Letter Recognition Using Artificial Neural Network Classifier and Genetic Algorithm

Yuchen Li

Research School of Computer Science, The Australian National University
u6013787@anu.edu.au

Abstract. Letter recognition nowadays is widely used in many areas. However, most methods are dependent on the image of letters, thus both training and classification need to store many images, which consumes huge storage space. In this paper, we will use a dataset that extracts 16 numerical attributes from character image, whose space consuming is far less than images. Based on that database we firstly use genetic algorithm to select relevant features, then we construct a two-layer artificial neural network classifier to solve the problem. After that, some neurons are pruned based on "similarity method" to improve the performance of the classifier. In the end, our accuracy reaches 94.77%
Keywords: letter recognition. Artificial neural network, neural network pruning, genetic algorithm, cross validation

1 Introduction

Letter recognition has been a popular problem in computer vision, and it is widely used in industry. For example, in the postal system, letter recognition system can automatically classify postcode, which dramatically increases the speed of mail sorting. With the development of deep learning, the error rate of character recognition (which is a superset of letter recognition) has been decreasing. The error rate has been reduced to 0.27% based on convolutional neural networks [1]. However, most classifiers are based on images. In this case, classifiers take in an image and extract features like corners or convolution with specific kernels, which is time and space consuming. In this paper, we will use a dataset created by Slate [9], which selects 16 features based on the position and number of pixels in a letter image, which takes less time and space. In the dataset, it is possible that some of the features are irrelevant, and these features may overshadow the features that can well explain the classification [8]. Therefore, irrelevant features need to be detected and pruned. Since feature selection can be regarded as an optimization problem, genetic algorithm is applied in this paper [8]. To solve the classification problem, we construct a two-layer artificial neural network. Then we test some network reduction techniques of Gedeon and Harris to improve the performance of our model [5].
The remainder of this paper is organized as follows: Section 2 clarifies the dataset and pre-processing of data. Section 3 talks about the structure of the neural network and the selection of hyperparameter. Section 4 discusses the evaluation method and results of our model. Section 5 introduces a network reduction method and its effect. Section 6 compares our result with another paper that cites the dataset. Section 7 concludes this work and proposes future possibilities.

2 Dataset and pre-process

2.1 Dataset Introduction

We are using dataset created by David [9].The dataset consists of 20,000 instances. Frey & Slate reports that 20,000 instances come from 20,000 character-images, which are based on 20 different fonts [4]. The character images are produced by the basic fonts with random distortion. Each instance consists of one label and 16 attributes. Attributes are extracted from the distribution of the features of pixel distribution. The details of attributes are shown in Table 1.

Table 1. Information of attributes

1	The horizontal position, counting pixels from the left edge of the image, of the center of the smallest rectangular box that can be drawn with all "on" pixels inside the box.
2	The vertical position, counting pixels from the bottom, of the above box.
3	The width, in pixels, of the box.
4	The height, in pixels, of the box.
5	The total number of "on" pixels in the character image.
6	The mean horizontal position of all "on" pixels relative to the center of the box and divided by the width of the box.
7	The mean vertical position of all "on" pixels relative to the center of the box and divided by the height of the box.
8	The mean squared value of the horizontal pixel distances as measured in 6 above.
9	The mean squared value of the vertical pixel distances as measured in 7 above.
10	The mean product of the horizontal and vertical distances for each "on" pixel as measured in 6 and 7 above.

11	The mean value of the squared horizontal distance times the vertical distance for each "on" pixel.
12	The mean value of the squared vertical distance times the horizontal distance for each "on" pixel.
13	The mean number of edges encountered when making systematic scans from left to right at all vertical positions within the box.
14	The sum of the vertical positions of edges encountered as measured in 13 above.
15	The mean number of edges encountered when making systematic scans of the image from bottom to top over all horizontal positions within the box.
16	The sum of horizontal positions of edges encountered as measured in 15 above.

Note. adapted from Frey, P. W., & Slate, D. J. (1991). Letter recognition using Holland-style adaptive classifiers. *Machine learning*, *6*(2), 161-182.

