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, 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 and three other recently proposed methods: Hessian-Free (Martens, 2010), Krylov Subspace Descent (Vinyals and Povey, 2012) and TONGA (Le Roux et al., 2008). We empirically evaluate the robustness of natural gradient to the ordering of the training set compared to stochastic gradient descent and show how unlabeled data can be used to improve generalization error. Another contribution is to extend natural gradient to incorporate second error information alongside the manifold information. Lastly we benchmark this new algorithm as well as natural gradient, where both are implemented 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; Sutskever et al., 2011; Chapelle and Erhan, 2011), Krylov Subspace Descent (Vinyals and Povey, 2012), 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) 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). 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 are as follows. In section 7 we extend natural gradient to incorporate second order information. 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 9 describes how unlabeled data can be incorporated into natural gradient in order to improve generalization error. Section 10 explores empirically natural gradient’s robustness to permutation of the training set. Finally in section 8 we provide a benchmark of the algorithm discussed, where natural gradient 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

Natural gradient 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 $F : \mathbb{R}^P \rightarrow (\mathbb{R}^N \rightarrow [0, \infty))$, where for every $\theta \in \mathbb{R}^P$, $F(\theta)$ defines a density probability function from $\mathbb{R}^N \rightarrow [0, \infty)$ over the random variable $z \in \mathbb{R}^N$. Any choice of $\theta \in \mathbb{R}^P$ defines a particular density function $p_\theta(z) = F(\theta)$ and by considering all possible $\theta$ values, we explore the set $\mathcal{F}$, which is our functional manifold.

Because we can define a similarity measure between nearby density functions, given by the KL-divergence which in its infinitesimal form behaves like a distance measure, $\mathcal{F}$ is a Riemannian manifold whose metric is given by the Fisher Information matrix $G$. Given a loss function $L$ parameterized by $\theta$, natural gradient attempts to move along the manifold by correcting the gradient of $L$ according to the local curvature of the KL-divergence surface, i.e. moving in direction $\nabla_N L(\theta)$:

$$\nabla_N L(\theta) \overset{\text{def}}{=} \nabla L(\theta) E_z \left[ (\nabla \log p_\theta(z))^T (\nabla \log p_\theta(z)) \right]^{-1} \overset{\text{def}}{=} \nabla L(\theta) G^{-1}. \quad (1)$$

We use $\nabla_N$ for natural gradient, $\nabla$ for gradients and $G$ 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 to be defined as the algorithm which, at each step, picks a descent direction such that the change induced in our model is constant. Specifically the KL-divergence between $p_\theta$ and $p_{\theta + \Delta \theta}$ has to be constant:

$$\arg \min_{\Delta \theta} L(\theta + \Delta \theta) \quad \text{s. t. } KL (p_\theta \mid\mid p_{\theta + \Delta \theta}) = \text{const}. \quad (2)$$

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 $\Delta \theta \rightarrow 0$, we can approximate the KL divergence by its Taylor series:

$$KL(p_\theta \mid\mid p_{\theta + \Delta \theta}) \approx (E_z [\log p_\theta] - E_z [\log p_{\theta + \Delta \theta}])$$

$$= E_z [\nabla \log p_\theta (z)] \Delta \theta - \frac{1}{2} \Delta \theta^T E_z [\nabla^2 \log p_\theta] \Delta \theta$$

$$= \frac{1}{2} \Delta \theta^T E_z [-\nabla^2 \log p_\theta (z)] \Delta \theta$$

$$= \frac{1}{2} \Delta \theta^T G \Delta \theta \quad (3)$$

The first term cancels out and because $E_z [\nabla \log p_\theta (z)] = \mathbf{0}$ 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 supplementary material).

