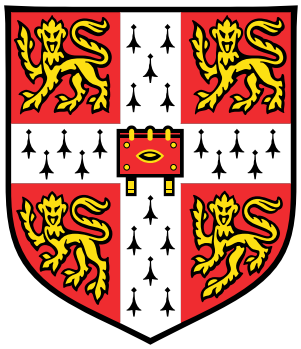
Optimisation Methods for Training Deep Neural Networks in Speech Recognition



Mustafa Adnan Haider

Department of Engineering

University of Cambridge

This dissertation is submitted for the degree of

Doctor of Philosophy

Darwin College

September 2018

Optimisation Methods for Training Deep Neural Networks in Speech Recognition

Mustafa Adnan Haider

Automatic Speech Recognition (ASR) is an example of a sequence to sequence level classification task where, given an acoustic waveform, the goal is to produce the correct word level hypotheses. In machine learning, a classification problem such as ASR is solved in two stages: an inference stage that models the uncertainty associated with the choice of hypothesis given the acoustic waveform using a mathematical model, and a decision stage which employs the inference model in conjunction with decision theory to make optimal class assignments. With the advent of careful network initialisation and GPU computing, hybrid Hidden Markov Models (HMMs) augmented with Deep Neural Networks (DNNs) have shown to outperform traditional HMMs using Gaussian Mixture Models (GMMs) in solving the inference problem for ASR. In comparison to GMMs, DNNs possess a better capability to model the underlying non-linear data manifold due to their deep and complex structure. While the structure of such models gives rich modelling capability, it also creates complex dependencies between the parameters which can make learning difficult via first order stochastic gradient descent (SGD). The task of finding the best procedure to train DNNs continues to be an active area of research and has been made even more challenging by the availability of ever more training data.

This thesis focuses on designing better optimisation approaches to train hybrid HMM-DNN models using sequence level discriminative criterion which is a natural loss function that preserves the sequential ordering of frames within a spoken utterance. The thesis presents an implementation of the second order Hessian Free (HF) optimisation method, and shows how the method can made efficient through appropriate modifications to the Conjugate Gradient algorithm. To achieve better convergence than SGD, this work explores the Natural Gradient method to train DNNs with discriminative sequence training. In the DNN literature, the method has been applied to train models for the Maximum Likelihood objective criterion. A novel contribution of this thesis is to extend this approach to the domain of Minimum Bayes Risk objective functions for discriminative sequence training. With sigmoid models trained on a 50hr and 200hr training set from the Multi-Genre Broadcast 1 (MGB1) transcription task, the NG method applied in a HF styled optimisation framework is shown to achieve better Word Error Rate (WER) reductions on the MGB1 development set than SGD from sequence training.

This thesis also addresses the particular issue of overfitting between the training criterion and WER, that primarily arises during sequence training of DNN models that use Rectified Linear Units (ReLUs) as activation functions. It is shown how by scaling with the Gauss Newton matrix, the HF method unlike other approaches can overcome this issue. Seeing that different optimisers work best with different models, it is attractive to have a consistent optimisation framework that is agnostic to the choice of activation function. To address the issue, this thesis develops the geometry of the underlying function space captured by different realisations of DNN model parameters, and presents the design considerations for an optimisation algorithm to be well defined on this space. Building on this analysis, a novel optimisation technique called NGHF is presented that uses both the direction of steepest descent on a probabilistic manifold and local curvature information to effectively probe the error surface. The basis of the method relies on an alternative derivation of Taylor's theorem using the concepts of manifolds, tangent vectors and directional derivatives from the perspective of Information Geometry. Apart from being well defined on the function space, when framed within a HF style optimisation framework, the method of NGHF is shown to achieve the greatest WER reductions from sequence training on the MGB1 development set with both sigmoid and ReLU based models trained on the 200hr MGB1 training set. The evaluation of the above optimisation methods in training different DNN model architectures is also presented.

I would like to dedicate this thesis to my loving parents ...

Declaration

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. It is not substantially the same as any that I have submitted, or, is being concurrently submitted for a degree or diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. I further state that no substantial part of my dissertation has already been submitted, or, is being concurrently submitted for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. However, some of the materials have been presented at international conferences and workshops [1, 2]. And finally, to the best of my knowledge, the work does not exceed the prescribed word limit for the relevant Degree Committee.

Mustafa Adnan Haider
September 2018

Acknowledgements

I would like to thank my PhD supervisor Prof. Phil Woodland, for his patience and feedback. I am grateful to my parents and my partner Camilla for their continuous support during these 4 years. I am especially thankful to the Islamic Development Bank and Cambridge International Trust for their support during my first three years with an IDB Cambridge International Scholarship.

From a mental perspective, a PhD is like a marathon. To succeed, one needs to be have discipline and mental strength to handle periods of fatigue and setbacks. The athletes Zlatan Ibrahimovic and Lebron James have been inspirational in my life. Their determination to improve every day and the discipline with which they work gave me the blueprint to successfully attain this feat.

Table of contents

List of figures

List of tables

Nomenclature

$\hat{Q}$ Empirical risk function

$\boldsymbol{z}$ Output of a DNN

Q Expected loss w.r.t arbitrary risk measure

$\boldsymbol{g}$ Gating units

$\boldsymbol{J}^r_{\boldsymbol{\theta},t}$ Jacobian of the DNN's pre-softmax output w.r.t $\boldsymbol{\theta}$ at time t

$\boldsymbol{v}_t(i) = \arg\max_{\boldsymbol{\Phi}_{1:t-1}} p_\theta(\boldsymbol{o}_1, \boldsymbol{o}_2 \cdots \boldsymbol{o}_t, \boldsymbol{\phi}_{1:t-1}, \boldsymbol{\phi}_t = j)$,the probability of being state j at time t after seeing all observations upto time t and passing through the most probable sequence $\boldsymbol{\Phi}_{1:t-1}$.

$\nabla \boldsymbol{L}^r_{\text{obj},t}$ Time dependent gradient of F_{obj} with respect to the linear DNN output activations

$\mathbf{I}$ Identity matrix

$(\boldsymbol{x}^r, \boldsymbol{y}^r)$ The rth generalised input-output pair

α Learning Rate

$\alpha_t(j) = p_\theta(\boldsymbol{o}_1, \boldsymbol{o}_2 \cdots \boldsymbol{o}_t, \boldsymbol{\phi}_t(j) = 1)$, the probability of being at state j at time t after seeing all observations upto time t

$\beta_t(j) = p_\theta(\boldsymbol{o}_{t+1}, \boldsymbol{o}_{t+2} \cdots \boldsymbol{o}_T | \boldsymbol{\phi}_t(j) = 1)$, the probability of seeing observations from time $t+1$ to the end of the end utterance given that the the hidden state at time t is j

$\boldsymbol{b}^l_{\boldsymbol{\theta}}$ The bias vector in layer l

$\boldsymbol{\gamma}_t$ Posterior probability of $\boldsymbol{\phi}$ at time t computed by the forward-backward algorithm

$\boldsymbol{\phi}$ Latent variable

$\boldsymbol{\Phi}_{\boldsymbol{w}_{1:L}}$ Set of all possible latent variable sequences associated with the particular hypothesis

$\boldsymbol{\theta}$ Neural Network parameters

$\boldsymbol{\xi}$ Sampled input. For ASR this corresponds to an utterance.

$\boldsymbol{h}_t$ The hidden state of a Recurrent Layer l

$\boldsymbol{W}_{\boldsymbol{\theta}}^{l}$ The weight matrix in layer l

$\hat{\boldsymbol{a}}$ The accumulated vectors of the network's pre-softmax output at different time steps.

$\hat{\mu}$ Tuneable hyper parameter

Λ Diagonal matrix

$\mathcal{H} = \{\boldsymbol{w}_1, \boldsymbol{w}_2 \cdots \boldsymbol{w}_L\}$ Hypothesis sequence

$\mathcal{M}$ Space of all probability distributions that can be generated by a particular model

$\mathcal{O} = \{\boldsymbol{o}_1, \boldsymbol{o}_2 \cdots \boldsymbol{o}_T\}$ Observation sequence

$\mathcal{X}$ Generic Input Space

$\mathcal{Y}$ Generic Output Space

μ Tuneable hyper parameter

$\odot$ Point wise application of function to elements of a vector

$\otimes$ Operator that denotes point wise multiplication between two vectors

Σ Covariance matrix of a Gaussian distribution

σ Layer activation function

ϑ Tuneable hyper parameter

ζ Eigenvalue

$a_{ij} = P_\theta(\boldsymbol{\phi}_t(i) = 1 | \boldsymbol{\phi}_{t-1}(j) = 1)$ (Transition probability of HMM states)

$b_j(\boldsymbol{o}_t) = p_\theta(\boldsymbol{o}_t|\bar{\mathcal{O}}, \boldsymbol{\phi}_t(j) = 1)$(Conditional probability of an observation vector given the state and its acoustic context)

c_{jm} Mixture components of a Gaussian Mixture Model

$d\Phi|_{\boldsymbol{\theta}}$ Directional derivative. When applied to DNNs, this is commonly referred to as the $\mathcal{R}$ operator

$E_{\boldsymbol{\xi}_k}$ Expected value taken with respect to the sampling distribution.

$f(\boldsymbol{x}, \boldsymbol{\theta})$ Map represented by a DNN model

$f^l(\boldsymbol{x}, \boldsymbol{\theta}^l)$ Map associated with layer l of a DNN

$g(\boldsymbol{\theta_k}, \boldsymbol{\xi}_k)$ Stochastic update vector

$G_{\boldsymbol{\theta}}$ Gauss Newton Matrix

$I_{\boldsymbol{\theta}}$ Fisher Information Matrix

$L(\mathcal{H}, \mathcal{H}^r)$ Loss associated with a proposed hypothesis $\mathcal{H}$ for a given utterance r

$P(\mathcal{H}|\mathcal{O})$ Discrete distribution of hypothesis word sequence given observation sequence

$p(\mathcal{O}|\mathcal{H})$ Probability density of observation sequence given observation sequence

$P_{\boldsymbol{\theta}}$ Discrete probability distribution yielded by a particular parameter setting

$p_{\boldsymbol{\theta}}$ Probability density yielded by a particular parameter setting

$T_{\boldsymbol{\theta}}X$ Tangent space of $\boldsymbol{\theta}$ on the manifold X

X Parameter manifold

$X/\sim$ Set of equivalent classes

Chapter 1

Introduction

Human speech is perhaps the most natural mode of communication, and can potentially provide an intuitive user interface to machines. The task of mapping a segment of spoken audio signal to a transcription of words has been a subject of continued interest for researchers for over fifty years, and has matured remarkably during this time [3, 4]. This is due in part to the increase in available computational resources, and in part to the rise of more sophisticated modelling techniques. Automatic Speech Recognition (ASR) formally can be described as a sequence to sequence level classification task where given an acoustic waveform $\mathcal{O}$, the goal is to produce the correct hypotheses sequence $\mathcal{H}$. In machine learning, such a problem is solved by partitioning the classification problem into two separate stages: an inference stage that models $P(\mathcal{H}|\mathcal{O})$ using training data and a subsequent decision stage where decision theory is employed to make optimal class assignments using the posterior probabilities. Assuming that the chosen inference model is correct, the task of ASR can formally described as employing a decision rule that yields the minimum hypothesis/sentence error rate:

$$\hat{\mathcal{H}} = \underset{\mathcal{H}}{\operatorname{argmax}}\, P(\mathcal{H}|\mathcal{O}) \tag{1.1}$$

For state of the art ASR systems, the inference model generally comprises of two components: an acoustic model, which represents the relationship between an audio signal and the phonemes or other linguistic units that make up speech, and a language model which presents a probability distribution over sequences of words. Until recently, the acoustic model in modern systems comprised of Hidden Markov models (HMMs), which dealt with the temporal variability of speech, and Gaussian Mixture Models (GMMs) which model how well each state of an HMM represents the spectral properties of the

acoustic input. With the advent of GPU computing and careful network initialisation, HMMs embedded with Deep Neural Networks (DNNs) have shown to outperform the traditional hybrid HMM-GMMs models on a wide range of small and large vocabulary tasks [5]. DNNs, by comparison with GMMs, possess a better capability to model the underlying non-linear data manifold due to their deep and complex structure. However, the complexity of their structures creates complex dependencies between their model parameters that can make learning difficult with standard gradient descent optimisation procedures [6, 7].

1.1 Problem Setting

To train parametric models such as DNNs, the default choice for an optimisation method is gradient descent. The method is a derivative based approach that iteratively improves upon the training objective by locally following the direction of steepest descent. However, when applied to DNNs, the method has been observed to be quite unstable with topologies where the gradient has to propagate through many layers [8, 9]. Layers close to the output layer will have gradients with higher magnitudes than layers close to the input. As a consequence, training may either fail because of *exploding* gradients or become increasingly slow due to gradients *vanishing* as we propagate down through the network. To stabilise training, the optimisation approach is often accompanied by some form of update clipping that ensures individual parameter updates are in feasible range. However, this comes at the cost of an increase in the number of hyper-parameters that need to be carefully tuned to make learning with gradient descent work [10].

In practice, designing a good optimisation algorithm to effectively probe the parameter surface is a complex task. To improve the training of deep models, the general strategy has not always been to improve the optimisation algorithm. Instead, many improvements in the optimisation of deep models have come from re-designing the models [11–13] such that a good set of parameters can be found by standard gradient descent. However, such an approach is sub-optimal as the choice of model architecture is restricted by its ability to be trained with SGD. The difficulty of training DNNs can be significantly reduced by second order optimisation methods [14]. By scaling the gradient direction by the inverse of the local curvature matrix, these methods can adjust the gradient direction when there is high non-linearity and ill conditioning of the objective function. For convex optimisation problems, it has been shown in [15] that such a modification allows the optimisation process to achieve both faster and better convergence. Even though second

order methods seem ideal, a major drawback of these methods is their inability to scale with the model size: if D is the number of parameters then constructing the Hessian of the second order derivative explicitly requires $\mathcal{O}(D^2)$ storage, and inverting it incurs a cost of $\mathcal{O}(D^3)$. Thus, the task of finding the best procedure to train DNNs continues to be an active area of research. This task has become even more complex with the availability of more training data.

From the perspective of data utilisation, all iterative optimisation schemes primarily fall into two categories: batch and stochastic. These frameworks differ in the amount of training data they employ to generate update statistics at each iteration. Batch methods employ the entire training or large subsets of the training set whereas stochastic approaches sample only a small subset of samples at each iteration to generate equivalent update statistics. Bottou [16] showed that by accumulating information from large batches, derivative based batch style frameworks do better than their stochastic counterpart in converging to a good solution. However, the ability of such methods to converge to a good local minima depends highly on the capacity to perform a large number of updates. When constrained to a single machine, batch approaches become impractical in the scenario where either the training dataset is too large or when the computational budget is fixed. In contrast, the per iteration cost of a stochastic approach is not tied to the training set size. The ability to perform more updates within an epoch allows such methods to achieve greater reductions in the training error. ASR is a sequence to sequence modelling task for which discriminative sequence training is an appropriate training loss criterion as it preserves the sequential ordering of frames within utterances. To minimise such an objective function, stochastic optimisation frameworks are restricted to operate within a utterance randomisation scheme rather than the standard frame randomisation approach. Under such context, when faced with thousands of hours of training data, methods like utterance level Stochastic Gradient Descent (SGD) at times may not be able to make a complete pass of the training set.

One might argue that such issues can be alleviated by having a parallel optimisation framework. With stochastic methods like standard SGD, parallelisation can be achieved either in a synchronous or asynchronous way. In synchronous SGD (SSGD) [17], local workers compute the gradients over their own mini-batches[1] and then propagate the gradients to the global model. In the context where the optimisation approach requires a lot of updates to converge, using SSGD can lead to significant computational overhead as the global model always has to wait for the slowest worker to reply before applying an

[1] for sequence training, this corresponds to a subset of utterances.

update. To improve training efficiency, asynchronous SGD (ASGD) was proposed in [18] where each local worker continues its training process right after its gradient is added to the global model. Although ASGD can achieve faster speed due to no waiting overhead, it suffers from a problem called delayed gradient [19] where the global model receives 'stale' gradients (i.e gradients computed with respect to previous parameter settings) from worker nodes. This makes the approach apart from being not mathematically sound also not repeatable.

1.2 Thesis Outline

In [20], it has been shown that solving the optimisation problem for sequence training within a second order Hessian Free (HF) optimisation framework achieves a good balance between parallelisation and faster convergence. This thesis builds upon that work and focuses on designing efficient synchronous optimisation frameworks, that have the same advantages as the large batch HF method but leads to larger Word Error Rate (WER) reductions than SGD from sequence training of DNN acoustic models. The prime focus of this work will be to improve the training of HMM-DNN models using the Minimum Phone Error (MPE) criterion. The efficacy of various optimisation approaches are evaluated using a 50hr or 200hr training set taken from the the 2015 Multi-Genre Broadcast ASRU challenge task (MGB1) task [21]. A brief description of the data preparation is presented in Appendix C.2 but further details can be found in [22]. To investigate the effectiveness of various setups, comparisons based on the amount of compute time, the number of epochs, the expected phone accuracy, and the number of updates will be presented.

This thesis is organised as follows:

- Chapter 2 provides an review of the fundamental components of an ASR system that are relevant to the research in this PhD. A distinction between generative and discriminative learning frameworks is given along with a review of the different forms of Minimum Bayes Risk (MBR) training.

- Chapter 3 presents a literature review of Deep Neural Networks within the context of acoustic modelling in ASR. The chapters reviews the concept of an Artificial Neural Network and discusses how the choice of DNN architecture controls the representational capability of a chosen model. A review of the back-propagation algorithm is given along with the form of the gradients of the various sequence

level losses discussed in Chapter 2. The chapter concludes with a discussion of the instabilities associated with SGD training.

- Since the focus of this thesis is optimisation, a separate chapter has been included to review derivative based optimisation approaches. A review of the first and second order optimisation method is presented along with a discussion on how such methods are applied within batch and stochastic frameworks. The chapter presents a careful analysis to understand the behaviour of stochastic approaches and reviews the standard regularisation methods currently used to improve the generalisation performance of derivative based learning.

- Chapter 5 presents the implementation details of the Hessian Free (HF) optimisation method investigated in this thesis. The chapter begins with an analysis of batch and stochastic methods and discusses the advantages and disadvantages of existing distributed optimisation frameworks. A review of the Conjugate Gradient (CG) algorithm(the core component of the HF approach) is presented followed by a detailed description on how curvature vector products using the Gauss Newton (GN) matrix can be computed in a DNN. The chapter presents novel procedures to stabilise CG training and adapt it to effectively handle tied architectures. The chapter concludes with preliminary experiments on the 50hr MGB1 training set.

- Chapter 6 extends the method of Natural Gradient (NG) to the domain of MBR objective functions for discriminative sequence training. The method is presented within an HF style optimisation framework where instead of using the GN matrix, the CG algorithm is equipped with the empirical Fisher Information matrix. The efficacy of the method is shown using experiments with sigmoid DNNs on the 50hr and 200hr MGB1 training set. In practice, the empirical Fisher Information is only guaranteed to be positive semi definite. To address this issues, this chapter provides the derivation of a damped FI matrix which when used with CG has the property that directions considered important by the empirical Fisher are traversed first during the initial stages of a CG run.

  The chapter also raises the particular issue of over-fitting due to mismatch of training criterion and WER that often companies training DNN models using the ReLU activation function, and shows how this problem can be partially alleviated by scaling the update directions with the GN matrix.

- Chapter 7 develops the topological structure of the DNN function space $\mathcal{M}$ and presents the property needed for optimisation algorithms to be well defined on the space. The chapter introduces a novel optimisation approach that effectively combines the method Natural Gradient with second order approaches. The method termed as NGHF is derived from an alternative re-derivation of Taylor's theorem using concepts from Information Geometry [23, 24]. The chapter details the various steps of this derivation and shows how the method can be applied within an HF styled optimisation framework. Experiments on the 50hr and 200hr MGB1 training set show that the method is agnostic with respect to both the choice of feed forward architecture and choice of DNN activation functions. This chapter also investigates the efficacy of the various optimisation approaches for training standard RNNs with different activation functions. On the 200hr MGB1 training set, the method of NGHF will be shown to be the most effective in achieving consistent reductions in WER from sequence training.

- Chapter 8 presents conclusions and suggestions for future work.

1.3 Key Contributions

The key contributions of this thesis are as follows:

1. A procedure to stabilise CG training that allows effective updates to be found in only a few iterations is devised. Equipping CG with this proposed modification significantly reduces the contribution of the algorithm to the overall computational cost. It is also shown how the CG algorithm can be adjusted to better handle DNN architectures with tied parameters.

2. Extended the method of Natural Gradient to the domain of Minimum Bayes Risk discriminative sequence training for hyrbid HMM-DNN models. The NG method was first proposed by Amari [25] as an effective optimisation method for training parametric density models w.r.t the Maximum Likelihood (ML) objective criterion. Section 6.1.1 attributes these observed gains to the fact that for ML methods, the method of NG becomes increasingly similar to a second order approach in the context of a large amount of data. A novel contribution of this work is to extend the approach to training models with loss functions that do not correspond to the ML objective. When framed within a Hessian Free style optimisation framework, the method is shown to yield better WER convergence than SGD on sigmoid models.

3. Instead of making a strict assumption on the structure of the matrix, this thesis derives an alternative dampened positive definite Fisher matrix which when used with CG has the property that directions considered important by the empirical Fisher are first traversed during the initial stages of a CG run.

4. Addressed the issue of over-fitting due to mismatch between training criterion and Word Error Rate (WER) that primarily arises during sequence training of ReLU-DNN models. It is shown how this particular form of over-fitting can be alleviated by scaling the update directions with the Gauss Newton matrix.

5. Developed the geometric structure of the function space captured by different parameter realisations of a DNN model, and present the property needed for optimisation methods to be well defined on this space.

6. Introduced NGHF, a novel optimisation approach that combines the method of Natural Gradient with second order approaches. The method is derived from an alternative reformulation of Taylor's theorem using principles from Information Geometry and manifold theory. By utilising both the direction of steepest descent on a probabilistic manifold and local curvature information, the efficacy of the method is shown to be agnostic with respect to both the choice of DNN architecture and activation functions.

7. Evaluated all of the above methods for discriminative sequence training on large vocabulary speech transcription task.

Chapter 2

Fundamentals of Automatic Speech Recognition

The dynamic nature of human speech makes the automatic process of translating a speech waveform into its corresponding transcription quite challenging. Utterances can differ in their duration even when they represent the same content and are spoken by the same speaker. To add further complexity, the number of potential hypotheses that the system must search through increases exponentially as we increase the size of the vocabulary. These properties make the task of ASR quite difficult for Large Vocabulary Continuous Speech Recognition (LVCSR).

This chapter presents a background review of the main modules of an ASR system that are relevant to this thesis. From a high level perspective, a standard ASR system has three distinct modules:

1. Front end processing: this involves mapping the input signal to a series of observation feature vectors $\mathcal{O} = \{\boldsymbol{o}_1, \boldsymbol{o}_2 \cdots \boldsymbol{o}_T\}$. Here $\boldsymbol{o}_t$ is the feature vector at time t derived from the corresponding speech frame (an audio segment), and T is the total number of frames in the utterance.

2. Inference stage: this models the uncertainties associated with various hypotheses using information from the extracted features. A hypothesis corresponds to a variable word sequence $\boldsymbol{w}_{1:L}$ with L denoting the number of words.

3. Decision stage: this refers to the decoding phase where decision theory in used in conjunction with the inference model to find the 'best' word sequence (hypothesis) for a given observation sequence.

In the rest of this chapter, each of the above modules will be reviewed. As this thesis focusses in improving hybrid HMM-DNN models, particular emphasis will be made on the various learning strategies used to train these hybrid models.

2.1 Feature Extraction

The first step in any ASR system is to identify the components of an audio signal that are good in discriminating the linguistic content and discard details that carry information of background noise, emotion etc. This stage is known as front end preprocessing and is achieved by mapping the input waveform into a series of feature vectors, $\mathcal{O} = \{\boldsymbol{o}_1, \boldsymbol{o}_2 \cdots \boldsymbol{o}_T\}$, through some feature extraction process. In speech technology, there are currently 3 standard approaches to perform this initial front end preprocessing: Mel-Frequency Ceptral Coefficient (MFCC)s [26, 27], filter-banks [27] and Perceptual Linear Prediction (PLP) [28]. It should be mentioned that there is also ongoing work that aims to use the acoustic waveform directly for training the acoustic models [29, 30].

Human speech is a non stationary signal where the acoustic realisations of individual linguistic units are distinguished by the envelope of the time dependent power spectrum. To address this dependency, all three approaches proceed by first segmenting the input waveform into frames of size 25 ms in length spaced 10 ms apart. On these short time scales, it is assumed that the audio doesn't change much and is statistically stationary. Among the mentioned approaches, the procedures used to compute filter banks and MFCCs from individual frames substantially overlap. In both cases, filter bank features are computed first, using the procedure below:

1. Apply the Discrete Fourier Transform (DFT) to obtain a frequency domain representation of the original input signal. This is motivated by the fact that the human cochlea (an organ in the ear) vibrates at different spots depending on the frequency of the incoming sounds.

2. Compute the mel-frequency spectrum by first filtering the frequency spectrum with N different band-pass filters and then computing the power associated with each frequency band. This filtering mimics the human ear since the cochlea can not discern the difference between two closely spaced frequencies.

3. Take the logarithm of all filter bank energies to mimic the human perception of loudness.

Once the filter bank features are computed, the computation of MFCC features is achieved by performing the following two additional steps:

1. Apply the Discrete Cosine Transform (DCT) to the log filter bank energies to de-correlate the overlapping filter bank features.

2. Take the first k DCT coefficients and discard the rest. The higher DCT coefficients represent fast changes in the filter bank energies and has been found to degrade ASR performance with traditional HMM-GMM systems [31].

Front end preprocessing using PLP features are often used as an alternative to MFCCs. These features mainly approximate three core perceptual aspects: the non-linear frequency resolution curves, the equal- loudness curve, and the intensity-loudness power-law relation [28]. The standard scheme to compute such features is given below:

1. Bark-frequency warping: the frequency is warped using the Bark-frequency scale [32]. As the power spectrum value, the square of the magnitude spectrum is used to extract PLP features. This process results in a scaled power spectrum.

2. Down-sampling and post-processing: the power spectrum is convolved with a number of critical band filters to get downsampled values. These values are then scaled by using the curve of equal-loudness and intensity-loudness power law.

3. Linear Prediction (LP) Analysis: the resulting spectrum is then mapped to an auto-correlation sequence in the time domain to yield LP coefficients.

4. Cepstral coefficients calculation: the inverse DFT transform is applied on the log magnitude of the spectrum of the LP coefficients to yield the required PLP coefficients.

Alternatively, PLP features can be also extracted based on the mel-frequency filter bank, referred to as MF-PLP [33]. In the ML-PLP approach, the mel filter bank coefficients are first computed and then re-weighted by the equal-loudness curve, and then compressed by taking the cubic root. The resultant spectrum is then fed as input to step 3 of the PLP extraction pipeline.

2.2 Inference Model

The goal of the inference stage is to learn the relationship between the input space $\mathcal{X}$ and the output space $\mathcal{Y}$ through the use of a statistical model $P_{\boldsymbol{\theta}}(\mathcal{Y}|\mathcal{X})$. Determining $P_{\boldsymbol{\theta}}(\mathcal{Y}|\mathcal{X})$ is complex and at present is solved by either employing a generative modelling approach or a discriminative modelling approach.

Generative models aim to model $P(\mathcal{Y}|\mathcal{X})$ by modelling the joint distribution $P(\mathcal{Y}, \mathcal{X})$ instead, which with respect to our problem corresponds to $p(\mathcal{O}, \mathcal{H})$. To clarify for the reader, $\mathcal{O} = \{\boldsymbol{o}_1, \boldsymbol{o}_2 \cdots \boldsymbol{o}_T\}$ and $\mathcal{H} = \boldsymbol{w}_{1:L}$. Using Bayes theorem, the posterior probabilities can be computed as:

$$P(\mathcal{H}|\mathcal{O}) = \frac{p(\mathcal{O}, \mathcal{H})}{p(\mathcal{O})} \tag{2.1}$$

Note that any of the other quantities appearing in (2.1) can be obtained from the joint distribution $p(\mathcal{O}, \mathcal{H})$ by either marginalising or conditioning with respect to the appropriate variables. In ASR, state of the art generative approaches assume the following factorisation of the joint distribution:

$$p(\mathcal{O}, \mathcal{H}) = p(\mathcal{O}|\mathcal{H})P(\mathcal{H})$$

The class conditional likelihood $p_{\boldsymbol{\theta}}(\mathcal{O}|\mathcal{H})$ is modelled by an Acoustic Model (AM) and probability of the word sequences is modelled separately by a Language Model (LM). The AM attempts to model the relationship between an audio signal and the phonemes or other linguistic units that make up speech. The acoustic scores produced by the AM are combined with the LM to provide an estimate of the joint distribution. The LM is often trained on a separate text corpus belonging to a different knowledge source and contains a great deal of prior information. Combining the LM with the AM have always been observed to result in significant improvement of recognition performance [31, 34].

In contrast to generative models, discriminative models directly model the posterior distribution $P(\mathcal{H}|\mathcal{O})$ using some form of a statistical model $P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})$. Under such a modelling scheme, more emphasis is put on the accurate estimation of class decision boundaries instead of the accurate estimation of $p(\mathcal{O}, \mathcal{H})$.

2.2.1 Hidden Markov Model based Generative Models

For sequence to sequence modelling tasks such as ASR, hybrid forms of Hidden Markov Models (HMMs) are the most preferred choice among generative models to model $p_{\boldsymbol{\theta}}(\mathcal{O}|\mathcal{H})$. HMMs [35] belong to the class of generative discrete latent variable graphical models that assume a particular factorisation of the distribution $p(\mathcal{O}|\mathcal{H})$:

$$\begin{aligned} p(\mathcal{O}|\mathcal{H}) &= p(\boldsymbol{o}_{1:T}|\boldsymbol{w}_{1:L}) \\ &= \sum_{\boldsymbol{\phi}_{1:T} \in \boldsymbol{\Phi}_{\boldsymbol{w}_{1:L}}} p(\boldsymbol{o}_{1:T}|\boldsymbol{\phi}_{1:T}) P(\boldsymbol{\phi}_{1:T}|\boldsymbol{w}_{1:L}) \end{aligned} \tag{2.2}$$

In HMM terminology, the latent variables $\{\boldsymbol{\phi}_t\}_t$ are commonly referred as states. This should not be confused with the notion of hidden states associated with Recurrent Neural Networks (RNNs).
The above factorisation yields two distinct models:

- $p(\boldsymbol{o}_{1:T}|\boldsymbol{\phi}_{1:T})$ which corresponds to *conditional* acoustic model and defines the probability of the observation sequence given the latent variable sequence.
- $P(\boldsymbol{\phi}_{1:T}|\boldsymbol{w}_{1:L})$ models the temporal variability in speech by mapping the 'relationship' between latent variable sequences with a given word sequence.

Here $\boldsymbol{\Phi}_{\boldsymbol{w}_{1:L}}$ represents the set of all possible latent variable sequences associated with the particular hypothesis $\boldsymbol{w}_{1:L}$.

HMMs in particular make strict conditional independence assumptions when modelling the above distributions:

$$P(\boldsymbol{\phi}_{1:T}|\boldsymbol{w}_{1:L}) = \prod_t P(\boldsymbol{\phi}_t|\boldsymbol{\phi}_{1:t-1}, \boldsymbol{w}_{1:L}) \approx \prod_t P(\boldsymbol{\phi}_t|\boldsymbol{\phi}_{t-1}) \tag{2.3}$$

$$p(\boldsymbol{o}_{1:T}|\boldsymbol{\phi}_{1:T}) = \prod_t p(\boldsymbol{o}_t|\boldsymbol{o}_{1:t-1}, \boldsymbol{\phi}_{1:T}) \approx \prod_t p(\boldsymbol{o}_t|\bar{\mathcal{O}}, \boldsymbol{\phi}_t) \tag{2.4}$$

Here $\bar{\mathcal{O}}$ can correspond to either $\boldsymbol{o}_{[t-k,t+k]}$, $\boldsymbol{o}_{1:t-1}$ or $\boldsymbol{o}_{1:T}$. Thus, under the HMM framework, the likelihood then corresponds to:

$$p(\mathcal{O}|\mathcal{H}) = \sum_{\boldsymbol{\phi}_{1:T} \in \boldsymbol{\Phi}_{\boldsymbol{w}_{1:L}}} \left[\prod_t^T p(\boldsymbol{o}_t|\bar{\mathcal{O}}, \boldsymbol{\phi}_t) P(\boldsymbol{\phi}_t|\boldsymbol{\phi}_{t-1}) \right] \tag{2.5}$$

For notational simplicity, we denote

$$b_j(\boldsymbol{o}_t) = p_\theta(\boldsymbol{o}_t|\bar{\mathcal{O}}, \boldsymbol{\phi}_t(j) = 1)$$
$$a_{ij} = P_\theta(\boldsymbol{\phi}_t(i) = 1|\boldsymbol{\phi}_{t-1}(j) = 1)$$

By multiplying (2.5) with the output of an LM i.e $P(\boldsymbol{w}_{1:L})$, HMMs combined with LMs have the ability to model the joint distribution $p(\mathcal{O}, \mathcal{H})$. In ASR, the HMM topology used to model the most basic unit of the AM corresponds to a left to right HMM with an entry and exit state and equipped with 3 or 1 emitting states (Figure 2.1).

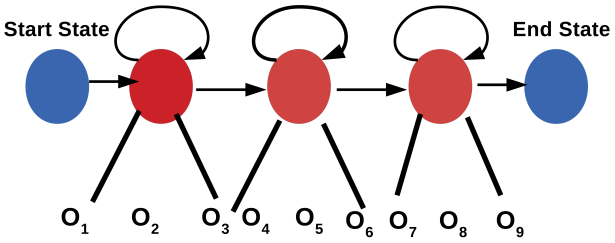


Fig. 2.1 Structure of a left to right HMM

This basic topology serves as the underlying model for individual phones, the basic unit of sound in a spoken utterance. Under such a framework, the latent variables $\{\boldsymbol{\phi}_t\}_t$ introduced in (2.2) correspond to sub-phone states. An important advantage of using this particular form of HMM topology is that larger HMMs can be constructed by the composition of these basic models. This allows word models to be constructed from sub-word units and consequently sentence models from word models. When building sentence level models, having such a topology also provides the flexibility to easily integrate LM scores as transition probabilities between states matching the end of one word and the start of another. Such a modified HMM can then be used to model $p_\theta(\mathcal{O}, \mathcal{H})$ for any given utterance $\mathcal{H}$.

2.2.1.1 Co-articulation and parameter tying

When processing continuous speech, an important factor to take into account is the co-articulation effect. Co-articulation refers to the phenomenon when an isolated speech sound is influenced by its neighbouring speech sounds due to the continuous movement of articulating speech organs [36]. This phonetic variation is typically modelled by introducing context-dependent HMMs. Context-dependent models [37] use the HMM topology described above to model phonemes along with the left and right context.

Examples include triphone and pentaphone HMMs which model the centre phoneme of sequences of three and five phonemes respectively. However, having a distinct model for every possible phonetic context significantly increases the model complexity and poses a threat of over-fitting when we integrate them with DNNs. To address this issue, the current approach is to tie the sub phone states of modular HMMs that represent 'similar' acoustic context. This is presently achieved by applying either bottom up or top down clustering. The bottom approach employs a data driven approach to cluster individual states under the influence of a metric [38, 39] while the top-down approach employs a decision tree to cluster states [40]. In practice, decision trees are widely used as they are more adept to handling unseen triphone/pentaphone states and better utilise linguistic knowledge.

2.2.1.2 Conditional Acoustic Model

Before it became feasible to train very deep networks, the traditional approach to modelling the distribution $p(\boldsymbol{o}_t|\bar{\mathcal{O}}, \boldsymbol{\phi}_t)$ was to employ Gaussian Mixture Models (GMMs):

$$p_\theta(\boldsymbol{o}_t|\bar{\mathcal{O}}, \boldsymbol{\phi}_t(j) = 1) = \sum_{m=1}^{M} c_{jm}\mathcal{N}(\boldsymbol{o}_t, \boldsymbol{\mu}_{jm}, \Sigma_{jm}) \tag{2.6}$$

where the mixture components c_{jm} satisfy $\sum_m c_{jm} = 1$ and $\mathcal{N}(.)$ is a Gaussian probability distribution with $\boldsymbol{\mu}$ and Σ representing its mean and covariance matrix.

$$\mathcal{N}(\boldsymbol{o}; \boldsymbol{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^D|\Sigma|}} \exp(-\frac{1}{2}(\boldsymbol{o} - \boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{o} - \boldsymbol{\mu})) \tag{2.7}$$

The normalisation term is independent of $\boldsymbol{o}$ and can be precomputed and cached for efficiency. Given enough components, GMMs can model probability distributions to any required level of accuracy, and they are fairly easy to fit to data using the Expected Maximisation (EM) algorithm [41]. For HMMs, it is not necessary for all clustered states to have the same form of output distribution or even for each state to have the same number of Gaussian components. However for d-dimensional feature vectors, assigning each gaussian with its own covariance matrix is very expensive. In practice, only block or diagonal covariance matrices are used but this in turn reduces the complexity of the models. To address this issue, one effective approach is to use semi-tied covariances (STC) [42] that reduces the number of parameters by sharing full covariance or precision matrices over a number of components.

With the advent of GPU computing and careful network initialisation (discussed in Sec. 3.3), as it become feasible to train very deep neural networks (DNNs), HMMs using DNNs to model $p_\theta(\boldsymbol{o}_t|\bar{\mathcal{O}}, \boldsymbol{\phi}_t)$ have shown to outperform the tradition HMM-GMM models on a wide range of small and large vocabulary tasks [5]. However, DNNs when applied to acoustic modelling represent discriminative models where their softmax outputs $\boldsymbol{z}_t^r$ correspond to $P_\theta(\boldsymbol{\phi}_t|\boldsymbol{o}_t^r, \bar{\mathcal{O}})$; for networks with feedforward architectures [43], $\bar{\mathcal{O}}$ corresponds to a splice $\boldsymbol{o}_{[t-k,t+k]}$ whereas for DNNs possessing recurrent connections, $\bar{\mathcal{O}}$ corresponds to either $\boldsymbol{o}_{1:t-1}$ or $\boldsymbol{o}_{1:T}$ (bi-directional). Using Bayes rule, the DNN frame posteriors can be mapped to HMM state probability density functions as follows:

$$p(\boldsymbol{o}_t, \bar{\mathcal{O}}|\boldsymbol{\phi}_t) = \frac{P_{\boldsymbol{\theta}}(\boldsymbol{\phi}_t|\boldsymbol{o}_t, \bar{\mathcal{O}})p(\boldsymbol{o}_t, \bar{\mathcal{O}})}{P(\boldsymbol{\phi})} \tag{2.8}$$

$$p(\boldsymbol{o}_t|\bar{\mathcal{O}}, \boldsymbol{\phi}_t) = \frac{P_{\boldsymbol{\theta}}(\boldsymbol{\phi}_t|\boldsymbol{o}_t, \bar{\mathcal{O}})p(\boldsymbol{o}_t, \bar{\mathcal{O}})}{p(\bar{\mathcal{O}}|\boldsymbol{\phi}_t)P(\boldsymbol{\phi})} \tag{2.9}$$

Taking logs

$$\log p(\boldsymbol{o}_t|\bar{\mathcal{O}}, \boldsymbol{\phi}_t) = \log P_{\boldsymbol{\theta}}(\boldsymbol{\phi}_t|\boldsymbol{o}_t, \bar{\mathcal{O}}) - \Big[\log p(\bar{\mathcal{O}}|\boldsymbol{\phi}_t) + \log P(\boldsymbol{\phi}) - \log p(\boldsymbol{o}_t, \bar{\mathcal{O}})\Big] \tag{2.10}$$

In practice, for computational simplicity, the standard approach [44–47] uses the following approximation to map DNN frame posteriors to HMM scaled state likelihoods.

$$\log p(\boldsymbol{o}_t|\bar{\mathcal{O}}, \boldsymbol{\phi}_t) \simeq \log P_{\boldsymbol{\theta}}(\boldsymbol{\phi}_t|\boldsymbol{o}_t, \bar{\mathcal{O}}) - \Big[\log P(\boldsymbol{\phi})\Big] \tag{2.11}$$

2.2.2 Sequence to Sequence Discriminative Models

An alternative approach to modelling the joint distribution $P(\mathcal{H}, \mathcal{O})$ is to model $P(\mathcal{H}|\mathcal{O})$ directly. Recently, there has been considerable interest in training end-to-end discriminative models [48–51] that employ only neural networks to directly recognise utterances without requiring separately-trained acoustic, pronunciation and language model components. Such models conform to a particular form of discrete latent variable graphical model that assumes the following factorisation of the posterior distribution $P(\mathcal{H}|\mathcal{O})$:

$$P(\mathcal{H}|\mathcal{O}) = \sum_{\boldsymbol{\phi}_{1:T} \in \boldsymbol{\Phi}_{\boldsymbol{w}_{1:L}}} P(\boldsymbol{w}_{1:L}|\boldsymbol{\phi}_{1:T})P(\boldsymbol{\phi}_{1:T}|\boldsymbol{o}_{1:T}) \tag{2.12}$$

where

- $P(\boldsymbol{w}_{1:L}|\boldsymbol{\phi}_{1:T})$ is the word alignment model and represents the 'relationship' of output word sequences with given latent variable sequences. In current literature, the latent variables $\{\boldsymbol{\phi}_t\}_t$ for NN based end-to-end models correspond to graphemes or sub-words (word pieces) [52]. Such an interpretation establishes a one to one correspondence between a word sequence with a given latent variable sequence as a consequence of which

$$P(\boldsymbol{w}_{1:L}|\boldsymbol{\phi}_{1:T}) = 1 \tag{2.13}$$

- $P(\boldsymbol{\phi}_{1:T}|\boldsymbol{o}_{1:T})$ is the discriminative acoustic model which models the uncertainty associated with different latent variable sequences for a given observation sequence.

The most common form of end-to-end DNN models use a Connectionist Temporal Classification (CTC) [53] approach that makes the following strict conditional independence assumption of the discriminative acoustic model:

$$P(\boldsymbol{\phi}_{1:T}|\boldsymbol{o}_{1:T}) = \prod_t P(\boldsymbol{\phi}_t|\boldsymbol{\phi}_{1:t-1}, \bar{\mathcal{O}}) \approx \prod_t P(\boldsymbol{\phi}_t|\bar{\mathcal{O}}) \tag{2.14}$$

where $\bar{\mathcal{O}}$ can correspond to either $\boldsymbol{o}_{[t-k,t+k]}$, $\boldsymbol{o}_{1:t-1}$ or $\boldsymbol{o}_{1:T}$. Thus, under CTC, the posterior distribution corresponds to:

$$P(\mathcal{H}|\mathcal{O}) = \sum_{\boldsymbol{\phi}_{1:T} \in \boldsymbol{\Phi}_{\boldsymbol{w}_{1:L}}} \left[\prod_t P(\boldsymbol{\phi}_t|\bar{\mathcal{O}}) \right] \tag{2.15}$$

From (2.15), it can be seen that such models are essentially alignment-free as to get the probability of an output sequence given an input sequence, the probability of all possible alignments between the two sequences are taken into account. However, a major limitation of the CTC approach is that unlike hybrid generative HMMs, the discriminative model is not capable of using information from text only data. This deficiency has been shown to make such models underperform against standard hybrid HMMs equipped with DNNs [54].

2.3 Forward-Backward algorithm

From (2.5) and (2.15), it can be seen that if $\boldsymbol{\phi}_t$ takes N values, the computation of the required probabilities will amount to summing over all possible N^T possible latent variable sequences. This makes the computation very expensive especially when utterances

having long temporal duration are considered. To make such computations tractable, the standard approach is to employ a dynamic programming framework known as the forward-backward algorithm [55] to compute the necessary probabilities. As this thesis focuses solely on hybrid HMM-DNN models, the workings of the algorithm will be explained within the context of HMM based acoustic models.

2.3.0.1 Algorithm sketch

The forward-backward algorithm consists of two separate recursive algorithms: a forward algorithm and a backward algorithm. When applied to HMMs, the forward algorithm provides an efficient way to compute $p_\theta(\mathcal{O}|\mathcal{H})$ by using a table to store intermediate values as it builds up the probability of the observation sequence. The algorithm computes the observation probability by summing over the probabilities of all possible latent variable sequences that could generate the observation sequence, but it does so efficiently by implicitly folding each of these paths into a single forward trellis. A forward trellis can be thought of as a two dimensional grid where the 'y-axis' represents the values the latent variable $\boldsymbol{\phi}_t$ can take, and the 'x-axis' represents time. Each cell in the trellis corresponds to $\alpha_t(j) = p_\theta(\boldsymbol{o}_1, \boldsymbol{o}_2 \cdots \boldsymbol{o}_t, \boldsymbol{\phi}_t(j) = 1)$, the probability of being at state j at time t after seeing all observations up to time t. This term can be written recursively as:

$$\alpha_t(j) = \sum_{i \in M_H} \alpha_{t-1}(i) a_{ij} b_j(\boldsymbol{o}_t) \tag{2.16}$$

where M_H represents the set of all states in the HMM. Alternatively, the backward algorithm too provides an efficient way to compute $p_\theta(\mathcal{O}|\mathcal{H})$ by a similar use of a trellis. In this case, each cell in the trellis corresponds to $\beta_t(j) = p_\theta(\boldsymbol{o}_{t+1} \cdots \boldsymbol{o}_T | \boldsymbol{\phi}_t(j) = 1)$, the probability of seeing observations from time $t+1$ to the end of the utterance given that the the hidden state at time t is j. This term too can be recursively computed as:

$$\beta_t(j) = \sum_{k \in M_H} a_{jk} b_k(\boldsymbol{o}_{t+1}) \beta_{t+1}(k) \tag{2.17}$$

The forward and backward probabilities $\alpha_t(j)$ and $\beta_t(j)$ also makes it possible to efficiently compute the posterior probability $P_{\boldsymbol{\theta}}(\boldsymbol{\phi}_t|\mathcal{O})$, which is conventionally expressed as $\boldsymbol{\gamma}_t$ in ASR research:

$$\boldsymbol{\gamma}_t(j) = \frac{\alpha_t(j)\beta_t(j)}{p_\theta(\mathcal{O}|\mathcal{H})} \tag{2.18}$$

As will be shown in Sec.3.6, γ_t is an important statistic needed to train model parameters using derivative based approaches.

2.4 Learning

Having selected an appropriate model, the task of inference is still incomplete as different settings of model parameters will yield different probability models. Let X denote the parameter manifold. As different realisations of model parameters lead to different probabilistic models $P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})$, the manifold represents the space of all probability distributions $\mathcal{M}$ that can be generated by a particular model. The goal of learning is to identify a viable candidate $f(\mathcal{O}, \boldsymbol{\theta}) \in \mathcal{M}$ that, when used in conjunction with decision theory allows optimal classifications to be made with respect to a given risk criterion. This section provides an overview of the two fundamental learning frameworks used in machine learning to identify optimal parameter settings for a chosen model.

2.4.1 Generative learning framework

Let D denote the set of training examples where each member $(\boldsymbol{x}, \boldsymbol{y}) \in D$ is a sample from $(\mathcal{X}, \mathcal{Y})$. The generative learning task is specific to generative models where the inference problem $P(\mathcal{Y}|\mathcal{X})$ is solved by modelling the joint probability density function $p(\mathcal{Y}, \mathcal{X})$ instead using observed training examples D. In eqn (2.1), it was observed how the joint probability density can be marginalised and conditioned to yield the necessary posterior probabilities required by decision theory frameworks to make optimum decisions. When used in conjunction with Bayesian decision theory, $p(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}|D, \mathcal{M})$ contains all the information necessary to classify an unlabelled test example $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$:

$$p(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}|D, \mathcal{M}) = \int p_{\boldsymbol{\theta}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}|\mathcal{M})\ p(\boldsymbol{\theta}|D, \mathcal{M}) d\boldsymbol{\theta} \tag{2.19}$$

The posterior probability $p(\boldsymbol{\theta}|D, \mathcal{M})$ encodes the uncertainty associated with the parameters of the model given the training data. Ideally, Bayesian inference should be used to infer the distribution $p(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}|D, \mathcal{M})$. However, the computation of the integral is often intractable due to either the high dimensionality of the parameter space or the complex analytical form of the posterior distribution $p(\boldsymbol{\theta}|D, \mathcal{M})$.

2.4.1.1 Maximum a *Posteriori*

Maximum a *posteriori* (MAP) model estimation can be thought of as an approximation to Bayesian inference, where the posterior distribution $p(\boldsymbol{\theta}|D, \mathcal{M})$ is assumed to be sharply peaked at its mode. In the context where there are large volumes of data, such an approximation is not limiting since by Von Mises's theorem [56]:

$$p(\boldsymbol{\theta}|D, \mathcal{M}) \approx \mathcal{N}(\hat{\boldsymbol{\theta}}, {I_{\hat{\boldsymbol{\theta}}}}^{-1}) \tag{2.20}$$

where $\hat{\boldsymbol{\theta}}$ corresponds to the true parameter that describes the underlying data generation process and $I_{\hat{\boldsymbol{\theta}}}$ is the associated Fisher information matrix (see Sec.6.1.1). Hence,

$$p(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}|D, \mathcal{M}) \simeq p_{\boldsymbol{\theta}_{\text{MAP}}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}|\mathcal{M}) \tag{2.21}$$

where

$$\begin{aligned} \boldsymbol{\theta}_{\text{MAP}} &= \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|D, \mathcal{M}) \\ &= \arg\max_{\boldsymbol{\theta}} \frac{p_{\boldsymbol{\theta}}(D|\mathcal{M})p(\boldsymbol{\theta}|\mathcal{M})}{\int p_{\boldsymbol{\theta}}(D|\mathcal{M})p(\boldsymbol{\theta}|\mathcal{M})d\boldsymbol{\theta}} \end{aligned} \tag{2.22}$$

Since the denominator term is independent of the parameters, it can be shown that

$$\begin{aligned} \boldsymbol{\theta}_{\text{MAP}} &= \arg\max_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(D|\mathcal{M})p(\boldsymbol{\theta}|\mathcal{M}) \\ &= \arg\max_{\boldsymbol{\theta}} \overbrace{\left[\prod_r p_{\boldsymbol{\theta}}(\boldsymbol{x}^r, \boldsymbol{y}^r|\mathcal{M})\right]}^{\text{Data likelihood}} \overbrace{p(\boldsymbol{\theta}|\mathcal{M})}^{\text{Prior distribution}} \end{aligned} \tag{2.23}$$

2.4.1.2 Maximum Likelihood

Maximum likelihood (ML) model estimation can be seen as a simplification of the MAP method by using an uninformative (or uniform) prior over the model parameters. The ML parameter estimate corresponds to:

$$\begin{aligned} \boldsymbol{\theta}_{\text{ML}} &= \arg\max_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(D|\mathcal{M}) \\ &= \arg\max_{\boldsymbol{\theta}} \left[\prod_r p_{\boldsymbol{\theta}}(\boldsymbol{x}^r, \boldsymbol{y}^r|\mathcal{M})\right] \end{aligned} \tag{2.24}$$

In the context of ASR, where the samples $\{(\boldsymbol{x}^r, \boldsymbol{y}^r)\}_r$ represent $\{(\mathcal{O}^r, \mathcal{H}^r)\}_r$, such a framework corresponds to ML acoustic model estimation of HMMs. HMMs assume a particular factorisation of the joint distribution $p(\boldsymbol{y}, \boldsymbol{x}|\mathcal{M}) = p(\boldsymbol{x}|\boldsymbol{y}, \mathcal{M})\ p(\boldsymbol{y})$ where $p(\boldsymbol{y})$ is modelled separately by an LM and is independent of the HMM parameters. In such a setting, acoustic model estimation corresponds to finding parameter settings that maximise the likelihood of an observation sequence given its sentence level HMM.

