Applying Genetic Algorithm for Initialization in Backpropagation Neural Network Model and Pruning with Distinctiveness Methodology

Yifei Tao
Research School of Computer Science, Australian National University
u7078518@anu.edu.au

Abstract. In this paper we used the dataset of sparse natural person photos [1] which the model training results can be greater than 75% (Caldwell 2021). The model aims to help judging whether the faces in different photos are the same person. This dataset is characterised by a small number of samples and a high number of sample features. In this paper, a fully connected feedforward neural network model using backpropagation is designed based on the above dataset, and the initial weights are generated by applying genetic algorithm. The neural network model is then pruned by using the distinctiveness methodology [2]. According to Gedeon and Harris (1991, pp. 25-34) for the distinctiveness methodology, we will prune the neural network model by calculating the vector angles between the pattern vectors of the hidden layer neurons and observe the performance of vector angles during the operations and compare the model performance before and after applying the pruning method. We can conclude from the experiment that the utilization of genetic algorithm and pruning technique with distinctiveness methodology will help a neuron network model to have good performance and faster training speed.

Keywords: Neuron Network · Backpropagation · Distinctiveness · Pruning · Genetic Algorithm

1 Introduction

1.1 Goal of the Paper

This paper aims to train a neuron network model for sparse natural person photos recognition by utilizing genetic algorithm, backpropagation algorithm, and the pruning technique with distinctiveness methodology. The paper will firstly state the design of the model and explain the settings in the model and the designed pruning method in our experiment. Then we will design specific experiments to observe changes in the performance of the model before and after using the pruning method with distinctiveness methodology, including loss, accuracy, running time and number of hidden layer neurons, to verify the stability of the model and the optimization results of the techniques we use.

1.2 Technique Background

When training data using simple neural network models, it has been a challenge to determine the initial state of weights and the appropriate number of hidden layer neurons. A good initialization of weight matrix will help the model to perform more stably. By using genetic algorithm, we can reduce the dependent of initial value in optimizing process [6]. And a proper number of hidden neurons can help give a better performance. In addition, as the backpropagation algorithm has the problem of running a large amount of computation, if the number of hidden layer neurons is not optimised, many computational resources will be wasted during the training process. In contrast, according to the results of the technical paper [2] we referred, based on the effect presented by the hidden layer neurons in the training, we can locate some redundant neurons and perform optimization operations such as merging and deleting, to optimize the number of hidden layer neurons without retraining the model and obtain a more concise and effective neural network model (Gedeon and Harris 1991). Distinctiveness methodology is used to determine the significance of a neuron by calculating the pattern vectors' angle of the hidden neurons, which is used as a basis for pruning the neuron (Gedeon and Harris 1991). Distinctiveness means that the magnitude of the vector angle between the output vectors of the hidden layer neurons is calculated to determine whether the hidden layer neurons perform similarly or oppositely, i.e., whether they have unique functions, so that the hidden layer neurons can be pruned accordingly.

1.3 Dataset Introduction

The dataset used in this paper is derived from several old historical photographs [1], which have in common the small number of photographs of each individual. For this dataset it was decided to mark the features of the dataset with facial feature markers (FFMs) and metrics (Caldwell 2021), and two lists of the dataset were generated using 36 images in groups of three, with the three images in each group being compared two by two. The expected results were that in each group picture A and picture B were matched, picture A and picture C were not matched, and picture B and picture C were

not matched. The first dataset generated is called the 'FFMs', which are individual facial feature points transformed into two-dimensional coordinates. Each row of the dataset is a two-by-two combination of all 12 sets of images, for a total of 36 rows. And each of its columns is the x- or y-coordinate of the individual facial feature points, and the last column is the expected match result. Since there are 14 facial feature markers on each image, the dataset has a total of 14×2×2 = 56 feature columns. The second dataset is called 'distances', similarly, each row of it is a two-by-two combination of all 12 sets of images, for a total of 36 rows. Each column is a two-by-two combination of 14 facial feature markers, and the Euclidean distance between every two marker points of each photo is calculated (Caldwell 2021). Because of the two-by-two combination, the dataset for this sheet has a total of $C_{14}^{2}\times 2 = 182$ feature columns.

