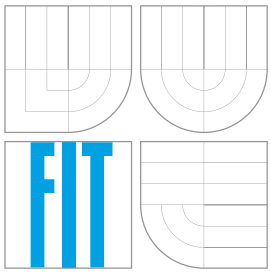
BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

RECOGNITION OF AUDIO EVENTS USING DEEP NEURAL NETWORKS

ROZPOZNÁVÁNÍ ZVUKOVÝCH UDÁLOSTÍ POMOCÍ HLUBOKÝCH NEURONOVÝCH SÍTÍ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR
AUTOR PRÁCE

ALBERT UCHYTIL

SUPERVISOR
VEDOUCÍ PRÁCE

Ing. PETR SCHWARZ, Ph.D.

BRNO 2016

Bachelor Project Specification/18850/2015/xuchyt03

Brno University of Technology - Faculty of Information Technology

Department of Computer Graphics and Multimedia Academic year 2015/2016

Bachelor Project Specification

For: **Uchytil Albert**
Branch of study: Information Technology
Title: **Recognition of Audio Events Using Deep Neural Networks**
Category: Speech and Natural Language Processing

Instructions for project work:

1. Study deep neural networks, their variants and their use for speech and audio processing.
2. Get acquainted with available tools for DNNs.
3. Select appropriate data and generate suitable labels.
4. Implement a method for audio event recognition using one of available tools.
5. Compare the results of several approaches and discuss the results.
6. Create a poster and/or video presenting your work.

Basic references:

- according to supervisor's recommendation.

Requirements for the first semester:
Items 1 to 3, partially item 4

Detailed formal specifications can be found at http://www.fit.vutbr.cz/info/szz/

The Bachelor Thesis must define its purpose, describe a current state of the art, introduce the theoretical and technical background relevant to the problems solved, and specify what parts have been used from earlier projects or have been taken over from other sources.

Each student will hand-in printed as well as electronic versions of the technical report, an electronic version of the complete program documentation, program source files, and a functional hardware prototype sample if desired. The information in electronic form will be stored on a standard non-rewritable medium (CD-R, DVD-R, etc.) in formats common at the FIT. In order to allow regular handling, the medium will be securely attached to the printed report.

Supervisor: **Schwarz Petr, Ing., Ph.D.**, DCGM FIT BUT
Beginning of work: November 1, 2015
Date of delivery: May 18, 2016

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

L.S.

Jan Černocký
Associate Professor and Head of Department

Abstract

A lot of information is carried in sound. The amount of audio data is increasing with a growing technical level of the society. With more data, the task of processing it gets harder for human beings. This thesis is about recognition of audio events using neural networks. We focused on classification of phonemes and their categories. We used the *Multilayer perceptron* model as a classifier. We examined the relation between the accuracy of the model and its properties. Our goal was to estimate the network setup to obtain the best results. The accuracy is influenced by input features. We examine the relation between a type of the features and the success rate. The differences between input feature types are reduced by using the context. The bigger context we use the better results we get. Problem is, when contexts overlap, overlapping leads to a higher error rate. We have used a neural network with three hidden layers.

Abstrakt

Zvuk je nositelem velkého množství informací. S rostoucí technickou úrovní společnosti se zvyšuje množství zvukových dat. Čím více dat máme, tím hůře se člověku zpracovávají. Tato práce se zabývá problematikou rozpoznávání zvukových událostí pomocí neuronových sítí. Konkrétně klasifikaci fonémů a jejich kategorií. Jako klasifikátor se používá model *vícevrstevného perceptronu*. Práce zkoumá závislost přesnosti tohoto klasifikačního modelu na nastavených vlastnostech a hledá optimální nastavení pro maximální přesnost. Přesnost je ovlivněna také vstupními daty. Práce zkoumá vztah mezi typem vstupních dat a úspěšností klasifikačního programu, a porovnává vlastnosti vybraných typů vstupních dat. Použití kontextu u vstupních dat redukuje rozdíly námi vybranými typy vstupních prvků. Čím větší kontext použijeme, tím větší přesnosti docílíme. Problém nastává v situaci, kdy začne kontext zasahovat do jiných tříd. Pro naše experimenty jsme používali neuronovou síť se třemi skrytými vrstvami.

Keywords

Sound recognition, Audio classification, Neural Networks, Phoneme classification

Klíčová slova

Rozpoznávání zvuku, Klasifikace audia, Neuronové sítě, Klasifikace fonémů

Reference

UCHYTIL, Albert. *Recognition of Audio Events Using Deep Neural Networks*. Brno, 2016. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Schwarz Petr.

Recognition of Audio Events Using Deep Neural Networks

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Mr. Ing. Petr Schwarz, Ph.D. The supplementary information was provided by Mr. Mgr. Lucas Ondel. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Albert Uchytil
May 18, 2016

Acknowledgements

I would like to thank my advisor Mgr. Lucas Ondel for the guiding me on my way to create this thesis. Without him this thesis would not exist. You have helped me a lot to understand how do Neural Networks work. Without this help, studying the theory for this thesis would be much tougher.

I would like to thank my supervisor Ing. Petr Schwarz PhD. for helping me to make this thesis real. Thank you for proving help every time I was in need.

I would like to thank my family for the support and constant encouragement I got over the years.

Contents

Chapter 1

Introduction

In recent years, there has been an exponential growth of produced data. [14] This growth affects audio data as well. It becomes harder for humans to process this data. *According to staff from the Sound Effects Library, it would take 60 years for a librarian to tag a collection of 2 million sounds.* [10] That is quite a lot of time. The librarian might not fulfill his potential, because he spent all of his time tagging sound. In this period of time, great things can get invented. That is why we do not want to waste time of the librarian.

The task of the librarian consists of two major parts, classifying the sound and writing down what sound it is. If we examine these two actions closer. We come to an assumption that, when the librarian classifies the sound (he knows what sound it is), writing it down is a rather trivial task. Our goal is to design a classifier to help the librarian live a better life.

Our motivation is also economical. Paying a person for 60 years to annotate sound is pretty expensive. Telling a computer to do it is much cheaper and faster.

At first we need to choose which mathematical model to use. Neural Networks are getting more and more interest since 90's.[12] They are used to recognize patterns, this recognition can be used for classification. When we classify a sound sample, we can write down an annotation.

We decided to help the librarian by creating a phoneme classifier. To design this system we needed to take some steps. At first we needed to understand the theoretical background of how Neural Networks work. Then, we had to analyze the data and how to use it. After that, we applied our knowledge to design the classifier. With the classifier built, we were able to run a number of experiments. We were able to tune our classifier according to the obtained results. All of these steps are presented in the upcoming chapters.

Chapter 2

Theory

In this chapter, we present important theory that is needed to accomplish this thesis.

2.1 Probability theory

The theory of probability is important for machine learning. Some classifiers can give us a probability of an instance of data being in a class.

2.1.1 Introduction

Lets suppose we have two events A and B, we will use them to explain two probability terms. First one is *joint probability*. It is written as $p(A, B)$, and it means „the probability of A and B“. This term gives us the probability of two events occurring together. It is the probability of an intersection between two or more events happening at the same time. On the other hand, the *conditional probability* stands for „the probability of A given B“, and is written as $p(A|B)$. It is a probability of event A occurring, given that event B occured.

These two probability types are used in some important rules of probability theory.

2.1.2 Important rules

There are two probability rules which are essential. The *sum rule* 2.1 and the *product rule* 2.2.

$$\textbf{sum rule } p(X) = \sum_{Y} p(X, Y) \tag{2.1}$$

$$\textbf{product rule } p(X, Y) = p(Y|X)p(X) \tag{2.2}$$

The *product rule* 2.2 means that the probability of two events occurring at the same time is equal to the probability of an event occurring times the probability of Y given X. These two simple rules provide the basis for all of the theory of probability used in machine learning.

One useful property of the joint probability is a symmetry:

$$p(X)p(Y) = p(Y)p(X) \tag{2.3}$$

This equation can be written as:

$$p(X, Y) = p(Y, X) \tag{2.4}$$

2.1.3 Bayes' theorem

Bayes' theorem 2.6 plays a very important role in the theory of probability. It gives us the probability of an event, based on conditions that might be related to the event.

Let's suppose that we want to know some individual probability of a person being bald. If being bald is related to age or gender, than the probability can be expressed more precisely using the theorem. To derive the theorem, let's we start with the *product rule* 2.2. Using its symmetry property, we get

$$p(Y|X)p(X) = p(X|Y)p(Y) \tag{2.5}$$

After that we can divide the whole equation by $p(X)$ to obtain the *Bayes' theorem*

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)} \tag{2.6}$$

Bayes' theorem can help a lot to understand the Logistic regression 2.2.

Let's apply the *sum rule* 2.1 and the symmetry to express $p(X)$

$$p(X) = \sum_{Y} p(X|Y)p(Y) \tag{2.7}$$

Now we are able to substitute 2.7 to 2.6 and derive the final form of the *Bayes' theorem*

$$p(Y|X) = \frac{p(X|Y)p(Y)}{\sum_{Y} p(X|Y)p(Y)} \tag{2.8}$$

This form of the theorem is used in the Logistic regression.

2.2 Logistic regression

Logistic regression (LR) is composed of two major parts. The first one is the regression part and a *linear discriminant function.*

2.2.1 Introduction to logistic regression

To make the understanding of the *Logistic regression model* easier, let's start with a 2 class classification problem a person attending a university. We have two classes, „attending a university" (C_1) and „not attending a university" (C_2). We don't actually know a lot about the person. We might know whether his or her parents went to university, probably his or her gender and age. These variables affect the probability whether the event is assigned to class C_1 or C_2. We call them features.

Using the *Bayes' theorem* 2.8, we can express the posterior probability of class C_1 as

$$p(C_1|x) = \frac{p(x|C_1)p(C_1)}{p(x|C_1) + p(x|C_2)p(C_2)} \tag{2.9}$$

We can further factorize the expression to obtain

$$p(C_1|x) = \frac{1}{1 + \frac{p(x|C_2)p(C_2)}{p(x|C_1)p(C_1)}} \tag{2.10}$$

Lets define a relation

$$exp(-a) = \frac{p(x|C_2)p(C_2)}{p(x|C_1)p(C_1)} \tag{2.11}$$

To get to the value of a, we take the logarithm of both sides of the equation and multiply it by -1.

$$a = -\ln \frac{p(x|C_2)p(C_2)}{p(x|C_1)p(C_1)} \tag{2.12}$$

Then we apply the logarithmic rule $-\ln x = \ln x^{-1}$.

$$a = \ln \frac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)} \tag{2.13}$$

In the next step, we can apply the substitution from equation 2.11 to 2.10, and get the *logistic sigmoid* function defined by

$$\sigma(a) = \frac{1}{1 + exp(-a)} \tag{2.14}$$

This function maps the real axis, the infinite values, into the interval from 0 to 1. The shape of the function looks like letter 'S', that is why it is called Sigmoid. This can be seen in Figure 2.2.

