Exploration of Improvement of Backpropagation Algorithm

Mark Musil, Jian Meng, Yuhao Zhou, Peilong Ning
Maseeh College of Engineering & Computer Science
Portland State University
Portland, Oregon, 97201

I. Executive Summary

This document details the work of the revised back-propagation capstone. It is not an exhaustive record of all work that was done but rather focuses on the most salient results. This document is accompanied by several supporting documents (project proposal, project design specification, and project schedule) as well a GitHub repository containing project source code.

The work in question consisted of researching and designing two novel backpropagation algorithms (two remaining among many considered). The results show the novel algorithms performing sometimes better than, but roughly equal to, stochastic gradient descent. The algorithms should be tested on more tightly constrained networks (networks where the number of data points is roughly equal to the number of parameters) and on noisier real-world data such that the algorithm most capable of finding good minima can be decided.

II. Introduction

The impetus for this capstone was the sponsor's desire to find an improved neural network training algorithm. The goal of such an algorithm is to reduce training time and resources while still achieving state-of-the-art accuracy and generalization. The sponsor gave guidance to the group by directing us to focus on the distribution of error in the network. Specifically, the sponsor hypothesized that error resulting from inadequately trained weights in the first layer's weight matrix is amplified by the following layers and propagates the error in the first layer. This is hypothesized to also occur in successive layers. In addition to above described problem of error propagation, the sponsor also tasked us with finding a method which could converge in less time and with fewer computations.

III. Background

The backpropagation algorithm is a classic algorithm for training neural networks. It was redefined many times in the 1970s and 1980s. Some of its algorithmic ideas come from the control theory of the 1960s. The main idea of the back propagation algorithm is:

1) Input the training set data to the input layer, go through the hidden layer, and finally reach the output layer and output the result, which is the forward propagation process;
2) Since there is an error between the output result and the actual result, the error between the estimated value and the actual value is calculated, and the error is propagated backward from the output layer to the hidden layer until it propagates to the input layer.

3) In the process of backpropagation, adjust the values of various parameters according to the error, continuously iterate the above process until convergence.

In the case where the input data is fixed, the backpropagation algorithm uses the output sensitivity of the neural network to quickly calculate various hyperparameters in the neural network. We can use nonlinear function Sigmoid or RELU as activation functions of the neuron. We get the gradient by using the chain rule, then train the network using the gradient descent method. Each time you move a small step along the negative direction of the gradient, repeat it until the network output error is minimal. First of all, let's define some variables using a three-layer neural network as an example.

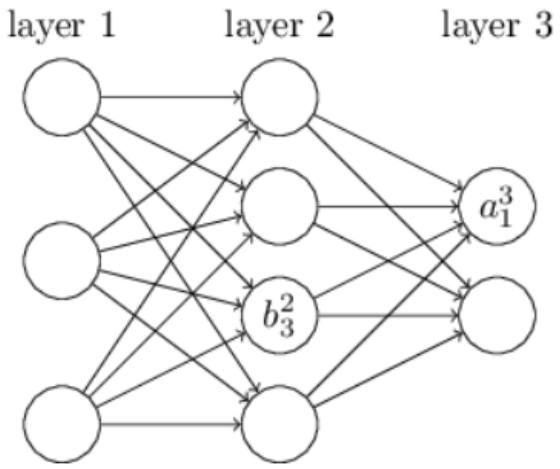


Fig. 1: 3 Layer Neural Network source: http://neuralnetworksanddeeplearning.com

Layer 1 to layer 3 are the input layer, the hidden layer, and the output layer, respectively.

ω_{jk}^l represents the weight of the kth neuron of the (l-1)th layer connected to the jth neuron of the lst layer;

b_j^l indicates the bias of the j-th neuron of the l-th layer;

z_j^l indicates the input of the j-th neuron of the l-th layer, $z_j^l = \Sigma_k\ \omega_{jk}^l\ a_k^{l-1} + b_j^l$;

a_j^l indicates the output of the j-th neuron of the l-th layer, $a_j^l = \sigma(\sum_k \omega_{jk}^l a_k^{l-1} + b_j^l)$, σ is activation function;

C represents the cost function, $C = \frac{1}{2n}\Sigma_x \parallel y(x) - a^L(x) \parallel^2$, x represents the input sample, y represents the actual classification, a^L represents the predicted output, and L represents the maximum number of layers of the neural network, the cost function is used to calculate the error between the output value and the actual value.