2.2 Pre-process Features

The scale of all features is from 0 to 15. We apply min-max normalization to all attributes to rescale them from 0 to 1. The rescaling formula are shown in (1)

$$x_i = (x_i - X_{min})/(X_{max} - X_{min}) \tag{1}$$

The reason to min-max normalize features to a 0-1 range is that min-max normalization is a linear transformation, it does not destroy the structure of the data. Meanwhile, because we are using sigmoid function as activation function and initial weights and biases are small, if we scale normalize features to 0-1 range, the input of hidden neurons would be around 0, where the gradient of sigmoid function is large, then training process might speed up.

2.3 Pre-process Labels

We use one-hot encode to represent the output. The process of encoding is shown in (2). The length of X is the number of the classes

$$X = (x_1, x_2, \cdots x_j) \qquad \begin{cases} x_i = 1 \; if \; y = i \\ x_i = 0 \; \; if \; y \; \neq i \end{cases} \tag{2}$$

The reason why we use one-hot encode is that in this problem, all classes are independent. The advantage of one-hot encode is that for each pair of the output vectors, they are orthogonal which means they are linear independent. Moreover, their distribution is well-distributed, that is, if we randomly construct a vector, the possibilities that it belongs to any class is equal.

3 Feature selection

In our dataset, there are 16 attributes. It is likely that some features are trivial. If we train our neural network based on these irrelevant features, then the NN is more likely to have a good result on training set while performing badly on the test set, because of the interference of the irrelevant features. Therefore, feature selections can improve the generalization and avoid overfitting. We can regard feature selection as an optimization problem where we need to select the best combination of the features that have the highest accuracy. If we use brute force, there are 2^{16} possible solutions, so we use genetic algorithm.

3.1 Genetic algorithm

Genetic algorithm(GA) is a heuristic algorithm that is widely used in optimization. It is also used in feature selection in machine learning domain [8]. Genetic algorithm stimulates the process of Darwinian evolution, to make solutions with better performance survive and produce offspring solutions with better potential. After a few generations, the optimal or local optimal solution is found. Genetic algorithm includes four main steps: initialization, selection, crossover, and mutation. The main process of GA in shown in Fig.1. The details of GA and how to use it in feature selection will be discussed below:

3.2 Representation

In genetic algorithm, the characteristic of each solution is represented by encoding, which is called chromosome. In this problem, since we are going to decide whether to use the features or not, we simply use binary representation. Each chromosome is a 16-bit binary string, where each feature has a corresponding bit. If a feature is selected, its corresponding bit is 1, otherwise, the corresponding bit is 0.

3.3 Initialization

In genetic algorithm, a set of solutions is called population. In the beginning, we need to initialize a population. We use discrete uniform distribution to randomly create the specific number of binary strings. The reason why we use discrete uniform distribution is that we want the initial population is uniformly distributed over the search space. At last, we add a string which represents that all features are selected. The reason is that after some initial tests, we found that when all features are selected, the result is around 80%. Therefore, we assume that the optimal solution is around where all features are selected. We add it to the initial population to shorten the time of convergence.

3.4 Fitness function

In genetic algorithm, the fitness function represents the performance of a solution, which is often the goal of the optimization problem. In feature selection, we use a simple neural network with a one-hidden layer of 256 hidden units (which is obtained by some pre-test based on all features). For individuals in the population, we apply cross-validation to the neural network based on corresponding features. The fitness function is the accuracy of individuals subtracts the minimum accuracy of the population and add a small number. The reason is that the accuracy is the goal of our optimization, therefore individuals with higher accuracy have higher fitness function. Moreover, after subtracting of the minimum value and adding a small value, the individual with the lowest accuracy has possibility close to 0 (but not 0) to be selected. On the other hand, individuals with high accuracy are more likely to be selected, which increases the speed of convergence.
The mathematical representation of fitness function is shown in (1), where ϵ is an extremely small value, and here we assign it 0.0001

