Computationally efficient natural gradient descent

Sara-Jayne Farmer

Master of Science by Research
in
Pattern Analysis and Neural Networks

Aston University

September 1998

Aston University

Computationally efficient natural gradient descent

Sara-Jayne Farmer

Master of Science by Research
in
Pattern Analysis and Neural Networks, 1998

This study examines the use of matrix momentum terms with the aim of creating a more computationally efficient natural gradient descent algorithm for on-line learning. It uses the statistical mechanics framework created by Saad and Solla to describe the evolution of order parameters for this algorithm in a two-layer student-teacher scenario, and compares this with results from matrix-momentum natural gradient learning of real datasets.

Acknowledgements

Thanks go to Magnus Rattray and David Saad for supervision, guidance and several patient explanations of strange matrix indices: any good stuff in this thesis is inevitably theirs. Thanks also go to Genny Orr for providing the phoneme classification dataset used in chapter 4.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This study examines the use of matrix momentum terms and natural gradient descent in the online learning of neural networks. Its aim is to create a more computationally efficient natural gradient descent algorithm for on-line learning. It uses the statistical mechanics framework created in [24] to describe the evolution of order parameters for this algorithm in two-layer networks within a student-teacher scenario.

This chapter introduces the tools needed for this work and some of the issues in their use.

1.1 Neural networks

A neural network is a flexible probabilistic model of an unknown mapping instanced by a set of examples. A network has nodes, parameters and activation functions. Nodes receive, modify and pass on (propagate) information. Input nodes receive information in the form of vectors or individual values from outside. Output nodes modify then pass information to the user. Hidden nodes modify and pass information between the input and output nodes, and play the main role in processing incoming data in a network. A network's structure (nodes and the layout of links between them) is usually specified by the person building the network, but there are some algorithms that can adapt the network structure to best represent the data being learnt.

Each node in a network may have several inputs, either from other nodes or from outside the network. The values input to a node are termed its activation; the value output from a node is a function of its activations and is termed its activation function. Activation functions are specified by the network designer and mostly range from simple summation

of all the inputs to a function of that sum. For nonlinear networks (networks that can represent nonlinear functions) and derivative-based learning algorithms, the activation function of the hidden nodes must be differentiable, for example tanh or erf fuctions (these also squash their outputs into a range between 0 and 1 or -1 and +1).

Networks also have numerical parameters which are used to control the behaviour of a network and are usually the weights on the links between nodes. the weight on a link is used to modify any output from one node before it is input to the other. Parameters can be modified to improve the performance of a network: this modification is known as learning or training. Learning algorithms adapt the parameters of a network so that the outputs from it are as close as possible to the outputs given by the system that they are modelling given random inputs from the same distribution as the examples.

Using a network consists of passing information to its input nodes and reading off the activations of its output nodes. Although the ability to store examples may be useful, the main aspect of using a neural network that we will focus on is generalisation: the ability to summarise information and return consistent outputs for inputs that the network has not been trained on. This is achieved with a learning algorithm. A learning algorithm adjusts the *parameters* Θ of a network to fit a set of *examples*, where each example consists of an input vector ξ^μ and the output ζ^μ that is expected from the network, given that input and the system that the network is trying to model(here, μ is an index to the example, as are most superscripts used in this document).

Because networks learn the underlying function that generated a set of examples rather than the exact connection between each example's input and output pair, they are robust: able to learn from noisy or incomplete examples. How well a network is able to generalise is determined by the user's choice of network and activation function; tools like cross-validation can be used to assess the expected performance. Neural networks are most commonly used to model systems which can produce example outputs or input-output pairs but whose underlying rules are either unknown or too expensive to model. There are several varieties of neural network architectures, for instance multilayer perceptrons and radial basis functions: in this document, only the networks known as multilayer perceptrons are considered.

1.2 Multilayer perceptrons

The most commonly-used neural networks are multilayer perceptrons, or MLPs. The nodes in an MLP are arranged into layers : an input layer, then one or more hidden layers and an output layer, where each layer is fully connected to the next layer in the network. Connections between layers can only be directed forwards in the network: for instance, connections can be made from a hidden layer to the output layer but connections from the output layer back to a hidden layer are not allowed. Although not strictly necessary, the same activation function is usually defined for all hidden and output nodes in an MLP.

Using an MLP consists of activating the input nodes, propagating that input between the network's layers and reading the output from its output nodes.

The most common MLP has one input layer, one hidden layer and one output layer and is known as a two-layer network (the input layer is ignored when counting the number of layers in a network). There are some variants of MLPs, for example a soft committee network or soft committee machine [4] is a two-layer MLP whose hidden to output weights are not allowed to vary, and are usually set to one, and output nodes whose activation functions are or are proportional to a simple sum of their input activations. Soft committee machines are used as example networks for much of this document and preserve most of the characteristics of general two layer networks [20]. Reviews of other MLP variants and their learning methods can be found in [5][9]).

1.2.1 Backpropagation learning

In the soft committee machines used in this document, the MLP parameters $\boldsymbol{\Theta}$ are restricted to the input-to-hidden weights $\mathbf{J}$. Before learning occurs, these are set to initial values which are usually small and randomly distributed.

Learning in MLPs is usually done by backpropagation, which works in two stages. First, an input ξ^μ is propagated (modified and passed on) forward from the input layer to the output layer through all the hidden layers of an MLP, to produce a network output $\sigma(\mathbf{J}, \xi^\mu)$. This is the forward pass of backpropagation. The network output $\sigma(\mathbf{J}, \xi^\mu)$ produced in the forward pass is then compared against the expected output ζ^μ for the corresponding input ξ^μ. This comparison produces a network error $\epsilon_{\mathbf{J}}(\xi^\mu, \zeta^\mu)$, which is then propagated back from the output units to the input units, to give the local gradient δ_i^μ for each node i. This is the backward pass which gives the algorithm its name. When the backward pass through the network is completed, the local gradients δ_i^μ (also known

as backpropagation delta functions) are used to adjust the network parameters, and the next forward pass can begin.

The network *error* $\epsilon_{\mathbf{J}}(\xi^{\mu}, \zeta^{\mu})$ is used in the backward pass of the backpropagation error. $\epsilon_{\mathbf{J}}(\xi^{\mu}, \zeta^{\mu})$ is a measure for the distance between the actual output $\sigma(\mathbf{J}, \xi^{\mu})$ and the expected output ζ^{μ} of a network with parameters $\mathbf{J}$ which has been given the input ξ^{μ}. In this document, this distance is taken to be quadratic:

$$\epsilon_{\mathbf{J}}(\xi^{\mu}, \zeta^{\mu}) = \frac{1}{2}\left[\sigma(\mathbf{J}, \xi^{\mu}) - \zeta^{\mu})\right]^2 \tag{1.1}$$

In *online learning*, the network error for just the latest example (ξ^{μ}, ζ^{μ}) is propagated back through the network in the backward pass of the algorithm; in *batch learning*, an average over all the available input patterns $\mathbf{D} = \{(\xi^1, \zeta^1)...(\xi^M, \zeta^M)\}$ is used. This means that all the examples must be available at every stage of batch learning, but online learning can process one example at a time.

Although the network error is used in this learning algorithm, the *generalisation error* $\epsilon_g(\mathbf{J})$ is more useful in analysing the performance of a learning algorithm. The generalisation error gives the probability that the network will produce the wrong output for an example that it has not been trained on but is drawn from the same distribution or model as the training examples; it is defined as the network error averaged over the distribution of network inputs:

$$\epsilon_g(\mathbf{J}) = \langle \epsilon_{\mathbf{J}}(\xi^{\mu}, \zeta^{\mu}) \rangle_{\{\xi\}} \tag{1.2}$$

1.2.2 Gradient descent

The errors for different values of the network parameters form an error landscape or surface, where each point represents the error for a particular choice of parameters and distances between points measure the differences between the errors.

The error term passed back through the network is not the error at each node, but the derivative of the network error with respect to the node's parameters. This is proportional to the derivative of the nodes activation function, which must therefore be differentiable for backpropagation to work.

Most MLP learning algorithms adjust the network parameters $\mathbf{J}$ by subtracting a fraction η of the error gradient $\nabla_{\mathbf{J}} \epsilon_{\mathbf{J}}(\xi^{\mu}, \zeta^{\mu})$ from them:

$$\mathbf{J}^{\mu+1} = \mathbf{J}^{\mu} - \eta \nabla_{\mathbf{J}} \epsilon_{\mathbf{J}}(\xi^{\mu}, \zeta^{\mu}) \tag{1.3}$$

where η is known as the *learning rate*. This ensures that, for a general series of examples (ξ^μ, ζ^μ) and an appropriate value of η, the learning algorithm will move the network error, on average, downhill in the error landscape.

The learning rate η controls the step-size on the error landscape. If it is constant throughout learning a network's parameters, then the learning algorithm will either be forced to take small steps and a long time traversing the landscape towards an error surface minimum, or larger steps which will overshoot the minimum in the final stages of learning. These problems can be avoided (or alleviated) by *annealing* the learning rate: gradually reducing the value of η over time.

1.3 Online learning

Algorithms where parameter adjustment is done as each new example is presented (rather than for all examples at once) are known as *on-line learning* or *pattern learning*. Online learning is useful when examples are only available serially, the task is nonstationary (the function that is being learnt by the network is changing over time) or there is less data, space or time available than is needed to process an entire dataset (group of examples) simultaneously. This report deals exclusively with online learning.

1.3.1 Online learning and time-varying data

For many of the equations shown below, the input data is assumed to be iid: each input is uncorrelated with the inputs that went before it. Data for online learning is produced sequentially or sampled from datasets too big to be loaded completely into a system. If data is sampled with replacement, then it will not be uncorrelated. Many examples of real sequential data are from underlying systems whose parameters change over time which often means that there are weak correlations between successive inputs, and may imply that older data will be of less value to the learning algorithm and might need to be forgotten. These factors have not been considered in this report, but may prove significant when using real data [1].

[1] For example, Heskes and Coolen[11] report on the effects of correlation between inputs to two-layer networks, but this has not been considered here

1.4 Neural networks as statistical approximators

A neural network models the expected value

$$\langle P(\zeta|\xi;\mathbf{J})\rangle_{\{\xi,\zeta\}} \tag{1.4}$$

of the conditional probability of its outputs ζ given its inputs ξ, parameterised by its weights $\mathbf{J}$ and sometimes also parameterised by its structure. Learning in a network is the process of adapting the parameters $\mathbf{J}$ to get the mapping from this model as close as possible to the underlying rule that generated a set of example input-output pairs $\mathbf{D} = \{(\xi^1,\zeta^1)...(\xi^M,\zeta^M)\}$. Given a prior distribution for the data $P(\mathbf{D})$, this can also be seen as maximising $P(\mathbf{J}|\mathbf{D})$ with respect to $\mathbf{J}$.

1.5 Neural networks as points in parameter space

Information geometry is the application of techniques from differential geometry to statistical models [13] [1]. It represents families of neural networks as manifolds (k-dimensional geometric objects which can be mapped onto Euclidean space) in a space (d-dimensional, where k¡d) whose coordinates are the parameters (weights and biases) of the networks, and each individual network is represented by a point on the manifold which corresponds to its particular parameters. As an example, a two-layer soft committee network with weights $\mathbf{J}_1...\mathbf{J}_N$ can be represented as a point on a manifold of soft committee network mappings in N-dimensional parameter space, and similar networks with different parameter values would be represented by different points on the same manifold.

The curvatures of the manifolds are described using tensors. A Riemann tensor is a matrix which describes the curvature of a surface at a specific point in space. In n-dimensional space, the tensor is an n-by-n symmetric matrix: e.g. in four dimensions, this is 4-by-4 with 16 elements but only 10 independent components because of its symmetry. A complete surface is described by placing tensors at different points on the surface, or by parametrising the tensor so that any point on the surface could be described: the resulting collection of tensors is known as a vector field, and differential geometry is the study of their properties.

1.5.1 Applying differential geometry to neural networks

In this representation of a neural network, learning can be seen as moving the point that represents a particular network closer to a (perhaps theoretical) set of parameters that

generated a set of given input-output examples. This can be a powerful tool for the analysis of learning, and it can also be used to create optimal or near-optimal learning algorithms.

Having described learning as moving points closer together in parameter space, we need to have a measure of how close two points in that space are in terms of the probability distributions realised by them. There are several measures [2] (for example Kullback-Leibler, Hellinger, squared and Euclidean) in statistics for the distance between two distributions p and q, but one of the most natural and useful is the Kullback-Leibler distance

$$KL(p(x), q(x)) = \int p(x) \log \left(\frac{p(x)}{q(x)} \right) dx \tag{1.5}$$

which for two distributions $p(\zeta, \xi; \mathbf{J})$ and $q(\zeta, \xi; \mathbf{J})$ of network inputs ξ and outputs ζ parameterised by $\mathbf{J}$ is

$$KL(p(\zeta, \xi; \mathbf{J}), q(\zeta, \xi; \mathbf{J})) = \int p(\zeta, \xi; \mathbf{J}) \log \left(\frac{p(\zeta, \xi; \mathbf{J})}{q(\zeta, \xi; \mathbf{J})} \right) d(\xi, \zeta) \tag{1.6}$$

We need to apply this to the parameter-space representation of neural networks.

In information geometry, the Kullback-Leibler distance between mappings realised by two nearby points in a space is known as a metric : if we are measuring small distances in parameter space (which is what we expect in online learning) and can ignore higher orders of $d\mathbf{J}$, then equation (1.6) becomes

$$KL(p(\zeta, \xi; \mathbf{J}), p(\zeta, \xi; \mathbf{J} + d\mathbf{J})) = \frac{1}{2} d\mathbf{J}^T \mathbf{G}(\mathbf{J}) d\mathbf{J} \tag{1.7}$$

where

$$\mathbf{G}(\mathbf{J}) = \langle (\nabla_{\mathbf{J}} \log P_{\mathbf{J}}(\zeta, \xi; \mathbf{J}) (\nabla_{\mathbf{J}} \log P_{\mathbf{J}}(\zeta, \xi; \mathbf{J})^T \rangle_{\{p(\zeta, \xi)\}} \tag{1.8}$$

is the metric for this space, and is known as the Fisher information matrix. Gradient descent in parameter spaces which use the Kullback-Leibler measure of distance is known as natural gradient descent.

1.6 Second-order learning methods

Several second order methods have been devised to speed up learning. Examining the first few terms of a Taylor expansion of an error $\epsilon_{\mathbf{J}}(\xi^\mu, \zeta^\mu)$ about a point $\mathbf{J}$ in weight-space gives [5]:

$$\epsilon_{\hat{\mathbf{J}}}(\xi, \zeta) = \epsilon_{\mathbf{J}}(\xi, \zeta) + (\hat{\mathbf{J}} - \mathbf{J}) \nabla_{\mathbf{J}} \epsilon_{\mathbf{J}}(\xi, \zeta)|_{\mathbf{J}} + \frac{1}{2} (\hat{\mathbf{J}} - \mathbf{J})^T \mathbf{H} (\hat{\mathbf{J}} - \mathbf{J}) \tag{1.9}$$

[2] these are all defined through delta-divergences [29][30]

where

$$\mathbf{H} = \left[\frac{\partial \epsilon_{\mathbf{J}}(\xi, \zeta)}{\partial \mathbf{J}_i \partial \mathbf{J}_k} |_{\mathbf{J}} \right] \tag{1.10}$$

is known as the Hessian matrix and

$$\nabla_{\mathbf{J}} \epsilon_{\mathbf{J}}(\xi, \zeta)|_{\mathbf{J}} = \frac{\partial \epsilon_{\mathbf{J}}(\xi, \zeta)}{\partial \mathbf{J}_i} |_{\mathbf{J}} \tag{1.11}$$

is the error surface gradient at $\mathbf{J}$.

Many gradient descent algorithms use the error's gradient $\nabla_{\mathbf{J}} \epsilon_{\mathbf{J}}(\xi, \zeta)$ to calculate where to move next in the energy landscape, and ignore any higher-order terms, but an improvement in their learning speed can be had from using the second-order term $\frac{1}{2}(\hat{\mathbf{J}} - \mathbf{J})^T \mathbf{H} (\hat{\mathbf{J}} - \mathbf{J})$ too. (Third and higher-order terms aren't used because they are usually insignificant and are more difficult to handle).

1.6.1 Newton's method

The most common second-order network learning method is based on Newton's method for finding the minimum of a function[5].

If we assume that the local gradient obtained by differentiating the second-order $\epsilon_{\mathbf{J}}(\xi, \zeta)$ estimate (equation 1.9) with respect to $(\hat{\mathbf{J}} - \mathbf{J})$ is zero, then we have

$$\nabla_{\mathbf{J}} \epsilon_{\mathbf{J}}(\xi, \zeta)|_{\mathbf{J}} + \mathbf{H}(\hat{\mathbf{J}} - \mathbf{J}) = 0 \tag{1.12}$$

i.e.

$$\hat{\mathbf{J}} = \mathbf{J} - \mathbf{H}^{-1} \nabla_{\mathbf{J}} \epsilon_{\mathbf{J}}(\xi, \zeta) \tag{1.13}$$

Since we have ignored higher-order terms in the Taylor expansion, this is not an exact equation for $\hat{\mathbf{J}}$, and an iterative procedure must be used (we also focus on online learning which requires iterative updates). This gives an update equation for the weights $\mathbf{J}$ of

$$\mathbf{J}^{\mu+1} = \mathbf{J}^{\mu} - \eta \mathbf{H}^{-1} \nabla_{\mathbf{J}} \epsilon_{\mathbf{J}}(\xi^{\mu}, \zeta) \tag{1.14}$$

η is the learning rate as before.

Learning with Newton's method converges faster than first-order gradient descent but it is very sensitive to rescaling (e.g. normalisation) of inputs, and it needs a lot of processing and system space to create, store and invert the Hessian matrix. In addition, employing Newton's method makes the algorithm converge towards and stabilise on any fixed points in the error landscape regardless of whether they are minima or saddlepoints, and this may result in suboptimal performance in complex learning dynamics.

