Ekonomická fakulta
Faculty of Economics
Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v Českých Budějovicích
Ekonomická fakulta
Katedra aplikované informatiky a matematiky

Diplomová práce

Tvorba chatovacího robota s využitím hlubokých neuronových sítí

Vypracoval: Bc. Tom Schuh
Vedoucí práce: doc. Ing. Ladislav Beránek CSc.

České Budějovice 2021

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH

Ekonomická fakulta

Akademický rok: 2019/2020

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Tom SCHUH**
Osobní číslo:	**E19127**
Studijní program:	**N6209 Systémové inženýrství a informatika**
Studijní obor:	**Ekonomická informatika**
Téma práce:	**Tvorba chatovacího robota s využitím hlubokých neuronových sítí**
Zadávající katedra:	**Katedra aplikované matematiky a informatiky**

Zásady pro vypracování

Práce se bude zabývat využitím hlubokých neuronových sítí při zpracování přirozeného jazyka za účelem vytvoření chatovacího robota. Bude se zabývat přístupy pro statistické modelování přirozeného jazyka, porovnáním různých přístupů. V praktické části bude cílem implementace architektury hlubokého učení s využitím jazykových vektorových modelů a zpracování trénovacích dat pro natrénování chatbota. Bude provedeno ověření dosažených výsledků.

Metodický postup:

1. Studium odborné literatury.
2. Teoretický popis konkrétních dostupných technologií pro chatovací roboty, teoretický popis principů neuronových sítí.
3. Popis přístupů pro statistické modelování přirozeného jazyka.
4. Vytvoření chatovacího robota s využitím neuronových sítí s použitím vhodných nástrojů.
5. Vypracování doporučení a závěrů.

Rozsah pracovní zprávy:	**50 – 60 stran**
Rozsah grafických prací:	
Forma zpracování diplomové práce:	**tištěná**

Seznam doporučené literatury:

1. Brownlee, J. (2017). *Deep Learning for Natural Language Processing. 1.1 editor*. Washington: Jason Brownlee.
2. Ram, S. (2016). *Chatbots with Seq2Seq*. [online]. Dostupné z: <http://complx.me/2016-06-28-easy-seq2seq/>.
3. Serban, I. V., & kol. (2018). *A Deep Reinforcement Learning Chatbot*. [online]. Dostupné z: <https://arxiv.org/abs/1801.06700>.
4. Další časopisecká a knižní literatura dle zaměření práce.

Vedoucí diplomové práce: **doc. Ing. Ladislav Beránek, CSc.**
Katedra aplikované matematiky a informatiky

Datum zadání diplomové práce: **17. ledna 2020**
Termín odevzdání diplomové práce: **16. dubna 2021**

JIHOČESKÁ UNIVERZITA
V ČESKÝCH BUDĚJOVICÍCH
EKONOMICKÁ FAKULTA (26)
Studentská 13
370 05 České Budějovice

doc. Dr. Ing. Dagmar Škodová Parmová
děkanka

doc. RNDr. Tomáš Mrkvička, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 26. března 2020

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své diplomové práce, a to v nezkrácené podobě/v úpravě vzniklé vypuštěním vyznačených částí archivovaných Ekonomickou fakultou - elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

…… ……

Datum Podpis

Poděkování

Rád bych tímto poděkoval vedoucímu své diplomové práce doc. Ing. Ladislavu Beránkovi CSc. za cenné rady a podporu při zpracování této diplomové práce.

Obsah

1 Úvod

Umělá inteligence je v dnešní době častým řešením nejrůznějších problémů. Setkáváme se s ní v každodenním životě; již při otevření webového prohlížeče se s její pomocí zpracovávají data o našich vyhledáváních. I když spadá pod obor informatiky, její implementace může pomáhat při řešení mnoha různých problémů téměř v každém oboru. Běžně se využívá například při konstrukci autonomních vozidel, zpracování dat v marketingu nebo při vytváření chatbotů.

Hlavní funkcí chatbotů je poskytovat uživatelům snadno dostupný zdroj informací, které se v danou chvíli snaží získat. V poslední době narůstá míra dezinformací šířených různými médii nebo jednotlivci, a proto je chatbot v této diplomové práci určený ke zodpovídání otázek týkajících se současného stavu pandemie COVID-19.

Čtenář se seznámí s několika typy umělé inteligence s detailním zaměřením na strojové učení pomocí různých typů hlubokých neuronových sítí, jako jsou například Long-Short Term Memory anebo Gated Recurrent Unit. Dále tato práce čtenáře seznámí s problematikou chatbotů a zpracování textu.

Teoretická část obsahuje stručný přehled o historii chatbotů a také popisuje různé typy chatbotů. Dále jsou zde vysvětleny pojmy týkající se hlubokých neuronových sítí, strojového učení atd. V neposlední řadě je zde věnován prostor různým metodám, problémům a postupům, se kterými je možné se setkat při vytváření chatbota pomocí hlubokých neuronových sítí.

Vývojem chatbota pomocí hlubokých neuronových sítí se zabývá praktická část této práce. Chatbot je následně zasazen do aplikace a podroben experimentům při jeho učení. Tyto experimenty s modelem jsou vykonány za použití dvou různých typů rekurentních neuronových sítí a následné optimalizace učení tohoto modelu.

2 Teoretická část

V této části diplomové práce jsou uvedeny informace o strojovém učení a umělé inteligenci. Dále je zde podrobněji představena teorie chatbotů.

2.1 Chatbot

Podle Cambridge Dictionary je pojem chabot definován jako:

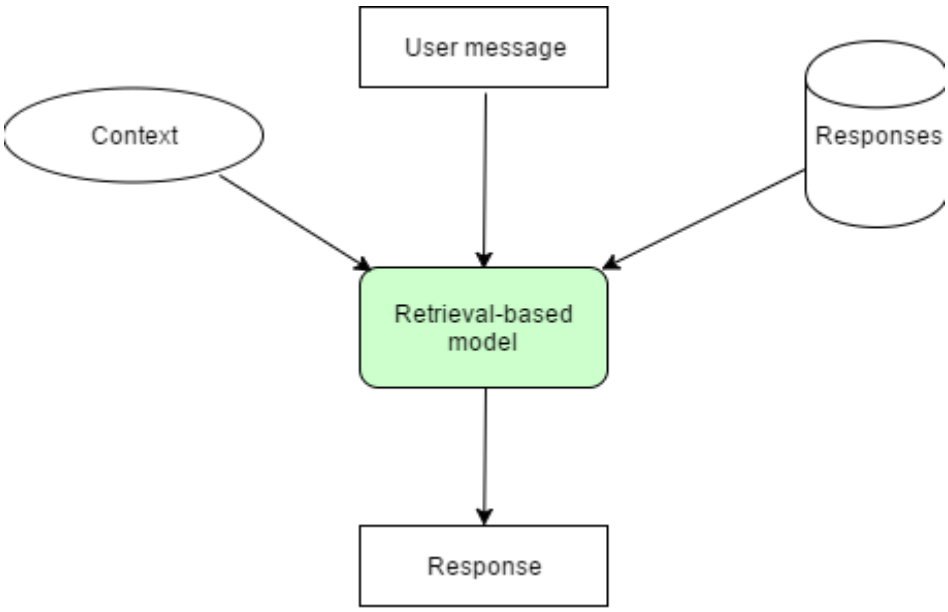
„A computer program designed to have a conversation with a human being, especially over the internet.“ (Cambridge University Press, 2021)

Volně přeloženo: Chatbot je počítačový program, který je navrhnutý pro komunikaci s lidskou bytostí, zejména přes internet. Základním principem fungování chatbota je interakce s člověkem pomocí textových zpráv (nebo pomocí mluveného slova), kdy chatbot porozumí vstupní informaci od člověka a je schopen vytvořit kontextuálně správnou odpověď jako výstup. Vytvoření prvního konceptu chatbota je připisováno Alanu Turingovi. Od doby A. Turinga se technologie posunula dopředu, a s ní i vývoj chatbotů. S tímto souvisí také tzv. Turingův test, který slouží k testování přesvědčivosti umělé inteligence (v tomto případě tedy chatbotů). Základní princip Turingova testu je, že osoba komunikuje s entitou; s počítačem nebo s jinou osobou. Pokud komunikující osoba nedokáže rozpoznat, zda komunikuje s počítačem nebo s člověkem, znamená to, že umělá inteligence splnila Turingův test (Turing, 1950).

Z podstaty testu vyplívá, že se zaměřuje především na tzv. generativní chatboty neboli chatboty, kteří jsou navrhnutí tak, aby byli schopni vést plynulou konverzaci podobnou dialogu dvou lidí. Tento typ chatbota ovšem není zcela vhodný pro komerční využití. Chatboty můžeme rozlišovat podle principu fungování na dva typy:

- Retrieval-Based Chatbot: Jedná se o nejběžnější typ chatbota, se kterým se dnes můžeme setkat. Disponuje uloženou konečnou sadu manuálně předpřipravených odpovědí (například v databázi) a snaží se z této sady vybrat nejvhodnější odpověď na základě vstupní zprávy. Schéma tohoto typu chatbota si lze prohlédnout na obrázku 2.1. Z podstaty jeho fungování vyplívá, že je značně omezen velikostí sady odpovědí. Výhodou je, že zřídkakdy dělá gramatické chyby a že k jeho vytvoření není potřeba velké datové sady. Proto je tento typ nejběžnější v komerční sféře (Mishra, 2020).

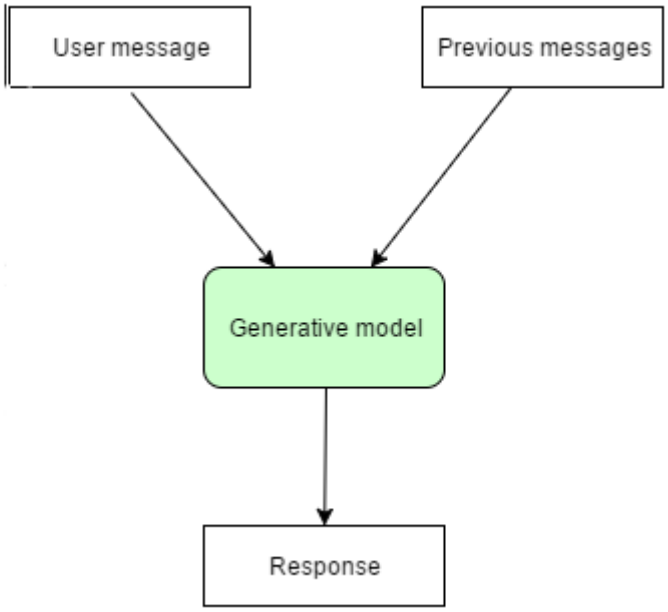
Obrázek 2.1 – Schéma chatbota typu Retrieval-Based



(Zdroj: https://medium.com/analytics-vidhya/different-types-of-approaches-to-make-your-own-chatbot-retrieval-vs-generative-8d524b0505db)

- Generative Model: Generativní model nepoužívá sadu již předpřipravených odpovědí. Naopak, jak název napovídá, generuje odpovědi sám. K jeho vytvoření je potřeba velkého počtu předchozích konverzací, na základě kterých se generují

Obrázek 2.2 – Schéma chatbota typu Generative



(Zdroj: https://medium.com/analytics-vidhya/different-types-of-approaches-to-make-your-own-chatbot-retrieval-vs-generative-8d524b0505db)

odpovědi. Pro lepší pochopení můžeme srovnat schéma generativního modelu (obrázek 2.2) a schéma Retrieval-Base modelu (obrázek 2.1) (Mishra, 2020).

Dalším důležitým pojmem, zejména pro komerční účely, je Goal-Oriented chatbot. Tento typ chatbota je určen především k řešení problémů uživatelů; může se jednat například o získání informací, koupi lístků do kina či zarezervování letenky. Chatbot se skládá ze tří hlavních částí: Dialog Manager (DM), Natural Language Understanding (NLU) a Natural Language Generator (NLG). Jednotka NLU slouží k porozumění vstupu, jednotka NLG slouží naopak ke generování odpovědi. Tyto dvě složky bývají obdobné ve všech chatbotech. Důležitou částí v GO chatbotech je Dialog Manager, který má za úkol vést dialog určitým směrem, a to tak, aby získal od uživatele veškeré informace potřebné k vyřešení jeho problému a umožnil tak dosažení jeho cíle (Brenner, 2018).

Můžeme se setkat také s rozdělením chatbotů na Transactional chatboty a Conversational chatboty; toto dělení se používá především v komerčním prostředí. Conversational chatboti byli již popsáni v předchozích odstavcích. Transactional chatbot splňuje definici chatbota, jedná se nicméně o jeho nejprimitivnější verzi, neboť tento typ má předdefinované jak odpovědi, tak i vstupy. Jeho schopnosti jsou proto velmi omezené a z programátorského hlediska se jedná o jednoduché zpracování vstupu pomocí podmínek (Robino, 2016).

2.2 Historie chatbotů

Jak již bylo uvedeno výše, chatbot je program, který pomocí umělé inteligence dokáže komunikovat s osobou nebo s jiným chatbotem. Největší zájem o tento fenomén (ať už s jednodušší nebo složitější umělou inteligencí) přišel s rozšířením internetu a sociálních sítí. Následně se začaly tyto programy používat v mnoha oblastech týkajících se e-Business (Zemčík, 2019).

Nejznámějším a zároveň jedním z nejstarších chatbotů je „Eliza“, který byl vytvořen profesorem Josephem Weizenbaumem v laboratoři umělé inteligence na MIT okolo roku 1965. Jeho název byl inspirován knihou Pygmalion. Tento chatbot se na dlouhou dobu stal inspirací pro mnoho vývojářů v tomto odvětví. Simuloval roli psychologa, který se ptá na otevřené otázky a následně na ně odpovídá (Zemčík, 2019).

Dalším příkladem známějšího chatbota z historie je Parry. Ten byl představen okolo roku 1972 psychiatrem K. M. Clobym, který působil na Standforském psychiatrickém oddělení. Parry byl v podstatě opakem chatbota Eliza – simuloval chování paranoidního

a schizofrenního pacienta. Sloužil hlavně jako pomocník ke studiu chování pacienta s touto poruchou, který využívali především mladší psychiatři. V roce 1972 byla na počítačové konferenci zorganizována konverzace mezi chatboty Eliza a Parry (Zemčík, 2019).

Poměrně známý je také Racter, což je chatbot vytvořený W. Chamberlainem a T. Etterem v roce 1983 pod záštitou společnosti Inrac Corporation. Původním záměrem bylo, aby generoval náhodnou anglickou prózu. Tento chatbot byl napsán v jazyce BASIC na mikroprocesoru Z80 s 64KB pamětí RAM, což je zařízení nesrovnatelné s dnešním hardwarem (Zemčík, 2019).

Dnes je již vývoj chatbotů samostatným programátorským odvětvím. Existují dokonce soutěže, které slibují zajímavé finanční ohodnocení skupinám vývojářů, kteří předvedou nejlepšího chatbota. Jednou takovouto soutěží je Alexa Prize, která je pořádána společností Amazon. Příkladem dnešního chatbota může být ten, kterého používá aerolinka KLM. Pomáhá klientům získat informace o odbavení či o zpožděných letech a dokáže také vystavit kopie palubních lístků. Zvládl odpovědět na 10 % otázek bez zásahu člověka a zvýšil rychlost odpovědi o 20 %. Chatboti se každým rokem stávají běžnějšími a kupředu se posouvá také jejich komplexnost. Velmi zajímavým příkladem je chatbot Sophia. Podobá se člověku a je specifický tím, že je schopen vyjadřovat určité výrazy obličeje (Zemčík, 2019).

2.3 Použití chatbotů

Chatboti mohou sloužit k mnoha účelům od zákaznické podpory, informačního kanálu až po pouhé pobavení. Jsou považováni za alternativu tradičního přístupu k zákaznickým službám, ale hojně jsou využívání také jako virtuální asistenti, kteří pomáhají při provádění určitého úkolu. Tímto způsobem jsou používáni například v Indii, kde pomáhají se zamluvením taxi služby nebo s objednáním jídla. Ve Velké Británii zase slouží k vyřizování stížností na parkovací pokuty nebo k poskytování rad zdravotního charakteru (Brandtzeag & Folstad, 2017).

Studie od P. B. Brandtzaega a A. Folstada, která byla zpracována na základě 146 odpovědí, zkoumala důvody pro používání chatbotů. Získané odpovědi byly rozděleny do 5 kategorií: produktivita, zábava, sociální, zvědavost a ostatní. Největší procento dotázaných uvedlo, že hlavní důvod pro používání chatbotů je produktivita. Většina lidí z této kategorie považovala za hlavní přínos jednoduchost jejich použití,

rychlost a spolehlivost. Z odpovědí vyplynulo, že dle názorů uživatelů je největší výhodou chatbotů to, že pomáhají ušetřit čas při získávání informací nebo poskytování asistence. Tato skupina taktéž potvrdila, že používá chatboty denně, například při plnění osobních úkolů. Další skupina (přibližně 20 % tázaných) používá chatboty pro zábavu. Pro sociální účely chatboty využívá přibližně 10 % dotazovaných lidí. Zvědavost a zájem o chatboty jako takové uvedlo jako hlavní důvod přibližně 10 % respondentů, přičemž většinou se zajímali především o jejich limity. Do kategorie „ostatní“, tzn. používání chatbotů k jiným účelům, které nebyly zahrnuty v žádné z předešlých kategorií, spadalo pouze několik málo odpovědí. Z této studie plyne, že většina lidí, kteří používají různé typy chatbotů, s nimi pracuje denně. Někteří z respondentů také uvedli, že preferují řešení problémů skrze chatbota před klasickým pojetím zákaznických služeb (Brandtzeag & Folstad, 2017).

2.4 Retrieval-Based chatbot

Nyní si detailněji představíme fungování chatbota typu Retrieval-Based. Schéma tohoto typu chatbota je k nahlédnutí na obrázku číslo 2.

V první fázi po přijetí zprávy dojde k upravení textu modulem pro zpracování jazyka. Dojde například k odstranění stopwords, stemování nebo k odstranění synonym. Tento krok pomáhá modelu k efektivnějšímu učení z dat a používá se především pro aplikace, které nedisponují velkou datovou sadou.

V další fázi model převede pomocí slovníku větu do sekvence čísel. Následně je tato sekvence pomocí třídy Embedding převedena na vektor čísel, který slouží jako reprezentace slov s kontextem. Tento vektor je poskytnut samotnému „před-trénovanému“ modelu, který se pokusí určit intent neboli záměr (může jím být například otázka).

V poslední fázi aplikace chatbota zkontroluje, zda pravděpodobnost správného určení záměru dosahuje určitého procenta a tím eliminuje možnost chybné odpovědi. Čím větší odpovědnost je kladena na správnost odpovědi, tím musí být akceptovatelná hranice procentuální pravděpodobnosti správné odpovědi vyšší. Pokud je míra pravděpodobnosti správného určení záměru vyhovující, dojde k vybrání příslušné odpovědi z již předem připravené sady. Pokud je pravděpodobnost nevyhovující, chatbot uživateli sdělí, že neporozuměl otázce.

2.5 Umělá inteligence a strojové učení

Podle slovníku Britannica lze umělou inteligenci definovat jako schopnost digitálních počítačů nebo robotů kontrolovaných počítači provádět úkoly, které jsou obvykle spojené s inteligentními bytostmi (Encyclopædia Britannica, Inc., 2021).

V dnešní době se termín umělá inteligence používá pro označení schopnosti počítače nebo stroje napodobit chování lidské mysli. To znamená učit se z příkladů a zkušenostní, rozeznávat objekty, rozumět jazyku, dokázat vhodně zareagovat aj. Pojem umělá inteligence zahrnuje veškeré systémy, které dokáží vyřešit problémy a učinit rozhodnutí na základě komplexních pravidel a logiky (IBM, 2020).

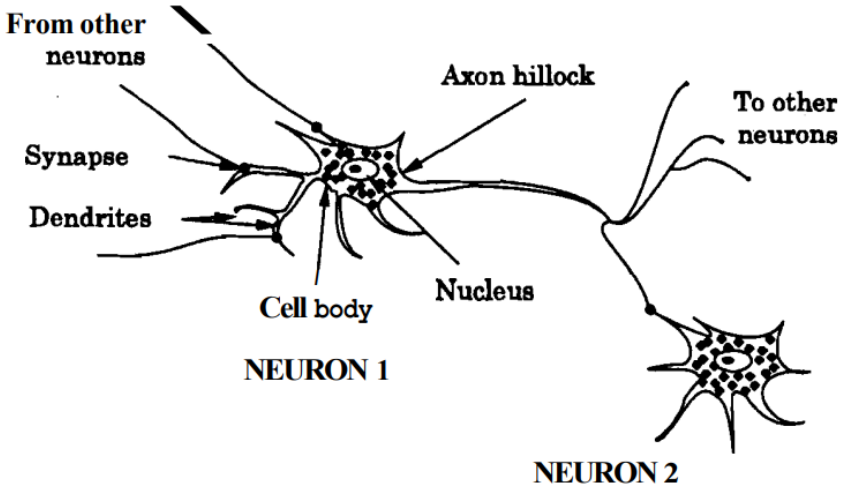
Do této definice lze tedy zařadit také strojové učení (v angličtině Machine Learning). Jeho základní myšlenka je taková, že aplikace využívající tohoto principu jsou schopny učit se z dat a zvyšovat tak jejich přesnost v čase bez toho, aby k tomu byly naprogramovány. Jedná se tedy o algoritmus, který lze „trénovat" k tomu, aby našel určitý vzor a vlastnosti ve velkém množství dat a aby poté mohl provádět rozhodnutí a predikce na základě těchto dat. Nyní si uvedeme příklady různých typů strojového učení:

- Regresní algoritmus – Zakládá se na lineární a logické regresi. Tyto regrese fungují pomocí principu předpovídání závislé proměnné na základě vztahů mezi daty. Logická regrese se používá v případech, kdy je závislá proměnná binární.
- Rozhodovací stromy – Jedná se o algoritmy, které se používají k rozhodování na základě dat a stanovených pravidel.
- Shlukovací algoritmus – Používá se k shlukování záznamů a následnému vytváření skupin (neboli shluků) na základě určitých vlastností dat.
- Neuronové sítě – Tento algoritmus definuje síť rozdělující se na vrstvy výpočtů. První je vstupní vrstva, která přijímá data. Následuje alespoň jedna skrytá vrstva, kde jsou prováděny výpočty a kde se vytváří závěry o vstupu. Poslední je výstupní vrstva, která poskytuje výstup s určitou pravděpodobností (IBM, 2020).

2.5.1 Neuron

Základní jednotka každé neurnové sítě se nazývá neuron. Neurony, které se používají ve strojovém učení, vycházejí z fungování neuronů biologických; tedy alespoň některé jejich funkce. Biologické neurony se někdy nazývají také nervové buňky. Zjednodušené schéma takové nervové buňky si můžeme prohlédnout na obrázku číslo 2.1. Z obrázku je patrné, že se biologický neuron skládá z buněčného těla, ve kterém se nachází buněčné jádro (v angličtině nucleus). Neuron dále obsahuje dendrity, které slouží k přijímaní signálů z jiných neuronů. Prodloužená část buněčného těla, která se nazývá axon, je rozvětvena do řetězců, které jsou napojeny na mnoho dalších buněk pomocí takzvaných synapsí. Je běžné, že je axon napojen na několik tisíc synapsí, které jsou spojeny jinými neurony. Průměrný počet synapsí v lidském mozku je přibližně 10^15. Přenosy signálů v synapsích probíhají pomocí složitých chemických procesů (Yegnanarayana, 2009).

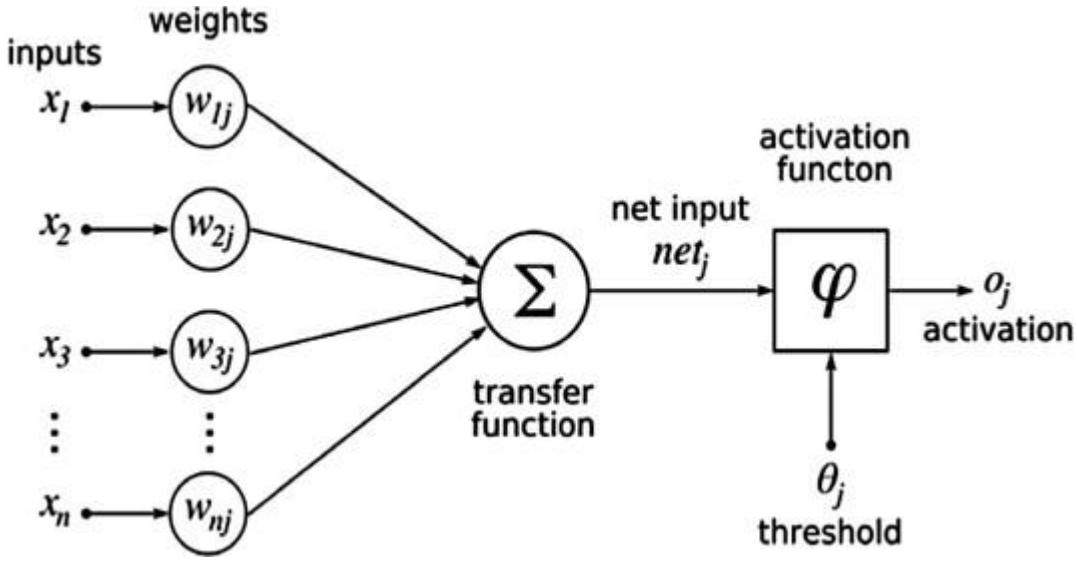
Obrázek 2.1 – Schéma biologického neuronu



(Zdroj: Yegnarareyana, 2009)

Nyní se přesuneme od biologického neuronu k umělému. Na obrázku číslo 2.2 si můžeme prohlédnout model neuronu z vícevrstvého perceptronu, který poslouží pro vysvětlení jeho základních částí. První částí jsou vstupy, které jsou upraveny podle nastavených (naučených) vah. Tyto hodnoty se sečtou a pomocí aktivační funkce převedou na výstup neuronu, který může být propojen s dalšími neurony anebo může sloužit jako výstup skryté vrstvy neuronové sítě (Yegnanarayana, 2009).

