Revisiting natural gradient for deep networks

Razvan Pascanu
Université de Montréal
Montréal QC H3C 3J7 Canada
r.pascanu@gmail.com

Yoshua Bengio
Université de Montréal
Montréal QC H3C 3J7 Canada
yoshua.bengio@umontreal.ca

Abstract

We evaluate natural gradient descent, an algorithm originally proposed in Amari (1997), for learning deep models. The contributions of this paper are as follows. We show the connection between natural gradient descent and three other recently proposed methods for training deep models: Hessian-Free Optimization (Martens, 2010), Krylov Subspace Descent (Vinyals and Povey, 2012) and TONGA (Le Roux *et al.*, 2008). We describe how one can use unlabeled data to improve the generalization error obtained by natural gradient and empirically evaluate the robustness of the algorithm to the ordering of the training set compared to stochastic gradient descent. Finally we extend natural gradient descent to incorporate second order information alongside the manifold information and provide a benchmark of the new algorithm using a truncated Newton approach for inverting the metric matrix instead of using a diagonal approximation of it.

1 Introduction

Several recent papers tried to address the issue of using better optimization techniques for machine learning, especially for training deep architectures or neural networks of various kinds. Hessian-Free Optimization (Martens, 2010; Martens and Sutskever, 2011; Chapelle and Erhan, 2011), Krylov Subspace Descent (Vinyals and Povey, 2012; Mizutani and Demmel, 2003), natural gradient descent (Amari, 1997; Park *et al.*, 2000), TONGA (Le Roux *et al.*, 2008; Roux and Fitzgibbon, 2010) are just a few of such recently proposed algorithms. They usually can be split in different categories: those which make use of second order information, those which use the geometry of the underlying parameter manifold (e.g. natural gradient descent) or those that use the uncertainty in the gradient (e.g. TONGA).

One particularly interesting pipeline to scale up such algorithms was originally proposed in Pearlmutter (1994) – finetuned in Schraudolph (2002) – and represents the backbone behind both Hessian-Free Optimization (Martens, 2010) and Krylov Subspace Descent (Vinyals and Povey, 2012; Mizutani and Demmel, 2003). The core idea behind it is to make use of the forward pass (renamed to *R-operator* in Pearlmutter (1994)) and backward pass of automatic differentiation to compute efficient products between Jacobian or Hessian matrices and vectors. These products are used within a truncated-Newton approach (Nocedal and Wright, 2000) which considers the exact Hessian and only inverts it approximately without the need for explicitly storing the matrix in memory, as opposed to other approaches which perform a more crude approximation of the Hessian (or Fisher) matrix (either diagonal or block-diagonal).

The original contributions of this paper to the study of natural gradient descent are as follows. Section 5 describes the connection between natural gradient and Hessian Free (Martens, 2010), section 6 looks at the relationship with Krylov Subspace Descent (KSD) (Vinyals and Povey, 2012). Section 7 describes how unlabeled data can be incorporated into natural gradient descent in order to improve generalization error. Section 8 explores empirically natural gradient descent's robustness to permutation of the training set. In section 9 we extend natural gradient descent to incorporate second order information. Finally in section 10 we provide a benchmark of the algorithm discussed,

where natural gradient descent is implemented using a truncated Newton approach for inverting the full metric matrix instead of the traditional diagonal or band-diagonal approximation.

2 Natural gradient descent

Natural gradient descent can be traced back to Amari's work on information geometry (Amari, 1985) and its application to various neural networks (Amari *et al.*, 1992; Amari, 1997), though a more in depth introduction can be found in Amari (1998); Park *et al.* (2000); Arnold *et al.* (2011). The algorithm has also been successfully applied in the reinforcement learning community (Kakade, 2001; Peters and Schaal, 2008) and for stochastic search (Sun *et al.*, 2009).

Let us consider a family of density functions $\mathcal{F}$ that maps a parameter $\theta \in \mathbb{R}^P$ to a probability density function $p(\mathbf{z})$, $p : \mathbb{R}^N \to [0, \infty)$, where $\mathbf{z} \in \mathbb{R}^N$. Any choice of $\theta \in \mathbb{R}^P$ defines a particular density function $p_\theta(\mathbf{z}) = \mathcal{F}(\theta)(\mathbf{z})$ and by considering all possible θ values, we explore the set $\mathcal{F}$, which is our functional manifold.

In its infinitesimal form, the KL-divergence behaves like a distance measure, so we can define a similarity measure between nearby density functions. Hence $\mathcal{F}$ is a Riemannian manifold whose metric is given by the Fisher Information matrix $\mathbf{F}_\theta$ defined in the equation below:

$$\mathbf{F}_\theta = \mathbb{E}_{\mathbf{z}} \left[\left(\nabla \log p_\theta(\mathbf{z}) \right)^T \left(\nabla \log p_\theta(\mathbf{z}) \right) \right]. \tag{1}$$

That is, locally around some point θ, the metric defines an inner product between vectors $\mathbf{u}$ and $\mathbf{v}$:

$$< \mathbf{u}, \mathbf{v} >_\theta = \mathbf{u} \mathbf{F}_\theta \mathbf{v},$$

and it hence provides a local measure of distance. It what follows we will write $\mathbf{F}$ for the Fisher Information Matrix, assuming an implicit dependency of this matrix on θ.

Given a loss function $\mathcal{L}$ parametrized by θ, natural gradient descent attempts to move along the manifold by correcting the gradient of $\mathcal{L}$ according to the local curvature of the KL-divergence surface, i.e. moving some given distance in the direction $\nabla_N \mathcal{L}(\theta)$:

$$\nabla_N \mathcal{L}(\theta) \stackrel{\text{def}}{=} \nabla \mathcal{L}(\theta) \mathbb{E}_{\mathbf{z}} \left[\left(\nabla \log p_\theta(\mathbf{z}) \right)^T \left(\nabla \log p_\theta(\mathbf{z}) \right) \right]^{-1}$$
$$\stackrel{\text{def}}{=} \nabla \mathcal{L}(\theta) \mathbf{F}^{-1}. \tag{2}$$

We use ∇_N for natural gradient, ∇ for gradients and $\mathbf{F}$ is the metric matrix given by the Fisher Information Matrix. Partial derivatives are usually denoted as row vectors in this work. We can derive this result by considering natural gradient descent to be defined as the algorithm which, at each step, attempts to pick a descent direction such that the amount of change (in the KL-sense) induced in our model is some given value. Specifically we look for a small $\Delta\theta$ that minimizes a first order Taylor expansion of $\mathcal{L}$ when the second order Taylor series of the KL-divergence between p_θ and $p_{\theta+\Delta\theta}$ has to be constant:

$$\arg \min_{\Delta\theta} \mathcal{L}(\theta + \Delta\theta)$$
$$\text{s. t. } KL(p_\theta \| p_{\theta+\Delta\theta}) = const. \tag{3}$$

Using this constraint we ensure that we move along the functional manifold with constant speed, without being slowed down by its curvature. This also makes learning locally *robust to re-parametrizations of the model*, as the functional behaviour of p does not depend on how it is parametrized.

Assuming $\Delta\theta \to 0$, we can approximate the KL divergence by its second order Taylor series:

$$KL(p_\theta \| p_{\theta+\Delta\theta}) \approx \left(\mathbb{E}_{\mathbf{z}} \left[\log p_\theta \right] - \mathbb{E}_{\mathbf{z}} \left[\log p_\theta \right] \right)$$
$$- \mathbb{E}_{\mathbf{z}} \left[\nabla \log p_\theta(\mathbf{z}) \right] \Delta\theta - \frac{1}{2} \Delta\theta^T \mathbb{E}_{\mathbf{z}} \left[\nabla^2 \log p_\theta \right] \Delta\theta$$
$$= \frac{1}{2} \Delta\theta^T \mathbb{E}_{\mathbf{z}} \left[-\nabla^2 \log p_\theta(\mathbf{z}) \right] \Delta\theta$$
$$= \frac{1}{2} \Delta\theta^T \mathbf{F} \Delta\theta \quad \text{1/2 Jacobian\_T @ F @ Jacobian} \tag{4}$$

2

The first term cancels out and because $\mathbb{E}_{\mathbf{z}}\left[\nabla \log p_\theta(\mathbf{z})\right] = 0^1$, we are left with only the last term. The Fisher Information Matrix form can be obtained from the expected value of the Hessian through algebraic manipulations (see the appendix).

We now express equation (3) as a Lagrangian, where the KL divergence is approximated by (4) and $\mathcal{L}(\theta + \Delta\theta)$ by its first order Taylor series $\mathcal{L}(\theta) + \nabla\mathcal{L}(\theta)\Delta\theta$:

∇ = derivative
Δ = change

$$\mathcal{L}(\theta) + \nabla\mathcal{L}(\theta)\Delta\theta + \frac{1}{2}\lambda\Delta\theta^T\mathbf{F}\Delta\theta \tag{5}$$

Solving equation (5) for $\Delta\theta$ gives us the natural gradient decent formula (2). Note that we get a scalar factor of $2\frac{1}{\lambda}$ times the natural gradient. We fold this scalar into the learning rate, which now also controls the weight we put on preserving the KL-distance between p_θ and $p_{\theta+\Delta\theta}$. The approximations we use are meaningful only around θ: in Schaul (2012) it is shown that taking large steps might harm convergence. We deal with such issues both by using damping (i.e. setting a trust region around θ) and by properly selecting a learning rate.