The equation for Back propagation is following:

Summary: the equations of backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \quad \text{(BP1)}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \quad \text{(BP2)}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \quad \text{(BP3)}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \quad \text{(BP4)}$$

Fig. 2: Equation of Back Propagation. source: http://neuralnetworksanddeeplearning.com

The error generated in the j-th neuron of the l-st layer (the error between the actual value and the predicted value) is defined as:

$$\delta^l_j = \frac{\partial C}{\partial z^l_j} \tag{1}$$

The first equation calculates the error generated by the last layer of the neural network, the second equation calculates the error generated by each layer of the neural network (from the back to the front), the third equation calculates the gradient of the bias, and the fourth equation calculates the gradient of the weight.

IV. Error in Stochastic Gradient Descent

Gradient Descent methods, as a set of iterative optimization algorithms, are widely used in machine learning method, especially for weight updating in neural network. Due to the large amount of samples in deep learning task, it is hard to compute the gradient of the whole dataset at once by using the traditional Gradient Descent.

To solve such problem, Stochastic Gradient Descent is developed which will only calculate gradient for one sample for each iteration. SGD will significantly improve the speed of calculating gradient. As a drawback, since the gradient of each sample is only an estimation of the whole dataset, the descending direction is not always correct.

In order to obtain a more accurate direction of descent along the error-surface, mini-batch Stochastic Gradient Descent is a common choice in training neural networks. This method will compute the gradient of a certain number of samples. Intuitively, training on more samples leads to more accuracy than on fewer samples. This doesn't means there is no error in the mini-batch Stochastic Gradient Descent.

One of our methods aims to solve the error in mini-batch Stochastic Gradient Descent. The problem space in deep learning is always high dimensional. The small changes in some dimensions could be "error" generated by the estimation error. Our method is trying to reduce such error. The details of our algorithms will be introduced in Section VI.

V. Related Works

At the beginning of this project, the team invested energy in investigating many different methods for improving backpropagation. We began our search consider how the activation functions could be impacting network convergence such as is addressed in [5], [6], [12]. We also explored many gradient free methods such as the directed random search [10].

Our sponsor asked us to focus on methods which reward movement along the most promising directions of the error vector δ. One paper which helped us along this route was [8] which preserves the sign in the δ term in the last layer. That is, $\delta E/\delta z_j = z_j - t_j$ where the z_j is the activation of a neuron at the output layer and t_j is the target value for that neuron. This work would help motivate our final two algorithms which concentrate on optimizing the role of the error vector δ at each layer (or the reslts movements directed by δ). We were also motivated by the success of the RPROP method [9] which uses information about the changing sign of the $\delta C/\delta w_{jk}$ terms to obviate the network becoming stuck.

VI. Methods

Under instruction of the sponsor, the capstone focused on finding ways to improve the issue of propagated, compounded error. Specifically, the forward and backward propagation steps can erroneously amplify errors present in the weights and biases. To address this issue, two methods were proposed and tested throughout the course of the capstone.

The Error-by-layer method is based on the phenomenon that the error, that back-propagates to each layers, are not equal. It focuses on reducing error propagation by more aggressively training at either the early layers or the later layers. Another fact is that the gradient calculation in mini-batch SGD is not perfectly accurate because the error of certain mini-batches is only an estimation of the error of the whole dataset. The goal of Error-by-neuron method is obtaining a more accurate, normalized global gradient estimation in mini-batch SGD which can represents a more general feature extraction of the mini-batch. These approaches were chosen because it was theorized that such methods would allow error to be quickly adjusted for during early training epochs resulting in more rapid convergence.

