MA4270 Project – Natural Gradients

Ang Ming Liang A0197132E

November 6, 2020

1 Introduction

A common activity in machine learning is optimisation. For instance, finding the linear classifier with the largest margins requires maximizing margins of our linear classifier. Another example, and the motivating example of this report, is the maximum likelihood estimate which is found by maximising the likelihood function. An optimisation algorithm that was seen in class for such optimisation problems is gradient descent or steepest descent. The idea of gradient descent is to move "downhill" in the steepest direction (i.e. opposite to the direction of the gradient) to quickly minimise the function of interest. Although gradient descent is commonly used in practice, there are several issues with such gradient-based or first-order optimisation methods that only rely on gradient information: slow convergence and local minimums. There have been many algorithms proposed for augmenting gradient descent in an attempt to address or mitigate some of the issues with the method. However, this report focuses only on the natural gradient descent method which incorporates information about the underlying geometry of the parameters to make gradient descent converge faster.

2 Description of Natural Gradients

2.1 Preliminaries: Maximum Likelihood Estimation

Following the lecture notes, we consider the general classification problem ($y \in \{1, 2, ..., m\}$) of learning a probabilistic classifier $P(y_t|\mathbf{x}_t; \boldsymbol{\theta}; \theta_0)$ from a sample of n independently and identically distributed (I.I.D) data points $\mathcal{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^n$, where $\mathbf{x}_t$ is a d-dimensional input vector, such that the resulting learned classifier gives us a reasonable probability a particular point $\mathbf{x}_t$ has the label y_t. In our setup in particular the model parameters $(\boldsymbol{\theta}, \theta_0)$ is found using maximum likelihood estimation by finding $(\boldsymbol{\theta}, \theta_0)$ that maximises the likelihood function eq (1).

$$L(\boldsymbol{\theta}; \theta_0|D) = \prod_{t=1}^{n} P(y_t|\mathbf{x}_t; \boldsymbol{\theta}, \theta_0) \tag{1}$$

Intuitively, this is like choosing $(\boldsymbol{\theta}, \theta_0)$ that "best explains" the data. However, if we are applying gradient descent directly on this function, taking the derivatives of the likelihood function directly is a messy process because of the product rule. In order to simplify calculation when we take the derivative of the likelihood

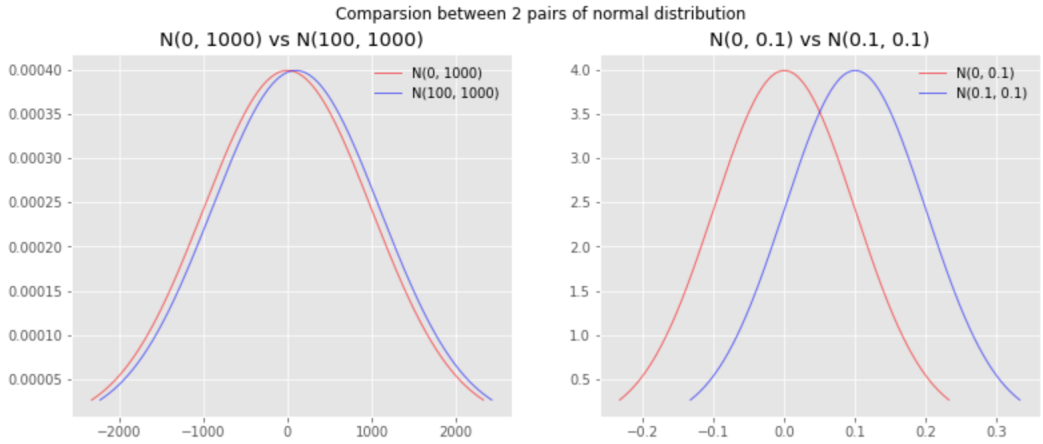

Figure 1: Comparison between the pair N(0,1000) and N(100,1000) and the pair N(0,0.1) and N(0.1,0.1)

function, we can maximise $\log L$ instead of L, so we get

$$(\hat{\boldsymbol{\theta}}, \hat{\theta_0}) = \arg\max_{\boldsymbol{\theta}, \theta_0} \sum_{t=1}^{n} \log P(y_t|\mathbf{x}_t; \boldsymbol{\theta}, \theta_0) = \arg\min_{\boldsymbol{\theta}, \theta_0} \sum_{t=1}^{n} -\log P(y_t|\mathbf{x}_t; \boldsymbol{\theta}, \theta_0) \tag{2}$$

The resulting function

$$L(\boldsymbol{\theta}, \theta_0) = -\sum_{t=1}^{n} \log P(y_t|\mathbf{x}_t; \boldsymbol{\theta}, \theta_0) \tag{3}$$

is known as the negative log-likelihood. Therefore, to find the maximum likelihood estimate of $(\boldsymbol{\theta}, \theta_0)$ we just need to find the parameters $(\boldsymbol{\theta}, \theta_0)$ that minimises the negative log-likelihood of the function.