2.1 Adapting natural gradient descent for neural networks

In order to use natural gradient descent for deterministic neural networks we rely on their probabilistic interpretation (see Bishop (2006), chapter 3.1.1 for linear regression and 4.2 for classification). For example, the output of an MLP with linear activation function can be interpreted as the mean of a conditional Gaussian distribution with a fixed variance, where we condition on the input. Minimizing the squared error, under this assumption, is equivalent to maximum likelihood.

For classification, depending on the activation function, we can define the output units as being success probability of each trial or event probability of a binomial or multinoulli distribution conditioned on the input $\mathbf{x}$.

By an abuse of notation[2], we will use $p_\theta(\mathbf{t}|\mathbf{x})$ to define this conditional probability density function described above. Because it is a conditional probability function, the formulation of natural gradient descent, equation (3), changes into the following equation:

$$\arg\min_{\Delta\theta}\mathcal{L}(\theta + \Delta\theta)$$
$$\text{s. t. } \mathbb{E}_{\mathbf{x}\sim\tilde{q}(\mathbf{x})}\left[KL(p_\theta(\mathbf{t}|\mathbf{x})||p_{\theta+\Delta\theta}(\mathbf{t}|\mathbf{x}))\right] = const. \tag{6}$$

Each value of $\mathbf{x}$ now defines a different family of density functions $p_\theta(\mathbf{t}|\mathbf{x})$, and hence a different manifold. In order to measure the functional behaviour of $p_\theta(\mathbf{t}|\mathbf{x})$ for different values of $\mathbf{x}$, we use the expected value (over x) of the KL-divergence between $p_\theta(\mathbf{t}|\mathbf{x})$ and $p_{\theta+\Delta\theta}(\mathbf{t}|\mathbf{x})$.

In defining the constraint of equation 6, we have chosen to allow ourselves the freedom to compute the expectation over $\mathbf{x}$ using some distribution $\tilde{q}$ instead of the empirical distribution q. Usually we want $\tilde{q}$ to be q, though one can imagine situations when this would not be true. E.g. when we want our model to look more carefully at certain types of inputs, which we can do by biasing $\tilde{q}$ towards that type of inputs.

Applying the same steps as in section 2 we can recover the formula for natural gradient descent. This formula can be massaged further (similar to Park *et al.* (2000)) for specific activations and error functions. We exclude these derivations due to space constraints (they are provided in the appendix). For convenience we show the formulas for two typical pairs of output activations and corresponding error function, where $\mathbf{J_y}$ stands for the Jacobian matrix $\frac{\partial \mathbf{y}}{\partial \theta}$, $\mathbf{y}$ being the output layer. By an abuse of notation, $\frac{1}{\mathbf{y}}$ indicates the element-wise division of 1 by $\mathbf{y}_i$.

$$\mathbf{F}_{linear} = \beta^2\mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\frac{\partial \mathbf{y}}{\partial \theta}^T\frac{\partial \mathbf{y}}{\partial \theta}\right] = \beta^2\mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\mathbf{J_y}^T\mathbf{J_y}\right] \tag{7}$$

$$\mathbf{F}_{sigmoid} = \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\mathbf{J_y}^T\text{diag}\left(\frac{1}{\mathbf{y}(1-\mathbf{y})}\right)\mathbf{J_y}\right] \tag{8}$$

---

[1]Proof: $\mathbb{E}_{\mathbf{z}}\left[\nabla \log p_\theta(\mathbf{z})\right] = \sum_{\mathbf{z}}\left(p_\theta(\mathbf{z})\frac{1}{p_\theta(\mathbf{z})}\frac{\partial p_\theta(\mathbf{z})}{\partial\theta}\right) = \frac{\partial}{\partial\theta}\left(\sum_\theta p_\theta(\mathbf{z})\right) = \frac{\partial 1}{\partial\theta} = 0$. The proof holds for the continuous case as well, replacing sums for integrals.

[2]E.g., for softmax output layer the random variable sampled from the multinoulli is a scalar not a vector

3 Properties of natural gradient descent

In this section we summarize a few properties of natural gradient descent that have been discussed in the literature.

Natural gradient descent can be used in the online regime. We assume that even though we only have a single example available at each time step one also has access to a sufficiently large set of held out examples. If we are to apply a second order method in this regime, computing the gradient on the single example and the Hessian on the held out set would be conceptually problematic. The gradient and Hessian will be incompatible as they do not refer to the same function. However, for natural gradient descent, the metric comes from evaluating an independent expectation that is not related to the prediction error. It measures an intrinsic property of the model. It is therefore easier to motivate using a held-out set (which can even be unlabeled data as discussed in section 7).

Desjardins *et al.* (2013) shows a straight forward application of natural gradient to Deep Boltzmann Machines. It is not obvious how the same can be done for a standard second order method. Probabilistic models like RBMs and DBMs not have direct access to the cost they are minimizing, something natural gradient does not require.

Natural gradient descent is robust to local re-parametrization of the model. This comes from the constraint that we use. The KL-divergence is a measure of how the probability density function changes, regardless on how it was parametrized. Sohl-Dickstein (2012) explores this idea, defining natural gradient descent as doing whitening in the parameter space.

The metric $\mathbf{F}$ has two different forms as can be seen in equation (4). Note however that while the metric can be seen as both a Hessian or as a covariance matrix, *it is not the Hessian of the cost, nor the covariance of the gradients we follow to a local minimum*. The gradients are of p_θ which acts as a proxy for the cost $\mathcal{L}$. The KL-surface considered at any point $\mathbf{p}$ during training always has its minimum at $\mathbf{p}_\theta$, and the metric we obtain is always positive semi-definite by construction which is not true for the Hessian of $\mathcal{L}$.

Because $\mathbf{F}$ can be interpreted as an expected Hessian, it measures how much a change in θ will affect the gradients of $\mathbb{E}_{\mathbf{t} \sim p(\mathbf{t}|\mathbf{x})} [\log p_\theta(\mathbf{t}|\mathbf{x})]$. This means that, in the same way second order methods do, *natural gradient descent can jump over plateaus* of $p_\theta(\mathbf{t}|\mathbf{x})$. Given the usual form of the loss function $\mathcal{L}$, which is just an evaluation of p for certain pairs $(\mathbf{x}, \mathbf{t})$, plateaus of p usually match those of $\mathcal{L}$ and hence the method can jump over plateaus of the error function.

If we look at the KL constraint that we enforce at each time step, it does not only ensure that we induce at least some epsilon change in the model, but also that the model does not change by more than epsilon. We argue that this provides some kind of robustness to overfitting. The model is not allowed to move *too far* in some direction $\mathbf{d}$ if moving along $\mathbf{d}$ changes the density computed by the model substantially.

4 Natural gradient descent and TONGA

In Le Roux *et al.* (2008) one assumes that the gradients computed over different minibatches are distributed according to a Gaussian centered around the true gradient with some covariance matrix $\mathbf{C}$. By using the uncertainty provided by $\mathbf{C}$ we can correct the step that we are taking to maximize the probability of a downward move in generalization error (expected negative log-likelihood), resulting in a formula similar to that of natural gradient descent.

While the probabilistic derivation requires the centered covariance, in Le Roux *et al.* (2008) it is argued that one can use the uncentered covariance $\mathbf{U}$ resulting in a simplified formula which is sometimes confused with the metric derived by Amari:

$$\mathbf{U} \approx \mathbb{E}_{(\mathbf{x},\mathbf{t}) \sim q} \left[\left(\frac{\partial \log p(\mathbf{t}|\mathbf{x})}{\partial \theta} \right)^T \left(\frac{\partial \log p(\mathbf{t}|\mathbf{x})}{\partial \theta} \right) \right] \tag{9}$$

The discrepancy comes from the fact that the equation is an expectation, though the expectation is *over the empirical distribution $q(\mathbf{x}, \mathbf{t})$ as opposed to $\mathbf{x} \sim q(\mathbf{x})$ and $\mathbf{t} \sim p(\mathbf{t}|\mathbf{x})$*. It is therefore not clear if $\mathbf{U}$ tells us how p_θ would change, whereas it is clear that Amari's metric does.

