CONNECTIONIST TEMPORAL CLASSIFICATION FOR ROBUST SPEECH RECOGNITION APPLICATIONS

Master Thesis at SONY Europe, Stuttgart, Germany

Presented on 29 August 2016
at IDIAP Research Institute in Martigny, Switzerland

Master Program in Communication Systems
École Polytechnique Fédérale de Lausanne

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Nadim Ghaddar

Accepted on the proposal of the jury :

Prof. Hervé Bourlard, President of jury

Lausanne, EPFL, 2016

Wings are a constraint that makes
it possible to fly.
— Robert Bringhurst

To my parents…

Acknowledgements

A lot of ups and downs have passed in order to accomplish this Master thesis, and a lot of people have helped me, whether directly or indirectly, to reach the finish line successfully.
First of all, many thanks to SONY Europe for accepting me to be part of their work, and in specific to the Speech and Sound Group for the stimulating environment and support. A Master student cannot imagine a better place to complete his thesis with this level of professionalism, freedom and autonomy in the work.
More specifically, I would like to express my deep gratitude to my supervisor, Mr. Wilhelm Hagg for his guidance and support throughout the six months and for the very useful discussions. I am also very grateful for Mr. Thomas Kemp, our group's leader, for the exciting ideas and research directions that he offered in our meetings.
A special thanks should go also to my university, École Polytechnique Fédérale de Lausanne for offering me the chance to apply to SONY Europe, and my academic advisor there Prof. Hervé Bourlard, for being always accessible for questions and updates throughout my thesis.
Finally, I would like to high-five all the students that I was able to meet at SONY Europe one-by-one. Rakshita, Vijay, Amira, Anna, Talaat, Xinyi, Luiz, you guys were amazing, and the main reason why this experience was exceptional!

Thank you!

Stuttgart, 12 August 2016 N. G.

Abstract

Automatic Speech Recognition (ASR) is the task of converting a speech signal into the sequence of words that it incorporates, by the help of an algorithm that is implemented on computers or computerized devices. Today, ASR systems find applications in a wide range of domains, including automatic call processing in telephone networks, automatic query-based information systems, human-to-machine interaction, and many others. One of the most popular approaches followed in ASR systems is to model the temporal variability of speech signals using hidden Markov Model (HMM) states. In the recent years, significant progress has been made into the acoustic modelling of these states based on the training of context-dependent deep neural networks (DNN's) with observed feature vectors extracted from the signals. The training procedure of a traditional DNN-based acoustic model requires first to obtain the state alignments of the input feature frames with the HMM phoneme states, because traditional objective functions require a network output target value for every input feature vector. Connectionist Temporal Classification (CTC) overcomes this requirement. CTC is a cost function that is well-suited for sequence labelling tasks, where large sequences of input data (e.g. input feature vectors of a speech signal) are transcribed with smaller sequences of discrete labels (e.g. phoneme labels spoken in the speech stream). On the other hand, another major factor that degrades performance of current ASR systems is background noise and reverberation in the recorded speech.

In this thesis, we try to show the suitability of CTC training for robust speech recognition applications. We conduct experiments on public speech corpora, invoking different noise levels on the speech data, while comparing with the traditional hybrid DNN-HMM framework. Our results show that CTC training achieves very much comparable results for the case of clean data, while still on a par with the traditional framework for noisy data. We argue that this is due to the size of the training set that we used in our experiments, which was relatively small.

***Keywords* - Speech Recognition • Hidden Markov Models • Neural Networks • Connectionist Temporal Classification • Robustness**

Contents

List of Figures

List of Tables

Introduction

The automatic generation of text transcriptions from speech has many applications in a vast range of domains. For automatic speech recognition systems, the input is the speech signal generated from the continuous excitation of the vocal tract, while the output is the sequence of words that should adhere to the laws of grammar and language.

A main problem in speech modelling is the variable length in the sequences of acoustic features, meaning that two speech utterances, even if identical, still can have different feature vectors, depending on how the words are pronounced and in what speed. Hidden Markov models (HMM's) come to handle this problem to some extent. HMM's, with their sequential structure, are able to model the temporal dimension of speech and to automatically segment the input feature vectors during training. One of the most common learning approaches is based on Gaussian mixture models-based HMM's (or GMM/HMM's), where each feature vector is assumed to be generated from a hidden state that follows a Gaussian distribution.

Recently, the introduction of deep neural networks (DNNs) into the acoustic modeling of speech signals showed a great success in speech recognition tasks, outperforming the traditional GMM/HMM systems. DNN's are also used in the context of HMM's, where multiple nonlinear processing layers allow to predict complicated boundaries between the HMM states. However, a fundamental difficulty in following a DNN/HMM hybrid approach is the need for pre-segmentation of the input feature frames and mapping each segment to a specific target label, which is in general not known or unavailable prior to DNN training.

Connectionist Temporal Classification (CTC) comes to relax this difficulty by defining a single target function for connected speech units rather than a per-frame criterion. This opens the door towards a pure connectionist approach for speech recognition where supervised training is done over continuous units of speech.

On a different scale, current ASR systems still suffer from severe degradation in performance when recordings are performed in noisy environments. Still a lot of work is to be done for further improving the robustness against environmental distortions. In this work, we investigate the suitability of CTC training for more robust speech recognition. We perform experiments over different levels of noisy data and compare with the DNN/HMM hybrid approach as a baseline.

The subsequent chapters are organized as follows. In Chapter 1, we give an overview of speech recognition systems and a peak into previous work towards achieving more robust speech recognition systems. In Chapter 2, we go through Hidden Markov Models (HMMs) and discuss their suitability for speech recognition tasks. We present an overview of neural networks and the DNN/HMM hybrid approach in Chapter 3. In Chapter 4, we discuss in detail Connectionist Temporal Classification (CTC) and a comparison with the traditional hybrid approach. Finally, we present our experimental results in Chapter 5, before concluding our work.

1 Overview of Speech Recognition

Automatic Speech Recognition (ASR) is the task of transforming an input speech signal into its corresponding word transcription. The ASR domain has impacts into a lot of human-to-human and machine-to-human applications. In this chapter, we introduce the basic architecture of ASR systems and the main implications facing the domain.

1.1 Basic Architecture of ASR Systems

The very basic architecture of an ASR system is shown in Fig. 1.1. As shown, an ASR system is made up of four main components: a signal processing and feature extraction component, an acoustic model (AM), a language model (LM) and a decoder.

1- Feature Extraction Module: This module takes as input the speech signal, enhances the signal by removing noise and channel distortions and extracts feature vectors that represent the signal's time and frequency representations. The feature vectors are inputted to the subsequent acoustic model.

2- Acoustic Model: The acoustic model integrates knowledge about phonetics and acoustics to generate from the inputted feature vectors a set of local hypothesis likelihoods, associated with certain phonetic classes that model the distinct units of sound in a certain language of

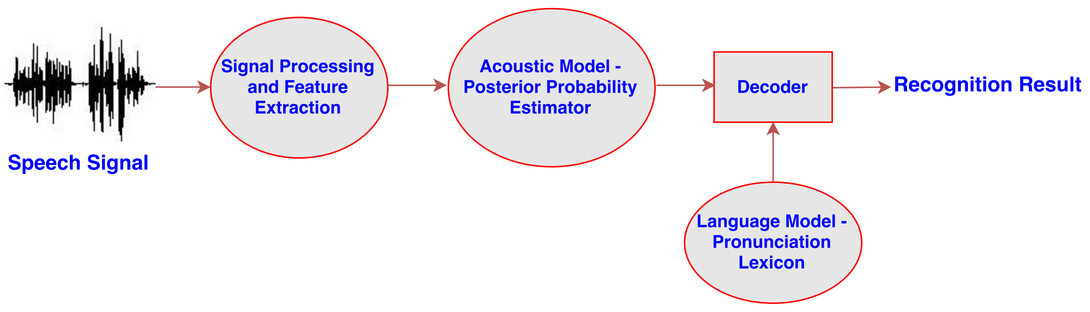
Speech Signal
Signal Processing and Feature Extraction
Acoustic Model - Posterior Probability Estimator
Decoder
Recognition Result
Language Model - Pronunciation Lexicon

Figure 1.1 – Basic Structure of an ASR System

speech (i.e. phonemes). The model is usually trained on a large amount of speech data which contains a large number of occurrences of the corresponding phonemes of the language. The output from this model is typically a vector of likelihoods for all possible phonemes and all acoustic frames.

3- Language Model: The language model estimates the probability of a certain word sequence, by learning the correlation between different words in a certain training speech corpus. Also, a pronunciation lexicon is used to model the phonemic units corresponding to each word, and often, even these units are subdivided into different contextual classes, depending on the neighboring units to the left and right. This allows to better represent the variability in the generated phonemes depending on the neighboring speech sound units that are pronounced. In this thesis, however, our work is focused on different variants of the afore-mentioned acoustic model.

4- Decoder: The decoder combines the outputs from the acoustic and language models and transforms the hypothesized phoneme sequence into a global decision that is suitable for word and sentence recognition.

Automatic speech recognition is therefore a difficult task for several reasons:

• The acoustic speech is often degraded due to additive and convolutional noise that are present in the speech signals while recording due to reverberation in acoustic environments, additive noise from microphone electronics, effects of using far-field microphones ... This can be combated by a range of front-end approaches introduced to the feature extraction module or to the acoustic model. This is to be discussed more in Section 1.2.

• Pronunciation is highly affected by the context within which each word or phoneme comes. Apart from the intra-speaker variabilities (caused by speed of speaking and emotional state of the speaker), there are also large inter-speaker variabilities, caused mainly by the gender of the speaker and the differences in regional dialects. These variabilities will show evidently in the feature vectors extracted from the different speech signals. Performing the training in one specific speaker can help adapt the acoustic model to this speaker and thus reduce the effects of inter-speaker variabilities.

• Speakers may tend to use words and produce sounds that are not within the recognizer's dictionary. Here, a larger dictionary should be used, along with more training data.

