Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

BAKALÁŘSKÁ PRÁCE

Plzeň 2016 **Jan Šmejkal**

Před svázáním místo téhle stránky vložit zadání práce s podpisem děkana.

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou/diplomovou práci vypracoval(a) samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne ....

....

Jan Šmejkal

Poděkování

Chtěl bych tímto poděkovat panu Ing. Janu Švecovi, Ph.D. za jeho ochotu a vzorné vedení po celou dobu vypracovávání této práce.

Jan Šmejkal

Abstrakt

Tato práce se zabývá problémem klasifikace textu pomocí neuronových sítí. Jsou v ní vysvětleny principy klasifikace a způsob fungování neuronových sítí. V práci je porovnána schopnost multi-class i multi-label klasifikace anglického respektive českého textu za použití konvenčních klasifikačních metod a dopředných neuronových sítí.

Klíčová slova

Neuronová síť, multi-label klasifikace, klasifikace textu, python

Abstract

This research paper is dealing with the issue of text classification using neural networks. The principles of classification and the way neural networks work are explained in it. Moreover, the abilities of multi-class and multi-label to classify English, respectively Czech, text using conventional classification methods and feedforward neural networks are compared in the paper.

Key words

Neural network, multi-label classification, text classification, python

Obsah

Úvod **8**

1 Klasifikace **9**

1.1 Obecně 9

1.2 Klasifikace textu 9

1.2.1 Typy klasifikace 10

1.2.2 Předzpracování dokumentu 10

1.3 Typy trénování klasifikátorů 12

1.3.1 Učení s učitelem 12

1.3.2 Učení bez učitele 13

1.3.3 Kombinovaná metoda 13

1.4 Vybrané typy klasifikátorů 14

1.4.1 Support Vector Machines 14

1.4.2 Bayesovská klasifikace 15

2 Neuronové sítě **18**

2.1 Definice 18

2.2 Historie 18

2.3 Struktura 19

2.3.1 Zpětnovazební neuronové sítě 19
2.3.2 Dopředné neuronové sítě 20
2.3.3 Vrstvy 21
2.3.4 Perceptron 21
2.4 Trénování neuronové sítě 25
2.4.1 Trénování jednovrstvé lineární sítě 25
2.4.2 Algoritmus zpětné propagace 27
2.5 Implementace v jazyce Python 28
2.5.1 Lasagne 29

3 Praktická část 32

3.1 Ohodnocení výkonu sítě 32
3.1.1 F1 skóre 32
3.1.2 Průměrné F1 skóre 33
3.1.3 Skóre přesnosti 33
3.2 Multi-class klasifikace 33
3.2.1 Trénovací data 33
3.2.2 Analýza trénovacích dat 33
3.2.3 Předzpracování dat 34
3.2.4 Struktura neuronové sítě 35
3.2.5 Trénování sítě 36
3.2.6 Porovnání s baseline klasifikátorem 36
3.3 Multi-label klasifikace 38
3.3.1 Trénovací data 38

3.3.2 Analýza trénovacích dat 38

3.3.3 Předzpracování dat 40

3.3.4 Struktura neuronové sítě 40

3.3.5 Trénování sítě 41

3.3.6 Výsledky trénování 41

3.3.7 Porovnání s baseline klasifikátorem 44

Závěr **45**

Seznam použitých zdrojů **46**

Úvod

V dnešní době informací se stala technika klasifikace textu jednou z klíčových metod při organizaci velkého množství textových dat (článků). Techniky klasifikace textu pomáhají s vybíráním doporučených článků či doporučených zpráv pro daného uživatele. Velké uplatnění nalezly také jako deketory spamu[1]. Abychom měli představu, jak velké množství dat existuje, můžeme si uvést výrok Erica Schmidta, výkonného předsedy Alphabet Inc.[2], který v roce 2010 prohlásil, že lidstvo každé dva dny vyprodukuje tolik informací, kolik vyprodukovalo dohromady do roku 2003 [20]. Lze předpokládat, že v dnešní době, o šest let později, je toto množství neporovnatelně větší.

Problémem textových klasifikátorů nezaložených na neuronových sítích ale může být jejich fixovanost na klíčová slova. Proto se v této práci zabývám možností využití textových klasifikátorů implementovaných pomocí tzv. neuronových sítí. Jejich historie se začala psát, podobně jako historie elektronických programovatelných počítačů, na začátku 40. let minulého století [8]. Jedná se o způsob vytváření počítačového programu, který vychází z naší velmi zjednodušené představě o funkci lidského mozku.

Tato práce je rozdělena celkem na tři hlavní kapitoly. První kapitola se zabývá problematikou klasifikace a klasifikace textu, druhá je zaměřena na teoretický základ a popis neuronových sítí a ve třetí části je pojednáno o reálném použití neuronových sítí na textových klasifikačních úlohách.

[1] Nežádoucí zpráva zpravidla reklamního rázu

[2] Společnost vlastnící Google Inc.

Kapitola 1

Klasifikace

1.1 Obecně

Samotné slovo klasifikace pochází z latinského *classis*. Je možné jej přeložit do češtiny jako „třídění“ neboli rozdělování do různých tříd, skupin. Typickým příkladem klasifikace může být školní známkování nebo algoritmus rozhodující, zda příchozí email je, či není, spam.

Z pohledu strojového učení lze říci, že klasifikátory se trénují metodou učení s učitelem[1]. Učení bez učitele[2] je pak označováno jako klastrování a data se v něm rozdělují do skupin dle podobnosti jejich vlastností [1] (viz kapitola 1.3 na straně 12).

1.2 Klasifikace textu

U klasických klasifikačních úloh, například kategorizace typu vozidla, které projede kolem kamery, lze jednoduše stanovit vlastnosti (běžněji označované jako příznaky), dle kterých se budou vozidla klasifikovat. U výše uvedeného příkladu to může být mimo jiné rychlost vozidla, jeho délka, jeho barva, popřípadě i hlasitost, máme-li k

[1]Je k dispozici trénovací množina dat

[2]Předem nejsou k dispozici žádná data

dispozici zvukový záznam. Stanovení příznaků u textových souborů je však výrazně složitější.

Jako nejpřímočařejší řešení se jeví použití všech slov obsažených v textovém souboru, dokumentu. Obecně však může mít takový dokument různou velikost, tedy různé množství příznaků. Také by to znamenalo, že čím delší dokument, tím časově náročnější jeho zpracování. Samotné trénování klasifikátoru by se pak mohlo prodloužit řádově o hodiny s nejistým výsledkem. Z těchto důvodů se před začátkem trénování provádí předzpracování dokumentů, které má za úkol redukovat jak množství příznaků, tak nepodstatné informace [6].

1.2.1 Typy klasifikace

Nejen text lze obecně klasifikovat třemi způsoby dle počtu tříd do kterých jsou vzorky klasifikovány:

- **Binární klasifikace**
  Klasifikátor může text zařadit do jedné ze dvou říd. Například spamový filtr (*je* nebo *není* spam) či klasifikátor pohlaví (osoba je *muž* nebo *žena*).
- **Multi-class klasifikace**
  Neboli klasifikace do více tříd. Například kategorizace kolemjedoucího vozidla (jedná se *motocykl*, *osobní automobil*, *nákladní vozidlo* či *kamion*).
- **Multi-label klasifikace**
  Mnohdy je možné klasifikovaný objekt zařadit do více tříd. Zejména novinové články mohou být zpravidla označkovány několika „štítky“. Například novinový článek o vlakové nehodě ve Spojených státech amerických může zároveň patřit do kategorií *neštěstí a nehoda*, *železnice* a *USA*.

1.2.2 Předzpracování dokumentu

Textové dokumenty lze předzpracovávat mnoha různými způsoby. Tato podkapitola je však zaměřena pouze na metody využité v této práci.

Tokenizace

Tokenizace rozdělí vstupní text na tokeny, zpravidla samostatná slova. Rozdělování se většinou řídí mezerami mezi slovy popřípadě interpunkčními znaménky a z výpočetního hlediska je realizováno pomocí regulárních výrazů. Jsou-li předem známa slova, která nebudou mít na následnou klasifikaci vliv (například spojky a předložky), mohou být v této fázi předzpracování zcela vypuštěny. Jednotlivým tokenům jsou pak přiřazeny číselné ID[3] hodnoty.

Počítání

V této fázi se jednoduše spočítá, kolik tokenů se stejným ID označením se v textu vyskytuje.

Tf-idf transformace

Název pochází ze dvou anglických spojení; Tf jako *Term Frequency* a idf jako *Inverse document frequency.*

- **Term Frequency** (četnost výrazu)
  Vyjadřuje, jak často se token vyskytuje v rámci jednoho dokumentu. Tato hodnota se obvykle dělí celkovým počtem slov v daném textovém souboru, aby se nemohlo stát, že by se delší dokumenty s vyšším výskytem hledaného tokenu označovaly jako relevantnější než kratší dokumenty s menším výskytem. Nechť $n_{i,j}$ je množství výskytů tokenu i v dokumentu j. Četnost výrazu je pak definována jako
  $$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{1.1}$$
- **Inverse document frequency** (převrácená četnost ve všech dokumentech)
  Určuje *užitečnost* tokenu. V čím více dokumentech se token vyskytuje, tím méně důležitý a užitečný je pro následnou klasifikaci (například spojka *a* či zvratné zájmeno *se* se vyskytují v naprosté většině česky psaných textů a

[3] Unikátní označení identity

jejich přítomnost nevypovídá nic o tématu dokumentu). Nechť D_{celkem} je celkový počet dokumentů a D_i je množství dokumentů z množiny D_{celkem}, které alespoň jednou obsahují token i. Převrácená četnost je pak definována jako

$$idf_i = \log \frac{D_{celkem}}{D_i} \tag{1.2}$$

Pro každý token i v každém dokumentu j se spočítá celková hodnota Tf-idf, která je definována jako součin složek $tf_{i,j}$ z definice (1.1) a idf_i z definice (1.2) [24], tedy

$$tfidf_{i,j} = tf_{i,j} \cdot idf_i = \frac{n_{i,j}}{\sum_k n_{k,j}} \cdot \log \frac{D_{celkem}}{D_i} \tag{1.3}$$

Výsledek předzpracování

Díky výše zmíněným metodám dostane vstupní soubor textů tvar velmi řídké matice[4]. Ta je, z výpočetního hlediska, velmi úsporná na paměť, neboť je nutné uchovávat pouze informace o nenulových prvcích – dokáže tak pojmout velké množství zpracovaného textu. Z pohledu klasifikace je výhodou tohoto tvaru zejména konstantní počet prvků pro každý text ve zpracovávaném datasetu. Nevýhodou naopak může být velký počet nul, který by mohl zmenšit schopnost konvergence trénovacího procesu.

