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, začaly 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 neuronových sítí a optimalizačních algoritmů. 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 Neuronové sítě

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

Základním rozdílem mezi běžnou optimalizací a optimalizací v neuronových sítí je, že optimalizace v neuronových sítích a ve strojovém učení obecně probíhá nepřímo. To znamená, že ačkoliv optimalizujeme zvolenou metriku P, která v jistém smyslu kvantifikuje výkon algoritmu na dané úloze, minimalizujeme jinou cenovou funkci $J(\boldsymbol{\theta})$ za účelem minimalizace metriky P. V běžném optimalizačním problému bychom minimalizovali přímo cenovou funkci $J(\boldsymbol{\theta})$ za účelem její minimalizace [2–4].

2.1 Rozpoznávání řeči pomocí neuronových sítí

TODO

2.2 Cenová funkce

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

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{data}} \log p_{model}(\boldsymbol{y} \mid \boldsymbol{x}). \tag{2.1}$$

Předpis cenové funkce se liší model od modelu v závislosti na tvaru $\log p_{model}$ a kromě definice ceny také může obsahovat regularizační prvky. Výhodou využití principu maximální věrohodnosti je, že specifikací modelu $p(\boldsymbol{y} \mid \boldsymbol{x})$ zároveň definujeme cenovou funkci $\log p(\boldsymbol{y} \mid \boldsymbol{x})$ [2, 4].

2.3 Dopředné neuronové sítě

Dopředné neuronové sítě, občas také nazývané vícevrstvé perceptrony, jsou základním typem neuronových sítí. Jejich cílem je aproximovat určitou funkci f^*, např. klasifikátor $\boldsymbol{y} = f^*(\boldsymbol{x})$ zobrazuje vstup $\boldsymbol{y}$ na výstupní třídu $\boldsymbol{x}$. Dopředná neuronová síť tedy definuje zobrazení $\boldsymbol{y} = f^*(\boldsymbol{x}; \boldsymbol{\theta})$ a učí se optimální hodnoty parametrů $\boldsymbol{\theta}$ při kterých by měla být dosažena nejlepší aproximace funkce.

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

Cílem trénování sítě je, aby se naučila odpovídající zobrazení mezi $f(\boldsymbol{x})$ a $f^*(\boldsymbol{x})$. Učení probíhá na základě trénovacích dat, která poskytují zašuměná aproximovaná pozorování $f^*(\boldsymbol{x})$ vyhodnocena v různých bodech. Ke každému pozorování $\boldsymbol{x}$ je k dispozici také skutečná hodnota $\boldsymbol{y} \approx f^*(\boldsymbol{x})$. Výstupní vrstva sítě se tedy pro každou trénovací hodnotu $\boldsymbol{x}$ snaží vyprodukovat hodnotu blízkou $\boldsymbol{y}$. Chování skrytých vrstev není přímo dáno trénovacími daty a síť se je musí naučit využívat k získání požadovaného výsledku, tedy k aproximaci $f^*(\boldsymbol{x})$ [2, 4].

2.3.1 Architektura

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

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

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right), \tag{2.2}$$

druhá vrstva jako

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{x} + \boldsymbol{b}^{(2)}\right), \tag{2.3}$$

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

Pro takto definovanou řetězovou architekturu je pak hlavním problémem určení hloubky sítě a šířky každé vrstvy. Vhodnou architekturu pro danou úlohu je třeba nalézt pomocí experimentů založených na sledování chyby na validační datové sadě a apriorní znalosti o úloze a datech [2–5].

2.3.2 Výstupní jednotky

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

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

$$\hat{\boldsymbol{y}} = P(\boldsymbol{y} = 1 \mid \boldsymbol{x}), \tag{2.4}$$

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

$$\boldsymbol{z} = \log \tilde{P}(\boldsymbol{y} = 1 \mid \boldsymbol{x}). \tag{2.5}$$

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

Pro generalizaci tohoto postupu pro diskrétní proměnnou s n hodnotami je tedy potřeba získat vektor $\hat{\boldsymbol{y}}$, kde $\hat{\boldsymbol{y}}_i = P(\boldsymbol{y} = i \mid \boldsymbol{x})$. Pro každý prvek $\hat{\boldsymbol{y}}_i$ musí platit

$\hat{\boldsymbol{y}}_i \in [0, 1]$ a zároveň $\sum_i \hat{\boldsymbol{y}}_i = 1$, aby bylo možné tento vektor interpretovat za hustotu pravděpodobnosti. Nejprve je potřeba provést predikci vrstvou s lineární aktivační funkcí, která predikuje nenormalizované logaritmované pravděpodobnosti

$$\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}, \tag{2.6}$$

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

$$softmax(\boldsymbol{z}_i) = \frac{\exp(\boldsymbol{z}_i)}{\sum_j \exp(\boldsymbol{z}_j)} \tag{2.7}$$

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

$$softmax(\boldsymbol{z}(\boldsymbol{x}; \boldsymbol{\theta}))_i \approx \frac{\sum_{j=1}^{m} 1_{\boldsymbol{y}^{(j)}=i, \boldsymbol{x}^{(j)}=\boldsymbol{x}}}{\sum_{j=1}^{m} 1_{\boldsymbol{x}^{(j)}=\boldsymbol{x}}} \tag{2.8}$$

2.3.3 Skryté jednotky

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

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

- **sigmoid** a **tanh** - tyto jednotky využívají logistickou funkci (sigmoid) $g(\boldsymbol{z}) = \sigma(\boldsymbol{z})$,

resp. hyperbolický tangens $\tanh(\boldsymbol{z}) = 2\sigma(2\boldsymbol{z}) - 1$. Hlavním nedostatkem sigmoidální funkce je její citlivost a náchylnost k saturaci, kdy může dojít k tzv. explozi či vymizení gradientu a síť nebude schopná se učit.

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

2.3.4 Algoritmus zpětného šíření

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

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

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

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}. \tag{2.9}$$

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

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}_i} = \sum_j \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{y}_j}\frac{\partial \boldsymbol{y}_j}{\partial \boldsymbol{x}_i} \tag{2.10}$$

Zapíšeme-li rovnici (2.10) vektorově, získáme tvar

$$\nabla_{\boldsymbol{x}} \boldsymbol{z} = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \right)^{\top} \nabla_{\boldsymbol{y}} \boldsymbol{z}, \tag{2.11}$$

kde $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \in \mathbb{R}^{n \times m}$ je Jakobián $\boldsymbol{x}$. Toto pravidlo lze dále snadno rozšířit i na tensory, jenž jsou v neuronových sítích často využívány (podrobněji v [2, 6]).

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

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

Algoritmus 2: Dopředné šíření.

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

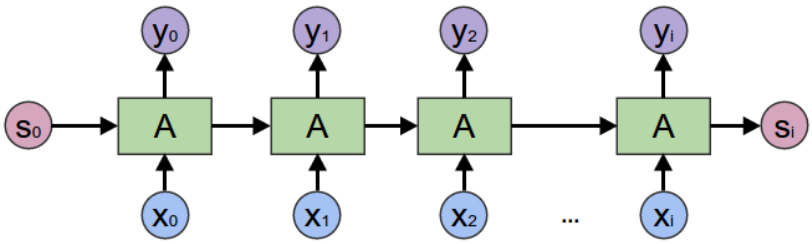
---

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

Algoritmus 3: Zpětné šíření.

2.4 Rekurentní neuronové sítě

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



Obrázek 1: Schéma rekurentní neuronové sítě. Převzato z [7].

