TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

# Jednoduchý systém pro hlasové ovládání počítače

## Bakalářská práce

*Studijní program:* B2646 – Informační technologie
*Studijní obor:* 1802R007 – Informační technologie

*Autor práce:* **Lukáš Böhm**
*Vedoucí práce:* prof. Ing. Jan Nouza, CSc.

Liberec 2020

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

## Zadání bakalářské práce

# Jednoduchý systém pro hlasové ovládání počítače

*Jméno a příjmení:* **Lukáš Böhm**
*Osobní číslo:* M17000070
*Studijní program:* B2646 Informační technologie
*Studijní obor:* Informační technologie
*Zadávající katedra:* Ústav informačních technologií a elektroniky
*Akademický rok:* **2019/2020**

### Zásady pro vypracování:

1. Cílem práce je vytvořit jednoduchý systém pro ovládání počítače pomocí cca 150 – 200 hlasových povelů.
2. Seznamte se se základy automatického rozpoznávání řeči a s různými přístupy k hlasovému ovládání počítače.
3. Sestavte vhodný seznam povelů, jimiž bude možné ovládat základní uživatelské akce prováděné jinak pomocí klávesnice a myši. Vytvořte dostatečně reprezentativní soubor nahrávek obsahujících tyto povely namluvené cca 30 ? 40 osobami. Nahrávky rozdělte na trénovací sadu a testovací sadu (data od 10 osob).
4. Implementujte (nejlépe v prostředí MATLAB) různé metody (od nejjednodušších až po aplikace neuronových sítí) pro rozpoznávání těchto povelů a vyhodnoťte je na testovací sadě z hlediska úspěšnosti a doby rozpoznávání.
5. S využitím nejlepší metody vytvořte jednoduchý ukázkový program pro skutečné hlasové ovládání PC se systémem MS Windows.

| | |
|---|---|
| *Rozsah grafických prací:* | podle potřeby dokumentace |
| *Rozsah pracovní zprávy:* | 30-40 stran |
| *Forma zpracování práce:* | tištěná/elektronická |
| *Jazyk práce:* | Čeština |

**Seznam odborné literatury:**


[1] Nouza, J., Koldovský Z., Vích. R.: Řeč a počítač – sborník článků. TUL, 2011.

[2] Nouza J., Červa P.: Hlasové systémy MyVoice a MyDictate pro handicapované uživatele počítačů. Sborník konference Handicap 2007. TUL, 2007.

[3] Nouza J., Červa P.: Design and Development of Voice Controlled Aids for Motor-Handicapped Persons, In: Conference of the International Speech Communication Association (Interspeech 2007), pp. 2521 – 2524, Antwerp, August 2007


| | |
|---|---|
| *Vedoucí práce:* | prof. Ing. Jan Nouza, CSc.<br>Ústav informačních technologií a elektroniky |
| *Datum zadání práce:* | 9. října 2019 |
| *Předpokládaný termín odevzdání:* | 18. května 2020 |

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

L.S.

Technická univerzita v Liberci
-18-

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 18. října 2019

# Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že texty tištěné verze práce a elektronické verze práce vložené do IS STAG se shodují.

31. 5. 2020

Lukáš Böhm

# Jednoduchý systém pro hlasové ovládání počítače

## Abstrakt

Tato bakalářská práce se zabývá problematikou rozpoznávání řeči, konkrétně izolovaných slov. V práci je popsáno několik metod využívaných pro implementaci rozpoznávacíh systémů. Nejprve jsou vysvětleny vzdálenostní metody lineární časové transformace LTW a metoda dynamického borcení času DTW. Dále práce zkoumá metodu skrytých markovských modelů HMM, pracujících se statistickými modely. Poslední popsanou metodou jsou umělé neuronové sítě a jejich využití v dané problematice.

Praktickou částí práce je vyhotovení ukázkové aplikace pro jednoduché ovládání operačního systému Microsoft Windows na základě nejúspěšnější z metod. Tomu předchází kromě implementace metod také vytvoření slovníku povelů pro ovládání, nasbírání dostatečného množství nahrávek pro každý z povelů a jejich následná parametrizace.

**Klíčová slova:** Audio signál, LTW, DTW, HMM, ANN, časová transformace, skryté markovské modely, umělé neuronové sítě, konvoluční sítě.

# Simple voice control system

## Abstract

This bachelor thesis deals with the issue of speech recognition, specifically isolated words. The work describes several methods used for the implementation of recognition systems. First, the distance methods linear time transformation LTW and the method of Dynamic Time Warping DTW are explained. Furthermore, the work examines the method of Hidden Markov Models HMM, working with statistical models. The last described method are artificial neural networks and their use in the issue.

The practical part of the work is the creation of a sample application for simple control of the Microsoft Windows operating system based on the most successful of the methods. This is preceded not only by the implementation of methods, but also by the creation of a dictionary of commands for OS control, the collection of a sufficient number of recordings for each of the commands and their parameterization.

**Keywords:** Audi signal, LTW, DTW, ANN, time transformation, hidden markov models, artificial neural network, convolution network.

# Poděkování

Rád bych poděkoval vedoucímu této práce, profesoru Janu Nouzovi, za cenné rady a vynaložený čas.

# Obsah

# Seznam obrázků

# Seznam tabulek



# Seznam zkratek

| | |
|---|---|
| **TUL** | Technická univerzita v Liberci |
| **FM** | Fakulta mechatroniky, informatiky a mezioborových studií Technické univerzity v Liberci |
| **LTW** | Linear Time Warping |
| **DTW** | Dynamic Time Warping |
| **ANN** | Artificial Neural Network |
| **Inf** | Infinity |
| **HTK** | Hidden Markov Model Toolkit |
| **HZ** | Hertz |
| **ZCR** | Zero-Crossing Rate |
| **FT** | Fourierova Transformace |
| **MFCC** | Mel-Frequence cepstral coeficients |
| **GUI** | Graphic User Interface |
| **API** | Application Programming Interface |
| **OS** | Operační systém |
| **MS** | Microsoft |

# 1 Úvod

V posledních desítkách let je nepřehlédnutelný prudký nárůst technologií a jejich adaptace do každodenních životů lidí. Dlouhá léta byly novinky ve světě technologií brány širokou veřejností spíše jako hračka než schopný pomocník při práci. Až s nárustem výpočetního výkonu počítačů, kdy se objevily systémy, které dokáží plně nahradit lidskou práci, začala veřejnost vnímat nové technologie jako věc nezbytnou. Porovnáním současných mobilních telefonů s počítači na palubě Apollo 11 z roku 1969 lze zjistit, že většina lidí nosí v kapse počítač až 100 000x výkonnější než ten, který ovládal vesmírné plavidlo.

Právě výkonnost současných systémů umožňuje řešit výpočetně náročné operace, jako je strojové učení. Přitom už v roce 1949 Donald Hebb poprvé zmínil principy strojového učení ve jedné ze svých knih.[1] O pouhých pár let později vědec Arthur Samuel použil první chytrý systém ve své počítačové variantě hry dáma. [2] Výzkum různých metod strojového učení pokračoval, nicméně v té době chyběl právě výpočetní výkon, aby byla možná reálná implementace těchto algoritmů.

V současné době se můžeme setkat se systémy strojového učení, jako je rozpoznávání řeči, téměř v každém moderním zařízení. Většina z nich je založena na neuronových sítích, díky velkému množství dat, jež jsou veliké společnosti schopny nasbírat. Nicméně, ne vždy máme k dispozici tisíce až miliony vzorků dat pro trénování neuronových sítí. Proto se používaly a stále používají i jiné metody, které často dosahují při menším množství dat lepších výsledků.

Cílem této práce je prozkoumat různé přístupy k vytvoření systému schopného rozpoznávat jednoduché povely. Mezi metody zkoumané v této práci patří například metoda lineárního borcení času (LTW) porovnávající vzdálenost mezi neznámým slovem a referenčním vzorem. Metoda dynamického borcení času (DTW) umožňující vedle porovnání vzdálenosti i lokální zkrácení, či prodloužení, a tím pádem se stává robustnější vůči přízvukům a rytmizaci slov, nebo metoda skrytých markovských modelů (HMM), jež místo vzdáleností pracuje s pravděpodobnostním modelem.

V této práci budou tyto konkrétní metody, včetně neuronových sítí, popsány a aplikovány na dataset získaný pro tento konkrétní projekt. Dataset tvoří 33 mluvčích se 176 příkazy, které byly vybrány, pro ovládání počítače. Finálním produktem práce je ukázkový programu, pomocí kterého budeme schopni základním způsobem ovládat operační systém Microsoft Windows za použití nejúspěšnější z metod.

# 2 Rozpoznávání hlasových povelů

Rozpoznávání řeči je schopnost počítače nebo programu identifikovat slova v mluveném jazyce a převést je do podoby zpracovatelné počítačem. Omezení rozpoznávače spočívá v předem daném souboru slov nebo frází, které je systém schopen rozpoznat a mluvené slovo k nim přiřadit.

Problematiku rozpoznávání řeči je možné kategorizovat do několika základních skupin. První rozdělení je rozpoznávání spojité řeči, nebo rozpoznávání izolovaných slov. Druhým možným rozdělením je rozpoznávání závislé a nezávislé na konkrétním mluvčím. Třetí skupinou je hlučnost okolního prostředí. Je možné implementovat rozpoznávač, který bude schopen odfiltrovat rušivé elementy okolí, nebo musí očekávat, že koncový uživatel bude aplikaci využívat v tichém prostředí. V této práci bude snaha o implementaci systému pracujího s izolovanými slovy řečenými v tichém prostředí. Zároveň bude systém nezávislý (speaker independence), díky rozdělení řečníků do testovací a trénovací skupiny.

Aby bylo možné rozpoznávač vytvořit, je kromě implementace konkrétní metody nutné získat co nejvíce trénovacích dat a data zparametrizovat do vhodné podoby. Jednodušší metody porovnávají podobnost signálu vyřčeného slova se signály nasbíraných dat. Metody složitějšího charakteru, jako jsou neuronové sítě nebo metoda skrytých markovských modelů, využívají data ke skutečnému trénování modelu, kterým jsou následně hledaná slova vyhodnocována a přiřazována ke slovům ve slovníku.(Příloha A.2)

Kvalita rozpoznávače se udává dvěma veličinami. Přesnost a rychlost. Přesnost se určuje mírou chybovosti. Rychlost udává čas, za který rozpoznávač vyhodnotí, o jaké slovo se jedná. To zpravidla závisí na algoritmu využité rozpoznávací metody. Vyšší přesnosti rozpoznávače s konkrétní rozpoznávací metodou lze docílit nasbíráním většího množství dat či použitím jiných příznaků, kterými jsou data reprezentována. Aby se zvýšila rychlost, je třeba konkrétní metodu co nejlépe optimalizovat.

Každá z používaných metod v tomto projektu byla v minulosti intenzivně zkoumána, proto není účelem jejich zdokonalování. Cílem práce je vybrat tu nejvhodnější kombinaci metody a parametrů pro nasbíraná data. Metody prozkoumat, implementovat a nakonec dosáhnout co nejlepších výsledků při rozpoznávání povelů.

## 2.1 Linární časová transformace LTW

Metoda lineární časové transformace, volně přeloženo z anglického názvu ”linear time warping”, dále LTW. LTW funguje na principu porovnání dvou časových prů-

běhů (v tomto případě zvukových signálů). Ve slovníku máme pro každý z příkazů jeho referenční vzor. Ten je ve tvaru matice příznaků F x P [odkaz Param], jež jsme získali parametrizací signálu. Neznámé slovo porovnáme se všemi referenčními vzory. Výsledkem klasifikace jsou čísla udávající vzdálenost mezi neznámým slovem a konkrétním příkazem. Příkaz z našeho slovníku, u kterého vyšla vzdálenost nejmenší, je tím příkazem, k němuž se neznámé slovo přiřadí. Problémem je fakt, že většinou časové průběhy nejsou stejně dlouhé. V tomto případě, je to dané tím, že každý řečník slovo vyslovuje trochu jinak, tím pádem má vždy jinou délku a matice příznaků má rozdílné rozměry. Tento problém se řeší zásahem do časové osy lokálním zkracováním, či prodlužováním.[3]

### 2.1.1 Princip LTW

Mějme slovo X a slovo Y. Obě slova jsou reprezentována příznakovým vektorem velikosti F x P, kde F je počet framů a P počet příznaků. Pro každý frame testovaného slova

$$X = [x_1, x_2, x_3, ..., x_i, ..., x_I]$$

a referenčního slova

$$Y = [y_1, y_2, y_3, ..., y_j, ..., y_J]$$

mějme právě jeden příznak. Pokud by byly počty framů N a M shodné, vzdálenost lze jednoduše vypočítat součtem vzdáleností mezi framy na stejné pozici. V případě že $J \neq I$ je třeba posloupnost slova Y modifikovat tak, aby jeho délka byla J. Pokud je $J < I$ některé framy slova Y je třeba vynechat. Nebo v případě $J > I$ některé framy zopakovat. Tento princip je aplikovatelný pomocí lineární transformační funkce:

$$f(i) = Int\left[\frac{J-1}{I-1} \cdot (i-1) + 1 + 0.5\right] \tag{2.1}$$

Tato funkce představuje přímku (fukci) procházející body $(1, 1)$ a $(I, J)$. Ostatní body lze získat dle vztahu:

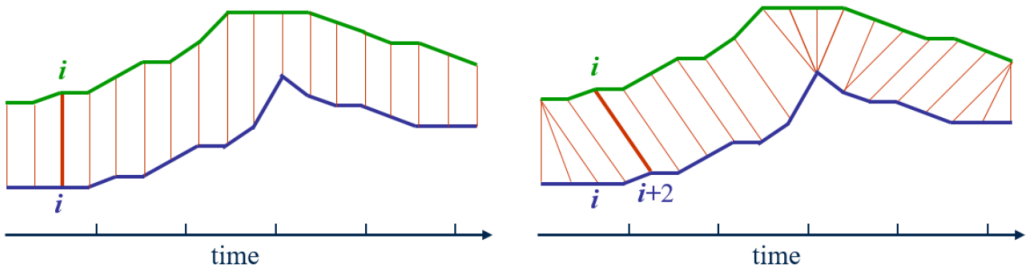
$$D(X, Y) = \sum_{i=1}^{I} d(x_i, y_{f(i)}) \tag{2.2}$$

Horizontální směr přímky představuje zopakování framů a tím celkové prodloužení referncního slova Y, kde $D(x_i, y_j)$ udává lokální vzdálenost mezi určitými framy. Vzdálenost lze vypočítat různými způsoby, zde se používá Euklidovská vzdálenost:

$$d(x_i, y_j) = \sqrt{\sum_{p=1}^{P} (x_{ip} - y_{jp})^2} \tag{2.3}$$

## 2.2 Dynamická časová transformace DTW

Lineární časová transformace (LTW) řeší různou délku slov. Při rozpoznávání řeči se ovšem lze setkat i s dalšími problémy. Může to být časový posun, rozdílná rychlost, lokální akcelerace, nebo odlišná rytmizace slabik, či slov. Pro vyřešení těchto problému už lineární transformace nestačí a je třeba využít dynamického přístupu. DTW je sada algoritmů měřící podobnost dvou signálů, nebo jiných dat reprezentovaných posloupností na časové ose stejně, jako je tomu u metody LTW. Na základě konkrétních podmínek nám DTW umožňuje nelineárně přizpůsobit signály pomocí takzvané bortivé funkce a tím vytvořit transformační cesty, pomocí nichž se zjistí vzdálenost mezi dvěma signály. Přizpůsobením rozumíme lokální zkrácení, nebo prodloužení testovaného signálu, či testovaného i referenčního signálu.




