AMTH 437 Final Project Report: Investigation of the saddle point problem in neural network optimization

Yutaro Yamada

May 5, 2016

1 Introduction

Efficient optimization becomes an important aspect of the recent advancement in deep learning. This is especially apparent in the Large Scale Visual Recognition Challenge, where deep learning has been sucessfully applied; Batch Normalization [4] contributed to improving the test error by allowing easier optimization of the network. K. He [1] reports that residual mapping inside the network architecture leads to easier optimization of very deep neural networks, resulting in the best performance in the object recognition division. Although these techniques empirically work well, their statistical properties are not well-known. In response to this question, Dauphin [2] performs a series of experiments, which suggests that the proliferation of saddle points is the major impediment of rapid non-convex optimization in the high dimensional spaces.

In his paper, Dauphin proposes the saddle-free Newton method, which utilizes curvature information to define the shape of the trust region method, allowing the alogrithm to escape from the saddle points more rapidly and efficiently than classical methods such as gradient descent and Newton method. Although it is reported that the saddle-free Newton algorithm works well, it requires to compute the entire Hessian matrix because the algorithm rescales gradients by the absolute value of the inverse Hessian. This computatioin cannot be scaled to higher dimensions, which is a major disadvantage of this algorithm.

Professor Negahban proposed a new algorithm that circumvents this issue by utilizing the information of the maximum eigenvalue of the Hessian. This method is computationally efficient because it only requires two function calls of power method and efficiently determines a descent direction when gradient descent gets stucked, which is assumed to be around saddle points.

Thus, this algorithm could potentially accelerate the training time for deep neural networks.

2 Optimization in Neural Network

The stochastic gradient descent is often used in neural network optimization, which solves the following optimization problem:

$$w = arg \min_{w \in \mathbb{R}^d} g(w), \quad \text{where} \quad g(w) = \mathbb{E}_{x \sim \mathcal{D}}[f(w, x)]$$

where x is a data point drawn from an unknown distribution $\mathcal{D}$, and f is a loss function. In the context of neural network, f can be represented as the following:

$$f(x) = \bar{f}_l(x) = f_l \circ \cdots \circ f_2 \circ f_1(x)$$

and

$$\mathcal{X} \xrightarrow{\Phi_1} \mathcal{F}_1 \xrightarrow{\Phi_2} \mathcal{F}_2 \rightarrow \cdots \xrightarrow{\Phi_l} \mathcal{F}_l \xrightarrow{A} \mathcal{Y}$$

where $\mathcal{X}$ is the original input domain, $\mathcal{F}_i$ is the i th latent domain, and $f_l : \mathcal{F}_{l-1} \rightarrow \mathcal{F}_l$.

Define the overall feature map as $\overline{\Phi}_l$. Then, we have

$$\Phi(x) = \overline{\Phi}_l(x) = \Phi_l \circ \cdots \circ \Phi_2 \circ \Phi_1(x)$$

So, in terms of feature map, we have $f(x) =< A, \overline{\Phi}_l(x) >$, where A is the linear map and x is a data point that is evaluated. In this view, we parameterize each feature map with a compound of non-linear and linear function: $\Phi_l = \sigma_l \circ A_l$. Examples of $\sigma(z)$ include sigmoid function, tanh function, and ReLU function.

The regular update equation for stochastic gradient descent follows

$$w_{t+1} = w_t - \alpha \nabla_{w_t} f(w_t, x_t)$$

When $||\nabla f(w_t, x_t)||$ is below a predefined threshold, we switch it to our algorithm described in the next section. Let v be the output of our algorithm, which is an eigenvector that is pointed to a descent direction. Then, the modified parameter update equation is

$$w_{t+1} = w_t - \alpha v$$

In our experiment, we use convolutional neural network, which is often used in computer vision problems. Using the above notations, convolutional neural network can be viewed as assigning some restriction on a feature mapping A_l. More precisely, we set A_l to be a group of matrices stacked row-wise, where each matrix is a cyclic matrix. This has the same effect as convoluting filters (= each matrix) with input images.

