Optimizing Neural-network Learning Rate by Using a Genetic Algorithm with Per-epoch Mutations

Yasusi Kanada

Media Research Department, Center for Technology Innovation – System Engineering, Hitachi, Ltd.
1-280 Higashi-koigakubo, Kokubunji, Tokyo 185-8601, Japan
Yasusi.Kanada.yq@hitachi.com

***Abstract* – Recently, performance of deep neural networks, especially convolutional neural networks (CNNs), has been drastically increased by elaborate network architectures, by new learning methods, and by GPU-based high-performance computation. However, there are still several difficult problems concerning back propagation, which include scheduling of learning rate and controlling locality of search (i.e., avoidance of bad local minima). A learning method, called "learning-rate-optimizing genetic back-propagation" (LOG-BP), which combines back propagation with a genetic algorithm by a new manner, is proposed. This method solves the above-mentioned two problems by optimizing the learning process, especially learning rate, by genetic mutations and by locality-controlled parallel search. Initial experimental results shows that LOG-BP performs better; that is, when required, learning rate decreases exponentially and the distances between chromosomes, which indicate the locality of a search, also decrease exponentially.**

***Key words* – Back propagation, Learning rate, Genetic algorithm, Multi-layer perceptron, Convolutional neural network (CNN), Deep learning, Search-locality control, Non-local search.**

I. Introduction

Recently, performance of deep neural networks, especially convolutional neural networks (CNNs) [LeC 98], has been drastically improved by elaborate network architectures [Kri 12][Sim 15], by back propagation-based learning methods with various new techniques (including auto-encoding [Hin 06], dropout [Sri 14], and dropConnect [Wan 13]), and by graphical processing unit (GPU)-based high-performance computation.

Although many techniques for improving back-propagation learning have been developed, two major problems concerning back propagation remain to be solved.

The first problem is decision (or scheduling) of learning rate. The learning rate may be constant, or may vary, during the whole learning process. Several scheduling methods, including time-dependent scheduling [Rob 51] [Dar 92] and adaptive scheduling [Mag 99][Duc 11][Zei 12][Sch 13], have been proposed. However, most of them have sensitive hyper parameters that are difficult to tune because methods of adaptively calculating learning rate strongly depend on the properties and details of back propagation [Sch 13]. Accordingly, the hyper parameters must be manually tuned.

The second problem is to control the locality of search properly (i.e., avoid bad local minima and find a solution efficiently). Widely used gradient-descent algorithms are based on a local search, so they can be easily got stuck at a bad local minimum. Stochastic gradient-descent (SGD) algorithms, including SGD with mini-batch [Li 14], may find better local minima; however, they do not search the space globally. To find a better solution efficiently, multiple trials with different initial values and an elaborate search strategy are required.

This paper proposes a learning method called "learning-rate-optimizing genetic back-propagation" (LOG-BP), which combines back-propagation and a genetic algorithm (GA) by a new manner; that is, multiple neural networks run in parallel, and per-epoch genetic operations (i.e., mutations and selections) are applied to them. LOG-BP solves the two above-described problems by optimizing the learning process, especially the learning rate, by using the GA. This learning method does not depend on learning-rate usage, and it can even be applied to iterative learning methods without using neural networks and to optimization methods with performance-related parameters instead of learning rates. Initial experimental results shows LOG-BP performs better; that is, learning rates decrease exponentially, so the search autonomously becomes coarser to finer, and the distances between chromosomes, which indicate the locality of the search, also decrease exponentially.

The rest of this paper is organized as follows. Section II describes related work. Section III describes the proposed learning method, i.e., LOG-BP. Section IV describes an application of LOG-BP, i.e., image recognition. Section V discusses several issues concerning LOG-BP, and Section VI presents the conclusions of this work.

II. Related Work

Hundreds of research papers on various combinations of neural networks and stochastic search methods, especially genetic algorithms (GAs), have been published. In particular, combinations of back propagation and GAs have been extensively studied. The most-popular studies are on methods of optimizing structures and/or weights of neural networks by using GAs [Fer 05][Leu 03]. GAs are also used for optimizing learning methods [Bel 90] [Cha 90] [Mar 91]. Shaffer, et al. surveyed various combinations of neural networks and GAs [Sha 92]. In addition, as a combination of neural networks and other stochastic search algorithms, a recently developed and

978-1-5090-0620-5/16/$31.00

successful stochastic method, called "particle swarm optimization" (PSO) [Ken 95], is combined with neural networks [Men 02]. However, the purposes of these methods are not to optimize the process of back propagation, which is the major purpose of the learning method proposed in the present study, i.e., LOG-BP.

LOG-BP is a type of parallel learning method for neural network. As for research on parallelization, Zinkevich, et al. developed a search method that parallelizes SGD [Zin 10].

LOG-BP adaptively determines the learning rate. Several adaptive learning-rate scheduling techniques [Mag 99] [Duc 11] [Zei 12] [Sch 13] have been proposed. However, most of them have sensitive hyper parameters that are difficult to tune because adaptively calculating learning rate strongly depends on the details of back propagation [Sch 13], so they require manual tuning.

III. LOG-BP Learning Method

This section describes the outline and the detailed algorithm of LOG-BP, which is a combination of back propagation and GA by a new manner.

