VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

NOVÉ TECHNIKY V OBLASTI TRÉNOVÁNÍ NEURONOVÝCH SÍTÍ - CONNECTIONIST TEMPORAL CLASSIFICATION

NEW TECHNIQUES IN NEURAL NETWORKS TRAINING - CONNECTIONIST TEMPORAL CLASSIFICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MATÚŠ GAJDÁR**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MARTIN KARAFIÁT, Ph.D.**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Gajdár Matúš**
Obor: Informační technologie
Téma: **Nové techniky v oblasti trénování neuronových sítí - Connectionist temporal classification**
New Techniques in Neural Networks Training - Connectionist Temporal Classification
Kategorie: Zpracování řeči a přirozeného jazyka

Pokyny:

1. Seznamte se s problematikou neuronových sítí (NN) a s jejich použitím v oblasti automatického přepisu řeči.
2. Prostudujte Connectionist Temporal Classification (CTC) přístup.
3. Prostudujte toolkit "EESEN" a CNTK umožňující CTC trénovaní/testování.
4. Použijte CNTK toolkit pro natrénování standardní a CTC neuronové sítě.
5. Otestujte různé architektury a efekt inicializace.

Literatura:

- Geoffrey Hinton, Li Deng, Dong Yu... **Deep Neural Networks for Acoustic Modeling in Speech Recognition**, IEEE Signal Processing Magazine (2012)
- LeCun, Y., Bengio, Y. and Hinton, G. E. (2015), **Deep Learning,** Nature, Vol. 521, pp 436-444.
- Alex Graves, Santiago Fernández, Faustino J. Gomez and Jürgen Schmidhuber, **Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks,** Proceedings of the 23th International Conference on Machine Learning (ICML-06)

Pro udělení zápočtu za první semestr je požadováno:

- Splnění prvních 3 bodů zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Karafiát Martin, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Táto bakalárska práca sa zaoberá problematikou neurónových sietí a ich využití v oblasti rozpoznávania reči. Na začiatok si priblížime teóriu rozpoznávania reči, následne na to nadväzuje problematika neurónových sietí spojená s vysvetlením metódy *connectionist temporal classification*. V ďalšej časti sú popísané nástroje vďaka ktorým sme mohli uskutočniť trénovanie neurónových sietí, spojené s popisom jednotlivých experimentov, ktoré sme spravili aby sme zistili vplyv metódy *connectionist temporal classification* na presnosť predpovedania správnych foném. V záverečnej časti sa nachádza zhrnutie práce a celkové zhodnotenie experimentov.

Abstract

This bachelor's thesis deals with neural network and their use in speech recognition. Firstly, there is some theory about speech recognition, afterwards we show theory around neural networks in connection with *connectionist temporal classification* method. In next chapter we introduce toolkits, which were used for training of neural networks and also experiments done by them to find out impact of *connectionist temporal classification* method on precision in phoneme decoding. The last chapter include summarization of work and overall evaluation of experiments.

Kľúčové slová

Rozpoznávanie reči, neurónové siete, CTC, LSTM, CNTK, EESEN

Keywords

Speech recognition, neural network, CTC, LSTM, CNTK, EESEN

Citácia

GAJDÁR, Matúš. *Nové techniky v oblasti trénování neuronových sítí - Connectionist temporal classification*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Karafiát Martin.

Nové techniky v oblasti trénování neuronových sítí - Connectionist temporal classification

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Martina Karafiáta, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Matúš Gajdár
16. mája 2017

Poďakovanie

Chcel by som sa poďakovať vedúcemu práce pánovi Ing. Martinovi Karafiátovi, Ph.D. za jeho pomoc a poskytnutie užitočných rád.

Obsah

Kapitola 1

Úvod

Najznámejším modelom v oblasti spracovania reči sú Skryté Markovove modely (HMM). Pokroky dnešnej doby v oblasti výpočtovej techniky a algoritmizácií vytvorili priestor pre nové technológie akými sú modely neurónových sietí (NN), ktoré sú vo väčšine prípadov rýchlejšie a efektívnejšie. Vďaka týmto pokrokom vzrástla sila jednotlivých komponentov v architektúre automatického rozpoznávania reči (ASR). V oblasti spracovania reči sa hlavne uchytil pokročilejší model neurónových sietí, a to model rekurentných neurónových sietí (RNN), ktorý je komplexnejší a vie si lepšie poradiť s úlohami, pri ktorých sekvencia dát má istú časovú závislosť, teda je dôležité, aby sekvencia dát bola ovplyvňovaná naraz aj predošlými a zároveň nasledujúcimi sekvenciami dát. Spočiatku sa robili pokusy na spojenie modelov RNN s už spomínaným modelom HMM vytvárajúce takzvané hybridné systémy. Tie sú ale v porovnaní so samostatným modelom RNN málo flexibilné a hlavne nevyužívajú plný potenciál sekvenčného modelovania.

Vďaka tomuto postupnému vývoju architektúr sa zjednodušil proces vytvárania nových systémov na spracovanie reči. Zároveň je tu snaha o prepojenie jednotlivých komponentov architektúry ASR a nahradiť ich ideálne jedným modelom. Práve to sčasti umožňuje zavedenie RNN, ktoré má v ponuke rôzne zlepšenia.

V tejto práci sa zameriame práve na jedno z týchto zlepšení, a to technológiu *Connectionist temporal classfication* (CTC). Vďaka tejto technológií je učenie flexibilnejšie, pretože sa nemusia zvukové dáta pred spracovaním neurónovou sieťou zarovnávať na sekvencie.

1.1 Úvod do rozpoznávania reči

Na začiatok si predstavíme pár pojmov, pretože v tejto problematike sa s nimi budeme často stretávať:

- rámec - parametrizovaná časť zvukového signálu
- fonéma - významotvorná hláska v jazyku
- lexikón - obsahuje veľké množstvo slov s ich prepisom do foném

Zvukový signál je vytváraný rečníkom, teda osobou, ktorá vygeneruje a vysloví sekvenciu slov W, ktorá prejde komunikačným kanálom. Vzduchom sa teda šíri akustický signál. Rozpoznávač reči tento signál zachytí, spracuje a následne dekóduje na slovnú sekvenciu W_2, pričom sa snaží o čo najväčšiu podobnosť s pôvodne vyslovenou sekvenciou W.

Typický rečový rozpoznávací systém obsahuje niekoľko častí. Začneme akustickým modelom, ktorý zahŕňa znalosti o zvukových dátach, teda o akustike, fonetike, vplyve prostredia na reč, pohlaví rečníka, jeho dialekte a mnoho ďalších. Jeho blízkou súčasťou je extrakcia významných častí zo zvukového signálu, ktoré sa dávajú na vstup akustickému modelu. Ďalšia časť v systéme sa nazýva jazykový model, ktorý zahŕňa znalosť o skladbe slov v danom jazyku, najmä o ich správnej sekvencii. Pomocou tohto modelu sa s prispením lexikónu z výsledkov na výstupe akustického modelu hľadá najpravdepodobnejšia sekvencia slov.

Spomínané časti obsahujú veľa neznámych spojených s jednotlivými vlastnosťami rôznych rečníkov ako je ich štýl reči, rýchlosť rozprávania, rôzna slovná zásoba, neznáme slová, rozličné nedokonalosti reči a mnoho iných neznámych, ktoré vplývajú na nedokonalosť systému. Úspešný rozpoznávač reči si musí poradiť s čo najväčším počtom týchto nedostatkov.

1.2 Extrakcia príznakov

Rečový signál je pred vstupom do systému rozpoznávania upravený. To znamená, že akustický signál reprezentujeme pomocou istých príznakov, inak povedané, parametrizujeme ho. Vytvorí sa postupnosť vektorov, zachovávajúcich najmä užitočné informácie zo signálu pre následné použitie. Extrakcia príznakov prebieha pomocou rôznych metód, najčastejšie používané sú *MFCC*, *PLP*, *FBank*. Podrobnejšie k týmto metódam v knihe [5]. V našom testovaní budeme používať metódu *FBank*.

1.3 Akustický model

Premieňa rámec rečového signálu na určitú alebo neurčitú fonému. Pre určovanie týchto pravdepodobností sa prevažne používajú už v úvode spomínané HMM. Iný možný prístup k tejto problematike je pomocou neurónových sietí. V akustickom modeli hľadáme najväčšiu pravdepodobnosť fonémy k vstupnému signálu.

$$argmax_W P(W|A) \tag{1.1}$$

Znamená nájdi sekvenciu foném W s najväčšou pravdepodobnosťou k prislúchajúcej postupnosti vektorov, ktoré reprezentujú rečový signál A.

1.4 Jazykový model

Model pomáha určiť sekvenciu slov bez ohľadu na zvukové dáta. Na vstup dostáva postupnosť foném, respektíve výstup z neurónovej siete, na základe ktorých sa ďalej rozhoduje. Pri určovaní pravdepodobnosti nasledujúceho slova sa v rámci tohto modelu berú do úvahy rôzne faktory, ako napríklad pravidlá istého jazyka a k nemu prislúchajúci slovník. Najlepšie je ak slovník obsahuje čo najviac slov z rôznych oblastí komunikácie. V tejto problematike sa najviac dáva prednosť používaniu n-gramových modelov. Najčastejšie sa používajú trigramové, teda vytvárajú postupnosť z troch foném, ale poznáme aj uni-gramové, bi-gramové a pod. Tri-gramové stavy pozostávajú z kombinácií troch foném často vyskytujúcich sa pri sebe. Túto informáciu získavame z lexikónu.

1.5 Hodnotenie chybovosti

Nasledujúca metrika chybovosti na zhodnotenie kvality rozpoznávača sa nazýva *Word Error Rate* (WER). Počíta sa s tromi druhmi chýb. Ak bolo slovo nahradené iným nesprávnym slovo, teda nastala **substitúcia (S)** slova. Správne slovo bolo vynechané, **zmazanie (Z)**, alebo naopak, slovo bolo pridané, **vložené (V)**. Ak všetky tieto typy chýb spočítame, podelíme **celkovým počtom slov vo vete (All)** a následným násobením konvertujeme na percentá, dostaneme hodnotu vyjadrujúcu WER, teda percentuálny úspech rozpoznávača.

$$WER = 100\% \times \frac{S + V + Z}{All} \tag{1.2}$$

Pri hodnotení sa taktiež môže do úvahy brať aj hodnotenie správnosti celých viet. Kvôli slabému slovníku, z ktorého sa nám nepodarilo vytvoriť dostatočne silný jazykový model, sa dá orientovať metrikou hodnotiacou nie kvalitu na základe slov ale foném. Výpočet *Phone Error Rate* je totožný s WER.

Kapitola 2

Neurónové siete v spracovaní reči

Neurónové siete sa v systémoch rozpoznávania reči používajú aj na tvorenie akustických modelov. Sieť na vstup dostane parametrizovaný zvukový rámec a vypočíta vektory posterióznych pravdepodobností (u ktorých platí že ich suma je rovná 1) pre jednotlivé triedy, v problematike reči myslíme prevažne fonémy. Vďaka univerzálnosti neurónových sietí a ich veľkým potenciálom môžeme povedať, že ich využitie je rozsiahle, preto by sme mohli tento model použiť aj pri iných problematikách, ako je napríklad rozpoznávanie jazyka, akustické vyhľadávanie kľúčových slov, identifikácia rečníka a pod. Kľúčovým faktorom pre učenie neurónovej siete je najmä množstvo ale hlavne kvalitné pripravenie dát, či už trénovacích alebo testovacích.

2.1 Neurónové siete

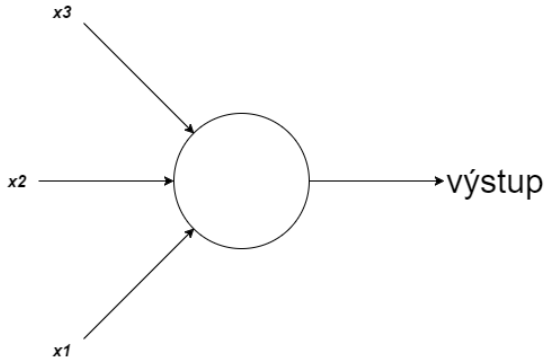
Matematický model vznikol na základe inšpirácie z biologických neurónových sietí. Spočíva v obsahu veľkého množstva jednoduchých prvkov - neurónov, tvoriacich jednotlivé vrstvy siete. Vrstvy sa delia na vstupné, výstupné a medzi nimi sa nachádza niekoľko skrytých vrstiev. Neuróny spolu komunikujú prostredníctvom posielania signálov cez váhové spojenia, tie počas trénovania neustále menia svoje hodnoty - váhy. Po natrénovaní modelu trénovacími dátami váhy nadobudnú fixné hodnoty.

