How Genetic Algorithm Can Improve Model Performance When Dataset Is Imbalanced

Yuzhou Chen

Research School of Computer Science,
The Australian National University, Acton, Canberra, ACT 0200 Australia

U6363358@anu.edu.au

Abstract. Building and training a model with limited size when the given dataset is small and highly imbalanced can be challenging, this paper investigates multiple approaches that help resolve these difficulties, including mini-batch gradient descent for backpropagation and genetic algorithm for weights training. How these methods improve model performance is recorded. In particular, a comparison between the above mentioned two methods is done to give more information on how genetic algorithms can improve model performance when dealing with imbalanced data.

1 Introduction

The relationship between human eyes movements and human cognitive processes has always been intriguing. Eyes movements, in particular eye gaze patterns, can often provide valuable information about decision making process of human brains [1]. Other than for research purposes however, there exist real world applications of eye gaze pattern recognition, for instance a game-like interactive scenario [2], where runtime efficiency is often vital due to the nature of the task, as any decision making process must finish within a split of a second. Due to this efficiency need, it is necessary to limit the size of the neural network, a model with many hidden layers and a couple hundreds of hidden units is obviously not practical.

In this paper, a light weight, feed-forward neural network that is able to recognize whether a person is scanning facial images or text documents based on their eye gaze patterns is constructed. This trained model is intended to run fast during testing, has small size, and be able to achieve as good performance as possible. Given that the data at hand is small and highly imbalanced, plus the size constraint on the model, to increase the model performance in terms of accuracy and recalls, various techniques are employed. Specifically, I examined two training methods, one is the more popular mini-batch gradient descent for backpropagation, which has been proven useful handling imbalanced dataset for its effectiveness of escaping local optimal, and the other one is a genetic algorithm for weights training. The reasons why genetic algorithm is chosen to perform the task are first, it is famous for its ability to approximate global optimal, and second, given the relatively small size constraint on my model, convergence time should not be of concerned in this particular case even if hundreds of iterations are needed. The performance of both of these methods are evaluated and compared. A similar model has already been constructed in [1], the resulted model in this paper in general provides a better performance.

2 Method

The dataset on which the model is trained and tested is from [1], which was collected through recording 10 participants eye gaze patterns when showing them facial images and text documents [1]. The features of each datapoint represent the vertical distances between the first 5 fixation points of eye gaze, each value is an integer, and the target value represents whether a facial image or a text document is displayed in front of the participant [1]. It is not necessary in this case to normalize input values because each of them represents a distance, no feature will heavily outweigh other features because they all have the same range and same meaning, and normalizing them between 0 and 1 could potentially decrease the precision of each value as all of them are in the range 0 to a couple hundreds. There are two major problems with this data, the first one being it has a relatively small size, with total only 323 rows recorded, the second one being it is highly imbalanced, with 276 data points belonging to one class, and 45 belonging to the other.

For simplicity, the model is a simple three layers feed-forward fully connected neural network, with 4 input neurons as there are 4 features, a number of hidden neurons, and 2 output neurons indicating the number of target classes. For training using backpropagation, mini-batch gradient descent is employed. Cross entropy is used as the loss function, as it is a common practice for binary classification, and sigmoid function is used as the activation function. As for the genetic algorithm used for weights training, a combination of tournament selection and elitism is used in the selection process, and one-point crossover is used as the crossover method. Details of both approaches are introduced below.

2.1 Mini-Batch Gradient Descent (MBGD)

There are two major disadvantages of using the traditional batch gradient descent (BGD) method for backpropagation. It calculates gradients of the loss function based on the entire dataset, one is that while it will certainly converge to a minimum, that minimum may not be the best one [3], in other words BGD has difficulty getting out of local minimums due to its lack of flexibility. The other disadvantage is that because the given data is highly imbalanced, the model trained on it would naturally lean towards the majority class when classifying, and since BGD is guaranteed to converge, it is highly likely whichever minimum it converges to would not lead the weights of the model to a place where the model is less bias than it should be. This has been proven by training and testing on the dataset using the same model, training with BGD always leads to all class 0 (the minority class) datapoints be classified as class 1 while testing.