A. Outline

LOG-BP searches for the best solution, i.e., the best learning results, in parallel; that is, multiple neural networks, which are called "individuals", are trained in parallel by using back propagation.

Each individual contains a chromosome that contains the weights and biases of the network as well as the learning rate used for training the network. For example, if the network is an N-layer perceptron, chromosome c is represented by the following sequence:

$$c = (\eta;\ \mathrm{w}11, \mathrm{w}12, \ldots, \mathrm{w}1n_1, \mathrm{b}1;\ \mathrm{w}21, \mathrm{w}22, \ldots, \mathrm{w}2n_2, \mathrm{b}2;\ \ldots;\ \mathrm{w}N1, \mathrm{w}N2, \ldots, \mathrm{w}Nn_N, \mathrm{b}N)$$

where η is learning rate, wij $(1 \le j \le n_i)$ is all the weights of the i-th layer of the network, and bi is the bias of the i-th layer. For simplicity, in this paper, the same learning rate is used for all the weights; however, two or more learning rates can be coded into a chromosome. If the network is a CNN, wij is all the weights of the i-th convolutional layer of the network, and bi is the bias of the same layer. Other types of neural network can also be coded in the same manner.

A GA is applied to these chromosomes. They are evaluated and selected, and the best one is duplicated. Learning rate η is changed by applying a mutation operation. However, as for LOG-BP the weights and biases are only updated by back propagation, that is, not by mutation or crossover operations, which easily degrade the effect of back propagation and seriously decrease the performance of learning. No crossover operation is applied to learning rates either, because learning rates have no structures that make crossovers effective.

B. Learning algorithm

LOG-BP learning algorithm is described in **Figure 1**(a) and is visualized in Figure 1(b). In an execution of this algorithm, first, chromosomes are created and randomly initialized. The number of individuals is given as an input, but it can be changed in the learning process. The distribution of the weights and learning rates is also defined by the input. If normal distributions are used for them, standard deviations must be specified. If the mean weight is assumed to be zero, it is not necessary to be specified. However, the mean initial learning rate must be specified. As stated by Bengio, et al. [Ben 12], the initial learning rate should be relatively large (unless the rate of divergence is not very high).

The learning proceeds in the following way. After the initialization, it starts. Each epoch corresponds to a generation in the GA. The individuals (i.e., chromosomes) are evaluated and selected and then muted; that is, the learning rates may be muted randomly, and the weights and biases are "muted" by back propagation. This means that the weights and biases are not changed by an external process but are updated autonomously; that is, acquired characters are passed on to the next generation (i.e., epoch) in the Lamarckian way instead of the normal Darwinian way. In each epoch, leaning, evaluation, termination test, and selection and mutation are applied (in that order) as described in Figure 1(a). The learning process in this figure assumes use of SGD with mini-batches, but any other learning techniques controlled by learning rates (or similar learning parameters) can be applied.

The learning process is explained in more detail as follows. In the learning step, the individuals are trained using training data. In the evaluation step, they are evaluated using validation data. The criterion for evaluation can be arbitrarily designed according to network type or applications of learning; however, if a type of loss function is used, the lower value the better. In this case, the highest, i.e., the worst, individual is killed, and the lowest, i.e., the best, individual is duplicated. The learning rate of the new individual is muted by the following expression:

$$\eta' = f\eta \quad \text{(probability of 0.5)},$$
$$\eta' = \eta / f \quad \text{(probability of 0.5)},\ f \neq 1.$$

This expression means learning rate is increased with a probability of 0.5 or decreased with a probability of 0.5. Although the weights and biases are the same for the original and duplicated individuals, they become different values in the next epoch because the learning rates are different. For example, an appropriate value of f may be 1.3. The above mutation rule is the simplest one to update the learning rate randomly. A more elaborate method, such as application of a binomial distribution, can be applied. The number of individuals to be killed and to be duplicated may be randomly selected as described in Figure 1(a). If the number is ng, the worst ng individuals and the best ng individuals should be selected. The mean value of ng can be given as an input. If the chromosomes contain multiple learning rates, all of them may be muted in the same manner.

[Input]
- *Dt*: training data as a collection of mini-batches.
- *Dv*: validation data (and test data) as a collection of mini-batches.
- *Nc*: number of individuals (i.e., number of chromosomes, assumed to be constant during the whole process).
- Distribution and mean value of initial learning rate η_0.
- Distribution of initial weights (and biases) of the network. (If the distribution is a normal distribution and the mean value of the weights is zero, only standard deviation is specified.)
- *Ns*: average number of selections and mutations in an epoch.

[Initialization]
Generate initial values of chromosomes; that is, determine the weights, the biases, and the learning rate by random numbers.

[Procedure]
i := 1
repeat
[Epoch *i*.1: learning]
- Advance the back-propagation process by one epoch (i.e., the network is trained by using all training data *Dt* once).
- Update the weights and biases in the chromosomes.

