TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Analýza textů online periodik pomocí metod strojového učení

Bakalářská práce

Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie

Autor práce: **Tomáš Krechler**
Vedoucí práce: Ing. Karel Paleček, Ph.D.
Ústav informačních technologií a elektroniky

Liberec 2020

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Analýza textů online periodik pomocí metod strojového učení

Jméno a příjmení: **Tomáš Krechler**
Osobní číslo: M16000039
Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie
Zadávající katedra: Ústav informačních technologií a elektroniky
Akademický rok: **2019/2020**

Zásady pro vypracování:

1. Seznamte se s problematikou neuronových sítí pro zpracovaní textu.
2. Sestavte dataset obsahující texty článků a uživatelských komentářů a vhodně upravte pro snadné zpracovaní neuronovými sítěmi.
3. Navrhněte systém pro automatické generování významných částí článku, příp. souvisejících uživatelských komentářů.
4. Vyhodnoťte úspěšnost pomocí standardně používaných metrik.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30-40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] Goodfellow, I., Bengio, Y., Courville, A. Deep learning. MIT Press, 2016 Bishop, C. Pattern Recognition and Machine Learning. 2006. ISBN 13: 978-038731073 Karpathy, A., Johnson, J., Li, F. Convolutional neural neworks for visual recognition. dostupné online: http://cs231n.stanford.edu/

Vedoucí práce:	Ing. Karel Paleček, Ph.D. Ústav informačních technologií a elektroniky
Datum zadání práce:	9. října 2019
Předpokládaný termín odevzdání:	18. května 2020

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 18. října 2019

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

4. září 2020 Tomáš Krechler

Poděkování

Rád bych poděkoval Ing. Karlu Palečkovi Ph.D. za odborné vedení, věcné připomínky a vstřícnost při vypracování této bakalářské práce.

Abstrakt

Cílem této práce je příprava pro analýzu politického smýšlení obyvatelstva, rozdílnost mezi komentáři různých webů a snaha o zjištění funkčnosti rekurentních neuronových sítí pro český jazyk. Zaměřil jsem se na webové portály novinky.cz a idnes.cz, data byla sbírána po dobu pěti let.

K řešení zvoleného problému jsem využil právě rekurentních neuronových sítí s Long short-term memory buňkami. Navrhl jsem tři různé modely. První po natrénování na textu generuje texty po znacích. Druhý použije word2vec slovník slov a jejich příslušných číselných vektorů ke klasifikaci sentimentu komentářů a poslední za použití stejných slovníků generuje texty po slovech.

K naprogramování modelů jsem použil jazyk Python a nástroj JupyterLab. Obě metody generování vytvářely text, vypadající jako čeština, ovšem občas postrádající smysl. Jelikož roli hraje náhoda, lépe vygenerované komentáře by mohly být na první pohled zaměnitelné s člověkem psaným textem. Klasifikace sentimentu dosáhla pro web iDnes 61 % přesnosti a pro web Novinky 71 %. Tyto dva modely se při klasifikaci shodovaly v 80 % případů. Provedený výzkum naznačil, s relativně nízkou hladinou významnosti, trend v podobných náladách diskutujících na obou periodikách. Při porovnání podobnosti slov jsou vidět patrné rozdíly v použití slova. Na každém webu je použito v různých větách s rozdílnými citovými zabarveními.

Hlavním zjištěním této práce je, že rekurentní neuronové sítě lze dobře použít i pro český jazyk. Vyšších přesností klasifikace a menší chybovosti a smysluplnosti generování by se dalo dosáhnout především delším či paralelním trénováním na více zařízeních. Na stejném principu lze analyzovat i další periodika a utvořit si tak ucelený přehled o politické náladě ve společnosti.

Klíčová slova: rekurentní neuronové sítě, Long short-term memory, Adam, Python, JupyterLab, klasifikace sentimentu, generování textu, po slovech, po znacích, word2vec, periodika

Abstract

The main goal of this bachelor thesis is to prepare basis for analysis of the political mindset of the population, differences between the comments from different web sites and to determine functionality of recurrent neural networks for Czech language. I focused on web portals novinky.cz and idnes.cz. The data were collected for five years.

To solve the problem, I used recurrent neural networks with long short-term memory cells. I designed three different models. The first for generating texts by characters. Second one uses word2vec dictionary of words and their respective number vectors to classify sentiment of the comments. Last one uses same dictionaries to generate text by words.

To program the models, I used language Python and JupyterLab tool. Both text generation models produced text that looked like Czech, but sometimes lacking in meaning. Since chance plays a role, better generated comments could at first glance be interchangeable with human written text. The sentiment classification model reached 61% accuracy for the website iDnes and 71% for Novinky. However, these two models agreed in classification of same comments in 80% of cases. The research indicated with relatively low level of significance a trend of similar moods of discussing in both periodicals. When comparing the similarities of words, obvious differences in different uses of the word are seen. On each site words are used in distinct sentences with unalike emotions behind.

The main finding of this thesis is that recurrent neural networks can also be well used for the Czech language. However, higher accuracy of classification and lower error of meaningfulness of generated texts could be achieved by longer or parallel training on multiple devices. The same principle can be applied to analyze other periodicals and thus create a comprehensive overview of political moods in societies.

Keywords: recurrent neural network, Long short-term memory, Adam, Python, JupyterLab, sentiment classification, text generating, word level, char level, word2vec, journal

Abstract

Seznam obrázků

Seznam rovnic

Seznam tabulek

Seznam zkratek

RNS	Rekurentní neuronová síť
RNN	Recurrent neural network
GRU	Gated recurrent unit
LSTM	Long short-term memory
GD	Gradient descent
MGD	Minibatch gradient descent
SGD	Stochastic gradient descent
Adam	Adaptive moment estimation

1 Úvod

Hlavním předmětem mé práce je analýza textů online periodik za použití neuronových sítí, což je v posledním desetiletí velmi populární téma, jak v České republice, tak ve světě. Na internetu se neustále objevují nové články o fungování a principech neuronových sítí a strojovém učení obecně. To indikuje zájem širší programátorské veřejnosti o tuto problematiku, ovšem většina pracuje pouze s anglickými texty, popřípadě dalšími rozšířenějšími jazyky založených na latince. Je tomu tak z důvodu, že angličtina je jazyk amorfní, tzn. že nepoužívá skloňování či časování slov a jejich forma zůstává stejná. To je výhodné z hlediska strojového učení textů, jelikož se model nemusí učit několik různých tvarů slov. Opakem je čeština jakožto jazyk flektivní, která je bohatá na ohýbání tvarů. Na internetu je několik článků, které ukazují, že se dají i veřejně přístupné neuronové sítě používané ke generování angličtiny celkem dobře použít k automatickému generování českých textů.

Podnětem k vypracování mé práce byl zájem o rozdílnost mezi názory čtenářů různých webových periodik a vyzkoušení chování rekurentních neuronových sítí při generování textů v českém jazyce, kde vstupními daty nejsou vydávané publikace, ale komentáře diskutujících, a to včetně pravopisných chyb a překlepů.

Předpokládám, že na různých serverech se utvořily různé komunity občanů s různými životními pozadími a názory. Domnívám se, že komentář, který by na jednom webu byl přijat diskutujícími kladně, by mohl být na jiném přijat záporně. Očekávám, že výsledky budou více rozdílné, pokud vstupními daty budou pouze články týkající se politiky namísto všech rubrik. U politických článků se také mnohem více diskutuje. K ověření těchto hypotéz použiji klasifikaci sentimentu komentářů z obou periodik a porovnám výsledky. Druhou metodou pro srovnávání výsledků bude rozdíl mezi podobností použití slov v komentářích. Poslední technikou pro odlišnost příspěvků je hledání rozdílů ve vygenerovaných komentářích.

V této práci nejdříve stručně nastíním, co to jsou neuronové sítě a jak fungují. Poté objasním, v čem se liší rekurentní neuronové sítě a představím tři různé architektury, které následně použiji v praxi. V neposlední řadě použiji knihovnu *gensim* pro vytvoření modelu *word2vec*, který každému slovu přiřazuje vektor čísel, pomocí kterého lze zjistit podobnosti mezi slovy.

Jako vstupní data poslouží nasbírané články (včetně komentářů) za období posledních pěti let z webových portálů *www.idnes.cz* a *www.novinky.cz*, které poskytl vedoucí práce Ing. Karel Paleček Ph.D.

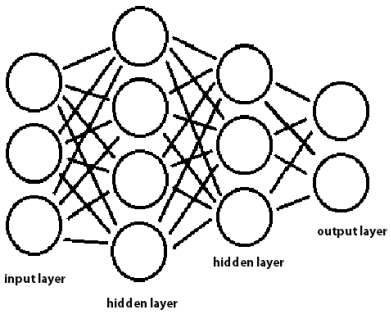
2 Problematika

2.1 Neuronové sítě

Může se zdát, že neuronové sítě jsou takzvané *black boxy*, do kterých se posílají vstupní data a neuronová síť si sama vyhledá a naučí se podobnosti a vzory z dat. To může na jednu stranu být pravda, ale je třeba znát, jak vrstvy modelu poskládat a jaká architektura je vhodná pro daný typ úlohy.

Neuronové sítě se mohou skládat z jedné až několika různých vrstev. Jedna vrstva je složena z různého počtu neuronů, které jsou navzájem propojeny mezi vrstvami.

Každé toto spojení má vlastní váhu – číslo, tato čísla jsou pak seskupena do matic. Každý neuron tak vezme všechna vstupní data, nebo hodnoty z předchozí vrstvy, ty přenásobí příslušnými vahami a sečte. Výsledný součet se pošle do aktivační funkce, která transformuje hodnotu a přiřadí ji neuronu. Aktivačních funkcí se používá mnoho, mezi často používané patří *sigmoid*, či *ReLU*.



Obrázek 1 Vícevrstvý perceptron

Neuron lze tedy zapsat jako: $y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right)$

Rovnice 1 Neuron

Výstup pro *k*-tý neuron je y_k, φ je prahová (aktivační) funkce, *m* je počet vstupů *x* a w_{kj} je váha pro *j*-tý vstup *k*-tého neuronu.

Sigmoid: $f(x) = \frac{e^x}{e^x+1}$

Rovnice 2 Aktivační funkce Sigmoid

ReLU: $f(x) = \max(0, x)$

Rovnice 3 Aktivační funkce ReLU

Takto se síť prochází vrstvu po vrstvě až k výstupní vrstvě, ve které se data dají interpretovat jako výsledek. Je možné, že výsledek neodpovídá skutečnosti, pak se použije algoritmus zpětného šíření, při němž procházíme sítí od konce na začátek a pozměňujeme jednotlivé váhy tak, aby lépe odpovídaly skutečnosti. Jednotlivé změny vah se pro zpětné šíření vypočítají za pomocí diferenciálních rovnic.

2.2 Učení modelů

2.2.1 Učení bez učitele

Při učení bez učitele se snažíme najít skryté struktury v datech, přestože nemáme ke vstupním datům žádné označení správných výsledků. Dvě nejčastější použití tohoto přístupu je průzkumná analýza a redukce dimenzí dat. Při tomto postupu jsou vhodné algoritmy *k-means clustering*, analýza hlavních komponent, detekce anomálií a *autoencoder*. [1]

Jelikož tyto modely nemají určené správné výstupy, tak není přesný postup, jak porovnávat úspěšnosti většiny modelů, využívající tento způsob učení.