2.1.1 Neurón

Vznikol z binárneho prvku *perceptron* (obrázok 2.1), ten berie na vstup niekoľko vstupov a vyprodukuje jeden binárny výstup. Každý vstup má pridelenú váhu, reálne číslo, ktoré hodnotí dôležitosť vstupnej hodnoty. Perceptron obsahuje ešte jeden významný parameter - hranicu, ten sa porovnáva so sumou všetkých váh násobenými príslušnými vstupmi vchádzajúcimi do perceptronu, a podľa toho sa určí hodnota výstupu buď 0 alebo 1.

$$output = \begin{cases} 0 & ak \quad \sum_j w_j x_j \leq hranica \\ 1 & ak \quad \sum_j w_j x_j > hranica \end{cases} \tag{2.1}$$

Neurón je prvok s výstupom v obore reálnych čísel. Ako už názov napovedá, neurón je základným kameňom neurónových sietí. Výpočet výstupnej hodnoty je v niečom podobný *perceptronu*, ale k súčtu vstupov sa pripočítava ešte konštanta *bias*, ktorá pomáha vyhnúť sa problémom a docieliť lepšie výsledky modelu, napríklad ak by sa na vstup neurónu vložil nulový vektor x, tak by na výstupe bola hodnota nula, a tá by ďalej nevhodne



Obr. 2.1: Príklad vstupného vektoru x do perceptronu

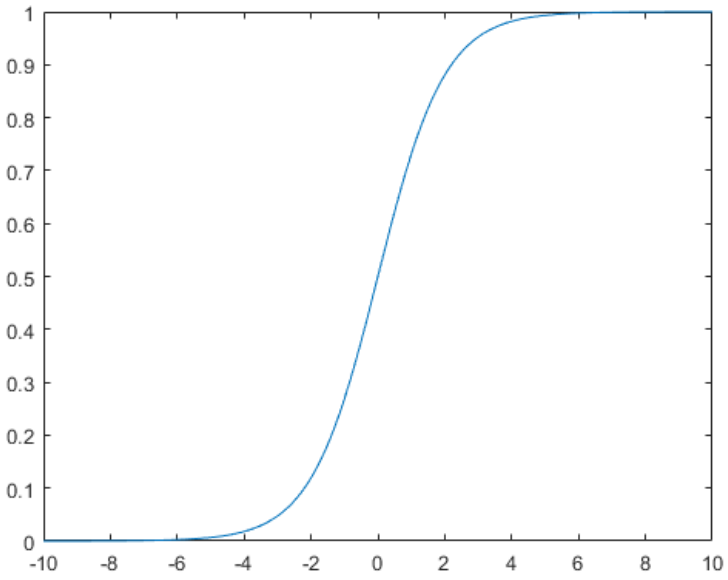
ovplyvňovala ďalšie vrstvy neurónov. Výsledok sčítania je nakoniec ešte transformovaný aktivačnou funkciou *f()*.

$$y = f(\sum_{j=1}^{N} w_j x_j + bias) \tag{2.2}$$

2.1.2 Aktivačné funkcie

Existuje niekoľko aktivačných funkcií. Častejšie používané sú jednoduchšie funkcie pre ich efektívnosť a rýchlosť pri výpočte. Jednou z najznámejších je *logistický sigmoid* (obrázok 2.2). Funkcia pri akomkoľvek vstupe nadobúda hodnoty medzi 0 až 1.

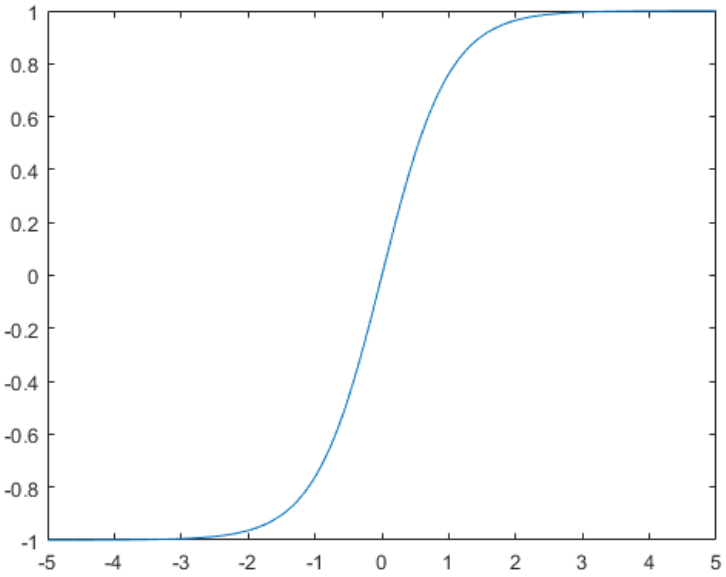
$$\Phi(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$



Obr. 2.2: Logistický sigmoid

Ďalšou taktiež často používanou funkciou je *hyperbolický tangens* (obrázok 2.3). Je výpočetne náročnejší ako sigmoid. Pri tejto funkcii získame z každého vstupu rozmedzie hodnôt medzi -1 až 1.

$$\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.4}$$



Obr. 2.3: Hyperbolický tangens

Výber funkcií závisí z časti na dátach, podľa toho či obsahujú aj záporné hodnoty alebo nie. Hlavne ale závisí na vlastnostiach siete, napríklad sa používa na komprimáciu vstupných hodnôt do prvku *Long-short term memory*, bližšie v kapitole 3.1.

2.1.3 Hodnotiaca funkcia

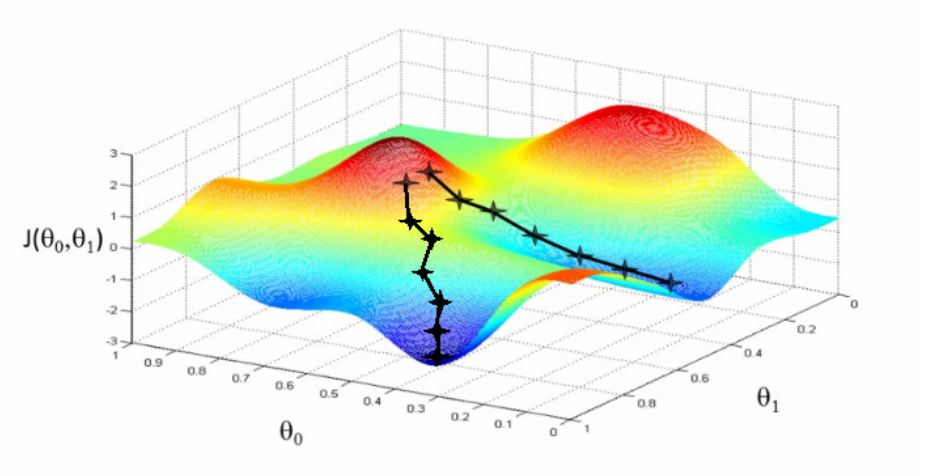
Na úpravu už spomínaných váh a biasu potrebujeme algoritmus. Ale k nemu je ešte potrebná metrika, ktorá zhodnotí ako a o koľko by sa mali jednotlivé váhové parametre upraviť. Postupnou aproximáciou sa snažíme aby vstupné dáta čo najlepšie sedeli do hypotézy $h(x)$. Pre zhodnotenie ako blízko správnemu nastaveniu váh, môžeme použiť jednu z hodnotiacich funkcií:

$$C(w,b) = \frac{1}{n}\sum_{x=1}^{n}[a_x \ln h(x) + (1 - a_x)\ln(1 - h(x))] \tag{2.5}$$

Váhy neurónovej siete označujeme w, písmenom b zase označujeme všetky biasy. Počet všetkých vstupných trénovacích dát je n. Ak ako vstupný vektor označíme x, očakávaným výstupom bude vektor a. Suma je cez všetky trénovacie dáta. Funkcia C sa nazýva *cross entropy error* [10]. Snažíme sa dopracovať k tomu aby $C(w,b) \approx 0$.

2.1.4 Trénovanie

Úlohou neurónovej siete je upraviť váhové koeficienty tak, aby čo najlepšie klasifikovali dáta z trénovacej sady. Využíva sa k tomu algoritmus *gradient descent* (obrázok 2.4), ktorý hľadá minimum funkcie.



Obr. 2.4: Gradient descent, čierne úsečky znázorňujú jednotlivé kroky, ktoré sa blížia k minimám funkcie. Prevzatý z [1]

Trénovanie prebieha v niekoľkých krokoch: **(1)** Najprv je potrebné vybrať sadu trénovacích dát, väčšinou sa tieto dáta ďalej delia na menšie časti, takzvané dávkovanie *batch*. **(2)** Následne na to je potrebné pre každú dávku urobiť tri kroky:

- feedforward - pre každú vrstvu l okrem vstupnej je potrebné vypočítať jej výstupný vektor z

$$z^{x,l} = \Phi(w^l a^{x,l-1}) + b^l \tag{2.6}$$

- výstupnú chybu - vypočítame chybový vektor pomocou závislosti medzi hodnotiacou funkciou a výstupnou vrstvou

$$\delta^{x,L} = C_x \odot \Phi(z^{x,L}) \tag{2.7}$$

- spätná propagácia - pomocou vypočítaného nového vektoru výstupnej chyby spočítame nové chybové vektory pre každú vrstvu

$$\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \Phi(z^{x,l}) \tag{2.8}$$

(3) Ako posledný krok rozšírime sieťou novo získané chybové vektory pre vrstvy a vypočítame nové váhové koeficienty v neurónovej sieti.

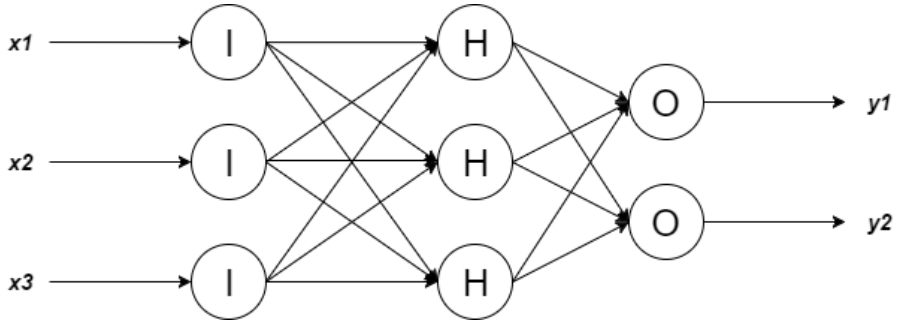
$$w^l \to w^l - \frac{n}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T \tag{2.9}$$

Postupnou iteráciou sa približujeme do minima funkcie. Najvhodnejšie sú funkcie konvexné, vtedy nájdeme globálne minimum. Naopak v prípade nekonvexnej funkcie nevieme zaručiť jeho nájdenie. Neurónove siete sú nekonvexné.

2.1.5 Typy sietí

Siete majú viacero delení. Priblížime si delenie z pohľadu prepojenia neurónov.

Prvou z nich je **dopredná neurónová sieť** (obrázok 2.5). V nej sú neuróny usporiadané do niekoľkých vrstiev, a to tak, aby sa neurón z nižšej vrstvy spájal so všetkými neurónmi vrstvy vyššej, nie naopak. Vstupný signál je teda postupne prenášaný na výstup cez skryté vrstvy. Algoritmy v tomto type siete sú jednoduchšie a ľahšie implementovateľné.



Obr. 2.5: **Vrstvy doprednej neurónovej siete**: symboly **I** označujú vstupnú vrstvu, **H** skrytú vrstvu a **O** výstupnú vrstvu.

V **rekurentnej neurónovej sieti** sú neuróny prepojené v oboch smeroch a taktiež je každý neurón prepojený sám so sebou. Algoritmy sú preto zložitejšie a ťažšie na implementáciu. Taktiež sú pamäťovo a časovo náročnejšie. Grafický nákres takejto siete je neprehľadný, čo už samo napovedá jeho väčšiu zložitosť oproti klasickej doprednej sieti. Viac detailov v kapitole **??**.

2.1.6 Vrstvy doprednej neurónovej siete

Pre doprednú neurónovú sieť s tromi vrstvami (obrázok 2.5) je výpočet neurónu skrytej *j-tej* vrstvy pre *n-tý* učiaci cyklus definovaný ako

$$net_j(n) = \sum_{i=1}^{l} w_{ji} x_j(n) + b_j, j = 1, \ldots, J \tag{2.10}$$

Kde J je počet neurónov v skrytej vrstve a I je počet vstupov do neurónu, w označuje maticu váhových činiteľov a x je vektor neurónov vstupujúcich do danej vrstvy.

Kapitola 3

Rekurentné neurónové siete

V tejto kapitole čerpáme najmä z dokumentov [4] [3] [13]. Hlavnou odlišnosťou ale hlavne výhodou rekurentných neurónových sietí (RNN) oproti klasickým dopredným sieťam je závislosť výstupov nie len na aktuálnych hodnotách na vstupe ale aj udržiavanie si vnútorného stavu histórie hodnôt, ktoré už boli spracované. Vďaka tomu je možné učiť sa časové závislosti medzi vstupnými hodnotami.

