Using the Natural Gradient for training Restricted Boltzmann Machines

Benjamin Schwehn

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2010

Abstract

Restricted Boltzmann Machines are difficult to train: Only an approximation of the gradient of the objective function is available. The objective function itself is intractable. Therefore most of the efficient convex optimisation techniques such as conjugate gradient descent or line search methods cannot be used. Using second order information is also not easy: evaluating the Hessian or an approximation of the Hessian is hard, and even if the Hessian is available, the dimensionality is usually too high to allow for inversion of the Hessian. Therefore RBMs are usually trained by stochastic gradient ascent, using a rough approximation of the gradient obtained by Gibbs sampling. It is known that learning by stochastic gradient ascent can be improved by including the covariance between stochastic gradients. However, this requires inverting a covariance matrix of size proportional to the dimensionality of the gradient. Recently, a new efficient approximate natural gradient method has been proposed. This method has shown to significantly improve training of ANNs, but hasn't been tried on RBMs yet. In this report I will show that natural learning can indeed improve the training of RBMs during the convergence phase. However, natural learning methods can lead to instabilities and very slow initial learning. If natural learning is to be used, it should not be used for the initial training phase. I will also describe how estimating the partition function by Annealed Importance Sampling (AIS) can lead to overly optimistic estimations of the data log-likelihood and develop some ideas how the estimation error may be kept small.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Benjamin Schwehn*)

Contents

List of Figures

Acknowledgements

First and foremost I would like to thank my supervisor Amos Storkey for his advice and great patience. I would also like to thank Joshua Bengio and Nicolas Le Roux for their immensely helpful suggestions. Finally, I would like to thank Jonathan Millin for his advice.

Nomenclature

α	step size/learning rate
$\bar{\mathcal{L}}$	empirical cost function (cost averaged over all training data)
β	alternative regularisation parameter for natural gradient methods. Higher values give more influence to the covariance information
$\boldsymbol{\mu}$	exponentially moving mean gradient
$\boldsymbol{d}$	direction given by training algorithm
$\boldsymbol{h}$	hidden nodes
$\boldsymbol{v}$	visible nodes
χ	momentum factor
γ	discount parameter for natural gradient methods. Lower values give less weight to past covariances
$\hat{\mathcal{L}}$	cost function approximated by averaging over a mini-batch of training data
κ	weight decay factor
λ	regularisation parameter for natural gradient methods. Lower values give more influence to the covariance information
$\tilde{\mathcal{L}}$	true cost function
b	bias term for visible node
c	bias term for hidden node
D	dimensionality of data set, number of visible nodes in a RBM
K	number of hidden nodes in a RBM
k	number of eigenvectors retained for low rank approximation
l	factor that scales the natural gradient so that its L2 norm becomes the same as that of the current underlying gradient
p	number of elements in a gradient

t	time step in an algorithm. Normally denoted with subscript (X_t). If the subscript is taken, denoted as super script (X^t)
W	weight parameters for RBMs
Z	partition function in a RBM
CD-n	contrastive divergence using n step of Gibbs sampling
natural CD-n	naïve natural gradient method, using CD-n as the underlying gradient
PCD-n	persistent contrastive divergence using n step of Gibbs sampling
TONGA-CD-n	TONGA using CD-n as the underlying gradient
meta-parameters	adjustable parameters for a training regime, *e.g.* step-size and weight decay
SGD	stochastic gradient descent
$\bar{\boldsymbol{g}}$	average gradient over full data set
$\boldsymbol{g}^*$	natural gradient
$\hat{\boldsymbol{g}}$	average gradient over samples from mini-batch

Chapter 1

Introduction

1.1 Motivation

Restricted Boltzmann Machines are graphical models used for unsupervised learning and also for dimensionality reduction and feature detection for supervised learning in combination with a classifier. While they are useful in itself, they've become more popular recently as a building block for larger, compositional models: deep belief nets. Restricted Boltzmann Machines can be trained much more efficiently than the more general (unrestricted) Boltzmann Machines. Still, they are quite difficult to train: Evaluating the objective function itself is intractable. Evaluating the gradient of the objective function is also intractable. Only an approximation of the gradient of the objective function is easily available. Therefore, efficient convex optimisation techniques such as conjugate gradient descent or line search methods are not easily applicable. Using second order information is also not easy: evaluating the Hessian or an approximation of the Hessian is hard, and even if the Hessian is available, the dimensionality is usually too high to allow for efficient inversion. Therefore, RBMs are usually trained by stochastic gradient ascent, using a rough approximation of the gradient obtained by Gibbs sampling. It is known that learning by stochastic gradient ascent can be improved by including the the covariance between stochastic gradients in a way reminiscent of Newton's method: the inverse of the covariance takes the place of the inverse of the Hessian. However, this requires inverting a covariance matrix of size proportional to the dimensionality of the gradient. This is far too expensive for larger-scale optimisation, as encountered in machine learning. Recently, a new efficient approximate natural gradient method has been proposed. This method has shown to significantly improve training of artificial neural networks [Le Roux et al., 2008]. This method hasn't been tried on RBMs yet even though the authors think "it should have been done for ages" ([Bengio, 2010]) and it's not clear if should work. There is a significant difference between RBMs and the ANNs used in [Le Roux et al., 2008]: In addition to the stochastic nature of on-line learning, the CD-1 gradient approximation RBMs are usually trained with is both noisy and biased. In practise, this additional noise may make it more difficult to extract useful information from the covariance between gradients. In theory, the fact that CD-1 is biased and in fact not a gradient of any function (*i.e.* following CD-1 may cause cycles) voids some of the mathematical justifications why natural gradient learning should work.

In this report I will show that training RBMs can indeed be improved by using the natural gradient. However, the algorithm is very sensitive to meta-parameter settings. Wrong meta-parameters can lead to catastrophic failure of training. It also probably shouldn't be used using fully on-line updates. Instead mini-batches should be used.

I should also disclose that my initial implementation contained a serious error that rendered most of my initial results invalid. The fact that I only discovered this mistake less than two weeks before the hand-in date resulted in me having to compromise on the experiments presented. In particular, some results are obtained from only single test runs. Given the stochastic nature of the algorithms used, we would normally prefer to obtain results from many runs using different initialisation. I also mostly present results obtained from training on a small data-set (down-sampled MNIST handwritten numbers). However, all results presented in this report are based on valid implementations and the more important insights are checked using results from at least 5 or 10 runs.

1.2 Outline

Chapter 2 gives the necessary background on the graphical model of Restricted Boltzmann Machines and the most common training method, contrastive divergence. Chapter 3 shows how the natural gradient is used in the context of stochastic gradient descent, and describes a fast, approximate, implementation: The Topmoumoute Online Natural Gradient Algorithm (Tonga). Chapter 4 shows if the natural gradient can be used to improve the performance of contrastive divergence on small scale problems and how approximations needed for computational reasons on large scale problems affect the training performance. Chapter 5 finishes with concluding remarks and ideas for future work.

Chapter 2

Background

2.1 Restricted Boltzmann Machines

Restricted Boltzmann machines (RBMs) [Smolensky, 1987] are undirected probabilistic networks, connecting a layer of hidden nodes to a layer of visible nodes. Every visible node is connected to every hidden node but no connections within either the hidden layer nor the visible layer are allowed. This restriction in connectivity makes efficient (approximate) training possible while maintaining the representational power of an universal approximator [Le Roux and Bengio, 2008]. Restricted Boltzmann machines have been used successfully for a variety tasks such as handwritten digit recognition [Hinton, 2000], face recognition [Teh and Hinton, 2000] and automated recommender systems [Salakhutdinov et al., 2007]. They are also often – and perhaps more successfully – used as individually trainable building blocks in multi layer networks (deep belief networks) used for tasks as diverse as digit recognition [Hinton, 2007] or speaker detection [Noulas and Kröse, 2008]. For classification problems, RBMs can either be used (with some modifications) directly (*e.g.* [Larochelle and Bengio, 2008]) or more commonly as a feature extraction layer to be used in combination with a classifier.

Unfortunately, using exact training methods based on the gradient of the training data likelihood function is intractable because the gradient cannot be calculated exactly and even approximating this gradient to a small ϵ is generally intractable. Thus a rough gradient approximation algorithm is used, the most popular being the Contrastive Divergence algorithm [Carreira-Perpinan and Hinton, 2005; Hinton, 2000]. The standard training regime then updates the model parameters by stochastic gradient descent, either fully on-line or using mini-batches.

2.2 Graphical Model – Energy Functions

As stated, Restricted Boltzmann Machines (RBMs) are undirected graphical models (Markov random fields) with a visible layer $\boldsymbol{v}$, consisting of D nodes and a hidden layer $\boldsymbol{h}$ consiting of K nodes, where no connections between nodes within either layer are allowed (Figure 2.1). Visible units represent the data, hidden units increase the modeling capacity.

Individual nodes are typically modelled as binary stochastic units: a node s_i is in state *on* (1) with a probability $P(s_i = 1)$ depending on a logistics function over its activation a_i or in state *off*

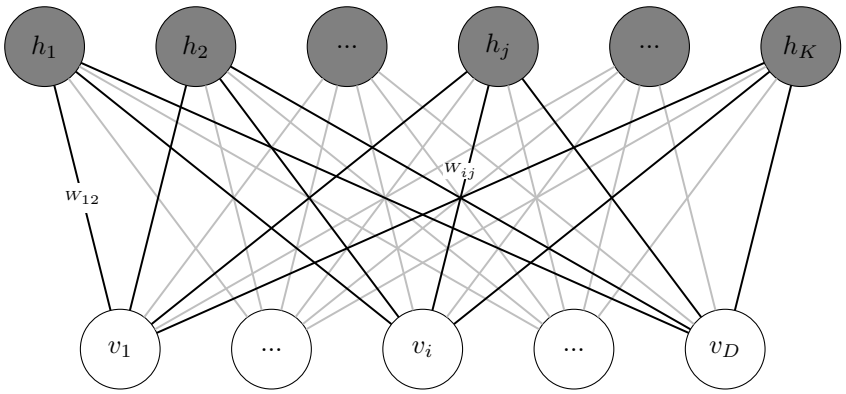


Figure 2.1: Schematic Restricted Boltzmann Machine (Biases not shown)

(0) otherwise:

$$P(s_i = 1) = \frac{1}{1 + e^{-a_i}} \tag{2.1}$$

with the activation a_i simply being the weighted sum of all connected nodes (*i.e.* all nodes in the other layer) plus a bias term b_i:

$$a_i = b_i + \sum_j s_j W_{ij} \tag{2.2}$$

If we sequentially update the probabilities for every node (*i.e.* use MCMC sampling), the system converges to its equilibrium state where the probabilities for each node don't change anymore. In this equilibrium state we define the energy of the joint configuration of the hidden and visible layer as the negative weighted sum of every node pair being *on* together minus the biases of all visible and hidden nodes. I will only call the bias term of the visible nodes $\boldsymbol{b}$ and denote the biases of hidden nodes as $\boldsymbol{c}$ to make the distinction more apparent:

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_{ij} v_i h_j W_{ij} - \sum_i v_i b_i - \sum_j h_j c_j \tag{2.3}$$

This energy function lets us write the joint probability of the system as:

$$P(\boldsymbol{v}, \boldsymbol{h}) = \frac{\exp(-E(\boldsymbol{v}, \boldsymbol{h}))}{\sum_{\boldsymbol{v}', \boldsymbol{h}'} \exp(-E(\boldsymbol{v}', \boldsymbol{h}'))} \tag{2.4}$$

where $\boldsymbol{v}' \in \mathcal{V}$ and $\boldsymbol{h}' \in \mathcal{H}$. $\mathcal{V}$ is the set of all configurations of the visible nodes and H the set of all possible configurations of the hidden nodes. The normalising constant in the denominator is the so called *partition function* Z. The likelihood of a data point is given by the marginal probability of the visible layer:

$$P(\boldsymbol{v}) = \frac{\sum_{\boldsymbol{h}} \exp(-E(\boldsymbol{v}, \boldsymbol{h}))}{\sum_{\boldsymbol{v}', \boldsymbol{h}'} \exp(-E(\boldsymbol{v}', \boldsymbol{h}'))} \tag{2.5}$$

In RBMs with binary nodes we can marginalise over the hidden nodes analytically and define $P(\boldsymbol{v})$ in terms of the *free energy* $F(\boldsymbol{v})$([Marlin et al., 2010]):

$$P(\boldsymbol{v}) = \frac{1}{Z} \exp(-F(\boldsymbol{v})) \tag{2.6}$$

with the partition function Z:

$$Z = \sum_{\boldsymbol{v}' \in \mathcal{V}} \exp(-F(\boldsymbol{v}')) \tag{2.7}$$

and the free energy:

$$F(\boldsymbol{v}) = -\left(\sum_{i=1}^{D} v_i b_i + \sum_{j=1}^{K} \log\left(1 + \exp\left(\sum_{i=1}^{N} v_i W_{ij} + c_j\right)\right)\right) \tag{2.8}$$

We can then write the average data log likelihood of a data set $\mathcal{D}$ as

$$\log(P(\mathcal{D})) = \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{v} \in \mathcal{D}} \left(\sum_{i=1}^{D} v_i b_i + \sum_{j=1}^{K} \log\left(1 + \exp\left(\sum_{i=1}^{N} v_i W_{ij} + c_j\right)\right)\right) - log(Z) \tag{2.9}$$

or, somewhat simpler, in vector notation:

$$\log(P(\mathcal{D})) = \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{v} \in \mathcal{D}} \left(\boldsymbol{v}^T \boldsymbol{b} + \sum_{j=1}^{K} \log\left(1 + \exp\left(\boldsymbol{v}^T W_j + c_j\right)\right)\right) - log(Z) \tag{2.10}$$

The first term, the unnormalised data log probability, can be evaluated in time polynomial to the dimensionality D of the data, the size of the hidden layer K and the size N of the data set. The partition function term Z (2.7), however, requires summing over all possible states of the visible nodes $\mathcal{V} = \{0, 1\}^D$ and therefore requires exponential time 2^D, making teh evaluation intractable even for relatively small data dimensionality. Observing that we can alternatively write the partition function as the sum of the free energy terms over all possible hidden states:

$$Z = \sum_{\boldsymbol{h}' \in \mathcal{H}} \exp(-F(\boldsymbol{h}')) \tag{2.11}$$

we can evalute Z in time 2^K, which will later allow us to evaluate Z for RBMs with a large visible, but small hidden layer.

2.3 Learning

When training a RBM we aim to maximise this average data probability $P(\boldsymbol{\mathcal{D}})$ for a given data set $\mathcal{D}$. A convenient fact in RBMs is that the gradient of the log likelihood for a data point is given by:

$$\frac{\partial \log p(\boldsymbol{v})}{\partial W_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \tag{2.12}$$

where the first term is the expectation of the two nodes $v_i h_j$ connected by weight W_{ij} being on together when the visible layer is "clamped" on the input data ("clamped phase") whereas the second term is this expectation when the clamp is removed ("free running phase"), in the equilibrium distribution. Since the nodes in the hidden layer are conditionally independent given the visible nodes, we can calculate the first term directly. Unfortunately, the second term is intractable to calculate. We can however try to approximate $\langle v_i h_j \rangle_{model}$ and the most common strategy is given in the next section.

The gradient for the biases is just a simplified version of 2.12:

$$\frac{\partial \log p(\boldsymbol{v})}{\partial b_j} = \langle h_j \rangle_{data} - \langle h_j \rangle_{model} \tag{2.13}$$

$$\frac{\partial \log p(\boldsymbol{h})}{\partial c_i} = \langle v_i \rangle_{data} - \langle v_i \rangle_{model} \tag{2.14}$$

2.3.1 Contrastive divergence

The most commonly used algorithm to approximate the gradient $\frac{\partial \log p(v)}{\partial W_{ij}}$ is the contrastive divergence approximation [Carreira-Perpinan and Hinton, 2005; Hinton, 2000] that approximates $\langle v_i h_j \rangle_{model}$ by performing t steps of Gibbs MCMC sampling. Again exploiting the conditional independence property of nodes in one layer given the nodes in the other layer, we can perform Gibbs sampling in two half steps:

- *Step 0:* Before the first half step, we set the state of the visible units $\boldsymbol{v}^{(0)}$ to any (*e.g.*. random) state
- *Step 1.1:* Given the state of $\boldsymbol{v}^{(0)}$ we can directly calculate the conditional probability for each hidden node $P(h_i = 1|\boldsymbol{v}^{(0)})$. We then set the state of each hidden node by sampling from this distribution, obtaining the state $\boldsymbol{h}^{(1)}$
- *Step 1.2:* Given $\boldsymbol{h}^{(1)}$ we calculate the conditional probability for each visible node $P(v_i = 1|\boldsymbol{h}^{(1)})$. We again set the state of each visible note by sampling from this conditional distribution to obtain $\boldsymbol{v}^{(1)}$ and continue with the next half step.

Calculating the expectation of $\langle v_i h_j \rangle$ over an infinite number of steps would give us the $\langle v_i h_j \rangle_{model}$ term required for the exact gradient, and a large number of steps would at least give a good approximation. But given that we have only limited time, contrastive divergence uses the following two adjustments to the general procedure:

1. We start the sampling at step 0 not by setting the visible nodes to any state, but instead start at the data distribution: we set the visible nodes to the state given by the input. For binary input we can directly set the state of the visible nodes to be the binary value of the input. For input $\in (0, 1)$, we use sampling from a Bernoulli distribution with mean given by the input. If the data distribution is close to the model distribution, this allows for faster convergence to the model distribution compared to when starting at an arbitrary state [Bengio et al., 2007].
2. We stop sampling after only a small number of Gibbs sampling steps n. In fact the most commonly used number of steps is only 1. This training regime is called CD-1 and it works surprisingly well [Carreira-Perpinan and Hinton, 2005].

Figure 2.2 shows an illustration of the CD algorithm.

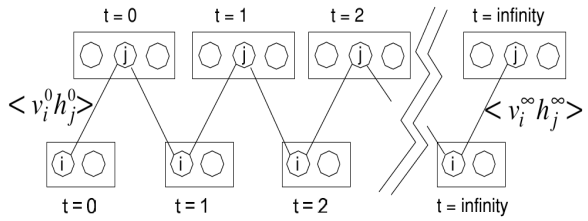


Figure 2.2: Contrastive Divergence (from [Hinton et al., 2006])

2.3.2 Properties of contrastive divergence

CD-n uses n steps of the Gibbs MCMC sampling routine described in the previous section. That is, CD-n uses the approximation:

$$\begin{aligned}\frac{\partial \log p(v)}{\partial W_{ij}} &= \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \\ &\approx P(h_j^{(0)} = 1 | \boldsymbol{v}^{(0)}) v_i^{(0)} - P(h_j^{(n)} = 1 | \boldsymbol{v}^{(n)}) v_i^{(n)}\end{aligned} \tag{2.15}$$

For $n \to \infty$, the second term converges to $\langle v_i h_j \rangle_{model}$ and CD-n converges to maximum likelihood learning. For the low number of steps used in practise ($n = 1$ for CD-1), the second term is biased by the fact that we started sampling with a data point and abort the sampling long before the Markov chain has had the opportunity to converge. However, Carreira-Perpinan and Hinton [2005] have shown this bias to be low for most data distributions and that fixed points for CD-1 generally exist, and while not coinciding with the fixed points of the true gradient, the fixed points of CD and the true gradient are, however, often close to one another. Sutskever and Tieleman [2010]; Tieleman [2007] have shown that CD is in fact not a gradient at all for any function and that theoretically fixed points must not exist (and thus training by CD-n may not converge even when using a suitably decaying step size regime), though this never seems to be a problem in practise: CD-n works well for training RBMs.

2.3.3 Persistent Contrastive Divergence

Tieleman [2008] notes that while CD with a low number of steps gives a reasonable approximation to the likelihood gradient, the small number of steps do not let the Markov chain, that is used to approximately sample from $\langle v_i h_j \rangle_{model}$ and that in CD starts at the data distribution, move far away from the data distribution $\langle v_i h_j \rangle_{data}$. He proposes an alternative method that does not start sampling at each iteration at a state given by the input data, but instead initialises a number of Monte Carlo chains for the Gibbs sampling at a random state and then at each iteration reuses the state where it ended at the last iteration. Since the model only changes a little in-between parameter updates, the previous chains should already be closer to the current model distribution compared to when restarting the chains at the data distribution for each iteration. This Persistent Contrastive Divergence (PCD) training method appears to often perform better than CD and can also result in a better generative model.

2.3.4 Parameter updates and additional meta-parameters

Both CD and PCD then use a simple gradient ascent method to for the weight parameter update:

$$W_{ij}^{(t+1)} = W_{ij}^{(t)} + \alpha \left(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_t \right) \tag{2.16}$$

and analogous for the biases with step size α customary called the *learning rate*.

This is commonly augmented ([Hinton, 2010]) by adding a momentum term to the current update that includes the weighted previous update:

$$\Delta_t W_{ij} = \alpha \left(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_t \right) + \chi \Delta_{t-1} W_{ij} \tag{2.17}$$

The momentum term helps avoid zig-zagging along valleys that gradient descent methods often suffer from.

Another common augmentation is to add an additional weight decay factor that penalises large parameter values to discourage over-fitting. It is common to either use an L1 weight decay term, that subtracts a term from every parameter update proportional to the current parameter value:

$$\Delta_t W_{ij} = \alpha \left(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_t \right) - \alpha \kappa W_{ij} \tag{2.18}$$

or an L2 weight decay term based on the square of the current parameter value:

$$\Delta_t W_{ij} = \alpha \left(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_t \right) - \alpha \kappa W_{ij}^2 \operatorname{sgn}(W_{ij}) \tag{2.19}$$

It should be noted that the weight decay term is also multiplied by the step size α in addition to the weight decay parameter κ. The main reason for using weight decay is to improve generalisation and to help "un-stick" units that are in state 1 with probabilities very close to 0 or 1. [Hinton, 2010] gives additional background and advise for choosing good values for the meta-parameters.