The inverse of the *Logistic sigmoid* function is given by

$$a = ln(\frac{\sigma}{1 - \sigma}) \tag{2.15}$$

and is known as the *Logit* function. It represents the log of the ratio of probabilities $\ln[p(C_1|x)/p(C_2|x)]$ for the two classes, also known as the *log odds*. [3, p. 197]

2.2.2 Linear discriminant function

Linear discriminant function is an important part of *Logistic regression*. Thanks to this function, we able to decide to which class the member belongs. For 2 class problems, we can define it as

$$y(x) = \mathbf{w}^T x + b \tag{2.16}$$

where x is an input vector, $\mathbf{w}$ is a *weight vector* and the b is a *bias*. The negative bias $-b$ can be called *threshold*. In our case, the goal of the linear discriminant function is to create a line which separates two classes. As we can see in Figure 2.1. The result of 2.16 tells us to which class the input vector x belongs. If $y(x) \geq 0$ then x belongs to C_1 otherwise it is C_2. According to the value of the result we are able to see how far from the separating line the point is. This means that we are able to tell a probability of a point belonging to the other class. The point A in the Figure 2.1 has definitely a higher chance of belonging to the class C_1 than the point B. The decision boundary is defined by $y(x) = 0$. The vector $\mathbf{w}$ determines the orientation of the decision surface.

2.2.3 Multiclass logistic regression

Multiclass logistic regression is able to separate multiple classes, not just two. However to achieve that we need to generalize the *Bayes' theorem* and the *linear discriminant function* for two classes, shown in the equation 2.9. We can use the *Bayes' theorem* from equation

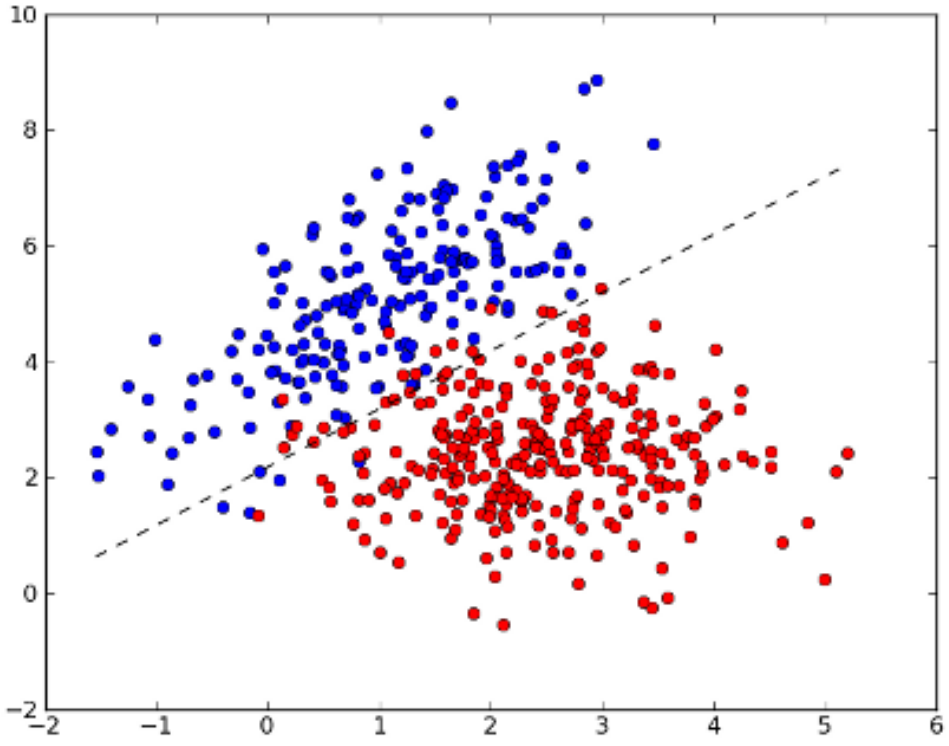


Figure 2.1: *Linear discriminant function* separating two classes.
Source: http://mlpy.sourceforge.net/docs/3.1/lin_class.html

2.8. For a number of classes K greater than 2, the posterior probability of a class C_k can be written as

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{\sum_j p(x|C_j)p(C_j)} \tag{2.17}$$

and further adjusted in a similar way as in 2.14.

$$p(C_k|x) = \frac{exp(a_k)}{\sum_j exp(a_j)} = \sigma(a)_j \tag{2.18}$$

This function is generalized *Logistic sigmoid* and is called *Softmax*.

To be able to classify the number of K classes, for K $>$ 2, we need to adapt the *linear discriminant function* 2.16. We can do that by creating a single K-class discriminant consisted of K linear functions in a form

$$y_k(x) = \mathbf{w}_k^T x + b_{k0} \tag{2.19}$$

Assigning a point x to class C_k if $y_k(x) > y_j(x)$ for all $k \neq j$, the decision boundary between class C_k and class C_j is given by $y_k(x) = y_j(x)$ and hence corresponds to a (D−1)-dimensional hyperplane defined by

$$(\mathbf{w}_k - \mathbf{w}_j)^T x + (b_{k0} - b_{j0}) \tag{2.20}$$

This has the same form as the decision boundary for the two-class case. [3, p.186]

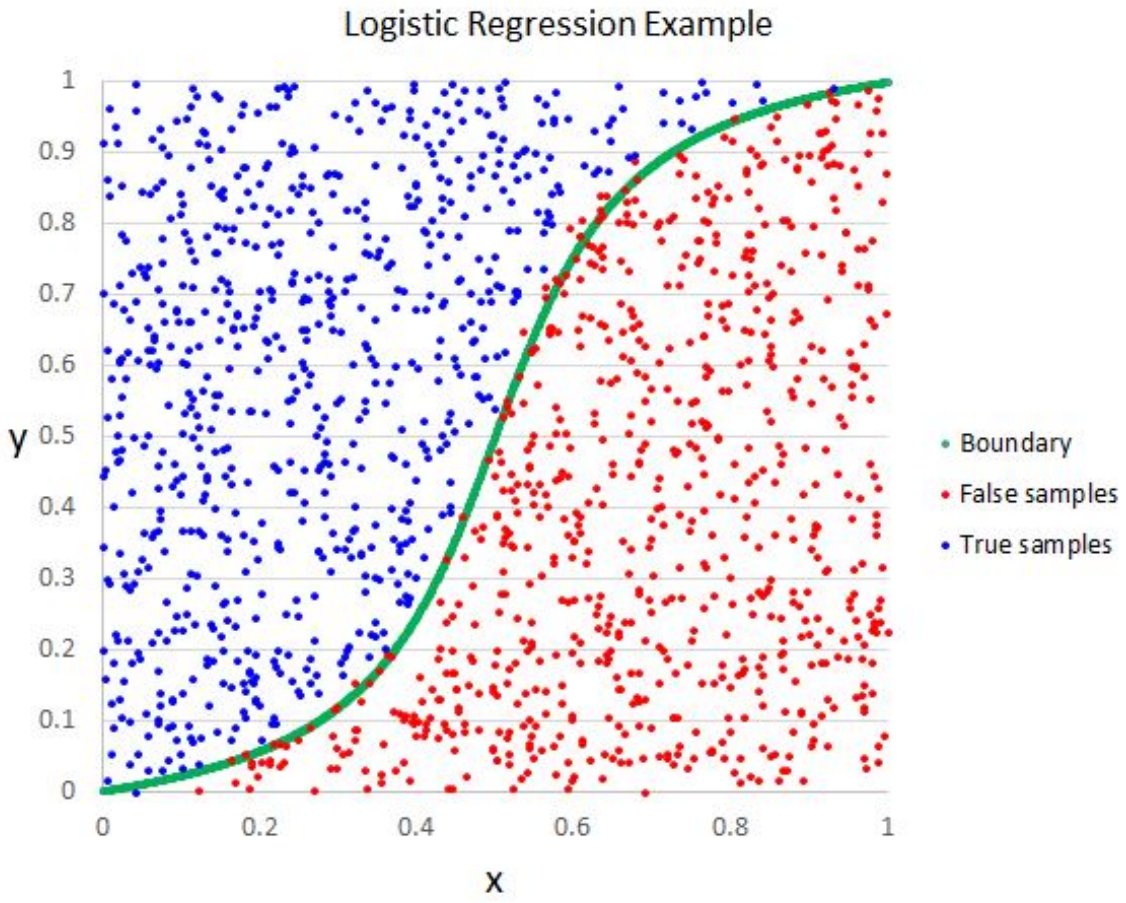


Figure 2.2: Result of Logistic regression
Source: https://www.mssqltips.com/sqlservertip/3471/introduction-to-the-sql-server-analysis-services-logistic-regression-data-mining-algorithm/

2.2.4 Training the Logistic regression

We need to train *Logistic regression* in order to be able to classify the data. We need a labeled dataset, defined as (x, t), where x is the vector of features and t is a vector of labels. A label is an information which assigns a data piece to a class. For previously mentioned 2 class problem with N data pieces, the label vector can be defined as $t \in 0, 1$ for $n = 1, ..., N$

LR has two kinds of parameters - weights and biases. These parameters are described in the Section 2.2.2.

We are searching for these parameters by maximizing some error function in the maximum likelihood manner. For the provided dataset a likelihood function can be written like this

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n} \{1 - y_n\}^{1-t_n} \tag{2.21}$$

Often the log of $p(t|w)$ is used and it is called *log likelihood*. This function is used because it is easier to optimize a sum than a product. Taking the negative of the *log likelihood* gives

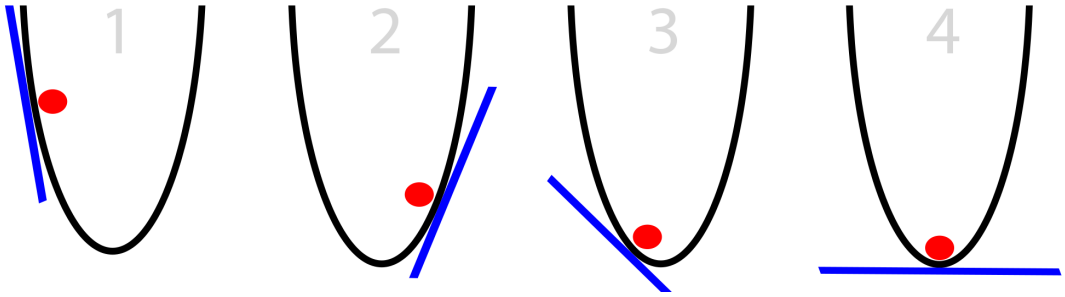


Figure 2.3: SGD steps.
Source: http://iamtrask.github.io/2015/07/27/python-network-part2/

us the error function called *cross entropy error function*

$$E(\mathbf{w}) = -\ln p(t|\mathbf{w}) = -\sum_{n=1}^{N} (y_n \ln y_n + (1 - t_n) \ln(1 - y_n)) \tag{2.22}$$

where $y_n = \sigma(a_n)$ and $a_n = w^T x_n$.

Our goal is to find the minimum of the error function. The minimal value of the error function is the maximum of the log-likelihood, to get the best parameters we use the *stochastic gradient descent* algorithm. Stochastic gradient descent (SGD) is a numerical method for finding a local minimum. Imagine 1000 values that are plotted by a function *f*. If we want to find a minimum from these values, there is no problem to check all of the values iteratively and select the smallest one. The difficulty arises when we add additional dimensions. If the input function is two dimensional than we need to go through $1000 * 1000$ values already. The complexity grows exponentially with bigger magnitude. More dimensions equal more values to check for possible minimum.

SGD solves this problem in an elegant way. When we derive a function in a certain point we also get a gradient value. This value tells us whether the function is increasing (the value is positive) or decreasing (the value is negative). Zero means that we found the minimum. SGD is trying to find the minimum by „moving“ through the function by some steps. The movement direction is chosen by derivative value and converges towards the minimum. This can be seen in Figure 2.3.