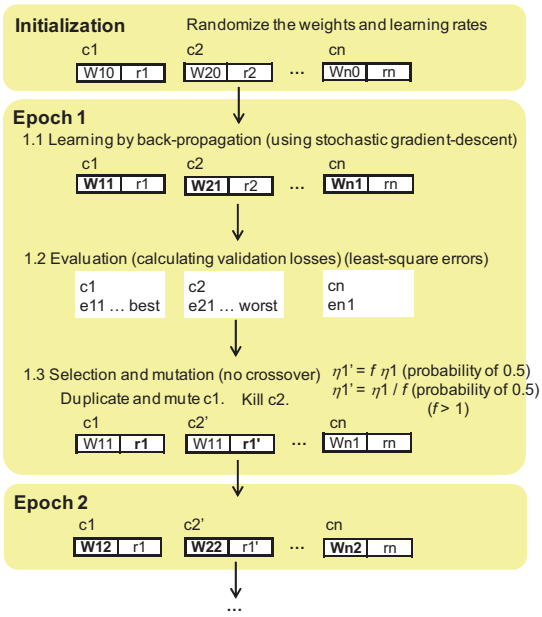
[Epoch *i*.2: evaluation]
Evaluate network performance by computing the loss function using validation set *Dv*.

[Termination test]
If the termination condition is met (e.g., if there is a chromosome with a sufficiently good evaluation result, terminate the process.

[Epoch *i*.3: selection and mutation]
- Decide number of selections and mutations, *ng* (≥ 0) by using *Ns* and a random-number generator.
- Repeat selection and mutation *ng* times; that is, delete chromosome *cw* with the worst value, and duplicate chromosome *cb* with the best value instead (to create *cb*') and update the learning rate of *cb* randomly by the following expression: $\eta' = f\eta$ (probability 0.5) or $\eta' = \eta / f$ (probability 0.5)

i := *i* + 1 -- Proceed to the next epoch.

(a) Learning algorithm



(b) Outline of algorithm

Figure 1. LOG-BP learning algorithm and outline

If the value of *ng* is appropriate, the locality of the search during the learning process is properly controlled. The range of the search becomes narrower as it proceeds. If the value of *ng* is appropriate, the individuals are initially spread across a global search space and perform efficient global search, and, as the learning process proceeds, they become closer because some individuals are duplicated and perform efficient local search in parallel. However, if the value of *ng* is too large, the range of the search decreases too quickly, so it becomes too local before the individuals find an appropriate area for a local search. If the value is too small the range decreases too slowly, so it is too wide, although they have already found appropriate areas for a local search. In this case, some individuals probably search around worse local minima, so the search is not efficient. The learning process continues until the termination condition is met (or until the user terminates the process).

IV. Application to Image Recognition

LOG-BP was applied to image-recognition tasks, specifically, pedestrian detection and hand-written character recognition.

A. Caltech benchmark – main example

The main benchmark used for evaluating LOG-BP in this study is the Caltech pedestrian dataset [Dol 09] [Cal]. This dataset is much larger than previously available pedestrian datasets and contains annotated videos with more than 190,000 pedestrians. The average height of pedestrians is approximately 50 pixels but this height is much smaller than that of most of other benchmarks, and any pedestrians are occluded, so this benchmark is known to be very hard.

A set of training data and test data, both of which are 24×48 and 32×64 images, were generated from the dataset. The mean pixel value of each image was adjusted to zero. Positive samples for training were generated from the video frames by using the bounding boxes in the annotations. The number of positive samples in the training set is 200,000, and that in the validation and test sets is 18,000 (including flipped images). The positive samples in these datasets are duplicated twice; that is, the total number of positive samples in the training set is 400,000 and that in the validation and test sets is 36,000. The number of samples in the test set is relatively small because the author intended to increase that in the training set. Negative samples were generated by randomly collecting the background images of the video frames that do not contain pedestrians. The number of negative samples in the training set is also 400,000, and that in the validation and test sets is also 36,000. The numbers of positive and negative images in each dataset are the same.

B. Environment for Caltech benchmark

The programs used for evaluating LOG-BP were described in Python 2.7 on Linux (Ubuntu 14.04LTS). To implement CNNs, Theano [Ber 11], which is an environment for deep learning, and cuDNN ver. 2. [Che 14] [NVI], which is

NVIDIA's deep-learning API, were used.

CNNs with two and three convolution layers were used for the evaluation. The structures of the CNNs are summarized in **Table 1**. CNN2 has two convolutional and two pooling layers, and CNN3 has three convolutional and three pooling layers. When the programs were written, a Theano-based CNN program in a deep-learning tutorial [DLT 15] was used as a reference. For both forward and backward calculations, an SGD method with mini-batches was used. The batch size was 250, and each mini-batch ontains the same set of images. The initial values of the weights are generated by uniform random numbers, and the bias values are zero. The range of weights is about (–0.03, 0.03), and the range depends on the layer.