1.2 Robustness in Speech Recognition

Background noise and reverberation are key factors of performance degradation of current automatic speech recognition systems. Therefore, there is a pressing need to make ASR systems robust against environmental distortion. A lot of approaches have been followed to address this issue. Three different categories of robustness approaches can be distinguished [1]:

• Front-end processing - this involves signal- or feature-level enhancement techniques that increases the robustness of the extracted feature vectors against noisy environments. Signal processing techniques include spectral subtraction, Wiener filtering and MMSE estimators [2, 3, 4]. In [5, 6], a joint approach of long-term log spectral subtraction along with noise cancellation is proposed to deal with noisy and reverberant speech data. By modeling reverberation as a *fixed* linear time-invariant filter, the filter response can be eliminated by subtracting the mean around each analysis frame from the log magnitude spectrum of that frame. A feature-level enhancement approach is presented in [7], where deep de-noising autoencoders are used to recover clean speech features from noisy reverberant ones. The autoencoder is first pre-trained as a sequence of restricted Boltzmann machines, and then the reconstructed feature vectors are fine-tuned by clean speech vectors. The reconstructed features are then passed to the acoustic model, and speech recognition is done over them.

• Model adaptation - this modifies the acoustic model parameters in order to make it less sensitive to speaker and environment variations. In [8], an ideal hidden-activation masking approach is presented, where inconsistent hidden activations in deep neural network (DNN) are discarded in a hybrid DNN/HMM framework. (A thorough explanation of a hybrid DNN/HMM is presented in section 3.5). The discarding procedure depends on the similarity between the DNN's noisy hidden activations and its clean hidden activations. This can be seen as a deterministic way of applying the dropout technique in neural network training by identifying the noise-invariant hidden nodes. On the other hand, recurrent neural networks in a hybrid setup are used in [9] to achieve better robustness.

• Output transformation - this transforms the output of the acoustic model to eliminate any kind of irrelevant variations in these outputs. In [10], for example, the state posteriors generated from the DNN are converted via an output discriminative linear transform.

In [11, 12], an end-to-end approach is proposed for automatic speech recognition using end-to-end deep learning. The system uses connectionist temporal classification as its cost function ([13]), and training is done using multiple GPU's and with thousands of hours of speech data. Connectionist Temporal classification is to be discussed more in chapter 4. The system does not implement any hand-designed components to model noise, reverberation or any environmental variations, but instead it learns a function that is inherently robust against such effects. In [11], authors report a 37.4% relative word error rate (WER) improvement over the state-of-the-art performance on noisy speech. The two papers [11, 12] are the motivation for our work in this thesis.

In the next chapter, we give an overview over Hidden Markov Models(HMM's) that are prominent in modern automatic speech recognition systems.

2 Hidden Markov Models

Hidden Markov Models (HMM's) are a key block element in a traditional ASR systems. HMM's are used to model the temporal variability of speech, with hidden states emitting observations. The popularity of HMM's stems to its ability of being a generative sequence model of acoustic speech features modeling the correlation between the temporal dimension in speech with its frequency-domain properties that are represented in the extracted feature vectors ([14]). We first go through an overview of Hidden Markov Models in general.

2.1 Definition

An HMM (Figure 2.1) is a stochastic process or a probabilistic graphical model that is made of states, and transitions between these states that are define by probabilities. The state at time t+1 is independent of the state at time t-1, given the state at time t, i.e. it is a first order Markov process. An HMM λ is defined by the following:

• A set of states $Q = \{s_1, s_2, \dots, s_N\}$.

• A transition matrix of probabilities, describing the possibility of transition of one state to another, i.e. $p(q_t = s_i | q_{t-1} = s_j)$, for all $s_i, s_j \in Q$.

• A probability distribution over the possible initial states $p(q_1 = s), \forall s \in Q$.

• An emission model of observations x, defined by probabilities $p(x|s)$, where $s \in Q$.

Let Q_T be the set of all possible state sequences of length T, i.e. $Q_T = \{\mathbf{q} = q_1, \dots, q_T : q_i \in Q, 1 \leq i \leq T\}$. The Markovian assumption of HMM's allows an easy formulation of the likelihood of an observation sequence $\mathbf{x} = \{x_1, \dots, x_T\}$, given a state sequence $\mathbf{q}$ of length T:

$$p(\mathbf{x}|\mathbf{q}, \lambda) = \prod_{t=1}^{T} p(x_t|q_t), \tag{2.1}$$

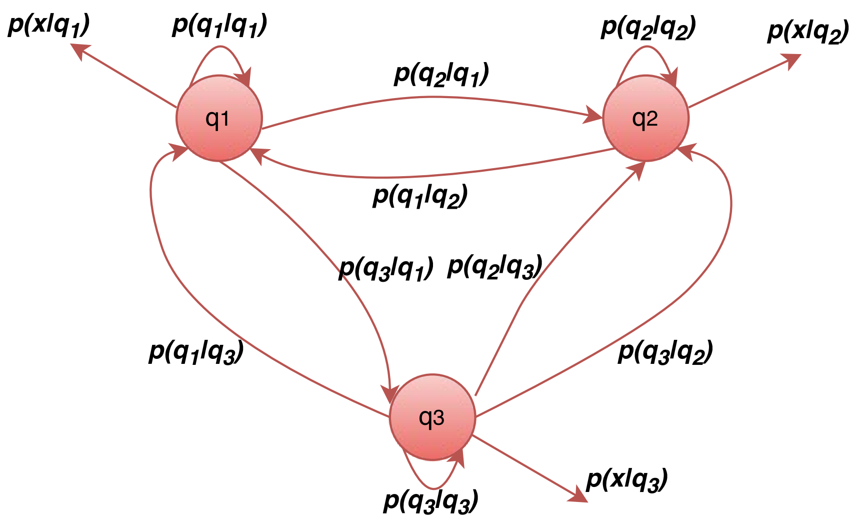

Figure 2.1 – A Hidden Markov Model

since given the current state, individual observations are independent. Also, the probability of a certain state sequence $\mathbf{q} \in Q_T$ is the following:

$$p(\mathbf{q}|\lambda) = p(q_1) \prod_{t=2}^{T} p(q_t|q_{t-1};\lambda) \tag{2.2}$$

Therefore, the likelihood of a certain observation sequence given the model is the summation over all possible state sequences that might have generated it:

$$p(\mathbf{x}|\lambda) = \sum_{\mathbf{q}\in Q_T} p(\mathbf{x}|\mathbf{q},\lambda)p(\mathbf{q}|\lambda) = \sum_{q\in Q_T} p(q_1)p(x_1|q_1) \prod_{t=2}^{T} p(x_t|q_t)p(q_t|q_{t-1};\lambda) \tag{2.3}$$

This summation over all possible state sequences is often not feasible. However, for an HMM, there are very well-known algorithms to:

• compute efficiently the probability of an observation sequence given a model

• find the most probable state sequence that could have generated a sequence of observations, or what is known as the decoding problem.

• find the parameters of a model that maximize the probability of observing a sequence of observations, or what is known as the training problem of an HMM.

The *forward-backward* algorithm described in [15] solves the problem of infeasible computation of Equation 2.3. The algorithm defines the forward probabilities as:

$$\alpha_t(s) = p(q_t = s, x_1^t), \qquad t = 1,\dots,T;\ s \in Q, \tag{2.4}$$

and the backward probabilities as:

$$\beta_t(s) = p(x_{t+1}^T | q_t = s), \qquad t = 1, \ldots, T-1;\ s \in Q, \tag{2.5}$$

where x_i^j defines the sub-sequence of observations from index i to index j, inclusive.
As shown in [15], the forward probabilities can be computed recursively as follows:

$$\begin{aligned} \alpha_1(s) &= p(q_1 = s)\cdot(x_1|s) \\ \alpha_t(s) &= p(x_t|q_t = s)\cdot \sum_{r\in Q} \alpha_{t-1}(r)p(q_t = r|qt-1 = r;\lambda) \end{aligned} \tag{2.6}$$

Similarly, backward probabilities are computed as follows:

$$\begin{aligned} \beta_T(s) &= 1 \\ \beta_t(s) &= \sum_{r\in Q} p(q_{t+1} = s|q_t = s;\lambda)p(x_{t+1}|q_{t+1} = r)\beta_{t+1}(r) \end{aligned} \tag{2.7}$$

It can be easily shown then that $p(\mathbf{x}|\lambda) = \sum_{s\in Q} \alpha_T(s)$.
On the other hand, the most probable state sequence $\mathbf{q}^* \in Q_T$ that could have generated a given observation sequence $\mathbf{x}$ can be found by solving the following maximization problem:

$$\mathbf{q}^* = arg\max_{\mathbf{q}\in Q_T} p(\mathbf{q}|\mathbf{x},\lambda) = arg\max_{\mathbf{q}\in Q_T} \frac{p(\mathbf{q},\mathbf{x}|\lambda)}{p(\mathbf{x})} \tag{2.8}$$

This can be solved using the Viterbi algorithm, described in [16]. Since $p(\mathbf{x})$ is the same for all state sequences, it consists of replacing the summation in equation 2.6 by a maximization and keeping track of the state r that maximizes the equation in each iteration.

Finally, training an HMM involves adjusting its parameters such that the likelihood of certain observation sequences in a training set is maximized. This can be done by an Expectation-Maximization (EM) approach ([17]), which computes first an objective function for a fixed HMM λ, and then re-estimates the parameters of the HMM by maximizing the objective function. For HMM, this iterative procedure is done by the Baum-Welch algorithm ([15, 18]). First the posterior probability of seeing state s at time t is computed (where α and β are as computed in equations 2.6 and 2.7):

$$p(q_t = s|\mathbf{x},\lambda) = \frac{\alpha_t(s)\beta_t(s)}{\sum_{r\in Q}\alpha_t(r)\beta_t(r)} \tag{2.9}$$

Also, the probability of seeing state s at time t and state r at time $t+1$:

$$p(q_t = s, q_{t+1} = r|\mathbf{x},\lambda) = \frac{\alpha_t(s)p(x_{t+1}|q_{t+1} = r)p(q_{t+1} = r|q_t = s,\lambda)\beta_{t+1}(r)}{\sum_{u\in Q}\alpha_t(u)\beta_t(u)} \tag{2.10}$$

Using equations 2.9 and 2.10, the parameters of the HMM can be updated to form a new HMM

$\hat{\lambda}$. The criterion to be maximized is the following:

$$Q(\lambda,\hat{\lambda}) = \sum_{\mathbf{q}\in Q_T} p(\mathbf{q}|\mathbf{x},\lambda)\log p(\mathbf{x},\mathbf{q}|\hat{\lambda}) \tag{2.11}$$

Maximizing this criterion guarantees an increase in the likelihood $p(\mathbf{x}|\lambda)$ ([15]).

