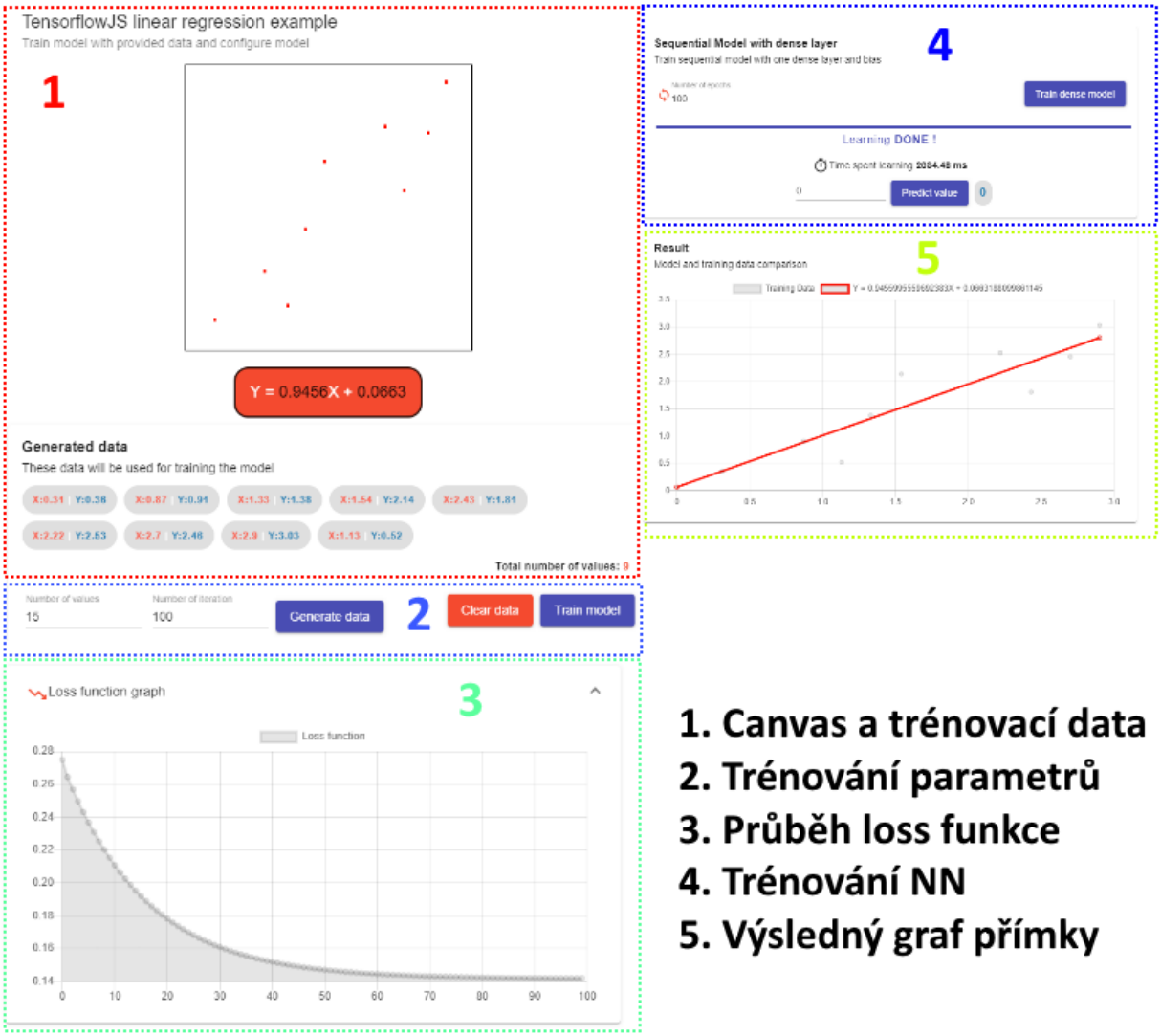
Box 2 obsahuje tlačítko pro generování náhodných dat včetně jejich mazání. Box také obsahuje ovládací prvek, pomocí kterého lze změnit počet iterací během odhadování parametrů *m* a *b* pro přímku. Pod canvasem se také zobrazuje definice přímky s parametry po procesu učení.

Box 3 obsahuje graf s průběhem loss funkce, počet bodů v grafu závisí na počtu iterací.

Box 4 obsahuje ovládací prvky pro trénování neuronové sítě, v boxu lze nastavit počet epochs, což je vlastně to samé jako počet iterací pro odhadování parametrů *m* a *b*. Box také obsahuje informaci o čase, který trval, než byla neuronová síť natrénována, po natrénování sítě se v boxu objeví také možnost zadat číslo a nechat si pro něj spočítat předpokládanou hodnotu.

Poslední box zobrazuje graf s body reprezentujícími trénovací data a také přímku, která byla vykreslena pomocí proměnných reprezentujících parametry přímky.

Obrázek 26 – Příklad lineární regrese – uživatelské prvky

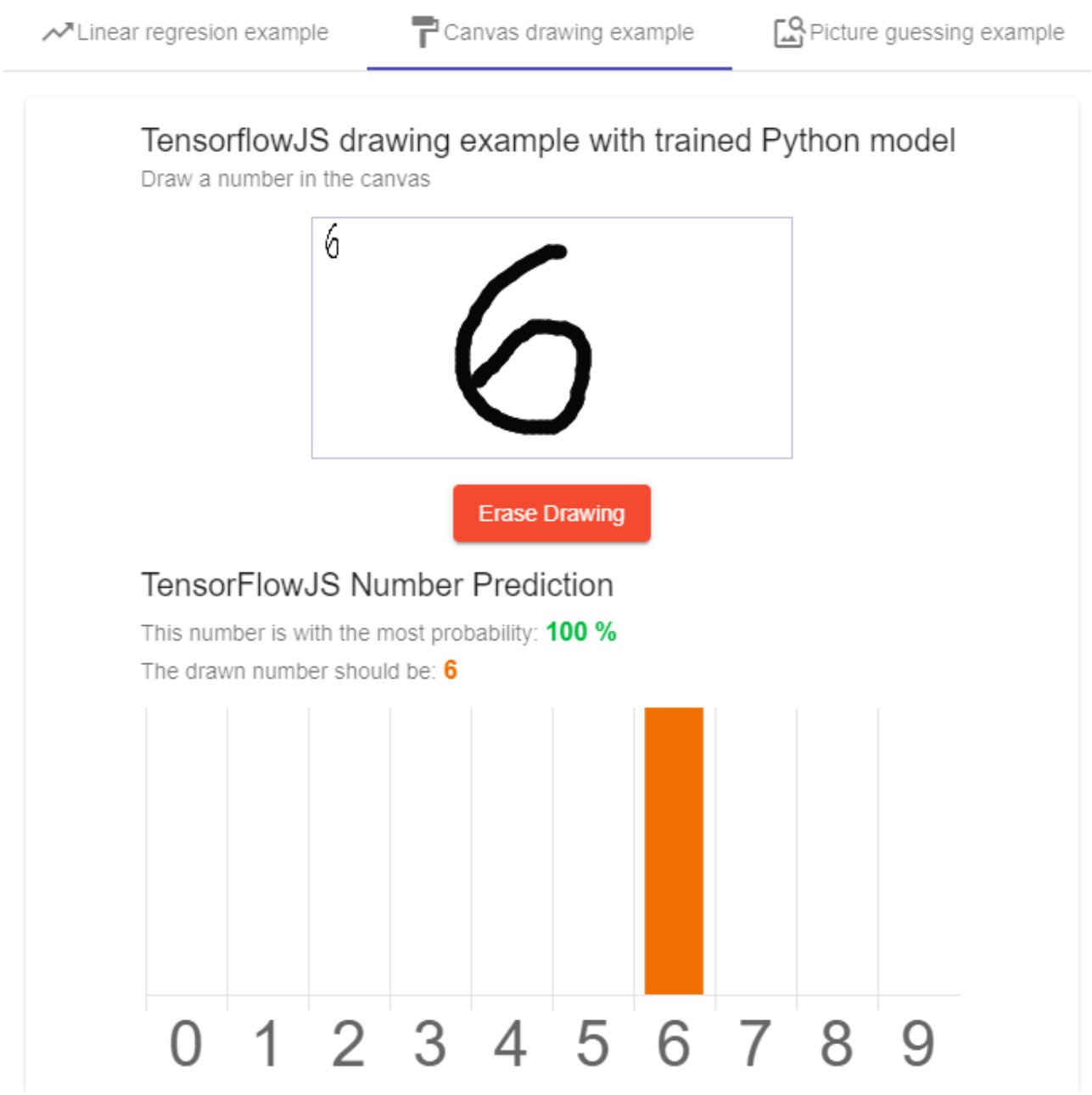


zdroj: autor

Druhý příklad využití knihovny bude pro rozpoznávání čísel, která uživatel pomocí myši kreslí do canvasu. Pro vyhodnocení nakreslené číslice je použit předtrénovaný model na databázi MNIST. Součástí kapitoly je porovnání úspěšnosti klasifikace čísel. Ukázka druhého příkladu praktické části je zobrazená na obrázku č. 26.

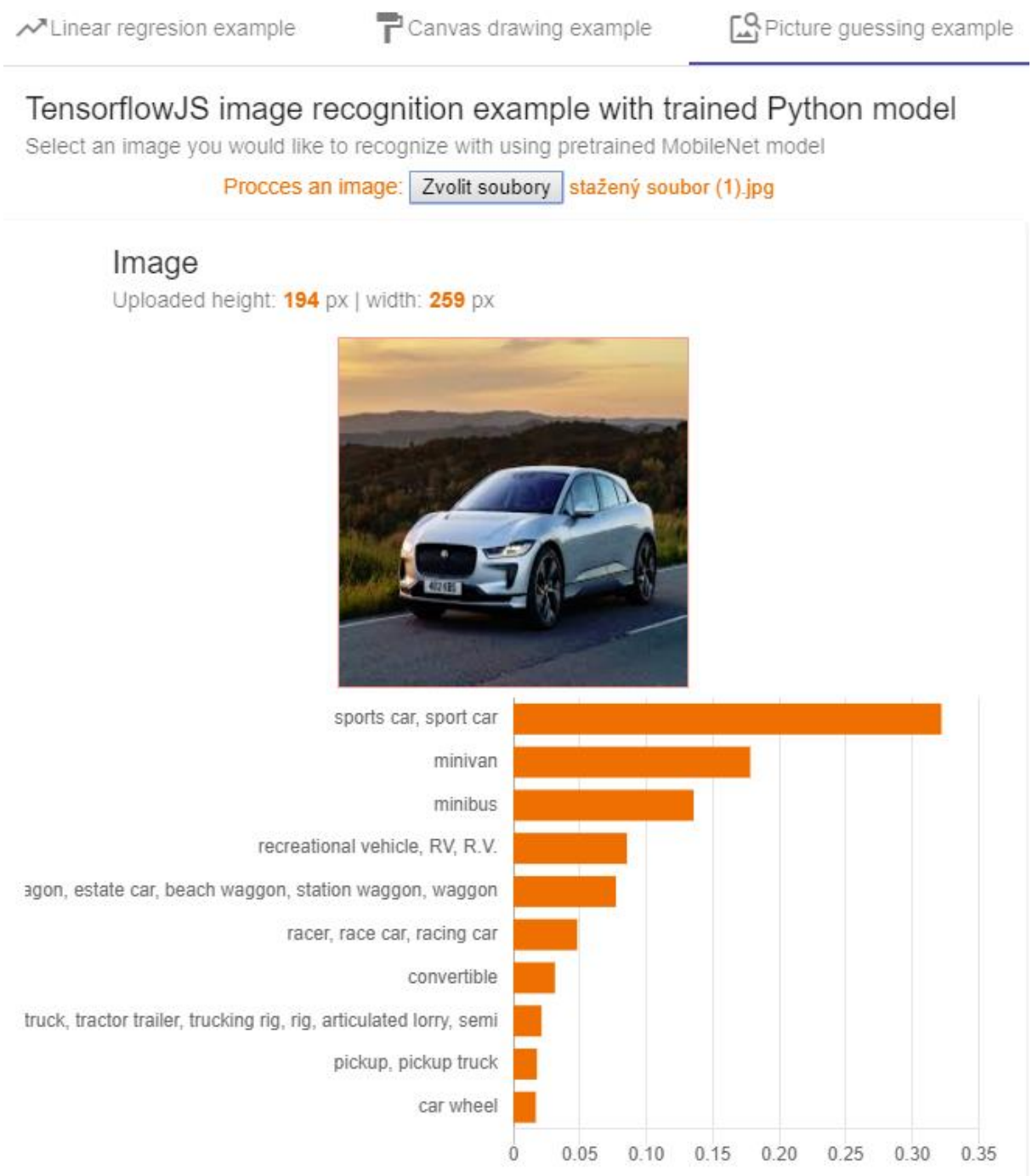
Posledním, třetím příkladem je klasifikace obrázků, k níž byl také použit předtrénovaný model MobileNet neuronové sítě, volně dostupný ke stažení. Kapitola obsahuje ukázku úspěšnosti klasifikace na jednotlivých vybraných obrazech a také porovnání, o kolik se klasifikace změní, pokud se stejný vstup upraví formou efektu blur nebo greyscale. Implementaci úlohy pro klasifikaci obrázků je možné vidět na obrázku č. 27.