3 The proposed algorithm and its implementation

3.1 How to find a descent direction using the maximum eigenvalue

The object function can be approximated by the 2nd order Taylor expansion as follows:

$$f(\theta + \Delta\theta) = f(\theta) + \Delta\theta \nabla f(\theta) + \frac{1}{2}\Delta\theta^T H \Delta\theta$$

When θ is close enough to a critical point θ^*, the above approximation is reliable, and the gradient vanishes, so we yield

$$f(\theta^* + \Delta\theta) = f(\theta^*) + \frac{1}{2}\Delta\theta^T H \Delta\theta$$

If we denote the eigenvectors of the Hessian by $e_1, ..., e_n$ and the corresponding eigenvalues by $\lambda_1, ..., \lambda_n$, since H is symmetric, we can express $H = \sum_{i=1}^n \lambda_i e_i e_i^T$. So, we have $\Delta\theta^T H \Delta\theta = \Delta\theta^T (\sum_{i=1}^n \lambda_i e_i e_i^T)\Delta\theta = \sum_{i=1}^n \lambda_i (e_i^T \Delta\theta)^2$

By making a change of coordinates into the space spanned by these eigenvectors, we have

$$\Delta v = \frac{1}{2}\begin{bmatrix} e_1^T \\ e_2^T \\ \vdots \\ e_n^T \end{bmatrix} \Delta\theta \tag{1}$$

By combining these results, we obtain

$$f(\theta^* + \Delta\theta) = f(\theta^*) + \frac{1}{2}\sum_{i=1}^n \lambda_i (e_i^T \Delta\theta)^2 = f(\theta^*) + \frac{1}{2}\sum_{i=1}^n \lambda_i (\Delta v_i)^2$$

which shows that when the eigenvalue is negative, the corresponding eigenvector decreases the object function.

Therefore, the problem is how to find a negative eigenvalue efficiently. The proposed algorithm addresses this issue by the following:

1. After the norm of gradient gets small enough, we calculate the largest eigenvalue of the Hessian at that point. Let this value be M. Note that we can achieve this by one call of power method.

2. Calculate the largest eigenvalue of $MI - H$, where I is an identity matrix. Let this value be L.

3. If $L > M$, then that means there exists a negative eigenvalue, with the size of $|L - M|$ approximately.

This process requires two calls of power method, and does not require all the eigenvalues of the Hessian in order to determine a descent direction, which is computationally efficient. Formally, it is described in section 3.4.

Since the exact computation of hessian is computationally intractable, and moreover what we need is a hessian/vector product Hv, we used the following finite difference approximation in our implementation:

$$\nabla^2 f(x_{k+1}) = \frac{\nabla f(x_{k+1} + \epsilon s_k) - \nabla f(x_k)}{\epsilon}$$

where

$$\epsilon = \frac{2\sqrt{\epsilon_m}(1 + ||x_{k+1}||)}{||s_k||}$$

and ϵ_m is a machine precision (1^{-16} if double and 6^{-8} if float) and x_{k+1} is the current point, and s_k is the vector at the k th iteration in the power method. This is from [5]

3.2 Algorithm

```
Data: $\{(x_i, y_i)\}_{i=1}^m$
Result: parameter vector $\theta$
initialize $\theta$ while $\theta$ not converged do
    for every mini batch mb do
        for i = 1,.., |mb| do
            u ← the norm of $\frac{\partial f(\theta)}{\partial \theta}$
            if $u < \epsilon$
            M ← compute the largest eigenvalue of $H$ via power method
            L ← compute the largest eigenvalue of $MI - H$ via power method
            if $L > M$
            Update $\theta$ with the corresponding eigenvalue of L
        end
    end
end
```

3.3 Implementation

We used Torch7 to implement the proposed algorithm. Torch7 is a neural network library based on C and a script language called Lua. Various optimization methods such as Stochastic Gradient Descent and L-BFGS are available through optim module.