We now express equation (2) as a Lagrangian, where the KL divergence is approximated by (3) and $L(\theta + \Delta \theta)$ by its first order Taylor series $L(\theta) + \nabla L(\theta) \Delta \theta$:

$$L(\theta) + \nabla L(\theta) \Delta \theta + \frac{1}{2} \lambda \Delta \theta^T G \Delta \theta \quad \quad (4)$$

Solving equation (4) for $\Delta \theta$ gives us the natural gradient formula (1). Note that we get a scalar factor of $\frac{1}{2} \lambda$ times the natural gradient. We fold this scalar into the learning rate, which now also controls

---

1 Proof: $E_z [\nabla \log p_\theta (z)] = \sum_z \left( p_\theta(z) \frac{1}{p_\theta(z)} \frac{\partial p_\theta(z)}{\partial \theta} \right) = \frac{\partial}{\partial \theta} \left( \sum \theta p_\theta(z) \right) = \frac{\partial}{\partial \theta} = 0$. The proof holds for the continuous case as well, replacing sums for integrals.
Each value of $x$, Natural gradient is robust to local re-parametrization of the model something natural gradient does not require. For natural gradient we need to estimate two different quantities. One is the gradient of the expected value of the KL-divergence between $p_{\theta}(t|x)$ and $p_{\theta+\Delta\theta}(t|x)$. In defining the constraint of eq. [5], we have chosen to allow ourselves the freedom to compute the expectation over $x$ 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. For 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. 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 supplementary material). For convenience we show the formulas for two typical pairs of output activations and corresponding error function, where $J_{y}$ stands for the Jacobian matrix $\frac{\partial y}{\partial \theta}$, $y$ being the output layer.

$$G_{\text{linear}} = \beta^{2} \mathbb{E}_{x \sim \tilde{q}} \left[ \frac{\partial y}{\partial \theta} \right] \left[ \frac{\partial y}{\partial \theta} \right]^T = \beta^{2} \mathbb{E}_{x \sim \tilde{q}} \left[ J_{y}^T J_{y} \right]$$

(6)

$$G_{\text{sigmoid}} = \mathbb{E}_{x \sim \tilde{q}} \left[ J_{y}^T \text{diag} \left( \frac{1}{y(1-y)} \right) J_{y} \right]$$

(7)

## 3 Properties of natural gradient

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

Natural gradient 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, 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 3).

If for an online regime one can have a sufficiently large set of held-out examples. But the same can be said about typical second order methods, which in principle are batch methods. For natural gradient we need to estimate two different quantities. One is the gradient of the loss, which can be our stochastic gradient computed over one sample. The other is the expected value of derivatives of the KL-divergence which can be computed using some held-out set of data.

Desjardins et al. (2013) shows a straightforward application of natural gradient to Deep Boltzmann Machines. Doing the same for a standard second order method is not as easy as for probabilistic models like RBMs and DBMs, since we do not have direct access to the cost we are minimizing, something natural gradient does not require.

Natural gradient 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 as doing whitening in the parameter space.

The metric $G$ has two different forms as can be seen in eq. (3). 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 $p_0$ which acts as a proxy for the cost $L$. The KL-surface considered at any point $p$ during training always has its minimum at $p_0$, and the metric we obtain is always positive semi-definite by construction which is not true for the Hessian of $L$.

Because $G$ can be interpreted as an expected Hessian, it measures how much a change in $\theta$ will affect the gradients of $\mathbb{E}_{t \sim p(t|x)} [\log p(t|x)]$. This means that, in the same way second order methods do, natural gradient can jump over plateaus of $p_0(t|x)$. Given the usual form of the loss function $L$, which is just an evaluation of $p$ for certain pairs $(x, t)$, plateaus of $p$ usually match those of $L$ 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 $d$ if moving along $d$ changes the density computed by model substantially.

### 4 Natural gradient 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 $C$. By using the uncertainty provided by $C$ 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.