However, for such latent variable models, maximising the likelihood of the observation sequence is not straight forward. It involves adapting the model parameters such that for each observation the sum over the joint probability of all possible sequence of states in the associated composite sentence HMM and the observation sequence is maximised. For HMMs, this is efficiently achieved by the Baum-Welch algorithm [55]. The algorithm employs the Expectation Maximisation (EM) algorithm [41] to find parameters that locally maximises an auxiliary function. The auxiliary function in question corresponds to a lower bound of the ML objective and maximising it guarantees increment in the data likelihood.

2.4.2 Discriminative Learning

Estimation of the joint density $p(\mathcal{X}, \mathcal{Y})$ is appealing because it leads to informative model estimation. However, it is not clear that this is an efficient use of the training data. Indeed if the goal of the learning process is to solve the inference problem $P(\mathcal{Y}|\mathcal{X})$, it will be more appropriate to directly estimate this probability instead of indirectly learning this function via the intermediate estimation of the joint probability distribution. Moreover, if the underlying generative models used are incorrect, i.e. if the family of joint probability distributions described by the models fail to capture the true data distribution, then the techniques derived from Bayesian inference such as MAP and ML cannot estimate the correct generative distribution, even with unlimited training data [57]. Consequently the performance of the resulting classifier is compromised. In contrast to generative learning, discriminative learning capitalises upon knowledge of the classification task and emphasises on the accurate estimation of class decision boundaries. This is achieved by solving the inference problem $P_{\boldsymbol{\theta}}(\mathcal{Y}|\mathcal{X}, \mathcal{M})$ directly using observed examples D:

$$p(\boldsymbol{y}|\boldsymbol{x}, D, \mathcal{M}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}, \mathcal{M})\ p(\boldsymbol{\theta}|D, \mathcal{M}) d\boldsymbol{\theta} \tag{2.25}$$

with the posterior distribution $p(\boldsymbol{\theta}|D, \mathcal{M})$ now corresponding to:

$$\begin{aligned} p(\boldsymbol{\theta}|D, \mathcal{M}) &= \frac{p_{\boldsymbol{\theta}}(D|\mathcal{M})\ p(\boldsymbol{\theta}|\mathcal{M})}{p(D|\mathcal{M})} \\ &= \frac{p(\boldsymbol{\theta}|\mathcal{M}) \prod_r p_{\boldsymbol{\theta}}(\boldsymbol{y}^r|\boldsymbol{x}^r, \mathcal{M})}{p(D|\mathcal{M})} \end{aligned} \tag{2.26}$$

In the context of ASR, where the samples $\{(\boldsymbol{x}^r, \boldsymbol{y}^r)\}_r$ represent $\{(\mathcal{O}^r, \mathcal{H}^r)\}_r$, this framework is referred to as discriminative sequence training or sequence training. If $\mathcal{M}$ corresponds to a family of generative models then using the discriminative learning framework, the parameters of the model will now characterise the conditional distribution $p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})$ rather than the joint distribution $p_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{x})$. In practice, computing the integral of eqn (2.25) is often intractable and is approximated through variational approaches.

2.4.2.1 Conditional MAP

Conditional Maximum Posteriori (CMAP) [57] is analogous to MAP learning in the generative learning framework where the posterior distribution of model parameters is assumed to be sharply peaked at its mode:

$$p(\boldsymbol{y}|\boldsymbol{x}, D, \mathcal{M}) \simeq p_{\boldsymbol{\theta}_{\text{CMAP}}}(\boldsymbol{y}|\boldsymbol{x}, \mathcal{M}) \tag{2.27}$$

with

$$\boldsymbol{\theta}_{\text{CMAP}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\mathcal{M}) \prod_r^R p_{\boldsymbol{\theta}}(\boldsymbol{y}^r|\boldsymbol{x}^r, \mathcal{M}) \tag{2.28}$$

2.4.2.2 Conditional Maximum Likelihood

Like CMAP, Conditional Maximum Likelihood (CML) is analogoues to ML learning in the generative learning framework, where the prior distribution $p(\boldsymbol{\theta}|\mathcal{M})$ is assumed to be uniform and uninformative. In the context of ASR this is equivalent to Maximum Mutual Information (MMI) [58, 59] discriminative training:

$$\boldsymbol{\theta}_{\text{CML}} = \arg\max_{\boldsymbol{\theta}} \prod_r p_{\boldsymbol{\theta}}(\mathcal{H}^r|\mathcal{O}^r, \mathcal{M}) \tag{2.29}$$

with the CMAP equivalent representing smoothed versions of MMI training [60]. In practice, to prevent numerical underflow, the function that is optimised in MMI training

is:

$$F_{\text{MMI}}(\boldsymbol{\theta}) = \frac{1}{R}\sum_{r}^{R} \log\ \text{p}_{\boldsymbol{\theta}}(\mathcal{H}^{\text{r}}|\mathcal{O}^{\text{r}}, \mathcal{M}) \tag{2.30}$$

Since log is a monotonic function, maximising this function automatically maximises the probability in eqn (2.29). For sequence to sequence discriminative models using the CTC approach, negating the log CML corresponds to the default CTC objective function:

$$\begin{aligned} F_{\text{CTC}}(\boldsymbol{\theta}) &= -\frac{1}{R}\sum_{r}^{R} \log\ \text{p}_{\boldsymbol{\theta}}(\mathcal{H}^{\text{r}}|\mathcal{O}^{\text{r}}, \mathcal{M}) \\ &= -\frac{1}{R}\sum_{r}^{R} \log \left[\sum_{\boldsymbol{\phi}_{1:\text{T}} \in \Phi_{\text{w}_{1:\text{L}}}} \text{P}(\text{w}_{1:\text{L}}|\boldsymbol{\phi}_{1:\text{T}}) \text{P}(\boldsymbol{\phi}_{1:\text{T}}|\mathcal{O}_{1:\text{T}}) \right] \\ &= -\frac{1}{R}\sum_{r}^{R} \log \left[\sum_{\boldsymbol{\phi}_{1:\text{T}} \in \Phi_{\text{w}_{1:\text{L}}}} \prod_{\text{t}} \text{P}_{\boldsymbol{\theta}}(\boldsymbol{\phi}_{\text{t}}|\mathcal{O}, \mathcal{M}) \right] \end{aligned} \tag{2.31}$$

Instead of summing over all possible alignments, if the posterior distribution is assumed to be concentrated solely around its mode then minimising w.r.t. the CTC criterion becomes equivalent to the standard Cross Entropy (CE) training used to train DNNs in Hybrid HMM systems:

$$\begin{aligned} F_{\text{CE}}(\boldsymbol{\theta}) &= -\frac{1}{R}\sum_{r}^{R} \log \prod_{\text{t}} \text{P}_{\boldsymbol{\theta}}(\boldsymbol{\phi}_{\text{t}}|\mathcal{O}, \mathcal{M}) \\ &= -\frac{1}{R}\sum_{r}^{R} \sum_{t} \log\ \text{P}_{\boldsymbol{\theta}}(\boldsymbol{\phi}_{\text{t}}|\mathcal{O}, \mathcal{M}) \end{aligned} \tag{2.32}$$

In practice, the alignments needed for CE training are often provided by an independent system (Eg. HMM-GMM) trained on the same dataset.

2.4.2.3 Minimum Bayes Risk

Under the CML training framework, the goal is to maximise the probability of the correct transcription for each utterance. This is ideal if the goal is to employ the inference model to make as few mis-classifications as possible during classification. However, in ASR, the performance of an classifier is often measured in terms of its ability to achieve low WERs. The WER is the minimum edit distance between the reference transcription and a given word sequence and such a metric weighs some mis-classifications more than others. This forms the basis behind the class of Minimum Bayes Risk (MBR) objective functions [61]

where a loss function $L(\boldsymbol{y}_k, \boldsymbol{y})$ is introduced to formalise the notion of distinguishing between different mis-classifications for each input $\boldsymbol{x}_k$.

The motivation behind MBR learning stems from the application of decision theory to make optimal classifications with respect to a given risk criterion. Equipped with $P(\mathcal{Y}|\mathcal{X})$, the decision stage of classification assigns each $\boldsymbol{x} \in \mathcal{X}$ to one of k classes in $\mathcal{Y}$. This partitions the feature space $\mathcal{X}$ into disjoint *classification* regions $\{\mathcal{R}\}_{i=1}^{K}$ where all points belonging the same region are classified as the same class. Given a set of examples $\{(\boldsymbol{x}^r, \boldsymbol{y}^r)\}_r$ where we already know the true region $\mathcal{R}_j$ for each input $\boldsymbol{x}^r$, the goal of MBR training is to find appropriate model parameters $\boldsymbol{\theta}$ such that for each $\boldsymbol{x}^r$, the weighted loss $\sum_k P_{\boldsymbol{\theta}}(\boldsymbol{y}_k|\boldsymbol{x})L(\boldsymbol{y}_k, \boldsymbol{y}_j)$ is minimised. With respect to a finite training set, the MBR loss objective corresponds to the following empirical loss:

$$F(\boldsymbol{\theta}) = \frac{1}{R}\sum_{r}^{R}\left[\sum_{k} P_{\boldsymbol{\theta}}(\boldsymbol{y}_k|\boldsymbol{x}^r, \mathcal{M})L(\boldsymbol{y}_k, \boldsymbol{y}^r)\right] \tag{2.33}$$

where $\boldsymbol{y}^r$ corresponds to the true class and $\mathcal{M}$ corresponds to our choice of model. In the context of ASR, this corresponds to:

$$F_{\text{MBR}}(\boldsymbol{\theta}) = \frac{1}{R}\sum_{r}^{R}\left[\sum_{\mathcal{H}} P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}^r, \mathcal{M})L(\mathcal{H}, \mathcal{H}^r)\right] \tag{2.34}$$

Broadly speaking MBR training can be seen to concentrate probability mass: a sufficiently flexible model trained to convergence with MBR will assign a high probability to those hypothesis that have the smallest loss.

2.4.2.4 Lattices

From (2.30) and (2.34), it can be seen that at each iteration of discriminative sequence training, the posterior distribution $P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})$ must be computed explicitly. For hybrid HMM based generative models, the required distribution is computed using Bayes rule:

$$P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}) = \frac{p_{\boldsymbol{\theta}}(\mathcal{O}, \mathcal{H})}{\sum_{\mathcal{H}} p_{\boldsymbol{\theta}}(\mathcal{O}, \mathcal{H})} \tag{2.35}$$

The computation of the denominator term involves taking into account composite sentence models of all possible transcriptions. For LVCSR, such a computation is very expensive. The standard approach is to approximate this probability by doing a recognition pass on each training utterance and considering only a subset of hypotheses that are most likely. An efficient way to represent these competing hypotheses is to encode them in

a lattice. A lattice is a compact representation of multiple overlapping hypotheses. It takes the form of an acyclic graph structure where nodes represent the ends of words at particular points in time, while the arcs that connect the nodes represent particular word pronunciations. Compared to 1-best or n-best hypotheses, the use of such a structure has been shown to improve robustness and task performance since it preserves ambiguity and avoids making hard decisions too early [62, 63]. The lattice arcs are normally annotated with the associated acoustic and language model scores. The use of lattices also extend to the numerator term to accumulate the necessary numerator statistics needed for discriminative sequence training. In cases where the recogniser-generated denominator lattice does not contain the correct sequence, a consolidated lattice is often formed by merging the recogniser lattice with the numerator lattice [64].

When constructing a consolidated lattice, the set of chosen hypotheses depends on how the AM interacts with the LM when traversing though the recognition trellis. For generative models such as (2.5), the language model probabilities are only considered for trellis nodes that match the final states of words. To increase the contribution of the LM, the LM probabilities are often scaled by a positive integer (commonly termed the grammar scale factor) [65] which is greater than one. Subsequently, the acoustic model log likelihood values are often scaled with the inverse of the LM scale factor. This has the effect of increasing the number of confusable paths in the lattice and has been found to improve the overall generalisation performance from discriminative sequence training [66].

The necessary training statistics needed to train the acoustic model parameters can be computed efficiently by an application of the Baum Welch algorithm. For GMM models, an extended form of the algorithm called the Extended Baum-Welch algorithm [58] is used to accumulate the necessary training statistics. Ideally, after each iteration of discriminative training, it is desirable to perform a recognition pass on all the training data and regenerate the lattice. While this is viable for small vocabulary tasks, it is too computationally expensive for large vocabulary tasks. The standard approach is to generate the lattices once using a CE trained HMM-DNN model and then proceed to use this lattice repeatedly at every iteration of discriminative sequence training.
Recently in [67], it has been shown that the necessary statistics required from the denominator term in (2.35) can be computed efficiently without the need of the lattice structure by loading the denominator graph directly on the GPU at each iteration of discriminative sequence training. The proposed approach relies on a combination of factors ranging from making the denominator graph small through simplifying the HMM

topology, reducing the frame rate, using a phone LM to using a modified objective function that integrates CE cost criterion.

2.4.2.5 Choices of Loss Functions for MBR Methods

There are different approaches for computing the loss $L(\mathcal{H}, \mathcal{H}^r)$ associated with a proposed hypothesis $\mathcal{H}$ for a given utterance r. These approaches differ at the level of granularity used to compute an appropriate mismatch between the proposed hypothesis and the correct hypothesis. Based on the level of granularity traversed, the loss functions currently used can be grouped into the following classes:

1. Sentence-level loss: this corresponds to 0 or 1 loss :

$$L(\mathcal{H}, \mathcal{H}^r) = \begin{cases} 0 & \mathcal{H} = \mathcal{H}^r \\ 1 & \text{otherwise} \end{cases}$$

   With respect to such a loss, an utterance is either correct or incorrect. In practice, this is not ideal as some mis-classifications can be more important than others.

2. Word level loss: in ASR, the performance of a classifier is often measured in terms of its ability to achieve low WERs. When the MBR criterion is equipped with the WER loss, then the method is referred to as the Minimum Word Error Rate (MWE) criterion [68]. The WER is the Levenshtein distance between $\mathcal{H}$ and $\mathcal{H}^r$ and corresponds to the minimum number of edits needed to convert the transcribed utterance into the reference transcription. The algorithm works within a dynamic programming framework, where it computes this distance by building a trellis to hold intermediate values. The trellis in this scenario can be viewed as a two-dimensional grid where the y-axis denotes the word sequence associated with the proposed hypothesis and the x-axis denotes the word sequence corresponding to the correct hypothesis, with each cell $(w_k^{\mathcal{H}}, w_j^{\mathcal{H}^r})$ holding the distance between the first k words of $\mathcal{H}$ and the first j words of $\mathcal{H}^r$. The cost of employing such a metric linearly scales with the length of the sequences. Thus making the use of such a metric quite computationally intensive. To reduce the computational cost, in recent work [69], it has been shown that by restricting training to only sampled paths in a lattice, the model can still be trained effectively using the Levenshtein distance as the loss metric.

3. Phone/State level loss: under the lattice based framework, not all word sequences will be observed. To assist generalisation, the loss function is often computed at a finer level of granularity such at phones or sub-phones rather than at word level. Applying the Levenshtein distance between every path in the phone/sub-phone marked lattice and the reference label sequence involves an impracticably large amount of computation. To reduce this computational overhead, the standard approach is to use an alignment-based error metric that approximates the accuracy $A(\mathcal{H}, \mathcal{H}^r)$ [66].

   The accuracy $A(\mathcal{H}, \mathcal{H}^r)$ is defined as the number of correct labels in $\mathcal{H}$ minus the number of inserted labels in the sequence and is related to the Levenshtein distance between the reference sequence $\mathcal{H}^r$ and hypothesis sequence $\mathcal{H}$ by:

$$\text{Lev}(\mathcal{H}, \mathcal{H}^{\text{r}}) = \text{M} - \text{A}(\mathcal{H}, \mathcal{H}^{\text{r}}) \tag{2.36}$$

   where M is the number of labels in the reference sequence. In standard MBR training, the accuracy $A(\mathcal{H}, \mathcal{H}^r)$ is approximated as the sum of local arc accuracies computed i.e $A(\mathcal{H}, \mathcal{H}^r) \approx \sum_{q \in \mathcal{H}_{\text{lattice}}} \text{arcAcc(q)}$.

   There are two main approaches in computing the accuracy arcAcc(q):

   (a) In the first approach, given hypothesis arc q, arcs associated with the reference hypothesis are found that overlaps in time with q. For each such arc z, the quantity $e(q, z)$ that denotes the proportion of the length of z that is overlapped with q is computed. Using this quantity, arcAcc(q) is then computed as follows:

$$\text{arcAcc(q)} = \arg\max_{\text{z}} \begin{cases} -1 + 2e(q, z) & \text{if z = q} \\ -1 + e(q, z) & \text{otherwise} \end{cases}$$

   This particular form of MBR training when applied at the level of phone sequences is known as Minimum Phone Error(MPE) [70] training. In [57, 66] it is shown that such a loss divergence acts as an upper bound to $\text{Lev}(\mathcal{H}, \mathcal{H}^{\text{r}})$ where it penalises insertions more than deletions.

   (b) An alternative approach to computing arcAcc(q) is to compute the accuracy on a frame by frame basis. For each hypothesis arc q,

$$\text{arcAcc(q)} = \sum_{\text{t=start}_{\text{q}}}^{\text{end}_{\text{q}}} \begin{cases} 1 & \text{if } q \equiv z \text{ for any arc z overlapping t} \\ 0 & \text{otherwise} \end{cases}$$

where $start_q$ and end_q correspond to the start and end frames of arc q, and z corresponds to any arc in the reference lattice. Under this construction, for each arc q in the recognition lattice, the raw accuracy is computed by counting the number of frames between $start_q$ and end_q where q agrees with an arc in reference lattice that shares the same phonetic identity as q. When the individual arcs correspond to phone arcs, this variant of MBR training is called Minimum Phone Frame Error(MPFE) [71] whereas when the arcs correspond to tied sub-phone states, this form of MBR training objective is popularly known as State Minimum Bayes Risk (s-MBR) [64].

2.5 Decoding

Equipped with the inference model $P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})$, the decision stage of an ASR system employs decision theory to find the 'best' word sequence (hypothesis) for a given observation sequence. In speech recognition, this process is commonly known as decoding. The decoding process relies on creating a finite state graph structure that integrates the acoustic model, dictionary and language model. This graph can be constructed statically [72, 73], dynamically [74], or in a hybrid manner that dynamically adds language model information to a static graph [75, 76]. Using an appropriate graph structure, decoding in ASR is performed by using one of following the two approaches: Viterbi decoding or Minimum Based Risk (MBR) decoding [77]. This section provides a brief review of the two methods with particular emphasis on how Viterbi decoding is performed with HTK [47].

2.5.1 Viterbi decoding

Viterbi decoding belongs to the class of classification approaches whose goal is to employ the inference model to make as few mis-classifications as possible:

$$\hat{\mathcal{H}} = \arg\max_{\mathcal{H}} P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}) \tag{2.37}$$

For generative based inference models, this is equivalent to selecting:

$$\hat{\mathcal{H}} = \arg\max_{\mathcal{H}} P_{\boldsymbol{\theta}}(\mathcal{H}, \mathcal{O}) \tag{2.38}$$

This particular approach to decoding assumes that the underlying inference model used takes the form of discrete latent variable graphical model discussed in Sec. 2.2.1 and Sec. 2.2.2 and solves the classification problem by finding the most likely latent variable sequence $\hat{\boldsymbol{\Phi}}_{1:T}$. With respect to hybrid HMMs combined with LMs, this corresponds to finding:

$$\hat{\boldsymbol{\Phi}}_{1:T} = \arg\max_{\boldsymbol{\Phi}_{1:T}} P_{\boldsymbol{\theta}}(\mathcal{O}, \mathcal{H}, \boldsymbol{\Phi}_{1:T}) \tag{2.39}$$

As it is impractical to perform an exhaustive search over all possible latent variable sequences, in practice, such a task is efficiently performed by employing the *Viterbi* algorithm [78]. The *Viterbi* algorithm is variant of the forward-backward algorithm that uses a trellis to store intermediate values as it processes a given observation sequence. In its case, each cell in the trellis corresponds to

$$\boldsymbol{v}_t(i) = \arg\max_{\boldsymbol{\Phi}_{1:t-1}} p_\theta(\boldsymbol{o}_1, \boldsymbol{o}_2 \cdots \boldsymbol{o}_t, \boldsymbol{\phi}_{1:t-1}, \boldsymbol{\phi}_t = i)$$

the probability of being state j at time t after seeing all observations up to time t and passing through the most probable sequence $\boldsymbol{\Phi}_{1:t-1}$. This term can be recursively written as:

$$\boldsymbol{v}_t(j) = \max_{i \in M_H} \boldsymbol{v}_{t-1}(i) a_{ij} b_j(\boldsymbol{o}_t) \tag{2.40}$$

where $b_j(\boldsymbol{o}_t) = p_\theta(\boldsymbol{o}_t|\bar{\mathcal{O}}, \boldsymbol{\phi}_t = j)$ and $a_{ij} = P_\theta(\boldsymbol{\phi}_t|\boldsymbol{\phi}_{t-1})$ for hybrid HMMs. To prevent numerical underflow, in practice $\boldsymbol{v}_t(j)$ is computed as:

$$\boldsymbol{v}_t(j) = \arg\max_{i \in M_H} \{\boldsymbol{v}_{t-1}(i) + \log a_{ij}\} + \log\ \mathrm{b_j(o_t)} \tag{2.41}$$

In practice, a direct implementation of the Viterbi algorithm results in a large search space for continuous speech. This is because to handle continuous speech, the context-dependent phone acoustic models need to integrated with a bigger language model and a pronunciation dictionary[1] to cover different pronunciations and larger vocabulary. This however results in the set M_H being very large. In HTK, a practical implementation of the *Viterbi* algorithm employs a *token passing* model [79]. At each time t, each HMM state j holds a moveable token that contains $\boldsymbol{v}_t(j)$ and the history to achieve it. At each subsequent time step, the token's value is increased by $\log\ \mathrm{a_{jk}} + \log\ \mathrm{b_j(o_t)}$ before being passed to all states succeeding the current state. After receiving incoming tokens from all preceding states, a successor state keeps only the token with the highest value and

[1] maps sequences of phonemes to words

discards the rest. When multiple HMMs are considered using composite HMMs (2.2.1), tokens can be easily passed from an exit state of one HMM to the start state of another.

If two HMMs belong to different words, say $\boldsymbol{w}_l$ and $\boldsymbol{w}_{l+1}$, the passing of the token corresponds to a word transition and log $\mathrm{a_{jk}}$ corresponds to the LM score. This enables the token passing algorithm to search for the maximum posterior probability rather than the maximum likelihood path. The LM log-probability is usually linearly scaled by the grammar scaling factor [65] when combined with the acoustic log-likelihood. This is necessary since HMM acoustic models often produce a wider dynamic range of likelihood values due to the underestimation of likelihood arising from invalid assumptions [60]. At the end of the search, every utterance is assumed to end with silence and the best path with the highest probability is acquired only from the tokens staying in the exit states of the silence HMM at time T. In addition, the token passing model is also useful for finding N-best paths by allowing each model state to hold multiple tokens to increase the number of different token histories that can be maintained. Here tokens from different preceding words are considered as distinct.

2.5.2 Minimum Bayes Risk decoding

Viterbi decoding doesn't distinguish between different degrees of utterance level mis-classification and hence weights all utterance level mis-classifications to be equal. In ASR, the performance of an classifier is often measured in terms of its ability to achieve low WERs. The WER is the edit distance between the true word sequence and the most probable word sequence emitted by the transcriber and under such a metric some mis-classifications are given more weight than others. To address this mismatch between the training and evaluation objective, we formalise the notion of distinguishing between different mis-classifications through the introduction of a loss function $L(\boldsymbol{y}_k, \boldsymbol{y}_j)$. This forms the basis behind the Minimum Bayes Risk (MBR) decoding [77].

MBR decoding mirrors Minimum Bayes Risk training discussed in Sec. 2.4.2.3. However, in MBR decoding, the knowledge of the true class is not available. Equipped with $P(\mathcal{Y}|\mathcal{X})$, the decision stage of classification assigns each $\boldsymbol{x} \in \mathcal{X}$ to one of k classes in $\mathcal{Y}$. Essentially, this corresponds to partitioning the feature space of $\mathcal{X}$ into *classification* regions $\{\mathcal{R}\}_{i=1}^{K}$ where all points belonging the same region are classified as the same class. For any $\boldsymbol{x}$, a mis-classification occurs when $\boldsymbol{x}$ is assigned to a region $\mathcal{R}_j$ that doesn't correspond to its real class with $L(\boldsymbol{y}_k, \boldsymbol{y}_j)$ depicting how significant such a mis-classification is. At prediction, the goal is therefore to make *decisions* such that we minimise the total loss. However, the loss function depends on the knowledge of the

true class, which is unknown. For a given input vector $\boldsymbol{x}$, the uncertainty in the true class is expressed through the joint probability distribution $p(\boldsymbol{x}, \boldsymbol{y})$ which in fact can be seen to depend on $P(\boldsymbol{y}|\boldsymbol{x})$ as the factor $p(\boldsymbol{x})$ is common to all classes. This leads to minimisation of the expected loss:

$$E[\mathbb{L}] = \sum_j \int_{\mathcal{R}_j} \sum_k P(\boldsymbol{y}_k|\boldsymbol{x}) L(\boldsymbol{y}_k, \boldsymbol{y}_j) p(\boldsymbol{x}) dx \tag{2.42}$$

where $\boldsymbol{y}_k$ denotes the true class. The goal of classification is then to choose for each $\boldsymbol{x}$ a region $\mathcal{R}_j$ such that $\sum_k P(\boldsymbol{y}_k|\boldsymbol{x}) L(\boldsymbol{y}_k, \boldsymbol{y}_j)$ is at minimum. In ASR, where $(\boldsymbol{x}, \boldsymbol{y}) \equiv (\mathcal{O}, \mathcal{H})$, such a approach is equivalent to choosing $\hat{\mathcal{H}}$ such that $\sum_{\mathcal{H}} P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}) L(\mathcal{H}, \hat{\mathcal{H}})$ is smallest.

As written above, the MBR criterion is too computationally expensive to implement directly as it involves a search over all possible word sequences to find the true minimum. Thus, the search is normally restricted to a subset of hypotheses by using recognition lattice. However, even when the hypotheses spaces are represented by recognition lattices, there may still be too many hypotheses to perform a full search. In practice, it is usually the case that the lattice is compressed in some way to reduce the search space. For example, pinched lattices [80], N-best lists [81] and confusion networks [82] have a considerably smaller search space than recognition lattices. It should be also mentioned that in [83], decoding is performing efficiently through using a recursive edit distance.

2.6 Summary

This chapter presented a review of the main components of an ASR system that are relevant to this thesis. The chapter began with a brief discussion of the front end processing stage of an ASR system, mainly focusing on MFCCs, filter banks and PLPs. This was followed by a review of the different types of models used to solve the ASR inference problem, with particular emphasis on the hybrid HMM-DNN model. A distinction between generative and discriminative learning frameworks was presented followed by a more detailed discussion of the different forms of discriminative MBR training. The chapter concluded with a discussion of the existing decoding frameworks used to extract the most likely hypothesis for a given acoustic waveform.

Chapter 3

Deep Neural Networks: an overview

This chapter presents a literature review of Deep Neural Networks within the context of acoustic modelling in ASR. The chapter begins with the definition of an Artificial Neural Network followed by a review of the Multi Layer Perceptron (MLP) model. The importance of starting DNN training from good initialisation is highlighted in Sec. 3.3. A review of the standard DNN architectures currently used for acoustic modelling is presented in Sec. 3.4. The section also discusses how the particular choice of DNN architecture controls the representational capability of a chosen model. The problem of model selection is presented in Sec. 3.5 while the back-propagation algorithm along with the gradient of various sequence level losses in presented in Sec. 3.6. The chapter ends with a discussion of the vanishing and exploding gradients problem that plagues gradient based training of deep models, and reviews the various approaches used to alleviate these issues.

3.1 ANN Definition

An Artificial Neural Network (ANN) is a form of connectionist[1] computational model loosely inspired from the compositional structure of biological neural networks. From a neuroscience point of view, Haykin [84] provides the following definition of an ANN:

'An Artificial Neural Network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. Such a model resembles the brain in two respects:

[1]Connectionism corresponds to a set of approaches in the fields of artificial intelligence, cognitive science and philosophy of mind, that attempts to represent mental or behavioural phenomena as emergent processes of interconnected networks of simple units.

1. Knowledge is acquired by the network through a learning process.

2. Interneuron connection strengths known as synaptic weights are used to store the knowledge.'

Under this interpretation, an ANN model can be viewed as a particular form of an input/output system with many simple processors, each having a small amount of local memory. These units are connected by communication channels which are pre-specified by the network topology.

However, as modern DNN research is guided by mathematical and engineering disciplines, this work will adopt Bishop's [85] mathematical view of an ANN model: 'An Artificial Neural Network is a powerful framework for representing non-linear mappings from several input variables to several output variables, where the form of the mapping is governed by a number of adjustable parameters.'

Adopting Bishop's viewpoint, a mathematical formulation of an ANN model is as follows. Given $\boldsymbol{x} = (\boldsymbol{x}_1 \cdots ..., \boldsymbol{x}_n) \in \mathcal{X}$ and some process that results in a corresponding set of outputs $\boldsymbol{y} = (\boldsymbol{y}_i, ...\boldsymbol{y}_m) \in \mathcal{Y}$, the goal is to model the underlying relationship between $\mathcal{X}$ and $\mathcal{Y}$ using some form of mathematical function ϕ i.e

$$\boldsymbol{y} = \phi(\boldsymbol{x})$$

The function ϕ may be very complicated and more importantly it is not expected that it can be computed exactly. In machine learning, the general practice is to use some form of mathematical model to capture the underlying relationship between $\mathcal{X}$ and $\mathcal{Y}$. ANNs belong to particular classes of parametric functions $f(\boldsymbol{x}, \boldsymbol{\theta})$ that aim to capture the relationship between $\mathcal{X}$ and $\mathcal{Y}$ by imposing various rules and regulations to optimise the choice of parameters. The distinctive aspect of these models is that they are constructed from the addition, multiplication and composition of parameterised adaptive basis functions [85, 86] of the following form:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma\left(\boldsymbol{\theta}^T(\boldsymbol{x}) + \theta\right) \tag{3.1}$$

where σ corresponds to a map from $\mathbb{R} \to \mathbb{R}$ and in neural network literature is common referred to as the activation function. Restricting the choice of basis functions to have the above form has special significance. In [87], it is shown that as long as the choice of σ corresponds to any Riemann Integral function that is not a polynomial, then the space

of functions given by:

$$M(\sigma) = \text{span}\{\sigma\left(\boldsymbol{\theta}^{\mathrm{T}}(\mathrm{x}) + \theta\right) : \boldsymbol{\theta} \in \mathbb{R}^{\mathrm{k}}, \theta \in \mathbb{R}\}$$

is topologically dense in the space of space of continuous functions defined on any compact subspace in the underlying domain. In simpler terms, this means that as long σ is continuous and not a polynomial, the behaviour of any continuous function on a closed disk in the Euclidean space can be closely modelled to a high accuracy by a member from $M(\sigma)$. This property allows ANNs to be *Universal Function Approximators* [87, 88].

3.2 Multilayer Perceptron

The quintessential example of an ANN model is the multilayer perceptron (MLP) [89]. The most basic form of the model consists of the serial composition of the basis functions described in (3.1) in vector valued form:

$$F(\boldsymbol{x}, \boldsymbol{\theta}) = z \circ f^{L-1} \cdots \circ f^{1}$$

where each f^l represents a bivariate map

$$f^l(\boldsymbol{x}, \boldsymbol{\theta}^l) : \mathbf{R}^{d_{l-1}} \times \mathbf{R}^{d_l \times (d_{l-1}+1)} \to \mathbf{R}^{d_l}$$

defined by:

$$f^l(\boldsymbol{x}, \boldsymbol{\theta}^l) = \sigma \odot \left(\boldsymbol{W}_{\boldsymbol{\theta}}^l \boldsymbol{x} + \boldsymbol{b}_{\boldsymbol{\theta}}^l\right) \tag{3.2}$$

Here

- d_{l-1} represents the dimensionality (number of nodes) of the layer at depth $l-1$
- d_l is the dimensionality (number of nodes) of the layer.
- $\odot$ denotes the point wise application of the function with the vector obtained from the affine transformation of $\boldsymbol{x}$.

In ANN terminology, these individual compositional maps f^l are popularly termed as layers. Each layer essentially represents a function that maps the previous layer's representation $\boldsymbol{x}$ to a new space $\mathbf{R}^{d_l}$. In the above formulation, the map $\boldsymbol{z}$ represents output of the MLP. For regression tasks, this corresponds to linear map of the penultimate

layer's representation. Whereas for classification tasks such as in ASR, by applying the softmax function on the linear map, the output models $P_\theta(\boldsymbol{\phi}_t|\boldsymbol{o}_t^r, \bar{\mathcal{O}})$, the posterior distribution of clustered states given the temporal context $\bar{\mathcal{O}}$ of the current frame t:

$$P_\theta(\boldsymbol{\phi}_t = i|\boldsymbol{o}_t^r, \bar{\mathcal{O}}) = \frac{\exp\left(\boldsymbol{\theta}_i{}^T(\boldsymbol{x}) + \theta)\right)}{\sum_j \exp\left(\boldsymbol{\theta}_j{}^T(\boldsymbol{x}) + \theta)\right)} \tag{3.3}$$

Here j denotes the dimension of the output layer and $\boldsymbol{x}$ represents the penultimate layer's representation. The dependency on t here has been dropped at the same function is applied to yield posteriors for every frame. Figure 3.1 shows an example of a standard feedforward network. In this work when presenting experimental results, the MLP model will be referred to as a DNN.

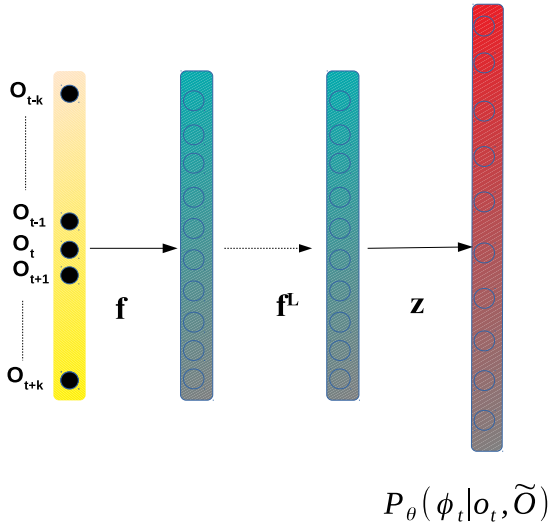


Fig. 3.1 Architecture of Multi Layer Perceptron (MLP).

3.3 Importance of Initialisation

Through compositionality, MLPs with more hidden layers possess the ability to learn much richer representations of the input. This makes such models quite attractive in

solving complex tasks. However until a decade ago it wasn't clear how to harness this rich representation capability of deep models: adding layers beyond 4 or 5 was found to both impede and de-stabilise SGD learning [90, 91][2].

The current resurgence of deep learning came from introducing strategies that focus on finding parameter initialisations that begin training in a region connected to the solution such that the path can be uncovered by local steepest descent. In neural network literature, this is termed as pre-training [93] . The motivation behind pre-training comes from the following observation: sometimes directly training a model to solve a specific task can be challenging if the model is complex and difficult to optimise. It can be more effective to either train a simpler model first and then gradually make it more complex or use the model to solve a simpler task before moving to the final task. Depending on the model type and task in question, the first or the latter approach to pre-training is used.

For training discriminative models to solve non-sequence classification tasks, initialising model parameters by either using layer by layer discriminative pre-training or stacking restricted Boltzmann machines (RBMs) [94, 95] have been observed to greatly improve the classification performance[3]. This strategy is equivalent to training a simple model first and then gradually make the model more complex. For training hybrid HMM-DNN models to solve the ASR task, initialising the DNN model parameters with CE or CTC training have been shown to lead to better overall WER convergence [69, 96]. This strategy can be thought of as using the model to solve a simpler task first before moving to a complex task. Following this approach, prior to sequence training in this work all DNN models will be initialised with CE training using standard SGD.

3.4 DNN Architectures used in ASR

When applied to acoustic modelling, DNNs represent discriminative models whose output $\boldsymbol{z}_t^r$ represents $P_\theta(\boldsymbol{\phi}_t|\boldsymbol{o}_t^r, \bar{\mathcal{O}})$ with $\bar{\mathcal{O}}$ presenting the choice of temporal context. The degree to which a DNN model can incorporate temporal contextual information is determined by its architecture. For Networks with feedforward architectures, $\bar{\mathcal{O}}$ corresponds to a splice $\boldsymbol{o}_{[t-k,t+k]}$ whereas for DNNs possessing recurrent connections, $\bar{\mathcal{O}}$ corresponds to either $\boldsymbol{o}_{1:t-1}$ or $\boldsymbol{o}_{1:T}$ (bi-directional). This section presents a review of the two standard architectures currently used in ASR.

[2]Notable exceptions include work on convolutional networks [92]

[3]It should be mentioned that for models using the ReLU activation function, good convergence with CE training can be achieved without any form of pre-training

3.4.1 Sub-sampled Time Delay Neural Networks

A Time Delay Neural Network (TDNN) [97, 98] corresponds to a MLP model that has been adapted to

- classify patterns with time shift-invariance.
- model context at each layer of the network.

Figure 3.2 shows an example of a TDNN network.

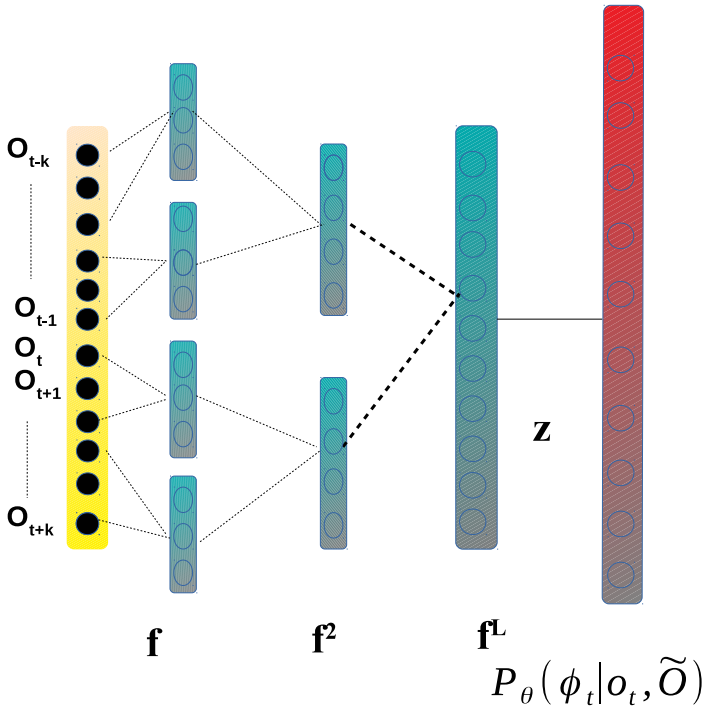


Fig. 3.2 Example of a sub-sampled TDNN network.

In this model, the initial transforms are learned on narrow contexts while the deeper layers process hidden activations from a wider temporal context. Table 3.1 compares how a DNN layer fares against a TDNN layer (given by Table 3.2). When operating on

the same temporal context, a TDNN layer in contrast to a standard DNN layer applies the initial affine transform on a narrower context. The activations associated with these transforms are cached and are utilised by the deeper layers in the network. This allows the space of functions that can be modelled by a TDNN to be much richer than an equivalent DNN having the depth.

Layer 1

$$
\begin{aligned}
\boldsymbol{a}_t^1 &= \boldsymbol{W}_{\boldsymbol{\theta}}^1 \begin{bmatrix} \boldsymbol{o}_{t-2} \\ \boldsymbol{o}_{t-1} \\ \boldsymbol{o}_t \\ \boldsymbol{o}_{t+1} \end{bmatrix} \\
&= \begin{bmatrix} \overbrace{\boldsymbol{W}_{\boldsymbol{\theta},1}^1}^{K \times D} & \boldsymbol{W}_{\boldsymbol{\theta},2}^1 & \boldsymbol{W}_{\boldsymbol{\theta},3}^1 & \boldsymbol{W}_{\boldsymbol{\theta},4}^1 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} \boldsymbol{o}_{t-2} \\ \boldsymbol{o}_{t-1} \\ \boldsymbol{o}_t \\ \boldsymbol{o}_{t+1} \end{bmatrix} \\
\boldsymbol{f}_t^1 &= \sigma \odot \boldsymbol{a}_t^1
\end{aligned}
$$

Layer 2

$$
\boldsymbol{a}_t^2 = \boldsymbol{W}_{\boldsymbol{\theta}}^2 \boldsymbol{f}_t^1
$$

Table 3.1 Example of standard DNN layer: K denotes the dimension of Layer 1 and D denotes the dimension of the individual vectors $\boldsymbol{o}_t$.

Layer 1

$$\boldsymbol{a}^1_{t-1} = \boldsymbol{W}^1_{\boldsymbol{\theta}} \begin{bmatrix} \boldsymbol{o}_{t-1} \\ \boldsymbol{o}_t \end{bmatrix} = \begin{bmatrix} \overbrace{\boldsymbol{W}^1_{\boldsymbol{\theta},1}}^{K\times D} & \boldsymbol{W}^1_{\boldsymbol{\theta},2} \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} \boldsymbol{o}_{t-2} \\ \boldsymbol{o}_{t-1} \end{bmatrix}$$

$$\boldsymbol{f}^1_{t-1} = \sigma \odot \boldsymbol{a}^1_{t-1}$$

$$\boldsymbol{a}^1_t = \boldsymbol{W}^1_{\boldsymbol{\theta}} \begin{bmatrix} \boldsymbol{o}_t \\ \boldsymbol{o}_{t+1} \end{bmatrix} = \begin{bmatrix} \overbrace{\boldsymbol{W}^1_{\boldsymbol{\theta},1}}^{K\times D} & \boldsymbol{W}^1_{\boldsymbol{\theta},2} \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} \boldsymbol{o}_t \\ \boldsymbol{o}_{t+1} \end{bmatrix}$$

$$\boldsymbol{f}^1_{t+1} = \sigma \odot \boldsymbol{a}^1_t$$

Layer 2

$$\boldsymbol{a}^2_t = \boldsymbol{W}^2 \begin{bmatrix} \boldsymbol{f}^1_{t-1} \\ \boldsymbol{f}^1_t \end{bmatrix} = \begin{bmatrix} \overbrace{\boldsymbol{W}^2_{\boldsymbol{\theta},1}}^{P\times K} & \boldsymbol{W}^2_{\boldsymbol{\theta},2} \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} \boldsymbol{f}^1_{t-1} \\ \boldsymbol{f}^1_{t+1} \end{bmatrix}$$

Table 3.2 Example of a TDNN layer operating on the same spliced input vectors, with P denoting the dimension of Layer 2. By having the initial transforms act on narrower contexts, the next layer can combine information from frames in a much richer way.

As the transforms in the TDNN architecture are tied across time steps, such a model is often seen as a precursor to the Convolutional Neural Networks (CNNs) [99]. During back-propagation (Sec. 3.6.1), as a consequence of parameter tying, the lower layers of the network are updated by a gradient accumulated over all the time steps of the input temporal context. This forces these layers to learn time invariant feature transforms.

The original TDNN formulation [97, 98] uses shifts of one frame in time for the individual hidden layers. This is very expensive both in terms of computation and the numbers of parameters when operating on large temporal contexts. In [100], it is shown that improved performance over standard DNNs can still be achieved by introducing gaps between the frames instead of splicing together contiguous temporal windows of frames at each layer. Such a modification results in a sub-sampled TDNN. The network shown in Figure 3.2 is an example of such a network. In recent work [101], it has been shown that by employing factorised semi-orthogonal weight matrices, such networks can achieve similar gains in WER as models that employ recurrent networks (discussed in next section) . Building upon the work in [100], this work uses the following context

specification for the various TDNN layers: [-2, +2][4] for layer 1, $\{-1, 2\}$[5] for layer 2, $\{-3, 3\}$ for layer 3, $\{-7, 2\}$ for layer 4 and $\{0\}$ for the remaining deeper layers.

3.4.2 Recurrent Neural Networks

Human speech is inherently a complex time-varying signal which contains intricate correlations at a range of different time scales. The sequential nature of speech makes it inadequate for feedforward architectures to model the underlying system as such models by design treat each input in a sequence independently. Recurrent neural networks (RNNs) [102–105] represent a family of neural networks that has been originally designed to process sequential data. Like TDNNs, these models take the advantage of one of the early ideas found in machine learning: sharing parameters across different parts of a model. However, contrast to TDNN's feed forward architecture, the particular architecture of a RNN allows greater parameter sharing across different parts of the model. This allows the model to share statistical strength [106] across both different sequence lengths and different positions in time [107].

This work will view RNNs as functional approximators of the underlying state transition function associated with a deterministic dynamical system. A dynamical system can be viewed as a tuple (S, f) where S corresponds to appropriate state space modelled by a geometric manifold and the function f defines the underlying the state transition. At any given time, the dynamical system has a state $\boldsymbol{h}_t$, a tuple of real numbers (a vector) that corresponds to a point in S. The state transition function f describes the evolutionary rule the governs the future state that can follow from the current state $\boldsymbol{h}_t$:

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}; \boldsymbol{\theta}) \tag{3.4}$$

[4] consider all contiguous frames with in this interval

[5] consider only activations associated with the time steps given in this set.

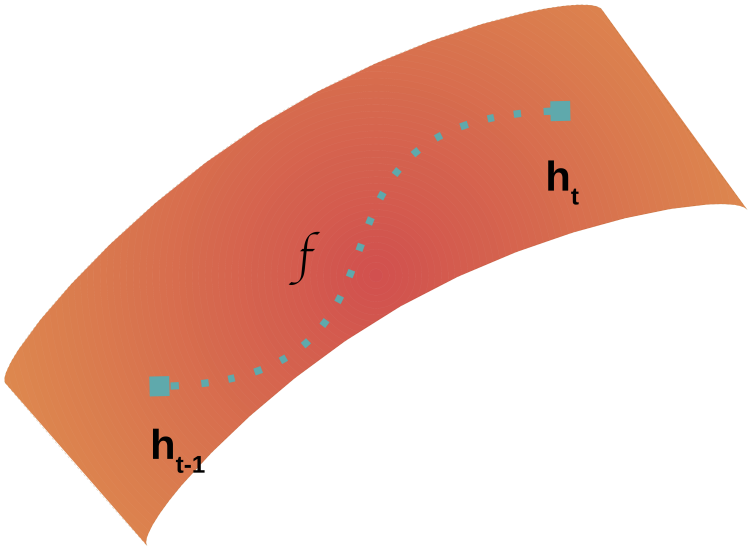


Fig. 3.3 Transition in a dynamical system

RNNs in particular describe a particular form of deterministic dynamical system where the evolutionary rule f utilises both the previous state $\boldsymbol{h}_{t-1}$ and an external stimulas $\boldsymbol{o}_t$ to decide on the next state $\boldsymbol{h}_t$:

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{o}_t; \boldsymbol{\theta}) \tag{3.5}$$

Equation (3.5) is recurrent because the definition of $\boldsymbol{h}_t$ refers backs to the same definition at time $t-1$. For a finite number of time steps t, by recursively applying the definition of the hidden state $t-1$ times, the transition function can be unrolled as:

$$\begin{aligned} \boldsymbol{h}_t &= f(\boldsymbol{h}_{t-1}, \boldsymbol{o}_t; \boldsymbol{\theta}) \\ &= f(f(f(\cdots, \boldsymbol{x}_{t-2}; \boldsymbol{\theta}), \boldsymbol{o}_{t-1}; \boldsymbol{\theta}), \boldsymbol{o}_t; \boldsymbol{\theta}) \qquad (3.6) \\ &= g(\boldsymbol{o}_1, \boldsymbol{o}_2 \cdots \boldsymbol{o}_t; \boldsymbol{\theta}) \qquad (3.7) \end{aligned}$$

Equation (3.6) shows how the computation of the current state $\boldsymbol{h}_t$ can represented as a form of a unrolled computational graph (shown in figure 3.4) by repeated application of the function f. Furthermore from eqn (3.7), it can be seen that RNNs model a specific form of state transition function that can take the whole past sequence as input to decide on the current state $\boldsymbol{h}_t$.

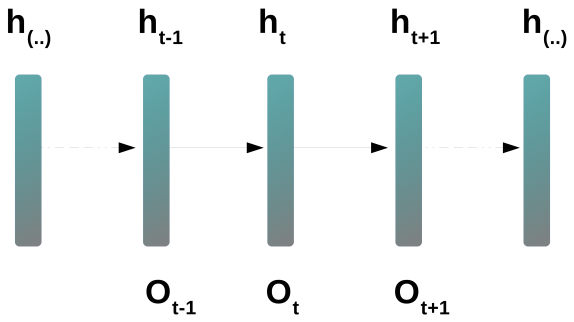


Fig. 3.4 Example of standard RNN network

To draw comparisons with feedforward networks, in this work a RNN layer will viewed as feedforward Network constructed for the repeated composition of single modular function f.

3.4.2.1 Form of RNN state transition function

For generic RNNs, the convention form of the state transition function is:

$$\begin{aligned} \boldsymbol{h}_t &= \sigma \odot \left[\boldsymbol{W}_{\boldsymbol{\theta}} \begin{bmatrix} \boldsymbol{o}_t \\ \boldsymbol{h}_{t-1} \end{bmatrix} + \boldsymbol{b} \right] \\ &= \sigma \odot \left[\begin{bmatrix} \boldsymbol{W}_{\boldsymbol{\theta},x} & \boldsymbol{W}_{\boldsymbol{\theta},h} \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} \boldsymbol{o}_t \\ \boldsymbol{h}_{t-1} \end{bmatrix} + \boldsymbol{b} \right] \\ &= \sigma \odot \left[\boldsymbol{W}_{\boldsymbol{\theta},x} \boldsymbol{o}_t + \boldsymbol{W}_{\boldsymbol{\theta},h} \boldsymbol{h}_{t-1} + \boldsymbol{b} \right] \end{aligned} \tag{3.8}$$

The form of (3.8) has special importance as it allows a finite RNN to be universal i.e the RNN can model any function that can be computed by a universal Turing machine [108–110].

