Karlsruhe Institute of Technology

An Optimization of Deep Neural Networks in ASR using Singular Value Decomposition

Bachelor Thesis
of

Igor Tseyzer

At the Department of Informatics
Institute for Anthropomatics (IFA)

Reviewer:	Prof. Dr. Alexander Waibel
Second reviewer:	Dr. Sebastian Stüker
Advisor:	Dipl.-Inform. Kevin Kilgour

Duration: January 26, 2014 – May 26, 2014

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, May 26, 2014 Igor Tseyzer

Contents

1. Introduction

Automatic Speech Recognition (ASR) is a powerful tool which opens a new level of man-machine interaction. ASR can be applied in various areas of everyday life, such as dictation, hands-free control of devices and machine translation. In recent decades the accuracy and performance of ASR systems has significantly improved. One of approaches which insured this improvement is Artificial Neural Networks (ANN) and particularly Deep Neural Networks (DNN). The DNNs allow extraction of significant information from acoustic input and in this way increase the accuracy of recognition. Until recent, training a DNN was not possible. A significant improvement in the field of deep learning was achieved thanks to the development of computing power and the introduction of deep learning methods. Therefore the recent decade has seen significant advance in the research of DNNs for ASR. Recent studies in this field have shown a big improvement in speech recognition accuracy.

An example of using a DNNs in practice is a lecture translation systems. These systems make it possible for students to listen to lectures in a language which is not their native one. Additionally this allows having a recording and transcription of lectures in both languages; in language of the lecturer and the student's native language. These kind of systems must fulfill strict requirements, such as quick performance and high accuracy in recognition. In order to make an ASR system fit these strict requirements, the question of optimizing the DNN is uppermost. The main problem is finding a balance between two properties of DNNs, which are present on two different poles: performance and accuracy. Improving one of them, in general, makes the other worse. To solve this problem various optimization techniques can be used. Today's studies are directed towards optimizing the topology, weights initialization and performance of DNNs.

Solving these problems and as a result optimizing the DNN would allow creation of better ASR systems. This can help to make communication between people in different counties easier and break the language barrier in education, tourism and business areas.

This work aims first of all to find the optimal topology for DNN and optimize this topology by using singular value decomposition. The major attempts will be

researched and evaluated. The main goal of this work is to develop a simple and effective method of optimizing DNN by using singular value decomposition.

In chapter 2, the background and related work is presented. Chapter 3 analyzes the modern approaches in optimizing DNN. Design of proposed models and their implementation is described in chapter 4. Finally, chapter 5 evaluates the proposed models.

2. Background and Related Work

In this chapter the background of ASR and related work in the field of optimizing DNN is presented. This would make the process of speech recognition understandable and describe recent achievement in optimizing of DNN.

2.1 Basics of Automatic Speech Recognition

In essence speech recognition can be seen as a transformation of human speech into a form which can be understand by machines. From a mathematical point of view the ASR process is represented as a classification task. Most of the systems employ probabilistic classifiers to find the most likely word sequence to a given acoustic signal. In order to perform this classification, two probabilities are estimated: probabilities of possible word sequences and probability distributions of the acoustic signals for each word sequence. Both probabilities are represented as two parametric models.

2.1.1 ASR as Pattern Recognition Process

Mathematically, parts of the process of speech recognition can be presented as Pattern, Classes and Classification Output [VSR12]. Where Pattern is an audio recording X, Classes are word sequences W from space $\mathcal{W}$ of all word sequences, and Classification Output is a word sequence W for recording X. For the given speech recording X, the sequence of word $\hat{W} = \hat{w}_1, \hat{w}_2, \ldots, \hat{w}_n$ can be estimated as follows [ST95]:

$$\hat{W} = \underset{W}{\operatorname{argmax}} P(W|X). \tag{2.1}$$

The equation 2.1 can be re-factored using Bayes' rule as follows:

$$\hat{W} = \underset{W}{\operatorname{argmax}} P(X|W) \times P(W) \tag{2.2}$$

The equation 2.2 can be also represented as:

$$P(W \mid X) = \frac{P(X \mid W) \times P(W)}{P(X)} \tag{2.3}$$

This equation is also known as fundamental equation of speech recognition. The parts of this equation are presented as acoustic model ($P(X|W)$) and language model ($P(W)$) [ST95].

Figure 2.1 presents the basic scheme of statistical ASR system.

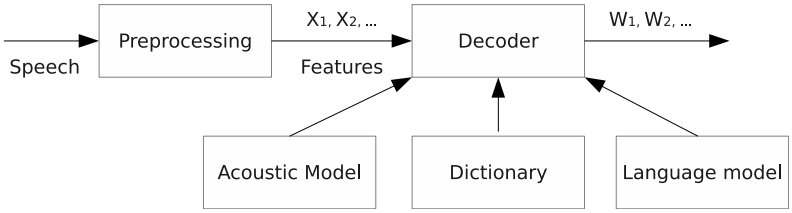

Figure 2.1: Scheme of statistical ASR system. Based on [ST95].

2.1.2 Extraction of Speech Features

The first step in the recognition process is the transformation of an acoustic input into a sequence of feature vectors. This transformation allows the representation of speech waveform with a relatively small number of dimensions. This process of transformation is called speech feature extraction.

The speech processing is divided into two major methods, non-parametric methods based on periodograms, and parametric methods, which use a small number of parameters from data [VSR12]. An overview of the methods is available in table 2.1.

Spectrum	Method	Properties		
		Frequency axis	Frequency resolution	Pitch sensitivity
PS	Nonparametric	Linear	Static	Very high
Warped PS	Nonparametric	Nonlinear	Static	Very high
Mel-filter bank	Nonparametric	Nonlinear	Static	High
LP	Parametric	Linear	Static	Medium
Perceptual LP	Parametric	Nonlinear	Static	Medium
Warped LP	Parametric	Nonlinear	Static	Medium
Warped-twice LP	Parametric	Nonlinear	Adaptive	Medium
MVDR	Parametric	Linear	Static	Low
Warped MVDR	Parametric	Nonlinear	Static	Low
Warped-twice MVDR	Parametric	Nonlinear	Adaptive	Low

PS = power spectrum, LP = linear prediction; MVDR = minimum variance distortionless response.

Table 2.1: Overview of spectral estimation methods [VSR12]

.

The process of extracting features starts with taking a frame at the start of the acoustic input, applying a windowing function and shifting the frame further over the acoustic input. The size of the frame should be chosen sensibly, so it would be short enough to provide the required time resolution and long enough to ensure

adequate frequency resolution. Another important parameter is the frame shift over the acoustic input. Together frame size and frame shift can have a big influence on speech recognition accuracy and their choice depends on the characteristics of the speech. In general, a sensible choice of frame size and shift is 16-32 ms for the frame size and 5-15 ms for the frame shift [VSR12].

The next step is transforming of the chosen frame with Short-Time Fourier Transform (STFT):

$$X(\tau, \omega) = \int_{-\infty}^{\infty} x(t)w(t-\tau)e^{-j\omega t}dt, \tag{2.4}$$

where $w(t-\tau)$ is window function, $x(t)$ is acoustic input at time t, τ is time index and ω is frequency. Various functions could be used as a window function, for example, Rectangular, Parzen, Hanning, Blackman, Kaiser, and Hamming windows. In speech recognition the Hamming window is commonly used. The Hamming window is defined as:

$$w(n) = \begin{cases} 0.54 - 0.46\cos(\frac{2\pi n}{N_w}), & \forall 0 \leq n \leq N_w, \\ 0, & \text{otherwise}, \end{cases} \tag{2.5}$$

Further processing includes calculating Cepstral Coefficients. The cepstrum is a result of the Fourier transform of the (warped) logarithmic spectrum. The real cepstrum uses a logarithmic function and is defined as follows [VSR12]:

$$c_x(k) = \frac{1}{2\pi}\int_{-\pi}^{\pi} \log|X(\mathrm{e}^{j\omega})|\mathrm{e}^{j\omega k}d\omega, \tag{2.6}$$

where $X(\mathrm{e}^{j\omega})$ is a spectral estimate, for example mel scale.

The speech features could be generated as

$$\hat{x}_{min}(k) = \begin{cases} 0, & \forall k < 0, \\ c_x(0), & k = 0, \\ 2c_x(k) & \forall k > 0 \end{cases} \tag{2.7}$$

Another way to generate features is Type 2 Discrete Cosine Transform (DCT) [VSR12].

$$\hat{x}_{min}(k) = \sum_{m=0}^{M-1} \log|X(\mathrm{e}^{j\omega_m})|T^{(2)}_{k,m}, \tag{2.8}$$

where $\log|X(\mathrm{e}^{j\omega_m})|$ is a log-power density vector and $T^{(2)}_{k,m}$ are components of the Type 2 DCT.

The general process of generating cepstral coefficients is represented in figure 2.2

2.1.3 Acoustic Models

As mentioned above, the Acoustic Model (AM) can be represented as part $P(X \mid W)$ in equation 2.3. After successful extraction of speech features, a classification method is required to classify extracted features to word sequences. A commonly used method in recent time is Hidden Markov Model (HMM).

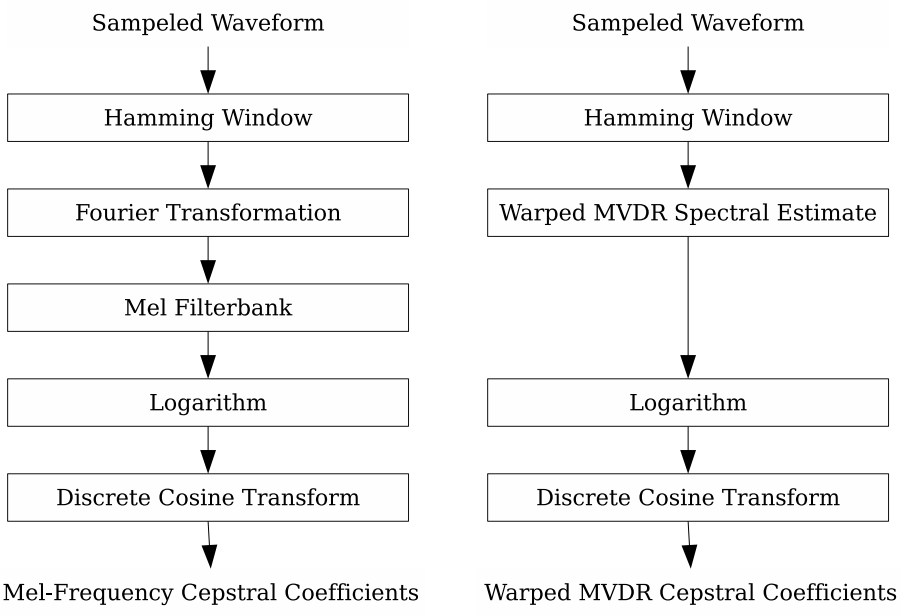

Figure 2.2: Block diagrams illustrating all the processing steps required for the calculation of the compared mel-frequency cepstral coefficient and warped MVDR cepstral coefficients front-ends [VSR12]

The HMM can be described as probabilistic function of the Markov Chain [ST95]. The HMM is divided into two levels, a finite set $Q = \{s_1, \ldots, s_N\}$ of states and output alphabet $K = \{v_1, \ldots, v_K\}$. The first level is a Markov chain, with specified initial state. The visible states of process are represented as a sequence of random variable $q = q_1, ..., q_T$, which take value from Q. The probability of moving from one state to another is described as follows [ST95].:

$$P(q_i \,|\, q_1 \ldots q_{t-1}) = P(q_i \,|\, q_{t-1}), \tag{2.9}$$

and can be represented as $N \times N$ transition matrix A:

$$A = [a_{ij}]_{N \times N} \;\; with \;\; a_{ij} = P(q_t = s_j \,|\, q_{t-1} = s_i). \tag{2.10}$$

The probability of process finding in certain state is described as probability π and transition matrix A:

$$\pi_i = P(q_1 = s_i), \;\; \sum_{i=1}^{N} \pi = 1 \tag{2.11}$$

In general the Markov Chain represents the manner in which the process moves through states.