2.2 HMM's for Speech Modeling and Recognition

The main benefit behind using hidden Markov models for speech modeling lies in their ability to perform the time alignment of the extracted feature vectors with the target labels. The maximum likelihood formulation of the parameter estimation allows the emission model of HMM's to act as a generative sequence model for speech features. One of the most common emission models used for HMM states are Gaussian Mixture Models (GMM's), where each observed feature vector $\mathbf{x}_t$ for a certain tied target label (i.e. HMM state) s_j is assumed to be generated from a Gaussian mixture, as follows:

$$p(\mathbf{x}_t|s_j) = \sum_{m=1}^{M} \pi_{jm}\mathcal{N}_{jm}(\mathbf{x}_t|s_j;\mu_{jm},\Sigma_{jm}), \tag{2.12}$$

where $\mathcal{N}(\cdot,\mu,\Sigma)$ is the multivariate Gaussian distribution of mean μ and covariance matrix Σ. The parameters π_{jm}, μ_{jm} and Σ_{jm} are the weight, mean and covariance matrix, respectively, of the m-th component of the Gaussian mixture model for state s_j. The parameters of the GMM can be computed in an EM approach that maximizes the log-likelihood of the observation feature vector $\mathbf{x}_t$. M is the number of Gaussian mixtures for state s_j; it is often set to 1 when training starts and then increased during the EM algorithm.

3 Deep Neural Networks

Neural networks were originally developed to model the activity of biological neurons, but then became popular methods for general pattern recognition. Recently, deep neural networks (DNN's) have showed a great success in acoustic modeling of speech. The multiple layers of nonlinear processing allows the neural network to learn very complicated decision boundaries between the HMM states. In this chapter, we give an overview of different neural network architectures and describe the very well-known hybrid setup of DNN's and HMM's which is an essential component of current ASR systems.

3.1 Multilayer Perceptrons

An artificial neural network (ANN) is a network of small processing units (analogous to neurons in a biological model), connected to each other by weighted links (resembling the strength of the connection between the neurons). An input of a specific dimensionality is provided to the network, and then the activation flows in the network along the weighted connections. ANN's can be divided into categories: those whose connections form cycles (often referred to as recursive, or recurrent, neural networks, to be discussed in Section 3.2), or those with acyclic connections (often referred to as feed-forward neural networks). The most common used form of feed-forward networks is the multilayer perceptron (MLP) [19], shown in Fig. 3.1.

As shown in the figure, input is presented at the input layer, propagates through the hidden layers and reaches the output layer. This is referred to as the forward pass of the network. An MLP, therefore, only defines a nonlinear function from the input to the output vectors, knowing the particular set of weight values currently characterizing the network. When the weight values are changed, a new function from input to output can be approximated.

3.1.1 Forward Pass

In the forward pass, each unit h in a hidden layer computes a weighted sum a_h of the units of the previous layer, $\mathbf{x}$. Then the activation function θ_h is applied to yield the output b_h of this

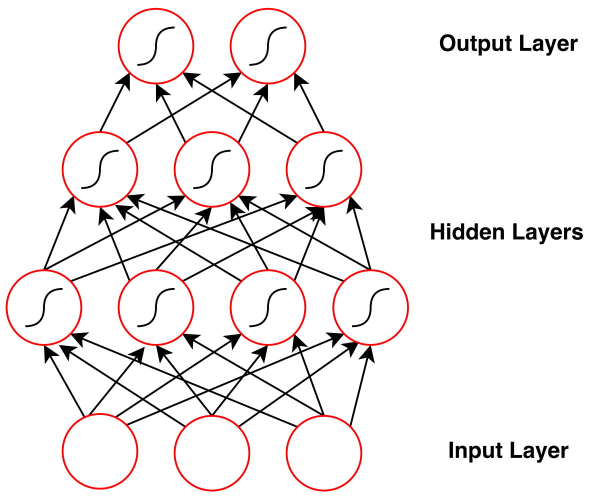

Figure 3.1 – **A multilayer perceptron**. The S-shaped curves denote the nonlinear activation functions.

hidden unit, as follows (w_{ij} is the weight from unit i to unit j and I is the number of unit in the previous layer):

$$a_h = \sum_{i=1}^{I} w_{ih} x_i \tag{3.1}$$

$$b_h = \theta_h(a_h) \tag{3.2}$$

The two most commonly used choices for the activation function θ_h are the hyperbolic tangent function

$$tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}, \tag{3.3}$$

and the logistic sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3.4}$$

A key property is that the activation functions are differentiable, so they can be used in network training using gradient descent.

The output of the neural network y is simply the activation of the units in the output layer. The number of units in the output layer as well the activation function used in this layer are task-dependent. For multi-class classification (as in speech recognition: classification of labels over speech frames), the *softmax* activation function is used for the output activations:

$$y_k = \frac{e^{a_k}}{\sum_{k'=1}^{K} e^{a_{k'}}} \tag{3.5}$$

3.1.2 Loss Functions

In principle, any differentiable loss function can be used for MLP training. Here, we will outline multi-class loss function that is derived using maximum likelihood. Treating each network output y_k as the probability of observing target label k and representing each target class z as a binary vector with all elements equal to zero except for the correct class k, then the target probabilities can be written as:

$$p(\mathbf{z}|\mathbf{x}) = \prod_{k=1}^{K} y_k^{z_k} \tag{3.6}$$

Therefore, the maximum likelihood loss for a single training example can be computed as:

$$\mathscr{L}(\mathbf{x},\mathbf{z}) = -\ln p(\mathbf{z}|\mathbf{x}) = -\sum_{k=1}^{K} z_k \ln y_k \tag{3.7}$$

3.1.3 Backward Pass

MLP's are usually trained to minimize the loss function using *gradient descent.* The basic idea is to adjust the weights of the network in the direction of the negative gradient. To efficiently compute the gradient, a technique known as backpropagation is used, known as the the backward pass of the network. Backpropagation is only a repeated application of the chain rule that allows to compute the partial derivative with respect to the network weights. In brief, for a multi-class classification problem, differentiating the loss function in Eq. 3.7 with respect to the network outputs gives:

$$\frac{\partial \mathscr{L}(\mathbf{x},\mathbf{z})}{\partial y_k} = -\frac{z_k}{y_k} \tag{3.8}$$

On the other hand, differentiating the softmax function gives

$$\frac{\partial y_{k'}}{\partial a_k} = \begin{cases} -y_k y_{k'} & k \neq k' \\ y_k(1-y_k) & k = k' \end{cases} \tag{3.9}$$

Since the softmax activation function depends on all network outputs, then the chain rule gives

$$\begin{aligned} \frac{\partial \mathscr{L}(\mathbf{x},\mathbf{z})}{\partial a_k} &= \sum_{k'=1}^{K} \frac{\partial \mathscr{L}(\mathbf{x},\mathbf{z})}{\partial y_{k'}} \cdot \frac{\partial y_{k'}}{\partial a_k} \\ &= y_k - z_k, \end{aligned} \tag{3.10}$$

where the fact that $\sum_{k=1}^{K} z_k = 1$ is used.

The chain rule is again re-applied, working backwards through the hidden layers, computing the gradient with respect to the hidden activations, as follows:

$$\begin{aligned} \frac{\partial \mathscr{L}(\mathbf{x},\mathbf{z})}{\partial a_h} &= \frac{\partial \mathscr{L}(\mathbf{x},\mathbf{z})}{\partial b_h} \frac{\partial b_h}{\partial a_h} \\ &= \theta'(a_h) \sum_{k=1}^{K} \delta_k w_{hk} \\ &= \theta'(a_h) \sum_{h' \in H_{l+1}} \delta_{h'} w_{hh'}, \end{aligned} \tag{3.11}$$

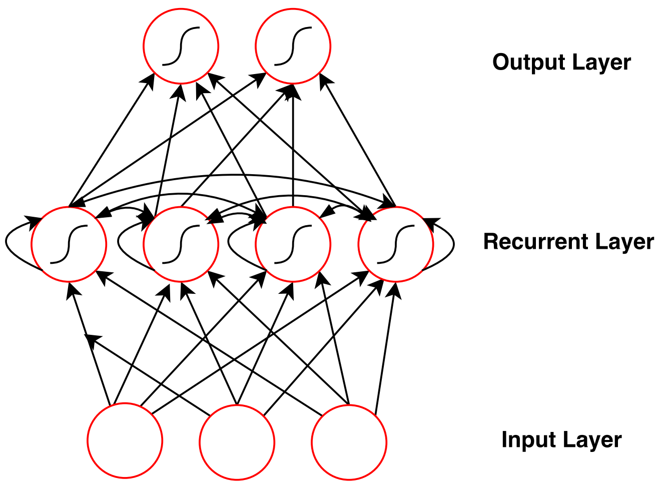

Figure 3.2 – A recurrent neural network

where H_{l+1} is the next hidden layer. Finally, the gradient with respect to the weights can be computed:

$$\frac{\partial \mathscr{L}(\mathbf{x},\mathbf{z})}{\partial w_{ij}} = \frac{\partial \mathscr{L}(\mathbf{x},\mathbf{z})}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} = b_i \cdot \frac{\partial \mathscr{L}(\mathbf{x},\mathbf{z})}{\partial a_j} \tag{3.12}$$

3.2 Recurrent Neural Networks

When we allow cyclical connections between the nodes of the hidden layers, we obtain a recurrent neural network (RNN). An RNN is shown in Fig. 3.2. The cyclical connections allow the RNN in principle to generate outputs based on the entire history of previous inputs, so a form of 'memory' can persist in the network's internal state parameters.

3.2.1 Forward Pass

The forward pass of an RNN is very similar to that of an MLP, except that activations arrive at the recurrent layer from the current external input and its own output activations from the previous timestep. So, for an input sequence of **x** of length T from an input layer with I input units to a recurrent layer with H hidden units, the activation of unit h at time t and the respective output are

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1} \tag{3.13}$$

$$b_h^t = \theta_h(a_h^t), \tag{3.14}$$

where x_i^t is the input i at time t, $b_{h'}^{t-1}$ is the output of the unit h' at time $t-1$ and $\theta_h(.)$ is the considered activation function.