Obrázek 27 – Hádání nakreslených čísel pomocí Tensorflow.js



zdroj: autor

Obrázek 28 – Klasifikace obrázku pomocí Tensorflow.js



zdroj: autor

5.1 Příprava prostředí a instalace knihovny

Pro tvorbu webové aplikace existuje mnoho programovacích jazyků a nástrojů, v následující kapitole jsou stručně popsány technologie použité pro implementaci, včetně dostupných alternativ, které lze také využít. Uvedený výběr technologií a nástrojů je zdůvodněn.

5.1.1 Systém pro správu zdrojového kódu

Téměř každá aplikace se již v dnešní době udržuje pomocí systému pro správu zdrojového kódu (version control system). Tyto systémy se starají o ukládání jednotlivých změn zdrojového kódu, včetně ukládání dodatečných informací o změně, do kterých například patří jméno autora, datum a popis změny. Jedním z hlavních předností systémů pro správu zdrojového kódu je možnost vrátit změny kódu do určitého bodu a možnost spolupráce na funkcionalitě s ostatními autory.

Pro verzování aplikace je použit systém Git a projekt je hostovaný na serveru GitHub, kde je volně k dispozici na adrese https://github.com/NutCrackee/angular-tensorflowjs. Git je oblíbený a mezi vývojáři velice rozšířený systém, umožňující vytváření branches, což lze nazvat kopií verze repositáře z určitého stavu, přičemž je možné na této kopii nezávisle pracovat a později tyto změny aplikovat. Celý systém je distribuovaný, tzn. že každý, kdo pracuje s repositářem, vlastní celou jeho kopii z vybraného bodu. Výhodou distribuovaných systémů je skutečnost, že neexistuje jediné místo, kde se ukládají změny repositáře, ale všichni uživatelé mohou zastoupit hlavní uzel v případě nějakého výpadku.

5.1.2 Program na správu balíčku

Pro instalaci balíčku knihoven třetích stran jsou k dispozici programy Node Package Manager (NPM), který je součástí instalace platformy Node.js. NPM je největší registr knihoven, obsahuje jich více jak 800 000. Alternativním programem pro správu balíčků je YARN, který se v některých funkcionalitách mírně liší a některé má navíc.

Programy pro správu balíčků pomáhají udržovat správné verze knihoven třetích stran a upozorňují, pokud dojde ke střetu nekompatibilních verzí balíčků. Ve složce s projektem pracují se souborem **Project.json**, který obsahuje seznam používaných knihoven včetně jejich verze. Pokud pracujeme s projektem, ve kterém je obsažen soubor Project.json a chceme nainstalovat potřebné závislosti, tak v příkazovém řádku projektu spustíme příkaz **npm install**. Package manager poté stáhne veškeré závislosti do složky **Node_modules**.

5.1.3 Front-end framework

Mezi lídry v oblasti frameworků určených pro vytváření uživatelského rozhraní lze zařadit frameworky Vue, React, Angular a jeho starší verzi AngularJS. Na obrázku č. 22 je uveden graf oblíbenosti frameworků na serveru GitHub, kde je většina frameworků k dispozici ke stažení.

AngularJS je vyvíjen společností Google a jeho první verze vyšla v roce 2010. V roce 2016 Google vydala Angular 2, který se již od původního frameworku lišil, a proto byl přejmenován z AngularJS na Angular, společnost se Google však i nadále snaží původní framework AngularJS udržovat aktualizovaný. Vue je nejmladším frameworkem z uvedené čtyřky, první verze vyšla roku 2014, framework si získal oblibu díky rychlé přímce učení a navíc svojí malou velikostí. React vyvíjený společností Facebook je framework založený na virtuálním DOM, díky kterému lze vykreslovat velké množství komponent, a přitom nezpomalovat zbytečně prohlížeč; první verze byla dostupná v roce 2013.

Tabulka 8 – Srovnání FE frameworků

	Angular	**React**	**Vue**
První verze	2010	2013	2014
Oficiální stránka	Angular.io	Reactjs.org	Vuejs.org
Velikost cca.	500	100	80
Současná verze	7	16.6.3	2.17
Používán	Google, Wix	Facebook, Uber	Alibaba, GitLab

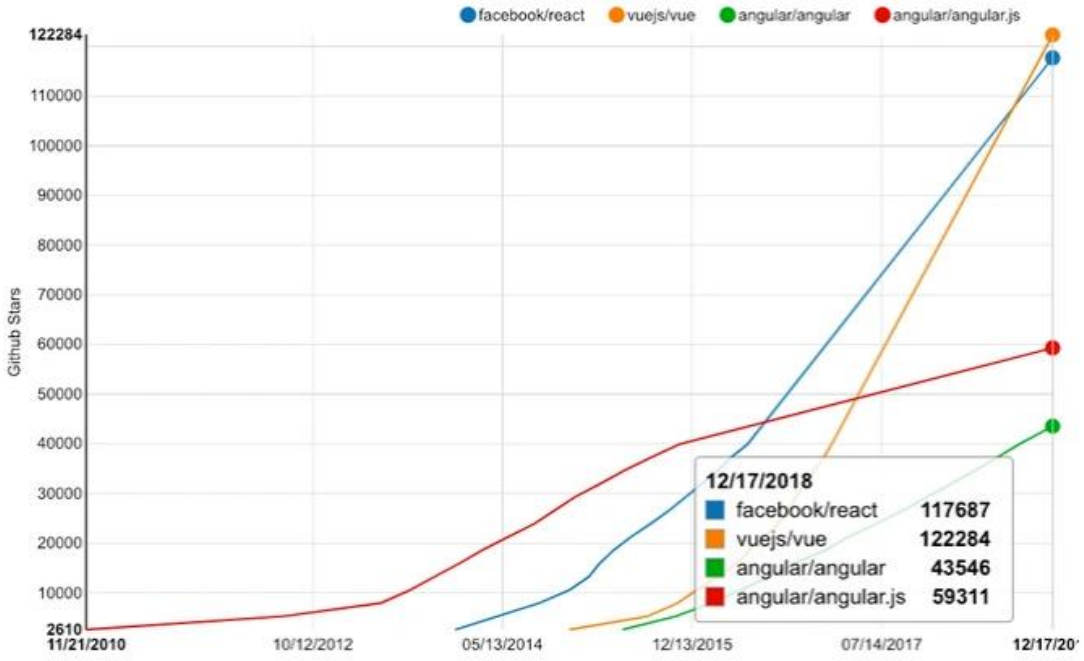
zdroj: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/

Pro vývoj webové aplikace lze použít všechny uvedené frameworky. Žádný z nich neovlivňuje práci s knihovnou Ternsorflow.js. Při použití frameworku React a Vue je trochu komplikovanější využití typové kontroly JavaScriptu pomocí jazyka TypeScript. Obě knihovny React a Vue byly původně založené pouze na JavaScriptu a podpora využití TypeScriptu se přidala do frameworků až později. Ke všem uvedeným frameworkům je k dispozici CLI[22], pomocí kterého lze generovat předpřipravené aplikace s různou možností úpravy požadavku uživatele nebo nastavit prostředí, v němž se webová aplikace spouští.

[22] Command Line Interface – program v příkazovém řádku, používaný převážně v terminálech.

Angular již byl vyvíjen pomocí jazyka TypeScript v první verzi, proto je jeho podpora ve frameworku větší, navíc je velice komplexní a obsahuje v sobě již mnoho předdefinované funkcionality, která může být pro méně zkušeného programátora těžká na pochopení. Pro zkušenějšího programátora však Angular nabízí výhody v rapidně rychlém vývoji aplikací, díky již předdefinovaným funkcionalitám lze ušetřit spoustu času.

Obrázek 29 – Srovnání Vue, React a Angular



zdroj: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/

5.2 Příprava dat

Modely určené pro aplikaci je třeba nejdříve natrénovat na trénovacích datech. Ve světě machine learningu se pro klasifikaci ručně psaných čísel často používá databáze MNIST[23]. Ukázku dat obsažených v databázi MNIST je možné vidět na obrázku č. 28. Databáze je veřejně přístupná, a proto je používána často pro vytváření různých modelů. MNIST obsahuje 70 tisíc černobílých obrázků číslic ve formátu 28×28 pixelů. Je rozdělená na 60 tisíc obrázků určených jako tréninková data a zbylých 10 tisíc slouží k ověřování přesnosti modelu.

[23] MNIST (Modified National Institute of Standards and Technology database) je databáze obrázků ručně psaných čísel. Databáze je dostupná na adrese http://yann.lecun.com/exdb/mnist/

Proběhlo již mnoho pokusů vytvořit model s co nejmenší chybovostí, autorům databáze se podařilo vytvořit model s chybovostí 0,8 % [48], nakonec byli překonáni metodou MCDNN[24] s chybovostí 0,23 % v práci Multi-column Deep Neural Networks for Image Classification [49].

Aplikace na rozeznávání čísel bude na základě natrénovaného modelu rozeznávat pomocí CNN, o jakou hodnotu číslice se jedná.

Obrázek 30 – MNIST

Zdroj: https://www.kaggle.com/oddrationale/mnist-in-csv

V ukázce pro rozpoznávání obrázku je použit již předtrénovaný model MobilNet. Různé verze modelu MobileNet[25] jsou k dispozici v repositáři na serveru GitHub. Modely si lze vybrat podle jejich složitosti a přesnosti. Nejsložitější model obsahuje 569 miliónů MAC[26] operací s TOP-1 přesností 70,9 % a nejjednodušší model s 14 MAC operacemi a s TOP-1 přesností 39,5 %. Při výběru modelu je tedy nutné zohlednit, v jakém prostředí se modely využijí a zda v těchto prostředích nebude výpočetní výkon brzdit komplexnější modely. Pro klasifikaci obrázků byl zvolen model MobileNet, který by měl být kompromisem mezi dostupným výkonem webové aplikace a přesností.

.....

[24] Multi-Column Deep Neural Networks.

[25] Modely jsou dostupné na adrese: https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md

[26] MAC (Multiplication and Addition operations) udává kolikrát se v modelu provádí operace násobení a sčítaní.

5.3 Návrh uživatelského prostředí

Framework Angular umožňuje integraci material design pomocí Angular Material. Material design je sada postupů a doporučení společnosti Google pro co nejpříjemnější uživatelskou zkušenost neboli UX (user experience).

