Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra kybernetiky

DIPLOMOVÁ PRÁCE

Plzeň, 2018

Martin Majer

Prohlášení

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

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

V Plzni dne 20. dubna 2018 .....

Poděkování

Tímto bych rád poděkoval vedoucímu diplomové práce, Ing. Luboši Šmídlovi, Ph.D., za cenné rady a připomínky.

Anotace

Tato práce se zabývá klasifikací izolovaných slov pomocí neuronových sítí, klasifikátoru SVM a klasifikátoru založeném na algoritmu Dynamic Time Warping s ohledem na nízkou výpočetní náročnost. V první části jsou představeny příznaky v časové a frekvenční oblasti a odvozeny využité klasifikační algoritmy. Ve druhé části jsou uvedeny zvolené parametrizace testovaných příznaků a struktura navržených klasifikačních algoritmů. V závěru práce je pak vyhodnocena přesnost klasifikace jednotlivých metod pro zvolené parametrizace příznaků.

Klíčová slova: zpracování akustického signálu, extrakce příznaků, detekce klíčových frází, dynamic time warping, support vector machine, neuronová síť

Abstract

This thesis focuses on low computational cost isolated word recognition using neural networks, SVM classifier and Dynamic Time Warping based classifier. First part of the thesis introduces features in time and frequency domain and used classification techniques are derived. Parameterizations of tested features and structure of proposed classification algorithms are described in the second part of the thesis. Classification accuracy results of proposed methods for feature parameterizations are presented at the end of the thesis.

Keywords: acoustic signal processing, feature extraction, keyword spotting, dynamic time warping, support vector machine, neural network

Obsah

1 Úvod

Neuronové sítě byly vyvinuty již v polovině minulého století, ale kvůli nedostatečné výpočetní kapacitě nemohly být plně využity pro řešení reálných problémů. Až v posledních letech, kdy došlo k významnému vývoji v oblasti hardwaru jak pro počítače, tak i pro mobilní a vložená zařízení, začali být plně využívány a to zejména pro komplexní úlohy v oblasti zpracování obrazu a hlasu, kde stabilně překonávají ostatní algoritmy strojového učení. Vývoj výkonných grafických karet a optimalizovaných knihoven pak umožnil rychlé trénování těchto modelů na velkém množství dat a díky výkonným čipům lze provádět predikci v reálném čase i na mobilních zařízeních.

Tato práce se věnuje využití neuronových sítí pro zpracování hlasu a to zejména úloze klasifikace fonémů s využitím základních metod zpracování akustického signálu představených v [1]. Ve zpracování hlasu jsou běžně využívány jak sítě dopředné, tak i rekurentní, které byly vytvořeny pro klasifikaci časových řad či sekvencí a jsou schopny využívat kontextu. Pro trénování neuronových sítí je potřeba velké množství dat, aby byla zajištěna robustnost a schopnost generalizace s tím, že v úloze klasifikace fonémů jsou tato data ve formě zvukových nahrávek a odpovídajících přepisů neboli transkripcí.

Cílem práce je porovnat různé architektury neuronových sítí, které by mohly být využity pro přepis mluvené řeči do textové podoby (speech-to-text) v reálném čase. Za tímto účelem byly voleny především jednodušší sítě s nízkým počtem parametrů.

V první části práce je uvedena obecná teorie optimalizačních algoritmů a neuronových sítí. Druhá část se pak věnuje metodě trénování rekurentních neuronových sítí, která nevyžaduje předsegmentovaná trénovací data (více o segmentaci v [1]). Nakonec jsou porovnány různé architektury jak dopředných, tak i rekurentních neuronových sítí, na několika datových sadách, které vznikly využitím různým typů příznaků.

2 Optimalizační algoritmy

Většina učících se algoritmů využívá jistou formu optimalizace. Optimalizací rozumíme úlohu, kdy minimalizujeme či maximalizujeme předem danou funkci $f(\boldsymbol{x})$ změnou parametru $\boldsymbol{x}$. Hledáme tedy hodnotu x takovou, aby funkce funkce $f(\boldsymbol{x})$ nabývala minimální či maximální hodnoty. Většina optimalizačních metod uvažuje minimalizace funkce $f(\boldsymbol{x})$ a její případná maximalizace se provádí minimalizací funkce $-f(\boldsymbol{x})$.

Funkce, kterou optimalizujeme, nazýváme kritérium (v terminologii strojového učení se také často objevují názvy cenová, ztrátová či chybová funkce). Tato práce se zabývá především optimalizací pro neuronové sítě, kde jsou nejčastěji využívány optimalizační metody založené na gradientu.

Základní metodou založenou na gradientu je tzv. gradientní sestup. Předpokládejme cenovou funkci $J(\boldsymbol{\theta})$ parametrizovanou souborem parametrů $\boldsymbol{\theta}$. Gradientní sestup hledá optimální soubor parametrů $\boldsymbol{\theta}^* = \operatorname{argmin} J(\boldsymbol{\theta})$ pomocí iterativního pravidla pro změnu parametrů

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \tag{2.1}$$

kde ϵ je tzv. konstanta učení, která udává velikost kroku v opačném směru největšího gradientu. Gradientní sestup pro konvexní cenové funkce vždy nalezne globální minimum a pro nekonvexní cenové funkce lokální minimum [2–4].

2.1 Stochastický gradientní sestup

Ačkoliv je gradientní sestup efektivní optimalizační metoda, při optimalizaci nad velkým objemem dat může být velice pomalá a výpočetně náročná, jelikož pro jednu změnu parametrů je třeba spočítat gradient nad celou datovou sadou. Jednou z nejpoužívanějších modifikací gradientního sestupu, která tyto problémy řeší, je stochastický gradientní sestup.

Předpokládejme cenovou funkci ve tvaru záporného logaritmu podmíněné pravděpodobnosti

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim p_{data}} L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{x}^i, \boldsymbol{y}^i, \boldsymbol{\theta}), \tag{2.2}$$

kde p_{data} je množina trénovacích dat o velikost m a $L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) = -\log p(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$ je cena pro jedno pozorování v závislosti na souboru parametrů $\boldsymbol{\theta}$. Pro takto definovanou cenovou

funkci by gradientní sestup musel spočítat gradient

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^i, \boldsymbol{y}^i, \boldsymbol{\theta}), \tag{2.3}$$

jehož komplexita je $O(m)$. Stochastický gradientní sestup nahlíží na gradient jako na střední hodnotu a tudíž se předpokládá, že může být aproximován menšími soubory pozorování náhodně vybíranými z datové sady, tzv. dávky. V optimalizačním kroku je tedy náhodně vybrána dávka pozorování $\boldsymbol{b} = \{x^1, \ldots, x^{m'}\}$, kde m' se volí jako malé číslo v závislosti na výpočetní kapacitě a velikosti m úplné datové sady. Při trénování modelu na grafické kartě (GPU) je vhodné volit velikost dávky jako mocninu dvou a to v rozmezí 8 až 256, aby mohly být plně využity vektorizované operace GPU. Menší dávky mohou mít zároveň regularizační efekt díky šumu, který vnášejí do učícího se procesu. Odhad gradientu je pak vypočten s využitím dávky $\boldsymbol{b}$

$$g = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(\boldsymbol{x}^i, \boldsymbol{y}^i, \boldsymbol{\theta}). \tag{2.4}$$

Změna parametrů je pak dána dle pravidla

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon g. \tag{2.5}$$

Pro pevně danou velikost modelu tedy výpočetní cena nezávisí na velikosti datové sady m [2, 4].

2.2 Další modifikace gradientního sestupu

Jedním z největších problémů při využití gradientního sestupu a gradientního stochastického sestupu je výběr konstanty učení ϵ. Vysoká hodnota ϵ může způsobit fluktuaci okolo lokálního minima nebo dokonce divergenci optimalizačního procesu a nízká hodnota ϵ může mít za následek výrazné zpomalení učení. Konstanta učení se tedy v praxi dynamicky mění v závislosti na počtu epoch k, tedy ϵ_k. Dalším běžným problémem je uvíznutí v mělkém lokálním minimu či sedlovém bodě, což znemožní další učení. Bylo tedy vytvořeno několik modifikací základních gradientních algoritmů, které tyto problémy do jisté míry řeší [2, 4].

2.2.1 Moment

Moment byl navržen za účelem zrychlení trénování a to především při optimalizace ztrátových funkcí, které mají mnoho mělkých lokálních minim nebo v případech, kdy jsou gradienty značně zašuměné. Algoritmus momentu využívá plovoucí průměr minulých gradientů s exponenciálním zapomínáním a pokračuje v jejich směru.

Tento algoritmus zavádí parametr α, který udává, jakou rychlostí mají být minulé gradienty zapomínány. Pravidlo pro změnu parametrů pak vypadá následovně

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right), \tag{2.6}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}. \tag{2.7}$$

Proměnná $\boldsymbol{v}$ akumuluje prvky gradientu $\nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right)$ s tím, že čím větší je parametr α vůči konstantě učení ϵ, tím více minulé gradienty ovlivňují aktuální směr kroku. Stejně jako u konstanty učení, i parametr α může být měnen v závislosti na čase. Většinou je jako počáteční hodnota zvolena 0.5 a je navyšována až na 0.99 [2–4].

2.2.2 Nesterovův moment

Dalším variantou je Nesterovův moment, který dále rozšiřuje algoritmus momentu. Pravidlo pro změnu parametrů se změní na

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} + \alpha \boldsymbol{v}), \boldsymbol{y}^{(i)}) \right), \tag{2.8}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}, \tag{2.9}$$

kde parametry α a ϵ odpovídají stejným parametrům jako u algoritmu momentu. Hlavním rozdílem oproti algoritmu momentu je, že gradient je vyhodnocen až po aplikaci minulých gradientů. Nejprve je tedy proveden krok ve směru akumulovaných minulých gradientů a následně je provedena korekce [2–5].

2.3 ADAM

Cenová funkce bývá často citlivá na určité směry v prostoru parametrů a naopak necitlivá na směry jiné. Moment tento problém do určité míry řeší, ale za cenu zavedení

nového parametru, který je třeba správně nastavit. ADAM je adaptivní metoda, která tento problém řeší tím, že zavede vlastní konstantu učení pro každý parametr a tyto konstanty pak automaticky v průběhu učení adaptuje.

ADAM používá ke změně parametrů plovoucí průměr (první centrální moment) gradientů s exponenciálním zapomínáním r a druhý necentrální moment v. Počáteční hodnota obou momentů je nastavena na nulu a zejména v prvních krocích algoritmu je potřeba momenty opravit korekčním faktorem. Parametry β_1 a β_2 udávají rychlost zapomínání a δ je malá konstanta [2–4, 6].

---

inicializace

$\boldsymbol{r} = 0$, $\boldsymbol{v} = 0$, $\boldsymbol{t} = 0$

while *zastavovací podmínka není splněna* **do**

$t \leftarrow t + 1$

výpočet gradientu

$g = \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

změna odhadu prvního centrálního momentu

$\boldsymbol{r} \leftarrow \beta_1 \boldsymbol{r} + (1 - \beta_1) g$

změna odhadu druhého necentrálního momentu

$\boldsymbol{v} \leftarrow \beta_2 \boldsymbol{v} + (1 - \beta_2) g \odot g$

korekce odhadu prvního centrálního momentu

$\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \beta_1^t}$

korekce odhadu druhého necentrálního momentu

$\hat{\boldsymbol{v}} \leftarrow \frac{\boldsymbol{v}}{1 - \beta_2^t}$

změna parametrů

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + -\epsilon \frac{\hat{\boldsymbol{r}}}{\sqrt{\hat{\boldsymbol{v}} + \delta}}$

end

Algoritmus 1: ADAM.

3 Neuronové sítě

Neuronová síť je algoritmus, který je schopen aproximovat i silně nelineární funkce a zároveň je schopen dosáhnout vysoké míry statistické generalizace. Tento parametrický model bývá zpravidla složen z několika vrstev reprezentovanými vektory, jejichž dimenze udává šířku modelu. Prvky těchto vektorů, jednotky či perceptrony, pracují paralelně a reprezentují funkce zobrazující vstupní vektor na skalár. Cílem je natrénovat parametry tohoto modelu tak, aby dokázal splnit zadanou úlohu s minimální chybou. Jedná se tedy o optimalizační úlohu.

Základním rozdílem mezi běžnou optimalizací a optimalizací v neuronových sítí je, že optimalizace v neuronových sítích a ve strojovém učení obecně probíhá nepřímo. To znamená, že ačkoliv optimalizujeme zvolenou metriku P, která v jistém smyslu kvantifikuje výkon algoritmu na dané úloze, minimalizujeme jinou cenovou funkci $J(\boldsymbol{\theta})$ za účelem minimalizace metriky P. V běžném optimalizačním problému bychom minimalizovali přímo cenovou funkci $J(\boldsymbol{\theta})$ za účelem její minimalizace [2, 3, 7].

3.1 Cenová funkce

Jedním ze zásadních aspektů při návrhu neuronových sítí je právě výběr cenové funkce. Výběr cenové funkce závisí na dané úloze a požadovaném výstupu. Cenovou funkci definujeme stejným způsobem jako u ostatních parametrických modelů, které generují hustotu pravděpodobnosti $p(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$ a při trénování využívají principu maximální věrohodnosti. Většinou tedy cenovou funkci definujeme jako vzájemnou entropii mezi trénovacími daty a predikcemi modelu

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{data}} \log p_{model}(\boldsymbol{y} \mid \boldsymbol{x}). \tag{3.1}$$

Předpis cenové funkce se liší model od modelu v závislosti na tvaru $\log p_{model}$ a kromě definice ceny také může obsahovat regularizační prvky. Výhodou využití principu maximální věrohodnosti je, že specifikací modelu $p(\boldsymbol{y} \mid \boldsymbol{x})$ zároveň definujeme cenovou funkci $\log p(\boldsymbol{y} \mid \boldsymbol{x})$ [2, 7].

3.2 Dopředné neuronové sítě

Dopředné neuronové sítě, občas také nazývané vícevrstvé perceptrony, jsou základním typem neuronových sítí. Jejich cílem je aproximovat určitou funkci f^*, např. klasifikátor

$\boldsymbol{y} = f^*(\boldsymbol{x})$ zobrazuje vstup $\boldsymbol{y}$ na výstupní třídu $\boldsymbol{x}$. Dopředná neuronová síť tedy definuje zobrazení $\boldsymbol{y} = f^*(\boldsymbol{x}; \boldsymbol{\theta})$ a učí se optimální hodnoty parametrů $\boldsymbol{\theta}$ při kterých by měla být dosažena nejlepší aproximace funkce.

Jak název napovídá, tok informací směřuje od vstupu $\boldsymbol{x}$ přes několik po sobě jdoucích výpočetních vrstev definujících f až k výstupu $\boldsymbol{y}$. Jedná se tedy o složení několika různých funkcí do řetězové struktury. Takto definovaný model lze také popsat jako orientovaný acyklický graf, který určuje závislosti mezi funkcemi. Mějme například síť složenou ze tří funkcí $f^{(1)}$, $f^{(2)}$ a $f^{(3)}$ spojených do řady $f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x})))$. V tomto případě nazýváme $f^{(1)}$ první skrytou vrstvou, $f^{(2)}$ druhou skrytou vrstvou a $f^{(3)}$ výstupní vrstvou. Celková délka tohoto řetězce pak udává hloubku modelu [2].

Cílem trénování sítě je, aby se naučila odpovídající zobrazení mezi $f(\boldsymbol{x})$ a $f^*(\boldsymbol{x})$. Učení probíhá na základě trénovacích dat, která poskytují zašuměná aproximovaná pozorování $f^*(\boldsymbol{x})$ vyhodnocena v různých bodech. Ke každému pozorování $\boldsymbol{x}$ je k dispozici také skutečná hodnota $\boldsymbol{y} \approx f^*(\boldsymbol{x})$. Výstupní vrstva sítě se tedy pro každou trénovací hodnotu $\boldsymbol{x}$ snaží vyprodukovat hodnotu blízkou $\boldsymbol{y}$. Chování skrytých vrstev není přímo dáno trénovacími daty a síť se je musí naučit využívat k získání požadovaného výsledku, tedy k aproximaci $f^*(\boldsymbol{x})$ [2, 7].

3.2.1 Architektura

Klíčovým prvkem při návrhu neuronových sítí je jejich architektura. Architekturou sítě či modlu rozumíme celkovou strukturu sítě - kolik má mít jednotek a jak mají být tyto jednotky mezi sebou propojeny.

Jak již bylo uvedeno, neuronové sítě jsou tvořeny skupinami jednotek, které jsou uspořádány do jednotlivých vrstev. Většina architektur tyto vrstvy uspořádává do řetězové struktury, kde každá vrstva je funkcí vrstvy, která ji předchází. V této struktuře je pak první vrstva definována jako

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right), \tag{3.2}$$

druhá vrstva jako

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{x} + \boldsymbol{b}^{(2)}\right), \tag{3.3}$$

a tak dále, kde $g^{(i)}$ je aktivační funkce vrstvy i, $\boldsymbol{W}^{(i)\top}$ je váhová matice vrstvy i a $\boldsymbol{b}^{(i)}$ je

prahový vektor vrstvy i.

Pro takto definovanou řetězovou architekturu je pak hlavním problémem určení hloubky sítě a šířky každé vrstvy. Vhodnou architekturu pro danou úlohu je třeba nalézt pomocí experimentů založených na sledování chyby na validační datové sadě a apriorní znalosti o úloze a datech [2, 3, 7, 8].

3.2.2 Výstupní jednotky

Reprezentace výstupu je úzce spojena s danou úlohou a tím i s výběrem cenové funkce. Předpokládejme, že dopředná neuronová síť poskytuje výstupní vrstvě soubor skrytých příznaků $\boldsymbol{h} = f(\boldsymbol{x}; \boldsymbol{\theta})$, tj. výstup poslední skryté vrstvy. Cílem výstupní vrstvy je pak provést určitou transformaci těchto příznaků, aby síť plnila úlohu, pro kterou byla navržena. Tato transformace je provedena použitím aktivační funkce $g(\boldsymbol{h})$. Ačkoliv existuje mnoho aktivačních funkcí, které lze ve výstupní vrstvě využít, zde se zaměříme pouze na aktivační funkci softmax, která je využívána v experimentech provedených v rámci této práce.

Aktivační funkce softmax je využívána pro úlohy, kde je třeba reprezentovat výstup jako hustotu pravděpodobnosti diskrétní proměnné s n možnými hodnotami (výstupními třídami). K odvození aktivační funkce softmax využijeme znalosti o úloze binární klasifikaci, při které predikujeme hodnotu

$$\hat{\boldsymbol{y}} = P(\boldsymbol{y} = 1 \mid \boldsymbol{x}), \tag{3.4}$$

kde $\hat{\boldsymbol{y}} \in [0, 1]$. Aby byla zajištěna numerická stabilita při optimalizaci, budeme raději hodnotu

$$\boldsymbol{z} = \log \tilde{P}(\boldsymbol{y} = 1 \mid \boldsymbol{x}). \tag{3.5}$$

Aplikací exponenciální funkce a následnou normalizací bychom pak dostali Bernoulliho rozložení pravděpodobnosti řízeného sigmoidální funkcí.

Pro generalizaci tohoto postupu pro diskrétní proměnnou s n hodnotami je tedy potřeba získat vektor $\hat{\boldsymbol{y}}$, kde $\hat{\boldsymbol{y}}_i = P(\boldsymbol{y} = i \mid \boldsymbol{x})$. Pro každý prvek $\hat{\boldsymbol{y}}_i$ musí platit $\hat{\boldsymbol{y}}_i \in [0, 1]$ a zároveň $\sum_i \hat{\boldsymbol{y}}_i = 1$, aby bylo možné tento vektor interpretovat za hustotu pravděpodobnosti. Nejprve je potřeba provést predikci vrstvou s lineární aktivační funkcí,

která predikuje nenormalizované logaritmované pravděpodobnosti

$$\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}, \tag{3.6}$$

kde $\boldsymbol{z}_i = \log \tilde{P}(\boldsymbol{y} = i \mid \boldsymbol{x})$. Softmax funkce pak aplikuje exponenciální funkci a normalizuje $\boldsymbol{z}$, čímž získáme požadované $\hat{\boldsymbol{y}}$.

$$softmax(\boldsymbol{z}_i) = \frac{\exp(\boldsymbol{z}_i)}{\sum_j \exp(\boldsymbol{z}_j)} \tag{3.7}$$

Natrénováním parametrů modelu pak bude výstupní vrstva s aktivační funkcí softmax predikovat podíly počtů všech pozorovaných výsledků v trénovací datové sadě [2].

$$softmax(\boldsymbol{z}(\boldsymbol{x}; \boldsymbol{\theta}))_i \approx \frac{\sum_{j=1}^{m} 1_{\boldsymbol{y}^{(j)}=i, \boldsymbol{x}^{(j)}=\boldsymbol{x}}}{\sum_{j=1}^{m} 1_{\boldsymbol{x}^{(j)}=\boldsymbol{x}}} \tag{3.8}$$

3.2.3 Skryté jednotky

Jak název napovídá, skryté jednotky jsou jednotky skryté vrstvy, jejichž vstupem je vektor $\boldsymbol{x}$, který je transformován na $\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b}$. Na takto transformovaný vstup je pak po prvcích aplikována nelineární aktivační funkce $g(\boldsymbol{z})$. Volba aktivačních funkcí skrytých vrstev vyžaduje mnoho experimentů a vyhodnocení přesnosti modelu na validační datové sadě. V dnešní době se pro dopředné neuronové sítě většinou volí jednotky s aktivační funkcí ReLU (z anglického "rectified linear unit") či její modifikace, pro sítě rekurentní jsou pak voleny funkce hyperbolický tangens a hard sigmoid.

- **ReLU** - tyto jednotky využívají aktivační funkci $g(\boldsymbol{z}) = \max\{0, \boldsymbol{z}\}$. Jedná se tedy o lineární jednotky s prahem v bodě nula - levá polovina jejich definičního je rovna nule. To zaručuje rychlý výpočet a vysokou hodnotu gradientu, kdykoliv je jednotka aktivní. Zásadním nedostatkem je, že tyto jednotky se pomocí gradientních metod nemohou učit z pozorování, které mají aktivační hodnotu rovnou nule. Existuje proto několik modifikací, které zajistí, že jednotky budou mít gradient všude (např. Leaky ReLU, PReLU, Maxout).

- **sigmoid** a **tanh** - tyto jednotky využívají logistickou funkci (sigmoid) $g(\boldsymbol{z}) = \sigma(\boldsymbol{z})$, resp. hyperbolický tangens $\tanh(\boldsymbol{z}) = 2\sigma(2\boldsymbol{z}) - 1$. Hlavním nedostatkem sigmoidální funkce je její citlivost a náchylnost k saturaci, kdy může dojít k tzv. explozi či

vymizení gradientu a síť nebude schopná se učit.

- **hard sigmoid** - tyto jednotky využívají aktivační funkci $g(\boldsymbol{z}) = \max(0, \min(1, \frac{z+1}{2})$. Jedná se o lineární aproximaci funkce sigmoid a díky své výpočetní nenáročnosti jsou využívány v sítích typu LSTM [2, 3].

