HOKKAIDO UNIVERSITY

Title	Why Does Large Batch Training Result in Poor Generalization? A Comprehensive Explanation and a Better Strategy from the Viewpoint of Stochastic Optimization
Author(s)	Takase, Tomoumi; Oyama, Satoshi; Kurihara, Masahito
Citation	Neural Computation, 30(7), 2005-2023 https://doi.org/10.1162/neco_a_01089
Issue Date	2018-07
Doc URL	https://hdl.handle.net/2115/71558
Rights	© 2018 Massachusetts Institute of Technology
Type	journal article
File Information	Tomoumi Takase.pdf

Hokkaido University Collection of Scholarly and Academic Papers : HUSCAP

LETTER **Communicated by Kenji Kawaguchi**

Why Does Large Batch Training Result in Poor Generalization? A Comprehensive Explanation and a Better Strategy from the Viewpoint of Stochastic Optimization

Tomoumi Takase
takase_t@complex.ist.hokudai.ac.jp
Satoshi Oyama
oyama@ist.hokudai.ac.jp
Masahito Kurihara
kurihara@ist.hokudai.ac.jp
Graduate School of Information Science and Technology, Hokkaido University, Sapporo, Hokkaido 060-0814, Japan

We present a comprehensive framework of search methods, such as simulated annealing and batch training, for solving nonconvex optimization problems. These methods search a wider range by gradually decreasing the randomness added to the standard gradient descent method. The formulation that we define on the basis of this framework can be directly applied to neural network training. This produces an effective approach that gradually increases batch size during training. We also explain why large batch training degrades generalization performance, which previous studies have not clarified.

1 Introduction

Nonconvex optimization techniques are one of the most important components of machine learning because a better solution generally leads to a more accurate prediction for new data. Minimizing an objective function is difficult because it generally has a complex geometry with a large number of convex-concave structures.

Because few analytical solutions are available, numerical optimization methods are indispensable. A representative numerical optimization method for nonconvex problems is gradient descent,

$$x_{k+1} = x_k - \eta \cdot \mathrm{grad}\, f(x_k), \tag{1.1}$$

where x_k is the parameter at step k, $f(x)$ is the loss function, and η is the step size, a scaling value for adjusting the update amount. The loss is reduced by moving the parameter x in the direction opposite that of the gradient at each point. Convergence to the global minimum is not guaranteed, so

Neural Computation **30, 2005–2023** (2018) © 2018 Massachusetts Institute of Technology
doi:10.1162/neco_a_01089

the variables are often trapped in a poor local minimum or a saddle point, resulting in poor generalization performance.

Several methods for avoiding undesired solutions have been proposed, including AdaGrad (Duchi, Hazen, & Singer, 2011), RMSProp (Tieleman & Hinton, 2012), AdaDelta (Zeiler, 2012), and Adam (Kingma & Ba, 2015). These methods are based on the gradient descent and are often used in neural network training. They adjust the update amount and reduce the step size during training, so the parameter can converge more stably, although a wide search for a solution is performed at the beginning.

A simple way to avoid poor local minima is to start searching from different initial values and then choose the solution with the lowest loss. The initial values are important because different initial values lead to different solutions. This approach works, but the repeated trials are time-consuming. Several researchers have focused on the importance of initialization for optimization problems (Sutskever, Martens, Dahl, & Hinton, 2013; Mishkin & Matas, 2015).

Another approach is to use randomization in which the algorithm chooses a solution nondeterministically using randomness. Simulated annealing (Kirkpatrick, Gelatt, & Vecchi, 1983; Černý, 1985; Rere, Fanany, & Arymurthy, 2015) in particular is a powerful algorithm for global optimization problems. It moves a parameter in the direction in which the function value increases with the probability defined by the temperature parameter, which decreases with time. The details of this algorithm are described in section 2.2.

A generalization for the idea of simulated annealing can lead to the development of other effective approaches for nonconvex optimization problems. Therefore, in this letter, we present a comprehensive framework based on simulated annealing. It is that the randomness is strategically added: the randomness is strong at the beginning of the search and gradually weakens during the search, that is, the search strategy changes from unstable to stable. We apply the formulation based on this framework to minibatch training in neural networks.

The rest of this letter is organized as follows. In section 2, we describe the preliminaries and our motivation. In section 3, we describe our comprehensive framework of effective search approaches for nonconvex optimization problems. In section 4, we explain why the training with a large batch size in neural networks degrades generalization performance (Keskar, Mudigere, Nocedal, Smelyanskiy, & Tang, 2017) and apply our framework to minibatch training. Our experimental results support the validity of the framework. In section 5, we give a short summary.

2 Preliminaries and Motivation

2.1 Nonconvex Optimization Problems. Nonconvex optimization problems are a major issue in machine learning because the loss function

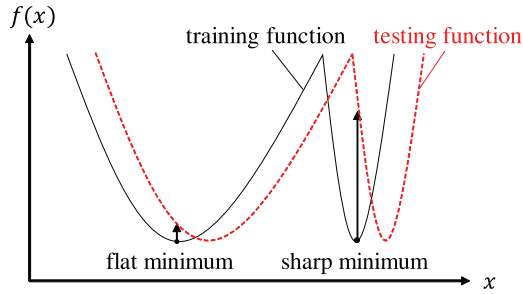


Figure 1: Conceptual sketch of flat and sharp minima (based on Keskar et al., 2017). The y-axis represents value of loss function, and the x-axis represents variables.

