이학석사 학위논문

Analysis of Gradients of Loss Function in Deep Learning Networks

(딥러닝 네트워크에서 손실함수의 그래디언트 분석)

2019년 2월

서울대학교 대학원

수 리 과 학 부

강 두 영

Analysis of Gradients of Loss Function in Deep Learning Networks

(딥러닝 네트워크에서 손실함수의 그래디언트 분석)

지도교수 강 명 주

이 논문을 이학석사 학위논문으로 제출함

2018년 10월

서울대학교 대학원

수 리 과 학 부

강 두 영

강 두 영의 이학석사 학위논문을 인준함

2018년 12월

위 원 장 _____ (인)

부 위 원 장 _____ (인)

위 원 _____ (인)

Analysis of Gradients of Loss Function in Deep Learning Networks

by

Duyoung Kang

A DISSERTATION

Submitted to the faculty of the Graduate School
in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Mathematical Sciences
Seoul National University
February 2019

Abstract

Analysis of Gradients of Loss Function in Deep Learning Networks

Duyoung Kang

Department of Mathematical Sciences
The Graduate School
Seoul National University

It is widely known that training with small batch size tend to converge to flat minimizers of loss function while large batch method does not. It is also known that sharp minima generalize worse than flat minima, so relation between batch size and generalization is crucial issue among researchers. Smith found that the smaller batch size, the more noise is introduced in gradients during training. Wen suggested a new method to train that is good at avoiding sharp minima under less effect of batch size. Keskar guessed that 'piggybacked method', which is training with large batch after some epochs of small batch methods, generalize better than original small batch method. In this work, we focused the 'piggybacked method' in Keskar's paper and analyzed gradients through experiments to find the reason of good generalizing ability of this method. Also, we introduce the process of getting idea of experiments with summarizing the paper of Smith and Wen.

Key words: Loss Function, Gradient, Batch Size, Deep Learning
Student Number: 2016-20226

Contents

Abstract **i**

1 Introduction **1**

2 Gradient Distribution and Smoothing Effect **5**
2.1 Notation 5
2.2 Gradient Distribution according to the Batch Size 6
2.3 Smoothing Effect 7
2.4 Idea of Experiments 9

3 Analysis of Gradients and Visualization Experiments **12**
3.1 Notation and Setting of Experiments 12
3.2 Experiments(A): Visualization of loss surface 15
3.3 Experiments(B): Gradient analysis 19

4 Conclusion **29**

The bibliography **32**

Abstract (in Korean) **34**

Chapter 1

Introduction

Deep learning is one of the most attracting technology for its high accuracy among various machine learning methods. Deep learning is currently better for image processing and natural language processing (NLP) than original methods, so many researchers are studying its field to improve the performance. However as we know from its name, 'Deep learning', this technology needs training deep network. Parameters to train in the network usually ranges from hundreds of thousands to millions, so it is actually difficult to analyze the mechanism of deep networks and the reason of outstanding accuracy clearly. Thus those analysis is one of the most interesting field among many branches from deep learning and there are various kinds of attempt to clarify its mechanism. Researchers approach theoretically for clear understanding, or sometimes empirically to get intuition of the behavior of deep networks with data from a lot of experiments.

Analysis of loss function, which shows how much the difference between target value and actual value(output) for fixed parameter and input data is, is a basic approach to understand the deep networks. Since there are usually tremendous numbers of parameters in a network, it is never simple thing to understand the loss function which lives in extremely high dimensional space. Nevertheless, there have been many kinds of approach to understand the loss function such as visualization(loss surface) and finding the relation between performance of the network and structure of the loss surface [2], [3], [4], [12].

Roughly speacking, deep learning is training the parameters through back-

propagation [16], and the training can be seen as a process of finding a minimizer of loss function(or finding a minima in loss surface). Mathematically, if we say that the number of training data is M and loss function of i-th datum is f_i, total loss function f would be as below.($i \in \{1, 2, \ldots, M\}$)

$$f(w) = \frac{1}{M} \sum_{i=1}^{M} f_i(w), \tag{1.1}$$

where w is training parameters(can be seen as a vector). Then deep learning can be seen as a process of finding w_0 which minimizes the loss function f.

$$w_0 = \arg\min_{w} f(w), \tag{1.2}$$

Backpropagation refers to a method of iteratively updating the parameter w to find such a minimizer w_0. The most basic method is to extract the gradient of the loss function about the data belonging to a certain batch of the training data at current parameter and to update the parameter along the reverse direction of that gradient, which is called stochastic gradient descent methods(SGD) [1].

$$w_{k+1} = w_k - \alpha(\frac{1}{B} \sum_{i} \nabla f_i(w_k)), \tag{1.3}$$

where w_k is the training parameter after k-th iteration, α is learning rate and B is size of a batch which summation above is conducted in.

The simplest case is when the batch size B is equal to the number of whole training data, M(full-batch case). In this case, a gradient of the total loss function f is extracted and become involved in updating the training parameter. However full-batch method is never conducted for practitioners because of the limit of computing power and the probability of converging to a local minimum of the loss function which is highly non-convex. Thus B is usually set far smaller than M ($M \gg B$) and a lot of research show that this small batch method usually converges to a good minimizer which generalize well through theoretical and empirical analysis [5], [6], [11]. Also, limit of computing power can be overcome through small batch method, so

SGD and its variants with small batch have been widely used currently. In addition, since large batch method also has advantages with the increase of ability of parallelization, researchers are still interested in training with large batch size which occupies about 3∼10% of total training data as well as small batch size which occupies under 3%.

However it is continuously found that the generalization is not good when training the network with large batch size compared with the small batch case [8]. In other words, both methods achieve high accuracy on training set but small batch method is clearly better on test set than large batch method. Many researchers says that this gap happens because small batch method usually converges to flat minima which is less sensitive to test set while large batch method converges to sharp minima in loss surface. Below is the figure that explains about this opinion in Keskar's paper [8].

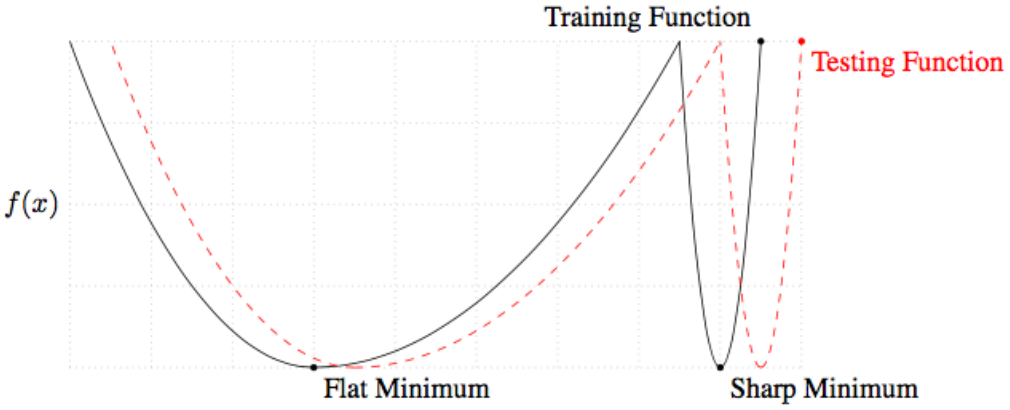

Figure 1.1: The figure explaining the difference between a sharp minimum and a flat minimum in [8]

.

In the paper, authors visualized the loss surface in practice along the line passing through two parameters, one is the final point after small batch method converges, and the other is a point after large batch method is complete. With the graph, they guess that a gradient extracted from small batch may have more noise so tend to escape sharp minima, while a gradient from large batch may have less noise so tend to converge to sharp minima which is usually close to the starting point. They also show that these sharp min-

ima actually achieve bad accuracy on test set empirically, thus batch size and generalization have inverse proportion. Nevertheless they say that the property that a gradient from large batch may have less noise could be somehow useful. They try an experiment called 'piggybacked method' which is the training with large batch after some epochs of training with small batch, and they suggest these may achieve better accuracy on test set than original methods. They conclude this could happen since initial small batch method decay the sharpness of minima and subsequent large batch method which seems to have less noise would help achieving high accuracy on test set.

In this thesis, we focused on the piggybacked method mentioned in Keskar's paper. In practice, calculating the distribution of gradients extracted from small batch and large batch during training a few networks and visualizing loss surface were conducted to clarify the mechanism of piggybacked method. Also, some related works which were helpful to get an idea of the experiments would be introduced. The rest of the paper is organized as follows. We introduce statistical approach to gradients of loss function and new training algorithm of deep networks that are suggested in Smith's papers[13], [14] and Wen's paper[15] respectively in **Chapter 2**, as well as theoretical explanation. We propose some idea of piggyback method and conduct experiments which analyze gradients distribution and visualize the loss surface to clarify the mechanism of the method in **Chapter 3**. We conclude in **Chapter 4**.

Chapter 2

Gradient Distribution and Smoothing Effect

2.1 Notation

Here we introduce the notation to be used in this chapter. Many things are same as the notation used in **Chapter 1**. First, we usually regard that there are M training data, so set of their index would be $\{1, 2, \ldots, M\}$. w will represent a training parameter(as a vector) in deep networks. w can be subscripted like w_0, w_k depending on the situation. B will be used as batch size, and $B = M$ in full-batch case. Loss function is the function of parameter w, and we use f_i as loss function of i-th data and f as loss function of whole training set. Since train loss function is the average of all f_i's, we can write as below.

$$f(w) = \frac{1}{M} \sum_{i=1}^{M} f_i(w). \tag{2.1}$$

We can also express k-th iteration of stochastic gradient descent(SGD) algorithm as below.

$$w_k = w_{k-1} + \alpha(\frac{1}{B} \sum_{i=i_1}^{i_B} \nabla f_i(w_{k-1})), \tag{2.2}$$

where α is learning rate, $\{i_1, \ldots, i_B\}$ is set of index of a batch from training set($\{i_1, \ldots, i_B\} \subset \{1, 2, \ldots, M\}$), and B is naturally batch size. In addition,

we use g_B and g_{train} as gradients extracted from a batch(size of B) and whole train set respectively. In other words,

$$g_B(w) := \frac{1}{B}\sum_{i=i_1}^{i_B} \nabla f_i(w), \tag{2.3}$$

$$g_{train}(w) := \frac{1}{M}\sum_{i=1}^{M} \nabla f_i(w) (= \nabla f(w)). \tag{2.4}$$

Finally the variable w can be omitted for simplicity. For example, we can use the notation f, f_i, ∇f, ∇f_i, g_B, g_{train} instead of $f(w)$, $f_i(w)$, $\nabla f(w)$, $\nabla f_i(w)$, $g_B(w)$, $g_{train}(w)$.

