Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Použití umělých neuronových sítí v Javě pro úlohu rozpoznání falešných zpráv

BAKALÁŘSKÁ PRÁCE

Studijní program: Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Autor: Marek Vávra

Vedoucí bakalářské práce: doc. Ing. Ondřej Zamazal, Ph.D.

Praha, květen 2020

Prohlášení

Prohlašuji, že jsem bakalářskou práci „Použití umělých neuronových sítí v Javě pro úlohu rozpoznání falešných zpráv“ vypracoval samostatně za použití v práci uvedených pramenů a literatury.

V Praze dne 11. května 2020 Marek Vávra

Poděkování

Děkuji panu doc. Ing. Ondřej Zamazal, Ph.D. za cenné rady, trpělivost a čas, který strávil vedením této práce.

Abstrakt

Umělé neuronové sítě se v poslední době dostávají do popředí zájmu v oblasti strojového učení. Používání konvolučních a rekurentních architektur umožnuje jejich efektivní použití v různých úlohách dolování z dat. Jednou z úloh je i klasifikace falešných zpráv, jejichž prudký nárůst v posledních letech vedl k nutnosti nalezení nových metod, jak je klasifikovat a zabránit jejich šíření. Touto oblastí se zaobírá i tato práce a má několik cílů. V teoretické rovině jsou jimi představení oblasti dolování z dat (data mining), představení umělých (hlubokých) neuronových sítí obecně a v prostředí Javy a představení aplikační oblasti. Tyto cíle jsou řešeny rešerší související literatury. V teoretické části práce je nejdříve vymezena oblast dolování z dat, následují umělé neuronové sítě a jejich implementace v Javě. V závěru teoretické části je představena aplikační oblast falešných zpráv. Praktickými cíli práce jsou navrhnout a realizovat model umělé neuronové sítě v Javě pro klasifikační úlohu z aplikační oblasti falešných zpráv a porovnat a vyhodnotit dosažené výsledky oproti dalším vybraným metodám z dobývání znalostí z databází. Realizovány byly modely využívající konvoluční a rekurentní architektury. Pro implementaci byla použita knihovna Deeplearning4j v programovacím jazyce Java. Praktická část se zaobírá nutnými kroky předzpracování dat pro použití v neuronových sítích. Navazuje návrh a realizace použitých modelů pro klasifikaci falešných zpráv. Jedná se o 3 různé rekurentní modely a 2 konvoluční. Na závěr jsou výsledky klasifikace porovnány s výsledky vybraných metod dobývání znalostí z databází. Pro tuto klasifikační úlohy dosahovaly lepších výsledků rekurentní modely s úspěšností 75%, konvoluční modely dosáhly úspěšnosti klasifikace 68%.

Klíčová slova

Deeplearning4j, falešné zprávy, klasifikace textu, konvoluční neuronová síť, rekurentní neuronová síť, umělé neuronové sítě.

JEL klasifikace

C88

Abstract

In recent years, artificial neural networks established themselves as a key component of machine learning. Recurrent and convolutional architectures enabled their usage for wide variety of problem. Classification of fake news is one of those problems and a theme of this work. Five different goal were set up for this thesis, three theoretical and two practical. Theoretical goals are to introduce field of data mining, to introduce deep neural networks and their implementation in Java environment and to introduce fake news. This is done by research of related literature. The theoretical part of the thesis first defines the area of data mining, followed by artificial neural networks and their implementation in Java. At the end of the theoretical part, the application area of false news is presented. Practical goals are to design and implement model of artificial neural network able to classify fake news, implemented in Java and to compare results with result of other selected methods used in data mining. The practical part of this thesis first deals with the necessary steps of data pre-processing enabling use in neural networks. Then models using convolutional and recurrent architectures are presented. In total there are five models, three recurrent and two convolutional. In the end of the thesis, result of these models are compared to results of other selected methods. For this classification task, recurrent models achieved better results with a success rate of 75%, convolution models achieved a success rate of 68%.

Keywords

Artificial neural networks, Deeplearning4j, fake news, text classification, convolutional neural network,

JEL Classification

C88

Obsah

Seznam obrázků

Úvod

Umělé neuronové sítě jsou stále populárnější a v popředí zájmu akademiků i praktiků strojového učení. Kromě počítačového vidění, kde slaví největší úspěch, se postupně aplikují i na další oblasti se zajímavými výsledky. Jednou z těchto oblastí je i práce s přirozeným jazykem, kde jsou publikovány novinové články psané umělou inteligencí či napodobováni slavní autoři klasické literatury. Umělé neuronové sítě se ale používají i na rozpoznávání závislostí v textu či na klasifikační úlohy, kde můžeme sledovat nárust zájmu o klasifikaci pomocí konvolučních neuronových sítí, které používají podobné metody pro klasifikaci textů jako jsou používané pro klasifikaci obrázků.

Jednou z oblastí, kde se vyskytla potřeba klasifikovat texty, jsou falešné zprávy. Ty můžeme označit za fenomén posledních let, k prudkému nárustu o ně došlo po amerických prezidentských volbách v roce 2016. Nadužívání a politizace termínu falešné zprávy v poslední době vedlo až k rozhodnutí odborných organizací, například UNESCA, k doporučení používat jiné názvy k jejich označování. Šíření falešných zpráv se také často dává do souvislosti se sociálními sítěmi jako je Facebook či Twitter. Tyto společnosti byly nuceny na tento fenomén reagovat a zavádějí opatření k potlačení šíření falešných zpráv. Nutnou podmínkou úspěchu takovýchto opatření je schopnost falešné zprávy správně identifikovat.

V souvislosti s růstem zájmu o neuronové sítě se objevují stále nové softwarové prostředky umožňující jejich použití. Hlavní programovací jazyky již mají knihovny dedikované pro práci s neuronovými sítěmi. Nejpoužívanější v této oblasti je programovací jazyk Python, následovaný C/C++, Javou, R a Javascriptem. (1)

Prvním teoretickým cíle této práce je představení oblasti dolování z dat (data mining), následuje cíl představení umělých (hlubokých) neuronových sítí obecně a v prostředí Javy. Posledním teoretickým cílem je představení aplikační oblasti falešných zpráv. Tyto teoretické cíle byly řešeny rešerší související literatury.

Prvním praktickým cílem bakalářské práce bylo navrhnout a realizovat model umělé neuronové sítě v Javě pro klasifikační úlohu z aplikační oblasti falešných zpráv. Pro tuto klasifikační úlohy byly vybrány dvě odlišné architektury, konvoluční a rekurentní neuronová síť. Pro jejich návrh a realizaci byl použit programovací jazyk Java, konkrétně knihovna Deeplearning4j. Druhým praktickým cílem bylo porovnat a vyhodnotit dosažené výsledky oproti dalším vybraným metodám z dobývání znalostí z databází. Porovnání proběhlo oproti dvěma tradičním metodám, klasifikaci rozhodovacími stromy a naivním bayesem.

Práce je strukturovaná do teoretické a praktické části, V teoretické části je nejdříve představena oblast dolování z dat. Následuje představení umělých neuronových sítí a možnosti jejich implementace v Javě pomocí knihovny Deeplearning4j. Na závěr teoretické části je představena aplikační oblast falešných zpráv. V praktické části jsou ukázány

navržené a realizované modely, jejich výsledky a porovnání úspěšnosti klasifikace s dalšími vybranými metodami dobývání znalostí z databází.

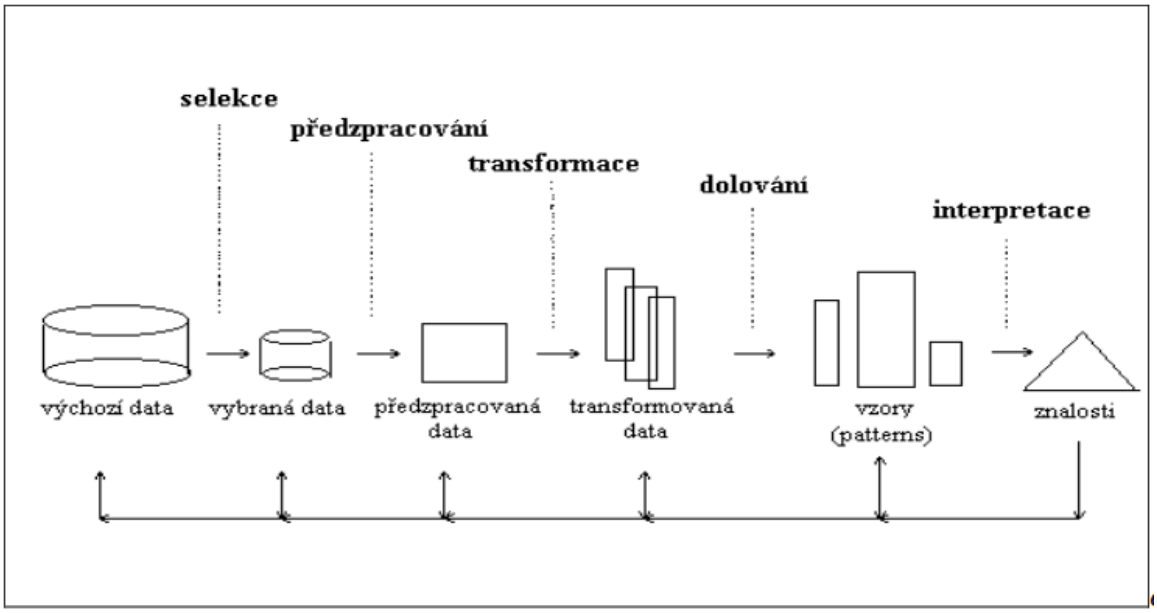
1 Dolování z dat

1.1 Dolování z dat v kontextu dobývání znalostí z databází

Dobývání znalostí z databází je koncept, který se začal používat v 90. letech v souvislosti s rozvojem databázových technologií. Zvětšující se objem dat ukládaných v databázích a potřeba tyto data využívat pro podporu rozhodování firem vedly k propojení metod a principů používaných v několika oborech. Znalosti z oborů jako statistika, databáze, strojové učení, umělá inteligence ve spojení s výkonnými výpočetní prostředky daly vzniknout procesu dnes známému jako dobývání znalostí z databází. Ten můžeme definovat jako:

„netriviální extrakci implicitních, dříve neznámých a potenciálně užitečných informací z dat“ (2)

Celý proces se dá rozdělit do několika kroků. Grafické znázornění těchto kroků můžeme vidět na Obr. 1



Obr. 1 Proces dobývání znalostí z databází (2)

Prvním krokem je **selekce** dat, kde je nutné projít všechna dostupná data a vybrat z nich ta, která jsou relevantní k problému, který se snažíme řešit.

Druhým krokem je **předzpracování** dat, kde je třeba se vypořádat s problémem, že ne všechna data jsou kvalitní a odpovídají definovanému formátu. Je tedy třeba data vyčistit, což může zahrnovat vyřazení odlehlých hodnot či šumu, vypořádání se s daty s chybějícími údaji (ať už jejich vyřazení či doplnění).

Cílem **transformace** dat je připravit data pro použití vybranou metodou. Může zahrnovat například vyřazení některých atributů s cílem zmenšit objem dat a tím snížit výpočetní náročnost.

Následuje samotné **dolování** z dat, kde jsou vybrané analytické metody použity. Cílem je za pomoci těchto metod najít zajímavé vztahy v datech. Metody se ladí na daný problém jejich opakovaným použitím a upravování na základě předchozích výsledků.

Posledním krokem je **interpretace** dosažených výsledků. Některé výsledky mohou být nezajímavé či samozřejmé a je lepší je neuvádět. Zajímavé nálezy je potřeba vyjádřit srozumitelným způsobem lidem, kteří s nimi budou dále pracovat. (2,3)

S pojmem dolování dat ještě úzce souvisí **strojové učení**. Oproti dolování z dat, kde je cílem najít zajímavé vztahy v datech, je cílem strojového učení je navrhnout algoritmus, který se učí z dat.

Učení probíhá s ohledem na určitou úlohu, (více v podkapitole Typy úloh dolování z dat), a výkonnostní metriku (těmi se blíže zabývá podkapitola Evaluace modelů). Výstupem je natrénovaný model, který je schopen dělat predikce na datech. Tyto modely umožnují zvládnout problémy, které jsou příliš složité na řešení explicitně napsanými programy. (4,5)

1.2 Typy úloh dolování z dat

Při dolování z dat můžeme narazit na různé typu problémů, které vyžadují použití odlišných metod. Rozlišujeme zde několik základních typů úloh, které se často vyskytují.

První z nich je klasifikace. Cílem klasifikace je zařadit data do správné třídy. Cílem procesu dolování z dat je tedy najít funkci, která je schopná zařadit vstup do správné třídy, případně určit pravděpodobnost, že daný vstup patří do dané třídy. Příkladem může být zařazení obrázku psa algoritmem pro rozpoznávání obrázků do třídy “zvíře“.

Dalším typem úlohy je regrese. Regrese je podobná klasifikaci, ale jejím výstupem je numerická hodnota. Příkladem problému, který tato metoda řeší může být předpověď, kolik bude stát ropa či jiná komodita po uplynutí určitého časového období.

Pokud je třeba roztřídit data do skupin, které nemáme předem definované, používá se metoda známá jako shlukování. Zde je cílem nalézt funkci, která rozdělí data do několika skupin (shluků) tak, aby si objekty náležející do stejného shluku byly podobnější než objekty ostatních shluků.

Úloh existuje více, jako další můžeme zmínit například hledání anomálií v datech. (3,5)

1.3 Evaluace modelů

Způsoby vyhodnocování modelů se liší na základě typu řešené úlohy, tato kapitola se zabývá hlavně prostředky pro vyhodnocování modelů pro potřeby klasifikace. Vycházíme tedy

z myšlenky, že máme určité množství objektů a několik kategorií, do kterých tyto objekty spadají. Metody vyhodnocení jsou poté založeny na porovnání klasifikace našeho modelu a správného výsledku.

Testovat můžeme na různých datech. První variantou je testování na trénovacích datech. Tato metoda má nízkou vypovídající hodnotu o kvalitě modelu, jelikož zde snadno může dojít k přeučení, kdy model klasifikuje podle náhodných charakteristik v datech, která se nedají generalizovat. Pokud máme dostatek dat, můžeme část oddělit a použít pro testování. Zbavujeme se tím ale části dat, která bychom jinak mohli použít pro lepší trénování modelu.

Jednou z používaných metod, jak netestovat na trénovacích datech, ale zároveň použít všechna data pro trénovaní modelu je **křížová validace**. Je založena na rozdělení dat na několik částí. Pro proces trénování se vždy jedna část oddělí a trénuje se na zbylých. Na oddělené části proběhne testování modelu. Celý proces se opakuje tolikrát, na kolik částí jsme data rozdělily. Výsledkem testování je zprůměrovaný výsledek z každé části. (2)

Cílem testování je tedy určit, v kolika případech model úspěšně zařadil objekt do správné kategorie. Jednou z používaných metod pro zobrazení těchto výsledků je **matice záměn.** Příklad pro dvě kategorie + a – můžeme vidět na Obr. 2

TP (true positive) – počet příkladů správně klasifikovaných jako třída +

TN (true negative) - počet příkladů správně klasifikovaných jako třída -

FP (false positove) – počet příkladů chybně zařazených do třídy +

FN (false negative) - počet příkladů chybně zařazených do třídy -

	Klasifikace systémem	
Správné zařazení	+	-
+	TP	FN
-	FP	TN

Obr. 2 Matice záměn (2)

Metrika pro vyjádření úspěšnosti klasifikace je **správnost**. Ta vyjadřuje poměr správně klasifikovaných objektů ku všem klasifikovaným objektům.

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

Správnost se nehodí pro případy, kde jsou jednotlivé kategorie zastoupeny nerovnoměrně, v těchto případech bychom dostali značně zkreslený výsledek. Pro takové případy je lepší sledovat správnost pro jednotlivé třídy.

Správnost pro třídu +

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

Správnost pro třídu -

$$Acc = \frac{TN}{TN + FN}$$

Dalšími z používaných metrik jsou přesnost a úplnost. Ty se používají především při vyhledávání dokumentů. **Přesnost** nám vyjadřuje pravděpodobnost, že daný dokument je relevantní.

$$Přesnost = \frac{TP}{TP + FP}$$

Úplnost vyjadřuje, kolik dokumentů týkající se tématu jsme nalezli. (2)

$$Úplnost = \frac{TN}{TN + FN}$$

1.4 Vybrané analytické metody pro dolování z dat

Analytických metod pro dolování z dat existuje mnoho, pro tuto práci byly vybrány dvě, které jsou v praktické části použity pro porovnání s neuronovými sítěmi. Analytická metoda neuronových sítí je teoreticky vysvětlena ve dvou samostatných kapitolách, těmi jsou Neuronové sítě a Hluboké neuronové sítě.

1.4.1 Rozhodovací Stromy

Jedna z nejznámějších metod strojového učení. Důvodů pro jejich použití je několik, jsou snadno interpretovatelné a přehledné, což umožnuje uživatelům jednoduše vyhodnocovat získané výsledky. Popularitě napomáhá i možnost vizualizace výsledků do stromové struktury, která je známá z jiných oblastí. (6)

Pokud chceme vytvořit rozhodovací strom, postupujeme metodou “rozděl a panuj“. Začínáme u kořene stromu a data postupně rozdělujeme tak, aby v daném uzlu (datové podmnožině) převládaly data jedné třídy. Pro určení atributu (A), podle kterého je nejvhodnější v daném kroku dělit, se používá například informační zisk. Ten udává očekávané snížení entropie (neurčitosti), pokud bychom data rozdělili dle tohoto atributu. Spočítá se podle vzorce:

$$Zisk(A) = H(C) - H(A)$$

Kde H(C) je entropie pro celá data a H(A) je entropie pro daný atribut. Entropie se spočítá podle vzorce:

$$H(C) = -\sum_{t=1}^{T} \frac{n_t}{n} \, log_2 \frac{n_t}{n}$$

Kde T značí počet tříd, n_t pak počet výskytů pro danou třídu.

Pro další dělení tedy hledáme atribut s maximální hodnotou informačního zisku. Informační zisk ovšem nebere v potaz počet hodnot zvoleného atributu, což může snadno vést k přeučení. Proto se spíše používá poměrný informační zisk, který bere do úvahy i počet hodnot atributu. (2)

1.4.2 Naivní bayesovský klasifikátor

Naivní bayesovský klasifikátor je postaven na předpokladu podmíněné nezávislosti evidencí, která bývá v reálných úlohách málokdy splněna (proto je nazýván naivní). Přesto dosahuje dobrých výsledů při klasifikaci do více tříd, je velmi rychlý a jednoduchý. Často se proto používá jako jeden z prvních klasifikátorů na danou datovou sadu. Vysoké úspěšnosti dosahuje při úlohách klasifikace textů do několika tříd, běžně se používá pro analýzu sentimentu či určení spamu. Z těchto důvodů je použit i v praktické části této práce. (7)

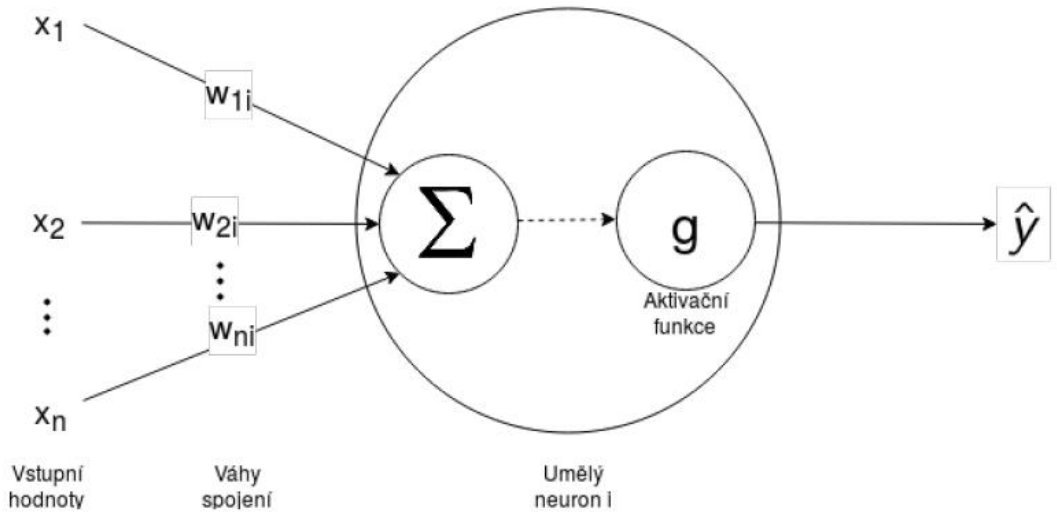
2 Neuronové sítě

2.1 Umělý neuron

Základem matematického modelu neuronové sítě je umělý neuron. Ten má obecně n vstupů, které tvoří vstupní vektor x = (x1,..,xn). Každý vstup je ohodnocen váhou, dostaneme tedy vektor vah w= (w1,...,wn). Suma násobků vstupu s přiřazenou váhou nám pak dá vážený součet vstupů neuronu (SUM), tedy

$$SUM = \sum_{i=1}^{n} w_i \, x_i$$

První umělé neurony fungovaly na principu, že pokud tento vážený součet přesáhne prahovou hodnotu, kterou má neuron danou, pak se neuron aktivuje a výstup z neuronu bude 1. V případě, že součet bude menší než prahová hodnota aktivační funkce, výstup bude 0. Postupem času se způsob transformace vstupů na výstupy měnil, dnes se již používají jiné způsoby. Více v kapitole aktivační funkce.