2.2 Natural Gradients

As we have mentioned in our introduction, one common way to solve this optimisation problem is to use gradient descent eq (4).

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \gamma \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(i)}, \theta_0) \tag{4}$$

which updates our current model parameter $\boldsymbol{\theta}$ by the gradient of the negative log-likelihood likelihood function or, equivalently, the direction of steepest descent by picking a vector d such that the new parameter $\boldsymbol{\theta} + d$ that is within the local ϵ-neighbourhood of $\boldsymbol{\theta}$ minimises the negative log-likelihood or more formally

$$\frac{-\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \theta_0)}{||\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \theta_0)||} = \lim_{\epsilon \to 0} \frac{1}{\epsilon} \arg\min_{||d||^2 \leq \epsilon} L(\boldsymbol{\theta} + d, \theta_0) \tag{5}$$

where the local neighbourhood is expressed by the means of Euclidean norm/distance. However, this leads to gradient descent moving suboptimaly because Euclidean distance is not a good metric for how changes of parameters $(\boldsymbol{\theta}; \theta_0)$ changes the probability distribution $P(y_t|\mathbf{x}_t; \boldsymbol{\theta}; \theta_0)$. This is illustrated by considering the two pairs of distributions N(0,1000), N(100,1000) and N(0,0.1), N(0.1,0.1) as shown in figure (1). Even though the Euclidean distance between the parameters of the first pair (euclidean distance is 100) is larger

than the second pair (euclidean distance is 0.1), the first pair of distribution has a smaller KL divergence than the second pair both i.e the first pair of distributions is more similar to each other than the second pair. The reason for this is because probability distributions do not naturally live in Euclidean space but rather on a statistical manifold. Hence, if we take the steepest descent on this statistical manifold instead of Euclidean space we will get a more efficient optimiser for minimising the negative log-likelihood, this is the key idea of natural gradient descent. This has also the added advantage of making the optimiser more robust to different reparametrization of the model as the optimiser only cares about the distribution induced by the model parameters, leading to more effective optimisation.[1] [1].

All that sounds great, but how do we take the steepest descent on the statistical manifold ? To answer that, let us first consider the Euclidean space P such that the point $p \in P \subset \mathbb{R}^n$. In this simple case, the squared distance between two points p and $p + dp$ is given by the usual Euclidean inner product

$$||dp||^2 = \langle dp, dp \rangle \tag{6}$$

However, for a manifold P the previous distance is now given by the bilinear form

$$||dp||_P^2 = \langle dp, \mathbb{G}(p)dp \rangle = \sum_{ij} g_{ij}(p) dp_i dp_j \tag{7}$$

where the matrix $\mathbb{G}(p) = [g_{ij}(p)]$ is called the Riemannian metric tensor. The Riemannian metric tensor is an inner product on the tangent space of the manifold P at the point p allowing us to define a distances on the manifold. [2] As a concrete example to illustrate what that means, consider two ants standing on top of two different ant hills. If one of the ants can fly then the distance they would fly to the other ant hill is the Euclidean distance. If the other ant cannot fly it would need to walk on the surface of the Earth. The Riemannian metric tensor tells us how far this ant must walk on the earth given the distance the other ant flew i.e the Euclidean distance. [3] The Riemannian metric tensor, thus, describes how the geometry of a manifold affects a differential patch, dp, at the point p.

Hence, by defining the ϵ-neighbourhood using the norm of the manifold $|| \cdot ||_P$ instead of the Euclidean norm. We find that the steepest descent direction d for a general loss function $L(\boldsymbol{\theta}, \theta_0)$ on a manifold is

$$d = -\frac{1}{2\lambda} \mathbb{G}^{-1}(\boldsymbol{\theta}) \nabla L(\boldsymbol{\theta}, \theta_0) \tag{8}$$

as proven in theorem 1.