Considering that facial markers are fixed points on the coordinate axes, i.e., absolute data, while distance is a comparison between every two points, which is relative data, in terms of model's generalization ability, if the input data is inconsistent in terms of portrait size or facial angle, we can get more stable and accurate results by training the model using relative positions such as 'distance', while using absolute positions such as 'FFMs' may lead to more severe overfitting. We tested this conjecture by designing a neural network model of equal complexity using Pytorch using the two dataset lists above, as described in the 'Method-Data Selection with Pytorch' section of this paper. We finally chose dataset 'distances' for the subsequent experiments on pruning the neural network model.

In general, the dataset used in this experiment was characterised by a small amount of data overall and a large number of features over the number of samples.

2 Method

2.1 Data Selection and Comparison between Dataset 'distances' and 'FFMs'

In the Introduction section we mentioned the comparison and selection of two datasets, 'FFMs' and 'distances', which we will train by Pytorch and select one of the datasets based on the results.

The first one is the training of the dataset 'distances'. We first process the input training dataset. The dataset consists of 184 columns, we remove the first index column and use the last column as the target outcome 'y'. The remaining 182 columns are the data for each of the two comparing images, each having 91 columns. As each row of data is used to compare the differences between picture 1 and picture 2, and thus conclude whether the two pictures are images of the same person, we should show the difference in distance between the same two facial feature markers for picture 1-1 and picture 1-2 and use these differences as input features. Taking the first row of the dataset 'distances' as an example, which is shown in Table 1: this row records 91 distance calculations for each of picture 1-1 and picture 1-2, and the feature '101' represents the distance between the two feature points 'rtex' and 'rten' of picture 1-1 and the feature '201' represents the distance between the two feature points 'rtex' and 'rten' of picture 1-2. If '101' and '201' are used directly as the input data for training, the matching result will directly rely on the values of '101' and '201' respectively. If we take the difference between '101' and '201', temporarily called 'diff1', as the input data for training, our conclusion of "whether or not to match" will be related to the difference between the distances of two images, picture 1-1 and picture 1-2, when calculating the same distance.

Table 1. Sample of dataset 'distances'.

	101	201	outputs
1-1 & 1-2	42.1900	23.0868	1

In the actual image recognition and comparison of each image, what we are really interested in is the difference between the two images for the same feature points and the same distance. Therefore, we took the 182 columns of the dataset 'distances', corresponding to the two samples above, and made the difference of two pictures in each row, to obtain the final input data, i.e., a 36×91-dimensional dataset. Then we used the scale function of Scikit-learn's pre-processing to normalise the input data. In the end, we randomly selected about 2/3 of the dataset as the training set and the rest as the test set.

We used the sequential container in Pytorch to set up the required neural network model. Due to the small number of samples in the dataset, we use a neural network consisting of only an input layer, a single hidden layer, and an output layer to prevent overfitting from an overly complex neural network model, setting the number of neurons in the input layer to 91 as the number of columns in the dataset, the number of neurons in the hidden layer to 50 and the number of neurons in the output layer to 2. We used Sigmoid as the activate function, set the learning rate to 0.01, set the loss function to cross entropy, set the optimization method to stochastic gradient descent, and set the algorithm to back propagate. After several trials, we noticed that the accuracy of this dataset was stable at around 80% on both the training and test sets. The training loss, training accuracy and testing accuracy of dataset 'distances' in this model are shown in Figure 1.

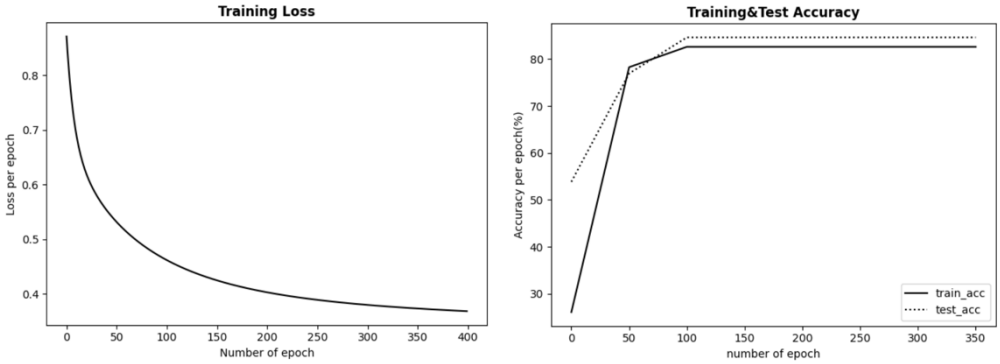

Fig. 1. The loss function image and the accuracy image of the model.