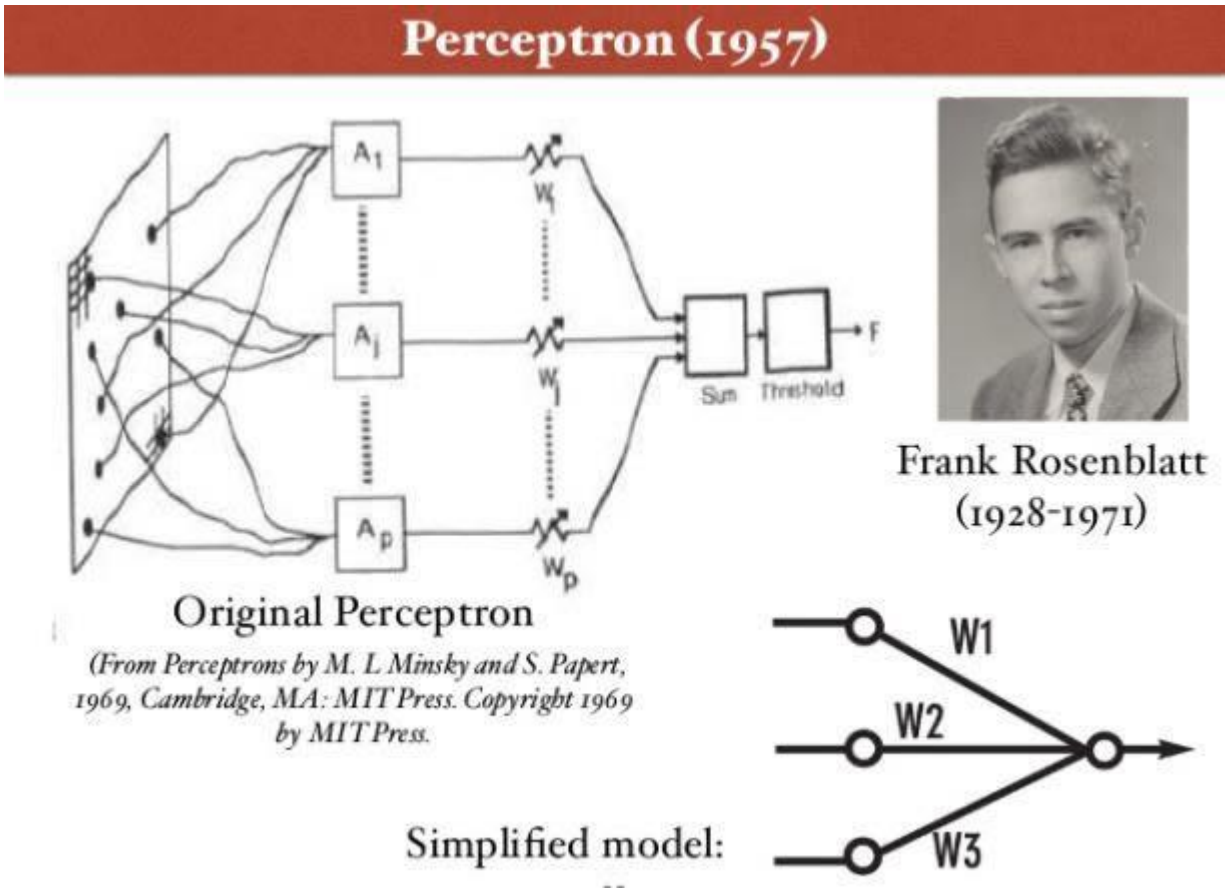


Obr. 3 Model jednoho neuronu (Převzato z prezentace Zamazal O., předmět 4IZ231, VŠE)

Prvním umělým neuron byl takzvaný logický neuron z roku 1943. Tento neuron pracoval pouze s hodnotami 0 a 1, a to jak na vstupu, tak na výstupu. Důležitý byl tento neuron proto, že jeho autoři ukázali, že tyto logické neurony jsou schopné jednoznačně reprezentovat základní spojky matematické logiky. (2,8)

2.2 Perceptron

Autorem perceptronu je Frank Rosenblatt, který ji v roce 1957 návrh jako model zrakové soustavy (viz Obr. 4). Sám autor perceptron zamýšlel jako model mozku.



Obr. 4 Rosenblattův perceptron (9)

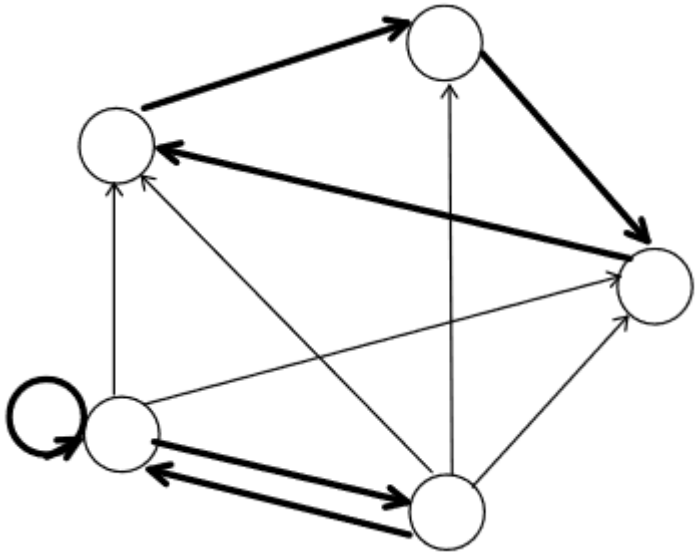
Perceptron si můžeme představit jako sytém tvořený třemi úrovněmi. První z nich je tvořena receptory, což jsou prvky, jejichž výstup nabývá hodnot 0 nebo 1 v závislosti na tom, zda jsou excitovány. Druhá vrstva je tvořena asociativními elementy. Vstupy do těchto elementů jsou signály z receptorů, každý vstup zde má pevně danou váhu 1 či -1. Pokud součet vstupů překročí práh, asociativní element se aktivuje. Výstupy z asociativních elementů jsou napojeny na třetí vrstvu, která je tvořena reagujícími elementy. Jejich počet odpovídá počtu tříd, do kterých klasifikujeme. Tyto elementy provedou vážený součet a v poslední vrstvě, takzvané vrstvě maxima se z nich vybere ten reagující element, který má nejvyšší výstup.

Problémem Perceptronu je, že je jeho pomocí možné řešit pouze lineárně separovatelné problémy. Tento problém je možné řešit pomocí kombinací více vrstev dohromady, čímž se zabývá další kapitola. (2)

2.3 Složitější neuronové sítě

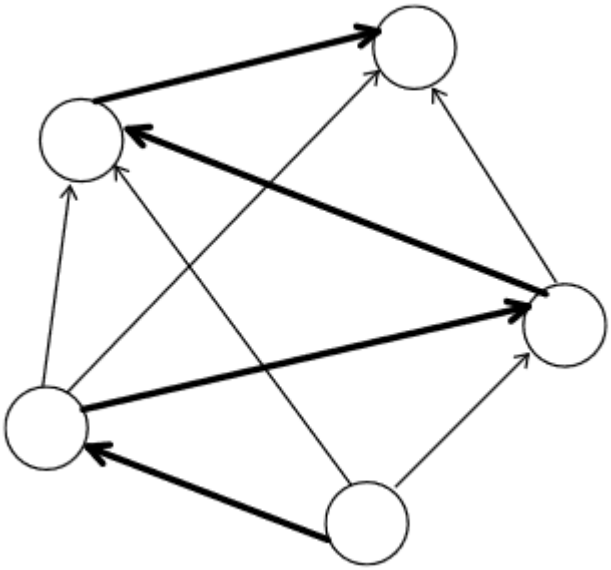
Pro řešení složitějších úloh, než jsou lineárně separovatelné problémy, se používají neuronové sítě, které se skládají z více vrstev neuronů. Rozlišujeme dva základní typy architektury, síť cyklickou a acyklickou.

Cyklická síť je síť, v které existují neurony, mezi kterými nalezneme cyklus. Příklad této sítě můžeme vidět na Obr. 5, kde jsou vyznačeny tři různé cykly. Do tohoto cyklu může být zapojeno libovolné množství neuronů, nejjednodušším příkladem cyklu je zpětná vazba, kdy signál na výstupu neuronu je znovu použit jako vstup. Příkladem cyklických sítí jsou rekurentní sítě.



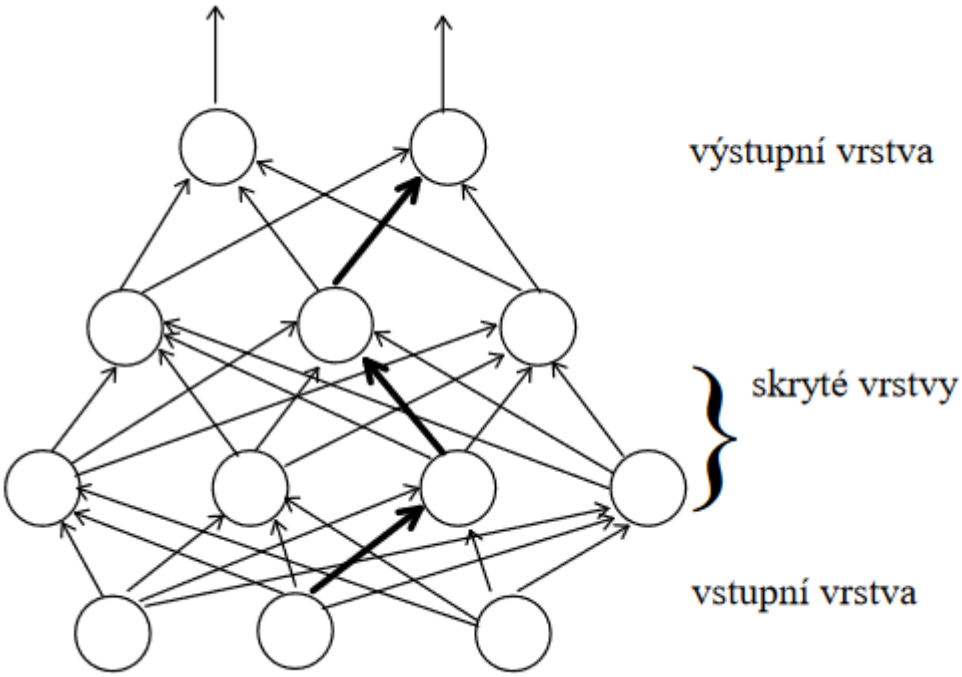
Obr. 5 Cyklická síť (10)

Acyklická síť je síť, kde naopak cyklus neexistuje. Příklad můžeme vidět na Obr. 6. Neurony v těchto sítích můžeme rozdělit do vrstev tak, že výstupy z jedné vrstvy vedou jen do vrstev vyšších (můžou některé vrstvy přeskočit).



Obr. 6 Acyklická síť (10)

Speciálním případem acyklické sítě jsou takzvané **vícevrstvé neuronové sítě**, které jsou charakterizovány tím, že každý neuron jedné vrstvy je spojen s každým neuronem další vrstvy. Vrstvy v těchto sítích dělíme do 3 základních skupin. První z nich je vstupní vrstva, která přijímá vstupy a posílá je do dalších vrstev. Následuje jedna či více vrstev skrytých. Síť je zakončena výstupní vrstvou, která nám dává výstupní hodnoty. Příklad sítě se 2 skrytými vrstvami můžeme vidět na Obr. 7



Obr. 7 Vícevrstvá neuronová síť (10)

Vícevrstvé neuronové sítě jsou základem pro většinu dnešních v praxi používaných sítí. Při návrhu těchto sítí bývá velkým problémem zvolit správné množství skrytých vrstev a neuronů v nich. Větší počet vrstev a neuronů v nich vede k větším nárokům na výpočetní kapacitu a hrozí zde přeučení sítě. Naopak pokud je počet vrstev či neuronů v nich příliš malý, tato síť nebude schopna řešit komplikovanější problémy. Pro praxi byly vyvinuty heuristiky, které odhadují optimální počet skrytých sítí a neuronů v nich pro daný problém. (10)

2.3.1 Aktivační funkce

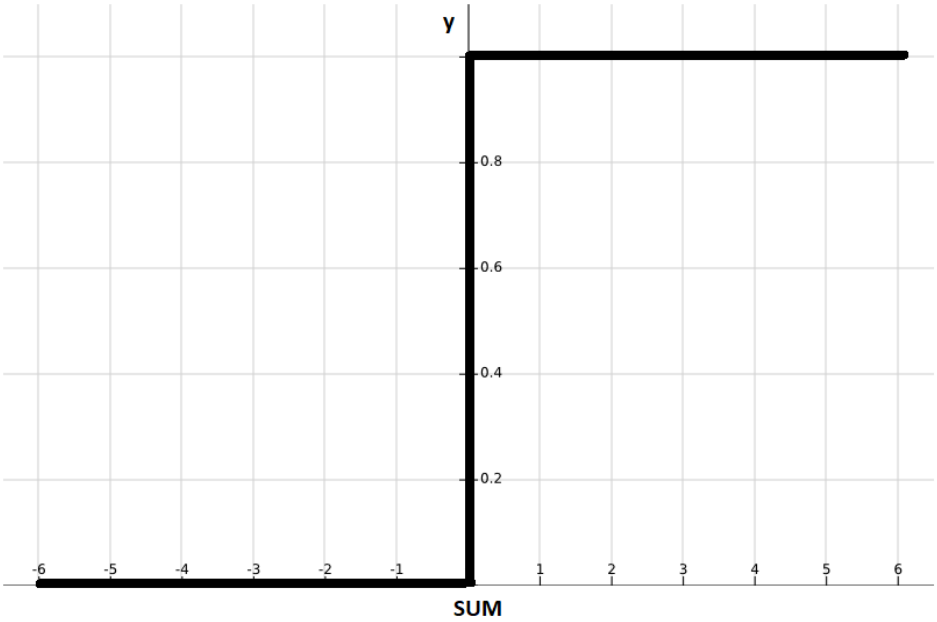
Aktivační funkce nám určuje, jakým způsobem transformuje vstupy neuronu, tedy vážený součet vstupů, na výstup z jednotlivých neuronů. Výběr aktivační funkce je důležitou součástí návrhu architektury neuronové sítě, má dopad jak na přesnost výsledků, tak na čas nutný k trénování sítě. Dále použitá funkce určuje, jak rychle bude neuronová síť konvergovat při trénování ke správným výsledkům. Aktivačních funkcí existuje mnoho, níže jsou představeny některé z nich. (11)

Skoková aktivační funkce

Skoková aktivační funkce, viz Obr. 8, je spíše historická, používala se např. v prvním perceptronu či v neuronu Adaline. Její výstup je bud 1, pokud vážený součet vstupů přesáhne práh w_0, či 0, pokud k tomu nedojde. To je ale v mnoha případech nevhodné, jelikož to znamená, že pro dva vážené součty, které jsou blízko sebe, každý ale na opačné straně aktivační funkce, dostáváme dva úplně rozdílné výsledky. Stejně tak nám tato funkce neumožnuje klasifikovat do více než dvou kategorií. (2, 11)

$$\boldsymbol{y = 1 \quad pro\ SUM > w_0}$$

$$\boldsymbol{y = 0 \quad pro\ SUM < w_0}$$

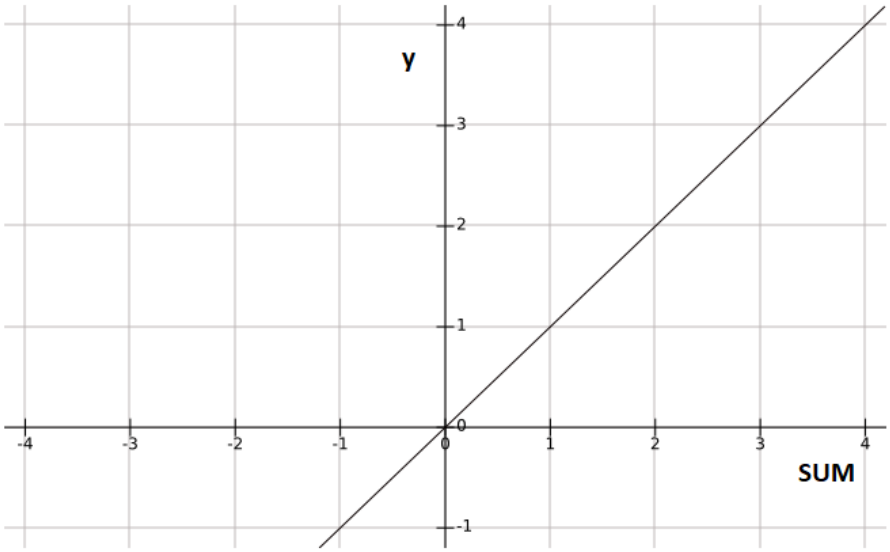


Obr. 8 Skoková aktivační funkce (autor)

Lineární aktivační funkce

Lineární aktivační funkce je funkce, která vezme vážený součet vstupů a pošle ho na výstup. Její graf můžeme vidět na Obr. 9. Na rozdíl ad skokové aktivační funkce umožnuje různorodý výstup (nejen 0 nebo 1). Pro použití ve složitějších neuronových sítích je ale nevhodná, není schopná si poradit s komplikovanějšími vstupy. (11)

$$y = f(SUM) = SUM$$

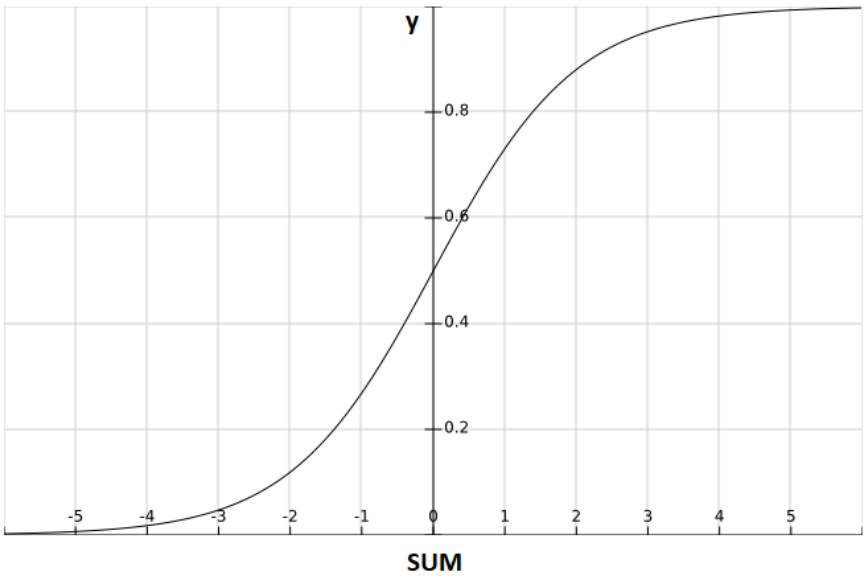


Obr. 9 Lineární aktivační funkce (autor)

