Univerzita Hradec Králové
Přírodovědecká fakulta
Katedra matematiky

Neuronové sítě a jejich aplikace

Bakalářská práce

Autor: Tomáš Kratochvíl
Studijní program: Aplikovaná matematika
Studijní obor: Finanční a pojistná matematika
Vedoucí práce: doc. Mgr. Dušan Bednařík, Ph.D.

Zadání bakalářské práce

Autor:	**Tomáš Kratochvíl**
Studium:	S17AM010BP
Studijní program:	B1103 Aplikovaná matematika
Studijní obor:	Finanční a pojistná matematika
Název bakalářské práce:	Neuronové sítě a jejich aplikace
Název bakalářské práce v Aj:	Neural networks and their applications
Zásady pro vypracování:	Cílem práce je teoretický popis neuronové sítě, jejích základních typů a metod tréninku na vzorku dat. Práce by měla též obsahovat některé vybrané aplikace neuronových sítí nejen z oblasti financí. Cílem je porovnat u daného problému výsledky predikce obdržené z neuronové sítě s ohledem na typ sítě, její architektuře a dalších parametrech modelu.
Garantující pracoviště:	Katedra matematiky, Přírodovědecká fakulta
Vedoucí práce:	doc. Mgr. Dušan Bednařík, Ph.D.
Oponent:	Mgr. Tomáš Zuščák, Ph.D.
Datum zadání práce:	23. 1. 2019
Datum odevzdání práce:	17. 7. 2020

Poděkování

Tímto bych rád poděkoval svému vedoucímu, doc. Mgr. Dušanu Bednaříkovi, Ph.D., za cenné rady, odborné připomínky a zajímavé diskuze v průběhu vedení této práce.

Prohlášení:

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a že jsem v seznamu použité literatury uvedl všechny prameny, z kterých jsem vycházel.

V Hradci Králové dne 17. 7. 2020

.....

Tomáš Kratochvíl

Anotace

KRATOCHVÍL, Tomáš. *Neuronové sítě a jejich aplikace*. Hradec Králové, 2020. Bakalářská práce. Univerzita Hradec Králové, Přírodovědecká fakulta, Katedra matematiky.

Tato bakalářská práce se zabývá úvodem do problematiky neuronových sítí. V práci jsou popsány základní typy neuronových sítí, některé algoritmy a metody využívané při jejich návrhu a trénování. Většina z představených metod je pak aplikována na analýzu sentimentu recenzí produktů z internetového obchodu Amazon.com.

Klíčová slova

Neuronové sítě, klasifikace textů, analýza sentimentu, algoritmus zpětného šíření, gradientní sestup, optimalizace hyper-parametrů

Annotation

KRATOCHVÍL, Tomáš. *Neural networks and their applications*. Hradec Králové, 2020. Bachelor thesis. University of Hradec Králové, Faculty of Science, Department of Mathematics.

This bachelor thesis focus on introduction to neural networks. In this thesis we describe common types of neural networks, some algorithms and methods used for their training and design. Most of them we show on sentiment analysis of reviews from e-shop Amazon.com.

Keywords

Neural networks, text classification, sentiment analysis, backpropagation, gradient descent, hyper-parameter optimization

Obsah

Úvod

V dnešní době narůstá množství vyprodukovaných dat exponenciálně. Za posledních pár let vzniklo víc dat, než za celou dosavadní historii lidstva. Takovéto velké objemy dat mohou obsahovat pro firmy celkem užitečné informace, které jim mohou pomoci s rozvojem a zlepšením podpory zákazníků a marketingu. S rostoucím počtem generovaného textu začíná být obtížné pro firmu procházet všechny komentáře a hodnocení adresované na jejich produkty nebo služby a na základě jejich analýzy je zlepšovat. Jedním z řešení je pak použít program, který dokáže automaticky najít negativní recenze, na které se potom firma může zaměřit a zlepšit tak své služby. Jedním z možných řešení je použití technik z oblasti strojového učení, které se hodí na problémy, kde je algoritmizace úloh nemožná nebo příliš složitá.

Tato bakalářská práce se zabývá umělými neuronovými sítěmi, které se v současnosti pro úlohy zpracování přirozeného jazyka velmi často používají. Jsou zde popsány nejpoužívanější typy neuronových sítí: plně propojené, konvoluční a rekurentní neuronové sítě. Jsou zde představeny nejpoužívanější algoritmy pro nalezení parametrů neuronové sítě a také vybrané metody používané pro vylepšení. Dále jsou zde popsány některé metody vhodné pro převádění textu do číselné reprezentace. Následně je v praktické části této práce provedena analýza sentimentu recenzí různých produktů z internetového obchodu Amazon.com. Na této úloze jsou zde vyzkoušeny všechny představené typy neuronových sítí a většina představených metod.

Cílem této práce je tedy poskytnout teoretický úvod do problematiky neuronových sítí a způsobu jejich trénování. Dále na analýze sentimentu recenzí porovnat dosažené výsledky v závislosti na typu, parametrech sítě a použitých metod.

Pro vytvoření obrázků bylo využito balíčku TikZ [27] a knihovny Matplotlib [17]. Praktická část je provedena v programovacím jazyce Python3 s využitím služby Google Colaboratory [14]. V této službe má uživatel zdarma možnost využít výkonné grafické karty, pomocí kterých je trénování neuronových sítí mnohem rychlejší. Napsané skripty jsou postavené na knihovnách Keras [12] a Tensorflow [1], které poskytují vysokoúrovňové API pro návrh a trénování neuronových sítí. Dále byla využita především knihovna NLTK [6] pro zpracování přirozeného jazyka, knihovna scikit-learn [10] pro manipulaci s trénovacími množinami a například knihovna Gensim [23] pro manipulaci se slovními vektory.

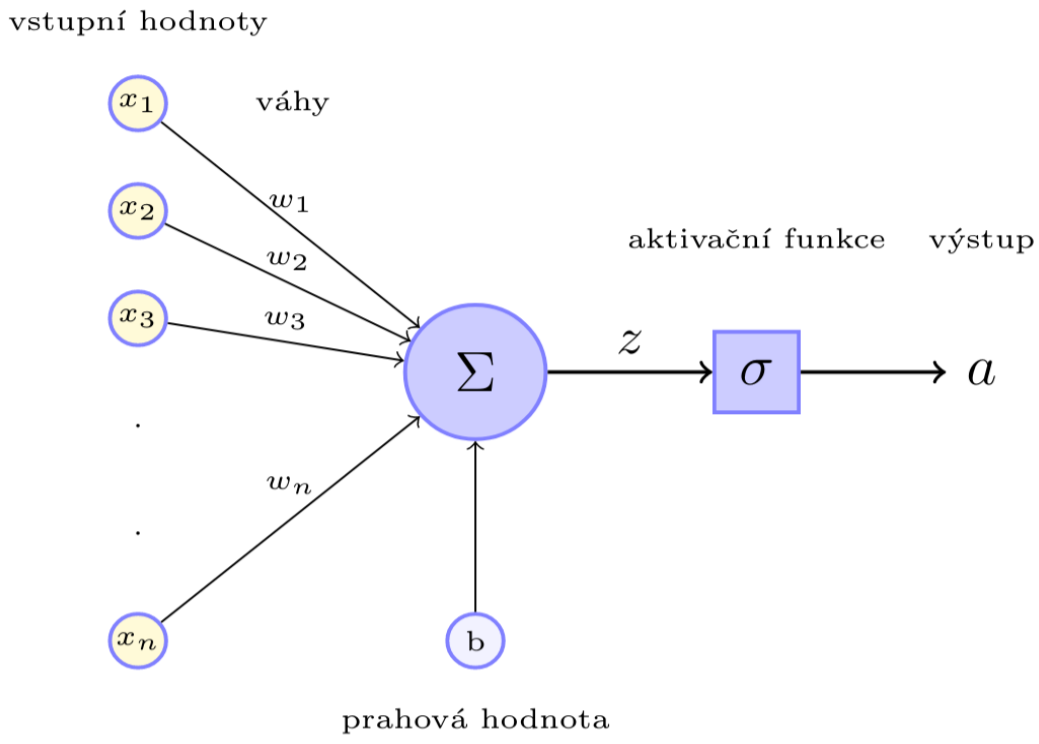
1 Umělá neuronová síť

Obsah této kapitoly vychází z literatury [21].

Umělá neuronová síť je výpočetní model inspirovaný biologickými neuronovými sítěmi se schopností naučit se složité nelineární problémy, které je příliš složité algoritmicky popsat. Neuronové sítě vynikají nad ostatními metodami strojového učení především ve schopnosti rozpoznávání a úpravě obrázků, ale i při dalších složitých problémech jako je překlad textu z jednoho jazyka do jiného. Základem umělé neuronové sítě je umělý neuron.

1.1 Umělý neuron

Do umělého neuronu vstupují reálná čísla $x_1, x_2, \ldots, x_n$, která reprezentují nějakou informaci z vnějšího okolí nebo výstupy z jiných neuronů. Neuron pak podle těchto přijatých hodnot vyšle vlastní signál (vypočte svoji výstupní hodnotu), který poslouží jako vstup do dalších neuronů nebo bude tvořit výstup (nebo jeho část) neuronové sítě. Každá ze vstupních hodnot má určitý vliv na výstupní hodnotu neuronu. Tento vliv je vyjádřen jednotlivými váhami, kterými se vstupní hodnoty násobí. Součet těchto násobků je neuronem zpracován a na základě jeho velikosti je pomocí tzv. aktivační funkce vypočtena výstupní hodnota neuronu. Tato aktivační funkce většinou nabývá hodnoty blízké nule, dokud signál nepřekročí určitou mez. K váženému součtu vstupních hodnot se proto ještě přičítá hodnota, která simuluje velikost této meze. Mluvíme o tzv. prahové hodnotě. O váženém součtu vstupních hodnot a prahové hodnoty můžeme přemýšlet jako o celkovém, neuronem přijatém signálu. Schéma umělého neuronu je znázorněno na obrázku č. 1.



Obrázek 1: Schéma umělého neuronu

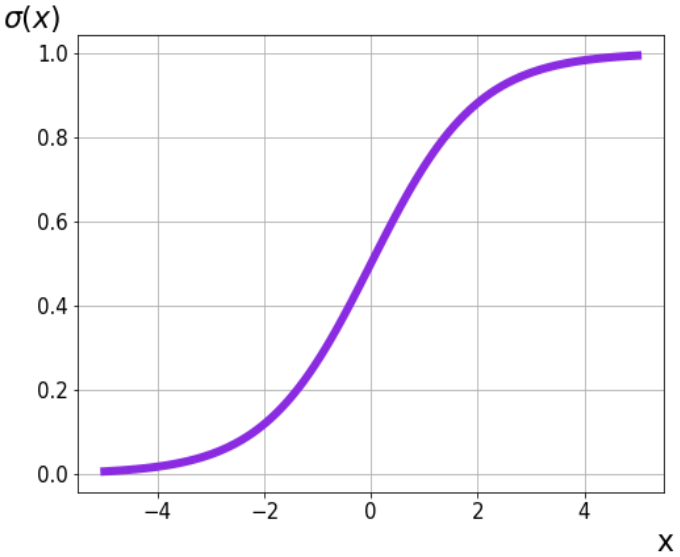
1.2 Aktivační funkce

Aktivační funkce je důležitým prvkem neuronu, který určuje jeho chování. Při učení neuronové sítě hledáme optimální váhy a prahové hodnoty tak, aby neuron či neuronová síť dokázali co nejlépe vyřešit danou úlohu. Aktivační funkce zůstávají v průběhu učení stejné a jejich volba bude záviset na typu úlohy a architektuře neuronové sítě. Takovéto prvky nebo parametry, které se v průběhu trénování nemění a které musíme zvolit při návrhu sítě, se nazývají hyper-parametry. Jedním z možných způsobů jejich hledání se budeme zabývat později.

Mezi klasické aktivační funkce patří logistická funkce (sigmoida). Její předpis je následující:

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}, \quad \text{kde } x \in \mathbb{R}. \tag{1.1}$$

Název tato funkce dostala podle tvaru její křivky připomínající tvar písmene S nebo jeden z tvarů malého řeckého písmene sigma ς (druhý tvar tohoto písmene je σ). Její graf je znázorněn na obrázku č. 2.



Obrázek 2: Funkce sigmoida

Výstup neuronu s touto aktivační funkcí lze vypočítat podle následujícího vztahu:

$$a = \sigma\left(\sum_{i=1}^{n} x_i w_i + b\right), \tag{1.2}$$

kde $n \in \mathbb{N}$ udává počet vstupních hodnot. Výraz uvnitř funkce je vhodné označit samostatnou proměnnou, nejen kvůli tomu, že vyjadřuje jakýsi celkový, neuronem přijatý signál, ale také nám to zjednoduší zápis v následujících výpočtech.

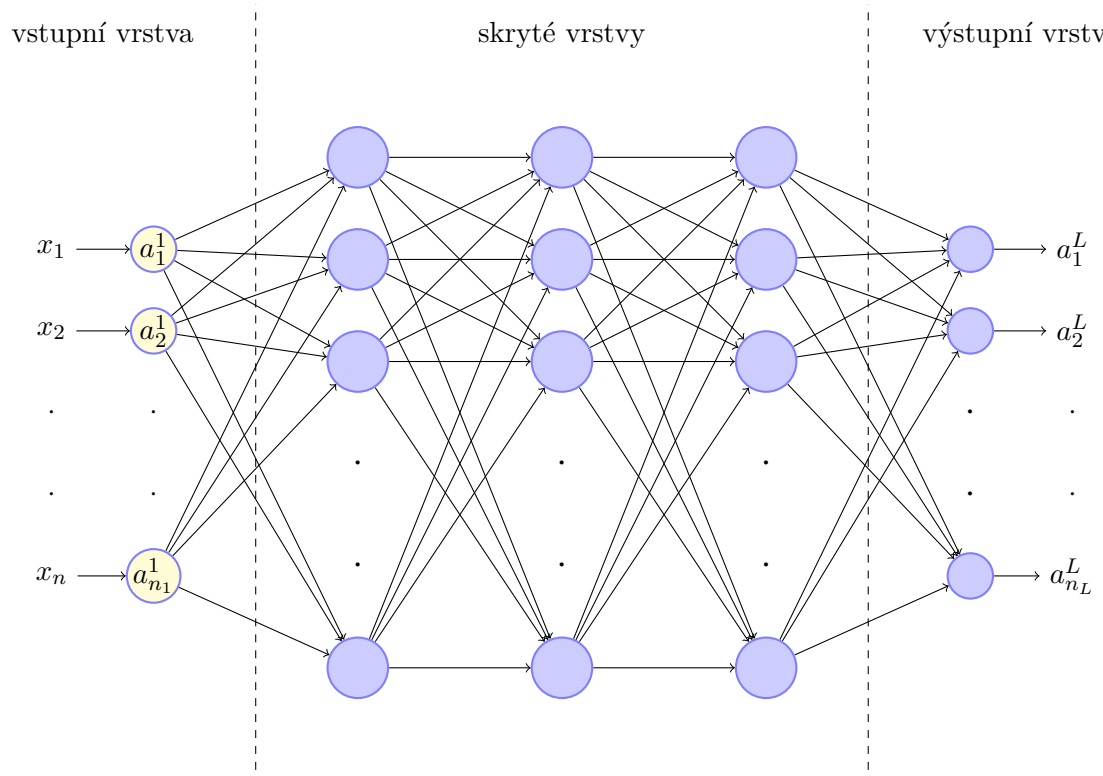
Označme tedy:

$$z = \sum_{i=1}^{n} x_i w_i + b. \tag{1.3}$$

Mezi další velmi často používané aktivační funkce patří hyperbolický tangens a rectified linear unit (ReLU). Prozatím si ale vystačíme s logistickou funkcí.

1.3 Plně propojená neuronová síť

Plně propojená neuronová síť se skládá ze tří částí: vstupní vrstvy, skryté vrstvy a výstupní vrstvy. Schéma je zobrazeno na obrázku č.3. Vstupní vrstva je první vrstvou v síti. Tato vrstva není složená z neuronů, ale z jednotlivých vstupních hodnot, které jsou předány nezměněny neuronům v první skryté vrstvě. Skryté vrstvy jsou vrstvy neuronů mezi vstupní a výstupní vrstvou. Název skryté vrstvy nemá z hlediska matematiky žádný význam. Tento název pravděpodobně vznikl z toho, že neuronové sítě připomínají černé skříňky, kterým se předhodí vstup a dostaneme výstup. Během tohoto procesu není moc jasné, co se uvnitř děje za abstrakci. Výstup každého neuronu v jedné vrstvě je použit jako vstup pro každý neuron v následující vrstvě, proto se tato architektura nazývá plně propojená. Poslední vrstva, výstupní vrstva, je opět složena z neuronů. Jejich výstup je výstupem celé neuronové sítě.



Obrázek 3: Schéma plně propojené neuronové síťě

Zavedeme si označení w_{jk}^l pro váhu j-tého neuronu v l-té vrstvě náležící pro výstup k-tého neuronu v předchozí vrstvě. Prahovou hodnotu j-tého neuronu v l-té vrstvě budeme značit b_j^l. Dále n_l bude značit počet neuronů v l-té vrstvě. Výstup j-tého neuronu v l-té vrstvě označíme a_j^l a vypočítáme následovně:

$$a_j^l = \sigma\left(\sum_{k=1}^{n_{l-1}} w_{jk}^l a_k^{l-1} + b_j^l\right). \tag{1.4}$$

Označíme si také celkový přijatý signál j-tého neuronu v l-té vrstvě:

$$z_j^l = \sum_{k=1}^{n_{l-1}} w_{jk}^l a_k^{l-1} + b_j^l. \tag{1.5}$$

2 Trénování neuronové sítě

2.1 Chybová funkce

I přes to, že nás primárně zajímá například počet správně předpovězených kategorií, není moc výhodné snažit se přímo tento počet maximalizovat. Malé změny vah a prahových hodnot nemusí způsobit žádnou změnu v predikci kategorií a my nezjistíme, jestli jsme síť zlepšili nebo naopak zhoršili. Pro trénování neuronové sítě je mnohem lepší použít spojitě diferencovatelnou funkci, která nám bude měřit velikost chyby předpovědí neuronové sítě. Tam pak poznáme, jestli malá změna zlepšila hodnotu této funkce nebo ne. Tuto funkci budeme nazývat chybovou funkcí a budeme se snažit minimalizovat její hodnotu. Při minimalizaci této funkce pak dochází ke zvýšení počtu správně předpovězených kategorií [21].

Hodnotu chybové funkce budeme počítat pro určitou trénovací množinu a budeme se jí snažit snížit pomocí nějakého optimalizačního algoritmu. Trénovací množina se skládá z dvojic $(X_1, y_1), (X_2, y_2), \ \ldots \ , (X_N, y_N)$, kde X_i je vstup a y_i je požadovaný výstup. Jako trénovací vstup si můžeme představit například text recenze nějakého produktu převedený do vektoru nebo matice. Požadovaným výstupem může být kategorie této recenze vyjádřená číslem nebo vektorem. Tvary těchto vstupů a výstupů mohou být velmi různé. Jeden vstup může být složen například z několika různých vektorů a různých matic. Důležité však je, aby ho dokázala síť správně zpracovat. Trénovací množinu si pak můžeme představit jako příklady recenzí a jejich kategorií, na kterých chceme síť natrénovat. Výstup (předpověď) sítě pro vstup X_i budeme značit a_i.

Jednou z funkcí, kterou můžeme použít jako chybovou funkci, je střední kvadratická chyba:

$$C = \frac{1}{N} \sum_{i=1}^{N} \|y_i - a_i\|^2, \tag{2.1}$$

kde a_i závisí na váhách, prahových hodnotách a vstupu X_i. Střední kvadratická chyba je jednou z funkcí, která se používá především pro regresní úlohy. Protože se budeme v praktické části zabývat klasifikační úlohou, představíme si potom vhodnější chybové funkce používané pro klasifikační problémy. Střední kvadratická chyba nám na příkladě pomůže ukázat důvod pro jejich zvolení.

Označme si $C_{X_i} = \|y_i - a_i\|^2$. Poté dostaneme zápis ve tvaru:

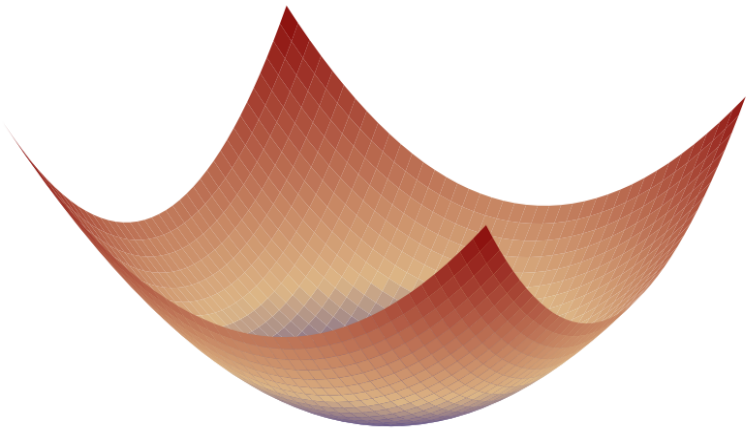
$$C = \frac{1}{N} \sum_{i=1}^{N} C_{X_i}. \tag{2.2}$$

Stejným zápisem lze zapsat všechny použité chybové funkce představené v kapitole č. 3. Toto označení je výhodné i proto, že při trénování sítě dochází k postupnému výpočtu C_{X_i} pro každou dvojici (X_i, y_i) a všechny představené chybové funkce pak počítají jejich aritmetický průměr. Proč je výhodné počítat jejich aritmetický průměr a ne jenom provést součet, je vysvětleno v následující podkapitole č. 2.2, která se zabývá minimalizací této funkce.

2.2 Gradientní sestup

„Gradientní sestup je iterativní optimalizační algoritmus používaný pro nalezení lokálního minima diferencovatelné funkce. Pro nalezení minima provádí kroky z aktuální pozice v opačném směru gradientu nebo jeho aproximaci. (vlastní překlad z anglického jazyka)“[28]

Pro ilustraci principu tohoto algoritmu si představme, že chceme najít minimum funkce $C(v_1, v_2)$ o dvou proměnných, jejíž graf může vypadat například jako graf na obrázku č. 4.



Obrázek 4: Ilustrativní graf funkce o dvou proměnných

Pokud bychom do libovolného bodu této plochy umístili kuličku, začala by se kutálet směrem dolů do údolí. Gradientní sestup funguje na podobném principu. Náhodné umístění kuličky odpovídá náhodné inicializaci proměnných v_1, v_2 a vypočtením gradientu funkce zjistíme směr dolů do údolí. K těmto proměnným pak budeme přičítat Δv_1 a Δv_2 tak, abychom se více přiblížili minimu. Tento proces opakujeme, dokud se nedostaneme do požadovaného minima [21].

Parciální derivace funkce $C(v_1, v_2)$ podle proměnné v_1 nám řekne aktuální sklon ve směru osy proměnné v_1. Pokud bude kladná, pak hodnota $C(v_1, v_2)$ vzroste při zvětšení proměnné v_1 a klesne při zmenšení proměnné v_1. Bude-li záporná, pak hodnota funkce klesne při zvětšení proměnné v_1 a vzroste při zmenšení proměnné v_1. Bude-li tedy mít naše Δv_1 opačné znaménko oproti této derivaci, hodnota naší funkce klesne. Nastavíme-li navíc změnu váhy závislou na velikosti této derivace, bude změna větší při větším sklonu a menší, pokud budeme blízko minima.