2.2.2 Učení s učitelem

Naopak při učení s učitelem učení probíhá tak, že jako data se použijí odpovídající dvojice – vstup *x* a správný výstup *y*. Po průchodu vstupních dat sítí dostaneme aktuální výsledek a po porovnání s *y* určíme velikost chyby. Poté se za použití zpětného šíření korigují hodnoty vah. Tento postup se opakuje, dokud rozumně klesá chybovost. Při delším učení, kdy už chybovost klesá pomalu, se stává, že se model až příliš dobře naučí na daný dataset a poté má horší přesnost při klasifikování podobných dat, která ovšem nebyla v trénovacím datasetu. Proto je vhodné trénovací data rozdělit na trénovací a validační.

Přeučení se dá minimalizovat za pomoci regularizace. Mezi metody regularizace patří například včasné zastavení učení – když začne růst chybovost na validačních datech i přesto, že stále klesá na trénovacích, nebo zařazení *dropoutu*, který s námi určenou pravděpodobností propouští výstupy z neuronů dále sítí.

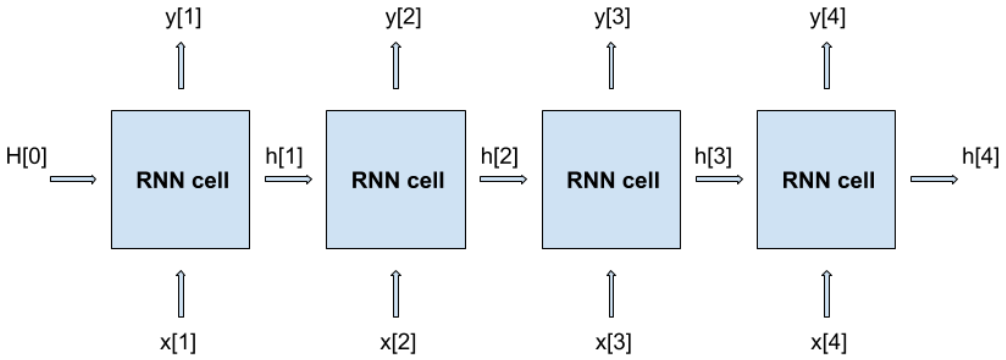
Z použitých datasetů utvářím dvojice následujících slov nebo znaků, které představují dvojice vstup–správný výstup, a proto tento přístup používám k trénování modelů v této práci.

Mezi druhy strojového učení s učitelem patří především regrese a klasifikace.

2.3 Rekurentní neuronové sítě

Rekurentní neuronová síť je druh umělých neuronových sítí, kde jejich propojení jednotlivých jednotek tvoří směrovaný graf (Obr. 2). To síti umožňuje zohlednit předchozí vstupy a vykazovat dynamické chování v čase. [2] Oproti dopředným neuronovým sítím tedy mají vnitřní paměť (skrytý stav) h, jež slouží právě k uchování předchozích stavů. Jako vstup tedy používají mimo vstupní data x i skrytý stav z předchozího kroku h_{t-1} (ten je při prvním kroku inicializován nejčastěji na nuly). Jejich výstupem je y, stejně jako u dopředných sítí, a nový skrytý stav h_t. Proto se tato architektura sítí hodí pro textové úlohy a sekvenční úlohy obecně, kde je potřeba znát kontext ke vstupu. Naopak v dopředných sítích jsou na sobě jednotlivé vstupy nezávislé. I rekurentní neuronové sítě používají a učí se za pomoci zpětného šíření.

Rekurentní neuronové sítě se dnes používají v predikcích slov, rozpoznávání řeči, kompozici hudby, predikci akciových trhů a časových řad.



Obrázek 2 Rekurentní neuronová síť

Mezi hlavní problémy klasických rekurentních neuronových sítí patří neschopnost zapamatovat si kontext z více než několik kroků nazpět. A to z důvodu, že skryté stavy po každém kroku prochází aktivační funkcí a nelinearita je pro dlouhodobou paměť nevhodná. Pro uchování starších stavů je vhodné váhy skrytého stavu pouze odečítat, nebo přičítat. [3]

Dalším problémem je pak mizející, či naopak explodující gradient – to způsobuje, že síť, stejně jako v předchozím problému, přestává zohledňovat starší vstupy. Jako jedno z možných řešení je vytvořit maximální a minimální hodnoty pro gradient, či použít jinou architekturu.

K vyřešení problému s krátkodobou pamětí byly navrženy buňky *GRU* a *LSTM*. V dnešní době u většiny problémů, kde se používají rekurentní neuronové sítě, se aplikují právě výše zmíněné architektury.

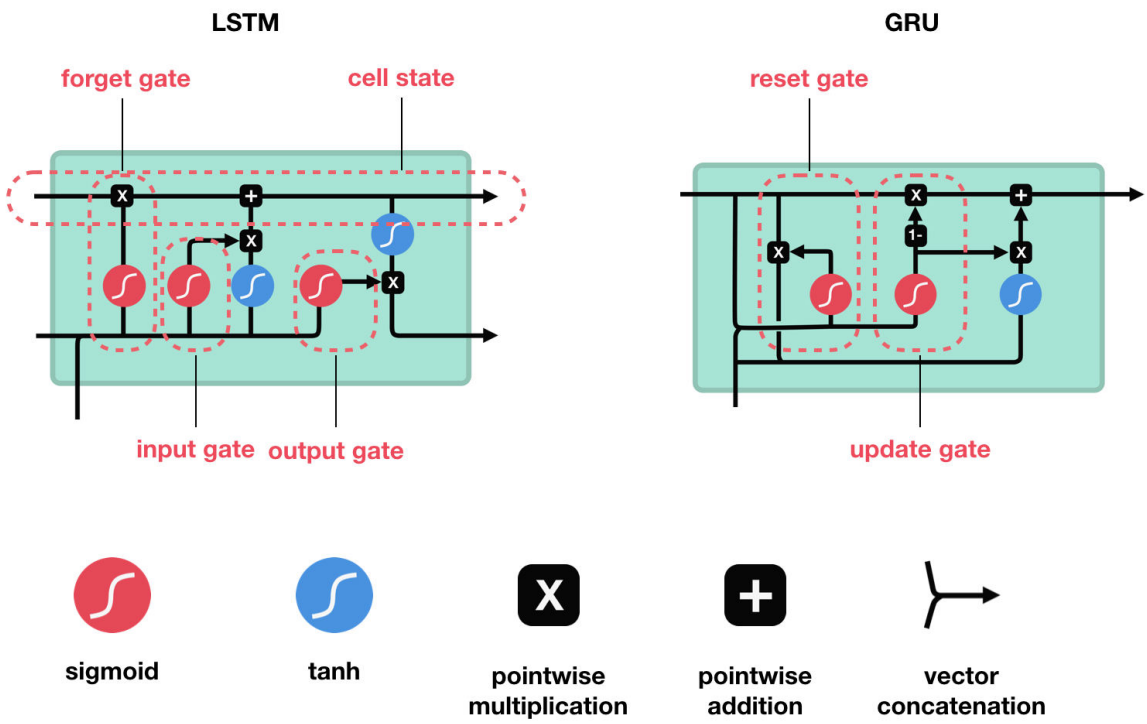
2.3.1 LSTM

Long short–term memory sítě (poprvé představeny již v roce 1997) mají podobný tok dat jako klasická RNS (rekurentní neuronová síť), dopředně zpracuje data, vyhodnotí výsledek a použije zpětné šíření. Rozdíl je v obsahu buňky, zatímco RNS pouze spojí předchozí skrytý stav se vstupem a vypočítá hyperbolickou tangentu čísel, *LSTM* má tři různé brány – vstupní, výstupní a zapomínající bránu. Navíc má dva různé skryté stavy, a to skrytý stav, který se předává mezi jednotlivými kroky a stav buňky, využívaný jako dlouhodobá paměť.

Zapomínající brána (na Obr. 3 *forget gate*) slouží k určení, jaké informace jsou důležité. Předchozí skrytý stav a vstup se zřetězí a pošlou se do sigmoid funkce. Ta má v tomto případě oproti hyperbolické tangentě výhodu v tom, že má rozsah hodnot od 0 do 1 namísto od -1 do 1. Pokud se má informace zapomenout, je vhodnější násobit nulou. Výsledek po průchodu sigmoidou se pak vynásobí Hadamardovým součinem (násobení stejně rozměrných matic nebo vektorů po prvcích) se stavem buňky, tím se tato paměť aktualizuje vynásobením nedůležitých hodnot čísly blížícími se nule.

Ve vstupní bráně (na Obr. 3 *input gate*) se nachází dvě funkce, hyperbolický tangens a sigmoid. Do obou se pošle zřetězený předchozí stav a vstupní data a výsledky těchto funkcí se mezi sebou pronásobí Hadamardovým součinem. Tento součin se poté přičte k paměti buňky. Tak paměť získává novou informaci z momentálních vstupů, která se použije v dalším kroku.

Poslední branou je výstupní brána (na Obr. 3 *output gate*). Tato brána slouží k vytvoření nového skrytého stavu a výstupu *LSTM* buňky. Na zřetězený vstup a předchozí skrytý stav se použije funkce sigmoid, a poté se Hadamardovým součinem vynásobí s aktualizovanou pamětí buňky ze vstupní brány, na kterou již byl aplikován hyperbolický tangens. Výsledek se rozdělí na výstup a nový skrytý stav do dalšího kroku.



Obrázek 3 Buňky LSTM a GRU, Dostupné z https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

2.3.2 GRU

Gated recurrent unit byla představena v roce 2014. Architekturou se podobá buňce *LSTM*, ale je zjednodušena ochuzením o paměť buňky. Přesto sítě s *GRU* dokážou na některých menších datasetech překonat v přesnosti *LSTM*. Ovšem ve většině případů má lepší výsledky architektura používající *LSTM*.

GRU pro přenos informace používá pouze jeden skrytý stav, a má jen dvě brány, resetovací a aktualizační.

Resetovací brána (na Obr. 3 *reset gate*) slouží k výběru informací, které budou zapomenuty. Nejprve se zřetězí předchozí skrytý stav h_{t-1} se vstupními daty x_t a na ty se použije funkce sigmoid. Výsledek sigmoidy se Hadamardovým součinem opět pronásobí s h_{t-1} a zřetězí s x_t. Součin po průchodu hyperbolickou tangentou vytvoří r_t.

V aktualizační bráně (na Obr. 3 update *gate*) se zřetězený x_t a h_{t-1} pošle do sigmoidy a výsledná matice z_t se použije na dvou různých místech. Jednou se jako $(1 - z_t)$ Hadamardovým součinem vynásobí s h_{t-1} a utvoří n_t. Podruhé se z_t v Hadamardově součinu s r_t, součin se poté sečte s n_t a vzniká nový skrytý stav h_t a y_t.

2.4 Loss funkce a hledání minimální chyby

2.4.1 Loss funkce

Loss funkce je taková funkce, jejíž hodnotu se podle druhu úlohy snažíme buď maximalizovat nebo minimalizovat. Ukazuje, jak daleko je výsledek od očekávaného výsledku neboli určuje velikost chyby na základě dané predikce.

Nejjednodušší a bezesporu nejpoužívanější na strojové učení obecně je střední kvadratická chyba, známá pod zkratkou *MSE (mean squared error)*, což je suma čtverců rozdílů predikované hodnoty od správné. Ta je vhodná a hojně využívaná pro úlohy regrese.

Modely v této práci jsou ovšem klasifikace. Funkce pro binární klasifikaci jsou vhodné například *hinge loss* a *binary cross entropy*, ten také používám pro klasifikaci sentimentu komentářů. Ovšem pro klasifikaci do více tříd, kterou používám pro generování textů, je vhodné použít zobecnění binární křížové entropie – *multi-class cross entropy loss*.