Obrázek 2.2 – Schéma umělého neuronu



(Zdroj: https://www.researchgate.net/figure/A-typical-multilayer-perceptron-ANN-architecture-Ahmed-et-al-2013_fig1_262526787)

Pro lepší pochopení lze porovnat umělý a biologický neuron a povšimnout si, že v tomto srovnání jsou náhradou za dendrity vstupy umělého neuronu a náhradou za synapse jsou váhy umělého neuronu. Buněčné jádro by bylo zastoupeno transferovou a aktivační funkcí a prodloužená část buněčného těla (axon) výstupem umělého neuronu (Yegnanarayana, 2009).

2.6 Metody používané v hlubokém učení

V této kapitole jsou představeny metody, které se používají k zajištění správného fungování hlubokých neuronových sítí.

Pro lepší pochopení těchto metod je nezbytné vysvětlit několik základních pojmů, které definují fungování jednotlivých neuronových sítí:

Typy neuronů – Neuronem rozumíme jednotku, která disponuje vstupy a produkuje jeden výstup, který může být použit jako vstup pro další neuron nebo jako výstup skryté vrstvy neurnové sítě. Každá neuronová síť má určitý typ neuronů neboli uzlů. Příklady mohou být: McCulloch-Pitts neuron, binární neurony, lineární prahová funkce a další (Shanmuganathan & Samarasinghe, 2016).

Architektura neuronových sítí – Tento pojem udává, jakým způsobem jsou neurony neboli uzly mezi sebou propojené a tento způsob popisuje jako architekturu. Mezi neurony může docházet k takzvanému úplnému propojení (Fully connected neural

network). To znamená, že každý uzel v každé vrstvě je spojený s každým uzlem v další vrstvě. Druhým typem propojení je částečné (Partically connected neural network), kdy sítě obsahují pouze podmnožinu z množiny všech možných spojení modelu neuronové sítě. Důvodem pro používání těchto sítí je, že částečné propojení může pomoci generalizaci a snížit tak složitost sítě, nároky na hardware a na úložiště a dobu učení (Elizondo & Fiesler, 1997).

Dalšími vlastnostmi skrývajícími se pod pojmem architektura spojení je počet vstupních neuronů, výstupních neuronů a počet vrstev. Podle těchto parametrů se neuronové sítě dělí na auto-asociativní a hetero-asociativní. Auto-asociativní jsou trénovány tak, aby byly vstupní a výstupní data identické. Jinými slovy jsou vstupní neurony zároveň neurony výstupními. Oproti tomu vstupní a výstupní neurony hetero-asociativní neuronové sítě jsou oddělené, jako například v případě MLP (Kropas-Hughes, Oxley, Rogers, & Kabrisky, 2000).

Typy architektury neuronových sítí lze také dělit na dopředné (Feedforwad) a zpětné (Feedback). Dopředná architektura nemá žádné zpětné spojení mezi jednotlivými neurony a nevznikají zde tedy žádné cykly. Síť si nepamatuje předešlé hodnoty vstupů a stavy neuronů. Zpětná architektura obsahuje zpětné spojení mezi výstupními a vstupními neurony; neuronové sítě tohoto typu si drží v paměti předešlé stavy, které následně používají k vyhodnocení dalších hodnot (Shanmuganathan & Samarasinghe, 2016).

Učící algoritmus – Posledním důležitým pojmem, který je třeba vysvětlit, je samotný učící algoritmus, respektive jeho typy. Existuje široká škála učících algoritmů, ale pro zjednodušení lze vymezit tyto tři skupiny:

- Kontrolované učení (Supervised learning) funguje tak, že se neuronová síť snaží přiřadit ke každému vstupnímu vektoru *x* odpovídající výstupní vektor *y*.
- Učení bez dozoru (Unsupervised learning) je považováno za další skupinu učících algoritmů, kde je neuronové síti poskytnut pouze vstupní vektor *x* a síť se následně snaží naučit se některé vnitřní rysy celé datové sady.
- Poslední skupinou učících algoritmů je „Posílené učení“ neboli Reinforcement learning, které se někdy označuje také jako učení pomocí trestu a odměny. Princip jeho fungování je takový, že neuronová síť se pokusí vypočítat odpovídající výstup, a pokud je tento výstup správný, váhy daného

propojení se zvýší (odměna). Pokud výstup správný není, váhy daného propojení se sníží (trest) (Shanmuganathan & Samarasinghe, 2016).

2.6.1 Ztrátová funkce

Objektivní funkce slouží k vyhodnocení nastavení vah v modelu. Výstup této funkce se můžeme snažit minimalizovat nebo maximalizovat. Protože se u většiny neuronových sítí snažíme snížit chybu na co nejnižší hodnotu (tedy ji minimalizovat), tak se tato objektivní funkce často označuje jako ztrátová a výstup této funkce se pak označuje jako ztráta. Existuje několik typů ztrátových funkcí dělených podle toho, jak se počítají (Brownlee, 2019).

Níže jsou uvedeny nejzákladnější typy ztrátových funkcí:

Regresní ztrátová funkce vychází z matematické metody lineární regrese. Regresních ztrátových funkcí je několik druhů, například L1 Loss, L2 Loss, Huber Loss atd. Pro představu, jak takováto funkce funguje, jsou zde popsány jejich základní typy (v následujících vzorcích je použito následující označení: L – ztráta, y – skutečná hodnota, $f(x)$ – výsledná hodnota predikční funkce) (Mahendru, 2019):

- Absolute Error Lost nebo také L1 Loss se vypočítá podle vztahu 2.1 jako vzdálenost mezi predikovanými a skutečnými hodnotami v absolutní hodnotě.

$$L = |y - f(x)| \qquad (2.1)$$

- Squared Error Loss nebo také L2 Loss se vypočítá podle vztahu 2.2 jako rozdíl mezi skutečnou hodnotou a predikovanou hodnotou na druhou (Mahendru, 2019).

$$L = (y - f(x))^2 \qquad (2.2)$$

Funkce ztráty binární klasifikace jsou, jak lze vyvodit z jejich názvu, založené na binární klasifikaci, tedy na zařazení objektu do jedné ze dvou tříd. Může se jednat například o rozhodnutí, zda patří e-mailová zpráva do spamu či nikoliv. Příkladem funkce ztráty binární klasifikace budiž funkce Binary Cross Entropy Loss. Ta se opírá o pojem entropie, který lze definovat jako míru neurčitosti. Souvisejícím pojmem je křížová entropie, která se používá ke kvantifikaci rozdílu mezi dvěma pravděpodobnostními rozděleními a je proto vhodná jako ztrátová funkce u modelů, jejichž výstupem je pravděpodobnost (Mahendru, 2019).

Funkce ztráty binární křížové entropie je definována podle vztahu 2.3, kde y značí původní hodnotu a může nabývat pouze hodnot 1 nebo 0 a p značí pravděpodobnost, že prvek patří právě do třídy 1 (Mahendru, 2019).

$$L = -y * \log(p) - (1-y) * \log(1-p) = \begin{cases} -\log(1-p), & pokud\ y = 0 \\ -\log(p), & pokud\ y = 1 \end{cases} \quad (2.3)$$

Funkce ztráty při klasifikaci více tříd je obdobou funkce ztráty binární klasifikace s tím rozdílem, že v tomto případě je objekt možné zařadit do více než dvou tříd. Pro představu si opět uvedeme příklad s e-mailovými zprávami. Tentokrát lze zprávu klasifikovat do několika různých kategorií jako je práce, zábava, nákupy, rodina atd. Jako zástupce těchto ztrátových funkcí je níže popsána Multi-Class Cross Entropy Loss, která je obdobou Binary Cross Entropy Loss, neboť také využívá křížovou entropii pro výpočet ztráty. Tato funkce nicméně vychází z pravděpodobnosti *p*, že vstupní vektor X_i odpovídá právě jednomu výstupnímu vektoru Y_i. Výpočet ztrátové funkce Multi-Class Cross Entropy Loss je zaznamená vztahem 2.4, kde y_{ij} se rovná 1, pokud je *i-tý* element obsažený v třídě *j*. V ostatních případech se rovná 0 (Mahendru, 2019).

$$L(X_i, Y_i) = -\sum_{j=1}^{c} y_{ij} * \log(p_{ij}) \quad (2.4)$$

2.6.2 Activation function

Zjednodušeným vysvětlením je, že aktivační funkce rozhoduje o tom, zda bude neuron aktivován nebo nikoliv. Jinými slovy posuzuje, zda je neuronem poskytovaný vstup relevantní (Sharma, 2019).

Podíváme-li se na schéma neuronu (obrázek číslo 2.2), uvidíme, že aktivační funkce je označena jako jeho poslední část. Z obrázku je také patrné, že výstup aktivační funkce je finální výstupem neuronu. Existuje několik typů aktivačních funkcí:

1. Funkce binárního kroku (Binary Step Function);
2. Lineární funkce (Linear Fuction);
3. Sigmoidní funkce (Sigmoid Function);
4. Hyperbolická tangensová funkce (Hyperbolic Tangent Fuction);
5. ReLu;
6. Leaky Relu;
7. Funkce Softmax (Softmax function).

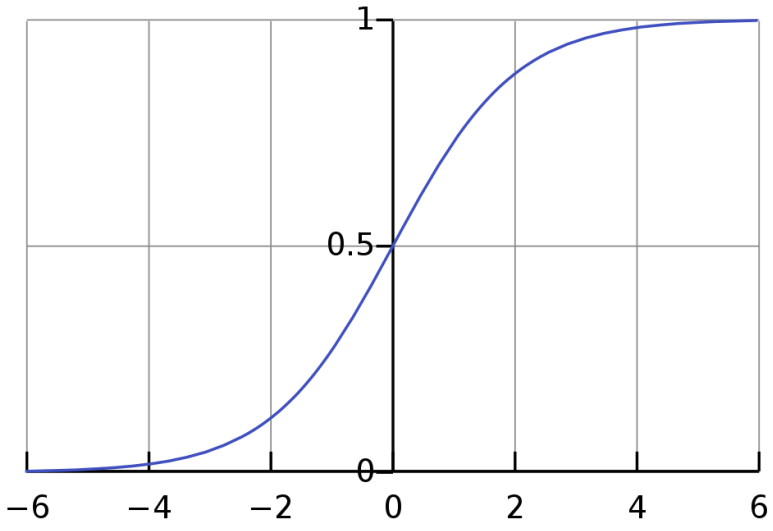
Nyní budou popsány nezákladnější a nejčastěji používané aktivační funkce (Sharma, 2019).

Sigmoidní aktivační funkce vrací výstup v intervalu 0 až 1. Jedná se o nejčastěji používanou aktivační funkci. Nejvíce užitečná je pro taková učící data, která se také pohybují mezi 0 a 1 (Sibi, Jones, & Siddarth, 2013). Její výpočet se provádí podle vztahu 2.5, kde *x* znázorňuje vstup do aktivační funkce (tedy vstupní data neuronu aktualizované podle nastavení vah) a S označuje samotnou sigmoidní funkci (Sharma, 2019).

$$S(x) = \frac{1}{1+e^{-x}} \qquad (2.5)$$

Na obrázku číslo 2.3 je znázorněn grafický zápis funkce. Podle údajů na ose y je z grafu patrné, že výstup z této funkce je v intervalu 0 až 1. Dále lze z grafu vyčíst, že mezi hodnotami -4 až 4 dochází na ose y k velkým změnám. Přesněji řečeno, malá změna mezi vstupními hodnotami -4 a 4 bude mít za následek velkou změnu výstupu. Z této vlastnosti vychází jisté omezení této funkce, které postihne právě ty hodnoty, které jsou mimo interval -4 a 4. Výstup této funkce se pro ně bude přibližovat nule, a i po jejich vynásobení bude končený výstup velmi malý. Tento problém se nazývá mizející gradient (Sharma, 2019).

Obrázek 2.3 – Graf sigmoidní funkce

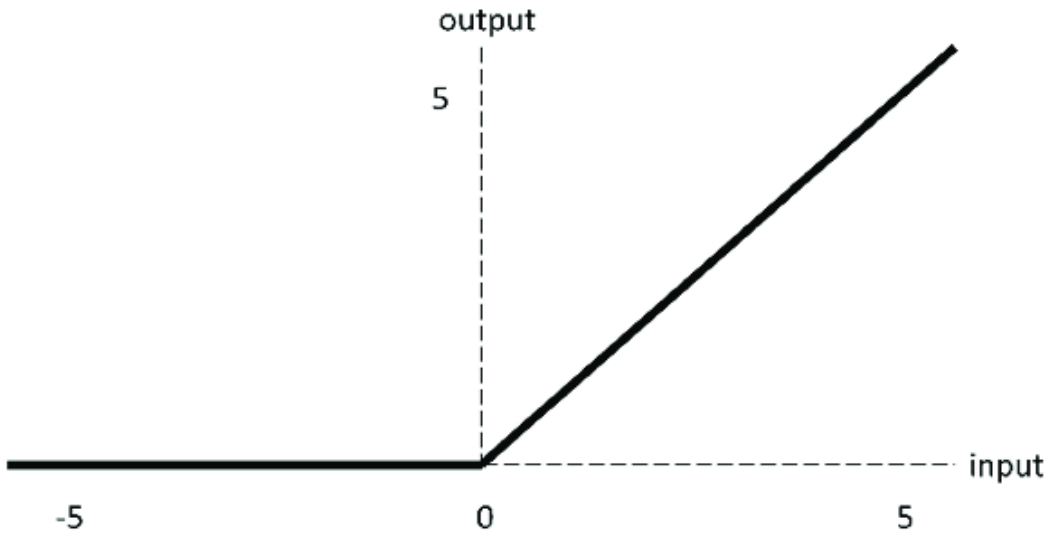


(Zdroj: https://en.wikipedia.org/wiki/Sigmoid_function)

ReLu (Rectified Linear Unit, v překladu „usměrněná lineární jednotka“) je aktivační funkce, která řeší problém sigmoidní funkce. Její grafické znázornění si lze prohlédnout na obrázku číslo 2.4, kde jsou jakákoliv záporná čísla zaměněná za 0. Z toho vyplívá vztah 2.6 (Sharma, 2019).

$$f(x) = x^{+} = max\ (0, x)\ (2.6)$$

Obrázek 2.4 – Graf ReLu aktivační funkce



(Zdroj: https://www.researchgate.net/figure/ReLU-activation-function_fig7_333411007)

Bohužel i u této funkce dochází k problému; neurony, které mají jako výstup záporné hodnoty, nebudou nikdy aktivovány. Tento nedostatek je nicméně vyřešen vylepšením původní aktivační funkce Relu, která se jmenuje Leaky ReLu. Místo toho, aby změnila negativní hodnoty na nulu je pouze zamění za nízké nezáporné hodnoty. Výsledný výpočet aktivační funkce Leaky ReLu je uveden ve vztahu 2.7 (Sharma, 2019).

$$f(x) = \begin{cases} x & pokud\ x > 0, \\ 0.01x & jinak. \end{cases} \qquad (2.7)$$

Softmax je aktivační funkce, která je někdy označována také jako funkce softargmax nebo logistická regrese více tříd. To proto, že funkce softmax je zobecněním logistické regrese pro více tříd. Aktivační funkci softmax lze tedy použít v případě, kdy se výstupní třídy vzájemně vylučjí. Jejím výstupem je vektor, který velikostí odpovídá počtu vstupních a výstupních tříd. Součet hodnot jednotlivých výstpuních tříd je vždy 1. Vstup funkce může být kladný, záporný nebo roven 0. Výstup bude ovšem vždy převeden

na hodnotu mezi 0 a 1 a lze ho proto interpretovat jako pravděpodobnost. Aktivační funkce softmax je velmi užitečná právě proto, že výsledný výstup je normalizovaným rozdělením pravděpodobnosti. Výpočet této funkce je zaznamenán vztahem číslo 2.8, kde z je vstupní vektor a K je počet klasifikovaných tříd. Jmenovatelem ve vztahu je normalizační výraz, jehož úkolem je, aby výstupní součet hodnot byl 1 a aby každá výstupní hodnota byla v intervalu mezi 1 a 0 (Wood, 2019).

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_i}} \; pro\; i = 1, \dots, K \; a \; z = (z_1, \dots, z_K) \; \epsilon \; \mathbb{R}^K \qquad (2.8)$$

2.6.3 Optimalizátor

Během učení modelu je zapotřebí optimalizovat váhy tak, aby se co nejvíce minimalizoval výstup ztrátové funkce, což zajistí, že predikovaná hodnota bude co nepřesnější. K tomuto účelu slouží právě optimalizátor, který propojuje ztrátovou funkci a parametry modelu tím, že aktualizuje model v závislosti na výstupu ztrátové funkce. Funkce ztráty optimalizátoru předávají informaci o tom, jestli se parametry modelu pohybují správným směrem (Algorithmia, 2018).

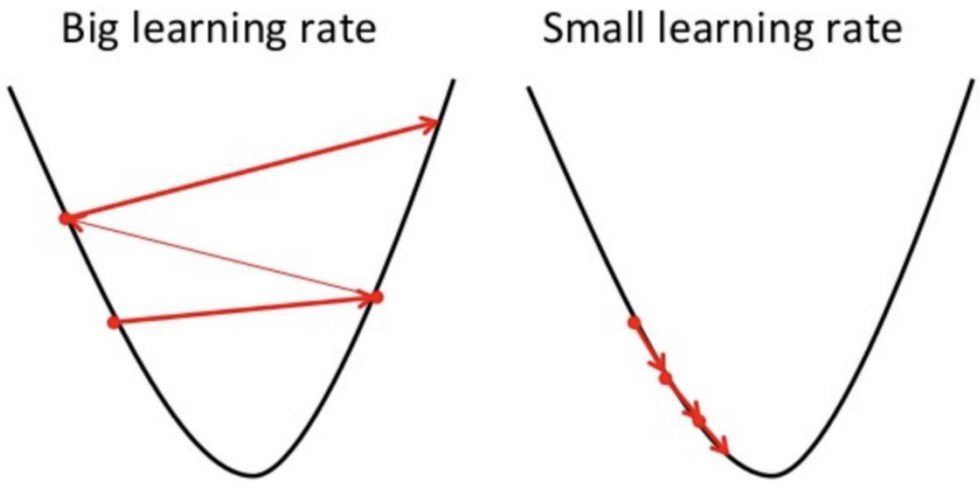
Nejznámějším optimalizátorem je Gradient Descent, v českém překladu přechodové klesání. Funguje následovně:

- Prvním krokem je výpočet hodnoty ztrátové funkce při malých změnách jednotlivých vah.
- Druhým krokem je reakce optimalizátoru na změnu výstupu ztrátové funkce. Podle toho, zda se hodnota výstupu ztrátové funkce zvýší nebo sníží, dojde k aktualizaci vah, které změnu vyvolaly.
- Tyto dva kroky jsou opakovány do chvíle, kdy se ztrátová funkce dostane na nejnižší možnou hodnotu (Algorithmia, 2018).

V některých případech ale nastává problém, který využití optimalizátoru komplikuje. Optimalizátor se může dostat na lokální minimum ztrátové funkce a na něm se „zaseknout“, protože se bude mylně domnívat, že se jedná o celkové minimum. K tomuto problému dochází častěji při práci s datovými sadami, které mají velký počet parametrů. K jeho eliminaci může pomoci nastavení správné rychlosti učení (Algorithmia, 2018).

Příliš velké změny vah mohou zapříčinit nesprávnou minimalizaci ztrátové funkce. Rychlost učení modelu se nastavuje na základě parametru, který se nazývá míra učení (v angličtině learning rate). Hodnota tohoto parametru se obvykle pohybuje okolo jedné tisíciny, nesmí ale být příliš malá, protože i to by mohlo vést k nesprávnému určení minima, a to kvůli již zmíněnému problému s lokálním minimem. Pro lepší představu rozdílnosti míry učení se můžeme podívat na obrázek číslo 2.5, který zobrazuje ztrátovou funkci při malé a a při velké míře učení. Na tomto obrázku je vidět, jakým způsobem určují velké či malé změny vah minimum ztrátové funkce (Algorithmia, 2018).

Obrázek 2.5 – Graf rozdílu mezi vysokou a nízkou mírou učení



(Zdroj: https://algorithmia.com/blog/introduction-to-optimizers)

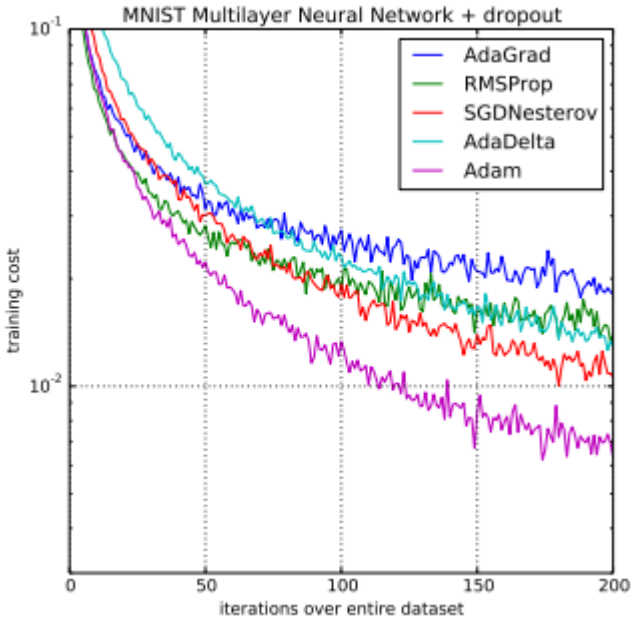
Dalším častým problémem spojeným s učením je takzvaný overfitting. K němu může dojít, pokud se v datové sadě nachází určitý parametr, který dominuje ve vzoru a příliš ovliňuje výsledek. Overfitting poté způsobuje to, že při učení na učících datech dosahuje model přesných predikcí, ale při použití nových dat se jeho přesnost zhorší. Tento problém je řešen procesem regularizace, který je součástí optimalizátoru: ztrátová funkce, která penalizuje model za velké hodnoty vah, je upravena tak, aby ho penalizovala nejen za chybnou predicki, ale také za přiřazení vysoké váhy určitému parametru (Algorithmia, 2018).

Níže je uvedeno několik běžně používaných optimalizátorů vycházejících z principu Gradient Descent:

- Adagrad – Tento typ optimalizátoru nastavuje hodnotu míry učení jednotlivým vahám. Jeho nevýhodou je, že má tendenci postupně snižovat míru učení na příliš nízké hodnoty.
- RMSprop – Jedná se o verzi optimalizátoru Adagrad, jehož chybám se RMSprop snaží předcházet (Algorithmia, 2018).
- Adam – Zkratka Adam vychází ze slovního spojení adaptivní odhadnutí momentu. Adam je velmi rozšířený a vhodný pro práci s datovými sadami s velkým počtem parametrů. Metoda kombinuje přístupy obou výše zmíněných optimalizátorů; také vypočítává jednotlivé rychlosti učení pro různé parametry, činí tak ale na základě odhadů prvního a druhého řádu (Brownlee, 2017).

Na obrázku číslo 2.6 si můžeme prohlédnout příklady křivek několika druhů optimalizátorů, kde na ose x jsou znázorněny jednotlivé interakce cyklů učení na celé datové sadě a na ose y jsou zaneseny výstupní hodnoty ztrátové funkce. Z grafu lze vyčíst, že optimalizátor Adam je obecně oproti ostatním optimalizátorům efektivnější, nemusí tomu tak ale být v každém případě.

Obrázek 2.6 – Graf porovnání efektivnosti jednotlivých optimalizátorů



(Zdroj: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/)

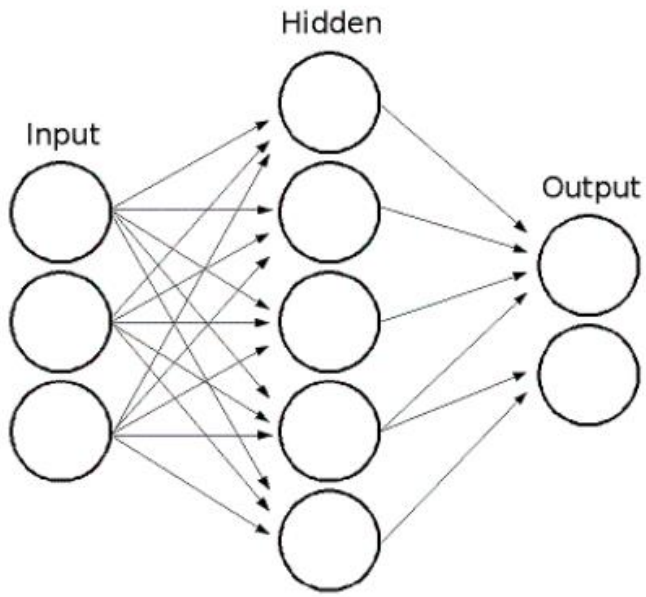
2.7 Neuronové sítě

Tato kapitola je věnována popisu základních druhů hlubokých neuronových sítí, jako je například LSTM, GRU atd., a také je zde představen model SeqToSeq sloužící mimo jiné k vývoji generativních chatbotů.