**Obrázek 2.1:** Znázornění lienárního a nelineárního přiřazení dvou signálů [4]

### 2.2.1 Princip DTW

Mějme stejný neznámý signál $X$ a referenční signál $Y$ jako u metody LTW. Algoritmus hledá optimální cestu $(j = f(i))$ v matici A(I x J), která minimalizuje funkci D, udávající všechny celkové vzdálenosti mezi signály X a Y.

$$D(X,Y) = \sum_{i=1}^{I} d(x_i, y_{f(i)}) \tag{2.4}$$

Kde $d(x_i, y_{f(i)})$ reprezentuje lokální vzdálenost mezi i-tým vektorem testovaného signálu a j-tým $(j = f(i))$ vektorem referenčního signálu. Vzdálenosti lze vypočítat pomocí euklidovské vzdálenosti. (vztah 2.3).

Ze vztahu $(j = f(i))$ je možné vyjádřit optimální cestu pomocí funkčního vztahu mezi i a j. Pokud by se uvažovala pouze základní pravidla DTW, jimiž jsou okrajové podmínky $p_1 = (1,1)$, $p_k = (I,J)$ kde $p_1$ je první bod cesty a $p_k$ poslední a podmínky, které nám zakazují obracet časový tok, či ho přerušovat. Získalo by se velké množství cest, což s sebou nese problém s vysokou paměťovou náročností. Při složitosti algoritmu $O(IxJ) => O(N^2)$, by se dostavila i enormní výpočetní náročnost. Tomu lze částěčně předejít použitím Ikaturových podmínek (vztah 2.5).

$$f(i) = j \iff f(i-1) = j \wedge f(i-2) \neq j(a)$$
$$f(i-1) = j-1(b)$$
$$f(i-1) = j-2(c) \tag{2.5}$$

I za použití stanovených podmínek by bylo stále vygenerováno velké množství transformačních cest, ze kterých by bylo třeba vybrat tu nejkratší. Proto metoda DTW, jak již z názvu vyplývá, používá způsoby dynamického programování. To umožňuje vyčíslování vzdáleností cest, již při jejich tvoření a to pomocí rekurzivního vztahu: [3]

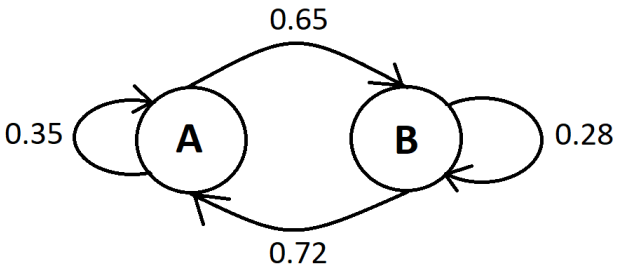
$$A(i,j) = Min[A(i-1,j), A(i-1,j-1), A(i-1,j-2)] + d(x_i, y_i) \tag{2.6}$$

Inicializujme tedy matici A o velikosti $IxJ$ a o hodnotách Inf, kde $A(i,j)$ je akumulovaná vzdálenost v bodě $(i,j)$ Počáteční bod $(1,1)$ se získá výpočtem vzdálenosti $d(x_1, y_1)$. V následujících iteracích se postupuje maticí P při plnění Ikaturových podmínek a použití dynamického principu programování (vztah 2.6). Bod $(i,j)$ tedy vypočteme jako součet vzdálenosti $d(x_i, y_j)$ a nejmenší z hodnot vzdáleností na pozicích $(i-1,j)$, $(i-1,j-1)$, nebo $(i-1,j-2)$. Po kompletní iteraci maticí A, bude se hodnota výsledné vzdálenosti nacházet v bodě $(I,J)$. Pokud by bylo cílem vizuální vyobrazení cest, muselo by se inicializovat pole ukazatelů, kde by se pro každý bod nacházela informace o j-té souřadnici předchozího bodu s nejnižší vypočtenou vzdáleností.[3]

## 2.3 Metoda skrytých markovských modelů HMM

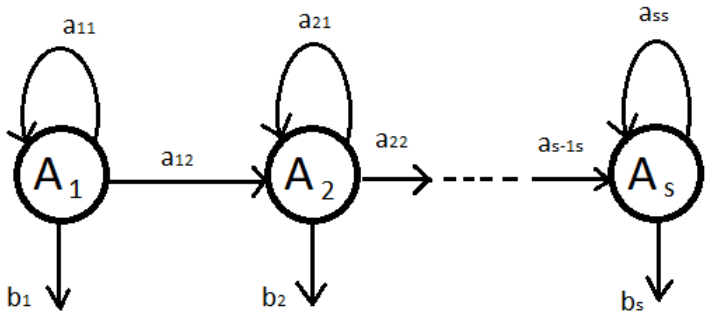
Předchozí metody vždy porovnávaly hledané slovo s referenčními vzory. Aby dosahovaly větších úspěchů, bylo třeba získat více referenčních vzorů od každého slova ze slovníku, vytvořeného pro konkrétní aplikaci. Z toho ovšem vyplývá problém, kterým je výpočetní náročnost. Každým novým refernčním vzorem přibývá výpočet v podobě porovnání hledaného slova a referenčního vzoru. Nápadem pro vyřešení tohoto problému bylo vytvoření univerzálního vzoru, zprůměrováním všech referenčních vzorů reprezentujících stejné slovo. Tento postup nepřinesl příliš velký úspěch.

Řešením se ukázaly být metody ruského matematika Andreje Andrejeviče Markova, využívající stejnojmenné Markovovy řetězce (obr.2.2). Kdy namísto posloupnosti framů reprezentujících slovo se využívají takzvané stavy. Hlavní myšlenkou je, že pravděpodobnost přechodu z akutálního stavu do stavu jiného závisí pouze na aktuálním stavu. Na obrázku 2.2 je model obashující stav A a stav B, kde každý ze stavů nese informaci o pravděpodobnosti, zda v konkrétním stavu setrvá, nebo se přesune do stavu jiného. Stav A tedy má pravděpodobnost 35% , že model ve stavu A zůstane a pravděpodobnost 65% , že se přesune do stavu B a stav B má pravděpodobnost 28% , že model zůstane ve stavu B a 72% , že dojde k přesunu do stavu A. Informace o tom, jak se systém do aktuálního stavu dostal neexistuje.




**Obrázek 2.2:** Markovův pravděpodobností dvoustavový systém

Metoda markovských modelů využívá principu markovských řetězců. Stav nese informaci o specifických vlastnostech těch framů, které reprezentuje. Obecně a také v tomto případě bude využito normálního (Gaussova) rozdělení, v jednopřízankovém případě daného rozptylem $\sigma_s$ a střední hodnotou $\mu_s^2$. V ostatních případech dané kovariační maticí $\Sigma$ specifikovanou hodnotami vektorů přiřazeným během trénování ke konkrétnímu stavu s.




**Obrázek 2.3:** Levo-pravý model slova s $S$ stavy, výstupními pravděpodobnostmi $b$ a pravděpodobnostmi přechodů $a$

### 2.3.1 HTK Toolkit

The Hidden Markov Model Toolkit (HTK) je, jak už název napovídá, sada nástrojů určená k implementaci metody skrytých markovských modelů (HMM). HTK se primárně používá ke strojovému rozpoznávání řeči, nicméně jeho využití zasahuje až do biochemických metod DNA sekvenace. Toolkit se skládá ze sady knihoven a modulů napsaných v jazyce C. Tyto nástroje slouží pro analýzu signálu, trénink HMM, následné testování a vyhodnocování výsledků. Software podporuje jak HMM s rozdělením Gaussovským, tak rozdělením diskrétním. HTK byl vyvinut laboratoří Machine Intelligence Laboratory na Cambridgeské Univerzitě, katedře Inženýrství.

A první verze se objevili už v roce 1993. Verze softwaru používaného v této práci je z roku 2016 a nese označení 3.5.[1]

### 2.3.2 Princip HMM

Pricnipem metody HMM je najít takový model, který by s nejvyšší pravděpodobností vygeneroval hledané slovo.[5] Lineární uspořádání S stavů, tvoří typický model slova. Zda model v určitém stavu setrvá, nebo se přesune do stavu následujícího uvádí přechodová matice $A(ss)$, která udává s jakou pravděpodobností se model přesune ze stavu $s$ v čase $t$ do stavu $s+1$ v čase $t+1$.

Na příkladu markova řetězce se stavy A a B (obr. 2.2) si lze všimnout, že se model vrací ze stavu B zpět do stavu A. Při implementaci HMM se používá levo-pravý model, který umožňuje přesun pouze ve směru časového toku, tím pádem zleva doprava.(obr.2.3). Součet pravděpodobnosti setrvání v aktálním stavu s pravděpodobností, že dojde k přechodu se rovná 1. Zpravidla je pravěpodobnost setrvání v současném stavu vyšší, z důvodu většího počtů framů slov, než stavů modelu. Dalším parametrem modelu je pravděpodobnostní hustota výstupních vektorů modelu $B = [b_s(x)]_{s=1}^{S}$, která uvadí zda frame neznámého slova X přísluší konkrétnímu stavu.[6] Model M je možné zapsat $M = (A, B)$

Namísto referenčních vzorů, jako tomu bylo v předchozích kapitolách, se nyní používají modely, které se liší pravděpodobnostmi přechodů mezi stavy a výstupní funkcí. Aby bylo možné klasifikitovat, musí se v první řadě systém natrénovat. Trénováním se rozumí zíkání matic přechodu A a výstupních funkcí B pro konkrétní model reprezentující všehny nahrávky jednoho povelu. Bude tedy vytvořeno tolik modelů, kolik slov obsahuje slovník. Jedná se o výpočetně nejsložitější část této metody. Problémém při trénování je inicializace modelů, zde budeme brát nejjednodušší metodu, kdy jsou framy všech realizací daného slova přidělovány stavům rovnoměrně lineárně. Po inicializaci je možné vypočítat první odhady parametrů modelu. Dále pomocí Viterbiho algoritmu lze nalést nové přiřazení framů ke stavům a znovu přepočítat parametry modelu. Tento proces se opakuje dokud se pomocí testování jeho skóre zlepšuje.