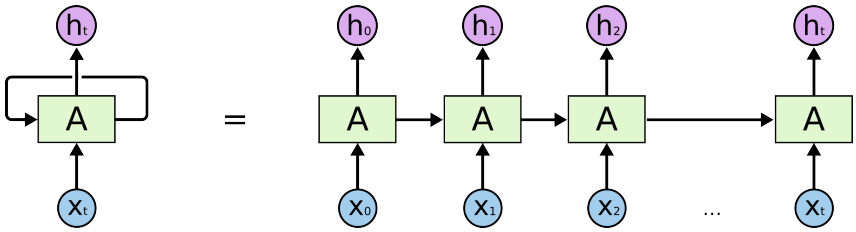
Sdílení parametrů je zajištěno ”rozvinutím”rekurentního výpočtu do grafu s opakující se strukturou. Prvek sekvence v každém časovém kroku je tedy zpracováván stejnou sítí. Pro jednoduchost předpokládejme, že RNN pracuje se sekvencí obsahující vektory $\boldsymbol{x}^{(t)}$, $t = 1, \ldots, \tau$ a model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

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta}), \tag{2.12}$$

kde $\boldsymbol{s}$ je stav systému. Tento rekurzivní vztah je možné pro konečný počet kroků τ rozvinout τ-krát. Například pro model daný vztahem (2.12) a $\tau = 3$ získáme

$$\boldsymbol{s}^{(3)} = f(\boldsymbol{s}^{(2)}; \boldsymbol{\theta}) = f(f(\boldsymbol{s}^{(1)}; \boldsymbol{\theta}); \boldsymbol{\theta}). \tag{2.13}$$

Tímto rozvinutím zajistíme, že vztah neobsahuje žádnou rekurenci a je možné ho reprezentovat jako acyklický orientovaný graf [2, 8].



Obrázek 2: Ilustrace rozvinutí rekurentní neuronové sítě. Převzato z [8].

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

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}), \tag{2.14}$$

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

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

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \tag{2.15}$$

$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)}), \tag{2.16}$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \tag{2.17}$$

$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}(\boldsymbol{o}^{(t)}), \tag{2.18}$$

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

2.4.1 Algoritmus zpětného šíření časem

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

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

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

$$L(\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\}) = -\sum_t \log_{p_{model}} \left(\boldsymbol{y}^{(t)} \mid \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\}\right), \tag{2.19}$$

$$\frac{\partial L}{\partial L^{(t)}} = 1. \tag{2.20}$$

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

$$(\nabla_{\boldsymbol{o}^{(t)}} L)_i = \frac{\partial L}{\partial \boldsymbol{o}_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial \boldsymbol{o}_i^{(t)}} = \hat{\boldsymbol{y}}_i^{(t)} - \mathbf{1}_{i,\boldsymbol{y}^{(t)}}. \tag{2.21}$$

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

$$\nabla_{\boldsymbol{h}^{(\tau)}} L = \boldsymbol{V}^\top \nabla_{\boldsymbol{o}^{(\tau)}} L. \tag{2.22}$$

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

$$\begin{aligned} \nabla_{\boldsymbol{h}^{(t)}} L &= \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{o}^{(t)}} L) \\ &= \boldsymbol{W}^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) \operatorname{diag}\left(1 - (\boldsymbol{h}^{(t+1)})^2\right) + \boldsymbol{V}^\top (\nabla_{\boldsymbol{o}}^{(t)} L), \end{aligned} \tag{2.23}$$

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

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

$$\nabla_{\boldsymbol{c}} L = \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}} \right)^\top \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L \tag{2.24}$$

$$\nabla_{\boldsymbol{b}} L = \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}} \right)^\top \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\left(1 - (\boldsymbol{h}^{(t)})^2\right) \nabla_{\boldsymbol{h}^{(t)}} L \tag{2.25}$$

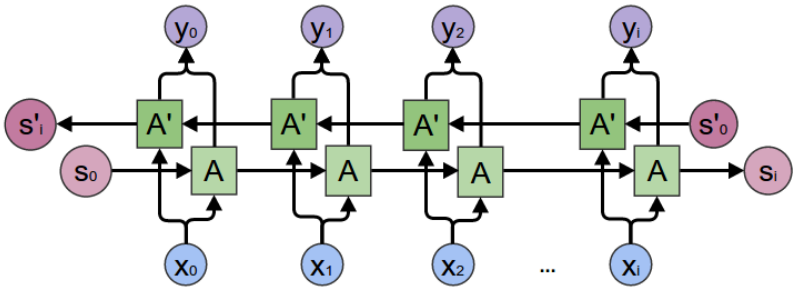
$$\nabla_{\boldsymbol{V}} L = \sum_t \sum_t \left(\frac{\partial L}{\partial \boldsymbol{o}_i^{(t)}} \right) \nabla_{\boldsymbol{V}} \boldsymbol{o}_i^{(t)} = \sum_t (\nabla_{\boldsymbol{o}^{(t)}} L) \boldsymbol{h}^{(t)\top} \tag{2.26}$$

$$\nabla_{\boldsymbol{W}} L = \sum_t \sum_t \left(\frac{\partial L}{\partial \boldsymbol{h}_i^{(t)}} \right) \nabla_{\boldsymbol{W}^{(t)}} \boldsymbol{h}_i^{(t)} = \sum_t \operatorname{diag}\left(1 - (\boldsymbol{h}^{(t)})^2\right) (\nabla_{\boldsymbol{h}}^{(t)} L) \boldsymbol{h}^{(t-1)\top} \tag{2.27}$$

$$\nabla_{\boldsymbol{U}} L = \sum_t \sum_t \left(\frac{\partial L}{\partial \boldsymbol{h}_i^{(t)}} \right) \nabla_{\boldsymbol{u}^{(t)}} \boldsymbol{h}_i^{(t)} = \sum_t \operatorname{diag}\left(1 - (\boldsymbol{h}^{(t)})^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L) \boldsymbol{x}^{(t)\top} \tag{2.28}$$

2.4.2 Obousměrné rekurentní neuronové sítě

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



Obrázek 3: Schéma obousměrné rekurentní neuronové sítě. Převzato z [7].

BRNN kombinuje dvě RNN, kde jedna síť se pohybuje v čase kupředu od začátku sekvence, zatímco druhá se pohybuje v čase zpět od konce sekvence. Tyto dvě sítě mezi sebou sdílí skryté stavy $\boldsymbol{h}^{(t)}$ (síť dopředná) a $\boldsymbol{g}^{(t)}$ (síť zpětná), což zajistí, že výstupní jednotky $\boldsymbol{o}^{(t)}$ jsou závislé jak na minulosti, tak na budoucnosti. Výstupy skrytých stavů $\boldsymbol{h}^{(t)}$ a $\boldsymbol{g}^{(t)}$ nejsou mezi sebou nijak propojeny. Pro výpočet gradientů lze opět využít algoritmus zpětného šíření časem dle postupu, který je shrnut v algoritmu č. 4 [2, 9].

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

Algoritmus 4: Postup trénování obousměrné rekurentní neuronové sítě.