Chapter 3

Natural Stochastic Gradient Descent

3.1 Learning and optimisation

3.1.1 General optimisation methods

In a typical optimisation problem we try to find the parameters $\boldsymbol{\theta}$ for which an objective function $f(\boldsymbol{\theta})$ is (locally or globally) minimised or maximised. A common class of algorithms used for iterative optimisation are *line search methods*. In general, line search methods start at parameter vector $\boldsymbol{\theta}_t$ and move to a new parameter vector $\boldsymbol{\theta}_{t+1}$ by moving a step size α into a direction $\boldsymbol{d}$:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t \boldsymbol{d}_t \tag{3.1}$$

The problems lies in finding a good descent direction $\boldsymbol{d}$ and step size α. If f is differentiable, the simplest algorithm in this class is *gradient descent*, where direction $\boldsymbol{d}$ is set to the steepest descent direction at the current point in parameter space $\boldsymbol{\theta}$ given by the gradient of the objective function f at point $\boldsymbol{\theta}$: $-\nabla f(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t \nabla f(\boldsymbol{\theta}_t) \tag{3.2}$$

If f is twice continuously differentiable we can use the *Newton direction*. The Newton direction is derived by minimising the second order Taylor expansion of $f(\boldsymbol{\theta} + \boldsymbol{d})$

$$f(\boldsymbol{\theta}_t + \boldsymbol{d}) \approx f(\boldsymbol{\theta}_t) + \boldsymbol{d}^T \nabla f(\boldsymbol{\theta}_t) + \frac{1}{2} \boldsymbol{d}^T \nabla^2 f(\boldsymbol{\theta}) \boldsymbol{d} \tag{3.3}$$

by setting its derivative to zero:

$$\nabla f(\boldsymbol{\theta}_t) + \nabla^2 f(\boldsymbol{\theta}) \boldsymbol{d} = 0$$

$$\boldsymbol{d} = -(\nabla^2 f(\boldsymbol{\theta}_t))^{-1} \nabla f(\boldsymbol{\theta}_t) \tag{3.4}$$

$$\boldsymbol{d} = -H^{-1} \nabla f(\boldsymbol{\theta}_t) \tag{3.5}$$

For batch optimisation, Newton methods enjoy a much faster convergence rate to the optimum (quadratic vs linear)([Bottou, 2003]). Newton's direction has a natural step length α of 1 (because

for that step size, the quadratic approximation is minimised) that is commonly used, but other step-sizes may be more appropriate, after all, the underlying quadratic approximation may not be very good fit the the objective function. Other line search methods need to find good values for α, typically by trying candidate values for α and checking for the satisfaction of certain conditions, such as the Wolfe conditions. These optimisation methods are well documented, *e.g.* in [Nocedal and Wright, 2000].

3.1.2 Optimisation in machine learning

A common application of optimisation techniques in the context of machine learning is the problem of minimising a cost function $\tilde{\mathcal{L}}(\boldsymbol{\theta})$ that is defined as the expected value of a loss function L over parameters $\boldsymbol{\theta}$ and random variable $\boldsymbol{x}$ with distribution $p(\boldsymbol{x})$:

$$\tilde{\mathcal{L}}(\boldsymbol{\theta}) = \int_x L(\boldsymbol{\theta}, \boldsymbol{x}) p(\boldsymbol{x}) dx \tag{3.6}$$

In machine learning, the distribution p is over the all possible data inputs and normally not known analytically. Therefore, we can only approximate $\tilde{\mathcal{L}}$ by averaging over all data points x_i drawn from a training data set $\mathcal{D}$:

$$\bar{\mathcal{L}}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{x_i \in \mathcal{D}} L(\boldsymbol{\theta}, \boldsymbol{x_i}) \tag{3.7}$$

If we can calculate the gradient $\boldsymbol{g}_i(\boldsymbol{\theta})$ (leaving the dependence of $\boldsymbol{g}_i$ on $\boldsymbol{x}_i$ implicit in the subscript notation to simplify the notation) of the cost function at sample $\boldsymbol{x}_i$:

$$\boldsymbol{g}_i(\boldsymbol{\theta}) = \frac{\partial L(\boldsymbol{\theta}, \boldsymbol{x}_i)}{\partial \boldsymbol{\theta}} \tag{3.8}$$

we can calculate the average gradient over a data set of size N:

$$\bar{\boldsymbol{g}}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{g}_i(\boldsymbol{\theta}) \tag{3.9}$$

and use $\bar{\boldsymbol{g}}(\boldsymbol{\theta})$ as the direction $\boldsymbol{p}$ in formula 3.1 for *batch gradient descent*:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t \bar{\boldsymbol{g}}(\boldsymbol{\theta}) \tag{3.10}$$

However, this requires computing the average gradient over the entire data set for every single update. This makes batch gradient descent too expensive for large scale learning.

3.1.3 On-line stochastic gradient descent

Instead, *on-line stochastic gradient descent* picks only single data point $\boldsymbol{x}_i \in \mathcal{D}$ at random, and then immediately uses the gradient $\boldsymbol{g}_i(\boldsymbol{\theta})$ as the direction $\boldsymbol{d}$ in formula 3.1

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t \boldsymbol{g}_i(\boldsymbol{\theta}), \tag{3.11}$$

with a much smaller step size α compared to batch gradient descent. The hope is, that many such small updates result in good optimisation on average.

3.1.4 Mini-batch gradient descent

Often an intermediate between full on-line descent and batch descent is used: Instead of using only a single data point (as in fully on-line gradient descent) or the mean gradient over the full training data set $\mathcal{D}$ (as in batch gradient descent), we optimise the cost function over over a small, random subset of of $\mathcal{D}$. If the subset contains S data-points (usually $S \ll N$), the cost function over the mini-batch is given by:

$$\hat{\mathcal{L}}(\boldsymbol{\theta}) = \frac{1}{S}\sum_{x_i}^{S} L(\boldsymbol{\theta}, \boldsymbol{x_i}) \tag{3.12}$$

and the mini-batch gradient $\hat{\boldsymbol{g}}$ by

$$\hat{\boldsymbol{g}}(\boldsymbol{\theta}) = \frac{1}{S}\sum_{i=1}^{S} \boldsymbol{g}_i(\boldsymbol{\theta}) \tag{3.13}$$

and the update rule becomes:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t \hat{\boldsymbol{g}}(\boldsymbol{\theta}) \tag{3.14}$$

It is not uncommon to include mini-batch learning when talking about "on-line" methods. To make the distinction more clear, I will call on-line learning using only a single data-point "fully on-line" whenever the distinction is relevant. When algorithms are given in the mini-batch formulation, they trivially reduce to fully on-line learning by setting the batch size S to 1.

3.1.5 On-line versus batch learning

At this point we should note an important difference between machine leaning and optimisation: the goal in machine learning is to minimise the cost function $\mathcal{L}$ not on the training set, but on unknown future data. We thus differentiate between the *empirical* cost on the training data and the *expected* cost on unknown data. If the training data set is large enough, optimising the empirical cost gives also a good optimisation for the expected cost. But if only limited training data is available, methods performing best in optimising the empirical cost $\bar{\mathcal{L}}$ may generalise badly to future data and do not necessarily make the best learning algorithms (a problem known as *overfitting*).

Bottou [2003] shows that, even though on-line methods may perform much worse than batch methods when measured by convergence of the *empirical* cost to the optimum, they may actually optimise the *expected* cost better. This good generalisation property and the great computational advantage on large data sets explain why on-line stochastic gradient descent is normally preferred over batch gradient descent in machine learning. In fact, Bottou and LeCun [2004] show that in tasks where data is abundant, suitable on-line methods outperform any batch method.

3.1.6 Using second order information

If available, we can also include second order information, just like in batch learning, using the *second order stochastic gradient descent* update rule:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t H^{-1} \hat{\boldsymbol{g}}_i(\boldsymbol{\theta}). \tag{3.15}$$

with H denoting the Hessian or an approximation of the Hessian obtained from the mini-batch.

3.1.7 Step size selection

When the evaluation of $\mathcal{L}(\boldsymbol{\theta})$ is intractable and an approximation of $\mathcal{L}(\boldsymbol{\theta})$ using formulas 3.7 or 3.12 is costly, it is intractable to use one of the standard line search methods to find a good step size parameter α. A common strategy is to try different values for α on a validation set in order to find a reasonable fixed value for α. If we want to ensure convergence, we must use a decreasing step size subject to the well known conditions $\sum \alpha_t^2 < \infty$ and $\sum \alpha_t = \infty$. One common regime satisfying these two conditions is to use a step size decaying proportional to the inverse of the optimisation time: $\alpha \propto \frac{1}{t}$.

3.2 The Natural Gradient

If we use formula 3.13 to get the average gradient over a set of stochastic gradients, it may seem wasteful to compute many gradients, but then only retain their mean. The natural gradient method tries to use more information obtained from calculating the gradients by multiplying the mean gradient $\bar{\boldsymbol{g}}$ over the full data set by the inverse of the covariance C over the individual gradients:

$$\begin{aligned} C &= \mathbb{E}[(\boldsymbol{g} - \bar{\boldsymbol{g}})(\boldsymbol{g} - \bar{\boldsymbol{g}})^T] \\ &= \frac{1}{N}\sum_{i}^{N}(\boldsymbol{g_i} - \bar{\boldsymbol{g}})(\boldsymbol{g_i} - \bar{\boldsymbol{g}})^T] \end{aligned} \tag{3.16}$$

$$\boldsymbol{g}^* = C^{-1}\bar{\boldsymbol{g}} \tag{3.17}$$

Intuitively, this has the effect of weighting components of the gradient by the certainty of that component as schematically shown in figure 3.1.

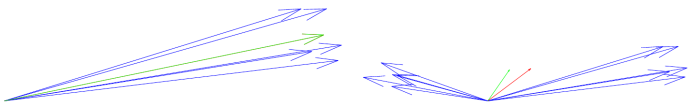


Figure 3.1: Left: If stochastic gradients have low variance, the natural gradient (green) almost coincides with the mean gradient. **Right**: If stochastic gradients show higher variance in one direction, the natural gradient gives lower weight to this direction and points more towards the direction in which the gradients agree (from [Le Roux et al., To Appear 2010])

Since C is by construction positive *semi*-definite, it may not be invertible. To make C invertible, we can add regularisation terms along the diagonal by adding λI. Adding λI does not change the eigenvectors of the matrix, but only adds λ to all eigenvalues. Since C is Hermitian and semi-definite, all its eigenvalues are non-negative. Adding λI with $\lambda > 0$ thus makes C positive definite and invertible. The direction of the regularised natural gradient $\boldsymbol{g}^*$ is then:

$$\boldsymbol{g}^* = (C + \lambda I)^{-1}\bar{\boldsymbol{g}}. \tag{3.18}$$

While this is the notation used in [Le Roux et al., 2008], using this regularisation has an important drawback: for covariance matrices with low or zero eigenvalues (this will be very

common), multiplying a gradient by the inverse of the regularised matrix has the effect of increasing the norm of this gradient by a factor of $\approx \frac{1}{\lambda}$. So for low values of λ, the norm of the resulting natural gradient is much larger than that of the underlying gradient. We then have to either re-normalise the gradient or adjust our learning rate accordingly.

An alternative regularisation is:

$$\boldsymbol{g}^* = (\beta C + I)^{-1}\bar{\boldsymbol{g}}. \tag{3.19}$$

This of course has a similar effect for large values of β, but for the usual settings for λ or β, this regularisation results in a natural gradient with a norm more comparable to the underlying gradient, and is also used in [Le Roux et al., To Appear 2010].

An important observation is that the dot product of natural and stochastic gradient is greater than zero (because $(C + \lambda I)$ is positive definite):

$$\begin{aligned} \bar{\boldsymbol{g}}^T(C + \lambda I)^{-1}\bar{\boldsymbol{g}} &> 0 \\ \bar{\boldsymbol{g}}^T\boldsymbol{g}^* &> 0 \end{aligned} \tag{3.20}$$

Therefore, the natural gradient $\boldsymbol{g}^*$ direction never diverges more than 90° degrees from $\bar{\boldsymbol{g}}$. As long as $\bar{\boldsymbol{g}}$ is the true gradient of the objective function f, updating in $\boldsymbol{g}^*$ direction for a sufficiently small step will always decrease f.

3.2.1 Bayesian justification for the natural gradient

[Le Roux et al., 2008] outline the following Bayesian justification for the natural gradient:

We get the approximate mean gradient $\hat{\boldsymbol{g}}$ over a batch of data by averaging gradient samples $\boldsymbol{g}_i$ using formula 3.13. If we assume the samples $\boldsymbol{g}_i$ to be independent and identically distributed (i.i.d.) then the central limit theorem gives that in the limit of infinite batch size, the samples are normally distributed around the true mean $\tilde{\boldsymbol{g}}$:

$$\boldsymbol{g} \sim \mathcal{N}\left(\tilde{\boldsymbol{g}}, \frac{\tilde{C}}{n}\right) \tag{3.21}$$

where $\tilde{C}$ is the true covariance matrix:

$$\tilde{C} = \mathbb{E}[(\boldsymbol{g} - \tilde{\boldsymbol{g}})(\boldsymbol{g} - \tilde{\boldsymbol{g}})^T] \tag{3.22}$$

In the Bayesian setting we reasonably define the prior distribution over $\boldsymbol{g}$ as a Gaussian with zero mean, and isotropic covariance $\sigma^2 I$:

$$\boldsymbol{g} \sim \mathcal{N}\left(0, \sigma^2 I\right) \tag{3.23}$$

After observing a number n gradients with average $\bar{\boldsymbol{g}}$ we can calculate mean and covariance of posterior distribution, using the fact that Gaussians are conjugate prior. The posterior mean is given by (cmp. *e.g.* [Gelman et al., 2003]):

$$\left(\Sigma_0^{-1} + n\Sigma^{-1}\right)^{-1}\left(\Sigma_0^{-1}\boldsymbol{\mu}_0 + n\Sigma^{-1}\bar{\boldsymbol{x}}\right) \tag{3.24}$$

and the posterior covariance by:

$$\left(\Sigma_0^{-1} + n\Sigma^{-1}\right)^{-1} \tag{3.25}$$

Plugging in $\boldsymbol{\mu}_0 = 0$, $\Sigma_0 = \sigma^2 I$, $\bar{\boldsymbol{x}} = \bar{\boldsymbol{g}}$ and $\Sigma = \frac{\tilde{C}}{n}$ we get for the posterior covariance:

$$C = \left(\frac{n}{\tilde{C}} + \frac{I}{\sigma^2}\right)^{-1} \tag{3.26}$$

and posterior mean:

$$\mu = \left(\frac{\tilde{C}}{n\sigma^2} + I\right)^{-1} \bar{\boldsymbol{g}} \tag{3.27}$$

Therefore:

$$\tilde{g}|\bar{\boldsymbol{g}}, \tilde{C} \sim \mathcal{N}\left(\left(\frac{\tilde{C}}{n\sigma^2} + I\right)^{-1} \bar{\boldsymbol{g}}, \left(n\tilde{C}^{-1}\right)^{-1} + \frac{I}{\sigma^2}\right) \tag{3.28}$$

If we now want to choose a direction $\boldsymbol{d}$, so that $\mathbb{E}[\boldsymbol{d}^T\tilde{\boldsymbol{g}}]$ is maximised/minimised, clearly we should choose $\boldsymbol{d}$ to be:

$$\begin{aligned} \boldsymbol{d} &\propto \pm\mathbb{E}[\tilde{\boldsymbol{g}}] \\ &\propto \pm\left(\frac{\tilde{C}}{n\sigma^2} + I\right)^{-1} \bar{\boldsymbol{g}} \end{aligned} \tag{3.29}$$

which is equivalent to the direction of the regularised natural gradient in 3.18 for some λ. Thus, using the natural gradient is nicely justified by a Bayesian argument.

3.3 Natural gradient descent

Using the natural gradient direction 3.17 as the direction in our standard optimisation method 3.1 gives as the update rule (we subtract if we want to minimise)

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t C^{-1}\bar{\boldsymbol{g}}_t \tag{3.30}$$

We may observe that 3.30 is very similar to Newton's method stated in 3.5 and wonder if the covariance C ought to be thought of as an approximation of the Hessian H. Le Roux [2010a] argues that this is not the case: The Hessian measures the curvature and thus how different the gradient would be if we moved a small step in parameter space. The covariance on the other hand measures how different the gradient is under the influence of slight variations in the training data. In fact, Le Roux [2010a]; Le Roux et al. [To Appear 2010] report good results *combining* the natural gradient with Newton's method, using the update rule:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t \left(C^H + \lambda I\right)^{-1} \bar{\boldsymbol{d}}_t \tag{3.31}$$

where $\bar{\boldsymbol{d}}$ is the Newton direction $\bar{\boldsymbol{d}}_t = H^{-1}\bar{\boldsymbol{g}}$ and C^H the covariance matrix of the Newton directions. If the Hessian or a good approximation of the Hessian can be computed efficiently, this combined method can perform better than either using the first-order natural gradient or Newton's method alone.

3.3.1 Mini-batch natural gradient descent

For stochastic natural gradient descent using mini-batches we calculate the average gradient $\hat{\boldsymbol{g}}$ over a small number S of training points (S can be 1 for fully on-line learning). We could now simply use the covariance over all the gradients in the current mini-batch. However, the small number of gradients may not give a good approximation of the covariance C. Instead we use an exponentially moving mean of the batch gradients:

$$\boldsymbol{\mu}_t = \gamma \boldsymbol{\mu}_{t-1} + (1-\gamma)\hat{\boldsymbol{g}}_t \tag{3.32}$$

and then use an exponentially moving mean of the covariance:

$$\begin{aligned} C_t &= \gamma C_{t-1} + (1-\gamma)\left(\hat{\boldsymbol{g}}_t - (\gamma\boldsymbol{\mu}_{t-1} + (1-\gamma)\hat{\boldsymbol{g}}_t)\right)\left(\hat{\boldsymbol{g}}_t - (\gamma\boldsymbol{\mu}_{t-1} + (1-\gamma)\hat{\boldsymbol{g}}_t)\right)^T \\ &= \gamma C_{t-1} + \gamma(1-\gamma)(\hat{\boldsymbol{g}}_t - \boldsymbol{\mu}_{t-1})(\hat{\boldsymbol{g}}_t - \boldsymbol{\mu}_{t-1})^T \end{aligned} \tag{3.33}$$

The parameter $\gamma \in (0, 1]$ controls how much older gradients are discounted. Since the parameters $\boldsymbol{\theta}$ of the objective function is changing between mini-batches as we make updates, we want older gradients to have progressively less influence than gradients obtained in the current batch. This decay rule is used in [Le Roux et al., To Appear 2010], but is has the problem that weighting the second term by $(1-\gamma)$ has the effect that the gradient at step 0 gets a higher weight than following gradients. I therefore do not weight the second term by $(1-\gamma)$ and instead use:

$$\boldsymbol{\mu}_t = \gamma \boldsymbol{\mu}_t + \hat{\boldsymbol{g}}_t \tag{3.34}$$

$$C_t = \gamma C_{t-1} + (\hat{\boldsymbol{g}}_t - \frac{1}{r}\boldsymbol{\mu}_t)(\hat{\boldsymbol{g}}_t - \frac{1}{r}\boldsymbol{\mu}_t)^T \tag{3.35}$$

and normalising the mean and also the covariance by a factor $r = \sum_{i=0}^{t-1} \gamma^i$ when appropriate to keep the magnitudes constant. This is an important point to note when trying to compare meta-parameter settings for the parameter γ.

Algorithm 1 outlines a simple natural stochastic gradient descent method using this approach.

3.4 Topmoumoute Online Natural Gradient (TONGA)

Two issues make using simple natural stochastic gradient descent approach infeasible on large scale problems:

- We need to calculate and store the full covariance matrix C. For gradients of size n, this requires $O(n^2)$ storage which is too expensive for large n
- We need to either invert the full covariance matrix C or solve $\boldsymbol{g}^* C = \hat{\boldsymbol{g}}$ at each mini-batch iteration. C has size $n \times n$, and those operations take typically $O(n^3)$, far too expensive for large n.

The topmoumoute online natural gradient (TONGA) first described in [Le Roux et al., 2008] deals with both issues by

- Only keeping a low rank approximation of C

Algorithm 1 Simple stochastic natural gradient descent

$r \leftarrow 1$
repeat
  get mini-batch t
  $r \leftarrow r + \gamma^{t-1}$
  $G \leftarrow$ gradients from mini-batch
  $\hat{\boldsymbol{g}} \leftarrow \frac{1}{S}\sum_i^S G_i$ {mini-batch gradient}
  if $t = 0$ **then**
    $\boldsymbol{\mu} \leftarrow \hat{\boldsymbol{g}}$
    $C \leftarrow \hat{\boldsymbol{g}}\hat{\boldsymbol{g}}^T$
  else
    $\boldsymbol{\mu} \leftarrow \gamma\boldsymbol{\mu} + \hat{\boldsymbol{g}}$
    $C \leftarrow \gamma C + (\hat{\boldsymbol{g}} - \frac{1}{r}\boldsymbol{\mu})(\hat{\boldsymbol{g}} - \frac{1}{r}\boldsymbol{\mu})^T$
  end if
  $\boldsymbol{d} \leftarrow (\frac{1}{r}\beta C + I)^{-1}\hat{g}$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha\boldsymbol{d}$ {gradient descent rule with step size α}
until stop criterion

- Never actually inverting C or even calculating C. Instead the algorithm only requires the inversion and eigen-decomposition of a much smaller matrix G.
- Using a cheap update rule for the low rank approximation of C

In the following section I present the TONGA algorithm as set up in [Le Roux et al., To Appear 2010] in more detail.[1] Since this publication is still in pre-print, the interested reader should refer to [Le Roux et al., 2008] instead that gives a similar exposition. The main difference between both references is that the latter uses the uncentred covariance as described in section 3.4.1.