1.6.2 Approximating the Hessian

Accurately calculating the inverse Hessian $\mathbf{H}^{-}1$ used in Newton's method requires an average over all the input data (this is, of course, only available in batch learning), followed by a (large) matrix inversion, at every step of the gradient descent algorithm. This can take a long time to calculate, and since the inverse Hessian is a very large matrix, it can be impractical or impossible to store during calculations. It therefore makes sense to look for more efficient Hessian algorithms and reasonable approximations to these matrices.

One approximation to the Hessian is to use only some of the input data to evaluate the Hessian matrix. Where only the latest input is used, this is termed here a single-step approximation. The calculations used to create the elements of the Hessian matrix can also be approximated : variations of Newton's method that use this are known as pseudo-Newton or quasi-Newton algorithms. Simple approximations to the Hessian include diagonal approximations and the Levenberg-Marquardt approximation[5]. Diagonal approximations only contain the elements on the diagonal of the Hessian. they are not very precise: the off-diagonal elements of the Hessian are significant in many learning scenarios, and inverting a diagonal approximation of the Hessian differs significantly from the diagonal of its inverse [16]. the Levenberg-Marquardt approximation depends on the expansion of the Hessian for a sum-of-squares error $\epsilon_{\mathbf{J}}(\xi, \zeta) = \frac{1}{2}\sum_n (\sigma_n - \zeta_n)^2$ into its components:

$$\frac{\partial^2 \epsilon_{\mathbf{J}}(\xi, \zeta)}{\partial \mathbf{J}_i \partial \mathbf{J}_k} = \sum_n \frac{\partial \sigma_n}{\partial \mathbf{J}_i}\frac{\partial \sigma_n}{\partial \mathbf{J}_k} + \sum_n (\sigma_n - \zeta_n)\frac{\partial^2 \sigma_n}{\partial \mathbf{J}_i \partial \mathbf{J}_k} \tag{1.15}$$

and ignoring the second term in that equation. This is a reasonable approximation if $(\sigma_n - \zeta_n)$ is small. Other algorithms build up the Hessian iteratively from first-order terms: the most common of these are the BFGS (Broyden-Fletcher-Goldfarb-Shanno) and the Davidson-Fletcher-Powell procedures.

Other algorithms include approximations to the inverse Hessian [8], but since the Hessian $\mathbf{H}$ is often used multiplied by a vector $\mathbf{v}$, it can be more practical to create the product $\mathbf{Hv}$ directly. Pearlmutter [16] gives an algorithm for doing this efficiently and exactly by setting $\mathbf{J} - \hat{\mathbf{J}} = r\mathbf{v}$ in the Taylor expansion above then taking limits as r approaches zero to give a differential operator which can be applied to a network to generate $\mathbf{Hv}$.

1.7 Matrix Momentum

Inverting the Hessian in Newton's method can be avoided by incorporating it into a matrix momentum term[15].

Momentum is a method used mostly in batch learning to counter the effect of gradient descent algorithms oscillating across large gradient changes in one dimension whilst giving little effort in a gradually-changing but consistent downward slope in another: this is most apparent when the error surface is a gently-descending valley with steep sides. One way to ameliorate this effect is to add a *momentum* term [22] to the weight update equations, which adds a fraction of the previous weight update into the current one. This reduces the strength of the oscillation across the surface whilst emphasising the smaller (but consistent) movement in the other dimension. Including a momentum term into the update equation 1.3 in an online learning scenario gives:

$$\mathbf{J}^{\mu+1} = \mathbf{J}^{\mu} - \eta \nabla_{\mathbf{J}} \epsilon_{\mathbf{J}}(\xi^{\mu}, \zeta) + \beta(\mathbf{J}^{\mu} - \mathbf{J}^{\mu-1}) \tag{1.16}$$

where β is the *momentum parameter* and is usually set between 0 and 1 (e.g. ref [10] suggest $\beta = 0.9$) but it may be adaptive in an attempt to speed up network convergence. Suggestions for speeding up network convergence also include using a separate η, β for each network weight, but using momentum already has this effect.

A simple momentum term (e.g. $\beta = constant$) is not terribly useful in online learning because its effect is merely to rescale the learning rate η. This is because, for an error surface with a gradient which is roughly constant near the current network parameters $\mathbf{J}^{\mu}$ (this is the situation that we expect in online learning), using the update equation 1.16 is equivalent to using a learning rate of [5] [27]

$$\eta_{\text{eff}} = \frac{\eta}{(1-\beta)} \tag{1.17}$$

If, however, the momentum term β in equation 1.16 is replaced with a matrix $\beta = I - k\mathbf{H}$ and the learning rate η is replaced with $\eta = k\eta_{\alpha}$ where η_{α} is a scalar learning rate which can be annealed (depend on the normalised example index $\alpha = \mu/N$), the update equation for the network becomes

$$\mathbf{J}^{\mu+1} = \mathbf{J}^{\mu} - k\eta_{\alpha} \nabla_{\mathbf{J}} \epsilon_{\mathbf{J}}(\xi^{\mu}, \zeta) + (I - k\mathbf{H})(\mathbf{J}^{\mu} - \mathbf{J}^{\mu-1}) \tag{1.18}$$

and the effective learning rate is

$$\eta_{\text{eff}} = \mathbf{H}^{-1}\eta_{\alpha} \tag{1.19}$$

This is known as *matrix momentum*: it is important because it gives a way of effectively premultiplying the gradient by a matrix inverse $\mathbf{H}^{-1}$ without having to calculate that matrix inverse [15]. Matrix momentum has been used by Orr and Leen[15] with the matrix $\mathbf{H}$ set to the Hessian to give Newton's method without inverting the Hessian.

Note that the constant k is used to balance the contributions from the gradient and momentum parts of the update equation, and that the matrix momentum equations above are only valid for large values of k[18].

1.7.1 Balancing gradient descent and matrix momentum

If we scale both η and β by N and introduce a new variable Ω where

$$\mathbf{\Omega}_i^\mu = N(\mathbf{J}_i^\mu - \mathbf{J}_i^{\mu-1}) \tag{1.20}$$

then we can substitute these into equation 1.16 to get an update equation for $\mathbf{J}$ of

$$\mathbf{J}_i^{\mu+1} = \mathbf{J}_i^\mu - \frac{\eta}{N}\nabla_{\mathbf{J}}\epsilon_{\mathbf{J}}(\xi^\mu, \zeta) + \frac{\beta}{N}\mathbf{\Omega}_i^\mu \tag{1.21}$$

We can then use equations 1.20 and 1.21 to also get an update equation for $\mathbf{\Omega}$ of

$$\mathbf{\Omega}_i^{\mu+1} = \beta\mathbf{\Omega}_i^\mu - \eta\nabla_{\mathbf{J}}\epsilon_{\mathbf{J}}(\xi^\mu, \zeta) = \mathbf{\Omega}_i^\mu - \eta\nabla_{\mathbf{J}}\epsilon_{\mathbf{J}}(\xi^\mu, \zeta) + (\beta - 1)\mathbf{\Omega}_i^\mu \tag{1.22}$$

where equation 1.21 and equation 1.22 form a coupled pair of equations.

Using this coupled pair of equations, reference [15], which is restricted to the asymptotic regime, suggests an optimal matrix momentum $\beta = \mathbf{I} - \eta_0\mathbf{H}$ and annealed learning rate $\eta = \eta_0/\alpha$ where η_0 is an initial learning rate. This gives an effective learning rate of

$$\eta_{\text{eff}} = \frac{\eta}{(1-\beta)} = \frac{\mathbf{H}^{-1}}{\alpha} \tag{1.23}$$

which is similar to Newton's method and does not require inverting the matrix $\mathbf{H}$.

An alternative scaling of the η, β terms was suggested by [17], where η and $(1-\beta)$ are both scaled by N. To do this, η and β are replaced by

$$\begin{aligned} \beta &= (1 - \frac{\gamma}{N}) \\ \eta &= k/N \end{aligned}$$

If we now use this scaling and let $\gamma \to \infty$, $k \to \infty$ with k/γ constant, the effective learning rate becomes [17]

$$\eta_{\text{eff}} = k/\gamma \tag{1.24}$$

This holds for $\beta = \mathbf{I} - k\mathbf{H}/N$ and $\eta = k\eta_\alpha/N$ with large values of k [18], again giving an effective learning rate η_{eff} of

$$\eta_{\text{eff}} = \eta_\alpha H^{-1} \tag{1.25}$$

where k balances the contributions of the gradient and momentum terms as before. Note that, for effective training, the learning rate η should start from a constant e.g. η_0, and only asymptotically decay as $\eta_\alpha \propto 1/\alpha$. These results are equally valid for any matrix, including the Fisher information matrix $\mathbf{G}(\mathbf{J})$, and this scaling will be used throughout the rest of this document.

1.8 Natural gradient descent

Natural gradient descent uses the Fisher information matrix as the natural metric in parameter space. For gradient descent, the effect of using this metric is to premultiply the gradient by the inverse of the Fisher information matrix, changing the update equation for online standard gradient descent from equation 1.3 to

$$\mathbf{J}^{\mu+1} = \mathbf{J}^\mu - \frac{\eta}{N}\mathbf{G}(\mathbf{J})^{-1}(\mathbf{J}^\mu)\nabla_{\mathbf{J}}\epsilon_{\mathbf{J}}(\xi,\zeta) \tag{1.26}$$

Useful features of natural gradient learning are that it is invariant to reparameterisation of the model distribution (e.g. scaling the parameters does not change the efficiency of the learning algorithm), is asymptotically optimal [2], less prone than Newton's method to trapping in symmetric phases (trapping occurs when the algorithm becomes stable at a saddlepoint or local minimum in the error landscape rather than its global minimum) and the Fisher matrix $\mathbf{G}(\mathbf{J})$ is always positive definite.

The natural gradient descent algorithm requires the inverse $\mathbf{G}(\mathbf{J})^{-1}(\mathbf{J}^\mu)$ of the Fisher information matrix. Inverting the Fisher matrix can take a long time. Schemes for calculating this inversion range from exact algorithms like block-wise partitioning and some data preprocessing to facilitate the calculation [28] to diagonal approximations [2] [21].

In general, these algorithms can be computationally expensive to calculate and calculating the average <> over all inputs in the Fisher matrix also requires knowledge of all input data: this may not be sensible or practical for many cases of on-line learning (which is based on the premise that data is not all available at the same time). This document concentrates on the first problem: the second is addressed in [25].

1.8.1 The Fisher information matrix

The loss function L of a network is defined [28] as the negative log-likelihood function

$$L(\xi, \zeta; \mathbf{J}) = -\log p(\xi, \zeta; \mathbf{J}) = -\log p(\zeta|\xi; \mathbf{J}) + \log p(\xi) \tag{1.27}$$

where $p(\xi, \zeta; \mathbf{J})$ is the joint probability density function of the network inputs ξ and outputs ζ parameterised by its weights $\mathbf{J}$.

The Fisher information matrix $\mathbf{G}(\mathbf{J})$ of a network is defined from L as

$$\mathbf{G}(\mathbf{J}) = [G_{i\alpha,k\beta}] \tag{1.28}$$

where α and β are input node indices $(1 \leq i, k \leq N)$, i and k are hidden node indices $(1 \leq i, k \leq K)$ and [.] is a block matrix whose elements are

$$\mathbf{G}_{i\alpha,k\beta} = \left\langle \frac{\partial L}{\partial \mathbf{J}_i} \frac{\partial L}{\partial \mathbf{J}_k} \right\rangle_{\{\zeta,\xi\}} = \left\langle \frac{\partial \log p(\zeta|\xi; \mathbf{J})}{\partial \mathbf{J}_i} \frac{\partial \log p(\zeta|\xi; \mathbf{J})}{\partial \mathbf{J}_k} \right\rangle_{\{\zeta,\xi\}} \tag{1.29}$$

where $<>_{\{\zeta,\xi\}}$ is an average over the output distribution ζ, followed by an average over the input distribution ξ.

1.8.2 Matrix momentum for natural gradient descent (MM-NGD)

Matrix momentum is valid for any matrix. If we use the Fisher information matrix, we have a learning algorithm that is equivalent to natural gradient descent without inverting the Fisher information matrix.

The update equations for a matrix momentum form of natural gradient descent (from now on, abbreviated to MM-NGD) with η and β scaled by the input size N are

$$\mathbf{J}_i^{\mu+1} = \mathbf{J}_i^{\mu} - \frac{\eta}{N} \nabla_{\mathbf{J}} \epsilon_{\mathbf{J}}(\xi, \zeta) + \frac{\beta}{N} \mathbf{\Omega}_i^{\mu} \tag{1.30}$$

$$\mathbf{\Omega}_i^{\mu+1} = \beta \mathbf{\Omega}_i^{\mu} - \eta \nabla_{\mathbf{J}} \epsilon_{\mathbf{J}}(\xi, \zeta) \tag{1.31}$$

$$\beta = I - \frac{k\mathbf{G}(\mathbf{J})}{N} \tag{1.32}$$

$$\eta = \frac{k\eta_\alpha}{N} \tag{1.33}$$

where $\mathbf{\Omega}_i^{\mu} = N(\mathbf{J}_i^{\mu} - \mathbf{J}_i^{\mu-1})$, η_α is a scalar learning rate which can be annealed (depend on the value of $\alpha = \mu/N$, the normalised example index), and $\mathbf{G}(\mathbf{J})$ is the Fisher information matrix.

When η is significantly larger than β, the learning is dominated by the gradient and is expected to be very similar to gradient descent. When β is significantly larger than η, then learning is dominated by the momentum terms and can be expected to depend

largely on the value of the Fisher information matrix. The next chapter takes this as a starting point, and analyses the effects of varying the parameters of MM-NGD learning for soft committee networks.

Chapter 2

Analysis of MM-NGD for online learning

This chapter introduces the tools used to analyse the behaviour of a network's parameters and output errors during learning. It gives examples of this and equations for learning in a type of MLP known as a soft committee machine.

2.1 Statistical mechanics analysis of online learning

Statistical mechanics seeks, in general, to describe a system of many interacting particles (in this case, network parameters) in terms of a smaller number of *order parameters.* These can be used to describe the system only if it is *self-averaging* : if the parameters for each individual particle are identical (or very similar to) the average parameters for all the particles. In the discussion that follows, all the order parameters have very sharply peaked distributions with small variance, so only their mean values will be analysed.

It is difficult to reduce the number of parameters studied if we are looking at an individual network, but if we assume that the outputs in the 'correct' input-output examples given to a learning algorithm are generated by feeding the inputs into another network, then we can describe the learning algorithm in terms of the difference between this 'teacher' network and the 'student' network which is learning from the examples. A covariance matrix can be formed between the student and teacher activations. This contains the overlaps $\mathbf{Q}_{ik} = \mathbf{J}_i \cdot \mathbf{J}_k$, $\mathbf{R}_{in} = \mathbf{J}_i \cdot \mathbf{B}_n$, $\mathbf{T}_{nm} = \mathbf{B}_n \cdot \mathbf{B}_m$ between the two networks : each of these overlaps can be used as an order parameter for online learning [24][4]. Note that the teacher network does not have to produce perfect outputs: it may include some

noise ρ on its output, and that the teacher network does not have to exist: it is merely a construction to give us a better insight into how a network is learning.

2.1.1 Evolution of order parameters

To derive differential equations to describe the evolution of order parameters, we need to use a continuous time variable α. Since μ is discrete, α is constructed by setting $\alpha = \mu/N$, which in the thermodynamic limit of $N \to \infty$ becomes continuous.

Equations for the evolution of network order parameters, derived from the simple weight update equation given above are provided in [24].

2.1.2 Evolution of generalisation error

The generalisation error $\epsilon_g(\mathbf{J}) =< \epsilon_{\mathbf{J}}(\xi, \zeta) >_{\{\xi\}}$ of the student network can be described wholly in terms of $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{T}$. Unlike $\mathbf{J}$, the evolution of $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{T}$ over time (or over presented input patterns) can be described deterministically for large values of N: the derivatives of the overlaps form a coupled and closed differential equation. In this case, the reduction in number of variables is considerable: since much of the maths in, for example, [24] assumes an infinite input size N, the order parameters also have the advantage of being finite.

Evolution of the generalisation error $\epsilon_g(\mathbf{J})$ for a network has a characteristic shape: the error will initially drop rapidly, then sit on a plateau until it continues to drop again. The plateau is where the network is close to an unstable fixed point of the error landscape where the network weights have become symmetric : this fixed point is unstable and eventually the symmetry will be broken and the learning algorithm will continue towards an asymptotic stable fixed point. Both the length of the plateau and the rate at which the error drops after it are determined by the system size and the learning rate η : a larger value of η will reduce the plateau length but will take much longer to run if η-annealing is used.

2.2 Example networks

To illustrate the efficiency and behaviour of different gradient descent learning algorithms, the dynamics of their order parameters and generalisation error have been calculated for a specific type of network. The example networks used in this section are soft committee machines with erf hidden unit activations.

Soft committee machines are used as examples here because they can model a wide range of functions (in fact, they are universal approximators if node biases are included [26]). A soft committee machine is a two-layer network with positive, unit-strength couplings from its hidden units to a single output unit[4]. The activation function for hidden units must be differentiable: the erf (error) function is used here because its integral is not too complicated.

Each network has a fixed number N of input nodes, one output node and weights on the connections between the input and hidden layer which are labelled $\mathbf{J} \equiv \{\mathbf{J}_i\}_{1 \leq i \leq K}$ in the student network and $\mathbf{B} \equiv \{\mathbf{B}_n\}_{1 \leq n \leq M}$ in the teacher network, and $\mathbf{J}_i = (\mathbf{J}_{i1},..,\mathbf{J}_{iN})$ is the vector of input-to-hidden weights for the i-th hidden node in the student network. Note that the number of hidden nodes in the student network K does not have to equal the number of hidden nodes in the teacher network M.