As for the exact opposite of BGD, stochastic gradient descent (SGD), where gradients are computed based on one sample datapoint at a time, has been proven in practice not as suitable as MBGD for this particular scenario. The performance fluctuates more greatly than when using MBGD, which is likely due to the fact that SGD induces too much randomness and thus has a higher probability of keep overshooting and never converge [3], especially when the size of the dataset is small.

In general, mini-batch gradient descent is a balance between the two, it introduces just enough randomness to the gradient descent process to help the model jump out of local minimums, and potentially land on somewhere closer to a place where the model would be in favor of the minority class, since naturally it would lean toward the majority class due to the flaws of this dataset.

2.2 Genetic Algorithm

A variation of genetic algorithm is used for model weights training, by selecting the weights and bias combination of the model that has the highest fitness value. Here, a chromosome is defined as a combination of weights and bias of each neuron, and a gene is defined as a component of a weights vector. Fig.1 demonstrates what a chromosome looks like for a model with N hidden neurons, 4 input neurons and 2 output neurons. A gene is a weight or bias, for instance w_{N1} is a gene.

Hidden1_weights = [w_{11}, w_{12}, w_{13}, w_{14}, $bias_1$]
Hidden2_weights = [w_{21}, w_{22}, w_{23}, w_{24}, $bias_2$]
...
HiddenN_weights = [w_{N1}, w_{N2}, w_{N3}, w_{N4}, $bias_N$]
Out1_weights = [w_o_{11}, w_o_{12}, … w_o_{1N}, $bias\_o_1$]
Out2_weights = [w_o_{21}, w_o_{22}, … w_o_{2N}, $bias\_o_2$]

Fig.1 Chromosome

2.2.1 Initial Population Generation

Initially a pool of chromosomes is randomly generated, with weights and biases of each hidden units capped between 0.5 and -0.5, and between 3 and -3 respectively. And weights and biases of each output units are capped between 3 and -3, and between 1.5 and -1.5 respectively. These intervals are obtained through empirical analysis of multiple trained models with relatively good performance. Although a random generation can increase diversity of the initial population, setting an interval on each gene helps limit the initial search space and lets each chromosome be potentially closer to the global optimal, therefore leads to a faster convergence [4]. Note that it is still possible for these values to be out of the initial intervals after mutation.

2.2.2 Selection

A combination of tournament selection and elitism is used in the selection process for this genetic algorithm. At each iteration, half of the current population are selected to the next generation along with their offspring. Each individual chromosome of this selected half is the winner of their respective tournament. Each tournament consists of a small number of randomly sampled participants from the current population, and the winner is the one with the highest fitness value. After the winner is chosen in each tournament, it is not removed from the current pool, so it is possible to sample it again. If the current population has a size N, then in total N / 2 tournaments are held and the winners will survive to the next generation with their offspring. Hence the size of the next generation is the same as the current one.

The reason why this approach is adopted is because it helps bring balance between quality and diversity to the population. It ensures the former by always choosing the best candidate from each tournament and passing them to the next generation, and the latter by making sure every chromosome has a greater than 0 probability of being chosen for crossover and survives to the next generation regardless of its fitness value, if the tournament size is small relative to the population size. For example, for a population of size 100, and tournament size 2, it is easy to see that the probability of a chromosome being chosen to participate in at least one tournament is 0.64, regardless of its performance. Which means even if it has a low fitness value, as long as it beats the other competitor in its tournament, it will be chosen for crossover

and survives this selection round. This way, there is always a little room for underperformers from the last generation in the current pool, hence diversity increases. Although it is important to make sure that the tournament size is within a reasonable range, for instance a tournament size of 20 when the population is of size 100 will pretty much guarantee only the best chromosomes survive, and a tournament size of 1 is not much different than random selection.

