Natural Boltzmann Machines

Adrian Millea - am2714@ic.ac.uk
Supervisor: Abbas Edalat - ae@ic.ac.uk
Department of Computing
Imperial College London

Abstract

Deep networks require heavy computing capabilities to learn classifications tasks, especially if they deal with large datasets as ImageNet. Different methods exist to speed up learning, but usually with significant more computation per epoch. For example, some second order methods, make use of the curvature matrix, which is hard to compute for high dimensional spaces. Moreover it changes in time, as the parameters themselves change, so it needs updating regularly. The inverse Fisher Information Matrix (FIM) plays the role of curvature matrix and models dependencies between model parameters. When the FIM is the identity map, the stochastic gradient (SGD) descent is equivalent to the natural gradient [Amari, 1998], which is optimal in some sense. In short, transforming into the space where FIM is the identity map, amounts to multiplying the original parametrization of the network with the square root of the inverse FIM. This is an expensive matrix decomposition operation. For example, the singular value decomposition cost is cubic in the number of parameters of the network. We propose two alternatives: an approach where we incrementally decorrelate the Fisher by adding covariance penalty cost to the original cost function of the network, without adding any parameters to the network, and a second approach where we augment the network with a set of whitening parameters, that learn (through the optimization of a contrast function) to decorrelate the network activations.

1 Introduction

When dealing with high dimensional parametric models like deep networks, the cost functions defined on such models are usually highly non-convex. Optimization methods on such spaces require high computational resources. One of the main issues is that the dependency between the parameters of the model induces a highly non-Euclidean geometry. The geometry is generally not known and makes the usual approaches like (stochastic) gradient descent work far from the optimum regime. A solution to finding out the dependency between the parameters of the model, is to compute the Fisher Information Matrix - FIM (or an approximation to it), which is the metric tensor of choice [Rao, 1945] on such parametric spaces. It has been shown that the Fisher metric has some unique properties with respect to the family of parametric probabilistic models (invariance with respect to the sufficient statistics) [Cencov, 2000].

2 Related work - The FIM for deep models

From the perspective of differential geometry, the FIM is the metric tensor defined on the space of all possible models defined by the set of parameters. It is a way of taking into account the dependency between the parameters when moving onto this high dimensional set (called a manifold). For statistical models, the Fisher matrix is defined as:

$$G = \mathbb{E}_{p(x;\theta)}\big[\frac{\partial \log p(x;\theta)}{\partial \theta^i}\frac{\partial \log p(x;\theta)}{\partial \theta^j}\big]$$

where θ are the parameters of the model and x is the given data. So $p(x;\theta)$ is the probability function of the input and the model parameters. The FIM is $n \times n$, describing the relation of each parameter with every other parameter. In models where this probability function is known (e.g. Boltzmann machines), then the FIM has a more tractable form, however, most of the time, we don't know the actual form of $p(x;\theta)$ and this needs to be approximated as well. The new direction has been shown to be the optimal gradient move and is called the natural gradient [Amari, 1998], defined as:

$$\nabla_N = G^{-1}\nabla_\theta L$$

where G is the Fisher matrix, and $\nabla_\theta L$ is the standard Euclidean gradient of the loss function L with respect to the parameters of the model θ.

2.1 Whitening in parameter space

One of the interesting approaches to approximating the Fisher matrix is to change the representation of the model, or the parametrization. The goal is to search for a transformation of the original (canonical) parameter space to a new space where the Fisher matrix is the identity map, the normal Euclidean metric. This has been shown to be equivalent to doing whitening in parameters space [Sohl-Dickstein, 2012]. (to be more exact, it is actually whitening in the space of the network statistics). The main idea is to decorrelate the statistics such that the units of the network have independent activations and unit variance. In this new space, the metric is

the identity map and thus the standard Euclidean gradient is equivalent to the natural gradient. For **deterministic neural networks** this has been developed as natural neural networks [Desjardins *et al.*, 2015], which uses the eigendecomposition of the expectation (under some independency assumptions of the underlying distribution) of the (uncentered) covariance matrix of the activation vector h (layer-wise), given by:

$$\mathbb{E}\left[hh^T\right] = U\Sigma U^T$$

where U is the matrix of eigenvectors and Σ is the diagonal matrix of eigenvalues. The transformation applied to the original space, giving the identity matrix as the metric has been shown to be $U\Sigma^{-\frac{1}{2}}$, which, we see, is exactly the standard whitening transformation.