Table 1. Structures of the CNN used for the evaluation

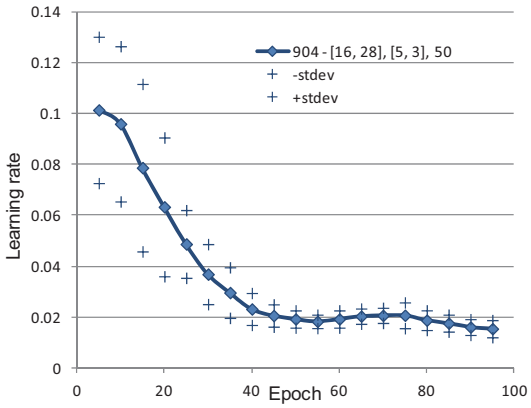
Layer	CNN2	CNN3
(0.c) 1st convolutional layer	Filter size: 5×5. Number of filters: 16. Activation function: ReLU (rectified linear).	
(0.p) 1st pooling layer	Max pooling. Pooling size: 2×2.	
(1.c) 2nd convolutional layer	Filter size: 3×3. Number of filters: 26 or 32. Activation function: ReLU.	Filter size: 3×3. Number of filters: 22. Activation function: ReLU.
(1.p) 2nd pooling layer	Max pooling. Pooling size: 2×2.	
(2.c) 3rd convolutional layer	-	Filter size: 3×3. Number of filters: 22. Activation function: ReLU.
(2.p) 3rd pooling layer	-	Max pooling. Pooling size: 2×2.
(3) Hidden layer (single)	Number of neurons: 50. Activation function: ReLU.	
(4) Output layer	Logistic regression (softmax). Number of neurons: 2.	

For the hardware environment, a GPU, NVIDIA GeForce GTX TITAN X, was used. Because this GPU has 12 GB of large high-speed GDDR5 memory, all 944,000 images can be stored in the GPU memory as 32-bit float data.

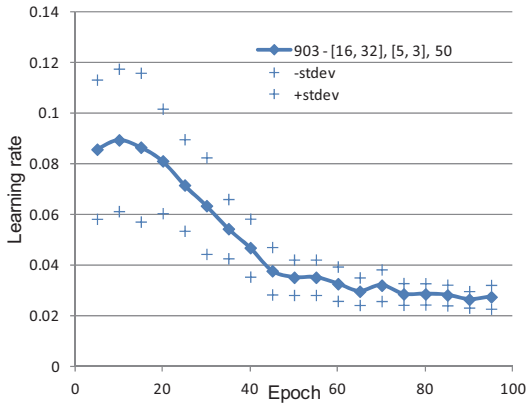
Theano compiles the programs into the machine code for the CPU and GPU when Theano function definitions are executed. The same compiled code is used for all the chromosomes in this evaluation because they use the same network structure. The values of the weights and biases are thus loaded, and the updated values are unloaded (i.e., moved between the memories of the CPU and GPU), every time the compiled code is used. The run time per epoch per individual in CNN2 was 31.2 seconds.

C. Observation: change of learning rate

The change of learning rate was observed by using CNN2 and CNN3. In two trials using CNN2 (called trials 2a and 2b here) with the data sets described above, the changes of the mean value and standard deviation of the learning rate are shown in **Figure 2**. The image size used in these trials was 24×48. The distribution of the learning rates is plotted every five epochs. Number of individuals was 12, and average number of chromosomes to be muted in an epoch was 0.6 for both trials. In trial 2a, shown in Figure 2(a), the average learning rate monotonically decreases until epoch 40, and it is mostly constant after that. The standard deviation decreases until epoch 65, and it slightly increases after that. Probably, the initial learning rate was appropriate, so the network converged very quickly. However, in trial 2b, shown in Figure 2(b), the learning rate is slightly increased before epoch 10, probably because the initial value was too low, so it was autonomously adjusted. This means, the learning rate can both decrease and increase in LOG-BP. This is different from Adagrad [Duc 11], in which the learning rate decreases monotonically. However, the learning rate mostly decreases from epoch 10 to 95, although it decreases very slowly after epoch 45.



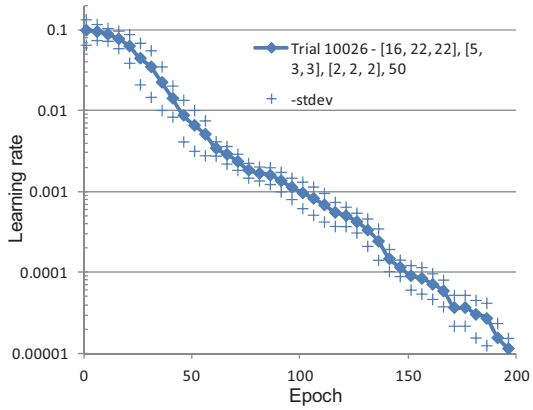
(a) A trial with 12 individuals (filters = [16, 26])



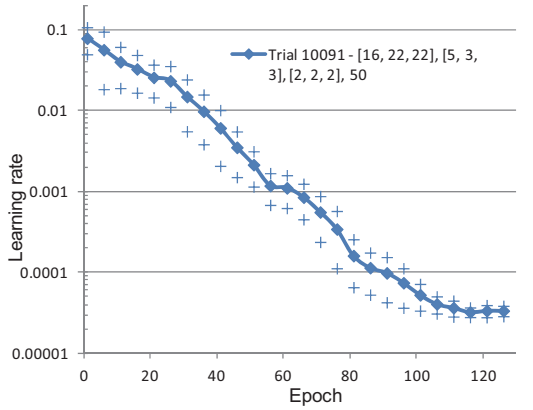
(b) A trial with 12 individuals (filters = [16, 32])

Figure 2. Chang of learning rate with the Caltech benchmark (CNN2)

The results of two trials with CNN3 (called "trials 3a" and "3b" hereafter) are shown in **Figure 3**. The image size used in these trials is 32×64. The dataset and numbers of individuals used for these trials are 24 for trial 3a and 12 for trial 3b. The average number of chromosomes to be muted in an epoch is 1.0 for trial 3a and 0.5 for trial 3b. The development process is quite different from that of CNN2. That is, the learning rate exponentially decreased three to four orders of magnitude.