2.4.3 LSTM

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

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

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

$$\boldsymbol{f}_i^{(t)} = \sigma\left(\boldsymbol{b}_i^f + \sum_j \boldsymbol{U}_{i,j}^f \boldsymbol{x}_j^{(t)} + \sum_j \boldsymbol{W}_{i,j}^f h_j^{(t-1)}\right), \tag{2.29}$$

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

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

$$\boldsymbol{s}_i^{(t)} = \boldsymbol{f}_i^{(t)} \boldsymbol{s}_i^{(t-1)} + \boldsymbol{g}_i^{(t)} \sigma \left(\boldsymbol{b}_i + \sum_j \boldsymbol{U}_{i,j} \boldsymbol{x}_j^{(t)} + \sum_j \boldsymbol{W}_{i,j} \boldsymbol{h}_j^{(t-1)} \right), \tag{2.30}$$

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

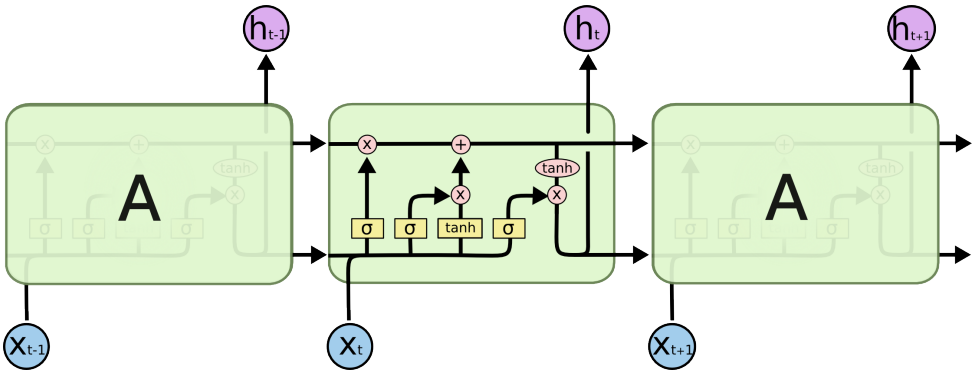
$$\boldsymbol{g}_i^{(t)} = \sigma \left(\boldsymbol{b}_i^g + \sum_j \boldsymbol{U}_{i,j}^g \boldsymbol{x}_j^{(t)} + \sum_j \boldsymbol{W}_{i,j}^g h_j^{(t-1)} \right), \tag{2.31}$$

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

$$\boldsymbol{h}_i^{(t)} = \tanh(\boldsymbol{s}_i^{(t)}) \boldsymbol{q}_i^{(t)}, \tag{2.32}$$

$$\boldsymbol{q}_i^{(t)} = \sigma \left(\boldsymbol{b}_i^o + \sum_j \boldsymbol{U}_{i,j}^o \boldsymbol{x}_j^{(t)} + \sum_j \boldsymbol{W}_{i,j}^o \boldsymbol{h}_j^{(t-1)} \right), \tag{2.33}$$

kde $\boldsymbol{b}^o$, $\boldsymbol{U}^o$ a $\boldsymbol{W}^o$ jsou popořadě prahový vektor, váhová matice vstupu a rekurentní váhová matice výstupní brány [2, 8, 10].



Obrázek 4: Schéma LSTM buňky. Převzato z [8].

2.4.4 GRU

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

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

$$\boldsymbol{h}_i^{(t)} = \boldsymbol{u}_i^{(t-1)}\boldsymbol{h}_i^{(t-1)} + (1 - \boldsymbol{u}_i^{(t-1)})\sigma\left(\boldsymbol{b}_i + \sum_j \boldsymbol{U}_{i,j}\boldsymbol{x}_j^{(t-1)} + \sum_j \boldsymbol{W}_{i,j}r_j^{(t-1)}\boldsymbol{h}_j^{(t-1)}\right), \quad (2.34)$$

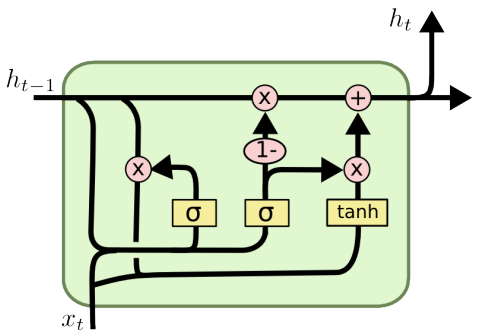
kde jednotka $\boldsymbol{u}$ modifikuje stav buňky

$$\boldsymbol{u}_i^{(t)} = \sigma\left(\boldsymbol{b}_i^u + \sum_j \boldsymbol{U}_{i,j}^u\boldsymbol{x}_j^{(t)} + \sum_j \boldsymbol{W}_{i,j}^u\boldsymbol{h}_j^{(t)}\right) \quad (2.35)$$

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

$$\boldsymbol{r}_i^{(t)} = \sigma\left(\boldsymbol{b}_i^r + \sum_j \boldsymbol{U}_{i,j}^r\boldsymbol{x}_j^{(t)} + \sum_j \boldsymbol{W}_{i,j}^r\boldsymbol{h}_j^{(t)}\right). \quad (2.36)$$

Takto definované jednotky s branami umožňují vhodně vybírat části stavového prostoru a uchovávat je ve stavu buňky. Zároveň sloučením výstupu a stavu buňky dochází k významnému snížení paměťové a výpočetní náročnosti [2, 8].



Obrázek 5: Schéma GRU buňky. Převzato z [8].

3 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

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \tag{3.1}$$

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, 3, 11].

3.1 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

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim p_{data}} L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{x}^i, \boldsymbol{y}^i, \boldsymbol{\theta}), \tag{3.2}$$

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

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^i, \boldsymbol{y}^i, \boldsymbol{\theta}), \tag{3.3}$$

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 $\boldsymbol{b}$

$$g = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(\boldsymbol{x}^i, \boldsymbol{y}^i, \boldsymbol{\theta}). \tag{3.4}$$

Změna parametrů je pak dána dle pravidla

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon g. \tag{3.5}$$

Pro pevně danou velikost modelu tedy výpočetní cena nezávisí na velikosti datové sady m [2, 11].

3.2 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, 11].

3.2.1 Moment

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

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

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right), \tag{3.6}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}. \tag{3.7}$$

Proměnná $\boldsymbol{v}$ akumuluje prvky gradientu $\nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right)$ s tím, že čím větší je parametr α vůči konstantě učení ϵ, tím více minulé gradienty ovlivňují aktuální směr kroku. Stejně jako u konstanty učení, i parametr α může být měnen v závislosti na čase. Většinou je jako počáteční hodnota zvolena 0.5 a je navyšována až na 0.99 [2, 3, 11].

3.2.2 Nesterovův moment

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

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} + \alpha \boldsymbol{v}), \boldsymbol{y}^{(i)}) \right), \tag{3.8}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}, \tag{3.9}$$

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, 3, 11, 12].

3.3 ADAM

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

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

ADAM používá ke změně parametrů plovoucí průměr (první centrální moment) gradientů s exponenciálním zapomínáním r a druhý necentrální moment v. Počáteční hodnota obou momentů je nastavena na nulu a zejména v prvních krocích algoritmu je potřeba momenty opravit korekčním faktorem. Parametry β_1 a β_2 udávají rychlost zapomínání a δ je malá konstanta [2, 3, 11, 13].

---

inicializace

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

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

$t \leftarrow t + 1$

výpočet gradientu

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

změna odhadu prvního centrálního momentu

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

změna odhadu druhého necentrálního momentu

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

korekce odhadu prvního centrálního momentu

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

korekce odhadu druhého necentrálního momentu

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

změna parametrů

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

end

Algoritmus 1: ADAM.