5 Natural gradient descent and Hessian-Free Optimization

Hessian-Free Optimization as well as Krylov Subspace Descent[3] rely on the *extended Gauss-Newton approximation of the Hessian*, $\mathbf{G_N}$, instead of the actual Hessian (see Schraudolph (2002)):

$$\mathbf{G_N} = \frac{1}{n} \sum_{i=1}^{n} \left[\left(\frac{\partial \mathbf{r}}{\partial \theta} \right)^T \frac{\partial^2 \log p(\mathbf{t}^{(i)}|\mathbf{x}^{(i)})}{\partial \mathbf{r}^2} \left(\frac{\partial \mathbf{r}}{\partial \theta} \right) \right]$$
$$= \mathbb{E}_{\mathbf{x} \sim \tilde{q}} \left[\mathbf{J_r}^T \left(\mathbb{E}_{\mathbf{t} \sim \tilde{q}(\mathbf{t}|\mathbf{x})} [\mathbf{H}_{\mathcal{L} \circ \mathbf{r}}] \right) \mathbf{J_r} \right] \tag{10}$$

The last step of equation (10) assumes that $(\mathbf{x}^{(i)}, \mathbf{t}^{(i)})$ are i.i.d samples, and $\tilde{q}$ stands for the distribution represented by the minibatch over which the matrix is computed. J stands for Jacobian and H for Hessian. The subscript describes for which variable the quantity is computed. A composition in the subscript, as in $\mathbf{H}_{\mathcal{L} \circ \mathbf{r}}$, implies computing the Hessian of $\mathcal{L}$ with respect to $\mathbf{r}$, with r being the output layer before applying the activation function.

The reason for choosing this approximation over the Hessian is not computational, as computing both can be done equally fast. The extended Gauss-Newton approximation is better behaved during learning. This is assumed to hold because $\mathbf{G_N}$ is positive semi-definite by construction, so one needs not worry about negative curvature issues.

It is known that the Gauss-Newton approximation (for linear activation function and square error) matches the Fisher Information matrix. In this section we show that *this identity holds also for other matching pairs like sigmoid and cross-entropy or softmax and negative log-likelihood for which the extended Gauss-Newton is defined*. By choosing this specific approximation, one can therefore view both *Hessian-Free Optimization and KSD as being implementations of natural gradient descent*. We make the additional note that Heskes (2000) makes similar algebraic manipulations as the ones provided in this section, however for different reasons, namely to provide a new justification of the algorithm that relies on distance measures. The original contribution of this section is in describing the relationship between Hessian-Free Optimization and Krylov Subspace Descent on one side and natural gradient descent on the other. This relation was not acknowledged anywhere in the literature as far as we are aware of. While Heskes (2000) precedes both Schraudolph (2002); Martens (2010) both Hessian Free and Krylov Subspace Descent are introduced as purely approximations to second order methods.

In the case of sigmoid units with cross-entropy objective (σ is the sigmoid function), $\mathbf{H}_{\mathcal{L} \circ \mathbf{r}}$ is

$$\begin{aligned} \mathbf{H}_{\mathcal{L} \circ \mathbf{r}_{ij, i \neq j}} &= \frac{\partial^2 \sum_k (-t_k \log(\sigma(r_k)) - (1 - t_k) \log(1 - \sigma(r_k)))}{\partial r_i \partial r_j} \\ &= \frac{\partial \sigma(r_i) - t_i}{\partial r_j} = 0 \\ \mathbf{H}_{\mathcal{L} \circ \mathbf{r}_{ii}} &= ... = \frac{\partial \sigma(r_i) - t_i}{\partial r_i} = \sigma(r_i)(1 - \sigma(r_i)) \end{aligned} \tag{11}$$

If we insert this back into the Gauss-Newton approximation of the Hessian and re-write the equation in terms of $\mathbf{J_y}$ instead of $\mathbf{J_r}$, we get the corresponding natural gradient metric, equation (8). $diag(\mathbf{v})$ stands for the diagonal matrix constructed from the values of the vector $\mathbf{v}$ and we make an abuse of notation, where by $\frac{1}{\mathbf{y}}$ we understand the vector obtain by applying the division element-wise (the i-th element of is $\frac{1}{\mathbf{y}_i}$).

$$\begin{aligned} \mathbf{G_N} &= \frac{1}{n} \sum_{\mathbf{x}^{(i)}, \mathbf{t}^{(i)}} \mathbf{J_r}^T \mathbf{H}_{\mathcal{L} \circ \mathbf{r}} \mathbf{J_r} \\ &= \frac{1}{n} \sum_{\mathbf{x}^{(i)}} \mathbf{J_r}^T diag\left(\mathbf{y}(1 - \mathbf{y})\right) diag\left(\frac{1}{\mathbf{y}(1-\mathbf{y})}\right) diag\left(\mathbf{y}(1 - \mathbf{y})\right) \mathbf{J_r} \\ &= \mathbb{E}_{\mathbf{x} \sim \tilde{q}} \left[\mathbf{J_y}^T diag\left(\frac{1}{\mathbf{y}(1-\mathbf{y})}\right) \mathbf{J_y} \right] \end{aligned} \tag{12}$$

The last matching activation and error function that we consider is the softmax with cross-entropy. The derivation of the Gauss-Newton approximation is given in equation (13).

---

[3]By an abuse of langue, the term Hessian-Free Optimization refers to the specific algorithm proposed by Martens (2010), whily KSD stands for the specific algorithm proposed by Vinyals and Povey (2012)

$$
\begin{aligned}
\mathbf{H}_{\mathcal{L}\text{or}_{ij,i\neq j}} &= \frac{\partial^2 \sum_k (-t_k \log(\phi(r_k)))}{\partial r_i \partial r_j} = \frac{\partial \sum_k (t_k \phi(r_i)) - t_i}{\partial r_j} \\
&= -\phi(r_i)\phi(r_j) \\
\mathbf{H}_{\mathcal{L}\text{or}_{ii}} &= \ldots = \frac{\partial \phi(r_i) - t_i}{\partial r_i} = \phi(r_i) - \phi(r_i)\phi(r_i)
\end{aligned}
\tag{13}
$$

$$
\begin{aligned}
\mathbf{F} &= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\sum_{k=1}^{o}\frac{1}{y_k}\left(\frac{\partial y_k}{\partial \theta}\right)^T\frac{\partial y_k}{\partial \theta}\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\mathbf{J}_{\mathbf{r}}^T\left(\sum_{k=1}^{o}\frac{1}{y_k}\left(\frac{\partial y_k}{\partial \mathbf{r}}\right)^T\left(\frac{\partial y_k}{\partial \mathbf{r}}\right)\right)\mathbf{J}_{\mathbf{r}}\right] \\
&= \frac{1}{N}\sum_{\mathbf{x}^{(i)}}\left(\mathbf{J}_{\mathbf{r}}^T\mathbf{M}\mathbf{J}_{\mathbf{r}}\right)
\end{aligned}
\tag{14}
$$

$$
\begin{aligned}
\mathbf{M}_{ij,i\neq j} &= \sum_{k=1}^{o}\frac{1}{y_k}\frac{\partial y_k}{\partial r_i}\frac{\partial y_k}{\partial r_j} = \sum_{k=1}^{o}(\delta_{ki}-y_i)y_k(\delta_{kj}-y_j) \\
&= y_i y_j - y_i y_j - y_i y_j = -\phi(r_i)\phi(r_j) \\
\mathbf{M}_{ii} &= \sum_{k=1}^{o}\frac{1}{y_k}\frac{\partial y_k}{\partial y_i}\frac{\partial y_k}{\partial r_j} = y_i^2\left(\sum_{k=1}^{o}y_k\right) + y_i - 2y_i^2 \\
&= \phi(r_i) - \phi(r_i)\phi(r_i)
\end{aligned}
\tag{15}
$$

Equation (14) starts from the natural gradient metric and singles out a matrix $\mathbf{M}$ in the formula such that the metric can be re-written as the product $\mathbf{J}_{\mathbf{r}}^T\mathbf{M}\mathbf{J}_{\mathbf{r}}$ (similar to the formula for the Gauss-Newton approximation). In equation (15) we show that indeed $\mathbf{M}$ equals $\mathbf{H}_{\mathcal{L}\text{or}}$ and hence the natural gradient metric is the same as the extended Gauss-Newton matrix for this case as well. Note that δ is the Kronecker delta, where $\delta_{ij,i\neq j}=0$ and $\delta_{ii}=1$.

There is also a one to one mapping for most of the other heuristics used by Hessian-Free Optimization. Following the functional manifold interpretation of the algorithm, we can recover the Levenberg-Marquardt heuristic used in Martens (2010). This is a heuristic for adapting α in the damping term $\alpha\mathbf{I}$ that is added to the Hessian before inverting it. This additive term helps making the matrix easier to invert (for example when it is close to singular) or to deal with negative curvature. $\mathbf{I}$ is the identity matrix, $\alpha \in \mathbb{R}$ is a scalar.

The justification of the method comes from a trust region approach. We look at how well our second order Taylor approximation of the function predicts the change when taking a step. If it does well, then we can trust our approximation (and decrease the damping α). If our approximation does not predict well the change in the error, we increase the damping factor. We can see that in the limit, when $\alpha \to 0$, we completely trust our approximation and use no damping. When α is very large, there are two things happening. First of all, we are taking much smaller steps (scaled by $\frac{1}{\alpha}$). Secondly, the direction that we follow is mostly that of the gradient (ignoring the curvature information of the Hessian).

