SOBOLEV GRADIENTS AND NEURAL NETWORKS

Michael R. Bastian, Jacob H. Gunther, Todd K. Moon

Utah State University
Department of Electrical and Computer Engineering
4120 Old Main Hill, Logan, UT 84322-4120

ABSTRACT

By using a formulation similar to a Sobolev Gradient for the Natural Gradient a new algorithm has been developed that converges faster, uses fewer additional parameters and has smaller storage requirements all while not overtraining to a training set. Simulation results show the improvements for an applicable problem.

Index Terms– Algorithms, Feedforward Neural Networks, Newton Methods

1. INTRODUCTION

Many signal processing problems are formulated to minimize the norm of an error vector [1]. The solution to these formulations is either a vector or a matrix. By applying a similar formulation to the error function of a multilayer perceptron and understanding the structure of the error function, a new algorithm has been developed. The performance of this algorithm is compared to the original backpropagation method [2] and the Adaptive Natural Gradient [3] [4]. In [5] prior research is reported.

2. SOBOLEV GRADIENTS EXPLAINED

This section explains Sobolev Gradients and the new formulation of the gradient of a multilayer perceptron.

2.1. Sobolev Spaces

A *Sobolev Space* [6] [7] is an inner product space on functions. Sobolev Spaces are commonly used to find solutions to partial differential equations. Let $f^{(k)}(t)$ denote the k-th derivative of the function $f(t)$. Then the inner product between two functions in a Sobolev Space is defined as

$$\langle f, g\rangle = \sum_{k=0}^{n} \int_a^b f^{(k)}(t) g^{(k)}(t)\, dt. \tag{1}$$

The norm of an element of a Sobolev Space is then

$$\|f\|^2 = \sum_{k=0}^{n} \int_a^b \left| f^{(k)}(t) \right|^2 dt. \tag{2}$$

A *Sobolev Gradient* [8] is derived from the norm of the Sobolev Space. The gradient of $h(f) = \frac{1}{2}\|f\|^2$ is

$$\nabla h = (f, f', \ldots, f^{(n)}). \tag{3}$$

2.2. Application to Multilayer Perceptrons

This article does not describe a Sobolev Space comprised specifically of multilayer perceptrons. However, it does use a similar structure. The gradient of a simple perceptron is a matrix. The gradient of a multilayer perceptron is a block-diagonal matrix and its structure is similar to that of the gradient given in (3). Understanding that the inner product space of the parameters differs from the inner product space of the errors is the key principle of this new method.

The inner product of a Sobolev Space is the sum of inner products between the derivatives of two functions. The backpropagation algorithm [2] uses the chain rule for differentiation to compute the gradient for each network layer (n.b. a network layer is a matrix). The learning format of a multilayer perceptron and the training set dictate to a large extent the structure. The error function is the sum of the error vectors' L_2 norms. Instead of one index for the order of the derivative there are two indices: one for the training example and another for the network layer.

2.2.1. Error Function of a Multilayer Perceptron

The parameters of the multilayer perceptron are represented as the block-diagonal matrix

$$\mathcal{A} = \begin{bmatrix} A_m & 0 & \cdots & 0 \\ 0 & A_{m-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_1 \end{bmatrix}. \tag{4}$$

The output of the network given input $\mathbf{x}$ is the result of the vector-valued function $\mathbf{F}(\mathbf{x}, \mathcal{A})$. Let $\{\Phi_1, \ldots, \Phi_m\}$ be a set of recursively defined functions. The output function $\mathbf{F}$ is equal to the m-th function. That is $\mathbf{F}(\mathbf{x}, \mathcal{A}) = \Phi_m(\mathbf{x}, \mathcal{A})$.

$$\Phi_1(\mathbf{x}, \mathcal{A}) = \mathbf{h}_1\left(A_1 \mathbf{x}\right) \tag{5}$$

$$\Phi_i(\mathbf{x}, \mathcal{A}) = \mathbf{h}_i\left(A_i \Phi_{i-1}\left(\mathbf{x}, \mathcal{A}\right)\right) \tag{6}$$

Each $\mathbf{h}_i$ is a differentiable nonlinear threshold function with Jacobian matrix H_i. The error function of a multilayer perceptron is given by

$$J(\mathcal{A}) = \frac{1}{2}\beta \sum_{k=1}^{n} \|\mathbf{y}_k - \mathbf{F}(\mathbf{x}_k, \mathcal{A})\|^2 + \frac{1}{2}\alpha \|\mathcal{A}\|^2. \quad (7)$$

The α and β in (7) are hyperparameters in prior distributions of the norm of the parameter $\mathcal{A}$ and the SNR of the data respectively [9].