The second level is a set of state output probability distributions for each state. When the process comes to state i at time t, an observation symbols $O = O_1 \ldots O_T$ are generated based on state output distribution [ST95]:

$$P(O_t \,|\, O_1 \ldots O_{t-1},\, q_i \ldots q_t) = P(O_t \,|\, q_t), \tag{2.12}$$

The output symbols probability distribution for state j on the second level can be found using this equation and represented as matrix B:

$$b_{jk} = P(O_t = v_k \,|\, q_t = s_j) \;\; and \;\; B = [b_{jk}]_{N \times N} \tag{2.13}$$

The Hidden Markov Model from this point of view is described as:

$$\lambda = \pi, A, B \tag{2.14}$$

After describing the HMM, the best parameter λ should be found, which would represent the set of given symbols $O = O_1 \dots O_T$. This training procedure can be done using maximum-likelihood estimation (MLE), Baum–Welch algorithm and Expectation–maximization algorithm (EM) [ST95].

The output of HMM is modeled as a probability function. In speech recognition, a state output distribution $P(O_t \mid q_t)$ is usually modeled as Gaussian Mixture densities [VSR12]:

$$P(O_t \mid q_t) = \sum_{k=1}^{K} \omega_{i,k} \mathcal{N}(x; \mu_{i,k}, \Theta_{i,k}), \tag{2.15}$$

where $\mathcal{N}(x; \mu, \Theta)$ is multivariate Gaussian density. The $\omega_{i,k}, \mu_{i,k}, \Theta_{i,k}$ are mixture weight, mean vector, and covariance matrix of the kth Gaussian in the mixture Gaussian state output distribution for state i. K is the number of Gaussians in the mixture.

As can be seen, the AM allows the representation of extracted speech features as output symbols. This output should be formed as a word sequence using a Language Model, which is described in the next section.

2.1.4 Language Models

The next step in the speech recognition process is to build hypotheses of word sequences according to given output symbols probability distribution from AM. Generating of the hypotheses can be done using statistical language models. This modeling process is represented as part $P(W)$ in equation 2.3 [ST95].

The first step in building a statistical language model is to define equivalence classes for words $Q = \{s_1, \dots, s_N\}$. This allows the representation of a sequence of words $w_1, w_2, \dots, w_m$ as a state of deterministic finite state machine [ST95]:

$$\begin{aligned} q_0 &= s[\emptyset] \in Q \\ q_1 &= s[w_1] \in Q \\ q_2 &= s[w_1 w_2] \in Q \\ &\vdots \\ q_m &= s[w_1 ... w_m] \in Q \end{aligned}$$

By using this model, the probability of a sentence can be represented as follows:

$$P(w) = \prod_{i=1}^{m} \Big(\sum_{n=1}^{N} P(w_i \mid q_{i-1} = s_n) \times P(s[w_1 \dots w_{i-1} = s_n]) \Big) \tag{2.16}$$

As can clearly be seen in the equation above, the probability of one particular word depends on the sequence of words which appeared before it. The models which can compute this dependency are called n-gramm [ST95]. The most useful for speech

recognition are unigram, bigram and trigram. The computation of probability with unigram, bigram and trigram is described in following equations:

$$P(w) = \prod_{i=1}^{m} P(w_i), \tag{2.17}$$

$$P(w) = P(w_1) \times \prod_{i=2}^{m} P(w_i \,|\, w_{i-1}), \tag{2.18}$$

$$P(w) = P(w_1) \times P(w_2 \,|\, w_1) \times \prod_{i=3}^{m} P(w_i \,|\, w_{i-2} w_{i-1}). \tag{2.19}$$

However, the amount of parameters in the n-gram could be very big. For example, the trigram model with 1000 words contains about 10^9 parameters [ST95]. In order to avoid this, the words are classified to disjunct categories:

$$\mathcal{C} = \{C_1, \ldots, C_N\} \;\; with \;\; \cup_{k=1}^{N} C_k = \mathcal{W} \tag{2.20}$$

A word can be ordered to a particular category $C(w) \in \mathcal{C}$ and $w \in C(w)$. So for example, the probability for bigram can be represented as follows[ST95]:

$$P(w) = P(w_1) \times \prod_{i=2}^{m} P(w_i \,|\, C(w_i)) \times P(C(w_i)|C(w_{i-1})) \tag{2.21}$$

Other models apart from n-gramms can be used, for example:

- multilevel n-gram language models
- morpheme-based language models
- context-free grammar language models

Using language models together with acoustic models and a dictionary enables the production of the most likely hypotheses for a given acoustic input. The existing methods of building LM and AM are continuously improved and extended, which will ensure the further improvement of speech recognition accuracy and performance.

2.2 Artificial Neural Networks in ASR

Artificial Neural Networks (ANNs) started to be commonly used in ASR at the end of the 1980s. The main idea was to replace Gaussian Mixture Model (GMM) with ANN. The ANNs are very good at the task of classification, so they could be successfully applied to transform acoustic input into the states of HMM. There are various approaches that use hybrid ANN/HMM in the task of ASR. They are based on ANN's architectural and algorithmic solutions. According to Trentin et al. [TG01], ANN/HMM hybrid systems can be divided into five major categories:

Early attempts

Early approaches (between the late 1980s and the beginning of the 1990s) relied on ANN architectures that attempted to emulate HMMs.

ANNs to estimate the HMM state-posterior probabilities
Some ANN/HMM hybrids assume that the output of an ANN is sent to an HMM.

Global optimization
Introduction of a training scheme aimed at the optimization of a global criterion function, defined at the whole-system (i.e., ANN and HMM simultaneously) level.

Networks as vector quantizers for discrete HMM
Unsupervised ANNs are used to perform a quantization in the acoustic feature space for discrete HMMs.

Other approaches
Hybrid systems based on particular combination techniques between ANNs and HMMs, not belonging to any of the previous categories, and often focused on specific tasks.

According to [HDY+12], the main reason for using ANN was the statistical inefficiency of GMM for modeling a non-linear manifold in a data space, and the ability of ANN to learn model of data in non-linear manifold. However in the early stages of ANN, learning algorithms and hardware were not able to provide efficiency as good as GMM. Since then, development of hardware and new deep learning algorithms have made it possible to train big ANNs with many hidden layers and big output layers to estimate the emission probabilities for HMM [HDY+12], which was not possible before.

Hinton presented the first method of deep learning that makes it possible in 2007 [Hin07]. The main idea of his work is to pre-train each layer of multi-layer neural network as an unsupervised Restricted Boltzmann Machine (RBM) and to fine-tune using supervised backpropagation.

The ability of deep learning provided a further development of architectures and training methods of DNN. Since 2007 DNN has played an increasingly important role in ASR.

2.3 Related Work

Optimizing the DNN is one of a key task in ASR. As shown by Dahl et al.[DYDA12], using DNN instead of GMMs significantly improves the performance of speech recognition systems. But the DNNs still have problems; various pre-training, training and DNN typologies have been researched in order to solve them. This section presents recent achievements in this field.

2.3.1 Deep Belief Networks

One of the ways to optimize DNNs is the sensible initializing of weights before training. In 2012 Mohamed et al. presented their work “Acoustic Modelling Using Deep Belief Networks” [MDH12]. According to their work training a multilayer network could be represented from two points of view; directed and undirected. In the undirected view, the model contains separate layers which are not connected to

each other, but the activity of one layer can be given as input to the next layers. A simple example of the undirected view is the Restricted Boltzman Machine (RBM). RBM contains visible and hidden units. Visible units are connected to the input, and hidden units help visible units to represent learned features. Restricted means that there is no connection between visible-visible or hidden-hidden units.

In the directed point of view, hidden layers represent binary features and the visible layer represents a data vector. The Deep Belief Network (DBN) has top-down connections, so the probability of generating date on visible units is bigger than on others. The difference between RBM and DBN can be seen on figure 2.3.

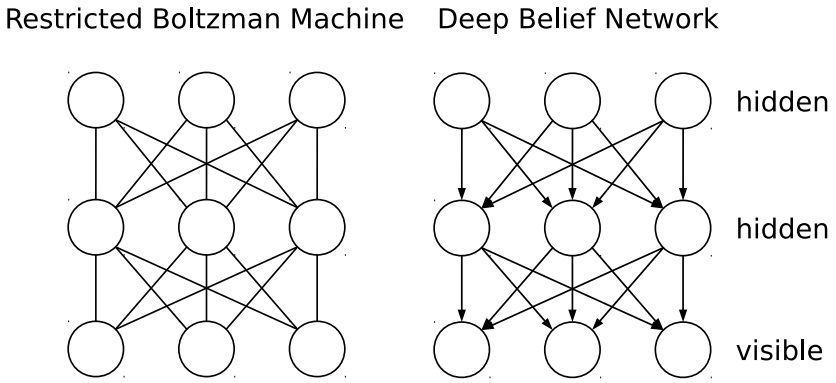

Figure 2.3: A difference between RBM and DBN

Training the DBN proceeds in two stages. First comes layer-wise unsupervised pre-training, which contains the step of learning with tied weights. Second comes learning the different weights in each layer, and then the final layer with softmax activation function is added to the top of the network. The pre-training is followed by discriminative fine-tuning using backpropagation to maximize the log probability of the correct HMM state.

The experiments of Mohamed et al. were performed on the TIMIT corpus [MDH12]. Their experiments showed that the Phone Error Rate (PER) of the best DBN is 20.7%, which is 24.17% less than the PER of Context-Dependent HMM system (27.3%).

Mohamed et al. concluded the following about factors which influence DNN performance [MDH12]:

1. Better recognition performance can be achieved by using 40 filter-bank coefficients.
2. Adding hidden layers improves performance, if the number of layers is less than 4. Further addition does not give much of a bigger improvement.
3. More small layers are better than one big layer.
4. Pre-training is necessary when the network has more than 1 hidden layers.
5. Pre-training prevents overfitting of network.
6. The best number of frames in the input window is 11, 17, or 27 frames.

7. Adding a last bottle-neck hidden layer helps to prevent overfitting of the network by reducing the number of not pre-trained weights.

2.3.2 Stacked Autoencoders

Initializing of the weights can be done in various ways. As described above, Hinton et al. [HOT06] achieved great success in deep learning with DBN. At the same time, Bengio et al. introduced another approach [BLPL06]. They used the same layer-by-layer initialization using auto-encoders. In 2008 Vincent et al. introduced their work in which they tried to extract and compose Robust Features with Denoising Autoencoders [VLBM08]. Their research was based on the hypothesis that partially destroyed input should have almost the same representation as not destroyed input.

The basic auto-encoder can be described as follows [VLBM08]. First the autoencoder receives input vector $x \in [0,1]^d$ and, using the function

$$y = f_\theta(x) = s(Wx + b), \tag{2.22}$$

maps it to hidden representation $y \in [0,1]^{d'}$. The function f_θ is parametrised by $\theta = \{W, b\}$, where W is weight matrix $d' \times d$ and b is a bias vector. The hidden representation y is then mapped back to vector $z \in [0,1]^d$ with function

$$z = g_{\theta'}(y) = s(W'y + b'), \tag{2.23}$$

where $\theta' = \{W', b'\}$. The matrix W' is constrained by $W' = W^T$. In this case the autoencoder has tied weights. The parameters are optimized to minimize average reconstruction error θ^*,

$$\theta'^* = argmin \frac{1}{n} \sum_{i=1}^{n} L(x^{(i)}, z^{(i)}), \tag{2.24}$$

where L is a loss function. The reconstruction cross-entropy can be presented as follows:

$$L_{\mathbf{H}}(x, z) = \mathbf{H}(\mathcal{B}_x \parallel \mathcal{B}_z) \tag{2.25}$$

Second, to construct the denoising autoencoder, a modification of the basic encoder is needed. It can be represented as the application of stochistic process $q_D(\tilde{x} \mid x)$ to input vector x. It means that the encoder should be partly destroyed. In order to do this, the fixed number of components vd are chosen and their value set to 0. In this way the initial input x is mapped to partly destroyed version $\tilde{x}$. Then the partly destroyed input $\tilde{x}$ is used to a receive hidden representation:

$$y = f_\theta(\tilde{x}) = s(W\tilde{x} + b) \tag{2.26}$$

Then follows the reconstruction:

$$z = g_{\theta'}(y) = s(W'y + b'). \tag{2.27}$$

Now z is a deterministic function of $\tilde{x}$, which represents a stochastic mapping of x. This procedure can be seen on figure 2.4.

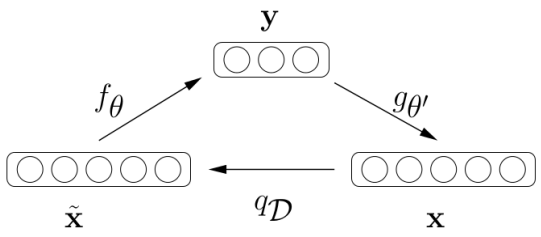

Figure 2.4: An example x is corrupted to $\widetilde{x}$. The autoencoder then maps it to y and attempts to reconstruct x[VLBM08].