We similarly delete the first index column for the dataset list 'FFMs', use the last column as the target outcome 'y', and use the remaining 56 columns as the feature columns for the samples. We trained this dataset using a neural network of the same size as above. Since the number of neurons in the input layer is 56, we set the number of neurons in the input layer to 56 and the number of neurons in the hidden layer to 30. The remaining parameters refer to the neural network model used in 'distances'. After several trials, we noticed that the accuracy of this dataset was stable at around 80% on the training set, but the accuracy on the test set had a wide range of fluctuation and eventually only stabilised at around 55%. The training loss, training accuracy and testing accuracy of dataset 'FFMs' in this model are shown in Figure 2.

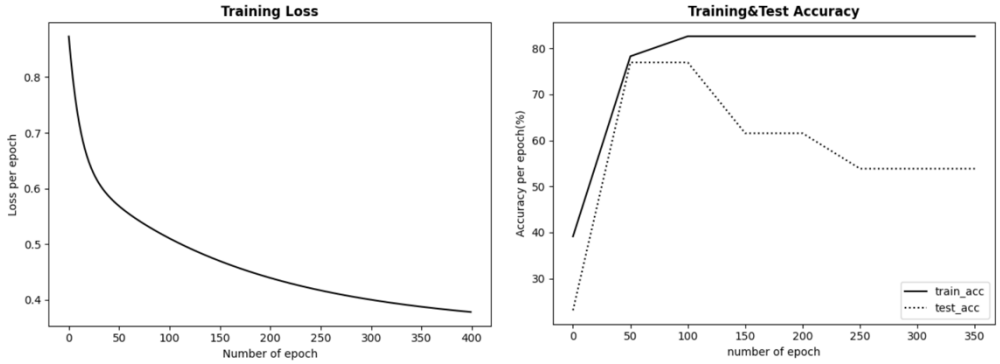

Fig. 2. The loss function image and accuracy image of the model.

Comparing the results of the two datasets in the model, we found that the optimisation of the loss function of the dataset 'distances' was faster, and the curve was more reasonable, and the accuracy of the training set of both was almost equal, but according to the performance of the accuracy of the test sets, the dataset 'FFMs' showed obvious overfitting results. Based on the performance of both, we finally decided to use the dataset 'distances' for subsequent experiments.

2.2 Design of Genetic Algorithm

First, we used genetic algorithm and backpropagation to obtain a simple three-layer fully connected feed-forward neural network model which has been fully trained. The genetic algorithm has stronger adaptiveness, allowing the algorithm to avoid falling into local minima by using selection, crossing and mutation. And it also has better global search performance, allowing all the weights to be searched and tried in parallel to get the best choice [7]. Our genetic algorithm has the following main functions:

- fitness function: We took the inverse of the loss value as the fitness function. Below is the detailed equation. In this equation, 'x' stands for the x^{th} input sample and 'X' stands for the number of inputs. 'i' stands for the i^{th} neuron's output and 'N' stands for the number of neurons. 'y' stands for the target value, and '$\hat{y}$' stands for the actual output value.

$$fitness = \frac{1}{\sum_{x=1}^{X}\sum_{i=1}^{N}|y_x^i - \hat{y}_x^i|} \quad (1)$$

- selection function: We used the Roulette Wheel method for selection. The probability in the wheel is proportional to the fitness value. And we ensure that the number of individuals in the new population and the original population are equal.
- crossover function: We set a crossover rate, and by crossover we can get more different individuals from current

existed individuals for selection.

- mutation function: We set a probability of mutation as well as the upper bound and lower bound of mutation values. With the help of crossover and mutation, we can get more possibility to explore the better choice.
- Elitism: The individual with better fitness result should be selected for inheritance to the next generation. Therefore, we use the Elitism method to copy the best chromosome to the new population before doing the crossover and mutation during each generation [6].

According to the above functions, we introduced a series of hyper-parameters for adjusting the performance of the genetic algorithm and generated the initial weight matrices, including the weight matrix between input layer and hidden layer, and the weight matrix between hidden layer and output layer. And we applied the result of those generation into our neuron network model. The flow chart of the working steps for the genetic algorithms in our model is shown in Appendix A.

