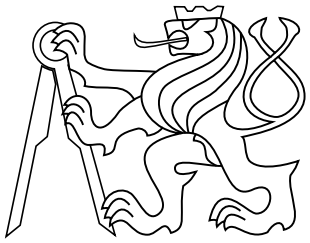
bakalářská práce

Implementace fonémového rozpoznávače s nástroji TNet Toolkitu

Jiří Fiala



květen 2014

vedoucí práce: Doc. Ing. Petr Pollák, CSc.

České vysoké učení technické v Praze
Fakulta elektrotechnická, Katedra kybernetiky

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Jiří F i a l a

Studijní program: Kybernetika a robotika (bakalářský)

Obor: Robotika

Název tématu: Implementace fonémového rozpoznávače s nástroji TNet Toolkitu

Pokyny pro vypracování:

1. Seznamte se s problematikou extrakce řečových příznaků na bázi neuronových sítí s užším zaměřením na následnou realizaci fonémového rozpoznávače.
2. Implementaci MLP sítě realizujte pomocí volně dostupné sady nástrojů z TNet Toolkitu a zaměřte se na možnost paralelizace velkého množství výpočtů při trénování MLP sítě.
3. Přesnost klasifikace MLP sítě analyzujte na úloze rozpoznávání fonémů a v rámci experimentů nalezněte optimální řečové příznaky použitelné v této úloze.

Seznam odborné literatury:

[1] Schwarz, P.; Matejka, P.; Cernocky, J.: Hierarchical Structures of Neural Networks for Phoneme Recognition. In Proc. of ICASSP 2006, vol.1, no., pp.I,I, 14-19, 2006.
[2] Psutka, J.; Müller, L.; Matoušek, J.; Radová, V.: Mluvíme s počítačem česky.Academia, 2006.
[3] Uhlíř, J. a kol.: Technologie hlasových komunikací. Nakladatelství ČVUT, Praha, 2007.
[4] Huang, X.; Acero, A.; Hon, H.V.: Spoken Language Processing. Prentice Hall, 2001.
[5] Fousek, P.: Extraction of Features for Automatic Recognition of Speech Based on Spectral Dynamics. PhD Thesis, ČVUT FEL, Praha, 2007.
[6] Neural Network Trainer TNet. WEB-page http://speech.fit.vutbr.cz/cs/software/neural-network-trainer-tnet , 2013.

Vedoucí bakalářské práce: doc. Ing. Petr Pollák, CSc.

Platnost zadání: do konce letního semestru 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 1. 2014

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce Doc. Ing. Petru Pollákovi, CSc. za uvedení do problematiky rozpoznávání řečového signálu, za výborné vedení a jeho ochotu a trpělivost při objasňování jednotlivých postupů. Dále bych rád poděkoval všem členům Laboratoře zpracování řečového signálu za jejich výpomoc a konzultace.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne _____ _____

podpis

Abstrakt

Tato bakalářská práce se zabývá aplikací umělých neuronových sítí typu vícevrstvý perceptron (MLP) v úloze rozpoznávání fonémů v řečovém signálu. Implementace těchto sítí byla provedena prostřednictvím balíčku softwarových nástrojů s označením TNet, který umožňuje inicializaci sítě, trénovaní, testovaní a další funkce. Hlavním cílem práce bylo poskytnout informace pro používání tohoto nástroje a demonstrovat jeho schopnosti při paralelním trénování. V konkrétní aplikaci se vstupní vektory pro MLP skládaly z příznaků na bázi dlouhých časových trajektoroií (TRAPs), výstupy sítě odpovídaly aposteriorní pravděpodobnosti přiřazení do jedné z 45 tříd fonémů. Pro zlepšení výsledků klasifikace byly použity skryté Markovovy modely. Předpokládané vlastnosti MLP klasifikátoru potvrdila řada experimentů, provedených se signály z databáze SPEECON. Úspěšnost klasifikace a doba trénování byly analyzovány na základě různých parametrů sítě a v nejlepším případě bylo dosaženo úspěšnosti klasifikace 82 % na úrovni fonémů. Dobu trénování výrazně zkrátila jeho paralelizace, a to přibližně o 50 %. Nejdelší čas trénování pro nejrozsáhlejší síť tvořenou 12 000 skrytými neurony trénovanou na 40000 signálech nepřekročil 13 h.

Klíčová slova

rozpoznávání řeči; fonémový rozpoznávač; vícevrstvý perceptron; umělé neuronové sítě; skryté Markovovy modely; HMM; MLP; TRAP

Abstract

This bachelor thesis deals with an application of artificial neural networks (ANN), specifically multilayer perceptron (MLP), in the area of speech recognition. Used ANN was implemented using software toolkit TNet, which includes the tools for a work with MLPs at various levels such as initialization, training, testing, etc.The main goal of this thesis is to provide instructions and information for the usage of this toolkit same as to demonstrate its capabilities for parallel training. In our application MLP input vectors consisted of long temporal features (TRAPs) and phoneme posterior probabilities were represented by the output of the net. At the end, the Hidden Markov Models (HMM) were used for the output classification enhancement. Theoretical properties of MLP approved series of experiments with speech signals from SPEECON database. Classification accuracy and training time rely on several ANN parameters and these dependencies were also analyzed. The best obtained classification accuracy was 82 % at the level of phoneme classification. Parallel training significantly reduced the training time approximately about 50 %. For the most complex neural network with 12 000 hidden neurons and training set of size 40 000 signals, the training process lasted less then 13 hours.

Keywords

speech recognition; phoneme recognizer; multilayer perceptron; artificial neural networks; Hidden Markov Models, HMM; MLP; TRAP

Obsah

Seznam obrázků

Použité zkratky

TRAPs	TempoRAl Patterns (časové trajektorie)
MLP	MultiLayer Perceptron (vícevrstvý perceptron)
MLF	Master Label File
SCP	SCriPt File
DCT	Discrete Cosine Transformation (diskrétní kosinová transformace)
ANN	Artificial Neural Net (umělá neuronová síť)
SGE	Sun Grid Engie
BPG	Back PropaGation (zpětné šíření)
HMM	Hidden Markov Model (skrytý Markovovův model)

1 Úvod

Rozpoznávání mluvené řeči a schopnost její interpretace v podobě digitálních dat umožňuje nejpřirozenější způsob interakce člověka s počítačem či výpočetními zařízeními obecně. Schopnost zpracování řeči nalézá své uplatnění v mnoha aplikacích. Umožňuje ovládat dané zařízení v případech, kdy jiný způsob ovládání je nevhodný, například používání mobilního telefonu v automobilu, nebo pro handicapovaného člověka dokonce není ze zdravotních důvodů možný. Řeč rovněž obsahuje jedinečné prvky charakteristické dané osobě, což umožňuje tuto osobu na základě její promluvy identifikovat. Dále je možné rozpoznávat vady řeči spojené s různými chorobami či duševní rozpoložení člověka.

Jádrem složitějších systémů zpracování řeči je často fonémový rozpoznávač, který umožňuje právě interpretaci mluvené řeči v podobě vhodné k dalšímu strojovému využití. Fonémový rozpoznávač ve své podstatě provádí segmentaci promluvy na jednotlivé základní jednotky - hlásky. Do fonémového rozpoznávače vstupuje zaznamenaný signál obsahující promluvu. Na výstupu je informace, kde se v jaký časový okamžik nachází jaký foném.

Využití samostatného signálu v časové oblasti je pro proces rozpoznávaní nevhodné a jeho přílišná variabilita by zhoršovala úspěšnost rozpoznávání. Signál je nejprve nutné předzpracovat a použít vhodné parametrizace k výpočtu příznaků, na jejichž základě daný klasifikátor přiřadí zpracovávané části signálu daný výstup, v tomto případě foném. Pro výpočet příznaků se v posledních letech dostala do popředí metoda založená na delších časových trajektoriích nazývaná TRAP (TempoRAl Patterns). V aplikacích zpracování řeči se využívá několika přístupů ke klasifikaci příznaků. Jejich výběr a použití závisí na zvolené aplikaci. Patří mezi ně například gaussovské modely (GMM - Gaussian Mixture Models), Markovovy řetězce nebo Support Vector Machine. V aplikacích rozpoznávání fonémů nalézají uplatnění umělé neuronové sítě (ANN - Artificial Neural Networks). V případě ANN klasifikátorů se jako nejvhodnější jeví vícevrstvý perceptron (MLP - Multi Layer Perceptron), provádějící nelineární mapování vstupu na výstup. Přístup klasifikace příznaků v implementovaném rozpoznávači je založen rovněž na MLP síti.

S použitím neuronových sítí jako klasifikátoru je spjata řada procesů, předcházející vlastnímu používání sítě. Kompletní řešení práce s MLP sítěmi umožňují různé softwa-

rové nástroje balíčku TNet. Cílem této práce je rovněž osvojit si používání jednotlivých nástrojů a poskytnout potřebné informace pro práci s tímto balíčkem. Hlavní výhoda používání těchto nástrojů spočívá v možnosti paralelního trénovaní MLP sítí, což značně ovlivňuje dobu potřebnou k natrénování sítě, zejména pak při práci s velkým množstvím trénovacích dat.

V první části této práce je popsán proces rozpoznávání řeči, tedy zpracování řečového signálu, zvolená metoda parametrizace a použitý klasifikátor. Ve druhé části je podrobněji popsána vlastní implementace fonémového rozpoznávače s důrazem na používání jednotlivých nástrojů balíčku TNet. Ve třetí části jsou prezentovány jednotlivé experimenty a dosažené výsledky. Celkové shrnutí práce je uvedeno v závěru.

2 Princip rozpoznávání fonémů

K popisu činnosti fonémového rozpoznávače je nejprve nutné objasnit význam pojmu, se kterým rozpoznávač pracuje, tedy samotného fonému. Foném je pojem zavedený ve fonologii, která zkoumá povahu řečových zvuků z hlediska postavení, funkce a vztahů mezi zvuky v rámci celého jazykového systému [1].Je to základní abstraktní lingvistická jednotka, schopná rozlišovat významové jednotky.

Fonémový rozpoznávač ve své podstatě provádí fonetickou transkripci mluvené řeči, tedy proces, kdy je zaznamenaný signál převeden na sled fonetických elementů [2]. Rozpoznávání fonémů v mluvené řeči je proces skládající se z několika kroků, které budou podrobně popsány.

Dalším souvisejícím pojmem je hláska, pro niž se z anglického jazyka používá termín fón. Mezi hláskou a fonémem je ovšem nutné rozlišovat. Hláska představuje konkrétní zvukovou realizaci vysloveného fonému [2]. Rozdíl mezi těmito pojmy může být demonstrován na slovech hrabě a hrábě. Jelikož změna hlásky [a] za [á] změní význam slova, představují tyto dvě hlásky dva odlišné fonémy. Velikost inventáře fonémů je odlišná pro různé jazyky. Například [ŋ] a [n] jsou pro anglický jazyk dva různé fonémy, ale v českém jazyce jejich záměna význam slova nemění, proto jsou jedním fonémem.

Prvním krokem v procesu rozpoznávání je parametrizace zaznamenaného řečového signálu. Z vlastností řeči vyplývá, že jeho kontinuální záznam v časové oblasti je pro účely rozpoznávání nevhodný a je nutné provést segmentaci na krátkodobé úseky. Po těchto úpravách se přistoupí k výpočtu vhodných parametrů, příznaků, na jejichž základě se provádí samotné rozpoznávání. Vhodná parametrizace signálu má klíčový vliv na úspěšnost rozpoznávání.

Dalším krokem je klasifikace příznaků. Vypočítané příznaky slouží jako vstup klasifikátoru, který provede přiřazení požadovaného výstupu. Klasifikátorem je v případě konkrétní realizace fonémového rozpoznávače neuronová síť, provádějící transformaci příslušného počtu vstupních příznaků do počtu tříd, odpovídající počtu fonémů použité abecedy. Schématické znázornění průběhu rozpoznávání fonémů zobrazuje ilustrace 2.0.1.

Obrázek 2.0.1 Průběh rozpoznávání fonémů.

2.1 Výpočet příznaků řečového signálu

Následující metody zpracování a úprav řečového signálu uvažují zdigitalizovaný průběh a pracují proto s jednotlivými vzorky signálu.

Úprava a zpracování řečového signálu probíhá jak v časové, tak ve frekvenční oblasti. Tyto oblasti spolu navzájem souvisejí a je tedy nutné provádět takové úpravy, které se neprojeví významnou ztrátou nebo naopak vznikem redundantní informace při přechodu mezi jednotlivými oblastmi zpracování. V časové oblasti se provádí segmentace signálu a váhování těchto segmentů Hammingovým oknem. Frekvenční oblast je pak využita k výpočtu příznaků pro klasifikaci. Důvody použití jednotlivých metod zpracování řečového signálu jsou popsány v následujících kapitolách.

Jednotlivé vzorky signálu, tedy resp. jeho časový průběh je pro vstup do klasifikátoru nevhodný. Časový průběh je velmi dynamický, citlivý na změnu. Odlišný průběh dostaneme vždy pro stejnou hlásku při opakovaném záznamu od stejného mluvčího. Odráží v sobě mnoho aspektů, jako je rychlost promluvy, šum prostředí a další. Pro účely klasifikace je nutné zajistit robustnější vstup klasifikátoru a zejména pak takový, který bude zbaven redundantní informace.

Právě takovou úpravou signálu představuje parametrizace. Na jejím základě jsou z daného signálu za použití zvolených metod extrahovány parametry signálu, označované jako příznaky. Tyto příznaky poté představují jednotlivé složky vstupního vektoru klasifikátoru a určují tak jeho dimenzi. Parametrizace hraje důležitou roli z hlediska počtu příznaků a dimenzionality vstupního vektoru. Příliš velký počet příznaků vede k jevu označovaného jako prokletí dimenzionality (Curse of Dimensionality), kdy se zvyšujícím se počtem dimenze vstupního vektoru roste exponenciálně počet dat nutných k namapování tohoto vstupního prostoru. Více o tomto jevu uvádí například [3].

2.1.1 Segmentace a váhování řečového signálu

Řeč vzniká vlivem rezonance základního hlasivkového tónu v dutinách řečového ústrojí. Změnou tvaru těchto dutin dochází ke změně rezonančních frekvencí (formantů) a tím pádem ke změně výsledného tónu, resp. hlásky. Hlasový trakt není schopen měnit své vlastnosti, tedy ovlivňovat jednotlivé formanty, nekonečnou rychlostí. Řečový signál je označován jako kvazistacionární v krátkých časových úsecích. Délka těchto úseků se pohybuje v rozmezí 10 - 30 ms. Právě krátkodobá stacionarita řeči umožňuje samotné

rozpoznávání, které probíhá v těchto krátkých úsecích.