2.7.1 Multilayer Perceptron

MLP neboli Multilayer Perceptron je takzvaná dopředná síť, která obsahuje jednu vstupní, jednu výstupní a jednu skrytou vrstvu. Ke klasifikaci nelineárních dat používá nelineární aktivační funkce, například logistickou regresi. Síť je plně propojená a její architektura se skrytou vrstvou dala vzniknout základům architektury hlubokého učení. MPL se používá zejména pro rozpoznávání řeči a překládání. Schéma neuronové sítě Multilayer Perceptron s jednou skrytou vrstvou je znázorněno na obrázku číslo 2.7 (Bhagwat, 2018).

Obrázek 2.7 – Schéma neuronové sítě MLP



(Zdroj: Bhagwat, 2018)

2.7.2 Konvoluční neuronová síť (CNN)

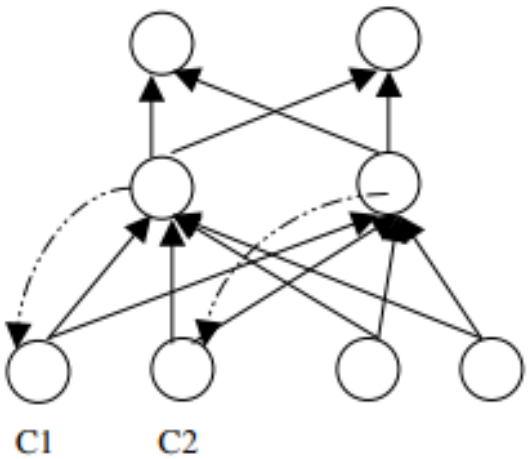
Konvoluční neuronová síť (v angličtině Convolutional Neural Network) je varianta MLP s minimálně jednou vrstvou navíc. Jak už název napovídá, tato vrstva pomáhá snížit komplexnost sítě pomocí aplikování konvoluční funkce na vstup a předání výstupu další vrstvě. Redukce komplexnosti sítě pomocí konvoluční funkce umožňuje, aby byla síť

hlubší a aby dokázala vyhodnocovat složitější data (Young et. al., 2018). Konvoluční neuronové sítě se používají především pro úkoly spjaté s počítačovým viděním. CNN sítě se také používají pro takzvané značkování řeči neboli tagging (tagging slouží ke klasifikaci jednotlivých slov na slovesa, podstatná jména atd.) a pro sémantickou analýzu (Cohen, 2018).

2.7.3 Rekurentní neuronová síť (RNN)

Rekurentní neuronová síť (v angličtině Recurrent neural network) byla navrhnuta pro učení se sekvenčními nebo časovými vzory a přitahovala pozornost zejména v roce 1990. Rekurentní síť pracuje s určitým feedbackem, jedná se tedy o zpětnou architekturu. Co se týče architektury spojení, rekurentní neuronová síť může být částečně propojená, a to i s několikavrstvou dopřednou neuronovou sítí. Příklad neuronové sítě s částečnou propojeností a částečnou dopřednou strukturou si lze prohlédnout na obrázku 2.8 (Medsker & Jain, 2001).

Obrázek 2.8 – Schéma RNN s částečnou propojeností



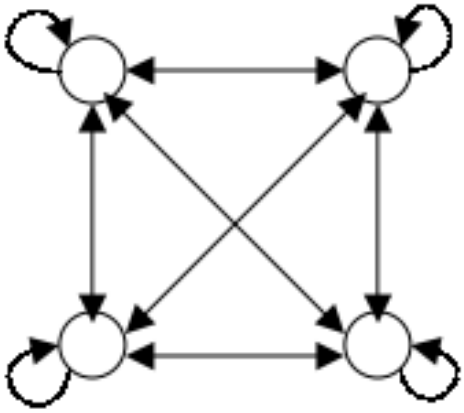
(Zdroj: Medsker & Jain, 2001)

Architektura může být i plně propojená. V takovém případě není rozdíl mezi vstupními uzly. V plně propojené rekurentní neuronové sítí je každý uzel vstupem pro všechny ostatní, jinými slovy uzel poskytuje feedback i sám sobě. Takový případ je znázorněn na obrázku 2.9 (Medsker & Jain, 2001).

Existují dva způsoby přidání možnosti feedbacku do dopředných neuronových sítí. Jednu z těchto metod představil J. L. Elman a její princip je takový, že se jako feedback používá výstup ze skrytých vrstev. Při implementaci tohoto přístupu klade model větší

důraz na posloupnost vstupních hodnot. Druhou metodu představil americký profesor M. I. Jordan. Spočívá v použití feedbacku z výstupu kontextových uzlů vstupní vrstvy. Tato metoda věnuje větší pozornost posloupnosti výstupních hodnot (Medsker & Jain, 2001).

Obrázek 2.9 – Schéma plně propojené sítě RNN

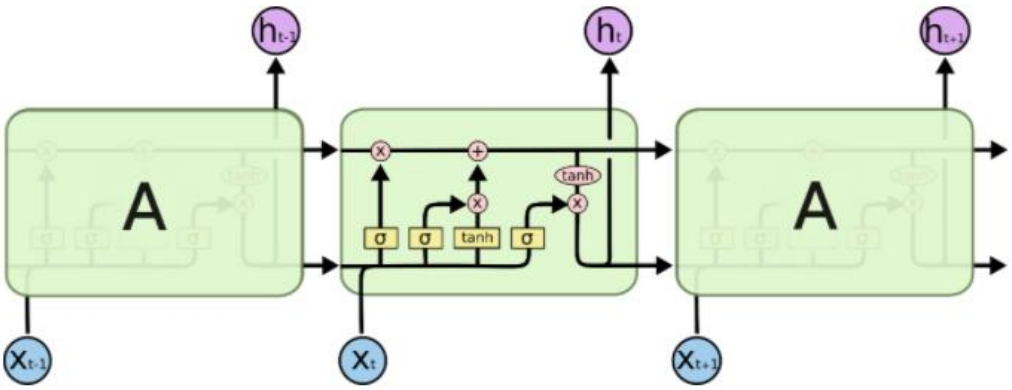


(Zdroj: Medsker & Jain, 2001)

2.7.4 Long Short-Term Memory (LSTM)

Long Short-Term Memory (v češtině dlouhodobá krátkodobá paměť) je speciální druh rekurentní neuronové sítě schopné učit se dlouhodobé závislosti. Byla představena v roce 1997 počítačovými vědci S. Hochreiterem z Technické univerzity v Mnichově a J. Schmidhuberem z IDSIA. Neuronová síť LSTM byla původně navrhnuta proto, aby se s její pomocí předešlo problému s dlouhodobými závislostmi, tedy aby rekurentní neuronová síť dokázala uchovat informace delší dobu (Olah, 2015).

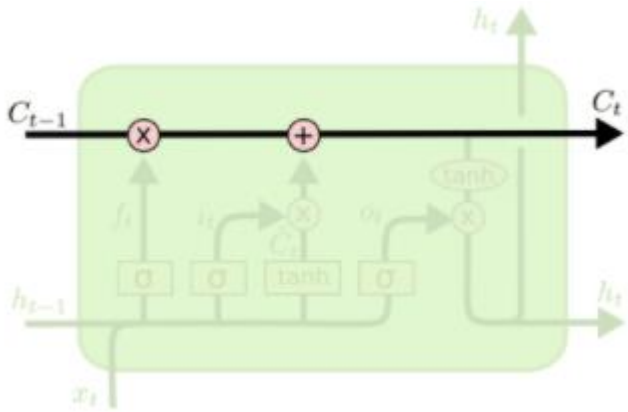
Obrázek 2.10 – Schéma LSTM se zobrazením 4 navzájem komunikujících vrstev.



(Zdroj: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Standartní rekurentní neuronové sítě mají všechny strukturu řetězce, ve kterém se opakují moduly. LSTM má také strukturu řetězce, ale místo jediné neuronové vrstvy má čtyři, které spolu komunikují speciálním způsobem. Tyto vrstvy jsou žlutými obdélníky označeny na obrázku číslo 2.10 (Olah, 2015).

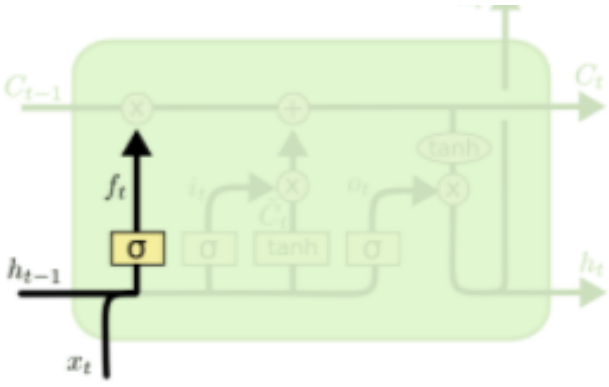
Obrázek 2.11 – Schéma LSTM s vyznačnou stavovou buňkou



(Zdroj: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Jádrem celého LSTM je takzvaná stavová buňka neboli článek. Tato buňka je vidět na obrázku číslo 2.11, kde je znázorněna horizontální přímkou v horní části diagramu. LSTM přidává nebo odebírá informaci stavové buňky pomocí takzvaných bran. Ty jsou z části tvořené vrstvou sigmoidní neurální sítě a z části operací bodového násobení. Sigmoidní vrstva poskytuje výstup v intervalu mezi 1 a 0. Čím vyšší je hodnota výstupu sigmoidní vrstvy, tím více informací bude puštěno skrze stavovou buňku a naopak. Proces fungování modulu LSTM je následující (Olah, 2015):

Obrázek 2.12 – První krok zpracování informace pomocí LSTM



(Zdroj: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

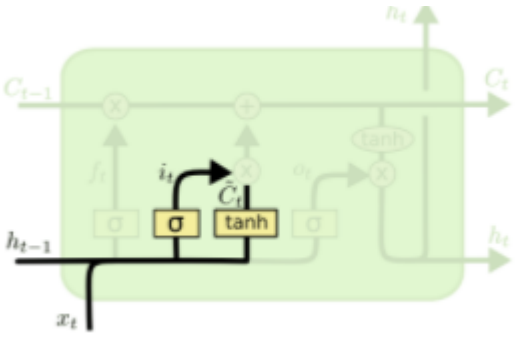
Prvním krokem (viz obrázek číslo 2.12) je rozhodnout pomocí sigmoidní vrstvy (v angličtině se nazývá „forget gate layer“) zda informace, která je nyní uložená ve stavové buňce, má být zahozena. Výsledkem tohoto rozhodnutí je číslo v intervalu mezi 0 a 1, přičemž výsledek této funkce je přímo vynásobený hodnotou stavové buňky. Z toho vyplívá, že pokud je výsledek sigmoidní vrstvy roven 0, hodnota stavové buňky je zcela vymazána. Funkce znázorněná vztahem 2.9 rozhodne na základě těchto dvou vstupů (Olah, 2015):

- x – značí vstupní sekvenci;
- h – značí výstup předchozí skryté vrstvy.

$$f_t = \sigma(W_f * [h_{t-1,} x_t] + b_f) \qquad (2.9)$$

Následně je potřeba rozhodnout, zda bude nová informace uložena ve stavové buňce (viz obrázek číslo 2.13). O toto rozhodnutí se stará brána s názvem „input gate layer“, která se skládá ze dvou částí. První část značená písmenem *C* vytvoří vektor hodnot jako vhodných kandidátů pro změnu stavové buňky (viz rovnice číslo 2.10). Získaný výsledek je vynásobený druhou částí (rovnice číslo 2.11), která je značena písmenem *i* a udává, jak moc chceme stav změnit (Olah, 2015).

Obrázek 2.13 – Druhý krok zpracování informace pomocí LSTM



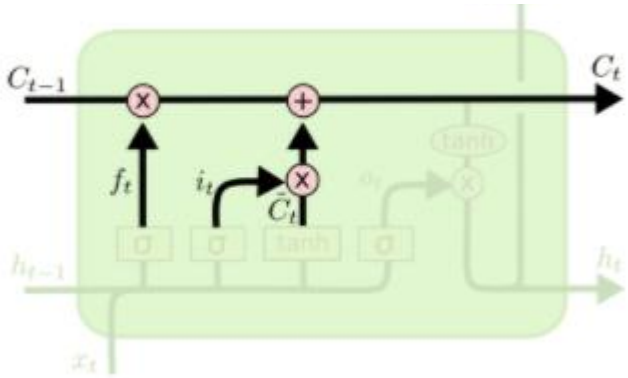
(Zdroj: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

$$\tilde{C}_t = \tanh(Wc * [h_{t-1}, x_t] + b_C) \qquad (2.10)$$

$$i_t = \sigma(Wc * [h_{t-1}, x_t] + b_i) \qquad (2.11)$$

Výsledek této brány je součin $i * C$, který je poté přičten k předešlé hodnotě stavové buňky. Tím je získána finální hodnota statové buňky, která je zobrazena na následujícím obrázku číslo 2.14 (Olah, 2015).

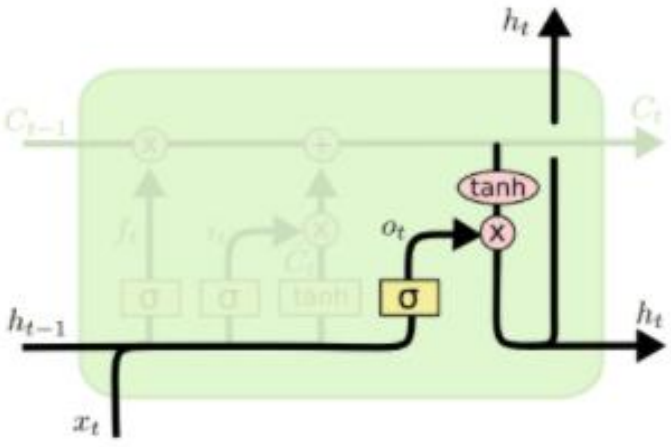
Obrázek 2.15 – Výsledný proces upravení stavové buňky v LSTM



(Zdroj: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Třetím a posledním krokem je samotný výstup skryté vrstvy. Ta je také tvořena dvěma částmi. První částí je opět sigmoidní funkce, která určí, jaká část buňky bude výstupem (vztah 2.12). Následně se pomocí funkce *tanh* upraví výstup stavové buňky tak, aby byla jeho hodnota v intervalu mezi -1 a 1. Finální výstup zachycený vztahem 2.13 vznikne vynásobením výstupu stavové buňky výstupem sigmoidní funkce. Tato poslední operace je znázorněna na obrázku 2.15 (Olah, 2015).

Obrázek 2.14 - Vytvoření výstupu LSTM



(Zdroj: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

$$o_t = \sigma(W_0 * [h_{t-1}, x_t] + b_0) \quad (2.12)$$

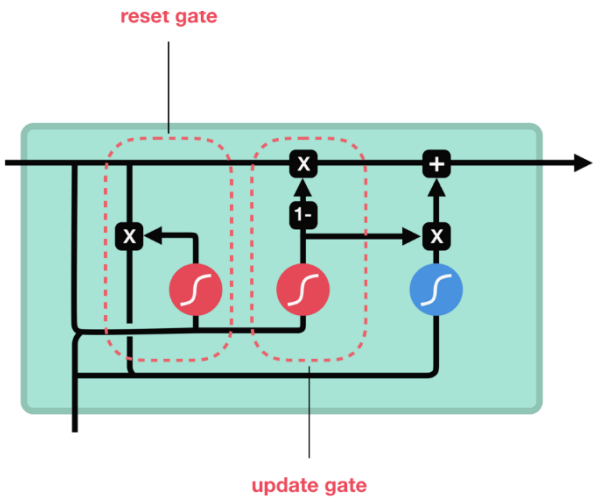
$$h_t = o_t * tanh(C_t) \quad (2.13)$$

2.7.5 Gated Recurrent Unit (GRU)

GRU neboli Gated Recurrent Unit je nejnovější generací rekurentních neuronových sítí. Funkčností se dá připodobnit k rekurentní neuronové sítí LSTM s tím rozdílem, že místo stavu neuronové buňky se používá skrytý stav pro přenos informací. Schéma GRU buňky si lze prohlédnout na obrázku číslo 2.16, kde je vidět, že buňka GRU používá dvě brány – tak, jako je tomu u LSTM. Brány použité v GRU jsou následující:

- Update gate – funguje podobně jako brána forgate v LSTM, tedy rozhoduje, které informace zůstanou uložené a které se zahodí.
- Reset gate – rozhoduje o tom, jaký objem minulých informací zahodit (Phi, 2018).

Obrázek 2.16 – Schéma struktury neuronové buňky GRU



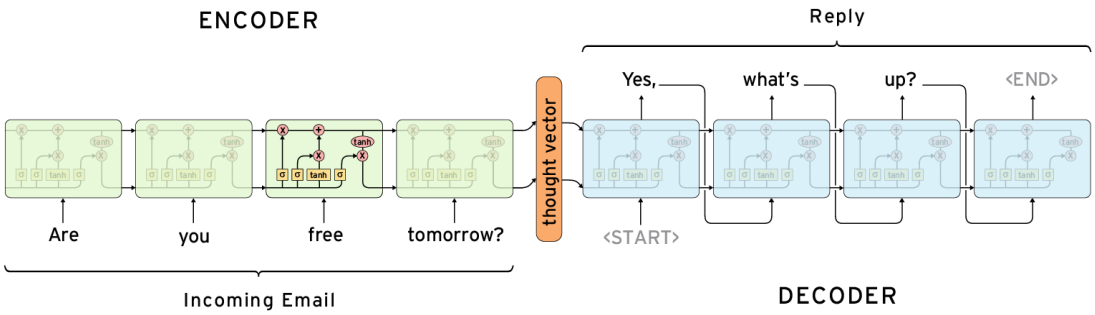
(Zdroj: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21)

2.7.6 Sequence to Sequence Model (Seq2Seq)

Sequence to Sequence je založený na principu architektury rekurentních neuronových sítí. Obsahuje dvě RNN, přičemž jedna slouží jako kodér a druhá jako dekodér. Kodér zpracuje vstupní sekvenci $x = (x_1, \dots, x_m)$ obsahující m elementů a jeho výstupem je reprezentace určitého stavu pomocí vektoru $z = (z_1, \dots, z_m)$. Tento vektor z je vstupním prvkem dekodéru, který následně vygeneruje vstupní sekvenci $y = (y_1, \dots, y_m)$. Pro lepší představu o fungování Seq2Seq je níže přiložen obrázek číslo 2.17, na kterém je zachyceno jeho schéma. Pro tento model se hojně používá rekurentní neuronová síť LSTM, která pomocí metody bran dokáže uchovat určité

informace z předcházejícího kroku (Csáky, 2017). O LSTM pojednává podkapitola 2.3.4 Long-Short Term Memory.

Obrázek 2.17 – Schéma modelu Seq2Seq



(Zdroj: https://github.com/pranoyr/seq-to-seq)

2.8 Zpracování jazyka

V této kapitole jsou přiblíženy metody, které se používají ke zpracování textu neboli jazyka a k reprezentaci slov. Jejich použití většinou předchází samotnému učení modelu chatbota.

2.8.1 Bag Of Words

Bag Of Words (zkráceně BoW) je jeden z nejznámějších modelů pro reprezentaci slov. Jeho použití je velmi jednoduché a flexibilní. Je často voleným prostředkem nejen k reprezentaci slov ale například i ke klasifikaci obrázků.

Jeho princip je založen na popisu výskytu slov v textu. Správné fungování BoW zajišťují dvě části. Nezbytný je slovník, který obsahuje všechna známá slova (resp. ta, která jsou použita v daném textu). Druhou položkou je míra přítomnosti známých slov. Právě proto se modelu říká „bag“ neboli pytel; informace o pořadí nebo struktuře jednotlivých slov v textu není relevantní. Bag Of Words se zabývá pouze tím, zda se slovo v textu vyskytuje nebo ne. Příklad takového modelu je k nahlédnutí na obrázku číslo 2.18, kde jsou v první řádce tabulky v abecedním pořadí uvedena veškerá slova v jeho slovníku. Další řádky tabulky představují jednotlivé věty převedené do modelu Bag Of Words. V případě prvního řádku se tedy jedná o větu *Hello, how are you?* V řádku je uveden počet jednotlivých slov v každé větě. Jak již bylo avizováno výše, informace o pořadí slov není neuchovávána, na základě tohoto modelu již tedy není možné zpětně rekonstruovat větu. Nevýhodou je, že při jeho použití na obsáhlý text může

slovník (a tedy i tabulka) dosahovat opravdu velkých rozměrů. Pro zmenšení velikosti slovníku se používají určité postupy, které slouží k vyčištění textu. Níže jsou uvedeny některé z nich (Brownlee, 2017):

- Ignorování velikosti písmen (to znamená převedení všech slov na malá písmena);
- Ignorování znaků jako jsou čárky, vykřičníky, otazníky atd.;
- Ignorování tzv. stop words. Jedná se o výrazy, která se sice používají často, ale nenesou velké množství informací (příkladem českých stop words může být následující: a, aby, do, ho, i, je, ke, ku atp.);
- Stemming slov, tzn. zkrácení původního slova na pouhý kořen pomocí určitého algoritmu – například ze slova *skříňka* se do slovníku uloží pouze slovo *skříň* (Brownlee, 2017).

Obrázek 2.18 – Tabulka modelu Bag of Words

	are	call	from	hello	home	how	me	money	now	tomorrow	win	you
0	1	0	0	1	0	1	0	0	0	0	0	1
1	0	0	1	0	1	0	0	1	0	0	2	0
2	0	1	0	0	0	0	1	0	1	0	0	0
3	0	1	0	1	0	0	0	0	0	1	0	1

(Zdroj: https://user-images.githubusercontent.com/14823445/38343939-c03c0b18-3854-11e8-9fb8-7b4ea8be01cb.png)

2.8.2 Word Embedding

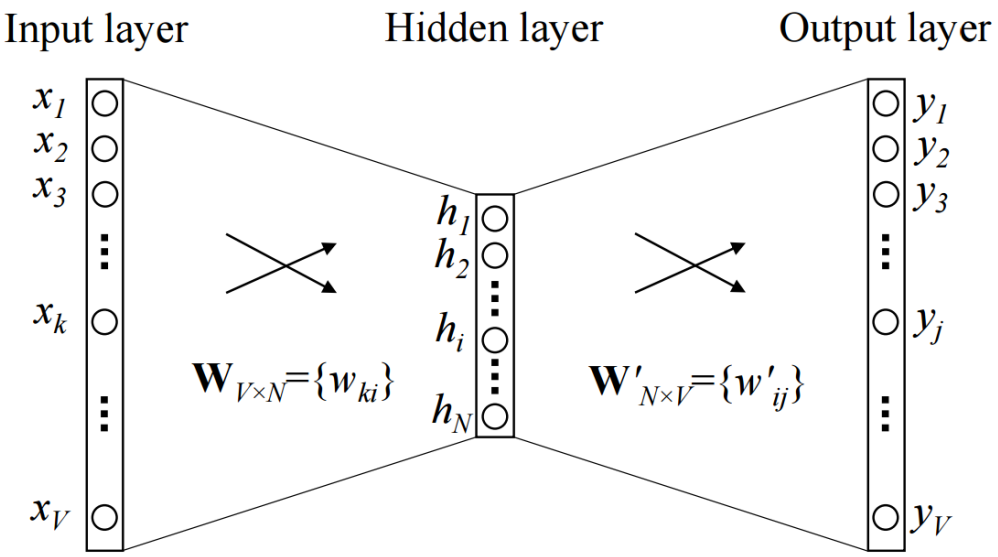
Termín Word Embedding lze volně přeložit jako reprezentace slov. Patří k nejčastěji používaným postupům k převedení slov na vektor s využitím jejich kontextu ve větě. Tento proces reprezentace slov hraje velkou roli při zpracování přirozeného jazyka a také při učení hlubokých neuronových sítí. Oproti tradičním přístupům, jako je například Bag-Of-Words, který je popsán v předchozí kapitole, disponuje tato metoda vylepšením. Word Embedding totiž dokáže narozdíl od tradičních přístupů zachytit sémantickou a syntaktickou informaci o slovech (Y. Liu, Z. Liu, Chua, Sun, 2015).

Nejpoužívanějšími modely sloužícími k reprezentaci slov při strojovém učení jsou Word2Vec a GloVe. Navzájem se od sebe se liší především způsobem učení. Model GloVe funguje na principu využití globálního počtu výskytů každého slova na základě všech textů datové sady. V praxi to znamená, že nejprve vytvoří velkou matici informací

a spočítá výskyt každého slova (řádky matice) v určitém kontextu (sloupce matice). Matice může dosahovat obrovských rozměrů, protože počet kontextů má v zásadě kombinatorickou velikost (MLNerds, 2019).

Model Word2Vec využívá naopak výskyt v lokálních kontextech. Jednodušeji řečeno, jsou pro něj stěžejní pouze sousední slova. Využívá vlastní neuronovou síť, která vyhledává reprezentaci slov. K učení těchto neuronových sítí se nejčastěji používají dvě metody, a to Skip-Gram a CBOW. Obě budou vysvětleny v následujících odstavcích (MLNerds, 2019).