3 Background - RBM

For **stochastic neural networks**, in particular, Restricted Boltzmann machines (RBM), the same idea has been applied by making use of a sparse graphical model to model the dependency between variables (with some independency assumptions as well, or assuming the visible and hidden states are binary). When assuming a dense graphical model structure, this has been shown to be equivalent to the Cholesky factorization of the covariance matrix of the network statistics [Grosse and Salakhudinov, 2015]. The Restricted Boltzmann machine has a set of visible units $\mathbf{v}$ which models the data, and a set of binary hidden units $\mathbf{h}$ which detect features in the input data. The two sets of units are connected through a weight matrix $\mathbf{W}$, the parameters of the model. In addition, each set of visible and hidden units have biases ($\mathbf{a}$ and $\mathbf{b}$ respectively, that are part of the set of parameters as well, for which the update rules are simpler) which are also learned. The joint probability of visible and hidden units is given by:

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z}\exp(-E(\mathbf{v}, \mathbf{h}))$$

where Z is the partition function and is computed by summing over all possible pairs of visible and hidden units:

$$Z = \sum_{\mathbf{v},\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}))$$

and $E(\mathbf{v}, \mathbf{h})$ is the energy of the specific configuration given by:

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{a}\mathbf{v} - \mathbf{b}\mathbf{h} - \mathbf{v}\mathbf{W}\mathbf{h}^T$$

The probability of a visible vector is given by marginalizing over the hidden units:

$$p(\mathbf{v}) = \frac{1}{Z}\sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}))$$

To maximize the probability of a given visible (input) vector, we need to modify the parameters (weights and biases) to lower the energy of that input. The gradient of the log probability of an input vector is given by:

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \mathbb{E}_{data}[v_i h_j] - \mathbb{E}_{model}[v_i h_j] \tag{1}$$

where the first expectation is taken with respect to the data distribution, while the second one is taken with respect to the model distribution. Then the change in weights is given by the stochastic gradient is:

$$\delta w_{ij} = \alpha(\mathbb{E}_{data}[v_i h_j] - \mathbb{E}_{model}[v_i h_j])$$

where α is the learning rate. To get a sample from $\mathbb{E}_{data}[v_i h_j]$, given an input sample $\mathbf{v}$, considering that there are no connection between the hidden units $\mathbf{h}$, the probability of a hidden unit j to take the value 1 is given by:

$$p(h_j = 1|\mathbf{v}) = \sigma(b_j + \sum_i v_i w_{ij}) \tag{2}$$

where the function σ is the sigmoid function $\sigma(x) = 1/(1 + \exp(-x))$. It can be shown that the sample we get this way $v_i h_j$ is an unbiased sample. The same procedure applies to the visible (binary) units, because the visible units don't have connections between them as well:

$$p(v_i = 1|\mathbf{h}) = \sigma(a_i + \sum_j h_j w_{ij}) \tag{3}$$

To get an unbiased sample of $\mathbb{E}_{model}[v_i h_j]$ is harder, we need to perform alternate Gibbs sampling starting from random states of the visible units and then using equations 2, 3 alternatively for many iterations. A practical procedure [Hinton, 2002] uses a training example for the start point and then performs just one iteration of the above equations. This works well in practice, even though this approximation is a rough approximation to the difference between two KL divergences, called Contrastive Divergence [Hinton, 2002]. More Gibbs steps improve performance with the additional computational cost. We are interested in computing the natural gradient instead of the stochastic gradient, for which we need the Fisher matrix. For a small RBM, (10-20 hidden units) the Fisher matrix can be computed by summing over all values of the hidden units $\mathbf{h}$, similar to the exact partition function computation performed in [Salakhutdinov and Murray, 2008]. Following [Grosse and Salakhudinov, 2015], we show how the exact Fisher matrix can be computed. Because of the dual parametrization for the exponential family (natural parameters and the expectation of sufficient statistics) we can compute the FIM by making use of the expectation of the sufficient statistics of the joint distribution of $(\mathbf{v}, \mathbf{h})$. Denoting by $\mathbf{g}$ the vector of sufficient statistics, given by: $(\mathbf{v}, \mathbf{h}, vec(\mathbf{v}\mathbf{h}^\mathbf{T}))$, the Fisher matrix is given by its covariance $\mathbf{G} = Cov(\mathbf{g}, \mathbf{g}) = \mathbb{E}[\mathbf{g}\mathbf{g}^\mathbf{T}] - \mathbb{E}[\mathbf{g}]\mathbb{E}[\mathbf{g}]^T$, with:

$$\mathbb{E}[\mathbf{g}] = \sum_{\mathbf{h}} p(\mathbf{h})\mathbb{E}[\mathbf{g}|\mathbf{h}]$$

$$\mathbb{E}[\mathbf{g}\mathbf{g}^T] = \sum_{\mathbf{h}} p(\mathbf{h})(\mathbb{E}[\mathbf{g}|\mathbf{h}]\mathbb{E}[\mathbf{g}|\mathbf{h}]^T + Cov(\mathbf{g}|\mathbf{h}))$$

where the conditional expectation is over $\mathbf{v}|\mathbf{h}$, and is given by:

$$\mathbb{E}[\mathbf{g}|\mathbf{h}] = \sum_{\mathbf{v}|\mathbf{h}} p(\mathbf{v}|\mathbf{h})\mathbf{g}|\mathbf{h}$$

where $p(\mathbf{v}|\mathbf{h}) = \prod_i (p(v_i|\mathbf{h}))$, and the individual factors of the product can be computed using Equation 3. The process

is as follows: assume we have 10 hidden units. We thus have 2^{10} hidden configurations. We first select one of these $\mathbf{h}$, then we get a sample of $\mathbf{v}$ and then we compute their product to get $\mathbf{vh^T}$. We then have a $\mathbf{g}|\mathbf{h}$ and can compute $\mathbb{E}[\mathbf{g}|\mathbf{h}]$ by summing over the 2^{10} visible configurations given $\mathbf{h}$. Similarly for summing over $\mathbf{h}$, with $p(\mathbf{h}|\mathbf{v}) = \prod_j (p(h_j|\mathbf{v}))$ using Equation 2. We show in Figure 1 how the various quantities in the computation of the inverse Fisher matrix look like.

3.1 Overview

One of the main issues with the decomposition of the covariance matrix of the network statistics is the computational overhead, for example for the eigendecomposition the cost is cubic in the layer size, while for the graphical model, the cost is cubic in the clique size, which is still quite expensive. Because obtaining the approximation to the FIM is so expensive, we need to mention here, that the update to the FIM is done once every T iterations, where T is much bigger than 1. The reason for this is that the approximation degrades slowly over time (as W changes slowly), so that the last one is still a reasonable approximation.

We ask the following question: can we find a cheap to update approximation to the FIM such that we can 1. update at every step and 2. still get a good fit to it?

4 Natural Boltzmann Machine

We believe that for deep networks the right approach for decorrelating the activations is an incremental one. Meaning, we make use of the iterative nature of stochastic gradient descent and add at each step an update for decorrelation. We will tackle this problem in two ways. First, we add a covariance penalty to the original cost function which should penalize large covariance values, but not adding any parameters to the network. And second, we add a set of parameters to the original network, which incrementally learns to decorrelate the network activations, we call these whitening parameters. The two functions to be optimized in this case are independent in the parameters to be learned, for each one we update a different set of parameters (W for the negative log probability and U, V for whitening). This approach is very similar with the previously mentioned techniques, specifically the natural neural networks, but with the notable difference that we do not perform the expensive singular value decomposition, but incrementally update a matrix (the inverse covariance). In Machine Learning, many techniques exist for iteratively updating a matrix, such that the output of a vector matrix multiplication is a set of independent vectors. In Independent Component Analysis such constraints are ubiquitous. We will choose the one from the seminal work of [Cardoso and Laheld, 1996] where a simple iterative rule was derived for the matrix of interest. We will proceed next to describe the first approach, where the objective function is a convex combinations of the two functions of interest, but both are optimized with respect to the same set of parameters, W, the canonical parameters of the network.

4.1 Covariance penalty

Even though the technique described next applies equally well to MLPs or ConvNets, we restrict our description (and implementation) to Restricted Boltzmann Machines. We propose to add additional terms to the cost function which are squared norms of covariances of different pairs of hidden and visible units. For RBMs the gradient of the original cost function with respect to the parameters is given in equation 1. We propose to add the squared (Frobenius) norms of covariances and then move in the direction of their gradient to minimize the squared norms. In theory one should minimize all pairwise covariance-norms present in $\|cov(g, g)\|$ but because some terms depend on others, minimizing the most important terms at first is reflected in the later covariances. Writing the Fisher matrix unfolded (effectively multiplying the individual components of $\mathbf{g}$ and then rewriting as covariances) will shed some light on what terms are most important:

$$G = \mathbb{E}\left[\begin{bmatrix} \mathbf{v} & \mathbf{h} & vec(\mathbf{v}\mathbf{h}^T) \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \mathbf{h} \\ vec(\mathbf{v}\mathbf{h}^T) \end{bmatrix} \right]$$
$$-\mathbb{E}\left[\begin{bmatrix} \mathbf{v} & \mathbf{h} & vec(\mathbf{v}\mathbf{h}^T) \end{bmatrix} \right] \mathbb{E}\left[\begin{bmatrix} \mathbf{v} \\ \mathbf{h} \\ vec(\mathbf{v}\mathbf{h}^T) \end{bmatrix} \right]$$

Then taking the squared norm of G gives:

$$\|G\|^2 = trace(G^T G) = trace(cov(\mathbf{v}, \mathbf{v})^2 + cov(\mathbf{h}, \mathbf{h})^2 + \\ + 2cov(\mathbf{v}, \mathbf{h})^2 + 2cov(\mathbf{v}, \mathbf{v}\mathbf{h}^T)^2 + 2cov(\mathbf{h}, \mathbf{v}\mathbf{h}^T)^2 \\ + cov(\mathbf{v}\mathbf{h}^T, \mathbf{v}\mathbf{h}^T)^2)$$