The first idea of the algorithm is to obtain the natural gradient without actually having to invert C. In the simple natural SGD, we used the update rule 3.35 to update the covariance between mini-batches (omitting the normalisation by r for clarity):

$$C_t = \gamma C_{t-1} + (\hat{\boldsymbol{g}}_t - \boldsymbol{\mu}_{t-1})(\hat{\boldsymbol{g}}_t - \boldsymbol{\mu}_t)^T \tag{3.36}$$

Equivalently we can write C_t as the product $X_t X_t^T$ with

$$\begin{aligned} X_1 &= \hat{\boldsymbol{g}}_1 \\ X_t &= [\sqrt{\gamma}X_{t-1} \qquad \hat{\boldsymbol{g}}_t - \boldsymbol{\mu}_t] \end{aligned} \tag{3.37}$$

Using X we can write the natural direction $\boldsymbol{g}^*$ as:

$$\begin{aligned} \boldsymbol{g}_t^* &= (\beta C_t + I)^{-1}\boldsymbol{g}_t \\ &= (\beta X_t X_t^T + I)^{-1} X_t \boldsymbol{y}_t + \boldsymbol{\mu}_t \qquad \text{with } \boldsymbol{y}_t = [0, \ldots, 0, 1]^T \end{aligned} \tag{3.38}$$

[1] Again, I use the alternative formulation that does not discount the second term of moving averages by $(1 - \gamma)$

Assuming that $\boldsymbol{g}^*$ has the form $\boldsymbol{g}^* = X_t\boldsymbol{x} + \boldsymbol{\mu}_t$ for some vector $\boldsymbol{x}$ we find $\boldsymbol{x}$:

$$\begin{aligned} X_t\boldsymbol{x} + \boldsymbol{\mu}_t &= (\beta X_t X_t^T + I)^{-1} X_t \boldsymbol{y}_t + \boldsymbol{\mu}_t \\ X_t\boldsymbol{x} &= X_t\boldsymbol{y}_t - \beta(X_t X_t^T X_t\boldsymbol{x} + X_t X_t^T \boldsymbol{\mu}_{t-1}) && \text{(assuming X to be full rank)} \\ \boldsymbol{x} &= (\beta X^T X + I)^{-1}(\boldsymbol{y}_t - \beta X^T \boldsymbol{\mu}_t) && (3.39) \end{aligned}$$

Thus the natural gradient is given by

$$\boldsymbol{g}_t^* = X_t \left(\beta X^T X + I\right)^{-1} \left(\boldsymbol{y} - \beta X^T \boldsymbol{\mu}_{t-1}\right) + \boldsymbol{\mu}_{t-1} \tag{3.40}$$

Using this transformation we can now calculate the natural gradient using a matrix inversion of the(regularised) gram matrix $G = X_t^T X_t$ instead of $C = X_t X_t^T$. As long as the number of columns in X_t is much smaller than the number of rows, inverting G is much cheaper than inverting C. Initially, X_1 only contains 1 column: the first on-line gradient. Inverting G is a trivial operation at this stage. At later time steps we use formula 3.37 to update X_t. This lets the number of columns of X_t grow by one for every observed gradient. Clearly we cannot let X_t grow too large, or else inverting G will actually become more expensive than inverting C. This brings the second idea of the Tonga algorithm: To keep the computational cost down, the authors propose replacing X_t with a corresponding low rank approximation $\hat{U}_t$. This low rank approximation is obtained by calculating the eigenvectors and eigenvalues of C and then keeping only the first k eigenvectors that correspond to the k largest eigenvalues. This eigendecomposition of C let us then obtain the low rank approximation $\hat{U}_t$ so that $\hat{U}_t\hat{U}_t^T \approx C$. Since the goal is to avoid having to calculate C at all, let alone do a direct eigendecomposition on it, we need to find an alternative way to calculate this low rank approximation. It turns out that we can calculate the eigendecomposition of C using only G. To obtain $\hat{U}_t$, let's consider the compact SVD of X_t

$$X_t = UDV^T \tag{3.41}$$

then

$$\begin{aligned} C_t = X_t X_t^T &= UDV^T VD^T U^T && (V^T \text{ orthogonal, } D \text{ diagonal}) \\ &= UD^2U^T && (3.42) \end{aligned}$$

and analogous

$$G_t = X_t^T X_t = VD^2V^T \tag{3.43}$$

So U contains the eigenvectors of C and V the eigenvectors of G. Now, if we right multiply 3.41 by V and D^{-1}:

$$X_t V D^{-1} = U \tag{3.44}$$

so we can write C in terms of X_t and V by plugging into 3.42:

$$C_t = (X_t V D^{-1}) D^2 (X_t V D^{-1})^T \tag{3.45}$$

So if we take the first k eigenvectors of V into $\hat{V}$ and the first k eigenvalues of D into $\hat{D}$, and define $\hat{U} = X_t\hat{V}\hat{D}^{-1}$. we obtain an low rank approximation of C:

$$C \approx \hat{C} = \hat{U}\hat{D}^2\hat{U}^T \tag{3.46}$$

without ever even computing C. Instead, we calculate V, $\hat{V}$, D, $\hat{D}$ from G using the SVD (3.43). $\hat{U}$ then replaces X_t in update rule 3.37 and the eigendecomposition gives us an efficient way to calculate C^{-1}:

$$C^{-1} = U^T D^{-2} U \tag{3.47}$$

$$\approx \hat{U}^T \hat{D}^{-2} \hat{U}. \tag{3.48}$$

where the inverse of D^2 is trivial to calculate since D^2 is diagonal. Since regularisation along the diagonal (3.18) only adds to the eigenvalues, we can use

$$(C + \lambda I)^{-1} \approx \hat{U}^T (\hat{D} + \sqrt{\lambda} I)^{-2} \hat{U} \tag{3.49}$$

for regularisation.

This give us the following algorithm: We calculate the eigendecomposition of G at every step. We calculate $\hat{C}^{-1}$ and $\boldsymbol{g}^*$ from the eigendecomposition. Instead of letting X_t grow, we replace X_t by $\hat{U}$ before appending the next gradient. The size of G remains constant at $k+1$ and the cost of the eigendecomposition is in $O(k^3)$. The cost of calculating the eigenvectors of C is $O(pk^2)$ where p denotes the number of elements in the gradient $\boldsymbol{g}$ (and thus the number of rows in X. This gives a total cost of $O(k^3 + pk^2)$. The cost of calculating C^{-1} directly is $O(p^3)$. Therefore, calculating the eigendecomposition of G is much more efficient as long as $k \ll p$.

The cost of calculating the natural gradient from 3.40 is $O(k^3)$ for the inversion plus $O(pk)$ for the multiplication. As long as we keep k much smaller than the square root of the number of elements in the gradient ($k^2 \ll p$), this reduces to $O(pk)$. We can see that this operation is much cheaper than the eigendecomposition (as long as $k^2 \ll p$). Therefore a further improvement is to not calculate the eigendecomposition at every step, but instead let X grow in-between eigendecompositions for b steps and use 3.40 to calculate $\boldsymbol{g}^*$. After b steps, we apply the eigendecomposition, to keep X from growing too large. The authors give the average cost per gradient when using a mini-batch of size m, as $O\left(p\left(m + k + b + \frac{(k+b)^2}{b}\right)\right)$ ([Le Roux et al., To Appear 2010]. Standard mini-batch gradient descent is in $O(pm)$. Therefore, TONGA is within the same order of magnitude as standard mini-batch gradient descent, as long as $k + b$ is close to m. Algorithm 2 shows the complete algorithm.

3.4.1 Using the uncentered covariance

In an alternative formulation of the natural gradient we use the *uncentred* covariance instead of the normal, centred, covariance. The only change necessary is to replace $\boldsymbol{\mu}$ by $\mathbf{0}$ in the algorithms above. In particular, the uncentred covariance is given by:

$$\begin{aligned} \tilde{C} &= \mathbb{E}[\boldsymbol{g}\boldsymbol{g}^T] \\ &= \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{g_i} \boldsymbol{g_i}^T, \end{aligned} \tag{3.50}$$

the simple covariance update rule 3.35 becomes

$$C_t = \gamma C_{t-1} + \boldsymbol{g}\boldsymbol{g}^T \tag{3.51}$$

Algorithm 2 TONGA: topmoumoute on-line natural gradient descent

$X = [\,], G = [\,]$
$r \leftarrow 1$
repeat
  get next mini-batch t
  $r \leftarrow r + \gamma^{t-1}$
  $\hat{\boldsymbol{g}} \leftarrow$ current mini-batch gradient
  if $t = 0$ **then**
    $\boldsymbol{\mu} \leftarrow \hat{\boldsymbol{g}}$
  else
    $\boldsymbol{\mu} \leftarrow \gamma\boldsymbol{\mu} + \hat{\boldsymbol{g}}$
  end if
  if columns$(x) > k + b$ **then**
    $VD^2V^T \leftarrow \text{SVD}(G)$
    $U \leftarrow XVD{-}1$
    $C^{-1} \leftarrow U^T D^{-2} U$
    $d = (\beta C^{-1} + I)\hat{\boldsymbol{g}}$
    $\hat{V} \leftarrow$ first k eigenvectors of V
    $\hat{U} \leftarrow X\hat{V}D^{-1}$
    $X \leftarrow \left[\hat{U}\right]$
    $G \leftarrow X^T X$
  else
    $X = \left[\gamma X \qquad \hat{\boldsymbol{g}} - \frac{1}{r}\boldsymbol{\mu}\right]$
    $G \leftarrow X^T X$ {(an iterative update rule exists for centred formulation)}
    $y \leftarrow [0, \ldots, 0, 1]^T$
    $d \leftarrow (\beta G + I)^{-1}(y - \beta X^T \frac{1}{r}\boldsymbol{\mu}) + \frac{1}{r}\boldsymbol{\mu}$
  end if
  $\theta \leftarrow \theta - \alpha d$ {gradient descent rule with step size α}
until stop criterion

and the TONGA calculation $\boldsymbol{g}^*$ (3.40) becomes:

$$\boldsymbol{g}_t^* = X_t(\beta X_t^T X_t + I)^{-1}\boldsymbol{y}_t \tag{3.52}$$

Le Roux et al. [2008] vaguely claims that using the uncentred covariance is numerically more stable, though they use the centred covariance in their follow-up paper [Le Roux et al., To Appear 2010], so it's not really clear what they recommend one should use. I will therefore compare results using both the centred and the uncentred covariance. It should be noted that the Bayesian justification given in 3.2.1 only applies for the centred covariance. We may also observe that at an optimum, when the average gradient is zero, the centred and uncentered covariance matrices become equal.

Chapter 4

Natural Gradient Learning in RBMs

4.1 Methods

4.1.1 DEEP

I used an existing implementation of Restricted Boltzmann Machines called "Deep", originally implemented by Athina Spiliopoulou[1] and Peter Orchard[2] at the University of Edinburgh. The implementation is done in Matlab. My main contributions include the implementation of the different natural gradient methods described in the chapter 3. I also added an implementation of the Persistent Contrastive Divergence training method described in section 2.3.3. Furthermore, I added code to calculate the true gradient to allow training small RBMs by the true gradient or to compare the directions given by other training methods to the true gradient and back-ported the code to run on Matlab version 2007b. I adapted code originally due to Ruslan Salakhutdinov[3] and Iain Murray[4] to either exactly calculate the partition function (for small RBMs) or to give an approximation based on the annealed importance sampling method described in [Salakhutdinov, 2008; Salakhutdinov and Murray, 2008] (for larger RBMs). This allows evaluating the exact or approximate data log-likelihood.

4.1.2 Evaluation

To compare different training regimes, we need a measure of performance. The training methods I compare are all based on CD-1 training. CD-1 is justified by being a rough approximation of the training data log-likelihood gradient and the intention of and justification for following the CD-1 direction is to increase the average training data log-likelihood. It therefore seems reasonable to simply use the training data log-likelihood (*i.e.* the *empirical* cost) as the measure of performance. This lets us focus on the optimisation performance, purely based on how well the underlying objective function is optimised. However, it should be noted that decreasing the data likelihood is not necessarily the main objective when training models for machine learning. Rather, we are usually more interested in generalisation properties (*i.e.* optimising the *expected* cost and avoiding

[1] University of Edinburgh, a.spil@ed.ac.uk
[2] University of Edinburgh, P.R.Orchard@sms.ed.ac.uk
[3] Massachusetts Institute of Technology, rsalakhu@mit.edu
[4] University of Edinburgh, i.murray@ed.ac.uk

over-fitting on the training data). Still, the training data log-likelihood gives a good indication how well one training regime performs compared to another. Also, RBMs are commonly used for large-scale unsupervised learning, where data is abundant, a scenario where we have to worry much less about over-fitting issues ([Bottou et al., 2008]). Using estimation of the data log-likelihood is also one of the recommended measures of performance in Hinton [2010]. Using the training data log-likelihood gives us a very simple measure: We simply compare the reached likelihood by CPU time spent.

For small models, I simply calculate the exact data log-likelihood using formulas 2.10 and 2.11. This is infeasible for normal sized models due to fact that computation time for the partition function 2.11 increases exponentially with the number of nodes in the smaller layer, so I instead calculate an approximation of the data log-likelihood, using annealed importance sampling, using an implementation of AIS available on-line by the authors of [Salakhutdinov and Murray, 2008].[5]

4.1.3 Data set

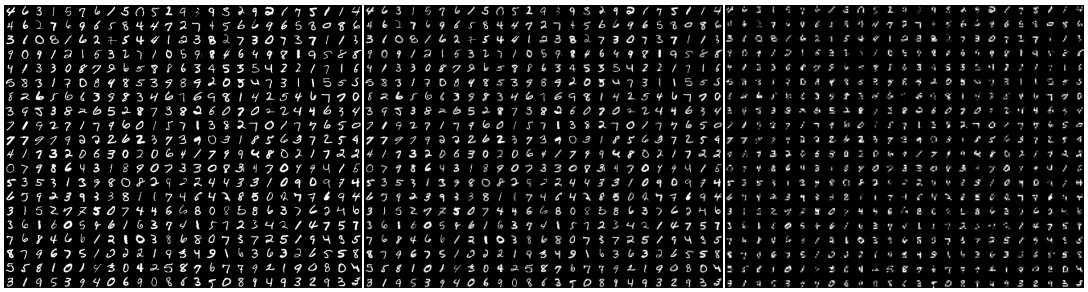
4.1.3.1 Full data set – MNIST

The MNIST data set ([Lecun and Cortes]) is a standard data set that contains 70,000 28×28 grey-scale images of handwritten digits, split in a training set of 60k and a test set of 10k. For many experiments I only use a 10,000 example sub-set of the full data set. This improves the performance of the estimation of the average data log-likelihood, but is still a large enough data set to give reliable comparison between different training methods. The images are in 256 level grey-scale. When using the images as input, I sample from a Bernoulli distribution using the grey-scale level as the mean. Considering that binary inputs work most naturally with the binary units of a RBM, I thresholded the images to give black and white images for some experiments. The original NIST data set, from which MNIST is derived, consists of only black and white images itself – the grey scale tones in MNIST are an artifact of the re-sampling and anti-aliasing done in the pre-processing for the MNIST set. Therefore, this thresholding only changes the data very slightly. There seems to be no significant difference in performance and unless otherwise mentioned, the experiments described all use grey-scale images.

4.1.3.2 Reduced data set – re-sampled MNIST

Evaluating the performance of different training regimes on RBMs is difficult. Calculating neither the true data likelihood gradient over the full data-set (to compare directions) nor the current data log-likelihood (*i.e.* the objective function) is tractable for normal sized RBMs. But, on very small RBMs. both can be evaluated in a reasonable amount of time. After running into problems evaluating the performance on full sized RBMs, I decided to run experiments on a data set of much lower dimensionality. I generated a low dimensional data set by re-sampling (bi-cubic, no anti-aliasing) the MNIST data set to 14×14 as well as 7×7 pixels. Figure 4.1 shows samples from both the original and the re-sampled data set. This low dimensional data set allows to quickly run through a wide range of meta-parameters and to calculate the exact data log-likelihood as an exact measure of performance.

[5] `http://web.mit.edu/~rsalakhu/www/rbm_ais.html`



(a) Original mnist **(b)** Scaled to 14×14 **(c)** Scaled to 7×7

Figure 4.1: MNIST data set

4.2 Initial experiments

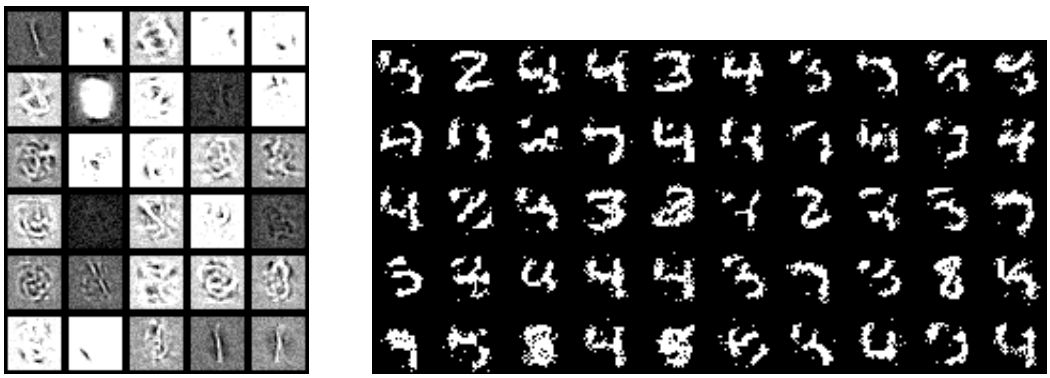
4.2.1 Setup

Initially, I tried to directly compare the performance of Tonga and CD-1 on the full MNIST set. I used the thresholded MNIST set (*i.e.* black and white only) for the initial tests, but there is little difference in performance for both the original and the dichromatic set. I generally used RBMs with 784 visible units (as prescribed by the 28×28 size of MNIST) and 500 hidden units (a very common number used in the literature when training RBMs on MNIST). For CD-1, I ran a large series of training runs with different values for the meta-parameters step-size α, L1/L2 weight-decay κ and momentum χ. I also ran tests with CD-n with n up to 20. I used AIS to get an estimate of the partition function and thus the training data log-likelihood. I also visually compared the weight matrix of trained RBMs to get an idea what kind of features (if any) individual hidden nodes would learn. Generating samples from a trained RBM gave another indication of the quality of training.

4.2.2 Results

Figure 4.2 shows an exemplar result from the CD-n training runs. Things look fine: The weights between nodes seem to encode recognisable features and generated samples have some resemblance to the training data. Figure 4.3 shows the estimated data log-likelihood and also Z. The data log-likelihood nicely increases during the test. Things appear to work.