However, SGD is not perfect. It always finds a local minimum, not a global one. On the other hand, the functions are often very complex, that means that finding a global minimum is almost impossible. Other difficulty comes in hand with the step size, which might be too big and SGD might skip the desired minimum. This problem can be solved by dynamically changing the step size. The closer we get to the minimum, the smaller step size we use.

It is good to know that maximum likelihood training procedure can also suffer from over-fitting. This problem is visually explained in Figure 2.4. Instead of fitting a line between the points, it goes directly through all of them.

Multiclass LR is similar to 2 class problem, with the nonlinear function like *Softmax* 2.18, which was explained earlier.

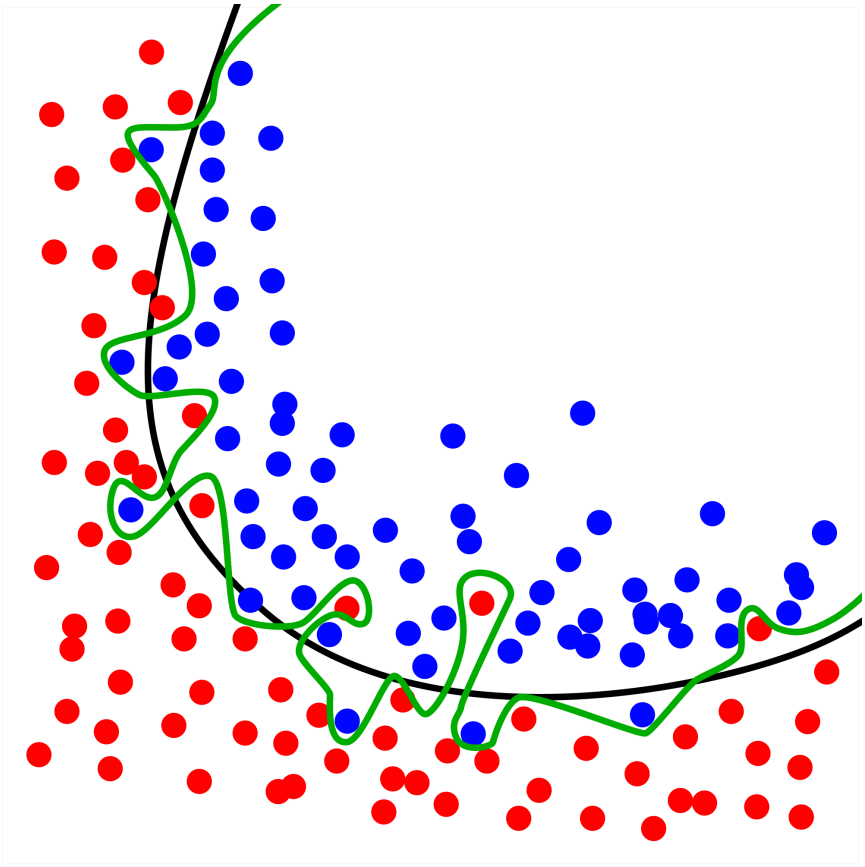


Figure 2.4: The green line shows how over-fitting (over-learning) looks like.
Source: https://commons.wikimedia.org/wiki/File:Overfitting.svg

2.3 Neural Networks

Neural Networks (NNs) are mathematical models that are inspired by the human brain. They are composed of many small computational units called *neurons*. These units are connected. According to the type of the connections we differentiate between various types of NNs. Some NN types are described in the subsequent parts of this thesis.

2.3.1 Artificial Neuron

Artificial neurons are the building blocks of neural networks. Each neuron is a single computational unit with n *inputs*. The neuron is capable to represent a *state*. Each state is described by a vector $w = (w_0, w_1, ..., w_p) \in \mathbb{R}^p$ of p real numbers, known as *weights* or *parameters*. The number p depends on the number of *inputs* n. Based on the output function, the neuron can be a *linear threshold* unit or *sigmoid unit*.[2] Visual representation of this neuron is shown in Figure 2.5.

2.3.2 Multilayer Perceptron

Multilayer perceptrons (MLPs) are the most popular type of neural networks in use today. They belong to a general class of structures called *feed-forward neural networks*. This type of neural network is capable of approximating functions. [15].

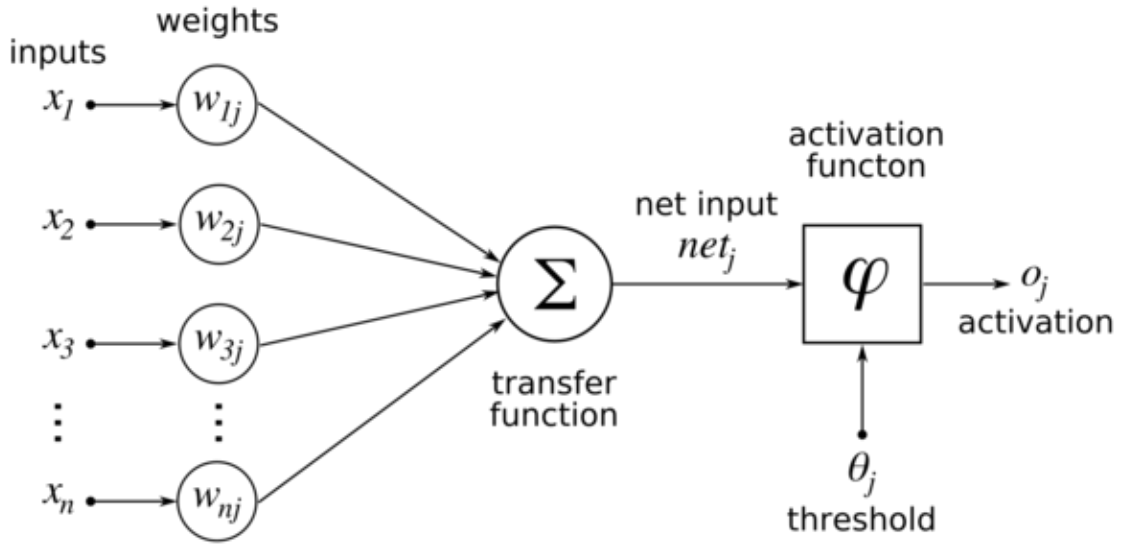


Figure 2.5: Image of an artificial neuron.
Source: https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Print_Version

MLP consists of three layers: the input layer, the hidden layers and the output layer (Figure 2.6). The input layer transforms the input data. The hidden layer does some linear transformations of the inputs. The output layer transforms the data to some kind of a scale. We can use the *Logistic regression* model as an output layer. In that case, we can view the MLP as an extended LR.

MLP can be viewed as a Logistic regression classifier, which projects the input data to a linearly separable space using learnt non-linear transformation. For this operation, a single hidden layer is sufficient.[8]

MLP is trained using *Back-Propagation* (BP) algorithm. This algorithm belongs to a category of supervised learning methods. *BP is used to train the weights of networks by gradient descent in an objective function, such as the total classification error evaluated on a given training set of input patterns and corresponding labels.*[16]

By providing a probabilistic interpretation of the network outputs, we can get a general view on the network training.[3] That means that the network outputs a vector of probabilities for each of the classes. This approach gives us an opportunity to check other possible results. For example our goal is to classify 3 classes (A, B, C). Instead of saying that the result is class A, the MLP output looks like $(0.65, 0.25, 0.10)$. This means, that the result is for 65 % A, for 25 % B and for 10 % C. With this kind of result, we are able work further. We can pass the output of MLP to input of another network.

2.3.3 Other architectural types of the networks

Neural Networks may differ in the type of the connections, their numbers or the numbers of units in the layers.

Apart from *feed-forward* Networks like MLP, different architectures exist. An introduction of two other architecture types follows.

Recurrent neural network can have many different forms. They are often based on a MLP model with, at least one, added feed-back connection. These connections create a loop. The activations can round in these loops. This creates a sort of memory inside the

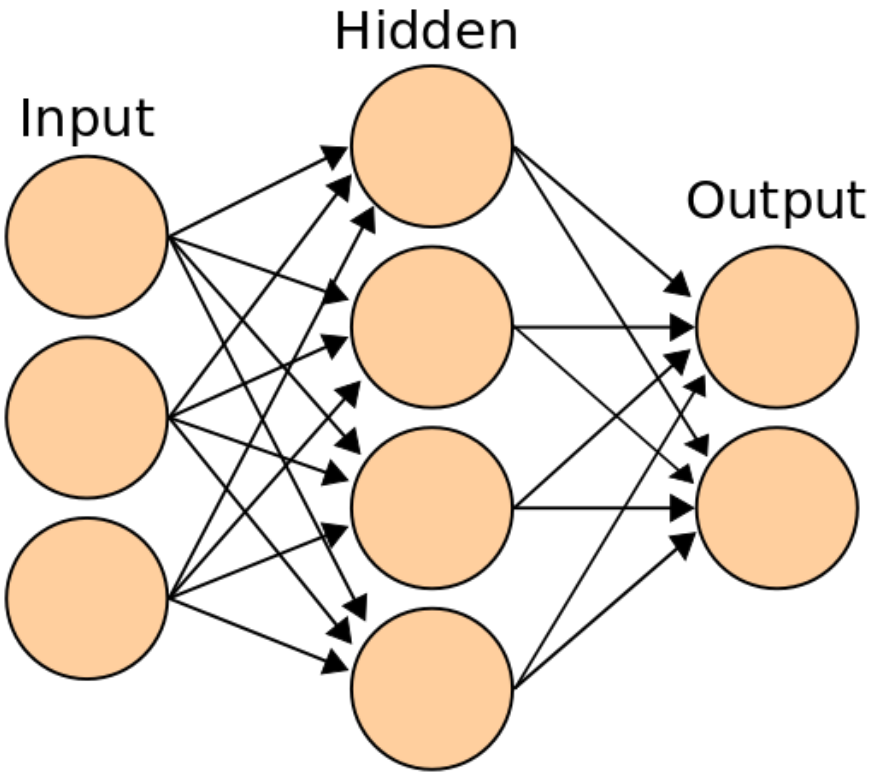


Figure 2.6: Visualization of Multilayer Perceptron model
Source: https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Print_Version

RNN. This memory enables the network to do temporal processing and to learn sequences. [4]

Modular feed-forward networks (MFNs) are systems with multiple independent neural networks. This architecture is inspired by the human brain. It is more efficient to split a problem into smaller ones. Let's suppose we want to classify waste. One part of the system can classify shapes independently on the color classifier. MFNs help to reduce complexity of problems and improve the network performance. [17]

Chapter 3

Method

In this chapter, we describe how the provided theory is used to design a classifier that is able to describe sound. Our goal is to build a system that can classify phonemes or phoneme categories.

3.1 Objectives

We would like to design a classifier using an existing deep neural network learning framework. We want to be able to tune the properties of the classifier using command line tools. We also want to run a number of experiments on a data set to determine the best configuration of the phoneme classification.

3.2 Construction of the classifier

We need to select the best mathematical model on which our classifier will be based. We decided to use MLP, described in the section 2.3.2. The model has three hidden layers, because three hidden layers should be sufficient for this task. However, we would like to select the best number of units within these layers based on the results of experiments. We would also like to test whether the classifier can be more efficient using some larger audio context. The context is given by multiple consecutive feature vectors representing about 10 ms of audio each.

The weights in the classifier are initialized randomly. During the training we also check the stability of the model, by looking at the variability of results.