1.3 Typy trénování klasifikátorů

Obecně lze trénování všech klasifikátorů, bez ohledu na jejich typ, rozdělit na tři skupiny [6].

1.3.1 Učení s učitelem

Existují data, která mají již označenou třídu, do které patří. Tato data jsou přivedena na vstup klasifikátoru, který následně určí pravděpodobnosti náležitostí do jednotlivých tříd. Dle srovnání predikovaného výstupu a reálné náležitosti se pak upraví parametry klasifikátoru.

[4]Matice, která má řádově větší množství nulových prvků než prvků s nějakou hodnotou

Zpravidla se všechna označená data před samotným trénováním ještě rozdělí na tři skupiny, aby se mohlo předejít tzv. přetrénování, které nastává, když se klasifikátor natrénuje až moc přesně na trénovacích datech a není pak schopen správné klasifikace dat s nimiž se při trénování nesetkal.

1. **Trénovací data**
   Data, pomocí kterých se přímo upravují parametry klasifikátoru

2. **Validační data**
   Data, pomocí kterých se při trénování hlídá přetrénování. Začne-li výrazně klesat chyba na trénovacích datech avšak chyba na validačních datech roste, nastalo přetrénování.

3. **Testovací data**
   Data, pomocí kterých se na natrénovaném klasifikátoru určuje jeho úspěšnost. S touto množinou dat by se klasifikátor ve fázi učení nikdy neměl setkat.

1.3.2 Učení bez učitele

Data nemají k dispozici označení třídy, do které patří. Klasifikátor tak u dat hledá různé podobnosti, pomocí kterých dokáže data sám roztřídit do skupin. Tato metoda se u klasifikace textu většinou nepoužívá, neboť obecně dosahuje horších výsledků a vyžaduje zásahy uživatele (pojmenování nalezených tříd).

1.3.3 Kombinovaná metoda

Je k dispozici množina dat s označenou třídou, do které patří a množina dat bez označené třídy (zpravidla řádově větší). Nejdříve proběhne učení s učitelem na označených datech a následně se klasifikátor pokusí zařadit data neoznačená. Z těch se vyberou ty, které mají největší pravděpodobnosti náležitostí do tříd a přidají se k označeným datům. Tento postup se opakuje, dokud není množina neoznačených dat minimální.

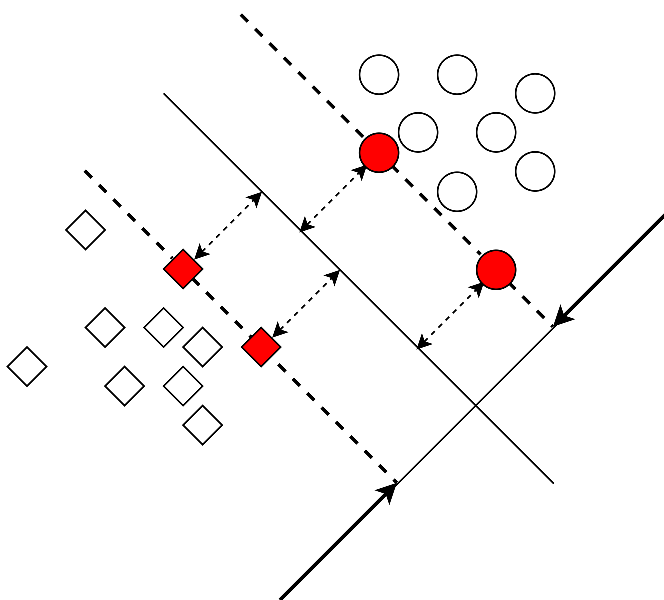
1.4 Vybrané typy klasifikátorů

Klasifikátorů existuje velké množství, proto jsou v této podkapitole uvedeny pouze ty, jež byly během práce použity.

1.4.1 Support Vector Machines

Support Vector Machines (dále SVM), česky metoda podpůrných vektorů, je jedna z nejúspěšnějších klasifikačních metod [23].

Pro vysvětlení budeme uvažovat jednoduchý binární klasifikační problém. Mějme množinu trénovacích vektorů $\{x_1, x_2, \ldots, x_n\}$ a k ní množinu požadovaneného označení $\{y_1, y_2, \ldots, y_n\}$, kde (jelikož jde o binární problém) $y_i \in \{-1, 1\}$. Nejjednodušeji lze SVM popsat jako nadrovinu rozdělující trénovací data s maximálním okrajem. Všechny vektory ležící na jedné straně nadroviny jsou označeny jako 1 a všechny vektory na druhé straně jako -1. Vektory ležící nejblíže k nadrovině jsou označovány jako podpůrné vektory (na obrázku 1.1 červeně vyznačeny) – odtud pochází název metody [6, 23].



Obrázek 1.1: Ukázka rozdělení binárního prostoru dle SVM

Označme $w = [w_1, w_2, \ldots, w_m]$ a b jako parametry klasifikátoru, kde m je dimenze

vektorů x_i. Rovnici binárního SVM pak definujeme jako

$$w \cdot x - b = 0. \tag{1.4}$$

Klasifikaci následně provádíme porovnáním s nadrovinou dané třídy. Tedy pro náš ukázkový příklad to bude

$$w \cdot x - b \geq 1 \tag{1.5}$$

respektive

$$w \cdot x - b \leq 1. \tag{1.6}$$

Při splnění nerovnice (1.5) bude tedy bod patřit do třídy 1 a při splnění nerovnice (1.6) do třídy -1. Při trénování tak řešíme optimalizační problém

$$(w, b) = \arg\max_{w,b} \frac{1}{2}||w||^2. \tag{1.7}$$

Z výpočetního hlediska je ale mnohem jednodušší počítat parametr w přes tzv. duální případ [6, 23]. Ten vychází z rovnice

$$\overrightarrow{\alpha} = \arg\max_{\overrightarrow{\alpha}} \left(\sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j (x_i, x_j) \right), \tag{1.8}$$

jejímž vyřešením za podmínek

$$\alpha_i \geq 0, \quad i = 1, 2, \ldots, l \quad \text{a} \quad \sum_{i=1}^{l} a_i \cdot y_i = 0 \tag{1.9}$$

získáme parametry α_i. Následně přejdeme zpět k přímému řešení a dle vzorce

$$w = \sum_{i=1}^{l} \alpha_i y_i x_i \tag{1.10}$$

již jednoduše dopočítáme hodnotu w [6].

1.4.2 Bayesovská klasifikace

Tento klasifikátor vychází z Bayesovy věty o podmíněných pravděpodobnostech.

Teoretický základ

Bayesův vztah pro výpočet pravděpodobnosti platnosti hypotézy H, pokud nastala událost E je

$$\mathrm{P(H|E)} = \frac{\mathrm{P(E|H) \cdot P(H)}}{\mathrm{P(E)}}, \tag{1.11}$$

přičemž pravděpodobnost $P(H)$ se označuje jako *apriorní* a odpovídá informaci o zastoupení jednotlivých hypotéz. Podmíněná pravděpodobnost $P(H|E)$, označována jako *aposteriorní*, vyjadřuje, jak již bylo řečeno, pravděpodobnost hypotézy H, nastala-li událost (popřípadě je-li pravda skutečnost) E.

Při klasifikaci pak bude zkoumaných hypotéz takové množství, do kolika tříd klasifikujeme, přičemž budeme hledat největší aposteriorní pravděpodobnost. Předpokládejme, že pro každou hypotézu H_t, $t = 1, 2, \ldots, n$ (kde n je počet tříd) můžeme určit její podmíněnou pravděpodobnost. Hledáme tedy

$$\mathrm{P(H}_J\mathrm{|E)} = \max_t \frac{\mathrm{P(E|H}_t\mathrm{) \cdot P(H}_t\mathrm{)}}{\mathrm{P(E)}}. \tag{1.12}$$

Vzhledem k tomu, že nás pro klasifikaci nezajímají konkrétní hodnoty, pouze zjišťujeme, které jsou největší, tak můžeme ve vztahu zcela zanedbat jmenovatele $P(E)$ a upravit vztah na

$$\mathrm{P(E|H}_J\mathrm{) \cdot P(H}_J\mathrm{)} = \max_t \left(\mathrm{P(E|H}_t\mathrm{) \cdot P(H}_t\mathrm{)}\right). \tag{1.13}$$

Můžeme také předpokládat, že na $P(H)$ nezáleží (všechny hypotézy jsou stejně pravděpodobné) a celý vztah ještě zjednodušit na

$$\mathrm{P(E|H}_J\mathrm{)} = \max_t \mathrm{P(E|H}_t\mathrm{)}, \tag{1.14}$$

čímž dostaneme oproti výrazu (1.12) výraz výpočetně velmi nenáročný [3, 17].

