THE REPORT OF BP-NEURAL NETWORK AND GA

The Project of COMP90051: Statistical and Evolutionary Learning

Tengfu Fang 568785; Di Sun 575588

Abstract

In this project, we firstly implement a simple BP-neural network that has a fixed architecture: single hidden layer of 100 nodes. Secondly, we implement a more complex BP- neural network that has arbitrary number of layers and number of nodes in each layer. Then in order to find out the optimal network architecture, we implement the Genetic Algorithm (GA). We implement all codes ourselves.

1. BP-Neural Network Implement

In order to implement feed forward neural network and back propagation algorithm, we used Java as coding language. And we used multi for-loops rather than matrix.

The structure of our project stage1 consists of two classes, one for neural network class with required parameters and the other classes for tasks such as training and predicting.

A possible innovation in our project is how to define convergence. Usually distance to describe convergence in this project is to calculate the distance between output vector and expected vector. But since the 60k instances make it hard to converge in this way, we compromise in another way to describe the convergence. That is, to find the largest value in the output vector and determine if it matches the column with the value of 1. (To do this, we assume everyone is pre-processing the given float number as expected value in each train instances into a 10-demension vector of -1 and 1. The column with the float value is to be assigned with 1 and others to be -1).

As the project stage1 requires a single hidden layer model and possibility to extend it to multi-hidden-layer one, we built a neural network class with arguments to determine its structure, including both hidden layers numbers and node number in individual hidden layer.

The class of neural network accepts three arguments which are the numbers of hidden layers, a string of certain form suggesting each hidden layer's node number (To be explained below) and the number of input to be considered by the network (for debugging). As the 36th line of NeuralNetwork.java suggests, the class accepts three arguments. For the second argument, it's a string containing each hidden layer's node number, but it only needs to contain existing hidden layer's node numbers. For example, a neural network of 1 hidden layer with 100 nodes should have that string of "100," and network of 3 hidden layer with 50,100,50 nodes should have a string of "50,100,50,". It should be noted that the "," in the final position is a must.

Basically the class includes an initializing stage with a few methods, a public training method to be called and supporting methods such as activation function. In the initializing stage, as code from 36th line to 45th line shows, the neural network will determine model structure from class initializing arguments, initialize all parameters of weights and bias with random and read instances input from certain training file. For these methods we are not running into any trouble so we will not go deep into them. I would just mention that the data required in one layer such as input, output, weights are stored in multi-dimension arrays. And a arrayList stores information for many layers, in one of which is an array as above.

The trainOnce() method is the core part of our neural network, in which we suffers a lot in debugging. Following feed forward and back propagation algorithm, the trainOnce() consist of stage of forward feeding which to reach output of a network and back propagation which tries to update weights and bias. In the first stage, we start with calculate X and Y for each hidden layer. We progressively calculate each hidden layer's output and consider to be X input for next layer. An exception is to treat input as Y output of input layer and input X of the first hidden layer. After this, we use an arrayList to store the information for multiple hidden layers. This stage is expressed in 145th to 178th line. Afterwards, 180th line to 196th line shows how to calculate X/Y only for output layer. Note that we have commented the momentum for output gradient because we found that it does not bring improvement as expected. 201th line to 210th line shows update on output weight and bias. 212th line to 247th line shows how to calculate multiple hidden layers' gradients and in this part momentum is also not activated. 249th line

to 272th line shows update on hidden-to-hidden layer weights and bias that belong to each hidden layer. There exist if/else options in the stage cause the train method has taken multiple hidden layer situation into consideration, thus it is required to deal with special case for input layer and output layer according to algorithm.

Except for other supportive methods, the prediction function is solely prepared for stage1 prediction. It already contains all I/O behaviors but it's required to change file folder before execution. The function has an argument of how many of test instances are to consider by prediction. But for most cases the number of whole test file that is 10k is suggested to use.