Výpočet parametrů začíná střední hondotou

$$\bar{\mu}_s = \frac{\sum_{t=1}^{T} L_s(t) x_t}{\sum_{t=1}^{T} L_s(t)} \tag{2.7}$$

váhový koeficient

$$\bar{c}_s = \frac{\sum_{t=1}^{T} L_s(t)}{\sum_{t=1}^{T} L_s(t)} \tag{2.8}$$

a kovariační maticí

$$\Sigma_s = \frac{\sum_{t=1}^{T} L_s(t)(x_t - \bar{\mu}_s)(x_t - \bar{\mu}_s)^T}{\sum_{t=1}^{T} L_s(t)} \tag{2.9}$$

[1] odkaz na stránky HTK toolkitu a jeho verze http://htk.eng.cam.ac.uk/

kde $L_s(t)$ značí pravděpodobnost, že se model nachází ve stavu $s$ v čase $t$. Po vypočtení Kovariační matice $\Sigma$ lze vypočítat hodnotu výstupní funkce stavů pomocí vztahu:

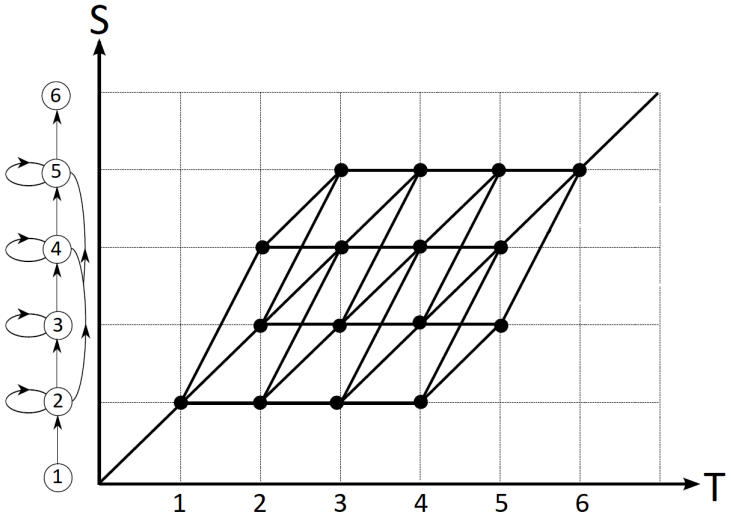
$$b_s(x) = \frac{1}{\sqrt{2\pi}^p det\Sigma_s} exp\left[-\frac{1}{2}(x-\bar{x}_s)^T\Sigma_s^{-1}(x-\bar{x}_s)\right] \tag{2.10}$$

Klasifikace poté probíhá počítáním míry pravděpodobnosti, že neznáme slovo náleží některému z modelů. Mějme neznáme slovo reprezentované posloupností příznakových vektorů X, s-stavový model M(A,B) a vztah:

$$P(X,M) = Max_f\left[a_{f(0)(1)}\prod_{t=1}^{T} a_{f(t)f(t+i)}b_{f(t)}(x_t)\right] \tag{2.11}$$

Kde $a_{f(0)f(1)} = 1$. Vztah nám udává maximální z pravděpodobností, že hledané slovo X náleží modelu M ze slovníku natrénovaných modelů.[6] Klasifikace probíhá podobným způsobem jako u předchozích metod s referenčními cestami, s tím rozdílem, že zde nepřiřazujeme framy nezmámého slova k framům referenčních vzorů, ale framy neznámeho slova ke stavům modelu. Přičemž nehledáme nejkratší ze všech vzdáleností, ale nejvyšší z pravěpodobností všech přípustných přiřazení framu ke stavům modelu. Funkce $f(0)$ je počáteční stav a $f(T)$ koncový stav modelu. K výpočtu této pravděpodobnosti se využívá Viterbiho algoritmus.

Viterbiho algoritmus slouží k vyhledání nejpravěpodobnější skevence stavů M, která generovala neznámé slovo X. Jde o hledání maximálně pravděpodobné cesty modelem. Hledání sekvence s nejvyšší pravděpodobností si lze představit jako hledání cesty v prostoru *V(T x S)*.




**Obrázek 2.4:** Schéma cesty prostorem V kde tečky značí výstupní funkci $b_s(t)$ a tučné čáry přechodové pravděpodobnosti $a_s$ z času $t$ do času $t+1$

Cesta jako u předchozích metod začíná v bodě $(1,1)$ a končí v bodě $(T,S)$. Pravděpodobnost hledané sekvence lze spočítat rekurzivním algoritmem.[3]

$$V(t,s) = Max[a_{ss}V(t-1,s), a_{s-1s}V(t-1,s-1)]b_s(x_t) \tag{2.12}$$

kde $V(1,1) = b_1(x_1)$ $V(1,s) = -\infty$ pro $1 < s < S$ Maximální pravděpodobnost je poté dána vztahem:

$$V(T,S) = Max[V(T,s)a_{sS}] \tag{2.13}$$

S koncem procesu tedy platí $P(X,M) = V(T,S)$.

## 2.4 Umělé neuronové sítě ANN

Jak už z názvu vyplývá, umělé neuronové sítě jsou systémy inspirované biologickými neurony v mozku. Cílem těchto systému je poskládat umělé neurony tak, aby byly schopny zpracovávat informace stejně jako je tomu v hlavách savců. Aneb upevňovat spoje mezi neurony, kde je to žádoucí a naopak oslabovat tam, kde informace je nepřesná, neboli nežádoucí. Tento proces lze nazvat učení. Stejně jako je mozek schopný učit se na základě předchozích zkušeností a zpětné vazby, kterou po vykonání určitého úkonu získal. Stejný princip využívají umělé neuronové sítě v informatice. Jedná se o soubor výpočetních operací, které díky konkrétním architekturám jsou schopny pomocí zpětné propagace určité operace postupně optimalizovat a tím dosahovat lepších výsledků. Z toho je již patrná jedna z hlavních výhod neuronových sítí a tou je schopnost vyhodnocovat výsledky, bez předem konkrétně definovaných pravidel.

### 2.4.1 Obecný princip ANN

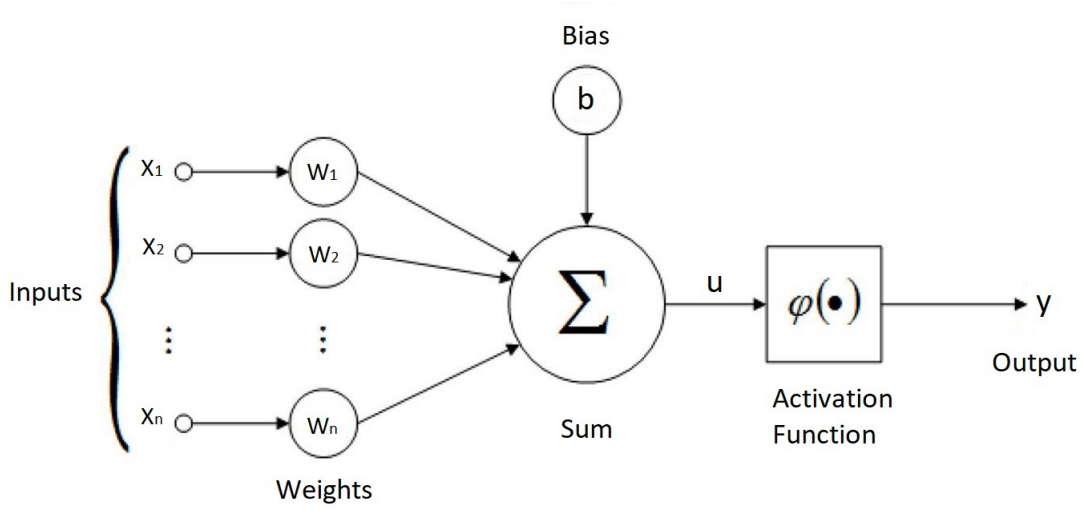
Základ každé sítě je umělý neuron (obr.2.5), který pomocí matematické funkce reprezentuje model biologického neuronu. Umělý neuron přijímá jeden nebo více vstupů, které násobí konkrétními váhami, poté je sčítá (skalární součin) a pomocí aktivační funkce generuje výstup. Podrobněji. Na vstupu do neuronu je vektor $X = [x_1, x_2, ..., x_n]$, neuron obashuje váhy o stejné velikosti jako vstup $W = [w_1, w_2..., w_n]$ a bias b. Neuron sečte všechny prvky vstupního signálu vynásobené jednotlivými vahami a přičte hodnotu biasu pomocí vztahu:

$$u = \sum_{i=1}^{N} x_i w_i + b \tag{2.14}$$

Následný výstup vstupuje do takzvané aktivační (přenosové) funkce. Jedná se o nelineární funkci převádějící vstup do diferencovatelného výstupu v určitém intervalu. Tím docílí, že se malá změna na vstupu projeví malou změnou na výstupu a naopak. Aktivačních funkcí je několik druhů (obr.2.6).

Mezi ty základní patří funkce lineární:

$$f(x) = x, , f(x) \in (-\infty, \infty) \tag{2.15}$$




**Obrázek 2.5:** Umělý neuron se vstupm X, váhami W, biasem b, aktivační funkcí $\varphi$ a výstupní hodnotou y

která není příliš používaná právě kvůli její linearitě. Dále funkce sigmoid:

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}, f(x) \in (0, 1) \tag{2.16}$$

funkce hyperbolický tangent:

$$f(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}, f(x) \in (-1, 1) \tag{2.17}$$

a funkce ReLu

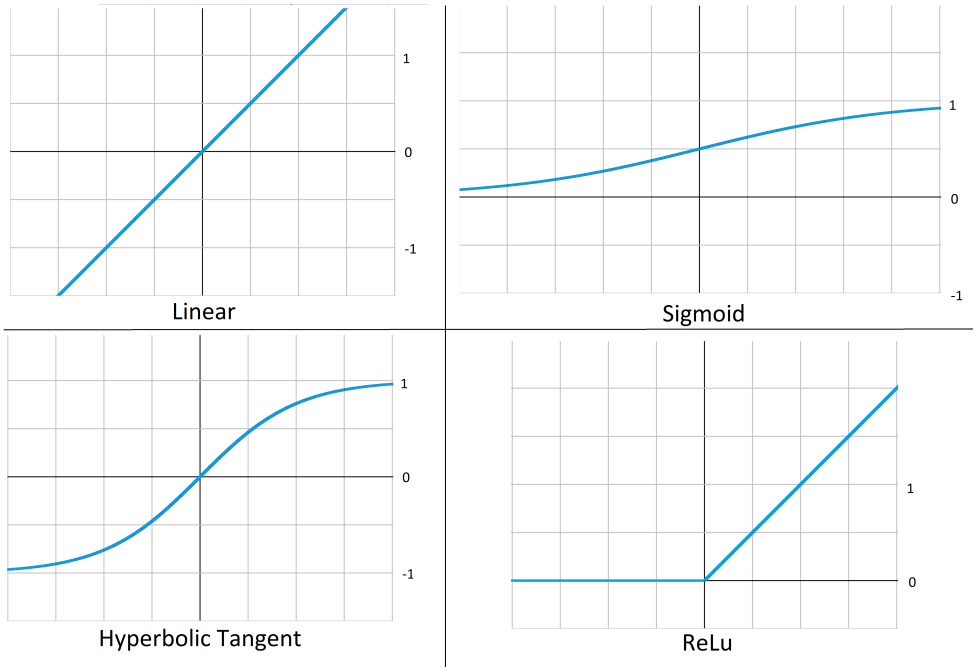
$$f(x) = max(0, x) \tag{2.18}$$

Umělá neuronová síť je kolekcí neuronů zapojených v acyklickém grafu. To znamená že výstupy určitých neuronů jsou vstupy neuronů v následující vrstvě sítě. Sítě jsou organizovány po vrstvách, které obsahují paralelně poskládané navzájem nepropojené neurony. Jedna vrstva může obsahovat jednotky, až tísice umělých neuronů. Pokud síť obsahuje více než dvě skryté vrstvy, nazýváme ji hlubokou neuronovou sítí(obr.2.8). "Základním" modelem je dopředná neuronoová síť s jednou skrytou vrstvou (dvouvrstvá síť, vstupní vrstva se nezapočítává) (obr.2.7).

Neurony v poslední (výstupní) vrstvě sítě většinou neobsahují aktivační funkci, protože požadovaným výstupem je rozdělení mezi konkrétní počet tříd reprezentovaných celými čísli, nebo pravděpodobnost příslušnosti k dané třídě. Proto dělíme sítě na klasifikační a regresní.

Velikost sítě se udává buď v počtu neuronů, nebo v počtu parametrů, což se rovná počtu spojů neboli cest v grafu.

Jedním z hlavních důvodů, proč jsou umělé neuronové sítě organizovány do vrstev, je možnost využití maticových výpočetních operací. To ve spojení se současným

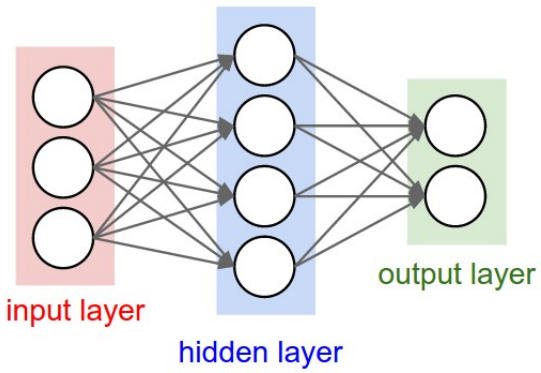



**Obrázek 2.6:** Aktivační funkce

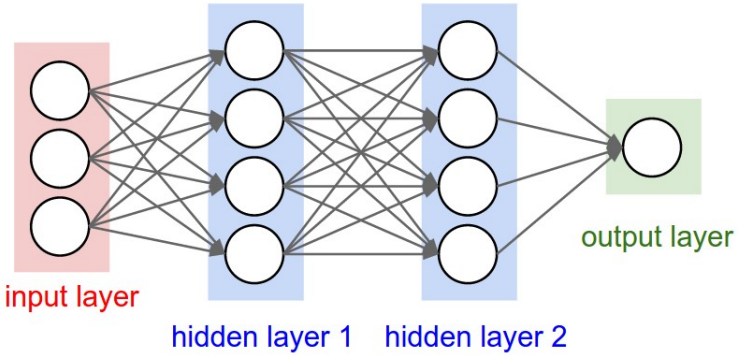
výkonem grafických karet, které jsou pro takové operace konstruovány, dělá z neuronových sítí velice silný nástroj v oblasti strojového učení. Otázkou zůstává jak zvolit počet vrstev. U konvolučních sítí (2.4.2.) je hloubka velice důležitým a žádaným parametrem, nicméně u ostatních architektur to tak nebývá. Pokud zvolíme nevhodně příliš velký počet vrstev může dojít k takzvanému přetrénování, při kterém síť dosahuje "lepších" výsledků na úkor kvality.

Neuronové sítě dělíme na několik základních typů. První rozdělení již bylo zmíněno a to jsou sítě jednoduché a hluboké. Dále je dělíme na dopředné a zpětnovazebné. Speciálním typem jsou pak konvoluční neuronové sítě. Vše co se zde zatím popisovalo bylo pouze v rámci dopředného průchodu.