All the codes are uploaded on `https://github.com/yutaroyamada/hessian/blob/master/`

4 Experiment

To test our algorithm, we used MNIST, a dataset of handwritten digits, and CIFAR-10, tiny colar images dataset.

4.1 Experiment 1: First test

For the sake of computational speed, we used 2000 samples of MNIST data as a training set for our first test to compare our algorithm with regular SGD.

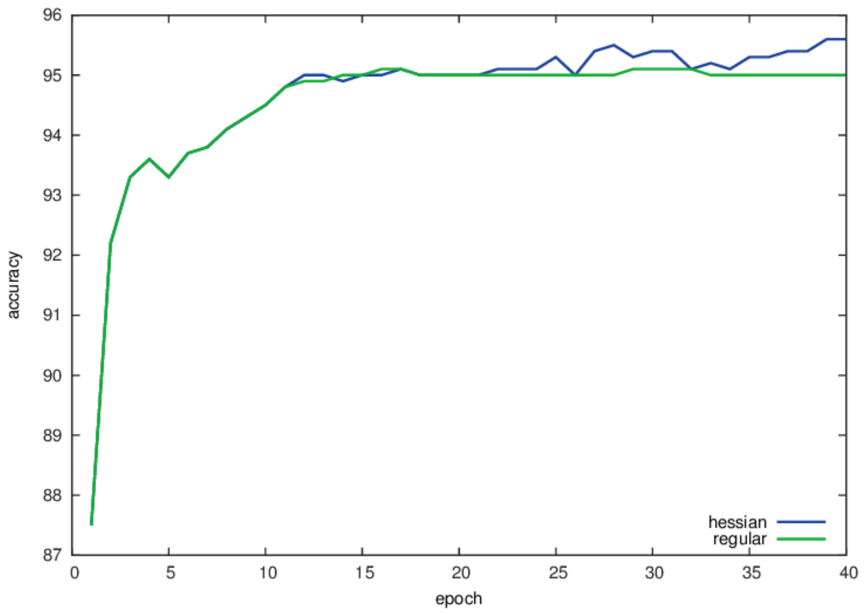

Figure 1: comparing our algorithm with the regular stochastic gradient descent

We see the promising result; the proposed algorithm is activated after epoch 10 and increases the accuracy in comparison to the regular method.

Next, we increase the size of the dataset.

4.2 Experiment 2:

We first performed a parameter experiment in order to determine the best learning rate using the MNIST dataset. We tested learningRate * $\{1,5,10,15,20,25\}$. The result is shown in Figure 2. We see that learningRate * 5 works best.

Table 1 shows the result of the experiment on MNIST using the proposed algorithm. We recorded the number of parameter updates, and for each update, we checked whether or not the norm of gradient is close enough (a), if there exists a negative eigenvalue (b), and whether or not the cost function increases (c and d).