2.2 Gradient Distribution according to the Batch Size

Like many researchers, Keskar guessed that the smaller batch size is, the more noise the gradients have, and this noise would be helpful to avoid sharp minima in loss surface. Then Smith tried to clarify the notion of 'noise'. He defined 'noise' as the difference between a gradient extracted from a batch(g_B) and a gradient extracted from full-batch(g_{train}), and found the relation between batch size and the noise using central limit theorem [14].

Specifically speaking, say the training parameter w in the network is fixed as w_0 at the moment. We can think M gradients $\{\nabla f_1(w_0), \ldots, \nabla f_M(w_0)\}$ of each loss function f_i at $w = w_0$. Let the covariance matrix of these M gradients be $F(w_0)$. It is obvious that the gradient extracted from whole training set g_{train} is mean vector of them. Since a gradient from a batch with a size of B, $g_B(w_0)$, is the average of randomly chosen B vectors in $\{\nabla f_1(w_0), \ldots, \nabla f_M(w_0)\}$, we can say $g_B(w_0)$ is a random vector following a Gaussian probability distribution with mean g_{train} and covariance $(\frac{1}{B} - \frac{1}{M})F(w_0)$ by multidimensional central limit theorem. Usually B is chosen much smaller than $M (B \ll M)$, so the covariance of this probability distribution is approximately $\frac{1}{B}F(w_0)$. Thus as B gets smaller, covariance F becomes larger, so g_B will be extracted much further from g_{train}, and it means the

noise gets larger. We can write it as below.

$$g_B = g_{train} + \eta, \tag{2.5}$$

where η is noise which is inversely proportional to batch size B and affected by covariance at current location.

In this thesis, authors say that keeping this noise scale above certain amount is crucial to avoid sharp minima, thus small batch training is better than large batch training. Then some of those authors published additional paper with other researchers, saying that since practitioners often decay the learning rate as the training epochs increase, this noise scale would decrease but it can happen through increasing batch size, instead of decaying learning rate [13]. They conclude that with parallelization, increasing batch size would be faster than decaying the learning rate, but there might be some accuracy degradation.

2.3 Smoothing Effect

In Smith's paper [14], it was good approach to clarify what the noise is and its relation with batch size. Wen, however, pointed that even though the noise can be analyzed, relating the noise and sharpness of minima is still lack of theoretical validity. Thus Wen suggested new training algorithm which can explain why it avoids sharp minima well theoretically [15].

Algorithm 2.1 (Smoothout, in Wen [15])**.** *Let the training parameter w is a vector in $\mathbb{R}^n$ and θ be a random vector which follows a probability distribution $U(-a, a)$ that is uniform on $[-a, a]^n$. a is hyper-parameter. Then k-th iteration of* ***smoothout*** *algorithm is*

$$w_k = w_{k-1} + \alpha(\frac{1}{B}\sum_{i=i_1}^{i_B} \nabla f_i(w_{k-1} + \theta)), \tag{2.6}$$

where α is learning rate and $\{i_1, \ldots, i_B\}$ is an index set of a batch from training set.

For simplicity let's consider when $B = M$(It is similar when $B < M$). In this case, **smoothout** algorithm would be

$$w_k = w_{k-1} + \alpha(\frac{1}{M}\sum_{i=1}^{M}\nabla f_i(w_{k-1}+\theta)) = w_{k-1} + \alpha(\nabla f(w_{k-1}+\theta)). \quad (2.7)$$

Remember f is loss function of whole training set. In the equation above, $\nabla f(w_{k-1}+\theta)$ is usually thought as the gradient of $f(w)$ at $w = w_{k-1}+\theta$, that is

$$\nabla f(w_{k-1}+\theta) = \frac{df(w)}{dw}|_{w=w_{k-1}+\theta}. \quad (2.8)$$

By the way, since

$$\frac{df(w)}{dw}|_{w=w_{k-1}+\theta} = \frac{df(w+\theta)}{dw}|_{w=w_{k-1}}, \quad (2.9)$$

it can be thought as the gradient of $f(w+\theta)$, which is translated function of original f, at $w = w_{k-1}$.

In other words, **smoothout** algorithm is similar with (stochastic) gradient descent, but the difference is that when extracting gradient, loss function f is translated by θ. θ is a random vector so each iteration f would be translated in random direction. It might be easier to understand if we think that parameter w regards the loss function as a random function. Then, the expectation of θ is zero vector so the expectation of randomly perturbed loss function $f(w+\theta)$ would be the average of those uniformly translated functions, which is **smoothed** function of the original. If we let the expectation be $\bar{f}(w) = E(f(w+\theta))$, we can say **smoothout** algorithm update parameters using the gradient of $\bar{f}(w)$, which is smoothed from original loss function $f(w)$, in average. Below is the figure explaining this situation which was used in Wen's paper [15].

From the figure below, we can see that sharp minima of f is weak for **smoothout**, while flat minima is not. Therefore the gradients extracted during the algorithm usually help the parameter to converge to the flat minima. Also, as Wen said in his paper, this mechanism is more transparent theoretically, that is better to understand why training parameter avoid sharp minima well.

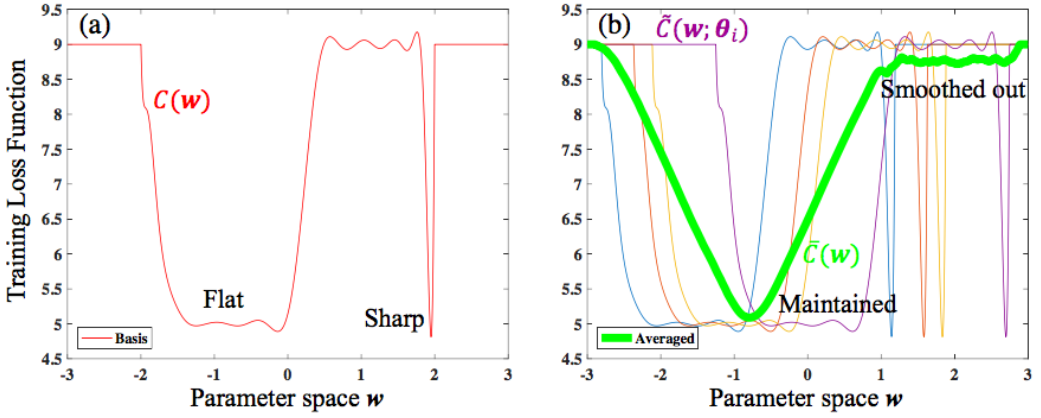

Figure 2.1: The figure explaining smoothed loss function $\bar{f} = E(f(w+\theta))$ in [15]. In this figure, C is same as original loss function f (red line, left) and $\bar{C}$ is same as $\bar{f}$ (green line, right). Other lines show many perturbed loss function by θ, that is a random vector.

2.4 Idea of Experiments

Now, we think the gradient $\nabla f(w+\theta)$ which is main factor in **smoothout** algorithm. When a is small enough(recall that $\theta \sim U(-a,a)$), θ would be so. Then we can apply Taylor's approximation for the gradient(First order).

$$\nabla f(w+\theta) \approx \nabla f(w) + \theta^T(\nabla^2 f(w)) \tag{2.10}$$

In the equation above, $\nabla f(w)$ is actually the gradient extracted from whole train set, which is g_{train}. Thus if we regard the last term as noise, we can think the equation above is similar with the equation (2.5). Obviously, $\theta^T(\nabla^2 f(w))$ and η(in (2.5)) are different noise. The former noise is proportional to θ(or a) and the curvature of loss surface $\nabla^2 f$, and the latter noise is inversely proportional to batch size B and related with the covariance matrix F of the gradients at the current point. Nevertheless we can guess that original stochastic gradient descent method is a kind of **smoothout** (but the way of smoothing loss function is quite different), and the smoothing power will increase as the batch size decreases.

If we think that stochastic gradient descent is a kind of smoothing the loss

function and noise is a kind of smoothing power, the opinion that noise should be above certain amount in Smith's paper [14] or another suggestion that increasing batch size gradually is faster and has same effect with decreasing learning rate no longer seem to the best choice since noise(or smoothing) could be helpful to find a flat minimum but it would not be helpful anymore after entering in the minimum. Thus we pointed out that the those kinds of training process should be divided into two process. First process should be the training with large noise(or smoothe strongly) to avoid sharp minima and enter the flat one, and second one should be the training with less noise to find good minimizer in the flat minimum.

In other words, we thought that 'piggybacked method' in Keskar's paper [8], which is large batch training after a few small batch training, could be reasonable attempt. In practice, 'piggybacked method' also showed higher accuracy than original small batch method in our experiments. Remember the crucial point in our work is dividing the training process into two process: large noise step for finding flat minima & small noise step for finding better minimizer in final minimum. As we guessed, decreasing noise gradually in Smith's paper [13] is a kind of decreasing smoothing power gradually and this approach seems to be lack of enough reason; It does not seem to be the best strategy for finding flat minima and finding the best point in final minimum. In practice, the way to increase batch size gradually in Smith's paper usually showed noticeable accuracy drop on test set, compared to decreasing learning rate. It might be fast, but it could not achieve better performance. Thus we decided to analyze gradients during 'piggybacked method' to show that it is reasonable method to achieve high accuracy on test set, empirically.