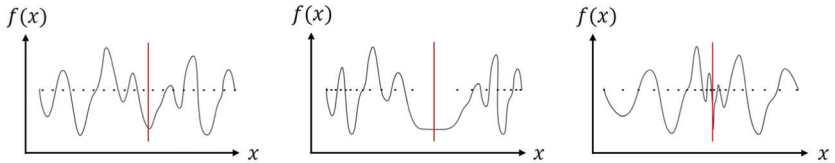
often has a complex shape in a large-dimensional space. Many search algorithms, such as hill climbing, simulated annealing, tabu search (Glover, 1986), ant colony optimization (Bonabeau, Dorigo, & Theraulaz, 1999), and genetic algorithm have been proposed for finding better solutions to problems with complicated loss functions.

A good understanding of the shape of the loss function is required for avoiding poor local minima. Thanks to many theoretical or empirical studies, several facts about the shape of the loss function have been revealed.

As an example of a theoretical achievement, Kawaguchi (2016) proved on the basis of some assumptions that every local minimum is a global minimum and every critical point that is not a global minimum is a saddle point. However, he concluded that because the assumptions were strong, poor local minima would arise in practice with deep nonlinear neural networks. Lu and Kawaguchi (2017) proved that without nonlinearity, even though depth of neural networks creates a nonconvex loss surface, it does not create poor local minima. However, they did not discuss the nonconvexity caused by the nonlinearity of activation functions. Thus, search methods for avoiding poor local minima are still required.

As an example of an especially remarkable empirical achievement, the existence of sharp minima was hypothesized by Keskar et al. (2017). Rather than providing theoretical support, they supported their hypothesis with many experimental results. Because their study motivated our research, we present a detailed explanation.

As shown in Figure 1, a sharp minimum has a high sensitivity, whereas a flat minimum has low sensitivity. In particular, when sharp and flat minima are defined for the training function, the loss for the testing function at the sharp minimum tends to be greater than that at the flat minimum. Therefore, the parameters trapped in sharp minima tend to result in degraded test accuracy. It is difficult to prove the existence of sharp minima because the loss function is generally complex, so Keskar et al. (2017) experimentally tried to demonstrate it. They claimed that the parameters



(a) Default parametrization (b) Reparametrization (c) Another reparametrization

Figure 2: A one-dimensional example of how much geometry of loss function depends on parameter space chosen (based on Dinh et al., 2017). The x-axis represents parameter value, and the y-axis represents loss.

are trapped in sharp minima when a large batch is used for training, as described in detail in section 4.

However, Dinh, Pascanu, Bengio, and Bengio (2017) considered such a notion of sharp minima ill defined. They insisted that Keskar et al.'s (2017) definitions of sharpness and flatness were not appropriate because, if we follow their definition of sharpness, the loss function can be freely deformed, meaning that all minima can be made flat by reparameterization, as shown in Figure 2. They concluded that the definition of sharpness and flatness should be reconsidered.

2.2 Simulated Annealing. Simulated annealing is an effective randomized algorithm for finding a global minimum. A related method, quantum annealing (Kadowaki & Nishimori, 1998), is mainly used to solve combinational optimization problems. The D-wave machine (Johnson et al., 2011), the first commercial computer incorporating quantum annealing, has been attracting attention.

In the gradient descent, the search always proceeds in the direction in which the function value decreases, while in the simulated annealing method (see algorithm 1), the search stochastically proceeds in the direction in which the function value increases, with the probability controlled by the temperature parameter. In algorithm 1, x_t denotes the parameter at step t, $\mathcal{N}(x_t)$ denotes a set of solutions in the neighborhood of x_t, and Temp_{t+1} denotes the temperature at step $t+1$. The temperature gradually decreases during search, so the probability also decreases. If the range of temperature reduction is infinitesimal, the parameter always converges to a globally optimal solution (Lundy & Mees, 1986). In actual use, the change of the temperature should be appropriately scheduled.

3 Comprehensive Framework

Training with a large batch size in a neural network tends to result in poor generalization performance (Keskar et al., 2017). In order to avoid this

Algorithm 1: Simulated Annealing.

Require: number of updates: S; initial parameter: x_0

for t from 0 to $S-1$ **do**

Choose $x' \leftarrow$ a random element from $\mathcal{N}(x_t)$

if $f(x') \leq f(x_t)$ **then**

$x_{t+1} \leftarrow x'$

else

$x_{t+1} \leftarrow x'$ with probability $e^{-[f(x')-f(x_t)]/\mathrm{Temp}_{t+1}}$

otherwise $x_{t+1} \leftarrow x_t$

end if

end for

return x_S

problem, we present in this section a comprehensive framework of effective approaches for solving nonconvex optimization problems.

3.1 Update with Noise. It is difficult to escape from poor local minima with the standard gradient descent, equation 1.1, because the update needs to proceed in the upward direction (i.e., the direction in which the function value increases). Therefore, an effective way to avoid poor local minima is to adjust grad $f(x_k)$ in equation 2.1 so that the update sometimes proceeds in a random direction. Thus, a noise ξ_t depending on search step t is often added to grad $f(x_k)$, meaning that the update does not always proceed in the downward direction because of the randomness created by the added noise. This random movement greatly contributes to avoiding poor local minima.