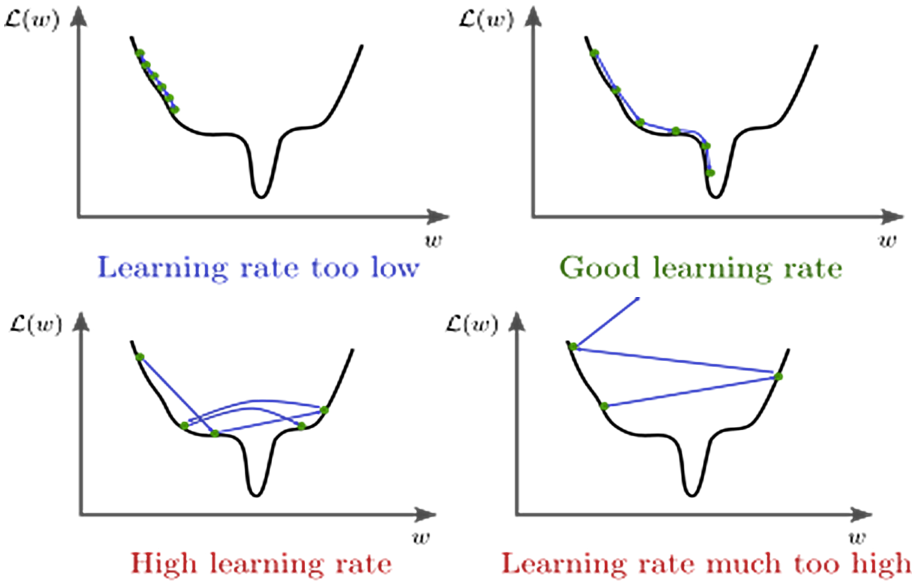
Změnu proměnné v_1 spočítáme jako:

$$\Delta v_1 = -\mu \frac{\partial C(v_1, v_2)}{\partial v_1}, \tag{2.3}$$

kde konstanta $\mu > 0$ je parametrem, který udává rychlost učení a nazývá se mírou učení. Proměnné po spočtení parciálních derivací upravíme následovně:

$$v_1' = v_1 + \Delta v_1 \tag{2.4}$$
$$v_2' = v_2 + \Delta v_2. \tag{2.5}$$

Poté celý proces opakujeme. Pokud nastavíme míru učení příliš velkou, tak se může stát, že vhodné lokální minimum po úpravě proměnných „přeskočíme“. Nastavíme-li míru učení zase příliš malou, hodnota funkce se bude blížit k lokálnímu minimu velmi pomalu.



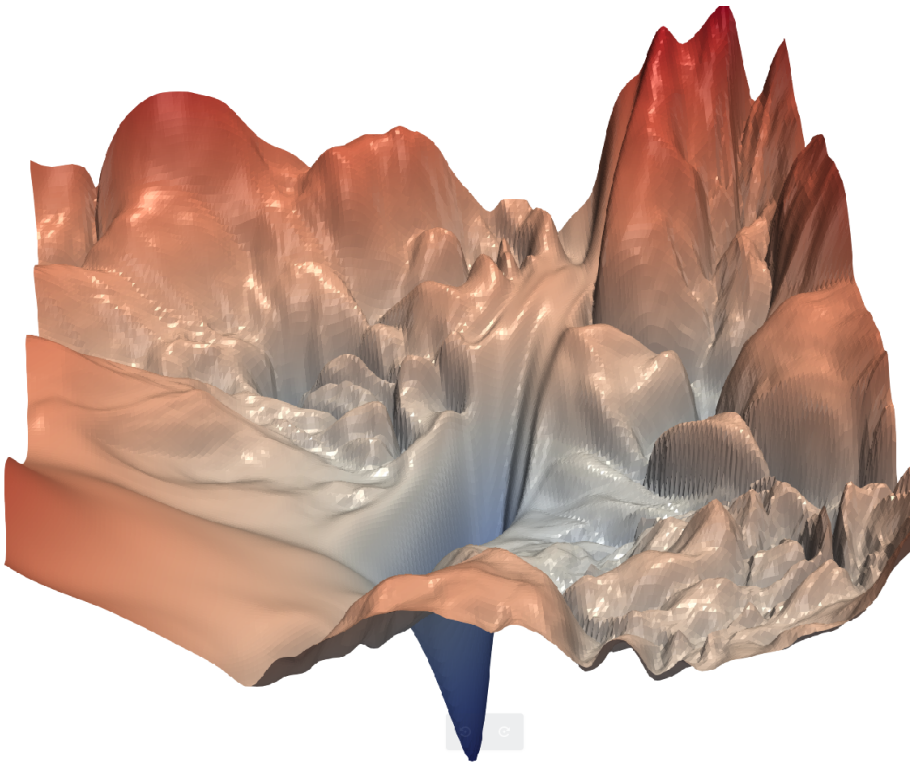
Obrázek 5: Ilustrace vlivu velikosti míry učení μ na hodnotu chybové funkce L(w). Převzato z [15]

Použití gradientního sestupu pro nalezení vhodných vah a prahových hodnot pro neuronovou síť je skoro stejné jako v ilustrativním příkladě. Jediný rozdíl je v tom, že musíme pro každou dvojici (X_i, y_i) z trénovací množiny spočítat C_{X_i} a poté vypočítat hodnotu chybové funkce C jako jejich průměr. Poté provedeme úpravu vah a prahových hodnot podle následujících rovnic:

$$w'^{\,l}_{jk} = w^l_{jk} - \mu \frac{\partial C}{\partial w^l_{jk}} = w^l_{jk} - \frac{\mu}{N} \sum_{i=1}^{N} \frac{\partial C_{X_i}}{\partial w^l_{jk}} \tag{2.6}$$
$$b'^{\,l}_{j} = b^l_j - \mu \frac{\partial C}{\partial b^l_j} = b^l_j - \frac{\mu}{N} \sum_{i=1}^{N} \frac{\partial C_{X_i}}{\partial b^l_j}. \tag{2.7}$$

Kdybychom při výpočtu nepočítali průměr hodnot C_{X_i}, byla by velikost chybové funkce a velikost gradientu větší s rostoucím počtem dvojic v trénovací množině. Museli bychom tak pro trénovací množinu s jiným počtem dvojic zmenšovat naši vhodně nalezenou míru učení μ (toto zmenšení by právě odpovídalo vydělením μ číslem N).

Protože u neuronové sítě není problém přesáhnout i 1 milion parametrů, je velmi obtížné si aspoň trochu představit, jaký graf může chybová funkce mít. Velmi krásnou vizualizaci provedli v práci [16] s názvem Visualizing the Loss Landscape of Neural Nets a jedna z jejich vizualizací grafu chybové funkce je zobrazena na obrázku č. 6.



Obrázek 6: Vizualizace grafu chybové funkce. Převzato z [16]

Gradientní sestup má několik variant. Tato, kde provádíme aktualizaci vah po výpočtu na celé trénovací množině, se nazývá dávkový gradientní sestup.

2.2.1 Stochastický gradientní sestup

U Stochastického gradientního sestupu vybereme v každé iteraci náhodně jednu dvojici z trénovací množiny a pro vstup z této dvojice vypočteme C_{X_i}. Poté provedeme úpravu vah. Liší se tak tím od dávkového gradientního sestupu, který provádí úpravu vah až po výpočtu na celé trénovací množině. Po větším množství úprav na náhodně vybraných trénovacích dvojicích bychom měli dostat aproximaci gradientu chybové funkce pro celou trénovací množinu.

Při používání tohoto algoritmu pro trénování neuronových sítí se jeho náhodnost většinou mírně omezuje. Neuronovou síť totiž většinou nebudeme trénovat tak dlouho, abychom náhodným vybíráním využili všechny dvojice z trénovací množiny. Vytvoříme tedy náhodnou permutaci trénovací množiny $(\hat{X}_1, \hat{y}_1), (\hat{X}_2, \hat{y}_2), \ \ldots\ , (\hat{X}_N, \hat{y}_N)$ a postupně pro každou dvojici $(\hat{X}_i, \hat{y}_i)$ z této permutace provedeme výpočet $C_{\hat{X}_i}$ a provedeme úpravu vah a prahových hodnot podle následujících rovnic:

$$w'^{\,l}_{\,jk} = w^l_{jk} - \mu \frac{\partial C_{\hat{X}_i}}{\partial w^l_{jk}} \tag{2.8}$$

$$b'^{\,l}_{\,j} \;=\; b^l_j \;\; - \mu \frac{\partial C_{\hat{X}_i}}{\partial b^l_j}. \tag{2.9}$$

Jakmile dokončíme úpravy vah pro všechny dvojice z této permutace trénovací množiny, vytvoříme novou permutaci a celý proces opakujeme. Toto jedno zpracování všech dvojic z celé trénovací množiny budeme nazývat jednou epochou v trénovacím procesu sítě. V případě dávkového gradientního sestupu dochází k úpravě vah pouze jednou za celou epochu. V stochastickém gradentním sestupu je počet úprav vah za jednu epochu roven počtu dvojic v trénovací množině.

2.2.2 Gradientní sestup s menšími dávkami

Gradientní sestup s menšími dávkami je kombinací stochastického gradientího sestupu a dávkového gradientního sestupu. Trénovací množina se náhodně rozdělí na podmnožiny, kde postupně na každé z nich spočítáme gradient chybové funkce a aktualizujeme váhy. Nevýhodou dávkového gradientního sestupu je pomalejší proces trénování pro početnější trénovací množiny, protože provádíme málo aktualizací po velkém množství výpočtů. Na druhou stranu má gradient chybové funkce stabilní směr. Stochastický gradientní sestup konverguje rychleji, ale zase u něho dochází k velkým výkyvům. Gradientní sestup s menšími dávkami kombinuje výhody obou a z části odstraňuje jejich nedostatky. Jeví se jako rozumnou kombinací mezi nimi a je proto z nich nejpoužívanější. Do procesu návrhu sítě nám však přibude jeden parametr a to velikost těchto podmnožin (dávek).

Trénovací dvojice v každé dávce, která bude obsahovat M dvojic, budeme značit $(\hat{X}_1, \hat{y}_1), (\hat{X}_2, \hat{y}_2),\ \ldots\ , (\hat{X}_M, \hat{y}_M)$. Při výpočtu gradientu chybové funkce pro jednu trénovací dávku dojde k přibližné aproximaci gradientu chybové funkce pro celou trénovací množinu.

$$\frac{1}{N} \sum_{i=1}^{N} \frac{\partial C_{X_i}}{\partial w^l_{jk}} \approx \frac{1}{M} \sum_{i=1}^{M} \frac{\partial C_{\hat{X}_i}}{\partial w^l_{jk}} \tag{2.10}$$

Poté upravíme váhy velmi podobně jako při dávkovém gradientím sestupu:

$$w'^{\,l}_{\,jk} = w^l_{jk} - \frac{\mu}{M} \sum_{i=1}^{M} \frac{\partial C_{\hat{X}_i}}{\partial w^l_{jk}} \tag{2.11}$$

$$b'^{\,l}_{\,j} \;=\; b^l_j \;\; - \frac{\mu}{M} \sum_{i=1}^{M} \frac{\partial C_{\hat{X}_i}}{\partial b^l_j}. \tag{2.12}$$

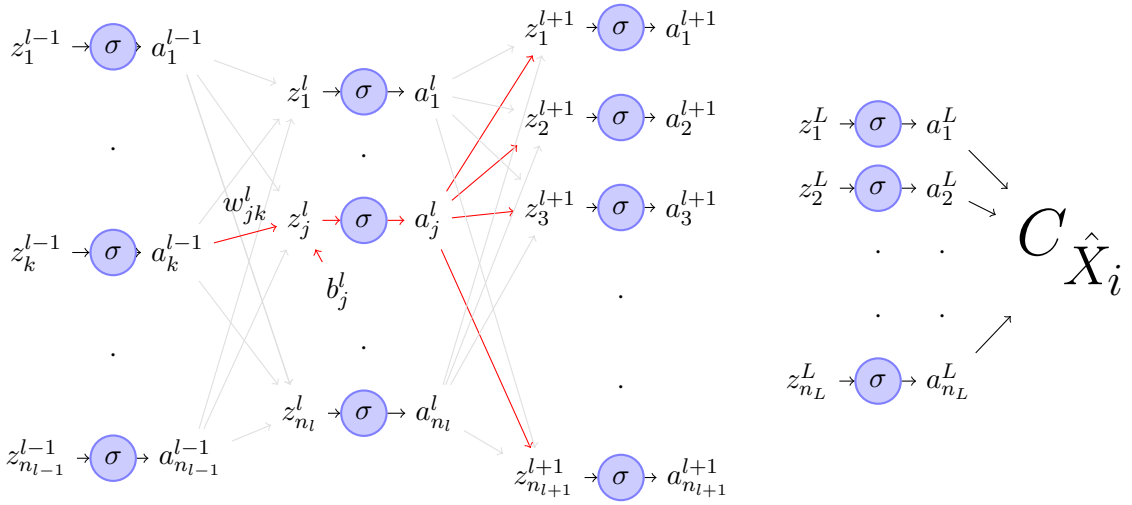
To samé provedeme pro všechny zbývající podmnožiny. Po dokončení znovu náhodně rozdělíme trénovací množinu na nové podmnožiny a celý proces opakujeme.

2.3 Algoritmus zpětného šíření

Obsah této podkapitoly vychází z literatury [21].

Pomocí algoritmu zpětného šíření efektivně spočítáme gradient chybové funkce pro každou trénovací dvojici, aniž bychom prováděli výpočty některých hodnot vícekrát. Při výpočtu se používá poměrně velké množství indexů, proměnných a využívat řetízkového pravidla pro derivace. Pro snadnější orientaci je zde zobrazeno schéma na obrázku č. 7.

Nejprve odvodíme vzorce pro výpočet parciální derivace chybové funkce podle jednotlivých vah w^l_{jk} a prahových hodnot b^l_j pro jednu trénovací dvojici $(\hat{X}_i, \hat{y}_i)$. Poté si představíme jednotlivé kroky tohoto algoritmu pro trénování neuronové sítě s použitím gradientního sestupu s menšími dávkami.



Obrázek 7: Ilustrace řetízkového pravidla

Váha w^l_{jk} a prahová hodnota b^l_j nejdříve ovlivní hodnotu chybové funkce přes z^l_j. Tento vztah můžeme napsat pomocí následujících rovnic (2.13) a (2.14):

$$\frac{\partial C_{\hat{X}_i}}{\partial w^l_{jk}} = \frac{\partial C_{\hat{X}_i}}{\partial z^l_j}\frac{\partial z^l_j}{\partial w^l_{jk}} = \frac{\partial C_{\hat{X}_i}}{\partial z^l_j}a^{l-1}_k \tag{2.13}$$

$$\frac{\partial C_{\hat{X}_i}}{\partial b^l_j} = \frac{\partial C_{\hat{X}_i}}{\partial z^l_j}\frac{\partial z^l_j}{\partial b^l_j} = \frac{\partial C_{\hat{X}_i}}{\partial z^l_j}. \tag{2.14}$$

Nyní vliv z^l_j na chybovou funkci vyjádříme pomocí vlivu a^l_j:

$$\frac{\partial C_{\hat{X}_i}}{\partial z^l_j} = \frac{\partial C_{\hat{X}_i}}{\partial a^l_j}\sigma'(z^l_j). \tag{2.15}$$

a^l_j pak ovlivňuje chybovou funkci skrze proměnné $z^{l+1}_1, z^{l+1}_2, \ldots, z^{l+1}_{n_{l+1}}$ v následující vrstvě:

$$\begin{aligned}\frac{\partial C_{\hat{X}_i}}{\partial a_j^l} &= \frac{\partial C_{\hat{X}_i}}{\partial z_1^{l+1}}\frac{\partial z_1^{l+1}}{\partial a_j^l} + \frac{\partial C_{\hat{X}_i}}{\partial z_2^{l+1}}\frac{\partial z_2^{l+1}}{\partial a_j^l} + \ \ldots\ + \frac{\partial C_{\hat{X}_i}}{\partial z_{n_{l+1}}^{l+1}}\frac{\partial z_{n_{l+1}}^{l+1}}{\partial a_j^l} = \\ &= \frac{\partial C_{\hat{X}_i}}{\partial z_1^{l+1}}\, w_1^{l+1} \ + \frac{\partial C_{\hat{X}_i}}{\partial z_2^{l+1}}\, w_2^{l+1} \ + \ \ldots\ + \frac{\partial C_{\hat{X}_i}}{\partial z_{n_{l+1}}^{l+1}}\, w_{n_{l+1}}^{l+1} \ = \\ &= \sum_{m=1}^{n_{l+1}} \frac{\partial C_{\hat{X}_i}}{\partial z_m^{l+1}} w_{mj}^{l+1}.\end{aligned} \tag{2.16}$$

Po dosazení vlivu a_j^l na chybovou funkci do rovnice (2.15) dostaneme vztah, díky kterému budeme schopni vypočítat vliv z_j^l na chybovou funkci pomocí vlivů proměnných $z_1^{l+1}, z_2^{l+1}, \ \ldots\ , z_{n_{l+1}}^{l+1}$:

$$\frac{\partial C_{\hat{X}_i}}{\partial z_j^l} = \sigma'(z_j^l) \sum_{m=1}^{n_{l+1}} \frac{\partial C_{\hat{X}_i}}{\partial z_m^{l+1}}\, w_{mj}^{l+1}. \tag{2.17}$$

Po spočítání vlivů z_p^L ve výstupní vrstvě na chybovou funkci, budeme moci zpětně dopočítat vliv z_j^l z rovnice (2.17):

$$\frac{\partial C_{\hat{X}_i}}{\partial z_p^L} = \frac{\partial C_{\hat{X}_i}}{\partial a_p^L}\sigma'(z_p^L). \tag{2.18}$$

Pro výpočet $\dfrac{\partial C_{\hat{X}_i}}{\partial a_p^L}$ již nebudeme potřebovat řetízkové pravidlo a můžeme ho snadno spočítat. Nyní máme potřebné rovnice a můžeme si představit celý algoritmus zpětného šíření.

Pro každou trénovací dvojici z dávky se provede následujících pět kroků:

1. Nastavení vstupní vrstvy pro vstup $\hat{X}_i$ z trénovací dvojice.
2. Postupný výpočet vážených vstupů z_j^l a výstupních hodnot a_j^l pro všechna $j \in \{1, 2, \ldots, n_l\}$ ve všech vrstvách $l \in \{2, \ldots, L\}$:

$$z_j^l = \sum_{k=1}^{n_{l-1}} w_{jk}^l a_k^{l-1} + b_j^l \tag{2.19}$$

$$a_j^l = \sigma(z_j^l). \tag{2.20}$$

3. Výpočet derivace chybové funkce podle vážených vstupů z_p^L v poslední vrstvě pro všechna $p \in \{1, 2, \ldots, n_L\}$:

$$\frac{\partial C_{\hat{X}_i}}{\partial z_p^L} = \frac{\partial C_{\hat{X}_i}}{\partial a_p^L}\sigma'(z_P^L). \tag{2.21}$$

4. Zpětný dopočet derivací chybové funkce podle vážených vstupů pro všechny zbývající vrstvy $l \in \{L, L-1, \ldots, 2\}$:

$$\frac{\partial C_{\hat{X}_i}}{\partial z_j^l} = \sigma'(z_j^l) \sum_{m=1}^{n_{l+1}} \frac{\partial C_{\hat{X}_i}}{\partial z_m^{l+1}}\, w_{mj}^{l+1}. \tag{2.22}$$

5. A jako poslední se už dopočítají derivace všech vah a prahových hodnot v síti:

$$\frac{\partial C_{\hat{X}_i}}{\partial w_{jk}^l} = \frac{\partial C_{\hat{X}_i}}{\partial z_j^l} a_k^{l-1} \tag{2.23}$$

$$\frac{\partial C_{\hat{X}_i}}{\partial b_j^l} = \frac{\partial C_{\hat{X}_i}}{\partial z_j^l}. \tag{2.24}$$

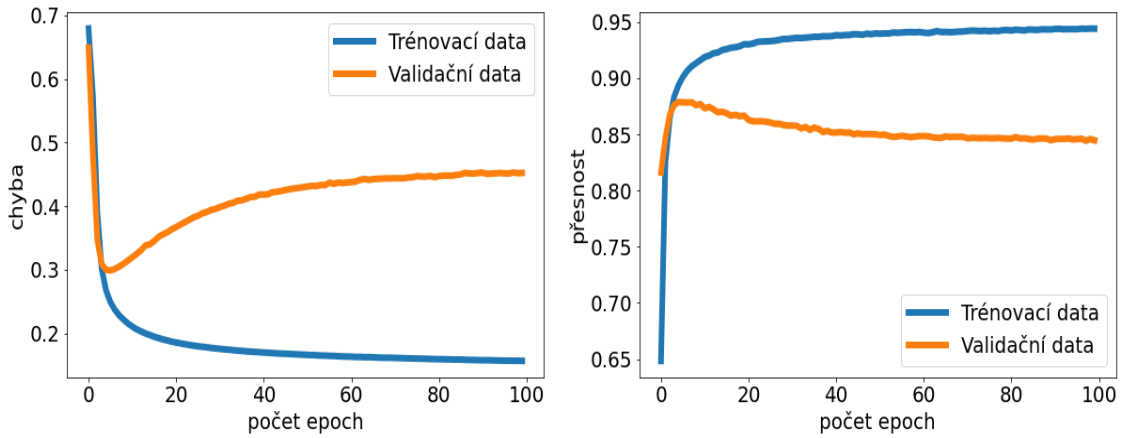
Tyto výpočty probíhají paralelně s využitím maticového počtu. Dochází tak ke zrychlení výpočtu. Po vypočtení těchto hodnot pro aktuální dávku se provádí aktualizace vah a prahových hodnot podle rovnic (2.11) a (2.12). Celý proces se poté opakuje pro další dávku.

3 Vylepšení predikce a trénování neuronové sítě

3.1 Předčasné ukončení

Počet epoch trénování sítě je důležitým hyper-parametrem, na kterém závisí výsledná kvalita předpovědi sítě. Pokud nebudeme síť trénovat dostatečně dlouho, nenaučí se podstatné rysy trénovací množiny a bude mít špatné výsledky jak na trénovací množině, tak na datech, které jsme nepoužily pro trénování. Pokud naopak budeme síť trénovat příliš dlouho, naučí se i nepodstatné detaily trénovací množiny. Neuronová síť pak bude mít výborné výsledky na trénovací množině, protože si ji doslova zapamatuje. Pro nová data už nebudou předpovědi tak moc přesné [7].

Pro určení počtu epoch stačí, aby se v průběhu trénování sledovala také chyba na datech, které se nepoužívají pro trénovaní (tzv. validační data). Pro zastavení trénování je potřeba zvolit veličinu, kterou budeme sledovat, okamžik ukončení trénování a z jaké epochy použijeme váhy pro síť (pokud si je budeme nějakým způsobem ukládat). Jako sledovaná veličina se často volí úspěšnost nebo hodnota chybové funkce pro validační data. Jako okamžik ukončení trénování můžeme zvolit např. moment, kdy se hodnota sledované veličiny přestala zlepšovat po určitý počet epoch a váhy pro model můžeme zvolit buď z konce trénování nebo i z konce epochy, pro kterou měla sledovaná veličina nejlepší hodnotu [7].