(a) A trial with 12 individuals
(mutation rate = 8.3% (1/12))



(b) A trial with 24 individuals
(mutation rate = 4.2% (1/24))

Figure 3. Change of learning rate with the Caltech benchmark (CNN3 with filters [16, 22, 22])

In trial 3a, shown in Figure 3(a), the learning rate continuously decreases (mostly) exponentially (e^{-at}) until the learning terminated at epoch 200; that is, it may be much faster than t^{-1}, which has been conventionally used for non-adaptive scheduling.

In contrast, in trial 3b, shown in Figure 3(b), the decrease seems end by epoch 115. The cause of this difference between the learning-rate changes is unknown.

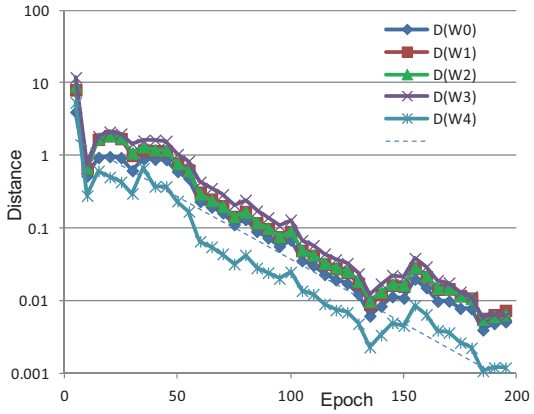
The results of these trials can be summarized as below.

- Learning rate usually decreases, and it may become mostly stationary at some epoch or it may continuously decrease by more than three orders of magnitude. And the decrease rate may be exponential.
- The learning rate may increase if the initial value is too small.

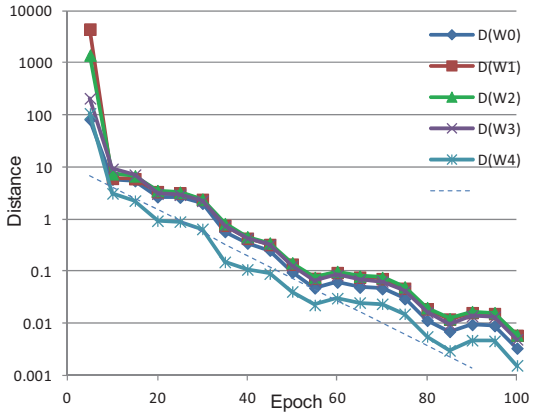
D. Observation: locality

To measure the locality of search, the distance between each layer of a neural network (chromosome) and that of the best network was measured and plotted every five epochs. The distance between each layer of two networks is defined by the Euclidian distance between the weight vectors of the networks. The reason the distance between two whole chromosomes was not calculated is that each layer may have a different nature and the average initial distances are different.

The measured learning rates for trial 3a with CNN3 are shown in **Figure 4**(a), and those for trial 3b are shown in Figure 4(b). D(Wi) means the average distance between the (i+1)-th layer from the best chromosome in the epoch. The 1st to 3rd layers are the convolutional layers (see Table 1). All the distances decrease exponentially in the same manner as the learning rate. This figure shows the locality of the search could be controlled by LOG-BP algorithm. Although the range selected for the search does not necessarily contain global minima or good local minima, good local minima can probably be captured by an elaborate design and application of the GA.



(a) A trial with 12 individuals



(b) A trial with 24 individuals

Figure 4. Change of average weight-distance with the Caltech benchmark (CNN3 with filters [16, 22, 22])

The results of these trials can be summarized as below.

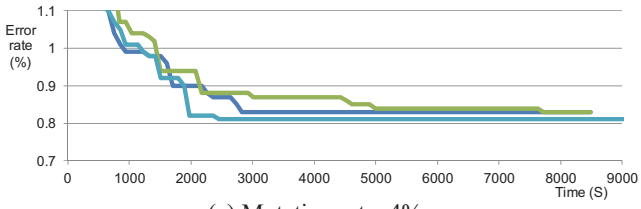
- The distances of each layer decrease mostly in parallel (in a similar way).

- The distances decrease almost exponentially, and they may continuously decrease by more than three orders of magnitude. (The decrease rate probably depends on the mutation ratio and the number of individuals.)
- The distances may increase drastically (four times or more) for a while but decrease after that.

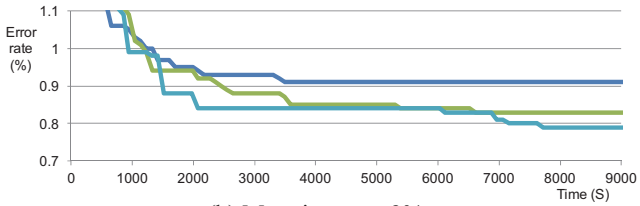
E. MNIST benchmark – another example

This section shows that in the case that a constant learning rate works well, LOG-BP may still have benefits in terms of parallel-search performance.