$$f = accuracy - min(accuracy) + \epsilon \tag{1}$$

3.5 Selection

According to Darwinian model, individuals with higher fitness function value will be more likely to stay in the population and to reproduce [3]. The purpose of selection is to eliminate bad solutions from the population. In our feature selection system, based on the fitness function, individuals have their own possibility to be selected. In each iteration, an individual is selected and added to the new population. Repeat the step until the number of the new population reaches predefined size.
The mathematical representation of possibility based on fitness function is shown in (2), where $\sum f$ is the sum of all fitness function value in the population

$$P_x = \frac{f_x}{\sum f} \tag{2}$$

3.6 Crossover

The goal of crossover phase is to produce more solutions. Besides, these solutions are produced by recombining the solutions in the new population after selection [3]. The source solutions are called parents, and the new solutions are called offspring. Because the new solutions are the recombination of their parents, they are partially similar to their parents but have their own differences. Therefore, they can be regarded as the neighbor solutions of some suboptimal solutions.
In our feature selection system, we use scattered crossover like the methods of [8]. We predefine a crossover rate to keep some solutions from parent generation. For individuals in the new population, they have a certain possibility of crossover. While crossover for each individual, we randomly pick another individual from the population and create a random binary vector based on uniform distribution. Then, the binary bits are selected from the first parent if the corresponding bit in the new vector is 1, otherwise, the bits are selected from the latter parent.

3.7 Mutation

Since the offspring are similar to their parents, genetic algorithm easily stuck in local optimal. Therefore, mutation is applied to increase the diversity of the solutions [3], which can increase the search space. After crossover, we will go through all bits of the offspring, and each bit has a possibility to become its logical negation.

3.8 Parameter setting

There are several parameters in GA that need to decide. For DNA size, which is the length of the binary string, obviously equals to the number of features. As for population size, we firstly set 100 as Sharma & Gedeon did in [8]. However, we find that it is too time-consuming. Sharma & Gedeon Use 100 population size over 215 features, therefore we scale down the population size to 30, which has an acceptable speed and ratio of the population size and features is higher than Sharma & Gedeon's setting [8]. The setting of crossover rate is also based on Sharma & Gedeon's setting [8]. We choose mutation rate as 0.005 because every generation 2~3 mutations happen. As for the number of generation, we run 25 generations and stop because the results converge.

Table 2. parameter setting

DNA Size	16
population size	30
crossover rate	0.8
mutation rate	0.002
number of generation	25

