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

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

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

Proměnná 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

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

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

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

3 Neuronové sítě

Základním rozdílem mezi čistou optimalizací a optimalizací v neuronových sítích je především ten, že optimalizace ve strojovém učení probíhá nepřímo, tj. optimalizujeme vybranou metriku P měřící výkon našeho algoritmu. Minimaluzujeme jinou ztrátovou funkci $J(\theta)$ za účelem minimalizace metriky P, zatímco u čisté optimalizace minimaluzjeme přímo ztrátovou funkci $J(\theta)$ za účelem její minimalizace.

4 Kapacita modelu

5 Connectionist temporal classification

6 Klasifikace fonémů

7 Vyhodnocení

8 Závěr

9 Neuronové sítě

9.1 Dopředné neuronové sítě

Dopředné neuronové sítě, občas také nazývané vícevrstvé perceptrony jsou základními deep learningovým modely. Jejich cílem je aproximovat určitou funkci f^*, např. klasifikátor $y = f^*(x)$ zobrazuje vstup x na výstupní třídu y. Dopředná neuronová síť definuje toto mapování $y = f^*(x; \theta)$ a učí se hodnoty parametrů θ, jejímž výsledkem je by měla být nejlepší aproximace funkce.

Název dopředný implikuje, že tok informace putuje od dopředu od vstupu x přes několik po sobě jdoucích výpočetních vrstev definujících f až k výstupu y. Nevyskytují se zde žádné zpětné vazby od výstupu modelu do předešlých vrstev. Zavedením těchto zpětných vazeb bychom pak hovořili o rekuretních neuronových sítích.

Dopředné neuronové sítě jsou nazývaány sítěmi právě kvůli jejich typické reprezentaci - složení několika různých funkcí. Model lze popsat jako directed acyklický graf, který určuje závislosti mezi funkcemi. Například pro síť skládající se ze tří funkcí $f^{(1)}$, $f^{(2)}$ a $f^{(3)}$ spojených do řady $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. Tato řetězová struktura je typická pro definici neuronových sítí. V tomto případě nazýváme $f^{(1)}$ první vrstvou, $f^{(2)}$ druhou vrstvou, atd. Celková délka tohoto řetězce udává hloubku modelu. Poslední vrstva se nazývá výstupní vrstva.

Během trénování neuronové sítě cílíme na to, aby se síť naučila odpovídající mapovaní (match) mezi $f(x)$ a $f^*(x)$. Trénovací data nám poskytují zašuměné a aproximované pozorování (examples) $f^*(x)$ vyhodnoceny v různých training points. Ke každému pozorování x máme k dispozici také label (skutečnou hodnotu) $y \approx f^*(x)$. Trénovací pozorování přesně určují, co výstupní vrstva musí udělat pro bod x - musí vyprodukovast hodnotu blízkou y. Chování ostatní vrstev není přímo dáno trénovacími daty. Učící algoritmus se musí rozhodnout, jak tyto vrstvy využít k získání požadovaného výsledku, ale trénovací data přesně neříkají, co mají jednotlivé vrstvy udělat. Namísto toho se musí učící alogritmus rozhodnot, jak tyto vrstvy co nejlépe použít k co nejlepší aproximaci $f^*(x)$. Vzhledem k tomu, že trénovací adta neukazují požadovaný výstupní pro tyto vrstvy, říkáme jim skryté vrstvy.

Tyto modely jsou nazývanými neuronový dle jejich inspirace lidským mozkem. Každá skrytá vrstva je typicky vektor, jehož dimenze udává šířku modelu. Každý prvek tohoto

vektoru může pak být považován za jednotku analogickou s neuronem v lidském mozku (perceptron). Tyto jednotky/prvky pracují paralelně a reprezentují funkci zobrazující vektor na skalár. Analogie k neuronu spočívá v tom, že přijme vstup od několika různých jednotek (z předchozí vrstvy) a spočítá svojí aktivační hodnotu. V dnešní době už je ale zřejmé, že neuronové sítě ani zdaleka nekopírují funkci mozku a nahlížíme na ně pouze jako algoritmus aproximující funkce tak, aby dosáhli vysoké míry statistické generalizace.

DEEP LEARNING

9.1.1 Učení založené na gradientu

Návrh a trénování neuronové sítě není nijak rozdílný od ostatní algoritmů strojového učení založených na gradientním sestupu. Největší rozdíl mezi takovými algoritmy a neuronovými sítěmi je, že nelinearita neuronové sítě může způsobit, že ztrátová funkce se stane nekonvexní. To znamenám, že neuronové sítě jsou většinou trénovány pomocí iterativních optimalizačních technik, které cost function dostanou na nízkou hodnotu místo toho, aby zajistili globální konvergenci jako např. u logistické regrese nebo SVM. KOnvexní optimalizace konverguje z libovolných počátečních parametrů. Stochastický gradientní sestup aplikovaný na nekonvexní loss function negarantuje konvergenci a je citlivý na hodnoty počáteční parametrů. Pro dopředné neuronové sítě je důležité incializovat všechny váhy nmalými náhodnými čísly. Biasy mohou být inicviálzovany nulou či velmi malou klasdnou hodnotu. V následující kapitole si odvodíme algoritmus, kterým lze spočítat gradient pro neuronové sítě - backpropagation. Srtejně jako u ostatních modelů založených na gradientu i zde musíme definovat cost function a vhodně reprezentovat výstup modelu.

9.1.2 Cost function

Základním aspektem návrhu (hlubokých) neuronových sítí je výběr cost function. Naštěstí, pro neuronové sítě jsou cost function definovány téměř stejně jako pro ostatní parametrické modely. Ve většině případech parametrický model definuje rozložení pravděpodobností $p(y \mid x; \theta)$ a využívá pricnipu mximum likelyhood.To znamenám, že využíváme crossentropy mezi trénovacími daty a predikcemi modelu jako cost function. Nebo můžeme takové zvolit jednodušší přístup, kde místo kompletního rozložení pravděpodobnosti y predikujeme pouze dané statistiky y (např. průměr y) conditioned on x. S vhodnout ztrátovou funkcí psk můžeme natrénovat prediktor těchto statistik.

Předpis cost function kromě hlavní části cost function také často obsahuje regularizační část, např. weight decay.

Většina moderních neuronových sítí je trénovaná pomocí maximum likelyhood. To znamená, že cost function je dána jako záporná log-likelyhood, ekvivalentně popsána jako crossentripoe mezi trénovacími daty a model distribution. Cenová funkce je tedy dána jako

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

Tvar cost function se liší modelem od modelu v závislosti na tvary $\log p_{model}$. Rozšířením rovnice výše pak získáváme členy, které nejsou přímo závislé na parametrech modelu a mohou být discarded. Výhodou tohoto přístupu (maximum likelyhood) je, že specifikací modelu $p(y \mid x)$ automaticky definujeme cost function $\log p(y \mid x)$.

9.1.3 Výstupní jednotky

Výběr cost function je velice úzce svázán s výběrem výstupních jednotech. Většinou využíváme cross entropii mezi rozložením data a modelu. Výběrem výstupních jednotek pak udává tvar crossentropy function. Libovolná jednotka, která může být použita jako výstupní jednotka také může být použita jako jednotka ve skryté vrstvě. Předpokládejme, že dopředná neuronová síť dává soubor skrytých příznaků $h = f(x;\theta)$. Rolí výstupní vrstvy je provést nějakou transformaci příznaků, aby neuronová síť splnila úlohu, pro kterou byla navržena. Vzhledem k tomu, že v rámci této práce predikujeme pouze pravděpodobnosti jednotlivých tříd na výstupu sítě, budeme se zde věnovast pouze aktivační funkci softmax.

Aktivační funkci softmax využíváme tehdy, pokud chceme na výstupu reprezentovat rozložení pravděpodobnosti nad diskrétní proměnnou s n možnými hodnotami. Na funkci softmax můžeme nahlížet jako na generalizaci sigmoid funkce, která byla navržena pro reprezetnaci rozložení pravděpodobnosti nad binární proměnnou. Softmax funkce je nejčastěji využívána na výstupu klasifikátoru k reprezetnaci rozložení ppsti nad n různými třídami.

K odvození funkce softmax využijeme znalosti o lineární aktivační funkci a aktivační funkci sigmoid. V případě binární klasifikace bychom chtěli na výstupu dostat jedno číslo

$$\hat{y} = P(y = 1 \mid x).$$

Vzhledem k tomu, že požadujeme, aby toto číslo leželo mezi 0 a 1 a zároveň abychom udrželi numerickou stabilitu při optimalizace založené na gradientním sestupu log-likelyhood, budeme radši predikovat číslo $z = \log \tilde{P}(y = 1 \mid x)$. Zexponenciálněním a normalizací dostaneme Bernoulliho rozložení controlledby sigmoidální funkcí.

Generalizací pro diskrétní proměnnou s n hodnotami potřebujeme vytvořit vektor $\hat{y}$, kde $\hat{y}_i = P(y = i \mid x)$. Nyní již požadujeme nejen tom, aby každý prvek $\hat{y}_i$ ležel mezi 0 a 1, ale zároveň aby suma celeého vektoru byla 1 a bylo možné ji považovat za hustotu pravděpodobnosti. Provedmě tedy nejprve predikci vrstvou s lineární vrtvou, která predikuje nenormalizované log pravděpodbnosti

$$z = W^T h + b,$$

kde $z_i = \log \tilde{P}(y = i \mid x)$. Softmax funkce pak provede operaci exponentu a normalizace z abychom získali požadované $\hat{y}$

$$softmax(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Natrénovaním parametrů modelu pak bude softmax predikovat podíly počtů všech možných viděných výsledků v trénovacím datasetu

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

DEEP LEARNING

9.1.4 Skryté jednotky a aktivační funkce

Nyní se budeme zabývat nejčastěji používanými aktivačními funkcemi pro skryté jednotky. Většinou je primárně volena rectified linear unit aktivační funkce, která ale ne vždy zaručuje nejlepší možné výsledky. Volba aktivačních funkcí ve skrytých vrstvách vyžaduje mnoho trial and error a evaluaci perfomance na validační sadě. Předvedeme si pouze aktivační funkce využité v rámci této práce.

Některé z déle uvedených funkcí nejsou diferencovatelné ve všech bodech, což by naznačovalo, že tyto funkce nelze použít při optimalizaci pomocí graidentní setupu. V praxi ovšem gradientní sestup funguje velice dobře i při proušení toho předpokladu. Vzhledem

k tomu, že neuronové sítě nenacházají globální a velice často an ilokální minimum, ale pouze výrazně minimaluzjí cost func, tzn. neočekáváme, že by byl gradient nulový, je přijatelné aby minimum cost func odpovídalo bodu s nedefinovaným gradientem.

Skryté jednotky mohou být popsány jako jednotky, jejich vstupem je vektor vstupů x a počítají afinní transformaci $z = W^T x + b$ a poté po prvcích aplikují nelineární funkci $g(z)$.

RELU - recitfied linear units používají aktivační funkci $g(z) = \max\{0, z\}$. Tyto jednotky jsou snadno optimalizovatelné jelikož jsou podobné lineární jednotkám. Jediny rozdíl je, že RELU vrací nulu pro polovinu svého definičního oboru. Gradienty jsou pak velké, kdykoliv je jedjnotka aktivní. Nejenom ž jsou velké, ale raké konzistentní. Druhá derivace je téměř všude 0 a 1 tam, kde jsou jednotky aktivní. To znamená, že směr gradientu je mnohem užitečnější pro učení než s aktivačními funkcemi, které využívají efekty druhého řádu.