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

$$U \approx \mathbb{E}_{(x,t) \sim q} \left[ \left( \frac{\partial \log p(t|x)}{\partial \theta} \right) ^T \left( \frac{\partial \log p(t|x)}{\partial \theta} \right) \right]$$  \hspace{1cm} (8)

The discrepancy comes from the fact that the eq. is an expectation, though the expectation is over the empirical distribution $q(x,t)$ as opposed to $x \sim q(x)$ and $t \sim p(t|x)$. It is therefore not clear if $U$ tells us how $p_0$ would change, whereas it is clear that Amari’s metric does.

### 5 Natural gradient and Hessian-Free

Hessian-Free as well as Krylov Subspace Descent rely on the extended Gauss-Newton approximation of the Hessian, $G_N$, instead of the actual Hessian (see Schraudolph (2002)):

$$G_N = \frac{1}{n} \sum_i \left[ \left( \frac{\partial r}{\partial \theta} \right)^T \frac{\partial^2 \log p(t(i|x(i))}{\partial r^2} \left( \frac{\partial r}{\partial \theta} \right) \right]$$

$$= \mathbb{E}_{x \sim q} \left[ J_r^T (\mathbb{E}_{t \sim \tilde{q}(t|x)} [H_{L_{ort}}]) J_r \right]$$  \hspace{1cm} (9)

The last step of eq. (3) assumes that $(x(i), t(i))$ 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 $H_{L_{ort}}$, implies computing the Hessian of $L$ with respect to $r$, 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 $G_N$ 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 and KSD as being implementations of natural gradient. We make the additional note that Heskes (2000) makes similar algebraic manipulations as the ones provided in this section, however for different reasons. The original contribution of this section is in describing the relationship between Hessian-Free and Krylov Subspace Descent on one side and natural gradient 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.

The last matching activation and error function that we consider is the softmax with cross-entropy.

In the case of sigmoid units with cross-entropy objective ($\sigma$ is the sigmoid function), $H_{\text{cor}}$ is

$$H_{\text{cor},i,i} = \frac{\partial^2 \sum_k (-t_k \log(\sigma(r_k)) - (1-t_k) \log(1-\sigma(r_k)))}{\partial r_i \partial r_i} = \frac{\partial \sigma(r_i) - t_i}{\partial r_i} = 0$$

$$H_{\text{cor},i,j} = \ldots = \frac{\partial \sigma(r_i) - t_i}{\partial r_j} = \sigma(r_i)(1 - \sigma(r_i))$$

If we insert this back into the Gauss-Newton approximation of the Hessian and re-write the equation in terms of $J_r$ instead of $J_r$, we get the corresponding natural gradient metric, equation (7). $d[v]$ stands for the diagonal matrix constructed from the values of the vector $v$.

$$G_N = \frac{1}{n} \sum_{x(i),t(i)} J_r^T H_{\text{cor}} J_r$$

$$= \frac{1}{n} \sum_{x(i)} J_r^T \hat{y}(1 - \hat{y}) \hat{y} \frac{1}{1 - \hat{y}} [y(1 - y)] J_r$$

$$= \mathbb{E}_{x \sim q} [J_r^T \text{diag}(\frac{1}{1 - \hat{y}}) J_r]$$

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 (12).

$$H_{\text{cor},i,j} = \frac{\partial^2 \sum_k (-t_k \log(\phi(r_k)))}{\partial r_i \partial r_j} = \frac{\partial \sum_k (t_k \phi(r_k)) - t_i}{\partial r_j}$$

$$H_{\text{cor},i,j} = \ldots = \frac{\partial \phi(r_i) \phi(r_j)}{\partial r_i} = \phi(r_i) - \phi(r_i) \phi(r_j)$$

Equation (13) starts from the natural gradient metric and singles out a matrix $M$ in the formula such that the metric can be re-written as the product $J_r^T M J_r$ (similar to the formula for the Gauss-Newton approximation). In (14) we show that indeed $M$ equals $H_{\text{cor}}$ and hence the natural gradient metric is the same as the extended Gauss-Newton metric for this case as well. Note that $\delta$ is the Kronecker delta, where $\delta_{ij,i \neq j} = 0$ and $\delta_{ii} = 1$.