2.2.3 Crossover

A form of onepoint crossover operator is used in this genetic algorithm. Two parents are needed, and two offspring are generated as a result. The parents are the selected chromosomes from the selection process, each one of them will perform crossover with another exactly once. For example, if the total population has size N, then N / 2 chromosomes will be chosen for crossover, and there will be in total N / 4 couples among them, thus crossover is performed N / 4 times, and since each crossover will produce two offspring, then in total N / 2 offspring are produced. These N / 2 offspring, along with their parents, form the next generation of chromosomes.

Onepoint crossover is defined as followed. First, a random number p in interval (1, number of hidden units] is generated. Then, for hidden units in the first parent indexed between 1 and p – 1 (inclusive), weights (from input units) and biases of them, along with the weights between them and the output units, are transferred to one child, then do the same for the hidden units indexed between p and N (the number of hidden units) of the second parent. The other offspring would be generated the other way around, using hidden units indexed between 1 and p – 1 from the second parent and hidden units between p and N from the first parent. Fig.2 illustrates how crossover is performed when there are 4 input units, 2 hidden units and 2 output units. p is 2. Dot lines represent the weights in parent 1 and solid lines represent the weights in parent 2.

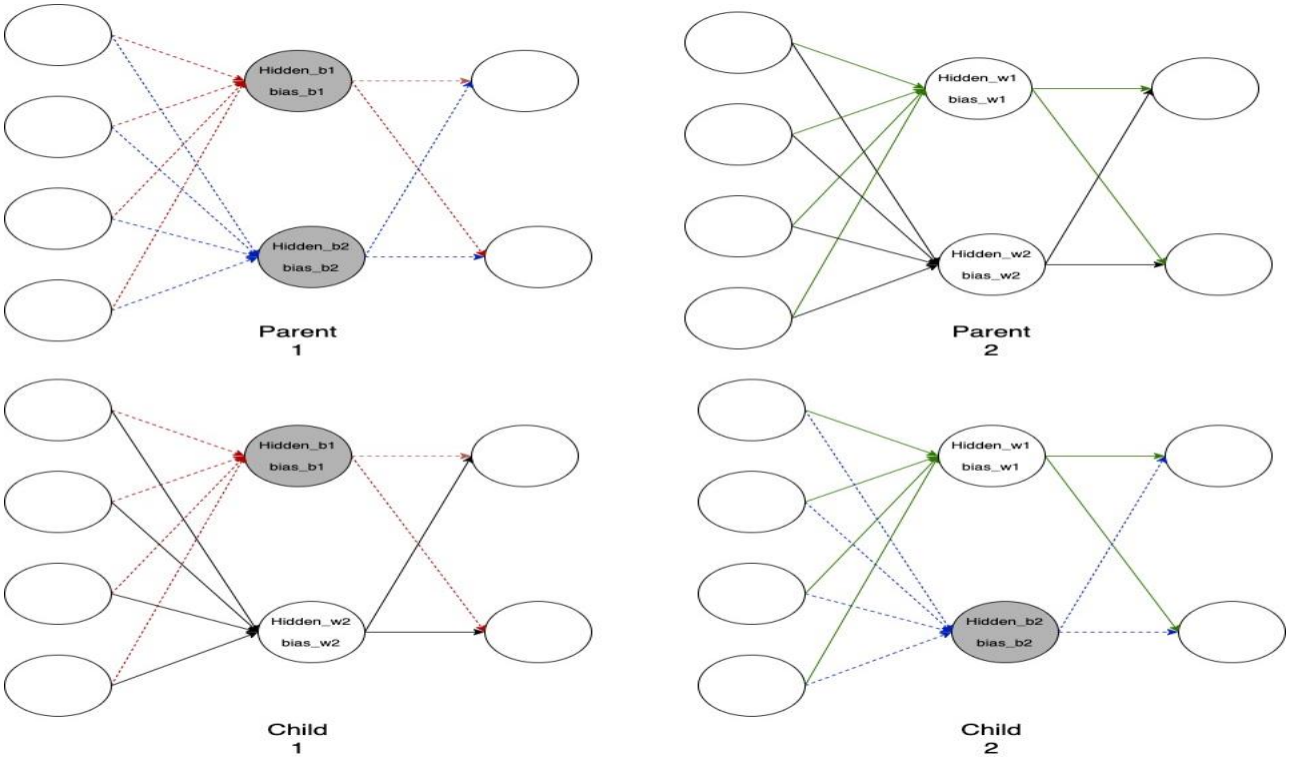

Fig.2 Crossover

Note that no bias of any output unit is inherited, they are randomly generated in the two children.

The reason why this method rather than for instance linear recombination is employed is because, essentially, the weights vector (including bias) of each hidden unit represents the information learnt by this unit from the input features, and weights from it to the output units represent how this hidden unit perceives the relation between the information learnt from the input features and the final outputs. These along with the biases of output units determine how well a model performs. This approach allows children to take the best parts from both parents, i.e. it is possible that a child can take the parts where each parent is "right" about the meaning of input features and their relations with the final output, and then combine them together to be potentially stronger. As for why output biases are not inherited is because it helps creates more diversity in the offspring.