3.3 Setup

At first we need to prepare the inputs for the network. That means that we need to split the data into three data sets, one for training, one for validation and one for testing. The training set is used to build the model, the validation set checks how well did we train the model, and the testing sets is used to test the model on real data. We also have to create a computer program that demonstrates the selected model. For this part, there are actually two options.

1. Write everything from scratch.
2. Use an existing deep learning framework.

The second option was selected, as we know that instead of a lot of hours spent debugging, we are able to use a reliable software that has already been tested and run more experiments.

Chapter 4

Analysis

In this chapter we present our findings. We went through some available tools for machine learning to select the best one which was used for implementation of the audio event classifier. The audio database was analyzed and prepared for experiments.

4.1 Tools

An important part of problem solving is to choose the right tools for the job. In our case, we were supposed to classify phonemes and phoneme groups using Neural Networks. One possibility was to implement it ourselves. However, this way takes a lot of time and may not compete with existing libraries. That is why we decided to go with a machine learning framework. Using a framework, the work can be simplified a lot, because a lot of functionality is already inside the framework. Currently there are multiple popular frameworks. We needed to compare their positives and negatives in order to chose one which would suit our needs in a best way. We present the considered frameworks in upcoming subsections.

4.1.1 Theano Framework

Theano[18] is a Python library that allows to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. It is used for compilation of the mathematical expressions. Theano itself is not written in Python but uses Python bindings. This approach gives Theano a huge performance improvement. The models are compiled to machine code instead of being interpreted. Theano compiles the code to run on CPUs as well as GPUs. The framework stands on top of the *SciPy* stack.

Even though it is not a deep learning framework, it is widely used in the machine learning community. A lot of tools have been built on top of Theano. The official website of the project is `http://deeplearning.net/software/theano/`.

4.1.2 TensorFlow

TensorFlow[1] is one of the newest computational libraries. It is an open source software library for numerical computation using data flow graphs. It was released by Google. This framework is not complete and is relying on community support. One of the main positives is that it contains pre-trained models. The models can be trained using both CPUs and GPUs. TensorFlow Python API is interpreted in comparison to the Theano Framework

which is compiled into native code. On the other hand, TensorFlow has also C++ API which profides significant performance improvements. The official website of the project is https://www.tensorflow.org/.

4.1.3 Keras

Keras [6] is a minimalistic framework which is built upon different frameworks. By default, it runs upon Theano Framework 4.1.1. However, there is a possibility to change the backend framework to TensorFlow 4.1.2. Its simplicity is similar to Torch 4.1.4. Keras stands on four principles:

1. modularity
2. minimalism
3. easy extensibility
4. work with python

Each model is made of modules that can be easily plugged to each other. Each module should be kept short and easily understandable. It is easy to write new modules in Keras. All of these properties, combined with the simplicity of Python, create an easy-to-use framework for experimenting with neural networks.

However, it provides a certain level of abstraction, which means that we can't „get our hands dirty“ directly with the models and directly experiment with them. The official website of the project is http://keras.io/.

4.1.4 Torch

Torch [7] is not written in Python, like the other frameworks, but in Lua. This framework is mainly focused on training the models using GPUs. Torch has a large ecosystem of community-driven packages for machine learning. Torch is used in Facebook[5]. Goggle DeepMind has recently moved to TensorFlow [11]. The official website of the project is http://torch.ch/.

4.1.5 Framework selection

We have decided to go with Theano Framework as it is highly supported in the academic environment. It is the oldest one mentioned, meaning that it has a lot of online resources available. A huge advantage is that it is a Python framework. Python is known for rapid development, it makes the preparation of the experiments faster, making it easier to make modifications of the code.

The problem with Torch was that it is written in Lua and we were not familiar with this programming language.

If we compare Theano to Keras, the benefit of Theano is that we are able to work with the models on a lower level. This means that we are able to do low level optimizations. Theano gives us freedom to experiment with the neural networks.

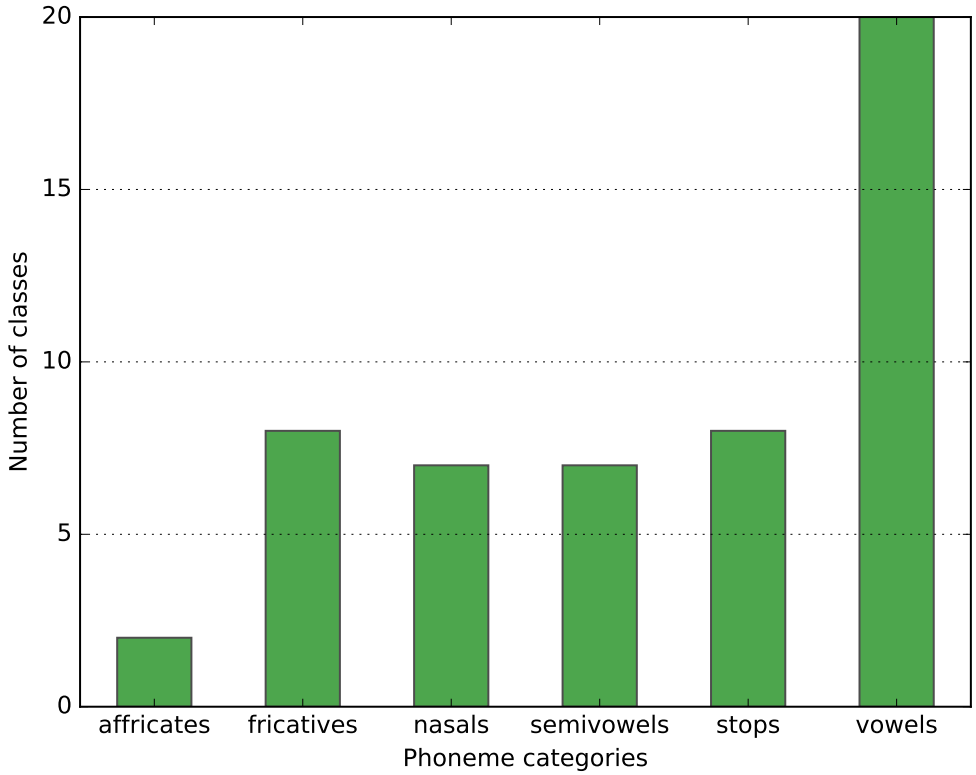


Figure 4.1: Class distribution within the phoneme categories.

4.2 TIMIT Database

We have been provided with a database called *The DARPA TIMIT Acoustic-Phonetic Continuous Speech Corpus (TIMIT)* [9]. TIMIT was designed to provide speech data for the acquisition of acoustic-phonetic knowledge and for the development and evaluation of automatic speech recognition systems. This database was published in October 1990. It contains a total number of 6300 sentences, 10 sentences spoken by each of 630 speakers.

The dataset is split to two section, the training data and the testing data. However for our needs we need to split these two lists into three. We decided to have a training set of size 4000, a validation set with 1300 sentences and a testing set with 1000 sentences.

Each entry in the TIMIT datasets contains an audio file and three label files. These files contain a table with three columns. The last column specifies a class label and the two first columns show when the event occurs. The first one stands for the event start and the second one is the end of the event. Their units are in *samples*. The audio is sampled at a frequency of 16kHz. This means that for one second of audio there are 16 000 samples.

There are 57 classes in total. According to the recommendation from my supervisor, we have decided not to include the silence and non-speech events. This step reduced the number of classes to 52. These classes are organized into 6 categories (Tables 4.1 and 4.2). The number of classes within the groups is not equally distributed (Figure 4.1). On the other hand, the distribution of classes in general is quite even (Figure 4.2).

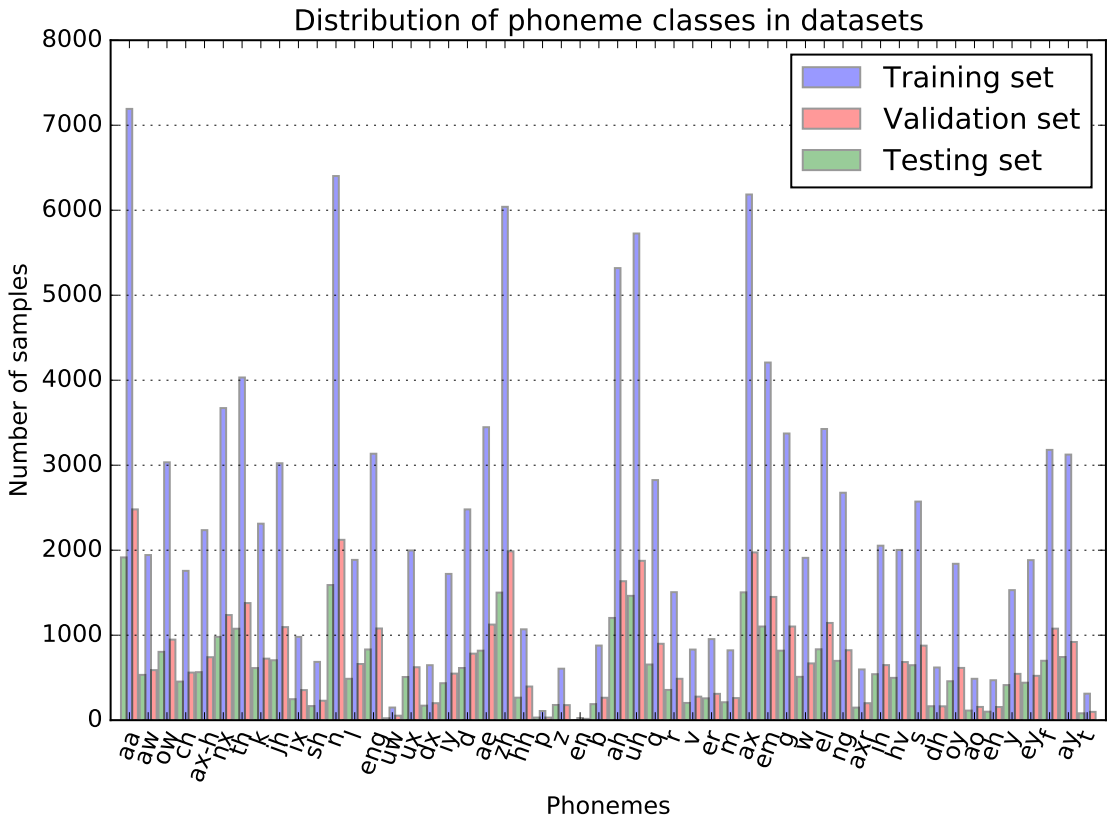


Figure 4.2: Class distribution within the data.

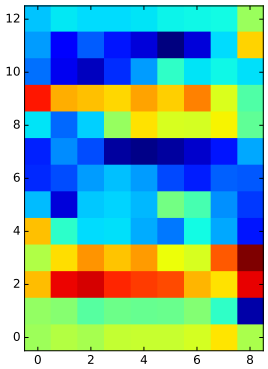
4.3 Features

To be able to classify data, we need to convert the data to a representation which is suitable for NNs. It is also necessary to filter out unwanted parts of the audio, like noise. The process of data conversion is called *feature extraction*. In our case, we are dealing with audio data. Sound is sampled at a particular rate. There is one feature vector per 10 ms, so the result of the extraction is an array of feature vectors.