Obrázek 2.19 – Architektura neuronové sítě metody CBOW

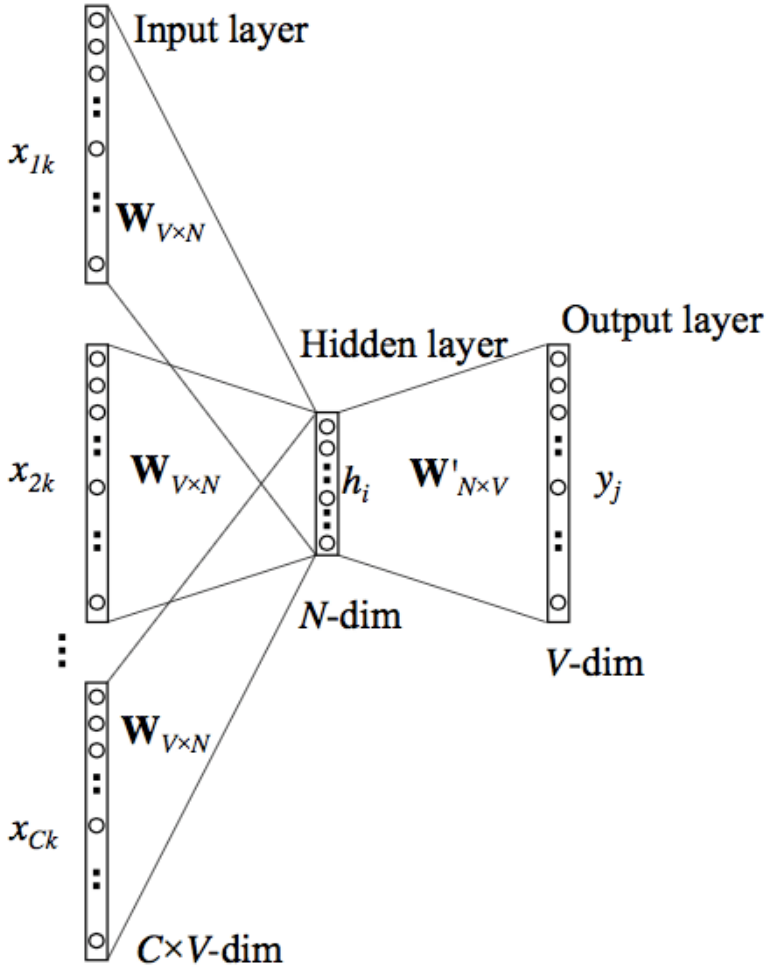


(Zdroj https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa)

CBOW je zkratka názvu metody Continuous Bag Of Words, která jako vstup do neuronové sítě používá kontext každého slova a snaží se předpovědět slovo následující vzhledem ke kontextu. Pro lepší představu o fungování této metody bude použita modelová věta „*Zítra bude pršet*“. Vstupem pro neuronovou síť bude kontext slova *bude*. Nyní je žádoucí, aby neuronová síť správně predikovala slovo *pršet*. Pro tento účel použijeme metodu one-hot-encoding na vstupní slovo *bude* a změříme výstupní chybu, kterou porovnáme s metodou one-hot-encoding použitou na cílové slovo *pršet*. Metoda one-hot-encoding vychází z reprezentace slov pomocí Bag-Of-Words. Architekturu metody CBOW si můžeme prohlédnout na obrázku číslo 2.19 (Karani, 2018).

Jak je zřejmé z obrázku, vstupem pro neuronovou síť je kontext pouze jednoho slova o velikosti *V*. Tento vektor vstupní vrstvy je převeden podle počtu neuronů *N* skryté vrstvy pomocí matice vah W_{nxm}. Následně je výstup skryté vrstvy neuronové sítě převeden na finální výstup neuronové sítě pomocí matice vah W'_{nxv}. Jak již bylo zmíněno, v případě obrázku číslo 2.19 je vstupem kontext pouze jednoho slova. Pro získání představy o tom, jak by vypadala architektura neuronové sítě, kde je vstupem kontext více slov, se můžeme podívat na obrázek číslo 2.20. Princip metody nicméně zůstává stále stejný (Karani, 2018).

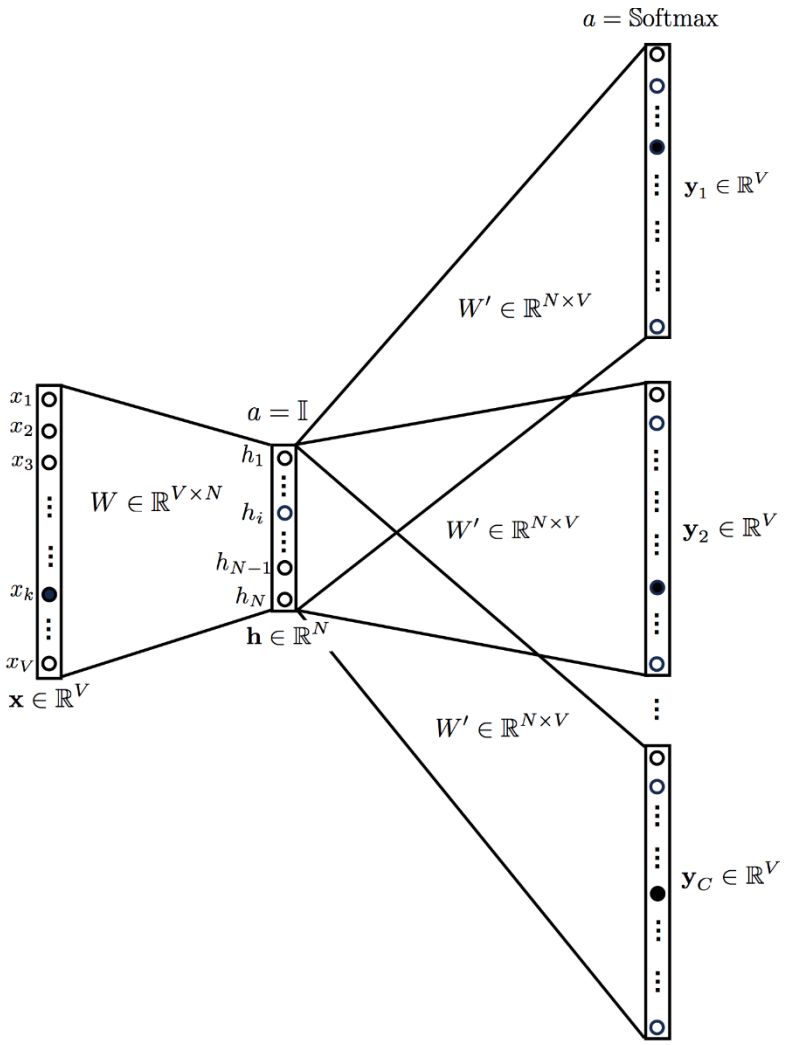
Obrázek 2.20 – Architektura neuronové sítě metody CBOW za využití kontexut více slov



(Zdroj https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa)

Metoda **Skip-Gram** funguje obdobným způsobem, s tím rozdílem, že neuronová síť je učena naopak. Vstupem do neuronové sítě je v tomto případě cílové slovo *pršet* a metoda Skip-Gram se snaží predikovat kontext tohoto slova. Architektura této neuronové sítě je znázorněna na obrázku číslo 2.21. Lze vidět, že vektor cílového slova je opět převeden podle počtu neuronů skryté vrstvy a že výstupem neuronové sítě je poté pravděpodobnostní rozdělení. Pro každý kontext je výstupem neuronové sítě pravděpodobnostní rozdělení, které obsahuje počet pravděpodobností *V* pro každé slovo (Karani, 2018).

Obrázek 2.21 – Architektura neuronové sítě pro metody Skip-Gram



(Zdroj https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa)

2.8.3 Padding

Texty, které jsou použity jako vstup pro neuronovou síť, jsou převedeny na sekvence nezáporných čísel o různé délce. Neuronová síť LSTM nebo například konvoluční neuronová síť nicméně přijímají pouze vstupy stejné délky a rozměru. Z toho vyplívá, že původní sekvence představující text je potřeba upravit tak, aby měly stejnou délku. Pro tuto úpravu se používá právě metoda padding která doplní původní sekvenci nulami na základně maximální délky sekvence. Způsob, kterým je padding použit, může ovlivnit fungování sítě a přesnost predikce (Dwarampudi & Suba Reddy, 2019).

K problému maximální délky sekvence je možné přistoupit dvěma způsoby. Maximální délku sekvence je možné nastavit dle vlastního uvážení. Je potřeba pamatovat na to, že sekvence by neměla být zbytečně dlouhá, neboť příliš velký počet nul vyskytujících se ve vstupní sekvenci může snižovat výkonost a přesnost modelu. Druhou možností je určení maximální délky podle nejdelší vstupní sekvence. I v tomto případě je nutné věnovat pozor možnému úskalí; vstup přesahující velikost maximální vstupní sekvence nebude správně zpracován.

Samotné doplnění nul může proběhnout dvěma způsoby, a to:

- Post-Padding – Nuly se doplní za poslední číslo sekvence až do její maximální délky.
- Pre-Padding – Počet nul, který je dán rozdílem mezi maximální délkou sekvence a délkou vstupní sekvence, se doplní před vstupní sekvenci (Dwarampudi & Suba Reddy, 2019).

Ze studie Manipalského Institutu Technologie vyplívá, že při doplnění nul před vstupní sekvencí je přesnost modelu větší než při doplnění nul za vstupní sekvenci. Tento závěr je v publikaci doplněn hypotézou, která se opírá o způsob komunikace mezi lidmi. Tato hypotéza říká, že pokud spolu hovoří dvě osoby, osoba 1 předává informace osobě 2. V takovém případě je pro osobu 2 efektivnějším způsobem komunikace, pokud osoba 1 čeká po určitý čas a následně předává informaci. Osoba 2 může ihned po získání informace odpovědět. Zároveň je při tomto způsobu komunikace větší pravděpodobnost, že si osoba 2 pamatuje více informací, které získala od osoby 1, a že na ně odpoví s větší přesností, než kdyby musela vyčkávat a pamatovat si po dobu čekání vstupní informace (Dwarampudi & Suba Reddy, 2019).

2.9 Softwarové možnosti

V dnešní době lze při vyvíjení jakéhokoliv softwaru využít již vytvořených modulů, frameworků atd. Aplikace jako je chatbot s využitím umělé inteligence není výjimkou. I vytvoření umělé inteligence od nuly je cestou, kterou se lze vydat, i když ne snadnou. V následujících podkapitolách budou představeny jazyky a nástroje vhodné pro práci s hlubokými neuronovými sítěmi.

2.9.1 Jazyky pro vývoj DNN

Při výběru programovacího jazyka pro vývoj umělé inteligence nebo aplikace s umělou inteligencí jsou možnosti téměř neomezené. Některé programovací jazyky jsou ovšem pro tento druh vývoje vhodnější než jiné. Nejčastěji se používá jazyk Python, který své obliby dosáhl hlavně proto, že není obtížný na naučení, je jednoduše čitelný, má bohatou základnu knihoven pro různé účely a podporuje většinu nejpoužívanějších frameworků pro práci s umělou inteligencí. Dalšími známějšími jazyky, se kterými se můžeme často setkat, jsou například Javascript, Java, C/C++.

Za zmínku určitě stojí i jazyky, které se často používají při vývoji umělé inteligence, ale ne při vývoji standartních aplikací. Tyto jazyky často slouží k řešení velmi specifických problémů. Jako první příklad si uvedeme jazyk R, který se používá především ke zpracování statistické analýzy. Podobným případem může být jazyk Matlab, který slouží primárně k provádění pokročilých matematických výpočtů. Zvláštní příklad je jazyk Scala, který funguje na bázi Java Virtual Machine. Usnadňuje práci s jazykem Java a je velmi efektivním nástrojem pro zpracování velkého množství dat. Posledním příkladem je Julia. Jedná se o jazyk používaný pro dynamické programování, který je nicméně vhodný i pro složité numerické operace (Costa, 2020).

2.9.2 TensorFlow

TensorFlow je framework vyvinutý společností Google a je jedním ze dvou nejpoužívanějších frameworků pro práci s hlubokými neuronovými sítěmi. Poskytuje širokou škálu funkcí od předzpracování až po modelování dat. K jeho napsání byly použity jazyky Python, C++ a CUDA. Spolupracuje téměř se všemi operačními systémy a podporuje zpracování procesů jak pomocí procesoru, tak grafické karty. Tento framework používají velké firmy jako je Google, Uber, 9GAG nebo Deep Speech (Bhattacharyya, 2020).

2.9.3 PyTorch

PyTorch byl vyvinut společností Facebook a je druhým nejpoužívanějším frameworkem pro práci s hlubokými neuronovými sítěmi. Kvůli jeho podobnosti s TensorFlow se jedná o dva hlavní konkurenty v tomto odvětví. Aktivní vývojářská komunita se zabývá především počítačovým viděním a zpracováním přirozeného jazyka. Použit byl ale také například k vývoji Tesla Autopilot. Společnosti, které používají tento framework jsou například NVIDIA, Apple a Ford (Bhattacharyya, 2020).

2.9.4 Keras

Keras je knihovna pro práci s neuronovými sítěmi napsaná v Pythonu. Je vystavěna na frameworku TensorFlow a velmi často se používá při nasazení neuronových sítí do webového rozhraní nebo do mobilních aplikací. Keras je oblíbený pro rychlost výpočtů a pro uživatelskou přívětivost. Používají ho společnosti jako je CERN, NASA nebo Ford Motor Company (Bhattacharyya, 2020).

3 Úvod do problematiky praktické části

V této kapitole je představen problém, který bude následně řešen v dalších kapitolách praktické části této práce.

3.1.1 Zaměření chatbota

S ohledem na aktuální situaci, která vznikla v důsledku pandemie COVID-19, se chatbot popsaný v následujících kapitolách zaměřuje na zodpovídání otázek týkajících se covidové situace. Lidé k ní nemají dostatek pravdivých a relevantních informací a tento chatbot by mohl přispět k vyřešení tohoto problému.

Bohužel v současné době není k dispozici žádná volně dostupná datová sada týkající se přímo otázek o pandemii COVID-19 v české jazyce. Model tohoto chatbota je proto naučen otázkám pomocí datové sady, která je v jazyce anglickém.

3.1.2 Cíl praktické části

Cílem praktické části této diplomové práce je vyvinutí chatbota, který bude splňovat níže popsané požadavky.

Jelikož se jedná o typ chatbota, který se nazývá Retrieval-Based, bude sloužit k odpovídání na otázky položené uživatelem. Z datové sady vyplývá, že naučený model chatbota by měl být schopen rozpoznat jeden ze 62 záměrů otázek s minimální přesností 70 %. Pokud neporozumí otázce, měl by na to upozornit uživatele, což zabrání předání chybné informace. Pokud záměr otázky určí správně, uživatel obdrží odpověď z předem připravené databáze. Výsledný model by také měl být použitelný v reálném prostředí. K simulaci tohoto cíle poslouží jednoduchá aplikace nasazená v lokálním prostředí.

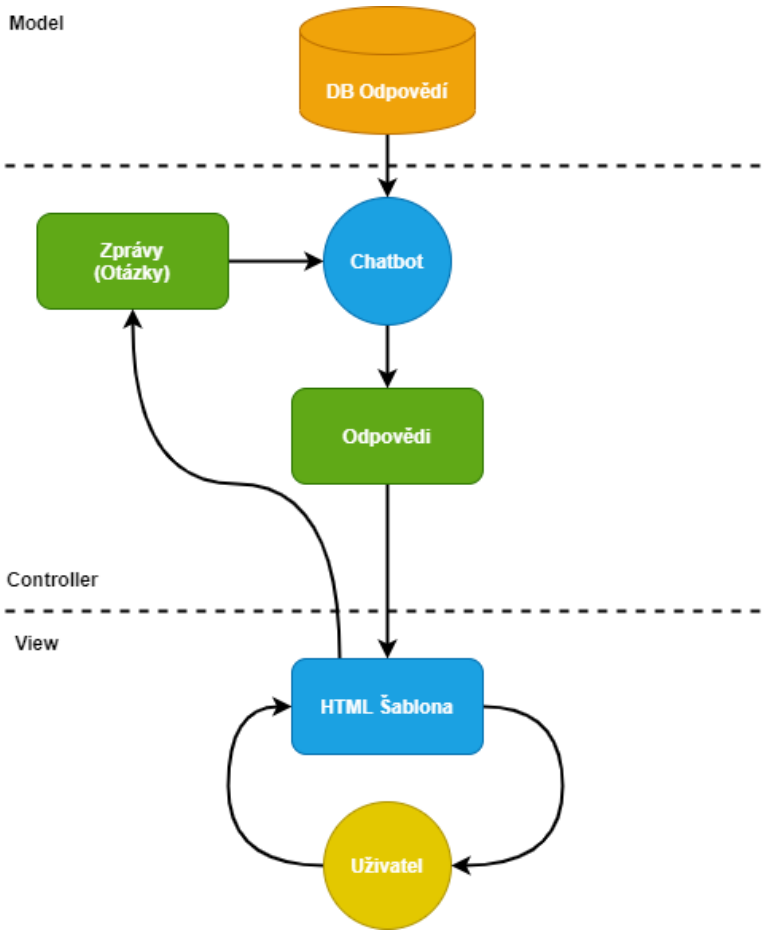
4 Softwarová architektura

Níže je představena a detailně popsána architektura chatbota a jeho modulů, které jsou potřebné pro správné fungování aplikace jako celku.

4.1 Rozvržení chatbota

Samotný chatbot se skládá z několik částí a pro jeho úspěšné použití v praxi je potřeba několika dalších komponentů. Pro zjednodušení je možné na chatbota nahlížet jako na aplikaci s třívrstvou architekturou, která obsahuje Model, Controller a View. První vrstva (View) obsahuje uživatelské rozhraní sloužící k vytvoření zprávy a k její jednoduché validaci. Zpráva, v tomto případě otázka, může být definována jako řetězec znaků odeslaný frontendem aplikace, tzn. uživatelem. Zprávy je za účelem využití zpětné vazby k postupnému vylepšování chatbota možné ukládat.

Obrázek 4.1 – Architektura aplikace chatbota



(Zdroj: Autor)

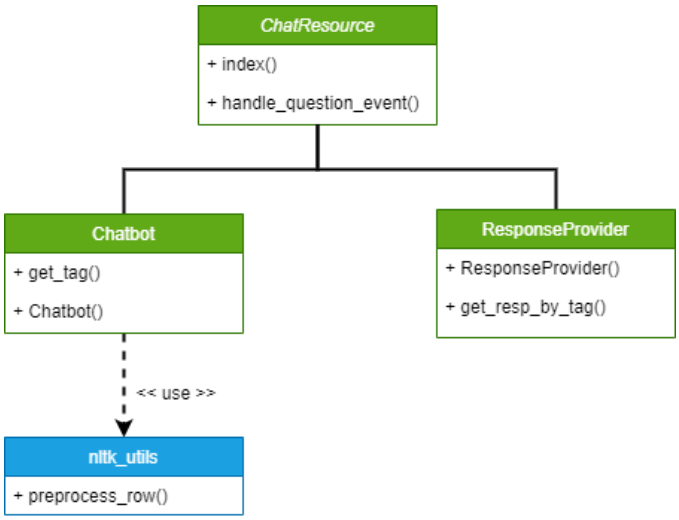
Další krok se odehrává ve vrstvě Controlleru. Jedná se o samotné zpracování zprávy (neboli otázky) chatbotem a její vyhodnocení. Po vyhodnocení se provolá databáze, ze které je vybrána vhodná odpověď. Tato odpověď se následně zašle pomocí backendu aplikace na její frontend, kde se zobrazí uživateli, který má poté možnost položit další otázku.

Na diagramu je úložiště odpovědí znázorněno databází, která může mít téměř jakoukoliv formu. Také odpovědi mohou být uloženy pomocí různých způsobů, například jako soubor JSON nebo CSV.

4.2 Diagram tříd

Nyní si představíme části aplikace chatbota a způsob komunikace mezi nimi. Následující diagram zobrazuje moduly použité v aplikaci.

Obrázek 4.2 – Diagram tříd aplikace chatbota



(Zdroj: Autor)

První třída ChatResource obsahuje jednoduché Rest Api, které umožňuje komunikaci mezi backendem a šablonou frontendu sloužící k zobrazení uživatelského rozhraní. Komunikace mezi HTML šablonou frontendu probíhá za využití JSON a JavaScriptu. Třída ChatResouce obsahuje třídy ResponseProvider a Chatbot, pomocí nichž zajišťuje fungování aplikace.

Třída Chatbot je vybavena funkcí, která pomocí naučeného modelu rozpoznává záměr otázky týkající se Covid-19. Pokud chatbot rozpozná záměr otázky s pravděpodobností

nižší než 75 %, označí ho za neznámý; tím se minimalizují omyly v odpovědích. Vstupem funkce, která zajišťuje predikci, je text předzpracovaný třídou NltkUtils.

Poslední třídou je ResponseProvider, která je napojena na databázi odpovědí nebo na soubor obsahující odpovědi. Na základě vyhodnoceného záměru otázky získává z databáze nebo ze souboru vhodnou odpověď.

5 Softwarová implementace

Tato kapitola je zaměřena na detailní zobrazení vývoje aplikace Chatbota. Také se zabývá implementací skriptu použitého pro předzpracování dat a učením samotného modelu.

5.1 Datová sada

K naučení chatbota byla použita datová sada, která je k nahlédnutí buď v příloze nebo na portálu GitHub (https://github.com/botxo/corona_dataset). Datová sada je ve formátu CSV s následujícími atributy:

- Language – Jazyk použitý v otázce;
- Domain – Téma, kterého se otázka týká;
- Intent – Detailnější téma otázky;
- Industry – Nadbytečný parametr, který obsahuje pouze jednu hodnotu;
- Text – Text samotné otázky tázajícího;
- Want To – Přímá informace, kterou chce tázající získat.

Samotný příklad jednoho záznamu vypadá takto: *English Coronavirus Recommendations, Pregnant, Coronavirus, What should a pregnant woman do to stay safe from the Coronavirus?, know what to do if you're pregnant.*

Datová sada čítá přibližně 1000 záznamů. Některé z nich nebylo možné použít kvůli chybějícím hodnotám u některých atributů. Velikost datové sady není optimální, obzvláště pro chatbota, který má poskytovat informace o tak závažném problému, jako je covidová pandemie. V době zpracování této diplomové práce bohužel nebyly veřejně dostupné žádné obsáhlejší datové sady, které by jistě zlepšily přesnost vyhodnocování chatbota.

5.2 Učící skript

Výběr knihoven a frameworků záleží na způsobu jejich využití a na preferenci programátora. Pro naučení tohoto modelu byly využity tyto knihovny a frameworky: Pandas, Sklearn, Tensorflow, Pickle, matplotlib.

Prvním krokem je načtení dat z datové sady (tento krok je zobrazen na ukázce kódu číslo 5.1). Na řádku 21 lze vidět načtení dat do třídy DataFrame. Tato třída je poskytována knihovnou Pandas a umožňuje jednoduchou manipulaci s daty. Následně je DataFrame pomocí funkce *dropna* zbaven řádků s nevyplněnými atributy. Na závěr dojde k upravení každého řádku třídou NltkUtils, která edituje text samotného uživatelského vstupu. Detailnější charakteristika třídy NltkUtils je v kapitole 5.4.

Ukázka kódu 5.1 – Načtení datové sady

```
19  From Dataframe get unique tags and drop rows with Nan values in any field of table.
20  """
21  df = pd.read_csv('corona_dataset.csv')
22  # Remove rows witn NaN values.
23  df = df.dropna(axis='rows', how='any')
24  # Get list of unique Tags.
25  tags = df['Tag'].unique()
```

(Zdroj: Autor)

Z ukázky kódu číslo 5.2 je zřejmé, že dalším krokem je převedení textu otázky na sekvenci nezáporných čísel. K tomuto účelu slouží třída Tokenizer poskytovaná knihovnou Tensorflow. Konstruktor této třídy má jako parametr počet unikátních slov, které chceme brát v potaz při převádění textu na sekvenci. Dále se pomocí funkce *fit_on_texts()* aktualizuje třída Tokenizer, neboli její interní slovník na základně předložených dat. V poslední části tohoto kódu je převeden text otázky na sekvenci nezáporných čísel představujících jednotlivá slova v aktualizovaném slovníku.

Ukázka kódu 5.2 – Nastavení třídy Tokenizer a převedení textu na sekvenci nezáporných čísel

```
31  Create tokenizer.
32  With tokenizer create tokenized vectors from dataset senteces.
33  """
34  tokenizer = Tokenizer(10000)
35  # Updates dictionary based on preprocessed dataset
36  tokenizer.fit_on_texts(df['Text'])
37  train = tokenizer.texts_to_sequences(df['Text'])
```

(Zdroj: Autor)

Na řadu přichází použití metody padding na vstupní sekvence čísla (více o metodě padding v podkapitole číslo 2.8.3). Tuto funkci poskytuje knihovna Tensorflow a její

implementace je zobrazena v ukázce kódu číslo 5.3. V tomto skriptu byla použita metoda pre-padding, která je nastavena jako defaultní hodnota argumentu funkce *pad_sequence()*. Taktéž bylo použito defaultní nastavení pro maximální délku sekvence, která je nastavena podle nejdelší vstupní sekvence v použité datové sadě. Její délka je 24; tato hodnota je pro uživatelský vstup vzhledem k velikosti samotné datové sady dostačující.

Ukázka kódu 5.3 – Použití metody padding na datovou sadu

```
39  """
40  Padding sequences.
41  Pads sequences to the same length.
42  """
43  x_train = pad_sequences(train)
```

(Zdroj: Autor)

5.2.1 Label Encoder

Pro správné naučení a fungování modelu je potřeba převést každý tag z datasetu na číselnou hodnotu. K tomuto účelu slouží třída LabelEncoder, kterou poskytuje knihovna Sklearn. Pomocí funkce *fit_transform()* převede celý sloupec datové sady obsahující tagy otázek na pole, které obsahuje pouze čísla. Získaná čísla poté odkazují na jednotlivé tagy. Tuto funkci si lze prohlédnout na ukázce kódu číslo 5.4.