Sigmodiální funkce

Sigmodiální funkce je první nelineární funkce, která je zde uvedena. Hlavním výhodou nelineárních funkcí je možnost jejich použití pro trénování neuronových sítí pomocí algoritmu zpětného šíření. Sigmodiální funkce nabývá na výstupu hodnot z intervalu (0,1). Dlouhou dobu byla jednou ze dvou nejpoužívanějších aktivačních funkcí, než byla nahrazena novější funkcí ReLU a jejími deriváty. Problémů této funkce je několik, jednak je náročná na výpočet, dále pomalu konverguje a není centrovaná okolo 0, jak můžeme vidět na Obr. 10. Dalším velkým problémem je, že při použití této funkce, dochází při trénování sítě k mizení gradientů. (11)

$$y = f(SUM) = \frac{1}{1 + e^{-SUM}}$$

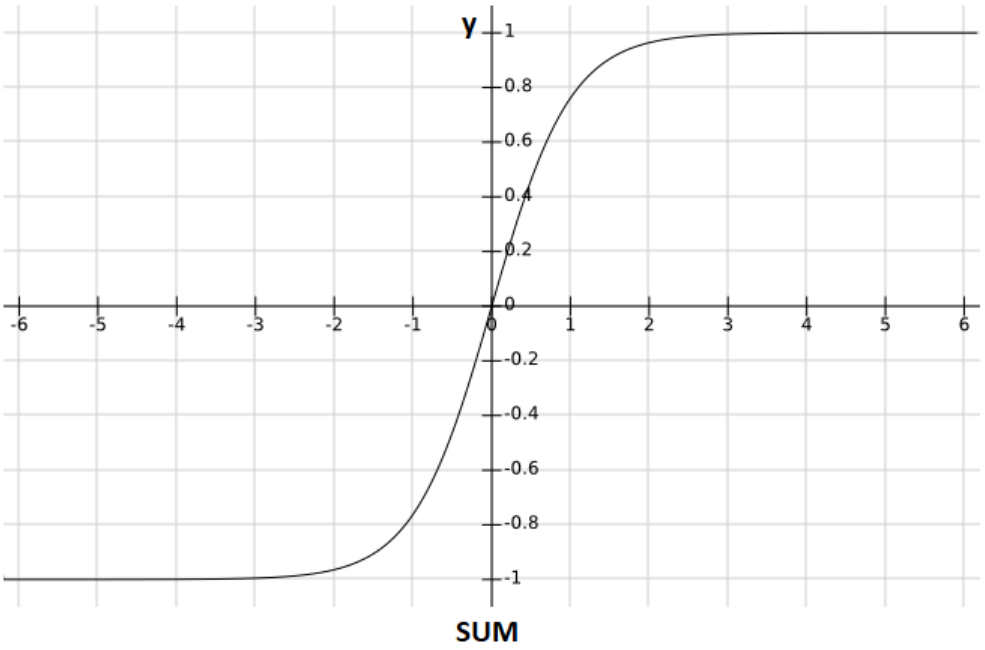


Obr. 10 Sigmoida (autor)

Hyperbolický tangens

Funkce podobná sigmodiální funkci, druhá z dříve často používaných funkcí. Oproti sigmodiální funkci je její výstup vycentrovaný kolem nuly, jak můžeme vidět na Obr. 11, tzn. má výstupy z intervalu (-1,1). Ostatní problémy, jako je náročnost na výpočet a pomalá konvergence, zde ale přetrvávají. (11)

$$y = f(SUM) = tgh(SUM)$$

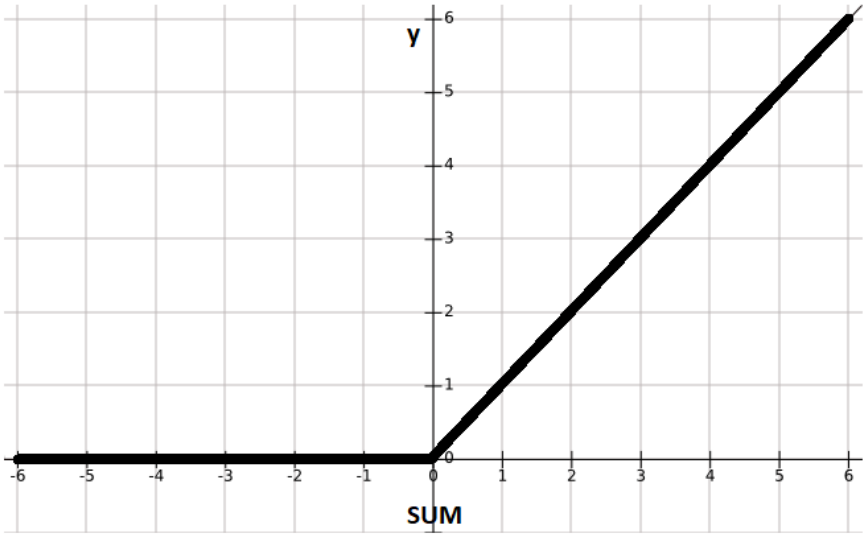


Obr. 11 Hyperbolický tangens (autor)

Relu

V dnešní době nejpoužívanější aktivační funkce v hlubokých neuronových sítích, můžeme ji vidět na Obr. 12. Používá se kvůli jednoduchému výpočtu, netrpí problémem mizejících gradientů a rychle konverguje. Je vhodná ale jen na použití ve skrytých vrstvách a pokud se nám součet vstupů dostane do záporných čísel, ztrácí možnost použít algoritmus zpětného šíření a nemůže se tudíž učit. (12)

$$\boldsymbol{f(SUM) = \max(0, SUM)}$$



Obr. 12 ReLU (autor)

2.4 Proces učení sítě

Po vytvoření architektury sítě následuje proces učení. Zde existují dva základní přístupy, učení s učitelem a bez učitele. Pro aplikaci každého z těchto přístupů existují algoritmy, které tyto myšlenky převádějí do praktického využití.

2.4.1 Učení sítě s učitelem

Pro učení sítě máme data, která máme již zařazená do správných tříd. Tato data dáme síti k dispozici, a ta si pomocí některého z používaných algoritmů přizpůsobí svoje vnitřní nastavení tak, aby dokázala klasifikovat data do správných tříd. Například pokud učíme neuronová síť rozlišit, zda je na obrázku pes či kočka, dáme ji k dispozici obrázky z každé kategorie. Neuronová síť se pak na těchto datech naučí zařadit obrázky do těchto dvou kategorií. Správnost výsledků při testování pak závisí na architektuře sítě a kvalitě testovacích dat.

Jako v jiných oblastech strojového učení se zde data, která máme již klasifikovaná, dělí pro účel trénování sítě na dvě skupiny, a to data trénovací, kde neuronovou síť učíme data klasifikovat, a data testovací, kde pomocí metrik zjišťujeme, jak dobře zvládá neuronová síť řešit problém na neznámých datech. (10)

2.4.2 Učení sítě bez učitele

Zde nemáme žádná data zařazená do tříd. Dáme neuronové síti k dispozici vstupní data a očekáváme, že v nich najde závislosti a vzory, která se v datech vyskytují. Cílem tedy může být buď rozdělit data která máme k dispozici (například údaje o návštěvnících webové stránky), do kategorií, se kterými pak můžeme pracovat, či najít vztahy, která spojují jednotlivé údaje v datech. (10)

2.4.3 Učení jednoho neuronu

Učení v neuronech probíhá na principu nastavování vah (w) na jednotlivých vstupech. Níže jsou představeny dvě metody, které se používají pro učení neuronů.

Gradientní metoda

Jednou z metod pro učení neuronů je takzvaná gradientní metoda. Základem této metody je výpočet střední kvadratické chyby, kterou se snažíme minimalizovat.

$$Err(w) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Kde **y** značí hodnotu cílového atributu, ŷ hodnotu současného zařazení sítí, **w** váhový vektor a **n** počet příkladů z trénovací množiny.

Pokud se budeme snažit o minimalizaci této chyby, dostaneme pravidlo pro modifikaci vah, kde výslednou váhu **w** dostaneme jako původní váhu **w**, ke které přičteme **Δw**, kterou dostaneme jako

$$\Delta w = -\eta \frac{\partial Err}{\partial w} = \eta \sum_{i=1}^{n} (yi - \hat{y}i)\, xi$$

η zde znamená míru učení (délku kroku), což je konstanta, která se nastavuje před trénováním sítě. Určuje, jak rychle se bude měnit **w**.

Tímto postupem nalezneme váhy **w**, které budou odpovídat globálnímu minimu Err(**w**).

Jak můžeme z výše uvedených vzorců vidět, při gradientní metodě nejdříve vyhodnotíme celý testovací set a až poté měníme váhy u jednotlivých vstupů.

Stochastická gradientní metoda

Tato varianta je používanější než gradientní metoda. Hlavním rozdílem je, že při použití této metody měníme váhu **w** po každé klasifikaci jednoho případu z testovacích dat, pokud v daném případě došlo k odchylce mezi požadovaným a skutečným výstupem systému.

$$w^{(i+1)} = w^{(i)} + \Delta w^{(i)}$$

$$\Delta w^{(i)} = \eta(y_i - \hat{y}_i)x_i$$

Tento postup nám nedává žádnou zpětnou vazbu, musíme tedy testovacími daty procházet tak dlouho, dokud není neuron naučen. (2, 10)

2.4.4 Učení vícevrstvé sítě

Pro učení neuronových sítí se nejčastěji používá algoritmus zpětného šíření. Princip tohoto algoritmu můžeme rozdělit do několika kroků.

Inicializace vah v jednotlivých spojeních mezi neurony. Váhy se nejčastěji inicializují náhodnými malými čísly.

Dopředná propagace je proces, kdy posíláme do sítě testovací příklad. Každý neuron ve vstupní vrstvě obdrží signál, zprostředkuje jeho přenos neuronům v další vrstvě. Neurony v této vrstvě spočítají svojí aktivaci a pošlou signál na svoje výstupy. Výsledek, který dostaneme na výstupní vrstvě pak porovnáme s tím, co jsme chtěli dostat. K tomuto porovnání nám slouží ztrátová funkce.

Ztrátová funkce nám tedy říká, jak moc se liší současný výsledek od chtěného. Cílem je hodnotu této funkce minimalizovat. Existují dva základní způsoby, jak spočítat ztrátovou funkci. Prvním a nejpoužívanějším je suma čtverců.

$$Err = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Kde y_i je výsledek neuronové sítě a $\hat{y}_i$ je správný výsledek.

Druhým pak křížová entropie

$$Err = -\sum_{i=1}^{n} y_{o,c} \log (p_{o,c})$$

Kde y binární indikátor který dává 1, pokud c je správně klasifikováno pro případ o. p je pravděpodobnost že případ o patří do třídy c.

Tuto chybu (ztrátovou funkci) pak zpětně šíříme do celé sítě a podle ní aktualizujeme váhy na jednotlivých spojeních neuronů. K aktualizaci vah se používá gradientní metody, která je již uvedena u učení jednoho neuronu.

Proces učení pomocí algoritmu zpětného šíření běží do té doby, než dojde k zastavovací podmínce. To může být například ustálení chybové funkce, proběhnutí daného počtu iterací či pokles pod předem danou hladinu chybové funkce. (10, 13)

3 Hluboké neuronové sítě

Hluboké neuronové sítě můžeme považovat za rozšíření vícevrstvých neuronových sítí. Kniha "Deep Learning - Practical Neural Networks with Java" je například definuje jako:

- Sítě s více neurony než předcházející sítě
- Více způsobů pro propojení jednotlivých vrstev
- Umožněné strmým nárůstem výpočetního výkonu v posledních letech pro jejich trénování
- Umí automaticky nalézt závislosti v datech

Tato kapitola popisuje několik používaných metod z oblasti hlubokých neuronových sítí a možnost jejich použití v Javě pomocí knihovny Deeplearning4j[1]. (14)

3.1 Historie

První studie dokumentující použití hlubokých neuronových sítí byla publikována v roce 2006 profesorem Hintnem z univerzity v Torontu. Profesor Hinton zde ukázal možnost použití hlubokých neuronových sítí pro rozpoznávání ručně psaných číslic jedna až devět. K tomu byla použita databáze MNIST, která obsahuje 70 tisíc obrázků, 60 tisíc pro trénování a 10 tisíc pro testování. Každý obrázek má rozměry 28*28 pixelů a představuje jednu ručně psanou číslici a ke každému obrázku je uvedeno, o jakou číslici se jedná. I když v této studii dosáhl dobrých výsledků v porovnání s ostatními metodami, studii se nedostalo velké pozornosti. (15)

Pozornosti se této metodě dostalo až v roce 2012, kdy tým SuperVision z university v Torontu za použití metody vyvinuté doktorem Hintnem vyhrál soutěž Imagenet Large Scale Visual Recognition Challenge[2]. V této soutěži je cílem správně zařadit obrázek do jedné z kategorií, například kočka, pes, letadlo, auto a mnoha dalších. Tým SuperVision zde dosáhl chybovosti kolem 15% a vyhrál s náskokem před ostatními týmy, jejichž chybovost byla kolem 25%. Zajímavé bylo, že zatímco rozdíl mezi prvním a druhým týmem byl kolem 10%, rozdíly mezi třemi dalšími týmy byly jen 0,1%. To ukázalo převahu této nové metody nad dosud používanými metodami a hluboké neuronové sítě se dostali do popředí zájmu mnoha lidí zabývajících se strojovým učením. (14)

[1] https://deeplearning4j.org/
[2] http://www.image-net.org/challenges/LSVRC/

3.2 Deeplearning4j

Jedná se o knihovnu pro programování neuronových sítí. Knihovna je psaná jako open-source a vydávaná pod licencí Apache 2.0. Knihovna je psaná v Javě a umožnuje integraci s technologiemi Hadoop a Apache Spark. K výpočtům používá open-source knihovnu ND4j a podporuje tvorbu neuronových sítí na CPU i GPU.

Použití knihovny pro tvorbu vícevrstvých sítí

- Příprava dat
  Knihovna Deeplearning4j používá pro trénování sítí takzvané "Dataset iterators". To je třída, která pomáhá dávkovat a iterovat napříč našimi trénovacími daty. Pro konkrétní ukázkové případy můžeme použít již vytvořenou třídu, pro všechna ostatní data je ale třeba tuto třídu přetvořit tak, aby vyhovovala konkrétním potřebám.

- Vytvoření sítě
  Pro vytváření neuronových sítí existují v knihovně dvě různé třídy. Jednodušší z nich je třída MultiLayerNetwork. Ta má právě jednu vstupní vrstvu, následuje libovolný počet skrytých vrstev a na konci je právě jedna výstupní vrstva. Příklady neuronových sítí budou v této práci vždy uváděny s konstruktorem této třídy. V knihovně Deepleraning4j existuje ještě složitější třída, a to sice ComputationGraph. Ta se liší od předchozí třídy tím, že umožnuje použít více vstupních a více výstupních vrstev. Dalším rozšířením pak je možnost propojení skrytých vrstev acyklickým grafem. (16)

3.2.1 Příklad vytvoření sítě v několika krocích

1) Vytvoření objektu třídy MultiLayerNetwork pomocí konstuktoru

```
1.  MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
```

2) Globální nastavení neuronové sítě, zde postupně volíme jednotlivé parametry pro neuronovou síť. Parametry lze přenastavit u lokální vrstvy, pokud je to žádoucí.
   a) seed: slouží pro reprodukci výsledků při několikanásobném běhu sítě. Jedná se o integer který ovlivňuje počáteční nastavení sítě, hlavně co se týká nastavení vah před trénováním, ale i dalších náhodných parametrů v síti.
   b) learningRate: tento parametr ovlivňuje, jak moc se budou měnit váhy na jednotlivých spojeních pokaždé, když dojde k aktualizaci modelu
   c) optimizationAlgo: určuje který optimalizační algoritmus se použije pro trénování neuronové sítě
   d) updater: zde nastavuje algoritmus, který se použije pro aktualizaci vah

e) Celý kod pro globální nastavení sítě může vypadat například takto:

```
1.  .seed(123)
2.  .learningRate(0.1)
3.  .optimizationAlgo  (OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
4.  .updater(new Nesterovs(0.9))
```

3) Vytváření jednotlivých vrstev
   a) Nejdříve zavoláme metodu, která nám poskytne potřebné API pro vytváření jednotlivých vrstev

```
1.  .list()
```

   b) Nyní již postupně vytváříme jednotlivé vrstvy, u každé vrstvy musíme nastavit několik parametrů
      i) nIn: udává počet vstupů do dané vrstvy (neboli počet výstupů z předchozí), můžeme vidět na řádku 2 a 8.
      ii) nOut: udává počet výstupů z dané vrstvy (neboli počet vstupů do další), můžeme vidět na řádku 3 a 9.
      iii) activation: nastavení aktivační funkce, můžeme vidět na řádku 4 a 10.

      Příklad přidání dvou vrstev, první je vrstva vstupní, druhá už je vrstva výstupní:

```
1.  .layer(newDenseLayer.Builder()
2.      .nIn(784)
3.      .nOut(100)
4.      .activation(Activation.RELU)
5.      .build())
6.
7.  .layer(newOutputLayer.Builder()
8.      .nIn(100)
9.      .nOut(10)
10.     .activation(Activation.SIGMOID)
11.     .build())
```

4) Nastavení, zda chceme použít algoritmus zpětného šíření

```
1.  .backprop(true)
```

5) Zavolání funkce, která nám podle zadaných parametrů síť nakonfiguruje

```
1.  .build()
```

Celý kód pro vytvoření sítě by tedy mohl vypadat například takto:

```
1.  MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
2.
3.  .seed(123)
4.  .learningRate(0.1)
```

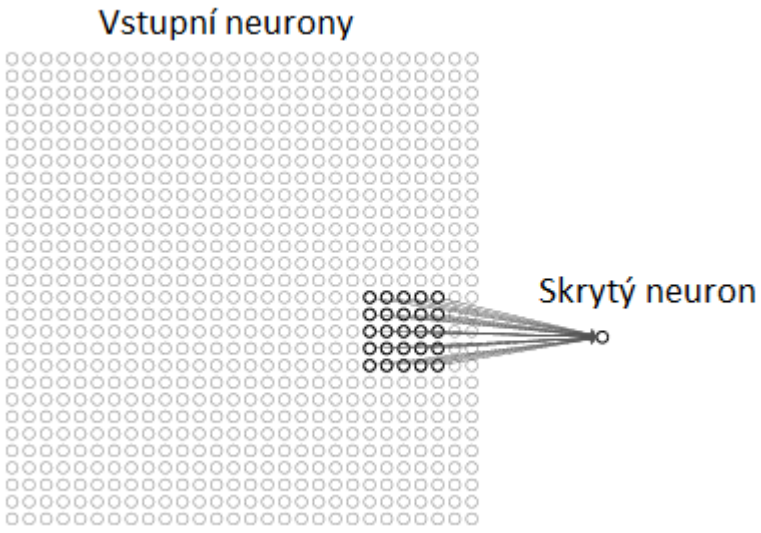
```
5.  .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
6.  .updater(new Nesterovs(0.9))
7.
8.  .list()
9.
10. .layer(newDenseLayer.Builder()
11.     .nIn(784)
12.     .nOut(100)
13.     .activation(Activation.RELU)
14.     .build())
15.
16. .layer(new OutputLayer.Builder()
17.     .nIn(100)
18.     .nOut(10)
19.     .activation(Activation.SIGMOID)
20.     .build())
21.
22. .build()
```

(17)

3.3 Konvoluční neuronové sítě

Konvoluční neuronové sítě jsou speciálním případem neuronových sítí, které se používají hlavně pro analýzu a klasifikaci obrázků, i když svoje uplatnění nacházejí i v jiných úlohách. Od klasických neuronových sítí se odlišují svojí architekturou, která je založena na 3 základních myšlenkách. První je použití lokálních polí, další pak použití sdílených vah a třetí použití sdružovací vrstvy.