3.2.2 Backward Pass

Knowing the partial derivatives of used the loss function $\mathscr{L}$ with respect to the network outputs, the partial derivatives with respect to the weights of the network can be computed using Backpropagation through time (BPTT) ([20]). Similar to standard backpropagation, BPTT is only a repeated application of the chain rule, except that the loss function is also influenced by the dependence on activations from previous timesteps. It can be shown that, for an RNN, Eq. 3.11 becomes:

$$\frac{\partial \mathscr{L}}{\partial a_h^t} = \theta'(a_h^t)\left(\sum_{k=1}^{K} \frac{\partial \mathscr{L}}{\partial a_h^t} w_{hk} + \sum_{h'=1}^{H} \frac{\partial \mathscr{L}}{\partial a_{h'}^{t+1}} w_{hh'}\right), \tag{3.15}$$

where a_h^t is the activation of hidden unit h at time t. Finally, since the same weights are used at each timestep, the gradients with respect to the network weights can be computed by summing over the whole input sequence:

$$\frac{\partial \mathscr{L}}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}}{\partial a_j^t} \cdot \frac{\partial a_j^t}{\partial w_{ij}} \tag{3.16}$$

3.2.3 Bidirectional Networks

Often, it is useful to incorporate future as well as previous context in your network. Bidirectional Recurrent Neural Networks (BRNN's) ([21]) achieve this concept. Each training sequence is provided, forwards and backwards, to two separate recurrent layers, both connected to the same output layer. Thus, the output can have complete past and future information for each frame in the input sequence. A bidirectional recurrent neural network, unfolded in time, is shown in Fig. 3.3.

The forward pass is similar to the case of a uni-directional network, except that the input sequence is provided to the two recurrent layers in opposite directions, so the output layer cannot be updated except when the whole input sequence is processed by the recurrent layers. Similarly, the backward pass proceeds as in BPTT for a uni-directional RNN, except that all the propagated gradients from the output layer are computed for the whole sequence, and then fed to the recurrent layers in opposite directions.

3.3 Long Short-Term Memory

The problem with standard RNN architectures is that the influence of the current input on the nodes of the recurrent layer is quite limited across time, i.e. the accumulated gradients either decay or explode exponentially through time. In literature, this is often referred to as the problem of *vanishing gradients*. The Long Short-Term Memory (LSTM) architecture comes to solve this effect [22]. An LSTM layer is made up of a set of recurrently connected subnets (known as memory blocks), each consisting of one or more nodes (known as cells) and three

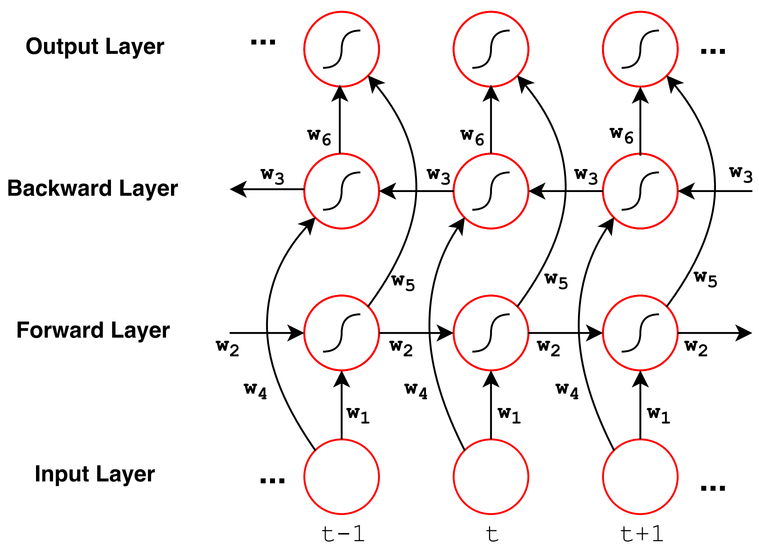

Figure 3.3 – A bidirectional recurrent neural network, unfolded in time

multiplicative units, referred to as the input, output and forget gates. An LSTM cell is shown in Fig. 3.4.

The multiplicative gates allow the LSTM cell to learn information over long periods of time. For example, if the input gate is closed (i.e. has an activation close to zero), the new inputs to the network will almost not be used to activate the cell, and therefore can be used much later in the sequence, when the output gate is opened. In brief, the forward pass for a single LSTM cell will consist of the following (w_{ij} denotes the weight from unit i to unit j, a_j^t the network input to unit j at time t, and b_j^t its activation. Subscripts ι, ϕ and ω refer to input, forget and output gates respectively. Subscript c denotes the memory cell, and s_c^t is its state at time t. $w_{c\iota}$, $w_{c\phi}$ and $w_{c\omega}$ denote *peephole* weights from the cell c to the input, forget and output gates. f is activation function of the gates. g and h are the activation functions of the cell input and output respectively):

The input gate controls the influence of the cell input on the cell's state:

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} s_c^{t-1} \tag{3.17}$$

$$b_\iota^t = f(a_\iota^t) \tag{3.18}$$

The forget gate controls the influence of the cell's previous state on the cell's current state:

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1} \tag{3.19}$$

$$b_\phi^t = f(a_\phi^t) \tag{3.20}$$

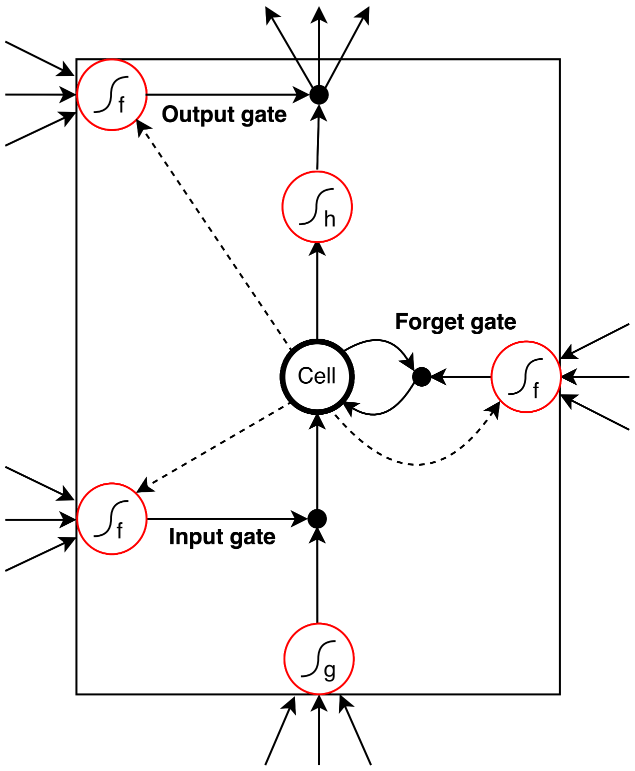

Figure 3.4 – An LSTM cell

The cell's current state is the sum of the scaled versions of its previous state and the cell input:

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1} \tag{3.21}$$

$$s_c^t = b_\phi^t s_c^{t-1} + b_\imath^t g(a_c^t) \tag{3.22}$$

The output gate controls the output of the LSTM unit:

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^t \tag{3.23}$$

$$b_\omega^t = f(a_\omega^t) \tag{3.24}$$

The output of cell is nothing but the scaled version of the cell state, after applying the activation function h to it:

$$b_c^t = b_\omega^t h(s_c^t) \tag{3.25}$$

In principle, the backward pass of LSTM's is very much similar to the BPTT pass of standard uni-directional RNN's, taking into account the new equations within an LSTM cell and the order in which the equations are calculated. For the sake of this Master thesis, we skip mentioning the equations involved in the backward pass of LSTM's, but we refer interested readers to [22].

3.3.1 Bidirectional Long Short-Term Memory

Bidirectional LSTM layers are nothing but an LSTM layer used in the same architecture as in bidirectional recurrent neural networks (Section 3.2.3). Bidirectional LSTM's allow to learn information from long-range contexts in both input directions, and are used extensively in our subsequent experiments.

3.4 Network Training

The most common method to train neural networks is gradient descent, where weights are adjusted by taking a small step in the direction of the negative gradient of the loss function:

$$\Delta w^n = -\alpha \frac{\partial \mathscr{L}}{\partial w^n}, \tag{3.26}$$

where Δw^n is the n^{th} weight update for a particular weight parameter of the network and $\alpha \in [0,1]$ is the learning rate. This is done for all weight parameters, and repeated for all the available training examples until, for example, the loss function is no more reduced for a number of epochs.

3.5 Hybrid NN/HMM Systems for Speech Recognition

In the hybrid NN/HMM approach ([23]), neural networks come to replace GMM's in modeling the emission probabilities of the hidden states in the HMM. The neural network provides the discriminative state posteriors $p(s|x_t)$, rather than the generative likelihoods $p(x_t|s)$. So Bayes' rule can be used to compute the state likelihoods:

$$p(x_t|s) = p(x_t)\frac{p(s|x_t)}{p(s)} \tag{3.27}$$

The introduction of NN's allows more improved modeling of a label duration and a richer approximation of the nonlinear function from the inputs to the outputs. On the other hand, HMM's are used to provide the temporal alignment of the label classifications provided by the neural network.

The two components can be trained independently, but often a combined optimization is used, where the alignments provided by the HMM are used to successively retrain the neural network.

4 Connectionist Temporal Classification

Neural Networks require to have a target label for every timestep in the input sequence. In speech recognition, this is not always feasible, because in most cases the alignment of the input frames with the output labels is not known. Two consequences follow from this fact: first, a segmentation of the training data should precede any network training in order to provide the network with the alignment, and second, post-processing of the posterior probabilities provided from the neural network is needed to capture the global aspects of the label sequence rather than the local classifications.

Connectionist Temporal Classification (CTC) comes to relax these requirements. Using CTC training, the network is allowed to make label predictions at any point in time, as long as the overall label sequence is correct. So no pre-segmented data is needed, and since CTC can output probabilities of the whole label sequence, no post-processing of the local posterior probabilities is required.

In this chapter, we discuss in detail CTC training and decoding.

4.1 Training

4.1.1 Preliminaries

CTC ([13]) is nothing but a cost function that is applied to the softmax output of a neural network, with one extra label added to the label dictionary. The extra label is referred to as the *blank* label. The blank label is there to model inter-label silences that do not correspond to any of the other target labels. As we go through this chapter, the role of the blank label will become clearer.