Material design společnosti Google se stal velice oblíbeným, protože společnost Google ho začala aplikovat ve svých populárních aplikacích (Gmail, Youtube, GoogleDrive). Uživatelé si na čistý styl uživatelského prostředí zvykli a stal se defacto standardem pro UX vývojáře. V obsáhlé příručce se nachází vizuální definice pro jednotlivé elementy, jejich velikost, stíny, animace a barvy. Příručka se stala standardem pro vizuální část aplikace pro velkou část vývojářů.

5.4 Implementace aplikace

Základním kamenem webové aplikace je framework Angular, založený na technologii SPA (Single page application). Webové aplikace typu SPA celou webovou aplikaci nahrají již během prvního vstupu do aplikace, přesměrování funguje na straně klienta pomocí JavaScriptu a nedochází k dotazování serveru a znovunačítaní celých stránek. Komunikace se serverem probíhá na pozadí pomocí AJAX nebo WebSocket technologií. SPA aplikace reagují prakticky okamžitě na jakoukoliv akci uživatele, v případě, že akce vyžaduje dodatečná data, se kontaktuje server na pozadí a data se po chvíli objeví na stránce bez nutnosti opět celou stránku načíst, aplikace působí velmi responzivně.

Angular je component-oriented, to znamená, že během vývoje webové aplikace se snažíme identifikovat samostatné logické bloky-funkcionality, které nazýváme komponentami. Komponenty poté programujeme samostatně jako jeden stavební blok, který lze použít znovu v jiných částech aplikace. Stavební bloky celé aplikace se umísťují do modulu. Modul reprezentuje definici webové aplikace a obsahuje seznam jednotlivých komponent, včetně pomocné funkcionality, která je definována nejčastěji pomocí directives, pipes nebo services.

5.4.1 Component

Component lze v programátorském světě chápat jako view. View je definice, jak zobrazovat výstup aplikace uživateli. Zároveň view umožňuje uživateli interakci s webovou aplikací pomocí ovládacích prvků. Komponenta také obsahuje definici obslužné logiky ve formě JavaScriptové třídy.

5.4.2 Directive

Directive se používá, pokud chceme někde rozšířit možnosti jazyka HTML. Directive jsou funkce, které Angular spustí, pokud je najde v DOM. Directive můžeme chápat jako nadstavbu nad HTML, pomocí directive si můžeme definovat nový atribut pro vybraný HTML element, díky kterému poté rozšíříme funkcionalitu daného elementu.

5.4.3 Pipe

Pokud potřebujeme formátovat jakýkoliv výstup uživateli (měna, datum), tak si lze definovat třídu, která nám výstup naformátuje. Na pipe se lze dívat jako na mikro komponentu, která je zodpovědná za formátování výstupu uživateli.

5.4.4 Service

V případě, že potřebujeme mít nějaký sdílený zdroj dat pro webovou aplikaci, který využijeme ve více komponentách, tak si můžeme vytvořit objekt typu service. Objekt můžeme poslat do konstruktorů komponent a pracovat s ním uvnitř komponenty. Častým využitím service objektu je funkcionalita, která načítá data ze serveru pomocí API.

5.4.5 Module

Modul lze chápat jako kontejner, který zapouzdří všechny definované stavební bloky aplikace. Moduly by měly reprezentovat aplikační doménu a její workflow. Malé aplikace většinou obsahují pouze jeden modul, stejně jako v aplikaci s příklady TensorFlow.

5.5 Implementace lineární regrese

Úlohu lineární regrese můžeme považovat za příklad typu HelloWorld, často používaného v programátorském světě. Jeden z nejtriviálnějších příkladů na vyřešení může být nalezení hodnot pomocí lineární regrese definované vztahem:

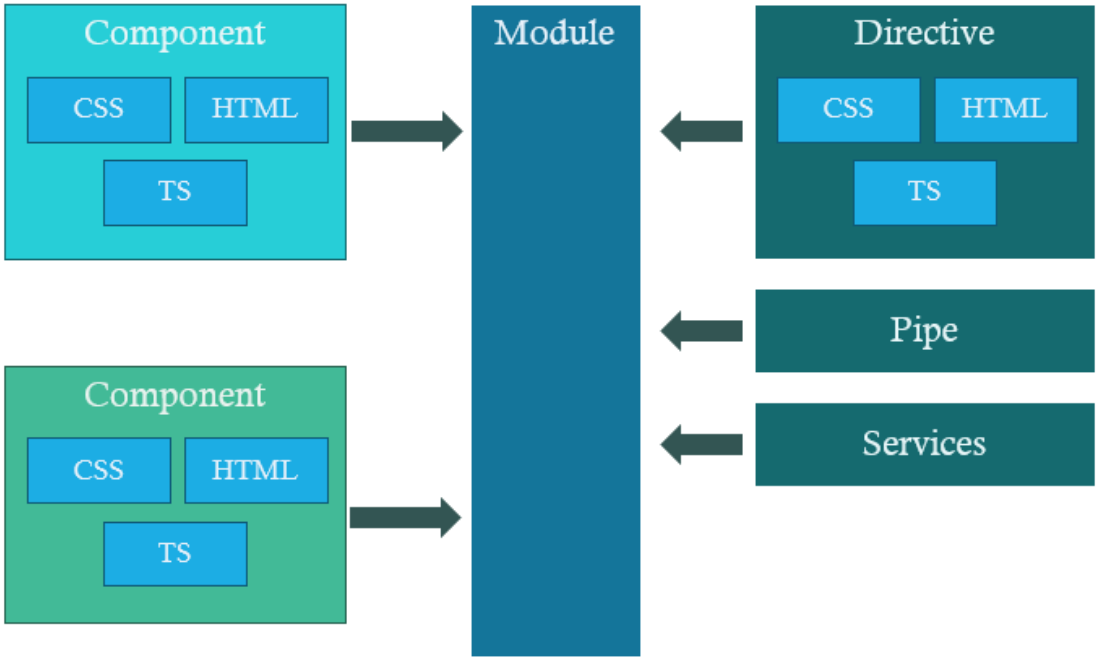
$$Y = mx + b$$

Webová aplikace bude obsahovat plátno (canvas), které uživatel může použít pro generování trénovacích dat. Trénovací data lze také vygenerovat automaticky, dle vstupních parametrů uživatelského rozhraní. Poté se aplikace pokusí nasimulovat parametry **m** a **b** s využitím Tensorflow.js proměnných a konfigurace modelu.

Po natrénování modelu dojde k vizualizaci výsledku pomocí grafu, ve kterém uživatel uvidí vykreslenou přímku mezi tréninkovými daty, která jsou zobrazena formou

bodů. Na stránce s úlohou lineární regrese lze odhadnout parametry nebo natrénovat neuronovou síť.

Obrázek 31 – Architektura frameworku Angular



Zdroj: autor

5.5.1 Implementace

Nejdříve začneme vytvořením komponenty v projektu pomocí ***ng generate component linearRegressionExample***. Příkaz vygeneruje prázdné soubory pro konfiguraci komponenty.

V souboru HTML ve složce s názvem Komponenty definujeme vzhled komponenty pomocí HTML elementů, včetně definice plátna, pomocí kterého může uživatel vkládat trénovací data. Rozměr plátna je definován ve zdrojovém kódu komponenty *linearRegressionExample.component.ts* pomocí proměnné **canvasSize**, která je přednastavená na velikost 320. Uživatel tedy může definovat v plátně trénovací hodnoty v intervalu <0–320> pro X a Y. Dále jsou v komponentě definované dvě proměnné, které reprezentují parametry **m** a **b** modelu pomocí **variable** funkce s náhodnou hodnotou počátečního čísla. Hodnoty z canvasu bylo nutné normalizovat na menší hodnoty, protože při použití hodnot v intervalu velikosti plátna měl JavaScript

problém s počítáním větších hodnot a často výsledek učení byl NaN[27]. Proto je nastavená normalizace hodnot zadaných do plátna tak, aby byly v rozmezí <0–3,2>.

Definici neuronové sítě je možné vidět níže na obrázku č. 32.

Obrázek 32 – Trénování neuronové sítě

```
async train() {
  // Definice modelu
  this.linearModel = tf.sequential();
  this.linearModel.add(tf.layers.dense({
    units: 1,
    inputShape: [1],
    useBias: true
  }));

  // Definice ztratove a optimalizacni funkce
  this.linearModel.compile({
    loss: 'meanSquaredError',
    optimizer: 'sgd',
    metrics: ['mse']
  });

  // Trenovaci data
  const learningInput = tf.tensor1d(this.xVals);
  const learningOutput = tf.tensor1d(this.yVals);

  // Uceni modelu
  await this.linearModel.fit(learningInput, learningOutput, {
    epochs: this.epochs,
    callbacks: {
      onTrainBegin: async (logs) => {
        this.scriptPerformance.start = performance.now();
        this.pbStatus = LearningStatus.PROGRESS;
      },
      onEpochBegin: async (epoch, logs) => {
        this.pbValue = Number((((epoch + 1)  / this.epochs) * 100).toFixed(0));
      },
      onTrainEnd:  async (logs) => {
        this.pbStatus = LearningStatus.DONE;
        this.scriptPerformance.end = performance.now();
        this.scriptPerformance.timeElapsed =
        (this.scriptPerformance.end -  this.scriptPerformance.start).toFixed(2);
      }
    }
  });
```

Zdroj: autor

Komponenta obsahuje funkci **train**, ve které je definována optimalizační funkce **sgd** (SGDOptimizer), používající stochastický gradient descent s hodnotou pro

.....

[27] NaN znamená Not a number, NaN je speciálním typem JavaScriptu

rychlost učení (learning rate) **0.1**. Optimalizační funkce se pokusí vypočítat současnou hodnotu pro Y s parametry **b** a **m** a poté se pokusí při další iteraci minimalizovat chybu pomocí změn hodnot u těchto parametrů. Počet iterací je definován pomocí proměnné **epochs**, která je přednastavená na hodnotu 100. To znamená, že optimalizační funkce spočítá chybu a upraví parametry 100×. Na konci procesu trénování, po všech iteracích, se uživateli vykreslí graf pomocí Charts.js reprezentující přímku s hodnotou parametrů **b** a **m**.

V tabulce níže jsou trénovací data, která byla použita pro porovnání metody odhadu parametrů a vytvoření modelu pomocí neuronové sítě. Dataset obsahuje 14 trénovacích hodnot se vstupem X a výstupem Y.

Tabulka 9 – Tabulka s trénovacími daty

X	0.21	0.98	0.73	2.02	1.45	1.85	1.94	2.13	2.78	2.62	2.89	2.98	0.91	0.73
Y	0.29	0.8	1.51	0.87	1.66	1.44	1.91	2.46	2.2	2.76	2.7	3.08	0.48	0.86

Tabulka 10 – Průběh ztrátové funkce

Loss funkce	Iterace 10	Iterace 100	Iterace 1000
0.253237	0.1893392	0.185688	0.185684

V tabulce č. 12 si lze všimnout, že při zvýšení počtu iterací předpis přestává být variabilní a ustaluje se na jedné hodnotě.

Tabulka 11 – Definice přímky dle počtu iterací

Pokus	Iterace 10	Iterace 100	Iterace 1000
1	Y = 0.7214X + 0.4463	Y = 0.8590X + 0.1607	Y = 0.8658X + 0.1464
2	Y = 0.7447X + 0.3979	Y = 0.8668X + 0.1443	Y = 0.8658X + 0.1464
3	Y = 0.6882X + 0.5153	Y = 0.8585X + 0.1617	Y = 0.8658X + 0.1464

Tabulka č. 13 ukazuje čas učení, který trval při odhadování parametrů pomocí TensorFlow proměnných dle počtu iterací, a čas učení neuronové sítě.