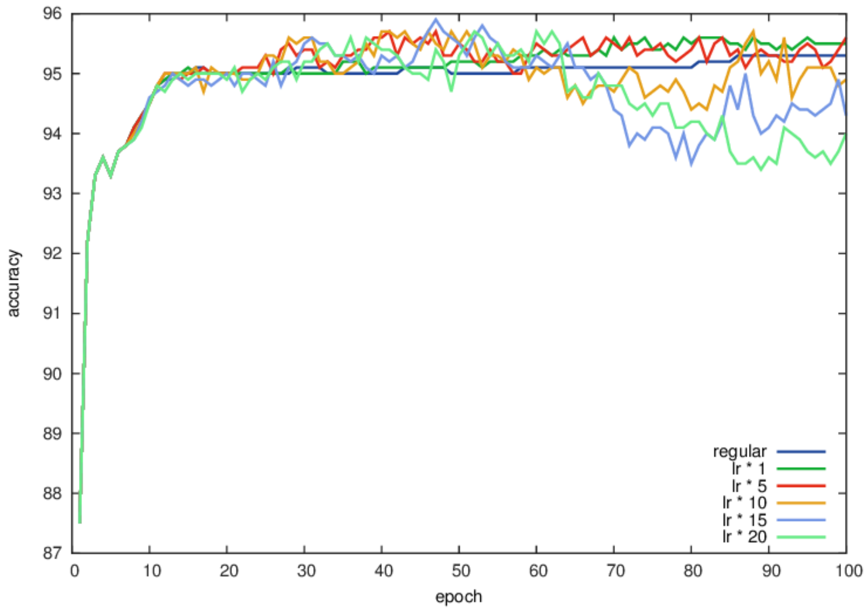

Figure 2: Parameter experiment

Table 1:

	The number of the cases in which...	
(a)	The norm of gradient is close to zero	26728 / 90000
(b)	The second test passed (L $>$ M)	26672 / 90000
(c)	The cost function decreases	12531 / 90000
(d)	The cost function increases	14141 / 90000

(a) and (b) demonstrate that for full mnist, whenever the norm of gradient is close to zero, the algorithm is actually near a saddle point instead of a local minimum since the number in the second column is almost the same. This observation matches with the results of Dauphin's paper.

(c) and (d) show that more than half of the time, the algorithm increases the loss, instead of decreasing it, which is not what we expects.

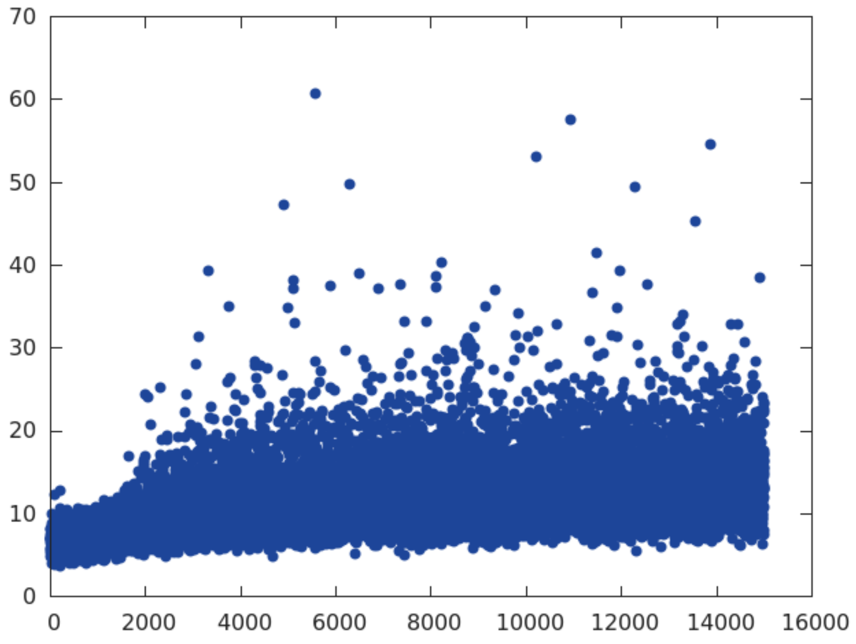

Figure 3: CIFAR-10 for 100 epoch. Using 50000 samples. We see that the gradient is not converged.

We also performed an experiment on CIFAR-10. Figure 3 shows that the gradient does not converge, and requires more investigation of parameters.

4.3 Experiment 3:

In order to test the efficiency of our algorithm, we want the size of the gradient to be close to zero. The previous section shows that the gradient will not converge for CIFAR-10, and we observed that it is overfitting. In order to address this issue, we get rid of one of the fully connected layers from our initial model, which reduces the parameter size from 105506 to 54666 for the mnist net and from 846890 to 90282 for the cifar net. (Note: generally in neural network model, what takes up the majority of the parameter size is the fully connected layer.)

The unstability of the descent direction can also be attributed to our bad approximation of Hessian. This unstability can be mitigated by increasing batch size; we evaluate gradient per batch as an approximation of the whole dataset in an exchange for computational efficiency and randomness, but the result is a noisy approximation of gradient. For the sake of our purpose, we can use the entire dataset to test our algorithm instead of the mini-batch method.