Rekurentná neurónová sieť má výhodu učenia sa časových závislostí zo sekvencií. Napríklad ak bude daná vstupná sekvencia $\mathbf{X} = (\mathbf{x_1}, \ldots, \mathbf{x_T})$ a jednotlivé stavy skrytej vrstvy siete $\mathbf{H} = (\mathbf{h_1}, \ldots, \mathbf{h_T})$. Iterovanie bude prebiehať od $t = 1$ po T, tak pre dopredný smer bude vektor skrytej vrstvy vypočítaný nasledovne:

$$h_t = \Phi(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \tag{3.1}$$

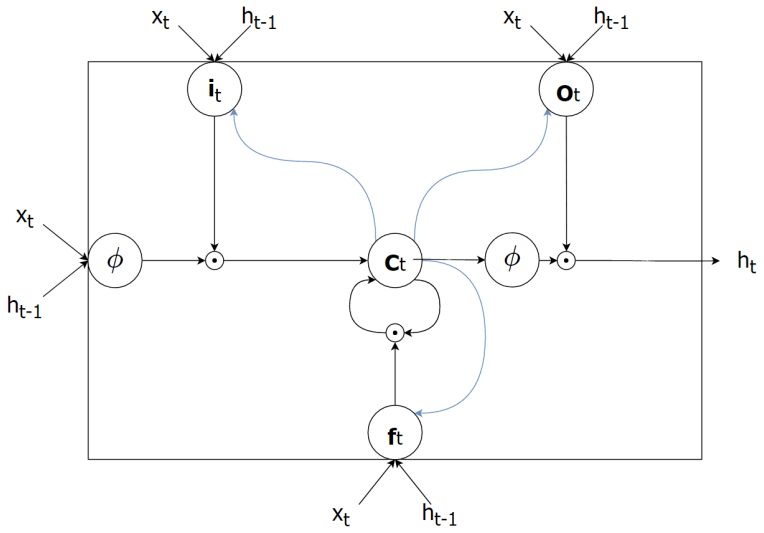
$\mathbf{W_{hx}}$ je váhová matica medzi vstupnou a skrytou vrstvou, $\mathbf{W_{hh}}$ je medzi dvoma skrytými vrstvami. Ku aktuálnemu časovému kroku vstupov $\mathbf{x_t}$ je ešte prirátaná skrytá vrstva $\mathbf{h_{t-1}}$ z predchádzajúceho časového kroku. Popísaný výpočet je nazývaný dopredným, v prípade ak máme dvojsmernú rekurentnú sieť, tak na výpočet spätného smeru použijeme rovnakú rovnicu, akurát budeme iterovať od $t = T$ po 1, čiže v opačnom smere. V každom časovom kroku t potom konkatenujeme výstup z dopredného a spätného výpočtu a dáme na vstup ďalšej, respektíve predchádzajúcej vrstve. Rozšíriť sieť vieme o niekoľko skrytých vrstiev, čím nám vznikne takzvaná hlboká sieť.

3.1 Učenie a Long-short term memory

Učenie RNN je vypočítané pomocou metódy spätnej časovej propagácie. Pri učení RNN dlhodobé časové závislosti sa stretávame s problémom *vanishing gradientu* [2]. Ako riešenie tohto problému sa naskytá použitie prvku *Long-short term memory* (LSTM) (obrázok 3.1).

Tento blok obsahuje pamäťovú bunku c, ktorá je prepojená sama so sebou. Slúži na udržanie informácie o stave prvku. Aktivačné brány, alebo inak povedané vstupné brány, kontrolujú informačný tok v prvku. Výpočet jedného prechodu prvkom v čase t je popísaný ako:

$i_t = \phi(W_{ix}x_t + W_{ih}h_{t-1} + W_{ic}c_{t-1} + b_i)$
$f_t = \phi(W_{fx}x_t + W_{fh}h_{t-1} + W_{fc}c_{t-1} + b_f)$
$c_t = f_t \odot c_{t-1} + i_t \odot \sigma(W_{cx}x_t + W_{ch}h_{t-1} + b_c)$
$o_t = \phi(W_{ox}x_t + W_{oh}h_{t-1} + W_{oc}c_t + b_o)$
$h_t = o_t \odot \sigma(c_t)$



Obr. 3.1: **Long-short term memory**: Jedna pamäťová bunka obsahuje 4 vstupné brány, ktorých vstupom je aktuálny vstupný vektor x_t a stav siete v čase *t-1* označený h_{t-1}. Vo vnútri bunky sa nachádza prvok c_t, ktorý udržiava vnútorný stav bunky. Jeho hodnota ovplyvňuje hodnotu vstupov, výstupov, ale taktiež aj seba samu.

kde $\mathbf{i_t}$ označuje vstupnú bránu, $\mathbf{o_t}$ ako výstupnú bránu, zabúdaciu bránu $\mathbf{f_t}$ a stav pamäťovej bunky $\mathbf{c_t}$. Váhové matice $\mathbf{W_{.x}}$ spájajú vstupy s LSTM prvkom a $\mathbf{W_{.h}}$ matice spájajú predchádzajúcu skrytú vrstvu s LSTM prvkom, $\mathbf{W_{.c}}$ matice sú špeciálne diagonálne prepojenia vychádzajúce zvnútra prvku, teda z pamäťovej bunky, a sú dané na vstup hraničným bránam LSTM prvku. Znak ϕ označuje nelineárny logistický sigmoid a σ je nelineárny hyperbolický tangens.

3.2 Connectionist temporal classification (CTC)

Metóda výstupnej vrstvy rekurentnej neurónovej siete bola špeciálne vytvorená pre úlohy s časovou klasifikáciou, napríklad pre značenie sekvencií, kde je neznáme zarovnanie medzi vstupom a výstupom.

Problém, ktorému sa snaží táto metóda predísť vzniká pri učení s učiteľom, určovaním trénovacích cieľov pre každý segment zo vstupu. To znamená, že potrebujeme segmentovať ručne trénovacie dáta aby sa výstup zo siete zhodoval s testovaným výstupom. Ten je nám vopred známy, aby sme vedeli určiť úspešnosť odhadovania siete pre danú sekvenciu znakov.

3.2.1 Výstup CTC

Výstupná vrstva neurónovej siete pozostáva z počtu labelov, ktoré sú v abecede A plus jeden znak označovaný ako prázdny (*blank*). Hodnoty prvých neurónov určujú pravdepodobnosť znakov z abecedy A, posledný neurón určuje pravdepodobnosť prázdneho znaku. Zadefinujeme si rozšírenú abecedu $A' = A \cup \{blank\}$. Znak y_k^t, určuje pravdepodobnosť že výstup neurónovej siete bude výstupný element k v čase t zo vstupu dĺžky T, vstupnej

sekvencie x.

$$p(\pi|x) = \prod_{t=1}^{T} y_{\pi_t}^t \tag{3.2}$$

Sekvencie π označujeme ako reťazce určujúce pravdepodobnosť sekvencie znakov z množiny A' zo vstupu x. Ďalšou časťou algoritmu pre metódu CTC je spôsob akým tieto sekvencie redukujeme. Potrebná redukcia znakov v sekvencii π prebieha odstránením opakovaných znakov aj prázdnych znakov, ako napríklad v tomto prípade, kde R označujeme redukciu: $R(a - ab-) = R(-aa - -abb) = aab$. Z rozličných reťazcov transformujeme vstup na značenia l. Sekvencia znakov l patrí do množiny A a jej dĺžka je kratšia ako dĺžka reťazcov. Pravdepodobnosť značenia vieme vypočítať ako:

$$p(l|x) = \sum_{\pi \epsilon R^{-1}(l)} p(\pi|x) \tag{3.3}$$

Po výpočte sa nám podarilo zredukovanie niekoľkých rôznych reťazcov na rovnaké značenia. Tento proces umožňuje CTC používať nesegmentované dáta, čo dovoľuje neurónovej sieti predpovedať znak bez znalosti toho, kde sa nachádza.

Blank label

Na začiatku vývoja metódy CTC sa ešte nepoužíval prázdny znak v dôsledku čoho vznikali komplikácie. Ak podľa trénovacieho výstupu mali byť dve rovnaké znaky za sebou, tak redukcia spôsobila, že sa tieto znaky zredukovali na jeden, čo je nesprávny prístup. Ďalším problémom bola zbytočná záťaž systému pri predikovaní znakov aj v miestach kde je len šum alebo hluk medzi jednotlivými slovami. Zavedenie prázdneho znaku teda viedlo k zefektívneniu tejto metódy.

3.2.2 Rozdiel jedno-/dvoj- smernej sieti

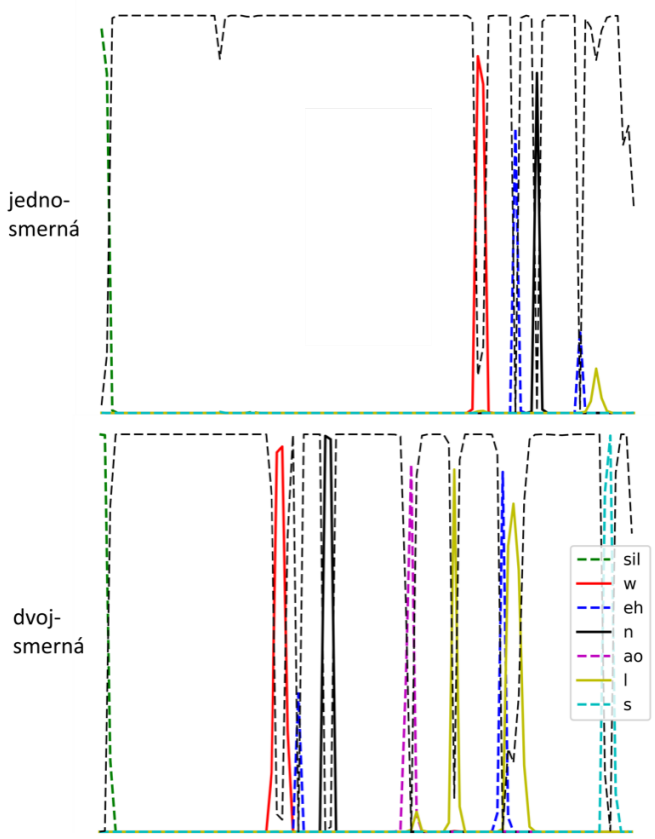
Pri trénovaní jednosmernej sieti výpočet pravdepodobnosti výstupných znakov v čase t ovplyvňujú len vstupné dáta po čas aktuálny teda $t-1$ až 1. Výsledkom toho je predpovedanie znakov prichádza na výstup z určitým oneskorením. Pri obojsmernej sieti je výstup ovplyvnený aj časom $t+1$ až T . Tento prístup má za následok, že znak bude predpovedaný skôr (obrázok 3.2).

3.2.3 Dopredný spätný algoritmus

V kapitole 3.2.1 bol predstavený popis výpočtu pravdepodobnosti $p(l|x)$ (vzorec 3.3), teda možnej sekvencii znakov zo vstupných dát. Priblížime si efektívnejší výpočet tejto pravdepodobnosti. Pretože suma všetkých reťazcov zhodujúcich sa so značeniami závisí na ich dĺžke a toto číslo možných kombinácii rastie exponenciálne.