From (3.8), it can be seen that the at each time step, the next state $\boldsymbol{h}_t$ is computed by applying a point wise application of a non linear function on a vector that linearly combines the information associated with the previous hidden state $\boldsymbol{h}_{t-1}$ and the current stimulus $\boldsymbol{o}_t$. Gated RNNs [12, 111, 112] extend generic RNNs to provide a more flexible framework where the next state is computed as a weighted combination of the previous state and a non linear function of $\boldsymbol{o}_t$ and $\boldsymbol{h}_{t-1}$ with the weights also being adaptive functions of $\boldsymbol{o}_t$ and $\boldsymbol{h}_{t-1}$:

$$\boldsymbol{h}_t = \boldsymbol{g}_{\boldsymbol{h}} \otimes \boldsymbol{h}_{t-1} + \boldsymbol{g}_{\boldsymbol{o}} \otimes \sigma \odot \left[\boldsymbol{W}_o \boldsymbol{o}_t + \boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{b} \right] \tag{3.9}$$

Here $\otimes$ denotes point wise multiplication between two vectors. In RNN terminology, the adaptive weights $\boldsymbol{g_h}$ and $\boldsymbol{g_o}$ are referred to as gating transfer functions are of the form (3.8). By independently reading, writing and erasing content from the stored hidden state, the gates allow RNNs to better process data with complex and separated interdependencies. More discussion on gated RNNs will follow when LSTMs will be introduced later in this chapter.

By employing either transition function it can be seen that RNNs can potentially utilise information from frames at individual time steps in a much richer way than models having feed-forward architectures. When fully unrolled across time, under the above transition rules, the model will yield posteriors $P_\theta(\boldsymbol{\phi}_t|\boldsymbol{o}_t^r, \bar{\mathcal{O}})$ where $\bar{\mathcal{O}}$ presents $\boldsymbol{o}_{1:t}$. For applications such as ASR, it is often beneficial to incorporate the entire $\boldsymbol{o}_{1:T}$ when generating frame posteriors. To address this issue, bi-directional RNNs were invented in [113]. These networks along maintaining the hidden state $\boldsymbol{h}_t$ employ a separate transition function that operates by reading inputs backwards in time.

3.5 Problem of Model Selection

Let X denote the parameter space $\boldsymbol{\theta}$. Since different realisations of model parameters lead to different $f(\boldsymbol{x}, \boldsymbol{\theta})$, the manifold X essentially captures the set of all $P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})$ distributions $\mathcal{M}$ that can be generated by a particular model. In the context of ASR, where the samples $(\boldsymbol{x}, \boldsymbol{y})$ represent $(\mathcal{O}, \mathcal{H})$, an ideal candidate $f(\mathcal{O}, \boldsymbol{\theta}) \in \mathcal{M}$ is one that avoids rote memorisation and instead generalises on the concepts necessary to perform optimum inference. To aid good selection, a candidate f is picked from $\mathcal{M}$ that minimises the expected loss Q using some form of risk measure over the adequately selected family of predictive models:

$$\hat{f}(\mathcal{O}, \boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} Q\Big(f(\mathcal{O}, \boldsymbol{\theta})\Big) \tag{3.10}$$

$$= \arg\min_{\boldsymbol{\theta}} \mathbb{E}\left[\mathrm{L}\Big(\mathrm{f}(\mathcal{O}, \boldsymbol{\theta})\Big)\right]$$

$$= \arg\min_{\boldsymbol{\theta}} \int \sum_{\mathcal{H}} P(\mathcal{O}, \mathcal{H}) L\left(\mathcal{H}, \mathcal{H}_{f(\mathcal{O}, \boldsymbol{\theta})}\right) d\mathcal{O} \tag{3.11}$$

where

$$\mathcal{H}_{f(\boldsymbol{\theta}\mathcal{O})} = \underset{\mathcal{H}}{\operatorname{argmax}}\, P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}) \tag{3.12}$$

Although it is desirable to minimise the expected loss, in reality complete information of the true joint distribution $p(\mathcal{O}, \mathcal{H})$ is unavailable. Hence, the expected loss is approximated by the empirical risk function $\hat{Q}$:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \hat{Q}\Big(f(\mathcal{O}, \boldsymbol{\theta})\Big) \tag{3.13}$$

$$= \arg\min_{\boldsymbol{\theta}} \frac{1}{R} \sum_{r}^{R} L\left(\mathcal{H}^r, \mathcal{H}_{f(\mathcal{O}^r, \boldsymbol{\theta})}\right) \tag{3.14}$$

using a set of $r \in R$ independently drawn input-output samples $\{\mathcal{O}^r, \mathcal{H}^r\}_{r=1}^{R}$. The mismatch between the expected and the empirical loss often leads to phenomenon known as overfitting to the training data where careful searching through the parameter space X leads to the selection of a candidate f that achieves the greatest reduction in the empirical loss but fails to generalise to new examples. To alleviate this issue, training is often accompanied with regularisation approaches that serve to penalise candidates that over fit to the training data. Section 4.4 discusses the standard regularisation methods currently used with derivative based optimisation methods for DNN based models.

From (3.11) and (3.14), it can be also seen how the eventual choice of the inference model $P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})$ depends on the loss function used in the respective risk criterion. In ASR, the WER is the evaluation metric of interest which however corresponds to a discontinuous function of the model parameters. Hence, employing such a metric directly within a empirical risk criterion is not viable with standard derivative based optimisers. The standard approach is to formulate either generative or discriminative learning framework as discussed in Sec.2.4 to identify a set of viable parameter candidates $f(\mathcal{O}, \boldsymbol{\theta}) \in \mathcal{M}$ using the training set. The generalised performance of each of these candidates is then estimated using the validation set and the best function is chosen.

3.6 Error Gradient of DNN based models

For DNN based models, the objective function $F_{\text{obj}}(\boldsymbol{\theta})$ over model parameters can be expressed as the composition $L \circ \hat{\boldsymbol{z}} \circ \hat{\boldsymbol{a}}$ where

- $\hat{\boldsymbol{a}}$ represents the accumulated m dimensional vector representations $\boldsymbol{a}_t$ of the network's pre-softmax output at different time steps.

Cost function	Form of $\nabla \boldsymbol{L}^r_{\text{obj},t}$
F_{ML}	$\nabla \boldsymbol{L}^r_{\text{ML,t}} = \boldsymbol{z}^r_t - \boldsymbol{\gamma}^{r,\text{Num}}_t$ (see Appendix B.2) where $\boldsymbol{\gamma}^{r,Num}_t$ is the posterior probability of HMM states at time t computed over the numerator lattice associated with utterance r.
F_{CTC}	$\nabla \boldsymbol{L}^r_{\text{CTC,t}} = \boldsymbol{z}^r_t - \boldsymbol{\gamma}^{r,\text{CTC}}_t$ where $\boldsymbol{\gamma}^{r,CTC}_t$ is the normalised forward-backward probabilities computed over the CTC state model [53, 114].
F_{CE}	$\nabla \boldsymbol{L}^r_{\text{CE,t}} = \boldsymbol{z}^r_t - \boldsymbol{y}^r_t$ where $\boldsymbol{y}^r_t$ is the target label of frame t associated with utterance r.
F_{MMI}	$\nabla \boldsymbol{L}^r_{\text{MMI,t}} = \boldsymbol{\gamma}^{r,\text{Den}}_t - \boldsymbol{\gamma}^{r,\text{Num}}_t$ where $\boldsymbol{\gamma}^{r,Den}_t$ is the posterior probability of HMM states at time t computed over the denominator lattice associated with utterance r [115, 116].
F_{MBR}	$\nabla \boldsymbol{L}^r_{\text{MBR,t}} = \boldsymbol{\gamma}^r_t \odot \boldsymbol{L}$ $\boldsymbol{\gamma}^r_t$ here is the posterior probability associated with HMM states at time t and the entries of $\boldsymbol{L}$ corresponds to the loss associated with these arcs within the consolidated lattice [1, 115, 116].

Table 3.3 shows the form of the gradient of various cost functions w.r.t DNN linear output activations

- $L \circ \hat{\boldsymbol{z}}$ is the composition of the loss function L with the softmax $\boldsymbol{z}_t$ accumulated at various time instances.

Using the chain rule, the gradient $\nabla \boldsymbol{F}_{\text{obj}}(\boldsymbol{\theta})$ with respect to θ at time t is:

$$\nabla \boldsymbol{F}_{\text{obj}}(\boldsymbol{\theta})|_t = \frac{1}{R} \sum_r^R \left(\nabla \boldsymbol{L}^r_{\text{obj,t}} \right)^{\text{T}} \boldsymbol{J}^r_{\boldsymbol{\theta},t} \tag{3.15}$$

where $\nabla \boldsymbol{L}^r_{\text{obj},t}$ is the time dependent gradient of F_{obj} with respect to the linear DNN output activations and $\boldsymbol{J}^r_{\boldsymbol{\theta},t}$ is the Jacobian of the DNN's pre-softmax output w.r.t $\boldsymbol{\theta}$ at time t. For the types of objective functions discussed in Sec. 2.4, it can be seen that to compute $\nabla \boldsymbol{F}_{\text{obj}}(\boldsymbol{\theta})$, all that is necessary is to derive $\nabla \boldsymbol{L}^r_{\text{obj,t}}$ w.r.t to the DNN's pre-softmax activations at each time step t. Table 3.3 shows the form of $\nabla \boldsymbol{L}^r_{\text{obj},t}$ associated with the various sequence level loss functions. With the exception of the CE objective function, it can be seen that the computation of the $\nabla \boldsymbol{L}^r_{\text{obj},t}$ term for the individual loss functions relies on the statistic $\boldsymbol{\gamma}^r_t$ being computed. In Sec. 2.3, it was discussed how such a statistic can be computed efficiently using the forward backward algorithm.

Accumulating the frame-level gradients across time, the gradient w.r.t. a sequence level loss corresponds to:

$$\nabla \boldsymbol{F}_{\text{obj}}(\boldsymbol{\theta}) = \frac{1}{R}\sum_{r}^{R}\sum_{t}\left(\nabla \boldsymbol{L}^{r}_{\text{obj,t}}\right)^{\text{T}} \boldsymbol{J}^{r}_{\boldsymbol{\theta},t} \tag{3.16}$$

3.6.1 Computing the Jacobian w.r.t Pre-Softmax Activations

The back-propagation algorithm [102] is the method that is most widely used to compute the Jacobian $\boldsymbol{J}^{r}_{\boldsymbol{\theta},t}$. The algorithm uses a local message passing scheme in which information is sent alternately forwards and backwards through the network. At the heart of back-propagation is the chain rule which is used recursively to compute the Jacobian $\boldsymbol{J}^{r}_{\boldsymbol{\theta}^{l},t}$ at every layer l. Let $\boldsymbol{a}^{l}$ denote the output of l in a network of k layers deep. Then for feedforward networks,

$$\boldsymbol{J}_{\boldsymbol{\theta}^{l},t} = \left[\prod_{l\leq i\leq k} \nabla_{\boldsymbol{a}^{i-1}_{t}} \boldsymbol{a}^{i}_{t}\right] \nabla_{\boldsymbol{\theta}^{l}} \boldsymbol{a}^{l}_{t} \tag{3.17}$$

To keep the notation uncluttered, the dependency on r here has been dropped. In Sec. 3.4.2, it was shown how standard RNNs can be unrolled across time in the form of a feedforward network where parameters are tied across different layers. In this case, $\boldsymbol{J}_{\boldsymbol{\theta},t}$ takes the form:

$$\boldsymbol{J}_{\boldsymbol{\theta},t} = \sum_{l}^{k}\left[\prod_{l\leq i\leq k} \nabla_{\boldsymbol{a}^{i-1}_{t}} \boldsymbol{a}^{i}_{t}\right] \nabla_{\boldsymbol{\theta}} \boldsymbol{a}^{l}_{t} \tag{3.18}$$

This modification of the back-propagation algorithm is popularly known as back-propagation through time (BPTT) [102, 103]. For TDNNs, a similar expression to (3.18) can be derived for each individual layer with the summation now performed over the number of context shifts.

3.7 Problem of Vanishing and Exploding Gradients

Using chain's rule, the derivative $\nabla_{\boldsymbol{a}^{i-1}_{t}} \boldsymbol{a}^{i}_{t}$ in eqn. (3.17) and eqn. (3.18) can be shown to correspond to:

$$\nabla_{\boldsymbol{a}^{i-1}_{t}} \boldsymbol{a}^{i}_{t} = \text{diag}(\nabla_{\boldsymbol{a}^{i}_{t}}\sigma \odot \boldsymbol{a}^{i}_{t})\boldsymbol{W}^{i}_{\boldsymbol{\theta}} \tag{3.19}$$

where $\boldsymbol{W}_{\boldsymbol{\theta}}^{i}$ is the weight matrix associated with the layer i. Taking norms on both sides:

$$\|\nabla_{\boldsymbol{a}_t^{i-1}} \boldsymbol{a}_t^i\| = \|\text{diag}(\nabla_{\boldsymbol{a}_t^i} \sigma \odot \boldsymbol{a}_t^i) \boldsymbol{W}_{\boldsymbol{\theta}}^{i}\| \tag{3.20}$$

$$\leq \|\text{diag}(\nabla_{\boldsymbol{a}_t^i} \sigma \odot \boldsymbol{a}_t^i)\| \|\boldsymbol{W}_{\boldsymbol{\theta}}^{i}\| \tag{3.21}$$

By *singular value decompositon theorem* [117],

$$\|\nabla_{\boldsymbol{a}_t^{i-1}} \boldsymbol{a}_t^i\| = \|\text{diag}(\nabla_{\boldsymbol{a}_t^i} \sigma \odot \boldsymbol{a}_t^i)\| \|\boldsymbol{U}^i \Sigma^i (\boldsymbol{V}^i)^{\mathrm{T}}\|$$

$$\leq \zeta_{max}^{i} \|\text{diag}(\nabla_{\boldsymbol{a}_t^i} \sigma \odot \boldsymbol{a}_t^i)\| \|\boldsymbol{U}^i (\boldsymbol{V}^i)^{\mathrm{T}}\| \tag{3.22}$$

Equ.(3.22) provides an upper bound on the norm of $\nabla_{\boldsymbol{a}_t^{i-1}} \boldsymbol{a}_t^i$ where ζ_{max}^{i} corresponds to the largest singular value of the matrix $\boldsymbol{W}_{\boldsymbol{\theta}}^{i}$. In the scenario when $\|\sigma'(x)\| \leq \epsilon \in \mathbb{R}$ and $\epsilon \cdot \zeta_{max}^{i} \leq 1 \ \forall i$, information from gradients of deeper layers rapidly attenuate as they are backpropagated down the network. This phenomenon is termed as *vanishing gradients* [118] and impedes deep networks from learning useful intermediate representations. The problem of *vanishing gradients* is specially prevalent when the choice of σ corresponds to a member from the parameterised sigmoid family [119]:

$$\sigma(x) = \mu . \frac{1}{1 + \exp(-\vartheta x)} - \hat{\mu} \tag{3.23}$$

where μ, $\hat{\mu}$ and ϑ correspond to either fixed or tuneable parameters. From (3.23), it can be seen that $\|\sigma'(x)\|$ is bounded and for certain choices of μ, $\hat{\mu}$ and ϑ [6], its norm is less than 1. For such cases, employing these functions makes the network more susceptible to suffer from *vanishing gradients*.

The sigmoid function is a smooth approximation of the Heaviside step function that exhibits asymptotic behaviour when $\boldsymbol{\theta}$ takes large values. In the scenario when $\epsilon \cdot \zeta_{max}^{i} \geq 1 \ \forall i$, the norm of gradients from deeper layers grow as they are backpropagated down the network. This phenomenon is popularly known as the problem of *exploding gradients* [118] and impedes learning by saturating the individual nodes.

3.7.1 Rectified Linear Units

To stabilise training, there is ongoing research of finding suitable choices of σ that can lead to considerable speeds up in learning [120]. In literature, the current most popular

[6] for the standard sigmoid function, both μ and ϑ correspond to 1 and $\hat{\mu}$ is 0, whereas for the tanh function μ and ϑ equal to 2 and $\hat{\mu}$ is 1.

choice is the rectified linear unit (ReLU) unit [121]:

$$\sigma(\boldsymbol{x}) = \max\Big(0, \boldsymbol{\theta}^T(\boldsymbol{x}) + b_{\boldsymbol{\theta}}\Big) \tag{3.24}$$

The function corresponds to an identity map when $\boldsymbol{\theta}^T(\boldsymbol{x}) + b_{\boldsymbol{\theta}} \geq 0$ and 0 otherwise. The use of such a function can be seen to have a regularising effect as depending on the input signal, nodes at intermediate layers can switch off and not partake in producing the underlying mapping. This induces sparsity in the model and reduces the model's ability to overfit the training data. The derivative of the ReLU function corresponds to either 0 or 1 which means that gradient information from deep layers can propagate well to active nodes in the lower layers. However for deactivated nodes as a consequence of the gradient being 0, nodes which go into that state will stop responding to variations in error or input. This is called dying ReLU problem [122] and can result in the network being passive. To address this issue, recent works have modified the activation function to allow a small, non-zero gradient when the unit is not active [123, 124].

This work will focus on training standard RNNs with both the sigmoid and the standard ReLU activation function. In comparison to their sigmoid counterparts, RNNs using ReLUs have been observed to less prone to vanishing gradients [125].

3.7.2 Model Re-Design

To improve learning, the easiest strategy is not always to improve the optimisation algorithm. Instead many improvements in the training deep models have come from re-designing the models such that they can be better optimised with SGD. To train recurrent models over longer sequences, the standard approach is to enforce some form of gating mechanism to control the information inflow to the individual units in a layer. In Sec. 3.4.2.1, it was shown how this corresponds to using a state transition function of the form (3.9). One of the very first gated RNNs introduced is the Long Short Term Memory(LSTM) network [12]. A LSTM is an artificial neural network that contains specialised gated 'cells' instead of regular network units. Each cell can be described as a "smart" network that maintains its own local memory. At each time step, the memory cell is modified using 3.4.2.1 as shown below:

$$\boldsymbol{m}_t = \boldsymbol{g}_{\boldsymbol{m}} \otimes \boldsymbol{m}_{t-1} + \boldsymbol{g}_{\boldsymbol{x}} \otimes \sigma \odot \Big[\boldsymbol{W}_{\boldsymbol{\theta},x}\boldsymbol{x}_t + \boldsymbol{W}_{\boldsymbol{\theta},h}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{\theta}}\Big] \tag{3.25}$$

$$\boldsymbol{h}_t = \boldsymbol{g}_{\boldsymbol{h}} \otimes \boldsymbol{m}_t \tag{3.26}$$

The information from the memory cell gets diluted using an output gate that controls the output flow of cell activations into the rest of the network (3.26). Through the use of the gates, the model can maintain knowledge of important events across multiple time steps. Although gated RNNs are able to better model long term trends from sequential data, the use of such structures contribute to significant computational cost during training. As the gates themselves are functions of the form (3.8), training a gated RNN is significantly slower than a standard RNN of the same depth as these structures posess 4 times as many parameters as the RNN. To address the computational drawback, Gated Recurrent Units (GRU) [111] and recently Semi-Tied Units (STUs) [126] have been proposed to solve the inefficiency issue by using some form of parameter tying of the gated units.

To improve training of deep feed forward networks, adding additional connections between non adjacent layers have to shown to improve learning with SGD. For standard RNNs using the state transition of the form 3.8, the same idea was pursued with High Order RNN architecture (HORNN) [127]. The activation of hidden states of τ steps back are directly propagated to compute the current state $\boldsymbol{h}_t$. The HORNN architecture can be viewed as a precursor to the recently proposed attention mechanism [128] where the hidden states associated with every times steps are used to compute the activation of the next layer.

3.8 Summary

This chapter presented a literature review of Deep Neural Networks within the context of acoustic modelling in ASR. Section 3.1 presented the definition of an Artificial Neural Network while Sec. 3.2 reviewed the principle concept behind the Multi Layer Perceptron (MLP) model. Section 3.3 discussed the particular importance of using a good initial starting point when training deep models. Section 3.4 presented the standard DNN architectures currently used for acoustic modelling, and discussed how the choice of DNN architecture controls the representational capability of a model. The problem of model selection was presented in Sec. 3.5 while the back-propagation algorithm along with the gradient of various sequence level losses was presented in Sec. 3.6. The chapter concluded with a discussion of the vanishing and exploding gradients problem that plagues gradient based training of deep models alongside a review of the various approaches used to alleviate these issues.

Chapter 4

Derivative based optimisation methods

While the structure of DNN models allows rich modelling capacity, it also creates complex dependencies between the parameters which makes learning appropriate parameter values for such models quite a complex and challenging task. This chapter presents a literature review of derivative based optimisation approaches used to train DNN models. The chapter begins with a description of first and second optimisation frameworks and presents the two most popular approaches used to model the Hessian of DNN based cost functions. This is followed by the development of a framework to formally describe a stochastic optimisation algorithm (Sec. 4.2). The rest of this chapter is organised as follows: section 4.3 presents a careful analysis on the behaviour of stochastic method at each iteration. Using this analysis, Sec. 4.3.2 shows how noise reduction methods like momentum and Nesterov momentum can help SGD in finding better paths in the parameter space. The chapter concludes with a review of the standard regularisation approaches used with DNN training to improve the generalisation performance of these models.

4.1 Framework behind Derivative based Optimisation Approaches

Let $\boldsymbol{\theta}_0$ be a element in X. Initialised with a starting point, the goal of an optimisation algorithm is to construct a path $\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3 \cdots \boldsymbol{\theta}_k$ in X that will converge to a local

optimum of using the following update rule:

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}_k \tag{4.1}$$

The core premise behind standard derivative based optimisation methods is Taylor's theorem. Assuming that the objective function $F_{\text{obj}}(\boldsymbol{\theta})$ is sufficiently smooth, Taylor's second order approximation models the local behaviour of the function by the following quadratic:

$$F_{\text{obj}}(\boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}) \simeq F_{\text{obj}}(\boldsymbol{\theta}_k) + \Delta\boldsymbol{\theta}^T \nabla F_{\text{obj}}(\boldsymbol{\theta}_k) + \frac{1}{2}\Delta\boldsymbol{\theta}^T H \Delta\boldsymbol{\theta} \tag{4.2}$$

where $\Delta\boldsymbol{\theta}$ represents any offset within a convex neighbourhood of $\boldsymbol{\theta}_k$ and H is the Hessian of F_{obj} computed w.r.t $\boldsymbol{\theta}_k$. Instead of directly optimising the objective function, derivative based approaches generate a candidate $\Delta\boldsymbol{\theta}_k$ at each iteration by minimising the above quadratic approximation. Depending on the number of components considered on the right hand side of (4.2), standard derivative methods can be categorised as either first or second order. First order approaches consider only the first two terms to generate a candidate $\Delta\boldsymbol{\theta}_k$ at each training iteration. In other words, these methods use the direction of steepest descent to traverse along the parameter space. Second order approaches on the hand, attempts to minimise (4.2) directly using some form of approximation of the Hessian:

$$\hat{\Delta\boldsymbol{\theta}} = \underset{\Delta\boldsymbol{\theta}}{\text{argmin}}\, F_{\text{obj}}(\boldsymbol{\theta}_k) + \Delta\boldsymbol{\theta}^T \nabla F_{\text{obj}}(\boldsymbol{\theta}_k) + \frac{1}{2}\Delta\boldsymbol{\theta}^T B \Delta\boldsymbol{\theta} \tag{4.3}$$

where the matrix B is an approximation of the Hessian H. Differentiating (4.3) and setting it to zero yields the critical point $\Delta\boldsymbol{\theta} = -B^{-1}\nabla F_{\text{obj}}(\boldsymbol{\theta})$. This is the Newton direction [129] and corresponds to a unique minimiser within a convex neighbourhood of $\boldsymbol{\theta}_k$ when B is positive definite. There are different approaches to constructing the matrix B. The following highlights the two most common approaches used for DNN based models.

4.1.1 The Broyden-Fletcher-Goldfarb-Shanno(BFGS) method

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) [130] belongs to the class of Quasi-Newton methods that approximates the Hessian matrix using updates specified by previous gradient evaluations. Like 2nd order methods, Quasi-Newton methods model the local behaviour of the objective function with (4.3) but with the added condition

that the fitted quadratic should not only match the gradient of the objective function F_{obj} at $\boldsymbol{\theta}_k$ but also the gradient at $\boldsymbol{\theta}_{k-1}$:

$$\nabla F_{\text{obj}}\Big(\boldsymbol{\theta}_k - \Delta\boldsymbol{\theta}_{k-1}\Big) = \nabla F_{\text{obj}}(\boldsymbol{\theta}_k) - B_k\Delta\boldsymbol{\theta}_{k-1} = \nabla F_{\text{obj}}(\boldsymbol{\theta}_{k-1}) \tag{4.4}$$

Rearranging the above terms gives us the secant equation [130]:

$$B_k^{-1}\left(\nabla F_{\text{obj}}(\boldsymbol{\theta}_k) - \nabla F_{\text{obj}}(\boldsymbol{\theta}_{k-1})\right) = \Delta\boldsymbol{\theta}_k$$

The matrix B_k^{-1} has $D(D+1)/2$ degrees of freedom whereas the above system comprises of D independent equations of D variables. To determine B_k^{-1} uniquely, Quasi Newton approaches impose the additional conditions:

$$\begin{aligned} B_k^{-1} &= \underset{\hat{B}}{\text{argmin}}\, \|\hat{B} - B_{k-1}^{-1}\|_p \\ \text{subject to } \hat{B} &= \hat{B}^T,\ \hat{B}\Delta y_k = \Delta\boldsymbol{\theta}_k \end{aligned} \tag{4.5}$$

where y_k denotes $\nabla F_{\text{obj}}(\boldsymbol{\theta}_{k-1}) - \nabla F_{\text{obj}}(\boldsymbol{\theta}_k)$. Difference choices of the matrix norm $\|A\|_p$ give rise to different Quasi-Newton methods. In the BFGS approach,, the matrix norm chosen is the *weighted Frobenius norm*:

$$\|A\|_W = \|W^{1/2}AW^{1/2}\|$$

where the matrix W is chosen to be the inverse of the average Hessian :

$$W^{-1} = \int_0^1 \nabla^2 F_{\text{obj}}(\boldsymbol{\theta}_{k-1} - t\Delta\boldsymbol{\theta}_{k-1})dt$$

Using the weighted Frobenius norm, (4.5) has the unique solution:

$$B_k^{-1} = \left(I - \rho_k\Delta\boldsymbol{\theta}_k y_k^T\right) B_{k-1}^{-1}\left(I - \rho_k y_k\Delta\boldsymbol{\theta}_k^T\right) + \rho_k\Delta\boldsymbol{\theta}_k\Delta\boldsymbol{\theta}_k^T \tag{4.6}$$

where $\rho_k = \frac{1}{y_k^T\Delta\boldsymbol{\theta}_k}$. Therefore in the BFGS approach, instead of computing a completely new B_k at each iteration, the method iteratively updates an existing B_k by making a rank 2 modification of the matrix using the most recent iterate and gradient displacements. By choosing the initial guess B_0^{-1} to be positive definite and ensuring $\rho_k > 0$, it can be shown by induction that the updated matrices $\{B_k^{-1}\}$ will always be positive definite [131]. A key feature of this approach is that the matrices $\{B_k^{-1}\}$ need not be formed

explicitly; instead, each product can be computed using a formula that only requires recent elements of the sequence of displacement pairs that have been saved in storage. However, as seen from eqn. (4.6), the method demands that a history of $\{(y_k, \Delta\boldsymbol{\theta}_k)\}$ is maintained for each iteration. This results in substantial memory overhead especially for large models. A common solution for this is to employ a limited memory scheme where a limited number of the most recent iterate and gradient displacements are used. This approach is known popularly known as Limited Memory BFGS [132].

4.1.2 Gauss Newton approach

In Sec. 3.6, it was shown how for the types of learning frameworks discussed in Sec. 2.4 , the gradient $\nabla \boldsymbol{F}_{\text{obj}}(\boldsymbol{\theta})$ takes the form:

$$\nabla \boldsymbol{F}_{\text{obj}}(\boldsymbol{\theta}) = \frac{1}{R}\sum_{r}^{R}\sum_{t}\left(\nabla \boldsymbol{L}^{r}_{\text{obj,t}}\right)^{\text{T}} \boldsymbol{J}^{r}_{\boldsymbol{\theta},t}$$

where $\nabla \boldsymbol{L}^{r}_{\text{obj},t}$ corresponds to the time dependent gradient of F_{obj} with respect to the linear DNN output activations and $\boldsymbol{J}^{r}_{\boldsymbol{\theta},t}$ is the Jacobian of the DNN's pre-softmax output w.r.t $\boldsymbol{\theta}$ at time t. Then the Hessian of $\boldsymbol{F}_{\text{obj}}(\boldsymbol{\theta})$ at any point $\boldsymbol{\theta}$ equates to:

$$\begin{aligned}\nabla^2 \boldsymbol{F}_{\text{obj}}(\boldsymbol{\theta}) &= \nabla\left(\frac{1}{R}\sum_{r}^{R}\sum_{t}\left(\nabla \boldsymbol{L}^{r}_{\text{obj,t}}\right)^{\text{T}} \boldsymbol{J}^{r}_{\boldsymbol{\theta},t}\right) \\ &= \frac{1}{R}\sum_{r}^{R}\sum_{t}\left(\boldsymbol{J}^{r}_{\boldsymbol{\theta},t}\right)^{\text{T}}\left(\nabla^2 \boldsymbol{L}^{r}_{\text{obj,t}}\right)^{\text{T}} \boldsymbol{J}^{r}_{\boldsymbol{\theta},t} + \frac{1}{R}\sum_{r}^{R}\sum_{t}\left(\nabla \boldsymbol{L}^{r}_{\text{obj,t}}\right)^{\text{T}} \nabla \boldsymbol{J}^{r}_{\boldsymbol{\theta},t}\end{aligned} \tag{4.7}$$

From the above expression, it can be seen that when working with DNN models, the Hessian at any particular point $\boldsymbol{\theta}$ on the error surface can be written as the sum of two matrices:

$$H_{\boldsymbol{\theta}} = G_{\boldsymbol{\theta}} + K_{\boldsymbol{\theta}} \tag{4.8}$$

where $G_{\boldsymbol{\theta}} = J_{\boldsymbol{\theta}}^{T}\nabla^2 L_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}$ and $K = \nabla L_{\boldsymbol{\theta}} \nabla J_{\boldsymbol{\theta}}$. The matrix $G_{\boldsymbol{\theta}}$ is called the Gauss Newton (GN) [133] matrix and if the loss function is convex, it is guaranteed to positive semi-definite. Under the Gauss Newton approximation, the quadratic minimised at each iteration then corresponds to:

$$\hat{\Delta\boldsymbol{\theta}} = \underset{\Delta\boldsymbol{\theta}}{\text{argmin}}\, F_{\text{obj}}(\boldsymbol{\theta}_k) + \Delta\boldsymbol{\theta}^{T}\nabla F_{\text{obj}}(\boldsymbol{\theta}_k) + \frac{1}{2}\Delta\boldsymbol{\theta}^{T} G_{\boldsymbol{\theta}} \Delta\boldsymbol{\theta} \tag{4.9}$$

If $K_{\boldsymbol{\theta}}$ is sparse, then such a quadratic becomes a good approximation of the local behaviour of the error surface.

4.2 Stochastic and Batch Optimisation Methods

Irrespective of whether a method is first or second order, all optimisation methods in machine learning can be essentially categorised into two groups: stochastic and batch. Batch based optimisation methods correspond to the class of approaches where an update to the model parameters is made only after processing the entire training dataset. A prototypical batch optimisation method is the full gradient descent where at each iteration k, the model parameters are updated using the following rule:

$$\boldsymbol{\theta_{k+1}} \leftarrow \boldsymbol{\theta}_k - \alpha_k \nabla \boldsymbol{F}_{\text{obj}}(\boldsymbol{\theta}) \tag{4.10}$$

with α_k being the learning rate. An alternative to the batch based approach is stochastic optimisation. At iteration k, a random seed $\boldsymbol{\xi}_k$[1] is sampled to generate a stochastic direction $g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k)$ which is used to update the model parameters as follows:

$$\boldsymbol{\theta_{k+1}} \leftarrow \boldsymbol{\theta}_k - \alpha_k \; g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k) \tag{4.11}$$

For any general stochastic optimisation approach, the following 3 mechanisms need to exist:

1. a mechanism for generating a realisation of a random variable $\boldsymbol{\xi}_k$. This normally involves picking a sample or subset of samples from the training data.

2. given an iterate $\boldsymbol{\theta}_k$ and the realisation of $\boldsymbol{\xi}_k$, a mechanism for computing a stochastic vector $g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k)$.

3. a mechanism for setting the learning rates α_k.

This framework can be inherently seen to be Markovian: $\boldsymbol{\theta}_{k+1}$ is a random variable that depends only on the iterate $\boldsymbol{\theta}_k$, the seed $\boldsymbol{\xi}_k$, and the step size α_k, and not on any past iterates. In this work, the direction $g(\boldsymbol{\theta_k}, \boldsymbol{\xi_k})$ will correspond to either:

[1] $\boldsymbol{\xi}$ denote a sampled utterance

$$g(\boldsymbol{\theta_k}, \boldsymbol{\xi_k}) = \begin{cases} \nabla F_{\text{obj}}(\boldsymbol{\theta_k}, \boldsymbol{\xi_k}) \\ A^{-1} \frac{1}{N_b} \sum_k^{N_b} \nabla F_{\text{obj}}(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k) \end{cases} \quad \text{where } \boldsymbol{\xi}_k \text{ corresponds to a sampled utterance} \tag{4.12}$$

This gives the flexibility for describing stochastic methods using different batch sizes, and also provides a framework to describe both first order and second methods under a stochastic approach.

4.3 Effectiveness of Stochastic methods

Having introduced the two broad categories of optimisation methods, this section the expected behaviour of stochastic based methods at each training iteration. Let the objective function $F_{\text{obj}} : \mathbb{R}^d \to \mathbb{R}$ be such that it is continuously differentiable and its gradient $\nabla F_{\text{obj}} : \mathbb{R}^d \to \mathbb{R}^d$ is Lipchitz continuous. Formally, such an assumption implies $\forall \Delta\boldsymbol{\theta}$:

$$F_{\text{obj}}(\boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}) \leq F_{\text{obj}}(\boldsymbol{\theta}_k) + \nabla F_{\text{obj}}(\boldsymbol{\theta}_k)\Delta\boldsymbol{\theta} + \frac{1}{2} L \Delta\boldsymbol{\theta}^T \Delta\boldsymbol{\theta} \tag{4.13}$$

Proof: By *Fundamental theorem of Calculas*

$$\begin{aligned} F_{\text{obj}}(\boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}) &= F_{\text{obj}}(\boldsymbol{\theta}_k) + \int_0^1 \frac{d}{dt} F_{\text{obj}}(\boldsymbol{\theta}_k + t\Delta\boldsymbol{\theta})\ dt \\ &= F_{\text{obj}}(\boldsymbol{\theta}_k) + \int_0^1 \nabla F_{\text{obj}}(\boldsymbol{\theta}_k + t\Delta\boldsymbol{\theta})^T \Delta\boldsymbol{\theta}\ dt \\ &= F_{\text{obj}}(\boldsymbol{\theta}_k) + \nabla F_{\text{obj}}(\boldsymbol{\theta}_k)\Delta\boldsymbol{\theta} + \int_0^1 \left[\nabla F_{\text{obj}}(\boldsymbol{\theta}_k + t\Delta\boldsymbol{\theta}) - \nabla F_{\text{obj}}(\boldsymbol{\theta}_k)\right]^T \Delta\theta\ dt \\ &\leq F_{\text{obj}}(\boldsymbol{\theta}_k) + \nabla F_{\text{obj}}(\boldsymbol{\theta}_k)\Delta\boldsymbol{\theta} + \int_0^1 L \left[t\Delta\boldsymbol{\theta}\right]^T \Delta\theta\ dt \end{aligned}$$

Requiring that the gradient at any point is Lipchitz continuous ensures that it does not change arbitrarily quickly with respect to the parameter vector. Such an assumption is essential for convergence analyses of most gradient-based methods [16] and without it, derivative based stochastic approach will not be a good algorithm for achieving decrement in F_{obj}.

Now, let $E_{\boldsymbol{\xi}_k}$ denote the expected value taken with respect to the distribution $\boldsymbol{\xi}_k$ given $\boldsymbol{\theta}_k$. From (4.11), it can be seen that $E_{\boldsymbol{\xi}_k}[F(\boldsymbol{\theta}_{k+1})]$ will then be a meaningful quantity as

$\boldsymbol{\theta}_{k+1}$ depends on $\boldsymbol{\xi}_k$. By substituting $(\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k)$ with $g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k)$ in eqn 4.13, the iterates of any general stochastic algorithm can hence be shown to satisfy:

$$E_{\boldsymbol{\xi}_k}\left[F_{\text{obj}}(\boldsymbol{\theta}_{k+1})\right] - F_{\text{obj}}(\boldsymbol{\theta}_k) \leq -\alpha_k \nabla F_{\text{obj}}(\boldsymbol{\theta}_k^T) E_{\boldsymbol{\xi}_k}\left[g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k)\right] + \frac{1}{2}\alpha_k^2 L E_{\boldsymbol{\xi}_k}\left[|g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k)|^2\right] \tag{4.14}$$

The above inequality shows that regardless of how any stochastic method arrived at $\boldsymbol{\theta}_k$, the expected decrease yielded by the kth step is bounded above by a quantity that involves:

1. the expected directional derivative of F at $\boldsymbol{\theta}_k$ along $g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k)$
2. the second moment of $g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k)$

where

$$V_{\boldsymbol{\xi}_k}\left[g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k)\right] = E_{\boldsymbol{\xi}_k}\left[|g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k)|^2\right] - |E_{\boldsymbol{\xi}_k}\left[g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k)\right]|^2 \tag{4.15}$$

To clarify for the reader, the term $V_{\boldsymbol{\xi}_k}$ here is the variance of $g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k)$. The first term of right hand of the inequality of eqn. (4.14) will be negative if in expectation, the vector $-g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k)$ is a direction of sufficient descent for F_{obj} from $\boldsymbol{\theta}_k$. However, the second term in eqn. (4.14) could be large enough to allow the objective value to increase. Balancing these terms is critical in the design of a good stochastic optimisation algorithm.

4.3.1 Learning Rate Schedulers

From eqn. (4.14), it can be seen how the choice of learning rate influences the expected decrease yielded by the k th step of the stochastic optimisation process. In [134], it is shown that SGD will converge to a local minimum as long the choice of learning rates satisfy:

$$\sum_k^{\infty} \alpha_k = \infty \text{ and } \sum_k^{\infty} \alpha_k^2 < \infty \tag{4.16}$$

This can be achieved through the use of good learning rate scheduler that makes to appropriate changes the learning rate over time. A good strategy as mentioned in [16], is to have larger step sizes at the beginning that enable a rapid increase in the objective function, and then switch to smaller learning rates at the later stages to allow SGD to descend into finer features of the loss landscape.

For training DNN models, a brief review of the standard learning rate schedulers is given below:

1. List: this is most naive learning rate scheduler that uses a pre-determined learning rate through out the epoch.

2. Exponential Scheduling: the learning rate decays exponentially after each epoch as shown below

$$\alpha_k = \alpha_0 \times p^{-k}$$

where p controls the rate the decay.

3. NewBob [45]: the learning rate is kept fixed as long as the classification error on the cross-validation (CV) set improves by at least defined percentage. In all subsequent epochs, the learning rate is halved. Training is terminated when the improvement falls below a certain threshold.

4. Adaptive Learning Rate Scheduling: corresponds to a class of approaches that adapt the general learning rate α at each time step t for each parameter θ based on the past gradients that have been computed for θ. For highly curved error surfaces, such approaches has been argued [135] to help SGD reach a good local minimum. Apart from Adam [136] which will be introduced later, the following procedures are most commonly used:

   (a) AdaGrad [137, 138]: this approach adapts the learning rate of parameters such that smaller updates (i.e. low learning rates) are made to parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) are applied to parameters associated with infrequent features. For this reason, it is well-suited for dealing with sparse data [139]. Formally, under this scheduler, the parameters are updated as follows:

$$\theta_{k+1} \leftarrow \theta_k - \frac{\alpha}{\sqrt{D_{\theta,t} + \epsilon}}\, g(\theta_k, \boldsymbol{\xi}_k) \tag{4.17}$$

   where $g(\theta_k, \boldsymbol{\xi}_k)$ is the noisy gradient w.r.t the parameter θ and $D_{\theta,t}$ is the accumulated sum of squared gradients up to time t. However, since every added term to $D_{\theta,t}$ is positive, the accumulated sum keeps growing during training. This in turn causes the learning rate to shrink and eventually become

infinitesimally small, at which point the algorithm is no longer able to acquire additional knowledge. This is the main weakness of AdaGrad.

(b) RMSprop [140] is an extension of AdaGrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, RMS prop restricts the window of accumulated past gradients to some fixed size w. Instead of inefficiently storing w previous squared gradients, the sum of gradients is recursively defined as a decaying average of all past squared gradients.

$$D_{\theta,t+1} \leftarrow \vartheta D_{\theta,t} + (1-\vartheta) g(\theta_k, \boldsymbol{\xi}_k)^2 \tag{4.18}$$

(c) Adadelta [141] can be seen as a further extension on RMSprop. In comparison to RMSprop, the method differs in the aspect that the numerator term of the update in eqn. (4.17) is replaced by a decaying average of squared updates.

$$\Delta\theta_k \leftarrow \frac{\sqrt{P_{\theta,t-1} + \epsilon}}{\sqrt{D_{\theta,t} + \epsilon}}\, g(\theta_k, \boldsymbol{\xi}_k) \tag{4.19}$$

$$\theta_{k+1} \leftarrow \theta_k + \Delta\theta_k \tag{4.20}$$

where $P_{\theta,t-1}$ is the decaying average of all past squared updates similar to eqn (4.18). Although the method doesn't require an initial learning rate, the use of moving averages makes the method heavily biased to the values of the decay parameters ϑ.

4.3.2 Noise Reduction Methods

In the last section, it was shown how the progress of a stochastic optimisation algorithm is hindered by the variance associated with $g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k)$. To limit the harmful effect of the second term in eqn. (4.14), stochastic methods are often equipped with dynamic sampling or gradient aggregation based approaches or a even combination of both. Dynamic sampling methods achieve noise reduction by gradually increasing the mini batch size used at each iteration [142]. Gradient aggregation methods on the other hand achieve noise reduction by storing gradient estimates employed in previous iterations. Information of the previous directions are employed to generate a $g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k)$ whose variance is bounded [16]. To train DNN models, the use of the following two gradient aggregation methods are commonly used:

1. Gradient methods with momentum
2. Accelerated gradient based methods

4.3.2.1 Gradient Methods with Momentum

Gradient methods with momentum are procedures in which the update chosen at each iteration is a combination of the steepest descent direction and the most recent iterate displacement. Specifically, in comparison to the generalised stochastic optimisation algorithm presented in Sec. 4.2, momentum based methods only differ in the aspect that they maintain a separate scalar sequence $\{\vartheta_k\}$ that they use to scale the most recent displacement. Like learning rates, these parameters are either predetermined or set dynamically. Given an initial point $\boldsymbol{\theta}_0$, the update at each iteration corresponds to:

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \alpha_k g(\boldsymbol{\theta_k}, \boldsymbol{\xi_k}) + \vartheta_k(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1}) \tag{4.21}$$

Now if set $\boldsymbol{v}_{k+1} \leftarrow -\alpha_k g(\boldsymbol{\theta_k}, \boldsymbol{\xi_k}) + \vartheta_k(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1})$ then the above update can be broken down into the following steps :

$$\boldsymbol{v}_{k+1} \leftarrow -\alpha_k g(\boldsymbol{\theta_k}, \boldsymbol{\xi_k}) + \vartheta_k \boldsymbol{v}_k \tag{4.22}$$

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \boldsymbol{v}_{k+1} \tag{4.23}$$

The above representation is the most popular formulation of gradient based momentum updates [102]. An alternative view of this approach can be done by expanding (4.23):

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \alpha_k \sum_{j=1}^{k} \vartheta^{k-j} g(\boldsymbol{\theta_j}, \boldsymbol{\xi_j}) \tag{4.24}$$

Thus, each step can be viewed as an exponentially decaying average of past search directions. Assuming that $g(\boldsymbol{\theta_k}, \boldsymbol{\xi_k})$ is chosen such that

$$E_{\xi_k}[g(\boldsymbol{\theta_k}, \boldsymbol{\xi_k})]^T F(\boldsymbol{\theta}_k) \geq \varepsilon \|F(\boldsymbol{\theta}_k)\|^2$$

where $\varepsilon \geq 0$ then by writing the iteration this way, one can see that the steps tend to accumulate contributions in directions of persistent descent, while directions that oscillate tend to get cancelled, or at least remain small.

4.3.2.2 Accelerated Gradient Methods

The Accelerated Gradient (Nesterov) method was first proposed by Nesterov [143] and is similar to the gradient methods with momentum. As a two step procedure, it involves the updates:

$$\hat{\boldsymbol{\theta}}_k \leftarrow \boldsymbol{\theta}_k + \vartheta_k(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1}) \tag{4.25}$$

$$\boldsymbol{\theta}_{k+1} \leftarrow \hat{\boldsymbol{\theta}}_k - \alpha_k g(\hat{\boldsymbol{\theta}}_{\boldsymbol{k}}, \boldsymbol{\xi_k}) \tag{4.26}$$

which leads to the condensed form:

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \vartheta_k(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1}) - \alpha_k g(\boldsymbol{\theta}_k + \vartheta_k(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1}), \boldsymbol{\xi_k}) \tag{4.27}$$

In particular, the difference between Nesterov and standard momentum based approach can be described as a reversal in the order of computation. In eqn. (4.23), one can imagine taking the steepest descent step and then applying the momentum term, whereas eqn. (4.27) results when one follows the momentum term first, then applies a steepest descent step. If both ϑ_k and α_k are fixed across all iterations, then the accelerated gradient can be also presented as:

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \alpha \sum_{j=1}^{k} \vartheta^{k-j} g(\boldsymbol{\theta}_j + \vartheta_k(\boldsymbol{\theta}_j - \boldsymbol{\theta}_{j-1}), \boldsymbol{\xi_j}) \tag{4.28}$$

4.3.3 Adaptive Moment Estimation (Adam)

Having discussed adaptive learning rate schedulers and noise reduction methods, it is worth discussing the method of Adaptive Moment Estimation (Adam) [136]. Adam effectively combines RMSprop with noise reduction methods. In addition to storing an exponentially decaying average of past squared gradients like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients similar to momentum. Whereas momentum can be seen as a ball running down a slope, Adam behaves like a heavy ball with friction, which thus prefers flat minima in the error surface.

In recent work [144, 145], the method has been shown to be outperformed by SGD with momentum when applied to an object recognition and machine translation task. In [146], the authors attributes this to the use of the exponential moving average of past squared gradients. In settings where Adam converges to a sub-optimal solution, it was observed that some mini-batches provide large and informative gradients, but as these

mini-batches only occur rarely, exponential averaging diminishes their influence, which leads to its poor convergence.

4.4 Regularisation

The space of functions $\mathcal{M}$ that result from different realisations of model parameters $\boldsymbol{\theta}$ should be considered with two potentially competing goals in mind. First, $\mathcal{M}$ should contain functions that are able to achieve a low empirical risk over the training set, so as to avoid bias or underfitting the data. Second, the gap between the expected and the empirical risk should be small for all $f \in \mathcal{M}$. The first goal can be achieved by selecting a rich family of functions or by using a prior knowledge to select a well-targeted family. Deep Neural Networks consist of large number of parameters that provides enough flexibility to describe a wide range of phenomena. However due to their large model complexity, it is possible for such models to describe almost any data set of a fixed size without capturing any genuine insights of the underlying phenomenon. This as mentioned in Sec. (3.5) often leads to the phenomenon of overfitting and results in a situation where satisfying the first goal is at odds with accomplishing the second.

4.4.1 Early stopping

To tackle overfitting, the most common approach is to use an experimental procedure that involves splitting the training examples into three disjoint subsets: a training set, a validation set, and a testing set. Having a separate held out validation set allows us to approximate the generalisation performance of the network after each training epoch. By monitoring how the network fares on the held out set, training is stopped as soon as performance on this subset saturates. This strategy is called early stopping.

4.4.2 LP Norm Regularisation

Along with employing early stopping using a held out validation set, the training of DNN models often employ L^p-norm regularisation techniques. The idea behind L^p-norm [147] regularisation methods is to add an extra term (often called as the regularisation term) of the form shown below to the training criterion:

$$\hat{F}_{\text{obj}}(\boldsymbol{\theta}) = F_{\text{obj}}(\boldsymbol{\theta}) + \frac{\mu}{2}\|\boldsymbol{\theta}\|^p \tag{4.29}$$

Intuitively, adding such a term to the cost function can be seen to have the effect on inducing the network to learn parameters with small magnitude while minimising the original cost function. Large parameter values are only allowed if they considerably improve the first part of the cost function. The value μ in (4.29) controls the compromise made on the relative importance of the two elements.

To understand how L^p norm regularisation methods impact learning, suppose the network parameters take small values as will tend to happen in a regularised network. When the parameter values are small, that the behaviour of the network won't change too much to changes in input values. This makes it more difficult for a regularised network to learn the effects of local noise in the training data. An intuitive way to understand this is that single pieces of evidence no longer matters as much to the output of the network. Instead, a regularised network learns to respond to types of evidence which are seen often across the training set. By contrast, a network with large parameters may change its behaviour quite a lot in response to small changes in the input. In the context of neural network training, the two approaches from the family of L^p-norm regularisation methods that are mostly used are L2 and L1 norms.

4.4.2.1 L2 Norm Regularisation

In L2 norm regularisation [148], the regularisation term added to the objective function takes the form $\frac{\mu}{2}\|\boldsymbol{\theta}\|^2$:

$$\hat{F}_{\text{obj}}(\boldsymbol{\theta}) = F_{\text{obj}}(\boldsymbol{\theta}) + \frac{\mu}{2}\|\boldsymbol{\theta}\|^2$$

When applied with in a stochastic optimisation framework, the update produced at each iteration then corresponds to:

$$\begin{aligned}\boldsymbol{\theta}_{k+1} &= \boldsymbol{\theta}_k - \alpha_k\Big(g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k) + \mu\boldsymbol{\theta}_k\Big) \\ &= (1 - \alpha_k\mu)\,\boldsymbol{\theta}_k - \alpha_k g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k)\end{aligned} \tag{4.30}$$

From (4.30), it can be seen that applying L2 norm regularisation has the effect of re-scaling the model parameters by a factor $(1 - \alpha_k\mu)$ before applying the stochastic update rule. This re-scaling is sometimes referred to as weight decay.