However, simple addition of such a noise does not necessarily work well because an update in such a direction slows convergence or makes it difficult. Thus, there is a trade-off between poor local minima avoidance and stable convergence.

The degree of randomness of the added noise should thus be controlled according to the stages of the search process, as follows. At the beginning of search, it is more important to search for a better solution than it is to reach convergence. Therefore, a large ξ_t should be added to grad $f(x_k)$. Then as the search proceeds, convergence to a target solution becomes more important,

Update number:	t	$\longrightarrow$	∞
Key parameter:	ϕ_t	$\longrightarrow$	α
Noise:	$\xi(\phi_t)$	$\longrightarrow$	0
Update direction:	$\mathrm{grad}(f(x_k) + \xi(\phi_t))$	$\longrightarrow$	$\mathrm{grad}f(x_k)$

Figure 3: Change in each variable during search.

so ξ_t should be decreased to weaken the effect of the randomness. When ξ_t approaches 0, the update method is almost the same as the standard gradient descent.

3.2 Noise Decreasing Strategy. How to produce the noise is different for each search method, but the noise should be controlled by a key parameter ϕ_t at search step t—that is, the noise is given by $\xi(\phi_t)$. In simulated annealing, ϕ_t corresponds to the conditional probability p,

$$p = \exp\left(-\frac{f(x') - f(x_t)}{\mathrm{Temp}_{t+1}}\right), \tag{3.1}$$

that when x' is a candidate state for the next destination of x_t such that $f(x') > f(x_t)$, x' is actually adopted as the next state.

The change in each variable during search is shown in Figure 3. Note that as the search proceeds (with t approaching ∞), ϕ_t approaches a constant (α), the noise $\xi(\phi_t)$ approaches 0, and the search strategy approaches the standard gradient descent in equation 1.1. In simulated annealing, α corresponds to 0, meaning that the update never proceeds in the upward direction. Because temperature Temp_{t+1} in equation 3.1 gradually decreases, p approaches 0, which means that simulated annealing follows this framework.

4 Application to Neural Network Learning

As an application of our framework, we focus on the problem of minimizing the training loss in neural networks. The loss function for a neural network, especially a deep neural network, is generally very complex and therefore difficult to minimize.

Keskar et al. (2017) claimed that training with a large batch can degrade generalization performance because it tends to be attracted to regions with sharp minima. Dinh et al. (2017) argue that there are critical issues in the concept of flatness (which is not well defined), but the experimental finding of Keskar et al. (2017) that a large batch size degraded generalization performance is remarkable. Although the cause of the degradation might

Algorithm 2: Minibatch Stochastic Gradient Descent.

Require: number of updates: S; batch size: N; number of training samples: M

$i \leftarrow 0$

for t from 0 to $S-1$ **do**

if $i+N \geq M$ **then**

$\theta \leftarrow \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i;M)}; y^{(i;M)})$

$i \leftarrow 0$

else

$\theta \leftarrow \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i;i+N)}; y^{(i;i+N)})$

$i \leftarrow i+N$

end if

end for

be attraction to a sharp minimum, the reasons a large batch size led to such a result were not discussed by Keskar et al. (2017). We speculate that this phenomenon is explained by the stability of the loss function and present a new approach to avoiding poor solutions, following the framework described in the previous section.

4.1 Minibatch Stochastic Gradient Descent. Minibatch stochastic gradient descent (MSGD) is widely used in neural network training. The algorithm is shown in algorithm 2, where the sample number ranges from 0 to $M-1$, and $(i; M)$ and $(i; i+N)$, intervals of the sample number for minibatch, denote $[i, M)$ and $[i, i+N)$, respectively. Here, the samples for each batch are chosen so that all samples are used without replacement in one epoch.

Before discussing the stability of a neural network, we give a convenient replacement for the loss function in MSGD. The gradient of the loss function, that is, the update amount, in MSGD is given using the update amount for each sample by

$$\frac{\partial E_t}{\partial \theta} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial E^{(n)}}{\partial \theta}, \tag{4.1}$$

where θ denotes the set of parameters, E_t denotes the loss function for the tth minibatch with batch size N, and $E^{(n)}$ denotes the loss function for the nth sample in the minibatch. Therefore, $\partial E_t/\partial\theta$ is the mean of the update amounts for the tth minibatch, which is used for the gradient descent as $\mathrm{grad} f(x_k)$ in equation 1.1. In MSGD, only one update for each minibatch is performed by calculating $\partial E_t/\partial\theta$. The right-hand side of equation 4.1 can be deformed by

$$\frac{1}{N}\sum_{n=1}^{N}\frac{\partial E^{(n)}}{\partial\theta} = \frac{\partial}{\partial\theta}\left(\frac{1}{N}\sum_{n=1}^{N}E^{(n)}(\theta)\right). \tag{4.2}$$

This means that the mean of the gradient of each loss function is the same as the gradient of the mean of the loss functions. Therefore, the mean of the loss functions can be used for the update in minibatch training instead of the mean of the update amounts, as given by

$$E_t(\theta) = \frac{1}{N}\sum_{n=1}^{N}E^{(n)}(\theta). \tag{4.3}$$

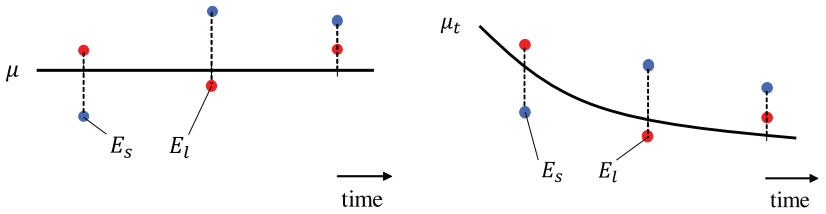
This is convenient when the behavior of the loss function in minibatch training is discussed.