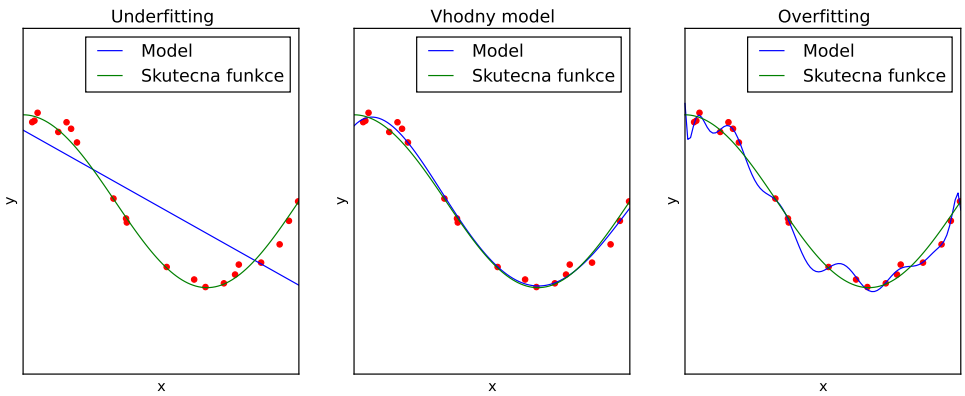
4 Kapacita modelu

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

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

Během optimalizace parametrů často dochází ke dvěma nežádoucím jevům:

- **underfitting** (podtrénování) - k underfittingu dochází, pokud model není schopný dostatečně minimalizovat chybu na trénovací sadě, tj. není schopný se naučit informační souvislosti obsažené v trénovacích pozorováních;
- **overfitting** (přetrénování) - k overfittingu dochází, pokud model není schopný minimalizovat rozdíl mezi chybou nad trénovací a testovací sadou, tj. model si ”zapamatuje”trénovací data místo toho, aby se naučil souvislosti obsažené v trénovacích pozorováních a na nových, dosud neviděných pozorováních selhává [2, 3, 5].



Obrázek 6: Příklad overfittingu a underfittingu.

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

4.1 Regularizace

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

4.1.1 Penalizace velikosti parametrů

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

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

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

4.1.2 Zanesení šumu

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

4.1.3 Předčasné ukončení

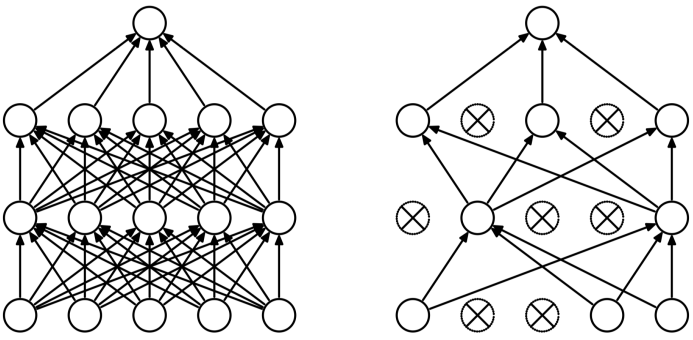
Při trénování velkých modelů s dostatečně velkou reprezentační kapacitou často dochází k jevu, kdy chyba na trénovací sadě pomalu během času klesá, zatímco chyba na validační sadě pomalu roste. Vrátíme-li se k nastavení parametrů v čase s nejnižší validační chybou, můžeme získat lepší model s nižší chybou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

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

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

4.1.4 Dropout

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



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

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

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

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

5 CTC

Tato práce se věnuje úloze klasifikace fonémů, jejímž cílem je pro danou zvukovou nahrávku získat odpovídající sekvenci fonému. Zvuková nahrávka je tedy rozdělena na malé segmenty a těmto segmentům jsou pak přiřazeny fonémy, které se v jednotlivých segmentech vyskytují (více v [1]). K tomuto označkování segmentů se většinou využívají předtrénované modely, které ovšem vyžadují jisté předpoklady, mohou být náchylné vůči šumu a výsledné označkování může být nepřesné, jelikož ne vždy lze přesně určit hranici mezi jednotlivými fonémy.

Běžně využívané cenové funkce jsou ovšem definovány pro jednotlivé prvky trénovací sekvence (segmenty) a umožňují pouze klasifikaci nezávislých fonému. To znamená, že takto definované cenové funkce je možné využít pouze pro nezávislou predikci fonémů v jednotlivých segmentech, kdy výstupní sekvence má stejnou délku jako vstupní sekvence. K získání výsledné sekvence fonémů je třeba výstupy sítě dále zpracovat pomocí dekodéru (např. pro vstupní sekvenci o dvanácti segmentech: $aahahhoohooj \rightarrow ahoj$).

V této kapitole si uvedeme metodu CTC (z anglického "connectionist temporal classification"), díky které lze využít rekurentní neuronové sítě přímo k predikci výsledné sekvence fonémů bez potřeby dalšího zpracování výstupu sítě [1, 15, 16].

5.1 Formální definice úlohy

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 L značek (fonémů). Prvky L^* budeme nazývat sekvencí značek (sekvence fonému) či značkováním. Každé pozorování v S je tvořeno párem $(\boldsymbol{x}, \boldsymbol{z})$, kde cílová sekvence $\boldsymbol{z} = (z_1, z_2, \ldots, z_U)$ je nejvýše tak dlouhá, jako vstupní sekvence $\boldsymbol{x} = (x_1, x_2, \ldots, x_T)$, tedy $U \leq T$. Vzhledem k tomu, že vstupní a cílové sekvence nejsou obecně stejně dlouhé, není možné je předem zarovnat, tj. přiřadit prvky cílové sekvence k vstupním prvkům sekvence.

Cílem je natrénovat klasifikátor $h : \mathcal{X} \mapsto \mathcal{Z}$ za využití S, který minimalizuje vhodně zvolenou metriku pro danou úlohu (např. minimalizace počtu transkripčních chyb) [15,16].

5.2 Reprezentace výstupu

Aby bylo možné využít RNN spolu s CTC, je třeba vhodně zvolit reprezentaci výstupu sítě. Využijeme-li aktivační funkci softmax pro výstupní jednotky, můžeme transformovat výstupy sítě do tvaru podmíněné hustoty pravděpodobnosti nad danou abecedou značek a využít síť jako klasifikátor, kdy vstupní sekvence klasifikujeme výběrem nejvíce pravděpodobného značkování. Výstupní vrstva zároveň musí obsahovat o jednu jednotku více, než je počet značek v L. Aktivace prvních $|L|$ jednotek je interpretována jako pravděpodobnost pozorování odpovídajících značek a aktivace $|L| + 1$ jednotky je interpretována jako pravděpodobnost pozorování prázdné značky. Prázdná značka umožňuje zobrazit více prvků vstupní sekvence na jednu značku či ignorovat prvky sekvence, které odpovídají tichu, a z výsledné sekvence je odstraněn. Celkovou pravděpodobnost libovolného značkování pak můžeme vyčíslit sečtením pravděpodobností jeho různých zarovnání.

Mějme tedy RNN s m vstupními jednotkami, n výstupními jednotkami a váhovým vektorem $\boldsymbol{w}$, která slouží jako zobrazení $\mathcal{N}_{\boldsymbol{w}} : (\mathbb{R}^m)^T \mapsto (\mathbb{R}^n)^T$. Dále mějme výstupní sekvenci sítě $\boldsymbol{y} = \mathcal{N}_{\boldsymbol{w}}(\boldsymbol{x})$, jejíž prvky $\boldsymbol{y}_k^{(t)}$ odpovídají aktivaci výstupní jednotky k v čase t a lze je interpretovat jako pravděpodobnost pozorování značky k v čase t. Hustota pravděpodobnosti značkování nad množinou L'^T sekvencí délky T nad abecedou $L' = L \cup \{\textit{prázdná značka}\}$, je pak

$$p(\boldsymbol{\pi} \mid \boldsymbol{x}) = \prod_{t=1}^{T} \boldsymbol{y}_{\boldsymbol{\pi}_t}^{(t)}, \forall \boldsymbol{\pi} \in L'^T, \tag{5.1}$$

kde $\boldsymbol{\pi}$ budeme nazývat cestami a odpovídají prvkům L'^T.