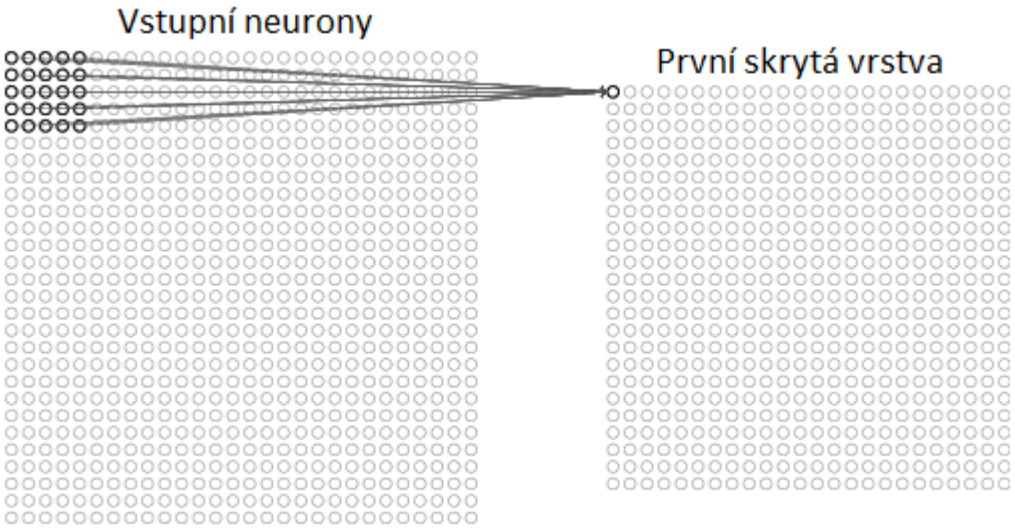
V případě vícevrstvých neuronových sítí je každý neuron na vstupu napojen na každý neuron první skryté vrstvy. Při použití **lokálních polí** ale vybereme část vstupních neuronů a tu napojíme na jeden neuron skryté vrstvy. Každý neuron v první skryté vrstvě bude tedy napojen jen na vybrané vstupní neurony. Příklad můžeme vidět na Obr. 13, kde je vybraná část 25 (5x5) vstupních neuronů napojena na jeden skrytý neuron.



Obr. 13 Použití lokálních polí (18)

Pro použití ještě potřebujeme zvolit délku kroku. Délka kroku určuje, o kolik polí se posuneme poté, co zvolíme jedno lokální pole. Délka kroku 1 v našem případě tedy znamená, že pokud první lokální pole vezmeme 5x5 vlevo nahoře, druhé bude posunuté o jedna doprava, bude tedy začínat ve druhém sloupci.

Pokud bychom v našem ukázkovém případě prošli vstupní neurony s krokem 1 a filtrem o velikosti 5x5, dostali bychom pro pole vstupů 28x28 ve skryté vrstvě 24x24 neuronů. Pro lepší představu je toto znázorněno na Obr. 14.

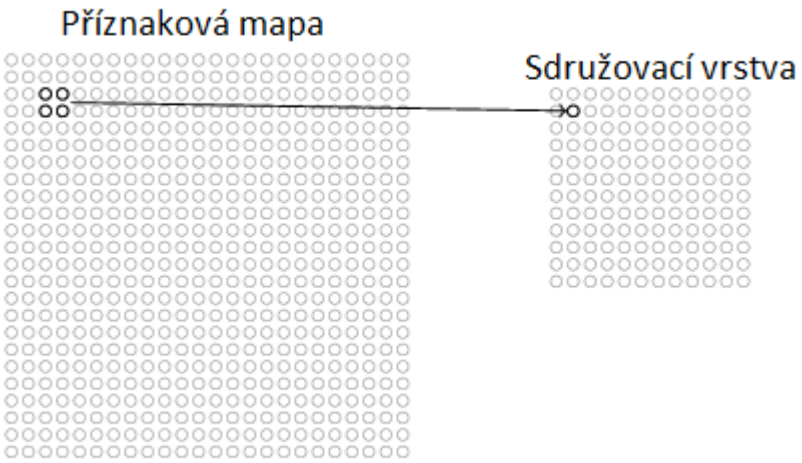


Obr. 14 Tvorba první skryté vrstvy (18)

Všechny tyto neurony ve vytvořené vrstvě mají **sdílené váhy**. Použití sdílených vah způsobí, že všechny neurony v této vrstvě detekují stejný příznak, pokud bychom například měli na vstupu obrázek, daná vrstva může být schopna identifikovat svislou čáru nezávisle na její pozici.

Použitím filtru (v našem případě 5x5) a sdílených vah nám vznikne příznaková mapa. Abychom byli schopni rozeznat více než jeden příznak, konvoluční vrstva v neuronové síti musí mít těchto map více.

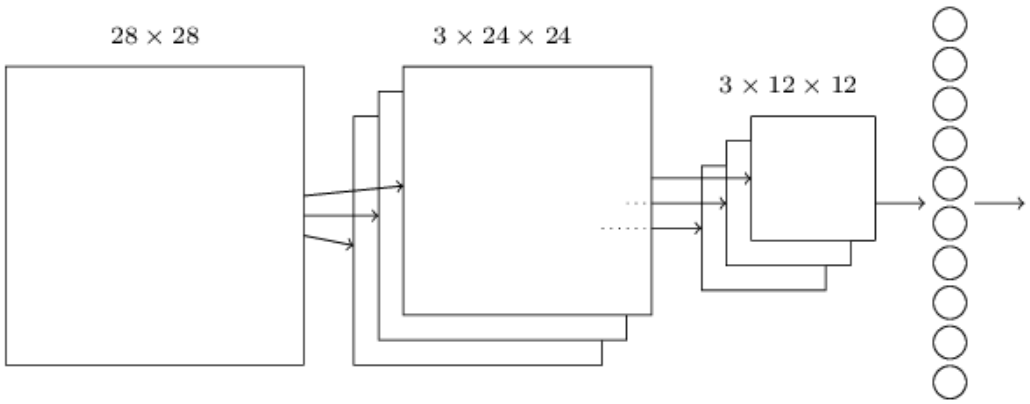
Sdružovací vrstva bývá navázána přímo na konvoluční vrstvu a jejím účelem je zhotovit zhuštěnou příznakovou mapu. Toho docílíme tak, že vybereme oblast příznakové mapy, např 2x2, a tyto neurony spojíme do jednoho. Existuje více způsobů, jak toho dosáhnout, jeden z nejčastějších je sdružování dle maxima, kde ve zhuštěné příznakové mapě zůstane neuron s největší hodnotou na výstupu. Tvorba zhuštěné příznakové mapy je znázorněna na Obr. 15, kde jsme z původních 24x24 neuronů dostali ve sdružovací vrstvě 12x12 neuronů.



Obr. 15 Tvorba zhuštěné příznakové mapy (18)

Použití sdružovací vrstvy snižuje počet parametrů a neuronů v síti a tím i nároky na výpočetní techniku. Při jejím použití nás nezajímá přesné umístění daného příznaku, ale spíše jeho poloha vzhledem k ostatním příznakům.

V příkladu na Obr. 16 můžeme vidět již celou konvoluční neuronovou síť s jednou konvoluční a jednou sdružovací vrstvou. Z pole vstupních dat 28x28 jsme zde vytvořily tři příznakové mapy. Z těchto map byly ve sdružovací vrstvě vyrobeny 3 zhuštěné příznakové mapy, každá z nich obsahuje 12x12 neuronů. Všechny tyto neurony jsou pak napojeny na každý neuron výstupní vrstvy. (18)



Obr. 16 Příklad konvoluční neuronové sítě (18)

Použití v knihovně Deeplearning4j

Pokud chceme v knihovně Deeplearning4j použít princip konvolučních neuronových sítí, máme možnost při nastavování sítě vytvořit rovnou konvoluční vrstvu. Ta má několik speciálních parametrů, které nastavujeme při vytváření této vrstvy. Prvním z nich je kernel size, který určuje velikost filtru. Druhým je parametr stride, který určuje délku kroku. Konstruktor pro tvorbu konvoluční vrstvy tedy vypadá následovně:

```
1.  Builder(int[] kernelSize, int[] stride)
```

Pro náš příklad by tedy tvorba konvoluční vrstvy mohla vypadat takto:

```
1.  .addLayer(new ConvolutionLayer.Builder()
2.  .kernelSize(5)
3.  .stride(1)
4.  .build())
```

Konvoluční vrstvu často následuje vrstva sdružovací, kde navolíme parametry kernel a stride, podobně jako u konvoluční vrstvy. Kromě toho musíme určit, podle jakého principu bude sdružování probíhat, v ukázkovém případě dole je použit nejpoužívanější způsob, a to max pooling.

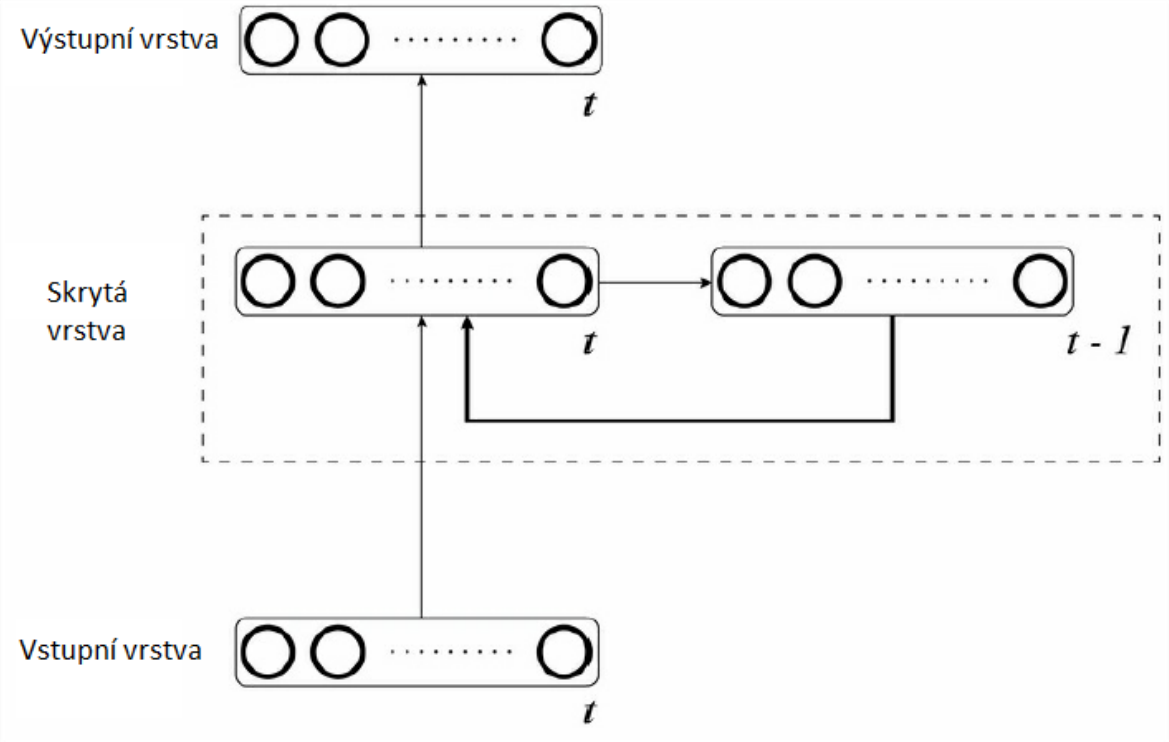
Tvorba vrstvy by tedy mohla vypadat takto:

```
1.  .addLayer(newSubsamplingLayer.Builder(SubsamplingLayer
2.  .PoolingType.MAX))
3.  .kernelSize(5)
4.  .stride(1)
5.  .build())
```

(19)

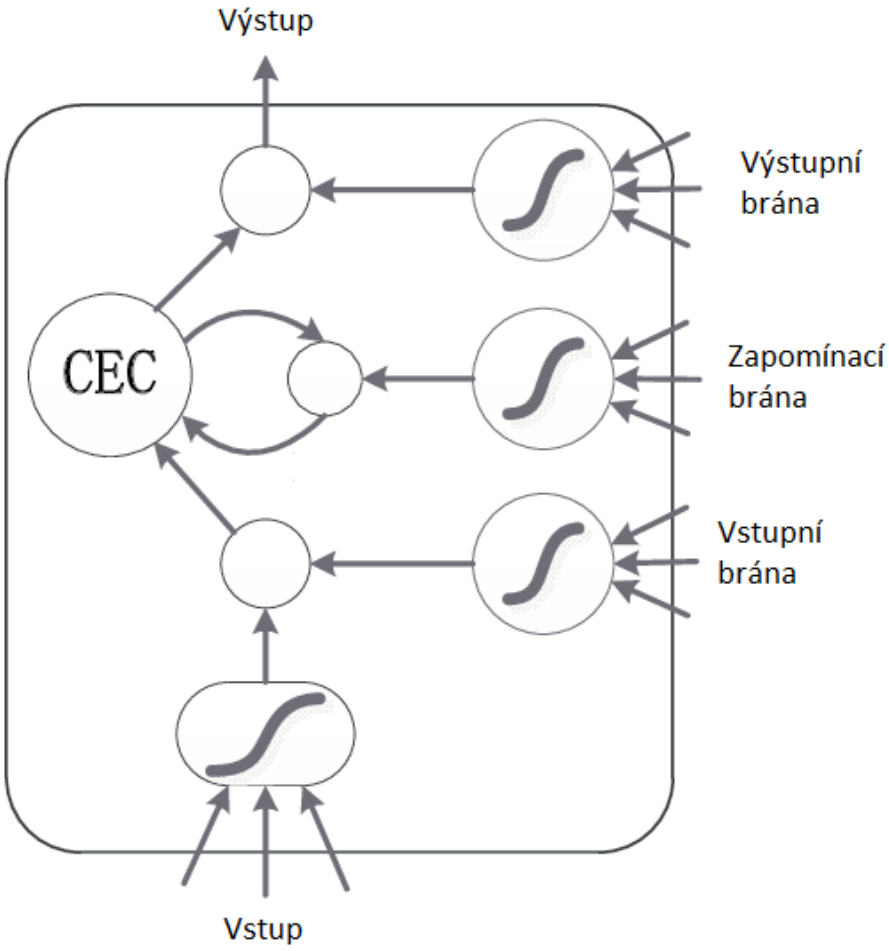
3.4 Rekurentní neuronové sítě

Rekurentní neuronové sítě jsou dalším speciálním případem neuronových sítí, dosahují dobrých výsledků při řešení problémů, v kterých se vyskytuje posloupnost dat, například porozumění přirozenému jazyku či data s časovou závislostí. Z hlediska architektury se jedná o cyklické sítě, každý výstup je zde funkcí předchozího výstupu. To můžeme vidět na Obr. 17, t nám zde značí současnou iteraci, t-1 iteraci minulou.



Obr. 17 Grafický model rekurentní neuronové sítě (14)

I když tyto sítě dosahují lepších výsledků při zpracování kontextuálních dat než klasické vícevrstvé sítě, dostávají se do problémů, pokud jsou závislé údaje ve vstupních datech dále od sebe. Řešení tohoto problému nabízí LSTM (long short term memory) sítě. Ty zavádí do sítě jednotku CEC (Constant Error Carousel), která uchovává vnitřní stav. K této jednotce se ještě přidávají tři brány. Vstupní brána se stará o to, aby se neuron aktivoval pouze v případě, že přijde relevantní informace, ne jinak. Výstupní brána zase o to, aby se hodnota neuronu šířila jen v případě, že je relevantní. Třetí je brána zapomínací, která určuje, která data budou uchovávána a která zapomenuta. Tento model je zachycen na Obr. 18, kde můžeme vidět grafický model jednoho LSTM bloku.



Obr. 18 Grafický model LSTM bloku (20)

Tato základní architektura LSTM bloku bývá často ještě rozšiřována o spojení z CEC k jednotlivým branám. Tyto spojení mohou vést bud ke všem třem branám, či jen k některým z nich. V anglické literatuře se nazývají peephole connections (nazývá je tak i knihovna Deeplearning4j) a jedná se o standartní spojení s váhami, které se ale neúčastní algoritmu zpětného šíření chyby. (14)

Pro rekurentní sítě také existují v knihovně Deeplearning4j předpřipravené možnosti pro vytváření jednotlivých vrstev. Na příkladu dole můžeme vidět neuronovou síť se dvěma vrstvami, první je LSTM vrstva (řádek 1), která v případě použití tohoto konstruktoru nemá peephole connections. Výstupní vrstvou je pak vrstva rekurentní (řádek 7), která nám na výstupu dává požadované hodnoty, v tomto případě se jednalo o předpověď teploty vody v moři pro další den.

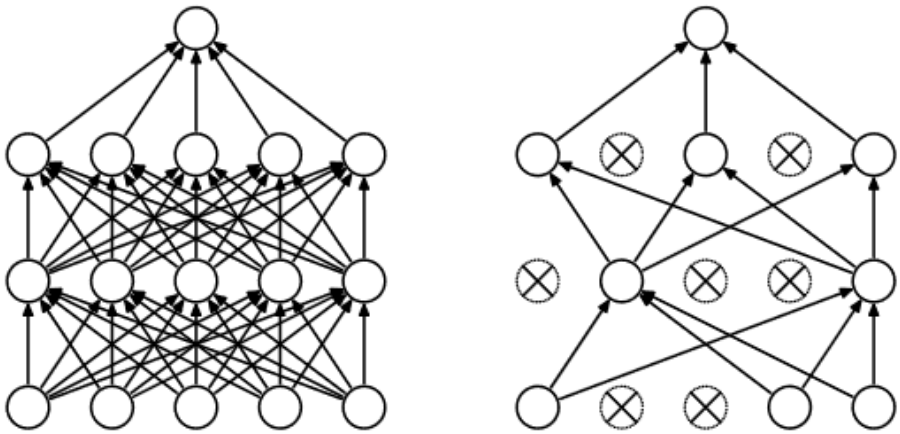
```
1.  .layer(new LSTM.Builder()
2.      .activation(Activation.TANH)
3.      .nIn(84)
4.      .nOut(200)
5.      .build())
6.
7.  .layer(new RnnOutputLayer.Builder()
8.      .activation(Activation.IDENTITY)  //jedná se o lineární aktivační funkci
9.      .nIn(200)
10.     .nOut(52)
11.     .build())
```

(21)

3.5 Výpadek

Výpadek je metoda, která pomáhá řešit dva časté problémy neuronových sítí, a to problém přeučení a problém mizení gradientu.

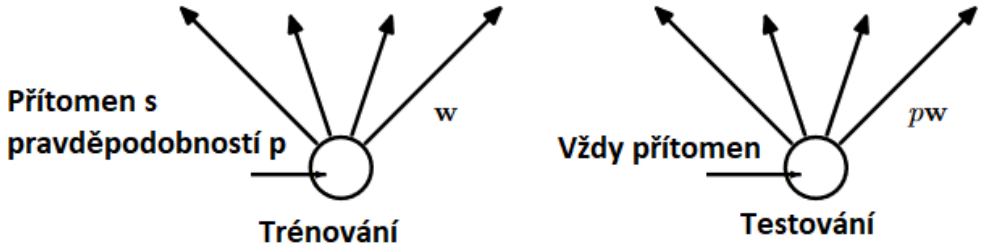
Výpadek funguje na principu vyřazení některých neuronů. Neurony k vyřazení vybíráme náhodně, v praxi se doporučuje pravděpodobnost vyřazení kolem 50% (p=0,5) pro skryté vrstvy. Použití výpadku můžeme vidět na Obr. 19, kde vlevo je původní neuronová síť a vpravo síť s vyřazenými neurony. (22)



Obr. 19 Příklad použití výpadku (24)

Výpadek se používá před každým trénování na části dat. Dostáváme tedy různé sítě, pokud n je počet neuronů, můžeme dostat až 2^n různých sítí. Tyto sítě pořád sdílí všechny váhy,

počet parametrů tedy zůstává stejný. Pro účely testování sítě se použije celá síť bez výpadku, každá výsledná váha je zde vynásobena pravděpodobností vyřazení (p) neuronu, ze kterého vychází. Tento proces nám zachycuje Obr. 20.