Theorem 1. *The steepest descent direction d of a general loss function $L(\boldsymbol{\theta}, \theta_0)$, in other words the direction d such that*

$$d = \underset{||d||_P^2 = \epsilon}{\arg\min} L(\boldsymbol{\theta} + d, \theta_0) \tag{9}$$

[1] Take note that in this report we only consider $\boldsymbol{\theta}$ as the arguments presented for $\boldsymbol{\theta}$ can similarly be applied to θ_0

[2] Note that G depends on the location of the point p

[3] Don't take this illustration too seriously since technically all of this needs to take place in a differential patch (a small rectangle whose side lengths go to 0).

in a manifold P for a small ϵ constant, is given by

$$d = -\frac{1}{2\lambda}\mathbb{G}^{-1}(\boldsymbol{\theta})\nabla L(\boldsymbol{\theta}, \theta_0) \tag{10}$$

where λ is a real number, $\mathbb{G}^{-1}$ is the inverse of the Riemannian metric tensor $\mathbb{G}$ and the ∇L is the conventional gradient vector [1].

Proof. Using Lagrange multipliers we can rewrite the constraint optimisation problem in equation (9) as

$$\begin{aligned} d &= \arg\min_d L(\boldsymbol{\theta} + d, \theta_0) + \lambda(||d||^2 - \epsilon) \\ &= \arg\min_d L(\boldsymbol{\theta} + d, \theta_0) + \lambda(d^T\mathbb{G}(\boldsymbol{\theta})d - \epsilon^2) \end{aligned}$$

Approximating $L(\boldsymbol{\theta} + d, \theta_0)$ as

$$L(\boldsymbol{\theta} + d, \theta_0) \approx L(\boldsymbol{\theta}, \theta_0) + \nabla L(\boldsymbol{\theta}, \theta_0)^T d$$

we can rewrite our original optimisation problem (9) as the following

$$d = \arg\min_d L(\boldsymbol{\theta}, \theta_0) + \nabla L(\boldsymbol{\theta}, \theta_0)^T d + \lambda(d^T\mathbb{G}(\boldsymbol{\theta})d - \epsilon^2)$$

Taking the derivatives with respect to d of

$$L(\boldsymbol{\theta}, \theta_0) + \nabla L(\boldsymbol{\theta}, \theta_0)^T d + \lambda(d^T\mathbb{G}(\boldsymbol{\theta})d - \epsilon^2)$$

and setting it to zero we obtain

$$\nabla L(\boldsymbol{\theta}, \theta_0) + 2\lambda\mathbb{G}(\boldsymbol{\theta})d = 0$$

Therefore,

$$d = -\frac{1}{2\lambda}\mathbb{G}^{-1}(\boldsymbol{\theta})\nabla L(\boldsymbol{\theta}, \theta_0)$$

□

This leads to the question, what is $\mathbb{G}(\boldsymbol{\theta})$ for our statistical manifold ? For the case of our statistical manifold P , $\mathbb{G}(\boldsymbol{\theta})$ is the Fisher information matrix[4] if its local distance metric is the symmetrized Kullback-Leibler (KL) divergence

$$\mathrm{KL}_{sum}(f_1, f_2) = \frac{1}{2}(\mathrm{KL}(f_1||f_2) + \mathrm{KL}(f_2||f_1)) \tag{11}$$

This arises from the fact that for small values of d the second order Taylor series expansion of the KL divergence is $\frac{1}{2}d^T\mathbb{F}d$ i.e. $\mathrm{KL}[p(x|\boldsymbol{\theta})||p(x|\boldsymbol{\theta} + d)] \approx \frac{1}{2}d^T\mathbb{F}d$[5] or more rigorously from placing an inner product on the statistical manifold of log-probability densities [10].

[4] [12] contains a list implementation for the Fisher Information Matrix of several common activation functions used in neural networks

[5]The proof is in the appendix

Corollary 1. *The steepest descent direction d of $L(\boldsymbol{\theta}, \theta_0)$ in the space of probability distributions is*

$$d = -\frac{1}{2\lambda}\mathbb{F}^{-1}(\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \theta_0) \tag{12}$$

where $\mathbb{F}^{-1}$ is the inverse Fisher information matrix.

Hence, we have the following algorithm

Algorithm 1: Natural Gradient Descent

initialization;

while *not early stopping condition* **do**

1. Compute forward pass on our model and compute loss $L(\boldsymbol{\theta}^{(i)}, \theta_0)$
2. Compute the gradient $\nabla_{\boldsymbol{\theta}^{(i)}} L(\boldsymbol{\theta}^{(i)}, \theta_0)$
3. Compute the Fisher Information Matrix $\mathbb{F}(\boldsymbol{\theta}^{(i)})$
4. Compute $\mathbb{F}^{-1}(\boldsymbol{\theta}^{(i)})\nabla_{\boldsymbol{\theta}^{(i)}} L(\boldsymbol{\theta}^{(i)}, \theta_0)$
5. Update the parameter: $\boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)} - \alpha\mathbb{F}^{-1}(\boldsymbol{\theta}^{(i)})\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(i)}, \theta_0)$

end

where $\mathbb{F}^{-1}(\boldsymbol{\theta}^{(i)})\nabla_{\boldsymbol{\theta}^{(i)}} L(\boldsymbol{\theta}^{(i)}, \theta_0)$ is known as the natural gradient and the $\frac{1}{2\lambda}$ becomes part of the learning rate α choosen at the start.