More specifically, we expect the result as the figure below. First, we define piggybacked method more clearly, that is large batch training after enough epochs of small batch training. Here the word "enough" means "until the accuracy on test set(or validation set depending on the situation) converges(or does not improve any more)". Now assume we just finished the training a network with small batch and we are training with large batch. Then we guess that the gradients extracted from small batch and large batch will distribute as the figure below at the moment (To explain, we simplified the situation. We assumed gradients and parameters exist in $\mathbb{R}^2$ space). The purple point

represents g_{train} and might be located near the origin since we conducted enough training (with small batch). We can also think that g_{train} would be almost zero vector since now parameter would be on some flat minimum. The red point represents g_{total}(It will be used continuously in next chapter), which is a gradient extracted from total set(= training set + test set). We just put g_{total} near g_{train} because we have no information about total set in practice. Last, blue points and orange points represent gradients extracted from large batch and small batch respectively. As Smith says, gradients from large batch will locate more close to g_{train} with less noise. Finally our hypothesis is that during this large batch process of piggybacked method, gradients from large batch would be more helpful than those from small batch since g_{total} might be close enough to g_{train} so gradients extracted from large batch will be more distributed around g_{total} than gradients extracted from small batch. In addition, if a gradient from large batch is not close to g_{total}, it is less risky since it would be almost zero vector with higher probability than that from small batch. In practice, we got good numerical results supporting our hypothesis, and it will be introduced in next chapter.

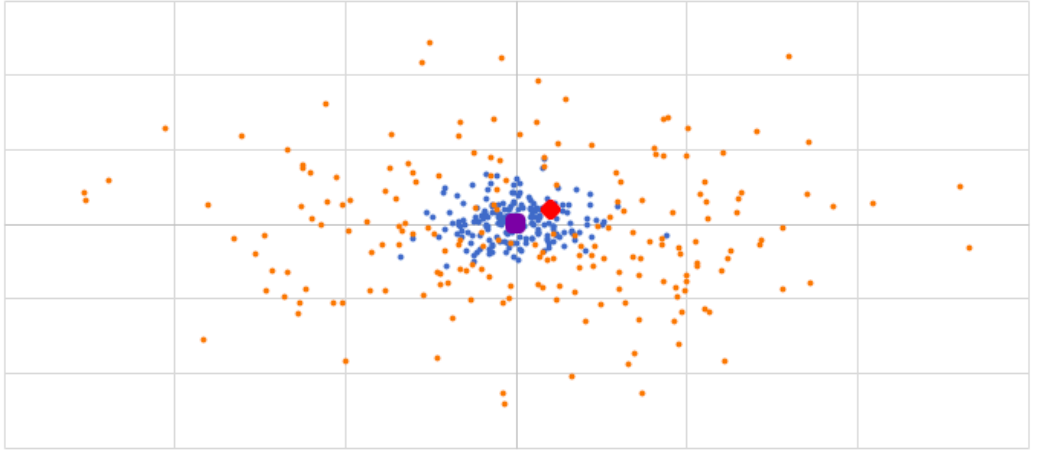

Figure 2.2: Simplification of g_{train}(purple point), g_{total}(red point), gradients from small batch(orange points) and gradients from large batch(blue points) during the large batch training of piggybacked method.

Chapter 3

Analysis of Gradients and Visualization Experiments

3.1 Notation and Setting of Experiments

Notation

All notation used in **Chapter 2** will be used in this chapter. Plus, we cover gradients extracted from total data set which includes training set and test set here. Thus we will use g_{total} for it. Also, we will use g_B with specific value of B. For example, g_{256} means a gradient extracted from a batch with a size of 256 and so on.

Setting of Experiments

We used 3 kinds of networks for experiments, which are called F_1, C_1, C_3 in Keskar's paper [8]. The reason of choosing those networks is that we needed the result to be compared with accuracy of piggybacked method. In our work, we found that piggybacked method perform obviously better than original small batch method.

For F_1 network, we use a 784-dimensional input layer followed by 5 batch-normalized [7] layers of 512 neurons each with ReLU activations. The output layer consists of 10 neurons with the softmax activation since F_1 uses MNIST data [10]. The C_1 network is consisted of 2 sets of (64 kernels with a size of

5×5 and stride 2)–(MaxPool 3) followed by 2 dense layers of sizes (384, 192) and finally, an output layer of size 10 since C_1 uses CIFAR-10 [9]. We use batch normalization for all layers and ReLU activations. The network C_3 is identical to C_1 except it uses 100 softmax outputs instead of 10, since C_3 uses CIFAR-100 data [9].

There are 2 kinds of experiments. (1)We will compare small batch method and piggybacked method through visualization of loss surface and (2) analyze gradients distribution from various batch during training. As mentioned in the last section of **chapter 2**, all piggybacked method here means large batch training after enough small batch training (after small batch training converges). We set small batch as 256, which is same as Keskar's paper and large batch as 1000, 2000, 5000. Specifically, we will train with small batch(256) enough and save training parameter, and from the save point we will train in 4 different ways, which are training with batch of size 256, 1000, 2000, 5000. Keeping batch size for 256 represents original small batch method(SGD), and training with batch of size 1000, 2000, 5000 after training with batch size 256 represents 3 kinds of piggybacked method. Since there are 3 networks, we actually conduct 12 different training. The figure and the table below explain the experiments.

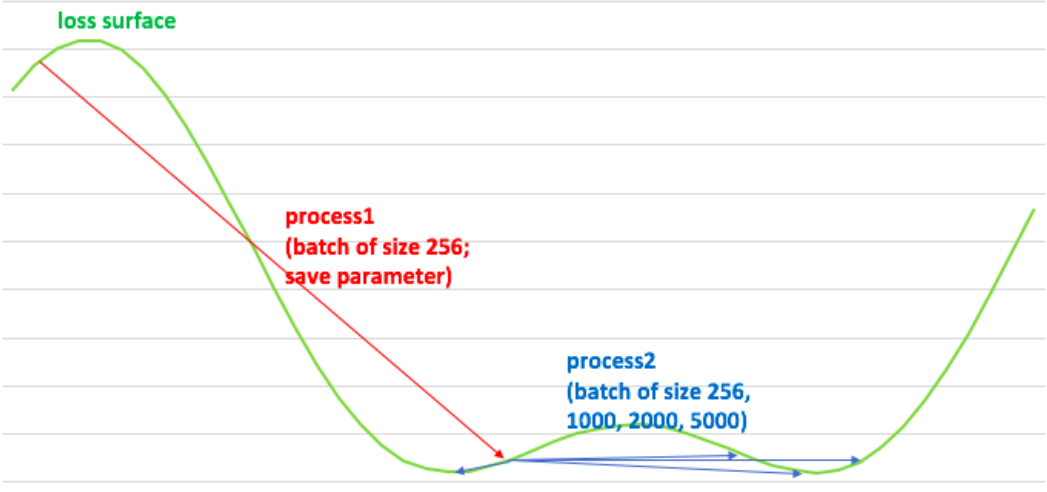

Figure 3.1: Experiment explanation (2)

		change of batch size		
		process 1	$\longrightarrow$	process 2
F_1	small batch method	256	$\longrightarrow$	256
	piggybacked method(1)	256	$\longrightarrow$	1000
	piggybacked method(2)	256	$\longrightarrow$	2000
	piggybacked method(3)	256	$\longrightarrow$	5000
C_1	small batch method	256	$\longrightarrow$	256
	piggybacked method(1)	256	$\longrightarrow$	1000
	piggybacked method(2)	256	$\longrightarrow$	2000
	piggybacked method(3)	256	$\longrightarrow$	5000
C_3	small batch method	256	$\longrightarrow$	256
	piggybacked method(1)	256	$\longrightarrow$	1000
	piggybacked method(2)	256	$\longrightarrow$	2000
	piggybacked method(3)	256	$\longrightarrow$	5000

Table 3.1: Experiment explanation (1)

3.2 Experiments(A): Visualization of loss surface

Experiment

Let $w_{256}, w_{1000}, w_{2000}, w_{5000}$ be final parameters after small batch method and piggybacked method (1), (2), (3) finished (subscripts represent size of batches from which gradients are extracted). Here, we will show the graph of loss function along the line passing w_{256} and $w_n (n = 1000, 2000, 5000)$. Specifically, we will calculate $f((1-\alpha)w_{256} + \alpha w_n)(n = 1000, 2500, 5000, -2 \leq \alpha \leq 3)$ and plot it. As the graphs show below, we can think that large batch training during piggybacked method is no longer attracted to sharp minima in loss surface. Rather, it is reasonable to think that large batch training during piggybacked method is a process finding better minimizer in the flat minimum which initial small batch method entered since the final parameter from piggybacked method shows higher accuracy on total set than that from small batch method, and does not live in a sharp minimum.

Result

We show 6 results which are visualizations of loss function of network $C1$. The first 3 figures are graphs of same loss function along different lines below(That is, we found the final parameters from same process1). For example, the first one is plotted along the line passing w_{256} and w_{1000}. Then the next 3 figures are loss surfaces along w_{256} and w_{5000} from different experiment.(That is, we initialized parameters differently).

As we can see in figures below, both final parameters which are found through normal small batch method and piggybacked method respectively seem to live in the same flat minimum. Also, It doesn't seem that large batch training of piggybacked method is attracted to sharper minima than small batch method. However, piggybacked methods always show the same or greater accuracy on test set than the original method. More specifically, the accuracy on test set obtained by the piggybacked method was usually 0 to 1 percent higher than the accuracy obtained by normal small batch method.