The biggest issue that the previous section reveals is that more than half of the time, our algorithm increases the loss, instead of decreasing it. This suggests that either the computation of the direction is very noisy and/or the stepsize is too big. The first issue can be solved by using the whole dataset instead of a minibatch as discussed above. The second issue could pose a larger problem. We will test three methods to address this stepsize issue.

1. Learning rate * Constant

We will test the following cases learning rate * $\{2^{-3}, 2^{-2}, 2^{-1}, 2^{0}, 2^{1}, 2^{2}\}$

2. Newton-like method : learning rate devided by $|lambda|$

This will have a similar effect as the Newton method.

3. Line search.

Since we are locally approximating the loss function as a quadratic, the eigenvector that is returned by the power method could be either direction (both should be equally good because of a quadratic nature). However, in practice, we are not exactly at a saddle point, so it would be worth trying both directions in Line search.

4.3.1 The norm of gradients in CIFAR-10 with different minibatch size

This experiment shows how the norm of gradients will become accurate as we increase the size of minibatch in CIFAR-10. The results show that the norm of gradient gets close to zero as the training goes on. We experimented with minibatch size 256, 512, and 2000. We also varied the architecture of models such as adding Dropout and replacing activation function with ReLU. (Due to the limitation of space, we will omit adding the corresponding plots.)

4.3.2 Line search v.s. baseline

We used batchsize 200 with 100 epoch and threshold of gradient norm to be 0.01.

They look almost the same. We suspect that this is because the number of times that the algorithm was called was very small, as confirmed by the information below.

The number of the case in which...

a) The norm of gradient is close to zero : 62/30000

b) The second test passed $(L > M)$: 62/30000

c) The cost function decreases: 62/30000

d) The cost function increases: 0/30000

This suggests that the threshold of gradient norm might have been too big. Possible solutions include: 1. to make the gradient norm threshold bigger. But this could potentially be harmful because it means we are switching to our algorithm while SGD could still work. 2. to increase minibatch size to have better approximation of gradient. This is a trade-off between sacrifying the number of parameter updates and obtaining better approximation of gradient. 3. reduce more parameters in the model.

4.3.3 Line search, Newton-like method, LearningRate*constant comparison

Reflected on the above issues, we performed another experiment. Figure 4 shows the result.

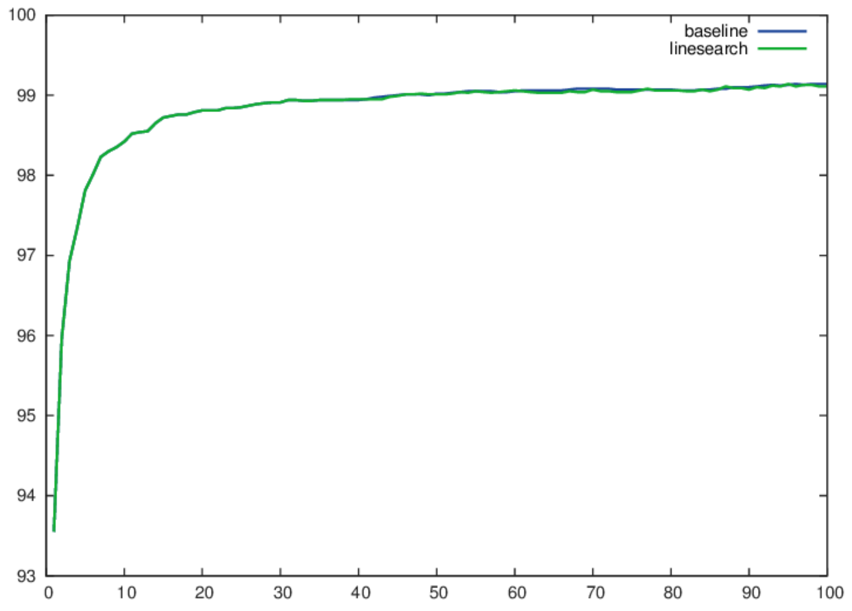

Figure 4: Line search does not affect our result.