Prvním krokem v procesu rozpoznávání při zpracování řečového signálu pro účely rozpoznávání je jeho segmentace. Jednotlivé krátkodobé segmenty jsou nazývány okna, rámce (frames). Nejčastěji používaná délka okna je 25 ms. Segmentace signálu, tedy zpracování signálu konečné délky, způsobuje prosakování ve spektru a jelikož další analýza probíhá ve frekvenční oblasti, je nutné tento jev ošetřit.

Prosakovaní ve spektru znamená, že vlivem neperiodicity časového průběhu, ze kterého je spektrum počítáno, dochází v jeho spektru ke vzniku vyšších harmonických složek, což vyplývá z principu Fourierovy transformace při aplikaci na signál konečné délky, a proto je prosakování ve spektru nežádoucí jev, který je nutné potlačit. Způsobem, jak zabránit prosakování ve spektru je váhování. Při váhování je okno signálu násobeno (váhováno) funkcí příslušného okna. Nejčastěji je používáno Hammingovo okno. Tato funkce je dána matematickým předpisem 2.1.1.

$$w(n) = \begin{cases} 0,54 - 0,46 \cos\left(\dfrac{2\pi n}{L-1}\right) & \text{pro } 0 \leq \text{n} \leq \text{L - 1} \\ 0 & \text{pro další n} \end{cases} \tag{2.1.1}$$

L je délka okna a n je daný vzorek signálu. Jak ukazuje obrázek 2.1.1, pokud je segment signálu, který obsahuje hlásku /a/ váhován Hammingovým oknem stejné délky, je v jeho spektru mnohem lépe patrná formantová struktura. Z průběhu této váhovací funkce je zřejmé, že dochází k potlačování krajních vzorků vybraného segmentu. Aby se omezila ztráta informace, při segmentaci řečového signálu se používá překrývání dílčích oken. V praxi se osvědčila délka okna 25 ms s krokem segmentace 10 ms. Obrázek 2.1.2 ilustruje proces segmentace a váhování řečového signálu.

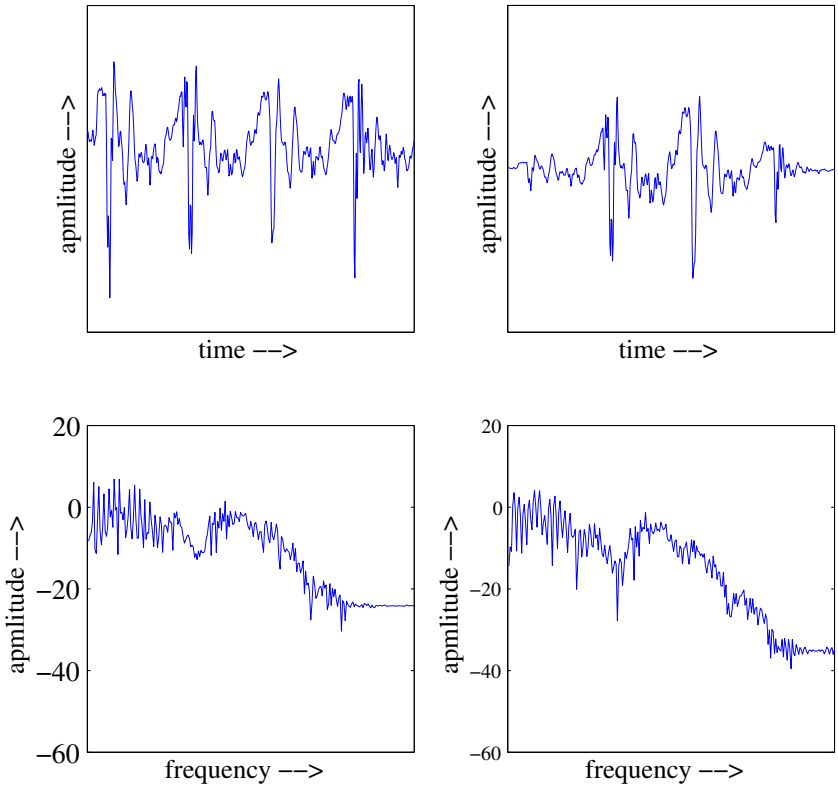
2.1.2 Banka filtrů

Vzhledem ke zvolené parametrizaci signálu je nutné provést pásmovou filtraci za použití banky filtrů. Pásmová filtrace je aplikována ve frekvenční oblasti a její význam spočívá zejména v redukci nadbytečné informace snížením dimenze spektra. Při pásmové filtraci dochází také k vyhlazení spektrální obálky, jejíž vývoj v čase je základem pro rozpoznávání.

Lidské ucho nevnímá zvuk lineárně, ale v logaritmické míře. Je tedy zřejmé, že pokud provedeme obdobnou úpravu řečového signálu, jakou provádí naše sluchové orgány, můžeme tím přispět k lepší účinnosti rozpoznávání řeči. Navržená banka filtrů používá tedy nelineární melovskou škálu, určenou vztahem 2.1.2.

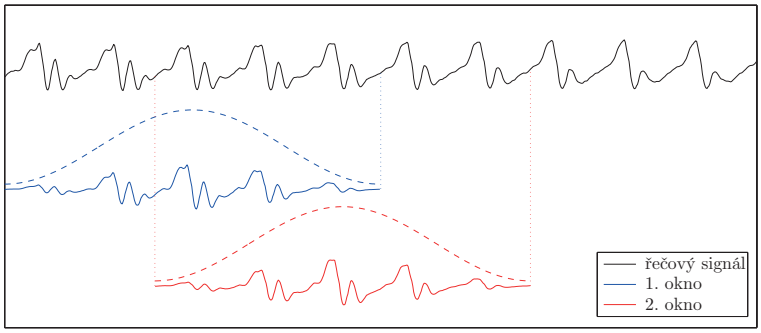
$$f_m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \quad \text{[mel]} \tag{2.1.2}$$

f je frekvence v lineárním měřítku, jejíž jednotkou je [Hz]. Jednotlivé filtry v melovské škále mají tvar rovnoramenných trojúhelníku a pokrývají rovnoměrně celé frekvenční



Obrázek 2.1.1 Potlačení prosakování ve spektru

.

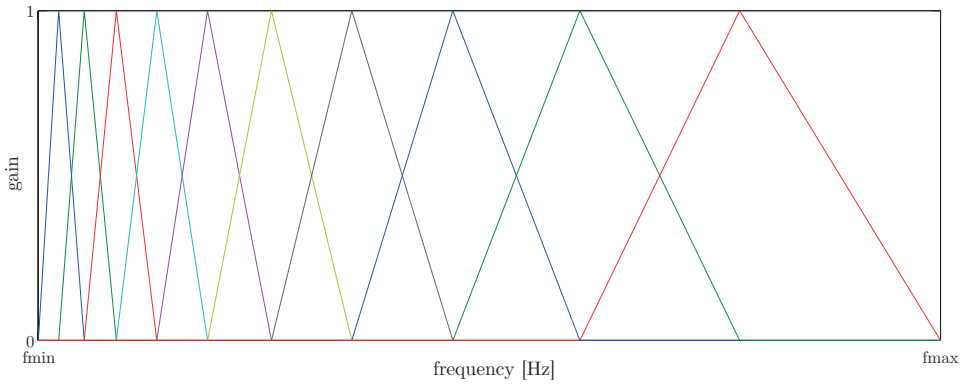


Obrázek 2.1.2 Segmentace a váhování řečového signálu.

pásmo až do poloviny vzorkovací frekvence.

Při analýze řeči není nutné používat celé kmitočtové spektrum, příliš nízké nebo naopak vysoké frekvence obsahují zbytečnou informaci. Kmitočtové pásmo, které banka filtrů pokrývá, je tedy omezeno minimální a maximální frekvencí f_{min}, resp. f_{max}. Počet pásem banky filtrů je volen s ohledem na vzorkovací frekvenci signálu a zvolenou šířku kmitočtového pásma. Obvyklý počet pásem je 23.

Postup aplikace banky filtrů je následující. Pro každý jednotlivý segment signálu, jež byl popsán výše, se vypočítá kmitočtové spektrum za použití algoritmu rychlé Fourie-

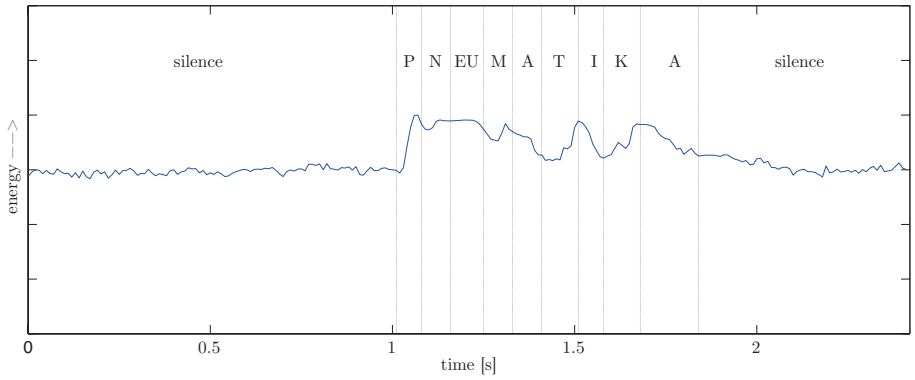


Obrázek 2.1.3 Banka filtrů

rovy Transformace (FFT). Používá se buď amplitudové nebo výkonové spektrum. Každý koeficient tohoto spektra je poté násoben ziskem jednotlivého filtru a výsledky jsou sčítány. Pro každé pásmo je tedy výsledkem jedna číselná hodnota. Aby nemusely být jednotlivé koeficienty FFT přepočítány do melovské škály, používá se přepočtené banky filtrů do měřítka v Hz. Podoba této banky je znázorněna na obrázku 2.1.3.

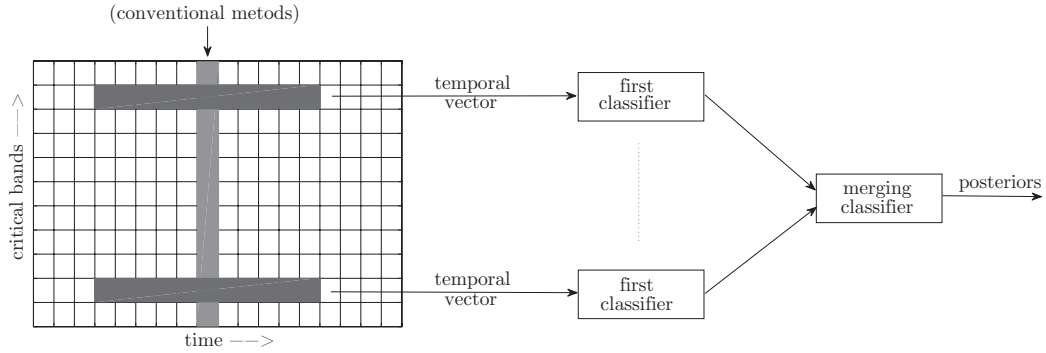
2.1.3 Časové trajektorie

Výstup z banky filtrů je nyní využit k výpočtu příznaků. Běžné techniky používají jako příznakový vektor na vstupu klasifikátoru výstup z kritických pásem banky filtrů pro délku jednoho okna. Se zvýšením výkonu výpočetních zařízení v posledních letech se do popředí dostává technika pracující s vývojem trajektorie spektrální energie v čase a jako vstup klasifikátoru je použit vektor odpovídající této delší trajektorii. Tato metoda se nazývá TRAPs (z anglického TempoRAl Patterns).



Obrázek 2.1.4 Průběh energie ve 4. kritickém pásmu.

Právě vývoj spektrální energie v čase v sobě nese informaci o typu dané hlásky. Obrázek 2.1.4 ukazuje průběh energie spektra ve 4. kritickém pásmu pro slovo "pneumatika". Je možné si všimnout jisté podobnosti průběhu energie pro opakující se hlásku `a`. Delší časové trajektorie v sobě obsahují informaci o okolí fonému a umožňují řešit problém s jejich splýváním při artikulaci.



Obrázek 2.1.5 Princip metody TRAP.

Ilustrace 2.1.5 ukazuje princip výpočtu TRAPs. Základem vektoru příznaků je centrální okno. Vzhledem k centrálnímu oknu je zvolen stejný počet hodnot směrem do minulosti a do budoucnosti. První experimenty používaly délku trajektorie 1 s, ovšem jako osvědčená hodnota ze současných aplikací [13] se jeví délka trajektorie odpovídající 310 ms, což při 10 ms posuvu oken odpovídá 31 segmentům. Výstup z každého pásma je poté použit jako vstup klasifikátorů první úrovně. Výstupy z klasifikátorů pro každé pásmo jsou poté spojeny pomocí dalšího klasifikátoru, jehož výstup odpovídá aposteriorním pravděpodobnostem přiřazení do dané třídy pro centrální rámec.

Možností, jak zvýšit účinnost rozpoznávání je použití jako vstup do pásmového klasifikátoru vektory z více sousedních pásem. Postupem času vzniklo několik modifikací této metody, některé z nich jsou popsány dále.

Zjednodušená varianta

Výše popsaná metoda je základním způsobem výpočtu příznaků náročným na výpočetní výkon. Při použití 23 pásem banky filtrů a délky trajektorie 31 segmentů je dimenze příznakového vektoru 713. S ohledem na zrychlení a efektivnost byla v [14] popsaná zjednodušená varianta základní TRAPs. Podstatou této metody je DCT transformace, umožňující snížit počet příznaků zanedbáním vyšších složek, které nejsou výraznými nositeli informace, což vede ke kompresi informace. Každý vektor, odpovídající jednotlivým pásmům, je váhován okenní funkcí, aby byly hraniční hodnoty potlačeny. Na tento vektor je aplikována DCT transformace, ze které je použito několik prvních hodnot. Osvědčeným počtem složek DCT je 16. Tyto hodnoty jsou poté spojeny do jednoho

vektoru příznaků, který slouží jako vstup jediné neuronové sítě. Pro obvyklé hodnoty 23 pásem banky filtrů, kde je pro každé pásmo vypočítáno 16 hodnot DCT, dostaneme 368-mi rozměrný vstupní vektor.

Rozdělení časového kontextu

Další variantout je rozdělení časového vektoru na levou a pravou část. Na každou z těchto částí je aplikována váhovací funkce a diskrétní kosinova transformace. Oběma částem přísluší vlastní neuronová sít. Výstupy z těchto klasifikátorů slouží jako vstup další umělé neuronové síti.