An input pattern is $\xi^\mu = (\xi_1^\mu,...,\xi_N^\mu)$ where μ is the current input. The pattern output by the teacher network in response to ξ^μ is ζ^μ; training example μ is therefore the input-output pair (ξ^μ, ζ^μ). The activation of the hidden units given an input pattern ξ^μ is $\mathbf{x}_i = \mathbf{J}_i \cdot \xi^\mu$. Similarly, the activation of hidden units in the teacher network given ξ^μ is $\mathbf{y}_n = \mathbf{B}_n \cdot \xi^\mu$. The output from a student network is

$$\sigma(\mathbf{J}, \xi) = \sum_{i=1}^{K} g(\mathbf{x}_i) \tag{2.1}$$

where $g(\mathbf{x}_i)$ is the activation function of hidden unit $\mathbf{J}_i$.

2.2.1 Input data

For this analysis, inputs must be i.i.d. samples from a Gaussian variable $N(0,1)$. Input units ξ are also assumed to be uncorrelated (ie ξ_i is uncorrelated with ξ_j, $i \neq j$) with zero mean and unit variance (although this is not true for many examples of real data), and the maths assumes an infinite input size N, although the analysis is still effective for finite N [3].

2.2.2 Activation functions

Although the input-to-hidden activation function $g(\mathbf{x}_i)$ is only restricted to being differentiable, the erf function, or more specifically $g(x) = \mathrm{erf}(x/\sqrt{2}) = \frac{2}{\sqrt{2\pi}} \int_0^x e^{-t^2/2} dt$ has been used in this section.

2.2.3 Generalisation error

The generalisation error of a soft committee network with erf hidden unit activations and normalised inputs has been calculated in [24] in terms of the order parameters:

$$\epsilon_g(\mathbf{J}) = \frac{1}{\pi}\Big[\sum_{ik}\arcsin\left(\frac{\mathbf{Q}_{ik}}{\sqrt{1+\mathbf{Q}_{ii}}\sqrt{1+\mathbf{Q}_{kk}}}\right)+\sum_{nm}\arcsin\left(\frac{\mathbf{T}_{nm}}{\sqrt{1+\mathbf{T}_{nn}}\sqrt{1+\mathbf{T}_{mm}}}\right) + \sum_{in}\arcsin\left(\frac{\mathbf{R}_{in}}{\sqrt{1+\mathbf{Q}_{ii}}\sqrt{1+\mathbf{T}_{nn}}}\right)\Big] \tag{2.2}$$

2.3 Analysis of gradient descent algorithms

It is instructive when looking at the evolution of order parameters for natural gradient descent to have something to compare its speed and complexity against. The obvious candidates are gradient descent (already analysed in [24]) and the matrix momentum form of Newton's method (second-order, fast and analysed in [19][17]). The calculations used are similar for the various learning algorithms, and are shown for natural gradient descent in appendix (B).

For the purposes of analysis, it is assumed that all data is generated by another soft committee machine: to avoid confusion, this is called the 'teacher' network and the network that is being adapted is its 'student'. Although the student and teacher networks have the same structure and inputs, they do not necessarily have the same number of hidden nodes. For all forms of learning algorithm shown here, the network error is assumed to be quadratic and the error gradient $\nabla_{\mathbf{J}}\epsilon_{\mathbf{J}}(\xi^\mu,\zeta^\mu)$ is therefore

$$\nabla_{\mathbf{J}}\epsilon_{\mathbf{J}}(\xi^\mu,\zeta^\mu) = \frac{\partial}{\partial \mathbf{J}_i}\left(\frac{1}{2}\left(\sum_{n=1}^{M} g(\mathbf{B}_n\cdot\xi^\mu)-\sum_{j=1}^{K} g(\mathbf{J}_j\cdot\xi^\mu)\right)^2\right) = -\delta_i^\mu\xi^\mu \tag{2.3}$$

where

$$\delta_i^\mu = g'(\mathbf{J}_i\cdot\xi^\mu)\left(\sum_{n=1}^{M} g(\mathbf{B}_n\cdot\xi^\mu)-\sum_{j=1}^{K} g(\mathbf{J}_j\cdot\xi^\mu)\right) \tag{2.4}$$

is the backpropagation delta function.

2.3.1 Gradient descent

Saad and Solla [24] develop update equations for a soft committee network using gradient descent. The update for a student hidden unit $\mathbf{J}_i$ in a soft committee network using this learning algorithm, the learning rate η scaled by the input size N and a quadratic error function $\epsilon_{\mathbf{J}}(\xi,\zeta)$ as given in equation 1.1 is [24]:

$$\mathbf{J}_i^{\mu+1} = \mathbf{J}_i^\mu + \frac{\eta}{N}\delta_i^\mu\xi^\mu \tag{2.5}$$

The order parameters for gradient descent are the overlaps between the student and teacher weight vectors $\mathbf{J}$ and $\mathbf{B}$: these are $\mathbf{Q}_{ik} = \mathbf{J}_i \cdot \mathbf{J}_k$, $\mathbf{R}_{in} = \mathbf{J}_i \cdot \mathbf{B}_n$ and $\mathbf{T}_{nm} = \mathbf{B}_n \cdot \mathbf{B}_m$. Since the teacher network is constant, $\mathbf{T}$ does not evolve and its derivative $\frac{d\mathbf{T}_{nm}}{d\alpha}$ is zero. The update equations for the other order parameters $\mathbf{Q}$ and $\mathbf{R}$ are:

$$\begin{aligned}
\frac{d\mathbf{Q}_{ik}}{d\alpha} &= \eta \langle \delta_i \mathbf{x}_k \rangle + \eta \langle \delta_k \mathbf{x}_i \rangle + \eta^2 \langle \delta_i \delta_k \rangle \\
\frac{d\mathbf{R}_{in}}{d\alpha} &= \eta \langle \delta_i \mathbf{y}_n \rangle
\end{aligned}$$

2.3.2 Matrix momentum version of Newton's method

For matrix momentum, it is convenient to define a new variable $\mathbf{\Omega}_i^\mu = N(\mathbf{J}_i^\mu - \mathbf{J}_i^{\mu-1})$. The order parameters are now the overlaps between $\mathbf{J}$, $\mathbf{B}$ and $\mathbf{\Omega}$, i.e. $\mathbf{Q}$, $\mathbf{R}$, $\mathbf{T}$ and $\mathbf{C}_{ik} = \mathbf{\Omega}_i \cdot \mathbf{\Omega}_k$, $\mathbf{D}_{in} = \mathbf{\Omega}_i \cdot \mathbf{B}_n$ and $\mathbf{E}_{ik} = \mathbf{J}_i \cdot \mathbf{\Omega}_k$. Unlike the overlaps between $\mathbf{J}$ and $\mathbf{B}$, the overlaps with $\mathbf{\Omega}$ do not appear to have a direct physical meaning.

With $\mathbf{J}$ and $\mathbf{\Omega}$ both being updated simultaneously, it is important that contributions towards updating the same quantity occur on the same time scale. This is achieved by scaling both η and β. For the Newton's method calculations, it is sufficient to scale both η and $1-\beta$ by N (i.e. $\beta = 1 - \gamma/N$) as described in [14]. For the $\mathbf{J}$, $\mathbf{\Omega}$ updates given in equations 1.21 and 1.22 with $\beta = \mathbf{I} - \frac{k\mathbf{H}}{N}$, $\eta = \frac{k\eta_\alpha}{N}$ and a soft committee machine (i.e. $\nabla_{\mathbf{J}} \epsilon_{\mathbf{J}}(\mathbf{xi}^\mu, \zeta^\mu) = -\delta_i^\mu \xi^\mu$), the order parameter evolutions are [18]:

$$\begin{aligned}
\frac{d\mathbf{Q}_{ik}}{d\alpha} &= \mathbf{E}_{ik} + \mathbf{E}_{ki} \\
\frac{d\mathbf{R}_{in}}{d\alpha} &= \mathbf{D}_{in} \\
\frac{d\mathbf{C}_{ik}}{d\alpha} &= k\eta_\alpha \langle \delta_i z_k + \delta_k z_i \rangle + k^2 \eta_\alpha^2 \langle \delta_i \delta_k \rangle - k \sum_m (\mathbf{c}_{im}\mathbf{D}_{km} + \mathbf{c}_{km}\mathbf{D}_{im}) \\
&\quad -k \sum_j (\mathbf{a}_{ij}\mathbf{C}_{kj} + \mathbf{a}_{kj}\mathbf{C}_{ij} + \mathbf{b}_{ij}\mathbf{E}_{jk} + \mathbf{b}_{kj}\mathbf{E}_{ji}) \\
\frac{d\mathbf{D}_{ik}}{d\alpha} &= k\eta_\alpha \langle \delta_i \mathbf{y}_n \rangle - k \sum_j (\mathbf{a}_{ij}\mathbf{D}_{jn} + \mathbf{b}_{ij}\mathbf{R}_{jn}) - k \sum_m \mathbf{c}_{im}\mathbf{T}_{nm} \\
\frac{d\mathbf{E}_{ik}}{d\alpha} &= \mathbf{C}_{ik} + k\eta_\alpha \langle \delta_k \mathbf{x}_i \rangle - k \sum_j (\mathbf{a}_{kj}\mathbf{E}_{ij} + \mathbf{b}_{kj}\mathbf{Q}_{ij}) - k \sum_m \mathbf{c}_{km}\mathbf{R}_{im}
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{a}_{ij} &= (1 + \delta_{ij}) \frac{\partial \epsilon_g(\mathbf{J})}{\partial \mathbf{Q}_{ij}} \\
\mathbf{b}_{ij} &= (1 + \delta_{ij}) \left[\sum_{lk} (1 + \delta_{lk}) \mathbf{E}_{lk} \frac{\partial^2 \epsilon_g(\mathbf{J})}{\partial \mathbf{Q}_{ij} \partial \mathbf{Q}_{kl}} + \sum_{kn} \mathbf{D}_{kn} \frac{\partial^2 \epsilon_g(\mathbf{J})}{\partial \mathbf{Q}_{ij} \partial \mathbf{R}_{kn}} \right]
\end{aligned}$$

$$\mathbf{c}_{in} = \sum_{lk}(1+\delta_{lk})\mathbf{E}_{lk}\frac{\partial^2\epsilon_g(\mathbf{J})}{\partial\mathbf{R}_{in}\partial\mathbf{Q}_{kl}} + \sum_{km}\mathbf{D}_{km}\frac{\partial^2\epsilon_g(\mathbf{J})}{\partial\mathbf{R}_{in}\partial\mathbf{R}_{km}}$$

2.3.3 MM-NGD with the full Fisher information matrix

As with the matrix momentum version of Newton's method, the order parameters for MM-NGD are the overlaps between the variables $\mathbf{J}$, $\mathbf{B}$ and $\mathbf{\Omega}$: $\mathbf{Q}_{ik} = \mathbf{J}_i \cdot \mathbf{J}_k$, $\mathbf{R}_{in} = \mathbf{J}_i \cdot \mathbf{B}_n$ $\mathbf{T}_{nm} = \mathbf{B}_n \cdot \mathbf{B}_m$, $\mathbf{C}_{ik} = \mathbf{\Omega}_i \cdot \mathbf{\Omega}_k$, $\mathbf{D}_{in} = \mathbf{\Omega}_i \cdot \mathbf{B}_n$ and $\mathbf{E}_{ik} = \mathbf{J}_i \cdot \mathbf{\Omega}_k$.

Since this is a toy problem, with the input distribution and the model both known, we can actually calculate the Fisher information matrix directly. For a soft committee network and a quadratic error function $\epsilon_{\mathbf{J}}(\xi, \zeta_{\mathbf{J}})$, the exact calculation for the Fisher information matrix given in equation (1.29) reduces to

$$G(\mathbf{J}) = [G_{i\alpha,k\beta}] = \left[\frac{1}{\sigma_m^4}\left\langle \frac{\partial\epsilon_{\mathbf{J}}(\xi,\zeta_{\mathbf{J}})}{\partial\mathbf{J}_{i\alpha}}\frac{\partial\epsilon_{\mathbf{J}}(\xi,\zeta_{\mathbf{J}})}{\partial\mathbf{J}_{k\beta}}\right\rangle_{\{\xi\}}\right] \tag{2.6}$$

where i, k are indices over the hidden units, α, β are indices over the input elements and σ_m^2 is the variance of the teacher output noise ρ. This is a matrix consisting of $K \times K$ blocks which each contain $N \times N$ values. Further manipulation of this equation (this is given in appendix C) gives

$$\mathbf{G}_{ik} = \frac{1}{\sigma_m^2} < A_{ik}(\xi) >_{\{\xi\}} \tag{2.7}$$

where $\mathbf{A}_{ik}$ is

$$A_{ik} = \frac{2}{\pi\sqrt{\Delta_k}}\Big[\mathbf{I} - \frac{1}{\Delta_k}((1+\mathbf{Q}_{kk})\mathbf{J}_i\mathbf{J}_i^T + (1+\mathbf{Q}_{ii})\mathbf{J}_k\mathbf{J}_k^T - \mathbf{Q}_{ik}(\mathbf{J}_i\mathbf{J}_k^T + \mathbf{J}_k\mathbf{J}_i^T))\Big] \tag{2.8}$$

I is the identity matrix and

$$\Delta_k = (1+\mathbf{Q}_{ii})(1+\mathbf{Q}_{kk}) - \mathbf{Q}_{ik}^2 \tag{2.9}$$

For $\mathbf{J}$, $\mathbf{\Omega}$ update equations 1.30 and 1.31, with $\beta = I - \frac{k\mathbf{G}(\mathbf{J})}{N}$ and $\eta = \frac{k\eta_\alpha}{N}$, the equations of motion for the order parameters are:

$$\frac{d\mathbf{Q}_{ik}}{d\alpha} = \mathbf{E}_{ik} + \mathbf{E}_{ki} \tag{2.10}$$

$$\frac{d\mathbf{R}_{in}}{d\alpha} = \mathbf{D}_{in} \tag{2.11}$$

$$\begin{aligned}\frac{d\mathbf{C}_{ik}}{d\alpha} = &-\sum_j \frac{2k}{\pi\sqrt{\Delta_{kj}}}\Big[\mathbf{C}_{ij} - \frac{1}{\Delta_{kj}}((1+\mathbf{Q}_{jj})\mathbf{E}_{kj}\mathbf{E}_{ki} + (1+\mathbf{Q}_{kk})\mathbf{E}_{jj}\mathbf{E}_{ji} \\ &-\mathbf{Q}_{kj}(\mathbf{E}_{jj}\mathbf{E}_{ki}+\mathbf{E}_{kj}\mathbf{E}_{ji}))\Big] \\ &-\sum_j \frac{2k}{\pi\sqrt{\Delta_{ij}}}\Big[\mathbf{C}_{jk} - \frac{1}{\Delta_{ij}}((1+\mathbf{Q}_{jj})\mathbf{E}_{ij}\mathbf{E}_{ik} + (1+\mathbf{Q}_{ii})\mathbf{E}_{jj}\mathbf{E}_{jk}\end{aligned}$$

$$-\mathbf{Q}_{ij}(\mathbf{E}_{jj}\mathbf{E}_{ik}+\mathbf{E}_{ij}\mathbf{E}_{jk}))\Big] + k\eta\,\langle\delta_k z_i + \delta_i z_k\rangle + k^2\eta^2\,\langle\delta_i\delta_k\rangle \tag{2.12}$$

$$\frac{d\mathbf{D}_{in}}{d\alpha} = -\sum_j \frac{2k}{\pi\sqrt{\mathbf{\Delta}_{ij}}}\Big[\mathbf{D}_{jn} - \frac{1}{\mathbf{\Delta}_{ij}}((1+\mathbf{Q}_{jj})\mathbf{E}_{ij}\mathbf{R}_{in} + (1+\mathbf{Q}_{ii})\mathbf{E}_{jj}\mathbf{R}_{jn} - \mathbf{Q}_{ij}(\mathbf{E}_{jj}\mathbf{R}_{in}+\mathbf{E}_{ij}\mathbf{R}_{jn}))\Big] + k\eta\,\langle\delta_i\mathbf{y}_n\rangle \tag{2.13}$$

$$\frac{d\mathbf{E}_{ik}}{d\alpha} = \mathbf{C}_{ik} - \sum_j \frac{2k}{\pi\sqrt{\mathbf{\Delta}_{kj}}}\Big[\mathbf{E}_{ij} - \frac{1}{\mathbf{\Delta}_{kj}}((1+\mathbf{Q}_{jj})\mathbf{E}_{kj}\mathbf{Q}_{ik} + (1+\mathbf{Q}_{kk})\mathbf{E}_{jj}\mathbf{Q}_{ij} - \mathbf{Q}_{kj}(\mathbf{E}_{jj}\mathbf{Q}_{ik}+\mathbf{E}_{kj}\mathbf{Q}_{ij}))\Big] + k\eta\,\langle\delta_k\mathbf{x}_i\rangle \tag{2.14}$$

The calculations for these are given in appendix B.

There is a more compact form of these equations. If the update equations for $\mathbf{C}$, $\mathbf{D}$ and $\mathbf{E}$ are written as

$$\frac{d\mathbf{C}_{ik}}{d\alpha} = -k\mathbf{\Omega}_i^T\sum_j A_{kj}\mathbf{\Omega}_j - k\sum_j (A_{ij}\mathbf{\Omega}_j)^T\mathbf{\Omega}_k + k\eta\langle\delta_k z_i + \delta_i z_k\rangle + k^2\eta^2\langle\delta_i\delta_k\rangle \tag{2.15}$$

$$\frac{d\mathbf{D}_{in}}{d\alpha} = -k\sum_j (A_{ij}\mathbf{\Omega}_j)^T\mathbf{B}_n + k\eta\,\langle\delta_i\mathbf{y}_n\rangle \tag{2.16}$$

$$\frac{d\mathbf{E}_{ik}}{d\alpha} = \mathbf{C}_{ik} - k\mathbf{J}_i^T\sum_j A_{kj}\mathbf{\Omega}_j + k\eta\,\langle\delta_k\mathbf{x}_i\rangle \tag{2.17}$$

then the sum $\sum_{ij} A_{ij}\mathbf{\Omega}_j$ can be calculated from

$$\sum_j A_{ij}\mathbf{\Omega}_j = \frac{2}{\pi}\sum_j \frac{1}{\sqrt{\mathbf{\Delta}_{ij}}}\Big[\mathbf{\Omega}_j - \frac{1}{\mathbf{\Delta}_{ij}}((1+\mathbf{Q}_{jj})\mathbf{E}_{ij}\mathbf{J}_i + (1+\mathbf{Q}_{ii})\mathbf{E}_{jj}\mathbf{J}_j - \mathbf{Q}_{ij}(\mathbf{E}_{jj}\mathbf{J}_i + \mathbf{E}_{ij}\mathbf{J}_j))\Big] \tag{2.18}$$

and substituted in.