Figure 4.4 shows the example of a training run with Tonga-CD-1. The weight matrix has many values above 50 and, visually, no feature detecting structure can be seen. Generated samples from the trained model have some similarity to the training set in the sense that the samples are mostly black with some white strokes, but the model does not generate anything resembling the training set. Now, this in itself may not be a major problem, as long as our chosen measure of performance, the training data log-likelihood, is increased by training. Figure 4.5 gives the results as estimated by AIS. Now this *is* a major problem: the estimated log-likelihood increases above 0 and then becomes `inf` at epoch 7! Clearly, the likelihood estimator cannot be trusted. I observed this behaviour not just when training with Tonga but also with standard CD-1, especially when using high learning rates, though it occurred more often when using



(a) Excerpt from weight matrix

(b) Generated samples

Figure 4.2: Example. **Left:** Excerpt of the weight matrix of an RBM trained by CD-1 on MNIST, blockwise arranged by connections of a single hidden node to all visible nodes. We can see that some nodes encode simple strokes. Some remind of Gabor filters. One node seems to encode "image is mostly black at edges, with some white near the centre". **Right:** Generated samples from the model. Each sample was generated by starting with a random visible state and running 1000 steps of Gibbs sampling. This RBM produces fairly nice samples and we can clearly recognise some numbers. This particular RBM was actually trained with CD-16. CD-1 typically gave less clear samples

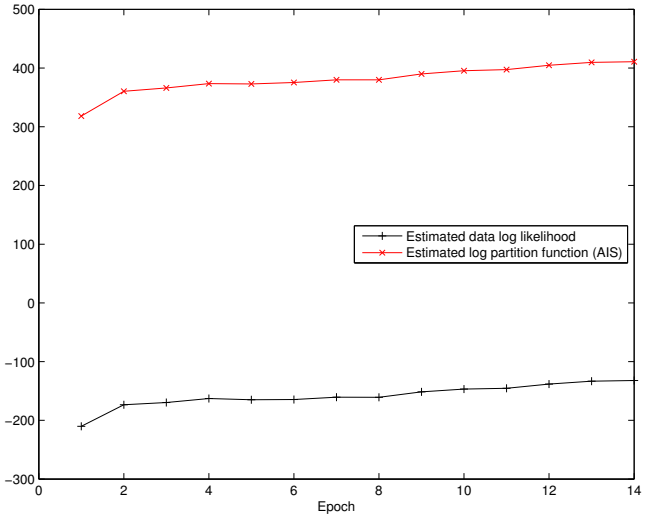
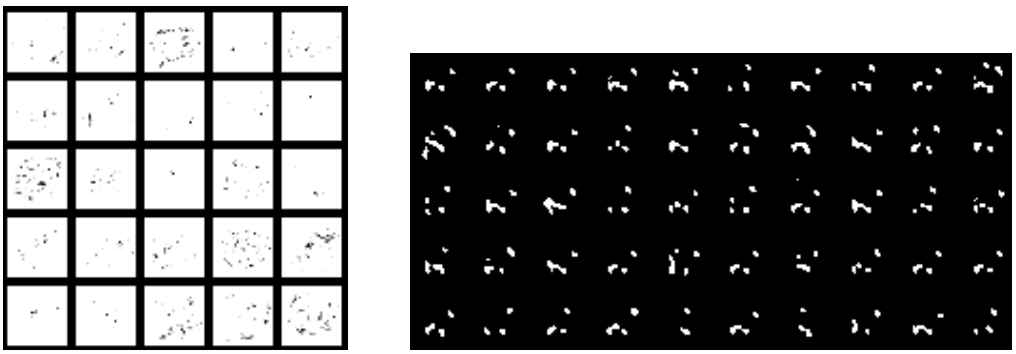


Figure 4.3: Example: Data log-likelihood during training by CD-1. Both data log-likelihood and partition function Z increase during training and things seem to work

Tonga.[6] Further experiments indicated that the estimation of the partition function by AIS becomes quite in-exact when the weight parameters become large, and unfortunately not just in an random, unbiased way, but strongly biased to *over*-estimate the data log-likelihood. This is especially bad, because we now cannot reliably distinguish between a training method that works really well and a training methods that simply leads to extreme weight and bias parameters: both

[6]This may likely have been caused by a fault in the initial implementation, though I will show later that natural CD-1 and Tonga can indeed tend towards very bad configurations with high weights when choosing wrong meta-parameters



(a) Excerpt from weight matrix

(b) Generated samples

Figure 4.4: Example: Excerpt of a weight matrix and generated samples, trained by Tonga-CD-1. The weight matrix has many very high values and shows little evidence for learnt features. Generated samples do not resemble the training data well, but are somewhat close (*i.e.* mostly black with some white strokes and blotches)

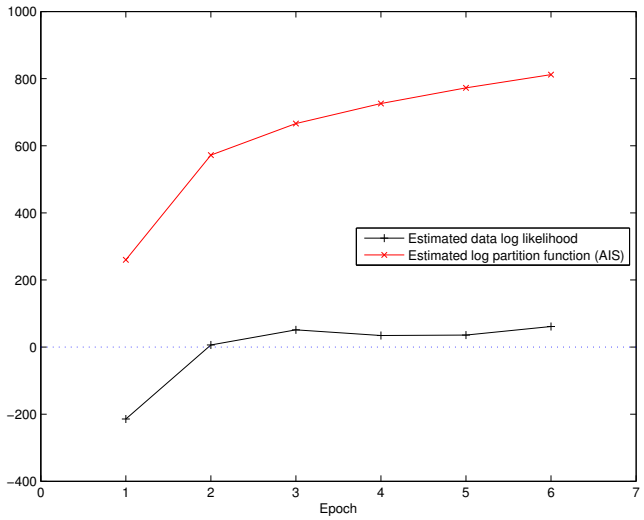


Figure 4.5: Example: Data log-likelihood during training by Tonga-CD-1. Both data log-likelihood and partition function Z increase during training. But: Estimated data log-likelihood increases above 0 and actually becomes inf at epoch 7. Clearly this result is invalid

will have a higher data log-likelihood, according to the estimator. This problem could be mitigated somewhat by using a training regime that discourages extreme weight parameters. A simple way is to include a L2 weight-decay with a relatively large factor κ. Another method I tried was to simply restrict parameters to be in a range of $(-20, 20)$. However, both methods seemed to be an unjustified hack.[7] More importantly, there was also no good way to find out exactly how (in-)accurate the AIS estimation is and what numeric ranges for the weight and bias parameters should be avoided for accurate results. After all, I could only know for certain that the estimate was wrong, when the AIS estimator reported entirely unreasonably high (or even positive)

[7] I have since learnt that experts actually recommend using weight-decay, and a higher than usual κ when using AIS [Hinton, 2010])

data log-likelihoods. This left me with no reliable method to actually compare the performance of different training regimes. Therefore, I decided to downscale and work with much smaller RBMs an much smaller data sets, where the data log-likelihood can be evaluated exactly.

4.3 Small scale experiments – natural CD-1

4.3.1 Experimental setup

Unless otherwise mentioned, the following experiments all use an RBM with 49 visible and 20 hidden nodes. Test are done on the re-sampled MNIST data set shown in figure 4.1c. There are three main advantages to using this small RBM and the small data set:

1. Training is much faster, therefore the meta-parameter space (α, γ, β etc.) can be explored exhaustively.

2. When using only 20 hidden nodes, the partition function Z and thus the training data log-likelihood can still be calculated exactly in a reasonable amount (< 10 seconds) of time. Therefore, results can be compared with confidence. 20 is the upper limit where calculating Z repeatedly is still fast enough. For 25 hidden units, the calculation already takes hours.

3. The true data log-likelihood gradient can be calculated exactly in a very short amount of time. This allows comparing the training direction to the true gradient.

In the following sections I use the naïve implementation of stochastic natural gradient descent as described in algorithm 1. This simple implementation allows observing the effect of using the natural gradient for learning in isolation and avoids having to also worry about the effect of the approximations used in Tonga and the additional meta-parameters of Tonga. I will call the training regime that uses this simple natural gradient implementation over the directions obtained from CD-1 "natural CD-1" in the following sections.

I do not give results compared by CPU time. Instead I compare data log likelihood by iteration (*i.e.* number of mini-batches/epochs). In most machine learning applications, the former will be the most relevant factor. because unsupervised machine learning problems typically have data available in abundance, but in this section I only intend to show if using the natural gradient can work at all. The simple natural gradient implementation used for the small scale test is certainly *not* competitive to CD-1 in terms of CPU time and does not scale to larger scale problems. Performance will only become relevant when switching to the Tonga implementation (that performs farily well when using block diagonal approximation) and only if Tonga at least works well in the comparison data log likelihood/iteration. In some of the experiments, I compare results obtained from a single run only instead of reporting the results over a number of runs. Since training is stochastic, results can vary in-between runs even when using the same meta-parameters. In practise, I've found that runs on the 7×7 data set actually only show little variance between different runs, so I argue even results from only a single run are significant enough to select the best meta-parameters, but this is certainly not ideal. Whenever available time allowed it, experiments have been done on averages over 5-10 runs.

The goal of the first set of experiments was to see if using the natural gradient can improve learning at all and also to explore the meta-parameter space for the parameters *discount factor* (γ) and *regularisation factor* (β) described in section 3.3. The results are compared to CD-1 learning with the same step size.

For these experiments I used a sub-set of re-sampled MNIST with $N = 10000$, using a mini batch size of 60. I chose this batch size as a compromise between very small mini-batches, where the naïve implementation is too slow, and larger mini-batches, where the on-line learning aspect would be lost. I used only a subset of 10k data points to keep calculating the exact likelihood very quick. Also, the re-sampled data set contains little variation, and preliminary experiments using CD-1 have actually shown little difference between using a subset of only 1k and the full 60k set. The mini-batches consist of 60 consecutive data points from the shuffled data set. That is, the same training point is only re-visited after one full iteration over the entire data set ("one epoch"). An alternative method would have been to choose the 60 training points in an epoch by random sampling, but there should be little difference in both approaches. I initialise the weights using a Gaussian with standard deviation 0.01, as recommended in [Hinton, 2010].

As mentioned before, using the regularisation of the covariance matrix in formula 3.18 has the unfortunate side effect that the magnitude of the resulting natural gradient can not only differ a lot from the magnitude of the CD-1 direction but also depends on the regularisation factor λ. I use the alternative regularisation method according to 3.19, which suffers less from this problem, for the typical amount of regularisation we want to use. However, the norm of the natural gradient still depends on parameter β and this makes results obtained for different β difficult to compare. I therefore use an additional normalisation factor, that adjusts the natural gradient, so that its L2-norm becomes the same as the L2-norm of the underlying gradient would be. This normalisation factor l is simply computed as:

$$l = \begin{cases} \frac{||\hat{\boldsymbol{g}}||_2}{||\boldsymbol{g}^*||_2}, & \text{for } ||\boldsymbol{g}^*||_2 > 0 \\ 0, & \text{for } ||\boldsymbol{g}^*||_2 = 0 \end{cases} \tag{4.1}$$

Section 4.3.2.2 shows results when omitting l for comparison.

For sake of easier implementation, my algorithms train the weights and the biases separately. That is, I'm working with *three* covariance matrices, the first for training the gradient for the weights, the other two for training the biases of the hidden and visible nodes. Since the RBM used has 49 visible and 20 hidden nodes, the gradient for the weights has size $49 \times 20 = 980$ whereas the gradients for the biases have size 49 and 20. Arguably, I should at least initially not have separated the training of biases and weights, however for large scale problems we will want to use a block-diagonal approximation any way. So any useful training method has to work well, even when the gradient is split in subsections.

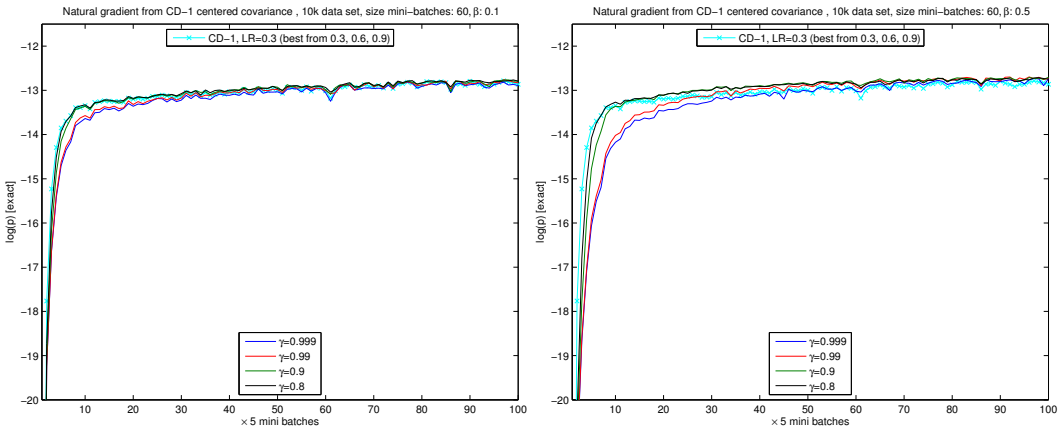
4.3.2 Comparing natural CD-1 (centred covariance) to CD-1

Preliminary tests running CD-1 with different fixed step sizes between 0.2 and 1 on this data set indicated that a fixed step size of 0.3 gives a good compromise between initial speed of learning and convergence in the limit. Using a decaying step size gives better convergence in the end, but using a fixed step size should make the result more easily comparable as it gives "slow starters" a

chance to catch up and avoids having to consider the effect of the decay schedule as yet another parameter. I also don't use momentum and weight-decay for the same reasons.

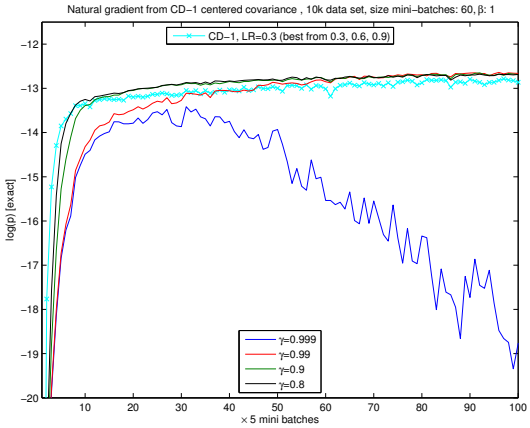
4.3.2.1 Exploring the meta-parameter space: effect of γ and β when using the centred covariance

I ran a batch of 16 experiments for different combinations of γ and β, comparing results of natural CD-1 to the results obtained for CD-1 with the same learning rate. The normalisation factor l described above ensures that the learning rates are comparable for different values of β. Figure 4.6 show the results for the different meta-parameters and figure 4.7 shows a more detailed view of the quality of the trained model near convergence.

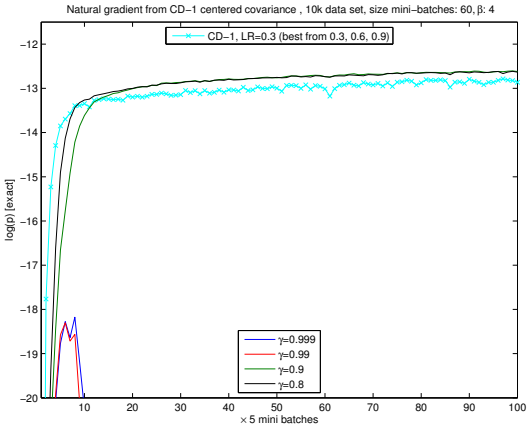


(a) $\beta = 0.1$

(b) $\beta = 0.5$



(c) $\beta = 1$

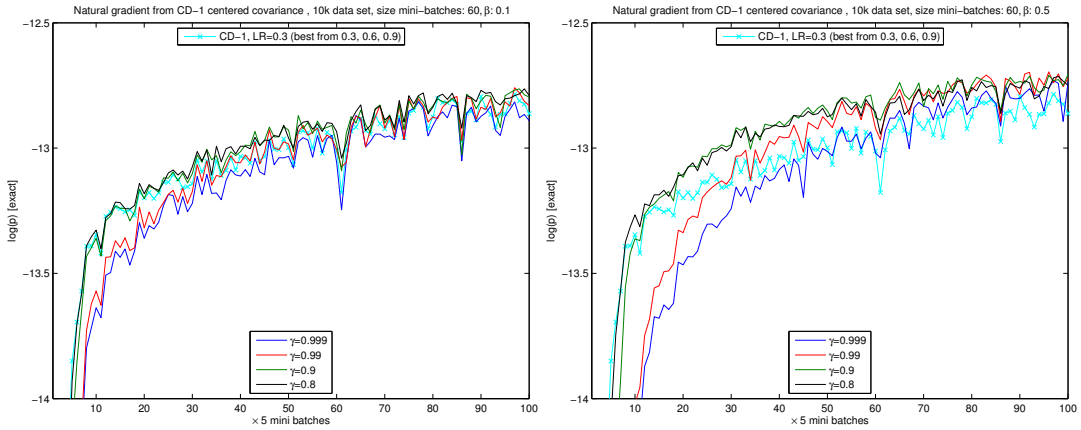


(d) $\beta = 4$

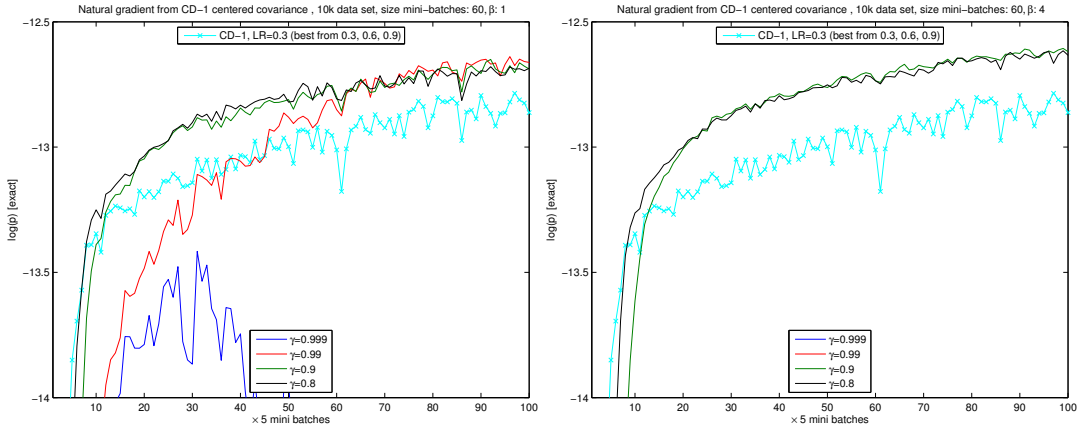
Figure 4.6: Results training with natural CD-1, centred covariance, with normalisation to norm of CD-1. CD-1 training for comparison is in cyan. Low values of β result in little difference to CD-1. High values of β can give better results than CD-1, but we risk unstable behaviour for values of γ close to 1. Learning is also slower in the initial phase for all meta-parameters

There are three observations from this experiment:

1. For low values of β, the results of natural CD-1 are very close to the results of CD-1. This



(a) $\beta = 0.1$ **(b)** $\beta = 0.5$



(c) $\beta = 1$ **(d)** $\beta = 4$

Figure 4.7: Results training with natural CD-1, centred covariance, with normalisation to norm of CD-1, close-up of the convergence region. Natural CD-1 can clearly outperform CD-1, when choosing suitable meta-parameters. High values of β give best overall results, but also the most unstable

is the expected result, as the natural gradient converges to the underlying gradient for low β. Therefore, considering the additional cost of calculating the natural gradient, using low values of β is not advisable. Should low values of β ever give the best results, we should instead simply not use the natural gradient method at all.

2. For higher values of β with lower values of γ, natural CD-1 starts slower, but clearly performs better than CD-1 in the limit. But:

3. For higher values of β and values of γ close to 1, training by natural CD-1 can become unstable: Figures 4.6c and 4.7c show how training by $\gamma = 0.99$ trails CD-1 for the first 250 mini-batches and using $\gamma = 0.999$ even decreases the objective function after reaching a peak at about mini-batch 150. Figures 4.6d and 4.8 show that this effect is more pronounced if we regularise less (*i.e.* use higher β). This effect may be caused because higher values of β give higher weight to the covariance and higher values of γ give higher weight to mean and

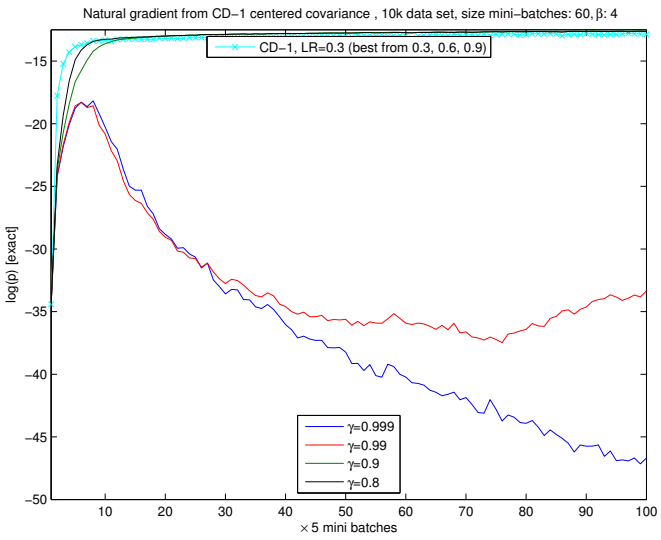


Figure 4.8: Results training with natural CD-1, centred covariance, showing the divergent behaviour that can occur for high values of γ

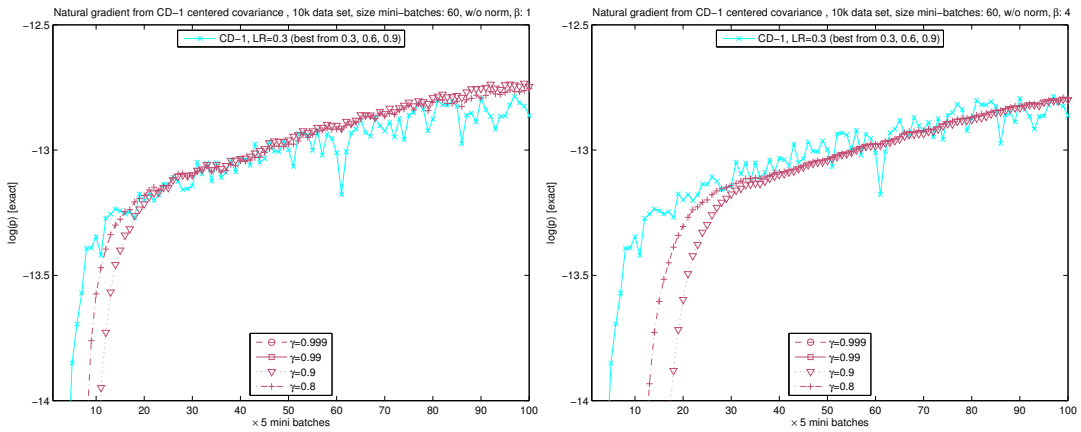
covariance from gradients encountered far away in the past. Especially at the early stages, when the gradient may change much more between batches than in later stages, including terms from gradients far in the past may not be a good thing. Possibly, we should start out with a lower value of γ, and only increase γ as the model converges more and more, but I did not attempt this. As we will see, there are other ways to help avoiding instabilities.

4.3.2.2 Effect of not normalising to the norm

Since Le Roux [2010b] reports slow learning when using normalisation factor l (4.1), I include results when omitting the normalisation term in figure 4.9. In direct comparison to the results for normalised natural CD-1 (fig. 4.7), results here are significantly worse. It seems that using the normalisation l is required for obtaining best results. It also seems that training by natural CD-1 leads to a much smoother increase of the data log-likelihood. Training by CD-1 results in many downward spikes in the data log-likelihood curve, presumably happening when mini-batches differ the most from the over-all data set. The smoothing effect is also visible in figures 4.6 and 4.7 for higher β, but is more pronounced in this experiment. I conjecture that the smoother training stems from including information from previous mini-batches in natural CD-1 which may have a similar effect to using momentum whereas CD-1 (without using momentum) gives a direction based on the current batch only.