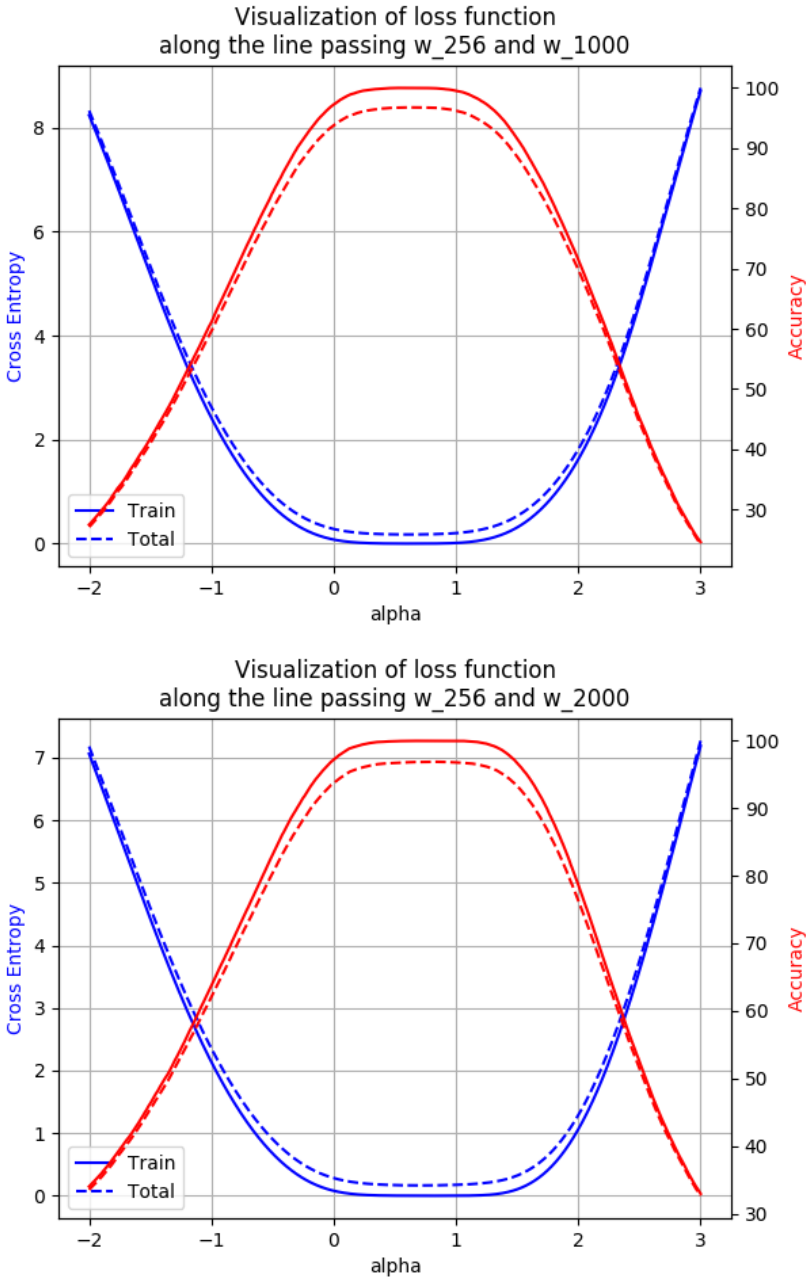

Figure 3.2: Result of Experiment(A)

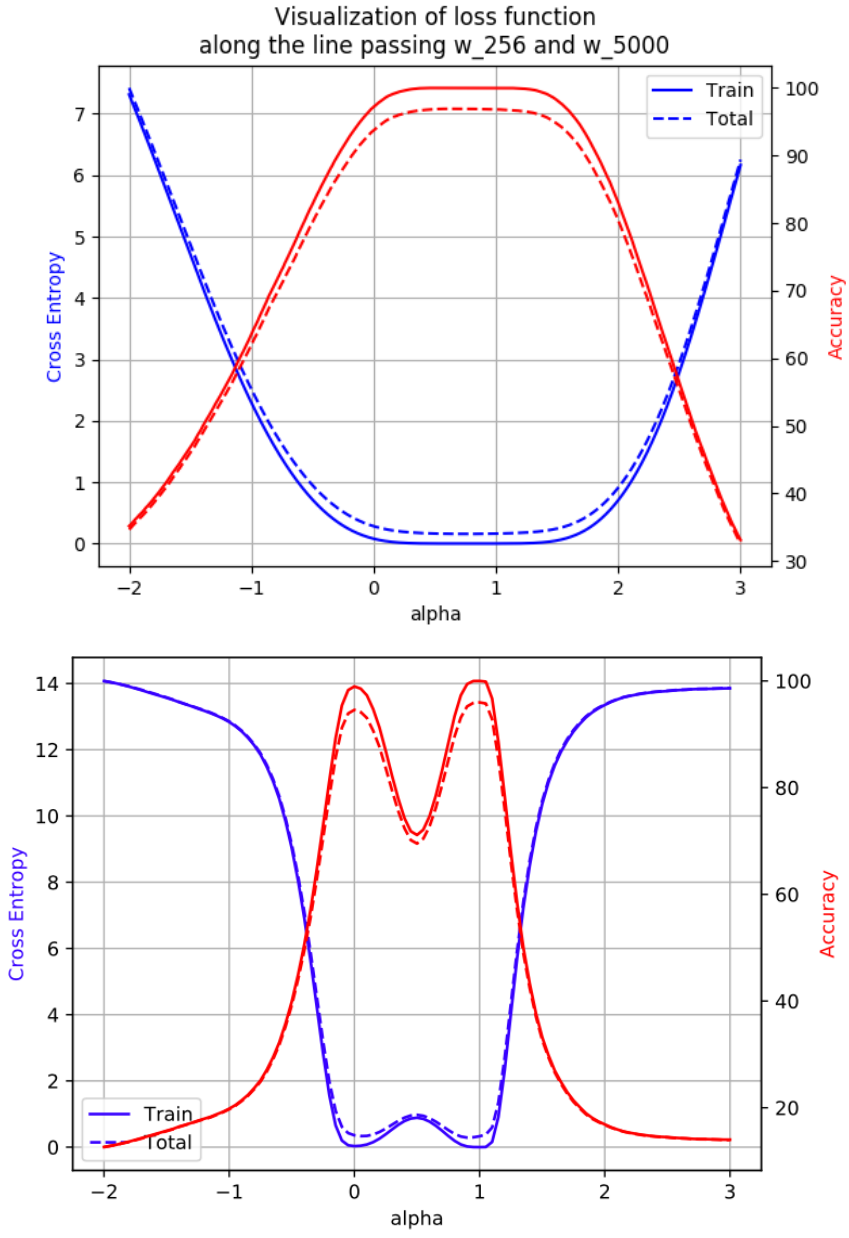

Figure 3.3: Result of Experiment(A)

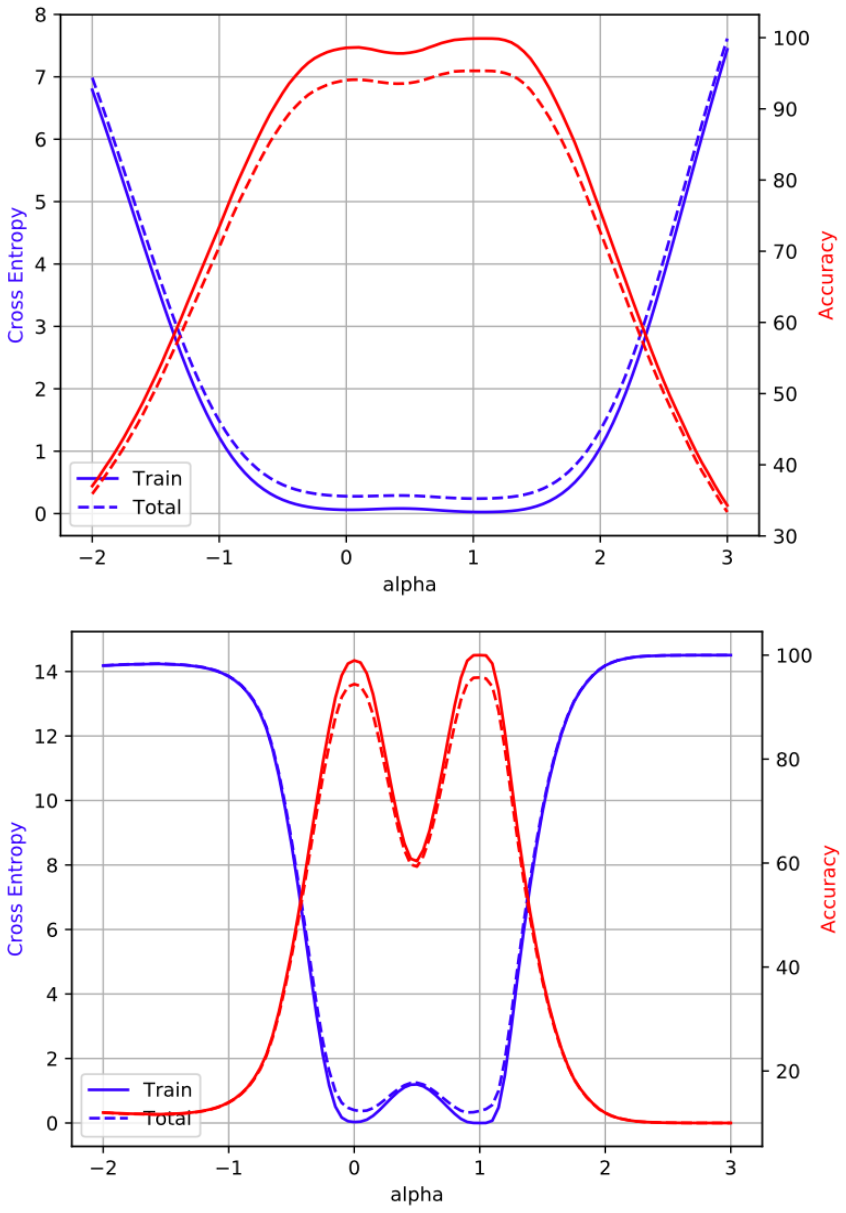

Figure 3.4: Result of Experiment(A)

3.3 Experiments(B): Gradient analysis

Experiment

As we explained above, we divide the piggybacked training into two process. Process1 is small batch training that parameter is finding a flat minimum and process2 is large batch training that parameter is finding a better minimizer in the flat minimum. Here, we will show various gradient distributions depending on the batch size around g_{train} and g_{total} during process2 of piggybacked training. We will explain how the gradients analysis is conducted in detail with figures below. Now assume we just finished process1. Training parameter might find some flat minimum.