The last three terms are the biggest in size and they depend on the first three terms, thus it makes sense to first minimize the first three, which will then minimize the last three as well. We will show as an example, how to get the penalty term for $cov(\mathbf{v}, \mathbf{h})^2$. The calculations for the other terms are similar. As we want to minimize the negative log probability, we want to minimize this squared covariance-norm as well, thus the new cost function will be:

$$\mathcal{L} = (1 - \gamma)(-\log P(v)) + \gamma \|cov(\mathbf{v}, \mathbf{h})\|^2$$

where γ is a weight coefficient which will change over time. At first, when feeding the input data, we would like the network to become rapidly decorrelated, so we want a bigger value for γ, but as the covariance decreases, we want to decrease γ to minimize the negative log probability. This is an unsupervised approach and in some respects similar to pretraining, but comes to complement it by steering the network into the natural regime. Here we don't learn the structure of the data, but we modify the parameters to decorrelate the network activations as the input arrives, so it is an incremental whitening of the network activations. We are slowly moving the network towards the natural regime, where the covariance is the identity map. Even though we might not get it, the Euclidean gradient will get closer to the natural gradient at each step. We show next how to get the required terms. We note first that $cov(\mathbf{v}, \mathbf{h})$ is a matrix, so in effect we want to compute the partial derivatives of a function (which is the square) of a matrix (the covariance matrix), which is itself a function of the weight matrix $\mathbf{W}$ (the function is the covariance). From [Petersen *et al.*, 2008] we know that taking derivative

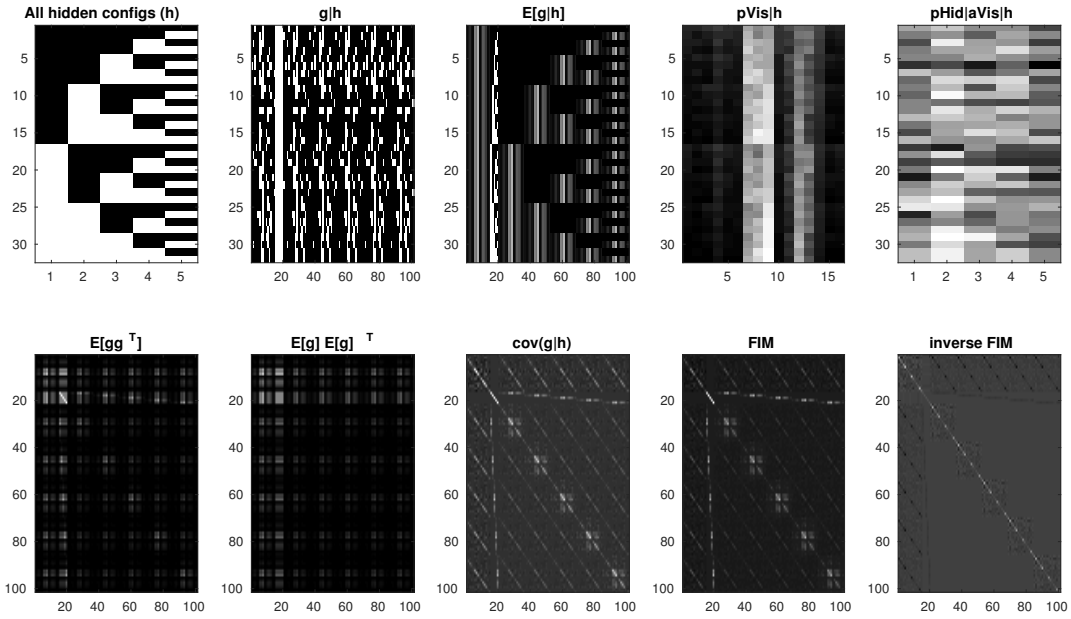


Figure 1: Quantities in the computation of the inverse Fisher matrix.

of a function of a matrix with respect to another matrix using the chain rule gives:

$$\frac{g(f(\mathbf{X}))}{\partial \mathbf{X}} = Tr\left[(\frac{\partial g(f(\mathbf{X}))}{\partial f(\mathbf{X})})^T \frac{\partial f(\mathbf{X})}{\partial \mathbf{X}}\right]$$

where $f(\mathbf{X})$ is the covariance matrix, g is the square function (see Section 5 for details of removing the norm) and $\mathbf{X}$ is the weight matrix $\mathbf{W}$. Replacing our quantities in the above formula, for each element of $\mathbf{W}$, w_{ij} gives:

$$\frac{\partial cov(\mathbf{v},\mathbf{h})^2}{\partial w_{ij}} = Tr\left[(\frac{\partial cov(\mathbf{v},\mathbf{h})^2}{\partial cov(\mathbf{v},\mathbf{h})})^T \frac{\partial cov(\mathbf{v},\mathbf{h})}{\partial w_{ij}}\right]$$

