Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Klasifikace textu konvoluční neuronovou sítí

Bc. Jiří Podivín

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Jiří Podivín

Systémové inženýrství a informatika
Informatika

Název práce

Klasifikace textu konvoluční neuronovou sítí

Název anglicky

Text classification with convolutional neural network

Cíle práce

Cílem práce je navrhnout a implementovat architekturu konvoluční neuronové sítě schopné klasifikovat rizika popsaná v databázi systému Evropské Unie RAPEX (Rapid Alert System for Non-Food Products), na základě slovního popisu závady nebo rizika.

Metodika

Student popíše současný stav výzkumu v oblasti konvolučních sítí a hlubokého učení.

Podá přehled oblastí, ve kterých byly konvoluční sítě úspěšně použity a popíše některé typické aplikace.

Dále popíše jakou podporu pro jejich realizaci poskytují softwarové systémy určené pro matematické modelování jako je Matlab nebo jeho volně šiřitelná verze Octave.

Student implementuje klasifikátor založený na konvolučních sítích a aplikuje jej na řešení klasifikace rizik v systému RAPEX.

Literatura:

K. Taylor: Deep learning using Matlab. Neural network applications

P. Kim: Matlab deep learning: With Machine Learning, Neural Networks and Artificial Intelligence

Doporučený rozsah práce

60

Klíčová slova

konvoluční neuronové sítě, klasifikace, keras, strojové učení

Doporučené zdroje informací

Gulli A., Pal S.: Deep learning with Keras, 2017

Kingma, Diederik P. and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” CoRR abs/1412.6980 (2015): n. pag.

ŠTĚPÁNKOVÁ, O. – MAŘÍK, V. – LAŽANSKÝ, J. *Umělá inteligence. (6) Vladimír Mařík, Olga Štěpánková, Jiří Lažanský a kolektiv.* Praha: Academia: Praha: Academia, 2013. ISBN 978-80-200-2276-9.

Zeiler, Matthew D.. “ADADELTA: An Adaptive Learning Rate Method.” CoRR abs/1212.5701 (2012): n. pag.

Předběžný termín obhajoby

2019/20 LS – PEF

Vedoucí práce

doc. Ing. Arnošt Veselý, CSc.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 25. 03. 2020

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Klasifikace textu konvoluční neuronovou sítí" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 25.3.2020 Jiří Podivín

Poděkování

Rád bych touto cestou poděkoval docentu Arnoštu Veselému za odborné vedení, doktoru Eduardu Bakšteinovi za odbornou konzultaci, Martě Podivínové za pravopisnou kontrolu a mé rodině za psychologickou podporu.

Klasifikace textu konvoluční neuronovou sítí

Abstrakt

Cílem práce je navrhnout konvoluční neuronovou síť k řešení úlohy klasifikace textových popisů rizik ze záznamů databáze Safety Gate Evropské unie, evidující nebezpečné nepotravinové produkty, za použití nástrojů knihoven TensorFlow a Keras.

Teoretická část práce je věnována popisu technik analýzy textu, popisu technik normalizace vstupních dat, a popisu metrik ohodnocení kvality modelů analýzy textu.

Teoretická část práce se dále věnuje popisu fungování neuronových sítí, s důrazem na techniky hlubokého učení. Pozornost je věnována předchozím pracím v oblastech klasifikace a zpracování textu.

Vlastní práce se skládá z popisu navrženého modelu, vstupních dat, postupu jejich zpracování a hardwaru použitého k učení modelu. Součástí vlastní práce je i ohodnocení výsledného modelu a popis postupu testování výkonnosti modelu.

Bylo zjištěno, že klasifikace textu konvoluční neuronovou sítí je možná a prakticky proveditelná. A to i za použití relativně jednoduchého modelu a minimálních výpočetních prostředků.

Klíčová slova: konvoluční neuronové sítě, klasifikace, keras, strojové učení

Text classification with convolutional neural network

Abstract

The work describes the design of a convolutional neural network capable of classifying descriptions of risks from the EU Safety Gate database of dangerous non-food products, using Keras and TensorFlow libraries.

The theoretical section describes common techniques of text mining, along with methods of text preprocessing, specifically the normalization of text datasets. The section also describes the principles and function of neural networks, along with metrics evaluating the performance of neural models. With an additional focus on methods in the area of deep learning.

Existing work in areas of text mining and classification is discussed.

The practical part describes the design, training, testing, and evaluation of the proposed model. Characteristics and preprocessing of the dataset are also described.

The work shows that the classification of selected texts by the convolutional neural network is both possible and practical, even with relatively limited resources and simple model.

Keywords: convolutional neural network, machine learning, keras, classification

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek a termínů

Termín	Vysvětlivka
Adadelta	Adaptivní optimalizační algoritmus
Adam	Adaptivní optimalizační algoritmus
CNN	Konvoluční neuronová síť
DSS	Systém pro podporu rozhodování
Embedding	Vnoření slov
Ensambling	Spojení více modelů k získání modelu výhodnějších vlastností
GRU	Gated Recurrent Unit, skrytá výpočetní jednotka používaná v RNN
Hyperparametr	Veličina určující vlastnosti modelu zvolená před zahájením procesu učení a dále neměnná
Keras	Knihovna funkcí neuronových sítí
Padding	Operace připojení prvků ke změně tvaru tensoru
Pandas	Knihovna funkcí pro manipulaci s daty
Parametr	Veličina určující vlastnosti modelu, podléhající změnám během procesu učení
RMSprop	Adaptivní optimalizační algoritmus
RNN	Rekurentní neuronové sítě
SGD	Stochastický gradientní sestup
Tensor	Stavební blok modelů vycházející ze stejnojmenného matematického objektu
Tensorboard	Program vytvářející vizualizace modelů, jejich vlastností a průběhu procesu učení
TensorFlow	Knihovna funkcí tensorového počtu
Tokenizace	Proces nahrazení slov zástupnými symboly
Validace	Ověření, ověření vlastností modelu
Vrstva	Funkčně definovaná komponenta modelu

1 Úvod

Během poslední dekády prošel obor umělé inteligence obecně, a oblast strojového učení zvláště, značným rozvojem. Řada závažných teoretických problémů, dříve bránících aplikaci známých technik umělých neuronových sítí na obtížné problémy, byla vyřešena.
Znatelný pád cen výkonného hardware dále umožnil využít již dříve známé i nově objevené přístupy v oblastech, kde to dříve nebylo prakticky možné.

V neposlední řadě napomohly rozvoji oboru investice ze strany velkých podniků z oblastí internetových služeb, telekomunikací, prodeje a reklamy, jejichž předmět podnikání vyžadoval stále rychlejší zpracování velkých dat.
Zvláště těsný vztah mezi reklamou, prodejem produktů a monitorováním chování potenciálních zákazníků, vedl k nutnosti rozvoje automatické analýzy textu, obrazu, zvuku a dalších druhů dat vytvářených uživateli daných platforem.

Vývoj i nasazení nástrojů využívajících strojového učení je tak dnes, v důsledku zmíněných sil, snadnější a levnější než kdy dříve. S využitím služeb podniků z oblasti cloud computingu, například Google Cloud, AWS nebo Azure, je navíc možné snadno nasazovat vyvinutá řešení i v nedostatečně vybavených lokalitách.

Mezi možnými uživateli strojového učení není možné zapomenout na veřejný sektor. Státní správa České republiky, i mezinárodní organizace, vyžadují ke svému fungování rychlé zpracování značného objemu dat.

Chyby zpracování dat, jakékoliv původu, mají přitom velmi závažné důsledky. A to i v případě relativně prostých operací. Veřejný sektor proto již dlouho využívá nástrojů strojového učení pro automatizaci úkonů vyžadujících značné soustředění, minimální aktivitu a co nejvyšší rychlost.

Již od devadesátých let jsou techniky strojového učení používány pro čtení adres a následného třídění listovních zásilek poštou USA. (Denker, a další, 1989) Přičemž automatické rozeznávání znaků a textu má v oblasti pošty o několik dekád delší historii. (Office of Technology Assessment, 1984)

Mnoho dalších činností podobného rázu je zvláště vhodné pro aplikaci technik strojového učení, například zadávání údajů do databází nebo vyplňování formulářů, jejichž pole spolu navzájem souvisí.

V některých případech musí zaměstnanec úřadu, popřípadě jiná osoba vyplňující formulář, dvakrát nebo vícekrát zaznamenat stejnou informaci pro potřeby pozdějšího zpracování. Možnost lidské chyby je přitom nezanedbatelná, ať už během samotné činnosti, nebo během interpretace výstupu jinou osobou.

Nevhodně zaznamenané a interpretované informace mohou být dokonce naprosto nesmyslné, a tak snižují hodnotu celého dokumentu nebo databáze. Automatizace alespoň některých ze zmíněných úkonů, by nejenom zjednodušila práci lidí, ale i zvýšila její kvalitu.

1.1 Překážky rozvoje

Širšímu nasazení metod strojového učení jsou ovšem kladeny nezanedbatelné překážky, a to jak pro veřejný, tak i soukromý sektor.

Jedná se nejenom o problémy technického, ale hlavně společenského rázu. Vlastnosti některých technik strojového učení jsou pro laickou veřejnost těžko uchopitelné. Stejně jako potenciální užitek z vyvinutých nástrojů a jejich možnosti.

Jistým dílem přispívá ne zcela pozitivní obraz umělé inteligence v produktech zábavního průmyslu a také nepřesné informování o technologiích v populárně naučné sféře.
V posledních letech navíc pozornost veřejnosti upoutala řada incidentů nevhodného použití umělé inteligence, které byly umocněny intenzivním, a mnohdy nepříliš spolehlivým mediálním pokrytím.

Pozornost světové veřejnosti například upoutala skupina nástrojů, známých také jako DeepFake, umožňujících kombinovat několik obrazových záznamů do relativně snesitelného, ale stále nedokonalého celku. Jedním z nejznámějších aplikací našly nástroje

DeepFake v oblasti pornografie, neboť umožňovaly vkládat charakteristiky, například tváře, veřejně činných osob, především celebrit, do existujících pornografických produkcí.

Vzniklé koláže byly sice evidentně neautentické, nicméně veřejné mínění, vedené reakcemi některých zainteresovaných osob, donutilo stažení původního nástroje z řady internetových platforem, kde byl publikován.

Někteří komentátoři poukazovali na možnost užití DeepFakes technologie k šíření dezinformací a propagandy. Fenomén DeepFakes vyvolal i reakce v politické sféře, včetně legislativních návrhů. (Mosley, 2019) (House of Representatives, 2019)

Dalším, nechvalně proslulým případem využití umělé inteligence, byl chatbot firmy Microsoft, nazvaný Tay, jenž měl reagovat a napodobovat komunikaci uživatelů platformy Twitter. (Sinders, 2016) (Lee, 2016) Návrh Tay ovšem nepočítal s monitorováním komunikace uživatelů a redakční úpravou odpovědí chatbotem generovaných.

Po krátké době začal chatbot, relativně přesně, napodobovat styl komunikace, kterému byl vystaven. Generované tweety byly ovšem, stejně jako podstatná část zpráv uživatelů, naplněn vyjádřeními rasistickými a sexistickými, s nezanedbatelným zlomkem vulgarit.

Reakce veřejnosti byla kombinací pobavení a odsouzení. Z některých chatbotem odeslaných zpráv se staly základy vtipů. (Know Your Meme, 2020) Provoz Tay byl záhy ukončen a vedení Microsoftu se za událost veřejně omluvilo.

1.2 Problém vysvětlitelnosti

Na pomezí problémů společenských a technických, kterým obor umělé inteligence čelí, se nachází otázky kontroly, interpretace a pochopení výstupů technik strojového učení, známý také jako problém vysvětlitelné umělé inteligence.[1]

[1] Označováno také jako Explainable artificial intelligence

Otázkou přitom není pouhá interpretace výsledků samotných a pochopení principu jejich získání, ale především objasnění, proč systém dosáhl určitého výsledku.
Vysvětlení přesného postupu dosažení výstupů některých technik strojového učení je přitom téměř nemožné a v některých případech může i zhoršit kvalitu řešení. (Lipton, 2016)

Obtížná vysvětlitelnost řešení poskytnutých technikami strojového učení vedla již ke vzniku prvních legislativních opatření na celoevropské úrovni. Efektivita i odůvodněnost vzniklé legislativy je ovšem sporná. A to jak z důvodů technických, tak legálních. (Edwards & Veale, 2017)

2 Cíl práce a metodika

2.1 Cíl práce

Cílem práce je navrhnout konvoluční neuronovou síť k řešení úlohy klasifikace popisů rizik ze záznamů databáze Safety Gate Evropské unie. Databáze eviduje nepotravinové produkty, prodávané na trzích EU, vykazující vlastnosti ohrožující zdraví nebo bezpečnost spotřebitelů, jejich původ, účel a krátký slovní popis.

2.1.1 Databáze Safety Gate

Databáze Safety Gate, evidující nebezpečné nepotravinové výrobky vyskytující se na trzích států EU, je součástí informačního systému RAPEX, umožňujícího rychlé vydání varování a výměnu informací o nebezpečných produktech napříč státy EU a EEA. Právní základy systému pocházejí z roku 2001, spolu s Obecnou Směrnicí o Bezpečnosti produktů Evropského parlamentu a komise EU. (European Parliament, Council of the European Union, 2001)

Hlášení o rizikových produktech jsou do systému odesílány podniky, a dále jsou zpracovány příslušnými orgány jednotlivých států unie. Úřady států mají dále povinnost předat zpracovaná hlášení pro další posouzení. Nakonec jsou schválená hlášení uvedena do databáze. Přesné povinnosti všech aktérů se liší v závislosti na závažnosti rizika, které produkt působí, původu produktu a dalších faktorech. (EUROPEAN COMMISSION, 2019) Samotné ohodnocení míry rizika závisí na řadě faktorů, včetně zranitelnosti spotřebitele a závažnosti potenciálně způsobeného zranění. Nicméně značný počet potenciálních ohlašovatelů, několikanásobné předávání informací mezi zodpovědnými orgány a složitost požadovaného postupu ohodnocení a ohlášení, zvyšuje pravděpodobnost lidské chyby.

Styl popisů a kvalita informací záznamů tedy není konzistentní. Pokyny pro ohlašovatele, dostupné na webovém portálu Safety Gate, jsou relativně jednoduché, nicméně postrádají přesné definice v oficiálním dokumentu Evropské Komise. (EUROPEAN COMMISSION, 2018)

Nežádoucí vlastnosti produktů jsou rozděleny do kategorií podle negativního dopadu, který mohou mít na spotřebitele. Kategorie sahají od velmi obecných, jako zranění nebo oheň, k velmi specifickým, jako je elektromagnetické rušení.
Evidované produkty mohou mít přiřazeny jednu nebo více kategorií.
Ale přesný postup, podle kterého jsou kategorie určeny, není evidentní ani z pokynů pro ohlašovatele, ani z rozhodnutí Evropské komise. Definice některých kategorií se navíc překrývají.

V některých případech se nejasnost významu projevuje v záznamech databáze.
Rizika některých produktů jsou kategorizována nekonzistentně, a to i vzhledem k textu popisu rizika. Například rizika specifické pyrotechniky mohou být zařazena do kategorie zranění, ale bez uvedení popálenin a ohně, ačkoliv jsou v popisu uvedeny. Přičemž samotné přiřazení více kategorií jednomu nebezpečnému produktu není samo o sobě nežádoucí, ale je dokonce nutné ke splnění účelu databáze.
Je evidentní, že stávající postup neposkytuje vždy spolehlivé záznamy a vyžaduje značné úsilí na straně personálu, zejména při zadávání údajů a udržování jejich konzistence napříč požadovanými položkami

2.1.2 **Automatická klasifikace rizik**

Část z úkonů, nyní prováděných manuálně, je ovšem možné zjednodušit s využitím technik strojového učení. Kategorizace rizik, působených produktem, je právě takovým úkonem. Samotný slovní popis rizika umožňuje člověku snadno pochopit podstatu nalezené závady, nicméně kategorie, ze slovního popisu nutně vycházející, jsou zásadní pro usnadnění analýzy dat obecně a vyhledávání v databázi zvláště.
Automatická klasifikace rizik, využívající jejich člověkem psaný popis jako vstup, by mohla podstatně ulehčit práci úředníkům a současně zlepšit kvalitu databáze.

Vlastnosti databáze vyžadují, aby navržená neuronová síť byla schopna přiřadit každému produktu ne jednu, ale více kategorií, odpovídajících popisu rizika produktem působeného.
Vstupem je přitom pouze člověkem psaný, řádově desítky slov dlouhý, slovní popis rizika.

2.2 Metodika

Základní analýza dat proběhla v prostředí Microsoft Excel z balíku Microsoft Office 2016. Software byl zvolen především vzhledem k jednoduchosti základních databázových operací, umožňující rychlé ověření vlastností tabulek a získání informacích nutných k vývoji modulu předzpracování dat.

Vývoj programu k předzpracování dat, verifikací předpokladů nutných k použití modelů, tvorbě modelů samotných a jejich následného provozu a zhodnocení, probíhal v prostředí Visual Studio 2019 Community Edition s použitím jazyka Python verze 3.6 (Python Software Foundation, 2019).

Základní knihovny jazyka Python sloužily především k práci se souborovým systémem a vývoji nižších úrovní funkcí předzpracování dat. Předzpracování dat proběhlo s použitím knihoven Scipy (SciPy developers, 2019), specificky Pandas (Pandas developers, 2019) a Numpy (NumPy developers, 2019).

Vývoj modelů samotných využíval platformu TensorFlow (Alphabet Inc., 2019), s funkcemi a nástroji knihovny Keras (Keras Team, 2019) sloužícími pro práci s vyššími stupni abstrakcí modelů a jejich součástí. Hlavním důvodem pro volbu nástrojů Keras s backendem TensorFlow byla jednoduchost tvorby, analýzy a správy vytvářených modelů. Druhým významným důvodem byla podrobnost poskytované dokumentace a rozsáhlá komunita uživatelů.
K předběžné analýze kvality navržených modelů, ohodnocených metrikami založenými na matici záměn, byl použit Microsoft Excel. Podrobná analýza vlastností navržených modelů, včetně jejich struktury a kvality výstupu, proběhla v programu Tensorboard, dodávaném spolu s platformou TensorFlow. Program Tensorboard také posloužil k tvorbě některých vizualizací navržených modelů v přílohách práce.

Obsažené tabulky a obrazové materiály byly vytvořeny autorem práce, pomocí programů Excel, Word a Tensorflow. Vizualizace Eggholder funkce byla vytvořena v prostředí Octave, programem obsaženým v přílohách práce. Uvedené rovnice a funkce byly převzaty z příslušné literatury.

Soubory projektu, včetně zdrojového kódu, formátovaných a neformátovaných vstupních dat, parametrů navržených modelů a vybraných výstupů programu, byly spravovány systémem kontroly verzí git. Vzdálený git server a úložiště souborů bylo zdarma poskytnuto jako součást studentské verze programu Azure DevOps společnosti Microsoft.

3 Umělá inteligence

Předmět analýzy textu zapadá do širšího oboru zpracování přirozeného jazyka, a obecněji do oborů zabývajících se extrakcí informace. Ačkoliv se jedná o úlohu v mnoha podstatných ohledech odlišnou od známějších příkladů statistických problémů, řídí se stejnými základními pravidly a používá do značné míry shodné názvosloví.

Analýza textu je také jedním z nejznámějších využití umělé inteligence[2]. Rozvoj oboru během posledních deseti let byl umožněn, mimo jiné, zvýšenou dostupností rozsáhlých[3] množin dat strojem zpracovatelného textu.

Druhým významným faktorem nedávného rozvoje technik umělé inteligence obecně je potřeba velkých podniků, států a nadnárodních organisací zpracovat a analyzovat velká množství dat vytvářených moderními komunikačním prostředky obecně[4] a jejich lidskými uživateli specificky.

3.1 Klasifikace úloh umělé inteligence

Úlohy analýzy textu, někdy také nazývaného text mining[5], se dají zobecnit na stejné kategorie, používané pro označení úloh napříč oborem strojového učení. Jedná se o úlohy klasifikace, regrese a shlukování.

[2] Nazývané také strojové učení, machine learning atd. použití těchto pojmů závisí spíše na osobních názorech autorů publikací než na všeobecně uznávaném standardu. Viz široce rozšířený citát „Součástí historie oboru umělé inteligence je, že kdykoliv někdo přišel na to, jak přimět počítač něco dělat – hrát dobře dámu, řešit jednoduché, ale relativně neformální úlohy – objevil se sbor kritiků prohlašujících: to není přemýšlení!“ (McCorduck, 2004)
[3] Rozsahu jednoho a více gigabytů
[4] Moderní síťová infrastruktura generuje značný objem dat i během klidového provozu, bez nutnosti lidské činnosti
[5] Viz obecné označení technik zpracování a extrakce informací z velkých objemů dat data mining

3.1.1 **Regresní úlohy**

Cílem regresních úloh je vytvořit model $f(X, \beta) \approx Y$, reprezentující odhad vztahu hodnot jedné nebo více závislých proměnných Y a hodnot nezávislých proměnných X v podobě parametrů β jejichž hodnotu je třeba určit.
Samotný odhad hodnot parametrů a struktura modelu závisí na zvoleném algoritmu. Mezi nejznámější patří Běžná metoda nejmenších čtverců nebo regrese logistická.

3.1.2 **Klasifikační úlohy**

Klasifikační úlohy se obecně zabývají určením příslušnosti objektů z množiny pozorování do jedné nebo více z n známých podmnožin, kategorií nebo tříd, výpočtem n hodnot $p_i \in \langle 0,1 \rangle$ vektoru $\bar{P}$, pro danou hodnotu vstupu, kde n je rovno počtu možných kategorií nebo tříd a p_i značí pravděpodobnost příslušnosti pozorování, nebo objektu, reprezentovaného hodnotami nezávislých proměnných do dané třídy. Vektor pravděpodobností příslušnosti je dále zpracován postupem závislým na zadané úloze.

Klasifikační úlohy dále podrobněji klasifikujeme podle délky vypočítaného vektoru a počtu označených tříd. (Chollet, 2019) Mezi nejobvyklejší patří zvolení prvku p_k s nejvyšší hodnotou pravděpodobnosti. Výstupem modelu je hodnota indexu k, značící příslušnost vstupu do stejně označené kategorie. Existují ale i úlohy vyžadující označení více kategorií.

	Počet označených kategorií	
Rozměr vektoru $\bar{P}$	1	$\langle 0,1 \rangle$
2	Klasifikace do dvou tříd s jediným označením *(Single-label, binary)*	Klasifikace do dvou tříd s více označeními *(Multilabel, binary)*
> 2	Klasifikace do více tříd s jediným označením *(Single-label, multiclass)*	Klasifikace do více tříd s více označeními *(Multilabel, multiclass)*

Tabulka 1 Rozdělení klasifikačních úloh podle počtu tříd a označení

Hlavním rozdílem oproti regresním úlohám je právě kvalitativní, kategorický výstup modelu, zatímco v případě úloh regresních se jedná o výstup kvantitativní.

V závislosti na zvoleném algoritmu může být klasifikační úloha převedena na ekvivalentní úlohu regresní, a rozdíly mezi úlohami skryty za dodatečnou vrstvou abstrakce. (Veselý, 2012)

3.1.3 **Úlohy shlukové analýzy**

Cílem úloh shlukové analýzy je rozklad množiny objektů χ na více neprázdných disjunktních podmnožin podle vybraných charakteristik objektů, maximalizovat podobnost mezi objekty vzniklé podmnožiny a současně minimalizovat podobnost s objekty mimo podmnožinu

$$\chi = \bigcup_{i}^{m} C_i$$

$$C_i \subset \chi$$

Kde χ značí původní množinu objektů a C_i jeden z vytvořených shluků.
Vzniklé podmnožiny nazýváme shluky. Charakteristiky, narozdíl od úloh regresních a klasifikačních, jsou úlohy shlukové analýzy, příkladem učení bez učitele. Metody shlukové analýzy se obecně dělí na hierarchické a nehierarchické.
Hierarchické metody jsou sérií vnořených rozkladů, začínajících rozkladem na jednoprvkové podmnožiny původní množiny χ, a konče rozkladem na jednu podmnožinu, obsahující všechny objekty χ.
Nehierarchické metody vytvářejí systém vzájemně různých disjunktních podmnožin původní množiny χ. (Kelbel & Šilhán, 2002)

3.2 **Metriky kvality metod umělé inteligence**

Různorodost úloh vedla ke vzniku celého souboru metrik určování kvality, přesnosti, výkonnosti a dalších vlastností metod použitých k jejich řešení. V závislosti na specifících jednotlivých úloh, může být nutné posoudit kvality aplikované metody mnoha metrikami, s často protichůdnými cíli.
Znalost vlastností použitých metrik je zásadní pro jejich posouzení. Bohužel názvosloví není vždy jednoznačné a může být i zavádějící.

3.2.1 **Chybová funkce**

Ze všech metrik kvality modelů strojového učení je chybová funkce[6] tím nejpřímočařejším a zároveň nejnáchylnějším k misinterpretaci. Chybová funkce je matematickou funkcí, reprezentující rozdíl mezi očekávanou a odhadovanou hodnotou závislé proměnné.
Funkce je zvolená v závislosti na zadané úloze a zvoleném modelu. Může se jednat o součet čtverců odchylek, manhattanskou vzdálenost, součet absolutních hodnot odchylek nebo o křížovou entropii.