3.2.4 Algoritmus zpětného šíření

Dopředná neuronová síť přijme na vstupu počáteční informaci o pozorování $\boldsymbol{x}$, kterou poté šíří skrz skryté jednotky ve všech skrytých vrstvách až k výstupní vrstvě, která vygeneruje odhad $\hat{\boldsymbol{y}}$. Informační tok tedy proudí skrz síť směrem dopředu a tomuto procesu se říká dopředné šíření. V trénovací fázi probíhá dopředné šíření tak dlouho, dokud není vygenerována skalární cena $J(\boldsymbol{\theta})$. Algoritmus zpětného šíření (anglicky backpropagation) pak umožní zpětný tok informace od ceny skrz síť za účelem výpočtu gradientu.

Analytický výpočet gradientu je poměrně přímočarý, nicméně numericky může být velice náročný. Algoritmus zpětného šíření provádí výpočet gradientu $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ pomocí jednoduché a výpočetně nenáročné procedury. Takto vypočtený gradient je pak využit při učení pomocí gradientních metod [2, 3].

Algoritmus zpětného šíření využívá k výpočtu gradientu řetězové pravidlo. Řetězové pravidlo se běžně využívá k výpočtu derivací složených funkcí, které jsou složeny z funkcí, jejichž derivace jsou známé. Mějme reálné číslo x a funkce $f : \mathbb{R} \mapsto \mathbb{R}$ a $g : \mathbb{R} \mapsto \mathbb{R}$. Dále předpokládejme, že $y = g(x)$ a $z = f(g(x)) = f(y)$. Řetězové pravidlo je pak dáno jako

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}. \tag{3.9}$$

Toto pravidlo lze snadno rozšířit ze skalárního případu. Předpokládejme, že $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{y} \in \mathbb{R}^n$, $g : \mathbb{R}^m \mapsto \mathbb{R}^n$ a $f : \mathbb{R}^n \mapsto \mathbb{R}$. Pokud $\boldsymbol{y} = g(\boldsymbol{x})$ a $\boldsymbol{z} = f(\boldsymbol{y})$, pak

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}_i} = \sum_j \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{y}_j}\frac{\partial \boldsymbol{y}_j}{\partial \boldsymbol{x}_i} \tag{3.10}$$

Zapíšeme-li rovnici (3.10) vektorově, získáme tvar

$$\nabla_{\boldsymbol{x}} \boldsymbol{z} = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\top} \nabla_{\boldsymbol{y}} \boldsymbol{z}, \tag{3.11}$$

kde $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \in \mathbb{R}^{n \times m}$ je Jakobián $\boldsymbol{x}$. Toto pravidlo lze dále snadno rozšířit i na tensory, jenž jsou v neuronových sítích často využívány (podrobněji v [2, 9]).

Předpokládejme síť o hloubce l, kde každé vrstvě náleží váhová matice $\boldsymbol{W}^{(i)}$, $i \in \{1, \ldots, l\}$ a prahový vektor $\boldsymbol{b}^{(i)}$. Ztrátová funkce $L(\boldsymbol{y}, \hat{\boldsymbol{y}})$ závisí na skutečné hodnotě $\boldsymbol{y}$ a na výstupu sítě $\hat{\boldsymbol{y}}$, který síť vygeneruje pro vstup $\boldsymbol{x}$. Celková cena J pro zjednodušení odpovídá přímo ztrátové funkci L a neobsahuje žádnou regularizační složku.

$\boldsymbol{h}^{(0)} = \boldsymbol{x}$
for $k = 1, \ldots, l$ **do**
  $\boldsymbol{a}^{(k)} = \boldsymbol{W}^{(k)} \boldsymbol{h}^{(k-1)} + \boldsymbol{b}^{(k)}$
  $\boldsymbol{h}^{(k)} = f(\boldsymbol{a}^{(k)})$
end
$\hat{\boldsymbol{y}} = \boldsymbol{h}^{(l)}$
$J = L(\boldsymbol{y}, \hat{\boldsymbol{y}})$

Algoritmus 2: Dopředné šíření.

Při zpětném průchodu jsou počítány gradienty aktivací $\boldsymbol{a}^{(k)}$ pro každou vrstvu k počínaje výstupní vrstvou až k první skryté vrstvě. Tyto gradienty indikují, jak by se měly změnit výstupy jednotlivých vrstev, aby došlo ke snížení chyby. Spočtené gradienty vah a prahů mohou být rovnou využity pro změnu parametrů pomocí optimalizačního algoritmu [2].

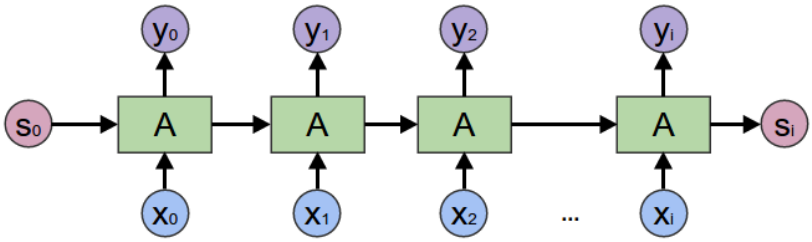
---

výpočet gradientu výstupní vrstvy (po dopředném průchodu sítě)
$g \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\boldsymbol{y}, \hat{\boldsymbol{y}})$
for $k = l, l-1, \ldots, 1$ **do**
  převedení gradientu do tvaru před aplikací nelineární aktivační funkce
  $g \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = g \odot f'(\boldsymbol{a}^{(k)})$
  výpočet gradientů prahových vektorů a váhových matic
  $\nabla_{\boldsymbol{b}^{(k)}} J = g$
  $\nabla_{\boldsymbol{W}^{(k)}} J = g \boldsymbol{h}^{(k-1)\top}$
  šíření gradientu do nižší vrstvy
  $g \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top} g$
end

Algoritmus 3: Zpětné šíření.

3.3 Rekurentní neuronové sítě

Rekurentní neuronové sítě (dále jen RNN z anglického ”recurrent neural networks”) jsou typem neuronových sítí, které umí zpracovávat sekvenční data, tj. sekvenci hodnot $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}$, a předávat si informace mezi jednotlivými časovými kroky. Základním rozdílem oproti dopředným neuronovým sítím je zavedení zpětné smyčky a využití sdílení parametrů. RNN tedy netrénuje pro každý časový krok samostatný soubor parametrů, ale tyto parametry jsou sdíleny přes všechny prvky sekvence. Díky tomu jsou RNN schopny generalizovat na různé délky sekvencí a na různé pozice důležitých informací v čase (např. dvě téměř stejné věty, kde jedna obsahuje datum na začátku věty a druhá na konci).



Obrázek 1: Schéma rekurentní neuronové sítě. Převzato z [10].

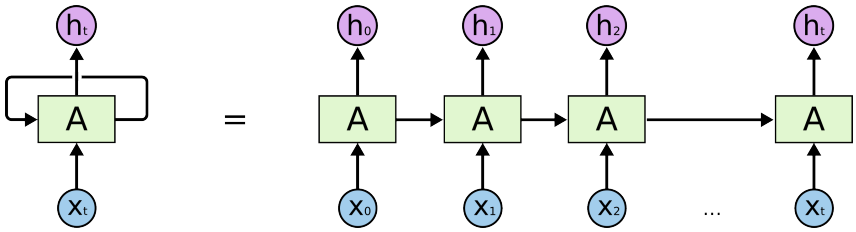
Sdílení parametrů je zajištěno ”rozvinutím”rekurentního výpočtu do grafu s opakující se strukturou. Prvek sekvence v každém časovém kroku je tedy zpracováván stejnou sítí. Pro jednoduchost předpokládejme, že RNN pracuje se sekvencí obsahující vektory $\boldsymbol{x}^{(t)}$, $t = 1, \ldots, \tau$ a model je ve tvaru

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta}), \tag{3.12}$$

kde $\boldsymbol{s}$ je stav systému. Tento rekurzivní vztah je možné pro konečný počet kroků τ rozvinout τ-krát. Například pro model daný vztahem (3.12) a $\tau = 3$ získáme

$$\boldsymbol{s}^{(3)} = f(\boldsymbol{s}^{(2)}; \boldsymbol{\theta}) = f(f(\boldsymbol{s}^{(1)}; \boldsymbol{\theta}); \boldsymbol{\theta}). \tag{3.13}$$

Tímto rozvinutím zajistíme, že vztah neobsahuje žádnou rekurenci a je možné ho reprezentovat jako acyklický orientovaný graf [2, 11].



Obrázek 2: Ilustrace rozvinutí rekurentní neuronové sítě. Převzato z [11].

Základní rovnici RNN získáme přidáním závislosti na externím vstupu $\boldsymbol{x}^{(t)}$

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}), \tag{3.14}$$

kde $\boldsymbol{h}$ je stav skrytých jednotek. Při trénování sítě na danou úlohu se pak síť učí využívat $\boldsymbol{h}^{(t)}$ jako zobrazení relevantních informací z minulých časových kroků od vstupu až po t. Toto zobrazení je ztrátové, jelikož se jedná o zobrazení sekvence o proměnné délce $(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$ na vektor pevně dané délky $\boldsymbol{h}^{(t)}$.

Definujme si tedy rovnice pro dopředné šíření základní RNN s aktivační funkcí hyperbolický tangens ve skryté vrstvě. Síť je určena ke klasifikaci n tříd, kde výstup $\boldsymbol{o}$ reprezentuje nenormalizované pravděpodobnosti jednotlivých tříd výstupní diskrétní proměnné a aplikací operace softmax je pak získán výsledný vektor $\hat{\boldsymbol{y}}$ normalizovaných pravděpodobností nad výstupem. Dopředné šíření vyžaduje specifikaci počátečního stavu $\boldsymbol{h}^{(0)}$, poté jsou pro každý časový krok $t = 1, \ldots, \tau$ vypočteny následující rovnice

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \tag{3.15}$$

$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)}), \tag{3.16}$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \tag{3.17}$$

$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}(\boldsymbol{o}^{(t)}), \tag{3.18}$$

kde parametry $\boldsymbol{b}$ a $\boldsymbol{c}$ jsou prahové vektory a váhové matice $\boldsymbol{U}$, $\boldsymbol{V}$ a $\boldsymbol{W}$ popořadě odpovídají skrytým spojením mezi vstupem a skrytou vrstvou, mezi skrytou vrstvou a výstupem a mezi skrytou vrstvou a předcházející skrytou vrstvou [2].

3.3.1 Algoritmus zpětného šíření časem

Pro výpočet gradientu RNN se využívá algoritmus zpětného šíření časem. Jedná se o generalizovaný algoritmus zpětného šíření nad rozvinutým výpočetním grafem a jeho

časová náročnost je $O(\tau)$. Časovou náročnost tohoto algoritmu není možné snížit paralelizací, jelikož dopředných průchod je sekvenční a hodnoty v aktuálním časovém kroku jsou závislé na předchozích hodnotách.

Předpokládejme reprezentaci rozvinuté RNN ve tvaru acyklického orientovaného grafu, jehož uzly obsahují parametry $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{W}$, $\boldsymbol{b}$, $\boldsymbol{c}$, a sekvenci uzlů indexovaných časem t pro $\boldsymbol{x}^{(t)}$, $\boldsymbol{h}^{(t)}$, $\boldsymbol{o}^{(t)}$ a $L^{(t)}$. Pro každý uzel grafu N pak potřebujeme rekurzivně spočítat gradient $\nabla_N L$ v závislosti na uzlech grafu, které tento uzel následují. Začneme tedy rekurzi v uzlu, který bezprostředně předchází výslednou ztrátu L

$$L(\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\}) = -\sum_t \log_{p_{model}} \left(\boldsymbol{y}^{(t)} \mid \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\}\right), \tag{3.19}$$

$$\frac{\partial L}{\partial L^{(t)}} = 1. \tag{3.20}$$

Pro výpočet gradient $\nabla_{\boldsymbol{o}^{(t)}} L$ nad všemi výstupy v čase t pro všechna i, t je opět využito řetězové pravidlo

$$(\nabla_{\boldsymbol{o}^{(t)}} L)_i = \frac{\partial L}{\partial \boldsymbol{o}_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial \boldsymbol{o}_i^{(t)}} = \hat{\boldsymbol{y}}_i^{(t)} - \mathbf{1}_{i,\boldsymbol{y}^{(t)}}. \tag{3.21}$$

Výpočet gradientu dále pokračuje přes další uzly počínaje koncem sekvence. V konečném časovém kroku τ má $\boldsymbol{h}^{(\tau)}$ za následníka pouze $\boldsymbol{o}^{(\tau)}$, tedy

$$\nabla_{\boldsymbol{h}^{(\tau)}} L = \boldsymbol{V}^\top \nabla_{\boldsymbol{o}^{(\tau)}} L. \tag{3.22}$$

Gradient se dál šíří od $t = \tau - 1$ až k $t = 1$ s tím, že $\boldsymbol{h}^{(t)}$ (pro $t < \tau$) má jako své předchůdce $\boldsymbol{o}^{(t)}$ a $\boldsymbol{h}^{(t+1)}$. Gradient skryté vrstvy je pak

$$\begin{aligned} \nabla_{\boldsymbol{h}^{(t)}} L &= \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{o}^{(t)}} L) \\ &= \boldsymbol{W}^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) \operatorname{diag}\left(1 - (\boldsymbol{h}^{(t+1)})^2\right) + \boldsymbol{V}^\top (\nabla_{\boldsymbol{o}}^{(t)} L), \end{aligned} \tag{3.23}$$

kde $\operatorname{diag}\left(1 - (\boldsymbol{h}^{(t+1)})^2\right)$ je Jakobián hyperbolického tangensu pro skryté jednotky i v čase $t + 1$. Na základě gradientů skrytých stavů lze dopočítat gradienty parametrů. Vzhledem k tomu, že parametry jsou sdíleny mezi jednotlivými časovými kroky, zavedem nové proměnné $\boldsymbol{W}^{(t)}$, resp. $\boldsymbol{U}^{(t)}$, které jsou kopiemi váhových matic $\boldsymbol{W}$, resp. $\boldsymbol{U}$ a značí, jakou

mírou tyto váhy přispívají ke gradienty v čase t [2].

$$\nabla_{\boldsymbol{c}} L = \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}} \right)^\top \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L \tag{3.24}$$

$$\nabla_{\boldsymbol{b}} L = \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}} \right)^\top \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\left(1 - (\boldsymbol{h}^{(t)})^2\right) \nabla_{\boldsymbol{h}^{(t)}} L \tag{3.25}$$

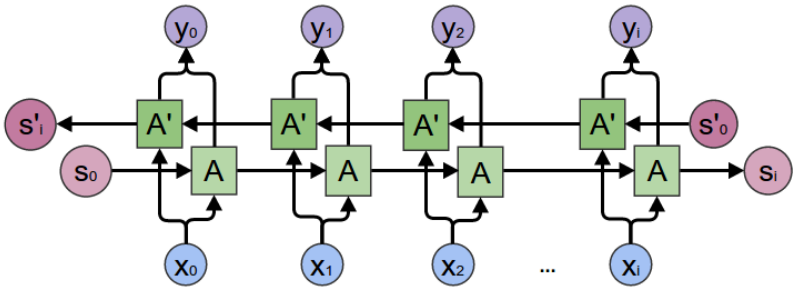
$$\nabla_{\boldsymbol{V}} L = \sum_t \sum_t \left(\frac{\partial L}{\partial \boldsymbol{o}_i^{(t)}} \right) \nabla_{\boldsymbol{V}} \boldsymbol{o}_i^{(t)} = \sum_t (\nabla_{\boldsymbol{o}^{(t)}} L) \boldsymbol{h}^{(t)\top} \tag{3.26}$$

$$\nabla_{\boldsymbol{W}} L = \sum_t \sum_t \left(\frac{\partial L}{\partial \boldsymbol{h}_i^{(t)}} \right) \nabla_{\boldsymbol{W}^{(t)}} \boldsymbol{h}_i^{(t)} = \sum_t \operatorname{diag}\left(1 - (\boldsymbol{h}^{(t)})^2\right) (\nabla_{\boldsymbol{h}}^{(t)} L) \boldsymbol{h}^{(t-1)\top} \tag{3.27}$$

$$\nabla_{\boldsymbol{U}} L = \sum_t \sum_t \left(\frac{\partial L}{\partial \boldsymbol{h}_i^{(t)}} \right) \nabla_{\boldsymbol{u}^{(t)}} \boldsymbol{h}_i^{(t)} = \sum_t \operatorname{diag}\left(1 - (\boldsymbol{h}^{(t)})^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L) \boldsymbol{x}^{(t)\top} \tag{3.28}$$

3.3.2 Obousměrné rekurentní neuronové sítě

Rekurentní neuronové sítě zachycují do stavu v čase t pouze informaci z minulosti, tj. informace ze vstupů $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t-1)}$ a aktuálního vstupu $\boldsymbol{x}^{(t)}$. V mnoha úlohách ovšem může správná predikce $\boldsymbol{y}^{(t)}$ záviset i na budoucích vstupech či na celé sekvenci. Tento problém řeší obousměrná rekurentní neuronová síť (dále BRNN z anglického "bidirectional recurrent neural network").



Obrázek 3: Schéma obousměrné rekurentní neuronové sítě. Převzato z [10].

BRNN kombinuje dvě RNN, kde jedna síť se pohybuje v čase kupředu od začátku sekvence, zatímco druhá se pohybuje v čase zpět od konce sekvence. Tyto dvě sítě mezi sebou sdílí skryté stavy $\boldsymbol{h}^{(t)}$ (síť dopředná) a $\boldsymbol{g}^{(t)}$ (síť zpětná), což zajistí, že výstupní jednotky $\boldsymbol{o}^{(t)}$ jsou závislé jak na minulosti, tak na budoucnosti. Výstupy skrytých stavů $\boldsymbol{h}^{(t)}$ a $\boldsymbol{g}^{(t)}$ nejsou mezi sebou nijak propojeny. Pro výpočet gradientů lze opět využít algoritmus zpětného šíření časem dle postupu, který je shrnut v algoritmu č. 4 [2, 12].

for $t = 1, \ldots, \tau$ **do**
  dopředné šíření dopřednou sítí od $t = 1$ do $t = \tau$
  dopředné šíření zpětnou sítí od $t = \tau$ do $t = 1$
  dopředné šíření nad výstupními jednotkami obou sítí
end
for $t = 1, \ldots, \tau$ **do**
  zpětné šíření nad výstupními jednotkami obou sítí
  zpětné šíření dopřednou sítí od $t = \tau$ do $t = 1$
  zpětné šíření zpětnou sítí od $t = 1$ do $t = \tau$
end
změna parametrů

Algoritmus 4: Postup trénování obousměrné rekurentní neuronové sítě.

3.3.3 LSTM

Rekurentní sítě jsou velice náchylné na problém vymizení či saturace gradientu. K těmto problémům dochází zejména při modelování dlouhodobých závislostí, kdy jsou při výpočtu váhy násobeny několikrát samy sebou a v závislosti na jejich řádu mohou nabývat velmi malých či velmi velkých hodnot. To má pak za následek trvalou deaktivaci některých neuronů či neschopnost sítě se učit. Tento problém řeší sítě LSTM (z anglického "long short-term memory") a jejich modifikace GRU (z anglického "gated reccurent unit").

LSTM sítě, na rozdíl od běžných RNN, nejsou tvořeny jednotkami, ale buňkami, které kromě vnější rekurence obsahují i rekurenci vnitřní (smyčky). Každá buňka má stejný vstup a výstup jako obyčejná rekurentní jednotka, ale má více parametrů díky systému jednotek opatřených bránou, které řídí tok informací stavem buňky. Jedná se o vstupní bránu, zapomínací bránu a výstupní bránu, které určují, jaké vstupní informace mají být vpuštěny do stavu buňky, jaké informace ze stavu buňky odstranit a jaké informace ze stavu buňky mají být vypuštěny na její výstup.

Nejdůležitější komponentou LSTM je stavová jednotka $\boldsymbol{s}_i^{(t)}$ (pro buňku i v čase t), která je opatřena lineární smyčkou. Váha této smyčky je řízena jednotkou se zapomínací bránou $\boldsymbol{f}_i^{(t)}$, která díky aktivační funkci sigmoid umožňuje nastavovat tuto váhu na hodnoty mezi 0 a 1

$$\boldsymbol{f}_i^{(t)} = \sigma\left(\boldsymbol{b}_i^f + \sum_j \boldsymbol{U}_{i,j}^f \boldsymbol{x}_j^{(t)} + \sum_j \boldsymbol{W}_{i,j}^f h_j^{(t-1)}\right), \tag{3.29}$$

kde $\boldsymbol{x}^{(t)}$ je vstupní vektor v čase t, $\boldsymbol{h}^{(t)}$ vektor skryté vrstvy obsahující výstupy všech

LSTM buňek, $\boldsymbol{b}^f$, $\boldsymbol{U}^f$ a $\boldsymbol{W}^f$ jsou popořadě prahový vektor, váhová matice vstupu a rekurentní váhová matice zapomínací brány. Stav buňky se pak řídí pravidlem

$$\boldsymbol{s}_i^{(t)} = \boldsymbol{f}_i^{(t)} \boldsymbol{s}_i^{(t-1)} + \boldsymbol{g}_i^{(t)} \sigma \left(\boldsymbol{b}_i + \sum_j \boldsymbol{U}_{i,j} \boldsymbol{x}_j^{(t)} + \sum_j \boldsymbol{W}_{i,j} \boldsymbol{h}_j^{(t-1)} \right), \tag{3.30}$$

kde $\boldsymbol{b}$, $\boldsymbol{U}$, $\boldsymbol{W}$ jsou popořadě prahový vektor, váhová matice vstupu a rekurentní váhová matice buňky. Stav buňky je také závislý na jednotce s vstupní bránou $\boldsymbol{g}_i^{(t)}$, která se chová podobně jako jednotka se zapomínací branou

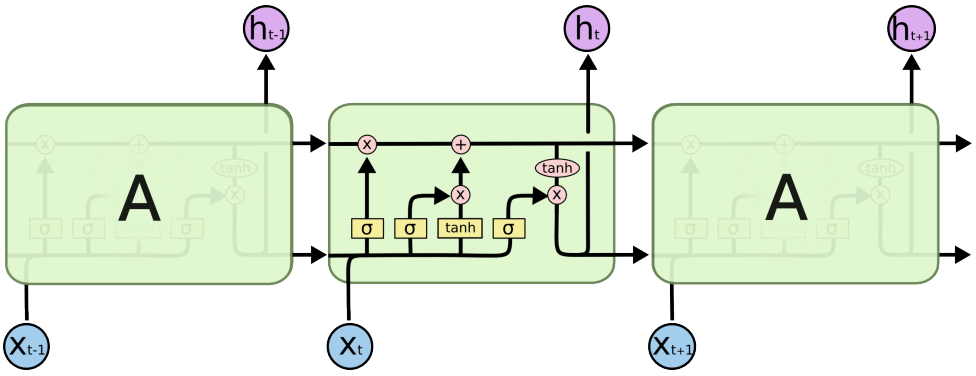
$$\boldsymbol{g}_i^{(t)} = \sigma \left(\boldsymbol{b}_i^g + \sum_j \boldsymbol{U}_{i,j}^g \boldsymbol{x}_j^{(t)} + \sum_j \boldsymbol{W}_{i,j}^g h_j^{(t-1)} \right), \tag{3.31}$$

kde $\boldsymbol{b}^g$, $\boldsymbol{U}^g$ a $\boldsymbol{W}^g$ jsou popořadě prahový vektor, váhová matice vstupu a rekurentní váhová matice vstupní brány. Výstup buňky $\boldsymbol{h}_i^{(t)}$ je pak řízen pomocí výstupní brány $\boldsymbol{q}_i^{(t)}$, která opět řídí propustnost pomocí aktivační funkce sigmoid

$$\boldsymbol{h}_i^{(t)} = \tanh(\boldsymbol{s}_i^{(t)}) \boldsymbol{q}_i^{(t)}, \tag{3.32}$$

$$\boldsymbol{q}_i^{(t)} = \sigma \left(\boldsymbol{b}_i^o + \sum_j \boldsymbol{U}_{i,j}^o \boldsymbol{x}_j^{(t)} + \sum_j \boldsymbol{W}_{i,j}^o \boldsymbol{h}_j^{(t-1)} \right), \tag{3.33}$$

kde $\boldsymbol{b}^o$, $\boldsymbol{U}^o$ a $\boldsymbol{W}^o$ jsou popořadě prahový vektor, váhová matice vstupu a rekurentní váhová matice výstupní brány [2, 11, 13].