$$G = \mathbb{E}_{x \sim q} \left[ \sum_{k=1}^o \frac{1}{y_k} \left( \frac{\partial y_k}{\partial r_i} \right)^T \frac{\partial y_k}{\partial r_j} \right]$$

$$= \mathbb{E}_{x \sim q} \left[ J_r^T \left( \sum_{k=1}^o \frac{1}{y_k} \left( \frac{\partial y_k}{\partial r_i} \right)^T \left( \frac{\partial y_k}{\partial r_j} \right) \right) J_r \right]$$

$$= \frac{1}{N} \sum_{x(i)} (J_r^T M J_r)$$

$$M_{ij} = \sum_{k=1}^o \frac{1}{y_k} \frac{\partial y_k}{\partial r_i} \frac{\partial y_k}{\partial r_j} = \sum_{k=1}^o (\delta_{j,k} - y_k) y_k (\delta_{k,i} - y_k)$$

$$= y_j y_i y_i - y_i y_j - y_i y_j = -\phi(r_i) \phi(r_j)$$

$$M_{ii} = \sum_{k=1}^o \frac{1}{y_k} \frac{\partial y_k}{\partial r_i} y_i = y_i^2 (\sum_{k=1}^o y_k) + y_i - 2y_i^2$$

$$= \phi(r_i) - \phi(r_i) \phi(r_i)$$

There is also a one to one mapping for most of the other heuristics used by Hessian-Free.

Following the functional manifold interpretation of the algorithm, we can recover the Levenberg-Marquardt heuristic used in Martens (2010) to adapt the damping factor by considering a first order Taylor approximation, where for any function $f$,

$$f \left( \theta_t - \eta G^{-1} \frac{\partial f(\theta_t)}{\partial \theta_t} \right) \approx f(\theta_t) - \eta \frac{\partial f(\theta_t)}{\partial \theta_t} G^{-1} \frac{\partial f(\theta_t)}{\partial \theta_t} \quad \text{T}$$
This gives as the reduction ratio given by equation (31), which can be shown to behave identically with the one in Martens (2010).

\[ \rho = f \left( \theta_t - \eta \mathbf{G}^{-1} \frac{\partial f(\theta_t)}{\partial \theta_t}^T \right) - f(\theta_t) \]  

(16)

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

\[
\arg \min_{\Delta \theta} L(\theta + \Delta \theta) \\
\text{s. t. } \mathbb{E}_{x \sim \tilde{q}(x)} \left[ KL(p(\theta_t|x)||p(\theta_t+\Delta \theta|x)) \right] = \text{const.} \\
\text{and } \mathbb{E}_{x \sim \tilde{q}(x)} \left[ KL(p(h|x)||p(h+\Delta h|x)) \right] = \text{const.} \tag{17}
\]

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 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) opts for restricting \( \Delta \theta \) to a lower dimensional Krylov subspace given by \( Gx = \nabla L \) and then, using some other second order method like LBFGS, to solve for \( \Delta \theta \) within this space.

Formally we are looking for \( \gamma_1, \gamma_2, \ldots, \gamma_k \) such that:

\[
\min_{\gamma} L \left( \theta + \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_k \end{bmatrix} \begin{bmatrix} \nabla L \\ \nabla L \mathbf{G} \\ \vdots \\ \nabla L \mathbf{G}^{k-1} \end{bmatrix} \right) \tag{18}
\]

Because we are using the Krylov Subspace, solving for \( \gamma \) is akin to computing \( \nabla L \mathbf{G}^{-1} \) and finding an optimal step size for this direction.