As shown by [BLPL06], greedy layer-wise training can be applied to initialize weights with the autoencoder, where the k layer is used as input for $k+1$ layer. The same procedure can be used for denoising encoder [VLBM08]. The denoising autoencoder is trained by standard backpropogation to minimize loss function $L = (x, z)$. Vincent et al. concluded initialization of layers using denoising encoders helps to capture interesting structures in the input distribution [VLBM08].

A practical application of stacked denosing encoders was done by Gehring et al. [GMMW13]. In their work they used stacked autoencoders to extract bottleneck features for ASR tasks. They used the method proposed by Vincent et al. [VLBM08].

The experiments were performed on IARPA Babel Program Cantonese language collection babel101-v0.1d, Cantonese language collection babel101-v0.4c with 80 hours of actual speech and the Tagalog language collection [GMMW13].

One of the experiments, the evaluation on babel101-v0.4c, showed significant improvement of the autoencoder based network compared to the network that used MFCC. The results can be seen in table 2.2

System	Model	DBNF (by AE Layers)					
		1	2	3	4	5	6
CER (%)	66.4	63.6	62.0	60.9	60.3	60.5	61.1

Table 2.2: Character error rates of the Cantonese babel101-0.4c system with MFCC and DBNF input features[GMMW13]

The denoising autoencoders can be used for the initialization of DNN and show better results than MFCC based systems.

2.3.3 Deep Stacking Network

Another approach in initializing weights is the Deep Stacking Network (DSN), which was presented by Deng et al. in 2012 [DYP12]. The main goal of this network is to provide a method for building deep classification architectures which can also be parallellized. The research was based on the idea of composing simple functions into a chain and then stacking them together. All functions estimate the same target. This method helps to avoid overfitting due to learning complex functions.

The structure of DSNs can be seen in figure 2.5. The first layer is a linear layer. It corresponds to raw input data. Second is the non-linear sets of sigmoidal hidden

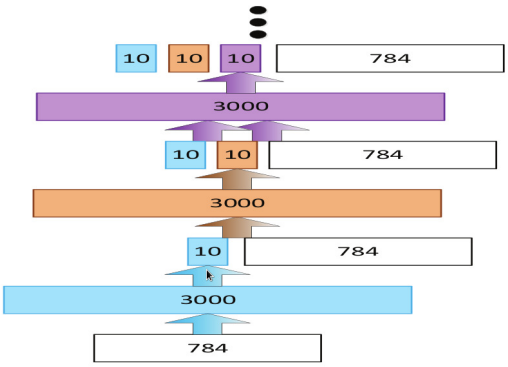

Figure 2.5: Illustration of the basic architecture of DSN [DYP12]

units. Third, the output layer is the set of C-linear output units. The experiments were done on MNIST image classification and on TIMIT corpus [DYP12]. The results of speech classification experiments show that the DSN achieved frame-level error rate of 43.86%, which is 1.18% better than a DNN tested on the same dataset (45.04%).

2.3.4 Rectified Linear Units and Dropout

An important part of optimizing a DNN is the activation function of neurons. Choosing a sensible activation function for a DNN can not only improve performance, but also prevent quick system overfitting by deep learning methods. In 2013 Dahl et al. presented an improvement of DNN that used Rectified Linear Units (ReLU) and Dropout [DSH13]. Rectified Linear Units use the simple activation function $max(0, x)$, meaning they can easily be used in DNN or RBM. The problem of ReLU is that they can more easily overfit, than the sigmoid activation function. In order to prevent this problem, Dahl et al. [DSH13] combined ReLU with Dropout, which avoid overfitting by adding noise zeros or "dropout" to hidden units. The use of dropout can be presented as follows:

$$y_t = f(\frac{1}{1-r} y_{t-1} * mW + b), \tag{2.28}$$

where f is the activation function of $t-1$ layer, W and b are weights and bias for current layer, $*$ is element-wise multiplication. The input from the previous layer would be computed with factor $\frac{1}{1-r}$, where r is the dropout probability for units in the previous layer. The ability to avoid overfitting allows training of big DNN, but using dropout slows learning time by a factor of about two.

Dahl et al. performed their experiments on 50 hours of English Broadcast News. The DARPA EARS rt03 set was used for development/validation and final testing was performed on the DARPA EARS dev04f evaluation set [DSH13]. The comparison of results with full sequence training can be seen in table 2.3

They show that combined ReLU and dropout work well. The dropout can also be used in combination with other activation functions and might improve their performance.

Model	rt03	dev04f
GMM baseline	10.8	18.8
ReLUs, 3k, dropout, sMBR	9.6	16.1
Sigmoids, 2k, sMBR	9.6	16.8

Table 2.3: Results with full sequence training [DSH13]

2.3.5 Context-Dependent Deep-Neural-Network

The second way to optimize a DNN is to use a larger number of output units. The first ANN used in ASR mostly used context-independent (CI) HMM. Their results were comparable or better than GMM. The logical development of DNN was to apply context-dependent (CD) HMM as output for DNN. In 2011 Seide et al. proposed Context-Dependent Deep Neural Network HMMs, or CD-DNN-HMM [SLY11]. The experiments were performed on the task of speech-to-text transcription using the 309-hour Switchboard-I training set. A GMM-HMM system with 40 Gaussian mixtures trained with maximum likelihood (ML) was used as a baseline. The training of the DNN was performed in two steps. First, the DNN was pre-trained with DBN; second, it was fine-tuned with back-propagation. The experiments showed significant improvement in word-error rate (WER) in about 33% of CD-DNN-HMM systems (18.5%) compared to a GMM 40-mix, BMMI system (27.4%) tested on an RT03S task [SLY11].

Extracting Bottle-Neck Features (BNF) can be also done using DNN. In 2013 Gehring et al. [GMMW13] performed an extraction of deep BNF using stacked auto-encoders. The description, results and evaluation of the experiments performed by Gehring et al. can be found in section 2.3.2.

2.3.6 Deep Convolutional Neural Networks

As described above, DNN achieved great success in the task of ASR. Another type of ANN can be used apart from DNN: Convolutional Neural Networks (CNNs). Unlike DNN, whose units are fully connected, CNN computes only a small local input for each unit. In 2013 Sainath et al. introduced an application of deep CNN in the ASR [SMKR13]. They tried to find an optimal architecture for CNN. Figure 2.6 shows a typical CNN architecture.

The computation of hidden activation proceeds in three steps. First, the weights W are multiplied with local input $v1, v2, v3$. Second, the weights W are shared across the entire input space, and third, max-pooling is used to remove variability in the hidden units.

Sainath et al. first trained the CNN on a 50-hour English Broadcast News task and tested it on EARS dev04f set [SMKR13]. 40 dimensional log mel-filterbank coefficients were used as acoustic input. The performance of the CNN was compared with a fully connected DNN with a hidden layer size of 1024 units and output layer of 512 units. Table 2.4 shows the results of comparing DNN with CNN.

Other test results showed that the best number of hidden units for CNN is 128 for the first convolutional layer and 256 for the second layer. Table 2.5 shows results for the architecture used. The CNN hybrid is 3-5% better relative to the DNN hybrid.

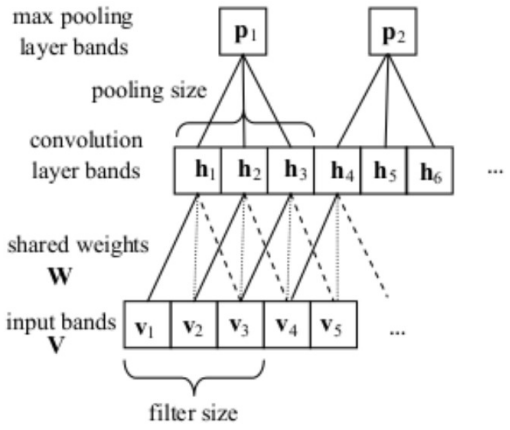

Figure 2.6: Diagram showing a typical convolutional network architecture consisting of a convolutional and max-pooling layer. In this diagram, weights with the same line style are shared across all convolutional layer bands [SMKR13]

Number of Convolutional vs. Fully Connected Layers	WER
No conv, 6 full (DNN)	24.8
1 conv, 5 full	23.5
2 conv, 4 full	22.1
3 conv, 3 full	22.4

Table 2.4: WER as a Function of number of Convolutional Layers [SMKR13]

Model	dev04f	rt04
Baseline GMM/HMM	18.8	18.1
Hybrid DNN	16.3	15.8
DNN-based Features	16.7	16.0
Hybrid CNN	15.8	15.0
CNN-based Features	15.2	15.0

Table 2.5: WER for NN Hybrid and Feature-Based Systems [SMKR13]

The further experiments were performed on a large task: on 400 hours of English Broadcast News and 300 hours of conversational American English telephony data from the Switchboard corpus. DARPA EARS dev04f set was used for development and DARPA EARS rt04 for evaluation. In the next experiment Hub5'00 set was used for development and rt03 set for evaluation, with separate testing of Switchboard (SWB) and Fisher (FSH) portions of the set [SMKR13]. The results can be seen in tables 2.6 and 2.7.

Model	dev04f	rt04
Baseline GMM/HMM	16.0	13.8
Hybrid DNN	15.1	13.4
DNN-based Features	14.9	13.3
CNN-based Features	13.1	12.0

Table 2.6: WER on Broadcast News, 400 hrs [SMKR13]

Model	dev04f	rt03	
	SWB	FSH	SWB
Baseline GMM/HMM	14.5	17.0	25.2
Hybrid DNN	12.2	14.9	23.5
CNN-based Features	11.5	14.3	21.9

Table 2.7: WER on Switchboard, 300 hrs [SMKR13]

As can be seen from all experiments, both CNN and DNN perform better than GMM/HMM systems. For the Broadcast News test, the CNN showed a 10-12% relative improvement over the DNN-based features. For conversational American English they are between 4-7% better than the hybrid DNN.

2.3.7 Deep Recurrent Neural Networks

Recurrent neural networks (RNN) are a powerful tool due to their ability to remember and produce sequences of actions from one input. RNNs achieved great success in handwriting recognition. Graves et al. introduced the Long Short-Term Memory (LSTM) RNNs in ASR [GMH13]. They also developed an end-to-end method which included a joint train of two RNNs, for the acoustic and the language model. The LSTM RNN is based on purpose-built memory cells which include an input gate, a forget gate, an output gate and cell activation vector (see figure 2.7).

The main power of RNN is its ability to use previous context from a network, but RNN can in addition use future context. Such an RNN is called a Bidirectional RNN (BRNN) (see figure 2.8). The BRNN computes the forward hidden sequence $\overrightarrow{h}$, the backward hidden sequence $\overleftarrow{h}$ and the output sequence y. The combination of BRNN and LSTM gives Bidirectional LSTM; the deep RNN can then be created by stacking RNN layers one on the top of the other. This Bidirectional LSTM architecture was applied to the TIMIT corpus.

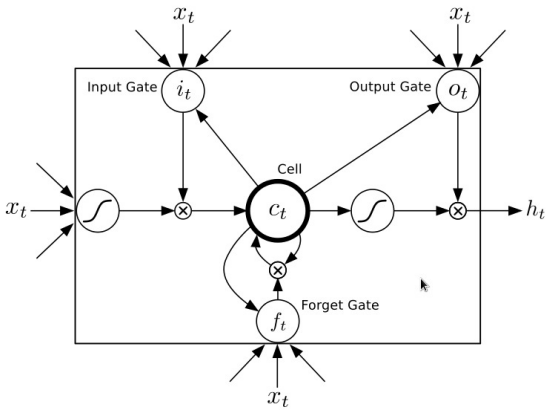

Figure 2.7: Long Short-term Memory cell [GMH13]

The Bidirectional LSTM was trained on a 40-coefficient filter-bank with three training method: Connectionist Temporal Classification (CTC), Transducer and pre-trained Transducer [GMH13]. The best system showed a Phone Error Rate (PER) of 17.7%.