2.3 Design of Backpropagation, Pruning and Experiments

After that, we began the work of distinctiveness calculation and pruning. The distinctiveness methodology is based on the technique paper [2] about pruning a neural network with backpropagation, where there are some redundant units in the hidden layer that are not functioning. The main disadvantage of using backpropagation is that the training is overall slower. Distinctiveness refers to determining whether the hidden layer neurons act similarly or oppositely to each other, which means whether they have unique functions, by calculating the magnitude of the vector angle between the output vectors of the hidden layer neurons in two-by-two, so that the hidden layer neurons could be pruned accordingly [2]. According to this methodology, we designed the pruning steps as follows: firstly, we obtained the pattern vector of all the hidden layer neurons in the model, then we calculated the vector angle size between the pattern vectors two by two, and finally we filtered the neurons corresponding to the target vector angle that needed to be pruned and performed the merge or delete operation [5].

During the experiments, we noticed that the distribution of vector angle size results computed from the dataset was mostly concentrated in the interval of less than 5°. Based on this feature, we designed a certain method for pruning the neural network in this paper. Our method sorted the results of vector angles less than 5° in ascending order, adding a percentage factor, filtering the vector angles less than 5° by the percentage, and merging the corresponding hidden neurons. The reason for designing this method is to design a pruning method with larger pruning magnitude and smaller selected angles based on the referred technique paper [2]. In specific experiments, we would use the Controlled Variables method with the same rest parameter configuration for observing and comparing the performance of the models before and after pruning.

Our pre-defined experimental comparison terms include the value of loss, the accuracy of the model on the test set, the accuracy on the training set, the degree of loss convergence, the number of neurons in the hidden layer, and the running time with the same number of epochs. We presupposed the following experimental results. Based on the theory of distinctiveness of hidden layer neurons, we concluded that the pruned neural network should have a shorter running time and faster convergence of the loss function, a smaller number of hidden layer neurons, an essentially unchanged degree of lost convergence and accuracy on training set compared to the neural network before pruning. And as for the comparison of the accuracy on the training and test sets, we would observe the actual experimental data before making a judgment.

2.4 Hyper-parameter Setting and Special Design in the Model

As Pytorch is a highly encapsulated package from which it is not intuitive to extract the performance of neurons and weights, instead of using the highly encapsulated Pytorch, we additionally trained a backpropagation neural network model using the dataset 'distances' during our experiment. The pre-processing of the dataset 'distances' is described in the chapter 'Method- Data Selection with Pytorch' in this paper, and we ended up using a 36×91-dimensional dataset with the same normalisation as described above, and we separate 60% of dataset as training set, the remaining data as test set.

The initialisation of the weight matrix can always have a significant impact on the effect of the model. Therefore, we used the genetic algorithm to initialise the weight matrix. In our genetic algorithm, we set a series of hyper-parameters. First, the population number is generally between 20 to 100. It shouldn't be too small to provide not enough samples and shouldn't be too large to waste more computation resource. Considering the relatively large number of neurons in our hidden layer, we generated an initial population with 100 individuals. Second, by several times experiment, we chose to set the maximum generation time to 500 in case the model has enough round to get a well-trained result. Moreover, the crossover rate and the probability for mutation are factors helping to explore the better result of genetic algorithm. The setting for those two factors also shows a balance between exploration and exploitation. After several experiment, we finally set the crossover rate and the probability of mutation to 0.6 and 0.1 respectively. The upper bound and lower bound for mutation should be in a proper range for generating new values of weights. Finally, by the experiment in adjusting the initial random value of weight for neuron network in previous work, we finally chose 0.2 and -0.1 as the upper and lower bound.

The structure of our neural network model is a three-layer fully connected feedforward neural network with the same parameters as the model using Pytorch described in chapter '2.1 Data Selection with Pytorch': 91 neurons in the input layer, 50 neurons in the hidden layer and 1 neuron in the output layer, which means the output classification result is a number between 0 and 1, and this number is also used for calculating loss value. As for the confusion matrix, we calculated the result by justifying whether the output number is smaller than 0.5 or larger than 0.5 and classify the result into 0 or 1. Moreover, because we have already used genetic algorithm for better initialization, we set a small learning rate 0.01 in case the model missed the minimum. And we use Sigmoid as the activate function. In addition, due to the large computational effort in the backpropagation algorithm of this model, to speed up the training of the model, inspired by the method of weight decay, we introduce a weight change acceleration factor and a matrix variable that records the change value of the last backpropagation, so that each time the weight value is updated, a part of the change value of the last weight update is added as well [3]. Therefore, our backpropagation weight update formula is as follows:

Assuming that the t-times weight is w_t, the learning rate is η, the value of weight change for the t-times backpropagation is Δt, and the acceleration factor is β, then the formula for the t-times weight update is:

$$w_t \leftarrow w_{t-1} + \eta \times \Delta_t + \beta \times \Delta_{t-1}. \quad (2)$$

We set the value of β to 0.1 and because the number of neurons per pruning is small and our initial number of hidden layer neurons is large, to make the model complete pruning and be fully trained after pruning [4], we set the number of executed epochs to 500.

