FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Rozpoznávání souvislé řeči s využitím neuronových sítí
Student:	Adam Zvada
Vedoucí:	Ing. Miroslav Skrbek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Provedte rešerši metod pro rozpoznávání souvislé řeči s využitím neuronových sítí. Uvažujte rekurentní neuronové sítě a zvažte také možnost použití neuronových turingových strojů. Na základě rešerše a po dohodě s vedoucím práce vyberte vhodné řešení pro robota NAO. Maximálně využívejte existujících knihoven s implementacemi potřebných metod. Navržené řešení otestujte na reálných datech. Rozsah práce upřesněte po dohodě s vedoucím práce.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 13. února 2018

Czech Technical University in Prague
Faculty of Information Technology
Department of Theoretical Informatics

Bachelor's thesis

Continuous Speech Recognition by Neural Networks

Adam Zvada

Supervisor: Ing. Miroslav Skrbek Ph.D

May 11, 2018

Acknowledgements

THANKS

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 11, 2018 ....

Citation of this thesis

Zvada, Adam. *Continuous Speech Recognition by Neural Networks.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018.

Abstrakt

V několika větách shrňte obsah a přínos této práce v českém jazyce.

Klíčová slova Replace with comma-separated list of keywords in Czech.

Abstract

In my bachelor thesis Summarize the contents and contribution of your work in a few sentences in English language.

Keywords Replace with comma-separated list of keywords in English.

Contents

List of Figures

Introduction

What is speech recognizer? History... Why to build? Why it helps? How i will do it.

Chapter 1

Neural Network

Neural networks have a remarkable ability to derive meaning from complicated data. They can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques[16]. Even though they have been around since the 1950s, it is only in the last decade when they started to outperform robust system or even humans in specify tasks. However, they require a huge amount of training examples and computational power to be trained for preforming a reasonable prediction. Fortunately, GPUs has seen enormous increase in performance[1] and 90% of the data in the world today has been created in the last two years alone, at 2.5 quintillion bytes of data a day[11]. That's why ANN is big topic in Computer Science and in the technology industry and it currently provides the best solutions to many problems such as speech recognition, image recognition, and natural language processing.

1.1 Inspiration in Nature

Artificial neural network (ANN) is heavily inspired by the way how biological neural networks process information in the human brain. Even though our brain is extremely complex and still not fully understand, we just need to know how information is being transferred. The basic building block is nerve cell called *neuron.* It receives, processes, and transmits information through electrical and chemical signals[14]. It's estimated that an average human has 86 billion neurons[5].

As shown on Figure 1.1, *dendrites* are extensions of a nerve cell that propagate the electrochemical stimulation received from other neurons to the cell body. You may think of them as inputs to neuron, whereas neuron's output is called *axon*, a long nerve fiber that conducts electrical impulses away from

[1] GPUs are explicitly designed to handle multiple matrix calculations at the same time. Evaluation and training of artificial neural networks are mostly matrix operations.

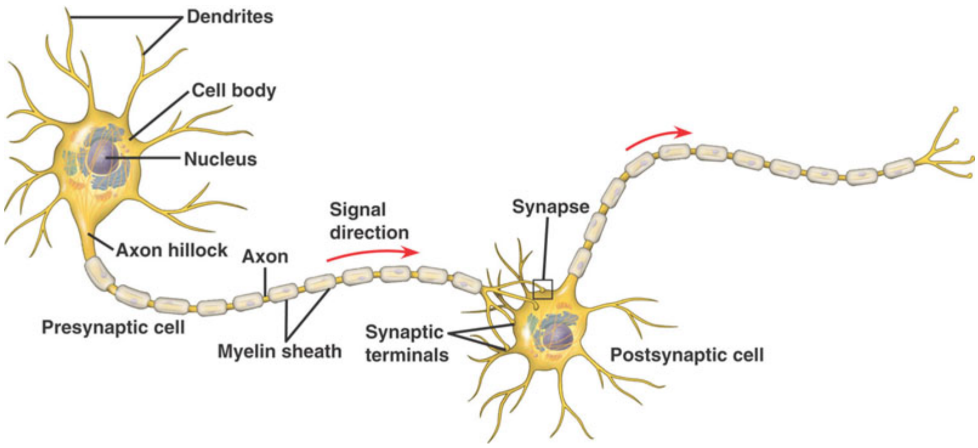

Figure 1.1: Illustration of nerve cell and communication flow

the cell body. The end of axon is branched to many axon terminals which can be again connected to other dendrites. The connection is managed by *synapses* that can permit the passing of electrical signal to cell body. Once the cell reaches a certain threshold, an action potential will fire, sending the electrical signal down the axon to other connected neurons.

1.2 Artificial Neuron

Artificial neuron is a generic computational unit, basic building block for artificial neural network (ANN). It's simplified version of the biological counterpart and we are able to map parts of biological neuron with the artificial one. It takes n inputs represented as a vector $x \in \mathbb{R}^n$ which correspond to dendrites. Generaly artificial neuron produces single output $y \in \mathbb{R}$ as biological neuron where we call it axon. Each neuron's input $i = 1, 2, \ldots, n$ has assigned weight (synapse) $w_1, w_2 \ldots w_n$, they refer to the connection strength between neurons. Weights and same as for synapse are the backbone of learning because in training phases, they keep changing to produce wanted output. Inside the artificial neuron, input vector with their weights are combined and run through an activation function producing some output y. This process is illustrated in Figure 1.2.

1.3 Perceptron

Perceptron is the simplest ANN with just one neuron and since we covered the basic intuition about artificial neuron we may proceed further and take a look at how output is actually calculated. The equation for a perceptron can

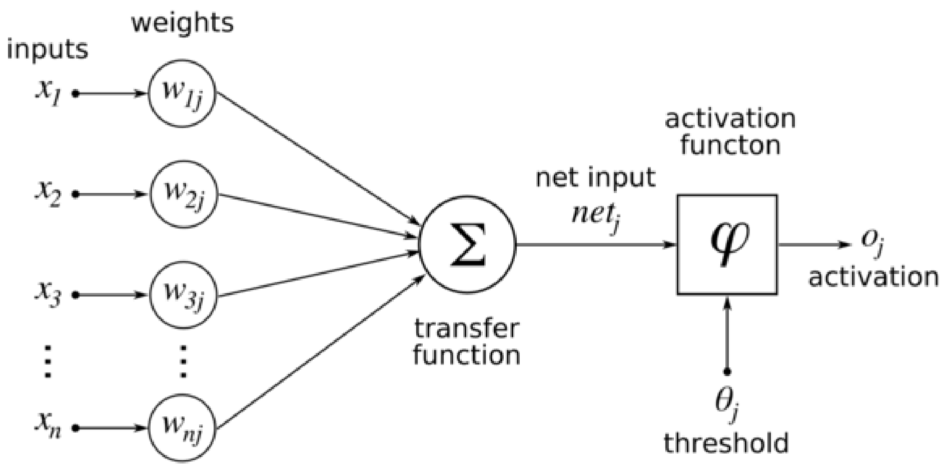

Figure 1.2: Illustration of nerve cell and communication flow

be written as

$$y = f(\sum_{i=1}^{N} w_i \cdot x_i + b) \tag{1.1}$$

where

- x - input vector
- y - predicted output
- f - activation function
- w - weights
- b - bias

Perceptron is a basically linear classifier, therefore the data has to be linearly separable otherwise we would not be able to make the correct prediction. Problems such as speech recognition are not definitely linearly separable, however we can solve non-linear decisions for example by introducing another layer of neurons, thus creating *Multilayered Perceptron.*

1.3.1 Activation Functions

We have stated that biological neuron fires electrical signal to other connected neurons whenever it reaches a certain threshold of incoming electrical impulses. Activation function is based on that concept and inside an artificial neuron it is used for calculating output signal via equation 2.1. It introduces non-linear properties to our ANN and without an activation function would be just a regular linear regression model. Nowadays many different activation function are being used and their performance varies from model to model.