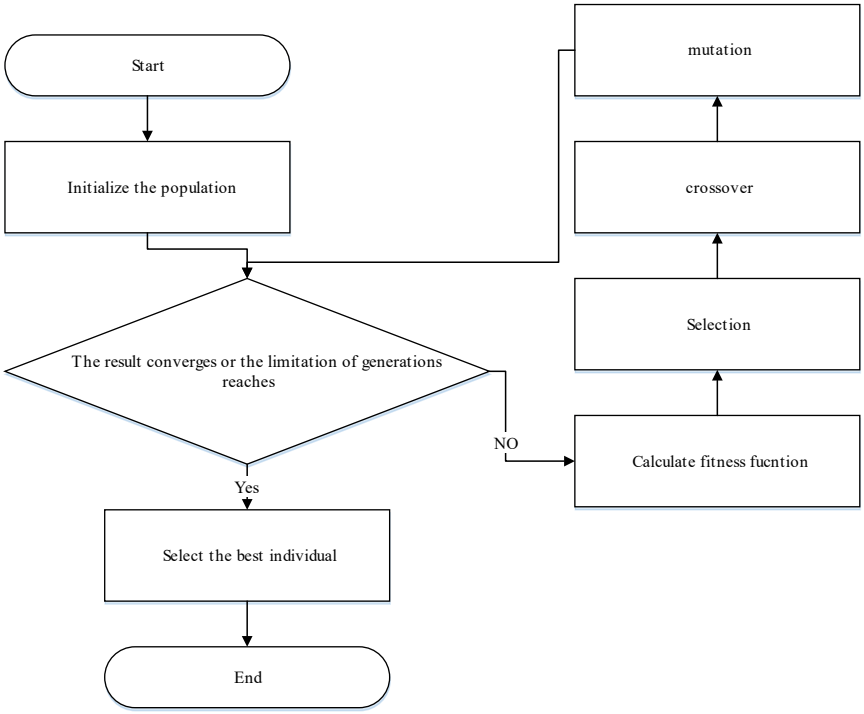

Fig. 1. The flowchart for genetic algoritm

4 Artificial Neural Network

4.1 Structure

Initially, our neural network has an input layer with 14 neurons and an output layer with 26 neurons. Each layer is fully connected with its successor layer, and each layer takes in the linear combination of its previous layer. For hidden neurons, sigmoid function is used as activation function because it is the most common activation function [6]. Our problem is a classification problem, and cross entropy function leads to fast training and better generalization [7]. Thus, cross entropy function is used for loss function.

The mathematical representation of a neuron is shown in (3), a_j' is the output of its previous layer; $w_{i,j}$ is the weight connecting this layer and its previous layer; b_j is the bias; h_j is the input of a neuron; a_i is the output of a hidden neuron

$$h_j = \sum_{i=1}^{..} w_{i,j} * a'_j + b_j \quad (1)$$

$$a_i = \sigma(h_i)$$

At first, we do not decide other hyperparameters like the number of hidden layers or neurons. It is known that it is hard to decide the structure from scratch, therefore we start from some rule of thumb, and try different hyperparameters, compare them together and select the best model.

We start from one hidden layer and 16 hidden neurons. Every time we double the number of hidden neurons until the performance no longer increases. Then we add one hidden layer, and test if the performance increases.

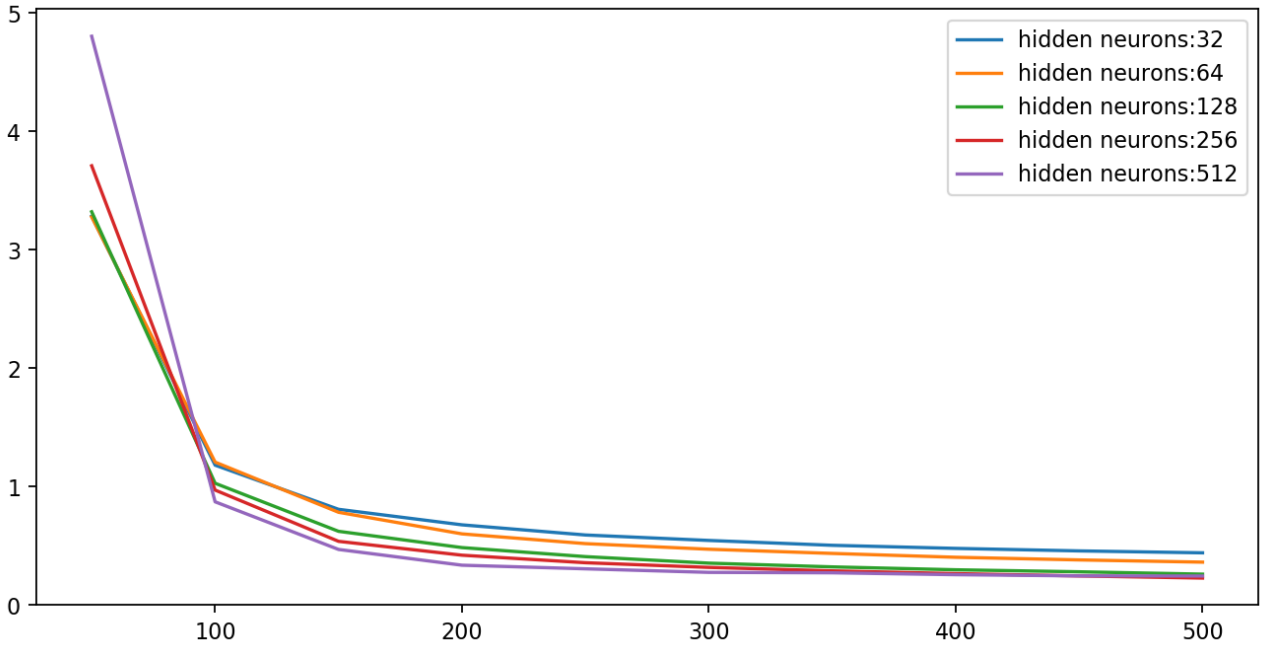

Fig. 2. The loss of validation set for different one-layer hidden neurons (500 epochs)

From Fig.2. we can see that when the number of hidden neurons is 512, the model is overfitting. Thus, we choose 256 hidden neurons.