Obrázek 4: Schéma LSTM buňky. Převzato z [11].

3.3.4 GRU

Sítě GRU dále zjednodušují LSTM architekturu ponecháním pouze nezbytných komponent. Výstup buňky a stav buňky je sloučen do jednoho stavu, který je řízen jednotkou

vzniklou sloučením jednotky s vstupní bránou a zapomínací bránou. Rovnice dopředného šíření má tedy tvar

$$\boldsymbol{h}_i^{(t)} = \boldsymbol{u}_i^{(t-1)}\boldsymbol{h}_i^{(t-1)} + (1 - \boldsymbol{u}_i^{(t-1)})\sigma\left(\boldsymbol{b}_i + \sum_j \boldsymbol{U}_{i,j}\boldsymbol{x}_j^{(t-1)} + \sum_j \boldsymbol{W}_{i,j}r_j^{(t-1)}\boldsymbol{h}_j^{(t-1)}\right), \quad (3.34)$$

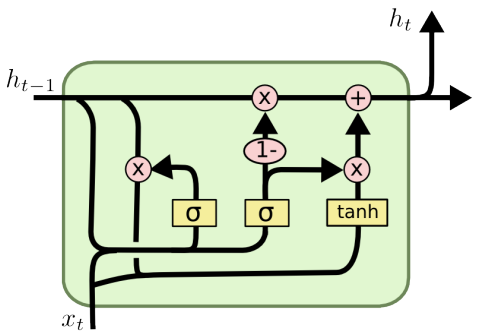
kde jednotka $\boldsymbol{u}$ modifikuje stav buňky

$$\boldsymbol{u}_i^{(t)} = \sigma\left(\boldsymbol{b}_i^u + \sum_j \boldsymbol{U}_{i,j}^u\boldsymbol{x}_j^{(t)} + \sum_j \boldsymbol{W}_{i,j}^u\boldsymbol{h}_j^{(t)}\right) \quad (3.35)$$

a jednotka $\boldsymbol{r}$ řídí, které informace stavu budou využity pro výpočet příštího cílového stavu

$$\boldsymbol{r}_i^{(t)} = \sigma\left(\boldsymbol{b}_i^r + \sum_j \boldsymbol{U}_{i,j}^r\boldsymbol{x}_j^{(t)} + \sum_j \boldsymbol{W}_{i,j}^r\boldsymbol{h}_j^{(t)}\right). \quad (3.36)$$

Takto definované jednotky s branami umožňují vhodně vybírat části stavového prostoru a uchovávat je ve stavu buňky. Zároveň sloučením výstupu a stavu buňky dochází k významnému snížení paměťové a výpočetní náročnosti [2, 11].



Obrázek 5: Schéma GRU buňky. Převzato z [11].

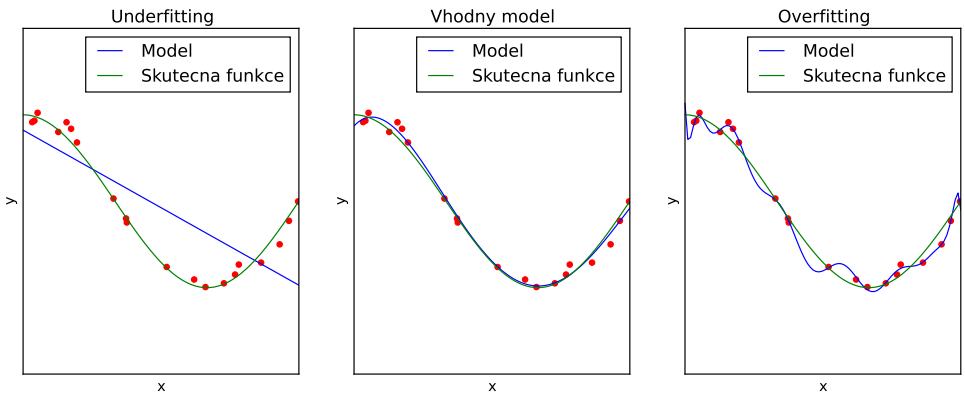
4 Kapacita modelu

Základní vlastností každého modelu by měla být generalizace. To znamená, že model musí být schopný správně klasifikovat nejen pozorování, na kterých byl natrénován, ale i většinu nových, dříve neviděných pozorování. Při trénování modelu se tedy běžně dělí datová sada na tři - trénovací, validační a testovací, kdy předpokládáme, že tyto sady podléhají stejnému rozložení a jednotlivá pozorování jsou nezávislá. Cílem trénování modelu je nalézt takové parametry, pro které bude mít model podobnou chybu na všech třech datových sadách. Model by měl tedy splňovat tyto vlasnosti:

1. musí minimalizovat chybu na trénovací sadě,
2. musí minimalizovat rozdíl mezi chybou nad trénovací a testovací (popř. validační) sadou.

Během optimalizace parametrů často dochází ke dvěma nežádoucím jevům:

- **underfitting** (podtrénování) - k underfittingu dochází, pokud model není schopný dostatečně minimalizovat chybu na trénovací sadě, tj. není schopný se naučit informační souvislosti obsažené v trénovacích pozorováních;
- **overfitting** (přetrénování) - k overfittingu dochází, pokud model není schopný minimalizovat rozdíl mezi chybou nad trénovací a testovací sadou, tj. model si "zapamatuje"trénovací data místo toho, aby se naučil souvislosti obsažené v trénovacích pozorováních a na nových, dosud neviděných pozorováních selhává [2, 3, 8].



Obrázek 6: Příklad overfittingu a underfittingu.

Oba tyto jevy lze ovlivnit pomocí tzv. kapacity modelu nebo pomocí regularizačních metod. Modely s nízkou kapacitou jsou náchylné na underfitting a naopak modely s vysokou kapacitou jsou náchylné na overfitting. Kapacita modelu je většinou dána přímo zvolenou architekturou, kde snížením počtu parametrů sítě omezíme počet volných stupňů volnosti a tím lze získat vyšší generalizaci [2, 3].

4.1 Regularizace

Regularizace umožňuje navýšit generalizaci modelu bez toho, aby byla ovlivněna kapacita modelu - nevyžadují změnu architektury modelu. Jedná se o modifikaci učícího se algoritmu, která snižuje generalizační chybu, ale zároveň nesnižuje chybu trénovací [2].

4.1.1 Penalizace velikosti parametrů

Populární metodou pro regularizaci jsou penalizační metody a to zejména L1 a L2 penalizace. Tyto penalizační metody nepřímo omezují kapacitu modelu přidáním penalizace $\Omega(\boldsymbol{\theta})$, která odpovídá normě parametrů, k cenové funkci J. Regularizovaná cenová funkce pak vypadá následovně

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{x}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta}), \tag{4.1}$$

kde síla regularizace je řízena parametrem α (s vyšší hodnotou regularizace roste). Trénovací algoritmus pak minimalizuje jak původní cenovou funkci J, tak i zvolenou metriku velikosti parametrů $\Omega(\boldsymbol{\theta})$. V případě neuronových sítí jsou regularizovány pouze váhy a prahy jsou ponechány bez regularizace [2, 3].

4.1.2 Zanesení šumu

Další metodou regularizace, která neovlivňuje model samotný, je augmentace trénovacích dat ve formě aditivního šumu. Ačkoliv neuronové sítě nejsou obecně robustní vůči šumu, většinu klasifikačních i regresních úloh jsou schopny řešit i pokud jsou vstupní data zatíženy malým náhodným šumem. Trénovací pozorování jsou tedy vždy před vstupem do modelu zatížena novým náhodným šumem, který vede k vyšší generalizaci výsledného modelu [2].

4.1.3 Předčasné ukončení

Při trénování velkých modelů s dostatečně velkou reprezentační kapacitou často dochází k jevu, kdy chyba na trénovací sadě pomalu během času klesá, zatímco chyba na validační sadě pomalu roste. Vrátíme-li se k nastavení parametrů v čase s nejnižší validační chybou, můžeme získat lepší model s nižší chybou i na testovací sadě.

Algoritmus předčasného ukončení (early stopping) je snadno implementovatelný a výpočetně nenáročný. Pokaždé, když dojde k poklesu chyby na validační sadě, uložíme si kopii parametrů modelu. Jakmile trénování modelu skončí, vrátíme se k nejlepšímu souboru parametrů místo ponechání souboru parametrů z poslední trénovací iterace. Trénování je předčasně ukončeno, pokud žádná změna parametrů nevede k nižší chybě na validační sadě, než které bylo dosaženo s aktuálně nejlepším souborem parametrů, pro předem daný počet trénovacích iterací.

Tato forma regularizace nevyžaduje téměř žádné změny trénovacího procesu či modifikaci cenové funkce. Tento algoritmus lze využít buď samostatně nebo spolu s libovolnou regularizační metodou [2].

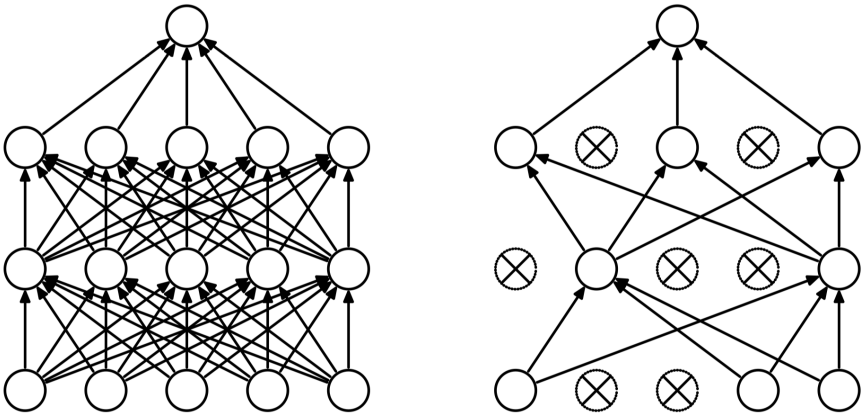
4.1.4 Dropout

Dropout je výpočetně nenáročnou metodou regularizace, která nejlépe funguje pro modely, které využívají dávkové učení s malými kroky. Pokaždé, kdy pozorování vstoupí do dávky, je vytvořena binární maska, která je pak aplikována na všechny vstupy skrytých jednotek v síti. Tato maska je pro každou jednotku vytvořena náhodně nezávisle na ostatních. Jednotky, jejichž odpovídající prvek v masce je roven nule, jsou pak pro dané pozorování vyloučeny z učícího se procesu.

Pravděpodobnost, že v masce na danou pozici bude vybrána hodnota 1, je dána předem zvoleným parametrem. Většinou se volí pravděpodobnost 0.8, že bude při trénování využita vstupní jednotka, a pravděpodobnost 0.5, že bude využita skrytá jednotka.

Maskováním se do učícího procesu vnáší jistá forma šumu, která má za následek adaptivní poškození informačního obsahu na vstupu sítě a na vstupu skrytých jednotek. Tím je zamezeno tomu, aby se určitá jednotka zaměřila na rozpoznávání jednoho signifikantního příznaku, zatímco jiná jednotka by byla téměř nevyužita. Vyloučením náhodných jednotek z učícího se procesu jsou tak jednotky donuceny ”rozdělit”si informace mezi sebou.

Dropout opět může být využit spolu s ostatními metodami regularizace [2, 3, 14].



Obrázek 7: Vlevo - před aplikací dropoutu, vpravo - po aplikaci dropoutu. Převzato z [14].

5 CTC

6 Klasifikace fonémů

7 Vyhodnocení

8 Závěr

9 Connectionist temporal classification

Mnoho úloh, kde je třeba se naučit sekvence, vyžadují predikci labelů sekvencí ze zašuměných a nesegmentovaných vstupních dat. Například ve zpracování řeči je akustický signál přepsán na slova nebo na jednotky podúrovní slova, např. fonémy. RNN, jakožto modely schopny se naučit sekvence, se pro tento typ úloh zdají býti vhodné. Nicméně vyžadují předsegmentovaná trénovací data a zároveň je třeba jistý post-processing jejich výstupu k transformaci na sekvence labelů.

Labelování nepředsegmentovaných sekvenčích dat je dlouhodobý problém v úlohách, kde je třeba se naučit sekvenci. Je to zejména běžné v úlohách vnímání, jako je například rozpozánávní ručně pssaného písma, rozpoznávání řeči a rozpoznávání gest, kde vstupní data jsou reprezentována jako zašuměný tok (stream) reálných čísel anotovaných pomocí řetězců diskrétních labelů, jako jsou například písmena či slova.

Na tyto úlohy labelování byly velmi často využívány modely jako například HMM, které ovšem vyžadujíc jisté předpoklady k jejich využití: specifická znalost domény úlohy pro správný návrh stavových modelů, předpoklad, že vstupní pozorování jsou nezávislá a hlavně trénování HMM je generativní zatímco úloha olabelování sekvence je diskriminativní. Na druhou stranu RNN nevyžadují žádné předešlé znalosti o datech, pouze je třeba zvolit reprezentaci jejich vstupních a výstupních dat. Zároveň mohou být trénovaný diskriminativně a jejich vnitřní stavy představují velmi silný mechaniskus pro modelování časových řad. Zároveň jsou velice robustní vůči temporal a spatial šumu.

Dříve ovšem nebylo možné RNN na úlohu olabelování sekvence napřímo využít, jelikož jejich objective functions jsou definovány samostatně pro každý bod v trénovací sekvenci - jinými slovy, RNN mohou být natrénovíny pouze ke klasifikaci series nezávislých labelů. To znamená, že trénovací data musejí být předsegmentována a výstup sítě musí projít jistým post-processingem, abychom získali výslednou sekvenci labelů.

V této kapitole si uvedeme metodu, která nám umožní využít RNN přímo k predikci sekvence labelů bez toho, abych potřebovali olabelovaná předsegmentovaná data a postprocessingu výstupu. Základním principem této metody je interpretovat výstupy sítě jako pravděpodobnostní rozložení (hustota?) nad všemi možnými sekvencemi labelů v závislosti na vstupní sekvenci. Na základě této hustoty můžeme odvodit objective function, která přímo maximalizuje pravděpodobnost správného olabelování. Vzhledem k tomu, že tarto obj. func. je diferencovatelná, můžeme pak využít standarní BPTT k natrénovaání

sítě.

Úloze labelování nepředsegmentované sekvence dat budeme říkat temporal classification a pokud k této úlzoe využijeme RNN, budeme používat connectionist temporal classification (volně přeloženo XYZ, dále jen CTC). Oproti tomu, úloze nezávislého labelování každého kroku (framu či okénka) vstupní sekvence budeme nazývat framewise classification (podrobnyý postup, jak je provedena framewise clf. naleznete v [moje bakalářka jako zdroj]).))

Nejprve si připravíme matematickou formalizaci pro temporal classification, dále si popíšeme reprezentaci výstupu, která nám umožní využít RNN jako temporal klasifikátor a nakonec si uvedeme, jak je taková síť potom trénována.

CTC PAPER

Connectionism is a set of approaches in the fields of artificial intelligence, cognitive science and philosophy of mind, that attempts to represents mental or behavioral phenomena as emergent processes of interconnected networks of simple units.

WIKI

Mějmě úlohu rozpoznávání řeči. Pro tuto úlohu máme typicky kdispozici zvukové narhrávjky/stopy a odpodvídající transkripce/přepisy. Bohužel ale neznáme, které písmeno v přepisu odpovídá jako části zvukové stopy. Proto je třeba ručně či dle předtrénovaným modelem přiřadit ke každému časovému okamžiku foném, která se v tomto okamžiku vyskytuje. Toto může být pro velké datasety poměrně pracné a ne vždy je možné přesně určit hranici mezi jednotlivými fonémy.

DISTILLPUB

9.1 Temporální klasifikace

Mějme trénovací množinu S z pevně daného rozložení $\mathcal{D}_{\mathcal{X}\times\mathcal{Z}}$. Prostor vstupních hodnot $\mathcal{X} = (\mathcal{R})^*$ je množina všech sekvencí tvořených m dimenzionálními reálnými vektory. Prostor výstupních hodnot $\mathcal{Z} = L^*$ je množina všech sekvencí nad konečnou abecedou (alphabet) L labelů. Obecně budeme nazývat prvky L^* sekvencemi labelů či labellings. Každé pozorování v S sestává z páru (x, z). Cílová (target) sekvence $z = (z_1, z_2, \ldots, z_U)$ je nejvýše tak dlouhá, jako vstupní sekvence $x = (x_1, x_2, \ldots, x_T)$, tedy $U \leq T$. Vzhledem k tomu, že vstupní a cílové sekvence nejsou obecně stejně dlouhé, není možné je a priori align them.

Cílem je za využití S natrénovat temporal klasifikátor $h : \mathcal{X} \mapsto \mathcal{Z}$, který bude klasifikovat předem neviděné vstupní sekvence tak, aby minimalizoval nějakou error measure specifickou pro danou úlohu (pro klasifikaci foném se může jednat například o LER, tj. label error rate, kde je cílem minimalizovat počet transkripčních chyb).

CTC PAPER

Hledáme přesné zobrazení z X na Y s tím, že X a Y se mohou lišit svoji délkou a přesně neznáme, které prvky X odpovídají kterým prvkům Y. CTC tento problém řeší a pro vstupní sekvenci X najde výstupní rozložení (distribution) nad všemi možnými Y.

DISTILLPUB

9.2 CTC

Dále se budeme věnovat reprezentacu výstupu, který nám umožní využít RNN spolu s CTC. Zásádním krokem je transformovat výstupy sítě do podmíněné hustoty pravděpodobnosti nad sekvencemi labelů. Síť pak může být použita jako klasifikátor výběrem nejvíce pravděpodobného labellingu pro danou vstupní sekvenci.

CTC síť vyžaduje softmax výstupní vrstvu, která obsahuje o jednu jetnoku více, než je počet labelů v L. Aktivace prvních $|L|$ je jednotek je interpretována jako pravděpodobnost pozorování koespundíjících labelů v daných časech. Aktivace $|L| + 1$ jednotky (jednotky navíc) je pak interpretována jako pravděpodobnost pozorování "blank"/"prázdného"čí žádného labelu. Dohromady tyto výstupy definují pravděpodobosti všech možných zarovnání (alignment) sekvencí labelů vůči vstupní sekvencu. Celková pravděpodobnost libovolné sekvece labelů apk může být nalezena sečtením pravděpodobností jejich různých zarovnání.

Formálnější, pro vstupní sekvence x délky T definujeme RNN s m vstupy, n výstupy a váhovým vektorem w, který slouží jako spojité zobrazení $\mathcal{N}_w : (\mathbb{R}^m)^T \mapsto (\mathbb{R}^n)^T$. Nechtť $y = \mathcal{N}_w(x)$ je sekvence výstupu sítě a $y_k^{(t)}$ je aktivace výstupní jednotky k v čase t. Potom můžeme $y_k^{(t)}$ interpretovat jako pravděpodobnost pozorování labelu k v čase t, které definuje distribution nad množinou L'^T sekvence délky T nad abecedou $L' = L \cup \{blank\}$:

$$p(\pi \mid x) = \prod_{t=1}^{T} y_{\pi_t}^{(t)}, \forall \pi \in L'^T.$$

Prvky L'^T budeme nazývat cestami a budeme je označovat π. Z výše uvedené rovnice

(odkaz) je implicitní předpoklad, že výstup sítě v různých časových okamžicích je conditionally nezávislý, given the interní stav sítě. Toto je zaručení tím, že neexistuje žádná zpětná vazba z výstupní vsrtvy sítě zpět do sebe čí kamkoliv jinak v síti.

Dalším krokem je definece many-to-one zobrazení $\mathcal{B} : L'^T \mapsto L^{\leq T}$, kde $L^{\leq T}$ je množina možných labbelingu, tj. množina sekvenci o délce menší či rovné T nad původní abecenou labelů L). Toto zobrazení získáme jednodušše odstraněním prázdných labelů a labelů, co se opakují ze získaných cest, např. $\mathcal{B}(a - ab-) = \mathcal{B}(-aa - -abb) = aab$. Nakonec využijeme $\mathcal{B}$ k definici podmíněné pravděpodobnosti daného labellingu $l \in L^{\leq T}$ jako sumu pravděpodobností všech odpovídajících cest:

$$p(l \mid x) = \sum_{\pi \in \mathcal{B}^{-1}(l)} p(\pi \mid x)$$

CTC PAPER

Vzhledem k tomu, že ne pro všechny prvky vstupní sekvence dává smysl mapovat je na výstup, zavádíme nový prázdný token. Díky nim můžeme např. ignorovat ticho, které není obsaženo ve výstupu či povolit několik opakujících se znaků v řadě (aby z neexistuje nevzniklo nexistuje). Tento token neodpovídá žádnému znaku a z výsledné sekvence je odstraněn.

DISTILL PUB

9.3 Konstrukce klasifikátoru

Výstupem klasifkátoru by mělo být nejvíce pravděpodobné olabelování pro vstupní sekvenci

$$h(x) = \text{argmax}_{l \in L^{\leq T}} p(l \mid x)$$

(FIXNOUT ARGMIN a UNDERSCRIPT)

V terminologii HMM bychom nazvali tento proces hledání labellingu dekódováním. V tomto případě se pro dekódování nabízí dvě aproximativní metody.

První metoda, dekódování nejlepší cesty (best path decoding), je založeno na předpodkladu, že nejpravděpodobnější cesta odpovídá nejpravděpodobnějšímu labellingu:

$$h(x) \approx \mathcal{B}(\pi^*)$$

kde

$$\pi^* = \text{argmax}_{\pi \in N^t} p(\pi \mid x).$$

Dekódování nejlepší cesty lze snadno vypočítat, jelikož π^* je pouze konkatenace nejvíc aktivních výstupů v každém časovém kroce. Nicméně není zaručeno, že tato metoda nalezne nejlepší labelling.

Druhá metoda, prefix search decoding, sice zaručuje nalezení nejlepšího labellingu, alej e výpočetně náročnější a v jistých případech může její heurestika selhat. Tato metoda vyžaduje úpravu dopředného a zpětného algortimu uvedeného dále a maximální počet prefixů, které musí být expandovány roste exponencielně s rostoucí délkou vstupní sekvence. Aby tato metoda byla upočitatelná, většinou je kombinována s dalšími heuristikami (např. BEAM prořezávání).