As for the hyper-parameters in the pruning method with distinctiveness, we need to first observe the vector angles' performance in experiment, then adjust the pruning method to make a more suitable way for pruning. Detailed information will be discussed in chapter '3.3 Model Performance after Pruning'.

3 Results and Discussion

Our work is based on a neural network model that uses the genetic algorithm for initialization and uses the backpropagation algorithm with a high number of neurons in the hidden layer. Then we use the neuron distinctiveness methodology to prune the neurons in the hidden layer. We are going to compare the performance of our model before and after applying the pruning method.

3.1 Model Performance before Pruning

We utilized genetic algorithm and backpropagation algorithm with the hyper-parameters which were already described in chapter '2.3 Hyper-parameter Setting and Special Designs in Our Algorithm ' into our model, and the observed results are shown in Figure 3.

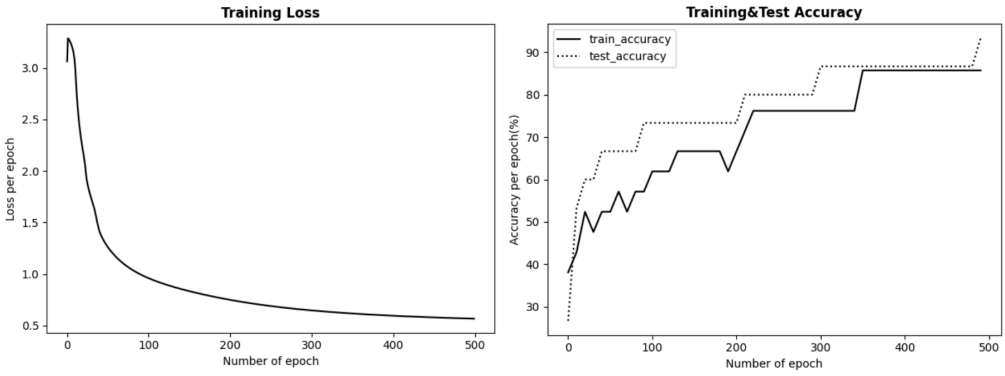

Fig. 3. The loss function image and accuracy image of the model before pruning.

We listed the features of this model:

- The accuracy of the training set is quite high, with the accuracy of the training set basically reaching 90%.
- The accuracy of the test set is around 70%.
- Basically, the loss function can converge quickly and has an ideal convergence curve.

And these are disadvantages of this model:

- The model has some overfitting.
- The model is not always stable, the performance of convergence and the accuracy of test sets can have a small probability (around 20%) of fluctuation, and the accuracy for test sets can fluctuate between 40% and 80%.

- Long run time, basically around 100 second for backpropagation after 500 epochs.

In order to demonstrate the actual effects of the model for subsequent comparisons, the average performance of the data for each comparison item over ten runs was collected and shown in Table 2.

Table 2. The average performance in Training Loss, Training Accuracy, Test Accuracy, the Running Time for Genetic Algorithm (abbreviated as 'GA') and the Running Time for Backpropagation (abbreviated as 'BP') for 500 epochs over 10 runs.

SN	Training Loss	Training Accuracy	Test Accuracy	GA Running Time (Second)	BP Time for 500 epochs (Second)
1	0.558962873	95.24%	80.00%	30	95
2	0.429014118	95.24%	80.00%	30	95
3	0.335046367	100.00%	80.00%	32	103
4	0.070200505	100.00%	53.33%	31	107
5	0.040503257	100.00%	66.67%	33	108
6	0.295112507	95.24%	66.67%	32	105
7	0.023218384	100.00%	73.33%	33	107
8	0.09178553	100.00%	73.33%	33	106
9	0.173912432	100.00%	53.33%	31	103
10	0.052193346	100.00%	60.00%	32	105
Average	0.206994932	98.57%	68.67%	32	103

3.2 Observation during the Pruning

Based on the above neural network model, and according to our pre-defined experimental steps and methods, we started the work of calculating and pruning the distinctiveness of neurons in the model.

It is very important to design a suitable method for pruning, otherwise the pruning method may cut too many neurons or bring negative influence on the performance of model. Therefore, we need to observe the vector's performance before designing the pruning method. Firstly, according to the description of the distinctiveness methodology, we need to construct a matrix for hidden neurons and their pattern vector. we performed a feed forward before pruning and record the performance of each hidden neurons into a separate matrix. Since there were a total of 36 samples and 50 hidden neurons, we ended up with a Neuron Matrix of shape 36×50.