Obr. 20 Použití trénovacích vah pro testování (24)

Protože jednotlivé sítě vždy trénujeme na malém vzorku dat, snižuje se zde pravděpodobnost přeučení. Kombinací modelů se také dosahuje lepších výsledků než při trénování pouze jedné sítě. (22)

V knihovně Deeplearning4j je již metoda pro použití výpadku vytvořena, a vypadá takto:

```
1.  public Dropout(double activationRetainProbability)
```

parametr `activationRetainProbability` zde určuje pravděpodobnost ponechání daného neuronu v síti, při p=1 tedy žádný neuron nevyřadíme, při p=0 v dané vrstvě žádný neuron nezůstane.

Tuto metodu můžeme použít dvěma způsoby, bud ji použijeme v nastavení globálních parametrů sítě, pak bude automaticky výpadek s danou pravděpodobností aplikován na každou vrstvu. Nebo můžeme metodu volat až při vytváření dané vrstvy, pak bude výpadek použit jen na této vrstvě. Kód je v obou případech stejný, liší se jen jeho umístění.

```
1.  .dropOut(double inputRetainProbability)
```

(23)

3.6 Word2Vec

Word2Vec je jedna z důležitých metod při práci s přirozeným jazykem, umožnuje nám zjistit a matematicky vyjádřit, jak jsou si dvě slova podobná. K tomu se využívá vnoření slov, kde každému slovu přiřadíme vícerozměrný vektor. Na těchto vektorech pak můžeme provádět matematické operace jako sčítání a odečítání, např.

Vektor (“Praha“) = Vektor (“Berlín“) – Vektor (“Německo“) + Vektor (“Česko“)

Pomocí vektorů také můžeme hledat podobná slova, např. při zadání slova Sweden (Švédsko) dostaneme jako 9 nejpodobnějších slov další názvy zemí. Výsledek takového dotazu můžeme vidět na Obr. 21

Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

Obr. 21 Podobnost slov pomocí Word2Vec (24)

Tvorba modelu

Algoritmus pro vytvoření slovních vektorů funguje na principu učení bez učitele. Nejdříve z dostupných textů vytvoří slovník všech slov, které se v nich vyskytují. Poté se ke každému slovu vytváří vektor, a to na základě kontextu, ve kterém se dané slovo vyskytuje. Kontext zde znamená slova, která se vyskytují před a za daným slovem. Základem je zde tedy myšlenka, že významově podobná slova jsou obklopena stejnými uvozovacími a významovými slovy a používají se ve větách, které jsou z velké části stejné. Čím podobnější jsou věty, ve kterých se daná slova nachází, tím bližší si budou vektory těchto slov.

Použití pomocí Deeplearning4j

Pokud chceme použít tuto metodu v knihovně Deeplearning4j, musíme nejdřív rozdělit věty na jednotlivá slova. Toho dosáhneme pomocí metody TokenizerFactory. K odstranění interpunkčních znamének a velkých písmen zavoláme metodu CommonPreprocessor.

```
1.  TokenizerFactory t = new DefaultTokenizerFactory();
2.  t.setTokenPreProcessor(new CommonPreprocessor());
```

S vytvořenou instancí této metody již můžeme zhotovit neuronovou síť:

```
1.  Word2Vec model = new Word2Vec.Builder()
2.                  .minWordFrequency(5)
3.                  .layerSize(100)
4.                  .iterate(iter)
5.                  .tokenizerFactory(t)
6.                  .build();
```

Tato metoda má několik specifických parametrů. Tyto parametry jsou:

minWordFrequency nám zde udává, kolikrát se slovo musí v našich datech vyskytnout, aby pro něj byl vytvořen vektor.

layerSize určuje, kolik dimenzí bude mít vektor.

Iterate řekne neuronové síti, na kterých datech má trénovat.

Pokud chceme s modelem dále pracovat, je třeba ho uložit

```
1.  WordVectorSerializer.writeWord2VecModel(model, "názevmodelu.zip");
```

Word2Vec modely můžeme znovu načíst a dále s nimi pracovat:

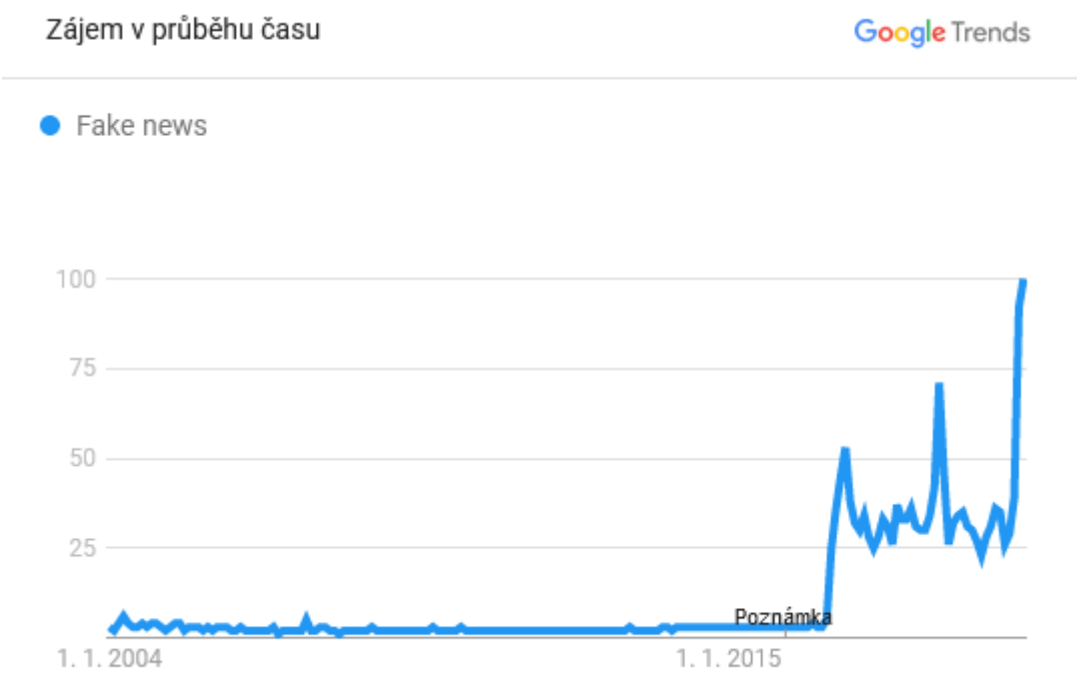
```
1.  WordVectors wordVectors = WordVectorSerializer.loadStaticModel(new File(Path));
```

(25)

4 Falešné zprávy

První známé případy šíření nepravdivých informací jsou datovány až do antiky, kdy Octavian, budoucí první římský císař, jenž při své inauguraci přijal jméno Augustus, vedl kampaň proti svému rivalovi Antonymu. Kampaň měla více podob, jednou z nich byly krátké slogany označujícího Antonyho za opilce či nevěrníka, které byly ryty do mincí. Další taktikou byly šíření zprávy, že Antony je loutka egyptské královny Kleopatry. (27)

Takovýchto případů se dá v historii najít celá řada, v 21. století se začalo mluvit o falešných zprávách hlavně v souvislosti s americkými prezidentskými volbami v roce 2016. Vzrůstající zájem o termín falešné zprávy (fake news) můžeme dobře vidět na Obr. 22. Tento graf nám zobrazuje popularitu vyhledávání daného termínu vzhledem k největšímu zájmu (osa y zde značí procenta, 100% znamená nejvyšší počet vyhledávání). Od roku 2004 (kdy Google začal data sbírat) do roku 2016 se popularita termínu fake news pohybovala kolem 3–5% zájmu v roce 2020. První velkou vlnu zájmu můžeme vidět v souvislosti s americkými volbami v roce 2016. Druhý vzestup můžeme vidět v souvislosti s brazilskými prezidentskými volbami v roce 2018. Poslední nárůst pak souvisí s pandemií coronaviru v roce 2020, kdy můžeme vidět nejvyšší počet vyhledávání termínu fake news. (27)



Obr. 22 Popularita vyhledávání fake news na Googlu (26), poznámka značí vylepšený způsob sbírání dat

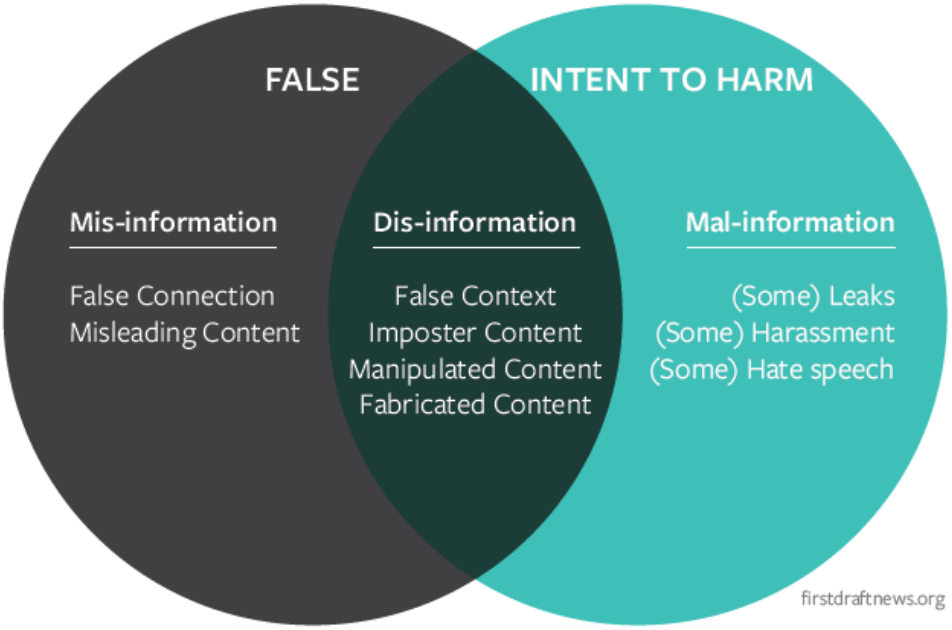
4.1 Co se považuje za falešné zprávy

Unesco a další organizace zabývající se touto problematikou termín falešné zprávy nepoužívají. Považují ho za příliš zpolitizovaný a často používaný proti institucím, jejichž zprávy se snaží daná osoba zdiskreditovat. Unesco ve své zprávě definuje tři kategorie falešných zpráv, lišící se úmyslem autora zprávy a tím, zda je zpráva nepravdivá či jen zavádějící. Grafické znázornění můžeme vidět na Obr. 23.

První kategorie je označována anglickým termínem „disinformation“, což můžeme přeložit jako dezinformace. Charakteristikou je, že autor zprávy ví, že daná informace není pravdivá. Jedná se tedy o úmyslnou lež s cílem přesvědčit audienci o něčem, co není pravda. Příkladem může být zpráva z období francouzských prezidentských voleb, kdy někdo vytvořil webovou stránku podobnou belgickému deníku *LeSoir* a uvedl na ní zprávu, že prezidentský kandidát Emanuel Macron je sponzorován Saudskou Arábií.

Druhá kategorie je označována anglickým termínem „misinformation“, který můžeme přeložit jako mylnou informaci. V tomto případě se autor zprávy domnívá, že zpráva je pravdivá. Úmyslem autora není způsobit škodu. Příkladem můžou být poplašné zprávy objevující se v krizových momentech, například při teroristických útocích. Často jsou sdíleny s dobrými úmysly, ale mohou zahltit či přetížit policii či jiné instituce.

Třetí kategorie je označována anglickým termínem „malinformation“. Tato zpráva je pravdivá, ale jejím cílem je někomu uškodit. Příkladem může být zveřejňování citlivých informací ze soukromí člověka, například jeho sexuální orientace. (28)



Obr. 23 Druhy falešných zpráv (28)

4.2 Typy falešných zpráv

Termín falešné zprávy může mít pro různé lidi rozdílný význam. V roce 2017 vyšla studie, která na základě 34 akademických článků o falešných zprávách definovala několik typů falešných zpráv.

Satirické a parodické články a pořady

Jednou z kategorií falešných zpráv je satira a parodie. Často jsou zaměřeny na současné události a ve své televizní podobě používají obrazový a zvukový materiál, který může připomínat zpravodajské vysílání. U satiry bývají zprávy zveličovány či jsou prezentovány s humorem, přičemž vypouštějí část informací. Parodie přímo používají vymyšlený obsah k přitáhnutí publika. Cílem zde sice není šířit falešné zprávy, ale pokud někdo neví, že daná stránka je satirická či parodická, může zprávě uvěřit. Velkým rozdílem oproti ostatním falešným zprávám je zde úmysl a vystupování autorů. Ti se neprezentují jako novináři, ale jako komici či baviči. Na svých stránkách také často uvádějí, že se nejedná o seriózní žurnalistiku. Bývá zde tedy porozumění mezi čtenářem a autorem, že se jedná o zprávu s účelem pobavit a nemá se brát vážně.

Účelově vytvořené falešné zprávy

Většina lidi si nejspíše pod pojmem falešné zprávy představí právě toto. Jedná se o zprávy, které nestojí na faktech, ale jsou zcela vymyšlené. Svým vzhledem se snaží kopírovat známé zpravodajské portály. Cílem autora často bývá přesvědčit čtenáře o něčem, co není pravda. Neobvyklá ale není ani finanční motivace. Články svým obsahem cílí na určitý okruh lidí s názory, které jsou v souladu s vyzněním článku. Autoři si snaží vybírat témata, která mezi lidmi rezonují a vzbuzují emoce. Šíření těchto článků hodně napomáhají sociální sítě, které umožnují sdílení dané informace velkému množství lidí. Jsou zaznamenány i případy, kdy autoři používají automatizované algoritmy tvářící se jako reální uživatelé k tomu, aby svému článku dodali kredibilitu a popularitu.

Manipulace s fotografickým materiálem

Se vzrůstající kvalitou softwaru, který umožnuje práci s fotkami, se začaly objevovat digitálně upravené fotografie, jejichž vyznění je úplně jiné než původní fotografie. Kromě upravených fotografií do této kategorie řadíme i fotografie, jež byly vytrženy z původního kontextu a použity znovu k jinému účelu.

Propaganda

Za propagandu se považuje zpráva vytvořená politickým subjektem s cílem ovlivnit veřejné mínění. Zpráva není nezaujatá a jejím cílem je přesvědčit. (29)

4.3 Identifikace falešných zpráv

S nárůstem počtu účelově vytvořených falešných zpráv v posledních letech vznikly i iniciativy, které se je snaží rozpoznat a minimalizovat jejich dopad a šíření. Jednou z institucí, která s falešnými zprávami bojuje jsou zpravodajské portály. Ti se snaží nejrozšířenější falešné zprávy demaskovat a ukázat, jak to bylo ve skutečnosti. Příkladem může být článek deníku New York Times, který v roce 2016 analyzoval vznik a šíření hodně sdílené zprávy na Twitteru o zaplacených protestujících. (30)

Příkladem české iniciativy může být web demagog, který ověřuje faktická vyjádření politické elity. Inspirací webu demagog byl americký portál politifact.com, který funguje od roku 2007 a za svou práci byl v roce 2009 oceněn Pulitzerovou cenou. Za zmínku stojí i stránky jako factcheck.org nebo Washington Post‘s Fact Checker.

Kromě ověřování jednotlivých zpráv existují i webové stránky, které evidují seznamy stránek, které často publikují falešné zprávy. Tyto seznamy se ale potýkají se dvěma problémy. Jedním z nich je fakt, že velká část stránek na těchto seznamech nepublikuje pouze falešné zprávy, ale určitý mix pravdivých a nepravdivých zpráv. Samotná přítomnost stránky na seznamu nám nedává jasnou odpověď, zda je daná zpráva pravdivá či nepravdivá. Druhý problém spočívá v nekompletnosti seznamu. Pro autory, kteří účelově šíří nepravdivé zprávy, není problém vytvořit novou stránku, která na seznamu není a pomocí fiktivních účtů na sociálních sítí ji rychle rozšířit. (31)

5 Návrh a realizace vlastních modelů pro detekci falešných zpráv

5.1 Zdrojová data

Zdrojem dat je platforma Kaggle[3]. Platforma Kaggle byla založena v roce 2010 Anthonym Goldbloomem a Jeremym Howardem. V roce 2017 přešla do vlastnictví Googlu, nyní Alphabetu. V současnosti nabízí několik služeb. V první řadě jsou to soutěže ve strojovém učení. Firma či jiná organizace vyvěsí na stránku problém, který potřebuje vyřešit, a odměnu pro autora nejlepšího řešení. Zúčastnit se může každý registrovaný uživatel. Kromě soutěží stránka slouží jako komunitní fórum pro lidi zajímající se o strojové učení a analýzu dat. Lze zde nalézt různé datové sady a podívat se, jak je různí autoři řešily a jakých výsledků se svými řešeními dosáhly. (32, 33)

Datová sada použitá pro tuto bakalářskou práci je z jedné komunitní soutěže, která proběhla v roce 2018. Soutěž se v překladu nazývá falešné zprávy a jejím cílem bylo vyvinout algoritmus schopný rozpoznat nepravdivé novinové články. Cílem je tedy klasifikovat daný článek do jedné ze dvou kategorií, bud jako pravdivý či jako nepravdivý. Novinové články jsou v anglickém jazyce.

Datová sada je rozdělena na trénovací a testovací data. Trénovací data jsou v jednom souboru a testovací data jsou zde rozdělena do dvou souborů, jeden z nich obsahuje data určená pro klasifikaci a druhý výsledky.

Trénovací data obsahují 20800 článků. Jsou zde rovnoměrně zastoupeny pravdivé a nepravdivé novinové články. Soubor obsahuje 5 atributů a jeho struktura je následující:

ID: Jednoznačný numerický identifikátor daného článku (0-20799)

Title: Titulek daného novinového článku

Author: Autor daného novinového článku

Text: Text daného novinového článku

Label: Numerické označení článku jako nepravdivého (1), či pravdivého (0)

Testovací data obsahují 5200 článků rovněž rovnoměrně rozdělené do skupin pravdivé a nepravdivé. Struktura je stejná jako u trénovacích dat s tím rozdílem, že atribut label je vyčleněn do samostatného souboru. Atribut ID je zde použit jako jednoznačný identifikátor spojující daný článek v jednom souboru s atributem label ve druhém souboru. (34)

[3] https://www.kaggle.com/

5.2 Předzpracování dat

Před klasifikací samotnou neuronovou sítí je třeba určit, která data a jakým způsobem budeme používat jako vstupy. Pro tuto bakalářskou práci byly z dostupných atributů vybrány dva. Těmi jsou texty novinových článků a nadpisy těchto článků. Atributy byly použity odděleně, to znamená, že stejná síť byla použita jak pro klasifikaci pomocí textů, tak pro klasifikaci pomocí nadpisů. Předpokladem zde je, že se texty pravdivých a falešných novinových článků liší. Odlišnosti by mohli být v tématech kterými se zabývají, v odlišných slovních spojeních či jednotlivých slovech, které se zde vyskytují. Neuronové sítě představené v této práci nedávají odpověď na otázku, čím se liší, pouze zda se za pomoci těchto atributů dají rozlišit.

V této kapitole je ve zkrácené podobě uváděna i použitá implementace. Všechen použitý kód lze nalézt na adrese uvedené v příloze A.

5.2.1 Čištění dat

Při bližší pohledu na data se ukázalo, že některé zprávy v datech mají chybějící atributy. Pro zjištění počtu chybějících textů byl použit kód ve Výpis kódu 1, který najde chybějící text a vypíše celkový počet takovýchto článků. Například u textů novinových článků jich 47 chybí.

```
1.  for(int i=0; i<texty.size(); i++) {
2.      if(texty.get(i).isEmpty()) {
3.          chybiText++;
4.      }
5.  System.out.println("chybí text v "+ chybiText + " případech");
```

Výpis kódu 1 Zjištění počtu chybějících textů (autor)

Jelikož podle prázdných textů nemůžeme nic klasifikovat, tyto záznamy bude potřeba odstranit.