2.2 Klasifikace do fonetických tříd

Vypočítané příznaky jsou dále klasifikovány do příslušných počtů tříd. K tomu účelu se používá umělá neuronová síť. V aplikacích rozpoznávaní řeči se osvědčila síť typu vícevrstvý perceptron (MLP - Multi Layer Perceptron), a to z důvodu poměrně snadné implementace a možnosti změny svých rozměrů, které mají zásadní vliv na úspěšnost klasifikace.

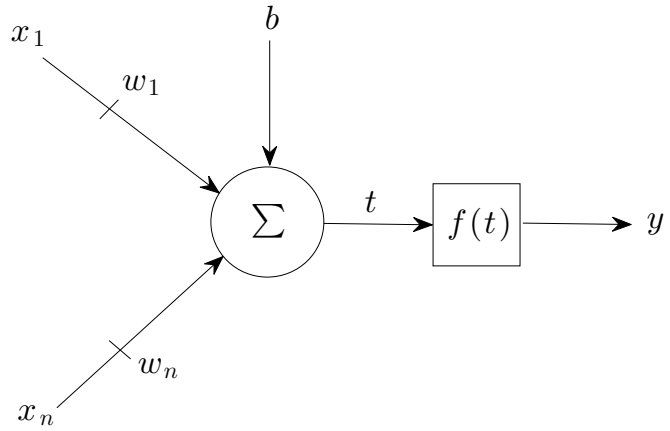
2.2.1 Umělé neuronové sítě typu MLP

Umělé neuronové sítě (anglicky Artificial Neural Networks - ANN) jsou nástrojem umělé inteligence, který provádí mapování z n rozměrného vstupního vektoru na m rozměrný vektor výstupní. Tato vlastnost umožňuje používání ANN jako účinných klasifikátorů. Princip činnosti ANN je založen na fyzikální podstatě fungování biologických neuronových sítí. Základní jednotkou ANN je tedy umělý neuron, znázorněný na obrázku 2.2.1, matematický model biologického neuronu.

Každý umělý neuron má několik vstupů a pouze jediný výstup. Každá vstupní hodnota má přiřazenu prioritu prostřednictvím synaptické váhy. Tělo umělého neuronu se skládá z obvodové a aktivační (přenosové funkce). Obvodová funkce určuje, jak budou vstupní hodnoty kombinovány uvnitř neuronu, zatímco aktivační funkce udává, jakým způsobem bude výsledek obvodové funkce přenesen na výstup [18].

Obvodová funkce má nejčastěji tvar lineární kombinace vážených vstupních hodnot. Označme vektor vstupních hodnot $\mathbf{x}$, jemu přísluší vektor synaptických vah $\mathbf{w}$. Potom výstup obvodové funkce, označovaný jako potenciál, je dán vztahem 2.2.1.

$$t = \sum_{i=1}^{N} x_i w_i + b \tag{2.2.1}$$



Obrázek 2.2.1 Matematický model neuronu.

kde b je prahová hodnota daného neuronu, označována také jako bias nebo threshold. Tato hodnota určuje excitaci neuronu, tedy kdy je neuron aktivní. Pokud je vážená suma větší než tento práh, je neuron aktivován. Často se práh označuje jako synaptická váha s indexem 0. V tomto případě má vstupní hodnota odpovídající této váze hodnotu 1 nebo -1 a potenciál má tvar 2.2.2.

$$t = \sum_{i=0}^{N} x_i w_i \tag{2.2.2}$$

Výstup umělého neuronu je vypočítán podle vztahu 2.2.3. Funkce f označuje přenosovou funkci.

$$y = f(t) \tag{2.2.3}$$

Přenosových funkcí existuje celé řada a je nutné je zvolit na základě požadované aplikace. Proces učení ANN znamená nastavování takových hodnot synaptických vah, aby bylo dosaženo požadovaného výstupu. Síť tak prostřednictvím synaptických vah získá schopnost uložení informace a její následné vybavení. Fáze nastavování vah se nazývá učení (trénink). Způsobů a algoritmů učení existuje rovněž celá řada a jejich volba závisí na konkrétním typu dané sítě.

Spojováním výše popsaných neuronů vznikají ANN. Jako perceptron je označována síť takových neuronů, které jako svoje přenosové funkce používají saturační skokové funkce, například 2.2.4.

$$f(t) = \begin{cases} 1, & \text{pokud } t \geq 0 \\ -1 \text{ nebo } 0, & \text{pokud t} < 0 \end{cases} \tag{2.2.4}$$

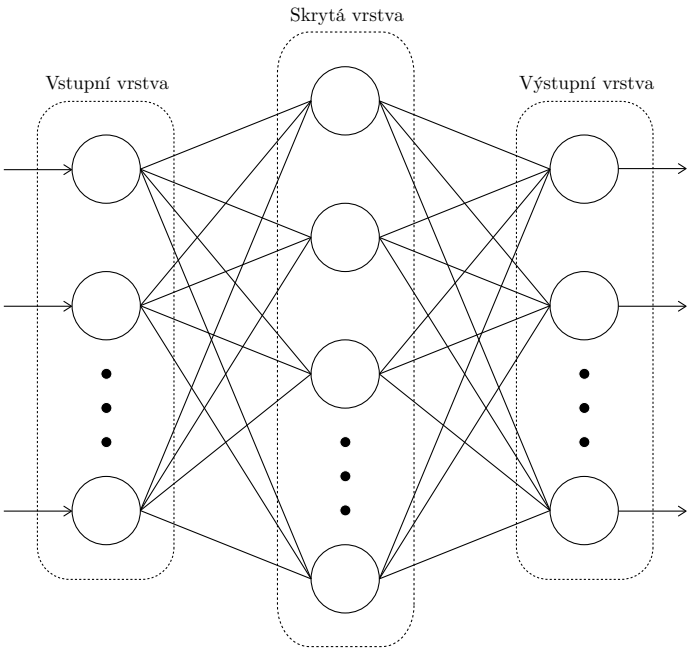
Tuto funkci ovšem není možné z důvodu své nespojitosti derivovat a použít tak algoritmus učení vícevrstvých sítí. Proto je perceptron jednovrstvá síť. Z tohoto faktu plyne

základní omezení, a to schopnost klasifikovat pouze lineárně separabilní vstupní datový prostor. To znamená, že vstupní data lze rozdělit do jednotlivých tříd přímkou. Toto základní omezení znemožňuje použití perceptronu ve většině aplikacích.

Pokud je skoková přenosová funkce nahrazena spojitou nelineární funkcí, je možné vytvořit vícevrstvou síť, pro kterou výše zmíněné omezení neplatí. Tuto funkci nejčastěji představuje sigmoida 2.2.5

$$f(t) = \frac{1}{1 + e^{-t/\tau}} \tag{2.2.5}$$

kde parametr τ určuje strmost přechodu průběhu. Síť neuronů s těmito přenosovými funkcemi se, i přes původní definici perceptronu, označuje jako vícevrstvý perceptron. V aplikacích rozpoznávání řeči se používá třívrstvý MLP, skládající se ze vstupní, skryté a výstupní vrstvy. Síť je schematicky znázorněna na obrázku 2.2.1. Bylo dokázáno, že pokud tato síť obsahuje dostatečný počet neuronů ve skryté vrstvě, je schopna aproximovat jakoukoliv nelineární funkci.



Obrázek 2.2.2 Vícevrstvý perceptron

Mnohdy nebývá vstupní vrstva do celkového počtu tříd započítávána a celkový počet vrstev sítě pak tvoří skrytá a výstupní vrstva. Vícevrstvý perceptron používaný pro účely řečových aplikací používá v jednotlivých vrstvách následující přenosové funkce. Vstupní vrstva provádí lineární transformaci, výstup neuronů této sítě je dán vztahem 2.2.1. Aktivační funkci neuronů skryté vrstvy představuje sigmoida s parametrem $\tau = 1$. Přenosová funkce výstupní vrstvy softmax dána vztahem 2.2.6

$$y = \frac{exp(x)}{\sum_j exp(x_j)} \tag{2.2.6}$$

zajišťuje, že výstup sítě může nabývat pravděpodobnostního charakteru. Součet výstupních hodnot sítě je tedy roven 1. V aplikacích zpracování řeči se používá téměř výhradně ve výstupní vrstvě tato funkce. Aposteriorní pravděpodobnosti, které jednotlivé výstupní hodnoty sítě představují, jsou často použity k dalšímu zpracování, a to zejména v hybridních rozpoznávačích. Jedním z příkladů hybridního rozpoznávače je spojení ANN a skrytých Markovových řetězců, kde výstupy sítě jsou připojeny k příznakovému vektoru, použitém v HMM, ke zlepšení úspěšnosti rozpoznávání.

2.2.2 Učení MLP sítě

Při učení jsou váhy neuronové sítě nastavovány tak, aby byla do sítě uložena požadovaná informace. Proces učení lze chápat jako proces minimalizace chybové funkce, tedy rozdílu výstupu sítě od požadované hodnoty.

Pro trénování sítí za účelem zpracování řečových příznaků je potřeba k dispozici velký počet trénovacích dat. Síť je používána ke generalizaci, tedy ke klasifikaci takových vstupních dat, které nebyly součástí trénovací množiny. Pro tento účel je nutné zamezit přetrénování sítě rozdělením množiny dat na podmnožinu dat trénovacích, validačních a testovacích. Z důvodu lepší konvergence chybové funkce se používá dávkové trénování, kdy hodnota vah je aktualizována po přivedení určitého počtu příznaků do sítě. Tento počet je označován anglickým slovem *bunch*. Při tomto způsobu trénování je během průchodu jedné bunch množiny počítán gradient chybové funkce. Tento algoritmus je označován jako *gradient descent (steepest descent)*.

Hodnota vah v následujícím kroku je dána vztahem 2.2.7.

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \mu \nabla E \tag{2.2.7}$$

Počáteční odhad vah je náhodný. Algoritmus nastavuje hodnotu vah ve směru poklesu hodnoty chybové funkce. K tomuto účelu používá záporný gradient chybové funkce ∇E. Pro získání gradientu je nezbytné provést zpětné šíření chyby (*error backpropagation*). Velikost kroku, po kterém je upravována hodnota vah, udává parametr μ, který je označován jako rychlost učení (*learning rate*). Volba tohoto parametru je klíčová pro úspěšný proces učení. Příliš velká hodnota vede k situaci, že průběh konvergence chyby přejde lokální minimum a začne oscilovat. Naproti tomu příliš malá hodnota způsobí pomalou konvergenci průběhu. Proto je vhodné hodnotu parametru μ během učení vhodným způsobem měnit. V praxi se osvědčilo schéma pro nastavování rychlosti učení označované jako *new bob*. Rychlost učení je během trénování zachovávána konstantní po takovou dobu, dokud je přesnost validace větší než předem definovaná prahová hodnota, poté je v následujících epochách hodnota rychlosti učení zmenšována o konstantu, která v anglické literatuře nese označení *halving factor*. Učení je zastaveno v momentě, kdy nárůst přesnosti validace dvou po sobě následujících epoch nedosáhne dané velikosti.

Velikost bunch množiny a hodnota rychlosti učení hrají klíčovou roli pro úspěšné trénování sítě. Jejich nastavení bylo předmětem experimentů a projevilo se na dosažených výsledcích.

2.2.3 Paralelizace učení sítě

Paralelizace učícího algoritmu není z důvodu vzájemné závislosti dat příliš jednoduchá. Během výpočtů je potřebná znalost hodnot téměř o celé síti. Pro paralelizaci trénování existují dva přístupy:

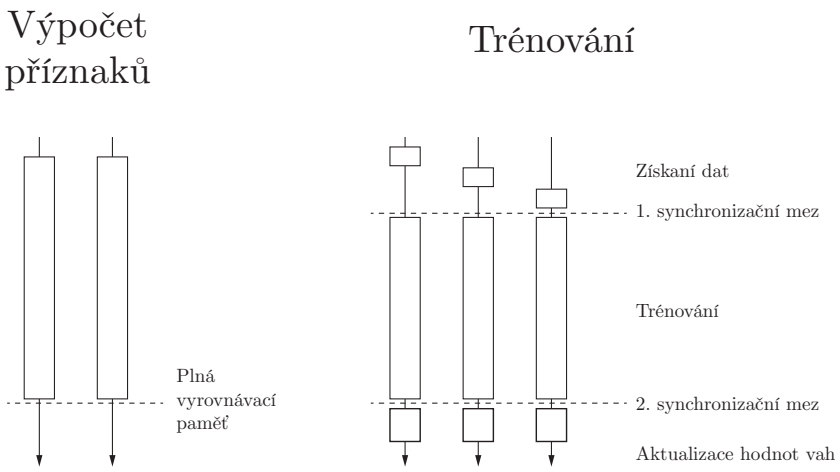
- Paralelizace uzlů: Při použití této metody je síť rozdělena a každé vlákno zpracovává danou množinu neuronů. Po ukončení tréninku je rozdělená síť opět spojena. Pokud sítě, vzniklé rozdělením, tvoří disjunktní množiny neuronů, tak spojená síť vykazuje lepší výsledky. V nejlepším případě bude spojené síť vykazovat stejné výsledky, jako síť s dávkovým učením. [16]. Nevýhodou je, že malý výkon vyrovnávací paměti zpomalí proces trénování v případě, pokud jsou jednotlivé podmnožiny neuronů malé. Tato metoda vyžaduje častější synchronizaci. Jednotlivá vlákna jsou synchronizována tzv. synchronizační mezí (anglický výraz barrier), před tím, než se přistoupí k další vrstvě.

- Paralelizace dat: Trénovací množina je rozdělena na disjunktní podmnožiny trénovacích dat. Každé vlákno pak pracuje s kopií sítě, kterou trénuje na přiřazené podmnožině. Váhy jsou synchronizovány po každé bunch množině. Jednotlivé matice rozdílů vah jsou sečteny a je vypočítána nová množina vah, která je distribuována mezi jednotlivé kopie sítě. [17]

Nástroj TNet používá ve své implementaci paralelizaci dat. Trénování je rozděleno do několika vláken, a to do dvou vláken pro výpočet příznaků a několika vláken pro trénink. Proces je rozdělen do tří částí:

1. Distribuce dat: prováděna v sérii
2. Samotný proces trénování: prováděn paralelně
3. Spojení matic rozdílů vah a aktualizace hodnot vah: prováděn paralelně

Synchronizace jednotlivých vláken ukazuje obrázek 2.2.3. Synchronizační hranice vláken výpočtu příznaků je zaplnění vyrovnávací paměti. Pro synchronizaci vláken při tréninku jsou použity dvě meze a to před začátkem tréninku a před spojováním hodnot [17].