2.2.2. Directional Derivative

Let $\gamma(t) = \mathcal{A} + t\mathcal{P}$ be a curve through the set of block-diagonal matrices as described in (4). The directional derivative of the curve $\dot{\gamma}(0) = \mathcal{P}$ will be assumed to be the direction of steepest descent. The inner product of the gradient with the direction of steepest descent is the directional derivative.

$$\langle \mathcal{P}, \nabla J \rangle = \lim_{t \to 0} \frac{1}{t} \big[J(\mathcal{A} + t\mathcal{P}) - J(\mathcal{A}) \big]. \quad (8)$$

To solve this equation for ∇J, the t must be broken out of the expression $\mathbf{F}(\mathbf{x}, \mathcal{A} + t\mathcal{P})$.

2.2.3. Taylor Series Expansion

For a function of several variables f, the first two terms of the Taylor series expansion about $\mathbf{x}$ is

$$f(\mathbf{x} + t\mathbf{p}) = f(\mathbf{x}) + t\nabla f(\mathbf{x})^T \mathbf{p}.$$

Similarly, for the mapping $\mathbf{h}(A\mathbf{x})$, there is a Taylor series expansion about the matrix A.

$$\mathbf{h}((A + tP)\mathbf{x}) = \mathbf{h}(A\mathbf{x}) + tH(A\mathbf{x})P\mathbf{x} \quad (9)$$

In (9), $H(A\mathbf{x})$ is the Jacobian matrix of $\mathbf{h}$ at the point $\mathbf{u} = A\mathbf{x}$,

$$H(\mathbf{u}) = \left[\frac{\partial h_i}{\partial u_j} \right]_{\mathbf{u}=A\mathbf{x}}.$$

2.2.4. Error Function Gradient

The output function $\mathbf{F}(\mathbf{x}, \mathcal{A})$ can be rewritten as

$$\mathbf{F}(\mathbf{x}, \mathcal{A}) = \begin{bmatrix} I & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} \Phi_m(\mathbf{x}, \mathcal{A}) \\ \vdots \\ \Phi_1(\mathbf{x}, \mathcal{A}) \end{bmatrix}. \quad (10)$$

For each threshold function $\mathbf{h}_i$,

$$H_i(\mathbf{z}) = \left[\frac{\partial h_k}{\partial z_l} \right]_{\mathbf{z}=\Phi_{i-1}(\mathbf{x},\mathcal{A})} \quad (11)$$

is the respective Jacobian matrix.

As described in [2], the error is propagated backwards by the chain rule. The matrices $\{B_1, \ldots, B_m\}$ are the backpropagation transformations. They are determined recursively from the Jacobian matrices of the threshold functions $\{H_1, \ldots, H_m\}$ and the multilayer perceptron parameters $\{A_1, \ldots, A_m\}$.

$$B_m(\mathbf{x}) = H_m(\Phi_{m-1}(\mathbf{x}, \mathcal{A})) \quad (12)$$

$$B_i(\mathbf{x}) = B_{i+1}(\mathbf{x}) A_{i+1} H_i(\Phi_{i-1}(\mathbf{x}, \mathcal{A})) \quad (13)$$

These transformations are assembled in a block-diagonal matrix to form the backpropagation transformation,

$$\mathcal{B}(\mathbf{x}) = \begin{bmatrix} B_m(\mathbf{x}) & 0 & \cdots & 0 \\ 0 & B_{m-1}(\mathbf{x}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & B_1(\mathbf{x}) \end{bmatrix}. \quad (14)$$