Jedním z nedostatků RELU je, že pomocí gradientních metod se nemůžou učit z pozorování, jejichž aktivace je rovna nule. Mnoho generalizací však RELU zajišťuje, že budou mít fgradient všude (leaky relu, prelu, maxout unit).

sigmoid a tanh - před příchodem aktivační funkce RLEU byla hojně využívana logistická sigmoidální aktivční funkce $g(z) = \sigma(z)$ nebo hyperbolický tangens $g(z) = \tanh(z)$, které jsou si spolu velmi blízké, jelikož $\tanh(z) = 2\sigma(2z) - 1$. Hlavním problémem sigmoidální funkce je její citlivost a náchylnost k saturaci - pro velké kladní hodnoty z saturuje k vyskoým hodnotám a naopak. Funkce je citlivá pouze na hodnoty z blízké nule. Kvůli tomuto chování je učení pomocí gradientních metod značně obtížné. Když už tato funkce musí být využíta ve skryté vrstvě, doporučuje se raději využít hyperbolický tangen, který dosahuje lepších výsledků.

hard sigmoid $g(a) = \max(0, \min(1, a))$.

DEEP LEARNING

9.1.5 Architektura neuronových sítí

Další klíčovým prvkem při návrhu neuronových sítí je jejich architektura. Architekturou sítě či modelu rozumíme celkovou strukutru sítě - kolik jednotek má síť mít a jak mají být tyto jednotky mezi sebou propojeny.

Neuronové sítě jsou většinou tvořeny skupinami jednotek (neuronů), které jsou uspořádány

do tzv. vrstev. Většina architektur uspořádává tyto vrstvy do řetězové (chain, chain-based) struktury, kdy každá vrstva je funkcí vrstvy, která ji předchází. V této struktuře je první vrstva dána jako

$$h^{(1)} = g^{(1)}\left(W^{(1)T}x + b^{(1)}\right),$$

druhá vrstva je dána jako

$$h^{(2)} = g^{(2)}\left(W^{(2)T}x + b^{(2)}\right),$$

a tak dále.

V této řetězové architektuře je pak hlavní problémem určení hloubky sítě a šířky každé vrstvy s tím, že ideální architekturu je třeba najít pro každou úlohu pomocí experimentů zlaožených na sledování chyby na validační datové sadě a apriorní znalosti o úloze a datech.

9.1.6 Backpropagation

Když využíváme dopřednou neuronovou síť, která na vstupu přijme x a na výstupu vyprodukuje $\hat{y}$, informace/informační tok proudí skrz síť dopředu (proto dopředná síť). Vstup x dává síti počáteční informaci, která je poté propagována do skrytých jednotek ve všech vrstvách až k výstupní vrstvě, která vyprodukuje $\hat{y}$. Tomuto procesu se říká dopředná propagace. Během trénování probíhá dopředná proapgace dokud nevyprodukuje skalární cost $J(\theta)$. Algoritmus zpětného šíření (backpropagation) pak umožní zpětný tok informace od cost/ceny skrz síť za účelem výpočtu gradientu.

Analytický výpočet gradientu je poměrně přímočarý, nicméně numericky může být výpočetně velmi náročný. Algoritmus zpětného šíření výpočet gradientu provádí pomocí jednoduché a výpočetně nenáročené procedury.

Velmi často je algoritmus zpětného šíření považován za celý učící algoritmus vícevrstvých neuronových sítí. Jedná se ovšem pouze o efektivní metodu výpočtu gradientu a samotné učení obstarává jiný optimalziační gradient (např. SGD), který využívá gradient spočítaný metodou zpětného šíření. Nyní se budeme zabývat tím, jak pomocí tohoto algoritmu spočítat gradient $\nabla_x f(x, y)$ pro libovolnou funkci, kde x je soubor proměnných, jejichž gradient chceme spočítat a y je soubor proměnných, jež jsou vstupem funkce, ale pro něž nevyžadujeme výpočet gradientu. V učících algoritmech jako je neuronová síť většinou počítáme gradient cenové funkce (cost function) with regard to parameters, $\nabla_\theta J(\theta)$.

řetězové pravidlo

Řetězové pravidlo se běžně využívá k výpočtů derivací složených funkcí, které jsou složeny z funkcí, jejichž derivace jsou známé. Algoritmus zpětného šíření je algoritmus, tkerý počítá řetězové pravidlo v přesně daném sledu operací tak, aby byl co nejvýkonnější.

Mějmě reálné číslo x a funkce f a g, které zobrazují reálné číslo na reálné číslo. Dáke předpokládejme, že $y = g(x)$ a $z = f(g(x)) = f(y)$. Pak řetězové pravidlo říká, že

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

Toto pravidlo můžeme rozšířit ze skalárního případu. předpokládejme, že $x \in \mathbb{R}^n, y \in \mathbb{R}^n$, g zobrazuje $\mathbb{R}^m$ na $\mathbb{R}^n$ a f je zobrazení z $\mathbb{R}^n$ do $\mathbb{R}$. Pokud $y = g(x)$ a $z = f(y)$, pak

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}$$

Zapíšeme-li rovnici (odkaz) vektorově, získáme tvar

$$\nabla_x z = \left(\frac{\partial y}{\partial x}\right)^T \nabla_y z.$$

, kde $\frac{\partial y}{\partial x}$ je $n \times m$ je Jakobián (Jacobian matrix) x. V tomto tvaru řetězové pravidla vidíme, že gradient proměnné x lze snadno získat vynásobením Jakobiánu $\frac{\partial y}{\partial x}$ gradientem $\nabla_y z$.

Toto pravidlo lze dále snadno rozšířit i na tensory, jenž jsou v neuronových sítích velice často používány.

dopředné a zpětné šíření pro MLP

Uveďmě si nyní příklad dopředného a zpětného šíření v dopředné neuronové síti. Předpokládejme síť o hloubce l, kde každé vrstvě náleží váhová matice $\mathbf{W}^{(i)}, i \in \{1, \ldots, l\}$ a parametr bias $b^{(i)}, i \in \{1, \ldots, l\}$. Loss function $L(y, \hat{y})$ závisí na targetu y a výstupu sítě $\hat{y}$, která síť vyprodukuje, pokud dostane na vstupu x. Celková cena J pro zjednodušení neobsahu žádnou regularizační složku a odpovídá zrtrátové funcki L.

$h^{(0)} = x$

for $k = 1, \ldots, l$ **do**

  a .. aktivační hodnota (TODO)

  $a^{(k)} = b^{(k)} + W^{(k)} h^{(k-1)}$

  $h^{(k)} = f(a^{(k)})$

end

$\hat{y} = h^{(l)}$

$J = L(y, \hat{y})$

Algoritmus 1: Dopředné šíření.

Při zpětném průchodu jsou počítány gradienty aktivací $a^{(k)}$ pro každou vrstvu k počínaje výstupní vrstvou až k první skryté vrstvě. Tyto gradienty mohou být interpretovány jako indikátor, jak by se měl výstup vrstvy změnit, aby snížil chybu. Gradienty vah a biasů mohou být rovnou využity pro optimalizace jako součást stochastic grad. updatu (provedení změny parametrů hned po výpočtu gradinetů) popř. mohou být využity později pro jinou optimalizační metodu založenou na gradientu.

výpočet gradientu výstupní vrstvy (po dopředné průchodu sítě)

$g \leftarrow \nabla_{\hat{y}} J = \nabla_{\hat{y}} L(y, \hat{y})$

for $k = l, l-1, \ldots, 1$ **do**

  zkonvertování gradientu na výstupu vrstvy do gradientu před aplikací nelinární aktivace

  $g \leftarrow \nabla_{a^{(k)}} J = g \odot f'(a^{(k)})$

  výpočet gradientů vah a biasu

  $\nabla_{b^{(k)}} J = g$

  $\nabla_{W^{(k)}} J = g h^{(k-1)\top}$

  propagace/šíření gradientu wrt aktivaci skryté vrstvy o úroveň níže

  $g \leftarrow \nabla_{h^{(k-1)}} J = W^{(k)\top} g$

end

Algoritmus 2: Zpětné šíření.

TODO: ztučnit $\boldsymbol{x}^\top$, y,W,b,g hat(y)?

DEEP LEARNING

9.2 Rekurentní neuronové sítě

Rekurentní neuronové sítě (dále RNN) jsou typem neuronových sítí, která umí zpracovávat sekvenční data, tj. sekevenci hodnot $x^{(1)}, \ldots, x^{(\tau)}$. Abychom mohli od vícevrstvých

sítí přejít k rekurentním, musíme využít principu z raných let strojového učení a statistického modelování z 80. let: sdílení parametrů mezi různými částmi modelu. Sdílení parametrů umožní rozšířit a aplikovat model na různé tvary pozorování (forms, v tomto případě různé délky) a generalizovat mezi nimi. Kdybych měli samostatné parametry pro každou hodnotu v každém časové bodě (time index), nemohli bychom generalizovat na délky sekvencí, které nebyly viděny během trénovácí fázi a také bychom nemohli sdílet statistickou sílu (statistical strength) napříč různými délkami sekvencí a napříč různými pozicemi v čase. Toto sdílení je zejména důležité v případech, kdy specifická informace může nastat na více pozicích v sekvenci (např. stejné datum na různých pozicích ve větě).

Každý prvek výstupu je funkcí prvků minulých prvků výstupu. Každý prvek výstupu je vytvořen stejným updatovacím pravidlem jako minulé výstupy. -¿ sharing through very deep computitaional graph

Pro jednoduchost předpokládejme, že RNN budu pracovat se sekvencí obsahující vektory $x^{(t)}$ s časovými indexy t v rozsahu 1 až τ. Ve skutečnosti RNN většinou pracují s minibatchemi takových sekvencí, s tím, že každý člen minibatche může mít různé τ.

Abychom zajistili sdílení parametrů, musíme rekurentní výpočet "rozvinout"(unfolding) do výpočetní grafu s opakující se strukturou. Předpokládejme model v klasickém tvaru

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

kde $s^{(t)}$ nazýváme stavem systému. Tato rovnice (odkaz) je rekurzivní, jelikož definice s v čase t závisí na té samé definici v čase $t-1$. Pro konečný počet časových kroků τ můžeme rozvinout graf aplikací vzorce (definition) $\tau - 1$-krát. Např. pro rovnici výše (odkaz) pro rozvinutí $\tau = 3$ časových kroků, získáme

$$s^{(3)} = f(s^{(2)}; \theta) = f(f(s^{(1)}; \theta); \theta)$$

Tímto opakovaným rozvinutím zajistíme, že rovnice již neobsahjuja žádný rekurenci a mlže být reprezentována tradičním acyklickým directed comp. grafem.

OBRAZEK? DL 370

Přidáním závislosti na externím vstupu $x^{(t)}$ můžeme zadefinovat hodnoty skrytých

jednotek. Základní rovnice pro RNN je pak

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

kde h je stav. Typická RNN pak rozšíří tuto rovnici a další architectural feasturas jako např. výstupní vrstvu která čte informaci ze skrytých stavů h za účelem predikce.

Při trénování sítě na danou úlohu, která vyžaduje predikci budoucnosti z minulosti, síť se běžně učí jak používat $h^{(t)}$ jako ztrátovou summary relevantních aspektů úlohy z minulosti sekvence od vstupu až po t. Tato summary musí být z logiky ztrátová, jelikož mapuje sekvence o proměnné délce $(x^{(t)}, x^{(t-1)}, x^{(t-2)}, \ldots, x^{(2)}, x^{(1)})$ do vektoru pevně dané délky $h^{(t)}$. V závisloti na trénovacím kritériu, toto summary si může ponechat jisté aspekty z minulosti s jinou přesností než jiné.