CTC PAPER

9.4 Trénování sítě

Zatím jsme si uvedli, jak reprezentovat výstup sítě tak, abychom mohli použít RNN s CTC. Nyní si odvodíme objective function, která nám uméžní natrénovat CTC síť pomocí metod založených na gradientnm sestupu.

Objective function odvodíme ze základního principu maximum likelyhood, tedy že minimalizací této funkce maximalizujeme log likelyhood cíloveho olabelování. Váhy této sítě pak můžeme snadno spočítat pomocí obyčejného zpětnného šíření v čase (BPTT).

CTC PAPER

CTC modely jsou věšinou trénovány s využitím RNN k odhadu pravděpodobností v každém kroce $p_t(a_t \mid X)$, která většinu funguje velice dobře díky vzhledem k tomu, že bere ohled i na kontext ve vstupní sekvenci. Nicméně je možné využít libovolný algortimus, který dokáže vygenerovat hustotu pravděpodobnosti nad výstupními třídami pro vstup o pevně dané velikosti.

DIS—TILL PUB

9.4.1 CTC zpětný a dopředný algoritmus

Potřebujeme efficient způsob, jak spočítast podmíněnou pravděpodobnost $p(l \mid x)$ jednotlivých labellingů. To je značně problematické, jelikož je potřeba spočítat sumu přes

všechny cesty odpovídající danému labbelingu, kterých může být velmi mnoho. Proto tento problém vyřešíme pomocí algoritmu dynamického programování, který je podobný dopřednému-zpětnému algoritmu pro HMM. Hlavním idea, jak spočítat sumu všech cest odpovídajících labellingu je, že tato suma může být rozložena na iterativní sumu přes cesty odpovídající prefixu toho labellingu. Iterace pak mohou být eficciently spočteny pomocí rekurzivních dopředných a zpětných proměnných.

Pro nějakou sekevenci q délky r kde $q_{1:p}$ a $q_{r-p:r}$ označuí prvních a posledních p symbolů. Potom pro labelling l zavedeme dopřednou proměnnou $\alpha_t(s)$, která odpovídá celkové pravděpodobnosti $l_{1:s}$ v čase t, tedy

$$\alpha_t(s) = \sum_{\pi \in N^T : \mathcal{B}(\pi_{1:t}) = l_{1:s}} \prod_{t'=1}^{t} y_{\pi_{t'}}^{t'}$$

kde $\alpha_t(s)$ může být rekurzivně spočtena z $\alpha_{t-1}(s)$ a $\alpha_{t-1}(s-1)$.

Abychom povolili využití prázdných labelů ve výstupní cestě, předpokládejme modifikovanou sekvenci labelů $l^{'}$, která vloží prázdný label na začátek a na konec sekvence a mezi každé dva labely. Délka sekvence $l^{'}$ je tedy $2|l|+1$. Při výpočtu pravděpodobností prefixů $l^{'}$ umožníme přechody mezi prázdnými a neprázdnými labely a také mezi každým párem unikátních neprázdných labelů. Dále umožníme (allow), aby všechny prefixy začínali buď prázdných symbolem b nebo prvním symbolem v l. Tím získáváme inicializační hodnoty algoritmu

$$\alpha_1(1) = y_b^{(1)}$$

$$\alpha_1(2) = y_{l_1}^{(1)}$$

$$\alpha_1(s) = 0, \forall s > 2$$

a rekurzi

$$\alpha_t(s) = \begin{cases} \bar{\alpha}_t(s) y_{l'_s}^t & \text{pokud } l'_s = b \text{ nebo } l'_{s-2} = l'_s \\ (\bar{\alpha}_t(s) + \alpha_{t-1}(s-2)) y_{l'_s}^t & \text{jinak} \end{cases}$$

kde

$$\bar{\alpha}_t(s) = \alpha_{t-1}(s) + \alpha_{t-1}(s-1)$$

.

Pravděpodobnost l je potom suma výsledných pravděpodobností $l^{'}$ s a bez finální prázdného labelu v čase T

$$p(l \mid x) = \alpha_T(|l^{'}|) + \alpha_T(|l^{'}| - 1)$$

Obdobně zadefinujeme zpětnou proměnnou $\beta_t(s)$ jako výslednou pravděpodobnsot $l_{s:|l|}$ v čase t:

$$\beta_t(s) = \sum_{\pi \in N^T : \mathcal{B}(\pi_{t:T}) = l_{s:|l|}} \prod_{t^{'}=t}^{T} y^{t^{'}}_{\pi_{t^{'}}}$$

$$\beta_T(|l^{'}|) = y^{(T)}_b$$

$$\beta_T(|l^{'}| - 1) = y^{(T)}_{l_l}$$

$$\beta_T(s) = 0, \forall s < |l^{'}| - 1$$

$$\beta_t(s) = \begin{cases} \bar{\beta}_t(s) y^t_{l^{'}_s} & \text{pokud } l^{'}_s = b \text{ nebo } l^{'}_{s+2} = l^{'}_s \\ (\bar{\beta}_t(s) + \beta_{t+1}(s+2)) y^t_{l^{'}_s} & \text{jinak} \end{cases}$$

kde

$$\bar{\beta}_t(s) = \beta_{t+1}(s) + \beta_{t+1}(s+1)$$

.

OBRAZEK Z CTC PAPERU

Aby při výpočty nedošlo k podtečení a byla zajištěna numerická stabilita, zadefinujme si přeškálované dopředné a zpětné proměnné

$$\hat{\alpha}_t(s) = \frac{\alpha_t(s)}{\sum_s \alpha_t(s)}$$

$$\hat{\beta}_t(s) = \frac{\beta_t(s)}{\sum_s \beta_t(s)}$$

a nahraďmě α za $\hat{\alpha}$ v rovnicích (odkaz na tu s vidličkou a tu pod ní) a β za $\hat{\beta}$ v rovnicích (odkaz na tu s vidličkou a tu pod ní).

CTC PAPER

Dosazením vzorců a úpravou vzorců (podrobnější v [odkaz na CTC paper]) pak můžeme síť natrénovat na základě maximum likelihood, kde zpětné šíření gradientu skrz softmax

vrstvu je definování následovně

$$\frac{\partial O^{ML}(\{x,z\},\mathcal{N}_w)}{\partial u_k^{(t)}} = y_k^{(t)} - \frac{1}{\underset{k}{(t)} Z_{(t)}} \sum_{s \in lab(z,k)} \hat{\alpha}_t(s)\hat{\beta}_t(s)$$

(POROVNAT ZAVORKY u t)

, kde $u_k^{(t)}$ jsou nenormalizované výstupy a $lab(z,k) = lab(l,k) = \{s : l'_s = k\}$ je množina pozic a který se v labellingu l vyskytuje label k, a kde

$$Z_t = \sum_{s=1}^{|l'|} \frac{\hat{\alpha}_t(s)\hat{\beta}_t(s)}{y^t_{l'_s}}$$

CTC PAPER

Výpočet CTC může být výpočetně velice náročný. Naivním přístupem pro výpočert by bylo sečíst skóre všech zarovnání, nicméně počet zarovnání může být velmi velký. Výpočet tedy vyřešíme algortimem dynamického programování. Hlavní ideou je, že dvě zarovnání, která dosáhla stejné výstupu v daném časovém okamžice mohou být spojeny.

CTC funkce je diferencovatelná vzhledem k pravděpodobnostem v každém časosvém kroku a lze použít gradientní metody.

Lze využít BEAM search (trade-off mezi accuracy a runtime) a language model modifikací klasifikátoru

$$Y^* = \text{argmax } p(y \mid X) \cdot p(Y)^\gamma \cdot L(Y)^\beta$$

, kde $p(y \mid X)$ je podmíněná CTC pravděpodobnost, $p(Y)^\gamma$ je pravděpodobnost daná jazykovým modelem a $L(Y)^\zeta$ a je penalizace za vložení slova. Parametry γ a ζ jsou většinou získány pomocí crossvalidace.

DISTILL PUB

PREPSAT DO TVARU $\alpha_s^{(t)}$?

10 Klasifikace fonémů

11 Vyhodnocení

12 Závěr

13 Úvod

V posledních letech došlo k významnému vývoji v oblasti mobilních zařízení a s tím i k nárůstu popularity hlasového ovládání. Aplikace společnosti Google nabízejí možnost vyhledávání hlasem, Apple a Microsoft vytvořili inteligentní osobní asistentky Siri, resp. Cortana a stále častěji se hlasové ovládání začíná objevovat i v automobilovém průmyslu.

Jedním z řešených problémů hlasového ovládání je detekce klíčových frází v proudu řeči (KWS - Keyword spotting). Na pozadí zařízení nepřetržitě běží KWS systém, který naslouchá mluvenému slovu a při zaznění klíčové fráze provede určitou akci. V dnešní době se nejčastěji používají dva typy KWS systémů - systémy s předdefinovanými klíčovými frázemi a systémy využívající obsáhlý slovník řeči v textové podobě (LVCSR - Large vocabulary continuous speech recognition). LVCSR systémy jsou výpočetně velmi náročné, jelikož je potřeba nejprve převést mluvenou řeč do textové podoby a následně vyhledat klíčovou frázi v databázi. Vzhledem k tomu, že tyto systémy neumožňují vytvoření vlastní bezpečné klíčové fráze, propojují se většinou se systémem pro identifikaci řečníka [**?**].

Bylo by tedy vhodné vytvořit KWS systém, který by nebyl závislý na předem definovaných frázích a umožnil by uživateli volbu vlastních klíčových frází v libovolném jazyce. Také by měl být výpočetně nenáročný, aby byla zajištěna odezva v reálném čase na mobilních zařízeních. Tyto požadavky splňuje metoda Query-by-Example, kdy si uživatel vytvoří vzorovou klíčovou frázi a všechny budoucí promluvy jsou porovnávány vůči ní [**?**, **?**].

Práce se věnuje zpracování akustického signálu a především problematice rozpoznání izolovaných slov. Uvedeme si nejčastěji využívané algoritmy strojového učení a porovnáme jejich rozpoznávací schopnosti a jejich vhodnost potenciálního využití v KWS systému.

14 Klasifikace

Klasifikace neboli rozpoznávání je úloha, kde je cílem správně zařadit objekt do jedné z k tříd. Aby bylo možné objekt klasifikovat, je třeba ho nejprve vhodně popsat pomocí tzv. příznakového vektoru (obrazu). Příznakový vektor se skládá z příznaků, které reprezentují jednotlivé měřitelné či pozorovatelné vlastnosti objektu. Příznaky je třeba volit v závislosti na řešeném problému tak, aby dostatečně popisovaly pozorovaný objekt. Může se zdát, že vytvořením příznakového vektoru ze všech měřitelných veličin objektu získáme dokonalý popis objektu, který přispěje k přesnosti klasifikace. Opak je však pravdou, jelikož velké množství příznaků s nedostatečnou informativní hodnotou může mít negativní vliv na přesnost klasifikace a vést k přetrénování modelu [**?**].

Algoritmus, který provádí klasifikaci, se nazývá klasifikátor. Tato práce se zabývá především klasifikátory založenými na učení s učitelem a klasifikátoru podle minimální vzdálenosti k vzorovým obrazům.

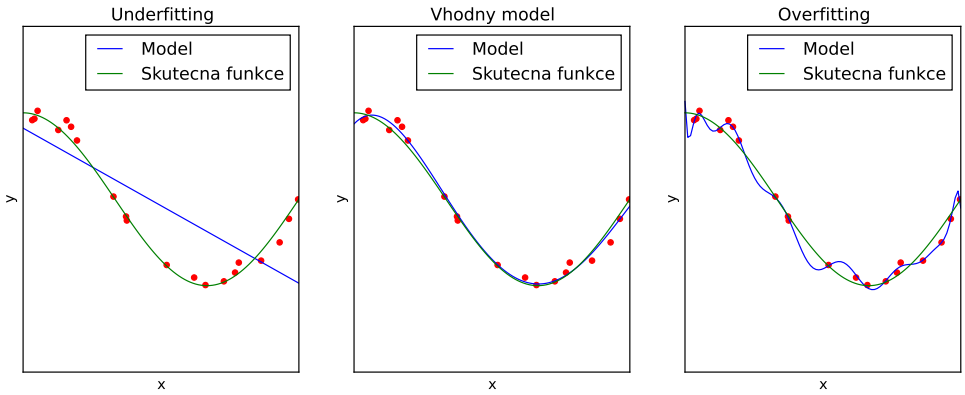
Učící se klasifikátor pracuje ve dvou fázích - trénovací fáze a klasifikační fáze:

1. **trénovací fáze** - v trénovací fází jsou klasifikátoru předkládány dvojice $\{x_i, y_i\}$, $i = 1, 2, \ldots, l$, kde x_i jsou příznakové vektory a y_i jsou cílové třídy jednotlivých obrazů z trénovací množiny. Klasifikátor vypočte svůj výstup pro příznakový vektor x_i (odhadovaná třída) a porovná jej s cílovou třídou. V případě neshody odhadované a cílové třídy upraví své parametry tak, aby minimalizoval danou chybovou funkci (jedná se tedy o optimalizační úlohu). Trénovací dvojice jsou klasifikátoru předkládány tak dlouho, dokud nedosáhne optimálního nastavení.

2. **klasifikační fáze** - klasifikátoru předkládáme neznámé obrazy a na základě natrénovaných parametrů klasifikujeme do jedné z k tříd [**?**, **?**].

Při trénování dochází velmi často k tzv. přetrénování modelu (overfitting). Klasifikátor se naučí dokonale rozpoznávat pozorování z trénovací množiny, ale ztrácí schopnost generalizace a nová, neznámá pozorování klasifikuje chybně. Overfitting je způsoben volbou příliš komplexního modelu, nedostatkem trénovacích dat nebo vysokou dimenzí obrazů. Komplexitu modelu je tedy třeba volit s ohledem na povahu klasifikovaných dat, popř. lze využít jednu z metod, která overfitting omezí (regularizace, cross-validation, dropout vrstvy u neuronových sítí, atd.) [**?**]. Opačným případem overfittingu je tzv. underfitting,

kdy model na úkor generalizace ztrácí klasifikační schopnosti. Vliv volby modelu s různým počtem stupňů volnosti můžeme vidět na obrázku 8.



Obrázek 8: Příklad overfittingu a underfittingu.

15 Zpracování akustického signálu

Základním krokem pro klasifikaci řeči je samotné zpracování akustických signálů a jejich transformace na příznakové vektory. Při zpracování akustického signálu předpokládáme, že se jeho vlastnosti mění pomalu a na krátkých úsecích je téměř stacionární (v časové i frekvenční oblasti). Signál je tedy rozdělen na kratší úseky neboli mikrosegmenty, které jsou zpracovány samostatně, jako by se jednalo o různé signály. Pro každý mikrosegment pak vypočteme číslo (příznak), popřípadě vektor čísel na základě zvolené metriky. Délka mikrosegmentů se nejčastěji volí mezi 10 až 25ms a jednotlivé mikrosegmenty na sebe mohou navazovat nebo se i překrývat. Díky tomu lze celý signál popsat posloupností čísel (příznakový vektor), jejíž délka je přímo úměrná délce slova a počtu mikrosegmentů [**?**].

15.1 Okénková funkce

Jednotlivé mikrosegmenty jsou ze signálu vybírány pomocí okénka o určité délce, které se posouvá o daný počet vzorků signálu. Okénko také slouží k přidělení vah zpracovávaným vzorkům signálu. Mezi nejčastěji používané okénkové funkce při zpracování řeči patří pravoúhlé a Hammingovo okénko:

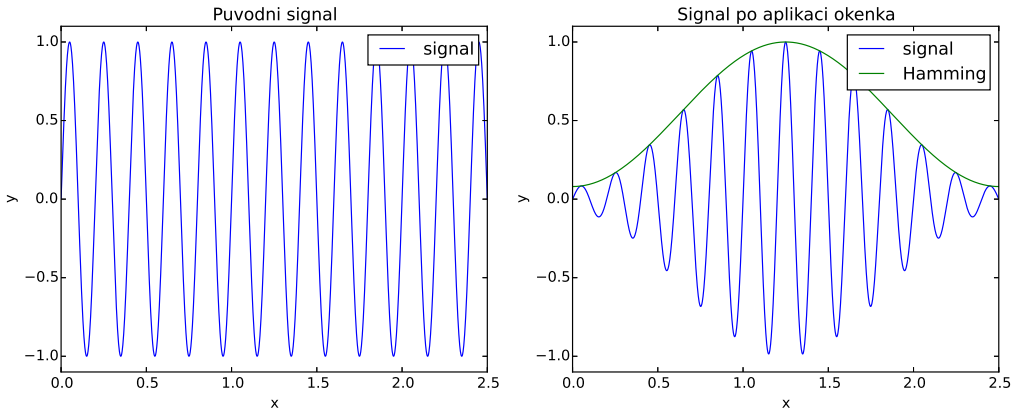
- **pravoúhlé okénko** - pravoúhlé okénko přidělí všem vzorkům mikrosegmentu stejnou váhu. Lze jej definovat vztahem

$$w(n) = \begin{cases} 1 & \text{pro } 0 \leq n \leq N-1 \\ 0 & \text{jinak,} \end{cases} \tag{15.1}$$

  kde N je počet vzorků mikrosegmentu.

- **Hammingovo okénko** - Hammingovo okénko je vhodné použít v případě, kdy je třeba potlačit vzorky na krajích zpracovávaného mikrosegmentu [**?**, **?**]. Lze jej definovat vztahem

$$w(n) = \begin{cases} 0.54 - 0.46 \cos\left(\dfrac{2\pi n}{N-1}\right) & \text{pro } 0 \leq n \leq N-1 \\ 0 & \text{jinak.} \end{cases} \tag{15.2}$$



Obrázek 9: Aplikace Hammingova okénka na funkci $f(x) = \sin(10\pi x)$.

15.2 Příznaky v časové a frekvenční oblasti

15.2.1 Krátkodobá energie signálu

Funkce krátkodobé energie signálu je definována vztahem

$$E_n = \sum_{k=-\infty}^{\infty} [s(k)w(n-k)]^2, \tag{15.3}$$

kde $s(k)$ je vzorek signálu v čase k a $w(n)$ je daný typ okénka. Hodnoty této funkce poskytují informaci o průměrné hodnotě energie v každém mikrosegmentu signálu. Hlavním nedostatkem funkce krátkodobé energie signálu je její vysoká citlivost na velké výkyvy v amplitudě signálu.

15.2.2 Krátkodobá intenzita signálu

Funkce krátkodobé intenzity signálu je definována vztahem

$$I_n = \sum_{k=-\infty}^{\infty} |s(k)|w(n-k). \tag{15.4}$$

Na rozdíl od funkce krátkodobé energie signálu není tato funkce citlivá na velké změny amplitudy signálu.

15.2.3 Krátkodobé průchody nulou

Funkce krátkodobých průchodů nulou je definována vztahem

$$Z_n = \frac{1}{2} \sum_{k=-\infty}^{\infty} |sign\left[s(k)\right] - sign\left[s(k-1)\right]| w(n-k). \tag{15.5}$$

Funkce krátkodobých průchodů nulou nese informaci o frekvenci signálu - čím více průchodů nulou, tím vyšší je v daném úseku frekvence signálu a naopak. Frekvenci průchodů signálu nulovou úrovní můžeme tedy využít jako jednoduchou charakteristiku, která popisuje spektrální vlastnosti signálu. Hodnoty této funkce se nejčastěji využívají k detekci řeči v akustickém signálu (určení začátku a konce promluvy) [**?**, **?**].

15.2.4 Mel-frekvenční kepstrální koeficienty

Nejčastěji používaným typem příznaků v rozpoznávání řeči jsou mel-frekvenční kepstrální koeficienty (MFCC). Jedná se o velice robustní typ příznaků ve frekvenční oblasti, který respektuje nelineární vlastnosti vnímání zvuků lidským uchem. Lineární frekvence f[Hz] je převedena na frekvenci f_m[mel] v nelineární melovské frekvenční škále

$$f_m = 2595 \log_{10}\left(\frac{f}{700}\right). \tag{15.6}$$

Při výpočtu MFCC se nejčastěji volí segmentace signálu na mikrosegmenty o délce 10 až 30ms, na které se aplikuje Hammingovo okénko s posunem o 10ms. Segmentovaný signál je následovně zpracován rychlou Fourierovou transformací (FFT), díky které získáme amplitudové spektrum $|S(f)|$ analyzovaného signálu. Vzhledem k použití FFT je vhodné volit počet vzorků okénka při dané frekvenci roven mocnině 2.

Dalším krokem výpočtu je melovská filtrace, při níž využijeme banku trojúhelníkových pásmových filtrů. Jednotlivé trojúhelníkové filtry jsou rozloženy přes celé frekvenční pásmo on nuly až do Nyquistovy frekvence a jejich střední frekvence jsou rovnoměrně rozloženy podél frekvenční osy v melovské škále. Střední hodnoty filtrů b_m lze vyjádřit vztahem

$$b_{m,i} = b_{m,i-1} + \Delta_m, \tag{15.7}$$

kde $i = 1, 2, \ldots, M^*$ je počet trojúhelníkových filtrů v bance, $b_{m,0} = 0$ mel a $\Delta_m = B_{m,w}/(M^* + 1)$.

Dále je třeba vypočítat odezvy všech filtrů, čehož dosáhneme jejich vyjádřením ve frekvenční škále s měřítkem v herzích za využití původních koeficientů získaných FFT. Přepočteme tedy všechny střední frekvence v melovské škále $b_{m,i}, i = 1, \ldots, M^*+1$ pomocí inverzního vztahu $f = 700[exp(0.887 \cdot 10^{-3} f_m) - 1]$ na střední frekvence $b_i, i = 1, \ldots, M^*+1$. Nyní můžeme trojúhelníkové filtry vyjádřit vztahem

$$u(f,i) = \begin{cases} \dfrac{1}{b_i - b_{i-1}}(f - b_{i-1}) & \text{pro } b_{i-1} \leq f < b_i \\ \dfrac{1}{b_i - b_{i+1}}(f - b_{i+1}) & \text{pro } b_i \leq f < b_{i+1} \\ 0 & \text{jinak} \end{cases} \tag{15.8}$$

a odezvy jednotlivých filtrů $y_m(i)$ vztahem

$$y_m(i) = \sum_{f=b_{i-1}}^{b_{i+1}} |S(f)| u(f,i), \tag{15.9}$$

kde $i = 1, 2, \ldots, M^*$ a f jsou frekvence využité při výpočtu FFT. Při průchodu signálu filtrem je každý koeficient FFT násoben odpovídajícím ziskem filtru a výsledky pro jednotlivé filtry jsou akumulovány a následně zlogaritmovány. Tím dojde k dekorelaci energií filtrů a získané hodnoty lze použít jako plnohodnotné příznaky.

Nakonec provedeme zpětnou diskrétní kosinovou transformaci (DCT). Značnou výhodou DCT je vysoká nekorelovanost vzniklých koeficientů. Koeficienty jsou dány vztahem

$$c_m(j) = \sum_{i=1}^{M^*} \log y_m(i) \cos\left(\frac{\pi j}{M^*}(i - 0.5)\right) \quad \text{pro } j = 0, 1, \ldots, M, \tag{15.10}$$

kde M je počet mel-frekvenčních kepstrálních koeficientů [**?**, **?**].