Multi-class cross entropy loss: $MCEL(X_i, Y_i) = -\sum_{j=1}^{c} y_{ij} * log_2(p_{ij})$

Rovnice 4 Multi-class cross entropy loss

Výsledek *loss* funkce pro vektor predikcí jednotlivých tříd X a *onehot* kódovaný vektor správných tříd Y je tedy mínus suma přes všechny třídy logaritmů o základu 2 pravěpodobností p_{ij}, která říkají, s jakou pravděpodobností patří i-tý element do třídy j, vynásobená jedničkou, pokud i-tý element patří do třídy j, jinak nulou. C je počet tříd. Pravděpodobnosti jsou vypočítány *softmaxem*.

Jinými slovy, pokud je vždy správná pouze jedna z predikovaných tříd, pak je hodnota *loss* funkce mínus jedna krát logaritmus o základu dva pravděpodobnosti, že nastala právě správná třída.

2.4.2 Optimalizace loss funkce

Dalším krokem je hledat takové parametry, kde zvolená funkce bude minimální neboli model bude predikovat s nejmenší chybou.

Gradient descent je jedním z nejpopulárnějších a nejběžnějších algoritmů pro optimalizaci neuronových sítí. Idea za tímto a algoritmem je ta, že pokud jsme schopni vypočítat derivaci funkce, můžeme najít její minimum. Jedná se o iterativní algoritmus, který určuje, jakým směrem ve funkci se vydat k dosažení minima za pomoci derivace (při více proměnných za pomoci gradientu, což je vektor parciálních derivací). Poté stačí pouze zvolit velikost kroku učení a přepočítat parametry. Je potřeba dbát na velikost kroku, protože pokud je příliš vysoká, nemusíme dojít k minimu a vypočtené hodnoty pro jednotlivé kroky mohou divergovat. Při příliš malých krocích musí algoritmus opakovat mnoho iterací náročných na procesor, zvláště při velkém objemu dat. Algoritmus končí, pokud parametry konvergují pod určenou hodnotu, či dosáhne určeného počtu iterací.

Složitost *gradient descentu* (*GD*) roste s počtem dat kvadraticky, proto se pro zjednodušení používá *Stochaistic gradient descent (SGD)*, případně *Minibatch gradient descent* (*MGD*). *SGD* a *MGD* oproti *Gradient descentu* gradient pouze minimum aproximují, proto kolem minima většinou oscilují. *GD* do minim konverguje přesně, ale naopak má větší náklonost spadnout do lokálního minima. Ovšem vzhledem k paměťovým a výpočetním nárokům *GD* se používají pro trénink neuronových sítí především *SGD* a *Minibatch GD*.

Existuje mnoho vylepšení klasického *SGD* a *MGD*, které konvergují k minimu rychleji. Mezi ně patří například algoritmus *Adam*.

Optimizér *Adam (Adaptive moment estimation)* byl představen v roce 2015 a od té doby strmě získává na popularitě v oblasti hlubokého učení, jelikož dokáže rychle dosáhnout dobrých výsledků. Během posledních 5 let přišla i další různá vylepšené algoritmy vycházející z optimizéru *Adam*, ale i když by mohly dojít k výsledkům rychleji, ještě nejsou tolik rozšířené mezi knihovnami s nástroji pro strojové učení. Z těchto důvodů jsem si vybral *Adam* jako hlavní optimizér v mé práci.

Adam je vhodný i na nekonvexní funkce, kombinuje výhody *AdaGrad (Adaptive gradient algorithm)*, vhodný na jazykové úlohy a počítačové vidění, a *RMSProp (Root mean square propagation)*, vhodný na zašuměná data. Stejně jako *Adam* jsou *AdaGrad* a *RMSProp* algoritmy založené na *SGD*. Velmi často je výchozí nastavení hyperparametrů pro optimizér *Adam* dobré a nepotřebuje příliš ladění. [4]

Místo aby *Adam* přizpůsobil rychlost učení parametrů pouze na základě prvního momentu (průměr) jako u *RMSProp*, využívá také druhého momentu (necentrální rozptyl). Konkrétně algoritmus počítá exponenciální proměnlivý průměr gradientu a kvadratický gradient. Uchování předchozích stavů momentů pak řídí β_1 a β_2.

První moment: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$

Rovnice 5 Adam: Výpočet prvního momentu

Druhý moment: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

Rovnice 6 Adam: Výpočet druhého momentu

Další krok momentu tedy vypočteme jako minulou hodnotu momentu vynásobenou daným hyperparametrem β, ke které přičteme (kvadratický) gradient vynásobený hodnotou jedna mínus daný hyperparametr β. Těmito hodnotami se poté upraví váhy.

Úprava vah (bez korekcí): $W = W - \alpha * \frac{m_t}{\sqrt{v_t + \varepsilon}}$

Rovnice 7 Adam: Úprava vah

V této rovnici jsou hyperparametry α velikost kroku učení a ε malý skalár, aby se zamezilo případnému dělení nulou. *W* je matice vah.

2.5 Metriky využívané pro hodnocení úspěšnosti

Jakmile je model natrénovaný, nastává otázka, jak dobrý opravdu je. Všechny úlohy řešené v této práci jsou klasifikace, ať už binární, či více třídové, a proto k měření úspěšnosti budu používat následující metriky.

Nejčastější metriky pro měření predikcí, klasifikací, úspěšnosti modelu obecně jsou správnost *(accuracy)*, přesnost *(precision)*, úplnost *(recall / sensitivity)* a F1 skóre *(F1-score)*.

K určení těchto hodnot potřebujeme znát chybovou matici *(confusion matrix)*. Tu zjistíme porovnáváním predikovaných hodnot s reálnými a rozřazováním výsledků do čtyř různých skupin (Tab. 1).

		Predikovaná třída	
		Positive	Negative
Reálná třída	Positive	TP	FN
	Negative	FP	TN

Tabulka 1 Confusion matrix

První dvě hodnoty z chybové matice *true positive* a *true negative* jsou pozorování, které model správně predikoval a v tabulce jsou zelenou barvou. Nacházejí se na hlavní diagonále chybové matice. Mimo hlavní diagonálu jsou predikce chybné (v tabulce znázorněny červeně).

True positives (TP) jsou případy, které byly sítí predikovány kladně / ano a reálně tomu tak bylo.

True negatives (TN) naopak jsou případy, které byly sítí predikovány záporně / ne a reálně tomu tak bylo.

False positives (FP) nastává, když model predikuje pozitivní výsledek, ale reálně je záporný. Tyto chyby jsou také nazývané Chyby I. typu.

False negatives (FN) také nastávají, když predikce modelu neodpovídá skutečnosti. Model predikoval negativně, skutečnost byla pozitivní. Tato chyba je Chybou II. typu. [5]

Správnost (Rovnice 8) je nejintuitivnější metrika úspěšnosti modelů. Jedná se o počet správných odpovědí vydělený počtem celkových pozorování. Kritérium přesnosti je vhodné pouze v případě symetrického počtu příkladů pro každou jednotlivou třídu datasetu.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Rovnice 8 Výpočet správnosti

Preciznost (Rovnice 9) udává poměr správně kladně vyhodnocených pozorování vůči celkovým pozorováním kladných jevů.

$$Precision = \frac{TP}{TP + FP}$$

Rovnice 9 Výpočet preciznosti

Úplnost (Rovnice 10) se vypočítá podílem správně vyhodnocených pozorování kladné třídy vůči všem jevům dané třídy.

$$Recall = \frac{TP}{TP + FN}$$

Rovnice 10 Výpočet úplnosti

Poslední z nejpoužívanějších metrik pro klasifikaci je F1 skóre (Rovnice 11). Jedná se o vážený průměr preciznosti a citlivosti. Tato metrika bere v potaz obě skupiny špatných predikcí. F1 skóre je obvykle užitečnější než přesnost obzvláště, když jsou nevyrovnané počty dat jednotlivých tříd v datasetu. [5]

$$F1\ Score\ = 2 * \frac{(Recall * Precision)}{(Recall + Precision)}$$

Rovnice 11 Výpočet F1 skóre

2.6 Existující projekty

2.6.1 Char-RNN

Jedním z nejpopulárnějších projektů posledních let je Char-RNN od Andreje Karpathyho, kterou publikoval 21. května 2015.

Ve svém blogovém příspěvku stručně a názorně představil a vysvětlil princip fungování rekurentních neuronových sítí. Včetně váhových matic, jejich aktualizací za pomoci zpětného šíření. Také představil parametr *temperature* pro korigování náhodnosti generování nových znaků.

To také inspirovalo mnoho dalších programátorů v zájmu o rekurentní neuronové sítě. Většina článků na toto téma má datum zveřejnění mezi roky 2018 – 2020. Přičemž první výzkumné práce na tuto oblast vznikly už před desetiletími.

K tréninku využívá *softmax*, též známý jako *Cross entropy loss* a jako algoritmus pro trénink sítě *Adam* či *RMSProp*. Architektura sítě, která vygenerovala několik ukázek, byla dvouvrstvá *LSTM* s 512 skrytými uzly a *dropout* vrstvou s pravděpodobností propusti 0,5. Při tréninku na pouhém 1 MB velkém souboru (okolo milionu znaků) vypadaly výsledky na první pohled věrohodně. Při bližším prozkoumání ovšem vcelku normálně vypadající věty v angličtině postrádaly smysl.

V dalších příkladech vyzkoušel různé obměny architektury RNS, přičemž síť dokázala generovat kód v jazyce C (sice nezkompilovatelný, ale pro laika nerozeznatelný od reálného kódu), články ve Wikipedii (s občasným nesmyslným leč validním *XML*) či text pro kompilaci v *LaTeXu*. Věrohodně také vypadají dramata od Williama Shakespeara. Generovaný text byl včetně jmen osob, které zrovna danou pasáž promlouvali, a působí jako skutečné drama.

V další části se věnuje vývoji sítě v učení generování textů. Model se nejprve naučí dělit slova mezerami a poté se učí generovat smysluplná slova od kratších k delším. Nakonec vhodně přidá interpunkci a další nealfanumerické znaky. To vše během prvních několika tisíc iterací. Aby se generování týkalo pouze jednoho tématu a obecně dlouhých závislostí, musí síť trénovat násobně déle.

Na závěr je zajímavé vyobrazení, kdy se různé neurony aktivují. Některé jsou aktivovány tím silněji, čím jsou dále v řádku, nebo čím hlouběji vnořené v kódu jsou. Jiné se aktivují například pouze v komentářích, nebo v uvozovkách. Ovšem většina neuronů vykazuje chování, jehož funkci nelze interpretovat.

2.6.2 Open AI

Mnoho nových projektů v oblasti neuronových sítí je zaštiťuje Open AI. Mimo jiné vytvořili například klasifikátor sentimentu, generátor hudby, umělou inteligenci ovládající tým *botů* ve hře Dota 2, jejíž celosvětový turnaj má největší odměnu pro výherce. Podařilo se vytvořit tak silnou umělou inteligenci, že dokázala porazit profesionální týmy hráčů, kteří trénují a hrají tuto hru několik let. Z dalších projektů využívající rekurentní neuronové sítě je mimo jiné právě generátor textu.