List of some activation function:

- *Sigmoid*
$$\sigma(x) = \frac{1}{1+e^{-x}}$$
- *Hyperbolic Tangent*
$$\tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$$
- *ReLU*
$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$
- *Softmax*
$$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}, \quad i = 1, 2 \ldots J$$
where i is number of output

1.3.2 Bias

We can think of bias as a value stored inside neuron and being used to calculate it's output. The bias value allows the activation function to be shifted to the left or right, to better fit the data.

1.4 Topology of Artificial Neuron Network

Basic ANN as feedforward model is a directed graph with nodes as neurons and edges with weights representing connection to other neurons. ANN can be divided to three important layers as shown in Figure 1.3. Yellow nodes is an input layer which takes input data, dimension of input vector has to correspond to number of input nodes. Hidden layer as the green nodes is most important to ANN and that is where the training and evaluation happens. Number of hidden layers and neurons needs to be in a good ratio between its size and its effectiveness. Output layer produces output vector as the prediction for given input.

1.4.1 Network Evaluation

ANN are sometimes called feedforward neural network. The reason behind is that the input is feeded into the neuron and then forward to another layer, thus ANN are evaluated layer by layer. All neurons calculates the output using similar formula as Perceptron 2.1.

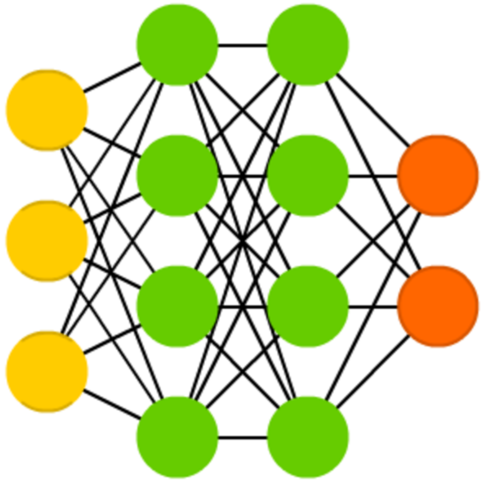

Figure 1.3: Basic topology of fully connected artificial neuron network with input vector of size 3, output vector of size 2 and two hidden layers.

1.5 Training

The greatest trait of ANN is ability to learn from given data and then make the best approximate prediction. The aim of the learning process is to find the most optimal values for network's weights and biases while minimizing error on predicated values. For ANN to learn we have to introduce training data consisted of input vector which will be feeded to the network and desired output value (label) for calculating our loss. This approach is called supervised learning[2].

1.5.1 Loss Function

Loss function compares the prediction from ANN with the desired output and returns the error of the prediction. During a training ANN, the goal is to minimize given loss function. The most common and most intuitive loss function is Mean squared Error (MSE),

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2.$$

1.5.2 Backpropagation

Backpropagation algorithm is responsible for the ability to learn from given training data. It is an iterative algorithm which for each training data from

[2] ANN can be also trained using unsupervised learning.

given training dataset backpropagates the error and adjust the weights and biases accordingly to get desired output.

1.5.2.1 Optimization

Backpropagation requires optimizer to minimize the error on the training data. We will describe backpropagation with using *gradient descent* as the most common optimization algorithm.
Weights and biases are updated using formula,

$$W_{jk}^l := W_{jk}^l - \alpha \frac{\partial E}{\partial W_{jk}^l} b_j^l := b_j^l - \alpha \frac{\partial E}{\partial b_j^l} \tag{1.2}$$

where $\boldsymbol{W_{jk}^l}$ is weight with connection between unit j in layer l and unit i in layer $l+1$, $\boldsymbol{b_j^l}$ is bias associated with unit i in layer $l+1$, $\boldsymbol{\alpha}$ is a learning rate [19], and $\boldsymbol{\frac{\partial E}{\partial W_{jk}^l}}$ or $\boldsymbol{\frac{\partial E}{\partial b_j^l}}$ can be interpreted as minimizing loss function with respect to given weight and bias respectively.

By applying a chain rule twice on the partial derivative of the loss function with respect to a weight, we get

$$\frac{\partial E}{\partial W_{jk}^l} = \frac{\partial E}{\partial a_j^l} \frac{\partial a_j^l}{\partial z_j^l} \frac{\partial z_j^l}{\partial W_{jk}^l} \tag{1.3}$$

where $\boldsymbol{z_j^l}$ is a sum of weighted inputs to unit j in layer l

$$z_j^l = b_j^l + \sum_{k=1}^{K} w_{jk}^l a_k^{l-1} \tag{1.4}$$

and $\boldsymbol{a_j^l}$ is an output of node j in layer l

$$a_j^l = f(z_j^l). \tag{1.5}$$

Let's calculate the last two products of equation 2.3:

$$\frac{\partial a_j^l}{\partial z_j^l} = f'(z_j^l) \frac{\partial z_j^l}{\partial W_{jk}^l} = \frac{\partial W_{jk}^l a_k^l}{\partial W_j k^l} = a_k^{l-1} \tag{1.6}$$

We introduce a new varibale $\boldsymbol{\delta_j^l}$ which represents the error in unit j in layer l and helps us to better understand and calculate real interested value of $\frac{\partial E}{\partial W_{jk}^l}$ and $\frac{\partial E}{\partial b_j^l}$.

$$\delta_j^l = \frac{\partial E}{\partial z_j^l} \tag{1.7}$$

We will simplify the error equation on neuron j in output layer L as

$$\delta_j^L = \frac{\partial E}{\partial z_j^L} = \frac{\partial E}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial E}{\partial a_j^L} f'(z_j^L) \tag{1.8}$$

Now we have enough information to reformulate equation 2.3 for output layer to

$$\frac{\partial E}{\partial W_{jk}^l} = \delta_k^L a_j^L. \tag{1.9}$$

However, to be able to update weights inside the hidden layers, we have to redefine the calculation of δ_j^l. We know that the error produced by an output neuron is just influencing the output value but inside a hidden layer the produced error propagates to all following layers. Therefore we have calculate the δ_j^l where layer l is inside a hidden layer and take into account all δ^{l+1} from following layer $l+1$.

$$\delta_j^l = \frac{\partial E}{\partial z_j^l} = \sum_i \frac{\partial E}{\partial z_i^{l+1}}\frac{\partial z_i^{l+1}}{\partial z_j^l} = \sum_i \frac{\partial E}{\partial z_i^{l+1}}\frac{\partial z_i^{l+1}}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l} = \sum_i \delta_i^{l+1} W_{ij}^{l+1} f'(z_j^l) \tag{1.10}$$

where the sum index i iterates over all neurons in layer $l+1$ and Notice that we have substituted $\frac{\partial E}{\partial z_i^{l+1}}$ with δ_i^{l+1} which is calculated from previous iteration**??**. Finally, we may calculate all weights adjustments through the whole network as

$$W_{jk}^l := W_{jk}^l - \alpha \delta_k^l a_j^l \tag{1.11}$$

where

$$\delta_k^l = \frac{\partial E}{\partial a_j^L} f'(z_j^L), \quad l = L \tag{1.12}$$

or

$$\delta_k^l = \sum_i \delta_i^{l+1} W_{ij}^{l+1} f'(z_j^l), \quad l = 2, \ldots, L-1. \tag{1.13}$$