Algoritmus spočíva v hľadaní zhody prefixov. K tomu znova dopomôže prázdny znak a modifikovaná sekvencia l' doplnená o prázdne znaky na začiatok a koniec sekvencie a medzi každý znak v l. Ak veľkosť l je U tak sa veľkosť sekvencií l' doplnením prázdnych znakov zväčší na $U' = 2U + 1$. Ak budeme mať t ako počet reťazcov a u ako ich dĺžku tak pomocou mapovacej funkcie $R()$ hľadáme prefix z l o dĺžke $u/2$. $V(t,u) = \{\pi \epsilon A'^t : R(\pi) = l_{1:u/2}, \pi_t = l'_u\}$. Pomocou tejto rovnice vieme zadefinovať doprednú premennú $\alpha(t,u)$:

$$\alpha(t,u) = \sum_{\pi \epsilon V(t,u)} \prod_{i=1}^{t} y_{\pi_i}^i \tag{3.4}$$



Obr. 3.2: **Rozdiel medzi jednosmernou a dvojsmernou sieťou**. Výstupné pravdepodobnosti fonému sú reprezentované plnou farebnou čiarou, pravdepodobnosť prázdneho znaku je zobrazená prerušovanou čiernou čiarou. Z obrázku môžeme vidieť že obe siete, či už jednosmerná alebo dvojsmerná správne odhadnú fonémy z dát ale rozdiel je v tom že ich odhadnú v rôznych časoch. Jednosmerná sieť musí počkať kým daný segment istej fonémy prejde sieťou vtedy určí foném, dvojsmerná sieť vie určiť foném pred, počas alebo aj potom. Zaujímavosťou je že dvojsmerná sieť má tendenciu spájať po sebe idúce fonémy, ktoré sa často objavujú spolu (napríklad v angličtine: fonémy 'eh' a 'l', ktoré v kombinácii vytvoria harmonický zvuk ako je v anglických slove 'else')

Všetky správne reťazce musia začať buď z prázdnym znakom alebo prvým symbolom z l, aj z obrázku 3.3 vieme teda zistiť počiatočné podmienky:

$\alpha(1,1) = y_b^1$
$\alpha(1,2) = y_{l_1}^1$
$\alpha(1,u) = 0, \forall u > 2$

Ďalšie premenné sú vypočítané rekurzívne:

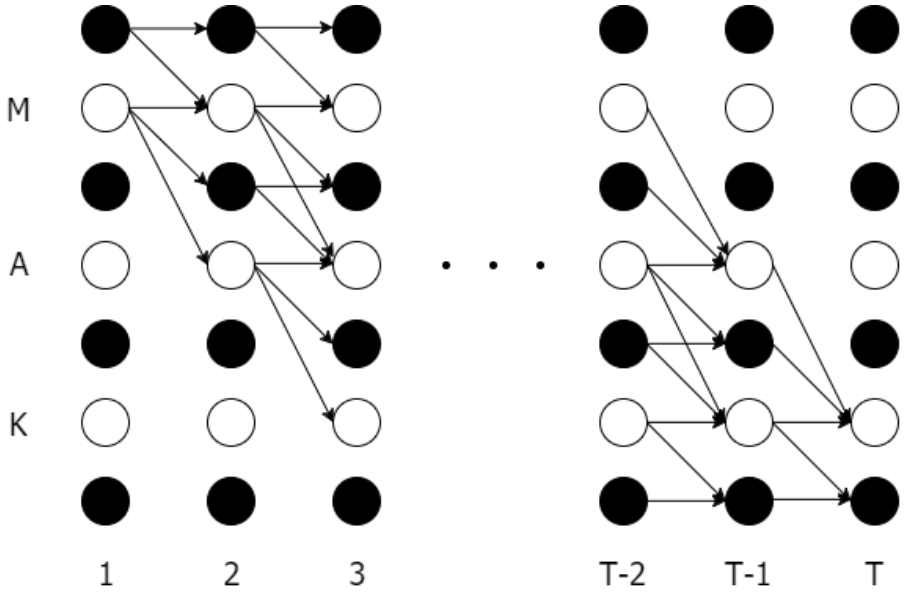
$$\alpha(t,u) = y_{l'_u}^t \sum_{i=f(u)}^{u} \alpha(t-1,i) \tag{3.5}$$

kde ak $l'_u = b$ alebo $l'_{u-2} = l'_u$ tak $f(u) = u - 1$ inak bude $f(u) = u - 2$. $\alpha(t,u) = 0$ v stavoch, ktoré už nemajú dostatok časových krokov pre dokončenie sekvencie.

Pre spätnú premennú je výpočet podobný akurát v opačnom poradí. Podrobnejšie vysvetlenie v literúre [3]. Efektívnejší výpočet $p(l|x)$ dosiahneme práve vďaka výpočtu týchto

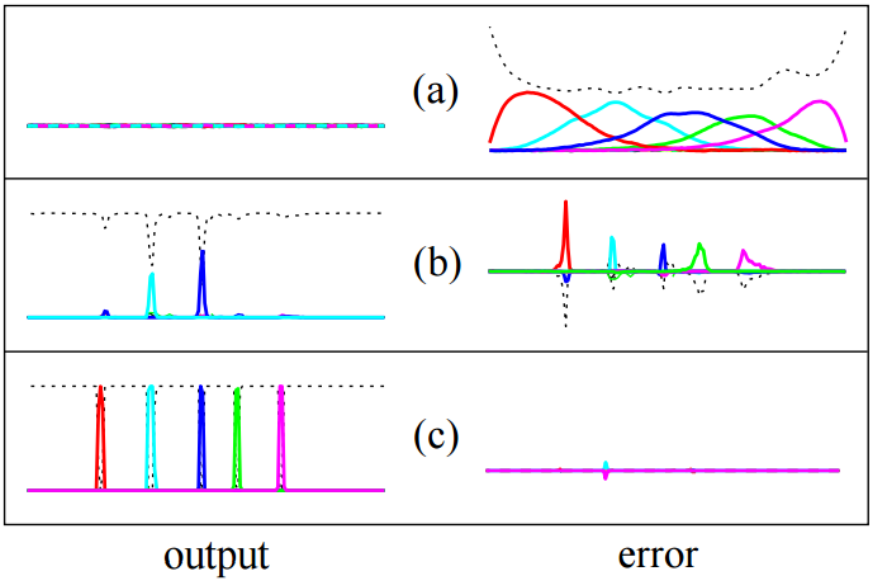
premenných a ich sumy podľa veľkosti rozšíreného značenia l':

$$p(l|x) = \sum_{u=1}^{|l'|} \alpha(t,u)\beta(t,u) \tag{3.6}$$



Obr. 3.3: **Dopredný a spätný algoritmus**: Čierne body označujú prázdne znaky, biele body neprázdne znaky, šípky povolené prepojenia. Dopredné premenné sú prevedené v smere šípok, spätné premenné zas v ich opačnom smere.

Tento výpočet nám umožní určiť *hodnotiacu funkciu*, ktorá zhodnotí nakoľko sa správne zhoduje správna sekvencia znakov z odhadnutou a túto chybu následne rozšírime spätne po sieti vývoj hodnôt môžeme vidieť v obrázku 3.4.



Obr. 3.4: **Vývoj chybového signálu v CTC**: ľavá strana ukazuje výstupné hodnoty pre rovnakú sekvenciu v rôznych štádiách trénovania, pravá strana ukazuje príslušný chybový signál. Ak je chyba nad horizontálnou čiarou výsledky výstupných hodnôt stúpajú, ak pod tak naopak klesajú. Počiatočný stav siete (a). Sieť začne robiť prvé odhady (b). Natrénovaná sieť (c). Prevzatý z [3].

Kapitola 4

Implementácia

Pre trénovanie rekurentných neurónových sietí metódou CTC boli použité dva nástroje: EESEN [8] a CNTK [6]. V nasledujúcich kapitolách si ich predstavíme a urobíme v nich rôzne experimenty.

4.1 TIMIT

Databáza obsahujúca 6300 viet, po 10 vetách povedaných 630 rôznymi rečníkmi v 8 dialektoch zo Spojených štátov amerických. Okrem zvukových nahrávok, obsahuje aj súbor kde sú jednotlivé rámce označkované príslušnými fonémami, ktoré sa majú rozpoznať. Taktiež obsahuje aj prepisy viet k jednotlivým nahrávkam. Nahrávky sú v tejto databáze delené na mužské a ženské nahrávky ale v tomto prípade túto informáciu zanedbávame. Zvolená databáza je pomerne malá. Zvolili sme si ju preto aby výpočty jednotlivých experimentov netrvali príliš dlho.

4.2 EESEN

Nástroj EESEN vychádza zo známejšieho nástroja na trénovanie akustických modelov Kaldi [11]. Ten v sebe ale nemá implementovanú potrebnú metódu CTC, ktorá je popísaná v kapitole 3.2.

4.2.1 Príprava dát

Na trénovanie siete bola použitá databáza TIMIT 4.1.

Na začiatok je potrebné tieto dáta sformátovať a pripraviť aby mohli byť spracované neurónovou sieťou. Dáta sú rozdelené do dvoch priečinkov. V priečinku s názvom *train* sú dáta ktoré budú slúžiť na trénovanie akustického modelu, teda neurónovej siete. Druhý priečinok s názvom *test* sa použije na overovanie percentuálnej správnosti modelu po natrénovaní, tento priečinok obsahuje niekoľko násobne menšie množstvo dát. Vnútri priečinkov sa nachádza niekoľko delení dát do podpriečinkov. Tie obsahujú identifikačný reťazec *id* zvukovej nahrávky a k nemu prislúchajúci prepis slov. Taktiež obsahuje aj súbor so segmentáciou na fonémy. Skripty na spracovávanie sa veľkou časťou zhodujú s postupom v Kaldi.

Dôležitou podmienkou je aby sa v týchto priečinkoch nenachádzali rovnaké dáta, pretože by to mohlo ovplyvniť výsledky. Keďže sieť bude dávať lepšie výsledky na vstupoch, ktoré ňou už prešli. Môže dôjsť aj k takzvanému pretrénovaniu siete na trénovacích dátach. Čo má

za dôsledok neschopnosť siete predpovedať relevantné výsledky z testovacích dát a úspešnosť tohto dekódovania by bola takmer nulová.

Pri prevode slov na fonémy si môžeme vybrať koľko foném budeme používať, originálna databáza obsahuje 60 foném. Pre lepšiu prehľadnosť a presnosť ale zvolíme na učenie nášho modelu menší počet 39 foném, tie upravíme podľa mapy v dokumente [7].

Ako ďalšie si vytvoríme slovník z ktorého budeme vytvárať jazykový model. Slovník sa nachádza v databáze TIMIT ako textový súbor obsahujúci na jednotlivých riadkoch slovo a k nemu prislúchajúci prepis do foném. Tento súbor sa upraví tak že sa nahradia jednotlivé fonémy za ich identifikačné čísla *id*, to sa určí podľa súboru, kde je určené mapovanie fonémy a jeho *id*.

Jazykový model

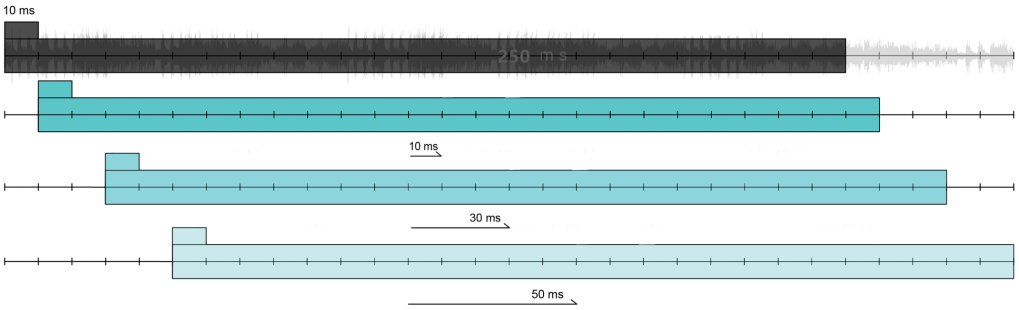
Jazykový model vytvoríme pomocou nástroja *OpenFST* s rozšíreniami [9]. Pre vytvorenie *FST* využívame funkcie so spomínaného nástroja. Ako prvé skompiluje vstupné tokeny (fonémy), ktoré budeme využívať v modely vytvorenom pomocou *fst* nástrojov a skompiluje súbor *T.fst*. Pre EESEN a CNTK sa bude pozícia prázdneho tokenu líšiť, v prvom menovanom bude na začiatku po symbole epsilon v druhom prípade bude posledný. Pre vytvorenie ďalšej časti jazykového modelu ktorou je Lexikón *L.fst*, potrebujeme dať na vstup funkcie symboly foném *units* a slovník *words* s prepisom slov na fonémy. Ako ďalšiu si vytvoríme gramatiku, ktorá sa vytvorí pomocou viet ktoré zozbierame z dát prepisov reči v databáze TIMIT. Kde pomocou nástrojov z Kaldi vytvoríme n-gramy zo slov, vytvárame tri-gramy. Už potrebujeme len zoznam slov *words* a môžeme vytvoriť poslednú komponentu *G.fst*. Nakoniec všetky tieto časti **.fst* skompilujeme a vytvoríme dekódovací nástroj, ktorým budeme neskôr hodnotiť vytvorený akustický model v našom prípade neurónovú sieť. Kvôli zlým výsledkom vytvoreného jazykového modelu so slovníka v databáze TIMIT, bol použitý jazykový model vytvorený z foném. V slovníku sa namiesto prevodu slov na fonémy nachádzajú len dvojice identických foném.

Extrakcia audio signálu

Ako ďalšie potrebujeme spracovať audio nahrávky. Z jednotlivých audio súborov si pomocou nástroja *compute-fbanks* z nástroja Kaldi vytvoríme takzvané *fbank*, čo sú inak povedané reálne hodnoty ktoré opisujú audio signál. Nastavenia prevodu sú určené v konfiguračnom súbore alebo sa použijú východiskové hodnoty. Dĺžku rámca ponecháme na 250 milisekundách. Jeden rámec bude prevedený na vektor 40 reálnych čísel. Rámec sa bude posúvať vo východiskovom nastavení o 10 milisekúnd (obrázok 4.1). Použijeme však aj posuny o väčší počet napríklad o 30, 50 milisekúnd aby sme zistili aký dopad to bude mať na jednotlivé modely. Hodnoty sú vždy vytvorené pre EESEN a potom pre CNTK sú konvertované na formát HTK, podrobnejšie v dokumente [14].