Výše zmiňovaný klasifikátor sentimentu pracuje na pozoruhodném přístupu. Model se učí predikovat následující znaky na velkém datasetu. Pak je schopný pomocí pouhých několika set správným sentimentem označených recenzí velmi dobře predikovat celkové hodnocení jiných recenzí. Autoři také naznačují, že tento přístup by mohl být slibný, co se týče obecného učení bez učitele, jakožto vedlejší produkt jazykových modelů. Modelu se pouhým učením predikování následujícího znaku a poté testování predikce recenzí z Amazonu podařilo dosáhnout 91.8 % přesnosti. Model byl tvořen *multiplicative LSTM* s 4096 jednotkami a trénoval se na 82 milionech recenzí přibližně měsíc na čtyřech grafických kartách zároveň. [6]

Generátor textu GPT-2 byl natrénován na 40 GB textu z internetu. Obsahuje 1,5 miliardy parametrů a byl vytvořen s jediným cílem, predikovat následující slova vzhledem ke všem slovům zadaného vstupního textu. Model v celé velikosti nebyl uveřejněn, jelikož se autoři obávají, že by model mohl být využit jak k prospěšnému, tak škodlivému generování obsahu. Uveřejněny byly pouze menší modely pro výzkum a testování.

Vytvářený text generátorem GPT-2 budí dojem, že byl napsán člověkem, nebo při nejmenším se kvalitě lidsky psaného textu přibližuje. Ale i přesto se občas vyskytují takzvané oslí můstky, opakující se text, nebo věcně nesmyslně psaný text, ač gramaticky správný. I přesto že byl model trénován na obecných datech, v několika zkouškách překonal přesnosti jiných modelů trénovaných na specifických datech pro daný test. I proto v jiných testech zdaleka nedosahoval úspěšnosti specializovaných modelů. To ovšem nemusí platit i v následujících letech, jelikož křivka funkce (podobné logaritmické křivce, osa y = přesnost, osa x = počet parametrů modelu) byla stále rostoucí i na největším modelu. [7]

3 Cíl práce

Cílem práce bylo za pomoci vhodných prostředků zmapovat politické nálady na nejčtenějších serverech v České republice. Jako základ této analýzy by měly posloužit různé modely rekurentních neuronových sítí. Porovnat výsledky, zda budou pro jednotlivá periodika shodná, či rozdílná.

3.1 Generování textů

Cílem práce bylo vytvořit modely pro generování textu pro vybraná online periodika, přesněji se jednalo o nadpisy článků a komentáře od diskutujících z webových portálů www.idnes.cz a www.novinky.cz. Text by měl být generován pomocí jak znakových, tak slovních RNS.

3.2 Klasifikace sentimentu

Dalším z cílů bylo navrhnout a natrénovat model, který by predikoval, zda bude komentář diskutujícími přijat kladně, či záporně. Vyzkoušet různá nastavení parametrů a pokusit se dosáhnout co nejvyšší přesnosti a dalších metrik úspěšnosti modelu. Dalším cílem bylo otestovat, zda budou stejné komentáře přijaty stejně nebo různě podle toho, kde ho komentující zveřejní.

4 Realizace řešení

Všechny skripty jsou napsány v jazyce Python za pomocí nástroje JupyterLab, což je interaktivní webové vývojové prostředí. Jednotlivý kód, či text je možné mít rozdělený v buňkách a také ho po jednotlivých buňkách libovolně spouštět.

4.1 Vytvoření datasetů

Prvním krokem bylo vytvořit různé datasety z poskytnutých databází souborů. Nejdříve jsem procházel strom složek – rubrik portálu Novinky a z jednotlivých souborů *parsoval* nadpisy a obsahy článků, komentáře diskutujících a jejich hodnocení z textových souborů. Z textů jsem odstranil interpunkci, ale vytvořil jsem soubory jak s diakritikou, tak bez. To samé jsem udělal pro soubory z portálu iDnes ve formátu *json*. Výsledkem byly pro každý portál textové soubory, které měly na každém řádku jednu položku. Kladné a záporné komentáře obsahují na začátku každé položky i jejich hodnocení.

Datasety	Počet položek	μ slov v řádku	σ slov v řádku	μ znaků v řádku	σ znaků v řádku
IDnes – nadpisy	750 373	8,82	2,69	56,32	15,67
Novinky – nadpisy	136 550	9,39	2,47	61,20	14,41
IDnes – obsah článků	682 919	358,10	311,22	2311,77	1975,40
Novinky – obsah článků	136 550	318,07	244,80	2072,61	1542,76
IDnes – komentáře	45 689 180	24,95	28,58	145,87	168,89
Novinky – komentáře	7 834 618	39,02	35,73	237,58	219,53

Tabulka 2 Informace o datasetech

A další derivované datasety:

- Hodnocení komentářů
- Kladné komentáře
- Záporné komentáře

V datasetu komentářů serveru iDNES je 14 650 447 (32,06 %) kladných, 27 766 827 (60,77 %) neutrálních komentářů (jejich hodnocení je přesně 0) a pouze 3 271 906 (7,16 %) záporných komentářů. Kladné komentáře jsou průměrně o 1 slovo / 6 znaků delší než záporné.

V datasetu komentářů serveru Novinky je 6 252 310 (79,80 %) kladných, pouze 293 691 (3,75 %) neutrálních komentářů a 1 288 617 (16,45 %) záporných komentářů. V tomto datasetu naopak jsou průměrně o 4 slova / 30 znaků delší komentáře s hodnocením záporným.

Z vlastní zkušenosti mohu potvrdit, že na Novinkách se hlasuje u komentářů opravdu hodně a pouze komentáře u nepolitických článků, či maximálně několik minut staré, ještě neobdržely žádný hlas.

4.2 Generování textů po znacích

Dalším krokem bylo vytvořit model, který bude schopen se naučit generovat texty po znacích. Průchodem z již upravených datasetů jsem vytvořil list všech možných znaků – abecedu. Abeceda pro datasety s diakritikou měla délku 53 a pro datasety bez diakritiky 38. Z abecedy jsem vytvořil slovník, který každému písmenu, číslu a znaku pro nový řádek přiřadil index. Dále bylo nutné vytvořit histogram prvních písmen pro každou položku a ten následně vydělit počtem položek datasetu. Tím jsem získal pravděpodobnosti, že daný text bude začínat právě tímto písmenem. Tyto pravděpodobnosti se využijí, pokud není zadán žádný inicializační text pro generování.

Nyní se dají řádky textů, převedené na jejich indexy dle slovníku, použít k učení. Pro rychlejší učení (na grafické kartě) je možné list indexů převést na matici o velikosti počtu řádků datasetu krát zvolená maximální délka pro řádek ve znacích (reálně se jedná o *tensor* knihovny *Pytorch*). To ovšem naopak nese tu nevýhodu, že řádky nemohou být proměnlivě dlouhé jako v případě učení na procesoru. Kratší řádky se doplní do stanovené délky nulami (index nového řádku), ovšem u delších textů se musí konec oříznout.

4.2.1 Definice a inicializace modelu

Nyní už nic nebránilo vytvořit vlastní model. K tomu jsem použil třídu *Module* knihovny *Pytorch,* z které jsem odvozením nadefinoval vlastní model, jeho inicializaci, dopředný průchod (o zpětný průchod se stará knihovna automaticky i podle ručně nadefinovaného dopředného průchodu) a metodu pro inicializaci skrytého stavu.

```
class RNN(nn.Module):
    def __init__(self, voc_size, emb_dim, hidden_size, output_size,
n_layers=1, max_length=150):
        super(RNN, self).__init__()

        self.emb = nn.Embedding(voc_size, emb_dim)
        self.rnn = nn.LSTM(emb_dim, hidden_size, num_layers=n_layers)
        self.fc = nn.Linear(hidden_size, output_size)
        self.hidden_size = hidden_size
        self.n_layers = n_layers

    def forward(self, x, hidden):
        # input a hidden -> output a hidden
        output, hidden = self.rnn(self.emb(x).reshape(1, 1, -1), hidden)
        # plně propojená vrstva, vrací pro každý znak hodnotu
        score = self.fc(output)
        return score, hidden

    def init_hidden(self):
        # Skrytý vektor
        hidden = torch.zeros(self.n_layers, 1, self.hidden_size)
        # Vrácen dvakrát pro hidden state a cell state
        return hidden, hidden
```

Zdrojový kód 1 Definice modelu pomocí knihovny Pytorch

Na Zdrojový kód 1 Definice modelu pomocí knihovny PytorchZdrojový kód 1 můžeme vidět nadefinovaný model. Obsahuje:

- *Embedding* vrstvu, která každému znaku přiřazuje vektor délky d_e reálných čísel
- *LSTM* buňku se vstupem o velikosti d_e a výstupem o velikosti d_h, což je zvolený rozměr skrytých stavů
- Plně propojenou vrstvu neuronů se vstupem o velikosti d_h a výstupem o velikosti předem vytvořené abecedy – vrací pro každý znak pravděpodobnost, že bude následovat

Dopředný průchod jsem nadefinoval tak, že ze vstupního znaku se vytvoří vektor čísel a společně s předchozím skrytým stavem se pošle do buňky *LSTM*. Výsledek se vyhodnotí *fully connected* vrstvou.

Velmi podobně jsou nadefinovány i klasická RNS a síť s buňkami *GRU,* liší se pouze v rozměrech a datových typech skrytého stavu. Rozhodl jsem se využívat *LSTM*, jelikož dosahovala o něco lepších výsledků než *GRU*.

Poté jsem nastavil parametry pro síť – velikosti *Embedding* vrstvy a skrytého vektoru, a inicializoval ji. Jako *loss* funkci jsem použil *Cross entropy loss* a jako optimizér algoritmus *Adam* s *amsgrad* s krokem učení 0,001.

Funkce pro trénování modelu obsahuje cyklus *for*, který pro každou epochu (většinou jsem trénoval epoch 20) vytvoří permutaci indexů vstupních dat a vybere jich prvních 10 %. Následuje snížení míry učení po osmé, čtrnácté a osmnácté epoše vždy na polovinu předchozí hodnoty. Při jiném počtu epoch než 20 jsem vždy snižoval míru učení po přibližně 40 – 50, 70 – 80 a 90 – 95 procentech učení. To umožňuje, aby se model naučil závislosti ještě o něco lépe. U úloh, kde se dá měřit přesnost, jde až o jednotky procent.

Následuje vnořený cyklus *for*, který se opakuje pro každou položku z vybrané podmnožiny indexů. Začíná vynulováním předchozích gradientů zpětného průchodu modelu, vynulováním hodnoty *loss* a nainicializováním nové skryté stavy – vynulování.

Poté se přiřadí do proměnné x list znaků (indexů). Pokud trénujeme na GPU, vezme se řádek z předem vytvořené matice vstupních dat. Do y se přiřadí správné výstupy. Je žádoucí, aby správný výstup pro znak x_i byl následující znak x_{i+1}. Tím pádem list cílů y bude stejný jako list x, pouze začínající od druhého znaku. Na konec y se přidá index nového řádku, kterým získáme rovnost velikostí x a y, a navíc značí konec generování textu.

Nyní následuje třetí vnořený *for* cyklus, který proběhne pro každý znak řádku. V tomto posledním vnoření se už pouze pošle znak x_i a skrytý stav h dopředným průchodem nadefinovaným modelem. Výsledný skrytý stav se uchová pro další znak, zatímco skóre pro jednotlivé znaky se společně se správným výstupem y_i pošlou do optimizéru, který vrací hodnotu chyby, která se připočte do proměnné uchovávající sumu chyby přes celý řádek.

Po průchodu všech znaků textu, se výsledná suma chyb vydělí délkou textu. Na *loss* se zavolá metoda *backward*, která vypočítá *lossy* pro jednotlivé součásti sítě. Ihned je následována funkcí *step* zavolanou na optimizér, který výsledné gradienty připočte k váhovým maticím.