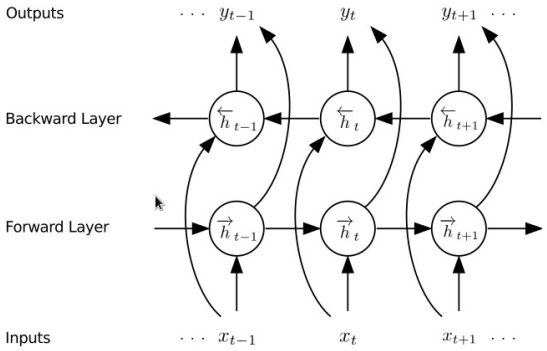

Figure 2.8: Bidirectional RNN [GMH13]

2.3.8 Restructuring of DNN with Singular Value Decompositions

Training a DNN is a process that requires high computation costs. As shown above, DNN perform better if their size is increased. In 2013 Xue et al. introduced the method of compressing a DNN without losing performance [XLG13]. Their work is based on applying singular value decomposition (SVD) to decompose weight matrices of a DNN and then restructuring the model in order to keep the similarity to the original model. To maintain the performance of the restructured model, fine-tuning is applied.

Figure 2.9 shows a typical DNN in ASR with fully connected layers. In order to perform decomposition, the SVD can be applied to weight matrices between layers.

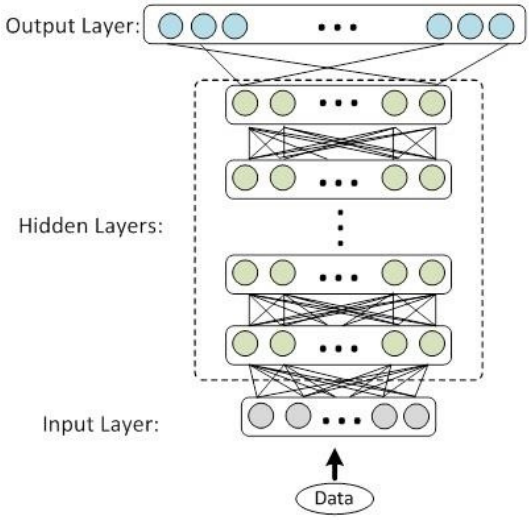

Figure 2.9: DNN used in ASR systems [XLG13]

Applying SVD to the weight matrix A with size $m \times n$ gives the following result:

$$A_{m\times n} = U_{m\times n} \sum_{n\times n} V^T_{n\times n} \tag{2.29}$$

According to Xue et al. [XLG13], about 15% of singular values contribute 50% of total values and about 40% - 80% of total size. It means that the number of elements in matrix A can be reduced without losing important information. After leaving only k-biggest singular values on A, the final matrix would look as follows:

$$A_{m\times n} = U_{m\times k} \sum_{k\times k} V^T_{k\times n} \tag{2.30}$$

The $A_{m\times n}$ can also be presented as

$$A_{m\times n} = U_{m\times k} W_{k\times n}, \tag{2.31}$$

where $W_{k\times n} = \sum_{k\times k} V^T_{k\times n}$.

After applying SVD, two smaller matrices U and W can be applied back to the original model to replace A. The single layer in the original model would be replaced by two layers. The number of parameters is changed from $m \times n$ to $(m + n) \times k$. The replacement of the single layer can be seen in figures 2.10 and 2.11.

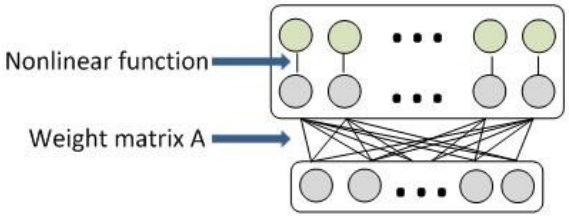

Figure 2.10: One layer in original DNN model [XLG13]

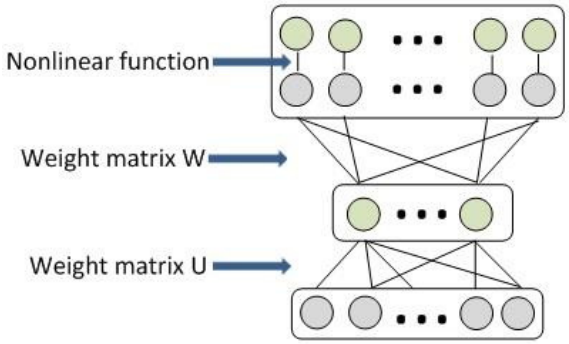

Figure 2.11: Two corresponding layers in new DNN model [XLG13]

Xue et al. [XLG13] applied this procedure to a Microsoft internal task (task1), which consisted of 750 hours of audio. As input, the CD-DNN-HMM system with 13-dimension mean-normalized MFCC feature was used. The system layout is as follows: input layer with 572 units, 4 hidden layers with 2048 units each and output layer with 5976 units. The number of parameters for system: $572 \times 2048 + (2048 \times 2048) \times 4 + 2048 \times 5976 \approx 29M$. Table 2.8 shows the results of applying SVD on whole system.

Xue et al. were able achieve reduction of the DNN model on task 1 by about 80%. By keeping singular values proportional to the total amount of values a reduction of 73% could be reached, with less than 1% accuracy loss.

Acoustic model		WER	Number of parameters
All hidden layers (512)	Before fine-tune	26.0%	21M
	After fine-tune	25.6%	
All hidden layers (256)	Before fine-tune	27.0%	17M
	After fine-tune	25.8%	
All hidden layers and output layers (256)	Before fine-tune	29.7%	7M
	After fine-tune	25.4%	
All hidden layers and output layers (192)	Before fine-tune	36.7%	5.6M
	After fine-tune	25.5%	

Table 2.8: Results of SVD restructuring on the whole model on task 1 [XLG13]

2.3.9 Bottle-neck Features

The DNN can also be used to extract speech features. As described earlier, extracting probabilistic features is a very important part of ASR, and one where various approaches can be applied. One of the most efficient is bottle-neck features (BNF). The use of BNF was first proposed by Grézl et al. [GKKC07]. They experimented with 5 layer multi-layer perception (MLP) with bottle-neck layers in the middle. After training, the output of bottle-neck layers was used as features for a GMM-HMM recognition system.

The system was trained using NIST RT'05 meeting data. The results of the evaluation compared with probabilistic features of the same size are presented in table 2.9. BNF show a improvement of system accuracy compared to probabilistic features.

feature size (NN output)	64 (25)	69 (30)	74 (35)	79 (40)	84 (45)
probabilistic	26.1	25.9	25.6	25.7	25.7
bottle-neck	25.2	25.2	**24.9**	25.2	25.0

Table 2.9: WER for probabilistic and bottle-neck features in full feature extraction framework [GKKC07]

3. Analysis of Existing Methods in the Optimization of DNN

This chapter presents methods of optimizing DNN from various points of views. It compares and analyzes the recent technologies and achievements in using DNN in ASR. Optimizing a DNN requires finding balance between ASR's system accuracy and performance. For example, big neural networks give better results, but they require a longer time to be trained. Training time and training accuracy depends on acoustic input, network size and topology, activation methods, pre-training and weights initialization. The main goal of this work is to develop simple and effective methods to optimize DNN. This goal could be achieved by finding optimal topology for DNN by applying decomposing algorithms to DNN, in order to compress the model or improve its performance.

3.1 Optimizations in Acoustic Input

The choice of acoustic input for training DNN can have a big influence on performance. Early attempts of DNN training used the Mel Frequency Cepstral Coefficients (MFCC). Recent studies have showed that DNN perform better on simple spectral features [DLH+13], [MDH12], [GMMW13]. In their works, the best performance was shown on 40 log filter-banks. The advantage of spectral features over MFCC can be explained by the fact that spectral features have more information (including possibly redundant or irrelevant information) [DLH+13] which might help DNN in classification task.

3.2 Optimizations in Output Layer

The performance of ANN can be also improved by using context dependent HMM as the output layer for DNN [DYDA12]. This allows coverage of a large number of HMM states. This, however, increases network size and negatively influences its performance. Modern computing technologies such as the Graphics Processing Unit (GPU) allow much faster training of the DNN, but cannot help it to avoid performance problems in the speech recognition process later, if the CPU is used.

These problems can be solved by restructuring the DNN. The main idea is to reduce the size of the DNN, without affecting its performance. See chapter 2.3.8 for one of the possible methods, Restructuring of DNN with Singular Value Decomposition by Xue et al. [XLG13].

3.3 Prevention of Overfitting

The most commonly used activation functions for hidden layers are linear, hyperbolic tangent and sigmoid. The main remaining problem is overfitting. Usually the problem of overfitting is solved by using cross-validation during training. This stops a network from overfitting and works perfectly for ANN. However cross-validation is not able to stop units from being overfitted during deep training. That is why new methods for DNN should be found and applied. One of the way to solve this problem is "dropout". The idea is to add particular noise (zero values or "dropout") to the activation function during training. Hinton et al. first proposed "dropout" in 2012 [HSK+12]. In 2013 Dahl et al. applied "dropout" for ASR task in combination with rectified linear units [DSH13].

3.4 Pre-training and Weights Initialization

Deep learning is divided into two important parts: pre-training and fine-tuning. The main task of pre-training is to find sensible weights for training, because a neural network with directly connected layers is difficult to train and it takes a long time. To avoid this problem Hinton proposed using pre-training based on Restricted Boltzmann Machine (RBM) [Hin07]. Training a RBM has an advantage compared to training directly connected network. First, hidden units help visible units to find the right distributions. Second, a poor approximation to the data distribution is generated. Third, directly connected neural network tries to learn hidden causes that are marginally independent.

The RBM can be replaced by autoencoders with a single hidden layer [Hin07]. This further development showed that denoising autoencoders perform well [VLBM08] in speech recognition tasks [GMMW13].

3.5 Optimization of Size and Topology of DNN

The introduction of deep learning algorithm allowed the training of very deep networks with a big number of units in layers. However, unlimited increases of the depth and size of networks do not bring the same improvement. According to Mohamed et al. [MDH12] and Graves et al. [GMH13], increasing the network's depth stops bringing significant improvement after fifth layer. Another important property of DNN is that many small-sized layers are better than one big one. In general, the best topology should be found according to acoustic input, the activation function used and other parameters. Choosing the size and topology of DNN is a trade-off between the network's number of parameters and its accuracy.

3.6 Types of ANN

The three major types of ANN used in ASR are usually deep full-connected neural networks (DNN), Deep Convolutional Networks (CNN) and Deep Recurrent Networks (RNN). Both CNN and RNN showed they could be applied ASR and achieve better accuracy than DNN [SMKR13] and [GMH13]. However, DNN with sensible weight initialization and activation function show comparable accuracy with the benefits of simple implementation, training and optimization. The sensible choice for ASR task could be a DNN using denoising auto-encoders.

3.7 Restructuring of DNN

One of the recent approaches in optimizing DNN is restructuring. As Xue et al. proposed [XLG13], restructuring can decrease the number of parameters in DNN, which improves its performance. The potential application of restructuring could not only leads to performance improvement, but also the network's recognition accuracy. This approach could be achieved by optimizing the size and topology of DNN, which cannot be done by pre-training and fine-tuning.

3.8 Summary

The sensible way to implement and optimize the DNN is to use full-connected neural networks together with various optimization methods. Autoencoders provide the possibility of sensible initial weight initialization as a DBN, but also control overfitting of units as "dropout". Additionally, recent methods of restructuring DNN can be easily applied to full-connected DNN.

4. Model Design and Implementation

In this chapter a design of used models and their implementation are presented. The model design shows in which way a DNN can be optimized, how this models can be implemented and a possible advantage of using this models.

4.1 Baseline System

The models would be applied on a baseline system, which can be used with a set of DNN with various topologies and a size of hidden layers. The description of baseline system and DNN can be found in the following sections.

4.1.1 Baseline System

The system used in this experiment was trained and decoded using the Janus Recognition Toolkit (JRTk) developed at Karlsruhe Institute of Technology and Carnegie Mellon University [SMFW01]. The neural networks were trained on 237 hours of Quaero training data of German language from 2010 to 2012.

For the audio sampling was done at 16Hz with 32ms window size. The feature extraction was done on 40 log mel scale filterbank coefficients (lMEL) with Pitch and FFV, which results a feature vector with 54 elements. The acoustic models of all systems are context-dependent quinphones with three states per phoneme, using a left-to-right HMM topology without skip states. The GMM models were trained with incremental splitting of Gaussians training (MAS), followed by optimal feature space training (OFS) and refined by one iteration of Viterbi training. A vocal tract length normalization (VTLN) was not used.

The training of neural networks was performed using Theano library [BBB+10].