Topológia siete

Ďalším krokom je modelovanie topológie neurónovej siete, kde si určíme počet vstupných hodnôt siete. Tie sa odvíjajú od počtu hodnôt, ktorými je opísaný jeden rámec (teda 40) ale aby bol signál audia popísaný podrobnejšie pridáme delty (opisujúce dynamiku signálu vo fixných hodnotách priradených na koniec rámcového vektoru), ktoré znásobia počet hodnôt na 120. Určíme si aj vhodný počet LSTM vrstiev a počet buniek ktoré budú obsahovať. Počet výstupných hodnôt sa určí podľa počtu foném plus jeden prázdny znak. Teda v



Obr. 4.1: **Rámec a jeho rámcové posuny**: Jeden diel predstavuje 10 milisekundovú dĺžku audio signálu, rámec má dĺžku 250 milisekúnd, pod sebou vidíme rôzne posuny o ktoré sa rámec posúva prípade rôznych rámcových posunov o 10, 30 a 50 milisekúnd.

našom prípade 39 foném a jeden prázdny foném takzvaný *blank* (tabuľka 4.1). Rozdiel medzi EESEN a CNTK je pri poradí výstupných foném kde v EESEN je blank ako prvý v poradí a v CNTK je ako posledný, preto bolo potrebné vytvoriť aj dva jazykové modely.

Na začiatku trénovania určíme *learning rate*, jeho hodnota sa počas učenia mení podľa toho ako sa zmenšuje percentuálny rozdiel presnosti siete po sebe nasledujúcich epoch. Ak toto zlepšenie je už zanedbateľné menšie ako 0.05% tak sa učenie ukončí. Prepočítanie tohto rozdielu sa deje po každej epoche, teda jednom kole trénovania testovacích dát. Množstvo epoch počas trénovania môže byť rôzne. Dôležité je nájsť správny počet a predísť tak pretrénovaniu. *Momentum* je ďalšia hodnota, ktorú potrebujeme správne nastaviť, určuje akú veľkú váhu budú mať predchádzajúce váhy na aktuálne. Väčšinou táto hodnota pomáha niekoľkonásobne urýchliť trénovanie. V našom prípade budeme používať najmä hodnotu 0.9.

Nastavenie topológie siete je veľmi diskutovaná téma a neexistuje jednoznačný vzorec podľa ktorého sa dá vypočítať dokonalé nastavenie vrstiev a počet ich buniek. Zlým nastavením by bolo ak by skryté vrstvy mali menší počet neurónov ako je veľkosť výstupného vektoru. Vtedy by sme stratili množstvo informácii na rozhodnutie ku ktorej kategórii priradiť vstupný vektor.

4.2.2 Experimenty

Pomocou objektívnej metódy CTC a dvojsmernej rekurentnej neurónovej sieti s bunkami LSTM sme pomocou dát z databáze TIMIT vyskúšali rôzne hodnoty vrstiev od jednej až po sedem s počtom neurónov 256. A zistili ako sa bude sieť chovať. Počiatočný learning rate je 0.00004, momentum má hodnotu 0.9.

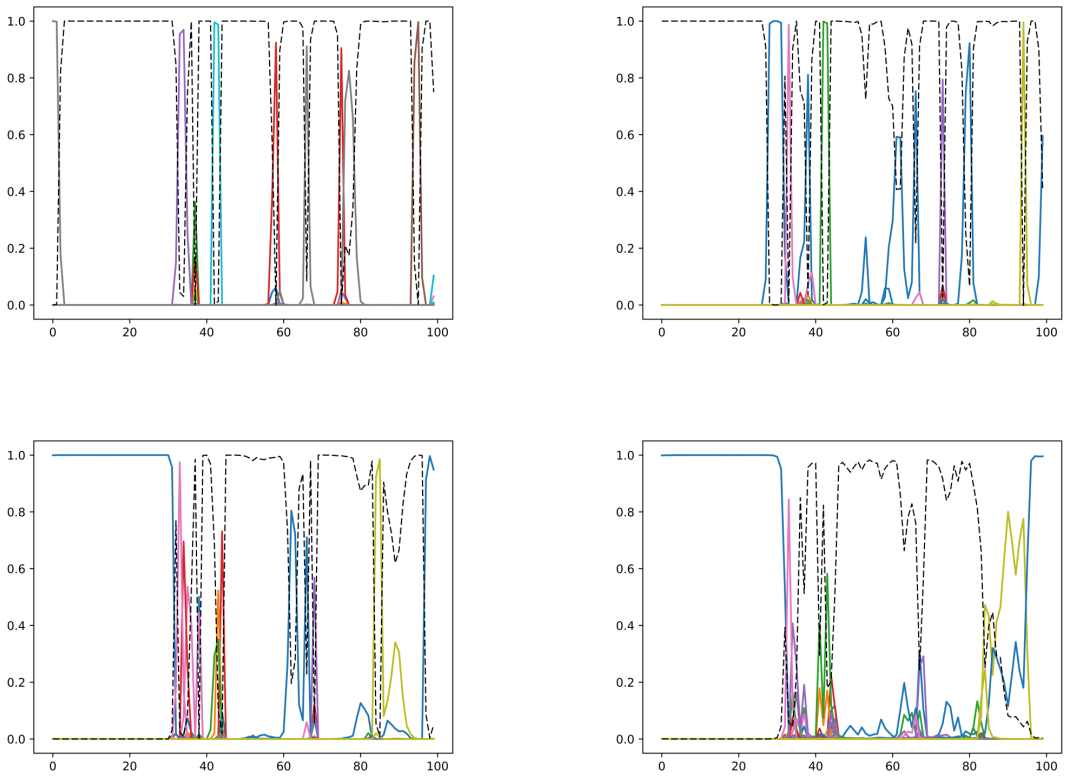
Vyberieme si topológiu s počtom vrstiev s najlepším výsledkom. Vyberieme si teda sieť so 4 vrstvami. Vyskúšame vplyv *learning rate* argumentu na neurónovú sieť s metódou CTC. Skúsime sa posunúť v rozmedzí desatiny a desaťnásobku (tabuľka 4.2). Po vykonaní experimentu pozorujeme pri znížení *learning rate*, že sieť podáva presnejšie výsledky a výpočet zbehol rýchlejšie. Naopak pri zmenšení hodnoty sa sieť nedokázala v rozumnom čase ani len priblížiť k relevantným výsledkom.

blank	0/39	k	20
aa	1	l	21
ae	2	m	22
ah	3	n	23
aw	4	ng	24
ay	5	ow	25
b	6	oy	26
ch	7	p	27
d	8	r	28
dh	9	s	29
dx	10	sh	30
eh	11	sil	31
er	12	t	32
ey	13	th	33
f	14	uh	34
g	15	uw	35
hh	16	v	36
ih	17	w	37
iy	18	y	38
jh	19	z	39

Tabuľka 4.1: Fonémy s identifikačným číslom

V konfigurácii topológie siete sa dá nastaviť používanie buď jednosmernej alebo dvojsmernej siete. Tento parameter má menší vplyv na úspešnosť siete. Hlavným rozdielom je ako už sme spomínali v časti 3.2.2, že jednotlivé fonémy sú pri dvojsmernej sieti predpovedané skôr. Sieť nastavíme rovnako ako v predchádzajúcom prípade zvolíme *learning rate* 0,0004 so 4 vrstvami a 256 bunkami LSTM (tabuľka 4.4).

Pri trénovaní rozpoznávača reči používame rámce o dĺžke 250 milisekúnd z ktorého vyextrahuje príznaky (podkapitola 1.2). Tieto rámce sa ale posúvajú v rôznych intervaloch. Klasickým a overeným intervalom je posun o 10 milisekúnd. Iné rámcové posuny sa používajú prevažne v prípade experimentovania. V rozpoznávačoch aké poznáme dnes je tento posun väčšinou klasický. Vyskúšali sme si teda trénovanie neurónovej siete s rámcovým posunom iným ako je 10 milisekúnd. Rôzny posun vplýva aj na čas trénovania, keďže máme väčšie posuny máme menej vstupných dát pre neurónovú sieť. Najrýchlejšie sa teda natrénuje sieť s najväčším rámcovým posunom. Z audio signálu sme si pripravili posuny o 20, 30 a taktiež 50 milisekúnd (obrázok 4.1). Výsledky niektorých posunov by podľa dokumentu [12] mali dosahovať lepšie percentuálne výsledky pri trénovaní s metódou CTC. Nám sa podarilo sieť natrénovať lepšie len v prípade 20 milisekundového posunu (tabuľka 4.5), pravdepodobne kvôli malému množstvu trénovacích dát. V iných prípadoch ale dostávame horšie výsledky ako v prípade základného posunu. Zmeny výstupov siete môžeme vidieť na obrázku 4.2.



Obr. 4.2: **Sieť natrénovaná metódou CTC s rôznym rámcovým posunom**: Farebné čiary označujú neprázdne fonémy. Čierna čiarkovaná čiara označuje prázdny znak. Vľavo hore je klasický posun o 10 ms. Vpravo hore je posun o 20 ms na ktorom vidíme prázdny znak odhadovaný s menšou presnosťou, nie je jednoznačne predpovedaný v polohe 1.0 ako v prípade 10 ms posunu. Vľavo dole máme posun o 30 ms na ktorom môžeme pozorovať väčšie nepresnosti prázdneho znaku, ten už neurčuje tak jednoznačne miesta kde by sa mali nachádzať fonémy. Spodný graf sú rámce s posunom 50 ms, tu pozorujeme ešte väčšiu nejednoznačnosť pri určovaní foném.

počet vrstiev	WER
1	29.26 %
2	34.64 % (len 5 epoch)
3	26.31 %
4	25.61 %
5	26.79 %
6	25.97 %
7	27.69 %

Tabuľka 4.2: Skryté vrstvy

learning rate	WER
0.00004	25.61 %
0.0004	25.21 %
0.000004	fail

Tabuľka 4.3: Learning rate

4.3 CNTK

Nástroj CNTK je vyvíjaný firmou Microsoft a jeho zdrojové súbory sú voľne dostupné. Aktuálne sa na vývoji nástroja pracuje, nové verzie sa vydávajú priemerne raz za mesiac. Jeho dynamický vývoj spôsobil občasné problémy pri prekladaní a spúšťaní, keďže sa funkcionalita neustále rozširuje. Na rozdiel od nástroja EESEN sa CNTK vie uplatniť aj v iných problematikách akou je rozpoznávanie reči. Dobré výsledky má najmä v oblasti rozpoznávania obrazu. Tento nástroj je taktiež používaný pri vývoji virtuálneho asistenta Microsoftu, nazývaného *Cortana*. Zaujímavým rozšírením pre CNTK je jeho spojenie s nástrojom Kaldi, vďaka čomu sme mohli dekódovanie urobiť pomocou už fungujúcich skriptov s pár úpravami. Podobne ako pri EESEN budeme skúšať natrénovať neurónové siete na databáze TIMIT.

4.3.1 Konfigurácia

Na to aby sme mohli trénovať sieť s nástrojom CNTK, budeme potrebovať vytvoriť konfiguračný súbor, ktorý sa priradí ako parameter *configFile* pre binárny program CNTK. Konfiguračný súbor obsahuje niekoľko častí, tou najhlavnejšou je príkaz ktorý určí čo sa má vykonať, poznáme akcie typu *train*, *evaluate*, *deploy*, *write*. Využijeme najmä akciu *train* a akciu *write* spojenou s nástrojom Kaldi pre dekódovanie a výpočet WER. V súbore sa jednotlivé úseky oddeľujú hranatými zátvorkami. V nich sú premenné určujúce akým spôsobom bude daná sieť nakonfigurovaná, väčšina hodnôt už je implicitne nastavená.

typ siete	WER
jednosmerná	26.06 %
dvojsmerná	25.21 %

Tabuľka 4.4: Typ siete

posun rámca (v ms)	WER
10	25.21 %
20	23.48 %
30	26.57 %
50	40.80 %

Tabuľka 4.5: Rámcové posuny

```
#CNTK TIMIT training
command = TIMITtrain
...
TIMITtrain = [
        action = "train"
        ...
]
```

Vnútri sekcie príkazu *TIMITtrain* sa nachádzajú ďalšie podčasti:

- konfigurácia učenia stochastického gradientu - *SGD*
- čítačka dát - *reader*
- popis samotnej siete pomocou jazyka *Brainscript*.

Pri konfigurácii sekcie učenia nastavujeme *learning rate*. Ten je buď fixný pre všetky epochy alebo jeho hodnoty môžeme určiť pre jednotlivé epochy zvlášť a to pomocou znaku dvojbodky *':'* čo označuje jednu epochu, napríklad *0.05:0.02*, posledná hodnota v poradí bude používaná až do poslednej epochy, ak nie je nastavený *AutoAdjust*. Zápis sa dá aj zjednodušiť z formátu *0.5:0.05:0.05:0.02* na *0.5:0.05*2:0.02*. Pri premennej *momentum* zápis funguje rovnako. *Learning rate* vieme určiť pre jeden *minibatch*, po prípade ho určujeme na jednej vzorke. Vtedy by sme museli zmeniť premennú *learningRatesPerMB* na *learningRatesPerSample*.