We won't be exampling the equation for biases adjustments because it follows a similar process shown above with just little changes, resulting to equation

$$b_j^l := b_j^l - \alpha \delta_l^j \tag{1.14}$$

1.5.2.2 Backpropagation Algortihm

Backpropagation algortihm in pseudocode:

Algorithm 1 Backpropagation

```
Initialize network weights and biases
for each training data from training dataset do
    Forward pass and calculate network prediction for given training input
    Calculate error δ^L for output layer
    Calculate errors δ^l for hidden layers
    Update weights and biases using precalculated δ^l
```

Chapter 2

Recurrent Neural Network

Neural networks are powerful learning models that achieve state-of-the-art results in a wide range of machine learning tasks. Nevertheless, they have limitations in the field of sequential data. Standard ANNs rely on the assumption of independence among the training examples but if data points are related in time or space then ANNs would not be the right model for the task[12].
Recurrent neural network (RNN) is type of neural network which is precisely designed to work with sequential data through time. The key difference is that RNN's neurons in hidden layer have a special edge (recurrent edge) to a next time step which can be interpreted as a loop. In RNN, the neuron's output is dependent on the previous computations which is sent through the recurrent edge. Basically, the recurrent edges or loops allow persistence of information from one time step to the next one as shown on Figure 2.1 [6].

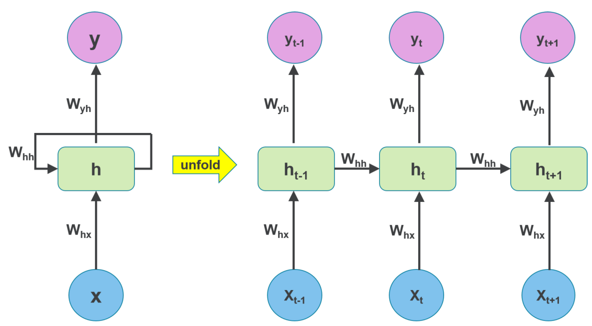

Figure 2.1: Simple RNN topology and illustration of unrolled RNN through time[18]

2.1 Evaluation

In *Figure 3.1.* we may see simplification of evaluation process of RNN through the time steps. RNN's neuron cell in hidden layer takes two inputs, x_t and h_{t-1} which is value (hidden state) sent through the recurrent edge from previous time-step. The cell also produces two outputs, h_t as hidden state for upcoming time-setp

$$h_t = f(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$$

where f is arbitrary non-linear activation function, W_{hx} is matrix of conventional weights, W_{hh} is the matrix of recurrent weights and b_h is a bais. The second output from cell is y_t which outputs the predication using precalculated hidden state h_t,

$$y_t = W_{hy}h_t + b_y$$

where W_{hy} is matrix of output weights.

2.1.1 Softmax Fucntion

It is very common for RNN models to use *softmax* as activation function for output layer. Softmax function helps to get probability distribution of outputs so it's useful for finding most probable occurrence of output with respect to other outputs.

$$\text{softmax}(y)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \quad \text{for } j = 1, \ldots, K$$

Softmax is being used for calculting output value of y_t resulting to formula

$$y_t = \text{softmax}(W_{hy}h_t + b_y).$$

2.2 Training

Training a RNN is similar to training a traditional ANN. We also use the backpropagation algorithm, but since the parameters are shared by all time-steps in the network, the gradient at each output depends not only on the calculations of the current time-step, but also the previous time-steps[3].

2.2.1 Backpropagation Through Time

The most used algortihm to train RNN is *backpropagation through time* (BPTT), introduced by Werbos in 1990 [17]. BPTT is basically an extended version of backpropagation algorithm where we not only propagate the error to all following layers but also through the hidden states. We may think of it as unrolling the RNN to sequence of identical ANNs where the recurrent edge

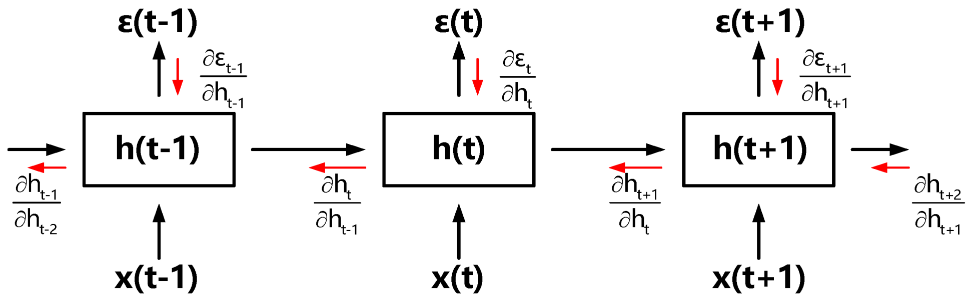

Figure 2.2: Deriving the gradients according to the back-propagation through time (BPTT) method. Notaion for output value $\epsilon(t)$ corresponds to our y_t[13].

connects the sequences of neurons in hidden layer together as shown on Figure 2.1 and 2.2. On Figure 3.2 2.2 is also indicated how the errors are propagated. The propagation of errors through hidden states allows the RNN to learn long term time dependencies. The calculated gradients of the loss function for defined parameter (W, b) through the sequence of unrolled RNN are then sum up, producing the final gradient for updating the weights or biases, *Equation bottom*.

$$\frac{\partial E}{\partial W_i j^l} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W_i j^l}$$

where E is predefined loss function, W_{jk}^l is weight with connection between unit j in layer l and unit i in layer $l+1$, T is number of input sequences and $\frac{\partial E_t}{\partial W_{ij}^l}$ is calculated similarly as in backpropagation with just considering existence of recurrent edges

$$\frac{\partial E_t}{\partial W_{ij}^l} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_{ij}^l}$$

To compute the $\frac{\partial h_t}{\partial h_k}$ we use simple chain rule over all hidden states in interval $[k,t]$.

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}$$

Putting equations together, we have the following relationship[15].

$$\frac{\partial E}{\partial W_{ij}^l} = \sum_{t=1}^{T} \sum_{j=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} (\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}) \frac{\partial h_k}{\partial W_{ij}^l}$$

2.2.2 Exploding and Vanishing Gradients

Even though, RNNs had achieved success in learning short-range dependencies, they haven't been showing any worth mentioning achievement with learning mid-range dependencies. That was mainly cause by problems of *vanishing* and *exploding gradients*, introduced in Bengio in 1994 [2].

The exploding gradient problem occurs when backpropagating the error across many time steps, that could lead to exponentially grow of gradient for long-term components. Basically, a small change in parameters at initial stages can get accumulated through the time-steps resulting to the exponentially grow. The values of weights can become so large as to overflow and result in *NaN* values.

The vanishing gradient problem refers to opposite behavior when the gradient values are shrinking exponentially fast and eventually vanishing completely. Gradient contributions from later time-steps become zero and the states at those steps doesn't contribute so we end up not learning long-range dependencies. Vanishing gradients aren't exclusive to RNNs, they also happen in deep ANN[4].

2.2.2.1 Solutions

To overcome problem with exploding gradient we can apply *gradient clipping* method. The values of the error gradient are checked against a predefined threshold value and clipped or set to that threshold value if the error gradient exceeds the threshold[1].