Obrázek 8: Ukázka hodnot chybové funkce a přesnosti na trénovacích a validačních datech

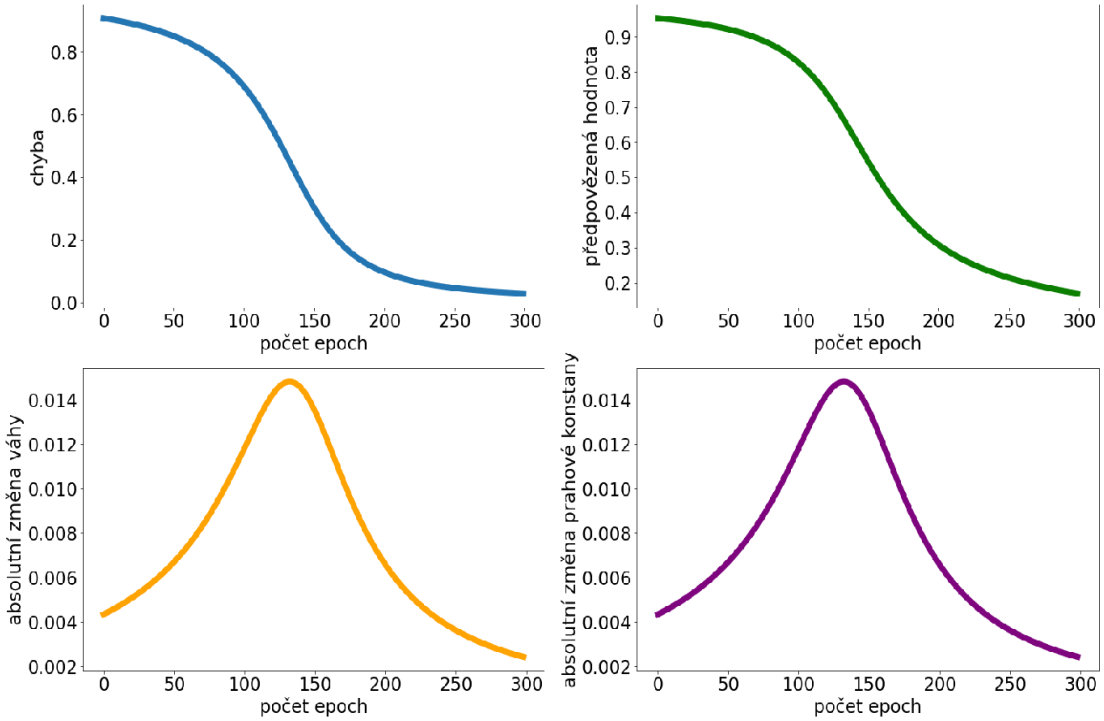
Na obrázku č. 8 je zobrazen průběh trénování, kde došlo k výraznému přetrénování sítě. V tomto případě by bylo nejlepší pro síť použít váhy nejspíš z 5. epochy. Dále bychom zde mohli přidat nějaké pravidlo pro zastavení trénování, abychom netrénovali model tak dlouho, protože asi 80 epoch bychom mohli vynechat.

3.2 Problém zpomaleného trénování

Obsah této podkapitoly vychází z literatury [21].

Při používání sigmoidy jako aktivační funkce v poslední vrstvě a střední kvadratické chyby jako chybové funkce dochází k nechtěnému efektu, který zpomaluje trénování při špatné předpovědi. Pro demonstraci tohoto problému použijeme jednoduchou neuronovou síť, která bude tvořena pouze jedním neuronem se sigmoidou jako aktivační funkcí. Naše síť bude mít jediný úkol, dostane jako vstup číslo jedna a požadovaný výstup bude nula. Aby byla předpověď neuronu na začátku trénování špatná, zvolíme jeho váhu $w = 2$ a prahovou hodnotu $b = 1$. Jejo výstup pro číslo jedna bude tedy na začátku roven přibližně 0,95. Při trénování tohoto neuronu sledujeme:

1. Hodnotu chybové funkce na konci každé epochy.
2. Predikci neuronu v každé epoše.
3. Absolutní změnu váhy mezi jednotlivými epochami.
4. Absolutní změnu prahové hodnoty mezi jednotlivými epochami.



Obrázek 9: Některé hodnoty sledované v průběhu trénování

I při velké chybě na začátku se hodnota chybové funkce snižovala velmi pomalu. Hodnota chybové funkce se začala rychleji snižovat zhruba od doby, kdy byla předpověď neuronu menší než 0.8. Zajímavé je také to, že v úseku rychlejšího trénování byly větší změny váhy a prahové hodnoty mezi jednotlivými epochami. Proč ale byly

změny váhy a prahové hodnoty mezi epochami na začátku tak malé? Velikost konstanty μ je po celý proces trénování stejná. To, že změny byly malé znamená, že v rovnicích (3.1) a (3.2) gradientního sestupu byly na začátku trénování malé hodnoty $\frac{\partial C}{\partial w}$ a $\frac{\partial C}{\partial b}$.

$$w' = w - \mu \frac{\partial C}{\partial w} \tag{3.1}$$

$$b' = b - \mu \frac{\partial C}{\partial b} \tag{3.2}$$

Abychom zjistily proč, budeme je muset spočítat. Pro naši zjednodušenou síť obsahující jeden neuron vypočteme hodnotu chybové funkce takto:

$$C = (y - a)^2 = (y - \sigma(z))^2. \tag{3.3}$$

Bude tedy platit:

$$\frac{\partial C}{\partial w} = -2(y - a)\sigma'(z)x \tag{3.4}$$

$$\frac{\partial C}{\partial b} = -2(y - a)\sigma'(z). \tag{3.5}$$

Po dosazení počátečních parametrů $w = 2$, $b = 1$ a hodnot $x = 1$, $y = 0$ dostaneme:

$$\frac{\partial C}{\partial w} = -2(0 - 0,95)\sigma'(3) = 1,9\sigma'(3) = 1,9 \cdot 0,045 = 0,0855 \tag{3.6}$$

$$\frac{\partial C}{\partial b} = -2(0 - 0,95)\sigma'(3) = 1,9\sigma'(3) = 1,9 \cdot 0,045 = 0,0855. \tag{3.7}$$

Po vynásobení konstantou $-\mu = -0.05$ dostaneme změnu rovnou číslu -0.0043, to odpovídá absolutním změnám pro první epochu na obrázku č. 9. Nyní je vidět, že naše trénování zpomaluje derivace sigmoidy. Pokud použijeme sigmoidu jako aktivační funkci v poslední vrstvě a použijeme střední kvadratickou chybu jako chybovou funkci, bude se nám tato malá hodnota promítat do úpravy všech vah a prahových hodnot v neuronové síti. Tento problém vyřešíme použitím vhodnější chybové funkce s názvem binary cross-entropy.

3.3 Binary cross-entropy

Obsah této podkapitoly vychází z literatury [21].

Binary cross-entropy je pro binární klasifikaci definovaná takto:

$$C = -\frac{1}{M} \sum_{i=1}^{M} [y_i \ln a_i + (1 - y_i) \ln(1 - a_i)], \tag{3.8}$$

kde M je počet dvojic v dávce, $y_i \in \{0; 1\}$ je správná předpověď pro vstup X_i v i-té dvojici a $a_i \in (0; 1)$ je předpověď sítě pro tento vstup.

Použijeme-li binary cross-entropy jako chybovou funkci pro případ z předchozí podkapitoly 3.2, vypočteme její hodnotu pro síť z předchozího příkladu takto:

$$C = -[y \ln(a) + (1-y)\ln(1-a)] = -y\ln(\sigma(z)) - (1-y)\ln(1-\sigma(z)). \tag{3.9}$$

Pro derivaci chybové funkce podle váhy bude platit:

$$\frac{\partial C}{\partial w} = -\frac{y}{\sigma(z)}\sigma'(z)x + \frac{1-y}{1-\sigma(z)}\sigma'(z)x = \frac{-y(1-\sigma(z)) + (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\sigma'(z)x. \tag{3.10}$$

Pro derivaci sigmoidy platí vztah:

$$\sigma'(z) = \sigma(z)(1-\sigma(z)). \tag{3.11}$$

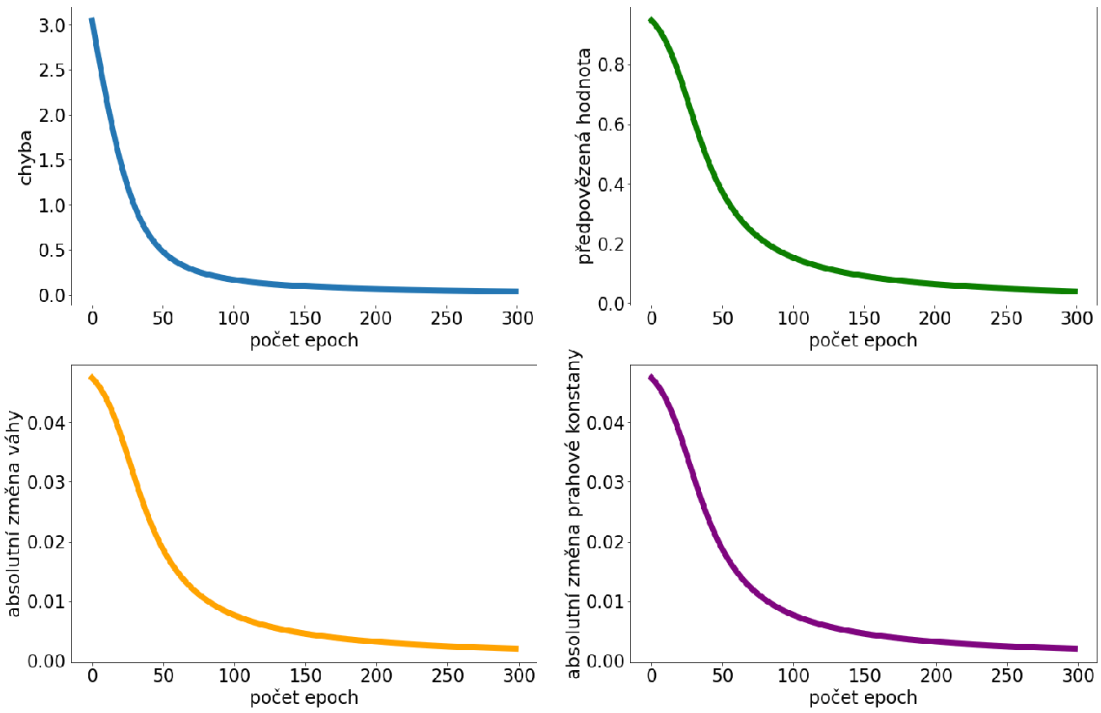
Derivace sigmoidy se zkrátí a výsledná rovnice bude:

$$\frac{\partial C}{\partial w} = -y(1-\sigma(z)) + (1-y)\sigma(z)x = (\sigma(z) - y)x. \tag{3.12}$$

Pro dosazení hodnot počátečních parametrů $w = 2$, $b = 1$, $x = 1$ a $y = 0$ dostaneme:

$$\frac{\partial C}{\partial w} = \sigma(3) = 0,95. \tag{3.13}$$

Derivace chybové funkce podle váhy je větší než při použití střední kvadratické chyby. Na obrázku č. 10 vidíme, že v průběhu trénování se hodnota chybové funkce snižuje rychleji a neuron se učí rychleji než předtím.



Obrázek 10: Některé hodnoty sledované v průběhu trénování při použití binary cross-entropy

3.4 Softmax

Funkce softmax je zobecněním logistické funkce. Používá se u klasifikačních úloh (3 a více kategorií), kde je správná pouze 1 kategorie (tzv. multi-class klasifikace). Funkce softmax transformuje vstupní vektor na vektor se stejným počtem prvků tak, aby jeho prvky byly kladné a jejich součet byl roven jedné. Jednotlivé prvky ve vektoru pak můžeme brát jako předpovězené pravděpodobnosti pro jednotlivé kategorie. Softmax funkce transformuje i-tý prvek vstupního vektoru takto:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \quad \text{pro } i \in \{1, 2, ..., n\} \text{ a } (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n. \tag{3.14}$$

Funkce softmax se používá na výstupy neuronů v poslední vrstvě. V klasifikační úloze, ve které lze předpovědět více na sobě nezávislých kategorií (tzv. multi-label klasifikace), není vhodné funkci softmax použít, protože by hodnoty předpovězených kategorií byly na sobě závislé a není u ní možné dosahovat pro dvě kategorie větší hodnoty než 0,5 najednou. U multi-label klasifikace použijeme pro aktivační funkci výstupní vrstvy sigmoidu.

3.5 Categorical cross-entropy

U funkce softmax dochází ke zpomalení trénování stejně jako u sigmoidy. Z toho důvodu se při použití funkce softmax používá jako chybová funkce tzv. categorical cross-entropy (nazývaná také jako log-likelihood), která nám podobně jako binary cross-entropy vyřeší problém s derivací ve výstupní vrstvě [21]. Její předpis je následující:

$$C = -\frac{1}{M} \sum_{i=1}^{M} \sum_{j=1}^{n} y_{ij} \ln(a_{ij}), \tag{3.15}$$

kde n značí počet prvků ve výstupním vektoru a M počet prvků v dávce.

Funkce softmax a categorical cross-entropy se nehodí pro multi-label klasifikaci, ale používají se pro multi-class klasifikaci. Pokud trénujeme pouze pro jednu správnou kategorii (multi-class klasifikace), bude ve vektoru $\vec{y_i}$ jeden prvek roven jedné (správná kategorie) a ostatní budou rovny nule. Označme si tento jediný nenulový prvek $y_{ik} = 1$ a předpověď klasifikátoru pro tuto kategorii a_{ik}, kde $i \in \{1, 2, \ldots, M\}$ a $k \in \{1, 2, \ldots, n\}$. Potom bude pro součet v (3.15) platit:

$$\sum_{j=1}^{n} y_{ij} \ln(a_{ij}) = y_{ik} \ln(a_{ik}) = \ln(a_{ik}). \tag{3.16}$$

Hodnotu této chybové funkce ovlivňují pouze předpovědi klasifikátoru pro správnou kategorii. Pokud upravíme váhy tak, aby se předpověď pro tuto kategorii zlepšila, tj. aby se hodnota a_{ik} přiblížila jedné, předpovědi pro ostatní kategorie se automaticky díky funkci softmax přiblíží více k nule a to je přesně to co v multi-class klasifikaci potřebujeme.

V multi-label klasifikaci při použití sigmoidy by nám, v případě požadované předpovědi $y_{ij} = 0$ pro nějakou kategorii, neumožňovala snižovat předpovědi a_{ij}, protože

by hodnota $y_{ij} \ln(a_{ij})$ v rovnici (3.15) byla rovna nule a nezvětšovala by hodnotu chybové funkce. V případě multi-label klasifikace použijeme upravenou binary cross-entropy pro více kategorií:

$$C = -\frac{1}{M} \sum_{i=1}^{M} \sum_{j=1}^{n} [y_{ij} \ln a_{ij} + (1 - y_{ij}) \ln(1 - a_{ij})]. \tag{3.17}$$

Takto upravená binary cross-entropy opět při použití sigmoidy vyřeší problém zpomaleného trénování, zachová nám nezávislost předpovědí pro jednotlivé kategorie a bude se zvyšovat chyba i při $y_{ij} = 0$.

3.6 RMSprop

RMSprop je algoritmus spadající do algoritmů s adaptivní mírou učení, to znamená, že se míra učení v průběhu trénování mění. Míra učení je vydělena klouzavým průměrem čtverečných gradientů. Pokud je tento klouzavý průměr vysoký, zmenšíme tak vydělením míru učení a měli bychom tak zabránit přeskočení minima. Pokud je tento průměr naopak malý, míru učení trénování nám to zvětší a urychlí tak trénování [11]. Rovnice tohoto algoritmu jsou následující:

$$g' = \rho g + (1 - \rho)(\frac{\partial C}{\partial w})^2 \tag{3.18}$$

$$w' = w - \frac{\mu}{\sqrt{g'}} \frac{\partial C}{\partial w}. \tag{3.19}$$

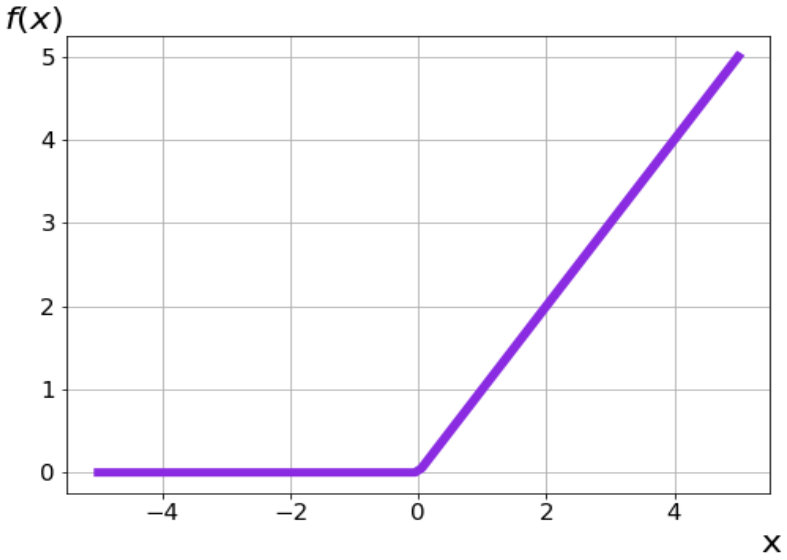
Pro tento algoritmus volíme míru učení μ a parametr ρ, který ovlivňuje klouzavý průměr. Parametr ρ má smysl volit z intervalu $\langle 0, 1)$.

3.7 ReLU

Rectified linear unit (ReLU) je v současnosti nejvíce používanou aktivační funkcí v hlubokých neuronových sítích. Je definovaná předpisem:

$$f(x) = \begin{cases} x & \text{pro } x \geq 0 \\ 0 & \text{pro } x < 0. \end{cases} \tag{3.20}$$

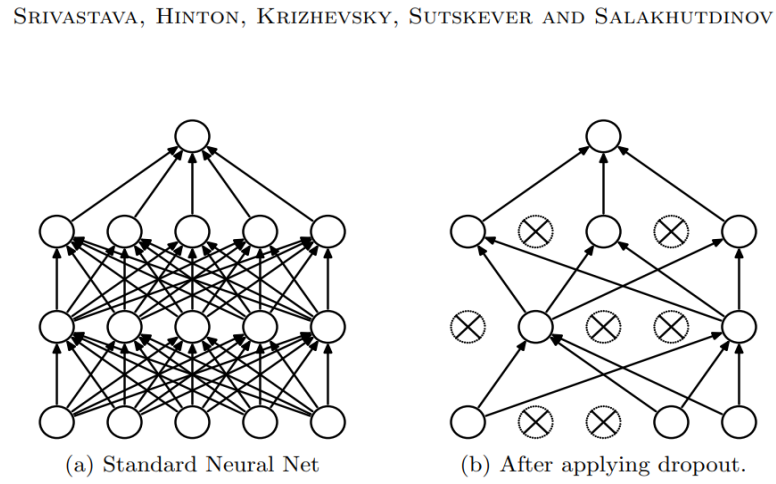
Funkce ReLU netrpí mizejícími gradienty díky své derivaci. Naproti tomu nulová derivace pro $x < 0$ způsobuje problém tzv. mrtvých neuronů. Může se stát, že pro některé neurony bude vážený vstup vždy záporný a díky nulové derivaci se váhy nezmění. Výstup takového neuronu pak bude vždy roven nule [25].



Obrázek 11: Funkce ReLU

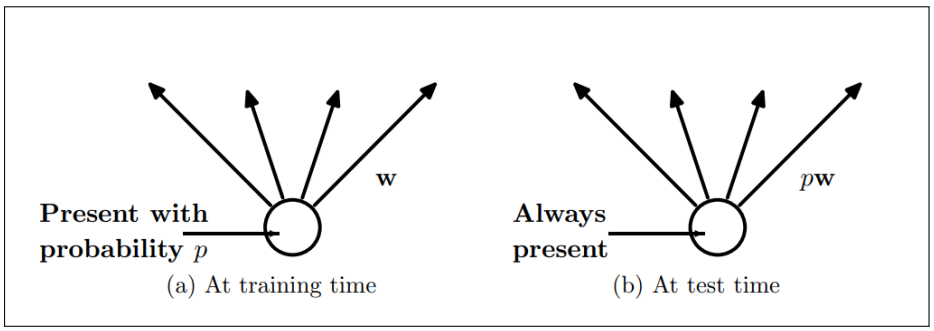
3.8 Dropout

Dropout je technika, která pomáhá snížit přeučení neuronové sítě. Při použití této techniky se při trénování sítě náhodně vyberou neurony, které se v dané trénovací iteraci vynechají. Neurony v síti by se pak díky této technice měly naučit více využívat výstupy ze všech neuronů z předchozí vrstvy a nebýt závislé pouze na pár neuronech. Navíc díky technice dropout vznikají při trénování různé zmenšené modely, které sdílejí své váhy. Kombinace různých modelů pak velmi často zlepšuje predikce. Po ukončení trénování se již dropout nepoužije a dochází tak ke kombinaci předpovědí více modelů aniž by se zvětšil počet parametrů [26].



Obrázek 12: Ukázka sítě s technikou dropout a bez. Převzato z [26]

V nejjednodušším případě je každý neuron vynechán s pravděpodobností p. Po dokončení trénování se všechny váhy k tomuto neuronu vynásobí touto pravděpodobností. Tím se zajistí, že aktuální výstup neuronu při testování bude stejný jako očekávaný výstup neuronu v průběhu trénování [26].



Obrázek 13: Výstup neuronu v průběhu trénování a testu. Převzato z [26]

3.9 Optimalizace hyper-parametrů

Proces hledání optimálních hyper-parametrů pro model a trénovací algoritmus je obvykle časově náročnější proces. Abychom zjistili vhodné hodnoty hyper-parametrů (například počet neuronů nebo rychlost učení), musíme celý model natrénovat a porovnat ho s modelem s jinak nastavenými hyper-parametry. To může být zvlášť u neuronových sítí dost zdlouhavý proces díky výpočetní náročnosti a velkému množství hyper-parametrů.

Tradičně nejefektivnějším a nejrychlejším hledačem hyper-parametrů jsou lidé. Lidé většinou dokáží z malého množství pokusů nastavit hyper-parametry tak, aby model dosahoval docela dobrých výsledků [4]. Množství kombinací je však většinou (zvlášť u neuronových sítí) obrovské a vliv jednoho hyper-parametru na úspěšnost modelu se může hodně lišit v závislosti na kombinaci zbývajících (například počet skrytých vrstev v závislosti na zvolené aktivační funkci a rychlosti učení) a záleží také na složitosti úlohy, velikosti a kvalitě trénovací množiny.

