Genetic Algorithm for Simple Neural Network Optimization

Jiabo Xu

Research School of Computer Science
Australian National University
ACT 2601 AUSTRALIA
u6435287@anu.edu.au

Abstract. This paper mainly studies the genetic algorithm performance on standard neural network training (weight optimization). Besides, two selection strategies together with two crossover and two mutation method of genetic algorithm have been implemented and compared to further explore the feature of weight optimization. The selection method includes proportional selection, which selects each population with probability depending on their fitness, and elitism selection, which always keeps the top fitness population for each generation. The final part is the analysis and comparison between classical optimizer, such as SGD and Adam, and genetic algorithm on doing same tasks. Same as expectation, genetic algorithm performance worse in limited time than gradient method.

Keywords: Genetic algorithm, Neural Network, Weight Optimization, Shared Weight

1 Introduction

Genetic algorithm is one of famous evolutionary algorithm which is used for optimization and search problem. The optimization problem includes the neural network weight optimization, which is in float-representation, and hyperparameter optimization, which is in discrete representation. Compared with classical optimizer, such as gradient decent and momentum method, genetic algorithm is good at finding the global optimum while the gradient methods are likely to be trapped by local optimum. However, in terms of neural network training, especially the supervised learning network, gradient method is known to be the standard optimization method because local optimum is no much different from global optimum in general cases (Bengio, 2009). Thus, applying genetic algorithm on neural network may result in worse solution than gradient method. It should not be the reason that prevents people from utilizing the method, instead genetic algorithm will help people understand neural network better.

There are five main steps on doing genetic algorithm, representation, fitness function, population selection, crossover and mutation (Whitley, 1995). The representation part is to convert the data into binary representation so that crossover and mutation can be used on the form easily. The fitness function is the judgement on how good the population is. With respect to neural network, the fitness function is the loss function. Population selection consists of parent selection and the offspring selection. The parent selection is used to pick proper parents from current population, usually the most adapted ones, to generate offspring for next generation. The offspring selection or called replacement, is for determining which population to be replaced by which offspring. In a word, the meaning of selection step is to keep better population which can yield better results for future. The crossover step generates new population by recombining part of allele from one parent with part of allele from another so that better genotype may be found after sufficient generations. However, if offspring only own genes (features) from his ancestors, the diversity (search space) will be limited. Hence, mutation is applied for rising the variety. In optimization thinking, mutation expands the search space from local to global. Consequently, Different problem uses different strategies of the five steps. In the following section, strategies applied for neural network weight optimization will be illustrated in detail.

2 Method

As the task focus on the performance of genetic algorithm for auto-associate neural network weight optimization, the method will only talk about the feature of core components of genetic algorithm rather then neural network. In detail, the section contains data and preprocessing, overall method, selection and replacement, crossover, mutation.

2.1 Data and Preprocessing

Different from data used for general auto-associate neural network, the experiment in this paper focus on the structure of genetic algorithm. Thus, only “Lenna” is used. The first reason for using the data is because it is simple enough. It is grayscale, one channel image and the size is also small 64 by 64. The second reason is that, in terms of the data amount, training the network to be able to reconstruct many images is quite hard for genetic algorithm. In fact, it is not easy to fit the single image in no more than 1000 generations.

In terms of preprocessing, “Lenna” is separated into 16 identical size patterns. Each pattern is a 16 by 16 sub-image (Gedeon, 1995). As training with genetic algorithm is distinguished from gradient method, it regards the overall loss as the fitness of the model (population). Therefore, there is only one batch rather than a few batches when use SGD.

2.2 Overall Process

As it is known that genetic algorithm needs a population which is a set of neural network models in the weight optimization task. Thus, the chromosome is all the weight of a model including weights and bias of input layer, weights and bias of output layer. In detail, the standard network is a three-layer (input, hidden, output) network. The input layer gets 256 inputs and 16 outputs, which means there are $256 \times 16 + 16 = 4112$ variables. Besides, the hidden layer gets 16 inputs and 256 outputs, which indicates there are $16 \times 256 + 256 = 4352$ variables. Thus, in total, there are around 8500 genes insides each chromosome. As the genes (variables) is represented in float type, float-representation is applied directly rather than binary representation. The advantage of avoiding converting them to binary is that the accuracy is unrestricted and many good crossover methods, such as blend crossover, can be easily applied. Nevertheless, the search space becomes larger. The default data type of Pytorch is Float which is a 32-bit number. Thus, the DNA size is 32 bits, which is a very large search space. To reduce the space, they are set to be in range from -1 to 1.