Obrázek 2.2.3 Synchronizace jednotlivých vláken vláken, zdroj: [17], upraveno

2.2.4 Vyhlazení klasifikace pomocí HMM

Výsledky trénování, tedy přiřazení daných segmentů do jedné ze zvoleného počtu tříd, které reprezentují jednotlivé fonémy, lze provést přímo na základě aposteriorní pravděpodobnosti, která reprezentuje výstupní hodnoty. Přiřazení fonému danému vstupnímu segmentu zjistíme snadno podle nejvyšší hodnoty pravděpodobnosti na výstupu.

Tímto způsobem není ovšem nijak možné zohlednit pravděpodobnosti ostatních tříd a pokusit se tak kompenzovat chybu klasifikace. Jiný způsobem vyhodnocování výsledků klasifikace v oblastech rozpoznávání řeči je přístup založený na akustickém modelování, kde jsou účinným nástrojem skryté Markovovy modely.

Skryté Markovovy modely

Skrytý Markovovův model (HMM - Hidden Markov Model)je stochastický proces, u něhož nelze jednotlivé stavy pozorovat přímo, ale pouze prostřednictvím dalších náhodných procesů, které generují sekvenci pozorování. Tento princip je analogický vytváření řeči, kdy během promluvy posluchač neví, v jakém stavu se artikulační ústrojí mluvčího nachází, ale pozoruje (slyší) to, co tento stav produkuje. Skryté Markovovy modely charakterizují následující prvky:

1) Počet stavů modelu **N**. Jednotlivé stavy jsou označeny jako $S = \{S_1, S_2, \ldots, S_N\}$, stav v čase t je pak q_t.

2) Počet jednotlivých symbolů pozorování během jednoho stavu **M**, kde jednotlivé symboly označujeme jako $\mathbf{O} = \{o_1, o_2, \ldots, o_M\}$.

3) Rozdělení pravděpodobnosti přechodů mezi stavy $A = \{a_{ij}\}$, kde platí vztah 2.2.8.

$$a_{ij} = \mathrm{P}[q_{t+1} = S_j | q_t = S_i], \quad 1 \leq i, j \leq N \tag{2.2.8}$$

Tedy pravděpodobnost, s jakou bude v čase t ze stavu S_j dosažen stav S_i v čase $t+1$.

4) $B = \{b_j(k)\}$, rozdělení pravděpodobností symbolů pozorování ve stavu j, které je dáno

$$b_j(k) = \mathrm{P}[o_k \text{v čase t } | q_t = S_j], \quad \begin{matrix} 1 \leq j \leq N \\ 1 \leq k \ \leq M \end{matrix} \tag{2.2.9}$$

což je pravděpodobnost, s jakou bude v čase t se stavu S_j generován symbol o_k. Nejčastěji se jedná o Gaussovské funkce, které jsou charakterizovány rozptylem σ a střední hodnotou μ.

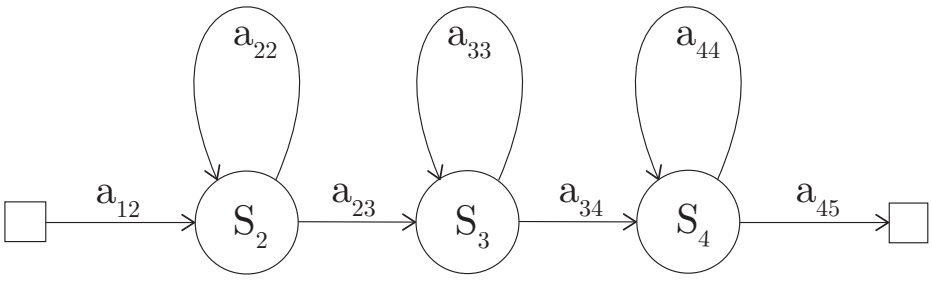
5) Rozdělení počátečních stavů $\pi = \{\pi_i\}$, pro které platí

$$\pi_i = \mathrm{P}[q_1 = S_i] \quad 1 \leq i \leq N \tag{2.2.10}$$

Často se pro charakterizaci modelu λ používá zkrácené notace:

$$\lambda = (A, B, \pi) \tag{2.2.11}$$

Struktur HMM existuje několik a jejich volba záleží na zvolené aplikaci. V oblastech rozpoznávání řeči se používají výhradně tzv. levo-pravé modely s 5 stavy, které modelují jazykové jednotky menší, než jsou celá slova, a to z důvodu potřeby obrovského množství trénovacích dat, které by byly potřebné k natrénování modelů na úrovni slov, jelikož pro každé slovo by bylo nutné mít několik jeho realizací od různých mluvčích. Použitím např. fonémů, jejichž výskyt je v jednom slově často opakovaný, se velikost trénovací databáze značně redukuje. Příklad 5-ti stavového modelu fonému je znázorněn na obrázku 2.2.4.



Obrázek 2.2.4 Příklad levo - pravého HMM.

Vlastností takového modelu je, že s postupujícím časem je dovolen přechod pouze mezi stavem, jehož index je větší než index stavu předchozího, platí tedy

$$a_{ij} = 0 \quad i > j. \tag{2.2.12}$$

S postupujícím časem se mezi jednotlivými stavy v modelu přechází zleva doprava. Rovněž pro pravděpodobnost počátečního stavu v tomto případě platí

$$\pi_i = \begin{cases} 0 & i \neq 1 \\ 1 & i = 1 \end{cases} \tag{2.2.13}$$

V tomto modelu existují dva typy stavů, a to emitující, který generuje pozorování na základě rozdělení pravděpodobnosti B, a neemitující, který žádné pozorování negeneruje a slouží zejména k propojování modelů. Jednotlivými pozorováními v aplikaci HMM pro rozpoznávání řeči jsou vektory příznaků. Každý foném je reprezentován vlastním modelem. Rozpoznávání v tomto případě znamená zjistit, který model by s největší pravděpodobností generoval danou posloupnost pozorování. Pro každý model λ a pozorování O je tedy nutné vypočítat pravděpodobnost $P(O|\lambda)$. Pokud známe sekvenci stavů (příslušnost jednotlivých pozorovaní ke stavům), která tuto posloupnost pozorování vygenerovala, pak je výpočet této pravděpodobnosti (uvažujme 5 stavový model, sekvenci pozorování $O = \mathbf{o}_1, \mathbf{o}_2, \ldots \mathbf{o}_7$, kde stav 2 generoval $\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3$, stav 3 $\mathbf{o}_4, \mathbf{o}_5$ a stav 4 generoval vektory pozorování $\mathbf{o}_6, \mathbf{o}_7$):

$$P(O|\lambda) = a_{12}b_2(\mathbf{o}_1)a_{22}b_2(\mathbf{o}_2)a_{22}b_2(\mathbf{o}_3)a_{23}b_3(\mathbf{o}_4)a_{33}b_3(\mathbf{o}_5)a_{34}b_4(\mathbf{o}_6)a_{44}b_4(\mathbf{o}_7)a_{45} \tag{2.2.14}$$

Při rozpoznávání samozřejmě příslušnost pozorování ke stavům neznáme, a proto nejpřímočařejším způsobem jak $P(O|\lambda)$ zjistit, je její výpočet přes všechny možné kombinace stavů. Pokud je délka pozorování T a počet stavů N, pak počet potřebných výpočtů je $2TN^T$, což je velmi vysoký počet i pro malé hodnoty N a T. Způsob, jak tento počet snížit, poskytuje tzv. dopředně zpětná metoda (anglicky Forward-Back Procedure) nebo Viterbiův algoritmus. Jakmile je pro každý model vypočítána $P(O|\lambda)$, vybere se takový, pro který je tato hodnota největší

$$W = \underset{1 \leq v \leq V}{\operatorname{argmax}} \; P(O|\lambda^v) \tag{2.2.15}$$

kde W je hledaný model a V je celkový počet modelů.

HMM generují pozorování na základě rozdělení pravděpodobnosti A a B. Samotnému procesu rozpoznávání předchází trénování, kdy jsou nastavovány vhodné parametry modelu tak, aby byla maximalizována pravděpodobnost $P(O|\lambda)$.

HMM v tomto případě slouží k vyhlazení výstupních aposteriorních pravděpodobností sítě a přispívají tak ke zlepšení úspěšnosti klasifikace. V tomto případě není nutné tyto modely trénovat, protože jejich parametry jsou přesně stanoveny vzhledem ke známým výstupům MLP, které je nutné zlogaritmovat, aby odpovídaly gaussovskému rozložení. Pro fonémový rozpoznávač reprezentuje každý model jednotlivý foném. Přechod mezi stavy je nastaven na konstantní hodnotu, střední hodnota je nulová a vektor rozptylů

Gaussovských funkcí obsahuje špičku v takovém prvku, který reprezentuje danou třídu, tedy odpovídající foném. V případě, že je modelováno 45 fonémů a třídu pro foném [a] reprezentuje první prvek vektoru, dostáváme následující parametry:

- Vektor střední hodnoty:

$$\boldsymbol{\mu}(\mathrm{a}) = (\mu_1, \ldots, \mu_{45}) \quad \text{kde } \mu_n = 0 \text{ pro } n = 1 \ldots 45 \tag{2.2.16}$$

- Vektor rozptylů:

$$\boldsymbol{\sigma}(\mathrm{a}) = (1, \sigma_2, \ldots, \sigma_{45}) \quad \text{kde } \sigma_n = 1.10^{30} \text{ pro } n = 2 \ldots 45 \tag{2.2.17}$$

Obdobně pak pro foném [b]

- Vektor střední hodnoty:

$$\boldsymbol{\mu}(\mathrm{b}) = (\mu_1, \ldots, \mu_{45}) \quad \text{kde } \mu_n = 0 \text{ pro } n = 1 \ldots 45 \tag{2.2.18}$$

- Vektor rozptylů:

$$\boldsymbol{\sigma}(\mathrm{b}) = (\sigma_1, 1, \ldots, \sigma_{45}) \quad \text{kde } \sigma_n = 1.10^{30} \text{ pro } n = 1, 3 \ldots 45 \tag{2.2.19}$$

Matice přechodů má pro všechny modely stejný tvar:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.2.20}$$

Průchod skrytými Markovovými modely bude mít na výsledek největší vliv v případě, kdy se jednotlivé hranice hlásek budou sobě blížit. HMM pak bude mít funkci filtru a vyhladí výstupní hodnoty.

3 Implementace fonémového rozpoznávače

Na základě popsané teorie v předešlých kapitolách jsem implementoval fonémový rozpoznávač. K jeho realizaci byly použity softwarové nástroje balíčku TNet [12], který zastřešuje práci s MLP sítí. Pro zpracování řečového signálu byly použity nástroje HTK toolkitu [4].

Trénování a testování umělých neuronových sítí umožňuje také **NN toolbox Matlabu**. Výhodou tohoto toolboxu je možnost výběru z několika typů sítí a kompletní správa nad jejich parametry. Tyto sítě je rovněž možné graficky znázornit a ověřit si nastavení dané sítě. Značnou nevýhodou tohoto nástroje jsou vysoké hardwarové požadavky pro trénování sítí. Nástroj selhává pro sítě, u kterých počet skrytých neuronů dosáhne několika málo stovek. Umožňuje ovšem načtení vah, které mohou být výsledkem tréninku některého z předchozích nástrojů. Z tohoto důvodu je tento nástroj vhodný zejména pro experimentální účely.

Dalším nástrojem pro simulaci ANN je **Neuroph**. Neuroph je framework pro jazyk Java, umožňující implementaci základních typů umělých neuronových sítí. Skládá se z knihovny funkcí, které zastřešují operace s ANN, jako trénování, testování, výběr přenosových funkcí a další. Dále je dispozici grafické vývojové prostředí Neuroph Studio, v němž je možné pracovat se sítěmi bet nutnosti hlubší znalosti jazyka Java. Neuroph je možné využívat zdarma.

Existuje ovšem řada dalších nástrojů, které si liší svými vlastnostmi. Mezi rozšířený nástroj realizující MLP sítě pro rozpoznávání řeči patří **Quicknet** [10]. Tento nástroj neumožňuje urychlení výpočtů a proto časy potřebné k natrénování sítě jsou mnohonásobně větší než v případě TNetu. Práce s tímto nástrojem byla ukázaná například v [9]. **TNet** je poměrně nový nástroj určený k realizaci neuronových sítí primárně v oblastech rozpoznávání řeči. Jeho hlavním znakem je možnost paralelního trénování sítě a využití výpočetního výkonu grafických karet. Další vlastností je schopnost výpočtu příznaků až při samotném trénovaní nebo testování, což značně šetří úložný datový prostor. Zejména pro tyto vlastnosti byl k realizaci rozpoznávače v této práci zvolen právě tento nástroj.

Výše uvedené nástroje jsou jen jedny z mnoha jiných a jejich používání je nutné si nejprve osvojit a aplikovat s ohledem na požadované aplikace.

3.1 Výpočetní hardware

Pro účely časově náročných výpočtů jsem měl k dispozici server Amagi s jednotlivými výpočetními uzly, který je používán skupinou pro zpracování řeči na Katedře teorie obvodů. Hlavní uzel Amagi slouží k ukládání dat a obsahuje jednotlivé adresáře. K výpočetním účelům je určeno celkem 18 počítačů. Pro účely mé práce jsem měl k dispozici uzly s označením magi201, 205 a 208 osazené dvoujádrovým procesorem s frekvencí 2,6 GHz (magi205,208) a 2,2 GHz (magi201). Pro trénování za použití více vláken jsem měl k dispozici uzel magi217 s 8-mi vláknovým procesorem o frekvenci 3,5 GHz. Nainstalovaným operačním systémem na výpočetním serveru je Linux. Pro správu a obsluhu celého výpočetního clusteru je používán nástroj *Sun Grid Engine*, umožňující paralelní zpracování úloh, což vede ke značné redukci výpočetního času a zároveň slouží k rovnoměrnému vytěžování systému. Jednotlivé úlohy je možné řadit mezi dvě fronty, a to do hlavní fronty *main* a do fronty výkonnějších počítačů s označením *fast*.

3.1.1 Konkrétní softwarová implementace

S ohledem na používaný server a jeho parametry je fonémový rozpoznávač implementovaný formou skriptů, které byly napsány ve skriptovacích jazycích bash a perl.

3.2 Použitá databáze signálů

Zdrojem řečových signálů byla databáze SPEECON. Databázi tvoří promluvy dospělých a mluvčích dětského věku. Celkový počet 550 dospělých a 50 dětských mluvčích poskytuje velmi značnou variabilitu jednotlivých, foneticky bohatých promluv. K dispozici jsou nahrávky z různých prostředí, tedy s různou úrovní šumu. Dalším faktorem je možnost výběru kvality záznamu na základě použitého mikrofonu. Jednotlivé soubory jsou uloženy v bezhlavičkovém formátu vhodném k dalšímu zpracování. Bližší informace lze nalézt v dokumentaci k této databázi. Důležitým údajem pro zpracování signálů z této databáze je vzorkovací frekvence a velikost a uspořádání bitů kvantizačního slova. Na základě těchto údajů jsou voleny parametry pásmové filtrace. Svým bohatým obsahem, poskytujícím promluvy od různých mluvčích a s různou úrovní šumu, je tato databáze ideálním zdrojem dat pro trénování zvoleného klasifikátoru a testování jeho vlastností.