2.3.4 MM-NGD with single-input Fisher information

When the network being analysed is very large, it may not be possible or practical to calculate the full Fisher information matrix for each input to it. A single-input approximation to the Fisher information matrix which only uses one example is

$$\mathbf{G}_{ik} = \frac{1}{\sigma_m^2}A_{ik}(\xi) \tag{2.19}$$

where $A_{ik}(\xi)$ is defined as before. The update equations for this have been calculated by Scarpetta [25] and are

$$\frac{d\mathbf{Q}_{ik}}{d\alpha} = \mathbf{E}_{ik} + \mathbf{E}_{ki} \tag{2.20}$$

$$\frac{d\mathbf{R}_{in}}{d\alpha} = \mathbf{D}_{in} \tag{2.21}$$

$$\frac{d\mathbf{C}_{ik}}{d\alpha} = k\langle(\eta_\alpha\delta_i - \phi_i)z_k + (\eta_\alpha\delta_k - \phi_k)z_i\rangle + k^2\langle(\eta_\alpha\delta_i - \phi_i)(\eta_\alpha\delta_k - \phi_k)\rangle \quad (2.22)$$

$$\frac{d\mathbf{D}_{in}}{d\alpha} = k\langle(\eta_\alpha\delta_i - \phi_i)\mathbf{y}_n\rangle \quad (2.23)$$

$$\frac{d\mathbf{E}_{ik}}{d\alpha} = \mathbf{C}_{ik} + k\langle(\eta_\alpha\delta_k - \phi_k)\mathbf{x}_i\rangle \quad (2.24)$$

$$\text{where } \phi_i = g'(\mathbf{x}_i)\sum_j z_j g'(\mathbf{x}_j)$$

Although the behaviour of MM-NGD with single-input Fisher information matrix approximations is important for predicting the behaviour of MM-NGD learning when it is difficult to evaluate the Fisher information matrix, its analysis does not form part of this report: preliminary results and discussion of this topic can be found in [25].

Chapter 3

Numerical results

This chapter shows some of the characteristics of the order parameter and generalisation error evolutions calculated in the last chapter, calculated for soft committee networks with two-hidden-unit student networks, two-hidden-unit teacher networks and erf activation functions. Emphasis has been placed on the behaviour of generalisation error over time for several values of the learning parameters η and k and teacher network noise σ_m^2.

3.1 Learning parameters

The training parameters of the MM-NGD learning algorithm are the learning rate η and the parameter k, representing the balance between gradient descent and momentum. The noise variance of the teacher network outputs is termed σ_m^2. Other variables that can be adjusted are the numbers of teacher and student hidden nodes, and whether the learning rate η is annealed (gradually reduced over time).

Another factor that affects the learning algorithm outputs, if we have noise on our inputs, is the point at which learning rate annealing is started. It should be noted that all times are described in terms of α, and that the interplay between the values of α and the number of datasteps used for each stage of learning is also significant to the efficiency of the learning algorithm.

3.2 Initial conditions

All order parameters are initially zero or sampled from Gaussian distributions:

$$\mathbf{Q}_{ik} \sim N(0, 0.5), i = k; \ \mathbf{Q}_{ik} \sim N(0, 0.001), i \neq k$$

$$
\begin{aligned}
\mathbf{R}_{in} &\sim N(0, 0.001) \\
\mathbf{C}_{ik} &\sim N(0, 0.001), i = k;\ \mathbf{C}_{ik} = 0, i \neq k \\
\mathbf{D}_{in} &= 0 \\
\mathbf{E}_{ik} &= 0 \\
\mathbf{T}_{nm} &= \mathrm{t_{init}}, n = m;\ \mathbf{T}_{nm} = 0, n \neq m
\end{aligned}
$$

where the teacher covariance matrix is set to the same values as T_{nm} and $\mathrm{t_{init}}$ is set to 1 if there is no teacher noise σ_m^2, and 0.5 for a teacher noise of 0.01. All the order parameter derivatives were initially set to zero. With these initial values, the networks were initially symmetric. The same random seed was used for each algorithm so that when their generalisation errors are compared in this chapter, their initial conditions are identical.

3.3 Phases of learning

In these output plots, the generalisation errors follow a characteristic curve in time. First, an initial phase where the generalisation error drops rapidly to a value. Then, a symmetric phase where the generalisation error stays at that value for some time (this is known as the symmetric plateau), then a drop to an asymptotic phase during which the error reduces gradually. In the plots, the initial and symmetric phases are shown for ϵ_g against α; the asymptotic phases are shown on a log-log scale.

3.4 Comparing standard, natural and MM-NGD learning algorithms

Figure 3.1 shows the expected generalisation errors of optimal (with respect to the learning rate) standard gradient descent [23], optimal natural gradient descent and MM-NGD gradient descent algorithms, with the optimal learning rate ($\eta = 0.144855$) for the example network and $k = 10$ in the MM-NGD algorithm (note that α is written as alpha in these graphs).

The natural gradient descent algorithm reaches a plateau less quickly that the standard gradient descent algorithm as it reweighs the gradient in all directions reducing the strong difference between gradients in standard gradient descent which drives the system very quickly towards the symmetric fixed point. The plateau height for the standard gradient

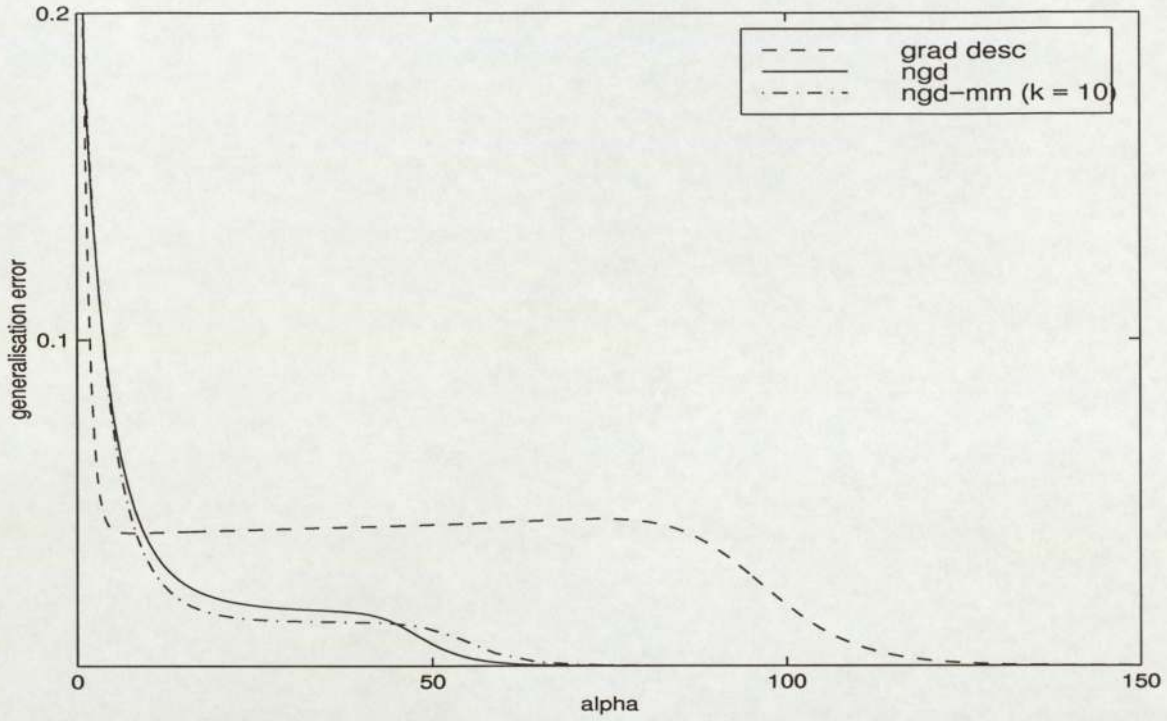


Figure 3.1: MM-NGD vs NGD and optimal gradient descent

descent is much higher than the natural gradient plateau: this occurs because although gradient descent moves quickly towards a minimum in the most significant gradient direction, it does not move much in the other gradient directions and will therefore be at a higher level in those smaller directions when it reaches the symmetric phase.

The difference between the MM-NGD and natural gradient plateaux is discussed in the next section.

3.4.1 MM-NGD without teacher noise

Figure 3.2 shows the symmetric-phase performance of natural gradient descent algorithms on a two-hidden-unit network ($K = M = 2$) when there is no noise on the teacher network outputs ($\sigma_m^2 = 0$). In this plot, all algorithms have a learning rate of $\eta = 0.15$. The solid line is the generalisation error of the natural gradient descent (NGD) algorithm, and the dotted lines (from right to left) are MM-NGD with k set to 0.5, 1.4, 2.1 and 10.

As the value of k is reduced, the length of the plateau in the MM-NGD error curve approaches the plateau length of the NGD curve. Experiments with larger values of k (up to K=100) show that the MM-NGD curve approaches the NGD curve without reaching it, and with diminishing returns for larger values. Since the number of datapoints needed

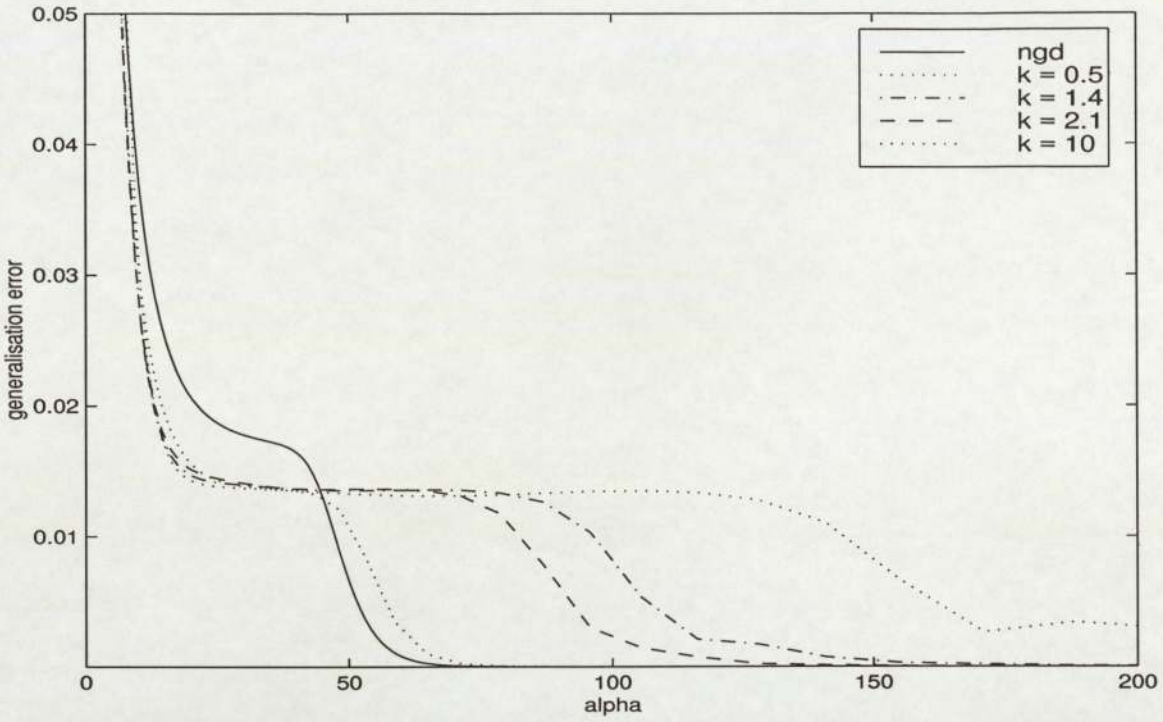


Figure 3.2: Symmetric phase behaviour of MM-NGD (teacher noise=0)

to calculate these error curves increases with k and $k = 20$ is not visibly very different from $k = 50$ or $k = 100$, none of the plots shown here use a larger value than 20 for k.

If MM-NGD and NGD are equivalent for large values of k, then the curve for large k should be very similar to that for NGD. This is not the case: although the plateaux lengths are similar, the MM-NGD and NGD algorithms have different plateau heights. It is not certain why this should be the case: the plateau height should be determined by the η^2 terms in the order parameter update equations but experiments with these terms have proved inconclusive.

3.4.2 MM-NGD with teacher noise

Figure 3.3 shows the asymptotic behaviour of NGD and MM-NGD with teacher noise variance set to $\sigma_m^2 = 0.01$, the teacher covariance matrix T set to 0.5 on their diagonals ($T_{mn} = \delta_{mn}0.5$ where δ_{mn} is the Kronecker delta). The upper dotted line is the (theoretical) optimal gradient descent bound: gradient descent learning is much worse than this, and outputs from this algorithm have not been shown.

The lower dotted line is the Cramer-Rao lower bound: this shows a theoretical limit

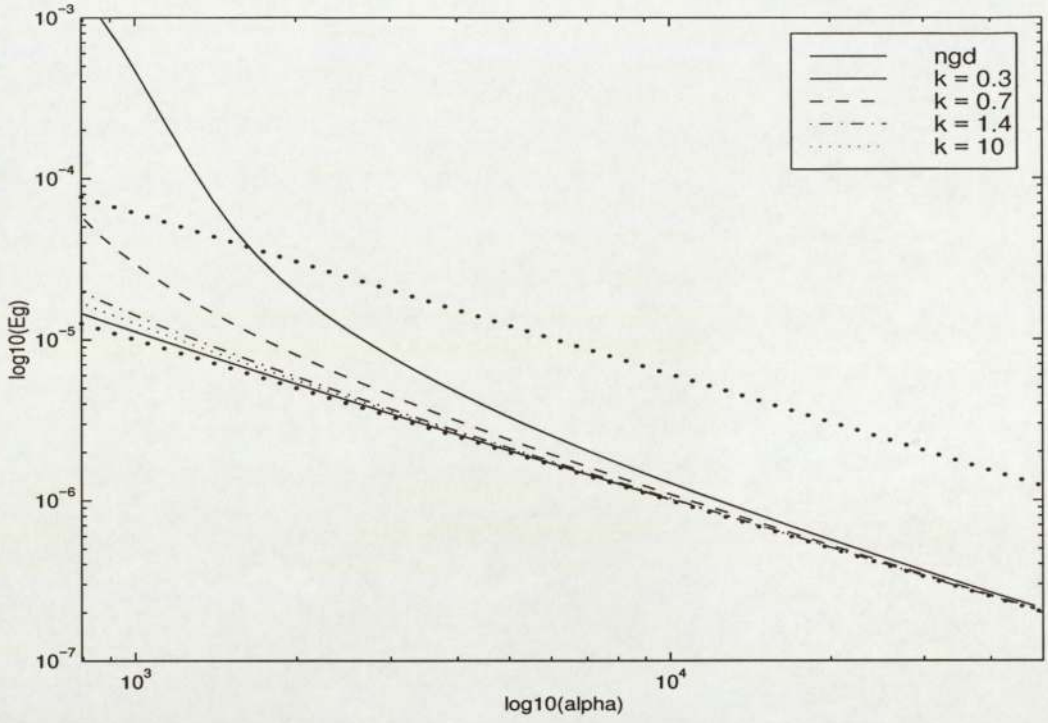


Figure 3.3: Asymptotic behaviour of MM-NGD for noisy teacher ($\sigma_m^2 = 0.01$)

on the speed at which all algorithms can learn, and is defined as

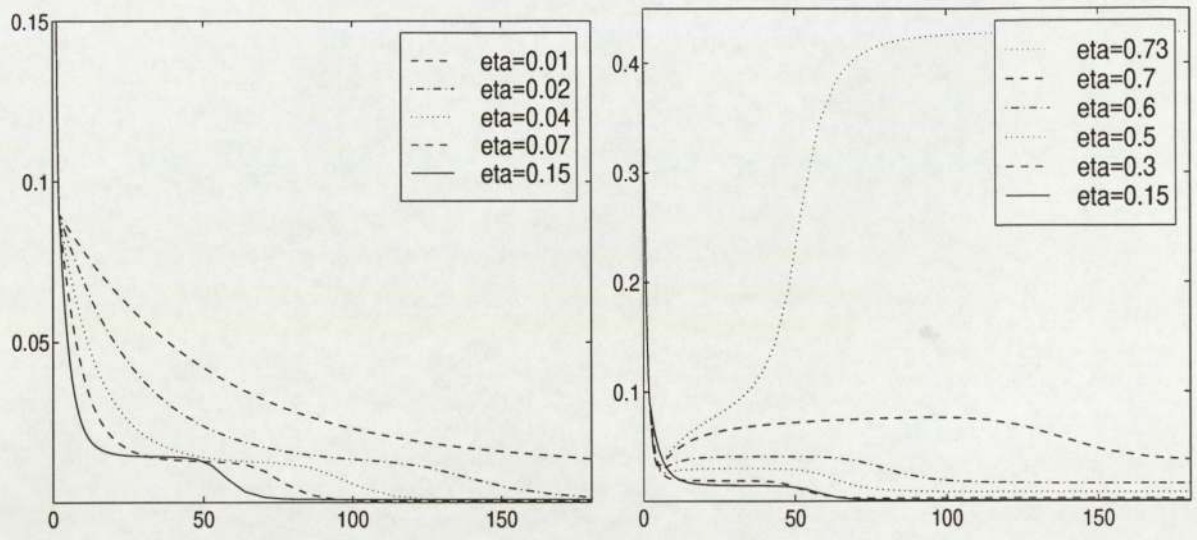
$$V \geq \frac{1}{M} G(\mathbf{J})^{-1} \tag{3.1}$$

where $G(\mathbf{J})$ is the Fisher information matrix and M is the number of examples available.