Dále bylo třeba se vypořádat s texty, které jsou krátké a neodpovídají představě novinového článku složeného z vět. Kód ve Výpis kódu 2 nám vypíše texty, jež mají méně znaků než nastavená hodnota delkaTextu. To umožňuje udělat si představu o tom, zda dává smysl dané články zachovávat. Zároveň je vhodné vědět, kolik takových článků v datech existuje, tuto informaci nám podává řádek 6.

```
1.  if(texty.get(i).length() < delkaTextu) {
2.      System.out.println("text je kratší než " +delkaTextu+ " znaků " + i );
3.      System.out.println(texty.get(i));
4.      kratkyText++;
5.  }
6.  System.out.println("text kratší než "+ delkaTextu + " znaků v " + kratkyText +
    " případech");
```

Výpis kódu 2 Zjištění počtu krátkých textů (autor)

Pro lepší představu můžeme pět takových ukázkových článků vidět v Tab. 1. Jedná se spíše o náhodné věty či jenom slova, pro klasifikaci se jeví jako nevhodná, proto bude třeba i tyto příliš krátké záznamy odstranit.

ID článku	Text
196	`They got the heater turned up on high`
4234	`Good one Doc!`
18632	`Not likely.`
19648	`Obama's weakness is very dangerous.`
20418	`Guest Guest`

Tab. 1 Vybrané ukázky krátkých článků (autor)

Bližší pohled na počet článků, které jsou velmi krátké nám ukazuje Tab. 2. Jako určující atribut délky článku je zde použit počet znaků. Počty jsou kumulativní. Hranice potřebného počtu znaků, aby se podle článku dalo klasifikovat, byla stanovena na 100. Určitě by se dalo uvažovat o dalším zvýšení této hranice, pokud bychom chtěli pracovat pouze s koherentními novinovými články, které mají vnitřní strukturu a kontext.

Délka textu menší než	Trénovací data	Testovací data
0 znaků	40	7
20 znaků	170	44
50 znaků	236	71
100 znaků	455	105

Tab. 2 Počet krátkých textů (autor)

Nadpisů chybělo v datech více, bylo potřeba odstranit celkem 680 prázdných záznamů, což představuje přibližně 2,5% záznamů v celé datové sadě. Minimální délka pro klasifikaci pomocí nadpisů byla stanovena na 10 znaků. Kratší nadpisy nebyly použity.

Délka textu menší než	Trénovací data	Testovací data
0 znaků	558	122
5 znaků	561	123
10 znaků	575	125

Tab. 3 Počet krátkých nadpisů (autor)

Kromě chybějících či příliš krátkých záznamů se zde vyskytovalo i několik zpráv, které nebyly v anglickém jazyce. Jednalo se pravděpodobně o azbuku. Pro vypořádání se s těmito texty byl použit byl regulární výraz na řádku 3 ve Výpis kódu 3, který odstraní všechny znaky kromě písmen, číslic a mezer. Tímto také zbavíme texty interpunkčních znamének.

```
1.  for(int i=0; i< texty.size(); i++) {
2.      String veta = texty.get(i);
3.      String upravenaVeta = veta.replaceAll("[^a-zA-Z0-9\\s]", "");
4.      texty.set(i, upravenaVeta);
5.  }
```

Výpis kódu 3 Odstranění nežádoucích znaků pomocí regulárního výrazu (autor)

Po vymazání nežádoucích znaků se nám dále zvýší počet zpráv, které mají méně než 100 znaků u textů, respektive 10 znaků u nadpisů a nebudeme je dále používat. Celkový počet záznamů, které byly označeny jako nevhodné pro klasifikaci můžeme vidět v tabulce Tab. 4

	Nevhodné záznamy	
	Trénovací data	Testovací data
Texty	459	110
Nadpisy	600	139

Tab. 4 Počet nevhodných záznamů v datech (autor)

Počet záznamů, které budeme po předzpracování dat používat pro další klasifikaci, můžeme vidět v Tab. 5.

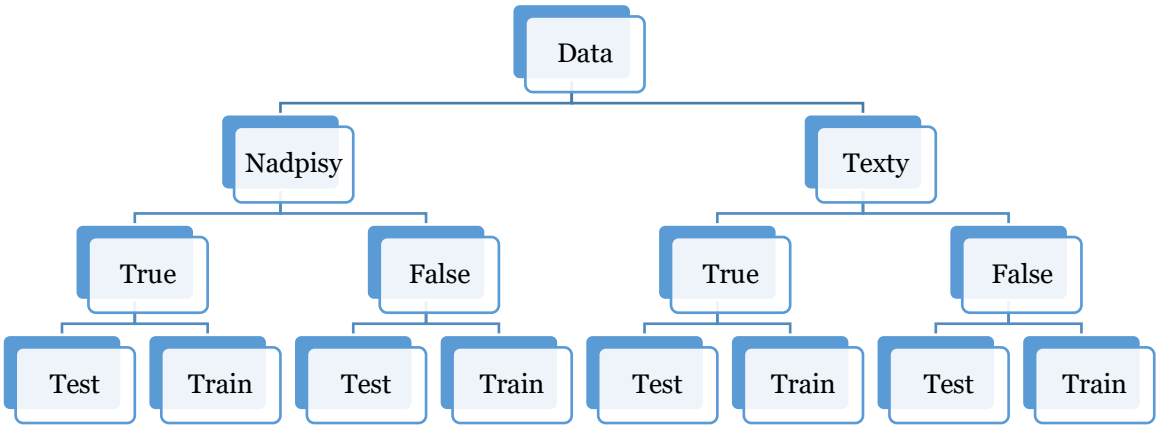
		Počet záznamů	
		Trénovací data	Testovací data
Texty	**Pravdivé**	9956	2752
	Falešné	10385	2338
Nadpisy	**Pravdivé**	9813	2730
	Falešné	10387	2331

Tab. 5 Počet záznamů po předzpracování dat (autor)

5.2.2 Příprava dat pro použití v Deeplearning4j

Pro použití dat neuronovou sítí pomocí knihovny Deeplearning4j musíme nejdřív vytvořit objekt, zvaný datasetIterator. Ten se stará o postupný přenos dat na vstupní vrstvu neuronové sítě. Tento objekt tudíž musí být přizpůsoben našim datům a zároveň síti, kterou chceme použít. Knihovna jich nabízí několik desítek již vytvořených, žádný ale nevyhovoval textovým datům ve formátu csv (comma separated values), ve kterém máme vstupní data.

Jako nejjednodušší cesta se ukázalo vytvořit z každého textu či nadpisu samostatný textový soubor, kde atribut ID bude jeho název a který bude zařazen do složky, jenž odpovídá jeho klasifikaci (atribut label). Výsledkem je tedy následující souborový systém, ukázaný na Obr. 24:



Obr. 24 Použitá adresářová struktura (autor)

Ve výsledných složkách jsou pak uloženy textové soubory, kde každý soubor představuje jednu zprávu. Na takto uložená data se již dá napasovat existující FileLabeledSentenceProvider.

O předzpracování dat uvedené v kapitole 5.2.1 a jejich převod do textových souborů se v práci stará třída ZpracováníDat. Výpis kódu 4 se stará o převedení zpráv, které dosahují zvolené délky (pro tuto práci tedy 100 u textů, respektive 10 u nadpisů) do textových souborů ve vybraných složkách.

Názvem nově vzniklých souborů je tedy ID a obsahem text. Zařazení do správné složky se určí podle atributu label, jak můžeme vidět na řádku 2 pro falešné, respektive na řádku 7 pro pravdivé.

```
1.  for(int i=0; i< labels.size(); i++) {
2.    if(Integer.parseInt(labels.get(i)) == 0 && texty.get(i).length() > delkaTextu
3.    {
4.    FileUtils.writeStringToFile(new File("path", String.valueOf(i)), texty.get(i));
5.    }
6.
7.    if(Integer.parseInt(labels.get(i)) == 1 && texty.get(i).length() > delkaTextu)
8.    {
9.    FileUtils.writeStringToFile(new File("path", String.valueOf(i)), texty.get(i))
10.   }
```

Výpis kódu 4 Vytvoření nových souborů a jejich zařazení do složek (autor)

5.3 Tvorba Modelů

V této práci byly pro klasifikaci použity hlavně dva modely, konvoluční a rekurentní neuronová síť. Abychom mohli tyto sítě použít, potřebujeme způsob, jakým převést textová data na data numerická. Pro převod dat do numerické podoby jsou zde použity word2Vec modely. Pro každou klasifikaci, myšleno klasifikaci pomocí textů i nadpisů byly použity 3 různé modely distribuované reprezentace slov. Jedním z nich je převzatý model od Googlu. Ten byl vytvořen v roce 2013 na novinových článcích obsahujících dohromady kolem 3 bilionů slov. Model obsahuje 3 miliony slov a frází, každé slovo je zde reprezentováno 300 dimenzionálním vektorem. Model je dostupný na stránkách Googlu[4].

Kromě toho byly z datové sady vytvořeny další 4 modely distribuované reprezentace dat, 2 z nadpisů, 2 z textů.

Modely jsou vytvářeny pomocí testovacích i trénovacích dat. Před modelováním jsou všechny znaky převedeny na malá písmena, což omezuje různorodost danou začátky vět, která není žádoucí.

Kód pro vytváření modelů můžeme vidět na Výpis kódu 5. Je zde několik důležitých parametrů. Prvním z nich je parametr minWordFrequency (řádek 4), který určuje, kolikrát se dané slovo musí v datech objevit, aby se pro něj vytvářel vektor. Dalšími parametry jsou layerSize (řádek 5) udávající počet dimenzí vektoru a parametr windowsSize (řádek 7), který říká, kolik slov se bude brát do kontextu.

```
1.  final Word2Vec model = new Word2Vec.Builder()
2.        .iterate(iterator)
3.        .tokenizerFactory(tokenizerFactory)
4.        .minWordFrequency(2)
5.        .layerSize(1)
6.        .epochs(10)
7.        .windowSize(5)
8.        .build();
```

Výpis kódu 5 Nastavení pro tvorbu Word2Vec modelů (autor)

[4] https://code.google.com/archive/p/word2vec/

Pro nadpisy je vytvořen jeden jednoduchý model, kde každé slovo je reprezentováno pouze vektorem s jednou dimenzí. Atribut layerSize, který udává dimenzionalitu vektoru je tedy nastaven na 1. Druhý použitý model je složitější a každé slovo je zde reprezentováno 32 dimezionálním vektorem. Jsou zde použity jednodušší modely, jedná se totiž o nízký počet slov, na kterém se modely trénují. Ze stejného důvodu je i atribut minWordFrequncy, který udává, kolikrát se slovo musí v textech vyskytnout, nastaven na nízkou hodnotu 2. Nastavení tohoto atributu určuje celkový počet slov v modelu, pro nadpisy obsahují modely 10908 slov.

Pro klasifikaci pomocí textů byl vytvořen jeden jednoduchý model distribuované reprezentace slov s parametrem layerSize 1, a jeden složitější, kde je tento parametr nastaven na 100. Protože se zde vyskytuje asi 80x více slov než u nadpisů, jsou zde použitá pouze slova, která se vyskytnou minimálně pětkrát. Vytvořené modely pro texty s tímto nastavením obsahují 56079 slov.

5.3.1 Rekurentní model

Rekurentní sítě se dají implementovat několika způsoby, pro náš problém se nejlépe hodí vrstva GravesLSTM[5]. Jedná se o implementaci LSTM modelu s peephole connections. Rozšířením je GravesBidirectionalLSTM[6], což je obousměrná implementace stejného modelu.

Pro klasifikaci pomocí těchto vrstev musíme jako poslední vrstvu mít RNNOutputLayer[7], která se stará o převod výsledný krok klasifikace do daných tříd. Mezi tyto vrstvy je možno vkládat libovolný počet dopředných vrstev.

Parametry LSTM sítí

Počet neuronů ve skrytých vrstvách se ukázal jako parametr, jehož vliv na úspěšnost sítí byl v desetinách procent. Vyzkoušeny byly modely s počtem 120, 256 a 750 neuronů. Z těchto nejlépe vycházel model s počtem 256 neuronů, proto jsou všechny představené rekurentní sítě právě s 256 neurony ve všech vrstvách. Největší zaznamenaný vliv počtu neuronů byl na čas nutný k trénování. Ten byl u sítě s 750 neurony skoro dvojnásobný oproti síti s 256. Pokud bychom chtěli minimalizovat dobu trénování, mohli bychom uvažovat o dalším snížení počtu neuronů v síti.

Počet dopředných vrstev v modelu měl zanedbatelný vliv na úspěšnost v první epoše. Se stoupajícím počtem epoch se ale výsledky začínaly rozcházet. Testovány byly tři sítě, s žádnou, jednou a dvěma skrytými vrstvami. Sítě s jednou či dvěma vrstvami ukazovaly

---

[5] https://deeplearning4j.org/api/latest/org/deeplearning4j/nn/conf/layers/GravesLSTM.html
[6] https://deeplearning4j.org/api/latest/org/deeplearning4j/nn/conf/layers/GravesBidirectionalLSTM.html
[7] https://deeplearning4j.org/api/latest/org/deeplearning4j/nn/layers/recurrent/RnnOutputLayer.html

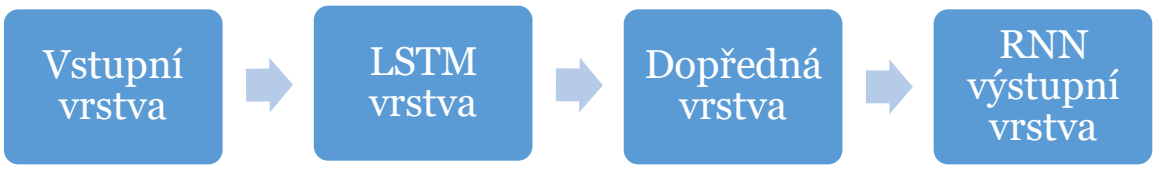
větší tendenci zlepšovat výsledky na trénovacích datech, ne ale na datech testovacích, kde se úspěšnost naopak zhoršovala. Přidání výpadku řešilo tento problém jen částečně. Zde představené sítě jsou s jednou dopřednou vrstvou.

Pro LSTM vrstvy se jako aktivační funkce doporučuje hyperbolický tangens[8]. Ten je použit i zde u všech LSTM vrstev. Při pokusu ho nahradit funkcí Relu přestala síť fungovat. U dopředných vrstev je použita funkce Relu a u výstupní vrstvy funkce softmax.

Zde jsou postupně představeny tři různé sítě, každá se třemi různými způsoby zpracování vstupů pomocí vnoření.

LSTM síť

První síť má čtyři vrstvy se strukturou zobrazenou na Obr. 25.



Obr. 25 Struktura LSTM sítě (autor)

Postup Vytvoření modelu můžeme vidět ve Výpis kódu 6. Model LSTM sítě má dva základní stavební kameny. Prvním je vstupní LSTM vrstva, kde počet vstupů záleží na parametru vectorSize, který je daný použitým modelem distribuované reprezentace slov. Můžeme vidět na řádku 2. Druhým je RNN výstupní vrstva, kde je počet výstupů daný počtem tříd, do kterých chceme data klasifikovat. Můžeme vidět na řádku 17. Ostatní parametry se dají

```
1.              .layer(new GravesLSTM.Builder()
2.                      .nIn(vectorSize)
3.                      .nOut(256)
4.                      .activation(Activation.TANH).build())
5.
6.
7.              .layer(new DenseLayer.Builder()
8.                      .nIn(256)
9.                      .nOut(256)
10.                     .activation(Activation.RELU)
11.                     .build())
12.
13.
14.             .layer(new RnnOutputLayer.Builder().activation(Activation.SOFTMAX)
15.                     .lossFunction(LossFunctions.LossFunction.MCXENT)
16.                     .nIn(256)
17.                     .nOut(2)
18.                     .build())
```

Výpis kódu 6 Vytvoření rekurentní sítě (autor)

[8] https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-using-keras-f8e9ed76f046

libovolně nastavovat, je třeba brát v potaz to, že počet vstupů se musí vždy rovnat počtu výstupů u předchozí vrstvy.

LSTM síť s výpadkem

Jedná se o stejnou síť s jediným rozdílem, a to je přidání výpadku. Ten je přidán do LSTM i dopředné vrstvy. Jeho hodnota je nastavena na 0,5. Nastavení výpadku můžeme vidět na řádku 5 ve Výpis kódu 7. Stejný postup je použit i u dopředné vrstvy.

```
1.  .layer(new GravesLSTM.Builder()
2.                      .nIn(vectorSize)
3.                      .nOut(256)
4.                      .activation(Activation.TANH)
5.                      .dropOut(0.5)
6.                      .build())
```

Výpis kódu 7 Přidání výpadku (autor)

BidirectionalLSTM

Oproti předchozímu případu se kód této sítě liší v jediném řádku, a to je použití jiného konstruktoru pro tvorbu LSTM vrstvy. Můžeme vidět na řádku 1 ve Výpis kódu 8, kde je použit GravesBidirectionalLSTM oproti GravesLSTM v předchozím případě. Funkčně se tato vrstva od předchozí liší tím, že běží obousměrně, uchovává tak nejenom informace o slovech, které jsou před daným slovem, ale i o slovech, které jsou za ním. Tato implementace by měla být schopna lépe porozumět kontextu (35). Pro tuto síť bude dále používána zkratka BLSTM.

```
1.  .layer(new GravesBidirectionalLSTM.Builder()
2.          .nIn(vectorSize)
3.          .nOut(256)
4.          .dropOut(0.5)
5.          .build())
```

Výpis kódu 8 Vytvoření BLTSM sítě (autor)

Výsledky LSTM sítě

V Tab. 6 můžeme vidět dosaženou úspěšnost klasifikace. Nejvyšší dosažená úspěšnost je zvýrazněna červeně, jedna pro klasifikaci pomocí textů a druhá pro klasifikaci pomocí nadpisů. Jak můžeme z tabulky vidět, klasifikace pomocí nadpisů dosahuje lepší úspěšnosti než klasifikace pomocí textů. Při použití složitějších distribuovaných modelů reprezentace slov stabilně narůstala úspěšnost klasifikace na trénovacích datech. Tento trend můžeme pozorovat jak při klasifikaci pomocí nadpisů, tak i při klasifikaci pomocí textů. Zvýšená úspěšnost se nepřenesla na testovací data, došlo tedy k přeučení. Nejlepších výsledků tedy tato síť dosahovala při reprezentaci slov distribuovaným modelem vektorem s jednou dimenzí. Kromě toho můžeme při klasifikaci pomocí nadpisů sledovat, že vzrůstající počet epoch má negativní vliv na úspěšnost u testovacích dat. U textů je trend spíše opačný.

Pro sloupec udávající dobu potřebnou k trénování a otestování výsledků je nutno dodat, že testování probíhalo na 6-ti jádrovém procesoru Ryzen 5 s frekvencí 3400 Mhz a 16GB Ram paměti. Všechny proběhlé testy používaly stejný hardware. Vzhledem k použitému iterátoru bereme z každé kategorie stejný počet dat, to znamená že náhodná klasifikace by dala úspěšnost 50%. Přebytečná data nebyla použita.

Vstupní Data	**Dimenzionalita vstupního vektoru**	**Typ Dat**	**Úspěšnost v %**				**Čas (min)**
			Epocha				
			1	2	3	4	
Nadpisy	**1**	Testovací	75	68	67	67	5
		Trénovací	80	85	86	86	
	32	Testovací	64	62	62	63	6
		Trénovací	91	90	91	92	
	300	Testovací	68	63	63	62	10
		Trénovací	88	91	94	95	
Texty	**1**	Testovací	54	67	67	69	80
		Trénovací	52	60	61	62	
	100	Testovací	58	60	66	60	98
		Trénovací	72	77	62	87	
	300	Testovací	61	61	62	62	97
		Trénovací	84	91	95	98	

Tab. 6 Výsledky LSTM sítě (autor)

Výsledky LSTM sítě s výpadkem

Při pohledu na Tab. 7 můžeme vidět, že přidání výpadku mělo malý vliv na celkovou úspěšnost. K největší změně došlo u vlivu počtu epoch. Nyní již nedochází ke snižování úspěšnosti u klasifikace pomocí nadpisů s rostoucím počtem epoch. Je patrná také celková nižší úspěšnost na trénovacích datech, hlavně u klasifikace pomocí textů.