3.3 Používané typy souborů v HTK toolkitu

S ohledem na instruktážní charakter praktické části práce, která má za úkol demonstrovat používání nástrojů softwarového balíčku TNet, budou popsány jednotlivé soubory

používané během chodu těchto programů. Tyto soubory jsou spjaty s konvencemi zavedenými rozšířením HTK toolkitu a v aplikacích zpracování řeči je vhodné si používání těchto souborů osvojit.

3.3.1 Script File

V aplikacích rozpoznávání řeči se pro analýzu a zpracování používá velký počet souborů. Jednotlivé nástroje zpracovávají tyto soubory postupně, a proto by bylo nutné je pro každý používaný soubor spouštět zvlášť, což by vedlo ke složitější implementaci a zpomalení průběhu celého procesu. Tento problém řeší seznamy zpracovávaných souborů označované jako *script file*, zkráceně scp s příponou *.scp*. Veškeré soubory, které má daný nástroj zpracovat, jsou uloženy ve formě seznamu do scp souboru, jehož obsahem je název a cesta k danému souboru. Tento nástroj pak není nutné volat pro každý soubor zvlášť, ale jednotlivé soubory jsou vyčítány z scp souboru, který je vstupním parametrem tohoto nástroje. Pokud daný nástroj na základě vstupu generuje výstup, pak scp může obsahovat dva sloupce, kde první obsahuje cestu ke vstupnímu souboru a druhý pak cestu pro ukládání výstupu. Příklad takového scp souboru:

```
directory1/input_file1   directory1/output_file1
directory1/input_file2   directory1/output_file2
directory1/input_file3   directory1/output_file3
```

3.3.2 Label File

Klasifikátory je nutné nejprve natrénovat. MLP sítě jsou trénovány s učitelem, je tedy zapotřebí, aby pro trénovací data byly k dispozici i požadované výstupní hodnoty. V případě klasifikace fonémů to znamená, že pro daný vstupní řečový signál musí existovat soubor, který bude obsahovat jednotlivé fonémy, které se v signálu vyskytují, tedy bude představovat jeho fonetickou transkripci. Takový soubor je označován jako *label file* s příponou *.lab*. Tento soubor se skládá z časových značek a jednotlivých fonémů. Nejčastěji se skládá ze tří sloupců. První sloupec představuje časovou značku začátku fonému, druhý sloupec pak časovou značku konce fonému a třetí sloupec obsahuje samotný foném.

```
0 13925000 sil
13925000 15125000 cc
15125000 17425000 t
17425000 18625000 i
18625000 19725000 rr
19725000 21525000 i
```

Obrázek 3.3.1 Ukázka lab souboru

Příklad lab souboru ukazuje 3.3.1. Jednotlivé časové značky představují údaj ve $100\,ns$ jednotkách. Soubory mohou místo jednotlivých fonémů obsahovat pouze celá slova nebo obě varianty dohromady, což závisí na zvolené aplikaci.

3.3.3 Master Label File

Databáze řečových signálů čítají velké množství souborů a ke každému jednotlivému z nich by musel existovat příslušný lab soubor, uložený ve stejném adresáři, čímž by se počet souborů v databázi zdvojnásobil. Tento problém řeší tzv. *master label file* s extenzí .mlf, který umožňuje spojit fonetické transkripce souborů v databázi do jediného souboru.

```
#MLF!#
''*/ctyri.lab''
0 13925000 sil
13925000 15125000 cc
15125000 17425000 t
17425000 18625000 i
18625000 19725000 rr
19725000 21525000 i
```

Obrázek 3.3.2 Ukázka mlf souboru

Obrázek 3.3.2 ilustruje reprezentaci lab souboru 3.3.1 v jednom master label souboru. Každý master label file je uvozen hlavičkou `#MLF!#`. Dále následují název každého lab souboru a jeho obsah.

3.3.4 Konfigurační soubory

Každý nástroj v rámci HTK toolkitu vyžaduje nastavení různých parametrů, které se liší v závislosti na konkrétní aplikaci. Aby nebylo nutné při spuštění nástrojů vždy vypisovat seznam konfiguračních parametrů, umožňují tyto nástroje jejich načítaní z konfiguračních souborů, které bývají uloženy s koncovkou *.cfg*. Je tedy možné mít tyto soubory předpřipraveny a používat je pro konkrétní aplikace opakovaně. Tyto soubory zároveň specifikují dané aplikace globálních nástrojů. Například pro nástroj HCopy je možné vytvořit sadu konfiguračních souborů, specifikujících výpočet banky filtrů, mfcc, a dalších.

3.4 Používané typy souborů v TNet toolkitu

Nástroje TNet toolkitu umějí pracovat jak s scp, tak mlf soubory. Odlišný je ovšem přístup v předávání informací o požadovaných výpočtech, tedy o transformacích, a informacích o síti. Tyto informace jsou ukládány do textových souborů, které se skládají

z jednotlivých komponent. Název každé komponenty je uvozen znaky <>, za kterým následuje informace o vstupní a výstupní dimenzi. Komponenty mohou specifikovat například používanou přenosovou funkci, potom nemají žádný obsah, například komponenta

```
<softmax> 45 45
```

říká, že má být použita přenosová funkce softmax pro vrstvu se 45 neurony. Z důvodů šetření místa na disku jsou příznaky na vstupu sítě počítány za chodu při trénování. K jejich výpočtu slouží komponenty, které obsahují vektor nebo matici. Například pro aplikaci Hammingova okna délky 30 slouží komponenta

```
<window> 30 30
v 30
0.0800 0.0908 0.1225 0.1738 0.2422 0.3245 0.4169 0.5151 0.6144 0.7103
0.7981 0.8740 0.9342 0.9759 0.9973 0.9973 0.9759 0.9342 0.8740 0.7981
0.7103 0.6144 0.5151 0.4169 0.3245 0.2422 0.1738 0.1225 0.0908 0.0800
```

Kde `v` označuje, že se jedná o vektor a číslo 30 pak udává počet prvků vektoru. Pokud je obsahem dané komponenty matice, například pro aplikaci diskrétní kosinové transformace, nachází se pod názvem komponenty písmeno `m`, následováno počtem řádků a sloupců matice.

V podobě těchto souborů jsou do sítě načítány i inicializační váhy a váhy, které jsou výsledkem po trénování. Bližší popis jednotlivých komponent obsahuje textový soubor, přiložený v adresáři TNetu při jeho stažení. Znalost syntaxe těchto souborů je důležitá zejména pro vyčítání hodnot. K jejich tvorbě slouží jednotlivé nástroje, které jsou distribuovány spolu s hlavními programy TNet toolkitu.

3.5 Popis používaných nástrojů

3.5.1 HTK toolkit

Pro účely konkrétní implementace byly z HTK toolkitu použity následující nástroje.

HCopy

Kopíruje soubory mezi požadovanými adresáři. Během kopírování na základě zvolených parametrů provádí úpravu kopírovaných souborů. Tato úprava představuje výpočet různých parametrů, jako například MFFC parametrizace, výpočet banky filtrů a další. V mojí práci je tento nástroj použit k výpočtu banky filtrů pro soubory definované scp souborem. Konfigurační soubor, zajišťující požadovaný výpočet je vypsán v 3.6.1. Tento nástroj je součástí skriptu `FBankComputation.pl`

HList

Tento nástroje vypisuje obsah souborů v HTK formátů a umožňuje číst data také přímo z audio signálů. HList je aplikován ve skriptu `Normalize.sh`, kde vyčítá velikost dimenze vstupních dat z vypočítané banky filtrů.

HBuild

Slouží k realizaci sítě jednotlivých HMM. Definuje jejich uspořádání a propojení. Je použit při vyhodnocování výsledků za pomocí HMM. Vstupem pro tento nástroj je slovník používaných fonémů. HBuild je spolu spolu s následujícími dvěma nástroji použit ve skriptu `Result.sh`.

HVite

Realizuje obecný Viterbiho slovní rozpoznávač. Do tohoto nástroje vstupují vypočítané zlogaritmované pravděpodobnosti, síť HMM a slovníky fonémů. Výstupem jsou rozpoznané fonémy v podobě mlf souboru.

HResult

Slouží k analýze úspěšnosti klasifikace na základě referenčního mlf souboru signálů, které byly vypočítány pravděpodobnosti a dekódovaného souboru, produkovaného HVite.

Bližší popis funkce jednotlivých nástrojů je uveden v dokumentaci k HTK toolkitu [11].

3.5.2 TNet toolkit

TNet toolkit se skládá z 5 hlavních nástrojů. Spolu s nim jsou distribuovány další pomocné nástroje, zejména ve formě skriptů, které slouží ke generování různých souborů pro hlavní programy. Mezi hlavní nástroje patří:

TNet

Tento program realizuje trénování MLP sítě. Umožňuje počítání příznaků za chodu trénování a krosvalidaci. Výsledkem trénování jsou váhy sítě uložené v příslušných komponentách v textovém souboru. Je základem skriptu `TNetTrain.sh`

TNorm

Nástroj pro normalizaci vstupních dat. Používám jej ve skriptu `Normalize.sh`

TFeaCat

Slouží k simulaci MLP sítě. Váhy, které jsou výsledkem trénování, jsou tímto nástrojem načteny a pro požadovaný vstupní soubor je vypočítán výstup sítě. Spolu s HVite tvoří základ skriptu `Result.sh`.

TJoiner

Slučuje příznaky do větších souborů z důvodu rychlejšího kopírování a vyčítání dat z disku. Výsledný soubor je obdobou pfile souborům používaných v Quicknet toolkitu.

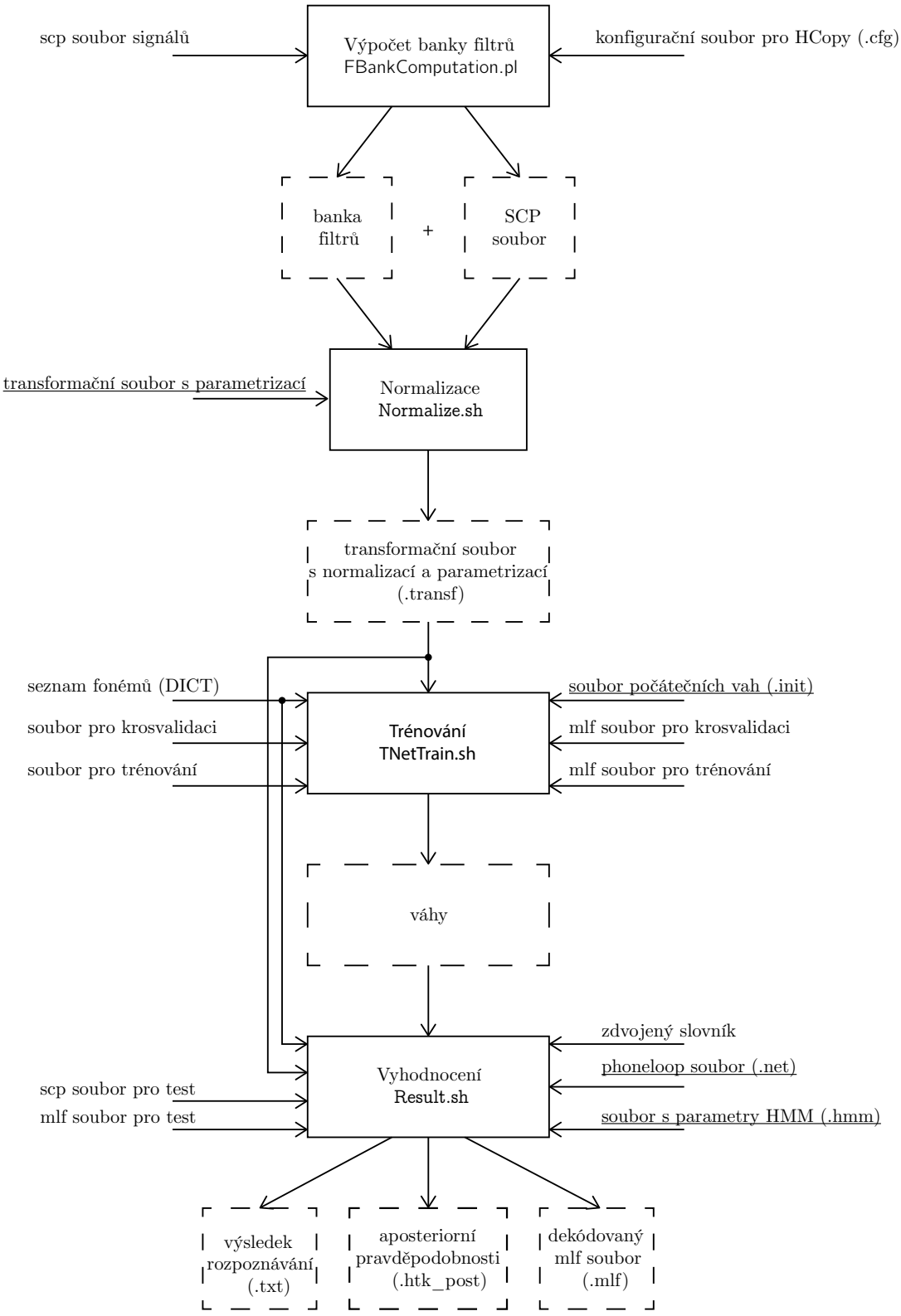
TSegmenter

Rozděluje sloučené příznaky na jednotlivé soubory. TSegmenter a TJoiner nejsou v současné implementaci rozpoznávače použity.

Jednotlivé parametry těchto nástrojů je možné získat z nápovědy a budou popsány v konkrétních aplikacích. Společnými pro všechny je parametr **-A**, který tiskne zadané argumenty, **-D** slouží k zobrazení zadaných proměnných v parametrech, **-T** nastavuje výpis průběhu a **-V** tiskne informaci o verzi nástroje.

3.6 Realizace fonémového rozpoznávače

Rozpoznávání fonémů je složeno z několika kroků, kterým odpovídají jednotlivé skripty rozpoznávače. Jedná se tedy o `FBankComputation.pl`, `Normalize.sh`, `TNetTrain.sh` a `Result.sh`. V prvním kroku je vypočítána banka filtrů, dále jsou vypočítané hodnoty normovány. Tyto výsledky spolu s bankou filtrů jsou použity k trénování MLP sítě. Výsledkem trénování jsou váhy, dále použité k vygenerování pravděpodobností, které jsou nakonec vyhodnoceny. Jednotlivé kroky jsou realizovány jako jednotlivé skripty, a to z důvodu jejich samostatného spouštění či možnosti použití v jiných aplikacích. Rozpoznávač je znázorněn v diagramu 3.6.1. Bloky vykreslené přerušovanou čarou představují výstupní soubory z jednotlivých skriptů. Podtržení znamená, že příslušný soubor je možné do skriptu dodat nebo jej nechat vygenerovat daným skriptem.