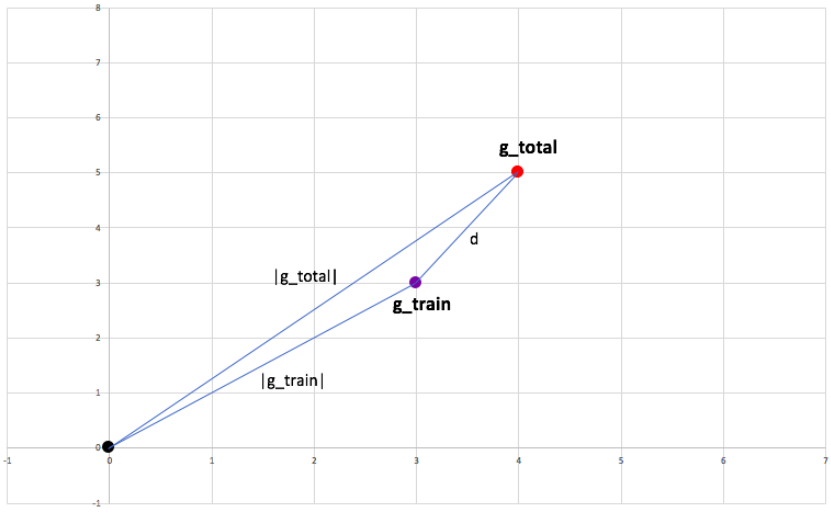

Figure 3.5: Explanation of Experiment(B)

At the moment we can get g_{train} and g_{total} from data and current parameter. Ultimately we want to know how gradients from small batch and large batch are distributed and we expect that the reason of high performance of piggybacked method is that gradients distribution from large batch is more suitable for extracting a gradient near g_{total} which really helps us to get higher accuracy on test set. Before finding the distribution, first we need the unit for calculating norm and distance. For example, if what we assumed is true, g_{train} should be near the origin but we can not judge g_{train} is near the zero

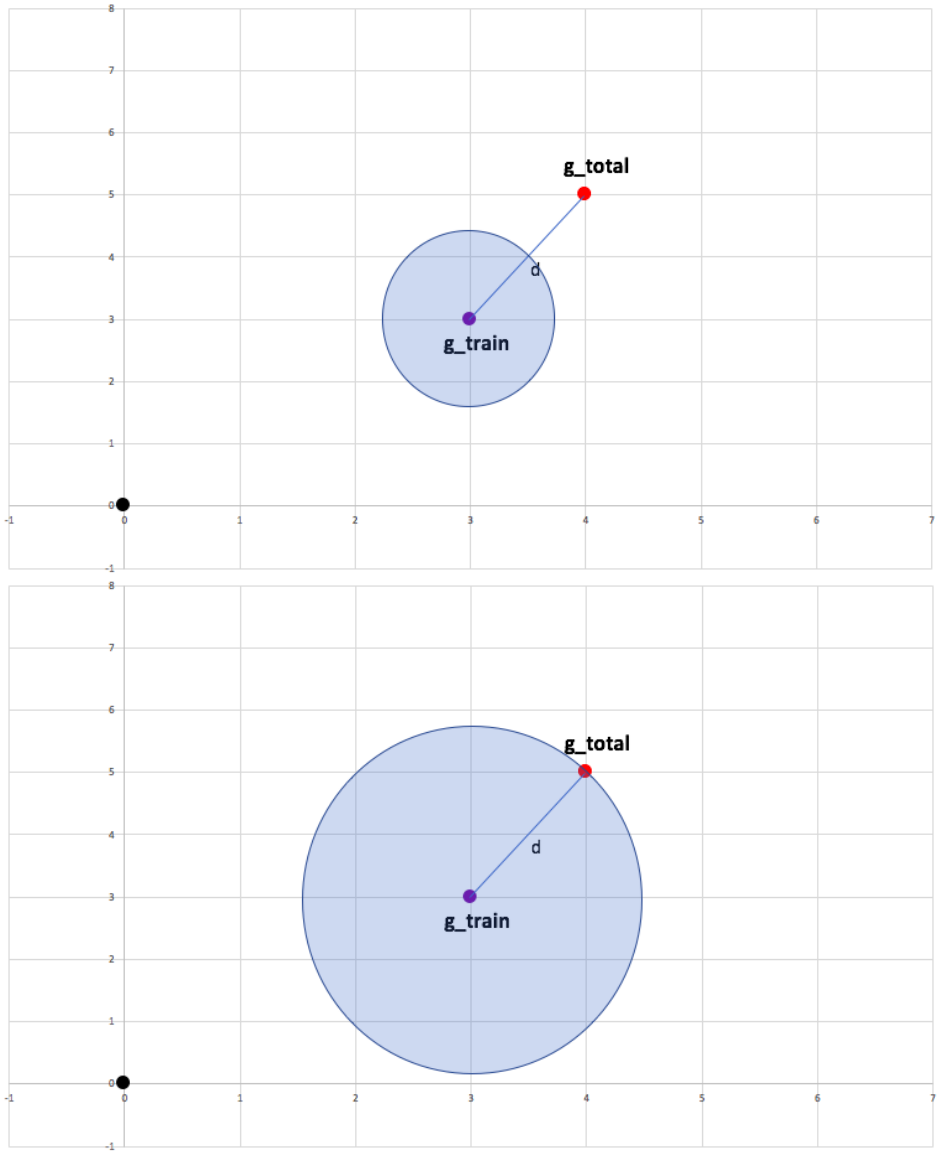

Figure 3.6: Explanation of Experiment(B)

vector even though we have the exact norm of g_{train}. Thus we decided to take the unit of measuring the norm or distance as $d = |g_{train} - g_{total}|$. From now, questions such as "how far are g_{train} and the origin?" or "how far are g_{256} and g_{total}?" would be answered and judged based on d. Anyway the first step

of this experiment is to calculate d and to check $|g_{train}|$ is near zero(based on d). Also, calculate $|g_{total}|$ to compare with $|g_{train}|$.

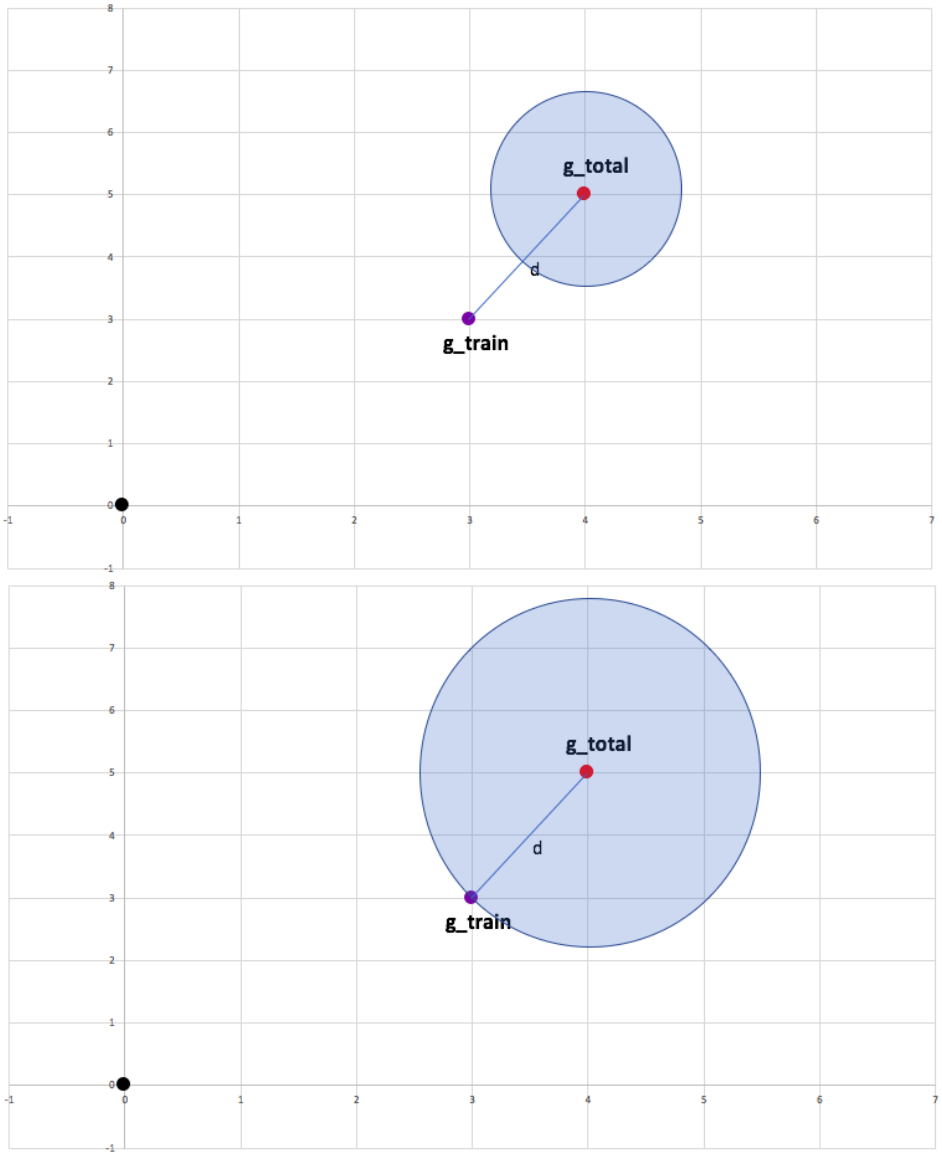

Figure 3.7: Explanation of Experiment(B)

Next, we will extract $g_{256}, g_{1000}, g_{2000}, g_{5000}$ from training data, 100 times for each kind of gradient. As Smith said in his paper, the less batch size is,

the closer the gradients from the batch are to g_{train}. we will check Smith's research based on d. In other words, for example we will count how much g_{256} is distributed $0d$ to $0.5d$ from g_{train} and how much g_{256} is distributed $0.5d$ to $1d$ from g_{train} and keep going. Surely, we will repeat same thing for $g_{1000}, g_{2000}, g_{5000}$.