DEEP LEARNING

Nyní můžeme zadefinovat rovnice pro dopředné šíření pro RNN zobrazenou v obrázku níže (odkaz). Ačkoliv obrázek přesně neurčuje aktivační funkci skrytých jednotek, zde budeme předpokládat hyperbolický tangens, který je v RNN využíván nejčastěji. Přirozenou cestou, jak reprezentovat diskrétní proměnné je vracet ve výstupu o nenormalizované log pravděpodobnosti pro každou možnou diskrétní proměnnou. Aplikací operace softmax pak získáme vektor $\hat{y}$ normalizovaných pravděpodobností nad výstupem. Dopředné šíření začíná specifikací počátečního stavu $h^{(0)}$. Poté jsou pro každý časový krok $t = 1$ až $t = \tau$ aplikovány následující rovnice updatu:

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$

$$h^{(t)} = \tanh(a^{(t)})$$

$$o^{(t)} = c + Vh^{(t)}$$

$$\hat{y}^{(t)} = \text{softmax}(o^{(t)}),$$

kde parametry jsou bias vektory b a c spolus s váhovými maticemi U, V a W, které popořadě odpovídají skrytým spojením (hidden connection) mezi input-to-hidden, hidden-to-output a hidden-to-hidden. Předpokládejme naoříklad RNN, která mapuje vstupní sekvenci na výstupní sekvenci o stejné délce. Celková ztráta pro danou sekvenci hodnot x paired with sekvence hodnot y by byla pouze suma ztrát ve všech časových okamžících.

Např. pokud $L^{(t)}$ je záporní log-likelyhood $y^{(t)}$ pro $x^{(1)}, \ldots, x^{(t)}$, pak

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

kde $p_{model}\left(y^{(t)} \mid \{x^{(1)}, \ldots, x^{(t)}\}\right)$ je dán čtením vstupu $y^{(t)}$ z výstupního vektoru modelu $\hat{y}^{(t)}$. Výpočet gradientu této ztrátové funkce vzhledem k parametrům je drahá operace. Výpočet gradientu zahrnuje výpočet dopředné propagace v našem obrázku zleva doprava rozvinutého grafu následovaný zpětnou propagací zprava doleva skrz uvedený graf. Časová náročnost je $O(\tau)$ a není možné ji snížit paralelizací, jelikož dopředný průchod je sekvenční a závislý na předchozích hodnotách. Algoritmus zpětného šíření aplikovaný na rozvinutý graf s cenou $O(\tau)$ se nazývá zpětné šíření časem (back-propagation through time, BPPT).

OBRAZEK, DL str 373

DEEP LEARNIN—G

9.2.1 Backpropagation through time

Výpočet gradientu skrze RNN je přímočarý. Jedná se pouze o generalizovanou aplikaci algoritmu zpětného šíření nad rozvinutým výpočetním grafem. Gradienty získané pomocí algoritmu zpětné šíření pak mohou být využity s libovolnou technikou založenou na gradientu pro trénování RNN.

Uzly rozvinutého grafu obsahují parametry U, V, W, b a c a sekvenci uzlu indexovaných časem t pro $x^{(t)}$, $h^{(t)}$, $o^{(t)}$ 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:

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

V této derivaci předpokládáme, že výstupy $o^{(t)}$ jsou použity jako arguemnt softmax funkce k tomu, abychom získali vektor pravděpodobností výstupu $\hat{y}$. Dále předpokládáme, že ztráta je definována jako negative log-likelyhood skutečných targetu $y^{(t)}$ pro dosud dodaný vstup. Gradient $\nabla_{o^{(t)}} L$ nad všemi výstupy v čase t pro všechna i, t vypadá následovně:

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

Pokračujeme dále s výpočtem gradientu přes další uzly počínaje koncem sekvence. Ve finálním časovém kroku τ, $\boldsymbol{h}^{(\tau)}$ má jako následníka pouze $o_{(\tau)}$, tudíž výpočet gradientu je jednoduchý:

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

Nyní již můžeme iterovat zpět v čase a zpětně šířit gradienty od $t = \tau - 1$ až k $t = 1$ s tím, že $h^{(t)}$ (pro $t < \tau$) má jako své předchůdce $o^{(t)}$ a $h^{(t+1)}$. Gradient skryté vrstvy je pak

$$\nabla_{h^{(t)}} L = \left(\frac{\partial h^{(t+1)}}{\partial h^{(t)}}\right)^\top (\nabla_{h^{(t+1)}} L) + \left(\frac{\partial o^{(t)}}{\partial h^{(t)}}\right)^{top} (\nabla_{o^{(t)}} L) = W^\top (\nabla_{h^{(t+1)}} L) \mathrm{diag}\left(1 - (h^{(t+1)})^2\right) + V^\top (\nabla_o^{(t)} L)$$

kde $\mathrm{diag}\left(1 - (h^{(t+1)})^2\right)$ je Jakobián hyperbolického tangensu pro skryté jednotky i v čase $t + 1$.

Jakmile máme spočtené gradienty stavů, můžeme dopočítat gradienty parametrů. Vzhledem k tomu, že parametry jsou sdílené mezi jednotlivými časovými kroky, zavedeme nové proměnné $W^{(t)}$, resp. $U^{(t)}$, které jsou kopiemi váhových matic W, resp. U a značí, jakou mírou tyto váhy přispívají ke gradientu v čase t. Gradienty parametrů tedy můžeme určit jako

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

$$\nabla_b L = \sum_t \left(\frac{\partial h^{(t)}}{\partial b^{(t)}}\right)^\top \nabla_{h^{(t)}} L = \sum_t \mathrm{diag}\left(1 - (h^{(t)})^2\right) \nabla_{h^{(t)}} L$$

$$\nabla_V L = \sum_t \sum_t \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_V o_i^{(t)} = \sum_t (\nabla_{o^{(t)}} L) h^{(t)\top}$$

$$\nabla_W L = \sum_t \sum_t \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{W^{(t)}} h_i^{(t)} = \sum_t \mathrm{diag}\left(1 - (h^{(t)})^2\right) (\nabla_h^{(t)} L) h^{(t-1)\top}$$

$$\nabla_U L = \sum_t \sum_t \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{u^{(t)}} h_i^{(t)} = \sum_t \mathrm{diag}\left(1 - (h^{(t)})^2\right) (\nabla_{h^{(t)}} L) x^{t\top}$$

ZKONTROLOVAT VZORCE a ZAROVNAT VLEVO

DEEP LEARNING

9.3 Bidirectional RNN

Zatím jsme se věnovali pouze rekurentním sítím, které zachycovaly do stavu v čase t pouze informaci z minulosti, tedy informace pro vstupy $x^{(1)}, \ldots, x^{(t-1)}$ a aktuální vstupu $x^{(t)}$. Nicméně v mnoha aplikacích výstupní predikce $y^{(t)}$ může záviset na celé sekvenci. Například při zpracování řeči je běžné, že správná interpretace fonémů závisí kvůli jevu koartikulace na minulých fonémech i na budoucích a na svém blízkém kontextu. Tento problém řeší obousměrné rekuretní neuronové sítě.

Obousměrná RNN kombinuje dvě RNN, kdy jedna se pohybuje v čase kupředu, od začátku sekvence, zatímco druhá RNN se pohybuje v čase zpět, od konce sekvence. Tyto sítě mezi sebou sdílí skryté stavy $h^{(t)}$ (síť dopředná) a $g^{(t)}$ (síť zpětná), což umožní, že výstupní jednotky $o^{(t)}$ závisí jak na minulosti, tak na budoucnosti, ale zároveň jsou nejvíce citlivé na vstupní hodnoty okolo času t.

OBRAZEK z DL str. 389 plus popisek

DEEP LEARNING

9.3.1 Long short-term memory network

Rekuretní sítě jsou velice náchylné na problém vanishing a exploding gradientu. Hlavně v případech dlouhodobých závislostí dochází k velmi malým hodnotám vah vah, které jsou poté násobeny mnoha Jakobiány. Násobíme-li váhu w několikrát samu s sebou, dojde buď k jejímu téměř vynulování/vymizení či k explozi v závislosti na řádu w. Obvzál˘ět pro dlouhodobé vzdálenosti je řád těchto vah a gtadientu exponencielně menší než u krátkodobích vzdáleností. Jednou z možností, jak řešit problém obtížného trénování sítí s dlouhodobými závilosti je využití architektury dlouhých sítí s krátkodobou pamětí (long short-term memory network, dále jen LSTM) nebo rekurerní jednotku s branami (gated reccurent unit, dále jen GRU)

OBRAZKY OD COLAHAc NEBO z DL str 405?

Základním principem LSTM a zároveň jejím největším přínosem je zavedení smyček (self-loop), které vytvoří cesty, kudy může gradient po dlouho dobu proudit (flow). Váha těchto smyček je pak trénována a měněna (conditioned) v závislosti na kontextu. Díky tomu, že je tato váha opatřena tzv. bránou (gated), tedy řízena jinou skrytou jednotkou, může být časová integrace této informace měněna dynamicky. Díky této vlastnosti se LSTM stala velice úspěšnou pro úlohy jako například rozpoznání psaného písma, stro-

jového překladu, rozpoznávání řeči či k popisu obrázků.

Rovnice pro dopředné šíření pro "mělkou síť"jsou uvedeny níže. Nicméně byly testovány i hluboké sítě (Graves), které ukázali velkou reprezentativní schopnost.

POPIS diagramu ze strany 405: Jednotlivé jednotky v LSTM jsou nazývány buňkami a jsou mezi sebou rekurentně propojeny. Vstupní příznak/hodnota je spočten(a) obyčejným perceptronem. Tato hodnotam může být akumulována ve stavu, pokud to sigmoidální vstupní brána umožní. Stavová jednotka má lineární smyčku, jejíž váha je řízena zapomínací bránou. Výstup buňky může být vypnut pomocí výstupní brány. Všechny jednotky s bránou mají sigmoidální nelinearitu zatímco vstupní jednotka může mít libovolný typ nelineariy. Stavová jednotka může být také využita jako vstup navíc pro ostatní jednotky s bránou.

Místo jednotek, které jednoduše aplikují nelinearity po prvcích k afinní tranformaci vstupu a rekurentních jednotek, LSTM rekurentní sítě mají tzv. "LSTM buňky", které mají kromě vnější rekurence RNN i interní rekurenci - smyčku (selfloop). Každá buňka má stejné vstupy a výstupy jako obyčejná rekurentní síť ale také má více parametrů a systém jednotek opatřených bránou (gating units), které řídí tok informací. Nejdůležitější komponent je stavová jednotka (state unit) $s_i^{(t)}$, která je opatřena lineární smyčkou, jejíž váha je řízena jednotkou se zapomínací bránou (forget gate unit) $f_i^{(t)}$ (pro čas t a buňku i), která nastavuje hodnoty této váhy na hodnoty mezi 0 a 1 pomocí jednotky se sigmoid aktivační funkcí (sigmoid unit):

$$f_i^{(t)} = \sigma\left(b_i^{(t)} + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\right),$$

kde $x^{(t)}$ je aktuální vstupní vektor a $h^{(t)}$ je aktuální vektor skryté vrstvy obsahující výstupy všech LSTM buňek a b^f, U^f, W^f jsou biasy, váhy vstupu (input weights) a rekurentní váhy (recurrent weights) pro zapomínací brány. Interní/vnitřní stav LSTM buňky je pak updatován následovně, ale s conditional smyčkou váhou $f_i^{(t)}$:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \sum_j U_{i,j} x_j^{(t)} + sum_j W_{i,j} h_j^{(t-1)}\right),$$

kde b, U a W značí biasy, vstupní váhy a rekurentní váhy v LSTM buňce. Jednotka s vnější vstupní bránou (external input gate unit) $g_i^{(t)}$ je počítána podobně jako zapomínací

brána (se sigmoidální jednotkou k zisku hodnoty brány (gating value) mezi 0 a 1), ale s vlastními parametry:

$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\right).$$

Výstup LSTM buňky $h_i^{(t)}$ může být vypnut pomocí výstupní brány $q_i^{(t)}$, která také využívá sigmoidální jednotku pro nastavení hodnoty brány (gating.. propustnost?):

$$h_i^{(t)} = \tanh(s_i^{(t)}) q_i^{(t)},$$

$$q_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\right),$$

kde paramtery b^o, U^o, W^o jsou biasy, vstupní výhy a rekurentní váhy.