V současné době jsou počítače dost výkonné na to, aby se dalo provést více pokusů a vylepšit tak odhad hyper-parametrů. V praxi člověk většinou sám experimentuje s různými hyper-parametry a potom stanoví rozsah hyper-parametrů, ve kterém pak algoritmus hledá nejlepší kombinaci. Nejjednodušími hledacími algoritmy jsou mřížkové a náhodné vyhledávání.

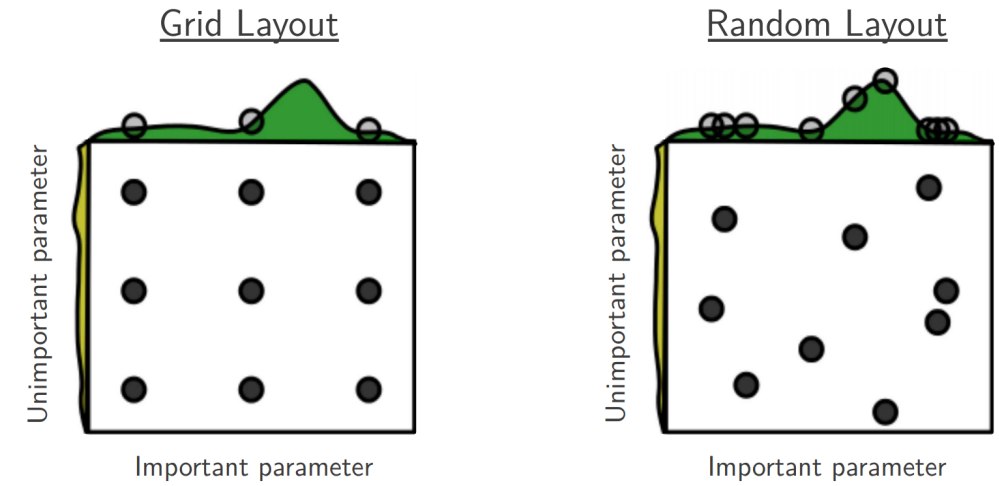
3.9.1 Mřížkové vyhledávání

Mřížkové vyhledávání je jedna z jednodušších metod pro automatické hledání hyper-parametrů. Pro každý hyper-parametr se vytvoří konečná množina, která obsahuje jednotlivé možnosti volby. Potom se postupně vyzkouší všechny možné kombinace hyper-parametrů vytvořené z těchto množin. Nevýhodou mřížkového vyhledávání je, že počet všech možných kombinací může být obrovský a hledání tak může trvat velmi dlouho [18].

Rozestupy mezi hodnotami ve zvolené konečné množině při mřížkovém vyhledávání mohou být také moc velké a optimální hodnota se může nacházet někde mezi dvěmi zvolenými hodnotami [5].

3.9.2 Náhodné vyhledávání

U náhodného vyhledávání se kombinace hyper-parametrů generují náhodně z určitého rozdělení. Výhodou náhodného vyhledávání je, že si můžeme zvolit počet iterací a po shlédnutí výsledků můžeme upravit rozdělení pro jednotlivé hyper-parametry [18]. Ukázalo se, že náhodné vyhledávání je efektivnější než mřížkové hledání [5].



Obrázek 14: Rozdíl mezi mřížkovým a náhodným hledáním. Převzato z [5]

3.9.3 Křížová validace

Křížová validace slouží k lepšímu posouzení kvality modelu. Při použití křížové validace rozdělíme trénovací data na určitý počet podmnožin. Jedna podmnožina slouží jako validační a zbylé jako trénovací. Model se natrénuje na trénovacích podmnožinách a otestuje na validační. Tento proces se opakuje tak dlouho, dokud nebude každá podmnožina použita jako validační. Dostaneme tak výsledky pro různé validační množiny a budeme mít lepší přehled o úspěšnosti modelu a jeho hyper-parametrech. Při rozdělování je dobré zachovat v podmnožinách poměr mezi jednotlivými kategoriemi. Existuje několik typů křížové validace. Tento typ se nazývá k-fold validace [8].

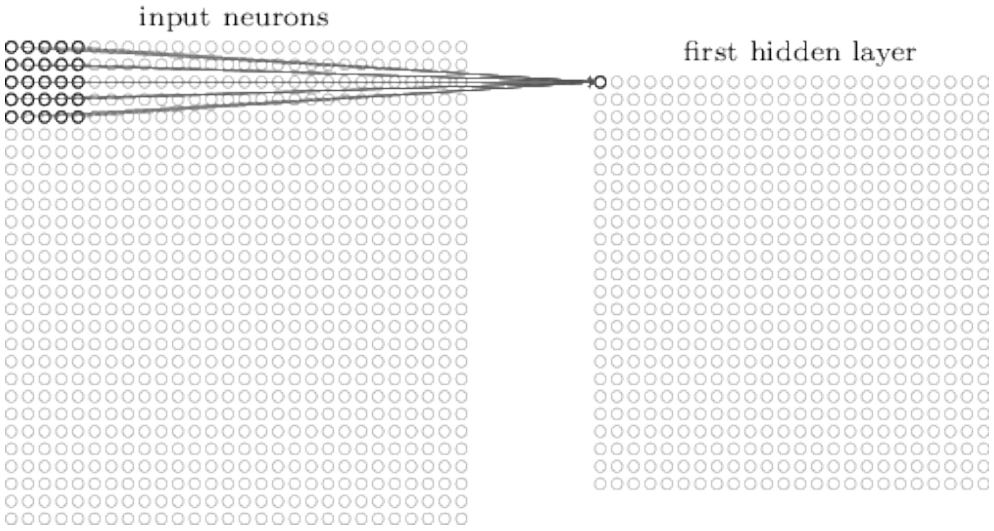
Křížová validace se může použít v kombinaci s automatickým hledáním hyper-parametrů. Časová náročnost se zvětší, budeme mít ale větší jistotu, že nalezené parametry jsou optimální.

4 Konvoluční neuronové sítě

Konvoluční neuronové sítě jsou speciálním typem neuronové sítě používaným především pro klasifikaci obrázků. Jejich výhodou oproti plně propojené neuronové síti je schopnost detekovat určité rysy na obrázku nezávisle na jejich pozici. Další výhodou konvoluční neuronové sítě je také menší počet parametrů než u plně propojené neuronové sítě [21][24].

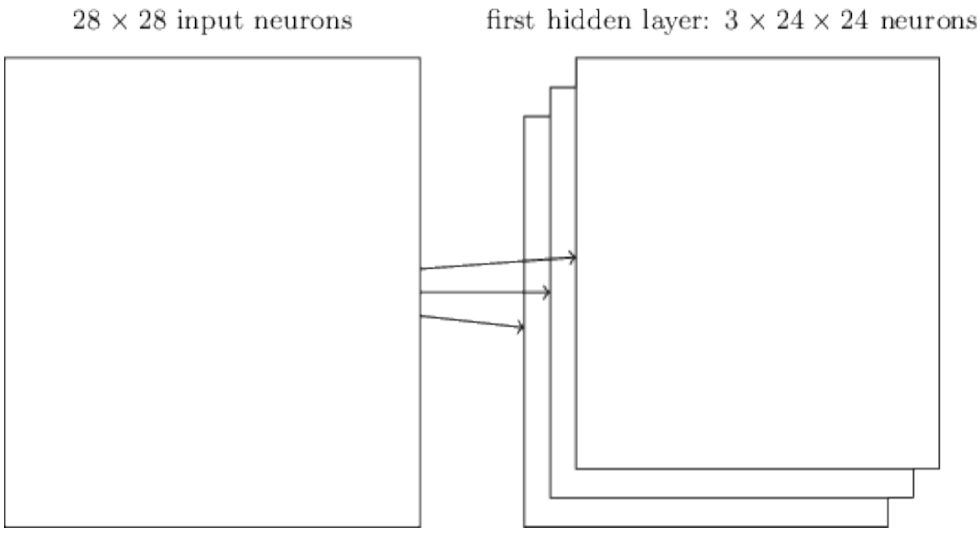
4.1 Konvoluční vrstva

Základní vrstvou konvoluční neuronové sítě je tzv. konvoluční vrstva. V konvoluční vrstvě jsou neurony propojeny pouze s určitou podoblastí vstupu do vrstvy. Tuto podoblast můžeme chápat jako určité „okno“ (např. o rozměru 5×5). Rozměr tohoto „okna“ je zpravidla stejný pro všechny neurony v konvoluční vrstvě. Neurony jsou uspořádány do tzv. příznakových map, kde v jedné příznakové mapě neurony sdílejí své váhy a prahovou hodnotu. Díky tomuto sdílení dochází v příznakové mapě k detekci stejného rysu nebo vzoru nezávisle na jeho pozici na obrázku. Tyto váhy spolu s prahovou hodnotou si můžeme představit jako jakýsi filtr, který se posouvá po obrázku a detekuje nějakou hranu nebo barvu. Rozměr a počet příznakových map (tj. počet neuronů a jejich uspořádání) závisí na rozměru obrázku(vstupu do konvoluční vrstvy), počtu a rozměru filtrů a způsobu jejich posouvání [21].



Obrázek 15: Schéma konvoluční vrstvy tvořící jednu příznakovou mapu. Převzato z [21]

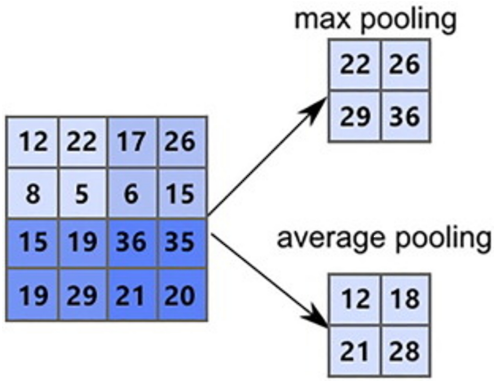
Protože jedna příznaková mapa detekuje pouze jeden rys nebo vzor, volíme v konvoluční vrstvě více příznakových map. Na obrázku č. 16 je zobrazeno schéma konvoluční vrstvy se třemi příznakovými mapami.



Obrázek 16: Schéma konvoluční vrstvy se třemi příznakovými mapami. Převzato z [21]

4.2 Pooling vrstva

Po konvolučních vrstvě se běžně používá tzv. pooling vrstva, která slouží k redukci výstupu předchozí vrstvy. Výstup předchozí vrstvy se rozdělí na určité podoblasti a každá se zredukuje stejnou metodou na jednu hodnotu. Tato metoda zůstává v průběhu učení stejná. Jednou z možných metod je redukce každé podoblasti na její maximální hodnotu. Tato metoda se nazývá tzv. maximální pooling. Další metodou je redukce na aritmetický průměr podoblasti. Tato metoda se nazývá tzv. průměrný pooling.



Obrázek 17: Maximální pooling a průměrný pooling. Převzato z [2]

Pokud pooling vrstvě předchází vrstva konvoluční, rozdělí se každá příznaková mapa v konvoluční vrstvě na podoblasti o předem stanovené velikosti (např. 2×2) a z každé podoblasti je brána maximální hodnota nebo její aritmetický průměr. Počet příznakových map zůstane zachován a zredukuje se jejich rozměr. Tím dojde ke zmenšení počtu parametrů sítě a zmenší se výpočetní náročnost sítě (v případě velikosti regionu 2×2 dojde ke zmenšení počtu prvků o 75 %). Speciálním případem je tzv. globální pooling, který zredukuje celou příznakovou mapu na jedinou hodnotu. Nejčastěji se používá globální průměrný pooling nebo globální maximální pooling.

4.3 Plně propojená vrstva

Poslední vrstvy v konvoluční neuronové síti bývají tvořeny plně propojenou vrstvou. První plně propojená vrstva je propojena se všemi neurony z předchozí konvoluční nebo pooling vrstvy. Další případné plně propojené vrstvy jsou napojeny stejně jako v plně propojené neuronové síti. Pro klasifikační úlohy slouží poslední plně propojená vrstva jako vrstva výstupní.

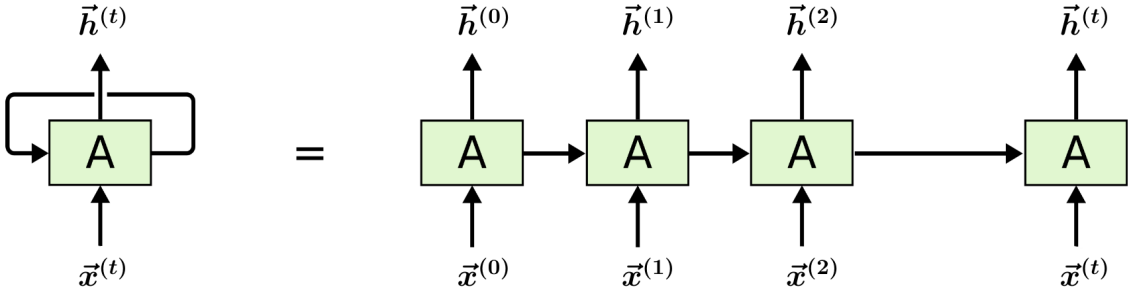
5 Rekurentní neuronové sítě

Plně propojené a konvoluční neuronové sítě patří mezi tzv. dopředné sítě. U dopředných neuronových sítí se signál šíří pouze jedním směrem a výstup neuronů v těchto sítí závisí pouze na aktuálním vstupu do sítě. Pokud na sebe jednotlivé vstupy do sítě navazují (např. časové řady), přidá se do sítě jakýsi vnitřní stav, který je určen na základě předchozích vstupů a ovlivňuje výstup sítě zároveň se vstupem aktuálním. Takovéto sítě se nazývají rekurentní [21].

5.1 Jednoduchá rekurentní vrstva

Obsah této podkapitoly vychází z literatury [22].

V jednoduché rekurentní vrstvě je tímto vnitřním stavem její předchozí výstup. Každý neuron v této vrstvě má váhy zvlášť pro tento vnitřní stav a pro nový vstup.



Obrázek 18: Schéma rekurentní vrstvy. Převzato a upraveno z [22]

Mějme jednoduchou rekurentní vrstvu, která obsahuje n neuronů. Jejím aktuálním vstupem je vektor $\vec{x}^{(t)} \in \mathbb{R}^m$ a jejím předchozím výstupem vector $\vec{h}^{(t-1)} \in \mathbb{R}^n$. Váhu j-tého neuronu ke k-té složce vektoru $\vec{x}^{(t)}$ označme w_{jk}. Váhu j-tého neuronu k l-té složce předchozího výstupu $\vec{h}^{(t-1)}$ označme u_{jl}. Prahovou hodnotu j-tého neuronu označme b_j. V rekurentní vrstvě se zpravidla používá jako aktivační funkce hyperbolický tangens. Pro výstup $\vec{h}^{(t)}$ platí:

$$h_j^{(t)} = \tanh\left(\sum_{k=1}^{m} w_{jk} x_k^{(t)} + \sum_{l=1}^{n} u_{jl} h_l^{(t-1)} + b_j\right), \tag{5.1}$$

kde $j \in \{1, 2, \ldots, n\}$. U výstupu $\vec{h}^{(0)}$ předpokládáme, že skrytý stav je nulový vektor a výstup se vypočítá stejně jako u vrstvy v plně propojené neuronové síti:

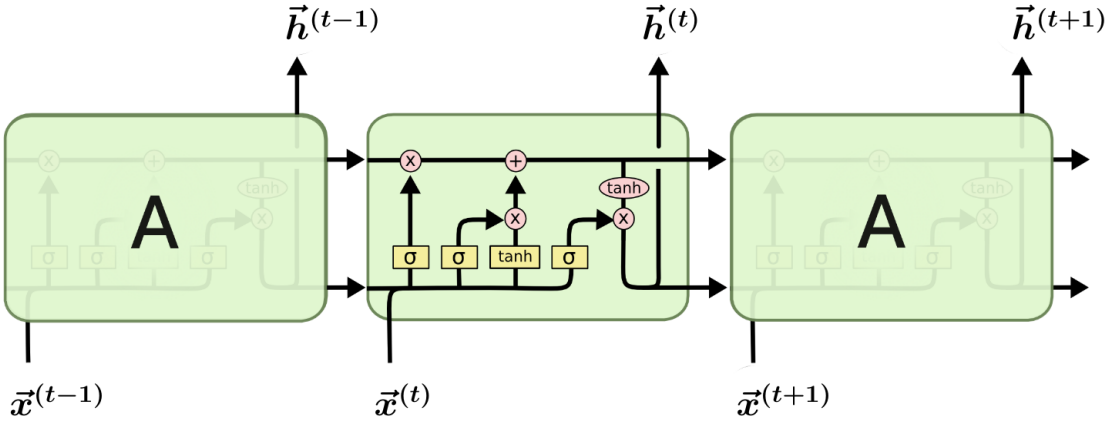
$$h_j^{(0)} = \tanh\left(\sum_{k=1}^{m} w_{jk} x_k^{(0)} + b_j\right). \tag{5.2}$$

5.2 Long short-term memory

Obsah této podkapitoly vychází z literatury [19] a [22].

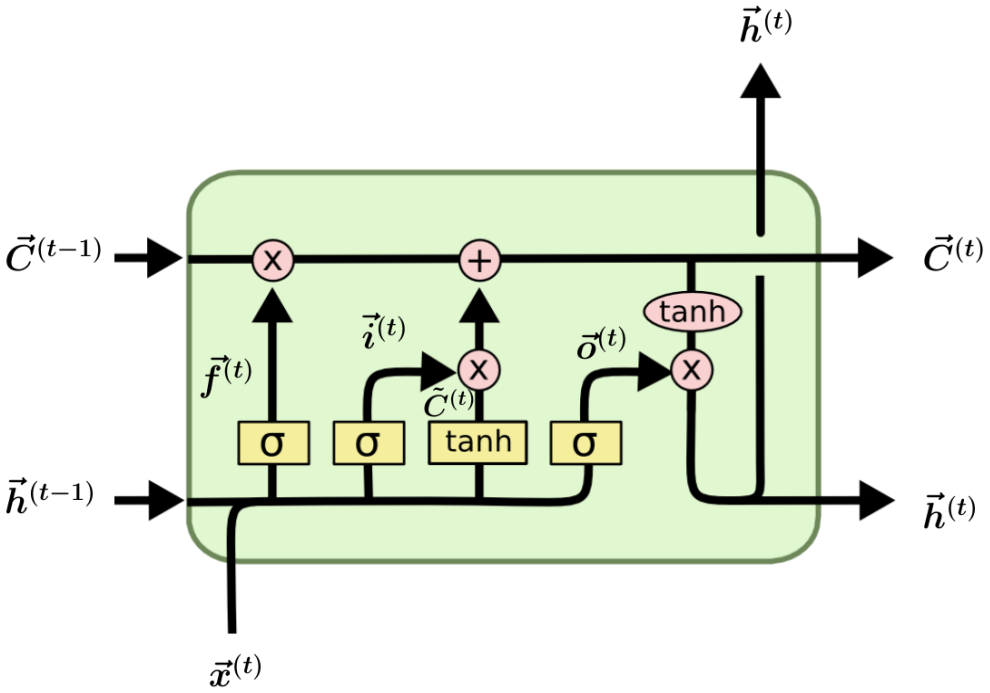
Long short-term memory (LSTM), je speciální druh rekurentní vrstvy, ve které je buňka rozšířená o tzv. „brány“. LSTM vrstva si dokáže uchovat dlouhodobější

závislosti než jednoduchá rekurentní vrstva. Díky tomu našly uplatnění v rozpoznávání řeči, predikci časových řad a také v klasifikaci textu.



Obrázek 19: Schéma LSTM vrstvy. Převzato a upraveno z [22]

Důležitou součástí LSTM buňky je stav $\vec{C}^{(t)} \in \mathbb{R}^n$. Tento stav je ovlivňován pouze lineárními operacemi a díky tomu je pro LSTM vrstvu mnohem snadnější uchovat si informaci po delší dobu. LSTM buňka má schopnost zapomenout nebo přidat informace do stavu $\vec{C}^{(t)}$ skrz brány. Schéma LSTM buňky je zobrazeno na obrázku č. 20.



Obrázek 20: Detail LSTM buňky. Převzato a upraveno z [22]

Jedna brána je složena z plně propojené vrstvy se sigmoidou jako aktivační funkcí a z operace násobení dvou vektorů po složkách. Výstupem této vrstvy je vektor, jehož složky jsou desetinná čísla mezi nulou a jedničkou. Jiný vektor procházející

skrz tuto bránu je vynásoben s výstupem této vrstvy po složkách. Je-li j-tá složka výstupu této vrstvy skoro rovna jedné, pak j-tá složka vektoru procházejícího skrze tuto bránu zůstane téměř stejná. Je-li j-tá složka výstupu této vrstvy skoro rovna nule, pak j-tá složka vektoru procházejícího skrze tuto bránu se téměř vynuluje. LSTM buňka má celkem tři brány: zapomínající, vstupní a výstupní. Počet neuronů je v každé bráně stejný. Výstup $\vec{h}^{(t)}$ a stav $\vec{C}^{(t)}$ mají právě tolik složek, kolik je neuronů v každé z těchto bran. Hovoříme o tzv. počtu jednotek.