4.3.2.3 Comparing natural CD-1 (centred covariance) to CD-1: Conclusions

First, for low values of β, natural CD-1 performs similar to CD-1. This is the expected result: low β give little influence to the covariance and for $\beta \longrightarrow 0$, the natural gradient converges to the underlying gradient. For higher β, the results are ambivalent. On one hand natural CD-1 can improve on CD-1: during the convergence phase natural CD-1 leads to a significantly higher data log-likelihood, as long as meta-parameter γ isn't set to high. On the other hand training by



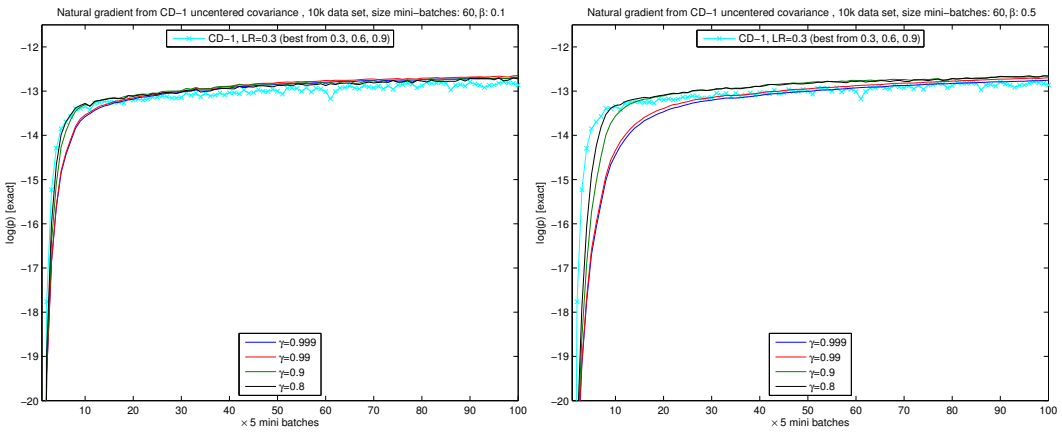
(a) $\beta = 1$ **(b)** $\beta = 4$

Figure 4.9: Results training with natural CD-1, centred covariance, without normalisation to norm of CD-1, close up of the most interesting region. Results are worse than when normalising to the norm (cmp. 4.7), but still competitive to CD-1. Also, the data likelihood increases more smoothly between mini-batches. Training results for $\gamma = 0.999$ and $\gamma = 0.99$ are similar to 4.8 and off the chart. Results for lower β are again comparable to CD-1 and are not shown

natural CD-1 can lead to very bad results when choosing high settings for parameter γ. Unless we find ways to avoid unstable results, using the natural gradient will not be a viable training method, especially on bigger models, when it will be more difficult to even detect when bad training occurs. The bad results for high values of γ is a little surprising, as Le Roux et al. [2008] actually recommends setting γ to 0.995. One reason for this may be that the CD-1 direction is especially noisy during the first few mini-batch iterations. But the main cause will turn out to be the fact we're using the centred covariance. According to [Le Roux et al., 2008], using the uncentred covariance may lead to more stable results. This is investigated in the next section. As an aside, we've also seen that scaling the natural CD-1 gradient to have the same norm as the CD-1 direction is justified by results.

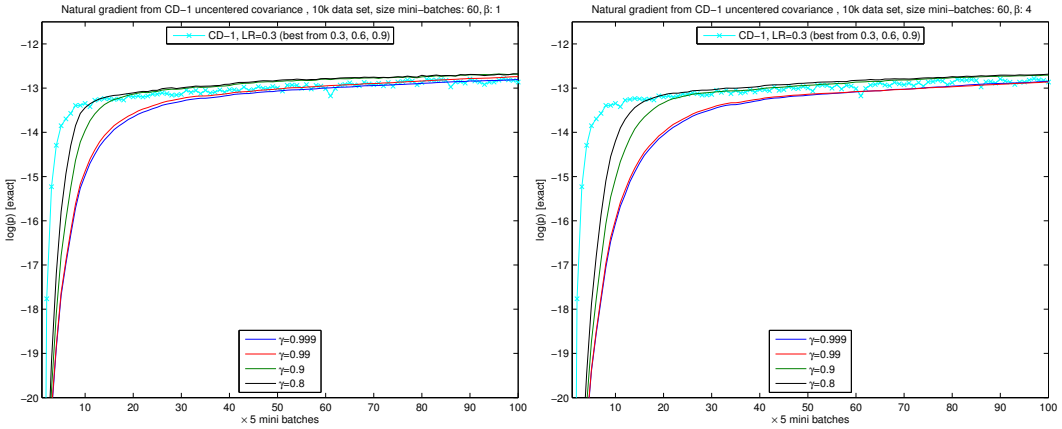
4.3.3 Using the uncentred covariance

I repeated the same batch of tests using the uncentred implementation of natural CD-1 (*i.e.* using (3.51)). The results are shown in figures 4.10 and 4.11. Most importantly, we can see that using the uncentred covariance avoids the unstable results we've seen when using the centred covariance and results are comparable to CD-1 for all tried values for γ. Training is still slow for high values for γ, when at the same time also using high values of β, but nowhere as bad as when using the centred covariance. If we directly compare 4.11 to 4.7, we can also see that using the uncentred implementation leads to a much smoother increase in training data log-likelihood, even for very low values of β. We also see that using natural CD-1 with the centred covariance can lead to better final results than with the uncentred covariance. But it is questionable if we should risk unstable behaviour in order to possibly find a better optimum, especially considering how difficult it will to detect unstable training in large models.



(a) $\beta = 0.1$

(b) $\beta = 0.5$



(c) $\beta = 1$

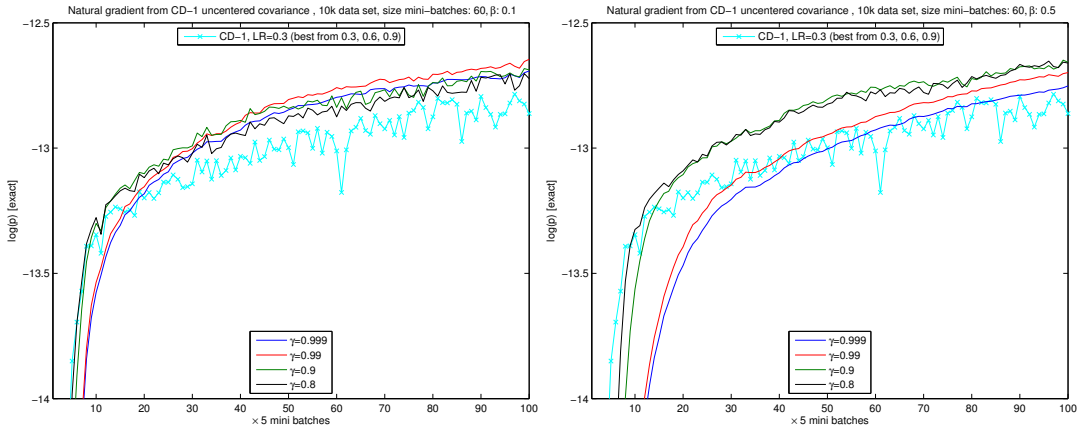
(d) $\beta = 4$

Figure 4.10: Results training with natural CD-1, uncentred covariance, with normalisation to norm of CD-1. Low values of β result in little difference to CD-1. High values of β can give slightly better results than CD-1 and we don't experience the same instability for higher values of γ as we do when using the centred covariance. Slow starting

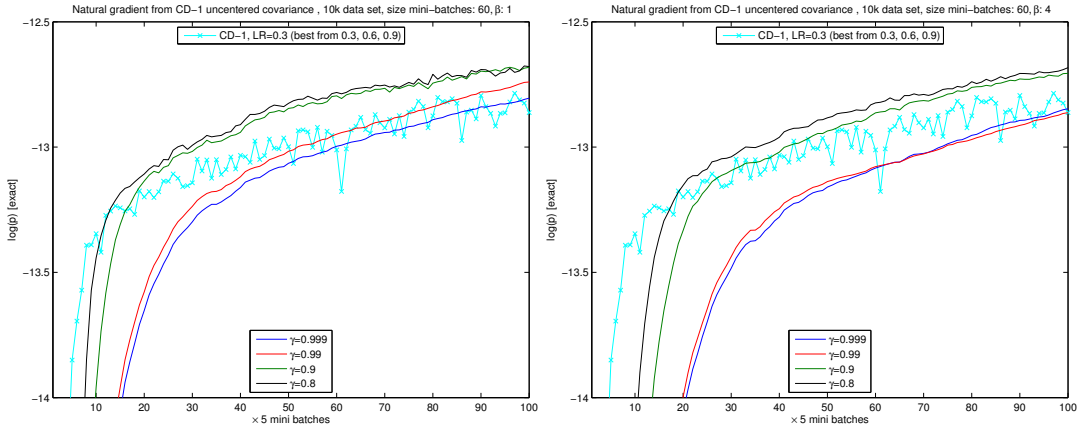
It is perhaps not that surprising that the centred formulation can give unstable results: Especially during the initial stage, the true gradient (and therefore also the CD-1 approximation) changes rapidly. For high γ, the moving average of the mean (3.34) gives high weight to gradients far in the past. This makes the moving average a very bad approximation of the current mean and this should greatly and erroneously increase the calculated covariance.

4.3.4 Using persistent contrastive divergence (PCD)

Considering that PCD (also known as Stochastic Maximum Likelihood – SML) gives a better direction in terms of going down the data likelihood gradient [Bengio, 2010], it seems reasonable to try if we get better results for the natural learning, if we use PCD as the underlying gradient. Figure 4.12 shows some results comparing PCD-1 and natural PCD-1. I used 60 independent Markov chains for this experiment. Using the same number of Markov chains as the size of



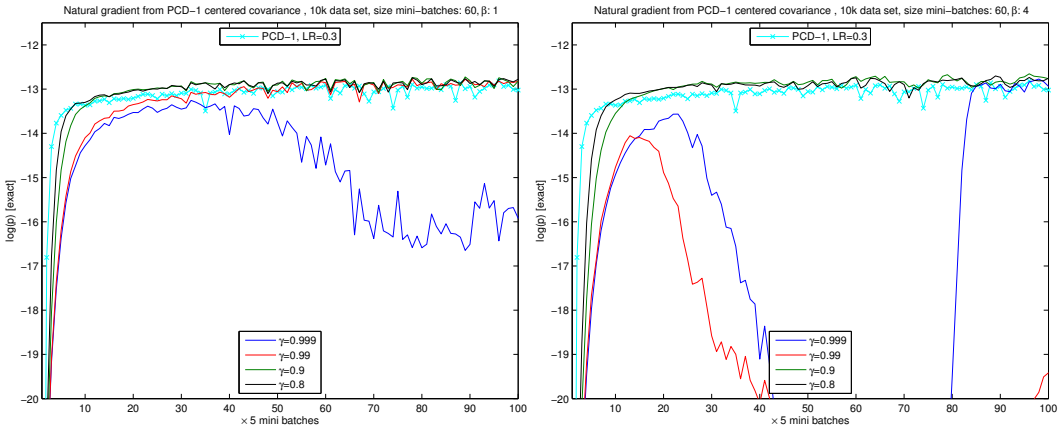
(a) $\beta = 0.1$ **(b)** $\beta = 0.5$



(c) $\beta = 1$ **(d)** $\beta = 4$

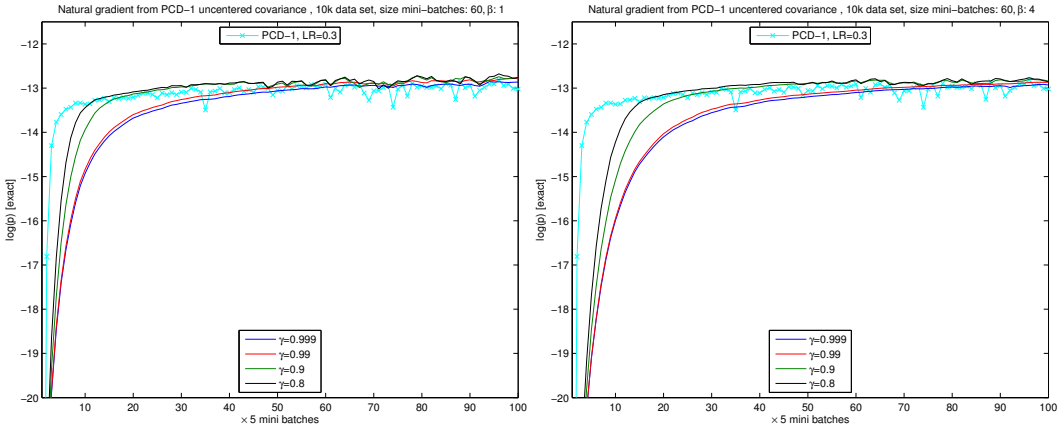
Figure 4.11: Results training with natural CD-1, uncentred covariance, with normalisation to norm of CD-1, close-up of the most interesting region. Natural CD-1 can again slightly outperform CD-1, when choosing suitable hyper-parameters. Increase in data log-likelihood is much smoother compared to CD-1

the mini-batches is customary, since it simplifies the implementation, but there is no theoretical justification. 60 is well above the recommended minimum number of chains. [Tieleman, 2008] uses 100. Natural PCD-1 is competitive to PCD-1, but unlike in the comparison of natural CD-1 and CD-1, using the natural gradient does not appear to give a significant advantage for any of the tried meta-parameter. Again, for larger values of γ instability can occur and using the uncentred covariance helps to avoid this problem. To keep results comparable, I used the same step size of 0.3 for this experiment. Generally we should use a lower learning rate when using PCD-1 compared to CD-1 ([Hinton, 2010]) and it is therefore possible that I could get better results by lowering the step size, but I did not try this. This may also explain why PCD-1 converges to a slightly lower data likelihood than CD-1. But since PCD-1 does not significantly improve on PCD-1, I did not follow-up on this experiment further.



(a) $\beta = 1$, centred covariance

(b) $\beta = 4$, centred covariance



(c) $\beta = 1$, uncentred covariance

(d) $\beta = 4$, uncentred covariance

Figure 4.12: Results training with natural PCD-1, centred and uncentred covariance, with normalisation to norm of PCD-1. Natural PCD-1 is competitive to PCD-1, but gives no additional advantage. Again, using the uncentered covariance seems to help with unstable results for high values of γ

4.4 Towards Tonga

4.4.1 Using block diagonal approximation

4.4.1.1 Why should we use a block diagonal approximation?

Le Roux et al. [2008] recommend approximating the full covariance matrix by only retaining elements block-wise along the diagonal. This approximation is obtained by splitting the weight gradient into segments that each correspond the connections of a single hidden unit to all visible unit, then calculate covariances only between elements within each segment.

Let us consider the reasons why using the block diagonal approximation may give better results:

- If we use a block diagonal approximation, the covariance matrices are of much smaller dimensionality. This reduces computational cost, especially for the direct matrix inversion.

- Le Roux et al. [2008] report that on experiments using artificial neural networks, the block diagonal approximation of the weight covariance matrix gives a very good approximation of the full covariance, so we don't actually loose much information when approximating the full covariance matrix by blocks along the diagonal.
- The smaller diagonal block matrices can be low rank approximated much more accurately when retaining only the primary k eigenvectors of the matrices. This is the main motivation for using a block diagonal approximation in TONGA [Le Roux et al., 2008]
- Arguably, the weights of the connections of a single hidden unit are more related to each other than to weights "further away". Thus, the covariance between the more closely related components of a gradient is more relevant than the covariance between components further apart. Dropping the latter may then improve the overall performance of the natural gradient method.

I will first investigate if the second point holds for RBMs, then give comparative results showing if the method helps or hinders learning.

4.4.1.2 Approximation quality of the block diagonal approximation

To calculate how good the full covariance matrix is approximated by retaining only the blocks along the diagonal, I calculate the ratio of the Frobenius norms: $\frac{||C-\bar{C}||_F}{||C||_F}$ where C is the full covariance matrix and $\bar{C}$ the block-diagonal approximation. If the covariance matrix is perfectly approximated by the block diagonals, this ratio is 1. The closer the ratio is to 0, the worse is the approximation. [Le Roux et al., 2008] reports a ratio as low as 0.35 when training a $16-50-26$ ANN, even though $\bar{C}$ only includes 1.74% of the elements if C. I cannot reproduce this results on RBMs: When training the 49×20 RBM on the small 7×7 problem, I've found that a block diagonal approximation of the covariance matrix is a much less accurate approximation and the ratio is mostly above 0.85, even though I'm retaining 5% of the total elements. Figure 4.13 visualises a typical covariance matrix encountered during training with natural CD-1. We can see that the matrix is approximately sparse with most values close to 0, and does not show much of a block-diagonal structure. Therefore, the justification for using a block-diagonal approximation given in [Le Roux et al., 2008] for ANNs does not hold: the block-diagonal approximation does not seem to give a very close approximation for RBMs.

4.4.1.3 Using the block-diagonal approximation: results

Considering how badly the full matrix is approximated by only retaining the elements of blocks along the diagonal, I did not expect good results from using this approximation. However, I actually obtained pretty good results in my experiments. Figure 4.14 compares the results of natural CD-1 and natural CD-1 with a block diagonal approximation to CD-1. I used the best over-all observed meta parameters (so far) for training natural CD-1: $\gamma = 0.8$, $\beta = 4$, using normalisation factor l and using the centred covariance. We can see that natural CD-1 with the block-diagonal approximation starts slower, but results in a better trained model during the later stages of training.

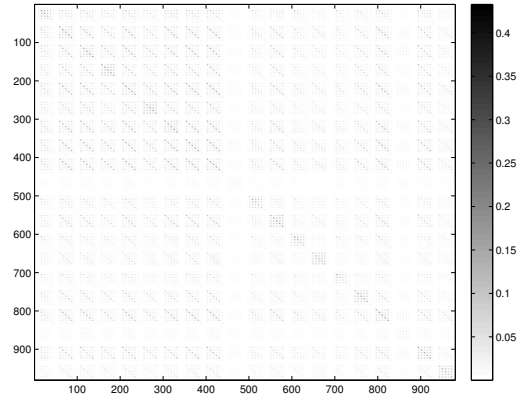


Figure 4.13: Example of a covariance matrix during natural CD-1 training. This particular covariance matrix shows the state after 100 mini-batches, training with $\alpha = 0.3$, $\beta = 4$ and $\gamma = 0.8$. Elements in the matrix are arranged, so that the covariance terms for weights connecting a single hidden unit to all visible units are in 49×49 blocks along the diagonal. While there is a discernibly over-all block-wise pattern, this covariance matrix is not well approximated by only retaining the elements of blocks along the diagonal. The matrix also has most elements very close to 0

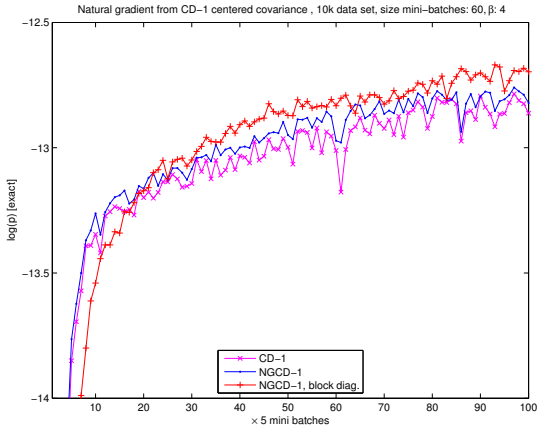
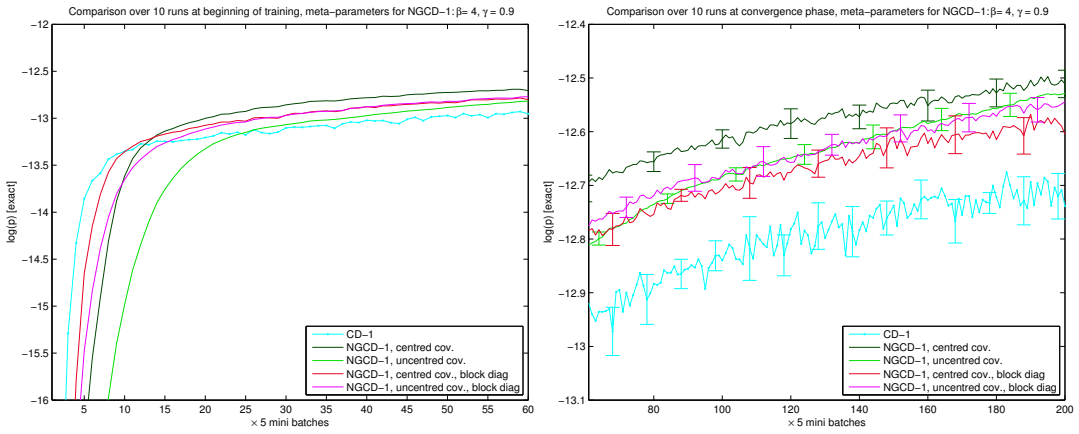


Figure 4.14: Comparison of training with CD-1, natural CD-1 and natural CD-1 using the block diagonal approximation. Using natural CD-1 with block-diagonal approximation leads to slower learning in the beginning, but then outperforms both CD-1 and natural CD-1 without block-diagonal approximation