Since this neural network class can already deal with multiple hidden layer structure, the only change we should make for it to prepare for stage 2 is to write a new prediction method. And in our case, such method is achieved outside the network class. For the detail of the network in a multi-hidden-layer case, most is covered above, in one-hidden-layer case. What is worthy to talk about is that in order to implement the algorithm, we use an extra loop of hidden layer loop to go from input to output in the forward feeding stage and backwards in back propagation stage.

2. Genetic Algorithm Implement

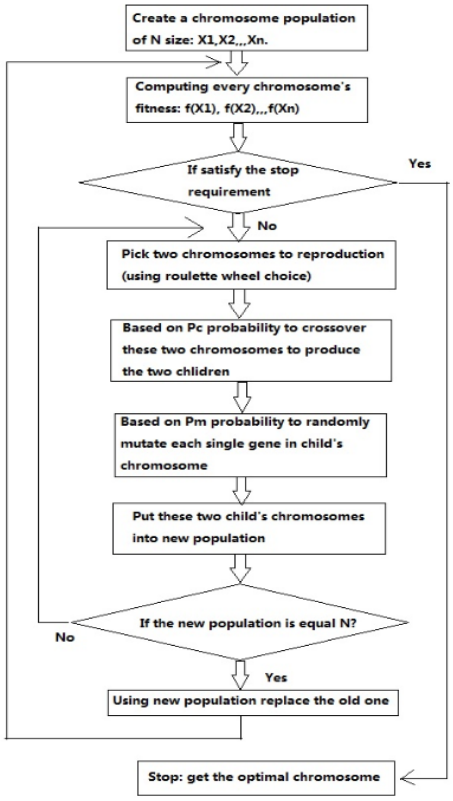
In this section, we will discuss the Genetic Algorithm (GA) part that is related with the code of the GA.java.

2.1. The Purpose of Genetic Algorithm

After implementing the complex BP- neural network that has arbitrary number of layers and number of nodes in each layer, the main issue is how we can find the optimal network architecture that has optimal performance. In our project's requirements, the number of hidden layers can be 1-10 and for each hidden layer, the number of nodes can be 1-100. Therefore, the number of all possibly network architectures is the 10th power of 100. This figure is gigantic so that we cannot simply use the brute force algorithm to check every possibly network architecture. In this situation, GA is a good strategy that can greatly reduce the computation.

2.2. How does Genetic Algorithm Work?

The main idea of GA is to simulate nature's evolution and selection. The most important feature of evolution and selection is that parents that have better genes (better genes mean better feature that can survive in the competition) have higher probability to survive and reproduction its offspring that inherits parents' better genes. To simulate this mechanism, GA can also process the data evolution and selection. For each generation reproduction, good parameters (seem as genes) that related with good performance have higher probability to be retained in the next generation. Therefore, after several rounds of reproduction, the good parameters will be more and more concentrated and common in the population. Then we could find out the optimal one. The whole process of GA is represented as the graph 2.1.



Graph 2.1: The whole flowchart of GA

2.3. Population Initialization and the Fitness Function

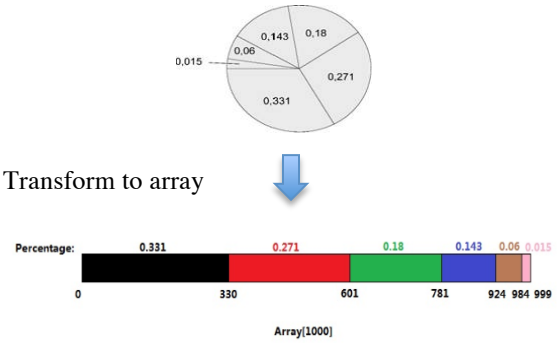
In this step, we just set the initial population has 20 chromosomes (each chromosome has 11 genes and the first gene indicates the number of hidden layer and the next 10 genes indicate the number of node for each hidden layer) since more chromosomes will spend more runtime and it is unaccepted for this project. Because of same reason, the evolution time is 20. One most important point is that if all chromosomes in the initial population are randomly picked, it cannot ensure the result of evolution has at least one chromosome that has good performance. Therefore, we add one known chromosome that has good performance into the initial population in order to ensure that there is at least one good gene in the initial population. In our case, the one known chromosome is "1,100,". (From

line 31 to line 60)