Since natural gradient descent algorithms are asymptotically optimal, they should be expected to approach the Cramer-Rao bound: this is seen in the plots, where the lower solid line is the generalisation error for NGD, and the other learning curves are for MM-NGD and (from top to bottom) $k = 0.3$, 0.7, 1.4 and 10.

3.5 Varying other parameters

The results shown in this chapter have all been for optimal or near-optimal values of the learning rate η and different values of the momentum balance parameter k. This reflects our interest in the momentum parameters and the fact that the behaviour of generalisation error with different values of η has been well studied elsewhere. However, before this algorithm is applied to real data, it is interesting and perhaps prudent to note what happens to the generalisation error evolution when we vary η: since online learning is sensitive to parameter settings, this might provide some clues to what might be amiss



(a) Smaller than optimal η values

(b) Larger than optimal η values

Figure 3.4: Generalisation error evolution for various η values

with them.

Figure 3.4 shows the behaviour of mm-ngd generalisation error for several values of η ($\eta = 0.01$, 0.02, 0.04, 0.07, 0.15 and $\eta = 0.73$, 0.7, 0.6, 0.5, 0.3: note that η is written as eta in the graphs) with $k = 8$, σ_m^2=0.01, and $t_{init} = 1$. For decreasing values of η below the optimum value $\eta = 0.144885$ ($\eta = 0.15$ is used as an approximation to this in figure 3.4), the initial phase before a plateau is reached is longer, and the symmetric plateau height increases until the initial phase and symmetric plateau appear to be part of one smooth curve. For increasing values of η above the optimum value, the initial phase decreases in time, and the symmetric plateau height increases until at about $\eta = 0.5$, learning is not possible.

Also of interest is the behaviour of generalisation error with different combinations of values of η and k, but this has not been studied here.

Chapter 4

Using MM-NGD on real data

The matrix momentum form of natural gradient descent works well with toy examples, but its potential value is in its application to real data. To this end, the datasets used for matrix momentum with standard gradient descent have also been tried with MM-NGD.

The networks used here are multilayer perceptrons with linear hidden to output connections and tanh hidden-layer activation functions. The learning algorithm used was online learning, using sampling of the input data with replacement. Code was written in both Matlab (to fit into the Netlab framework) and C++.

4.1 MM-NGD for MLPs

The parameter update equation 1.30 with $\beta = I - \frac{k\mathbf{G}(\Theta)}{N}$ and $\eta = \frac{k\eta_\alpha}{N}$ was used here. This gives a parameter update equation of

$$\Theta_i^{\mu+1} = \Theta_i^{\mu} - \frac{k}{N^2}\nabla_\Theta \epsilon_{\mathbf{J}}(\xi,\zeta) + (\mathbf{I} - \frac{k\mathbf{G}(\Theta)}{N\eta_\alpha})(\Theta_i^{\mu} - \Theta_i^{\mu-1}) \tag{4.1}$$

where Θ is the set of network parameters $\{\mathbf{J_{ij}}, b_j, a_{jo}, c_o\}$ for a multilayer perceptron with input-to-hidden weights J_{ij}, hidden unit biases b_j, hidden-to-output weights a_{jo} and output unit bias c_o for $1 \leq i \leq N$, $1 \leq j \leq K$, $1 \leq o \leq W$, N is the number of input nodes, K the number of hidden nodes, and W the number of output nodes. Note that the notation is changed slightly since we are no longer just updating the input-to-hidden layer weights $\mathbf{J}$.

4.1.1 Calculating the Fisher information matrix

The empirical Fisher information matrix $\mathbf{G}(\Theta)$ for a multilayer perceptron with parameters Θ, network output σ, expected output ζ and a loss function for the network of

$$L(\zeta|\xi;\mathbf{J}) = \frac{1}{2}(\sigma - \zeta)^2 \tag{4.2}$$

is $\mathbf{G}(\Theta) = \mathbf{A}(\Theta)$, where

$$\mathbf{A}(\Theta) = \left\langle \left[\frac{\partial L}{\partial J_{ij}}, \frac{\partial L}{\partial b_j}, \frac{\partial L}{\partial a_{jo}}, \frac{\partial L}{\partial c_o}\right]^T \left[\frac{\partial L}{\partial J_{ij}}, \frac{\partial L}{\partial b_j}, \frac{\partial L}{\partial a_{jo}}, \frac{\partial L}{\partial c_o}\right] \right\rangle_{\{\xi,\zeta\}} \tag{4.3}$$

The components of $\mathbf{A}(\Theta)$ are calculated in appendix C and are

$$\frac{\partial L_o^{d\mu}}{\partial J_{ij}^{\mu}} = a_{jo}^{\mu} g'(\sum_l J_{lj}{}^{\mu} \xi_l^d + b_j^{\mu}) \xi_i^d \tag{4.4}$$

$$\frac{\partial L_o^{d\mu}}{\partial b_j^{\mu}} = a_{jo}^{\mu} g'(\sum_l J_{lj}^{\mu} \xi_l^d + b_j^{\mu}) \tag{4.5}$$

$$\frac{\partial L_o^{d\mu}}{\partial a_{jo}^{\mu}} = g(\sum_l J_{lj}^{\mu} \xi_l^d + b_j^{\mu}) \delta_{ko} \tag{4.6}$$

$$\frac{\partial L_o^{d\mu}}{\partial c_k^{\mu}} = \delta_{ko} \tag{4.7}$$

where o is an index over output nodes, i is an index over input nodes, j is an index over hidden nodes, $g(x)$ is the hidden-layer activation function for the network and $g'(x)$ is its derivative, δ_{ij} (double index) is the Kronecker delta function, d is an index over the examples that are available to calculate the Fisher information matrix with, and μ indexes the current input example.

4.1.2 Approximating the Fisher information matrix

The Fisher information matrix is calculated by averaging over all the input data. One approximation for the matrix is to only calculate it for a single input (usually the current input). The equations for each input can be reduced for special cases of the input distribution, but, unlike the Hessian matrix, no other approximation or simplifications appear to have been developed.

4.2 Small datasets : the Iris dataset

MM-NGD was run on some small datasets to check for obvious errors and test the effects of varying the learning parameters η and k before a large dataset with long learning-times was used (some variations of σ_m were also tried).

One dataset chosen was Fisher's iris plants database [6] from the UCI repository[12], because it is well known, easily available and fairly small. The iris dataset consists of four measurements and a classification for 50 examples of each of three different classes of iris plant, with no missing data values.

The network architecture used is four input units, five hidden units and one output unit. This does not seem to be the optimal architecture for the problem (better results were obtained with seven or eight hidden units) but provides a network with just 31 parameters to study. The Fisher information for the Iris network and all other small (less than 1000 parameters) networks was calculated in full from all the examples in the dataset.

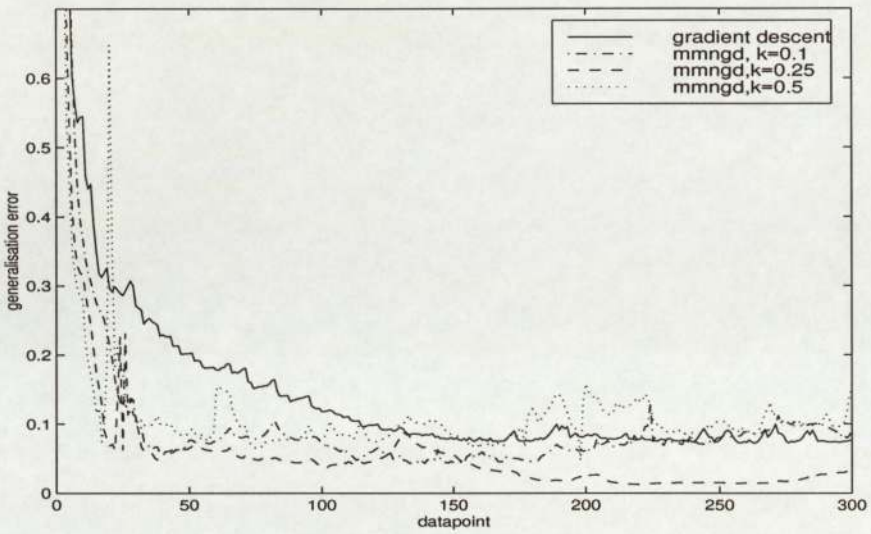


Figure 4.1: Iris data, $\eta_\alpha = 0.16$, various k

Figure 4.1 shows generalisation errors ϵ_g for the iris data with a learning rate $\eta_\alpha = 0.16$, for gradient descent (solid line) and mmngd with $k = 0.1$ (dash-dotted line), $k = 0.25$ (dashed line), $k = 0.5$ (dotted line). Here, the values of ϵ_g for mmngd either drop to a roughly constant value of about 0.1 which is below the gradient descent curve and remain near that value ($k = 0.1$, $k = 0.5$) or continue to drop to lower values ($k = 0.25$). After presentation of most of the dataset, ϵ_g for the gradient descent algorithm is a roughly constant value of 0.1. Another features to note is that the initial descent to $\epsilon_g = 0.1$ is faster for increasing values of k. The behaviour of mmngd with $k = 0.5$ may be explained by the fact that the algorithm is becoming unstable at this point: for values of $k = 1$ and above (not shown on this plot), ϵ_g drops rapidly for the first few datapoints then rises again and settles on a value near $\epsilon_g = 0.33$. There does not appear to be a symmetric plateau: there are slight kinks in each of the mmngd plots at ϵ_g values between 0.25 and

0.35, but they are not significant enough to draw any conclusions.

4.3 Medium-sized datasets : wine classification

The wine recognition dataset is taken from the UCI repository[12]. It contains the quantities of 13 chemical constituents for three types of wine. There are 178 examples and no missing data.

The network architecture used is 13 input units, 13 hidden units and 1 output unit, which gives 196 network parameters.

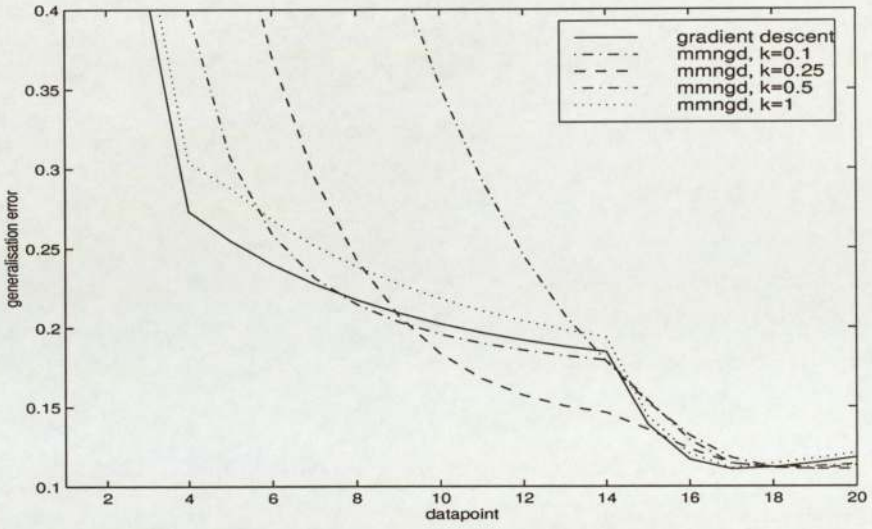


Figure 4.2: Wine data, varying k

Figure 4.2 shows ϵ_g for gradient descent (solid line) and mmngd with $k = 0.1$ (rightmost dot-dashed line), $k = 0.25$ (dashed line), $k = 0.5$ (leftmost dot-dashed line), $k = 1$ (dotted line). For values of k above $k = 1$, ϵ_g is very similar to the $k = 1$ line, with increasing jitter around it until $k = 2$, where the algorithm breaks down and ϵ_g diverges to infinity. For all the lines shown, ϵ_g drops rapidly to a value ($\epsilon_g = 0.27$, datapoint=3 for gradient descent), then drops at a slower rate to a second value ($\epsilon_g = 0.19$, datapoint 14 of gradient descent) and finally drops to a value near $\epsilon_g = 0.11$, and stays near $\epsilon_g = 0.11$ for the rest of training. This appears to be some form of plateau, but the behaviour with k is not as expected: the plateau height increases with increasing values of k and, although the plateau is reached at earlier times (datapoints) for larger values k, it is always left at approximately the same time.

4.3.1 varying η,β definitions

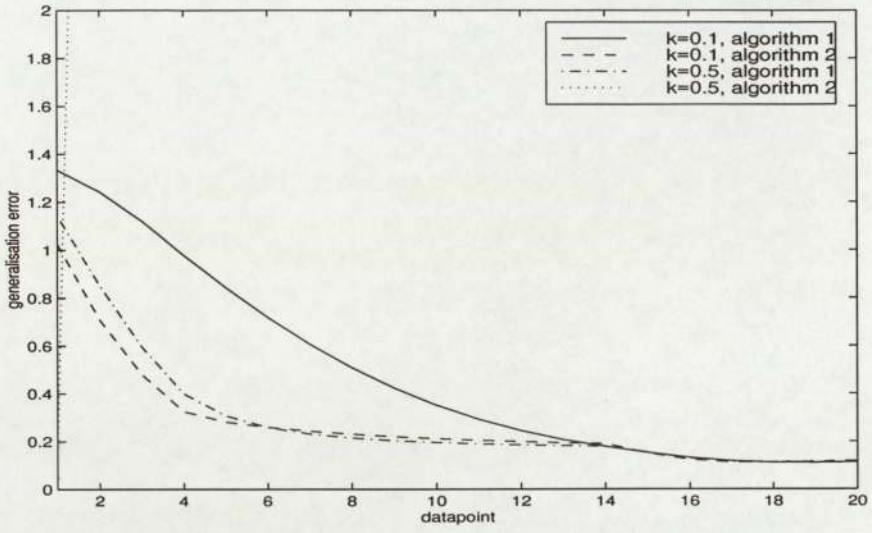


Figure 4.3: Comparing MM-NGD algorithms

Two equations for η and β that both appear to give an equivalent learning rate of $\mathbf{G}(\mathbf{\Theta})^{-1}\eta_\alpha$ are

$$\beta = I - \frac{k\mathbf{G}(\mathbf{\Theta})}{N}; \eta = \frac{k\eta_\alpha}{N} \tag{4.8}$$

and

$$\beta = I - \frac{k\mathbf{G}(\mathbf{\Theta})}{N\eta_\alpha}; \eta = \frac{k}{N} \tag{4.9}$$

These have very different learning behaviours. Figure 4.3 shows generalisation errors for the wine dataset with mm-ngd learning and these two η, β equations and η_α set to 0.13. The four lines shown are k set to 0.1 with equations 4.8 (solid line); k set to 0.1 with equations 4.9 (dashed line); k set to 0.5 with equations 4.8 (dot-dashed line); k set to 0.5 with equations 4.9 (dotted line, just visible at left of plot). It is seen that while the generalisation errors for learning using the first algorithm (equations 4.8 behave as predicted in chapter 3, the behaviour is very different for the second algorithm (equations 4.9) and, for larger values of k, the generalisation error diverges (this is not shown in figure 4.3 since the divergence is rapid and not on the same scale). It is easy to confuse these two sets of β, η equations: all other mm-ngd plots in this document use the first set (equations 4.8).

4.4 Large datasets : speech (phoneme classification) data

MM-NGD as also run on a large dataset to check for practical problems with scale, learning times and resources.

The database used is taken from [14]. It consists of 9000 71-element vectors in a training set and 1000 71-element vectors in a test set, where each vector contains a phoneme class label (1 to 39) and 70 perceptual linear predictive (PLP) coefficients.

The network architecture used in [14] was a standard fully connected feedforward network with 70 input nodes + 1 bias, 70 hidden nodes + 1 bias and 39 output nodes. This has also been used here.

4.4.1 Large Fisher information matrices

The first problem encountered was one of size: for reasonably large networks, the Fisher information matrix can be larger than the available computer memory. In the speech data example, there are just under 8000 parameters (weights and biases) in the network, giving a Fisher matrix with 8000×8000 elements, which was impossible [1] to store. Storage space can be reduced slightly by calculating half (the upper or lower triangle) of the matrix, but 8000×4000 is still large, so the Fisher matrix was created implicitly by storing the multiplication $(\mathbf{I} - \frac{k\mathbf{G}(\Theta)^{-1}}{N\eta_\alpha})(J_i^\mu - J_i^{\mu-1})$ instead of the Fisher matrix.

One question arising from this treatment of the Fisher matrix for large networks is whether a scheme similar to Pearlmutter's algorithm [16] for fast multiplication of the Hessian by a vector could be devised. Examination of [16] suggests that this might be possible, but it has not been attempted here.

4.4.2 Slow learning times

The next problem was speed: although the learning algorithm ran reasonably quickly in single-step mode (only calculating the Fisher information of the last input seen rather than all inputs), it took a very long time to run if the full Fisher matrix was calculated. Results from this dataset have not been shown here because it was impractical to do enough runs of the algorithm to be confident that the algorithm was working or to show the effects of adjusting the learning parameters k, η_α.

[1] $8000 \times 8000 \times 64$ bits = 4 Gbytes is not practical in any normal computer system

Chapter 5

Conclusions

Natural gradient descent learning is algorithmically more efficient than (will reach a small value of ϵ_g faster than) standard gradient descent but is computationally expensive because of the average over all input data and large matrix inversion in its calculation. Matrix momentum can be used to invert the Fisher information matrix used in natural gradient descent, to give a matrix momentum form of natural gradient descent (MM-NGD).

MM-NGD is computationally more efficient (needs less computer operations to calculate) than natural gradient descent and algorithmically more efficient than standard gradient descent algorithms. Natural gradient descent is asymptotically statistically efficient (its generalisation error curve converges close to the Cramer-Rao lower bound on the algorithmic efficiency of learning algorithms) when it uses an exact Fisher information matrix: MM-NGD's generalisation error plots converge close but not exactly to the corresponding natural gradient for large values of the momentum-gradient balancing constant k. It is not known why this is not an exact match, but it may not be significant as we are more concerned with the symmetric plateau length and asymptotic behaviour of the algorithm.