Volba by měla být motivována dvěma hledisky:

1. Vážnost následků nepřesného odhadu
2. Matematické vlastnosti vybrané funkce

Chybové funkce mohou vykazovat asymetrie, které mohou ovlivňovat významnost některých odchylek na úkor jiných. Příkladem může být střední kvadratická chyba[7]:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Kde Y_i značí očekávanou hodnotu závislé proměnné, $\hat{Y}_i$ odhadovanou hodnotu závislé proměnné a n počet pozorování, pro které byla hodnota závislé proměnné vypočítána.
Nevýhodou střední kvadratické chyby je citlivost na odlehlá a extrémní pozorování.
Znalost specifik chybové funkce je zásadní pro správnou interpretaci výsledků modelu. Nicméně ani vhodně zvolená a interpretovaná hodnota chybové funkce, není sama o sobě dostatečná k řádnému posouzení kvality modelu, řešícího klasifikační úlohu a potažmo jeho dalšímu vývoji.

3.2.2 **Metriky založené na matici záměn**

Oproti chybové funkci jsou metriky odvozené z matice záměn[8] vhodné k ohodnocení kvality klasifikátoru. Umožňují také snadnější posouzení důsledků použití klasifikátoru v praxi.

[6] V anglickém jazyce loss function, v literatuře někdy ztrátová funkce
[7] Dále označená anglickou zkratkou MSE
[8] V anglickém jazyce confusion matrix

Ve své nejjednodušší podobě sestává matice záměn ze čtyř polí. Jedno pro každou situaci, která může nastat během řešení úlohy binární klasifikace.

	Skutečná kategorie		
Předpovídaná kategorie	$n = a + b + c + d$	+	-
	+	a	b
	-	c	d

Tabulka 2 Binární matice záměn

Metriky jsou vypočítány kombinací hodnot v polích. Nejpoužívanější vzorce jsou uvedeny v následující tabulce.

Základní metriky odvozené z matice záměn	
Přesnost (Accuracy)	$\frac{a+d}{a+b+c+d}$
Senzitivita (Sensitivity, Recall)	$\frac{a}{a+c}$
Preciznost (Precision)	$\frac{a}{a+b}$
Specificita (Specificity)	$\frac{d}{d+c}$
F-míra (F-measure)	$\frac{2*a}{2*a+b+c}$

Tabulka 3 Metriky odvozené z matice záměn

Uvedené metriky tvoří základ složitějších indikátorů kvality modelu, jakou je například ROC křivka, informovanost nebo F-M index[9]. Stejně tak je možné, s určitými omezeními, zobecnit některé z metrik na úlohy nebinární klasifikace.

Ve všech případech je ale nutné mít na paměti specifika úlohy. Senzitivita může být, v některých případech, důležitější než preciznost a naopak. Zavedení hodnoty ceny nebo také následků, rozhodnutí klasifikátoru jako proměnné při výpočtu metrik, může interpretaci výsledků podstatně zjednodušit.

[9] Fowlkes–Mallows index

3.3 Klasifikace algoritmů umělé inteligence

Algoritmy strojového učení se dělí podle míry informace o požadovaném výstupu modelu na tři základní kategorie: učení s učitelem, učení bez učitele a učení posilovací.[10]

Implementace daného algoritmu, spolu s typem úlohy a zvolenou formou uživatelského rozhraní, nakonec určuje formu výstupu. Od zmíněného vektoru pravděpodobností, přes binární pravdivostní hodnoty, až ke slovnímu potvrzení určené kategorie.

Správnou funkci algoritmu je nutné zajistit splněním předpokladů jeho použití a vhodně připravenou množinou dat. K ověření funkce modelu je třeba zvolit metriky s ohledem na úlohu a princip fungování algoritmu. Nevhodné metriky mohou v lepším případě poskytovat informace nesmyslné a v horším případě informace zavádějící. Pozornost je třeba věnovat i samotné implementaci metrik a API vybraných nástrojů.

Většina algoritmů umožňuje modifikovat celou řadu aspektů[11] procesu učení. Velké množství možných kombinací ale vynucuje ověření správné funkce algoritmu ještě před zahájením hledání jejich hodnot.

3.3.1 Učení s učitelem

Algoritmy spadající mezi metody učení s učitelem vyžadují data ve formě množiny příkladů, kde je ke každému vektoru vstupních hodnot přiřazena hodnota očekávaného výstupu.

Očekávaná hodnota je srovnána se skutečným výstupem modelu. Výsledek srovnání je použit v dalším kroku algoritmu k úpravě modelu.

Ke srovnání hodnot očekávaného a skutečného výstupu je použita chybová funkce zvolená s ohledem na specifika úlohy. Mezi nejznámější metody učení s učitelem patří například lineární a logistická regrese, metoda podpůrných vektorů[12] nebo perceptronový algoritmus.

[10] V anglickém jazyce supervised learning, unsupervised learning, reinforcement learning. Učení bez učitele je také nazýváno učením bez klasifikovaných příkladů. Posilovací učení je někdy nazýváno učením zpětnovazebním nebo posilovaným.
[11] Neboli hyperparametrů
[12] V anglickém jazyce Support Vector Machines

3.3.2 **Učení bez učitele**

Na rozdíl od učení s učitelem, nepoužívají algoritmy učení bez učitele informaci o správném výstupu k tvorbě modelu. Namísto toho nacházejí v datech nové vzory a struktury, s pomocí vybraných charakteristik dat samotných.
Nacházejí hlavní využití v předzpracování dat, snižování jejich dimenze, extrakci nových rysů[13]a dalších úlohách, kde není možné předem odhadnout hodnoty očekávaného výstupu. Nevýhodou algoritmů učení bez učitele je nutnost interpretovat nalezené struktury a vzory. Nevhodná interpretace může znehodnotit výsledky analýzy.
Mezi nejznámější příklady učení bez učitele patří shluková analýza, detekce hran a analýza hlavních komponent. Samotná analýza hlavních komponent je jedním ze základních nástrojů snížení dimenze dat a předzpracování vstupu modelu (Novovičová, Pudil, & Somol, 2013)

3.3.3 **Učení posilovací**

Posilovací učení je v mnoha ohledech podobné učení s učitelem a současně nejbližší obecně chápanému významu slova učení. Metody posilovacího učení upravují chování softwarového agenta k maximalizaci určitého druhu „odměny“, například počtu nasbíraných bodů ve virtuálním prostředí. (Sutton & Barto, 2018) Vytvořený model má tedy nepřímý přístup k informaci o očekávaném výstupu.

Nejznámějším využitím posilovacího učení je tvorba agentů ke hře složitých her, vyžadujících dlouhodobé plánování a zohlednění více faktorů prostředí, například Go nebo Starcraft. (Google DeepMind, 2019) (Google DeepMind, 2019)

[13] Neboli extrakci příznaků

4 Předzpracování textu

Zásadním krokem během tvorby modelu, nehledě na druh úlohy a zvolený algoritmus, je takzvané předzpracování, v angličtině preprocessing, tedy zajištění konzistentního a validního vstupu modelu.
Nevhodné nebo nedostatečné očištění vstupu může mít katastrofální důsledky na kvalitu výsledného modelu. V nejlepším případě utrpí přesnost získaných informací, v případě nejhorším bude model poskytovat informace zdánlivě hodnověrné, ale ve skutečnosti nepřesné nebo naprosto nesmyslné.

Řada nedávných, vesměs nechvalně proslulých případů nevhodné aplikace strojového učení, vycházela právě z nedostatků procesu ošetření vstupu modelu, respektive jeho trénovací množiny. (Clune, Lehman, & Misevic, 2018) (Knight, Biased Algorithms Are Everywhere, and No One Seems to Care, 2017)

Očištění vstupu někdy nazýváme předzpracováním nebo normalizací, v závislosti na kontextu a druhu úlohy využívající vstup. Celá řada postupů používaných pro očištění vstupu je společná, ale s jistými úpravami, většině algoritmů pro vytváření modelů, včetně obecně známých jako je například logistická regrese nebo BMNČ.

Zmíněný fakt často umožňuje vytvoření více modelů různými postupy pro vzájemné ověření a porovnání jejich funkce a efektivity, s využitím stejné množiny vstupních dat.
V své podstatě můžeme celý úkol rozdělit na dvě části: odstraňování nevhodných vstupních hodnot a zajištění vlastností vstupu zlepšujících výkonost modelu.

4.1 Odstranění nevhodných vstupních hodnot

Pro úlohy analýzy textu se k odstraňování nevhodných hodnot využívá široce rozšířených softwarových nástrojů, jejichž přesné nastavení a užití závisí na specifících dané úlohy. Jenom pro jazyk Python existuje celá řada modulů, vyvinutých právě k předzpracování textu. (Davydova, 2018)

Hlavní uplatnění zde nacházejí takzvané regulární výrazy, s jejichž pomocí lze efektivně vyhledat a popřípadě nahradit všechny výskyty nežádoucích řetězců v textu.
Je nutné upozornit, že většina moderních nástrojů, využívajících regulární výrazy, svými možnostmi neodpovídá definici regulárních jazyků podle teorie formálních jazyků, a umožňují definovat i gramatiky popisující jazyky bezkontextové.

Regulárním výrazem je dnes v oblasti vyhledávání řetězců rozuměn jakýkoliv vzor umožňující definovat množinu řetězců, se syntaxí vycházející z původního standardu POSIX. V následujícím textu budeme termín regulární výraz používat právě tímto způsobem.

Řetězec považujeme za nežádoucí, pokud nijak nepřispívá k informačnímu obsahu daného textu ve vztahu ke specifické úloze. Je proto zvláště důležité správně identifikovat požadavky dané úlohy, neboť stejný vstupní text může v závislosti na úloze projít řadou rozdílných úprav, vedoucích k značně odlišným výsledným textům předaných k dalšímu zpracování.

Příkladem může být analýza sentimentu, kdy je cílem odhadnout názor autora na pojednávanou látku. Místní názvy, vlastní jména obecně a číselné údaje nejsou z tohoto hlediska podstatné, naopak přídavná jména nabývají vrcholného významu.
Analýza zdrojového kódu programu (např. k odhalení původce malware), oproti tomu vyžaduje co nejlepší zachování struktury kódu a nahrazení specifických struktur vhodnými tokeny.

Vhodným regulárním výrazem je možné v textu vyhledat celou řadu řetězců, od specifických slov, slovních druhů, až k určeným tvarům vět.
Výhodou regulárních výrazů a nástrojů, které je využívají, je především jejich flexibilita, relativní jednoduchost a těsný vztah k teoretické informatice a gramatice.

Použití regulárních výrazů k analýze textu je běžnou praxí již mnoho desetiletí a bylo úspěšně použito k řešení řady obtížných úloh. (Zananir, 2008) (Kandalgaonkar, 2000)

Mezi nevýhody použití regulárních výrazů patří nutnost naprosto přesné definice hledaného řetězce, s ohledem na specifika implementace. Nevhodně napsaný výraz může, zdánlivě bezchybně, vyhledávat požadované řetězce a selhávat v řadě mezních případů.

Další podstatnou nevýhodou jsou značné rozdíly v syntaxi a vnitřním fungování jednotlivých nástrojů.

Stejný regulární výraz může být interpretován naprosto odlišně
nástroji POSIX a knihovnou PCRE[14] nebo dokonce stejným nástrojem s různým nastavením.

$$[A-Z][a-z][a-z]*$$
$$[A-Z][a-z]+$$

Fig. 1 Oba výrazy odpovídají stejným řetězcům, druhý ale vyžaduje spuštění nástroje grep v rozšířeném modu

V neposlední řadě je třeba upozornit, že složitější regulární výrazy se rychle stávají člověku nesrozumitelnými, zvláště pokud jsou generovány specializovaným softwarem. Následná údržba a změny výrazu mohou být velmi obtížné. (Atwood, Regex use vs. Regex abuse, 2005)

4.2 Padding

Celá řada modelů vyžaduje konzistentní rozsah vstupu. Pro zajištění shodné délky všech textů předkládaných modelu je běžně používána technika zvaná padding[15].
Padding využívá vhodně vybraný token, jehož přidáním je délka textu dorovnána. Stejně tak je možné délku textu vhodně omezit, ovšem oproti paddingu tak není možné činit bez pečlivého zvážení charakteristik zpracovaného textu.

Aplikace paddingu nevyhnutelně zvyšuje nároky modelu na paměť. Pro prostředky moderních počítačů, se ale vyjma extrémních případů, jedná o zanedbatelný efekt[16].

[14] Perl Compatible Regular Expressions
[15] V překladu vycpání nebo vycpávka
[16] Kompletní text všech článků Wikipedie v anglickém jazyce je možné komprimovat do rozsahu zhruba 16 GB. (Wikipedia org, 2019)

4.3 Optimalizace množiny dat

Optimalizace množiny dat obecně vyžaduje vyrovnání dvou vzájemně protichůdných cílů. Snahy redukovat dimenzionalitu dat a snahy zachovat, pokud možno celou informaci, obsaženou ve zpracovaném textu. Mezi nejčastěji používané nástroje patří stemming[17], lemmatizace, tokenizace a odstraňování stop words.

Velký rozvoj v oblasti zpracování textu si vynutily internetové vyhledávače, které vyžadují co nejpřesnější zjištění významu textu a jeho srovnání s dotazem uživatele, za použití minimálních zdrojů, a v co nejkratším možném čase.

4.3.1 Tokenizace

Nejjednodušší ze zmíněných technik, ačkoliv většinou aplikovanou až posledních fázích předzpracování dat, je tokenizace. Tokenizací rozumíme přiřazení vhodně zvolených jednoznačných symbolů, tokenů, řetězcům znaků původního textu. Tokeny mohou být použity v dalších krocích předzpracování, například k řízení chodu nástrojů jako je stemmer nebo lemmatizér. V některých případech může být původní řetězec tokenem zcela nahrazen, a výsledný text využit k další analýze.

Vhodným nástrojem k provedení tokenizace jsou bezesporu regulární řetězce, které ve svém tokenizačním modulu využívá široce rozšířená sada nástrojů NLTK. (NLTK Project, 2019) Modely se vstupem ve formě tenzorů vyžadují tokenizaci celého textu.

Tokenizovaný text je dále možné upravit vhodnou projekcí, embeddingem, a přivést na vstup. Tvorba a využití embeddingu je v současnosti aktivní oblastí výzkumu na poli umělé inteligence. (Mikolov, Chen, Corrado, & Dean, 2013)

[17] Stem – doslova kmen nebo stonek

4.3.2 **Stop words a jejich odstraňování**

Stop word, nebo také stop slova, jsou řetězce znaků s minimem poskytované informace, jejichž odstranění z textu nemůže negativně ovlivnit řešení úlohy. Většinou se jedná o některá z nejobvyklejších slov daného jazyka, například spojky a zájmena.
Nutně tedy neexistuje žádný univerzální seznam nebo slovník stop words, a jejich odstranění vyžaduje znalost frekvence výskytu slov v jazyku textu.

V případě korpusů, vytvořených z českých textů, tvoří stopslova nezanedbatelné procento všech obsažených slov. V případě Korpusů Voice of America, Czech Broadcast News, UWB_S01 a korpusu LN, tvoří předložky a spojky většinu deseti nejčastěji se objevujících slov. (Psutka, Müller, Matoušek, & Radová, 2006)

Odstraňování stop words je jednou z nejznámějších technik pro optimalizaci vyhledávání ve velkých databázích obsahujících text. Zkrácení délky dotazu snižuje nároky na výpočetní prostředky a umožňuje rychlejší zobrazení výsledku. (Atwood, Podcast #32, 2008)

Minimálním požadavkem na odstraňování stop words je jejich slovník vytvořený s ohledem na specifika daného jazyka a textu. Jeden ze slovníků vytvořených pro češtinu je přílohou práce. (ranks.nl, 2019)
Nicméně odstraňování stop words je třeba důkladně zvážit. Zvláště při analýze textu omezeného rozsahu se může jednat o nadbytečný úkon s minimálním přínosem ke konečnému výsledku, který může nepřípustně zredukovat původní text.

Přínos odstranění stop words závisí i na druhu úlohy. Mezi stop words je řada nejobvyklejších sloves, jejichž odstranění z textu může zcela změnit jeho význam nebo ho naprosto znehodnotit.

kde domov můj, kde domov můj, voda hučí po lučinách, bory šumí po skalinách, v sadě skví se jara květ, zemský ráj to na pohled! a to je ta krásná země, země česká domov můj, země česká domov můj!	kde domov kde domov voda hučí lučinách, bory šumí skalinách, v sadě skví jara květ, zemský ráj pohled! a krásná země, země česká domov, země česká domov!

Fig. 2 První sloka české hymny v minuskulích, před a po odstranění stop words[18]

4.3.3 **Stemming**

Stemming spočívá v nalezení kmene, popřípadě kořenu daného slova, a jeho následné nahrazení v textu.

Takzvaný stemmer může využívat prostou vyhledávací tabulku, obsahující všechny možné formy slov, seznam substitučních pravidel, aplikovaných bez dalších úprav nebo s ohledem na vybraná kritéria. Některé stemmery používají i techniky strojového učení, jako neurální sítě. (Mezher & Omar, 2015)

Každý stemming algoritmus je ovšem vyvinut specificky pro daný jazyk, a není možné jej bez úprav použít pro jazyk jiný. Mezi nejznámější stemming algoritmy patří porterův algoritmus (Porter M. , 2006) vytvořený pro anglický jazyk.

Nevýhodou stemmerů jsou značné rozdíly mezi jednotlivými implementacemi algoritmů, často plynoucích z nepřesných definic. (Porter M. , 2001)

Odstranění předpon a přípon může podstatně zredukovat počet unikátních slov v textu.

Takto upravený text má nejenom menší nároky na paměťová média, ale i vlastnosti prospěšné v dalších krocích předzpracování a následného řešení úlohy.

[18] K odstranění slov byl použit krátký script v jazyce python, využívající regulární výrazy. Cílem nebylo odstranit všechna vyskytující se stop words, ale demonstrovat, jak jejich nevhodné odstranění může znehodnotit text.

Čeština ovšem na zmíněné prvky spoléhá pro předání významu více než angličtina. Proto se obávám, že stemming se pro předzpracování českého textu příliš nehodí. (Heidenreich, 2018)

4.3.4 **Lemmatizace**

Ze zmíněných postupů vyžaduje lemmatizace nejrozsáhlejší přípravu a potenciálně nabízí největší přínos. Lemmatizátor nahrazuje slova jejich základním tvarem, lemmou. Na rozdíl od stemmingu, tokenizaci nebo odstraňování stop words, tedy lemmatizace pracuje nejenom se samotnými řetězci znaků, ale se slovy a jejich vzájemnými souvislostmi.
Nalezení lemmy není pouze otázkou substituce znaků, ale znalosti mluvnických kategorií, příslušných k dané formě slova. Je pochopitelné, že vývoj spolehlivého lemmatizátoru vyžaduje lingvistickou odbornost, nezanedbatelné množství dat a úsilí.
Lemmatizátory se mohou podstatně odlišovat co do pravidel a slovníku, a to i v rámci jednoho jazyka. Základní pravidla lemmatizace pro český jazyk jsou ale relativně prostá. (Český národní korpus, 2019)

- Podstatná jména do jednotného čísla prvního pádu
- Přídavná jména do jednotného čísla, prvního pádu, mužského rodu
- Slovesa do infinitivu

Původní slovo	**Lemma**
mostem	most
oranžovou	oranžový
slibuji	slibovat

Tabulka 4 Příklady slov a jim přiřazených lemm

Řada lemmatizátorů byla vyvinuta i pro Český jazyk. Existují však podstatné rozdíly v jejich možnostech, licenčních ujednáních a požadavcích na provoz. (Ústav formální a aplikované lingvistiky, 2019)

V současnosti je k dispozici řada sad nástrojů ke splnění takového zadání. Bohužel, většina není určena ke zpracování českého textu. (Ústav formální a aplikované lingvistiky, 2019)

To neznamená, že je nelze vůbec použít, naopak většina jejich funkcí je aplikovatelná napříč jazyky, včetně jazyka českého. Jmenovitě například tokenizace a odstraňování řetězců markup tagů.
Na druhé straně například stemming nebo lemmatizace se, přinejmenším bez úprav a lingvistické konzultace, přímo použít nedají. Jedná se totiž o techniky příliš závislé na specificích daného jazyka.
Přesto existuje poměrně široký výběr nástrojů určených právě k zpracování českého jazyka. Jejich kvalita ovšem může být pochybná a určení vhodného balíků nástrojů může být poměrně složité.
Navíc, v některých případech mohou analýzu textu zkomplikovat, protože vedou k redukci jeho rozsahu a potenciální ztrátě informace. Vzhledem k tomuto faktu, by jejich aplikace vyžadovala konzultaci s odborníky. (Vallantin, 2019)

V závislosti na dimenzionalitě vstupu a úloze ovšem nemusí být složitější předzpracování nutné, a je možné použít vstupní data bez větších změn. Krátké texty, sestávající z relativně malého množství unikátních slov, nemusí být ani žádoucí dále upravovat.
Jakékoliv úpravy totiž mohou připravit text o výraznou část původní poskytované informace. Znovu zde hraje roli potenciálně nepřípustná ztráta informace v textu.
Mezi další běžně používané techniky patří převedení textu do malých písmen, odstraňování diakritiky a interpunkce. V případě posledního jmenovaného často dochází k nahrazení znaménka tokenem.

4.4 Analýza textu

Metody analýzy textu mají často značně odlišné požadavky na formu vstupu, úroveň informací poskytnutých modelu z externích zdrojů a výpočetní výkon. Výběr vhodné metody závisí především na účelu analýzy a uspokojení jejích minimálních požadavků, neboť obtížnost ladění modelu se zvyšuje spolu s jeho složitostí.

Úlohy analýzy textu jsou ale v mnoha ohledech blízké ostatním úlohám oboru umělé inteligence a aplikuje se na ně řada stejných omezení a pravidel.

Neuronová síť, použitá ke kategorizaci textu, se řídí stejnými zákonitostmi jako neuronová síť klasifikující květy kosatců. (Fisher, 1936)
Původní data obou modelů se mohou zdát těžko srovnatelná, ale transformace, kterými před přivedením na vstup prochází, je převádí do shodné formy.

4.4.1 **Jazykový korpus**

Jedním z nejdůležitějších nástrojů pro analýzu textu je korpus daného jazyka. Jazykový korpus je souborem většího množství samostatných textů, ve kterém je možné vyhledávat specifické jazykové jevy, při zachování jejich kontextu. Korpus je třeba vytvářet s ohledem na zvolenou úlohu. (Křen M. , 2017)

Texty, ze kterých se korpus skládá, by měly vykazovat všechny jevy ve všech kontextech, které mohou hrát roli v řešení dané úlohy. Zároveň je třeba zajistit, že je frekvence jazykových jevů ve vzniklém korpusu reprezentativní pro daný jazyk, období a typ textu.

V neposlední řadě musí korpus obsahovat informace o vlastnostech každého obsaženého textu, většinou ve formě strojem zpracovatelných metadat vhodného formátu. Právě metadata, jejich spolehlivost, podrobnost a struktura, jsou klíčovým faktorem pro ohodnocení kvality korpusu.
Kvalitní metadata umožňují snadné ověření původu a základních charakteristik obsažených textů. Současně pomáhají zajistit, že se texty v korpusu neopakují a nezpůsobují předpojatost korpusu vůči určitým jazykovým jevům nebo tématům. Stejně tak může korpus obsahovat informace o obsažených jazykových jevech.

Pro český jazyk byla vyvinuta celá řada jazykových korpusů, různého zaměření a rozsahu. (Škrabal, 2018)

Projekt Českého národního korpusu, spadající pod Filozofickou fakultu UK, spravuje celou řadu korpusů českého jazyka. Jedním z nejznámějších je korpus SYN.

SYN je synchronním korpusem psaného jazyka, vytvořeným především z publicistických textů, každoročně zveřejňovaný v nové rozšířené podobě. SYN 2018 je zatím poslední verzí korpusu, obsahuje 4255 milionů slov ve více než šestnácti milionech textů[19].
SYN 2018 je již sedmou iterací korpusu a skládá se ze všech verzí předchozích, opatřených referencemi. (Křen, 2018)
Vzhledem k původu velké části obsažených textů, je právě SYN příkladem nereprezentativního korpusu.

Korpus tvořený vhodně vybranými texty, přizpůsobený vyhledávání jazykových jevů a opatřen nutnými referencemi, je nutným základem pro tvorbu stemmerů a lemmatizátorů. Kvalita nástrojů zpracování přirozeného jazyka je nevyhnutelně závislá na kvalitě korpusu, použitého k jejich tvorbě.

Znalost korpusu, použitého během vývoje nástrojů zvolených k další analýze textu, je nutnou podmínkou ověření jejich kvality, stejně jako kvality samotné analýzy.