4.1.2 Deep Neural Network Topology and Training

In this work different typologies with various sizes and numbers of hidden layers will be tested. The input layer contains 702 units and has a hyperbolic tangent (tanh)

activation function. The size of the hidden layers varies from 400 units to 2000 units; the number of hidden layers varies from 1 to 10. The visible activation function of units in hidden layers is sigmoid. The output layer has 6016 units, based on 6016 context dependent phone states. The activation function of output layer is softmax.

During pre-training the weights of hidden layers are initialized using stacked denoising autoencoders, proposed by [VLBM08]. The denoising autoencoder works the same as the autoencoder proposed by Bengio et al. [BLPL06], but with the modification, that input is partly corrupted. This corruption is to avoid overfitting and allows features to be extracted from large hidden layers. The input corruption can be represented as a stochistic process $q_D(\tilde{x} \mid x)$ to input vector x. To destroy the input a fixed number of components vd are chosen and their value set to 0. So initial the input x is mapped to the corrupted version $\tilde{x}$. The hidden representation now looks as follows:

$$y = f_\theta(\tilde{x}) = s(W\tilde{x} + b) \tag{4.1}$$

Then the reconstruction z can be computed as:

$$z = g_{\theta'}(y) = s(W'y + b') \tag{4.2}$$

Now z is a deterministic function of $\tilde{x}$, which represents a stochastic mapping of x. The cross-entropy error is used to compare input with output in order to find the necessary adjustment for neural network weights:

$$L_{\mathbf{H}}(x, z) = \sum_i x_i log z_i + (1 - x_i) log z_i \tag{4.3}$$

The hidden layers of neural networks were pre-trained with the following parameters:

Parameter	First hidden layer	Following hidden layers
corruption	0.3	0.3
loss function	mean squared error	cross entropy
batch-size	128	128
minibatches	2M	1M
learning-rate	0.001	0.01

Table 4.1: Parameters for hidden layer pre-training

After pre-training the output layer is added to neural network and then fine-tuned using back-propagation. The results with various sizes and numbers of hidden layers are available in chapter 6 “Results and Evaluation”.

4.2 Singular Value Decomposition

Singular Value Decomposition (SVD) is a factorization of a matrix. The decomposition of $m \times n$ matrix A is seen as:

$$A_{m \times n} = U_{m \times n} \Sigma_{m \times n} V^T_{n \times n}, \tag{4.4}$$

where U is unitary matrix, V^T is conjugate transpose of the unitary matrix V, and Σ is diagonal matrix with non-negative real numbers on the diagonal.

The matrix A can be decomposed by using approximation with the matrix lower rank A_k. This can be done by replacing the lowest singular values in matrix Σ with zeros. The result of the approximated matrix is

$$A_k = U_{m\times k}\Sigma_{k\times k}V^T_{k\times n}. \tag{4.5}$$

The equation below shows a preciser presentation of approximation problem.

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} = \begin{bmatrix} u_{11} & \dots & u_{1n} \\ \vdots & \ddots & \vdots \\ u_{m1} & \dots & u_{mn} \end{bmatrix} \times \begin{bmatrix} \sigma_{11} & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_{kk} & \dots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & \sigma_{nn} \end{bmatrix} \times \begin{bmatrix} v_{11} & \dots & v_{1n} \\ \vdots & \ddots & \vdots \\ v_{n1} & \dots & v_{nn} \end{bmatrix}$$

$$\approx \begin{bmatrix} u_{11} & \dots & u_{1n} \\ \vdots & \ddots & \vdots \\ u_{m1} & \dots & u_{mk} \end{bmatrix} \times \begin{bmatrix} \sigma_{11} & \dots & 0 \\ \vdots & \ddots & \vdots \\ \sigma_{m1} & \dots & \sigma_{kk} \end{bmatrix} \times \begin{bmatrix} v_{11} & \dots & v_{1n} \\ \vdots & \ddots & \vdots \\ v_{k1} & \dots & v_{kn} \end{bmatrix} \tag{4.6}$$

After approximation the matrix A can be replaced with two smaller matrices, U and W, where $W_{k\times n} = \Sigma_{k\times k}V^T_{k\times n}$.

The final equation for matrix decomposition is

$$A_{m\times n} = U_{m\times k}W_{k\times n}. \tag{4.7}$$

The resulting equation is able to decompose the weights matrix in the DNN, in order to compress the original DNN [XLG13].

4.3 Applying SVD on Weight Matrix of DNN

The connections between layers in a DNN can be represented as weight matrices. The weight matrix $A_{m\times n}$ between layers i and j has the dimension $m \times n$, where m is the number of units in layer i and n is the number of units in layer j. After applying SVD to matrix $A_{m\times n}$, we have two matrices of a smaller size $U_{m\times k}$ and $W_{k\times n}$. Then the matrix A is replaced with matrices U and W in that order: first matrix U and then W. This process is illustrated in figure 4.1.

The new layer between layers i and j has k units. For new layer the sigmoidal activation function is chosen. In order to insure better fine-tuning for new weigths, the values of bias in each of three layers after SVD are set to 0.

4.4 Non-rank Decomposition of Hidden Layers

This model applied SVD to weight matrices between the first and the last hidden layers. Weight matrices between input layer and first hidden layer, as well as between last hidden layer and output layer remain untouched.

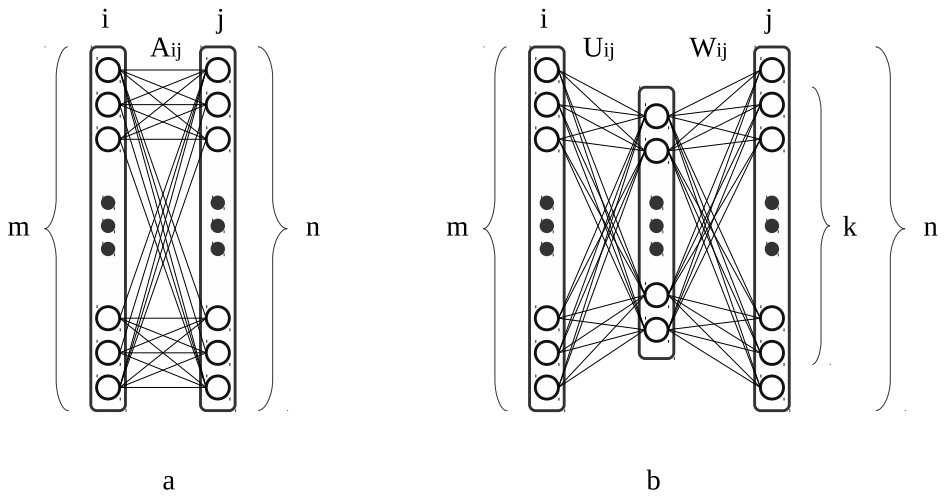

Figure 4.1: An example of SVD applied to a weight matrix in a DNN. **a**: matrix before SVD. **b**: matrix after SVD

SVD applied on a weight matrix, produces two matrices U and W, as described in chapter 4.2. If a rank of matrices is not reduced after applying SVD on square weight matrix of hidden layer $A_{m\times m}$, the resulting matrices would also be square $U_{m\times m}$ and $W_{m\times m}$. So basically the hidden layer is doubled. This procedure may allow an increasing of the depth of a network and achieve better accuracy, because the weights of the DNN would be set to sensible values.

4.5 Low-rank Decomposition of Hidden Layers

Decomposing a DNN can bring the advantage of removing units in hidden layers that have not been trained during a fine-tuning, from the network. The main goal of this attempt is to increase the network's performance, but to keep the total amount of units constant. Theoretically, this would give the performance of much bigger network while still having the advantages of a small network. The sensible way of performing this task is to find the topology, then improve by adding new hidden layers. As shown by Mohamed et al. [MDH12], adding more than four hidden layers to DNN does not show any significant improvement. This improvement is too small compared to the size of the DNN and possible performance problems.

In order to perform such a decomposition, the parameter k is set to the half number of singular values on the diagonal in matrix Σ. If matrix $A_{m\times m}$ is square, the k is set as half of m. For example, the matrix with an input layer 702 units, four hidden layers at size 1200 units and an output layer at size of 6016 units is taken (figure 4.2).

The SVD could applied to weight matrices between layers 1 and 2, 2 and 3, and 3 and 4. The matrices between the input layer 0 and 1, the last hidden layer 4 and the output layer 5 remain the same.

After applying SVD, the resulting neural network has more layers (figure 4.3), but the number of units remains the same (table 4.2).

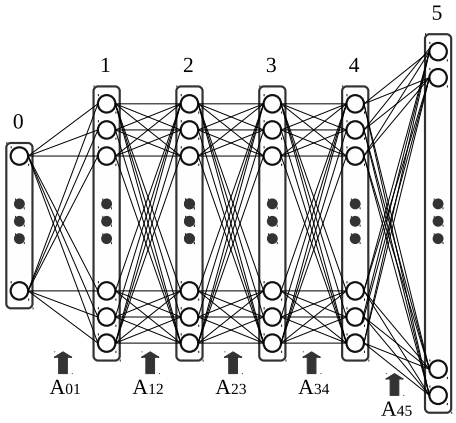

Figure 4.2: Neural network before applying SVD

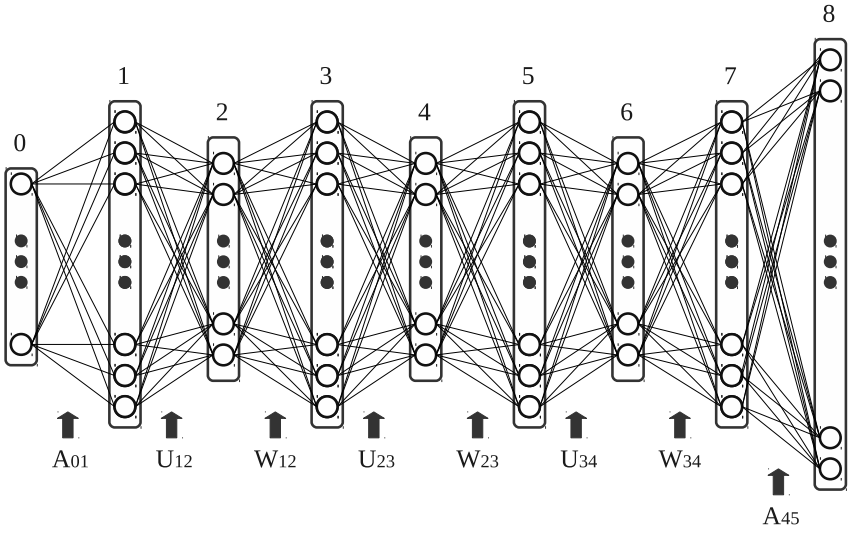

Figure 4.3: Neural network after applying SVD

Network	Input layer	Hidden layers	Output layer	Number of hidden layers
Before SVD	$842.4K$	$3 \times 1.2K \times 1.2K = 4320K$	$7219.2K$	4
After SVD	$842.4K$	$6 \times 1.2K \times 0.6K = 4320K$	$7219.2K$	7

Table 4.2: Number of parameters and hidden layers after applying SVD to a network with 4 hidden layers

4.6 Low-rank Decomposition of Output Layer

Due to using Context-Dependent HMM as output for DNN, the output layer has a relatively big size. The next attempt is thus to find an influence of applying SVD to the output layer (figure 4.4). The For example, in the system used here, the output layer has 6016 units, so the weight matrix, which connects it with previous layer has $1200 \times 6016 = 7219200$ parameters.

Applying SVD to the output layer could significantly decrease this number. The result of applying SVD to the output layer's weight matrix with a size of 1200 by 6016 units can be seen in table 4.3: leaving only 50% of singular values allows the

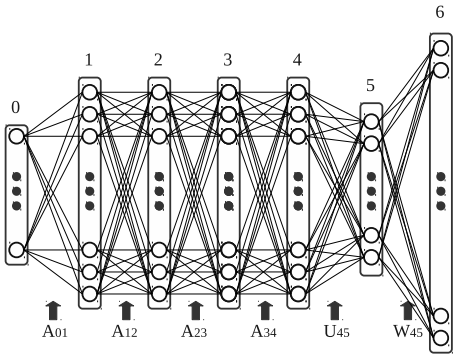

Figure 4.4: Low-rank Decomposition of Output layer

number of parameters to decrease by 40%. Theoretically, this approach could not only decrease number of parameters, but also improve DNN accuracy by deleting not-trained units.