Naivní bayesovský klasifikátor

Tento klasifikátor předpokládá, že jednotlivá pozorování $E_1, E_2, \ldots, E_k$ jsou při platnosti hypotézy H podmíněně nezávislá. Protože je tento požadavek v reálných úlohách splněn jen zřídka, dostal přídomek *naivní*. Díky tomuto předpokladu však

můžeme vztah

$$\mathrm{P}(\mathrm{H}|\mathrm{E}_1,\ldots,\mathrm{E}_k) = \frac{\mathrm{P}(\mathrm{E}_1,\ldots,\mathrm{E}_k|\mathrm{H})\cdot\mathrm{P}(\mathrm{H})}{\mathrm{P}(\mathrm{E}_1,\ldots,\mathrm{E}_k)} \tag{1.15}$$

počítat jako

$$\mathrm{P}(\mathrm{H}|\mathrm{E}_1,\ldots,\mathrm{E}_k) = \frac{\mathrm{P}(\mathrm{H})}{\mathrm{P}(\mathrm{E}_1,\ldots,\mathrm{E}_k)}\cdot\prod_{k=1}^{K}\mathrm{P}(\mathrm{E}_k|\mathrm{H}). \tag{1.16}$$

Jak již víme, tak tento vztah můžeme zjednodušit vypuštěním jmenovatele a budeme hledat pro kterou hypotézu H_t (třídu) bude mít tento vztah nejvyšší hodnotu, což bude námi hledaná třída H_{MAP}:

$$\mathrm{H}_{MAP} = \mathrm{H}_J \Leftrightarrow \mathrm{P}(\mathrm{H}_J)\cdot\prod_{k=1}^{K}\mathrm{P}(\mathrm{E}_k|\mathrm{H}_J) = \max_t\left(\mathrm{P}(\mathrm{H}_t)\cdot\prod_{k=1}^{K}\mathrm{P}(\mathrm{E}_k|\mathrm{H}_t)\right). \tag{1.17}$$

Fáze učení je tedy u toho klasifikátoru nahrazena počítáním pravděpodobností $P(H_t)$ a $P(E_k|H_t)$ na trénovací sadě [3, 11, 17].

Kapitola 2

Neuronové sítě

2.1 Definice

Neuronová síť je struktura, která paralelně zpracovává distribuované informace. Skládá se z elementů (které mohou mít lokální paměť), které jsou mezi sebou propojené. Každý element má jeden výstup, který se větví do libovolného počtu dalších spojení (přičemž každé toto spojení stále nese stejnou informaci – výstupní signál elementu). Výstupní signál elementu může být libovolného matematického typu. Veškeré výpočty, které probíhají v každém elementu, musejí být kompletně lokální; to znamená, že musí záležet pouze na současných hodnotách vstupních signálů příchozích spojení a na hodnotách uložených v lokální paměti elementu [5].

2.2 Historie

Neuronová síť je způsob psaní počítačového programu, algoritmu, kterému dali za vznik neurofyzik Warren McCulloch a matematik Walter Pitts již v roce 1943. Ti definovali základní strukturu neuronových sítí, konkrétně jednotlivé výpočetní jednotky – neurony a jejich propojení mezi sebou – synapse. Také stanovili, že každý neuron má svojí prahovou hodnotu. Pokud je součet všech vstupů daného neuronu větší než jeho prahová hodnota, tak vyšle signál do všech dalších neuronů, které jsou

k tomuto neuronu připojeny [10].

Neuron podle Warrena a Pittse však byl schopen pouze binárních operací a nebyl schopen se učit. Toho si byl vědom americký psycholog Frank Rosenblatt, který v roce 1958 přišel s pojmem „perceptron“. Ten nahradil nedokonalý neuron a v mnohém jej převyšoval. Například váhové a prahové hodnoty v něm byly definovány jako parametry modelu, které se mohly nastavit procesem učení, který Rosenblat také definoval (viz kapitola 2.3.4 na straně 21) [18].

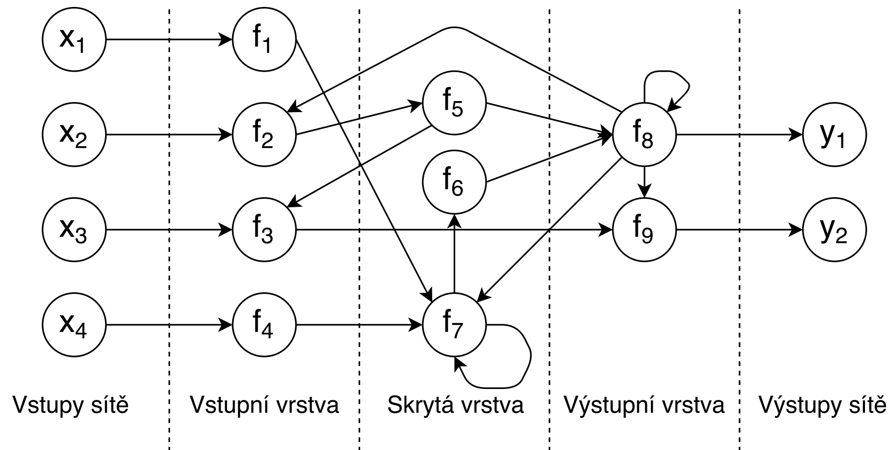
Další důležitý milník vývoje neuronových sítí nastal v 80. letech minulého století s příchodem algoritmu zpětné propagace. Ten umožňuje relativně rychlé trénování i vícevrstvých sítí. Funguje na principu propagace gradientu chyby z výstupu sítě (tzn. rozdíl mezi skutečným a požadovaným výstupem), viz kapitola 2.4.2 na straně 27.

2.3 Struktura

Neuronové sítě můžeme rozdělit na zpětnovazební a dopředné. Každou síť pak dělíme na její jednotlivé vrstvy, které dále můžeme rozdělit až na jednotlivé perceptrony.

2.3.1 Zpětnovazební neuronové sítě

Ve zpětnovazebních sítích je signálu umožněno šířit se všemi směry. Takovéto sítě jsou zpravidla velmi mocnými nástroji, ale zároveň mohou být velmi komplikované. Jsou dynamické; jejich vnitřní „stav“ se po přivedení vstupu mění do té doby, dokud síť nedosáhne rovnovážného bodu [21].

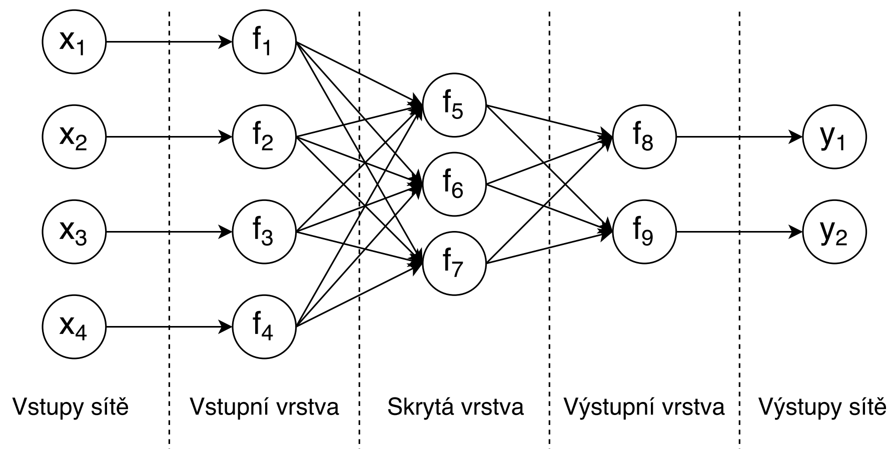


Obrázek 2.1: Příklad zpětnovazební sítě

Jelikož nebyl tento typ sítě v této práci použit, nebude mu nadále věnována žádná pozornost.

2.3.2 Dopředné neuronové sítě

V dopředných neuronových sítích je signálu umožněno šířit se pouze jedním směrem, konkrétně od vstupu k výstupu. Neexistují zde žádné smyčky (zpětné vazby), to znamená, že výstup z vrstvy j může vždy ovlivnit pouze vrstvu $j+1$. Právě takovéto vrstvy mají uplatnění v úlohách detekce, rozpoznávání a klasifikace [21, 10].

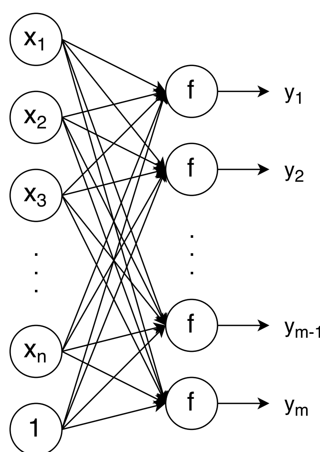


Obrázek 2.2: Příklad dopředné sítě

2.3.3 Vrstvy

Neuronové sítě používané v dnešní době se skládají z vrstev, které můžeme obecně rozdělit na tři základní skupiny:

1. Vstupní vrstva – vrstva, jež přijímá vstupní data
2. Skrytá vrstva – jedna nebo více vrstev, na které zpravidla nelze přistupovat přímo
3. Výstupní vrstva – vrstva, na jejímž výstupu je výstup celé neuronové sítě



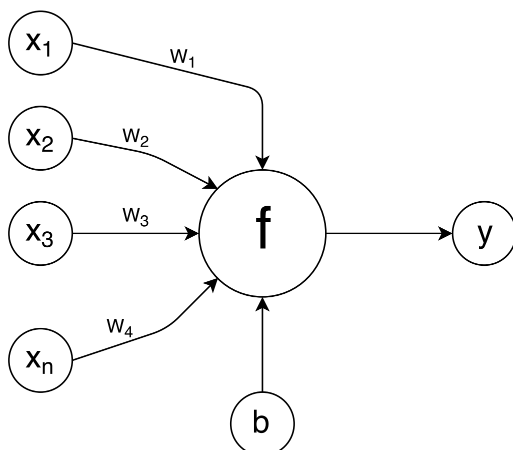
Obrázek 2.3: Jedna vrstva sítě

Každá vrstva se dále skládá z alespoň jednoho perceptronu.