Dále definujme zobrazení $\mathcal{B} : L'^T \mapsto L^{\leq T}$, kde $L^{\leq T}$ je množina všech možných značkování, tj. množina výstupních sekvencí o délce menší či rovné T nad původní abecedou značek L. Toto zobrazení získáme odstraněním opakujících se značek a prázdných značek ze získaných cest (např. $\mathcal{B}(aa - ab-) = \mathcal{B}(a - a - bb) = aab$) a využijeme ho k získání podmíněné pravděpodobnosti daného značkování $\boldsymbol{l} \in L^{\leq T}$ [15, 16]

$$p(\boldsymbol{l} \mid \boldsymbol{x}) = \sum_{\boldsymbol{\pi} \in \mathcal{B}^{-1}(\boldsymbol{l})} p(\boldsymbol{\pi} \mid \boldsymbol{x}), \tag{5.2}$$

5.3 Dekódování

Výstupem klasifikátoru by mělo být nejvíce pravděpodobné značkování pro vstupní sekvenci

$$h(\boldsymbol{x}) = \underset{\boldsymbol{l} \in L^{\leq T}}{\operatorname{argmax}}\ p(\boldsymbol{l} \mid \boldsymbol{x}). \tag{5.3}$$

Proces hledání značkování nazýváme dekódováním a pro metodu CTC lze využít dvě aproximativní metody.

První metoda, dekódování nejlepší cesty, je založena na předpokladu, že cesta s nejvyšší pravděpodobností odpovídá nejvíce pravděpodobnému značkování

$$h(\boldsymbol{x}) \approx \mathcal{B}(\boldsymbol{\pi}^*), \tag{5.4}$$

$$\boldsymbol{\pi}^* = \underset{\boldsymbol{\pi} \in N^t}{\operatorname{argmax}}\ \ p(\boldsymbol{\pi} \mid \boldsymbol{x}). \tag{5.5}$$

Dekódování nejlepší cesty lze snadno najít zřetězením výstupu s nejvyšší aktivační hodnotou v každém časovém kroku. Není ovšem zaručeno, že tato metoda nalezne nejlepší možné značkování.

Druhá metoda využívá slučování cest, které procházejí stejnou sekvencí značek, a zaručuje nalezení nejlepšího značkování. Je však výpočetně náročná a aby tato metoda byla upočitatelná, je třeba využít další heuristiky (např. BEAM prořezávání, které umožní kompromis mezi rychlostí výpočtu a přesností).

Obě tyto metody mohou být zároveň obohaceny o jazykový model

$$h(\boldsymbol{x}) = \underset{\boldsymbol{l} \in L^{\leq T}}{\operatorname{argmax}}\ p(\boldsymbol{l} \mid \boldsymbol{x}) \cdot p(\boldsymbol{l})^{\gamma} \cdot g(\boldsymbol{l})^{\zeta}, \tag{5.6}$$

kde $p(\boldsymbol{l})$ je pravděpodobnost značkování daná jazykovým modelem, $g(\boldsymbol{l})$ je penalizace za vložení slova a parametry γ a ζ jsou předem dané váhy [15, 16].

5.4 Dopředný a zpětný algoritmus

Pro získání výsledného značkování je třeba spočítat podmíněnou pravděpodobnost $p(\boldsymbol{l} \mid \boldsymbol{x})$ jednotlivých značkování. To může být značně problematické, jelikož je potřeba spočítat sumu přes všechny cesty odpovídající danému značkování. Využijeme k tomu algoritmus dynamického programování, který vychází z předpokladu, že suma všech cest odpovídajících značkování může být rozložena na iterativní sumu přes cesty, které v daném

časovém kroku dosáhly stejné značky. Tyto iterace pak mohou být snadno spočteny pomocí rekurzivních dopředných a zpětných proměnných.

Mějme sekvenci $\boldsymbol{q}$ délky r, kde $\boldsymbol{q}_{1:p}$ a $\boldsymbol{q}_{r-p:r}$ značí prvních a posledních p značek. Potom pro značkování $\boldsymbol{l}$ zavedeme dopřednou proměnnou $\alpha_t(s)$, která odpovídá celkové pravděpodobnosti $\boldsymbol{l}_{1:s}$ v čase t

$$\alpha_t(s) = \sum_{\substack{\boldsymbol{\pi} \in N^T: \\ \mathcal{B}(\boldsymbol{\pi}_{1:t}) = \boldsymbol{l}_{1:s}}} \prod_{t'=1}^{t} \boldsymbol{y}_{\pi_{t'}}^{(t)'}, \tag{5.7}$$

kde $\alpha_t(s)$ může být rekurzivně spočtena z $\alpha_{t-1}(s)$ a $\alpha_{t-1}(s-1)$. Aby bylo možné využít prázdné značky ve výstupní cestě, je třeba upravit značkování $\boldsymbol{l}$ na $\boldsymbol{l}'$ vložením prázdné značky na začátek a konec sekvence a mezi každé dvě značky. Délka $\boldsymbol{l}'$ je tedy $2|\boldsymbol{l}| + 1$. Při výpočtu pravděpodobnosti značkování $\boldsymbol{l}'$ umožníme přechody mezi prázdnými a neprázdnými značkami a také mezi každým párem unikátních neprázdných značek. Zároveň musí všechny sekvence začínat buď prázdnou značkou b nebo prvním značkou v l. Těmito omezeními získáme inicializační hodnoty algoritmus

$$\alpha_1(1) = \boldsymbol{y}_b^{(1)}, \tag{5.8}$$

$$\alpha_1(2) = \boldsymbol{y}_{l_1}^{(1)}, \tag{5.9}$$

$$\alpha_1(s) = 0, \forall s > 2 \tag{5.10}$$

a rekurzi

$$\alpha_t(s) = \begin{cases} \bar{\alpha}_t(s) \boldsymbol{y}_{l'_s}^{(t)} & \text{pokud } \boldsymbol{l}'_s = b \text{ nebo } \boldsymbol{l}'_{s-2} = \boldsymbol{l}'_s \\ (\bar{\alpha}_t(s) + \alpha_{t-1}(s-2)) \boldsymbol{y}_{l'_s}^{(t)} & \text{jinak} \end{cases}, \tag{5.11}$$

kde

$$\bar{\alpha}_t(s) = \alpha_{t-1}(s) + \alpha_{t-1}(s-1). \tag{5.12}$$

Pravděpodobnost $\boldsymbol{l}$ je pak dána sumou výsledných pravděpodobností $\boldsymbol{l}'$ s prázdnou značkou a bez prázdné značky v čase T

$$p(\boldsymbol{l} \mid \boldsymbol{x}) = \alpha_T(|\boldsymbol{l}'|) + \alpha_T(|\boldsymbol{l}'| - 1). \tag{5.13}$$

Obdobně zadefinujeme zpětnou proměnnou $\beta_t(s)$ jako celkovou pravděpodobnost $\boldsymbol{l}_{s:|\boldsymbol{l}|}$ v čase t

$$\beta_t(s) = \sum_{\substack{\boldsymbol{\pi}\in N^T: \\ \mathcal{B}(\boldsymbol{\pi}_{t:T})=\boldsymbol{l}_{s:|\boldsymbol{l}|}}} \prod_{t'=t}^{T} \boldsymbol{y}_{\boldsymbol{\pi}_{t'}}^{(t)'}, \tag{5.14}$$

$$\beta_T(|\boldsymbol{l}'|) = \boldsymbol{y}_b^{(T)}, \tag{5.15}$$

$$\beta_T(|\boldsymbol{l}'|-1) = \boldsymbol{y}_{l_l}^{(T)}, \tag{5.16}$$

$$\beta_T(s) = 0, \forall s < |\boldsymbol{l}'|-1, \tag{5.17}$$

$$\beta_t(s) = \begin{cases} \bar{\beta}_t(s)\boldsymbol{y}_{\boldsymbol{l}'_s}^{(t)} & \text{pokud } \boldsymbol{l}'_s = b \text{ nebo } \boldsymbol{l}'_{s+2} = \boldsymbol{l}'_s \\ (\bar{\beta}_t(s) + \beta_{t+1}(s+2))\boldsymbol{y}_{\boldsymbol{l}'_s}^{(t)} & \text{jinak} \end{cases}, \tag{5.18}$$