We can easily compute the two terms in this equation. The first term is simply: $2cov(\mathbf{v},\mathbf{h})$. The second term is given by:

$$\frac{\partial cov(\mathbf{v},\mathbf{h})}{\partial w_{ij}} = \frac{\partial \mathbb{E}[\mathbf{v}\mathbf{h}^T]}{\partial w_{ij}} - \frac{\partial \mathbb{E}[\mathbf{v}]\mathbb{E}[\mathbf{h}]^T}{\partial w_{ij}} - \frac{\mathbb{E}[\mathbf{v}]\partial \mathbb{E}[\mathbf{h}]^T}{\partial w_{ij}} \quad (4)$$

where all the expectations are with respect to the model distribution. We can readily compute the partial derivatives of the expectations with finite differences, given by:

$$\frac{\partial \mathbb{E}[\mathbf{v}\mathbf{h}^T]}{\partial w_{ij}} = \frac{\mathbb{E}[\mathbf{v}_{t+1}\mathbf{h}_{t+1}^T] - \mathbb{E}[\mathbf{v}_t\mathbf{h}_t^T]}{w_{ij}^{t+1} - w_{ij}^t}$$

where w_{ij} is the (i,j) entry of the weight matrix $\mathbf{W}$. We already have the required computations from the gradient updates. Each gradient update moves the parameters $\mathbf{W}$ with a small increment $d\mathbf{W}$, such that $\mathbf{W}_{t+1} = \mathbf{W}_t + d\mathbf{W}$ while the next forward propagation step gives us the required increment for the respective activations $\mathbf{v}_{t+1}\mathbf{h}_{t+1}^T$. We can get the other terms in Equation 4 similarly:

$$\frac{\partial \mathbb{E}[\mathbf{v}]\mathbb{E}[\mathbf{h}]^T}{\partial w_{ij}} = \frac{(\mathbb{E}[\mathbf{v}_{t+1}] - \mathbb{E}[\mathbf{v}_t])\mathbb{E}[\mathbf{h}_{t+1}]^T}{w_{ij}^{t+1} - w_{ij}^t}$$

$$\frac{\mathbb{E}[\mathbf{v}]\partial \mathbb{E}[\mathbf{h}]^T}{\partial w_{ij}} = \frac{\mathbb{E}[\mathbf{v}_{t+1}](\mathbb{E}[\mathbf{h}_{t+1}] - \mathbb{E}[\mathbf{h}_t])^T}{w_{ij}^{t+1} - w_{ij}^t}$$

5 Results and Discussion

For experiments we chose the ubiquitous MNIST dataset, the reduced version, since this is more efficient and if there is a significant difference between two training algorithms, this should show up clearly when dealing with this version as well. Also, the superior performance of one algorithm should easily be translated to the full version. The reduced MNIST consists of 10000 samples for training and 2000 samples for testing, but keeping the same resolution as the original images, i.e. 28×28. When implementing the covariance penalty algorithm, we need to consider some practical issues.

The first issue is the fact the the actual covariance is quite expensive to compute (squared in the dimensionality and linear in the number of samples) and the trace is actually not too informative, because it is just one real number. So we decided (after careful experimentation) to leave out the actual covariance from the penalty (this being equivalent to removing the square from the covariance penalty term) and remove the trace operator altogether since the remaining terms have a direct correspondence to the weight matrix. The product of the derivative of the visibles and the hiddens is the size of the matrix and they are directly related. Another way of looking at it is saying we don't actually have a real valued function of

the weight matrix, which we want to derivate, but a matrix-valued function.

Another practical issue is: if the difference in the weights from on step to the next is too small, then the division by such a small number in Equation 4 is unstable and renders our algorithm much less effective. We tried setting a fixed (not too small) number for the weight differences which are too small, but this seems quite arbitrary and does not help too much either.