The multilayer perceptron output function can then be expanded as

$$\mathbf{F}(\mathbf{x}, \mathcal{A} + t\mathcal{P}) = \mathbf{F}(\mathbf{x}, \mathcal{A}) + t \begin{bmatrix} I & 0 & \cdots & 0 \end{bmatrix} \mathcal{B}(\mathbf{x}) \mathcal{P} \mathbf{u} \quad (15)$$

where $\mathbf{u}$ is the vector of inputs defined as

$$\mathbf{u} = \begin{bmatrix} \Phi_{m-1}(\mathbf{x}, \mathcal{A}) \\ \vdots \\ \Phi_1(\mathbf{x}, \mathcal{A}) \\ \mathbf{x} \end{bmatrix}. \quad (16)$$

The inner product of the gradient ∇J and the direction $\mathcal{P}$ can now be rewritten as

$$\begin{aligned} \langle \mathcal{P}, \nabla J \rangle &= -\beta \sum_{k=1}^{n} \langle \mathcal{B}(\mathbf{x}_k) \mathcal{P} \mathbf{u}_k, \mathcal{I} \mathbf{z}_k \rangle + \alpha \langle \mathcal{P}, \mathcal{A} \rangle \\ &= -\beta \sum_{k=1}^{n} \langle \mathcal{P}, \mathcal{B}(\mathbf{x}_k)^T \mathcal{I} \mathbf{z}_k \mathbf{u}_k^T \rangle + \alpha \langle \mathcal{P}, \mathcal{A} \rangle \\ &= -\beta \sum_{k=1}^{n} \sum_{i=1}^{m} \langle P_i, B_i(\mathbf{x}_k)^T \mathbf{z}_k \Phi_{i-1}(\mathbf{x}_k)^T \rangle \\ &\quad + \alpha \sum_{i=1}^{m} \langle P_i, A_i \rangle \end{aligned}$$

where $\mathbf{z}_k = \mathbf{y}_k - \mathbf{F}(\mathbf{x}_k, \mathcal{A})$, $\mathcal{I} = \begin{bmatrix} I & I & \cdots & I \end{bmatrix}^T$ and the matrix B_k^i is the backpropagation matrix for layer i and training example k. Let $\Phi_0(\mathbf{x}_k, \mathcal{A}) = \mathbf{x}_k$.

Thus, the gradient of the error function for layer i can be rewritten as

$$\nabla J_i = -\beta \sum_{k=1}^{n} B_i(\mathbf{x}_k)^T \mathbf{z}_k \Phi_{i-1}(\mathbf{x}_k)^T + \alpha A_i. \quad (17)$$

The gradient ∇J is a block-diagonal matrix with each block denoted as ∇J_i.

2.3. Summary

The innerproduct in a Sobolev Space is the sum of the inner products of the derivatives of two functions. Similarly, the inner product of two multilayer perceptrons is the sum of the inner products of their weight matrices.

The Sobolev Gradient of a function is a vector composed of a function and its derivatives. The Sobolev Gradient of a Multilayer Perceptron is a block-diagonal matrix of gradients of each layer. Each gradient is essentially the backpropagation matrix multiplied by the outer product of the error vector and the input vector to the specific layer.

This organization produces an algebraic structure that allows an easy formulation of the Natural Gradient.

3. NATURAL GRADIENT

The natrual gradient estimates the Riemannian metric $G(\mathcal{A})$ at every point in the parameter space of all possible values of $\mathcal{A}$.

3.1. The Fisher Information Matrix

The metric for estimation problems is the Fisher Information Matrix [3]. The probablistic models for the errors and the weights simplify the Fisher Information Matrix.

$$G(\mathcal{A}) = E\left[(\nabla J)(\nabla J)^T\right]. \tag{18}$$

The Fisher Information Matrix of a multilayer perceptron is a block-diagonal matrix. Each block corresponds to a specific layer i and is

$$G_i = \beta \sum_{k=1}^{n} \|\Phi_{i-1}(\mathbf{x}_k)\|^2 B_i(\mathbf{x}_k)^T \mathbf{z}_k \mathbf{z}_k^T B_i(\mathbf{x}_k) + \alpha I. \tag{19}$$

3.2. Learning Rule

The learning rule for batch-training a multilayer perceptron is given as

$$\mathcal{A}_{t+1} = \mathcal{A}_t - \eta_t G(\mathcal{A}_t)^{-1} \nabla J(\mathcal{A}_t). \tag{20}$$

The index variable t denotes the learning epoch. The learning rate for the learning epoch is η_t.