$$\bar{\beta}_t(s) = \beta_{t+1}(s) + \beta_{t+1}(s+1). \tag{5.19}$$

Aby byla při výpočtech zajištěna numerická stabilita, je třeba dopředné a zpětné proměnné přeškálovat

$$\hat{\alpha}_t(s) = \frac{\alpha_t(s)}{\sum_s \alpha_t(s)}, \tag{5.20}$$

$$\hat{\beta}_t(s) = \frac{\beta_t(s)}{\sum_s \beta_t(s)}. \tag{5.21}$$

Dosazením přeškálovaných proměnných do vzorců a jejich úpravou (podrobněji v [16]) pak můžeme síť natrénovat pomocí metod založených na maximální věrohodnosti. Zpětné šíření gradientu skrz výstupní vrstvu s aktivační funkcí softmax je definováno následovně

$$\frac{\partial O^{ML}(\{\boldsymbol{x},\boldsymbol{z}\},\mathcal{N}_{\boldsymbol{w}})}{\partial \boldsymbol{u}_k^{(t)}} = \boldsymbol{y}_k^{(t)} - \frac{1}{\boldsymbol{y}_k^{(t)} Z_{(t)}} \sum_{s\in lab(\boldsymbol{z},k)} \hat{\alpha}_t(s)\hat{\beta}_t(s), \tag{5.22}$$

kde $\boldsymbol{u}_k^{(t)}$ jsou nenormalizované pravděpodobnosti na výstupu sítě, $lab(\boldsymbol{z},k) = lab(\boldsymbol{l},k) = \{s : \boldsymbol{l}'_s = k\}$ je množina pozic, na kterých se ve značkování $\boldsymbol{l}$ vyskytuje značka k, a kde [15, 16]

$$Z_t = \sum_{s=1}^{|\boldsymbol{l}'|} \frac{\hat{\alpha}_t(s)\hat{\beta}_t(s)}{\boldsymbol{y}_{\boldsymbol{l}'_s}^{(t)}}. \tag{5.23}$$

6 Klasifikace fonémů

V této práci bylo porovnáno šest architektur neuronových sítí pro úlohu klasifikace fonémů. Experimenty byly provedeny nad dvěma datovými sady pro čtyři různé parametrizace příznaků. K implementaci byl využit programovací jazyk Python 3.4 spolu s knihovnami pro neuronové sítě Tensorflow a Keras (kód dostupný na [17]). Sítě byly trénovány na grafické kartě Nvidia Tesla K20m s pamětí 5GB a knihovnami CUDA 8.0 a CuDNN 6.0.

6.1 Předzpracování dat

Architektury byly testovány na dvou datových sadách, ŠkodaAuto a SpeechDat-E, které obsahují české nahrávky se vzorkovací frekvencí 8 kHz. Obě tyto sady byly dále rozděleny na sadu trénovací (75%), validační (12.5%) a testovací (12.5%) tak, aby každý řečník byl v právě jedné sadě. Základní informace o datových sadách jsou uvedeny v tabulce 1.

	ŠkodaAuto	SpeechDat-E
počet řečníků	47	924
počet nahrávek	14523	39560
počet řečníků v trénovací sadě	35	693
počet řečníků ve validační sadě	6	115
počet řečníků v testovací sadě	6	116

Tabulka 1: Popis datových sad a jejich rozdělení.

Pro obě datové sady pak byly spočteny čtyři parametrizace příznaků [1], na které se dále budeme odkazovat tučně uvedenými zkratkami:

- **LFE** - logaritmované energie banky filtrů, pro výpočet využito 40 filtrů,
- **LFE ΔΔ** - logaritmované energie banky filtrů, pro výpočet využito 40 filtrů, vypočteny delta a delta-delta koeficienty,
- **MFCC** - mel-frekvenční kepstrální koeficienty, pro výpočet využito 40 filtrů, ponecháno 13 koeficientů,

- **MFCC ΔΔ** - mel-frekvenční kepstrální koeficienty, pro výpočet využito 40 filtrů, ponecháno 13 koeficientů, vypočteny delta a delta-delta koeficienty.

Všechny parametrizace byly vygenerovány s využitím Hammingova okénka o délce 25ms s posunem 10ms (jednotlivé segmenty se překrývají), a 512 bodové FFT.

Takto spočtené příznaky pak byly po sloupcích normalizovány tak, aby měly nulovou střední hodnotu a jednotkový rozptyl. K tomu byla využita Z-score normalizace, která přetransformuje vstupní data $\boldsymbol{x}$ na data $\boldsymbol{z}$ dle vzorce

$$\boldsymbol{z} = \frac{\boldsymbol{x} - \mu_{\boldsymbol{x}}}{\sigma_{\boldsymbol{x}}}, \tag{6.1}$$

kde $\mu_{\boldsymbol{x}}$ je střední hodnota $\boldsymbol{x}$ a $\sigma_{\boldsymbol{x}}$ je rozptyl $\boldsymbol{x}$.

6.2 Architektury a trénování sítí

Celkem bylo navrhnuto šest architektur sítí, na které se dále budeme odkazovat tučně uvedenými názvy:

- **dopředná**

typ vrstvy	počet neuronů	aktivační funkce
plně propojená	1024	ReLU
dropout (0.5)	-	-
plně propojená	512	ReLU
dropout (0.5)	-	-
plně propojená	256	ReLU
dropout (0.5)	-	-
plně propojená	40	softmax

- **LSTM**

typ vrstvy	počet neuronů/buněk	aktivační funkce
LSTM	$\{100, 150, 200, 250\}$	tanh
LSTM	$\{100, 150, 200, 250\}$	tanh
plně propojená	40	softmax

- **GRU**

typ vrstvy	počet neuronů/buněk	aktivační funkce
GRU	$\{100, 150, 200, 250\}$	tanh
GRU	$\{100, 150, 200, 250\}$	tanh
plně propojená	40	softmax

- **obousměrná CTC LSTM**

typ vrstvy	počet neuronů/buněk	aktivační funkce
obousměrná LSTM	$\{100, 150, 200, 250\}$	tanh
obousměrná LSTM	$\{100, 150, 200, 250\}$	tanh
plně propojená	41	softmax

- **dávková CTC LSTM**

typ vrstvy	počet neuronů/buněk	aktivační funkce
LSTM	$\{100, 150, 200, 250\}$	tanh
LSTM	$\{100, 150, 200, 250\}$	tanh
plně propojená	41	softmax

- **obousměrná dávková CTC LSTM**

typ vrstvy	počet neuronů/buněk	aktivační funkce
obousměrná LSTM	$\{100, 150, 200, 250\}$	tanh
obousměrná LSTM	$\{100, 150, 200, 250\}$	tanh
plně propojená	41	softmax

Dopředná síť byla trénována pomocí stochastického gradientního sestupu s konstantou učení $\epsilon = 0.01$ a Nesterovovým momentem $\alpha = 0.9$. Pokud během 5 trénovacích epoch nedošlo ke snížení ztráty na validační sadě, konstanta učení byla snížena na polovinu. Všechny rekurentní neuronové sítě pak využívaly optimalizační algoritmus ADAM s doporučenými hodnotami [13]. Všechny sítě byly trénovány po dávkách o velikosti 64 pozorování.

Obousměrná CTC LSTM využívá k predikci celou trénovací sekvenci. Celá sekvence ovšem v případě predikce v reálném čase není dostupná, a proto byly také testovány dávkové CTC sítě, které stejně jako dopředné či LSTM a GRU sítě využívají k predikci pouze kratší sekvence o předem definované délce .

