ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

# DNN-based Phoneme Models for Speech Recognition

Diana Ponce-Morado

Master Thesis MA-2015-01

Computer Engineering and Networks Laboratory
Institute of Neuroinformatics

Supervisors:
Dr. Beat Pfister, TIK, ETH

Dr. Michael Pfeiffer, INI, ETH/UZH

Professor:
Prof. Dr. L. Thiele

November 16, 2015

# Abstract

State-of-the-art speech recognition systems include a hybrid architecture of hidden Markov models and deep neural networks (DNNs) for classifying phonemes. DNNs that are appropriately tuned can show significant improvement for phoneme classification. However, multi-layer perceptrons are complex models to tune, and now with even *deeper* neural networks, new methods are continually sprouting in the field to improve generalization of such models. We aim to disambiguate how new approaches better cope with the variability of phonemes between speakers and improve phoneme classification.

This task was achieved by conducting exploratory studies on feed-forward DNNs with artificial data to approximate optimal performance, i.e. Bayes classification rate. The report is divided in two parts that aim to answer: (1) how to build an appropriate DNN architecture given an infinite sample size, (2) at least how many training samples are required to build an optimal classifier for our data sets, (3) which features in processed TIMIT data contribute the most contextual information, and (4) does pre-training a DNN with artificial "speech-like" data configure weights of a DNN optimally for training TIMIT data.

Four sets of data were used in this study: three artificial sets that ranged from simple to complex and the TIMIT corpus. TIMIT is a widely used speech database for phoneme recognition. Part one of the report showed DNNs trained with artificial data achieved the Bayes error with dropout and rectified linear units. We observed a logarithmic relationship between the training sample size and the generalization error. In part two, delta features are found to improve classification performance of phonemes on the TIMIT test. Pre-training with artificial data and later fine-tuning DNNs with TIMIT data does improve generlization performance for networks of depth between two and six hidden layers. Deeper networks than six hidden layers are prone to shift weights away from an optimal configuration. Conclusively, we demonstrated DNNs are optimal discriminant classifiers when appropriately tuned and given sufficient samples. Future investigations would aim to apply alternative deep learning methods for feature extraction of speech data, e.g. convolutional neural networks, and investigating how recurrent neural networks can improve phoneme classification.

# Acknowledgements

I would like to thank Dr. Beat Pfister for the opportunity to explore deep neural networks in his lab. I would like to acknowledge Michael Pfeiffer, Giacomo Indiveri, and the Speech Processing group for their helpful suggestions.

# Contents

# Chapter 1

# Preliminaries

## 1.1 Introduction

Deep neural networks (DNNs) for phoneme classification have provided state-of-the-art results compared to the classical approach with hidden Markov models (HMMs) and Gaussian mixture models (GMMs). HMMs statistically describe temporal variability of phonemes and GMMs are used to approximate the statistical distributions of phonemes. One of the advantages of introducing a DNN to speech recognition systems is that a neural network can estimate the phoneme posteriors of feature vectors processed from speech fairly well which can be used further up the recognition pipeline. This approach replaces GMM for approximating likelihoods. Even though GMMs are suitable for modeling the probability distributions over feature vectors, their shortcoming is that GMMs require large covariance matrices and most often the underlying structure of speech is low-dimensional, not requiring a full-covariance matrix to capture parameters within a frame [HDY$^+$12].

Recent advancements in deep neural networks have focused to improve initialization techniques, regularization methods to prevent networks from overfitting on training data, activation functions to improve trainability of a network, and computational power harnessed in the application of Graphical Processing Units (GPUs) to build deeper networks. In this thesis, the objective was to apply popular methods to build a *good* DNN for the task of phoneme classification. We methodically approached this task by first working with artificial data and computed its Bayes error rates. For clarification, we consider a DNN *good* when its classification performance has reached the Bayes error for data that the underlying distribution is known. The Bayes error is the lowest error bound on any given classifier and can be thought of as the ground truth for what a classifier can potentially predict correctly.

In Chapter II, we introduce the background on DNNs and HMMs, and describe the methods used to conduct our experiments. Chapter III and part of Chapter IV present DNN experiments performed on artificial data with known Bayes error rates. These studies evaluate how well DNNs can approximate the Bayes error for a given data set. Specifically, we investigated how certain hyperparameters can affect training and the number of samples needed to achieve the

Bayes error. Also in Chapter IV, we conduct exploratory studies on feature vectors processed from speech data and utilize this information to select an appropriate set of features. Altogether, we combine the results from previous chapters to pre-train DNNs with artificial data and later fine-tune with feature vectors processed from speech. We motivate this study by attempting to understand how DNNs can be improved for phoneme classification, and therefore, decrease the classification error on a widely used speech database.

# Chapter 2

# Materials and methods

## 2.1 Hidden Markov models for phoneme recognition

Current speech recognizers include HMMs to model the temporal structure of speech flow and GMMs for estimating the likelihood of a phoneme. GMM is a parametric probability density function represented as a weighted sum of Gaussian component densities. In this study, we use a frame-based approach to train a HMM-GMM framework to estimate the means and covariances of 39 phoneme distributions. The setup of the framework was provided in the hidden Markov model toolkit (HTK), a software package developed by [SY06] in Cambridge University .

## 2.2 Deep neural networks

Deep, feedforward neural networks have been successfully applied to applications in speech recognition [HDY$^+$12]. The basic framework of a DNN include three types of layers stacked in the following manner: an input layer, multiple hidden layers, and an output layer. Figure 2.1 illustrates an example of a feedforward network with units in the input layer ($\mathbf{x}_{\text{input}} = \mathbf{h}^{\mathbf{0}}$), followed by the *k*th hidden layer consisting of *d* units ($\mathbf{h}_d^k$), and an output layer ($\hat{\mathbf{y}}_{\text{output}} = \mathbf{h}^\ell$). The last layer of the neural network outputs posterior probability predictions for a realized model when given an observation, $x^i$. Unlike output values from hidden layers, units in the last layer transform values into probabilities via a softmax function Equation 2.2.

Each unit in a hidden layer is defined by Equation 2.1. A unit computes a linear transformation followed by a squashing nonlinearity. For example, a set of inputs $\mathbf{h}^{k-1}$ is projected to a different space by the weight matrix $\mathbf{W}^k$ and are then shifted by values in the bias vector $\mathbf{b}^k$. This result is sent through a saturating nonlinear function $\phi$ called an activation function,

$$\mathbf{h}^k = \phi\left(\mathbf{b}^k + \mathbf{W}^k, \mathbf{h}^{k-1}\right) \tag{2.1}$$

$$h_i^\ell = \frac{\exp(b_i^\ell + W_i^\ell h^{\ell-1})}{\sum_{j=1}^{J} \exp(b_j^\ell + W_j^\ell h^{\ell-1})} \tag{2.2}$$

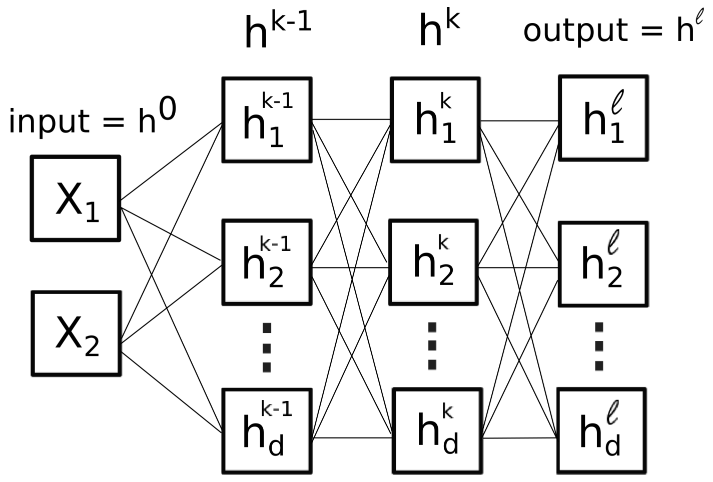

**Figure 2.1:** *Illustration of a feedforward neural network. A feature vector is representative of units in the input layer. Hidden units perform operations on incoming values sent from its previous layer and are squashed through a rectified linear function. Probabilities for class assignments are outputs from a softmax function only assigned in the last layer.*

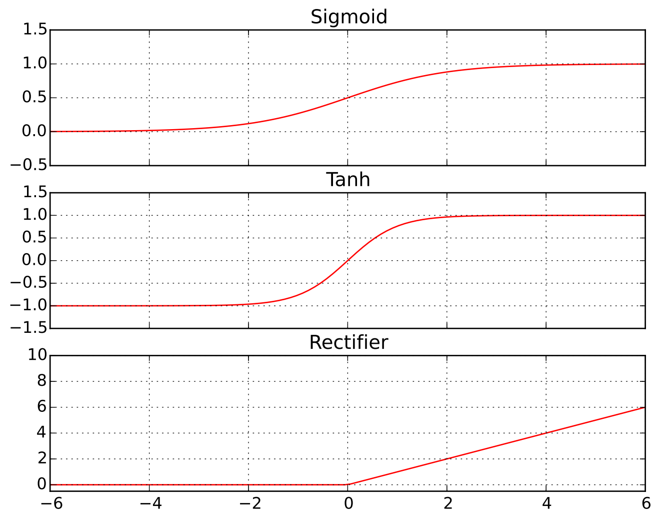

**Figure 2.2:** *Activation functions often cited in DNN studies: sigmoid, hyperbolic tangent, and Rectifier Linear*

Figure 2.2 shows three types of commonly used activation functions found in DNN literature: sigmoid, hyperbolic tangent, and rectified linear. In the next chapter, it is described how to properly work with certain types of activation functions in order to achieve good classification performance.

DNNs are discriminative classifiers and are trained with backpropagation and stochastic gradient descent [RHW88]. For computational efficiency, it is advisable to compute derivatives on small subsets of training data called "mini batches" rather than the whole training set itself [HDY$^+$12]. Weights are updated in proportion to the gradients and can be fine-tuned by directly applying hyperparameters such as learning rate, momentum term, and regularizers (e.g. L1- , L2-norms, or dropout).

### 2.2.1 Experimental setup

**Hardware, software, and libraries** Experiments were performed on NVIDIA GPUs and a Macintosh laptop. DNNs were implemented in Python using Theano libraries [BLP$^+$12] and extended from an open-source deep learning toolkit [Mia14].

**Initialization of weights** DNN weights were randomly drawn between:

$$W_{ij} \quad \sim \quad U\left[-\sqrt{\frac{6}{n_{\text{input}}+n_{\text{output}}}}, \sqrt{\frac{6}{n_{\text{input}}+n_{\text{output}}}}\right] \tag{2.3}$$

where $U[-a,a]$ is the uniform distribution in the interval $(-a,a)$, $n_{\text{input}}$ is the size of the previous layer, and $n_{\text{output}}$ is the size of the current layer. A random generator was initialized by a seed to make the results in these experiments repeatable.

**Dropout in neural networks** When applying dropout for training, the number of active weights signifies connections that were not randomly chosen by the dropout algorithm to be silenced. These numbers were computed as the product of two adjacent layers, $h^{k-1}$ and $h^k$, and a probability dropout term $P_{\text{dropout}}$ that certain connections will be turned off during an epoch.

$$w_{\text{active}} = (h_i^{k-1} \times h_j^k) \times (1 - P_{\text{dropout}}), \qquad i = 1...N, j = 1...M \tag{2.4}$$

where $w_{\text{active}}$ is the number of active weights during an epoch.

**Monitoring DNN performance** During training, the state of a network was measured by closely monitoring classification errors between training and validation sets. This method allowed us to diagnose DNNs for signs of underfitting or overfitting. In addition, we measured the generalization performance on a third set called the test. The test set serves as a measure for how good the generalization ability of a trained network actually is.

## 2.3 Datasets

Four types of data sets are used in this thesis of which three sets are generated artificially. Data generated from a normal distrubution is described first. A second set is natural speech data. In third place, data sets generated from Gaussian mixture densities (GMD) are discussed. Lastly, data generated from a HMM-based synthesis (HBS) method is discussed. A summary of all data sets can be found on Table 2.1.

| | Data set | Type | Dimensions | # Gaussian components | Bayes error ( % ) |
|---|---|---|---|---|---|
| 1) | Normal distribution | Artificial | 2 | – | 26 |
| | | | 13 | – | 22 |
| 2) | TIMIT | Natural | 273 | – | – |
| 3) | Gaussian mixtures (GMD) | Artificial | 39 | 1 | 40 |
| | | | 39 | 12 | 28 |
| | | | 273 | 12 | $\sim 1$ |
| 4) | HMM-based synthesis (HBS) | Artificial | 273 | – | – |

**Table 2.1:** *Summary of data used in this thesis*

### TIMIT database

The TIMIT database contains natural speech data. It is the most widely used database for phoneme recognition [HDY+12], and has provided a convenient way of testing new approaches to speech recognition problems. TIMIT contains recordings of phonetically-balanced prompted English speech which are captured in a total of 6300 sentences (5.4 hours) [LP11]. A total of 630 speakers from 8 major dialects across the United States recited 10 sentences that were later manually segmented at the phoneme level [LP11]. Table 2.2 summarizes the TIMIT corpus training and test sets as described in [LP11]. A detailed description about the database itself can be found in [ZSG90].