4.2 Stability of Loss Function. We speculate that a large batch size stabilizes the loss function; that is, the shape of the function is almost the same for each update. If the loss function is stable, escape from a poor local minimum is difficult because the landscape around there does not change so much. If the loss function is unstable, escape from a poor local minimum is easier because the shape changes for each update.

Let us consider why the loss function is stable when using a large batch size. Here, we evaluate the stability of the value of the loss function instead of the stability of the shape of the loss function because the shape of the loss function is too complex to analyze due to the multilayer and nonlinear activation function. An unstable loss function changes the loss value significantly, so this value can be used to investigate the stability of the loss function.

As discussed in section 4.1, the mean of the gradients of loss functions is the same as the gradient of the mean of the loss functions, that is, the loss $E_N(\theta)$ with batch size N is given by $(1/N)\sum_{n=1}^{N}E^{(n)}(\theta)$. Therefore, the loss $E_s(\theta)$ with a small batch size N_s and the loss $E_l(\theta)$ with a large batch size N_l are given by

$$E_s(\theta) = \frac{1}{N_s}\sum_{n=1}^{N_s}E^{(n)}(\theta), \tag{4.4}$$



(a) True loss is fixed.

(b) True loss decreases. (actual neural network training)

Figure 4: Conceptual sketch of variance of losses in each batch.

$$E_l(\theta) = \frac{1}{N_l} \sum_{n=1}^{N_l} E^{(n)}(\theta). \tag{4.5}$$

By denoting the variance of loss E for each batch by $\mathrm{var}(E)$, we can prove $\mathrm{var}(E_s(\theta)) > \mathrm{var}(E_l(\theta))$.

Considering the special case where all the samples constitute a single batch, we get the mean of the losses for all M samples as $(1/M)\sum_{n=1}^{M} E^{(n)}(\theta)$. Here, we call this the true loss, although the true loss is usually considered to be the mean of the loss for the original distribution from which training data were sampled. Each batch is independently sampled from training data. As shown by the bold line in Figure 4a, which is a conceptual sketch of the variance of losses in each batch, if the true loss μ for all training data is fixed, the mean of the loss for each batch follows the central limit theorem. That is, the mean of the loss for each batch follows a normal distribution with mean μ and variance σ^2/N. Because $N_s < N_l$, $\sigma^2/N_s > \sigma^2/N_l$, and therefore $\mathrm{var}(E_s(\theta)) > \mathrm{var}(E_l(\theta))$ holds.

This relationship is not so simple in neural networks because the loss decreases during training by the gradient descent, as shown by the bold line in Figure 4b, which represents true loss with all training data. The true loss tends to decrease as t increases. In this case, E_t does not follow a normal distribution since μ_t changes.

However, a decrease in the loss during training is necessary for neural network training because decreasing the loss is the purpose of training. Therefore, the variance of the difference between μ_t and the mean loss for each batch (see Figure 4b) is important, rather than the difference between μ and the mean loss for each batch (see Figure 4a). The variance of the difference follows a normal distribution with mean μ_t due to the central limit theorem, as is the case with the fixed loss in Figure 4a. Therefore, the relationship $\mathrm{var}(E_s(\theta) - \mu_t) > \mathrm{var}(E_l(\theta) - \mu_t)$ is obtained. When this inequality holds, we say in this letter that the loss function for the right-hand side is more stable than that for the left-hand side.

In general, when the data are underfitting on a fixed loss function, escape from a local minimum and then seeking a smaller loss can provide a better generalization performance than being stuck at a poor local minimum. On the other hand, in the overfitting case, such escape and seeking can cause worse generalization performance. Thus, being stuck at a local minimum in the overfitting case might result in better generalization, as it avoids further overfitting. Therefore, there is a good reason to assume that a suitable strategy is to find a better local minimum by searching a wide region during the underfitting stage and then try to stably converge to the minimum during the overfitting stage.

4.3 Variable Batch Size. Based on the discussion of stability in the previous section, we assume that a smaller batch is more appropriate for finding a better solution and a larger batch more appropriate for obtaining stable convergence. Our approach to minibatch training effectively uses a strategy based on this assumption. More precisely, we apply the framework discussed in section 3 to minibatch training as follows. We associate the noise-controlling key parameter ϕ_t in Figure 3 with the batch size at update t. When all training data are used for each batch, the loss function is unique, and we regard its gradient as the true gradient. When samples are chosen from training data, the gradients calculated for each batch are different, and we regard each of such differences from the true gradient as the noise $\xi(\phi_t)$ added to the true gradient. Since the change in the gradient results in the change in the resultant loss, we incorporate this idea into our framework by setting the noise $\xi(\phi_t) = |E_t(\theta) - E_M(\theta)|$, that is, the difference between the training loss for the minibatch and the true loss.