Na konci jsou pouze informativní funkce, vypisující rychlost průchodu, *progress bar* epochy a každých 100 iterací zavolají funkci *sample*, která vrací pokus sítě o vygenerování textu.

```
example = sample(rnn)
max_per_epoch = len(lines)//10

for epoch in range(50):
    if(epoch in [20, 35, 45]):
        optimizer.param_groups[0]['lr'] /= 2

    # data budou nahodne prehazena
    train_ids = np.random.permutation(len(lines))[:max_per_epoch]
    # progressbar
    pb = tqdm.notebook.tqdm(train_ids, desc='ep {:03d}'.format(epoch))

    for it, idx in enumerate(pb):
        hidden = rnn.init_hidden()
        rnn.zero_grad()
        loss = 0.
        x, y = lineToIntList(lines[idx])

        for ic in range(len(lines[idx])):
            # dopredny pruchod pro `ic`-ty znak
            score, hidden = rnn(torch.tensor(x[ic]), hidden)
            # scitani lossu -> loss += crit(input, target)
            loss += criterion(score[0,:,:], torch.tensor([y[ic]]))

        loss /= len(x)
        loss.backward()
        optimizer.step()
```


```
        if it % 100 == 0:
            example = sample(rnn)
        pb.set_postfix(loss='{:.3f}'.format(stats.ravg('train', 'loss')),
ex=example[:40])
```

Zdrojový kód 2 Trénování rekurentní neuronové sítě

Funkce *sample* slouží ke generování textu. Začíná klasicky inicializací skrytého stavu na nuly a následně rozhodnutím, zda existuje inicializační text. Pokud ano, převede se na *tensor* indexů daných znaků. Dále se nechá dopředně projít sítí a vybere poslední znak jako následující vstup. Pokud neexistuje, vybere se náhodný první znak sekvence, dle rozložení pravděpodobností začínajících písmen textů. Dalším krokem je nainicializovat vrstvu *softmax*.

Následuje *while* cyklus, kde v každém kroku pošlu do sítě poslední znak prozatím vygenerovaného textu a skrytý stav. Výsledný výstup se předá do vrstvy *softmax*, která pro každý znak vypočítá pravděpodobnost, že bude následovat. Nyní je několik možností, jak z pravděpodobností vybrat index následujícího znaku.

První způsob, který napadne asi hned každého, je vybrat ten s nejvyšší pravděpodobností (*argmax*). Ten má ale jednu nevýhodu, a to že síť bude predikovat a neustále opakovat velmi častou frázi (příklady z nadpisů článků: „dnes se stalo se stalo se stalo…“ nebo „policie zatkla v praze na slovensku v praze na slovensku…“).

Druhým způsobem je vybrat následující znak náhodně s takovou pravděpodobností, jaká byla vypočítaná *softmaxem*. Tento způsob už dosahoval smysluplnějších výsledků a fungoval celkem dobře, ale stále tu byl problém s nastavením, jak moc náhodné by výsledky měly být pro dosažení lepších a rozmanitějších výsledků.

Tento problém řeší třetí způsob výběru následujícího znaku pomocí výběru z multinomického rozložení. Výsledné skóre znaků z dopředného průchodu se vydělí proměnnou *temperature*. Následně se jednotlivá skóre dají do exponentu Eulerova čísla a výsledky poté do funkce *multinomial*, která vybere prvních *n* vzorků (v mém případě pouze první) z hodnot skóre na základě multinomického rozložení.

Hodnota t*emperature*, v rozmezí (0, 1⟩, dělí pravděpodobnosti před *softmaxem*, takže nižší hodnota způsobuje pravděpodobnější, kontroverznější předpovědi. S velmi nízkým číslem (hodnoty 0.2 a nižší) při generování nastává stejný problém, jako při výběru funkcí *argmax*. Naopak vyšší teploty působí rozmanitější generovaný text, za cenu vyšší chybovosti. [8] Při vyšších hodnotách *temperature* jsou výběry velmi náhodné, což zapřičiňuje, zvláště u češtiny a jiných flektivních jazyků, komolení tvarů slov a gramatiky.

Po vybrání následujícího znaku se zkontroluje, zda nebyl vybrán index 0 pro nový řádek, tím by se generování textu ukončilo. Pokud to je jakýkoliv jiný znak, připojí se na konec generovaného textu a posoudí se, zda není dosaženo maximální délky. Pokud je i tato podmínka splněna nově vygenerovaný znak se použije jako vstup pro generování dalšího znaku.

```
def sample(rnn, init_text='', hidden=None, maxlen=150,
mode='multinomial', temperature=0.5):
    # Pokud není zadán inicializační text, vybereme náhodné první
písmeno, dle rozložení prvních znaků
    out_text = list(init_text)
    if not out_text:
        s = np.random.choice(len(chars), p=p0)
        out_text = [chars[s]]
    if hidden is None: hidden = rnn.init_hidden()

    # Vstup projdeme sítí pro získání aktuálního skrytého stavu
    x = char_tensor(out_text)
    for i in range(len(out_text)):
        score, hidden = rnn(x[i], hidden)

    # následující znak je poslední znak momentálního výstupu
    x = char_tensor(out_text[-1])
    # softmax sloužící k počítání pravděpodobností
    softmax = nn.Softmax(dim=2)

    while True:
        # Dopředný průchod
        score, hidden = rnn(x, hidden)
        # Pravděpodobnosti pro jednotlivé znaky
        p = softmax(score).detach().numpy().squeeze()
```


```
        # výběr následujícího znaku
        if mode == 'multinomial':
            # Následující znak vybrán dle multinomiálního rozložení
            # Temperature blízký 1 dává náhodnější výsledky
            k = torch.multinomial(score.view(-1).div(temperature).exp(),
1)[0]
        elif mode == 'argmax':
            # Vybírá znak s nejvyšší pravděpodobností
            k = np.argmax(p)
        elif mode == 'proportional':
            # Následující znak vybrán náhodně s pravděpodobností ze
softmaxu
            random = np.random.choice(p, 1, p=p)
            k = np.where(p == random)[0][0]

        # Podmínky zastavení generování
        if chars[k] == '\n': break
        out_text.append(chars[k])
        if len(out_text) >= maxlen: break

        # Nový vektor posledního znaku momentálního výstupu
        x = char_tensor(chars[k])

    return ''.join(out_text)
```

Zdrojový kód 3 Samplování RNS

4.2.2 Trénink

Jako CPU k trénování bylo využito 10 let staré Intel iCore i5 760 2.8 GHz, s reálně naměřenou rychlostí 2.93 GHz. Jako GPU pak byla použita NVIDIA GeForce RTX 2060, jejíž jádro běží na rychlosti pouze 300 MHz. Při porovnání rychlostí jader CPU a GPU a naměřených rychlostí je vidět silná korelace. Trénování pro rekurentní sítě nemá na grafickém procesoru smysl, pokud se model neučí po velkých dávkách. Celý proces zpomaluje neustále předávání *embedded* vektoru z CPU na GPU a zpět. V mém případě je i starý procesor mnohonásobně rychlejší pod podmínkou, že se síť učí a generuje znak po znaku.

Porovnání rychlostí iterací:

Popis tréninku	Rychlost
CPU na Google Colab délka 64	15.5 it/s
CPU délka 64	7.5 it/s
CPU s diakritikou délka 150	2.1 it/s
GPU délka 64	0.83 it/s

Tabulka 3 Porovnání rychlostí tréninku modelů

Začátek učení: křřsoťougí1l30fqqč pšořůgasóůhqed

Po půl minutě: m zateni me prelini promamy set vezala

Po 51 200 komentářích (temp 0.5): pane postavit stát a ten nebude to kdo není postavili za podporovat

Po 76 800 komentářích (temp 0.4): poslance tak to se měli jednou stále to se nevydrží státní okamuru a kde je to si nestavět kalouska

Po 102 400 komentářích (temp 0.6): poslanechaku opět modra v dostatní

já jseme víc nemůže ti to ods vlády je si musí nepodávat

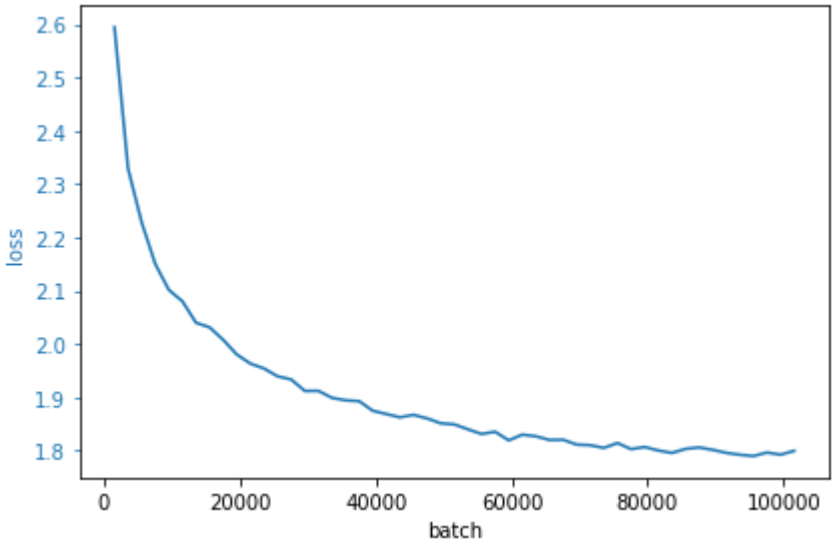
prezident a národ je ho nechat s tim nezase jinak zeman to nedá vykouká po nadívá stranu co nepolitice nikdo

jen největší stboupou stát kamarádované strany a bylo podle kradený

Po 102 400 komentářích (argmax): pane podporu nemá podporu nemá podporu

Při vybírání pomocí módu *proportional* jsou predikované texty většinou do 10 slov – je mnohem častěji predikován konec textu než při módu *multinomial*.

Na příkladech generování je vidět, že z nesmyslně poskládaných znaků už za půl minuty síť pozná, že texty se skládají ze slov a jsou mezi nimi mezery. Stále ale ještě neví, jak poskládat smysluplná slova, přesto je vidět velmi rychlý pokrok (a také pokles hodnoty *loss*). To je také patrné z následující grafu hodnot (Obr. 4).



Obrázek 4 Graf hodnoty loss znakové RNS

K snížení míry učení na polovinu došlo po 51 200 a 76 800 položkách. Je nepatrně vidět, že první snížení *learning rate* pomohlo, síť by jinak konvergovala k *loss* ~ 1,85.

Měřit přesnost generování textů znakovou rekurentní neuronovou sítí nedává smysl, zvlášť když následující znak vybíráme náhodně, a tedy hodnocení zůstává pouze na lidském pozorovateli.

Smysl by mohl být například v měření správnosti tvarů slov – zda síť negeneruje neexistující slova či tvary. Přesto se tyto architektury sítí naučí velmi rychle nevytvářet neexistující slova. Problémy stále mohou být v koncovkách slov, zvláště u jazyku flektivních, jako je čeština, nebo celkové smysluplnosti vět.

4.3 Word2vec model

Existuje možnost vzít již vytvořené a předtrénované *word2vec* modely na obecných datech a volně je použít pro vlastní potřebu. Ovšem v mé práci chci porovnávat rozdíly obou serverů v náhledu na různá slova a zjistit, zda se do vektorů slov promítnou rozdíl v datasetech.