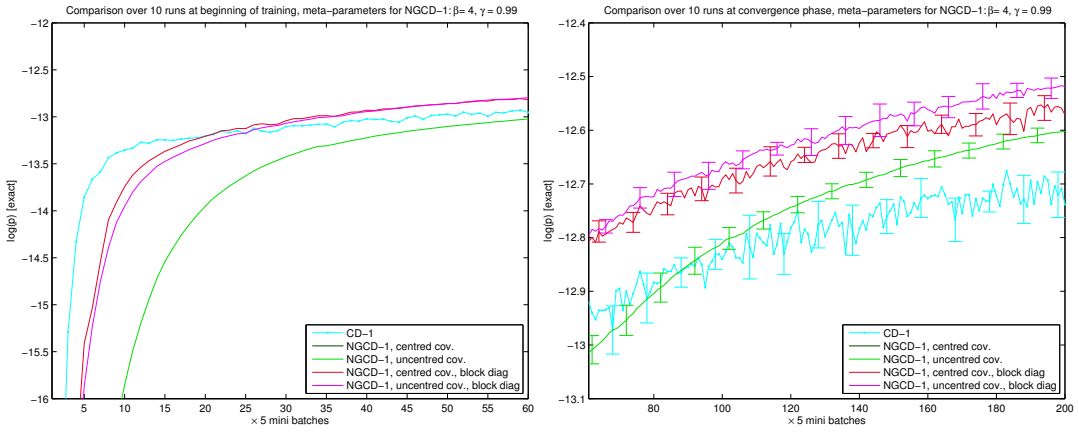
To confirm these finding I ran a more extended set of comparisons using meta-parameter $\beta = 4$ with 10 runs each for CD-1 and natural CD-1 with and without block diagonal approximation. In contrast to the previous tests, that only used a 10k subset, I ran these test over the full 60k MNIST training set, still re-sampled to 7×7. Figure 4.15 shows the results for $\gamma = 0.9$ and figure 4.16 for $\gamma = 0.99$. Again, using a high γ with centred covariance leads to unstable results when used without the block-diagonal approximation (figure 4.17), though it looks as if the training would catch up if only run for long enough. For all other tried meta-parameters, the natural gradient training methods again causes a slower training in the first few dozen mini-batches but significantly outperform CD-1 in the later stages, when closer to convergence. It seems that



(a) Start of training

(b) Convergence phase. Errorbar denote 1 σ over 10 runs

Figure 4.15: Results natural CD-1 vs CD-1 with and without block-diagonal approximation for $\gamma = 0.9$, averaged over 10 runs, but using identical mini-batches between runs. Difference between runs is in random initialisation and sampling only. Error bar only shown every 10 data points to avoid cluttering. All tried natural methods start slow, but find a higher data log-likelihood in the end. Using the centred covariance gives the best over-all result. For the uncentred covariance, using BD-approximation makes surprisingly little difference



(a) Start of training

(b) Convergence phase. Errorbar denote 1 σ^2 over 10 runs

Figure 4.16: Results natural CD-1 vs CD-1 with and without block-diagonal approximation for $\gamma = 0.99$, averaged over 10 runs, but using identical mini-batches between runs. Difference between runs is in random initialisation and sampling only. Error bar only shown every 10 data points to avoid cluttering. Results for using the centred covariance without block-diagonal diverge and are off-chart. They are shown in figure 4.17 instead. We can see that for high γ, using BD-approximation can help avoid instabilities when using the centred covariance and improves performance for the uncentred covariance also

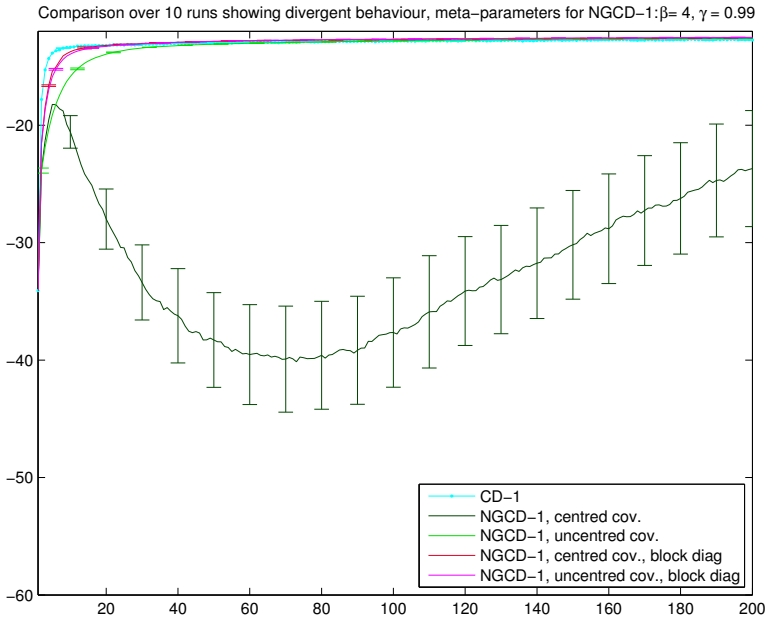


Figure 4.17: View of the divergent behaviour for $\gamma = 0.99$, centred covariance in figure 4.16

natural CD-1 performs best during the convergence phase, where the true gradient does not change much anymore, but slows down learning compared to CD-1 in the beginning.

Unfortunately, the experiments do not give a clear answer to the question "does using a block-diagonal approximation lead to a better trained model". For $\gamma = 0.99$ the answer is a clear yes: Using the block-diagonal approximation results in a higher data likelihood whereas not using it leads to slow learning or even instability. For $\gamma = 0.9$, however *not* using the block-diagonal approximation gives slightly better results, though all results are fairly close to one another for this meta-parameter setting. Overall, given these results, we have to recommend using the BD-approximation: in scenarios where instability is not an issue, using the BD-approximation does at least not hurt much and in some scenarios it can help avoid instabilities, then making the difference between a training method that does not work at all and a training method that is comparable to CD-1.

But this is only half of the picture: So far, I've only compared learning per number of mini-batches. In reality, what we're really interested in is learning per CPU time spent. Therefore we also have to consider the great improvement in computational performance gained by reducing the size of the covariance matrix inversions. If we assume for simplicity that the computational cost for a matrix inversion is $O(p^3)$, where p is the number of elements in a gradient, and that the cost of storing the matrices is $O(p^2)$, replacing the single 980×980 matrix over the weight parameters in our small model with 20 49×49 sub-matrices, lowers the cost of the matrix inversions by a factor of about 20 in space and 400,000 in time. Clearly, as long as the block-diagonal model can still reach a similarly well trained state as the full model, there will be an p above which the block-diagonal approximation performs better than the full model. TONGA's complexity on the other hand only depends linearly on p, so this argument does not hold. For TONGA, the main reason why we may want to use the block-diagonal approximation is that smaller matrices are low-rank approximated more exactly when retaining a fixed number k of eigenvectors/values.

4.4.2 Conclusions from the experiments with natural CD-1

In the sections above, I presented small scale experiments using a simple implementation of natural gradient learning. The goal was to explore if natural gradient learning can improve on CD-1 training at all (initial experiments indicated otherwise) and if so, what the best meta-parameters are. I draw the following conclusions from these experiments:

- natural CD-1 training is very sensitive to meta-parameters when using the centred covariance. With careful selection of β and γ, natural CD-1 can indeed improve convergence on the data likelihood. However, if we choose high values β and at the same time values for γ close to 1, instability can occur. This may explain the bad results in the initial tests on the full MNIST set.
- Using the uncentered covariance instead of the centred covariance can help avoiding instabilities. This confirms the claim in [Le Roux et al., 2008].
- The covariance matrices used in natural CD-1 training are not well approximated by a block-diagonal approximation. This differs from the results given in [Le Roux et al., 2008] for artificial neural networks. It may be the case that this is a general property of RBMs, but we also can't rule out that the result is particular to the used data-set. A possible reason may be, that the re-sampled MNIST data set is relatively uniform (mostly black with some white strokes). Further experiments would have to be made for a definite conclusion. Either way, the fact that the block-diagonal approximation is a fairly inexact representation does not prevent good results when using it and it seems to also help in avoiding instabilities.
- None of the tried natural training regimes gives good performance in the beginning of training. As it stands, there is no reason to recommend doing the entire training session using natural methods. So far, the natural methods have only shown advantages during the convergence phase.

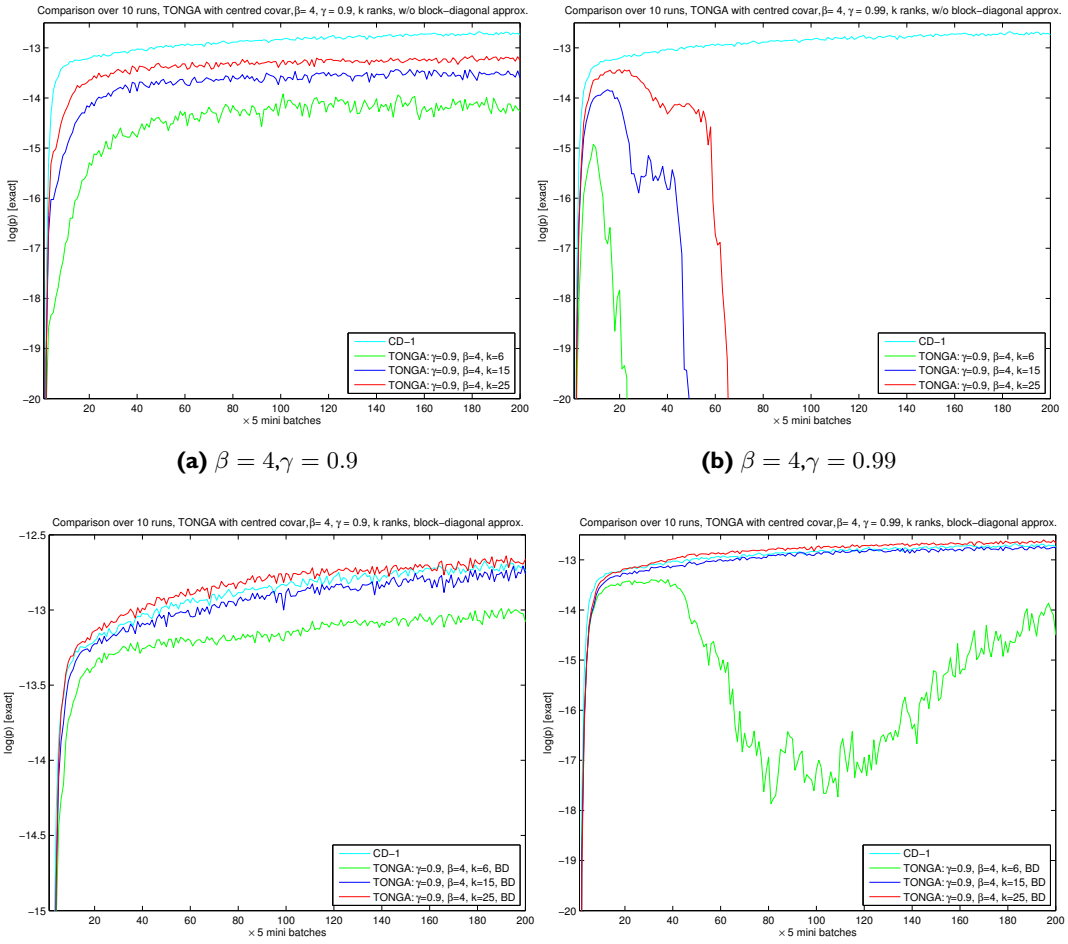
This concludes the experiments using natural CD-1. In the next sections I will present results for Tonga-CD-1.

4.5 Tonga

4.5.1 Experiments on re-sampled MNIST

4.5.1.1 Centred Tonga

The main difference between natural CD-1 and Tonga is that the latter uses a low-rank approximation for the covariance matrix. As described in Chapter 3, this makes using the natural direction tractable for larger models, where doing full matrix inversions would be infeasible. Initial experiments were done on the same re-sampled MNIST set, to allow for a direct comparison between the results for natural CD-1 and Tonga. Any differences should mainly be due to the low-rank approximation.



(a) $\beta = 4, \gamma = 0.9$

(b) $\beta = 4, \gamma = 0.99$

(c) $\beta = 4, \gamma = 0.9$, block-diagonal approximation

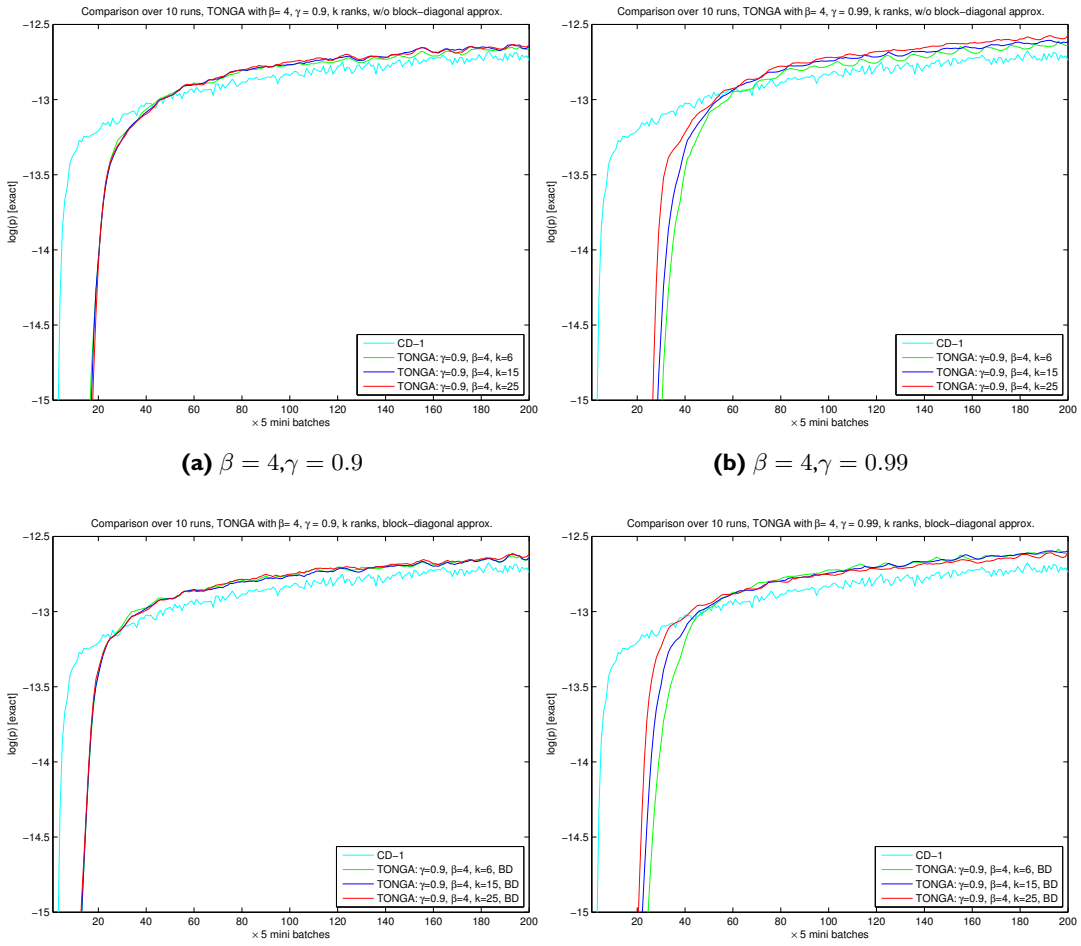
(d) $\beta = 4, \gamma = 0.99$, block-diagonal approximation

Figure 4.18: Results mini-batch Tonga for different k, with and without block-diagonal approximation. Using the centred covariance implementation of Tonga. While for some meta-parameters Tonga is slightly better than CD-1, instabilities and premature convergence can occur. Centred Tonga should not be used for training RBMs

The first experiments use the centred covariance version of Tonga. While centred natural CD-1 did not give good results, centred Tonga is the version used in the authors' latest publication ([Le Roux et al., To Appear 2010]) and also the setting that is theoretically justified by the Bayesian argument described in section 3.2.1, so we have to at least check if centred Tonga suffers from the same instabilities as centred natural CD-1. I only ran tests using a relatively high value for β (4), for low values for β, Tonga converges to CD-1. Instead, we have to investigate the effect of an additional meta-parameter: k, the number of retained eigenvectors/values for the low-rank approximation. Figure 4.18 shows results for different k for different settings for γ and β that have shown to be the most promising in the natural CD-1 test. Unfortunately, centred Tonga is unable to significantly improve on CD-1 training for any of the tried settings, and again instabilities can occur. BD-approximation yet again helps avoid the worst instabilities, but all in

all, centred TONGA is too sensitive to meta-parameter settings and too prone to instabilities to be useful. Slow initial learning, at least, is not an issue. This is the last experiment using the centred formulation of the natural gradient. While the centred version is theoretically justified, the results presented for both natural CD-1 and TONGA have shown that in practise, we should avoid using the centred natural gradient. It is simply too sensitive to meta-parameter settings and too often leads to instabilities with terrible training results.

4.5.1.2 Uncentred Tonga



(a) $\beta = 4$,$\gamma = 0.9$

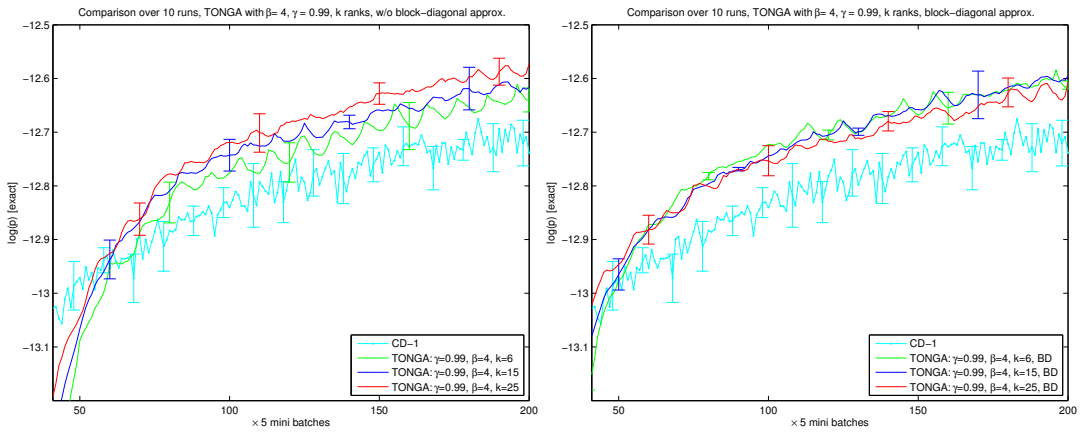
(b) $\beta = 4$,$\gamma = 0.99$

(c) $\beta = 4$,$\gamma = 0.9$, block-diagonal approximation

(d) $\beta = 4$,$\gamma = 0.99$, block-diagonal approximation

Figure 4.19: Results mini-batch Tonga for different k, with and without block-diagonal approximation. Results are obtained from 10 runs using different (random) initialisation. Uncentred Tonga shows much slower initial learning, but converges to a better data log-likelihood than CD-1 in the end. Figure 4.20 shows a close-up of the convergence phase

Using the uncentred covariance helped avoiding instabilities in the natural CD-1 experiments and results for centred TONGA show the same effect. Figure 4.19 shows the results over 10 independent runs for each of the displayed meta-parameter setting. No instabilities occur, but initial



(a) $\beta = 4$,$\gamma = 0.99$

(b) $\beta = 4$,$\gamma = 0.99$, block-diagonal approximation

Figure 4.20: Results mini-batch Tonga for different k, with and without block-diagonal approximation. Results are obtained from 10 runs using different (random) initialisation and the error-bar indicates 1 σ. Tonga significantly improves the convergence on the data log-likelihood compared to CD-1

learning is very slow. Yet, the model reaches a higher data log-likelihood during the convergence phase (figure 4.20) than CD-1. The number or retained eigenvalues has a only a minor influence on final convergence quality. Most likely this is caused by the fact that the covariance matrices for the 49×20 RBMs are quite small (980×980 without block-diagonal approximation and only 49×49 with block-diagonal approximation). This is supported by the observation that low values of k slightly decrease performance when not using the block-diagonal approximation (fig. 4.20a). When using the block-diagonal approximation (fig. 4.20b), using a lower k does not decrease performance: The 49×49 covariance matrices should be well approximated even when using $k = 6$,the lowest tried value for k.

4.5.1.3 Fully on-line Tonga

In all previous experiments, I used mini-batches of size of 60. Of course, Tonga can also be used fully on-line (*i.e.* mini-batch size of 1). Figure 4.21 shows results for this setting, comparing uncentred fully on-line Tonga to both mini-batch CD-1 and fully on-line CD-1. Tonga shows erratic learning and seems to diverge in the end. Fully on-line CD-1 on the other hand works fairly well and finds a state of similar data log-likelihood as mini-batch CD-1. Though only the result of a single run, Tonga does not seem to be a good choice when doing fully on-line training, therefore, I did not use fully on-line Tonga in any further experiments.