2.2.4 Mutation

Mutation is done by adding a random value in the range [-*mutation_range*, +*mutation_range*] with a certain probability, to each gene in a chromosome. *mutation_range* is a given parameter. Gene is defined as the vector component of weight

vectors that consist a chromosome. Fig.3 shows a mutated version of the chromosome in Fig.1, where r_i is a random value in some range.

Hidden1_weights = [w_{11}, w_{12} + $\mathbf{r_1}$, w_{13}, w_{14}, $bias_1$]
Hidden2_weights = [w_{21}, w_{22}, w_{23}, w_{24}, $bias_2$ + $\mathbf{r_2}$]
…
HiddenN_weights = [w_{N1}, w_{N2}, w_{N3}, w_{N4}, $bias_N$]
Out1_weights = [w_o_{11} + $\mathbf{r_4}$, w_o_{12} + $\mathbf{r_5}$, … w_o_{1N}, $bias\_o_1$]
Out2_weights = [w_o_{21}, w_o_{22} + $\mathbf{r_6}$, … w_o_{2N}, $bias\_o_2$ + $\mathbf{r_7}$]

Fig.3 Mutation

2.2.5 Fitness Function

Fitness value of a chromosome is defined as the product of accuracy and recalls for both classes when testing on the training set. This function is preferable over others such as product of recalls or accuracy alone because it takes all three measures into account, it helps balance between recalls and accuracy, hence reducing bias and overfitting given that the data is incredibly imbalanced. Also, it is better than using the loss function because it does not matter whether the fitness function is differentiable or not as no backpropagation is performed here, and using loss function makes the model lean towards the majority class and resulted in overfitting.

This genetic algorithm implements a reinitialization mechanism. If the best performing individual in the current population hasn't become better after a certain number of iterations, the whole population will be deleted and reinitialized, while preserving the best outcome so far. It is a variation of the reinitialization algorithm proposed in [5], in which the best outcome seen so far is put back into the new population after the reinitialization and can be used for crossover in future iterations, while here it is only remembered but not being put back in, because I want to increase diversity of the population. In short, this mechanism helps getting out of local minimums and expands the search space, it's been proven quite effective in this scenario.