Output layer	Number of parameters	% of original
Original 1200×6016	7219200	100%
After SVD with $k = 600$	4329600	59.97%
After SVD with $k = 120$	865920	11.99%
After SVD with $k = 42$	303072	4.20%

Table 4.3: Number of parameters in output layer after applying SVD

4.7 Insertion of Bottle-neck Layers to DNN with SVD

This attempt explores the influence of bottle-neck layers in DNN. The bottle-neck is a layer in a neural network, whose size is significantly smaller than of other layers. The bottle-neck is usually placed before the last layer in the DNN. This can help to prevent overfitting by reducing the number of not pre-trained weights [MDH12]. By using SVD the bottle-neck can be inserted after training the DNN. The idea is to leave only significant singular values, so theoretically this might help to improve DNN performance. In order to achieve this goal, various numbers of bottle-neck layers would be added to the DNN. The process of inserting bottle-neck to a DNN with 4 hidden layers (figure 4.2) is illustrated by figure 4.5.

4.8 Extracting Bottle-neck Features with SVD

As proposed by Grézl et al. [GKKC07], a bottle-neck layer could be used to extract features. SVD allows to creation of such a bottle-neck layer, leaving only significant information. The extraction of BNF could be done in two ways: with fine-tuning after SVD and without. After creating the DNN, the acoustic model and the resulting system would be trained.

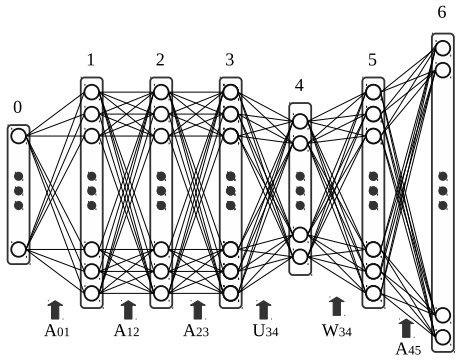

Figure 4.5: An insertion of bottle-neck layer to DNN with 4 hidden layers

4.9 Step-by-Step Fine-tuning of Low-rank Decomposed DNN

After implementing all of approaches detailed above, the next sensible idea is to combine the best of them in one DNN. Various approaches are going to be applied and fine-tuned one followed by the next. Theoretically, this could give a better improvement than each approach by itself.

4.10 Summary

Decomposing a DNN without increasing the network's size, and inserting of bottle-neck layers have a good theoretical basis to be able to improve the performance of a DNN. In the next chapter these approaches will be evaluated.

5. Results and Evaluation

5.1 Topology and Performance Analysis

The topology analysis started from the layers pre-training. The pre-trained setup consisted of five sets with various hidden layer with sizes of 400, 800, 1200, 1600 and 2000 units. Each set contained 11 pre-trained layers. Table 5.1 presents the time taken to pre-train 11 layers for each set. The dependence between size and pre-training time has a linear character (see figure 5.1).

Hidden layer size	400	800	1200	1600	2000
Pre-training time	13:30	27:12	39:47	61:29	102:12

Table 5.1: Pre-training time (h:m) for 11 layers

After pre-training, the DNNs with various parameters were fine-tuned and evaluated on a Quaero evaluation task. Figure 5.2 presents the dependence between the number of layers and the time spent for fine-tuning. As it can be seen, as the number of units in hidden layers increases, the time for fine-tuning increases dramatically and reaches almost 168 hours for the biggest network.

Figure 5.3 presents the dependence between the number of layers and the validation error for networks with different hidden layer sizes. The validation error stops decreasing significantly after the network has more then 5 layers and after 7th or 8th layer starts growing again. The dependence between the number of hidden layers and the Word Error Rate (WER) is noticeable (see figure 5.4). Networks with hidden layers of sizes from 1200 to 2000 units, showed a similar level of WER. So it can be assumed that increasing the size of hidden layers to more than 1200 units is not sensible.

In order to prove this, the corrected sample standard deviation s of WER would is analyzed:

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2} \tag{5.1}$$

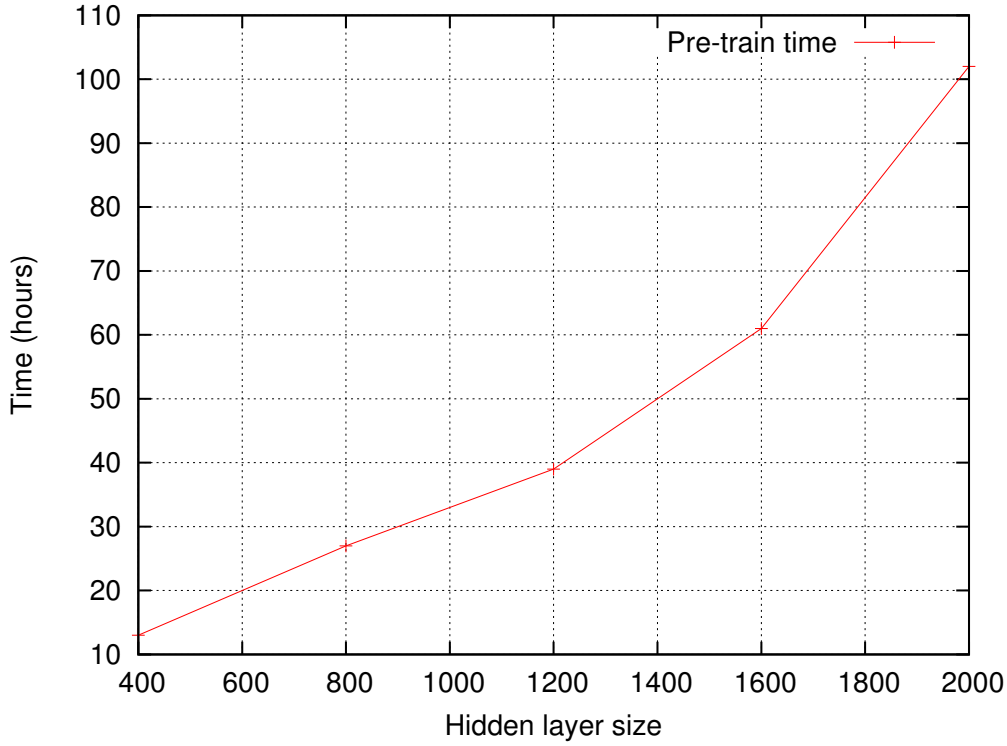

Figure 5.1: Dependence between size and pre-training time

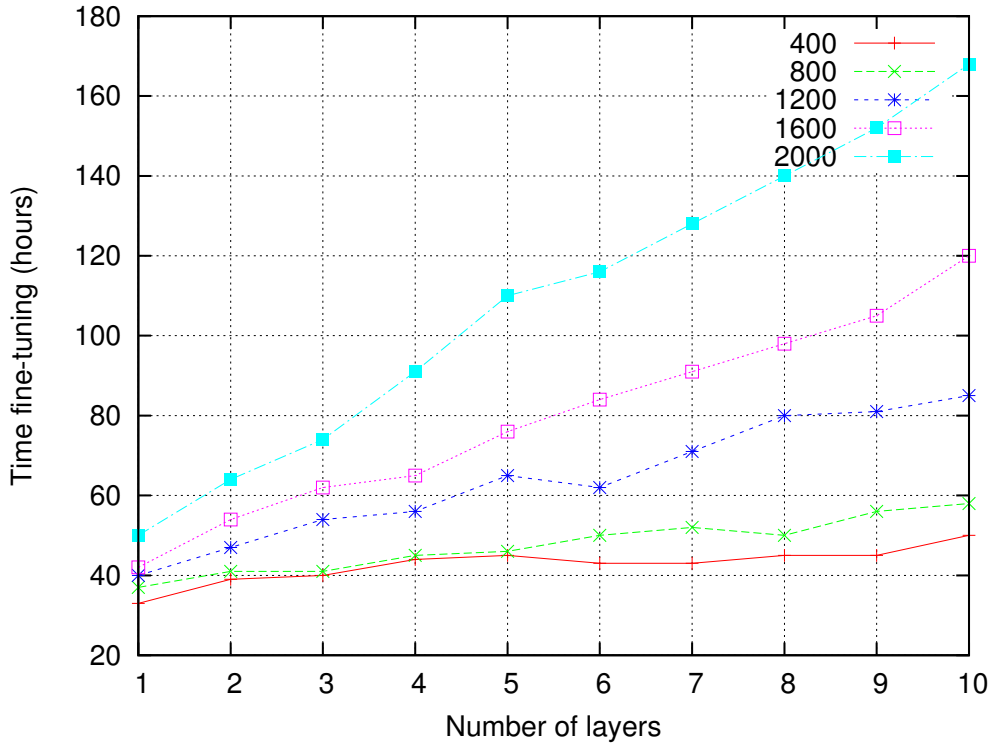

Figure 5.2: Dependence between the number of layers and fine-tuning time

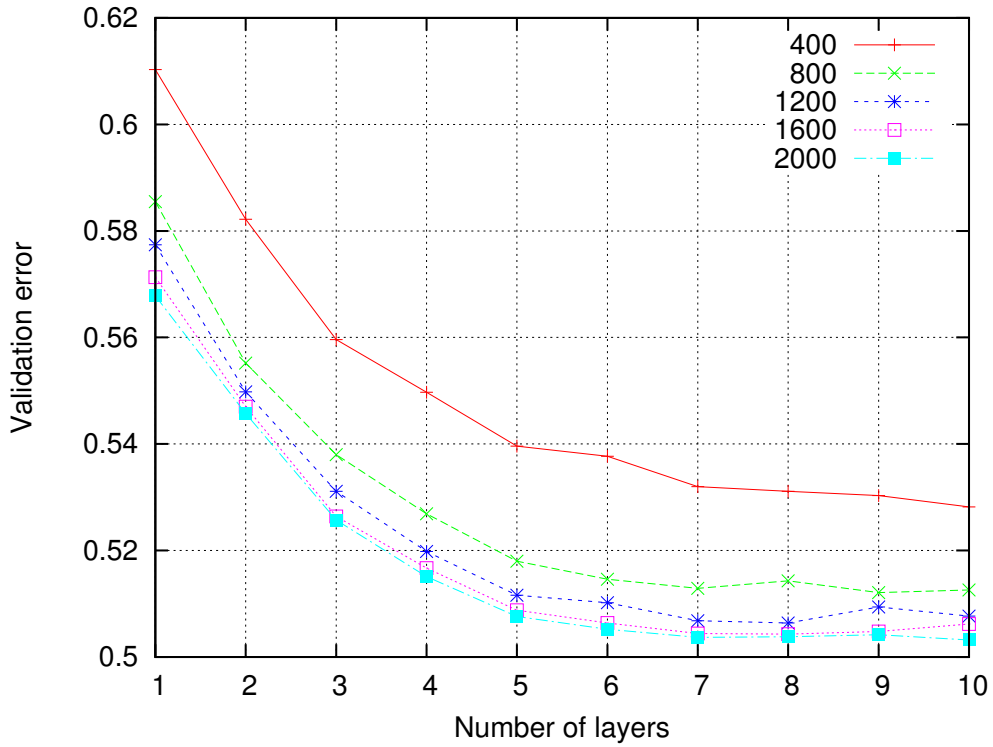

Figure 5.3: Dependence between the number of layers and validation error

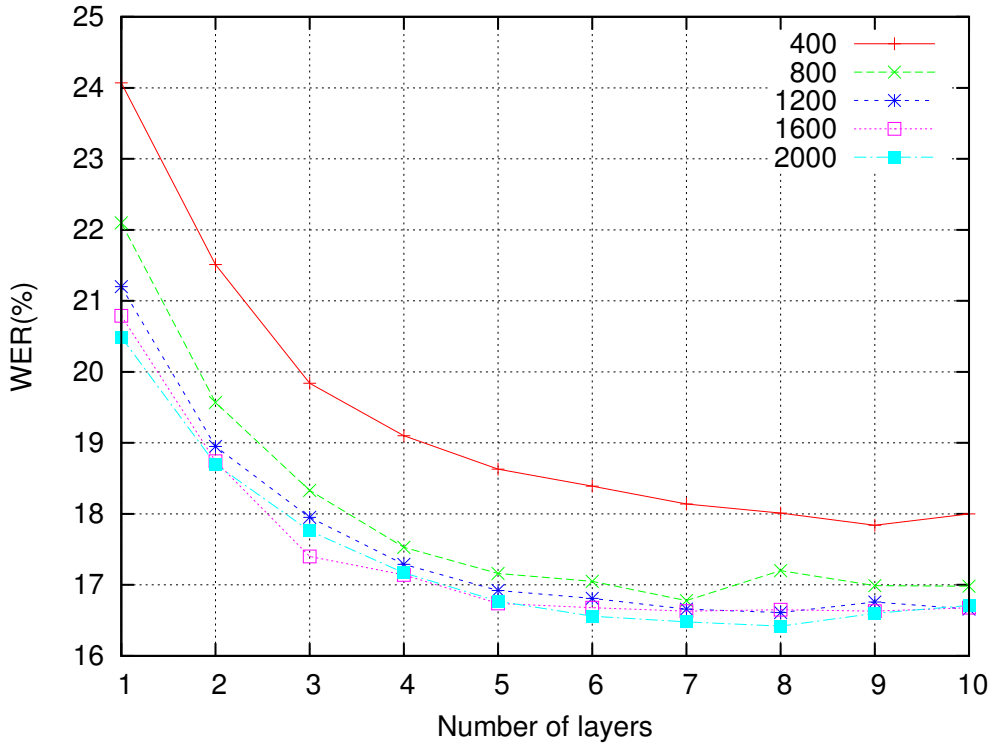

Figure 5.4: Dependence between the number of layers and WER(%)

The sample standard deviation s of all networks is presented in table 5.2. According to this table, the deviation gets smaller when the number of hidden layers is increased. It means, that the difference between WER of networks with different size of hidden layers get smaller with increasing the number of layers.

