Natural Gradient

Sylesh Suresh

February 2019

1 Introduction

The natural gradient is an alternative to standard stochastic gradient that has been a popular area of research in recent years as it has shown to lead to much faster neural network convergence than standard gradient descent even with the best optimizers like Adam and Adadelta. Many popular techniques that are used with neural networks such as batch normalization and second-order methods can also be shown to be approximations of the natural gradient, and recent research has been focused on developing new approximations to the natural gradient, KFAC (Kronecker-factored approximate curvature).

2 Motivation

The idea is that the neural network parameter space is arbitrary, and that it isn't the normal Euclidean space we are used to. Our standard gradient descent method would be perfect if the space was Euclidean, but it turns out that it isn't - the space is actually a Riemannian manifold. We are trying to apply a Euclidean method to a space which is non-Euclidean. The parameter space is like Earth - round, curved, but the standard gradient is treating it as if it's flat.

Let's look for evidence that our parameter space isn't flat. Let's consider our current gradient update under two different coordinate systems. We will represent the weights of our neural network as θ (a vector containing all the weights), the loss function as J, and the learning rate as α.

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\theta_k)$$

where θ_k represents the weights at iteration k. Consider moving to a different coordinate system, $x = A^{-1}\theta$. Let's optimize the function $J(Ax)$ with respect to x instead of θ now.

$$x_{k+1} = x_k + \alpha \nabla_x J(Ax_k)$$

We *should* be able to arrive at the first equation by multiplying the second equation by A, but instead, we arrive at:

$$Ax_{k+1} = Ax_k + \alpha A A_T \nabla_{Ax} J(Ax_k)$$

which is

$$\theta_{k+1} = \theta_k + \alpha A A_T \nabla_\theta J(\theta_k)$$

(the A_T comes from the chain rule). That's not our original equation. This means that applying a linear transformation (A) to the coordinate system changes our weight update, so we *have* to take the structure of our coordinate system (or space) into account. We will want an affine invariant parameter update in the end (meaning that going from θ to $x = A^{-1}\theta$ won't change the parameter update).

Imagine that our update was actually

$$\theta_{k+1} = \theta_k + \alpha H^{-1}(\theta_k) \nabla_\theta J(\theta_k)$$

where H is the *Hessian* (a matrix of second derivatives, where $H_{ij}(\theta) = \frac{\partial J}{\partial \theta_i \partial \theta_j}$. Why? When we go from x to θ, the Hessian transforms from $H(x)$ to $A_T H(\theta) A$ (again, this is the chain rule). When we optimize the function $J(Ax)$ with respect to x instead, we get:

$$x_{k+1} = x_k + \alpha (A_T H A)^{-1} A_T \nabla_\theta J(A x_k).$$

This is

$$x_+ \alpha A^{-1} H^{-1} A_T^{-1} A_T \nabla_\theta J(A x_k) = \alpha A^{-1} H^{-1} \nabla_\theta J(A x_k)$$

If we multiply by A, we get our original equation! This leads us to believe that a natural gradient could be defined as something like $H^{-1}\nabla_\theta J$. Now, it turns out that this step direction is invariant to affine transformations to our coordinate system, but not to *general* coordinate transformations. This does, however, help us get an idea of what a truly natural gradient might look like.

3 Derivation

Now that we have our bearings and a more clear idea of what we are looking for, let us approach the problem from scratch. Let us define our problem as follows. We want to calculate what change in the weight vector will minimize the cost function the most. Let's define the change in the weight vector $d\theta = \epsilon a$ where a is a unit vector and ϵ is some small constant. In general, $f(r+dr) = f(r) + \nabla f_T dr$. So,

$$J(\theta + d\theta) = J(\theta) + \epsilon \nabla J_T(\theta) a$$

This is true for small $d\theta$. We want to minimize $J(\theta + d\theta)$ with respect to a with the constraint that a is a unit vector. The constraint is $|a|^2 = 1$. In Euclidean space, this is simply $\sum_i (a_i^2)$ where a_i is the ith component of a, but this isn't the case for the parameter space of neural networks. Instead, the general formula for the magnitude of a vector is $a_T G a$ for some positive-definite matrix G (a matrix defined such that $v_T G v$ for any vector v is always positive). Notice that if G is the identity, the formula is just $a_T a$ is the formula we are used to in

Euclidean space. The matrix G needed for normalization depends on the space. We can use Lagrangian multipliers to solve this minimization problem.

$$\frac{\partial}{\partial a_i}[J(\theta) + \epsilon \nabla J_T(\theta) a] = \lambda \frac{\partial}{\partial a_i}[a_T G a]$$