Vstupní Data	**Dimenzionalita vstupního vektoru**	**Typ Dat**	**Úspěšnost v %**				**Čas (min)**
			Epocha				
			1	2	3	4	
Nadpisy	**1**	Testovací	74	70	73	70	4
		Trénovací	79	77	76	77	
	32	Testovací	67	66	64	67	7
		Trénovací	92	92	90	92	

	300	Testovací	63	62	61	63	10
		Trénovací	87	88	89	92	
Texty	**1**	Testovací	56	60	57	61	79
		Trénovací	57	59	57	59	
	100	Testovací	51	50	55	55	90
		Trénovací	54	54	55	55	
	300	Testovací	62	62	62	63	87
		Trénovací	72	92	95	96	

Tab. 7 Výsledky LSTM sítě s výpadkem (autor)

Výsledky BLSTM sítě

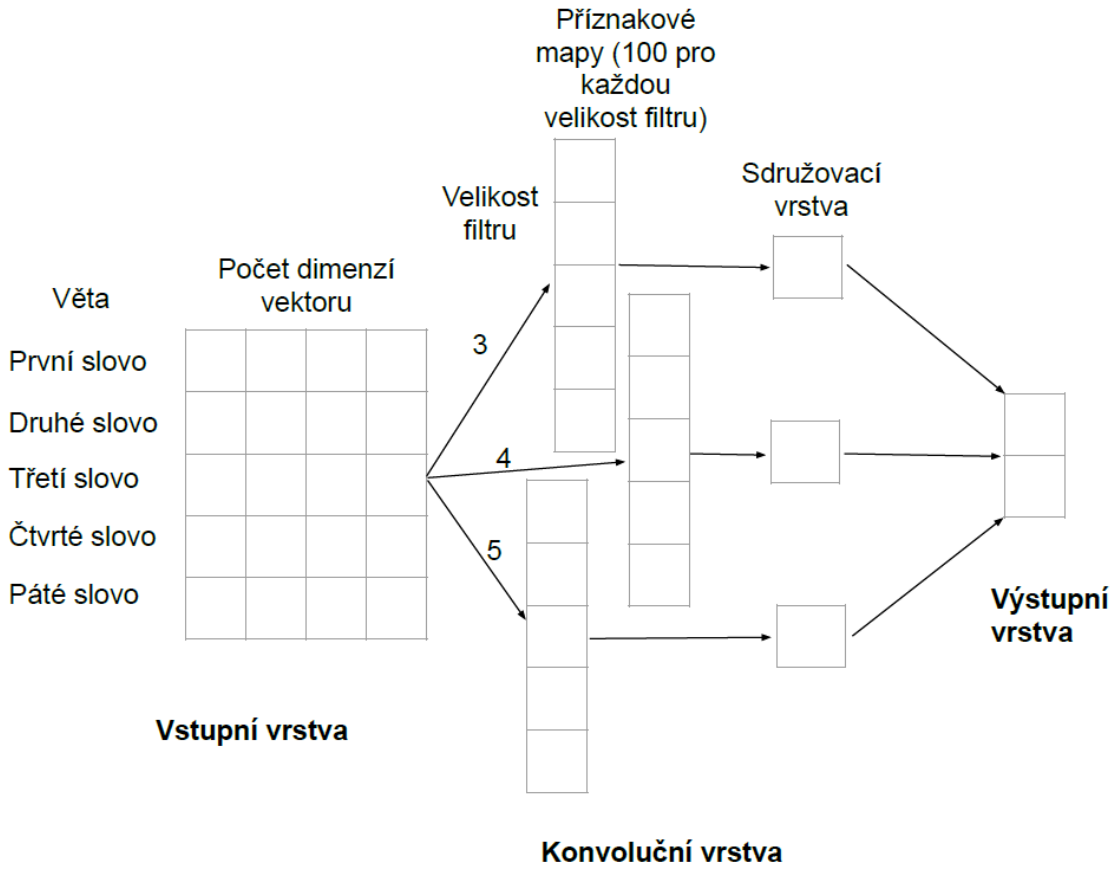
Jak můžeme vidět v Tab. 8, při použití této sítě na klasifikaci pomocí nadpisů měl počet epoch zanedbatelný vliv na úspěšnost. I když se zde nejlepší úspěšnost nerovná LSTM síti bez výpadku, je zde vidět větší stabilita výsledků napříč vstupy i epochami. Také můžeme pozorovat nejvyšší nároky na výpočetní výkon, časová náročnost vzrostla průměrně o pětinu, vysvětlení lze hledat ve složitější architektuře sítě.

Vstupní Data	**Dimenzionalita vstupního vektoru**	**Typ Dat**	**Úspěšnost v %**				**Čas (min)**
			Epocha				
			1	2	3	4	
Nadpisy	**1**	Testovací	75	75	75	75	5
		Trénovací	80	80	80	80	
	32	Testovací	68	68	68	68	7
		Trénovací	91	91	91	91	
	300	Testovací	66	64	66	63	11
		Trénovací	85	88	89	90	
Texty	**1**	Testovací	65	59	65	65	100
		Trénovací	61	58	61	64	
	100	Testovací	54	62	61	62	114
		Trénovací	62	75	86	90	
	300	Testovací	61	62	62	63	140
		Trénovací	82	84	92	95	

Tab. 8 Výsledky BLSTM sítě (autor)

5.3.2 Konvoluční model

V této práci byl použit model, který je zjednodušeně zobrazen na Obr. 26. Můžeme ho rozdělit do tří částí. První z nich je, stejně jako u LSTM modelu, vstup pomocí vnoření slov. Ten je zobrazen dvourozměrnou maticí, kde každý řádek reprezentuje jedno slovo, počet sloupců pak odpovídá počtu dimenzí použitého distribuovaného modelu reprezentace slov. Následuje konvoluční vrstva a na ní navázaná sdružovací vrstva. Třetí částí je dopředná výstupní vrstva. Mezi konvoluční a výstupní vrstvu je možno vložit libovolný počet dopředných vrstev.



Obr. 26 Náčrt použitého konvolučního modelu neuronové sítě (autor)

Parametry konvolučních sítí

Prvním parametrem při tvorbě modelu je velikost filtru. Pro klasifikaci textů se doporučuje velikost 2-10. Pro tuto práci byly použity tři, a to o velikosti 3,4 a 5. Při zkoušce několika různých velikostí a počtu se neukázalo, že by tento parametr měl velký vliv na úspěšnost. Pro každou velikost filtru je ještě třeba nastavit, kolik příznakových map bude vytvořeno.

Doporučený počet pro počáteční experimenty je 100[9] pro každou velikost filtru, větší počet nepřinesl zvýšenou úspěšnost, vliv měl hlavně na dobu testování, proto byl použit i zde.

Na příznakové mapy je navázána sdružovací vrstva s funkcí sdružování dle maxima, které by mělo dosahovat nejlepších výsledků[10]. Použity zde byly dvě sítě, jedna s přidanou dopřednou vrstvou a výpadkem.

CNN síť

Vytvoření konvoluční sítě s více filtry vyžaduje pojmenování každé vytvořené vrstvy, jelikož jedna vrstva nevstupuje přímo do vrstvy následující. Pojmenování každé vrstvy nám dává možnost definovat při jejím vytváření, co bude vstupem. Příklad můžeme vidět na řádcích 7, 12 a 17 ve Výpis kódu 9, kde všude používáme stejné vstupy do tří vytvořených vrstev. Tyto vrstvy představují použité velikosti filtrů, například vrstva "cnn3" na řádku 3 představuje použití filtru o velikosti 3*počet dimenzí vstupního vektoru (zde reprezentován atributem vectorSize). Vrstvy "cnn3", "cnn4" a "cnn5" jsou poté spojeny, což můžeme vidět na řádku 18. Spojení vrstev může být provedeno před nebo po použití sdružovací vrstvy, zde je tedy použita první varianta. Následuje již zmiňovaná sdružovací vrstva s výpadkem, řádek 19-22. Jako poslední je vytvořena výstupní dopředná vrstva, můžeme vidět na řádku 23.

```
1.  .graphBuilder()
2.                  .addInputs("input")
3.                  .addLayer("cnn3", new ConvolutionLayer.Builder()
4.                          .kernelSize(3,vectorSize)
5.                          .stride(1,vectorSize)
6.                          .nOut(cnnLayerFeatureMaps)
7.                          .build(), "input")
8.                  .addLayer("cnn4", new ConvolutionLayer.Builder()
9.                          .kernelSize(4,vectorSize)
10.                         .stride(1,vectorSize)
11.                         .nOut(cnnLayerFeatureMaps)
12.                         .build(), "input")
13.                 .addLayer("cnn5", new ConvolutionLayer.Builder()
14.                         .kernelSize(5,vectorSize)
15.                         .stride(1,vectorSize)
16.                         .nOut(cnnLayerFeatureMaps)
17.                         .build(), "input")
18.                 .addVertex("merge", new MergeVertex(), "cnn3", "cnn4", "cnn5")
19.                 .addLayer("globalPool", new GlobalPoolingLayer.Builder()
20.                         .poolingType(globalPoolingType)
21.                         .dropOut(0.5)
22.                         .build(), "merge")
23.                 .addLayer("out", new OutputLayer.Builder()
24.                         .lossFunction(LossFunctions.LossFunction.MCXENT)
25.                         .activation(Activation.SOFTMAX)
26.                         .nOut(2)
27.                         .build(), "globalPool")
```

Výpis kódu 9 Vytvoření konvoluční sítě (autor)

[9] https://machinelearningmastery.com/best-practices-document-classification-deep-learning/

[10] https://machinelearningmastery.com/best-practices-document-classification-deep-learning

CNN síť s výpadkem

Tato síť se od předchozí liší přidáním výpadku do vrstev “cnn3“, “cnn4“ a “cnn5“. Kromě toho je přidána jedna dopředná vrstva s výpadkem, kterou můžeme vidět ve Výpis kódu 10.

```
1.  .addLayer("dense", new DenseLayer.Builder()
2.    .nOut(cnnLayerFeatureMaps/2)
3.    .activation(Activation.RELU)
4.    .dropOut(0.5)
5.    .build(), "globalPool")
```

Výpis kódu 10 Konvoluční síť s výpadkem (autor)

Výsledky CNN sítě

Úspěšnost konvoluční sítě můžeme vidět v Tab. 9. Počet trénovacích epoch zde hrál velmi nízkou roli. Pro trénování by zde stačila jedna epocha. Největší vliv na výsledek zde má způsob reprezentace vstupních dat. Nejlepších výsledků dosahuje předtrénovaný word2Vec model od Googlu. Můžeme zde také vidět propastný rozdíl mezi úspěšností na trénovacích a testovacích datech. I když model dosahuje na trénovacích datech úspěšnosti 99%, na datech testovacích je to pouze kolem 63%.

Pro lepší porozumění výsledkům je třeba uvést, že zde, na rozdíl od LSTM sítí nebylo nutné mít stejný počet zpráv v kategorii pravdivých a falešných zpráv. Náhodná úspěšnost pro trénovací data je tak přibližně 51% a pro testovací data přibližně 54%.

Vstupní Data	**Dimenzionalita vstupního vektoru**	**Typ Dat**	**Úspěšnost v %**				**Čas (min)**
			Epocha				
			1	2	3	4	
Nadpisy	**1**	Testovací	56	56	56	55	1
		Trénovací	69	69	69	69	
	32	Testovací	58	58	57	58	1
		Trénovací	77	77	77	77	
	300	Testovací	64	65	65	65	6
		Trénovací	97	98	98	98	
Texty	**1**	Testovací	54	49	56	53	6
		Trénovací	56	55	54	59	
	100	Testovací	61	61	60	62	28
		Trénovací	90	95	95	96	
	300	Testovací	63	62	62	62	50
		Trénovací	98	96	99	99	

Tab. 9 Výsledky CNN sítě (autor)

Výsledky CNN sítě s výpadkem

Jak můžeme vidět v Tab. 10, přidání výpadku snížilo rozdíl mezi úspěšností klasifikace na trénovacích a testovacích datech. Výsledky u klasifikace pomocí nadpisů zůstali podobné jako v předchozím případě. U klasifikace pomocí textů ale došlo k velkému propadu, zvláště při použití vlastních modelů distribuované reprezentace dat.

Vstupní Data	Dimenzionalita vstupního vektoru	Typ Dat	Úspěšnost v %				Čas (min)
			Epocha				
			1	2	3	4	
Nadpisy	1	Testovací	52	53	53	54	2
		Trénovací	64	67	65	67	
	32	Testovací	58	58	59	58	2
		Trénovací	73	75	75	74	
	300	Testovací	68	68	67	67	7
		Trénovací	91	92	94	94	
Texty	1	Testovací	Vše v jedné kategorii				9
		Trénovací					
	100	Testovací	57	54	56	56	26
		Trénovací	65	60	64	64	
	300	Testovací	61	61	60	62	47
		Trénovací	90	85	95	92	

Tab. 10 Výsledky CNN sítě s výpadkem (autor)

Porovnání LSTM a CNN sítí

U klasifikace pomocí nadpisů si vedly lépe rekurentní LSTM sítě. Dosahovaly o několik procentních bodů lepších výsledků, konkrétně kolem 75% oproti 68% u konvolučních. U klasifikace pomocí textů byli výsledky podobné, dvě konvoluční a dvě LSTM sítě dosáhly úspěšnosti kolem 62-63%. Jedinou výjimkou byla nejjednodušší LSTM síť s úspěšností 69%. Největší rozdíl ale můžeme sledovat u úspěšnosti při použití různých modelů reprezentované distribuce slov. Zatímco LSTM sítě dosahovaly nejlepších výsledků s natrénovanými modely reprezentované distribuce slov, u konvolučních sítí to bylo přesně naopak, zde si vždy vedl nejlépe předtrénovaný word2Vec model od Googlu.

Co se týká časové náročnosti na trénování, konvoluční sítě vyžadovaly poloviční až třetinový čas.

I když konvoluční síť vypadá na pohled komplikovaněji, náročnější na vytvoření v Javě byly LSTM sítě. Důvodem byl neexistující iterator, zatímco pro konvoluční síť se dal použít již

existující CnnSentenceDataSetIterator[11]. I když knihovna Deeplearning4J má v nabídce iterátorů několik desítek, pro úlohu klasifikace pomocí LSTM sítí nebyl žádný nalezen. Vytvoření způsobu, jakým dostat data do sítě je poté mnohem náročnější proces než samotné vytvoření sítě.

5.3.3 Porovnání výsledků s vybranými metodami dolování dat

Pro porovnání s jinými metodami strojového učení byl vybrán nástroj Weka[12], konkrétně metoda rozhodovacích stromů (J48) a metoda naivního Bayese. Weka je open- source software vyvinutý na univerzitě Waikato na Novém Zélandu. Nabízí širokou škálu dalších nástrojů pro strojové učení, pro porovnání byli zvoleny tyto dva.

Před klasifikací pomocí nástroje Weka musíme nejdříve předzpracovat data. K tomu byla použita funkce StringToWordVector, která převede slova do numerické podoby. Za zmínku stojí, že je zde použita pouze vybraná podmnožina slov, konkrétní nastavení bylo 1000 pro každou třídu, toto se rovná necelých 1500 pro nadpisy a kolem 1200 pro texty. Pro porovnání, u trénovaných word2Vec modelů to bylo kolem deseti tisíc u nadpisů a přibližně šedesát šest tisíc u textů. I když tedy modely v nástroji Weka využívají stejná vstupní data, klasifikují pouze podle menšího množství vybraných atributů.

V Tab. 11 můžeme vidět výsledky při klasifikaci pomocí rozhodovacího stromu (J48). Úspěšnost u textů zde dosahuje podobných výsledků jako u rekurentních a konvolučních modelů. Úspěšnost u nadpisů dosahuje horších výsledků než LSTM, ale podobných jako konvoluční modely.

Také zde je vidět velký rozdíl mezi úspěšností na trénovacích a testovacích datech. Za povšimnutí stojí nižší doba nutná pro trénování. Ta je podobná u času potřebnému pro klasifikaci neuronovými sítěmi pomocí nadpisů, ale průměrně několikrát nižší než u klasifikaci neuronovými sítěmi pomocí textů. K tomu je nutno dodat, že těchto výsledků bylo dosaženo s použitím pouze jednoho jádra a necelých 2GB Ram paměti, tedy s přibližně šestinovými nároky na výpočetní výkon. Hardwarové omezení je takto dáno použitým nástrojem Weka, proto zde nelze dobu nutnou ke klasifikaci porovnávat přímo.

	Nadpisy	**Texty**
Úspěšnost testovací	66%	64%
Úspěšnost trénovací	94%	99%
Celkový čas	8 min	8 min

Tab. 11 Klasifikace pomocí rozhodovacího stromu (autor)

[11]https://deeplearning4j.org/api/latest/org/deeplearning4j/iterator/CnnSentenceDataSetIterator.html

[12] https://www.cs.waikato.ac.nz/ml/weka/

Druhou vybranou metodou byl naivní bayes, výsledky můžeme vidět v Tab. 12. Úspěšnost je zde o něco nižší než u rozhodovacích stromů či neuronových sítí. Doba pro klasifikaci se zde ale počítá na vteřiny, čímž dalece překonává ostatní metody.

	Nadpisy	**Texty**
Úspěšnost testovací	64%	62%
Úspěšnost trénovací	96%	97%
Celkový čas	3 sec	30 sec

Tab. 12 Klasifikace pomocí naivního Bayes (autor)

5.4 Shrnutí výsledků modelů klasifikace falešných zpráv

5.4.1 Možnosti vylepšení úspěšnosti

Vliv délky použitého textu na úspěšnost

Pro představené modely byla použita maximální délka textu 256 znaků. Toto nastavení nemělo žádný vliv na úspěšnost u nadpisů, jejichž délka tento limit málokdy přesáhla. Texty ale ve výjimečných případech dosahují délky necelých 5000 znaků, v některých případech tak byla odstraněna velká část dat z novinového článku. Jelikož je trénování na takovém množství dat náročné na čas, vliv tohoto parametru byl otestován pouze na jedné konvoluční a jedné rekurentní síti. Konktrétně se jedná o BidirectionalLSTM model se 100-dimenzionálním word2Vec modelem a o CNN síť s 300-dimenzionálním word2Vec modelem od Googlu.

Při trénování BLSTM s maximální délkou textu 5000 došlo k nárustu úspěšnosti, konkrétně o 11%. Výsledky s tímto nastavením můžeme vidět v Tab. 13, pro porovnání v Tab. 14 výsledky s maximální délkou textu 256. Z výsledků vyplývá, že by bylo lepší provádět trénování LSTM sítí na delším textu. Vyšší úspěšnost je vykoupena nárůstem doby potřebné pro trénování, doba potřebná na trénování jedné epochy zde vyskočila z 28 minut na 330 minut, tedy více než 13x.

Epocha	**1**	**2**
Úspěšnost testovací	**73%**	**63%**
Úspěšnost trénovací	**82%**	**74%**
Celkový čas	762 min	

Tab. 13 BLSTM - maximální délka textu 5000 (autor)

Epocha	1	2	3	4
Úspěšnost testovací	54%	62%	61%	62%
Úspěšnost trénovací	62%	75%	86%	90%
Celkový čas	114 min			

Tab. 14 BLSTM - maximální délka textu 256 (autor)

U konvoluční sítě se rozdíl neprojevil, dosahovala s maximální délkou textu 5000 znaků stejných výsledků jako s maximální délkou textu 256 znaků.

5.4.2 Porovnání s jinými modely umělých neuronových sítí

V rámci komunitní soutěže, ze které pochází tato sada, bylo na stránkách kaggle publikováno třináct různých postupů použitých ke klasifikaci falešných zpráv. Dva z nich používaly pro klasifikaci neuronové sítě, v jednom případě proběhla klasifikace pomocí nadpisů, ve druhém pomocí textů. V obou případech jsou kódy představeny pomocí jazyka Python.