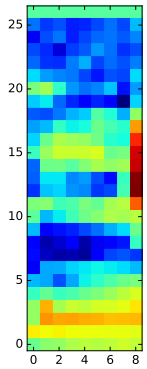
We have been provided with four different types of extracted features from TIMIT database. The types we got were *Mel Frequency Cepstral Coefficient* (MFCC), *Filter bank* (FBANK) features and both of them having the double delta modification. All of them we extracted using *Hidden Markov Model Toolkit* (HTK). These features have different properties, their difference can be seen in a Figure 4.3. In the upcoming text, we would like to compare them.

4.3.1 Mel Frequency Cepstral Coefficient (MFCC)

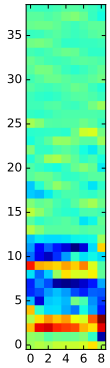
The main point to understand about speech is that the sounds generated by a human are filtered by the shape of the vocal tract including tongue, teeth etc. This shape determines what sound comes out. Determining the shape accurately should give us an accurate representation of the phoneme. The job of MFCC is to accurately represent the power spectrum which was produced by the vocal tract.[13]



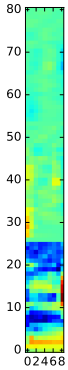
(a) MFCC



(b) FBANK



(c) MFCC with double delta



(d) FBANK with double delta

Figure 4.3: Several types of features for phoneme 'en'

MFCCs are often used for speech recognition. We extract the features using these steps:

1. We frame the signal into short frames.
2. For each frame calculate the periodogram estimate of the power spectrum.
3. Apply the mel filterbank to the power spectra, sum the energy in each filter.
4. Take the logarithm of all filterbank energies.
5. Take the discrete cosine transform (DCT) of the log filterbank energies.
6. Keep DCT coefficients 2-13, discard the rest.

4.3.2 Filter Bank (FBANK)

FBANK features are extracted from a signal. The difference from MFCC 4.3.1 is that we don't take the logarithm of the filter bank energies, instead we take these energies directly.

4.3.3 Double delta features

Also known as differential and acceleration coefficients. The MFCC feature vector describes only the power spectral envelope of a single frame. However, the speech also has some information carried in the dynamics. Over time, they are in the trajectories of the MFCC coefficients. If we calculate the MFCC trajectories and append them to the original feature vector, we get better ASR performance. [13] Better performance was verified by our experiments.

Table 4.1: Table of Phonemes part 1

Group	Phoneme class	Example word	Phonetic transcription
Stops	b	bee	BCL B iy
	d	day	DCL D ey
	g	gay	GCL G ey
	p	pea	PCL P iy
	t	tea	TCL T iy
	k	key	KCL K iy
	dx	muddy	m ah DX iy
	q	bat	bcl b ae Q
Affricates	jh	joke	DCL JH ow kcl k
	ch	choke	TCL CH ow kcl k
Fricatives	s	sea	S iy
	sh	she	SH iy
	z	zone	Z ow n
	zh	azure	ae ZH er
	f	fin	F ih n
	th	thin	TH ih n
	v	van	V ae n
	dh	then	DH e n
Nasals	m	mom	M aa M
	n	noon	N uw N
	ng	sing	s ih NG
	em	bottom	b aa tcl t EM
	en	button	b ah q EN
	eng	washington	w aa sh ENG tcl t ax n
	nx	winner	w ih NX axr
Semivowels and Glides	l	lay	L ey
	r	ray	R ey
	w	way	W ey
	y	yacht	Y aa tcl t
	hh	hay	HH ey
	hv	ahead	ax HV eh dcl d
	el	bottle	bcl b aa tcl t EL

Table 4.2: Table of Phonemes part 2

Group	Phoneme class	Example word	Phonetic transcription
Vowels	iy	beet	bcl b IY tcl t
	ih	bit	bcl b IH tcl t
	eh	bet	bcl b EH tcl t
	ey	bait	bcl b EY tcl t
	ae	bat	bcl b AE tcl t
	aa	bott	bcl b AA tcl t
	aw	bout	bcl b AW tcl t
	ay	bite	bcl b AY tcl t
	ah	but	bcl b AH tcl t
	ao	bought	bcl b AO tcl t
	oy	boy	bcl b OY
	ow	boat	bcl b OW tcl t
	uh	book	bcl b UH kcl t
	uw	boot	bcl b UW tcl t
	ux	toot	bcl b UX tcl t
	er	bird	bcl b ER dcl t
	ax	about	AX bcl b aw tcl t
	ix	debit	dcl d eh bcl b IX tcl t
	axr	butter	bcl b ah dx AXR
	ax-h	suspect	s AX-H s pcl p eh kcl k tcl t

Chapter 5

Discussion

In this chapter, we present the results that were obtained during the experimentation with the neural networks. We experimented with three variables, the number of hidden units, the context size and the learning rate. We ran the experiments multiple times to check how stable the models are.

5.1 Preparing the experiments

To be able to run the experiments, we need to prepare the data and design the classifier.

5.1.1 Architecture of the classifier

The classifier is built in a modular way using Python3 with the Theano Framework. At first, the system needs to have the training data prepared. The system accepts three text files which contain a list of pairs. These pairs are made of two values. The first value represents a path to phoneme transcription and the second one is a path to a feature file. After the lists are parsed, the labels and the feature files are loaded into memory. Then we need to filter the features and match them with labels. The timing of label units is in samples 4.2, that is why we need to convert it into milliseconds.

$$FeatureIndex = \frac{1000 \cdot Sample}{SamplingFrequency} \tag{5.1}$$

Each of the phonemes has its own length. We should select the one in the middle to get the most representative feature. The central feature vector can be selected using this equation:

$$FeatureIndex = \frac{EndingIndex - StartingIndex}{2} \tag{5.2}$$

where *StartingIndex* is the beginning of the phoneme and the *EndingIndex* is the end. We can also include the context of the feature sample. This may improve the accuracy of the model. The size of the context can be defined using a command line parameter. Some improvements from the context are discussed in the next section.

The prepared data is passed into function that builds the MLP model. The MLP model has a variable amount of units within the hidden layer and the learning rate. According to the settings, the program trains the model.

The training dataset is used to train the network, the validation set is used to prevent the model from over-fitting to the dataset and the test set is there to evaluate the accuracy of our classifier.

The program outputs some info about the setup and the error rate for the experiment. The format of the output is '<property>:<tab><value>'. An example of the program output looks like:

```
$ python main.py --hidden 2048 --rate 0.1 train.txt valid.txt test.txt

learning rate:          0.100000
batch size:             200
hidden units:           2048
context size:           10
classifying:            phonemes
==================================
before training:        98.276316 %
----------------------------------
normalized mutual information: 0.425323549414
normalized mutual information: 0.45555220168
normalized mutual information: 0.46656820087
normalized mutual information: 0.491831599369
iteration number:       2000
best validation:        30.864322 %
test performance:       28.601974 %
**********************************
```

5.2 Experimenting with the parameters

The goal was to compare performance of the model with varying input features and various argument combinations. At first, we have experimented with different number of units within the hidden layers. The results can be seen in Figures 5.1 and 5.2. For basic feature categories, the more units we add the better result we get. However, after 512 units per hidden layer the improvement is not that significant. On the other hand, the double delta features had an error rate drop at 256, then it slightly increased, and then it started decreasing again. The results indicate that we are able to build a sufficient classifier using less units per layer.

We have decided to experiment with the context. The context size defines the number of feature frames that are taken from each side starting from the middle one. All of these samples are taken and merged into a single vector which is fed to the model. In our case the difference between the MFCC and FBANK was not that big, MFCCs performed a bit better. We can see that in all of the cases (Figures 5.3 and 5.4), the bigger context the better. At a certain size, the context starts to worsen the results. This started happening because the context overlaps the surrounding phonemes, and we no longer feed the neural network with features that are strictly separated.

There is a significant difference between the normal features and the double delta features. We can see that feeding the double delta features to the neural network without context gives us better results. However, this advantage can be simply overtaken by adding more context.

The last part of these experiments was to evaluate the effect of learning rate (Figures 5.5 and 5.6). The best learning rate for our data set is 0.1 according to our results. Bigger learning rate means smaller precision in gradient descent. With smaller precision, it is

harder to find the right minimum. However, a learning rate that is too small causes the model training to take a lot of time, and stuck in local minimum with higher error rate in the end.

5.3 Testing the variance of the model

Our model is initialized using random values. We need to check whether the setup of the system influences the results. If yes, than we need to check how much. This information gives us a deeper insight into our model and to its trainings. To check the stability, we ran the same experiment 100 times with the same arguments. The results can be seen in the Figure 5.7. We can see that our results are pretty stable. Even when we initialize the model weights at random, we see that the results differ by less than 0.5%. This means that initialization does not have a significant impact on the accuracy of the classifier.

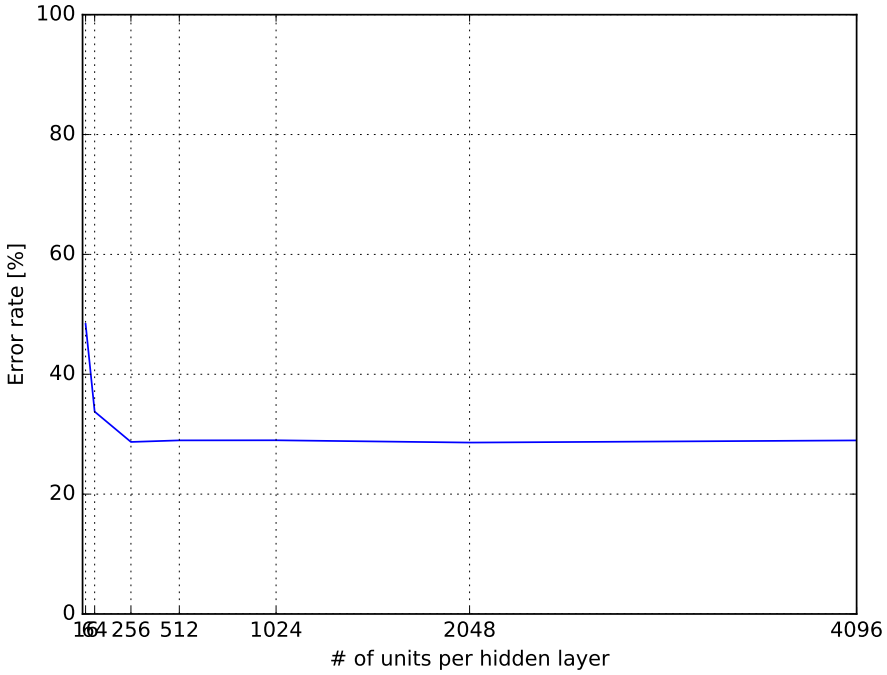
5.4 Classifying phoneme categories

We ran experiments to classify the phoneme categories. In this case, the classifier had significantly better accuracy. A lot of phonemes within a category are similar. Because of that the classifier in the previous had a difficulty separating these phonemes. However, in this experiment the number of classes decreased to 6 and the similar phonemes were merged into categories. We ran the experiment on four provided feature sets. The *double delta* feature results had the context-less boost we know from the previuos experiment. FBANK features had a better performance than MFCC. The results are shown in Figure 5.8. We can see the same difference between *double delta* and normal features. With decent context, the difference between the feature types disappeared.