MM-NGD with exact calculations for the Fisher information matrix has been used with real data. It has been found to be computationally very slow for large datasets, spending most of its computation time in the calculation of the Fisher information matrix. No exact algorithms to reduce this caculation time have been found, although it may be possible to adapt fast algorithms for calculating the Hessian matrix to this task. It was also impossible for our (and most current) computer systems to store the Fisher information matrix for the large (8000 parameter) network that was used, forcing the algorithm to calculate the Fisher information matrix implicitly instead.

5.1 Possible extensions to this work

This work has raised many questions about MM-NGD and natural gradient descent with real-world data.

One of the problems with running MM-NGD on large datasets was the long time that the algorithm took. It would seem sensible that to reduce the training time for large datasets, an algorithm similar to Pearlmutter's algorithm for Hessian matrix calculation be found for the Fisher information matrix. Other possibilities are approximations to the Fisher information matrix and single-input or few-input calculations of it.

Bibliography

[1] S-I Amari. *Differential-geometrical methods in statistics.* Number 28 in Lecture notes in statistics. Springer-Verlag, Berlin, 1985.

[2] S-I Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10:251–276, 1998.

[3] D Barber, D Saad, and P Sollich. Finite-size effects in on-line learning of multilayer neural networks. *Europhysics Lett.*, 34(2):151–156, 1996.

[4] M Biehl and H Schwarze. Learning by online gradient descent. *J. Phys. A*, 28(643), 1995.

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

[6] R.A Fisher. The use of multiple measurements in taxonomic problems. *Annual Eugenics*, 7(II):179–188, 1936. also in Contributions to Mathematical Statistics (John Wiley, NY, 1950).

[7] F.A Graybill. *Introduction to matrices with applications in statistics.* Wadsworth Publishing Co Inc, Belmont, California, 1969.

[8] B Hassibi and D.G Stork. Second order derivatives for network pruning: optimal brain surgeon. In C.L Giles, S.J Hanson, and J.D Cowan, editors, *Advances in Neural Information Processing Systems (NIPS) 5*, pages 164–171, San Mateo, California, 1993. Morgan Kaufmann.

[9] S Haykin. *Neural networks a comprehensive foundation.* Prentice Hall, 1994.

[10] J Hertz, A Krogh, and R.G Palmer. *Introduction to the theory of neural networks.* Addison-Wesley, 1991.

[11] Tom Heskes and Jeroen Coolen. Learning in two-layered networks with correlated examples. *J. Phys. A*, 30(14):4983–4992, 1997.

[12] C.J Merz and P.M Murphy. UCI repository of machine learning databases, 1998.

[13] M.K Murray and J.W Rice. *Differential geometry and statistics.* Number 48 in Monographs on Statistics and Applied Probability. Chapman and Hall, London, 1993.

[14] G.B Orr. *Dynamics and algorithms for stochastic search.* PhD thesis, Oregon Graduate Institute of Science and Technology, 1995.

[15] G.B Orr and T.K Leen. Using curvature information for fast stochastic search. In M.C Mozer, M.I Jordan, and T Petsche, editors, *Advances in Neural Information Processing Systems (NIPS) 9*, pages 606–612, Cambridge MA, 1996. MIT Press.

[16] B.A Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 1994.

[17] A Prugel-Bennett. On-line learning with momentum. Unpublished report, Nordita, Denmark, 1996.

[18] M Rattray and D Saad. Incorporating curvature information into on-line learning. In D Saad, editor, *Online learning in Neural Networks*, Isaac Newton Institute, Cambridge, UK, 17-21 November 1997.

[19] M Rattray and D Saad. The dynamics of matrix momentum. In *ICANN*, 1998.

[20] P Riegler and M Biehl. On-line backpropagation in two-layered neural networks. *J. Phys. A*, 28(20):L507–L513, 1995.

[21] J.J Rissanen. Fisher information and stochastic complexity. *IEEE Trans. Inf. Theory*, 42(1):40–47, 1996.

[22] D.E Rumelhart, G.E Hinton, and R.J Williams. Learning internal representations by error propagation. In D.E Rumelhart and J.L McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume I, Cambridge, MA, 1986. MIT Press.

[23] D Saad and M Rattray. Globally optimal parameters for on-line learning in multilayer neural networks. *Phys. Rev. Lett.*, 79(13):2578–2580, 29 September 1997.

[24] D Saad and S.A Solla. On-line learning in soft committee machines. *Phys. Rev. E*, 52(4225), 1995.

[25] S Scarpetta, S-J Farmer, M Rattray, and D Saad. Natural gradient matrix momentum. Submitted, 1998.

[26] A.H.L West, D Saad, and I.T Nabney. The learning dynamics of a universal approximator. In M.C Mozer, M.I Jordan, and T Petsche, editors, *Advances in Neural Information Processing Systems (NIPS) 9*, pages 288–294, Cambridge MA, 1996. MIT Press.

[27] W Wiegerinck, A Komoda, and T Heskes. Stochastic dynamics of learning with momentum in neural networks. *J Phys A*, 27(4425), 1994.

[28] H.H Yang and S-I Amari. Natural gradient descent for training multi-layer perceptrons. *IEEE Trans. Neural Networks*, 1997.

[29] H Zhu. Bayesian geometric theory of learning algorithms. In *Intl. Conf. Neural Networks(ICNN'97)*, volume 2, pages 1041–1044, 1997.

[30] H Zhu and R Rohwer. Information geometry, Bayesian inference, ideal estimates and error decomposition. Technical Report SFI 98-06-044, Santa Fe Institute, 1998.

Appendix A

Variables and notation

The appendices to this report contain mathematical proofs and notes that support the main report but would interrupt its flow if included. Where possible, all variables and notation are standardised and follow the notation used in [24]. This section contains a list of those variables and an explanation of notation where it is deemed necessary.

A.1 Variables

Network inputs	
M	Number of example input-output patterns
μ	index over example set
$\mathbf{D} = \{(\xi^1, \zeta^1), ..., (\xi^M, \zeta^M)\}$	example set
ξ^μ	input pattern μ
ξ	input data, consisting of all input patterns
ζ^μ	expected output pattern μ
ζ_n^μ	expected output for output node n, input ξ^μ
σ_m	variance of noise on input-output example pairs
Multilayer perceptrons	
N	number of input nodes
K	number of hidden nodes
W	number of output nodes
$\mathbf{J}$	input-to-hidden weights
b_j	bias on hidden node j
$g(x)$	activation function of hidden nodes
$\mathbf{x}_i^\mu$	activation of hidden node i for input ξ^μ
a_{jo}	hidden-to-output weights for hidden node j, output node o
c_o	bias on output node o
$\sigma_n(J, \xi^\mu)$	network output for output node n and input ξ^μ
$\mathbf{\Theta}_i^\mu = \{\mathbf{J}_{ij}, b_j, a_{jo}, c_o\}$	network parameters for i^{th} input node, μ^{th} input pattern
Soft committee networks	
N	number of input nodes
K	number of hidden nodes
$\mathbf{J}$	input-to-hidden weights, and the only network parameters
$g(x)$	activation function of hidden nodes (usually $erf(x_i/\sqrt{2})$)
$\mathbf{x}_i^\mu = \mathbf{J}_i \cdot \xi^\mu$	activation of hidden node i for input ξ^μ
$\sigma(J, \xi^\mu) = \sum_{i=1}^K g(\mathbf{x}_i^\mu)$	network output for input ξ^μ
Soft committee "teacher" networks	
M	number of hidden nodes in teacher network
$\mathbf{B}$	input-to-hidden weights in teacher network
$\mathbf{y}_n^\mu = \mathbf{B}_n \cdot \xi^\mu$	activation of hidden node n in teacher network
$\rho \sim N(0, \sigma_m^2)$	output noise in teacher network
$\zeta^\mu = \sum_{n=1}^M g(\mathbf{y}_i^\mu) + \rho^\mu$	output of teacher network and expected output of student network

Error functions	
$\epsilon_{\mathbf{J}}(\xi^\mu, \zeta^\mu)$	error function for example μ and network parameters $\mathbf{J}$
$\epsilon_{\mathbf{J}}(\xi^\mu, \zeta^\mu) = \frac{1}{2}[\sigma(\mathbf{J}, \xi^\mu) - \zeta^\mu]^2$	quadratic error function
$\epsilon_g(\mathbf{J}) \equiv \langle \epsilon_{\mathbf{J}}(\xi^\mu, \zeta^\mu) \rangle_{\{\xi\}}$	generalisation error
Learning parameters	
$\alpha = \frac{\mu}{N}$	Normalised pattern index, used as time variable
η	learning rate
η_0	initial learning rate
η_α	learning rate at timestep α
η_{eff}	effective learning rate
β	momentum parameter
Ω	momentum variable
Ω_i^μ	Ω for pattern μ, hidden variable i
γ	momentum parameter used to rescale $1 - \beta$
k	Balance between gradient descent and momentum terms
δ_i^μ	backpropagation delta for hidden node i
$\nabla_{\mathbf{J}} \epsilon_{\mathbf{J}}(\xi^\mu, \zeta^\mu) = \frac{\partial \epsilon}{\partial \mathbf{J}}$	gradient at $\mathbf{J}$
$\mathbf{H} = \left[\frac{\partial^2 \epsilon_{\mathbf{J}}(\xi^\mu, \zeta^\mu)}{\partial \mathbf{J}_{i\alpha} \partial \mathbf{J}_{k\beta}}\right]$	Hessian matrix
$\mathbf{G}(\Theta), \mathbf{G}(\mathbf{J}) = [\mathbf{G}_{i\alpha,k\beta}]$	Fisher information matrix
$\mathbf{G}_{i\alpha,k\beta} = \frac{1}{\sigma_{\mathbf{m}}{}^4} \left\langle \frac{\partial \epsilon_{\mathbf{J}}(\xi,\zeta_{\mathbf{J}})}{\partial \mathbf{J}_{i\alpha}} \frac{\partial \epsilon_{\mathbf{J}}(\xi,\zeta_{\mathbf{J}})}{\partial \mathbf{J}_{i\beta}} \right\rangle_{\{\xi\}}$	element of Fisher information matrix
$\mathbf{G}_{ik}$	block of Fisher information matrix
$\mathbf{A}(\mathbf{J})$	Matrix whose elements are proportional to $\mathbf{G}$'s
$\mathbf{A}_{ik}$	block of $\mathbf{A}(\mathbf{J})$
$\mathbf{\Delta}_{ij}$	Used in the calculation of $\mathbf{A}_{ik}$
$P(\zeta \vert \xi; \mathbf{J})$	Probability of output ζ given input ξ
$L(\xi, \zeta \vert \mathbf{J})$	Loss function for input ξ, output ζ
$\text{KL}(p(x), q(x))$	Kullback-Leibler distance between distributions p and q, parameterised by x
δ_{ab}	Kronecker delta
Order parameters	
$\mathbf{Q}_{ik} = \langle \mathbf{x}_i \mathbf{x}_k \rangle \equiv \mathbf{J}_i \cdot \mathbf{J}_k$	order parameters
$\mathbf{R}_{in} = \langle \mathbf{x}_i \mathbf{y}_n \rangle \equiv \mathbf{J}_i \cdot \mathbf{B}_n$	
$\mathbf{T}_{nm} = \langle \mathbf{y}_n \mathbf{y}_m \rangle \equiv \mathbf{B}_n \cdot \mathbf{B}_m$	
t_{init}	Diagonal values for T_{nm}
$C_{ik} = \Omega_i \cdot \Omega_k$	
$D_{in} = \Omega_i \cdot \mathbf{B}_n$	
$E_{ik} = \mathbf{J}_i \cdot \Omega_k$	

A.2 Notation

The notation in this document is fairly simple. Notations that the reader may not be familiar with are $\langle A \rangle_{\{\xi\}}$ denoting an average of A over variable ξ and $[A]$ denoting a block matrix. Occasionally an Einstein summation convention where $A_i B_i$ is shorthand for $\sum_i A_i B_i$ might creep into the calculations but this should be noted when it is used.

Appendix B

Analysis of MM-NGD for soft committee machines

This appendix contains the workings needed to create the update equations for the order parameters of matrix momentum for natural gradient descent. The order parameters are $\mathbf{Q}_{ik} = \mathbf{J}_i^T \mathbf{J}_k$, $\mathbf{R}_{in} = \mathbf{J}_i^T \mathbf{B}_n$, $\mathbf{T}_{nm} = \mathbf{B}_n^T \mathbf{B}_m$, $\mathbf{D}_{in} = \mathbf{\Omega}_i^T \mathbf{B}_n$, $\mathbf{E}_{ik} = \mathbf{J}_i^T \mathbf{\Omega}_k$ and $\mathbf{C}_{ik} = \mathbf{\Omega}_i^T \mathbf{\Omega}_k$. The update equations used here are for MM-NGD with η and β scaled by the input size N. Assumptions made are that the input size N is infinite and that the inputs are taken from a normal distribution so that $\left\langle \xi^T \xi \right\rangle = N$.

Expanding out the update equations for $\mathbf{J}$ and $\mathbf{\Omega}$ given in equations 1.30, 1.31 with an error gradient of $\nabla_{\mathbf{J}} \epsilon_{\mathbf{J}}(\xi^\mu, \zeta^\mu) = -\delta_i^\mu \xi^\mu$, we get

$$
\begin{aligned}
\mathbf{J}_i^{\mu+1} &= \mathbf{J}_i^\mu + \frac{k\eta}{N^2} \delta_i^\mu \xi^\mu + \left[\frac{1}{N} \left(1 - \frac{k}{N} \mathbf{A}(\mathbf{J}) \right) \Omega \right]_i \\
&= \mathbf{J}_i^\mu + \frac{k\eta}{N^2} \delta_i^\mu \xi^\mu + \frac{1}{N} \sum_j (\delta_{ij} - \frac{k}{N} A_{ij}^\mu) \mathbf{\Omega}_j^\mu \\
&= \mathbf{J}_i^\mu + \frac{k\eta}{N^2} \delta_i^\mu \xi^\mu + \frac{1}{N} \mathbf{\Omega}_i^\mu + O\left(\frac{1}{N^2} \right) \\
\mathbf{\Omega}_i^\mu &= N(\mathbf{J}_i^\mu - \mathbf{J}_i^{\mu-1}) \\
&= N(\mathbf{J}_i^{\mu-1} + \frac{k\eta}{N^2} \delta_i^{\mu-1} \xi^{\mu-1} + \frac{1}{N} \sum_j (-\delta_{ij} \frac{k}{N} A_{ij}^{\mu-1}) \mathbf{\Omega}_j^{\mu-1} - \mathbf{J}_i^{\mu-1}) \\
&= N(\frac{k\eta}{N^2} \delta_i^{\mu-1} \xi^{\mu-1} + \frac{1}{N} \mathbf{\Omega}_i^{\mu-1} - \frac{k}{N^2} \sum_j A_{ij}^{\mu-1} \mathbf{\Omega}_j^{\mu-1}) \\
\mathbf{\Omega}_i^{\mu+1} &= \mathbf{\Omega}_i^\mu + \frac{k}{N} \eta \delta_i^\mu \xi^\mu - \frac{k}{N} \sum_j A_{ij}^\mu \mathbf{\Omega}_j^\mu
\end{aligned}
$$

where $x_i = \mathbf{J}_i^T \xi$, $y_n = \mathbf{B}_n^T \xi$, $z_i = \mathbf{\Omega}_i^T \xi$ and

$$
A_{ik} = \frac{2}{\pi \sqrt{\Delta_{ik}}} [I - \frac{1}{\Delta_{ik}} ((1+\mathbf{Q}_{kk}) \mathbf{J}_i \mathbf{J}_i^T + (1 + \mathbf{Q}_{ii}) \mathbf{J}_k \mathbf{J}_k^T - \mathbf{Q}_{ik} (\mathbf{J}_i \mathbf{J}_k^T + \mathbf{J}_k \mathbf{J}_i^T))] \qquad \text{(B.1)}
$$

One term that crops up frequently in the update equations is the sum $\sum_j A_{ij} \mathbf{\Omega}_j$.