Obrázek 3.6.1 Schéma implementovaného rozpoznávače.

3.6.1 Adresářová struktura

Adresář fonémového rozpoznávače nese označení `PhonemeRecognizer` a je rozdělen do podadresářů `files`, `mlf`, `results`, `scp` a `tools`. `files` obsahuje různé soubory, které jednotlivé skripty vyžadují, jako např. konfigurační, transformační, inicializační aj. V podadresáři `mlf` jsou uloženy veškeré mlf soubory, se kterými rozpoznávač pracuje. Do `results` se ukládají výsledky klasifikace. Tento adresář je dále rozdělen na `log_files` a `weights`. Jak je patrné, do podadresáře `weights` jsou ukládány váhy z jednotlivých iterací a výsledné váhy trénovacího procesu. Pod adresář `log_files` obsahuje textové soubory se záznamem průběhu daného skriptu. Veškeré scp soubory odkazující na požadovaná data jsou obsahem adresáře `scp`. Ten je rozdělen na podadresáře odpovídající velikostem jednotlivým trénovacím množinám. Adresář `tools` obsahuje všechny nástroje potřebné pro rozpoznávání. Pro potřeby TRAP parametrizace je k dispozici ještě napočítaná banka filtrů, umístěná v adresářích `fbank_test` a `fbank_train_crosvall`.

```
PhonemeRecognizer
| -- files                  | -- results -- weights
| -- fbank_test                            \ -- log_files
| -- fbank_train_crosvall   | -- scp -- 500_set
| -- mlf                                 \ -- ...
                            | -- tools
```

3.6.2 Pojmenování výstupních souborů

Výsledkem procesu rozpoznávání je soubor s vahami, výsledkem klasifikace a průběhem trénování. Všechny tyto soubory jsou ukládány do složky results. Výsledky jsou získávány pro různé parametrizace signálu nebo parametry sítě. Aby bylo možné se mezi jednotlivými výsledky jednoduše orientovat a vyhnout se tak dlouhým názvům souborů, tak název každého výsledného souboru je uvozen daným typem, pro váhy `weights`, pro záznam průběhu `log` a pro výsledek klasifikace `results`. Následuje velikost trénovací množiny a reference na nastavené parametry. V adresáři s výsledky je uložen soubor se seznamem těchto referencí, ve kterém je možné si ověřit, pro jaké parametry byl soubor s výsledky získán. Pokud se jedná o záznam průběhu, potom na posledním místě v názvu je název skriptu, pro který byl tento záznam pořízen. Příklady názvů těchto souborů:

- soubor vah - `weights_TS500_ref_1`
- soubor s výsledky klasifikace - `results_TS500_ref_1.txt`
- soubor se záznamem průběhu trénování - `log_TS500_ref_1_TNetTrain.sh.out`

3.6.3 Paralelní spuštění skriptů

Každá část fonémového rozpoznávače může být spuštěna na některém z počítačů serveru, a to prostřednictvím SGE a příkazu qsub. Tento příkaz vyčítá parametry pro svůj běh přímo z jednotlivých skriptů, které jsou uloženy ve formě komentářů a uvozeny znakem `$`. Stačí tedy pouze spustit příkaz qsub s cestou k požadovanému skriptu. S ohledem na zvolenou adresářovou strukturu se jednotlivé skripty spouštějí z hlavního adresáře `PhonemeRecognizer`.

3.6.4 Výpočet banky filtrů

Výpočet banky filtrů realizuje skript `FBankComputation.pl` za použití nástroje HCopy. Vzorkovací frekvence signálů v databázi SPEECON je 16 kHz, proto je zvolený počet pásem 23. Frekvenční pásmo je pokryto v rozsahu 100 - 7000 Hz. Signál je segmentován s krokem 10 ms na jednotlivá okna délky 25 ms, je tedy realizován více jak 50 % překryv. Jednotlivé segmenty jsou váhovány Hammingovým oknem. Pro spektrální analýzu je použito výkonové spektrum. Kompletní konfiguraci pro HCopy udává tabulka 3.6.1.

Tabulka 3.6.1 Konfigurační parametry HCopy

SOURCEKIND	WAVEFORM	HIFREQ	8000
SOURCEFORMAT	ALIEN	NUMCHANS	23
HEADERSIZE	0	USEPOWER	T
NATURALREADORDER	T	USEHAMMING	T
NATURALWRITEORDER	F	PREEMCOEF	0
SOURCERATE	625	TARGETRATE	100000
BYTEORDER	VAX	WINDOWSIZE	250000
TARGETFORMAT	HTK	SAVEWITHCRC	F
TARGETKIND	FBANK	NUMCEPS	12
LOFREQ	0		

Pro spuštění skriptu je nutné definovat následující proměnné z tabulky 3.6.2. Na základě

Tabulka 3.6.2 Proměnné ve skriptu FBankComputation.pl

`fbank_dir`	adresář pro výpočet banky filtrů
`input_train_scp`	vstupní scp soubor s řečovými signály
`hcopy_twocolumn_scp`	umístění dvou-sloupcového scp souboru pro HCopy
`hcopy_configuration`	cesta ke konfiguračnímu souboru pro HCopy
`hcopy_configuration`	cesta ke konfiguračnímu souboru pro HCopy
`parts`	počet částí paraleního výpočtu

vstupního scp souboru, obsahujícího seznam signálů, pro které se má vypočítat banka filtrů, a zvoleného adresáře pro vypočítané hodnoty, je vytvořen scp soubor pro HCopy a

další scp soubor, obsahující cesty k vypočítaným hodnotám. Z důvodů velkého množství zpracovávaných souborů se používá paralelního výpočtu, kdy scp soubor pro HCopy je rozdělen na zadaný počet částí, které jsou prostřednictvím SGE rozdělělny ke zpracování.

3.6.5 Normalizace

Normalizace vstupních dat zajišťuje, že vstup sítě se pohybuje ve stejném intervalu, jako hodnoty výstupu sítě. Z tohoto důvodu je možné, aby byly váhy sítě inicializovány před trénováním náhodně. V případě, že by neproběhla normalizace vstupních dat, bylo by nutné najít takové řešení, kde by se některé hodnoty vah výrazně lišily od jiných [3].

Pro výpočet normalizace vstupních dat sítě je použit nástroj TNorm, který je součástí TNet toolkitu. Normalizaci provádí skript `Normalize.sh`. Proměnné, které se zde definují ukazuje tabulka 3.6.3.

Tabulka 3.6.3 Proměnné skriptu Normalize.sh

`SCP_IN`	scp soubor s daty, které mají být normovány
`OUT`	adresář pro uložení výsledku
`FRM_EXT`	délka trajektorie
`DCT_BASE`	počet hodnot DCT

Vlastnímu trénování již nepředchází žádná další úprava vstupních dat a jelikož se výpočet příznaků provádí za chodu při trénování sítě, je v tomto kroku generován soubor, obsahující informaci o prováděném výpočtu příznaků. Pro specifikaci délky trajektorie pro TRAP parametrizaci slouží proměnná `FRM_EXT`, určující délku levého a pravého časového kontextu. Výsledná délka je potom dvojnásobek této hodnoty navýšená o jeden rámec. Pro délku trajektorie 310 ms, při segmentaci s 10 ms překryvem, je hodnota této proměnné 15, tedy celková délka trajektorie je 31 rámců. Proměnná `DCT_BASE` udává počet použitých složek DCT. Ve skriptu se dále vyskytuje proměnná `DIM_IN`, která udává rozměr vstupních dat. Tato proměnná nemusí být nastavována ručně. Její hodnota je vyčtena pomocí nástroje HList ze vstupních dat, zadaných scp souborem. Pro zvolený počet pásem banky filtrů je její hodnota 23. Tyto proměnné jsou použity jako vstup nástroje `gen_ham_dct.py`, který vygeneruje Hammingovo okno a matici pro aplikaci DCT a uloží je v příslušných komponentách do zvoleného adresáře v souboru s názvem `tr_23Tcontext31_Ham_dct16`. Název a cesta tohoto souboru jsou uloženy do proměnné `MMF`. Normalizace je lineární transformace, a proto může být zkombinována s lineární funkcí, kterou realizuje vstupní vrstva. Jelikož je nutné provést normalizaci vstupních hodnot sítě, kterými jsou TRAP příznaky, které se počítají za chodu při učení, musí je normalizační nástroj také nejprve za chodu vypočítat a až poté normovat. Proto je soubor tr_23Tcontext31_Ham_dct16 spolu s proměnnou FRM_EXT použit jako

parametr pro nástroj TNorm, který se spouští s následujícími parametry:

```
TNorm -D -A -T 1 -S $SCP_IN -H $MMF --TARGET-MMF=$NORM
            --START-FRM-EXT=$FRM_EXT --END-FRM-EXT=$FRM_EXT
```

parametr `-S` předává scp soubor s daty pro normalizaci, `-H` nastavuje soubor s parametrizací. `-TARGET-MMF` nastavuje výstupní soubor, proto proměnná `NORM` obsahuje název souboru, do kterého je výsledek normalizace uložen. `-START-FRM-EXT` a `-END-FRM-EXT` nastavují počet rámců časové trajektorie. Výsledek normalizace je zapsán do komponenty `<bias>`, která přidává práh vstupní vrstvě. Dále výsledný soubor obsahuje komponentu `<window>`, do které je uložen výsledek normalizace rozptylu. Tyto dvě komponenty jsou uloženy v souboru `tr_23Tcontext31_Ham_dct16.norm`. Posledním krokem, který skript provede, je vygenerování souboru s výsledkem normalizace a informací o prováděné parametrizaci. Tento soubor vznikne sloučením tr_23Tcontext31_Ham_dct16 a tr_23Tcontext31_Ham_dct16.norm souborů do výsledného transformačního souboru s názvem tr_23Tcontext31_Ham_dct16.transf.

3.6.6 Trénování sítě

K natrénování MLP sítě slouží skript `TNetTrain.sh`. Ve své podstatě se jedná o konfigurační skript, kde jsou nastavovány parametry a soubory potřebné pro trénování. Parametry, které se zde nastavují, obsahuje tabulka 3.6.4. `BUNCHSIZE` je počet vstup-

Tabulka 3.6.4 Proměnné ve skriptu TNeTrain.sh.

`THREADS`	počet vláken paralelního výpočtu
`SCP_CV_LOCAL`	scp soubor s daty pro krosvalidaci
`SCP_TRAIB_LOCAL`	scp soubor s daty pro trénování
`MLF_CV`	mlf soubor pro data při krosvalidaci
`MLF_TRAIN`	mlf soubor pro data při trénování
`PHONELIST`	slovník používaných fonémů
`FEATURE_TRANSFORM`	transformační soubor s normalizací a parametrizací
`FRM_EXT`	délka trajektorie
`LEARNRATE`	rychlost učení
`BUNCH_SIZE`	velikost bunch množiny
`CACHE_SIZE`	velikost míchacího bufferu
`RANDOMIZE`	náhodné vybírání dat při učení
`TRACE`	nastavení výpisu

ních vektorů, po kterých se provádí změna hodnoty vah. `CACHESIZE` představuje velikost bufferu, do kterého jsou vstupní hodnoty načteny. Poté jsou náhodně zamíchány a tento buffer je rozdělen do na jednotlivé množiny, jejichž velikost udává `BUNCHSIZE`. Čím je hodnota `CACHESIZE` větší, tím více dat je mezi sebou promícháno, jsou také ovšem

kladeny větší nároky na paměť. Proměnné odkazující na soubory spolu s nastavením rychlosti učení jsou povinnými parametry. Počet vláken paralelního výpočtu závisí na parametrech procesoru, který bude při výpočtu použit. Pro maximální rychlost výpočtu je nutné aby, počet vláken odpovídal počtu jader procesoru. Ke spuštění trénování sítě nástroj TNet vyžaduje seznam výstupních hodnot, tedy fonémů, které odpovídají jednotlivým třídám. Tento seznam obsahuje soubor, na který odkazuje proměnná `PHONELIST`. Jedná se o textový soubor, kde každý foném je na jednotlivém řádku. Takový soubor tedy představuje slovník používaných fonémů. Výsledek normalizace spolu s informacemi potřebnými pro výpočet požadované parametrizace je uložen v souboru, který je výstupem z předchozího kroku a odkazuje na něj proměnná `FEATURE_TRANSFORM`.

Dále se nastavují parametry 3.6.5, které přímo ovlivňují proces učení. Je možné nasta-

Tabulka 3.6.5 Nastavení parametrů učení.

`MAX_ITER`	maximální počet iterací
`MIN_ITER`	minimální počet iterací
`KEEP_LRATE_ITER`	počet iterací s konstantní hodnotou rychlosti učení
`END_HALVING_INC`	prahová hodnota pro ukončení trénování
`START_HALVING_INC`	prahová hodnota zahájení změny učící ryhlosti
`HALVING_FACTOR`	parametr změny rychlosti učení

vit maximální a minimální počet iterací prostřednictvím `MAX_ITER` a `MIN_ITER`. Parametr `START_HALVING_INC` zahájí změnu rychlosti učení v případě, že přesnost klasifikace v současné iteraci je pouze o tento parametr větší, než v iteraci předchozí. Tedy pokud se příliš nezměnila. Proměnná `END_HALVING_INC` určuje, kdy dojde k ukončení trénování. Pokud již proběhla úprava rychlosti učení a přesnost v současné iteraci je maximálně o tento parametr větší, je trénování ukončeno. `HALVING_FACTOR` určuje o jakou hodnotu je v každé následující iteraci zmenšena rychlosti učení.

Dále je z transformačního souboru určen počet vstupů a ze slovníku fonémů pak počet výstupů sítě. Důležitým parametrem je `hidden`, který nastavuje počet neuronů ve skryté vrstvě. Informace o rozměrech sítě jsou použity nástrojem `gen_mlp_init.py`. Tento generuje inicializační soubor sítě, tedy nastavení jednotlivých přenosových funkcích ve všech vrstvách a počáteční hodnoty vah. Název inicializačního souboru má následující strukturu názvu: `nnet_`*počet vstupů*`_`*počet skrytých neuronů*`_`*počet výstupů*`.init`, například pro 368-mi rozměrný vstupní vektor, 1000 skrytých neuronů a 45 výstupních tříd je název souboru `nnet_368_1000_45.init`.