Zapomínající brána určuje, jak velká část jednotlivých složek z předchozího stavu buňky $\vec{C}^{(t-1)}$ má být zapomenuta. Váhu j-tého neuronu plně propojené vrstvy v zapomínající bráně ke k-té složce vektoru $\vec{x}^{(t)}$ označme w^f_{jk}. Váhu k l-té složce předchozího výstupu $\vec{h}^{(t-1)}$ označme u^f_{jl} a prahovou hodnotu tohoto neuronu označme b^f_j. Výstupem plně propojené vrstvy v této bráně je vektor $\vec{f}^{(t)}$ pro jehož j-tou složku platí:

$$f^{(t)}_j = \sigma\left(\sum_{k=1}^{m} w^f_{jk} x^{(t)}_k + \sum_{l=1}^{n} u^f_{jl} h^{(t-1)}_l + b^f_j\right). \tag{5.3}$$

Vstupní brána určuje, jak velká část jednotlivých složek vektoru $\tilde{C}^{(t)}$ se přidá do stavu $\vec{C}^{(t-1)}$. Výstup vrstvy v této bráně se spočítá stejně jako v zapomínající bráně. Jiné budou pouze váhy, které označíme indexem i. Výstupem vrstvy v této bráně je vektor $\vec{i}^{(t)}$, pro jehož j-tou složku platí:

$$i^{(t)}_j = \sigma\left(\sum_{k=1}^{m} w^i_{jk} x^{(t)}_k + \sum_{l=1}^{n} u^i_{jl} h^{(t-1)}_l + b^i_j\right). \tag{5.4}$$

Vektor $\tilde{C}^{(t)}$, který prochází skrze vstupní bránu, je výstupem jiné plně propojené vrstvy, ve které mají neurony jako aktivační funkci hyperbolický tangens. Váhy v této vrstvě označíme indexem C. Pro j-tou složku tohoto vektoru platí:

$$\tilde{C}^{(t)}_j = \tanh\left(\sum_{k=1}^{m} w^C_{jk} x^{(t)}_k + \sum_{l=1}^{n} u^C_{jl} h^{(t-1)}_l + b^C_j\right). \tag{5.5}$$

Nyní můžeme vypočítat nový stav $\vec{C}^{(t)}$, pro který bude platit:

$$C^{(t)}_j = f^{(t)}_j C^{(t-1)}_j + i^{(t)}_j \tilde{C}^{(t)}_j. \tag{5.6}$$

Výstupní brána rozhoduje o výstupu LSTM buňky $\vec{h}^{(t)}$. Váhy pro tuto bránu označíme indexem o. Výstupem vrstvy v této bráně je vektor $\vec{o}^{(t)}$, pro jehož j-tou složku platí:

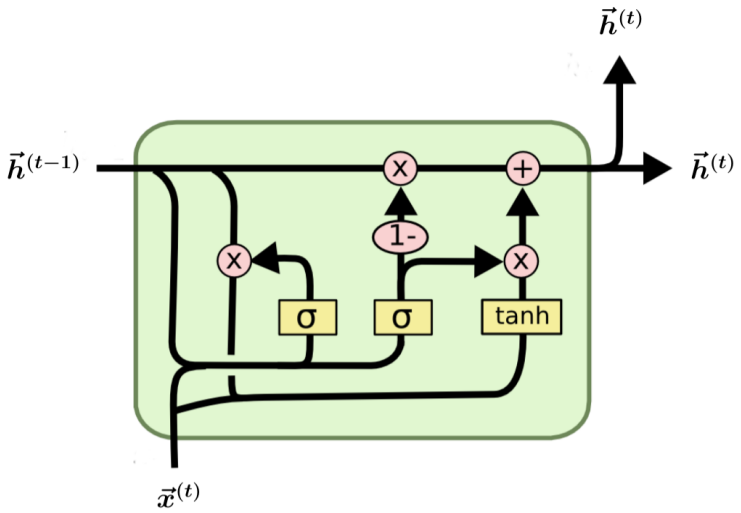
$$o^{(t)}_j = \sigma\left(\sum_{k=1}^{m} w^o_{jk} x^{(t)}_k + \sum_{l=1}^{n} u^o_{jl} h^{(t-1)}_l + b^o_j\right). \tag{5.7}$$

Nyní lze již spočítat výstup $\vec{h}^{(t)}$, pro jehož j-tou platí:

$$h^{(t)}_j = o^{(t)}_j \tanh(C^{(t)}_j). \tag{5.8}$$

5.3 Gated recurrent unit

Gated recurrent unit (GRU) využívá brány podobně jako LSTM. Narozdíl od LSTM obsahuje pouze skrytý stav $h^{(t)}$ a má jenom dvě brány. V GRU je zapomínající a vstupní brána zkombinována do jedné tzv. aktualizační brány. GRU se rychleji a snadněji trénuje než LSTM, protože má méně parametrů.



Obrázek 21: Schéma GRU buňky. Převzato a upraveno z [22]

5.4 Obousměrná vrstva

Obousměrná rekurentní vrstva se skládá například z klasické LSTM vrstvy, která nejprve projede posloupnost od začátku do konce a poté provede to samé, ale pozpátku. Výstupy obou směrů se pak předem zvoleným způsobem sloučí do jednoho výstupu. Běžně se volí prosté spojení výstupu (výstupní vektor je tak dvakrát delší).

6 Číselná reprezentace textu

6.1 TF-IDF

Pro neuronovou síť potřebujeme metodu na převod textu do číselné reprezentace. Jednou z nejpoužívanějších metod je TF-IDF (term frequency-inverse document frequency), která umožňuje převod libovolně dlouhého textového dokumentu (například jedné recenze) na vektor s pevně daným počtem složek. Každá složka tohoto vektoru odpovídá jednomu konkrétnímu slovu. Musíme tedy nejdříve vybrat slova, která nás budou v textu zajímat. Typicky se volí deset tisíc nejčastěji se vyskytujících slov v celé datové sadě nebo její části určené pro trénování. Metoda TF-IDF nám pomůže určit „důležitost“ slova v textovém dokumentu podle jeho výskytu v tomto dokumentu a celkového výskytu v datové sadě.

Složku vektoru pro textový dokument d z datové sady D označme $tfidf_{t,d}$, kde $t \in \{1, 2, \ldots, n\}$. Číslo t bude reprezentovat index konkrétního slova, které daná složka vektoru reprezentuje. Hodnotu této složky spočítáme jako součin složek $tf(t, d)$ a $idf(t, D)$:

$$tfidf_{t,d} = tf(t, d) \cdot idf(t, D). \tag{6.1}$$

6.1.1 Četnost slova v dokumentu

Složka $tf(t, d)$ nám udává hodnotu, která souvisí s četností slova s indexem t v textovém dokumentu d. Četnost nebo frekvence daného slova by totiž měla mít určitý vliv na kategorii dokumentu. Existuje několik variant metody TD-IDF. Jedna z nich spočítá tuto složku následovně:

$$tf(t, d) = \frac{n_{t,d}}{\sum_{i=1}^{n} n_{i,d}}, \tag{6.2}$$

kde $n_{t,d}$ je četnost slova s indexem t v textovém dokumentu d a $\sum_{i=1}^{n} n_{t_i,d}$ je celková četnost všech uvažovaných slov vyskytujících se v textovém dokumentu d.

6.1.2 Převrácená četnost slova

Složka $idf(t, D)$ nám pomáhá určit užitečnost slova s indexem t. Jedná se o převrácenou četnost slova ve všech dokumentech. Některá slova se budou pravděpodobně vyskytovat téměř v každém dokumentu a nebudou mít příliš velký vliv na kategorii dokumentu. Jedna z variant jejího výpočtu je následující:

$$idf(t, D) = \log \frac{|D|}{D_t}, \tag{6.3}$$

kde $|D|$ je počet textových dokumentů v datové sadě a D_t je počet textových dokumentů ve kterém se vyskytuje slovo s indexem t. Funkce log zde značí logaritmus o základu deset.

Pro převod textových dokumentů (recenzí) do TF-IDF se může použít třída Tokenizer v pythonovské knihovně Keras. V této třídě se k transformaci používá tato varianta:

$$tf(t,d) = 1 + \ln(n_{t,d}) \tag{6.4}$$

$$idf(t,D) = \ln(1 + \frac{|D|}{1 + D_t}). \tag{6.5}$$

6.2 Slovní vektory

Text může být převeden do číselné reprezentace také tak, že se jednotlivá slova převedou na konkrétní vektory.

6.2.1 One-hot encoding

Nejjednodušším způsobem převedení slov na vektory je tzv. one-hot encoding. Slova se převedou na vektory, jejichž počet složek odpovídá zvolenému počtu uvažovaných slov. Každému slovu odpovídá jedna složka vektoru podobně jako v TF-IDF metodě. Slovo se nahradí vektorem, jehož složky jsou nulové, kromě složky odpovídající danému slovu. Tuto složku položíme rovnu jedné. Tato metoda je však velmi neefektivní, protože zabírá obrovské množství paměti a navíc jsou slova v mnoharozměrném prostoru uspořádány náhodně bez ohledu na význam. One-hot encoding se používá, pokud bychom na text nepřeváděli slova, ale samotná písmena a znaky.

6.2.2 Vnoření slov

Mnohem lepší způsob převedení slov na vektory nabízí metody, které se snaží převést slova do vektorů takovým způsobem, aby slova s podobným významem ležela v mnoharozměrném prostoru blízko sebe. Takové metody se označují názvem vnoření slov. Tyto metody mohou využít one-hot encoding, aby se naučily reprezentaci slov v méně rozměrném prostoru. Nalezené vektory mají pevný a menší počet složek nezávisle na počtu uvažovaných slov.

Příkladem je například metoda Word2vec, která se pomocí plně propojené neuronové sítě s jednou skrytou vrstvou snaží předpovědět konkrétní slovo podle slov, která ho obklopují (CBOW) a nebo naopak podle konkrétního slova předpovědět ty, která ho v daném textu obklopují (skip-gram). Jako slovní vektor pro konkrétní slovo se použijí váhy jednotlivých neuronů k odpovídající složce one-hot vektoru. Nalezená vektorová reprezentace bude mít počet složek stejný, jako je počet neuronů v plně propojené skryté vrstvě. Dále se mohou použít vektorové reprezentace naučené například pomocí metody GloVe [9].

7 Analýza sentimentu recenzí

Analýzu sentimentu recenzí provedeme na datové sadě Amazon reviews polarity[1], kterou vytvořil Xiang Zhang pro použití v práci [30] jako jedno z měřítek úspěšnosti klasifikace textu. Tato datová sada obsahuje texty recenzí různých produktů z internetového obchodu Amazon.com spolu s informací, zda se jedná o pozitivní nebo negativní recenzi. Za negativní se považují recenze s hodnocením 1 a 2. Za pozitivní se považují recenze s hodnocením 4 a 5. Recenze s hodnocením 3 se v datové sadě nevyskytují. Z Každé kategorie (pozitivní nebo negativní) je v datové sadě 1 800 000 recenzí pro trénovaní a 200 000 recenzí pro vyhodnocení modelu. Dohromady tedy 3 600 000 pro trénování a 400 000 pro závěrečné vyhodnocení.

Datová sada obsahuje soubory *train.csv*, *test.csv* a *readme.txt*. Soubor *train.csv* obsahuje výše zmíněných 3 600 000 recenzí pro trénování a soubor *test.csv* 400 000 recenzí pro závěrečné vyhodnocení modelu. V souboru *readme.txt* je popis datové sady. Každý řádek souborů *train.csv* a *test.csv* obsahuje jednu recenzi a je rozdělen na tři části. Každá část je ohraničena uvozovkami a oddělena čárkou. V prvním části řádku je uvedeno číslo kategorie recenze (1 pro negativní a 2 pro pozitivní). Druhá část obsahuje nadpis recenze a třetí text recenze. Případné uvozovky v textu recenze jsou zdvojené. Na obrázku č. 22 je ukázka dvou recenzí na posledních dvou řádcích souboru *train.csv*.

```
tomas@kratochvil-tomas:~/git/amazon/amazon_review_polarity_csv$ tail -n 2 data/train.csv
"1","what is it saying?","not sure what this book is supposed to be. It is really just a
rehash of very old ideas about Carroll with some pop culture uncomfortably tacked on. The
 'myth' has been dealt with far better by people who really seem to understand it (it's t
oo deep I think for Brooker's milieu), and the pop culture is presented without any kind
of analysis or penetration.I think you are better off with Leach's 'In the Shadow of the
Dreamchild' or Sigler's 'Alternative Alices'."
"2","Makes My Blood Run Red-White-And-Blue","I agree that every American should read this
 book -- and everybody else for that matter. I don't agree that it's scholarly. Rather, i
t's a joy to read -- easy to understand even for a person with two master's degrees! Betw
een McElroy's chapter on How American Culture was Formed and Ken Burns' Lewis & Clark, I
don't know which makes my blood run red-white-and-bluer. And as a child of the anti-estab
lishment `60s, it's done a lot toward helping me understand why we Americans do what we d
o. It's the best history book I've ever read, the best history course I've ever taken or
taught. I'm buying it for my home library for my grandchildren to use as a resource. We'r
e also using it as a resource for a book on urban planning."
```

Obrázek 22: Ukázka recenzí ze souboru *train.csv*

7.1 Předzpracování textu

Nadpis recenze a text recenze byly spojeny do jednoho textu mezerou. Všechna velká písmena byla převedena na malá a z textu byly odstraněny následující znaky:

```
\'!"#$%&()*+,-./:;<=>?@[\\]^_‘{|}~\t\n.
```

Některé znaky se zapisují pomocí speciální posloupnosti znaků, která začíná tzv. únikovým znakem. Jako únikový znak se v jazyce Python používá zpětné lomítko.

[1]Tato datová sada byla stažena z: `https://drive.google.com/drive/folders/0Bz8a_Dbh9Qhbfll6bVpmNUtUcFdjYmF2SEpmZUZUcVNiMUw1TWN6RDV3a0JHT3kxLVhVR2M`

Například dvojice znaků \t zastupuje tabulátor a dvojice \\ samotné zpětné lomítko. V textu bylo nahrazeno více po sobě jdoucích mezer jednou mezerou. Dále byla odstraněna všechna čísla a slova kratší než 3 písmena. Od čísla kategorie byla odečtena jednička, aby byl požadovaný výstup sítě roven 0 nebo 1.

Při úpravě textu se často odstraňují tzv. stop slova (anglicky stop words), která se v textu často vyskytují, ale nenesou významnou informaci. Z textu byla odstraněna anglická stop slova, která se nacházela v seznamu pythonovské knihovny NLTK. Jedná se konkrétně o následující slova:

i, me, my, myself, we, our, ours, ourselves, you, you're, you've, you'll, you'd, your, yours, yourself, yourselves, he, him, his, himself, she, she's, her, hers, herself, it, it's, its, itself, they, them, their, theirs, themselves, what, which, who, whom, this, that, that'll, these, those, am, is, are, was, were, be, been, being, have, has, had, having, do, does, did, doing, a, an, the, and, but, if, or, because, as, until, while, of, at, by, for, with, about, against, between, into, through, during, before, after, above, below, to, from, up, down, in, out, on, off, over, under, again, further, then, once, here, there, when, where, why, how, all, any, both, each, few, more, most, other, some, such, no, nor, not, only, own, same, so, than, too, very, s, t, can, will, just, don, don't, should, should've, now, d, ll, m, o, re, ve, y, ain, aren, aren't, couldn, couldn't, didn, didn't, doesn, doesn't, hadn, hadn't, hasn, hasn't, haven, haven't, isn, isn't, ma, mightn, mightn't, mustn, mustn't, needn, needn't, shan, shan't, shouldn, shouldn't, wasn, wasn't, weren, weren't, won, won't, wouldn, wouldn't.

Dále byla provedena tzv. lemmatizace slov, která jednotlivá slova nahradí jejím základním tvarem. Pokud použijeme lemmatizaci například na anglické slovo *doing*, bude toto slovo nahrazeno slovem *do*. Dále jsou také slova množného čísla nahrazeny tvarem jednotného čísla. Lemmatizace tak pomáhá zajistit, aby měly různé tvary jednoho slova pro klasifikátor stejný význam. Lemmatizace slov byla provedena pomocí lemmatizátoru knihovny NLTK.

7.2 Návrh sítí a náhodné vyhledávání

Pro návrh jednotlivých sítí bylo náhodně vybráno 50 000 recenzí ze souboru *train.csv*. Tyto recenze byly vybrány tak, aby byl zachován poměr mezi jednotlivými kategoriemi celé datové sady (tj. 25 000 pozitivních a 25 000 negativních recenzí). Recenze ze souboru *test.csv* byly použity pro finální vyhodnocení nejlepších sítí. Nejprve byly prováděny experimenty. V těchto experimentech bylo použito 40 000 recenzí pro natrénování sítě a 10 000 pro validaci. Po těchto experimentech byl stanoven rozsah hodnot jednotlivých hyper-parametrů pro náhodné vyhledávání. Náhodné vyhledávání bylo zvoleno kvůli jednoduché implementaci a kvůli velkému počtu všech možných kombinací hyper-parametrů. Výsledky náhodného vyhledávání byly průběžně ukládány a v případě nečekaného ukončení programu stačilo program znovu spustit. U mřížkového vyhledávání by se muselo zjišťovat, které kombinace byly prohledány a program upravit tak, aby nedošlo k prohledávání úplně od začátku. Spolu s náhodným vyhledáváním byla také použita křížová validace (k-fold) pro objektivnější posouzení jednotlivých kombinací. Recenze byly rozděleny do pěti stejně velkých podmnožin po 10 000 recenzí. Pro každou kombinaci hyper-parametrů

tak bylo natrénováno pět sítí. Každá síť byla trénována na jiné kombinaci skládající se ze čtyř podmnožin a validována na té zbývající. Poměr mezi kategoriemi byl v každé podmnožině zachován. Po náhodném vyhledávání byly nejúspěšnější kombinace použity pro trénování na celé datové sadě. Použití náhodného vyhledávání a křížové validace bylo hlavním důvodem, proč byla pro návrh použita pouze část trénovacích recenzí. Na všech 3 600 000 recenzí by použití těchto metod bylo časově velmi náročné.

7.2.1 Plně propojená neuronová síť

Jako vstup do plně propojené neuronové sítě použijeme text recenze převedený do TF-IDF vektoru. Síť má několik skrytých vrstev a výstupní vrstva sítě obsahuje jeden neuron se sigmoidou jako aktivační funkcí. Výstupem sítě je desetinné číslo mezi nulou a jedničkou. Jako ztrátovou funkci zvolíme binary cross-entropy. Výstup sítě menší než 0,5 bereme jako předpověď pro kategorii 0 (negativní recenze) a výstup větší než 0,5 jako kategorie 1 (pozitivní recenze). Je také možné použít ve výstupní vrstvě 2 neurony a na jejich výstup použít funkci softmax spolu se ztrátovou funkcí categorical cross-entropy. Výstupem sítě by tak byl vektor o dvou složkách, kde jedna z nich by představovala pravděpodobnost pro negativní recenzi a druhá pro pozitivní.

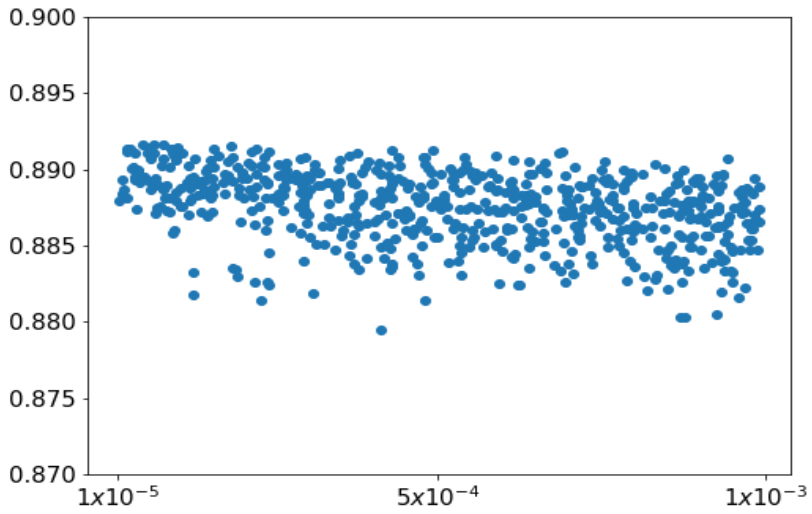
Při experimentech dosahovaly sítě nejlepšího výsledku na validační množině do 80. epochy. Proto byl pro trénování sítí v náhodném vyhledávání nastaven počet epoch na 80. Pokud se nezlepšila nejlepší dosažená úspěšnost sítě na validační množině v průběhu 10 epoch, trénování se předčasně ukončilo. Bylo také experimentováno na recenzích bez odstranění stop slov a bez použití lemmatizace. Sítě dosahovaly na textu bez provedení této úpravy větší úspěšnosti. Nebylo ale jisté, jestli by se zvolením jiných hodnot hyper-parametrů nedosáhlo opačného výsledku. Proto bylo náhodné vyhledávání provedeno na obou variantách předzpracování textu.

Pro náhodné vyhledávání byly použity následující parametry:

- Počet složek TF-IDF vektorů: 10 000
- Počet skrytých vrstev: 1–4
- Počet neuronů ve skrýtých vrstvách:
  - 1. skrytá vrstva: 40–500
  - 2. skrytá vrstva: 30–počet neuronů v 1. skryté vrstvě
  - 3. skrytá vrstva: 20–počet neuronů v 2. skryté vrstvě
  - 4. skrytá vrstva: 10–počet neuronů v 3. skryté vrstvě
- Aktivační funkce ve skrytých vrstvách: sigmoida nebo ReLU
- Počet neuronů ve výstupní vsrtvě: 1
- Aktivační funkce neuronu ve výstupní vrstvě: sigmoida
- Dropout: $(0; 0, 5)$

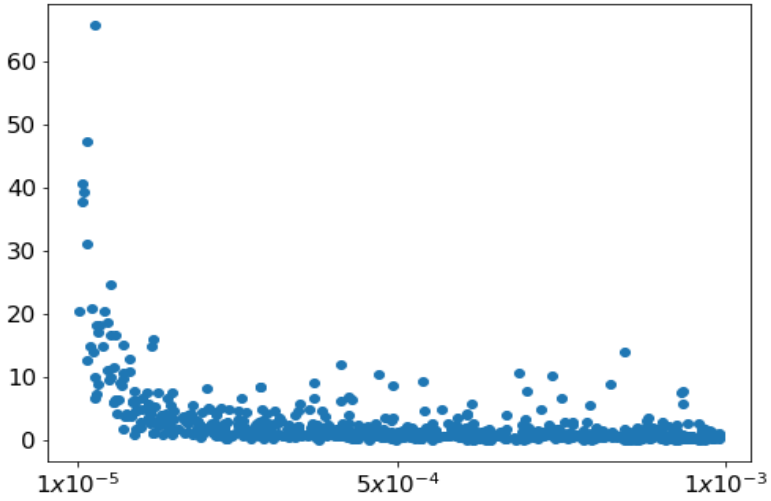
- Trénovací algoritmus: RMSprop
- Míra učení μ: $(0,00001;\ 0,001)$
- Parametr ρ: $(0,6;\ 0,95)$
- Velikost Dávky: $\{10, 20, \ldots, 500\}$
- Chybová funkce: binary cross-entropy

V průběhu náhodného vyhledávání bylo natrénováno celkem 374 sítí na recenzích, kde byly odstraněny stop slova a byla použita lemmatizace slov. Na recenzích, kde tyto metody použity nebyly, bylo natrénováno celkem 650 sítí. Sítě trénované na recenzích bez odstranění stop slov a bez použití lemmatizace dosahovaly větší úspěšnosti. Nejlepší dosažená úspěšnost na jedné z validačních podmnožin byla 89,56 %, zatímco sítě trénované na recenzích s odstraněnými stop slovy a lemmatizací dosáhly největší úspěšnosti 88,13 %. Pro srovnání vlivu některých hyper-parametrů tak budeme uvažovat úspěšnosti sítí trénovaných na textu bez odstranění stop slov a bez použití lemmatizace. Pro každou síť budeme brát průměr nejlepších dosažených úspěšností na každé z validačních podmnožin. Snižování míry učení vedlo ke stabilnějším výsledkům a mírnému zlepšení sítě. Úspěšnost sítě v závislosti na míře učení je zobrazena na obrázku č. 23.



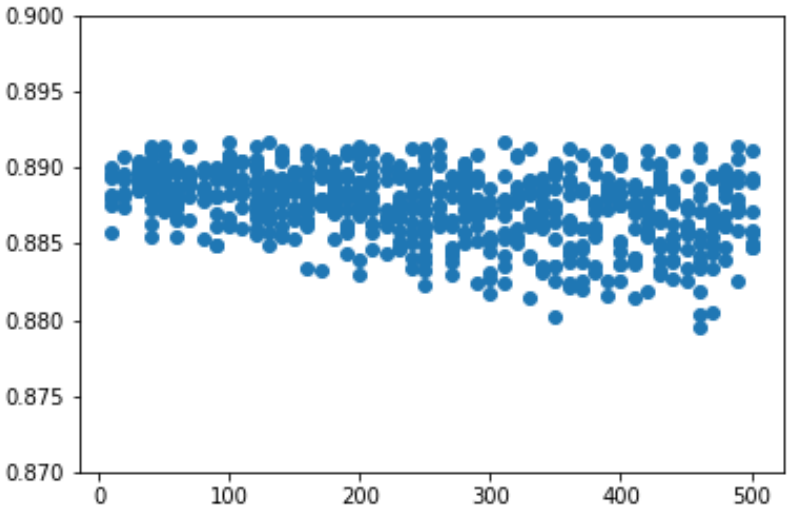
Obrázek 23: Úspěšnost sítě v závislosti na velikosti míry učení

Z obrázku č. 23 se může zdát, že další snižování míry učení by mohlo vylepšit úspěšnost sítě. Podíváme-li se na průměrný počet epoch potřebných pro dosažení nejlepší úspěšnosti zobrazený na obrázku č. 24 zjistíme, že dalším snižováním bychom velmi prodloužili dobu trénování.