4.4.2 **Závislostní korpus**

Závislostní korpusy poskytují, vyjma informací obsažených ve všech korpusech, syntaktickou, morfologickou a sémantickou anotaci obsažených textů. Korektní anotace, automatické nebo manuální, umožňují podrobnější analýzu jednotlivých textů a jazykových prostředků v nich použitých.
Mezi závislostní korpusy českého jazyka patří Pražský závislostní korpus[20], dále PDT, nyní ve své třetí verzi. PDT sestává z textů vybraných z Českého národního korpusu, anotovaných ve třech úrovních: morfologické, analytické a tektogramatické, přičemž vyšší úrovně pokrývají pouze část slov anotovaných úrovněmi předchozími.
Morfologická úroveň pokrývá anotacemi dva miliony slovních symbolů, z nich jeden a půl milionu slov, odpovídající 88 tisícům vět, je anotováno na analytické úrovni a osm set tisíc z nich, tvořících čtyřiceti devíti vět, i na úrovni tektogramatické. (Ústav formální a aplikované lingvistiky, 2019)

[19] Přesně 16 377 839
[20] Prague Dependency Treebank

4.4.3 **Kategorizace textu**

Kategorizace textu patří mezi klasifikační úlohy a obecně se řídí stejnými zákonitostmi. Stejně jako u ostatních klasifikačních úloh, je cílem přiřadit každému objektu n hodnot, reprezentujících pravděpodobnost jeho příslušnosti k jedné nebo více z n kategorií.
Kategorie mohou být určeny zdrojem původních dat, nebo nalezeny jinou metodou strojového učení, například formou shlukové analýzy.

4.4.4 **Shrnutí textu**

Automatické shrnutí neboli sumarizace textu, spočívá ve vytvoření textu nového, zahrnujícího co největší podíl informace textu původního.
Shrnutí může probíhat buď extrakcí nebo abstrakcí. Extrakce využívá významné části původního textu, a následně jejich kombinací vytváří text nový. Abstrakce vytváří originální text s využitím informací z textu původního, umožnuje tak vytvořit kratší a efektivnější shrnutí.

Jedním z nejznámějších aplikací automatické sumarizace, vycházející z extrakce, je softwarový agent AutoTLDR Bot, aktivní a populární ve vybraných okruzích internetové sociální platformy Reddit.

AutoTLDR Bot vytváří shrnutí mediálních článků, odkazovaných ve vybraných komunitách Reddit.com s pomocí algoritmu SMMRY, a následně je vkládá jako příspěvek do diskuze. (Bot, 2015)

4.4.5 **Extrakce entit**

Úlohy extrakce entit spočívají v nalezení vybraných subjektů, například osob, podniků a míst v textu, a jejich následném vyznačení pro další zpracování. Může tak tvořit jeden z kroků procesu anonymizace textu, popřípadě sloužit jako základ pro hledání vztahů mezi nalezenými entitami.

Jedná se o poměrně rozvinutou techniku. Podle studie vedené Retrieval Group při NIST[21] již v roce 1998 dosahovaly nejlepší řešení přesnosti srovnatelné s člověkem. (Voorhees & NIST, 2001)

4.4.6 **Analýza sentimentu**

Cílem analýzy sentimentu je odhadnout subjektivní informace z poskytnutého textu. V závislosti na úloze se může jednat o citové rozpoložení autora, názor autora na předmět popisovaný textem nebo jinou informaci, která není v textu explicitně vyjádřena.
Techniky, používané analýzou sentimentu, jsou hojně používány k analýze ohlasů a názorů na poskytovaný výrobek nebo službu. S výsledky analýzy sentimentu je možné následně vytvořit individuální nabídku produktů pro individuální zákazníky nebo určit úroveň spokojenosti s jednotlivými aspekty předmětu textu.

[21] *National Institute of Standards and Technology* – spadá pod ministerstvo obchodu USA

5 Umělé neuronové sítě

Technologie umělých neuronových sítí, rozvíjená již od čtyřicátých let dvacátého století, dostála během posledních deseti let podstatných změn, ačkoliv její základní charakteristiky zůstaly nezměněny.

Umělé neuronové sítě vycházející z idealizovaného modelu biologických neuronových sítí a na teoretické úrovni z konekcionistického paradigmatu, sestávají z množiny výpočetních jednotek, nazývaných umělé neurony, a jejich propojení.

Neformálně je možné umělou neuronovou síť pojmout jako soustavu zesilovačů s jistým počtem vstupů. Přičemž výstup celé sítě je možné upravit pomocí soustavy potenciometrů, měnících napětí na vstupu zesilovačů. První příklady umělých neuronů, takzvaný perceptronový model, dokonce byly fyzickými stroji. (Rosenblatt, 1957)

Výpočetní jednotky, dále umělé neurony, jsou organizovány do grafu, jehož topologie závisí na zvolené aplikaci modelu. Hrany grafu, propojení umělých neuronů, obvykle nazývané váhy, vytvářejí z jednotek komplexní systém.

Za vstupní označujeme neurony přijímající signál z vnějšího prostředí. Vstupní neurony přijímaný signál neupravují, pouze ho předávají dál. Umělé neurony, jejichž výstupem je odezva celé neuronové sítě, označujeme jako výstupní. Ostatní neurony sítě, jež jsou propojené pouze s dalšími neurony sítě, jsou označovány jako skryté.

Umělé neuronové sítě je možné klasifikovat vzhledem k topologii jejich grafu.
V závislosti na přítomnosti cyklů v grafu sítě, rozlišujeme sítě dopředné a rekurentní.
Umělé neurony dopředné neuronové sítě, spolu s jejich propojeními, tvoří acyklický graf.
Graf rekurentních neurnových sítí je cyklický.

Díky přítomnosti cyklů v grafu mohou rekurentní neuronové sítě, dále RNN, postupně přijímat hodnoty sekvenčních dat, například časové řady a zachovávat informace o již přijatých hodnotách náležících dané sekvenci.

RNN tak nacházejí využití především v úlohách, jejichž řešení zahrnuje analýzu časových řad, textu a zvukových záznamů.

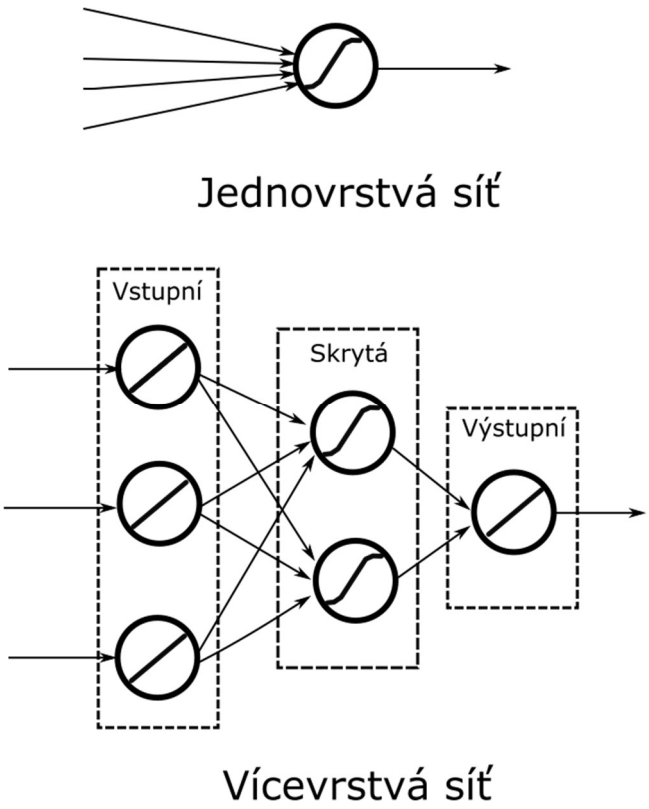


Fig. 3 Zjednodušený diagram jednovrstvé a vícevrstvé dopředné sítě, vrstvy vícevrstvé sítě jsou hustě propojené

Podle organizace neuronů v grafu rozlišujeme dopředné sítě jednovrstvé a vícevrstvé. Jednovrstvé sítě sestávají pouze ze vstupních a výstupních neuronů, příkladem může být síť tvořená jedním perceptronem. (Rosenblatt, 1957) Umělé neuronové sítě sestávající z jednoho perceptronu ovšem nejsou schopné řešit úlohy, vyžadující klasifikaci lineárně neseparabilních množin bodů.

Vícevrstvé sítě zahrnují takzvané skryté neurony, organizované do jedné nebo více vrstev, analogicky nazývaných skryté vrstvy. Jednotlivé skryté vrstvy sítě jsou indexovány podle vzdálenosti k vrstvě vstupní.

Vrstvy tvořené neurony, pro které platí, že výstup každého neuronu je propojen se vstupem každého neuronu vrstvy následující, se nazývají hustě nebo plně propojené.

Výstupem umělého neuronu je hodnota aktivační funkce, jejíž argument, takzvaný postsynaptický potenciál, je váženým součtem hodnot vstupů spolu s hodnotou prahu, podle vztahu:

$$\varphi(\sum_{i=0}^{n} w_i * x_i + b)$$

Kde φ značí aktivační funkci, x_i hodnotu i-tého vstupu neuronu, w_i hodnotu příslušné váhy a b hodnotu prahu. Alternativní, mírně jednodušší, zápis aktivační funkce, přesouvá hodnotu prahu mezi ostatní vstupy neuronu a přidává omezení hodnoty x_0.

$$\varphi(\sum_{i=0}^{n} w_i * x_i)$$

$$x_0 = 1$$

Úpravou hodnot vah, včetně prahu, je možné měnit hodnotu výstupu neuronu. Umělou neurnovou síť o m neuronech je možné chápat jako funkci $G(X)$ s počtem parametrů rovných:

$$N = \sum_{j=0}^{m} n_j$$

Kde n_j značí počet vah j-tého neuronu sítě. Proces učení je možné pojmout jako proces s cílem aproximovat neznámou funkci $F(X)$ již zmíněnou funkcí $G(X)$, vhodnou úpravou hodnot N parametrů funkce $G(X)$.

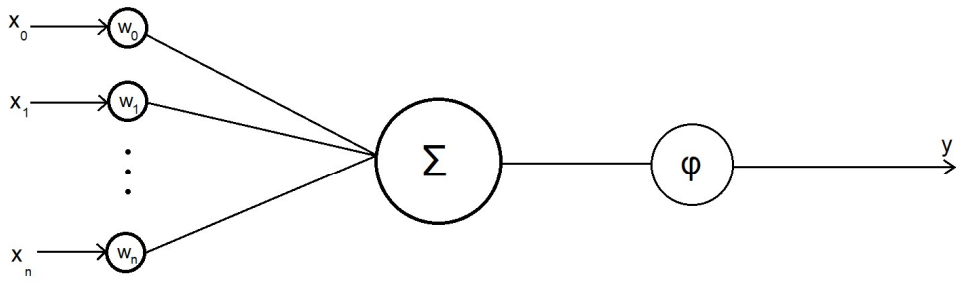


Fig. 4 Diagram umělého neuronu

Během procesu učení jsou na vstup sítě přiváděny vektory z trénovací množiny, a výstup sítě je srovnáván s výstupem očekávaným.

V závislosti na zvolené chybové funkci, optimalizačním algoritmu, aktivačních funkcích neuronů a topologii sítě, jsou hodnoty vah upraveny. Proces učení pokračuje, dokud vlastnosti modelu, měřené vhodnými metrikami, nedosáhnou určených hodnot nebo není překročen přidělený čas.

Již velmi rané neuronové sítě umožňovaly řešit řadu úloh. Nicméně vykazovaly nežádoucí chování, které omezovalo jejich praktické aplikace. Jednalo se nejenom o teoretické překážky, ale především o vysoké požadavky na výpočetní výkon, které činily neuronové sítě méně dostupnými oproti řadě alternativ. (Minky & Seymour, 1969)

Mezi teoretické problémy, bránící širšímu přijetí neuronových sítí, patřily především otázky zpětnovazebního signálu sítě, gradientu, jeho šíření v síti a tendencí během procesu učení, známých jako problém mizejícího a explodujícího gradientu. (Hochreiter, 1991) Jejich řešení vyžadovalo nových optimalizačních algoritmů, inicializačních schémat vah a aktivačních funkcí.

5.1 Aktivační funkce

Charakteristiky aktivační funkce závisejí na algoritmu učení a úloze řešené modelem. Mezi nejdůležitější patří požadavek na nelinearitu funkce, bez níž není možné řešit úlohy vyžadující aproximaci nelineárních funkcí, nehledě na topologii a další vlastnosti sítě.

Z již uvedené definice umělé neuronové sítě jako funkce $G(X)$ vyplývá, že pokud je aktivační funkce φ všech výpočetních jednotek lineární, je výstup neuronové sítě nutně lineární kombinací hodnot vstupního vektoru.

Druhou důležitou vlastností aktivační funkce jsou charakteristiky její derivace, především, zda je funkce spojitě diferencovatelná.

V současnosti patří mezi nejpoužívanější aktivační funkce ReLU, aproximující funkci $f(x) = \ln(1 + e^x)$, známou jako softplus. (Glorot, Bordes, & Bengio, 2011)

	Vzorec
Lineární	$f(x) = cx$
Sigmoid / Logistická	$f(x) = \frac{1}{1 + e^{-x}}$
Tanh	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Binární krok / Heavisideova funkce	$f(x) = \begin{cases} 0, x < 0 \\ 1, x \geq 0 \end{cases}$
ReLU	$f(x) = \max(0, x)$

Tabulka 5 Vzorce vybraných aktivačních funkcí

ReLU sice není diferencovatelná pro $x = 0$, ale přináší výhody z hlediska výpočetní složitosti, a především pomáhá řešit problém mizejícího gradientu. V případě aktivačních funkcí, které nejsou diferencovatelné pro každé $x \in \mathbb{R}$, závisí hodnota derivace v daných bodech na volbě vývojáře dané implementace.

Hodnota derivace aktivační funkce ReLU pro $x = 0$ závisí, v implementaci frameworku Keras, na argumentech zadaných při vytvoření instance příslušné třídy. Nicméně základní nastavení argumentů staví derivaci pro $x = 0$ rovnou 0. (Keras Team, 2020)

5.2 Algoritmus učení umělé neuronové sítě

Formálně je proces učení s n vektory trénovací množiny shodný s hledáním minima chybové funkce:

$$C = \frac{\sum_{i=0}^{n} C_i(G(X_i), Y_i)}{n}$$

Kde X_i značí i-tý vektor trénovací množiny a Y_i příslušný, očekávaný výstup.

Určení optimálního nastavení všech N parametrů neuronové sítě, reprezentované funkcí $G(X)$, je potom shodné s vyřešením polynomu o N proměnných.
Nalezení globálního minima, a tedy i optimálního nastavení všech parametrů sítě, se tak stává obtížnou úlohou i pro relativně jednoduchou neuronovou síť.

Alternativní geometrická interpretace ztotožňuje učení s hledáním bodu minima, na N rozměrné ploše, definované chybovou funkcí.

Přičemž každý bod plochy odpovídá jednomu možnému nastavení parametrů modelu a jedné hypotéze z prostoru hypotéz. Některé vlastnosti plochy, například počet lokálních minim, jejich spojitost nebo ohraničení spojitými oblastmi extrémů, mají na složitost úlohy negativní dopad.

Řada funkcí, definující plochy vykazující nežádoucí charakteristiky, je v praxi používána pro ohodnocení kvality optimalizačních algoritmů.(Pohl, Jirsík, & Honzík, 2014)

Příkladem funkce definující povrch, vykazujícího vlastnosti komplikující nalezení globálního minima, je Eggholder funkce, z důvodů velkého počtu lokálních minim a nepravidelnosti povrchu. (Surjanovic & Bingham, 2020)

$$f(x_1, x_2) = -(x_2 - 47)\sin\sqrt{\left|x_2 + \frac{x_1}{2} + 47\right|} - x_1 \sin\sqrt{|x_1 - (x_2 + 47)|}$$

S hodnotami x_i, podle konvence, omezenými na $x_i \in \langle -512{,}512 \rangle$ pro $i = 1{,}2$.

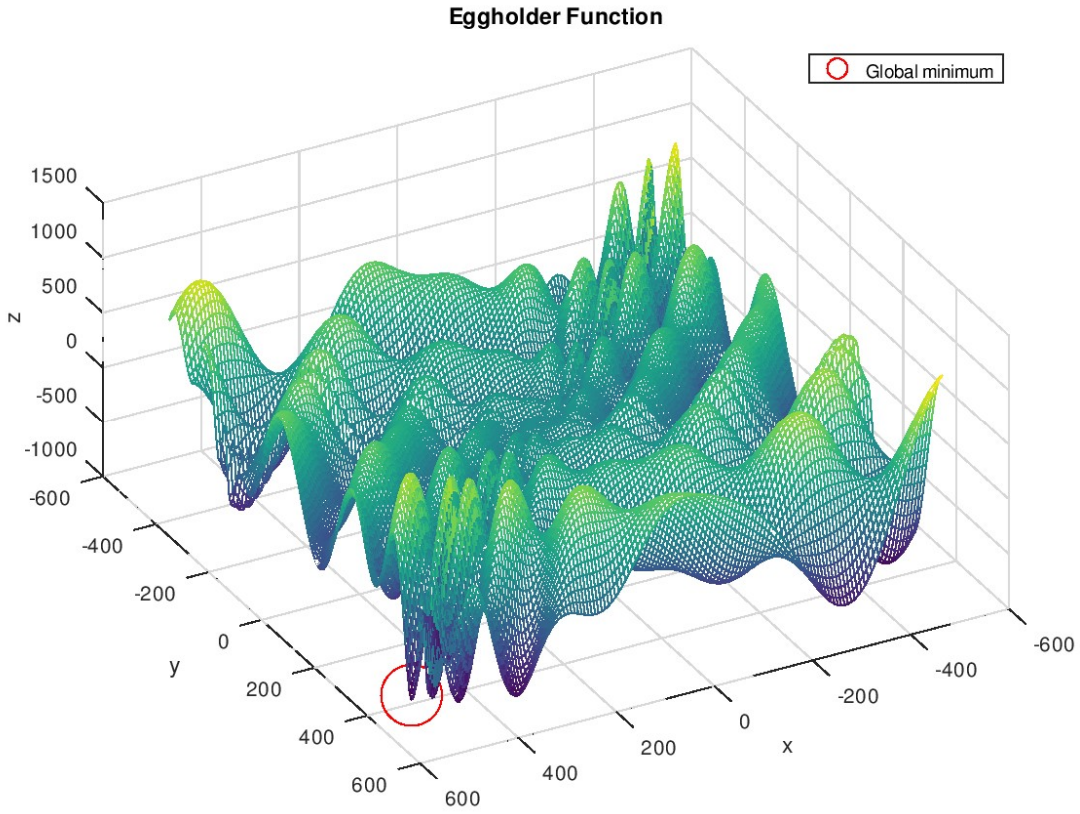


Fig. 5 Graf Eggholder function (Karton na vejce), globální minimum označené rudým kruhem

Jak je patrné, požadavky kladené na charakteristiky topologií umělých neurálních sítí a použitých aktivačních funkcí, především jejich derivací, nutně vycházejí z pojetí procesu učení, jako hledání minima chybové funkce.

V praxi je k nalezení minima funkce používána řada iterativních a heuristických metod, které sice negarantují nalezení globálního minima chybové funkce, ale jsou schopné nalézt přibližné řešení zadané úlohy, nebo k řešení konvergovat v rozumném čase.

5.2.1 **Gradientní sestup a varianty**

Mezi iterační algoritmy hledání lokálního minima funkce patří gradientní sestup a algoritmy z něj odvozené, využívající k určení nové hodnoty parametrů gradientu funkce.

Algoritmy, vycházející z gradientního sestupu, je možné klasifikovat podle počtu prvků, trénovací množiny použitých k výpočtu gradientu. Algoritmus gradientního sestup využívá k výpočtu gradientů všechny prvky množiny trénovací množiny:

$$T = T_s \subseteq T$$

Kde T značí trénovací množinu a množinu T_s prvků zvolených k výpočtu gradientu. Algoritmy stochastického gradientního sestupu, dále SGD, a minidávkového SGD využívají pouze vzorek trénovací množiny různého rozsahu.

Rozsah vzorku trénovací množiny T_s	**Algoritmus**
$\|\boldsymbol{T_s}\| = \|\boldsymbol{T}\|$	Gradientní sestup / Dávkový SGD
$\mathbf{1} < \|\boldsymbol{T_s}\| < \|\boldsymbol{T}\|$	Minidávkový SGD
$\|\boldsymbol{T_s}\| = \mathbf{1}$	SGD

Tabulka 6 Klasifikace algoritmů gradientního sestupu podle rozsahu vzorku trénovací množiny

Stejně jako klasický gradientní sestup, využívá i SGD gradientu funkce k určení nové hodnoty parametru. Oproti algoritmu gradientního sestupu, nevyžaduje SGD znalost skutečného gradientu funkce, vypočítaného na celé množině trénovacích dat T, ale pouze jeho odhad, vypočítaný na podmnožině T_s. (Song, Montanari, & Nguyen, 2018)

Podle $|T_s|$ rozlišujeme SGD, a minidávkový SGD. Je zřejmé, že $|T_s|$ ovlivňuje počet operací nutných k provedení jedné iterace algoritmu a současně přesnost odhadu gradientu. Volba $|T_s|$ tak závisí na $|T|$, dostupných výpočetních prostředcích a přesnosti odhadu gradientu vyžadované k nalezení minima funkce.

Míra změny parametru je přitom závislá na hodnotě parametru $krok$, někdy nazývaného rychlost učení. Součin rychlosti učení a gradientu nazýváme aktualizací váhy.
Nová hodnota parametru k je určena vztahem:

$$W_t = W_{t-1} - \eta \nabla \mathrm{C}$$

$$\Delta W_{t-1} = -\eta \nabla \mathrm{C}$$

$$W_t = W_{t-1} + \Delta W_{t-1}$$

Kde W_t značí matici parametrů modelu v čase t, η hodnotu kroku, ∇C gradient chybové funkce C a ΔW_{t-1} aktualizaci hodnot parametrů modelu, určenou příslušným gradientem v čase $t-1$. (Boyd & Vandenberghe, 2004)

Volba hodnoty parametru $krok$ má zásadní vliv na průběh učení modelu a jeho kvalitu. Nevhodně zvolená hodnota může podstatně komplikovat nebo dokonce znemožnit nalezení vhodného minima chybové funkce.

Oba extrémy, příliš velká a příliš malá hodnota, vedou jinému druhu nežádoucího chování. Příliš velká hodnota může zabránit konvergenci funkce k minimu, způsobit, že váhy modelu budou vhodný bod stále „míjet“.

Nevhodně malá hodnota konvergenci umožní, ale algoritmus nebude schopen opustit nalezené lokální minimum. Nedostatečná úprava parametrů ponechá model ve velmi blízkém okolí předchozího bodu plochy chybové funkce, a v následné iteraci algoritmu bude gradient funkce znovu směřovat k předchozí poloze.

Je nutné si uvědomit, že povrch chybové funkce a její gradient, je unikátní nejenom pro každou úlohu, neboť funkce vychází z topologie neuronové sítě a použitých aktivačních funkcí, ale i pro každou množinu trénovacích příkladů, použitých k výpočtu gradientu. Algoritmy SGD, pracující s odhadem gradientu závislém na vzorku příkladů trénovací množiny, tedy v každé iteraci pracují s jiným tvarem chybové funkce.

K potlačení následků volby nevhodné hodnoty parametru $krok$, je využívána řada metod, pracujících s hodnotou parametru, měnící se během procesu učení, nebo přizpůsobenou jednotlivým parametrům.

5.2.2 **Algoritmus zpětného šíření chyby**

K výpočtu gradientu chybové funkce je využíván algoritmus zpětného šíření chyby, jehož základní podoba vychází z principů metody nejmenších čtverců. Jak napovídá název,

algoritmus zavádí šíření chyby, začínající na výstupu sítě, konečnou hodnotou chybové funkce do dalších vrstev.

Příspěvek jednotlivých vah k postsynaptickému potenciálu neuronu, a tedy i odezvě celé neuronové sítě a hodnotě chybové funkce, není stejný. Hodnota aktualizace musí být proporční, vzhledem k vlivu váhy, pokud má hodnota chybové funkce klesat.

Algoritmus určuje hodnotu aktualizace váhy $w_i^{(k-1)}$ jako derivaci chybové funkce vzhledem k $w_i^{(k-1)}$ podle vztahu:

$$\Delta w_i^{(k-1)} = \frac{\partial C}{\partial w_i^{(k-1)}} = \sum_{i=0}^{n} \frac{\partial C}{\partial \varphi_i^{(k)}} \frac{\partial h_i^{(k)}}{\partial w_i^{(k-1)}} \frac{\partial \varphi_i^{(k)}}{\partial h_i^{(k)}}$$

Vycházejícího z pravidla pro derivaci složených funkcí. Kde k je indexem příslušné vrstvy sítě, C danou chybovou funkcí, $h_i^{(k)}$ hodnotou postsynaptického potenciálu a $\varphi_i^{(k)}$ hodnotu aktivační funkce daného neuronu. S výsledným gradientem chybové funkce pro síť o m vrstvách a n neuronech ve výstupní vrstvě:

$$\nabla C \left(\frac{\partial C}{\partial w_0^{(0)}}, \dots, \frac{\partial C}{\partial w_i^{(k-1)}}, \dots, \frac{\partial C}{\partial w_i^{(n)}} \right)$$

Je evidentní, že jedna iterace procesu učení vyžaduje řady derivací, jejichž počet roste s rozsahem sítě, co do počtu vrstev i neuronů. Optimalizací výpočtu derivací je tedy možné dosáhnout podstatného snížení nároků na výpočetní prostředky.

5.2.3 **SGD s hybností**

Optimalizační algoritmy pracující s pojmem hybnosti, určují aktualizaci hodnoty parametru nejenom podle současné hodnoty gradientu, ale i podle hodnoty předcházející aktualizace. (Robbins & Monro, 1951)

$$\Delta W_{t-1} = \alpha \Delta W_{t-1} - \eta \nabla \mathrm{C}$$

Kde α značí váhu předchozí aktualizace. Zavedením hybnosti je možné omezit tendenci algoritmu, nacházet lokální minima a alespoň částečně zmírnit následky nevhodné volby parametru η. Algoritmy využívající hybnosti vyžadují znalost předchozí hodnoty aktualizace každé váhy, pro použití v následující iteraci, a dodatečné operace při samotném výpočtu její nové hodnoty, což zvyšuje jejich nároky na výpočetní prostředky.