Po nastavení všech výše popsaných parametrů je možné přistoupit k samotnému trénování sítě. K tomuto účelu je použit jeden z distribuovaných skriptů s názvem `training_scheduler.sh`. Pokud nebyly parametry trénování nastavení prostřednictvím výše popsaných hodnot, jsou nastaveny implicitní hodnoty 3.6.6.

Tabulka 3.6.6 Výchozí parametry trénování.

BUNCHSIZE	512
CACHESIZE	32768
RANDOMIZE	TRUE
TRACE	5
TNET_FLAGS	-A -D -V
MAX_ITER	20
MIN_ITER MIN_ITER	1
KEEP_LRATE_ITER	0
END_HALVING_INC	0.1
START_HALVING_INC	0.5
HALVING_FACTOR	0.5

Samotné trénování zajišťuje funkce `run_tnet_parse_accu`, která jako vstupní argument přebírá příkaz ke spuštění TNetu a po jeho skončení ukládá do proměnné `ACCU` úspěšnost klasifikace. Při spuštění skriptu je v prvním kroku provedena počáteční krosvalidace. Déle je v cyklu prováděno trénování a krosvalidace přičemž výsledné váhy jsou vždy použity v následujícím kroku. Během trénování je kontrolována řada podmínek. Váhy mohou být akceptovány bez ohledu na porovnání úspěšnosti z předchozí iterace a hodnota rychlosti učení je zachována konstantní. Počet iterací, po jejichž dobu je zachována konstantní hodnota rychlosti učení, určuje proměnná `KEEP_LRATE_ITER`. Další podmínka kontroluje úspěšnost klasifikace v současné iteraci oproti iteraci předchozí. Pokud je současném kroku úspěšnost menší, zachová váhy z minulé iterace, v opačném případě jsou v následujícím kroku použity současné váhy. Dále je kontrolována změna úspěšnosti, podle které se zahajuje změna rychlosti učení nebo je trénování ukončeno.

Spuštění krosvalidace se provede následujícím příkazem:

```
TNet -A -D -V -T 2 -H nnet_368_1000_45.init
-I train_crossvall_set.mlf
-L '*/' -X CS0.lab -S features_crossval_set_50.scp
--BUNCHSIZE=512 --CACHESIZE=32768
--RANDOMIZE=FALSE --CROSSVALIDATE=TRUE
--OUTPUTLABELMAP=dict
--STARTFRMEXT=15 --ENDFRMEXT=15
--FEATURETRANSFORM=norm_train_set_ref_1.transf --THREADS=2
```

Ke spuštění trénování slouží příkaz:

```
TNet -A -D -V -T 2 -H nnet_368_1000_45.init
-I train_crossvall_set.mlf
-L '*/' -X CS0.lab -S features_train_set_450.scp
--LEARNINGRATE=0.008 --BUNCHSIZE=512
--CACHESIZE=32768 --RANDOMIZE=TRUE
--OUTPUTLABELMAP=dict --TARGETMMF=nnet_368_1000_45_iter01
--STARTFRMEXT=15 --ENDFRMEXT=15
--FEATURETRANSFORM=norm_train_set_ref_1.transf --THREADS=2
```

Paremetr `-H` nastavuje soubor s inicializačními váhami. Pro nastavení mlf souboru, odpovídajícího zpracovávaným datům, slouží parametr `-I`. Argument `-L` specifikuje cestu k jednotlivým lab souborům. Příponu těchto lab souborů lze určit parametrem `-X`. Scp soubor s trénovacími nebo krosvalidačními daty se předává prostřednictvím parametru `-S`. Provedení krosvalidace určuje `-CROSSVALIDATE=TRUE`. Seznam používaných fonémů předává parametr `-OUTPUTLABELMAP`. Váhy jsou ukládány do souboru, který je určen parametrem `-TARGETMMF`. Transformační soubor příznaků, tedy normalizace a parametrizace je načítán prostřednictvím parametru `-FEATURETRANSFORM`.

Výstupní váhy z jednotlivých iterací jsou ukládány v souborech, které ve svém názvu obsahují informaci o rozměrech sítě, pořadí iterace a úspěšnosti při trénování a krosvalidaci. Soubor výsledných vah v názvu obsahuje navíc slovo final.

3.6.7 Vyhodnocení výsledků

Pro vyhodnocení úspěšnosti klasifikace jsou použity HMM, se kterými pracují nástroje HTK. Vyhodnocení aposteriorních pravděpodobností sítě provádí skript `Result.sh`. V prvním kroku je v kořenovém adresáři vytvořena složka `result_files`, do které budou zapisovány jednotlivé soubory a výsledky v průběhu chodu skriptu. Tento způsob je zvolen kvůli jednotlivým experimentům, kdy je nutné provádět vyhodnocování vždy pro jiné váhy sítě. Stačí tedy tento adresář smazat a spustit vyhodnocování znovu. Základem tohoto skriptu jsou nástroje HTK `HBuild`, `HVite` a `HResult` a nástroj TNet toolkitu `TFeaCat`. Nástroj `HBuild` generuje smyčku, resp. síť rozpoznávaných slov, tedy strukturu propojení jednotlivých HMM. Vstupním parametrem je slovník používaných fonémů `dictmono` na jehož základě je vygenerována výsledná síť do souboru v proměnné `phoneloop`. Parametry jednotlivých HMM, matice přechodu, rozptyl a střední hodnotu a jejich strukturu, generuje nástroj distribuovaný s TNetem označený jako `gen_HTK_gmmbypass_1s.sh` podle konvencí HTK na základě slovníku `dictmono` do souboru definovaného v proměnné `gmmbypass`. `TFeaCat` generuje na základě vah, které jsou

výstupem z trénování sítě, výstupní pravděpodobnosti testovacích dat. Tento nástroj je spouštěn s následujícími parametry:

```
TFeaCat -D -A -T 1 -S $scptest -H $weights -l $posteriorsdir -y htk_post
        --FEATURETRANSFORM=$featuretransform --GMMBYPASS=true
        --START-FRM-EXT=15 --END-FRM-EXT=15
```

proměnná `scptest` obsahuje scp soubor s odkazy na vypočítanou banku filtrů. `weights` předává váhy z trénování. Výsledné pravděpodobnosti jsou ukládány do `posteriorsdir`. Parametr `-y` nastavuje příponů souborů s pravděpodobnostmi, které jsou uloženy v htk formátu. Opět z důvodu výpočtu příznaků za chodu nastavují zbylé hodnoty parametry pro jejich výpočet, jako v předešlých případech. HMM, které jsou použity pro vyhodnocení klasifikace, požadují zlogaritmované hodnoty výstupů sítě, což zajišťuje parametr `-GMMBYPASS`. Nástroj `HVite` vygeneruje na základě výstupů sítě rozpoznané promluvy v podobě MLF souboru, uloženého v proměnné `decodedmlf`. Tento nástroj rovněž vyžaduje slovník s výslovností jednotlivých slov, kde v prvním sloupci je dané slovo a ve druhém pak jeho výslovnost. Jelikož pracujeme s fonémy obsahuje tento soubor dva stejné sloupce a je předáván prostřednictvím proměnné `dict`. SCP soubor s odkazy na vypočítané pravděpodobnosti je uložen v `posteriorsscp`. Nakonec vyhodnocení na základě dekódovaného mlf souboru a referenčního mlf souboru testovacích dat `mlftest` provede nástroj `HResults`. Výsledek rozpoznávání je uložen v souboru `output_result_file`. Podstatným parametrem ohodnocení úspěšnosti je `Acc`, který je vypočítán podle vztahu 3.6.1.

$$Acc = \frac{H - I}{N} \cdot 100\% \tag{3.6.1}$$

kde H je počet správně klasifikovaných fonémů, I je počet fonémů, které byly vloženy navíc N je celkový počet fonémů vyskytujících se v testovacím mlf souboru. Tento parametr v sobě zahrnuje veškeré chyby, k jakým při klasifikaci došlo, a proto se jedná o objektivní ohodnocení úspěšnosti. Dalším parametrem ve výsledném souboru je `Corr`, který vyjadřuje pouze procentuální podíl 3.6.2 správně rozpoznaných fonémů vůči jejich celkovému počtu.

$$Corr = \frac{H}{N} \cdot 100\% \tag{3.6.2}$$

Dalšími parametry ve výsledném souboru jsou D, počet chybějících fonémů a S, představující počet špatně rozpoznaných (nahrazených) fonémů.

4 Experimenty

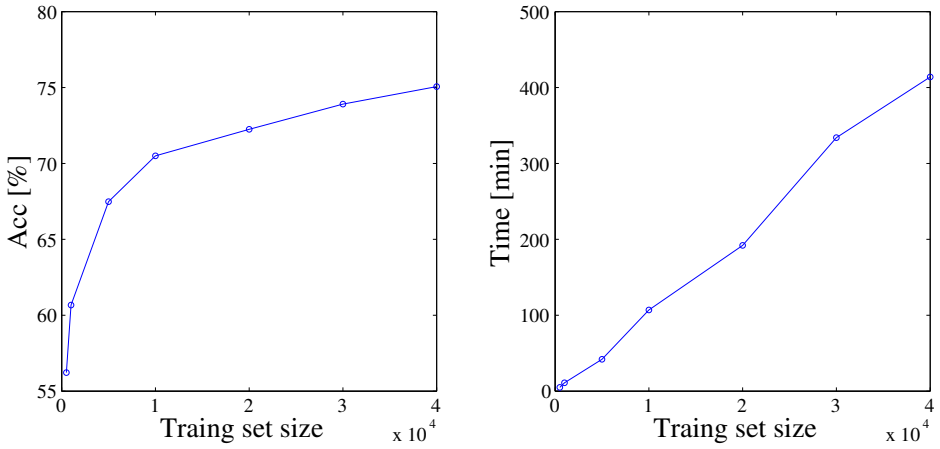
Úspěšnost klasifikace a délka trénování je ovlivněna několika faktory. Následující výsledky ukazují vliv vybraných parametrů na tyto hodnoty. Výstupní pravděpodobnosti sítě byly vyhlazeny za použití HMM a výsledná přesnost je udávána na úrovni fonémů. Pro testování sítě byla ve všech případech použita množina 4035 signálů databáze SPEECON. Pro parametrizaci řečového signálu byla použita metoda TRAP s délkou trajektorie 31 rámců, která při 23 pásmech banky filtrů generovala vektor příznaků o velikosti dimenze 368. Rovněž pro všechny případy platilo rozdělení trénovací množiny na 10% dat krosvalidačních a 90% dat trénovacích. Výstup sítě tvoří 45 hodnot, odpovídajících jednotlivým fonémům.

4.1 Velikosti trénovací množiny

Velikost trénovací množiny má vliv na schopnost generalizace sítě, tedy klasifikace takových vzorů, které nebyly obsaženy v trénovací množině. Čím víc trénovacích dat má síť během tréninku k dispozici, tím lépe je schopna kompenzovat variabilitu příznaků patřících do stejné třídy. Počet potřebných trénovacích vzorů lze přibližně určit podle vztahu 4.1.1 uvedeného v [18].

$$N_p = \frac{N_w}{e} \tag{4.1.1}$$

N_p představuje počet trénovacích vzorů, N_w počet vah a e je požadovaná chyba. Pokud tedy při konstantní velikosti sítě požadujeme snižovat chybu klasifikace, zvyšuje se počet potřebných trénovacích dat. Obdobně pak lze tvrdit, že při zachování velikosti sítě a zvyšování velikosti trénovací množiny, klesá chyba klasifikace. Graf 4.1.1 ukazuje závislost počtu trénovacích dat na úspěšnosti klasifikace. Tyto hodnoty byly dosaženy pro síť s 1000 skrytými neurony. K trénování byla použita 2 vlákna paralelního trénování. Tato krosvalidační množina byla v průběhu celého procesu trénování stejná. Graf 4.1.1 rovněž ukazuje závislost trénovací množiny na čase trénování. Jak je z průběhů vidět, lineární nárůst velikosti trénovací množiny má za následek téměř exponenciální nárůst úspěšnosti klasifikace, přičemž závislost počtu trénovacích dat na čase, potřebném k natrénování sítě, se blíží lineárnímu průběhu. Jednotlivé hodnoty uvádí tabulka 4.1.1.



Obrázek 4.1.1 Závislost úspěšnosti a doby trénování na velikosti trénovací množiny.

velikost trénovací množiny	Acc [%]	čas	počet iterací
500	56,22	5 min	7
1000	60,67	11 min	9
5000	67,48	42 min	8
10000	70,50	1h 47 min	10
20000	72,25	3h 12 min	9
30000	73,91	5h 34 min	8
40000	75,06	6h 54 min	10

Tabulka 4.1.1 Hodnoty úspěšnosti a doby trénování na zvolených parametrech.

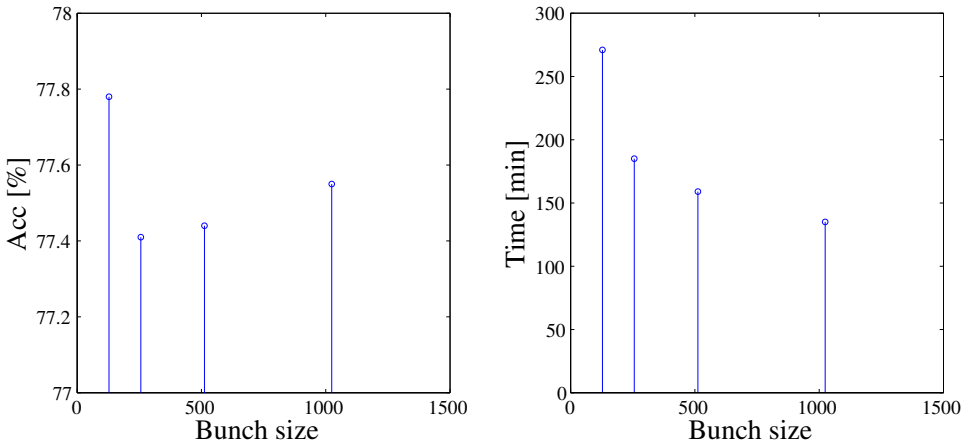
4.2 Velikost dávkové množiny

Hodnota vah je při trénování aktualizována po zvolených dávkách (parametr BUNCH-SIZE). Velikost této dávkové (bunch) množiny ovlivňuje zejména dobu potřebnou k natrénování sítě. Příliš malá hodnota zpomaluje trénování a naopak příliš velká hodnota způsobuje špatnou konvergenci trénovací chyby. Trénování bylo ve všech případech paralelizováno na 8-mi vláknech s rychlostí učení 0.008. Tabulka 4.2.1 obsahuje naměřené hodnoty. Závislost velikosti dávkové množiny na úspěšnosti a čase je znázorněna v grafu 4.2.1.