4.4.2.2 L1 Norm Regularisation

In L1 norm regularisation [149], on the other hand the regularisation term takes the form $\mu \sum_i |\theta_i|$ which results in the following modified objective function:

$$\hat{F}_{\text{obj}}(\boldsymbol{\theta}) = F_{\text{obj}}(\boldsymbol{\theta}) + \mu \sum_i |\theta_i|$$

With in a stochastic optimisation framework, the update produced at each iteration then corresponds to:

$$\begin{aligned} \boldsymbol{\theta}_{k+1} &= \boldsymbol{\theta}_k - \alpha_k \left(g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k) + \mu \sum_i |(\theta_i)| \right) \\ &= \boldsymbol{\theta}_k - \alpha_k \mu \text{sign} \odot (\boldsymbol{\theta}_k) - \alpha g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k) \end{aligned} \tag{4.31}$$

In both expressions (4.30) and (4.31), the effect of adding the regularisation term can be seen to shrink the parameters during optimisation. However, the relative effectiveness of both approaches is dependent on the norm of $\boldsymbol{\theta}$. In L1 regularisation, the parameters are shrunk by a constant amount toward 0 while in L2 regularisation, the parameters are rescaled by a factor less than 1. When $\|\boldsymbol{\theta}\|$ is large, it can be hence seen that L2 is more effective than L1 in shrinking the parameters quickly. However, when $\|\boldsymbol{\theta}\|$ is small, L1 regularisation can be seen to be more effective in driving the parameters to 0. The net result is that L1 regularisation tends to concentrate the parameters of the network in a relatively small number of high-importance connections, while driving the rest toward zero. This tends to make the network weights sparse.

4.4.3 Weight Clipping

One way to ensure the parameters of the network are less responsive to local noise is to restrict them between the interval $[\theta_{min}, \theta_{max}]$. This is known as 'clipping'. An minimum lower bound on the value that a parameter can take ensures that our model doesn't under-fit the data while an upper bound ensures that the network's output doesn't change too much for small changes in input.

There are two ways to achieve this: In the first approach, individual parameters are clipped after each update using the following rule:

$$\theta = \begin{cases} \theta_{min} & \text{if } \theta \leq \theta_{min} \\ \theta & \text{if } \theta_{min} < \theta < \theta_{max} \\ \theta_{max} & \text{if } \theta \geq \theta_{max} \end{cases} \tag{4.32}$$

Alternatively, clipping can be directly enforced on the search direction $g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k)$ rather than on the updated $\boldsymbol{\theta}$:

$$g(\theta_k, \boldsymbol{\xi}_k) = \begin{cases} \theta_{min} & \text{if } \theta \leq \theta_{min} \\ g(\theta_k, \boldsymbol{\xi}_k) & \text{if } \theta_{min} < g(\theta_k, \boldsymbol{\xi}_k) < \theta_{max} \\ \theta_{max} & \text{if } \theta \geq \theta_{max} \end{cases} \tag{4.33}$$

By setting appropriate values for θ_{max} and θ_{min}, only extreme values of the individual updates $g(\theta_k, \boldsymbol{\xi}_k)$ are clipped to prevent saturation.

4.4.4 Dropout

Apart from the above procedures, a recent popular approach that is popularly used to regularise network training is Dropout [150]. Dropout is a technique where randomly selected hidden units are ignored during training i.e their contribution to the activation of downstream hidden units are temporally removed on the forward pass and their parameters are not updated during the backward pass. The purpose of dropout is to make the network become less sensitive to the specific parameters of individual hidden units. During the course of training, parameters of hidden units are tuned for specific features providing some specialisation. Neighbouring hidden units become to rely on this specialisation, which if taken too far can result in a fragile model that is too specialised to the training data. This reliance on context for a neuron during training is referred as complex co-adaptations. It can be imagined that when hidden units are randomly dropped out of the network during training, that other hidden units will have to step in and handle the representation required to make predictions for the missing hidden units. This is believed to result in multiple independent internal representations being learned by the network.

From a modelling perspective, the method of dropout essentially can be viewed as form of ensemble learning. In ensemble learning a number of 'weaker' classifiers are trained separately and then at test time some form of averaging of the responses of all ensemble members is used. Since each classifier has been trained separately, it has learned different 'aspects' of the data and their mistakes are different. Combining them helps to produce an stronger classifier, which is less prone to overfitting. Random Forests [151] are a typical example of ensemble classifiers. One ensemble variant is bagging, in which each member of the ensemble is trained with a different sub sample of the input data, and thus has learned only a subset of the whole possible input feature space. Dropout, then, can be seen as an extreme version of bagging [152]. Applying dropout to a neural network amounts to sampling a 'thinned' network to process each mini-batch and only updating the parameters of the sampled network based on the data corresponding to the mini-batch.

At test time, it is not feasible to explicitly average the predictions from exponentially many thinned models. However, a very simple approximate averaging method works well in practice. The idea is to use a single neural net at test time without dropout. The parameters of this network are scaled-down versions of the trained parameters. If a unit is retained with probability p during training, the outgoing parameters of that unit are multiplied by p at test time. This ensures that for any hidden unit the expected output (under the distribution used to drop units at training time) is the same as the actual output at test time. The undesirable property of this scheme presented above is that the activations must be scaled by p at test time. Since test-time performance is so critical, it is always preferable to use inverted dropout, which performs the scaling at train time, leaving the forward pass at test time untouched. Inverted dropout is used in the HTK implementation of dropout.

4.5 Summary

This chapter presented a review of derivative based optimisation methods used to train parametric models like DNNs. At the beginning of the chapter, a description of first and second optimisation frameworks was presented followed by a review of the two most popular approaches used to model the Hessian of DNN based cost functions. Section 4.2 presented a formal framework to describe any stochastic optimisation algorithm. This was followed by careful analysis to understand the behaviour of any generalised stochastic algorithm. Using this mathematical framework, in Sec. 4.3.2, it was shown how noise

reduction methods like momentum and Nesterov momentum can potentially aid SGD to construct better paths along the parameter space. The chapter concluded with a review of standard regularisation approaches currently used with DNN training to improve the generalisation performance of DNN models.

Chapter 5

Framework for Large Scale Optimisation

This chapter presents the implementation details of the Hessian Free (HF) optimisation method investigated in this thesis. The chapter begins with an analysis of batch and stochastic methods, and presents a discussion on the potential advantages of having a batch styled optimisation framework that achieves a good balance between data parallelisation and requiring far less updates than SGD to converge. The chapter is organised is as follows: Section 5.2 presents a review the HF method with particular emphasis to the Conjugate Gradient (CG) algorithm, which serves as the core component of the HF approach. The section also presents a detailed description on how curvature vector products using the Gauss Newton (GN) matrix can be computed in a DNN. Section 5.4 provides the details about the implementation of CG algorithm, and presents two novel contributions of this thesis:

- A procedure to stabilise CG training to allow effective updates to be yielded only from few iterations.
- An approach to adapt the CG algorithm to better handle DNN architectures with shared parameters.

The chapter concludes with preliminary experiments using the HF method on the 50hr MGB1 training set.

5.1 Motivation for Batch Styled Second Order Optimisation Frameworks

When processing extremely large datasets, both batch and stochastic based optimisation methods have significant impact on the computational workload associated with solving an underlying machine learning problem. Suppose that both the expected risk Q and the empirical risk $\hat{Q}$ attain their minima with parameter vectors $\boldsymbol{\theta}^* \in \operatorname{argmin} Q(\boldsymbol{\theta})$ and $\boldsymbol{\theta}_n \in \operatorname{argmin} \hat{Q}(\boldsymbol{\theta})$ respectively. In addition, let $\hat{\boldsymbol{\theta}}_n$ be the approximate empirical risk minimiser returned by a given optimisation algorithm when the time budget $\mathcal{T}_{max}$ is exhausted. Then, the error associated with $\hat{\boldsymbol{\theta}}_n$ can be represented as:

$$Q(\hat{\boldsymbol{\theta}}_n) = Q(\boldsymbol{\theta}^*) + [Q(\boldsymbol{\theta}_n) - Q(\boldsymbol{\theta}^*)] + [Q(\hat{\boldsymbol{\theta}}_n) - Q(\boldsymbol{\theta}_n)] \tag{5.1}$$

Accounting for different possible initialisations, the expected error associated with $\hat{\boldsymbol{\theta}}_n$ corresponds to:

$$\mathbb{E}[Q(\hat{\boldsymbol{\theta}}_n)] = Q(\boldsymbol{\theta}^*) + [Q(\boldsymbol{\theta}_n) - Q(\boldsymbol{\theta}^*)] + \mathbb{E}[Q(\hat{\boldsymbol{\theta}}_n) - Q(\boldsymbol{\theta}_n)] \tag{5.2}$$

Taking expectation to all possible data sets

$$\mathbb{E}[Q(\hat{\boldsymbol{\theta}}_n)] = Q(\boldsymbol{\theta}^*) + \mathbb{E}[Q(\boldsymbol{\theta}_n) - Q(\boldsymbol{\theta}^*)] + \mathbb{E}[Q(\hat{\boldsymbol{\theta}}_n) - Q(\boldsymbol{\theta}_n)] \tag{5.3}$$

From (5.3), it can be seen that minimisation of the expected error requires finding a careful balance between the contributions made by each of the three terms on the right-hand side. The term $\mathbb{E}[Q(\hat{\boldsymbol{\theta}}_n) - Q(\boldsymbol{\theta}_n)]$ is the expected offset between the minimiser found by the optimisation algorithm and the minimum of the empirical risk $\hat{Q}(\boldsymbol{\theta})$. The term $Q(\boldsymbol{\theta}^*)$ is dependent on the choice of the model while $\mathbb{E}[Q(\boldsymbol{\theta}_n) - Q(\boldsymbol{\theta}^*)]$ will be minimised as the size of the training set is increased. Thus, minimising $\mathbb{E}[Q(\hat{\boldsymbol{\theta}}_n)]$ w.r.t. a fixed amount of training data corresponds to finding a candidate in $\mathcal{M}$ that minimises $\mathbb{E}[Q(\hat{\boldsymbol{\theta}}_n) - Q(\boldsymbol{\theta}_n)]$.

In [16], Bottou shows how batch methods in comparison to stochastic approaches do better in converging to a good $\hat{\boldsymbol{\theta}}_n$ for both convex and non-convex problems. However, the ability of such methods to converge to a good local minima depends highly on the capacity to perform a large number of updates. When constrained to a single machine, this becomes impractical in the scenario where either the training dataset is too large or when the computational budget is fixed. In contrast, the per iteration cost of a stochastic

approach is not tied to the training set size. The ability to perform more updates within an epoch allows such methods to achieve greater reductions in $\mathbb{E}[Q(\hat{\boldsymbol{\theta}}_n) - Q(\boldsymbol{\theta}_n)]$. In the particular case of sequence training, stochastic optimisation frameworks are restricted to operate within a utterance randomisation scheme rather than the standard frame randomisation approach: the gradients associated with individual time steps rely on the forward and backward statistics computed on the whole utterance, which forces the gradient computation to be serial at the level of utterances. When faced with thousands of hours of training data, methods like utterance level stochastic gradient descent may not at times be able to complete an entire epoch as the cost of an epoch scales linearly with the number of utterances in the training set.

These issues can be alleviated by having a parallel optimisation framework where the computation is shared between different worker nodes. With stochastic methods like standard SGD, parallelisation can be achieved either in a synchronous or asynchronous way. In synchronous SGD (SSGD) [17], local workers compute the gradients over their own mini-batches[1] and then propagate the gradients to the global model. In this framework, the workers wait for each other and will only proceed their own local training once gradients from all the workers have been added to the global model. When the gradient is computed over the entire training set, such an approach can be seen to be equivalent to a full batch approach where gradient computations are inherently data parallel. A major drawback is SSGD is that the global model needs to always wait for the slowest worker to reply before applying an update. In the scenario where the optimisation approach requires a lot of updates to converge, this waiting process makes a significant contribution to the computational overhead incurred during training.

To improve training efficiency, asynchronous SGD (ASGD) is proposed in [18] where each local worker continues its training process right after its gradient is added to the global model. Although ASGD can achieve faster speed due to no waiting overhead, it suffers from a problem called delayed gradient [19], where the global model receives 'stale' gradients from worker nodes i.e gradients computed with respect to previous parameter settings. This makes the approach not mathematically sound and may cause the training trajectory to suffer from unexpected turbulence. The drawbacks of ASGD can be alleviated by having a synchronous optimisation framework that finds convergence within very few updates.

In [15], it has been shown that both faster and better convergence can be achieved for convex problems by employing second order approaches to adjust the gradient direction

[1] for sequence training, this corresponds to a subset of utterances.

when there is high non-linearity and ill conditioning of the objective function. For discriminative sequence training, it has been shown in [20] that better convergence than serial SGD can be achieved by having a synchronous second order Hessian Free (HF) optimisation framework where the master (i.e the global model) employs second order curvature information to rescale the accumulated gradients before applying an update. For the particular case of lattice based sequence training, apart from transferring DNN models parameters, lattices associated with individual utterances needs to transferred across multiple workers. In [20, 153], it is shown that compared to parallel forms of SGD training, the HF approach, by employing large batch sizes, achieves a better balance between data parallelisation and communication overhead. Following the work in [20], the next section presents our implementation of a batch style HF optimisation framework using the HTK DNN Toolkit [47].

5.2 Hessian Free Optimisation Framework

Assuming that the objective criterion $F_{\text{obj}}(\boldsymbol{\theta})$ is sufficiently smooth, within a convex neighbourhood of a given point $\boldsymbol{\theta}_k$ in the parameter space X, the behaviour of F_{obj} can be locally approximated as:

$$F_{\text{obj}}(\boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}) \simeq F_{\text{obj}}(\boldsymbol{\theta}_k) + \Delta\boldsymbol{\theta}^T \nabla F_{\text{obj}}(\boldsymbol{\theta}_k) + \frac{1}{2}\Delta\boldsymbol{\theta}^T H \Delta\boldsymbol{\theta} \tag{5.4}$$

In Sec. 4.1, it was shown how second order approaches instead of minimising $F_{\text{obj}}(\boldsymbol{\theta})$ directly, at each iteration, choose to minimise a quadratic of the form:

$$\hat{\Delta\boldsymbol{\theta}} = \underset{\Delta\boldsymbol{\theta}}{\text{argmin}}\, F_{\text{obj}}(\boldsymbol{\theta}_k) + \Delta\boldsymbol{\theta}^T \nabla F_{\text{obj}}(\boldsymbol{\theta}_k) + \frac{1}{2}\Delta\boldsymbol{\theta}^T B \Delta\boldsymbol{\theta} \tag{5.5}$$

where the matrix B is an approximation of the Hessian H. Differentiating eqn (5.5) and setting it to zero yields the Newton direction $\Delta\boldsymbol{\theta} = -B^{-1}\nabla F_{\text{obj}}(\boldsymbol{\theta})$. However, the Newton direction does not scale well with the dimensionality of the parameter space X. Computing this direction directly is expensive in terms of both computation and storage as storing B requires $\mathcal{O}(D^2)$ storage and inverting it incurs a cost of $\mathcal{O}(D^3)$. These obstacles however, can be overcome if we employ inexact Newton methods. In particular, rather than computing the Newton direction exactly through matrix factorisation techniques, the system is solved $B\Delta\boldsymbol{\theta} = -\nabla F(\boldsymbol{\theta})$ using the iterative linear Conjugate Gradient(CG) algorithm [154]. An overview of the algorithm is presented in Algorithm 1.

Algorithm 1 Overview of Linear Conjugate gradient method

1: Initialise initial search direction $\boldsymbol{b}$, #CGIter)
2: Set $\boldsymbol{r}_0 \leftarrow \boldsymbol{b}$, $\boldsymbol{d}_0 \leftarrow \boldsymbol{r}_0, k \leftarrow 0$
3: **while** k < #CGIter **do**
4: Compute $\boldsymbol{r}_k^T \boldsymbol{r}_k$
5: Set $a_k \leftarrow \frac{\boldsymbol{r}_k^T \boldsymbol{r}_k}{\boldsymbol{d}_k^T B \boldsymbol{d}_k}$
6: Update $\Delta\boldsymbol{\theta}_{k+1} \leftarrow \Delta\boldsymbol{\theta}_k + a_k \boldsymbol{d}_k$
7: Update $\boldsymbol{r}_{k+1} \leftarrow \boldsymbol{r}_k - a_k B \boldsymbol{d}_k$
8: Compute $\boldsymbol{r}_{k+1}^T \boldsymbol{r}_{k+1}$
9: Set $\beta_{k+1} \leftarrow \frac{\boldsymbol{r}_{k+1}^T \boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^T \boldsymbol{r}_k}$
10: Update $\boldsymbol{d}_{k+1} \leftarrow \boldsymbol{r}_{k+1} + \beta_{k+1} \boldsymbol{d}_k$
11: Return $\Delta\boldsymbol{\theta}_k$

5.2.1 The CG algorithm

The linear conjugate gradient (CG) is an iterative method that have been originally developed to solve a linear system of the form

$$A\Delta\boldsymbol{\theta} = \boldsymbol{b} \tag{5.6}$$

where A corresponds to a symmetric positive definite matrix. Alternatively, it can be also interpreted as a method for finding the minimum of a quadratic error surface

$$F(\Delta\boldsymbol{\theta}) = \frac{1}{2}\Delta\boldsymbol{\theta}^T A \Delta\boldsymbol{\theta} - \boldsymbol{b}\Delta\boldsymbol{\theta} + c \tag{5.7}$$

The above quadratic $F(\Delta\boldsymbol{\theta})$ is minimised at the critical point $\Delta\boldsymbol{\theta} = A^{-1}\boldsymbol{b}$. For high-dimensional spaces, computing $A^{-1}\boldsymbol{b}$ is highly expensive. One way to avoid this expensive computation is to solve the system $A\Delta\boldsymbol{\theta} = \boldsymbol{b}$. This is where CG comes into the picture. It is an efficient iterative method for solving such systems. It should be mentioned that the quadratic problem of (5.7) can be also solved with alternative iterative techniques like the method of steepest descent, where exact step sizes can be computed for each descent direction. To understand why the steepest descent method is sub optimal, recall that the method is an iterative algorithm that attempts to find the minimum of the quadratic (5.7) using the following update rule:

$$\Delta\boldsymbol{\theta}_{k+1} = \Delta\boldsymbol{\theta}_k + \alpha_i \boldsymbol{r}_k$$

where $\boldsymbol{r}_k = \boldsymbol{b} - A\Delta\boldsymbol{\theta}_k$ is called the residual and corresponds to the direction of steepest decent i.e $-\nabla F(\Delta\boldsymbol{\theta})$ of eqn.(5.7). For quadratic problems such as eqn.(5.7), the step sizes α_k can be computed exactly by taking the derivative of $F(\Delta\boldsymbol{\theta}_{k+1})$ with respect to α and setting to 0. The value of α_k that minimises of $F(\Delta\boldsymbol{\theta}_{k+1})$ occurs when of $\nabla F(\Delta\boldsymbol{\theta}_{k+1})^T\boldsymbol{r}_k = 0$. i.e

$$\boldsymbol{r}_{k+1}^T\boldsymbol{r}_k = 0 \tag{5.8}$$

$$(\boldsymbol{b} - A\Delta\boldsymbol{\theta}_{k+1})^T\boldsymbol{r}_k = 0 \tag{5.9}$$

$$\Big(\boldsymbol{b} - A(\Delta\boldsymbol{\theta}_k + \alpha_k\boldsymbol{r}_k)\Big)^T\boldsymbol{r}_k = 0 \tag{5.10}$$

$$\alpha_k = \frac{\boldsymbol{r}_k^T\boldsymbol{r}_k}{\boldsymbol{r}_k^T A\boldsymbol{r}_k} \tag{5.11}$$

Intuitively, this means that the optimisation algorithm will move in the negative direction of the current gradient until it reaches a point where the new derivative is orthogonal to direction currently being traversed. This causes the method to move in a zig-zag path to reach the minimum of the quadratic surface. Such a feature is quite inefficient as it forces the method to often visit previous directions.

Let the error at step k be $\boldsymbol{e}_k = \Delta\boldsymbol{\theta}_k - \Delta\boldsymbol{\theta}_{\min}$, a vector that indicates the offset from the minimum $\Delta\boldsymbol{\theta}_{\min}$. Also let $\boldsymbol{d}_k$ denote the search direction used at step k. Instead of revisiting the same direction multiple times, it will be ideal to take an appropriate step along a chosen direction once. This provides the motivation to consider directions that ensures $\boldsymbol{e}_{k+1}$ is orthogonal to previously visited directions. Mathematically, this complies to:

$$\boldsymbol{e}_{k+1}^T\boldsymbol{d}_k = 0 \tag{5.12}$$

To achieve this, α_k must satisfy:

$$(\boldsymbol{e}_k + \alpha\boldsymbol{d}_k)^T\boldsymbol{d}_k = 0$$

$$\alpha_k = \frac{\boldsymbol{d}_k^T\boldsymbol{e}_k}{\boldsymbol{d}_k^T\boldsymbol{d}_k} \tag{5.13}$$

Unfortunately, to compute the correct step size, knowledge of the error offset is required which is unavailable. The next best alternative is to have conjugate search directions A

orthogonal to each other

$$\boldsymbol{d}_k^T A \boldsymbol{d}_j = 0$$

$\forall k \neq j$. The value of α that minimises $F(\Delta\boldsymbol{\theta}_{k+1})$ will then occur when

$$\boldsymbol{d}_k^T A \boldsymbol{e}_{k+1} = 0 \tag{5.14}$$

Using the fact that $\boldsymbol{r}_k = \boldsymbol{b} - A\Delta\boldsymbol{\theta}_k$, it can be shown that eqn (5.14) simplifies to:

$$\boldsymbol{d}_k^T \boldsymbol{r}_{k+1} = 0 \tag{5.15}$$

Solving the above equation results in

$$\begin{aligned} \alpha_k &= \frac{\boldsymbol{d}_k^T A \boldsymbol{e}_k}{\boldsymbol{d}_k^T A \boldsymbol{d}_k} \\ &= \frac{\boldsymbol{d}_k^T \boldsymbol{r}_k}{\boldsymbol{d}_k^T A \boldsymbol{d}_k} \end{aligned} \tag{5.16}$$

Let $\mathcal{D}_i$ be the the i dimensional subspace spanned by the first i conjugate directions $\{\boldsymbol{d}_0, \boldsymbol{d}_1 \cdots \boldsymbol{d}_i\}$. It can be shown that the steps taken by CG along these directions is optimal [154] in the sense that the subspace will no longer be revisited at subsequent iterations of the algorithm. This is basis of the CG algorithm. From the residual directions, the method constructs conjugate directions on which it takes appropriate step sizes only once. This makes the algorithm, in comparison to steepest descent, more efficient in finding the minimum of the quadratic function (5.7).

5.2.1.1 Key features of the CG algorithm

- Like many iterative linear system techniques, CG applied to eqn (5.5) does not require access to the Hessian itself, only Hessian-vector products. In this sense, such a method is called **Hessian-Free** (HF) [16, 155] when B is chosen to approximate the Hessian.

- If $\Delta\boldsymbol{\theta}_0$ is set to 0, the linear CG algorithm immediately improves upon the direction used to initialise the algorithm. If this corresponds to the full gradient or stochastic gradient step, then the very first iteration of CG finds an appropriate step size (learning rate) for this direction while subsequent iterations monotonically improve upon this step.

- If A has only r distinct eigenvalues, then the CG iteration will terminate at the solution in at most r iterations [131]. Furthermore, the subspace associated with the eigenvectors having larger eigenvalues will be traversed first.

5.3 Computing Matrix Vector Products with the Gauss Newton matrix

Martens [155] in his initial paper on the HF method used the Gauss Newton (GN) approximation to successfully apply the HF optimisation in training auto-encoders on three small standard datasets using the squared error criteria. Wiesler and Li [156] have applied this approximation for HF training w.r.t the CE loss criteria on the MNIST [157] task and on a small message dictation task. For sequence training, Sainath and Kingsbury [153] have reported to achieve significant reductions in WER on large datasets, when they employ a GN approximation of the Hessian matrix. Following their approach, the HF method implemented implemented in this thesis will employ the GN matrix as an approximation to the Hessian.

In Sec. 4.1.2, it was shown how the GN matrix can be expressed as the product of three matrices:

$$G_{\boldsymbol{\theta}} = \frac{1}{R}\sum_{r}^{R}\sum_{t}\left(\boldsymbol{J}^{r}_{\boldsymbol{\theta},t}\right)^{\mathrm{T}}\nabla^{2}\boldsymbol{L}^{r}_{\mathrm{obj,t}}\boldsymbol{J}^{r}_{\boldsymbol{\theta},t}$$

where

- $\nabla^{2}\boldsymbol{L}^{r}_{\mathrm{obj,t}}$ is the Hessian of the loss associated with the rth utterance at time t w.r.t the DNN linear output activations.
- $\boldsymbol{J}^{r}_{\boldsymbol{\theta},t}$ is the Jacobian of the DNN output activations w.r.t $\boldsymbol{\theta}$ at time t when processing the rth utterance.

Thus, a multiplication of the GN matrix with a vector $\boldsymbol{v}$ amounts to a sequential multiplication of the vector with the three matrices [158] using the following steps.

1. The product of $\boldsymbol{J}^{r}_{\boldsymbol{\theta},t}(\boldsymbol{v})$ corresponds to the directional derivative. In [159], Pearlmutter showed how this can be performed efficiently through a single forward pass of the network by employing a method called forward differentiation [159, 160].
2. Multiplying the output of the forward differentiation with $\nabla^{2}\boldsymbol{L}^{r}_{\mathrm{obj,t}}$ results in $\left(\nabla^{2}\boldsymbol{L}^{r}_{\mathrm{obj,t}}\right)\boldsymbol{J}^{r}_{\boldsymbol{\theta},t}(\boldsymbol{v})$.

3. The back-propagation algorithm takes derivatives with respect to predictions as input i.e $\nabla \boldsymbol{L}^r_{\text{obj,t}}$ and computes the derivative with respect to the parameters i.e $\left(\boldsymbol{J}^r_{\boldsymbol{\theta},t}\right)^{\mathrm{T}} \nabla \boldsymbol{L}^r_{\text{obj,t}}$. Replacing $\nabla \boldsymbol{L}^r_{\text{obj,t}}$ with $\nabla^2 \boldsymbol{L}^r_{\text{obj,t}} \boldsymbol{J}^r_{\boldsymbol{\theta},t}(\boldsymbol{v})$, allows the required curvature vector product to be computed.

5.3.0.1 Computing the Directional Derivative

The computation of $\boldsymbol{J}^r_{\boldsymbol{\theta},t}(\boldsymbol{v})$ can be achieved by making appropriate modifications to forward propagation equations such that the output of the resultant forward pass corresponds to the necessary directional derivative. To see how this can be achieved, let us consider a standard DNN with 2 hidden layers. For notational clarity, let the layer ids be denoted by superscripts and the node ids corresponding to the t-th input be denoted by subscripts. Using this notation, the forward pass equations for a particular node at each layer can be represented as:

$$a^1_{t,j} = \sum_i^N \theta^1_{ji} x_{t,i} + \theta_{j0} \tag{5.17}$$

$$z^1_{t,j} = \sigma(a^1_{t,j}) \tag{5.18}$$

$$a^2_{t,k} = \sum_j \theta^2_{kj} z^1_{t,j} + \theta_{k0} \tag{5.19}$$

$$z^2_{t,k} = \sigma(a^2_{t,k}) \tag{5.20}$$

$$a^3_{t,l} = \sum_k \theta^3_{lk} z^2_{t,k} + \theta_{l0} \tag{5.21}$$

$$z^3_{t,l} = \frac{\exp(a^3_{t,l})}{\sum_{l'} \exp(a_{t,l'\}^3)} \tag{5.22}$$

Here $\boldsymbol{a}^i_t$ represents the linear activations associated with nodes in layer i after the application of the affine map and $\boldsymbol{z}^i_t$ corresponds to the activations of the layer after the point wise application of σ. Appendix A.4 introduces the concept of a directional derivative $d\Phi|_{\boldsymbol{\theta}}$ which corresponds to a linear map of the vector $\boldsymbol{v}$ from the tangent space of $\boldsymbol{\theta}$ to the tangent space of $\Phi(\boldsymbol{\theta})$. In DNN literature [86, 159], this is commonly referred to as the $\mathcal{R}$ operator $\mathcal{R} = \boldsymbol{v}\nabla$:

$$\begin{aligned}\mathcal{R}(f(\boldsymbol{\theta})) &= \frac{\partial f}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta} + t\boldsymbol{v}) \mid_{t=0} \\ &= (\nabla_{\boldsymbol{\theta}} f)^{\mathrm{T}} \boldsymbol{v}\end{aligned}$$

For the individual coordinates $\theta^k_{i,j}$, applying this operator results in:

$$\mathcal{R}(\theta^k_{i,j}) = \sum_{i',j',k'} \frac{\partial \theta^k_{i,j}}{\partial \theta^{k'}_{i',j'}} v^{k'}_{i',j'} = v^k_{i,j}$$

For the first layer, applying the operator to the forward equations yields:

$$\mathcal{R}(a^1_{t,j}) = \sum_i^N v^1_{ji} x_{t,i} \tag{5.23}$$

$$\mathcal{R}(z^1_{t,j}) = \sigma'(a^1_j)\mathcal{R}(a^1_{t,j}) \tag{5.24}$$

where (5.24) is a consequence of applying the chain rule. Since the operator $\boldsymbol{v}\nabla$ is a linear map, it can be also shown to satisfy the following conditions:

$$\mathcal{R}(cf(\boldsymbol{\theta})) = c\mathcal{R}(f(\boldsymbol{\theta})) \tag{5.25}$$

$$\mathcal{R}(f(\boldsymbol{\theta}) + g(\boldsymbol{\theta})) = \mathcal{R}(f(\boldsymbol{\theta})) + \mathcal{R}(g(\boldsymbol{\theta})) \tag{5.26}$$

Using the above expressions, the forward differentiation equations associated with individual nodes in all subsequent layers equates to:

$$\mathcal{R}(a^2_{t,k}) = \sum_j v^2_{kj} z^1_{t,j} + \sum_j w^2_{kj} \mathcal{R}(z^1_{t,j}) \tag{5.27}$$

$$\mathcal{R}(z^2_{t,k}) = \sigma'(a^2_{t,k})\mathcal{R}(a^2_{t,k}) \tag{5.28}$$

$$\mathcal{R}(a^3_{t,l}) = \sum_k v^3_{lk} z^2_{t,k} + \sum_k w^3_{lk} \mathcal{R}(z^2_{t,k}) \tag{5.29}$$

5.4 Implementation Details

In the last section, it was shown how the computation of the matrix vector products within each CG iteration is as expensive as a gradient evaluation. For large scale problems, this is quite a concern as for tasks like sequence training, the cost of a single CG iteration increases linearly with the number of utterances in the training set. This makes CG quite uncompetitive against alternative approaches such as SGD or a limited memory BFGS method. Interestingly, however, the structure of the risk measures given by eqn (3.14) and eqn (2.34) can be exploited so that the resulting method has lower computational overhead as described next.

5.4.1 Sub-sampled Hessian Free method

To reduce the cost associated with individual CG iterations, this thesis employs a sub-sampled Hessian Free method, where an approximation of B is used when minimising the quadratic function of eqn (5.5). The motivation stems from the observations made by [155] where the author discovered that with inexact Newton methods, the B matrix need not be as accurate as the gradient to yield an effective update. Translated to the context of large-scale machine learning applications, this means that the CG iteration is more tolerant to noise in the Hessian estimate than it is to noise in the gradient estimate. Based on this idea, the technique stated here employs a smaller sampled subset for estimating the candidate matrix B than for the gradient estimate. So for a given min-batch S_k, if the gradient estimate is

$$\nabla \boldsymbol{F}_{\text{obj}}(\boldsymbol{\theta}) = \frac{1}{|S_k|} \sum_{r}^{|S_k|} \sum_{t} \left(\nabla \boldsymbol{L}^{r}_{\text{obj,t}} \right)^{\text{T}} \boldsymbol{J}^{r}_{\boldsymbol{\theta},t} \tag{5.30}$$

then the approximation of the candidate B matrix is made on a subset $S_k^B \subset S_k$. If the subset S_k^B is chosen to be small, the cost of each CG iteration can be reduced significantly. On the other hand, the size of S_k^B should be at least large enough for the information carried by the matrix vector products to be productive. From preliminary experiments on the WSJ0 debugging dataset (Appendix C.1) and on the 50hr MGB1 training set, it was observed that using 0.5 hrs of randomly sampled audio data provides a good balance between increased computational cost and improved training.

5.4.2 Improving CG Training

The main reason why HF has seen fewer practical applications than SGD is due to the CG algorithm dominating the computation cost. Each matrix vector product is as expensive as a forward and backward pass. In sequence training, this equates of processing a single utterance. As the number of forward and backward passes scales linearly with the number of CG iterations and size of CG mini batch, running CG for many iterations can be seen to computationally expensive. Martens in [155], claims that reasonable updates from CG can be only achieved by running the algorithm for more than 150-200 iterations. The same claims were made by the authors in [153] when they applied HF to do discriminative sequence training. Careful investigation in this thesis has found that the need to have long CG runs is a consequence of how the matrix vector products are computed. Recall that at each iteration of the CG algorithm (see

Sec. 5.2.1), the step size to be taken along each update direction is computed by:

$$\alpha_k = \frac{\boldsymbol{r}_k^T \boldsymbol{r}_k}{\boldsymbol{d}_k^T B \boldsymbol{d}_k} \tag{5.31}$$

This is guaranteed to be positive when the choice of B is restricted to a positive definite matrix. For CE training, the associated the GN matrix is guaranteed to be positive semi-definite, it was observed that the denominator term in eqn (5.31) was frequently negative at the initial CG iteration even though the code implementation was correct. Martens describes this phenomenon as the problem of negative curvature and proposes to run the CG algorithm longer with the aim of having enough iterations where the directions and associated step sizes are valid. Such an approach, however, greatly increases the computational workload and makes the method less competitive against standard SGD.

This thesis has found the prevalence of negative curvature to be the result of the norm of the conjugate search direction being too small in relation to the norm of the model parameters. Recall that when B corresponds to the GN matrix, the matrix vector product $B(\boldsymbol{d}_k)$ is computed by sequentially multiplying the vector with $J_{\boldsymbol{\theta}}$ followed by $\nabla^2 L_{\boldsymbol{\theta}}$ and $J_{\boldsymbol{\theta}}^T$. The parameter vector $\boldsymbol{d}_k$ can be viewed as alternative choice for the weight matrices associated with the various layers. Section 5.3.0.1 showed how the computation of the product $J_{\boldsymbol{\theta}}(\boldsymbol{d}_k)$ can be achieved efficiently by running the $\mathcal{R}$ operator on the forward propagation equations: in the very first layer, applying this operator conforms to performing an alternative affine transformation of the input with the layer's weight matrix replaced by the associated matrix in $\boldsymbol{d}_k$:

$$\mathcal{R}(a_{t,j}^1) = \sum_i^N d_{ji}^1 x_{t,i} \tag{5.32}$$

This is followed by point wise multiplication of the result with cached forward activations:

$$\mathcal{R}(z_{t,j}^1) = \sigma'(a_j^1)\mathcal{R}(a_{t,j}^1) \tag{5.33}$$

For all subsequent layers l, applying the $\mathcal{R}$ operator corresponds to taking the sum of two affine transformations:

$$\mathcal{R}(a_{t,k}^l) = \sum_j d_{kj}^l z_{t,j}^{l-1} + \sum_j \theta_{kj}^l \mathcal{R}(z_{t,j}^{l-1}) \tag{5.34}$$

In the scenario when $\|\boldsymbol{d}_k\| << \|\boldsymbol{\theta}\|$, eqn (5.34) gets reduced to an affine transformation with respect to the actual network weights:

$$\mathcal{R}(a^l_{t,k}) = \sum_j d^l_{kj} z^{l-1}_{t,j} + \sum_j \theta^l_{kj} \mathcal{R}(z^{l-1}_{t,j}) \approx \sum_j \theta^l_{kj} \mathcal{R}(z^{l-1}_{t,j}) \tag{5.35}$$

The prevalence of numerical underflow prevents the directional derivatives from being properly computed, resulting in the inner product to be at times negative. By maintaining a sparse initialisation, Martens implicitly allows the norm of the weights to be comparable to search directions computed on the CG mini-batch [155]. To make CG robust, this thesis found the following approach to greatly stabilise CG training:

- Before computing $J(\boldsymbol{d}_k)$, temporarily scale $\boldsymbol{d}_k$ to have the same norm as $\boldsymbol{\theta}$ and then proceed to apply the $\mathcal{R}$ operator.
- After the directional derivatives have been computed, rescale $\boldsymbol{d}_k$ to its original norm.

Employing this temporal scaling was found to allow CG to yield effective updates only from few iterations.

5.4.3 Adapting CG for shared architectures

For tied architectures such as RNNs, initial experiments have found improvements from HF training to be considerably slow. A recurrent layer unrolled for T time steps is often modelled as a feed forward network of depth T with the weights of the layers being tied. Inspection of the CG algorithm in Sec. 5.2 shows how the choice of the step size and scaling of the conjugate directions depend on the dot product $r_k^T r_k$ and the vector Gp_k. For the first iteration of CG, the vector r_0 corresponds to the accumulated average gradient. In Sec. 5.3.0.1 it is shown how the product Gp_k, like the gradient, can be efficiently computed by a forward and backward pass. For tied architectures like an RNN, shared parameters receive more updates and hence will contribute more to the norm of the vectors r_0 and Gp_k than parameters which are not shared. Careful investigation in the work reported here found that in situations where the shared parameters dominate the norm of the mentioned vectors, the CG algorithm becomes considerably slow in finding updates that improve upon the MPE criterion. This thesis has found that this issue can be alleviated by scaling the conjugate search directions with a diagonal matrix Λ, whose non-zero entries correspond to the square root of the reciprocal of the number

of times a parameter has been shared. Scaling r_0 and Gp_k with Λ ensures that the norms of these vectors are not dominated by the contribution of the tied weights and makes the information carried by untied weights to be more available when constructing an update direction. It is shown in Sec. 5.5.4 that using this modification allows HF to produce progressively better updates at each iteration of training.

5.4.4 Choice of CG initialisation

In this thesis, the initial default direction probed by the CG algorithm is an approximation of the gradient. At each subsequent iteration, the algorithm iteratively finds conjugate directions that improve upon this initial direction. At the end of each run, the resultant update is the search direction that yields the greatest improvement on the CG mini-batch w.r.t the training criterion. When constrained with a hard limit on the number of allowed iterations, the ability of CG to find a reliable update becomes highly dependent on the choice of the initial search direction. In Sec. 4.3.2, it was discussed how noise reduction methods like momentum can help reduce the variance associated with estimation of the gradient estimate. Motivated by this, apart from using large batches to get better gradient estimates, Sec. 6.6.2, investigates the effect of initialising CG with:

$$\boldsymbol{d}_0 \leftarrow -\sum_{j=1}^{k} \vartheta^{k-j} \nabla \hat{F}_{\text{obj}}(\boldsymbol{\theta}_j) \tag{5.36}$$

where $\nabla \hat{F}_{\text{obj}}(\boldsymbol{\theta}_j)$ is the stochastic approximation of the true gradient computed at the point $\boldsymbol{\theta}_j$, and ϑ is the momentum coefficient. This variant of HF can be shown to be equivalent to the DSAG-HF method proposed by Dognin and Goel in [161] and will be referred to as DSAG-HF in this work.

5.4.5 Damping

Due to noise associated with the estimate of B, minimising eqn (5.5) may not necessarily guarantee a decrement in the objective function F_{obj}. Martens in [155] proposed to add an extra L2 norm regularisation term to eqn (5.5) to give:

$$\Delta\hat{\boldsymbol{\theta}} = \underset{\Delta\boldsymbol{\theta}}{\text{argmin}}\, F_{\text{obj}}(\boldsymbol{\theta}_k) + \Delta\boldsymbol{\theta}^T \nabla F_{\text{obj}}(\boldsymbol{\theta}_k) + \frac{1}{2}\Delta\boldsymbol{\theta}^T B \Delta\boldsymbol{\theta} + \mu\frac{1}{2}\Delta\boldsymbol{\theta}^T\Delta\boldsymbol{\theta} \tag{5.37}$$

Differentiating (5.37) and equating it to zero yields critical point:

$$\Delta\boldsymbol{\theta} = -(B + \mu I)^{-1}\nabla F_{\text{obj}}(\boldsymbol{\theta}_k) \tag{5.38}$$

When such a linear system is solved with CG, the effect of adding the L2 regulariser corresponds to taking comparatively more conservative steps along the individual conjugate directions. This approach is the most general form of Tikonov damping [148, 162]. However, a costly side effect of this approach is that training slows down considerably and more CG iterations are required to make relatively good progress with each update. This significantly increases the computational overhead and makes HF un-competitive against alternative approaches for solving large scale problems. An alternative to Tikonov damping is to restrict the number of CG iterations and periodically check the quality of the current CG direction on the CG mini-batch. For sequence training, this thesis has found the latter approach to be particularly useful. From initial experiments on the WSJ0 debugging (Appendix C.1) and MGB1 50hr training set, it was found that under the latter approach, running CG for 5-8 iterations was sufficient to make good progress while contributing only a small fraction to the total computational cost.

5.5 Preliminary Experiments with the 50h MGB1 dataset

The chapter concludes with preliminary experiments with MPE training on the 50hr MGB1 training set. This section investigates the following:

1. How re-scaling the gradient directions with matrices that capture local curvature information can speed up the convergence of batch methods.

2. How the noise associated with the gradient estimate influences the ability of HF to yield good solutions under a fixed computational budget.

3. How modifications presented in Sec. 5.4.3 to the algorithms improves training for tied architectures.

5.5.1 Details of Experimental Setup

To investigate the importance of using curvature information and effect of initialising CG with increasingly noisy gradients, experiments were conducted using a standard DNN

composed of 5 hidden layers each with 1000 nodes and equipped with sigmoid activation function on the 50hr MGB1 training set (Appendix C.2). To investigate CG training with tied architectures, a sigmoid RNN consisting of two recurrent layers of width 500 stacked one over the other followed by a feedforward layer with 500 nodes was used. The input to all the models was produced by splicing together 40 dimensional log-Mel filter bank (FBK) features extended with their delta coefficients. Prior to sequence training, the model parameters were initialised using frame-level CE training and during training, the unrolling of the recurrent layers was performed from -15 to +5 time steps[2]. For both models, the output softmax layer had 4k nodes context dependent sub-phone targets formed by conventional decision tree context dependent state tying. To make comparisons between different HF optimisation setups, information about the amount of compute time and the number of epochs are also presented. All experiments in this chapter have been conducted on machines using GPU cards of the Tesla K series. For all experiments, decoding was performed using the weak MPE LM[3] at intermediate stages of training to monitor how learning to minimise the MPE loss correlates with the model's ability to minimise the WER, the criterion of interest.

Details of lattice generation: To perform MPE training, word lattices were generated once using the CE trained HMM-DNN model and a pruned bigram LM constructed from an extended transcript containing utterances belonging to both the training and validation dataset. These lattices were then used repeatedly for several iterations of MPE training. Since the official MGB1 dev.sub dataset is used as the validation set, a new pronunciation dictionary was created to handle missing pronunciations.

5.5.2 Effectiveness of using Curvature Information

In [16], it is shown how second order optimisation approaches can help speed up convergence by re-scaling the stochastic update direction $g(\boldsymbol{\theta}_k, \boldsymbol{\xi}_k)$ to adjust for the high non-linearity and ill-conditioning of the objective function. To verify this claim, this section compares an optimisation framework that uses gradient descent with large batch sizes to the HF optimisation framework presented in Sec. 5.2. For this experiment, the batch sizes corresponding to roughly 25hr of audio was used and the maximum number of training epochs was restricted to 15. Figure 5.1 shows the differences in performance between the two optimisers at intermediate stages of training. The plot also presents the

[2] RNNs by maintaining an internal state are better equipped in incorporating longer context [104]

[3] Using a weaker LM allows the acoustic model to have better leverage in choosing the best path during the Viterbi pass.

performance of utterance level standard SGD for comparison. For the system trained with batch styled gradient descent, the best results were found using larger learning rates than the ones used with standard online SGD.

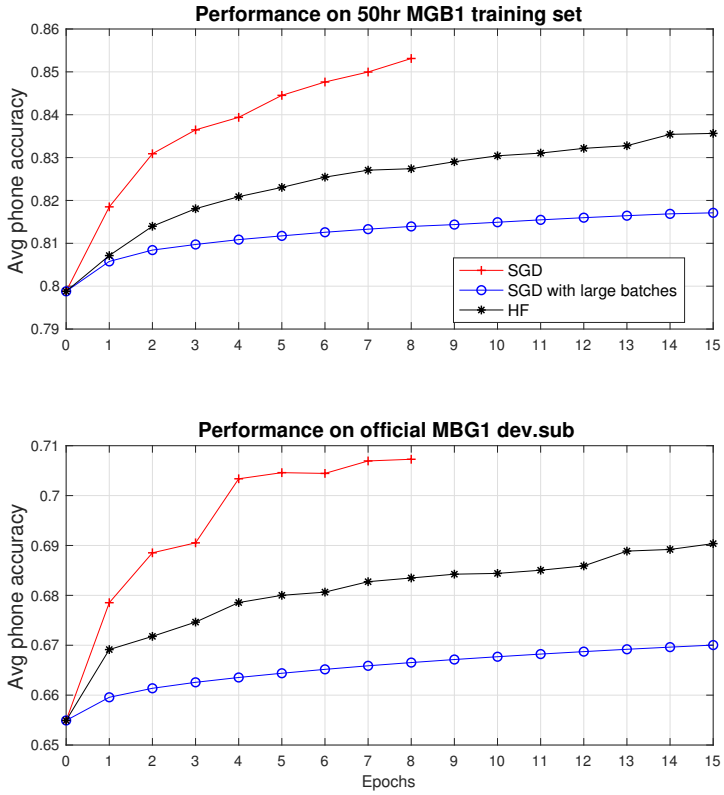


Fig. 5.1 The blue and black plots show the differences in performance between batch styled gradient descent and HF optimisation at intermediate stages of sequence training. To show how both these optimisers fare with standard SGD, the performance achieved at intermediate stages of training with SGD is also given (red).

method	#epochs	#updates	phone acc.		WER
			train	dev.sub	dev.sub
CE	N/A	N/A	0.7988	0.6549	45.2
SGD	8	2.62×10^5	0.8532	0.7044	42.0
HF	15	30	0.8356	0.6904	42.5
SGD-Large Batches	15	30	0.8171	0.670	44.0

Table 5.1 Sequence training and dev.sub MPE phone accuracy/WER for the 50hr training set with HF and variants of gradient descent. The WERs shown at the last column were computed using the weak pruned biased MPE LM.

Average computation cost per Epoch	
Gradient	CG
3.68 hours	0.62 hours

Table 5.2 Computation cost (elapsed time) associated with gradient accumulation and CG training in HF

From Figure 5.1, it can be observed that although increasing the batch size reduces the variance associated with the noisy estimate of the gradient, it now causes the optimiser to perform far fewer updates within each epoch. After 15 epochs, the improvement on the validation dataset achieved with gradient descent using large batch sizes is far less than for standard SGD. The figure also shows how by scaling the gradient direction with local curvature information, the HF optimiser outperforms the batch styled gradient descent within two iterations. From observation of Table 5.1, it can be seen that after performing the same number of updates as the batch styled SGD, the HF optimiser achieves a relative Word Error Rate Reduction (WERR) of 6% over the CE trained model and 3% over the batch styled SGD. However, in comparison to utterance based SGD, the method fails to achieve a better solution after 30 updates. By performing many more updates within few epochs, SGD can be seen to relative WERR of 7% over the CE trained model.

Table 5.2 shows how under the proposed setup, the CG iterations only correspond to 15% of the total computational cost in terms of elapsed time. The low computational overhead along with improved performance over batch styled SGD makes HF an effective optimisation framework that can exploit the use of large batches. Even though the method doesn't achieve better convergence than SGD, HF in comparison to SGD was observed to be highly stable and did not require any form of additional update clipping.

5.5.3 Investigating the influence of using noisy gradients

The CG algorithm minimises a local quadratic by iteratively improving the search direction it initially starts with. In the implementation developed here, the very first iteration of each CG run immediately improves upon the stochastic gradient direction by computing an appropriate step size to be taken along that direction. Thus, initialising CG with a good estimate of the true gradient can allow the algorithm to make significantly more progress in the initial iterations. This proves to be an advantage especially when there is a hard limit on the maximum number of allowed CG iterations. However, this improvement from CG comes at the cost of fewer updates per epoch as large batch sizes are needed to reduce the variance associated with the gradient estimate. Thus, to find a balance between doing more intra-epoch updates and using a better estimate of the gradient, this section investigates the performance of the HF optimiser with different gradient batch sizes. Table 5.3 summarises the setup.

Experimental Run	Gradient batch size	Updates Per Epoch
1	25 hours	2
2	12.5 hours	4
3	6.25 hours	8

Table 5.3 Batch sizes used in different HF setups.

To achieve fair comparisons, the same number of updates were applied at each run, which for this experiment was restricted to 80. The maximum number of allowed CG iterations was kept to 5 and the HF mini-batch size was set 0.5 hours (default). Table 5.4 summarises the results while Figure 5.2 shows the performance of the different variants of the HF optimiser at intermediate stages of training.

method	#epochs	#updates	phone acc.		WER
			train	dev.sub	dev.sub
CE	N/A	N/A	0.7988	0.6549	45.2
HF-25hr	40	80	0.8461	0.6967	42.2
HF-12.5hr	20	80	0.8427	0.6949	42.4
HF-6.25hr	10	80	0.8421	0.6932	42.4

Table 5.4 Sequence training and dev.sub MPE phone accuracy/WER for the 50hr training set with HF using different batch sizes. The WERs were computed using the weak pruned biased LM used in MPE training.

Method	Average cost per epoch with gradient estimate (hours)	Average cost per epoch with CG (hours)
HF-25 hr	3.68	0.62
HF-12.5 hr	3.67	1.39
HF-6.25 hr	3.66	2.94

Table 5.5 Computation cost (elapsed time) associated with gradient accumulation and CG training in HF when employed with different batch sizes

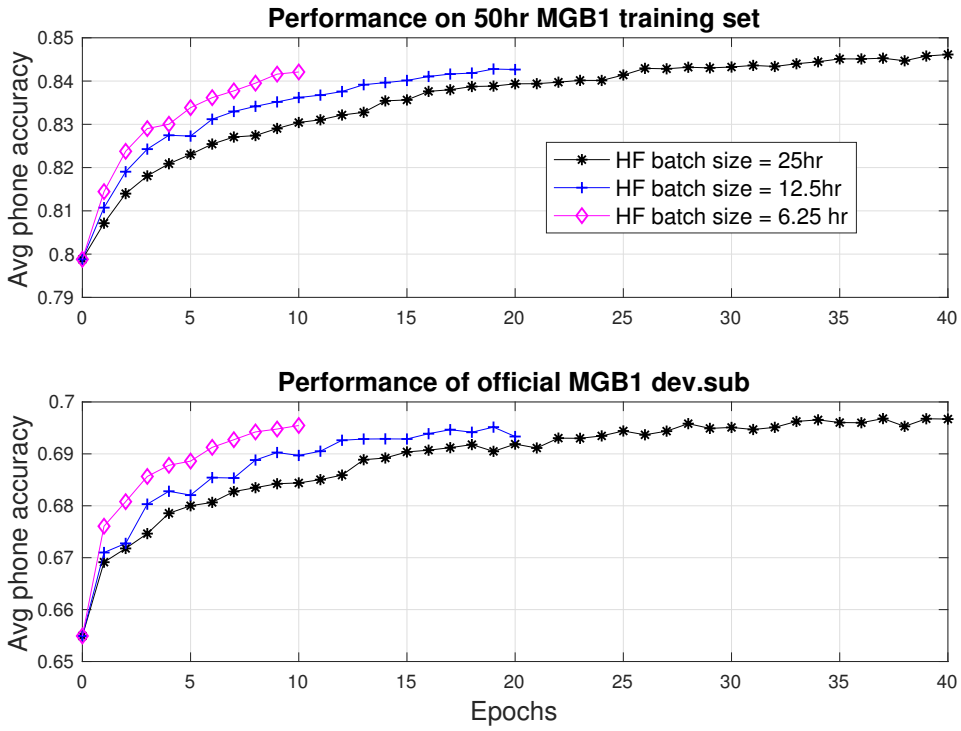


Fig. 5.2 Sequence training a standard DNN with different variants of HF optimisation.