The original number of phonemes in TIMIT was 61. Later proposals of how data in TIMIT should be utilized resulted in transforming the 61 classes into 39 phonemes, which has become standard practice. Table 2.3 presents 39 phonemes used for classification in this thesis.

| Set | Number of speakers | Number of sentences | Number of Hours |
|---|---|---|---|
| Training | 462 | 3696 | 3.14 |
| Core test | 24 | 192 | 0.16 |
| Complete test | 168 | 1344 | 0.81 |

**Table 2.2:** *Summary of TIMIT corpus training and test sets [ZSG90]*

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

**Table 2.3:** *A list of* 39 *phonemes*

## Data generated from normal distributions

Random samples were drawn from a multivariate normal distribution function in Python. Distributions were defined by their means and covariances which were drawn at random from a uniform distribution. Chapter III presents experiments for two subsets of normally distributed data. Each subset is different in dimensionality. Subset one included 4 clusters of 2-dimensions and subset two included 20 clusters of 13-dimensions. As shown in Figure 2.3, clusters were designed to overlap by some amount in order to generate data with Bayes error rates between 20 and 30%.

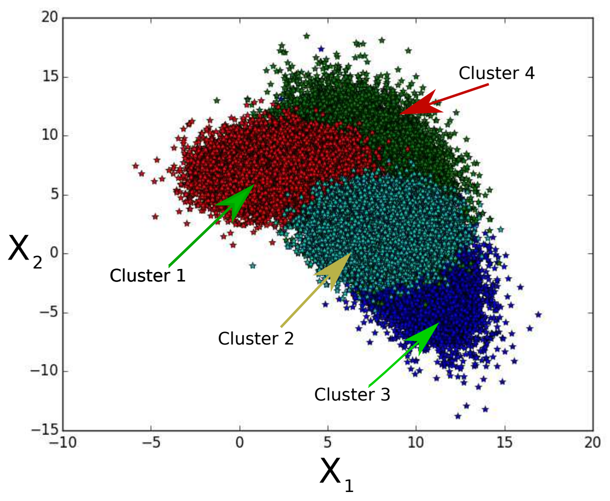

**Figure 2.3:** *Samples were randomly drawn from a* 2*-dimensional normal distribution. Four colors (red, blue, green, and cyan) label each cluster. Clusters intentionally overlap to give the dataset a Bayes error of* 26%

### Data generated from Gaussian mixture densities

A second type of artificial data are drawn from a mixture of Gaussians. Gaussian likelihoods were estimated with feature vectors created from natural speech data (TIMIT). Specifically, the HMM-GMM framework (provided in the HTK software) was used to estimate distributions for $39$ phonemes. One can specifiy the number of Gaussian components used to estimate the likelihood distribution of a phoneme.

In total, three subsets of data using GMD were created. Each subset differed according to the number of Gaussian components defined and its dimensionality. For example, a subset of feature vectors were created with only one Gaussian component and the dimensionality of the feature vector was $39$. A second subset of feature vectors were created with $12$ Gaussian components and its dimensionality was $39$. A third subset of feature vectors were created from reusing samples in subset two. This was accomplished by taking $7$ sequential samples from subset two, each of $39$ dimensions, and appending them together. The result was a feature vector with $273$ dimensions. Because samples in subset two were estimated from $12$ Gaussian components, samples in subset three are also documented with $12$ Gaussian components (Table 2.1). The Bayes error rates for all subsets were determined.

An important remark about samples drawn from this method is that values within a single feature vector were independently drawn at random. Unlike a feature vector processed from speech, discrete values are co-dependent and one value can influence another within a lag time. Therefore, it is incorrect to assume feature vectors generated from GMD closely resemble natural speech. Instead samples only contain information about the *estimation* of phoneme distributions by approximating them with Gaussians.

### Data generated with HMM-based speech synthesis

HBS for speech generation is a method for synthesizing a unit of speech. In this system, HMMs are trained from a natural speech database (TIMIT) to generate speech waveforms from HMMs themselves. This system offers the ability to create variations of speech without requiring a large database of recordings. The training part is similar to that used in speech recognition systems [ZNY$^+$07].

Speech waveforms generated from HMMs are speaker-dependent. This implies that each speech waveform was modeled from properties that differentiate speech across genders, dialects, and prosodic aspects. The system was trained on $461$ speakers. For each speaker, $500$ non-sensible sentences were artificially generated and were given labels at the phoneme level. Because the artificially generated data is speaker-dependent, it is impossible to determine the Bayes error rate of the data set. The advantage sought behind using this method is that data generation is cheap and contains properties that closely mimic natural speech data.

## 2.4 Feature extraction

Many feature extraction techinques exist for speech recognition. Here we used a common approach for converting natural speech waveforms into feature vectors called Mel-frequency cepstrals coefficients (MFCCs). MFCCs were chosen as the preferred features of choice because the filter banks are designed to resemble human hearing frequencies. Below we describe two stages to create $39$-dimensioal MFCC feature vectors from the TIMIT training set.

Initially, a raw waveform is partitioned into frames. The window size of the frame was $25$ msec and is shifted by $10$ msec across the file. The next step is to apply the discrete fourier transform to each frame, which is followed by performing a computation with a Mel-spaced filterbank. A total of $13$ triangular filters comprise the Mel-spaced filterbank. The final output of the extraction process is a $13$-dimensional MFCC vector.

The second stage focuses on how to compute delta and delta-delta coefficients, which are appended to the $13$-dimensional MFCCs computed in the last stage. In contrast to "static" MFCCs, deltas and delta-deltas carry "dynamical" information about frame-to-frame differences from the $13$-dimensional MFCC vectors. An intuitive way to understand delta features is that it relates to differential coefficients whereas delta-delta features contain information about the acceleration amongst frames.

Initial steps for computing the first set of deltas is taking a single MFCC vector and its three sequential vectors that come before it and three vectors that follow after. Coefficients from the same time position in the seven vectors are added together and divided by the number of total coefficients, in this case seven. The weighted sum of coefficients for that one time point is then representative of one delta coefficient. This step is repeated for all time points in a set of $13$-dimensional MFCC vector. The equation for computing delta vectors can be found under Equation 2.5.

$$d_t = \frac{\sum_{n=1}^{N} n\left(c_{t+n} - c_{t-n}\right)}{2\sum_{N}^{n=1} n^2} \tag{2.5}$$

Delta-deltas are computed in the same manner as delta coefficients. The dimensions of the final delta and delta-delta vectors are both $13$-dimensional. The last step is to append each $13$-dimensional MFCC vector to its corresponding delta and delta-delta vectors. The result is a $39$-dimensional MFCC-delta vector.

### Normalization

Data processing plays a crucial role in training a neural network. Normalization of all datasets was carefully considered as a factor that could affect how well a network were to train and perform on the test set. We investigated two types of normalization techniques, standardization and feature scaling, to determine which of the two would perform best when training DNNs.

### Standardization

Data are transformed to obtain a new data set with zero mean and a unit variance. It is important to note that the transformed data is not bounded between 0 and 1, which can affect the learning of a DNN designed with sigmoid and hyperbolic tangent activation functions. Data was transformed using a pre-processing function in the scikit-learn library for Python.

$$\mathbf{x}' = \frac{x - \mu}{\sigma} \tag{2.6}$$

### Feature scaling

Data was normalized using Equation 2.7. The normalized values are bounded between 0 and 1. A suitable function used for feature scaling can be found in Python's scikit-learn library.

$$\mathbf{x}' = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \tag{2.7}$$

# Chapter 3

# Exploratory studies of DNNs

## 3.1 Motivation

DNNs are powerful models that can be applied to a wide range of machine learning applications. However, knowing how to properly train DNNs on very complex and high-dimensional data requires a lot of experience. In this chapter, we show how a DNN can be trained to reach the Bayes error rate for low- and high-dimensional toy data. We attempt to address the implications of training a DNN with certain hyperparameters over others such as types of activation functions, small versus large learning rates, and the amount of regularization needed to reduce overfitting. These empirical results help to stage the following experiments with data generated from GMD and HBS which is far more complex and higher dimensional.

## 3.2 Case studies with normally distributed data

### 3.2.1 Activation functions

DNNs for phoneme recognition were traditionally implemented with sigmoidal hidden layers, but increasing evidence has shown that these nonlinear functions may not be the best choice for trainability of a network [GB10], [GBB11], [MHN13]. Here we conducted a series of experiments on normally distributed data of 2- and 13-dimensions to show that sigmoids can be well-behaved for data that is only bounded between $0$ and $1$. Otherwise a network trained on data not bounded between this interval will not train well and should then be normalized to achieve good results (Figure 3.1). Figure 3.1.C-D shows DNNs trained with normalized data that were feature scaled. It is shown that training curves for all networks are able to approximate the Bayes error rate at $22\%$. Figure 3.1.A-B illustrates the effect of training with un-normalized data and shows that training curves for networks with sigmoid and tanh cannot train well compared to rectified.

Based on these experiments, we observe that sigmoid and tanh can propagate forward in-

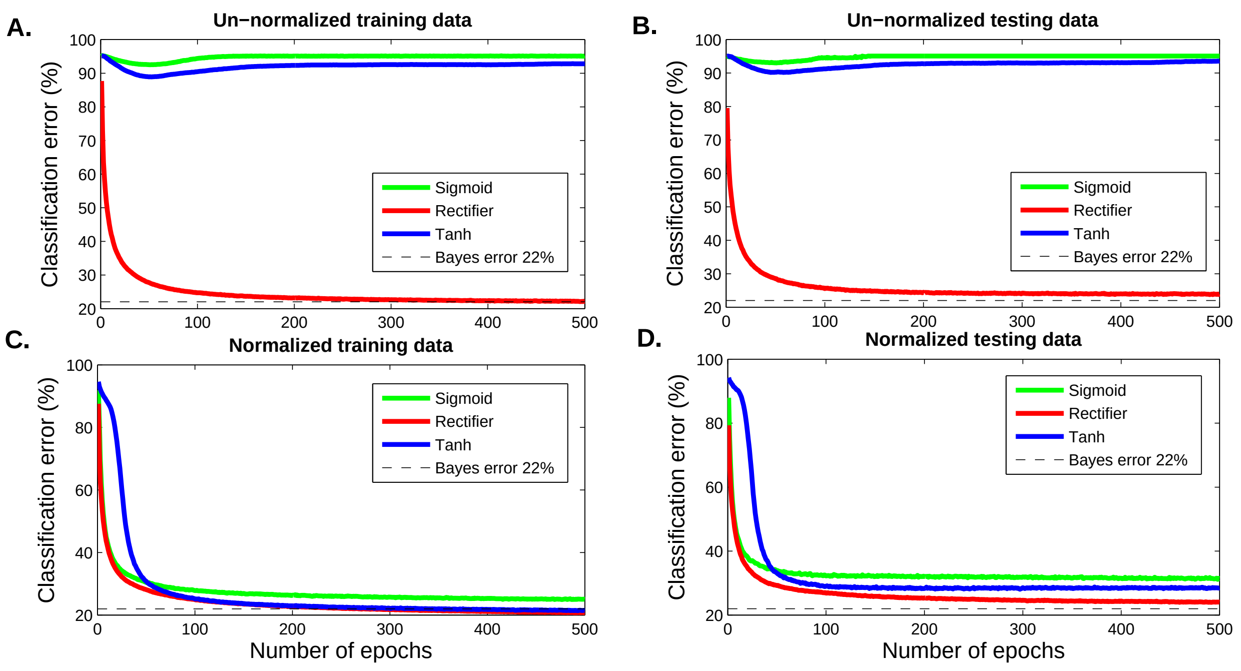

**Figure 3.1:** *Data used for normalization experiments was* $13$*-dimensional drawn from normal distributions. A) Networks were trained with un-normalized data. DNNs with sigmoid and tanh activation functions were not able to learn, while a network with rectified units succeeded to learn. B) Test curves for DNNs trained with un-normalized data. C) Networks trained with normalized data show that training is successful for all three functions. D) Test curves for DNNs trained with normalized data. The Bayes error rate is indicated by the black-dashed line at* $22\%$.

formative values only if data is processed in a particular manner. For instance, input values on the real number line that are very close to zero will be mapped to informative probabilities. If values are extremely large in either positive or negative directions, then ouputs of the sigmoid function are always mapped to either $1$ or $0$. Hence, a binary number does not give sufficient information about the example.