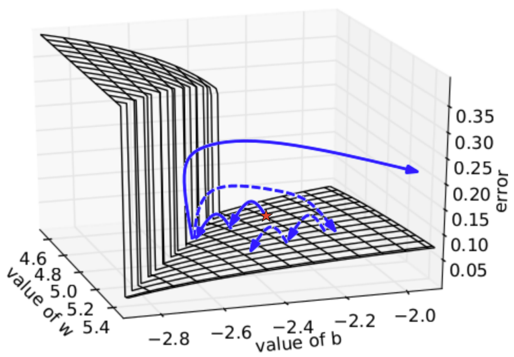

Figure 2.3: Situation of using gradient clipping (dashed line) against the exploding gradient

Another possibility is to use *ReLU* activation function which tends to reduce the the exploding gradient problem. To fix the problem of vanishing gradient is little more complicated. We can always try perform more careful initialization process but it does not always help. It requires different architecture approach achieved by updating the RNN neuron to more complex LSTM cells.

2.3 LSTM

Long-Short-Term-Memories (LSTM) is special kind of RNN cell, introduced by Hochreiter and Schmidhuber in 1997 [9]. Conventional RNNs are only just able to learn short-term dependencies because of vanishing gradient problem. However, LSTM does not get effected and it's capable of learning long-term dependencies.

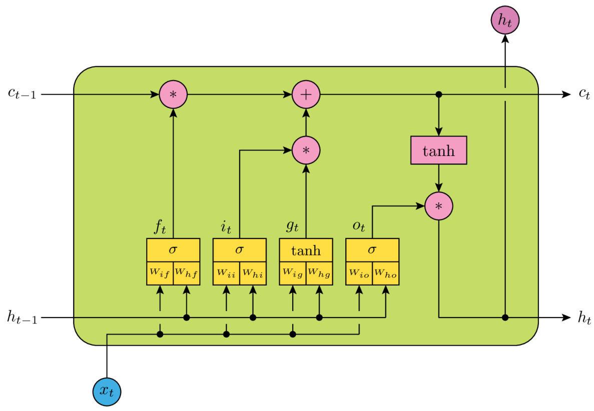

Figure 2.4: Diagram of LSTM cell[10].

As shown on Figure 2.4 we notice that LSTM is just more complex activation units. Similarly as basic RNN cell which propagates hidden state of h_t to another time-step and also as cell output, the LSTM cell has extra state denoted as c_t and called *cell state* and it's just being propagated to another time-step. The cell state is more of a cell's memory.

LSTM architeture follows stages during the evaluation where first we have to decide what information we want to get rid of from cell state, that is achived aplyting formula using sigmoid function

$$f_t = \sigma(W_f h_{t-1} + W_f x_t + b_f) \tag{2.1}$$

and we call f_t as forget gate. Another step is to calculate so-called *input gate* denoted as i_t, it determines whether or not the input is worth preserving.

$$i_t = \sigma(W_i h_{t-1} + W_i x_t + b_i) \tag{2.2}$$

The third value is *memory gate* as g_t, it is using the input with the previous hidden state to observe the input in the context of the past.

$$g_t = \tanh(W_g h_{t-1} + W_g x_t + b_c) \tag{2.3}$$

Using equation 2.1, 2.2 and 2.3 we may calculate the new cell state using formula

$$c_t = f_t c_{t-1} + i_t g_t \tag{2.4}$$

Basically, c_t is constructed by applying the forget gate on the previous cell state and the memory gate gets augmented by the input gate. The last value to produce is hidden state which will be a sort of filtered cell state

$$h_t = \tanh(c_t) o_t \tag{2.5}$$

where o_t is called *output gate* and it augments input information using formula

$$o_t = \sigma(W_o h_{t-1} + W_o x_t + b_o) \tag{2.6}$$

The whole process of the LSTM cell evaluation is also illustrated on Figure 2.4.

2.4 Connectionist temporal classication

Connectionist temporal classication (CTC) is a loss function used for classification of sequential data, initially presented by Alex Graves in 2006 [8]. The idea of CTC is that the label is not generated directly by the RNN, but instead we calculate a probability distribution over all possible characters at every time-step.

For a sequence labelling task where the labels are from an alphabet L, we introduce extra unit as *blank* character, $\hat{L} = L \cup \{blank\}$. CTC consists of a softmax output layer which estimates the probabilities of observing the corresponding labels at particular times[7].

Let's denote that ${y^t}_k$ of output unit k at time-stap t is interpreted as the probability of observing label k at time t and input sequence x of length T. Now we can calculate a probability of path sentence $\pi \in \hat{L}$ using formula

$$p(\pi|x) = \prod_{t=1}^{T} y^t_{\pi_t}. \tag{2.7}$$

Now let's define many-to-one mapping β which simplifies the sentence path by striping the multiple trailing character to just one and then removing the *blank* characters altogether.

$$\beta(- - hh - -e - -ll - lll - oo-) = \beta(-h - e - l - l - o-) = hello$$

We may calculate the marginal probability of the sequence l using the defined β mapping from given path:

$$p(l|x) = \sum_{\pi=\beta^{-1}(l)} (\pi|x) \tag{2.8}$$

This so-called *collapsing together* of different paths onto the same labelling is what allows CTC to use unsegmented data, because it removes the requirement of knowing where in the input sequence the labels occur. However, it also makes CTC unusable for tasks where the location of the labels must be determined[7].

To decode the output for input sequence, we have to maximize the probability of sequence in respect to input data.

$$h(x) = \operatorname*{argmax}_{l} p(l|x) \tag{2.9}$$

For efficent calculation of $p(l|x)$ we use *backward-forward* algorithm with detail explenantion on [8].

To use CTC for RNN training, we have to define the loss function for the BPTT algorithm. CTC loss function is derived from the principle of maximum likelihood with formula

$$E = -ln(\prod_{x,z} p(z|x)) = -\sum_{x,z} ln(p(z|x)) \tag{2.10}$$

where (x, z) are from the training dataset[7].

Chapter 3

Speech Recognition

Speech recognition is the task of converting speech audio to text reprezentation. It has been attracting researchers for many years with a goal to produce efficient speech recognizer, beacuse it's a very easy and natural human-machine interface tool.

Speech recognition system (ASR) takes audio signal as an input and predicts its transcript. ASR are normally divided into two important stages as shown on *Figure 1.1*. The Feature Extractor block generates a sequence of feature vectors which are then feeded to the recognizer block genereting the correct output word.

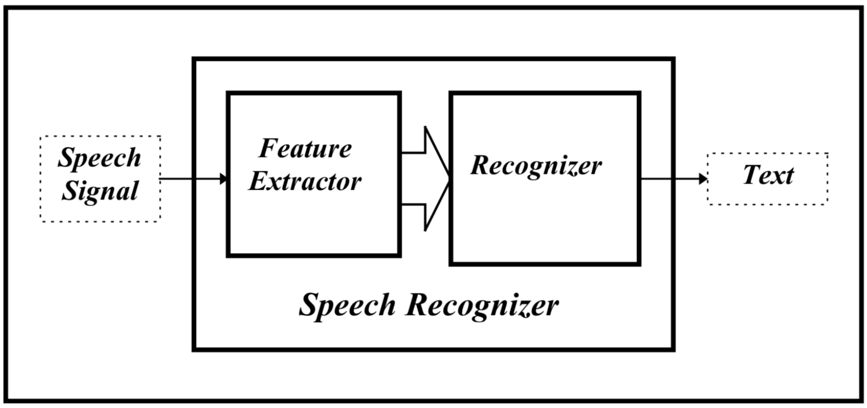

Figure 3.1: Basic building blocks of a Speech Recognizer

3.1 Feature Extraction

The feature extraction (FE) block used in speech recognition should aim towards reducing the complexity of the problem, it should derive descriptive features from speech signal to enable a classification of sounds. It is needed because the raw speech signal contains other information besides the linguistic message which would be counterproductive for recognizer.