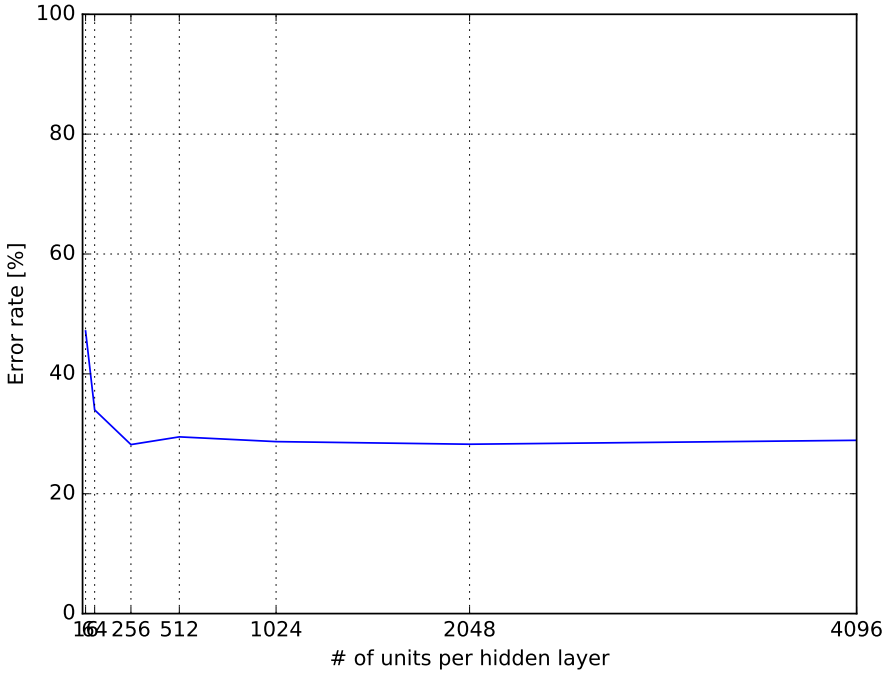
5.5 Utilities

During the development and testing of our classifier a lot of tasks were quite repetitive. We have run the experiments on multiple separated machines, so we had to deal with moving the dataset around.

However, the most repetitive task was plotting of the results. We did a lot of calculations and plotting the results was a tedious task. That is why we have decided to create a utility for automatizing of output files parsing. It plots the specified properties and is really handy for examination of the results. This utility allowed us to generate a lot of different plots in a short period of time.

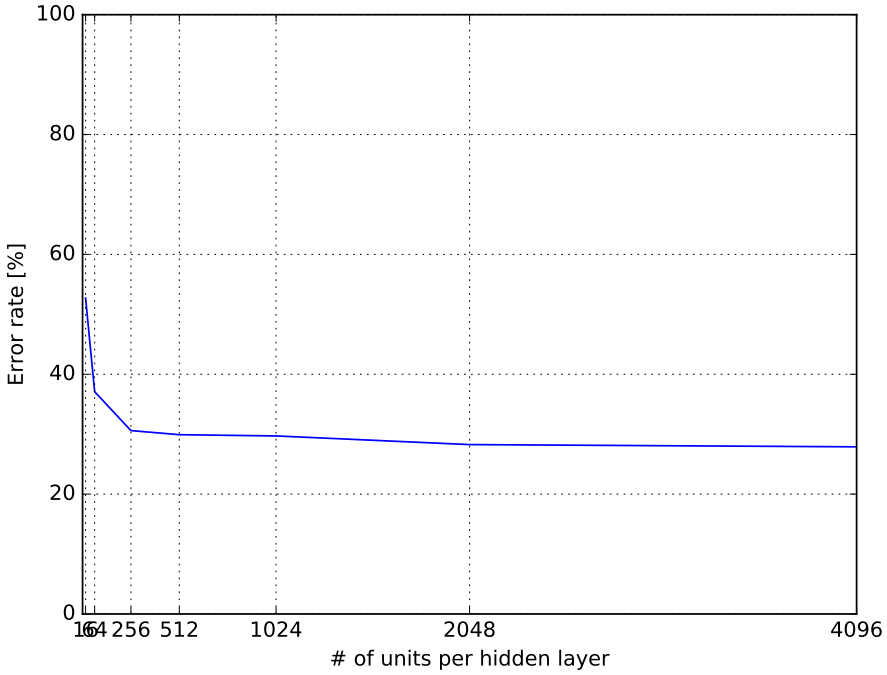


(a) FBANK

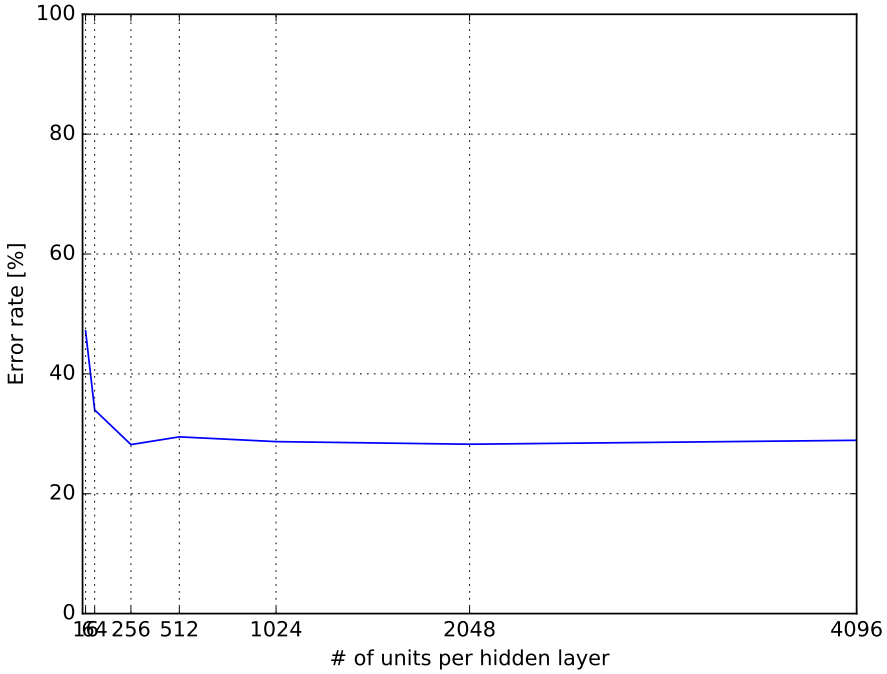


(b) FBANK double delta

Figure 5.1: Relation between error rate and the number of units per hidden layer with FBANK features, context size of 10 and the learning rate 0.1.

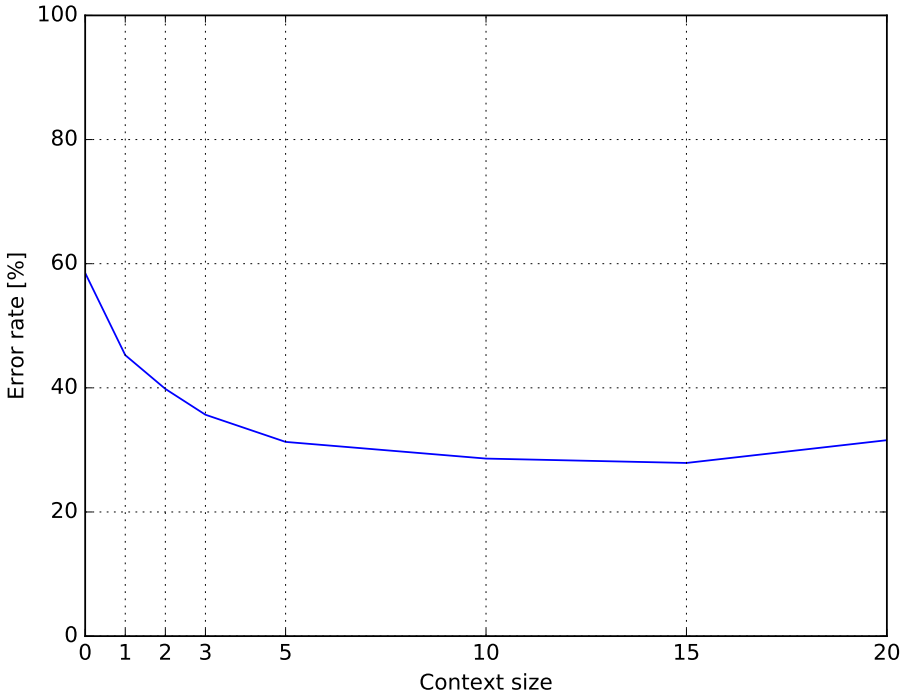


(a) MFCC

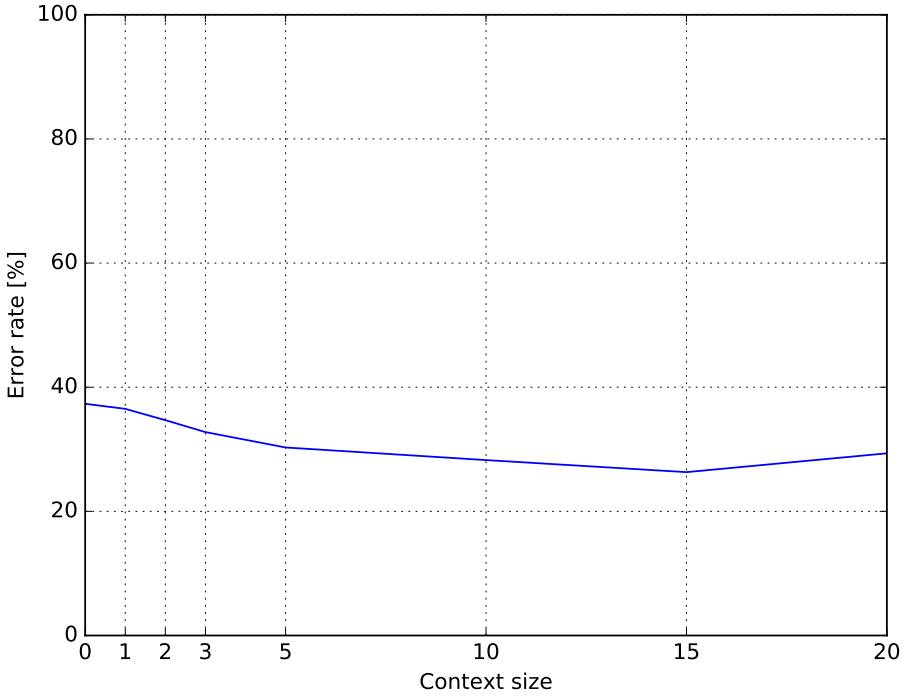


(b) MFCC double delta

Figure 5.2: Relation between error rate and the number of units per hidden layer with MFCC features, context size of 10 and the learning rate 0.1.