In order to mimic Hessian-Free’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. To show this we can rewrite the new subspace as:

\[
\min L \left( \theta + \beta d_{i-1} + \alpha \begin{bmatrix} \frac{\gamma_1}{\alpha} \\ \frac{\gamma_2}{\alpha} \\ \vdots \\ \frac{\gamma_k}{\alpha} \end{bmatrix} \begin{bmatrix} \nabla L \\ \nabla L \mathbf{G} \\ \vdots \\ \nabla L \mathbf{G}^{k-1} \end{bmatrix} \right) \tag{19}
\]

From this formulation one can see that the previous direction plays a different role than just initializing the linear solver close to a solution.

7 Natural conjugate gradient

As natural gradient can be viewed as a first order method in the space of functions, one could wonder if the algorithm can be improved by considering second order information. Unfortunately computing the Hessian on the manifold can be daunting (especially from a computational perspective). 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.

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 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 $p(\theta)$ 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 $G_{t-1}$, while the geometry around the new direction is defined by $G_t$, where, in principle, $G_{t-1} \neq G_t$. Following Absil et al. (2008) we would need to “transport” $d_{t-1}$ into the space of $\nabla N_t$, 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. (2008) address these issues by making the assumption that $G_{t-1}$ and $G_t$ 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 cannot 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 $p_\theta$. We propose instead to solve for the coefficients $\beta$ and $\alpha$ that minimizes our cost, where $\frac{\partial}{\partial \alpha}$ is the coefficient required in computing the new direction and $\alpha$ is the step size:

$$\min_{\alpha, \beta} \mathcal{L} \left( \theta_{t-1} + \alpha \left[ \begin{array}{c} \alpha_t \\ \beta_t \end{array} \right] \left[ \begin{array}{c} \nabla N_t \\ d_{t-1} \end{array} \right] \right)$$

(20)

The new direction therefore is:

$$d_t = \nabla N_t + \frac{\beta_t}{\alpha_t} d_{t-1}$$

(21)

This is identical to nonlinear conjugate gradient, just that we use the natural direction $\nabla N_t$ instead of the gradients $\nabla_t$ everywhere and we use an off the shelf solver to optimize simultaneously for both the step size $\alpha_t$ and the correction term $\frac{\beta_t}{\alpha_t}$ used to compute the new direction.

This approach gives us an approximation to the right new direction $d_t$, and one can show that in the Euclidian space, given a second order Taylor approximation of $\mathcal{L}(\theta_{t-1})$, under the assumption that the Hessian $H_{t-1}$ is symmetric, this approach will result in exactly conjugate direction.

Let $d_{t-1}$ be the previous direction and $\gamma_{t-1}$ the step size such that $\mathcal{L}(\gamma_{t-1} d_{t-1})$ 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 $\gamma_{t-1}$ we have that:

$$\frac{\partial \mathcal{L}}{\partial \gamma} \gamma_{t-1} d_{t-1} + \gamma_{t-1} d_{t-1}^T H d_{t-1} = 0$$

(22)

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

$$\mathcal{L}(\theta_t + \beta_t d_{t-1} + \alpha_t \nabla N_t)$$

$$= \mathcal{L}(\theta_{t-1} + \gamma_{t-1} d_{t-1} + \beta_t d_{t-1} + \alpha_t \nabla N_t)$$

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

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

Using the previous relation (22) this implies that $(\alpha_t \nabla N_t \mathcal{L}^T + \beta_t d_{t-1})^T H d_{t-1} = 0$, i.e. that the new direction is conjugate to the last one.
In principle this approach is not more correct than employing Polak-Ribiere and ignoring the difference between $G_{t-1}$ and $G_t$. The advantage of this approach however is that we are guaranteed to minimize the cost $L$ that we care about, at each step.

Finally, to respect the spirit of nonlinear conjugate gradients on the manifold, the line search for $\alpha$ should move along the geodesic of the manifold. Both our suggested algorithm and previous work ignore this aspect.