4.5.2 Does Tonga give a direction closer to the true gradient than CD-1?

We've seen that (for suitable meta-parameters) natural CD-1 and Tonga can significantly improve upon CD-1, but we do not yet know why. There may be at least three different explanations:

1. The Tonga direction simply is closer to the true gradient than the CD-1 directions

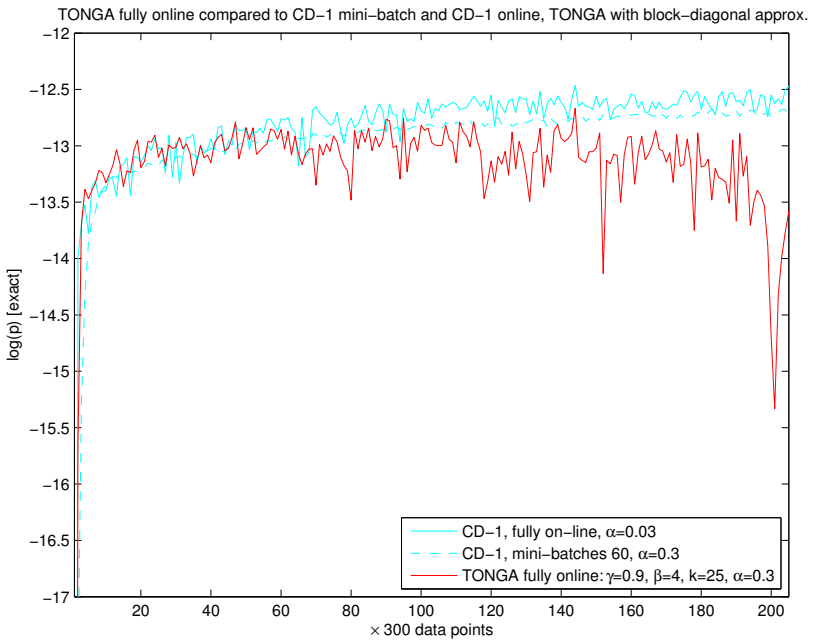


Figure 4.21: uncentred Tonga, fully on-line

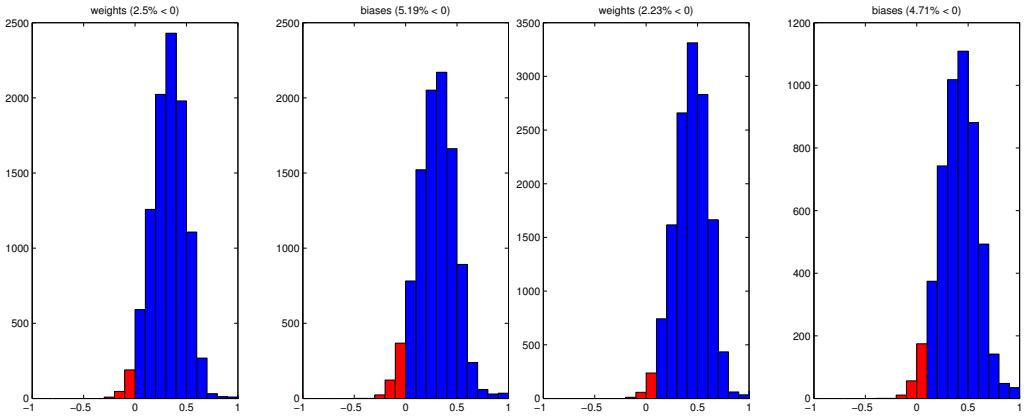
2. The Tonga direction isn't closer to the true gradient, but captures some aspect of the curvature of the optimisation function and points more directly towards towards the optimum

3. The Tonga direction simply works better together with the fixed step size α, set to 0.3 in all experiments

Since the small RBM used allows calculating the true gradient quickly, at least the first point is easily checked by simply comparing the direction given by the training regime ($\boldsymbol{d}$) to the true gradient ($\bar{\boldsymbol{g}}$) by calculating the dot product $\boldsymbol{d}^T\bar{\boldsymbol{g}}$. A value of 1 means complete agreement in direction. Negative values mean that $\boldsymbol{d}$ actually points more than 90° away from $\bar{\boldsymbol{g}}$.

To this end, I calculated to degree of agreement between the training direction and the true gradient for two Tonga training regimes, the first using the most promising meta-parameters found so far: $\gamma = 0.99$, $\beta = 4$, block-diagonal approximation, uncentred covariance. The second regime uses identical meta-parameters, but no block-diagonal approximation. k is set to a relatively high value of 25, so that Tonga can work with a good approximation even when not using BD-approximation. Both regimes have shown good performance in previous experiments, shown in 4.20. Results are obtained from training on the 10k 7×7 data-set, 10 runs for each training regime. Training lasted for 1000 mini-batches (= 6 epochs for the 10k data-set). This gives a total of 10,000 gradient comparisons per training regime.

To see if there are any interesting differences in the quality of the Tonga direction, I split the total gradient in a weight and bias part and compare to the true gradient separately. The true gradient is obtained from Gibbs sampling using mean field updates until convergence is reached, where convergence is declared when the Frobenius norm of the difference between three consecutive gradient estimates falls below 0.0001. To avoid declaring convergence prematurely, I always



(a) CD-1 **(b)** CD-1, weighted by norm

Figure 4.22: Comparison of training direction to true gradient *over the full data-set*. The histogram shows the values for $\boldsymbol{d}^T\bar{\boldsymbol{g}}$, where $\boldsymbol{d}$ is the direction given by the mini-batch training regime and $\bar{\boldsymbol{g}}$ is the true gradient over the full 10k data-set, using the same set-up as for the results presented in figure 4.23. In comparison, the CD-1 direction follows the true gradient much better

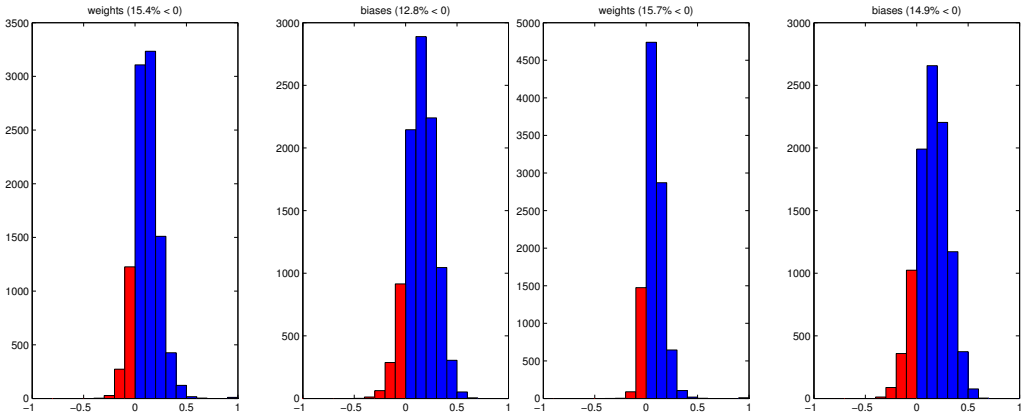
run at least 20 steps of Gibbs sampling. This has shown to be more than enough when using mean-field updates, but I confirmed the result with a second run doing normal Gibbs sampling using at least 1000 steps (this is very slow and entirely more than necessary, but the confirmation should give us more confidence in the results). Then the average gradient over the full data-set is taken as the current true gradient. The true gradient is re-calculated after every parameter-update (*i.e.* once per mini-batch).

Figure 4.22 shows a histogram of $\boldsymbol{d}^T\bar{\boldsymbol{g}}$ for CD-1 training. CD-1 gives a sign-disagreement of at most around 5%, *i.e.* following the CD-1 direction for a suitably small step size α should increase the data log-likelihood at least 95% of the time. This value confirms the results reported in [Bengio and Delalleau, 2009].

Figures 4.23 show the results when comparing the direction given by the two TONGA regimes to the true gradient. I find the result is very surprising: The natural methods show a sign-disagreement of up to 15%. That is, 15% of all training directions actually point away from the true gradient and therefore, if followed, decrease the data log-likelihood. This is 3 times the rate of CD-1. Yet, the two TONGA regimes actually lead to a better final convergence, as shown over 10 trials in figure 4.20.

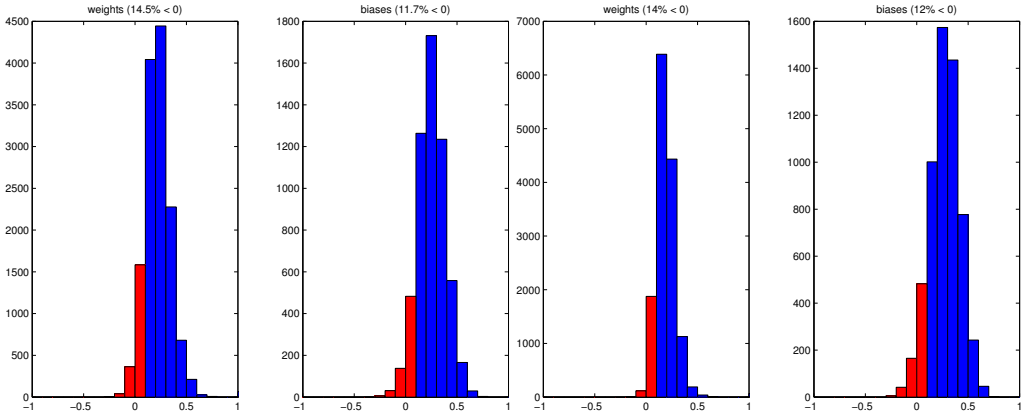
Both histogram figures also show the results when the product is weighted by the norm of the training direction. This shows, that the effect is not simply caused by gradients that point in the wrong direction, but only have a very small norm (and thus a small effect on training).

This answers the question if TONGA gives a better estimate of the true gradient: it does not. Indeed, I'm surprised that TONGA can work so well despite the high sign-disagreement. But it is also pretty unlikely that TONGA would somehow predict the direction of future gradients and can follow a more direct path towards an optimum. I believe the most likely reason is simply that the fixed step size of 0.3 is too high and that the more frequent sign-disagreements have a similar effect to decreasing the step size ("six steps forward and one step back"). The next experiment



(a) Tonga, $\beta = 4$, $\gamma = 0.99$, block-diagonal approximation, uncentred

(b) Tonga, $\beta = 4$, $\gamma = 0.995$, no approximation, uncentred



(c) Tonga, $\beta = 4$, $\gamma = 0.99$, block-diagonal approximation, uncentred, weighted by norm

(d) Tonga, $\beta = 4$, $\gamma = 0.995$, no approximation, uncentred, weighted by norm

Figure 4.23: Comparison of training direction to true gradient *over full data-set*. The histogram shows the values for $\boldsymbol{d}^T \bar{\boldsymbol{g}}$, where $\boldsymbol{d}$ is the direction given by the mini-batch training regime and $\bar{\boldsymbol{g}}$ is the true gradient over the full 10k data-set. (both normalised to unit length). The graph includes data from 10 runs each for 2 epochs (*i.e.* 333 mini-batches each). The second row weights the product by the norm of the gradient. This shows that gradients, that point away from $\bar{\boldsymbol{g}}$ on average only have a slightly smaller norm. Figure 4.22 shows the results for CD-1 for comparison. This result is surprising: Training by Tonga outperforms CD-1, even though CD-1 follows the true gradient much better

gives some support to this conjecture, showing that the optimal step size is far below 0.3.

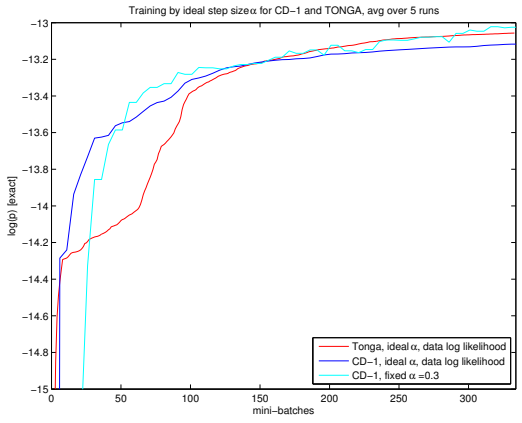
4.5.3 Step size selection for Tonga

So far, I have completely avoided the topic of step size selection. I simply used $\alpha = 0.3$ based on some experiments using CD-1 with different values for α between 0.2 and 1. Training with lower values for α did find states with better data log likelihood in the convergence phase, but are a bit slow in the beginning. Increasing α led to a decrease in convergence quality. 0.3 seemed to be a good compromise. Using only a single, fixed step size has much simplified the direct

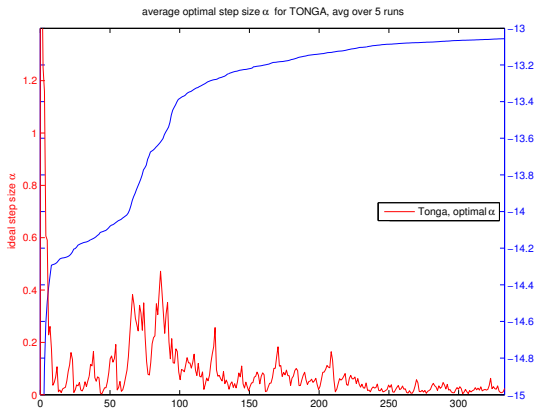
comparison between the natural methods and CD-1. Also, by scaling the direction given by the natural methods to have the same norm as the CD-1 direction, I argued that a direct comparison between regimes using the same, fixed step size gives a good enough comparison to decide if either regime outperforms the other. Yet, the previous experiments have given some cause for concern that this assumption may not have been valid: CD-1 seems to enter the convergence phase fairly quickly and may benefit from a lower step size. Tonga on the other hand suffers from very slow learning in the beginning, and can possibly benefit from choosing a larger step size.

I therefore ran a series of experiments calculating what I will call the "optimal step size" for every single training direction. I will use the term "optimal step size" in the sense that it is the step size that globally maximises the objective function when following a training direction $\boldsymbol{d}$ for a variable, but positive step size α. I obtain an estimate of the optimal step size by a simple line-search method. This is feasible for very small RBMs, because we can evaluate the objective function (*i.e.* the data log-likelihood) exactly in a reasonable amount of time. Because available implementation time was scarce and CPU time plenty, the line-search method used is embarrassingly simple: I start with a minimum step-size of 0 and choose a maximum expected step size based on a small multiple (1.5) of the last step size. I then use a trial increase in step size based on this interval, by dividing the length of the interval by 15. This gives trial step sizes (α_i). I then simply evaluate the objective function for $f(\boldsymbol{\theta} + \alpha_i \boldsymbol{d})$, starting at the minimum step size. As soon as an increased step size leads to a decrease in the objective function, I accept the previous step size. This roughly finds the best positive step size (since the objective function in a binary RBM is convex). If the optimal step size is negative, step size 0 is used instead, simply to avoid going in the opposite direction as $\boldsymbol{d}$.

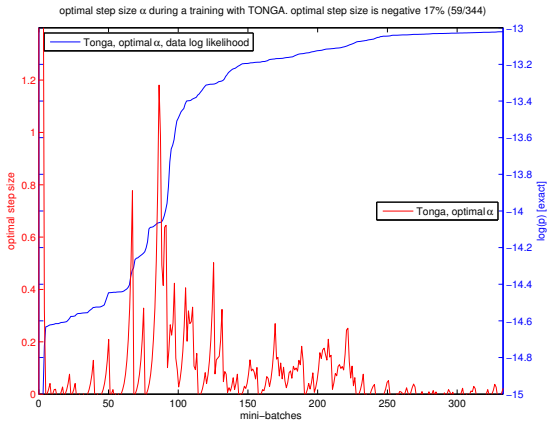
Figure 4.24 shows the results for this comparison, both for CD-1 and Tonga. Figure 4.24a shows that following the direction given by Tonga-CD-1, even when using the ideal step size, leads to much slower learning than both CD-1 with fixed α and CD-1 with optimal α for the first 125 mini-batches. Only in later batches can Tonga catch up with CD-1. Figure 4.24c gives an indication why: It shows the optimal step size found for a single run as well as the data log-likelihood. We can see that after an a few initial high optimal step sizes of about 1.5 for the first 3 mini-batches, the optimal step size is very low (or negative) for the following 60 batches and the data log-likelihood does not increase by much. Then, quite suddenly, the optimal step size increases by a large margin and learning is fast for about 100 mini-batches, before finally entering the convergence phase. Figure 4.24b shows the optimal step size averaged over 5 runs, and it seems that the single run is representative of all 5 runs: again, the optimal step size is very low or negative for the first 50 batches (excluding the first 5), then increases, before falling off again in the convergence phase. The data log-likelihood curve during training is not concave, *i.e.* the rate of learning is lower in the first 75 mini-batches than in the following 75. 4.24d gives the result for a CD-1 training run and we can see that CD-1 does not suffer from this phenomenon: The data log-likelihood curve is concave and in direct comparison to 4.24c, the optimal step size does not have significant spikes and rather seems to (very noisily) follow a curve inverse proportional to the training time. This actually nicely follows theoretical consideration for optimal step size selection in stochastic processes ([Robbins and Monro, 1951]).



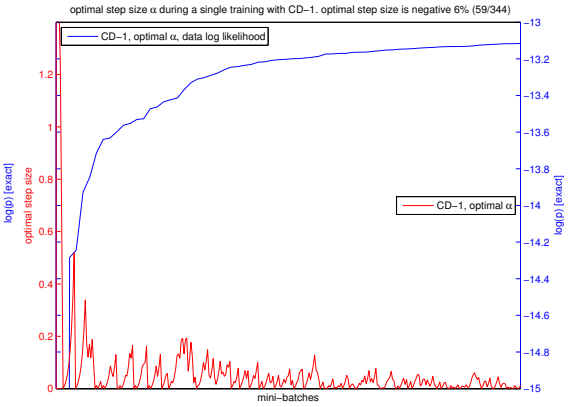
(a) Comparison:CD-1 and Tonga trained with optimal α compared to CD-1 trained with fixed $\alpha = 0.3$, averaged over 5 runs



(b) Tonga trained with optimal α, averaged over 5 runs, also showing the optimal found α (red)



(c) Tonga trained with optimal step size, single run



(d) CD-1 trained with optimal step size, single run

Figure 4.24: Training with optimal steps size found by line search.

Again, this is a somewhat surprising result. Having a big initial step size is reasonable. In fact, for the first few mini-batches n, the covariance matrix has at most rank n, that is, all but n eigenvalues will be zero. Using regularisation (3.49) before calculating the inverse of the covariance matrix will have the effect that the inverse of the covariance matrix is close to identity during the first few runs, and the TONGA direction is very close to CD-1, and it is known that we can use relatively high step size for CD-1 in the beginning of training. It is not clear, however, what then makes the TONGA direction so bad in the following few dozen mini-batch gradient evaluations. But it shows that the initial slow learning cannot be helped by simply increasing the initial step size. For both CD-1 and TONGA, the average optimal step size is far below 0.3 over most of the training duration. At least for CD-1, we can see that the optimal step size decreases during training. This indicates that using a decaying step size should give better results, and I will use a decaying step size in a later experiment.

It is also interesting to see, that CD-1 with fixed α actually converges to a slightly better data log-likelihood compared to CD-1 with the optimal step size. This is probably due to the well

known problem of gradient descent tending to oscillate ("zig-zag") along ridges. The fixed step size may help to jump out of local ridges and then find a better route to the global optimum.

4.5.3.1 Conclusions

We've seen that centred TONGA suffers the same instability problems as centred natural CD-1. This method should therefore be avoided. Fully on-line TONGA likewise gives unstable results and should also be avoided. Uncentred mini-batch TONGA, on the other hand, did improve on CD-1 in the small scale setting and is far less sensitive to meta-parameter settings. However, it seriously suffers from slow learning in the beginning. It is also not clear why TONGA works at all – the TONGA direction points in a direction opposite that of the true gradient in well over 10% of the mini-batches, and this does not instill trust in the method. We have to fear that the good results are at least partially a side-effect of using too high a step size for this particular data-set.

4.6 Estimation error of the AIS partition function estimation

4.6.1 Setup

For the next set of experiments, I used the full sized MNIST data set (*i.e.* images of size 28×28, giving a visible layer of size 784). As explained in section 4.2.2, we do not yet have a reliable way to estimate the data log-likelihood for large RBMs: The AIS method I used in the initial experiments gave blatantly inaccurate estimates and exact evaluation of the data log-likelihood is intractable for large RBMs. The most likely culprit for the bad quality of the AIS estimate seemed to be numerical inaccuracy when the weight parameters become too high. When weight parameters become even higher, numeric overflow occurs, and the Matlab routines only return `inf` as the estimated data log-likelihood. Some investigations showed two simple changes to improve the accuracy:

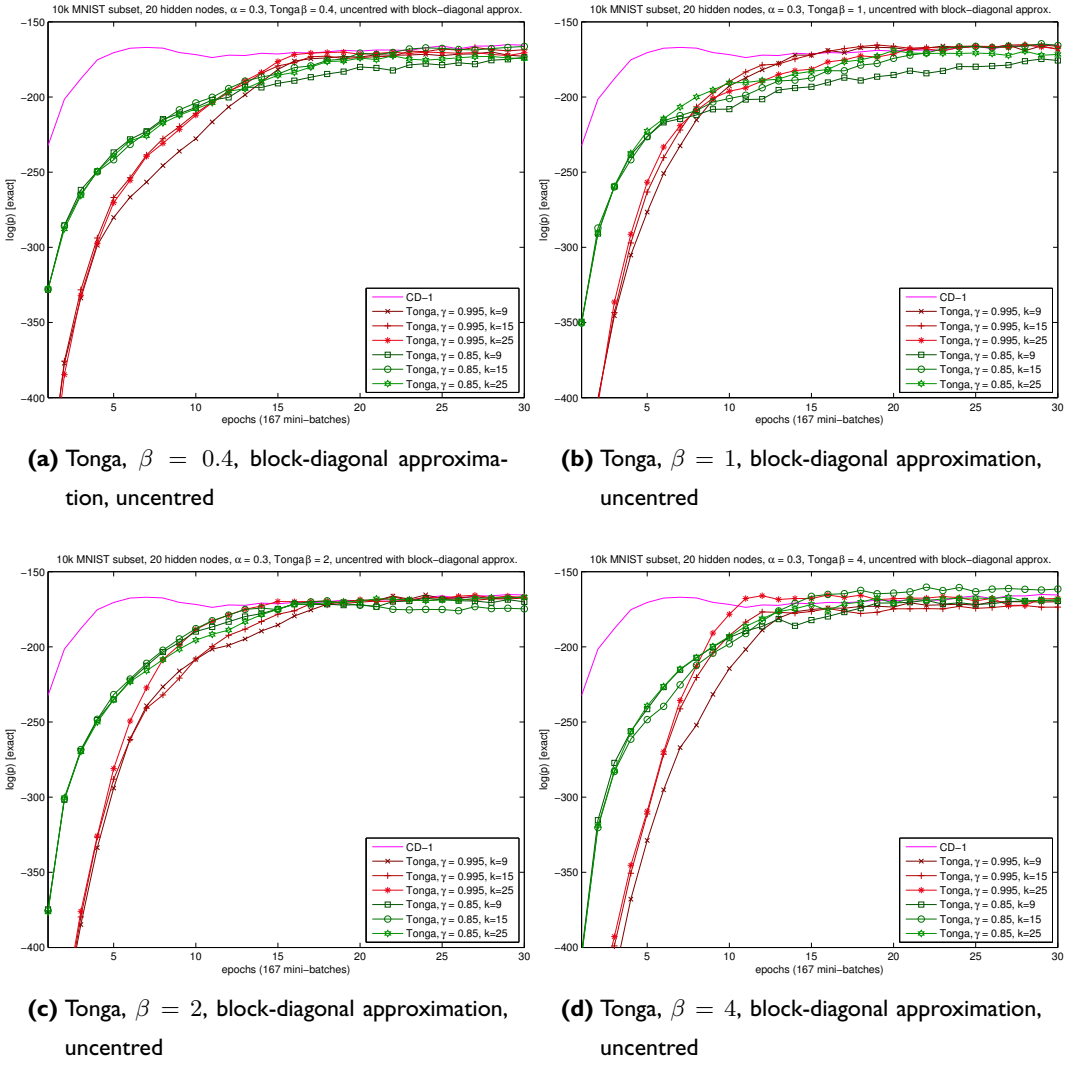
- To reduce memory consumption and possibly to improve performance, the DEEP framework uses the `single` precision floating point data type for storing the training data set. Matlab then, unless specifically instructed to do otherwise, automatically uses `single` precision also for the routines that calculate the un-normalised data log probability (2.9) that use the data set as input. A simple cast to `double` precision should improve accuracy considerably

- Calculating the free energy function (2.8) to obtain the un-normalised data log-likelihood requires repeated evaluation of $\log(1 + e^x)$ with $x = \boldsymbol{v}^T W_j + c_j$. This caused the observed numeric overflow for large values of the exponent. To avoid the overflow, I simply replaced $\log(1 + e^x)$ by the approximation $\log(1 + e^x) \approx x$ for $x > 70$. The error for this approximation is well below the accuracy achievable when working with `double` floating point for the not approximated $\log(1 + e^x)$

I then ran a series of tests using a RBMs with a hidden layer of size $K = 20$. Since Z can be evaluated in time exponential to the hidden layer (2.11), it is still possible to calculate the exact data log-likelihood. I also calculate the AIS estimate for comparison, to get an idea of how large

the estimation error is, and to find out, if the error can be made sufficiently small by using an appropriately large weight-decay factor κ.