Po skúšaní rôznych nastavení *learning ratu* sme zistili že najlepšia možnosť je automatické upravovanie počas učenia, podobne ako v prípade nástroja EESEN. Určiť nám stačí počiatočný stav, hranicu pri ktorej sa upraví learning rate o násobok znižovacieho faktoru. V tejto sekcii určujeme aj počet epoch a taktiež veľkosť epochy. V prípade hodnoty 0 sa použijú všetky dáta. Je tu možnosť nepoužiť pri každej epoche všetky dáta z trénovacej sady, ale len istý počet ak nastavíme hodnotu na inú ako 0. *Minibatch size* určuje z koľkých dát budeme vypočítavať *gradient*, slúži hlavne na zrýchlenie učenia ale môže ovplyvniť aj presnosť.

```
TIMITtrain = [
        action = "train"
        SGD = [
                epochSize = 0
        minibatchSize = 6000
        learningRatesPerSample = 0.0005
        AutoAdjust=[
            autoAdjustLR="adjustAfterEpoch"
            reduceLearnRateIfImproveLessThan = 0.0001
            learnRateDecreaseFactor = 0.5
        ]
        maxEpochs = 100
        keepCheckPointFiles = true
        ]
        reader = [
                ...
        ]
        BrainScriptNetworkBuilder = {
                ...
        }
]
```

Ďalšou časťou je sekcia čítačky dát (*reader*), pomocou ktorej spracovávame dáta pre neurónovú sieť. Premenná *features* obsahuje cestu k súboru s cestami ku zvukovým dátam vo formáte *fbank*, v prípade CNTK sú dáta vo formáte HTK. Transformovanie formátovania urobíme pomocou Kaldi nástroja *copy-feats-to-htk*.

Čítačka si načítava dopredu viac rámcov. Vstupný rozmer zo zvukových dát je n-násobok veľkosti jedného rámca, v našom prípade je to $n = 5$. Formát *mlf* sa používa na popis *labelov* a k nim prislúchajúcim časovým ohraničeniam v ktorých sa vyskytujú. Premenná *labelMappingFile* ukazuje na súbor v ktorom sa nachádzajú fonémy na rozpoznávanie v jednotlivých riadkoch. Musia sa zhodovať s fonémami v súbore *.mlf*. Zároveň sa musí rovnať rozmer a počet foném.

```
reader = [
        ...
    deserializers = (
    [
        type = "HTKFeatureDeserializer"
        module = "HTKDeserializers"
        input = [
          features = [
          dim=600
          scpFile = "$DataDir$/feats.scp"
          ]
        ]
    ]:
    [
        type = "HTKMLFDeserializer"
```

```
            module = "HTKDeserializers"
            input = [
              labels = [
                        dim = 40
                mlfFile="$DataDir$/phone_timing.mlf"
                labelMappingFile = "$DataDir$/phone.list"
                labelType=Category
                phoneBoundaries=true
              ]
            ]
        ]
      )
]
```

Sieť v CNTK je poskladaná s komponentov LSTM, ktoré majú nasledujúci popis (vzorce 3.1), *it* označuje vstupnú bránu, *ft* zabúdaciu bránu, *ot* výstupnú bránu a *ct* je stav bunky. Táto bunka sa pozerá len na minulé hodnoty takže je jednosmerná. Na vstup dostáva hodnoty rozmerov vstupného a výstupného vektoru, počet LSTM buniek a samozrejme váhové hodnoty ako premennú *inputx*.

```
LSTMPComponentWithSelfStab(inputDim, outputDim, cellDim, inputx) =
  [
   B() = BiasParam(cellDim)
   Wmr = WeightParam(outputDim, cellDim);
   W(v) = WeightParam(cellDim, inputDim) * Stabilize(v)
   H(h) = WeightParam(cellDim, outputDim) * Stabilize(h)
   C(c) = DiagTimes(WeightParam(cellDim, 1), Stabilize(c))
   // LSTM cell
   dh = PastValue(outputDim, output);
   dc = PastValue(cellDim, ct);

   it = Sigmoid(W(inputx) + B() + H(dh) + C(dc))
   bit = it .* Tanh(W(inputx) + (H(dh) + B()))

   ft = Sigmoid(W(inputx) + B() + H(dh) + C(dc))
   bft = ft .* dc

   ct = bft + bit

   ot = Sigmoid(W(inputx) + B() + H(dh) + C(ct))
   mt = ot .* Tanh(ct)
   output = Wmr * Stabilize(mt)
]
```

Na konci popisu topológie siete sa nastavujú jednotlivé vektory pre čítanie dát pomocou skôr spomínaného *readeru*. Následne sú tieto dáta ešte upravené. Môžeme si všimnúť posun rámcov ktoré sa dávajú na vstup LSTM. Ako posledná vrstva sa dá jednoduchá vrstva s počtom neurónov, ktorý je rovný počtu výstupných fonémov vrátane prázdneho znaku.

Na trénovanie siete v CNTK je potrebné využiť nižšie opísané funkcie.

Pre učenie sa používa funkcia *ForwardBackward* na ktorej vstup sa dávajú argumenty v nasledovnom poradí: graf vytvorený z labelov, výstupná vrstva, pozícia prázdneho znaku (po testovaní sme prišli na to že toto číslo musí byť vždy posledné v poradí), argument meškania výstupu, tag, ktorý označuje aká funkcia sa má použiť na učenie.

Pre výpočet chyby sa používa funkcia *EditDistanceError* na ktorej vstup sa dávajú argumenty v nasledovnom poradí: vektor veľkosti výstupných labelov, výstup zo siete, argument určenia či brať do úvahy časové ohraničenia, poradie tokenu ktorý sa má ignorovať, tag, ktorý označuje aká funkcia sa má použiť na výpočet chyby.

Funkcie *LabelsToGraph* používa sa na inicializáciu grafu pre CTC metódu (obrázok 3.3).

```
// training
graph = LabelsToGraph(labels)
cr = ForwardBackward(graph, NetOutput, 39, delayConstraint=3,
tag="criterion")
Err = EditDistanceError(labels, NetOutput, squashInputs=false,
tokensToIgnore = 39, tag="evaluation")
```

4.3.2 Experimenty

Topológiu siete sme nastavili na základe experimentov z nástroja EESEN. Sieť bude mať 4 skryté vrstvy po 256 bunkách LSTM, na vstupe bude rámec opísaný 120 hodnotami a výstup bude mať 40 prvkov. V ďalších experimentoch sa budeme zaoberať inicializáciou siete, rôznymi nastaveniami premenných vo funkciách, úpravou vstupných dát a pod. Našim cieľom je priblížiť sa inicializáciou výsledkom podobným v nástroji Kaldi.

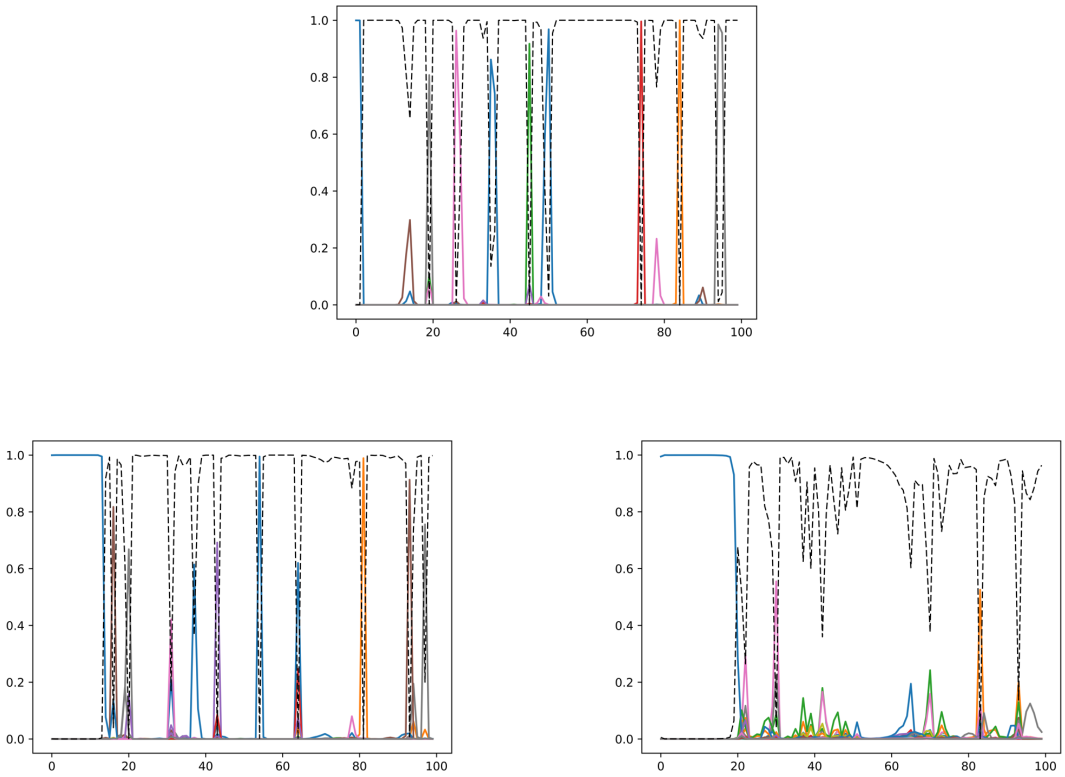
Vplyv argumentov funkcií

Argument **tokensToIgnore** nastavuje index prázdneho znaku, ten by sa mal zhodovať so vstupným súborom *labelMappingFile*, v ktorom sú jednotlivé prázdne aj neprázdne znaky. Pozícia prázdneho znaku musí byť vždy na konci tohto súboru, teda aj index musí byť vždy rovný maximálnemu počtu znakov mínus jedna, pretože indexujeme od nuly. Pri skúšaní iných indexov pre prázdny znak sme neuspeli a sieť vždy považovala prázdny znak ako posledný.

Nastavenie argumentu **squashInputs** vo funkcii *EditDistanceError* určuje či budeme redukovať sekvencie identických sekvencií (podkapitola 3.2.1). Experimentom sme zistili, že na výsledky siete to nemalo žiaden vplyv v obidvoch prípadoch sme dostali rovnaké výsledky WER rovné 28.54 %.

Argument **phoneBoundaries** sa má podľa dokumentácie v prípade CTC trénovania zapnúť, pretože v tom prípade sa bude brať v úvahu ohraničenie jednotlivých fonémov, čo sa ale veľmi nezhoduje s teóriou vysvetlenou v dokumente [3]. V čítačke dát sme teda nastavili argument na *true*, inak výpočet funkcie *ForwardBackward* nekonvergoval a pri pohybe vo veľmi vysokých hodnotách nakoniec spadol na chybe *nan*.

Argument **delayConstraint** nadobúda celé hodnoty začínajúce od -1. Nižšie číslo zapríčiňuje kratšie oneskorenia *labelov* pri dekódovaní. Vplyv na presnosť sme preto skúsili otestovať s rôznymi hodnotami.



Obr. 4.3: **CTC vplyv delayConstraint**: Graf vľavo dole je sieť natrénovaná s delayConstraint=7, vpravo dole delayConstraint=10. Prázdny znak *(čiarkovaná čierna čiara)* je najviac predpovedaný token v druhom prípade, čo je chyba. V prvom prípade pri nižšej hodnote delayConstraint sa situácia o čosi zlepšuje, ale niektoré neprázdne znaky *(farebné čiary)* nie sú tak jednoznačne predpovedané. Lepšie výsledky dosahujeme ak je delayConstraint=-1 ako je aj vidieť na hornom grafe; prázdny znak a neprázdne znaky majú vhodný priebeh.

Pri pohľade na grafy, kde má *delayConstraint* nižšie hodnoty pozorujeme vo výsledkoch menšie rozdiely v úspešnosti, pri vyšších sa nám už dekódovanie vymyká očakávaným výsledkom. V prípade *delayConstraint* rovnému 10 sa nám ani nepodarilo odmerať úspešnosť (tabuľka 4.6).

Pred vstupom do siete si vstupný vektor upravíme pomocou funkcie ***RowSlice****(začiatok, počet, vstup)*. V nasledujúcej ukážke sekcie konfiguračného súboru si zo série piatich vektorov o dĺžke 120 vyberáme piaty. Urobili sme tak posun $x_{t+5} -> x_t$. Vyskúšali sme posun o rôzne hodnoty (musí sa jednať o nepárne číslo) a porovnali výsledky (tabuľka 4.7).

```
baseFeatDim = 120
featDim = 5 * baseFeatDim
features = Input{featDim}
feashift = RowSlice(featDim - baseFeatDim, baseFeatDim, features);
```

Pri každom výpočte nového gradientu, bude jeho posun ovplyvňovať istý počet vstupných dát, podľa toho aké číslo bude dosadené do premennej **miniBatchSize**. Tento prístup

delayConstraint	WER
-1	27.80 %
1	31.10 %
3	28.54 %
5	27.36 %
7	33.08 %
10	fail

Tabuľka 4.6: Vplyv konštanty delayConstraint na trénovanie

posun vektorov o	WER
1	28.81 %
3	29.09 %
5	28.54 %
7	30.12 %
11	30.48 %

Tabuľka 4.7: Posuny vektorov

má väčšinou za následok rýchlejšie natrénovanie siete pretože sa nemusí prepočítavať po každom vstupe. Preto nás zaujímalo akú váhu bude mať tento parameter pri trénovaní siete s metódou CTC (tabuľka 4.8).