15.3 Delta a delta-delta koeficienty

Při výpočtu příznakových vektorů předpokládáme, že signál je na krátkém úseku stacionární. Je tedy zřejmé, že tyto příznakové vektory budou popisovat pouze statické vlastnosti signálu a dynamické vlastnosti se ztrácí. Tento nedostatek můžeme vyřešit rozšířením příznakových vektorů o dynamické koeficienty.

Využívají se delta (diferenční) koeficienty a delta-delta (akcelerační) koeficienty, které odpovídají první, respektive druhé derivaci příznakového vektoru. Delta koeficienty lze

definovat vztahem

$$d_t = \frac{\sum_{n=1}^{N} n(c_{t+n} - c_{t-n})}{2\sum_{n=1}^{N} n^2}, \tag{15.11}$$

kde d_t je delta koeficient v čase t a N je volitelný parametr. Tento parametr se většinou volí $N = 1$ nebo $N = 2$ a určuje, z kolika sousedních mikrosegmentů budou vypočítány delta koeficienty (jeden, resp. dva leví a praví sousedé mikrosegmentu). Delta-delta koeficienty lze spočítat využitím stejného vztahu s tím rozdílem, že se nepočítají ze statických koeficientů, ale z delta koeficientů [**?**, **?**].

15.4 Normalizace příznaků

Jednotlivé příznaky se mohou pohybovat na různých definičních oborech hodnot a při klasifikačních úlohách se mohou projevit s různou váhou. Proto je vhodné příznaky transformovat na stejný definiční obor hodnot, popř. do rozdělení o stejných parametrech, a tím zaručit, že všechny příznaky budou mít stejný vliv. Tuto transformaci nazýváme normalizací dat a v praxi se nejčastěji používají následující dvě metody:

- **Z-score normalizace** - tato metoda se používá v případě, kdy data odpovídají normálnímu rozložení. Z-score normalizací přetransformujeme vstupní data x, na data z, které odpovídají normálnímu rozdělení o střední hodnotě $\mu = 0$ s rozptylem $\sigma = 1$.

$$z = \frac{x - \mu}{\sigma} \tag{15.12}$$

- **Min-Max normalizace** - Min-Max normalizace přeškáluje data do definovaného intervalu (nejčastěji $[0, 1]$ nebo $[-1, 1]$, popř. $[0, 255]$ při zpracování obrazu). Tím dosáhneme menšího rozptylu a eliminujeme vliv odlehlých bodů (tzv. outliers).

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{15.13}$$

Některé klasifikační algoritmy přímo vyžadují, aby data byla normalizována. Příkladem mohou být algoritmy založené na gradientních metodách (logistická regrese, SVM, neuronové sítě, atd.), kdy jsou změny parametrů modelu při trénování přímo závislé na vstupním příznakovém vektoru. Při velkých rozdílech definičních oborů jednotlivých příznaků se tedy některé parametry mohou měnit výrazně rychleji než ostatní. Normalizací dat zajistíme rychlejší a stabilnější konvergenci parametrů. Dalším příkladem mohou být shlu-

kovací algoritmy pracující s Euklidovou vzdáleností, kde je vhodné, aby všechny příznaky měly na shlukování stejný vliv [**?**].

16 Support Vector Machine

Předpokládejme, že máme trénovací množinu $\{x_i, y_i\}$, $i = 1, 2, \ldots, l$, $y_i \in \{-1, 1\}$, která je složena z příznakových vektorů x_i a odpovídajících cílových tříd y_i (binární klasifikace). Pro tuto množinu si odvodíme lineární klasifikátor založený na podpůrných vektorech pro separabilní a neseparabilní případ a nelineární klasifikátor. Nakonec si uvedeme metody, jak lze klasifikovat do více tříd [**?**, **?**].

16.1 Nadrovina

Pro odvození klasifikátoru SVM (a později i pro odvození neuronové sítě) je vhodné nejprve zavést pojem nadrovina. Předpokládejme tedy, že máme p-dimenzionální prostor. Nadrovinou pak rozumíme libovolný afinní podprostor o dimenzi $p - 1$. Například ve dvoudimenzionálním prostoru je nadrovinou jednodimenzionální podprostor - přímka, ve třídimenzionálním prostoru je nadrovinou plocha. Nadrovinu o dimenzi p lze definovat následovně

$$b + w_1 x_1 + w_2 x_2 + \ldots w_p x_p = 0, \tag{16.1}$$

kde $x = (x_1, x_2, \ldots, x_p)^T$ je bod v p-dimenzionálním prostoru vyhovující rovnici. Bod x tedy leží na nadrovině.

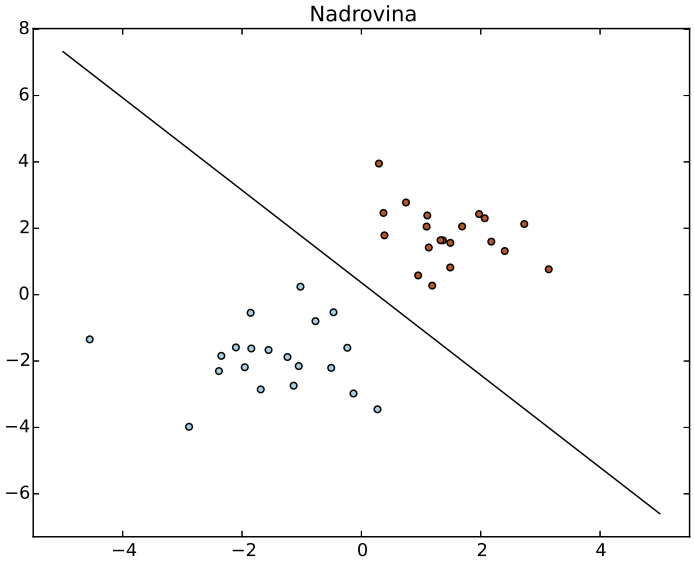
V případě, že bod x nevyhovuje rovnici 16.1, tedy

$$b + w_1 x_1 + w_2 x_2 + \ldots w_p x_p > 0, \tag{16.2}$$

bod leží na jedné straně poloroviny, resp. pokud

$$b + w_1 x_1 + w_2 x_2 + \ldots w_p x_p < 0, \tag{16.3}$$

bod leží na druhé straně poloroviny. Nadrovina tedy dělí p-dimenzionální prostor na dvě poloviny, tzv. poloprostory. Náležitost bodu do těchto poloprostorů pak můžeme zjistit jednoduchým výpočtem znaménka levé strany rovnice 16.1 [**?**].



Obrázek 10: Nadrovina oddělující body ve dvoudimenzionálním prostoru [**?**].

16.2 Lineárně separabilní případ

Předpokládejme, že existuje nadrovina, která oddělí body trénovací množiny jednotlivých tříd od sebe (oddělující nadrovina). Body x, které leží na této rovině splňují rovnici $w \cdot x + b = 0$, kde w je normála nadroviny. Dále si definujme kolmou vzdálenost oddělující nadroviny k počátku jako $\frac{|b|}{||w||}$, kde $||w||$ je Euklidovská norma w.

Dále si zaveďme nejkratší vzdálenost d_+ (d_-) mezi oddělující rovinou a pozitivním (negativním) příkladem a tzv. margin (odstup) jako $d_+ + d_-$. Pro lineárně separabilní případ hledá klasifikátor takovou nadrovinu, pro kterou nabývá margin nejvyšší hodnoty. Všechny body trénovací množiny tedy splňují tyto podmínky

$$x_i \cdot w + b \geq +1 \quad \text{pro } y_i = +1, \tag{16.4}$$

$$x_i \cdot w + b \leq -1 \quad \text{pro } y_i = -1, \tag{16.5}$$

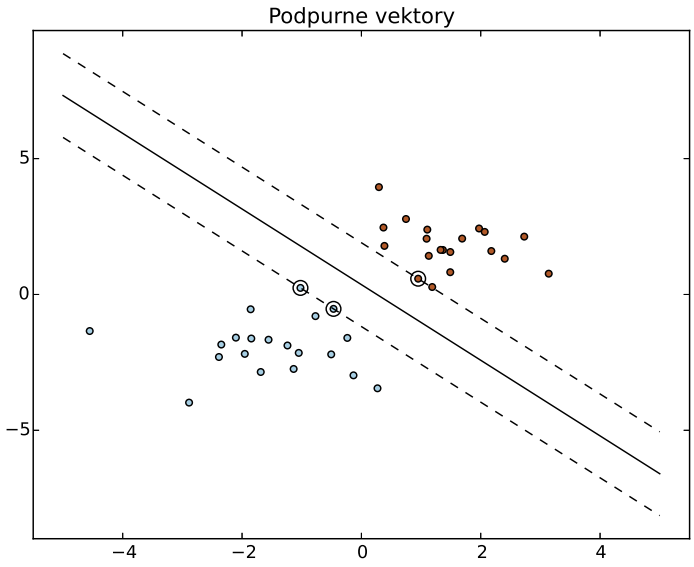
které lze sloučit do jedné množiny nerovností

$$y_i(x_i \cdot w + b) - 1 \geq 0 \quad \forall i. \tag{16.6}$$

Body, pro které v nerovnici 16.4 platí rovnost, leží na nadrovině $H_1 : x_i \cdot w + b = +1$ a jejich kolmou vzdálenost vůči počátku lze vyjádřit jako $\frac{|1-b|}{||w||}$. Obdobně body, pro které

v nerovnici 16.5 platí rovnost, leží na nadrovině $H_2 : x_i \cdot w + b = -1$ v kolmé vzdálenosti od počátku $\frac{|-1-b|}{||w||}$. Z toho plyne, že $d_+ = d_- = \frac{1}{||w||}$ a margin je roven $\frac{2}{||w||}$. Nadroviny H_1 a H_2 mají stejnou normálu w (jsou rovnoběžné) a nenachází se mezi nimi žádný bod z trénovací množiny. Nalezení takových nadrovin, které maximalizují margin, provedeme minimalizací $||w||^2$ za respektování omezujících podmínek.

Body ležící na nadrovinách H_1 a H_2 se nazývají podpůrné vektory (support vectors, zvýrazněné body na obrázku 11) a oddělující rovina je na nich přímo závislá. Pokud dojde k jejich pohybu nebo odstranění, změní se i výsledné řešení.



Obrázek 11: Vizualizace podpůrných vektorů [**?**].

Formulujme si nyní tento problém pomocí Lagrangeových multiplikátorů α_i, $i = 1, 2, \ldots, l$ - jeden pro každou nerovnost 16.6, neboli jeden pro každý prvek trénovací množiny $\{x_i, y_i\}$. Dostáváme Lagrangeovu funkci

$$L_P \equiv \frac{1}{2}||w||^2 - \sum_{i=1}^{l} \alpha_i y_i (x_i \cdot w + b) + \sum_{i=1}^{l} \alpha_i. \tag{16.7}$$

Nyní stačí minimalizovat L_P podle w a b a zároveň požadujeme, aby $\frac{\partial L_P}{\partial \alpha_i} = 0$, $\alpha_i \geq 0$ (označme si tuto množinu omezujících podmínek $\mathcal{C}_1$). Vzhledem k tomu, že se jedná o úlohu konvexního kvadratického programování, můžeme ekvivalentně řešit duální problém: maximalizace L_P za omezujících podmínek $\frac{\partial L_P}{\partial w} = 0$ a $\frac{\partial L_P}{\partial b} = 0$ a zároveň $\alpha_i \geq 0$ (označme tuto množinu omezujících podmínek jako $\mathcal{C}_2$). Tato duální formulace problému má tu

vlastnost, že maximum L_P za podmínek $\mathcal{C}_2$ nastává při stejných hodnotách w, b a α jako minimum L_P za podmínek $\mathcal{C}_1$.

Omezeními gradientu $\frac{\partial L_P}{\partial w} = 0$ a $\frac{\partial L_P}{\partial b} = 0$ získáme rovnice

$$w = \sum_i \alpha_i y_i x_i, \tag{16.8}$$

$$\sum_i \alpha_i y_i = 0. \tag{16.9}$$

Dosazením rovnic 16.8 a 16.9 do 16.7 dostaneme

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j. \tag{16.10}$$

Po vyřešení optimalizační úlohy (trénovací fáze) můžeme klasifikovat libovolný vektor x na základě toho, na které straně oddělující nadroviny leží. Třída vektoru x tedy bude $\operatorname{sgn}(w \cdot x + b)$ [**?**].

16.3 Lineárně neseparabilní případ

Pokud použijeme výše zmíněný algoritmus na lineárně neseparabilní data, nenalezneme žádné přijatelné řešení, jelikož duální Langrangeova funkce L_D poroste nade všechny meze. Chceme-li algoritmus rozšířit i pro neseparabilní data, musíme uvolnit podmínky 16.4 a 16.5. Toho dosáhneme zavedením volných (slack) proměnných ε_i, $i = 1, 2, \ldots, l$

$$x_i \cdot w + b \geq +1 - \varepsilon_i \quad \text{pro } y_i = +1 \tag{16.11}$$

$$x_i \cdot w + b \leq -1 + \varepsilon_i \quad \text{pro } y_i = -1 \tag{16.12}$$

$$\varepsilon_i \geq 0 \quad \forall i. \tag{16.13}$$

Chyba klasifikace tedy nastane v případě, že $\varepsilon > 1$. Horní mez počtu chyb klasifikace prvků trénovací množiny je tedy $\sum_i \varepsilon_i$. Vhodným způsobem, jak navýšit hodnotu kriteriální funkce za každou chybu, je minimalizovat $\frac{||w||^2}{2} + C(\sum_i \varepsilon_i)^k$ místo původního kritéria $\frac{||w||^2}{2}$. Parametr C se volí a odpovídá penalizaci za chybnou klasifikaci - čím vyšší hodnota C, tím větší penalizace. Pro $k > 0$, $k \in \mathbb{Z}$ se jedná o úlohu konvexního programování, pro $k = 2$ a $k = 1$ se jedná přímo o úlohu kvadratického programování. Volbou $k = 1$ navíc

zajistíme, že ε_i ani jejich multiplikátory se neobjeví v definici duálního problému

$$L_D \equiv \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \tag{16.14}$$

za podmínek

$$0 \leq \alpha_i \leq C, \tag{16.15}$$

$$\sum_i \alpha_i y_i = 0. \tag{16.16}$$

Řešením je pak

$$w = \sum_{i=1}^{N_S} \alpha_i y_i x_i, \tag{16.17}$$

kde N_S je počet podpůrných vektorů.

Primární úloha je dána Lagrangeovou funkcí

$$L_P = \frac{1}{2}||w||^2 + C \sum_i \varepsilon_i - \sum_i \alpha_i \{y_i(x_i \cdot w + b) - 1 + \varepsilon_i\} - \sum_i \mu_i \epsilon_i, \tag{16.18}$$

kde μ_i jsou Lagrangeovské multiplikátory zajišťující kladnost ε_i. Řešitelnost 16.18 je dána Karush-Kuhn-Tucker podmínkami (více v [**?**]). Pro výpočet prahu b postačí pouze dvě z Karush-Kuhn-Tucker podmínek

$$\alpha_i \{y_i(x_i \cdot w + b) - 1 + \varepsilon_i\} = 0, \tag{16.19}$$

$$\mu_i \epsilon_i = 0. \tag{16.20}$$

Ačkoliv k výpočtu prahu b stačí znát pouze jeden prvek trénovací množiny splňující podmínku $0 < \alpha_i < C$ a zároveň $\varepsilon_i = 0$, z numerického hlediska je rozumnější určit výsledný práh jako průměr prahů přes všechny prvky trénovací množiny [**?**].

16.4 Nelineárně separabilní případ

Uvažujme nyní případ, kdy oddělující nadrovina není lineární funkcí trénovacích dat - potřebujeme tedy zobecnit rovnice 16.14, 16.15 a 16.16 pro nelineární oddělující nadrovinu. Všimněme si, že v těchto rovnicích se data trénovací množiny vyskytují vždy jako skalární součin $x_i \cdot x_j$. Předpokládejme, že existuje zobrazení Φ z n-dimenzionálního pro-

storu do jiného Euklidovského prostoru $\mathcal{H}$ o vyšší dimenzi (až nekonečnědimenzionálního)

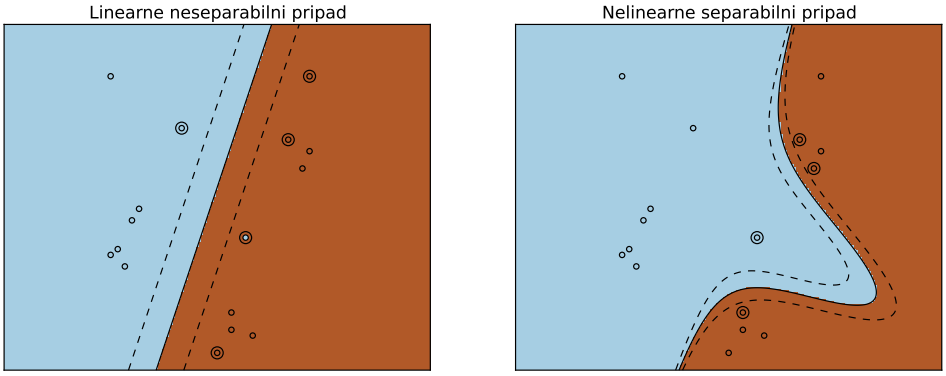
$$\Phi : \mathbb{R}^n \mapsto \mathcal{H}. \tag{16.21}$$

Potom by trénovací algoritmus závisel pouze na skalárních součinech $\Phi(x_i) \cdot \Phi(x_j)$ v prostoru $\mathcal{H}$. Definujme si tedy jádrovou funkci (kernel function) $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$, kterou můžeme nahradit skalární součin $x_i \cdot x_j$ v trénovacím algoritmu 16.14 a opět dopočítat normálu w oddělující nadroviny. Nemusíme tedy explicitně znát zobrazení Φ, pouze jádrovou funkci.

Klasifikaci libovolného bodu x provedeme výpočtem znaménka funkce $f(x)$

$$f(x) = \sum_{i=1}^{N_S} \alpha_i y_i \Phi(s_i) \cdot \Phi(x) + b = \sum_{i=1}^{N_S} \alpha_i y_i K(s_i, x) + b, \tag{16.22}$$

kde s_i jsou podpůrné vektory [**?**, **?**].



Obrázek 12: SVM s lineární a polynomiální jádrovou funkcí [**?**].

16.5 Klasifikace do více tříd

Zatím jsme se zabývali pouze klasifikaci do dvou tříd neboli binární klasifikací. Nyní uvažujme trénovací množinu $\{x_i, y_i\}$, $i = 1, 2, \ldots, l$, $y_i \in \{1, 2, \ldots, k\}$, kde $k > 2$ je počet cílových tříd. Uveďme si dva základní přístupy, jak do těchto tříd klasifikovat:

- **One-Versus-One** - natrénujeme $\frac{k(k-1)}{2}$ binárních klasifikátorů, kde každý z nich porovnává dvě různé třídy navzájem. Klasifikace testovacího vektoru je pak založena na hlasování, kdy každý klasifikátor hlasuje pro jednu třídu a testovací vektor je zařazen do té třídy, která dostala nejvíce hlasů.

- **One-Versus-All** - natrénujeme k binárních klasifikátorů, kde každý z nich porovnává jednu z k tříd oproti zbylým $k-1$ třídám. Testovací vektor je zaklasifikován do té třídy, pro kterou nabývá oddělující nadrovina 16.22 nejvyšší hodnoty (tj. bod leží nejdále od oddělující nadroviny a byl zaklasifikován s největší "jistotou") [**?**, **?**].

17 Neuronové sítě

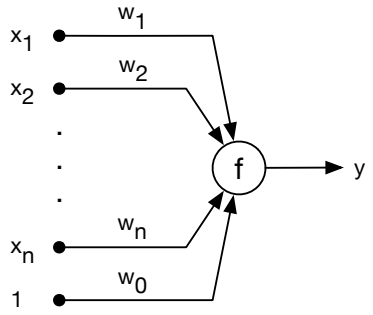
Neuronová síť je algoritmus inspirovaný funkcí neuronů a jejich propojením v lidském mozku. Skládá se z mnoha výpočetních jednotek (neurony) propojených pomocí numerických parametrů (synapse) a úpravou těchto parametrů je schopna se učit. Tato práce je omezena pouze na dopředné neuronové sítě a jejich trénování pomocí učení s učitelem.

17.1 Perceptron a aktivační funkce

Základní výpočetní jednotkou neuronových sítí je tzv. perceptron. Výstupní hodnota perceptronu je dána vztahem

$$y = f\left(\sum_{i=1}^{n} w_i x_i + w_0\right) = f(w^T x + w_0), \tag{17.1}$$

kde $w = [w_1, w_2, \ldots, w_n]^T$ je váhový vektor, $x = [x_1, x_2, \ldots, x_n]^T$ je vstupní vektor, w_0 je práh a $f(\cdot)$ je aktivační funkce. Práh reprezentuje váhu vedoucí od jednotkového vstupního bodu, který se zavádí z důvodu generalizace sítě.



Obrázek 13: Perceptron.

Pro zjednodušení následujících odvození si rozšiřme váhový vektor w o práh w_0 a vstupní vektor o jednotkový vstupní bod

$$w = [w_0, w_1, \ldots, w_n]^T, \tag{17.2}$$

$$x = [1, x_1, \ldots, x_n]^T. \tag{17.3}$$

Uveďme si také příklady nejznámějších aktivačních funkcí:

- **Sigmoidální aktivační funkce** - sigmoidální aktivační funkce transformuje reálné

číslo do intervalu $(0, 1)$. Nevýhodou tohoto rozsahu je, že velmi malá čísla jsou transformována na hodnoty blízké nule, což má za následek i velice nízkou hodnotu lokálního gradientu a neuronem tak projde minimum signálu (více u algoritmu back-propagation). Při inicializaci vah vysokými hodnotami naopak může dojít k saturaci signálu a síť nebude schopná se učit. Výstupy neuronu s touto aktivační funkcí zároveň nemají střední hodnotu v nule, což má za následek, že pro kladný vstup budou mít všechny váhy vedoucí k neuronu stejné znaménko.

$$f(\xi) = \frac{1}{1 + e^{-\xi}} \tag{17.4}$$

- **Tanh** - aktivační funkce tanh transformuje reálně číslo do intervalu $(-1, 1)$. Výstupní interval má střední hodnotu v nule a řeší nedostatky sigmoidální aktivační funkce.

$$f(\xi) = \tanh(\xi) = \frac{2}{1 + e^{-2\xi}} - 1 \tag{17.5}$$

- **ReLU (Rectified Linear Unit)** - aktivační funkce ReLU je lineární aktivační funkce s prahem v hodnotě nula. Oproti výše zmíněným aktivačním funkcím výrazně urychluje konvergenci gradientu a není tak výpočetně náročná. Při nevhodně zvolené konstantě učení však muže dojít k "deaktivaci"neuronů, kdy jejich gradient poklesne na nulu a tyto neurony již nikdy nebudou aktivovány.

$$f(\xi) = \max(0, \xi) \tag{17.6}$$

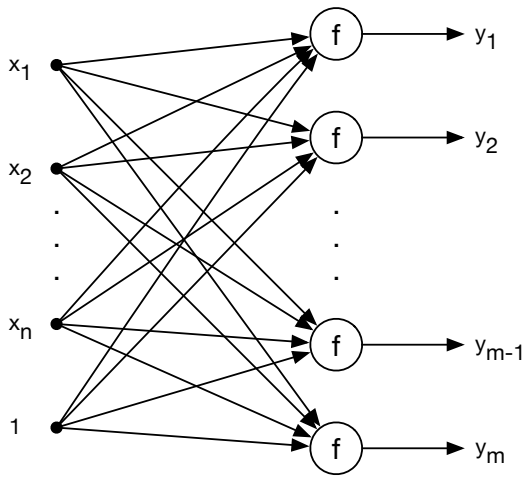
- **Maxout** - Maxout je generalizací aktivační funkce ReLU. Na rozdíl od výše zmíněných aktivačních funkcí nemá stejný funkcionální tvar $f(w^T x + b)$, ale počítá hodnotu funkce $\max(w_1^T x + b_1, w_2^T x + b_2)$. Maxout řeší nedostatky ReLU, ovšem za cenu zdvojnásobení počtu parametrů pro každý neuron [3].