Sítě zpětnovazebné využívají plný potenciál neuronových sítí. Jedná se o síť kte-

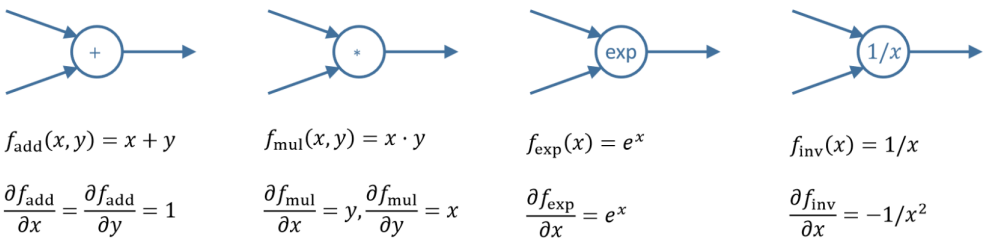



**Obrázek 2.7:** Dvouvrstvá síť (jedna skrytá vrstva se 4 neurony a jedna výstupní vrstva se 2 neurony [7]




**Obrázek 2.8:** Třívrstvá síť (dvě skryté vrstvy se 4 neurony a jedna výstupní vrstva se 2 neurony) [7]

rá nevyhodnocuje pouze výstup v podobě přiřazení do tříd, ale využívá informaci o své koreknosti ke zpětnému průchodu, kdy pomocí derivací jednotlivých prvků sítě, upravuje váhy, které vedly ke špatnému výsledku. Pří dopředném průchodu na konci sítě vznikne podle zvoleného kritéria odchylka, nebo chyba. Cílem procesu učení je tuto chybu minimalizovat. Minimalizace chyby se provádí pomocí postupného výpočtu gradientu na každý blok sítě pomocí algoritmu zpětné propagace. Výpočet gradientu spočívá ve výpočtu derivace přes každý blok sítě v respektu k jeho výstupu a tento princip se nazývá řetízkové pravidlo. Jednotlivé bloky v grafu sítě se opakují při dopředném, i zpětném průchodu. Na obrázku 2.9 jsou vyobrazena pravidla pro výpočet gradientu každého bloku. Vynásobení váhy konkrétním vypočteným gradientem síť upravujeme neboli učíme .Učení sítě má důležitý hyperparametr, který se nazývá Learning rate. Learning rate udává míru, jakou budeme upravovat váhy neuronů. Pokud nastavíme tento parametr příliš malý minimalizace chyby bude příliš pomalá, pokud příliš velký dojde ke konvergenci, případně k divergenci a síť nenalezne požadované minimum.




**Obrázek 2.9:** Pravidla výpočtu gradientu (derivace) přes bloky sítě

Proces zpětné propagace se v praxi neprovádí po každém dopředném průchodu, ale využívá se metoda Mini-batch Gradient Descent, která rozděluje trénovací data na dávky (batch). Pokud je dána dávka 100, v praxi to znamená, že síť provede dopředný průchod se 100 vzorky a poté provede jeden zpětný průchod, kterým upraví váhy. Pokud by síť po každém dopředném průchodu prováděla i zpětný průchod, bylo by to výpočetně neefektivní. Jelikož mezi vzorky je zpravidla jistá míra kolerace, můžeme si dovolit využít tento přístup. Při návrhu sítě je také na místě kvalitní

inicializace vah. Váhy nelze nastavit na nuly, nebo jiná stejná čísla, ale je třeba porušit symetrii, díky čemuž bude každý neuron dělat něco jiného.

### 2.4.2 Konvoluční Neuronové sítě (CNN)

V této práci se využíváji takzvané konvoluční neuronové sítě. Konvoluční sítě jsou velice polulárním modelem využívaným v mnoha odvětvích, převážne ve zpracování obrazových dat. Nicméně můžou být použity i ve zpracování řeči neboli dat zvukových. Představíme-li si spektrogram zvukového signálu, jedná se o vizualizovatelná dvourozměrná data, stejně jako je tomu u obrazu. Konvoluční sítě jsou populární díky několika nezanedbatelným výhodám. První výhodou je eliminace potřeby ruční extrakce příznaků, síť se příznaky sama naučí. Další výhodou jsou bezkonkurenční výsledky rozpoznávání. A třetí výhodou je možnost předtrénování sítí, případně použití již předtrénovaných sítí.

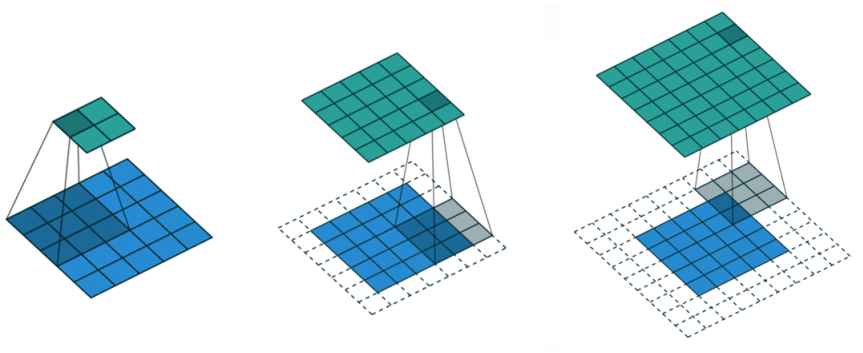
Konvoluční sítě se liší v jedné věci. Přepokládají že vstupem budou obrazová data. Díky tomu může být jejich architektura upravena a zefektivněn postup při práci s příznaky. Síť může obsahovat stovky vrstev, kde se každá učí detekovat určité příznaky obrazu. Mezi vrstvami se nacházejí filtry které mohou pouze měnit hodnotu jasu, měnit rozměry obrázku, ale i umocňovat některé charakteristické rysy. (obr.2.11)

Síť tradičně tvoří tři základní druhy vrstev. Vrstva konovluční, pooling vrstva a takzvaná fully-connected vrstva. Konvoluční vrstva vezme vstupní obraz a pomocí sady filtrů z obrazu vyextrahuje určité příznaky. Pooling vrstva upravuje její vstup tak, že redukuje počet příznaků, které se musí síť naučit. Poslední fully-connected vrstva stejně jako u tradičních sítí vypočítá skóre přiřazení do tříd. V konvoluční síti se může vyskytovat i ReLu vrstva aplikující aktivační funkci ReLu.

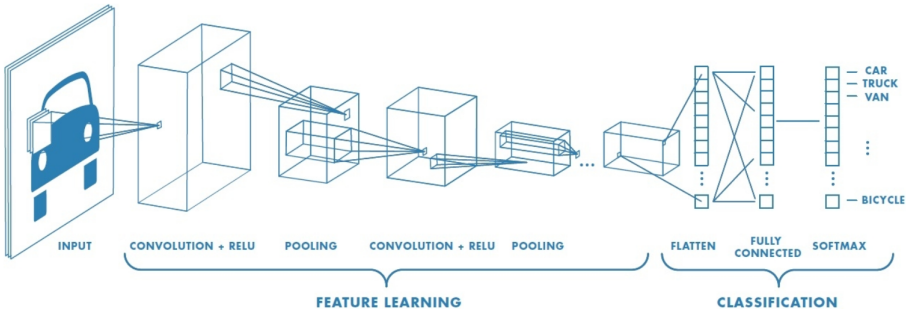
Mějme 10 tříd a příklad vstupního vektoru reprezentující obraz o velikosti [32x32x3] (32x32xRGB kanály). Následuje konvoluční vrstva počítající výstup pro každý neuron připojený se vstupní vrstvou a obsahující 12 filtrů. Její výstup je [32x32x12]. ReLu vrstva ponechá rozměry, pouze na všechna čísla aplikuje aktivační funkci ReLu. Následující poolingová vrstva sníží rozměry na [16x16x12]. A poslední fully-connected vrstva vypočítá skóre pro každou z 10 tříd a jejím výstupem je tedy [1x1x10]. Zjednodušeně, konvoluční síť postupnými kroky ze vstupního obrázku vyvtvoří vektor udávající jeho příslušnost k jedné ze tříd.[7]

Konvoluční vrstva obsahuje řadu filtrů. Uvažujme filtr rozměru [5x5x3]. Tento filtr postupně posouváme, přesněji konvolujeme přes všechny kanály originálního obrazu (obr.2.10) a pomocí skalárního součinu hodnot vstupu s hodnotami filtru získáváme model obrazu zvýrazňující určité charakteristické parametry, jako jsou například hrany. Každý z filtrů je navržen tak, aby zvýraznil různé paramerty obrazu.

V případě tohoto projektu budou sítě pracovat přesně výše zmíněným způsobem. Vstupními daty budou jendotlivé spektrogramy audio signálu o velikosti FxPx1 kde je F počet framů a P počet parametrů. Jaká bude zvolena architektura a druhy filtrů zjistíme pomocí experimentů v kapitole (4.4). Poté co síť projde vrstvami a naučí se konkrétní příznaky, přijde na řadu klasifikační část sítě. Na výstupu celé sítě je tedy



**Obrázek 2.10:** Aplikace filtru, vlevo metodou ”valid”, uprostřed ”same”, vpravo ”full”[8]




**Obrázek 2.11:** Vizualizace konvoluční sítě s několika vrstvami [7]

vektor $K = [1xI]$ kde I je počet tříd (v této práci I=176).[7] A každé pole vektoru obsahuje číslo udávající pravděpodobnost, že vstupní obraz připadá této třídě $K_i$.

# 3 Příprava dat

Základním kamenem rozpoznávání řeči je vedle samotné rozpoznávací metody také soubor dat, se kterým budou rozpoznávací algoritmy pracovat. V první řadě je nezbytné promyslet, jaká data jsou pro řešení problému nezbytná. Cílem této práce je aplikace, která pomocí slovních povelů bude schopna ovládat základní funkce počítače. Proto je potřeba vytvořit slovník povelů, pomocí nichž bude uživatel schopen komunikovat s operačním systémem.(Příloha A.2)

Pro tuto konkrétní práci byl vytvořen slovník 176 slov, obsahující povely pro ovládání klávesnice, klávesových zkratek a myši. Mezi povely jsou znaky abecedy i jejich varianta v podobě jmen, pro přesnější rozpoznávání. Dále se zde nachází základní ze speciálních znaků, funkční tlačítka, šipky, speciální klávesy, základní klávesové zkratky, zkratky pro pohyb v internetovém prohlížeči a v poslední řadě povely pro pohyb myši.

Každá z metod rozpoznávání řeči potřebuje vzorové nahrávky pro každý z povelů od více řečníků. Obecně platí, že čím více nahrávek jsme schopni předat rozpoznávacímu algoritmu, tím lepších výsledků lze dosáhnout. Nicméně získávání nahrávek je časově náročné, proto je na místě provést rešerši konkrétního problému a zjistit nezbytný počet nahrávek pro jeho řešení. V této práci bylo cílem získat nahrávky povelů od minimálně 30 řečníků, abychom byli schopni naimplementovat dostatečně funkční aplikaci.

## 3.1 Nahrávání

Nedílnou součástí aplikací tohoto typu je kolekce dat. V mnoha případech můžeme využít datasety nashromážděné velkými technologickými společnostmi, ale pokud se jedná o zpracování řeči, musíme se většinou spokojit s anglickým nebo jiným rozšířeným jazykem. Pokud je cílem udělat aplikaci využívající českých povelů, je třeba nashromáždit vlastní data. Proces sběru dat může být často časově nejnáročnější operací celého projetku, bereme-li v potaz že chceme alespoň 30 řečníků, kdy každý musí být seznámen s pravidly nahrávání a namluvit správným způsobem 176 povelů.

Aby se proces co nejvíce usnadnil a urychlil, byla vytvořena aplikace, která intuitivně navádí řečníka k samostatnému nahrávání a ukládání nahrávek požadovaným způsobem. Nejprve je řečník seznámen se všemi povely ve vytvořeném slovníku, dále s pravidly nahrávání, jako je dvouvteřinový interval, který musí obsahovat ticho, povel a ticho. A poté byl seznámen s jednoduchým přepínáním mezi povely a jejich ukládáním. Aplikace vizuálně zobrazuje zaznamenaný signál, u kterého je možné s minimem zkušeností vypozorovat, jestli byl signál vhodně zaznamenán. Případně je možné nahrávku přehrát.

Během nahrávání byl každý řečník autorem práce kontrolován. Celkově se nashromáždilo 33 řečníků, kdy každý namluvil požadovaných 176 povelů. Řečníci byli rozděleni do trénovací a testovací skupiny. Pro aplikaci neuronových sítí byli rozděleni do třech skupin, kdy přibyla navíc skupina validační. Tabulka (3.1) znázorňuje rozdělení do konkrétních skupin. V levém sloupci jsou označení řečnící. První písmeno značí, jestli jde o ženu - Z, nebo o muže - M. V následujících sloupcích je vyobrazena příslušnost řečníka do skupiny. V tabulce (3.2) je znázorněn počet nahrávek pro trénovací a testovací sadu a také počty pro sady určené neuronovým sítím.