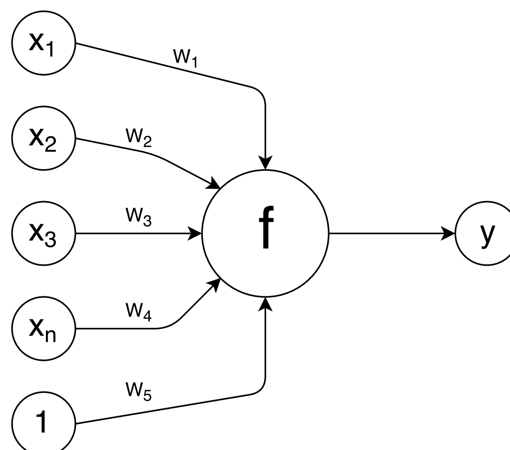
2.3.4 Perceptron

Perceptron je základní stavební jednotka neuronových sítí. Je definován váhami, prahem a aktivační funkcí. Váhy definujeme jako vektor hodnot, kterým přenásobíme vektor vstupních hodnot perceptronu pro získání celkového vstupu. Práh je hodnota, kterou přičítáme k celkovému vstupu neuronu. Z výpočetního hlediska je výhodné nemít tuto hodnotu zvlášť, ale jako součást váhového vektoru a ke vstupnímu vektoru připojit hodnotu 1. Aktivační funkce je funkce, na jejímž vstupu je celkový

vstup perceptronu (tzn. váhový vektor vynásobený vstupním vektorem, popř. váhový vektor vynásobený vstupním vektorem s přičtenou prahovou hodnotou) a na výstupu pak výstup samotné funkce.



Obrázek 2.4: Perceptron s rozděleným váhovým vektorem a prahem



Obrázek 2.5: Perceptron se sloučeným váhovým vektorem a prahem

Celkový výstup jednoho perceptronu y můžeme definovat jako

$$y = f\left(\sum_{i=1}^{n}(x_i \cdot w_i) + b\right) = f(w^T \cdot x + b), \tag{2.1}$$

kde n je celkový počet vstupů, $w = [w_1, w_2, w_3, \ldots, w_n]^T$ je váhový vektor, $x = [x_1, x_2, x_3, \ldots, x_n]^T$ je vstupní vektor, b je prahová hodnota a $f()$ je aktivační funkce.

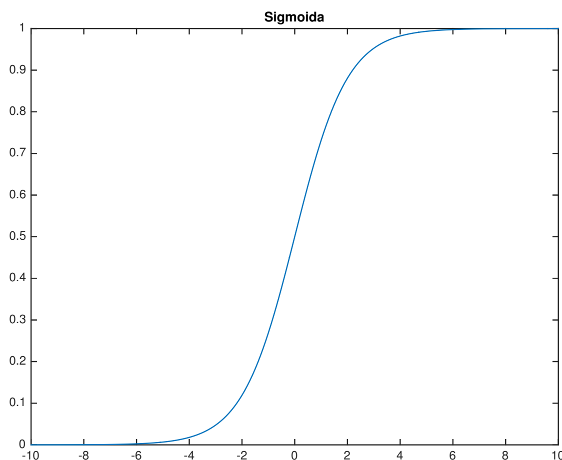
Aktivační funkce

Aktivační funkcí může být jakákoliv matematická funkce u které platí, že libovolnému x přiřadí právě jednu hodnotu y. Mezi nejčastěji používané aktivační funkce patří:

- **Sigmoida (sigmoidální logistická funkce)**
  Sigmoida je funkce s předpisem

$$f(x) = \frac{1}{1 + e^{-x}}, \tag{2.2}$$

která transformuje reálné číslo x do intervalu $(0, 1)$. Toto omezování vstupních dat lze považovat za jednu z nejvýhodnějších vlastností sigmoidy (a hyperbolického tangens), neboť se nemůže stát, že by se na výstupu objevovaly velmi rozdílné hodnoty. Také symetrické sigmoidy často konvergují rychleji než standartní logistické funkce [9].



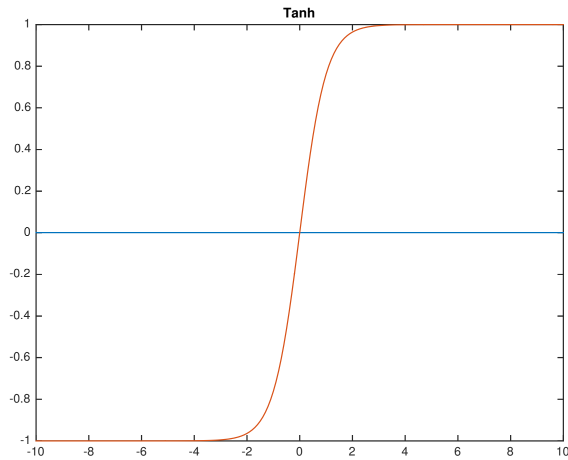
Obrázek 2.6: Sigmoida

- **Tanh (hyperbolický tangens)**

  Tanh je funkce s předpisem

$$f(x) = tanh(x) = \frac{2}{1 + e^{-2x}} - 1, \tag{2.3}$$

která transformuje reálné číslo x do intervalu $(-1, 1)$. Hyperbolický tangens má všechny dobré vlastnosti sigmoidy a navíc k nim přidává skutečnost, že jeho výstup má průměrnou hodnotu 0, což je ještě vhodnější z hlediska normalizace než sigmoida. Další výhodou je fakt, že lze tuto funkci celkem jednoduše a přesně aproximovat a tím značně urychlit výpočet [9].



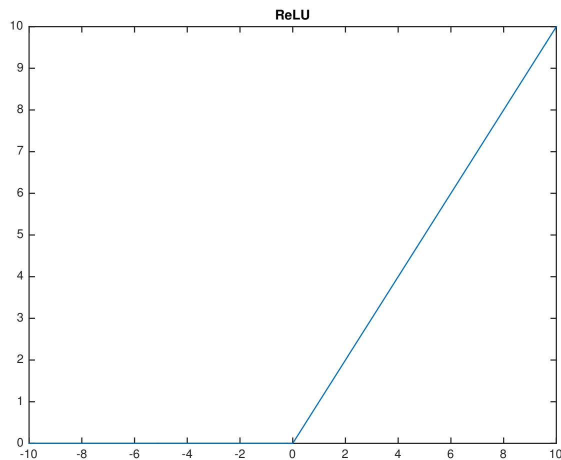
Obrázek 2.7: Tanh

- **ReLU (Rectified Linear Unit)**

  ReLU je funkce s předpisem

$$f(x) = max(0, x), \tag{2.4}$$

  která transformuje negativní hodnoty x na hodnotu 0. Mezi její výhody patří velmi malá výpočetní náročnost a jednoduchá derivace.



Obrázek 2.8: ReLU

- **Softmax (normalizovaná exponenciála)**

  Softmax je funkce využívaná zejména u klasifikačních úloh, neboť dokáže vstupní

vektor x dimenze N transformovat na vektor $f(x)$ dimenze N, který má hodnoty omezené na intervalu (0,1) a jejich součet je 1. Předpis této funkce je

$$f(x)_i = \frac{e^{x_i}}{\sum_{n=1}^{N} e^{x_n}} \text{ pro } i = 1, \dots, N \tag{2.5}$$

2.4 Trénování neuronové sítě

Obecně lze způsob trénování (obdobně jako u klasifikátorů) rozdělit na tři základní způsoby [19]:

- **Učení s učitelem**
  Neuronové síti jsou prezentovány postupně dvojice vstupů a známých požadovaných výstupů. Pokud je výstup sítě jiný než požadovaný výstup, jsou patřičně upraveny parametry sítě.
- **Učení bez učitele**
  Neuronové síti jsou postupně předkládány vstupní data bez požadovaných výstupů – síť je donucena si sama hledat relevantní příznaky, což vede ke shlukování vstupních dat.
- **Zpětnovazební učení**
  Tato učící metoda je založena, jak její název napovídá, na zpětné vazbě. Síť na vstupy reaguje nějakým výstupem, načež je tento její výstup ohodnocen numerickou hodnotou (dle požadovaného chování) a podle této hodnoty se patřičně upraví parametry sítě.

2.4.1 Trénování jednovrstvé lineární sítě

Definice

Jednovrstvá lineární síť je tvořena jednou vrstvou neuronů s lineární aktivační funkcí. Jak již víme z definice (2.1), výstup jednoho perceptronu definujeme jako $y = f(w^T \cdot x + b)$. Budeme-li uvažovat více perceptronů, musíme v definici zavést indexy. Také

zavedeme namísto váhového vektoru w pro jeden perceptron váhovou matici W pro celou vrstvu, kterou definujeme jako

$$W = \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{12} & \dots & w_{2n} \\ \dots & \dots & \dots & \dots \\ w_{m1} & w_{m2} & \dots & w_{mn} \end{pmatrix}. \tag{2.6}$$

Uvažujme vstupní vektor $x = [x_1, x_2, \dots, x_n]^T$, prahový vektor $b = [b_1, b_2, \dots, b_m]^T$ a výstupní vektor $y = [y_1, y_2, \dots, y_m]^T$ a definujme výstup pro j-tý perceptron jako

$$y_j = f\left(\sum_{i=1}^{n} W_{ji} \cdot x_i + b_j\right). \tag{2.7}$$

Jelikož je aktivační funkce lineární, můžeme zavést strmost aktivační funkce λ a celý vztah zjednodušit na

$$y_j = \lambda\left(\sum_{i=1}^{n} W_{ji} \cdot x_i + b_j\right). \tag{2.8}$$

Nahrazením indexovaných proměnných celými vektory a maticí dostaneme vztah pro výpočet výstupu celé vrstvy

$$y = \lambda(W \cdot x + b). \tag{2.9}$$

Trénování

Nechť máme k dispozici trénovací množinu, tj. dvojice $[x_p, u_p]$, kde x_p je vstup p-té dvojice a u_p je požadovaný výstup p-té dvojice. U takovýchto dvojic chceme minimalizovat výstupní chybu sítě, kterou definujeme jako

$$E = \frac{1}{2}\sum_{p=1}^{P} \left\| u_p - y_p \right\|^2. \tag{2.10}$$

Z tohoto vztahu definujeme chybu jedné dvojice jako

$$\varepsilon = \frac{\left\| u_p - y_p \right\|^2}{2}. \tag{2.11}$$

Nechť k označuje krok algoritmu a c tzv. učící konstantu (také označovanou jako rychlost učení). Dosadíme-li vztah (2.8) do vztahu (2.10) a upravíme, dostaneme vztah pro změnu vah v závisloti na chybě výstupu

$$W(k+1) = W(k) - c \cdot \frac{\partial \varepsilon}{\partial W}. \tag{2.12}$$

Analogicky určíme i vztah pro změnu prahů

$$b(k+1) = b(k) - c \cdot \frac{\partial \varepsilon}{\partial b}. \tag{2.13}$$

U těchto vztahů je důležité při samotném trénování dbát na správnou volbu konstanty učení; příliš malá hodnota učení zpomaluje, u příliš velké hodnoty hrozí nestabilita procesu učení [14].