Tabulka 12 – Čas učení[28]

Počet iterací / epochs	Odhad parametrů	Neuronová síť
10	64 ms	268 ms
100	349 ms	1773 ms
1000	3613 ms	16881 ms
10 000	147 542 ms	122 536 ms

Tabulka 13 – Porovnání výsledků odhadu parametrů a neuronové sítě po 100 iteracích

	Využití parametrů		Neuronová síť	
Y	**Výsledek**	**Chyba**	**Výsledek**	**Chyba**
0.29	0.3282371	*0.0382371*	0.2571	*0.0329*
0.8	0.9949197	*0.1949197*	0.9531	*0.1531*
1.51	0.7784643	*0.7315357*	0.7271	*0.7829*
0.87	1.8953739	*1.0253739*	1.8932	*1.0232*
1.66	1.4018557	*0.2581443*	1.378	*0.282*
1.44	1.7481843	*0.3081843*	1.7395	*0.2995*
1.91	1.8261082	*0.0838918*	1.8209	*0.0891*
2.46	1.9906144	*0.4693856*	1.9926	*0.4674*
2.2	2.5533984	*0.3533984*	2.5802	*0.3802*
2.76	2.4148667	*0.3451333*	2.4356	*0.3244*
2.7	2.6486387	*0.0513613*	2.6796	*0.0204*
3.08	2.7265625	*0.3534375*	2.761	*0.319*
0.48	0.9343122	*0.4543122*	0.8898	*0.4098*
0.86	0.7784643	*0.0815357*	0.7271	*0.1329*

[28] Výsledný čas je průměrem získaným ze tří pozorování.

5.6 Příklad s klasifikací čísel

V následujícím příkladu si ukážeme, jak lze pracovat s již předtrénovaným modelem. Trénování modelu je občas velice náročné na výpočetní prostředky CPU a GPU, proto je někdy vhodné pro již existující aplikaci použít předtrénovaný model. Pro příklad s klasifikací čísel použijeme připravený model z jazyka Python a zkonvertujeme jej do formátu JSON, který lze načíst pomocí Tensorflow.js.

Model očekává na vstupu čtyřdimenzionální pole (batch, výška, šířka, barevné kanály). Kanály ukazují, zda pracujeme s barevným, nebo černobílým obrázkem. Jelikož byl model trénovaný na datové sadě MNIST, která obsahuje černobílé ručně psané číslice, připravíme si aplikaci, tak aby její výstup byl rovněž černobílý. Díky tomu můžeme nastavit hodnotu pro kanály na konstantu 1; pokud bychom pracovali s barevnými obrázky, tak nastavíme 3 (RGB).

5.6.1 Konverze modelu

Předtrénovaný model z jazyka Python lze zkonvertovat jako KERAS model do formátu JSON pomocí Python CLI příkazem ***tensorflowjs_converter –input_format keras \ keras/nazevModelu.h \ src/assets.*** Před použitím příkazu je však zapotřebí mít nainstalovaný interpret jazyka Python a Python CLI, včetně pip[29], které je již součástí instalace jazyka Python verze 3.4 a vyšší, a také mít nainstalovaný balíček Tensorflow.js pro Python pomocí příkazu ***pip install tensorflowjs***. Zkonvertovaný model si ve webové aplikaci uložíme do složky Src/assets, která je přístupná pro webový server. Do této složky se běžně dávají zdroje, které webová aplikace potřebuje (obrázky, video, soubory ke stažení).

Model lze mít také uložený na internetu a nahrávat ho pomocí URL, tento typ použití si ukážeme v příštím příkladu s klasifikací obrázků.

5.6.2 Implementace

Pro příklad si ve webové aplikaci vytvoříme novou komponentu drawningExample pomocí příkazu ***ng generate component drawningExample***. Příkaz ng je součástí Angular CLI a vygeneruje nám ve složce s projektem novou složku drawningExample se soubory pro konfiguraci komponenty.

Dále je třeba definovat také plátno (canvas) pro kreslení číslic, které následně bude model zkoušet klasifikovat. Plátno je implementované jako directive, protože funkcionalitu plátna lze použít i v jiných částech aplikace. Funkcionalita plátna bude

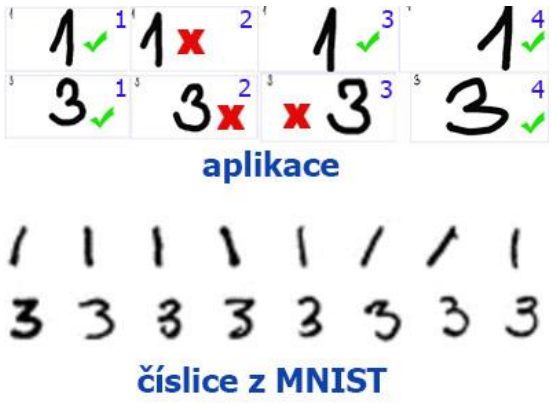
……

[29] Python pip je správce balíčku pro jazyk Python.

definovaná pomocí directive, vytvoříme proto soubor drawable.directive.ts ve složce Src/app. Directive obsahuje kód, který zajistí zachycení tahů uživatele ve formě obrázku. Zaznamenaná data jsou poté zaslána komponentě, ve které je vyhodnotí model. Důležitý pro implementaci je správný poměr stran plátna a správná tloušťka štětce; v případě, že se změní poměry stran plátna nebo tloušťka štětce, dojde k ovlivnění klasifikace obrázku. Model poté může vyhodnocovat číslice chybně.

Obslužný kód pro komponentu drawningExample obsahuje inicializační funkci, ve které nahráváme zkonvertovaný model ve formátu JSON ze složky Assets. Funkce pro nahrání modelu je asynchronní, protože v ní chceme použít klíčové slovo await. Komponenta obsahuje funkci ***predictNumber***, do které vstupuje nakreslený obrázek z plátna. Funkce je zabalená do asynchronní funkce ***tidy***, která spravuje paměť grafické karty a měla by zabránit únikům paměti (*memory leak*). Obrázek je transformován na správnou velikost a poté převeden na tensor. Pro práci s obrázky se v knihovně Tensorflow.js nachází pomocné API. Obrázek lze převést na tensor pomocí funkce ***fromPixels***, poté je třeba transformovat tensor na správnou velikost, k tomu použijeme funkci ***reshape***, do níž předáme tensor s obrázkem, a druhý parametr je tvar tensoru, do kterého chceme transformovat. V našem případě je to tvar [1, 28, 28, 1]. Pro transformation je nutné převést typ hodnoty tensoru na float32 pomocí funkce ***cast***. Nyní je již tensor připravený a lze ho vyhodnotit pomocí modelu. Model vrací pole s deseti prvky (0–9), každý prvek reprezentuje šanci, že vstupní obrázek byl číslice rovná indexu pole. Predikce využívá funkce známé jako softmax[30].

Obrázek 33 – Ukázka úspěšnosti klasifikace číslic



Zdroj: autor

[30] Softmax funkce vrací vektor s distribucí pravděpodobností možných výsledných hodnot. Součet pravděpodobností je roven 1. Softmax se používá během klasifikačních úloh.

Pro lepší interpretaci výsledku vytvoříme v příkladu ještě komponentu ***drawningExampleChart***, která bude zodpovědná za vykreslení vertikálního grafu čísel, reprezentující distribuci pravděpodobnosti pro nakreslené číslo.

Obrázek č. 33 ukazuje výsledek klasifikace uživatelského vstupu, pro porovnání jsou přiložena i trénovací data z databáze MNIST.

5.7 Příklad s klasifikací obrázku

Dalším příkladem použití knihovny v Tensorflow.js ve webovém prohlížeči je úloha klasifikace obrázků. Úloha je velice oblíbená a lze ji aplikovat do velké většiny webových nebo mobilních aplikací, ve kterých se pracuje s obrázky. V úloze je integrován předtrénovaný model MobileNet, který je optimalizován pro použití ve webových aplikacích. Lze ho využít také během implementace mobilních aplikací.

Klasifikaci pomocí modelu MobileNet lze reálně využít pro třídění nahrávaných obrázků a videí na server. Model lze aplikovat také pro třídění fotografií v mobilním telefonu, díky zabudované klasifikaci v aplikaci lze pomoci uživateli s tříděním, indexací a hledáním mediálních souborů (obrázky, fotografie, videa).

V příkladu je možné nahrát pomocí uživatelského rozhraní obrázek, který poté projde transformací a normalizací. Bez transformace a normalizace by model nebyl schopný obrázek dobře rozeznat, model MobileNet je natrénovaný na pevné velikosti obrázků [50]. Po klasifikaci je uživateli zobrazeno pravděpodobnostní rozdělení klasifikace obrázku do kategorií pomocí komponenty horizontálního grafu.

5.7.1 Implementace

Komponenta pro klasifikaci obrázků se jmenuje *imageClassification* a vygenerujeme ji stejným způsobem jako u předchozí úlohy pomocí Angular CLI a příkazu *ng generate*. Součástí implementace je také vizuální zobrazení pro pravděpodobnostní rozdělení klasifikace obrázku pomocí NPM balíčku **ng2-charts**[31].

Komponenta ng-2charts je pouze obalem pro populární knihovnu Charts.js pro framework Angular. Instalaci balíčku ng2-charts provedeme pomocí příkazů:

npm install ng-charts –save
npm install chart.js --save

[31] Ng-2charts je dostupné na adrese https://www.npmjs.com/package/ng2-charts

NPM poté nainstaluje závislosti do projektu a uloží je do souboru *project.json*. Dále je nutné nainstalovaný balíček ng2-charts registrovat do souboru *app.module*, který obsahuje závislosti webové aplikace.

V komponentě *imageClassification* nahrajeme model MobileNet pro predikci, tentokrát z webového uložiště[32] společnosti Google. K nahrání modelu použijeme funkci **loadLayersModel**, parametrem funkce je URL modelu. Funkce loadLayersModel je asynchronní, nechceme totiž, aby nahrávání blokovalo interakci s webovou aplikací. Do komponenty je nutné nahrát soubor s kategoriemi objektů, které dokáže model MobileNet rozeznat. Seznam těchto kategorií uložíme do souboru **imageNet_classes** do složky s komponentou. Soubor obsahuje 1000 kategorií, které dokáže model rozeznat, s anglickým popisem.

Počet kategorií, které zobrazíme v grafu distribuce pravděpodobnosti, se ovlivňuje pomocí konstanty *TOPK_PREDICTIONS*, která je přednastavena na hodnotu 10. Po nahrání obrázku dojde v metodě **loadImage** k jeho zpracování, v metodě se vytvoří nový HTML image element, který je vložen do stránky jako náhled pro nahraný obrázek. Pomocí API prohlížeče načteme soubor funkcí **readAsDataURL**, abychom mohli pro nově vzniklý HTML image element definovat jeho zdroj pomocí *src* atributu. Nově vytvořený element s definovaným atributem src slouží jako vstupní parametr do metody **predict**, která využívá pomocné funkce z API knihovny Tensorflow.js **fromPixels**, funkce vrátí tensor popisující obrázek. Obrázek se poté normalizuje a převádí na správnou velikost pomocí tensor funkce **reshape**. Připravený tensor poté poskytneme jako vstupní parametr do funkce predict modelu MobileNet. Výsledkem je poté seznam všech kategorií s pravděpodobnostním rozdělením. Poté se již seznam seřadí podle nejvyšší pravděpodobnosti v metodě **getTopCategories** a vrátí počet nejvyšších kategorií závislý na konstantě *TOPK_PREDICTIONS*. Zpracované kategorie z metody jsou vstupním parametrem pro komponentu *imageClassificationChart***,** která je zodpovědná za vykreslení uživatelského výstupu.

[32] Model lze stáhnout na této URL: https://storage.googleapis.com/tfjs-models/tfjs/mobilenet_v1_0.25_224/model.json

Obrázek 34 – Obrázky určené ke klasifikaci

Na implementované úloze provedeme měření, jak se klasifikace jednotlivých obrázků změní, pokud na obrázky aplikujeme nějaký filtr. Filtry, které použijeme, jsou grayscale a blur. Obrázek na grayscale převedeme pomocí metody BT-709 (HDTV), metoda je také známa jako ITU-R BT-709. Blur provedeme pomocí metody Gausian s parametrem kernel size nastaveným na hodnotu 10.