Ukázalo se, že LS—TM sítě se snadno dokáží naučit (nebo lépe než ostastní) dlouhodobé závislosti v datech.

DEEP LEARNING

9.3.2 Gated Recurrent Units

Ačkoliv architektura LSTM je výrazným vylepšením běžných RNN, ne všechny její prvky jsou nezbytně nutné. Proto byly vyvinuty tzv. rekurentní jednotky s branami (gated recurrent unit, dále jen GRU). Hlavním rozdíl oproti LSTM je, že jediná jednotka s branou (gating unit) současně řídí jak zapomínací faktor, tak i rozhodnutí k update stavové jednotky. Updatovací rovnice vypadají následovně:

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + (1 - u_i^{(t-1)}) \left(b_i + \sum_j U_{i,j} x_j^{(t-1)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)}\right)$$

kde u značí updatovací bránu a r značí resetovací bránu. Jejich hodnoty jsou definovány jako:

$$u_i^{(t)} = \sigma\left(b_i^u + \sum_j U_{i,j}^u x_j^{(t)} + \sum_j W_{i,j}^u h_j^{(t)}\right)$$

a

$$r_i^{(t)} = \sigma\left(b_i^r + \sum_j U_{i,j}^r x_j^{(t)} + \sum_j W_{i,j}^r h_j^{(t)}\right).$$

Resetovací a updatovací brány mohou samostatně "ignorovat"části stavového vektoru.

Updatovací brány se pak mohou vytvořit kopii libovolné dimenze (jeden extrém sigmodiy) či kompletně ignorovat (druhý extrém) tím, že nahradí stav novým cílovým stavem. Resetovací brána pak řídí, které části stavu budou použity pro výpočet příštího cílového stavu. T—ím zavádí nový nelineární efekt ve vztahu mezi minulým a budoucím stavem.

DEEP LEARNING

9.4 ADAM

Jelikož konstanta učení je parametr, který je nejtěžší nastavit ručně správně a který má největší vliv jak na rychlost, tak na konvergenci trénovaní, výzkumnící se dlouho zabývali metodami, jak provádět toto nastavování ručně. Cena/ztráta je velmi často citlivá na určité směry v prostoru paraemtrů na necitlivá na jiné. Momentum tento problém do určité míry řeší, ale za cenu zavedení nového hyperparametru. Bylo by tedy vhodné vytvořit vlastní konstantu učení pro každý parametr a každou z těchto konstant automaticky adaptovat během učení. V této práci byla využita metoda ADAM, která je společně s RMSprop jednou z nejpoužívanšějších adaptivních optimalizačních metod.

Název ADAM je odvozen ze slov adaptive moment (adaptivní moment) a lze na něj nahlížet jako na RMSprop s momentem. Momentum je zavedeno přímo do odhadu momentu prvního řádu (prvního momentu?) gradientu s exponenciálním zapomínáním/vážením. Nejvíce přímočará cesta, jak přidat momentum k RMSprop je přidání k přeškálovaným gradientům. ADAM dále provádí korekci biasu odhadu jak prvního centrálního momentu, tka druhého necentrálního momentu, aby bral v úvahu počáteční inicializaci. ADAM je jedním z nejrobustnějším optimalizačních algoritmů pro deep learning, ale občas vyžaduje změnu počátečního learning ratu oproti doporučeným hodnotám.

předběžný update

$$\tilde{\theta} \leftarrow \theta + \alpha v$$

výpočet gradientu

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

akumulace gradientu

$$r \leftarrow \rho r + (1 - \rho) g \odot g$$

výpočet updatu rychlosti/velocity

$$v \leftarrow \alpha v - \frac{\epsilon}{\sqrt{r}} \odot g$$

aplikace updatu

$$\theta \leftarrow \theta + v$$

DEEP LEARNING, str. xyz

10 Kapacita modelu, underfitting a overfitting

Základní vlastností každého modelu ve strojovém učení by měla být generalizace. To znamená, že model musí správně predikovat i většinu nových, dříve neviděných pozorování a nejen ta pozorování, na kterých byl natrénován. Při trénování modelu tedy rozdělíme datovou sadu na tři: trénovací, validační a testovací. Cílem je, abychom po natrénovaní modelu dostali obdobnou trénovací, validační i testovací chybu. Při rozdělení datové sady předpokládáme, že jednotlivá pozorování jsou navzájem nezávislá a všechny tři sady podléhají stejnému rozložení. Kvalitní model by tedy měl splňovat tyto vlastnosti:

1. musí minimalizovat chybu na trénovací sadě

2. musí minimalizovat rozdíl mezí chybou na trénovací a testovací sadou

Při optimalizaci parametrů ovšem často dochází ke dvěma nežádoucím jevům: underfitting a overfitting. K underfittingu dochází, pokud model není schopný dostatečně minimalizovat chybu na trénovací sadě, tj. není schopný se naučit informaci obsaženou v trénovacích datech. K overfittingu naopak dochází, pokud model není schopný splnit druhou podmínku - minimalizaci rozdílu chyby mezi trénovací a testovací sadou, tj. model si spíše zapamatuje trénovací data místo toho, aby se naučil souvislosti v datech a na nových, dosud neviděných pozorováních selhává.

OBRAZEK NA POLYNOMECH

Oba tyto jevy můžeme ovládat pomocí tzv. kapacity modelu. Modely s nízkou kapacitou jsou náchylné na underfitting a naopak modely s vysokou kapacitou jsou náchylné na overfitting. Jedna z možností, jak ovládat kapacitu modelu, je pomocí vhodného výběru prostoru hypotéz neboli omezením výběru možných funkcí, které model může využít

(např. počet stupňů volnosti u lineární regrese). Omezením výběru funkce zároveň dochází k omezení počtu využitelných příznaků a snížení počtu volných parametrů, které model musí optimalizovat.

OBRAZEK OPTIMALNI KAPACITY (DL, 112)

Je zřejmé, že neexistuje jediný model, který dokáže řešit libovolný problém - vždy je potřeba zvolit vhodný model pro danou úlohu a odladit jeho parametry.

Deep Learning, 107-115

TODO: generalizace pomocí snížení počtu parametrů

10.1 Regularizace

Jedním ze způsobů, jak navýšit generalizaci modelu bez ovlivnění kapacity modelu, je regularizace. Regularizace je modifikace učícího se algoritmu, která snižuje generalizační chybu, ale zároveň nesnižuje chybu trénovací. Regularizaci lze implementovat mnoha způsoby a opět je třeba vybrat vhodnou metodu pro daný problém.

Deep Learning, 117

Hlavní problémem strojového učení je, jak zařídit, aby algoritmus fungoval dobře nejen na nových datech, ale také na nových, dosud neviděných vstupech. Mnoho strategiií ve strojovém učení jsou explicitně navrženy tak, aby snížovaly chybu na testovací sadě, ovšem i za ceny vyšší chyby na trénovací sadě.

Existuje mnoho metod regularizace - některé jsou založeny na zavedení omezujících podmínek modelu/hodnot jeho paraemtrů, jiné zase zavádějí nové člney do objective function. Tato omezení a penalizace často vycházejí z aprioirní znalosti problému, ale kolikrát jen zavádějí preferenci po jednodušší třídě modelu za účelem vyšší generalizace.

V deep learningu většina regularizačních strategií regularizuje estimátory pomocá trade-offu vyšší bias za nižší variance. Cílem regularizace je snižení overfittingu tak, aby průběh naučené funkce odpovídal skutečnému trendu dat. (tuhle sračku dhodně přepsat).

10.2 Penalizace normy parametrů

Populární metodou pro regularizaci jsou penalizační metody a to zejména L1 a L2 penalizace. Tyto penalizační metody omezují kapacitu modelu přidáním penalizace $\Omega(\theta)$ k objective function J, která odpovídá normě parametrů. Regularizovaná objective funtion

pak vypadá následnovně:

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta),$$

kde $\alpha \in [0, \infty)$ je hyperparametr, který váží relativní kontribuce penalizačního členu Ω relativně je standardní funkci J. Pokud $\alpha = 0$, poté nedochází k žádné regularizaci a vyšší hodnout regularizace roste.

Když pak trénovací algoritmus minimalzucje regularizavnou objective function $\tilde{J}$, snižuje hosnotu jak originální objec. func. J na trénovacích datech, tak i nějakou metriku velikosti parametrů θ. Různé volby normy Ω pak vedou na preferenci různých rešení. U neuronových sítí regularizujeme pouze váhy a biasy zůstavají bez regularizace.

DEEP LEARNING

10.3 Noise injection

Další metoda regularizace, která neovlivňuje model samotný a byla využita jako primární metoda regularizace v této práci, je augmentace trénovacích dat ve formě aditivního šumu. Ačkoliv neuronové sítě nejsou moc robustní vůči šumu, pro většinu klasifikačních i regresních úloh schopny řešit úlohy i po zatížení malým náhodným šumem přidaným ke vstupním datům. Pro některé modely je přidání šumu ekvivalentní přidání penalizace normy vah.

DEEP LEARNING

10.4 Early Stopping

Při trénování velkých modelů (s vysokou hloubkou, šířkou či obojím) s dostačující reprezentační kapacitou k přetrénovaná nad úlohou můžeme často sledovat, že chyba na térnvoací sadě pomalu a jistě během času klesá, zatímco chyba na validaeční sadě začne stoupat.

OBRAZEK

Z rtohoto tedy můžeme usoudit, že můžeme získat lepší model s lepší validační chybou (a doufejme i s lepší testovací chybou), pokud se vrátíme k parametrům v čase s nejnižší validační chybou. Pokaždé, když se chyba na validační sadě sníží, uložíme si kopii parametrů modelu. Jakmile trénování skončí, vrátíme se k nejlepšímu souboru pa-

rametrů místo ponechání posledních. Algoritmus končí, pokud se žádná změna parametrů nepřekoná dosud nejlepší validační chybu pro předdefinovanou počet iterací.

Algoritmus předčasného zastavení je forma regularizaxce, která nevyžaduje téměř žádné změny trénovací procesu, modifikaci ovbjective function nebo souboru možných hodnot parametrů. Je tedy snadné tento alogritmus použít bez poškození dynamiky učení. Early stopping lze využít budď samosratně nebo spolu s výše zmíněnými metodami regularizace.