It can be observed from Table 5.4 that gains in the MPE accuracy observed with having a better CG initialisation do carry over to larger WER reductions. Using batch sizes of 25 hours, the improvements achieved by the HF optimiser after 80 updates in comparison to its other variants was found to statistically **significant** under the p-value approach. Although, decreasing the gradient batch size allows more CG runs to performed at each epoch, as evident from Table 5.5, this has the side effect of increasing the contribution of the CG iterations to the cost associated with doing one training epoch. Therefore, having a optimisation framework that employs large batch sizes can

be seen to be advantageous as it leads to both improved convergence and reduced cost through gradient parallelisation.

5.5.4 Improving CG training with Tied Architectures

This section investigates the efficacy of using the modification proposed in Sec. 5.4.3 to improve CG training for tied architectures. Figure 5.3 compares how the evolution of MPE phone accuracy on the training and validation set varies under the proposed modification, while Table 5.6 summaries the results.

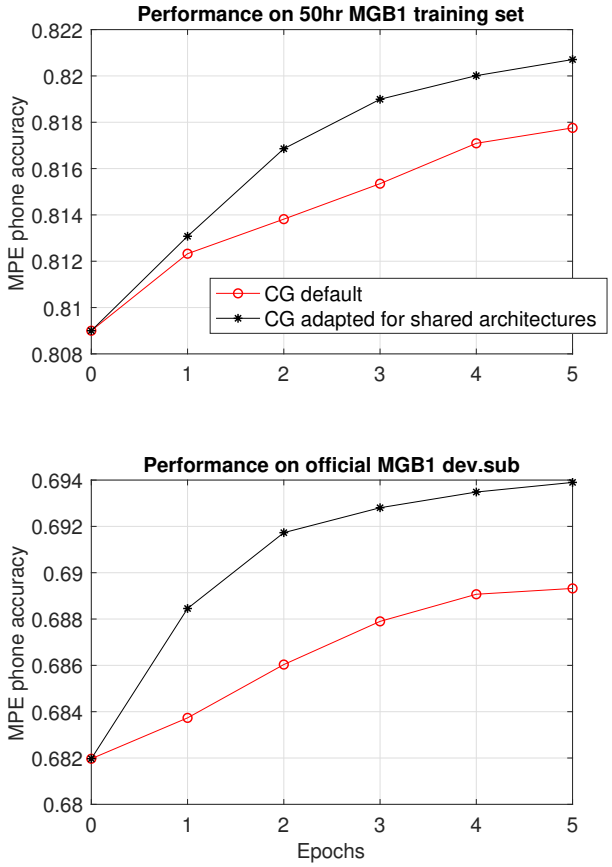


Fig. 5.3 Comparison of CG variants for tied architectures

method	#epochs	#updates	phone acc.		WER with
			train	dev.sub	MPE LM
CE	N/A	N/A	0.809	0.682	43.7
HF	5	10	0.818	0.689	43.0
HF-adapted	5	10	0.821	0.694	42.7

Table 5.6 Sequence training and dev.sub MPE phone accuracy for the 50hr training set with HF adapted for tied architectures. The WERs shown at the last column were computed using the weak pruned biased MPE LM.

From Figure 5.3, it can be observed that by allowing information from un-tied parameters to play more of a role in constructing CG updates, the HF optimiser progressively finds better updates that improve upon the MPE criterion at each iteration of training. An observation on Table 5.6 shows how the improvements on the MPE criterion carry over to better WER reductions on the dev.sub i.e the validation set.

5.6 Summary

This chapter presented the implementation details of the Hessian Free (HF) optimisation method investigated in this thesis. At the beginning of the chapter, a careful analysis of batch and stochastic methods was presented along with a discussion of the potential advantages of using batch styled optimisation frameworks like HF. A key contribution of this thesis has been to provide an effective procedure to stabilise the CG algorithm for DNN models. Table 5.1, shows how under the proposed modifications, effective updates can be generated by running only few iterations of CG. From preliminary experiments on the MGB1 50hr training set, in comparison to standard SGD, the method was found to be quite stable and didn't require any form of addition update clipping. However as shown in Table 5.1, although the method requires far fewer updates, it fails to achieve better generalisation performance on the validation set than standard SGD. This provides the motivation to explore alternative optimisation approaches that provide the same advantages as HF in terms of stability and being data parallel but have the ability to converge to better solutions than SGD. Also in this chapter, an approach to improve CG training for shared architectures was presented. On preliminary experiments with a standard RNN, it was shown how under the proposed modification, the HF optimiser progressively finds better updates at each iteration of training.

Chapter 6

Sequence training with Natural Gradient

In the last chapter, experiments on the 50hr MGB1 training set using the Hessian Free (HF) approach showed that although the method of HF makes more progress per update, it fails to converge to a better solution than SGD. This provides the motivation to explore alternative optimisation approaches that offer the same advantages as HF in terms of stability and being data parallel, but have the ability to converge to better solutions than SGD. One particular approach for training DNNs that has recently experienced renewed popularity is the method of Natural Gradient (NG)[6, 163–165]. This method was first proposed in [25] as an effective optimisation method for training parametric density models with the Maximum Likelihood (ML) objective criterion[1]. Instead of formulating steepest descent in the flat Euclidean parameter space, the method of steepest descent is formulated directly on the model space $\mathcal{M}$ (Sec. 6.1). Section 6.2 shows that when re-framed in the parameter space, this is equivalent to rescaling the gradient direction with the inverse of the model's positive definite Fisher Information (FI) matrix. In DNN literature [6, 164, 165], it has been reported that applying such a rescaling allows DNNs to be trained in a more effective way then using SGD. Section 6.1.1 attributes these observed gains to the fact that for ML methods, using NG becomes increasingly similar to a second order approach in the context of large amount of data. A novel contribution of this thesis is to extend this approach to the domain of MBR objective functions for discriminative sequence training.

For CE training, the approach of NG has previously been applied under a different optimisation framework [7, 165–167]. In all of these works, the authors assume a block

[1]For HMM based discriminative models, this is equivalent to MMI training.

diagonal structure for the FI matrix. In contrast, this work makes no assumption about the structure of the empirical FI matrix and uses a stochastic estimate of the matrix. However in practice, approximating the FI matrix by its empirical estimate results in a matrix that is only guaranteed to be positive semi-definite. To ensure mathematical consistency, this chapter provides the derivation of a damped FI matrix, which when used in an HF framework has the property that directions considered important by the empirical Fisher are traversed first during the initial iterations of CG.

The chapter also addresses the particular issue of overfitting due to criterion mismatch that particularly accompanies MBR training with ReLU DNNs. A key contribution of this thesis is to how this particular form of overfitting can be alleviated by scaling the update directions with the Gauss Newton matrix. The organisation of the chapter is as follows: Section 6.1 develops the necessary geometry needed to develop steepest descent on the DNN function space $\mathcal{M}$. Section 6.2 formulates the NG method for sequence training. Section 6.4 adapts the empirical Fisher to yield a proper Riemannian metric. Section 6.5 presents the experimental setup for training sigmoid based DNN models while results of sigmoid models trained with different optimisers are presented in Sec. 6.6. Section 6.7 extends our investigation to training ReLU based DNNs and addresses the particular issue of overfitting due to criterion mismatch. Section 6.8 investigates the use of standard regularisation methods to alleviate this issue while the last section discusses the particular effect of scaling the DNN frame posteriors with the Gauss Newton matrix.

6.1 Defining a geometry in the space of $\mathcal{M}$

Before formulating steepest descent on the space of $\mathcal{M}$, it is first necessary to define an appropriate geometry in $\mathcal{M}$ that allows measurements such as length and area to be performed on the manifold. The manifold $\mathcal{M}$ consists of the family of all probability distributions $P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})$ that result from different realisations of $\boldsymbol{\theta}$. As each $P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}) \in \mathcal{M}$ corresponds to a unique discrete distribution, the space $\mathcal{M}$ can be represented as a probability simplex $\mathbb{S}_k$ parameterised by a coordinate vector $\boldsymbol{\eta}$ in $\mathbb{R}^k$.[2] A curve on $\mathcal{M}$ is a smooth map $\boldsymbol{c} : (a, b) \subset \mathbb{R} \rightarrow \mathcal{M}$. By endowing $\mathcal{M}$ with a coordinate chart $\boldsymbol{\eta}$, the derivative of $\boldsymbol{c}$ at given point t_0 can then be defined as the linear map from the tangent

[2] k is an arbitrary large number denoting the number of distinct hypotheses.

space (see Appendix A.3) $T_{t_0}\mathbb{R}$ at t_0 to the tangent space $T_{\boldsymbol{c}(t_0)}\mathcal{M}$:

$$\begin{aligned}\mathrm{dc}|_{\mathrm{t}_0}\left(\frac{\mathrm{d}}{\mathrm{dr}}\right) &= \left(\frac{\partial}{\partial \boldsymbol{\eta}_1}, \frac{\partial}{\partial \boldsymbol{\eta}_2} \cdots \frac{\partial}{\partial \boldsymbol{\eta}_k}\right) \begin{bmatrix} \frac{d\boldsymbol{c}^1}{dt}|_{t_0} \\ \\ \frac{d\boldsymbol{c}^2}{dt}|_{t_0} \\ \vdots \\ \frac{d\boldsymbol{c}^k}{dt}|_{t_0} \end{bmatrix} \\ &= \sum_i \frac{d\boldsymbol{c}^i}{dt}|_{t_0} \ \frac{\partial}{\partial \boldsymbol{\eta}_i} \end{aligned} \tag{6.1}$$

Here $\dfrac{d}{dr}$ denotes the basis vector of $T_{t_0}\mathbb{R}$ and $\{\dfrac{\partial}{\partial \boldsymbol{\eta}_i}\}_i$ is the basis of $T_{\boldsymbol{c}(t_0)}\mathcal{M}$ w.r.t the chart $\boldsymbol{\eta}$ (Appendix A.1). Assuming that the curve is regular i.e. dc is injective at every point $t \in (a, b)$, the arc length of the curve from a is:

$$s(\tau) = \int_a^\tau \parallel \langle \mathrm{dc(t)}, \mathrm{dc(t)} \rangle \parallel dt \tag{6.2}$$

To be able to compute the above measurement and similar measurements like area, eqn (6.2) shows that it is necessary to assign the tangent space associated with $\boldsymbol{\eta} \in \mathcal{M}$, a geometric structure of an inner product.

A Riemannian metric [168] is a smooth map that assigns to each $\boldsymbol{\eta} \in \mathcal{M}$ an inner product $I_{\boldsymbol{\eta}}$ on $T_{\boldsymbol{\eta}}\mathcal{M}$. One way to construct such a metric is to establish the notion of a divergence measure in the space $\mathcal{M}$. Let P and Q be two points in $\mathcal{M}$ with coordinates $\boldsymbol{\eta}_P$ and $\boldsymbol{\eta}_Q$. A divergence $D[P : Q]$ [23] is a function of $\boldsymbol{\eta}_P$ and $\boldsymbol{\eta}_Q$ which satisfies the following criteria:

1. $D[P : Q] \geq 0$.
2. $D[P : Q] = 0$ when $\boldsymbol{\eta}_P \equiv \boldsymbol{\eta}_Q$
3. When P and Q are sufficiently close i.e $\boldsymbol{\eta}_Q = \boldsymbol{\eta}_P + d\boldsymbol{\eta}$, the Taylor expansion of D takes the form:

$$D[P : Q] \simeq \frac{1}{2} \sum g_{i,j}(\boldsymbol{\eta}_P) d\boldsymbol{\eta}_i d\boldsymbol{\eta}_j \tag{6.3}$$

where the matrix $\boldsymbol{I}_{\boldsymbol{\eta}_P} = (g_{i,j}(\boldsymbol{\eta}_P))$ is positive definite.
A divergence measure represents a degree of separation from the perspective of an anchored point and should not be confused with the notion of metric. For any two points

P and $Q \in \mathcal{M}$, $D[P:Q]$ is neither guaranteed to be symmetric nor satisfy the triangular inequality. However as shown above, when two points in $\mathcal{M}$ are sufficiently close, the second order Taylor approximation of such a measure allows $\mathcal{M}$ to be annotated with the Riemannian metric $I_{\boldsymbol{\eta}}$. As the space $\mathcal{M}$ represents a set of probability distributions $P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})$, an ideal choice for divergence measure is the KL divergence [169]. The KL-divergence $\mathrm{KL}\left(p_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})\|p_{\boldsymbol{\theta}+\Delta\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})\right)$ is a binary functional that maps the space of distributions $\mathcal{M}$ to $\mathbb{R}$. From an information theory view point, it describes the extra number of bits needed to convert an arbitrary distribution to an anchored distribution.

6.1.1 The Riemannian metric of the KL divergence

By construction, each distribution $P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}) \in \mathcal{M}$ is a smooth map of $\boldsymbol{\theta}$. Thus, the derivatives of the distribution satisfy:

$$\sum_{H} \frac{\partial}{\partial \theta_i} P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}) = \frac{\partial}{\partial \theta_i} \sum_{H} P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}) = \frac{\partial}{\partial \theta_i} 1 = 0 \tag{6.4}$$

As the map between the parameter space X and $\mathcal{M}$ is surjective, adding a small quantity $\Delta\boldsymbol{\theta}$ to the current iterate $\boldsymbol{\theta}$ results in a unique distribution $P_{\boldsymbol{\theta}+\Delta\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})$. The degree of separation of this new point in $\mathcal{M}$ from the previous point $P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})$ is captured by $KL\left(P_{\boldsymbol{\theta}}(\mathcal{H},|\mathcal{O})\|P_{\boldsymbol{\theta}+\Delta\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})\right)$. As $P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})$ itself is function of $\boldsymbol{\theta}$, by utilising compositionally, the local behaviour of this divergence within a convex neighbourhood of $\boldsymbol{\theta}$ can be captured by Taylor's second order approximation:

$$\begin{aligned} KL\left(P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})\|P_{\boldsymbol{\theta}+\Delta\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})\right) =& E_{P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})}\left[\ \log P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}) - \ \log P_{\boldsymbol{\theta}+\Delta\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})\right] \\ \simeq& -\frac{1}{2}\Delta\boldsymbol{\theta}^T E_{P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})}\left[\nabla^2 \log P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})\right]\Delta\boldsymbol{\theta} \end{aligned} \tag{6.5}$$

In (6.5), the first order term is dropped since it equates to zero by (6.4). This allows the KL divergence to be approximate by the above bilinear norm.
For HMM-DNN models, the term $\nabla^2 \log P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})$ in (6.5) represents the sample Hessian of the MMI objective criterion (Sec. 2.4.2.2). Such a matrix decomposes into the particular sum

$$\nabla^2 \log P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}) = -\bar{I}_{\boldsymbol{\theta}} + K_{\boldsymbol{\theta}} \tag{6.6}$$

where

$$\bar{I}_{\boldsymbol{\theta}} = \left(\nabla \log P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})\right)\left(\nabla \log P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})\right)^T$$

and $K_{\boldsymbol{\theta}}$ has the property of $E_{P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})}[K_{\boldsymbol{\theta}}] = 0$. This means in the context of large amounts of data, $\bar{I}_{\boldsymbol{\theta}}$ increasingly becomes a close estimate of the Hessian. For parametric models, the Fisher Information $I_{\boldsymbol{\theta}}$ corresponds to the expected outer product of gradient of the log likelihood:

$$I_{\boldsymbol{\theta}} = E_{P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})}\left[(\nabla \log P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}))\,(\nabla \log P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}))^T\right] \tag{6.7}$$

which equates to the expectation of $\bar{I}_{\boldsymbol{\theta}}$ for disciminative HMM-DNN models. Thus, using the fact that $E_{P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})}[K_{\boldsymbol{\theta}}] = 0$, the Fisher Information $I_{\boldsymbol{\theta}}$ can be seen to be the negative curvature of the MMI criterion in expectation. By substituting the expression of $I_{\boldsymbol{\theta}}$ into (6.5), the KL divergence measure is locally equivalent to the inner product:

$$KL\left(P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})\|P_{\boldsymbol{\theta}+\Delta\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})\right) \simeq \frac{1}{2}\Delta\boldsymbol{\theta}^T I_{\boldsymbol{\theta}}\Delta\boldsymbol{\theta} \tag{6.8}$$

Eqn. (6.8) shows that annotating the tangent space of each $\boldsymbol{\theta} \in X$ with $I_{\boldsymbol{\theta}}$ defines an appropriate Riemannian metric $\boldsymbol{I}_{\eta_P}$ for the KL divergence measure. Equipping $\mathcal{M}$ with $I_{\boldsymbol{\theta}}$ defines a well defined framework to make geometric measurements such as length and areas on the surface $\mathcal{M}$.

6.2 Formulating the Natural Gradient method for Sequence Training

Having setup the geometry needed to probe the space of $\mathcal{M}$, the next task is to develop an algorithm that traverses this space to find a minimiser w.r.t. an appropriate loss functional $L : P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}) \in \mathcal{M} \rightarrow \mathbb{R}$. Each iteration of a typical iterative optimisation algorithm computes an iterate $P_{\boldsymbol{\theta}_{k+1}}(\mathcal{H}|\mathcal{O})$ on the basis of information pertaining to the current iterate $P_{\boldsymbol{\theta}_k}(\mathcal{H}|\mathcal{O})$. Since the Riemannian geometry describes only the local behaviour around $P_{\boldsymbol{\theta}_k}(\mathcal{H}|\mathcal{O})$, the following greedy strategy is employed to generate a candidate function:

$$\hat{P}_{\boldsymbol{\theta}_{k+1}}(\mathcal{H}|\mathcal{O}) = \arg\min_{P_{\boldsymbol{\theta}_{k+1}}} L(P_{\boldsymbol{\theta}_{k+1}}) \text{ s.t } \mathrm{KL}\left(P_{\boldsymbol{\theta}_k}\|P_{\boldsymbol{\theta}_k+\Delta\boldsymbol{\theta}}\right) \leq \epsilon_k \tag{6.9}$$

Let $\mathcal{M}' \subset \mathcal{M}$ be the subset that contains all distributions that place probability mass on areas in the hypothesis space where the previous iterate $P_{\boldsymbol{\theta}_k}(\mathcal{H}|\mathcal{O})$ is non-zero. The KL divergence constraint forces the candidate update chosen at the current iterate to be a

member of $\mathcal{M}'$. In Sec. 2.4.2.3, it was shown how for sequence to sequence classification tasks such as ASR, an appropriate loss function is the MBR family of objective functions:

$$F_{\text{MBR}}(\boldsymbol{\theta}) = \frac{1}{R}\sum_{r}^{R}\left[\sum_{\mathcal{H}} P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}^r, \mathcal{M})L(\mathcal{H}, \mathcal{H}^r)\right] \tag{6.10}$$

MBR training essentially aims to concentrate probability mass: a sufficiently flexible model trained to convergence with MBR will assign a high probability to those hypotheses that have the smallest loss. It is expected that by equipping MBR training with the KL divergence constraint, training will converge to the candidate distributions that concentrate probability mass in those regions covered by the initial $P_{\boldsymbol{\theta}_0}(\mathcal{H}|\mathcal{O})$ that correspond to low loss. In other words, the generative model becomes increasingly more discriminative at each iteration.

As $P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})$ itself is a function of $\boldsymbol{\theta}$, eqn (6.9) can be reformulated as an equivalent optimisation problem in the parameter space:

$$\boldsymbol{\theta}_{k+1} = \arg\min_{\Delta\boldsymbol{\theta}} F_{\text{obj}}(\boldsymbol{\theta}_k) \quad \text{s.t } \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_k)^T I_{\boldsymbol{\theta}_k}(\boldsymbol{\theta} - \boldsymbol{\theta}_k) \leq \epsilon_k \tag{6.11}$$

When the exploration of the parameter space is restricted to be within an region of $\boldsymbol{\theta}_k$, where $F_{\text{obj}}(\boldsymbol{\theta}_k)$ closely approximates its first order approximation, the optimisation problem corresponds to a first order minimisation problem within a trust region. As the minimum of this linearised problem is at infinity, the optimisation dynamics will always jump to the border of the trust region. The constrained optimisation problem solved at each iteration then corresponds to:

$$\Delta\hat{\boldsymbol{\theta}} = \arg\min_{\Delta\boldsymbol{\theta}} F_{\text{obj}}(\boldsymbol{\theta}_k) + \Delta\boldsymbol{\theta}^T\nabla F_{\text{obj}}(\boldsymbol{\theta}_k) + \frac{\lambda}{2}\Delta\boldsymbol{\theta}^T I_{\boldsymbol{\theta}_k}\Delta\boldsymbol{\theta} \tag{6.12}$$

Intuitively, the method of NG can be interpreted as applying a linear transform on the gradient that results in the learning algorithm to land on a region in the parameter space where the new distribution $P_{\boldsymbol{\theta}+\Delta\boldsymbol{\theta}}(\mathcal{Y}|\mathcal{X})$ has the same support as the distribution $P_{\boldsymbol{\theta}}(\mathcal{Y}|\mathcal{X})$. This is illustrated by Figure 6.1.

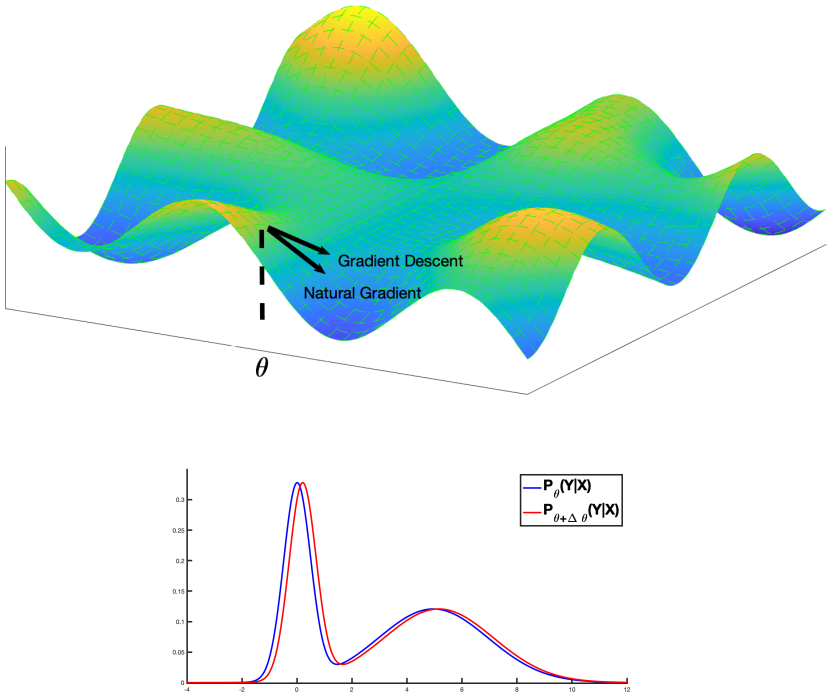


Fig. 6.1 NG learning adapts the direction of steepest descent such that the distribution yielded by the resultant update has the same support as the previous distribution.

6.3 Computing the Natural Gradient Direction

Under the above formulation, solving eqn (6.12) is equivalent to solving a quadratic of the form of eqn (4.3) where $B = \lambda I_{\boldsymbol{\theta}_k}$. Differentiating eqn (6.12) and setting to 0 leads to the NG direction:

$$\Delta\boldsymbol{\theta} = (\lambda I_{\boldsymbol{\theta}_k})^{-1} \nabla F_{\text{obj}}(\boldsymbol{\theta}_k) \tag{6.13}$$

The value λ controls the compromise between the relative importance of minimising the MBR gradient and satisfying the KL divergence constraint. For large networks, with potentially millions of parameters, computing this inverse naively is computationally impractical. To overcome this issue, in present literature [7, 165–167], most approaches assume a block diagonal structure of the Fisher Information Matrix. Among these works, at present the most popular is the approach proposed by Martens and Grosse in their paper [7]. In their approach, the approximation of the FI matrix is built in two stages: first, the rows and columns of the Fisher are divided into groups, each of which

corresponds to all the weights in a given layer, and this gives rise to a block-partitioning of the matrix. These blocks are then approximated as Kronecker products between much smaller matrices. In the second stage, this matrix is further approximated as having an inverse which is either block-diagonal or block-tridiagonal. Before describing the KFAC approach in more detail, it is first necessary to define the following notations. Let

- $\boldsymbol{g}_i$ be the gradient back-propagated to layer i and $\boldsymbol{z}_i$ be the output of that layer.
- $D\boldsymbol{\theta}$ denote the gradient of the log likelihood w.r.t the parameter vector $\boldsymbol{\theta}$. For the weight matrix of the l th layer, this corresponds to DW^l which equates to $DW^i = \boldsymbol{g}_i \boldsymbol{z}_{i-1}^T$
- *vec* denote the operator which vectorizes matrices by stacking their columns together.

Then the Fisher Information matrix can be written as:

$$
\begin{aligned}
I_{\boldsymbol{\theta}} &= E\left[D\boldsymbol{\theta} D\boldsymbol{\theta}^T\right] \\
&= E\left[[vec(DW^1)^T vec(DW^2)^T vec(DW^3)^T..]^T [vec(DW^1)^T vec(DW^2)^T vec(DW^3)^T..]\right] \\
&= \begin{bmatrix} E[vec(DW^1)vec(DW^1)^T] & E[vec(DW^1)vec(DW^2)^T] & ... \\ \vdots & \vdots & \vdots \\ E[vec(DW^L)vec(DW^1)^T] & E[vec(DW^L)vec(DW^2)^T] & .. \end{bmatrix}
\end{aligned}
$$

Under the above decomposition, $I_{\boldsymbol{\theta}}$ can be viewed as an $L \times L$ matrix where L denotes the depth of the network and the $I_{\boldsymbol{\theta}}(i,j)$ block is given by $E[vec(DW^i)vec(DW^j)^T]$. Noting that $DW^i = \boldsymbol{g}_i \boldsymbol{z}_{i-1}^T$ and that $vec(\boldsymbol{u}\boldsymbol{v}^T) = \boldsymbol{u} \otimes \boldsymbol{v}$, then

$$
\begin{aligned}
E[vec(DW^i)vec(DW^j)^T] &= E[vec(\boldsymbol{g}_i \boldsymbol{z}_{i-1}^T)vec(\boldsymbol{g}_j \boldsymbol{z}_{J-1}^T)^T] \\
&= E[(\boldsymbol{g}_i \otimes \boldsymbol{z}_{i-1})(\boldsymbol{g}_j \otimes \boldsymbol{z}_{j-1})] \\
&= E[\boldsymbol{g}_i \boldsymbol{g}_j^T \otimes \boldsymbol{z}_{i-1} \boldsymbol{z}_{j-1}^T]
\end{aligned}
$$

In their paper [7], the authors make the following assumptions:

1. $E[\boldsymbol{g}_i \boldsymbol{g}_j^T \otimes \boldsymbol{z}_{i-1} \boldsymbol{z}_{j-1}^T] \approx [E(\boldsymbol{g}_i \boldsymbol{g}_j^T) \otimes E(\boldsymbol{z}_{i-1} \boldsymbol{z}_{j-1}^T)]$
2. $I_{\boldsymbol{\theta}}$ is assumed to be block diagonal or triagonal i.e only (i,j) blocks are considered where $i = j$ or $i = j + 1$. Using the identity $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$. Under the

diagonal approximation, $I_{\boldsymbol{\theta}}^{-1}$ becomes:

$$I_{\boldsymbol{\theta}}^{-1} = diag\left(E(\boldsymbol{g}_1\boldsymbol{g}_1^T)^{-1} \otimes E(\boldsymbol{z}_0\boldsymbol{z}_0^T)^{-1}, E(\boldsymbol{g}_2\boldsymbol{g}_2^T)^{-1} \otimes E(\boldsymbol{z}_1\boldsymbol{z}_1^T)^{-1} \cdots\right)$$

and to compute $I_{\boldsymbol{\theta}}^{-1}\boldsymbol{v}$, the following identity can be applied $(A \otimes B)(vec(X)) = vec(BXA^T)$ to get the approximate NG direction. In terms of implementation, this corresponds to adding additional transforms between layers. This however ties this approach to optimising only ML based objectives.

3. $E(\boldsymbol{g}_i\boldsymbol{g}_j^T)$ is computed by its Monte-Carlo estimate that is obtained by sampling $\mathcal{Y}$ from the network's predictive distribution and then rerunning the backwards phase of back-propagation as if these were the training targets.

When extended to sequence training, such an approach becomes infeasible as the method can not be easily extended to MBR loss functions. Also in the scenario where $\mathcal{Y}$ corresponds to $\mathcal{H}$, sampling Y corresponds to doing an additional forward backward pass through the lattices. This is makes the method quite expensive in the regime where the method is applied within a *stochastic* context and requires many more updates to converge.

In section 6.1.1, it is shown for sequence level discriminative models, the Fisher Information Matrix takes the form:

$$I_{\boldsymbol{\theta}} = E_{P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})}\left[(\nabla \log P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}))\,(\nabla \log P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}))^T\right] \tag{6.14}$$

$$= E_{P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})}\left[J_{\boldsymbol{\theta}}^T(\nabla L_{MMI}\nabla L_{MMI}^T)J_{\boldsymbol{\theta}}\right] \tag{6.15}$$

By comparing the above decomposition with the Gauss Newton matrix in Sec 5.3, it can be seen that the quadratic problem of eqn (6.12) can also be iteratively solved using linear Conjugate Gradient (CG) algorithm within the HF style optimisation framework. Such an approach makes no assumption about the structure of the FI matrix and can be easily used to do NG training with objective functions other than ML objectives.

It is worth mentioning that when (6.12) is solved approximately with CG, λ controls the speed of training. To understand why, recall how in the CG algorithm highlighted in Sec. 5.2, the step α_k taken at each CG direction is computed as $\alpha_k = \frac{\boldsymbol{r}_k^T\boldsymbol{r}_k}{\boldsymbol{d}_k^T B\boldsymbol{d}_k}$. When B corresponds to $\lambda I_{\boldsymbol{\theta}_k}$, large values of λ effectively scales down the steps taken along the individual conjugate directions.

In practice since the true distribution is unknown, the Fisher Information is approximated

by its Monte-Carlo estimate:

$$\hat{I}_{\boldsymbol{\theta}} = \frac{1}{R}\sum_{r=1}^{R}\left[(\nabla \log P_{\boldsymbol{\theta}}(\mathcal{H}^r|\mathcal{O}^r))\,(\nabla \log P_{\boldsymbol{\theta}}(\mathcal{H}^r|\mathcal{O}^r))^T\right] \tag{6.16}$$

The matrix $\hat{I}_{\boldsymbol{\theta}}$ is the empirical FI matrix and corresponds to the outer product of the Jacobian of the MMI criteria. However such a matrix is only guaranteed to be positive semi-definite and thus its inverse does not exist. To address this issue, the next section shows how the empirical FI matrix can be adapted to yield a proper Riemannian metric which when used with CG guarantees that directions relevant to the empirical Fisher are first traversed during a CG run.

6.4 Adapting the empirical Fisher to a yield a proper Riemannian metric

To recap, a Riemannian metric on a smooth parameter manifold X (see Appendix A.1) is a smooth map that assigns to each $\boldsymbol{\theta} \in X$ an inner product $I_{\boldsymbol{\theta}}$ in $T_{\boldsymbol{\theta}}X$. As $\hat{I}_{\boldsymbol{\theta}}$ is real and symmetric, by the *spectral decomposition theorem* [117], there exists a unitary basis of eigenvectors w.r.t the matrix becomes diagonalisable:

$$\hat{I}_{\boldsymbol{\theta}} \equiv V_{\boldsymbol{\theta}}\Lambda_{\boldsymbol{\theta}}V_{\boldsymbol{\theta}}^T \tag{6.17}$$

where $V_{\boldsymbol{\theta}}$ is a square matrix whose i-th column corresponds to the i-th eigenvector of $\hat{I}_{\boldsymbol{\theta}}$, and $\Lambda_{\boldsymbol{\theta}}$ is the diagonal matrix whose non-zero entries represent the associated eigenvalues. It should be noted that the entries of these two matrices are functions of $\boldsymbol{\theta}$. To keep the notation uncluttered, the dependency on $\boldsymbol{\theta}$ will be dropped for the remainder of this section whenever any of the individual factors in $V_{\boldsymbol{\theta}}\Lambda_{\boldsymbol{\theta}}V_{\boldsymbol{\theta}}^T$ is mentioned. Since $\hat{I}_{\boldsymbol{\theta}}$ is only guaranteed to be positive semi-definite, there will exist zero diagonal entries in Λ resulting in its rank being m where $m < D$ (the dimensionality of the parameter space). Under such circumstances, $\hat{I}_{\boldsymbol{\theta}}^{-1}$ will not exist and it will no longer be possible to endow X with Riemannian metric of the form $\hat{I}_{\boldsymbol{\theta}}$. To address this issue, this section derives an alternative metric to $\hat{I}_{\boldsymbol{\theta}}$ that has the same structure and properties as the damped FI matrix but is guaranteed to be positive definite. From a high level perspective, the construction of the proposed metric is achieved in two stages.

Step 1: partition the tangent space $T_{\boldsymbol{\theta}}X$ into two disjoint subspaces such that one subspace is spanned by eigenvectors of $\hat{I}_{\boldsymbol{\theta}}$ associated with non-zero eigenvalues.

By re-arranging the columns of V such that the eigenvectors associated with the non-zero eigenvalues occupy positions within the first m columns, the image space of $\hat{I}_{\boldsymbol{\theta}}$ can be then essentially captured by the matrix $V_{:,1:m}\Lambda_{1:m,1:m}V^T_{:,1:m}$.

Let ϕ be a map from X to $\mathbb{R}^m$ given by

$$\phi(\boldsymbol{\theta}) = V^T_{:,1:m}\boldsymbol{\theta} \tag{6.18}$$

By the *Replacement theorem* [117], such a map can be shown to be maximal i.e the derivative $\mathrm{d}\phi$ is of full rank. Under the *Implicit Function theorem* [129], there exists a chart h (see Appendix A.1) on X and a neighbourhood $\mathcal{V}$ of $h(\boldsymbol{\theta})$ such that $\phi \circ h^{-1}|_{\mathcal{V}} = \pi|_{\mathcal{V}}$ with $\pi : \mathbb{R}^D \to \mathbb{R}^m$ being the projection map. It is easy to see that from the definition of ϕ that such a chart does exist and corresponds to $V_{:,1:m}$. Thus, within the open neighbourhood $\mathcal{V}$, the derivative of $\phi \circ h^{-1}$ corresponds to the linear map:

$$\mathrm{d}\left(\phi \circ \mathrm{h}^{-1}\right)(\mathrm{v}) = \left[\begin{array}{c|c} \mathbf{I}_{m\times m} & 0_{(m+1)\times D} \end{array}\right](\mathrm{v}) \tag{6.19}$$

With respect to the map ϕ, the tangent space $T_{\boldsymbol{\theta}}X$ can thus be expressed the disjoint sum:

$$T_{\boldsymbol{\theta}_k}X = T_{\phi\circ h^{-1}(\boldsymbol{\theta})}\mathbb{R}^m \bigoplus \ker\left(\mathrm{d}(\phi \circ \mathrm{h}^{-1})\right) \tag{6.20}$$

where $\ker\left(\mathrm{d}(\phi \circ \mathrm{h}^{-1})\right)$ denotes the null space[3]. By the above construction, $T_{\phi\circ h^{-1}(\boldsymbol{\theta})}\mathbb{R}^m$ can now be identified as a subspace of $T_{\boldsymbol{\theta}}X$.

Step 2: assign the identity matrix scaled by a very small number ϵ to the tangent subspace captured by the kernel of $\mathrm{d}(\phi \circ \mathrm{h}^{-1})$. As the entries in the diagonal of $\Lambda_{1:m,1:m}$ are functions of $\boldsymbol{\theta}$, endowing $T_{\phi\circ h^{-1}(\boldsymbol{\theta})}\mathbb{R}^m$ with the inner product $\Lambda_{1:m,1:m}$ is equivalent to assigning a Riemannian metric to the associated subspace in $T_{\boldsymbol{\theta}}X$. Together, the tangent space of X can now be assigned with the following Riemannian metric:

$$\left[\begin{array}{c|c} \Lambda_{1:m,1:,m} & 0 \\ 0 & \epsilon\ \mathbf{I}_{(D-m)\times(D-m)} \end{array}\right]$$

[3] the ker of a linear map $L : N \to W$ between two vector spaces N and W, is the set of all elements $\boldsymbol{v} \in N$ for which $L(\boldsymbol{v}) = 0$

Switching back to the original basis coordinates, the proposed metric then takes the particular form:

$$V \left[\begin{array}{c|c} \Lambda_{1:m,1:m} & 0 \\ 0 & \epsilon \ \mathbf{I}_{(D-m)\times(D-m)} \end{array} \right] V^T = \tilde{I}_{\boldsymbol{\theta}} \tag{6.21}$$

It can be seen that by construction $\tilde{I}_{\boldsymbol{\theta}}$ corresponds to a positive definite matrix whose image space is the direct sum of the image and the kernel space of the empirical Fisher matrix $\hat{I}_{\boldsymbol{\theta}}$. Apart from being a proper Riemannian metric, using CG to solve the appropriate linear system $\tilde{I}_{\boldsymbol{\theta}} \Delta \boldsymbol{\theta} = -\nabla F(\boldsymbol{\theta}_k)$, has one particular advantage: the very first directions explored by the CG algorithm constitute to directions in the image space of $\hat{I}_{\boldsymbol{\theta}}$ [131] i.e directions considered important by the empirical Fisher are traversed first during the initial stages of a CG run.

6.5 Experimental Setup

To evaluate the effectiveness of NG, the method will now be compared with standard SGD and variants of HF using the 50hr and 200hr MGB1 training datasets (Appendix C.2). This section presents results on training hybrid DNN HMMs using traditional fully-connected feed-forward layers. The input to the DNN was produced by splicing together 40 dimensional log-Mel filter bank (FBK) features extended with their delta coefficients across 9 frames to give a 720 dimensional input per frame. These features were normalised at the utterance level for mean and at the show-segment level for variance [22]. The DNN used an architecture of 5 hidden layers each with 1000 nodes with both sigmoid and ReLU activation functions (see Sec. 6.7 for results with ReLU models). The output softmax layer had context-dependent sub phone targets formed by conventional decision tree context-dependent state tying. The output layer contained 4k/6k nodes for the 50h/200h training sets.

To make fair comparisons between different optimisation frameworks all models were trained using lattice-based MPE training. Prior to sequence training, the DNN model parameters were initialised using frame-level CE training. To check the efficacy of each optimisation method, decoding was performed on the validation set at intermediate stages of training using the same weak pruned biased LM used to create the initial MPE lattices. Monitoring the intermediate stages of training allows us to detect the emergence of overfitting due to training criterion mismatch between the MPE criterion and the WER.

6.5.1 Training configuration for SGD

From initial experimental runs, the best results with SGD were observed using a fixed learning rate of 1×10^{-4} for the sigmoid models and a learning rate of 1×10^{-5} for the ReLU model. As an optimiser, SGD on its own can be fairly unstable with network topologies where the gradient has to propagate through many layers. To stabilise training, sequence training with SGD was accompanied by layer dependent gradient clipping to prevent network saturation and ensure smooth propagation of gradients during the training process.

6.5.2 Training configuration for NG & HF-variants

When the number of CG iterations is limited, initialising CG with a good estimate of the gradient is desirable, as within a few iterations a good descent direction can be found. However this comes at the cost of fewer HF/NG updates per epoch as larger batch sizes are needed to reduce the variance associated with the gradient estimates. From the preliminary experiments in Sec. 5.5.3, using batch sizes of roughly 25 hrs was found to give a good balance between the number of updates and using good gradient estimates. The CG mini-batch used in these experiments comprised 0.5 hours of sampled audio. From preliminary experimental runs, the best value of λ for NG training was found to be 16.

6.6 Experimental Results on sigmoid DNNs

6.6.1 Experiments on 50hr MGB1 dataset

On the 50hr training setup, the efficacy of the proposed NG optimisation framework was investigated and compared to both SGD and HF. Table 6.1 summarises the results.

method	#epochs	#updates	phone acc.		WER
			train	dev.sub	dev.sub
CE	N/A	N/A	0.7988	0.6549	45.2
SGD	8	2.62×10^5	0.8476	0.7044	42.0
HF	40	80	0.8461	0.6967	42.2
NG	30	60	0.8677	0.7089	41.6

Table 6.1 Sequence training and dev.sub MPE phone accuracy/WER for the 50hr training set for sigmoid DNN. The WERs shown at the last column were computed using the weak pruned biased LM used in MPE training.

Table 6.1 shows that of the sequence training approaches, the NG method produces the highest training and validation (dev.sub) MPE phone accuracies and the lowest WER using the MPE small biased LM. The method achieves a relative WER reduction of 8% over the CE trained model and 1% over SGD. Replacing the 'Hessian' component of the HF optimiser with the damped FI matrix can be seen to lead to both faster and improved convergence in terms of number of updates. The NG optimiser requires just 60 updates to both better model the training data and improve the WER on the validation set[4]. In comparison to the SGD baseline, both HF and NG training were observed to be highly stable and did not require any form of additional update clipping. This is attributed to the fact that at each iteration, the direction explored is a modified gradient estimate that has been appropriately rescaled using either the local KL divergence or error curvature information.

While both the HF and NG optimisers use more training epochs than SGD, each epoch has far fewer updates and is somewhat more efficient in constructing a geodesic in the parameter space. The relative contribution to the total computational cost by the CG iterations were 15% for HF and 18% NG[5]. Since the batch gradient computations dominate the computation cost and are inherently data parallel, a batch style optimisation framework such as HF/NG can be seen to be time efficient in a synchronous distributed setting. Furthermore, unlike ASGD such frameworks apart from being mathematically sound have the ability to produce identical results on repeated runs.

[4]These gains were found to be consistent when we re-ran training with different sampling schemes.

[5]The extra CG cost associated with NG is attributed to doing more CG iterations. With HF, in our preliminary runs, running CG beyond 5 iterations was not found to yield any significant improvement in the quality of the updates. Since NG is effectively a first order method, we found that in its case we needed to do slightly more iterations at each run.

To investigate whether the WER reductions still hold with stronger LMs, additional decoding passes on the dev.sub validation set were run with 158k vocabulary bigram and trigram LMs used in [22]:

LM	SGD	HF	NG
158k Bigram	39.2	39.4	39.0
158k Trigram	32.8	33.0	32.6

Table 6.2 WER for optimisers on dev.sub with 158k vocabulary bigram/trigram LMs on dev.sub (50hr training set)

From Table 6.2, it can be seen that the trends seen with MPE LM on dev.sub continue with NG giving greater reductions in WER than models trained with either SGD or HF.

6.6.2 Experiments using 200hr MGB1 training dataset

The experiments on the 200hr training set were performed to ensure that the sequence training techniques generalise to a somewhat larger training set (and output layer size) and also to present more detailed comparisons of how training proceeds. On this set, the use of NG was also compared to the Dynamic Stochastic Average Gradient HF variant (DSAG-HF) discussed in Sec 5.4.4. Table 6.3 provides a summary of the results while Fig. 6.2 compares the performance at intermediate stages of training.

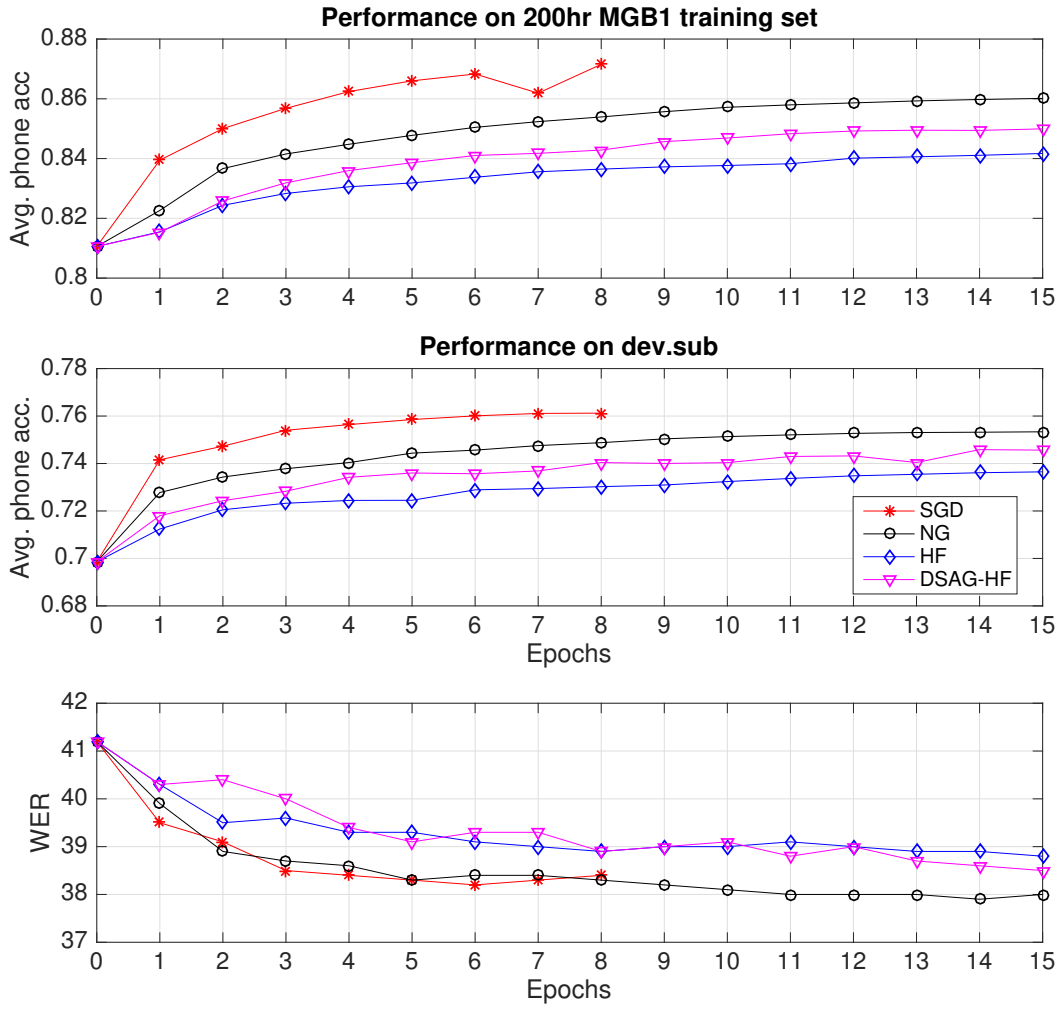


Fig. 6.2 The evolution of the MPE phone accuracy criterion on the training and validation (dev.sub) sets (top 2 graphs) with 200hr sigmoid DNN. Also (lower graph) WER with MPE LM on dev.sub as training proceeds (200hr).

method	#epochs	#updates	phone acc.		WER
			train	dev.sub	dev.sub
CE	N/A	N/A	0.8106	0.6986	41.2
SGD	8	9.27×10^5	0.8684	0.7601	38.2
HF	15	120	0.8417	0.7365	38.8
DSAG-HF	15	120	0.8499	0.7456	38.5
NG	15	120	0.8601	0.7534	37.9

Table 6.3 Performance of the 200hr sigmoid DNN with different optimisers. The WERs were computed using the weak MPE LM.

Table 6.3 shows that as for the 50hr case, sequence training produces increases in MPE phone accuracy and reductions in validation set WER for all optimisers. In this case a total of 8 epochs were performed with SGD and 15 epochs (120 updates) with HF, DSAG-HF and NG. In DSAG-HF, the HF optimiser is equipped with a 'momentum' component which essentially contributes to initialising CG with a weighted sum of the current and previous gradient directions. In Fig. 6.2, it can be observed that for the sigmoid DNN model, such a modification improves both the speed and convergence of HF training. However, this comes at the cost of introducing more hyper-parameters in training. By contrast, replacing the GN with the empirical FI matrix leads to progressively better updates at each epoch. From Table 6.3, the NG method can be seen to achieve a relative WER reduction of 9% over the CE trained model. In comparision to SGD, from Fig. 6.2, the method can be seen to achieve similar WER reductions in validation set over the first 8 epochs but by making far fewer updates. Furthermore, it can be seen that the NG optimiser follows a path in the parameter space where optimising w.r.t MPE criterion better correlates with WER reductions.

LM	SGD	HF	DSAG-HF	NG
158k Bigram	35.0	35.3	35.2	34.7
158k Trigram	29.3	29.3	29.2	29.0

Table 6.4 WER differences between different optimisers on dev.sub with 158k vocabulary bigram/trigram LMs on dev.sub (200hr).

The resulting DNN acoustic models were tested with stronger LMs as for the 50hr setup setup and the results are shown in Table 6.4. It can be seen again that the NG method results in lower WERs than either SGD or the HF variants.

LM	CE	SGD	HF	DSAG-HF	NG
158k Trigram	33.1	30.8	31.0	30.8	30.5

Table 6.5 WER differences between different optimisers on dev.sub2 with 158k trigram (200hr)

Finally, these DNNs were tested on the dev.sub2 set ,as an evaluation set that was not used for setting training hyper-parameters or used in any way during the training process to ensure that the trends observed above generalise. Table 6.5 shows the WER results using the 158k trigram model and shows that again that the model trained with NG achieves the largest reductions in the WER due to sequence training. Furthermore these improvements have been fairly consistent between the validation dev.sub and evaluation dev.sub2 test sets.

On average the NG method can be seen to provide approximately a 1% relative reduction in WER over both the SGD baseline and the DSAG-HF variant. While this improvement it fairly small, it is consistent and a statistical significance test (sign. test of the word error rates at the episode level) showed that the improvement due to NG over each of the other methods is highly statistically significant ($p < 0.001$). Thus even though under the MBR criterion, the method no longer corresponds to a second order approach, it can be seen to be the most effective in training sigmoid based DNN models.