A. Method 1: Error-by-layer method

This method uses the following modified update equations during the weight and bias update steps of the training process.

$$w^l_{jk} = w^l_{jk} - \eta(a^{l-1}\delta^l_j)\kappa^l \tag{2}$$

$$b^l_j = b^l_j - \eta\delta^l_j\kappa^l \tag{3}$$

This method introduces another tunable (not trainable) parameter k^l for each layer which, like in the case of the momentum driven learning rate method, must be chosen for each model and a single best value cannot be defined [9]. It is important that a relatively mild value is chosen ($0 < k < 4$) since accelerating learning too much can cause oscillations, some of which can be seen in the experimental data (in the GitHub repository folder `root/NNNP_TEST/loss_figure/`). It is possible to either define the k^l values such that more learning is allocated at former or latter layers.

B. Method 2: Error-by-neuron method

In mini-batch Stochastic Gradient Descent (SGD), the calculation of gradient is only based on a few samples, which is different from the correct gradient of the whole dataset. It reveals that there exists "noise" in gradient calculation in mini-bath SGD. Reducing such "noise" could improve the accuracy of the gradient estimation. As a widely used tool, Singular Value Decomposition (SVD) has been useful in many different applications like dimensionality reduction, image denoising, and principle component analysis [11]. Applying SVD in mini-batch SGD can reduce the potential "noise" in the gradient estimation.

1) Singular Value Decomposition: In our method, we only focus on the real field. So the Singular Value Decomposition can be expressed by following statement [4]. Suppose matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, there exists:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{4}$$

where $\mathbf{V}^T$ is the *transpose* of $\mathbf{V}$, $\mathbf{U} \in \mathbb{R}^{m \times m}$ and $\mathbf{V} \in \mathbb{R}^{n \times n}$ are the orthogonal matrices, $\Sigma \in \mathbb{R}^{m \times n}_{\geq 0}$ is a diagonal matrix. The items on the diagonal of Σ are the singular value of the matrix $\mathbf{A}$, which represents the energy distribution of the corresponding singular vectors in $\mathbf{U}$ and $\mathbf{V}$. Since there are many different possible decompositions, it is a common convention to order the Σ matrix in descending order to make sure Σ is unique. Our following discussion is based on this common convention.

2) Error-by-neuron: :

Pick the first k singular values and vectors in the decomposition, we reconstruct the original matrix in lower dimensionality. After this operation, we get $\mathbf{U}_k \in \mathbb{R}^{m \times k}, \mathbf{\Sigma}_k \in \mathbb{R}^{k \times k}, \mathbf{V}_k \in \mathbb{R}n \times k$. Then we can reconstruct $\mathbf{A}_k = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}^T$. $\mathbf{A}_k$ has the same shape with $\mathbf{A}$ but have a low rank representation.

This method can be used in mini-batch SGD to find the lower rank representation of error matrix $\delta \in \mathbb{R}^{D \times N}$, where N is the number of neurons of the certain layer, and N is the batch size. The Error-by-neuron method calculates the SVD of δ firstly:

$$\delta = U\Sigma V^T \tag{5}$$

Then, as we described above, the first k singular values and vectors of U, Σ, and V in 5 is used to reconstruct the low rank δ_k:

$$\delta_k = U_k\Sigma_k V_k^T \tag{6}$$

After the reconstruction, the value in δ_k is different with δ but the shape of them are the same. Then we used δ_k to calculate the gradient and update the weight by $W = W - \eta\delta_k a^T$, where W is the weight, η is the learning rate and a is the activation output of previous layer. It is same with the classic SGD optimizing steps, with δ replaced by δ_k.

In Error-of-neuron method, by multiplying δ_k with the activation output to update the weight, weight of every neuron is minorly modified comparing to the classic SGD. The name "Error-by-layer" is from the way the algorithm adjust the weight.

We choose δ to do the SVD rather than the gradient δa^T because δ contains the error of each sample as vectors stacked in the matrix. Calculating SVD on δ can generalize the error in one batch. And it simplified the gradient calculation of biases in the neural network.