Expanding it out gives

$$\begin{aligned}\sum_j A_{ij}\mathbf{\Omega}_j &= \sum_j \frac{2}{\pi\sqrt{\Delta_{ij}}}[I - \frac{1}{\Delta_{ij}}((1+\mathbf{Q}_{jj})\mathbf{J}_i\mathbf{J}_i^T + (1+\mathbf{Q}_{ii})\mathbf{J}_j\mathbf{J}_j^T - \mathbf{Q}_{ij}(\mathbf{J}_i\mathbf{J}_j^T + \mathbf{J}_j\mathbf{J}_i^T))]\mathbf{\Omega}_j \\ &= \frac{2}{\pi}\sum_j \frac{1}{\sqrt{\Delta_{ij}}}[\mathbf{\Omega}_j - \frac{1}{\Delta_{ij}}((1+\mathbf{Q}_{jj})\mathbf{E}_{ij}\mathbf{J}_i + (1+\mathbf{Q}_{ii})\mathbf{E}_{jj}\mathbf{J}_j - \mathbf{Q}_{ij}(\mathbf{E}_{jj}\mathbf{J}_i + \mathbf{E}_{ij}\mathbf{J}_j))]\end{aligned}$$

Using these definitions, the equations of motion for the order parameters are:

$$\begin{aligned}\mathbf{Q}_{ik}^{\mu+1} &= \mathbf{J}_i^{\mu+1T}\mathbf{J}_k^{\mu+1} \\ &= \left(\mathbf{J}_i^{\mu T} + \frac{k\eta}{N^2}\delta_i^\mu \xi^{\mu T} + \frac{1}{N}\mathbf{\Omega}_i^T + O\left(\frac{1}{N^2}\right)\right)\left(\mathbf{J}_k^\mu + \frac{k\eta}{N^2}\delta_k^\mu \xi^\mu + \frac{1}{N}\mathbf{\Omega}_k + O\left(\frac{1}{N^2}\right)\right) \\ &= \mathbf{J}_i^{\mu T}\mathbf{J}_k^\mu + \mathbf{J}_i^{\mu T}\frac{1}{N}\mathbf{\Omega}_k + \frac{1}{N}\mathbf{\Omega}_i^T\mathbf{J}_k^\mu + O\left(\frac{1}{N^2}\right) \\ &= \mathbf{Q}_{ik}^\mu + \frac{1}{N}\left[\mathbf{J}_i^{\mu T}\mathbf{\Omega}_k + \mathbf{\Omega}_i^T\mathbf{J}_k^\mu\right] + O\left(\frac{1}{N^2}\right) \\ \frac{d\mathbf{Q}_{ik}}{d\alpha} &= \mathbf{J}_i^T\mathbf{\Omega}_k + \mathbf{\Omega}_i^T\mathbf{J}_k = \mathbf{E}_{ik} + \mathbf{E}_{ki}\end{aligned}$$

$$\begin{aligned}\mathbf{R}_{in}^{\mu+1} &= \mathbf{J}_i^{\mu+1T}\mathbf{B}_n^{\mu+1} \\ &= \left(\mathbf{J}_i^{\mu T} + \frac{k\eta}{N^2}\delta_i^\mu \xi^{\mu T} + \frac{1}{N}\mathbf{\Omega}_i^T + O\left(\frac{1}{N^2}\right)\right)\mathbf{B}_n \\ &= \mathbf{J}_i^{\mu T}\mathbf{B}_n + \frac{1}{N}\mathbf{\Omega}_i^T\mathbf{B}_n + O\left(\frac{1}{N^2}\right) \\ \frac{d\mathbf{R}_{in}}{d\alpha} &= \mathbf{\Omega}_i^T\mathbf{B}_n = \mathbf{D}_{in}\end{aligned}$$

$$\begin{aligned}\mathbf{D}_{in}^{\mu+1} &= \mathbf{\Omega}_i^{\mu+1T}\mathbf{B}_n^{\mu+1} \\ &= \left(\mathbf{\Omega}_i^{\mu T} - \frac{k}{N}\sum_j \left(A_{ij}^\mu \mathbf{\Omega}_j^\mu\right)^T + \frac{k}{N}\eta\delta_i^\mu \xi^{\mu T}\right)\mathbf{B}_n \\ &= \mathbf{D}_{in}^\mu - \frac{k}{N}\sum_j \left(A_{ij}^\mu \mathbf{\Omega}_j^\mu\right)^T \mathbf{B}_n + \frac{k}{N}\eta\delta_i^\mu y_n \\ \frac{d\mathbf{D}_{in}}{d\alpha} &= -k\sum_j \left(A_{ij}\mathbf{\Omega}_j\right)^T \mathbf{B}_n + k\eta\left\langle \delta_i y_n\right\rangle \\ &= -k\Big(\frac{2}{\pi}\sum_j \frac{1}{\sqrt{\Delta_{ij}}}\Big[\mathbf{\Omega}_j - \frac{1}{\Delta_{ij}}\big(\left(1+\mathbf{Q}_{jj}\right)\mathbf{E}_{ij}\mathbf{J}_i + \left(1+\mathbf{Q}_{ii}\right)\mathbf{E}_{jj}\mathbf{J}_j \\ &\quad -\mathbf{Q}_{ij}\left(\mathbf{E}_{jj}\mathbf{J}_i + \mathbf{E}_{ij}\mathbf{J}_j\right)\big)\Big]\Big)^T\mathbf{B}_n + k\eta\left\langle \delta_i y_n\right\rangle \\ &= -k\Big(\frac{2}{\pi}\sum_j \frac{1}{\sqrt{\Delta_{ij}}}\Big[\mathbf{D}_{jn} - \frac{1}{\Delta_{ij}}\big(\left(1+\mathbf{Q}_{jj}\right)\mathbf{E}_{ij}\mathbf{R}_{in} + \left(1+\mathbf{Q}_{ii}\right)\mathbf{E}_{jj}\mathbf{R}_{jn} \\ &\quad -\mathbf{Q}_{ij}\left(\mathbf{E}_{jj}\mathbf{R}_{in} + \mathbf{E}_{ij}\mathbf{R}_{jn}\right)\big)\Big]\Big) + k\eta\left\langle \delta_i y_n\right\rangle\end{aligned}$$

$$\begin{aligned}\mathbf{E}_{ik}^{\mu+1} &= \mathbf{J}_i^{\mu+1T}\mathbf{\Omega}_k^{\mu+1} \\ &= \left(\mathbf{J}_i^{\mu T} + \frac{k\eta}{N^2}\delta_i^\mu \xi^{\mu T} + \frac{1}{N}\mathbf{\Omega}_i^T + O\left(\frac{1}{N^2}\right)\right)\left(\mathbf{\Omega}_k^\mu - \frac{k}{N}\sum_j A_{kj}^\mu \mathbf{\Omega}_j^\mu + \frac{k}{N}\eta\delta_k^\mu \xi^\mu\right)\end{aligned}$$

$$
\begin{aligned}
&= \mathbf{J}_i^{\mu T}\mathbf{\Omega}_k^{\mu} - \frac{k}{N}\mathbf{J}_i^{\mu T}\sum_j A_{kj}^{\mu}\mathbf{\Omega}_j^{\mu} + \frac{k}{N}\mathbf{J}_i^{\mu T}\eta\delta_k^{\mu}\xi^{\mu} + \frac{1}{N}\mathbf{\Omega}_i^T\mathbf{\Omega}_k + O\Big(\frac{1}{N^2}\Big)\\
&= \mathbf{E}_{ik}^{\mu} + \frac{1}{N}\left(\mathbf{\Omega}_i^T\mathbf{\Omega}_k - k\mathbf{J}_i^{\mu T}\sum_j A_{kj}^{\mu}\mathbf{\Omega}_j^{\mu} + k\mathbf{J}_i^{\mu T}\eta\delta_k^{\mu}\xi^{\mu}\right)
\end{aligned}
$$

$$
\begin{aligned}
\frac{d\mathbf{E}_{ik}}{d\alpha} &= \mathbf{\Omega}_i^T\mathbf{\Omega}_k - k\mathbf{J}_i^T\sum_j A_{kj}\mathbf{\Omega}_j + k\eta\left\langle\delta_k x_i\right\rangle\\
&= \mathbf{C}_{ik} - k\mathbf{J}_i^T\Big(\frac{2}{\pi}\sum_j\frac{1}{\sqrt{\Delta_{kj}}}\Big[\mathbf{\Omega}_j - \frac{1}{\Delta_{kj}}\big(\,(1{+}\mathbf{Q}_{jj})\,\mathbf{E}_{kj}\mathbf{J}_k + (1{+}\mathbf{Q}_{kk})\,\mathbf{E}_{jj}\mathbf{J}_j\\
&\quad -\mathbf{Q}_{kj}\,(\mathbf{E}_{jj}\mathbf{J}_k{+}\mathbf{E}_{kj}\mathbf{J}_j)\,\big)\Big]\Big) + k\eta\left\langle\delta_k x_i\right\rangle\\
&= \mathbf{C}_{ik} - k\Big(\frac{2}{\pi}\sum_j\frac{1}{\sqrt{\Delta_{kj}}}\Big[\mathbf{E}_{ij} - \frac{1}{\Delta_{kj}}\big(\,(1{+}\mathbf{Q}_{jj})\,\mathbf{E}_{kj}\mathbf{Q}_{ik} + (1{+}\mathbf{Q}_{kk})\,\mathbf{E}_{jj}\mathbf{Q}_{ij}\\
&\quad -\mathbf{Q}_{kj}\,(\mathbf{E}_{jj}\mathbf{Q}_{ik}{+}\mathbf{E}_{kj}\mathbf{Q}_{ij})\,\big)\Big]\Big) + k\eta\left\langle\delta_k x_i\right\rangle
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{C}_{ik}^{\mu+1} &= \mathbf{\Omega}_i^{\mu+1T}\mathbf{\Omega}_k^{\mu+1}\\
&= \left(\mathbf{\Omega}_i^{\mu T} - \frac{k}{N}\sum_j\left(A_{ij}^{\mu}\mathbf{\Omega}_j^{\mu}\right)^T + \frac{k}{N}\eta\delta_i^{\mu}\xi^{\mu T}\right)\left(\mathbf{\Omega}_k^{\mu} - \frac{k}{N}\sum_j A_{kj}^{\mu}\mathbf{\Omega}_j^{\mu} + \frac{k}{N}\eta\delta_k^{\mu}\xi^{\mu}\right)\\
&= \mathbf{\Omega}_i^{\mu T}\mathbf{\Omega}_k^{\mu} - \mathbf{\Omega}_i^{\mu T}\frac{k}{N}\sum_j A_{kj}^{\mu}\mathbf{\Omega}_j^{\mu} + \mathbf{\Omega}_i^{\mu T}\frac{k}{N}\eta\delta_k^{\mu}\xi^{\mu} - \frac{k}{N}\sum_j\left(A_{ij}^{\mu}\mathbf{\Omega}_j^{\mu}\right)^T\mathbf{\Omega}_k^{\mu} + \frac{k}{N}\eta\delta_i^{\mu}\xi^{\mu T}\mathbf{\Omega}_k^{\mu}\\
&\quad + \frac{k^2}{N^2}\eta^2\delta_i^{\mu}\xi^{\mu T}\delta_k^{\mu}\xi^{\mu} + O\Big(\frac{1}{N^2}\Big)\\
&= \mathbf{C}_{ik}^{\mu} + \frac{1}{N}\left[-k\mathbf{\Omega}_i^{\mu T}\sum_j A_{kj}^{\mu}\mathbf{\Omega}_j^{\mu} - k\sum_j\left(A_{ij}^{\mu}\mathbf{\Omega}_j^{\mu}\right)^T\mathbf{\Omega}_k^{\mu} + k\eta\delta_k^{\mu}z_i^{\mu} + k\eta\delta_i^{\mu}z_k^{\mu} + k^2\eta^2\delta_i^{\mu}\delta_k^{\mu}\right]\\
&\quad + O\Big(\frac{1}{N^2}\Big)
\end{aligned}
$$

$$
\begin{aligned}
\frac{d\mathbf{C}_{ik}}{d\alpha} &= -k\mathbf{\Omega}_i^T\sum_j A_{kj}\mathbf{\Omega}_j - k\sum_j\left(A_{ij}\mathbf{\Omega}_j\right)^T\mathbf{\Omega}_k + k\eta\left\langle\delta_k z_i + \delta_i z_k\right\rangle + k^2\eta^2\left\langle\delta_i\delta_k\right\rangle\\
&= -k\mathbf{\Omega}_i^T\Big(\frac{2}{\pi}\sum_j\frac{1}{\sqrt{\Delta_{kj}}}\Big[\mathbf{\Omega}_j - \frac{1}{\Delta_{kj}}\big(\,(1{+}\mathbf{Q}_{jj})\,\mathbf{E}_{kj}\mathbf{J}_k + (1{+}\mathbf{Q}_{kk})\,\mathbf{E}_{jj}\mathbf{J}_j\\
&\quad -\mathbf{Q}_{kj}\,(\mathbf{E}_{jj}\mathbf{J}_k{+}\mathbf{E}_{kj}\mathbf{J}_j)\,\big)\Big]\Big)\\
&\quad -k\Big(\frac{2}{\pi}\sum_j\frac{1}{\sqrt{\Delta_{ij}}}\Big[\mathbf{\Omega}_j - \frac{1}{\Delta_{ij}}\big(\,(1{+}\mathbf{Q}_{jj})\,\mathbf{E}_{ij}\mathbf{J}_i + (1{+}\mathbf{Q}_{ii})\,\mathbf{E}_{jj}\mathbf{J}_j\\
&\quad -\mathbf{Q}_{ij}\,(\mathbf{E}_{jj}\mathbf{J}_i{+}\mathbf{E}_{ij}\mathbf{J}_j)\,\big)\Big]\Big)^T\mathbf{\Omega}_k + k\eta\left\langle\delta_k z_i + \delta_i z_k\right\rangle + k^2\eta^2\left\langle\delta_i\delta_k\right\rangle\\
&= -k\frac{2}{\pi}\sum_j\frac{1}{\sqrt{\Delta_{kj}}}\Big[\mathbf{C}_{ij} - \frac{1}{\Delta_{kj}}\big(\,(1{+}\mathbf{Q}_{jj})\,\mathbf{E}_{kj}\mathbf{E}_{ki} + (1{+}\mathbf{Q}_{kk})\,\mathbf{E}_{jj}\mathbf{E}_{ji}\\
&\quad -\mathbf{Q}_{kj}\,(\mathbf{E}_{jj}\mathbf{E}_{ki}{+}\mathbf{E}_{kj}\mathbf{E}_{ji})\,\big)\Big]\\
&\quad -k\frac{2}{\pi}\sum_j\frac{1}{\sqrt{\Delta_{ij}}}\Big[\mathbf{C}_{jk} - \frac{1}{\Delta_{ij}}\big(\,(1{+}\mathbf{Q}_{jj})\,\mathbf{E}_{ij}\mathbf{E}_{ik} + (1{+}\mathbf{Q}_{ii})\,\mathbf{E}_{jj}\mathbf{E}_{jk}\\
&\quad -\mathbf{Q}_{ij}\,(\mathbf{E}_{jj}\mathbf{E}_{ik}{+}\mathbf{E}_{ij}\mathbf{E}_{jk})\,\big)\Big] + k\eta\left\langle\delta_k z_i + \delta_i z_k\right\rangle + k^2\eta^2\left\langle\delta_i\delta_k\right\rangle
\end{aligned}
$$

Appendix C

The Fisher information matrix

The Fisher information matrix is used in natural gradient descent, but it is also used in information theory and in the calculation of the Cramer-Rao bound $Tr(\mathbf{G}^{-1}(\mathbf{J}))$ on a network's performance (the Cramer-Rao bound gives a minimum distance between the network's estimates of a parameter value and the true parameter value. $Tr(\mathbf{A})$ is the trace of matrix $\mathbf{A}$.

This chapter gives some notes on the maths used to calculate the Fisher information matrix for a soft committee network and multilayer perceptron. These calculations are taken/ based on Magnus Rattray's notes and [2].

C.1 Exact Fisher information matrix for a soft committee network

The Fisher information matrix for a neural network is given as $\mathbf{G}(\mathbf{J}) = [\mathbf{G}_{i\alpha,k\beta}]$, where $\mathbf{G}_{i\alpha,k\beta}$ is

$$\mathbf{G}_{i\alpha,k\beta} = \left\langle \frac{\partial \log p_{\mathbf{J}}(\zeta_m|\xi)}{\partial \mathbf{J}_{i\alpha}} \frac{\partial \log p_{\mathbf{J}}(\zeta_m|\xi)}{\partial \mathbf{J}_{k\beta}} \right\rangle_{\{\zeta_m,\xi\}} = \frac{1}{\sigma_m^4} \left\langle \frac{\partial \epsilon_{\mathbf{J}}(\xi,\zeta_{\mathbf{J}})}{\partial \mathbf{J}_{i\alpha}} \frac{\partial \epsilon_{\mathbf{J}}(\xi,\zeta_{\mathbf{J}})}{\partial \mathbf{J}_{k\beta}} \right\rangle_{\{\xi\}} \tag{C.1}$$

It is useful because it is invariant to re-parameterisations of the input space ξ.