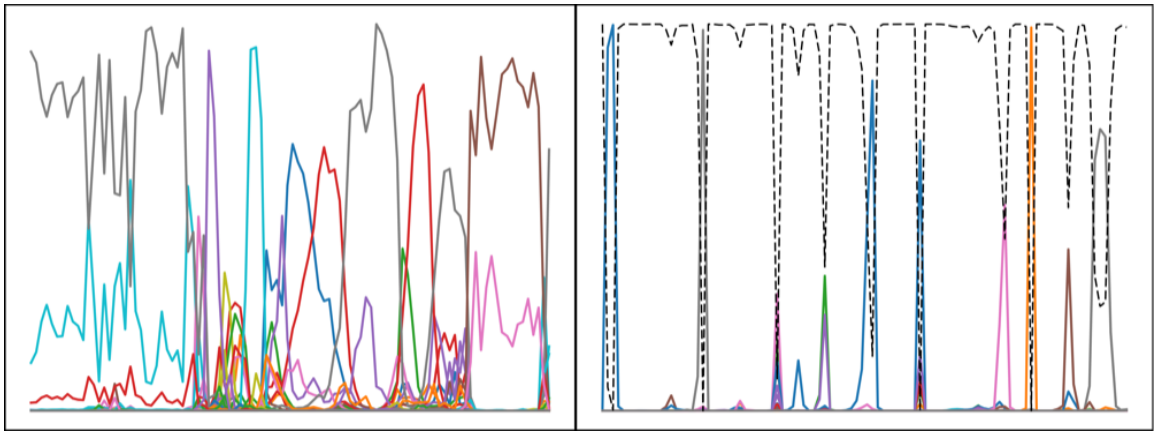
miniBatch	WER	čas jednej epochy (sekúnd)
1	fail	-
10	29.59 %	1818.5
3000	29.76 %	319.7
6000	28.54 %	198.8

Tabuľka 4.8: Vplyv miniBatch

Po otestovaní sme zistili výrazný vplyv *miniBatch* na rýchlosť trénovania, v čoho dôsledku sme v každom inom teste používali len najvyššiu možnú hodnotu. Vyššiu hodnotu konštanty *miniBatch* sme nemohli otestovať kvôli nedostatočným prostriedkom.

Dopredná sieť a metóda CTC

Pre vytvorenie klasickej neurónovej siete bez komponentov LSTM stačí zostaviť sieť z klasických dopredných vrstiev a naskladať ich na seba. Hlavným rozdielom medzi natrénovaním siete čisto pomocou hlbokých neurónových sietí a rekurentných neurónových sietí s metódou CTC je hodnota výstupného vektora, ktorý v prípade CTC metódy z veľkej časti jednoznačne určí fonému ktorá sa dekóduje z určitého rámca, na rozdiel od klasickej siete kde to už nie je tak jednoznačné a fonéma si svoju pravdepodobnosť drží po dlhšiu dobu, dochádza často aj k prekrývaniu.



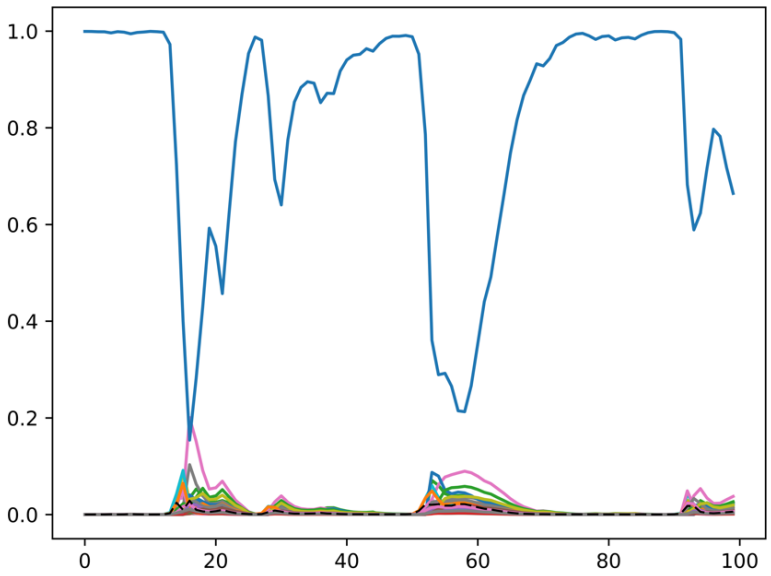
Obr. 4.4: **Rozdiel medzi výstupom s klasickým Cross Entropy a CTC**: vľavo je sieť natrénovaná pomocou klasickej metódy cross entropy, vidíme že fonémy sú roztiahnuté cez viacero rámcov, na rozdiel od metódy CTC vpravo, kde môžeme pozorovať aj prázdny znak *(čiarkovaná čierna čiara)* ktorý má väčšinou hodnotu 1 v prípade že nie je rozpoznaný žiadna foném.

Skúsili sme aj spojenie doprednej neurónovej siete s metódou CTC, výsledky sa ani po dlhom trénovaní nedostali na úroveň poskytujúcu aspoň čiastočné výsledky. Z obrázka 4.5 si môžeme všimnúť, že graf sa veľmi nepodobá grafu so správne natrénovanou sieťou s CTC na obrázku 4.4 vpravo.

Vplyv vstupných dát

Po vytvorení konfiguračného súboru je potrebné pripraviť dáta ktoré sa budú spracovávať. Na to sme využili dáta z nástroja Kaldi, ktoré boli konvertované pomocou nástrojov do formátu HTK pre CNTK. Súbor *mlf* obsahuje časové zarovnanie v ktorom sa vyskytuje daný foném napísaný na konci riadku. Z pohľadu CTC je tento súbor nepotrebný pretože hlavnou výhodou tohto prístupu je, že tieto zarovnania vôbec nepotrebujeme vedieť, stačí nám len sekvencia foném ako idú za sebou. Ako to je v prípade trénovania pomocou nástroja EESEN, kde sa nachádzajú na riadku len *id* nahrávky a k nemu vypísaná postupnosť foném bez akéhokoľvek časového ohraničenia.

```
#!MLF!#
"*/fcjf0_si1027.lab"
0 900000 sil
900000 1900000 sil
1900000 2800000 iy
2800000 3200000 v
3200000 3900000 ih
3900000 4600000 n
...
```



Obr. 4.5: **Dopredná neurónová sieť s CTC**: Čiarkovaná čiara značí prázdny znak. Môžeme si všimnúť že sieť úplne zlyhala v jeho predpovedi. Najviac výrazný je foném *sil*.

Ako prvú sme skúsili inicializáciu bez núl, označili teda fonémy čisto podľa toho do koľkých rámcov spadajú. Takže súbor vyzeral na rozdiel od pôvodného takto:

```
...
9 19 sil
19 28 iy
28 32 v
32 39 ih
39 46 n
46 65 eh
...
```

Podľa očakávaní táto zmena nemala na trénovanie zásadný vplyv (tabuľka 4.9). Čo znamená, že tento súbor asi nemusí dodržiavať formu tak ako v prípade trénovanie pomocou klasických metód pre neurónové siete.

Ako ďalšie sme chceli vyskúšať čo sa stane, ak sa s týmto súborom trocha pohráme a poposúvame, respektíve skrátime časové ohraničenia foném. Tie sú vypísané v jednoduchom textovom súbore, kde každý foném je ukončený zalomením na nový riadok. Pri skrátení ohraničenia a doplnením prázdneho znaku do týchto medzier bude súbor vždy jednu fonému definovať ako jeden rámec (tabuľka 4.9). Tento experiment nebol úspešný. Počas trénovania sme pri výpočtoch hodnotiacej funkcií dostávali chyby typu *nan*.

```
...
1600000 1700000 w
```

```
1700000  2200000  s_blank
2200000  2300000  er
2300000  3000000  s_blank
3000000  3100000  y
3100000  3500000  s_blank
...
```

Ďalším pokusom bolo predĺženie respektíve skrátenie rámcov tak, aby sme zistili či je dôležité mať presne orámcované fonémy pre trénovanie CTC pomocou nástroja CNTK (tabuľka 4.9). Sieť sa nám natrénovať podarilo, ale podávala horšie výsledky ako v prípade klasického ohraničenia.

```
...
1600000  3200000  w
3200000  3300000  er
3300000  3400000  y
...
```

formátovanie mlf	WER
klasické ohraničenia	28.54 %
ohraničenia bez núl	28.54 %
ohraničené prázdnymi znakmi	fail
poposúvané ohraničenia	29.90 %

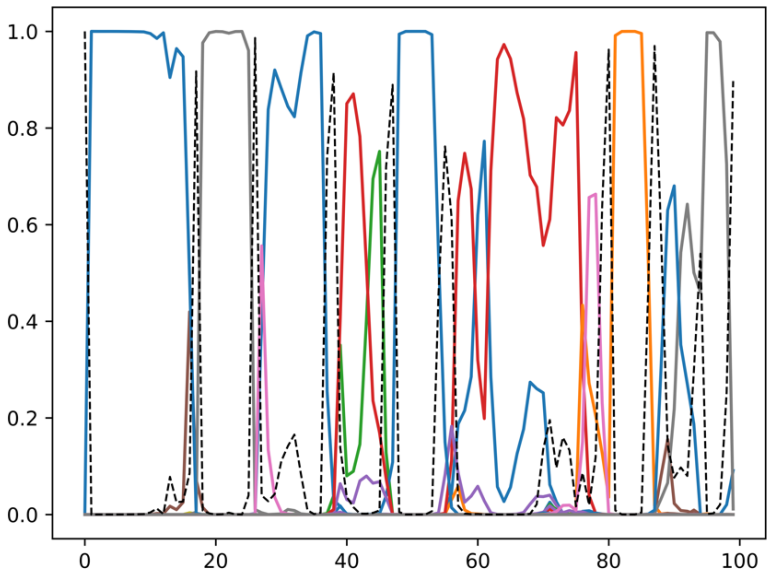
Tabuľka 4.9: Rôzne formáty *mlf*

Inicializácia

Na inicializáciu siete sme v predchádzajúcich experimentoch používali vstavané inicializačné funkcie nástroja CNTK. Skúsili sme preto aj iný prístup inicializácie. Najprv sme si natrénovali sieť pomocou klasických hodnotiacich funkcií *CrossEntropyWithSoftmax* a *ClassificationError*. Naskytol sa tu ale problém. Sieť trénovaná klasickými metódami nepozná prázdny znak a kedže výstup sietí sa musí zhodovať, museli sme ho pridať. Tento problém sme vyriešili tak, že sme mu priradili jeden rámec medzi fonémami. Vyskúšali sme ale aj štyri rámce pre prázdny znak. Po natrénovaní týchto sietí sme ako inicializačnú sieť použili najlepšie natrénovanú sieť s jedným rámcom (obrázok 4.6) a štyrmi rámcami (obrázok 4.7) pre prázdny znak.

Použitím spomínanej inicializácie sme očakávali, že sa natrénované prázdne znaky začnú predĺžovať a neprázdne znaky skracovať. Očakávali sme, že postupnou iteráciou cez epochy sa dostaneme k vhodne natrénovanej sieti, ktorá bude pripomínať graf ako je na obrázku 3.2. Nám ale v obidvoch prípadoch sieť potlačila prázdny znak na takmer najnižšiu pravdepodobnosť (obrázok 4.8).

Skúsili sme inicializovať aj pomocou zle natrénovaných sietí, kde prázdny znak má dĺžku štyroch rámcov (obrázok 4.9). Ale opäť sme sa dostali k podobnému grafu ako v obrázku



Obr. 4.6: **Sieť s jedným rámcom pre prázdny znak**: prázdny znak *(čiarkovaná čierna čiara)* je úzky.