Dopředná síť, síť LSTM a síť GRU využívají k dekódování výsledné sekvence Viterbiho algoritmus se zerogramovým jazykovým modelem, CTC sítě využívají dekódování nejlepší cesty.

Jako regularizační metody byly využity zanesení šumu do trénovacích dat a předčasné ukončení o toleranci 10 epoch. Gaussovský šum o směrodatné odchylce 0.6 [18] byl nově vygenerován pro každé pozorování před vstupem do sítě v každé epoše. Metoda dropout mohla být využita pouze pro dopřednou síť, jelikož pro rekurentní sítě byly využity optimalizované vrstvy pro knihovnu CuDNN, které tuto metodu zatím nepodporují. Nicméně se ukázalo, že regularizace metodou zanesení šumů je pro tuto úlohu dostačující a dropout, jehož rekurentní implementace bývá výpočetně velice pomalá, není třeba.

7 Experimenty a vyhodnocení

Nejprve byly provedeny experimenty nad datovou sadou ŠkodaAuto a její parametrizací LFE. Nejlepší nastavení jednotlivých architektur pak bylo testováno nad zbylými parametrizacemi jak na datové sadě ŠkodaAuto, tak na datové sadě SpeechDat-E. Byly uvažovány tři metriky, na které se dále budeme odkazovat tučně uvedenými názvy:

- **přesnost modelu** [%] - přesnost klasifikace modelu vyhodnocována po jednotlivých segmentech nahrávek

$$\text{přesnost modelu } = \frac{C_s}{N_s} \cdot 100, \tag{7.1}$$

  kde C_s je počet správně klasifikovaných pozorování (segmentů) a N_s je počet všech pozorování,

- **fonémů správně** [%] - procento správně klasifikovaných fonémů do dekódování

$$\text{fonémů správně } = \frac{C_f}{N_f} \cdot 100, \tag{7.2}$$

  kde C_f je počet správně klasifikovaných fonémů a N_f je počet všech fonémů,

- **přesnost dekódování** [%] - přesnost modelu po dekódování

$$\text{přesnost dekódování } = \frac{C_f - I}{N_f} \cdot 100, \tag{7.3}$$

  kde C_f je počet správně klasifikovaných fonémů, I je počet navíc vložených znaků (fonémů) a N_f je počet všech fonémů.

Pro sítě využívající metodu CTC nebyla sledována přesnost modelu, ale pouze přesnost dekódování. Během trénování modelů pak docházelo ke třem jevům

1. předčasné ukončení z důvodu stagnace učícího se procesu, tj. chyba na validační sadě se po dobu 10 epoch téměř nezměnila (*),
2. předčasné ukončení z důvodu overfittingu, tj. chyba na validační sadě začala růst a trénování bylo ukončeno (**),
3. síť se nepodařilo natrénovat (***).

Zároveň je třeba rozlišit tři důvody, proč se síť nepodařilo natrénovat:

1. síť se nepodařilo natrénovat kvůli vysoké výpočetní a časové náročnosti (např. pro datovou sadu SpeechDat-E s parametrizací LFE $\Delta\Delta$ trvala jedna epocha trénování dopředné sítě přes 45 hodin),

2. síť se nepodařilo natrénovat kvůli jevům jako je exploze či vymizení gradientu (např. sítě GRU pro datovou sadu ŠkodaAuto s parametrizací LFE),

3. síť se nepodařilo natrénovat kvůli neznámé chybě v knihovně Keras, kdy během jedné epochy dojde k výraznému poklesu přesnosti - pravděpodobně numerická nestabilita (např. sítě LSTM a GRU pro datovou sadu SpeechDat-E s parametrizacemi LFE a LFE $\Delta\Delta$).

7.1 Experimenty nad datovou sadou ŠkodaAuto

levý kontext	pravý kontext	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
0	0	60.54	64.79	53.17	150	710604
10	5	81.27	87.26	79.99	150	1327404
20	5	82.07	87.46	81.19	150	1738604
20	10	83.34	87.13	82.09	150	1994204

Tabulka 2: Výsledky experimentů pro dopřednou síť pro parametrizaci LFE nad datovou sadou ŠkodaAuto.

počet buněk	délka sekvence	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
100	10	70.37*	79.35	67.73	31	141640
150	10	72.12*	81.83	69.54	30	302440
200	10	72.80*	82.89	70.39	25	523240
250	10	73.49*	83.35	69.79	31	804040
100	20	76.03*	84.37	76.45	58	141640
150	20	77.73**	85.11	76.31	48	302440
200	20	78.51**	86.42	77.66	27	523240
250	20	78.63**	86.72	77.76	19	804040
100	30	79.59**	78.45	73.12	39	141640
150	30	81.57**	79.95	74.76	24	302440
200	30	82.02**	80.93	75.23	25	523240
250	30	82.70**	80.93	75.24	26	804040

Tabulka 3: Výsledky experimentů pro LSTM síť pro parametrizaci LFE nad datovou sadou ŠkodaAuto.

počet buněk	délka sekvence	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
100	10	69.22	77.99	66.81	41	107240
150	10	70.32*	79.91	68.30	34	228340
200	10	70.86*	80.75	68.70	39	394440
250	10	71.14*	81.05	68.46	56	605540
100	20	73.63*	80.40	72.84	30	107240
150	20	56.13***	62.28	55.41	25	228340
200	20	50.45***[2]	55.69	49.19	16	394440
250	20	52.18***[2]	57.89	51.03	12	605540
100	30	53.80***[2]	53.23	47.22	43	107240
150	30	56.00***[2]	56.92	49.57	18	228340
200	30	48.02***[2]	45.62	40.04	15	394440
250	30	49.97***[2]	48.87	43.43	12	605540

Tabulka 4: Výsledky experimentů pro GRU síť pro parametrizaci LFE nad datovou sadou ŠkodaAuto.

počet buněk	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
100	94.03	92.97**	124	363441
150	94.07	92.65**	68	785141
200	94.32	93.36**	60	1366841
250	95.18	94.43**	67	2108541

Tabulka 5: Výsledky experimentů pro obousměrnou CTC LSTM síť pro parametrizaci LFE nad datovou sadou ŠkodaAuto.

počet buněk	délka sekvence	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
100	20 (přesah 10)	81.31	63.04*	73	141741
100	20	78.61	61.55*	63	141741
200	10 (přesah 5)	85.8	51.13*	72	523441
200	10	83.76	45.61*	54	523441

Tabulka 6: Výsledky experimentů pro pro dávkovou CTC LSTM síť pro parametrizaci LFE nad datovou sadou ŠkodaAuto.

počet buněk	délka sekvence	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
100	20 (přesah 10)	87.55	72.31*	73	363441
100	20	86.42	67.93**	65	363441
200	10 (přesah 5)	87.34	53.22*	67	1366841
200	10	85.83	49.39**	42	1366841

Tabulka 7: Výsledky experimentů pro pro obousměrnou dávkovou CTC LSTM síť pro parametrizaci LFE nad datovou sadou ŠkodaAuto.