DEEP LEARNING

10.5 Dropout

Dropout nabízí výpočetně nenáročné ale výkonnou metodu regularizace pro širokou škálu modelů. Trénování s dropoutem funguje nejlépe pro učící algorimty založených na minibachtchi, který dělá malé kroky, např. SGD. Pokáždé, kdy pozorování vstoupí do minibatche, náhodným výběrem (randomly sample) je vytvořená different binární maska, která je aplikována na všechny vstupy skrytých jednotek v síti. Tato maska je pro každou jednotku samplována nezávisle na ostatních. Pravděpodobnost samplingu hodnoty jedna v masce (způsobijící, že jednotka bude zahrnuta do trénování) je hyperparametr,k terý se určuje před trénováním sítě. Většinou se volí pravděpobnost 0.8, že vstupní jednotku bude použita a ppst 0.5, že skrytá jednotka bude použita. Poté trénování pokračuje jak obvykle - dopředné šíření, backprop, zpětné šíření.

Jednou z hlavních východ kromě výpočetní a pamětové nenáročnosti je, že nijak výrazně neomezuej výběr modelu a trénovací proceduru, která může být použita. Funguje dobře téměř s každým modelem„ který lze trénovat pomocí SGD (včetně rekurntních neuronových sítí, kde je třeba dropout zobecnict na rekurentní spojení jednotek)-

Největší síla dropout vychází z toho, že maskování vnáší jistou formu šumu do skrytých jednotek. Lze na něj nahlížet jako na adaptivní destrukci informačního obsahu vstupu než-li na destrukci raw values inputu. Tím zamezuje, aby se určitá jednotka zaměřila na rozpoznávání jednoho signifikatního příznaku, zatímco jiná jednotka by byla téměř nevyužita. Nághodnou deaktivací jednotek tak donutí ostatní jednotky, aby se také naučili tuto informaci.

Dropout opět může být využit spolu s ostatními metodami regularizace.

DEEP LEARNING

11 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

11.1 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

11.2 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

11.3 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

11.4 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

11.4.1 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 $\alpha_s^{(t)}$?

12 Klasifikace fonémů

13 Vyhodnocení

14 Závěr

15 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 [6].

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 [7, 8].

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.

16 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 [9].

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.

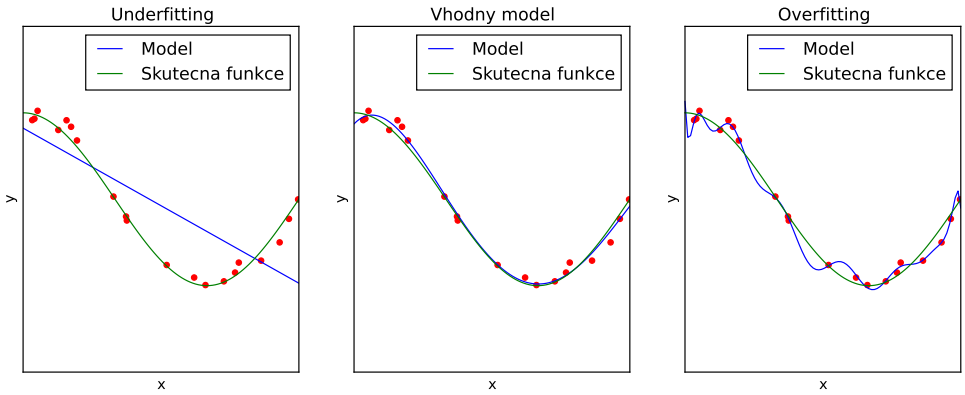
Učící se klasifikátor pracuje ve dvou fázích - trénovací fáze a klasifikační fáze:

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 [10, 11].

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.) [12]. 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 1.



Obrázek 1: Příklad overfittingu a underfittingu.

17 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 [9].

17.1 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:

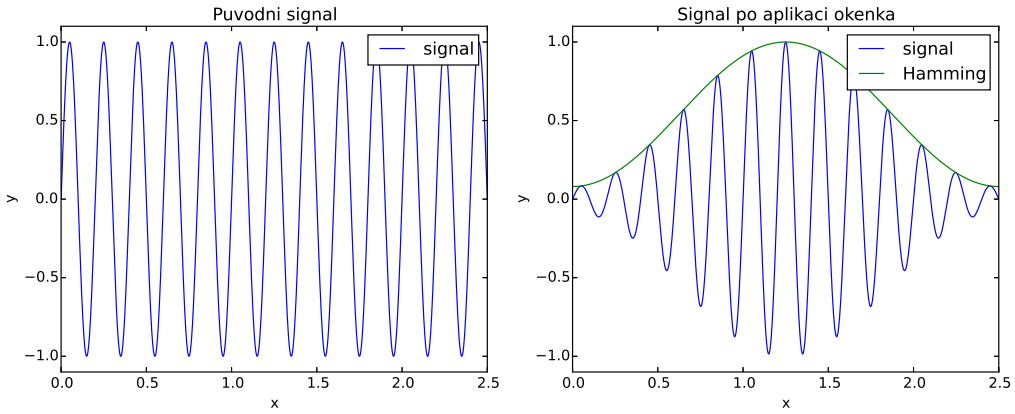
- **pravoúhlé okénko** - pravoúhlé okénko přidělí všem vzorkům mikrosegmentu stejnou váhu. Lze jej definovat vztahem

$$w(n) = \begin{cases} 1 & \text{pro } 0 \leq n \leq N-1 \\ 0 & \text{jinak,} \end{cases} \tag{17.1}$$

  kde N je počet vzorků mikrosegmentu.

- **Hammingovo okénko** - Hammingovo okénko je vhodné použít v případě, kdy je třeba potlačit vzorky na krajích zpracovávaného mikrosegmentu [9, 13]. Lze jej definovat vztahem

$$w(n) = \begin{cases} 0.54 - 0.46 \cos\left(\dfrac{2\pi n}{N-1}\right) & \text{pro } 0 \leq n \leq N-1 \\ 0 & \text{jinak.} \end{cases} \tag{17.2}$$



Obrázek 2: Aplikace Hammingova okénka na funkci $f(x) = \sin(10\pi x)$.

17.2 Příznaky v časové a frekvenční oblasti

17.2.1 Krátkodobá energie signálu

Funkce krátkodobé energie signálu je definována vztahem

$$E_n = \sum_{k=-\infty}^{\infty} [s(k)w(n-k)]^2, \tag{17.3}$$

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.

17.2.2 Krátkodobá intenzita signálu

Funkce krátkodobé intenzity signálu je definována vztahem

$$I_n = \sum_{k=-\infty}^{\infty} |s(k)|w(n-k). \tag{17.4}$$

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

17.2.3 Krátkodobé průchody nulou

Funkce krátkodobých průchodů nulou je definována vztahem

$$Z_n = \frac{1}{2} \sum_{k=-\infty}^{\infty} |sign\left[s(k)\right] - sign\left[s(k-1)\right]| w(n-k). \tag{17.5}$$

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) [9, 14].

17.2.4 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

$$f_m = 2595 \log_{10}\left(\frac{f}{700}\right). \tag{17.6}$$

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

$$b_{m,i} = b_{m,i-1} + \Delta_m, \tag{17.7}$$

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

$$u(f,i) = \begin{cases} \dfrac{1}{b_i - b_{i-1}}(f - b_{i-1}) & \text{pro } b_{i-1} \leq f < b_i \\ \dfrac{1}{b_i - b_{i+1}}(f - b_{i+1}) & \text{pro } b_i \leq f < b_{i+1} \\ 0 & \text{jinak} \end{cases} \tag{17.8}$$

a odezvy jednotlivých filtrů $y_m(i)$ vztahem

$$y_m(i) = \sum_{f=b_{i-1}}^{b_{i+1}} |S(f)| u(f,i), \tag{17.9}$$

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

$$c_m(j) = \sum_{i=1}^{M^*} \log y_m(i) \cos\left(\frac{\pi j}{M^*}(i - 0.5)\right) \quad \text{pro } j = 0, 1, \ldots, M, \tag{17.10}$$

kde M je počet mel-frekvenčních kepstrálních koeficientů [14, 15].

17.3 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

$$d_t = \frac{\sum_{n=1}^{N} n(c_{t+n} - c_{t-n})}{2\sum_{n=1}^{N} n^2}, \tag{17.11}$$

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 [14, 15].

17.4 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 $\sigma = 1$.

$$z = \frac{x - \mu}{\sigma} \tag{17.12}$$

- **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).

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{17.13}$$

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].

18 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 [17, 18].

18.1 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

$$b + w_1 x_1 + w_2 x_2 + \ldots w_p x_p = 0, \tag{18.1}$$

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

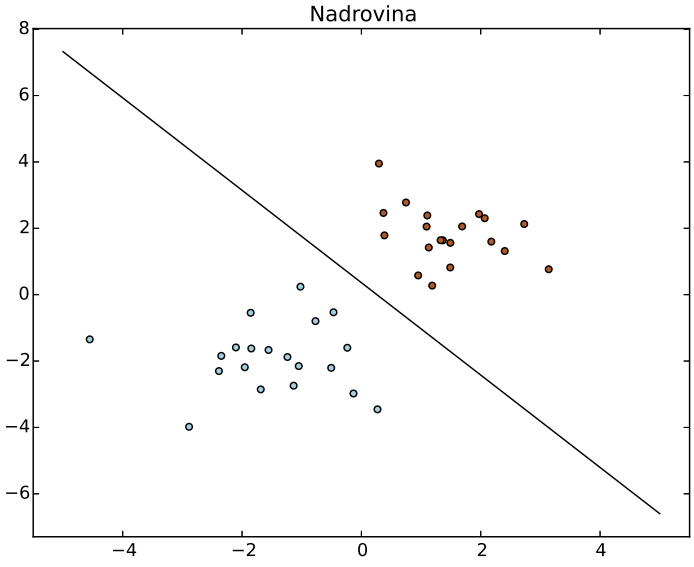
V případě, že bod x nevyhovuje rovnici 18.1, tedy

$$b + w_1 x_1 + w_2 x_2 + \ldots w_p x_p > 0, \tag{18.2}$$

bod leží na jedné straně poloroviny, resp. pokud

$$b + w_1 x_1 + w_2 x_2 + \ldots w_p x_p < 0, \tag{18.3}$$

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 18.1 [19].



Obrázek 3: Nadrovina oddělující body ve dvoudimenzionálním prostoru [20].

18.2 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

$$x_i \cdot w + b \geq +1 \quad \text{pro } y_i = +1, \tag{18.4}$$

$$x_i \cdot w + b \leq -1 \quad \text{pro } y_i = -1, \tag{18.5}$$

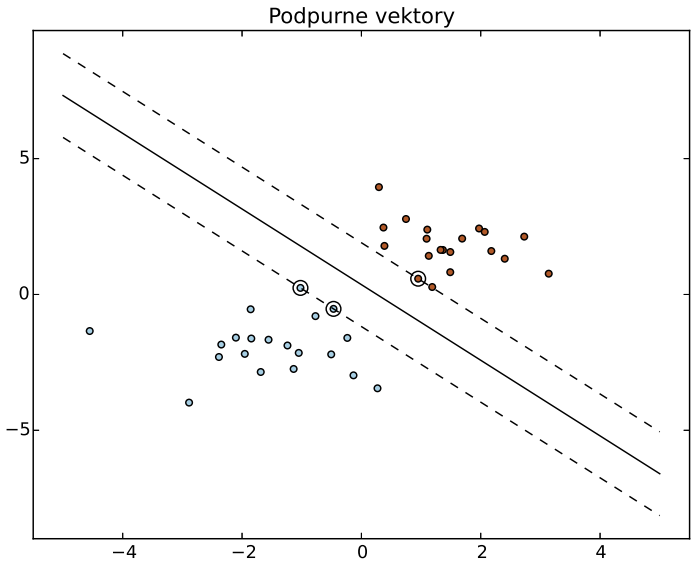
které lze sloučit do jedné množiny nerovností

$$y_i(x_i \cdot w + b) - 1 \geq 0 \quad \forall i. \tag{18.6}$$

Body, pro které v nerovnici 18.4 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 18.5 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 4) a oddělující rovina je na nich přímo závislá. Pokud dojde k jejich pohybu nebo odstranění, změní se i výsledné řešení.