Number of hidden layers	$\overline{x}$	s^2	s
1	21.73	2.08	1.44
2	19.49	1.39	1.18
3	18.26	0.90	0.95
4	17.65	0.68	0.83
5	17.24	0.63	0.79
6	17.10	0.55	0.74
7	16.94	0.46	0.68
8	16.98	0.42	0.65
9	16.96	0.26	0.51
10	17.01	0.33	0.57

Table 5.2: Corrected sample standard deviation s for all size hidden layers

As mentioned above, networks with hidden layers of sizes from 1200 to 2000 units showed a similar level of WER. In order to prove that, the networks with hidden layers size of 1200, 1600 and 2000 would be only taken. As it can be seen from table 5.3, the corrected sample standard deviation becomes much smaller. The corrected sample standard deviation of networks with 7 hidden layers is 0.1, which is about 7 times smaller than for all networks.

Further important information is that average WER ($\overline{x}$) stops decreasing at 8 layers and starts growing again.

Number of hidden layers	$\overline{x}$	s^2	s
1	20.83	0.13	0.36
2	18.80	0.02	0.13
3	17.70	0.08	0.28
4	17.20	0.01	0.08
5	16.81	0.01	0.10
6	16.68	0.02	0.13
7	16.59	0.01	0.10
8	16.56	0.02	0.12
9	16.66	0.01	0.09
10	16.68	0.00	0.03

Table 5.3: Corrected sample standard deviation s for network size hidden layers equal to 1200, 1600 and 2000

The results of the analysis are:

- The addition of further hidden layers does not bring much improvement for networks with more than 5 hidden layers.

- Increasing the size of hidden layers does not bring much improvement for networks with a size of more than 1200 units.
- For networks with hidden layers between 1200 and 2000 units, the WER stops decreasing at layer 8.

The sensible choice based on these results, is a network with a number of hidden layers between 4 and 8 and with hidden layers with a size of about 1200 units.

5.2 Analysis of Singular Values in DNN

To perform decomposition on a DNN, singular values of weight matrices of the DNN should be analyzed. After singular value decomposition, singular values are represented as numbers on a diagonal in matrix Σ. This numbers are sorted and divided into 5 groups. The results can be seen on figure 5.5.

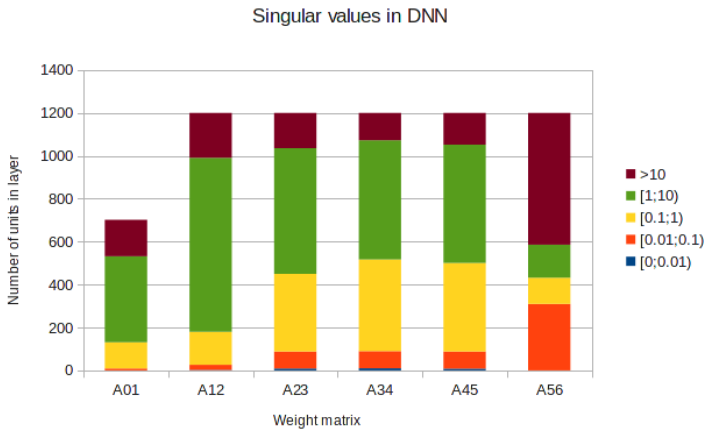

Figure 5.5: Analysis of singular values of DNN with 5 hidden layers

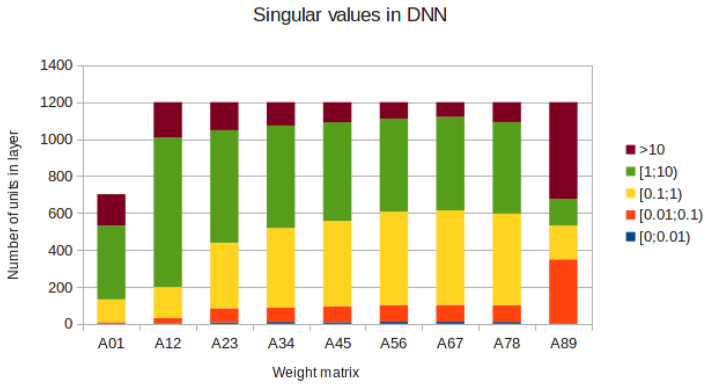

Figure 5.6: Analysis of singular values of DNN with 8 hidden layers

The values are not equally distributed over the weight matrices. Indirectly, this could indicate the amount of information stored in every weight matrix. For example, the values in matrices A_{12}, A_{23} and A_{56} are higher than in others. This

is comparable with the results of the number of layers/WER, presented in chapter 5.1. There has been shown, that increasing the number of layers does not give constant improvement. Theoretically, this is related to the number of untrained units in hidden layers. A similar distribution can be seen if the DNN with 8 hidden layers is taken (figure 5.6). The high of values decreases as the number of hidden layers increases. Theoretically weight matrices between the 4th and 8th layer could be cut using SVD without losing significant information that additionally might improve DNN accuracy.

5.3 Non-rank Decomposition of Hidden Layers

In order to found influence of SVD on DNN, a basis network with 5 hidden layers each with a size of 1200 units was chosen. Non-rank decomposition is applied on hidden layers as proposed in section 4.4. After applying SVD the new network has 9 hidden layers with 1200 units in each. In table 5.4 listed the evaluation's results for the original network, the network after SVD and a network with the same parameters for comparison. As a result, applying SVD allowed to decrease WER to 16.39%, which is 3.1% improvement compared to the original network and 2.2% to a network with the same number of parameters.

Network (Number of hidden layers × size of each layer)	WER (%)	Number of parameters	Fine-tuning time (h:m)
Original network (5 × 1200)	16.92	13.18M	65:27
Comparable network (9 × 1200)	16.76	18.67M	81:10
Network after SVD (9)	16.39	18.67M	91:10

Table 5.4: Non-rank decomposition of DNN

5.4 Low-rank Decomposition of Hidden Layers

In this section low-rank decomposition of hidden layers in the deep neural network is performed. The main goal is to achieve improved performance of the DNN while keeping the number of parameters constant. Low-rank decomposition will be applied to networks with different depths and sizes of layers.

5.4.1 Small 4 Layer DNN

First SVD would be performed on a small network with 4 hidden layers each with a size of 1200 units. The number of parameters in the original network is $702 \times 1200 + 3 \times 1200 \times 1200 + 6016 \times 1200 \approx 11.80M$. The WER of the original network is 17.29%. After decomposition the number of hidden layers increased to 7, but the number of parameters stayed constant. The WER of the decomposed network is 16.60%, which is 4% better, than the WER of the original network. The network after SVD showed better results than the similar network, but the number of parameters in the decomposed network is 25.9% fewer (see table 5.5).

Fine-tuning the original network took 56 hours, 54 minutes; for the network with 7 hidden layers time fine-tuning took 71 hours, 23 minutes; and for the network after SVD it was 58 hours, 29 minutes. So the total time for fine-tuning the original network and the network after SVD was longer, but it brings the advantages of a smaller number of parameters and better performance than a similar network with 7 hidden layers.

Network (Number of hidden layers × size of each layer)	WER (%)	Number of parameters	Fine-tuning time (h:m)
Original network (4 × 1200)	17.29	11.80M	56:54
Comparable network (7 × 1200)	16.66	15.93M	71:23
Network after SVD (7)	16.60	11.80M	58:29

Table 5.5: Applying SVD to a network with 4 layers each with a size of 1200 units

5.4.2 Small 5 Layer DNN

Next network to apply SVD has 5 hidden layers each with a size of 1200 units. The WER, number of parameters and fine-tuning time is presented in table 5.6. The network after SVD showed 16.41% WER, which is 3.01% better than the original network and 2.08% better than a comparable network with 9 hidden layers.

Network (Number of hidden layers × size of each layer)	WER (%)	Number of parameters	Fine-tuning time (h:m)
Original network (5 × 1200)	16.92	13.18M	65:27
Comparable network (9 × 1200)	16.76	18.67M	105:49
Network after SVD (9)	16.41	13.18M	67:12

Table 5.6: Applying SVD to a network with 5 layers each with a size of 1200 units

5.4.3 Large 4 Layer DNN

Next, a network with a bigger size of hidden layers will be decomposed. The main idea is to see if a bigger size of network could bring any significant improvement. The results are summarized in table 5.7. As it can be seen, the WER achieved by the network after SVD is 16.42%, which is almost the same as network with 5 hidden layers each 1200 units in size, but the number of parameters is almost twice bigger. From this point of view, it would be more sensible to use a network with a smaller number of units in hidden layers.

Network (Number of hidden layers × size of each layer)	WER (%)	Number of parameters	Fine-tuning time (h:m)
Original network (4 × 2000)	17.17	24.25M	91:53
Comparable network (7 × 2000)	16.48	35.70M	128:10
Network after SVD (7)	16.42	24.25M	108:22

Table 5.7: Applying SVD to a network with 4 layers each with a size of 2000 units

5.4.4 Validation of Results

In order to prove whether SVD could really improve the performance of a DNN, validation is performed. To validate the results, a DNN with the same topology as a DNN after SVD is taken, pre-trained and fine-tuned. This allows to find out if the improvement is determined only by the topology of the network or by SVD. For validation a small 5 layer DNN is chosen, because it showed the best results

in evaluation. A validation network with the same topology as the small 5 layers network after SVD was pre-trained and fine-tuned.

As it can be seen from table 5.8, just choosing the same topology as network after SVD does not bring the same improvement as performing the SVD. The validation network showed an even worse WER as comparable network with 9 $\times$ 1200 hidden layers (table 5.6)

Network (Number of hidden layers)	WER (%)	Number of parameters	Fine-tuning time (h:m)
Network after SVD (9)	16.41	13.18M	67:12
Validation network (9)	16.83	13.18M	66:59

Table 5.8: Validation results of network with 5 layers (size 1200) after SVD

5.4.5 Summary

Low-rank decomposition of the hidden layers showed sufficient performance improvement (4.37%) in the best case and additionally allowed the same number of parameters to be kept.

5.5 Low-rank Decomposition of Output layer

In order to find the influence of output layer decomposition, SVD will be applied and various numbers of singular values will be left. The results are summarized in table 5.9. As it can be seen the best accuracy improvement has been shown by $k = 600$, which is equal to 2.23% of improvement compared to original network, also the number of parameters was reduces on 15.9 %. Low-rank decomposition of output layer can be also applied to reduce the number of parameters and keep the accuracy on the level of original network. For example by $k = 300$ the WER is 1.15 % better, but the number of parameters is on 27.9 % smaller.

Network (Number of hidden layers)	WER (%)	Number of parameters	Fine-tuning time (h:m)
Original (8)	16.61	17.30M	80:37
After SVD $k = 900$ (9)	16.37	16.60M	67:05
After SVD $k = 600$ (9)	16.24	14.55M	59:56
After SVD $k = 300$ (9)	16.42	12.48M	56:14
After SVD $k = 120$ (9)	16.74	11.22M	53:53
After SVD $k = 42$ (9)	17.33	10.71M	52:59

Table 5.9: Results of low-rank decomposition of output layer

5.6 Insertion of Bottle-neck Layer to DNN with SVD

Mohamed et al. explained that inserting a bottle-neck layer to a DNN can help to improve its performance by reducing number of not pre-trained weights [MDH12].