Nejlepší nastavení jednotlivých architektur uvedených výše, která byla využita pro další experimenty, jsou shrnuty v tabulce 8.

architektura	levý kontext	pravý kontext
dopředná	10	5
architektura	počet buněk	délka sekvence
LSTM	100	20
GRU	100	20
obousměrná CTC LSTM	100	-
dávková CTC LSTM	100	20 (přesah 10)
obousměrná dávková CTC LSTM	100	20 (přesah 10)

Tabulka 8: Nejlepší nastavení jednotlivých architektur získané experimenty.

architektura	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
dopředná	81.27	87.26	79.99	150	1327404
architektura	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
LSTM	76.03*	84.37	76.45	58	141640
GRU	73.63*	80.40	72.84	30	107240
obousměrná CTC LSTM	-**	94.03	92.97	124	363441
dávková CTC LSTM	-*	81.31	63.04	73	141741
obousměrná dávková CTC LSTM	-*	87.55	72.31	73	363441

Tabulka 9: Výsledky experimentů pro parametrizaci LFE nad datovou sadou ŠkodaAuto.

architektura	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
dopředná	80.15	87.10	80.15	150	2643244
architektura	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
LSTM	77.13**	85.39	78.17	59	173640
GRU	72.53*	81.23	73.49	22	131240
obousměrná CTC LSTM	-**	94.22	93.34	120	427441
dávková CTC LSTM	-*	86.05	68.58	98	173741
obousměrná dávková CTC LSTM	-**	89.97	72.74	83	427441

Tabulka 10: Výsledky experimentů pro parametrizaci LFE $\Delta\Delta$ nad datovou sadou ŠkodaAuto.

architektura	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
dopředná	81.16	87.06	79.63	150	883308
architektura	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
LSTM	71.53**	81.01	67.10	32	501640
GRU	73.70*	80.56	73.12	35	99140
obousměrná CTC LSTM	-**	94.20	93.03	116	341841
dávková CTC LSTM	-*	80.34	65.28	69	130941
obousměrná dávková CTC LSTM	-*	87.54	72.19	68	341841

Tabulka 11: Výsledky experimentů pro parametrizaci MFCC nad datovou sadou ŠkodaAuto.

architektura	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
dopředná	82.35	87.76	81.06	150	1310956
architektura	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
LSTM	78.82**	86.58	79.56	44	141240
GRU	77.03**	84.61	77.65	71	106940
obousměrná CTC LSTM	-**	94.66	93.70	12	5 362641
dávková CTC LSTM	-*	87.02	70.80	73	141341
obousměrná dávková CTC LSTM	-**	90.93	75.19	45	362641

Tabulka 12: Výsledky experimentů pro parametrizaci MFCC $\Delta\Delta$ nad datovou sadou ŠkodaAuto.

7.2 Experimenty nad datovou sadou SpeechDat-E

architektura	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
dopředná	72.5	71.68	68.65	150	1329717
architektura	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
LSTM	-***[3]	-	-	-	142549
GRU	-***[3]	-	-	-	108149
obousměrná CTC LSTM	-**	83.10	81.22	104	365250
dávková CTC LSTM	-*	69.40	58.32	50	142650
obousměrná dávková CTC LSTM	-*	73.12	62.42	40	365250

Tabulka 13: Výsledky experimentů pro parametrizaci LFE nad datovou sadou SpeechDat-E.

architektura	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
dopředná	-***[1]	-	-	-	2645557
architektura	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
LSTM	-***[1]	-	-	-	174549
GRU	-***[1]	-	-	-	132149
obousměrná CTC LSTM	-**	83.79	82.11	112	429250
dávková CTC LSTM	-*	73.28	60.87	61	174650
obousměrná dávková CTC LSTM	-*	77.85	66.81	28	429250

Tabulka 14: Výsledky experimentů pro parametrizaci LFE $\Delta\Delta$ nad datovou sadou SpeechDat-E.

architektura	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
dopředná	72.45	71.99	68.92	150	885621
architektura	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
LSTM	65.41*	66.64	62.87	41	131749
GRU	63.37*	63.31	59.86	23	100049
obousměrná CTC LSTM	-**	81.73	80.42	96	343650
dávková CTC LSTM	-*	70.70	60.05	86	131850
obousměrná dávková CTC LSTM	-*	73.66	63.42	25	343650

Tabulka 15: Výsledky experimentů pro parametrizaci MFCC nad datovou sadou SpeechDat-E.

architektura	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
dopředná	73.13	72.64	69.88	150	1313269
architektura	přesnost modelu [%]	fonémů správně [%]	přesnost dekódování [%]	počet epoch	počet parametrů
LSTM	69.94*	72.18	68.80	73	142149
GRU	67.34	70.23	65.53	42	107489
obousměrná CTC LSTM	-*	84.82	83.23	71	364450
dávková CTC LSTM	-*	75.07	63.81	91	142250
obousměrná dávková CTC LSTM	-*	78.21	67.87	48	364450

Tabulka 16: Výsledky experimentů pro parametrizaci MFCC $\Delta\Delta$ nad datovou sadou SpeechDat-E.

7.3 Porovnání výsledku nad ŠkodaAuto a SpeechDat-E

	přesnost modelu [%]				přesnost dekódování [%]			
architektura	LFE	LFE ΔΔ	MFCC	MFCC ΔΔ	LFE	LFE ΔΔ	MFCC	MFCC ΔΔ
dopředná	81.27	80.15	81.16	82.35	79.99	80.15	79.63	81.06
LSTM	76.03	77.13	71.53	78.82	76.45	78.17	67.10	79.56
GRU	73.63	72.53	73.70	77.03	72.84	73.49	73.12	77.65
obousměrná CTC LSTM	-	-	-	-	92.97	93.34	93.03	93.70
dávková CTC LSTM	-	-	-	-	63.04	68.58	65.28	70.80
obousměrná dávková CTC LSTM	-	-	-	-	72.31	72.74	72.19	75.19

Tabulka 17: Výsledky experimentů pro datovou sadou ŠkodaAuto.

	přesnost modelu [%]				přesnost dekódování [%]			
architektura	LFE	LFE ΔΔ	MFCC	MFCC ΔΔ	LFE	LFE ΔΔ	MFCC	MFCC ΔΔ
dopředná	72.50	-	72.45	73.13	68.65	-	68.92	69.88
LSTM	-	-	65.41	69.94	-	-	62.87	68.80
GRU	-	-	63.37	67.34	-	-	59.86	65.53
obousměrná CTC LSTM	-	-	-	-	81.22	82.11	80.42	83.23
dávková CTC LSTM	-	-	-	-	58.32	60.87	60.05	63.81
obousměrná dávková CTC LSTM	-	-	-	-	62.42	66.81	63.42	67.87

Tabulka 18: Výsledky experimentů pro datovou sadou SpeechDat-E.

8 Závěr

	ŠkodaAuto				SpeechDat-E			
architektura	LFE	LFE ΔΔ	MFCC	MFCC ΔΔ	LFE	LFE ΔΔ	MFCC	MFCC ΔΔ
dopředná	79.99	80.15	79.63	81.06	68.65	-	68.92	69.88
LSTM	76.45	78.17	67.10	79.56	-	-	62.87	68.8
GRU	72.84	73.49	73.12	77.65	-	-	59.86	65.53
obousměrná CTC LSTM	92.97	93.34	93.03	93.70	81.22	82.11	80.42	83.23
dávková CTC LSTM	63.04	68.58	65.28	70.80	58.32	60.87	60.05	63.81
obousměrná dávková CTC LSTM	72.31	72.74	72.19	75.19	62.42	66.81	63.42	67.87

Tabulka 19: Porovnání přesnosti dekódování [%] mezi datovou sadou ŠkodaAuto a SpeechDat-E.

..

..

..

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.

Porovnáním průměrných přesností klasifikace (tabulka **??**) 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 genero-

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

Seznam obrázků

Seznam tabulek

Reference

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[15] A. Hannun. Sequence modeling with ctc. *Distill*, 2017. https://distill.pub/2017/ctc.

[16] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 369–376, New York, NY, USA, 2006. ACM.

[17] M. Majer. Diplomová práce - GitHub repozitář, 2018. `https://github.com/MajerMartin/phoneme_classification`.

[18] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778, 2013.