3) Implementation details: In practice, since we only need first k singular values and vectors in SVD, it is not necessary to compute all values in once. Instead, we can only compute the singular values we need. Hence, we used a sparse SVD solver in `Scipy` to speed up calculation.

Compare to computing the accurate global gradient, one potential advantage of SGD is that the gradient estimation can reduce the possibility of being stuck at a saddle point, because the perturbation in batches helps SGD to be unstable at the saddle point. Therefore, if the "noise" in the gradient is reduced in Error-by-neuron method, it is more likely to stuck a saddle point, which is shown in our experimental results. Due to the properties of mini-batch, the estimation cannot be perfectly accurate even using Error-by-neuron method. The Error-by-neuron method will leave the saddle point after a few iterations. This phenomenon can be found in the experimental results as well.

VII. Experimental Results

In this section, simulation and the comparison are completed to evaluate the efficiency and the performance of the revised methods that described in the previous section. To measure the performance of the algorithms, the multiple neural network structure has been applied to the different size(type) datasets.

The simulation interface and all the experiments are comparing the performance of the following three different types of the algorithm:

1) *Algorithm #1: Error by layer method*
2) *Algorithm #2: Error by neural method (Based on SVD)*
3) *Traditional Stochastic gradient descent (Control group)*

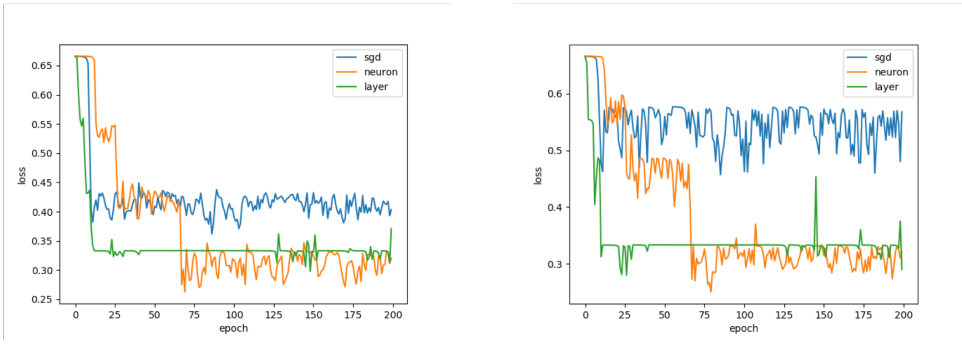
The objective of all the numerical simulations (experiments) is to examine the following perspectives:

(a) The speed of the convergence in the training phase.
(b) The overall accuracy of the test phase.
(c) The performance of the method with different size of the neural network structure.

To measure the accuracy of the classification, the methods are tested based on 4 different datasets. Table 1 illustrate the dimension and the size of the datasets that has been applied to the algorithms.

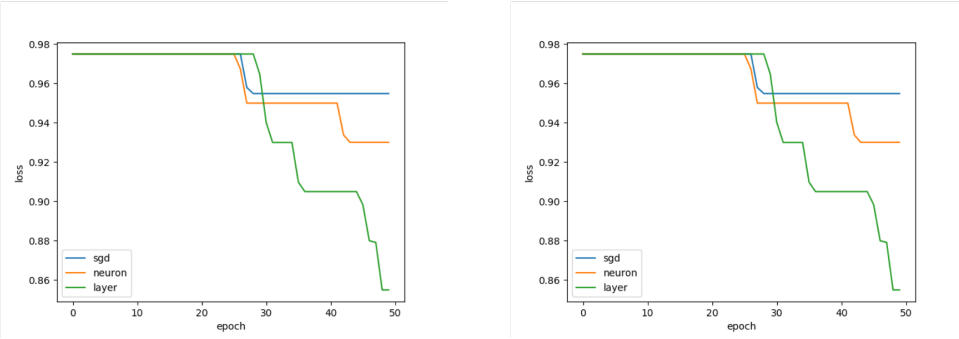
Dataset	**Features**		
	Classes	**Input features**	**Number of samples**
MNIST	10	784	$60,000$
IRIS	4	150	150
Coverttype	7	54	$581,012$
Olivetti Face	40	$4,096$	400