In the resulting Neuron Matrix, each column of data is the pattern vector presented by a neuron at that time after training, which represents the result of that neuron's performance in the model. According to Gedeon and Harris (1991, pp. 25-34), we calculated the vector angle size of the pattern vector between each two neurons by using the following formula [2]:

$$angle(u, v) = \arccos\left(\frac{u \cdot v}{||u|| \cdot ||v||}\right) \quad (3)$$

We had a total of 1225 vector angles calculated during each epoch. According to the distinctiveness methodology, a vector angle of less than 15° is considered too similar for the effect of the two neurons and needs to be combined, while a vector angle of more than 165° is considered too complementary to the effect of the two neurons and needs to be removed at the same time [2]. We observed and counted the vector angles of Neuron Matrix after 100 epochs and found that nearly all the 1225 computed vector angles were less than 15°, and most of the vector angles that less than 15° were less than 5°. Only one of the vector angles was greater than 15°, and no angles were greater than 165°. Details are shown in Figure 4.

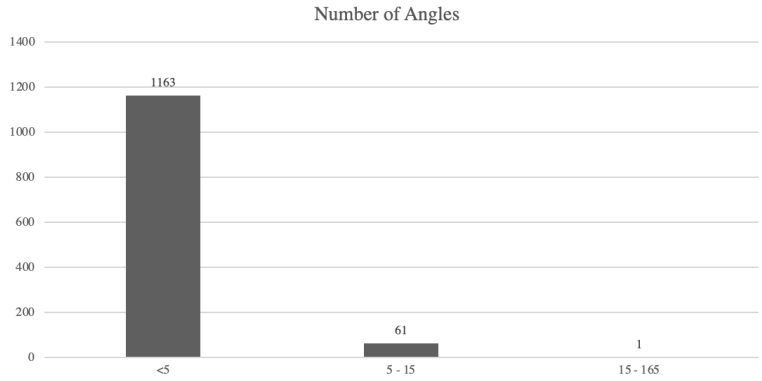

Fig. 4. The number of angles greater than 165°, between 15° and 165°, between 5° and 15°, and smaller than 5° among 1225 vector angles' calculation.

We took a statistical look at the results of the vector angle calculations after every 100 epochs and found that the overall statistics were like those above, with most of the vector angles in each count concentrated below 15° and most below 5°, while there were never more than 20 vector angles greater than 15°, and nearly all the results showed the number of angles greater than 165° were zero.

Based on the distribution of vector angles, we noted that most of the vector angles computed in this model were concentrated at less than 5°, and to avoid pruning the neural network with too large a magnitude that would affect the final model performance, we modified the threshold for screening vector angles to be less than 5°. At the same time, to increase the magnitude of pruning, we added a percentage factor γ for the pruning of neurons corresponding to vector angles less than 5° and set it to 10%, i.e., after filtering all vector angles less than 5° each time, the vector angles are sorted in ascending order by their size, and subsequent merging operations are performed on the neurons corresponding to the top 10% of all vector angles less than 5°. The logic of the merging algorithm is to add the input-hidden weight and hidden-output weight of one of the neurons to the input-hidden weight and hidden-output weight of the other neuron respectively and remove that neuron. Moreover, we would not perform the pruning for angles greater than 165°.

In our experiments, we found that the frequency and timing of the execution of the pruning computation have an impact on the post-pruning effect of the model. For example, if neural network pruning is performed at the end of each epoch, the pattern vector presented by the weights is not very stable after only one epoch calculation, and it is likely to start changing significantly after a few dozen epochs. Therefore, if pruning is started after only a relatively small number of executions, a lot of neurons will be pruned while their pattern vector is not stably represented. Conversely, if the pruning algorithm is executed less frequently, it may not optimise the running time of the model.

We randomly selected 50 vector angles from the 1225 vector angle calculations and observed their changes in the first 50 epochs and first 100 epochs respectively. As shown in Figure 5, we noticed that nearly all the vector angles were quite stable during the 50 epochs and 100 epochs, changes were basically within 15°, and no significant change happened. Based on this behaviour among vector angles' change in different epochs, combined with our pruning method, we decided to set the execution frequency of the pruning method to be executed after every epoch, and started pruning from the beginning of our backpropagation neuron network model.