**Tabulka 3.1:** Rozdělení datasetu

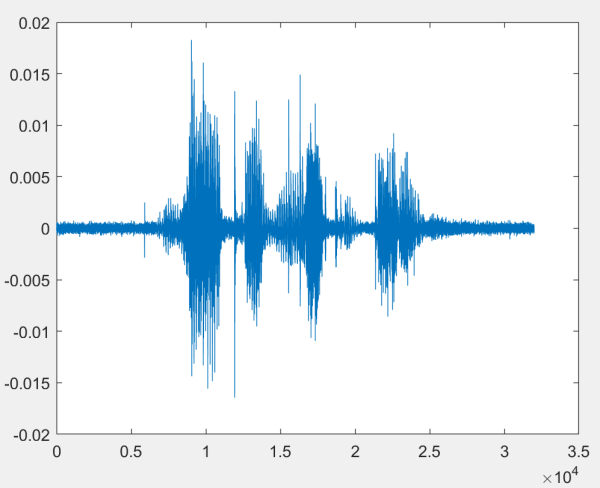
| Mluvčí | Sada | Sada(ANN) |
|---|---|---|
| MAN | trén | valid |
| MDB | trén | train |
| MDT | trén | valid |
| MDX | trén | valid |
| MJB | trén | trén |
| MJK | trén | trén |
| MJS | trén | trén |
| MJX | trén | trén |
| MLB | trén | trén |
| MLW | trén | trén |
| MMX | trén | trén |
| MPP | trén | trén |
| MRR | trén | trén |
| MTY | trén | trén |
| MTZ | trén | trén |
| MXX | trén | trén |
| ZAN | trén | trén |
| ZAS | trén | trén |
| ZIB | trén | trén |
| ZKM | trén | trén |
| ZMP | trén | trén |
| ZMS | trén | trén |
| ZSR | trén | trén |
| ZTF | trén | trén |
| ZXX | trén | trén |
| MFC | test | test |
| MFR | test | test |
| MPS | test | test |
| MVS | test | test |
| MWW | test | test |
| ZJS | test | test |
| ZKF | test | test |
| ZVR | test | test |

**Tabulka 3.2:** Počet nahrávek pro jednotlivé sady

| | trén | test | valid |
|---|---|---|---|
| Původní slovník | 4400 | 1408 | x |
| Původní slovník pro aplikaci ANN | 3872 | 1408 | 528 |

## 3.2 Digitalizace

Dalším krokem je převod audio signálu do digitální, tedy počítačem lépe zpracovatelné podoby. Při digitalizaci je nezbytné zvolit dva parametry, jimiž jsou vzorkovací frekvence a kvantizační krok. Pokud bychom zkoumali fyzikální vlastnosti lidské řeči, dostali bychom se u vzorkovací frekvence k číslu 16kHz, jenž je jistým standardem v této disciplíně. Podobným způsobem lze vyvodit kvantizační krok, který je v tomto konkrétním případě 16 bitů. Aby byla práce s nahrávkami snažší, byl sjednocen čas všech nahrávek na dvě vteřiny. Číslo bylo nastaveno na základě experimentu trvání nejdelších z povelů(obr.3.1).




**Obrázek 3.1:** Vizualizace povelu ”Základní skupina”

## 3.3 Segmentace signálu a nalezení hrany slova

Následuje rozdělení signálu na krátké časové úseky. Tento proces se nazývá segmentace signálu. Úseky se volí na základě trvání nejkratší hlásky. Standardem je 10–25$ms$. Pro tuto práci byla zvolena hodnota 25 ms. Tyto úseky se nazývají podle anglického výrazu framy. Aby byly zachovány všechny důležité části signálu, je na místě, aby se sousední framy překrývaly. Proto nový frame začíná každých 10 ms signálu (framová frekvence). Každá nahrávka je tímto rozdělena na 200 framů.

Každá nahrávka obsahuje příkaz a ticho. Díky rozdělení na jednotlivé framy je možné ticho od příkazu oddělit a dále pracovat se zkráceným signálem, který obsahuje pouze hlas. Pro každý frame se vypočte energie, která se následně zlogaritmuje (vztah3.1). Získáním průměrné hodnoty log(ene) v místech ticha a v místech příkazu lze stanovit hodnotu prahu T. Framy F které nesou hodnotu $F_i < T$ se odstraní a zbyde pouze signál samotného příkazu.

$$E = log\left(\sum_{i=0}^{N-1} x_i^2\right) \tag{3.1}$$

Vhodné je na všechny nahrávky aplikovat filtr, kterým sjednotíme barvu a hlasitost nahrávky.

## 3.4 Parametrizace

Dalším krokem přípravy dat je parametrizace. Parametrizací se rozumí proces při kterém pro každý frame vypočítáme číslo, nebo vektor, který tento krátký časový úsek reprezentuje. Způsob jakým číslo, případně vektor čísel získáme se dopředu určí druhem parametrů, kterými chceme signál reprezentovat. Parametry signálu, by měly být voleny tak, aby co nejlépe popsaly daný signál, zvýraznily jeho důležité aspekty, a potlačily redundantní informace (barva, emoce ad.). Obecným tvarem slova bude matice o rozměrech FxP, kde P je počet parametrů a F počet framů.V následujicích podkapitolách budou parametry používané v této práci detailněji popsány

### 3.4.1 Energie

Nejjednodušší z příznaků je logaritmus energie jenž se vypočítá pro každý frame (vztah 3.1). Jelikož výsledkem vztahu je jedno číslo, příznakový vektor povelu bude mít rozměr Fx1.

### 3.4.2 Energie + ZCR

Druhá sada příznaků bude obsahovat opět energii a spolu s ní i hodnotu ZCR (Zero-crossing rate).

$$ZCR = \frac{1}{2}\left(\sum_{n=1}^{N} |sgn(n) - sgn(n-1)|\right) \tag{3.2}$$

ZCR je hodnota, která uvádí, počet průchodů signálu osou na hodnotě 0. Zde bude každý příkaz maticí o velikosti Fx2.

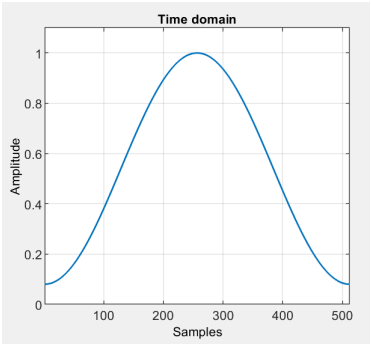
### 3.4.3 Spektrální příznaky

Další sadou jsou příznaky spektrální. Spektrální příznaky jsou založeny na frekvencích, proto je třeba na jednotlivé framy aplikovat diskrétní N-bodovou Fourierovu transformaci:

$$X_{DFT}[k] = \sum_{n=0}^{N-1} x(n)\,[cos(2\pi nk/N) - jsin(2\pi nk/N)] \tag{3.3}$$

ale ještě před tím aplikujeme na frame Hammingovo okénko(obr.3.2):

$$w(s_n) = 0.54 - 0.46 \cdot cos\left(\frac{2\pi n}{N-1}\right) \tag{3.4}$$




**Obrázek 3.2:** 512-bodové hammingovo okno

které dá vyšší hodnotu prostředním hodnotám, jelikož krajní hodnoty jsou více ovlivněné okolím a případnému chybnému ořezu. Fourierova transformace pomocí harmonických signálů (sin a cos) převádí signál z časové oblasti do oblasti frekvenční.[9] Dále se vypočte amplituda, která je rovna asbolutní hodnotě a určíme počet kanálu K, kterými určujeme typ spekrtálních příznaků. Používané kanály jsou K = 8, 16, 32, 64. Následná velikost příznakové matice bude FxK

### 3.4.4 Kepstrální příznaky

Kepstrální příznaky jsou modifikací příznaků spektrálních. Definují se zpětnou Fourierovou transformací logaritmu absolutní hodnoty spektra signálu.

$$c[n] = DFT^{-1} log(|DFTx[n]|) \tag{3.5}$$

Zde budou použiti MFCC kepstrální příznaky. MFCC pracují s mel-frekvencí signálu. Vztah pro převod frekence na mel-frekvence:

$$Mel(f) = 2595 \cdot log_{10}\left(1 + \frac{f}{700}\right) \tag{3.6}$$

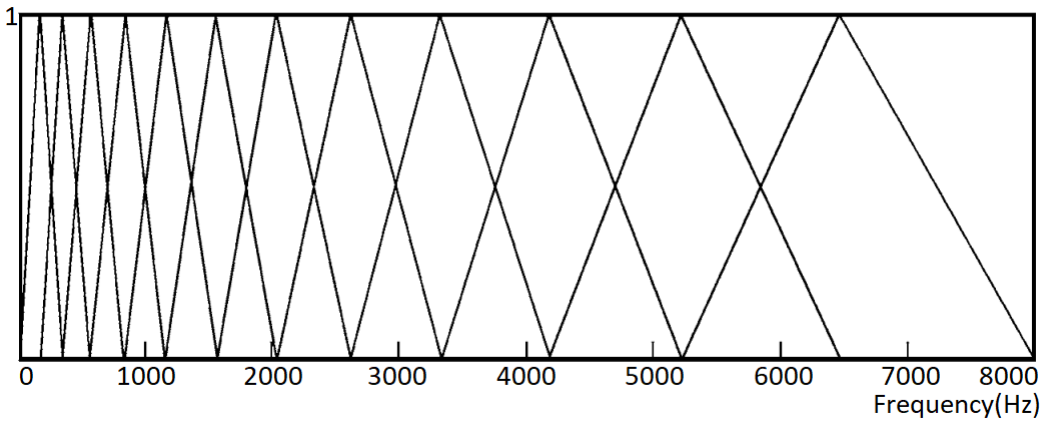
Extrakce příznaků MFCC začíná výpočtem amplitudového spektra pro každý frame signálu pomocí FFT. Dále se amplitudové spektrum přefiltruje speciálně navrženou bankou filtrů obsahující M trojúhelníkových filtrů(obr.3.3)(vztah 3.7).

$$E(m) = \sum_{k=b_m-\delta_m}^{b_m+\delta_m} X(k) \cdot U_{\delta m}(k + b_m) \tag{3.7}$$

Tyto filtry pracují s faktem, že lidský sluch, vnímá frekvence signálu přibližně v logaritmické stupnici. Výsledkem po aplikaci banky filtrů je vektor energie $E = (E_1, E_2, ..., E_m)$. Podle vzorce (vztah3.8) vypočteme kepstrální koeficienty pomocí diskrétní kosinové transformace

$$c(k) = \sum_{i=1}^{N} log(m_i) cos\left(k(i - 0.5)\frac{\pi}{N}\right) \tag{3.8}$$

Kde $m_i$ jsou výstupy jedontlivých filtrů a M počet trojúhelníkových melovských filtrů.




**Obrázek 3.3:** Banka filtrů MFCC

# 4 Implementace metod a výsledky

V této práci je popsána implementace každé z výše zmíněných metod, nástroj ve kterém implementace proběhla a její vyhodnocení v podobě výsledků. V předchozí kapitole jsme si ukázali, jak byly získány konkrétní parametry. Proto se tomu již nebudeme v této kapitole dále věnovat a přejdeme rovnou k implementačním krokům.

Trénovací sada nahrávek obsahuje povely namluvené jinými řečníky než sada testovací, tedy nezávislé na mluvčích (Speaker Independenc). Testování všech metod probíhalo nejdříve na kompletním slovníku povelů. Po nalezení nejúspěšnější metody byl slovník postupně upravován a optimalizován pro lepší chod závěrečné aplikace. Původním plánem bylo rozdělit slovník do skupin oddělujících příkazy pro ovládání klávesnice, myši a případně internetového prohlížeče. Během experimentů bylo zjištěno, že nejvyšší chybovosti dosahují povely pro ovládání myši. Konkrétně šlo o záměnu směrů př. DOLEVA-50 a DOPRAVA-50 a záměnu hodnot posunu př. DOLEVA-10 a DOLEVA-500. Další množina chyb se vyskytovala u samostatných písmen př. BE, CE, E apod. a ovelů pro stisk funkčních kláves př. FUNKCE-1 a FUNKCE-11. To znamená, že rozdělení do jednotlivých skupin by nerozdělilo konkrétní množiny chyb. Proto byla ponechána jedna skupina, ze které byly odstraněny povely pro samostatná písmena a byla ponechána pouze jejich reprezentace pomocí jmen a dále byla odstraněna polovina povelů pro pohyb myši. Po experimentech bylo usouzeno, že pohyb o 1 nebo 5 pixelů je téměř zbytečný. Byly tedy odstraněny posuny o 1, 5, 20 a 100 pixelů a ponechány posuny o 10, 50, 200 a 500.

Pro všechny metody proběhly tedy dvě sady testů. Jedna s původním slovníkem (176 povelů), druhá s upraveným zkráceným slovníkem (135 povelů). Zároveň byly použity 4 sady parametrů (energie, energie + zcr, FBANK a MFCC příznaky) pro vzdálenostní metody (LTW a DTW) a pro metody HMM a ANN pak dvě úspěšnější sady příznaků (FBANK a MFCC). Číslo udávající finální skóre v procentech se vypočítalo pomocí počtu všech správně klasifikovaných povelů vydělené počtem všech testovacích povelů.

K popisu tabulek s výsledky slouží tato legenda:

- Původní slovník - 176 povelů (trénovací sada = 4400 nahrávek, testovací sada = 1408 nahrávek, pro ANN trénovací sada = 3872 nahrávek, testovací sada 1400 nahrávek a validační sada 528 nahrávek)
- Zkrácený slovník - 135 povelů (trénovací sada = 3375 nahrávek, testovací sada = 1080 nahrávek, pro ANN trénovací sada = 2970 nahrávek, testovací sada 1080 nahrávek a validační sada 405 nahrávek)
- Příznaková sada energie - jeden příznak na frame spočítaný podle kapitoly 3.4.1
- Příznaková sada energie + ZCR - dva příznaky na fram spočítané podle kapitoly 3.4.2
- Příznaková sada FBANK - 16 spektrálnách příznaků odvozených od kapitoly 3.4.3
- Příznaková sada MFCC - 39 příznaků na frame odvozených od kapitoly 3.4.4

## 4.1 LTW

Metoda LTW je nejjednodušší z metod používaných v této práci. Proto se dá předem očekávat, že její výsledky nebudou natolik kvalitní, abychom na jejím principu mohli postavit finální aplikaci. Nicméně pochopení jejího principu usnadňuje chápání metod následujících.