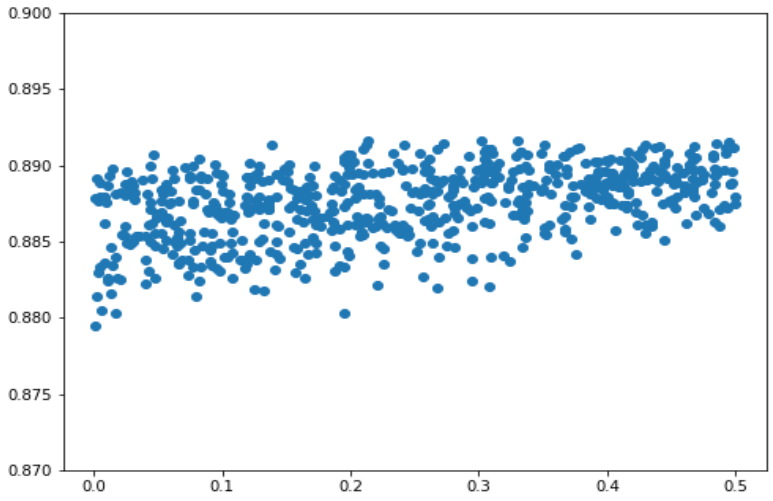
Obrázek 24: Průměrný počet epoch, ve kterých byla dosažena maximální úspěšnosti

Mírný vliv na úspěšnost sítě lze pozorovat i pro různě zvolenou velikost dávky. S rostoucí velikostí dávky byly dosahované úspěšnosti více rozptýlené a sítě dosahovaly o menších úspěšností.



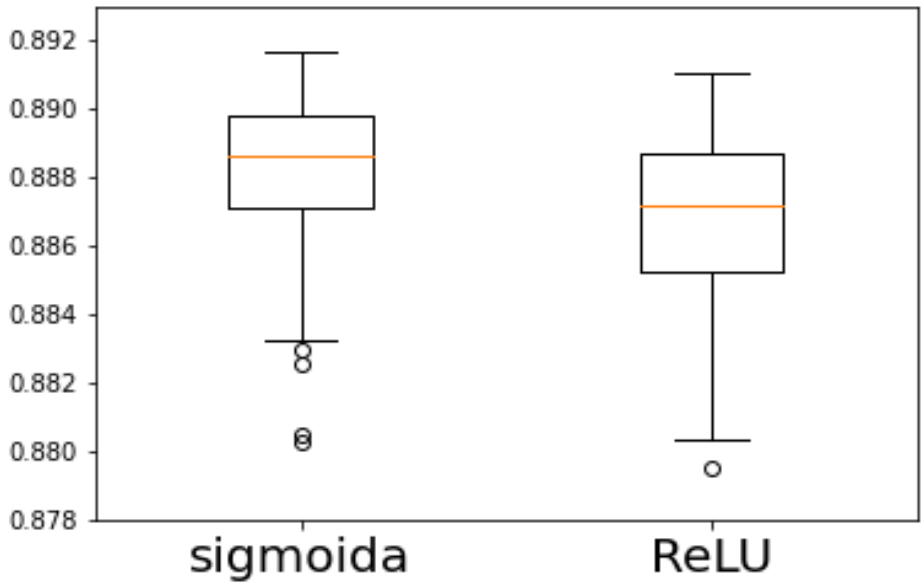
Obrázek 25: Úspěšnosti sítí v závislosti na velikosti dávky

Úspěšnost sítí v závislosti na velikosti pravděpodobnosti vynechání neuronu je zobrazena na obrázku č. 26. Se zvyšováním pravděpodobnosti docházelo ke stabilnějším výsledkům a mírnému zlepšení. Zde by se mohl zvolit větší interval pro pravděpodobnost vynechání neuronu a zjistit, do jaké hodnoty pravděpodobnosti by tento trend pokračoval a od jaké hodnoty pravděpodobnosti by se začala dosahovaná úspěšnost sítě snižovat.



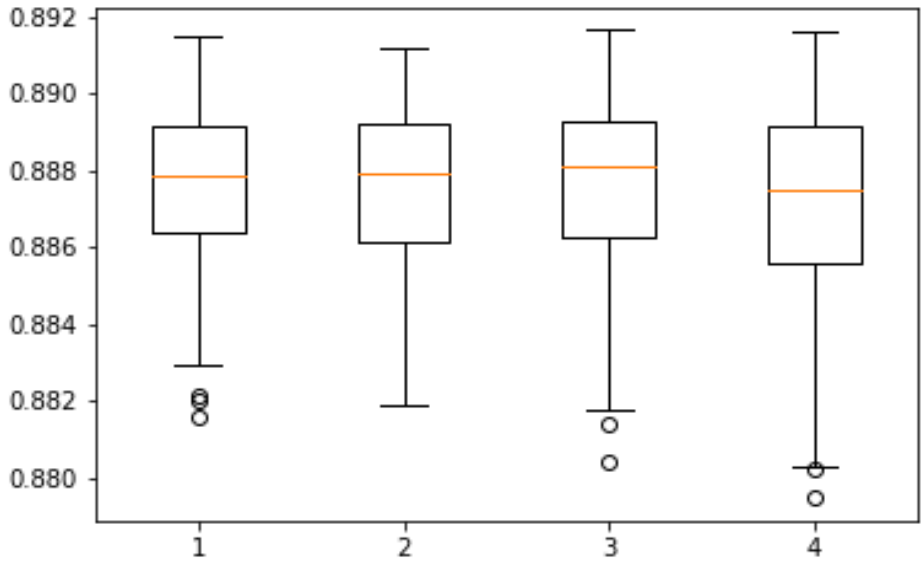
Obrázek 26: Závislost úspěšnosti sítě na pravděpodobnosti vynechání neuronu

Plně propojené sítě s aktivační funkcí sigmoidou ve skrytých vrstvách, dosahovaly celkově lepších úspěšností než sítě, kde byla použita funcke ReLu. Rozdíl mezi nimi však nebyl příliš výrazný. Jejich porovnání je zobrazeno na obrázku č. 27.



Obrázek 27: Porovnání dosažených úspěšností v závislosti na zvolené aktivační funkci

Počet skrytých vrstev v plně propojené neuronové síti neměl žádný vliv na její úspěšnost. Pokud bychom zvyšovali počet skrytých vrstev, pravděpodobně by docházelo k většímu přetrénování a menší úspěšnosti sítě. Toto tvrzení však nemusí být pravdivé a bylo by potřeba ho ověřit.



Obrázek 28: Porovnání dosažených úspěšností v závislosti na počtu skrytých vrstev

Nejlepší průměru bylo dosaženo 89,17 %. Síť s touto kombinací jako jediná přesáhla hranici 89 % na každé z validačních podmnožin. Tato síť byla vybrána pro trénování na všech trénovacích recenzích. Kombinace hyper-parametrů této sítě jsou následující:

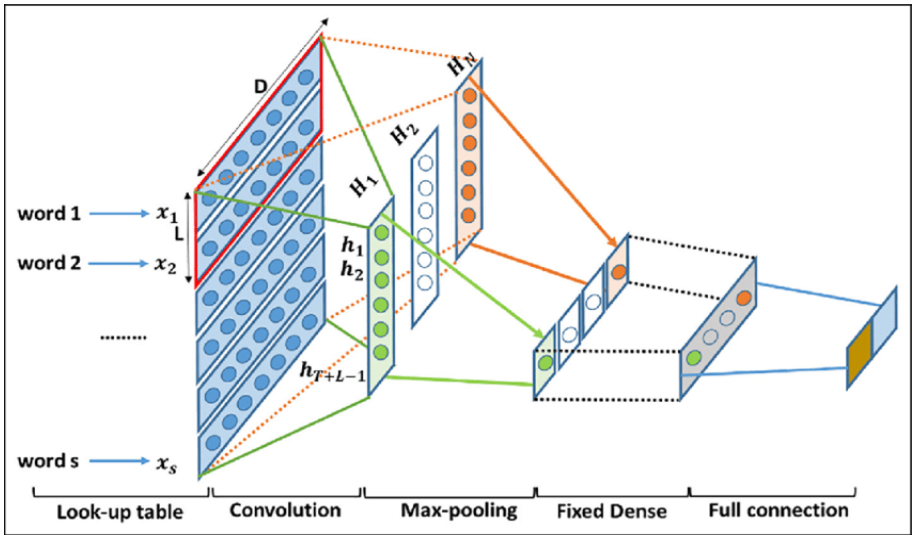
- Počet složek TF-IDF vektorů: 10 000
- Počet skrytých vrstev: 3
- Počet neuronů ve skrýtých vrstvách:
  - 1. skrytá vrstva: 139
  - 2. skrytá vrstva: 119
  - 3. skrytá vrstva: 97
- Aktivační funkce ve skrytých vrstvách: sigmoida
- Počet neuronů ve výstupní vsrtvě: 1
- Aktivační funkce neuronu ve výstupní vrstvě: sigmoida
- Dropout: $0,33$
- Trénovací algoritmus: RMSprop
- Míra učení: $0,00008$
- Parametr ρ: $0,72$
- Velikost Dávky: 310

- Chybová funkce: binary cross-entropy

Průběh trénování této sítě je zobrazen na obrázku č. 34 nacházejícího se v příloze A.

7.2.2 Konvoluční neuronová síť

Konvoluční neuronové sítě se také používají ke zpracování přirozeného jazyka, přestože nacházejí uplatnění především ve zpracování obrazu. Slova jsou nejdříve převedena na vektory. Toto převedení neprobíhá přímo před započetím trénování. Slova se nahradí pouze jednotlivými indexy vektorů. Po vstupní vrstvě následuje tzv. embedding vrstva, která tyto indexy nahradí jednotlivými vektory v průběhu trénování. Tento způsob má dvě výhody. Je snížen nárok na paměť a v případě potřeby (např. pokud nemáme k dispozici předtrénované slovní vektory) nám to umožní složky těchto vektorů brát jako hledané parametry sítě a trénovat je spolu s váhami a prahovými hodnotami. Poté následuje konvoluční vrstva. Jeden rozměr filtru v konvoluční vrstvě odpovídá počtu složek těchto vektorů. Druhý rozměr filtru odpovídá počtu slov, které bude konvoluční vrstva najednou zpracovávat. Filtr v konvoluční vrsvě se poté posouvá od začátku textu až do konce. Každá příznaková mapa tak tvoří vektor. Za konvolučními vrstvami následuje pooling vrstva. Z každé příznakové mapy je pak možné vzít maximální hodnotu nebo spočítat průměr jeho složek. Tím dostaneme pokaždé vektor, jehož počet složek bude odpovídat počtu příznakových v přecházející konvoluční vrstvě a to nezávisle na délce textu. Poté následují plně propojené vrstvy.



Obrázek 29: Příklad architektury konvoluční neuronové sítě vhodné pro klasifikaci textu. Převzato z [20]

I když tato architektura dokáže zpracovat libovolně dlouhé texty, běžně se text zkracuje nebo prodlužuje (tzv. padding) na pevnou délku. Důvod je ten, že v případech libovolně dlouhého vstupu nelze využít paralelního výpočtu, protože by měly matice různé rozměry. Trénování by v takovémto případě bylo pomalé a Keras umožňuje použít pouze velikost dávky rovnu jedné, protože vyžaduje, aby měly vstupy v jedné dávce stejné rozměry.

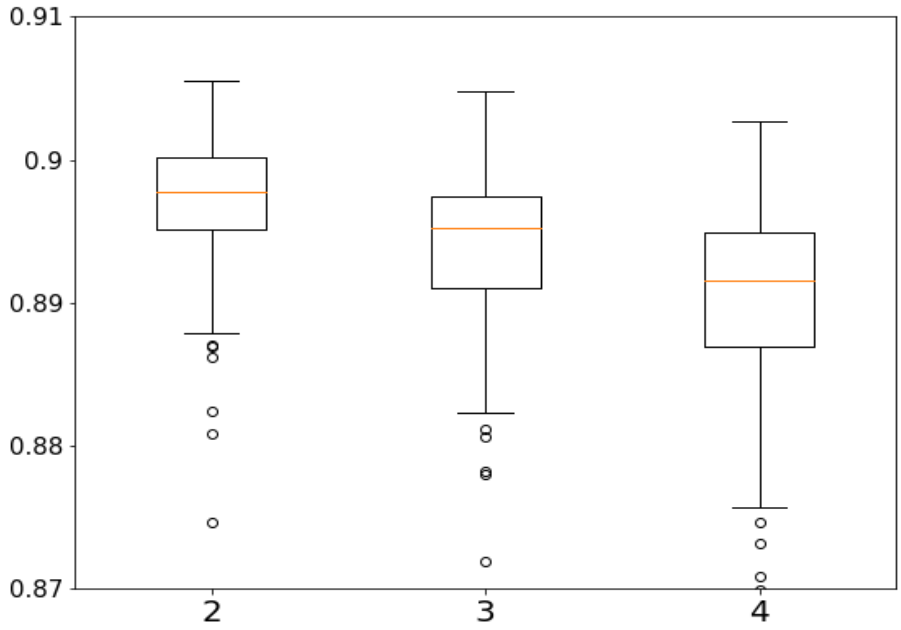
Pro převod textu byly použity předtrénované slovní vektory[2] o počtu složek 100 a 300, které byly vytvořeny pomocí nástroje Wikipedia2Vec[29]. Složky těchto vektorů zůstaly v průběhu trénování nezměněny. Pevná délka textu byla zvolena na 250 slov. Počet epoch byl nastaven na 100. V případě nezlepšení úspěšnosti sítě v průběhu 10 epoch se trénování opět předčasně ukončilo. Pro náhodné vyhledávání byly použity následující rozsahy parametrů:

- Počet uvažovaných slov: 30 000
- Počet složek slovních vektorů: 100 nebo 300
- Počet konvolučních vrstev: 1–4
- Počet neuronů v konvolučních vrstvách:
  - 1. konvoluční vrstva: 40–200
  - 2. konvoluční vrstva: 30–počet neuronů v 1. konvoluční vrstvě
  - 3. konvoluční vrstva: 20–počet neuronů v 2. konvoluční vrstvě
  - 4. konvoluční vrstva: 10–počet neuronů v 3. konvoluční vrstvě
- Velikost filtru v konvoluční vrstvě: 2, 3 nebo 4
- Aktivační funkce ve konvolučních vrstvách: ReLU
- Pooling vrstva: globální průměrný pooling
- Počet neuronů v plně propojené vrstvě: 5–100
- Aktivační funkce v plně propojené vrstvě: ReLU
- Počet neuronů ve výstupní vrstvě: 1
- Aktivační funkce neuronu ve výstupní vrstvě: sigmoida
- Dropout: $(0; 0, 3)$
- Trénovací algoritmus: RMSprop
- Míra učení: $(0, 0001;\ 0, 005)$
- Parametr ρ: $(0, 6;\ 0, 95)$
- Velikost Dávky: $\{10, 20, \ldots, 500\}$
- Chybová funkce: binary cross-entropy

[2]Tyto slovní vektory byly staženy z `https://wikipedia2vec.github.io/wikipedia2vec/pretrained/`

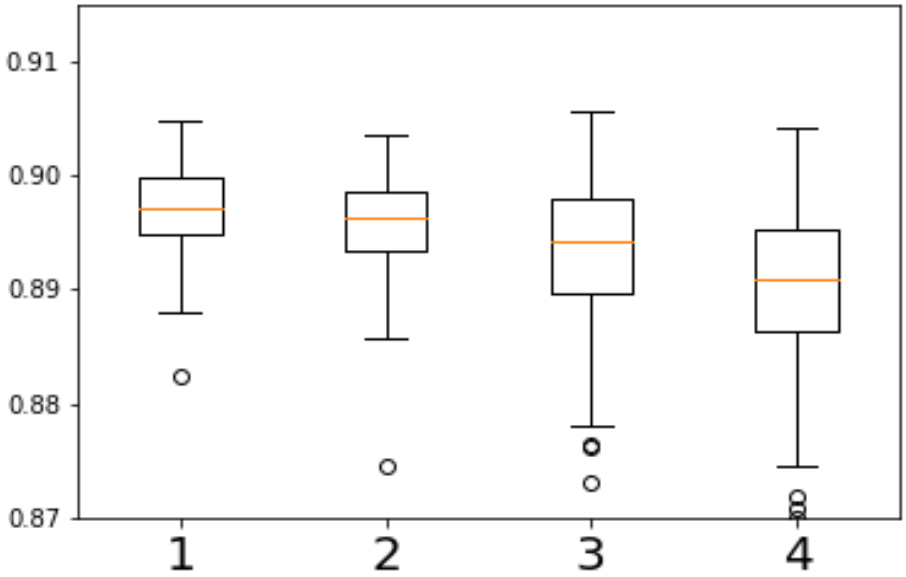
Celkem se podařilo natrénovat 508 sítí se slovními vektory o počtu složek 300 a 517 sítí se slovními vektory o počtu složek 100. Bylo zjištěno, že použitím slovních vektorů s počtem složek 300 je dosahováno lepší úspěšnosti než s vektory o počtu složek 100. Nejlepší úspěšnosti s vektory o počtu složek 100 bylo dosaženo 90,64 %. S vektory o počtu složek 300 bylo dosaženo nejlepší úspěšnosti 90,85 %.

Dále měl u konvolučních sítí největší vliv na dosahované úspěšnosti rozměr filtru. Lépe si vedly sítě s rozměrem filtru 2×300. Na obrázku č. 30 jsou zobrazeny dosahované úspěšnosti síti používající slovní vektory o počtu složek 300.



Obrázek 30: Porovnání dosažených úspěšností v závislosti na rozměru filtru

Lepší úspěšností celkově dosahovaly sítě s menším počtem konvolučních vrstev, jak je vidět na obrázku č. 31. Nicméně nejlepších úspěšností bylo dosaženo se sítí, která používala 3 konvoluční vrstvy. Lepších úspěšností je většinou dosahováno pomocí konvolučních sítí s vysokým počtem kovolučních vrstev. Využívají se zde ale ještě jiné techniky (např. normalizace dávky), které pomáhají s trénováním takovýchto sítí. Pro naši použitou architekturu by však zvýšení počtu konvolučních vrstev vedlo pravděpodobně ke snížení úspěšnosti sítě.



Obrázek 31: Porovnání dosažených úspěšností v závislosti na počtu konvolučních vrstev

Pro trénování na celé datové sadě byla opět vybrána síť s nejlepší průměrnou úspěšností na validačních podmnožinách a to konkrétně 90.55 %. Tato síť dosáhla také nejlepší úspěšnosti na jedné z validačních podmnožin a to 90.85 %. Parametry této sítě jsou:

- Počet slovních vektorů: 30 000
- Počet složek slovních vektorů: 300
- Počet konvolučních vrstev: 3
- Počet neuronů v konvolučních vrstvách:
  - 1. konvoluční vrstva: 182
  - 2. konvoluční vrstva: 164
  - 3. konvoluční vrstva: 115
- Velikost filtru: 2×300
- Pooling: globální průměrný pooling
- Počet neuronů v předposlední plně propojené vrstvě: 16
- Aktivační funkce: ReLU
- Dropout: 0,28
- Velikost dávky: 240

- Optimalizační algoritmus: RMSprop
- Míra učení: 0,0015
- Parametr ρ: 0,88

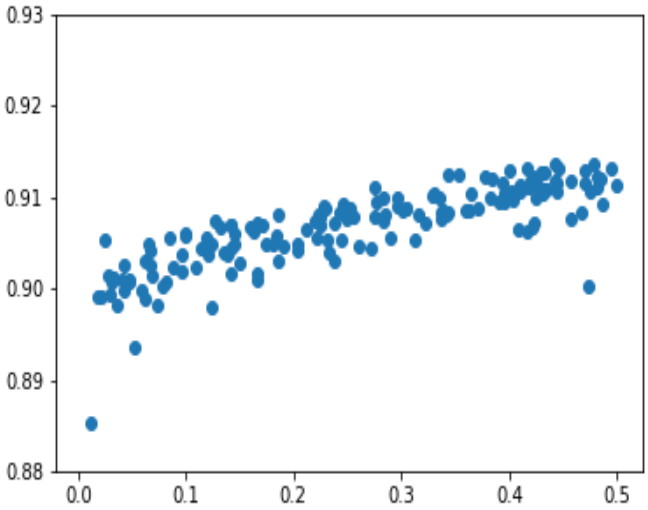
Průběh trénování tohoto modelu je zobrazen na obrázku č. 35 nacházejícího se v příloze A.

7.2.3 Rekurentní neuronová síť

Architektura sítě používající rekurentní vrstvu byla podobná jako u konvoluční sítě. Konvoluční vrstvy byly nahrazeny vrstvami rekurentními s tím, že byly použity všechny výstupy $\vec{h}^{(t)}$, které sloužily buď jako posloupnost vstupů do další rekurentní vrstvy nebo následoval globální pooling. Rekurentní sítě již byly trénovány pouze na slovních vektorech o počtu složek 300. Pro náhodné vyhledávání byly použity následující parametry:

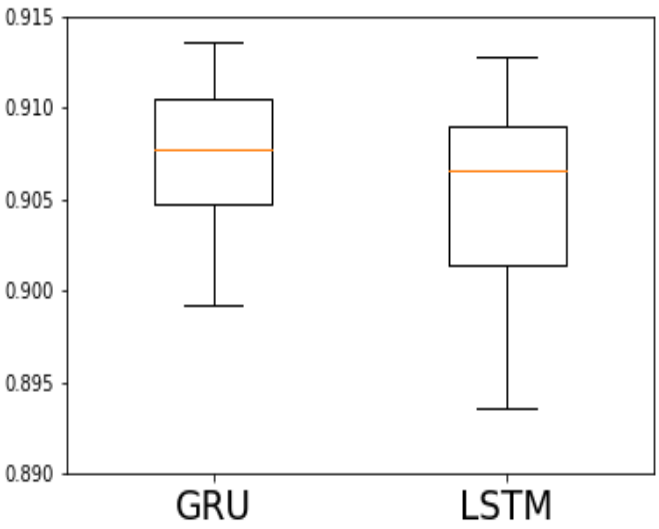
- Počet uvažovaných slov: 30 000
- Dimenze slovních vektorů: 300
- Typ rekurentní vrstvy: LSTM nebo GRU
- Obousměrná vrstva: ano nebo ne
- Počet rekurentních vrstev: 1–2
- Počet jednotek v rekurentních vrstvách:
  - 1. rekurentní vrstva: 40–200
  - 2. rekurentní vrstva: 30–počet jednotek v 1. rekurentní vrstvě
- Pooling: globální průměrný pooling
- Počet neuronů v plně propojené vrstvě: 5–100
- Aktivační funkce v plně propojené vrstvě: sigmoida nebo ReLU
- Počet neuronů ve výstupní vsrtvě: 1
- Aktivační funkce neuronu ve výstupní vrstvě: sigmoida
- Dropout: $(0; 0, 5)$
- Trénovací algoritmus: RMSprop
- Míra učení μ: $(0, 0001;\ 0, 005)$
- Parametr ϱ: $(0, 6;\ 0, 95)$
- Velikost Dávky: $\{10, 20, \ldots, 500\}$
- Chybová funkce: binary cross-entropy

Rekurentních sítí bylo natrénováno celkem 169. Tento počet byl nižší oproti plně propojeným a konvolučním sítím díky větší časové náročnosti na trénování rekurentní sítě. Největší vliv na úspěšnost sítě měla velikost pravděpodobnosti vynechání neuronu. Závislost dosažené úspěšnosti sítě na velikosti pravděpodobnosti vynechání je zobrazena na obrázku č. 32.