5.2.4 **Adaptivní algoritmy**

Ačkoliv hybnost dokáže podstatně omezit důsledky nevhodné volby η, není je schopna zcela potlačit. Další nevýhodou algoritmů využívajících hybnost je, že stejně jako ostatní popsané metody, používají stejnou hodnotu parametru η pro všechny parametry modelu.

Adaptivní algoritmy, vycházejících z optimalizační algoritmů s hybností, upravují hodnotu η během celého procesu učení, a to i pro jednotlivé váhy modelu. (Ruder, 2016)

Zcela tak eliminují možnost a důsledky volby nevhodné hodnoty η.
Nevýhodou adaptivních algoritmů je nutnost provedení dodatečných výpočtů k určení hodnoty η pro každý parametr modelu. Jejich vhodnost tak závisí na vlastnostech zadané úlohy a dostupných výpočetních prostředcích.

Algoritmus Adagrad vychází z předpokladu, že méně obvyklé příznaky mohou být stejně, nebo podstatně více informativní než příznaky obvyklé, a přizpůsobuje hodnotu η vzhledem k vlastnostem dat.

Algoritmus přiřazuje vyšší hodnoty η parametrům souvisejícím s méně obvyklými příznaky, a umožňuje tak modelu zohlednit i relativně nepravděpodobné, ale významné charakteristiky příkladů trénovací množiny. (Duchi, Hazan, & Singe, 2011)

Adagrad je zvláště vhodný pro optimalizaci modelů, zahrnujících příznaky, jejichž hodnota je pro většinu pozorování trénovací množiny nulová, a byl úspěšně použit k optimalizaci vektorů embeddingu GloVe. (Pennington, Socher, & Manning, 2014)

Algoritmus Adagrad určuje hodnotu η pro jednotlivé parametry modelu podle hodnot dříve zaznamenaných gradientů podle vztahu:

$$W_t = W_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla \mathrm{C}$$

$$\epsilon \cong 10^{-8}$$

Kde G_t značí hodnotu akumulátoru gradientů příslušného parametru modelu v čase t a ϵ relativně malou hodnotu, bránící přítomnosti jmenovatele rovného nule. Hodnoty G_t příslušné každému parametru, jsou v paměti uloženy jako prvky diagonální matice.

Nevýhodou Adagrad je růst hodnoty akumulátoru, a z něho plynoucí klesající hodnota η během učení modelu. V určitém čase t se tak hodnota η bude nevyhnutelně blížit nule, a proces učení se zastaví.

Jedním z algoritmů, řešících postupný pokles hodnoty η, je RMSprop. (Hinton, 2014) (TensorFlow community, 2020)
Namísto součtu hodnot všech předchozích gradientů, RMSprop dělí η odmocninou klouzavého průměru jejich druhých mocnin zvýšenou o ϵ.

$$W_t = W_{t-1} - \frac{\eta}{\sqrt{E[\nabla \mathrm{C}^2]_t + \epsilon}} \nabla \mathrm{C}$$

$$E[\nabla \mathrm{C}^2]_t = \rho E[\nabla \mathrm{C}^2]_{t-1} + (1 - \rho) \nabla \mathrm{C}^2$$

Kde $E[\nabla \mathrm{C}^2]_t$ značí průměr druhých mocnin gradientu příslušného parametru v čase t a ρ je parametrem míry útlumu, snižující vliv $E[\nabla \mathrm{C}^2]_{t-1}$ na nově vypočtený průměr. Většina implementací nastavuje $\rho = 0.9$, podle příkladu v původním návrhu algoritmu. Nicméně je možné zvolit i hodnotu jinou.

Algoritmem do značné míry podobným RMSProp je, nezávisle vyvinutý, Adadelta. Oproti RMSProp, Adadelta zcela odstraňuje parametr η, a nahrazuje ho odmocninou průměru druhých mocnin aktualizací parametru. (Zeiler, 2012)

Mezi nejefektivnější adaptivní algoritmy patří Adam, vycházející z principů algoritmů RMSProp a Adagrad. (Kingma & Ba, 2015)

Algoritmus Adam interpretuje průměr druhých mocnin gradientu jako odhad rozptylu předchozích hodnot gradientu a současně zavádí odhad průměru předchozích hodnot gradientu, podle vztahu:

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla C$$
$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * \nabla C$$
$$v_0 = m_0 = 0$$

Kde m_t značí odhad průměru a v_t odhad rozptylu v čase t, β_1 míru útlumu předchozího průměru a β_2 míru útlumu předchozího rozptylu, s doporučenými hodnotami:

$$\beta_1 = 0.9$$
$$\beta_2 = 0.999$$

Inicializace hodnot odhadu průměru a rozptylu hodnotou 0 zkresluje hodnoty obou odhadů směrem k 0, zvláště během prvních iterací algoritmu. K potlačení zkreslení zavádí autoři algoritmu korigované hodnoty odhadů. Závislé na příslušných mírách útlumu a počtu proběhlých iterací algoritmu:

$$\widehat{m_t} = \frac{m_t}{(1 - \beta_1^t)}$$
$$\widehat{v_t} = \frac{v_t}{(1 - \beta_2^t)}$$

Korigované odhady průměru a rozptylu gradientů jsou použity k výpočtu aktualizace vah modelu podle vztahu:

$$W_t = W_{t-1} - \alpha \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon}$$

Kde α odpovídá velikosti kroku η v jiných optimalizačních algoritmech a přibližně omezuje délku kroku $|\Delta W_t| < \alpha$. Hodnotu α je možné pojmout jako nastavení oblasti důvěry algoritmu.

Nutnost nastavit délku kroku tak zcela neodpadá. Nicméně doporučená hodnota $\alpha = 0.001$ je podle autorů algoritmu vhodná pro většinu úloh.
Vyjma základního verze algoritmu navrhli autoři ve stejné práci i rozšířenou variantu Adamax, využívající L^p normy předchozích a současných gradientů s $p \to \infty$.

5.3 Tensorové přístupy

Řada současných nástrojů a knihoven strojového učení, mimo jiné TensorFlow, PyTorch a Theano, využívá k reprezentaci modelů, a jejich struktury, konceptu tensorů.
Ačkoliv jsou tensory, v pojetí knihoven strojového učení, založeny na vícerozměrných maticích, nesoucími stejné jméno, ne zcela jim odpovídají, a jejich koncepce do značné míry vychází z principů objektového paradigmatu. Podstatnou výhodou pojetí tensorů jako objektů je snadná aplikace technik automatické diferenciace. (Neidinger, 2010)

Stejně jako matematický objekt tenzoru, i tenzory frameworků strojového učení jsou definované svým řádem, počtem prvků a tvarem. Tvarem tensoru rozumíme n-tici $(a_0, \dots, a_n)$, kde prvek a_i značí počet rozměrů tensoru podél dané osy.
Vlastnosti tensorů jsou ovšem určeny i datovým typem jejich prvků. V závislosti na implementaci, mohou být prvky tensoru i referencemi objektů jakéhokoliv datového typu. (The SciPy community, 2020)

Třídy tensorů podporují dědičnost a kompozici, stejně jako další vlastnosti tříd.
Instance tříd tensorů obsahují, stejně jako ostatní objekty, i tensory obsahují data a metody. V případě tensorů data sestávají také z hodnot parametrů modelu a rozměrů matice.
Metody tensoru umožňují manipulaci s jeho vlastnostmi a hodnotami prvků.

Z hlediska konstrukce modelů strojového učení jsou především důležité tensorové operace vysílání a změny tvaru, které umožňují práci s více tensory rozdílných rozměrů.

Využití objektového paradigmatu přináší podstatné výhody pro návrh a vývoj modelů. Tenzory a jejich operace je možné použít jako stavební bloky modelu, snadno zaměnitelné, znovupoužitelné, upravitelné a optimalizované pro efektivní zpracování.

Výpočet odezvy, gradientů i nových hodnot parametrů, je možné provést voláním metod příslušných objektů, aniž by bylo třeba psát další řádky kódu. Program nutný k implementaci jednoduchého modelu je tak podstatně čitelnější a kratší.

Zavedení automatické diferenciace umožňuje výpočet nových hodnot parametrů s libovolnou přesností a v čase srovnatelném s ručně programovanými derivacemi. (Margossian, 2019)

5.4 Konvoluční neuronové sítě

Z principů dopředných umělých neuronových sítí vycházejí modely, zahrnující jednu nebo více konvolučních vrstev, které nazýváme konvolučními neuronovými sítěmi.

Na rozdíl od neuronů hustě propojených vrstev, nepřijímají neurony vrstev konvolučních výstup všech neuronů předcházející vrstvy, ale pouze neuronů nacházejících se v menší oblasti vrstvy předcházející. Konvoluční vrstvy jsou tak schopné detekovat lokální vzory, oproti hustě propojeným vrstvám, které detekují vzory globální.

Neurony konvoluční vrstvy jsou přitom schopné nalezené vzory detekovat nehledě na polohu v původním vstupu.

Vhodným spojením více konvolučních vrstev je možné vytvořit model detekující nejenom lokální vzory původního vstupu, ale i vztahy mezi vzory nalezenými předcházejícími vrstvami. Konvoluční neuronové sítě tak mohou rozpoznávat hierarchie vzorů.

Struktura neuronu konvoluční vrstvy se v mnoha ohledech liší od struktury neuronu vrstvy hustě zapojené. Zatímco výstup hustě zapojeného neuronu je určen hodnotami vah, prahu a aktivační funkce, chování konvoluční neuronů je definováno hodnotami vah, prahu a tvarem oblasti, okna, určující neurony přecházející vrstvy, jejichž výstup neuron přijímá.

V některých ohledech je vhodnějším označením konvolučního neuronu filtr nebo konvoluční jádro. Konvoluční vrstvu potom můžeme považovat za tenzor, skládající se z jednoho nebo více konvolučních jader, popřípadě za soubor filtrů, jejichž charakteristiky podléhají během procesu učení změně.

Poslední uvedená definice je zvláště vhodná při zpracování vizuálních dat, neboť činí popis modelu a jeho vývoj podstatně bližší ostatním technikám zpracování obrazu.

Konvoluční vrstvy modelu jsou tak definovány svým počtem konvolučních jader, velikostí okna, kroku a použitou technikou paddingu. Rozsah konvoluční vrstvy je přitom analogický k počtu filtrů, a tedy i nových reprezentací původního vstupu.

Vyjma vizuálních, dvourozměrných dat, je možné konvoluci aplikovat i na N-rozměrná data, včetně dat jednorozměrných, tedy sekvenčních, například textu nebo časových řad.

Konvoluční neuronové sítě, dále CNN, nacházejí široké uplatnění nejenom v řešení úloh počítačového vidění, ale ve všech úlohách, kde je možné vstup pojmout jako mapu příznaků, jejichž význam je ovlivněn vzájemnou polohou.
Stejně jako tradiční umělé neuronové sítě, jsou i sítě konvoluční inspirovány biologickými neuronovými sítěmi, jmenovitě organizací vizuální kůry mozkové. (Fukushima, 1980)

Konvolucí rozumíme operaci dvou funkcí jejíž výsledkem je jiná funkce podle vztahu:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(\alpha) g(x - \alpha)\, d\alpha$$

Kde $*$ značí operaci konvoluce, $f(x)$ vstup modelu a $g(x)$ funkci zvanou konvoluční jádro. Výsledná funkce je stejně jako operace nazývána konvolucí. Opakovanou konvolucí $f(x)$ s různými konvolučními jádry je možné vytvořit řadu konvolucí původní funkce, společně tvořících mapu výstupních příznaků.

Algoritmy konvolučních neuronových sítí vycházejí z diskrétní varianty operace, implementované jako operaci dvou N-rozměrných tensorů, mapy vstupních příznaků, reprezentované tensorem A tvaru $(a_0, \dots, a_n)$, a konvolučního jádra K tvaru $(k_0, \dots, k_n)$.

Kde hodnota prvku a_n tvaru tensoru mapy vstupních příznaků označuje její hloubku a prvních $n-1$ prvků tvaru konvolučního jádra značí rozměr použitého okna.

Předpokládejme $k_n = a_n$. Nic nebrání určení $1 \geq k_n \geq a_n$, nicméně postup výpočtu prvků mapy výstupních příznaků je v takovém případě složitější.

Mapa vstupních příznaků A je rozdělena na m oblastí, tensorů tvaru $(k_0, \dots, k_n)$, kde m závisí na tvarech obou tensorů A a K, a na použité délce kroku, značené s.
Rozdělování prvků původního tensoru A probíhá podél všech n os, přičemž hodnota s určuje navýšení hodnoty indexu, přináležejícího zvolené ose.

Vzniklé tensory jsou dále, spolu s tensorem konvolučního jádra K, rozděleny na k_n tensorů rozměru $N-1$ a tvaru $(k_0, \dots, k_{n-1})$, značených $K_i^*, A_i^*; i \in \langle 0, k_n \rangle$.
Hodnota prvku C_j výstupního tensoru C^* odpovídá:

$$C_j = \sum_{i=0}^{k_n} \sum_{r} K_{i_r}^* * A_{i_r}^* + b$$
$$r \in R(K_i^*)$$
$$R(K_i^*) = R(A_i^*)$$

Kde $R(K_i^*)$ je funkcí definující množinu indexů označujících prvky daného tensoru, $K_{i_r}^*$ prvek tensoru K_i^* označený indexem r a $A_{i_r}^*$ prvek tensoru A_i^* označený indexem r.
Je evidentní, že tvar výsledného N-rozměrného tensoru C^*, $(c_0^*, \dots, c_n^*)$, není shodný s tvarem tensoru A, přesněji že:

$$c_i^* < a_i, \qquad i \neq n$$

Rovnost rozměru tensorů v dané ose je možné zajistit rozšířením tensoru A o znaky paddingu. Tvar tensoru C^*, $(c_0^*, \dots, c_n^*)$, je tak závislý nejenom na tvarech mapy vstupních příznaků a konvolučního jádra, ale i na délce kroku a počtu prvků paddingu.
Pokud je původní tensor rozšířen symetricky, konkatenací tensorů paddingu s původním tensorem, je tvar nového tensoru určen:

$$c_i^* = \frac{(a_i - k_i + 2p)}{s} + 1$$

Kde s značí délku kroku a p počet prvků paddingu, o které byl původní tensor rozšířen.
Obvyklé použití paddingu v oblasti konvolučních neuronových sítí rozšiřuje tensor A symetricky, konkatenací tensorů P_i pro jejichž prvky platí:

$$(P_i)_r = c = 0$$
$$r \in R(P_i)$$

Kde $R(P_i)$ definuje množinu indexů všech prvků tensoru P_i.
Je ovšem možné zvolit $c \neq 0$, popřípadě určit hodnotu prvků tensorů P_i vhodnou funkcí.
Stejně tak je možné zavést padding nesymetrický. (TensorFlow community, 2020)

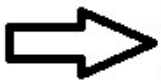
Konkatenací tensorů, vytvořených konvolucí mapy vstupních příznaků a konvolučních jader, je vytvořena N-rozměrná mapa výstupních příznaků, tensor M_{out} tvaru $(l_0, \dots, l_n)$, přičemž hodnota l_i je určena:

$$l_i = c_i^*, \qquad i \neq n$$
$$l_n = \sum_{i=0}^{D} c_{n_i}^*$$

Předpokládejme tensor mapy vstupních příznaků, tvaru $(5, 5, 1)$, odvozený z černobílé bitmapy rozměru $5 * 5$ pixelů, rozšířený pomocí paddingu na tensor tvaru $(6, 6, 1)$ a tenzor konvoluční vrstvy tvaru $(3, 3, 2)$, sestávající ze dvou jader a hodnotou prahu rovnou 0.

Mapu vstupních příznaků, po aplikaci paddingu, je možné reprezentovat jako matici A o rozměrech $6 * 6$, kde hodnota každého prvku odpovídá odstínu šedi, a tensory jader jako dvě matice K_0 a K_1 rozměrů $3 * 3$.

0	0	210	0	0
0	195	0	0	0
0	200	221	220	0
0	200	0	218	0
0	0	205	0	0



0	0	0	0	0	0	0
0	0	0	210	0	0	0
0	0	195	0	0	0	0
0	0	200	221	220	0	0
0	0	200	0	218	0	0
0	0	0	205	0	0	0
0	0	0	0	0	0	0

Fig. 6 Příklad mapy vstupních znaků před a po paddingu

1	0	0
1	1	1
-1	1	-1

K_0

1	1	1
-1	-1	-1
0	-1	1

K_1

Fig. 7 Příklad dvou konvolučních jader

Aplikací konvolučních jader, s hodnotou prahu rovnou 0, získáme tensor mapy výstupních příznaků tvaru $(5, 5, 2)$, tvořený dvěma maticemi rozměru $5 * 5$.

-195	405	15	210	0
-5	174	-4	209	-220
0	621	418	659	2
200	-5	823	234	438
0	205	405	205	218

195	-405	-210	-21	0
5	36	14	-10	0
195	-426	-228	-659	-220
0	426	18	223	2
200	-5	213	13	218

Fig. 8 Příklad mapy výstupních znaků

Analogicky by aplikace konvoluční vrstvy s velikostí okna $3 * 3$ a 32 filtry, bez paddingu a délkou kroku rovnou 1, vytvořila mapu výstupních příznaků jako tensor tvaru $(4, 4, 32)$, kde je každý výstupní kanál mapou odpovědí jednoho použité filtru.

5.4.1 **Vrstvy sdružování**

Vyjma vrstev konvolučních využívají CNN také vrstvy sdružování implementující stejnojmennou unární operaci. Zatímco konvoluční vrstvy mapu příznaků rozšiřují, vrstvy sdružování mapu příznaků redukují. Operace sdružování je tak analogická převzorkování, respektive kompresi vstupu, zvolenou nelineární funkcí.

Postupná komprese mapy vstupních příznaků, spolu s vrstvami konvoluce, umožňuje vytvoření hierarchie prostorových filtrů. Model je tak schopen rozpoznat nejenom jednotlivé vzory vstupu, ale i jejich hierarchie.

Druhou výhodou použití sdružování v CNN je redukce počtu parametrů modelu. Sdružení tak nejenom snižuje nároky modelu na výpočetní prostředky, ale současně brání přeučení.

Oproti konvolucím nejsou parametry operace sdružování optimalizovány během procesu učení. Sdružení je zajišťováno funkcí knihovny, zvolenou během návrhu modelu, jejíž charakteristiky se během procesu učení nemění. Mezi nejobvyklejší funkce, používané operacemi sdružování, patří volba nejvyšší a průměrné hodnoty vstupního tensoru.

Dalším podstatným rozdílem mezi operacemi konvoluce a sdružování je obvyklá volba velikosti kroku větší než 1, popřípadě délky kroku zajišťující výběr, nepřekrývajících se oblastí mapy příznaků. Aplikace operace sdružování tak redukuje rozsah mapy příznaků a počet parametrů modelu.

Formálně je operací sdružování možné definovat podobně jako konvoluci.
Vstupní tensor A je rozdělen na více menších tensorů, tvarů a počtu určených zvolenou velikostí okna a kroku, podle stejných pravidel jako v případě konvoluce, přičemž tvar výstupního tensoru P^* je určený vztahem:

$$p_i^* = \frac{(a_i - p_{w_i})}{s} + 1$$

Kde p_i^*, p_{w_i} a a_i značí rozměr výstupního tensoru, použitého okna a tensoru A podél osy i, a s zvolenou velikost kroku. Hodnota prvku P_S výsledného tensoru P je určena aplikací zvolené funkce $P(x)$:

$$P_S = P(A_S^*)$$

Kde A_S^* značí jeden z tensorů vytvořených rozdělením vstupního tensoru A, a S příslušný index prvku výsledného tensoru. Je evidentní, že operace sdružování je, z hlediska potřebných výpočetních prostředků, méně náročná než konvoluce, ale řídí se řadou stejných pravidel.

Pokračujme v příkladu započatém při popisu konvoluce. Z původní mapy vstupních příznaků tvaru $(5, 5, 1)$ jsme konvolucí, s použitím konvolučních jader K_0 a K_1 ,vytvořili mapu výstupních příznaků tvaru $(5, 5, 2)$. Operace sdružování s délkou kroku rovnou 1 a velikostí okna $2 * 2$ transformuje mapu uvedenou výstupních příznaků na nový tensor tvaru $(4, 4, 2)$, jehož prvky jsou vypočítány zvolenou funkcí.

405	405	210	210
621	621	659	659
621	823	823	659
205	823	823	438

195	36	14	0
195	36	14	0
426	426	223	223
426	426	223	223

Fig. 9 Příklad tensoru vzniklého sdružováním podle maxima

Sdružení podle maxima vytvoří matice nejvyšších hodnot příslušných oblastí mapy výstupních příznaků. Prvky vzniklého tensoru je možné interpretovat jako nejvýznamnější příznaky příslušné oblasti mapy příznaků.

Důsledkem nastavení délky kroku rovné 1, je překrývání oken sdružení vedoucí k méně radikální redukci rozměru tensoru a opakování hodnot řady prvků výsledného tensoru. Některé prvky mapy výstupních příznaků jsou tedy nadbytečné.

95	148	108	50
198	302	321	163
204	464	534	333
100	357	417	274

-42	-141	-57	-8
-48	-151	-221	-222
49	-53	-162	-164
155	163	117	114

Fig. 10 Příklad tensoru vzniklého sdružováním podle průměru

Analogicky, operace sdružování podle průměrné hodnoty, vytvoří tensor shodného tvaru, tvořeného prvky s hodnotami rovnými průměru hodnot prvků příslušné oblasti mapy příznaků. Oproti sdružení podle maxima je sdružení podle průměrné hodnoty příznaků v praxi méně obvyklé. (Chollet, 2019)

5.4.2 **Alternativní konvoluční operace**

Navzdory širokému přijetí CNN, zůstává řada úloh, k jejichž řešení nejsou tradiční implementace konvolučních operací, v důsledku omezení kladených na čas nebo dostupné výpočetní prostředky, vhodné. Především se jedná o praktické aplikace CNN v oblastech robotiky, dopravní automatizace a spotřební elektroniky.

Ze základních principů konvoluce tak vychází řada alternativních postupů, vytvořených s cílem snížit výpočetní náročnost, popřípadě zlepšit další vlastnosti CNN.

Mezi nejznámější patří separovatelná konvoluce, vytvářející mapu výstupních příznaků postupnou aplikací více menších konvolučních jader, jejichž součinem je původní tensor. Počet elementárních operací nutných k vykonání součinu konvoluční jádra a oblasti mapy vstupních příznaků je závislý na zvolené velikosti okna konvoluce.

Podstatnou nevýhodou základní separovatelné konvoluce je požadavek na separovatelnost konvolučního jádra, kterou většina možných konvolučních jader neuspokojuje.

Model využívající separovatelnou konvoluci tedy nemůže využít celou šíři potenciálních konvolučních jader, a je omezen na menší škálu konvolucí odvozených příznaků.

Ze separovatelné konvoluce vychází separovatelná konvoluce podél hloubky, jež je možné, oproti původní metodě, aplikovat na jakékoliv konvoluční jádro.
Konvoluce separovatelná podél hloubky rozděluje operaci konvoluce na konvoluce podél hloubky a bodové konvoluce.

Konvoluce podél hloubky rozděluje mapu vstupních příznaků podle osy hloubky na a_n tensorů, na které následně aplikuje příslušná konvoluční jádra. Následná aplikace bodové konvoluce, definované jako operace konvoluce s rozměrem okna 1, vytváří lineární kombinaci vzniklých tensorů.

Modely využívající konvoluci separovatelnou podél hloubky dosahují výsledků srovnatelných s klasickými CNN, při použití podstatně menšího počtu elementárních operací a optimalizovaných parametrů. (Howard, a další, 2017)

Dalším alternativní implementací konvolučních operací jsou takzvané zploštěné CNN, pracující s řadou postupně aplikovaných, jednorozměrných konvolučních jader.
Stejně jako v případě konvoluce separovatelné podél hloubky, i výsledky zploštěných CNN jsou srovnatelné se standartní implementací konvolučních operací. (Jin, Dundar, & Culurciello, 2014)

6 Vlastní práce

Vývoj konvoluční neuronové sítě vyžadoval otestování řady možných konfigurací modelů a jejich prvků, především parametrů vrstev neuronové sítě a optimalizačního algoritmu. Současně bylo třeba evidovat otestované konfigurace a jejich ohodnocení. I relativně malá změna struktury modelu navíc mnohdy vyžadovala úpravu formátu vstupních dat, nebo přehodnocení vhodnosti navazující prvků.

Řada konfigurací se ukázala během vývoje nejenom jako nevhodná, ale jako v použitém frameworku neimplementovatelná. Celou řadu souvisejících, relativně jednoduchých úkonů bylo přitom nutné opakovat pro každou testovanou konfiguraci, bez odchýlení od předchozího postupu. Jejich automatizace se zdála přínosnou nejenom pro kvalitu navržených modelů, ale i pro kvalitu samotné práce.

K zjednodušení tvorby, analýzy a správy modelů, spolu s příslušnými množinami vstupních dat, byl vytvořen program, automatizující tvorbu modelů i předzpracování dat v řadě možných konfigurací. Podstatnou výhodou programu, oproti přímému použití frameworků TensorFlow a Keras, bylo zajištění vytvoření relativně složitého a spustitelného, byť ne nutně praktického modelu, i při zadání minima argumentů.