3.1.1 Preprocessing

It is adventegous to apply preprocessing to raw speech signal before moving to feature extraction block. Using some type of preprocessing leads to easier feature extraction and faster training phase.

Usually, speech is recorded with a sampling frequency of 44.1kHz (44,100 readings per second). According to The Shannon Theorem *Reference*, a bandwidth limited signal can be reconstructed if the sampling frequency is more than double the maximum frequency meaning that frequencies up to almost 8kHz* are constituted correctly. Other part of preprocessing is to remove the parts between the recording starts and the user starts talking as well as after the end of speech. That helps to speed up the training phase because it reduces the size of training data*Reference*.

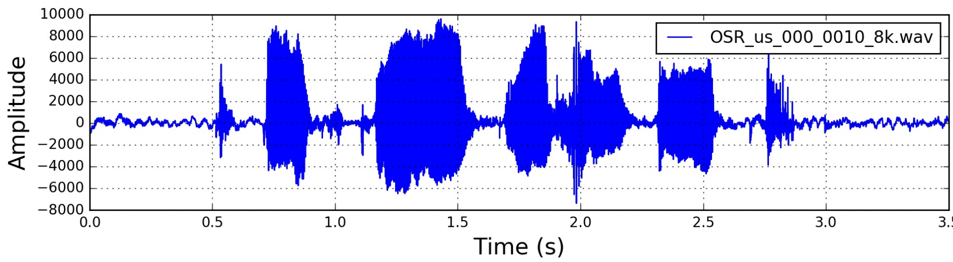

Figure 3.2: Illustration of raw speech signal from wav file with sampling frequency of 8kHz

3.1.2 MFCC

Mel Frequency Cepstral Coefficents (MFCCs) are a feature widely used in speech recognition. They were introduced by Davis and Mermelstein in the 1980's, and have been state-of-the-art ever since.

MFCC mimics the logarithmic perception of loudness and pitch of human auditory system and tries to eliminate speaker dependent characteristics by excluding the fundamental frequency and their harmonics.

To obtain MFFC features we have to follow operation steps as shown on *Figure*:

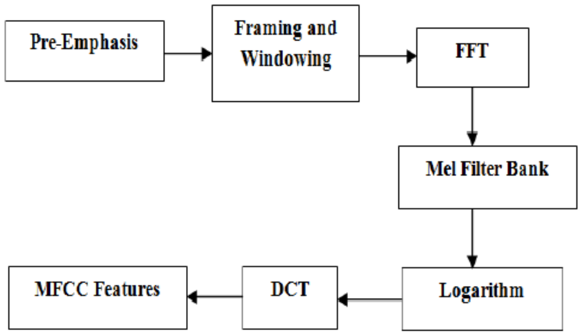

Figure 3.3: Steps of MFCC.

- ***Pre-Emphasis*** - This step applies filter on the speech signal to amplify the high frequencies. It balances the frequency spectrum and avoids numerical problems during the Fourier transform operation.

$$y(t) = x(t) - \alpha x(t-1)$$

  where $x(t)$ is amplitude of signal in time t and α is filter coefficient which typical values are 0.95, $y(t)$ pre-emphasis speech signal.

- ***Framing*** - The process of segmenting the speech sinal into a small frames with the length within the range of 10 to 40 milliseconds. Speech is non-stationary signal but we consider all frames behave stationary so they describe a phonemes. In SR we process overlapping frames because phonemes can dependent, resulting to smoother changes in values. Popular settings are 25 ms for the frame size, 10 ms stride (15 ms overlap)*Reference*.

- ***Windowing*** - This step applies Hamming window function*Refrence* on each speech signal frame. This is common operation for sound signal before applying FFT. *Refernce why*

- ***FFT*** - This step converts all speech frames from time domain into frequency domain using Fast Fourier Transform (FFT).*Reference FFT*

- ***Mel Filter Banks*** - This step applies the mel-filterbank which consists of triangular overlapping windows that are spread over the whole frequency range, outputing mel frequency spectrum. It mimics the non-linear human ear perception of sound, these filters are more discriminative at lower frequencies and less discriminative at higher frequencies.

- ***Logarithm*** - This step computes the logarithm of the mel frequency spectrum, to mimic the human perception of loudness because perceive loudness on a logarithmic scale*Reference*.

- ***DCT*** - This step converts mel spectrum into time domain using Discrete Cosine Transform (DCT)*Reference*, resulting to MFCC vectors.

We have just given a theoretical overview how MFCC is calculated, for more detailed explenation consider reading *Reference MFCC explanation*. On *figure mfcc* is vector of MFCCs calculated from speech signal *figure signal* where number of cepstral coefficents is set to 13. We have extracted the features of speech signal and vectors of MFCCs can be feeded to recognizer.

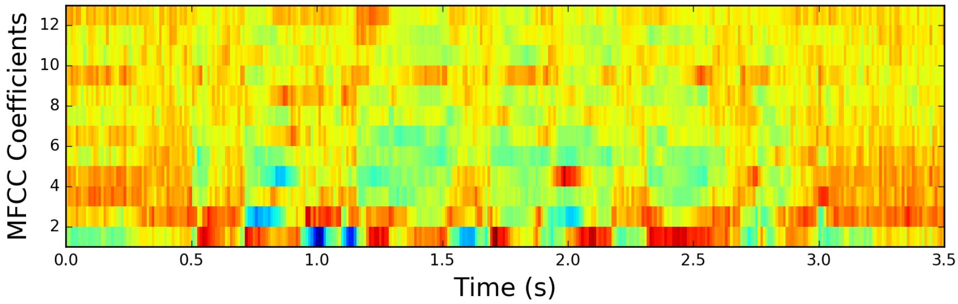

Figure 3.4: Vector of Mel Frequency Cepstral Coefficents through time.

3.2 Traditional Speech Recognizers

Historically, most speech recognition systems have been based on a set of statistical models representing the various sounds of the language to be recognized. We can define a problem of speech recognition as maximizing a probability of the word sequence given some utterance.

$$W^* = \operatorname*{argmax}_W P(W|X)$$

where X are acoustic vectors and transcribed W^* word sequence. However, calculating directly W^* is a very difficult task. We may simplify it by using Bayes rule resulting to equivalent equation

$$W^* = \operatorname*{argmax}_W P(X|W)P(W)$$

where the likelihood $P(X|W)$ is called the acoustic model and the prior $P(W)$ is the language model. In traditional speech recognizers we don't form words directly but we concatenating phonemes which are basic building block of words and they are defined by pronunciation model. As shown on *Figure diagram*, the decoder block works with language, acoustic and pronunciation model. The language model has a word sequences probabilities, while the acoustic model is generated by Hidden Markov Model (HMM) which is a tool

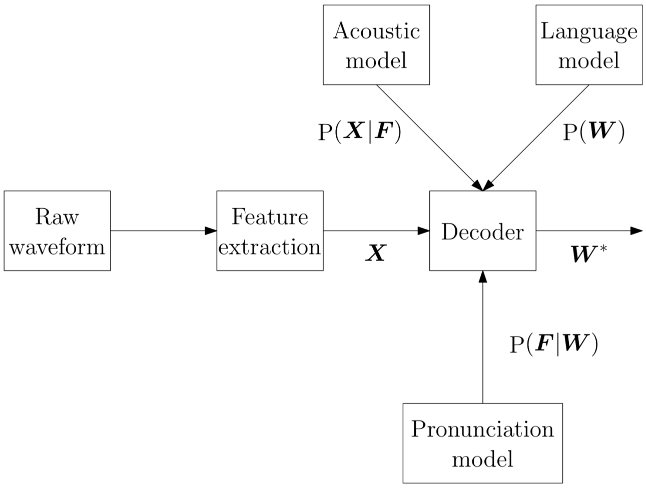

Figure 3.5: Diagram of traditional speech recognizer

for representing probability distribution over sequences of phonemes using pronunciation model.