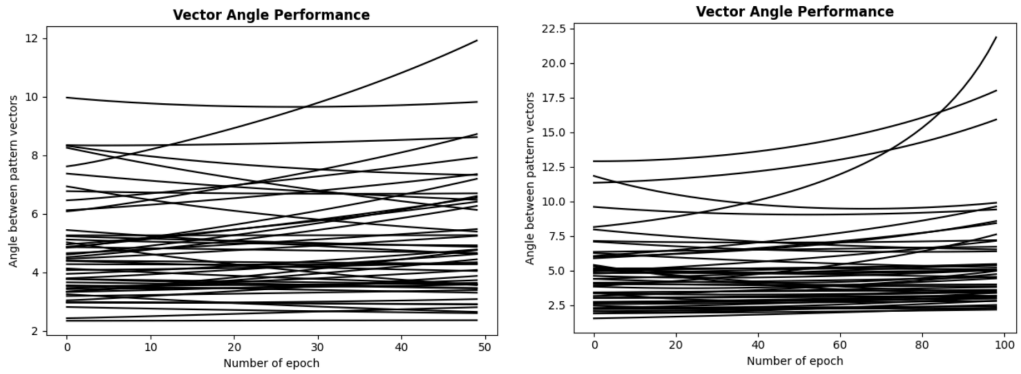

Fig. 5. The vector angle value change of randomly 50 vector angles among 1225 among 50 epochs and 100 epochs, nearly no significant change during different epochs.

3.3 Model Performance after Pruning

We observed the model's performance after applying the pruning method described above. The results are shown in Figure 6.

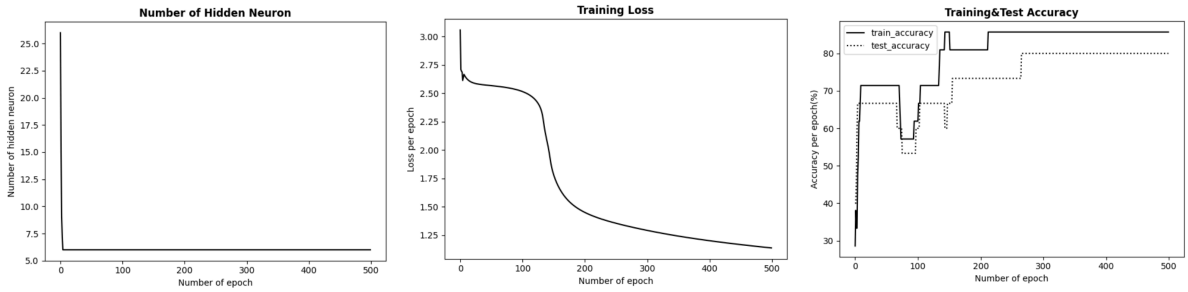

Fig. 6. The change of hidden vector numbers, the loss function image and accuracy image of the model after pruning.

To demonstrate the performance of the final model obtained by this pruning method for comparison purposes, the average performance of the data for each comparison item over ten runs was collected and shown in Table 3.

Table 3. The average performance in Training Loss, Training Accuracy, Test Accuracy, the Running Time for Genetic Algorithm (abbreviated as 'GA') and the Running Time for Backpropagation (abbreviated as 'BP') for 500 epochs over 10 runs.

SN	Training Loss	Training Accuracy	Test Accuracy	Number of Hidden Neuron	GA Running Time (Second)	BP Time for 500 epochs (Second)
1	1.21457907	90.48%	60.00%	7	33	15
2	1.516027059	80.95%	66.67%	6	30	14
3	1.415784679	85.71%	86.67%	8	29	18
4	1.120945948	85.71%	80.00%	7	30	15
5	0.347905381	100.00%	60.00%	8	30	17
6	1.223605942	90.48%	86.67%	8	29	17
7	1.414988891	85.71%	80.00%	8	31	18
8	1.092766747	100.00%	73.33%	6	29	14
9	1.208481591	90.48%	66.67%	7	31	16
10	1.137564758	90.48%	80.00%	7	32	16
Average	1.169265007	90.00%	74.00%	7	30	16

We listed the features of this model:

- The accuracy of the training set is quite high, with the accuracy of the training set basically reaching 90%.
- The accuracy of the test set is around 70%.
- Basically, the loss function can converge quickly and has an ideal convergence curve.