Finally, we will check what we guessed is correct. If what we expected is correct, g_{256} is less distributed near g_{total} and $g_{1000}, g_{2000}, g_{5000}$ are generally more distributed around g_{total}. Thus as above, we will count how much g_{256} is distributed $0d$ to $0.5d$ from g_{total} and how much g_{256} is distributed $0.5d$ to $1d$ from g_{total} and keep going. Likewise, we will repeat same thing for $g_{1000}, g_{2000}, g_{5000}$.

This is all of the gradients analysis. However, the important thing is that what we have just described is just one analysis after process1. we will repeat the whole thing as described above each epoch during process2. And finally, We expect the general trend that gradients extracted from large batch($g_{1000}, g_{2000}, g_{5000}$) are more distributed around g_{total} than gradients extracted from small batch(g_{256}).

Result

Table 3.2 and 3.3 show the distributions of gradients while process 1 of piggybacked method is conducting in $C1$ network. As we mentioned above, 60 epochs are enough for the training parameter to converge when small batch training is conducted for $C1$. Thus we trained 60 epochs in process 1 and we trained 10 epochs in process 2 and Table 3.2 and 3.3 shows the gradients from small batch(256) and large batch(5000) every 10 epochs in process 1. As many researchers including us expect, (1)gradients from large batch are very close to g_{train} and (2)norm of g_{train} is relatively big and is decreasing gradually. Plus, we can know that g_{total} is generally far closer to g_{train} than to origin, so its norm is also relatively big. Then, it seems that large noise from g_{train} doesn't interrupt training for this reason.

Table 3.4,3.5 and 3.6 show the distributions of gradients during process 2 of a piggybacked method in $C1$ network. This time, we extracted gradients from 4 kinds of batch, which are a size of 256, 1000, 2000 and 5000. We can see that (1)norm of gradients are relatively small, actually norm of g_{train} is almost $0d$ and norm of g_{train} is almost $1d$, and (2)gradients don't decrease continuously in norm from the tables of figures below. As we expected in previous chapters, during process 2, large noise is obstacle for training since norm of g_{train} is very small. Especially, we focused how many gradients distributed ranged from $0d$ to $1d$ around g_{total} since norm of g_{train} is almost $0d$ and norm of g_{train} is almost $1d$. Then we think figures below show good results that gradients from large batch(1000,2000,5000) distribute more near $g_{total}(0d \sim 1d)$ generally.

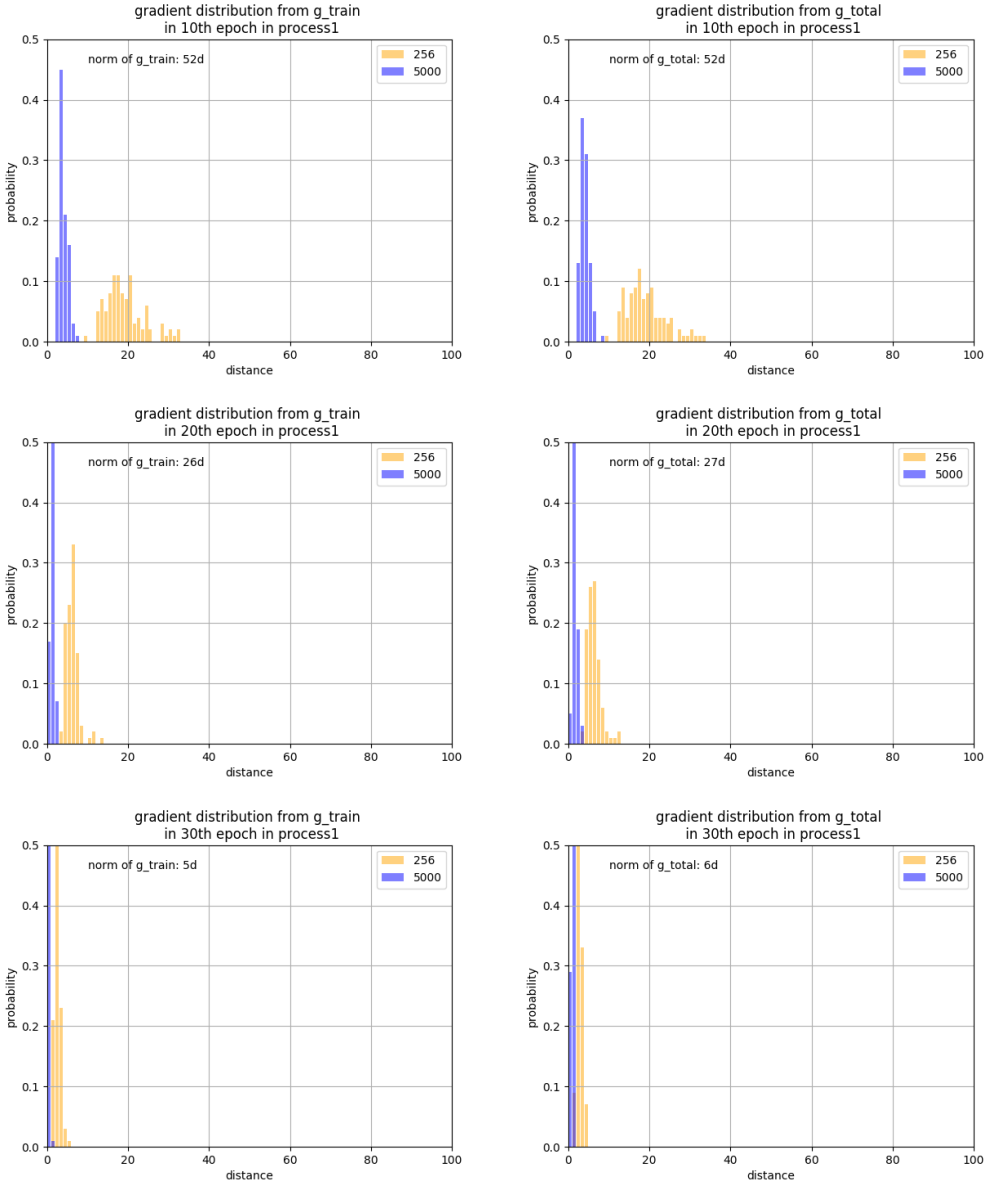

Table 3.2: Result of Experiment(B)

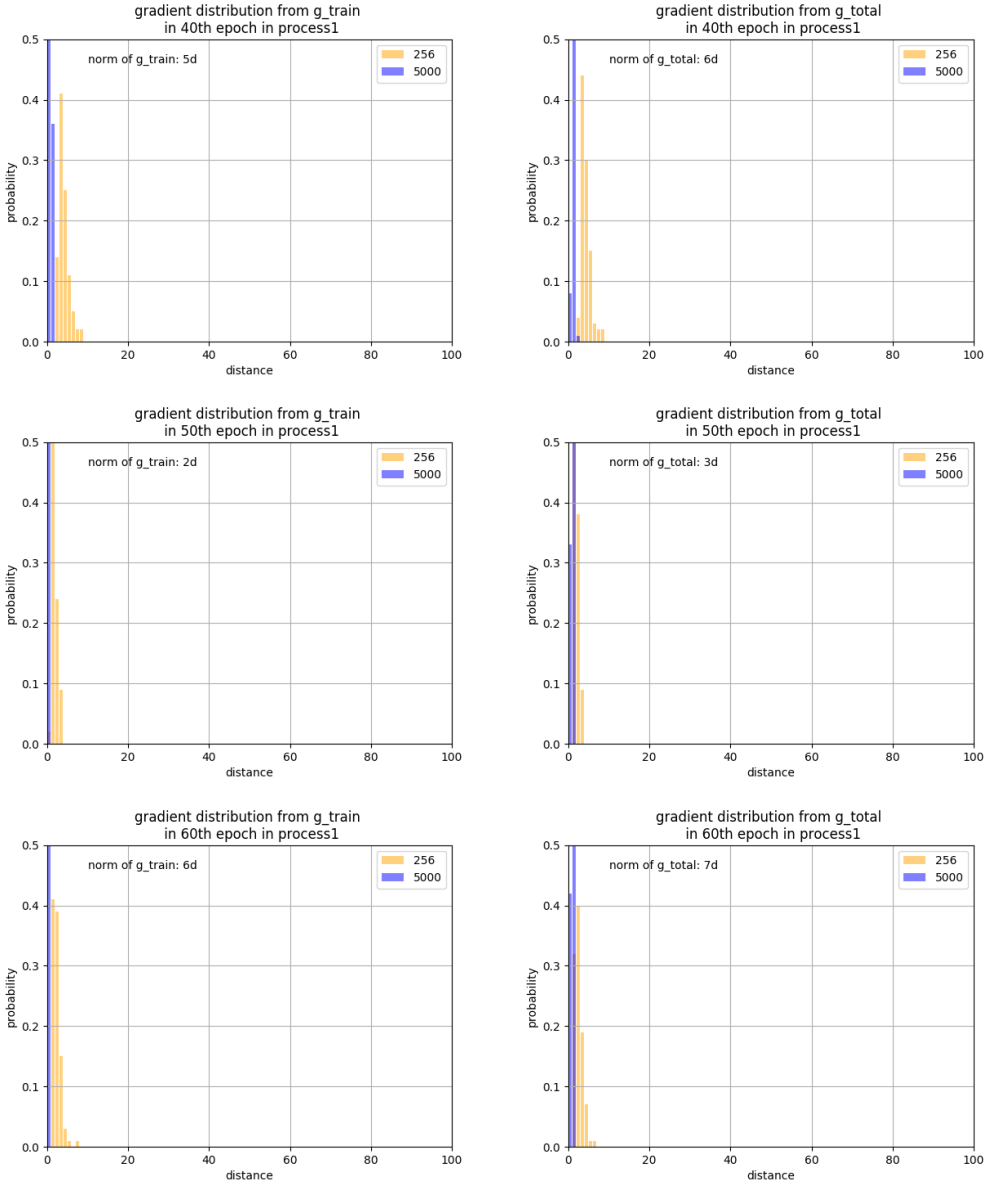

Table 3.3: Result of Experiment(B)