In this thesis we just give a basic overview how traditional speech recognizers, our main focus is on end-to-end recognizers.

3.3 End-to-End Speech Recognizers

Recent advances in algorithms and computer hardware have made it possible to train neural networks(*section*) in an end-to-end fashion for tasks that previously required significant human expertise. All of the state-of-the-art speech recognizers were HMM-based, they required pronunciation, acoustic and language model which were hand-engineered and trained separetly. Not only speech recognizers based on nerual networks networks require less human effort than traditional approaches, they generally deliver superior performance. Training independent components is complex and suboptimal compared to training all components as one. Because it replaces entire pipelines of hand-engineered components with neural networks, end-to-end learning allows us to handle a diverse variety of speech including noisy environments, accents and different languages. End-to-end speech recognizers simplifies the training and deployment process altogether.

3.3.1 Connectionist Temporal Classification

Connectionist Temporal Classification (CTC) were introduced in *section* First step torwards end-to-end speech recognizers were due to the introduction of recurrent neural network (RNN) *section* since they are designed to deal with sequenctional data through time.

3.3.2 CTC

Connectionist Temporal Classification make it possible to train RNNs for sequence labelling problems where the input-output align- ment is unknown.

Chapter 4

Implementation

The goal is to implement end-to-end speech recognizer using neural network. High-level concept, how the implemented speech recognition system works is illustrated on *figure*. It takes a wav file as an input generated from given microphone and performs preprocessing and feature extraction. The data are feeded to the recognizer which outputs the prediction of transcribed text from speech.

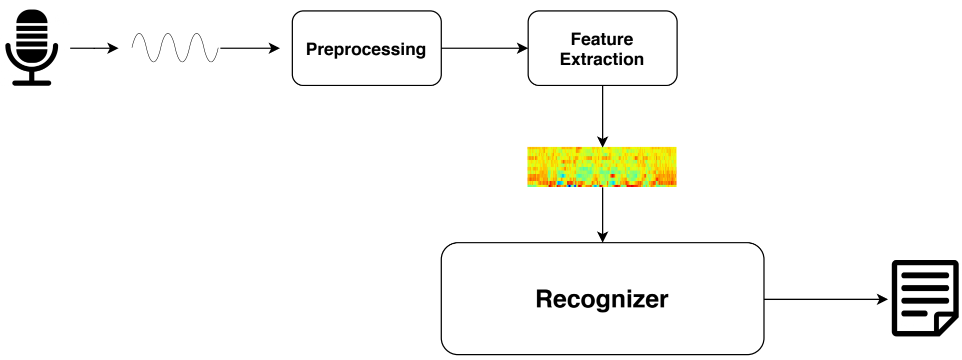

Figure 4.1: Speech Recognition System

The implemented recognizer is build on recurrent neural networks, therefore they need to be trained, in order to make a successful predictions. On *figure 4.2* is shown how the recognizer is being trained. It's done by providing speech and transcribed text from the training dataset. RNN feed-forwards all the vectors of MFCC and the RNN's output are processed by CTC. Obtaining the prediction text of the speech signal. Using backpropagtion through time algorithm we update the weights and biases of RNN which minimize the error of the loss function resulting to better prediction in future.

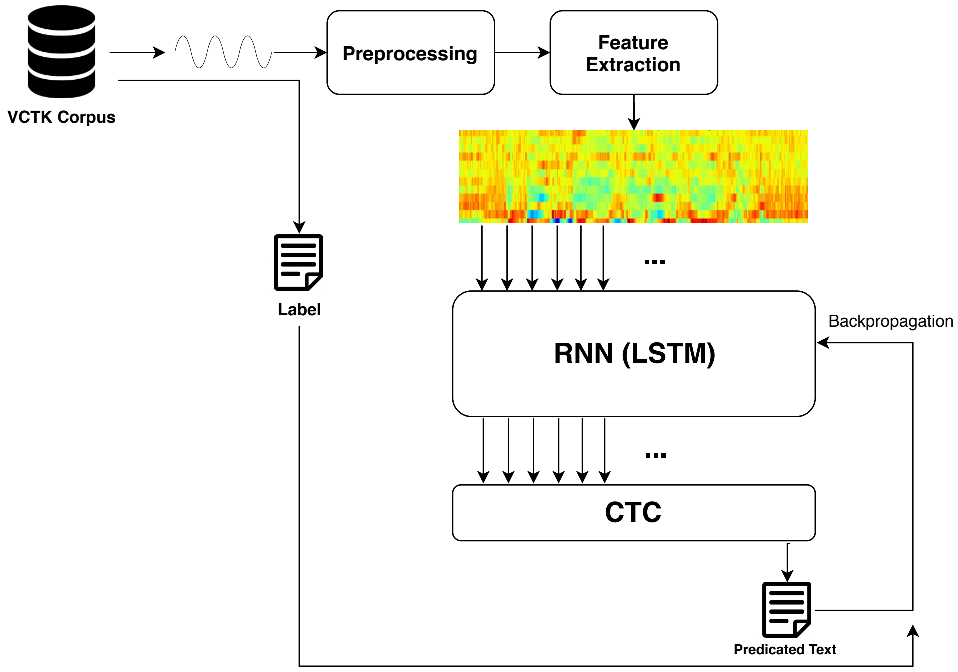

Figure 4.2: Diagram of the learning phase for the speech recognition system

4.1 Tools

4.1.1 Python

Speech recognition system is implemented in programming language `Python` which is currently most popular approach in machine learning and AI. Python is a very powerful, flexible, open source language that is easy to learn. The greatest strength however is wide range of libraries and frameworks for ML and AI.

4.1.2 Tensorflow

TensorFlow is open-source library developed by Google for deep learning and other algorithms involving large number of mathematical operations. The primary unit in TensorFlow is a tensor. A tensor consists of a set of primitive values shaped into an array of any number of dimensions. These massive numbers of large arrays are the reason that GPUs and other processors designed to do floating point mathematics excel at speeding up these algorithms. *Reference*

TensorFlow programs are structured into a construction phase that assembles a computational graph, and an execution phase that uses a session to execute operation in the graph. However, TensorFlow programs are hard to debug because of the structure. Fortunately, TensorFlow offers a built-in function for visualization of the computation called TensorBoard.

4.2 Training Data

Training data are essential for nerual networks performance and its quality, variety, and quantity determine the success of the learning models. Since we use approach of supervised learning for our recognizer, we have to provide labeled data.

4.2.1 Datset Base Class

In source code of the speech recognition system we have class `DatsetBase` which stores path to audios and transactions (labels) from our training dataset. It has also method `next_batch` which takes as a parameter `batch_size` and returns next batch of MFFC vectors and its labels. In method `next_batch` we retrive speech signal data from audio file and perform preprocessing and feature extraction, then also the text labels are loaded from its file path. Upon the text labels is called preprocessing method which simplifies the text and eliminites all the non-alphabetic characters.

