Application of Genetic Algorithm to Convolutional Neural Networks

Dae-Gon Yang, Sang-Soo Park, and Ki-Seok Chung
Department of Electronics and Computer Engineering, Hanyang University
Haengdang 1-dong, Seongdong-gu, Seoul, 133-791, South Korea
dgwgdgwg@naver.com, sonicstage12@naver.com, kchung@hanyang.ac.kr

Abstract

Convolutional Neural Network (CNN) is used extensively from digit recognition to natural language processing. Commonly, the error back propagation method is used to learn the CNN models. However, when learning with the error backpropagation method, the learning efficiency is not sufficiently high due to a problem called the vanishing gradient problem as the CNN model becomes deeper. In the feedforward neural networks, the error is propagated in the opposite direction, which makes it difficult for the human to intuitively understand. Therefore, the implementation of the learning algorithm is difficult. If the weights of a neural network model are regarded as genes and if we let the system learn these genes by evolution, the learning process may become more intuitive and efficient than the existing backpropagation method. In this paper, how to learn the model by applying a genetic algorithm is addressed. The simplest fully-connected model and the LeNet-5 model with MNIST, which is a widely used dataset for handwritten digits recognition are used to confirm the effectiveness of the proposed method.

Keywords: Genetic Algorithm, Convolutional Neural Network (CNN), Machine Learning, Evolutionary Computation.

1. Introduction

Recently, Convolutional Neural Network (CNN) has been widely used in the field of object recognition. The neural network model has been developed by increasing the scale of the model and it has become deeper to obtain higher accuracy [1]. However, in the backpropagation learning method, learning efficiency becomes worse as the network becomes deeper mainly due to a problem called vanishing gradient problem [2]. Even though some models can effectively make inferences, they are less portable and inflexible because their network structures can only be used in specific applications [3]. Due to these reasons, implementation based on conventional methods may not lead to an efficient design.

In this paper, we propose a method to learn neural networks with a genetic algorithm. The generic algorithm is advantageous in the sense that it is intuitive and applicable to a wide range of optimization problems, and therefore, our proposed implementation does not change structure or format of the existing neural network model. In particular, the generic algorithm can be used as a partial learning method to alleviate the vanishing gradient problem of the existing backpropagation learning method. In addition, it has a better chance to get good results than the backpropagation method because the genetic algorithm is less likely to get stuck in local minima.

Section 2 describes the genetic algorithm, and Section 3 describes how the genetic algorithm was applied to the neural network. The experimental results are shown in Section 4 and the paper is concluded in Section 5.

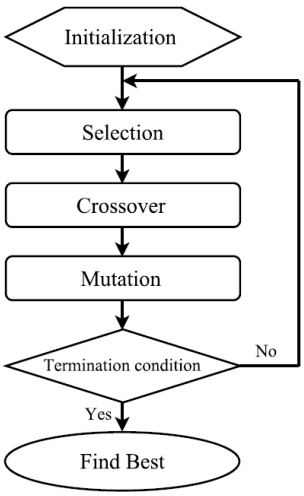

Figure 1. Overall process of the genetic algorithm

2. Genetic Algorithm

The genetic algorithm is a representative method of evolutionary computation that solves problems by mimicking real biological evolution. Based on natural genetics, Darwin's theory of survival of the fittest is the basic concept. After listing genes which are the set

of the solutions of the problem, we gradually change the genes to produce better solutions.

Figure 1 shows the overall process of the genetic algorithm. The entire process is performed in the order of initialization, repetition of selection-crossover-mutation, and termination [4].

The initialization operation expresses the solution to solve with the genetic algorithm as a gene. Just as the genetic traits of a living organism are represented by genes that are a collection of genomes, the set of solutions is expressed as a gene through a data structure such as an array of numbers or a string. To represent the solutions, a specific number of genes are initialized. Genotyping methods include binary, integer, real value, and permutation, and strategies for initializing the genes are random, heuristic, and so on.

The selection operation measures the fitness for each gene and selects the candidate of the parent from each generation to the next generation. The selection strategy includes roulette wheel, stochastic universal sampling, tournament, rank, and random.

The crossover operation generates next generation's genes through crossing between the selected parent candidate genes. Generally, two parents are selected and crossed between each other. Through the crossover of the parents, they construct a new gene called child by receiving parent's genetic factor at a specific location. The crossover strategy includes one-point, multi-point, uniform and whole arithmetic recombination.

The mutation operation mimics the natural phenomenon that the result of the preceding crossing operation may be irrelevant to the crossover strategy. This can increase the diversity of the gene so that it does not fall into the local optimum. The mutation strategy includes bit-flip, swap, scramble, inversion, and random.

Through repetition of these operations, only those genes that adapt to the environment with high fitness survive and these good genes are spread to the future, and those that do not cannot survive are abandoned automatically. If this process is repeated while a certain termination condition is met, the best gene among the remaining children is selected as the final solution and the process is terminated.

3. Implementation of Genetic Algorithm on Neural Networks

In order to apply the genetic algorithm to the neural network, we apply the neural network's weight as a gene and let the genes evolve. We use the MNIST dataset to learn the neural network with a genetic algorithm. We also use a fully-connected model and a LeNet-5 model as neural network models.