The fitness function is the loss function of neural network. Here mean square error is applied. As all the split image is in one batch, the loss will be the total loss of all 16 sub-images. Thus, the lower the loss is the better the fitness is.

The population size is one of hyperparameter indicating the amount of model inside the population. For example, we use 1000 for this parameter. Then the initialization step will initialize one thousand models by random. After initialization, genetic algorithm will use the fitness of every models to do selection. The selection output will be the selected parents (models). After selection, crossover is applied for breeding offspring. At the end of this generation, replacement followed by mutation is applied. The algorithm will keep iterating until reaches the required number of generation. All the detail will be explained in the following subsections.

2.3 Selection and Replacement

Selection determines how to selection proper parents from the population. The general selection criteria judges based on the fitness. In this paper, two selection methods are implemented.

The first method selects parents in a proportion calculated from their fitness. As lower loss implicit higher fitness, the probability is derived from the following equation:

$$p_i = \frac{\frac{SUM}{loss_i}}{\sum_j^{ns} \frac{SUM}{loss_j}}$$

where SUM is the sum of loss of all models in the population. Higher sum over loss, higher the fitness that the model has. Then normalize the result to get the probability of each model. With this equation, the higher fitness the model is the higher probability that the model can be selected as the parent. Then the amount of parents that are selected out is the half of the population size.

The replacement corresponds to this method is also replace the population in proportion way. On the contrary, higher fitness should be less possible to be replaced as we want to keep the quality of the population. The normalization only is performed on the loss itself. Thus, this time the equation is:

$$p_i = \frac{loss_i}{\sum_j^{ns} loss_j}$$

Use the probability calculated from this equation to replace N (usually is half) population with the new offspring which has already been breed.

The second method is an elitism strategy. Rather than selected by probability, it directly selection the top N (a hyperparameter) best model as the parent. Moreover, all of the selected parent will be used for breeding offspring.

When doing replacement, the offspring will directly replace the non-selected population derived from the selection step. If the length of offspring set is less than the length of non-selected set, then some of the rest population will be remained to next generation.

2.4 Crossover

Crossover is the process to absorb features from good identities and use the feature to breed new identities. For the weight of neural network, there are two kinds of crossover.

The ordinary one is float-point crossover for single weights. In this method, only weight on corresponding position of two parents are taken into account. There are many existing methods, such as directional heuristic crossover, arithmetic crossover. In this paper, blend crossover (BLX-α) is applied. Compared with the two mentioned methods, the explored region of BLX-α is larger and flexible.

The second crossover variety is special. In most discrete optimization task, the position of genes is fixed. We can not put the exchange the first gene to second. However, for the weight of neural network, the position switch is possible. It is valid to exchange the value of one weight to another. Thus, we can treat all the weight of same layer as a gene and do crossover for them. The crossover does not change the value of single weight instead it only changes their position, which likes the crossover for discrete data.

2.5 Mutation

Mutation is the process to enhance the diversity of population. In optimization word, it is used to expand the search space. For example, the search space of gradient decent is only its neighborhood and the route it passes. But for genetic algorithm, with the help of mutation, its search space is able to reach almost the whole solution space.

Similar to crossover, mutation can be applied for single weight or all weights of a layer. For the single weight crossover, in order to make the computing efficient we can not use a loop to visit each weight and use a random number to decide whether keep it or not. As a result, some weights are selected directly to be mutated. For example, if the mutation rate is 0.08 and the population is 1000, then randomly select 80 weights to mutate. Furthermore, the mutate for the weight is only to assign a random value within range to it.

Moreover, the whole layer has a chance to mutate. This mutation will randomize all the weight in such layer. The strategy speeds up the exploration, especially at the early stage.

3 Results and Discussion

This part shows and compares the optimization result among different selection strategies of genetic algorithm, different network and different optimizer. Network to be compared is standard neural network and shared weight neural network. Two selection strategies explained in the part two will be analyzed and compared. Finally, the comparison is on the gradient-based optimizer and genetic algorithm.