Obrázek 4: Vizualizace podpůrných vektorů [20].

Formulujme si nyní tento problém pomocí Lagrangeových multiplikátorů α_i, $i = 1, 2, \ldots, l$ - jeden pro každou nerovnost 18.6, neboli jeden pro každý prvek trénovací množiny $\{x_i, y_i\}$. Dostáváme Lagrangeovu funkci

$$L_P \equiv \frac{1}{2}||w||^2 - \sum_{i=1}^{l} \alpha_i y_i (x_i \cdot w + b) + \sum_{i=1}^{l} \alpha_i. \tag{18.7}$$

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

$$w = \sum_i \alpha_i y_i x_i, \tag{18.8}$$

$$\sum_i \alpha_i y_i = 0. \tag{18.9}$$

Dosazením rovnic 18.8 a 18.9 do 18.7 dostaneme

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j. \tag{18.10}$$

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 $\operatorname{sgn}(w \cdot x + b)$ [17].

18.3 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 18.4 a 18.5. Toho dosáhneme zavedením volných (slack) proměnných ε_i, $i = 1, 2, \ldots, l$

$$x_i \cdot w + b \geq +1 - \varepsilon_i \quad \text{pro } y_i = +1 \tag{18.11}$$

$$x_i \cdot w + b \leq -1 + \varepsilon_i \quad \text{pro } y_i = -1 \tag{18.12}$$

$$\varepsilon_i \geq 0 \quad \forall i. \tag{18.13}$$

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

$$L_D \equiv \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \tag{18.14}$$

za podmínek

$$0 \leq \alpha_i \leq C, \tag{18.15}$$

$$\sum_i \alpha_i y_i = 0. \tag{18.16}$$

Řešením je pak

$$w = \sum_{i=1}^{N_S} \alpha_i y_i x_i, \tag{18.17}$$

kde N_S je počet podpůrných vektorů.

Primární úloha je dána Lagrangeovou funkcí

$$L_P = \frac{1}{2}||w||^2 + C \sum_i \varepsilon_i - \sum_i \alpha_i \{y_i(x_i \cdot w + b) - 1 + \varepsilon_i\} - \sum_i \mu_i \epsilon_i, \tag{18.18}$$

kde μ_i jsou Lagrangeovské multiplikátory zajištující kladnost ε_i. Řešitelnost 18.18 je dána Karush-Kuhn-Tucker podmínkami (více v [17]). Pro výpočet prahu b postačí pouze dvě z Karush-Kuhn-Tucker podmínek

$$\alpha_i \{y_i(x_i \cdot w + b) - 1 + \varepsilon_i\} = 0, \tag{18.19}$$

$$\mu_i \epsilon_i = 0. \tag{18.20}$$

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 [17].

18.4 Nelineárně separabilní případ

Uvažujme nyní případ, kdy oddělující nadrovina není lineární funkcí trénovacích dat - potřebujeme tedy zobecnit rovnice 18.14, 18.15 a 18.16 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)

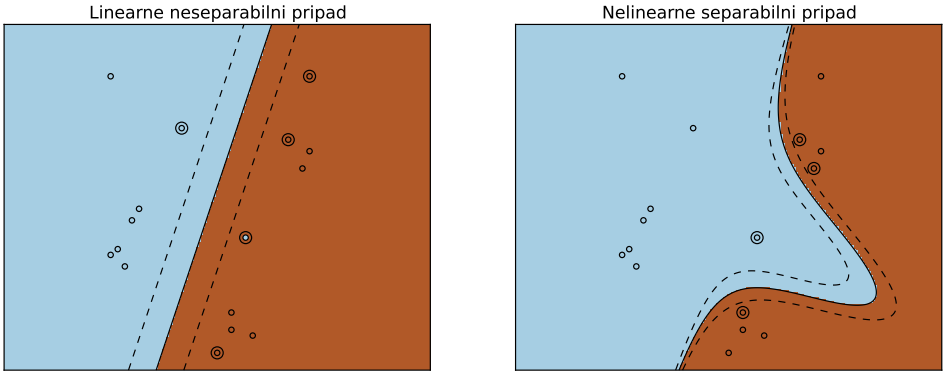
$$\Phi : \mathbb{R}^n \mapsto \mathcal{H}. \tag{18.21}$$

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 18.14 a opět dopočítat normálu w oddělující nadroviny. Nemusíme tedy explicitně znát zobrazení Φ, pouze jádrovou funkci.

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

$$f(x) = \sum_{i=1}^{N_S} \alpha_i y_i \Phi(s_i) \cdot \Phi(x) + b = \sum_{i=1}^{N_S} \alpha_i y_i K(s_i, x) + b, \tag{18.22}$$

kde s_i jsou podpůrné vektory [17, 21].



Obrázek 5: SVM s lineární a polynomiální jádrovou funkcí [20].

18.5 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 18.22 nejvyšší hodnoty (tj. bod leží nejdále od oddělující nadroviny a byl zaklasifikován s největší "jistotou") [18,21].

19 Neuronové sítě

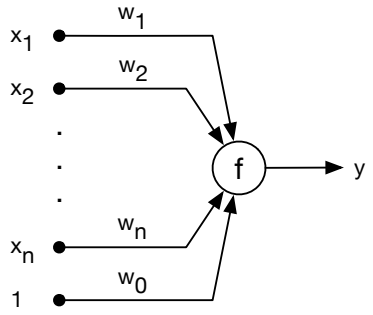
Neuronová síť je algoritmus inspirovaný funkcí neuronů a jejich propojením v lidském mozku. Skládá se z mnoha výpočetních jednotek (neurony) propojených pomocí numerických parametrů (synapse) a úpravou těchto parametrů je schopna se učit. Tato práce je omezena pouze na dopředné neuronové sítě a jejich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

19.1 Perceptron a aktivační funkce

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

$$y = f\left(\sum_{i=1}^{n} w_i x_i + w_0\right) = f(w^T x + w_0), \tag{19.1}$$

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 6: Perceptron.

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

$$w = [w_0, w_1, \ldots, w_n]^T, \tag{19.2}$$

$$x = [1, x_1, \ldots, x_n]^T. \tag{19.3}$$

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.

$$f(\xi) = \frac{1}{1 + e^{-\xi}} \tag{19.4}$$

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

$$f(\xi) = \tanh(\xi) = \frac{2}{1 + e^{-2\xi}} - 1 \tag{19.5}$$

- **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.

$$f(\xi) = \max(0, \xi) \tag{19.6}$$

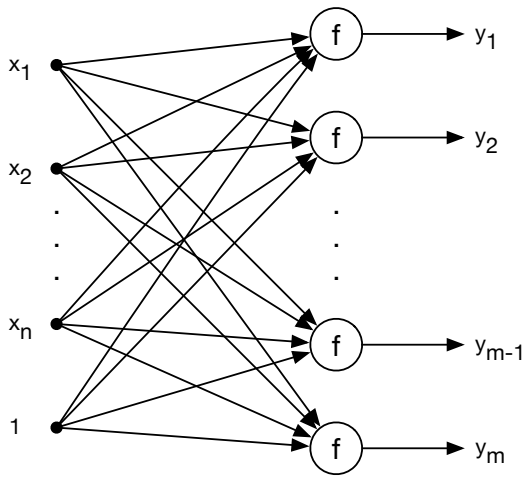
- **Maxout** - Maxout je generalizací aktivační funkce ReLU. Na rozdíl od výše zmíněných aktivačních funkcí nemá stejný funkcionální tvar $f(w^T x + b)$, ale počítá hodnotu funkce $\max(w_1^T x + b_1, w_2^T x + b_2)$. Maxout řeší nedostatky ReLU, ovšem za cenu zdvojnásobení počtu parametrů pro každý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

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

$$f(\xi) = \text{sign}(\xi) = \begin{cases} 1 & \text{pro } \xi \geq 0 \\ 0 & \text{jinak} \end{cases} \tag{19.7}$$

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 [10, 22].



Obrázek 7: Jednovrstvá neuronová síť s n vstupy, aktivační funkcí $f(\cdot)$ a m výstupy.

19.2 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

$$w_i(k+1) = w_i(k) + \Delta w_i(k) = w_i(k) + \alpha(t-y)x_i, \tag{19.8}$$

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

$$E(w) = \frac{1}{2}\sum_{i=0}^{n}(t_i - y_i)^2, \tag{19.9}$$

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

$$\nabla E = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right]. \tag{19.10}$$

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

$$w(k+1) = w(k) + \Delta w(k) = w(k) - \alpha \nabla E(w) = w(k) - \alpha(t-y)x^T. \tag{19.11}$$

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 [22].

19.3 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 8). 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

$$net_j = \sum_{i=1}^{I} x_i w_{ji} + x_0 w_{j0} = \sum_{i=0}^{I} x_i w_{ji} = w_j^T x, \tag{19.12}$$

$$z_j = f(net_j), \tag{19.13}$$

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

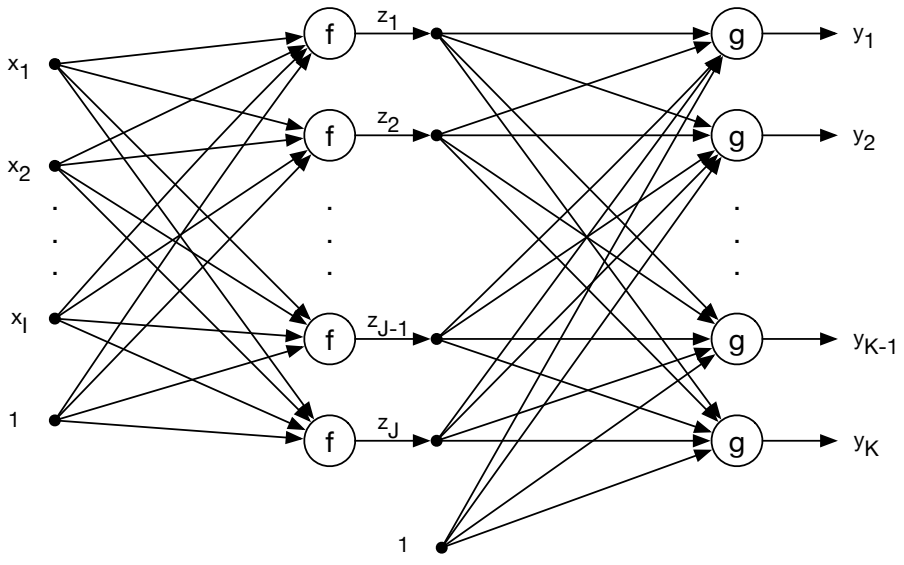
$$net_k = \sum_{j=1}^{J} z_j w_{kj} + z_0 w_{k0} = \sum_{j=0}^{J} z_j w_{kj} = w_k^T z, \tag{19.14}$$

$$y_k = g\left(net_k\right). \tag{19.15}$$

Úpravou lze k-tý výstup zapsat jako

$$y_k = g\left(\sum_{j=1}^{J} w_{kj} f\left(\sum_{i=1}^{I} x_i w_{ji} + x_0 w_{j0}\right) + w_{k0}\right). \tag{19.16}$$

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



Obrázek 8: Dvouvrstvá neuronová síť.