Program byl napsán v jazyce Python verze 3.6, jež je upřednostňovaným jazykem pro vývoj programů využívajících frameworků TensorFlow a Keras.
Proces učení, zhodnocení a testování modelů, probíhal na osobním počítači vybaveném operačním systémem Windows 10. Pro práci nejdůležitějším výpočetním prostředkem byla grafická karta GTX 1070 firmy NVidia, disponující 8 GB paměti a 1920 výpočetními jádry CUDA. (NVIDIA Corporation, 2020)

Návrh konvoluční neuronové sítě, spolu s volbou technik předzpracování dat, vycházel ze zjištěných charakteristik záznamů databáze Safety Gate. Vhodné předzpracování vstupních dat vyžadovalo znalost struktury textu popisů rizik a závad.

Během vývoje umělé neuronové sítě, softwaru zajišťujícího část předzpracování vstupních dat, analýzu kvality modelu a zpracování výstupních statistik, došlo k podstatným změnám frameworku Keras i knihovny TensorFlow.
Podstatnou část programu tak bylo třeba změnit. Další součásti se navíc ukázaly nepotřebnými, neboť je bylo možné nahradit importovanými funkcemi.

6.1 Popis vstupních dat

Databáze Safety Gate obsahuje v současnosti[22] 26057 záznamů o závadných produktech a jejich charakteristikách. Při zahájení vývoje programu byl použit menší počet, v té době již existujících, záznamů. Učení, validace a testování navržených modelů již probíhalo s novější verzí databáze v uvedeném rozsahu.

Rozhraní portálu umožňuje selekci záznamů z databáze podle řady kritérií, mimo jiné roku podání hlášení, názvu produktu, značky, členské země podávající hlášení a země původu.

6.1.1 Postup exportu záznamů databáze

Práce se záznamy je podstatně komplikována omezeními kladenými na jejich export, který je možný pouze postupně, v maximálním rozsahu deseti tisíc záznamů, a ve značně nestandardním formátu.

Navzdory faktu, že je volba exportu označena jako „Export to Excel“, a odkazuje na soubor s příponou xls, se nejedná o validní příklad formátu Microsoft Excel 97-2003 Worksheet, ale o zlomek dokumentu formátu HTML, respektive pouze kódu definujícího strukturu tabulky a obsah jednotlivých polí, bez deklarace typu dokumentu vyžadované standardem. (W3C, 2020)

[22] Poslední kontrola záznamů provedena 29.1.2020

```
<td valign="top">2010</td>
<td valign="top">52</td>
<td valign="top">
 Products with serious risks
</td>
<td valign="top"> Consumer</td>
<td valign="top">1984&#x2f;10</td>
<td valign="top">France</td>
<td valign="top">Hobby&#x2f;sports equipment</td>
<td valign="top">Laser sight for air gun&#xa;&#xd;&#xa;&#xd;</td>
```

Fig. 11 Část neupraveného exportu záznamu Safety Gate

Před dalším zpracováním bylo nutné převést soubory do formátu Comma Separated Values, který je pro další úkony vhodnější, a následně je konkatenovat do jedné tabulky.

6.1.2 **Vlastnosti atributů tabulky**

Záznamy nově vzniklé, konkatenované tabulky, sestávaly z 23 atributů, obsahujících informace o produktech, jejich závadách, okolnostech a čase jejich nálezu a o opatření přijatých členskými zeměmi EU po objevení závady.

Většina atributů nebyla z hlediska předmětu práce, ani analýzy záznamů databáze podstatná, a jejich obsah nebyl dále zpracován.

Atribut „Alert number“, tvořený řetězcem znaků unikátním pro jednotlivé záznamy, odpovídal primárnímu klíči tabulky. Formát obsahu sloupce „Alert number“ se během provozu databáze z neznámých důvodů změnil. Hodnoty sloupce ovšem zůstaly pro dříve vytvořené záznamy nezměněny.

Hodnotou atributů popisu produktu, popisu nalezené závady, závadou působeného rizika a dalších okolností příslušného hlášení, byl text v anglickém jazyce. Hodnoty kategorických proměnných, včetně kategorie produktu, působeného rizika a členské země ohlašující rizikový produkt, také využívaly anglickou terminologii.

Z původních 23 atributů se tři, „Technical defect“ a „Risk“ a „Description“, zdály vhodné k tvorbě příznaků. Jmenované sloupce obsahovaly text popisující specifické vlastnosti produktu, závady a závadou působeného rizika proměnlivé délky.

Ačkoliv část popisů sestávala z úplných vět, celá řada záznamů obsahovala v daném sloupci pouze označení šarže nebo jiné vlastnosti rizikového produktu. V některých případech dokonce chyběl popis úplně. Další záznamy obsahovaly v příslušných sloupcích URL odkazy k doplňujícím informacím.

VX1100 : the recall concerns PWCs built from 2005 to 2008Click here to view year, model, model code, Primary ID number range

*Fig. 12 Obsah sloupce „Description “*záznamu 0493/08

Označení jednotlivých kategorií produktů, obsažená v atributu „Category“ nebyla zastoupena v databázi stejnou mírou. Více než čtvrtina záznamů nesla označení „Toys“. Opačným extrémem bylo označení „Explosive atmospheres equipment“ které nesl pouze jediný záznam databáze.

Podobná, ačkoli ne tak extrémní nevyrovnanost, byla zjištěna i pro hodnoty atributu „Alert submitted by“ označující členskou zemi ohlašující nález nebezpečného produktu.
Z části bylo možné jev vysvětlit demografickou situací členských zemí. Nejvíce záznamů, více než třináct procent, uvádělo jako ohlašující zemi Německo, nejlidnatější a ekonomicky nejvýznamnější stát EU.

Nicméně Maďarsko, jež se populací i řadou dalších metrik, řadí mezi menší členské země, označovalo jako ohlašující zemi přes deset procent všech hlášení. Což byl zhruba trojnásobek počtu záznamů produktů ohlášených z České republiky nebo Švédska, a téměř čtyřikrát více než počet hlášení z Itálie.

Kontingenční tabulka dále odhalila závislost mezi ohlašující zemí a označením kategorie produktu, přičemž některé kategorie byly používány pouze v hlášení jedné členské země.

Není přitom důvod se domnívat, že srovnatelné závislosti neexistovaly i mezi hodnotami jiných atributů, například zemí původu a kategorií produktu.

Další analýza nevyrovnanosti označení kategorií produktů, anomálního počtu záznamů pocházející ze specifických zemí a vztahu mezi ohlašující zemí a kategorií produktu, ovšem přesahovala rozsah práce. I když by si zasloužil podrobnější šetření, nebyla další analýza zmíněných jevů, ani dalších vlastností záznamů databáze Safety Gate provedena. Vytvořená kontingenční tabulka je součástí příloh práce.

Vyjma již zmíněné nevyrovnanosti a závislostí mezi hodnotami některých atributů obsahovaly záznamy exportované databáze celou řadu dalších nežádoucích charakteristik, jejichž výskyt bylo třeba zohlednit. Jednalo se především o velký počet chybějících údajů některých atributů, přítomnost HTML značek uvnitř textu a nesprávná označení zemí původu produktů.

Vzhledem k předmětu práce byla nejzávažnějším nedostatkem záznamů nekonzistence významu sloupců „Technical defect“ a „Risk“ a „Description“, jejichž obsah, text popisující závadu a její následky, se zdál vhodný ke tvorbě příznaků.

Popis rizika působeného závadou byl u některých záznamů přítomen ve sloupci „Technical defect“, zatímco další záznamy neobsahovaly v jednom, nebo více, ze jmenovaných sloupců text žádný.

Pro řešení úlohy byla dále závažná chybějící pozorování sloupce „Description“ který neobsahoval ve více než 21 tisících záznamech žádný text. Sloupec „Description“ se tak stal pro potřeby tvorby příznaků ne příliš vhodným, neboť jeho využití by znamenalo vyřazení většiny záznamů původní množiny dat.

Jedním z možných řešení chybějících hodnot sloupce bylo zavedení nového, syntetického sloupce, vytvořeného konkatenací obsahu sloupce „Description“ a jiného sloupce stejného datového typu a významu proměnné. Zavedení syntetického sloupce by ovšem vyžadovalo reinterpretaci vstupních dat.

V důsledku výše uvedeného velkého počtu chybějících údajů sloupce „Description“ by navíc většina hodnot nově vytvořeného sloupce byla shodná s hodnotami druhého rodičovského sloupce. Sloupec „Description“ se tak zdál, oproti sloupcům „Risk type“ a „Technical defect“, méně vhodný k tvorbě příznaků.

Obsah většiny sloupců nebyl, vzhledem k předmětu práce, podstatný. Jednalo se mimo jiné o zemi původu produktu, odkaz URL na původní verzi záznamu a týden podání hlášení.

Nepodstatné sloupce tabulky byly, spolu se sloupcem „Description“, před dalším zpracováním dat odstraněny. V tabulce byly ponechány pouze sloupce: „Alert number“, „Risk type“, „Technical defect“ a „Risk“, které byly následně přejmenovány. Ostatní sloupce byly z tabulky odstraněny spolu se záznamy obsahující chybějící pozorování jednoho nebo více jmenovaných proměnných.

Původní název sloupce	Nový název sloupce	Typ dat	Popis proměnné
Alert number	alert_number	Identifikační / text	Řetězec znaků označujících záznam
Risk type	risk_type	Text	Výčet typů rizik působených produktem
Technical defect	defect_desc	Text	Popis závady
Risk	risk_desc	Text	Popis rizika působeného produktem

Tabulka 7 Popis sloupců obsahujících pro práci významné údaje

Sloupec „Alert number“ byl dále využíván během ladění programu k ověření fungování procesu předzpracování dat. Nekonzistentní obsah sloupců „Risk“ a „Technical defect“, spolu s mnoha chybějícími hodnotami, negativně ovlivňoval kvalitu vstupních dat a mohl značně redukovat jejich rozsah.

6.2 Předzpracování vstupních dat

Před přivedením na vstup umělé neuronové sítě je třeba vstupní data upravit do podoby, která umožní jejich zpracování. V případě klasifikace textu je nutné vytvořit vhodnou reprezentaci řetězců znaků jako čísel.

Navržená konvoluční neuronová síť vyžaduje předzpracování ve třech krocích: očištění, tokenizaci a embedding. První dva kroky předzpracování byly implementovány metodou `´import_data_text´` a pomocnými metodami z jednoduché knihovny funkcí zpracování dat, napsané pro potřeby práce.

Zvolený soubor csv, obsahující záznamy databáze Safety Gate, je programem načten jako objekt třídy DataFrame, a zpracováván jako celek, a to až do rozdělení na trénovací, validační a testovací množinu dat. (Pandas developers, 2020)

Alternativně program umožňuje načíst samostatný soubor testovacích příkladů.
S objekty třídy DataFrame je možné manipulovat podobně jako s tabulkami relačních databázových systémů a implementují všechny operace relační algebry.

Oproti datovým strukturám základní knihovny jazyka Python poskytuje DataFrame řadu zabudovaných funkcí, usnadňujících manipulaci se záznamy a analýzu jejich charakteristik. Další výhodou objektů třídy DataFrame je snadné uložení záznamů tabulky do souborů různého formátu, i jejich převedení do podoby tensorů.

Obsah sloupců, označených uživatelem k využití při tvorbě příznaků, je nejprve očištěn od znaků a jejich řetězců, nevhodných k tvorbě příznaků.

V základním nastavení jsou odstraněny pouze řetězce odpovídající značkám jazyka HTML. Ve striktním nastavení jsou odstraněny i všechny závorky, interpunkce a symboly matematických operací.

Nehledě na zvolené nastavení je text následně převeden do minuskulí.

Program následně vytvoří seznam označení příkladů, v podobě datového slovníku, vycházející z řetězců uživatelem zvoleného sloupce.
V závislosti na typu klasifikační úlohy program ponechá pouze jedno označení kategorie, nahrazené příslušným indexem z již vytvořeného slovníku nebo polem binárních hodnot, kde 1 určuje označení kategorií příslušné danému záznamu.

V případě, že uživatel označil více sloupců k tvorbě příznaků, je jejich obsah sloučen do jednoho sloupce. Je v principu možné ponechat více zvolených sloupců a připravit množinu dat s více samostatnými příznaky. Nicméně pro zadanou klasifikační úlohu nebylo takového postupu třeba.

Záznamy s chybějícím pozorováním ve zvoleném sloupci jsou následně odstraněny a zbývající záznamy jsou náhodně promíchány funkcí knihovny pandas. (Pandas developers, 2020)

Pro sloupec příznaků je vytvořen datový slovník, obsahující všechny přítomné unikátní řetězce, dále slova, oddělené znakem mezery. Do slovníku je přidán speciální řetězec zastupující případný padding původního textu.

```
"<PAD>"
```

Fig. 13 Řetězec paddingu použitého k dorovnání rozsahu prvků množiny dat

Slova jsou ve slovníku označena číselným indexem, jehož hodnota vychází z pozice slov v původních textech, ale není indikátorem významu slova nebo frekvence, s jakou se vyskytuje v původní množině dat.

Program vypočítá maximální délku textu obsaženého ve sloupci příznaků a podle volby uživatele dorovná délku ostatních záznamů paddingem, nebo je zkrátí na předem určenou délku, odstraněním přebytečných slov na konci.

Záznamy jsou následně tokenizovány nahrazením jednotlivých slov jejich indexy ve vytvořeném slovníku, a následně převedeny na pole číslic s počtem prvků, rovných počtu slov nejdelšího textu sloupce příznaků.

Texty sloupce příznaků jsou následně tokenizovány. Každý záznam je nahrazen polem hodnot typu integer, jejichž hodnoty jsou určeny indexem příslušných slov v předem vytvořeném slovníku.

Tabulka je následně rozdělena na trénovací, validační a testovací množinu dat, pokud není poskytnuta samostatná testovací množina. Rozsah vytvořených podmnožin je určen jako:

$$|T| = (d - 2) * \frac{|\chi|}{d}$$

$$|Test| = |V| = \frac{|\chi|}{d}$$

Kde d značí uživatelem zvolený jmenovatel, $|T|$ rozsah množiny trénovacích, $|V|$ rozsah množiny validačních a $|Test|$ rozsah množiny testovací dat.

Předcházející krok, promíchání záznamů tabulky, zajišťoval odlišné složení záznamů jednotlivých množin, při každém načtení souboru záznamů databáze Safety Gate. Nevýhodou zvoleného postupu ovšem byla nemožnost obnovení procesu učení, či zahájení procesu testování již existujícího modelu během příštího spuštění programu.

Promíchání záznamů, před rozdělením na jednotlivé podmnožiny, může vést ke „kontaminaci“ nové testovací množiny dat, záznamy přítomnými v dříve použitých množinách dat trénovacích, nebo validačních. Během testování by model mohl obdržet příklady použité již během procesu učení. Hodnoty metrik by tak mohly být zavádějící.

V případě přerušovaného učení modelu by bylo vhodnější vytvořit samostatnou množinu testovacích dat, sestavenou z předem vybraných prvků původní množiny. Další možnou alternativou by bylo využití k-násobné křížové validace.

Nicméně, díky relativně krátkým dobám učení navržených modelů, se případná ztráta času, způsobená přerušením a následným znovuzahájením procesu učení, zdála jako únosná.

Kompletní výčet hodnot parametrů metody ´`import_data_text`´, použitých k předzpracování dat pro potřeby práce, je popsán v příloze B.

Po úvodním předzpracováním sestávala množina vstupních dat z 26053 záznamů, složených z celkem 12225 unikátních slov, s průměrně 50.17 slovy v každém záznamu.

Počet záznamů původní množiny dat	26053
Počet záznamů trénovací množiny dat	15633
Počet záznamů validační množiny dat	5210
Počet záznamů testovací množiny dat	5210
Maximální délka textu	369
Průměrná délka textu	50.17
Počet unikátních slov	12225
Počet kategorií	19

Tabulka 8 Charakteristiky původní množiny dat po předzpracování

Poslední krok předzpracování, embedding, je zajištěn první vrstvou navržených modelů a je popsán v příslušných kapitolách.

Je nutné zdůraznit, že omezená délka záznamů databáze, ve spojení s relativně malým počtem unikátních slov, neumožňuje plně využít možnosti CNN k řešení zadané úlohy a demonstrovat jeho výhody oproti alternativním postupům.

Zadanou úlohu by mohlo být možné vyřešit i za pomoci klasické umělé neuronové sítě a vhodně zvoleného embeddingu. Vlastnosti výsledného modelu by se mohly, do značné míry, blížit srovnatelné CNN.

Jistou nevýhodou řešení, nevyužívajícího konvolučních vrstev, by mohla být zvýšená tendence k přeučení, způsobená navýšením relativního počtu parametrů první hustě

zapojené vrstvy vzhledem ke zbytku modelu. Změny hodnot parametrů během procesu učení, určené adaptivním algoritmem v závislosti na vlivu parametrů na hodnotu chybové funkce by byly ve srovnání s konvolučním modelem větší. Nicméně srovnání tradičních neuronových sítí a CNN přesahuje předmět práce.

6.3 Správa a spouštění modelů

Rozhraní mezi navrženým modelem a uživatelem zajišťovala třída ´`TextClassifier`´, vytvořená podle návrhového vzoru fasáda. (w3sDesign, 2018)
Samotné navržené modely byly objekty třídy Sequential. (Keras Team, 2020)
Hlavním účelem třídy, je zjednodušení tvorby a správy konvolučních neuronových sítí vhodných ke klasifikaci tokenizovaného textu, včetně specifikace hyperparametrů a množin dat použitých při procesu učení modelu.

Metody třídy, spravující inicializaci, učení a testování modelů, umožňují snadno přizpůsobit model úloze a specifikům množiny dat. Řada předdefinovaných pravidel navíc redukuje počet voleb, nutných k vytvoření první verze navrhovaného modelu.

Vlastnosti modelu a procesu učení, dále jen hyperparametry, je možné nastavit dvojím způsobem. Během vytvoření instance třídy jako parametry konstruktoru nebo specifikací parametrů příslušných metod. Přičemž nastavení, zadaná jako parametry metod, jsou použita přednostně.

Parametr konstruktoru ´`build_args`´ odpovídá hyperparametrům definujícím architekturu modelu, včetně topologie umělé neuronové sítě. Parametr ´`train_args`´ zastupuje hyperparametry procesu učení, včetně počtu epoch a velikosti dávky.

Oba parametry konstruktoru jsou slovníky, sestávajících z párů názvů hyperparametrů a příslušných hodnot. Nezahrnuté hyperparametry nabývají výchozích hodnot.

```
train_args = {
        'training_data':training_dataset,
        'evaluation_data':evaluation_dataset,
        'epochs':100,
        'steps_per_epoch':20,
        'batch_size':32,
        'label_dictionary':label_dictionary,
        'data_column':'risk_desc',
        'label_column':'risk_type',
        'shuffle':True
        }
```

Fig. 14 Příklad slovníku hyperparametrů procesu učení modelu

Zavedení jednotných výchozích nastavení pro většinu hyperparametrů usnadňuje následné srovnání kvality navržených modelů, neboť se vlastnosti explicitně nezadané uživatelem mezi modely shodují. Výchozí nastavení hyperparametrů dále umožňuje podstatně zkrátit čas, nutný k navržení nového nebo úpravě stávajícího modelu.

Metoda ´`build_model`´ vytváří a inicializuje model podle instrukcí uživatele.
Volbou parametrů metody je možné nastavit 23 aspektů vytvářeného modelu, včetně počtu a rozsahu hustě zapojených a konvolučních vrstev, jejich aktivačních funkcí a regularizace, úpravy vstupu modelu i použitého optimalizačního algoritmu.

Voláním metody ´`train_model`´ je zahájen proces učení již inicializovaného modelu. Stejně jako v případě tvorby modelu, i hyperparametry procesu učení je možné specifikovat slovníkem během vytvoření instance třídy, nebo hodnotami parametrů příslušné metody, přičemž hyperparametry nespecifikované nabývají výchozích hodnot.

Metoda ´`train_model`´ současně zajišťuje kompatibilitu datového typu objektů, reprezentujících trénovací a validační množinu dat s navrženým modelem.
Množiny dat nevhodného datového typu předává funkci ´`data_to_tensors`´ z již zmíněné knihovny pro zpracování dat, spolu s názvy sloupců příznaků, označení a hodnotami dalších parametrů.

Podle zadaných parametrů vytváří funkce ´data_to_tensors´ nové objekty třídy Dataset, jež je možné přímo použít k učení modelu. (TensorFlow community, 2020)

```
build_args = {
              'layers':[200,100,50],
              'convolution_layers':[128,512],
              'embedding_dim':5,
              'optimizer':keras.optimizers.Adam(),
              'model_name':'alpha',
              'vocabulary_size':len(word_dictionary)+1,
              'kernel_size':10,
              'pool_size':5,
              'max_pooling':True,
              'exclusive_categories': False,
              'input_size':max_desc_len,
              'categories':len(label_dictionary)
              }
```

Fig. 15 Příklad slovníku hyperparametrů konstrukce modelu "alpha"

V neposlední řadě metoda ´train_model´ spravuje dvojím způsobem záznam charakteristik modelu a jejich vývoje během procesu učení.

Soubor, popisující vlastnosti modelu a jejich vývoj během procesu učení, včetně hodnot vybraných metrik, parametrů a gradientů, je během procesu učení uložen do zvoleného adresáře pro vizualizaci a další analýzu v programu Tensorboard.

Záznamy hodnot metrik, naměřených během učení a validace modelu, jsou navíc obsaženy v objektu navraceném metodou. (TensorFlow community, 2020)

6.4 Navržené modely

K řešení úlohy byly navrženy čtyři, konvoluční, dopředné, neuronové sítě, s rozdílnými rozměry a počty vrstev, označené prvními čtyřmi písmeny řecké abecedy.

Parametry všech navržených modelů byly optimalizovány algoritmem Adam v jeho základním nastavení podle implementace frameworků Keras a TensorFlow. (TensorFlow community, 2020)

Využití CNN k analýze sekvenčních dat vychází ze stejných obecných principů jako v případě ostatních úloh. Text, zastupující jednorozměrnou mapu vstupních příznaků, je konvolučními operacemi převeden na mapu příznaků výstupních. Každý prvek tensoru mapy vstupních příznaků přitom odpovídá jednomu slovu původního textu.

Počet použitých konvolučních jader, určuje hodnotu posledního prvku tvaru tensoru mapy výstupních příznaků. Jednotlivá konvoluční jádra je možné interpretovat jako pravidla pro ohodnocení lokálních vztahů slov.

Úlohy zpracování sekvenčních dat, včetně textu, je možné řešit i s pomocí RNN, jejichž architektura umožňuje zpracování sekvence hodnot po jednotlivých prvcích a zohlednění jejich pořadí při výpočtu odezvy sítě.

Podstatnou nevýhodou RNN je jejich nezanedbatelný počet parametrů, a z něho plynoucí požadavky na výpočetní prostředky. Počet parametrů určujících vlastnosti jedné rekurentní vrstvy typu GRU je roven:

$$N_{GRU_w} = 3 * (n^2 + nm + n)$$

Kde N_{GRU_w} značí počet parametrů vrstvy, m je rozměr vstupního vektoru a n počet skrytých výpočetních jednotek, buněk, vrstvy. Vhodnost RNN k řešení úloh je tak závislá nejenom na vlastnostech samotné úlohy, ale i na dostupnosti výpočetních prostředků. (Dey & Fathi, 2017)

Pro většinu úloh zpracování textu dosahují CNN a RNN srovnatelných výsledků. RNN využívající GRU jednotek ovšem překonává CNN v případě úloh jejichž řešení závisí na analýze rozsáhlejších sémantických struktur, například složených vět s několikanásobným záporem. (Yin, Kann, Yu, & Schütze, 2017)

6.4.1 **Obecný popis navržených modelů**

První vrstva modelů zajišťovala embedding tokenizovaného textu.
Indexy prvků slovníku, tokeny, jsou vrstvou převedeny do podoby n-rozměrných vektorů, kde n je rovno rozměru výstupu vrstvy. Rozměr vstupu vrstvy, určený příslušným hyperparametrem, musí být roven nebo větší počtu prvků slovníku.

Každý prvek množiny dat přivedený na vstup modelu je nahrazen tensorem shodné délky, složeným z vektorů přiřazeným každému tokenu, při zachování vzájemné polohy tokenů, a jim přiřazených vektorů, v obou tensorech. Ostatní hyperparametry vrstvy byly ponechány v příslušném výchozím nastavení. (TensorFlow community, 2020)

Hodnoty hyperparametrů rozměru a délky vstupu vrstvy embeddingu, pevně určené před samotným vytvořením modelu, kladly podstatná omezení na použití modelu.

Rozsah vstupu vrstvy, odpovídají počtu prvků slovníku unikátních slov původní množiny dat, omezuje počet vrstvou vytvořených vektorů. Model je schopen jako vstup přijímat pouze slova, k jejichž tokenům existuje příslušný vektor embeddingu. Ostatní tokenizovaná slova není schopen využít jako příznaky.

Určení vyšší hodnoty hyperparametru by umožnilo klasifikaci záznamů obsahujících další slova. Nicméně hodnoty prvků, jim příslušných vektorů embeddingu, by během procesu učení zůstaly v nezměněné, náhodné podobě, a jako příznaky nepoužitelné.