The fitness function is the BP- neural network. It uses the NeuralNetwork class and will return the performance that based on the train set. (From line 64 to line 82)

2.4. Selection Based on Roulette Wheel

We use roulette wheel choice as the selection strategy to pick chromosomes from the population as the parents. The theory of roulette wheel choice is that the chromosome that has higher fitness has higher probability to be selected. In the code, we use array to implement this mechanism. (From line 160 to line 191) One example of roulette wheel structure and array structure is shown as the graph 2.2.



Graph 2.2: Roulette Wheel Structure and Array Structure

2.5. Crossover and Mutation

When two chromosomes are picked as parents, they will reproduce two children using crossover mechanism or not (it depends on the probability of crossover: Pc). The crossover is that at one randomly picked point, two chromosomes exchange their second part. Moreover, in our implement, the first gene never attends the crossover mechanism since it represents the number of hidden layer. (From line 87 to line 133)
One example can be:
The parents' chromosomes:
Chromosome A: {4,50,60,70,80,0,0,0,0,0,0}
Chromosome B: {8,10,20,30,10,50,60,70,80,0,0}
After crossover at point 6:
Chromosome A': {7,50,60,70,80,0,60,70,80,0,0}
Chromosome B': {5,10,20,30,10,50,0,0,0,0,0}

We can see that the first gene (represents the number of hidden layer) is also changed to meet the new contents. The updataLayerNum function is designed for this process. (From line 197 to line 226)
After crossover, mutation mechanism will start. The purpose of mutation is to ensure the GA will get the global optimal result, not the local optimal result. In other words, mutation can avoid the loss of genetic diversity. In our implement, each gene, except the first gene, will do mutation separately based on the probability of mutation (Pm). The mutation step is plus 4 and after plus 4, if the value of one gene is beyond 100, the value will be randomly chosen from 1 to 100. After finishing mutation, do same process about update the first gene. (From line 136 to line 156)

2.6. Update the Population

After finishing reproduction, the children population should be seen as the new parent population to process the next reproduction. The code that implements this is from line 232 to line 240.

2.7. The Optimal Structure

After finishing running GA, we find out that the structure has 2 hidden layers has better performance than the structure only has 1 hidden layer although the superiority is not obvious. We also find out that the structure that has 51,68 for each hidden layer has the best performance after the first round training (45000 samples per round). So we choose this structure as our optimal structure. Moreover, based on our experiments, almost all structures that have more than 3 hidden layers have bad performance.

3. Difficulties and Lessons

We see how difficult it can be to implement a math algorithm in developing this project. The difficulty is never comparable to an independent computing program, whose difficulty is purely a coding-level one. Considering a scientific computing project like this, another problem can be that sometimes we had errors in the algorithm itself but never realize. For example in our project, firstly we had an opposite sign in one step of calculating (confusing "+" and "-" in updating hidden gradient) so always see an unsatisfied training result. Besides, even the model is correct, we meet problem of parameter setting. In our case, the single hidden layer version returns satisfactory result but we witness terrible results from a 2 hidden layer network, with exactly the same parameters such as learning rate and random range. When a network returns unsatisfactory results but errors could come from various levels, or even the model is not "incorrect" one (for example the parameter setting issue mentioned above), it will be extremely difficult to correct it. A good lesson, also by which we successfully corrected our model, is to first make sure each stage is totally correct, avoiding the possibility of multi-errors happening at the same time. Advancing step by step, we should find where problem lies and correct it in a more efficient way.