Obrázek 35 – Greyscale obrázky určené ke klasifikaci

Tabulka 14 – Klasifikace sportovního auta

Kategorie	Originál obrázek		Grayscale obrázek	
1	sports car	**21,1 %**	station waggon	**20, 64 %**
2	station waggon	**13,29 %**	minivan	**15, 85 %**
3	race car	**13 %**	race car	**13, 83 %**

Tabulka 15 – Klasifikace kočky

Kategorie	Originál obrázek		Grayscale obrázek	
1	lynx, catamount	**21,1 %**	lynx, catamount	**73,77 %**
2	tiger cat	**13,29 %**	Egyptian cat	**18,12 %**
3	tabby cat	**13 %**	tabby cat	**5,26 %**

Tabulka 16 – Klasifikace hrníčku

Kategorie	Originál obrázek		Grayscale obrázek	
1	cup	**46,76 %**	cup	**42,52 %**
2	coffie mug	**23,03 %**	coffie mug	**32,01 %**
3	mixing bowl	**9,04 %**	caldron	**2,97 %**

Tabulka 17 – Klasifikace po použití blur efektu

	Originál obrázek		Blur efekt	
Kategorie				
1	golden retriever	**76,66 %**	golden retriever	**82,72 %**
2	cocker spaniel	**13,98 %**	cocker spaniel	**9,12 %**
3	clumber	**5,43 %**	clumber	**3,67 %**

Je zajímavé, že po aplikaci blur efektu došlo k přesnější klasifikaci.

5.8 Vyhodnocení a návrhy na zlepšení

Knihovna Tensorflow.js se vyvíjí velice rychle, během implementace aplikace došlo k vydání verze Tensorflow.js 1.0.0 z verze 0.15.3. Poslední verze knihovny kromě znatelného zrychlení v sobě obsahovala také rozsáhlé změny v API[33], které způsobily po aktualizaci knihovny v projektu s webovou aplikací znefunkčnění některé funkcionality.

Po aktualizaci bylo nutné projít zdrojový kód knihovny a ručně nalézt, kam se původní funkcionalita přemístila či jak se přejmenovala. Autoři knihovny však několik dní po uvedení nové verze knihovny aktualizovali i dokumentaci, bohužel však v dokumentaci chyběly informace o původních umístěních funkcí a jejich jmen v knihovně. Knihovna je poměrně mladá a neustále se vyvíjí, proto je nutné se na podobné scénáře připravit a počítat s možností, že se vyvíjený projekt po aktualizaci knihovny znefunkční. Nicméně by v budoucnosti nemělo již docházet k výraznějším změnám v API, jelikož autoři vydáním verze 1.0.0 naznačili, že by k větším změnám již nemělo docházet.

Během výběru předtrénovaného modelu je nutné zvážit poměr cena/výkon. Při výběru přesnějšího modelu se webová aplikace stává méně responzivní, protože predikce hodnoty bývá náročnější. Výběrem složitějšího modelu můžeme získat přesnější výsledek, ale výpočet může trvat o několik vteřin déle a to pro některé webové aplikace není žádoucí. Zároveň je třeba si uvědomit, že pokud se v aplikaci používá předtrénovaný model, tak bychom si měli ověřit, na jakých datech byl trénovaný. Na implementované úloze s klasifikací čísla si můžeme ověřit, že model poměrně často špatně klasifikuje nakreslené číslo.

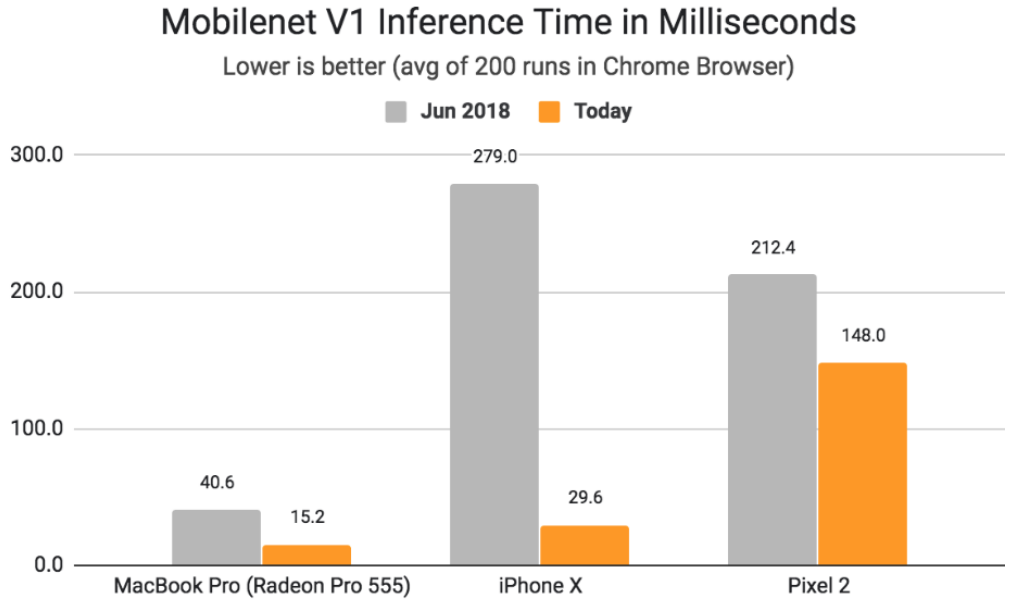
Model MobileNet_v1_0.25_224_quant určený pro prohlížeče už sám má poměrně nízkou šanci (48 %) klasifikovat správně číslici. Byl trénovaný na MNIST databázi číslic, a zvlášť pokud bude uživatel mít nestandardní písmo, je velice pravděpodobné, že model bude mít velké problémy s klasifikací.

Webovou aplikaci s klasifikací čísla a obrázku by bylo vhodné rozšířit o trénovací část, díky které by šlo vytvořit a natrénovat vlastní model. Před zmíněným rozšířením je však nutné zajistit trénovací vzorek dat. Databáze MNIST obsahuje 70 tisíc obrázků, podobnou sbírku dat je velice těžké sehnat.

[33] Rozsáhlé změny v API se často nazývají breaking changes.

Dalším možným rozšířením aplikace by bylo implementovat možnost vybírat si pro predikci hodnoty z více modelů, ať už z vlastních, nebo natrénovaných, a ke klasifikaci obrázků přidat i klasifikaci videa, která se implementuje velice podobně.

Obrázek 36 – Optimalizace Tensorflow.js po poslední aktualizaci



Zdroj: https://github.com/tensorflow/tfjs/releases

Využití frameworku Angular během implementace poskytlo možnost využít jazyk TypeScript místo JavaScriptu. TypeScript umožnil v projektu využít typovou kontrolu včetně usnadnění některých složitějších konstruktů jazyka JavaScript (importy, asynchronní funkce, problematika scopes). Knihovna Tensorflow.js obsahuje navíc definiční soubory pro jazyk TypeScript, který využívá intellisense[34] editoru zdrojového kódu, aby programátorovi radil a upozorňoval v případě nekompatibilních datových typů. TypeScript umožnil příjemnější vývoj webové aplikace.

[34] Intellisense můžeme chápat jako inteligentní doplňování kódu, urychluje vývoj webové aplikace díky okamžité kontrole syntaxe. Intellisense můžeme chápat jako prvek IDE, který snižuje počet chyb programátora.

Závěr

Cílem diplomové práce bylo popsání možnosti využití machine learningu ve webových aplikacích s pomocí jazyka JavaScript včetně implementace tří různých úloh s využitím JavaScriptové knihovny Tensorflow.js a frameworku Angular.

První příklad ukazuje, jak lze využít knihovnu Tensorflow.js k získání parametrů *m* a *b* pro lineární regresi. Příklad také obsahuje srovnání výsledků při využití neuronové sítě a modelu lineární regrese na stejných trénovacích datech. Neuronové sítě dokáží aproximovat jakoukoliv funkci a příklad s lineární regresi to ukazuje.

Ve druhém příkladu je využit již předtrénovaný model s MNIST databází pro rozpoznávání ručně psaných číslic. Na příkladu s rozpoznáváním čísel je možné si všimnout, že v případě využití modelu, který je optimalizován pro web, klesá přesnost klasifikace, a proto se stane, že klasifikace číslice je ve webové aplikaci chybná.

Třetím implementovaným příkladem je klasifikace obrázků pomocí předtrénovaného modelu se schopností klasifikovat obrázky do 1000 kategorií. Příklad s klasifikací obrázků může najít uplatnění v mnoha aplikacích, které pracují s mediálním vstupem uživatele. Příkladem se lze inspirovat během nahrávání obrázků uživatele na server. Nahrávané obrázky budou rovnou popsány kategoriemi, do kterých byly pomocí modelu kategorizovány, a tyto kategorie se uloží jako dodatečné informace na serveru s obrázkem.

Během implementace praktické části došlo k publikování nové verze knihovny Tensorflow.js a během přechodu na novější restrukturalizované API došlo k znefunkčnění webové aplikace. Vývoj knihovny Tensorflow.js je velice rychlý a proto je dobré počítat s nutnými aktualizacemi pro aplikace v případě vydání nové verze knihovny. Autoři knihovny naštěstí s vydáním nové verze vydají i aktualizovanou dokumentaci, kde si lze změnit vyhledat a ulehčit tím přechod na novější verzi.

Knihovně Tensorflow.js náleží v práci větší prostor z důvodu rostoucí obliby od roku 2018. Práce obsahuje popis knihovny včetně nejdůležitějších částí, které je před použitím knihovny nutné pochopit. Pomocí knihovny Tensorflow.js lze vytvářet modely včetně jejich aplikace přímo v prohlížeči a díky tomu umožnit webovým vývojářům jejich využití ve svých aplikacích. K dispozici jsou již předtrénované modely, například pro klasifikaci obrázků, které fungují poměrně spolehlivě.

V budoucnosti bude k dispozici čím dál tím větší počet těchto modelů a jejich využití v reálných aplikacích bude naprosto běžné. Třídění dokumentů, obrázků, videa a zvukových záznamů bude za pár let běžnou funkcionalitou malých aplikací, a ne pouze softwarových gigantů, jako jsou Google, Amazon nebo Microsoft. Je však nutné

při využívání jazyka JavaScript sledovat poměr cena/výkon, protože jazyk JavaScript není primárně určený pro machine learning, ačkoli může posloužit jako vstupní brána do oboru machine learningu pro webové vývojáře. Pro trénování a spouštění složitých modelů bude vždy lepší volbou jazyk Python, který je v oblasti machine learningu králem.

Machine learning zažívá posledních několik let obrovský rozvoj a je velice obtížné se zorientovat ve všech dostupných technologií. Diplomová práce může pomoci vývojářům využívající jazyk JavaScript získat přehled v dostupných technologií a začít využívat výhody machine learningu ve svých aplikací. Na práci lze navázat rozšířením příkladů implementovaných v praktické části.

Terminologický slovník

Termín	Zkratka	Význam
Application programming interface	*API*	Rozhraní pro programování aplikací. Sbírka funkcí, tříd a protokolů
Immediately Invoked Function Expression	*IIFE*	Způsob spuštění funkce v JavaScriptu
Convolutional neural network	*CNN*	Typ neuronové sítě určený převážně pro klasifikaci obrázků
Long short-term memory	*LSTN*	Typ neuronových sítí s možností pamatovat si předchozí stavy
Modified National of Standards and Technology	*MNIST*	Velká databáze ručně psaných číslic, určená převážně pro trénování modelů
Front-end	*FE*	Klientská část aplikace
Back-end	*BE*	Serverová část aplikace
Node package manager	*NPM*	Program pro instalování a správu Javascriptových balíčků
European Computer Manufactures Association Script	*ECMAScript*	Společnost vyvíjející standardy pro programovací jazyky
Canvas		HTML element určený pro kreslení grafiky na webové stránce
Callback		Spustitelný kód, který je předán argumentem
Framework		Sada nástrojů, knihoven a metodik pro vývoj aplikace.
WebGL		JavaScript API, pomocí kterého lze rendrovat 2D a 3D grafiku s využitím grafického adaptéru.