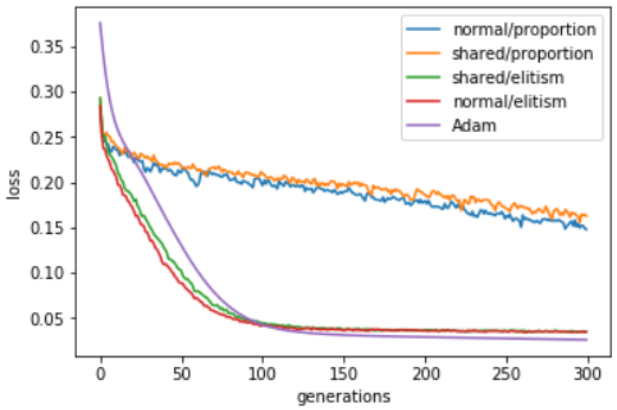

Figure 1: The loss tendency during training for 5 methods

3.1 Two Selection Strategies

In the experiment, the population size is 1000 and the generation is 300, the mutation rate is set to be 0.001. Other hyperparameters in practice not influence the result so much. As a result, the proportional selection is much worse than the elitism selection, according to **Figure 1**. Intuitively, compared with elitism selection, proportional selection uses all the population smoothly, as every population have a chance to be parent. But for elitism, only the top part population make sense while the other will be replaced finally. Thus, the proportional selection explores more "carefully". However, carefulness also has its side effect which makes the exploitation slower within same time limitation. With respect to neural network weight optimization, exploration is less important than exploitation in general cases. Because it is less likely for high dimensional function (neural network model) to have local optimum although we cannot ensure that the function is concave (Bengio, 2009). Thus, paying more attention on exploration like the proportional selection is not good for optimizing neural network.

3.2 Two Neural Network

Intuitively, shared weight network only has roughly half weight of standard network, which should make the genetic algorithm perform better. However, this intuition is a mistake because less weight leads to relatively higher loss. That is use same generation, the final loss may even worse as the **Figure 1** shown. Thus, current experiment cannot show anything that applying genetic algorithm helps shared weight network perform better.

3.3 Genetic algorithm and Adam

As it is mentioned in previous section, genetic algorithm focuses on broad (global optimum) rather than exploit deep (local optimum). However, gradient method, such as Adam, is designed to solving roughly concave function whose local optimum is not much different with the global optimum. Thus, the result (**Figure 1**) is clear. Although there is no much loss difference compared with Adam and elitism selection, the time consumption of genetic algorithm is hundred times large than Adam. It only takes around 1 second to train 300 epochs with Adam, but 300 generations for genetic algorithm takes around 5 minutes. Actually, if expand the generation to 1000 or more, the curve of genetic algorithm will become flat, which indicates it approaches convergence earlier than Adam.

3.4 Hyperparameters and Configurations

Genetic algorithm also has many hyperparameters as neural network. Besides the hyperparameter, the configuration of genetic algorithm, such as different selection method, different crossover method and different mutation method, also influences the performance a lot. Firstly, selection and crossover reflect the exploitation capability of the algorithm, which helps the algorithm dive in local optimum. Only if it remains the good solution, it can start exploit better solution based on that solution in the future generations. In other word, they also indicate how large the local search space is. The proportional method, compared with elitism, use selection that concentrate less on exploitation. Therefore, it obtains worse result for exploitation-focus problem.

On the contrary, mutation reflects the global search space or called exploration space. If a problem contains so many local optimum, the duty of mutation will be to find the new local optimum region. Thus, a good mutation strategy is the main reason that genetic algorithm is able to discover the global optimum. Hence, the parameter related to each component determines which part the algorithm should concentrate on more or less.

4 Conclusion and Future Work

Consequently, the work in this paper compares the performance among several methods. The result is that the performance of genetic algorithm is worse than classical optimization method for simple neural network. The worse performance consists of two aspects. One is the training speed, the other one is the loss. However, by researching, the reason that such result occurs is not due to the capability of genetic algorithm itself, but because such neural network optimization is not such suitable for genetic algorithm, compared with gradient-driven optimization methods.

Genetic algorithm can not only be utilized for weight optimization, but also the hyperparameter tuning for neural network. Thus, in the future, it is meaningful to apply genetic algorithm on optimization the hyperparameter. As the hyperparameter contains more discrete set, such as what optimizer to use, what number of hidden layers, what activation function is better, the effect after applying genetic algorithm may be better. Furthermore, training with genetic algorithm usually takes much more time than gradient method. Hence, in the future, it is possible to compare final loss between long time training models to see the upper performance limit of genetic algorithm.

References

Bengio, Y. (2009). Learning deep architectures for AI. Foundations and trends® in Machine Learning, 2(1), 1-127.

Gedeon, T. D. (1995). Indicators of hidden neuron functionality: the weight matrix versus neuron behaviour. In Artificial Neural Networks and Expert Systems, 1995. Proceedings., Second New Zealand International Two-Stream Conference on (pp. 26-29). IEEE.

Montana, D. J., & Davis, L. (1989). Training Feedforward Neural Networks Using Genetic Algorithms. In IJCAI (Vol. 89, pp. 762-767).

Whitley, D. (1995). Genetic algorithms and neural networks. Genetic algorithms in engineering and computer science, 3, 203-216.