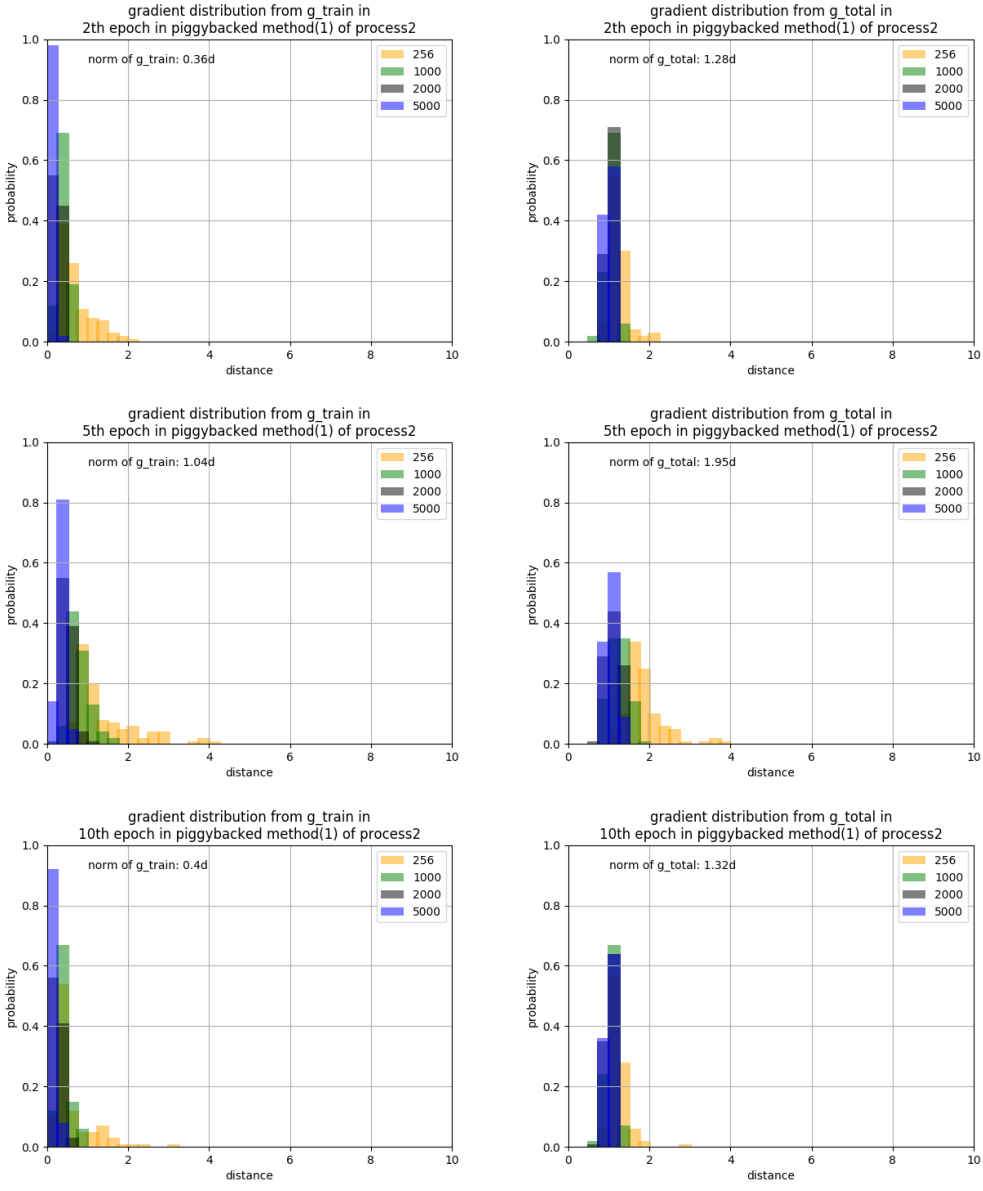

Table 3.4: Result of Experiment(B)

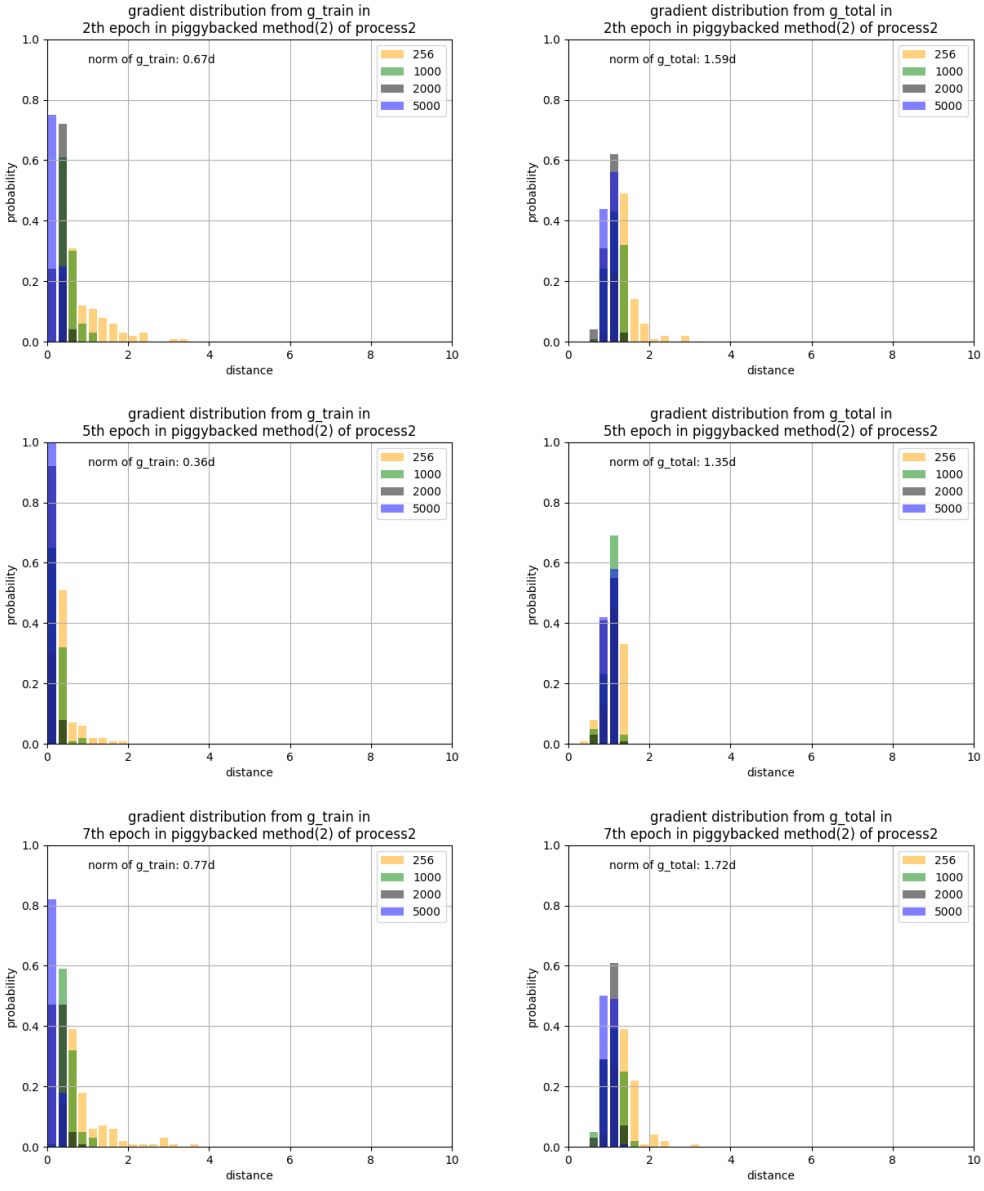

Table 3.5: Result of Experiment(B)