Tato metoda obsahuje dvě funkce. První je funkce která nám vyhotoví přímku, neboli transformační cestu dvou porovnávaných signálů (vztah 2.1). Druhou funkcí je výpočt euklidovské vzdálenosti (vztah 2.3) mezi vstupním signálem a signálem transformační cesty.

```
 %  Funkce vytvářející transformační cestu a vracející vzdálenost mezi ní
 %  a vstupem X
 %  X - hledaný signál, I - délka X
 %  Y - referenční signál, J - délka Y
 %  P - počet příznaků
function res = computeLTW(X,I,Y,J,P)
    f = zeros(I,1);
    for i = 1:I
        f(i) = int8( fix(((J-1)/(I-1)) * (i - 1) + 1 + 0.5));
    end
    Yltw = Y(f,:);
    res = computeEuclidDist(X,Yltw,P);
end
```

**Obrázek 4.1:** Skript metody LTW

Metoda byla implementována ve vývojovém prostředí MATLAB. A vstupem funkce jsou dva signály X a Y a jejich délka v počtu framů I a J. Třetím vstupním parametrem je P, který udává počet příznaků na jeden frame.

### 4.1.1 Výsledky

Při poměrně vysokém počtu tříd ke klasifikaci (počet povelů), jako je tomu v tomto případě a jedním, případně dvěma příznaky na frame, je patrné že metoda LTW selhává. Ani zkrácení o 40 povelů nedokázalo zlepšit její úspěšnost. Při 16 (FBANK), případně 39 (MFCC) příznacích na frame se již dostavují výsledky.

**Tabulka 4.1:** Výsledky metody LTW

| | Energie | Energie+ZCR | FBANK | MFCC |
|---|---|---|---|---|
| původní slovník | 2,53% | 4,67% | 17,65% | 31,21% |
| upravený slovník | 3,98% | 4,33% | 21,38% | 36,29% |

## 4.2 DTW

Metoda DTW byla také implmentována v prostředí MATLAB. Zde se již očekávaly lepší výsledky díky použití dynamických programovacích metod. Po implementování vlastní metody dtw, bylo zjištěno, že vývojové prostředí MATLAB nabízí funkci, která provádí výpočet DTW. Po experimentech bylo zjištěno že vlastní metoda dosahuje velmi podobných výsledků (rozdíl jednotek procent), jako metoda MATLABu, která je ale pochopitelně kvalitně optimalizována a tím pádem výpočetně rychlejší. Proto bylo rozhodnuto dále využívat metodu MATLABu.

Do metody byla předávána zparametrizovaná data neznámého povelu a data referenčního vzoru. Metoda má jednoduchý tvar $dtw(X, Y)$. Iterační metodou přes veškeré referenční vzory získáme nejmenší vzdálenost ke vzoru, jemuž je neznámý povel přiřazen.

### 4.2.1 Výsledky

U metody DTW jsou patrné rozdíly v počtu parametrů reprezentujících jeden frame. Vidíme, že čím lepší parametry metodě předáme, tím větší úspěšnost lze očekávat. Na druhou stranu i strmý nárust výpočetní náročnosti, který metodu i při téměř 90% činí nepoužitelnou pro naši aplikaci.

**Tabulka 4.2:** Výsledky metody DTW

| | Energie | Energie+ZCR | FBANK | MFCC |
|---|---|---|---|---|
| původní slovník | 12,78 % | 31,49% | 78,18% | 83,19% |
| upravený slovník | 13,88% | 37,31% | 85,09% | 89,26% |

## 4.3 HMM

Pro implementaci skrytých markovských modelů byl využit Cambridský HTK Toolkit. Ten obsahuje několik modulů naprogramovaných v jazyce C. Tyto moduly se

ovládají z příkazové řádky a volají se vždy s konkrétními soubory dat, které obsahují slovník, odkazy trénovacíh dat, nebo samotné hmm modely. Zde si popíšeme konkrétní postup.

Jako prvním vytvoříme adresář v němž budou všechny potřebné moduly a nástroje HTK Toolkitu a 6 podadresářů pojmenovaných *hmm0, hmm1,...,hmm6*. Prvním krokem při vytváření modelů je vytvoření souboru *grammar*. HTK poskytuje speciální jazyk pro sepcifikaci slov, které budou rozpoznávány a určení způsobu (jedno slovo, více slov atd.). V souboru *grammar* svislé závorky | oddělují možnosti povelů k rozpoznání. Definovali jsme tedy všchny povely pro které chceme vytvořit modely a určili jsme, že hledáme vždy jeden konkrétní povel před a po kterém předchází a následuje ticho (obr. 4.2).

```
 $word = ADAM | BOZENA | CYRIL | DAVID | EMIL | FRANTISEK | GUSTAV | HELENA | IVAN | JOSEF | ...
KONEC-KLAVESNICE | MYS | KONEC-MYSI | PREDCHOZI-SLOVO | DALSI-SLOVO | STRANKA-DOLU | ...
NAHORU-50 | NAHORU-200 | NAHORU-500 | DOLU-10 | DOLU-50 | DOLU-200 | DOLU-500;
( SENT-START ( $word ) SENT-END )
```

**Obrázek 4.2:** Zkrácená verze souboru grammar pro ilustraci

Pomocí nástroje HParse jsme ze souboru *grammar* vygenerovali síť slov ve formátu SLF (HTK standard Lattice Format), kde jsou instance povelů ve formátu zpracovatelném HTK Toolkitem. Dalším krokem bylo vytvoření listu samostatných abecedně seřazených povelů nazvaného *wlist* (obr.4.3), kde je každý povel na jednom řádku. Zde nesmíme zapomenout zahrnout povely *SENT-START []* a *SENT-END []*, které jsou používany k definování začátku a konce slova.

```
...
ROVNA-SE
RUDOLF
SEDM
SENT-END []
SENT-START []
SEST
STRANKA-DOLU
STRANKA-NAHORU
STREDNIK
...
```

**Obrázek 4.3:** Ilustrace části souboru wlist

Dalším potřebný soubor nazveme *lexicon* (obr.4.4). Ten specifikuje výslovnost pro každý model. Jelikož je cílem trénovat celoslovné modely, výslovnost bude stejného tvaru jako samotný povel.

Na soubor wlist je volán nástroj HDMan, jenž vytvoří soubor *wlist* za pomocí výslovností udávaných v souboru *lexicon* finální soubory *dict* a *models0*.

Dále následuje parametrizace dat. HTK Toolkit je schopen si sám zparametrizovat audio soubory na základě konfigurace, kterou mu zadáme. V dokumentaci HTK toolkitu (CITE HTKBOOK) můžeme nalézt možné volby parametrů. V této práci se budou používat pro metody HMM dvě základní , FBANK (logaritmus energie v mel-filtrových pásmech) a MFCC (kepstrální koeficienty). Na příkladu bude

```
...                 ...
ROVNA-SE            rovna-se
RUDOLF              rudolf
SEDM                sedm
SENT-END []         sil
SENT-START []       sil
SEST                sest
STRANKA-DOLU        stranka-dolu
STRANKA-NAHORU      stranka-nahoru
...                 ...
```

**Obrázek 4.4:** Ilustrace části souboru lexicon

uvedena verze s MFCC paramerty. Verze s parametry FBANK se liší pouze v konfiguraci parametrů. Vytvoříme tedy soubor *param-config* s parametry požadovaných příznaků (obr. 4.5).

```
# Coding parameters
SOURCEFORMAT = WAV
TARGETKIND = MFCC_0_D_A
TARGETRATE = 100000.0
SAVECOMPRESSED = F
SAVEWITHCRC = F
WINDOWSIZE = 250000.0
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
ENORMALISE = F
```

**Obrázek 4.5:** Konfigurační soubor config pro MFCC parametry

Parametrizace modelů učiníme prostřednictvím nástroje HCopy kterému předáme konfigurančí soubor *config* a list vstupních trénovacích souborů a jejich odpovídající výstup, který má podobu dvou sloupců. V prvním sloupci je cesta k wav souboru nahrávky:

$$C : /Users/bemic/Desktop/Bak/HMM/05/DATA/train/MAN/001.wav$$

Na druhém požadovaný výstupní soubor mfc:

$$C : /Users/bemic/Desktop/Bak/HMM/05/DATA/train/MAN/001.mfc$$

.

Následujícím krokem je výpočet statistického rozdělení modelu. V první řadě je třeba definovat topologii modelu tím, že vytvoříme dva prototypové soubory. Jeden 8-stavový pro modely povelů, druhý 3-stavový pro model ticha . Oba modely mají levopravou strukturu a obsahují vektory o délce 39. Toto číslo je vypočteno součtem zparametrizovaného vektoru *MFCC_0_D_A*, delta koeficientu a akceleračního koeficientu. Hodnoty v prototypovém modelu nejsou důležité a budou nahrazeny. Důležitá je struktura modelu. Na obrázku (obr.4.6) je příklad pro 3-stavový model. 8-stavový model vypadá totožně, pouze má o 5 stavů více. Jsou li definované modely, vytvoříme soubor v kterém uvedeme cesty ke všem trénovacím souborům, které

```
~o <VecSize> 39 <MFCC_0_D_A>
~h "proto-3s-39f"
<BeginHMM>
<NumStates> 5
<State> 2
<Mean> 39
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<Variance> 39
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
<State> 3
<Mean> 39
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<Variance> 39
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
<State> 4
<Mean> 39
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<Variance> 39
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
<TransP> 5
0.0 1.0 0.0 0.0 0.0
0.0 0.6 0.4 0.0 0.0
0.0 0.0 0.6 0.4 0.0
0.0 0.0 0.0 0.7 0.3
0.0 0.0 0.0 0.0 0.0
<EndHMM>
```

**Obrázek 4.6:** Prototyp 3-stavového modelu

mají být použity k výpočtu gaussova rozdělení jednotlivých stavů modelů. Soubor pojmenujeme train.scp a jeho obsahem bude tedy na každém řádku jedna cesta ve tvaru:

$$C : /Users/bemic/Desktop/Bak/HMM/05/DATA/train/MAN/001.mfc$$

Nakonec přidáme opět konfigurační soubor *train-config*, který má kromě prvního řádku "SOURCEFORMAT = WAV" stejný tvar jako *param-conifg*. Poté už můžeme 2x volat nástroj HCompV v následujícím tvaru:

*"HCompV -C train-config -f 0.01 -m -S train.scp -M hmm0 proto-8s-39f"*

a podruhé s 3-stavovým modelem *proto-3s-39f*. V adresáří *hmm0* je třeba vytvořit soubor *hmmdefs*, který bude obsahovat natrénovaný 8-stavový model tolikrát, kolik máme povelů a kde namísto pole *~h "proto-8s-39f"* bude vždy jeden z povelů *~h "tabulator"*. Na konec souboru přidáme natrénovaný 3-stavovový model reprezentující ticho *~h "proto-3s-39f"* –> *~h "sil"*

Posledním krokem je samotné trénovaní. K trénování využijeme již vytoření konfigurační soubor train-config a soubor s cestami ke zparametrizovavým povelům train.scp. Vytvoříme soubor *train.mlf*, který uvádí cesty k trénovacím souborům typu

```
#!MLF!#
"C:/Users/bemic/Desktop/Bak/HMM/05/DATA/train/MAN/001.lab"
sil
adam
sil
.
"C:/Users/bemic/Desktop/Bak/HMM/05/DATA/train/MAN/002.lab"
sil
bozena
sil
.
"C:/Users/bemic/Desktop/Bak/HMM/05/DATA/train/MAN/003.lab"
sil
cyril
sil
.
...
```

**Obrázek 4.7:** Část souboru train.mlf

*.lab*, které byly během procesu vytvořeny, a slovo které soubory reprezentují. Tvar souboru pak bude viz (obr. 4.7).

Nástrojem pro trénování je modul HERest, kterému předáme následující parametry:

*"HERest -C train-config -I train.mlf -t 250.0 150.0 1000.0 -S train.scp -H hmm0/hmmdefs -M hmm1 models0*

Tento příkaz zavoláme šestkrát pro *hmm models0, hmm1 models1,...,hmm6 models6*. Jakmile program ukončí trénování v adresáři hmm6 bude soubor hmmdefs obsahovat finálně natrénované modely.

Testování poté probíhá pomocí modulu HVite, kterému jsou předána testovací slova a on pomocí natrénovaných modelů je schopen přiřadit neznáme slovo k jednomu z nich. Příkaz pro testování voláme následujícím způsobem:

*HVite -H hmm6/hmmdefs -S test.scp -i recout.mlf -w wordnet -p -70.0 -s 0 dict models0*

Výsledkem je soubor *recout.mlf*, který udává jaký model byl konkrétnímu testovanému slovu přiřazen. Z toho lze vypočítat výslednou úspěšnost zadáním tohoho příkazu:

*HResults -e ??? SENT-START -e ??? SENT-END -t -I testref.mlf models0 recout.mlf*

### 4.3.1 Výsledky

Metoda HMM dosáhla nejvyšší úspěšnosti za celou dobu vyhotovování tohoto projektu. S téměř 92% jsme již schopni ovládat počítač na základní úrovni. Stále se vyskytují chyby, ale při vyvarování se povelů náchylných na špatnou klasifikaci je tato úspěšnost dostatečná pro naši aplikaci.

- U metody HMM je použita pouze příznaková sada FBANK a MFCC

**Tabulka 4.3:** Výsledky metody HMM

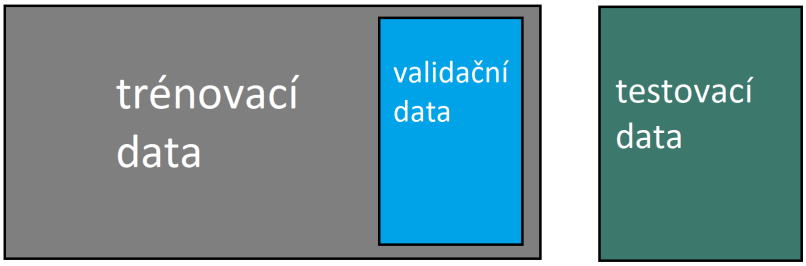
| | FBANK | MFCC |
|---|---|---|
| původní slovník | 27,13% | 89,68% |
| upravený slovník | 28,43% | 91,67% |

## 4.4 ANN

Na základě průzkumu a doporučení byl pro náš problém rozpoznávání izolovaných slov zvolen postup využívající konvoluční neuronové sítě. Jak již bylo zmíněno, jejich využití je především v úlohách pracujících obrazem. Nicméně audio signály vyobrazené spektrogramem, případně pomocí kepstra se dají považovat jako obrazová data a můžeme tedy využít tento přístup.