On the other hand, training curves for rectified networks show that they train well with either normalized or un-normalized data. Because these experiments show that rectified linear functions are reliable to train with, all networks discussed in the remainder of this thesis were defined with rectified linear units.

### 3.2.2 Overcoming overfit models

Introducing more parameters in a model than data points often leads to overfitting. Recent developed methods for regularization have shown improvements with training very large networks. Dropout is a regularization technique introduced by [HSK$^+$12] and has been used to prevent overfitting on large feed-forward neural networks by randomly omitting a subset of units during each epoch. In this section, we show how dropout can regularize large networks from overfitting and evaluate the behavior of networks that can reach the Bayes error rate without any regularization but are given dropout.

Figure 3.2.A presents four different network architectures trained on normal data of $13$-dimensions without dropout. It is shown that each network overfits on the training data, as indicated by the training curves dropping below the Bayes error at $22\%$ (black-dashed line). In constrast, the test curves increased its error relative to the Bayes error rate. To reduce the effect of overfitting (Figure 3.2.B), dropout is introduced into the largest network of $4$ hidden layers shown in Figure 3.2.A. Figure 3.2.B shows that omitting $50\%$ of units within each hidden layer of the largest network does reduce overfitting. Previous experiments on dropout have presented similar findings and noted that dropping $50\%$ of hidden units significantly improved classification for a variety of different network architectures [HSK$^+$12]. In this example, we conclude overfitting can greatly be reduced by using dropout to prevent "complex co-adaptations" between hidden units which are more likely to fit the training data too well.

A second set of experiments on dropout investigate the effect of dropping units on networks that do not need regularization. It is usually never the case that additional parameters are later included once a system has been found that best suits an objective. However, it is of interest here to evaluate how dropout can increase or decrease the classification error of networks that already have met the Bayes error. Figure 3.3 shows four experiments that were all performed on the same type of network architecture with different amounts of dropout added. Dropout was applied equally to all hidden layers of a network in amounts of $10$, $40$, and $80\%$. The control network was purposely designed to already have reached the Bayes error without dropout. As we expected, adding dropout to a network that does not require regularization increased the classification error.

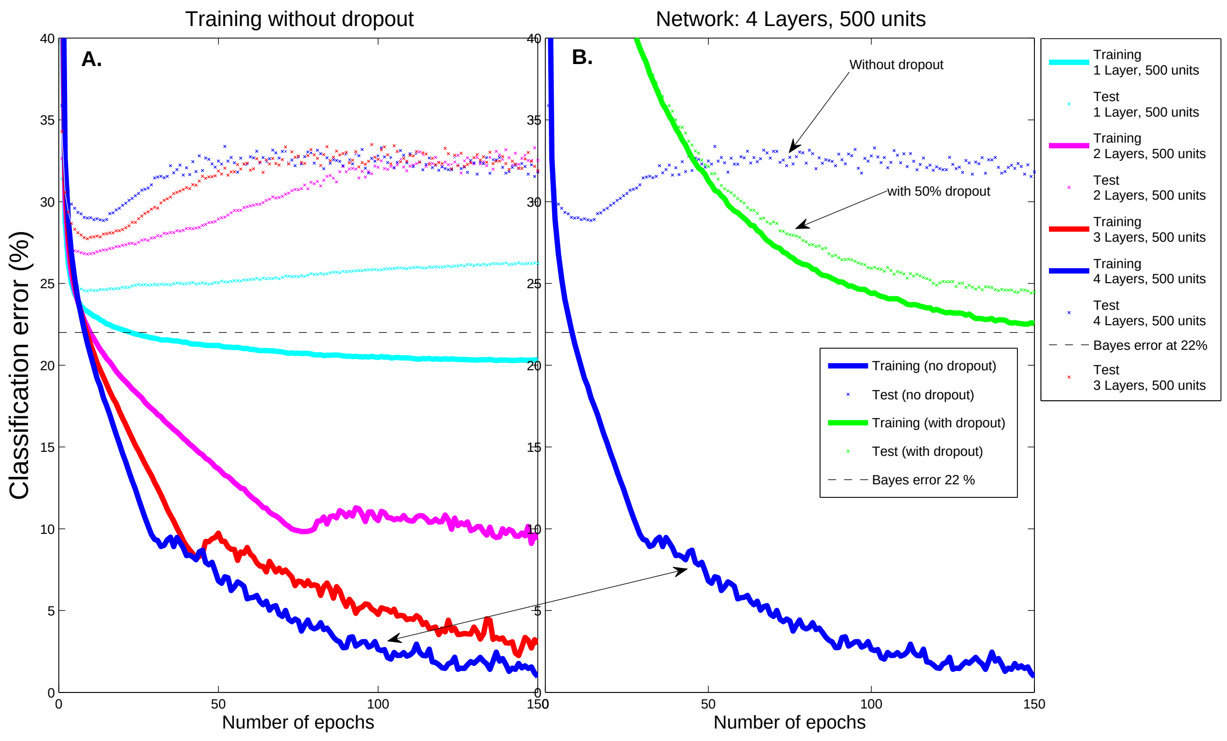

**Figure 3.2:** *Data used for dropout experiments was* 13*-dimensional drawn from normal distributions. A) DNNs with four different architectures were trained without dropout. Overfitting is present in all trainings. B) The largest network of four hidden layers shown in A (blue curves) was trained with* $50\%$ *dropout (green curves). Adding regularization to networks can prevent aggressive overfitting on the training data and achieve the Bayes error rate at* $22\%$

.

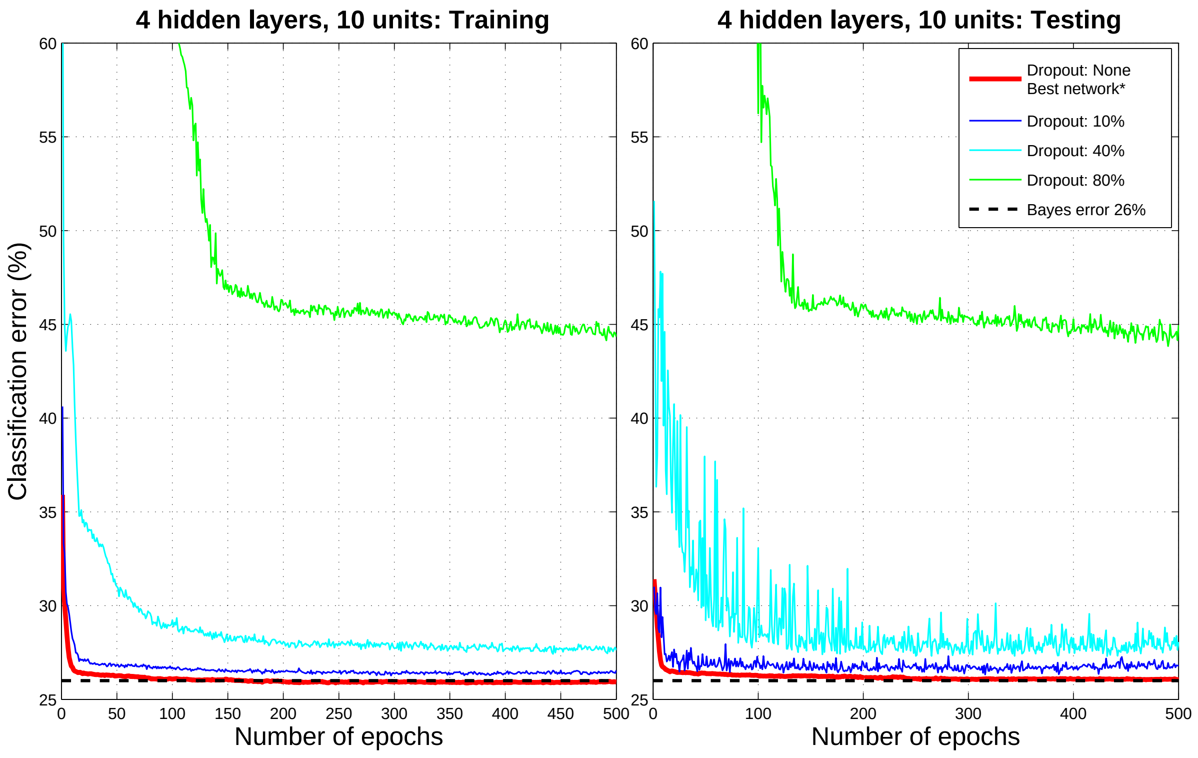

**Figure 3.3:** *Data used for dropout experiments was* $2$*-dimensional drawn from normal distributions. A total of four networks were trained with dropout at different quantities:* $0$, $10$, $40$, *and* $80\%$. *The bold red curve shows that the network had achieved the Bayes error at* $26\%$ *without dropout. Networks trained with dropout (blue, cyan, and green) show that regularization does increase the classification error on a network that has already achieved the Bayes error.*

### 3.2.3 Learning rates

Learning rates are difficult to handle properly, which is why many methods for training with good settings of learning rates have been proposed. Here we investigate the behavior of training a network under constant learning rates. Experiments with constant learning rates were chosen to provide an example of how training behaved under relatively high or low learning rates, without changing too many variables. Networks were trained with 2-dimensional data and identical network settings.

Figure 3.4.A-B shows networks trained with different constant learning rate settings. For example, adjusting the learning rate value at a relatively high value ($0.5$) can introduce large fluctuations in both the training and test curves (blue curves). Whereas, a small learning rate of $0.0005$ can dramatically increase the training period for a network to reach the Bayes error (magenta curves). The gray-dashed lines indicates training for the first $15$ epochs. We attempted to reduce the effect of large fluctuations after $15$ epochs by decreasing the learning rate by a factor of $10$, as shown in Figure 3.4.C-D. A slight improvement is observed for the blue test curve in Figure 3.4.D. The pair of learning rates starting at $0.005$ then reduced to $0.0005$ is shown to nearly eliminate large oscillations.

In constrast, learning rates manually changed from $0.05$ to $0.005$ remained with large oscillations in the test curves. The result of this hardcoded change shows that the learning rate does significantly contribute to oscillations found in the behavior of the test error curve. A third experiment shows a relatively smaller learning rate at $0.0005$ manually changed to a larger learning rate at $0.005$. The motivation behind this experiment was to determine if a small learning rate can converge faster to the Bayes error by increasing the value at $50$ epochs. Instead we found that the test error curve resulted in oscillating after the change.

For networks trained with data of $13$-dimensions, we find that a new set of learning rates behave differently when compared to learn rates chosen for $2$-dimensional data. For example, relatively high learning rates such as $0.05$ and $0.005$ result in networks that converged quickly to the Bayes error, but can overfit on the training data sooner than smaller learning rates (Figure 3.5). Too high of a learning rate such as $0.8$ shows to prevent the network from reaching the Bayes error.

In summary, the best learning rates were found using 5-fold cross-validation. It was unexpected to learn that the behavior of learning rates at different settings vary across data sets and architectures. A general method for selecting learning rates is difficult and an alternative method for handling these hyperparameters would encourage adaptive learning rates.