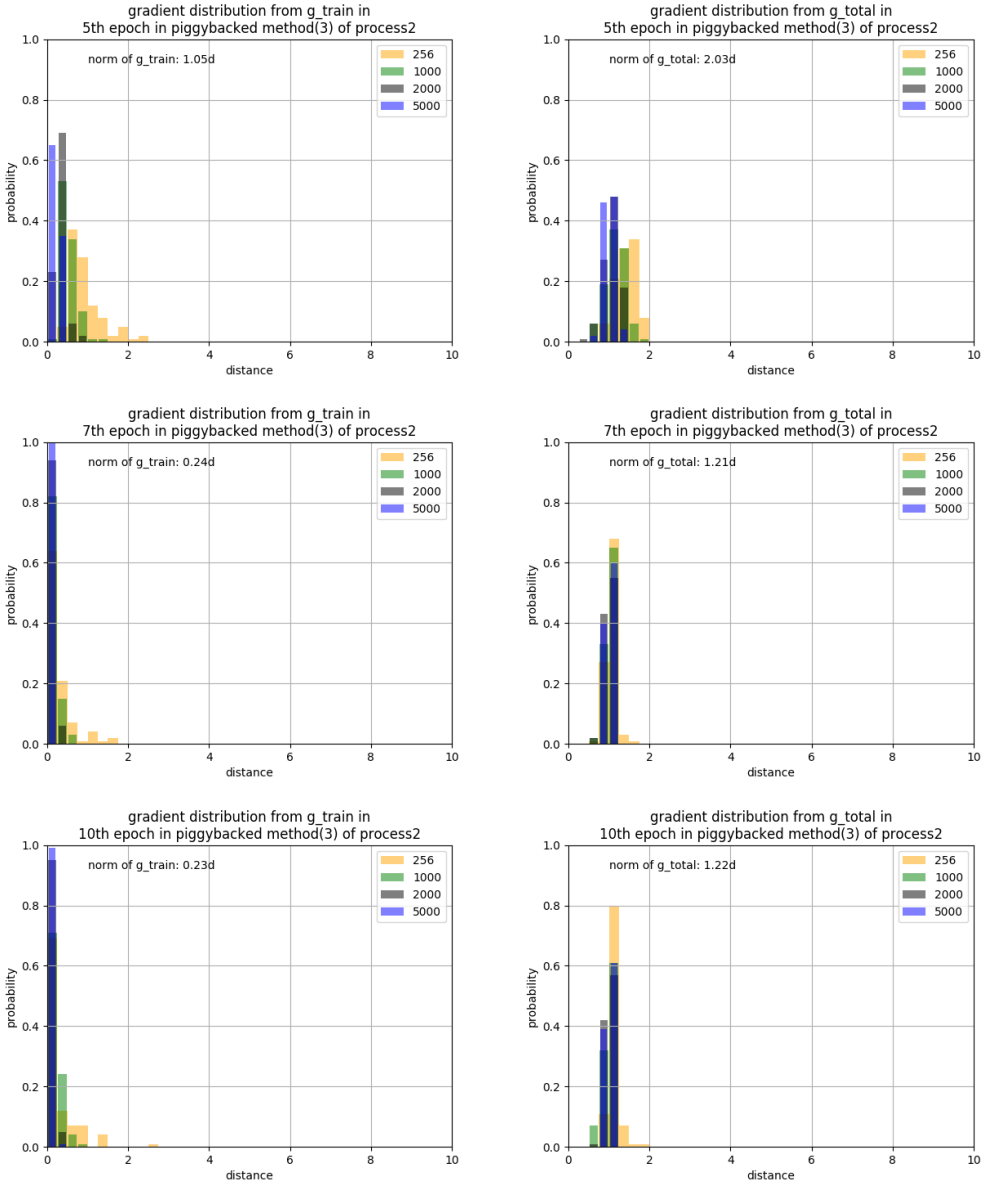

Table 3.6: Result of Experiment(B)

Chapter 4

Conclusion

In this thesis, we focused on the 'piggybacked method' which is simply mentioned in the last part in Keskar's paper. The 'piggybacked method' is large batch training after some epochs of small batch training and Keskar guessed that the new method would show better performance than original small batch training. We also thought that high performance of the new method is reasonable since we got an idea that controlling batch size is sort of like controlling power of smoothing loss function from Smith's papers and Wen's paper.

We intuitively thought that smoothing loss function helps a parameter to find flat minima well but rather hinders finding good minimizer once the parameter entered a flat minimum. Thus we guessed that training process should be divided into 2 processes. One should be the process for finding flat minima and the other should be the subsequent process for finding good minimizer in the flat minimum that the training parameter found. Since practitioners including us usually use stochastic gradient descent or its variants for training deep networks, we thought that taking small batch for smoothing strongly in the first process and taking large batch for getting good gradients in the second process would be better strategy (show higher performance) than original methods, and it was in accordance with the 'piggybacked method' Keskar claimed. In fact, our experiments showed that the 'piggybacked method' performed better than the conventional method of keeping the batch size small. Smith, who analyzed the gradients of the loss function statistically, suggested

a gradual reduction in batch size for decaying the learning time in his next paper, but it was generally lacked in accuracy on test set.

We conducted two kinds of experiments that will support our idea about high performance of 'piggybacked method' and got meaningful results. First experiment is visualizing the loss function along the line passing two parameters. One is the parameter that original small batch method converged to, and the other is what 'piggybacked method' found. It is widely known that large batch method is attracted to sharp minima while small batch method is attracted to flat one. However large batch training after small batch training didn't show that. The last parameter which large batch training in 'piggybacked method' found was not in that sharp minimum, even the minimum had similar curvature with the minimum which small batch method found, and both parameters seemed like that they were in the same flat minimum. However, the difference was that the parameter which 'piggybacked method' found always showed higher (or sometimes same depending on the networks) performance than the parameter which original one found.

Next, we analyzed the distribution of gradients from g_{train} and g_{total} during 'piggybacked training' in the second experiment. During the first process of 'piggybacked method', we got the results that g_{train} and g_{total} are very close compared to their norm, as expected. Thus gradients which are used for training seemed to need enough noise to explore the loss surface enough for finding flat minima, but gradients extracted from large batch do not have enough noise while those from small batch do. During the second process, we got really meaningful results. Once entering a flat minimum, gradients do not seem to need large noise as in the first process since g_{train} is near the zero vector and the norm of g_{train} and g_{total} are not that big. Opposed to the first process, gradients from small batch have so big noise that almost of them were very far from g_{total}, which helps the parameter to converge to better place. However, we found that many of gradients from large batch are near g_{total}. In addition, the gradients from large batch which are not close to g_{total} have far smaller norm than those from small batch so it can be said that gradients from large batch is absolutely less risky and advantageous.

We think these two kinds of results strongly support high performance of 'piggybacked method' and help clarifying the mechanism of learning deep

networks. Then finally, we expect the theoretical analysis which explain these results.

Bibliography

[1] L. Bottou, *Online learning and stochastic approximations*, On-line learning in neural networks, 17 (1998), p. 142.

[2] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, *The loss surfaces of multilayer networks*, in Artificial Intelligence and Statistics, 2015, pp. 192–204.

[3] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization*, in Advances in neural information processing systems, 2014, pp. 2933–2941.

[4] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio, *Sharp minima can generalize for deep nets*, arXiv preprint arXiv:1703.04933, (2017).

[5] R. Ge, F. Huang, C. Jin, and Y. Yuan, *Escaping from saddle points—online stochastic gradient for tensor decomposition*, in Conference on Learning Theory, 2015, pp. 797–842.

[6] M. Hardt, B. Recht, and Y. Singer, *Train faster, generalize better: Stability of stochastic gradient descent*, arXiv preprint arXiv:1509.01240, (2015).

[7] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, arXiv preprint arXiv:1502.03167, (2015).

[8] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, *On large-batch training for deep learning: Generalization gap and sharp minima*, arXiv preprint arXiv:1609.04836, (2016).

[9] A. Krizhevsky and G. Hinton, *Learning multiple layers of features from tiny images*, tech. rep., Citeseer, 2009.

[10] Y. LeCun, *The mnist database of handwritten digits*, http://yann. lecun. com/exdb/mnist/, (1998).

[11] J. D. Lee, M. Simchowitz, M. I. Jordan, and B. Recht, *Gradient descent converges to minimizers*, arXiv preprint arXiv:1602.04915, (2016).

[12] H. Li, Z. Xu, G. Taylor, and T. Goldstein, *Visualizing the loss landscape of neural nets*, arXiv preprint arXiv:1712.09913, (2017).

[13] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, *Don't decay the learning rate, increase the batch size*, arXiv preprint arXiv:1711.00489, (2017).

[14] S. L. Smith and Q. V. Le, *A bayesian perspective on generalization and stochastic gradient descent*, (2018).

[15] W. Wen, Y. Wang, F. Yan, C. Xu, Y. Chen, and H. Li, *Smoothout: Smoothing out sharp minima for generalization in large-batch deep learning*, arXiv preprint arXiv:1805.07898, (2018).

[16] P. Werbos, *Beyond regression: New tools for prediction and analysis in the behavioral sciences*, Ph. D. dissertation, Harvard University, (1974).

국문초록

깊은 신경망을 학습시킬 때, 배치 크기를 작게 하여 학습시키는 것이 손실 함수의 평평한 최소값으로 파라미터를 수렴시키고, 배치 크기가 큰 경우에는 그렇지 못함이 일반적으로 알려져있다. 또한 날카로운 최소값은 평평한 최소값에 비해 일반화가 좋지 못함이 알려져있기 때문에, 배치 크기와 일반화 정도 사이의 관계는 연구자들 사이에서 중요한 문제이다. 스미스는 배치 크기가 작을 수록 학습에 이용되는 그래디언트에 소음이 많이 섞이게 됨을 발견했고, 웬은 배치 크기의 영향을 줄이기 위해 날카로운 최소값을 피하는데에 적합한 새로운 학습 방법을 제안했다. 또한 케스카는 작은 배치 크기로만 학습시키는 기존의 방법에 비해, 배치 크기를 작게 했다가 일정 학습 이후에 크기를 키우는 '피기백 방법'이 일반화에 더 좋다고 추측하였다. 본 논문에서는, 케스카의 논문에서 언급된 '피기백 방법'에 초점을 맞추어 이 방법이 좋은 일반화 성능을 낼 수 있는 근거를 그래디언트 분석 실험을 통해 뒷받침하려 한다. 또한, 앞서 언급한 스미스와 웬의 논문을 요약하며 실험의 아이디어를 얻은 과정을 소개한다.

주요어휘: 손실함수, 그래디언트, 배치 크기, 딥러닝
학번: 2016-20226