The MNIST dataset used in numerical recognition consists of 60,000 training sets and 10,000 test sets which are classified into 10 classes from 0 to 9 black and white handwritten digits of 28 * 28 pixels images [5].

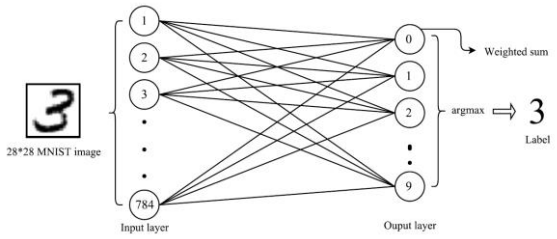

Figure 2. Fully-connected model

The fully-connected model infers a result by deducting argument of the maxima of the output, which is the sum of the 28 * 28 pixels MNIST image multiplied by 7,840 weights as shown in Figure 2.

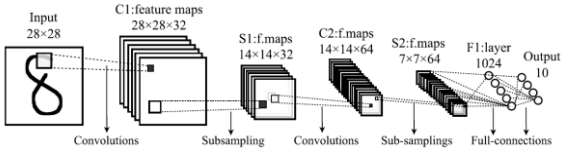

Figure 3. LeNet-5 model

Figure 3 shows the LeNet-5 model inferring the MNIST image, which can be used with high accuracy [6].

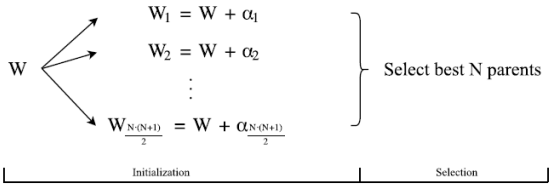

Figure 4. Initialization and selection process

In the initialization process, a weight vector listing all the weights is expressed as a gene. In this process, we used the real type, which is a common way of expressing weight values in general CNN models.

A combination of a heuristic method and a random method is employed to initialize genes. The heuristic method helps to speed-up learning by using already known information. We use a weight vector that has been pre-trained with about 92% accuracy in the fully-connected model and 99% in the LeNet-5 model. In addition, we add a random real type value α as truncated normal distribution to the existing weight vector. In this way, we have initialized $\frac{N*(N+1)}{2}$ parents as shown in Equation (1) as in the initialization part of Figure 4.

$$W_n = W + \alpha_n \quad (1)$$

(W: Initial weight vector, $\mu(\alpha_n) = 0$, $\sigma(\alpha_n) = 0.02$)

In the selection process, the fitness of $\frac{N*(N+1)}{2}$ genes are measured using a training set with a batch size of 1000 randomly selected for each generation. Fitness was calculated as the accuracy rate of the 1000 randomly selected images. Rank-based selection is simply picking out a certain number of best genes among the parent generation. We select N parents which will be transferred to the next generation from W_1 to W_N based on the rank of the genes with the highest fitness among the parents.

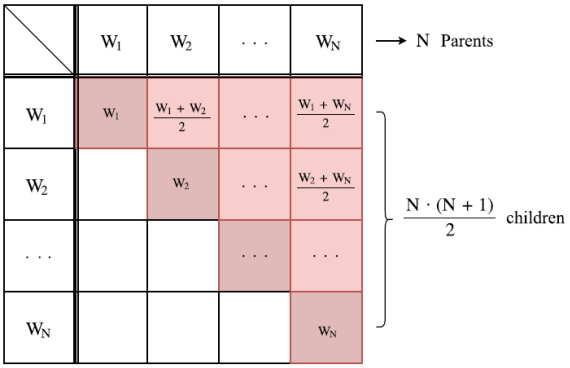

Figure 5. Child crossing with parents

In the crossing process, whole arithmetic recombination is used as a crossing strategy, crossing the N parents selected from W_1 to $W_{N,}$ respectively. This strategy is often used when the genotype of the parent is the real. If W_x and W_y are the selected genes, the gene of a child can be computed as Equation (2).

$$W_{child} = t * W_x + (1 - t) * W_y \qquad (2)$$

We set t = 0.5 so that the child can inherit the parental characteristics evenly. Therefore, it is no longer necessary to calculate the other half because of symmetric values as shown in Figure 5. Additionally, each W automatically inherits the existing parental traits in the next generation, ensuring that the genes of the next generation are not worse than those of the parental generation. The $\frac{N*(N+1)}{2}$ child generations of the next generation are made with crossover of N parent generations.

Mutations should be present for the diversity of genes in the crossing process. But it is more likely to be worse than the likelihood of a better outcome in a real value weight vector, and is naturally culled. For this reason, we skip the mutation process.

These $\frac{N*(N+1)}{2}$ child genes are the genes of the next generation of parents. After repeating the P-generation of the selection-crossover process, the gene with the highest fitness value is used as a final result.

We set N = 15 and repeat the genetic operation by the number of generations P = 10. After the repetition is completed, the best gene is used as the result.