Hodnota hyperparametru délky vstupu určovala rozsah tokenizovaných záznamů, které byl model schopen přijímat jako vstup, a následně klasifikovat. Záznamy kratšího než požadovaného rozsahu, je možné doplnit na požadovanou délku pomocí tokenů paddingu.

Delší záznamy by ovšem bylo třeba na požadovanou délku zkrátit. Výsledný text by tak mohl být nevhodně klasifikován, v důsledku ztráty informace obsažené v odstraněné části.

Následovaly vrstvy jednorozměrné konvoluce, rozsahu a počtu definovaného zadanými hyperparametry modelu, každá s navazující vrstvou jednorozměrného sdružování.

Vlastnosti vrstev konvoluce a sdružování byly určeny výchozími hodnotami příslušných hyperparametrů definovanými knihovnami Keras a TensorFlow.

Vrstvy obou typů nevyužívaly paddingu a přijímaly vstup ve formátu „channels last“, tedy s kanály jako posledním prvkem tvaru tensoru. (TensorFlow community, 2020)

Rozhraní mezi poslední vrstvou sdružování a dalšími vrstvami modelu zajišťovala vrstva Flatten, přizpůsobující tvar výstupního tensoru požadavkům následujících vrstev. (TensorFlow community, 2020)

Hustě zapojené vrstvy, s aktivační funkcí jednotek neuronů ReLU, navazovaly přímo na vrstvu Flatten. Knihovna Keras i třída ´`TextClassifier`´ umožňují snadné zavedení regularizace hodnot aktivačních funkcí i postsynaptického potenciálu neuronů pro zamezení, omezení nebo přeučení modelu.
Nicméně navržené modely ani jednu z uvedených možností nevyužívaly.

Stejně tak bylo možné zavést ensambling modelů, vložením vrstev implementujících dropout mezi vrstvy hustě zapojené. Nicméně ani tato možnost nebyla zvolena.

Poslední, výstupní vrstva modelu, sestávala z m neuronů, s logistickou aktivační funkcí. Kde m odpovídalo počtu druhů rizik nalezených ve vstupní množině dat.

Zvolená aktivační funkce výstupní vrstvy umožňovala modelu přiřadit každému příkladu libovolný počet označení. Výstupem modelu byl vektor rozměru $m,$ kde každý prvek odpovídal pravděpodobnosti označení s příslušným indexem pro daný příklad.

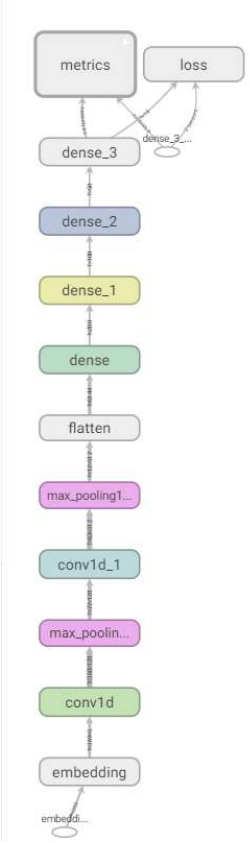


Fig. 16 Vizualizace hlavního výpočetního grafu modelu "alpha"

Jako chybová funkce byla zvolena binární křížová entropie, jejíž hodnota pro jednotlivé prvky vektoru výstupu modelu nezohledňuje hodnoty vypočítané pro prvky ostatní. Binární křížová entropie je tak vhodná pro úlohy, kde pravděpodobnost jednoho označení příkladu není ovlivněna pravděpodobností označení jiných.

6.4.2 **Srovnání hyperparametrů modelů**

Jednotlivé navržené modely se mezi sebou shodovaly ve většině ohledů, včetně počtu a rozsahu konvolučních vrstev, vrstev sdružování a jim příslušným hyperparametrům. Navržené modely se vzájemně odlišovaly pouze počtem hustě zapojených vrstev, jejich rozměry a rozměrem výstupu vrstvy embeddingu.

Pevným určením hodnot většiny hyperparametrů byla podstatně zjednodušena následná analýza charakteristik modelů a jejich srovnání. Především snížením počtu možných kombinací hyperparametrů, a tedy i počtu modelů, které bylo třeba podrobit analýze.

Návrh menšího počtu, ve většině hyperparametrů shodných, modelů dále zjednodušil určení vlivu hodnot volných hyperparametrů na vlastnosti modelu.

Navržené modely zahrnovaly tři nebo sedm, hustě zapojených skrytých vrstev, sestávajících z různého počtu neuronů.

Určení tří skrytých vrstev, jako minima pro navržené modely, vycházelo z významné vlastnosti umělých neuronových sítí, že model zahrnující tři hustě zapojené vrstvy je schopen, za předpokladu jejich dostatečného rozsahu, aproximovat jakoukoliv funkci, a tedy řešit jakoukoliv úlohu klasifikace. (Duda O., Hart, & David, 2001)

Parametr	**Hodnota**
Funkce vrstvy sdružování	Maximální
Rozměr okna vrstev sdružování	5
Rozsah konvolučních vrstev	128, 512
Rozměr okna konvolučních vrstev	10
Rozměr vstupu embeddingu	12226
Délka vstupu embeddingu	369
Aktivační funkce skrytých neuronů	ReLU

Tabulka 9 Hodnoty hyperparametrů společných všem navrženým modelům

Hustě zapojené vrstvy, společně s vrstvou embeddingu, zahrnují většinu parametrů modelu, jejichž optimalizace je cílem procesu učení. Rozsah modelu, a tedy i jeho tendence k přeučení, je tak z větší části určen právě jejich vlastnostmi.

Pro model „alpha“, sestávajícího z tří hustě zapojených a dvou konvolučních vrstev, spolu s příslušnými vrstvami sdružení a embeddingu a výstupu, je celkový počet parametrů obou konvolučních vrstev roven 662400. Vlastnosti první hustě zapojené vrstvy jsou ovšem určeny hodnotou 1229000 parametrů.

I v případě relativně jednoduchých modelů, jakým je i „alpha“, tak naprostá většina parametrů modelu přísluší hustě zapojeným vrstvám. Hodnota každého parametru navíc podléhá optimalizaci zvoleným algoritmem. Rozsah a počet hustě zapojených vrstev tak má podstatný vliv na čas nutný k vykonání jedné epochy procesu učení.

Model	Rozměry hustě zapojených vrstev	Rozměr výstupu embeddingu
Alfa	200, 100, 50	5
Beta	2000, 500, 400, 300, 200, 100, 50	5
Gama	200, 100, 50	20
Delta	2000, 500, 400, 300, 200, 100, 50	20

Tabulka 10 Hodnoty hyperparametrů specifických pro jednotlivé modely

Hyperparametry, určující rozměry vstupu a výstupu vrstvy embeddingu, ovlivňují počet vah modelu dvojím způsobem. Samotné parametry vrstvy embeddingu odpovídají prvkům vektorů, přiřazených jednotlivým unikátním tokenům z množiny dat, počet parametrů vrstvy je tak určen oběma hodnotami, jako:

$$|\mathrm{Emb}_{in}| * |\mathrm{Emb}_{out}| = N_{emb_w}$$

Kde N_{emb_w} značí počet vah vrstvy embeddingu, $|\mathrm{Emb}_{in}|$ rozsah vstupu a $|\mathrm{Emb}_{out}|$ rozsah výstupu vrstvy. Počet parametrů vrstvy embeddingu modelu „alpha“ je tak roven 61130.

Vrstva embeddingu navíc vytváří mapu vstupních příznaků pro první vrstvu konvoluční, jako tensor tvaru $(|\mathrm{Model}_{in}|, |\ \mathrm{Emb}_{out}|)$, kde $|\mathrm{Model}_{in}|$ značí rozsah vstupu modelu, v případě navržených modelů délku, počet tokenů, tokenizovaného textu.

Vrstvy konvolucí knihovny vytváří konvoluční jádra tvaru (k_0, k_1) kde k_0 značí rozměr okna konvoluční vrstvy a k_1 je rovno poslednímu prvku tvaru výstupního tensoru předcházející vrstvy.

Pro první konvoluční vrstvu navržených modelů je tak $k_1 = |\mathrm{Emb}_{out}|$.
Rozměr embeddingu tak ovlivňuje nejenom rozměr mapy výstupních příznaků, ale i počet parametrů navazující konvoluční vrstvy.

Důsledkem vlivu vrstvy embeddingu byl rozdílný počet parametrů mezi modely se shodným počtem vrstev. Model „delta“, využívající embeddingu s výstupem rozměru 20, tak zahrnoval více parametrů než model „beta“ se stejným počtem vrstev.

Model	**Počet parametrů**
Alpha	1,978,649
Beta	14,421,049
Gama	2,586,419
Delta	15,028,819

Tabulka 11 Počet parametrů navržených modelů

Je nutné zdůraznit, že všechny navržené modely jsou relativně jednoduché co do počtu parametrů, vrstev i dalších vlastností architektury oproti řadě modelů popsaných v literatuře a používaných v praxi.

6.5 Proces učení modelů

Proces učení navržených modelů probíhal po 100 epoch. Před zahájením každé epochy byly příklady trénovací množiny náhodně promíchány.

Jednotlivá epocha sestávala z padesáti kroků, během nichž byla na vstup modelu přivedena dávka, vzorek, příkladů trénovací množiny dat a vypočítána hodnota chybové funkce a dalších metrik. Při rozsahu dávky 64 příkladů, tak každá epocha využívala 3200 příkladů z trénovací množiny dat.
Každý krok epochy odpovídal jedné aktualizaci gradientů a parametrů modelu.

Validace probíhala na konci každé epochy, se shodným rozsahem dávky a počtem kroků. Čas potřebný k ohodnocení vlastností modelu na všech 3200 příkladech podstatně prodlužoval celkovou dobu učení.

Volbou menšího počtu kroků validace by bylo možné dobu učení modelu podstatně zkrátit, ale na úkor přesnosti vypočítaných metrik.

Model srovnatelných, nebo i lepších kvalit, by ovšem bylo možné získat volbou menšího počtu epoch, nebo aplikací metody předčasného zastavení učení.

Počet epoch byl ovšem zvolen s cílem demonstrovat efekt přeučení na hodnoty metrik, naměřených během validace modelu, a výhody využití adaptivních optimalizačních algoritmů k jeho potlačení.

7 Výsledky a diskuse

Pro jednotlivé navržené modely byly vytvořeny samostatné slovníky definující jejich hyperparametry a společný slovník určující počet epoch a další aspekty procesu učení.

Slovníky byly postupně použity jako argumenty konstruktoru k vytvoření příslušných instancí třídy ´TextClassifier´. Následným voláním metody ´build_model´, s výchozími hodnotami argumentů, byly modely inicializovány.

Proces učení byl zahájen, bez dalších změn instrukcí zadaných konstruktoru třídy, voláním metody ´train_model´ příslušné instance. Chování modelů na testovací množině dat bylo nakonec ověřeno metodou ´test_model´.

Mezi inicializací, učením a testováním modelů byl výpočetní graf tensorů a jim příslušných operací vždy resetován, pro odstranění již zbytečných prvků grafu a zmírnění nároků na paměť počítače.

7.1 Ohodnocení modelů

Během procesu učení byly modely ohodnoceny metrikami přesnosti, preciznosti a senzitivity, vypočítanými funkcemi frameworku TensorFlow

Hodnoty všech zvolených metrik byly vypočítány během každé epochy učení, za použití trénovací množiny dat, a po skončení epochy s příklady množiny validační.

Pro ohodnocení modelů byly především důležité hodnoty vypočítané na validační množině dat, které podávaly přesnější obraz o vlastnostech modelu při klasifikaci neznámých příkladů.

Hodnoty metrik a další vlastnosti modelů byly dále importovány do prostředí Tensorboard, které posloužilo k tvorbě grafické reprezentace vývoje metrik během procesu učení a struktury modelu. Hodnoty validačních metrik modelů, exportované jako samostatné soubory csv, jsou obsaženy v příloze D.

Kvalita modelů byla dále ověřena na testovací podmnožině dat, s využitím metody ´test_model´ náležící třídě ´TextClassifier´.
Pro všechny prvky testovací množiny dat je modelem vypočítán vektor pravděpodobnosti příslušnosti k jednotlivým kategoriím. Hodnoty prvků vektoru jsou zaokrouhleny, a výsledný binární vektor je srovnán s vektorem označení.

Prvky testovací množiny, s jednou nebo více nesprávně přiřazenými kategoriemi, jsou považovány za chybně označené.
Za chybně označený je považován i příklad, jemuž modelem žádná kategorie přiřazena nebyla. Počet chybně označených prvků testovací množiny, spolu s jejím rozsahem, je použit k výpočtu přesnosti modelu.

Chybně označené prvky testovací množiny jsou následně detokenizovány.
Binární reprezentace označených a očekávaných kategorií, spolu se slovníkem kategorií vytvořeným během předzpracování dat, je využita k vytvoření seznamů modelem přiřazených a očekávaných kategorií rizik.

```
the product poses a risk of burns because  in case it catches fire  the flames propagate too quickly  the product does not comply with the relevant european standard en 14878


Predicted: ['Fire', 'Burns'] Expected: ['Burns']


when opening the gate  a child may put his her fingers between the moving parts and trap its fingers  the product does not comply with the relevant european standard en 12227


Predicted: ['Burns'] Expected: ['Entrapment']



Final Accuracy 0.8595006347862886
```

Fig. 17 Příklad terminálového výstupu testování modelu

Text všech chybně označených prvků testovací množiny, spolu se seznamem označených a očekávaných kategorií, je vytisknut na obrazovce. Na posledním řádku výstupu je vypsána konečná přesnost modelu.

Ne všechny případy neshody mezi označenými a očekávanými kategoriemi bylo ovšem možné interpretovat jako chybu na straně modelu. Text některých záznamů, jejichž modelem přiřazené označení neodpovídalo označení očekávanému, byl významově bližší označením přiřazeným modelem. Další neshody v označení byly způsobeny použitím více, významu textu odpovídajících, označení než jenom jednoho obecnějšího.

```
the conductors in the cable are too thin and the cable can overheat which might cause a fire
in addition  if the user touches the cable  he or she may suffer burns  the product does not
comply with the requirements of the low voltage directive and the relevant european
standard en 60335

Predicted: ['Burns', 'Fire'] Expected: ['Fire']

there is the possibility of the fuel pipe touching the insulation in the engine bay  if this occurs
the fuel pipe may become damaged  which in isolated cases may cause leaks and  in extreme
cases  may cause a fire

Predicted: ['Fire'] Expected: ['Injuries']

the product poses a risk to health because the bracelets are made from seeds  the internal
contents of which contains the toxin abrin  with the outer seed coat pierced the toxin would
be available rendering them extremely toxic if accidentally consumed

Predicted: ['Chemical'] Expected: ['Health risk / other']
```

Fig. 18 Příklady vhodných ale neočekávaných označení

Je tedy otázkou, zda je zmíněná interpretace popsané neshody, označené jako chyby způsobené vlastnostmi modelu, vhodná. Za předpokladu, že záznamy tvořící původní množinu dat byly výsledkem lidské práce, a že člověk je omylný, je možné označit výše popsané případy za důsledek nevhodné volby označení tvůrcem záznamu.

Většina zaznamenaných neshod mezi předpokládaným a modelem přiřazeným označením byla ovšem evidentně způsobena nedostatečnou přesností modelu, nikoliv lidskou chybou.

7.1.1 Srovnání průběhu procesu učení modelů

Během procesu učení se vývoj hodnot chybové funkce, měřené na validační množině dat, do značné míry shodoval napříč navrženými modely. Po počátečním rychlém poklesu se hodnoty chybových funkcí, s výjimkou relativně zanedbatelných změn mezi jednotlivými kroky, relativně brzy ustálily.

Proces učení tak bylo možné přerušit již podstatně dříve než po průběhu zvolených 100 epoch, s minimálním vlivem na kvalitu modelu.

V prvních deseti, a zvláště prvních pěti, epochách prodělaly hodnoty chybových funkcí rychlý pokles. Po vykonání první epochy byly hodnoty chybových funkcí téměř shodné, a pohybovaly od 0.1773 pro model „gamma“ k 0.1854 pro model „delta“.

Na konci desáté epochy se míra chyby všech modelů snížila na polovinu, nebo méně, hodnoty naměřené v epoše první. Rozdíl mezi ohodnocením nejlepšího a nejhoršího modelu se ovšem podstatně zvětšil.

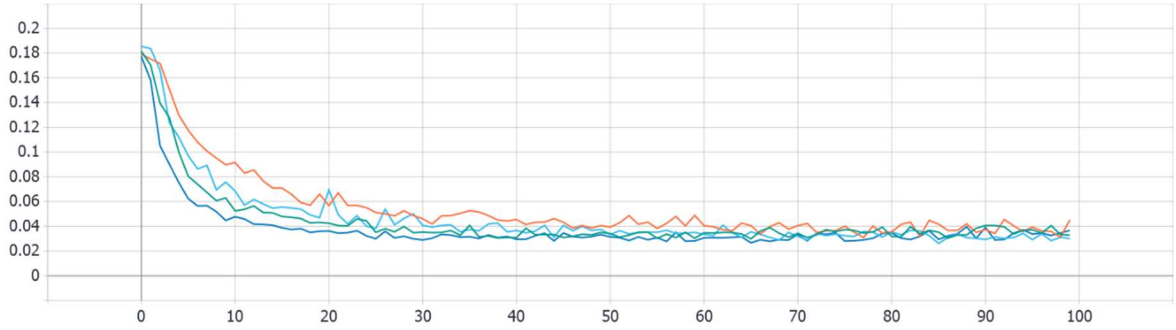


Fig. 19 Validační chyba modelů: alpha-zelená, beta-oranžová, gamma-tmavě modrá, delta-světle modrá

Zajímavé bylo především ohodnocení modelu „beta“ jehož hodnota chybové funkce, 0.09155, byla ze všech modelů nejvyšší a během prvních deseti epoch klesala nejpomaleji, a to i vzhledem k jednodušším modelům „alpha“ a „gamma“.

V průběhu epoch 11 až 40 se rozdíl v hodnotách chybových funkcí modelu „beta“ a modelů ostatních postupně zmenšoval, ale i nadále přetrvával. Současně se pokles hodnoty chybové funkce podstatně zpomalil.

Během epoch 41 až 70 se validační chyba modelů, vyjma modelu „beta“, ustálila a již nedocházelo k jejímu výraznému zlepšení. Validační chyba modelů „gamma“ a „alpha“ dosáhla svého minima po ukončení 65 epochy, s hodnotou chybové funkce 0.02668 pro model „gamma“ a 0.02988 pro model „alpha“.

Nicméně s výjimkou modelu „beta“ již chyba modelů neprocházela podstatnými změnami.

Během posledních třiceti epoch se naměřené hodnoty chybových funkcí všech modelů ustálily v rozmezí $(0.02, 0.05)$. Rozdíl mezi chybou modelu „beta“ a ostatních modelů současně téměř zmizel.

Po 77 epoše dosáhl model „beta“ minima chybové funkce s hodnotou 0.03095. Hodnota validační chyby modelu „delta“ dosáhla minimální hodnoty, 0.02616, po uplynutí 85 epoch procesu učení.

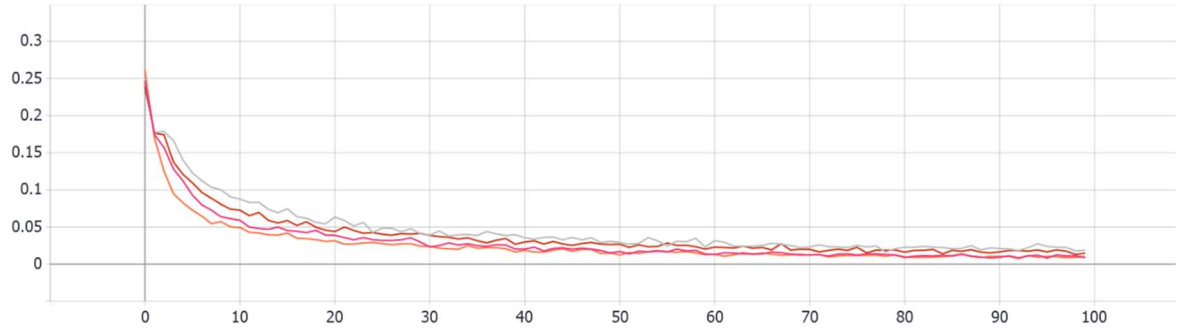


Fig. 20 Trénovací chyba modelů: alpha-magenta, beta-šedá, gamma-oranžová, delta-červená

Zatímco se validační hodnota chybové funkce navržených modelů postupně stabilizovala, hodnota chybové funkce naměřené na trénovací množině dat dále klesala až do ukončení procesu učení. Stabilizace validačních chyb, a z hodnoty chybové funkce vycházejících metrik navržených modelů tedy nebyla důsledkem podučení, nedostatečné schopnosti modelu rozpoznat vzory v příkladech trénovacích dat.

Vývoj hodnot metrik oproti tomu naznačuje dosažení nejvyšší možné míry generalizace na příslušné množině validačních dat.

Je nutné zdůraznit že minimální naměřená hodnota chybové funkce není vhodným ohodnocením kvality modelu, neboť závisí na složení dávky příkladů náhodně vybraných z validační množiny dat. Rozdílné vlastnosti příkladů, v případě řešené úlohy se může jednat o délku textu, nebo přesnost popisu, vedou k rozdílné obtížnosti jejich klasifikace.

Existuje řada technik, například zavedení vážených příkladů nebo výběr vyváženého vzorku množiny, které mohou zohlednit rozmanitost složení množin dat. Nicméně jejich využití není vždy vhodné a nemůže, vyjma velice jednoduchých úloh, zcela eliminovat výkyvy hodnot chybové funkce.

Sledovaný vývoj validační chyby modelů odpovídal očekávanému chování zvoleného optimalizačního algoritmu Adam.

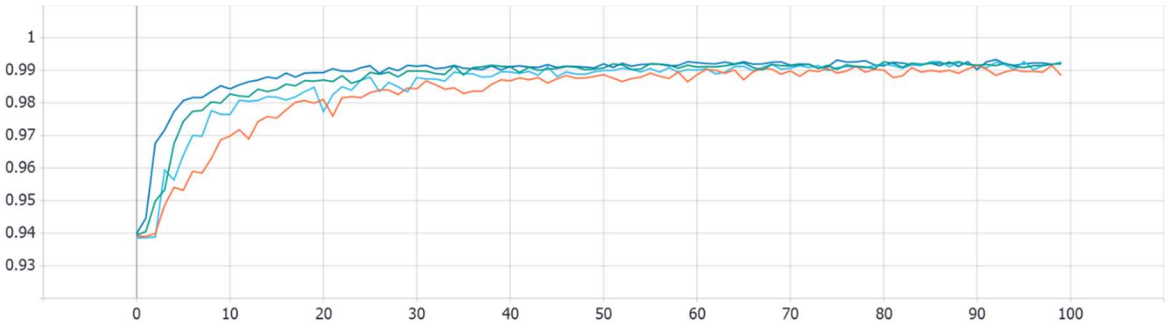


Fig. 21 Validační přesnost modelů: alpha-zelená, beta-oranžová, gamma-tmavě modrá, delta-světle modrá

Charakteristiky vývoje hodnot validační přesnosti modelů se nevyhnutelně, v podstatné míře shodovaly s vlastnostmi vývoje validačních hodnot chybových funkcí.
Validační přesnost modelů se během prvních desítek epoch rychle zvyšovala, aby se následně ustálila během 40 až 70 epochy, v rozmezí $(0.988, 0.998)$.
Nejdříve se ustálila validační přesnost modelu „gamma“. V následujících epochách dosáhly srovnatelné přesnosti ostatní modely.

Výjimkou byla validační přesnost modelu „beta“ která zůstala znatelně nižší než validační přesnost ostatních modelů až do konce procesu učení.

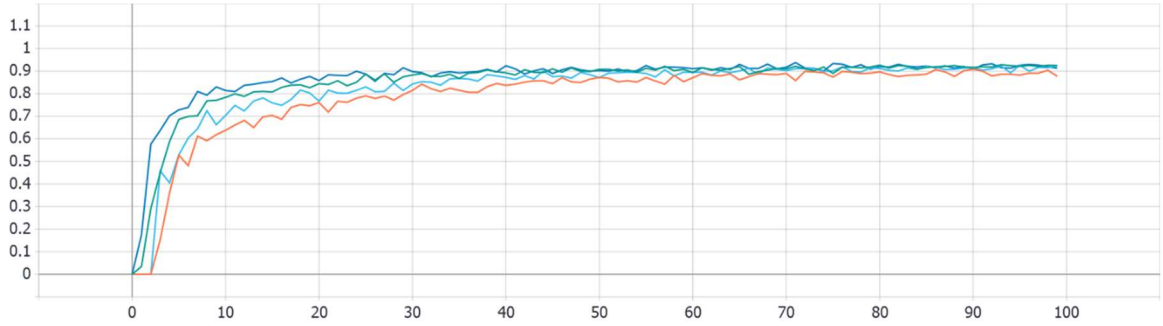


Fig. 22 Validační senzitivita *modelů: alpha-zelená, beta-oranžová, gamma-tmavě modrá, delta-světle modrá*

Vývoj validační preciznosti a senzitivity následoval srovnatelný vzor rychlého zlepšení na začátku procesu učení a následného ustálení v okolí maximálních dosažených hodnot. Stejně jako v případě přesnosti a hodnoty chybové funkce, i validační sensitivita modelu „beta“ nedosahovala hodnot naměřených pro ostatní modely. A hodnoty validačních metrik modelu „gamma“ se ustálily dříve.