3.3. Comparison with Other Methods

The Adaptive Natural Gradient [4] and the Levenburg-Marquardt algorithm [10] are two methods that function similarly to this Sobolev Natural Gradient. The advantage that this method has over the first is that the Fisher Information Matrix has fewer non-zero elements and is therefore computationally advantageous.

Let $\{m_1, m_2, \ldots, m_L\}$ represent the number of nodes in each layer of a multilayer perceptron. The number of elements in the Fisher Information Matrix for the Sobolev and Adaptive Natural Gradient methods are given in (21) and (22) respectively. For example, if a multilayer perceptron had dimensions $(10, 10, 10, 5)$ then the FIM for the Adaptive Natural Gradient would have 62,500 non-zero elements. The Sobolev Natural Gradient would only have 225.

$$N = m_2^2 + m_3^2 + \cdots + m_L^2 \tag{21}$$

$$N = (m_1 m_2 + m_2 m_3 + \cdots + m_{L-1} m_L)^2 \tag{22}$$

Levenburg-Marquardt has the same number of non-zero parameters in the matrix it inverts as the Adpative Natural Gradient. The only advantage it has over the Sobolev Gradient is the use of a trust region method to determine the learning rate for each epoch. A trust region method could also be used with this Sobolev Gradient method to determine the learning rate.

4. EXPERIMENTAL RESULTS

The problem chosen to compare performance is decoding a noise-corrupted (10,5) Low Density Parity Check code [11]. This problem was chosen because it is easy to generate a large number of examples. It is a also an engineering problem with practical application. The network has ten inputs and a bias, three layers each with ten hidden units and a bias, and five outputs that are the decoded message. The hyperparameters were set as $\alpha = 0.01$ and $\beta = 0.01$.

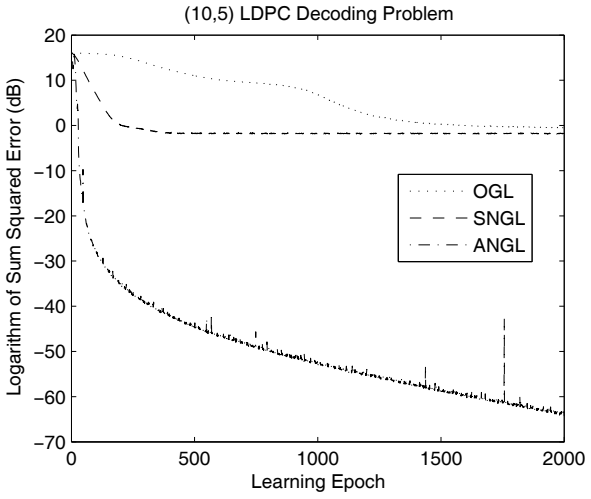

Fig. 1. Training Algorithm Sum-Squared Error

Figure 3 shows that all three algorithms train networks that perform equally well. Figure 1 shows that the Sobolev Natural Gradient Learning (SNGL) algorithm converges much faster than the Ordinary Gradient Learning (OGL) algorithm