2.3 Computing Fisher Information Matrix

However, how do we compute the Fisher Information Matrix $\mathbb{F}$? It turns out that the negative expected Hessian of our model log likelihood is equal to the Fisher Information Matrix $\mathbb{F}$.

Proposition 1. *The Fisher Information Matrix $\mathbb{F}$ is the covarience of $\nabla_{\boldsymbol{\theta}} \log p(x|\boldsymbol{\theta}, \theta_0))$ thus,*

$$\mathbb{F} = \mathop{\mathbb{E}}_{p(x|\boldsymbol{\theta},\theta_0)}[\nabla_{\boldsymbol{\theta}} \log p(x|\boldsymbol{\theta}, \theta_0)\nabla_{\boldsymbol{\theta}} \log p(x|\boldsymbol{\theta}, \theta_0)^T] \tag{13}$$

Proof. Observe that

$$\begin{aligned}
\mathop{\mathbb{E}}_{p(x|\boldsymbol{\theta},\theta_0)}[\nabla_{\boldsymbol{\theta}} \log p(x|\boldsymbol{\theta}, \theta_0)] &= \int \nabla_{\boldsymbol{\theta}} \log p(x|\boldsymbol{\theta}, \theta_0) p(x|\boldsymbol{\theta}, \theta_0) dx \\
&= \int \frac{\nabla_{\boldsymbol{\theta}} p(x|\boldsymbol{\theta}, \theta_0)}{p(x|\boldsymbol{\theta}, \theta_0)} p(x|\boldsymbol{\theta}, \theta_0) dx \\
&= \int \nabla_{\boldsymbol{\theta}} p(x|\boldsymbol{\theta}, \theta_0) dx \\
&= \nabla_{\boldsymbol{\theta}} \int p(x|\boldsymbol{\theta}, \theta_0) dx \\
&= \nabla_{\boldsymbol{\theta}} 1 \\
&= 0
\end{aligned}$$

Since $\mathbb{E}_{p(x|\boldsymbol{\theta},\theta_0)}[\nabla \log p(x|\boldsymbol{\theta})] = 0$, the Fisher Information Matrix, the covarience of $\nabla_{\boldsymbol{\theta}} \log p(x|\boldsymbol{\theta}, \theta_0)$,

$$\begin{aligned}
\mathbb{F} &= \mathop{\mathbb{E}}_{p(x|\boldsymbol{\theta},\theta_0)}[(s(\boldsymbol{\theta},\theta_0) - \mathop{\mathbb{E}}_{p(x|\boldsymbol{\theta},\theta_0)}[s(\boldsymbol{\theta},\theta_0)])(s(\boldsymbol{\theta},\theta_0) - \mathop{\mathbb{E}}_{p(x|\boldsymbol{\theta},\theta_0)}[s(\boldsymbol{\theta},\theta_0)])^T] \\
&= \mathop{\mathbb{E}}_{p(x|\boldsymbol{\theta},\theta_0)}[(s(\boldsymbol{\theta},\theta_0)(s(\boldsymbol{\theta},\theta_0))^T] \\
&= \mathop{\mathbb{E}}_{p(x|\boldsymbol{\theta},\theta_0)}[\nabla_{\boldsymbol{\theta}} \log p(x|\boldsymbol{\theta},\theta_0) \nabla_{\boldsymbol{\theta}} \log p(x|\boldsymbol{\theta},\theta_0)^T]
\end{aligned}$$

where $s(\boldsymbol{\theta}, \theta_0) = \nabla_{\boldsymbol{\theta}} \log p(x|\boldsymbol{\theta}, \theta_0)$. □

Theorem 2. *The negative expected Hessian of log likelihood $\mathbb{H}_{\log p(x|\boldsymbol{\theta},\theta_0)}$ is equal to the Fisher Information matrix $\mathbb{F}$*

$$\mathbb{F} = -\mathop{\mathbb{E}}_{p(x|\boldsymbol{\theta},\theta_0)}[\mathbb{H}_{\log p(x|\boldsymbol{\theta},\theta_0)}] \tag{14}$$