After analysing the actual data we decided to set all dW to 1, thus considering the changes in dW as discrete steps, just like time increments. When considering the derivative of the expectations, by looking just at one step, we get too much noise in the penalty, thus we decided to use a moving average of the last 15 steps (this was chosen by trial and error). We tried giving γ binary values with a specific schedule, in the beginning γ was 1 and as the covariance decreases, γ switches to 0, but this did not seem to work. We then tried to give γ real values between 0 and 1, with big values at first and smaller values later. This seemed to work a little better, but the actual schedule and covariance penalty magnitude seemed critical. Thus we decided to accept a small penalty (in magnitude), so quite far from having an identity covariance matrix, but still penalize large values for the covariance. So then we needed to choose how to (efficiently) measure this magnitude. Computing the covariance would be the easiest and the most informative but it is expensive. Thus we settled on having a certain percentage of weight penalties under a certain threshold. So in effect, the penalty we are trying to use is just a proxy of the true covariance. More specifically, it is a moving average derivative of sample covariances. The values we used were taken from the sets: $(0.0001, 0.001, 0.01, 0.1, 0.2)$ for the threshold value (v) and $(0.001, 0.01, 0.05, 0.1, 0.3, 0.5, 0.8)$ for the percentages (p). So if p percent of the values of the penalty matrix are bigger than v then increase the weight of the penalty, otherwise decrease it. The factor for the penalty was trimmed between 1 and 0.00001. This was just to try to figure out if this technique is promising or not, and if it is, we would then hopefully be able to eliminate the need for these hyper-parameters. We show in Figure 2 the results. To figure out if this has any potential we used a small network (10 hiddens) and a highly subsampled version of the MNIST dataset. The best result with SGD was at around 55 percent. We used 100 epochs in all experiments and took the minimum test accuracy value of these epochs. In some of the experiments we noticed two interesting types of behaviour. The first is that the network seems to generalize better (the test accuracy is almost always significantly lower than the train accuracy) and the second is that for certain values of the hyper-parameters we seem to be getting a sparse solution. We show this in Figure 3. We actually get a sparse weight matrix W, with the existing filters (columns of neurons that differ by a small value act like filters, or receptive fields) being more complex than in the usual SGD solution. This makes sense in a way, the network increases the information given by any given filter and decorrelation comes out of having a small number of more complex filters.

6 Conclusions and future work

Further investigations need to be carried out in order to draw some clear conclusions about this technique. If it does not work, we should be able to state the reasons for this. A natural follow up on this technique, is, as we mentioned augmenting the model with another set of parameters and then adjust those to minimize covariance. Incremental techniques are possible for decorrelating signals. We showed one usually used in the signal processing community, but many more exist. Other approaches for efficiently computing the covariance could be using random projections, where we cheaply project to a lower dimensional space, but we lose some amount of information (we are actually guaranteed to lose a finite amount of information with certain types of projection matrices, independent of the structure present in the data. This is based on the seminal work described in [Dasgupta and Gupta, 1999]). In this case we trade accuracy for efficiency. Other methods for efficiently computing covariances exist, which have a complexity lower than squared, for example [Kwatra and Han, 2010]. Future work includes dealing with other types of networks like convolutional or just normal MLPs. It is certain that for deep networks, computing the covariance and singular value decomposition is prohibitive, thus new methods that make use of the iterative nature of deep training need to be considered. Any type of covariance minimization technique, should, in theory, take the stochastic gradient closer to the natural gradient. This can include any type of decorrelation of the network activations, even for example, special inhibitory connections, or Hebbian like rules.

References

[Amari, 1998] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

[Cardoso and Laheld, 1996] Jean-François Cardoso and Beate Hvam Laheld. Equivariant adaptive source separation. *Signal Processing, IEEE Transactions on*, 44(12):3017–3030, 1996.

[Cencov, 2000] Nikolai Nikolaevich Cencov. *Statistical decision rules and optimal inference*. American Mathematical Soc., 2000.

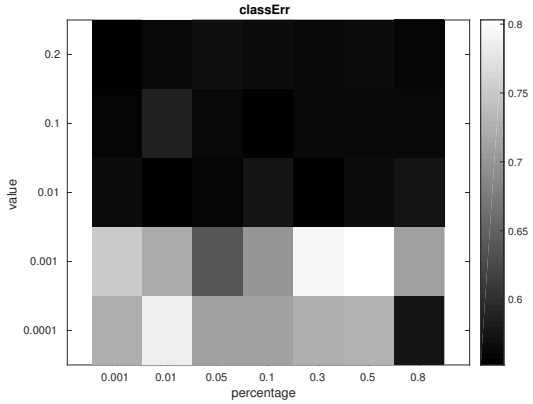
[Dasgupta and Gupta, 1999] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of the johnson-lindenstrauss lemma. *International Computer Science Institute, Technical Report*, pages 99–006, 1999.

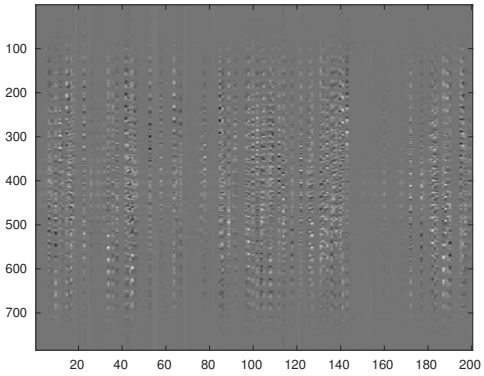
[Desjardins *et al.*, 2015] Guillaume Desjardins, Karen Simonyan, Razvan Pascanu, et al. Natural neural networks. In *Advances in Neural Information Processing Systems*, pages 2062–2070, 2015.