Seznam literatury

[1] **Cross, Amy.** Data Mining vs. Machine Learning: What's the Difference? *NG Data.* [Online] červen 2018. Dostupné z: https://www.ngdata.com/data-mining-vs-machine-learning/.

[2] **Doshi, Shweta. Quora.** ***How is machine learning different from data mining***. [Online] 2017. Dostupné z: https://www.quora.com/How-is-machine-learning-different-from-data-mining.

[3] **Team, Expert System. What is Machine Learning? A definition**. *Expert System.* [Online] Dostupné z: https://www.expertsystem.com/machine-learning-definition/.

[4] **Shaw, Reena. Top 10 Machine Learning Algorithms for Data Science Beginners**. *Data Quest.* [Online] květen 2018. Dostupné z: https://www.dataquest.io/blog/top-10-machine-learning-algorithms-for-beginners/.

[5] **Donges, Niklas. Gradient Descent in a Nutshell**. *Towards Data Science.* [Online] březen 2018. Dostupné z: https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0.

[6] **McDonald, Conor. Machine learning fundamentals (I): Cost functions and gradient descent**. *Towards Data Science.* [Online] listopad 2017. Dostupné z: https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220.

[7] **Glossary Machine Learning.** ***ML5.js.*** [Online] Dostupné z: https://ml5js.org/docs/glossary-machine-learning.

[8] Mittal, Akhil. Machine Learning Process And Scenarios. *eLearning Industry.* [Online] květen 2017. Dostupné z: https://elearningindustry.com/machine-learning-process-and-scenarios.

[9] What are the types of machine learning? *Towards Data Science.* [Online] prosinec 2018. Dostupné z: https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f.

[10] Ravanshad, Abolfazl. How to choose machine learning algorithms? *Medium.* [Online] duben 2018. Dostupné z: https://medium.com/@aravanshad/how-to-choose-machine-learning-algorithms-9a92a448e0df.

[11] Training Sets, Validation Sets, and Holdout Sets. ***DataRobot.*** [Online] Dostupné z: https://www.datarobot.com/wiki/training-validation-holdout/.

[12] Allibhai, Eijaz. Medium. ***Hold-out vs. Cross-validation in Machine Learning.*** [Online] říjen 2018. Dostupné z: https://medium.com/@eijaz/holdout-vs-cross-validation-in-machine-learning-7637112d3f8f.

[13] Reinforcement Learning Explained: Overview, Comparisons and Applications in Business. *Altexsoft.* [Online] leden 2019. Dostupné z: https://www.altexsoft.com/blog/datascience/reinforcement-learning-explained-overview-comparisons-and-applications-in-business/.

[14] What Are Neural Networks, Why They Are So Popular And What Problems Can Solve. *Steemit.* [Online] 2016. What Are Neural Networks, Why They Are So Popular And What Problems Can Solve. Dostupné z: https://steemit.com/academia/@krishtopa/what-are-neural-networks-why-they-are-so-popular-and-what-problems-can-solve.

[15] Bahmani, MJ. AI vs Machine Learning vs Deep Learning. *Medium.* [Online] listopad 2018. Dostupné z: https://medium.com/datadriveninvestor/ai-vs-machine-learning-vs-deep-learning-ba3b3c58c32.

[16] V, Avinash Sharma. Understanding Activation Functions in Neural Networks. *The Theory of Everything.* [Online] březen 2017. Dostupné z: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-network.

[17] What are Artificial Neural Networks ? *Towards Data Science.* [Online] květen 2017. Dostupné z: https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f.

[18] Why are deep neural networks hard to train? *Neural Networks and Deep Learning.* [Online] říjen 2018. Dostupné z: http://neuralnetworksanddeeplearning.com/chap5.html.

[19] Nielsen, Michael. How the backpropagation algorithm works. *Neural Networks and Deep Learning.* [Online] říjen 2018. Dostupné z: http://neuralnetworksanddeeplearning.com/chap2.html.

[20] A Beginner's Guide to Backpropagation in Neural Networks. *A.I. Wiki.* [Online] Dostupné z: https://skymind.ai/wiki/backpropagation.

[21] Feedforward Neural Network. *Wikipedia*. [Online] březen 2019. [Citace: 7. březen 2019.] Dostupné z: https://en.wikipedia.org/wiki/Feedforward_neural_network.

[22] Convolutional neural network. *Wikipedia.* [Online] Dostupné z: https://en.wikipedia.org/wiki/Convolutional_neural_network.

[23] Enterprise Deep Learning. *A Beginner's Guide to Convolutional Neural Networks (CNNs)*. [Online] 2018. [Citace:] Dostupné z: https://skymind.ai/wiki/convolutional-network.

[24] Olah, Christopher. *Understanding LSTM Networks.* 27. srpen 2015. Dostupné z: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[25] The MathWorks, Inc. What Is Deep Learning? *MathWorks.* [Online] Dostupné z: https://www.mathworks.com/discovery/deep-learning.html.

[26] McCarthy, John. What is artificial intelligence? [Online] 2007. Dostupné z: http://www-formal.stanford.edu/jmc/whatisai.pdf.

[27] Big data. *Wikipedia.* [Online] Dostupné z: https://en.wikipedia.org/wiki/Big_data.

[28] Kobielus, James. Measuring the Business Value of Big Data. *IBM Big Data Hub.* [Online] květen 2013. Dostupné z: https://www.ibmbigdatahub.com/blog/measuring-business-value-big-data.

[29] Tay, Liz. Inside eBay's 90PB data warehouse. *IT News.* [Online] květen 2013. Dostupné z: https://www.itnews.com.au/news/inside-ebay8217s-90pb-data-warehouse-342615.

[30] Walker, Jim. Big Data Defined – Part Deux: Value Definition. *Horton Works.* [Online] duben 2013. Dostupné z: https://hortonworks.com/blog/big-data-defined-part-deux-value-definition/.

[31] Bisht, Anurag Singh a Nayak, Swadhin Kumar. A study on Big Data analytics, approach and applications. *INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY.* [Online] 2019. Dostupné z: https://www.ijariit.com/manuscripts/v5i1/V5I1-1381.pdf. ISSN: 2454-132X.

[32] Elliott, Thomas. The State of the Octoverse: machine learning. *GitHub.* [Online] 2018. Dostupné z: https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/#programming-languages.

[33] Simpson, Kyle. *You Don't Know JS - Scope & Closures.* [editor] Simon St. Laurent a Brian Mac-Donald. 2. Sebastopol : O'Reilly Media, Inc., 2015. ISBN: 978-1-449-33558-8.

[34] Clanton, Josh. Understanding the Module Pattern in JavaScript. *A Drip of JavaScript.* [Online] 2015. Dostupné z: http://adripofjavascript.com/blog/drips/understanding-the-module-pattern-in-javascript.html.

[35] What to do when "this" loses context. *Free Code Camp.* [Online] červen 2018. Dostupné z: https://medium.freecodecamp.org/what-to-do-when-this-loses-context-f09664af076f.

[36] Antani, Ved. *Mastering JavaScript.* Birmingham : Packt Publishing Ltd., 2016. ISBN 978-1-78528-134-1.

[37] Copes, Flavio. The JavaScript Event Loop. *FlavioCopes.* [Online] duben 2018. Dostupné z: https://flaviocopes.com/javascript-event-loop/.

[38] Frame Timing API. *MDN web docs.* [Online] červen 2016. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Frame_Timing_API.

[39] Raveendran, Anoop. JavaScript Event Loop Explained. *Medium.* [Online] prosinec 2017. Dostupné z: https://medium.com/front-end-weekly/javascript-event-loop-explained-4cd26af121d4.

[40] Simpson, Kyle. *You Don't Know JS: ES6 & Beyond.* [editor] Simon St. Laurent and Brian MacDonald. 1. Sebastopol : O'Reilly Media, Inc., 2016. ISBN 978-1-491-90424-4.

[41] Shiffman, Daniel. Introduction to TensorFlow.js - Intelligence and Learning. *YoutTube.* [Online] duben 2018. Dostupné z: https://www.youtube.com/watch?v=Qt3ZABW5lD0.

[42] TensorFlow.js. [Online] duben 2018. Dostupné z: https://www.tensorflow.org/js/guide.

[43] Keras: The Python Deep Learning library. *Keras.* [Online] Dostupné z: https://keras.io/.

[44] Tensors and operations. *TensorFlow.* [Online] Dostupné z: https://www.tensorflow.org/js/guide/tensors_operations.

[45] Models and layers. *TensorFlow.* [Online] [Citace:] Dostupné z: https://www.tensorflow.org/js/guide/models_and_layers.

[46] Alyafeai, Zaid. A Gentle Introduction to TensorFlow.js. *Medium.* [Online] duben 2018. Dostupné z: https://medium.com/tensorflow/a-gentle-introduction-to-tensorflow-js-dba2e5257702.

[47] Tensorflow API reference. *Tensorflow.* [Online] 2019. Dostupné z: https://js.tensorflow.org/api/1.0.0/.

[48] Wikipedia. *MNIST database.* [Online] [Citace:] Dostupné z: https://en.wikipedia.org/wiki/MNIST_database.

[49] Ciresan, Dan, Meier, Ueli a Schmidhuber, Jurgen. Multi-column Deep Neural Networks for Image Classification. *SUPSI Instory.* [Online] únor 2012. [Citace:] Dostupné z: http://repository.supsi.ch/5145/1/IDSIA-04-12.pdf.

[50] GitHub. *MobileNet_v1.* [Online] březen 2018. Dostupné z: https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md.

[51] Sepp Hochreiter, Jurgen Schmidhuber. LONG SHORT-TERM MEMORY. *Institute of Bioinformatics, Johannes Kepler University Linz.* [Online] 1997. Dostupné z: http://www.bioinf.jku.at/publications/older/2604.pdf.

[52] Harlalka, Rajat. Choosing the Right Machine Learning Algorithm. *Hacker Noon.* [Online] červen 2018. Dostupné z: https://hackernoon.com/choosing-the-right-machine-learning-algorithm-68126944ce1f.

[53] Fumo, David. Types of Machine Learning Algorithms You Should Know. *Towards Data Science.* [Online] červen 2017. Dostupné z: https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861.

[54] Heidenreich, Hunter. What are the types of machine learning? *Towards Data Science.* [Online] prosinec 2018. Dostupné z: https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f.