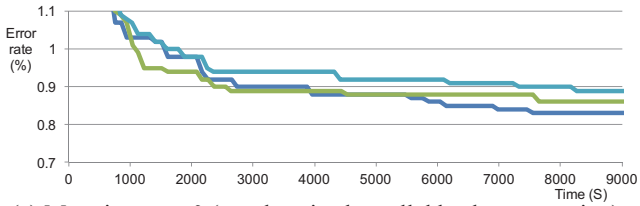
Learning rate does not necessarily always decrease. For example, when LOG-BP was applied to the MNIST benchmark [LeC 98], which is a set of hand-written digit images (with size of 28×28) containing a training set with 60,000 samples and a test set with 10,000 samples, the learning rate during the whole learning process was around 0.1. (A program described in the deep-learning tutorial uses a constant learning rate of 0.1 and it works well.) This means that the first advantage of LOG-BP, i.e., adaptive determination of learning rate, is not required for this benchmark.



(a) Mutation rate: 4%



(b) Mutation rate: 2%



(c) Mutation rate: 0 (synchronized parallel back-propagation)

Figure 5. Performance of the MNIST benchmark (CNN3 with filters [16, 22, 22])

Although learning rate is mostly constant, as described above, selection and mutation operations in LOG-BP still seem advantageous in regard to the MNIST benchmark. Nine typical changes of error rates (i.e., loss-function values) are shown in **Figure 5**. The mutation rate is 4% in the first three runs, shown in Figure 5(a), 2% in the second three runs, shown in Figure 5(b), and 0% in the third three runs, shown in Figure 5(c). (In this application of LOG-BP, maximum mutation rate is 4%, which is still relatively low, because the number of individuals is 25 and the maximum number of muted chromosomes in an epoch used with this experiment was restricted to one.) A run with mutation rate of 0% means a simple synchronized parallel execution, which is close to repeated executions of an SGD algorithm. The results of these runs are shown in **Table 2**. This table lists average error-rate convergence time and final error rates. The latter is the error rate when run time reach 9000 seconds. The error rate may decrease after that time, but it seems to converge in most cases. It is thus called "final error rate," *ef*, here, and the convergence time means the time error rate becomes lower than 1.05 *ef*. This table shows that convergence time is shorter and final error rate is smaller when mutation rate is larger. This result shows the selection and mutation operations are advantageous for the MNIST benchmark and suggests advantages over SGD algorithms.

Table 2. Convergence time of the MNIST benchmark (CNN3 with filters [16, 22, 22])

Mutation rate	Average convergence time (std. dev., s)	Final error rate (std. dev., %)
4%	2.6×10^3 (0.5×10^3)	0.82 (0.01)
2%	4.9×10^3 (2.6×10^3)	0.84 (0.07)
0%	5.6×10^3 (0.9×10^3)	0.86 (0.03)

V. Discussions

Five issues concerning LOG-BP and its extensions are discussed in the following.

A. Parameters for parallel-search

Parameters for parallel search, such as recommended number of individuals, recommended number of selections and mutations are discussed first.

The number of individuals used in the trials described above is 12 to 25 because of the balance between non-locality and computation time. The initial weight values of the individuals can be spread around the search space, and each individual searches for a solution independently. If more individuals exist, the whole search becomes non-local (i.e., wider). If the number of individuals is around 10 to 20, the search is not sufficiently non-local because the selection and mutation operation reduces the variability of chromosomes. The number of search areas, which originally equals the number of individuals, soon reduces to, for example, three. However, if a large number of individuals is used, the learning takes too much time because total run time is proportional to number of individuals. The locality of the search can be measured by the distances between chromosomes, such as shown in Figure 4(a).

The frequency and the probability of the selection-and-mutation operations must be properly controlled because the operation reduces the locality of the search (as described

above). If the number of individuals is around 10 to 20 and the operations are invoked in every epoch, the best probability seems to be around 5%; that is, when the number of all individuals is 15, the number of individuals to be replaced should be around 0.75.

B. Divergence of chromosomes

The weights can often diverge, i.e., they become "NaN", in a back-propagation step; however, diverged individuals, which are called "zombies" here, can be handled either normally or by a specially-designed way in LOG-BP. If the learning rate is too large, the weights can easily diverge. Because the performance is the worst, they are soon killed by the selection operation; thus, it is not necessary to introduce an additional operation that explicitly kills them. However, if multiple individuals diverge at the same time, the zombies might not be killed for a long time and waste the computing resources. Because they never generate fruitful results, it is probably better to introduce an additional operation to kill them.

C. Optimizing test performance by dual evaluation

To get better results for unknown datasets, the selection and mutation operation, which can be regarded as divine, may use the evaluation results of the test set, which are not normally used. Conventionally, the behavior of the neural network should never be influenced by the evaluation results of the test set. If the results are used for learning, the role of the test set becomes undistinguishable from that of the validation set. However, as for LOG-BP, the individuals and the selection and mutation operations, which is independent of behaviors of individuals and can be regarded as a divine process, may have different criteria for evaluation. The "god," i.e., the selection and mutation operator, thus may use the evaluation results of the test set. Because this usage adapts the individuals to the test set, one more test set, whose evaluation results are neither referenced by individuals nor by the "god" is required for estimating the performance. This dual-evaluation method has not yet been much compared with the single-evaluation method described in Section III, so the evaluation of this evaluation method is part of our future work.