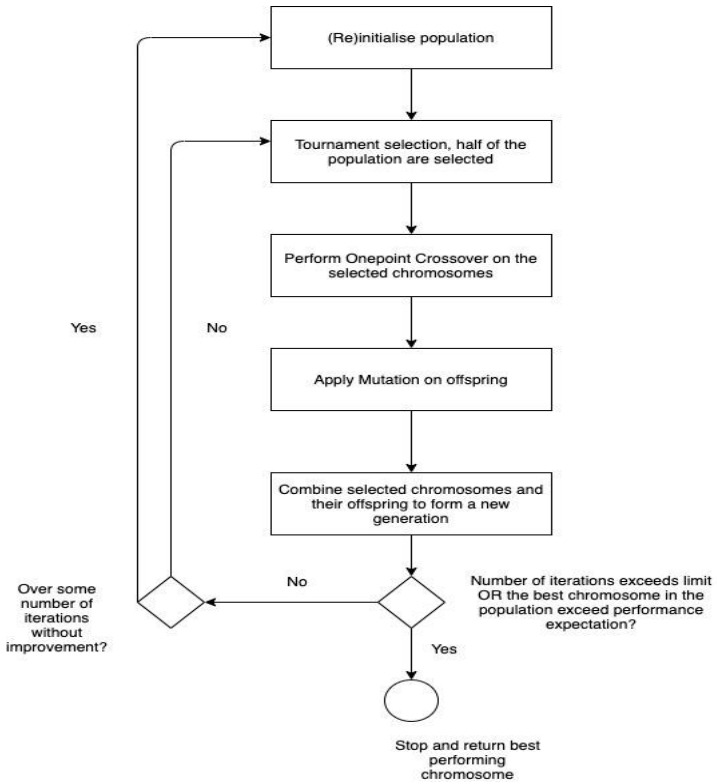

Fig.4 Flowchart

3 Results and Evaluation

In order to make comparisons between models' performance easier, I set the number of hidden units to be 8 in all cases, because after testing with different numbers, any number too large may result in overfitting when performing backpropagation due to the limited size of the dataset, and numbers too low will decreases its performance, any number between 8 and 10 can be viewed as a balance point between overfitting and underfitting.

Accuracy %	93.46	89.72	93.46	93.77
Recall for 0 %	80	31.11	68.89	60
Recall for 1 %	95.65	99.28	97.46	99.28

Table 1. SGD

Accuracy %	93.15	91.90	95.33	95.95
Recall for 0 %	91.11	55.56	84.44	84.44
Recall for 1 %	93.48	97.83	97.1	97.83

Table 2. MBGD

Table 1 and 2 show how overall accuracy and recalls vary for using SGD and MBGD after performing 5-fold cross validation 4 times respectively. For BGD, recall for class 0 are all 0s. Each training process performs 500 epochs. In all cases, learning rate is 0.01, size of each mini batch for MBGD is 5, cross entropy is used as the loss function and sigmoid function is used as the activation function. Although SGD does generate high enough accuracy, its recalls for minority class 0 fluctuates greatly, much higher than those generated by using MBGD. These results are expected as above mentioned, SGD induces too much randomness thus has a much fluctuating performance, and BGD lacks flexibility to escape local minimums and reduce bias inherited from the data. MBGD strikes a balance between the two and generate the best results.

There are a number of adjustable parameters for the genetic algorithm: *initial_pool_size*, the population size, the higher the bigger the search space thus slower convergence. *max_iteration*, the maximum number of iterations allowed before stopping the algorithm, the higher the bigger the search space. *threshold*, the expected performance, once a chromosome with fitness value exceeding it has been found the algorithm stops, it should be between 0 and 1, and again the higher the bigger the search space. *mutation_prob*, mutation probability, should be between 0 and 1, a higher means more diversity in the population. *mutation_range*, the random value added during the mutation process will be in [*-mutation_range*, *+mutation_range*], the higher the more diverse the population is. *tournament_size*, size of each tournament, a higher value relative to the population size means less diversity. And *restart*, which determines the maximum number of iterations without improvement in the current population before the whole population being reinitialized.