Klasifikace pomocí textů s použitím LSTM

V případě klasifikace pomocí textů autor[13] použil neuronovou síť ve Výpis kódu 11 s úspěšností 68%. Největším rozdílem oproti modelu rekurentní sítě vytvořenému v Javě je reprezentace vstupních dat. Jak můžeme vidět na řádku 1, zde je použita vrstva Embedding, která je přímo začleněna do modelu neuronové sítě, zatímco u LSTM modelu vytvořeného v Javě byl použit natrénovaný word2Vec model. Kromě toho se jedná o podobnou síť, jaká byla použita v představeném modelu LSTM síť s výpadkem.

```
1. lstm_model.add(layer = Embedding(input_dim = max_features, output_dim = 120,))
2. lstm_model.add(layer = LSTM(units = 120, dropout = 0.2, recurrent_dropout = 0.2
   ))
3. lstm_model.add(layer = Dropout(rate = 0.5))
4. lstm_model.add(layer = Dense(units = 120,  activation = 'relu', ))
5. lstm_model.add(layer = Dropout(rate = 0.5))
6. lstm_model.add(layer = Dense(units = len(set(y)),  activation = 'sigmoid'))
```

Výpis kódu 11 Ukázka porovnávané LSTM sítě (36)

[13] https://www.kaggle.com/jsvishnuj/fakenews-detection-using-lstm-neural-network/notebook

Pokud bychom chtěli stejnou neuronovou síť replikovat v Javě, narazíme na několik problémů. I když výše použitá vrstva embedding[14] existuje i v Javě, na stránkách Deeplearning4j se pro práci s přirozeným jazykem doporučuje použít word2Vec model, není tedy jasné, s jakým iterátorem či v jakém kontextu se má používat. Iterátory použité pro konvoluční i rekurentní modely přímo vyžadují word2Vec model. Při pokusu o vyhledání případů jejího použití mimo oficiální kanály pak narazíme pouze na příklady, kdy byla použita pro exekuci kódu z kerasu (knihovna pro tvorbu neuronových sítí v Pythonu). Její použití v Javě by bylo časově náročné. Bez příkladu užití či dokumentace by nalezení správného kódu na její vytvoření a použití pravděpodobně trvalo dlouho. Druhým problémem je použití rekurentního výpadku, které můžeme vidět na řádku 3. Ten není v knihovně Deeplearning4j podporován[15].

Klasifikace pomocí nadpisů s použitím BLSTM

Druhá představená neuronová síť od jiného autora[16] klasifikovala falešné zprávy pomocí nadpisů. S touto neuronovou sítí dosáhl úspěšnosti 94%. První vrstva sítě je stejně jako v předchozím případě reprezentována vrstvou embedding. Kromě toho můžeme vidět jednu obousměrnou LSTM vrstu (řádek 4) a jednu dopřednou vrstvu (řádek 7). Obě používají výpadek s hodnoutou 0,5. Tato síť je nejvíce podobná síti BidirectionalLSTM s 32-dimenzionální word2Vec modelem s úspěšností klasifikace 68%, autorovi se tedy podařilo dosáhnout mnohem lepších výsledků. To může být dáno jiným způsobem předzpracování dat či použitím vnoření s pouze 4000 slovy.

```
1.  model.add(Embedding(output_dim = output_length, input_dim = token_num, input_l
    ength = data_length))
2.  model.add(Dropout(dropout))
3.
4.  model.add(Bidirectional(LSTM(lstm_dim), merge_mode = 'sum'))
5.  model.add(Dropout(dropout))
6.
7.  model.add(Dense(units = 256, activation = 'relu'))
8.  model.add(Dropout(dropout))
9.
10. model.add(Dense(units = 1, activation = 'sigmoid'))
```

Výpis kódu 12 Ukázka porovnávané BLTSM sítě (37)

5.4.3 Shrnutí výsledků

Rekurentní sítě dosahovaly nejlepších výsledků na testovacích datech s distribuovanými modely reprezentace dat, které měly menší počet dimenzí. Vysvětlení nejspíše leží v menší

[14] https://deeplearning4j.org/api/latest/org/deeplearning4j/nn/layers/feedforward/embedding/EmbeddingSequenceLayer.html
[15] https://github.com/eclipse/deeplearning4j/issues/4614
[16] https://www.kaggle.com/thisjack/fake-news-detection-with-bltsm

náchylnosti k přeučení při použití takto řešených vstupů. Trend u trénovacích dat byl totiž přesně opačný, kde nejvyšší úspěšnosti dosahovaly sítě z předtrénovaným google word2Vec modelem. Rozdíl mezi úspěšností na trénovacích a testovacích datech se nepovedlo odstranit ani přidáním výpadku či použitím regularizačních metod. Při klasifikaci pomocí textů výsledky utrpěly nastavenou maximální délkou textu, trénování by bylo lepší provádět nad celými texty, rychle zde ale roste výpočetní náročnost.

Konvoluční sítě dosahovaly horších výsledků než rekurentní. Docházelo zde velmi rychle k přeučení, které se ani zde nepovedlo odstranit. Nejvyšší úspěšnosti dosahovaly stabilně s použitím předtrénovaných google word2Vec vektorů. Oproti rekurentním modelům vyžadují k trénování třetinový až poloviční čas.

Klasifikace pomocí rozhodovacích stromů dosahovala podobných výsledků jako konvoluční sítě. Pro klasifikaci pomocí nadpisů byla úspěšnost o dvě procenta horší než nejlepší konvoluční model. U klasifikace pomocí textů byla úspěšnost o procento lepší. I u rozhodovacích stromů bylo velkým problémem přeučení. Naivní bayes dosahoval ze všech použitých metod nejhorších výsledků. Jeho velkou výhodou je ovšem čas, za který výsledků dosahuje.

Nejlepších výsledků dosahovaly modely, kde docházelo nejméně k přeučení. To trápilo většinu vytvořených modelů a znemožňovala složitějším architekturám neuronových sítí klasifikovat falešné zprávy s větší úspěšností.

Pokud bychom porovnávaly obtížnost vytvoření v Javě pomocí knihovny Deeplearning4j, tak jednodušší na vytvoření byl konvoluční model. Rozdíl byl již v existujícím iterátoru, zatímco pro rekurentní model musel být vytvořen. Obecně práci s knihovnou Deeplearning4j komplikovala neúplná dokumentace a malé množství příkladů použití jednotlivých metod a architektur pro problém klasifikace přirozeného jazyka.

Závěr

Bakalářská práce měla pět cílů. První cíl, představení oblasti dolování z dat (data mining), je naplněn v první kapitole. V této kapitole je oblast dolování z dat stručně zařazena do konceptu dobývání dat z databází a jsou zde představeny základní úlohy a metody evaluace.

Druhý cíl, představení umělých (hlubokých) neuronových sítí obecně a v prostředí Javy, je naplněn v druhé a třetí kapitole. Ve druhé kapitole práce popisuje princip fungování umělého neuronu, na který navazuje vysvětlení perceptronu. Druhá kapitola dále popisuje cyklické a acyklické neuronové sítě, používané aktivační funkce a proces učení. Třetí kapitola začíná stručnou historií hlubokých neuronových sítí. Poté je zde představená knihovna Deeplearning4j. Následují témata věnovaná rekurentním a konvolučním architekturám neuronových sítí a možnosti jejich implementace pomocí knihovny Deeplearning4j.

Třetí cíl, představení aplikační oblasti falešných zpráv, je naplněn ve čtvrté kapitole. Tato kapitola se zabývá vymezením termínu falešné zprávy a popisuje, co všechno tento termín zahrnuje. Kromě toho jsou zde nastíněny vzniklé způsoby, jak se různé organizace snaží falešné zprávy identifikovat.

Prvním praktickým cílem bakalářské práce, tedy čtvrtým v pořadí bylo navrhnout a realizovat model umělé neuronové sítě v Javě pro klasifikační úlohu z aplikační oblasti falešných zpráv. Realizovány byly tři rekurentní a dva konvoluční modely. Konvoluční modely dosáhly nejvyšší úspěšnosti klasifikace 68%, rekurentní modely si vedli lépe a nejlepší z nich dosáhl nejvyšší úspěšnosti klasifikace 75%.

Převod textových dat na numerická byl řešen pomocí word2Vec modelů. Jednalo se o tři word2Vec modely natrénované na použitých datech a jeden předtrénovaný word2Vec model od Googlu. U rekurentních neuronových sítí fungovaly lépe word2Vec modely natrénované na datech ke klasifikaci. Naopak u konvolučních neuronových sítí dosáhl model lepších výsledků, pokud byla data reprezentována předtrénovaným Google word2Vec modelem.

Pokud se blíže podíváme na výsledky rekurentních neuronových sítí, tak lepších výsledků dosahovaly při použití word2Vec modelu, kde bylo každé slovo reprezentováno pouze jedno-dimenzionálním vektorem. Při použití těchto modelů distribuované reprezentace dat docházelo nejméně k přeučení, takže i při nižších úspěšnostech klasifikace na trénovacích datech dosahovaly nejvyšší úspěšnosti na testovacích datech.

U konvolučních sítí bylo použití předtrénovaných Google word2Vec modelů jednoznačně lepším řešením. Dosahovaly nejvyšší úspěšnosti na trénovacích i testovacích datech. Tyto modely ovšem také nejvíc trápil problém přeučení. To bylo velkým problémem u většiny použitých modelů a nepodařilo se odstranit ani použitím výpadku či jiných regularizačních metod.

Pokud se podíváme na náročnost vytvoření modelů v Javě s použitím knihovny Deeplearning4j, tak jednodušší na vytvoření se ukázaly konvoluční modely. Na rozdíl od rekurentních modelů zde již existoval vytvořený iterátor. Časově nejnáročnějším aspektem při návrhu modelu bylo právě vytvoření způsobu, jakým reprezentovat vstupní data. Celkově se při práci s knihovnou Deeplearning4j vyskytlo několik problémů. Největším z nich je nekompletní dokumentace, která ve spojení s chybějícími příklady užití dělá implementované metody náročné na použití. Kromě toho chybělo několik metod používaných v jiných jazycích.

Co se týká časové náročnosti na trénování, pak méně náročné jsou konvoluční modely, které vyžadují na trénování poloviční až třetinový čas oproti rekurentním modelům.

Druhým praktickým cílem bakalářské práce bylo porovnat a vyhodnotit dosažené výsledky oproti dalším vybraným metodám z dobývání znalostí z databází. Vybranými metodami byly rozhodovací stromy a naivní bayes. Klasifikace byla provedena pomocí nástroje Weka. Rozhodovací stromy dosáhly podobných výsledků jako konvoluční neuronové sítě, ale nedosáhly na výsledky rekurentních neuronových sítí. Naivní bayes dosáhl nejnižší úspěšnosti, ale s nejkratším časem potřebným na klasifikaci.

Modely neuronových sítí představené v této bakalářské práci trpěly problémem přeučení, které znemožňovalo dosáhnout vyšší úspěšnosti. V další práci by bylo vhodné zjistit, které metody regularizace by bylo vhodné zařadit, aby klasifikace dosahovala vyšší úspěšnosti.

Použitá literatura

1. VOSKOGLOU, Christina. What is the best programming language for Machine Learning? *Towardsdatascience.com*. [cit. 2020-05-8]. Dostupné z: https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7
2. BERKA, Petr. *Dobývání znalostí z databází*. Praha: Academia, 2003. ISBN 80-200-1062-9.
3. FAYAD, Usama, Gregory PIATETSKY-SHAPIRO a Padhraic SMYTH. *Knowledge Discovery and Data Mining:Towards a Unifying Framework* [online]. 2000 [cit. 2020-05-8].
4. AMATRIAIN Xavier. What's the relationship between machine learning and data mining? *Medium.com*. [cit. 2020-05-8]. Dostupné z: https://medium.com/@xamat/what-s-the-relationship-between-machine-learning-and-data-mining-8c8675966615
5. GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. *Deep learning* [online]. Cambridge: MIT Press, [2016] [cit. 2020-05-8]. Adaptive computation and machine learning (MIT Press). ISBN 978-026-2035-613.
6. GUPTA Prashant. Decision Trees in Machine Learning. *Towardsdatascience.com*. [cit. 2020-05-8]. Dostupné z: https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052
7. CHAUHAN Gaurav. All about Naive Bayes. *Towardsdatascience.com*. [cit. 2020-05-8]. Dostupné z: https://towardsdatascience.com/all-about-naive-bayes-8e13cef044cf
8. MATEMATICKABIOLOGIE.CZ. Logický neuron.[online]. [cit. 2020-05-8]. Dostupné z: https://portal.matematickabiologie.cz/index.php?pg=analyza-a-modelovani-dynamickych-biologickych-dat--matematicke-modely-v-biologii--matematicke-modely-neuronu--logicky-neuron
9. SIMPLILEARN.COM. What is Perceptron: A Beginners Tutorial for Perceptron. [online]. [cit. 2020-05-8]. Dostupné z: https://www.simplilearn.com/what-is-perceptron-tutorial
10. VOLNÁ, Eva. NEURONOVÉ SÍTĚ 1. *Osu.cz*. Ostrava: Ostravská univerzita v Ostravě, 2008 [cit. 2020-05-8]. Dostupné z: https://www1.osu.cz/~volna/Neuronove_site_skripta.pdf
11. MISSINGLINK.AI. 7 Types of Neural Network Activation Functions: How to Choose? [online]. [cit. 2020-05-8]. Dostupné z: https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/
12. AGARAP, Abien FRED. *Deep Learning using Rectified Linear Units (ReLU)* [online]. 2018 [cit. 2020-05-8].
13. READTHEDOCS.IO. Loss Functions. *Ml-cheatsheet.readthedocs.io* [online]. [cit. 2020-05-8]. Dostupné z: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html
14. SUGOMORI, Yusuke, Bostjan KALUZA, Fabio SOARES a Alan SOUZA. *Deep Learning: Practical Neural Networks with Java*. Packt Publishing, 2017. ISBN 9781788470315.

15. HINTON, Geoffrey, Simon OSINDERO a Yee-Whye TEH. *A fast learning algorithm for deep belief nets* [online]. Toronto, 2006 [cit. 2020-05-8]. Dostupné z: https://www.cs.toronto.edu/~hinton/absps/fastnc.pdf
16. DEEPLEARNING4J. *Deep Learning for Java* [online]. [cit. 2020-05-8]. Dostupné z: https://deeplearning4j.org/
17. KONDUIT.AI. MultiLayerNetwork And ComputationGraph. *Deeplearning4j.konduit.ai* [online]. [cit. 2020-05-8]. Dostupné z: https://deeplearning4j.konduit.ai/getting-started/tutorials/multilayernetwork-and-computationgraph
18. NIELSEN, Michael. Deep learning. *Http://neuralnetworksanddeeplearning.com* [online]. [cit. 2020-05-8]. Dostupné z: http://neuralnetworksanddeeplearning.com/chap6.html
19. SKYMIND, INC. Cnn Sentence Classification Example. *Github.com* [online]. [cit. 2020-05-8]. Dostupné z: https://github.com/eclipse/deeplearning4j-examples/blob/master/dl4j-examples/src/main/java/org/deeplearning4j/examples/convolution/sentenceclassification/CnnSentenceClassificationExample.java
20. WANG, Xin & Liu, Yuanchao & Sun, Chengjie & Wang, Baoxun & Wang, Xiaolong. (2015). Predicting Polarities of Tweets by Composing Word Embeddings with Long Short-Term Memory. 1343-1353. 10.3115/v1/P15-1130
21. KONDUIT.AI. Sea Temperature Convolutional LSTM. *Deeplearning4j.konduit.ai* [online]. [cit. 2020-05-8]. Dostupné z: https://deeplearning4j.konduit.ai/getting-started/tutorials/sea-temperature-convolutional-lstm
22. SRIVASTAVA Nitish, Geoffre HINTON, Alex KRIZHEVSKY, Ilya SUTSKEVER a Ruslan SALAKHUTDINOV. Dropout: A Simple Way to Prevent Neural Networks fromOverfitting. *Journal of Machine Learning Research*, 15(1), 1929-1958. 2014.
23. DEEPLEARNING4J.ORG. Dropout. [online]. [cit. 2020-05-8]. Dostupné z: https://deeplearning4j.org/api/latest/org/deeplearning4j/nn/conf/dropout/Dropout.html
24. PATHMIND.COM. A Beginner's Guide to Word2Vec and Neural Word Embeddings. [online]. [cit. 2020-05-8]. Dostupné z: https://pathmind.com/wiki/word2vec
25. KONDUIT.AI. Word2Vec. *Deeplearning4j.konduit.ai* [online]. [cit. 2020-05-8]. Dostupné z: https://deeplearning4j.konduit.ai/language-processing/word2vec
26. GOOGLE.COM. Fake news. [online]. [cit. 2020-05-8]. Dostupné z: https://trends.google.com/trends/explore?date=all&q=fake%20news
27. POSETTI Julia, Alice MATTHEWS. A short guide to the history of 'fake news' and disinformation. *Wan-ifra.org* [online]. [cit. 2020-05-8]. Dostupné z: https://blog.wan-ifra.org/sites/default/files/field_blog_entry_file/A%20Short%20Guide%20to%20History%20of%20Fake%20News%20and%20Disinformation_ICFJ%20Final.pdf
28. IRETON, Cherilyn a Julie POSETTI. *Journalism, fake news & disinformation: handbook for journalism education and training* [online]. Paříž: UNESCO Publishing, 2018 [cit. 2020-05-8]. ISBN 978-92-3-100281-6. Dostupné z: https://unesdoc.unesco.org/ark:/48223/pf0000265552
29. EDSON C. Tandoc Jr., Zheng Wei Lim & Richard Ling (2017): *Defining "FakeNews"*, Digital Journalism, DOI: 10.1080/21670811.2017.1360143

30. MAHESWARI, Sapna. How Fake News Goes Viral: A Case Study. *Nytimes.com*. [cit. 2020-05-8]. Dostupné z: https://www.nytimes.com/2016/11/20/business/media/how-fake-news-spreads.html
31. SHARNA, Karishma, Feng QIAN, He JIANG, Natali RUCHANSKY, Ming ZHANG a Yan LIU. *Combating Fake News: A Survey on Identification and Mitigation Techniques* [online]. 2019 [cit. 2020-05-8].
32. OMAR, Munira. Kaggle For Beginners : Getting Started. *Towardsdatascience.com*. [cit. 2020-05-8]. Dostupné z: https://towardsdatascience.com/kaggle-for-beginners-getting-started-75decb43c0c0
33. WIKIPEDIA.ORG. Kaggle. [online]. [cit. 2020-05-8]. Dostupné z: https://en.wikipedia.org/wiki/Kaggle
34. KAGGLE.COM. Fake News. [online]. [cit. 2020-05-8]. Dostupné z: https://www.kaggle.com/c/fake-news/data
35. BROWNLEE, Jason. How to Develop a Bidirectional LSTM For Sequence Classification in Python with Keras. *Machinelearningmastery.com*. [cit. 2020-05-8]. Dostupné z: https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/
36. JAIKRISNAN, Sabari. FakeNews Detection using LSTM Neural Network. *Kaggle.com*. [cit. 2020-05-10]. Dostupné z: https://www.kaggle.com/jsvishnuj/fakenews-detection-using-lstm-neural-network/notebook
37. THISJACK, Fake News Detection with BLTSM. *Kaggle.com*. [cit. 2020-05-10]. Dostupné z: https://www.kaggle.com/thisjack/fake-news-detection-with-bltsm

Přílohy

Příloha A: Kompletní zdrojový kód realizovaných modelů (elektronická příloha)

Projekt včetně všech použitých metod a tříd lze nalézt na GitHubu na adrese: https://github.com/paran5/BP/tree/master/BP_NN. Postup spuštění použitého kódu je uvedený ve složce Readme. Jsou přiložená i předzpracovaná data a natrénované word2Vec modely, lze tedy spustit rovnou neuronovou síť.