4. Experimental Results

Table 1. Fitness for Fully-Connected model

Gen	Best(%)	Worst(%)	Avg(%)
0	92.16	91.89	92.13
1	92.41	92.23	92.38
2	92.55	92.43	92.51
3	92.59	92.57	92.58
4	92.61	92.55	92.59
5	92.63	92.59	92.61
6	92.64	92.62	92.63
7	92.65	92.64	92.64
8	92.65	92.65	92.65
9	92.65	92.65	92.65
10	92.65	92.65	92.65

Table 2. Fitness for LeNet-5 model

Gen	Best(%)	Worst(%)	Avg(%)
0	99.18	99.11	99.15
1	99.22	99.20	99.19
2	99.23	99.21	99.20
3	99.24	99.23	99.23
4	99.24	99.24	99.23
5	99.24	99.24	99.23
6	99.24	99.24	99.23
7	99.25	99.24	99.24
8	99.25	99.25	99.25
9	99.25	99.25	99.25
10	99.25	99.25	99.25

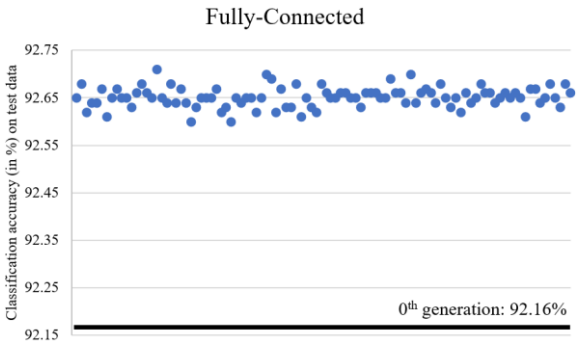

Figure 6. Distribution of Fully-Connected model results after 100 experiments

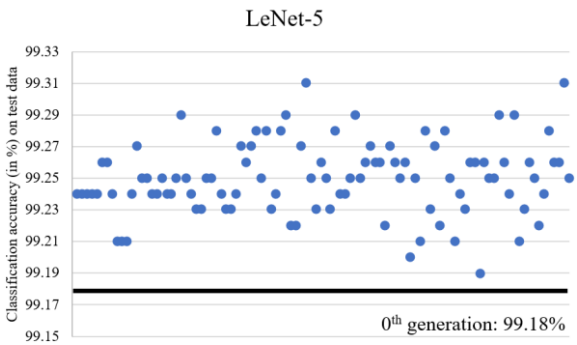

Figure 7. Distribution of LeNet-5 model results after 100 experiments

As shown in Table 1 and Table 2, we can observe that the genes have already converged to a good gene before the 10^{th} generation.

When the random method is used for initialization, the weights can be different. But, even with same weights, the results can be different since the batch is collected randomly for each generation. Figure 6 shows the distribution of results with 100 experiments under the same conditions.

In Figure 6, the fully-connected model shows an accuracy of up to 92.71%, with the average accuracy of 92.65%. As shown in Figure 7, the LeNet-5 model shows an accuracy of up to 99.31%, with the average accuracy of 99.24%. In Figure 6 and Figure 7, the accuracy of Generation 0 is 92.16% and 99.18%, respectively, and the accuracy is improved by learning the model using the proposed genetic algorithm.

5. Conclusion

In this paper, we apply the genetic algorithm to two types of neural network models: the fully-connected model and the LeNet-5 model, using the MNIST dataset used for numerical recognition. Since the genetic algorithm is flexible and widely applicable, one can automatically learn the model without modifying the model or adjusting the value format.

Also, in general, partial learning is impossible because we do not know what the hidden layer output is. However, in the genetic algorithm, it is possible to select a layer or region to be partially learned, and then automatically enhance the portion.

Since the genetic algorithm is an evolutionary technique that directly simulates the phenomenon, unlike existing machine learning techniques, the computational complexity is very high. However, as the model grows deeper and predicting or controlling a single layer directly comes harder, it is expected that the usage of genetic algorithm can leverage automatic learning of the model in more complex environments.

6. Acknowledgement

This work was supported by the Technology Innovation Program (10076583, Development of free-running speech recognition technologies for embedded robot system) funded By the Ministry of Trade, Industry & Energy (MOTIE, Korea).

References

[1] Simonyan, Karen, and Andrew Zisserman, “Very deep convolutional networks for large-scale image recognition”, ICLR, 2015.

[2] Bengio, Yoshua, Patrice Simard, and Paolo Frasconi, “Learning long-term dependencies with gradient descent is difficult”, *IEEE transactions on neural networks* 5.2, pp.157-166, 1994.

[3] Pan, Sinno Jialin, and Qiang Yang, “A survey on transfer learning”, *IEEE Transactions on knowledge and data engineering*, pp. 1345-1359, 2010.

[4] Whitley, Darrell, “A genetic algorithm tutorial” *Statistics and computing*, pp. 65-85, 1994.

[5] LeCun, Yann, Corinna Cortes, and C. J. Burges, “MNIST handwritten digit database”, *AT&T Labs, http://yann.lecun.com/exdb/mnist*, 2010.

[6] LeCun, Yann, “LeNet-5, convolutional neural networks”, *http://yann.lecun.com/exdb/lenet*, 2015.