Ukázka kódu 5.4 – Použití třídy LabelEncoder

```
45  """
46  Encode lables.
47  Transform lables/tags into numerical values.
48  """
49  lable_encoder = LabelEncoder()
50  y_train = lable_encoder.fit_transform(df['Tag'])
```

(Zdroj: Autor)

5.2.2 Hyper-parameters

Ještě před vytvořením neuronové sítě je potřeba definovat několik parametrů. Prvním parametrem je velikost slovníku, tedy počet slov v třídě Tokenizer. Následně je nutné definovat délku výstupu samotného modelu. Ta odpovídá počtu unikátních tagů otázek v datové sadě. Posledním parametrem je tvar vstupu; tato hodnota odpovídá maximální

délce sekvence. Všechny tyto hodnoty budou použity při vytváření modelu, jemuž je věnována následující kapitola.

Ukázka kódu 5.5 – Nastavení Hyper-parameters

```
"""
Hyperparameters
"""
vocabulary = len(tokenizer.word_index)
print("Number of unique words in vocabulary: ", vocabulary)
output_length = lable_encoder.classes_.shape[0]
print("Output lenght: ", output_length)
input_shape = x_train.shape[1]
print("Input shape:", input_shape)
```

(Zdroj: Autor)

5.2.3 Model

Další částí postupu je vytvoření modelu a definování všech jeho vrstev pomocí knihovny Tensorflow. K vytvoření vstupní vrstvy je použita třída Input. Ta jako parametr přijímá tvar vstupu, který byl popsán v kapitole výše a který odpovídá maximální délce vstupní sekvence. Další vrstva je tvořena třídou Embedding, jež poskytuje možnost Word Embedding neboli umožňuje převedení jednotlivých sekvencí slov reprezentovaných čísly na vektory s využitím kontextu slov (viz podkapitola 2.8.2).

Další vrstvou je samotné LSTM, které je vytvořeno pomocí stejnojmenné třídy. Jeho fungování je popsáno v podkapitole 2.7.4. Následující vrstvou je třída Flatten, která umožňuje převést několikadimenzionální výstup (matici) z LSTM do jednodimenzionální matice neboli pole.

Následující vrstvu tvoří třída Dense, která představuje plně propojenou neuronovou síť. Pomocí parametru aktivační funkce je vstup do této neuronové vrstvy převeden na výstup, který je normalizován na distribuci pravděpodobnosti pro predikovanou výstupní třídu.

Dalším krokem je použití třídy Model, která ze všech vrstev vytvoří samotný model. Následuje funkce *compile()*, která ho nastaví a připraví k učení pomocí následujících parametrů:

- Loss function neboli ztrátová funkce – Slouží k vypočítání hodnoty, kterou by měl model během tréninku minimalizovat. Detailněji se jí věnuje podkapitola 2.6.1. V tomto případě byla vybrána ztrátová funkce využívající křížovou entropii, a to

funkce *Sprase categorial crossentropy*, která vypočítává ztrátu křížové entropie mezi tagy a jejich predikcemi.

- Optimalizátor – Teorie optimalizátoru je popsána v podkapitole 2.6.3.V tomto modelu byl použitý optimalizátor nazývající se *Adam*.
- Metriky – Tyto funkce jsou podobné funkcím ztrátovým, nepoužívají se ale při samotném tréninku nýbrž pouze pro vizualizaci výsledků. Jako metriku lze použít jakoukoliv ztrátovou funkci.

Posledním krokem je samotný trénink. Zahájí se funkcí *fit()*, která je poskytována třídou Model. Zde se jako parametry předávají již upravená data a počet cyklů epoch. Počet cyklů epoch říká, kolikrát bude učící algoritmus procházet celou datovou sadu. Výsledné nastavení modelu si můžeme prohlédnout na ukázce kódu číslo 5.6. Na obrázku jsou také vidět vrstvy Dropout nebo nastavení regularizátoru. Tyto části slouží k optimalizaci a jejich funkce bude detailněji pospána v kapitole 6.3 Experimenty.

Ukázka kódu 5.6 – Výsledné nastavení modelu

```
Create learning model.
"""
i = Input(shape=(input_shape,))
x = Embedding(vocabulary + 1, 10)(i)
x = LSTM(10, return_sequences=True, activity_regularizer=regularizers.L2(0.001))(x)
x = Dropout(0.5)(x)
x = Flatten()(x)
x = Dense(output_length, activation="softmax")(x)
model = Model(i, x)

model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=['accuracy'])

history = model.fit(x_train, y_train, epochs=200, validation_split=0.1)
```

(Zdroj: Autor)

Nakonec je potřeba naučený model uložit, aby mohl později sloužit k výstavbě aplikace chatbota. Nejjednodušším způsobem uložení je použitím funkcí save, která je poskytována třídou Model. Zároveň je nezbytné uložit vytvořený slovník, který bude znovu použit při úpravě uživatelské zprávy v aplikaci. Slovník (neboli Tokenizer) je uložen pomocí knihovny Pickle, která má pro tento případ funkci *dump()*.

K porovnání jednotlivých výsledků byla použita knihovna matplotlib, která poskytuje jako výstup grafy. Získané grafy jsou později použity v kapitole Experimenty (6).

5.3 Chatbot

Třída Chatbot slouží k využití naučeného modelu pro predikování záměru otázky na základě uživatelského vstupu. Obsahuje dvě funkce, které budou představeny v následujících kapitolách.

5.3.1 Konstruktor

První funkcí je konstruktor třídy, který připraví objekt (a tím i model) k následnému využití. Nejprve jsou načteny všechny záměry z datové sady a je vytvořen objekt LabelEncoderu, který později slouží k transformování výstupu modelu zpátky do podoby tagů. Tato část kódu byla již podrobněji rozepsána v podkapitole 5.2.1. Následně můžeme sledovat načtení Tokenizeru neboli slovníku pomocí knihovny Pickle a její funkce *load()*. Tokenizer v této třídě slouží k upravení uživatelského vstupu na sekvenci kladných čísel. Vytvořen a upraven byl podle datové sady již při učení modelu, kde byl následně i uložen. Posledním krokem je načtení naučeného modelu pomocí funkce *load_model()*. Konstruktor třídy Chatbot je na ukázce kódu číslo 5.7.

Ukázka kódu 5.7 – Konstruktor třídy Chatbot

```
15    def __init__(self):
16        self.df = pd.read_csv('corona_dataset.csv')
17        # Get list of unique Tags.
18        self.tags = self.df['Tag'].unique()
19        # Remove rows with NaN values.
20        self.df = self.df.dropna(axis='rows', how='any')
21        self.lable_encoder = LabelEncoder()
22        self.y_train = self.lable_encoder.fit_transform(self.df['Tag'])
23
24        self.tokenizer = Tokenizer()
25        with open('tokenizer.pickle', 'rb') as handle:
26            self.tokenizer = pickle.load(handle)
27
28        self.model = keras.models.load_model('model')
```

(Zdroj: Autor)

5.3.2 Funkce get_tag

Druhou funkcí třídy Chatbot je *get_tag()*. Samotná aplikace ji vyvolá za účelem určení záměru otázky. Nejprve dojde k předzpracování uživatelského vstupu pomocí třídy NltkUtils (její fungování je detailněji pospáno v kapitole 5.4). Poté je uživatelský vstup převeden do sekvence kladných čísle pomocí třídy Tokenizer, která byla načtena v konstruktoru třídy. Dále je potřeba použít funkci padding, která byla také již popsána

výše. Pro její správné fungování je potřeba změnit tvar pole, v kterém se upravený uživatelský vstup nachází. Padding tedy doplní vstup nulami a tento vstup je následně použitý jako parametr funkce *predict()*, která je poskytována třídou Model. Instance této třídy byla vytvořena již v konstruktoru. Výstupem funkce *predict()* je pole obsahující předpověděné pravděpodobnosti všech tagů. Z nich je vybrán index s nejvyšší hodnotou pomocí funkce *argmax()*. Pravděpodobnost správného výsledku je uložena do proměnné *prop* a odpovídající záměr je vybrán použitím funkce *inverse_transform()*, která je poskytována třídou LableEncoder. Pokud je pravděpodobnost správnosti výsledku nižší než 0,5, výsledek je zahozen a uživatel upozorněn, že Chatbot neporozuměl otázce. Pokud je pravděpodobnost správnosti predikce vyšší než 0.7, dojde k vrácení výsledku predikce a jeho pravděpodobnosti. Zpracování funkce get_tag() si můžeme prohlédnout na ukázce kódu číslo 5.8.

Ukázka kódu 5.8 – Funkce get_tag()

```
30      def get_tag(self, input_message):
31          # Preprocess user input
32          texts_p = []
33          input_message = nltk_utils.preprocess_row(input_message)
34          texts_p.append(input_message)
35          input_message = self.tokenizer.texts_to_sequences(texts_p)
36          input_message = np.array(input_message).reshape(-1)
37          input_message = pad_sequences([input_message], 24)
38          # Model predict
39          output = self.model.predict(input_message)
40          # Get best of predicted options
41          prop = output[0][output.argmax()]
42          output = output.argmax()
43          # Transform output of model to Tag
44          response_tag = self.lable_encoder.inverse_transform([output])
45          print(f"Output of model: Predicted Tag is {response_tag} with prop: {prop}")
46          if prop < 0.5:
47              return
48
49          return response_tag[0], prop
```

(Zdroj: Autor)

5.4 NltkUtils

NltkUtils je třída, která předzpracovává vstup z datové sady nebo uživatelský vstup. Zahrnuje několik funkcí, které si představíme v následujících podkapitolách. Pro správné fungování byly použity následující knihovny:

- Nltk – Nabízí určité funkce sloužící k úpravě textu, které není potřeba znovu definovat.

- String – Umožňuje provádět běžné operace pro úpravu řetězců.
- Numpy – Poskytuje operace pro práci s vektory a maticemi.
- Re – Slouží k provádění operací s regulárními výrazy.

5.4.1 Funkce pro předzpracování textu

Tato funkce je jako jediná vyvolávána ostatními částmi aplikace a spojuje dohromady všechny ostatní funkce. Jako parametr dostane neupravený vstup z datové sady nebo uživatelský vstup. V prvním kroku vyvolá funkci pro odstranění synonym. Druhým krokem je převedení vstupu na malá písmena. Následně vyvolá funkci pro odstranění určitých znaků, poté provede stemování pomocí funkce stem. Nakonec spojí slova opět do řetězce a takto upravený řetězec vrátí. Tento postup si lze prohlédnout na ukázce kódu číslo 5.9.

Ukázka kódu 5.9 – Zpracování funkce preprocess_row()

```
23  def preprocess_row(sentece):
24      sentece = remove_synonyms(sentece)
25      sentece = sentece.lower()
26      sentece = remove_punctuation(sentece)
27      sentece = [stem(w) for w in sentece.split() if w not in string.punctuation]
28      #sentece = [lemm(w) for w in sentece.split() if w not in string.punctuation]
29      sentece = ' '.join(map(str, sentece))
30
31      return sentece
```

(Zdroj: Autor)

5.4.2 Funkce pro odstranění synonym

Protože celá datová sada i uživatelské vstupy obsahují několik slov se stejným významem a pro model není podstatné, v jakém jsou tyto slova tvaru, jsou tato slova odebrána a nahrazena stejným výrazem. Výrazy se stejným významem jsou např.: *'Coronavirus/COVID-19'*, *'covid19'*, *'COVID-19'*, *'corona'*, *'Coronavirus'*, *'Corona'*, *'Covid-19'*. Tato hesla jsou všechna nahrazena jedním – *coronavirus*, jak je vidět na ukázce kódu číslo 5.10. Tato funkce je provedená pomocí knihovny re.

Ukázka kódu 5.10 – Funkce umožňující odstranění synonym

```
53  # Synonyms for coronavirus
54  def remove_synonyms(sentece):
55      big_regex = re.compile('|'.join(map(re.escape, covid_synonyms)), re.IGNORECASE)
56      sentece = big_regex.sub("coronavirus", sentece)
57
58      return sentece
```

(Zdroj: Autor)

5.4.3 Funkce odstraňující interpunkci

Tato funkce slouží k odstranění znaků, které nemají význam při učení modelu. Jsou odstraněny pomocí knihovny re a samotné fungování je obdobou funkce *remove_synonyms()*. Zde jsou uvedeny příklady znaků, které jsou odstraněny: *'?', '!', '.', ',', '_', '('*. Řadí se mezi ně i otazník, protože se počítá s tím, že každý uživatelský vstup bude určitou formou otázky, která je chatbotu položena. Pokud by bylo potřeba, aby chatbot rozpoznal oznamovací větu od otázky, bylo by vhodné tyto znaky ve vstupu pro model ponechat.

5.4.4 Stemmování

Poslední, ale neméně důležitou funkcí, která slouží k předzpracování vstupu, je *stem()*. Z ukázky kódu číslo 5.11 je patrné, že celou funkčnost zajišťuje třída PorterStemmer() a její funkce stem(). Tato funkce zamění originální slovo za pouhý kořen slova. Stemmování si lze lépe představit na příkladu slova *connections*, které bude zaměněno za slovo *connect*. Tato funkce pomáhá modelu při učení zejména v případech, kdy datová sada není příliš obsáhlá. Příklad jejího využití pro tento chatbot bude vysvětlen na následujících zjednodušených větách: *Can my children get covid?* a *Can my child get covid?*. U obou příkladů se uživatel ptá na otázku se stejným záměrem, otázky jsou ale položeny jinak. Pokud by nedošlo ke stemmování, tak by model nevěděl že slova children a child mají v tomto smyslu stejný význam. Ke stemmování ale dojde, a proto bude mít model pokaždé jako vstup větu *Can my child get covid?*. Pokud existuje riziko, že metodou stemmování dojde ke ztrátě informace o kontextu, je lepší ji nepoužívat.

Ukázka kódu 5.11 – Zpracování funkce stem()

```
44  # Stemming words
45  def stem(word):
46
47      return stemmer.stem(word.lower())
```

(Zdroj: Autor)

5.5 Chat app

Fungování aplikace využívající chatbota je nasimulováno pomocí jednoduchého RestApi, které komunikuje za použití JSONu s nekomplikované HTML šablony.

V následujících podkapitolách si jednoduše představíme fungování chatbota v aplikačním celku.

Pro zjednodušení vývoje byly použity dva frameworky Pythonu. První umožňuje vytvořit lokální server a nasimulovat tak fungování aplikace na skutečném serveru. Tento framework se jmenuje Socket.IO. Druhý se jmenuje Flask a je to mikro webový framwork, který umožňuje například renderování šablony.

5.5.1 Chat resource

Jelikož se jedná pouze o simulaci fungování aplikace, nastavení severu není složité. Slouží k němu prvních šest řádků ukázky kódu číslo 5.12. Další tři řády vytváří instanci tříd, které poskytují níže popsané funkce. Třída Chatbot poskytuje funkci pro predikci neboli rozpoznání záměru otázky. Její fungování je rozebráno v kapitole číslo 5.3. Třída ResponseProvider poskytuje odpovědi na základě vyhodnocení modelu, popsána je v textu níže.

Ukázka kódu 5.12 – Nastavení lokálního serveru a vytvoření potřebných tříd

```
9   # App settings
10  async_mode = None
11  app = Flask(__name__)
12  app.config['SECRET_KEY'] = 'secret!'
13  socket_ = SocketIO(app, async_mode=async_mode)
14  thread = None
15  thread_lock = Lock()
16  # Chatbot classes
17  chatbot = Chatbot()
18  responseProvider = ResponseProvider()
19  dataUtils = DataUtils()
```

(Zdroj: Autor)

Fungování celého Rest Api je zajištěno pomocí dvou funkcí. První slouží k renderování šablony na výchozí URL adrese http://localhost:5000/. Druhá slouží k odpovídání na event, který je vyvoláván právě touto šablonou. Jak již bylo řečeno, data jsou předávána v JSON, tudíž je nutné je převést na objekt zpracovatelný Pythonem. Následně se vyvolá funkce chatbota *get_tag()*, která vyhodnotí otázku a vrátí predikovaný tag s pravděpodobností správnosti. Pokud není žádná hodnota třídou vrácena, znamená to, že procento správnosti určení záměru bylo příliš nízké a je potřeba předat uživateli zprávu o tom, že chatbot neporozuměl otázce. Pokud ale chatbot vrátí tag otázky, tak se zavolá třída ResponseProvider a její funkce *get_resp_by_tag()*. Ta poté vybere odpovídající odpověď z databáze. Pokud vše proběhne v pořádku, je HTML šabloně vrácen objekt ve formátu JSON, který obsahuje jak otázku, tak pravděpodobnost

správnosti odpovědi a samotnou odpověď. Obě tyto funkce zajišťující fungování Rest Api si lze prohlédnout na ukázce kódu číslo 5.13.

Ukázka kódu 5.13 – Zpracování Rest Api

```python
@app.route('/')
def index():
    return render_template('./ChatApp.html', async_mode=socket_.async_mode)

@socket_.on('question_event')
def handle_question_event( json_req ):
  req = json.loads(str(json_req))
  question = req['question']
  tag, prop = chatbot.get_tag(question)
  answer = responseProvider.get_resp_by_tag(tag)
  dataUtils.saveData(question, prop, tag)

  response = {
    "question": question,
    "prop": prop,
    "answer": answer
  }
  json_response = json.dumps(response)
  print(json_response)

  socket_.emit('response', json_response)
```

(Zdroj: Autor)

5.6 Uživatelské rozhraní

Uživatelské rozhraní neboli HTML šablona je vytvořené pomocí Bootstrapu, který umožňuje úpravu vzhledu základních prvků šablony. Dále je tvořen scriptem jQuery, který komunikuje s lokální serverem Pythonu. Jak je vidět na ukázce kódu

Ukázka kódu 5.14 – Script umožňující komunikaci HTML šablony

```javascript
var socket = io.connect( 'http://' + document.domain + ':' + location.port )
// broadcast a message
socket.on( 'connect', function() {
  socket.emit( 'connected_user', {
    data: 'User Connected'
  } )

  var form = $( 'form' ).on( 'submit', function( e ) {
    e.preventDefault()
    let user_input = $( 'input.message' ).val()
    if (user_input.length > 5) {
      socket.emit( 'question_event', "{\"question\":\" " + user_input + " \"}" )
    } else if (user_input.length > 0) {
      console.log('Input is short..' + user_input.length);
    }
    // empty the input field
    $( 'input.message' ).val( '' ).focus()
  } )
} )

// capture message
socket.on( 'response', function(response) {
  response = JSON.parse(response)
  console.log(response.answer + 'with prop: ' + response.prop)
  if( typeof response.answer !== 'undefined' ) {
    $( 'div.message_holder' ).append( '<div class="msg_bbl"><b style="color: #000">You: </b> '+response.question+'</div>' )
    $( 'div.message_holder' ).append( '<div class="msg_bbl"><b style="color: #000">Chatbot: </b> '+response.answer+'</div>' )
  }
} )
```

(Zdroj: Autor)

číslo 5.14, tento skript pouze jednoduše reaguje buď na uživatelské akce (neboli otázky), jako je tomu například u odeslání formuláře, nebo na odpovědi ze strany serveru, které následně zobrazuje pomocí HTML tagů. Zároveň ověřuje, jestli zpráva obsahuje alespoň 5 znaků, protože chatbot při takto krátkých otázkách nemusí predikovat správné záměry otázek.

5.7 Třída ResponseProvider

O poskytování odpovědí na základě predikce se stará třída ResponseProvider. Poskytuje funkci, která umožňuje dvěma různými způsoby získat odpověď na základě tagu. První způsob slouží pro aplikaci jako celek, protože odpovědi jsou poskytované pomocí databáze PostgreSQL, která umožňuje jednoduší správu v případě nasazení aplikace do reálného prostředí. Druhým způsobem je získání odpovědí ze souboru CSV. Tato možnost slouží spíše pro jednoduší testování a v případech, kdy není potřeba databáze. Funkci *get_resp_by_tag()* si můžete prohlédnout na ukázce kódu číslo 5.15, kde je vidět, že k vybrání možnosti získání odpovědi slouží parametr PSQL. Pokud je hodnota PSQL rovna *True,* dojde k propojení s databází a získání odpovědi pomocí standartního SQL dotazu. Pokud je PSQL rovno *False*, dojde ke čtení CSV souboru a získání odpovědi pomocí datové struktury DataFrame poskytované knihovnou Pandas.

Ukázka kódu 5.15 – Funkce poskytována třídou ResponeProvider

```
10      def get_resp_by_tag(self, tag, PSQL):
11          if PSQL:
12              try:
13                  conn = psycopg2.connect(user="chatbot",
14                                          password="password123",
15                                          host="127.0.0.1",
16                                          port="5432",
17                                          database="response_db")
18
19                  cursor = conn.cursor()
20                  select_response_query = "SELECT * FROM reponse WHERE tag = " + tag
21                  print("Executed query: " + select_response_query)
22                  cursor.execute(select_response_query)
23                  response = cursor.fetchall()
24                  print(response)
25                  return response[0]
26              except (Exception, Error):
27                  print("Error happen when SELECT query")
28              finally:
29                  if conn:
30                      cursor.close()
31                      conn.close()
32          else:
33              df = pd.read_csv('response.csv')
34              row = df.loc[df['tag'] == tag]
35              return row['response']
```

(Zdroj: Autor)

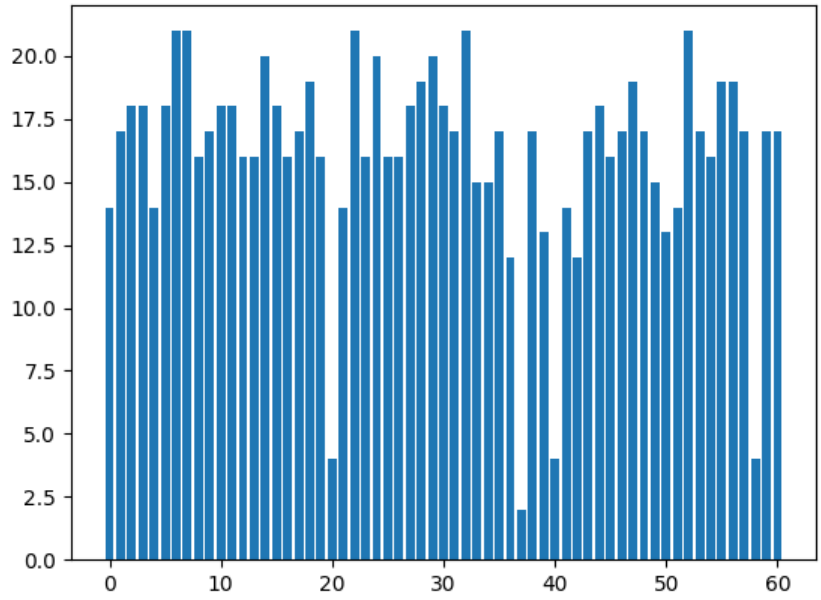
6 Experimenty

Tato kapitola obsahuje experimenty spojené s implementací chatbota, která je detailně rozebrána v kapitole 5. Dále se zabývá výsledky jednotlivých experimentů vykonaných na chatbotu, zejména přesností predikce.

6.1 Vlastnosti datové sady

Před samotným provedením experimentu je potřeba nahlédnout na data z širšího úhlu pohledu. K tomu nám poslouží histogram zobrazený na grafu číslo 6.1. Na ose x jsou zobrazeny jednotlivé tagy a na ose y počet otázek. Z tohoto histogramu lze usuzovat, že většina tagů má k sobě přiřazený přibližně stejný počet otázek. U tagů, které mají nižší počet otázek, dochází pravděpodobně k chybné predikci, protože se jim váhy dostatečně nepřizpůsobí.

Graf 6.1 – Histogram počtu otázek k jednotlivým tagům



(Zdroj: Autor)

6.2 Úvod do experimentů

Experimenty, které byly provedeny na chatbotu, jsou založeny na využití různých modelů a vrstev nebo na úpravě hyper-parametrů jednotlivých modelů. Základními veličinami, podle který budou posuzovány jednotlivé modely, jsou následující:

- Accuracy neboli přesnost vyjadřuje přesnost predikce, které model dosahuje při učení na datové sadě.
- Validation accuracy neboli validační přesnost vyjadřuje přesnost predikce, které model dosahuje na nových datech. Validační přesnost se testuje po průběhu každé epochy s využitím dat, na kterých se model neučil.

Obě tyto veličiny by měly v ideálním případě dosahovat co nejvyšší hodnoty, tedy hodnoty blížící se 1. Pokud se veličina Accuracy nepřibližuje optimálním hodnotám, může to značit například to, že je použitý nevhodný optimalizátor, aktivační funkce nebo architektura neuronové sítě. Také to může znamenat, že model neměl k dispozici dostatek dat nebo času na naučení se vzoru v datech a je tedy například potřeba zvýšit počet epoch.

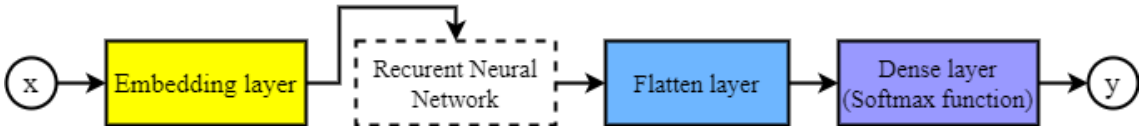
Pokud dochází k tomu, že Validation Accuracy nedosahuje požadovaných hodnot, může to opět znamenat, že model opět nemá dostatek dat anebo že dochází k overfittingu. Tento pojem byl již zmíněn v textu výše a znamená, že si neuronová síť vytvořila určité silné váhy na základě učících dat a nová data jsou proto predikovány s nižší přesností.