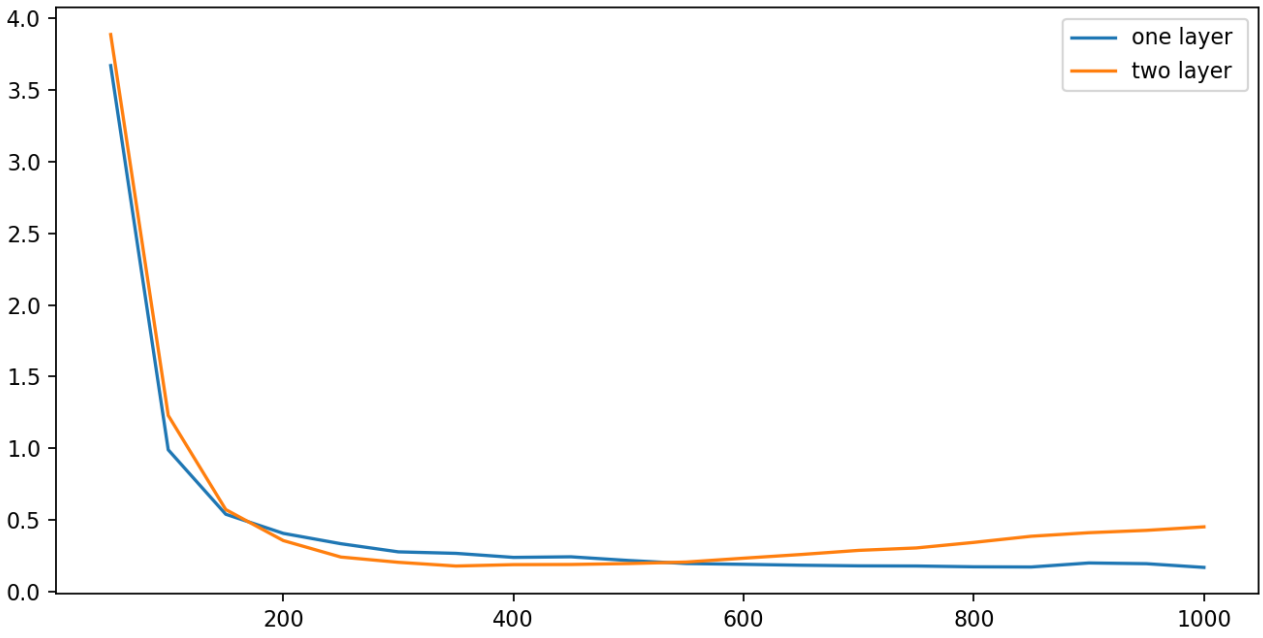

Fig.3. The loss of validation set for one-layer and two-layer hidden neurons (1000 epoch)

According to Fig.3., after adding a hidden layer, the model becomes easier to overfit, and at the same time consume more time and space. Therefore, we choose one hidden layer. In this problem, one hidden layer is enough

After a series of test, our final decision of hyperparameters is shown in Table 3.

Table 3. hyperparameters

Hidden neuron	Hidden layer	Epoch
256	1	1000

4.2 Train

To improve generalisation, we shuffle the data set at first and divide the dataset into 5 subsets. Among them, 4 subsets will be used for training. The last subset is divided into 2 part, one is validation set, the other one is test set, that is, the ratio of the training set, the validation set, and the test set is 8:1:1.
We start from 500 epochs and observe the line chart of loss of training set and validation set. If the gradient of loss is large, it means the model is not converged, then we need to increase the number of epochs. If the loss for validation set begins to increase, which means the model is overfitting, the appropriate number of epochs should be the value of inflection point.