A similar argument can be carried on for natural gradient descent, where we consider only a first order Taylor approximation on the manifold. For any function f, if $\mathbf{p}$ depicts the picked descent direction and η the step size

$$
f(\theta_t - \eta\mathbf{p}) \approx f(\theta_t) - \eta\frac{\partial f(\theta_t)}{\partial \theta_t}\mathbf{p}
\tag{16}
$$

This gives the reduction ratio given by equation (17) which can, under the assumption that $p = \frac{\partial f(\theta_t)}{\partial \theta_t}\mathbf{F}^{-1}$, be shown to behave identically with the one in Martens (2010) (under the same assumption, namely that CG is close to convergence).

$$
\rho = \frac{f(\theta_t - \eta\mathbf{p}) - f(\theta_t)}{-\eta\frac{\partial f(\theta_t)}{\partial \theta_t}\mathbf{p}} \approx \frac{f\left(\left(\theta - t - \eta\mathbf{F}^{-1}\frac{\partial f(\theta_t)}{\partial \theta_t}^T\right)\right) - f(\theta_t)}{-\eta\frac{\partial f(\theta_t)}{\partial \theta_t}\mathbf{F}^{-1}\frac{\partial f(\theta_t)}{\partial \theta_t}^T}
\tag{17}
$$

Structural damping (Martens and Sutskever, 2011), a specific regularization term used to improve training of recurrent neural network, can also be explained from the natural gradient descent perspective. Roughly it implies using the joint probability density function $p(\mathbf{t}, \mathbf{h}|\mathbf{x})$, where $\mathbf{h}$ is the hidden state, when writing the KL-constraint. $\log p(\mathbf{t}, \mathbf{h}|\mathbf{x})$ will break in the sum of two terms, one being the Fisher Information Matrix, while the other measures the change in $\mathbf{h}$ and forms the

structural damping term. While theoretically pleasing, however this derivation results in a fixed coefficient of 1 for the regularization term.

We can fix this by using two constraints when deriving the natural gradient descent algorithm, namely:

$$
\begin{aligned}
&\arg\min_{\Delta\theta} \mathcal{L}(\theta + \Delta\theta) \\
&\text{s. t. } \mathbb{E}_{\mathbf{x}\sim\tilde{q}(\mathbf{x})}\left[KL(p_\theta(\mathbf{t}|\mathbf{x})||p_{\theta+\Delta\theta}(\mathbf{t}|\mathbf{x}))\right] = const. \\
&\text{and } \mathbb{E}_{\mathbf{x}\sim\tilde{q}(\mathbf{x})}\left[KL(p_\theta(\mathbf{h}|\mathbf{x})||p_{\theta+\Delta\theta}(\mathbf{h}|\mathbf{x}))\right] = const.
\end{aligned}
\tag{18}
$$

If we apply the same steps as before for both constraints (i.e. replace them by a second order Taylor expansion), the second term will give rise to the structural damping term.

6 Natural gradient descent and Krylov Subspace Descent

Instead of using linear conjugate gradient descent for computing the inverse of the metric, Krylov Subspace Descent(KSD) Vinyals and Povey (2012) [4] opts for restricting $\Delta\theta$ to a lower dimensional Krylov subspace given by $\mathbf{F}\mathbf{x} = \nabla\mathcal{L}$ and then, using some other second order method like BFGS, to solve for $\Delta\theta$ within this space.

Formally we are looking for $\gamma_1, \gamma_2, .., \gamma_k$ such that:

$$
\min_{\gamma} \mathcal{L}\left(\theta + \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \dots \\ \gamma_k \end{bmatrix} \begin{bmatrix} \nabla\mathcal{L} \\ \nabla\mathcal{L}\mathbf{F} \\ \dots \\ \nabla\mathcal{L}\mathbf{F}^{k-1} \end{bmatrix}\right)
\tag{19}
$$

Because it relies on the extended Gauss-Newton approximation of the Hessian, like Hessian-Free Optimization, KSD implements natural gradient descent. But there is an additional difference between KSD and HF that can be interpreted from the natural gradient descent perspective.

In order to mimic Hessian-Free Optimization's warm restart, this method adds to the Krylov Subspace the previous search direction. We hypothesize that due to this change, KSD is more similar to natural conjugate gradient than natural gradient. Natural conjugate gradient (see section 9) is an extension of the natural gradient descent algorithm that incorporates second order information by applying the nonlinear conjugate gradient algorithm on top of the natural gradients.

To show this we can rewrite the new subspace as:

$$
\min \mathcal{L}\left(\theta + \beta d_{t-1} + \alpha \begin{bmatrix} \frac{\gamma_1}{\alpha} \\ \frac{\gamma_2}{\alpha} \\ \dots \\ \frac{\gamma_k}{\alpha} \end{bmatrix} \begin{bmatrix} \nabla\mathcal{L} \\ \nabla\mathcal{L}\mathbf{F} \\ \dots \\ \nabla\mathcal{L}\mathbf{F}^{k-1} \end{bmatrix}\right)
\tag{20}
$$

From this formulation one can see that the previous direction plays a different role than the one played when doing warm restart of CG. The algorithm is reminiscent of the nonlinear conjugate gradient. The descent directions we pick, besides incorporating the natural gradient direction, also tend to be locally conjugate to the Hessian of the error with respect to the functional behaviour of the model. Additionally, compared to nonlinear conjugate gradient BFGS is used to compute β rather then some known formula.

7 Using unlabeled data

When computing the metric of natural gradient descent, the expectation over the target $\mathbf{t}$ is computed where $\mathbf{t}$ is taken from the model distribution for some given $\mathbf{x}$: $\mathbf{t} \sim p(\mathbf{t}|\mathbf{x})$. For the standard neural network models this expectation can be evaluated analytically (given the form of $p(\mathbf{t}|\mathbf{x})$). This means that we do not need target values to compute the metric of natural gradient descent.

Don't need labeled data

Furthermore, to compute the natural gradient descent direction we need to evaluate two different expectations over $\mathbf{x}$. The first one is when we evaluate the expected (Euclidean) gradient, while the

---

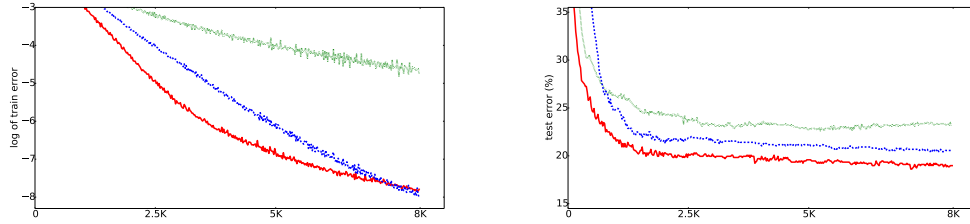[4]KSD stands for the exact algorithm proposed by Vinyals and Povey (2012)

Figure 1: The plot depicts the training error (left) on a log scale and test error (right) as percentage of misclassified examples for fold 4 of TFD. On the x-axis we have the number of updates. Dotted green line (top, worst) stands for using the same minibatch (of 256 examples) to compute the gradient and evaluate the metric. Dashed blue line (middle) uses a different minibatch of 384 examples from the training set to evaluate the metric. The solid red line (bottom, best) relies on a randomly sampled batch of unlabeled data to estimate the metric.

second is when we evaluate the metric. In this section we explore the effect of re-using the same samples in computing these two expectations as well as the effect of improving the accuracy of the metric $\mathbf{F}$ by employing unlabeled data.

Statistically, if both expectations over $\mathbf{x}$ are computed from the same samples, the two estimations are not independent from each other. We hypothesize that this can lead to the overfitting of the current minibatch. Fig. 1 provides empirical evidence that our hypothesis is correct. Vinyals and Povey (2012) make a similar empirical observation.

As discussed in section 3, enforcing a constant change in the model distribution helps ensuring stable progress but also protects from large changes in the model which can be detrimental (could result in overfitting a subset of the data). We get this effect as long as the metric provides a good measure of how much the model distribution changes. Unfortunately the metric is computed over training examples, and hence it will focus on how much p changes at these points. When learning overfits the training set we usually observe reduction in the training error that result in larger increases of the generalization error. This behaviour can be avoided by natural gradient descent if we can measure how much p changes far away from the training set. To explore this idea we propose to increase the accuracy of the metric $\mathbf{F}$ by using unlabeled data, helping us to measure how p changes far away from the training set.

We explore empirically these two hypotheses on the Toronto Face Dataset (TFD) (Susskind *et al.*, 2010) which has a small training set and a large pool of unlabeled data. Fig. 1 shows the training and test error of a model trained on fold 4 of TFD, though similar results are obtained for the other folds.

We used a two layer model, where the first layer is convolutional. It uses 512 filters of 14X14, and applies a sigmoid activation function. The next layer is a dense sigmoidal one with 1024 hidden units. The output layer uses sigmoids as well instead of softmax. The data is pre-processed by using local contrast normalization. Hyper-parameters have been selected using a grid search (more details in the appendix).