Datová sada je tedy rozdělena na učící a validační data při plnění modelu, a to v takovém poměru, že 10 % z celkové datové sady slouží k validaci a zbylých 90 % k učení modelu.

6.2.1 Výchozí struktura modelu

V experimentech vycházíme ze struktury, která je zobrazena na obrázku číslo 6.2. Podrobné vysvětlení všech jednotlivých vrstev je poskytnuto

Obrázek 6.1 – Výchozí struktura modelu



(Zdroj: Autor)

v kapitole 5 o implementaci chatbota. Vstup *x* projde vrstvou Embedding, která jej převeden na vektor čísel. Poté se vstup převedený na vektor vloží do Rekurentní neurnové sítě. Výstup z této sítě projde vrstvou Flatten, která upraví jeho tvar pro poslední vrstvu, Dense. Ta pomocí funkcí Softmax vytvoří distribuci pravděpodobnosti y, která se stane finálním výstupem modelu.

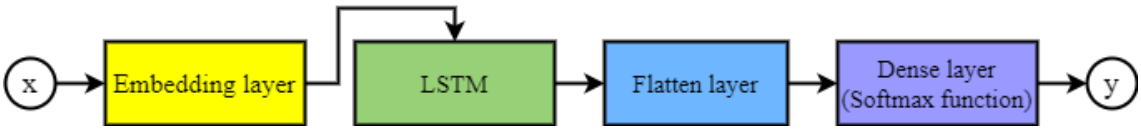
6.3 Experimenty

V následujících podkapitolách budou otestovány tři rekurentní neurnové sítě a jejich optimalizace.

6.3.1 Verze s neurnovou sítí LSTM

První struktura modelu obsahuje rekurentní neurnovou síť s názvem LSTM. Tuto strukturu si můžeme prohlédnout na obrázku číslo 6.2.

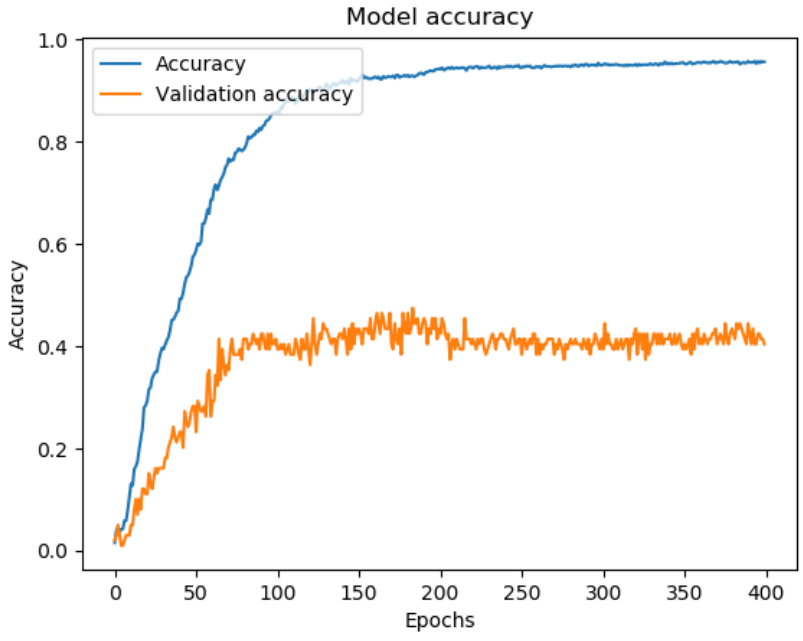
Obrázek 6.2 – Struktura modelu s rekurentní neuronovou sítí LSTM



(Zdroj: Autor)

Z grafu 6.2 je patrné, že hodnota přesnosti predikce při učení dosahuje požadovaných hodnot již po 100. cyklu epoch. Maximální hodnota přesnosti, které model dosáhnul, byla 0,97. Při pohledu na hodnoty přesnosti měřené na validační datech vidíme, že přesahují hranici 0,4 přibližně od 50. cyklu epoch. Maximální hodnota byla 0,47. Přesnost měřená na validačních datech tedy nedosahuje požadovaných hodnot, a proto se struktuře s neuronovou sítí LSTM budeme věnovat později v podkapitole Optimalizace modelů.

Graf 6.2 – Přesnost modelu s rekurentní neuronovou sítí LSTM

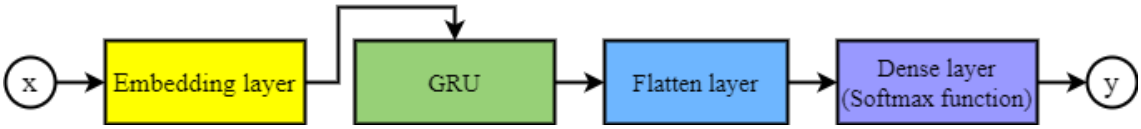


(Zdroj: Autor)

6.3.2 GRU

Struktura obsahující rekurentní neurnovou vrstvu GRU je znázorněna na obrázku číslo 6.3.

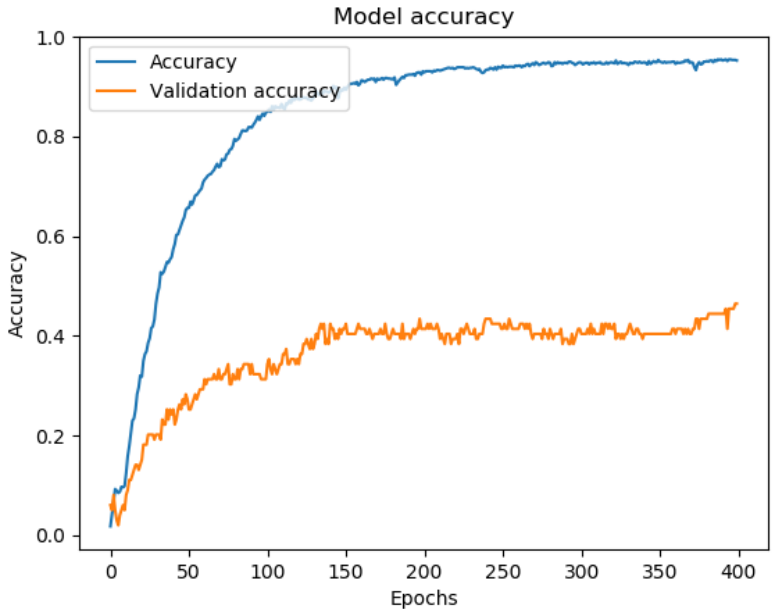
Obrázek 6.3 – Struktura modelu s rekurentní neurnovou sítí GRU



(Zdroj: Autor)

Průběh učení je zanesen do grafu číslo 6.3. Obě naměřené hodnoty přesnosti rostou pozvolněji než u modelu s rekurentní neuronovou sítí LSTM. Hodnota přesnosti při učení dosahuje požadovaných hodnot, a to s maximální hodnotou 0,96. U přesnosti měřené na validačních datech byla maximální naměřená hodnota 0,35, což je ještě nižší výsledek než u měření modelu s neurnovou sítí LSTM. Proto se i jí budeme věnovat v podkapitole o optimalizaci (6.3.5).

Graf 6.3 – Přesnost modelu s rekurentní neuronovou sítí GRU

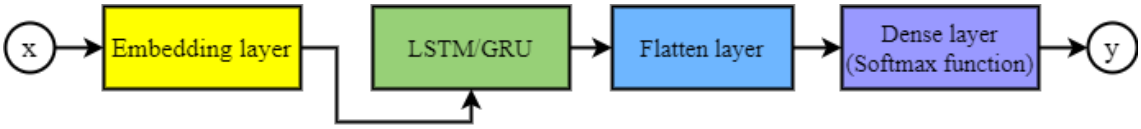


(Zdroj: Autor)

6.3.3 Inverzní GRU a LSTM

Modely, které obsahují rekurentní neuronovou sítí GRU/LSTM s inverzní vstupní sekvencí, si lze prohlédnout na obrázku číslo 6.4. Tyto neurnové sítě s inverzní vstupní sekvencí se od těch klasických liší tím, že mají obrácenou vstupní sekvenci. Příklad klasického vstupu bude *can my pet get covid* – v případě neurnové sítě s inverzní vstupní sekvencí bude vstup *covid get pet my can*. Tento postup učení s inverzní vstupní sekvencí může zvýšit přesnost predikce. Umožnění použití inverzní vstupní sekvence je možné použitím parametru *go_backwards*.

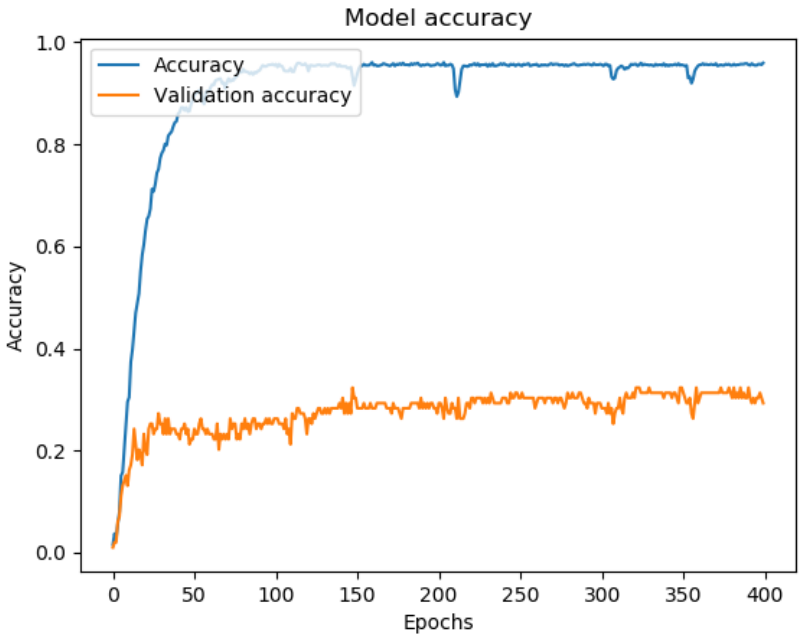
Obrázek 6.4 – Struktura modelu s inverzním vstupem pro LSTM/GRU



(Zdroj: Autor)

Průběh učení na rekurentní neuronové síti LSTM si lze prohlédnout na grafu číslo 6.4, kde lze vidět, že metoda použití neurnové sítě LSTM na datové sadě s inverzními vstupními sekvencemi modelu neprospěla. Validační přesnost dosáhla maximální hodnoty 0,32.

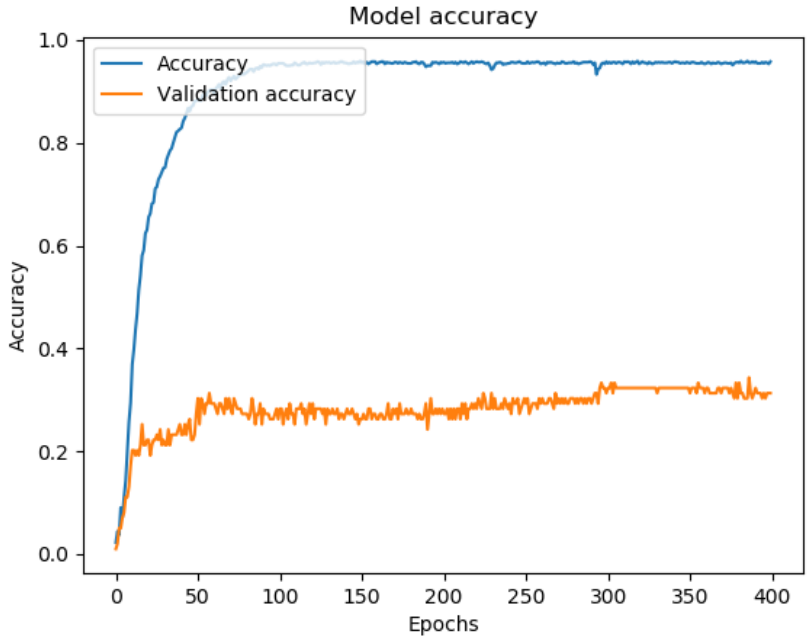
Graf 6.4- Přesnost modelu s inverzním vstupem pro LSTM



(Zdroj: Autor)

Průběh učení s inverzní vstupní sekvencí na neurnové sítí GRU je zobrazen grafem číslo 6.5. Ani v tomto případě metoda s inverzními vstupními sekvencemi nepomohla zvýšit validační přesnost, která dosáhla maximální hodnoty 0,34.

Graf 6.5 - Přesnost modelu s inverzním vstupem pro GRU

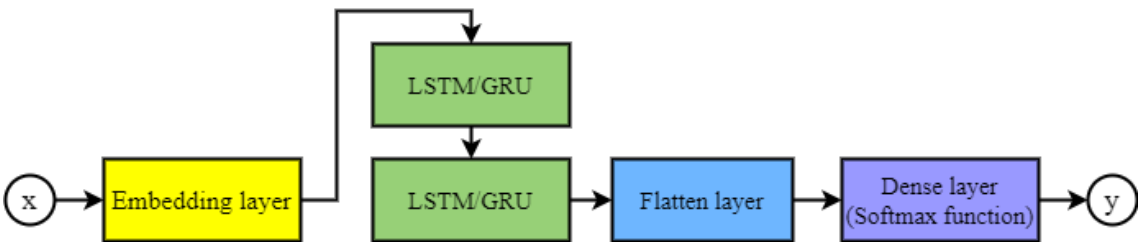


(Zdroj: Autor)

6.3.4 Dvouvrstvá struktura LSTM/GRU

Další metodou, která se používá pro zvýšení přesnosti modelu je takzvaná dvouvrstvá struktura. To znamená že se použijí dvě vrstvy stejných neurnových sítí. Strukturu tohoto modelu si můžeme prohlédnout na obrázku číslo 6.5. I tato metoda často pomáhá zvýšení přesnosti modelu, protože dvě vrstvy neuronových sítí umožňují uložit více informací.

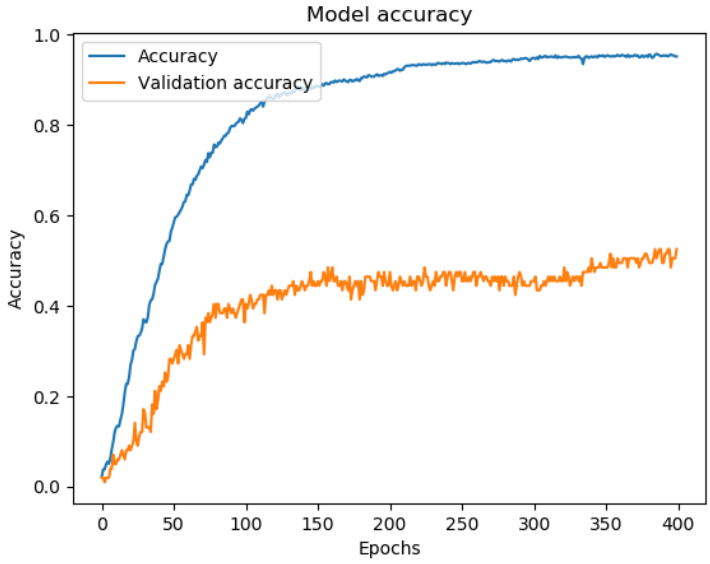
Obrázek 6.5 – Dvouvrstvá struktura modelu s LSTM/GRU



(Zdroj: Autor)

Na grafu číslo 6.6 je vidět průběh učení pomocí dvouvrstvé struktury LSTM. Z grafu je patrné, že tato metoda opravdu pomohla zvýšit validační přesnost. Maximální hodnota, které validační přesnost dosáhla, je 0,52.

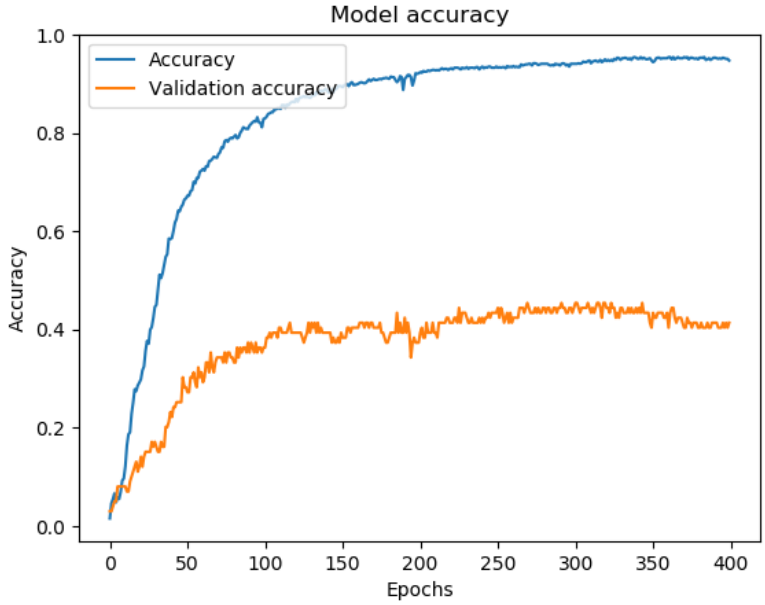
Graf 6.6 - Přesnost modelu s dvouvrstvou strukturou LSTM



(Zdroj: Autor)

Průběh učení dvouvrstvé struktury GRU si lze prohlédnout na grafu 6.7. I zde došlo ke zlepšení validační přesnosti, maximální hodnota (0,45) je nicméně pořád menší než u dvouvrstvé struktury s LSTM.

Graf 6.7 – Přesnost modelu s dvouvrstvou strukturou GRU



(Zdroj: Autor)

6.3.5 Výsledky a optimalizace modelů

Shrnutí výsledků přesnosti predikce při učení jednotlivých modelů si lze prohlédnout v tabulce číslo 6.1. Můžeme si všimnout, že všechny verze těchto modelů dosahovaly optimálních hodnot.

Tabulka 6.1 – Přesnost jednotlivých modelů na učících datech

	Jednoduchý model	Model s inverzní vstupní sekvencí	Model s dvouvrstvou strukturou
LSTM	0,9570	0,9604	0,9570
GRU	0,9604	0,9592	0,9547

(Zdroj: Autor)

Tabulka číslo 6.2 zaznamenává přesnost predikce na validačních datech. Zde naopak žádná z verzí modelu požadovaných hodnot nedosahuje. Nejvyšší maximální hodnotu (0,52) má model využívající dvouvrstvou strukturu LSTM. Naopak nejhoršího výsledku dosáhl model s neurnovou sítí LSTM, který byl učen pomocí metody inverzních vstupů.

Tabulka 6.2 – Přesnost jednotlivých modelů na učících datech

	Jednoduchý model	Model s inverzní vstupní sekvencí	Model s dvouvrstvou strukturou
LSTM	0,4747	0,3232	0,5253
GRU	0,3535	0,3434	0,4545

(Zdroj: Autor)

Jelikož ani jeden z modelů nedosahuje požadovaných hodnot, je nutné pokusit se modely optimalizovat. Nejpravděpodobnějšími problémy, kvůli kterým nedochází ke zvýšení přesnosti na validačních datech, jsou nedostatek dat a overfitting. Nejlepším řešením nedostatku dat je využití datové sady s větším počtem dat. Overftting lze řešit několika způsoby:

Prvním způsobem je použít takzvané dropout regularizace. Jejich principem je vyřazení určitých náhodných neuronů, které tím pádem nejsou při učení použité. Postup při procesu učení sítě je následující: váhy neuronů se příliš usazují v kontextu tréninkových dat, což má za následek, že se model příliš specializuje na tréninková data. Při testování na validačních datech poté není přesnost tak veliká. Při použití regularizace

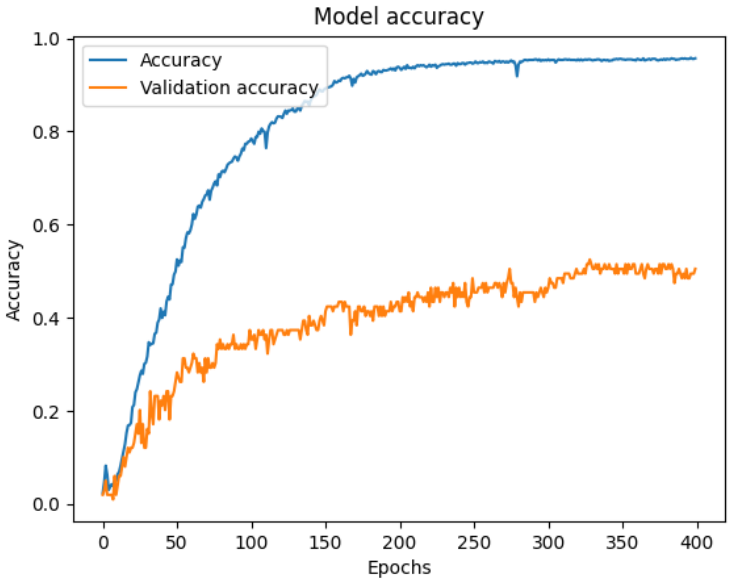
dropout některé neurony „vypadnou“ ze sítě a ostatní neurony tak musí zvládnout reprezentaci potřebnou k předpovědi chybějících neuronů. Výsledkem tohoto postupu je, že naučené neurony na sobě budou více nezávislé (Brownlee, 2020).

Druhým často používaným řešením jsou regulrizátory L1 a L2, které upravují ztrátovou funkci tak, aby byla neurnová síť penalizována za využití příliš silných vah v neuronech. L1 k tomuto účelu využívá součet absolutních hodnot vah, oproti tomu L2 používá součet druhé mocniny jejich hodnot.

Následující experimenty zaměřené na zlepšení přesnosti modelu budou vycházet ze dvou výše popsaných optimalizačních metod. Optimalizovaný model bude obsahovat dvouvrstvou strukturu s neuronovou sítí LSTM, jelikož byla na základě předchozích experimentů označena za nejpřesnější. S každým typem regularizátoru bylo provedeno několik experimentů s různým nastavením.

Zde jsou uvedeny výsledky pouze těch experimentů, u nichž byla výsledná přesnost při testování na validačních datech největší. První experiment se týkal modelu s regularizátorem L1, jehož průběh učení si můžeme prohlédnout na grafu číslo 6.8. Tento regularizátor nepřinesl zlepšení přesnosti predikce na validačních datech; maximální hodnota validační přesnosti dosáhla 0,52.

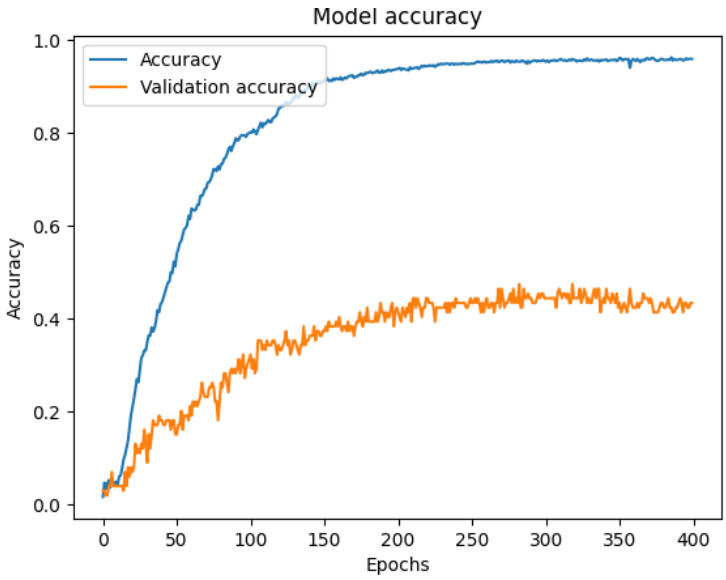
Graf 6.8 – Přesnost modelu s dvouvrstvou strukturou LSTM a regularizátorem L1



(Zdroj: Autor)

Další experiment byl proveden na modelu s regularizátorem L2. Tento model dosáhl dokonce horších výsledků; maximální hodnota se rovnala 0,47. Průběh tohoto učení si lze prohlédnout na grafu číslo 6.9, kde si lze všimnout také mírného poklesu okolo 350 cyklu epoch. To může být zapříčiněno právě overfittingem, jelikož váhy se po určitém cyklu epochy příliš přizpůsobili učícím datům.

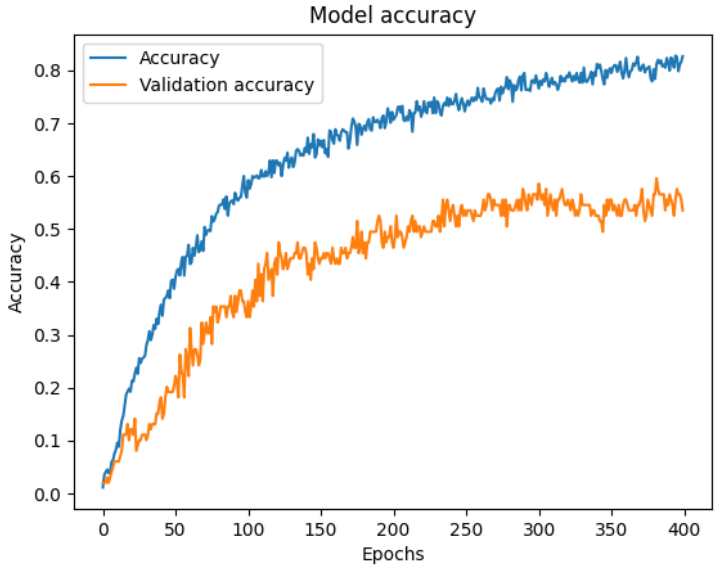
Graf 6.9 - Přesnost modelu s dvouvrstvou strukturou LSTM a regularizátorem L2



(Zdroj: Autor)

Další možností optimalizace modelu bylo využití dropout vrstvy. Bylo vykonáno několik experimentů s jejím různým nastavením. Průběh učení modelu s nejlepším

Graf 6.10 - Přesnost modelu s dvouvrstvou strukturou LSTM a Dropout vrstvou



(Zdroj: Autor)

nastavením pro dropout vrstvu si můžete prohlédnout na grafu číslo 6.10. To, že byla tato metoda optimalizace nejúspěšnější potvrzuje také potvrdila maximální hodnota přesnosti na validačních datech, která se rovná 0,59.