6.7 Sequence training with ReLU DNNs

Having seen the efficacy of the batch styled NG optimisation framework on sequence training sigmoid DNNs, this section extends the investigation to training ReLU DNN systems. For this preliminary work, the training data set used is the 50hr MGB1 dataset. The experiment setup and the DNN topology used is discussed in Sec 6.5. As before, prior to sequence training, the DNN model parameters were initialised using frame-based CE training using standard SGD. For NG and HF training, the setup described in Sec 6.5.2 is used. Table 6.6 compares the CE trained ReLU DNN model against its sigmoid counterpart. It can be seen equipping DNNs with the ReLU activation function allows sequence training to begin from a much better initialisation.

method	#epochs	#updates	phone acc.		WER
			train	dev.sub	dev.sub
CE-sigmoid	N/A	N/A	0.7988	0.6549	45.2
CE-ReLU	N/A	N/A	0.815	0.67	43.8

Table 6.6 Comparisons between DNNs using sigmoid and ReLU activation function on the 50hr MGB1 training set. The WERs are computed using the weak MPE LM

6.7.1 Preliminary Experimental Results

Figure 6.3 compares the performance of the NG optimiser against standard SGD and HF at intermediate stages of training. As the MPE criterion is an approximation to the WER which is the criterion of interest, the third plot of the figure compares WER generalisation performance of the intermediate trained models using the weak MPE LM. Using a weaker LM allows the acoustic model to have better leverage in choosing the best path during the Viterbi pass.

As shown in Figure 6.3, by doing considerably more updates at every epoch, the model trained with SGD achieves the best generalisation performance w.r.t the MPE criterion. Overfitting to the training data can be seen to occur from the 4th epoch onwards. The MPE training criteria is only an approximation of the WER, which is the criterion of interest. The mismatch between these two criterion can be seen to occur from the second epoch onwards[6]. This prevalence of overfitting as a consequence of criterion mismatch was also observed when the SGD optimisation was run with smaller learning rates. In this setup, even though the criterion mismatch occurred in the later epochs, the optimiser failed to achieve better WER reductions than shown in Figure 6.3.

Consistent with the observations seen with sigmoid DNNs, in Figure 6.3, it can be seen that replacing the Gauss Newton Matrix with the damped FI matrix leads to progressively better updates w.r.t MPE criterion. However, like SGD the method can be seen to also suffer from overfitting due to criterion mismatch, where improvements with the MPE criterion fails to correlate with reductions in WER.

To understand this phenomenon of overfitting due to criterion mismatch, various statistics were examined. It was found that the prevalence of this particular form of overfitting is highly correlated with the sharp decrease in the average entropy of the DNN frame posteriors. Figure 6.4 shows how the average entropy of the DNN frame posteriors varies during sequence training for ReLU DNNs.

[6]This behaviour was consistent when decoding was performed with 158k bigram and trigram LM

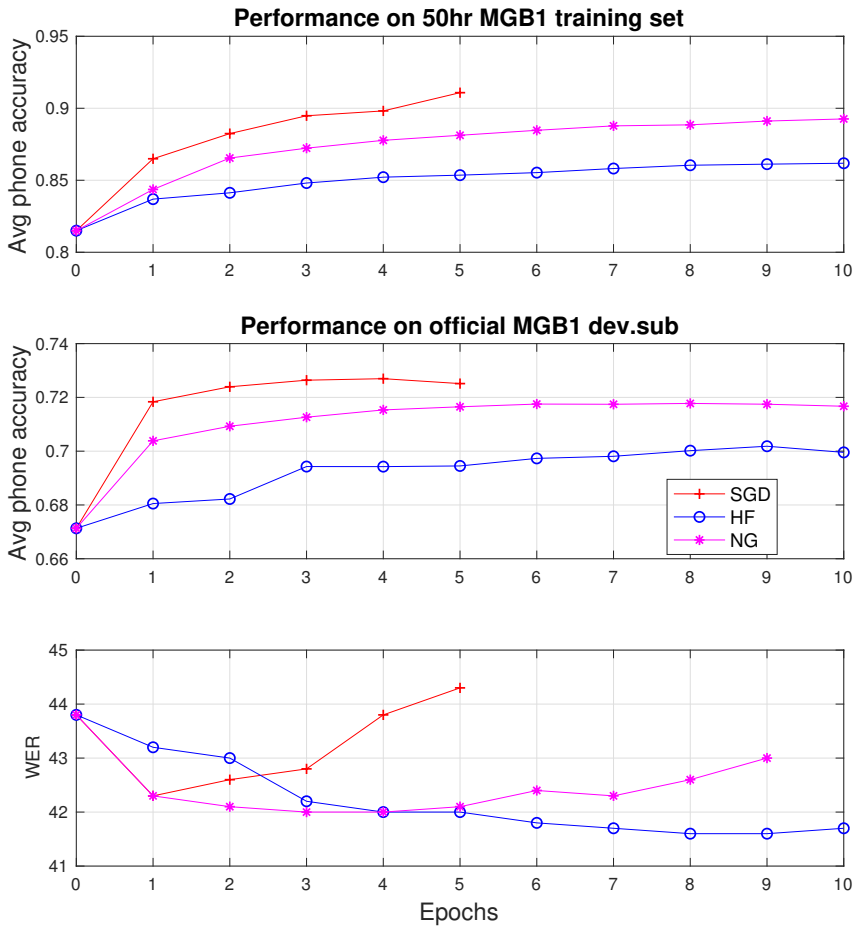


Fig. 6.3 Performance of the 50hr ReLU DNN on both the training and validation set w.r.t MPE criterion at intermediate stages of training with different optimisers. WER performance of the intermediate models using the MPE LM is also given.

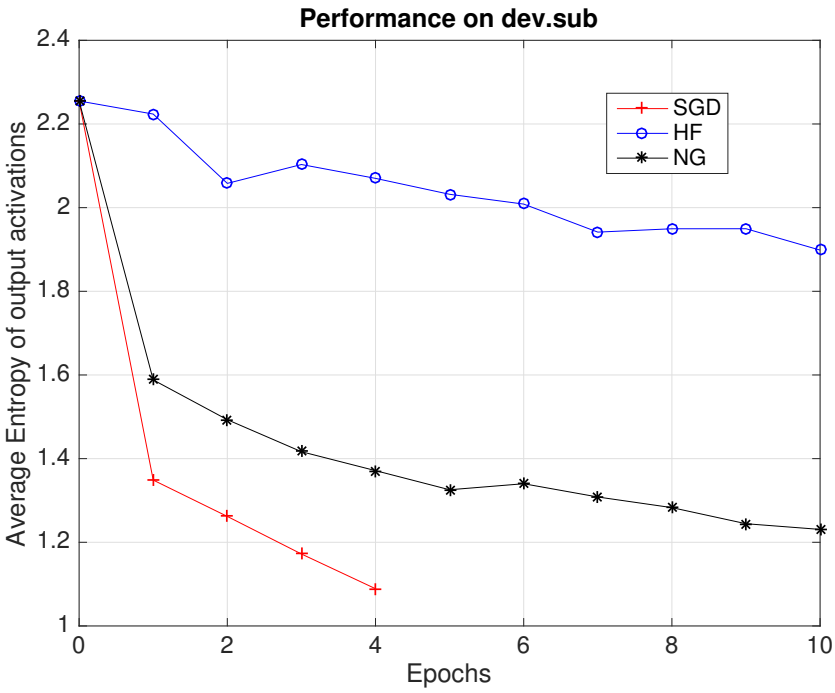


Fig. 6.4 Evolution of average entropy of DNN frame posteriors of ReLU DNNs during sequence training with different optimsers

Comparing Figure 6.4 with the WER plot in Figure 6.3, shows that beyond a certain point if the DNN frame posteriors become overly sharp, training the model using the MPE criterion fails to correlate with the model's ability to achieve better WER reductions. Table 6.7 compares the average entropy statistic of the sigmoid based model of Sec. 6.6.1 after it has been trained with different optimisers. Comparing the statistics with Figure 6.4, it can be seen that the drop in average entropy is not as pronounced with a sigmoid model as it is with a ReLU based system. This might explain why the prevalence of criterion mismatch was not observed with either SGD or NG training for sigmoid DNNs.

method	Average Entropy of DNN frame posteriors
CE	2.514
SGD	2.12
NG	2.05
HF	2.21

Table 6.7 Average entropy of DNN frame posteriors of sequence trained sigmoid DNN models w.r.t to different optimisers.

6.7.2 The Particular Effect of ReLUs

In Sec. 3.2, it was discussed how a standard DNN layer conforms to a bivariate map that maps parameters and the previous layer's representation to a new space:

$$f^l(\boldsymbol{x}, \boldsymbol{\theta}^l) : \mathbf{R}^{d_{l-1}} \times \mathbf{R}^{d_l \times d_{l-1}+1} \rightarrow \mathbf{R}^{d_l}$$

where d_{l-1} is the dimensionality of the layer at depth $l-1$ and d_l is the dimensionality of the transformed space. If $\boldsymbol{\theta}^l$ is fixed, the function $f^l_{\boldsymbol{\theta}^l}$ corresponds to a particular sample from the family of functions $\{f^l(:, \boldsymbol{\theta}^l) : \boldsymbol{\theta}^l \in \mathbf{R}^{d_l \times d_{l-1}+1}\}$. The derivative of $f^l_{\boldsymbol{\theta}^l}$ at a given point $\boldsymbol{x}$ is a linear map from the tangent space at $\boldsymbol{x}$ to the tangent space $f^l_{\boldsymbol{\theta}^l}(\boldsymbol{x})$:

$$df^l_{\boldsymbol{\theta}^l} : T_{\boldsymbol{\theta}^l}\mathbf{R}^{d_{l-1}} \rightarrow T_{f^l(.;\boldsymbol{\theta}^l)}\mathbb{R}^{d_l}$$

Let $\left\{\frac{\partial}{\partial x_i}\right\}_{i=1}^{d_{l-1}}$ be a basis of $T_{\boldsymbol{\theta}^l}\mathbf{R}^{d_{l-1}}$ and $\left\{\frac{\partial}{\partial y_j}\right\}_{j=1}^{d_l}$ be the basis for $T_{f^l(.;\boldsymbol{\theta}^l)}\mathbb{R}^{d_l}$. Then the linear map $df^l_{\boldsymbol{\theta}^l}$ conforms to:

$$df^l_{\boldsymbol{\theta}^l}\left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \cdots, \frac{\partial}{\partial x_{d_{l-1}}}\right) = \left(\frac{\partial}{\partial y_1}, \frac{\partial}{\partial y_2}, \cdots, \frac{\partial}{\partial y_{d_l}}\right) \text{diag}(\nabla_{\boldsymbol{a}^l}\sigma \odot \boldsymbol{a}^l)\ \boldsymbol{W}_{\boldsymbol{\theta}^l} \tag{6.22}$$

where $\boldsymbol{W}_{\boldsymbol{\theta}^l}$ is the layer's weight matrix and $\text{diag}(\nabla_{\boldsymbol{a}^l}\sigma \odot \boldsymbol{a}^l)$ is the diagonal matrix whose non-zero entries correspond to the derivative of the layer's output w.r.t its linear activation. Using the *Mean Value* theorem [129], the behaviour of $f^l_{\boldsymbol{\theta}^l}$ for small changes in $\Delta\boldsymbol{x}$ can be shown to be bounded by:

$$\left\| f^l(\boldsymbol{x} + \Delta\boldsymbol{x}, \boldsymbol{\theta}^l) - f^l(\boldsymbol{x}, \boldsymbol{\theta}^l) \right\| \leq \text{diag}(\nabla_{\boldsymbol{a}^l}\sigma \odot \boldsymbol{a}^l)\ \boldsymbol{W}_{\boldsymbol{\theta}^l}\Delta\boldsymbol{x} \tag{6.23}$$

If $\text{diag}(\nabla_{\boldsymbol{a}^l}\sigma \odot \boldsymbol{a}^l)$ is the identity and in the scenario where the largest singular value of $\boldsymbol{W}_{\boldsymbol{\theta}^l}$ is greater than 1, then small changes in $\boldsymbol{x}$ can potentially induce large variances in the subsequent layers. When the choice of σ is restricted to sigmoid activations, multiplying by $\text{diag}(\nabla_{\boldsymbol{a}^l}\sigma \odot \boldsymbol{a}^l)$ has the effect of ensuring that changes in the parameters at intermediate layers does not induce large changes on the output of subsequent layers. However, when the choice of σ is restricted to ReLUs, the individual entries of $\text{diag}(\nabla_{\boldsymbol{a}^l}\sigma \odot \boldsymbol{a}^l)$ will either be 1 or 0. This has the effect of producing sparse vectors but at the same time allows updates made to parameters at intermediate layers to have a greater influence on the

output of subsequent layers. When intermediate layer representations are learned with CE training, this results in sharper DNN frame posteriors than an equivalent sigmoid CE trained model (as evident from Table 6.7).

Referring back to Figure 6.4 and Table 6.7, the gradient of the MPE loss criterion corresponds to $\nabla \boldsymbol{L}^r_{\text{MBR,t}} = \boldsymbol{\gamma}^r_t \odot \boldsymbol{L}$ where $\boldsymbol{\gamma}^r_t$ is the posterior probability associated with HMM states at time t and the entries of $\boldsymbol{L}$ correspond to the loss associated with these arcs. By the same argument, the use of ReLUs allows the lower layers to learn representations that induce $\boldsymbol{\gamma}^r_t$ to be concentrated on states which contribute to lower loss. For NG training, this effect will be even more pronounced. As NG induces the model to become more discriminative, the use of ReLUs accelerates the DNN frame posteriors to become overly sharp within a few updates (as evident from Figure 6.4).

6.8 ReLU Sequence Training with Standard Regularisers

To reduce this strong prevalence of overfitting due to criterion mis-match seen with SGD and NG, this section investigates the use of standard regularisation methods with ReLU sequence training with SGD and NG. To be more specific, in this section the following are investigated:

1. Using L2 norm regularisation with both NG and SGD.
2. Using SGD with dropout.

6.8.1 L2 norm regularisation

To allow DNN models to generalise well, a popular approach is to employ L2 norm regularisation during training. The use of L2 norm regularisation belongs to the broader class of LP-norm regularisation methods which aims to bias the network to learn small parameters while minimising the original cost function. As discussed in Sec. 4.4.2, the basic principle behind such approaches hinges on modifying the objective function by adding a penalty term.

For L2-norm regularisation, this corresponds to adding a quadratic component $\mu \frac{1}{2} \boldsymbol{\theta}^T \boldsymbol{\theta}$ to the objective function being minimised.

$$\hat{F}_{\text{obj}}(\boldsymbol{\theta}) = F_{\text{obj}}(\boldsymbol{\theta}) + \frac{\mu}{2} \|\boldsymbol{\theta}\|^2$$

In eqn (4.30), it was shown how for SGD training, applying L2 norm regularisation conforms to re-scaling the model parameters by a factor $(1 - \boldsymbol{c}_k\mu)$ before applying the stochastic update rule. For second order methods and our proposed batched style optimisation framework, the effect of L2 norm regularisation is quite different. At each iteration, the function minimised is of the form (4.3). Applying the L2 norm thus corresponds to minimising a 'damped' quadratic of the form:

$$\hat{\Delta\boldsymbol{\theta}} = \underset{\Delta\boldsymbol{\theta}}{\operatorname{argmin}} F_{\text{obj}}(\boldsymbol{\theta}_k) + \Delta\boldsymbol{\theta}^T \nabla F_{\text{obj}}(\boldsymbol{\theta}_k) + \frac{1}{2}\Delta\boldsymbol{\theta}^T B \Delta\boldsymbol{\theta} + \mu\frac{1}{2}\Delta\boldsymbol{\theta}^T\Delta\boldsymbol{\theta} \tag{6.24}$$

where B represents either the scaled $\lambda\hat{I}_{\boldsymbol{\theta}_k}$ or the GN matrix $G_{\boldsymbol{\theta}}$. Differentiating eqn. (6.24) and equating it to 0 yields the critical point $\Delta\boldsymbol{\theta} = (B + \mu I)^{-1}\nabla F_{\text{obj}}(\boldsymbol{\theta}_k)$. When using CG to solve the equivalent linear system $(B + \mu I)\Delta\boldsymbol{\theta} = \nabla F_{\text{obj}}(\boldsymbol{\theta}_k)$, the effect of adding the extra diagonal matrix μI results in more conservative steps along each conjugate direction. This effect parallels with the effect of increasing λ when $B = \lambda\hat{I}_{\boldsymbol{\theta}_k}$. Although both parameters control the size of the CG steps, increasing the value of μ has a less severe effect on the speed of learning. Figure 6.5 compares how SGD and NG varies when used with L2 norm regularisation. For both optimisers, the best choices for μ were found through initial careful grid search.

From the average entropy plot in Figure 6.5, it can be seen that the modification to the NG steps can help to regulate the decrease in entropy. However as evident from the WER and MPE plots, the regularised NG fails to achieve better WER or MPE convergence on the validation set. Since the training configuration used for SGD training is already optimised, adding weight decay to optimiser does little to improve the MPE convergence on the validation dataset. Furthermore, as evident from Figure 6.5, the use weight decay fails to regularise the sharp drop in the average entropy of DNN frame posteriors and thus can be seen to be ineffective in alleviating the problem of overfitting due to criterion mismatch.

6.8.2 Using SGD with Dropout

Apart from L2 norm weight decay, at present the most popular regularisation approach used with SGD training is dropout. The purpose of dropout (Sec. 4.4.4) is to make the network less sensitive to the specific parameters of individual hidden units. Figure 6.6 shows how employing dropout at intermediate stages of training impacts validation performance, while Table 6.8 provides further information on the overall convergence of the optimiser.

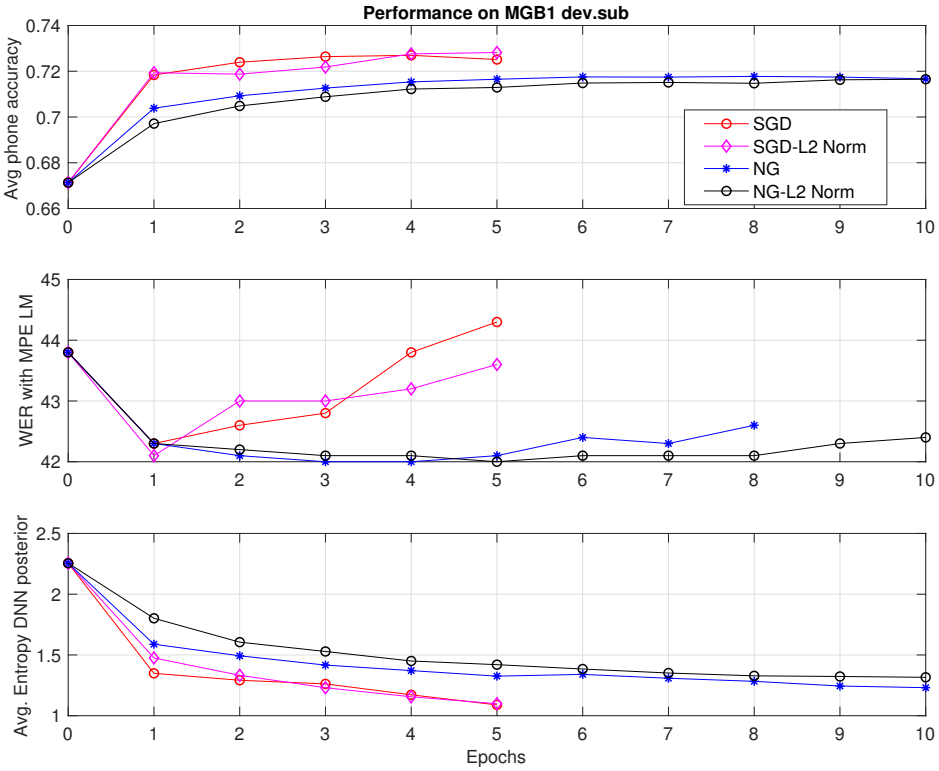


Fig. 6.5 Comparison of the performance of SGD and NG on the validation set when training 50hr ReLU model with L2 norm regularisation. The third plot show the evolution of average entropy of DNN frame posteriors at intermediate stages of training.

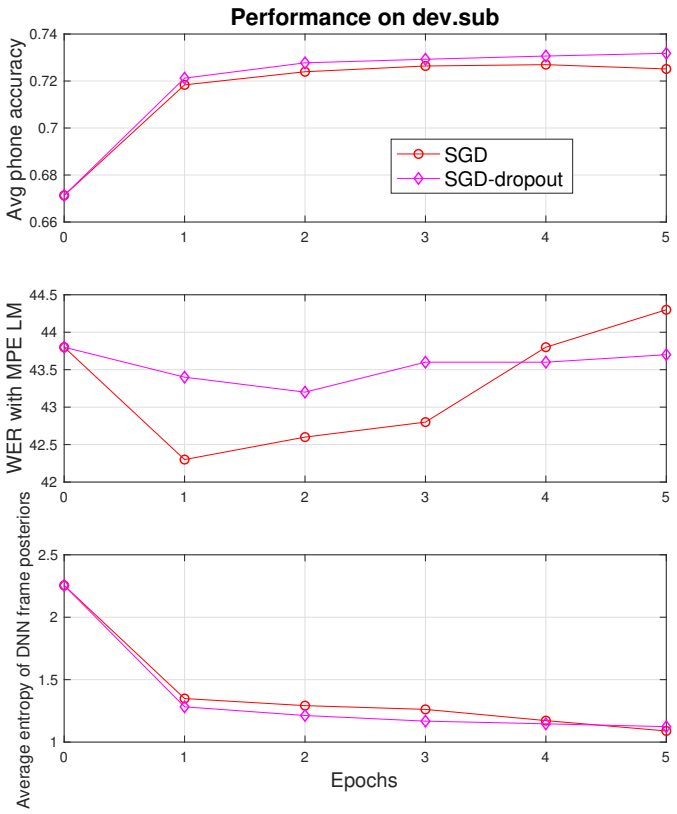


Fig. 6.6 Performance of SGD with dropout on the 50hr ReLU DNN model at intermediate stages of training. The third plot show the evolution of average entropy of DNN frame posteriors at intermediate stages of training.

method	#epochs	#updates	phone acc.		WER
			train	dev.sub	dev.sub
CE	N/A	N/A	0.815	0.671	43.8
SGD	1	3.2×10^4	0.865	0.718	42.3
SGD-Dropout	2	6.4×10^4	0.859	0.728	43.2

Table 6.8 Summary of the performance achieved with sequence training using SGD and SGD-dropout on the 50hr MGB1 training set. The WERs shown at the last column were computed using the weak pruned biased LM used in MPE training.

From Figure 6.6 and Table 6.8, it can be seen that enabling dropout allows the SGD optimiser to achieve better generalisation performance on the validation set w.r.t MPE criterion. However as evident from the WER plot in figure 6.6, the method can be seen to be ineffective in addressing this particular form of overfitting. The use of dropout can be seen to slightly accelerate the decrease in the average entropy of the DNN frame posteriors. These observations were consistent when training was conducted with different sampling schemes.

6.9 ReLU sequence training with Adam

Having seen the ineffectiveness of standard regularisation approaches in alleviating this particular form of overfitting, this section concludes with an experimental investigation of using Adam instead of standard SGD to sequence train ReLU models.

For this investigation, the DNN topology used is a subsampled TDNN. The network consisted of seven hidden layers each with 500 hidden units. The context specification used for the various TDNN layers is as follows: [-2, +2] for layer 1, $\{-1, 2\}$ for layer 2, $\{-3, 3\}$ for layer 3, $\{-7, 2\}$ for layer 4 and [0] for the remaining layers. The output layer consisted of 4k nodes and corresponds to context-dependent sub-phone targets formed by conventional decision tree context-dependent state tying. The input to the model consisted of 40 dimensional log-Mel filter bank features which were normalised at the utterance level for mean and at the show-segment level for variance [22]. Prior to sequence training, the models were initialised with standard CE training. Table 6.9 compares the performance of Adam against standard SGD.

method	#epochs	#updates	WER dev.sub
CE	N/A	N/A	33.2
SGD	2	6.4×10^4	31.2
Adam	2	6.4×10^4	32.5

Table 6.9 Summary of the performance achieved with sequence training using SGD and Adam on 50hr MGB1 training set. The WERs shown at the last column were computed using the trigram 158 k LM.

From Table 6.9, it can be seen that scaling the individual parameter gradients with the inverse of the running average of the squared gradients leads the optimiser to converge to a worse solution (in terms of WER) than before. Thus switching to an adaptive learning scheduler can not be considered as an appropriate solution to handle this form of overfitting.

6.10 Effect of Scaling Directions with the Gauss Newton Matrix

From Figure 6.3, it can be seen that only HF is effective in achieving WER reductions on the 50hr ReLU based model with sequence training . A quick glance on Figure 6.4 shows how scaling the gradient direction with the inverse of $G_{\boldsymbol{\theta}}$ regularises the change in entropy of DNN frame posteriors. To understand why, recall that for DNN models $G_{\boldsymbol{\theta}}$ can be shown to take the particular form of $J_{\boldsymbol{\theta}}^T \nabla^2 L_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}$ where

- $\nabla^2 L_{\boldsymbol{\theta}}$ is the Hessian of the loss function w.r.t the DNN linear output activations, with individual entries being functions of $\boldsymbol{\theta}$.
- $J_{\boldsymbol{\theta}}$ is the Jacobian of the DNN output activations w.r.t $\boldsymbol{\theta}$.

To keep the notation uncluttered, the dependency on $\boldsymbol{\theta}$ will be dropped for the remainder of this section when dealing with the individual factors of the product $J_{\boldsymbol{\theta}}^T \nabla^2 L_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}$. As both $\nabla^2 L$ and the product $J^T \nabla^2 L J$ are real and symmetric, by the *spectral decomposition theorem*:

$$\begin{aligned} J^T \nabla^2 L J &\equiv J^T U \Lambda U^T J \\ &\equiv V \hat{\Lambda} V^T \end{aligned} \tag{6.25}$$

Here $\hat{\Lambda}$ corresponds to Λ extended by padding zero rows and columns to become D by$D$ matrix. An interpretation of each eigenvector $\boldsymbol{v}_i$ in V will now be found. Let k denote the rank of the GN matrix. By re-arranging the columns of V such that the eigenvectors associated with the non-zero eigenvalues occupy positions within the first k columns, the image space of GN can be then essentially captured by:

$$J^T U \ \Lambda \ U^T J = V_{:,1:k} \ \hat{\Lambda}_{1:k,1:k} \ V^T_{1:k,:} \tag{6.26}$$

This construction is very important as it effectively allows us to establish a one to one correspondence between the eigenvectors of $V_{:,1:k}$ with the column vectors of $J^T U$ by matching the diagonal entries in Λ and $\hat{\Lambda}_{1:k,1:k}$. On these relevant directions, solving the linear system with CG corresponds to scaling the steps taken in the eigen directions with the inverse of the curvature/eigen value. To get an intuition behind how scaling with Λ^{-1} impacts training, the structure of individual $\boldsymbol{v}_i$s will now be derived.

A neural network can be essentially viewed as a mapping:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) : \mathbb{R}^{d_{\boldsymbol{x}}} \times \mathbb{R}^D \to \mathbb{R}^k \tag{6.27}$$

where $d_{\boldsymbol{x}}$ is the dimensionality of the input and k is the dimensionality of the output layer. Under this framework, fixing $\boldsymbol{x}$, the map $f_{\boldsymbol{x}}$ corresponds to a particular vector map $f_{\boldsymbol{x}} : \boldsymbol{\theta} \in \mathbf{R}^D \to \mathbf{R}^{d_l}$ that maps the networks parameters to its pre-softmax activations. The derivative of $f_{\boldsymbol{x}}$ at given point $\boldsymbol{\theta}$ is then a linear map from the tangent space at $\boldsymbol{\theta}$ to the tangent space $f_{\boldsymbol{x}}(\boldsymbol{\theta})$ and is given by the matrix J. Formally, the map is defined by:

$$df_{\boldsymbol{x}}(\boldsymbol{\theta}^l) : T_{\boldsymbol{\theta}}\mathbb{R}^D \to T_{f_{\boldsymbol{x}}(\boldsymbol{\theta})}\mathbb{R}^k$$

where

$$df_{\boldsymbol{x}}(\boldsymbol{\theta}) \left(\frac{\partial}{\partial \theta_1}, \frac{\partial}{\partial \theta_2}, \cdots, \right) = \left(\frac{\partial}{\partial y_1}, \frac{\partial}{\partial y_2}, \cdots, \frac{\partial}{\partial y_{d_l}} \right) J$$

Here $\left\{\frac{\partial}{\partial\theta_i}\right\}_{i=1}^{D}$ denotes the basis of $T_{\boldsymbol{\theta}}\mathbb{R}^D$ and $\left\{\frac{\partial}{\partial y_j}\right\}_{j=1}^{k}$ denotes the basis for $T_{f(\boldsymbol{x},:)}\mathbb{R}^k$ and J takes the form:

$$J = \begin{bmatrix} \frac{\partial f_{\boldsymbol{x},1}}{\partial\theta_1} & \frac{\partial f_{\boldsymbol{x},1}}{\partial\theta_2} & \dots \\ \frac{\partial f_{\boldsymbol{x},2}}{\partial\theta_1} & \frac{\partial f_{\boldsymbol{x},2}}{\partial\theta_2} & \dots \\ \vdots & & \\ \frac{\partial f_{\boldsymbol{x},k}}{\partial\theta_1} & \frac{\partial f_{\boldsymbol{x},k}}{\partial\theta_2} & \dots \end{bmatrix}$$

Multiplying J with the matrix U^T yields:

$$U^T J = \begin{bmatrix} \sum_i^k U_{1,i}^T \frac{\partial f_{\boldsymbol{x},i}}{\partial\theta_1} & \sum_i^k U_{1,i}^T \frac{\partial f_{\boldsymbol{x},i}}{\partial\theta_2} & \sum_i^k U_{1,i}^T \frac{\partial f_{\boldsymbol{x},i}}{\partial\theta_3} \dots \\ \sum_i^k U_{2,i}^T \frac{\partial f_{\boldsymbol{x},i}}{\partial\theta_1} & \sum_i^k U_{2,i}^T \frac{\partial f_{\boldsymbol{x},i}}{\partial\theta_2} & \sum_i^k U_{2,i}^T \frac{\partial f_{\boldsymbol{x},i}}{\partial\theta_3} \dots \\ \vdots & & \\ \sum_i^k U_{k,i}^T \frac{\partial f_{\boldsymbol{x},i}}{\partial\theta_1} & \sum_i^k U_{k,i}^T \frac{\partial f_{\boldsymbol{x},i}}{\partial\theta_2} & \sum_i^k U_{k,i}^T \frac{\partial f_{\boldsymbol{x},i}}{\partial\theta_3} \quad \dots \end{bmatrix}$$

Each eigenvector $\boldsymbol{v}_j$ of $J^T\nabla^2 LJ$ can now interpreted as a vector whose ith entry corresponds to a particular weighted sum of the gradients of the DNN pre-softmax activations w.r.t $\boldsymbol{\theta}$.

Switching from the standard basis to the basis spanned by $U^T J$, updates conforming to directions of steepest descent can be expressed as:

$$\Delta\theta = \sum_{j=1}^{k} \alpha \left(\boldsymbol{v}_j^T \nabla F_{\text{obj}}\right) \sum_i U_{j,i}^T \frac{\partial f_{\boldsymbol{x},i}}{\partial\theta}$$

where α is the learning rate. With respect to this basis, scaling with the inverse of the GN matrix effectively corresponds to rescaling the steps taken along individual $\boldsymbol{v}_j$ by a factor $\frac{1}{\zeta_j}$:

$$\Delta\theta = \sum_{j=0}^{k} \frac{1}{\zeta_j} \left(\boldsymbol{v}_j^T \nabla F_{\text{obj}}\right) \sum_i U_{j,i}^T \frac{\partial f_{\boldsymbol{x},i}}{\partial\theta} \tag{6.28}$$

where ζ_j is the eigenvalue associated with $\boldsymbol{v}_j$ in V. Recall that $\nabla^2 L$ can alternatively be presented as $\nabla.(\nabla \boldsymbol{L}_{\text{obj}})$ where $\boldsymbol{L}_{\text{obj}}$ is the loss w.r.t linear DNN output activations. Therefore, eigenvectors $\boldsymbol{u}_j$ in U with large eigenvalues correspond to directions that can induce large changes in the gradient of $\nabla \boldsymbol{L}_{\text{obj}}$. By establishing a one-to one correspondence between eigenvectors of $\nabla^2 L$ with eigenvectors of $J^T\nabla^2 LJ$, it can be seen that re-scaling

with $\frac{1}{\zeta_j}$ effectively de-weights back propagation information carried by those paths through the network that can induce large changes in L_{obj}. In the context of discriminative sequence training, this ensures that the DNN frame posterior distribution does not become overly sharp.

6.11 Summary

This chapter extends the method of Natural Gradient (NG) to the domain of discriminative sequence training. In the DNN literature, the method has been previously applied for non-sequence classification tasks within a framework that employs a block diagonal approximation of the Fisher Information (FI) matrix. This work makes no such assumption and uses a stochastic estimate of the FI matrix. To ensure mathematical consistency, it was shown in Sec. 6.4 how this matrix can be adapted to yield a positive definite matrix, which when used with CG has the property that directions considered important by the empirical Fisher are traversed first at the initial iterations of CG. By replacing the 'Hessian' component of the HF optimiser with the damped FI matrix, the NG method applied within the HF optimisation framework was shown to achieve both better and faster convergence (w.r.t number of updates) than variants of HF. On sigmoid DNNs trained on the 200hr MGB1 dataset, the method was shown to achieve a relative WER reduction of 8% over CE on both the dev.sub and test dataset. Over SGD, the NG method on average was shown to provide approximately a 1% relative reduction (statistically significant at the episode level) in WER.

In addition to extending the NG approach to the domain of MBR objective functions for discriminative sequence training, this chapter also addresses the difficulty of getting WER reductions with ReLU sequence training. From preliminary experiments with ReLU models, it was observed that both NG and SGD suffer from overfitting due to criterion mismatch. Improvements made w.r.t. the MPE criterion on the validation set fail to correlate with the WER reductions. A key contribution of this thesis has been to show how this problem can be alleviated by scaling the update directions with the GN matrix.

Chapter 7

Introducing NGHF: a novel optimisation approach

The goal of learning is to identify a viable candidate $f(\mathcal{O}, \boldsymbol{\theta}) \in \mathcal{M}$ which when used in conjunction with decision theory allows optimal classifications to be made with respect to a given risk criterion. For parametric models such as DNNs, this is achieved by formulating an appropriate optimisation problem on the parameter space X. As different realisations of model parameters lead to different $f(\boldsymbol{x}, \boldsymbol{\theta})$, finding a solution of the optimisation problem in X corresponds to selecting a good candidate function from $\mathcal{M}$.

In practice, designing a good optimisation algorithm to effectively probe X is a complex task. To improve the training of deep models, the general strategy has not always been to improve the optimisation algorithm. Instead, many improvements in the optimisation of deep models have come from re-designing the models [11–13] such that a good set of parameters can be found by standard gradient descent. This thesis however has taken a different route. In Sec. 6.1, by elucidating the space of functions $\mathcal{M}$ with the minimum geometry needed to make measurements of length, area etc, it was shown how the method of steepest descent can be formulated directly on the space of $\mathcal{M}$. When framed in a Hessian Free (HF) styled optimisation framework, the Natural Gradient (NG) method was shown to be the most effective optimiser in achieving WER reductions with discriminative sequence training for sigmoid models (Sec. 6.6). However, as highlighted in Sec 6.2, the efficacy of the method is reliant on the initial model initialisation used for sequence training. From preliminary experiments with ReLU DNNs, it was observed that NG similar to SGD was ineffective in achieving consistent WER reductions from a sharp CE initialised model. In comparison, the HF approach, which was previously found to sub-optimal with sigmoid DNNs, was observed to be

the most effective optimiser in achieving consistent WER reductions with ReLU based models. Seeing that different optimisers work best with different models, it is attractive to have a consistent optimisation framework that is agnostic to the choice of activation function.

This chapter addresses the design considerations that must be considered when developing an effective optimisation algorithm, and proposes a novel optimisation approach to effectively probe the DNN function space $\mathcal{M}$. The chapter begins with the development of the topological structure of the manifold $\mathcal{M}$ associated with DNN based models and discusses the minimum property required for optimisation methods to be well defined on the space $\mathcal{M}$. The chapter then introduces a new optimisation called NGHF that uses both the direction of steepest descent on a probabilistic manifold and local curvature information to effectively probe $\mathcal{M}$. Apart from being well defined on $\mathcal{M}$, the method when framed within the batch style HF optimisation framework will be shown to achieve the greatest WER reductions from sequence training with both sigmoid and ReLU based models. The derivation of the method relies on an alternative derivation of Taylor's theorem using the concepts of manifolds, tangent vectors and directional derivatives from the perspective of Information Geometry [23, 24]. Appendix A contains a glossary of terms referenced in this work but a more in-depth discussion can be found in Amari's text book [23].

This chapter is organised as follows: Section 7.1 develops the underlying geometric structure of the function space $\mathcal{M}$ and presents the minimum property needed for optimisation methods to be well defined on $\mathcal{M}$. Section 7.3 presents an alternative re-derivation of Taylor's theorem and formulates first and second order optimisation problems as minimisation problems in the tangent space of X. Using this alternative formulation of Taylor's theorem, the method of NGHF is developed in Sec. 7.5 and the efficacy of the method is shown in Sec. 7.6 using experiments on 200hr MGB1 training set with different feed forward architectures. The chapter concludes with experimental results from sequence training recurrent networks with the different optimisation methods.

7.1 Structure of Function Space of DNN Models

Although the parameter manifold X essentially captures the space of all functions $\mathcal{M}$ that can be generated by a particular model, the manifold cannot be used directly as a coordinate space of $\mathcal{M}$. DNNs as a consequence of their hierarchical architecture suffer from parameter space symmetries [86, 170] where multiple distinct choices of $\boldsymbol{\theta}$ can give

rise to the same input-output mapping. This prevents a one to one correspondence between vectors in X and functions in $\mathcal{M}$. The symmetries are induced by transpositions applied to intra layer nodes, where nodes are switched by interchanging all the weights (and the bias) leading both into and out of them[1]. Such a transposition can be seen to leave the network input-output mapping function unchanged but results in a new vector $\hat{\boldsymbol{\theta}}$. For M hidden units in a layer, assuming that the activation function is not odd, the number of such permutations is $M!$. Thus, for a network with K distinct layers, the overall weight space symmetry factor will be $K \times M!$.

To develop the notion of a coordinate space for the manifold $\mathcal{M}$, it is first necessary to formally define the concept of permutation acting on a vector space $\boldsymbol{\theta}$. A permutation map on $\boldsymbol{\theta}$ corresponds to a particular form of bijective linear map with the property that

- every row and column contains precisely a single 1 with 0s everywhere else. In other words, every row and column correspond to a distinct 1-K encoding.
- the transpose of the matrix is its inverse.

Let G denote the set of all permutation maps associated with the parameter vector $\boldsymbol{\theta}$. Equipping G with function composition operator '$\circ$' gives the set the structure of a group [171]:

1. $P \circ Z \in G$ for all $P, Z \in G$ (closed operation).
2. $(P \circ Z) \circ K = P \circ (Z \circ K)$ (associative law).
3. The identity map i_G belongs to G,
4. Each map has a unique inverse.

Let $N \subset G$ be the subset of permutations that induce parameter space symmetries in X. It is easy to see that such a set when equipped with the function composition operator '$\circ$' corresponds to a subgroup of G. Using this set, it is now possible to define an equivalence relation on X where the equivalence class for a vector $\boldsymbol{\theta}$ is defined as the set:

$$[\boldsymbol{\theta}] = \{\hat{\boldsymbol{\theta}} \in X | \exists P \in N \text{ which maps } \hat{\boldsymbol{\theta}} \xrightarrow{P} \boldsymbol{\theta}\} \tag{7.1}$$

[1]For odd activation functions, such symmetries can also arise from switching the signs of the incoming and out-going parameters of a node

Intuitively, the enforcement of an equivalent relation can be viewed as inherently gluing together points in the space, resulting in its geometric structure to considerably change. Figure 7.1 shows the action of particular equivalence relation that clusters points along the boundary of a plane to the same equivalence class. From the figure, it can be seen that by gluing together the edges, the equivalence relation inherently morphs the two dimensional plane into a torus.

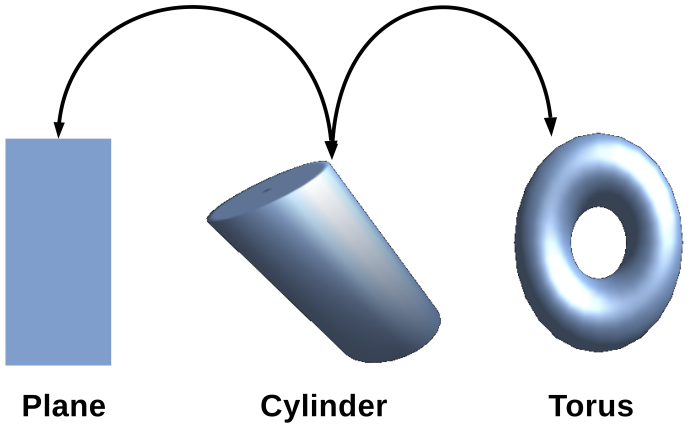


Fig. 7.1 Illustration of an equivalence relation that clusters points along the boundary of the plane. This effectively corresponds to gluing the edges of the plane to form an torus.

Under the equivalence relation given in eqn 7.1, the resultant quotient space $X/\sim$ now exhibits a one-one correspondence with the function space $\mathcal{M}$. In other words, each function $f(\boldsymbol{x}, \boldsymbol{\theta}) \in \mathcal{M}$ corresponds to the realisation of a unique element in $X/\sim$.

7.2 Formulating optimisation directly on the function manifold $\mathcal{M}$

The goal of statistical inference is to use observed data to identify the best viable candidate $f(\boldsymbol{x}, \boldsymbol{\theta}) \in \mathcal{M}$ that avoids rote memorisation and guarantees optimal classification performance. To aid good selection, a candidate f is picked from $\mathcal{M}$ that minimises the expected loss using some form of risk measure over the adequately selected family of predictive models:

$$\hat{f}(\boldsymbol{x}, \boldsymbol{\theta}) = \arg \min_{f(\boldsymbol{x}, \boldsymbol{\theta}) \in \mathcal{M}} \mathbb{E}\mathrm{L}[f(\boldsymbol{x}, \boldsymbol{\theta})] \tag{7.2}$$

Using the one to one correspondence between $X/\sim$ and $\mathcal{M}$, the above optimisation problem can be formulated as an equivalent optimisation problem in the $X/\sim$:

$$[\hat{\boldsymbol{\theta}}] = \arg\min_{[\boldsymbol{\theta}]\in X/\sim} \mathbb{E}\mathrm{L}[f(\boldsymbol{x}, \boldsymbol{\theta})] \tag{7.3}$$

Formulating optimisation in this partitioned space $X/\sim$ is more attractive as the function that gives the lowest expected loss corresponds to a unique element in $X/\sim$ i.e the partitioned space guarantees the existence of a unique global optimum.

Let $\boldsymbol{\theta}_0$ be a element of an equivalent class $[\boldsymbol{\theta}]$. Given this starting point, an optimisation algorithm acting on X will aim to construct a path $\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3 \cdots \boldsymbol{\theta}_k$ towards a local optimum using the following update rule at each training iteration:

$$\boldsymbol{\theta}_i = \boldsymbol{\theta}_{i-1} + \Delta\boldsymbol{\theta}_i$$

To be able to traverse on the quotient space $X/\sim$, it is necessary to have a well defined framework that allows similar paths to be constructed in $X/\sim$.

Let $\boldsymbol{\theta}'_0 = P(\boldsymbol{\theta}_0)$ where $P \in N$. Initialised with this different initial point, if the sequence of points $\boldsymbol{\theta}'_0, \boldsymbol{\theta}'_1, \boldsymbol{\theta}'_2, \boldsymbol{\theta}_3 \cdots \boldsymbol{\theta}'_k$ constructed by the chosen optimiser are such that at each point satisfies $\boldsymbol{\theta}'_i = P(\boldsymbol{\theta}_i)$, then such an algorithm can be seen to provide a well defined framework to traverse along the quotient space $X/\sim$. Eqn (7.4), shows how satisfying this criterion corresponds to the class of methods whose updates are invariant w.r.t permutation maps in X:

$$\begin{aligned}
\boldsymbol{\theta}'_k &= P(\boldsymbol{\theta}_k) \\
\boldsymbol{\theta}'_{k-1} + \Delta\boldsymbol{\theta}'_k &= P(\boldsymbol{\theta}_{k-1} + \Delta\boldsymbol{\theta}_k) \\
\boldsymbol{\theta}'_{k-1} + \Delta\boldsymbol{\theta}'_k &= P(\boldsymbol{\theta}_{k-1}) + P(\Delta\boldsymbol{\theta}_k)
\end{aligned}$$

As a consequence of weight space symmetry

$$\begin{aligned}
\boldsymbol{\theta}'_{k-1} + \Delta\boldsymbol{\theta}'_k &= \boldsymbol{\theta}'_{k-1} + P(\Delta\boldsymbol{\theta}_k) \\
\Delta\boldsymbol{\theta}'_k &= P(\Delta\boldsymbol{\theta}_k)
\end{aligned} \tag{7.4}$$

Intuitively, for an optimisation algorithm to be well defined on $\mathcal{M}$ means that if the algorithm requires k steps to find a local minimum, then when re-initialised with the permuted initialisation $\boldsymbol{\theta}'_0 = P(\boldsymbol{\theta}_0)$, the algorithm should take the same number of steps to find a minimum which has the same critical value as before.

7.2.1 Gradient Descent is well defined

With the method of Gradient Descent, the updates produced at each training iteration equates to a scaled multiple of the negative gradient:

$$\Delta\boldsymbol{\theta}_k = -\alpha\nabla F_{\text{obj}}(\boldsymbol{\theta}_k)$$

where F_{obj} is the objective function. When re-parameterised under a permutation $P \in N$, the gradient of F_{obj} w.r.t the individual components of the re-parameterised vector $\boldsymbol{\theta}'_k$ is:

$$\begin{aligned}\frac{\partial F_{\text{obj}}}{\partial \theta'_i}|_{\boldsymbol{\theta}'_k} &= \sum_j \frac{\partial F_{\text{obj}}}{\partial \theta_j}|_{\boldsymbol{\theta}_k}\frac{\partial \theta_j}{\partial \theta'_i} \\ &= \sum_j \frac{\partial F_{\text{obj}}}{\partial \theta_j}|_{\boldsymbol{\theta}_k} P^{-1}_{j,i}\end{aligned}$$

which in vector form can be compactly written as:

$$\nabla F_{obj}(\boldsymbol{\theta}'_k) = P^{-\text{T}}\nabla F_{\text{obj}}(\boldsymbol{\theta}_k) \tag{7.5}$$

Using the fact that for each $P \in N$, its transpose is its own inverse, $\nabla F_{\text{obj}}(\boldsymbol{\theta}'_k)$ corresponds to:

$$\Delta\boldsymbol{\theta}'_k = -P^{-\text{T}}\nabla F_{\text{obj}}(\boldsymbol{\theta}_k) \equiv -P\ \nabla F_{obj}(\boldsymbol{\theta}_k)$$

The above analysis shows although that method of gradient descent is not invariant to general linear re-parameterisations, the method is well defined to probe the space $X/\sim$.

7.2.2 NG and HF methods are well defined

Both NG and HF belong to the class of optimisation methods that employ the following update rule at each training iteration:

$$\Delta\boldsymbol{\theta}_k = -(B_{\boldsymbol{\theta}_k})^{-1}\nabla F_{\text{obj}}(\boldsymbol{\theta}_k)$$

where $B_{\boldsymbol{\theta}}$ corresponds to either the FI matrix or an estimate of the Hessian. When re-parameterised under a permutation $P \in N$,

$$\begin{aligned}\Delta\boldsymbol{\theta}'_k &= -(P^{-\mathrm{T}} B_{\boldsymbol{\theta}_k} P^{-1})^{-1} P^{-\mathrm{T}} \nabla F_{\text{obj}}(\boldsymbol{\theta}_k) \\ &= -P(B_{\boldsymbol{\theta}_k})^{-1} \nabla F_{\text{obj}}(\boldsymbol{\theta}_k) \\ &= P(\Delta\boldsymbol{\theta}_k)\end{aligned} \tag{7.6}$$

From eqn (7.6), it can be seen that both the NG and HF methods present a well defined framework to effectively probe the manifold $\mathcal{M}$. When $\mathcal{M}$ corresponds to the space of probabilistic distributions that can captured by the given model, NG apart from being well defined utilises information of the underlying geometry of $\mathcal{M}$. However the method as shown in Sec 6.2 doesn't take into account the second order information of the non ML objective functions. In the following sections, a detailed derivation of a well defined novel optimisation method will be provided which utilises both information about the underlying geometry of $\mathcal{M}$ and second order information of the cost surface. The method will be shown to achieve the greatest WER reductions from discriminative sequence training with relatively large models using either ReLU and sigmoid activation functions.

7.3 Alternative derivation of Taylor's theorem

A curve on a parameter manifold X (Appendix A.1) is a continuous map $c : (a, b) \subset \mathbb{R} \rightarrow X$. Let $\mathcal{U}$ be an open convex neighbourhood of the current iterate $\boldsymbol{\theta}_k$. Thus for any point $\boldsymbol{\theta}$ in $\mathcal{U}$, $\exists$ a curve of the form $\boldsymbol{\theta}_k + t(\boldsymbol{\theta} - \boldsymbol{\theta}_k)$ where $t \in [0, 1]$ that is contained in $\mathcal{U}$.

Let $c : [0, 1] \rightarrow X$ be a continuous curve such that:

$$c(t) = \boldsymbol{\theta}_k + t\Delta\boldsymbol{\theta} \tag{7.7}$$

where $\Delta\boldsymbol{\theta}$ corresponds to arbitrary offset from $\boldsymbol{\theta}_k$ such that c in contained in $\mathcal{U}$. The derivative of c at given point t_0 is then the linear map from the tangent space (Appendix A.3) at t_0 to the tangent space $T_{c(t_0)}X$:

$$\mathrm{dc}|_{\mathrm{t_0}}\left(\frac{\mathrm{d}}{\mathrm{dr}}\right) = (\frac{\partial}{\partial\hat{\theta}_1}, \frac{\partial}{\partial\hat{\theta}_2} \cdots \frac{\partial}{\partial\hat{\theta}_D}) \begin{bmatrix} \Delta\theta_1 \\ \Delta\theta_2 \\ \vdots \\ \Delta\theta_D \end{bmatrix}$$
$$= \sum_i \Delta\theta_i \frac{\partial}{\partial\theta_i} \tag{7.8}$$

where $\frac{d}{dr}$ denotes the basis vector of $T_{t_0}\mathbb{R}$.

Let $F(\boldsymbol{\theta})$ be a germ (Appendix A.2) that corresponds to a smooth map from the parameter manifold X to $\mathbb{R}$. In the context of optimisation, this corresponds to the smooth objective training criterion. The derivative of $F(\boldsymbol{\theta})$ at any given point $\boldsymbol{\theta}$ is a linear map from the tangent space $T_{\boldsymbol{\theta}}X$ to the tangent space $T_{F(\boldsymbol{\theta})}\mathbb{R}$:

$$\mathrm{dF}|_{\boldsymbol{\theta}}(\sum_{\mathrm{i}} \mathrm{a^i} \frac{\partial}{\partial\theta_{\mathrm{i}}}|_{\boldsymbol{\theta}}) = \left(\frac{d}{dr}\right) \left[(\frac{\partial F}{\partial\hat{\theta}_1}|_{\boldsymbol{\theta}}, \frac{\partial F}{\partial\hat{\theta}_2}|_{\boldsymbol{\theta}} \cdots \frac{\partial F}{\partial\hat{\theta}_D}|_{\boldsymbol{\theta}})\right] \begin{bmatrix} a^1 \\ a^2 \\ \vdots \\ a^D \end{bmatrix} \tag{7.9}$$

Here the vector $\nabla F(\boldsymbol{\theta}_k)$ is the Jacobian and denotes the particular vector in $T_{\boldsymbol{\theta}}M$ that yields the greatest directional derivative (Appendix A.4).