We notice that *re-using the same samples for the metric and gradient results in worse global training error* (training error over the entire train set) *and worse test error.*

Our intuition is that we are seeing the model overfitting, at each step, the current training minibatch. At each step we compute the gradient and the metric on the same example. There can be, within this minibatch, some direction in which we could overfit some specific property of these examples. Because we only look at how the model changes at these points when computing the metric, all gradients will agree with this direction. Specifically, the model believes that moving in this direction will lead to steady change in the KL (the KL looks at the covariance between the gradients from the output layer to the parameters, which will be low in the picked direction) and the model will believe it can take a large step, learning this specific feature. However it is not useful for generalization, nor is it useful for the other training examples (e.g. if the particular deformation is not actually

8

correlated with the classes it needs to predict). This means that on subsequent training examples the model will under-perform, resulting in a worse overall error.

On the other hand, if we use a different minibatch for the gradient and for the metric, it is less likely that the same particular feature to be present in both the set of examples used for the gradient and those used for the metric. So either the gradient will not point in said direction (as the feature is not present in the gradient), or, if the feature is present in the gradient, it may not be present in the examples used for computing the metric. That would lead to a larger variance, and hence the model is less likely to take a large step in this direction.

Using unlabeled data results in better test error for worse training error, which means that this way of using the unlabeled data acts like a regularizer.

8 Robustness to reorderings of the train set

We want to test the hypothesis that, by forcing the change in the model, in terms of the KL, to be constant, the model avoids to take large steps, being more robust to the reordering of the train set.

We repeat the experiment from Erhan *et al.* (2010), using the NISTP dataset introduced in Bengio *et al.* (2011) (which is just the NIST dataset plus deformations) and use 32.7M samples of this data. The original experiment attempts to measure the importance of the early examples in the learnt model. To achieve this we respect the same protocol as the original paper described below:

Algorithm 1 Protocol for the robustness to the order of training examples

1: Split the training data into two large chunks of 16.4M data points
2: Split again the first chunk into 10 equal size segments
3: **for** i between 1 and 10 **do**
4:     **for** steps between 1 and 5 **do**
5:         Replace segment i by new randomly sampled examples that have not been used before
6:         Train the model from scratch
7:         Evaluate the model on 10K heldout examples
8:     **end for**
9:     Compute the segment i mean variance, among the 5 runs in the output of the trained model (the variance in the activations of the output layer)
10: **end for**
11: Plot the mean variance as a function of which segment was resampled

---

Fig. 2 shows these curves for minibatch stochastic gradient descent and natural gradient descent.

Note that the variance at each point on the curve depends on the speed with which we move in functional space. For a fixed number of examples one can artificially tweak the curves by decreasing the learning rate. With a smaller learning rate we move slower, and hence the model, from a functional point of view, does not change by much, resulting in low variance. In the limit, with a learning rate of 0, the model always stays the same. In order to be fair to the two algorithms compared in the plot, natural gradient descent and stochastic gradient descent, we use the error on a different validation set as a measure of how much we moved in the functional space. This helps us to chose hyper-parameters such that after 32.7M samples both methods achieve about the same validation error of 49.8% (see appendix for hyper-parameters). Both models are run on minibatches of the same size, 512 samples. We use the same minibatch to compute the metric as we do to compute the gradient for natural gradient descent, to avoid favorazing NGD over MSGD.

The results are consistent with our hypothesis that natural gradient descent avoids making large steps in function space during training, staying on the path that induces least variance. Note that relative variance might not be affected when switching to natural gradient descent. That is to be expected, as, in the initial stages of learning, any gradient descent algorithm needs to choose a basin of attraction (a local minima) while subsequent steps make progress towards these minima. That is, the first examples must have, relatively speaking, a big impact on the learnt model. However what we are trying to investigate is the overall variance. Natural gradient descent (in global terms) has lower variance, resulting in more consistent functional behaviour regardless of the order of the input examples.
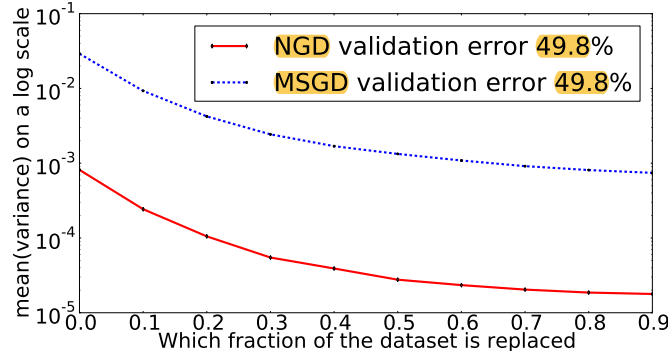
Figure 2: The plot describes how much the model is influenced by different parts of an online training set. The variance induced by re-shuffling of data for natural gradient descent is order of magnitudes lower than for SGD. See text for more information.

SGD might move in a direction early on that could possibly yield overfitting (e.g. getting forced into some quadrant of parameter space based only on a few examples) resulting in different models at the end. This suggests that natural gradient descent can deal better with nonstationary data and can be less sensitive to the particular examples selected early on during training.

9 Natural conjugate gradient

Natural gradient descent is a first order method in the space of functions, which raises the question: can the algorithm be improved by considering second order information? Unfortunately computing the *Hessian on the manifold* can be daunting (especially from a computational perspective). That is the Hessian of the error as a function of the density probability functions $p_\theta(\mathbf{t}|\mathbf{x})$ that give rise to the manifold structure. Note that in Roux and Fitzgibbon (2010) a method for combining second order information and TONGA is proposed. It is not clear how this algorithm can be applied to natural gradient descent. Also in Roux and Fitzgibbon (2010) a different Hessian matrix is considered, namely that of the error as a function of the parameter.

A simpler option is, however, to use an optimization method such as Nonlinear Conjugate Gradient, that is able to take advantage of second order structure (by following locally conjugate directions) while not requiring to actually compute the Hessian at any point in time, only the gradient. Absil *et al.* (2008) describes how second order methods can be generalized to the manifold case. In Honkela *et al.* (2008, 2010) a similar idea is proposed in the context of variational inference and specific assumptions on the form of p_θ are made. Gonzalez and Dorronsoro (2006) is more similar in spirit with this work, though their approach is defined for the diagonal form of the Fisher Information Matrix and differs in how they propose to compute a new conjugate direction.

Natural conjugate gradient, the manifold version of nonlinear conjugate gradient, is defined following the same intuitions (Shewchuck, 1994). The only problematic part of the original algorithm is how to obtain a new conjugate direction given the previous search direction and the current natural gradient. The problem arises from the fact that the two vectors belong to different spaces. The local geometry around the point where the previous search direction d_{t-1} was computed is defined by $\mathbf{F}_{t-1}$, while the geometry around the new direction is defined by $\mathbf{F}_t$, where, in principle, $\mathbf{F}_{t-1} \neq \mathbf{F}_t$. Following Absil *et al.* (2008) we would need to "transport" d_{t-1} into the space of ∇_{N_t}, an expensive operation, before we can compute a new direction using a standard formula like Polak-Ribiere. Furthermore, the line search should be done along the geodesic of the manifold.

Gonzalez and Dorronsoro (2006); Honkela *et al.* (2010) address these issues by making the assumption that $\mathbf{F}_{t-1}$ and $\mathbf{F}_t$ are identical (so d_{t-1} does not need to be transported). This assumption is detrimental to the algorithm because it goes against what we want to achieve. By employing a conjugate gradient method we hope to make large steps, from which it follows that the metric is very likely to change. Hence the assumption can not hold.

Computing the right direction is difficult, as the transportation operator for d_{t-1} is hard to compute in a generic manner, without enforcing strict constraints on the form of p_θ. Inspired by our new interpretation of the KSD algorithm, we propose instead to solve for the coefficients β and α that minimizes our cost, where $\frac{\beta}{\alpha}$ is the coefficient required in computing the new direction and α is the step size:

$$\min_{\alpha,\beta} \mathcal{L}\left(\theta_{t-1} + \begin{bmatrix} \alpha_t \\ \beta_t \end{bmatrix} \begin{bmatrix} \nabla_{N_t} \\ d_{t-1} \end{bmatrix}\right) \tag{21}$$

The new direction therefore is:

$$d_t = \nabla_{N_t} + \frac{\beta_t}{\alpha_t} d_{t-1} \tag{22}$$

Note that the resulting algorithm looks identical to standard nonlinear conjugate gradient. The difference are that we use the natural direction ∇_{N_t} instead of the gradients ∇_t everywhere and that we use an off the shelf solver to optimize simultaneously for both the step size α_t and the correction term $\frac{\beta_t}{\alpha_t}$ used to compute the new direction.

We can show that in the Euclidean space, given a second order Taylor approximation of $\mathcal{L}(\theta_{t-1})$, under the assumption that the Hessian $\mathbf{H}_{t-1}$ is symmetric, this approach will result in conjugate direction. Because we do not do the multidimensional line search along geodesics or transport the gradients in the same space, in general, this approach is also just an approximation to the locally conjugate direction on the manifold.

The advantage of our approach over using some formula, like Polak-Ribiere, and ignoring the difference between $\mathbf{F}_{t-1}$ and $\mathbf{F}_t$, is that we are guaranteed to minimize the cost $\mathcal{L}$ at at each step.