If the target labels for a certain task are drawn from an alphabet L, CTC defines an extended alphabet $L' = L \cup \{blank\}$. In what follows, we define y_k^t to be the activation of the network output $k \in L'$ at time t, for an input sequence $\mathbf{x}$ of length T. We also define L'^T to be the set of label sequences from L' of length T, modeling the possible output labels from the neural network. Knowing that the output probabilities from the neural network are conditionally independent given **x**, then the probability of a certain output label sequence $\pi \in L'$ (referred to as *path*) given **x** is:

$$p(\pi|\mathbf{x}) = \prod_{t=1}^{T} y^t_{\pi_t} \tag{4.1}$$

CTC then defines a many-to-one function $\mathscr{B} : L'^T \mapsto L^{\leq T}$, which maps a path π onto the set of possible labellings $L^{\leq T}$ (i.e. set of label sequences of size $\leq T$ and not considering the blank label). The function removes repeated labels, as well as the blanks from the input paths. For example, $\mathscr{B}(--a--ab) = \mathscr{B}(aaa-a-b) = aab$, i.e. a new label is outputted when the network switches from predicting a blank label to to predicting a target label, or when it switches from one target label to another. Since the paths are mutually exclusive, then the probability of a certain target label sequence can be computed as:

$$p(\mathbf{l}|\mathbf{x}) = \sum_{\pi \in \mathscr{B}^{-1}(\mathbf{l})} p(\pi|\mathbf{x}) \tag{4.2}$$

This mapping of multiple paths onto the same labelling is what allows CTC to deal with unsegmented data, because the network is allowed to predict target labels without knowing in advance at which time frame they exactly occur.

4.1.2 CTC Forward-Backward algorithm

Equation 4.2 suggests that computing the probability of a certain target label sequence can be problematic, because the number of paths π that can map to the target sequence **l** grows exponentially in T and $|\mathbf{l}|$. Luckily, the problem can be solved in a dynamic programming approach, analogous to the forward-backward algorithm of HMM's. The idea is to divide the computation of the probability of a labelling **l** into the iterative computation of the probabilities of the prefixes of the labelling.

First, to allow for blanks in the output paths, a blank label is inserted at the beginning, end and between every other pair of labels in the target sequence **l**, to generate a new modified label sequence **l'** of length $|\mathbf{l'}| = 2|\mathbf{l}| + 1$. In calculating probabilities of prefixes of **l'**, CTC allows transitions between blank and non-blank labels, as well as transitions between any pair of distinct non-blank labels. Namely, the forward variable $\alpha(t,u)$ is defined to be the probability of getting the first $u/2$ labels of **l** (i.e. $\mathbf{l}_{1:u/2}$) after seeing the all paths of length t:

$$\alpha(t,u) = \sum_{\pi \in V(t,u)} \prod_{i=1}^{t} y^i_{\pi_i}, \tag{4.3}$$

where $V(t,u) = \{\pi \in L'^t : \mathscr{B}(\pi) = \mathbf{l}_{1:u/2}, \pi_t = l'_u\}$ and $u/2$ is rounded down to an integer value. With this formulation in mind, it can be shown that the forward variables at time t can be computed recursively using the values at time $t-1$. For $t = 1$, all correct paths should start either with a blank label (b) or with first symbol label in **l** (l_1), so we get the following initial conditions:

$$\alpha(1,1) = y_b^1 \tag{4.4}$$

$$\alpha(1,2) = y_{l_1}^1 \tag{4.5}$$

$$\alpha(1,u) = 0 \qquad \forall u > 2 \tag{4.6}$$

For $t > 1$, the forward variables can be computed recursively as follows:

$$\alpha(t,u) = y_{l'_u}^t \sum_{i=f(u)}^{u} \alpha(t-1,u), \tag{4.7}$$

where

$$f(u) = \begin{cases} u-1 & \text{if } l'_u = \mathit{blank} \text{ or } l'_{u-2} = l'_u \\ u-2 & \text{otherwise} \end{cases} \tag{4.8}$$

It can be seen from the definition of the forward variables that the probability of the target label sequence **l** is the sum of the forward variables at time T, with and without the final blank:

$$p(\mathbf{l}|\mathbf{x}) = \alpha(T,|\mathbf{l'}|) + \alpha(T,|\mathbf{l'}|-1) \tag{4.9}$$

The CTC forward pass is shown in Fig. 4.1.

Similarly, backward variables $\beta(t,u)$ are defined to be the probabilities of all paths starting at $t+1$ that allow to get **l** after being appended to any path contributing to $\alpha(t,u)$, i.e.

$$\beta(t,u) = \sum_{\pi \in W(t,u)} \prod_{i=1}^{T-t} y_{\pi_i}^{t+i}, \tag{4.10}$$

where $W(t,u) = \left\{\pi \in L'^{T-t} : \mathscr{B}(\hat{\pi} + \pi) = \mathbf{l} \ \forall \hat{\pi} \in V(t,u)\right\}$.

This formulation allows the following initialization and recursion for the backward variables:

$$\beta(T,|\mathbf{l'}|) = \beta(T,|\mathbf{l'}|-1) = 1 \tag{4.11}$$

$$\beta(T,u) = 0, \qquad \forall u < |\mathbf{l'}|-1 \tag{4.12}$$

$$\beta(t,u) = \sum_{i=u}^{g(u)} \beta(t+1,i) y_{l'_i}^{t+1} \tag{4.13}$$

where

$$g(u) = \begin{cases} u+1 & \text{if } l'_u = \mathit{blank} \text{ or } l'_{u+2} = l'_u \\ u+2 & \text{otherwise} \end{cases} \tag{4.14}$$

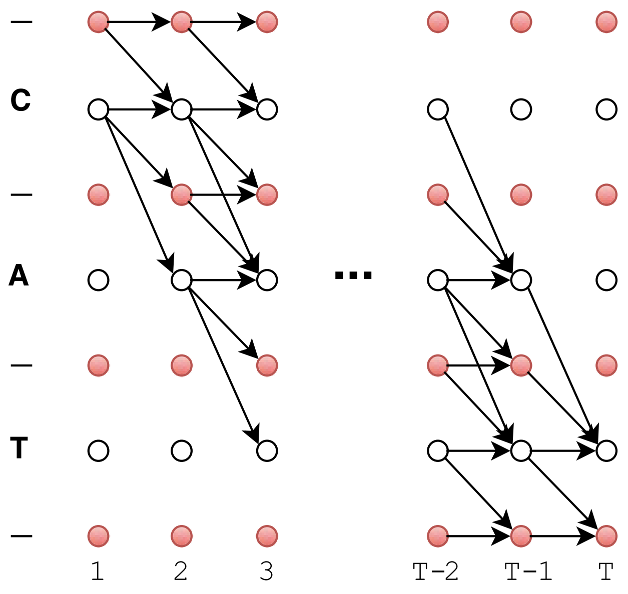

Figure 4.1 – **CTC forward-backward algorithm**. The filled circles represent the blank labels, and the empty circles represent labels. Arrows indicate allowed transitions.

4.1.3 Cost function

The CTC cost function $\mathscr{L}(S)$ is defined to be the negative log-probability of correctly labeling all training examples in a training set S:

$$\mathscr{L}(S) = -\ln \prod_{(\mathbf{x},\mathbf{z})\in S} p(\mathbf{z}|\mathbf{x}) = -\sum_{(\mathbf{x},\mathbf{z})\in S} \ln p(\mathbf{z}|\mathbf{x}) \tag{4.15}$$

Similarly, we define the loss function over one training example $(\mathbf{x}, \mathbf{z}) \in S$ as

$$\mathscr{L}(\mathbf{x},\mathbf{z}) = -\ln p(\mathbf{z}|\mathbf{x}) \tag{4.16}$$

Since the loss function is differentiable, gradient descent can be used along with backpropagation through time to update the network weights. From the definition of forward and backward variables in Section 4.1.2, it can be found that

$$\alpha(t,u)\beta(t,u) = \sum_{\pi\in X(t,u)} \prod_{t=1}^{T} y^t_{\pi_t}, \tag{4.17}$$

where $X(t,u) = \{\pi \in L'^T : \mathscr{B}(\pi) = \mathbf{z}, \pi_t = \mathbf{z'}_u\}$, i.e. the set of paths that map to $\mathbf{z}$ passing through $\mathbf{z'}_u$ at time t. From Eq. 4.1, we can get

$$\alpha(t,u)\beta(t,u) = \sum_{\pi\in X(t,u)} p(\pi|\mathbf{x}) \tag{4.18}$$

Therefore, for any t, we can get probability of label $\mathbf{z}$ by summing over all u to get

$$p(\mathbf{z}|\mathbf{x}) = \sum_{u=1}^{|\mathbf{z'}|} \alpha(t,u)\beta(t,u) \tag{4.19}$$

Deriving the gradients with respect to the network softmax output y_k^t from Eq. 4.16, we get

$$\frac{\partial \mathscr{L}(\mathbf{x},\mathbf{z})}{\partial y_k^t} = -\frac{\partial \ln p(\mathbf{z}|\mathbf{x})}{\partial y_k^t} = -\frac{1}{p(\mathbf{z}|\mathbf{x})}\frac{\partial p(\mathbf{z}|\mathbf{x})}{\partial y_k^t} \tag{4.20}$$

We now focus on computing the partial derivative $\frac{\partial p(\mathbf{z}|\mathbf{x})}{\partial y_k^t}$. We define the set $C(\mathbf{z},k)$ of positions where label k occurs in **z'**, i.e. $C(\mathbf{z},k) = \{u : z'_u = k\}$, which can be an empty set. To differentiate p(**z**|**x**) with respect to y_k^t, we can only consider paths that pass through label k at time t. From Eq. 4.17, we can see that

$$\frac{\partial \alpha(t,u)\beta(t,u)}{\partial y_k^t} = \begin{cases} \frac{\alpha(t,u)\beta(t,u)}{y_k^t} & \text{if } k \text{ occurs in } \mathbf{z'} \\ 0 & \text{otherwise} \end{cases} \tag{4.21}$$

Therefore, differentiating Eq. 4.19, we get

$$\frac{\partial p(\mathbf{z}|\mathbf{x})}{\partial y_k^t} = \frac{1}{y_k^t}\sum_{u\in C(\mathbf{z},k)} \alpha(t,u)\beta(t,u) \tag{4.22}$$

Finally, substituting this into Eq. 4.20, we get:

$$\frac{\partial \mathscr{L}(\mathbf{x},\mathbf{z})}{\partial y_k^t} = -\frac{1}{p(\mathbf{z}|\mathbf{x})y_k^t}\sum_{u\in C(\mathbf{z},k)} \alpha(t,u)\beta(t,u) \tag{4.23}$$

Since y_k^t is the output of the softmax function applied to the network output activations, it can be easily shown that the gradients of the CTC cost function with respect to the output activations a_k^t are as follows:

$$\frac{\partial \mathscr{L}(\mathbf{x},\mathbf{z})}{\partial a_k^t} = y_k^t - \frac{1}{p(\mathbf{z}|\mathbf{x})}\sum_{u\in C(\mathbf{z},k)} \alpha(t,u)\beta(t,u), \tag{4.24}$$

which is the error signal back-propagated during CTC training and used to update the weights, as discussed in Section 3.4.