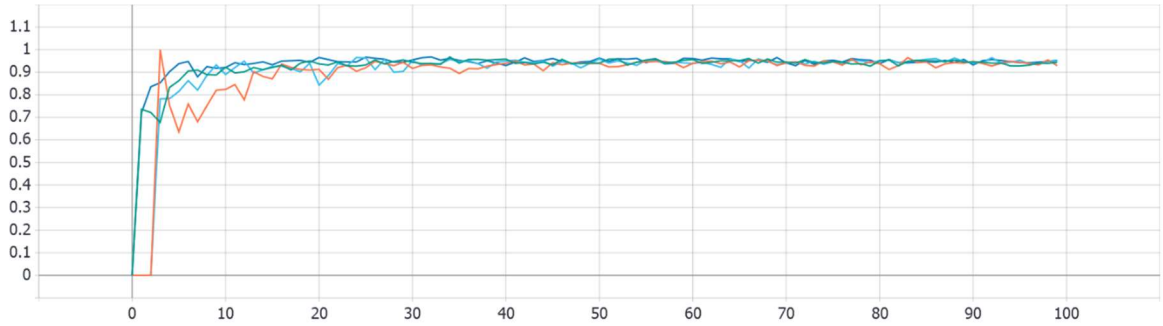


Fig. 23 Validační preciznost *modelů: alpha-zelená, beta-oranžová, gamma-tmavě modrá, delta-světle modrá*

Jisté rozdíly byly patrné v případě validační preciznosti, jejíž hodnota se ustálila dříve než hodnoty ostatních validačních metrik. Preciznost modelu „beta“ navíc během prvních tří epoch zůstávala nulová, aby na konci epochy čtvrté dosáhla hodnoty 1, znatelně vyšší hodnoty než preciznost modelů ostatních. V epoše následující ovšem klesla hodnota preciznosti modelu „beta“ na 0.75341.
Nicméně rozdíly mezi validační precizností modelu „beta“ a ostatních modelů byl méně znatelný než pro ostatní vypočítané metriky.

7.1.2 Srovnání modelů po zakončení procesu učení

Všechny navržené modely dosahovaly srovnatelných výsledků na validační, i testovací, množině dat, nehledě na počet a rozsah skrytých vrstev a zvolené hodnoty ostatních hyperparametrů. Průměrná validační přesnost modelu „alpha“ během posledních deseti epoch, 0.99156, se téměř rovnala průměrné validační přesnosti modelu „delta“ za stejné období, 0.99151, ačkoliv byl model „alpha“ podstatně jednodušší a čas vyžadovaný jednou epochou jeho procesu učení byl podstatně kratší.

Model	Přesnost	Precision	Recall	Čas epochy (v sekundách)
Alpha	0.991566616	0.93790431	0.922419083	1.2097
Beta	0.989860296	0.939806938	0.890519893	2.3779
Gamma	0.991994286	0.946539152	0.920879513	1.2585
Delta	0.991519827	0.946070564	0.91262688	2.4564

Tabulka 12 Srovnání průměru validačních metrik navržených modelů během posledních deseti epoch

Přesnost modelů na příkladech testovací množiny byla podobně vyrovnaná, ačkoliv pro všechny modely byla nižší než přesnost validační. Nejlepších výsledků dosahoval model „delta“, který nevhodně označil 591 příkladů z testovací množiny. Nicméně i nejhůře hodnocený model, „beta“, chybně označil o pouhých 157 více příkladů.

Model	Přesnost na testovací množině
Alpha	0.87715
Beta	0.85642
Gamma	0.88311
Delta	0.88656

Tabulka 13 Srovnání přesnosti modelů na testovací množině

Jediným kritériem, podle kterého bylo možné rozlišit jednotlivé modely, byl čas nutný k dokončení jedné epochy, a tedy i procesu učení. Ten byl, v případě rozsáhlejších modelů „beta“ a „delta“, téměř dvojnásobný oproti modelům jednodušším.

Nejlepší hodnota testovací přesnosti, rychlá stabilizace hodnot validačních metrik, spolu s druhým nejkratším časem epochy, určovaly model „gamma“ jako nejvhodnější k řešení zadané úlohy. Nicméně srovnatelných výsledků, s méně parametry a nižšími nároky na výpočetní prostředky dosahoval i model „alpha“. V obou případech se jednalo, co do počtu vrstev a parametrů, o méně rozsáhlé z navržených modelů.

Nejnižší testovací přesnost vykazoval model „beta“, který byl co do počtu parametrů druhý nejrozsáhlejší z navržených modelů. Delší čas jedné epochy, spolu s pomalejším ustálením hodnot validačních metrik, navíc naznačovaly, že ani předčasné zastavení procesu učení by nevedlo k vytvoření podstatně lepšího modelu.

Srovnání modelů tak do značné míry potvrdilo široce rozšířenou poučku, že rozsah modelu není určujícím faktorem jeho kvality.

Ačkoliv se zdá že existuje korelace mezi naměřeným hodnotami testovací přesnosti, počtem vrstev modelu a rozsahu použitého embeddingu, malý počet modelů, a tedy pozorování, neumožňuje provést korelační analýzu. Nicméně naměřené hodnoty odpovídají poznatkům o vlivu rozměru embeddingu na kvalitu modelů. (Patel & Bhattacharyya, 2017)