Dále byly vyzkoušeny různé kombinace regularizátorů a dropout vrstev a jejich nastavení. Nejpřesnější průběhy učení, které byly provedené na různých kombinacích optimalizačních metod, jsou k nahlédnutí v příloze. Výsledky těchto experimentů jsou zaznamenány v tabulce číslo 6.3. Vyplívá z nich, že nejpřesnější predikce na validačních datech dosáhl model s optimalizační metodou dropout v kombinaci s regularizátorem L2 – hodnota se rovná 0,63.

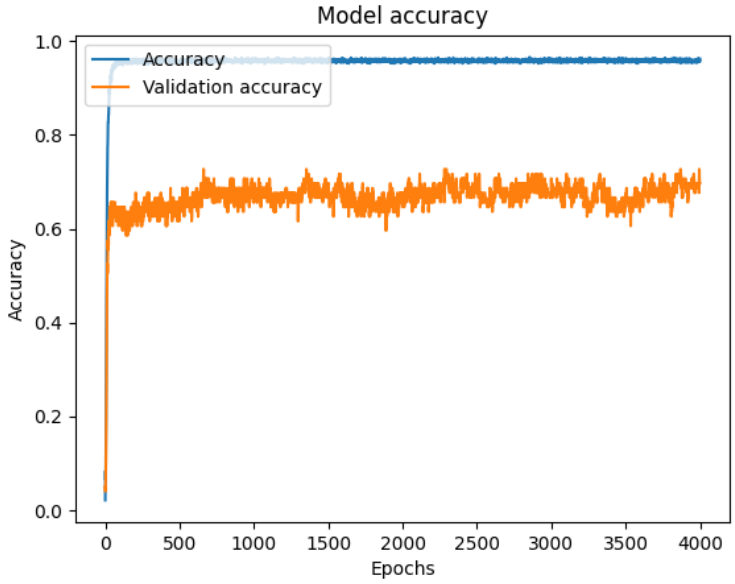
Tabulka 6.3 – Kombinace optimalizačních metod a jejich validační přesnost na dvojvrstvé struktuře LSTM

Dropout	L1	L2	Validační přesnost
	x		0,5252
		x	0,4747
	x	x	0,5151
x			0,5959
x	x		0,6161
x		x	0,6363
x	x	x	0,5757

(Zdroj: Autor)

Z grafu učení nejúspěšnějšího modelu lze vidět, že nedošlo k úplné stabilizaci hodnot. Proto je potřeba zjistit správný počet epoch, který povede k nejvyšší možné validační přesnosti. K učení modelu bylo tedy použito 4000 cyklů epoch, což sice signifikantně zvýšilo dobu učení, podařilo se ale ještě více optimalizovat stávající model. Ke zvýšení validační přesnosti přispěla také optimalizace parametrů Embedding vrstvy, konkrétně zvětšení výstupních vektorů, které jsou poskytovány jako vstupy pro první vrstvu LSTM. Průběh učení je znázorněn na grafu číslo 6.11. Je patrné, že po výrazném zvýšení cyklů epoch dosáhl model ještě lepších výsledků. Validační přesnost se ustálila přibližně na hodnotě 0,66, maximální hodnota se rovná 0,71.

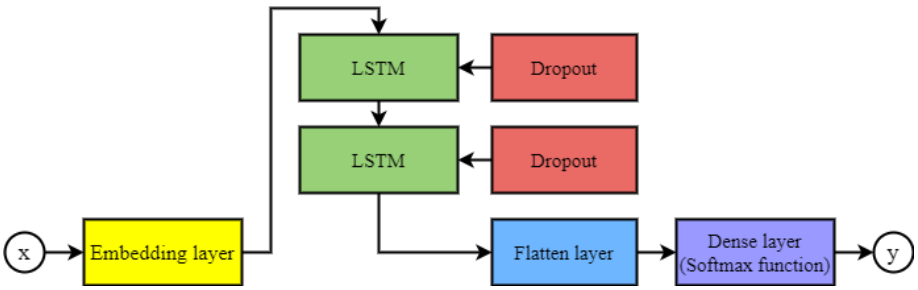
Graf 6.11 – Přesnost modelu s dvouvrstvou strukturou modelu LSTM s Dropout vrstvou a regularizátorem L2 (4000 cyklů epoch)



(Zdroj: Autor)

V posledních experimentech se samotná struktura modelu ani optimalizačních metod neměnila, změnu podstoupily pouze jejich parametry. Stejně jako v předchozím případě, i nyní bylo zvoleno 4000 cyklů epoch, tentokrát byl ale využit parametr callback neboli checkpoint. Tato funkce umožňuje uložení modelu s největší přesností v průběhu všech cyklů epoch. Výsledkem učení byl finální model, jehož strukturu si lze prohlédnout na obrázku číslo 6.6, kde je vidět postavení jednotlivých vrstev ve struktuře modelu.

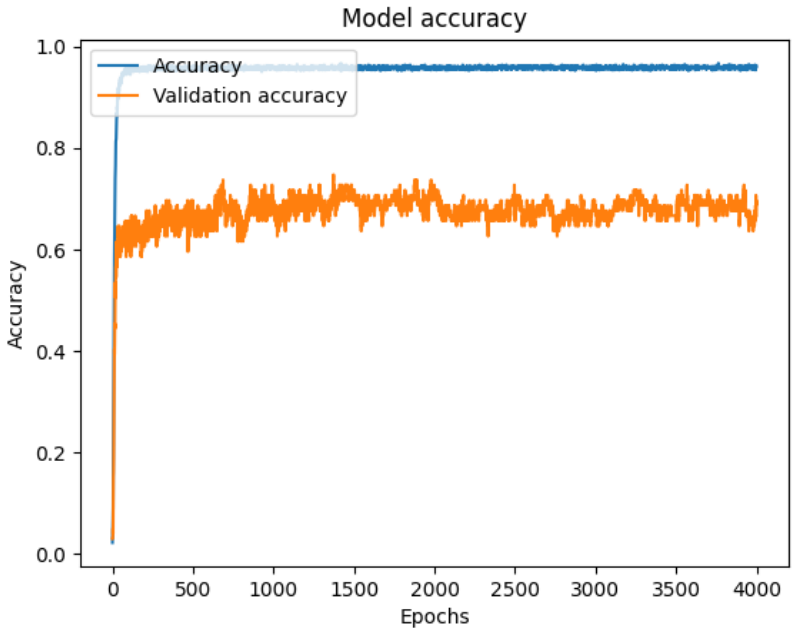
Obrázek 6.6 – Finální struktura modelu



(Zdroj: Autor)

Na grafu číslo 6.12 je zaznamenán průběh učení finálního modelu chatbota. Hodnota validační přesnosti se ustálila na 0,68. Pomocí parametru callback byl uložen model s maximální validační přesností 0,74.

Graf 6.12 – Přesnost finálního modelu

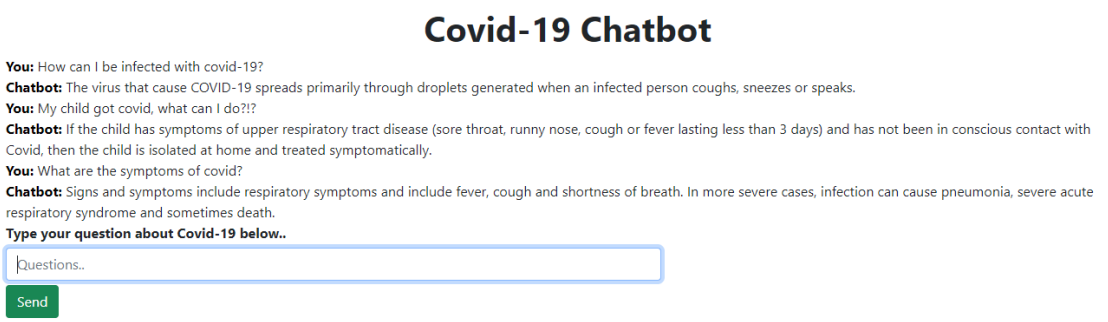


(Zdroj: Autor)

7 Výsledky a zlepšení

Výsledný chatbot je schopný fungování na základě naučeného modelu. Finální model dosahuje uspokojivé přesnosti predikce na učících datech. Na validačních datech dosáhl přesnosti přibližně 52 %. Protože tato hodnota nebyla uspokojivá, došlo k optimalizaci, která zvýšila přesnost modelu o 22 % na přibližně 74 %. Ani tato hodnota není optimální, splňuje nicméně cíl, který byl stanoven v podkapitole 3.1.2 Cíle praktické části. Aplikace chatbota byla úspěšně simulována v lokální prostředí. Následující obrázek číslo 7.1 je příkladem správného fungování aplikace chatbota.

Obrázek 7.1- Příklad fungování aplikace chatbota



(Zdroj: Autor)

Následující seznam shrnuje navrhovaná zlepšení aplikace chatbota jako celku ale i jeho modelu:

- Větší datová sada – Získání a použití větší datové sady by mohlo vést ke zvýšení přesnosti na validačních datech, a tedy i ke zvýšení efektivnosti chatbota.
- Rozšíření okruhu otázek – Pokud by chatbot měl fungovat v reálném prostředí, bylo by vhodné použít daleko širší okruh otázek, na které se uživatel může zeptat.
- Česká lokalizace – K využití chatbota v České republice je potřeba použít datovou sadu s českou lokalizací.
- Vylepšení uživatelského rozhraní – Stávající uživatelské rozhraní je pouze návrhem a před použitím v reálném prostředí by muselo být vylepšeno.
- Zlepšení optimalizace – I přesto, že bylo provedeno přibližně 40 experimentů s modelem a jeho optimalizací, která umožnila zlepšení přesnosti tohoto modelu přibližně o 22 %, je velmi pravděpodobné, že existuje varianta nastavení parametrů optimalizačních metod nebo kombinace jiných optimalizačních metod, která by zvýšila přesnost predikce.

8 Závěr

Je zřejmé, že metody a přístupy, které se používají při zpracování jazyka, se neustále vyvíjejí a umožňují tak dosahování lepších a lepších výsledků. Ještě rychlejším tempem se dopředu posouvají metody a přístupy týkající se umělé inteligence, strojového učení nebo hlubokých neuronových sítí. V teoretické části byl zmíněn pouze zlomek těchto přístupů a metod, které slouží k vytváření moderních aplikací na základě umělé inteligence.

V teoretické části byly vysvětleny základní termíny, které se skrývají pod pojmem umělá inteligence. Dále byly popsány různé typy chatbotů a různé přístupy k jejich vývoji. V rámci teoretické části byly také představeny nejzákladnější typy neuronových sítí a metod pro zpracování textu neboli jazyka. Na základě těchto znalostí získaných v teoretické části diplomové práce byla vypracována praktická část.

Výsledkem praktické části je plně fungující aplikace chatbota, která umožňuje uživateli zeptat se na otázky týkající se COVID-19. I přes to, že první pokusy naučení modelu chatbota nebyly příliš úspěšné, protože validační přesnost nedosahovala požadovaných hodnot, je výsledek uspokojující, a to zejména vzhledem k velikosti použité datové sady. Ke zvýšení přesnosti modelu nejvíce přispěly experimenty se samotnými modely a také experimenty s optimalizací. Optimalizace byla opravu úspěšná, jelikož se díky ní podařilo zvýšit přesnost modelu o více než 20 %. Finální verze chatbota dosahuje validační přesnosti přibližně 74 %. Pokud by se datová sada zvětšila objemem jednotlivých záznamů a počtem možných otázek, bylo by možné model použít v reálném prostředí. Jeho implementace by mohla pomoci zvýšit míru informovanosti a snížit množství dezinformací týkajících se probíhající covidové pandemie. Samotné fungovaní modelu chatbota v reálném prostředí bylo nasimulováno pomocí lokálního serveru a lokální databáze. Pro úplnost aplikace bylo vytvořeno jednoduché uživatelské rozhraní, které zjednodušuje použití chatbota.

I. Summary and keywords

The master thesis focuses on a creating chatbot with using artificial intelligence.

At the start of theoretical part, thesis briefly introduce a history of chatbots. Futhuremore the theoretical part introduces some important parts of neural network and explain their principle. Then the master thesis describes some types of neural networks, for exmaple LSTM and GRU, which are often used for creating chatbots. Furthermore, this part of thesis describes methods used for nature language processing, such as Bag-Of-Words or Word Embeggding.

The practical part of this master thesis deals with the creation of a retrival-based chatbot and simulation of an aplication for its use in a real environment. Chatbot serves to answer questions about the Covid-19 pandemic. First, practical part introduce the dataset that is used for chatbot training. Furthermore is explained in detail the software implementation of training script and chatbot application as whole. The last part deals with experiments on the model itself, where the effort to achieve the best possible accuracy on the validation data using optimization methods.

Keywords: Chatbot, artificial intelligence, neural networks, LSTM, development

II. Seznam použité literatury

Algorithmia. (2018, May 7). Introduction to optimizers. Retrieved from https://algorithmia.com/blog/introduction-to-optimizers

Bhagwat, V. A. (2018). *Deep Learning for Chatbots* (Master's thesis). Retrieved from https://scholarworks.sjsu.edu/etd_projects/630/

BHATTACHARYYA, J. (2020, November 5). Popular deep learning frameworks: An overview. Retrieved from https://analyticsindiamag.com/deep-learning-frameworks/

Brandtzaeg, P. B., & Følstad, A. (2017). Why people use chatbots. *Internet Science*, 377-392. doi:10.1007/978-3-319-70284-1_30

Brenner, M. (2018, December 3). Training a goal-oriented chatbot with deep reinforcement Learning — Part I. Retrieved from https://towardsdatascience.com/training-a-goal-oriented-chatbot-with-deep-reinforcement-learning-part-i-introduction-and-dce3af21d383

Brownlee, J. (2017, October 9). A gentle introduction to the bag-of-Words model. Retrieved from https://machinelearningmastery.com/gentle-introduction-bag-words-model/

Brownlee, J. (2017, July 3). Gentle introduction to the Adam optimization algorithm for deep learning. Retrieved from https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

Brownlee, J. (2019, January 22). Loss and loss functions for training deep learning neural networks. Retrieved from https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/

Brownlee, J. (2020, August 26). Dropout regularization in deep learning models with Keras. Retrieved from https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

Cambridge University Press. (2021). Chatbot | meaning in the Cambridge English dictionary. Retrieved from https://dictionary.cambridge.org/dictionary/english/chatbot

Cohen, W. (2018). *Proceedings of the 25th international conference on Machine learning*. NY: Association for Computing Machinery.

Costa, C. D. (2020, March 17). Top programming languages for AI engineers in 2020. Retrieved from https://towardsdatascience.com/top-programming-languages-for-ai-engineers-in-2020-33a9f16a80b0

Csáky, R. K. (2017). Deep Learning Based Chatbot Models. *SCIENTIFIC STUDENTS' ASSOCIATIONS REPORT*.

Dwarampudi M., & Suba Reddy, N. V. (2019). *EFFECTS OF PADDING ON LSTMS AND CNNS*. Cornell University.

Elizondo, D., & Fiesler, E. (1997). A survey of partially connected neural networks. *International Journal of Neural Systems, 08*(05n06), 535-558. doi:10.1142/s0129065797000513

Encyclopædia Britannica, Inc. (2021). Artificial intelligence | Definition, examples, and applications. Retrieved from https://www.britannica.com/technology/artificial-intelligence

IBM. (2020, June 3). What is artificial intelligence (AI)? Retrieved from https://www.ibm.com/cloud/learn/what-is-artificial-intelligence

IBM. (2020, July 15). What is machine learning? Retrieved from https://www.ibm.com/cloud/learn/machine-learning

Karani, D. (2018, September 1). Introduction to word embedding and Word2Vec. Retrieved from https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa

Kropas-Hughes, C. V., Oxley, M. E., Rogers, S. K., & Kabrisky, M. (2000). Autoassociative–heteroassociative neural networks. *Engineering Applications of Artificial Intelligence, 13*(5), 603-609. doi:10.1016/s0952-1976(00)00040-3

Liu, Y., Liu, Z., Chua, T. S., & Sun, M. (2015). Topical Word Embeddings. *Proceedings of the AAAI Conference on Artificial Intelligence*. Retrieved from https://ojs.aaai.org/index.php/AAAI/article/view/9522

Mahendru, K. (2019, August 14). A detailed guide to 7 loss functions for machine learning algorithms with Python code. Retrieved from https://www.analyticsvidhya.com/blog/2019/08/detailed-guide-7-loss-functions-machine-learning-python-code/

Medsker, L. R., & Jain, L. C. (2001). *Recurrent neural networks*. CRC Press, NY.

Mishra, M. (2020, June 2). Different types of approaches to make your own Chatbot — Retrieval vs generative. Retrieved from https://medium.com/analytics-vidhya/different-types-of-approaches-to-make-your-own-chatbot-retrieval-vs-generative-8d524b0505db

MLNerds. (2019, February 14). What is the difference between word2Vec and glove ? Retrieved from https://machinelearninginterview.com/topics/natural-language-processing/what-is-the-difference-between-word2vec-and-glove/

Olah, C. (2015, August 27). Understanding LSTM networks. Retrieved from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Phi, M. (2018, September 24). Illustrated guide to LSTM's and GRU's: A step by step explanation. Retrieved from https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Robino, G. (2016, March 21). Conversational vs transactional chatbots. Retrieved from https://convcomp.it/conversational-vs-transactional-chatbots-172166f3d2fc

Shanmuganathan, S., & Samarasinghe, S. (2016). *Artificial neural network modelling*. Springer.

SHARMA, A. K. (2019, September 22). Introduction to neural networks and activation functions. Retrieved from https://medium.com/analytics-vidhya/introduction-to-neural-networks-and-activation-functions-58e26887c52d

Sibi, P., Jones, S. A., & Siddarth, P. (2013). ANALYSIS OF DIFFERENT ACTIVATION FUNCTIONS USING BACK PROPAGATION NEURAL NETWORKS. *Journal of Theoretical and Applied Information Technology*. Retrieved from http://www.jatit.org/volumes/Vol47No3/61Vol47No3.pdf

Turing, A. M. (1950). *Computing machinery and intelligence*.

Wood, T. (2019, May 17). Softmax function. Retrieved from https://deepai.org/machine-learning-glossary-and-terms/softmax-layer

YEGNANARAYANA, B. (2009). *Artificial neural networks*. PHI Learning Pvt.

Young, T., Hazarika, D., Poria, S., & Cambria, E. (2018). Recent trends in deep learning based natural language processing [Review article]. *IEEE Computational Intelligence Magazine*, *13*(3), 55-75. doi:10.1109/mci.2018.2840738

Zemčík, T. (2019). *A Brief History of Chatbots*. 2019 International Conference on Artificial Intelligenc

III. Seznam obrázků

IV. Seznam ukázek kódu

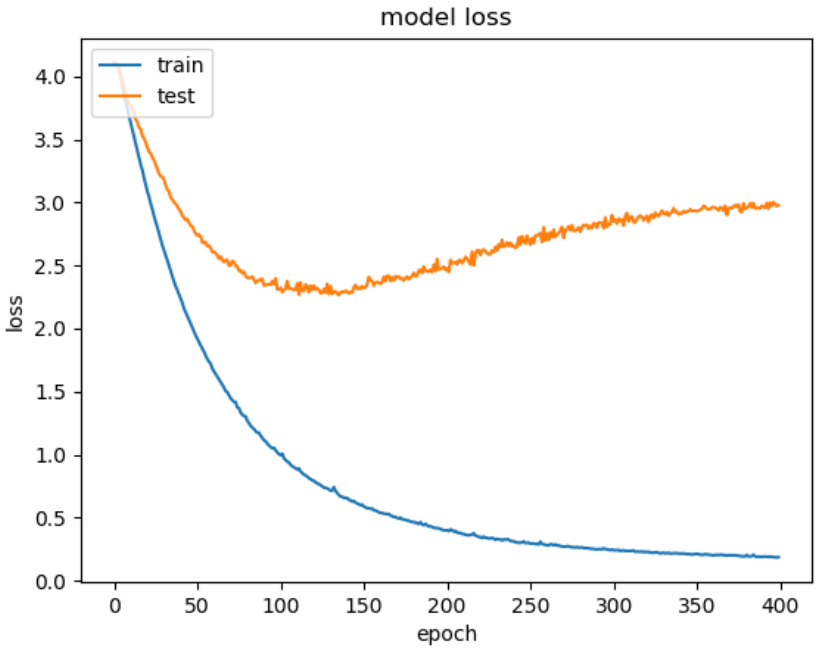
V. Seznam tabulek

VI. Seznam grafů

VII. Seznam příloh

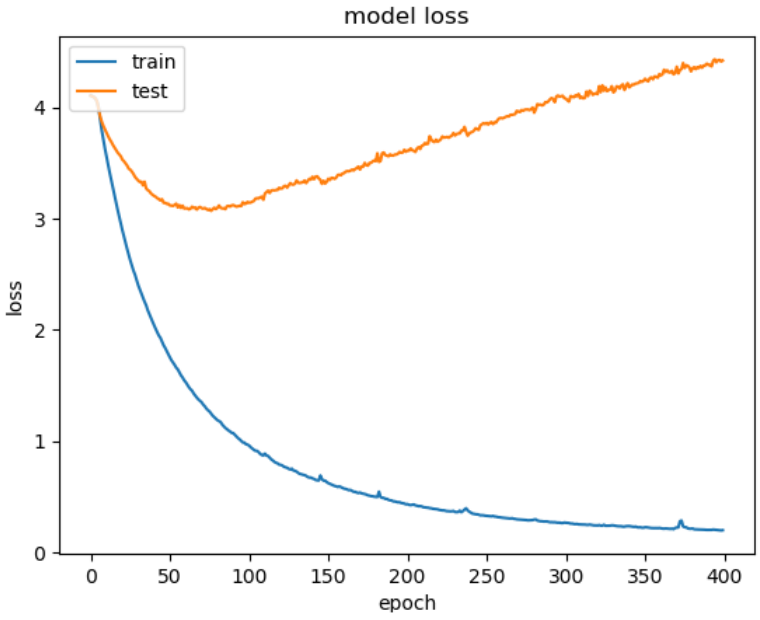
VIII. Přílohy

Příloha 1 - Ztrátová funkce modelu s LSTM



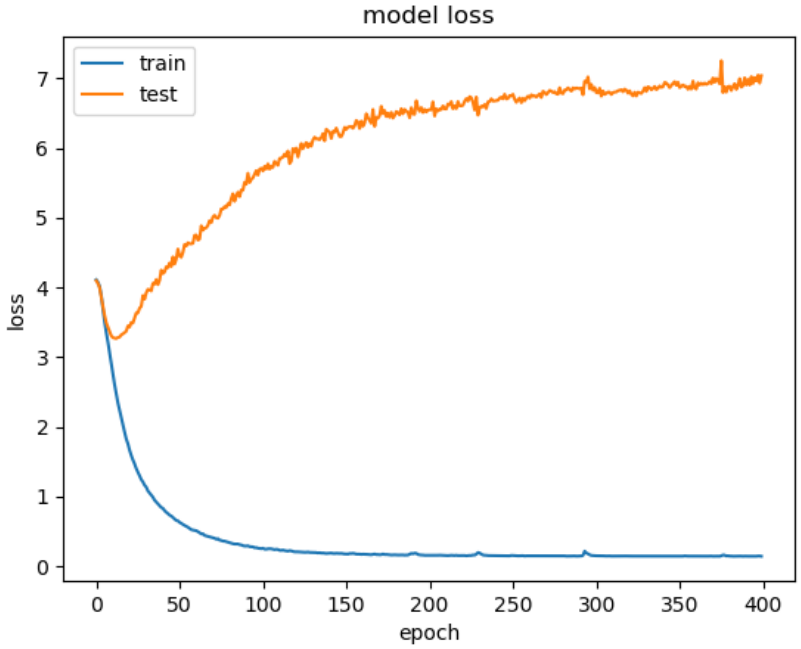
(Zdroj: Autor)

Příloha 2 - Ztrátová funkce modelu s GRU



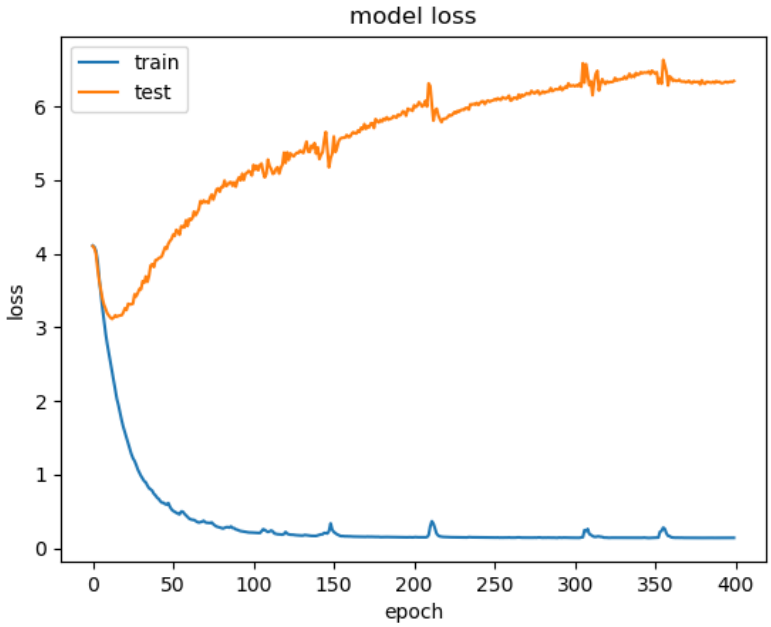
(Zdroj: Autor)