Proof. Notice that,

$$\begin{aligned}
\mathbb{H}_{\log p(x|\boldsymbol{\theta},\theta_0)} &= J\left(\frac{\nabla_{\boldsymbol{\theta}} \log p(x|\boldsymbol{\theta},\theta_0)}{p(x|\boldsymbol{\theta},\theta_0)}\right) \\
&= \frac{\mathbb{H}_{p(x|\boldsymbol{\theta},\theta_0)} - \nabla_{\boldsymbol{\theta}} p(x|\boldsymbol{\theta},\theta_0) \nabla_{\boldsymbol{\theta}} p(x|\boldsymbol{\theta},\theta_0)^T}{p(x|\boldsymbol{\theta},\theta_0) p(x|\boldsymbol{\theta},\theta_0)} \\
&= \frac{\mathbb{H}_{p(x|\boldsymbol{\theta},\theta_0)}}{p(x|\boldsymbol{\theta},\theta_0)} - \left(\frac{\nabla_{\boldsymbol{\theta}} p(x|\boldsymbol{\theta},\theta_0)}{p(x|\boldsymbol{\theta},\theta_0)}\right)\left(\frac{\nabla_{\boldsymbol{\theta}} p(x|\boldsymbol{\theta},\theta_0)}{p(x|\boldsymbol{\theta},\theta_0)}\right)^T
\end{aligned}$$

Hence,

$$\begin{aligned}
\mathop{\mathbb{E}}_{p(x|\boldsymbol{\theta},\theta_0)}[\mathbb{H}_{\log p(x|\boldsymbol{\theta},\theta_0)}] &= \mathop{\mathbb{E}}_{p(x|\boldsymbol{\theta},\theta_0)}\left[\frac{\mathbb{H}_{p(x|\boldsymbol{\theta},\theta_0)}}{p(x|\boldsymbol{\theta},\theta_0)} - \left(\frac{\nabla_{\boldsymbol{\theta}} p(x|\boldsymbol{\theta},\theta_0)}{p(x|\boldsymbol{\theta},\theta_0)}\right)\left(\frac{\nabla_{\boldsymbol{\theta}} p(x|\boldsymbol{\theta},\theta_0)}{p(x|\boldsymbol{\theta},\theta_0)}\right)^T\right] \\
&= \mathop{\mathbb{E}}_{p(x|\boldsymbol{\theta},\theta_0)}\left[\frac{\mathbb{H}_{p(x|\boldsymbol{\theta},\theta_0)}}{p(x|\boldsymbol{\theta},\theta_0)}\right] - \mathop{\mathbb{E}}_{p(x|\boldsymbol{\theta},\theta_0)}\left[\left(\frac{\nabla_{\boldsymbol{\theta}} p(x|\boldsymbol{\theta},\theta_0)}{p(x|\boldsymbol{\theta},\theta_0)}\right)\left(\frac{\nabla_{\boldsymbol{\theta}} p(x|\boldsymbol{\theta},\theta_0)}{p(x|\boldsymbol{\theta},\theta_0)}\right)^T\right] \\
&= \int \frac{\mathbb{H}_{p(x|\boldsymbol{\theta},\theta_0)}}{p(x|\boldsymbol{\theta},\theta_0)} p(x|\boldsymbol{\theta},\theta_0) dx - \mathop{\mathbb{E}}_{p(x|\boldsymbol{\theta},\theta_0)}\left[\nabla_{\boldsymbol{\theta}} \log p(x|\boldsymbol{\theta},\theta_0) \nabla_{\boldsymbol{\theta}} \log p(x|\boldsymbol{\theta},\theta_0)^T\right] \\
&= \mathbb{H}_{\int p(x|\boldsymbol{\theta},\theta_0) dx} - F \\
&= \mathbb{H}_1 - F \\
&= -F
\end{aligned}$$

Therefore,

$$\mathbb{F} = -\mathop{\mathbb{E}}_{p(x|\boldsymbol{\theta},\theta_0)}[\mathbb{H}_{\log p(x|\boldsymbol{\theta},\theta_0)}]$$

□

Thus, to compute the Fisher Information Matrix for natural gradient descent we need to compute the negative expected Hessian of log likelihood $\mathbb{H}_{\log p(x|\boldsymbol{\theta},\theta_0)}$. This interpretation of $\mathbb{F}$ as the expected Hessian also means that, in the same way second order methods do, natural gradient descent can jump over plateaus

of $p(x|\boldsymbol{\theta}, \theta_0)$ [12]. This gives additional insight as to why natural gradient descent is found in practice to converge in faster than traditional gradient descent.