8 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. 1 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 1000 examples, and one in which we use minibatches with around 5000 examples. In both cases all other hyper-parameters were tuned using a grid-search for improving convergence. We use a warm start for linear conjugate gradient in all cases, where the previous solution is first scaled down by multiplying by 0.6. The learning rate is fixed at 0.3 for the model using a minibatch size of 1000 and a line search was used when the minibatch size is 5000. For natural conjugate gradient we use scipy.optimize.fmin_cobyla to solve the two dimensional line search and the metric was computed over 5000 examples. We also forcefully reset natural conjugate gradient every 30 steps, by forcing it to use only the natural direction. For SGD we used a minibatch of 100 examples. More details about hyper-parameters are provided in the supplementary materials. 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 1: The plots depict the training curves on a log-log scale for different learning algorithms on the Curves dataset. Top plot shows the curves in terms of iterations, while the bottom plot show the curves as a function of clock time (seconds). NGD stands for natural gradient descent, the star is used to indicate the run that used minibatches of 5000 examples. NCG is natural conjugate gradient and SGD is minibatch stochastic gradient descent.

The first observation that we can make is that natural gradient can run reliably with small 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. This can be seen from comparing the two runs of natural gradient 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 can be beneficial. We see a speedup in convergence (in terms of iterations). The result agrees with the observation made in (Vinyals and Povey 2012), where Krylov Subspace Descent
(KSD) was shown to converge faster than Hessian-Free. Given the relationship between natural gradient and Hessian-Free, and between KSD and natural conjugate gradient, we expected to see the same difference between natural gradient 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. We think there is room for more exploration for this algorithm. Specifically, NCG moves as far as possible from the current position, making the warm initialization for CG not as helpful. Maybe preconditioning could help to make the algorithm be faster in CPU time as well.

9 Using unlabeled data

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

Furthermore, to compute the natural gradient direction we need to evaluate two different expectations over $x$. The first one is when we evaluate the expected (Euclidean) gradient, while the 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 $G$ by employing unlabeled data. We consider these observations to be important contributions of this paper.

Statistically, if both expectations over $x$ are computed from the same samples, the two estimations are not independent from each other. We hypothesize that this introduces a bias in the direction of natural gradient, bias that makes the algorithm overfit the current minibatch. Figure 2 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 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 $G$ 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. 2 shows the training and test error of a model trained on fold 4 of TFD, though similar results are obtained for the other folds as well.

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 supplementary materials).

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. This is because at each step we overfit the current training minibatch. 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.

10 Robustness to reorderings 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:
Figure 2: The plot depicts the training error (top) on a log scale and test error (bottom) 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.

Figure 3: 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 is order of magnitudes lower than for SGD. See text for more information.

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:
   4. For steps between 1 and 5
   5. Replace segment i by new randomly sampled examples
   6. Train the model from scratch
   7. Evaluate the model on 10K heldout examples
5. Compute the segment i mean variance, among the 5 runs, in the output of the trained model
8. Plot the mean variance as a function of which segment was resampled

Figure 3 shows these curves for minibatch stochastic gradient descent and natural gradient.

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 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 choose hyper-parameters such that after 32.7M samples both methods achieve about the same validation error of 49.8% (see supplementary material for hyper-parameters).

The results are consistent with our hypothesis that natural gradient avoids making large steps in function space during training, staying on the path that induces least variance. Such large steps may be present with SGD, possibly yielding the model to overfit (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 can deal better with nonstationary data and can be less sensitive to the particular examples selected early on during training.

11 Discussion and conclusions

In this paper we made the following original contributions. We showed that by employing the extended Gauss-Newton approximation of the Hessian both Hessian-Free and Krylov Subspace Descent can be interpreted as implementing natural gradient. 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 G 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 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 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 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 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

We would like to thank the developers of Theano [Bergstra et al., 2010b; Bastien et al., 2012]. We would also like to thank NSERC, Compute Canada, and Calcul Québec for providing computational resources. Razvan Pascanu is supported by DeepMind Fellowship.