Můžeme si všimnout, že argument aktivační funkce definuje nadrovinu v n-dimenzionálním prostoru. Volbou aktivační funkce

$$f(\xi) = \text{sign}(\xi) = \begin{cases} 1 & \text{pro } \xi \geq 0 \\ 0 & \text{jinak} \end{cases} \tag{17.7}$$

získáme jednoduché rozhodovací pravidlo, které přiřazuje body do poloroviny na základě znaménka argumentu aktivační funkce. Jedná se tedy o jednoduchý klasifikátor, který dokáže klasifikovat do dvou tříd (jedná se o analogii lineárně separabilního případu u klasifikátoru SVM).

Paralelním spojením více perceptronů získáme nejjednodušší typ dopředné neuronové sítě, tzv. jednovrstvou neuronovou síť [?, ?].



Obrázek 14: Jednovrstvá neuronová síť s n vstupy, aktivační funkcí $f(\cdot)$ a m výstupy.

17.2 Trénování jednovrstvé neuronové sítě

Jedním ze způsobů trénování jednovrstvé neuronové sítě je tzv. perceptronové pravidlo. Nejprve síti přidělíme náhodné váhy (většinou se volí malá čísla v okolí nuly) a poté přivádíme na vstup sítě jednotlivé příznakové vektory z trénovací množiny. V případě, že síť zaklasifikuje bod chybně, dojde k úpravě hodnot vah. Tento proces probíhá tak dlouho, dokud nejsou všechny body zaklasifikovány správně.

Jednotlivé váhy w_i vedoucí od i-tého vstupu x_i jsou modifikovány pomocí perceptronového pravidla

$$w_i(k+1) = w_i(k) + \Delta w_i(k) = w_i(k) + \alpha(t-y)x_i, \tag{17.8}$$

kde k je iterace učícího algoritmu, t je očekávaný výstup neuronu, y je skutečný výstup neuronu a $\alpha \in \mathbb{R}^+$ je konstanta učení. V případě, že je vhodně zvolena konstanta učení α a data jsou lineárně separabilní, algoritmus učení dokonverguje k optimálnímu nastavení

vah.

Druhým způsobem učení je tzv. delta pravidlo, které zajišťuje konvergenci i pro lineárně neseparabilní data. Pro zavedení delta pravidla si nejprve definujme trénovací chybu

$$E(w) = \frac{1}{2}\sum_{i=0}^{n}(t_i - y_i)^2, \tag{17.9}$$

kde n je počet prvků trénovací množiny.

Cílem učení je tuto chybu minimalizovat, čehož dosáhneme pomocí gradientního sestupu (gradient descent). Jedná se o iterativní algoritmus, který opět začíná s náhodně inicializovanými hodnotami vah a v každém kroku je upraví ve směru největšího sestupu gradientu. Tento proces probíhá tak dlouho, dokud není nalezeno globální minimum chybové funkce. Modifikace vah je tedy závislá na gradientu $E(w)$ podle w

$$\nabla E = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right]. \tag{17.10}$$

Gradient $\nabla E(w)$ udává směr největšího růstu chybové funkce - jeho zápornou hodnotou tedy získáme směr největšího poklesu a trénovací pravidlo v maticovém tvaru bude

$$w(k+1) = w(k) + \Delta w(k) = w(k) - \alpha \nabla E(w) = w(k) - \alpha(t-y)x^T. \tag{17.11}$$

Vzhledem k tomu, že daná chybová funkce má pouze jedno globální minimum, gradientní sestup při vhodně zvolené konstantě učení vždy dokonverguje k takovému váhovému vektoru, který zajišťuje minimální chybu [**?**].

17.3 Vícevrstvá dopředná neuronová síť

Jak již bylo zmíněno, jednovrstvé neuronové sítě umožňují vyjádřit pouze lineární rozhodovací hranici. Nelineární hranici můžeme vyjádřit pomocí vícevrstvé sítě, která se skládá z jedné vstupní vrstvy, jedné nebo více skrytých vrstev a výstupní vrstvy. Každá skrytá vrstva se skládá z libovolného počtu neuronů a prahové jednotky a jednotlivé skryté vrstvy mohou mít různé aktivační funkce.

Pro zjednodušení následujících odvození předpokládejme síť se vstupní vrstvou s I vstupními jednotkami, skrytou vrstvu s J neurony a výstupní vrstvu s K výstupními jednotkami. Neurony ve skryté vrstvě mají aktivační funkci $f(\cdot)$ a výstupní vrstva má ak-

tivační funkci $g(\cdot)$ (znázorněno na obrázku 15). Opět rozšíříme váhovou matici w o práh w_0 vedoucí k jednotkovému vstupnímu bodu x_0, kterým rozšíříme vstupní vektor x. Výstup neuronu j ve skryté vrstvě tedy můžeme zapsat jako

$$net_j = \sum_{i=1}^{I} x_i w_{ji} + x_0 w_{j0} = \sum_{i=0}^{I} x_i w_{ji} = w_j^T x, \tag{17.12}$$

$$z_j = f(net_j), \tag{17.13}$$

kde w_{ji} značí váhu mezi j-tým neuronem skryté vrstvy a i-tou vstupní jednotkou. Obdobně lze vypočítat výstup k-té výstupní jednotky

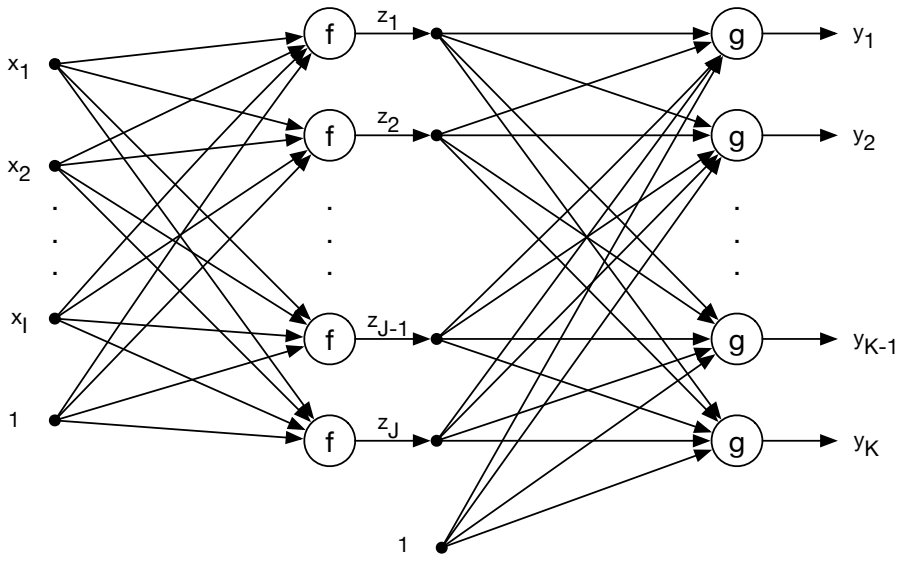
$$net_k = \sum_{j=1}^{J} z_j w_{kj} + z_0 w_{k0} = \sum_{j=0}^{J} z_j w_{kj} = w_k^T z, \tag{17.14}$$

$$y_k = g\left(net_k\right). \tag{17.15}$$

Úpravou lze k-tý výstup zapsat jako

$$y_k = g\left(\sum_{j=1}^{J} w_{kj} f\left(\sum_{i=1}^{I} x_i w_{ji} + x_0 w_{j0}\right) + w_{k0}\right). \tag{17.16}$$

Obdobným způsobem lze vyjádřit výstup pro neuronovou síť s libovolným počtem skrytých vrstev.



Obrázek 15: Dvouvrstvá neuronová síť.

17.4 Trénování vícevrstvé neuronové sítě

Pro trénování vícevrstvých neuronových sítí se využívá algoritmus backpropagation neboli algoritmus zpětného šíření chyby, který je stejně jako delta pravidlo založen na minimalizaci chybové funkce pomocí gradientu. Definujme si tedy trénovací chybu E jako kvadrát sumy rozdílů mezi skutečným výstupem k-té výstupní jednotky y_k a očekávaným výstupem t_k

$$E(w) = \frac{1}{2}\sum_{k=1}^{K}(t_k - y_t)^2 = \frac{1}{2}(t-y)^2. \tag{17.17}$$

Algoritmus backpropagation, stejně jako delta pravidlo, vychází z algoritmu gradientního sestupu. Jednotlivé váhy jsou inicializovány náhodnými malými čísly a jsou modifikovány ve směru největšího poklesu chybové funkce

$$\Delta w = -\alpha\frac{\partial E}{\partial w}, \tag{17.18}$$

čímž opět získáme pravidlo pro modifikaci vah

$$w(k+1) = w(k) + \Delta w(k). \tag{17.19}$$

Nejprve si odvoďme pravidlo pro modifikaci vah mezi skrytou a výstupní vrstvou. Vzhledem k tomu, že chybová funkce není přímo závislá na w_{jk}, musíme použít řetězové pravidlo.

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial w_{jk}} = \delta_k\frac{\partial net_k}{\partial w_{jk}} \quad \Rightarrow \quad \delta_k = -\frac{\partial E}{\partial net_k} \tag{17.20}$$

Hodnotu δ_k nazýváme citlivost neuronu a popisuje, jak se změní celková chyba při jeho aktivaci. Derivací rovnice 17.17 získáme hledané pravidlo

$$\Delta w_{jk} = \alpha\delta_k z_j = \alpha(t_k - y_k)f'(net_k)z_j. \tag{17.21}$$

Obdobným způsobem vyjádříme pravidlo pro modifikaci vah mezi vstupní a skrytou vrstvou

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} \quad \Rightarrow \quad \delta_j = f'(net_j)\sum_{k=1}^{K} w_{kj}\delta_k, \tag{17.22}$$

$$\Delta w_{ji} = \alpha x_i\delta_j = \alpha x_i f'(net_j\sum_{k=1}^{K} w_{kj}\delta_k). \tag{17.23}$$

Podrobnější odvození algoritmu backpropagation lze nalézt v [**?**, **?**].

17.4.1 Momentum

Chybová funkce vícevrstvé sítě může mít více lokálních minim a na rozdíl od chybové funkce jednovrstvé sítě (parabolická funkce s jedním globálním minimem) gradientní sestup nezaručuje nalezení globálního minima, ale pouze lokálního minima. Z tohoto důvodu se zavádí tzv. momentum (setrvačnost), které zabraňuje uváznutí učícího algoritmu v mělkém lokálním minimu a urychluje konvergenci na plochých částech povrchu chybové funkce. Přidáním momentového členu získáme následující pravidlo pro úpravu vah

$$\Delta w_{ji}(k) = \alpha x_i \delta_j + \mu \Delta w_{ji}(k-1), \tag{17.24}$$

kde $0 < \mu < 1$ je momentum a k je iterace učícího algoritmu. Změna vah tedy závisí i na změně vah v minulé iteraci [**?**, **?**].

17.4.2 Sekvenční a dávkové učení

Jak již bylo zmíněno, při trénování s učitelem jsou neuronové síti předkládány obrazy z trénovací množiny, která se pak snaží minimalizovat celkovou chybu mezi výstupy sítě a očekávanými hodnotami. V každém trénovacím cyklu algoritmu jsou síti předloženy všechny obrazy z trénovací množiny. V praxi se nejčastěji používají dva typy trénování neuronových sítí:

- **sekvenční učení** - neuronové síti jsou postupně předkládány jednotlivé obrazy z trénovací množiny (většinou v náhodném pořadí), pro každý obraz je spočtena chyba klasifikace a následně jsou upraveny parametry sítě.
- **dávkové učení** - neuronové síti jsou postupně předkládány jednotlivé obrazy z trénovací množiny, jednotlivé chyby klasifikace jsou akumulovány a k úpravě parametrů sítě dojde až na konci trénovacího cyklu s ohledem na celkovou chybu klasifikace [**?**].

18 Dynamic Time Warping

Rozpoznávání řeči je velmi obtížná úloha, jelikož žádné dvě promluvy nejsou stejné. Hlasy různých osob se liší a stejně tak se liší i jejich artikulace nebo tempo a barva řeči. Ani promluvy jedné osoby nejsou stejné - jedna promluva může být pronesena potichu, druhá nahlas nebo šeptem, mohou být proneseny různě rychle nebo např. pod vlivem nachlazení. Na akustickém signálu se dále projevuje přítomnost šumu a rušení na pozadí [**?**].

Jedním z řešení tohoto problému je využití klasifikátoru podle minimální vzdálenosti k vzorovým obrazům. Při klasifikaci se slovo zpracovává jako celek a je zařazeno do té třídy, k jejímuž vzorovému obrazu má nejmenší vzdálenost. Základním problémem je určení této vzdálenosti, jelikož obrazy mají různé délky v závislosti na délce signálu. Odlišnosti mezi podobnými signály tedy nejsou ve spektrální oblasti, ale v časové oblasti. K určení vzdálenosti mezi dvěma signály se tedy využívá algoritmus Dynamic Time Warping (DTW), neboli nelineární "borcení"časové osy, který je založen na metodě dynamického programování. "Borcením"časové osy obrazu jedné z nahrávek dojde k maximalizaci shody mezi nahrávkami.

18.1 Základní algoritmus

Předpokládejme, že máme dvě nahrávky, které jsou reprezentovány svými obrazy. Označme obraz testovaného slova

$$A = \{a_1, a_2, \ldots, a_n, \ldots, a_I\} \tag{18.1}$$

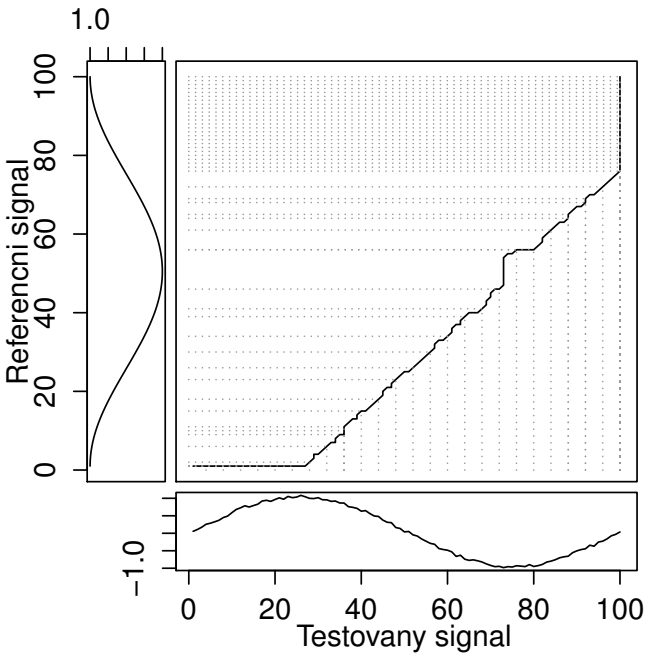
a obraz referenčního slova

$$B = \{b_1, b_2, \ldots, b_m, \ldots, b_J\}\,, \tag{18.2}$$

kde a_n je n-tý příznak testovaného slova a b_m je m-tý příznak referenčního slova. Algoritmus DTW pak hledá v rovině (n,m) optimální cestu $m = \Psi(n)$, která minimalizuje vzdálenost mezi obrazy A a B

$$D(A,B) = \sum_{n=1}^{I} \hat{d}\left(a_n, b_{\Psi(n)}\right), \tag{18.3}$$

kde $\hat{d}(a_n, b_{\Psi(n)})$ je vzdálenost mezi n-tým prvkem testovaného obrazu a m-tým prvkem referenčního obrazu.



Obrázek 16: Průběh funkce DTW pro $sin(x)$ se šumem a $cos(x)$.

18.2 Omezení pohybu funkce

Optimální cesta by měla zachovávat základní vlastnosti časových os obou obrazů (souvislost, monotónnost, atd.). Z toho důvodu se zavádí omezení na pohyb funkce DTW. Zavedeme obecnou časovou proměnnou k a časové proměnné m a n vyjádříme jako funkce k

$$n = i(k), \tag{18.4}$$

$$m = j(k), \tag{18.5}$$

kde $k = 1, 2, \ldots, K$ a K je délka obecné časové osy.

18.2.1 Omezení na hraniční body

Hraniční body funkce DTW jsou dány podmínkami

$$i(1) = 1 \quad i(K) = I, \tag{18.6}$$

$$j(1) = 1 \quad j(K) = J. \tag{18.7}$$

18.2.2 Omezení na lokální souvislost

Při průchodu funkce DTW může dojít k nadměrné expanzi či kompresi časové osy. Proto je vhodné omezit lokální monotónnost a souvislost DTW funkce

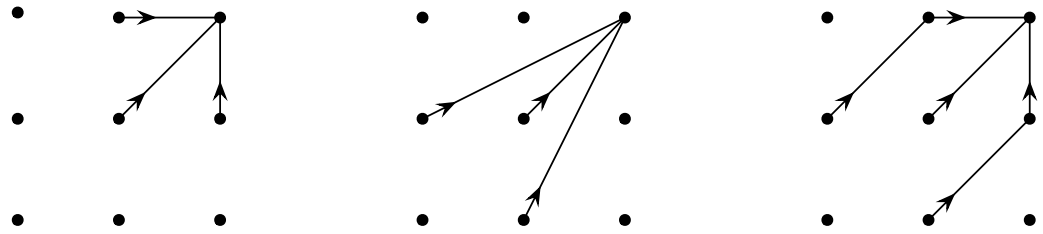
$$0 \leq i(k) - i(k-1) \leq \bar{I}, \tag{18.8}$$

$$0 \leq j(k) - j(k-1) \leq \bar{J}, \tag{18.9}$$

přičemž $\bar{I}$ a $\bar{J}$ jsou volitelné konstanty. Nejčastěji volíme hodnoty $\bar{I}, \bar{J} = 1, 2, 3$ s tím, že při hodnotě větší než 1 může funkce DTW při porovnávání některé mikrosegmenty přeskočit.

18.2.3 Omezení na lokální strmost

Pro funkci není vhodný příliš velký, ani příliš malý přírůstek, a tak se zavádí omezení na lokální strmost. Pokud se zastupující bod $[i(k), j(k)]$ pohybuje ve směru jedné osy $\bar{n}$-krát po sobě při rostoucím k, pak se v tomto směru nesmí nadále pohybovat, dokud neudělá $\bar{m}$ kroků v jiném směru.



Obrázek 17: Nejčastěji používaná lokální omezení (více v [**?**, **?**]).

18.2.4 Globální vymezení oblasti pohybu funkce

Splněním podmínek pro omezení na hraniční body a zobecněním podmínek omezení na lokální strmost na celou rovinu (n, m) lze vymezit přípustnou oblast průchodu funkce DTW

$$1 + \alpha\left[i(k) - 1\right] \leq j(k) \leq 1 + \beta\left[i(k) - 1\right], \tag{18.10}$$

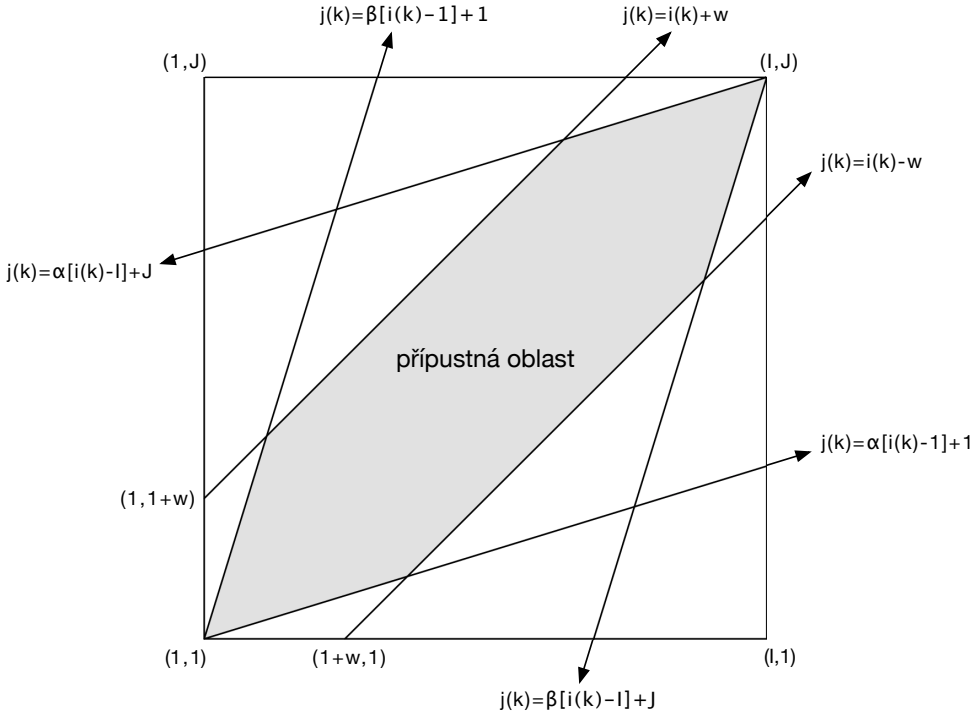
$$J + \beta\left[i(k) - I\right] \leq j(k) \leq J + \alpha\left[i(k) - I\right], \tag{18.11}$$

kde α je minimální směrnice a β maximální směrnice přímky vymezující přípustnou oblast.

Předpokládejme, že při porovnání testovaného a referenčního obrazu, které reprezentují stejné slovo, nemůže dojít k zásadním časovým rozdílům mezi příslušnými úseky stejných obrazů zapříčiněných kolísáním tempa řeči. S ohledem na tento předpoklad lze tedy stanovit podmínku pro druhé globální vymezení oblasti pohybu funkce DTW

$$|i(k) - j(k)| \leq w, \tag{18.12}$$

kde w je celé číslo, které určuje šířku okénka. Šířka okénka musí být menší než $|J - I|$, aby do přípustné oblasti funkce DTW bylo možné zahrnout i koncový bod (I, J).



Obrázek 18: Globální vymezení oblasti pohybu funkce.