4.6.1.1 Results



(a) Tonga, $\beta = 0.4$, block-diagonal approximation, uncentred

(b) Tonga, $\beta = 1$, block-diagonal approximation, uncentred

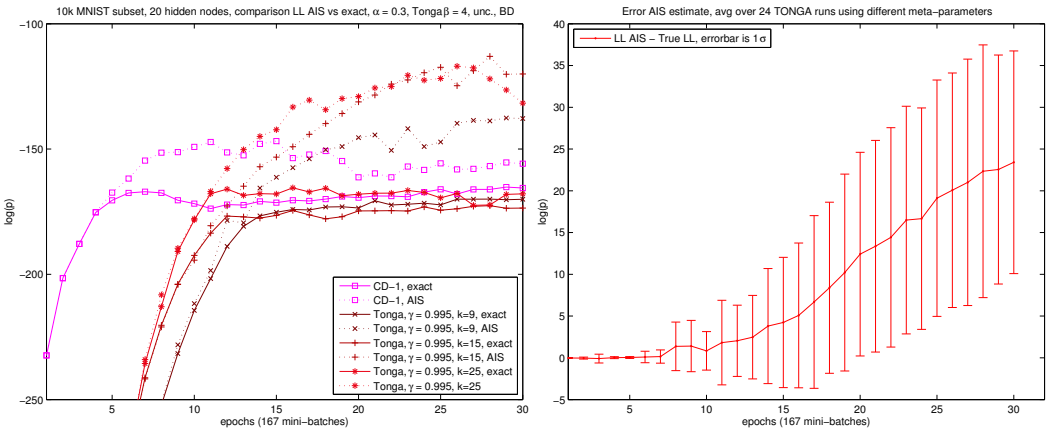
(c) Tonga, $\beta = 2$, block-diagonal approximation, uncentred

(d) Tonga, $\beta = 4$, block-diagonal approximation, uncentred

Figure 4.25: Tonga on MNIST, 20 hidden nodes

The experiment uses again a step size of 0.3. While this may have been too large for the re-sampled set, I expected it to be a rather low setting for the full MNIST set using mini-batches of size 60. Figure 4.25 shows results comparing TONGA with various settings for meta-parameters γ, k and β. Unfortunately, TONGA could not significantly improve on CD-1 in any of the experiments. Again, initial learning is very slow: While CD-1 reaches approximate convergence already after about 5 epochs (at which state, presumable, α again is too high to allow further convergence), TONGA takes about 3 times as many iterations, not a good result for TONGA and it seems unlikely at this stage, that using TONGA for RBMs is a viable method. We would never want to use TONGA during the initial phase – it's far too slow. At later stages, some experiments have

shown somewhat better convergence properties. But it is almost certain, that at least the same convergence can be reached, simply by fine tuning the training regime (*e.g.* using a decaying step size) for much less computational cost. I ran some additional tests, using an initially larger, then decaying step size and also tried to dynamically adjust γ to start at a low value and then increase at later epochs. But none of those experiments could convincingly increase the slow learning at the beginning or significantly outperform CD-1 in the convergence phase. I thus concluded that TONGA is unlikely to give consistently better results than CD-1 and it is not a recommended method for training RBMs.



(a) Example of AIS estimate compared to exact likelihood

(b) Average error and standard deviation of the AIS estimate

Figure 4.26: AIS estimation error. **Left**: We can see that the AIS estimate coincides with the exact calculation for the first stages of training. At later stages AIS gives a far to optimistic estimate. **Right:** Average error and deviation for the AIS estimate. We can again see, that the estimate starts at a high quality, but then gives an increasing error. On average, AIS underestimates the partition function which leads to an overestimation of data log likelihood

Additional, In an attempt to overcome the AIS estimation error, I compared the AIS estimate of the data log-likelihood to the exactly calculated data log-likelihood. Figure 4.26a plots the AIS and the exact calculation for a sub-set of the experiments. This shows that AIS gives only a very low error for the first few epochs but then diverges further and further from the exact value. It also shows how important it is to develop ideas to avoid having such a large estimation error: If we hadn't calculated the exact likelihood for comparison, we may not have noticed that AIS overestimates the data log-likelihood by a large margin. There is no immediately obvious indication that the estimate may be wrong: The data log-likelihood estimate by AIS smoothly increases during training and in some ways looks even more legitimate than the exact evaluation. AIS also doesn't estimates so blatantly wrong, that they are immediately obvious, as we've seen in 4.5. Without the exact log-likelihood for comparison, we may thus have wrongly concluded that TONGA significantly outperforms CD-1 in this test. Figure 4.26b shows the mean error and standard deviation for the 24 test-runs. We can see that initially the error is very low, but then increases quite rapidly.

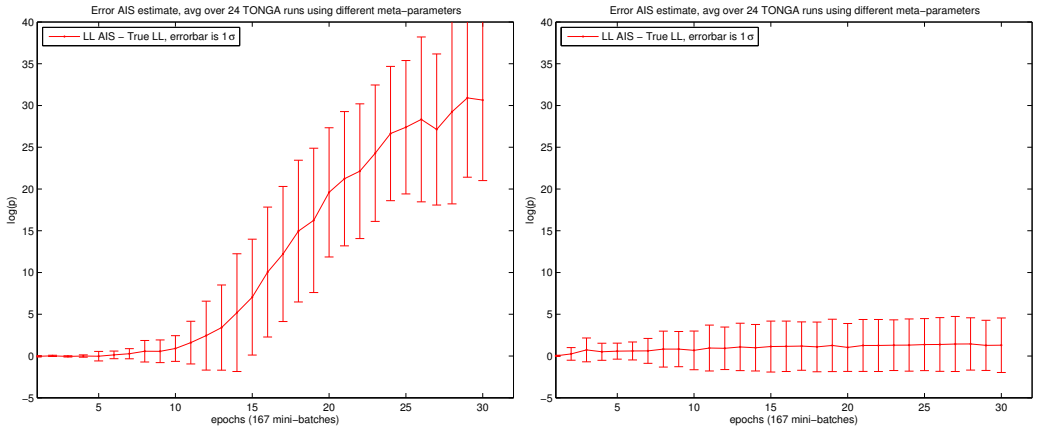
We should at least give some quick consideration to the possibility that in fact the "exact" calculation may have numerical problems. There are three very good reasons to think otherwise:

1. We've already observed that AIS overestimates the data log-likelihood in the initial tests, where AIS often even gave positive data log-likelihood.

2. Calculating the exact log-likelihood is computationally difficult, but conceptually simple: it only involves repeated computation of free energy terms (2.8). While it may be possible, that the numerical error from calculating many $\log(1 + e^x)$ terms adds up over the huge number of iterations, it is much more likely that it is the AIS sampling algorithm that gives bad estimations.

3. The data log-likelihood that AIS gives in 4.26a is far better than results given in [Tieleman and Hinton, 2009] for RBMs with 25 hidden units, well-trained on MNIST. This is highly unlikely.

As described in section 4.2.2, I've found that using weight-decay reduced at least the occurrence of blatantly wrong data log-likelihood estimates, but had no good way to evaluate how big the AIS error was, nor to quantify, how high a decay rate we should use. With the small hidden layer, this becomes much simpler, as we can directly compare the exact data log-likelihood to the AIS estimate and check if and how the error changes for differently large weight-decay factors. [Hinton, 2010] recommends to start with an L2 weight-decay factor $\kappa = 0.0001$ normally, but to use twice that value when using AIS to measure performance. As figure 4.27a shows, using

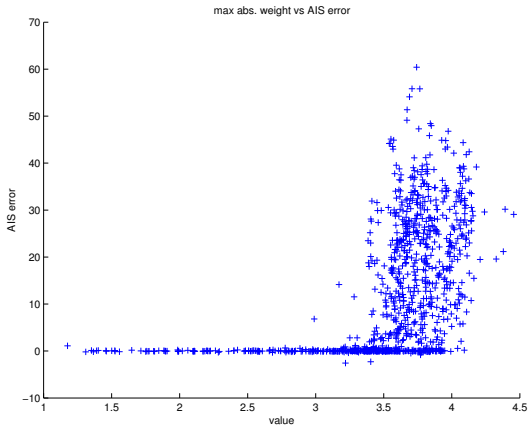


(a) AIS error for $\kappa = 0.0002$ **(b)** AIS error for $\kappa = 0.002$

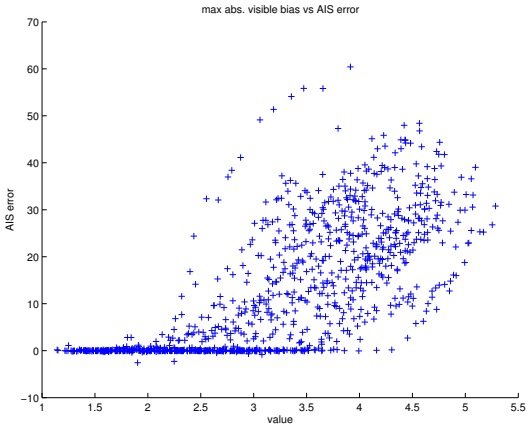
Figure 4.27: AIS error when using L2 weight decay with $\kappa = 0.0002$ and $\kappa = 0.002$. $\kappa = 0.0002$ does not reduce the AIS error. $\kappa = 0.002$ reduces mean error to almost 0 and also decreases variance. Tests used for the second graph used a higher initial step size, this explains the higher error seen in the first few epochs

a weight decay of $\kappa = 0.0002$ isn't sufficient to avoid the high estimation error. When using $\kappa = 0.002$ and also using the weight decay term for the biases (normally, weight decay is only recommended for the weights, not the biases), the AIS estimation seems to work well. Unfortunately, experiments showed that such a high weight decay also considerably hurts training. It

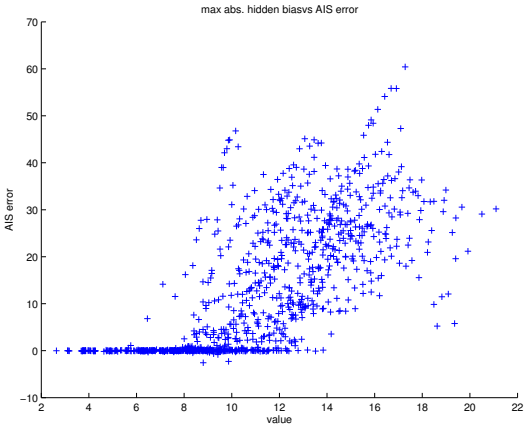
may be possible to avoid such a high weight decay, and still obtain a good estimate, but there is a drawback: As figure 4.28 shows, the error seems to be mostly related to too high biases settings. We can simply limit the values allowed for the bias. However, high values for the biases is often a desired property of RBMs and improves learning. Therefore, AIS (in the Matlab implementation at least) should probably only be used for some initial tests, where we trade-off the disadvantage that limiting the bias term will decrease the achievable data log likelihood, with the advantage of having a somewhat reliable estimator of the objective function. We can then disable the limit, once we're confident that we've found a good training regime.



(a) AIS error plotted against maximum, absolute weight



(b) AIS error plotted against maximum, absolute visible bias



(c) AIS error plotted against maximum, absolute hidden bias

Figure 4.28: This shows the correlation between the AIS error and the maximum (absolute) terms for the parameters of the RBM. The AIS error is most effected by large bias terms. If we want to use AIS, we should either hard-limit the bias terms or use weight decay also for the bias terms

Chapter 5

Conclusions

The goal of this project was to investigate if the training of Restricted Boltzmann Machines can be improved by using the natural gradient. Initial full scale experiments suggested otherwise, but evaluation of the training data log likelihood as a measure of performance proved difficult. The estimate given by AIS was blatantly inaccurate and not very useful. I therefore switched to a much smaller problem size, where both the exact data log-likelihood and the true gradient can be evaluated.

I then tried to systematically find reasons why the natural gradient method did not perform well. Starting with a straightforward implementation of the natural gradient, I gradually moved towards an efficient implementation, Tonga. This allowed to first investigate if the natural gradient method works at all, without having to take into account all the additional meta-parameter that Tonga introduces at once. The small scale experiments showed that using the centred formulation of the natural gradient can lead to instabilities, but this can be avoided by using the uncentred formulation instead. Using Tonga fully on-line likewise leads to instabilities and unreliable training and cannot be used. I've shown that the covariance matrix for the weights of RBM is not well approximated by the block-diagonal approximation, that only retains the covariance that occurs with in the set of weights that connect a single hidden node to all visible nodes. This differs from the results for ANNs given in [Le Roux et al., 2008]. Still, using the block-diagonal approximation did at least not hurt learning performance, and also helped avoiding instabilities. Using Tonga's low-rank approximation also did not seem to adversely affect training performance. But while natural learning did improve on the data log-likelihood convergence on the small-scale set there are some indications, that the method may not work well in general: the direction given by natural learning points away from the true gradient about 15% of the time, much more often than CD-1. The natural learning methods also all showed very slow learning during the early stages of training and experiments using the optimum step size indicated that this problem cannot simply be overcome by choosing an appropriate step size. Indeed, in none of the experiments did natural learning improve convergence during the initial phases of training. Therefore, the recommendation can only be to *not* use natural learning in the beginning. Only during the convergence phase have natural CD-1 and Tonga shown some limited merit: In the small scale experiments at least, both training methods could train models to converge to a better data log likelihood compared to CD-1. So it is possible, that initially using

CD-1 and switching to Tonga-CD-1 at later stages could help finding marginally better states during the final convergence phase. However, I could not even reproduce good convergence result on the full MNIST set and at this stage I find it unlikely that the additional computational cost needed for Tonga can be justified. It is very possible, that any improvements seen for Tonga in the small models could simply have been achieved by using a better meta-parameter regime for CD-1, *e.g.* using a decaying step size to improve final convergence and using momentum, to overcome some of the noise inherent in CD-1 directions, for a much cheaper cost than calculating the natural gradient. Unfortunately, after bad initial decisions and implementation problems, I did not have enough time to run experiments, conclusively showing if this is indeed the case. When considering weather to use Tonga or not, we also have to take into account, that RBMs already have a large number of meta-parameters that we all need to find good values for. Tonga then adds yet more parameters to be chosen, checked and validated.

I've also shown that the AIS sampling method is prone to numeric errors, leading to overly optimistic estimates of the training data log likelihood. This may potentially also affect published results that use the AIS method as an additional measure of performance, though all publications known to me that use AIS (*e.g.* [Tieleman, 2008]), do not base their conclusions on AIS alone, but also give other performance measures, such as detection rates when the RBMs are used for classification tasks.

5.1 Future work

A more thorough comparison of Tonga and CD-1, especially on larger scale problems, should be undertaken to confirm or disprove my claim, that Tonga is not an efficient method for training RBMs. More exhaustive test on larger data-sets may show, to what extent the improvement in convergence seen in the small models can translate to large scale problems. One should also make sure, that any improvements observed could not much more easily be accomplished by using a fine-tuned training regime, using the right balance of variable step size, weight decay and momentum. If so, this would be computationally much less expensive than using Tonga. It would also be interesting to investigate, if Tonga can improve generalisation properties. However, since Tonga seems to push RBMs to more extreme states during training, this seems very unlikely. It is possible that Tonga may be used with more success when using CD-n with a larger n as the underlying gradient as this would give a less noisy and biased estimate of the true batch-gradient. However, CD-n is generally thought of too inefficient and even if Tonga would be able improve in this setting, it would be of limited value.

Bibliography

Y. Bengio. Personal Communication, 2010.

Y. Bengio and O. Delalleau. Justifying and generalizing contrastive divergence. *Neural Comput.*, 21(6):1601–1621, 2009. ISSN 0899-7667. doi: http://dx.doi.org/10.1162/neco.2008.11-07-647. Revised 2009 version (not the 2007 version).

Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, U. D. Montréal, and M. Québec. Greedy layer-wise training of deep networks. In *In NIPS*. MIT Press, 2007.

L. Bottou. Stochastic learning. In O. Bousquet, U. von Luxburg, and G. Rätsch, editors, *Advanced Lectures on Machine Learning*, volume 3176 of *Lecture Notes in Computer Science*, pages 146–168. Springer, 2003. ISBN 3-540-23122-6.

L. Bottou and Y. LeCun. Large scale online learning. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004. URL `http://leon.bottou.org/papers/bottou-lecun-2004`.

L. Bottou, O. Bousquet, and G. Zuerich. The tradeoffs of large scale learning. In *In: Advances in Neural Information Processing Systems 20*, pages 161–168, 2008.

M. A. Carreira-Perpinan and G. E. Hinton. On contrastive divergence learning. *Artificial Intelligence and Statistics*, 2005.

A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis, 2nd Edition*. Chapman & Hall/CRC, 2003.

G. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:2002, 2000.

G. Hinton. A practical guide to training restricted boltzmann machines. Technical Report UTML TR 2010–003, Geoffrey Hinton Department of Computer Science, University of Toronto, 2010. Version 1.

G. E. Hinton. Learning multiple layers of representation. *Trends in Cognitive Sciences*, 11:428–434, 2007.

G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, 2006. ISSN 0899-7667. doi: http://dx.doi.org/10.1162/neco.2006.18.7.1527.

H. Larochelle and Y. Bengio. Classification using discriminative restricted boltzmann machines. In *In ICML '08: Proceedings of the 25th international conference on Machine learning. ACM*, 2008.

A. Le Roux, N. L. Fitzgibbon. A fast natural newton method. *Proceedings of the 27th International Conference on Machine Learning*, 2010a.

N. Le Roux. Personal Communication, 2010b.

N. Le Roux and Y. Bengio. Representational power of restricted boltzmann machines and deep belief networks. *Neural Computation*, 20(6):1631–1649, 2008.

N. Le Roux, P.-A. Manzagol, and Y. Bengio. Topmoumoute online natural gradient algorithm. In J. C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 849–856. MIT Press, Cambridge, MA, 2008. URL `http://books.nips.cc/papers/files/nips20/NIPS2007\_0056.pdf`.

N. Le Roux, Y. Bengio, and A. Fitzgibbon. Improving first and second-order methods by modelling uncertainty. In S. Sra, S. Nowozin, and S. J. Wright, editors, *Optimization for Machine Learning*. The MIT Press, To Appear 2010.

Y. Lecun and C. Cortes. The mnist database of handwritten digits. URL `http://yann.lecun.com/exdb/mnist/`.

B. M. Marlin, K. Swersky, B. Chen, and N. de Freitas. Inductive principles for restricted boltzmann machine learning. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 9, 2010.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, August 2000. ISBN 0387987932.

A. K. Noulas and B. J. A. Kröse. Deep belief networks for human computer interaction. In *AFFINE Workshop, International Conference on Multimodal Interfaces*, 2008.

H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

R. Salakhutdinov. Learning and evaluating Boltzmann machines. Technical Report UTML TR 2008-002, Department of Computer Science, University of Toronto, June 2008.

R. Salakhutdinov and I. Murray. On the quantitative analysis of deep belief networks. *Proceedings of the 25th International Conference on Machine Learning*, 2008.

R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 791–798, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3. doi: http://doi.acm.org/10.1145/1273496.1273596.

P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart, J. L. McClelland, et al., editors, *Parallel Distributed Processing: Volume 1: Foundations*, pages 194–281. MIT Press, Cambridge, 1987.

I. Sutskever and T. Tieleman. On the convergence properties of contrastive divergence. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, Volume 9 of JMLR: W&CP 9, 2010.

Y. W. Teh and G. E. Hinton. Rate-coded restricted boltzmann machines for face recognition. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *NIPS*, pages 908–914. MIT Press, 2000.

T. Tieleman. Some investigations into energy-based models. Master's thesis, University of Toronto, 2007.

T. Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 1064–1071, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: http://doi.acm.org/10.1145/1390156.1390290.

T. Tieleman and G. Hinton. Using fast weights to improve persistent contrastive divergence. *Proceedings of the 26th International Conference on Machine Learning*, 2009.