2.4 Experiments

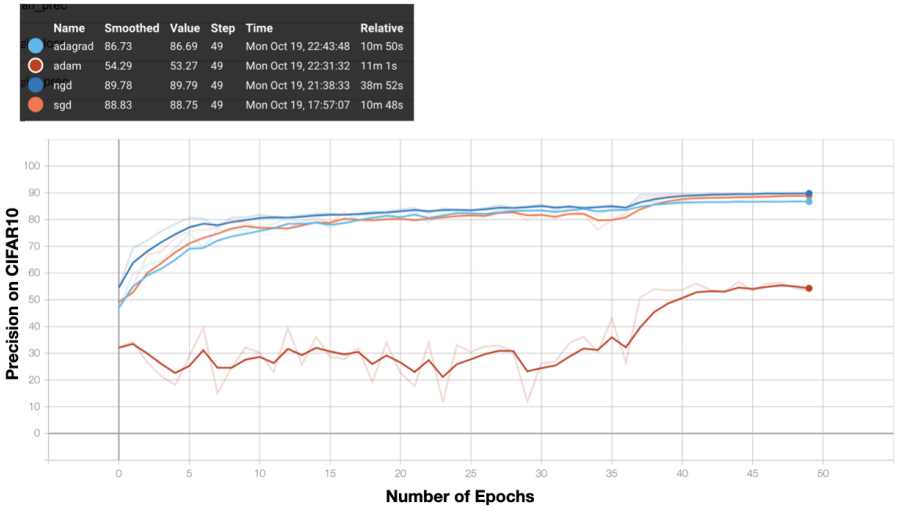

Figure 2: Plot of Densenet-19 [4] precision score on CIFAR10 validation dataset over 50 epochs

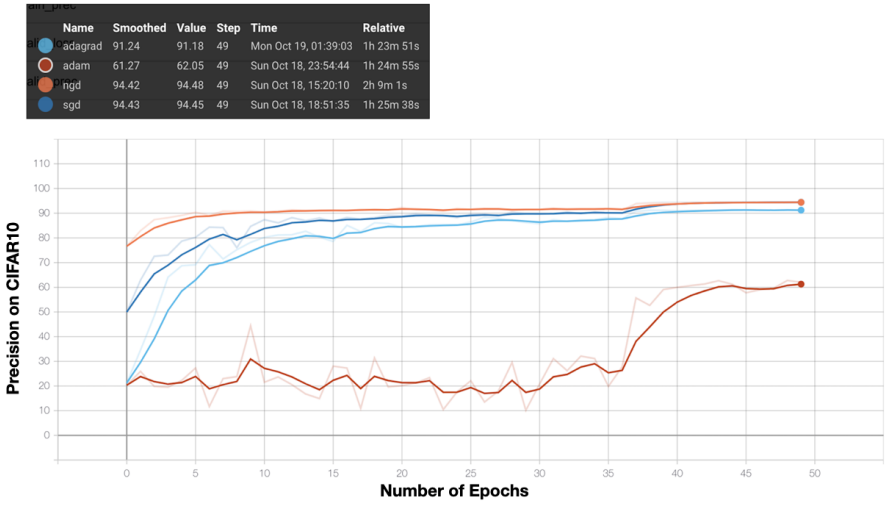

Figure 3: Plot of Wide Resnet [17] precision score on CIFAR10 validation dataset over 50 epochs

This is supported by the empirical experiments we ran on densenet-19 and wide resent, both widely used model for image recognition, on the CIFAR10 dataset. The results[6][7] shows that natural gradients does

[6]The lines on the graph is the moving average (smoothen line) rather than the actual data to account for the nosiness of the training procedure.The actual data are the faded lines on the graph.

[7]The step is 49 even though there are 50 epochs because the first epoch is counted as 0 in TensorBoard.

indeed converge in fewer steps than stochastic gradient descent and even other commonly used optimisation algorithms. However, the run time is longer because during each step we need to compute the inverse Fisher Information matrix which is an expensive operation. In practice for some models, it is sufficient to just only use the diagonals of the Fisher Information matrix as an approximation for the Fisher Information matrix as what is done for second-order methods like ADAM [6]. [8] This reduces the time and memory complexity for inverting the Fisher Information matrix in each step to $O(n)$ instead of the original time complexity of $O(n^3)$ and memory complexity of $O(n^2)$. However, for more complex models this approximation might not hold or have other issues which result in a decrease in precision and other performance metrics of the model as is seen in Figure 2 and 3. Solving these issues has been the topic of recent papers proposing possible improvements to the ADAM algorithm [8] [16].