19.4 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

$$E(w) = \frac{1}{2}\sum_{k=1}^{K}(t_k - y_t)^2 = \frac{1}{2}(t-y)^2. \tag{19.17}$$

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

$$\Delta w = -\alpha\frac{\partial E}{\partial w}, \tag{19.18}$$

čímž opět získáme pravidlo pro modifikaci vah

$$w(k+1) = w(k) + \Delta w(k). \tag{19.19}$$

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.

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial w_{jk}} = \delta_k\frac{\partial net_k}{\partial w_{jk}} \quad \Rightarrow \quad \delta_k = -\frac{\partial E}{\partial net_k} \tag{19.20}$$

Hodnotu δ_k nazýváme citlivost neuronu a popisuje, jak se změní celková chyba při jeho aktivaci. Derivací rovnice 19.17 získáme hledané pravidlo

$$\Delta w_{jk} = \alpha\delta_k z_j = \alpha(t_k - y_k)f'(net_k)z_j. \tag{19.21}$$

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

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} \quad \Rightarrow \quad \delta_j = f'(net_j)\sum_{k=1}^{K} w_{kj}\delta_k, \tag{19.22}$$

$$\Delta w_{ji} = \alpha x_i\delta_j = \alpha x_i f'(net_j\sum_{k=1}^{K} w_{kj}\delta_k). \tag{19.23}$$

Podrobnější odvození algoritmu backpropagation lze nalézt v [11, 22].

19.4.1 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

$$\Delta w_{ji}(k) = \alpha x_i \delta_j + \mu \Delta w_{ji}(k-1), \tag{19.24}$$

kde $0 < \mu < 1$ je momentum a k je iterace učícího algoritmu. Změna vah tedy závisí i na změně vah v minulé iteraci [11, 22].

19.4.2 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 [11].

20 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 [14].

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.

20.1 Základní algoritmus

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

$$A = \{a_1, a_2, \ldots, a_n, \ldots, a_I\} \tag{20.1}$$

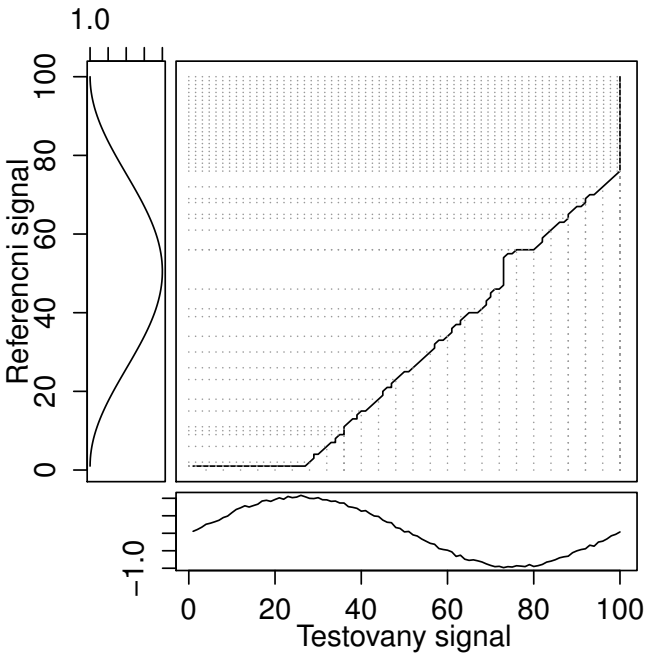
a obraz referenčního slova

$$B = \{b_1, b_2, \ldots, b_m, \ldots, b_J\}\,, \tag{20.2}$$

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

$$D(A,B) = \sum_{n=1}^{I} \hat{d}\left(a_n, b_{\Psi(n)}\right), \tag{20.3}$$

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



Obrázek 9: Průběh funkce DTW pro $sin(x)$ se šumem a $cos(x)$.

20.2 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

$$n = i(k), \tag{20.4}$$

$$m = j(k), \tag{20.5}$$

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

20.2.1 Omezení na hraniční body

Hraniční body funkce DTW jsou dány podmínkami

$$i(1) = 1 \quad i(K) = I, \tag{20.6}$$

$$j(1) = 1 \quad j(K) = J. \tag{20.7}$$

20.2.2 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

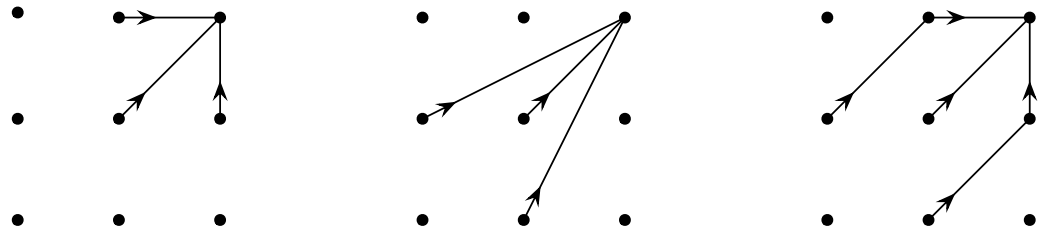
$$0 \leq i(k) - i(k-1) \leq \bar{I}, \tag{20.8}$$

$$0 \leq j(k) - j(k-1) \leq \bar{J}, \tag{20.9}$$

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.

20.2.3 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 10: Nejčastěji používaná lokální omezení (více v [9, 23]).

20.2.4 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

$$1 + \alpha\left[i(k) - 1\right] \leq j(k) \leq 1 + \beta\left[i(k) - 1\right], \tag{20.10}$$

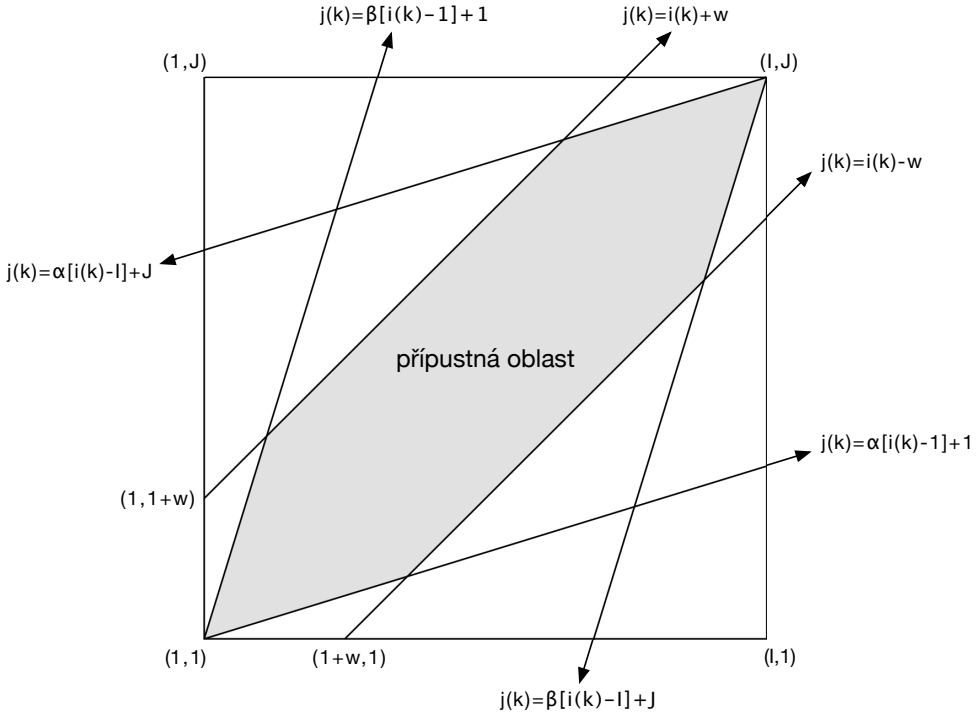
$$J + \beta\left[i(k) - I\right] \leq j(k) \leq J + \alpha\left[i(k) - I\right], \tag{20.11}$$

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

$$|i(k) - j(k)| \leq w, \tag{20.12}$$

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 11: Globální vymezení oblasti pohybu funkce.

20.3 Minimální vzdálenost

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

$$D(A,B) = \min_{\{i(k),j(k),K\}} \left[\frac{\sum_{k=1}^{K} d\left[i(k),j(k)\right] \hat{W}(k)}{N(\hat{W})} \right], \tag{20.13}$$

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

$$\hat{W}(k) = \left[i(k) - i(k-1)\right] + \left[j(k) - j(k-1)\right], \tag{20.14}$$

b) asymetrická váhová funkce

b1)

$$\hat{W}(k) = i(k) - i(k-1), \tag{20.15}$$

b2)

$$\hat{W}(k) = j(k) - j(k-1), \tag{20.16}$$

c)

$$\hat{W}(k) = \min\left[i(k) - i(k-1), j(k) - j(k-1)\right], \tag{20.17}$$

d)

$$\hat{W}(k) = \max\left[i(k) - i(k-1), j(k) - j(k-1)\right], \tag{20.18}$$

přičemž $i(0) = j(0) = 0$.

20.4 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

$$N(\hat{W}) = \sum_{k=1}^{K} \hat{W}(k). \tag{20.19}$$

Dosazením vztahů 20.14, 20.15, 20.16 pro váhové funkce typu a) a b) získáme normalizační faktory

$$N(\hat{W}_a) = \sum_{k=1}^{K} [i(k) - i(k-1) + j(k) - j(k-1)] = \tag{20.20}$$

$$= i(K) - i(0) + j(K) - j(0) = I + J, \tag{20.21}$$

$$N(\hat{W}_{b1}) = \sum_{k=1}^{K} [i(k) - i(k-1)] = i(K) - i(0) = I, \tag{20.22}$$

$$N(\hat{W}_{b2}) = \sum_{k=1}^{K} [j(k) - j(k-1)] = j(K) - j(0) = J. \tag{20.23}$$

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.

$$N(\hat{W}_c) = N(\hat{W}_d) = I \tag{20.24}$$

20.5 Rekurzivní algoritmus

Díky nezávislosti normalizačního faktoru na průběhu funkce DTW lze vztah 20.13 pro výpočet celkové minimální vzdálenosti mezi dvěma obrazy A a B zjednodušit do tvaru

$$D(A,B) = \frac{1}{N(\hat{W})} \{ \min_{\{i(k),j(k),K\}} \sum_{k=1}^{K} d\,[i(k), j(k)]\, \hat{W}(k) \} \tag{20.25}$$

Výslednou hodnotu vztahu 20.25 lze určit rekurzivně algoritmem dynamického programování, kdy zavedeme funkci g částečné akumulované vzdálenosti:

1. Inicializace

$$g\left[i(1), j(1)\right] = d\left[i(1), j(1)\right] \hat{W}(1) \tag{20.26}$$

2. Rekurze

$$g\left[i(k), j(k)\right] = \min_{\{i(k), j(k)\}} \{g\left[i(k-1), j(k-1)\right] + d\left[i(k), j(k)\right] \hat{W}(k)\} \tag{20.27}$$

3. Konečná normalizovaná vzdálenost

$$D(A, B) = \frac{1}{N(\hat{W})} g\left[i(K), j(K)\right] = \frac{1}{N(\hat{W})} g\left[I, J\right] \tag{20.28}$$

Rekurzivní vztahy pro různé typy lokálních omezení lze odvodit dosazením za $\hat{W}(k)$ [9,23].

21 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 [13], knihovny pro strojové učení scikit-learn [20] a knihovny pro neuronové sítě Lasagne [24].

21.1 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 [25])

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 [26]).

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.