[Grosse and Salakhudinov, 2015] Roger Grosse and Ruslan Salakhudinov. Scaling up natural gradient by sparsely factorizing the inverse fisher matrix. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2304–2313, 2015.

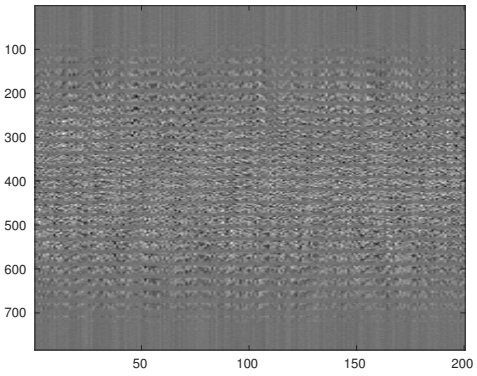
[Hinton, 2002] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

Figure 2: Testing different hyper-parameters.





(a) Weight matrix trained with covariance penalty.



(b) Weight matrix trained with SGD.

Figure 3: Comparison between SGD and the covariance penalty technique(CP).

[Kwatra and Han, 2010] Vivek Kwatra and Mei Han. Fast covariance computation and dimensionality reduction for sub-window features in images. In *Computer Vision–ECCV 2010*, pages 156–169. Springer, 2010.

[Petersen *et al.*, 2008] Kaare Brandt Petersen, Michael Syskind Pedersen, et al. The matrix cookbook. *Technical University of Denmark*, 7:15, 2008.

[Potter, 1963] JE Potter. New statistical formulas,". *Space Guidance Analysis Memo*, 40:1309–1314, 1963.

[Rao, 1945] C Radhakrishna Rao. Information and accuracy attainable in the estimation of statistical parameters. *Bull Calcutta. Math. Soc.*, 37:81–91, 1945.

[Salakhutdinov and Murray, 2008] Ruslan Salakhutdinov and Iain Murray. On the quantitative analysis of deep belief networks. In *Proceedings of the 25th international conference on Machine learning*, pages 872–879. ACM, 2008.

[Sohl-Dickstein, 2012] Jascha Sohl-Dickstein. The natural gradient by analogy to signal whitening, and recipes and tricks for its use. *arXiv preprint arXiv:1205.1828*, 2012.

7 Appendix A

We show next how we get from the square norm of G to the covariance between $\mathbf{v}$ and $\mathbf{h}$, through basic algebraic manipulations and grouping covariances from expectations. We denote by:

$$\mathbf{A} = \mathbb{E}\left[\begin{bmatrix} \mathbf{v} & \mathbf{h} & (\mathbf{v}\mathbf{h}^T) \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \mathbf{h} \\ (\mathbf{v}\mathbf{h}^T) \end{bmatrix} \right]$$

$$= \mathbb{E}\left[\begin{bmatrix} \mathbf{v}\mathbf{v} & \mathbf{v}\mathbf{h} & \mathbf{v}(\mathbf{v}\mathbf{h}^T) \\ \mathbf{h}\mathbf{v} & \mathbf{h}\mathbf{h} & \mathbf{h}(\mathbf{v}\mathbf{h}^T) \\ (\mathbf{v}\mathbf{h}^T)\mathbf{v} & (\mathbf{v}\mathbf{h}^T)\mathbf{h} & (\mathbf{v}\mathbf{h}^T)(\mathbf{v}\mathbf{h}^T) \end{bmatrix} \right]$$

$$\mathbf{B} = \mathbb{E}\left[\begin{bmatrix} \mathbf{v} & \mathbf{h} & (\mathbf{v}\mathbf{h}^T) \end{bmatrix} \right] \mathbb{E}\left[\begin{bmatrix} \mathbf{v} \\ \mathbf{h} \\ (\mathbf{v}\mathbf{h}^T) \end{bmatrix} \right] =$$

$$\begin{bmatrix} \mathbb{E}[\mathbf{v}]\mathbb{E}[\mathbf{v}] & \mathbb{E}[\mathbf{v}]\mathbb{E}[\mathbf{h}] & \mathbb{E}[\mathbf{v}]\mathbb{E}[(\mathbf{v}\mathbf{h}^T)] \\ \mathbb{E}[\mathbf{h}]\mathbb{E}[\mathbf{v}] & \mathbb{E}[\mathbf{h}]\mathbb{E}[\mathbf{h}] & \mathbb{E}[\mathbf{h}]\mathbb{E}[(\mathbf{v}\mathbf{h}^T)] \\ \mathbb{E}[(\mathbf{v}\mathbf{h}^T)]\mathbb{E}[\mathbf{v}] & \mathbb{E}[(\mathbf{v}\mathbf{h}^T)]\mathbb{E}[\mathbf{h}] & \mathbb{E}[(\mathbf{v}\mathbf{h}^T)]\mathbb{E}[(\mathbf{v}\mathbf{h}^T)] \end{bmatrix}$$