3 Extensions and Further Results

One interesting extention of natural gradients is the correspondence between natural gradient descent and mirror gradient descent. Mirror descent is a generalisation of gradient descent developed by Nemirovski and Yudin [11]

$$\boldsymbol{\theta}^{(i+1)} = \arg\min_{\boldsymbol{\theta}} \left\{ \left\langle \boldsymbol{\theta}, \nabla f(\boldsymbol{\theta}^{(i)}) \right\rangle + \frac{1}{2\alpha} ||\boldsymbol{\theta} - \boldsymbol{\theta}^{(i)}||^2 \right\} \tag{15}$$

Mirror gradient descent can be viewed as a trust region problem where the original objective function f is approximated using the first-order approximation (eq 16) [3]

$$f(\boldsymbol{\theta}) \approx f(\boldsymbol{\theta}^{(i)}) + \left\langle \nabla f(\boldsymbol{\theta}^{(i)}), \boldsymbol{\theta} - \boldsymbol{\theta}^{(i)} \right\rangle \tag{16}$$

which is only accurate near some point $\boldsymbol{\theta}^*$ (i.e within the ϵ-neighbourhood of $\boldsymbol{\theta}^*$). In which the solution to the following trust region problem [9]

$$\begin{aligned} \min_{\theta} \quad & \left\langle \boldsymbol{\theta}, \nabla f(\boldsymbol{\theta}^{(i)}) \right\rangle \\ \text{subject to} \quad & ||\boldsymbol{\theta} - \boldsymbol{\theta}^{(i)}||^2 \leq \epsilon \end{aligned}$$

then gives a general direction in some larger iterative procedure for minimizing the objective function f [10].

More generally the quadratic penalty in mirror descent can be replaced with any proximity function, a common proximity function is the Bergman divergence since it corresponds to the KL-divergenes for different exponential families

$$\boldsymbol{\theta}^{(i+1)} = \arg\min_{\boldsymbol{\theta}} \left\{ \left\langle \boldsymbol{\theta}, \nabla f(\boldsymbol{\theta}^{(i)}) \right\rangle + \frac{1}{2\alpha} B_G(\theta, \theta^{(i)}) \right\} \tag{18}$$

[8] ADAM is currently on the of the defacto standard optimisation algorithms for deep learning algorithms and is widely implemented in many commercial deep learning libraries such as Tensorflow and Pytorch

[9] The expressions $f(\boldsymbol{\theta}^{(i)})$ and $\left\langle \nabla f(\boldsymbol{\theta}^{(i)}), \boldsymbol{\theta}^{(i)} \right\rangle$ are ignored because they are constants with respect to $\boldsymbol{\theta}$.

[10] Observe that the Lagrange multipliers of the following optimisation problem is exactly the same as equation 15 where $\frac{1}{2\alpha}$ represents the Lagrange multiplier. The term $\frac{\epsilon}{2\alpha}$ is ignored because it is a constant with respect to $\boldsymbol{\theta}$

It turns out that the mirror descent step (18) with the Bergman divergence defined by G ,a strictly twice-differentiable function, is equivalent to the natural gradient step along the dual Riemannian manifold [14]. This gives a powerful interpretation of natural gradients as a kind of trust region optimisation. One insight that can be gain from this interpretation is that the KL constraint typically used in natural gradient ensures that the model does not change by more than epsilon. This provides some kind of robustness to overfitting as the model is not allowed to move too far in some direction d if moving along d changes the density computed by the model substantially [12]. This interpretation of natural gradients also has interesting application with modern policy gradient methods in reinforcement learning such as trust region policy optimisation (TRPO) [15] which is a more general case of natural policy gradient [11] [5].

4 Conclusion

In conclusion, we see that natural gradients present an easy to implement but unfortunately computationally intensive method for optimising machine learning models. They also have additional theoretical advantages such as being invariant to reparameterisation which makes them more robust and efficient compared to standard gradient based optimisation algorithms. However, with the current trend of ever larger and deeper models in recent years, where model sizes can be as big as 175 billion parameters [12], the computational cost outweigh the theoretical advantages enjoyed by natural gradients compared to other standard gradient based optimisers use in practice. Furthermore, these theoretical advantages may not always hold in reality [9]. Nonetheless, natural gradients remains a fruitful avenue of research as it is connected to many areas of machine learning while grounded in interesting mathematics like Riemannian geometry [1] and even optimal transport [7]. This makes it an interesting avenue of research for the foreseeable future.