Let us show that the resulting direction is conjugate to the previous one in Euclidean space. Let d_{t-1} be the previous direction and γ_{t-1} the step size such that $\mathcal{L}(\theta_{t-1} + \gamma_{t-1} d_{t-1})$ is minimal for fixed d_{t-1}. If we approximate $\mathcal{L}$ by its second order Taylor expansion and compute the derivative with respect to the step size γ_{t-1} we have that:

$$\frac{\partial \mathcal{L}}{\partial \theta} d_{t-1} + \gamma_{t-1} d_{t-1}^T \mathbf{H} d_{t-1} = 0 \tag{23}$$

Suppose now that we take the next step which takes the form of

$$\mathcal{L}(\theta_t + \beta_t d_{t-1} + \alpha_t \nabla_{N_t}^T)$$
$$= \mathcal{L}(\theta_{t-1} + \gamma_{t-1} d_{t-1} + \beta_t d_{t-1} + \alpha_t \nabla_{N_t}^T)$$

where we minimize for α_t and β_t. If we replace $\mathcal{L}$ by its second order Taylor series around θ_{t-1}, compute the derivative with respect to β_t and use the assumption that $\mathbf{H}_{t-1}$ (where we drop the subscript) is symmetric, we get:

$\frac{\partial \mathcal{L}}{\partial \theta} d_{t-1} + \alpha_t \nabla_{N_t} \mathbf{H} d_{t-1} + (\gamma_{t-1} + \beta_t) d_{t-1}^T \mathbf{H} d_{t-1} = 0$

Using the previous relation 23 this implies that

$(\alpha_t \nabla_{N_t} \mathcal{L}^T + \beta_t d_{t-1})^T \mathbf{H} d_{t-1} = 0$, i.e. that the new direction is conjugate to the last one.

10 Benchmark

We carry out a benchmark on the Curves dataset, using the 6 layer deep auto-encoder from Martens (2010). The dataset is small, has only 20K training examples of 784 dimensions each. All methods except SGD use the truncated Newton pipeline. The benchmark is run on a GTX 580 Nvidia card, using Theano (Bergstra *et al.*, 2010a) for cuda kernels. Fig. 3 contains the results. See supplementary materials for details.

For natural gradient we present two runs, one in which we force the model to use small minibatches of 5000 examples, and one in which we run in batch mode. In both cases all other hyper-parameters were manually tuned. The learning rate is fixed at 1.0 for the model using a minibatch size of 5000 and a line search was used when running in batch mode. For natural conjugate gradient we use `scipy.optimize.fmin_cobyla` to solve the two dimensional line search. This corresponds to the new algorithm introduced in section 9. We additionally run the same algorithm, where we rely on the Polak-Riebiere formula for computing the new direction and use standard line search for the

learning rate. This corresponds to our implementation of the algorithm proposed in Gonzalez and Dorronsoro (2006), where instead of a diagonal approximation of the metric we reply on a truncated Newton approach. In both cases we run in full batch mode. For SGD we used a minibatch of 100 examples and a learning rate of 0.01. Note that our implementation of natural gradient does not directly matches James Martens' Hessian Free as there are components of the algorithm that we do not use like backtracking and preconditioning.
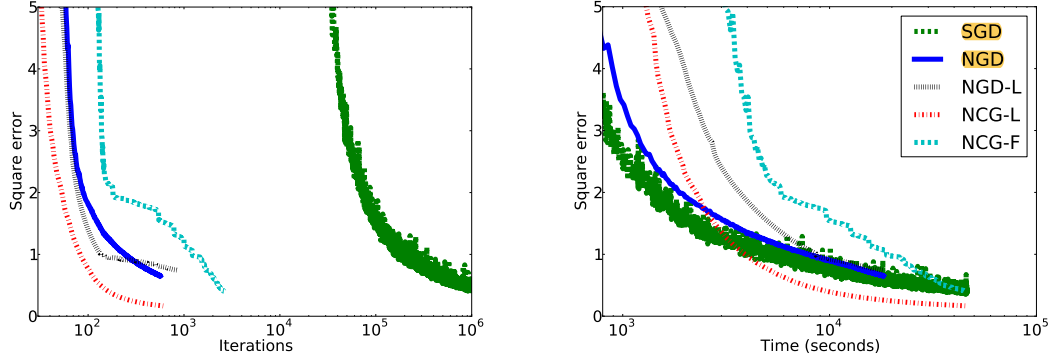

Figure 3: The plots depict the training curves on a log scale (x-axis) for different learning algorithms on the Curves dataset. On the y-axis we have training error on the whole dataset. The left plot shows the curves in terms of iterations, while the right plot show the curves as a function of clock time (seconds). NGD stands for natural gradient descent descent with a minibatch of 1000 and a fixed learning rate. NGD-L is natural gradient descent in batch mode where we use a line search for the learning rate. NCG-L uses an `scipy.optimize.fmin_cobyla` for finding both learning rate and the new conjugate direction. NCG-F employs the Polak-Ribiere formula, ignoring the change in the metric. SGD stands for stochastic gradient descent.

The first observation that we can make is that natural gradient descent can run reliably with smaller minibatches, as long as the gradient and the metric are computed on different samples and the learning rate used is sufficiently small to account for the noise on the natural direction (or the damping coefficient is sufficiently large). This can be seen from comparing the two runs of natural gradient descent descent. Our result is in the spirit of the work of Kiros (2013), though the exact approach of dealing with small minibatches is slightly different.

The second observation that we can draw is that incorporating second order information into natural gradient descent might be beneficial. We see a speedup in convergence. The result agrees with the observation made in Vinyals and Povey (2012), where Krylov Subspace Descent (KSD) was shown to converge slightly faster than Hessian-Free Optimization. Given the relationship between natural gradient and Hessian-Free Optimization, and between KSD and natural conjugate gradient, we expected to see the same trend between natural gradient descent and natural conjugate gradient. We regard our algorithm as similar to KSD, with its main advantage being that it avoids storing in memory the Krylov subspace. Furthermore, from the comparison between NCG-L and NCG-F we can see evidence for our argumentation that it is useful to use an optimizer for finding the new direction. The reason why NCG-F under performs is that the smooth change in the metric assumption contradicts what the algorithm is trying to do, namely move as far as possible in parameter space.

11 Discussion and conclusions

In this paper we have made the following original contributions. We showed that by employing the extended Gauss-Newton approximation of the Hessian both Hessian-Free Optimization and Krylov Subspace Descent can be interpreted as implementing natural gradient descent. Furthermore, by adding the previous search direction to the Krylov subspace, KSD does something akin to an approximation to nonlinear conjugate gradient on the manifold.

We proposed an approximation of nonlinear conjugate gradient on the manifold which is similar in spirit to KSD, with the difference that we rely on linear conjugate gradient to invert the metric **F**

instead of using a Krylov subspace. This allows us to save on the memory requirements which can become prohibitive for large models, while still retaining some of the second order information of the cost and hence converging faster than vanilla natural gradient.

Lastly we highlighted the difference between natural gradient descent and TONGA, and brought forward two new properties of the algorithm compared to stochastic gradient descent. The first one is an empirical evaluation of the robustness of natural gradient descent to the order of examples in the training set, robustness that can be crucial when dealing with non-stationary data. The second property is the ability of natural gradient descent to guard against large drops in generalization error especially when the accuracy of the metric is increased by using unlabeled data. We believe that these properties as well as others found in the literature and shortly summarized in section 3 might provide some insight in the recent success of Hessian Free and KSD for deep models.

Finally we make the empirical observation that natural gradient descent can perform well even when its metric and gradients are estimated on rather small minibatches. One just needs to use different samples for computing the gradients from those used for the metric. Also the step size and damping coefficient need to be bounded to account for the noise in the computed descent direction.

Acknowledgments

References

Absil, P.-A., Mahony, R., and Sepulchre, R. (2008). *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton, NJ.

Amari, S. (1985). Differential geometrical methods in statistics. *Lecture notes in statistics*, **28**.

Amari, S. (1997). Neural learning in structured parameter spaces - natural Riemannian gradient. In *In Advances in Neural Information Processing Systems*, pages 127–133. MIT Press.

Amari, S., Kurata, K., and Nagaoka, H. (1992). Information geometry of Boltzmann machines. *IEEE Trans. on Neural Networks*, **3**, 260–271.

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural Comput.*, **10**(2), 251–276.

Arnold, L., Auger, A., Hansen, N., and Olivier, Y. (2011). Information-geometric optimization algorithms: A unifying picture via invariance principles. *CoRR*, **abs/1106.3708**.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., Bouchard, N., and Bengio, Y. (2012). Theano: new features and speed improvements. Submited to Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

Bengio, Y., Bastien, F., Bergeron, A., Boulanger-Lewandowski, N., Breuel, T., Chherawala, Y., Cisse, M., Côté, M., Erhan, D., Eustache, J., Glorot, X., Muller, X., Pannetier Lebeuf, S., Pascanu, R., Rifai, S., Savard, F., and Sicard, G. (2011). Deep learners benefit more from out-of-distribution examples. In *JMLR W&CP: Proc. AISTATS'2011*.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010a). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010b). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc.