Applying SVD to the hidden layers of a DNN creates a bottle-neck layers, which can help improving accuracy of DNN. In this section the dependence between the size and number of bottle-neck layers will be presented. The experiment is performed on the neural network with 8 hidden layers each with a size of 1200 units.

5.6.1 Bottle-neck Layers with 50%-rank Decomposition

First the bottle-neck layer with a 50%-rank decomposition of hidden layer is inserted to the DNN. The bottle-neck layer is inserted into the last, two last and three last hidden layers of the network. The results of evaluation are summarized in table 5.10. As can be seen, inserting one bottle-neck layer improved the performance of the DNN by about 1%. Insertion of the second bottle-neck layer improved the performance on 1.56% and the improvement stops on the third bottle-neck layer. The number of parameters stays the same as in the original network, which means that a slight improvement can be reached without affecting the network's size.

Network (Number of hidden layers × size of each layer)	WER (%)	Number of parameters	Fine-tuning time (h:m)
Original (8 × 1200)	16.61	17.30M	72:37
Bottle-neck on last layer (9)	16.45	17.30M	62:28
Bottle-neck on last 2 layers (10)	16.35	17.30M	67:27
Bottle-neck on last 3 layers (11)	16.46	17.30M	76:50

Table 5.10: Bottle-neck layers with 50%-rank decomposition

5.6.2 Bottle-neck Layers with 10%-rank Decomposition

In the next experiment the influence of very small bottle-neck layers is researched. This time, 10% of the original hidden layer (120 units) is taken. The results are presented in table 5.11. A smaller bottle-neck shows less accuracy improvement, but it has the advantage of reducing the number of parameters by about 13.47% for the network with two last hidden layers: the one which shows the best accuracy improvement.

Network (Number of hidden layers × size of each layer)	WER (%)	Number of parameters	Fine-tuning time (h:m)
Original (8 × 1200)	16.61	17.30M	72:37
Bottle-neck on last layer (9)	16.48	16.20M	64:16
Bottle-neck on last 2 layers (10)	16.53	14.97M	66:48

Table 5.11: Bottle-neck layers with 10%-rank decomposition

5.6.3 Validation of Results

To validate the bottle-neck layer approach, a DNN with the same topology was trained. Validation was performed on the network showing the best accuracy improvement - DNN with two bottle-neck layers and 50%-rank decomposition (table 5.12). The validation results show, that inserting bottle-neck layers with SVD has positive influence on the DNN's accuracy.

Network (Number of hidden layers)	WER (%)	Number of parameters	Fine-tuning time (h:m)
SVD bottle-neck on last 2 layers (10)	16.35	17.30M	67:27
Validation network (10)	16.67	17.30M	81:48

Table 5.12: Validation bottle-neck layers with 50%-rank decomposition

5.7 Extracting Bottle-neck Features with SVD

Bottle-neck layers can be created very easily with SVD. Such a bottle-neck layer can be used to extract bottle-neck features (BNF). First, as a baseline, a network with 8 hidden layers each with a size 1200 units is chosen. The bottle-neck with a size of 42 units, created using SVD, is inserted into the last hidden layer. This bottle-neck layer is used for extracting features. First, the extraction is performed on the network without fine-tuning. Second, on the network, which was fine-tuned after SVD. In order to validate the results, a network with the same topology is trained and used for extracting features. The summary of results is presented in table 5.13

Network (Number of hidden layers)	WER (%)	Number of parameters	Fine-tuning time (h:m)
DBNF with SVD no fine-tuning (10l)	18.70	16.02M	-
DBNF with SVD with fine-tuning (10l)	17.96	16.02M	64:24
Validation DBNF (10l)	18.31	16.02M	77:57

Table 5.13: Results of extracting DBNF with SVD

As can be seen from the table, the network with no fine-tuning showes a worse WER as the network with fine-tuning after SVD. The WER of the network with fine-tuning is 1.9% better than DBNF network created without SVD. The results show that decomposing DNN with SVD, in order to extract bottle-neck features, can bring a slight improvement, which, however, is not very sensible because of the additional fine-tuning time.

5.8 Step-by-step Fine-tuning of Low-rank Decomposed DNN

After evaluating all proposed approaches, the sensible idea is to combine the best of them, in order to improve their accuracy. The best approaches should be applied to one network after another, and fine-tuned after each step. As a base network, a network with 8 hidden layers each with a size of 1200 units is chosen. First, two bottle-neck layers with 50%-rank decomposition are inserted to the base network. After SVD, the output layer of the resulting network is decomposed with leaving 600 singular values. In order to validate the resulting network, the neural network is created using the same approaches, but without fine-tuning after each step. This would allow to see if fine-tuning between applying various approaches can help to improve network accuracy. The results are summarized in table 5.14. There the results of each basic approach can also be found in the table.

Network (Number of hidden layers × size of each layer)	WER (%)	Number of parameters	Fine-tuning time(h:m)
Original (8 × 1200)	16.61	17.30M	72:37
Only Bottle-neck on 2 last layers (10)	16.35	17.30M	67:27
Only output layer with $k = 600$ (9)	16.24	14.55M	59:56
Bottle-neck 2 last layers + output layer with $k = 600$ (11)	16.35	14.55M	65:17
Bottle-neck 2 last layers + fine-tuning + output layer with $k = 600$ (11)	16.16	14.55M	67:27 & 65:28

Table 5.14: Result of step-by-step fine-tuning of low-rank decomposed DNN

The results show that combining the best approaches gave an improvement of WER (2.7%) compared to the original network. It is also 1,16% better than the approach of a bottle-neck on the last 2 layers approach and 0,5% better than the approach of an output layer with $k = 600$. Another interesting result is that a combining bottle-neck and SVD of output layer without fine-tuning after each step, shows worse accuracy than just a simple SVD of the output layer.

5.9 Summary

The results of the evaluation show that Singular Value Decomposition proved its ability in improving the accuracy of a network also by reducing number of parameters: so SVD can be efficiently used to optimize deep neural networks. The sensible way of applying SVD is to decompose output layer with $k = 600$. This attempt shows a balanced relation between key factors: accuracy improvement (2,23% in the best case), reducing the number of parameters and the time spent for fine-tuning. The comparison of the best attempts is listed in table 5.15

Network (Number of hidden layers × size of each layer)	WER (%)	Number of parameters	Fine-tuning time(h:m)
Original (5 × 1200)	16.92	13.18M	65:27
Non-rank SVD of hidden layers (9)	16.39	18.67M	91:10
Low-rank SVD of hidden layers (9)	16.41	13.18M	67:12
Original (8 × 1200)	16.61	17.30M	80:37
Output layer with $k = 600$ (9)	16.24	14.55M	59:56
Bottle-neck 2 last layers (50%) (10)	16.35	17.30M	67:27
Bottle-neck 2 last layers + fine-tuning + output layer with $k = 600$ (11)	16.16	14.55M	67:27 & 65:28

Table 5.15: Comparison of the best attempts

6. Conclusion and Outlook

In this work the optimal topologies for DNN were found and optimized by using singular value decomposition. Simple and effective methods of optimizing DNN were developed and evaluated.

The optimal topology for DNN in the used dataset are networks with 5 to 8 hidden layers each a size of about 1200 units. A bigger size of hidden layers does not bring significant improvement, but increases the number of parameters in network.

Choosing the best attempts to optimize DNN was based on the three main parameters: accuracy improvement, reducing the number of parameters and time spent for fine-tuning. The best way of optimizing the DNN is decomposition of the output layer with a rank of 50% of a last hidden layer. By using this approach, the decomposed network has a small number of parameters and shows the best WER (16.24 %) in the evaluation of single approaches. At the same time, this attempt requires a relative small time for a fine-tuning.

Non-rank decomposition of hidden layers shows a good improvement in WER (3%) compared to the original network, but it increases the number of parameters and needs a lot of time for fine-tuning. Low-rank decomposition of hidden layers with a rank of 50% of a network with five hidden layers has an advantage of keeping the number of parameters small, but shows a little accuracy improvement. Inserting bottle-neck layers with ranks of 50% and 10% shows a low accuracy improvement and has relative many parameters after decomposition.

Various approaches can be used in order to achieve different goals: maximal accuracy improvement, maximal reduction of number of parameters or a balanced solution with improvement in both accuracy and reduction of parameters. A combination of various approaches shows a significant accuracy improvement in WER (16.16 %), but has the disadvantage of a long fine-tuning time. All attempts that used singular value decomposition proved its ability to optimize DNN.

Future work for developing SVD is to find out how it would perform on a DNN build with different approaches and methods. The first important question is to find the best methods for reaching the maximum accuracy improvement and reduction

of parameters. Second, is to find the effect of using different activation functions for units or a combination of various activation functions. Another very important question is to research the performance of SVD on networks with different method of pre-training, for example Deep Belief Networks (DBN) or Restricted Boltzman Machine (RBM). Furthermore, as shown in section 5.7, SVD can be applied to extract bottle-neck features and shows better accuracy than original network.

The results of this work showed that singular value decomposition can be successfully applied in the task of automatic speech recognition and has a positive outlook for future development.

Bibliography

[BBB+10] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[BLPL06] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy Layer-Wise Training of Deep Networks. In *Neural Information Processing Systems*, pages 153–160, 2006.

[DHK13] Li Deng, Geoffrey Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *Proc. ICASSP*, 2013.

[DLH+13] Li Deng, Jinyu Li, Jui-Ting Huang, Kaisheng Yao, Dong Yu, Frank Seide, Michael Seltzer, Geoff Zweig, Xiaodong He, Jason Williams, et al. Recent advances in deep learning for speech research at microsoft. *ICASSP 2013*, 2013.

[DSH13] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *Proc. ICASSP*, 2013.

[DYDA12] George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition. *IEEE Transactions on Audio, Speech and Language Processing*, 20:30–42, 2012.

[DYP12] Li Deng, Dong Yu, and John Platt. Scalable stacking and learning for building deep architectures. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 2133–2136. IEEE, 2012.

[GKKC07] František Grézl, Martin Karafiát, Stanislav Kontár, and J Cernocky. Probabilistic and bottle-neck features for lvcsr of meetings. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 4, pages IV–757. IEEE, 2007.

[GMH13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks," icassp. 2013.

[GMMW13] Jonas Gehring, Yajie Miao, Florian Metze, and Alex Waibel. Extracting deep bottleneck features using stacked auto-encoders. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE*, 2013.

[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, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[Hin07] Geoffrey E Hinton. Learning multiple layers of representation. *Trends in cognitive sciences*, 11(10):428–434, 2007.

[HOT06] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[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.

[MDH12] Abdel-rahman Mohamed, George E Dahl, and Geoffrey Hinton. Acoustic modeling using deep belief networks. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):14–22, 2012.

[RHlBL07] Marc'aurelio Ranzato, Fu Jie Huang, Y lan Boureau, and Yann Lecun. Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition. In *Computer Vision and Pattern Recognition*, 2007.

[SLY11] Frank Seide, Gang Li, and Dong Yu. Conversational speech transcription using context-dependent deep neural networks. In *INTERSPEECH*, pages 437–440, 2011.

[SMFW01] Hagen Soltau, Florian Metze, Christian Fugen, and Alex Waibel. A one-pass decoder based on polymorphic linguistic context assignment. In *Automatic Speech Recognition and Understanding, 2001. ASRU'01. IEEE Workshop on*, pages 214–217. IEEE, 2001.

[SMKR13] Tara N Sainath, Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran. Deep convolutional neural networks for lvcsr. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8614–8618. IEEE, 2013.

[ST95] Ernst Günter Schukat-Talamazzini. *Automatische Spracherkennung: Grundlagen, statistische Modelle und effiziente Algorithmen*. Vieweg, 1995.

[TG01] Edmondo Trentin and Marco Gori. A survey of hybrid ann/hmm models for automatic speech recognition. *Neurocomputing*, 37(1):91–126, 2001.

[VLBM08] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[VSR12] Tuomas Virtanen, Rita Singh, and Bhiksha Raj. *Techniques for noise robustness in automatic speech recognition*. Wiley.com, 2012.

[XLG13] Jian Xue, Jinyu Li, and Yifan Gong. Restructuring of deep neural network acoustic models with singular value decomposition. In *Proc. Interspeech*, 2013.