21.2 Dynamic Time Warping

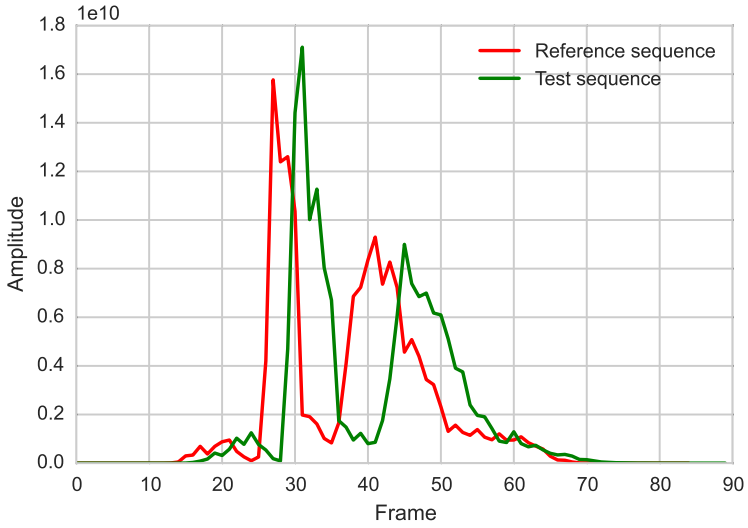
Pro výpočet DTW vzdálenosti byl vytvořen modul, který obsahuje základní DTW algoritmus bez globálního vymezení pohybu funkce a využívá symetrické omezení na lokální strmost (první zleva na obrázku 10). Jako vzdálenostní metrika pro všechny typy příznaků kromě *ste_sti_stzcr* byla zvolena Euklidova vzdálenost.

21.2.1 Optimalizace parametrů vzdálenostní metriky

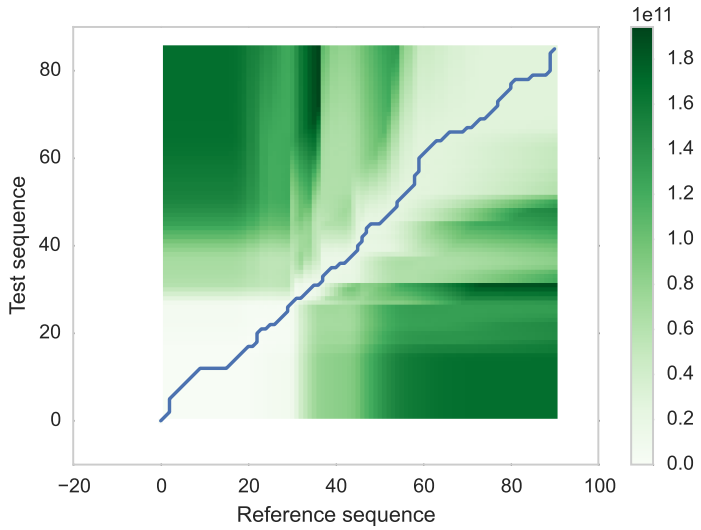
Pro kombinaci příznaků složenou z krátkodobé energie, intenzity a průchodů nulou byla využita vlastní vzdálenostní metrika

$$d(a_i, b_j) = \alpha|a_{ste,i} - b_{ste,j}| + \beta|a_{sti,i} - b_{sti,j}| + \gamma|a_{stzcr,i} - b_{stzcr,j}|, \tag{21.1}$$

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 12: Krátkodobé energie pro dvě různé nahrávky číslovky 7.



Obrázek 13: 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*.

21.3 Support Vector Machine

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

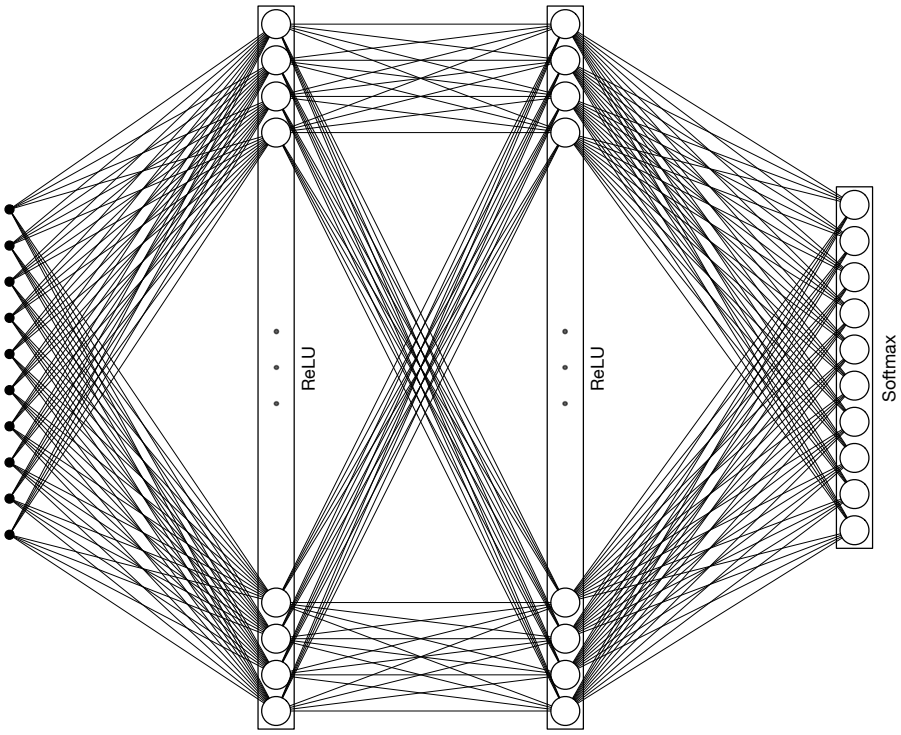
$$K(A, B) = 1 - DTW(A, B). \tag{21.2}$$

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

21.4 Neuronová síť

Pro každého řečníka byla natrénována třívrstvá neuronová síť (znázorněna na obrázku 14). 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.

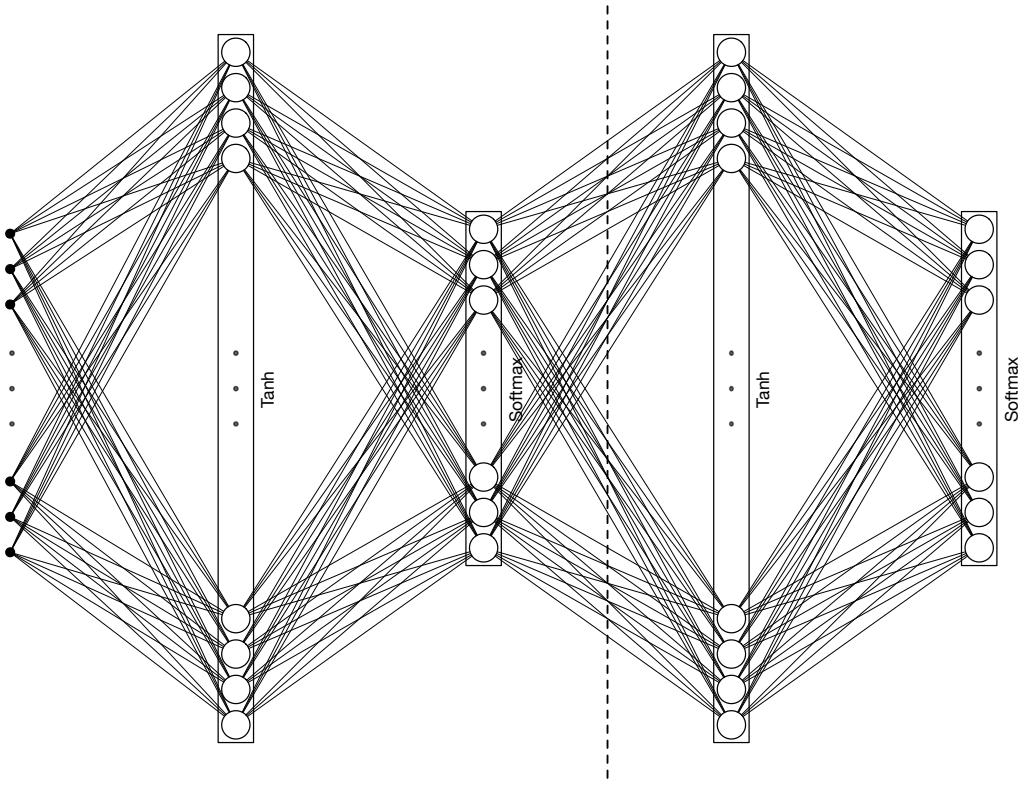


Obrázek 14: Třívrstvá neuronová síť.

21.5 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 15). 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 15: 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.

22 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.

22.1 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.

22.2 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.

22.3 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.

23 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., and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[3] Andrej 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] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. *CoRR*, abs/1212.0901, 2012.

[6] Ami Moyal, Vered Aharonson, Ella Tetariy, , and Michal Gishri. *Phonetic Search Methods for Large Speech Databases*. Springer-Verlag New York, 2013. ISBN: 978-1-4614-6488-4.

[7] G. Chen, C. Parada, and T. N. Sainath. Query-by-example keyword spotting using long short-term memory networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, 2015.

[8] T. J. Hazen, W. Shen, and C. White. Query-by-example spoken term detection using phonetic posteriorgram templates. In *Automatic Speech Recognition Understanding, 2009. ASRU 2009. IEEE Workshop on*, 2009.

[9] Josef Psutka. *Komunikace s počítačem mluvenou řečí*. Academia, 1995. ISBN: 978-8-020-00203-7.

[10] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2009. ISBN: 978-0-262-01243-0.

[11] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, 2000. ISBN: 978-0-471-05669-0.

[12] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012. ISBN: 978-1-600-49006-4.

[13] SciPy developers. Numpy and scipy documentation, 2016. [Online], Dostupné z: http://docs.scipy.org/doc/.

[14] Josef Psutka, Jindřich Matoušek, Luděk Muller, and Vlasta Radová. *Mluvíme s počítačem česky*. Academia, 2006. ISBN: 978-8-020-01309-5.

[15] M. A. Hossan, S. Memon, and M. A. Gregory. A novel approach for mfcc feature extraction. In *Signal Processing and Communication Systems (ICSPCS), 2010 4th International Conference on*, pages 1–5, 2010.

[16] Sebastian Raschka. About feature scaling and normalization, 2014. [Online], Dostupné z: http://sebastianraschka.com/Articles/2014_about_feature_scaling.html.

[17] Chris J.C. Burges. *A Tutorial on Support Vector Machines for Pattern Recognition*. Kluwer Academic Publishers, 1998.

[18] Jan Švec. *Diskriminativní model pro porozumeňí mluvené řeči*. PhD thesis, Západočeská univerzita v Plzni, 2013.

[19] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*. Springer, 2015. ISBN: 978-1-461-47137-0.

[20] scikit-learn developers. scikit-learn examples, 2016. [Online], Dostupné z: http://scikit-learn.org/stable/auto_examples/index.html.

[21] Jerome H. Friedman, Robert Tibshirani, and Trevor Hastie. *The Elements of Statistical Learning*. Springer, 2009. ISBN: 978-0-387-84857-0.

[22] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Education, 1997. ISBN: 978-0-070-42807-2.

[23] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1978.

[24] Lasagne contributors. Lasagne documentation, 2016. [Online], Dostupné z: https://lasagne.readthedocs.org/en/latest/.

[25] James Lyons. Python speech features, 2016. [Online], Dostupné z: https://github.com/jameslyons/python_speech_features.

[26] LibROSA development team. Librosa documentation, 2016. [Online], Dostupné z: https://bmcfee.github.io/librosa/.