And these are disadvantages of this model:

- The model has some overfitting.
- The model is not always stable, the performance of convergence and the accuracy of test sets can have a small probability (around 20%) of fluctuation, and the accuracy for test sets can fluctuate between 50% and 90%.
- The model has a short running time, basically around 16 second for backpropagation after 500 epochs.

We found that the number of neurons was eventually trimmed down to about 7, which was a significant reduction from the 50 neurons we had set at the beginning. In the training process, the total training loss was slightly higher than the model before pruning, and the training loss would occasionally have a rebound after each neuron reduction, but the overall trend of convergence can still be observed. For the accuracy, the training accuracy, although fluctuating, could still be maintained at around 90%, and the test accuracy, although fluctuating after pruning, could still be maintained at around 70%, and all the accuracy are about the same level as before pruning.

4 Conclusion

In the experiments above we trained a neuron network model with genetic algorithm and backpropagation algorithm, and we pruned the hidden layer neurons using the designed method based on the distinctiveness methodology, demonstrating the optimisation and impact of pruned neural network techniques on models using a high number of features and a high number of hidden layer neurons. We determined the distinctiveness of neurons by calculating the magnitude of the vector angle between neuron pattern vectors and used this as a basis to merge the corresponding hidden layer neurons from the model. Then we observed and compared the neural network model's performance before and after pruning, finding that

the performance of loss convergence and the accuracy of the training set and the test set before and after pruning were basically the same, the loss value after the pruning was a little bit higher than before pruning, and the number of hidden layer neurons after pruning was significantly reduced, and the running speed was significantly shorter. Therefore, in summary, we can conclude that:

- Utilizing genetic algorithm and backpropagation algorithm can help train a neuron network model to predict dataset with large number of features and small number of samples.
- Pruning the neural network model using the distinctiveness methodology can optimise the model in terms of number of hidden layer neurons and running speed, as well as keeping the same level of performance as before pruning.

5 Future Work

Due to the time constraint, we did not conduct experiments on other different methods of filtering the neurons to be pruned (e.g., when obtaining multiple neurons with similar vector angle sizes and all with small angles, we try pruning the neurons one by one and verify the result to determine which neurons should be pruned), and other factors, as well as other pruning methods with different settings, such as changing the number of neurons per pruning, changing the merging strategy during pruning, or changing the epoch and frequency at which the pruning operation is performed. Also, the using genetic algorithm and pruning technique at the same time still leaves some instability in the model's performance. We assume the reason behind this phenomenon is that, if we use genetic algorithm instead of randomly generating initial weights, we introduce a lot more hyper-parameters at the same time and make the situation more related to the setting of hyper-parameters, and both the genetic algorithm and pruning technique are methods for optimization on weights. So, it's worth doing deep research to figure out the detailed changes and performance of weights when applying both techniques in the same model.

In addition, the dataset used in this paper is limited by the sample size and therefore the overfitting problem can have a probability to happen during the training, and a good way to solve this problem is to expand the dataset by collecting more spare natural photos and use extra proper models to extract features from those photos (e.g., Convolutional Neural Networks), which requires more effort to investigate the accuracy of face recognition for historical photographs.

References

1. Caldwell, S. (2021) "Human interpretability of AI-mediated comparisons of sparse natural person photos," CSTR-2021-1, School of Computing Technical Report, Australian National University.
2. Gedeon, TD, Harris, D, “Network Reduction Techniques,” Proc. Int. Conf. on Neural Networks Methodologies and Applications, AMSE, San Diego, vol. 2, pp. 25-34, 1991.
3. Zeng, X. & Yeung, D.S. 2006, "Hidden neuron pruning of multilayer perceptrons using a quantified sensitivity measure", Neurocomputing (Amsterdam), vol. 69, no. 7, pp. 825-837.
4. Goh, Y. & Tan, E. 1994, "Pruning neural networks during training by backpropagation", IEEE, , pp. 805.
5. Lozowski, A., Miller, D.A. & Zurada, J.M. 1996, "Dynamics of error backpropagation learning with pruning in the weight space", IEEE, , pp. 449.
6. Yi, X. 2015, "Selection of initial weights and thresholds based on the Genetic Algorithm with the optimized Back-Propagation neural network", IEEE, , pp. 173.
7. Martínez-Morales, J.D., Palacios-Hernández, E.R. & Velázquez-Carrillo, G.A. 2014, "Artificial neural network based on genetic algorithm for emissions prediction of a SI gasoline engine", Journal of mechanical science and technology, vol. 28, no. 6, pp. 2417-2427.

Appendix A: The Flow Chart for Genetic Algorithm in the Model