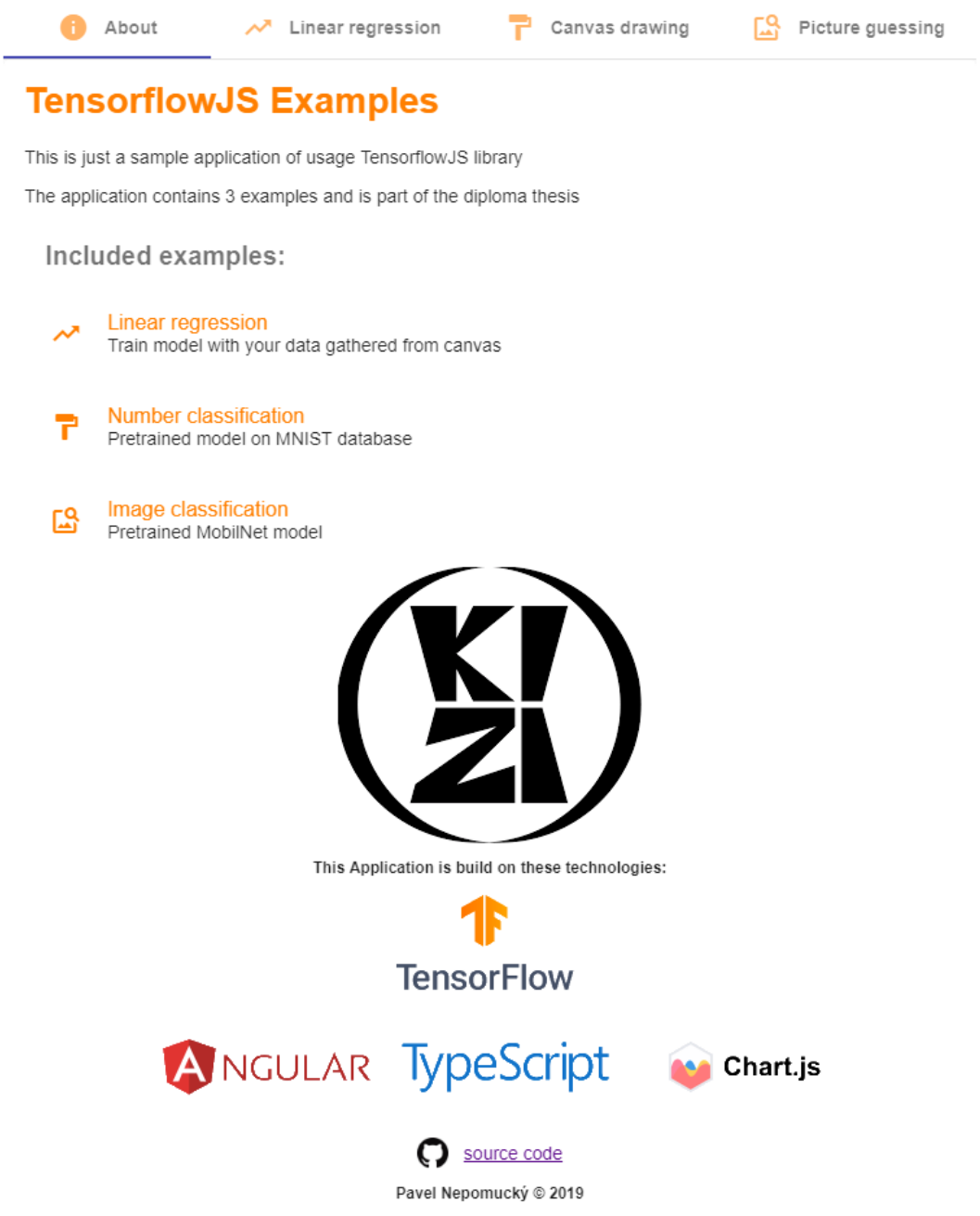
[55] Rokach, Lior a Maimon, O. *Data mining with decision trees: theory and applications.* World Scientific Pub Co Inc. , 2008. ISBN 978-9812771711.

Přílohy

Úvodní stránka aplikace

Stránka obsahuje informace základní informace o webové aplikaci včetně odkazu na repositář na server GitHub s veřejně dostupným zdrojovým kódem a seznam implementovaných příkladů pomocí knihovny Tensorflow.js. Aplikace se ovládá pomocí horizontálního menu.

Obrázek 37 - Úvodní stránka

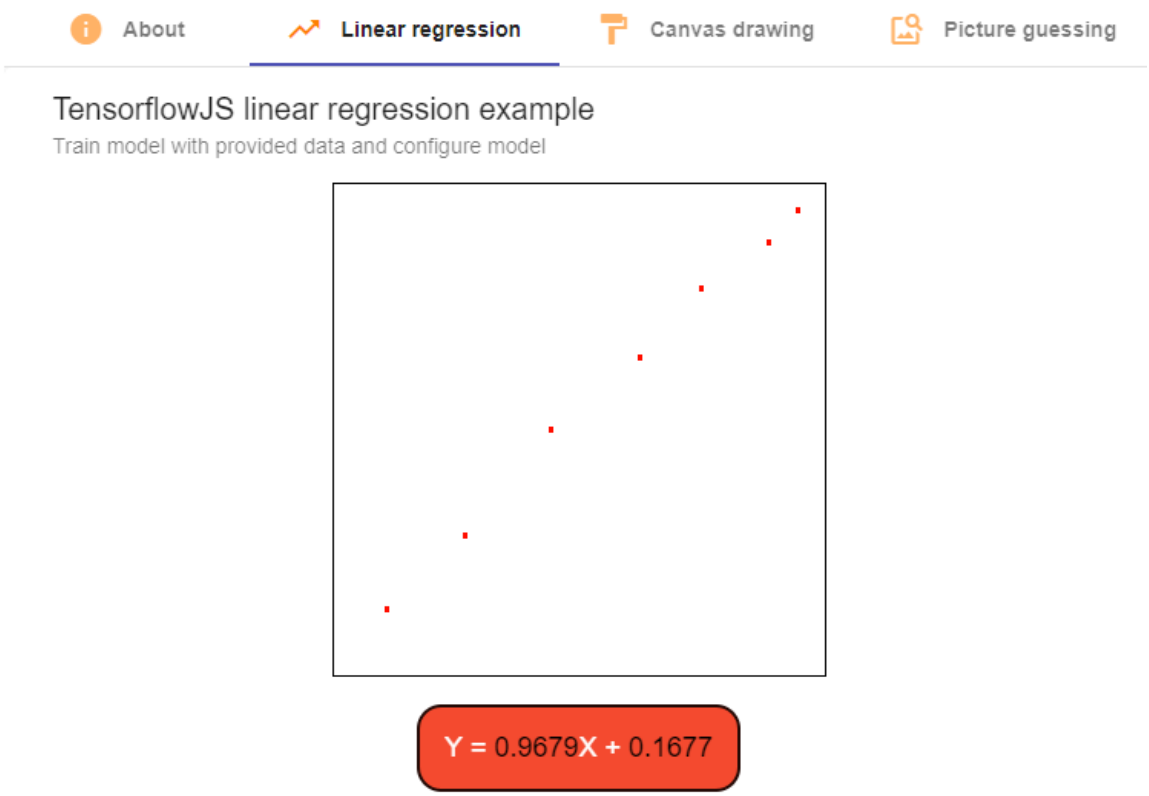


zdroj: autor

Lineární regrese

Uživatelské rozhraní s příkladem lineární regrese obsahuje více ovládací panelů. První panel obr. 38. obsahuje canvas, do kterého lze klikat myší a vytvářet trénovací data pro trénování. Pod canvasem se také nachází definice přímky s parametry **m** a **b** (y = mx + b), které jsou spočítány po kliknutí na tlačítko **Train model**. Canvas je definovaný velikostí *320 x 320* pixelů. Canvas lze zvětšit, ale poté je nutné upravit funkci, která normalizuje hodnoty.

Obrázek 38 - Canvas pro vkládání trénovacích dat



zdroj: autor

Pod canvasem se nachází shrnující panel (obr. 36), ve kterém jsou obsaženy všechny naklikané hodnoty z plátna. Hodnoty lze také náhodně vygenerovat pomocí tlačítka **Generate data**, počet generovaných dat se ovlivňuje pomocí pole *Number of values*. Trénování parametrů **m** a **b** pro přímku je zahájeno pomocí tlačítka **Train model**. Tlačítko **Clear data** smaže data určené pro trénování modelu.

Panel také obsahuje pole *Number of iteration* pro definici počtu iterací loss funkce. Průběh loss funkce lze pozorovat po rozkliknutí panelu **Loss function graph**, který obsahuje informace o velikosti chyby po každé iteraci. Graf funkce by měl být vždy klesající.

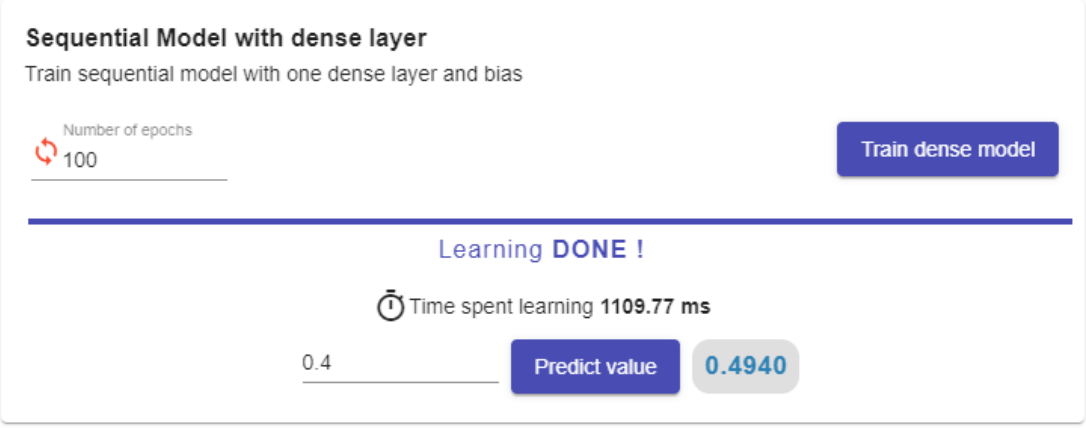
Obrázek 39 - Shrnující panel s grafem loss funkce

zdroj: autor

Další panel (obr. 39.) slouží k trénování neuronové sítě pomocí sequential modelu s definovanou jednou dense vrstvou. V panelu lze ovlivnit počet epochs, které lze označit jako počet iterací, stejně jako v minulém příkladu. Čím více iterací, tím přesnější model lze získat. Velký počet iterací, více než 1000, způsobí delší čas k získání výsledku. Větší hodnotu než 10 000 není doporučené nastavovat, kalkulace modelu trvá již delší dobu, až desítky minut, a získaná přesnost modelu se již prakticky nezmění.

Panel zobrazuje čas učení a po naučení modelu se v panelu zobrazí řádek s možností predikce hodnoty, dle naučeného modelu. Do pole stačí zadat vstupní hodnotu a model po stisknutí tlačítka **Preditct value** spočítá výslednou hodnotu.

Obrázek 40 - Panel pro trénování sequential modelu



zdroj: autor

Poslední panel (obr. 40.) obsahuje graf s body, které reprezentují trénovací data. Mezi body by měla být vykreslená přímka dle získaného modelu. Definice přímky se zobrazuje na obrázku č. 41. Přímka je vykreslená pomocí dvou bodů, pro *x = 0* a pro *x = max(množina trénovacích dat)*.