4.2 Decoding

Decoding is the task of choosing the most probable labelling **l*** knowing the corresponding input feature vector **x**, after having trained the network, i.e. choosing:

$$\mathbf{l}^* = \arg\max_{\mathbf{l}} p(\mathbf{l}|\mathbf{x}) \tag{4.25}$$

Two methods for computing this maximization problem is presented here.

4.2.1 Best-path Decoding

Best-path decoding assumes that the most probable label **l*** corresponds to the most probable path π^*, i.e.

$$\mathbf{l}^* = \mathscr{B}(\pi^*), \tag{4.26}$$

where $\pi^* = \arg\max_{\pi} p(\pi|\mathbf{x})$. π^* can be easily found by concatenating the most probable output labels at each timestep. However, best-path decoding does not guarantee finding the most probable labelling. This can be argued with an example: for a sequence of two timesteps ($T = 2$) and an alphabet with single label a, if $y^1_{blank} = y^2_{blank} = 0.6$ and $y^1_{'a'} = y^2_{'a'} = 0.4$, then the predicted label under best-path decoding would be a blank for both timesteps, which will map to a blank for the predicted label. However, if both timesteps are considered, we see that $p(\mathbf{l} = \text{blank}) = p(\text{blank}, \text{blank}) = 0.6 * 0.6 = 0.36$ and $p(\mathbf{l} = \text{'a'}) = p(\text{'a'}, \text{'a'}) + p(\text{'a'}, \text{blank}) + p(\text{blank}, \text{'a'}) = 0.4 * 0.4 + 0.4 * 0.6 + 0.6 * 0.4 = 0.64$. Therefore, the labelling $\mathbf{l}$ = 'a' has a higher probability. Best-path decoding fails in this case.

4.2.2 Prefix-search decoding

Prefix-search decoding looks at prefixes of labels, rather than whole labels. Prefixes are grown by appending the most probable label until it is more probable that the prefix ends. In brief, the algorithm can be described in the following steps:

• Maintain a list of growing prefixes, initialized to the empty prefix. Along with each prefix, we store its probability.

• Find the most likely prefix, and we consider each possible extension of this prefix.

• Compute the probability of each possible extension, as well as the probability of terminating the prefix and ending the string.

• If terminating the prefix has higher probability than extending this or any other prefix, then this prefix is our decoding result!

• Otherwise, extend the prefix and store the new set of probabilities, instead of the probability of the old, shorter prefix.

• Iterate until we find our decoding.

For more detailed explanation of CTC prefix-search decoding, we refer interested readers to Algorithm 7.1 in [24].

Prefix-search decoding converges to the right decoding label if given enough time, but the number of observed prefixes increases exponentially with the input sequence length. However, Graves et al. use a heuristic in [13], in which they cut the network output sequence into chunks that are so likely to start and end with a blank, and then run prefix-search decoding on the smaller chunks, concatenating the results at the end.

4.3 Discussion

4.3.1 Role of the Blank Symbol

In the original formulation of CTC, the idea of the blank symbol was not used, and the function $\mathscr{B}(\pi)$ was simply π with the repeated symbols removed. However, this caused two problems: first, label sequences should have no two consecutive labels in a row, because the transitions were allowed only when π passed between different labels, and second, the network had to continue to predict one label before switching to another label, which can be a burden in tasks like speech recognition, where there are often pauses between words in an utterance. Therefore, the blank symbol gives the option to mark these pauses as unlabelled data (or namely, align them with the blank state).

The key point in the way CTC uses its blank symbol is that although it is inserted between every other label in the target label sequence, going to the blank state is optional, i.e. the network learns whether it should go to the blank state or not, or in other terms, the network learns the alignment. This is critical to the *"peaky"* behavior obtained in CTC training, as discussed in the next chapter.

4.3.2 Comparison with HMMs

In this subsection, we briefly discuss the main differences between CTC-trained networks and the hybrid NN/HMM approach discussed in Section 3.5. First, CTC-trained networks are discriminative, providing posterior label probabilities, while HMMs are generative, providing unnormalized likelihoods of the label sequences. Although generative approaches are useful for generating synthetic data and using already trained systems, they tend to fail when the prior data distribution is hard to determine. Discriminative methods, on the other hand, tend to perform well in classification tasks, because they focus entirely on predicting the correct labels.

Second, in CTC-trained networks, there are no prior or transition probabilities to be learned between the different states. Apart from the allowed transitions between states, CTC uses a simple topology where each label is modeled as a single state, with an *optional* blank state between labels, without the need of aligning the input frames with the target labels, since, in the formulation of the CTC forward and backward variables, all possible alignments that correspond to a certain label sequence are considered. On the other hand, HMMs are constrained to segment the input sequence in order to find the sequence of hidden states, which is not always feasible, because the precise boundary between the states can be ambiguous.

Finally, and more critically, HMMs, unlike CTC-trained networks, assume that the probability of a certain observation is dependent only on the current hidden state, whereas CTC, being well-suited to RNNs, allow predictions to be based on the whole input sequence. Therefore, HMMs fail to model long range contextual dependencies between labels. This can be mitigated by using triphone states for phoneme recognition; however, increasing the number of states increases exponentially the number of learned parameters, which in turn increases the amount of required training data.

5 Experimental Results

In this chapter, we give an overview of our experimental setup and results, in which we tried to model the robustness of CTC training, in comparison to the hybrid NN/HMM approach. Before , we provide a brief explanation of the different platforms that we used in our experiments.

5.1 Experimental Setup

5.1.1 Kaldi

Kaldi is a free, open-source toolkit for speech recognition ([25]). It is written in *C++* and released under the Apache License v2.0. It is provided with many executable programs and example scripts with complete recipes for building speech recognition systems from widely available databases such as the ones provided by the Linguistic Data Consortium (LDC). The recognition systems implemented in Kaldi are based on finite-state transducers (FSTs), compiled using the OpenFst toolkit. Kaldi also includes extensive linear algebra support, using routines from the standard "Basic Linear Algebra Subroutines" (BLAS) and "Linear Algebra PACKage" (LAPACK) packages.

In our work, we use Kaldi for our experiments on the baseline DNN/HMM framework.

5.1.2 Lasagne

Lasagne is a lightweight library to build and train neural networks. The library is built over Theano and supports network training on GPU, requiring an NVIDIA GPU with CUDA support. Lasagne also requires Python 2.7 to 3.4 to run, along with "numpy" and BLAS libraries.

Lasagne is used in our work for the experiments that involved CTC training, where we used a public implementation of CTC over Theano ([26]).

5.1.3 TIMIT Corpus

All our experiments were done on the TIMIT corpus, which is a corpus of lexically and phonemically transcribed speech of 630 different American English speakers of different sexes and dialects ([27]). It was commissioned by the Defence Advanced Research Projects Agencies (DARPA) and worked on many sites including Texas Instruments (TI) and Massachusetts Institute of Technology (MIT), and it includes time-aligned phonetic and word transcriptions as well as a 16-bit, 16kHz speech waveform file for each utterance. In its training set, TIMIT includes a total of 3696 utterances, a total of 400 utterances in its development set and 192 utterances in its test set.

5.1.4 Evaluation Criterion

For temporal classification tasks, such as speech recognition, the *segment error rate*, or the normalized error in each segment frame, is not very meaningful, and sometimes not applicable, when the boundaries of each segment are not known. Therefore, we use what is known as the *label error rate*, which is defined as follows:

$$LER(h, S') = \frac{1}{|S'|} \sum_{(\mathbf{x},\mathbf{z})\in S'} \frac{ED(h(\mathbf{x}), \mathbf{z})}{|\mathbf{z}|}, \tag{5.1}$$

where S' is the testing set used, $h(.)$ is the classifier function of the input sequence and ED($\mathbf{p}$,$\mathbf{q}$) is the *edit distance* between two sequences **p** and **q**, i.e. the minimum number of insertions, deletions and/or substitutions that are required to turn **p** into **q**. $ED(\mathbf{p}, \mathbf{q})$ can be computed in $O(|\mathbf{p}||\mathbf{q}|)$ time using the Wagner-Fischer algorithm. As shown, the edit distance is normalized by the length of the target label sequence **z**. The label error rate is usually multiplied by 100 to be interpreted as a percentage, although, by definition, it can take values larger than 100.

The label error rate is referred to based on the type of label in hand. In our work, as we have used a phonemically transcribed corpus TIMIT, we measure our performance using the *phoneme-error-rate* (PER) criterion.

5.2 Experiments over TIMIT

In this section, we present the results of our experiments that are intended to show the robustness of CTC training against noisy data. In [11], authors from Baidu test their "Deep Speech" model on a testing set that includes 100 noisy and 100 noise-free utterances from 10 speakers. The noisy data was synthesized by mixing different noise sources (TV, radio, crowded cafeteria, restaurant noise, ...) with utterance texts from web search queries and text messages. Authors use a bidirectional recurrent neural network trained with more than 7000 hours of speech data from published corpora by the Linguistic Data Consortium (LDC) (namely, WSJ, Switchboard and Fisher) as well as an internally-synthesized dataset of read speech from 9600 speakers. A summary of the datasets used in [11] is shown in Table 5.1.

Table 5.1 – Summary of the datasets used to train Deep Speech

Dataset	Type	Hours	Speakers
WSJ	read	80	280
Switchboard	conversational	300	4000
Fisher	conversational	2000	23000
Baidu	read	5000	9600

The authors report a 37.44% relative improvement of their model over state-of-the-art system on the noisy set; however, the results on the clean set were very much comparable. Apart from the huge amount of training speech data, "Deep Speech" uses CTC as its training loss function. And this inspired our work: trying to study the effect of CTC training on the robustness of the speech recognizer. And we present out results in what follows.

5.2.1 DNN/HMM training

Using the Kaldi recipe, a network of 6 fully connected hidden layers and 512 nodes per layer is trained over the training set of the TIMIT corpus, which contains around 5 hours of speech data. The mel-frequency cepstral coefficients (MFCCs) are the extracted feature vectors used during training, which are 13-dimensional vectors extracted from the short-term log-power spectrum of the speech signal. The output layer has dimensionality of 48 (48 phonemes modeled in TIMIT). So the architecture has around 1.3 million total weight parameters to be optimized. Training is done over 30 epochs, with an initial learning rate of 0.008, that is halved after 20 epochs. The system achieves state-of-the-art performance over TIMIT, with a phoneme error rate of 24.9%.