Pro každý dataset jsem vytvořil několik vlastních *word2vec* modelů. Nejprve je nutné načíst datový soubor s texty. Pro menší soubory, které se vejdou do paměti RAM stačí vytvořit list listů slov. Každý řádek je jeden komentář, titulek, článek apod., a ten je rozdělen na jednotlivá slova, oddělená bílými znaky. Pro velké soubory jsem vytvořil *pythonový iterátor*, který operace jako rozdělení na slova a vyfiltrování *stopwords* provádí pokaždé na vyžádání. *Stopwords* jsou slova, která mají pouze syntaktický význam, nebo slova, která se v jazyce vyskytují často, ale nenesou žádnou významovou informaci, neexistuje žádný univerzální list těchto slov pro daný jazyk, často jsou to předložky, spojky, částice, zájmena. Ty je vhodné použít například při klasifikaci sentimentu.

Nyní již stačí naimportovat knihovnu *gensim,* nastavit parametry modelu – dimenze vektorů, posuvné okénko – kolik slov okolo má model zohledňovat, minimální počet výskytů slova v datasetu, vstupní data (list listů slov neboli rozdělených řádků), nebo iterátor (po řádcích přímo ze souboru) a o zbytek se již postará výše zmíněná knihovna.

```
model_emb = gensim.models.Word2Vec(sentences=file_iterator, size=50,
window=6, workers=4, min_count=1)
```

Zdrojový kód 4 Vytvoření a natrénování embedding vektorů pomocí knihovny gensim

V této práci jsem nastavoval, aby jednotlivá slova byla reprezentována vektory o délce 50 případně 128 reálných čísel. U moderních rekurentních sítí, zaměřených na jakékoliv generování textu, jsou slova reprezentována vektory o délce sto až tisíc. Ovšem nutně neplatí, že pokud by vektory byly násobně větší, že se model vylepší. Zde by bylo možné empiricky testovat úspěšnost modelů s měnící se velikostí *embedding* vektorů.

V komentářích (největší dataset) se vyskytuje mnoho pravopisných chyb, někde chybí mezi slovy mezery apod. Proto jsem při vytváření slovníků použil dva různé přístupy. V prvním přístupu jsem omezil slova (pojmem slova se rozumí i čísla oddělená mezerou, či kombinace písmen a čísel) na malá písmena, odstranil interpunkci a diakritiku.

V druhém přístupu jsem sice interpunkci, diakritiku, případně velká a malá písmena ponechal, ale přidal jsem podmínku pro zařazení slova do slovníku a to, že se slovo muselo v datasetu vyskytovat alespoň xkrát (3, 5, 8, 10krát). Tuto podmínku jsem zavedl, aby nebyl slovník přehlcený různými tvary (i chybnými). Předpokládal jsem, že správný tvar se v datasetu bude vyskytovat častěji nežli chybný.

Velikosti *word2vec* slovníků s vektory o dimenzi 50 pro 7 834 618 komentářů bez diakritiky z webu novinky.cz:

- 1,55 GB všechna slova
- 0,1 GB slova, která se vyskytovala alespoň desetkrát

Velikosti word2vec slovníků s vektory o dimenzi 50 pro nadpisy článků:

- 21 MB všechna slova, 136 550 nadpisů z novinky.cz
- 53 MB všechna slova, 750 373 nadpisů z idnes.cz

Velikosti pro prvních 100 000 komentářů bez diakritiky:

- 54 MB novinky.cz (3 343 435 slov, 16 427 434 znaků)
- 38 MB idnes.cz (3 265 187 slov, 15 327 203 znaků)

Je patrný nezanedbatelný rozdíl ve velikostech slovníků. Slovník iDNESu obsahuje o ~ 30 % méně různých slov. Příčinou může být pestrost vyjadřování diskutujících na Novinkách nebo více chyb při psaní. Mé měření ukázalo, že rozdíl v délce komentářů je pouze ~ 2,5 %. Tím se neprokázala souvislost s takto velkým rozdílem v počtu různých slov mezi oběma periodiky.

Mimo jiné je nutno podotknout, že 100 000 komentářů z webu novinky.cz obsahuje mnohonásobně více různých slov, než 136 550 nadpisů ze stejného webu a přibližně stejně jako 750 373 nadpisů z idnes.cz.

Má hypotéza byla, že pokud budeme zohledňovat slova, která se vyskytují alespoň xkrát, eliminuje se většina překlepů a velikosti slovníku se srovnají.

Porovnání velikosti slovníků pro prvních 100 000 komentářů. Slovníky obsahují pouze různá slova, která se vyskytla v datasetu alespoň třikrát nebo pětkrát.

Velikost (v kB)	Zdrojový web	Minimální počet výskytů
13 020	novinky.cz	3
10 864	idnes.cz	3
9 542	idnes.cz	5
8 184	novinky.cz	5

Tabulka 4 Porovnání slovníků pro prvních 100 000 komentářů

Po zvýšení minimálního počtu výskytů slov pro zařazení do slovníku na tři byl slovník komentářů iDNESu už jen o 17,5 % menší. Změna byla ještě znatelnější při zvýšení minimálního počtu výskytů na pět, poté byl slovník iDNESu dokonce o 16,5 % větší. Hypotézu o větší chybovosti na novinkách bych tedy nezamítal. Je možné, že kvalitnějším (a při zohledňování více výskytů i větším) slovníkem, alespoň co se týče komentářů od široké veřejnosti, je ten z komentářů portálu idnes.cz.

Po natrénování modelu tedy můžeme zkusit podobnosti a rozdílnosti mezi jednotlivými slovy. Modely fungují opravdu dobře. Z následujících obrázků je patrné, že diskutující na různých webových portálech mají nepatrně odlišné názory.

Na Obr. 5 v levé části (Novinky) se dá vypozorovat, že komentující nemají rádi Evropskou Unii a kohokoliv, co s ní má něco společného. Takové společenské atmosféry jsem si vědom i já z mých občasných návštěv právě těchto diskusí. Největší nepřítel „novinkářů“ jsou společně s EU Spojené státy americké. Naopak zde převládá proruská nálada.

To samé se dá říct o prvním českém prezidentovi Václavu Havlovi. Několik let zpět nebyl vůbec předmětem diskusí a všeobecně byl spíše považován za dobrého státníka. Ale v dotazech na slovník Novinek je vidět, že v posledních pěti letech, na něj diskutující nahlíží jako na jednu z hlavních záporných postav novodobé české historie, jakožto představitele „pražské kavárny“.

```
model_emb.most_similar('eurohujer')
```

```
[('patolizal', 0.8774008750915527),
 ('ritolezec', 0.8631651401519775),
 ('havlista', 0.834729015827179),
 ('zaprodanec', 0.8287860155105591),
 ('vitac', 0.8248715400695801),
 ('ritolez', 0.8230863213539124),
 ('probruselsky', 0.8178030252456665),
 ('mluvka', 0.8165966272354126),
 ('pritakavac', 0.8165704607963562),
 ('neomarxista', 0.8142543435096741)]
```

```
model_emb.most_similar('eurohujer')
```

```
[('neomarxista', 0.9094898700714111),
 ('populista', 0.907875120639801),
 ('ultralevicak', 0.8926825523376465),
 ('pravdolaskar', 0.891022801399231),
 ('vitac', 0.8825845718383789),
 ('fasoun', 0.8750708103179932),
 ('zemanovec', 0.8735561966896057),
 ('konzervativec', 0.8646520972251892),
 ('levicak', 0.8614441156387329),
 ('havlista', 0.8589779734611511)]
```

```
model_emb.most_similar('havloid')
```

```
[('kavarnik', 0.9212685227394104),
 ('slunickar', 0.9210271835327148),
 ('neomarxista', 0.8582838177680969),
 ('pravdolaskar', 0.8558661937713623),
 ('pomateny', 0.8552879691123962),
 ('havlista', 0.8536640405654907),
 ('nacek', 0.8531363010406494),
 ('fasoun', 0.85089111328125),
 ('trol', 0.8483622670173645),
 ('zemanovec', 0.8347148895263672)]
```

```
model_emb.most_similar('havloid')
```

```
[('pravdolaskar', 0.95391446352005),
 ('havlista', 0.947407603263855),
 ('kavarnik', 0.9080525040626526),
 ('kremlofil', 0.8929022550582886),
 ('neomarxista', 0.8899871110916138),
 ('kremlobot', 0.8871337175369263),
 ('zemanovec', 0.8869279623031616),
 ('islamofil', 0.8827241659164429),
 ('okamurovec', 0.8719762563705444),
 ('putinofil', 0.8702760934829712)]
```

```
model_emb.most_similar('brexit')
```

```
[('odchod', 0.7211025357246399),
 ('brexitu', 0.7126402854919434),
 ('piadolf', 0.6836544275283813),
 ('czexit', 0.6819798946380615),
 ('plebiscit', 0.6742558479309082),
 ('restart', 0.6731934547424316),
 ('eurozonu', 0.6702772378921509),
 ('ustup', 0.6622533202171326),
 ('vyjednavani', 0.6576657891273499),
 ('tentokrate', 0.654353141784668)]
```

```
model_emb.most_similar('brexit')
```

```
[('czexit', 0.7758542895317078),
 ('exit', 0.7628018856048584),
 ('odchod', 0.7287527322769165),
 ('debakl', 0.7223055362701416),
 ('impeachment', 0.7180121541023254),
 ('ukip', 0.7077677845954895),
 ('uspech', 0.7066375017166138),
 ('dt', 0.6900613307952881),
 ('propadak', 0.6839922666549683),
 ('farage', 0.6805858612060547)]
```

Obrázek 5 Novinky, iDNES 10 nejpodobnějších slov – politika

Na straně Novinek dále můžeme vidět, že diskutující berou odchod Velké Británie a případně i Česka z EU, jako restart státu, vyjednávání a spíše něco pozitivního. Kdežto diskutující na iDNESu spíše řeší, že odchod VB byl „debakl“, „propadák“ (to dokazuje např. klesání hodnoty libry po odhlasovaném odchodu z EU v celostátním referendu).

V dotazech na model iDNESu můžeme vidět, že Evropskou Unii nemají v lásce ani diskutující zde. Dále označení jedince jako „havloid“ je spíše mezi různými obecnějšími hanlivými označeními, kteří něco přehnaně adorují.

Obr. 6 zobrazuje porovnání mezi oběma weby pro slova „pivo“, „vegan“ a „pejskař“. Pivo je na obou webech podobné se slovy označující jídla a nápoje. Zbylé dva pojmy jsou na straně Novinek spíše neutrální, kdežto na portálu iDNES je vegan ve skupině s nadávkami a pejskař rovněž nepatří mezi kladná označení pro člověka se psem.