However, retriving data from file system and performing processing and feature extraction upon them during a training phase is slowing down the process. One of the solution could be to prepare the data beforehand and store it as some variable which would lead to lower retriving latency.

4.2.2 Numbers

Before using my learning model on large training dataset, I had been debuging and validating it on smaller dataset. I have used `Free Spoken Digit Dataset` from Github *reference*. The dataset provides three english speakers with 1500 recordings, 50 recordings for each digit per speaker.

In source code we have a class `DigitDataset` which extands the base class `DatsetBase`. Class `DigitDataset` provides method called `read_digit_dataset` which takes argument of digit dataset path and stores all training data paths in audios and labels variables. They are later used in `next_batch` method.

4.2.3 VCTK Corpus

VCTK Corpus is training dataset which includes speech data uttered by 109 native speakers of English with various accents. Each speaker reads out about 400 sentences, most of which were selected from The Herald newspaper *regerence*. Even though, this dataset was designed to maximise the contextual and phonetic coverage for HMM-based speech recognizers, we might as well use it for ANN-based speech recognizer. The dataset size is around 15GB which is still not enough to create robust production ready speech recognizer but it's enough for the purpose of this thesis.

4.3 Config Reader

To efficiently use different hyperparametrs, datasets or feature extraction configurations. We run the speech recognizer training with *YAML* configuration file.

4.4 Preprocessing and Feature Extraction

4.4.1 Audio

In source code we have python file `audio_utils` with implements a function called `audiofile_to_input_vector`. The function takes as parameter file path to wav file and the number of cepstrum coefictions. First it loads the wav file from the file system and downsize the sample rate to 16kHz as a part of preprocessing. Even this reduced sample rate contains enough speech information for our recognizer to make successful predication. Then feature extraction is called upon the preprocessed wav file which is done by MFCC. Library `python_speech_features` provides implementation of MFCC method, we just need to configure the used parameters such as the number of cepstrum coefictions, length of window or the length of overlap.

4.4.2 Text

In python file `text_utils` we have function `get_refactored_transcript` which takes string and performs multiple operations for simplification. It converts string to lowercases, eliminites all non-alphabetic characters besides the spaces between words. Then string is converted to *numpy* array of characters which gets encoded to integers values. Thanks to the encoding we can simply calculate the loss function for given text label.

4.5 Recognizer

Recognizer was created by using TensorFlow library. Before we begin to assembles a computational graph we

4.5.1 Computational Graph

TensorFlow requires to assemble a computational graph which will represent the computational steps.

4.5.1.1 CTC Network

In source code we have `CTCNetwork` class representing important features of the network such as input and output dimensions, loss function or used optimizer.

The first method is `generate_placeholders`. *Placeholders* are TensorFlow objects able to store tensors. They don't have to be initilized and input tensors are provided during runtime. Their main purpose is for input and output values. Therefore, the method `generate_placeholders` is creating input and output placeholders for the computational graph. Input placeholder for the network is created as three dimensional array. First dimension represents batch index, second is for number of timestaps and last is for the length of acoustic vector (MFCC vector). For input is also created another placeholder of sequence length for each one on the batched sentences. Output of network is represented by a sparse placeholder because it is required by TensorFlow's CTC.

Second method `loss_funtion` creates CTC loss function inside a computational graph. We use TensorFlow method `tf.nn.ctc_loss` which takes input parameters as a label in sparse matrix format, logits*footnote* which is the last layer of the network and sequence length. The TensorFlow method also performs softmax operation upon the input before applying CTC loss.

The third method is `train_optimizer` is defines the used optimizer in the graph. Optimizer is performing some type of gradient descent algorithm to minimize the error on the loss function. There are many optimizer to choose from but currently the recognizer uses one of the most popular and universal optimzer in deep learning which is *AdamOptimizer*.

Another method is `decoder` which decodes predicated sentence from outputed probabilities using argument of input sequence length placeholder and ouput from last layer. It uses TensorFlow method called `tf.nn.ctc_greedy_decoder`. The same output can be decoded also by using `tf.nn.ctc_beam_search_decoder` but it is little slower than the greedy decoder.

Last method is `compute_label_error_rate` which takes parameter as a decoded sparse label and computes its label error rate.

4.5.1.2 LSTM CTC

Class `LSTMCTC` extends from the `CTCNetwork` class and it defines the inner structure of the network. The constructor sets number of layers, hidden neurons, input dimension and the size of acoustic vector.

The class has method `define` which creates the part of the computation graph. It calls parent method for generating placeholders. Creates LSTM cells using `tf.contrib.rnn.LSTMCell` method for all layers then we stack the cells into multilayer RNN networks with method `tf.contrib.rnn.MultiRNNCell`, the stacked network is used in method `tf.nn.dynamic_rnn` which finalizes it with input placeholders. The method `define` returns the output layer of the network.

4.5.2 Training

Training phase of the recognizer is implemented in file `train.py` by method `train_network` which takes dataset and config reader object. The method first has to read the hyperparametrs of the network from the config reader and then the computational graph is contructed using the `LSTMCTC` methods.

In TensorFlow the computation on created graphs are perfromed inside a `tf.Session()`, thus the training phase is happing inside the session where we loop thorough all the training epoches. In the epoch we train RNN on all training data which are provided using dataset object's method `next_batch`. To run the c we will use function `session.run(fetches, feed_data)`. The *fetches* will be graph operation which are responsible for the training and *feed_data* are network's placeholder with assigned values from `next_batch` method in dictionary structuer. Example code of running the session for backpropagtion algortihm:

```
feed = {
  lstm_ctc.input_placeholder : train_x,
  lstm_ctc.label_sparse_placeholder : train_y_sparse,
  lstm_ctc.input_seq_len_placeholder : train_sequence_length
}

batch_cost, _ = session.run([loss_operation, optimizer_operation], feed)
```

TensorFlow also offers a way of restoring trainned networks. During a training we may save checkpoint files with operations variables because `tf.Variable` maintains state in the graph across the computaions. It's achived by an object `tf.train.Saver()`, upon the object we either call method `save(session, checkpoint_path)` or `restore(session, checkpoint_path)`.

4.6 Robot NAO

Robot Nao is an autonomous, programmable humanoid robot and the goal is to use the implemented speech recognizer as a voice-user interface.

`ALProxy` provides remote connection to the NAO robot and gives us access to all the robot's methods. Speech recognizer will be python module running remotly and using `ALProxy` object we are able to fetch the recorded robot's sound data. The sound data will be processed by the speech recognizer and robot can react to the predicated text.

Figure 4.3: Robot Nao

Chapter 5

Experiments

In this section we will review the speech recognizer performance. We will introduce some optimalization to increase the learning model accuracy and also by tweaking hyperparametrs of the network we can achive better results.

5.1 Computing Power

Training nerual networks could be considred as computationaly difficult problem. However, with the right hardware we can speed up the process significantly. Backpropagtion algorithm is mostly about multiplying matrices and GPUs are explicitly designed to handle multiple matrix calculations at the same time, therefore it is highly recommanded to use GPUs for training nerual networks.

Unfortunately, TensorFlow is just limited on using NVIDIA GPUs to properly work beause the python library `tensorflow-gpu` which handles the TensorFlow GPUs computaions is build upon CUDA toolkit. Therefore, I will be using CPU for the experiments section as the main computational resource. Because it would not be possible to train speech recognizer on the whole *VCTK dataset*, for the experiment part I will use *Free Spoken Digit Dataset*.

The final training of the speech recognizer using *VCTK dataset* is done on *Floyd Hub* which is commercial platform for cloud computing.