2.4.2 Algoritmus zpětné propagace

Algoritmus zpětné propagace přímo vychází z algoritmu pro jednovrstvou lineární síť. Uvažujme třívrstvou dopřednou síť s I vstupy, J perceptrony ve skryté vrstvě a M výstupy (perceptrony ve výstupní vrstvě). Aktivační funkce skryté vrstvy označme jako $f()$ a její výstup $z = [z_1, z_2, \ldots, z_J]^T$. Aktivační funkce výstupní vrstvy označme jako $g()$ a její výstup $y = [y_1, y_2, \ldots, y_M]^T$. Váhovou matici a prahový vektor skryté vrstvy označíme jako W (typu JxI) respektive b (dimenze J). Váhovou matici a prahový vektor výstupní vrstvy označíme jako V (typu MxJ) respektive d (dimenze M).

Ze vztahu (2.7) odvodíme výstup j-tého perceptronu skryté vrstvy

$$z_j = f\left(\sum_{i=1}^{I} W_{ji} \cdot x_i + b_j\right) \tag{2.14}$$

a výstup m-tého perceptronu výstupní vrstvy

$$y_m = g\left(\sum_{j=1}^{J} W_{mj} \cdot z_j + d_m\right) = g\left(\sum_{j=1}^{J} W_{mj} \cdot f\left(\sum_{i=1}^{I} W_{ji} \cdot x_i + b_j\right) + d_m\right) \tag{2.15}$$

Stejně jako u trénování jednovrstvé lineární sítě jednoduše podle funkce chyby sítě (2.10) určíme změnu vah jako

$$\triangle W = -c \cdot \frac{\partial \varepsilon}{\partial W}. \tag{2.16}$$

U všech vrstev sítě platí, že se její parametry (váhy a prahy) mění v závislosti na chybě, kterou určíme jako rozdíl mezi skutečným a požadovaným výstupem. Požadovaný výstup (a analogicky chybu výstupu ve vzorci (2.11)) máme však apriory určený pouze u výstupní sítě. U všech ostatních vrstev chybu dostaneme její zpětnou propagací na předchozí vrstvy, odtud pochází název algoritmu.

Výpočet chyby

Nejdříve odvozením ze vzorců (2.10) a (2.16) určíme, jak se změní chyba v závislosti na změně hodnoty konkrétní váhy ve výstupní vrstvě:

$$\frac{\partial E}{\partial W_{jm}} = z_j \cdot \delta_m, \tag{2.17}$$

kde

$$\delta_m = y_m(1 - y_m)(y_m - u_m) \Rightarrow \delta_m = g'(V_{mJ} \cdot z)(y_m - u_m) \tag{2.18}$$

Nyní analogicky pro skrytou vrstvu:

$$\frac{\partial E}{\partial W_{ij}} = x_i \cdot \delta_i \tag{2.19}$$

$$\delta_j = y_j(1 - y_j) \sum_{m=1}^{M} \delta_m \cdot W_{jm} \Rightarrow \delta_j = f'(W_{jI} \cdot x) \sum_{m=1}^{M} \delta_m \cdot W_{jm}. \tag{2.20}$$

Následně dosadíme do (2.16) a získáme vztah pro změnu vah ve výstupní vrstvě

$$\triangle W_{jm} = -c \cdot \delta_m \cdot z_j \tag{2.21}$$

a analogicky ve skryté vrstvě

$$\triangle W_{ij} = -c \cdot \delta_j \cdot x_i \tag{2.22}$$

I zde je třeba dbát na volbu konstanty učení, neboť její nevhodná hodnota může radikálně zpomalit nebo naopak znestabilnit proces učení [21, 15].

2.5 Implementace v jazyce Python

Dopředné neuronové sítě lze implementovat v téměř jakémkoliv programovacím jazyce. Protože jsem ale tuto práci vypracovával v jazyce Python (konkrétně verzi 2.7.10), tak se budu věnovat implementaci právě v něm. Zároveň existuje spousta programovacích knihoven, které práci s neuronovými síti usnadňují. Já jsem zvolil knihovnu Lasagne.

2.5.1 Lasagne

Lasagne je knihovna pro práci s neuronovými sítěmi za pomocí knihovny Theano. Mezi její hlavní výhody patří transparentní podpora výpočtů na CPU[1] i GPU[2], modularita a relativní jednoduchost [4].

Ukázka definice jednoduché sítě se třemi vstupy, jednou skrytou vrstvou o deseti neuronech a třemi výstupy s funkcí softmax (viz vzorec (2.5)):

Kód 2.1: Ukázka implementace neuronové sítě pomocí Lasagne

```
import lasagne
import theano
velikost_vstupu = 3
tvar_vstupu = theano.tensor.vector("vstup")
tvar_vystupu = theano.tensor.vector("vystup")
sit = lasagne.layers.InputLayer(shape=(None, velikost_vstupu),
    input_var=tvar_vstupu)
sit = lasagne.layers.DenseLayer(sit, num_units=10)
sit = lasagne.layers.DenseLayer(sit, num_units=3,
    nonlinearity=lasagne.nonlinearities.softmax)
```

Trénovací proces

Pro trénování sítě musíme nejdříve provést dva kroky: získat výstup sítě a následně vytvořit funkci upravující její parametry v závislosti na chybě výstupu sítě.

Uvažujeme-li kód 2.1, bude získání výstupu sítě vypadat následovně

Kód 2.2: Získání výstupu neuronové sítě pomocí Lasagne

```
y = lasagne.layers.get_output(sit)
```

Chybu výstupu a samotnou trénovací funkci určíme jako

[1] Procesor počítače

[2] Grafická karta počítače

Kód 2.3: Chyba sítě a trénovací funkce pomocí Lasagne

```
# chyba vystupu
err = lasagne.objectives.categorical_crossentropy(y, tvar_vystupu)
err = theano.tensor.mean(err)
# ziskani parametru site
parametry = lasagne.layers.get_all_params(sit, trainable=True)
# upraveni parametru dle vystupni chyby
update = lasagne.updates.nesterov_momentum(err, parametry)
# vytvoreni samotne trenovaci funkce
trenovaci_funkce = theano.function(inputs=[tvar_vstupu, tvar_vystupu],
    outputs=err, updates=update)
```

Pomocí tohoto kódu získáme trénovací funkci celé sítě. Při jejím zavolání získáme nejen chybu výstupu sítě, ale zároveň funkce sama upraví paramatry sítě v závislosti na této chybě. Můžeme si povšimnout, že tato funkce pochází z knihovny Theano.

Funkce knihovny Theano (theano.function) jsou funkce pracující se symbolickými proměnnými, které samotná knihovna při spuštění zkompiluje do C++ jazyka (resp. CUDA kódu, dle nastavení a dostupných hardwarových prostředků) a při volání dané funkce pak volá již zkompilovaný kód, čímž se značně zvýší rychlost výpočtů [2].

Chyba sítě

Knihovna Lasagne poskytuje z konvenčních důvodů tři předpřipravené funkce pro získání chyby sítě. V následujících ukázkách implementovaných funkcí předpokládáme, že $y = [y_1, y_2, \ldots, y_n]$ je výstup sítě a $u = [u_1, u_2, \ldots, u_n]$ požadovaný výstup sítě:

- **Binární cross-entropie (binary_crossentropy)**
  Funkce vhodná zejména pro úlohy binární klasifikace a sítě s výstupní sigmoidální funkcí. Předpis je

$$err = -(u \cdot \log(y) + (1 - u) \cdot \log(1 - y)). \tag{2.23}$$

- **Kategorická cross-entropie (categorical_crossentropy)**

  Funkce vhodná zejména pro úlohy klasifikace do více tříd a sítě s výstupní funkcí softmax. Její předpis je

  $$err = -\sum_{x} y(x) \cdot \log(u(x)). \tag{2.24}$$

- **Kvadratická chyba (squared_error)**

  Funkce vhodná zejména pro úlohy regrese a sítě s výstupní lineární funkcí. Předpis je

  $$err = (y - u)^2. \tag{2.25}$$

Po získání chyb z těchto funkcí jsou následně jejich hodnoty průměrovány přes celá testovací data.

Kapitola 3

Praktická část

Při řešení této práce jsem řešil úlohu multi-class klasifikace anglického textu a úlohu multi-label klasifikace textu českého. V této kapitole tedy popíši oba zmíněné problémy a způsob ohodnocení práce klasifikátoru. K porovnání výkonnosti různých klasifikačních metod bude využito tzv. F1 skóre, průměrného F1 skóre a skóre přesnosti.

3.1 Ohodnocení výkonu sítě

3.1.1 F1 skóre

F1 skóre je způsob určování přesnosti klasifikátorů zejména u multi-label problémů. Definujme p jako podíl mezi počtem správných pozitivních výsledků a počtem všech pozitivních výsledků a r jako podíl mezi počtem správných pozitivních výsledků a počtem pozitivních výsledků jež měly být vráceny. F1 skóre se pak vypočítá jako

$$F_1 = 2 \cdot \frac{p \cdot r}{p + r}. \tag{3.1}$$

Dále existují dvě varianty tohoto vzorce, konkrétně obecné f_β skóre a tzv. G-míra [13].