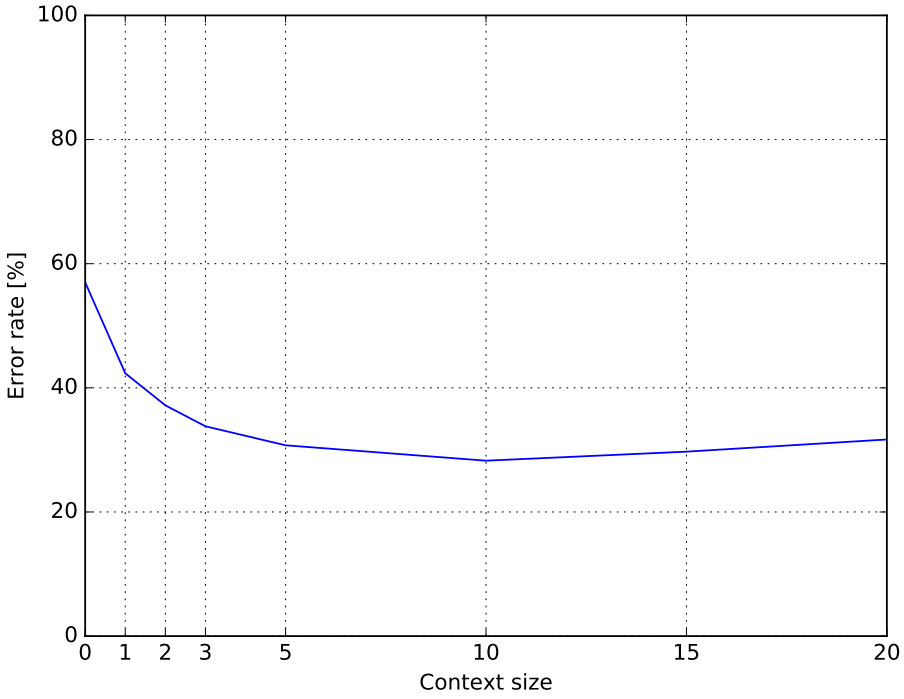


(a) FBANK

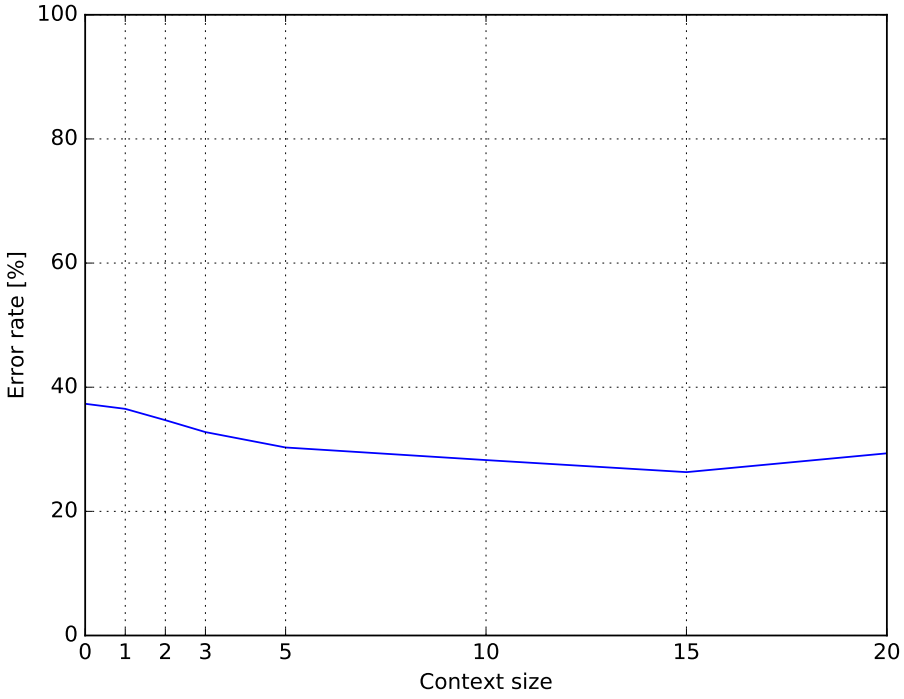


(b) FBANK double delta

Figure 5.3: Relation between error rate and the context size of the NN with FBANK features, with 2048 units per hidden layer with learning rate 0.1.

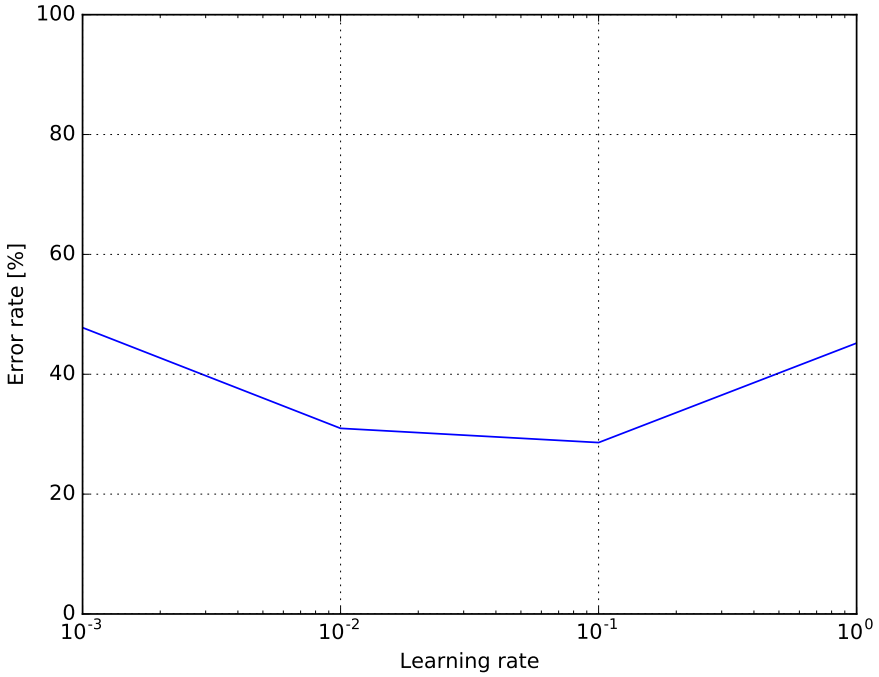


(a) MFCC

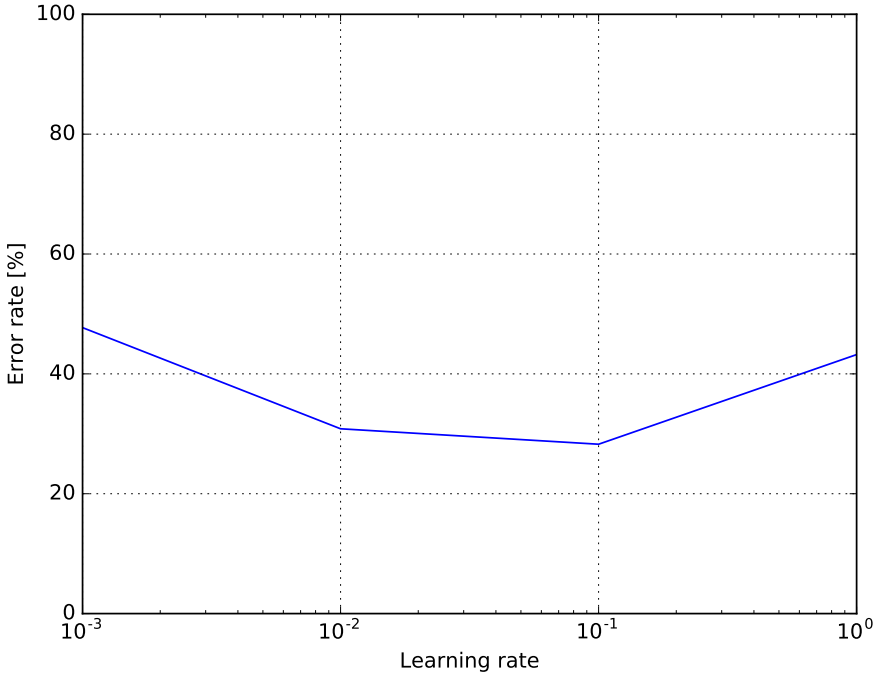


(b) MFCC double delta

Figure 5.4: Relation between error rate and the context size of the NN with MFCC features, with 2048 units per hidden layer with learning rate 0.1.

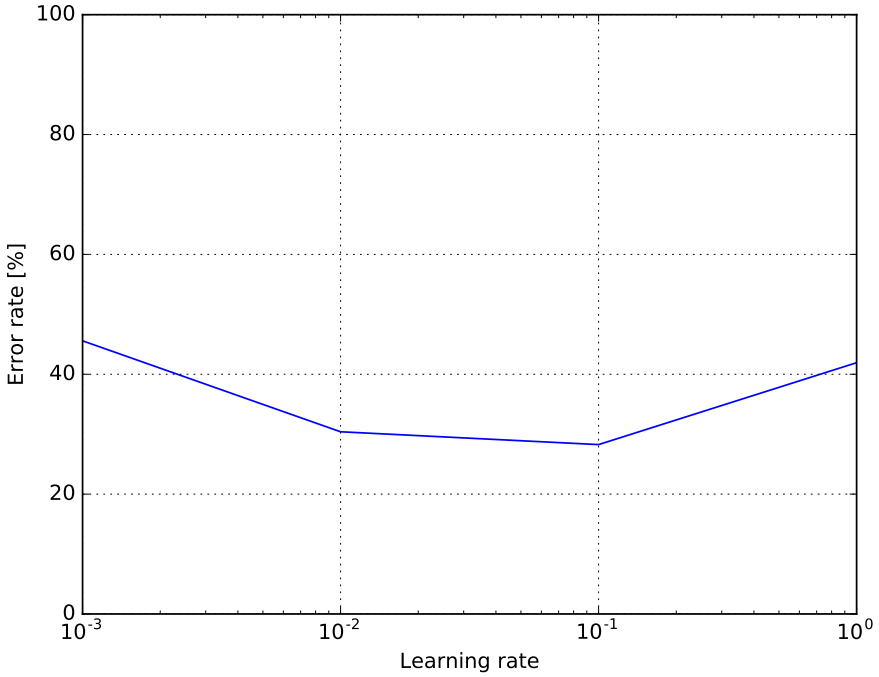


(a) FBANK

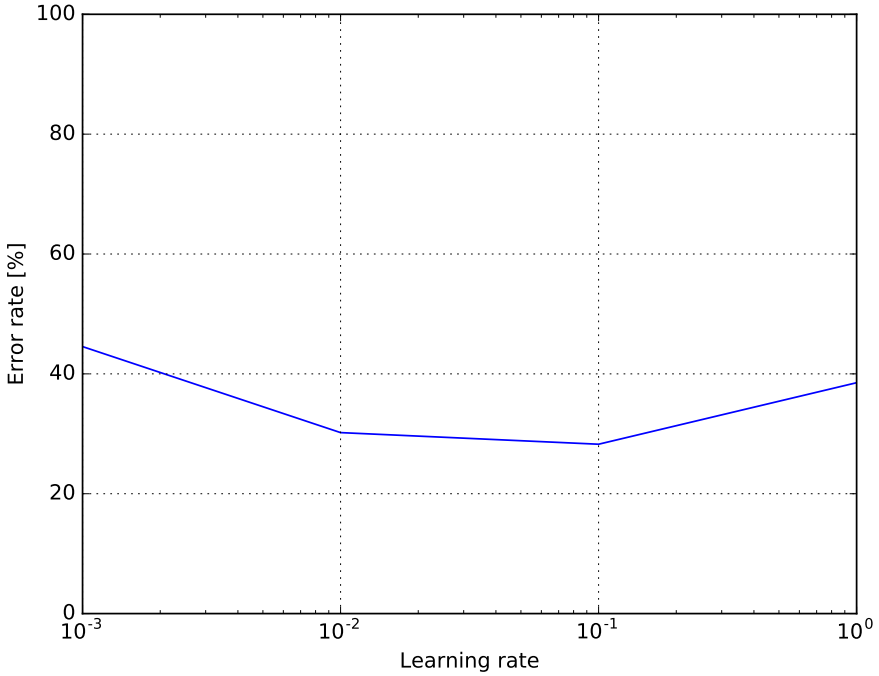


(b) FBANK double delta

Figure 5.5: Relation between error rate and the learning rate with FBANK features, context size of 10 and the number of units within the hidden layer 2048.