We can rewrite $a_T G a$ as $\sum_{i,j} g_{ij} a_i a_j$ where g_{ij} is the element of G in row i and column j. Differentiating, we arrive at

$$\epsilon \frac{\partial J(\theta)}{\partial \theta_i} = \lambda \sum_j g_{ij} a_j$$

for all i. Since this is true for every i, we can conclude:

$$\epsilon \nabla J(\theta) = \lambda G a$$

Solving for a, we finally arrive at:

$$a = \frac{\epsilon}{\lambda} G^{-1} \nabla J(\theta)$$

a is a unit vector, so $\frac{\epsilon}{\lambda}$ is just a normalizing constant. So our update, is a constant multiple of $-G^{-1}\nabla J(\theta_k)$. We define our weight update, or *natural gradient* as

$$\tilde{\nabla} J(\theta) = G^{-1} \nabla J(\theta)$$

We can show that G is not only a positive-definite matrix representative of the neural network parameter space but is also related to the Hessian of the weights. (It isn't the plain Hessian itself, however - the Hessian won't always have the properties that we're looking for. Specifically, the Hessian is not representative of the parameter space and does not act as the normalizing matrix in the generalized dot product formula ($a_T G a$). An easy way of seeing this is noticing that the Hessian isn't necessarily positive-definite - but the normalizing matrix *must* be positive-definite; otherwise, a vector's dot product with itself will be negative, which violates an essential property of the dot product.)

4 Calculation

Now, we will attack the natural gradient problem from *yet another* perspective - but this time, we will get an explicit formula for the form of the natural gradient, *including* the mysterious matrix G.

4.1 Parameterization-Invariant Metric

The problem was that our dot product/distance measure was not invariant to the weight parameterization - in standard gradient descent we assume that our space is Euclidean and that our distance measure is the Euclidean metric (hence $|a|^2 = 1$. In the derivation section, this assumption would have set G equal to

the identity, which would have given us the standard gradient update). Let us consider a distance measure that would be invariant to reparameterization. Consider KL divergence:

$$KL(P(x), Q(x)) = \int_{-\infty}^{\infty} p(x) \log \left(\frac{p(x)}{q(x)} \right) dx$$

where $p(x)$ and $q(x)$ are the probability density functions corresponding to $P(x)$ and $Q(x)$, respectively. (That is, $p(x)dx$ will give you the probability that x lies between x and $x + dx$ given $P(x)$ - $p(x)$ is a probability density). Let's make the transition from x to $y = f(x)$. $p(x)dx = p(y)dy$ since y is just a coordinate transformation that shouldn't change the probabilities. We then have, going from the x-world to the y-world:

$$KL(P, Q) = \int_{-\infty}^{\infty} p(y) \log \left(\frac{p(y)\frac{dy}{dx}}{q(y)\frac{dy}{dx}} \right) dy$$

The derivatives cancel, so we simply have:

$$KL(P(x), Q(x)) = \int_{-\infty}^{\infty} p(y) \log \left(\frac{p(y)}{q(y)} \right) dy$$

This is the exact same as our original equation except with y instead of x, implying that the KL divergence is invariant under *any* coordinate transformation $y = f(x)$.

4.2 Approximation

Recall that we are looking for a parameterization-invariant measure of the difference between the set of weights θ and $\theta + d\theta$ (Taking the difference and taking the magnitude squared wasn't invariant). Let's use an approximation of the KL divergence as our metric. First, we need to understand how we can treat a neural network as a probability distribution over its predictions, i.e. the classes. Given a particular set of weights, there is a certain probability that the network will predict a particular class, for any input. We represent this probability as $P(y|\theta)$, where y represents the class. By computing $KL(P(y|\theta), P(y|\theta + d\theta))$, we are calculating how much the change in weights $d\theta$ changes the network's output probability distribution. Intuitively, we can see that this is a better way of measuring the change in the network than simply measuring the magnitude of $d\theta$ since the change in the *output probability distribution* is what we really care about, not the weights themselves. We will consider the case of discrete classes; so we will use the discrete version of the KL divergence:

$$KL(P(y|\theta), P(y|\theta + d\theta)) = \sum_{y} P(y|\theta) \log \left(\frac{P(y|\theta)}{P(y|\theta + d\theta)} \right) dx$$

We'll exploit the Taylor series about $d\theta$ to approximate this metric:

$$KL(P(y|\theta), P(y|\theta+d\theta)) \approx \sum_{y} P(y|\theta) \left(\log \frac{P(y|\theta)}{P(y|\theta)} - \nabla_\theta \log P_T(y|\theta) d\theta - \frac{1}{2} d\theta_T \nabla_\theta^2 \log P(y|\theta) d\theta \right)$$

We can simplify this all the way to:

$$= \frac{1}{2} d\theta_T \left(\sum_y P(y|\theta) \nabla_\theta \log P(y_\theta) \nabla_\theta^T \log P(y_\theta) \right) d\theta = \frac{1}{2} d\theta_T F(\theta) d\theta$$

The expression sandwiched between $d\theta_T$ and $d\theta$ is called the Fisher Information matrix, $F(\theta)$. In other words, the matrix G that we discovered before is equal to the Fisher Information matrix. Our final natural gradient update, is, then:

$$\tilde{\nabla} J(\theta) = F^{-1} \nabla J(\theta)$$