Constraining F on the curve c is equivalent to applying the composite map $F \circ c$ from $\mathbb{R} \to \mathbb{R}$. The derivative of such a map at given point t_0 will now correspond to a linear map from a tangent space in $T_{t_0}\mathbb{R}$ to the tangent space of $T_{F\circ c(t_0)}\mathbb{R}$. By applying chain rule such a linear map can be shown to correspond to:

$$\mathrm{d(F \circ c)}|_{\mathrm{t_0}}\left(\frac{\mathrm{d}}{\mathrm{dr}}\right) = \left(\frac{d}{dr}\right)[F \circ c]_{t_0} \tag{7.10}$$

$$= \left(\frac{d}{dr}\right) \left[(\frac{\partial F}{\partial\hat{\theta}_1}|_{\boldsymbol{\theta}_k + t_0\Delta\boldsymbol{\theta}}, \frac{\partial F}{\partial\hat{\theta}_2}|_{\boldsymbol{\theta}_k + t_0\Delta\boldsymbol{\theta}} \cdots \frac{\partial F}{\partial\hat{\theta}_D}|_{\boldsymbol{\theta}_k + t_0\Delta\boldsymbol{\theta}})\right] \begin{bmatrix} \Delta\theta_1 \\ \Delta\theta_2 \\ \vdots \\ \Delta\theta_D \end{bmatrix}$$

$$= \frac{d}{dr} \sum_i \Delta\theta_i \frac{\partial F}{\partial\theta_i}|_{\boldsymbol{\theta}_k + t\Delta\boldsymbol{\theta}} \tag{7.11}$$

Using the fact that $F \circ c$ is differentiable and corresponds to a map from $\mathbb{R} \to \mathbb{R}$, by the *fundamental theorem of calculus*:

$$\begin{aligned} F(\boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}) &= F(\boldsymbol{\theta}_k) + \int_0^1 (F \circ c(t))' \, dt \\ &= F(\boldsymbol{\theta}_k) + \int_0^1 \sum_i \Delta\theta_i \frac{\partial F}{\partial \theta_i} (\boldsymbol{\theta}_k + t\Delta\boldsymbol{\theta}) \, dt \\ &= F(\boldsymbol{\theta}_k) + \sum_i \Delta\theta_i \int_0^1 \frac{\partial F}{\partial \theta_i} (\boldsymbol{\theta}_k + t\Delta\boldsymbol{\theta}) \, dt \end{aligned} \tag{7.12}$$

Since individual terms $\left(\int_0^1 \frac{\partial F}{\partial \theta_i}(\boldsymbol{\theta}_k + t\Delta\boldsymbol{\theta}) \, dt \right)$ themselves are smooth functions defined on the convex neighbourhood of $\boldsymbol{\theta}_k$, eqn (7.12) can be expanded even further by recursively applying the fundamental theorem of calculus:

$$\begin{aligned} F(\boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}) &= F(\boldsymbol{\theta}_k) + \sum_i \frac{\partial F}{\partial \theta_i}(\boldsymbol{\theta}_k)\Delta\theta_i + \sum_i \Delta\theta_i \int_0^1 \left(\sum_j \Delta\theta_j \int_0^1 \frac{\partial^2 F}{\partial \theta_j \partial \theta_i}(\boldsymbol{\theta}_k + t\Delta\boldsymbol{\theta}) dt \right) dt \\ &= F(\boldsymbol{\theta}_k) + \sum_i \frac{\partial F}{\partial \theta_i}(\boldsymbol{\theta}_k)\Delta\theta_i + \sum_i \Delta\theta_i \sum_j \Delta\theta_j \int_0^1 \left(\int_0^1 \frac{\partial^2 F}{\partial \theta_j \partial \theta_i}(\boldsymbol{\theta}_k + t\Delta\boldsymbol{\theta}) dt \right) dt \end{aligned} \tag{7.13}$$

As the function $\frac{\partial^2 F}{\partial \theta_j \partial \theta_i}(\boldsymbol{\theta})$ is continuous, when $\Delta\boldsymbol{\theta}$ is sufficiently small, $\frac{\partial^2 F}{\partial \theta_j \partial \theta_i}(\boldsymbol{\theta}_k + t\Delta\boldsymbol{\theta})$ can be approximated by $\frac{\partial^2 F}{\partial \theta_j \partial \theta_i}(\boldsymbol{\theta}_k + t\Delta\boldsymbol{\theta}) \, |_{t=0}$ Under this approximation, the local behaviour of the germ $F(\boldsymbol{\theta})$ can be approximated by:

$$\begin{aligned} F(\boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}) &\simeq F(\boldsymbol{\theta}_k) + \sum_i \frac{\partial F}{\partial \theta_i}(\boldsymbol{\theta}_k)\Delta\theta_i + \sum_i \Delta\theta_i \int_0^1 \left(\sum_j \Delta\theta_j \int_0^1 \frac{\partial^2 F}{\partial \theta_j \partial \theta_i}(\boldsymbol{\theta}_k) dt \right) dt \\ &\simeq F(\boldsymbol{\theta}_k) + \sum_i \frac{\partial F}{\partial \theta_i}(\boldsymbol{\theta}_k)\Delta\theta_i + \frac{1}{2} \sum_i \Delta\theta_i \sum_j \Delta\theta_j \frac{\partial^2 F}{\partial \theta_j \partial \theta_i}(\boldsymbol{\theta}_k) \end{aligned} \tag{7.14}$$

In vector notation, this corresponds to:

$$F(\boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}) \simeq F(\boldsymbol{\theta}_k) + \Delta\boldsymbol{\theta}^T \nabla F(\boldsymbol{\theta}_k) + \frac{1}{2} \Delta\boldsymbol{\theta}^T H \Delta\boldsymbol{\theta} \tag{7.15}$$

with entries $H_{j,i}$ of the Hessian matrix corresponding to $\left(\frac{\partial^2 F}{\partial\theta_j \partial\theta_i}(\boldsymbol{\theta}_k)\right)$. The above expression corresponds to Taylor's second order approximation. Re-deriving this expression from the perspective of manifold theory shows how the product $\Delta\boldsymbol{\theta}^T \nabla F(\boldsymbol{\theta}_k)$ can be interpreted as an **inner product** between vectors $\Delta\boldsymbol{\theta}$ and $\nabla F(\boldsymbol{\theta}_k)$ in $T_{\boldsymbol{\theta}} X$. This interpretation will be used in the next section.

7.4 Formulating Taylor's Quadratic as a Minimisation problem in Tangent Space

Instead of minimising the objective function directly, second order methods focus on minimising the quadratic function of (7.15) at each iteration. The quadratic corresponds to a local model of the behaviour of $F(\boldsymbol{\theta})$ within a convex neighbourhood of the current iterate $\boldsymbol{\theta}_k$. By deriving Taylor's second order approximation from the perspective of manifold theory, $\Delta\boldsymbol{\theta}$ can now be interpreted as a particular choice of a tangent vector from $T_{\boldsymbol{\theta}_k} X$ while $\nabla F(\boldsymbol{\theta}_k)$ represents the particular vector in $T_{\boldsymbol{\theta}} X$ that yields the greatest directional derivative under the linear map $\mathrm{dF}|_{\boldsymbol{\theta}}$. As $F(\boldsymbol{\theta}_k)$ is a constant, solving the minimisation problem in eqn (7.15) is equivalent to solving the following minimisation problem in $T_{\boldsymbol{\theta}_k} X$:

$$\Delta\hat{\boldsymbol{\theta}} = \arg \min_{\Delta\boldsymbol{\theta} \in T_{\boldsymbol{\theta}_k} X} F(\boldsymbol{\theta}_k) + \langle \Delta\boldsymbol{\theta}, \nabla F(\boldsymbol{\theta}_k) \rangle + \frac{1}{2} \Delta\boldsymbol{\theta}^T H \Delta\boldsymbol{\theta} \tag{7.16}$$

where $\langle \Delta\boldsymbol{\theta}, \nabla F(\boldsymbol{\theta}_k) \rangle$ corresponds to the standard inner product between vectors in $T_{\boldsymbol{\theta}_k} X$ and $\Delta\boldsymbol{\theta}^T H \Delta\boldsymbol{\theta}$ corresponds to a linear map $g : \boldsymbol{u} \in T_{\boldsymbol{\theta}_k} X \to \mathbb{R}$.

7.4.1 Relating Gradient Descent to Natural Gradient

Under this framework, first order methods can be seen to solve the following optimisation problem in the tangent space $T_{\boldsymbol{\theta}_k} X$:

$$\Delta\hat{\boldsymbol{\theta}} = \arg \min_{\Delta\boldsymbol{\theta} \in T_{\boldsymbol{\theta}_k} X} F(\boldsymbol{\theta}_k) + \langle \Delta\boldsymbol{\theta}, \nabla F(\boldsymbol{\theta}_k) \rangle \tag{7.17}$$

Since X is a manifold, the inner product endowed on the tangent space $T_{\boldsymbol{\theta}} X$ at any point $\boldsymbol{\theta}$ need not be just the identity matrix. The parameter manifold X can be equipped with any form of a Riemannian metric, a smooth map that assigns to each $\boldsymbol{\theta} \in X$ an

inner product in $T_{\boldsymbol{\theta}}X$. When the underlying model corresponds to a discriminative probabilistic model $P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})$, Sec. 6.2 showed an ideal choice of $I_{\boldsymbol{\theta}}$ corresponds to the expectation of the outer product of the Maximum Mutual Information (MMI) (3.3) gradient:

$$I_{\boldsymbol{\theta}} = E_{P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O})}\left[(\nabla \log P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}))\,(\nabla \log P_{\boldsymbol{\theta}}(\mathcal{H}|\mathcal{O}))^T\right]$$

Equipping the tangent space $T_{\boldsymbol{\theta}}X$ with the above Riemannian metric allows lengths of paths traversed to be interpreted in the parameter space as changes in the KL divergence (Sec. 6.1.1). When X possesses such a structure, performing first order optimisation within a trust region defined by $I_{\boldsymbol{\theta}}$ can be shown to produce the update $\Delta\boldsymbol{\theta} = -I_{\boldsymbol{\theta}}^{-1}\nabla F(\boldsymbol{\theta})$ at each iteration.

Using the fact that $I_{\boldsymbol{\theta}}$ is symmetric and positive definite, it is also possible to endow X with a Riemannian metric of the form $I_{\boldsymbol{\theta}}^{-1}$ by the spectral decomposition theorem. With respect to such a metric, solving the minimising problem of (7.17) in the tangent space $T_{\boldsymbol{\theta}_k}X$ becomes equivalent to performing NG on the parameter surface. Hence, recasting the optimisation problem to a minimisation problem in $T_{\boldsymbol{\theta}_k}X$ provides a nice framework to relate gradient descent with the method of NG.

7.5 Formulating NGHF

In Sec. 7.4, it was shown how solving the second order Taylor's quadratic is equivalent to solving an equivalent minimisation problem in the tangent space $T_{\boldsymbol{\theta}_k}X$, equipped with the standard inner product. As X is a manifold, equipping the tangent space $T_{\boldsymbol{\theta}_k}X$ with the inverse of the Fisher Information matrix $I_{\boldsymbol{\theta}}^{-1}$, and considering the entire eqn. (7.16) leads to the critical point:

$$\Delta\boldsymbol{\theta} = -H^{-1}I_{\boldsymbol{\theta}}^{-1}\nabla F(\boldsymbol{\theta}_k) \tag{7.18}$$

We call this approach NGHF as the resultant direction can be seen to correspond to an NG direction regularised by the curvature information of the underlying objective function.

When re-parameterised under a permutation $P \in N$, the updates produced by the method as shown below are well defined in $X/\sim$.

$$\begin{aligned}\Delta\boldsymbol{\theta}'_k &= -(P^{-\mathrm{T}}H_{\boldsymbol{\theta}_k}P^{-1})^{-1}(P^{-\mathrm{T}}I_{\boldsymbol{\theta}_k}P^{-1})^{-1}P^{-\mathrm{T}}\nabla F_{\mathrm{obj}}(\boldsymbol{\theta}_k)\\ &= -P(H_{\boldsymbol{\theta}_k})^{-1}P^{T}P(I_{\boldsymbol{\theta}_k})^{-1}\nabla F_{\mathrm{obj}}(\boldsymbol{\theta}_k)\end{aligned}$$

Using the fact that the inverse of P is its transpose,

$$\begin{aligned}\Delta\boldsymbol{\theta}'_k &= -P(H_{\boldsymbol{\theta}_k})^{-1}(I_{\boldsymbol{\theta}_k})^{-1}\nabla F_{\mathrm{obj}}(\boldsymbol{\theta}_k)\\ &= P(\Delta\boldsymbol{\theta}_k)\end{aligned} \tag{7.19}$$

Computing the individual inverse matrix scalings directly is expensive in terms of both computation and storage. Using the HF approach, the individual matrix scalings is approximated by solving equivalent linear systems using the Conjugate Gradient (CG) algorithm (Sec. 5.2.1). In this work, we approximate $I_{\boldsymbol{\theta}}$ with $\tilde{I}_{\boldsymbol{\theta}}^{-1}$ (see Sec. 6.4) and the Hessian with the GN matrix $G_{\boldsymbol{\theta}}$. A quick look at the CG algorithm presented in Sec. 5.2 shows how the update direction proposed at each CG iteration corresponds to:

$$\Delta\boldsymbol{\theta}_{k+1} \leftarrow \Delta\boldsymbol{\theta}_k + a_k\boldsymbol{d}_k$$

where $\boldsymbol{d}_k$ represents the current conjugate direction. At the very first iteration of CG, this corresponds to direction the algorithm has been initialised with. In contrast to NG and HF, in the method of NGHF, the initial direction corresponds to approximation of the NG direction instead of the gradient. Thus when $H^{-1}\left(I_{\boldsymbol{\theta}}^{-1}\nabla F(\boldsymbol{\theta}_k)\right)$ is solved with CG, the resultant update corresponds to:

$$\Delta\boldsymbol{\theta} = w_1\Delta\boldsymbol{\theta}_{NG} + w_2\Delta\boldsymbol{\theta}_{HF} \tag{7.20}$$

a weighted combination of the NG direction and conjugate directions computed using local curvature information. In Sec. 7.6.2, employing such a direction will be shown to lead to paths traversed in the parameter manifold where optimisation w.r.t. the MPE criterion closely mirrors with reductions in the WER.

7.6 Experiments on Feed Forward Architectures

7.6.1 Experimental Setup

The efficacy of the proposed novel optimisation method was evaluated on the MGB1 200hr training set (see Appendix C.2). The official MGB1 dev.sub set was employed as a validation set and consists of 5.5 hours of audio data. As in Sec. 6.6, the test set dev.sub2 was used to estimate the generalisation performance of candidate models. This comprises roughly 23 hours of audio from the remaining 35 shows belonging to the MGB1 dev.full set. Further details related to the data preparation can be found in [22].

In this chapter, experimental investigation has been extended to sub-sampled Time Delay Neural Networks (TDNNs) (Sec. 3.4.1) along with standard DNNs. Models with both ReLU and sigmoid activations were trained. The architecture used for DNNs consisted of five hidden layers each with 1000 nodes. For TDNNs, the network topology consisted of seven hidden layers each with 1000 hidden units. The context specification used for the various TDNN layers is as follows: [-2, +2] for layer 1, $\{-1, 2\}$ for layer 2, $\{-3, 3\}$ for layer 3, $\{-7, 2\}$ for layer 4 and [0] for the remaining layers. For both models, the output layer consisted of 6k nodes and corresponds to context-dependent sub-phone targets formed by conventional decision tree context-dependent state tying. For DNNs, the input to the model was produced by splicing together 40 dimensional log-Mel filter bank (FBK) features extended with their delta coefficients across 9 frames to give a 720 dimensional input per frame. While for TDNNs, only the 40 dimensional log-Mel filter bank features were considered. For all experiments, the input features were normalised at the utterance level for mean and at the show-segment level for variance [22].

All models were trained using lattice-based MPE training [70]. Prior to sequence training, the model parameters were initialised using frame-level CE training. To track the occurrence of overfitting due to training criterion mismatch, decoding was performed, at intermediate stages of sequence training, on the validation set using the same weak pruned biased LM used to create the initial MPE lattices. To evaluate the generalisation performance of the trained models, a 158k word vocabulary trigram LM was used to decode the validation and test set.

Training configuration for SGD: The best results with SGD were achieved through annealing of the learning rates at subsequent epochs. The initial learning rates were found through a grid search. For TDNNs, using momentum was found to yield the best WERs.

Training configuration for NGHF, NG & HF: The recipe described in Sec. 5.4.1 was used: gradient batches corresponding to roughly 25 hours and 0.5 hrs of audio were sampled for each CG run. In all experiments, running each CG run beyond 8 iterations was not found to be advantageous. The CG computations varied between 18% to 26% of the total computational cost. It should be noted that the DSAG-HF method is not explored in this chapter. In preliminary experiments with TDNNs, the method was not found to provide better WER reductions than HF.

7.6.2 Experimental Results

Figure 7.2 compares the performance of various optimisation methods on training a ReLU based 200hr HMM-DNN model while Table 7.1 shows the performance of these optimisers for a ReLU based HMM-TDNN system.

method	#epochs	#updates	phone acc.		WER with
			train	dev.sub	MPE LM
CE	N/A	N/A	0.870	0.754	36.9
SGD	4	4.64×10^5	0.888	0.760	36.4
NG	4	32	0.913	0.789	36.2
HF	4	32	0.899	0.783	35.9
NGHF	4	32	0.911	0.791	**35.6**

Table 7.1 Sequence training with different optimisers on the TDNN-ReLU model. WERs on dev.sub.

From Figure 7.2 and Table 7.1, it can be seen that among all the optimisers, NGHF is the most effective in achieving the largest WER reductions on dev.sub with the weak MPE LM. At each iteration, the update produced by the method conforms to $\Delta\boldsymbol{\theta} = w_1\Delta\boldsymbol{\theta}_{NG} + w_2\Delta\boldsymbol{\theta}_{HF}$, a weighted combination of NG direction and conjugate directions computed using local curvature information. In Fig. 7.2, it can be seen that by utilising information from both the KL divergence in the probabilistic manifold and local curvature information, the proposed method follows a path where optimising the MPE criterion better correlates with achieving reductions in WER. With the ReLU TDNN model, as evident from Table 7.1, the method also achieves the greatest WER reductions on the validation set. To investigate whether the relative gains are not constrained to only ReLU based systems, equivalent systems using sigmoids were trained. Table 7.2 compares the performance of the various optimisers on the validation set with the different models

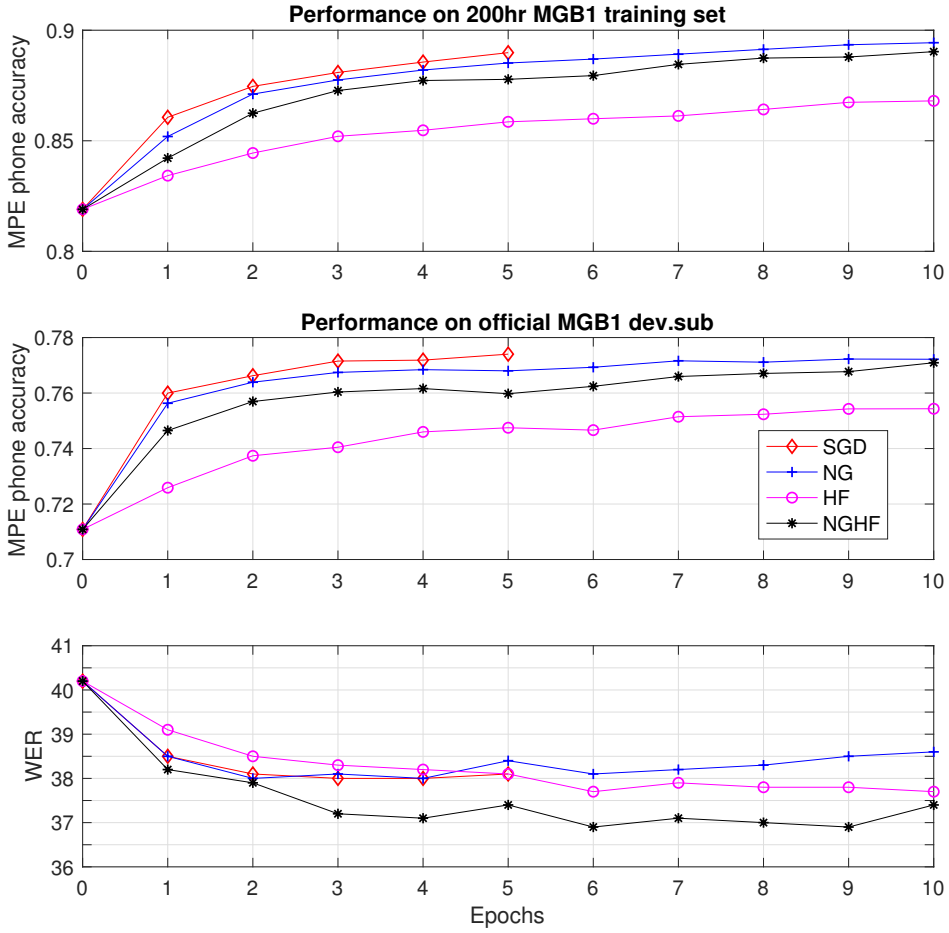


Fig. 7.2 Evolution of MPE phone accuracy criterion on the training and validation (dev.sub) sets with different optimisers in sequence training the ReLU based DNN (top 2 graphs). Also (lower graph) WER with MPE LM on dev.sub as training proceeds.

using the 158k LM. Here a stronger LM has been to use to ensure that WER reductions hold even when coupling between the AM and LM is changed during decoding.

Model	CE	MPE			
		SGD	NG	HF	NGHF
DNN-ReLU	30.9	29.9	29.8	28.9	**28.1**
TDNN-ReLU	28.6	28.5	28.7	28.1	**27.5**
DNN-sigmoid	31.9	29.3	29.0	29.3	**29.0**
TDNN-sigmoid	28.5	27.1	26.9	27.0	**26.6**

Table 7.2 WERs on MGB1 dev.sub from sequence training with 158k trigram LM.

From Table 7.2, it can be seen that again models using NGHF achieve the largest reductions in WER. For ReLU based models, sequence training with NGHF achieves a relative Word Error Rate Reduction (WERR) of 9% with the DNN and 4% with the TDNN over the CE trained models. Whereas with the sigmoid based models, the method achieves a relative WERR of 6% with the DNN and 7% with the TDNN over the CE trained models. Compared to SGD, NGHF is especially effective with the ReLU based models. For the DNN, the method achieves a relative WERR of 6% over SGD, while with the TDNN the relative WERR is 4%.

Finally, the generalisation performance of the trained models were estimated by performing Viterbi decoding on dev.sub2 using 158k LM. Results are shown in Table 7.3.

Model	CE	MPE			
		SGD	NG	HF	NGHF
DNN-ReLU	32.3	31.9	31.4	30.6	**29.8**
TDNN-ReLU	30.6	29.8	30.6	29.6	**29.3**
DNN-sigmoid	33.2	30.8	30.5	30.9	**30.5**
TDNN-sigmoid	29.9	28.6	28.2	28.4	**27.9**

Table 7.3 WERs on MGB1 dev.sub2 with 158k trigram LM.

It can be observed that as before the model trained with NGHF achieves the largest reductions in WER. With sigmoid based models, the method can be seen to achieve a relative WERR of 8% with the DNN and 7% with the TDNN over the CE trained models. Compared to NG, the method achieves a relative WERR of 1%. Although this improvement was fairly small, it was found to be statistically significant (sign. test of the WERs at the episode level). With the ReLU models, NGHF achieves a relative WERR of 7% with the DNN model and a 2% with the TDNN over standard SGD. In comparison to HF, using the method results in a further WERR of 3% with the DNN and 1% with

the TDNN. Both of these gains again was found to statistically significant (at the episode level).

From Tables 7.3 and 7.2, it can be seen that ReLU based systems failed to achieve similar WERRs as sigmoid based systems from sequence training with either SGD or NG. During training, it was observed that improvements made on the MPE criterion failed to correlate with the model's ability to achieve lower WERs after first few epochs. This effect was particularly noticeable with the TDNN model. MBR training broadly speaking tries to concentrate probability mass: a sufficiently flexible model trained to convergence with MBR will assign a high probability to those hypotheses that have the smallest loss. In Sec. 6.7.2, it was discussed that ReLUs improve the flow of gradients and help accelerate the posterior distribution of states $\boldsymbol{\gamma}_t^r(i)$, associated with high local losses $\boldsymbol{L}(i)$, to become zero. With hyper-parameters such as the LM and acoustic model scale factor fixed, the sharp decrease in the DNN output entropy shown in Fig. 7.3, directly reflects that the distribution $\boldsymbol{\gamma}_t^r$ is becoming overly sharp for the ReLU models when training is performed with SGD or NG.

From Fig. 7.3, it can also be seen how scaling with the GN matrix helps regularise the entropy of the DNN frame posteriors. When compared to HF, the proposed NGHF approach is better in finding a balance between improving the MPE criterion and regularising the entropy changes of the DNN frame posteriors.

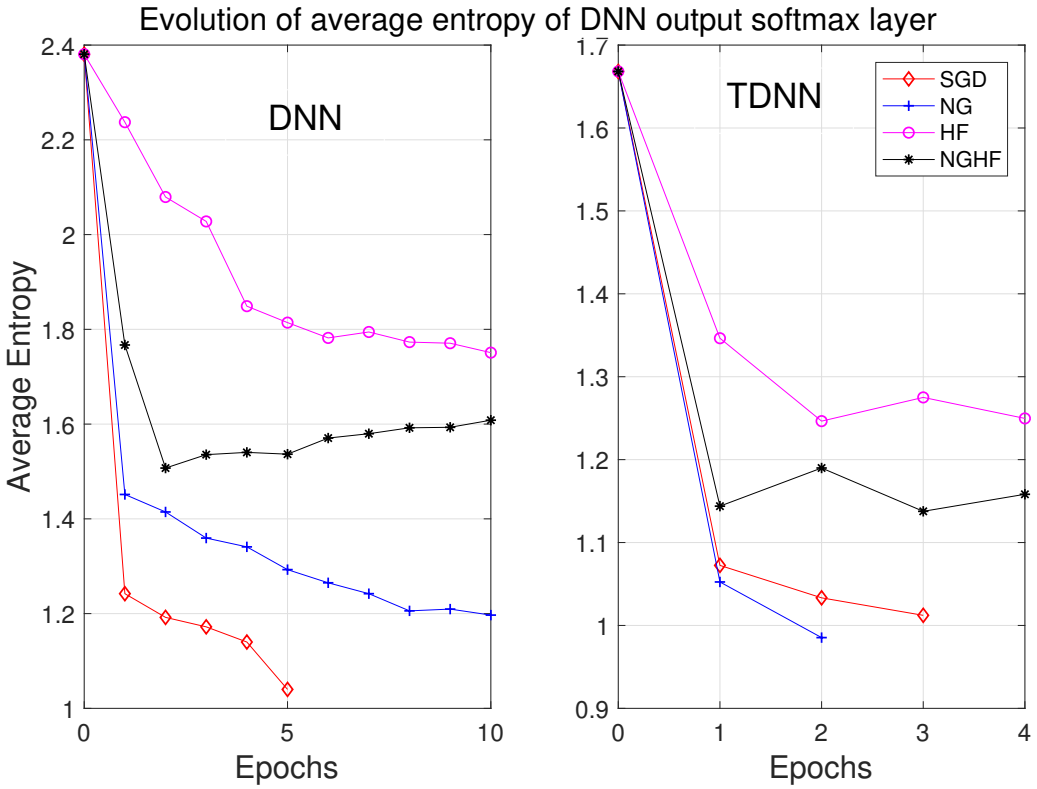


Fig. 7.3 Evolution of average entropy of DNN output activations during MPE training with ReLU based DNN models. Left graph is for DNNs and the right graph for TDNNs.

7.7 Experiments on Recurrent Neural Networks

Having investigated sequence training with feed forward architectures, the investigation will now be extended to improving lattice-based MPE training with standard recurrent neural networks. In this section, experiments on both the 50hr and 200hr MGB1 training set is presented. To make fair comparisons between the various optimisation approaches, models with both ReLU and sigmoid activations were trained.

7.7.1 Experimental Setup

The architecture used for the RNNs consisted of stacking two recurrent layers of width 1000 nodes followed by a feedforward layer composed of 1000 nodes. The output softmax layer had context-dependent sub-phone targets formed by conventional decision tree context-dependent state tying, and comprised of 4k/6k nodes for the 50h/200h training sets. The input to all the models was produced by splicing together 40 dimensional log-

Mel filter bank (FBK) features extended with their delta coefficients. Prior to sequence training, the model parameters were initialised using frame-level CE training and during training, the unrolling of the recurrent layers was performed from +5 to -15 time steps. To track the occurrence of overfitting due to training criterion mismatch, decoding was performed, at intermediate stages of sequence training, on the validation set using the same weak pruned biased LM used to create the initial MPE lattices. Finally, to evaluate the generalisation performance of the trained models, a 158k word vocabulary trigram LM was used to decode the validation and test set.

Training configuration for NGHF, NG & HF: As before, the recipe described in Sec. 5.4.1 was used: gradient batches corresponding to roughly 25 hours and 0.5 hrs of audio were sampled for each CG run. For the RNNs, running each CG run beyond 8 iterations was not found to be advantageous.

7.7.2 Difficulty in finding SGD baseline for ReLU RNN models

With the ReLU RNN models, finding improved WERs from sequence training with SGD was particularly difficult. The training of the models suffered from overfitting due to mismatch between training criterion and WER. To get stable WER reductions, it was found that using an extremely small learning rate in the region of 10^{-6} ensured that SGD traversed along paths where optimising the MPE criterion closely mirrored with reductions in the WER. Tables 7.4 and 7.5 compares two variants of SGD training for the 50hr ReLU RNN model: the first setup employs an initial learning rate that has been carefully tuned to maximise the MPE validation performance while the second setup employs an initial rate which is 100 times smaller. From comparing the two tables, it can be seen that the training setup that yields the best MPE generalisation performance on the validation fails to achieve WER improvements at intermediate stages of training. Figure 7.4 shows that initialising SGD with a larger learning rate speeds up the rate of the DNN frame posteriors becoming overly sharp.

Epochs	Avg.Phone acc. Training set	Avg.Phone acc. Validation set	WER with MPE LM Validation set
0	0.849	0.705	41.7
1	0.864	0.730	44.2
2	0.880	0.737	44.2
3	0.885	0.741	44.6
4	0.889	0.744	45.1

Table 7.4 shows the performance of sequence training on the 50hr ReLU RNN model with SGD with a learning rate that has been carefully tuned to improve the MPE validation performance. The table also shows the WERs on the validation set with the weak MPE LM

Epochs	Avg.Phone acc. Training set	Avg.Phone acc. Validation set	WER with MPE LM Validation set
0	0.849	0.705	41.7
1	0.851	0.708	41.2
2	0.854	0.711	41.1
3	0.856	0.712	41.0
4	0.857	0.713	41.0

Table 7.5 shows the performance of sequence training the 50hr ReLU RNN model with SGD using a learning of the order of 10^{-6}. The table also shows the WERs on the validation set with the weak MPE LM

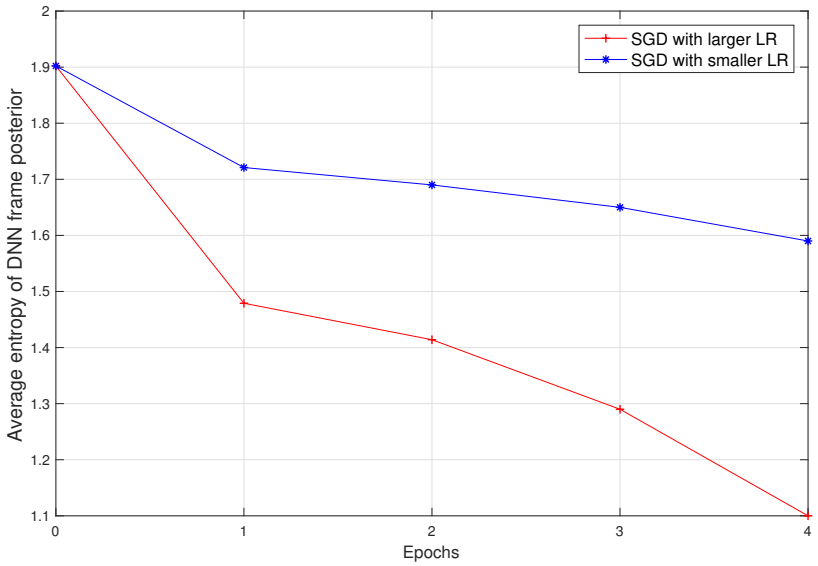


Fig. 7.4 Evolution of average entropy of ReLU RNN frame posteriors with different setups of SGD.

7.7.3 Experiments on 50hr MGB1 training set

Table 7.6 compares the performance of the various optimisation methods in training the 50hr ReLU RNN model. In comparison to SGD, the 3 batch style optimisation frameworks do better in achieving WER reductions on the validation set: HF achieves a WERR of 4% while both NG and NGHF achieves WERR of 3% over the CE trained model. For this particular model as highlighted in figure 7.4, the HF optimiser can be seen to achieve the best balance between improving the MPE criterion and regularising the entropy changes of the DNN frame posteriors.

method	#epochs	#updates	phone acc.		WER
			train	dev.sub	dev.sub
CE	N/A	N/A	0.849	0.705	41.7
SGD	4	1.31×10^5	0.857	0.713	41.0
HF	13	26	0.870	0.724	40.1
NG	15	30	0.884	0.734	40.4
NGHF	10	20	0.872	0.728	40.4

Table 7.6 shows the performance of different optimisers on the ReLU RNN. The WERs shown at the last column were computed using the weak pruned biased LM used in MPE training.

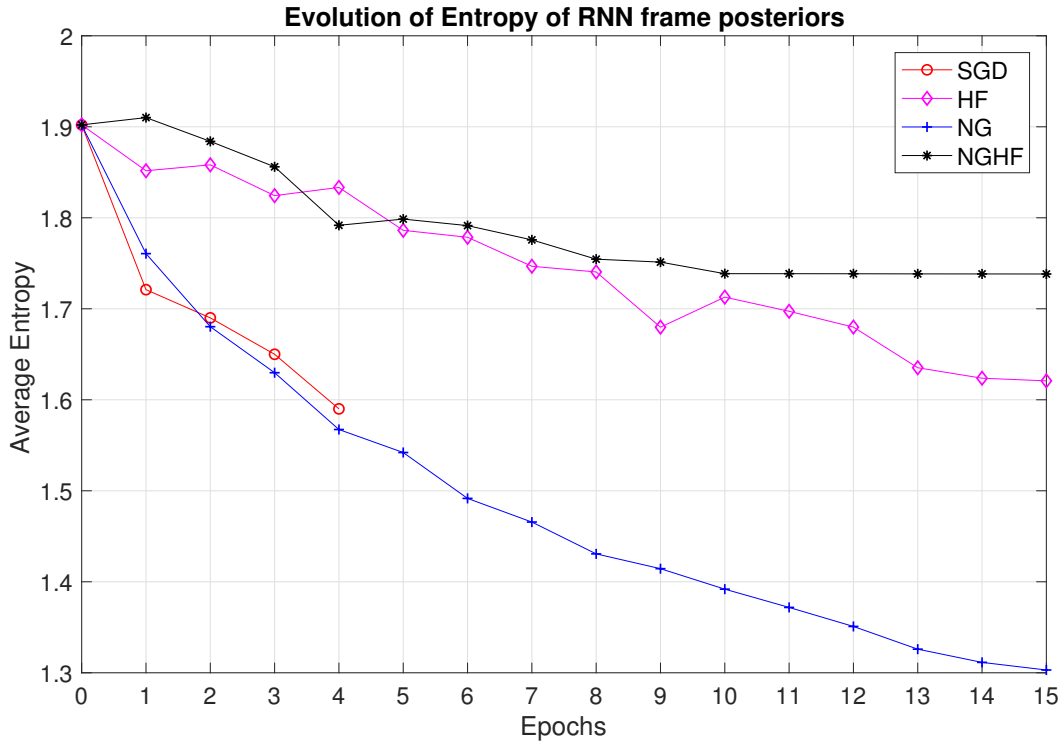


Fig. 7.5 Evolution of average entropy of ReLU RNN frame posteriors with different setups of SGD.

To investigate how these optimisers fair with different activation functions, equivalent systems using sigmoids were trained. Table 7.7 compares the performance of the various optimisers for a sigmoid RNN. Like in the ReLU case, the 3 batch style optimisation frameworks can be seen to be more effective than SGD in achieving better WERRs

with MPE training. SGD achieves a relative WERR of 3% while NG achieves a relative WERR of 4% over the CE trained model. In Sec 6.7.2, it was shown how sigmoids in comparison to ReLUs impedes the speed of learning. A glance in Table 7.8 shows that by adjusting the gradient with the local curvature of the KL-divergence, NG achieves the best balance in making the HMM-DNN sigmoid model discriminative without making the DNN frame posteriors overly sharp.

method	#epochs	#updates	phone acc.		WER with
			train	dev.sub	MPE LM
CE	N/A	N/A	0.83	0.682	43.9
SGD	4	1.31×10^5	0.876	0.718	42.6
NG	15	30	0.857	0.703	42.0
HF	15	30	0.852	0.697	42.4
NGHF	14	28	0.852	0.699	42.2

Table 7.7 shows the performance of different optimisers on the 50hr sigmoid-RNN. The WERs shown at the last column were computed using the weak pruned biased LM used in MPE training.

method	Average Entropy of DNN frame posteriors
CE	2.024
SGD	1.94
NG	1.833
HF	1.972
NGHF	1.927

Table 7.8 Average entropy of the DNN frame posteriors at the end of training of sigmoid DNN models using different optimisers.

Compared to the feed forward architectures, the RNN models investigated in this section have far fewer parameters. From Table 7.7 and Table 7.6, it was observed that the method of NGHF although being agnostic to the choice of activation function doesn't achieve the best WER reductions on the validation set. Figure 7.5 and Table 7.8 seems to suggest that for small models, the method appears to over regularise the decrease in the entropy of the DNN frame posteriors. To investigate this, the next section investigates using RNNs with a larger output layer (hence more parameters).

7.7.4 Experiments on 200hr MGB1 training set

Table 7.9 compares the performance of various optimisation methods on training the 200hr ReLU RNN model while Table 7.10 compares the optimisers on training an equivalent sigmoid model. As in the 50hr case, all 3 batch style optimisation frameworks can be seen to be more effective in achieving greater WERRs from sequence training that SGD. From Table 7.9 and Table 7.10, it can be seen that NGHF with the MPE LM achieves a relative WERR of 5% for both the ReLU and sigmoid model over CE. Over SGD, the method achieves a relative WERR of 4% for the ReLU model and 3% for the sigmoid model.

Comparing the average entropy plot given in Figure 7.5 with plot in Figure 7.6, it seems the method of NGHF becomes better in finding a balance between improving the MPE criterion and regularising the entropy changes of the DNN frame posteriors as we increase the model size. In future work, this will be investigated further.

method	#epochs	#updates	phone acc.		WER
			train	dev.sub	dev.sub
CE	N/A	N/A	0.825	0.736	38.7
SGD	4	4.63×10^5	0.851	0.748	38.5
HF	8	64	0.854	0.757	37.0
NG	5	40	0.862	0.768	37.3
NGHF	10	80	0.854	0.759	**37.0**

Table 7.9 shows the performance of different optimisers on the 200hr ReLU RNN. The WERs shown at the last column were computed using the weak pruned biased LM used in MPE training.

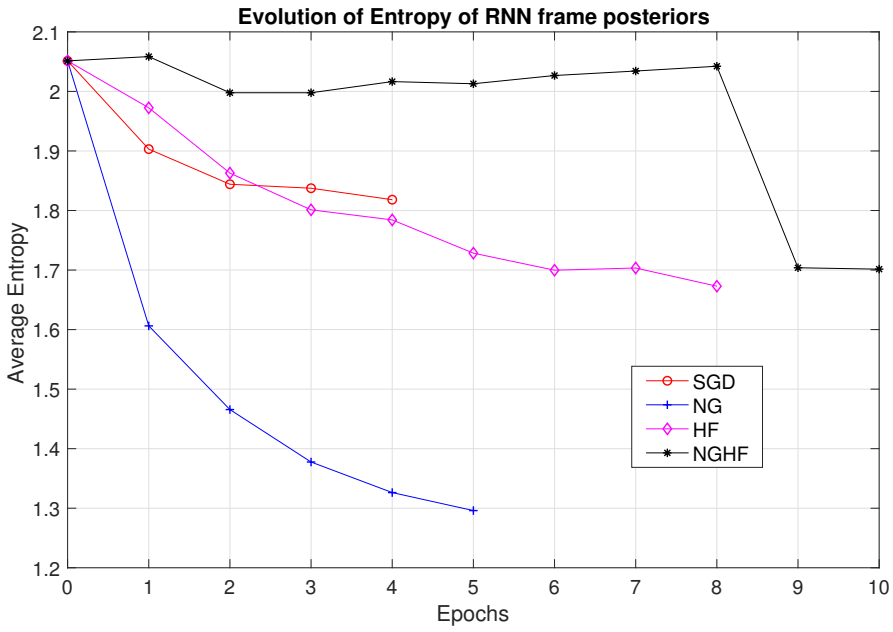


Fig. 7.6 Evolution of average entropy of ReLU RNN frame posteriors with different setups of SGD.

method	#epochs	#updates	phone acc.		WER with
			train	dev.sub	MPE LM
CE	N/A	N/A	0.80	0.725	39.9
SGD	4	4.63×10^5	0.858	0.749	39.1
NG	9	72	0.854	0.751	37.9
HF	10	80	0.846	0.743	38.3
NGHF	7	56	0.844	0.742	**38.1**

Table 7.10 shows the performance of different optimisers on the 200hr sigmoid-RNN. The WERs shown at the last column were computed using the weak pruned biased LM used in MPE training.

To investigate whether the relative reductions in WER are not constrained to the choice of LM, decoding on the validation set was performed using the 158k LM. Table 7.11 compares the results: the NGHF method achieves a relative WERR of 4% for the ReLU model and 6% for the sigmoid model over the CE trained model. Over SGD, the method achieves a relative WERR of 4% for the ReLU model and 3% for the sigmoid model.

Model	CE	MPE			
		SGD	NG	HF	NGHF
RNN-ReLU	29.7	29.6	29.2	28.9	**28.5**
RNN-sigmoid	31.0	30.4	29.2	29.6	**29.3**

Table 7.11 WERs on MGB1 dev.sub from sequence training RNNs with 158k trigram LM.

Finally, the generalisation performance of the trained models were estimated by performing Viterbi decoding on dev.sub2 using 158k LM. Results are shown in Table 7.12. It can be observed that NGHF achieves the best reductions in WER from sequence training for both the ReLU and sigmoid model. Over CE, the method can be seen to achieve a relative WERR of 5% for the sigmoid model and 4% for the ReLU model. Over SGD, the proposed method achieves a relative WERR of 3% for both the ReLU and sigmoid based model. These gains were found to be statistically significant (sign test of the WERs at the episode level).

Model	CE	MPE			
		SGD	NG	HF	NGHF
RNN-ReLU	31.1	30.8	30.7	29.8	**29.8**
RNN-sigmoid	32.2	31.6	30.6	30.7	**30.5**

Table 7.12 WERs on MGB1 dev.sub2 from sequence training RNNs with 158k trigram LM.

7.8 Summary

This chapter discussed the design considerations needed for developing an effective optimisation algorithm for the DNN function space $\mathcal{M}$, and presented a novel optimisation approach to probe this space. The chapter began with the development of the geometric structure of the function space captured by different parameter realisations of a DNN model, and discussed the property needed for optimisation methods to be well defined on this space. In Sec. 7.3, an alternative derivation of Taylor's theorem was presented using the concepts of manifolds, tangent vectors and directional derivatives from the perspective of Information Geometry. Using this alternative formulation of Taylor's theorem, it was shown how first and second order optimisation problems can be formulated as equivalent minimisation problems in the tangent space of X without any Riemannian structure.

By elucidating the tangent space with a Riemannian metric of the form of the Fisher Information, the method of NGHF was developed in Sec. 7.5. When framed within a Hessian Free style optimisation framework, the method enjoys the same benefits as HF in terms of stability and being data parallel but by utilising both the direction of steepest descent on a probabilistic manifold and local curvature information, the method was shown to be more effective in achieving consistent WER reductions from sequence training. The efficacy of the method was shown in Sec. 7.6 using experiments on 200hr MGB1 training set with different feed forward architectures. With sigmoid based models, the method achieved a relative WERR of 1% with the DNN and 3% with the TDNN over SGD. Whereas with the ReLU models, the method achieved a relative WERR of 7% with the DNN model and a 2% with the TDNN over SGD. Compared to HF, the method gave a further WERR of 3% with the DNN-ReLU model and 1% with the TDNN-ReLU model. These gains were found to be statistically significant (at the episode level). The performance of the various optimisers were also compared in training RNN models for the 50hr and 200hr MGB1 training set. On the 200hr training set, the method of NGHF was found to lead to the best reductions in WER from sequence training for both the ReLU and sigmoid models. This presents a promising argument that proposed method of NGHF is agnostic to the choice of activation functions and is quite effective in getting WERR from sequence training for large models.

Chapter 8

Conclusion and Future Work

8.1 Conclusion and Key Contributions

DNNs due to their deep and complex structure have the ability to effectively model the underlying non-linear data manifold. However, from experiments with sequence training of various DNN acoustic models, it was observed that although the structure of these models give rich modelling capability, it also creates complex dependencies between model parameters that makes learning difficult with gradient descent. This thesis focused on developing alternative optimisation techniques to train hybrid HMM-DNN models more effectively with discriminative sequence training. The work primarily investigated the design of batch styled optimisation frameworks that require far fewer updates than SGD and can be extended to a synchronous distributed setting. Furthermore, unlike ASGD, the frameworks proposed here are both mathematically sound and have the ability to produce identical results on repeated runs.

The major contributions of this thesis are summarised below:

1. An implementation of the Hessian Free optimisation framework was presented that is inherently data parallel in terms of gradient computation.

2. A procedure to stabilise CG training, that allows effective updates to be found in only a few iterations of the algorithm, was devised. It was also shown how the CG algorithm can be modified to yield updates that result in larger WER reductions for models with tied parameters.

3. The method of Natural Gradient was extended to the domain of Minimum Bayes Risk discriminative sequence training for hyrbid HMM-DNN models. In the DNN

literature, the method has been previously applied for non-sequence classification tasks within frameworks that employ a block diagonal approximation of the Fisher Information (FI) matrix. This work makes no such assumption about the structure of the matrix and uses a stochastic estimate of the FI matrix. When framed within an HF styled optimisation method, the method was shown to achieve both better and faster convergence (w.r.t. number of updates) than variants of HF. On sigmoid DNNs trained on the 200hr MGB1 dataset, the method was shown to achieve a relative WER reduction of 8% over CE on both the dev.sub and test dataset. Over SGD, the NG method on average was shown to provide approximately a 1% relative WER reduction (statistically significant at the episode level) in WER.

4. Instead of making a strict assumption on the structure of the FI matrix, this thesis derived an alternative dampened positive definite Fisher matrix, which when used with CG has the property that directions considered important by the empirical Fisher are first traversed during the initial stages of a CG run.

5. The issue of overfitting due to mismatch between training criterion and WER that primarily arises during sequence training of ReLU-DNN models was addressed. It was shown how this particular form of over-fitting can be alleviated by scaling the update directions with the Gauss Newton matrix.

6. The geometric structure of the function space captured by different parameter realisations of a DNN model was developed. The property needed for an optimisation method to be well defined on this space was also presented.

7. Introduced NGHF, a novel optimisation approach that combines the method of Natural Gradient with second order approaches. The method is derived from an alternative derivation of Taylor's theorem using principles from Information Geometry and manifold theory. When framed in an HF style optimisation framework, the method enjoys the same benefits as HF in terms of being inherently data parallel in gradient computation. But, by utilising both the direction of steepest descent on a probabilistic manifold and local curvature information, the proposed method is shown to converge to a better solution than any of the other methods within few updates. On models trained on the 200hr MGB1 training set, the efficacy of the method is shown to be agnostic to the choice of the network activation function. With standard DNNs, the method achieved a relative WERR of 7% on the test set for both ReLU and sigmoid models over CE. When applied to train sub-sampled

TDNNs, the relative WERRs corresponded to 7% for the sigmoid model and 4% with the ReLU model. Over SGD, the method was observed to consistently get larger WER reductions while making far lesser updates. The effectiveness of the approach was also found to extend to 200hr systems trained with standard RNNs. On the test set, the NGHF method was observed to lead to consistent lead to the greatest reductions in WER for ReLU and sigmoid models.

8.2 Future Work

A natural extension of this thesis is to extend the methods developed in this work to a synchronous distributed setting. As the proposed methods by design are inherently data parallel in terms of gradient computation, these computations can be parallelised across multiple machines. In collaboration with Dr.Chao Zhang[1], the methods proposed in this thesis will be extended within a distributed framework that has three distinct stages:

1. Parallel gradient computation: in Sec 5.5.2, it was shown how the gradient computations dominate the computation cost (measured in terms of elapsed time) in the implemented batch styled HF optimisation framework. In a distributed setting, this computation can be easily distributed, resulting in a significant decrease in the overall elapsed time of training. Figure 8.1 shows the operation of this process.