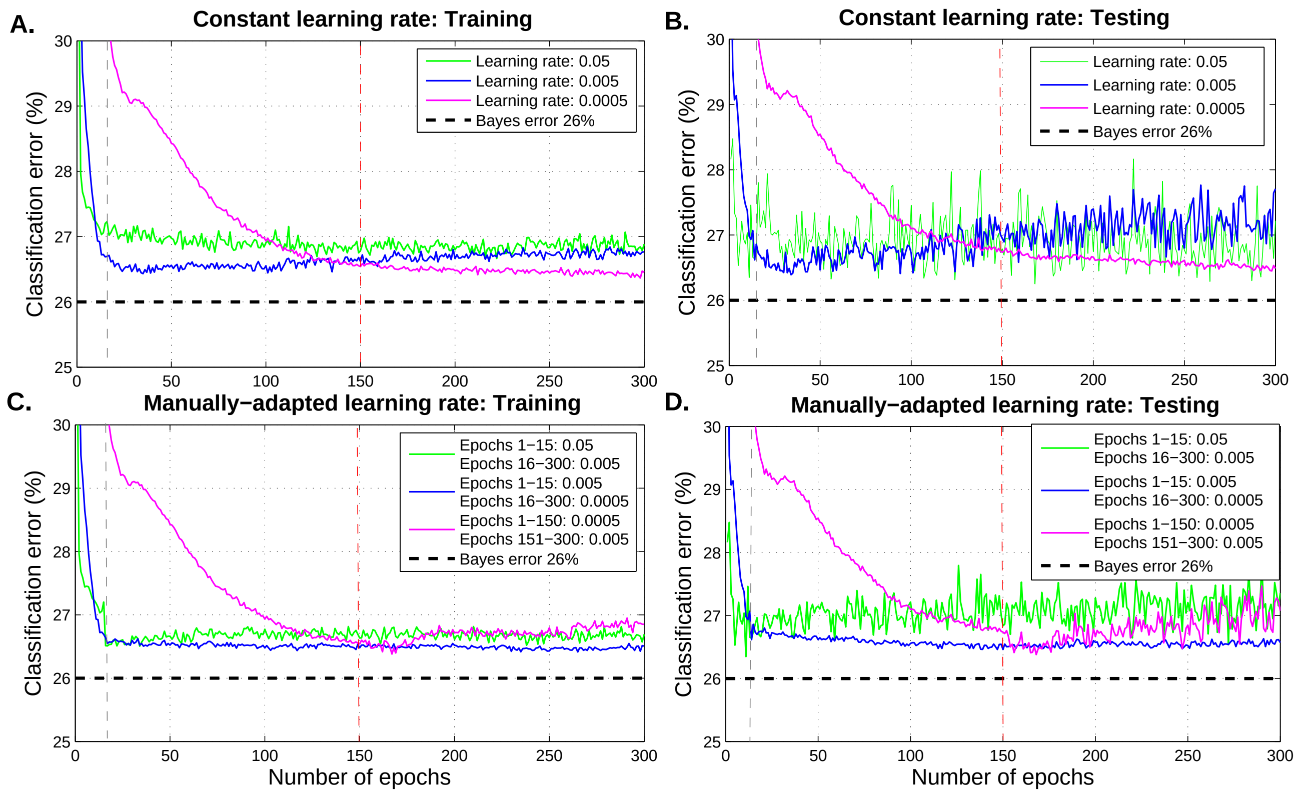

**Figure 3.4:** *Data used for learning rate experiments was* 2*-dimensional drawn from normal distributions. A) Comparison between three experiments for networks trained with constant learning rates of:* 0.05, 0.005, 0.0005*. DNNs with a relatively large learning rate* (0.05) *shows to reach the Bayes error rate sooner than smaller learning rates, but introduces fluctuations in the training curves. B) Testing curves for experiments in A. C) Experiments were extended from A and learning rates were manually changed at a specific epoch (indicated by the gray- and red-dashed vertical lines) to reduce flunctations in testing curves. D. Testing curves for networks in C that show learning rates can dramatically impact testing performance.*

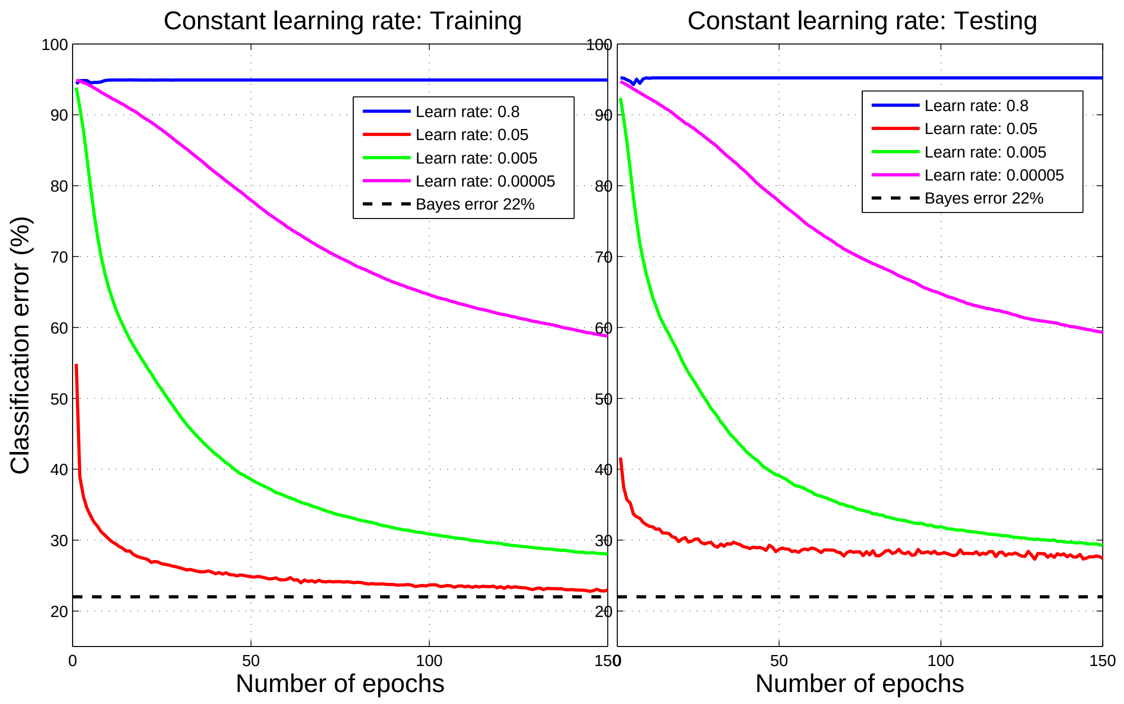

**Figure 3.5:** *Data used for learning rate experiments was* $13$*-dimensional drawn from normal distributions. A comparison is shown between DNNs trained with different constant learning rates. It is shown that too large of a learning rate* $0.8$ *is not effective for training and neither is too small of a learning rate* $0.00005$*. Too small of learning rate will take the network thousands of epochs to converge to the Bayes error rate at* $22\%$*.*

# Chapter 4

# Exploratory studies with artificial and speech data

## 4.1 Motivation

After reviewing how DNNs train under different hyperparameter settings, we further investigate the behavior of DNNs with data generated from GMD and HBS.

Normalizing all training, validation, and test sets in the same manner is important to reach good classification performance. Section 4.2 presents how GMD and HBS feature vectors were normalized in a manner that led to better classification performance on the TIMIT test. In Section 4.3, we are interested to determine whether the Bayes error can be reached for GMD data, and in 4.4 we investigate how many samples it took to reach the Bayes error. These exploratory experiments provide a foundation to later train a DNN with HBS and speech data, which do not come with knowing its Bayes error. In Section 4.5.1, we train a DNN with HBS data and report our best findings. In the last section, we pre-train or initialize a network's weights with either GMD or HBS, and later fine-tune the network with feature vectors converted from natural speech. Ultimately, we strive to decrease the classification error on the TIMIT test.

## 4.2 Normalizing with TIMIT

Normalization was carefully considered as a factor that could potentially prevent DNNs from reaching the Bayes error. However, the main objective in this chapter is to decrease the classification error on the TIMIT test, and the strategy taken to accomplish this goal begins with pre-training a DNN on either GMD or HBS feature vectors. This section motivates the question how GMD and HBS should be normalized to obtain good classification results when the DNN is fine-tuned with the TIMIT training set.

Table 4.1 compares two methods of standardizing GMD data. In this example, GMD feature vectors are 39-dimensional and were generated by 12 Gaussian components. A total of nine

| **Normalization** | | With TIMIT moments | | Without TIMIT moments | |
|---|---|---|---|---|---|
| **Model** (per hidden layer) | Hidden layers | GMD test error | TIMIT test error | GMD test error | TIMIT test error |
| 500 units | 1 Layer | 34.3612 | 50.0141 | 32.5765 | 57.6891 |
| | 2 Layers | 30.7946 | 49.3624 | 28.8271 | 56.5822 |
| | 3 Layers | 30.1687 | 49.5565 | 27.4885 | 56.4970 |
| 1000 units | 1 Layer | 33.4994 | 49.6896 | 31.1436 | 57.2542 |
| | 2 Layers | 30.1071 | 49.2318 | 25.3422 | 57.0078 |
| | 3 Layers | 30.0348 | 49.5349 | 22.4198 | 56.9802 |
| 2000 units | 1 Layer | 33.0348 | 49.7474 | 29.7390 | 57.1133 |
| | 2 Layers | 29.9948 | 49.4788 | 27.8450 | 57.7188 |
| | 3 Layers | 31.0464 | 50.0242 | 9.4809 | 58.1052 |

**Table 4.1:** *Data used for normalization experiments was* $39$*-dimensional generated from* $12$ *Gaussian components (GMD method). Data underwent two different standardization procedures: data standardized with moments computed from the TIMIT training set (columns* $3$ *and* $4$*), and data standardized with moments computed GMD training data (columns* $5$ *and* $6$*). The outcome of training a DNN with data standardized with TIMIT moments shows to decrease the test error on TIMIT.*

different network systems were trained to approximate the Bayes error rate at $28\%$. The classification error results are divided into two testing experiments: (1) showing that the GMD test can reach its Bayes error, and (2) testing how well TIMIT performs on networks trained with GMD standardized by two approaches.

The first approach, marked by the columns labeled "With TIMIT moments", was to compute the mean and standard deviation of the TIMIT training set. With these TIMIT moments, they are used to standardize GMD sets and the TIMIT test set. We show that under this standization approach the GMD test closely approximates the Bayes error. Networks with one hidden layer show that achieving the Bayes error was a bit more challenging than deeper networks (33-34$\%$ error). The results on the TIMIT test are about $20\%$ higher than the GMD test errors, but the TIMIT test errors show that information in the GMD training set can be transferred to a DNN. DNNs performed better than random on the TIMIT test.

The second approach, found under "Without TIMIT moments", does not standardize GMD with TIMIT moments but instead was standardized with moments computed from the GMD training set. The GMD test was standardized with GMD training moments, but the TIMIT test remains standardized with moments computed from the TIMIT training set. As expected, classification errors on the TIMIT test was much greater when DNNs are trained with training data not standardized with TIMIT moments. On the other hand, larger networks such as three hidden layers with units of $1000$ and $2000$ show that the GMD test errors fall below its Bayes error rate. One probable explanation for this behavior was that by chance the GMD test set closely resembles examples in the GMD training set.

## 4.3 Bayes error rate estimation with DNNs

DNNs trained with GMD subsets met the Bayes error rates with specific hyperparameter settings. Refer to Table 2.1 for details about the three subsets listed under GMD. Figure 4.1 plots the mean classification error curves (bold lines) and standard deviations (shaded regions) for each subset and is compared against its Bayes error rate. It is shown that on average the subsets were able to achieve the Bayes error rate with hyperparameters and network architectures listed on Table B.10 in Appendix B. The best settings for hyperparameters were found by performing a 5-fold cross validation. Networks fitted on a partition of the entire training set were tested on five test sets. Test errors for the best network were averaged and its standard deviation was computed.

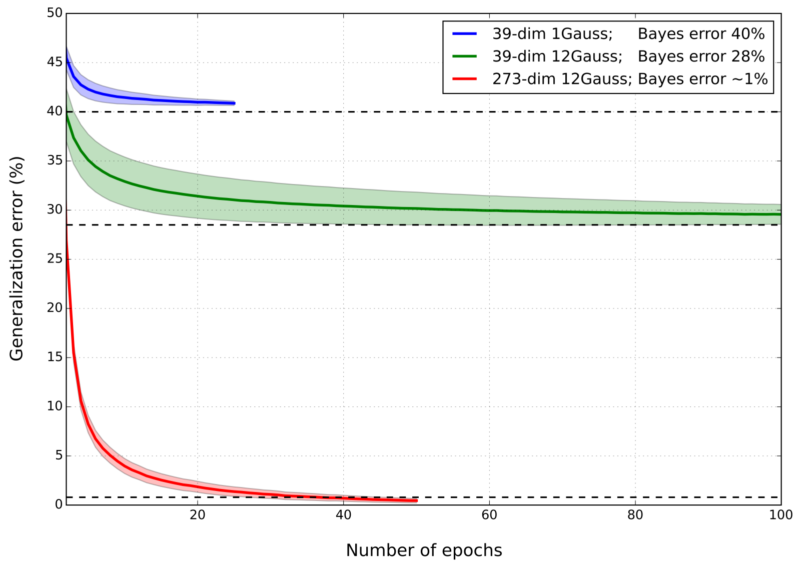

**Figure 4.1:** *Test errors for the best networks show to achieve Bayes error rates for three types of GMD feature vectors. Trained networks were tested on five test sets and were averaged (shown by the bold line). Their standard deviations are denoted by the lightly-shaded regions. The best DNNs were selected by performing $5$-fold cross validation.*

In addition, networks trained with at least one million samples were able to reach the Bayes error rate (Table 4.2). Interestingly, GMD of 39-dimensions generated by 12 Gaussian components needed about $2$ million more samples to reach its Bayes error than the other two subsets. Figure 4.1 also shows that the standard deviation for meeting the Bayes error at $28\%$ was greater than subsets with Bayes errors at $40\%$ and $1\%$. An explanation for why DNNs trained with 39-dimensions generated by $12$ Gaussians need a larger sample size to reach its Bayes error at $28\%$ is not well understood.

| GMD data (dim, # Gaussians) | # Samples |
|---|---|
| 39d, 1 | 1,170,000 |
| 39d, 12 | 3,000,000 |
| 273d, 12 | 1,950,000 |

**Table 4.2:** *Minimum number of samples required to reach the Bayes error rate on Figure 4.1*

## 4.4 Relationship between sample size and Bayes error rate

As it was shown in the previous section, the number of samples needed to reach the Bayes error rate was at least one million. Here we aim to find a general relationship that could estimate the test error for a DNN given a certain sample size.