[11] Natural gradient descent but applied to find good policies for a particular Markov Decision Process (MDP) instead of finding model parameters

[12] The GPT-3 model has a 175 billion parameters

Appendix

Proof of second order Talyor expansion of KL divergence

Proposition 2. *For small values of d the second order Taylor series expansion of the KL divergence is* $KL[p(x|\boldsymbol{\theta})||p(x|\boldsymbol{\theta}+d)] \approx \frac{1}{2}d^T\mathbb{F}d$

Proof. The second order Talyor expansion of KL divergence is

$$\begin{aligned}\mathrm{KL}[p(x|\boldsymbol{\theta})||p(x|\boldsymbol{\theta}+d)] &\approx \mathrm{KL}[p(x|\boldsymbol{\theta})||p(x|\boldsymbol{\theta})] + (\nabla_{\theta'}\mathrm{KL}[p(x|\boldsymbol{\theta})||p(x|\boldsymbol{\theta}')]|_{\boldsymbol{\theta}'=\boldsymbol{\theta}})^T d + \frac{1}{2}d^T\mathbb{F}d \\ &= \mathrm{KL}[p(x|\boldsymbol{\theta})||p(x|\boldsymbol{\theta})] - \mathbb{E}[\nabla_\theta \log p(x|\boldsymbol{\theta})]^T d + \frac{1}{2}d^T\mathbb{F}d\end{aligned}$$

Since $\mathbb{E}[\nabla_\theta \log p(x|\boldsymbol{\theta})] = 0$ by proposition 1 and $\mathrm{KL}[p(x|\boldsymbol{\theta})||p(x|\boldsymbol{\theta})]$ is 0 as both sides of the KL divergence have the same distribution, therefore

$$\mathrm{KL}[p(x|\boldsymbol{\theta})||p(x|\boldsymbol{\theta})] - \mathbb{E}[\nabla_\theta \log p(x|\boldsymbol{\theta})]^T d + \frac{1}{2}d^T\mathbb{F}d = \frac{1}{2}d^T\mathbb{F}d$$

□

An interesting corollary of the above result is that the KL divergence is symmetric locally as the above result holds true for both the forward and backward KL divergence.

Implementation details

The code is based on the OnlineNaturalGradient object in Kaldi src/nnet3/natural-gradient-online.h [13] and the github repo `https://github.com/YiwenShaoStephen/NGD-SGD`. The main changes that were made were to allow the program to run on a machine with multiple GTX 2080 Ti for efficient parallel training. The hyperparameters can be found in the tensorboard logs at the repo `https://github.com/Neoanarika/MA4270-project`. Similarly, the main instructions to run the code can be found in the README.md in the MA4270 project repo or attached in the zip file along with this document.

Wasserstein metric as the proximity function

Another thing about the proximity function in Mirror descent is that it also induces non-Euclidean geometry of the underyling manifold allowing for more efficient optimisation. This begs the question, what other geometries do proximity functions other than the Bergman divergence capture ? One interesting function is the Wasserstein metric, which has the advantage of being defined even when there is no shared support between the 2 distributions. This leads to the idea of Wasserstein natural gradient descent [7]. However, similar to conventional natural gradient methods there is a necessary matrix inversion step which makes the computation of each step expensive. This can be approximated using a dual formulation of the metric restricted to a Reproducing Kernel Hilbert Space leading to the idea of kernelized Wasserstein natural gradient descent [2].

References

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

[2] M Arbel, A Gretton, W Li, and G Montufar. Kernelized wasserstein natural gradient. In *International Conference on Learning Representations*, 2020.

[3] Yuxin Chen. Ele522: Large-scale optimization for data science mirror descent, 2019.

[4] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[5] Sham M Kakade. A natural policy gradient. In *Advances in neural information processing systems*, pages 1531–1538, 2002.

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[7] Wuchen Li and Guido Montúfar. Natural gradient via optimal transport. *Information Geometry*, 1(2):181–214, 2018.

[8] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[9] James Martens. New insights and perspectives on the natural gradient method, 2020.

[10] Michael K Murray and John W Rice. *Differential geometry and statistics*, volume 48. CRC Press, 1993.

[11] Nemirovskiĭ. *Problem Complexity and Method Efficiency in Optimization*.

[12] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks, 2014.

[13] Daniel Povey, Xiaohui Zhang, and Sanjeev Khudanpur. Parallel training of dnns with natural gradient and parameter averaging, 2015.

[14] Garvesh Raskutti and Sayan Mukherjee. The information geometry of mirror descent, 2014.

[15] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017.

[16] Phuong Thi Tran and Le Trieu Phong. On the convergence proof of amsgrad and a new version. *IEEE Access*, 7:61706â61716, 2019.

[17] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks, 2017.