5.1.1 Floyd Hub

FloydHub is a Platform-as-a-Service for training and deploying deep learning models in the cloud. *TODO*

5.2 First training

The hyperparametrs of the first training the speech recognizer on *Free Spoken Digit Dataset*:

- numer of hidden neruons - 100
- number of hidden layers - 1
- batch size - 8
- number of epoches - 150
- learning rate - 0.001
- dimension of acoustic vector - 13

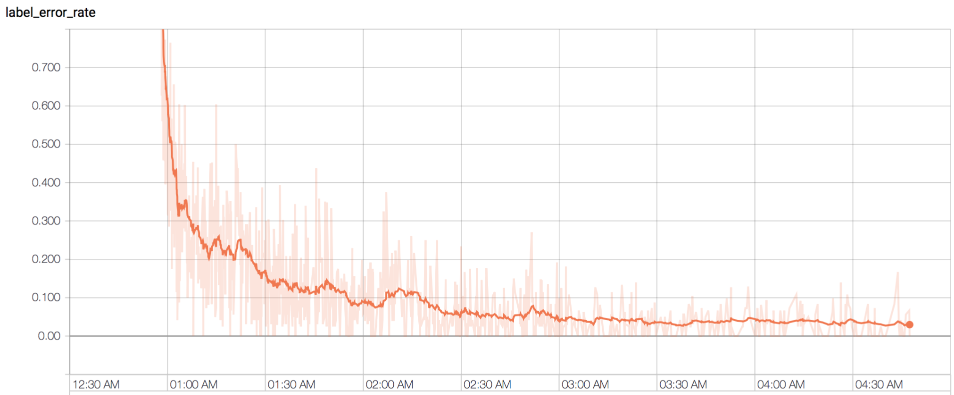

Figure 5.1: Learning Error rate for *TODO*

Validation of the speech recognizer performance is evaluated using label error rate. On *Figure 5.1.*

5.3 Dropout

Optimalization of the learning model can be achived by introducing dropout method.

Conclusion

The goal of the thesis was to implement speech recognizer using nerual networks which will be later used as voice-user interface on Robot NAO.

First we have introduced the concept of artifical neural networks and basics of the internal network architecture. We have explained the training phase of artifical neural networks and how backpropagtion algortihm is used to modify the weights and baises. We have extanded the knowleadge of nerual networks by introducing recurrent nerual networks and explained the improved version of backpropagtion algortihm, compatible with RNN. However, RNNs presented a problem with vanishing and exploding gradient and we explained a possible solution in type of LSTM cells as an improvment upon basic RNN's neuron.

The speech recognition systems has drasticly changed over the last decade. We have given an overview of HMM-based speech recognition system and talked about how the field is starting to shift to end-to-end speech recognizer systems.

We have introduced the speech recognition system based on reccurent nerual networks with LSTM cells and CTC loss function on the output layer and propoused the implementation of such a system using TensorFlow library. Validation of such a model was first performed on audio number dataset where

In future work we want to finish the integration of implemented speech recognition with Robot NAO and most imporantly traine it on more advance speech corpus with deeper network.

References

[1] *A Gentle Introduction to Exploding Gradients in Neural Networks.* URL: `https://machinelearningmastery.com/exploding-gradients-in-neural-networks/` (visited on 09/27/2017).

[2] Y. Bengio, P. Simard, and P. Frasconi. "Learning Long-term Dependencies with Gradient Descent is Difficult". In: *Trans. Neur. Netw.* 5.2 (Mar. 1994), pp. 157–166. ISSN: 1045-9227. DOI: `10.1109/72.279181`. URL: `http://dx.doi.org/10.1109/72.279181`.

[3] Denny Britz. *Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs.* URL: `http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/`.

[4] Denny Britz. *Recurrent Neural Networks Tutorial, Part 3 – Backpropagation Through Time and Vanishing Gradients.* URL: `http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/`.

[5] Kendra Cherry and Steven Gans. *How Many Neurons Are in the Brain?* URL: `https://www.verywellmind.com/how-many-neurons-are-in-the-brain-2794889`.

[6] Filipe. *Text Generation with Recurrent Neural Networks (RNNs).* URL: `https://blog.paperspace.com/recurrent-neural-networks-part-1-2/` (visited on 01/04/2017).

[7] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks.* Vol. 385. Jan. 2012.

[8] Alex Graves et al. "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks". In: *Proceedings of the 23rd International Conference on Machine Learning.* ICML '06. ACM, 2006, pp. 369–376. ISBN: 1-59593-383-2. DOI: `10.1145/1143844.1143891`. URL: `http://doi.acm.org/10.1145/1143844.1143891`.

[9] Sepp Hochreiter and JÃŒrgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. URL: `https://doi.org/10.1162/neco.1997.9.8.1735`.

[10] Andre Holzner. *LSTM cells in PyTorch*. URL: `https://medium.com/@andre.holzner/lstm-cells-in-pytorch-fab924a78b1c` (visited on 08/10/2018).

[11] Ralph Jacobson. *2.5 quintillion bytes of data created every day. How does CPG Retail manage it?* URL: `https://www.ibm.com/blogs/insights-on-business/consumer-products/2-5-quintillion-bytes-of-data-created-every-day-how-does-cpg-retail-manage-it/`.

[12] Z. C. Lipton, J. Berkowitz, and C. Elkan. "A Critical Review of Recurrent Neural Networks for Sequence Learning". In: *ArXiv e-prints* (May 2015). arXiv: `1506.00019 [cs.LG]`.

[13] Ye Ma et al. "Reconstruct Recurrent Neural Networks via Flexible Sub-Models for Time Series Classification". In: *Applied Sciences* 8.4 (2018). ISSN: 2076-3417. DOI: `10.3390/app8040630`. URL: `http://www.mdpi.com/2076-3417/8/4/630`.

[14] *Neuron*. URL: `https://en.wikipedia.org/wiki/Neuron` (visited on 08/10/2018).

[15] R. Pascanu, T. Mikolov, and Y. Bengio. "On the difficulty of training Recurrent Neural Networks". In: *ArXiv e-prints* (Nov. 2012). arXiv: `1211.5063 [cs.LG]`.

[16] Christos Stergiou and Dimitrios Siganos. *NEURAL NETWORKS*. June 29, 2007. URL: `https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#Introduction%20to%20neural%20networks`.

[17] Paul Werbos. "Backpropagation through time: what it does and how to do it". In: 78 (Nov. 1990), pp. 1550–1560.

[18] Huiting Zheng, Jiabin Yuan, and Long Chen. "Short-Term Load Forecasting Using EMD-LSTM Neural Networks with a Xgboost Algorithm for Feature Importance Evaluation". In: *Energies* 10.8 (2017). ISSN: 1996-1073. DOI: `10.3390/en10081168`. URL: `http://www.mdpi.com/1996-1073/10/8/1168`.

[19] Hafidz Zulkifli. *Understanding Learning Rates and How It Improves Performance in Deep Learning*. URL: `https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10`.

Appendix A

Acronyms

ANN Artifical Nerual Network

RNN Recurrent Nerual Network

CTC

MFCC

SR

GPU

CPU

Appendix B

Contents of enclosed CD

```
readme.txt ........................ the file with CD contents description
exe ...................................... the directory with executables
src ...................................... the directory of source codes
  wbdcm ..................................... implementation sources
  thesis .............. the directory of LaTeX source codes of the thesis
text ......................................... the thesis text directory
  thesis.pdf .......................... the thesis text in PDF format
  thesis.ps ............................ the thesis text in PS format
```