7.1.3 **Ukázka výstupu modelu**

Výstupem navržených modelů byl objekt třídy ndarray sestávající z prvků třídy float32, odpovídající vektoru pravděpodobností jednotlivých označení.
(The SciPy community, 2020)

```
[0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

Fig. 24 Příklad výstupu modelu přiřazující pravděpodobnost 1.0 označení s indexem 3

Výstup modelu je možné snadno upravit, v závislosti na okolnostech užití modelu a případných požadavcích zákazníka, součinem hodnot vektoru pravděpodobností vektorem vah jednotlivých kategorií. Váhy mohou být zvoleny s ohledem na závažnost rizika, empiricky zjištěnou pravděpodobnost jeho vzniku, nebo obecnost příslušné kategorie.

Stejně tak i zobrazení výstupu modelu vychází ze specifických okolností jeho nasazení. Příkladem může být forma výstupu metody ´test_model´ která podle hodnot vektoru pravděpodobností přímo určuje označení textů.

Alternativou může být využití modelu jako základu pro DSS, a zobrazování samotných pravděpodobností, nebo podle pravděpodobností seřazeného seznamu kategorií, jako doporučení pro uživatele systému.

7.2 Nedostatky řešení

Podstatnou nevýhodou navržených modelů byla již zmíněná omezení kladená na délku klasifikovaného textu a rozsah slovníku. Případné nasazení modelů v praxi by vyžadovalo zajištění odpovídajícího rozsahu a formátu vstupu modelu.

Nejedná se ovšem o problém neřešitelný. Základem pro úpravu vstupu modelu pro použití v praxi, mohou být některé postupy použité k předzpracování původní množiny dat, tokenizace a dorovnávání délky textu.

Vstup nevhodné délky je možné snadno zkrátit na požadovaný rozsah. Výhodami zkrácení vstupu je kompatibilita se stávajícím modelem, a schopnost nově vytvořeného systému klasifikovat text libovolné délky.

Alternativní řešení, nastavení hyperparametru délky vstupu embeddingu na podstatně vyšší hodnotu, by vyžadovalo vytvoření nového, co do počtu parametrů rozsáhlejšího, modelu, s upravenou hodnotou příslušného hyperparametru. Výsledný model by navíc stále nebyl použitelný ke klasifikaci textu libovolné délky.

Text, obsahující slova bez příslušného vektoru embeddingu, je možně klasifikovat pouze po odpovídající úpravě. Slova, neobsažená ve stávajícím slovníku, je možné nahradit předem určeným tokenem, podobný tokenu paddingu již použitého k dorovnání délky textu záznamů původní množiny dat.

Zavedení tokenu, reprezentujícího všechna neznámá slova, by vyžadovalo vytvoření nového modelu s větším počtem parametrů. Nicméně nový model by byl schopen přijímat jako vstup tokenizovaný text, nehledě na přítomnost neznámých slov.

V principu je možné ponechat stávající model beze změn a nahradit neznámá slova již zavedeným tokenem, například již uvedeným tokenem paddingu.
Nicméně použití stejného tokenu, k zastoupení dvou nebo více významem odlišných slov, vyústí k přiřazení jediného, shodného vektoru embeddingu, a znemožní jejich rozlišení.
Přesnost, a další charakteristiky výsledného modelu, by tak mohly být negativně ovlivněny.

Dalším nedostatkem navržených modelů, především v případě modelů „beta“ a „gamma“, byl nadbytečný počet hustě zapojených vrstev. Rozšíření počtu hustě zapojených vrstev nevedlo ke zlepšení vlastností modelů, a v případě modelu „beta“ dokonce způsobilo měřitelné zhoršení přesnosti ve srovnání s neupraveným modelem „alpha“.

Zahrnutí nadbytečných vrstev do navržených modelů dále podstatně navýšilo počet parametrů podléhajících změnám během procesu učení. V konečném důsledku rozšíření modelu o další hustě zapojené vrstvy vedlo k podstatnému zvýšení výpočetní náročnosti procesu učení modelu. Čas nutný k vykonání jednoho kroku byl, oproti neupraveným modelům, téměř zdvojnásoben. Navýšení počtu hustě zapojených vrstev jako cesty ke zlepšení kvality modelu tedy, přinejmenším z hlediska efektivity, není možné doporučit.

Jistého zlepšení kvality modelu by ovšem bylo možné dosáhnout zavedením dodatečných konvolučních vrstev, popřípadě úpravou vlastností vrstev stávajících.
Mezi nejjednodušší alternativní nastavení konvolučních vrstev patří změna počtu konvolučních jader, rozměru okna konvoluce nebo postupu aplikace paddingu pro zachování rozměru mapy výstupních příznaků.

Další cestou ke zlepšení vlastností navržených modelů může být zavedení regularizace hodnot postsynaptického potenciálu a aktivačních funkcí hustě zapojených vrstev.

V neposlední řadě je nutné zmínit možnost využití jedné nebo více rekurentních vrstev, a vytvoření modelu kombinujícího vlastnosti RNN a CNN. Konvoluční vrstvu je možné pojmout jako prostředek k tvorbě příznaků pro zpracování rekurentními vrstvami.

Nejednalo by se přitom o nijak radikální změnu stávajícího postupu. Díky principům tensorových operací je s použitím knihovny Keras relativně jednoduché vytvořit model sestávající z mnoha různorodých prvků, včetně kombinací konvolučních a rekurentních vrstev, popřípadě jejich variant.

Modely využívajících rekurentních a konvolučních vrstev byly navíc již úspěšně použity k řešení úloh zpracování sekvenčních i dvourozměrných dat. (Vinyals, Toshev, Bengio, & Erhan, 2016)

8 Závěr

Úloha klasifikace textů popisů rizik databáze Safety Gate do více tříd s více označeními se ukázala jako řešitelná i relativně jednoduchou konvoluční sítí a s využitím omezených výpočetních prostředků.
Všechny navržené modely dosahovaly na záznamech množiny testovacích dat přesnosti přesahující 0.8. Vzhledem k různorodé kvalitě záznamů databáze, a ne zcela přímočarého postupu používaného tvůrci záznamů během volby označení, se uvedená přesnost dala považovat za přípustnou.

Malý počet navržených modelů znemožňoval provedení plnohodnotné korelační analýzy hodnot volných hyperparametrů a naměřených metrik. Nicméně zjištěné vlastnosti modelů jsou v souhlasu s literaturou.

Vliv počtu a rozsahu hustě zapojených vrstev na přesnost modelu se ukázal jako sporný. Výsledky rozsáhlejších modelů byly srovnatelné, a v případě modelu „beta“ dokonce překonané, výsledky modelů jednodušších. Jediným, znatelným důsledkem vyššího počtu vrstev modelu, byl delší čas nutný k dokončení jedné epochy procesu učení.

Vhodnost algoritmu Adam, k optimalizaci hodnot parametrů modelů řešících srovnatelné klasifikační úlohy, byla potvrzena rychlým poklesem a následnou stabilitou hodnot chybové funkce a dalších metrik během procesu učení. Další výhodu využití adaptivního algoritmu byla jednodušší volba hyperparametrů modelů, především rychlosti učení η.

Vývoj validačních metrik modelů během procesu učení vykazoval důležité vlastnosti.
Nehledě na nepřiměřeně vysoký počet epoch nezačaly hodnoty validačních metrik navržených modelů vykazovat známky přeučení. Přičemž ani jeden z navržených modelů nevyužíval regularizace, dropoutu, ani dalších technik využívaných k jeho potlačení.

Je možné, že uvedené charakteristiky vývoje metrik byly důsledkem vlastností použitého optimalizačního algoritmu. Nicméně přesné určení důvodů nepřítomnosti přeučení přesahuje rozsah práce.

Druhotným přínosem práce bylo objevení nedostatků systému Safety Gate.
Zvláště jeho webového rozhraní, ale i postupů zadávání dat a jejich následné správy, včetně metody exportu záznamů databáze a formátu exportovaných dat.

Volba nestandartního formátu exportu záznamů, jmenovitě zlomku HTML dokumentu, se dá jen těžko vysvětlit jako implementace požadavků zadavatele. Možnost, že se jednalo o důsledek závady, vzniklé během vývoje nebo následného provozu systému, se ovšem zdá poměrně nepravděpodobná, vzhledem k relativně dlouhé době provozu systému Safety Gate.

Dotaz na stávající neuspokojivý stav systému, se seznamem nejvýznamnějších nedostatků, byl vznesen pomocí oficiálního kontaktního formuláře Evropské komise.

9 Seznam použitých zdrojů

Alphabet Inc. (2019). *TensorFlow*. Získáno 1. 7 2019, z https://www.tensorflow.org

Atwood, J. (16. 2 2005). *Regex use vs. Regex abuse.* Získáno 1. 7 2019, z https://blog.codinghorror.com/regex-use-vs-regex-abuse/

Atwood, J. (4. 12 2008). *Podcast #32.* Získáno 1. 7 2019, z https://stackoverflow.blog/2008/12/04/podcast-32/

Bot, A. (3. 4 2015). *[FAQ] AutoTLDR Bot.* Získáno 1. 7 2019, z https://www.reddit.com/r/autotldr/comments/31b9fm/faq_autotldr_bot/

Boyd, S., & Vandenberghe, L. (2004). Unconstrained minimization. V S. Boyd, & L. Vandenberghe, *Convex Optimization* (stránky 455-514). Cambridge, UK: Cambridge University Press.

Centrum zpracování přirozeného jazyka. (1. 1 2020). *Český stoplist.* Načteno z Centrum zpracování přirozeného jazyka: https://nlp.fi.muni.cz/cs/StopList

Český národní korpus. (4. 3 2019). *Lemma.* Získáno 1. 7 2019, z http://wiki.korpus.cz/doku.php/pojmy:lemma

Chollet, F. (2019). *Deep learning v jazyku Python* . Praha: Grada Publishing.

Clune, J., Lehman, J., & Misevic, D. (2018). The Surprising Creativity of Digital Evolution: A Collection of Anecdotes. Arxiv. Načteno z https://arxiv.org/pdf/1803.03453.pdf

Davydova, O. (15. 10 2018). *Text Preprocessing in Python: Steps, Tools, and Examples - The Comparative Table.* Získáno 1. 7 2019, z https://docs.google.com/spreadsheets/d/1-9rMhfcmxFv2V2Q5ZWn1FfLDZZYsuwb1eoSp9CiEEOg/

Denker, J. S., Gardner, W. R., Graf, H. P., Henderson, D., Howard, R. E., Hubbard, W., . . . Guyon, I. (1989). Neural Network Recognizer for Hand-Written Zip Code Digits. *Advances in Neural Information Processing Systems 1*, stránky 323-331.

Dey, R., & Fathi, M. S. (2017). Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks. *IEEE 60th International Midwest Symposium on Circuits and Systems*, (stránky 1597-1600).

Duchi, J., Hazan, E., & Singe, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, stránky 2121-2159.

Duda O., R., Hart, P., & David, S. (2001). MULTILAYER NEURAL NETWORKS. V R. Duda O., P. Hart, & S. David, *Pattern Classification* (stránky 282-350). New York: Wiley.

Edwards, L., & Veale, M. (2017). Slave to the Algorithm? Why a 'Right to an Explanation' Is Probably Not the Remedy You Are Looking For. *16 Duke Law & Technology Review 18*.

EUROPEAN COMMISSION. (9. 3 2018). *Product Safety Business Alert Gateway User manual for producers and distributors* . Načteno z Safety Gate: the rapid alert system for dangerous non-food products: https://ec.europa.eu/consumers/consumers_safety/safety_products/rapex/alerts/repository/content/pages/rapex/docs/User%20Manual.pdf

EUROPEAN COMMISSION. (15. 3 2019). Commission Implementing Decision (EU) 2019/417 of 8 November 2018 laying down guidelines for the management of the European Union Rapid Information System ‘RAPEX’ established under Article 12

of Directive 2001/95/EC on general product safety and its notif. *Official Journal of the European Union*, stránky 73-121.
European Parliament, Council of the European Union. (3. 12 2001). *EUR-Lex.* Načteno z Directive 2001/95/EC of the European Parliament and of the Council of 3 December 2001 on general product safety (Text with EEA relevance): https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:32001L0095
Fisher, F. R. (9 1936). THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS. *Annals of Eugenics*, stránky 179-188.
Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, stránky 193-202.
Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. *International Conference on Artificial Intelligence and Statistics* , (stránky 315-323). Fort Lauderdale, FL, USA.
Google DeepMind. (2019). *AlphaGo DeepMind*. Získáno 1. 7 2019, z https://deepmind.com/research/case-studies/alphago-the-story-so-far
Google DeepMind. (24. 1 2019). *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II.* Získáno 1. 7 2019, z https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii
Heidenreich, H. (21. 12 2018). *Stemming? Lemmatization? What?* Získáno 1. 7 2019, z https://towardsdatascience.com/stemming-lemmatization-what-ba782b7c0bd8
Hinton, G. (2014). *Overview of mini-batch gradient descent* . Načteno z Coursera: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
Hochreiter, S. (19. 6 1991). Untersuchungen zu dynamischen neuronalen Netzen.
House of Representatives. (12. 6 2019). H.R.3230 - Defending Each and Every Person from False Appearances by Keeping Exploitation Subject to Accountability Act of 2019. Washington, DC., USA.
Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., . . . Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *Arxiv*.
Jin, J., Dundar, A., & Culurciello, E. (2014). Flattened Convolutional Neural Networks for Feedforward Acceleration.
Kandalgaonkar, N. (1. 6 2000). *Abigail's regex to test for prime numbers.* Získáno 1. 7 2019, z http://neilk.net/blog/2000/06/01/abigails-regex-to-test-for-prime-numbers/
Kelbel, J., & Šilhán, D. (30. 5 2002). *Shluková analýza.* Získáno 1. 7 2019, z http://cmp.felk.cvut.cz/cmp/courses/recognition/zapis_prednasky/zapis_02/13/shlukovani.pdf
Keras Team. (1. 1 2019). *Embedding layers.* Získáno 1. 7 2019, z https://keras.io/layers/embeddings/
Keras Team. (29. 1 2020). *Docs » Activations.* Načteno z Keras Documentation: https://keras.io/activations/
Keras Team. (1. 1 2020). *The Sequential model API.* Načteno z Keras Documentation: https://keras.io/models/sequential/
Kingma, D. P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. *International Conference for Learning Representations.* San Diego, US.
Knight, W. (23. 11 2016). *How to Fix Silicon Valley's Sexist Algorithms.* Získáno 1. 7 2019, z MIT Technology Review:

https://www.technologyreview.com/s/602950/how-to-fix-silicon-valleys-sexist-algorithms/
Knight, W. (12. 7 2017). *Biased Algorithms Are Everywhere, and No One Seems to Care.* Získáno 1. 7 2019, z MIT Techonology Review: https://www.technologyreview.com/s/608248/biased-algorithms-are-everywhere-and-no-one-seems-to-care/
Know Your Meme. (1. 1 2020). *Tay AI.* Načteno z Know Your Meme: https://knowyourmeme.com/memes/sites/tay-ai
Křen, M. (8. 6 2017). *Příručka ČNK.* Získáno 1. 7 2019, z Korpus: https://wiki.korpus.cz/doku.php/pojmy:korpus
Křen, M. (20. 12 2018). *Korpus SYN verze 7.* Získáno 1. 7 2019, z Příručka ČNK: http://wiki.korpus.cz/doku.php/cnk:syn:verze7
Lee, P. (25. 3 2016). *Learning from Tay's introduction.* Načteno z Official Microsoft Blog: https://blogs.microsoft.com/blog/2016/03/25/learning-tays-introduction/#sm.00000gjdpwwcfcus11t6oo6dw79gw
Lipton, Z. C. (10. 6 2016). *The Mythos of Model Interpretability.* Načteno z arXiv.org: https://arxiv.org/abs/1606.03490
Margossian, C. C. (12. 5 2019). A Review of automatic differentiation and its efficient implementation. *WIREs Data Mining and Knowledge Discovery*.
Mezher, K., & Omar, N. (31. 12 2015). A backpropagation neural network to improve arabic stemming. *ournal of Theoretical and Applied Information Technology*, stránky 385-394.
Mikolov, T., Chen, K., Corrado, G., & Dean, J. (7. 9 2013). *Efficient Estimation of Word Representations in Vector Space.* Získáno 1. 7 2019, z https://arxiv.org/abs/1301.3781
Minky, M., & Seymour, P. (1969). *Perceptrons: an introduction to computational geometry.* Cambridge, MA, USA: The MIT Press.
Mosley, T. (2. 8 2019). *Perfect Deepfake Tech Could Arrive Sooner Than Expected.* Načteno z WBUR: https://www.wbur.org/hereandnow/2019/10/02/deepfake-technology
Neidinger, R. D. (5. 8 2010). Introduction to Automatic Differentiation and MATLAB Object-Oriented Programming. *SIAM REVIEW*, stránky 545-563.
NLTK Project. (4. 7 2019). *nltk.tokenize package.* Získáno 1. 7 2019, z https://www.nltk.org/api/nltk.tokenize.html
Novovičová, J., Pudil, P., & Somol, P. (2013). Moderní metody výběru příznaků ve statistickém rozpoznávání. V A. Baďura (Editor), *Umělá inteligence (6)* (stránky 425-468). Praha: Academia.
NumPy developers. (2019). *NumPy*. Získáno 1. 7 2019, z https://numpy.org/
NVIDIA Corporation. (1. 1 2020). *GeForce GTX 1070 Specifications.* Načteno z NVIDIA: https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-1070/specifications
Office of Technology Assessment. (6 1984). *Review of Postal Automation Strategy: A Technical and Decision Analysis.* Washington, DC.: U.S. Congress.
Pandas developers. (2019). *Pandas 0.25.1 documentation*. Získáno 1. 7 2019, z https://pandas.pydata.org/pandas-docs/version/0.25/
Pandas developers. (1. 1 2020). *pandas.DataFrame.* Načteno z pandas: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html#pandas.DataFrame

Pandas developers. (1. 1 2020). *pandas.DataFrame.sample.* Načteno z pandas: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sample.html

Patel, K., & Bhattacharyya, P. (2017). Towards Lower Bounds on Number of Dimensions for Word Embeddings. *International Joint Conference on Natural Language Processing.* Taipei, Taiwan.

Pennington, J., Socher, R., & Manning, C. D. (10 2014). Glove: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (stránky 1532–1543). Doha, Katar: Association for Computational Linguistics.

Pohl, J., Jirsík, V., & Honzík, P. (2 2014). Optimalizační algoritmys pravděpodobnostním směrovým vektorem a jejich srovnání s principiálně podobnými metodami. *Elektrorevue*, stránky 17-24.

Porter, M. (1. 10 2001). *Snowball: A language for stemming algorithms.* Získáno 1. 7 2019, z http://snowball.tartarus.org/texts/introduction.html

Porter, M. (1. 1 2006). *The Porter Stemming Algorithm.* Získáno 1. 7 2019, z https://tartarus.org/martin/PorterStemmer/

Psutka, J., Müller, L., Matoušek, J., & Radová, V. (2006). 8.2.3 Dokončovací fáze. V *Mluvíme s počítačem česky* (str. 482). Praha: Academia.

Python Software Foundation. (2019). *The Python Language Reference*. Získáno 1. 7 2019, z https://docs.python.org/3.6/reference/

ranks.nl. (1. 1 2019). *Czech Stopwords.* Získáno 1. 7 2019, z https://www.ranks.nl/stopwords/czech

Robbins, H., & Monro, S. (9 1951). A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, stránky 400-407.

Rosenblatt, F. (1 1957). The Perceptron—a perceiving and recognizing automaton.

Ruder, S. (19. 1 2016). *An overview of gradient descent optimization algorithms.* Načteno z Sebastian Ruder: https://ruder.io/optimizing-gradient-descent/index.html

SciPy developers. (2019). *SciPy.org*. Získáno 1. 7 2019, z https://www.scipy.org/

Sinders, C. (24. 3 2016). *Microsoft's Tay is an Example of Bad Design.* Načteno z Medium: https://medium.com/@carolinesinders/microsoft-s-tay-is-an-example-of-bad-design-d4e65bb2569f

Škrabal, M. (20. 12 2018). *Jaké korpusy zpřístupňuje Český národní korpus?* Získáno 1. 7 2019, z Příručka ČNK: http://wiki.korpus.cz/doku.php/cnk:uvod

Song, M., Montanari, A., & Nguyen, P.-M. (27. 7 2018). A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences of the United States of Americ*.

Surjanovic, S., & Bingham, D. (15. 1 2020). *EGGHOLDER FUNCTION.* Načteno z Virtual Library of Simulation Experiments: https://www.sfu.ca/~ssurjano/egg.html

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2.. vyd.). Cambridge, MA, USA: MIT Press.

TensorFlow community. (1. 1 2020). *Tensorflow.* Načteno z tf.data.Dataset: https://www.tensorflow.org/api_docs/python/tf/data/Dataset

TensorFlow community. (1. 1 2020). *tf.keras.callbacks.TensorBoard.* Načteno z TensorFlow: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/TensorBoard

TensorFlow community. (1. 1 2020). *tf.keras.layers.Conv1D.* Načteno z TensorFLow: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv1D

TensorFlow community. (1. 1 2020). *tf.keras.layers.Embedding.* Načteno z TensorFlow: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding
TensorFlow community. (1. 1 2020). *tf.keras.layers.Flatten.* Načteno z TensorFlow: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Flatten?hl=en
TensorFlow community. (1. 1 2020). *tf.keras.optimizers.Adam.* Načteno z Tensorflow API Documentation: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam
TensorFlow community. (1. 1 2020). *tf.keras.optimizers.RMSprop.* Načteno z TensorFlow: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/RMSprop
TensorFlow community. (1. 1 2020). *tf.pad.* Načteno z TensorFlow API Documentation: https://www.tensorflow.org/api_docs/python/tf/pad
The SciPy community. (1. 1 2020). *Data types.* Načteno z SciPy.org: https://docs.scipy.org/doc/numpy/user/basics.types.html
The SciPy community. (1. 1 2020). *The N-dimensional array (ndarray).* Načteno z NumPy v1.17 Manual: https://docs.scipy.org/doc/numpy/reference/arrays.ndarray.html
Ústav formální a aplikované lingvistiky. (1. 1 2019). *NLP Frameworks.* Získáno 1. 7 2019, z http://ufal.mff.cuni.cz/~zabokrtsky/courses/npfl092/html/nlp-frameworks.html
Ústav formální a aplikované lingvistiky. (2019). *Prague Dependency Treebank 3.0*. Získáno 1. 7 2019, z https://ufal.mff.cuni.cz/pdt3.0
Vallantin, W. L. (22. 1 2019). *medium.com.* Získáno 1. 7 2019, z https://medium.com/@wilamelima/why-is-removing-stop-words-not-always-a-good-idea-c8d35bd77214
Veselý, d. I. (2012). Model neuronu. V *Metody umělé inteligence* (stránky 9-10). Praha: Česká zemědělská univerzita v Praze.
Voorhees, E., & NIST. (12. 1 2001). *Message Understanding Conference Proceedings MUC-7 Table of Contents.* Získáno 1. 7 2019, z https://www-nlpir.nist.gov/related_projects/muc/proceedings/muc_7_toc.html#named
W3C. (20. 1 2020). *The global structure of an HTML document.* Načteno z W3C: https://www.w3.org/TR/html401/struct/global.html
w3sDesign. (1. 1 2018). *Facade.* Načteno z GoF Design Patterns Reference: http://w3sdesign.com/?gr=s05&ugr=struct#gf
Yin, W., Kann, K., Yu, M., & Schütze, H. (2017). Comparative Study of CNN and RNN for Natural Language Processing. *ArXiv*.
Zananir, E. (2008). *ezGestures*. Získáno 1. 7 2019, z http://www.silentlycrashing.net/ezgestures/
Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. *ArXiv*.

10 Přílohy

10.1 Příloha A: Ukázkový seznam českých „stopslov“

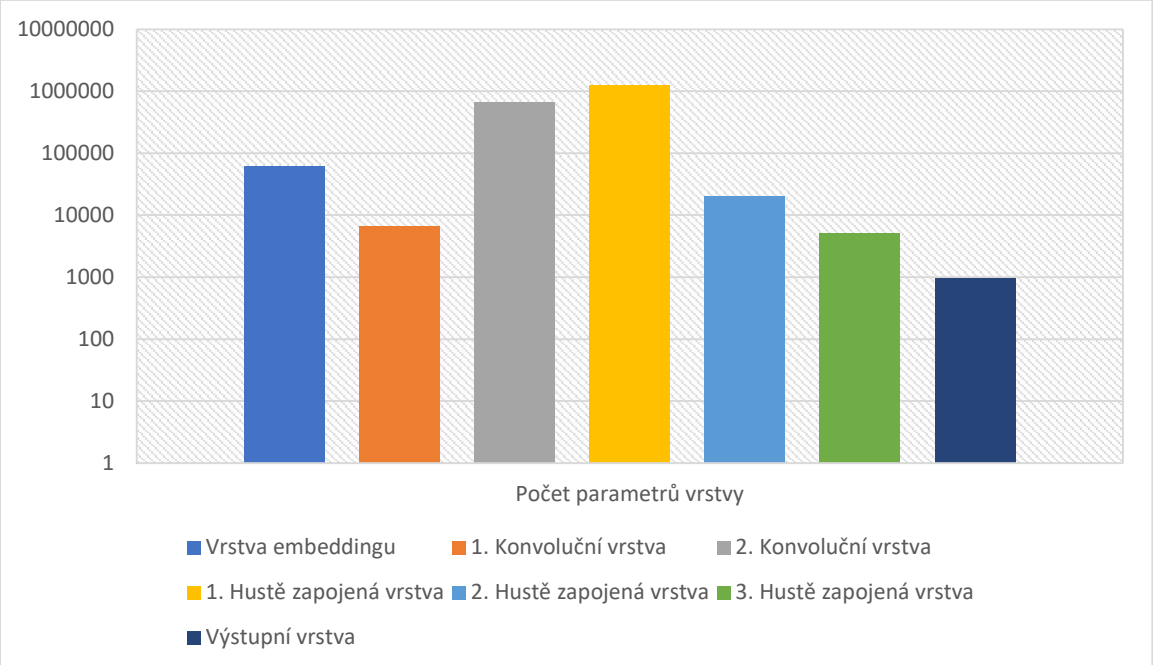
SLOVO		
a	jsou	jsme
v	které	není
se	který	jejich
na	jeho	ještě
je	však	ani
že	bude	mezi
o	nebo	byla
s	už	své
z	jen	roku
do	byl	již
i	jak	pak
to	u	první
k	co	roce
ve	při	kteří
pro	až	další
za	aby	proti
by	má	let
ale	když	tím
si	než	může
po	ze	korun
jako	která	řekl
podle	před	tom
od	být	kde
jsem	také	či
tak	bylo	tedy
pouze		

(Centrum zpracování přirozeného jazyka, 2020)

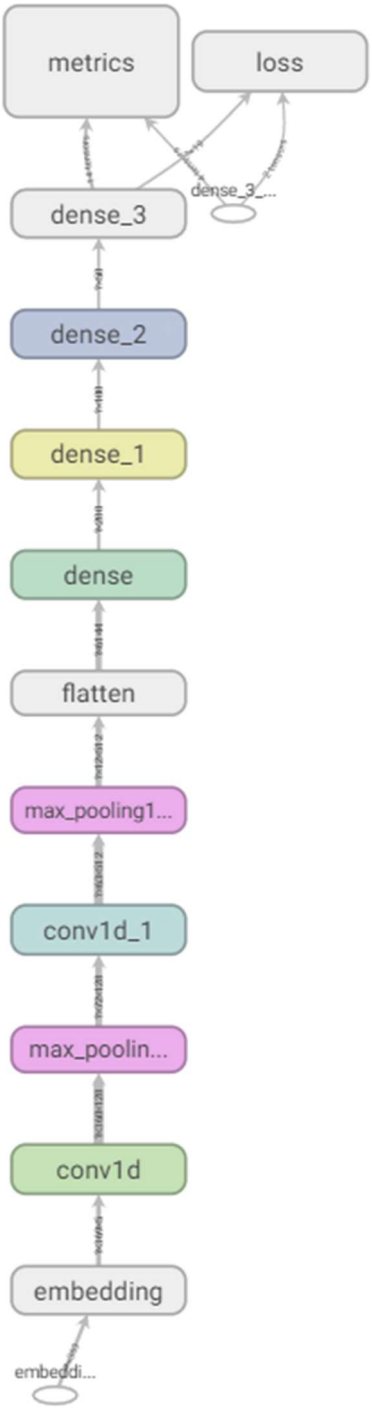
10.2 Příloha B: Parametry použité k předzpracování dat

Parametr	Hodnota
training_data_path	'./data/rapexdata.csv'
testing_data_path	None
feature_columns	['risk_desc']
label_column	'risk_type'
categorical_labels	True
main_feature_column	None
max_feature_len	0
keep_composite_labels	True
strict_sanitization	True
tokenize_words	True
matrix_form	True
training_noise_intensity	0.0
merge_input_cols	True
drop_columns	[]
categorical_weights	None
to_tensors	True

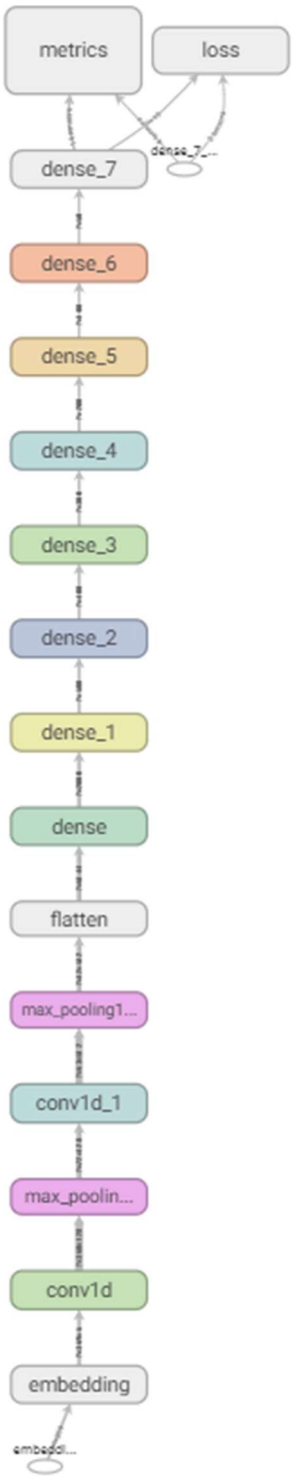
10.3 Příloha C: počet parametrů jednotlivých vrstev modelu „alpha“



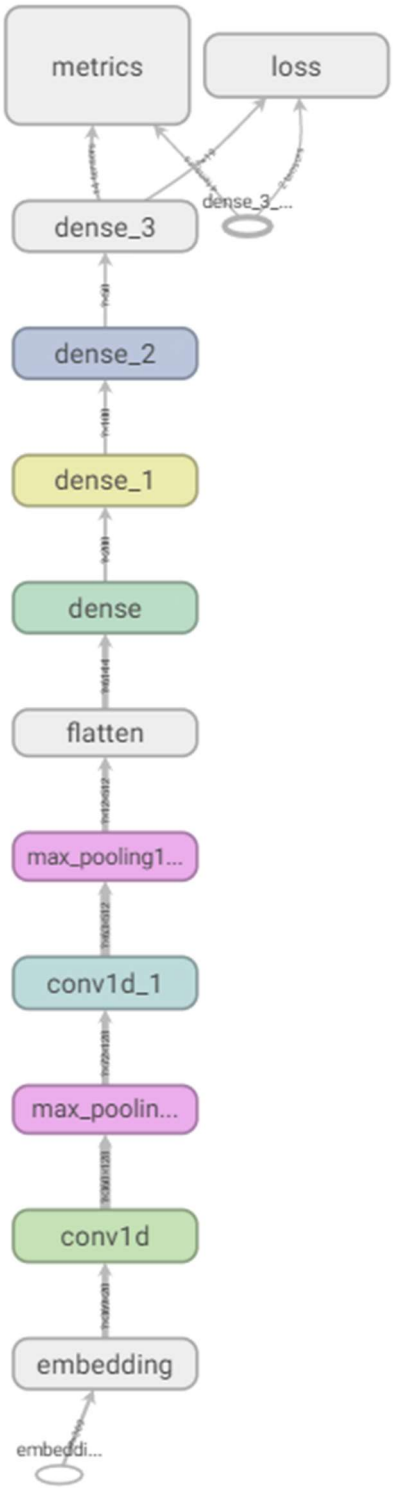
10.4 **Příloha D: Vizualizace navržených modelů**



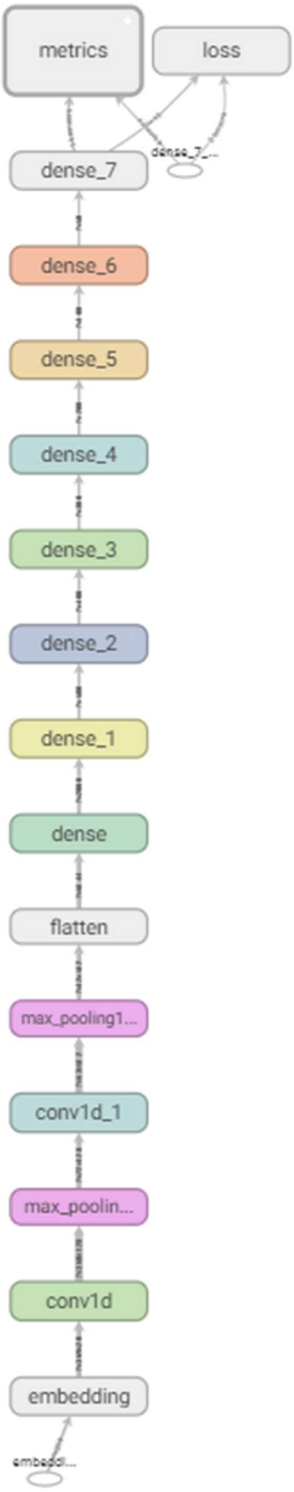
1 Model "Alpha"


metrics
loss
dense_7
dense_6
dense_5
dense_4
dense_3
dense_2
dense_1
dense
flatten
max_pooling1...
conv1d_1
max_poolin...
conv1d
embedding

2 Model "Beta"



3 Model "Gamma"



4 Model "Delta"

10.5 **Příloha E: Seznam souborů naměřených hodnot validačních metrik**

Název souboru	Metrika
run-alpha_validation-tag-epoch_accuracy.csv	Přesnost
run-alpha_validation-tag-epoch_loss.csv	Hodnota chybové funkce
run-alpha_validation-tag-epoch_precision.csv	Preciznost
run-alpha_validation-tag-epoch_recall.csv	Senzitivita
run-beta_validation-tag-epoch_accuracy.csv	Přesnost
run-beta_validation-tag-epoch_loss.csv	Hodnota chybové funkce
run-beta_validation-tag-epoch_precision.csv	Preciznost
run-beta_validation-tag-epoch_recall.csv	Senzitivita
run-gamma_validation-tag-epoch_accuracy.csv	Přesnost
run-gamma_validation-tag-epoch_loss.csv	Hodnota chybové funkce
run-gamma_validation-tag-epoch_precision.csv	Preciznost
run-gamma_validation-tag-epoch_recall.csv	Senzitivita
run-delta_validation-tag-epoch_accuracy.csv	Přesnost
run-delta_validation-tag-epoch_loss.csv	Hodnota chybové funkce
run-delta_validation-tag-epoch_precision.csv	Preciznost
run-delta_validation-tag-epoch_recall.csv	Senzitivita

10.6 **Příloha F: Seznam dodatečných souborů příloh**

Název a cesta souboru	Popis
/code/HypothesisSpace.m	Program pro tvorbu vizualizace v prostředí Octave
/tables/konvoluceDemo.xlsx	Demonstrace konvoluční operace sdružování
/tables/rapex_analysis.xlsx	Kontingenční tabulka vytvořená k průzkumové analýze databáze Safety Gate
/code/IncidentalAccident/	Adresář projektu Visual Studio 2019 příslušející programu tvorby a správy modelů
/code/IncidentalAccident/IncidentalAccident/IncidentalAccident.py	Skript v jazyce python spravující spuštění a analýzu navržených modelů
/code/IncidentalAccident/IncidentalAccident/models	Záznamy o vytvořených modelech pro další zpracování programem TensorBoard

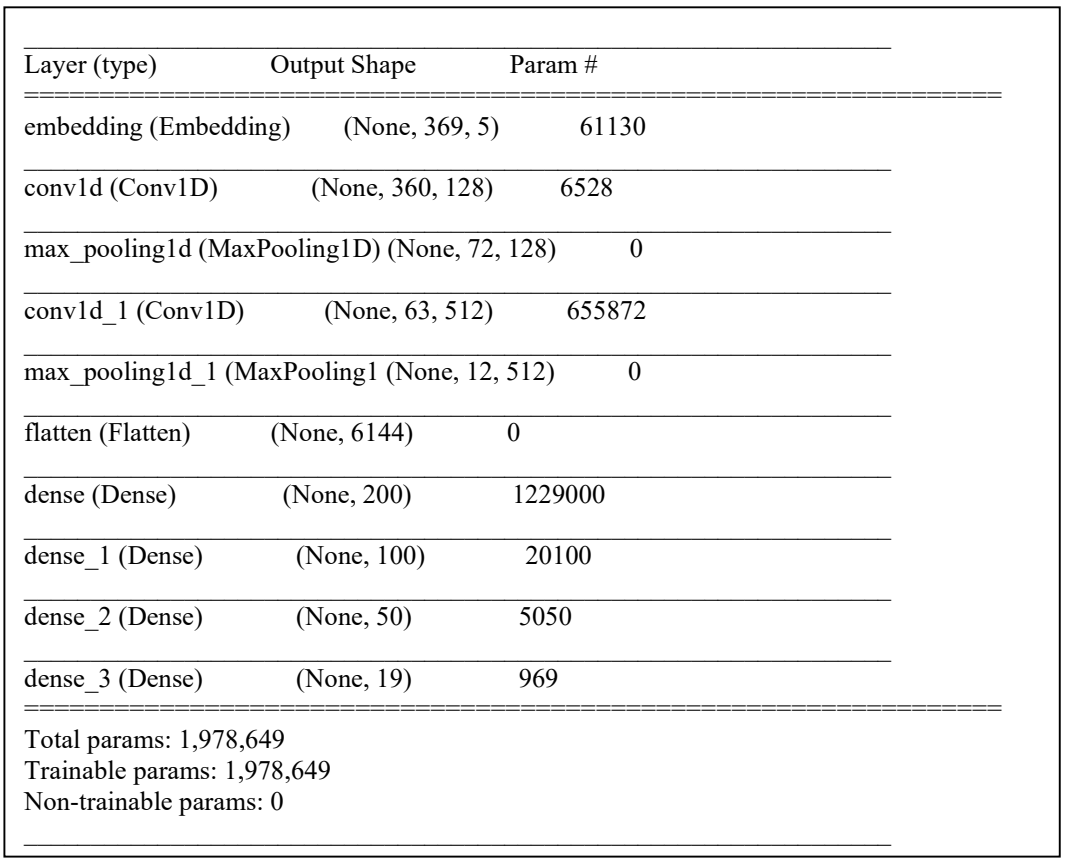
10.7 **Příloha G: Poznámky a doporučení pro replikaci výsledků práce**

V důsledku stochastické povahy použitého optimalizačního algoritmu, není možné přesně replikovat výsledné hodnoty metrik ohodnocení navržených modelů. Za předpokladu zachování v popsaného nastavení procesu učení, je ovšem dosažení srovnatelného ohodnocení téměř jisté.

Program je možné spustit na jakémkoliv počítači splňujícím požadavky knihovny TensorFlow a s prostředím zahrnujícím požadované balíky jazyka Python. Přesný výčet požadovaných balíků, včetně jejích verzí je obsažen v souboru requirements.txt.

Pro jednodušší práci s programem a analýzu kódu je vhodné otevřít soubor IncidentalAccident.sln ve vývojovém prostředí Visual Studio 2019, nebo kompatibilní alternativě, a využít funkcí IDE k instalaci požadovaných balíků.

10.8 **Příloha H: Výpis architektury modelů**



Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 369, 5)	61130
conv1d (Conv1D)	(None, 360, 128)	6528
max_pooling1d (MaxPooling1D)	(None, 72, 128)	0
conv1d_1 (Conv1D)	(None, 63, 512)	655872
max_pooling1d_1 (MaxPooling1	(None, 12, 512)	0
flatten (Flatten)	(None, 6144)	0
dense (Dense)	(None, 200)	1229000
dense_1 (Dense)	(None, 100)	20100
dense_2 (Dense)	(None, 50)	5050
dense_3 (Dense)	(None, 19)	969

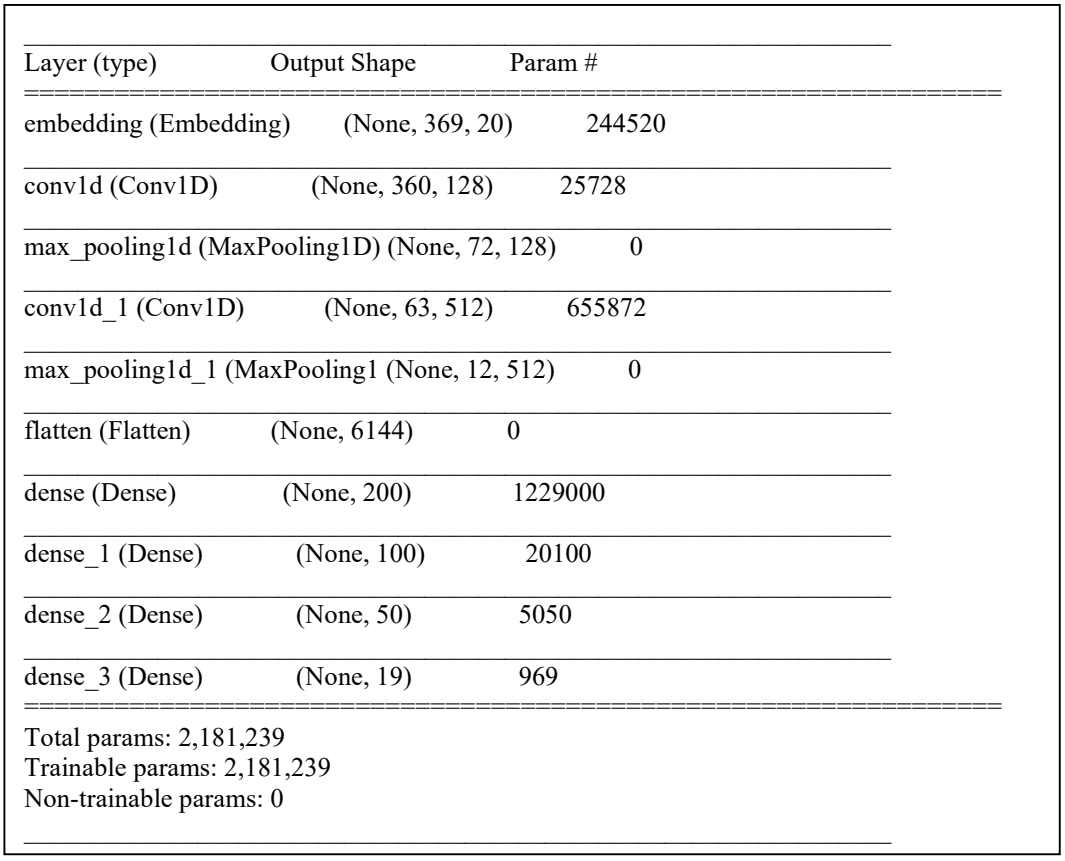
Total params: 1,978,649
Trainable params: 1,978,649
Non-trainable params: 0

1 Model Alpha

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 369, 5)	61130
conv1d (Conv1D)	(None, 360, 128)	6528
max_pooling1d (MaxPooling1D)	(None, 72, 128)	0
conv1d_1 (Conv1D)	(None, 63, 512)	655872
max_pooling1d_1 (MaxPooling1	(None, 12, 512)	0
flatten (Flatten)	(None, 6144)	0
dense (Dense)	(None, 2000)	12290000
dense_1 (Dense)	(None, 500)	1000500
dense_2 (Dense)	(None, 400)	200400
dense_3 (Dense)	(None, 300)	120300
dense_4 (Dense)	(None, 200)	60200
dense_5 (Dense)	(None, 100)	20100
dense_6 (Dense)	(None, 50)	5050
dense_7 (Dense)	(None, 19)	969

Total params: 14,421,049
Trainable params: 14,421,049
Non-trainable params: 0

2 Model Beta



3 Model Gamma

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 369, 20)           244520
_________________________________________________________________
conv1d (Conv1D)              (None, 360, 128)          25728
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 72, 128)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 63, 512)           655872
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 12, 512)           0
_________________________________________________________________
flatten (Flatten)            (None, 6144)              0
_________________________________________________________________
dense (Dense)                (None, 2000)              12290000
_________________________________________________________________
dense_1 (Dense)              (None, 500)               1000500
_________________________________________________________________
dense_2 (Dense)              (None, 400)               200400
_________________________________________________________________
dense_3 (Dense)              (None, 300)               120300
_________________________________________________________________
dense_4 (Dense)              (None, 200)               60200
_________________________________________________________________
dense_5 (Dense)              (None, 100)               20100
_________________________________________________________________
dense_6 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_7 (Dense)              (None, 19)                969
=================================================================
Total params: 14,623,639
Trainable params: 14,623,639
Non-trainable params: 0
_________________________________________________________________
```

4 Model Delta