Velikost trénovací množiny	Počet skrytých neuronů	Velikost dávkové množiny	Acc [%]	čas	počet iterací
40 000	2000	128	77.78	4 h 31 min	10
		256	77.41	3 h 5 min	9
		512	77.44	2 h 39 min	9
		1024	77.55	2 h 15 min	10

Tabulka 4.2.1 Vliv velikosti dávkové množiny.

Hodnotu 1024 jsem zvolil jako vhodný kompromis mezi dobou trénování a úspěšností pro další experimenty. Pro demonstraci uvádím výsledek pro hodnotu 8192 a 4000 skrytých neuronů. Úspěšnost v tomto případě klesla na 58.03 % a trénování trvalo 7 hodin a 39 minut. V případě času je nutné brát ohled na hardwarové možnosti, kdy patrně byla překročena maximální rychlost mezi jednotlivými komponentami z důvodu zpracovávání velkého množství dat.



Obrázek 4.2.1 Grafické znázornění vlivu velikosti dávkovací množiny.

4.3 Velikost skryté vrstvy

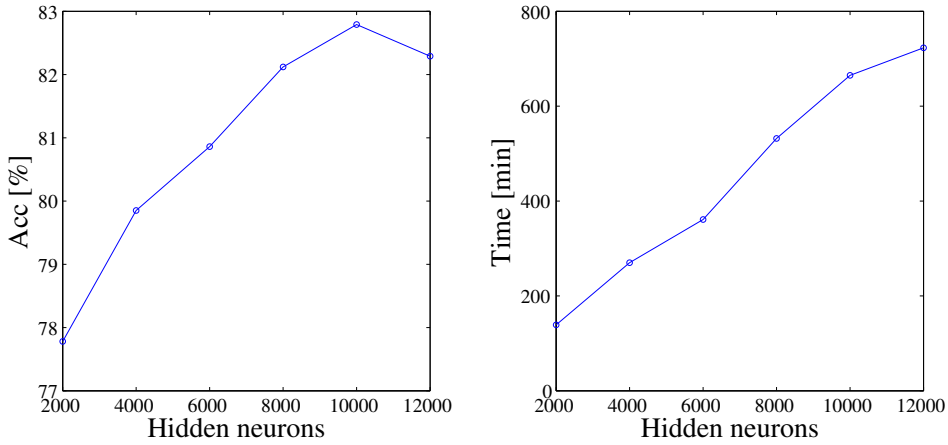
Dalším parametrem ovliňujícím schopnost klasifikace sítě je počet neuronů skryté vrstvy. S odkazem na 4.1.1 roste se zvyšováním počtu vah sítě také požadavek na velikost trénovací množiny. 40 000 trénovacích signálů, kde průměrný počet rámců jednoho signálu je 500, poskytuje dostatečný počet trénovacích vzorů. Jednotlivé výsledky ukazuje tabulka 4.3.1.

Velikost trénovací množiny	Počet skrytých neuronů	Acc [%]	Čas	Počet iterací
40 000	2000	77.78	2 h 19 min	10
	4000	79.85	4 h 30 min	10
	6000	80.66	6 h 1 min	9
	8000	82.12	8 h 52 min	10
	10000	82.79	11 h 5 min	10
	12000	82.29	12 h 3 min	9

Tabulka 4.3.1 Vliv velikosti skryté vrstvy.

Trénink byl parelelizován na 8-mi vláknech. Trénování probíhalo s rychlostí učení 0.008 a půlícím faktorem 0.5. Hodnota přesnosti klasifikace se ustálila na 82 %, další na-

vyšování velikosti skryté vrstvy již nepřinášelo zlepšení úspěšnosti. Grafické znázornění získaných výsledků ukazuje 4.3.1.



Obrázek 4.3.1 Grafické znázornění vlivu velikosti skryté vrstvy.

Časová náročnost rostla rostla se zvětšující se skrytou vrstvou téměř lineárně. V nejdelším případě trénování trvalo 12 h 3 min. Pro porovnání uvádím, že pro stejnou trénovací množinu a parametry trénování, trvalo při paralelizaci na 4 vláknech trénování sítě se 4000 skrytými neurony 5 h 32 m, což je o 62 minut více, než v případě 8 vláken.

4.4 Paralelizace trénování

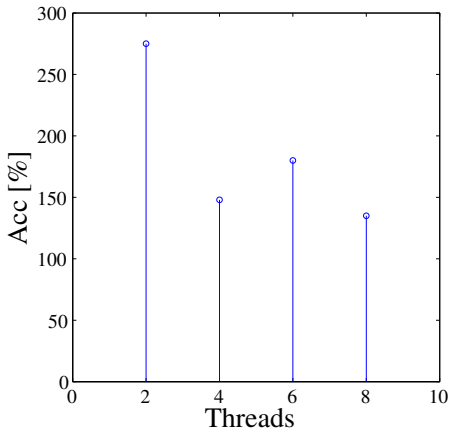
Hlavní výhoda nástroje TNet spočívá v možnosti paralelizace trénování, což značně redukuje dobu výpočtů. Trénování sítě probíhalo na uzlu magi217, který je osazen 4 jádrovým procesorem s frekvencí 3,5 GHz. Celkový počet vláken procesoru je 8. Tabulka 4.5.1 uvadí dosažené výsledky.

Velikost trénovací množiny	Skryté neurony	Počet vláken	Čas
40 000	2000	2	4 h 35 min
		4	2 h 28 min
		6	3 h 0 min
		8	2 h 15 min

Tabulka 4.4.1 Počet vláken při paralelním trénování.

Zdvojnásobení počtu vláken znamenalo zkrácení doby, potřebné k natrénování téměř o polovinu. Pokud byl počet vláken zvolen větší, než je počet fyzických jader procesoru, došlo opět k nárůstu trénovacího času. V tomto případě sice jedno jádro zpracovává dvě vlákna, ale takovým způsobem, že mezi nimi přepíná. Dále je nutné přihlédnou k možnostem daného hardwaru. Při vícevláknovém trénování jsou kladeny velké nároky na datový tok mezi jednotlivými komponentami počítače, zejména pak při čtení dat

z disku. Pro zrychlení výpočtů je rovněž nutné nevyčítat data ze síťového disku ale mít je nakopírovány na používaném počítači.



Obrázek 4.4.1 Graf závislosti doby trénování na počtu vláken.

4.5 Rychlost učení

Další parametrem, který má vliv na proces a výsledek trénování je rychlost učení (learning rate). Příliš malá hodnota vede k pomalé konvergenci chyby klasifikace a zvýšení doby trénování. Naopak příliš velká hodnota může zapříčinit přechod přes hledané minimum funkce a oscilaci v průběhu chybové funkce. Ověřená a používaná hodnota je 0.008. Zjišťoval jsem vliv změny na obě strany od této používané hodnoty.

Velikost trénovací množiny	Skryté neurony	Rychlost učení	Acc [%]	Čas	Počet iterací
40 000	2000	0.0008	74.81	2 h 2 min	9
		0.008	77.55	2 h 15 min	10
		0.08	36.97	7 h 33 min	13

Tabulka 4.5.1 Vliv rychlosti učení.

Jak je z hodnot vidět. Rychlost učení 0.08 je již příliš velká. Učení trvalo dále, než jak je tomu v případě hodnoty 0.008 a konvergence chyby nebyla optimální, proto bylo dosaženo úspěšnosti pouze 36.97 %. Došlo navíc k přetrénování sítě. Hodnota 0.0008 je naopak nedostačující. Učení trvalo kratší dobu, ovšem dosažená úspěšnost byla menší. Z výsledků vyplývá, že hodnota 0.008 je skutečně optimální.

5 Závěr

MLP sítě prokázaly velmi vhodné vlastnosti v řečových aplikacích, zejména pak v úloze extrakce příznaků na bázi dlouhých časových trajektorií, kde množství vstupních dat nelze přímo modelovat pomocí HMM, a proto se neuronové sítě používají jako mapující nástroj, jehož výstupem jsou příznaky odpovídající aposteriorním pravděpodobnostem jednotlivých hlásek. Proto k jejich implementaci vznikla řada nástrojů, z nichž jedním z posledních je TNet, který byl základem této práce, jejíž cíle je možné shrnout do dvou základních bodů:

1. Osvojit si práci s TNet toolkitem a poskytnout informace pro jeho používání.

   TNet zastřešuje kompletní realizaci MLP sítí, tedy jejich trénování, testování a normalizaci vstupních dat. K tomuto účelu obsahuje nástroje TNet (trénování), TNorm (normalizace), TFeaCat (testování), TJoiner (slučování dat) a TSegmenter(rozdělování dat). Dále jsou spolu s těmi hlavními nástroji distribuovány jednotlivé sktripty, které uživateli umožňují snáze realizovat požadované aplikace a generovat potřebné soubory podle konvencí zavedených pro používání hlavních nástrojů. Tuto skutečnost lze hodnotit jako velký přínos při osvojování práce s TNetem.

   Hlavní výhodou tohoto nástroje je paralelizace trénování sítě, což umožňuje implementaci a zkoumání vlastností rozsáhlých sítí a zpracování velkého počtu trénovacích dat. Bylo tak možné analyzovat vliv jednotlivých parametrů na úspěšnost klasifikace a dobu trénování, jelikož doba chodu jednotlivých výpočtů v nejdelším případě dosáhla řádu desítek hodin.

   Jednotlivé informace a zkušenosti získané vypracováním této práce umožní další aplikace nástrojů TNet toolkitu. Cílem bylo osvojit si základní postupy pro správný chod jednotlivých nástrojů, a proto zkoumaní dalších vlastností a vhodných parametrů při učení sítě k dosažení co nejlepší úspěšnosti klasifikace je námětem pro další práce, stejně tak jako optimalizace jednotlivých parametrů z hlediska doby trénování nebo implementace dalších typů sítí.

2. Prostřednictvím tohoto nástroje implementovat fonémový rozponávač, jehož výstup slouží k dalšímu zpracování.

   Výstupem realizovaného fonémového rozpoznávače jsou aposteriorní pravděpodobnosti jednotlivých hlásek, které jsou často dále zpracovávány a vyhodnocovány. V konkrétní aplikaci jsou tyto pravděpodobnosti dále vyhlazovány prostřednictvím HMM. Nejlepšího výsledku bylo dosaženo pro síť s 10 000 skrytými neurony, trénovanou na 40 000 signálech. V tomto případě byla úspěšnost klasifikace 82,79 % (17.21% PER, definovaná jako 1-ACC) a trénování trvalo 11 hodin a 5 minut. Tento výsledek je možné považovat za velmi uspokojivý.

   Pro srovnání lze uvést výsledek práce [6], kde bylo pro TRAP parametrizaci dosahováno kolem 30 % PER. V [19] je referován výsledek 24,84 % PER. Je ovšem nutné podotknout, že tyto výsledky jsou zde uváděny pouze jako příklad, jelikož v byly dosaženy pro odlišné konfigurace a parametry sítě.

   Primární využití výstupních pravděpodobností byla jejich aplikace v TANDEM architektuře rozpoznávače na bázi HMM, který byl předmětem paralelně zpracovávané práce Aleše Bricha [7]. Realizace celého hybridního ANN/HMM rozpoznávače je příliš komplexní, s ohledem na požadovaný rozsah jedné bakalářské práce, a proto jeho realizace byla rozdělena do dvou souběžných prací.

Svými vlastnostmi je TNet určen téměř výhradně pro aplikace zpracování řeči. Proto je jeho kompatibilita s jinými nástroji, zejména pak s HTK, velkým přínosem. Umožňuje tedy, právě v případě HTK, snadné propojení standardních statistických metod a umělých neuronových sítí. Výsledky práce dokazují, že MLP sítě mají v oblasti rozpoznávání řeči značný potenciál, a proto je vhodné si osvojit práci s nástroji, které implementaci takových sítí umožňují.

Literatura

[1] Josef Psutka et al. *Mluvíme s počítačem česky*. Academia, 2006.

[2] Jan Uhlíř et al. *Technologie hlasových komunikací*. Praha: Nakladatelství ČVUT, 2007.

[3] Christopher M. Bishop. *Neural Networks for Patter Recognition*. Oxford University Press, 1995.

[4] HTK Toolkit. *http://htk.eng.cam.ac.uk/*.

[5] Fredric M. Ham a Ivica Kostanic. *Principles of Neurocomputing for Science and Engineering*. McGraw-Hill Science/Engineering/Math, 2001.

[6] Petr Schwarz. "Phoneme Recognition Based on Long Temporal Context". Dis. VUT FIT, 2008.

[7] Aleš Brich. *Implementace rozpoznávače řeči na bázi TANDEM architektury*. 2014.

[8] Lawrence R. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE*. IEEE, 1989, s. 257–286.

[9] Vojtěch Ondráček. "Robustní parametrizace řeči na bázi časových trajektorií". Dis. České vysoké učení technické v Praze, 2012.

[10] QuickNet Toolkit. *http://www.icsi.berkeley.edu/Speech/qn.html*.

[11] Steve Young et al. *The HTK Book*. Cambridge University Engineering Department, 2009.

[12] TNet Toolkit. *http://speech.fit.vutbr.cz/software/neural-network-trainer-tnet*.

[13] Petr Schwarz, Pavel Matějka a Jan Černocký. "Recognition of Phoneme Strings using TRAP Technique". In: *Proceedings of 8th International Conference Eurospeech*. Sv. 2003. 9. Geneve, CH, 2003, s. 1–4.

[14] Petr Schwarz, Pavel Matějka a Jan Černocký. "Towards Lower Error Rates In Phoneme Recognition". In: *Lecture Notes in Computer Science* 2004.3206 (2004), s. 465–472.

[15] Hynek Hermansky a Sangita Sharma. "Temporal patterns (TRAPS) in ASR of noisy speech". In: *in Proc. ICASSP*. 1999, s. 289–292.

[16] Stanislav Kontár. "Parallel training of neural networks for speech recognition". In: *Proc. 12th International Conference on Soft Computing MENDEL'06*. Brno, CZ, 2006, s. 6.

[17] Karel Veselý, Lukáš Burget a František Grézl. "Parallel Training of Neural Networks for Speech Recognition". In: *Prof. Text, Speech and Dialogue 2010*. Sv. 2010. 9. Brno, CZ, 2010, s. 439–446.

[18] Jana Tučková. *Vybrané aplikace umělých neuronových sítí při zpracování signálů*. 978-80-01-04229-8. České vysoké učení technické v Praze, 2009.

[19] P. Schwarz, P. Matejka a J. Cernocky. "Hierarchical Structures of Neural Networks for Phoneme Recognition". In: *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*. 2006, s. I–I.