3.1.2 Průměrné F1 skóre

Chceme-li určit výkon celého multi-label klasifikátoru a ne jen na konkrétních štítcích, použijeme průměrné F1 skóre. Nejedná se o nic jiného než průměr ze všech hodnot F1 skóre pro jednotlivé štítky.

3.1.3 Skóre přesnosti

Nejjednodušší identifikátor výkonnosti klasifikátoru využívaný zejména u multi-class problémů. Je definovaný jako podíl mezi počtem správných klasifikací a celkovým množstvím příkladů.

3.2 Multi-class klasifikace

3.2.1 Trénovací data

V této úloze je jako trénovací množina použit tzv. *20 Newsgroups* dataset. Jedná se o soubor necelých 20000 vzorků, které jsou (téměř) rovnoměrně rozdělené do 20 různých kategorií. Tento dataset je v klasifikačních úlohách velmi oblíben a nachází využití jak v experimentech, tak ve zkoumání nových možností klasifikace [16].

3.2.2 Analýza trénovacích dat

Jak již bylo uvedeno výše, celý dataset je rozdělen do 20 kategorií, jejichž jména jsou uvedena v tabulce 3.1. Každý prvek (jichž je celkem 18846) je pak text představující tělo emailu (existuje i verze datasetu, ve které prvky obsahují i emailové hlavičky[1]). Dále má každý prvek k sobě přiřazenou číselnou hodnotu reprezentující jeho příslušnost ke kategorii. Jelikož se jedná o dataset určený pro multi-class klasifikaci, náleží každý prvek právě do jedné kategorie.

[1] tzn. email odesílatele, příjemce, předmět emailové zprávy, atp.

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

Tabulka 3.1: Jména kategorií datasetu 20 Newsgroups

3.2.3 Předzpracování dat

Jelikož je prakticky nemožné natrénovat jakýkoliv klasifikátor jen z textových dat, je nutné nejdříve provést jejich předzpracování. Metody předzpracování textu použité v této práci jsou již popsané v kapitole 1.2.2 na straně 10. Tato podkapitola bude tedy věnována jen jejich implementaci v jazyce Python.

Nejdříve je nutné data načíst. O to se stará funkce *fetch_20newsgroups* z knihovny *scikit-learn*. Ta nejdříve zjistí, zda-li byl již datový archiv stažen. Pokud ano, načte jej, v opačném případě jej stáhne z internetu.

Kód 3.1: Načtení datasetu

```
from sklearn.datasets import fetch_20newsgroups
fetched = fetch_20newsgroups(subset="all", shuffle=True, random_state=42)
```

Dále data čeká samotné předzpracování. Nejdříve je provedena tokenizace s počítáním dat a následně Tf-idf transformace.

Kód 3.2: Předzpracování dat jako dva kroky

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
count_vectorizer = CountVectorizer()
train_counts = count_vectorizer.fit_transform(fetched.data)
tfidf_transformer = TfidfTransformer()
```

```
train_tfidf = tfidf_transformer.fit_transform(train_counts)
```

Knihovna *scikit-learn* také umožňuje tyto dva kroky sloučit do jednoho

Kód 3.3: Předzpracování dat jako jeden krok

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()
train_tfidf = tfidf_vectorizer.fit_transform(fetched.data)
```

Pro rychlejší výpočet a trénovací proces je vhodné trénovací data přesunout na grafickou kartu, což zajišťuje knihovna *theano*

Kód 3.4: Přesun dat na grafickou kartu

```
import theano
train_tfidf_shared = theano.shared(train_tfidf)
```

3.2.4 Struktura neuronové sítě

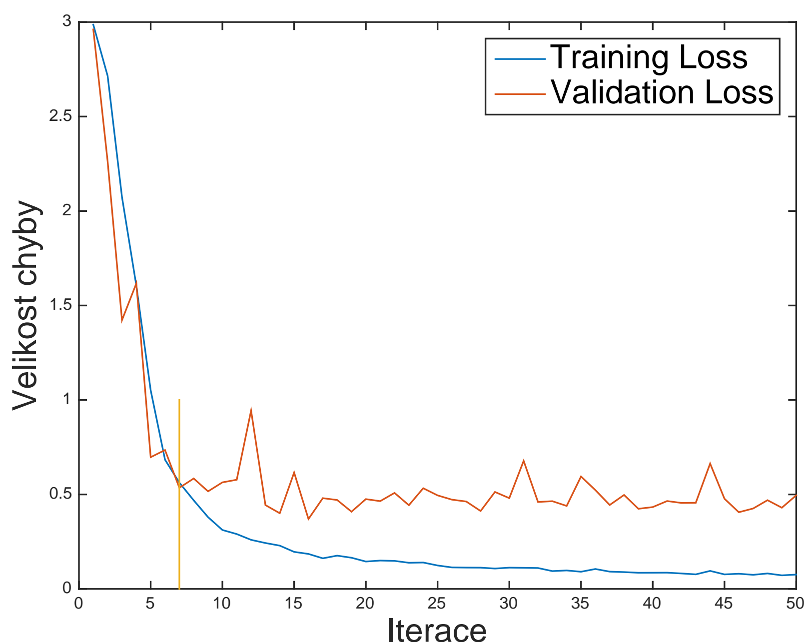
Experimentálně přístupem byla jako nejvhodnější zvolena následující struktura neuronové sítě

1. Vstupní vrstva o 173762 neuronech (velikost reprezentace jednoho dokumentu)
2. Dropout[2] vrstva s intenzitou 60 %
3. Plně propojená vrstva se 300 neurony a ReLU aktivační funkcí
4. Plně propojená vrstva se 100 neurony a ReLU aktivační funkcí
5. Plně propojená vrstva s 50 neurony a ReLU aktivační funkcí
6. Výstupní vrstva s 20 neurony a aktivační funkcí softmax

[2]vrstva, jež při trénování sítě přeruší spojení mezi určitým procentem náhodně vybraných neuronů

3.2.5 Trénování sítě

Pro natrénování sítě byl celý dataset rozdělen na trénovací, validační a testovací data v poměrech 70 %, 20 % a 10 %. Původní délka trénovacího procesu byla 200 cyklů, ale ukázalo se, že již po 7 iteracích (vyznačeno na obrázku 3.1 žlutě) se začala síť nepatrně přetrénovávat. Trénování tedy bylo ukončeno již po 50. iteraci.



Obrázek 3.1: Průběh trénování sítě pro klasifikaci 20 Newsgroup

3.2.6 Porovnání s baseline klasifikátorem

Parametry sítě byly ukládány vždy po deseti trénovacích krocích, bude tedy spočítáno celkem deset hodnot skóre přesnosti (pro každý uložený stav pro validační a testovací data). Ještě před samotným počítáním skóre je však nutné výstup sítě převést na binární[3] tvar. V kódu vytvořeném pro tuto práci jsou dva způsoby převodu

- **Naprahováním**

  Každý prvek výstupu sítě je porovnán s prahovou hodnotou. Je-li prvek větší nebo roven prahové hodnotě, mění se jeho hodnota na 1. V opačném případě je vynulován.

[3]Množina obsahující jen hodnoty 1 či 0

- **Dle maxima**

  Pro každý výstup sítě je nalezen maximální prvek výstupu. Následně je celý výstup sítě převeden na jedno číslo, jímž je index prvku s maximální hodnotou.

Pro počítání skóre přesnosti sítě byl v tomto případě kvůli tvaru cílových hodnot datasetu zvolen druhý způsob výstupu.

	10. iterace	20. iterace	30. iterace	40. iterace	50. iterace
Validační data	0,8347	0,8729	0,8798	0,8910	0,8772
Testovací data	0,8363	0,8842	0,8847	0,8937	0,8863

Tabulka 3.2: Skóre přesnosti sítě pro různé trénovací kroky neuronové sítě

Z tabulky 3.2 lze vyčíst , že síť dosahuje nejlepších výsledků ve 40. iteraci, ačkoliv by se mohlo z trénovacího grafu zdát, že byla v této iteraci již přetrénována. Dosažení této iterace trénovacímu procesu trvalo celkem 29,5 minut.

Jako baseline[4] klasifikátor byl zvolen naivní bayesovský klasifikátor, konkrétně jeho multinomická implementace z knihovny *scikit-learn* a SVM klasifikátor z téže knihovny.

Bayesovský klasifikátor natrénování dosáhl již za 0,23 sekund se skóre přesnosti 0,8389 na validačních datech a 0,8395 na testovacích datech. Oproti tomu SVM klasifikátor dokončil trénovací proces za 20,18 sekund se skóre přesnosti 0,8835 na validačních datech a 0,8959 na datech testovacích.

Z výsledných dat lze vyčíst, že neuronová síť je v tomto klasifikačním problému svým výkonem velmi podobná SVM klasifikátoru a lepší než naivní bayesovský klasifikátor. Značné rozdíly jsou však v době trénování jednotlivých řešení. Pro tuto konkrétní úlohu se tedy jako nejvhodnější jeví SVM klasifikátor.

[4]Výchozí, ten se kterým se porovnávají výsledky

	Validační data	Trénovací data	Doba trénování [s]
Naivní bayesovský klasifikátor	0,8389	0,8395	0,23
SVM klasifikátor	0,8835	0,8959	20,18
Neuronová síť	0,8910	0,8937	1772,34

Tabulka 3.3: Porovnání skóre klasifikátorů u problému multi-class klasifikace

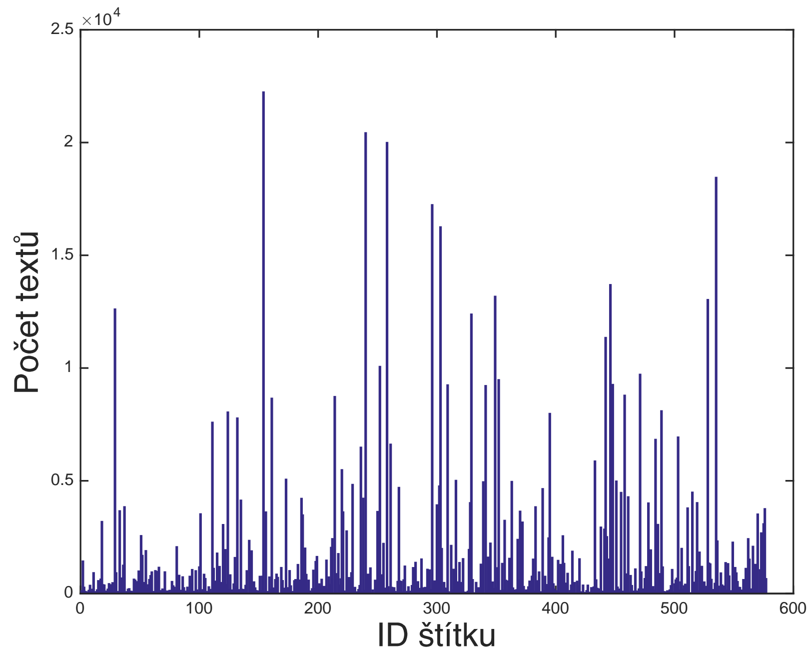
3.3 Multi-label klasifikace

3.3.1 Trénovací data

Jako trénovací dataset pro multi-label klasifikaci byl použit soubor různých českých novinových článků z posledních let.