Obrázek 41 - Graf obsahující přímku a trénovací data

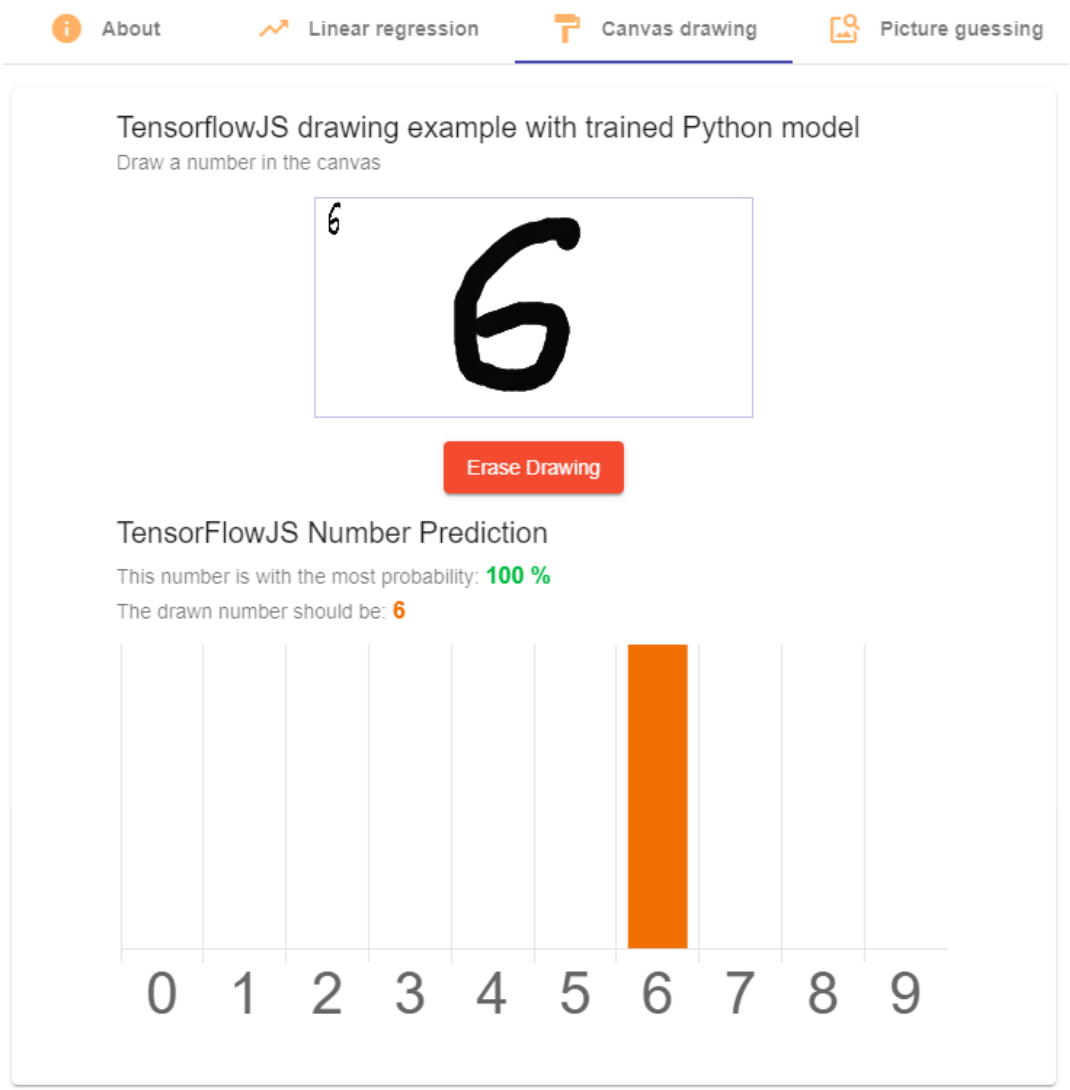
zdroj: autor

Klasifikace čísel

Dalším příkladem aplikace knihovny Tensorflow.js je stránka s klasifikací nakreslených čísel pomocí canvasu (obr. 42.). Na stránce se nachází canvas o velikost *300 x 150 pixelů*. Do kterého lze kreslit tahem myši, po uvolnění tlačítka myši dojde k získaní hodnot z canvasu a jejich transformaci. Je proto důležité číslice kreslit jedním tahem. Po transformaci se data předají jako vstup do modelu, model spočítá distribuci pravděpodobnosti pro každou hodnotu, která se poté zobrazí v grafu pod canvasem.

Model klasifikuje čísla velice striktně, často se stane, že je hodnota špatně klasifikována s pravděpodobností 100 %. Toto chování lze upravit použitím jiného modelu a úpravou velikosti plátna a transformace hodnot.

Obrázek 42 - Stránka s klasifikací čísel

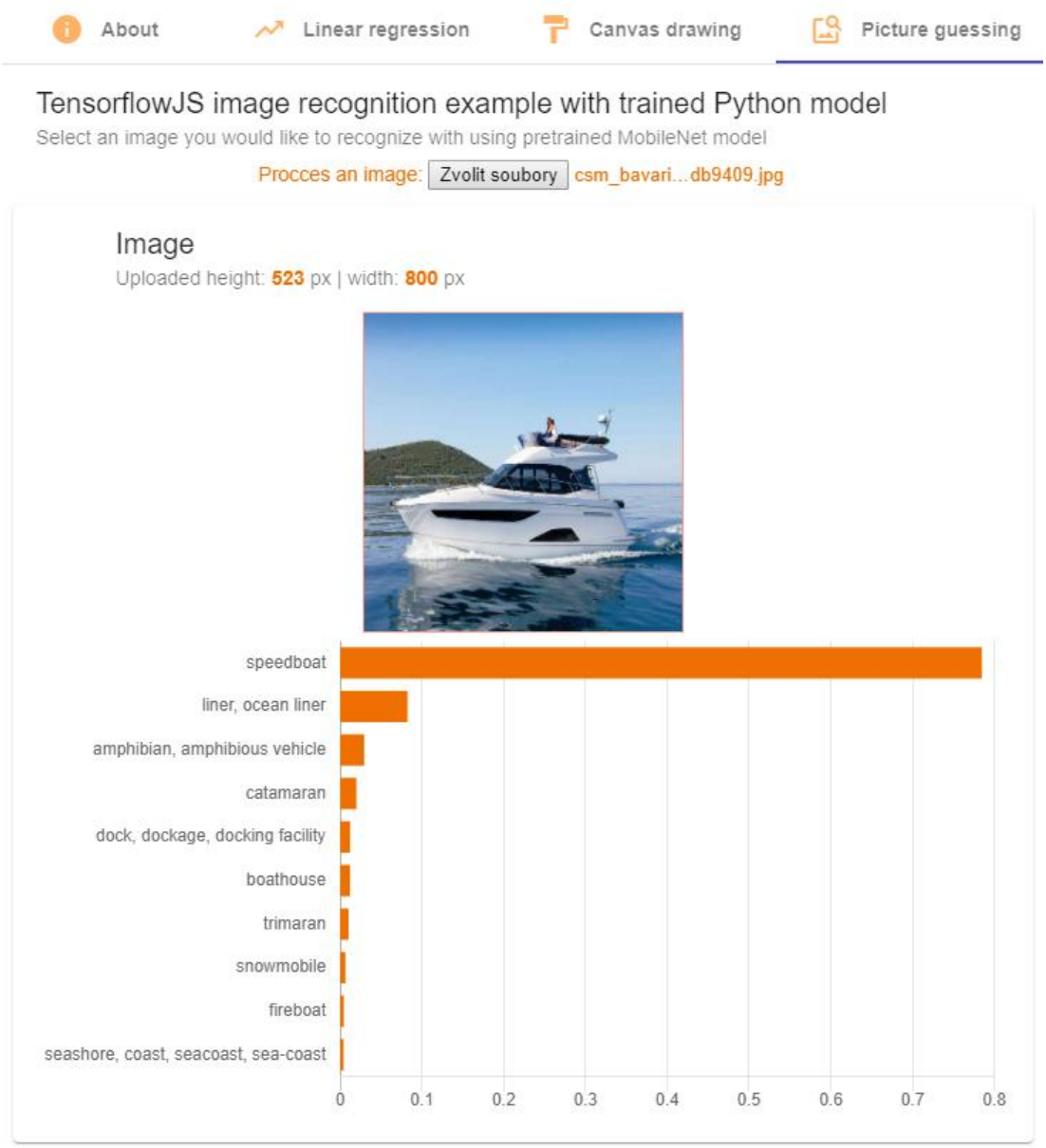


zdroj: autor

Klasifikace obrázků

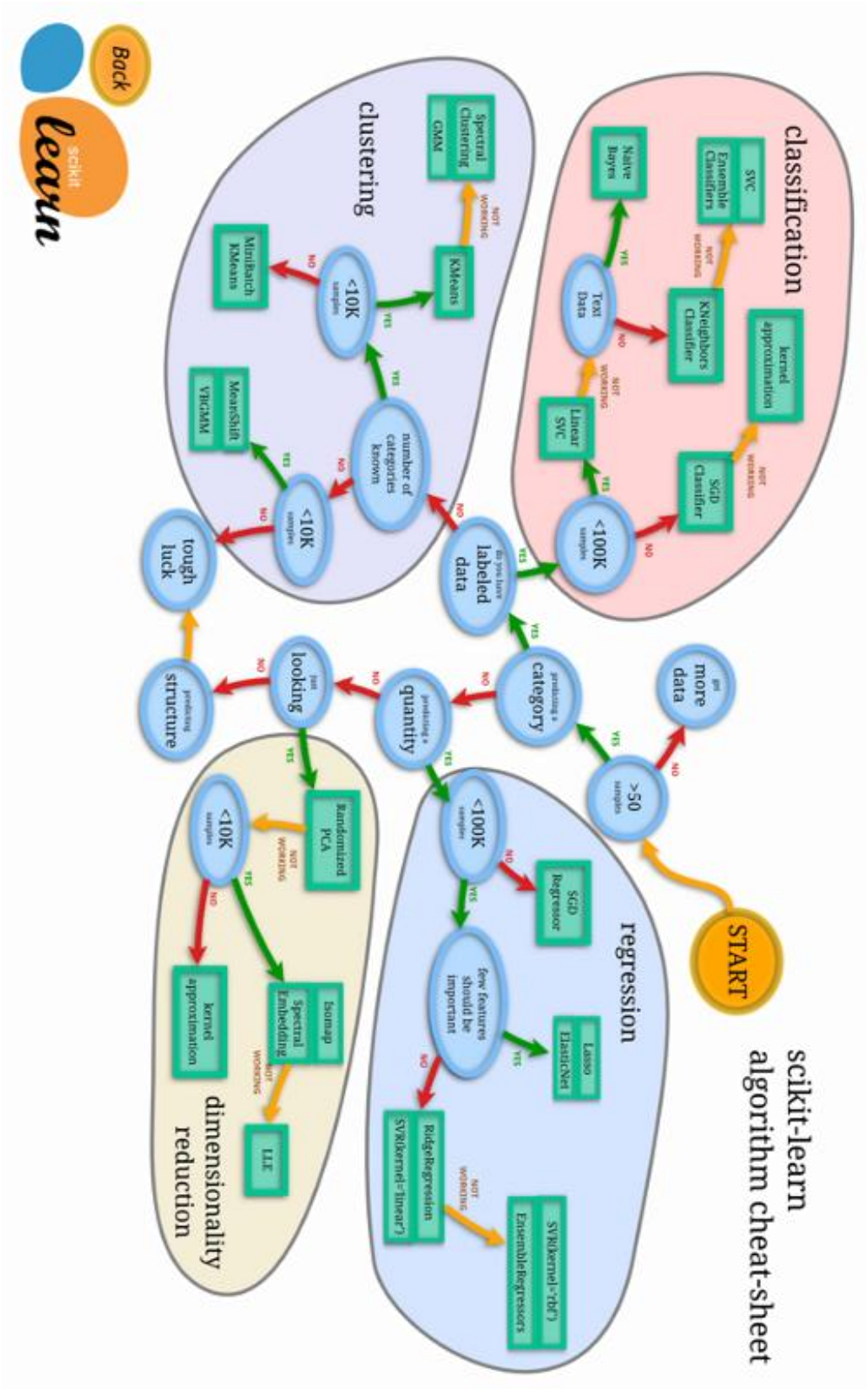
Rozhraní pro klasifikaci obrázků (obr. 43) obsahuje pouze jedno tlačítko pro nahraní souboru, který se pokusí model klasifikovat. Po nahraní prvního souboru dojde k načtení modelu z URL adresy, proto je čas ke klasifikaci prvního souboru delší. Panel zobrazuje náhled obrázku včetně dodatečné informace o jeho velikosti. Data obrázku se po nahrání transformují do tvaru, se který model očekává. Model umí klasifikovat do 1000 kategorií. Po klasifikaci je v grafu pod obrázkem zobrazené rozdělení pravděpodobností pro prvních 10 kategorií seřazených dle nejvyšší pravděpodobnosti.

Obrázek 43 - Příklad s klasifikací obrázků



zdroj: autor

Obrázek 44 - Pomůcka pro výběr správného algoritmu



zdroj: http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

Repositář se zdrojovým kódem

Zdrojový kód celé aplikace je k dispozici na adrese:

- *https://github.com/NutCrackee/angular-tensorflowjs*

Repositář si lze stáhnout pomocí příkazu (je nutné mít nainstalovaný systém pro správu verzí Git)

- *git clone https://github.com/NutCrackee/angular-tensorflowjs.git*

Zdrojové kódy webové aplikace jsou dostupné v příloze.