Příloha 3 - Ztrátová funkce modelu s inverzním vstupem pro LSTM



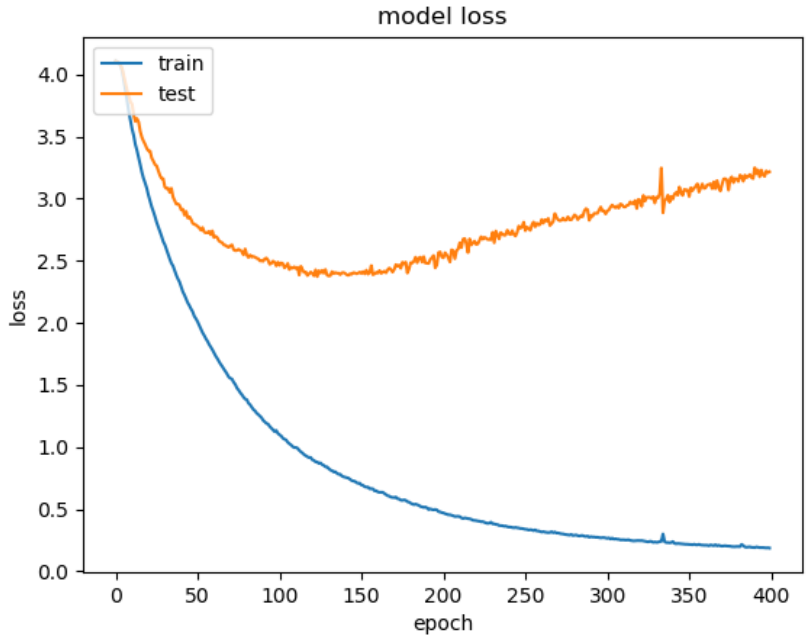
(Zdroj: Autor)

Příloha 4 - Ztrátová funkce modelu s inverzním vstupem pro GRU



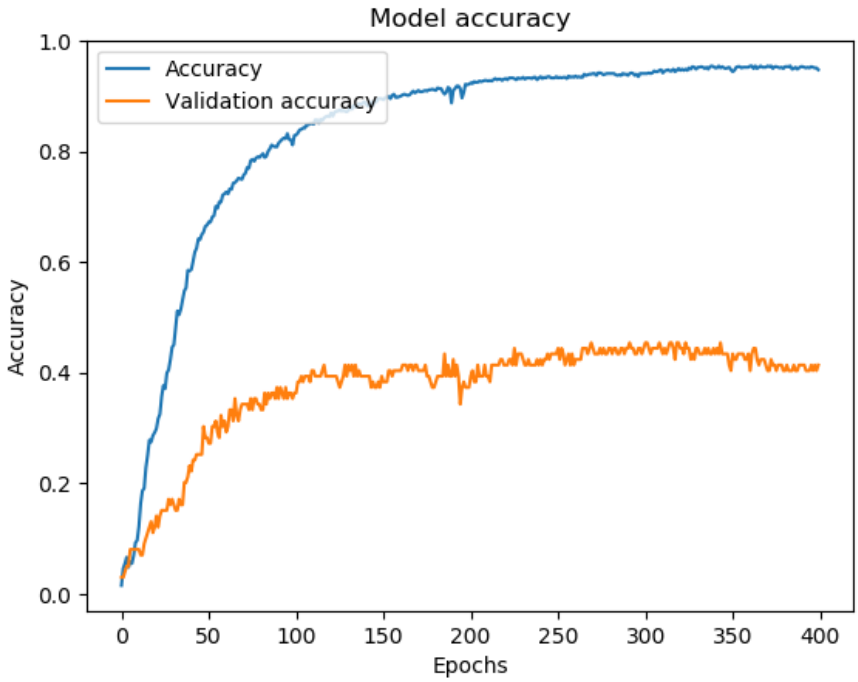
(Zdroj: Autor)

Příloha 5 - Ztrátová funkce dvouvrstvého modelu LSTM



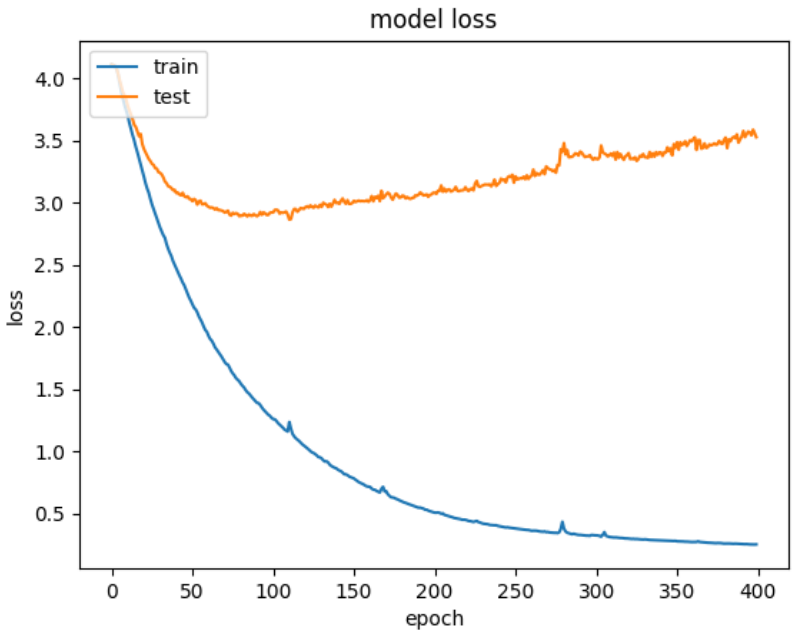
(Zdroj: Autor)

Příloha 6 - Ztrátová funkce dvouvrstvého modelu GRU



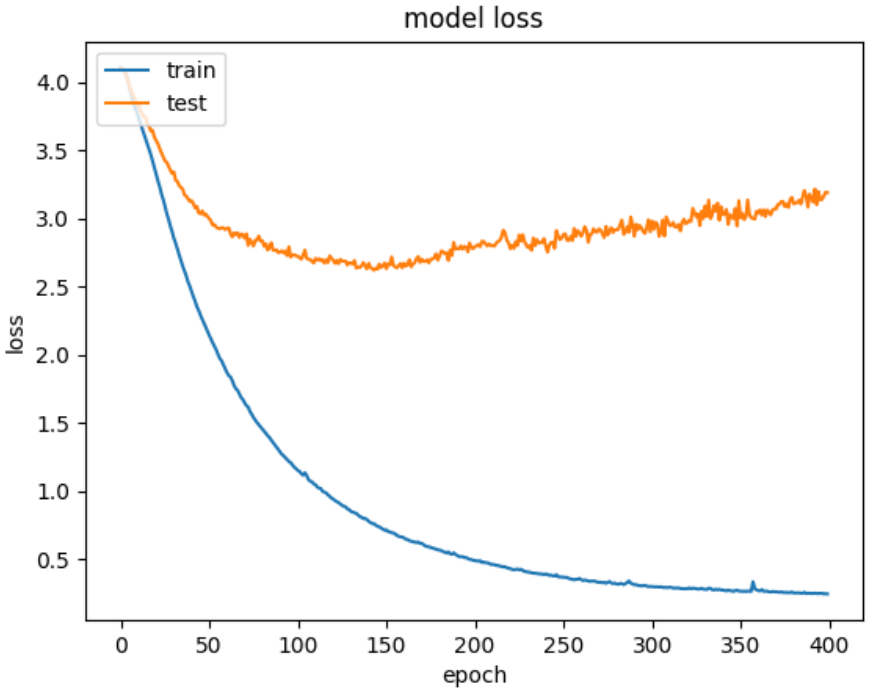
(Zdroj: Autor)

Příloha 7 - Ztrátová funkce modelu s dvouvrstvou strukturou modelu LSTM a regularizátorem L1



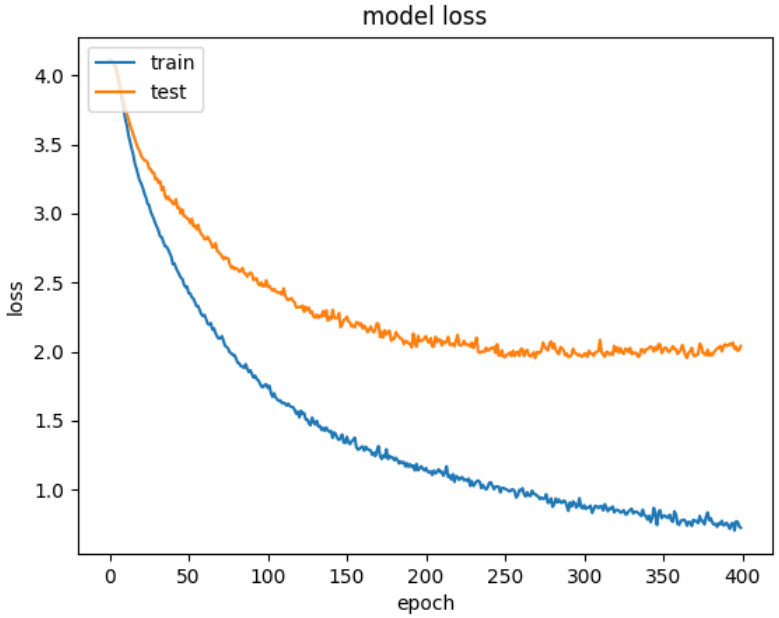
(Zdroj: Autor)

Příloha 8 - Ztrátová funkce modelu s dvouvrstvou strukturou modelu LSTM a regularizátorem L2



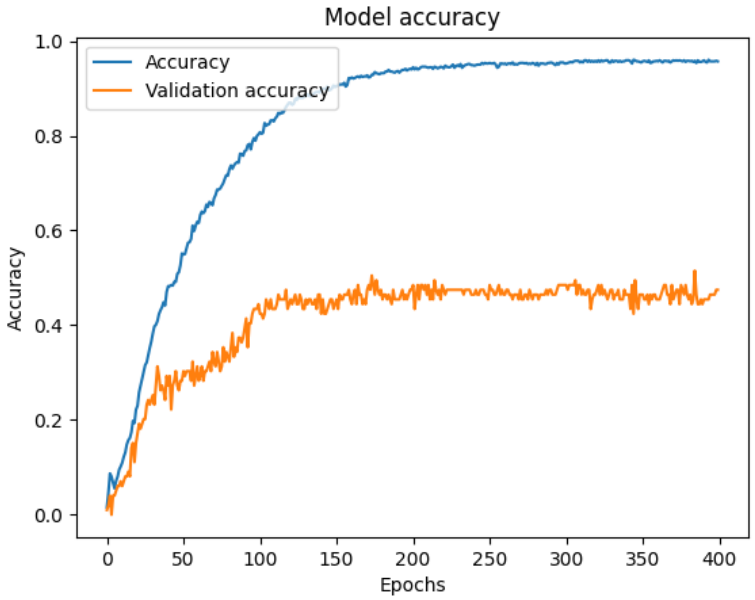
(Zdroj: Autor)

Příloha 9 - Ztrátová funkce modelu s dvouvrstvou strukturou LSTM a Dropout vrstvou



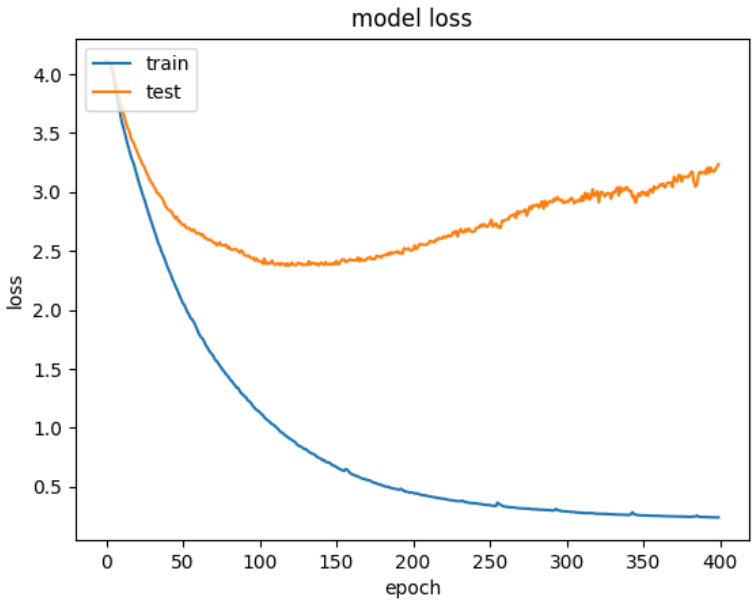
(Zdroj: Autor)

Příloha 10 - Přesnost modelu s dvouvrstvou strukturou LSTM a regularizátory L1, L2



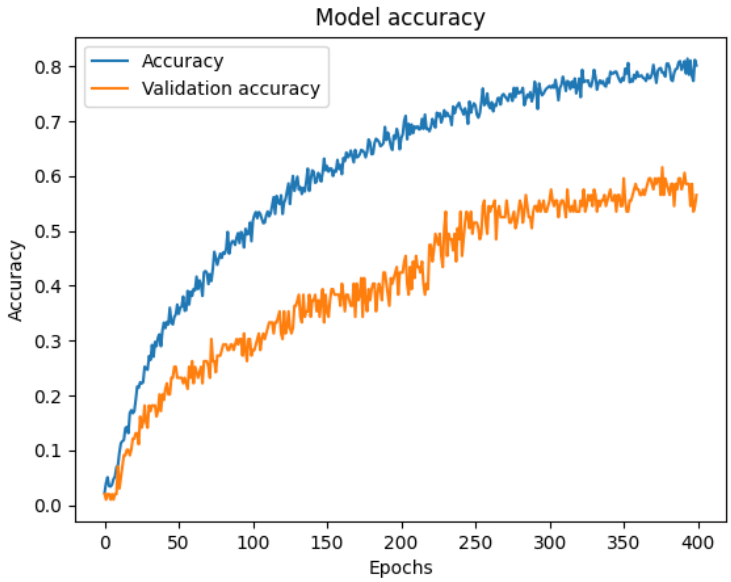
(Zdroj: Autor)

Příloha 11 - Ztrátová funkce modelu s dvouvrstvou strukturou LSTM a regularizátory L1, L2



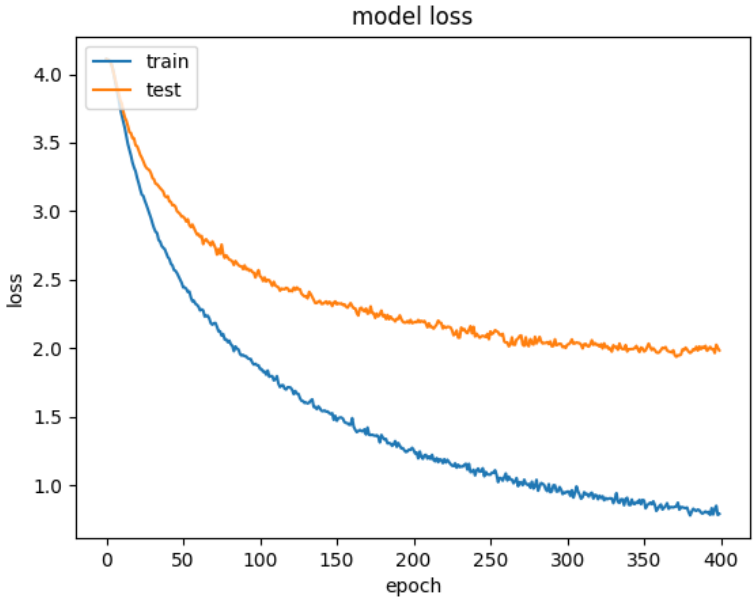
(Zdroj: Autor)

Příloha 12 - Přesnost modelu s dvouvrstvou strukturou LSTM, Dropout vrstvou a regularizátorem L1



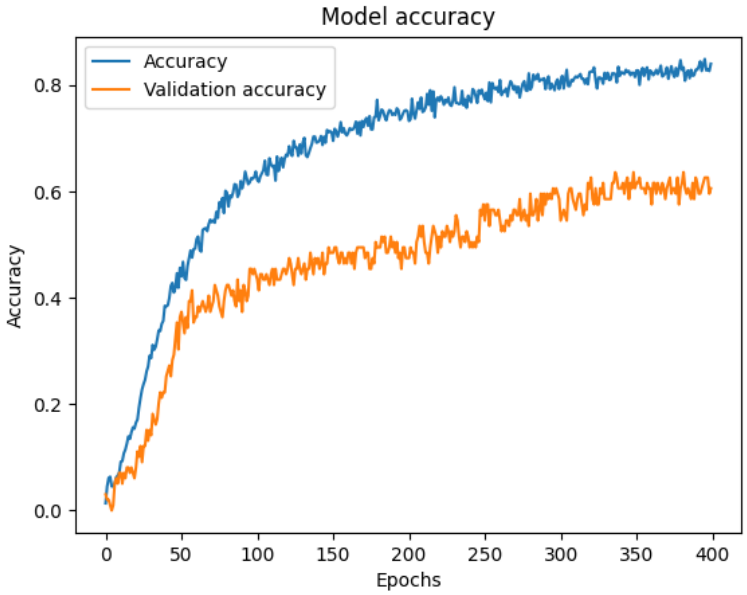
(Zdroj: Autor)

Příloha 13 - Ztrátová funkce modelu s dvouvrstvou strukturou LSTM, Dropout vrstvou a regularizátorem L1



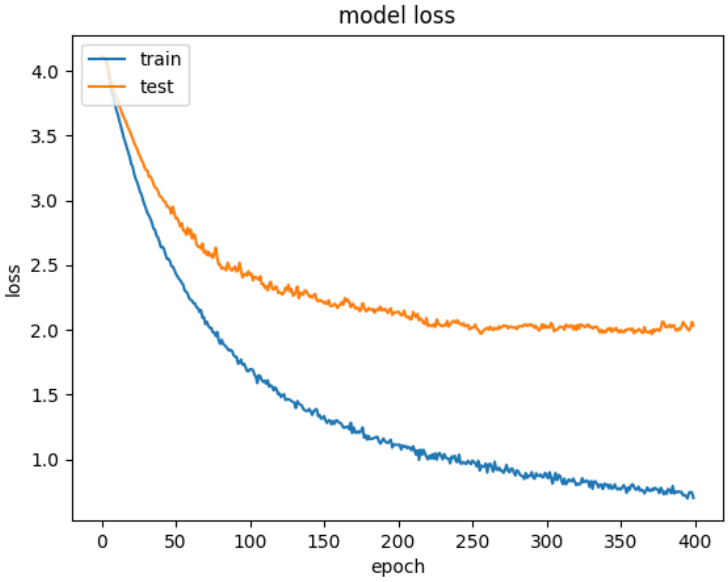
(Zdroj: Autor)

Příloha 14 - Přesnost modelu s dvouvrstvou strukturou modelu, Dropout vrstvou a reguralizátorem L2



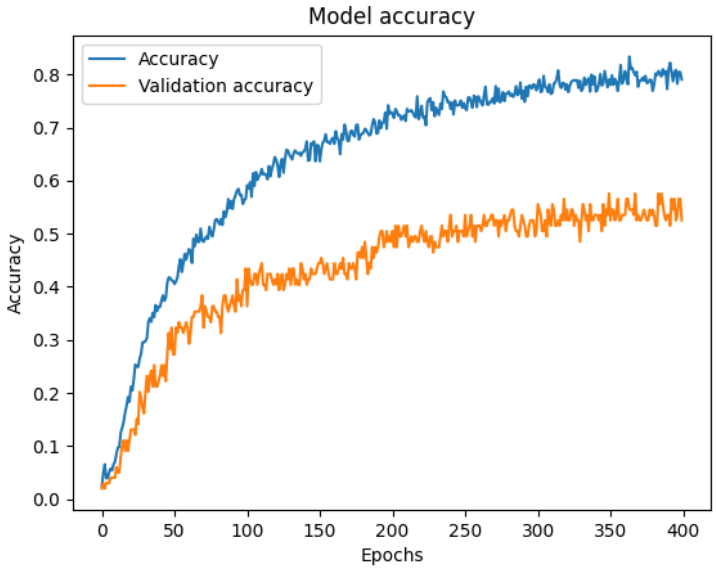
(Zdroj: Autor)

Příloha 15 - Ztrátová funkce modelu s dvouvrstvou strukturou LSTM, Dropout vrstvou a regularizátorem L2



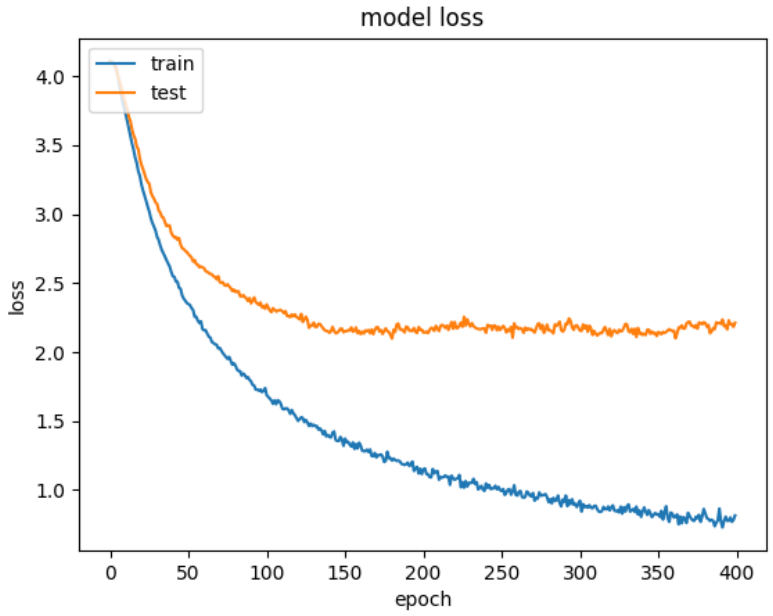
(Zdroj: Autor)

Příloha 16 - Přesnost model s dvouvrstvou strukturou LSTM, Dropout vrstvou a regularizátory L1, L2



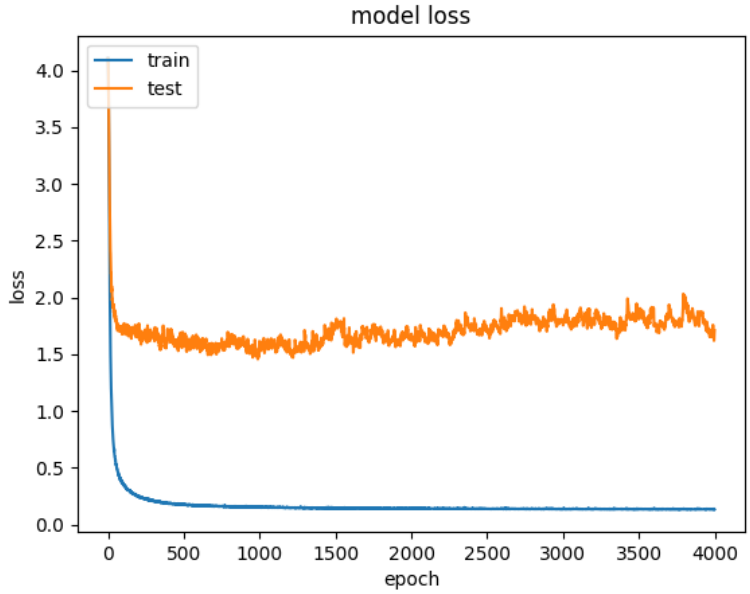
(Zdroj: Autor)

Příloha 17 - Ztrátová funkce modelu s dvouvrstvou strukturou LSTM, Dropout vrstvou a regularizátory L1, L2



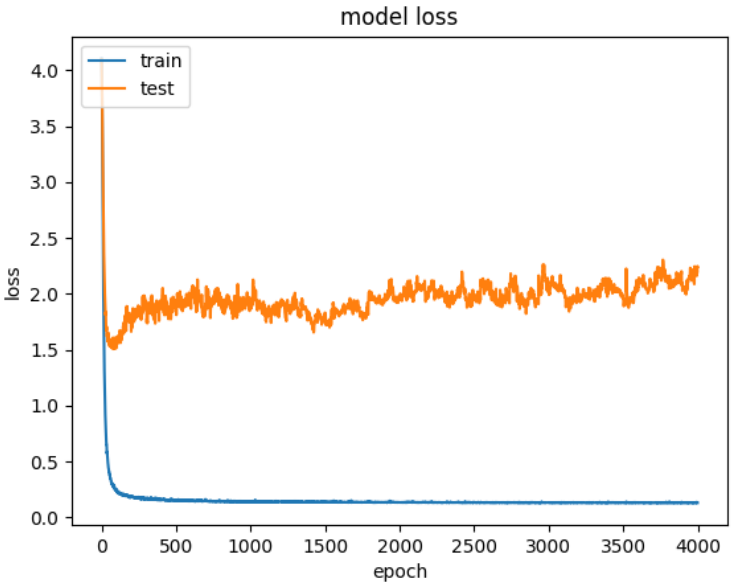
(Zdroj: Autor)

Příloha 18 – Ztrátová funkce modelu s dvojvrstvou strukturou modelu LSTM s Dropout vrstvou a regularizátorem L2 (4000 cyklů epoch)



(Zdroj: Autor)

Příloha 19 - Ztrátová funkce finálního modelu



(Zdroj: Autor)