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 *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. J., Bergeron, A., Bouchard, N., and Bengio, Y. (2012). Theano: new features and speed improvements. 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. (2010). Theano: a CPU and GPU math expression compiler. In Proceedings of the Python for Scientific Computing Conference (SciPy). Oral Presentation.

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.

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.

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.

Sutskever, I., Martens, J., and Hinton, G. E. (2011). Generating text with recurrent neural networks. In ICML, pages 1017–1024.

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

Expected Hessian to Fisher Information Matrix

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

\[
E_{\mathbf{x}} \left[ -\frac{\partial^2 \log p_\theta}{\partial \theta^2} \right] = \mathbb{E}_{\mathbf{z}} \left[ \frac{\partial}{\partial \theta} \left( \frac{1}{p_\theta} \frac{\partial p_\theta}{\partial \theta} \right) \right] = \mathbb{E}_{\mathbf{z}} \left[ -\frac{1}{p_\theta(z)} \frac{\partial^2 p_\theta}{\partial \theta^2} + \left( \frac{1}{p_\theta} \frac{\partial p_\theta}{\partial \theta} \right)^T \left( \frac{1}{p_\theta} \frac{\partial p_\theta}{\partial \theta} \right) \right] \\
= -\frac{\partial^2}{\partial \theta^2} \left( \sum_x p_\theta(z) \right) + \mathbb{E}_{\mathbf{z}} \left[ \left( \frac{\partial \log p_\theta(z)}{\partial \theta} \right)^T \left( \frac{\partial \log p_\theta(z)}{\partial \theta} \right) \right] \\
= \mathbb{E}_{\mathbf{z}} \left[ \left( \frac{\partial \log p_\theta(z)}{\partial \theta} \right)^T \left( \frac{\partial \log p_\theta(z)}{\partial \theta} \right) \right] 
\]

(23)

Derivation of the natural gradient metrics

Linear activation function

In the case of linear outputs we assume that each entry of the vector \( t \) comes from a Gaussian distribution centered around \( y_i(x) \) with some standard deviation \( \beta \). From this it follows that:

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

(24)

\[
G = \mathbb{E}_{\mathbf{x} \sim \tilde{q}} \left[ \sum_{i=1}^{o} \left( \frac{\partial \log p(t_i|y(x)_i)}{\partial \theta} \right)^T \left( \frac{\partial \log p_\theta(t_i|y(x)_i)}{\partial \theta} \right) \right] \\
= \mathbb{E}_{\mathbf{x} \sim \tilde{q}} \sum_{i=1}^{o} \left[ \mathbb{E}_{t \sim \mathcal{N}(t|y(x)_i, \beta^2)} \left[ \left( \frac{\partial \log p(t_i|y(x)_i)}{\partial \theta} \right)^T \left( \frac{\partial \log p(t_i|y(x)_i)}{\partial \theta} \right) \right] \right] \\
= \mathbb{E}_{\mathbf{x} \sim \tilde{q}} \sum_{i=1}^{o} \left[ \mathbb{E}_{t \sim \mathcal{N}(t|y(x)_i, \beta^2)} \left[ \left( \frac{\partial (t_i - y_i)^2}{\partial \theta} \right)^T \left( \frac{\partial (t_i - y_i)^2}{\partial \theta} \right) \right] \right] \\
= \mathbb{E}_{\mathbf{x} \sim \tilde{q}} \sum_{i=1}^{o} \left[ \mathbb{E}_{t \sim \mathcal{N}(t|y(x)_i, \beta^2)} \left[ (t_i - y_i)^2 \left( \frac{\partial y_i}{\partial \theta} \right)^T \left( \frac{\partial y_i}{\partial \theta} \right) \right] \right] \\
= \mathbb{E}_{\mathbf{x} \sim \tilde{q}} \sum_{i=1}^{o} \left[ \mathbb{E}_{t \sim \mathcal{N}(t|y(x)_i, \beta^2)} \left[ (t_i - y_i)^2 \left( \frac{\partial \gamma_i}{\partial \theta} \right)^T \left( \frac{\partial \gamma_i}{\partial \theta} \right) \right] \right] \\
= \mathbb{E}_{\mathbf{x} \sim \tilde{q}} \sum_{i=1}^{o} \left[ \left( \frac{\partial y_i}{\partial \theta} \right)^T \left( \frac{\partial y_i}{\partial \theta} \right) \right] \\
= \beta^2 \mathbb{E}_{\mathbf{x} \sim \tilde{q}} \left[ J_y^T J_y \right] \\
= \beta^2 G_{\mathbf{x} \sim \tilde{q}} \left[ J_y^T J_y \right]
\]