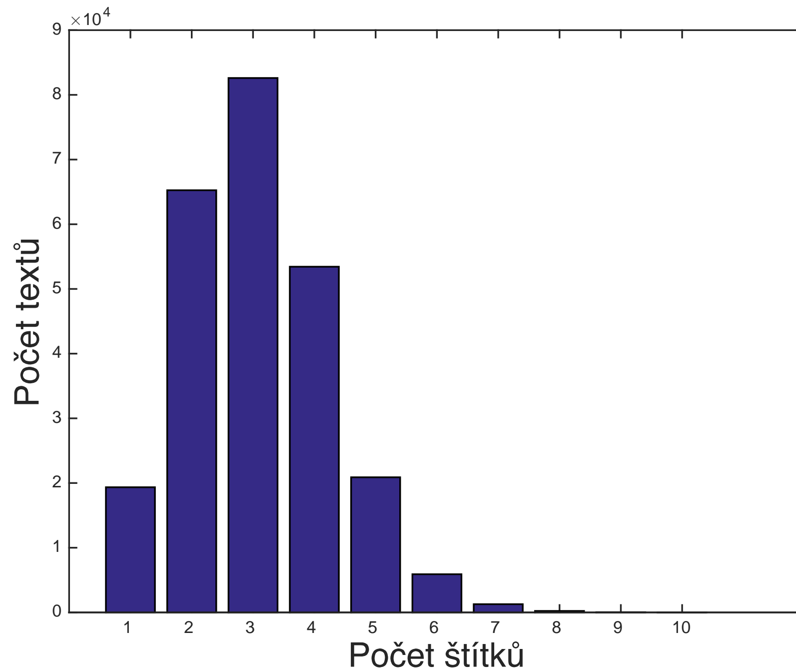
3.3.2 Analýza trénovacích dat

Korpus obsahuje celkem 248975 oštítkovaných dat a 577 různých štítků. Nejvíce použitý štítek je *USA*, pod který spadá celkem 16345 textů. Dalšími nejpoužívanějšími štítky jsou *fotbal* (15629 příslušností), *hospodářství,_podnikání_a_finance* (14814 příslušností), *zločin,_zákon_a_spravedlnost* (13809 příslušností), *lední_hokej* (13032 příslušností) a *ligová_soutěž* (12219 příslušností). Data jsou, co se týče počtu textů pro každý štítek, celkem nekonzistentní, jak ukazuje graf 3.2.



Obrázek 3.2: Počty textů zastupujících jednotlivé štítky

Dataset není konzistentní ani co se počtu štítků na jeden článek týká, jak lze vidět v grafu 3.3. Počty štítků na článek se pohybují od jedné do deseti, zpravidla má článek štítky tři.



Obrázek 3.3: Počet štítků na text

3.3.3 Předzpracování dat

Data z tohoto korpusu jsou zpracována obdobně jako data z datasetu 20 Newsgroups (kapitola 3.2.3). Následně byly takto předzpracované texty uloženy, aby se zrychlil proces trénování odbouráním nutnosti data pokaždé transformovat.

Vzhledem k obsáhlosti korpusu však byl počet trénovacích dat omezen na 10000 vzorků. Při neomezené trénovací množině nebylo možné na dostupném hardwaru sestavit neuronové sítě vzhledem k velikosti vstupního vektoru a nedostatku paměti. Omezením datasetu bylo zároveň ztraceno 24 štítků a trénování probíhalo tedy jen na 553 štítcích.

3.3.4 Struktura neuronové sítě

Bylo testováno několik neuronových sítí s rozdílnou architekturou. Kvůli tomu budou následné neuronové sítě vždy označeny zkratkovitým popisem, kde každá vrstva bude označena číslem udávající počet neuronů a zkratkou své aktivační funkce dle tabulky 3.4 a jednotlivé vrstvy budou odděleny pomlčkou.

Zkratkovité označení	Celý název aktivační funkce
I	vstupní vrstva sítě
linear	lineární funkce
ReLU	rectified linear unit
sigmoid	sigmoidální logistická funkce
tanh	hyperbolický tangens

Tabulka 3.4: Zkratky aktivačních funkcí

Například neuronová síť s výstupní sigmoidální aktivační funkcí obsahující jednu skrytou vrstvu o 200 neuronech s lineární aktivační funkcí bude mít zkratku I–200linear–553sigmoid.

U architektur s výstupní sigmoidální funkcí je chyba výstupu sítě počítána pomocí binární cross-entropie, u sítí s lineární výstupní funkcí je pak chyba počítána kvadraticky.

3.3.5 Trénování sítě

Trénovací proces každé sítě byl nastaven defaultně na 200 iterací s možností trénování ukončit, pokud v jakékoliv iteraci klesne chyba na trénovacích datech na hodnotu menší než 0,0001. Trénování probíhalo pomocí algoritmu $ADAM$ (viz [7]) a dávkově (s velikostí dávky vždy 128).

V práci se původně testovaly sítě se sigmoidální výstupní aktivační funkcí a hyperbolickou tangens funkcí na skrytých vrstvách, neboť takovéto sítě jsou uváděny jako vhodné pro klasifikační problémy [8, 9]. Výsledky takových architektur však nebyly příliš dobré (nejlepší dosažené F1 skóre bylo 0,6290) a byl tedy po inspiraci prací [22] zvolen lineární přístup, který podal mnohem lepší výsledky.

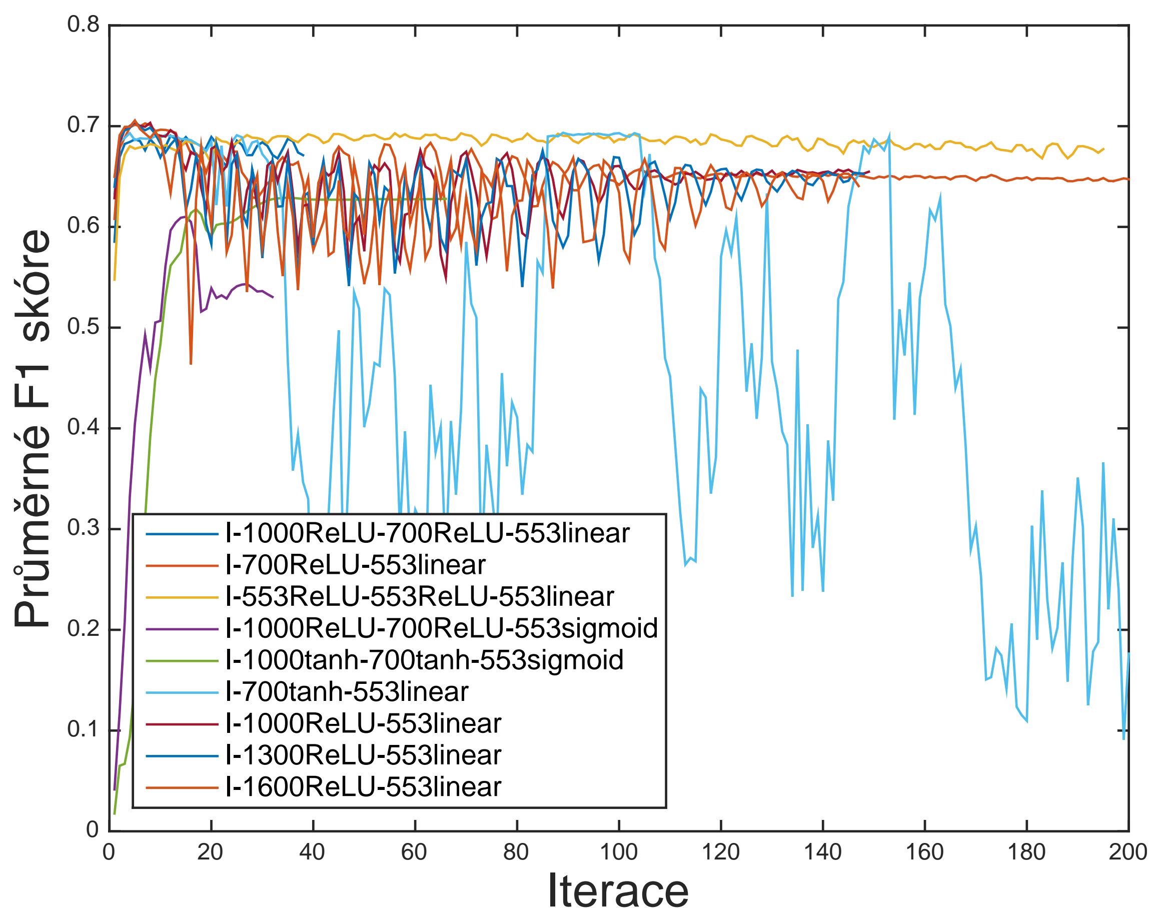
Na konci každé iterace se spočítá, jaký práh je pro každý výstupní neuron nejvhodnější (ve smyslu velikosti F1 skóre na validačních datech). Je tak docíleno největšího možného skóre i při nedokonalém natrénování sítě.