[1]Research Associate in Machine Intelligence Laboratory, Department of Engineering, University of Cambridge

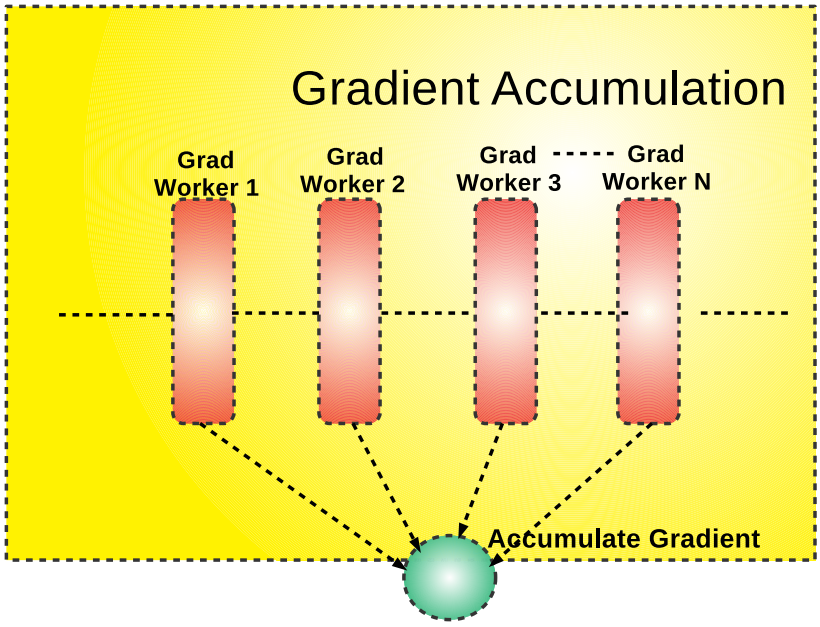


Fig. 8.1 Parellel gradient computation

2. Parallel CG runs: To reduce the cost associated with individual CG iterations, this thesis employs a sub-sampled approach where an approximation of the Gauss Newton or the FI matrix is made using only 0.5 hrs of sampled data. The motivation behinds this approach stems from the observations made by [155], where the author discovered that with inexact Newton methods, the estimate of the Hessian matrix need not be as accurate as the gradient to yield an effective update. If the CG mini-batch is chosen to be small, the cost of each CG iteration can be reduced significantly. But on the other hand, the size of this subset should be at least large enough for the information carried by the matrix vector products to be productive. In the distributed framework, to reduce the noise associated with the approximation of the various matrices, the individual CG runs will be parallelised across multiple machines. In this setting, each worker will start with the same accumulated gradient direction but will be operating on a different sampled mini-batch. Figure 8.2 illustrates this process. In addition to reducing the noise of the CG updates, having workers operating with different CG mini-batches has one other potential advantage. Recall, how the update direction proposed at each CG iteration corresponds to:

$$\Delta\boldsymbol{\theta}_{k+1} \leftarrow \Delta\boldsymbol{\theta}_k + a_k \boldsymbol{d}_k$$

where $\boldsymbol{d}_k$ represents the current conjugate direction. In Sec. 5.2.1, it was discussed how the algorithm initially focuses its search along the subspace spanned by the eigenvectors with the largest eigenvalues [131]. For different CG mini-batches, this subspace will be most likely different. This might allow the parallel CG framework to traverse along a larger subspace for the same number of CG iterations.

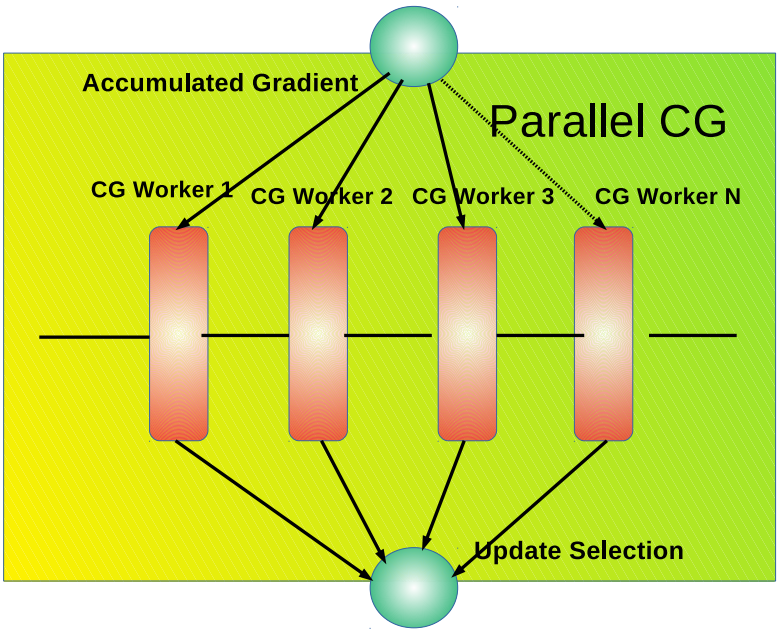


Fig. 8.2 Parallel CG runs

3. Model averaging: the updates produced by the different CG workers will be passed to a model averaging stage. Modelling averaging belongs to the class of approaches known as ensemble methods. In this particular form of ensemble learning, a number of classifiers are trained separately on different portions of the dataset. Since each classifier has been trained on a different portion of the dataset, it has learned a different 'aspects' of the data and will not make the same mistake as others. It is believed by combining the model parameters, a stronger classifier can be yielded which is less prone to overfitting. In [166, 172], it has been argued that averaging the parameter vectors from different workers can help reduce the variance associated with update directions and improve convergence to a better solution. Such an approach has been shown in [173] to particularly useful at the latter stages of training. Building on these works, the CG updates will be averaged before being applied to the global model. Such a framework has the flexibility that apart from decreasing the noise associated with the CG mini-batch, the updates can come

from different methods. For example some CG workers may be propagating an estimate of the NG direction while others can propagate an equivalent HF direction. Hence, this framework can provide an alternative way to combine the NG method with second order approaches.

Apart from extending the methods to a distributed setting, future work will involve applying the methods developed in this thesis to training different recurrent architectures such as LSTMs and bi-directional RNNs. It will be interesting to observe how optimisation methods, designed to exploit the underlying geometric structure of the DNN function space, improve the ability of these models to maintain information about important events across multiple time steps. Finally, in Appendix B.1, it is shown how the GN matrix for the MBR objective function is not guaranteed to be positive definite. Future work will investigate in constructing an appropriate divergence on the space of $\mathcal{M}$ such that the associated Riemannian metric encapsulated information of the MBR loss.

Appendix A

Elements of Differential Geometry

This section gives a brief introduction on the concept of manifolds, tangent vectors and directional derivatives from the perspective of differential geometry. The section begins with a formal description of a smooth manifold X and introduces the concept of a tangent vector associated with each point in X. By utilising the notion of a tangent space, the concept of directional derivatives is then defined.

Conceptually, a manifold corresponds to any geometric object embedded in $\mathbb{R}^k$ that is locally Euclidean i.e any local patch of the object is topologically equivalent to an open unit ball in a smaller dimensional Euclidean space. In $\mathbb{R}^3$, curves and surfaces are examples of an embedded manifold. When movement is constrained only along these objects, the degrees of freedom with which one can traverse is lower than dimensionality of the embedded 3 dimensional space.

A.1 Formal definition of a manifold

A topological manifold X of dimension D is a second countable Hausdorff topological space that is locally homeomorphic to $\mathbb{R}^D$; that is, for any point $\boldsymbol{\theta} \in X$, there exists an open neighbourhood U of $\boldsymbol{\theta}$ and a homeomorphism $g : U \to O \subset \mathbb{R}^D$, where O is open in $\mathbb{R}^D$. We call the homeomorphism $g : U \to O$ a chart, and the neighbourhood U a coordinate neighbourhood of $\boldsymbol{\theta}$.

To clarify for the reader, in topology the concept of a homeomorphism is equivalent to a bijective continuous map and the concept of Hausdorff means that for any two points $\hat{\boldsymbol{\theta}}, \bar{\boldsymbol{\theta}} \in X$, it is always possible to find disjoint open neighbourhoods that contain each point.

Let $r_i : \mathbb{R}^D \to \mathbb{R}$ denote the projection onto the ith coordinate. Given a chart $g : U \to O \subset \mathbb{R}^D$, let $\theta_i : r_i \circ g : U \to \mathbb{R}$. The functions θ_i are the local coordinates w.r.t the chart g on U. Having established a notion of what it means to be manifold, the next concept that will be introduced is the concept of a tangent vector. To formally define the concept of a tangent vector at any point $\boldsymbol{\theta} \in X$, it is first necessary to formalise the notion of *germs* associated with a given point in a manifold.

A.2 Germs

Let X be a manifold and $\boldsymbol{\theta} \in X$. Functions f, g defined on open subsets U, V respectively containing $\boldsymbol{\theta}$ are said to have the same germ at $\boldsymbol{\theta}$ if there exists a neighbourhood W of $\boldsymbol{\theta}$ contained in $U \cap V$ such that $f|_W \equiv g|_W$. The notion of germs therefore defines an equivalence relation on the space of functions defined on an open neighbourhood of $\boldsymbol{\theta}$ where $(U, f) \sim (V, g)$ if and only if there exists a neighbourhood W of $\boldsymbol{\theta}$ contained in $U \cap V$ such that $f|_W \equiv g|_W$. Let $C^\infty_{\boldsymbol{\theta}}$ be the set of all such equivalent function classes. Having defined the concept of class of germs associated with a given point $\boldsymbol{\theta} \in X$, the necessary machinery is now in place to define the concept of a tangent vector associated with the point $\boldsymbol{\theta} \in X$.

A.3 Tangent vector

A tangent vector $\boldsymbol{v}$ at given point $\boldsymbol{\theta} \in X$ is a linear derivation of $C^\infty_{\boldsymbol{\theta}}$, that is it a special form of a linear map $C^\infty_{\boldsymbol{\theta}} \to \mathbb{R}$ that satisfies the property $\boldsymbol{v}(f \cdot g) = f(\boldsymbol{\theta})\boldsymbol{v}(g) + \boldsymbol{v}(f)g(\boldsymbol{\theta})$ where $f \cdot g$ denotes the product of functions f and $g \in C^\infty_{\boldsymbol{\theta}}$. The tangent vectors form a real vector space in the obvious way; this space is denoted by $T_{\boldsymbol{\theta}}(X)$ and is called the tangent space to X at $\boldsymbol{\theta}$.

The concept of tangent vectors is necessary to do calculus on manifolds. Since manifolds are locally Euclidean, the usual notions of differentiation and integration make sense in any coordinate chart and can be carried over to manifold space.

Basis for Tangent space Let $(\theta_i, \theta_2, \cdots \theta_D)$ denote the standard coordinates of the parameter space X. Consider the operator $\frac{\partial}{\partial \theta_i}|_{\boldsymbol{\theta}}$ defined by $\dfrac{\partial}{\partial \theta_i}|_{\boldsymbol{\theta}}(f) = \dfrac{\partial f}{\partial \theta_i}(\boldsymbol{\theta})$. Then the set $\left\{\frac{\partial}{\partial \theta_i}|_{\boldsymbol{\theta}}\right\}_i$ can be shown to be linear derivations of $C^\infty_{\boldsymbol{\theta}}$ and hence members of

$T_{\boldsymbol{\theta}}(X)$. Furthermore, for $\forall\theta_j$, each operator in this set satisfies

$$\frac{\partial}{\partial\theta_i}|_{\boldsymbol{\theta}}(\theta_j) = \begin{cases} 1 & i \equiv j \\ 0 & \text{otherwise} \end{cases}$$

It can be shown that, by satisfying the above constraint, the members of $\{\frac{\partial}{\partial\theta_i}|_{\boldsymbol{\theta}}\}_i$ correspond to a basis of $T_{\boldsymbol{\theta}}(X)$. Therefore, w.r.t this basis if $\boldsymbol{v} \in T_{\boldsymbol{\theta}}(X)$ then $\boldsymbol{v} = \sum_i a^i \frac{\partial}{\partial\theta_i}|_{\boldsymbol{\theta}}$.

A.4 Concept of directional derivative

Let $\Phi : X \to N$ be a vector valued function that corresponds to a smooth map between two manifolds. The directional derivative of Φ at any point $\boldsymbol{\theta} \in X$ is the linear map

$$d\Phi|_{\boldsymbol{\theta}} : T_{\boldsymbol{\theta}}(X) \to T_{\Phi(\boldsymbol{\theta})}(N)$$

defined by:

$$\begin{aligned} \mathrm{d}\Phi|_{\boldsymbol{\theta}}(\sum_{\mathrm{i}} \mathrm{a}^{\mathrm{i}} \frac{\partial}{\partial\theta_{\mathrm{i}}}|_{\boldsymbol{\theta}}) &= \left(\frac{\partial}{\partial y_i}|_{\Phi(\boldsymbol{\theta})}, \frac{\partial}{\partial y_i}|_{\Phi(\boldsymbol{\theta})}, \cdots \frac{\partial}{\partial y_k}|_{\Phi(\boldsymbol{\theta})}\right) \begin{bmatrix} \frac{\partial\Phi_1}{\partial\theta_1}|_{\boldsymbol{\theta}} & \frac{\partial\Phi_1}{\partial\theta_2}|_{\boldsymbol{\theta}_2} \cdots \\ \frac{\partial\Phi_2}{\partial\theta_1}|_{\boldsymbol{\theta}} & \frac{\partial\Phi_2}{\partial\theta_2}|_{\boldsymbol{\theta}_2} \cdots \\ \vdots & \\ \frac{\partial\Phi_k}{\partial\theta_1}|_{\boldsymbol{\theta}} & \frac{\partial\Phi_k}{\partial\theta_2}|_{\boldsymbol{\theta}_2} \cdots \end{bmatrix} \begin{bmatrix} a^1 \\ a^2 \\ \vdots \\ a^D \end{bmatrix} \\ &= \left(\frac{\partial}{\partial y_i}|_{\Phi(\boldsymbol{\theta})}, \frac{\partial}{\partial y_i}|_{\Phi(\boldsymbol{\theta})}, \cdots \frac{\partial}{\partial y_k}|_{\Phi(\boldsymbol{\theta})}\right) J_\Phi \begin{bmatrix} a^1 \\ a^2 \\ \vdots \\ a^D \end{bmatrix} \end{aligned} \tag{A.1}$$

where $\{\frac{\partial}{\partial y_j}|_{\Phi(\boldsymbol{\theta})}\}_j$ denotes the basis of $T_{\Phi(\boldsymbol{\theta})}(N)$ and J_Φ is the Jacobian of Φ at the given point $\boldsymbol{\theta}$. Each coordinate of $\frac{\partial}{\partial y_j}|_{\Phi(\boldsymbol{\theta})}$ then corresponds to the directional derivative of Φ_j w.r.t $\boldsymbol{\theta}$. Under the above construction, the chain rule can be seen to be tautologous: if $\Psi : N \to P$ is a smooth map of smooth manifolds such that $\Psi \circ \Phi : M \to P$ is defined

then

$$\mathrm{d}(\Psi \circ \Phi)|_{\boldsymbol{\theta}}(\sum_{\mathrm{i}} \mathrm{a}^{\mathrm{i}} \frac{\partial}{\partial \theta_{\mathrm{i}}}|_{\boldsymbol{\theta}}) = (\frac{\partial}{\partial s_i}|_{\Psi(\Phi(\boldsymbol{\theta}))}, \frac{\partial}{\partial s_i}|_{\Psi(\Phi(\boldsymbol{\theta}))}, \cdots \frac{\partial}{\partial s_k}) J_\Psi J_\Phi \begin{bmatrix} a^1 \\ a^2 \\ \vdots \\ a^D \end{bmatrix} \tag{A.2}$$

Appendix B

B.1 Issues with the Gauss Newton

In the context of DNN models, the GN matrix takes the particular form :

$$\frac{1}{R}\sum_{r}^{R}\sum_{t}\left(\boldsymbol{J}_{\boldsymbol{\theta},t}^{r}\right)^{\mathrm{T}}\left(\nabla^{2}\boldsymbol{L}_{\mathrm{obj,t}}^{r}\right)^{\mathrm{T}}\boldsymbol{J}_{\boldsymbol{\theta},t}^{r}$$

where

- $\nabla^2\boldsymbol{L}^r_{\mathrm{obj,t}}$ is the Hessian of the objective function with respect to DNN output activations at time t for the r-th utterance.
- $\boldsymbol{J}^r_{\boldsymbol{\theta},t}$ is the Jacobian of the output activations of the network w.r.t the parameters at time t for the r-th utterance.

For MBR loss functions, $\nabla^2\boldsymbol{L}^r_{\mathrm{obj,t}}$ takes the following form:

$$\nabla^{2}\boldsymbol{L}_{\mathrm{obj,t}}^{r}=\frac{\kappa^{2}}{R}\left[\mathrm{diag}(\boldsymbol{\gamma}_{t}^{r})-\hat{\boldsymbol{\gamma}}_{t}^{r}(\boldsymbol{\gamma}_{t}^{r})^{T}\right] \tag{B.1}$$

with $\hat{\boldsymbol{\gamma}}^r_t = \boldsymbol{\gamma}^r_t \odot \boldsymbol{L}(\boldsymbol{s})$. Here,

- $\boldsymbol{L}(\boldsymbol{s})$ is a vector whose ith entry corresponds to the difference between $\check{L}(i)$, the posterior weighted sum of the local losses computed over all the lattice paths that pass through arcs containing the state i, and c^L_{avg}, the posterior weighted sum of the loss computed over all the lattice paths.
- the normalisation coefficient R corresponds to the total number of utterances.
- $\boldsymbol{\gamma}^r_t$ is a probability vector whose entries correspond to the posterior probability associated with the states(DNN output nodes) at time t within the consolidated lattice of the rth utterance.

- κ as the acoustic scaling factor

If we assume $L(\boldsymbol{s})$ to be a vector of 1s, then matrix $\left[\text{diag}(\boldsymbol{\gamma}_t^r) - \boldsymbol{\gamma}_t^r(\boldsymbol{\gamma}_t^r)^T\right]$ can be shown to be positive semi definite.

Sketch of the proof:

$$\begin{aligned}
\boldsymbol{v}^T[\text{diag}\boldsymbol{\gamma}_t^r - \boldsymbol{\gamma}_t^r(\boldsymbol{\gamma}_t^r)^T]\boldsymbol{v} &= \sum_i^k \boldsymbol{\gamma}_{t,i}^r v_i^2 - \sum_i^k (\boldsymbol{\gamma}_{t,i}^r v_i)^2 \\
&= \sum_i^k \boldsymbol{\gamma}_{t,i}^r v_i^2 - 2(\sum_i^k \boldsymbol{\gamma}_{t,i}^r v_i)(\sum_j^k \boldsymbol{\gamma}_{t,j}^r v_j) + \sum_j^k (\boldsymbol{\gamma}_{t,j}^r v_j)^2 \\
&= \sum_i^k \boldsymbol{\gamma}_{t,i}^r v_i^2 - 2(\sum_i^k \boldsymbol{\gamma}_{t,i}^r v_i)(\sum_j^k \boldsymbol{\gamma}_{t,j}^r v_j) + \sum_i^k \boldsymbol{\gamma}_{t,i}^r \sum_j^k (\boldsymbol{\gamma}_{t,j}^r v_j)^2 \\
&= \sum_i^k \boldsymbol{\gamma}_{t,i}^r (v_i - \sum_j^k \boldsymbol{\gamma}_{t,j}^r v_j)^2 \geq 0
\end{aligned}$$

However, in the context when $L(\boldsymbol{s})$ is not constrained to be a vector of 1s, $\nabla^2 \boldsymbol{L}_{\text{obj,t}}^r$ is **no** guaranteed to a positive semi definite matrix. Thus w.r.t MBR criteria, the GN matrix is not guaranteed to be positive semi-definite.

B.2 Gradient of ML objective function

For HMM-DNN models, the ML objective function when framed in the form of a cost function corresponds to:

$$F_{\text{ML}}(\boldsymbol{\theta}) = -\frac{1}{R}\sum_r^R \log\ \text{p}_{\boldsymbol{\theta}}(\mathcal{H}^{\text{r}}, \mathcal{O}^{\text{r}}|\mathcal{M}) \tag{B.2}$$

where $\mathcal{M}$ is space of the all probability densities captured by different DNN parameter realisations. Let $b_j(\boldsymbol{o}_t)$ denote $p_\theta(\boldsymbol{o}_t|\bar{\mathcal{O}}, \boldsymbol{\phi}_t(j) = 1)$ and a_{ij} denote the transition probability $P_\theta(\boldsymbol{\phi}_t(i) = 1|\boldsymbol{\phi}_{t-1}(j) = 1)$. Then,

$$\frac{\partial}{\partial b_j(\boldsymbol{o}_t^r)} \log p_\theta(\mathcal{O}^r, \mathcal{H}^r|\mathcal{M}) = \frac{1}{p_\theta(\mathcal{O}^r, \mathcal{H}^r|\mathcal{M})} \frac{\partial}{\partial b_j(\boldsymbol{o}_t^r)} p_\theta(\mathcal{O}^r, \mathcal{H}^r|\mathcal{M})$$

Using 2.16

$$= \frac{1}{p_\theta(\boldsymbol{O}^r|M_{Hr})^\kappa p(Hr)} \frac{\partial}{\partial b_j(\boldsymbol{o}_t^r)} \sum_{i \in \mathcal{M}} \alpha_t(i)\beta_t(i)$$

$$= \frac{1}{p_\theta(\boldsymbol{O}^r|M_{Hr})^\kappa p(Hr)} \frac{\partial}{\partial b_j(\boldsymbol{o}_t^r)} \sum_{i \in \mathcal{M}_\mathcal{H}} \Big(\sum_{k \in \mathcal{M}_\mathcal{H}} \alpha_{t-1}(k)\ a_{ki} \Big) b_i(\boldsymbol{o}_t^r)\beta_t(i)$$

$$= \frac{1}{p_\theta(\boldsymbol{O}^r|M_{Hr})^\kappa p(Hr)} \frac{\alpha_t(j)\beta_t(j)}{b_j(\boldsymbol{o}_t^r)}$$

$$= \frac{1}{b_j(\boldsymbol{o}_t^r)} \gamma_{t,j}^r$$

Thus,

$$\nabla_{\log p(\mathrm{o_t^r}|\phi_\mathrm{t})} \log p_\theta(\mathcal{O}^r, \mathcal{H}^r|\mathcal{M}) = \boldsymbol{\gamma}_t^r \tag{B.3}$$

where $\boldsymbol{\gamma}_t^r$ encodes the posterior probabilities of individual states in the composite HMM model constructed from the correct hypothesis. Using the above derivation,

$$\begin{aligned}
\frac{\partial F_{ML}}{\partial a_{t,k}^r} &= -\sum_p^K \frac{\partial F_{ML}}{\partial z_{t,p}^r} \frac{\partial z_{t,p}^r}{\partial a_{t,k}^r} \\
&= \text{Using eq (2.11)} \\
&= -\sum_p^K \frac{\partial F_{ML}}{\partial \log\, b_p(o_t^r)} \frac{\partial \log\, b_p(o_t^r)}{\partial \log\, z_{t,p}^r} \frac{\partial \log\, z_{t,p}^r}{\partial z_{t,p}^r} \frac{\partial z_{t,p}^r}{\partial a_{t,k}^r} \\
&= -\sum_p^K \gamma_{t,p}^r \frac{1}{z_{t,p}^r} \frac{\partial z_{t,p}^r}{\partial a_{t,k}^r} \\
&= \left[\sum_p^K \gamma_{t,p}^r \frac{1}{z_{t,p}^r} \Big(z_{t,p}^r z_{t,k}^r \Big) \right] - \gamma_{t,k}^r \frac{1}{z_{t,k}^r} z_{t,k}^r \\
&= z_{t,k}^r - \gamma_{t,k}^r
\end{aligned} \tag{B.4}$$

In vector notation, the ML gradient w.r.t DNN output activations is then the vector:

$$\nabla L_{\mathrm{ML,t}}^r = \boldsymbol{z}_t^r - \boldsymbol{\gamma}_t^r \tag{B.5}$$

When compared to MMI training, $\boldsymbol{\gamma}_t^r$ here is equivalent to $\boldsymbol{\gamma}_t^{r,Num}$,the posterior probabilities of individual states in the numerator lattice.

Appendix C

Dataset Descriptions

C.1 Debugging dataset

For this work, the debugging dataset used is the WSJ0 [174] SI84 training set from the Wall Street Journal. This particular training set consists of 7185 utterances recorded from 84 speakers and roughly encapsulates 15 hours of speech. Details of data preparation can be found in [116].

C.2 Details of 50hr and 200hr MGB1 dataset

The DNN training techniques investigated in this work were evaluated on data from the 2015 Multi-Genre Broadcast ASRU challenge task (MGB1) [21]. The full audio data consists of seven weeks of BBC television programmes with a raw total duration of 1,600 hours, and was sampled at a 16kHz sampling rate. The data covers a full range of genres, e.g. news, comedy, drama, sports, quiz shows, documentaries etc. The audio was pre-processed using a lightly supervised decoding process. Systems were trained using a 200hr training set sampled randomly from 2,180 shows for which the subtitles and the lightly supervised output had a phone matched error rate of less than 20% [175]. This sampled training set consists of 115,932 utterances that were automatically clustered into 10,930 speaker clusters [176]. For preliminary experiments, a 50 hour subset evenly sampled from this set was used. This consists of 32,771 utterances and 3,272 speaker clusters.

Choice of validation and test set: The official dev.sub set was employed as a validation set and consists of 5.5 hours of audio data from across 12 episodes. A separate evaluation set was used to estimate the generalisation performance of each candidate

model. For our experiments we took the remaining 35 shows from the MGB1 dev.full and denote this set as dev.sub2 for evaluation purposes. The segmentations used for all experiments were taken from the reference transcriptions. Further details of the data preparation are in [22].

References

[1] A. Haider and P. C. Woodland, "Sequence training of DNN Acoustic Models with Natural Gradient," in *Proc. ASRU*, 2017.

[2] A. Haider and P. C. Woodland, "Combining Natural Gradient with Hessian Free Methods for Sequence Training," in *Proc. InterSpeech*, 2018.

[3] G. Saon, T. Sercu, S. Rennie, and H.-K. Kuo, "The IBM 2016 English Conversational Telephone Speech Recognition System," *arXiv 1604.08242*, 2016.

[4] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig, "The Microsoft 2016 Conversational Speech Recognition System," *arXiv 1609.03528*, 2016.

[5] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and K. Brian, "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[6] G. Desjardins, K. Simonyan, R. Pascanu, and K. Kavukcuoglu, "Natural Neural Networks," in *Proc. NIPS*, 2015.

[7] R. Grosse and R. Salakhudinov, "Scaling up Natural Gradient by Sparsely Factorizing the Inverse Fisher Matrix," in *Proc. ICML*, 2015.

[8] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of Training Recurrent Neural Networks," in *Proc. ICML*, 2013.

[9] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very Deep Networks," in *Proc. NIPS*, 2015.

[10] L. Deng, G. Hinton, and B. Kingsbury, "New Types of Deep Neural Network Learning for Speech Recognition and Related Applications: An Overview," in *Proc. ICASSP*, 2013.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. CVPR*, 2016.

[12] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[13] F. L. Kreyssig, C. Zhang, and P. C. Woodland, "Improved TDNNs using Deep Kernels and Frequency Dependent Grid-RNNs," in *Proc. ICASSP*, 2018.

[14] S. Becker and Y. Le Cun, "Improving the Convergence of Back-Propagation learning with Second Order methods," in *Proceedings of the 1988 connectionist models summer school*, San Matteo, CA: Morgan Kaufmann, 1988.

[15] L. Bottou and Y. Cun, "Large Scale Online Learning," in *Proc. NIPS*, 2004.

[16] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for Large Scale Machine Learning," *arXiv preprint arXiv:1606.04838*, 2016.

[17] C. T. Chu, S. K. Kim, Y. A. Lin, Y. Yu, G. Bradski, K. Olukotun, and A. Y. Ng, "Map-Reduce for Machine Learning on Multicore," in *Proc. NIPS*, 2007.

[18] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, *et al.*, "Large scale Distributed Deep Networks," in *Proc. NIPS*, 2012.

[19] A. Srinivasan, A. Jain, and P. Barekatain, "An Analysis of the Delayed Gradients Problem in Asynchronous SGD," in *Proc. ICLR*, 2018.

[20] B. Kingsbury, T. Sainath, and H. Soltau, "Scalable Minimum Bayes Risk training of Deep Neural Networks Acoustic Models using Distributed Hessian-Free Optimization," in *Proc. InterSpeech*, 2012.

[21] P. Bell, M. J. Gales, T. Hain, J. Kilgour, P. Lanchantin, X. Liu, A. McParland, S. Renals, O. Saz, M. Wester, and P. C. Woodland, "The MGB challenge: Evaluating Multi-Genre Broadcast Media recognition," in *Proc. ASRU*, 2015.

[22] P. C. Woodland, X. Liu, Y. Qian, C. Zhang, M. J. F. Gales, P. Karanasou, P. Lanchantin, and L. Wang, "Cambridge University Transcription Systems for the Multi-Genre Broadcast Challenge," in *Proc. ASRU*, 2015.

[23] S. Amari, *Information Geometry and its Applications*. Springer, 2016.

[24] W. Byrne, "Information Geometry and Maximum Likelihood Criteria," in *Proc. ISS*, 1996.

[25] S. Amari, "Neural Learning in Structured Parameter Spaces - Natural Riemannian Gradient," in *Proc. NIPS*, 1997.

[26] S. Davis and P. Mermelstein, "Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences," in *Readings in Speech Recognition*, pp. 65–74, Elsevier, 1990.

[27] K. Tokuda, T. Kobayashi, T. Masuko, and S. Imai, "Mel Generalized Cepstral Analysis - a Unified Approach to Speech Spectral Estimation," in *Proc. Third International Conference on Spoken Language Processing*, 1994.

[28] H. Hermansky, "Perceptual Linear Predictive (PLP) Analysis of Speech," *The Journal of the Acoustical Society of America*, vol. 87, no. 4, pp. 1738–1752, 1990.

[29] Z. Tüske, P. Golik, R. Schlüter, and H. Ney, "Acoustic Modeling with Deep Neural Networks using Raw Time Signal for LVCSR," in *Proc. Interspeech*, 2014.

[30] T. N. Sainath, R. J. Weiss, A. Senior, K. W. Wilson, and O. Vinyals, "Learning the speech front-end with raw waveform CLDNNs," in *Proc. Interspeech*, 2015.

[31] L. R. Rabiner and J. B. Hwang, *Fundamentals of Speech Recognition*, vol. 14. Prentice Hall Signal Processing Series, 1993.

[32] E. Zwicker, "Subdivision of the Audible Frequency Range into Critical bands," *The Journal of the Acoustical Society of America*, vol. 33, no. 2, pp. 248–248, 1961.

[33] P. C. Woodland, M. J. F. Gales, D. Pye, and S. J. Young, "The Development of the 1996 HTK Broadcast News Transcription System," in *Proc. DARPA Speech Recognition Workshop*, Morgan Kaufmann Pub, 1997.

[34] C. Xie, *Scalable Recurrent Neural Network Language Models for Speech Recognition*. PhD thesis, University of Cambridge, 2017.

[35] M. J. F. Gales and S. J. Young, "The Application of Hidden Markov Models in Speech Recognition," *Foundations and Trends in Signal Processing*, vol. 1, no. 3, pp. 195–304, 2008.

[36] E. Selkirk, *Phonology and syntax: the Relationship between Sound and Structure*. MIT Press Cambridge, 1986.

[37] R. Schwartz, Y. Chow, O. Kimball, S. Roucos, M. Krasner, and J. Makhoul, "Context-Dependent Modeling for Acoustic Phonetic Recognition of Continuous Speech," in *Proc. ICASSP*, 1985.

[38] S. J. Young, J. J. Odell, and P. C. Woodland, "Tree based State Tying for High Accuracy Acoustic Modelling," in *Proc. HLT*, 1994.

[39] S. J. Young and P. C. Woodland, "The Use of State Tying in Continuous Speech Recognition," in *Proc. Third European Conference on Speech Communication and Technology*, 1993.

[40] L. R. Bahl, D. P, P. Gopalakrishnan, D. Nahamoo, and M. Picheny, "Decision Trees for Phonological Rules in Continuous Speech," in *Proc. ICASSP*, 1991.

[41] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B (methodological)*, pp. 1–38, 1977.

[42] M. Gales, "Semi-tied Covariance Matrices for Hidden Markov Models," *IEEE Transactions on Speech and Audio Processing*, vol. 7, pp. 272–281, 1999.

[43] P. Baldi and K. Hornik, "Neural Networks and Principal Component Analysis: Learning from examples without local minima," *Neural networks*, vol. 2, no. 1, pp. 53–58, 1989.

[44] H. Bourlard and C. J. Wellekens, "Links between Markov models and Multilayer Perceptrons," in *Proc. NIPS*, 1989.

[45] S. Renals, N. Morgan, M. Cohen, and H. Franco, "Connectionist Probability Estimation in the DECIPHER Speech Recognition System," in *Proc. ICASSP*, 1992.

[46] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, *et al.*, "The Kaldi speech recognition toolkit," in *Proc. ASRU*, 2011.

[47] S. J. Young, G. Evermann, M. J. F. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. J. Odell, D. Ollason, D. Povey, A. Ragni, V. Valtchev, P. C. Woodland, and C. Zhang, *The HTK Book (for HTK version 3.5)*. Cambridge, UK: Cambridge University Engineering Department, 2015.

[48] A. Graves and N. Jaitly, "Towards End-to-End Speech Recognition with Recurrent Neural Networks," in *Proc. ICML*, 2014.

[49] A. Graves, A. R. Mohamed, and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," in *Proc. ICASSP*, 2013.

[50] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, Attend and Spell: A Neural Network for Large Vocabulary Conversational Speech Recognition," in *Proc. ICASSP*, 2016.

[51] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, *et al.*, "Deep Speech 2: End-to-End Speech Recognition in English and Mandarin," in *Proc. ICML*, 2016.

[52] K. Rao, H. Sak, and R. Prabhavalkar, "Exploring Architectures, Data and Units for Streaming End-to-End Speech Recognition with RNN-Transducer," in *Proc. ASRU*, 2017.

[53] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks," in *Proc. ICML*, 2006.

[54] R. Prabhavalkar, K. Rao, T. N. Sainath, B. Li, L. Johnson, and N. Jaitly, "A Comparison of Sequence-to-Sequence Models for Speech Recognition," in *Proc. Interspeech*, 2017.

[55] L. E. Baum and J. A. Eagon, "An Inequality with Applications to Statistical Estimation for Probabilistic functions of Markov Processes and to a Model for ecology," *Bulletin of the American Mathematical Society*, vol. 73, no. 3, pp. 360–363, 1967.

[56] A. W. Van der Vaart, *Asymptotic Statistics*, vol. 3. Cambridge University Press, 1998.

[57] M. Gibson, *Minimum Bayes Risk Acoustic Model Estimation and Adaptation*. PhD thesis, University of Sheffield, 2008.

[58] Y. Normandin, *Hidden Markov Models, Maximum Mutual Information Estimation, and the Speech Recognition problem*. PhD thesis, McGill University, 1991.

[59] L. Bahl, P. Brown, P. De Souza, and R. Mercer, "Maximum Mutual Information Estimation of Hidden Markov Model parameters for Speech Recognition," in *Proc. ICASSP*, 1986.

[60] P. C. Woodland and D. Povey, "Large scale Discriminative training of Hidden Markov models for speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 25–47, 2002.

[61] J. Kaiser, B. Horvat, and Z. Kacic, "A Novel Loss function for the Overall Risk Criterion based Discriminative Training of HMM models," in *Proc. ICSLP*, 2000.

[62] V. Valtchev, J. Odell, P. C. Woodland, and S. J. Young, "Lattice-based Discriminative training for Large Vocabulary Speech Recognition," in *Proc. ICASSP*, vol. 2, 1996.

[63] V. Valtchev, J. Odell, P. C. Woodland, and S. J. Young, "MMIE training of Large Vocabulary Recognition Systems," *Speech Communication*, vol. 22, no. 4, pp. 303–314, 1997.

[64] M. Gibson and T. Hain, "Hypothesis Spaces for Minimum Bayes Risk training in Large Vocabulary Speech Recognition," in *Proc. InterSpeech*, 2006.

[65] L. Bahl, R. Bakis, F. Jelinek, and R. Mercer, "Language-model/Acoustic-channel-model Balance Mechanism," *IBM Technical Disclosure Bulletin*, vol. 23, pp. 3464–3465, 1980.

[66] D. Povey, *Discriminative training for Large Vocabulary Speech Recognition*. PhD thesis, University of Cambridge, 2005.

[67] D. Povey, V. Peddinti, D. Galvez, P. Ghahremani, V. Manohar, X. Na, Y. Wang, and S. Khudanpur, "Purely Sequence-Trained Neural Networks for ASR Based on Lattice-Free MMI," in *Proc. Interspeech*, 2016.

[68] R. Prabhavalkar, T. N. Sainath, Y. Wu, P. Nguyen, Z. Chen, C.-C. Chiu, and A. Kannan, "Minimum Word Error Rate Training for Attention-based Sequence-to-Sequence Models," *arXiv preprint arXiv:1712.01818*, 2017.

[69] M. Shannon, "Optimizing expected Word Error Rate via Sampling for Speech Recognition," *Proc. InterSpeech*, 2017.

[70] D. Povey and P. C. Woodland, "Minimum Phone Error and I-smoothing for improved Discriminative Training," in *Proc. Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 1, 2002.

[71] J. Zheng and A. Stolcke, "Improved Discriminative Training using Phone Lattices," in *Proc. Interspeech*, 2005.

[72] J. Baker, "The DRAGON system – An overview," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 23, pp. 24–29, 1975.

[73] M. Mohri, F. Pereira, and M. Riley, "Weighted Finite State Transducers in Speech Recognition," *Computer Speech and Language*, vol. 16, pp. 69–88, 2002.

[74] J. J. Odell, *The Use of Context in Large Vocabulary Speech Recognition*. PhD thesis, University of Cambridge, 1995.

[75] K. Demuynck, J. Duchateau, D. Van Compernolle, and P. Wambacq, "An Efficient Search Space Representation for Large Vocabulary Continuous Speech Recognition," *Speech Communication*, vol. 30, pp. 37–53, 2000.

[76] J. J. Odell, "Network and Language Models for use in a Speech Recognition System." US Patent 6668243 B1, 1999.

[77] V. Goel and W. J. Byrne, "Minimum Bayes Risk Automatic Speech Recognition," *Computer Speech & Language*, vol. 14, no. 2, pp. 115–135, 2000.

[78] A. Viterbi, "Error bounds for Convolutional Codes and an Asymptotically Optimum Decoding algorithm," *IEEE transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[79] S. Young, N. Russel, and J. Thornton, "Token passing: A Simple Conceptual Model for Connected Speech Recognition Systems.," tech. rep., CUED/FINFENG/TR38, University of Cambridge, 1989.

[80] V. Doumpiotis and W. Byrne, "Pinched lattice Minimum Bayes Risk Discriminative Training for Large Vocabulary Continuous Speech Recognition," in *Proc. InterSpeech*, 2004.

[81] L. Mangu, E. Brill, and A. Stolcke, "Finding Consensus in Speech Recognition: Word Error Minimization and other applications of Confusion Networks," *Computer Speech & Language*, vol. 14, no. 4, pp. 373–400, 2000.

[82] G. Evermann and P. C. Woodland, "Large Vocabulary Decoding and Confidence Estimation Using Word Posterior Probabilities," in *Proc. ICASSP*, 2000.

[83] H. Xu, D. Povey, L. Mangu, and J. Zhu, "Minimum Bayes Risk Decoding and System Combination based on a Recursion for Edit Distance," *Computer Speech & Language*, vol. 25, no. 4, pp. 802–828, 2011.

[84] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice-Hall, 1998.

[85] C. Bishop, "Neural Networks for Pattern Recognition," *Clarendon Press*, 1995.

[86] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[87] A. Pinkus, "Approximation theory of the MLP model in Neural Networks," *Acta numerica*, vol. 8, pp. 143–195, 1999.

[88] G. Cybenko, "Approximation by Superpositions of a Sigmoidal Function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[89] M. Minsky and S. Papert, *Perceptrons*. MIT Press, 1969.

[90] G. Tesauro, "Practical issues in Temporal Difference Learning," in *Proc. NIPS*, 1992.

[91] D. Ellis and N. Morgan, "Size Matters: An Empirical Study of Neural Network Training for Large Vocabulary Continuous Speech Recognition," in *Proc. ICASSP*, 1999.

[92] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[93] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy Layer-wise Training of Deep Networks," in *Proc. NIPS*, 2007.

[94] G. Hinton, S. Osindero, and Y. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural Computation*, vol. 18, pp. 1527–1554, 2006.

[95] G. Hinton, "A Practical Guide to Training Restricted Boltzmann Machines," tech. rep., University of Toronto, 2010.

[96] B. Kingsbury, "Lattice-based Optimization of Sequence Classification Criteria for Neural-Network Acoustic Modeling," in *Proc. ICASSP*, 2009.

[97] A. Waibel, "Modular Construction of Time-Delay Neural Networks for Speech Recognition," *Neural computation*, vol. 1, no. 1, pp. 39–46, 1989.

[98] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme Recognition using Time-Delay Neural Networks," in *Readings in Speech Recognition*, pp. 393–404, Elsevier, 1990.

[99] Y. LeCun, "Generalization and Network Design Strategies," tech. rep., University of Toronto, 1989.

[100] V. Peddinti, D. Povey, and S. Khudanpur, "A Time Delay Neural Network architecture for Efficient Modeling of Long Temporal Contexts," in *Proc. InterSpeech*, 2015.

[101] D. Povey, G. Cheng, Y. Wang, K. Li, H. Xu, Y. Mahsa, and S. Khudanpur, "Semi-Orthogonal Low-Rank Matrix Factorization for Deep Neural Networks," in *Proc. Interspeech*, 2018.

[102] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-Propagating Errors," *Nature*, vol. 323, no. 6088, p. 533, 1986.

[103] R. Williams and D. Zipser, "A Learning Algorithm for Continually Running fully Recurrent Neural Networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[104] A. J. Robinson, "An Application of Recurrent Nets to Phone Probability Estimation," *IEEE transactions on Neural Networks*, vol. 5, no. 2, pp. 298–305, 1994.

[105] T. Robinson, M. Hochberg, and S. Renals, "The Use of Recurrent Neural Networks in Continuous Speech Recognition," in *Automatic speech and speaker recognition*, pp. 233–258, Springer, 1996.

[106] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

[107] A. Robinson and F. Fallside, "The Utility driven Dynamic Error Propagation Network," tech. rep., University of Cambridge Department of Engineering, 1987.

[108] H. T. Siegelmann, "Computation beyond the Turing Limit," *Science*, vol. 268, no. 5210, pp. 545–548, 1995.

[109] H. T. Siegelmann and E. D. Sontag, "Turing Computability with Neural Nets," *Applied Mathematics Letters*, vol. 4, no. 6, pp. 77–80, 1991.

[110] H. T. Siegelmann and E. D. Sontag, "On the Computational Power of Neural Nets," *Journal of computer and system sciences*, vol. 50, no. 1, pp. 132–150, 1995.

[111] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of Gated Recurrent Neural Networks on Sequence Modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[112] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Gated Feedback Recurrent Neural Networks," in *Proc. ICML*, 2015.

[113] M. Schuster and K. K. Paliwal, "Bidirectional Recurrent Neural Networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[114] K. Sim, A. Narayanan, T. Bagby, T. Sainath, and M. Bacchiani, "Improving the Efficiency of Forward-Backward algorithm using Batched Computation in TensorFlow," in *Proc. ASRU*, 2017.

[115] K. Veselỳ, A. Ghoshal, L. Burget, and D. Povey, "Sequence-discriminative training of Deep Neural Networks," in *Proc. Interspeech*, 2013.

[116] C. Zhang, *Joint Training Methods for Tandem and Hybrid Speech Recognition Systems using Deep Neural Networks*. PhD thesis, University of Cambridge, 2017. Chapter 2, section 7.

[117] S. H. Friedberg, A. J. Insel, and L. E. Spence, *Linear Algebra-4th Edition*. Prentice Hall, 2002.

[118] X. Glorot and Y. Bengio, "Understanding the Difficulty of training Deep Feedforward Neural Networks," in *Proc. AISTATS*, 2010.

[119] C. Zhang and P. C. Woodland, "Parameterised sigmoid and ReLU hidden activation functions for DNN acoustic modelling," in *Proc. InterSpeech*, 2015.

[120] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-Normalizing Neural Networks," in *Proc. NIPS*, 2017.

[121] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks," in *Proc. AISTATS*, 2011.

[122] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-level Performance on Image-Net Classification," in *Proc. ICCV*, 2015.

[123] "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,"

[124] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier Nonlinearities Improve Neural Network Acoustic Models," in *Proc. ICML*, 2013.

[125] B. Neyshabur, Y. Wu, R. R. Salakhutdinov, and N. Srebro, "Path-normalized Optimization of Recurrent Neural Networks with ReLU Activations," in *Proc. NIPS*, 2016.

[126] C. Zhang and P. Woodland, "Semi-tied Units for Efficient Gating in LSTM and Highway Networks," in *Proc. Interspeech*, 2018.

[127] C. Zhang and P. Woodland, "High Order Recurrent Neural Networks for Acoustic Modelling," in *Proc. ICASSP*, 2018.

[128] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," in *Proc. ICLR*, 2015.

[129] M. Kline, *Calculus: An Intuitive and Physical Approach*. Dover Publications, 1998.

[130] D. F. Shanno, "On Broyden-Fletcher-Goldfarb-Shanno method," *Journal of Optimization Theory and Applications*, vol. 46, no. 1, pp. 87–94, 1985.

[131] J. Nocedal and S. Wright, *Numerical Optimization*. Springer, 2016.

[132] D. C. Liu and J. Nocedal, "On the Limited Memory BFGS method for Large Scale Optimization," *Mathematical programming*, vol. 45, no. 1-3, pp. 503–528, 1989.

[133] R. W. Wedderburn, "Quasi-likelihood functions, Generalized Linear models, and the Gauss Newton method," *Biometrika*, vol. 61, no. 3, pp. 439–447, 1974.

[134] H. Robbins and S. Monro, "A Stochastic Approximation Method," in *Herbert Robbins Selected Papers*, Springer, 1985.

[135] S. L. Smith and Q. V. Le, "A Bayesian Perspective on Generalization and Stochastic Gradient Descent," in *Proc. ICLR*, 2014.

[136] D. P. Kingma and J. L. Ba, "Adam: A method for Stochastic Optimization," in *Proc. International Conference on Learning Represenations*, 2015.

[137] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2010.

[138] A. Senior, G. Heigold, M. Ranzato, and K. Yang, "An Empirical Study of Learning Rates in Deep Neural Networks for Speech Recognition," in *Proc. ICASSP*, 2013.

[139] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for Word Representation," in *Proc. EMNLP*, 2014.

[140] G. Hinton, "Lecture 6.5-RMSprop: Divide the gradient by a Running Average of its recent Magnitude." COURSERA: Neural Networks for Machine Learning, 2012.

[141] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," *arXiv preprint arXiv:1212.5701*, 2012.

[142] S. L. Smith, P. J. Kindermans, and Q. V. Le, "Don't Decay the Learning Rate, Increase the Batch Size," *arXiv preprint arXiv:1711.00489*, 2017.

[143] Y. Nesterov, "A method of Solving a Convex Programming Problem with Convergence Rate O (1/k2)," *Soviet Mathematics Doklady*, vol. 27, no. 2, pp. 372–376, 1983.

[144] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *Proc. CVPR*, 2017.

[145] M. Johnson, M. Schuster, Q. V. Le, M. Krikun, Y. Wu, Z. Chen, N. Thorat, F. Viégas, M. Wattenberg, G. Corrado, *et al.*, "Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 339–351, 2017.

[146] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of Adam and Beyond," in *Proc. ICLR*, 2018.

[147] F. Treves, *Topological Vector Spaces, Distributions and Kernels: Pure and Applied Mathematics*, vol. 25. Elsevier, 2016.

[148] A. N. Tikhonov, A. S. Leonov, and A. G. Yagola, *Nonlinear ill-posed problems*. Springer, 1998.

[149] R. Tibshirani, "Regression Shrinkage and Selection via the Lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[150] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[151] T. K. Ho, "Random Decision Forests," in *Proc. Document Analysis and Recognition*, 1995.

[152] L. Breiman, "Bagging Predictors," tech. rep., University of California, 1994.

[153] T. Sainath, B. Kingsbury, H. Soltau, and B. Ramabhadran, "Optimization techniques to improve Training speed of Deep Neural Networks for Large Speech tasks," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 11, pp. 2267–2276, 2013.

[154] J. R. Shewchuk *et al.*, "An introduction to the Conjugate Gradient Method without the agonizing pain," 1994.

[155] J. Martens, "Deep learning via Hessian-free optimization.," in *Proc. ICML*, 2010.

[156] S. Wiesler, J. Li, and J. Xue, "Investigations on Hessian-Free Optimization for Cross-entropy Training of Deep Neural Networks.," in *Proc. InterSpeech*, 2013.

[157] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten Digit Database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, 2010.

[158] N. N. Schraudolph, "Fast Curvature Matrix-Vector products for Second-Order Gradient Descent," *Neural computation*, vol. 14, no. 7, pp. 1723–1738, 2002.

[159] B. A. Pearlmutter, "Fast exact multiplication by the Hessian," *Neural computation*, vol. 6, no. 1, pp. 147–160, 1994.

[160] P. Aubert and N. Di Césaré, "Expression Templates and Forward Mode Automatic Differentiation," in *Automatic differentiation of algorithms*, pp. 311–315, Springer, 2002.

[161] P. Dognin and V. Goel, "Combining Stochastic Average Gradient and Hessian-Free optimization for Sequence Training of Deep Neural Networks," in *Proc. ASRU*, 2013.

[162] A. Y. Ng, "Feature selection, L 1 vs. L 2 Regularization, and Rotational Invariance," in *Proc. ICML*, 2004.

[163] S. Amari, "Natural Gradient works Efficiently in Learning," *Neural computation*, vol. 10, no. 2, pp. 251–276, 1998.

[164] R. Pascanu and Y. Bengio, "Revisiting Natural Gradient for Deep Networks," arXiv preprint arXiv:1301.3584, 2013.

[165] J. Martens and R. Grosse, "Optimizing Neural Networks with Kronecker-factored Approximate Curvature," in *Proc. ICML*, 2015.

[166] D. Povey, X. Zhang, and S. Khudanpur, "Parallel Training of Deep Neural Networks with Natural Gradient and Parameter Averaging," *arXiv preprint arXiv:1410.7455*, 2014.

[167] Y. Ollivier, "Riemannian Metrics for Neural Networks I: Feedforward Networks," *Information and Inference: A Journal of the IMA*, vol. 4, no. 2, pp. 108–153, 2015.

[168] J. M. Lee, "Smooth manifolds," in *Introduction to Smooth Manifolds*, Springer, 2003.

[169] S. Kullback, "Letter to the editor: The Kullback-Leibler distance," *American Statistician*, 1987.

[170] Q. Liao, J. Z. Leibo, and T. A. Poggio, "How Important is Weight Symmetry in Backpropagation?," in *Proc. AAAI*, 2016.

[171] J. J. Rotman, *An Introduction to the Theory of Groups*, vol. 148. Springer, 2012.

[172] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized Stochastic Gradient Descent," in *Proc. NIPS*, 2010.

[173] X. Zhang, J. Trmal, D. Povey, and S. Khudanpur, "Improving Deep Neural Network Acoustic Models using Generalized Maxout Networks," in *Proc. ICASSP*, 2014.

[174] D. Paul and J. Baker, "The Design for the Wall Street Journal based CSR Corpus," in *Proc. DARPA SLS*, 1992.

[175] P. Lanchantin, M. J. F. Gales, P. Karanasou, X. Liu, Y. Qian, L. Wang, P. Woodland, and C. Zhang, "The Development of the Cambridge University alignment systems for the Multi-Genre Broadcast Challenge," in *Proc. ASRU*, 2015.

[176] P. Karanasou, M. J. F. Gales, P. Lanchantin, X. Liu, Y. Qian, L. Wang, P. C. Woodland, and C. Zhang, "Speaker Diarisation and longitudinal linking in Multi-Genre Broadcast Data," in *Proc. ASRU*, 2015.