5.2.2 CTC Training

Using Lasagne, a network consisting of 2 bidirectional LSTM layers, with 832 cells in each of the forward and backward directional layers, followed by 2 fully connected layers, each containing 512 nodes, is built and used to test our hypothesis over TIMIT. MFCCs are also the extracted feature vectors used in this experiment. The output layer has a size of 49 (48 phonemes of TIMIT + 1 blank symbol). Therefore, the total number of weights using this architecture is around 17.8 million parameters. This architecture was used because it gave the best results on TIMIT development set over a couple of other tried architectures. Training is done for 80 epochs, starting with an initial learning rate of 10^{-4} for the first 20 epochs, and then updating the rate depending on the network performance on the development set. RMSprop is also used to adapt the learning rate based on the computed gradient values.

A key point in CTC training is the observed peaks through the epochs (see Fig. 5.1). As shown, the predictions are mainly blanks, with peaks of certain phonemes emerging at specific timesteps. In the top left plot (epoch 1), we see the outputs before training. As no weights

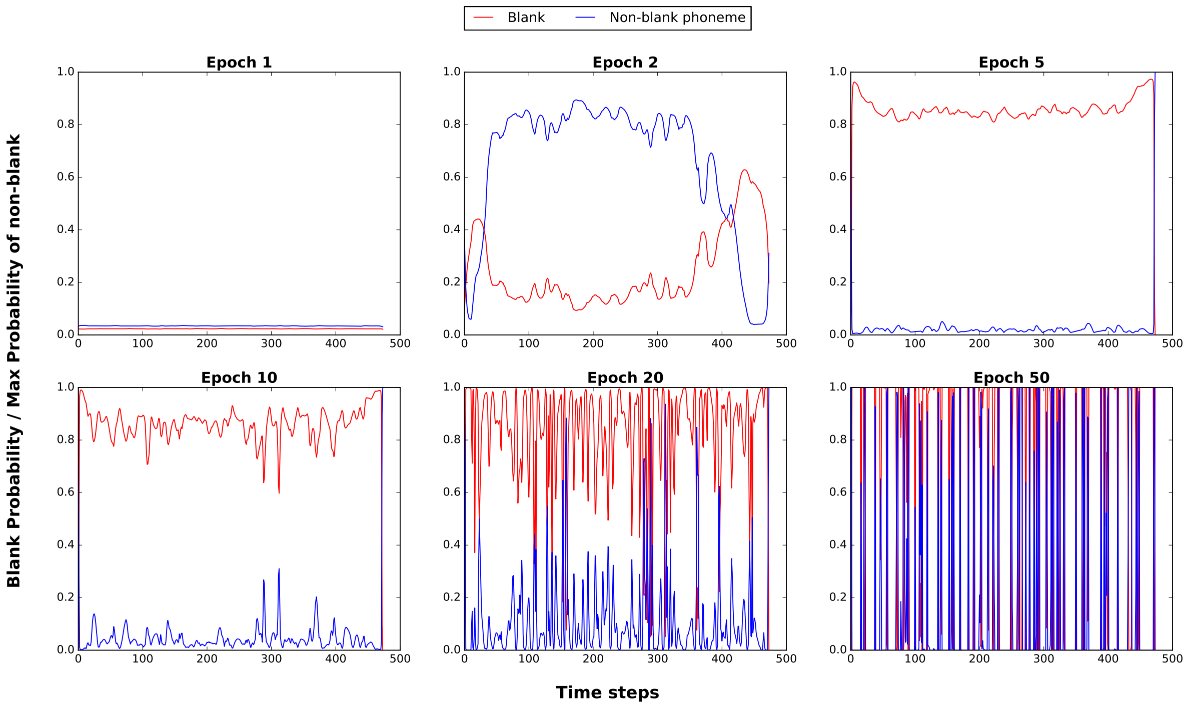

Figure 5.1 – **CTC network classifying a speech signal**. The red curve shows the probability of the blank symbol, as outputted from the neural network output softmax function, with respect to the corresponding time frame index for a certain training utterance in the corpus. The blue curve shows the maximum probability of any other non-blank phoneme. Different graphs show the variations for different epochs throughout CTC training.

have been adjusted at the time, the probabilities are random (almost equal to $\frac{1}{\#\text{phonemes}+1}$), due to the effect of random initialization of the network. As training proceeds, the output probabilities for the blank symbol increase, until the posterior probability is higher than any other phoneme (epoch 5). At this point, the network is only predicting blank labels. As training advances more, the probabilities of phoneme labels start to increase, and peaks emerge at certain locations in the input sequence.

This is predictable in some sense, considering the forward-backward procedure of CTC. At first, all paths are equally likely, and thus the posterior probabilities of the labels would follow, more or less, a uniform distribution. In the CTC graph however, there is a blank symbol between every other label. So the sum of the blank label posteriors for a specific timestep will be higher than that of any other non-blank posterior, causing the backpropagated error to adjust the weights in the direction that favors any path with many blanks in subsequent forward-backward computations of the cost.

Predicting only blanks will penalize more the cost function because of the low posterior probabilities computed for non-blank labels. Because of the formulation of CTC and the usage of the $\mathscr{B}(.)$ function, as described in Section 4.1.1, the network has to find only one location

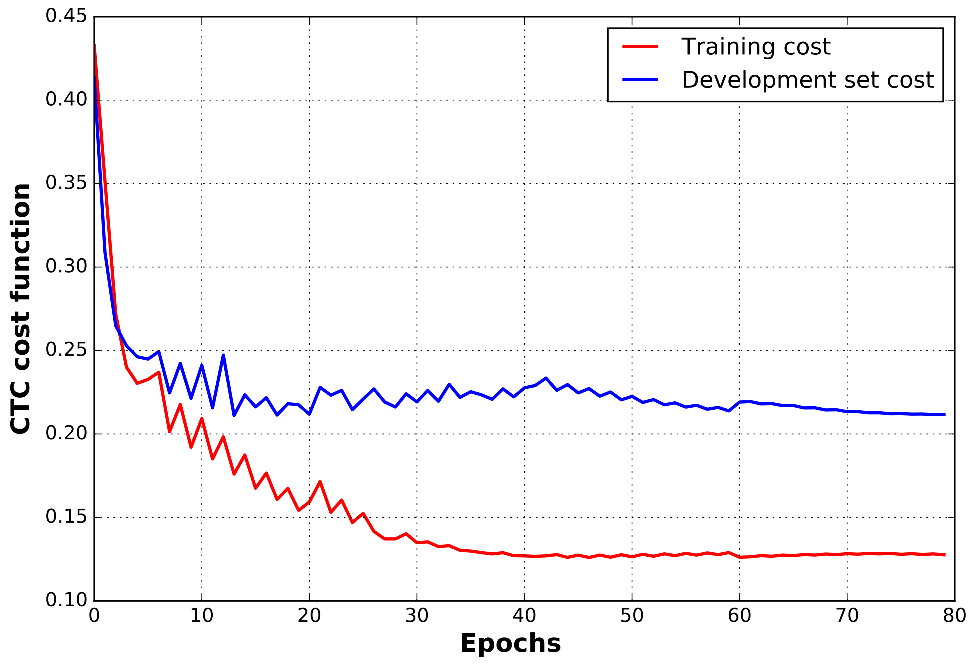

Figure 5.2 – CTC cost function over training epochs for training and development sets

to predict each non-blank label, in order to "*hop*" from one sequence of blanks to another, because any valid path should pass by non-blank labels. So the posterior probabilities of the non-blank labels at these locations will increase, causing the computed gradients to be high at these positions, and the network will learn to predict phoneme labels at specific locations. This path will have a very much higher probability, so the training algorithm will not consider any other segmentation.

It should be noted that, as reported in [28], the same *peaky* behavior of CTC training can be achieved when using an MLP network, instead of the LSTM network, which emphasizes the fact that this behavior is not an effect of the recurrence in the LSTM network nor the single-state phoneme model, but a result from the interaction between the blank symbol and the forward-backward implementation of CTC training.

Another interesting graph to look at is the learning curve of the network over the training and development sets, shown in Fig. 5.2. It can be seen that after 40 epochs, the mean cost function over the training set converges. The fluctuations are due to adaptive learning rate updates that cause the network to roam around an optimum point.

Using best-path decoding of CTC, our network with the proposed architecture achieves a phoneme error rate of 26.9%, which is very much comparable to the case of the DNN/HMM training (24.9%). Graves et al. report in [13] a phoneme error rate of 31.47% using best-path decoding, so our system achieves a 14.5% relative improvement over the system in [13].

Table 5.2 – Phoneme error rates for different SNR levels

SNR levels (dB)	DNN/HMM Hybrid	CTC
-10	52.1%	54.83%
-3	44.2%	47.04%
0	40.3%	43.22%
3	38.1%	41.33%
10	31.1%	34.84%
100	26.5%	32.3%

5.2.3 Noisy Data Experiments

Similar to the approach followed in [11], we collect noise sources from public databases that include noise clips recorded in noisy environments like in a car, restaurant, airport, ... The number of unique noise samples should be proportional to the size of the training data; otherwise, repeating the same noise clip over and over can allow the recurrent network to *learn* the noisy track and "subtract" it from the training samples. Therefore, we synthesize our own noisy data, by mixing a random number of these noise clips together, and then mixing our synthesized noise with the clean data that we have from TIMIT. In order to run experiments for different levels of noise, we modify the RMS amplitude of the synthesized noise clips in order to achieve signal-to-noise ratio (SNR) levels of -10dB, -3dB, 0dB, 3dB, 10dB, and 100dB.

We repeat the previous two experiments for the same proposed architectures and evaluate the performance of the DNN/HMM framework in comparison to CTC training. Results are shown in Table 5.2.

First, the table shows the need for robustness in current ASR systems. When the power of the speech signal is twice the power of the surrounding noise (SNR = 3dB), the phoneme error rate is 13.2% higher than the case of a noiseless environment (38.1% when SNR = 3dB, whereas 24.9% for the case of clean data for the DNN/HMM framework results), which is ≈ 53% relative degradation of the performance. This indicates the need to design more robust speech recognizers.

Also, CTC training, as shown, gives a bit higher phoneme error rates over TIMIT than the traditional DNN/HMM system; although it gets closer for noisier data (SNR = -10dB), results are still on a par with the hybrid DNN/HMM baselines. We address this issue in the following section.