So $G = A - B$, thus $tr(G^T G) = tr((A-B)^T(A-B))$, and since $tr((A-B)^T) = tr(A^T) - tr(B^T)$ we then get $tr(G^T G) = tr(A^T A) - tr(B^T A) - tr(A^T B) + tr(B^T B)$. We now calculate each term, but since we are interested only in the trace, we will show just the diagonal blocks, with (i) indicating the index of the block on the diagonal.

$$\mathbf{A}^T\mathbf{A}_{(1)} = \left[\mathbb{E}[\mathbf{v}^T\mathbf{g}]\mathbb{E}[\mathbf{g}^T\mathbf{v}]\right]$$

$$\mathbf{A}^T\mathbf{A}_{(2)} = \left[\mathbb{E}[\mathbf{h}^T\mathbf{g}]\mathbb{E}[\mathbf{g}^T\mathbf{h}]\right]$$

$$\mathbf{A}^T\mathbf{A}_{(3)} = \left[\mathbb{E}[\mathbf{v}^\mathbf{T}\mathbf{h}\mathbf{g}]\mathbb{E}[\mathbf{g}^T\mathbf{v}\mathbf{h}^T]\right]$$

$$\mathbf{A}^T\mathbf{B}_{(1)} = \left[\mathbb{E}[\mathbf{v}^T\mathbf{g}]\mathbb{E}[\mathbf{g}^T]\mathbb{E}[\mathbf{v}]\right]$$

$$\mathbf{A}^T\mathbf{B}_{(2)} = \left[\mathbb{E}[\mathbf{h}^T\mathbf{g}]\mathbb{E}[\mathbf{g}^T]\mathbb{E}[\mathbf{h}]\right]$$

$$\mathbf{A}^T\mathbf{B}_{(3)} = \left[\mathbb{E}[\mathbf{v}^\mathbf{T}\mathbf{h}\mathbf{g}]\mathbb{E}[\mathbf{g}^T]\mathbb{E}[\mathbf{v}\mathbf{h}^T]\right]$$

We form the other terms analogously. When dealing with block matrices, the sum of traces of the individual blocks equals the trace of the whole matrix. We can write the sum the traces for each of the three blocks, for example for the first block we have:

$$tr(\mathbf{A}^T\mathbf{A}_{(1)} - \mathbf{A}^T\mathbf{B}_{(1)} - \mathbf{B}^T\mathbf{A}_{(1)} + \mathbf{B}^T\mathbf{B}_{(1)}) =$$

$$= tr(\mathbb{E}[\mathbf{v}^T\mathbf{g}]\mathbb{E}[\mathbf{g}^T\mathbf{v}] - \mathbb{E}[\mathbf{v}^T\mathbf{g}]\mathbb{E}[\mathbf{g}^T]\mathbb{E}[\mathbf{v}] - \mathbb{E}[\mathbf{v}^T]\mathbb{E}[\mathbf{g}]\mathbb{E}[\mathbf{g}^T\mathbf{v}] + \mathbb{E}[\mathbf{v}^T]\mathbb{E}[\mathbf{g}]\mathbb{E}[\mathbf{g}^T]\mathbb{E}[\mathbf{v}])$$

Grouping the first and third terms and the second and fourth, we get:

$$= tr((\mathbb{E}[\mathbf{v}^T\mathbf{g}] - \mathbb{E}[\mathbf{v}^T]\mathbb{E}[\mathbf{g}])\mathbb{E}[\mathbf{g}^T\mathbf{v}] - (\mathbb{E}[\mathbf{v}^T\mathbf{g}] - \mathbb{E}[\mathbf{v}^T]\mathbb{E}[\mathbf{g}])\mathbb{E}[\mathbf{g}^T]\mathbb{E}[\mathbf{v}]$$

Grouping the covariances we then get:

$$= tr(cov(\mathbf{v}^T, \mathbf{g})\mathbb{E}[\mathbf{g}^T\mathbf{v}] - cov(\mathbf{v}^T, \mathbf{g})\mathbb{E}[\mathbf{g}^T]\mathbb{E}[\mathbf{v}] =$$
$$= tr(cov(\mathbf{v}^T, \mathbf{g})cov(\mathbf{g}^T, \mathbf{v}))$$

We see that this product of covariances is constituted of sums of individual product of covariances ($cov(\mathbf{v}^T, \mathbf{v})$, $cov(\mathbf{v}^T, \mathbf{h})$, $cov(\mathbf{v}^T, \mathbf{v}\mathbf{h}^T)$). Thus, as an example, we consider the case of $cov(\mathbf{v}, \mathbf{h})$. And since the trace is a linear operator, it commutes with the derivative operator:

$$\frac{\partial tr(A)}{\partial w} = tr(\frac{\partial A}{\partial w})$$