The approach used to determine a general relationship between test error and sample size was to first find the best networks that could meet the Bayes error rates. This was accomplished by performing $5$-fold cross validation to determine the best network settings. If the Bayes error rates were not met, then the sample size was increased. Once the best networks were found, we tested the performance of these networks on different sample sizes. The best networks served as a base to then find a DNN that can give good test results on a smaller sample set. For example, we divided a large pool of $3900000$ samples into five subgroups used for training. Each sample was randomly selected from the pool and assigned randomly to one of five groups. The sample sizes of each subgroup were: $195$, $1950$, $19500$, $195000$, and $1950000$. The remaining samples were assigned as a validation set to detect underfitting or overfitting. A second cross-validation step was performed on each subgroup to find the best DNN for that particular sample size. The best network was selected for each subgroup and was tested on four test sets. The sample size of each test set was $487500$.

Figure 4.2 shows the relationship between sample size and the average test error for the best selected networks. A logarithmic relationship is found that relates a sample size of at least one million to an error estimated to be below the Bayes error rate at $1\%$ (denoted by the horizontal red-dashed line). This curve estimates for networks trained with less than $100000$ samples to expect errors higher than the Bayes error rate. Table 4.3 summarizes the network complexities versus sample size for GMD of $273$-dimensions generated by $12$ Gaussian components. The last column in Table 4.3 lists the means and standard deviations for the test error of all subgroups. The network complexity is reported as two different parameters: $(1)$ number of active weights during an epoch, and $(2)$ total number of weights used in the network architecture. Recall that the term *active weights* referred to units that were not randomly chosen by the dropout algorithm to become silenced during an epoch. The number of *active weights* were computed with Equation 2.4.

We conclude that the Bayes error rate is achievable when the network has been trained with at least one million samples. This statement is only applicable to samples generated from the GMD method.

| # samples | # active weights | # total weights | test error (%) |
|---|---|---|---|
| 195 | 109,200 | 156,000 | $\mu$ =77.89, $\sigma$ =0.0726 |
| 1,950 | 338,450 | 406,000 | $\mu$ =38.01, $\sigma$ =0.0796 |
| 19,500 | 302,400 | 604,800 | $\mu$ =7.64, $\sigma$ =0.0633 |
| 195,000 | 528,320 | 604,800 | $\mu$ =1.82, $\sigma$ =0.6277 |
| 1,950,000 | 528,320 | 604,800 | $\mu$ =0.93, $\sigma$ =0.6593 |

**Table 4.3:** *Summary of statistics for five subgroups plotted in Figure 4.2. Data used for these experiments was GMD data of* 273*-dimensions generated by* 12 *Gaussian components. Neural network weights are expressed in two categories:* (1) *number of active weights during training (refer to Equation 2.4), and* (2) *total number of weights defined in a DNN.*

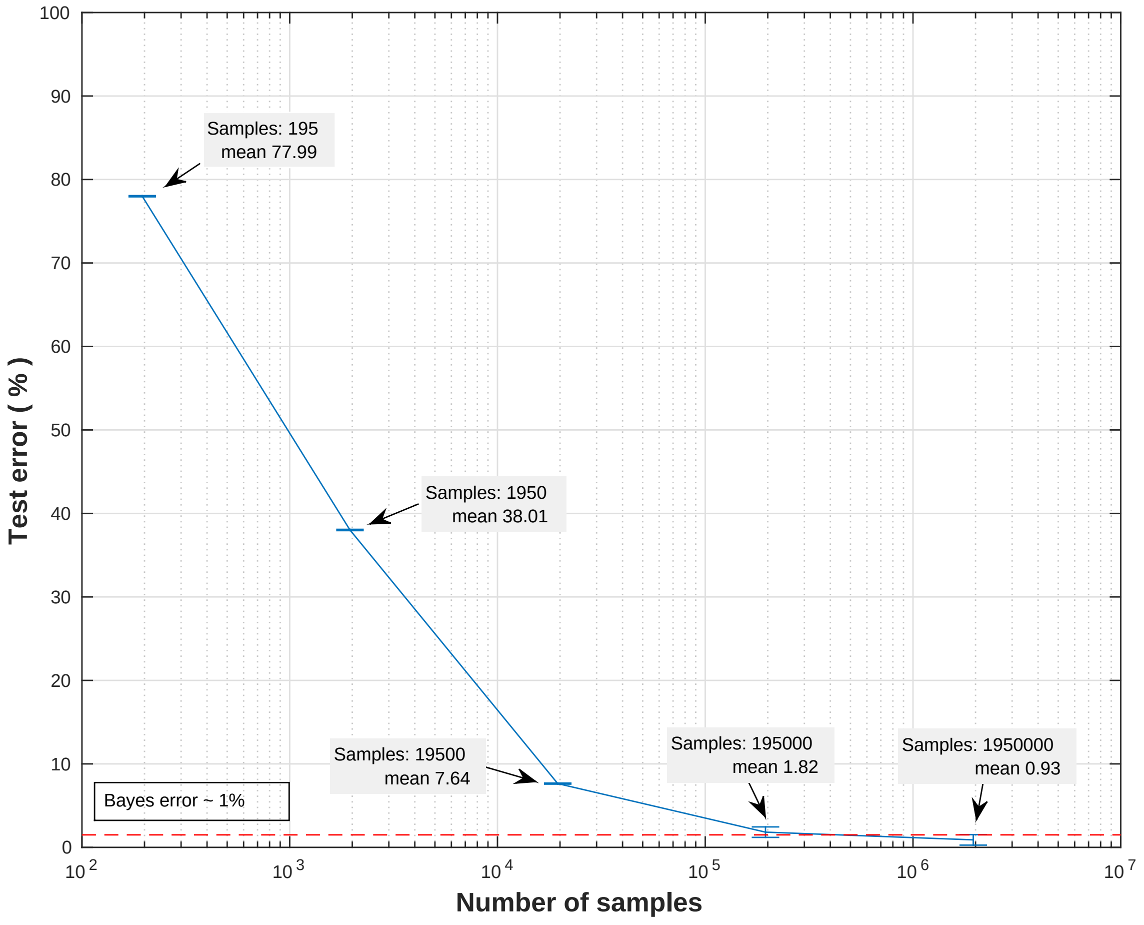

**Figure 4.2:** *Relationship between sample size and generalization error for the best DNNs chosen under* 5*-fold cross validation. Feature vectors of* 273*-dimensions generated by* 12 *Gaussian components were randomly assigned to one of five groups for subsets of size:* 195, 1950, 19500, 195000, 1950000. *A logarithmic relationship is identified when plotting the mean error for all subsets.*

## 4.5 Evaluation of feature properties for pre-training

Before pre-training a DNN with task A and later fine-tuning the same network with task B, we evaluate whether task A and task B share similar properties. In this example task A is GMD or HBS data and task B is TIMIT data. We approach this question by performing the normalization procedure described under Section 4.2 and determine whether TIMIT test errors are better than random. If the TIMIT test errors are better than random, then it proves that some underlying properities can be shared between these two sets. Tables 4.4, 4.5, 4.6 show results for nine DNNs systems trained with hyperparameter settings that met their Bayes errors. It is shown that TIMIT test errors are better than random for classifying $39$ phonemes. This implies that pre-training a DNN with GMD or HBS can potentially offer a better set of initial weights for fine-tuning the network with speech data.

| **Dataset: 39d 1 Gaussian** | | | |
|---|---|---|---|
| **Model** | Hidden layers | GMD test error $(\%)$ | TIMIT test error $(\%)$ |
| | 1 Layer | 40.6174 | 59.3503 |
| 500 units | 2 Layers | 41.6356 | 59.5539 |
| | 3 Layers | 41.5338 | 59.7062 |
| | 1 Layer | 40.7907 | 60.0577 |
| 1000 units | 2 Layers | 41.9258 | 60.3136 |
| | 3 Layers | 41.8871 | 60.1869 |
| | 1 Layer | 41.0507 | 60.2529 |
| 2000 units | 2 Layers | 42.6376 | 60.2667 |
| | 3 Layers | 43.1976 | 60.3845 |

**Table 4.4:** *Test errors for DNNs trained with GMD of* $39$*-dimensions generated by* $1$ *Gaussian component. GMD test errors approximate the Bayes error rate at* $40\%$*. TIMIT test errors are higher by* $20\%$*, but show to perform better than random.*

| **Dataset: 39d 12 Gaussians** | | | |
|---|---|---|---|
| **Model** | Hidden layers | GMD Test error $(\%)$ | TIMIT test error $(\%)$ |
| | 1 Layer | 34.3612 | 50.0141 |
| 500 units | 2 Layers | 30.7946 | 49.3624 |
| | 3 Layers | 30.1687 | 49.5565 |
| | 1 Layer | 33.4994 | 49.6896 |
| 1000 units | 2 Layers | 30.1071 | 49.2318 |
| | 3 Layers | 30.0348 | 49.5349 |
| | 1 Layer | 33.0348 | 49.7474 |
| 2000 units | 2 Layers | 29.9948 | 49.4788 |
| | 3 Layers | 31.0464 | 50.0242 |

**Table 4.5:** *Test errors for DNNs trained with GMD of* $39$*-dimensions generated by* $12$ *Gaussian components. GMD test errors approximate the Bayes error rate at* $28\%$*. TIMIT test errors are higher by* $20\%$*, but show to perform better than random.*

| **Dataset: 273d 12 Gaussians** | | | |
|---|---|---|---|
| **Model** | Hidden layers | GMD test error (%) | TIMIT test error (%) |
| | 1 Layer | 2.5733 | 47.8853 |
| 500 units | 2 Layers | 2.0641 | 47.3661 |
| | 3 Layers | 1.7471 | 47.0167 |
| | 1 Layer | 2.4492 | 47.8966 |
| 1000 units | 2 Layers | 2.0323 | 47.4043 |
| | 3 Layers | 1.7164 | 47.2042 |
| | 1 Layer | 2.3015 | 47.6253 |
| 2000 units | 2 Layers | 2.0005 | 47.3783 |
| | 3 Layers | 1.7328 | 46.9292 |

**Table 4.6:** *Test errors for DNNs trained with GMD of* $273$*-dimensions generated by* $12$ *Gaussian components. GMD test errors approximate the Bayes error rate at* $1\%$*. TIMIT test errors are higher by at least* $40\%$*, but show to perform better than random.*

### 4.5.1 Exploratory studies on DNNs with HMM-Based synthesis

HMM-based synthesis (HBS) is a technique used to generate an infinite amount of phoneme waveforms from which one can create sentences. In this thesis, we used HBS to generate a large sample size of phoneme waveforms to pre-train DNNs with. Similar to WAV files found in the TIMIT database, HBS phoneme waveforms are converted to feature vectors of $39$ dimensions (Section 2.4). On the other hand, because the variability between phonemes is high for samples generated from the HBS method, it is not possible to compute the Bayes error rate. Instead we attempt to train DNNs with HBS using techniques discussed in previous chapters and sections.

Figure 4.3 shows the classification performances for the best networks found under $5$-fold cross-validation for features with $39$- and $273$-dimensions. The test curves are shown to reach an error about $3\%$ for both types of feature vectors. The best networks were regularized with applying $20\%$ dropout to each hidden layer and early-stopping at $150$ epochs. Even though the networks reached a low training and testing error of about $3\%$, the number of speakers that were used to train the network was about one-third of the entire HBS data set. In Section 2.3, it was described that $500$ sentences were created for each speaker and processing these waveforms into feature vectors resulted in a database of over one billion samples. Because of memory space, DNNs were trained with one billion processed HBS feature vectors.