5.3 Discussion

5.3.1 Benefits of CTC Training

A distinctive feature of CTC training is its use of context-independent label sequences, where each target label is modeled as a single state, with an additional optional *blank* state. This

Table 5.3 – Comparison between single-state phoneme training and CTC training

SNR levels (dB)	Monophone	CTC
-10	58.2%	54.83%
-3	54.7%	47.04%
0	50.5%	43.22%
3	48.0%	41.33%
10	41.6%	34.84%
100	35.6%	32.3%
∞ (Clean Data)	32.2%	26.9%

drastically reduces the number of states from thousands of senones to tens of labels (either phonemes or characters) and allows to reduce the complexity of the decoding procedure. Authors in [29] propose a generalized decoding method based on weighted finite-state transducers (WFSTs) that incorporates CTC labels, lexicons and language models to compose a comprehensive search graph. The paper shows that this decoding implementation is 3.2× faster than that of a DNN/HMM, and significantly smaller in storage size. This allows to keep more various hypotheses during beam search decoding, which would enhance performance.

In this sense, a useful experiment to run is a comparison between monophone training of GMM/HMM's and CTC training, which would acquire comparable decoding complexities, due to the single-state representation of phonemes. Using Kaldi, similar experiments as before are held for GMM/HMM training using monophone states, and results are shown in Table 5.3, along with the already-obtained results for CTC training. It can be seen that, for the same order of decoding complexity, CTC-trained networks outperform the GMM/HMM hybrid approach when phonemes are represented as single states.

5.3.2 Influence of Training Data Size

The main characteristic of using TIMIT as the training corpus goes back to its relatively small size (≈ equivalent to 5 hours of speech data, in comparison to more than 7000 hours in "Deep Speech"), which directly impacts the training time, and allows us to perform more experiments within the limited time frame of the Master thesis. However, our hypothesis is that CTC "*shines*" more when large amounts of training data are available. Since the segmentation of the target labels with respect to the input frames is unknown prior to CTC training, the network has to *learn* this alignment. Knowing that the only kind of input-output correlation that the network can make for a given training utterance is between the *sequence* of the target labels and the *sequence* of feature vectors (since no mapping between input vectors and target labels is given a priori), a lot of these training utterances should be provided to the network.

In order to test this hypothesis, we perform the same previous experiments over a smaller subset of TIMIT (namely, 10% of the training data of TIMIT, equivalent to 370 training utterances) and see how worse things can get for the CTC-trained network in comparison with the

Table 5.4 – PER results when using a subset of TIMIT

SNR levels (dB)	Monophone	DNN/HMM Hybrid	CTC
-10	64.6%	65.1%	72.42%
-3	59.7%	61.9%	66.176%
0	56.8%	58.3%	64.25%
3	53.2%	54.4%	62.27%
10	47.3%	48.3%	57.73%
100	42.6%	44.1%	53.31%
∞ (Clean Data)	39.4%	42.7%	49.98%

DNN/HMM hybrid. The results are shown in Table 5.4, along with the corresponding results for monophone training.

A first direct observation is that monophone training of GMM/HMM's now outperforms both the DNN/HMM hybrid and the CTC-trained network. This behavior of single-state models is predictable for small amounts of training data, because there are less classes/states to discriminate for the network and more training samples per class on average.

However, for the CTC-trained network, results are way worse than the traditional hybrid approach. The average degradation in performance (i.e. the average of the difference between PER's) is 7.33% (equivalently, 14.32% of average *relative* degradation), whereas, in comparison with the case when the whole training data of TIMIT was used (Table 5.2), the average degradation was 3.32% (equivalently 9.9% of average relative degradation). Also, monophone-trained GMM/HMM's outperform in this case the CTC-trained network, which is not the case when all the training data of TIMIT was used (Table 5.3). This is a direct effect of the size of the training data, which supports our claim that CTC-trained networks are particularly suited for large amounts of training data; otherwise, if not available, the DNN/HMM hybrid approach is preferred. This is also supported by results in [29], where authors report that their CTC-trained models outperform the hybrid systems on large-sized datasets (namely, Switchboard with $\approx$ 300 hours of speech data), whereas the hybrid approach achieves better results on smaller datasets (Wall Street Journal with $\approx$ 80 hours of speech data).

5.3.3 CTC Robustness

The main obstacle that faces speech recognition is the variability of the speech input features with respect to the target labels pronounced, i.e. a certain phoneme in two different utterances will not have the same feature vectors, since the extracted features are highly dependent on existing inter- and intra-speaker variations, as described in Chapter 1.

For noisy speech recognition applications, this variability in the input data is even higher, due to noise presence. Therefore, having forced alignments in a DNN/HMM framework "confuses" the network, because it is highly probable that noisy feature frames (ones that are not representative of any target label) are mapped to a specific label. CTC-trained networks, on the other hand, always have the option to map "unclear" frames to the blank symbol, which causes less of a confusion in the network learning. That's why we expect CTC to outperform the traditional hybrid approach when dealing with noisy data. Yet, large corpora shall be used in CTC training, as discussed before.

Conclusions

In this Master thesis, we tried to model the suitability of CTC-trained networks for achieving more robust speech recognition systems, compared to the traditional DNN/HMM hybrid approach. As discussed in Chapter 5, our conclusions can be summarized in two main points:

• CTC is well-suited for large amounts of training data. As less amounts of information is given to the CTC-trained network compared to framewise training (i.e. the alignment of input features with target labels is not given), a larger training set is required for the network to *learn* this alignment. Once available, CTC-trained networks are expected to outperform the hybrid approach, especially for noisy data, where the confusion between input-output correlations is higher than the case of clean data.

• Due to its single-state modeling of target labels, CTC allows to achieve a faster decoding, compared to the context-dependent triphone-state hybrid approach. The decoding graphs are also smaller in size, which saves disk space for storing the graphs and allows to keep a larger number of hypotheses in beam-search decoding.

References

[1] T. Yoshioka and M. Gales, "Environmentally robust ASR front-end for deep neural network acoustic models," in *Computer Speech and Language, ELSEVIER* (31, ed.), pp. 65–86.

[2] A. Acero, "Environmental robustness in automatic speech recognition," in *ICASSP*, 1990.

[3] L. Deng, A. Acero, J. Li, and J. Droppo, "High-performance robust speech recognition using stereo training data," in *ICASSP*, 2001.

[4] D. Yu, L. Deng, J. Droppo, J. Wu, Y. Ging, and A. Acero, "A minimum-mean-square-error noise reduction algorithm on mel-frequency cepstra for robust speech recognition," in *ICASSP*, pp. 4041–4044, 2008.

[5] D. Gelbart and N. Morgan, "Double the trouble: Handling noise and reverberation in far-field automatic speech recognition," in *Proceedings of International Conference on Spoken Language Process*, pp. 2185–2188, 2002.

[6] D. Gelbart and N. Morgan, "Evaluating Long-term Spectral Subtraction for Reverberant ASR," in *ASRU*, 2001.

[7] X. Feng, Y. Zhang, and J. Glass, "Speech Feature Denoising and Dereverberation via Deep Autoencoders for Noisy Reverberant Speech Recognition," in *ICASSP*, 2014.

[8] B. Li and K. C. Sim, "An ideal hidden-activation mask for deep neural networks based noise-robust speech recognition," in *ICASSP*, 2014.

[9] C. Weng, D. Yu, S. Watanabe, and B.-H. Juang, "Recurrent deep neural networks for robust speech recognition," in *ICASSP*, 2014.

[10] K. Yao, D. Yu, F. Seide, H. Su, L. Deng, and Y. Gong, "Adaptation of context-dependent deep neural networks for automatic speech recognition," in *Proc. IEEE Workshop on Spoken Language Technology*, pp. 366–369, 2012.

[11] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Ng, "Deep Speech: Scaling up End-to-end Speech Recognition," 2014.

[12] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. Engel, L. Fan, C. Fougner, T. Han, A. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang, C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu, "Deep Speech 2: End-to-End Speech Recognition in English and Mandarin," 2015.

[13] A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber, "Connectionist temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks," in *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

[14] L. Rabiner and B.-H. Juang, "An Introduction to Hidden Markov Models," in *ASSP Masgazine, IEEE*, pp. 4–16, 1986.

[15] L. Baum, J. Eagon, *et al.*, "An inequality with applications to statistcal estimation for probabilistic functions of Markov processes and to a model for ecology," in *Bull. Amer. Math Soc*, pp. 360–363, 1967.

[16] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," in *IEEE Transactions on Information Theory*, pp. 260–269, 1967.

[17] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," in *Journal of the Royal Statistical Society. Series B (Methodolgical)*, pp. 1–38, 1977.

[18] L. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," in *The annals of mathematical statistics*, pp. 164–171, 1970.

[19] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," in *Psychological Review*, pp. 386–408, 1958.

[20] P. Werbos, "Backpropagation Through Time: What It Does and How to Do It," in *Proceedings of the IEEE*, pp. 1550–1560, 1990.

[21] M. Schuster and K. K. Paliwal, "Bidirectional Recurrent Neural Networks," in *IEEE Transactions on Signal Processing*, vol. 45, pp. 2673–2681, 1997.

[22] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," in *Neural Computation*, vol. 9, pp. 1735–1780, 1997.

[23] H. Bourlard and N. Morgan, "Connnectionist Speech Recognition: A Hybrid Approach," in *Kluwer Academic Publishers*, 1994.

[24] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks.* PhD thesis, Technische Universitat Munchen, 2008.

[25] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The Kaldi Speech Recognition Toolkit," in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, IEEE Signal Processing Society, Dec. 2011. IEEE Catalog No.: CFP11SRW-USB.

[26] M. Pezeshki, P. Brakel, S. Tan, and R. Var, "CTC-Connectionist Temporal Classification." https://github.com/mohammadpz/CTC-Connectionist-Temporal-Classification, 2015.

[27] M.A.Anusuya and S.K.Katti, "Speech Recognition by Machine: A Review," in *International Journal of Computer Science and Information Security*, vol. 6, 2009.

[28] T. Bluche, *Deep Neural Networks for Large Vocabulary Handwritten Text Recognition.* PhD thesis, Paris-Sud University, 2015.

[29] Y. Miao, M. Gowayyed, and F. Metze, "EESEN: End-to-End Speech Recognition using Deep RNN Models and WFST-based Decoding," in *Proc. ASRU 2015*, 2015.