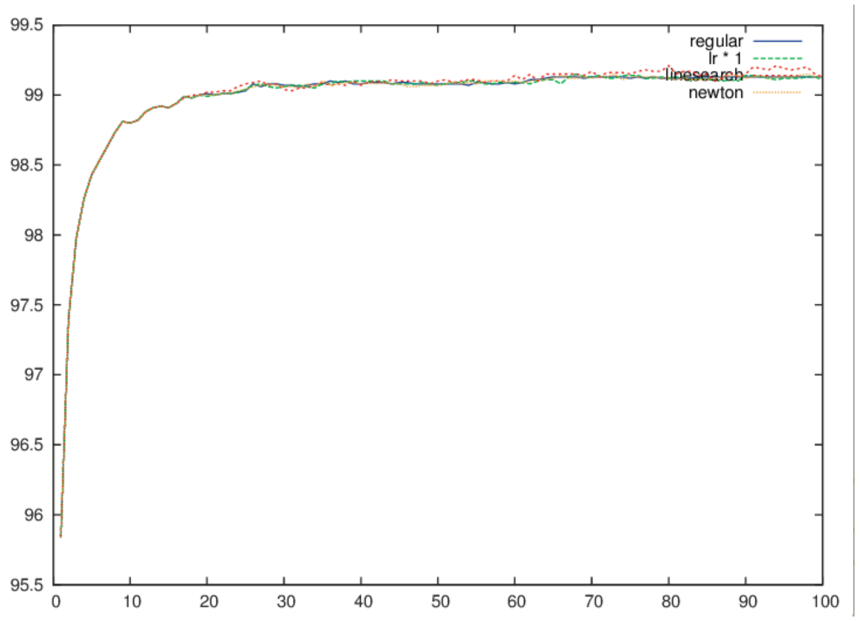

Figure 5: modified experiment

We see that Linesearch (red) works better than the other methods, but its effect is very marginal.

5 Summary and Future work

We implemented the proposed algorithm, and incorporated it into a regular stochastic gradient descent routine in neural network training. The first experiment shows the promising result with the reduced MNIST dataset. However, when we increase the size of the dataset, the proposed algorithm did not demonstrate enough improvement. We also observe that for CIFAR-10, the norm of the gradient did not converge. We found that a bad approximation of gradient causes serious issue in terms of the convergence of the norm of gradients, which means that the use of full dataset is necessary to obtain satisfying results. Line search worked best among the other methods that we tested. Yet, the same issue remained.

Given that the above experiments did not produce satisfying results, we suspected that the power method might cause numerical instability in our program. For better numerical stability, we decided to use Lanczos method instead of Power method for finding the maximum eigenvalue.

In order to compare our results with the ones of Dauphin [2], we prepared two other models: Deep Autoencoder and Recurrent Neural Network. We tested our algorithm on these two models with Lanczos method as the maximum eigenvalue finding routine. Although the loss function decreases almost always whenever the algorithm calls our method, the overall accuracy did not increase enough, which seems to be the case that our method still gets stucked with another saddle point. We are still investigating potential cause of the issue. It could be another numerical issue or a simple bug in our code. We will continue doing this project during the summer, and hope to produce some progress soon.

References

[1] K. He, X. Zhang, S. Ren, J. Sun *Deep Residual Learning for Image Recognition*

[2] Y. Dauphin et, al. *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization* : http://arxiv.org/pdf/1406.2572.pdf

[3] A. Choromanska et, al. *The Loss Surfaces of Multilayer Networks* : http://arxiv.org/pdf/1412.0233v3.pdf

[4] S. Ioffe, C. Szegedy *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift* : http://arxiv.org/pdf/1502.03167.pdf

[5] Neculai Andrei *Accelerated conjugate gradient algorithm with finite difference Hessian/vector product approximation for unconstrained optimization*