A decrease in noise is achieved by increasing the batch size, thus changing the loss function from unstable to stable. This corresponds to the change in the purpose of the search during the search: from finding a better solution to obtaining stable convergence. This strategy is very simple and easy to implement: the batch size is increased linearly during training with the initial and final sizes decided beforehand. By gradually increasing the batch size, we can expect that when the batch size is still smaller, the search procedure can easily escape from a sharp minimum in such a nonoverfitting stage with bigger noise, and when the batch size gets bigger, the search gradually and stably tends to converge to a flat minimum even in such an overfitting stage with smaller noise and thus finally succeed in better generalization rather than being stuck at a sharp minimum, which might have a smaller loss but less generalization performance.

This method is similar to stochastic average gradient (SAG; Roux, Schmidt, & Bach, 2012) in the sense that both methods focus on the difference from the true gradient. Like SGD, SAG only computes the gradient for a randomly chosen training example for each iteration, but unlike SGD, it preserves the gradients for all examples and calculates a new gradient by averaging them, except that with respect to only the chosen example, the

Table 1: Model Structures.

Layer Type	Channels/Units
5×5 conv, BN, ReLU	32
2×2 max pool, str. 2	32
5×5 conv, BN, ReLU	64
2×2 max pool, str. 2	64
BN, ReLU	64
Dropout with $p = 0.5$	64
Fully connected, BN, ReLU	1024
Dropout with $p = 0.5$	1024
Output softmax	10

Note: BN denotes batch normalization.

preserved gradient is replaced by the new gradient at the current position in the parameter space. Thus, the gradient calculated by this method yields an estimate for the true gradient efficiently. However, unlike our batch increase method, SAG does not take advantage of a large difference from the true gradient produced by SGD because their interests lie only in the optimization of strongly convex objective functions (with no local minima).

Stochastic variance reduction gradient (SVRG; Johnson & Zhang, 2013) is also the improvement of SGD. Reducing the variance of the difference between the noisy gradient and the true gradient, this method enjoys the same fast convergence as SAG without requiring the storage of gradients. However, like SAG, it does not take advantage of a large difference from the true gradient.

A gradual increase in the batch size is known to be effective for training. For example, in their recent study, Smith, Kindermans, and Le (2017) have demonstrated that an increase in the batch size without decay of the learning rate during training can be useful for accelerating the training. In this work, we have focused on the fact that this idea may also be useful for escaping from sharp minima and converging to a flat minimum.

4.4 Main Experiments. We investigated the effects of the variable batch size by an experiment as follows. We trained a convolutional neural network (CNN) by using the MNIST data set (LeCun, Bottou, Bengio, & Haffner, 1998). The MNIST data set consists of handwritten digital images, including 60,000 original and 10,000 test samples. The image size is 28×28 pixels, and the input values are the brightness, from 0 to 255. Images are divided into 10 classes, from 0 to 9.

We used a simple and relatively small CNN (outlined in Table 1). A rectified linear unit (ReLU) (Glorot, Bordes, & Bengio, 2011) was used as the activation function of the hidden layer, and the softmax function was used as the activation function of the output layer. Batch normalization (Ioffe &

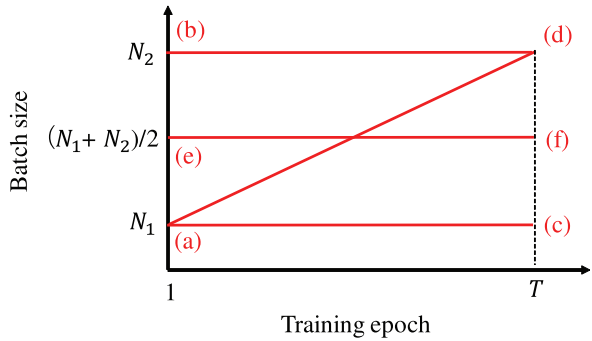


Figure 5: Batch size changing strategies. Four strategies were tested: (a)→(d), (a)→(c), (b)→(d), and (e)→(f).

Szegedy, 2015), which changes the mean to 0 and the variance to 1, was applied to the output of each layer, except the final output. Dropout (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014) was used in the fully connected CNN layer to avoid overfitting, where the probability for selecting units was always set to 0.5. A Glorot uniform distribution (LeCun, Bottou, Orr, & Müller, 1998; Glorot & Bengio, 2010) was used to initialize the weights in the neural network. The update method was AdaGrad with an initial learning rate of 0.01.

The number of updates required for convergence depends on the batch size because an unstable loss function due to a small size slows convergence. Therefore, to compare test losses for different batch sizes, we continued the training until it was regarded as almost converged instead of ending it in the fixed number of updates. Actually, 3000 epochs were sufficient for the convergence. The test loss was calculated for each epoch, and the training data were shuffled every epoch to remove the order effect.

We investigated the effect of changing the batch size by comparing the test losses with the four strategies to change the batch size, as shown in Figure 5: linear increase ((a)→(d)), fixed to N_1((a)→(c)), fixed to N_2((b)→(d)), and fixed to the mean of N_1 and N_2((e)→(f)).