Obrázek 32: Úspěšnosti rekurentní sítě v závislosti na pravděpodobnosti vynechání neuronu

Volba typu rekurentní vrstvy neměla na úspěšnost sítě nijak zásadní vliv. Pravděpodobně je to dáno tím, že typ rekurentní vrstvy GRU a LSTM fungují na velmi podobném principu. V případě použití jednoduché rekurentní vrstvy by zde pravděpodobně nějaký rozdíl byl, protože tato vrstva nevyužívá brány. Dosahované úspěšnosti sítí s vrstvami GRU a LSTM jsou zobrazeny na obrázku č. 33. Zde je vidět, že nejlepší dosažená úspěšnost obou vrstev je přibližně stejná, celkově ale dosahovali sítě s vrstvou GRU mírně stabilnějších výsledků.



Obrázek 33: Porovnání dosažených úspěšností v závislosti na zvoleném typu rekurentní vrstvy

Pro trénování na celé datové sadě byla vybrána síť s nejlepší průměrnou maximální úspěšností na validačních množinách. Tato síť dosáhla průměrné úspěšnosti 91,36 % a její parametry jsou následující:

- Počet slovních vektorů: 30 000
- Počet rekurentních vrstev: 1
- Počet jednotek v rekurentní vrstvě: 100
- Typ rekurentní vrstvy: GRU
- Obousměrná vrstva: ano
- Pooling: globální průměrný pooling
- Počet neuronů v předposlední plně propojené vrstvě: 8
- Aktivační funkce: sigmoida
- Dropout: 0,48
- Velikost dávky: 290
- Optimalizační algoritmus: RMSprop
- Míra učení μ: 0,0019
- Parametr ϱ: 0,75

Průběh trénování tohoto modelu je zobrazen na obrázku č. 36 nacházejícího se v příloze A.

7.3 Analýza na celé datové sadě

Pro trénování sítě na celé trénovací části datové sady bylo nejprve potřeba změnit způsob převádění textu do TF-IDF vektorů. Všech 3 600 000 recenzí převedených do TF-IDF vektorů při použití 16 bitového reálného čísla mělo velikost okolo 67 GiB (72 GB). Pokus o natrénování sítě se všemi recenzemi převedenými do TF-IDF vektorů tak skončil chybou celého programu z důvodu nedostatku paměti RAM. Bylo tedy nutné provádět převod textu do TF-IDF vektorů pouze pro aktuální trénovací dávku a ne pro všechny recenze najednou.

Pro závěrečné trénování sítě bylo rozděleno všech 3 600 000 recenzí ze souboru *train.csv* na trénovací množinu o 3 200 000 recenzí a validační množinu o 400 000 recenzí. Protože došlo k výraznému nárůstu počtu trénovacích dat, oproti těm použitým k náhodnému vyhledávání, bylo provedeno pár experimentů s různě nastavenou velikostí dávek, míry učení a parametru ρ. Bylo zjištěno, že zvýšením hodnoty parametru ρ na hodnotu větší než 0,9 došlo k poměrně výraznému zlepšení. Síť s hyper-parametry nalezenými pomocí náhodného vyhledávání dosáhla úspěšnosti 90,77 %. Po zvýšení parametru ρ na hodnotu větší než 0,9 byla tato úspěšnost překonána již po první epoše a síť dosahovala úspěšnosti okolo 91,7 %. Bylo to velmi pravděpodobně způsobeno tím, že výrazným zvětšením počtu recenzí v trénovací množině nedocházelo k tak výraznému přetrénování sítě. Díky tomuto významnému vlivu velikosti trénovací množiny bylo znovu natrénováno několik sítí s použitím křížové validace, kde byl nastaven parametr $\rho \in \{0,7; 0,9; 0,95; 0,98; 0,99\}$ spolu s kombinací velikostí dávky 310 a $\mu = 0,0007$ nebo velikost dávky 1000 a $\mu = 0,001$. Ostatní kombinace hyper-parametrů zůstaly stejné, protože se předpokládalo, že nebudou mít tak velký vliv. Bylo by také možné vyzkoušet síť s větším počtem neuronů ve skrytých vrstvách, ale další hledání vhodné kombinace hyper-parametrů nebylo provedeno kvůli časové náročnosti. Sítě byl trénovány 10 epoch. Natrénované sítě byl pokaždé uloženy, došlo-li ke zlepšení úspěšnosti na validační množině. Uložené sítě se použily pro vyhodnocení na recenzích ze souboru *test.csv*.

Nejlepších výsledků bylo dosaženo pomocí sítě s $\rho = 0,99$, $\mu = 0,001$ a velikostí dávky 1000. Sítí s touto kombinací hyper-parametrů bylo dosaženo úspěšnosti v průměru 91,89 % na třech validačních podmnožinách. Pro finální ohodnocení byla použita síť s těmito parametry, která dosáhla největší úspěšnosti na jedné z validačních podmnožin a to konkrétně 91,91 %. Tato natrénovaná síť dosáhla na testovacích recenzí úspěšnosti 91,82 %.

	Předpovězené kategorie				
Míra učení μ	0,0007			0,001	
Parametr ϱ	0,72			0,99	
Velikost dávky	310			1000	
Skutečné kategorie		Negativní	Pozitivní	Negativní	Pozitivní
	Negativní	181323	18677	184066	15934
	Pozitivní	18576	181424	16785	183215
Úspěšnost (%)		90,69 %		91,82 %	

Tabulka 1: Výsledky dosažené pomocí plně propojené neuronové sítě

Pro konvoluční a rekurentní sítě nebylo potřeba měnit způsob převádění textu. Po zkušenosti s plně propojenou vrstvou byly vyzkoušeny různé kombinace parametru ρ s velikostí dávky. Počet epoch byl nastaven na 100. Colab omezuje dobu použití GPU a po uplynutí různě dlouhé doby (která bývá maximálně 12 hodin za den) dojde k odpojení, proto tyto sítě nebyly trénovány celých 100 epoch. Sítě byly průběžně ukládány přímo na službu Google Disk, pokud došlo ke zlepšení úspěšnosti na validační množině. Konvoluční sít dosáhla nejlepší úspěšnosti na validační množině s parametrem $\rho = 0,95$ a velikostí dávky 1000. Na jedné z validačních množin síť dosáhla úspěšnosti 94,03 %. Na testovacích recenzích síť dosáhla úspěšnosti 93,96 %.

	Předpovězené kategorie				
Míra učení μ Parametr ϱ Velikost dávky	0,0015 0,885 1000			0,0015 0,95 1000	
Skutečné kategorie		Negativní	Pozitivní	Negativní	Pozitivní
	Negativní	187903	12097	187999	12001
	Pozitivní	12510	187490	12163	187837
Úspěšnost (%)		93,85 %		93,96 %	

Tabulka 2: Výsledky dosažené pomocí konvoluční neuronové sítě

U rekurentních sítí byl opět nastaven počet epoch trénování na 100 a pro ukládání byla zvolena stejná strategie, jako v případě trénování konvoluční sítě. Parametr ρ neměl vliv na výslednou úspěšnost. Na testovací části datové sady dosáhla síť s původními a upravenými parametry téměř stejné úspěšnosti. Rozdíl však můžeme spatřit v počtu špatně (či správně) předpovězených negativních a pozitivních recenzí.

	Předpovězené kategorie				
Míra učení μ Parametr ϱ Velikost dávky	0,0019 0,75 1000			0,0019 0,95 1000	
Skutečné kategorie		Negativní	Pozitivní	Negativní	Pozitivní
	Negativní	189598	10402	187794	12206
	Pozitivní	13309	186691	11537	188463
Úspěšnost (%)		94,07 %		94,06 %	

Tabulka 3: Výsledky dosažené pomocí rekurentní neuronové sítě

7.4 Shrnutí

Nejlepší úspěšnosti 94,07 % bylo dosaženo pomocí rekurentní neuronové sítě. Za ní následovala síť konvoluční s úspěšnosti pouze o 0,11 % menší. Nejmenší úspěšnosti bylo dosaženo pomocí plně propojené sítě a to 91,82 %. Je pravděpodobné, že by se úspěšnosti ještě zvýšily, pokud by náhodné vyhledávání proběhlo na všech recenzí z trénovací části a ne pouze na 50 000 recenzí. Takovéto vyhledávání by však bylo

časově mnohem náročnější a je možné, že by k lepším výsledkům stačilo i pouhé experimentování. Lepší úspěšnosti by se také mohlo dosáhnout vyzkoušením jiného počtu uvažovaných slov jak v metodě TF-IDF, tak i v počtu uvažovaných slovních vektorů. Dále by se dalo experimentovat s pevnou délkou textu a vzhledem k velikosti datové sady by stálo za to vyzkoušet trénovaní složek slovních vektorů pomocí embedding vrstvy.

Trénování sítí probíhalo převážně v cloudové službe Google Colaboratory, ve které má uživatel zdarma přístup k výkoným grafickým kartám Nvidia Tesla P100-PCIE-16GB. Časově nejnáročnější bylo trénování rekurentních sítí, pro které byla použita speciální implementace CuDNN, která je optimalizovaná pro grafické karty. Tuto implementaci lze spustit pouze na nich. Časová náročnost trénování jedné epochy je zobrazena v tabulce č. 4.

	Plně propojená síť		CNN		CuDNN GRU	
Velikost dávky	290	1000	210	1000	290	1000
40 000 recenzí	3s	3s	4s	4s	12s	9s
3 200 000 recenzí	580s	450s	530s	470s	820s	700s

Tabulka 4: Čas trvání jedné epochy trénování v sekundách

V práci [30], pro kterou byla tato datová sada vytvořena bylo dosaženo lepšího výsledku 94,5% a to při převádění jednotlivých písmen na one-hot vektory. S anglickou abecedou a dalšími znaky bylo dohromady uvažováno celkem 70 znaků. Tuto síť tak nebylo potřeba omezovat na určitý počet uvažovaných slov. Nicméně stále docházelo ke zkracování textu na pevný počet znaků. Při použití augmentace textu, kde docházelo k nahrazení určitého počtu slov synonymy, bylo dosaženo 95,07 %. Ještě lepšího výsledku 95,7 % bylo dosaženo v práci [13], kde se také převáděla jednotlivá písmena v textu na one-hot vektory s tím, že se použila konvoluční síť, která měla dokonce až 29 konvolučních vrstev.

Závěr

V teoretické části této práce byly popsány neuronové sítě, gradientní sestup a jeho varianty, od kterého se pak odvíjí algoritmy s adaptivní mírou učení, jako je algoritmus RMSprop. Součástí popisu byl také výpočetní algoritmus, který se používá pro výpočet derivací v gradientním sestupu. Následně byly představeny nejpoužívanější metody jako je předčasné ukončení, dropout a náhodné vyhledávání, které se používají pro dosažení lepší úspěšnosti sítě. Popsány byly také chybové funkce vhodné pro klasifikační úlohy, včetně vysvětlení důvodů pro jejich případné zvolení.

V praktické části této práce byly poznatky představené v teoretické části aplikovány na analýzu sentimentu recenzí na datové sadě Amazon reviews polarity. Návrh sítí probíhal na 50 000 recenzí z trénovací části datové sady. Byla zde navržena plně propojená, konvoluční a rekurentní neuronová síť. Nejprve bylo provedeno několik experimentů, které stanovily rozsah hyper-parametrů použitý v náhodném vyhledávání. Sítě dosahovaly lepších úspěšností, pokud z textu recenze nebyla odstraněna stop slova nacházející se v seznamu knihovny NLTK a nebyla použita lemmatizace slov. Byl zde dále ukázán vliv některých parametrů na úspěšnost sítě. Nejlepší úspěšnosti dosahovaly sítě rekurentní.

Nalezené hyper-parametry byly použity pro natrénování sítě na celé trénovací části datové sady. Zde se po pár experimentech ukázalo, že výrazným zvětšením trénovací množiny bylo potřeba některé hodnoty hyper-parametrů upravit. Optimální hyper-parametry se tak mění s rostoucím počtem trénovacích dat. Důvodem je pravděpodobně především pokles přetrénování sítě. Tento rozdíl byl největší u plně propojené sítě. Vhodnější by tedy bylo provést náhodné vyhledávání na celé části trénovacích dat i přes pokles počtu prohledaných kombinací a zvětšení časové náročnosti. Sítě byly po úpravě parametrů natrénovány na celé trénovací části a vyhodnoceny na testovacích recenzí datové sady. Nejlepšího úspěšnosti bylo dosaženo 94,07% a to rekurentní sítí používající obousměrnou vrstvu GRU.

Seznam použité literatury

[1] ABADI, Martin, et al. Google Brain. *TensorFlow 2.2.0: A System for Large-Scale Machine Learning* [software]. 6. 5. 2020 [cit. 17. 7. 2020]. Dostupné z: https://www.tensorflow.org/

[2] ALMRYAD, Ayad Saad, KUTUCU, Hakan. Automatic identification for field butterflies by convolutional neural networks. *Engineering Science and Technology, an International Journal* [online]. Elsevier. February 2020, 23(1), 189–195 [cit. 8. 7. 2020]. ISSN: 2215-0986. DOI: 10.1016/j.disc.2007.10.026.

[3] BENGIO, Yoshua. Practical Recommendations for Gradient-Based Training of Deep Architectures. *arXiv preprint 1206.5533v2 [cs.LG]* [online]. 16. 9. 2012 [cit. 8. 7. 2020]. Dostupné z: https://arxiv.org/pdf/1206.5533v2.pdf.

[4] BERGSTRA, James, BARDENET, Remi, BENGIO, Yoshua, KÉGL, Balázs, Algorithms for Hyper-Parameter Optimization. In: *Advances in Neural Information Processing Systems 24* [online]. Curran Associates Inc., 2011, s. 2546–2554 [cit. 8. 7. 2020]. ISBN 9781618395993. Dostupné z: https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf.

[5] BERGSTRA, James, BENGIO, Yoshua. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* [online]. JMLR.org. 2012, 13(10), 281–305 [cit. 8. 7. 2020]. ISSN 1532-4435. Dostupné z: http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf.

[6] BIRD, Steven, LOPER, Edward a KLEIN, Ewan. *NLTK 3.2.5: the Natural Language Toolkit* [software], 24. 9. 2017 [cit. 17. 7. 2020]. Dostupné z: https://www.nltk.org/

[7] BROWNLEE, Jason. A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks. In: *Machine Learning Mastery* [online]. Datum aktualizace 6. 8. 2019 [cit. 8. 7. 2020]. Dostupné z: https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/.

[8] BROWNLEE, Jason. A Gentle Introduction to k-fold Cross-Validation. In: *Machine Learning Mastery* [online]. Datum aktualizace 8. 9. 2018 [cit. 15. 5. 2020]. Dostupné z: https://machinelearningmastery.com/k-fold-cross-validation/.

[9] BROWNLEE, Jason. What Are Word Embeddings for Text? In: *Machine Learning Mastery* [online]. Datum aktualizace 11. 10. 2017 [cit. 22. 5. 2020]. Dostupné z: https://machinelearningmastery.com/what-are-word-embeddings/.

[10] BUITINCK, L. et al. *scikit-learn 0.22.1* [software]. 2. 1. 2020 [cit. 17. 7. 2020]. Dostupné z: https://scikit-learn.org/stable/

[11] BUSHAEV, Vitaly. Understanding RMSprop — faster neural network learning. In: *Towards Data Science* [online]. 2. 9. 2018 [cit. 21. 5. 2020]. Dostupné z: https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a.

[12] CHOLLET, François et al. *Keras 2.3.1: The Python Deep Learning library* [software]. 7. 10. 2019 [cit. 17. 7. 2020]. Dostupné z: https://keras.io

[13] CONNENAU, Alexis, SCHWENK, Holger, LECUN, Yann, BARRAULT, Löc. Very deep convolutional networks for text classification. In: *15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017 - Proceedings of Conference* [online]. Association for Computational Linguistics, 2017, s. 1107–1116 [cit 8. 7. 2020]. Dostupné z: https://www.aclweb.org/anthology/E17-1104.pdf.

[14] Google Research. *Google Colaboratory* [online]. 2017 [cit. 17. 7. 2020]. Dostupné z: https://colab.research.google.com/

[15] HAMMEL, B. D. *What learning rate should I use?* [online]. Datum publikování 23. 3. 2019 [cit 21. 5. 2020]. Dostupné z: http://www.bdhammel.com/learning-rates/.

[16] HAO, Li, ZHENG, Xu, GAVIN, Taylor and GOLDSTEIN, Tom. Visualizing the loss landscape of neural nets. *arXiv preprint 1712.09913v3 [cs.LG]* [online]. 7. 11. 2018 [cit. 8. 7. 2020]. Dostupné z: https://arxiv.org/pdf/1712.09913v3.pdf.

[17] HUNTER, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* [online]. IEEE COMPUTER SOC. 2007, 9(3), 90–95 [cit 17. 7. 2020]. DOI: 10.1109/MCSE.2007.55.

[18] JŮZLOVÁ, Markéta. *Model Performance Approximation in Hyper-parameter Optimization* [online]. Praha, 2018 [cit. 8. 7. 2020]. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018. Dostupné z: https://dspace.cvut.cz/bitstream/handle/10467/76421/F8-DP-2018-Juzlova-Marketa-thesis.pdf?sequence=-1&isAllowed=y.

[19] MYŠKA, Vojtěch. *Rekurentní neuronové sítě pro klasifikaci textů* [online]. Brno, 2018 [cit. 8. 7. 2020]. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=171408.

[20] NGUYEN, Van and YANG, Hyung-Jeong. Real-time event detection using recurrent neural network in social sensors. *International Journal of Distributed Sensor Networks* [online]. 2019, 15(6) [cit. 8. 7. 2020]. ISSN: 1550-1477. DOI: 10.1177/1550147719856492.

[21] NIELSEN, Michael. *Neural Networks and Deep Learning* [online]. Determination Press, 2015. Datum aktualizace 26. 12. 2019 [cit. 8. 7. 2020]. Dostupné z:
http://neuralnetworksanddeeplearning.com/index.html.

[22] OLAH, Christopher. *Understanding LSTM Networks* [online]. 27. 8. 2015 [cit. 8. 7.2020]. Dostupné z:
http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[23] ŘEHŮŘEK, Radim a SOJKA, Petr. *Gensim 3.6.0* [software]. 20. 9. 2018 [cit. 17. 7. 2020]. Dostupné z:
https://radimrehurek.com/gensim/

[24] SAHA, Sumit. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. In: Towards Data Science [online]. Datum publikování 15. 12. 2018 [cit. 8. 7. 2020]. Dostupné z:
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[25] SHRIDHAR, Kumar et al. ProbAct: A Probabilistic Activation Function for Deep Neural Networks. *arXiv preprint 1905.10761v1 [cs.LG]* [online]. 26. 5. 2019 [cit. 8. 7. 2020]. Dostupné z:
https://arxiv.org/pdf/1905.10761v1.pdf.

[26] SRISTAVA, Nitish at al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* [online]. JMLR.org. 2014, 15(56), 1929–1958 [cit. 8. 7. 2020]. ISSN 1532-4435. Dostupné z:
http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf.

[27] TANTAU, Till, Graph Drawing in TikZ. In: *Proceedings of the 20th International Conference on Graph Drawing* [online]. Springer-Verlag, 2013, s. 517–528 [cit 17. 7. 2020]. ISBN: 978-3-642-36762-5. DOI: 10.1007/978-3-642-36763-2_46.

[28] Wikipedia contributors. Gradient descent. In: *Wikipedia, The Free Encyclopedia* [online]. Datum poslední revize 6. 7. 2020 [cit. 8. 7. 2020]. Dostupné z:
https://en.wikipedia.org/wiki/Gradient_descent.

[29] YAMADA, Ikuya et al. Wikipedia2Vec: An Efficient Toolkit for Learning and Visualizing the Embeddings of Words and Entities from Wikipedia. *arXiv preprint 1812.06280v3 [cs.CL]* [online]. 30. 1. 2020 [cit. 8. 7. 2020]. Dostupné z:
https://arxiv.org/pdf/1812.06280v3.pdf.

[30] ZHANG, Xiang, ZHAO, Junbo and LECUN, Yann. Character-level Convolutional Networks for Text Classification. In: *Advances in Neural Information Processing Systems 28* [online]. Curran Associates, Inc., 2015, s. 649–657 [cit. 8. 7. 2020] ISBN 9781510825024. Dostupné z:
https://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.pdf.