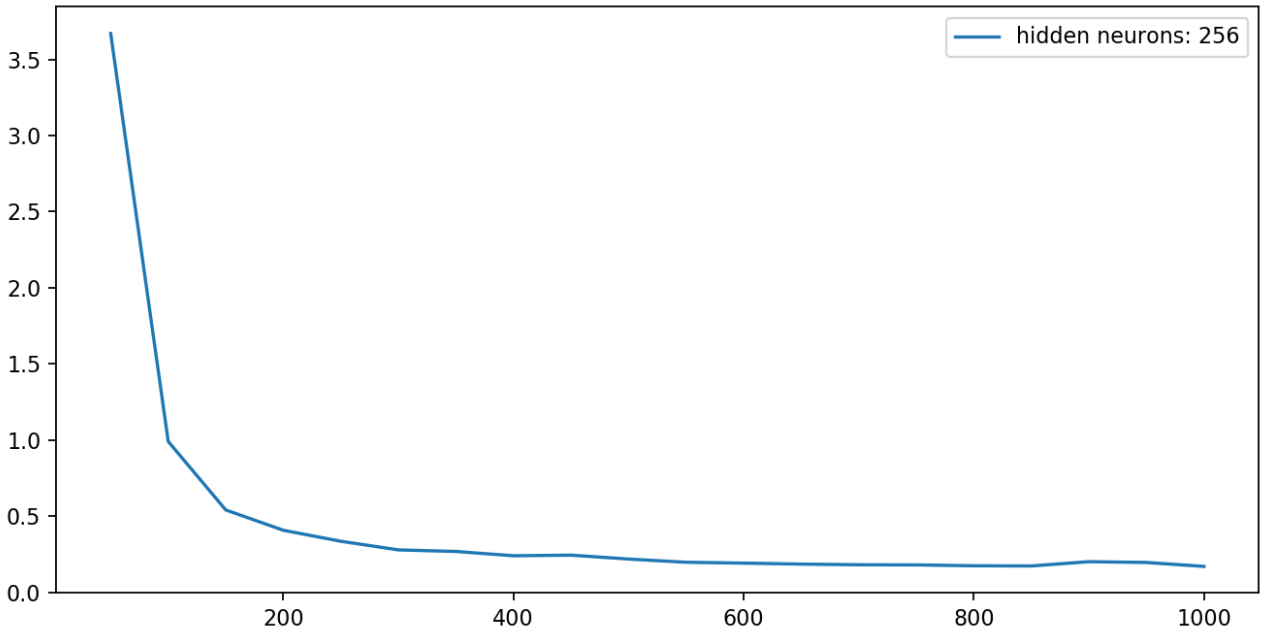

Fig. 4. The loss of validation set for different one-layer hidden neurons (1000 epochs)

As shown in Fig.4. we can see that the gradient of 256 hidden neurons is almost zero, and there is no overfitting, so we choose 1000 epochs.

4.3 Optimizer

We firstly choose Batch Gradient Descent(BGD) as our optimizer. We use experiments to compare BGD and stochastic gradient descent and we choose Resilient Backpropagation(Rprop) which has a faster speed of convergence.
The algorithm of Resilient Backpropagation is based on backpropagation.
For pure backpropagation, firstly we randomly initialize the weights in the network, then we fit a batch of data X into the network. Then we can obtain some output Y. After that, we use loss function between output Y and desired output Y‘. Based on the loss function and the chain rule, we update weights from output side to input side.
We repeat the process until the loss function converges or the model is overfitting.
The major difference between BGD and Rprop is that Rprop can automatically update its learning rate.
The main algorithm is that each weight has its own step size. When we do backpropagation, we record the sign of error of previous backpropagation. If the previous error has the same sign, which means that the weight has not crossed its optimal value, the step size increases. Otherwise, the step size decreases.
With automatically adaptive learnning rate, Rprop shows a much faster training speed.
Therefore we choose Rprop as our optimizer.