To derive the Fisher matrix for neural networks, we first take the log of $p_{\mathbf{J}}(\zeta_m, \xi)$ as defined in [18]:

$$p_{\mathbf{J}}(\zeta_m, \xi) = \frac{1}{\sqrt{2\pi\sigma_m^2}} \exp\left(\frac{-(\zeta_m - \phi_{\mathbf{J}}(\xi))^2}{2\sigma_m^2}\right) \tag{C.2}$$

$$-\log p_{\mathbf{J}}(\zeta_m, \xi) = \left[\zeta_m - \sum_{i=1}^{K} g\left(\sum_{\alpha} \mathbf{J}_{i\alpha}\xi_\alpha\right)\right]^2 / 2\sigma_m^2 + consts \tag{C.3}$$

Substituting this into the above definition of $\mathbf{G}_{i\alpha,k\beta}$ gives:

$$\mathbf{G}_{i\alpha,k\beta} = \frac{1}{4\sigma_m^4} \left\langle \frac{\partial}{\partial \mathbf{J}_{i\alpha}} \left[\zeta_m - \sum_{\mathbf{J}=1}^{K} g\left(\sum_{\gamma} \mathbf{J}_{\mathbf{J}\gamma}\xi_\gamma\right)\right]^2 \frac{\partial}{\partial \mathbf{J}_{k\beta}} \left[\zeta_m - \sum_{l=1}^{K} g\left(\sum_{\delta} \mathbf{J}_{l\delta}\xi_\delta\right)\right]^2 \right\rangle \tag{C.4}$$

which, since $\frac{\partial}{\partial \mathbf{J}_{i\alpha}}\left[\zeta_m - \sum_{j=1}^{K} g\left(\sum_\gamma \mathbf{J}_{j\gamma}\xi_\gamma\right)\right]^2 = -2\xi_\alpha g'\left(\sum_\gamma \mathbf{J}_{i\gamma}\xi_\gamma\right)\left[\zeta_m - \sum_{j=1}^{K} g\left(\sum_\gamma \mathbf{J}_{j\gamma}\xi_\gamma\right)\right]$, becomes

$$\mathbf{G}_{i\alpha,k\beta} = \frac{1}{\sigma_m^4}\left\langle \xi_\alpha\xi_\beta g'\left(\sum_\gamma \mathbf{J}_{i\gamma}\xi_\gamma\right) g'\left(\sum_\delta \mathbf{J}_{k\delta}\xi_\delta\right)\left[\zeta_m - \sum_{j=1}^{K} g\left(\sum_\gamma \mathbf{J}_{j\gamma}\xi_\gamma\right)\right]^2\right\rangle \tag{C.5}$$

and, since $\zeta_m = \sum_{j=1}^{K} g\left(\sum_\gamma \mathbf{J}_{j\gamma}\xi_\gamma\right) + \rho$ which means that $\left\langle\left[\zeta_m - \sum_{j=1}^{K} g\left(\sum_\gamma \mathbf{J}_{j\gamma}\xi_\gamma\right)\right]^2\right\rangle = \sigma_m^2$, this reduces to

$$\mathbf{G}_{i\alpha,k\beta} = \frac{1}{\sigma_m^2}\left\langle \xi_\alpha\xi_\beta g'(\mathbf{x}_i)\, g'(\mathbf{x}_k)\right\rangle_{\{\xi\}} \tag{C.6}$$

If $g(x) = \mathrm{erf}(x/\sqrt{2})$, we get (moving into matrix notation):

$$\begin{aligned}
\mathbf{G}_{ik} &= \frac{1}{\sigma_m^2}\left\langle g'\left(\mathbf{J}_i^T\xi\right) g'\left(\mathbf{J}_k^T\xi\right)\xi_\alpha\xi_\beta^T\right\rangle_{\{\xi\}} && \text{(C.7)}\\
&= \frac{1}{\sigma_m^2}\left\langle \frac{2}{\pi}\xi_\alpha\xi_\beta^T e^{-\frac{\left(\mathbf{J}_i^T\xi\right)^2}{2}-\frac{\left(\mathbf{J}_k^T\xi\right)^2}{2}}\right\rangle_{\{\xi\}} && \text{(C.8)}\\
&= \frac{1}{\sigma_m^2}\frac{2}{\pi}\int \frac{d\xi}{\sqrt{(2\pi)^N}} e^{-\frac{1}{2}\xi^T\left[\mathbf{I}+\mathbf{J}_i\mathbf{J}_i^T+\mathbf{J}_k\mathbf{J}_k^T\right]\xi}\xi_\alpha\xi_\beta^T && \text{(C.9)}\\
&= \frac{1}{\sigma_m^2}\frac{2}{\pi}\int d\xi \frac{1}{\sqrt{(2\pi)^N}} e^{-\frac{1}{2}\xi^T A_{ik}\xi}\xi_\alpha\xi_\beta^T && \text{(C.10)}
\end{aligned}$$

where $\int d\xi = \prod_\alpha \int d\xi_\alpha$ is a nest of integrals over $d\xi_\alpha$, $A_{ik} = \mathbf{I} + \mathbf{J}_i\mathbf{J}_i^T + \mathbf{J}_k\mathbf{J}_k^T$ is a scalar and $\xi_\alpha\xi_\beta^T$ is an $N \times N$ matrix.

[7] theorem 10.5.1 states that

$$\begin{aligned}
\mathbf{G} &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}\left(x'\mathbf{A}x + x'a + a_0\right) e^{-\left(x'\mathbf{B}x + x'b + b_0\right)} dx_1 dx_2 ... dx_n\\
&= \frac{1}{2}\pi^{n/2}|\mathbf{B}|^{-1/2} e^{(1/4)b'\mathbf{B}^{-1}b - b_0}\left[tr\left(\mathbf{A}\mathbf{B}^{-1}\right) - b'\mathbf{B}^{-1}a + \frac{1}{2}b'\mathbf{B}^{-1}\mathbf{A}\mathbf{B}^{-1}b + 2a_0\right]
\end{aligned}$$

which is

$$\begin{aligned}
\mathbf{G} &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}\left(x'\mathbf{A}x\right) e^{-\left(x'\mathbf{B}x\right)} dx_1 dx_2 ... dx_n\\
&= \frac{1}{2}\pi^{n/2}|\mathbf{B}|^{-1/2}\left[tr\left(\mathbf{A}\mathbf{B}^{-1}\right)\right]
\end{aligned}$$

for $a = a_0 = b = b_0 = 0$.

We can get $\mathbf{G}_{i\alpha,k\beta}$ into this form if we multiply it top and bottom by $\sqrt{|\mathbf{A}_{ik}^{-1}|}$ and modify $\xi_x\xi_y$:

$$\begin{aligned}
\mathbf{G}_{i\alpha,k\beta} &= \frac{1}{\sigma_m^2}\frac{2}{\pi}\int d\xi \frac{\sqrt{|\mathbf{A}_{ik}^{-1}|}}{\sqrt{(2\pi)^N |\mathbf{A}_{ik}^{-1}|}} e^{-\frac{1}{2}\xi^T A_{ik}\xi}\xi_\alpha\xi_\beta^T\\
\xi_\alpha\xi_\beta &= \xi^T\Delta_{\alpha\beta}\xi\\
\Delta_{\alpha\beta} &= \delta_{i\alpha}\delta_{j\beta}
\end{aligned}$$

$$\mathbf{G}_{i\alpha,k\beta} = \frac{1}{\sigma_m^2}\frac{2}{\pi}\frac{\sqrt{|\mathbf{A}_{ik}^{-1}|}}{\sqrt{(2\pi)^N |\mathbf{A}_{ik}^{-1}|}}\int \xi^T \nabla_{\alpha\beta}\xi e^{-\frac{1}{2}\xi^T A_{ik}\xi} d\xi$$

$$= \frac{1}{\sigma_m^2}\frac{2}{\pi}\frac{\sqrt{|A_{ik}^{-1}|}}{\sqrt{(2\pi)^N |A_{ik}^{-1}|}}\frac{1}{2}\pi^{\frac{N}{2}}|A_{ik}|^{-\frac{1}{2}}\left[tr\left(\Delta_{\alpha\beta}\left(A_{ik}\right)^{-1}\right)\right]$$

From [18]: $\mathbf{G}(\mathbf{J}) = \mathbf{A}(\mathbf{J})/\sigma_m^2$, where $\mathbf{A}(\mathbf{J}) = [A_{ik}]$ is

$$A_{ik} = \frac{2}{\pi\sqrt{\Delta_k}}\left[\mathbf{I} - \frac{1}{\Delta_k}\left(\left(1+\mathbf{Q}_{kk}\right)\mathbf{J}_i\mathbf{J}_i^T + \left(1+\mathbf{Q}_{ii}\right)\mathbf{J}_k\mathbf{J}_k^T - \mathbf{Q}_{ik}\left(\mathbf{J}_i\mathbf{J}_k^T + \mathbf{J}_k\mathbf{J}_i^T\right)\right)\right] \quad \text{(C.11)}$$

$$A_{ii} = \frac{2}{\pi\sqrt{1+2\mathbf{Q}_{ii}}}\left(\mathbf{I} - \frac{2\mathbf{J}_i\mathbf{J}_i^T}{1+2\mathbf{Q}_{ii}}\right) \quad \text{(C.12)}$$

$$\Delta_k = \left(1+\mathbf{Q}_{ii}\right)\left(1+\mathbf{Q}_{kk}\right) - \mathbf{Q}_{ik}^2 \quad \text{(C.13)}$$

C.2 Exact Fisher information matrix for a multilayer perceptron

The loss function of a neural network can be defined as

$$L1(\zeta|\xi;J) = \frac{1}{2\sigma_m^2}\rho^2 = \frac{1}{2\sigma_m^2}(\sigma-\zeta)^2 \quad \text{(C.14)}$$

where $\rho = (\sigma - \zeta)$ is the teacher error, σ_m^2 is the model noise, ζ is the expected output of the network and $\sigma = \sigma_{o1\leq o\leq W}$ is the output from the network, calculated as

$$\sigma_o^\mu = \sum_j x_j^\mu a_{jo} + c_o \quad \text{(C.15)}$$

Here, $x_j^\mu = x(\sum_i \xi_i^\mu J_{ij}^\mu + b_j^\mu)$ is the activation function of hidden unit j for input datapoint μ, with derivative $x_j'^\mu$.

The elements of the Fisher Matrix are constructed from the partial derivatives of the loss function with respect to the network parameters. These are:

$$\frac{\partial L1_o^{\mu d}}{\partial \mathbf{J}_{ij}^\mu} = -\frac{\rho}{\sigma_m^2}a_{jo}^\mu x_j'^\mu \xi_i^d$$

$$\frac{\partial L1_o^{\mu d}}{\partial b_j^\mu} = -\frac{\rho}{\sigma_m^2}a_{jo}^\mu x_j'^\mu$$

$$\frac{\partial L1_o^{\mu d}}{\partial a_{jk}^\mu} = -\frac{\rho}{\sigma_m^2}x_j^\mu \delta_{ok}$$

$$\frac{\partial L1_o^{\mu d}}{\partial c_k^\mu} = -\frac{\rho}{\sigma_m^2}\delta_{ok}$$

where $L1_o^{\mu d}$ is the loss function for output unit o, datapoint μ and example d, $\mathbf{J}_{ij}$ is the weight between input unit i and hidden unit j, b_j is the bias on hidden unit j, a_{jo} is the (constant) weight between hidden unit j and output unit o, c_k is the bias on output unit k and δ_{ij} (double

index) is the Kronecker delta function. Note that there are now two indices over datapoints : μ, which indexes the example (datapoint) being processed by the learning algorithm, and d, one of the input-output example pairs that the Fisher information matrix is being created from.

Since the Fisher matrix is an average over $\frac{\partial L}{\partial \theta_i}$ where θ_i is any of the parameters $\{\mathbf{J}_{ij}, a_{ij}, b_i\}$, and each element of the matrix contains $\rho^2 \sim N(0, \sigma_m^2)$, this can be simplified [28] to $\mathbf{G}(\mathbf{J}) = \frac{1}{\sigma_m^2}\mathbf{A}(\mathbf{J})$ where the elements of $\mathbf{A}(\mathbf{J})$ are constructed from

$$\frac{\partial L_o^{\mu d}}{\partial \mathbf{J}_{ij}^{\mu}} = a_{jo}^{\mu} x_j'^{\mu} \xi_i^d \tag{C.16}$$

$$\frac{\partial L_o^{\mu d}}{\partial b_j^{\mu}} = a_{jo}^{\mu} x_j'^{\mu} \tag{C.17}$$

$$\frac{\partial L_o^{\mu d}}{\partial a_{jk}^{\mu}} = x_j^{\mu} \delta_{ok} \tag{C.18}$$

$$\frac{\partial L_o^{\mu d}}{\partial c_k^{\mu}} = \delta_{ok} \tag{C.19}$$

$$\text{where } L_o^{\mu d}(\zeta|\xi; J) = \frac{1}{2}(\sigma - \zeta)^2$$

The matrix $\mathbf{A}(\mathbf{J})$ is created from an average over outputs and examples of these partial derivatives, and is

$$\mathbf{A}(\mathbf{J}) = \left\langle \left[\frac{\partial L}{\partial J_{ij}}, \frac{\partial L}{\partial b_j}, \frac{\partial L}{\partial a_{jo}}, \frac{\partial L}{\partial c_o}\right]^T \left[\frac{\partial L}{\partial J_{ij}}, \frac{\partial L}{\partial b_j}, \frac{\partial L}{\partial a_{jo}}, \frac{\partial L}{\partial c_o}\right] \right\rangle_{\{\xi,\zeta\}} \tag{C.20}$$

If $\mathbf{J}$, $\mathbf{b}$, $\mathbf{a}$, $\mathbf{c}$, $\mathbf{x}$ and $\mathbf{x}'$ have already been calculated, then $\mathbf{A}(\mathbf{J})$ can be calculated in $(2+N) \times M \times W \times P^2 \times D + 2$ flops (floating point operations) where N is the number of network input nodes, M is the number of hidden nodes, W is the number of output nodes, $P = N \times M + M + M \times W + W$ is the total number of parameters and D is the number of patterns used to calculate the Fisher information matrix. For the speech data with a 70-70-39 network architecture ($N = 70$, $M = 70$, and $W = 39$), this is between 2.07×10^{17} flops (for $D = 9000$) and 2.3×10^{13} (for $D = 1$).

C.2.1 Reducing the calculation time

The number of flops used in calculating $\mathbf{A}(\mathbf{J})$ can be reduced by calculating the average of the second partial derivatives of the loss function directly for each pair of network parameters rather than multiplying together two sets of first derivatives then taking the average of the resulting matrix.

For the network described above, the second partial derivatives of the loss function with respect to the network parameters averaged over outputs and examples are:

$$\left\langle \frac{\partial^2 L_o^{\mu d}}{\partial J_{\alpha j}^{\mu} \partial J_{\beta k}^{\mu}} \right\rangle_{od} = \left(\frac{1}{D}\sum_d \xi_\alpha^d \xi_\beta^d\right) x_j'^{\mu} x_k'^{\mu} \left(\frac{1}{W}\sum_o a_{jo}^{\mu} a_{ko}^{\mu}\right) \tag{C.21}$$

$$\left\langle \frac{\partial^2 L_o^{\mu d}}{\partial b_j^{\mu} \partial b_k^{\mu}} \right\rangle_{od} = x_j'^{\mu} x_k'^{\mu} \left(\frac{1}{W}\sum_o a_{jo}^{\mu} a_{ko}^{\mu}\right) \tag{C.22}$$

$$\left\langle \frac{\partial^2 L_o^{\mu d}}{\partial a_{jl}^{\mu} \partial a_{km}^{\mu}} \right\rangle_{od} = {x_j}^{\mu} {x_k}^{\mu} \frac{\delta_{lm}}{W} \tag{C.23}$$

$$\left\langle \frac{\partial^2 L_o^{\mu d}}{\partial c_j^{\mu} \partial c_k^{\mu}} \right\rangle_{od} = \frac{\delta_{jk}}{W} \tag{C.24}$$

$$\left\langle \frac{\partial^2 L_o^{\mu d}}{\partial J_{ij}^{\mu} \partial b_k^{\mu}} \right\rangle_{od} = \left(\frac{1}{D} \sum_d \xi_i^d \right) {x'_j}^{\mu} {x'_k}^{\mu} \left(\frac{1}{W} \sum_o a_{jo}^{\mu} a_{ko}^{\mu} \right) \tag{C.25}$$

$$\left\langle \frac{\partial^2 L_o^{\mu d}}{\partial J_{\alpha j}^{\mu} \partial a_{\beta k}^{\mu}} \right\rangle_{od} = \left(\frac{1}{D} \sum_d \xi_\alpha^d \right) {x'_j}^{\mu} x_\beta^{\mu} a_{jk} \tag{C.26}$$

$$\left\langle \frac{\partial^2 L_o^{\mu d}}{\partial J_{ij}^{\mu} \partial c_k^{\mu}} \right\rangle_{od} = \left(\frac{1}{D} \sum_d \xi_i^d \right) {x'_j}^{\mu} a_{jk} \tag{C.27}$$

$$\left\langle \frac{\partial^2 L_o^{\mu d}}{\partial b_j^{\mu} \partial a_{\beta k}^{\mu}} \right\rangle_{od} = {x'_j}^{\mu} x_\beta^{\mu} a_{jk} \tag{C.28}$$

$$\left\langle \frac{\partial^2 L_o^{\mu d}}{\partial b_j^{\mu} \partial c_k^{\mu}} \right\rangle_{od} = {x'_j}^{\mu} a_{jk} \tag{C.29}$$

$$\left\langle \frac{\partial^2 L_o^{\mu d}}{\partial a_{jk}^{\mu} \partial c_\alpha^{\mu}} \right\rangle_{od} = {x_j}^{\mu} \frac{\delta_{k\alpha}}{W} \tag{C.30}$$

This operation uses

$$(D + W + 5)N^2M^2 + (5W + 5)NM^2 + (3W + 3)M^2 + 3WNM + 4WM + W \tag{C.31}$$

flops. For the 70-70-39 speech data network, this is between 10^{11} flops (for $D = 9000$) and 10^9 flops (for $D = 1$). If the entire dataset is used to calculate the Fisher information matrix, then most of that effort (for the example given, this is 99.9 %) is in the calculation of $\left\langle \frac{\partial^2 L_o^{\mu d}}{\partial J_{\alpha j}^{\mu} \partial J_{\beta k}^{\mu}} \right\rangle_{od}$: precalculating $\sum_d \xi_\alpha^d \xi_\beta^d$ can reduce this from $(D + W + 4)N^2M^2$ to $D + (W + 4)N^2M^2$, giving an $\mathbf{A}(\mathbf{J})$ calculation of 10^9 flops for both $D = 9000$ and $D = 1$ (for $D = 1$, this reduction is insignificant). Precalculating $\sum_o a_{jo}^{\mu} a_{ko}^{\mu}$ reduces the expected number of flops to calculate $\mathbf{A}(\mathbf{J})$ again to

$$D + 5N^2M^2 + (4W + 5)NM^2 + (2W + 3)M^2 + 3WNM + 4WM + 2W \tag{C.32}$$

flops. This is 10^8 flops for the speech data example with $D = 9000$ or $D = 1$. A summary table of the predicted flops to calculate $\mathbf{A}(\mathbf{J})$ for the 70-70-39 speech data network and the percentage of that total which is used to calculate the $\left\langle \frac{\partial^2 L_o^{\mu d}}{\partial J_{\alpha j}^{\mu} \partial J_{\beta k}^{\mu}} \right\rangle_{od}$ term is given in table C.1.

Further reductions in running time can be had from precalculating the sum $\sum_d \xi_\alpha^d$ but anything beyond this is either expensive in storage space or insignificant: further improvements in calculation times must be algorithmic.

	$D = 1$	$D = 9000$
no precalculations	1.15×10^9 flops,93.93%	2.17×10^{11} flops,99.97%
precalculate $\sum_d \xi_\alpha^d \xi_\beta^d$	1.12×10^9 flops,93.8%	1.12×10^9 flops,93.8%
precalculate $\sum_d \xi_\alpha^d \xi_\beta^d$ and $\sum_o a_{jo}^\mu a_{ko}^\mu$	1.76×10^8 flops,68.1%	1.76×10^8 flops,68.1%
first derivative-based algorithm	2.30×10^{13} flops	2.07×10^{17} flops

Table C.1: Speech network calculation times (in flops) and dominant calculation sizes (%)