Ovšem tato hodnocení a hledání souvislostí s reálným světem je spíše práce pro experty z oborů politologie a sociologie, případně jazykové lingvisty, než pro mě, jakožto studenta informačních technologií.

```
model_emb.most_similar('pivo')
```

```
[('tocene', 0.8619036674499512),
 ('rizky', 0.8610831499099731),
 ('housky', 0.8598345518112183),
 ('pivecko', 0.8576186299324036),
 ('pecivo', 0.8564462661743164),
 ('rohliky', 0.8559372425079346),
 ('pivko', 0.8491687178611755),
 ('kafe', 0.8484054803848267),
 ('bramburky', 0.8422471284866333),
 ('knedliky', 0.8363829851150513)]
```

```
model_emb.most_similar('pivo')
```

```
[('pivko', 0.924048662185669),
 ('pivecko', 0.8710662126541138),
 ('rizky', 0.8695465326309204),
 ('smazak', 0.8679220080375671),
 ('lahvac', 0.8650273084640503),
 ('vinko', 0.8593454360961914),
 ('rizek', 0.8524378538131714),
 ('ciga', 0.8523666858673096),
 ('kofolu', 0.8464739918708801),
 ('cigo', 0.8454477787017822)]
```

```
model_emb.most_similar('vegan')
```

```
[('vegetarian', 0.8693679571151733),
 ('vodak', 0.7429549694061279),
 ('patok', 0.7380561828613281),
 ('budhista', 0.735357940196991),
 ('lunchmeat', 0.7341542840003967),
 ('kren', 0.734086275100708),
 ('zubr', 0.7326992750167847),
 ('cigos', 0.7287228107452393),
 ('vesnican', 0.7271274328231812),
 ('masochista', 0.7262815833091736)]
```

```
model_emb.most_similar('vegan')
```

```
[('vegetarian', 0.9174818992614746),
 ('homous', 0.8766340017318726),
 ('hulic', 0.8588755130767822),
 ('masozravec', 0.849952220916748),
 ('buzik', 0.8358224630355835),
 ('zavislak', 0.8243709206581116),
 ('pejskar', 0.8213536739349365),
 ('ekolog', 0.8211899995803833),
 ('heterak', 0.8182495832443237),
 ('puritan', 0.8143883347511292)]
```

```
model_emb.most_similar('pejskar')
```

```
[('myslivec', 0.7745527625083923),
 ('sobec', 0.7643674612045288),
 ('bezverec', 0.7607876658439636),
 ('germanec', 0.7594997882843018),
 ('neznaboh', 0.7527047395706177),
 ('vegetarian', 0.7473950386047363),
 ('abstinent', 0.7419548630714417),
 ('kurak', 0.7405494451522827),
 ('tezkoodenec', 0.7397149801254272),
 ('prazan', 0.7360270023345947)]
```

```
model_emb.most_similar('pejskar')
```

```
[('zavislak', 0.8408102989196777),
 ('hulic', 0.83954256772995),
 ('motorista', 0.8288330435752869),
 ('uchyl', 0.8276234269142151),
 ('heterak', 0.8224921822547913),
 ('vegan', 0.8213536739349365),
 ('cholerik', 0.8141686320304871),
 ('rybar', 0.8098354935646057),
 ('rvac', 0.8075230121612549),
 ('zahradkar', 0.8051025867462158)]
```

Obrázek 6 Novinky, iDNES 10 nejpodobnějších slov – typická česká

Na Obr. 7 je názorná ukázka, že *word2vec* model se naučí nejen podobnosti významu věcného, ale i gramatického. A proto při zadání přídavného jména predikuje přídavná jména a stejně tak v případě podstatných jmen. Dále, bez ukázky, funguje i co se týče vlastních jmen. Při zadání jména politiků vypíše jména dalších politiků a při zadání názvu města vypíše názvy dalších měst a zemí.

```
model_emb.most_similar('dreveny')
```

```
[('plechovy', 0.8622037172317505),
 ('skleneny', 0.8525114059448242),
 ('ocelovy', 0.8308625221252441),
 ('betonovy', 0.82524573802948),
 ('ruzovy', 0.8243383169174194),
 ('kovovy', 0.8121570944786072),
 ('dynamit', 0.8099435567855835),
 ('rozbity', 0.8062024116516113),
 ('papirovy', 0.804882287979126),
 ('gumovy', 0.8004394769668579)]
```

```
model_emb.most_similar('drevo')
```

```
[('zelezo', 0.9103676676750183),
 ('uhli', 0.8624215126037598),
 ('nabytek', 0.8357346057891846),
 ('beton', 0.8354306817054749),
 ('pily', 0.8352047204971313),
 ('obili', 0.824725329875946),
 ('pisek', 0.8217315673828125),
 ('cerpadlo', 0.8195743560791016),
 ('drivi', 0.8181615471839905),
 ('brikety', 0.8169382214546204)]
```

Obrázek 7 Srovnání podstatného a přídavného jména

Obr. 8 už jen dále potvrzuje výše zmíněné skutečnosti. Tato funkce vybere ze zadaných slov to, které se nejméně podobá ostatním zadaným slovům.

Pokud jsou čtyři slova, z nichž tři souvisí s vysokou školou a tři, které označují člověka, model určí, že je silnější podobnost mezi třemi lidmi než třemi pojmy z vysokoškolského prostředí, a vybere „univerzita“.

Jiný případ je, pokud se „maturant“ změní na „maturitu“, pak je pro model více podobnější vysokoškolské prostředí.

```
model_emb.doesnt_match(["profesor", "maturant", "univerzita", "student"])
```
```
'univerzita'
```
```
model_emb.doesnt_match(["profesor", "maturita", "univerzita", "student"])
```
```
'maturita'
```

Obrázek 8 Jaké slovo nepatří mezi ostatní

Na posledním ze série příkladů (Obr. 9) ukazuji, že „muž“ a „žena“ si nejsou vůbec podobní a vyskytují se ve velmi rozdílných kontextech. Za to „manžel“ a „manželka“ už mají podobnost vysokou. Zajímavé je, že právě tato dvě slova mají relativně vysokou podobnost s povoláními. Obdobně silné podobnosti mají i jiná slova, označující muže a ženu stejného povolání (např. lékař – lékařka).

```
model_emb.similarity("muz", "zena")
```
```
0.5621402
```
```
model_emb.similarity("manzel", "manzelka")
```
```
0.8391203
```

Obrázek 9 Rozdílnost mezi slovy

4.4 Klasifikace sentimentu

Klasifikace sentimentu je také rekurentní neuronová síť a relativně podobná dříve popisované znakové RNS. Hlavní změna je v abecedě. Při generování textů po znacích byla abecedou písmena a čísla, při této úloze je abecedou množina všech jednotlivých slov.

Nejprve načteme *word2vec* model, z kterého uděláme list slov seřazený podle abecedy a na index nula vložíme token *<PAD>*, který poslouží jako výplň před začátky kratších sekvencí.

Při klasifikaci sentimentu je výrazný rozdíl oproti generování textů po znacích, kde se index pro nový řádek mohl přidávat na konec řádků textu, jelikož index nula přerušil generování dalších znaků. Zde je tedy nutné kratší sekvence odsadit a nulami vyplnit začátek vektoru, jinak by síť po průchodu realných dat procházela a aktualizovala se na několika nulových informaci nenesoucích stavech, a zapomněla stavy předchozí. To by zapřičiňovalo, že by model predikoval krátké sekvence téměř náhodně.

List se slovy se použije pro záměnu všech slov v datovém souboru za indexy (při klasifikování sentimentu nemá smysl používat jiné části zpráv než právě komentáře diskutujících, které jsou ohodnoceny kladnými a zápornými hlasy). Tím dojde ke snížení složitosti trénování z $O(n^3)$ (Případně $O(n^2 \ log \ n)$ při vlastním naprogramování vyhledávání v listu pomocí půlení intervalu, jelikož Python při hledání objektu v listu prochází list celý od začátku) na $O(n^2)$ a to především díky tomu, že místo procházení *i* řádků o *j* slovech, kde se každému slovu musí najít index ve slovník o velikosti *k*, budou vstupní data mít *i* řádků o *j* indexech. Tím úplně odpadá *k*, jelikož přístup do pole má složitost $O(1)$.

Je nutné dát pozor, aby vstupní dataset byl stejný jako dataset, na kterém byl vytvořen slovník *word2vec*. Jinak by mohlo docházet k tomu, že plno slov ve slovníku nebude, a tudíž nebudou zohledňována při trénování modelu.

Následně utvoříme *tensory*, pro použití grafické karty. *Tensor* pro data bude mít rozměr [*i*, *j*], kde *i* je počet vstupních dat a *j* je zaokrouhlený součet průměrné délky a směrodatné odchylky komentářů (zkrácené o slova, která nedosáhla minimálního počtů výskytů je to 75 pro Novinky a 58 pro IDnes). Delší komentáře, než je maximální délka, jsou zkráceny, naopak kratší komentáře jsou připojeny na vektor nul délky – maximální délka mínus reálná délka komentáře. Není nutné, aby délka jednotlivých textů byla právě výše zmíněný zaokrouhlený součet. Maximální délka *tensoru* může být například délka nejdelšího textu, co do počtu slov. Právě tento součet se mi zdál jako vhodný kompromis mezi paměťovou náročností a zachování ~ 84 % komentářů nezkrácených.

Seřazenými *embedding* vektory dle slovníku se naplní *tensor* a tokenu *<PAD>* (první řádek) je přiřazen vektor nul. Ten se slouží jako váhy u *Embedding* vrstvy modelu. U celé této vrstvy můžeme vypnout učení. To je více než žádoucí, jelikož tato vrstva, pro slovník komentářů se slovy s minimálně deseti výskyty, obsahuje ~ 41 milionů parametrů. Navíc už je podobnost slov v těchto vahách relativně dobře zachycena a není potřeba vrstvu dále učit.

Dalšími vrstvami jsou stejně jako v předchozím případě *LSTM*, nebo *GRU* a poté lineární plně propojené vrstvy. Dopředný průchod se o něco liší, a to v tom, že můžeme nechat projít celý komentář *LSTM/GRU* buňkou jako vektor a použít pouze poslední výstup. To by, alespoň teoreticky, mělo návýšit rychlost průchodu.

Následuje trénovací část, která je v podstatě stejná jako u znakové RNS, jediný rozdíl je, že se nepřipravuje pro každý vstup *target label*, ale pouze pro ten poslední. Pokud tedy trénujeme na GPU je rychlost lehce vyšší, než v případě generování textů po znacích. Rozdíl v rychlostech by byl markantnější, pokud by řádky textu byly delší.

Porovnání rychlosti výpočtů, při testovací fázi se neaktualizují váhy modelu.

Popis	Fáze	Složitost	Rychlost
Data v listech na RAM, výpočty CPU	Trénink	$O(n^3)$	~ 0,2 it/s (5 s/it)
Data v listech na RAM, výpočty CPU	Trénink	$O(n^2)$	~ 5 it/s (25x zrychlení)
Data a výpočty v tensorech na GPU	Trénink	$O(n^2)$	15–28 it/s
Data a výpočty v tensorech na GPU	Testování	$O(n^2)$	50–130 it/s

Tabulka 5 Porovnání rychlostí tréninku RNS

Pořadové číslo	Webový portál	Min. výskytů slova	Míra učení	Počet iterací	Architektura
1	Novinky	10	0.001 > 0.00025	50 000	EMB50, LSTM200, FC500, FC1
2	Idnes	10	0.001 > 0.00025	50 000	EMB50, LSTM200, FC500, FC1
3	Novinky	10	0.001 > 0.00025	50 000	EMB50, LSTM200, FC1
4	Idnes	10	0.001 > 0.00025	50 000	EMB50, LSTM200, FC1
5	Novinky	10	0.001 > 0.00025	50 000	EMB50, 2LSTM200, FC500, FC1
6	Novinky	10	0.0001 > 0.000025	50 000	EMB50, LSTM200, FC500, FC1
7	Novinky	10	0.00001 > 0.0000025	25 000	EMB50, LSTM200, FC500, FC1
8	Idnes	10	0.001 > 0.00025	25 000	EMB50, LSTM256, FC256, FC128, FC1
9	Idnes	8	0.0005 > 0.0000625	200 000	EMB128, LSTM256, FC512, FC1

Tabulka 6 Trénované modely

Pořadové číslo	Přesnost	Preciznost	Citlivost	F1 skóre
1	71.17	74.18	65.24	69.42
2	54.27	51.05	87.60	56.66
3	70.90	74.46	64.75	69.26
4	50.21	50.26	90.81	64.71
5	70.38	73.51	64.37	68.64
6	66.94	68.63	61.75	65.01
7	54.31	59.11	31.71	41.28
8	52.52	51.62	89.30	65.42
9	61.03	60.37	64.74	62.48

Tabulka 7 Metriky natrénovaných modelů dostupné v Tabulka 6 Trénované modely

Síť se ve většině případů naučí lépe predikovat kladné komentáře.