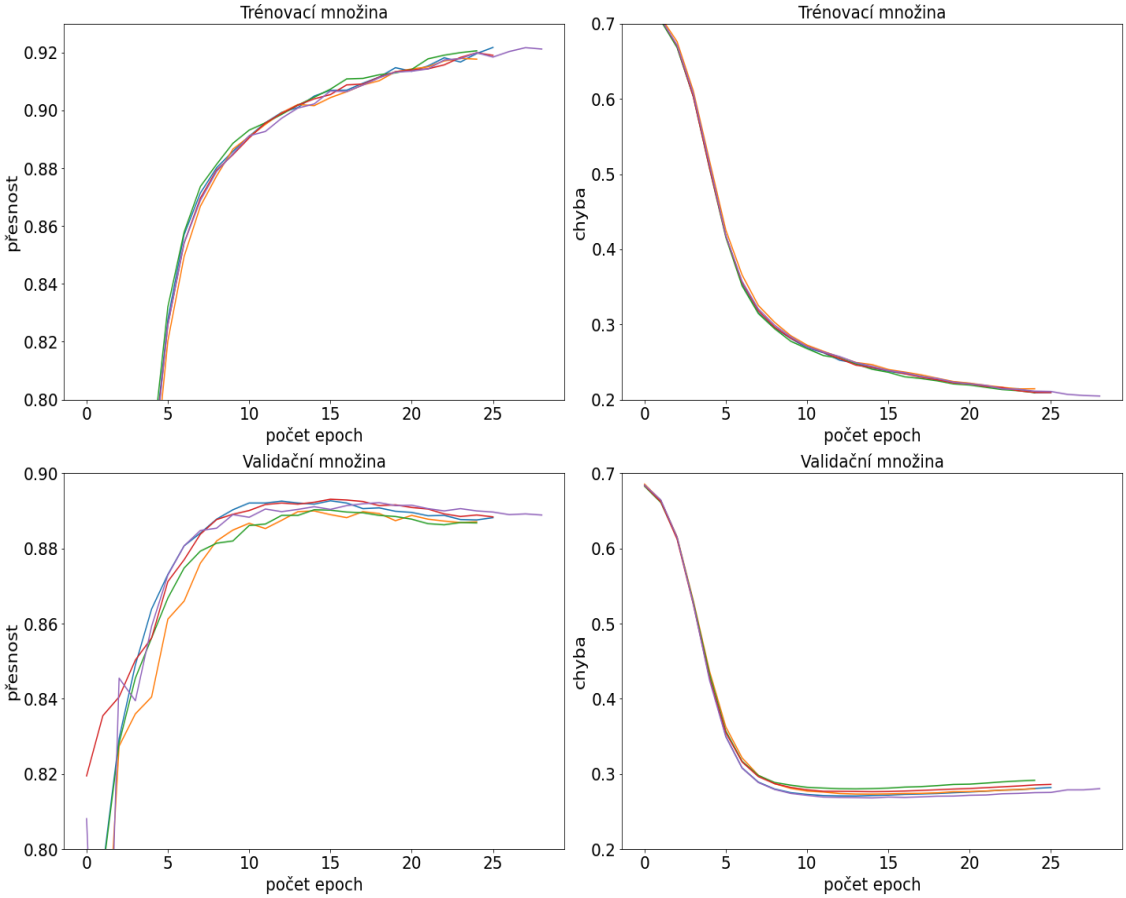
Seznam zkratek

API	Application programming interface
CBOW	Countinuous bag of words
CNN	Convolutional neural network
CUDA	Compute Unified Device Architecture
CuDNN	NVIDIA CUDA Deep Neural Network library
GB	Gigabyte
GiB	Gibibyte
GloVe	Global Vectors for word representation
GRU	Gated recurrent unit
JSON	JavaScript Object Notation
LSTM	Long short-term memory
MB	Megabyte
RAM	Random access memory
ReLU	Rectified linear unit
RMSprop	Root mean square propagation
TF-IDF	Term frequency-inverse document frequency

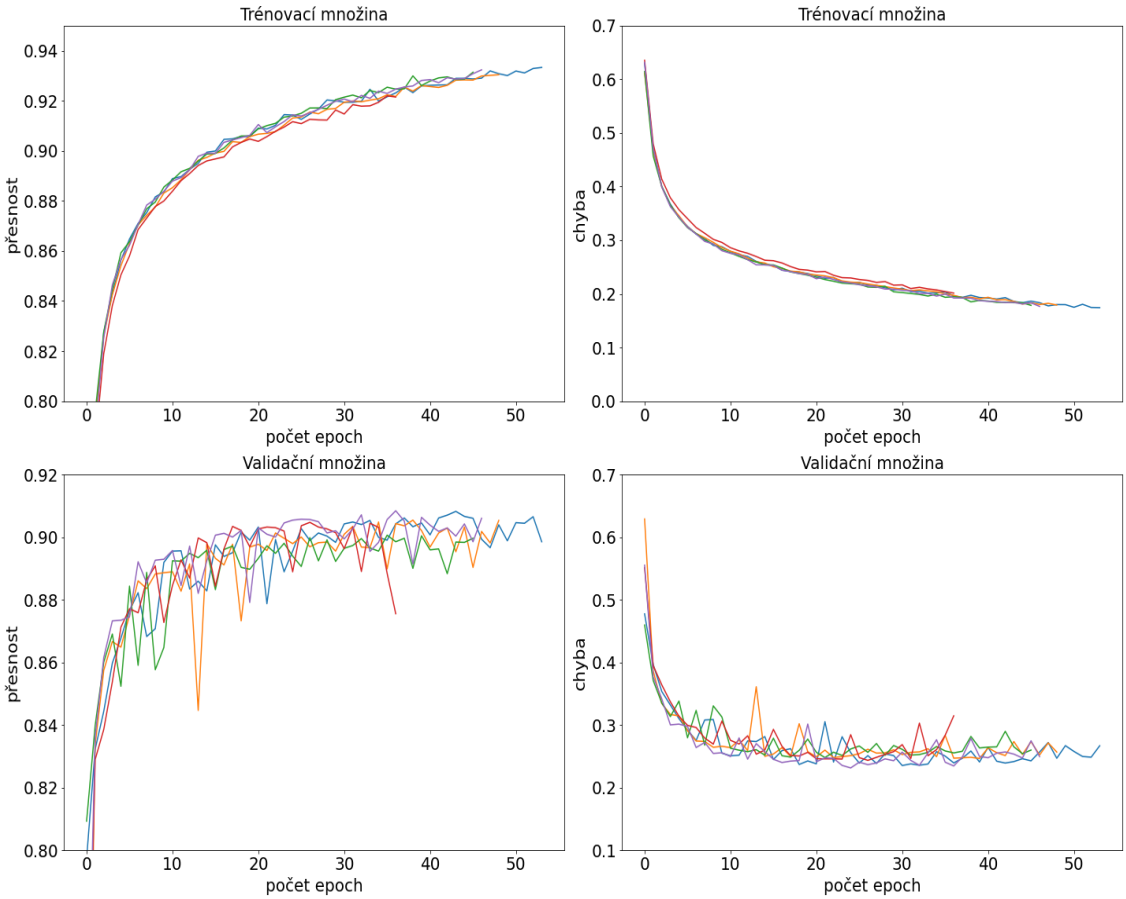
Seznam obrázků

Seznam tabulek

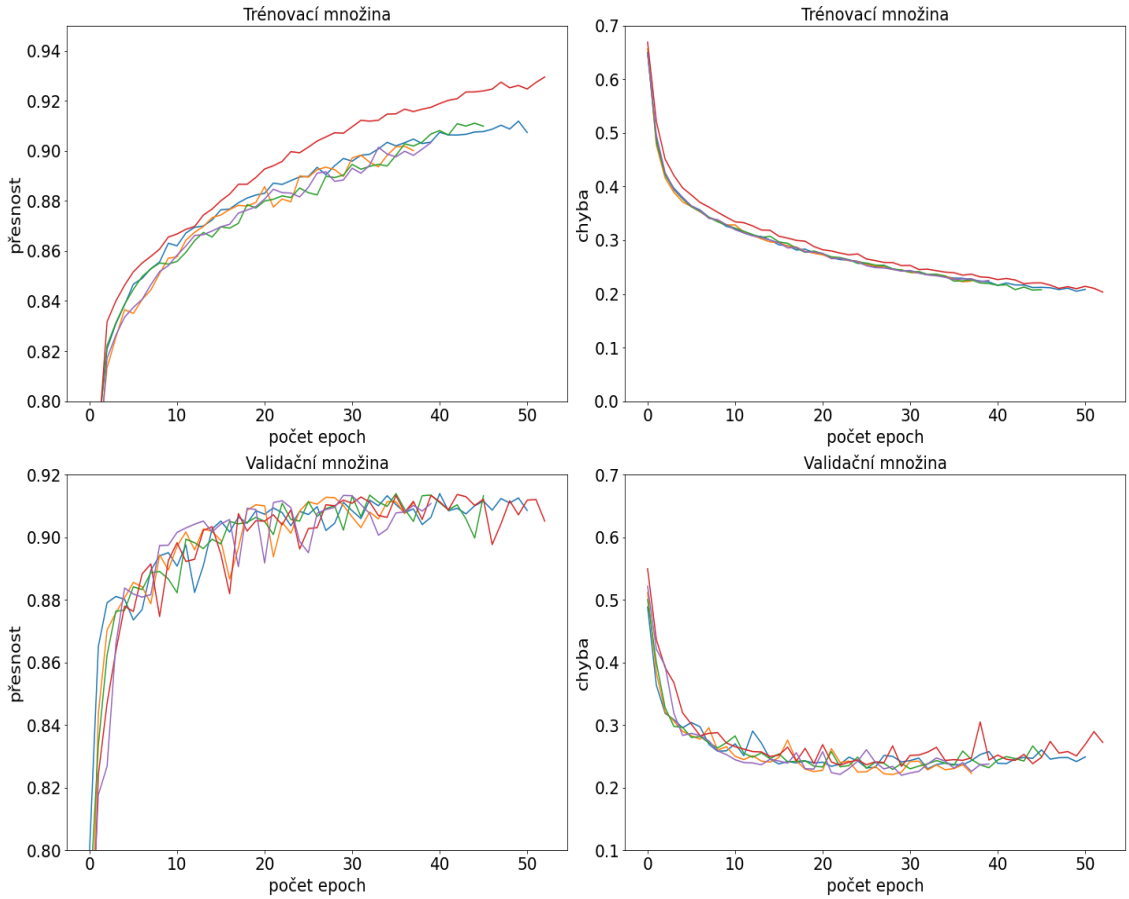
A Průběh trénování



Obrázek 34: Naměřené hodnoty v průběhu trénování plně propojené neuronové sítě s nejlepšími výsledky dosaženými v náhodném vyhledávání



Obrázek 35: Naměřené hodnoty v průběhu trénování konvoluční neuronové sítě s nejlepšími výsledky dosaženými v náhodném vyhledávání

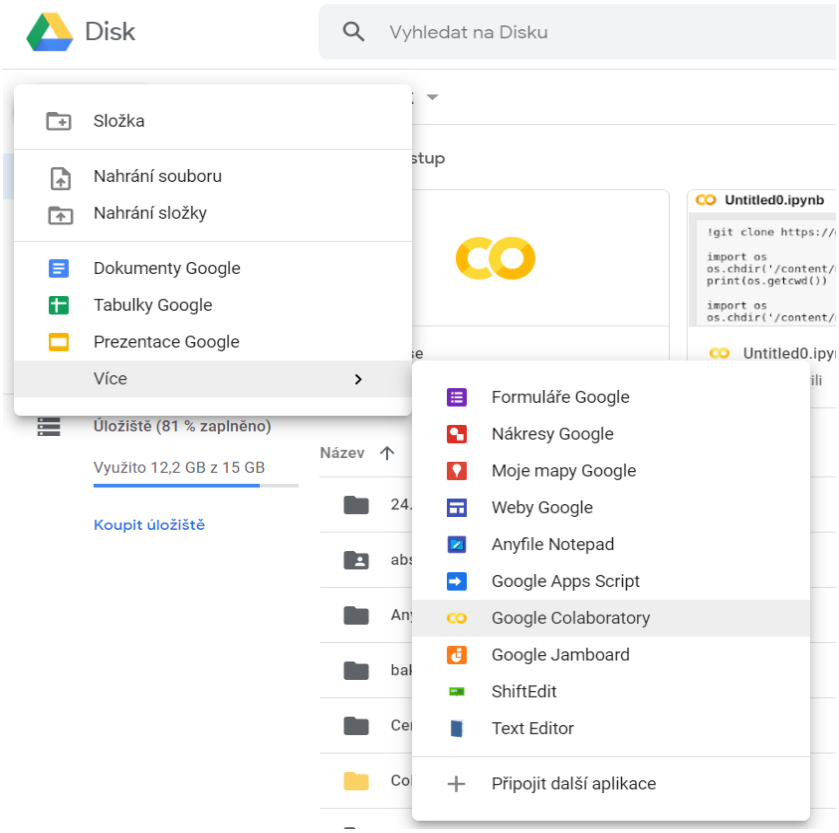


Obrázek 36: Naměřené hodnoty v průběhu trénování rekurentní neuronové sítě s nejlepšími výsledky dosaženými v náhodném vyhledávání

B Zprovoznění praktické části

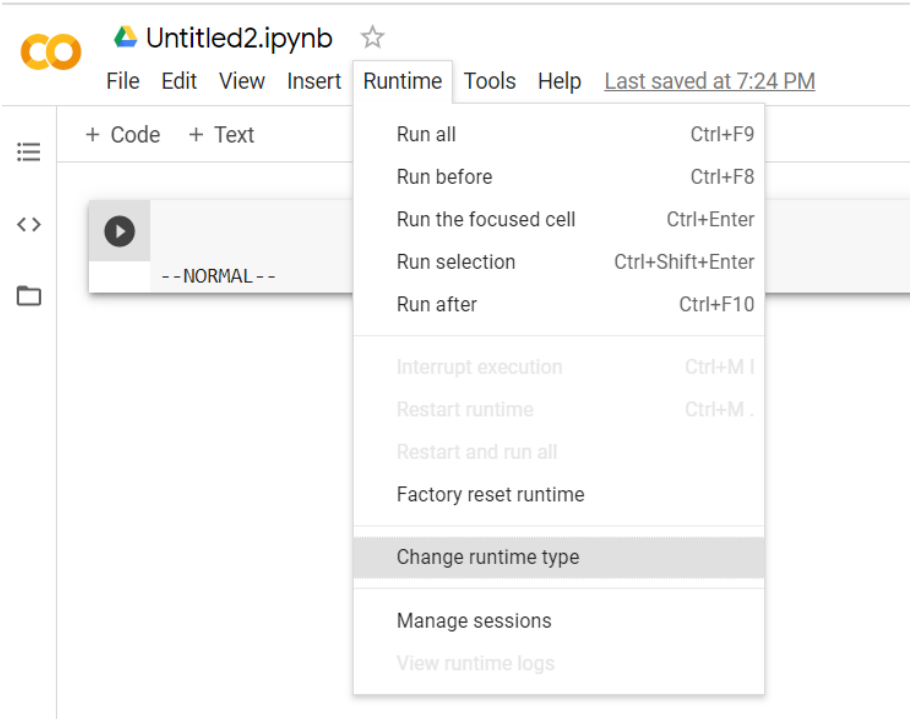
Repozitář s praktickou částí je nahrán na webové službě GitHub a je volně dostupný na adrese: `https://github.com/Tomiczeq/neuronove_site_a_jejich_aplikace`. Repozitář je možné stáhnout v souboru ZIP nebo naklonovat pomocí nástroje git.

Zprovoznění je možné ve službě Google Colaboratory. K využití této služby je nutné mít účet Google. Golaboratory notebook můžeme vytvořit ze služby Google disk kliknutím na tlačítko přidat.

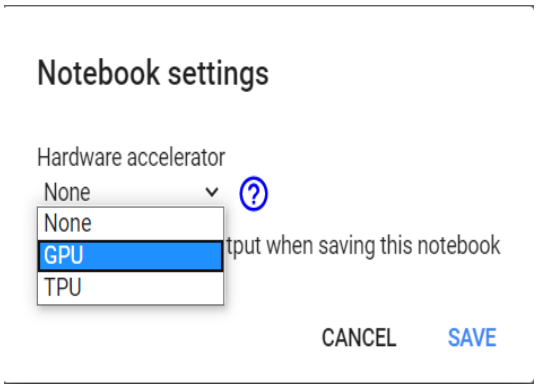


Obrázek 37: Vytvoření Colaboratory notebooku

Po vytvoření změníme běhové prostředí na GPU.



Obrázek 38: Změna běhového prostředí



Obrázek 39: Změna běhového prostředí na GPU

Po změně běhového prostředí spustíme v buňce následující příkaz, který naklonuje repozitář obsahující soubory a zdrojové kódy potřebné pro zprovoznění praktické části.

```
!git clone https://github.com/Tomiczeq/neuronove_site_a_jejich_aplikace.git
```

Nyní je nutné změnit pracovní adresář do tohoto repozitáře. Pracovní adresář lze změnit pomocí následujících příkazů. Tyto příkazy napíšeme do buňky a poté ji spustíme.

```
import os
os.chdir("/content/neuronove_site_a_jejich_aplikace")
```

Datová sada je v repozitáři ve složce `data` rozdělená na několik částí, protože je ve službě Github nastaven limit pro velikost jednoho souboru na 100MB. Do další buňky je tedy nutné napsat následující příkaz, který části sjednotí do jednoho zkomprimovaného souboru.

```
!cat data/amazon_review_polarity_csv.tar.gz.part* > \
     data/amazon_review_polarity_csv.tar.gz
```

Soubor s datovou sadou dekomprimujeme následujícím příkazem.

```
!tar -zxvf data/amazon_review_polarity_csv.tar.gz && \
     mv amazon_review_polarity_csv/* data/
```

Následujícím příkazem nainstalujeme potřebné verze knihoven.

```
!pip install -r requirements_colab.txt
```

Nyní je vše připravené pro spuštění skriptů. Ukážeme sadu příkazu pro spuštění náhodného vyhledávání hyper-parametrů plně propojené sítě. Rozsah hyper-parametrů je stejný, jako v části 7.2.1. Následující příkaz převede data do formátu JSON. S tímto formátem se poté jednodušeji pracuje.

```
!python3 preprocessing/format.py --filepath=data/train.csv \
         --savepath=data/train.json
```

Následujícím příkazem vytvoříme soubor s 50 000 recenzí pro náhodné vyhledávání.

```
!python3 preprocessing/search_split.py --filepath=data/train.json \
         --savepath=data/search_train.json --size=50000
```

Nyní provedeme předzpracování textu popsané v podkapitole 7.1 bez lemmatizace a odstranění stop slov.

```
!python3 preprocessing/text_preprocessing.py \
         --filepath=data/search_train.json \
         --savepath=data/search_data.json
```

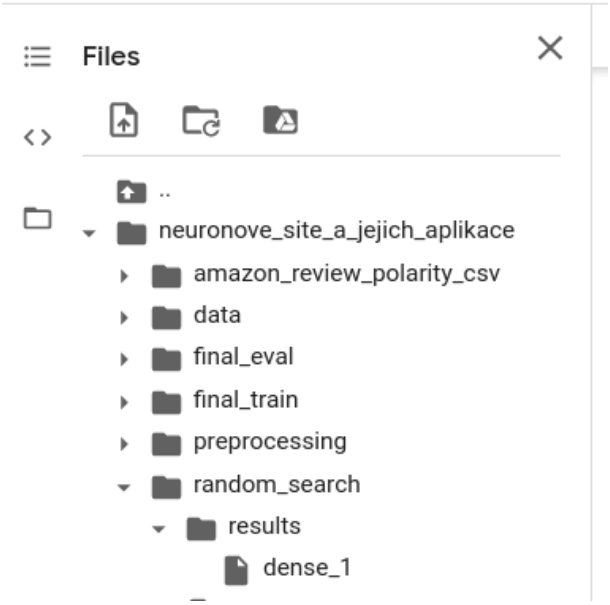
Pro případné použití lemmatizace a odstranění stop slov lze spustit následující příkazy.

```
import nltk
nltk.download("stopwords")
nltk.download("punkt")
nltk.download("averaged_perceptron_tagger")
nltk.download("wordnet")
!python3 preprocessing/lemmatize.py \
         --filepath=data/search_data.json \
         --savepath=data/search_data.json
```

Nyní máme připravené recenze pro náhodné vyhledávání. Následující příkaz spustí náhodné vyhledávání hyper-parametrů pro plně propojenou síť. Je zde možné nastavit počet iterací náhodného vyhledávání pomocí argumentu `--iterations`. Dále lze nastavit počet podmnožin v křížové validaci pomocí `--n_splits`. Pro vyzkoušení je zde nastavena jedna iterace a dvě podmnožiny. Pro každou kombinaci hyper-parametrů bude vytvořen textový soubor ve formátu JSON s názvem definovaným v argumentu `--name`. K tomuto názvu bude za podtržítkem připojené pořadové číslo prohledávané kombinace. Toto číslo bude pokaždé jiné, aby nedocházelo k přepsání již vytvořených souborů s průběhem trénování. Aby toto číslování správně fungovalo, nesmí být v argumentu `--name` obsažené podtržítko. Všechny soubory budou vytvořeny v adresáři nacházejícího se na relativní cestě definované argumentem `--savedir`.

```
!python3 random_search/random_search_dense.py \
        --datapath=data/search_data.json \
        --savedir=random_search/results \
        --name=dense --iterations=1 --n_splits=2
```

Soubory se ukládají průběžně po každé iteraci. Po první iteraci, spuštěné předchozím příkazem, se vytvoří soubor `dense_1`. Tento soubor bude vidět v levé liště jak je zobrazeno na obrázku č. 40 a dvojitým poklikem lze stáhnout.



Obrázek 40: Cesta uložení souboru s naměřenými hodnotami průběhu trénování

Pro trénování na všech recenzí ze souboru *test.csv* je nutné připravit všechny recenze. Tyto recenze již byly převedeny do formátu JSON pomocí následujícího příkazu a není třeba ho znovu pouštět.

```
!python3 preprocessing/format.py --filepath=data/train.csv \
        --savepath=data/train.json
```

Nyní provedeme předzpracování textu všech recenzí pomocí následujícího příkazu.

```
!python3 preprocessing/text_preprocessing.py \
        --filepath=data/train.json \
        --savepath=data/train_data.json
```

Následujícím příkazem dojde k připravení tokenizérů pro převod textu do TF-IDF vektorů.

```
!python3 preprocessing/pre_tok.py --datapath=data/train.json \
        --savepath=data/tokenizer
```

Recenze pro trénování a tokenizéry jsou nyní připraveny. Následujícím příkazem je možné spustit finální trénování sítí s nejlepšími parametry vypsanými v části 7.2.1. Dále je zde možné pomocí argumentů nastavit míru učení, velikost dávky, parametr ρ a počet epoch trénování.

```
!python3 final_train/final_dense.py --name=dense1000n099 \
        --datapath=data/train_data.json \
        --results_savedir=final_train/results \
        --models_savedir=final_train/saved_models \
        --lr=0.001 --batch_size=1000 --rho=0.99 --epochs=1
```

Pokud bude nastavena jedna epocha, bude tento skript běžet asi jednu hodinu. Protože od provedení náhodného vyhledávání byla na službe Colab omezena dostupná paměť RAM z 25GB na 12,5GB, bylo nutné skript pro finální trénování plně propojené sítě upravit a trénování tak trvá déle, než je zobrazeno v tabulce č. 4. Natrénované sítě budou uloženy v relativní cestě definované pomocí argumentu `--models_savedir`. Pro testování natrénovaných modelů musíme nejdříve připravit recenze ze souboru *test.csv* pomocí následujících dvou příkazů.

```
!python3 preprocessing/format.py --filepath=data/test.csv \
        --savepath=data/test.json
!python3 preprocessing/text_preprocessing.py \
        --filepath=data/test.json \
        --savepath=data/test_data.json
```

Následujícím příkazem otestujeme jeden z nich na rencenzích ze souboru *test.csv*.

```
!python3 final_eval/final_evaluate_dense.py \
        --datapath=data/train_data.json \
        --modelpath=final_tran/saved_models/dense1000n0990 \
        --tokenizerpath=data/tokenizer0.json
```

Výsledný výstup testu bude vypadat podobně, jako na obrázku č. 41.

```
              precision    recall  f1-score   support

           0     0.9171    0.9165    0.9168   1800000
           1     0.9165    0.9171    0.9168   1800000

    accuracy                         0.9168   3600000
   macro avg     0.9168    0.9168    0.9168   3600000
weighted avg     0.9168    0.9168    0.9168   3600000

correct          0        1
predicted
0.0        1649620   149190
1.0         150380  1650810
Done
```

Obrázek 41: Výsledek testu plně propojené sítě trénované po jednu epochu

Stručný popis celého repozitáře a příkazy potřebné pro zprovoznění trénování konvoluční a rekurentní neuronové sítě jsou popsány v souboru `README.md`. Repozitář také obsahuje naměřené hodnoty z náhodného vyhledávání a finálního trénování.