	MBGD	GA	MBGD	GA	MBGD	GA	MBGD	GA
Accuracy %	96.26	97.82	96.88	96.57	97.82	96.26	95.95	97.2
Recalls for 0 %	84.44	91.11	86.67	91.99	93.33	93.33	82.22	93.33
Recalls for 1 %	98.19	98.81	98.55	97.46	98.55	96.74	98.19	97.83

Table 3. Results

Table 3 shows the experimental results of performing 4 times 5-fold cross validation, using the genetic algorithm and mini batch gradient descent for backpropagation to train weights. The hyperparameters for using MBGD for backpropagation are identical as the above experiment: learning rate is 0.01, number of epochs run is 500, each mini batch is of size 5, cross entropy is used as the loss function and sigmoid function is used as the activation function. The parameters for the genetic algorithm is as followed: *initial_pool_size* = 200; *max_iteration* = 120; *threhold* = 0.95; *mutation_prob* = 0.3; *mutation_range* = 0.3; *tournament_size* = 5, hence the probability of any chromosome being chosen to participate in at least one tournament is around 93%, leaving a small yet greater than 0 probability for chromosomes with low fitness value to survive; *restart* = 20. All these values are determined empirically, they are able to produce the best outcomes compared to other reasonable combinations of parameters values, as there is no general rule of thumb to decide the values of these parameters since the performance of genetic algorithms highly depends on the practical situation at hand.

The average accuracy for training using backpropagation is 96.73%, the average recall for class 0 is 86.67%, and the average recall for class 1 is 98.37%. While the average accuracy for training using genetic algorithm is 96.96%, the average recall for class 0 is 92.44%, and average recall for class 1 is 97.71%. Although there is not much difference between recalls for class 1 (the majority class) and accuracy in both cases, the genetic algorithm performs better in terms of recalls for the minority class, proving that it is less bias when the give data is highly skewed than the traditional backpropagation algorithm. This is indeed expected because the optimization goal of the genetic algorithm is the product of all three measures, it would try to maximize all of them during the search, while backpropagation training is intended to minimize the loss value, it will by its nature favors the majority class.

Comparing with the results published in [1], which reports a 20 percent CLE (80 percent accuracy) when classifying on the same dataset, both training methods investigated in this paper can produce a better outcome in terms of accuracy. But this does not conclusively prove my models are better because recalls are not reported in [1].

4 Conclusion

The genetic algorithm proposed and investigated in this paper proves to outperform the traditional backpropagation training algorithm in this scenario in terms of accuracy and recalls. It is proven to be able to provide better outcome when the given data is imbalanced. Although it is worth noting that convergence time is not of concern here due to the size constraint put on the model as well as the small size of the data, as model grows larger and more data is being used the training time should significantly increase when using genetic algorithm.

For future work, RNN models can be investigated to see if they provide better results, because there exists a chronological order of the 4 features in the dataset, it is possible there is a link between each feature, for instance the vertical distance between the first fixation point and the second one may have an impact on that between the 4th and the 5th. Much like words in a fixed-length sentence, the model structures used for language modelling, such as LSTM [6], could potentially be applicable in this scenario to unveil these types of relations. Furthermore, it would be interesting to see how well would an RNN model perform if its weights are trained using genetic algorithm.

5 Reference

[1] D. Zhu, B. S. U. Mendis, T. Gedeon, A. Asthana and R. Goecke, A Hybrid Fuzzy Approach for Human Eye Gaze Pattern Recognition, Canberra: International Conference on Neural Information Processing, 2008.

[2] T. D. Gedeon, D. Zhu and B. S. U. Mendis, Eye Gaze Assistance for a Game-like, Interactive Task. International Journal of Computer Games Technology, 2008.

[3] S. Ruder, An overview of gradient descent optimization algorithms∗, Dublin, 2016.

[4] H. Maaranen, K. Miettinen and A. Penttinen, On initial populations of a genetic algorithm for continuous optimization problems, 2006.

[5] D. E. Goldberg, Sizing populations for serial and parallel genetic algorithms, Proceedings of the 3rd International Conference on Genetic Algorithms, 1989.

[6] M. Sundermeyer, R. Schuluter and H. Ney, LSTM Neural Networks for Language Modeling, Aachen: 13th Annual Conference of the International Speech Communication Association, 2012.