3.3.6 Výsledky trénování

Pro každou natrénovanou neuronovou síť se pro potřeby testování uvažují její parametry z iterace, ve které měla největší průměrné F1 skóre na validačních datech.

Zajímavostí trénování, které si lze povšimnout v grafu 3.4 je, že hodnoty průměrného F1 skóre zpravidla vystoupají téměř na svou maximální hodnotu v prvních dvou až pěti iteracích trénovacího procesu a následně se rozkmitají (což je způsobeno konstantní hodnotou učící konstanty). Dále si lze v témže grafu povšimnout, že všechny struktury neuronových sítí mají velmi podobný průběh změny F1 skóre až na architektury, které v sobě kombinují lineární a sigmoidální (popřípadě hyperbolické) aktivační funkce.

U všech architektur byla následně vybrána iterace, ve které měla neuronová síť nejvyšší hodnotu průměrného F1 skóre a na parametrech z této iterace se spočítalo průměrné F1 skóre sítě na validačních a testovacích datech omezeného datasetu (viz tabulka 3.5).



Obrázek 3.4: Změna průměrného F1 skóre validačních dat během tréninku

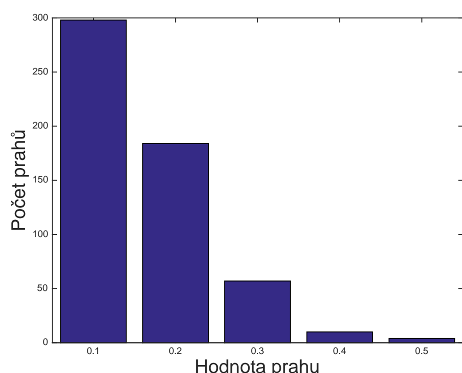
Struktura sítě	Průměrné F1 skóre na	
	validačních datech	testovacích datech
I-1000ReLU-700ReLU-553sigmoid	0,6101	0,5779
I-1000tanh-700tanh-553sigmoid	0,6290	0,6009
I-1000ReLU-700ReLU-553linear	0,6905	0,6655
I-553ReLU-553ReLU-553linear	0,6930	0,6672
I-700tanh-553linear	0,6935	0,6680
I-1000ReLU-553linear	0,7036	0,6766
I-1300ReLU-553linear	0,7052	0,6771
I-1600ReLU-553linear	0,7058	0,6780
I-700ReLU-553linear	0,7028	0,6782

Tabulka 3.5: Porovnání neuronových sítí s různou architekturou

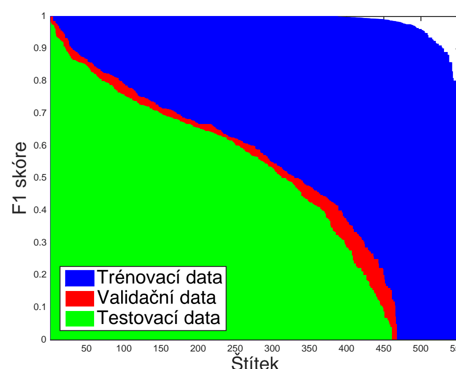
Analýza nejlepší architektury

Obecně podaly lepší výkon jednoduché architektury s lineární výstupní funkcí a s funkcí ReLU na skrytých vrstvách s kvadratickým počítáním chyby predikce. Nejlepší výkon (dle průměrného F1 skóre na testovacích datech) pak podala nejjednodušší z těchto sítí – neuronová síť s lineární aktivační výstupní funkcí a jednou skrytou vrstvou o 700 neuronech s funkcí ReLU.

Je zajímavé se podívat (na obrázku 3.5), jak si síť optimalizovala nastavení jednotlivých prahů. Dle očekávání se nejvíce prahů nastavilo na nejnižší možnou hodnotu 0,1. Tato hodnota je v algoritmu první testovanou, proto díky této informaci můžeme říci, že síť je velmi dobrá v rozhodování, zda daný štítek ke vstupnímu textu **nepatří**.



Obrázek 3.5: Množství jednotlivých prahů



Obrázek 3.6: Hodnoty F1 pro jednotlivé štítky

Další zajímavý graf lze získat spočítáním F1 skóre pro jednotlivé štítky a následným seřazením těchto hodnot dle velikosti (obrázek 3.6). Z něj lze vyčíst, že asi $\frac{1}{6}$ štítků dokáže síť správně určit pouze v trénovacím datasetu.

3.3.7 Porovnání s baseline klasifikátorem

Jako baseline klasifikátor byl použit soubor SVM klasifikátorů (jeden pro každý štítek). Ty byly trénovány algoritmem *stochastic gradient descent*[5] a určení regularizačního parametru bylo provedeno cross validací na validačních datech. Výstupní pravděpodobnosti byly kalibrovány opět na validačních datech pomocí přístupu *CalibratedClassifierCV*[6] z knihovny *scikit-learn*.

Porovnáme-li výsledky baseline klasifikátoru s výsledky nejlepší neuronové sítě (viz tabulka 3.6) zjistíme, že vybraná neuronová síť si u tohoto problému vedla lépe. Z testovaných architektur neuronových sítí podaly horší výkon pouze ty se sigmoidální výstupní funkcí.

	Průměrné F1 skóre na	
	validačních datech	testovacích datech
Baseline klasifikátor	0,6949	0,6589
I-700ReLU-553linear	0,7028	0,6782

Tabulka 3.6: Porovnání výsledků s baseline klasifikátorem

[5] Hodnota chyby je spočítána pro každý vzorek a klasifikátor je pomocí této chyby upraven s postupně klesající učící konstantou (viz [25])

[6] Klasifikátor je otestován na validačních datech a jeho výstup je při následné klasifikaci upraven právě dle těchto výsledků (viz [12])

Závěr

V této práci byla nejdříve představena problematika klasifikace s přímou ukázkou některých klasifikátorů, které následně byly využity pro porovnání výsledků práce.

Druhá kapitola byla věnována neuronovým sítím, problematice jejich struktury, trénování a byl zde uveden příklad využití dopředných neuronových sítí pomocí jejich implementace v Python knihovně *lasagne*.

Ve třetí části byl porovnán výkon klasifikátorů a neuronových sítích v problematikách multi-class a multi-label klasifikace, přičemž výsledky z druhé úlohy byly přinejmenším překvapivé. Ukázalo se, že architektury sítí vhodné dle literatury spíše pro regresní problémy podávají v problematice klasifikace textu lepší výkon, než architektury literaturou určené pro klasifikační problémy. Konkrétně bylo zjištěno, že neuronová síť kombinující lineární funkci s funkcí ReLU podává v multi-label klasifikaci textu lepší výsledky než architektury nelineární, popřípadě než jiné klasifikátory.

Tato práce by mohla sloužit jako základ pro další výzkum v oblasti klasifikace textu, zejména by se dala rozšířit o teoretickou analýzu získaných výsledků a porovnání výkonu konvolučních dopředných neuronových sítích s již zkoumanými přístupy.

Seznam použitých zdrojů

[1] Ethem Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning series)*. The MIT Press, 2009.

[2] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[3] Petr Berka. Bayesovská klasifikace. Dostupné z: `http://sorry.vse.cz/~berka/docs/izi456/kap_5.6.pdf` [citováno 2016-04-12].

[4] Lasagne contributors. Github - lasagne/lasagne: Lightweight library to build and train neural networks in theano. Dostupné z: `https://github.com/Lasagne/Lasagne` [citováno 2016-04-20].

[5] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural Networks, 1989. IJCNN., International Joint Conference on*, pages 593–605. IEEE, 1989.

[6] Michal Hrala. Automatická klasifikace dokumentů s podobným obsahem. Master's thesis, Západočeská univerzita v Plzni, 2012.

[7] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[8] David Kriesel. *A Brief Introduction to Neural Networks*. 2007.

[9] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. *Neural Networks: Tricks of the Trade*. Springer, 1998.

[10] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[11] Iveta Mrázová. Přednáška bayesovské modely. *Dobývání znalostí*, 2015.

[12] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[13] David M. W. Powers. Evaluation: From precision, recall and f-measure to roc, informedness, markedness & correlation. *Journal of Machine Learning Technologies*, 2011.

[14] Vlasta Radová. Přednáška vícevrstvé sítě - lineární síť. *Neuronové sítě a evoluční strategie*, 2015.

[15] Vlasta Radová. Přednáška vícevrstvé sítě - nelineární síť. *Neuronové sítě a evoluční strategie*, 2015.

[16] Jason Rennie. 20 newsgroups. Dostupné z: `http://qwone.com/~jason/20Newsgroups/` [citováno 2016-05-09].

[17] Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46. IBM New York, 2001.

[18] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[19] Daniel Shiffman. *The Nature of Code: Simulating Natural Systems with Processing*. The Nature of Code, 2012.

[20] MG. Siegler. Eric schmidt: Every 2 days we create as much information as we did up to 2003. Dostupné z: `http://techcrunch.com/2010/08/04/schmidt-data` [citováno 2016-04-18].

[21] Christos Stergiou and Dimitrios Siganos. Neural networks. Dostupné z: https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html [citováno 2016-04-20].

[22] Jan Švec. Semi-supervised Learning Algorithm for Binary Relevance Multi-label Classification. In Jianyong Wang, Wojciech Cellary, Dingding Wang, Hua Wang, Shu-Ching Chen, Tao Li, and Yanchun Zhang, editors, *Web Information Systems Engineering – WISE 2014 Workshops*, volume 9418 of *Lecture Notes in Computer Science*, pages 463–477, Cham, 2015. Springer International Publishing. Dostupné z: http://link.springer.com/10.1007/978-3-319-26190-4, doi:10.1007/978-3-319-26190-4.

[23] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *The Journal of Machine Learning Research*, 2:45–66, 2002.

[24] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 412–420, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. Dostupné z: http://dl.acm.org/citation.cfm?id=645526.657137.

[25] Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 694–699. ACM, 2002.