We set N_1 to 10, considering that a batch size smaller than 10 is rarely used in practice for efficiency reasons. As for N_2, we set two different conditions: $N_2 = 100$ as the first condition and $N_2 = 500$ as the second condition. From these settings, the batch sizes for the fixed strategy were 10, 55, and 100 for the first condition and 10, 255, and 500 for the second condition. The algorithm for the linear increase strategy is shown in algorithm 3, where the sample number ranges from 0 to $M - 1$, and $(i; i + N)$, an interval of the sample number for each mini-batch, denotes $[i, i + N)$.

The transitions of test loss, the average of the cross-entropy for all samples, plotted in Figure 6 show that the linear increase strategy resulted in the

Algorithm 3: Linear Increase Strategy.

Require: number of epochs required for convergence: $T(>1)$; initial batch size: N_1;

batch size at final epoch: $N_2(>N_1)$; number of training samples: M

for t from 0 to $T-1$ **do**

$i \leftarrow 0$

$N \leftarrow \text{round}(\frac{N_2-N_1}{T-1}t + N_1)_{int}$

while $i < M$ **do**

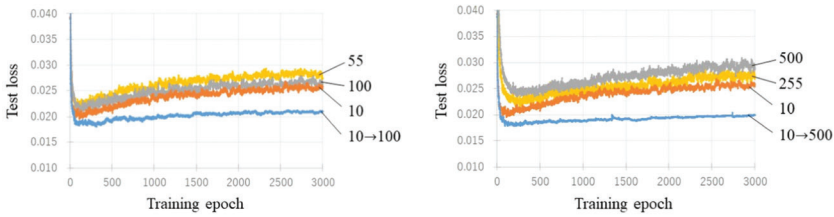
$i \leftarrow \min(i, M-N)$

$\theta \leftarrow \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i;i+N)}; y^{(i;i+N)})$

$i \leftarrow i+N$

end while

end for



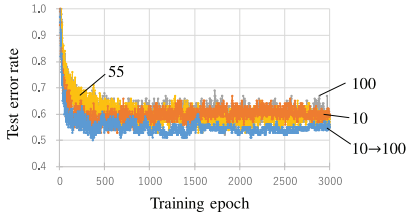
(a) First condition: $N_1 = 10$, $N_2 = 100$.

(b) Second condition: $N_1 = 10$, $N_2 = 500$.

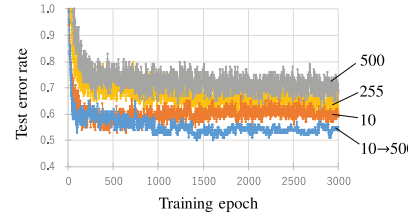
Figure 6: Effect of changing batch size on test loss.

lowest test loss and the most stable transitions. The transitions of test error rate plotted in Figure 7 also show that the linear increase strategy resulted in the lowest test error rate. Overall, the linear increase strategy demonstrated the best results.

From the transitions of the test loss in Figure 6 and the training loss plotted in Figure 8, we found that the fixed batch size strategy showed a strong tendency for overfitting, whereas the linear increase strategy did not show such a tendency. We can speculate that the parameter in the linear increase strategy converged to a flat minimum because the test loss was smaller than that for the fixed large batch size while the training loss was not. Therefore,

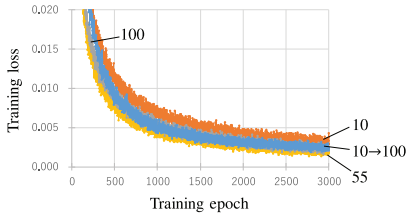


(a) First condition: $N_1 = 10$, $N_2 = 100$.

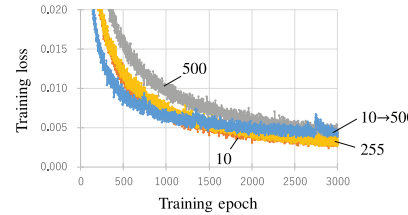


(b) Second condition: $N_1 = 10$, $N_2 = 500$.

Figure 7: Effect of changing batch size on test error rate.



(a) First condition: $N_1 = 10$, $N_2 = 100$.



(b) Second condition: $N_1 = 10$, $N_2 = 500$.

Figure 8: Effect of changing batch size on training loss.

we may say that gradual increase in the batch size is effective for a stable convergence to a flat minimum.

4.5 Investigation for Search Range. In the previous sections, we speculated that the smaller the batch size was, the wider the search range would become, and that the batch size increase method would search a wider range in the beginning and make it narrower gradually. In this section, we investigate through experiments such effects of the batch size on the search range.

Let D be the number of weight parameters of a neural network. We want to transform the D-dimensional parameter space into a mesh with many cells and then count the number of those cells, N_{mesh}, the search procedure visits during training. If one cell is visited twice or more, we count it as one visit. However, if all dimensions were divided into cells, N_{mesh} would become too large because D is 1,111,946 in our neural network model. In that case, the procedure would rarely visit the same cells during training, making it difficult to compare various values of N_{mesh} according to different batch sizes. To avoid this, we randomly extract D' dimensions from D dimensions and divide each of them into C sections with equal intervals, keeping the remaining $D - D'$ dimensions untouched. As a result, we have a mesh with as many as $C^{D'}$ cells. In the experiments, we set $D' = 1000$ and selected several values for C to investigate their effects on N_{mesh}.