D. Structural optimization

The structure and hyper-parameters of neural networks are not altered in the case of the basic LOG-BP algorithm described in Section III; however, because the algorithm is based on a GA, the structure and hyper-parameters can be optimized by extending the algorithm. This means that mutation or crossover of network structures or hyper-parameters can be introduced. For example, neuron-deletion operations, such as dropout [Sri 14], neuron-addition operators, connection-deletion operations, such as dropConnect [Wan 13], and connection-addition operations, can be introduced. More complex operations such as addition or deletion of layers can also be introduced. However, none of these structural optimizations have been tested.

If addition of neurons or connections is allowed, the loss function of individuals probably must contain a term for suppressing "structural explosion." That is, the structures of individuals may be become larger and more complex because larger networks have more chances to obtain better results, so it is probably necessary to restrict this tendency by, for example, introducing minimum description length (MDL) [Bar 98] to the loss function.

E. Application to other types of machine learning and optimization

LOG-BP can be extended to machine learning other than back propagation, and it can even be applied to some optimization methods. The basic idea of LOG-BP does not depend on the learning method of each individual or the method of learning-rate usage. Although "LOG-BP" means a combination of back propagation and a GA, back propagation can be replaced by another iterative learning method. This means it can probably be applied to learning methods without using neural networks. In each iteration step (which corresponds to an epoch) of such an application, the learning result is evaluated and a learning-performance-related parameter, which corresponds to learning rate, is muted. It may also be applicable to optimization methods with performance-related parameters.

VI. Conclusion

A learning method called "learning-rate-optimizing genetic back-propagation" (LOG-BP), which combines back propagation and a genetic algorithm (GA) by a new manner, is proposed. LOG-BP solves two problems concerning back propagation, i.e., scheduling of learning rate and controlling locality of search, by optimizing the learning process, especially learning rate, by locality-controlled parallel search and by genetic mutations. The initial experimental results of two benchmarks show high performance of this method. Although the proposed method was not directly compared with conventional SGD algorithms, the MNIST benchmarking suggests advantages over them. This method will make machine learning less dependent to properties of various applications and machine learning tasks easier.

The most important part of future work is to compare LOG-BP with conventional learning methods. Future work also includes optimization of LOG-BP by finding optimum values for the number of individuals and optimum distributions of the initial values and by improving the evaluation method, especially the loss function. In addition, LOG-BP should be applied to machine-learning methods other than back propagation and to optimization methods.

Acknowledgments

The author thanks Takuma Shibahara, Masahiro Ogino, Yasuhiro Akiyama, Koichi Hamada, and other members of CTI, Hitachi for commenting on LOG-BP and for comparing it with conventional methods. The author also thanks Takehito

Ogata, Yoshitaka Uchida, and other members of Clarion for commenting on LOG-BP and its applications.

REFERENCES

[Bar 98] Barron, A., Rissanen, J., and Bin Yu, "The Minimum Description Length Principle in Coding and Modeling", *IEEE Transactions on Information Theory*, Vol. 44, No. 6, pp. 2743–2760, October 1998.

[Bel 90] Belew, R. K., Mcinerney, J., and Schraudolph, N. N., "Evolving Networks: Using the Genetic Algorithm with Connectionist Learning", *2nd Artificial Life Conference*, pp. 511–547, Addison-Wesley, 1990.

[Ben 12] Bengio, Y., "Practical Recommendations for Gradient-based Training of Deep Architectures", *Neural Networks: Tricks of the Trade*, pp. 437–478, Springer, 2012.

[Ber 11] Bergstra, J., Bastien, F., Breuleux, O., Lamblin, P., Pascanu, R., Delalleau, O., Desjardins, G., Warde-Farley, D., Goodfellow, I., Bergeron, A., and Bengio, Y., "Theano: Deep Learning on GPUs with Python", *Journal of Machine Learning Research*, Vol. 1, pp. 1–48, 2011.

[Cal] Caltech Pedestrian Detection Benchmark, http://www.-vision.caltech.edu/Image_Datasets/CaltechPedestrians/

[Cha 90] Chalmers, D. J., "The Evolution of Learning: An Experiment in Genetic Connectionism", In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton (eds.), "1990 Connectionist Models Summer School", Morgan Kaufmann, 1990.

[Che 14] Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., and Shelhamer, E., "cuDNN: Efficient Primitives for Deep Learning", arXiv preprint arXiv:1410.0759, 2014.

[Dar 92] Darken, C., Chang, J., and Moody, J., "Learning Rate Schedules for Faster Stochastic Gradient Search", Revised and expanded version, August 1992, Original version: *IEEE 2nd Workshop on Neural Networks for Signal Processing*, 1992.

[DLT 15] "Deep Learning Tutorials – Convolutional Neural Networks (LeNet)", Deep Learning 0.1 Documentation, August 2015, http://www.deeplearning.net/tutorial/lenet.html

[Dol 09] Dollár, P., Wojek, C., Schiele, B., and Perona, P., "Pedestrian Detection: A Benchmark", *IEEE Conference on Computer Vision and Pattern Recognition 2009* (*CVPR 2009*), pp. 304–311, June 2009.

[Duc 11] Duchi, J., Hazan, E., and Singer, Y., "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", *Journal of Machine Learning Research*, Vol. 12, p. 2121–2159, February 2011.