Ve výsledcích je určitě prostor pro zlepšení, pokud by byla příležitost modely učit na výkonných jinak nepoužívaných PC v řádech dní.

Pro porovnání, zda budou stejné komentáře přijaty stejně na různých portálech, jsem vybral nejúspěšnější modely (1 a 9 dle Tab. 6 a Tab. 7).

I když jejich úspěšnosti predikce nejsou příliš vysoké, a výsledky by mohly být nevypovídající, přesto se modely 1 a 9 často shodly na sentimentu komentáře. Model 1 s přesností ~ 71 % a model 9 s přesností ~ 61 % se shodovaly v predikci v ~ 80 % případů.

Dle mého názoru i přes nepřesnosti v predikcích sentimentu je zde vidět náznak trendu, že komentáře budou přijaty na obou portálech relativně podobně.

4.5 Generování textu po slovech

Při generování po slovech je vhodné si předem připravit dataset z textů na jejich indexy ze slovníku, stejně jako v případě klasifikátoru sentimentu. To sníží složitost učícího algoritmu z kubické na kvadratickou.

Skript generování textů po slovech začíná načtením vstupního datasetu (nejlépe indexů slov) a předem natrénovaného modelu *word2vec*. Je nutné dbát na to, aby byl *word2vec* model trénovaný na stejném datasetu (nebo podobném) jako budou vstupní data pro síť, jinak se stane, že se mnoho slov ze vstupního datasetu vyřadí a při trénování se nebudou zohledňovat.

Klasicky také uděláme z *word2vec* slovníku seřazený list, jen na nultou pozici nepřijde token *<PAD>*, ale token *\n*. Dalším krokem je vytvořit histogram začínajících slov datasetu a vydělit počtem položek vstupních dat, tím vzinkne matice pravděpodobností. Ta bude ovšem velmi řídká, avšak je zde i slovo „To“, kterým začíná ~ 3,05 % všech komentářů na Novinkách. Seřazené vektory předáme do velkého *tensoru*, kterým nainicalizujeme váhy *embedding* vrstvy rekurentní neuronové sítě. Do sítě také nadefinujeme *LSTM* a lineární vrstvu.

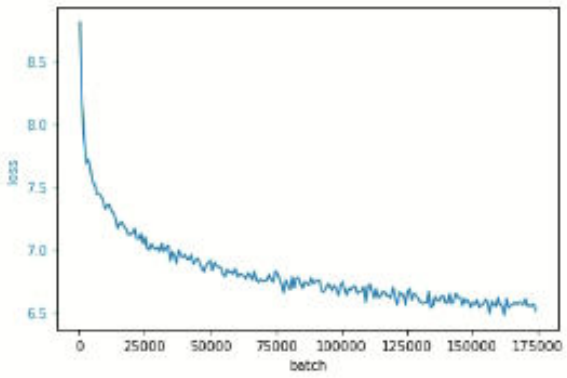
Dopředný průchod je v podstatě totožný jako u generování textů znakovou RNS s tím rozdílem, že zde má abeceda (slovník indexů a slov) rozměry v řádech statisíců slov místo desítek znaků.

Funkce *sample* také pracuje stejně jako u znakové RNS. Rozdílem je pouze mnohem větší množina možností a také nutnost za každé vygenerované slovo napojit mezeru.

Optimizérem je také *Adam*, pouze se sníženou mírou učení na polovinu. Pravděpodobnosti se počítají *softmaxem*, *loss* funkcí zůstává *Cross entropy loss*.

Tréninková funkce obdobná jako u předchozích modelů, obohocená o odchycování výjimek *ValueError*, kdy slovo není ve slovníku (pokud vstupním datasetem již nejsou vyfiltrované indexy). To nastává, pokud používáme slovník slov s minimálním počtem výskytů větším než jedna.

Na následujícím obrázku (Obr. 10) je možné pozorovat graf hodnot chyby. Při porovnání s grafem (Obr. 4) hodnot chyby rekurentní neuronové sítě, která se učila generovat texty po znacích, lze vypozorovat, že při stejné architektuře je hodnota *lossu* pro slovní RNS výrazně vyšší. Zatímco u znakové sítě klesala k hodnotě 1,8, tak slovní síť měla problém dostat hodnotu chyby pod 6,5. Toto je zcela jistě způsobeno rozdílným počtem možností, které síť může predikovat, kde abeceda slovní sítě má desetitisíce krát více možností. Přesto má křivka relativně podobný průběh.



Obrázek 10 Loss hodnoty při učení RNS po slovech

Jako inicializační text na Obr. 11 a Obr 12. bylo použito vždy první slovo v řádku.

```
prezident byl na svobode
zeman se stal sefem soudu za skodu
cesky trh se bude v zari i na svete
americky prezident byl v prvnim zari na slovensku
babis a ods chce zakazat volby v brne
policie zadrzela muze ktery se u soudu museli ozivovat
poslanecka konci
ano v zahranici a cssd se o ne
```

Obrázek 11 Generování novinářských titulků

```
prezident Miloš Zeman je na tom co je to za každé peníze
prezident Zeman se stáva způsobem že je to už od pana prezidenta a jeho parta
český narod a také si myslí že to že jsou to že se blíží
Babiš je na sjezdu
Babiš a Okamura má pravdu
americký rybníček
poslanecká republika byl tak jako je sice v politice a to je snad ne
poslanecká práva
ano skončí jako vždy řekl Babiš ČSSD a ten jsou fakt s prezidentem
paní vy jste vám přeju
paní Nováková je ta sebranka
to nevidím
to je vážně dobrý
to není ani náhodou a to je vidět
to bylo na babišovu ale pak to je zas za každou cenu
to je ubohost
```

Obrázek 12 Generování komentářů dle komentujících na Novinkách

Na předchozích obrázcích (Obr. 11 a Obr. 12) jsem vybral několik smysluplnějších výsledků generování slovní RNS. Pro slovo „to“ jsou výsledky častěji smysluplnější než pro jiná slova, je to pravděpodobně způsobeno tím, že na toto slovo začíná přes 3 % všech komentářů a je tak nejlépe zachyceno ve vahách modelu.

Výše uvedené vygenerované komentáře vypadají vcelku věrohodně. Věřím, že model by šlo natrénovat tak, aby ze zadaného titulku, či textu článku dokázal generovat co nejkladnější či nejzápornější komentáře. Toho by se dalo zneužít k propagaci různých názorů, či případně k vytvoření automatického internetového „trolla“, který by naopak psal komentáře, které by se diskutujícím nezamlouvaly.

5 Závěr

Cílem mé bakalářské práce bylo porovnání dvou online periodik iDNES a Novinky za pomoci různých architektur neuronových sítí. Jednalo se o klasifikaci sentimentu a generování textů po znacích nebo po slovech.

V teoretické části jsem nastínil podstatu fungování rekurentních neuronových sítí. Popsal jsem základní rozlišení způsobu učení – učení bez učitele a s učitelem. Dále jsem charakterizoval tři architektury. Klasickou RNS; *LSTM*, jež dosahuje jedněch z nejkvalitnějších výsledků a pochopitelně je tedy velmi populární, a z ní vycházející *GRU*, která je novější, jednodušší, a přesto dosahuje velmi dobrých výsledků. Objasnil jsem princip fungování rekurentních síti na základě hledání minima chybové funkce a následné zpětné propagace – aktualizací parametrů sítě. Popsal jsem čtyři základní metriky pro porovnávání úspěšnosti u klasifikačních modelů, jimiž jsou správnost, úplnost, preciznost a F1 skóre. Ve zkratce jsem napsal o již existujících projektech zabývajících se touto problematikou.

Praktickou částí bylo vytvoření souborů z databází článků a příslušných komentářů obou webů. Ty jsem následně aplikoval jako vstupní data pro trénink generování textů po jednotlivých znacích. Dále jsem vytvořil pomocí volně dostupných knihoven slovník slov a jejich vektorů, kterými se dají porovnávat a sledovat jejich podobnosti, jak významu lexikálního, tak mluvnického. Tyto slovníky byly základem pro modely klasifikující sentiment. Pomocí těchto modelů jsem se snažil prokázat hypotézu, že stejné komentáře budou na různých diskusních fórech přijaty jinak. To se mi podařilo jen částečně, jelikož mé modely nedosahovaly potřebné úspěšnosti, aby byla hladina významnosti přijatelně nízká. Další funkcí těchto slovníků byl základ pro váhy modelu generující text po celých slovech.

Přínos práce vidím jako základ pro rozsáhlou analýzu (doba sběru dat byla pět let) veřejného mínění a významů slov používaných širokou veřejností. Dle mého názoru práce nemá podstatný přínos pro obor informačních technologií, nicméně je zde potenciál pro různé další vědní obory, jako jsou lingvistika, marketing, sociologie a politologie.

Možným pokračováním vidím další způsoby generování textů. Ať už je to generování obsahu článku nebo komentářů podle nadpisu, nebo generování komentářů dle obsahu článků. Dále by bylo možné rozvíjet mnou započaté generování co nejkladněji a nejzáporněji hodnocených komentářů. Další možností rozvoje by mohlo být delší trénování, či použití výkonnějších počítačů, z důvodu enormně velkých dat. Pro trénink rekurentních neuronových sítí jsou v oboru často používána data v řádech megabytů, kdežto mně poskytnutá data byla v řádech gigabytů. Jejich trénink při klasifikaci sentimentu je v řádech hodin až dní, zatímco generování jak po znacích, tak po slovech pro takto rozměrný dataset může dosahovat až řádů týdnů.

Literatura

[1] D. Soni, „Supervised vs. Unsupervised Learning,“ 22 Březen 2018. [Online]. Available: https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d. [Přístup získán 8 Srpen 2020].

[2] A. Shekhar, „Understanding The Recurrent Neural Network,“ 14 4 2018. [Online]. Available: https://medium.com/mindorks/understanding-the-recurrent-neural-network-44d593f112a2. [Přístup získán 23 5 2020].

[3] Supervise.ly, „[Lecture] Evolution: from vanilla RNN to GRU & LSTMs,“ 21 8 2017. [Online]. Available: https://towardsdatascience.com/lecture-evolution-from-vanilla-rnn-to-gru-lstms-58688f1da83a. [Přístup získán 23 5 2020].

[4] J. Brownlee, „Gentle Introduction to the Adam Optimization Algorithm for Deep Learning,“ 3 Červenec 2017. [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. [Přístup získán 9 Srpen 2020].

[5] R. Joshi, „Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures,“ 9 9 2016. [Online]. Available: https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/. [Přístup získán 28 5 2020].

[6] I. S. R. J. J. C. G. B. Alec Radford, „Unsupervised Sentiment Neuron,“ Open AI, 6 Duben 2017. [Online]. Available: https://openai.com/blog/unsupervised-sentiment-neuron/. [Přístup získán 8 Srpen 2020].

[7] J. W. D. A. D. A. J. C. M. B. I. S. Alec Radford, „Better Language Models and Their Implications,“ Open AI, 14 Únor 2019. [Online]. Available: https://openai.com/blog/better-language-models/. [Přístup získán 31 Srpen 2020].

[8] A. Karpathy, „karpathy/char-rnn: Multi-layer Recurrent Neural Networks (LSTM, GRU, RNN) for character-level language models in Torch,“ 30 4 2016. [Online]. Available: https://github.com/karpathy/char-rnn. [Přístup získán 26 5 2020].