18.3 Minimální vzdálenost

Celkovou minimální vzdálenost mezi testovacím obrazem A a referenčním obrazem B lze vyjádřit vztahem

$$D(A,B) = \min_{\{i(k),j(k),K\}} \left[\frac{\sum_{k=1}^{K} d\left[i(k),j(k)\right] \hat{W}(k)}{N(\hat{W})} \right], \tag{18.13}$$

kde $d[i(k),j(k)]$ je lokální vzdálenost mezi n-tým úsekem testovaného obrazu A a m-tým úsekem referenčního obrazu B, $\hat{W}(k)$ je hodnota váhové funkce pro k-tý úsek a $N(\hat{W})$ je normalizační faktor, jenž je funkcí váhové funkce. Váhová funkce je závislá pouze na lokální cestě funkce DTW.

Implementace váhové funkce se volí na základě zvolených lokálních omezení funkce DTW. Nejčastěji se využívá jeden z těchto čtyř typů váhových funkcí:

a) symetrická váhová funkce

$$\hat{W}(k) = \left[i(k) - i(k-1)\right] + \left[j(k) - j(k-1)\right], \tag{18.14}$$

b) asymetrická váhová funkce

b1)

$$\hat{W}(k) = i(k) - i(k-1), \tag{18.15}$$

b2)

$$\hat{W}(k) = j(k) - j(k-1), \tag{18.16}$$

c)

$$\hat{W}(k) = \min\left[i(k) - i(k-1), j(k) - j(k-1)\right], \tag{18.17}$$

d)

$$\hat{W}(k) = \max\left[i(k) - i(k-1), j(k) - j(k-1)\right], \tag{18.18}$$

přičemž $i(0) = j(0) = 0$.

18.4 Normalizační faktor

Normalizační faktor $N(\hat{W})$ kompenzuje počet kroků funkce DTW, který se pro různě dlouhé testovací a referenční sekvence může výrazně lišit. Lze jej definovat jako

$$N(\hat{W}) = \sum_{k=1}^{K} \hat{W}(k). \tag{18.19}$$

Dosazením vztahů 18.14, 18.15, 18.16 pro váhové funkce typu a) a b) získáme normalizační faktory

$$N(\hat{W}_a) = \sum_{k=1}^{K} [i(k) - i(k-1) + j(k) - j(k-1)] = \tag{18.20}$$

$$= i(K) - i(0) + j(K) - j(0) = I + J, \tag{18.21}$$

$$N(\hat{W}_{b1}) = \sum_{k=1}^{K} [i(k) - i(k-1)] = i(K) - i(0) = I, \tag{18.22}$$

$$N(\hat{W}_{b2}) = \sum_{k=1}^{K} [j(k) - j(k-1)] = j(K) - j(0) = J. \tag{18.23}$$

Ze vztahů je patrné, že pro váhové funkce typu a) a b) je hodnota normalizačního faktoru nezávislá na konkrétním průběhu funkce DTW. Pro váhové funkce typu c) a d) je hodnota normalizačního faktoru silně závislá na průběhu funkce DTW a nelze ji určit pomocí metod dynamického programování. Pro tyto případy se hodnota normalizačního faktoru volí nezávisle na průběhu funkce DTW, aby bylo možné použít rekurzivní algoritmus.

$$N(\hat{W}_c) = N(\hat{W}_d) = I \tag{18.24}$$

18.5 Rekurzivní algoritmus

Díky nezávislosti normalizačního faktoru na průběhu funkce DTW lze vztah 18.13 pro výpočet celkové minimální vzdálenosti mezi dvěma obrazy A a B zjednodušit do tvaru

$$D(A,B) = \frac{1}{N(\hat{W})} \{ \min_{\{i(k),j(k),K\}} \sum_{k=1}^{K} d\,[i(k), j(k)]\, \hat{W}(k) \} \tag{18.25}$$

Výslednou hodnotu vztahu 18.25 lze určit rekurzivně algoritmem dynamického programování, kdy zavedeme funkci g částečné akumulované vzdálenosti:

1. Inicializace

$$g\left[i(1), j(1)\right] = d\left[i(1), j(1)\right] \hat{W}(1) \tag{18.26}$$

2. Rekurze

$$g\left[i(k), j(k)\right] = \min_{\{i(k), j(k)\}} \{g\left[i(k-1), j(k-1)\right] + d\left[i(k), j(k)\right] \hat{W}(k)\} \tag{18.27}$$

3. Konečná normalizovaná vzdálenost

$$D(A, B) = \frac{1}{N(\hat{W})} g\left[i(K), j(K)\right] = \frac{1}{N(\hat{W})} g\left[I, J\right] \tag{18.28}$$

Rekurzivní vztahy pro různé typy lokálních omezení lze odvodit dosazením za $\hat{W}(k)$ [**?**,**?**].

19 Klasifikace izolovaných slov

Pro porovnání jednotlivých metod byla vytvořena datová sada obsahující 240 nahrávek od šesti řečníků. Mezi řečníky byli čtyři muži (v tabulkách s výsledky značeni čísly 1, 2, 3 a 6) a dvě ženy (značeny čísly 4, 5). Každý řečník pronesl čtyřikrát po sobě číslovky nula až devět. První dvě nahrávání proběhla v tichém prostředí, druhá dvě za mírného okolního šumu, díky čemuž lze lépe porovnat robustnost zkoumaných metod. Tato datová sada byla následně rozdělena na referenční a testovací sadu. Jako referenční nahrávky byly zvoleny číslovky nula až devět z prvního nahrávání každého řečníka, ostatní nahrávky tvoří testovací sadu.

Nahrávky z referenční sady pak byly využity jako trénovací data pro klasifikátor SVM a neuronovou síť. Pro příznaky generované neuronovou sítí byla trénovací sada rozšířena o nahrávky vytvořené při vývoji hlasového rozhraní pro Škoda Auto, kdy každý řečník třikrát pronesl povely hlasového ovládání navigace, rádia, telefonu a poté promluvy města, ulice a číslovek nula až devět.

Porovnávané metody byly implementovány v programovacím jazyce Python s využitím knihoven pro vědecké výpočty NumPy a SciPy [**?**], knihovny pro strojové učení scikit-learn [**?**] a knihovny pro neuronové sítě Lasagne [**?**].

19.1 Zpracování akustického signálu

Prvním krokem pro klasifikaci izolovaných slov je extrakce příznaků z akustického signálu. Pro tyto účely byl vytvořen samostatný modul obsahující metody pro zpracování akustického signálu a transformaci příznakových vektorů. Modul umožňuje segmentaci signálu s využitím pravoúhlého, Hammingova nebo Hanningova okénka s volitelnou délkou v milisekundách a volitelným posunem.

Z důvodu úspornějšího zápisu výsledků si zaveďme značení pro jednotlivé typy příznaků, které modul implementuje:

značení	typ příznaků
ste	krátkodobá energie
sti	krátkodobá intenzita
stzcr	krátkodobé průchody nulou
ste_sti_stzcr	kombinace příznaků krátkodobé energie, intenzity a průchodů nulou (tvoří matici, kde sloupce odpovídají jednotlivým typům příznaků v čase)
log_fb_en	logaritmované energie banky filtrů
mfcc	mel-frekvenční kepstrální koeficienty (implementace byla převzata z modulu [**?**])

Tabulka 1: Značení typů příznaků.

Po vygenerování příznakového vektoru je možné aplikovat Z-score nebo Min-Max normalizaci a dopočítat delta a delta-delta koeficienty (implementace byla převzata z modulu LibROSA [**?**]).

Vygenerované parametrizace příznaků jsou uvedeny v tabulce 2. První číslo v názvu příznaků značí délku okénka v milisekundách, druhé posun okénka v milisekundách. Využití Hammingova okénka je značeno zkratkou *ham* (pokud není uvedeno, příznak byl vygenerován za využití pravoúhlého okénka), delta a delta-delta koeficienty zkratkou *deltas* a normalizované příznaky *norm*, např.:

- log_fb_en_25_10_ham_deltas - logaritmované energie banky filtrů, segmentováno pomocí Hammingova okénka o délce 25ms s posunem 10ms, vypočteny delta a delta-delta koeficienty,
- stzcr_10_10_norm - krátkodobé průchody nulou, segmentováno pomocí pravoúhlého okénka o délce 10ms s posunem 10ms, normalizovány

ste_10_10	ste_10_10_norm
sti_10_10	sti_10_10_norm
stzcr_10_10	stzcr_10_10_norm
ste_sti_stzcr_10_10	ste_sti_stzcr_10_10_norm
log_fb_en_25_10_ham	log_fb_en_25_10_ham_norm
log_fb_en_25_10_ham_deltas	log_fb_en_25_10_ham_deltas_norm
mfcc_25_10_ham	mfcc_25_10_ham_norm
mfcc_25_10_ham_deltas	mfcc_25_10_ham_deltas_norm

Tabulka 2: Vygenerované parametrizace příznaků.

Z tabulky 2 je patrné, že všechny příznaky v časové oblasti byly vygenerovány s využitím pravoúhlého okénka o délce 10ms, které se posouvá o 10ms. Nedochází tedy k přesahu mezi jednotlivými mikrosegmenty. Příznaky ve frekvenční oblasti byly vygenerovány s využitím Hammingova okénka o délce 25ms s posunem 10ms (jednotlivé mikrosegmenty se částečně překrývají) a 512 bodové FFT. Pro výpočet logaritmovaných energií banky filtrů bylo využito 40 filtrů, pro výpočet MFCC 26 filtrů.

19.2 Dynamic Time Warping

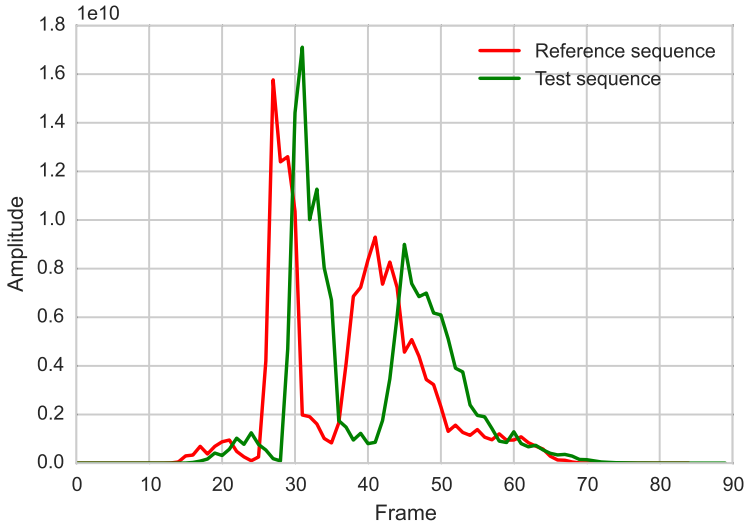
Pro výpočet DTW vzdálenosti byl vytvořen modul, který obsahuje základní DTW algoritmus bez globálního vymezení pohybu funkce a využívá symetrické omezení na lokální strmost (první zleva na obrázku 17). Jako vzdálenostní metrika pro všechny typy příznaků kromě *ste_sti_stzcr* byla zvolena Euklidova vzdálenost.

19.2.1 Optimalizace parametrů vzdálenostní metriky

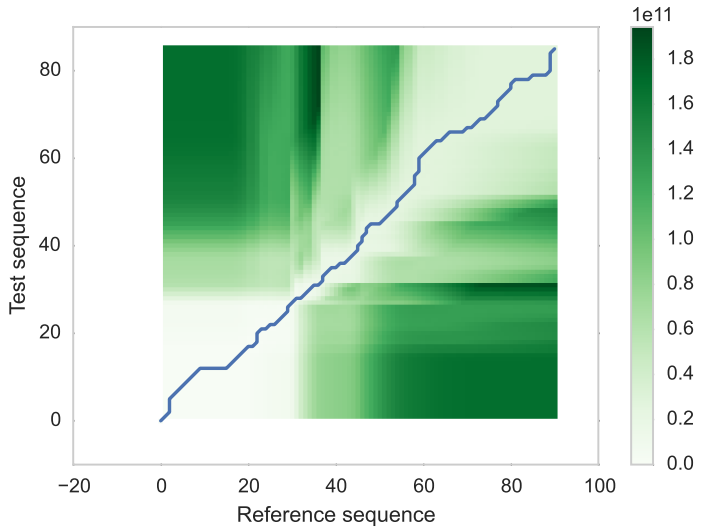
Pro kombinaci příznaků složenou z krátkodobé energie, intenzity a průchodů nulou byla využita vlastní vzdálenostní metrika

$$d(a_i, b_j) = \alpha|a_{ste,i} - b_{ste,j}| + \beta|a_{sti,i} - b_{sti,j}| + \gamma|a_{stzcr,i} - b_{stzcr,j}|, \tag{19.1}$$

kde a_i je i-tý příznak testovaného obrazu A, b_j je j-tý příznak referenčního obrazu B a α, β, γ jsou volitelné parametry.



Obrázek 19: Krátkodobé energie pro dvě různé nahrávky číslovky 7.



Obrázek 20: Průběh funkce DTW pro dva různé obrazy reprezentující číslovku 7.

Pro nenormalizovanou verzi byly zvoleny parametry $\alpha = \beta = \gamma = 1$, pro normalizovanou verzi byla provedena optimalizace parametrů, aby bylo možné určit, který z příznaků v časové oblasti má největší informativní hodnotu pro problém klasifikace izolovaných slov. Pro optimalizaci byla zvolena brute-force metoda, kdy byl procházen seznam možných parametrů s krokem 0.1 za podmínky $\alpha + \beta + \gamma = 1$. Pro každou trojici parametrů pak byla vyhodnocena přesnost klasifikace.

Nejvyšší přesnosti klasifikace v rámci jednoho řečníka bylo dosaženo s parametry $\alpha = 0.3$, $\beta = 0.3$, $\gamma = 0.4$ - každý typ příznaků se tedy projeví přibližně stejnou mírou. Optimalizací mezi všemi řečníky pak byly získány parametry $\alpha = 0$, $\beta = 0.6$, $\gamma = 0.4$ - je tedy zřejmé, že krátkodobá energie v kombinaci s krátkodobou intenzitou a průchody

nulou negativně ovlivňuje přesnost klasifikace (pravděpodobně z důvodu závislosti na řečníkovi, viz. kapitola Vyhodnocení).

Na tyto dvě parametrizace se dále budeme odkazovat jako *ste_sti_stzcr_10_10 _norm_single*, resp. *ste_sti_stzcr_10_10_norm_all*.

19.3 Support Vector Machine

Ke klasifikaci normalizovaných příznaků byl využit také SVM klasifikátor s předpočítanou jádrovou funkcí ve tvaru

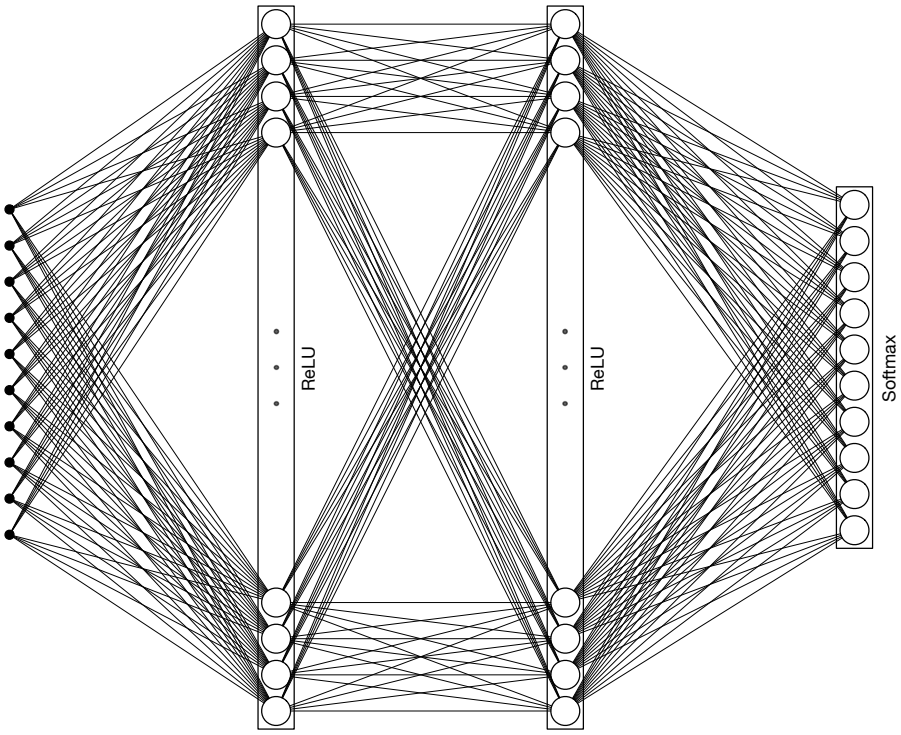
$$K(A, B) = 1 - DTW(A, B). \tag{19.2}$$

Jádrová funkce byla vypočtena ze všech referenčních obrazů navzájem a funkcionálně odpovídá Gramově matici $G = X^T X$.

19.4 Neuronová síť

Pro každého řečníka byla natrénována třívrstvá neuronová síť (znázorněna na obrázku 21). Vstupem této sítě je DTW vzdálenost testované nahrávky vůči referenčním nahrávkám, výstupem pak vektor obsahující pravděpodobnosti náležitosti do daných tříd (zajištěno aktivační funkcí Softmax). Po otestovaní několika různých parametrizací neuronové sítě byly zvoleny dvě skryté vrstvy o 200 neuronech s aktivační funkcí ReLU.

Síť byla trénována 1500 trénovacích epoch s využitím dávkového učení s dávkami o velikosti 10. Váhy sítě byly modifikovány stochastickým gradientním sestupem rozšířeným o Nesterovo momentum $\mu = 0.9$. Konstanta učení byla zvolena $\alpha = 0.01$. Kvůli předpokladům tohoto klasifikačního algoritmu byla neuronová síť natrénována pouze pro normalizovaná data.

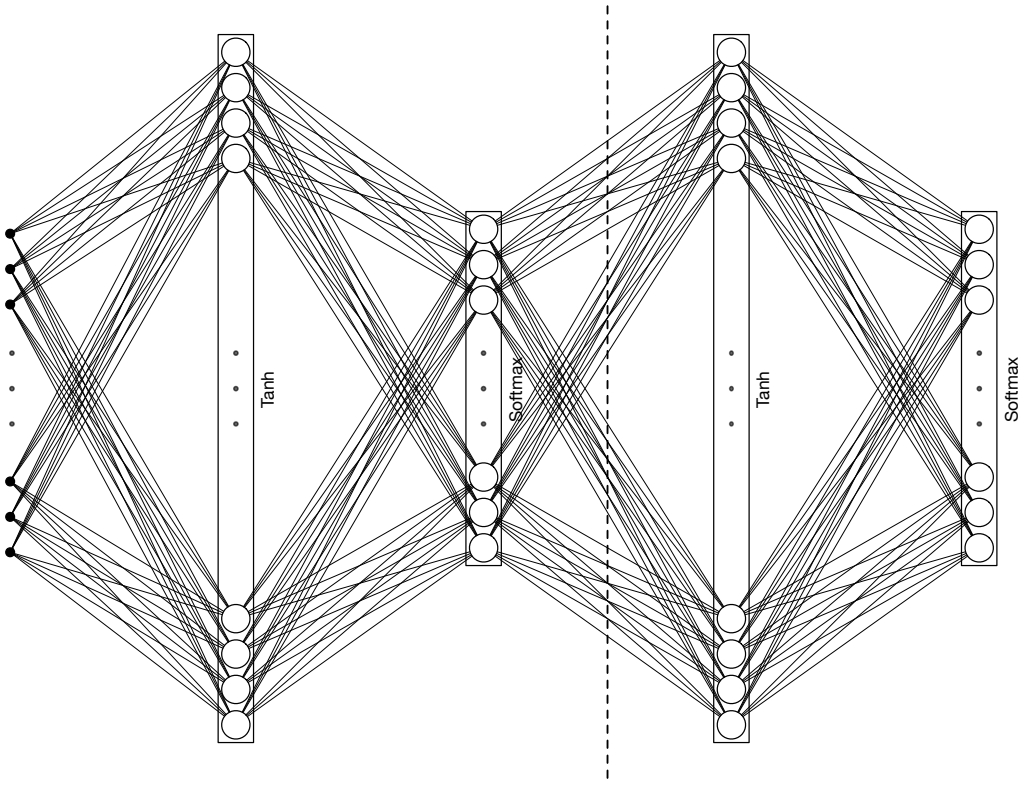


Obrázek 21: Třívrstvá neuronová síť.

19.5 Bottleneck

Pro generování příznaků pomocí neuronové sítě byl použit tzv. bottleneck. Bottleneckem je nazývána taková struktura neuronové sítě, kdy jedna ze skrytých vrstev (bottleneck vrstva) obsahuje výrazně nižší počet neuronů než její sousední vrstvy. Po natrénování neuronové sítě je bottleneck vrstva využita jako výstupní vrstva a všechny následující vrstvy jsou odstraněny. Tím dochází ke kompresi vstupní informace do příznakového vektoru o konstantní délce.

Bottleneck byl vytvořen natrénováním čtyřvrstvé neuronové sítě a odstraněním výstupní a poslední skryté vrstvy (znázorněno na obrázku 22). První a třetí skrytá vrstva se skládá z 1024 neuronů s aktivační funkcí Tanh, druhá (bottleneck) vrstva a výstupní vrstva mají aktivační funkci Softmax.



Obrázek 22: Čtyřvrstvá neuronová síť.

Neuronová síť byla natrénována pro dvě různé datové sady - původní datovou sadu šesti řečníků a původní datovou sadu rozšířenou o nahrávky pro Škoda Auto. Vstupem neuronové sítě je vektor o velikosti 440 složený z logaritmované energie banky filtru pro jeden foném (vektor o velikosti 40) a jeho kontextu zleva a zprava (vektory o velikosti $5 \cdot 40 = 200$).

Síť byla trénována pro klasifikaci 20 fonému pro původní datovou sadu s 8 a 16 neurony v bottleneck vrstvě a pro klasifikaci 40 fonému s 8, 16 a 32 neurony pro rozšířenou sadu. Pro přehlednější vyhodnocení si opět zavedeme značení pro vygenerované příznaky:

- *bn_X* - neuronová síť natrénovaná pro původní datovou sadu,
- *bn_SA_X* - neuronová síť natrénovaná pro rozšířenou datovou sadu,

kde X značí počet neuronů bottleneck vrstvy.

20 Vyhodnocení

Při vyhodnocení procentuální přesnosti klasifikace jednotlivých metod byly uvažovány tři různé varianty:

1. přesnost klasifikace v rámci jednoho řečníka (testovací nahrávky jednoho řečníka vůči svým referenčním nahrávkám)

2. přesnost klasifikace v rámci jednoho řečníka vůči ostatním (testovací nahrávky všech řečníků vůči referenčním nahrávkám jednoho řečníka)

3. přesnost klasifikace mezi všemi řečníky (testovací nahrávky všech řečníků vůči referenčním nahrávkám všech řečníků)

```
for speaker in speakers do
    reference_features, reference_targets = get_references(speaker)
    test_features, test_targets = get_test(speaker)
    model = train(reference_features, reference_targets)
    prediction = predict(test_features)
    accuracy = calculate_accuracy(prediction, test_targets)
end
```

Algoritmus 1: Vyhodnocení přesnosti klasifikace v rámci jednoho řečníka.

```
for speaker in speakers do
    /* combine test features/targets of all speakers                    */
    test_features, test_targets += get_test(speaker)
end
for speaker in speakers do
    reference_features, reference_targets = get_references(speaker)
    model = train(reference_features, reference_targets)
    prediction = predict(test_features)
    accuracy = calculate_accuracy(prediction, test_targets)
end
```

Algoritmus 2: Vyhodnocení přesnosti klasifikace v rámci jednoho řečníka vůči ostatním.

```
for speaker in speakers do
    /* combine reference and test features/targets of all speakers  */
    reference_features, reference_targets += get_references(speaker)
    test_features, test_targets += get_test(speaker)
end
model = train(reference_features, reference_targets)
prediction = predict(test_features)
accuracy = calculate_accuracy(prediction, test_targets)
```

Algoritmus 3: Vyhodnocení přesnosti klasifikace mezi všemi řečníky.