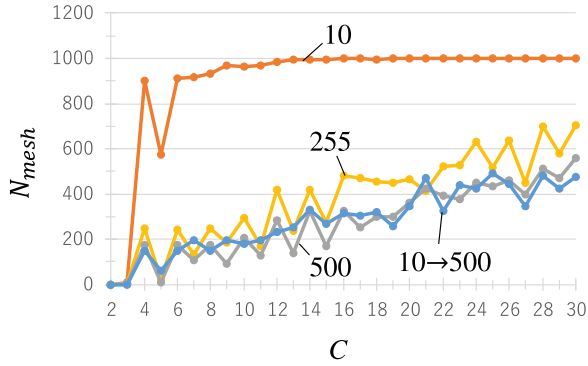


Figure 9: Comparison of the number of cells visited.

In the experiments, we fixed the domains of the parameters to $[-0.3, 0.3]$ for all dimensions because we had an observation that the values of all weights during training had fallen in $[-0.23, 0.19]$ for all batch sizes.

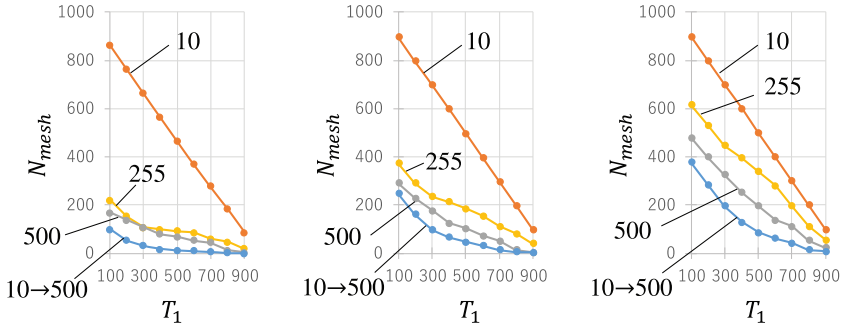
Using the same model and data as those in the previous section, we set N_1, N_2, and T in Figure 5 to 10, 500, and 1000, respectively. To compare the results according to different batch sizes, we plotted the values of each parameter at each epoch, resulting in 1000 plot data in all.

The experimental results given in Figure 9 show that N_{mesh} for the linear increase strategy was almost the same as that for the fixed batch size 500 and much less than the fixed batch size 10.

However, we would like to make a distinction between N_{mesh} values during the overfitting period and those before such a period. To investigate it, we divided the training period into the first and the latter parts with the boundary time T_1 for counting N_{mesh} before and after T_1, where we set $T_1 \in \{100x \mid 1 \leq x \leq 9, x \in \mathbb{N}\}$, considering that the model would reach convergence before epoch 1000. As for the structure of the mesh, we set $C \in \{10, 20, 30\}$.

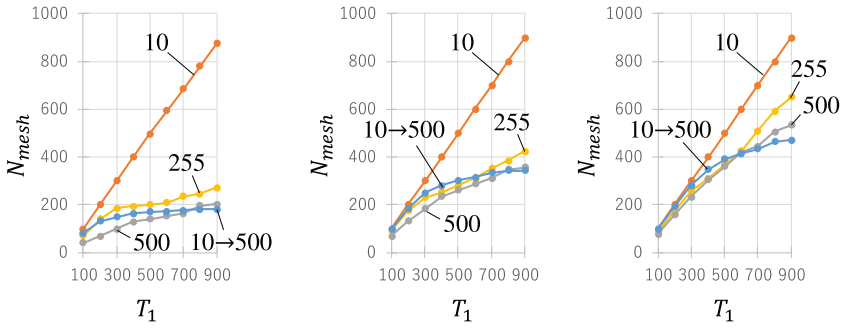
As seen in Figure 10, the linear increase strategy has made the smallest transition of N_{mesh} after T_1. Therefore, we can say that the batch size increase method resulted in the most stable convergence after the model began to overfit. Moreover, as seen in Figure 11, N_{mesh} in the linear increase strategy was larger than that for the fixed batch size 500 before the epoch reached $T_1 = 500$.

Searching a wide range in the nonoverfitting stage is important for effective training, especially in the region with a small training loss, because local minima generally have a very small loss compared to the loss at an initial state, and escaping from such local minima in the nonoverfitting stage can lead to a good generalization. To compare N_{mesh} in a region with a small loss, we divide the mesh into two regions depending on whether the loss



(a) $C = 10$. (b) $C = 20$. (c) $C = 30$.

Figure 10: Comparison of the number of cells visited in the latter part.

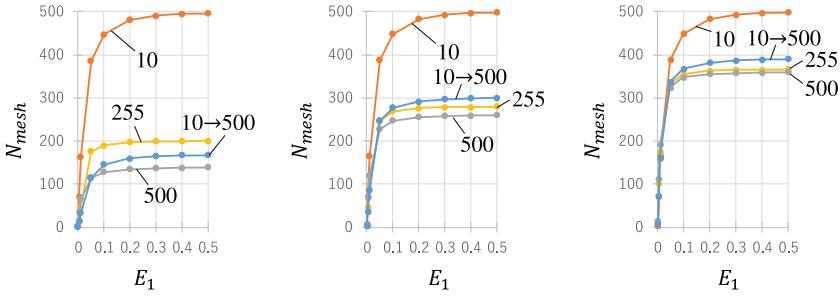


(a) $C = 10$. (b) $C = 20$. (c) $C = 30$.