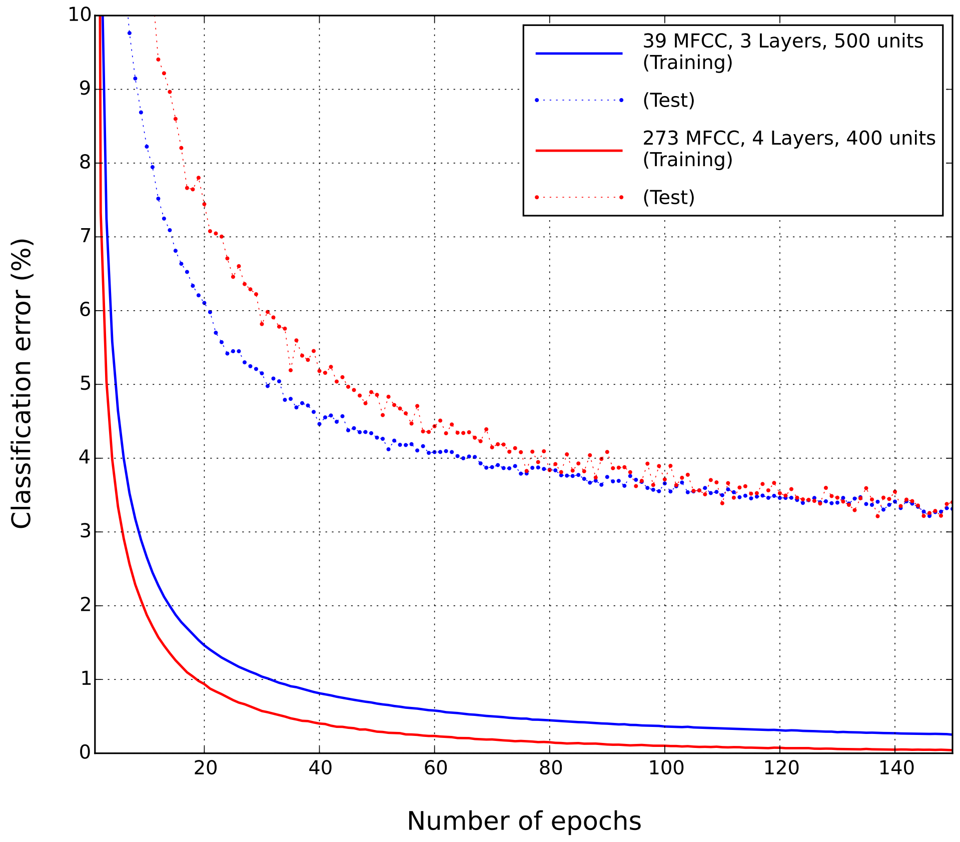

**Figure 4.3:** *DNNs trained with HBS feature vectors of* 39*-dimensions (blue curves) and* 273*-dimensions (red curves). Training and testing curves are indicated by the solid and dotted lines, respectively. The Bayes error rate is unknown for this data set, but the best performance found was at* $3.2\%$ *for both types of feature vectors*

## 4.6 Analysis of MFCC vectors

### 4.6.1 Importance of delta coefficients

In this section, we are interested to identify important coefficients in feature vectors that were processed from TIMIT in order to improve phoneme classification on the TIMIT test. Recall that a MFCC feature vector included 13 *static* MFCC and 26 *dynamical* delta coefficients. These experiments were performed by comparing test results for DNNs that were trained with and without delta coefficients. We hypothsized that DNNs trained with deltas would result in a lower classification error, and suggest that deltas are beneficial for improving phoneme classification. DNNs used in the control experiment were trained with MFCC vectors that included both MFCC and delta coefficients. We chose to train with feature vectors of 273-dimensions generated from the GMD method, because it scored the lowest error on the TIMIT test (Table 4.6).

Removal of deltas in a $273$-dimensional vector was performed by taking apart the $7$ sequential vectors that comprise the vector, each sub-vector was $39$-dimensions. For each $39$-dimensional vector, the last $26$ elements that make up delta coefficients were removed. The last step was to re-append the remaining $13$-dimensional MFCC vectors together and the final vector turns into $91$-dimensional.

However, we find that it is not an accurate comparison between the removal of delta coefficients resulting in $91$-dimensions and the control MFCC-delta $273$-dimensional feature vector. Because the inputs layers of the DNNs are different, this makes the network complexities not comparable. Instead we included an additional experiment that allows us to make a fair comparison between feature vectors. In this experiment, we investigated whether limiting the number of feature vectors used to compute delta coefficients would affect the classification error on the TIMIT test. The motivation behind this approach was to limit the amount of contextual information deltas carry to very little, enough to suggest they are insignificant. Under Section 2.4, specifically Equation 2.5, it shows that for one delta coefficient it is dependent on the total number of contributing elements provided, which depends on the number of $13$-dimensional vectors that are used in this equation. Here we decided to limit the number of adjacent MFCC vectors to 7. In Figure 4.4, this particular experiment is referred to as "7-framed MFCC+delta features" (blue line). The code used for calculating delta coefficients is provided under Appendix A.1.

Figure 4.4 shows that the removal of delta coefficients in MFCC vectors and limiting frame-to-frame information provided in deltas both have a higher classification error on the TIMIT test compared to the control case. A large difference gap of about $5\%$ can be observed between the control case and deltas that were either removed or limited to a small scope of frames.

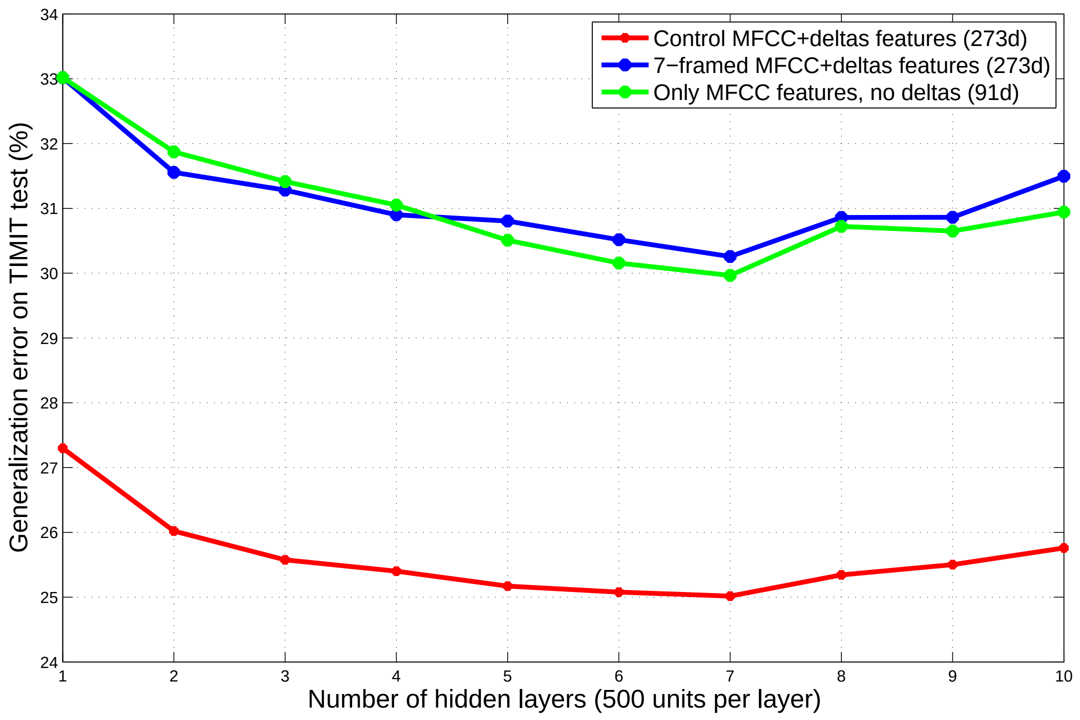

**Figure 4.4:** *Comparison between DNNs trained with feature vectors that contain delta coefficients (red curve) and the removal of deltas (green curve). A third type of feature vector was generated by computing deltas from a limited number of frames, in this case 7 (blue curve). Data used for these experiments were feature vectors converted from natural speech waveforms (TIMIT). The control experiment contains both delta and MFCC coefficients (red curve). It is shown that delta coefficients are valuable for decreasing the classification error on the TIMIT test set.*

### 4.6.2 Pre-training DNNs with artificial data

The last section in this chapter aims to answer if pre-training a DNN with feature vectors generated from GMD and HBS methods would improve the classification error on the TIMIT test. The intention behind pre-training a DNN is considered an alternative strategy to initializing weights for speech data. Figure 4.5 shows two types of pre-trainings with GMD feature vectors: (1) pre-training DNNs with vectors of 273-dimensions (blue curve), and (2) pre-training networks with vectors of 91-dimensions (green curve). The control case was a DNN trained with MFCC feature vectors that contained MFCC and delta coefficients (red curve). We hypothsized that a DNN pre-trained with GMD vectors with the same dimensionality as the control case would improve the classification error on the TIMIT test. Instead we found that pre-training overall did not improve the classification error much. A slight improvement is shown between hidden layers 3 and 6. Networks deeper than 6 hidden layers show to have higher test errors.

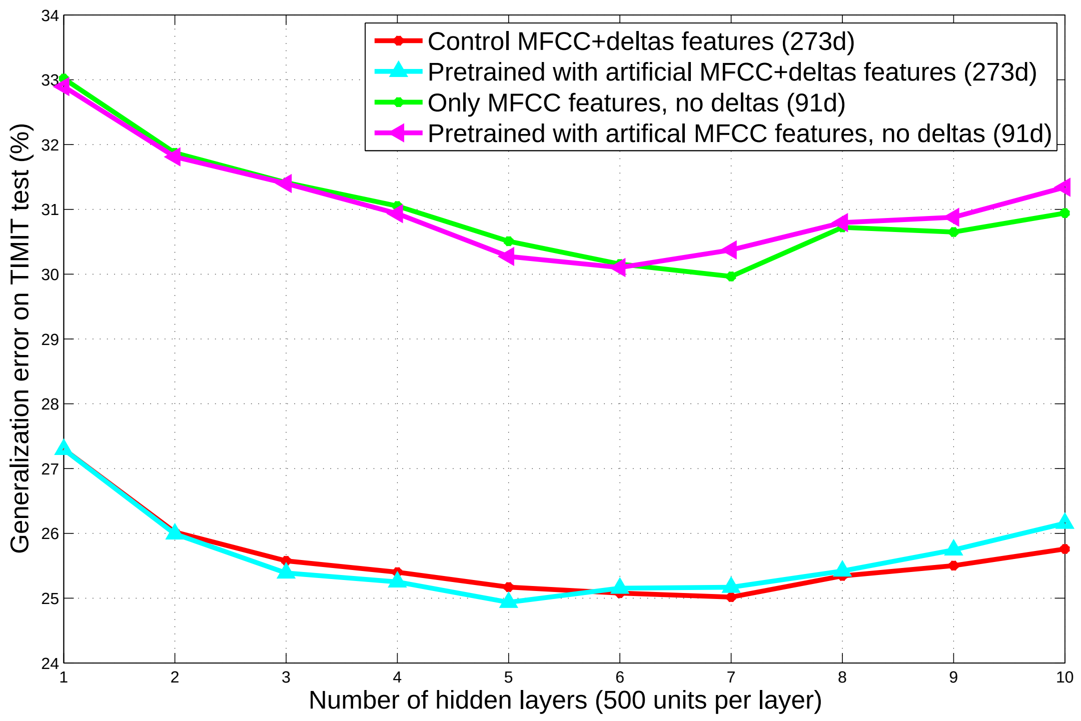

**Figure 4.5:** *Comparison between DNNs pre-trained with GMD feature vectors (cyan, triangular and magenta, triangular markers) and DNNs without pre-training. It is shown that pre-training does not decrease the test error much compared to the control case (red curve).*

Another pair of pre-training experiments attempted to relate the difference in error between a DNN trained with feature vectors that do not include deltas (same experiment described under Section 4.6.1) and pre-training a DNN with GMD feature vectors of the same dimension. As expected, Figure 4.5 shows pre-training does not help to improve the classification error for DNNs trained without deltas.

A third experiment compared the control case to a network pre-trained with feature vectors from the HBS method. Figure 4.6 shows that DNNs pre-trained with HBS feature vectors (green curve) do not improve the test error against the control DNN (red curve). Only a slight improvement is observed between hidden layers $3$ and $5$, but quickly led to poor performance of the deeper networks starting at 7 hidden layers.