Pro vytvoření konvoluční neuronové sítě byl použit MATLAB Deep Learning Toolbox. Pomocí tohoto toolboxu lze navrhovat architektury sítí a jejich parametry. Můžeme k tomu využít pouze grafické rozhraní, nebo vygenerovaný kód na základě vizuálního návrhu.

Proces natrénování konvoluční sítě lze rozdělit na několik specifických úkolů. Rozdělit dataset nahrávek. Data uložit ve vhodném formátu. Nadefinovat vrstvy konvoluční sítě. A vhodně nastavit ostatní parametry sítě. Nahrávky máme z předchozích metod rozděleny na trénovací a testovací sadu (tab.3.1). Ovšem u umělých neuronových sítí je vhodné vyčlenit ještě sadu validační. Jedná se vyčleněná trénovací data, které slouží k nalezení optimálních parametrů modelu během jeho učení.




**Obrázek 4.8:** Rozdělení dat

Trénovací data, která budou vstupovat do sítě musí být v jedné datové struktuře. Tato struktura bude reprezentovat všechny trénovací nahrávky ve spektrogramové (případně kepstrogramové) podobě. Obrazová data jsou reprezentováná třemi rozměry, šířka, výška a počet kanálů. Vyjádříme-li nahrávku spektrogramem, jedná se o obrazová data s rozměry *F x P x 1*. Kde F je počet framů, P počet parametrů (pro spektrum používáme 32 pro kepstrální příznaky 39) a kanál je pouze jeden. Při práci s konvolučními sítěmi používáme sjednocenou délku nahrávek na základě nejdelší z nahrávek $F = 198$. Celkově máme 2993 trénovacích nahrávek. Dáme-li nahrávky "za sebe" do jedné datové struktury, vytvoříme čtyřrozměrný vektor o velikosti $198x32x1x2993$. Máme tedy trénovací data $XTrain : [198x32x1x2993]$. Dále síť potřebuje informaci o jaká data (povely) se jedná. Proto vytvoříme druhý vektor popisující třídu pro všechna vstupní data. Vektor bude mít rozměry $2993x1$ a pojmenujeme ho YTrain. Kde každé pole bude definovat třídu pro příslušný signál vektoru XTrain. Stejným způsobem vytvoříme pro testovací data vektory XTest a YTest a pro validační XValidation a YValidation.

Následuje definice vrstev konovluční sítě. Vytvoříme pole layers do kterého definujeme seřazené vrstvy v pořadí, v jakém požadujeme architekturu sítě (Příloha A.1). Vrstev je na výběr několik desítek [1]. Jsou zde vrstvy vstupní, kterým zadáme rozměry obrazových dat a vrstva která během každé učící epochy data promíchává. Dále konvoluční vrstvy, jimž předáme hodnotu pro velikost filtru a hodnotu určující počet filtrů v jedné vrstvě. Aktivační ReLu vrstvy a Mini Batch Normalizační vrstvy, které se používají mezi konovlučními vrstvami stejně jako ReLu a rozdělují trénovací data do malých částí, přes které se lépe počítá gradient. Max Pooling vrstvy provádějí takzvaný down-sampling, přeložitelný jako zmenšování vstupu. Po konvolučních a poolingových vrstvách většinou následuje jedna nebo víc Fully Connected vrstev, které využívají naučené příznaky a hledají v obrazových datech charakteristické vzory. Posledním jsou vrstvy Softmax a Klasifikační vrstva.

V posledním kroku je třeba nastavit parametry sítě pomocí funkce trainingOptions().(Příloha A.1) Zde lze nastavit hodnoty learning rate, hodnotu mini batch, počet trénovacích epoch. Dále způsob vizuální zobrazení průběhu tréninku, použití validačních dat, případně minimalizační metodu a mnoho dalších.

Původní architekturu sítě a její parametry jsme nastavili na doporučené hodoty podle zdrojů knihoven Deep Learning Toolbox prostředí MATLAB. Sítě byly vždy natrénovany, vyhodnoceny a postupně upravovány. Byly vyzkoušeny desítky různých kombinací. Zde jsou vyobrazeny paramatery sítě s kterou bylo dosaženo nejlepších výsledků.

### 4.4.1 Výsledky

Výsledky se pohybovaly mezi 70% - 87% podle nastavení parametů sítě a její architektury. V práci je vyobrazeno nastavení, které vyprodukovalo nejlepší výsledky.(Příloha A.1)

[1] popis všech vrstev je k nalezení na https://www.mathworks.com/help/deeplearning/ug/list-of-deep-learning-layers.html

- U konvolučních neuronových sítí je použita pouze příznaková sada FBANK a MFCC

**Tabulka 4.4:** Výsledky metody ANN

| | FBANK | MFCC |
|---|---|---|
| původní slovník | 78,43% | 83,47% |
| upravený slovník | 84,09% | 86,03% |

## 4.5 Shrnutí výsledků

Ve finální tabulce výsledků (tab.4.5) můžeme vidět úspěšnost všech 4 metod na zkráceném slovníku s použitím parametrů MFCC. Výsledek metody LTW odpovídá očekávání, jelikož od začátku bylo zřejmé, že náš problém přesahuje využití této jednochué vzdálenostní metody. Naopak výsledek metody DTW je dost překvapivý a až překvapivě dobrý. Proto bylo vyhodnocování této metody několikrát přezkoumáno, ale výsledek byl stále stejný. Výsledek si u této metody samozřejmě nese svou daň a tou je výpočetní náročnost, kdy povel musí být porovnán se 3375 referenčními vzory reprezentovanými vektory o rozměru F x 39, kde F je počet framů. To navzdory kvalitnímu výsledku dělá tuto metodu pro naši aplikaci nepoužitelnou. Vyšší očekávání bylo u následujících metod. Metoda HMM již při prvních pokusech dopadla velice dobře a po upravení slovníku jsme se dostali až na úspěšnost 91,67%. Poslední metodou byly umělé neuronové sítě, konkrétně konvoluční sítě. Zde byla větší možnost konfigurace nastavení i vrstev sítí. Nicméně po několika desítkách natrénovaných sítí s různými parametry jsme se nebyli shopni dostat nad 90% úspěšnost.

- Výsledná tabulka (tab. 4.5) obsahuje úspěšnost pro příznaky MFCC a zkrácenou verzi slovníku (135 povelů)

**Tabulka 4.5:** Výsledná tabulka

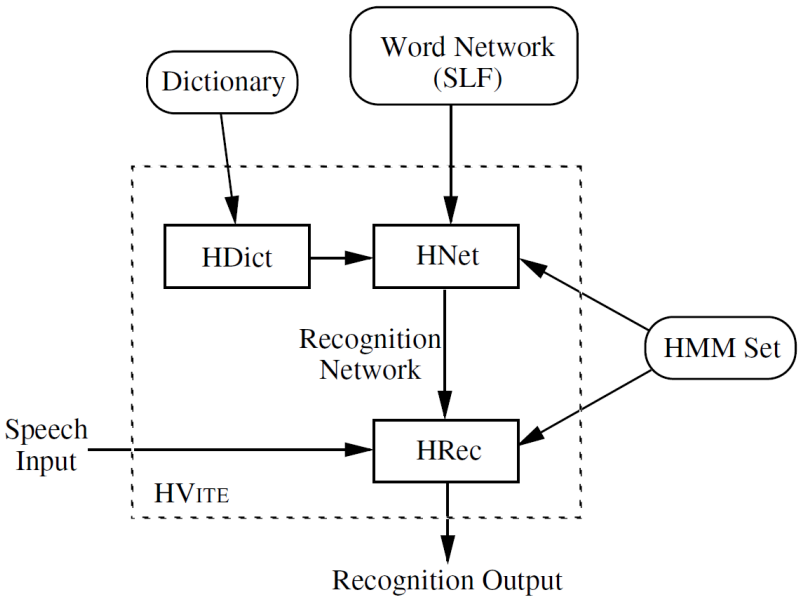
| | LTW | DTW | HMM | ANN |
|---|---|---|---|---|
| MFCC | 36,29% | 89,26% | 91,67% | 86,03% |

Pro aplikaci byla vybrána metoda HMM. Po získání výsledků přes 90% byla tato metoda favoritem pro výslednou aplikaci a to nejen z hlediska úspěšnosti, ale i díky kvalitní a rychlé práci s dobře zdokumentovaným HTK Toolkitem. Je možné, že existuje konfigurace konvoluční neuronové sítě, která by vedla k vyšší úspěšnosti. Bohužel v určitém momentu projektu bylo třeba začít pracovat na finální aplikaci a experimenty dát stranou.

# 5 Aplikace

Po prozkoumání různých přístupů a metod v dané problematice, je dalším cílem této práce aplikace, pomocí které bude uživatel schopen ovládat základní úkony operačního systému Microsoft Winodows. Metodou pro finální aplikaci, jak je patrné z výsledků, se stala metoda skrytých markovských modelů (HMM). Výsledná aplikace tak vznikla spojením modulů HTK Toolkitu a knihoven jazyka Python pro ovládání operačního systému a pro vytvoření uživatelského rozhraní (GUI).

Prvním úkolem bylo vyhodnocování povelů pomocí netrénovaných modelů a jejich následné použití pro ovládání OS. To obstarává nástroj HVite(obr.5.1), který je součástí HTK toolkitu a jeho funkcí je dékodování zaznamenaných povelů, jejich zpracování, porovnání s modely hmm a následné vyhodnocování. Samotný HVite obsahuje několik dalších modulů, které obstarávají dílčí úkoly (HDict, HNet, HRec). HDict poskytuje rozhraní pro přístup ke slovníku. HNet generuje síť HMM modelů. HRec, na jehož vstupu je síť vygenerovaná předchozím modulem a používá ji k vyhodnocování audio signálu z mikrofonu. Výstupem je přímo vyhodnocený povel.




**Obrázek 5.1:** Modul HVite [10]

Naše aplikace zavolá modul HVite pomocí knihovny *Subprocess* a její funkce *Popen()* (zdrojový kód 5.1, řádek 3). Tím dojde k vytvoření roury (pipe) mezi výstupem modulu HVite a naší aplikace. Aplikace poté čeká ná výstup modulu, který přijímá jako vstup. Zároveň aktivní modul HVite čeká na vstup mikrofonu. Dostane-li modul HVite vstupní povel, provede dekódování a přiřazení k jednomu z modelů, které jsou pojmenovány jako samotné povely. Výstupem HVite je například model pojmenovaný "DAVID". Aplikace tento textový výstup přebírá jako vstup.

```
1     calling = 'HVite -H hmm6/hmmdefs -C LiveConfig-MFCC39 -w
2         wordnet -p -70.0 -s 0 dict models0'
3     process = subprocess.Popen(calling, shell=True, stdout=subprocess.
      PIPE)
4     while True:
5         output = process.stdout.readline()
6         if output == '' and process.poll() is not None:
7             break
8         if output:
9             ...
```

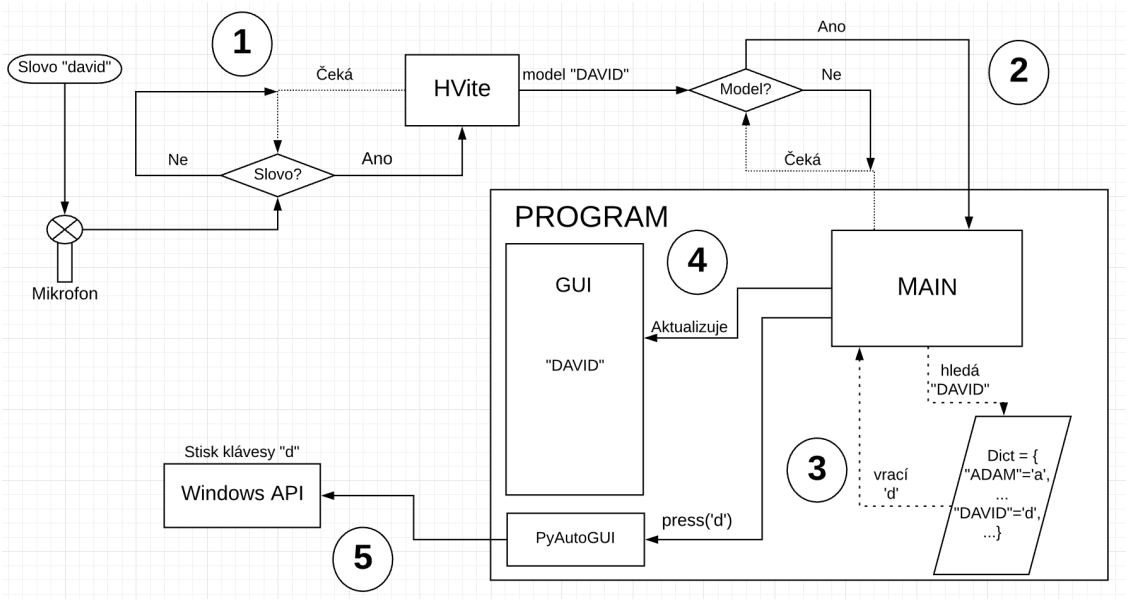
**Listing 5.1:** Funkce volá modul HVite a očekává jeho výstup.Po přijetí výstupu (output není prázdný) přechází do části hledání povelu pro windows API

V aplikaci je vytvořen slovník, který obsahuje povely a jejich vhodné zastoupení pro ovládání hardwarových periferií, konkrétně klávesnice a myši. Například povel "DAVID" ve slovníku reprezentován písmenem "d". Písmeno, list písmen, případně čísel je následně předáno funkcím knihovny PyAutoGui. Tato knihovna má přístup k Windows API (známé také jako WinAPI, nebo win32 API) skrze modul ctypes, proto jsme skrz ni schopni ovládat OS. PyAutoGui obsahuje několik metod pro klik myši, stisk klávesnice, pohyb myši ad. (zdrojový kód 5.2). Tento proces běží v jednom vlákně.

```
1 pyautogui.click(clicks=int(slovo))    # pro klik/dovjklik
2 pyautogui.hotkey(slovo[0],slovo[1])    # stisk klávesové zkratky
3 pyautogui.move( int(slovo[0]),int(slovo[1]))    # pro pohyb
```

**Listing 5.2:** Ukázka volání PyAutoGUI

Druhé vlákno obstarává chod GUI, které běží ve svém vlastním cyklu. V GUI lze vidět nabídku povelů a jaký povel byl zrovna vykonán. Problém nastával při otevření jiné aplikace, která GUI zakryla. Proto bylo rozhrání implementováno tak, aby bylo vždy tím nejvrchnějším oknem na ploše. To s sebou ovšem neslo nevýhodu, kterou je nežádoucí zakrytí důležitých informací ve spodních oknech. Kompromisem tedy bylo vybrat "nejméně" důležitou část monitoru a GUI navrhnout tak, aby zabíralo ce nejméně místa (obr. 5.3). K ovládání GUI jsou třeba 3 pokyny. Je jím posun nahoru a dolu v listu povelů a vypnutí aplikace. Pro tyto účely byly zvoleny ve stejném pořadí povely "page-down", "page-up" a "end". Schéma na obrázku 5.2 popisuje chod aplikace a komunikaci s moduly a knihovnami.