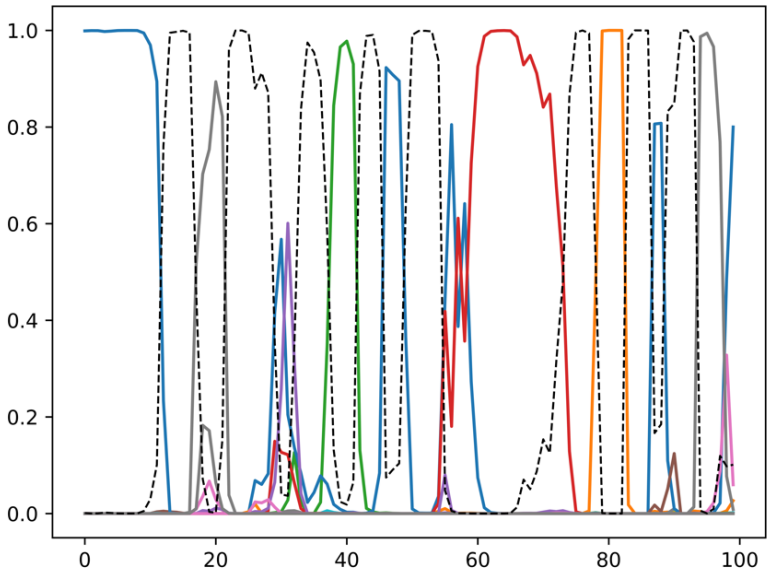
4.8, ktorý potlačil prázdne znaky. Vzali sme aj zle natrénovanú sieť po jednej iterácii (obrázok 4.10), ktorá prvou iteráciou pripomína priebeh prvej iterácie metódy CTC. Ale pri inicializácii touto sieťou, trénovanie siete úplne zlyhalo.

Integrovaná funkcia pre rekurentnú sieť

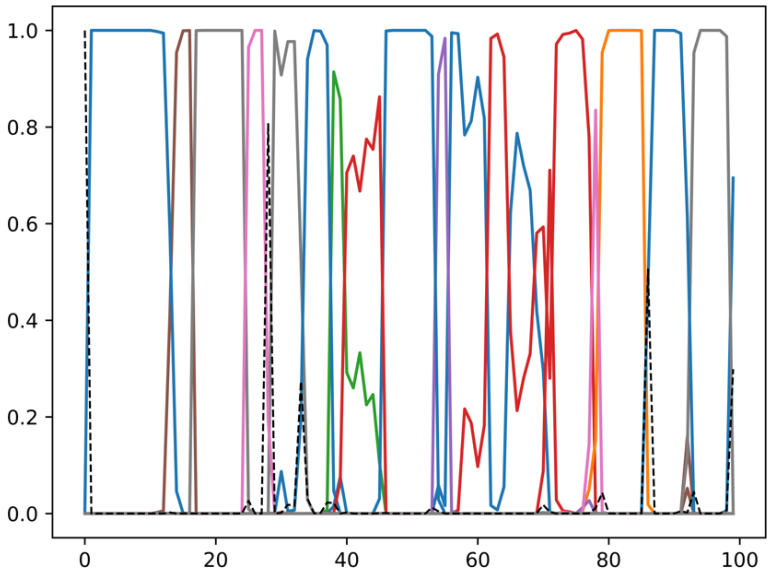
Skúsili sme aj funkciu, ktorá vytvára rekurentnú neurónovú sieť. Tá je priamo integrovaná v CNTK, netreba k nej vytvárať kód a funkcionalitu jednotlivých častí v bunke LSTM. Výhodou tejto funkcie je jednoduchosť použitia, ale najmä možnosť nastavenia smeru siete. Argumenty funkcie *OptimizedRNNStack* sú v nasledovnom poradí: ako prvé sú funkcii predané inicializované váhy, následne vstupný vektor, ďalší je počet prvkov v skrytej vrstve, za ním počet vrstiev, predposlednou je premenná, ktorá určuje jedno/dvoj smernosť siete a posledným je typ prvkov v sieti, v našom prípade volíme prvok LSTM (obrázok 3.1).

```
OptimizedRNNStack (weights, input,
                   hiddenDims, numLayers = 4,
                   bidirectional = true,
                   recurrentOp='lstm')
```

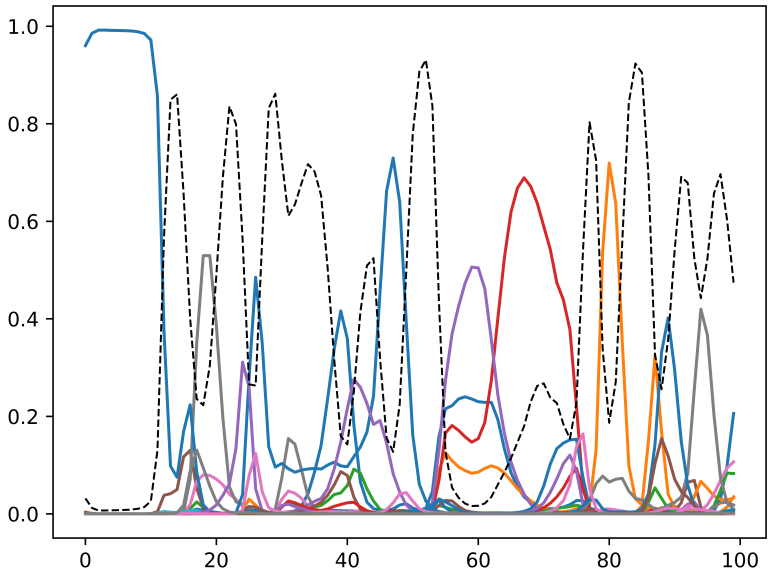
Po rôznych pokusoch konfigurácie parametrov sa nám nepodarilo sieť natrénovať tak aby dávala relevantné výsledky. Dôvodom môže byť zlá inicializácia pri vytváraní vrstiev siete alebo dokonca nekompatibilita funkcií pre CTC s danou funkciou.



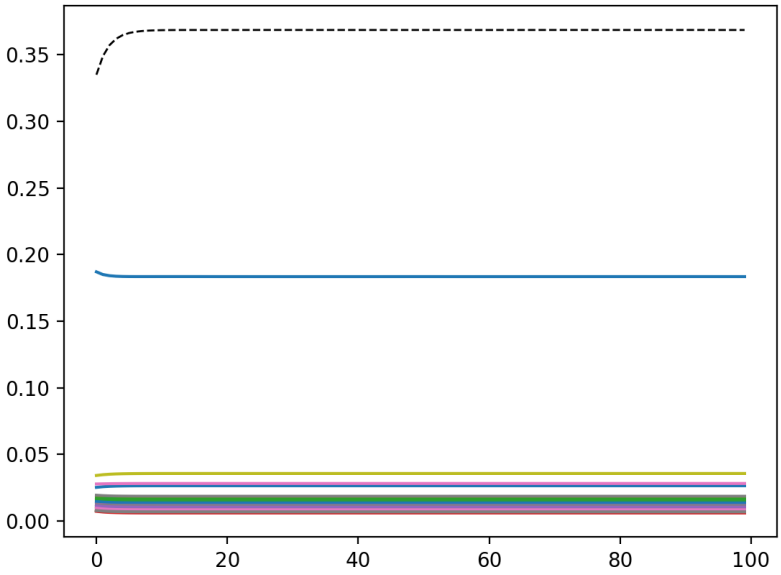
Obr. 4.7: **Sieť so štyrmi rámcami pre prázdny znak**: prázdny znak *(čiarkovaná čierna čiara)* je široký.



Obr. 4.8: **Sieť inicializovaná modelom natrénovaným klasickými metódami**: prázdny znak *(čiarkovaná čierna čiara)* takmer vymizol a neprázdne znaky teda fonémy *farebné čiary)* sa rozšírili.



Obr. 4.9: **Zle natrénovaná sieť pomocou klasických metód po niekoľkých iteráciach**: prázdny znak *(čiarkovaná čierna čiara)* je najpravdepodobnejší a neprázdne znaky teda fonémy *(farebné čiary)* sú potlačené.



Obr. 4.10: **Zle natrénovaná sieť pomocou klasických metód po jednej iterácii**: prázdny znak *(čiarkovaná čierna čiara)* je najpravdepodobnejší a neprázdne znaky teda fonémy *(farebné čiary)* sú potlačené.

Kapitola 5

Záver

V práci sme sa dozvedeli detaily ohľadom trénovania neurónových sietí metódou CTC v prostredí nástroja CNTK. Tento nástroj poskytuje väčšiu flexibilitu inicializácii ako nástroj EESEN. Na druhú stranu sa nám ale nepodarilo prekonať úspešnosť nástroja EESEN oproti nástroju CNTK. Pri inicializácii sme skúsili rôzne nastavenia, či už nastavovanie rôznych vstupných dát, alebo nastavenia rôznych parametrov pri učení. Nedostatočnú dokumentáciu jednotlivých argumentov funkcií sme otestovali experimentovaním. Z toho sme zistili, že dáta nemusia mať presnú časovú segmentáciu, sieť sa dokázala aj napriek nesprávnemu segmentovaniu naučiť celkom relevantným výsledkom. Pre rýchlejšie natrénovanie treba zvoliť vyššiu *miniBatch* konštantu. Konštantu premennej meškania *delayConstraint* treba zvoliť skôr v hodnote 5, alebo ju po prípade vypnúť. Vyššia hodnota konštanty výsledky len zhoršuje. Skúsili sme aj inicializáciu váhových hodnôt siete iným spôsobom ako použitím vstavaných funkcií. Natrénovali sme sieť najprv klasickou metódou a potom sme túto sieť použili ako inicializačnú pre natrénovanie siete metódou CTC. Sieť sa nám síce natrénovať podarilo ale výsledný model nebol ekvivaletný s modelmi natrenovanými metódou CTC. Týmto experimentom sme došli k záveru, že sieť je citlivá na počiatočnú inicializáciu. Vstavaná funkcia na tvorbu rekurentných sietí nám vôbec nedávala použiteľné výsledky. Algoritmy v rámci nástroja CNTK sa stále vyvíjajú. Je možné že sa algoritmy zrýchlia a možno budú podávať aj lepšie výsledky.

Literatúra

[1] Gradient. https://www.analyticsvidhya.com/blog/2017/03/introduction-to-gradient-descent-algorithm-along-its-variants/, accessed: 2017-04-30.

[2] Bengio, Y.; Simard, P.; Frasconi, P.: Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, ročník 5, č. 2, 1994: s. 157–166.
URL http://www.iro.umontreal.ca/~lisa/pointeurs/ieeetrnn94.pdf

[3] Graves, A.; Fernández, S.; Gomez, F.: Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *In Proceedings of the International Conference on Machine Learning, ICML 2006*, 2006, s. 369–376.

[4] Hochreiter, S.; Schmidhuber, J.: Long short-term memory. *Neural computation*, ročník 9, č. 8, 1997: s. 1735–1780.

[5] Huang, X.; Acero, A.; Hon, H.: *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall PTR, 2001, ISBN 9780130226167.
URL https://books.google.cz/books?id=reZQAAAAMAAJ

[6] Huang, X.; Deng, L.: *An Overview of Modern Speech Recognition*. Chapman & Hall/CRC, January 2010, s. 339–366.
URL https://www.microsoft.com/en-us/research/publication/an-overview-of-modern-speech-recognition/

[7] Lee, K. F.; Hon, H. W.: Speaker-independent phone recognition using hidden Markov models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ročník 37, č. 11, Nov 1989: s. 1641–1648, ISSN 0096-3518, doi:10.1109/29.46546.

[8] Miao, Y.; Gowayyed, M.; Metze, F.: EESEN: End-to-End Speech Recognition using Deep RNN Models and WFST-based Decoding. *CoRR*, ročník abs/1507.08240, 2015.
URL http://arxiv.org/abs/1507.08240

[9] Mohri, M.; Pereira, F.; Riley, M.: Weighted Finite-State Transducers in Speech Recognition. 2001.

[10] Nasr, G. E.; Badr, E. A.; Joun, C.: Cross Entropy Error Function in Neural Networks: Forecasting Gasoline Demand. In *FLAIRS Conference*, editace S. M. Haller; G. Simmons, AAAI Press, 2002, ISBN 1-57735-141-X, s. 381–384.
URL http://dblp.uni-trier.de/db/conf/flairs/flairs2002.html#NasrBJ02

[11] Povey, D.; Ghoshal, A.; Boulianne, G.; aj.: The Kaldi Speech Recognition Toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, IEEE

Signal Processing Society, Prosinec 2011, ISBN 978-1-4673-0366-8, iEEE Catalog No.: CFP11SRW-USB.

[12] Pundak, G.; Sainath, T.: Lower Frame Rate Neural Network Acoustic Models. In *Interspeech*, 2016.

[13] Schuster, M.; Paliwal, K.: Bidirectional Recurrent Neural Networks. *Trans. Sig. Proc.*, ročník 45, č. 11, Listopad 1997: s. 2673–2681, ISSN 1053-587X, doi:10.1109/78.650093.
URL http://dx.doi.org/10.1109/78.650093

[14] Young, S. J.; Kershaw, D.; Odell, J.; aj.: *The HTK Book Version 3.4*. Cambridge University Press, 2006.

Prílohy

Príloha A

Obsah priloženého DVD

DVD obsahuje:

- konfiguračné súbory pre CNTK
- skripty pre EESEN
- pomocné skripty na prípravu dát

Obsah zhodný s obsahom na repozitáry:
https://git.fit.vutbr.cz/karafiat/CNTK_CTC_Gajdar2016