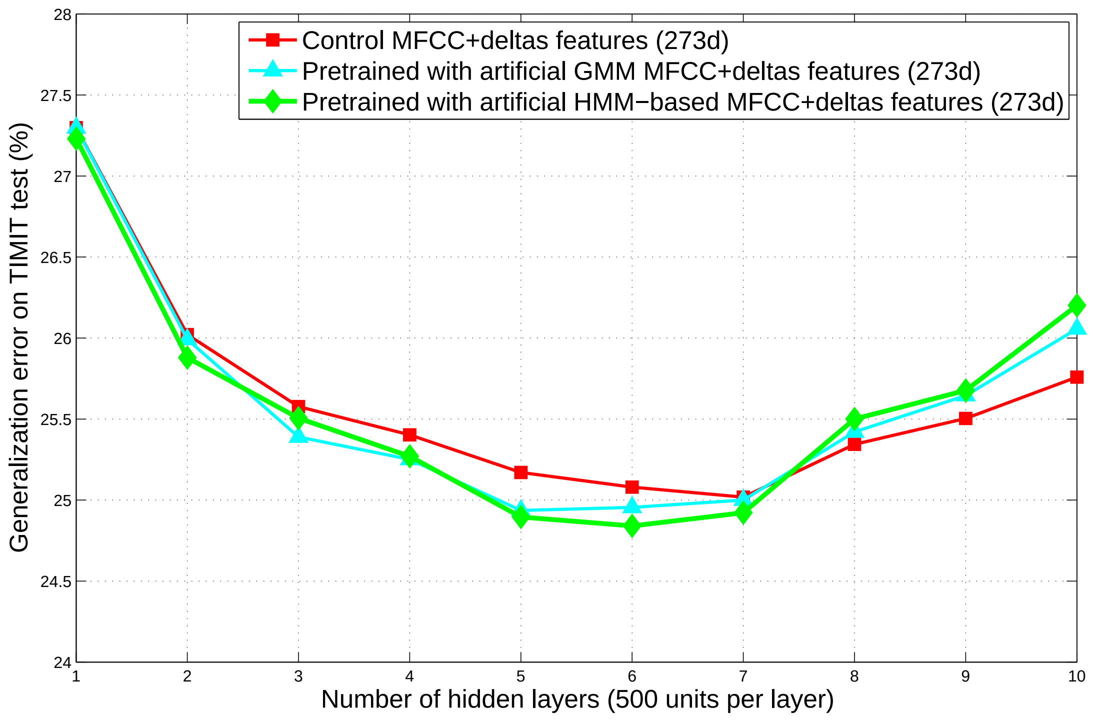

**Figure 4.6:** *Comparison between DNNs without pre-training (red curve) and DNNs pre-trained with GMD (blue, triangular markers) and HBS (green, triangular markers) feature vectors. It is shown that pre-training decreased on the TIMIT test error between hidden layers of* 3 *and* 6*, and increased the test error after* 6 *hidden layers.*

# Chapter 5

# Discussion

The application of modern DNNs for phoneme classification have proven advantageous, but a protocal for training a DNN is often left undocumented. In this thesis, exploratory studies were performed on DNNs to disambiguate how to train a relatively *deep* neural network by utilizing the Bayes error rate as an indicator of achieving optimality. This task was accomplished by training DNNs with artificial data generated from two methods for which the Bayes error rate was known: (1) samples drawn from a normal distribution, and (2) samples drawn from a mixture of Gaussian densities (GMD). The third chapter of this thesis discussed the behavior of DNNs for a range of hyperparameter settings. In Section 3.2.1, rectified linear units were shown to be a superior choice compared to sigmoid and tanh activation functions. For example, DNNs with rectified units were able to train well with both pre-processed and raw data. Meanwhile DNNs with sigmoid and tanh units did not decrease its error below $85\%$ for data that was not normalized (Figure 3.1). In addition, it was shown that performing standardization with moments from data similar to the test set is beneficial for decreasing the classification error (Section 4.2). Therefore, normalizing data in a methodical manner is a crucial step toward decreasing the test error.

During training, dropout was shown as an effective regularizer for reducing overfitting on networks of large complexities (Figure 3.2). It was found that regularizing a DNN with dropout can be approached by initially overfitting a network on the training set and then gradually apply dropout over time to decrease the effect of overfitting, as shown in Figure 3.2. A second useful approach that was applied for obtaining good weights is early-stopping. However, as expected, dropout is only useful for cases when overfitting appears during training. Otherwise dropout can potentially increase the classification error as shown in Figure 3.3.

Learning rates were a third type of hyperparameter that was investigated. As shown in Figures 3.4 and 3.5, it is challenging to assign only one learning rate for the entire training period. Particularly, too low of a learning rate shows that the test error slowly approaches the Bayes error rate and too high of a learning rate will prevent the network from learning on the training data. Even though cross validation was helpful for determining learning rate values, an adaptive learning rate would most likely be a suitable choice for training.

In Chapter IV, we experimented with deeper networks and data of higher complexities. We showed that DNNs can reach the Bayes error rate for GMD feature vectors with at least one million samples (Figure 4.1). A logarithmic relationship was identified between the sample sizes and estimated test errors, but one particular conclusion about the relationship is that the standard deviation increased as the test error approached the Bayes error rate. One possible explanation for this behavior could be a consequence of how samples were distributed into subgroups and test sets. Even though samples were distributed randomly, multiple similar samples could have been assigned to the same subgroup while few similar examples were representative in the test sets. In addition, delta coefficients were examined to evaluate their contribution toward decreasing the classification error on the TIMIT test set. It was found that MFCC feature vectors that include delta coefficients decrease the classification error by about $5\%$.

Altogether, we considered pre-training DNNs with GMD feature vectors of 273-dimensions, because the TIMIT test error was lowest compared to DNNs trained with feature vectors of 39-dimensions generated by 1 and 12 Gaussian components (Tables 4.4 and 4.5). The notion to pre-train was an alternative for initializing the weights of a DNN to a good configuration, from where we can fine-tune with MFCC feature vectors that included MFCC and delta coefficients. We find that an improvement was only achievable between three and six-hidden layers, even with close monitoring of DNNs during training. One reason for this result could be due to early-stopping during the fine-tuning stage. Early-stopping was applied between five and ten epochs and a few epochs may cause underfitting. Another assumption is that during the pre-training stage overfitting could have occurred with the GMD feature vectors. Tables 4.4 and 4.5 are networks that were used for pre-training with GMD feature vectors. The test errors listed under TIMIT test show an increase in classification error for deeper layers, which could indicate the overfitting was not controlled properly. This is a similar observation shown in the last 3 layers of Figures 4.4, 4.5, and 4.6.

In conclusion, we attempted to improve phoneme classification by pre-training DNNs with artificial "speech-like" data, but were not successful to decrease the test error by more than a percent. We hope to improve these experiments with advanced methods for monitoring DNN training and improvements on hyperparameters.

## Future directions

Future investigations would include understanding what prevented these deeper networks (of seven to ten hidden layers) to have a higher test error than the control case for the TIMIT test set. We would also consider convolutional neural networks (CNN) as an alternative method of processing speech into feature vectors and recurrent neural networks to improve phoneme classification.

# Appendix A

# Supplemental material

## A.1 MATLAB code

### Algorithm used for calculation of delta coefficients (deltaCal.m)

```
function new_mfcdelta = deltaCal(mfc_original)
[n_frame,n_coeff] = size(mfc_original);

vel_frame=(4:-1:-4)/60;
acc_frame=(1:-1:-1)/2;
ww=ones(5,1);

% compute deltas features
cx=[mfc_original(ww,:); mfc_original; mfc_original(n_frame*ww,:)];
velFeat=reshape(filter(vel_frame,1,cx(:)),n_framef+10,n_coeff);
velFeat(1:8,:)=[];

% compute delta-deltas features
accFeat=reshape(filter(acc_frame,1,velFeat(:)),n_frame+2,n_coeff);
accFeat(1:2,:)=[];

velFeat([1 n_frame+2],:)=[];

% append new features
new_mfcdelta = [mfc_original velFeat accFeat];
```

This example code was used to compute deltas and only applies to the window of length 5.

## A.2 Python code

### Algorithm used for calculation of the Bayes error rateut

```
import numpy as np
from numpy.linalg import inv, det

def bayes_error_rate( data ):

    ##### Compute likelihood of samples  ######
    # Compute posterior proportional likelihood probability P(x|w)
    # to assign sample to category
    likelihood=[]
    for i in range(data['classes']):
        coeff= (1 / np.sqrt(det(data['cov'][i])))
        x_m = data['samples'] - np.multiply(data['mean'][i],np.ones_like(
            data['samples']))
        x_m_cov= x_m * inv(data['cov'][i])
        exp_x= np.exp( -0.5 * np.multiply( x_m_cov, x_m ).sum(axis=1) )
        p_w_x = coeff * exp_x

        likelihood.append(p_w_x)    # storing likelihoods for all classes

    ###### Compare probabilities and assign labels #######
    pred=np.zeros((data['sample size'],1))
    val=likelihood[0]
    for i in xrange(1,data['classes']):
        L1=val  # initial condition
        L2=likelihood[i]
        booll=L1<L2
        for j in range(data['sample size']):
            if booll[j]==True:
                val[j]=L2[j]
                pred[j]=i

    ##### Compute number of mistakes - bayes error
    bayes= [pred==data['labels'].T]
    correct= np.sum(bayes)
    mistakes= data['sample size']-correct
    error_rate= float(mistakes) / ( data['sample size'] )

    return error_rate
```

This function was used to compute the Bayes error rate in Python 2.7.

# Appendix B

# DNN hyperparameter settings and architectures

Selection of hyperparameters for experiments in Chapters III and IV.

| **Line color** | **Hidden layers** | **Hidden units** | **Dropout** (%) | **Learn rate** | **Epochs** | **Activation func.** | **Batch size** | **Momentum** |
|---|---|---|---|---|---|---|---|---|
| green | 2 | 150 | none | 0.0005 | 500 | Sigmoid | 50 | 0.5 |
| red | 2 | 150 | none | 0.0005 | 500 | Rectifier | 50 | 0.5 |
| blue | 2 | 150 | none | 0.0005 | 500 | Tanh | 50 | 0.5 |

**Table B.1:** *Best selection of hyperparameters for Figure 3.1.A-B*

| **Line color** | **Hidden layers** | **Hidden units** | **Dropout** (%) | **Learn rate** | **Epochs** | **Activation func.** | **Batch size** | **Momentum** |
|---|---|---|---|---|---|---|---|---|
| green | 2 | 150 | none | 0.05 | 500 | Sigmoid | 50 | 0.5 |
| red | 2 | 150 | none | 0.0005 | 500 | Rectifier | 50 | 0.5 |
| blue | 2 | 150 | none | 0.005 | 500 | Tanh | 50 | 0.5 |

**Table B.2:** *Best selection of hyperparameters for Figure 3.1.C-D*

| **Line color** | **Hidden layers** | **Hidden units** | **Dropout** (%) | **Learn rate** | **Epochs** | **Activation func.** | **Batch size** | **Momentum** |
|---|---|---|---|---|---|---|---|---|
| cyan | 1 | 500 | none | 0.005 | 150 | Rectifier | 50 | 0.5 |
| magenta | 2 | 500 | none | 0.005 | 150 | Rectifier | 50 | 0.5 |
| red | 3 | 500 | none | 0.005 | 150 | Rectifier | 50 | 0.5 |
| blue | 4 | 500 | none | 0.005 | 150 | Rectifier | 50 | 0.5 |

**Table B.3:** *Best selection of hyperparameters for Figure 3.2.A*

| **Line color** | **Hidden layers** | **Hidden units** | **Dropout** (%) | **Learn rate** | **Epochs** | **Activation func.** | **Batch size** | **Momentum** |
|---|---|---|---|---|---|---|---|---|
| blue | 4 | 500 | none | 0.005 | 150 | Rectifier | 50 | 0.5 |
| green | 4 | 500 | 50 per layer | 0.005 | 150 | Rectifier | 50 | 0.5 |

**Table B.4:** *Best selection of hyperparameters for Figure 3.2.B*

| Line color | Hidden layers | Hidden units | Dropout (%) | Learn rate | Epochs | Activation func. | Batch size | Momentum |
|---|---|---|---|---|---|---|---|---|
| red | 4 | 10 | none | 0.005 | 500 | Rectifier | 50 | 0.5 |
| blue | 4 | 10 | 10 per layer | 0.005 | 500 | Rectifier | 50 | 0.5 |
| cyan | 4 | 10 | 40 per layer | 0.005 | 500 | Rectifier | 50 | 0.5 |
| green | 4 | 10 | 80 per layer | 0.005 | 500 | Rectifier | 50 | 0.5 |

**Table B.5:** *Best selection of hyperparameters for Figure 3.3*

| Line color | Hidden layers | Hidden units | Dropout (%) | Learn rate | Epochs | Activation func. | Batch size | Momentum |
|---|---|---|---|---|---|---|---|---|
| green | 2 | 10 | none | 0.05 | 300 | Rectifier | 50 | 0.5 |
| blue | 2 | 10 | none | 0.005 | 300 | Rectifier | 50 | 0.5 |
| magenta | 2 | 10 | none | 0.0005 | 300 | Rectifier | 50 | 0.5 |