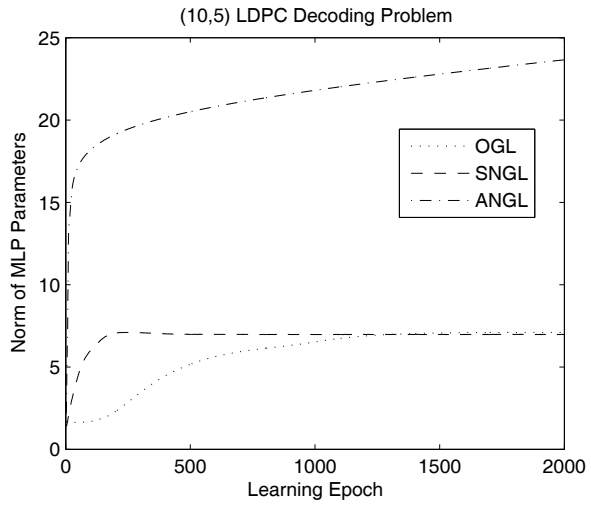

Fig. 2. Multilayer Perceptron Norm

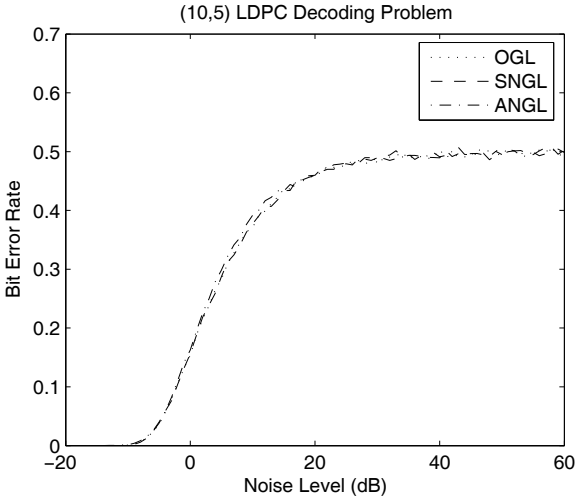

Fig. 3. LDPC Decoder Bit Error Rate

but that the Adaptive Natural Gradient Learning algorithm (ANGL) converges even faster. However, Figures 2 and 3 show that the ANGL algorithm overtrains without any increase in performance. The larger norm of the weight matrix means that when the network is right each output is very close to the target value. The weights are so large as to shape the resulting threshold functions so that they are almost step functions. The SNGL algorithm avoids this by using a prior distribution on the weights.

5. CONCLUSION

The error function of a multilayer perceptron is similar in structure to the norm in a Sobolev Space. Forming the Sobolev Gradient as a block-diagonal matrix means that the Fisher Information Matrix is also a block-diagonal matrix. This simplifies the inversion of the matrix to the inversion of the blocks. The hyperparameter of the prior distribution guarantees that the block-diagonal Fisher Information Matrix is well-conditioned.

This new training algorithm works as well as the Adaptive Natural Gradient method but requires many fewer parameters, is more computationally efficient and uses a prior distribution on the weights. This Bayesian learning prevents overtraining that is a flaw of many learning algorithms.

6. REFERENCES

[1] Todd K. Moon and Wynn C. Stirling, *Mathematical Methods and Algorithms for Signal Processing*, Prentice Hall, August 1999.

[2] David E. Rumelhart and James L. McClelland, *Parallel Distributed Processing*, MIT Press, Cambridge, MA, 1986.

[3] S. Amari, "Natural gradient works efficiently in learning," *Neural Computation*, vol. 10, no. 2, pp. 251–276, 1998.

[4] S. Amari, H. Park, and K. Fukumizu, "Adaptive method of realizing natural gradient learning for multilayer perceptrons," *Neural Computation*, vol. 12, no. 6, pp. 1399–1409, 2000.

[5] Michael R. Bastian, Jacob H. Gunther, and Todd K. Moon, "An improvement to the natural gradient learning algorithm for multilayer perceptrons," in *Proceedings of the 2005 IEEE International Conference on Acoustics, Speech, and Signal Processing*.

[6] E. Stein, *Singular Integrals and Differentiability Properties of Functions*, Princeton University Press, 1970.

[7] James P. Keener, *Principles of Applied Mathematics: Transformation and Approximation*, Perseus Books Group, revised updated edition, 2000.

[8] J. W. Neuberger, *Sobolev Gradients and Differential Equations*, Number 1670 in Lecture Notes in Mathematics. Springer, 1997.

[9] David J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, 2003.

[10] Jorge Nocedal and Stephen J. Wright, *Numerical Optimization*, Springer Series in Operations Research. Springer, 1999.

[11] Todd K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*, chapter 16, Wiley-Interscience, June 2005.