**Obrázek 5.2:** Schéma popisující funkčnost aplikace. (1) modul HVite čeká na hlasový povel (2) aplikace čeká na výstup modulu HVIte v podobě názvu modelu, kterému byl povel přiřazen (3) aplikace prohledává slovník s názvy modulů (povelů) a jejich reprezentaci pro ovládač PyAutoGUI, kterému ji následně předává (4) aplikace vypisuje povel do GUI, případně ovládá GUI a následně jej aktualizuje (5) funkce khivny PyAutoGUI komunikuje s Windows API a předává mu příkaz pro stisknutí klávesy

Aplikace byla implemntována pro operační systém MS Windows, nicméně knihovna PyAutoGui obsahuje moduly, které komunikují nejen s Windows API, ale i s Linuxovým a MAC OS API. Tím pádem by měla být schopna základním způsobem fungovat i na jíných operačních systémech. Problémem by mohly být některé klávesové zkratky, které se vzhledem k OS mohou lišit. Aplikace není bezchybná a její funkčnost vcelku dobře vystihuje úspěšnost nejlepší z metod. Jsme pomocí ní schopni pohybovat se operačním systémem i internetovým prohlížečem, případně psát a upravovat krátké texty. Ale někdy nastane okamžik, kdy aplikace rozpozná správný povel až na několikátý pokus.


**Obrázek 5.3:** Plocha OS Windows 10 se spuštěnou aplikací po vysloveném povelu ”dolu”

# 6 Závěr

Hlavním cílem této práce bylo seznámit se s různými metodami problematiky rozpoznávání řeči, tyto metody implementovat a pomocí vlastních dat je porovnat. Na základě nejúspěšnější z nich vytvořit desktopovou aplikaci, která bude schopna pomocí hlasových povelů ovládat základní vlastnosti operačního systému Microsoft Windows.

Prvním krokem pro splnění stanoveného cíle bylo seznámení se s problematikou rozpoznávání řeči. Byly zkoumány různé metody a přístupy. Nejdříve ty jednodušší porovnávající vzdálenost dvou signálů (LTW a DTW) pomocí kterých bylo snazší proniknout hlouběji do problematiky rozpoznávání. Následovaly metody pracující s pravděpodobnostními modely (HMM), které částečně vycházejí z principů předchozích metod. A v poslední řadě byly zkoumány umělé neuronové sítě.

Po obecném seznámení se s problematikou byl vytvořen slovník povelů vhodných pro ovládání počítače. Slovník obsahuje 176 příkazů vybraných pro intuitivní ovládání OS MS Windows. Dalším krokem byl sběr dat v podobě audionahrávek. Podařilo se nahrát 33 vzorů pro každý z povelů, což bylo dostatečné pro následující požadavky práce, kterými byla parametrizace a implementace jednotlivých metod.

Všechny z metod byly implementovány v prostředí MATLAB až na metodu skrytých markovských modelů, pro kterou byl použit Cambridský HTK toolkit. Pro každou metodu byly vyzkoušeny různé příznaky signálu, případně různé trénovací parametry a architektury u umělých neuronových sítí. Nejúspěšnější z metod se stala metoda skrytých markovských modelů (HMM) s úspěšností 91,47%. Této hodnoty bylo dosaženo po natrénování modelu 3375 nahrávkami získanými od 25 osob a následném testování 1080 nahrávkami získanými od 8 osob. Osoby v testovací sadě byly jiné než v sadě trénovací, proto se jedná o rozpoznávač nezávislý na mluvčím (speaker independence). Testování a výsledky docela dobře simulují případ reálného nasazení rozpoznávače v praxi.

Natrénované modely HMM byly použity pro implementaci závěrečné aplikace, která byla vypracována v jazyce Python. Zde bylo implementováno uživatelské rozhraní knihovnou Tkinter a pomocí modulu HTK toolkitu HVite, který přiřazuje neznámé slovo k jednomu z modelů, byl vytvořen ovladač pro klávesy, klávesové zkratky a pohyb myši.

# A Přílohy

## A.1 Konfigurace konvoluční sítě

```
numF = 12;
layers = [
    imageInputLayer([weight height])        % slouží k vložení obrazu do sítě

    convolution2dLayer(3,numF,'Padding','same') % první konvoluční vrstva s 12 filtry velikosti 3x3
                                                % metoda paddingu 'same'
    batchNormalizationLayer                 % normalizační vrstva ke zrychlení téninku
                                            % a snížení senzitivity inicializace sítě
                                            % s ReLu se používají mezi konvolučními vrstvami
    reluLayer                           % aktivační ReLu vrstva

    maxPooling2dLayer(3,'Stride',2,'Padding','same')    % pooling vrstva zmenšující objem dat průměrováním
                                                        % okolních hodnot(3x3 okolí) s krokem 2

    convolution2dLayer(3,2*numF,'Padding','same')       % 2x12 filtrů
    batchNormalizationLayer
    reluLayer
    convolution2dLayer(3,2*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer
    convolution2dLayer(3,2*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer


    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,4*numF,'Padding','same')       % 4x12 filtrů
    batchNormalizationLayer
    reluLayer
    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer
    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer
    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer
    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer([ceil(weight/8),1])

    dropoutLayer(0.2)                   % náhodně nastaví hodnoty na nuly s danou pravděpododobností
                                            % používí se proti přeučení
    fullyConnectedLayer(numClasses)         % specifikujem počet výstupní tříd, tato síť přenásobí vstup vahami
    softmaxLayer                        %   spočítá pravděpodobnost přiřazení do konkrétní třídy
    weightedClass(classWeights)];           % přiřazuje všem třídám stejnou váhu pro zpětnou propagaci
```

```
miniBatchSize = 80;
validationFrequency = floor(numel(YTrain)/miniBatchSize);
options = trainingOptions('adam', ...   % minimalizační metoda
    'InitialLearnRate',3e-4, ...        % learning rate
    'MaxEpochs',30, ...             % maximální počet trénovacích epoch
    'MiniBatchSize',miniBatchSize, ...  % hodnota miniBatch (rozdělení dat do epoch, po každé výpočet gradientu)
    'Shuffle','every-epoch', ...        % promíchaní dat každou epochu
    'Plots','training-progress', ...    % zobrazení průběhu tréninku
    'Verbose',false, ...            % paramter k vizualizaci procesu
    'ValidationData',{XValidation,YValidation}, ...     % data k validaci
    'ValidationFrequency',validationFrequency, ...      % frekvence validace
    'LearnRateSchedule','piecewise', ...    % postupné snižování learning rate
    'LearnRateDropFactor',0.1, ...          % míra intenzity snižování learning rate
    'LearnRateDropPeriod',20);          % po kolika epochách dochází ke snižování learning rate
```

## A.2 Slovník povelů

| | | | |
|---|---|---|---|
| Adam | té | Plus | Další_slovo |
| Božena | ú | Pomlčka | Stránka_dolů |
| Cyril | vé | Tečka | Stránka_nahoru |
| David | dvojitévé | Čárka | Vyřízni |
| Emil | iks | Dvojtečka | Kopíruj |
| František | ypsilon | Krát | Vlož |
| Gustav | zet | Středník | Na_úplný_začátek |
| Helena | mezera | Vykřičník | Na_úplný_konec |
| Ivan | Escape | Otazník | Konec_editace |
| Josef | Zruš | Podtržítko | Klikni |
| Karel | Backspace | Uvozovky | Dvojklik |
| Ludvík | Zpátky | Zavináč | Na_střed |
| Marie | Tabulátor | Procento | Doleva_1 |
| Norbert | Zpětný_tabulátor | Levá_závorka | Doleva_5 |
| Oto | Nahoru | Pravá_závorka | Doleva_10 |
| Petr | Dolů | Rovná_se | Doleva_20 |
| Quido | Doprava | Vymaž | Doleva_50 |
| Rudolf | Doleva | Delete | Doleva_100 |
| Svatopluk | Mínus | Insert | Doleva_200 |
| Tomáš | Nula | Na_začátek | Doleva_500 |
| Urban | Jedna | Na_konec | Doprava_1 |
| Václav | Dva | Nový_řádek | Doprava_5 |
| Xaver | Tři | Základní_skupina | Doprava_10 |
| Zuzana | Čtyři | Enter | Doprava_20 |
| Vezmi | Pět | Print_Screen | Doprava_50 |
| á | Šest | Caps_Lock | Doprava_100 |
| bé | Sedm | Windows | Doprava_200 |
| cé | Osm | Pause | Doprava_500 |
| dé | Devět | Page_Up | Nahoru_1 |
| é | Funkce_1 | Page_Down | Nahoru_5 |
| ef | Funkce_2 | Home | Nahoru_10 |
| gé | Funkce_3 | End | Nahoru_20 |
| há | Funkce_4 | Adresa | Nahoru_50 |
| í | Funkce_5 | Oblíbené | Nahoru_100 |
| jé | Funkce_6 | Dopředu | Nahoru_200 |
| ká | Funkce_7 | Aktualizovat | Nahoru_500 |
| el | Funkce_8 | První_odkaz | Dolů_1 |
| em | Funkce_9 | Další_odkaz | Dolů_5 |
| en | Funkce_10 | Předchozí_odkaz | Dolů_10 |
| ó | Funkce_11 | Klávesnice | Dolů_20 |
| pé | Funkce_12 | Konec_klávesnice | Dolů_50 |
| kvé | Lomítko | Myš | Dolů_100 |
| er | Lomeno | Konec_myši | Dolů_200 |
| es | Hvězdička | Předchozí_slovo | Dolů_500 |

# Použitá literatura


[1] HEBB, D.O. *The Organization of Behavior: A Neuropsychological Theory*. Taylor a Francis, 2002. ISBN 9781410612403. Dostupné také z: https://books.google.cz/books?id=VNetYrB8EBoC.

[2] MCCARTHY, John. *Arthur Samuel: Pioneer in Machine Learning*. Stanford university infolab. Dostupné také z: http://infolab.stanford.edu/pub/voy/museum/samuel.html.

[3] KOLEKTIV. *ŘEČ A POČÍTAČ principy hlasové komunikace, úlohy, metody a aplikace*. 1. vyd. Ed. JAN NOUZA Zbyněk Koldovský, Robert Vích. TUL, 2009. ISBN 978-80-7372-548-8.

[4] TSIPORKOVA, Elena. *Dynamic Time Warping Algorithm for Gene Expression Time Series*. Ghent: Ghent University, 2015. Dostupné také z: http://www.psb.ugent.be/cbd/papers/gentxwarper/DTWAlgorithm.ppt.

[5] VODIČKA, Radek. *Rozpoznávání izolovaných slov*. 2014. Dostupné také z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=85863. Bakalářská práce. Vysoké Učení Technické v Brně,Fakulta elektrotechniky a komunikačních technologií. Vedoucí práce Petr SYSEL.

[6] VAGASKÝ, Milan. *Metódy automatického rozpoznávania reči*. 2008. Bakalářská práce. Žilinská Univerzita v Žilině, Elektrotechnická fakulta Katedra telekomunikácií. Vedoucí práce Ing. Michal ĎURKOVIČ.

[7] KARPATHY, Andrej. *CS231n: Convolutional Neural Networks for Visual Recognition*. Stanford University. Dostupné také z: https://cs231n.github.io/.

[8] VDUMOULIN. *conv$_a$rithmetic*. Montreal. Dostupné také z: https://github.com/vdumoulin/conv_arithmetic.

[9] BÁRTEK, Luděk. *SIN04: Řečová interakce a sociální sítě*. Masarykovy univerzity. Dostupné také z: https://is.muni.cz/el/1433/podzim2011/SIN04/um/02/foil10.html.

[10] AL., Steve Young et. *The HTK Book*. 3.4. vyd. Cambridge University Engineering Department, 2009.

[11] KENDALL, Graham. *Apollo 11 anniversary: Could an iPhone fly me to the moon?* Ed. INDEPENDENT. 2019. Dostupné také z: https://www.independent.co.uk/news/science/apollo-11-moon-landing-mobile-phones-smartphone-iphone-a8988351.html.

[12] *List of Deep Learning Layers*. MathWorks. Dostupné také z: https://www.mathworks.com/help/deeplearning/ug/list-of-deep-learning-layers.html.