(25)

Sigmoid activation function

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

\[
p(t|x) = \prod_i y_i^t_i (1 - y_i)^{1-t_i} 
\]

(26)

\( \log p \) gives us the usual cross-entropy error used with sigmoid units. We can compute the Fisher information matrix as follows:
\[ G = \mathbb{E}_{x \sim q} \left[ \mathbb{E}_{t \sim p(t|x)} \left[ \sum_{i=1}^{o} \left( \frac{(t_i - y_i)^2}{y_i(1-y_i)^2} \left( \frac{\partial y_i}{\partial \theta} \right)^T \frac{\partial y_i}{\partial \theta} \right) \right] \right] \]

\[ = \mathbb{E}_{x \sim q} \sum_{i=1}^{o} \frac{1}{y_i(1-y_i)} \left( \frac{\partial y_i}{\partial \theta} \right)^T \frac{\partial y_i}{\partial \theta} \]

\[ = \mathbb{E}_{x \sim q} \left[ J_y^T \text{diag} \left( \frac{1}{y(1-y)} \right) J_y \right] \]

\[ (27) \]

Softmax activation function

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

\[ p(t|x) = \prod_{i} y_i^{t_i} \]

\[ (28) \]

\[ G = \mathbb{E}_{x \sim q} \left[ \sum_{i} \frac{1}{y_i} \left( \frac{\partial y_i}{\partial \theta} \right)^T \frac{\partial y_i}{\partial \theta} \right] \]

\[ (29) \]

Implementation Details

We have implemented natural gradient 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:

```python
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) is given below.

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 extend 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.
Algorithm 1 Pseudocode for natural gradient algorithm

# 'gfn' is a function that computes the metric times some vector

gfn ← (lambda v → Gv)

while not early_stopping_condition do

g ← ∂L/∂θ

# linear_cg solves the linear system Gx = ∂L/∂θ

ng ← linear_cg(gfn, g, max_iters = 20, rtol=1e-4)

# γ is the learning rate

θ ← θ − γ ng

end while

Even though we are ensured that G 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 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 (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] by considering a first order Taylor approximation, where for any function f,

\[ f(\theta_t - \eta G^{-1} \frac{\partial f(\theta_t)}{\partial \theta_t}^T) \approx f(\theta_t) - \eta \frac{\partial f(\theta_t)}{\partial \theta_t} G^{-1} \frac{\partial f(\theta_t)}{\partial \theta_t}^T \]  

(30)

This gives as the reduction ratio given by equation (31) which can be shown to behave identically with the one in Martens (2010).

\[ \rho = \frac{f(\theta_t - \eta G^{-1} \frac{\partial f(\theta_t)}{\partial \theta_t}^T) - f(\theta_t)}{-\eta \frac{\partial f(\theta_t)}{\partial \theta_t} G^{-1} \frac{\partial f(\theta_t)}{\partial \theta_t}^T} \]  

(31)

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 over minibatches of 512 examples. In the case of natural gradient 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 and 0.1 for MSGD.