[Fer 05] Ferentinos, K. P., "Biological Engineering Applications of Feedforward Neural Networks Designed and Parameterized by Genetic Algorithms", *Neural Networks*, Vol. 18, No. 7, pp. 934–950, September 2005.

[Hin 06] Hinton, G. E. and Salakhutdinov, R. R., "Reducing the Dimensionality of Data with Neural Networks", Science 313.5786, pp. 504–507, 2006.

[Ken 95] Kennedy, J. and Eberhart, R., "Particle Swarm Optimization", *1995 IEEE International Conference on Neural Networks*, Vol. 4, pp. 1942–1948, November/December 1995.

[Kri 12] Krizhevsky, A., Sutskever, I., and Hinton, G. E., "ImageNet Classification with Deep Convolutional Neural Networks", *Advances in Neural Information Processing Systems 25* (*NIPS 2012*), pp. 1097–1105, 2012.

[LeC 98] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P., "Gradient-based Learning Applied to Document Recognition", *Proceedings of the IEEE*, Vol. 86, No. 11, pp. 2278–2324, 1998.

[Leu 03] Leung, F. H. F., Lam, H. K., Ling, S. H., and Tam, P. K. S., "Tuning of the Structure and Parameters of a Neural Network Using an Improved Genetic Algorithm", *IEEE Transactions on Neural Networks*, Vol. 14, No. 1, pp. 79–88, January 2003.

[Li 14] Li, M., Zhang, T., and Chen, Y., "Efficient Mini-batch Training for Stochastic Optimization", *20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (*KDD 2014*), pp. 661–670, 2014.

[Mag 99] Magoulas, G. D., Vrahatis, M. N., and Androulakis, G. S., "Improving the Convergence of the Backpropagation Algorithm using Learning Rate Adaptation Methods", *Neural Computation*, Vol. 11, No. 7, pp. 1769–1796, 1999.

[Mar 91] Marshall, S. J. and Harrison, R. F., "Optimization and Training of Feedforward Neural Networks by Genetic Algorithms", *2nd International Conference on Artificial Neural Networks*, pp. 39–43, November 1991.

[Men 02] Mendes, R., et al., "Particle Swarms for Feedforward Neural Network Training", *2002 International Joint Conference on Neural Networks* (*IJCNN 2002*), Vol. 2, pp. 1895–1899, May 2002.

[Mor 95] Moreira, M. and Fiesler, E., "Neural Networks with Adaptive Learning Rate and Momentum Terms", IDIAP Technical Report 95-04, October 1995.

[NVI] NVIDIA cuDNN – GPU Accelerated Deep Learning, NVIDIA CUDA Zone, https://developer.nvidia.com/cudnn

[Pio] Piotr's Computer Vision Matlab Toolbox, http://vision.ucsd.edu/~pdollar/toolbox/doc/

[Psy] psycharo/read_seq.py, https://gist.github.com/psycharo/-7e6422a491d93e1e3219/, GitHub Gist

[Rob 51] Robbins, H. and Monro, S., "A Stochastic Approximation Method", *The Annals of Mathematical Statistics*, pp. 400–407, 1951.

[Sal 91] Salomon, R., "Improved Convergence Rate of Backpropagation with Dynamic Adaption of the Learning Rate", In "Parallel Problem Solving from Nature", *Lecture Notes in Computer Science*, Vol. 496, pp 269–273, 1991.

[Sal 92] Salomon, R. and van Hemmen, J. L., "A New Learning Scheme for Dynamic Self-Adaptation of Learning-Relevant Parameters", *International Conference on Artificial Neural Networks* (*ICANN-92*), pp. 1047–1050, 1992.

[Sch 13] Schaul, T., Zhang, S., and Lecun, Y., "No More Pesky Learning Rates", *30th International Conference on Machine Learning* (*ICML-13*), pp. 343–351, 2013.

[Sha 92] Schaffer, J. D., Whitley, D., and Eshelman, L. J., "Combinations of Genetic Algorithms and Neural Networks: a Survey of the State of the Art", *International Workshop on Combinations of Genetic Algorithms and Neural Networks* (*COGANN-92*), pp. 1–37, Jun 1992.

[Sim 15] Simonyan, K. and Zisserman, A., "Very Deep Convolutional Networks for Large-scale Image Recognition", arXiv preprint arXiv:1409.1556, 2014.

[Sri 14] Srivastava, N., Hinton, G., Krizhevsky, A., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, Vol. 15, No. 1, pp. 1929–1958, 2014.

[Sta] Reading .seq File with Matlab, http://stackoverflow.com/questions/28836147/reading-seq-file-with-matlab, Stack Overflow

[Wan 13] Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., and Fergus, R., "Regularization of Neural Networks Using DropConnect", *30th International Conference on Machine Learning* (*ICML-13*), pp. 1058–1066, 2013.

[Zei 12] Zeiler, M. D., "ADADELTA: An Adaptive Learning Rate Method", arXiv preprint, arXiv:1212.5701, 2012.

[Zin 10] Zinkevich, M., Weimer, M., Li, L., and Smola, A. J., "Parallelized Stochastic Gradient Descent", *Advances in Neural Information Processing Systems*, pp. 2595–2603, 2010.