Chapelle, O. and Erhan, D. (2011). Improved Preconditioner for Hessian Free Optimization. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.

Choi, S.-C. T., Paige, C. C., and Saunders, M. A. (2011). MINRES-QLP: A Krylov subspace method for indefinite or singular symmetric systems. **33**(4), 1810–1836.

Desjardins, G., Pascanu, R., Courville, A., and Bengio, Y. (2013). Metric-free natural gradient for joint-training of boltzmann machines. *ICLR*, **abs/1301.3545**.

Erhan, D., Courville, A., Bengio, Y., and Vincent, P. (2010). Why does unsupervised pre-training help deep learning? In *JMLR W&CP: Proc. AISTATS'2010*, volume 9, pages 201–208.

Gonzalez, A. and Dorronsoro, J. (2006). Natural conjugate gradient training of multilayer perceptrons. *Artificial Neural Networks ICANN 2006*, pages 169–177.

Heskes, T. (2000). On natural learning and pruning in multilayered perceptrons. *Neural Computation*, **12**, 1037–1057.

Honkela, A., Tornio, M., Raiko, T., and Karhunen, J. (2008). Natural conjugate gradient in variational inference. In *Neural Information Processing*, pages 305–314.

Honkela, A., Raiko, T., Kuusela, M., Tornio, M., and Karhunen, J. (2010). Approximate riemannian conjugate gradient learning for fixed-form variational bayes. *Journal of Machine Learning Research*, **11**, 3235–3268.

Kakade, S. (2001). A natural policy gradient. In *NIPS*, pages 1531–1538. MIT Press.

Kiros, R. (2013). Training neural networks with stochastic hessian-free optimization. *ICLR*.

Le Roux, N., Manzagol, P.-A., and Bengio, Y. (2008). Topmoumoute online natural gradient algorithm. In *NIPS'07*.

Martens, J. (2010). Deep learning via hessian-free optimization. In *ICML*, pages 735–742.

Martens, J. and Sutskever, I. (2011). Learning recurrent neural networks with hessian-free optimization. In *ICML*, pages 1017–1024.

Mizutani, E. and Demmel, J. (2003). Iterative scaled trust-region learning in krylov subspaces via peralmutter's implicit sparse hessian-vector multiply. In *NIPS*, pages 209–216.

Nocedal, J. and Wright, S. J. (2000). *Numerical Optimization*. Springer.

Park, H., Amari, S.-I., and Fukumizu, K. (2000). Adaptive natural gradient learning algorithms for various stochastic models. *Neural Networks*, **13**(7), 755 – 764.

Pearlmutter, B. A. (1994). Fast exact multiplication by the hessian. *Neural Computation*, **6**, 147–160.

Peters, J. and Schaal, S. (2008). Natural actor-critic. (7-9), 1180–1190.

Roux, N. L. and Fitzgibbon, A. W. (2010). A fast natural newton method. In *ICML*, pages 623–630.

Schaul, T. (2012). Natural evolution strategies converge on sphere functions. In *Genetic and Evolutionary Computation Conference (GECCO)*.

Schraudolph, N. N. (2002). Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, **14**(7), 1723–1738.

Shewchuck, J. (1994). An introduction to the conjugate gradient method without the agonizing pain. Technical report, CMU.

Sohl-Dickstein, J. (2012). The natural gradient by analogy to signal whitening, and recipes and tricks for its use. *CoRR*, **abs/1205.1828**.

Sun, Y., Wierstra, D., Schaul, T., and Schmidhuber, J. (2009). Stochastic search using the natural gradient. In *ICML*, page 146.

Susskind, J., Anderson, A., and Hinton, G. E. (2010). The Toronto face dataset. Technical Report UTML TR 2010-001, U. Toronto.

Vinyals, O. and Povey, D. (2012). Krylov Subspace Descent for Deep Learning. In *AISTATS*.

Appendix

Expected Hessian to Fisher Information Matrix

The Fisher Information Matrix form can be obtained from the expected value of the Hessian :

$$
\begin{aligned}
\mathbb{E}_{\mathbf{z}}\left[-\frac{\partial^2 \log p_\theta}{\partial \theta}\right] &= \mathbb{E}_{\mathbf{z}}\left[-\frac{\partial \frac{1}{p_\theta}\frac{\partial p_\theta}{\partial \theta}}{\partial \theta}\right] = \mathbb{E}_{\mathbf{z}}\left[-\frac{1}{p_\theta(\mathbf{z})}\frac{\partial^2 p_\theta}{\partial \theta^2} + \left(\frac{1}{p_\theta}\frac{\partial p_\theta}{\partial \theta}\right)^T\left(\frac{1}{p_\theta}\frac{\partial p_\theta}{\partial \theta}\right)\right] \\
&= -\frac{\partial^2}{\partial \theta^2}\left(\sum_{\mathbf{z}} p_\theta(\mathbf{z})\right) + \mathbb{E}_{\mathbf{z}}\left[\left(\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right)^T\left(\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right)\right] \\
&= \mathbb{E}_{\mathbf{z}}\left[\left(\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right)^T\left(\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right)\right] \qquad (24)
\end{aligned}
$$

Derivation of the natural gradient descent metrics

Linear activation function

In the case of linear outputs we assume that each entry of the vector t, t_i comes from a Gaussian distribution centered around $\mathbf{y}_i(\mathbf{x})$ with some standard deviation β. From this it follows that:

$$p_\theta(\mathbf{t}|\mathbf{x}) = \prod_{i=1}^{o} \mathcal{N}(t_i|\mathbf{y}(\mathbf{x},\theta)_i, \beta^2) \tag{25}$$

$$
\begin{aligned}
\mathbf{F} &= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\mathbb{E}_{\mathbf{t}\sim\mathcal{N}(\mathbf{t}|\mathbf{y}(\mathbf{x},\theta),\beta^2\mathbf{I})}\left[\sum_{i=1}^{o}\left(\frac{\partial\log_\theta p(t_i|\mathbf{y}(\mathbf{x})_i}{\partial\theta}\right)^T\left(\frac{\partial\log p_\theta(t_i|\mathbf{y}(\mathbf{x})_i}{\partial\theta}\right)\right]\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\sum_{i=1}^{o}\mathbb{E}_{\mathbf{t}\sim\mathcal{N}(\mathbf{t}|\mathbf{y}(\mathbf{x},\theta),\beta^2\mathbf{I})}\left[\left(\frac{\partial(t_i-y_i)^2}{\partial\theta}\right)^T\left(\frac{\partial(t_i-y_i)^2}{\partial\theta}\right)\right]\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\sum_{i=1}^{o}\mathbb{E}_{\mathbf{t}\sim\mathcal{N}(\mathbf{t}|\mathbf{y}(\mathbf{x},\theta),\beta^2\mathbf{I})}\left[(t_i-y_i)^2\left(\frac{\partial y_i}{\partial\theta}\right)^T\left(\frac{\partial y_i}{\partial\theta}\right)\right]\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\sum_{i=1}^{o}\mathbb{E}_{\mathbf{t}\sim\mathcal{N}(\mathbf{t}|\mathbf{y}(\mathbf{x},\theta),\beta\mathbf{I})}\left[(t_i-y_i)^2\right]\left(\frac{\partial y_i}{\partial\theta}\right)^T\left(\frac{\partial y_i}{\partial\theta}\right)\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\sum_{i=1}^{o}\mathbb{E}_{\mathbf{t}\sim\mathcal{N}(\mathbf{t}|\mathbf{y}(\mathbf{x},\theta),\beta^2\mathbf{I})}\left[(t_i-y_i)^2\right]\left(\frac{\partial y_i}{\partial\theta}\right)^T\left(\frac{\partial y_i}{\partial\theta}\right)\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\sum_{i=1}^{o}\mathbb{E}_{\mathbf{t}\sim\mathcal{N}(\mathbf{t}|\mathbf{y}(\mathbf{x},\theta),\beta^2\mathbf{I})}\left[(t_i-y_i)^2\right]\left(\frac{\partial y_i}{\partial\theta}\right)^T\left(\frac{\partial y_i}{\partial\theta}\right)\right] \\
&= \beta^2\mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\sum_{i=1}^{o}\left[\left(\frac{\partial y_i}{\partial\theta}\right)^T\left(\frac{\partial y_i}{\partial\theta}\right)\right]\right] \\
&= \beta^2\mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\mathbf{J}_\mathbf{y}^T\mathbf{J}_\mathbf{y}\right]
\end{aligned}
\tag{26}
$$

Sigmoid activation function

In the case of the sigmoid units, i,e, $\mathbf{y} = \text{sigmoid}(\mathbf{r})$, we assume a binomial distribution which gives us:

$$p(\mathbf{t}|\mathbf{x}) = \prod_i \mathbf{y}_i^{t_i}(1-\mathbf{y}_i)^{1-t_i} \tag{27}$$

$\log p$ gives us the usual cross-entropy error used with sigmoid units. We can compute the Fisher information matrix as follows:

$$
\begin{aligned}
\mathbf{F} &= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\mathbb{E}_{\mathbf{t}\sim p(\mathbf{t}|\mathbf{x})}\left[\sum_{i=1}^{o}\frac{(t_i-y_i)^2}{y_i^2(1-y_i)^2}\left(\frac{\partial y_i}{\partial\theta}\right)^T\frac{\partial y_i}{\partial\theta}\right]\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\sum_{i=1}^{o}\frac{1}{y_i(1-y_i)}\left(\frac{\partial y_i}{\partial\theta}\right)^T\frac{\partial y_i}{\partial\theta}\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\tilde{q}}\left[\mathbf{J}_\mathbf{y}^T diag\left(\frac{1}{\mathbf{y}(1-\mathbf{y})}\right)\mathbf{J}_\mathbf{y}\right]
\end{aligned}
\tag{28}
$$

Note that $diag(\mathbf{v})$ stands for the diagonal matrix constructed from the values of the vector $\mathbf{v}$ and we make an abuse of notation, where by $\frac{1}{\mathbf{y}}$ we understand the vector obtain by applying the division element-wise (the i-th element of is $\frac{1}{\mathbf{y}_i}$).