Figure 11: Comparison of the number of cells visited in the first part.

E in a cell is smaller than the threshold $E_{\min} + E_1(E_{\max} - E_{\min})$, where $E_{\max}$ and $E_{\min}$ are the maximum loss and the minimum loss, respectively, in the mesh, and E_1 is a value between 0 and 1 to determine the threshold. Considering $T_1 = 500$ as a rough criterion for the beginning epoch of overfitting, we counted N_{mesh} in the region where $E < E_1$ before the epoch reached T_1.

As seen in Figure 12, N_{mesh} for the linear increase strategy was larger than those for the fixed batch sizes 255 and 500, when $C = 20, 30$ and $E > 0.1$. This means that the linear increase strategy can explore a wider region than the fixed batch size strategy with large batch sizes, when C is large enough (to make finer mesh) and E is also large enough (to evaluate the shallow region visited before the overfitting period).



(a) $C = 10$. (b) $C = 20$. (c) $C = 30$.

Figure 12: Comparison of the number of cells visited in the region $E < E_1$ before epoch $T_1 = 500$.

5 Summary

We have presented a comprehensive framework of effective search methods, such as the simulated annealing method, for solving nonconvex optimization problems. The noise added to the direction of the iterated search steps differentiates the framework from the standard gradient descent method except that they are almost the same when the noise is very small. We have discussed why search methods based on this framework can avoid poor local minima.

Application of this framework to minibatch training for neural networks has demonstrated that a larger batch size tends to produce a more stable loss function, which explains why a larger batch size often degrades generalization. Using the framework and this finding, we have developed a linear increase strategy effective for solving nonconvex optimization problems with neural networks. We have empirically demonstrated that this strategy has resulted in a significant improvement in generalization performance.

Acknowledgments

This research was supported by Global Station for Big Data and CyberSecurity, a project of Global Institution for Collaborative Research and Education at Hokkaido University.

References

Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence: From natural to artificial systems*. New York: Oxford University Press.

Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, *45*, 41–51.

Dinh, L., Pascanu, R., Bengio, S., & Bengio, Y. (2017). *Sharp minima can generalize for deep nets*. arXiv:1703.04933.

Duchi, J., Hazen, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*, 2121–2159.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of Artificial Intelligence and Statistics Conference* (pp. 249–256).

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of Artificial Intelligence and Statistics Conference* (pp. 315–323).

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, *13*, 533–549.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning* (pp. 448–456).

Johnson, M. W., Amin, M. H. S., Gilbert, S., Lanting, T., Hamze, F., Dickson, N., & Rose, G. (2011). Quantum annealing with manufactured spins. *Nature*, *473*, 194–198.

Johnson, R., & Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., & K. Q. Weinberger. *Advances in neural information processing systems*, *26*. Red Hook, NY: Curran.

Kadowaki, T., & Nishimori, H. (1998). Quantum annealing in the transverse Ising model. *Physical Review E*, *58*, 5355–5363.

Kawaguchi, K. (2016). Deep learning without poor local minima. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems*, *29*. Red Hook, NY: Curran.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2017). On large-batch training for deep learning: Generalization gap and sharp minima. In *Proceedings of the International Conference on Learning Representations*. OpenReview.net.

Kingma, D., & Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*. OpenReview.net.

Kirkpatrick, S., Gelatt Jr., C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). *The MNIST database of handwritten digits*. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

LeCun, Y., Bottou, L., Orr, G. B., & Müller, K.-R. (1998). Efficient backProp. In G. B. Orr & K.-R. Müller (Eds.), *Lecture Notes in Computer Science: Vol. 1524: Neural Networks: Tricks of the Trade* (pp. 9–48). Berlin: Springer.

Lu, H., & Kawaguchi, K. (2017). *Depth creates no bad local minima*. arXiv:1702.08580.

Lundy, M., & Mees, A. (1986). Convergence of an annealing algorithm. *Programming*, *34*, 111–124.

Mishkin, D., & Matas, J. (2015). *All you need is a good init*. arXiv:1511.06422.

Rere, L. M. R., Fanany, M. I., & Arymurthy, A. M. (2015). Simulated annealing algorithm for deep learning. *Procedia Computer Science*, *72*, 137–144.

Roux, N. L., Schmidt, M., & Bach, F. R. (2012). A stochastic gradient method with an exponential convergence rate for finite training sets. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems*, *25*. Red Hook, NY: Curran.

Smith, S. L., Kindermans, P. J., & Le, Q. V. (2017). *Don't decay the learning rate, increase the batch size*. arXiv:1711.00489.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*, 1929–1958.

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning* (pp. 1139–1147).

Tieleman, T., & Hinton, G. E. (2012). Lecture 6.5: RMSProp. *COURSERA: Neural networks for machine learning*.

Zeiler, M. D. (2012). ADADELTA: *An adaptive learning rate method*. arXiv:1212.5701.

Received July 18, 2017; accepted January 28, 2018.