(a) MFCC



(b) MFCC double delta

Figure 5.6: Relation between error rate and the learning rate with MFCC features, context size of 10 and the number of units within the hidden layer 2048.

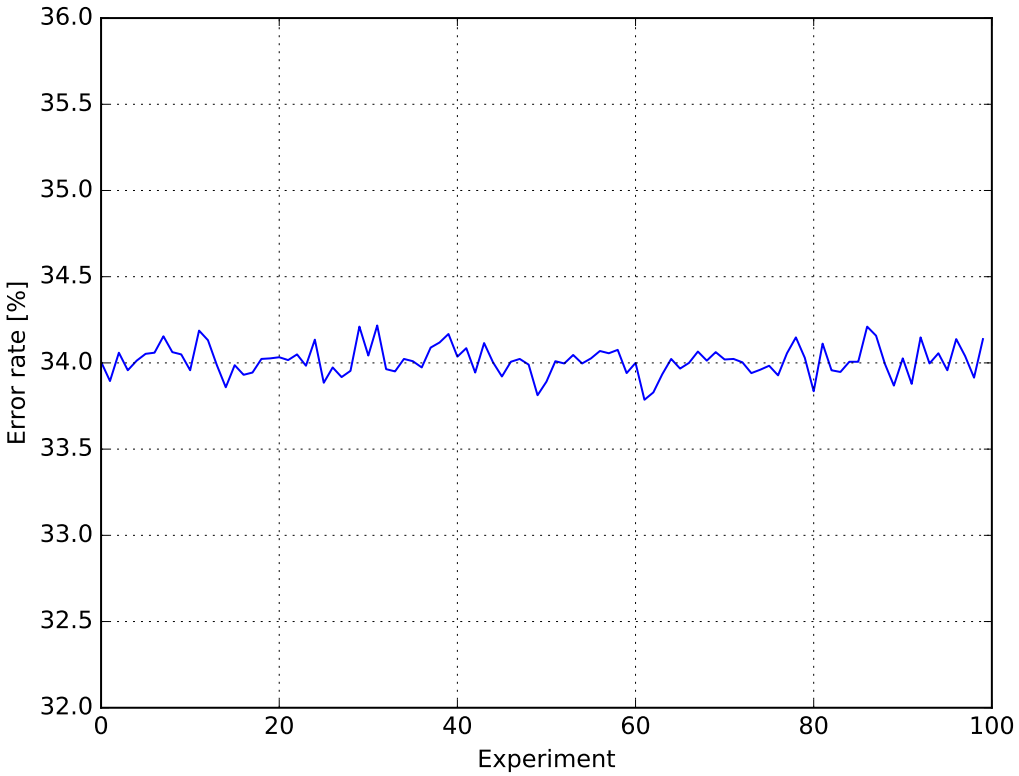
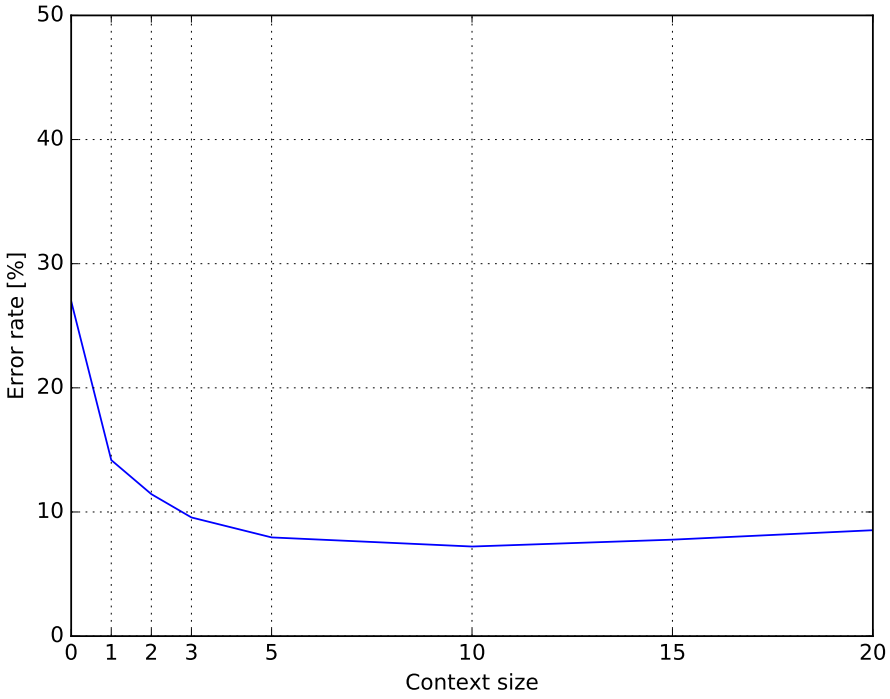
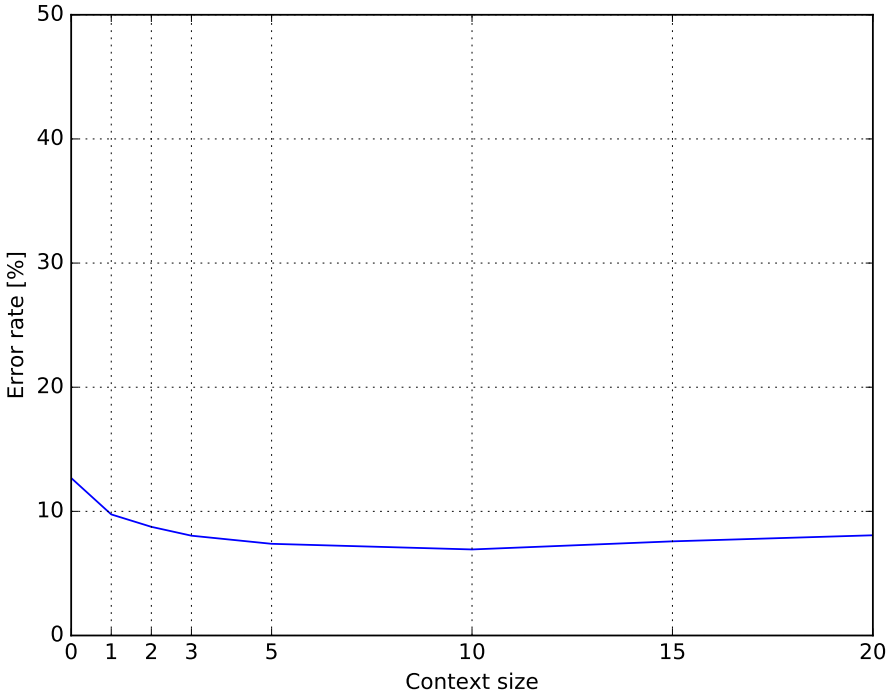


Figure 5.7: Plot showing the stability of the results when running the experiment 100 times with the same network configuration. The configuration in this case was FBANK features, context size of 10, learning rate 0.1, and the number of hidden units within the layer was 2048.



(a) FBANK



(b) FBANK double delta

Figure 5.8: Relation between error rate and the context size of the NN with FBANK features, with 1024 units per hidden layer with learning rate 0.1 for phoneme category classification.

Chapter 6

Conclusion

We designed an audio event classifier, which was based on the MLP model and had 3 hidden layers. Our implementation of this model has a possibility to tune the properties of the neural network using command line tools. We experimented with variables that influence the performance of our classifier. This process included changing the number of units within the hidden layers, different context sizes and a different learning rates. All of these experiments were run for different feature types. We used MFCC and FBANK features, also with their double delta variants.

We compared the results we got and found out that context size has a significant role in the audio recognition. In general, the bigger the context, the better the accuracy. However, when the context was too big, it overlapped to different classes and the accuracy worsened.

Double delta features had a better performance with smaller context, compared to the regular features. The experiments show that if we increase the audio context size, the accuracy of all classifiers across the examined feature types improve. For our database with context size of 10, there is no difference in error rate between the feature types. This means that the initial advantage of double delta features can be lowered by increasing the size of the context.

Neural networks with 512 units per hidden layer did not differ significantly in their accuracy. This means that we can build a classifier which is trained faster and its performance is similar to one with more units in the hidden layers.

We also classified phoneme categories, and have seen a significant drop in error rate. The reason is that there are only 6 phoneme categories compared to 52 phonemes.

In our application, audio annotation is done by feeding the test data set to MLP. The accuracy of the annotation is evaluated using an error rate.

In our future work, we suggest combining the output of the phoneme category classifier with the phoneme classifier, and examining whether this improves the results.

Some of the phoneme classes share a similar power spectrum, which presents difficulties for the classifier when separating them. This suggests an opportunity to examine the impact that merging the similar phoneme classes into one could have on the performance.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Martin Anthony. *Discrete Mathematics of Neural Networks*, chapter 1. Artificial Neural Networks, pages 1–8. SIAM, 2001.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[4] John A. Bullinaria. Recurrent neural networks. http://www.cs.bham.ac.uk/~jxb/INC/l12.pdf, 2015. Accessed: 2016-05-14.

[5] Soumith Chintala. I am one of the maintainers. from information first-hand, torch is used by:. https://news.ycombinator.com/item?id=7929216, 2014. Accessed: 2016-05-08.

[6] François Chollet. Keras. https://github.com/fchollet/keras, 2015.

[7] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. https://github.com/torch/torch7.

[8] Theano community. Multilayer perceptron. http://deeplearning.net/tutorial/mlp.html, 2010. Accessed: 2016-05-03.

[9] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren. DARPA TIMIT acoustic phonetic continuous speech corpus CDROM, 1993.

[10] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition, 2009.

[11] Koray Kavukcuoglu. Deepmind moves to tensorflow. http://googleresearch.blogspot.cz/2016/04/deepmind-moves-to-tensorflow.html, apr 2016. Accessed: 2016-05-14.

[12] Geoff rey Hinton Li Deng and Brian Kingsbury. New types of deep neural network learning for speech recogniti on and related applicat ions: An overview. http://research.microsoft.com/pubs/189004/ICASSP-2013-DengHintonKingsbury-revised.pdf, 2013.

[13] James Lyons. Mel frequency cepstral coefficient (mfcc) tutorial. http://www.practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/, 2013. Accessed: 2016-05-13.

[14] Andrew McAfee and Erik Brynjolfsson. Big data: the management revolution. http://www.rosebt.com/uploads/8/1/8/1/8181762/big_data_the_management_revolution.pdf, oct 2012.

[15] Franco Scarselli and Ah Chung Tsoi. Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural Networks*, 11(1):15–37, 1998.

[16] J. Schmidhuber. Deep Learning. *Scholarpedia*, 10(11):32832, 2015. revision 152272.

[17] Pejman Tahmasebi and Ardeshir Hezarkhani. Application of a modular feedforward neural network for grade estimation. *Natural Resources Research*, 20(1), mar 2011.

[18] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

Appendices

List of Appendices

Appendix A

Content of the CD

- BP.pdf - Bachelor's thesis text
- Poster.pdf - Poster presenting our work
- /code/ - project directory of the practical part of the thesis
- /code/src/ - directory that contains the source files of bachelor's thesis
- /code/README.md - README of the project

Appendix B

Poster