Softmax activation function

For the softmax activation function, $\mathbf{y} = \text{softmax}(\mathbf{r})$, $p(\mathbf{t}|\mathbf{x})$ takes the form of a multinoulli:

$$p(\mathbf{t}|\mathbf{x}) = \prod_{i=1}^{o} y_i^{t_i} \tag{29}$$

15

$$\mathbf{F} = \mathbf{E}_{\mathbf{x} \sim \tilde{q}} \left[\sum_{i=1}^{o} \frac{1}{\mathbf{y}_i} \left(\frac{\partial y_i}{\partial \theta} \right)^T \frac{\partial y_i}{\partial \theta} \right] = \mathbf{E}_{\mathbf{x} \sim \tilde{q}} \left[\mathbf{J}_{\mathbf{y}}^T diag \left(\frac{1}{\mathbf{y}} \right) \mathbf{J}_{\mathbf{y}} \right] \tag{30}$$

Implementation Details

We have implemented natural gradient descent descent using a truncated Newton approach similar to the pipeline proposed by Pearlmutter (1994) and used by Martens (2010). In order to better deal with singular and ill-conditioned matrices we use the MinRes-QLP algorithm (Choi *et al.*, 2011) instead of linear conjugate gradient for certain experiments. Both Minres-QLP as well as linear conjugate gradient can be found implemented in Theano at https://github.com/pascanur/theano_optimize. We used the Theano library (Bergstra *et al.*, 2010a) which allows for a flexible implementation of the pipeline, that can automatically generate the computational graph of the metric times some vector for different models:

```
import theano.tensor as TT
# `params` is the list of Theano variables containing the parameters
# `vs` is the list of Theano variable representing the vector `v`
# with whom we want to multiply the metric
# `Gvs` is the list of Theano expressions representing the product
# between the metric and `vs`

# `out_smx` is the output of the model with softmax units
Gvs = TT.Lop(out_smx,params,
             TT.Rop(out_smx,params,vs)/(out_smx*out_smx.shape[0]))

# `out_sig` is the output of the model with sigmoid units
Gvs = TT.Lop(out_sig,params,
             TT.Rop(out_sig,params,vs)/(out_sig*
                                        (1-out_sig)*
                                        out_sig.shape[0]))

# `out` is the output of the model with linear units
Gvs = TT.Lop(out,params,TT.Rop(out,params,vs)/out.shape[0])
```

The full pseudo-code of the algorithm (which is very similar to the one for Hessian-Free Optimization) is given below.

Algorithm 2 Pseudocode for natural gradient descent algorithm

---

1: # **'gfn' is a function that computes the metric times some vector**
2: gfn $\leftarrow$ (lambda $v \rightarrow \mathbf{F}v$)
3: **while** not early_stopping_condition **do**
4:     g $\leftarrow \frac{\partial \mathcal{L}}{\partial \theta}$
5:     # **linear_cg solves the linear system** $Gx = \frac{\partial \mathcal{L}}{\partial \theta}$
6:     ng$\leftarrow$ linear_cg(gfn, g, max_iters = 20, rtol=1e-4)
7:     # γ **is the learning rate**
8:     $\theta \leftarrow \theta - \gamma$ ng
9: **end while**

Note that we do not usually do a line search for each step. While it could be useful, conceptually we like to think of the algorithm as a first order method. Doing a line search has the side effect that can lead to overfitting of the current minibatch. To a certain extent this can be fixed by using new samples of data to compute the error, though we find that we need to use reasonably large batches to get an accurate measure. Using a learning rate as for a first order method is a good alternative if one wants to apply this algorithm using small minibatches of data.

Even though we are ensured that $\mathbf{F}$ is positive semi-definite by construction, and MinRes-QLP is able to find a suitable solutions in case of singular matrices, we still use a damping strategy for two

reasons. The first one is that we want to take in consideration the inaccuracy of the metric (which is approximated only over a small minibatch). The second reason is that natural gradient descent makes sense only in the vicinity of θ as it is obtained by using a Taylor series approximation, hence (as for ordinary second order methods) it is appropriate to enforce a trust region for the gradient. See Schaul (2012), where the convergence properties of natural gradient descent (in a specific case) are studied.

Following the functional manifold interpretation of the algorithm, we can recover the Levenberg-Marquardt heuristic used in Martens (2010). We consider only a first order Taylor approximation, where, due to the functional manifold, for any function f,

$$f\left(\theta_t - \eta \mathbf{F}^{-1}\frac{\partial f(\theta_t)}{\partial \theta_t}^T\right) \approx f(\theta_t) - \eta \frac{\partial f(\theta_t)}{\partial \theta_t} \mathbf{F}^{-1} \frac{\partial f(\theta_t)}{\partial \theta_t}^T \tag{31}$$

We now can check how well this approximation predicts the change in the error for the step taken during an iteration. Based on this comparison we can define our reduction ratio given by equation (32). Based on this ratio we can decide if we want to increase or decrease the damping applied to the metric before inverting. As we will show below, this ratio behaves identically with the one in Martens (2010).

$$\rho = \frac{f\left(\theta_t - \eta \mathbf{F}^{-1}\frac{\partial f(\theta_t)}{\partial \theta_t}^T\right) - f(\theta_t)}{-\eta \frac{\partial f(\theta_t)}{\partial \theta_t}\mathbf{F}^{-1}\frac{\partial f(\theta_t)}{\partial \theta_t}^T} \tag{32}$$

Additional experimental results

TFD experiment

We used a two layer model, where the first layer is convolutional. It uses 512 filters of 14X14, and applies a sigmoid activation function. The next layer is a dense sigmoidal one with 1024 hidden units. The output layer uses sigmoids as well instead of softmax.

The data is pre-processed by using local contrast normalization. Hyper-parameters such as learning rate, starting damping coefficient have been selected using a grid search, based on the validation cost obtained for each configuration.

We ended up using a fixed learning rate of 0.2 (with no line search) and adapting the damping coefficient using the Levenberg-Marquardt heuristic.

When using the same samples for evaluating the metric and gradient we used minibatches of 256 samples, otherwise we used 384 other samples randomly picked from either the training set or the unlabeled set. We use MinResQLP as a linear solver, the picked initial damping factor is 5., and we allow a maximum of 50 iterations to the linear solver.

NISTP experiment (robustness to the order of training samples)

The model we experimented with was an MLP of only 500 hidden units. We compute the gradients for both MSGD and natural gradient descent over minibatches of 512 examples. In the case of natural gradient descent we compute the metric over the same input batch of 512 examples. Additionally we use a constant damping factor of 3 to account for the noise in the metric (and ill-conditioning since we only use batches of 512 samples). The learning rates were kept constant, and we use 0.2 for the natural gradient descent and 0.1 for MSGD.

Curves experiment - Benchmark

For the curves experiment we used a deep autoencoder with 400-200-100-50-25-6 hidden units respectively where we apply sigmoid activation function everywhere except on the top layer. We used the sparse initialization proposed by Martens (2010) for all runs.

For natural conjugate gradient we use the full batch of 20000 examples to compute the gradients or metric as well as for evaluating the model when optimizing for α and β using

scipy.optimize.fmin_cobyla. We use the Levenberg-Marquardt heuristic to adapt damping (and start with a damping value of 3). We use MinRes-QLP that we allow to run for a maximum of 250 steps. Batch size, number of linear solver iterations and damping coefficient have been selected by hand tuning, looking at validation set performance.

For natural gradient we end up reporting two runs. One uses full batch and a line search while the other uses small batches of 5000 examples and a fixed learning rate of 1.0.

For the SGD case we use a smaller batch size of 100 examples. The optimum learning rate, obtained by a grid search, is 0.01 which was kept constant through training. We did not use any additional enhancement like momentum.

A final note is that we show square error (to be consistent with Martens (2010)) we minimize cross-entropy in practice (i.e. the gradients are computed based on the cross-entropy cost, square error is only computed for visualization reasons).