20.1 Přesnost klasifikace v rámci jednoho řečníka

Z tabulky 3 je zřejmé, že nejvyšší přesnosti klasifikace v rámci jednoho řečníka je dosaženo využitím příznaků generovaných neuronovou sítí. Pro příznaky generované neuronovou sítí natrénovanou nad původní sadou s bottleneck vrstvou o 8 neuronech je dokonce dosaženo nejvyšší přesnosti ze všech zkoumaných metod.

Velice vysoké přesnosti také dosahují příznaky ve frekvenční oblasti s tím, že Z-score normalizace jejich přesnost mírně snižuje. Ačkoliv by přidáním delta a delta-delta koeficientů mělo dojít k nárůstu přesnosti klasifikace, z neznámých důvodů došlo k jejímu poklesu. V případě normalizovaných příznaků se tento pokles pohybuje dokonce mezi 25-30%.

Jednotlivé příznaky v časové oblasti dle očekávání nedosahují vysokých přesností. Pro krátkodobou energii a intenzitu ovšem velmi pomáhá normalizace dat. Dále si můžeme povšimnout, že přesnost nenormalizované krátkodobé energie je stejná jako přesnost nenormalizované kombinace příznaků krátkodobé energie, intenzity a průchodů nulou. To je způsobeno tím, že nenormalizované hodnoty krátkodobé energie se pohybují v řádech desetitisíců, zatímco hodnoty krátkodobé intenzity a průchodů nulou v řádech desítek a oproti krátkodobé energii se projeví jen minimálně. Normalizací a kombinací těchto tří typů příznaků pak dostáváme typ příznaku s poměrně vysokou informační hodnotou. Optimalizací parametrů vzdálenostní metriky skutečně došlo k navýšení přesnosti a to o 1.1%.

Typ příznaků	Přesnost klasifikace [%]						
	Řečník						Průměr
	1	2	3	4	5	6	
bn_8	90.0	93.3	96.7	96.7	93.3	100.0	95.0
bn_16	100.0	90.0	93.3	96.7	86.7	90.0	92.8
bn_SA_8	90.0	83.3	86.7	83.3	83.3	93.3	86.7
bn_SA_16	96.7	93.3	93.3	96.7	86.7	80.0	91.1
bn_SA_32	93.3	93.3	96.7	93.3	76.7	93.3	91.1
log_fb_en_25_10_ham	80.0	83.3	100.0	93.3	83.3	93.3	88.9
log_fb_en_25_10_ham_norm	86.7	86.7	93.3	90.0	76.7	93.3	87.8
log_fb_en_25_10_ham_deltas	70.0	83.3	96.7	86.7	80.0	93.3	85.0
log_fb_en_25_10_ham_deltas_norm	63.3	60.0	73.3	76.7	66.7	76.7	69.4
mfcc_25_10_ham	86.7	83.3	100.0	93.3	90.0	90.0	90.6
mfcc_25_10_ham_norm	90.0	90.0	96.7	86.7	86.7	90.0	90.0
mfcc_25_10_ham_deltas	86.7	83.3	100.0	86.7	80.0	90.0	87.8
mfcc_25_10_ham_deltas_norm	70.0	60.0	76.7	70.0	66.7	76.7	70.0
ste_10_10	60.0	20.0	33.3	46.7	43.3	26.7	38.3
ste_10_10_norm	56.7	40.0	43.3	50.0	46.7	56.7	48.9
ste_sti_stzcr_10_10	60.0	20.0	33.3	46.7	43.3	26.7	38.3
ste_sti_stzcr_10_10_norm	96.7	86.7	80.0	80.0	66.7	76.7	81.1
ste_sti_stzcr_10_10_norm_single	96.7	90.0	83.3	83.3	66.7	73.3	82.2
sti_10_10	63.3	36.7	40.0	66.7	70.0	66.7	57.2
sti_10_10_norm	76.7	43.3	56.7	70.0	56.7	70.0	62.2
stzcr_10_10	56.7	70.0	60.0	63.3	60.0	60.0	61.7
stzcr_10_10_norm	60.0	66.7	53.3	60.0	43.3	53.3	56.1

Tabulka 3: Přesnost klasifikace v rámci jednoho řečníka pro DTW.

Výsledky dosažené klasifikátorem SVM (tabulka 4) jak pro příznaky ve frekvenční oblasti, tak pro bottleneck příznaky, jsou nižší než výsledky dosažené metodou DTW. Přidáním dynamických koeficientů opět došlo k výraznému poklesu přesnosti - 48.9% pro logaritmované energie banky filtrů a 32.3% pro MFCC. Přesnosti dosažené u příznaků v časové oblasti jsou srovnatelné s přesnostmi metody DTW.

Typ příznaků	Přesnost klasifikace [%]						
	Řečník						Průměr
	1	2	3	4	5	6	
bn_8	83.3	86.7	93.3	90.0	90.0	93.3	89.4
bn_16	90.0	83.3	90.0	93.3	86.7	86.7	88.3
bn_SA_8	86.7	73.3	80.0	70.0	80.0	90.0	80.0
bn_SA_16	83.3	80.0	76.7	90.0	76.7	76.7	80.6
bn_SA_32	80.0	70.0	86.7	80.0	70.0	80.0	77.8
log_fb_en_25_10_ham_norm	80.0	90.0	80.0	83.3	76.7	86.7	82.8
log_fb_en_25_10_ham_deltas_norm	60.0	46.7	20.0	20.0	26.7	30.0	33.9
mfcc_25_10_ham_norm	70.0	56.7	76.7	76.7	70.0	73.3	70.6
mfcc_25_10_ham_deltas_norm	60.0	46.7	30.0	30.0	26.7	36.7	38.3
ste_10_10_norm	60.0	33.3	46.7	43.3	50.0	53.3	47.8
ste_sti_stzcr_10_10_norm	90.0	76.7	73.3	60.0	66.7	73.3	73.3
sti_10_10_norm	73.3	40.0	43.3	60.0	53.3	66.7	56.1
stzcr_10_10_norm	60.0	63.3	53.3	50.0	40.0	50.0	52.8

Tabulka 4: Přesnost klasifikace v rámci jednoho řečníka pro SVM.

Přesnost klasifikace pomocí neuronové sítě (tabulka 5) pro příznaky ve frekvenční oblasti je srovnatelná s přesností příznaků generovaných neuronovou sítí a klasifikovaných pomocí DTW. Přidání dynamických koeficientů tuto přesnost opět snižuje. Vysokých přesností také dosahují bottleneck příznaky. Přesnost příznaků v časové oblasti je mírně horší než u SVM.

Typ příznaků	Přesnost klasifikace [%]						
	Řečník						Průměr
	1	2	3	4	5	6	
bn_8	83.3	90.0	93.3	96.7	90.0	96.7	91.7
bn_16	96.7	90.0	86.7	83.3	83.3	83.3	87.2
bn_SA_8	86.7	90.0	93.3	80.0	93.3	83.3	87.8
bn_SA_16	93.3	93.3	96.7	96.7	83.3	80.0	90.6
bn_SA_32	90.0	86.7	100.0	96.7	83.3	76.7	88.9
log_fb_en_25_10_ham_norm	86.7	93.3	93.3	100.0	93.3	90.0	92.8
log_fb_en_25_10_ham_deltas_norm	86.7	73.3	80.0	66.7	66.7	76.7	75.0
mfcc_25_10_ham_norm	93.3	96.7	93.3	96.7	86.7	90.0	92.8
mfcc_25_10_ham_deltas_norm	93.3	73.3	76.7	86.7	83.3	86.7	83.3
ste_10_10_norm	53.3	20.0	50.0	36.7	36.7	53.3	41.7
ste_sti_stzcr_10_10_norm	96.7	66.7	76.7	86.7	66.7	73.3	77.8
sti_10_10_norm	53.3	50.0	56.7	53.3	46.7	56.7	52.8
stzcr_10_10_norm	53.3	50.0	50.0	46.7	30.0	56.7	47.8

Tabulka 5: Přesnost klasifikace v rámci jednoho řečníka pro neuronovou síť.

20.2 Přesnost klasifikace v rámci jednoho řečníka vůči ostatním

Stejně jako při klasifikaci v rámci jednoho řečníka dosahuje nejlepší výsledků metoda DTW (tabulka 6) s využitím příznaků vygenerovaných neuronovou sítí. Nejvyšší přesnosti dosahuje bottleneck o 8 a 16 neuronech vytvořený natrénováním neuronové sítě nad původní datovou sadou.

Typ příznaků	Přesnost klasifikace [%]						
	Řečník						Průměr
	1	2	3	4	5	6	
bn_8	86.1	85.0	88.9	89.4	83.3	95.6	88.1
bn_16	90.6	87.8	86.1	90.0	86.1	92.2	88.8
bn_SA_8	85.6	71.7	80.0	67.8	80.0	81.7	77.8
bn_SA_16	86.7	81.1	85.6	80.0	87.2	81.1	83.6
bn_SA_32	77.8	76.7	78.9	80.0	83.9	87.2	80.7
log_fb_en_25_10_ham	68.9	70.6	82.8	70.6	77.8	82.8	75.6
log_fb_en_25_10_ham_norm	67.8	70.6	74.4	72.2	76.1	78.9	73.3
log_fb_en_25_10_ham_deltas	61.1	66.7	77.8	65.0	71.1	72.2	69.0
log_fb_en_25_10_ham_deltas_norm	49.4	52.8	58.3	53.9	57.2	55.0	54.4
mfcc_25_10_ham	70.6	72.8	81.7	72.8	75.6	87.2	76.8
mfcc_25_10_ham_norm	59.4	63.3	66.7	59.4	56.1	73.9	63.1
mfcc_25_10_ham_deltas	67.8	68.9	80.0	70.0	74.4	83.3	74.1
mfcc_25_10_ham_deltas_norm	47.8	46.7	52.2	42.8	48.3	47.2	47.5
ste_10_10	18.3	16.1	22.8	22.8	18.9	23.3	20.4
ste_10_10_norm	28.3	22.8	25.6	29.4	22.2	27.2	25.9
ste_sti_stzcr_10_10	18.3	16.1	22.8	22.8	18.9	23.3	20.4
ste_sti_stzcr_10_10_norm	70.0	60.6	65.6	60.6	59.4	66.7	63.8
ste_sti_stzcr_10_10_norm_all	70.0	60.6	68.3	65.6	61.7	64.4	65.1
sti_10_10	20.6	30.6	26.1	38.9	30.0	39.4	30.9
sti_10_10_norm	34.4	33.3	40.6	41.1	36.1	47.2	38.8
stzcr_10_10	52.2	49.4	48.3	41.7	54.4	56.1	50.4
stzcr_10_10_norm	50.6	47.8	49.4	44.4	47.8	51.1	48.5

Tabulka 6: Přesnost klasifikace v rámci jednoho řečníka vůči ostatním pro DTW.

Porovnáním tabulek 6, 7 a 8 je patrné, že příznaky ve frekvenční oblasti nejhůře klasifikuje SVM a příznaky v časové oblasti neuronová síť. Normalizace a aplikace dynamických koeficientů má na přesnost klasifikace obdobný vliv jako v případě klasifikace v rámci jednoho řečníka.

Typ příznaků	Přesnost klasifikace [%]						
	Řečník						Průměr
	1	2	3	4	5	6	
bn_8	81.1	80.0	83.9	82.2	82.8	93.9	84.0
bn_16	86.1	78.3	81.1	82.2	82.2	89.9	83.2
bn_SA_8	80.6	60.0	75.6	60.0	70.0	77.8	70.7
bn_SA_16	75.0	60.6	71.1	73.3	75.0	77.8	72.1
bn_SA_32	68.9	58.3	69.4	69.4	62.8	70.0	66.5
log_fb_en_25_10_ham_norm	62.2	65.0	64.4	62.2	65.6	66.7	64.4
log_fb_en_25_10_ham_deltas_norm	39.4	37.2	13.3	18.3	20.0	27.8	26.0
mfcc_25_10_ham_norm	50.6	46.7	43.3	41.1	35.0	53.3	45.0
mfcc_25_10_ham_deltas_norm	37.2	35.0	15.0	21.1	18.9	24.4	25.3
ste_10_10_norm	28.3	23.9	28.9	27.2	21.1	27.2	26.1
ste_sti_stzcr_10_10_norm	66.7	56.7	59.4	48.3	55.6	60.6	57.9
sti_10_10_norm	36.7	33.3	35.6	37.2	33.3	43.9	36.7
stzcr_10_10_norm	50.6	47.2	44.4	42.8	46.1	48.9	46.7

Tabulka 7: Přesnost klasifikace v rámci jednoho řečníka vůči ostatním pro SVM.

Typ příznaků	Přesnost klasifikace [%]						
	Řečník						Průměr
	1	2	3	4	5	6	
bn_8	77.2	80.6	83.9	88.3	78.9	88.3	82.9
bn_16	85.0	88.9	78.9	89.4	82.8	83.9	84.8
bn_SA_8	76.1	70.6	81.7	72.2	82.2	77.8	76.8
bn_SA_16	74.4	80.6	78.9	75.6	87.8	76.1	78.9
bn_SA_32	70.0	77.2	78.3	87.8	82.8	73.9	78.3
log_fb_en_25_10_ham_norm	70.0	74.4	80.0	74.4	80.0	85.6	77.4
log_fb_en_25_10_ham_deltas_norm	57.8	49.4	53.3	57.2	42.2	48.9	51.5
mfcc_25_10_ham_norm	68.3	72.8	68.3	66.7	68.9	78.3	70.6
mfcc_25_10_ham_deltas_norm	63.9	58.9	63.9	54.4	62.8	67.2	61.9
ste_10_10_norm	23.3	19.4	26.1	28.3	22.2	25.6	24.2
ste_sti_stzcr_10_10_norm	56.7	46.7	64.4	58.3	58.3	66.1	58.4
sti_10_10_norm	27.2	33.9	30.0	33.9	27.8	35.6	31.4
stzcr_10_10_norm	45.6	40.6	45.6	45.6	40.6	44.4	43.7

Tabulka 8: Přesnost klasifikace v rámci jednoho řečníka vůči ostatním pro neuronovou síť.

20.3 Přesnost klasifikace mezi všemi řečníky

Při klasifikaci mezi všemi řečníky (tabulka 9) navzájem dosahují opět nejlepších výsledků bottleneck příznaky a to zejména 8 neuronový natrénovaný nad původní datovou sadou a 16 neuronový natrénovaný nad rozšířenou sadou. Velmi vysokých přesností také dosahují příznaky ve frekvenční oblasti a kombinace příznaků v časové oblasti.

Jak již bylo zmíněno v teoretické části, krátkodobá energie je silně ovlivňována výkyvy v amplitudě akustického signálu, zatímco krátkodobá intenzita tento problém nemá. Všimněme si tedy, že krátkodobá intenzita dosahuje téměř dvakrát větší přesnosti než krátkodobá energie.

	Průměrná přesnost [%]		
Typ příznaků	DTW	SVM	NN
bn_8	98.9	94.4	98.9
bn_16	96.7	92.2	95.6
bn_SA_8	90.6	76.7	91.1
bn_SA_16	98.9	73.9	92.8
bn_SA_32	94.4	75.0	91.1
log_fb_en_25_10_ham	90.6	-	-
log_fb_en_25_10_ham_norm	92.8	74.4	91.1
log_fb_en_25_10_ham_deltas	86.1	-	-
log_fb_en_25_10_ham_deltas_norm	73.3	30.0	10.0
mfcc_25_10_ham	91.7	-	-
mfcc_25_10_ham_norm	90.6	52.8	65.0
mfcc_25_10_ham_deltas	89.4	-	-
mfcc_25_10_ham_deltas_norm	74.4	33.3	10.0
ste_10_10	32.8	-	-
ste_10_10_norm	36.1	29.4	52.2
ste_sti_stzcr_10_10	32.8	-	-
ste_sti_stzcr_10_10_norm	82.2	76.1	77.2
ste_sti_stzcr_10_10_norm_all	85.6	-	-
sti_10_10	60.6	-	-
sti_10_10_norm	62.2	46.1	57.2
stzcr_10_10	63.9	-	-
stzcr_10_10_norm	56.7	53.3	58.9

Tabulka 9: Přesnost klasifikace mezi všemi řečníky.

Přesnost klasifikátoru pro příznaky v časové a frekvenční oblasti SVM je výrazně horší než přesnost DTW. Neuronová síť se zdá být velice robustní pro normalizované logaritmované energie banky filtrů, nicméně pro příznaky s dynamickými koeficienty se nepodařilo síť s danou strukturou úspěšně natrénovat a přesnost je pouhých 10%. Příznaky generované pomocí neuronové sítě s bottleneck vrstvou o 8 neuronech nad původní datovou sadou se zdají být nezávislé na klasifikační metodě a dosahují velmi vysokých přesností. Z hlediska výpočetních nároků je ovšem vhodnější volit pouze metodu DTW, jelikož klasifikátor SVM i neuronová síť využívají předpočítané DTW vzdálenosti.

21 Závěr

Cílem této práce bylo porovnat různé typy příznaků pro úlohu klasifikace izolovaných slov. V první částí práce byly představeny metody zpracování akustického signálu včetně jednotlivých typů příznaků a byly odvozeny klasifikační algoritmy. Ve druhé části pak byly představeny testované parametrizace příznaků a navržené algoritmy využité ke klasifikaci.

Typ příznaků	Průměrná přesnost klasifikace [%]								
	DTW			SVM			NN		
	1to1	AlltoAll	Rozdíl	1to1	AlltoAll	Rozdíl	1to1	AlltoAll	Rozdíl
bn_8	95.0	98.9	3.9	89.4	94.4	5	91.7	98.9	7.2
bn_16	92.8	96.7	3.9	88.3	92.2	3.9	87.2	95.6	8.4
bn_SA_8	86.7	90.6	3.9	80.0	76.7	-3.3	87.8	91.1	3.3
bn_SA_16	91.1	98.9	7.8	80.6	73.9	-6.7	90.6	92.8	2.2
bn_SA_32	91.1	94.4	3.3	77.8	75.0	-2.8	88.9	91.1	2.2
log_fb_en_25_10_ham	88.9	90.6	1.7	-	-	-	-	-	-
log_fb_en_25_10_ham_norm	87.8	92.8	5.0	82.8	74.4	-8.4	92.8	91.1	-1.7
log_fb_en_25_10_ham_deltas	85.0	86.1	1.1	-	-	-	-	-	-
log_fb_en_25_10_ham_deltas_norm	69.4	73.3	3.9	33.9	30.0	-3.9	75.0	10.0	-65.0
mfcc_25_10_ham	90.6	91.7	1.1	-	-	-	-	-	-
mfcc_25_10_ham_norm	90.0	90.6	0.6	70.6	52.8	-17.8	92.8	65.0	-27.8
mfcc_25_10_ham_deltas	87.8	89.4	1.6	-	-	-	-	-	-
mfcc_25_10_ham_deltas_norm	70.0	74.4	4.4	38.3	33.3	-5	83.3	10.0	-73.3
ste_10_10	38.3	32.8	-5.5	-	-	-	-	-	-
ste_10_10_norm	48.9	36.1	-12.8	47.8	29.4	-18.4	41.7	52.2	10.5
ste_sti_stzcr_10_10	38.3	32.8	-5.5	-	-	-	-	-	-
ste_sti_stzcr_10_10_norm	81.1	82.2	1.1	73.3	76.1	2.8	77.8	77.2	-0.6
sti_10_10	57.2	60.6	3.4	-	-	-	-	-	-
sti_10_10_norm	62.2	62.2	0.0	56.1	46.1	-10.0	52.8	57.2	4.4
stzcr_10_10	61.7	63.9	2.2	-	-	-	-	-	-
stzcr_10_10_norm	56.1	56.7	0.6	52.8	53.3	0.5	47.8	58.9	11.1

Tabulka 10: Porovnání průměrné přesnosti klasifikace v rámci jednoho řečníka (*1to1*) a mezi všemi řečníky (*AlltoAll*).

Porovnáním průměrných přesností klasifikace (tabulka 10) se ukázalo, že nejpřesnějším a nejrobustnějším příznakem (nezávislý na řečníkovi) je příznak vygenerovaný neuronovou sítí s bottleneck vrstvou. Velice dobrých výsledků také dosahovaly příznaky založené na logaritmované energii banky filtrů klasifikovaných jak pomocí metody DTW, tak pomocí neuronové sítě.

Mezi nejméně přesné typy příznaků pak patřily příznaky v časové oblasti. Ukázalo

se ovšem, že zkombinováním jednotlivých příznaků v časové oblasti lze vytvořit příznak s poměrně vysokou informační hodnotou. Ačkoliv se čekalo, že příznaky v časové oblasti budou silně závislé na řečníkovi a při klasifikaci mezi všemi řečníky dojde k poklesu přesnosti, došlo k této situaci pouze pro krátkodobou energii u metod DTW a SVM a pro krátkodobou intenzitu u SVM. U ostatních příznaků došlo naopak k navýšení přesnosti.

Pro další zlepšení dosažených výsledků by bylo vhodné zaměřit se na příznaky generované neuronovou sítí a pokusit se optimalizovat její strukturu. Dalším krokem by pak bylo otestování rekurentních neuronových sítí (zejména typu LSTM), které v dnešní době pro podobné úlohy dosahují velice dobrých výsledků. Pro využití v praxi by pak bylo potřeba tento systém propojit se systémem pro detekci hlasové aktivity (voice activity detection).

Seznam obrázků

Seznam tabulek

Reference

[1] M. Majer. Detekce klíčových frází. Bakalářská práce, Západočeská univerzita v Plzni, 2016.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[3] A. Karpathy. Cs231n convolutional neural networks for visual recognition, 2016. [Online], Dostupné z: `http://cs231n.github.io`.

[4] S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[5] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu. Advances in optimizing recurrent networks. *CoRR*, abs/1212.0901, 2012.

[6] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[7] Ch. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[8] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning*. Springer, 2015. ISBN: 978-1-461-47137-0.

[9] T. Parr and J. Howard. The matrix calculus you need for deep learning. abs/1802.01528, 2018.

[10] C. Olah. Neural networks, types, and functional programming, 2015. [Online], Dostupné z: `http://colah.github.io/posts/2015-09-NN-Types-FP`.

[11] C. Olah. Understanding lstm networks, 2015. [Online], Dostupné z: `http://colah.github.io/posts/2015-08-Understanding-LSTMs`.

[12] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11):2673–2681, November 1997.

[13] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.