TABLE I: Table 1: Summary of the datasets



(a) Training phase (b) Test phase

Fig. 3: Experimental results on IRIS dataset. Hidden Dims: [60 30 15]; Learning Rate 0.05



(a) Training phase (b) Test phase

Fig. 4: Experimental results on MNIST dataset. Hidden Dims: [200 80 40]; Learning Rate 0.1

The experimental results based on *IRIS* and *MNIST* datasets are illustrated in Fig.3 and Fig.4. Considering the comparison between the revised methods and the traditional SGD, with a larger size structure, the performance of *Algorithm #2* is better than SGD.

Dataset	**Losses**	
	Final training loss	**Final test loss**
SGD - IRIS	0.23	0.24
SGD - MNIST	0.22	0.25
Algorithm #1 - IRIS	0.23	0.23
Algorithm #1 - MNIST	0.24	0.22
Algorithm #2 - IRIS	0.24	0.26
Algorithm #2 - MNIST	0.13	0.13

TABLE II: Summary of experimental results

VIII. Discussion

The results of the trials are promising although mixed. Some sets of hyperparameters generate robust performance with the Error-by-layer and Error-by-neuron methods, while other sets of parameters yielded oscillations, error-spikes, or slow convergence. For the error-by-layer method, performance is better when the layer weights are defined with the strongest k^l at the latter layers of the network, closer to the output.

It has been shown [2] that error surfaces have many minima across weight-space and finding them and staying in them is relatively easy to achieve while following a gradient [1], [3], [7]. Furthermore, many networks which have an abundance of trainable parameters are capable of memorizing the data because the parameters represent a large pattern storage medium [13]. This is why it is crucial to test the proposed algorithms on more tightly constrained networks.

These new navigation methods should be tested on networks with fewer parameters relative to data points and they're performance more closely examined. This will make the network more tightly constrained and make it more apparent which algorithm can find the minima most quickly [9].

In addition, these algorithms should be tested on other datasets and with other combinations of hyperparameters. The neural network program written for this project is very flexible in that regard. It can easily accept new datasets, hidden layer dimensions or other values [2].

References

[1] Devansh Arpit, Stanisław Jastrzebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 233–242. JMLR. org, 2017.

[2] Anna Choromanska, Mikael Henaff, Michaël Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surface of multilayer networks. *CoRR*, abs/1412.0233, 2014.

[3] J.N. Tsitsiklis D.P. Bertsekas. Gradient convergence in gradient methods. November 1997.

[4] Virginia Klema and Alan Laub. The singular value decomposition: Its computation and some applications. *IEEE Transactions on automatic control*, 25(2):164–176, 1980.

[5] Hahn-Ming Lee, Chih-Ming Chen, and Tzong-Ching Huang. Learning efficiency improvement of back-propagation algorithm by error saturation prevention method. *Neurocomputing*, 41(1-4):125–143, 2001.

[6] Youngjik Lee, Sang-Hoon Oh, and Myung Won Kim. An analysis of premature saturation in back propagation learning. *Neural Networks*, 6(5):719 – 728, 1993.

[7] Zachary C. Lipton. Stuck in a what? adventures in weight space. 2016.

[8] A. Van Ooyen and B. Nienhuis. Improving the convergence of the back-propagation algorithm. *Neural Networks*, 5(3):465 – 471, 1992.

[9] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rpop algorithm. 1993.

[10] U. Seiffert and B. Michaelis. Directed random search for multiple layer perceptron training. 2001.

[11] Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha. Singular value decomposition and principal component analysis. In *A practical approach to microarray data analysis*, pages 91–109. Springer, 2003.

[12] X.G. Wang, Z. Tang, H. Tamura, M. Ishii, and W.D. Sun. An improved backpropagation algorithm to avoid the local minima problem. *Neurocomputing*, 56:455 – 460, 2004.

[13] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *CoRR*, abs/1611.03530, 2016.