**Table B.6:** *Best selection of hyperparameters for Figure 3.4.A-B*

| Line color | Hidden layers | Hidden units | Dropout (%) | Learn rate | Epoch switch | Epochs | Activation func. | Batch size | Momentum |
|---|---|---|---|---|---|---|---|---|---|
| green | 2 | 10 | none | 0.05 to 0.005 | 15 | 300 | Rectifier | 50 | 0.5 |
| blue | 2 | 10 | none | 0.005 to 0.0005 | 15 | 300 | Rectifier | 50 | 0.5 |
| magenta | 2 | 10 | none | 0.0005 to 0.005 | 150 | 300 | Rectifier | 50 | 0.5 |

**Table B.7:** *Best selection of hyperparameters for Figure 3.4.C-D*

| Line color | Hidden layers | Hidden units | Dropout (%) | Learn rate | Epochs | Activation func. | Batch size | Momentum |
|---|---|---|---|---|---|---|---|---|
| blue | 3 | 500 | none | 0.8 | 150 | Rectifier | 50 | 0.5 |
| red | 3 | 500 | none | 0.05 | 150 | Rectifier | 50 | 0.5 |
| green | 3 | 500 | none | 0.0005 | 150 | Rectifier | 50 | 0.5 |
| purple | 3 | 500 | none | 0.00005 | 150 | Rectifier | 50 | 0.5 |

**Table B.8:** *Best selection of hyperparameters for Figure 3.5*

| Dropout (%) | Learn rate | Epoch switch | Epochs | Activation func. | Batch size | Momentum |
|---|---|---|---|---|---|---|
| 10 per layer | 0.1 to 0.001 | 9 | 150 | Rectifier | 50 | 0.5 |

**Table B.9:** *Best selection of hyperparameters used for all nine experiments on Tables 4.1, 4.4-6. Networks shown on Figures 4.4-6 also used the same system.*

| Line color | Hidden layers | Hidden units | Dropout (%) | Learn rate | Epochs | Activation func. | Batch size | Momentum |
|---|---|---|---|---|---|---|---|---|
| blue | 3 | 500 | 20 | 0.005 | 25 | Rectifier | 50 | 0.5 |
| green | 2 | 500 | 10 | 0.008 | 100 | Rectifier | 50 | 0.5 |
| red | 3 | 400 | 10 | 0.01 | 300 | Rectifier | 50 | 0.5 |

**Table B.10:** *Best selection of hyperparameters for Figure 4.1*

| # samples | Hidden layers | Hidden units | Dropout (%) | Learn rate | Epochs | Activation func. | Batch size | Momentum |
|---|---|---|---|---|---|---|---|---|
| 195 | 1 | 500 | 0.2 | 0.0005 | 150 | Rectifier | 50 | 0.3 |
| 1950 | 2 | 500 | 0.2 | 0.01 | 120 | Rectifier | 50 | 0.1 |
| 19500 | 4 | 400 | 0.1 | 0.01 | 120 | Rectifier | 50 | 0.1 |
| 195000 | 4 | 400 | 0.1 | 0.01 | 50 | Rectifier | 50 | 0.1 |
| 1950000 | 4 | 400 | 0.2 | 0.01 | 50 | Rectifier | 50 | 0.1 |

**Table B.11:** *Best selection of hyperparameters for Figure 4.2*

| **Line color** | **Hidden layers** | **Hidden units** | **Dropout** (%) | **Learn rate** | **Epochs** | **Activation func.** | **Batch size** | **Momentum** |
|---|---|---|---|---|---|---|---|---|
| blue | 3 | 500 | 0.2 | 0.005 | 150 | Rectifier | 50 | 0.2 |
| red | 4 | 400 | 0.1 | 0.01 | 150 | Rectifier | 50 | 0.5 |

**Table B.12:** *Best selection of hyperparameters for Figure 4.5*

# Bibliography

[BLP$^+$12] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, 2012.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[HDY$^+$12] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-Rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[HSK$^+$12] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[LP11] Carla Lopes and Fernando Perdigão. Phone recognition on the timit database. *Speech Technologies/Book*, 1:285–302, 2011.

[MHN13] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[Mia14] Yajie Miao. Kaldi+pdnn: Building dnn-based ASR systems with kaldi and PDNN. *CoRR*, abs/1401.6984, 2014.

[RHW88] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5:3, 1988.

[SY06] Mark Gales Steve Young, Gunnar Evermann. *The HTK Book (for HTK Version 3.4)*. Cambridge University Engineering Department, 2006.

[ZNY$^+$07] Heiga Zen, Takashi Nose, Junichi Yamagishi, Shinji Sako, Takashi Masuko, Alan W Black, and Keiichi Tokuda. The hmm-based speech synthesis system (hts) version 2.0. In *6th ISCA Workshop on Speech Synthesis*, 2007.

[ZSG90] Victor Zue, Stephanie Seneff, and James Glass. Speech database development at mit: Timit and beyond. *Speech Communication*, 9(4):351–356, 1990.

(MA-2015-01)

Task description for the master thesis

of

Ms. Diana Ponce

Supervisors: Dr. Beat Pfister, TIK, ETH
Dr. Michael Pfeiffer, INI, ETH/UZH

Issue Date: February 16, 2015
Submission Date: November 16, 2015

# DNN-based Phoneme Models for Speech Recognition

## Introduction

Speech is basically a sequence of elements, the so-called speech phones. The characteristics of the phones depend heavily on the speaker, i.e. on his or her physiology, dialect, speaking habits, mood, health, etc. Furthermore, phones are influenced by their neighboring phones and they depend on the syllable stress level and on the position within a word and a sentence. Last but not least, speech signals are affected by deficiencies of the transmission from the speaker to listener or the microphone.

Speech recognition, no matter if done by machines or humans, basically assigns these highly variable phones to a fixed set of classes, the phonemes. However, the variability of the phones is much too high to allow accurate phoneme classification, also for humans. Hence, the result of phoneme recognition is highly ambiguous, i.e., it is neither clear how many nor which phonemes are contained in a given speech signal.

Humans resolve this ambiguity very efficiently by applying lexical and syntactic knowledge and with contextual information they can decide what makes sense. Speech recognizers need also additional knowledge to compensate for the deficiencies of phoneme classification. Lexical knowledge (pronunciation lexicon) and syntactic information (mostly probabilistic grammars or so-called n-grams) are standard components of todays speech recognizers, but deciding what makes sense is generally infeasible.

The aim of this master thesis is not to improve this disambiguation, however, but to investigate a rather new approach to better cope with the variability of phones and thus improve phoneme classification.

The classical approach to statistically describe temporal and spectral variability of phonemes are hidden Markov models (HMMs). An HMM consists of a first order Markov model and a state-specific Gaussian mixture model (GMM). The GMMs are used to approximate the statistical distribution of features that are extracted from the speech signal. Features for speech recognition are chosen in a way that variability is reduced. The most common features are mel frequency cepstral coefficients (MFCCs). MFCCs are largely independent of the level, phase and periodicity of the speech signal.

GMMs provide a statistical description of speech units in a maximum likelihood sense, which is not optimal to discriminate between similar phonemes. To improve the discrimination capability artificial neural networks (ANNs) can be used. There are several possibilities to combine ANN and HMM to a so-called hybrid architecture:

a) The ANN is trained to estimate the phoneme posteriors of feature vectors, e.g. MFCCs. In the speech recognizer these posteriors are then used instead of the likelihoods from the GMMs.

b) The ANN can be trained as a feature transformer, e.g. to reduce the dimension of the feature vector which is typically 40 or higher. The transformed features are then used in a HMM/GMM architecture, whereby the complexity of the GMMs is reduced with the dimension of the transformed features.
Note that the phoneme posteriors of item a) can also be seen as kind of transformed features and thus are applicable in an HMM/GMM architecture as well.

c) The ANN can be trained to calculate features from the raw speech signal. These features can again be used in an HMM/GMM architecture.

So far it has not been said, what kind of ANN can be used in which hybrid architecture. Commonly, for architectures a) and b) multi-layer perceptrons (MLPs) have been used. An MLP is a feed-forward ANN with one or two hidden layers. Recently, MLPs in theses architectures have been replaced by deep neural networks (DNNs) that are also feed-forward ANNs but with considerably more hidden layers.

For feature extraction as mentioned in architecture c), convolutional neural networks (CNNs) are used.

## ANNs for phoneme classification

ANNs are known to be able to learn complex transformations and classification tasks from high-dimensional inputs. Good results are only attainable, however, if at least the following conditions are met:

- the ANN has the appropriate architecture,
- there is suitable and enough training data available,
- we have a training method that finds a close to optimal set of ANN weights.

One important problem in training ANNs is overfitting. This is particularly the case in ANNs with a high number of weights, such as DNNs. Known methods to prevent overfitting are generative pretraining or dropout (see [1] and [2], resp.).

## Task description

In the framework of this master thesis it has to be investigated, how the training of DNNs has to be organized to achieve optimal performance, i.e., approximately Bayes classification rate. For such investigations it is recommended to work first with synthetic data. It is suggested to start from simple cases and then to increase the complexity towards speech-like data. Of particular interest is to see how DNNs compare with MLPs from a standard backpropagation training.

In the second part of the thesis the gained knowledge has be used to train a DNN with real speech data and apply it in a hybrid DNN/HMM speech recognizer (architecture a) or b), depending on the configuration of the CNN and the type of training performed). Furthermore, the DNN/HMM approach has to be compared with other models such as MLP/HMM and the standard HMM/GMM.

## Recommended procedure

This master project includes several works that are not independent and thus have to be organized reasonably. It is recommended to proceed as follows:

A. Getting acquainted with fundamentals (as far as necessary)

   Read some fundamentals about the extraction of MFCC features (either [3] or some other suitable text). Study the training of ANNs in general (e.g. [4]) and of DNNs in particular ([1], [5]).

B. Selection of a toolkit for DNN training and testing

   - Collect information about available ANN toolkits that allow a layer-wise unsuperwised pretraining of DNNs (see [6] and [7]).
   - Perform some experiments with standard classification tasks. Check also the possibility of the toolkits for distributed computing.
   - Collect pros and cons for all toolkits.
   - Discuss this collection with your supervisors and decide which toolkit to used for the main part of this thesis.

C. Investigation of DNNs with synthetic data

   - Define a set of classification tasks (from simple to ASR-like) and discuss it with your supervisors.
   - Generate synthetic data for each one of these classification tasks.

- Train, test and evaluate different DNN configurations and training methods. These experiments should provide answers to the questions of overfitting and of the required size of the training data set (mainly for complex data that are similar to speech features).
- Compare the performance of the investigated DNNs with standard MLPs.
- Document your achievements and conclusions.

D. Application of DNNs in speech recognition (ASR)

- Study some references about speech recognition with DNNs (e.g. [1]).
- Based on the results of part C, define a phoneme recognizer architecture (cf. introduction above).
- Setup an ASR test environment: toolkit and database (it is recommeded to use the TIMIT speech corpus [8]).
- Perform phoneme recognition experiments with suitable DNNs and MLPs and evaluate the results.

The work done and the attained results have to be documented in a report (see recommendations [9]) that has to be handed in as PDF document. Furthermore, two presentations have to be given: the first one will take place about two weeks after the start of the work and is meant to give a short overview of the task and the initial planning. The second one at the end of the project is expected to present the task, the work done and the achieved results in a sufficiently detailed way. The dates of the presentations will be announced later.

## References

[1] G. Hinton, L. Deng, and D. Yu, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[2] N. Srivastava and G. Hinton, et al. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[3] B. Pfister und T. Kaufmann. *Sprachverarbeitung: Grundlagen und Methoden der Sprachsynthese und Spracherkennung.* Springer Verlag (ISBN: 978-3-540-75909-6), 2008.

[4] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[5] G. Dahl, T. Sainath, and G. Hinton. Improving deep neural networks for LVCSR using rectified linear units and dropout. In *ICASSP*, pages 8609–8613, 2013.

[6] G. Hinton and S. Osindero. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[7] G. Hinton. A Practical Guide to Training Restricted Boltzmann Machines. Technical report, Department of Computer Science, University of Toronto, 2010.

[8] J. Garofolo, et al. TIMIT Acoustic-Phonetic Continuous Speech Corpus LDC93S1. Linguistic Data Consortium, Philadelphia, 1993. (web download).

[9] B. Pfister. *Richtlinien für das Verfassen des Berichtes zu einer Semester- oder Master-Arbeit.* Institut TIK, ETH Zürich, Februar 2013. (http://www.tik.ee.ethz.ch/spr/sada/richtlinien_bericht.pdf).

[10] B. Pfister. *Hinweise für die Präsentation der Semester- oder Master-Arbeit.* Institut TIK, ETH Zürich, Februar 2013. (http://www.tik.ee.ethz.ch/spr/sada/hinweise_praesentation.pdf).

February 5, 2015