4.4 Evaluation

We choose 5-fold cross validation as our evaluation method. During the experiments, we observed overfitting. Therefore, cross-validation is applied to detect overfitting[7].
Due to a large number of data, leave-one-out cross-validation consumes too much time, thus -fold cross validation is used. In this problem, we use 5-fold cross-validation.
As discussed above, the dataset is shuffled every time and divided into 5 subsets. Every time we pick one subset for test and validation, and the rest for training. At last, we compute the average and variance of loss over 5 experiments.

4.5 Network Reduction

As shown above, we have used brute-force way to determine the number of hidden neurons, which means that there might exist redundancy hidden neurons. Therefore, some pruning techniques are applied to reduce redundancy hidden neurons.

There several types of hidden units that can be pruned[5]. Oue paper is based on "distinctiveness" method of Gedeon and Harris [5]. The main process of this method is to go through all hidden neurons and find pairs of neurons with similar or contradictory functionality. After training, we can obtain the outputs of hidden neurons for all data in the training set. For each hidden neuron, we use its outputs consist of a vector which has the same dimensionality as the number of data in training set[5]. The functionality of the hidden neurons is represented by the vector [5]. Thus, we can determine the similarity of two hidden neurons based on the distance of their vectors, and according to Gedeon and Harris, we will use the angle between them as described in (2) [5]. Because the range of sigmoid function is (0,1), the range of angles any two vectors is (0°,90°), thus we normalize the vector by minus 0.5 for each dimension. After computation of all angles, we removed all pairs of neurons that angles between them are larger than 165°, because their functionality is offset[5]. After that we select the angles that are smaller than 15°, then we prune one of the two neurons, because they have similar functionality we only need one of them [5]. After pruning, as Gedeon and Harris said, no further training is required[5]. However, in our experiment, the performance decreases.Therefore, We re-train the pruned neural network for another 100 epochs.

$$\langle A, B\rangle = \arccos\left(\frac{A \cdot B}{|A| \cdot |B|}\right) \tag{2}$$

5 Result and discussion

5.1 Feature selection

As show in Fig. 5., during the process of genetic algorithm, the average of fitness function values increases, which means the solutions in population become better. After 25 generations of genetic algorithm, we notice that the accuracy begins to fluctuate and converge around 85%. Therefore, we stop the process. We found that in the last 5 generations, there are two solutions that often have the highest fitness function value: one solution with all features selected and the other one with all solutions except the fourth feature (the height). Because of the random initialization, the same feature selection has different results. We compare the average and variance. At last, we choose to remove the height feature because that feature selection has lower variance.

Table 3. Description of featuer selection

	mean	Variance
All features	83.98%	7.4e-4
Remove the height feature	83.49%	4.6e-5

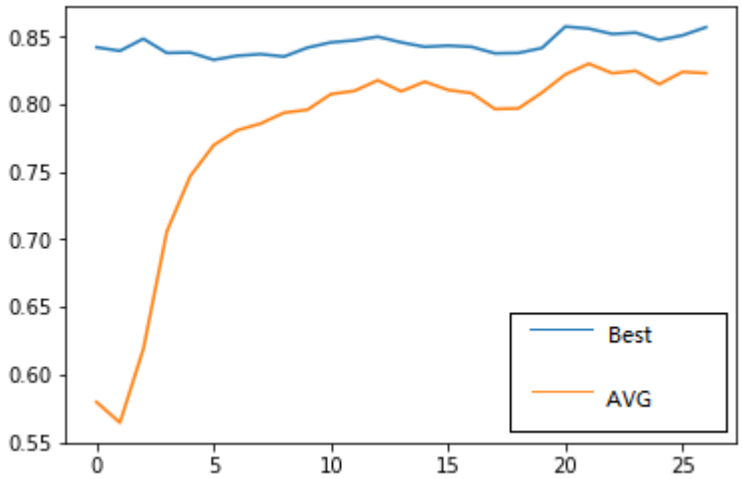

Fig. 5. The average and best fitness function value in the population

5.2 Neural network

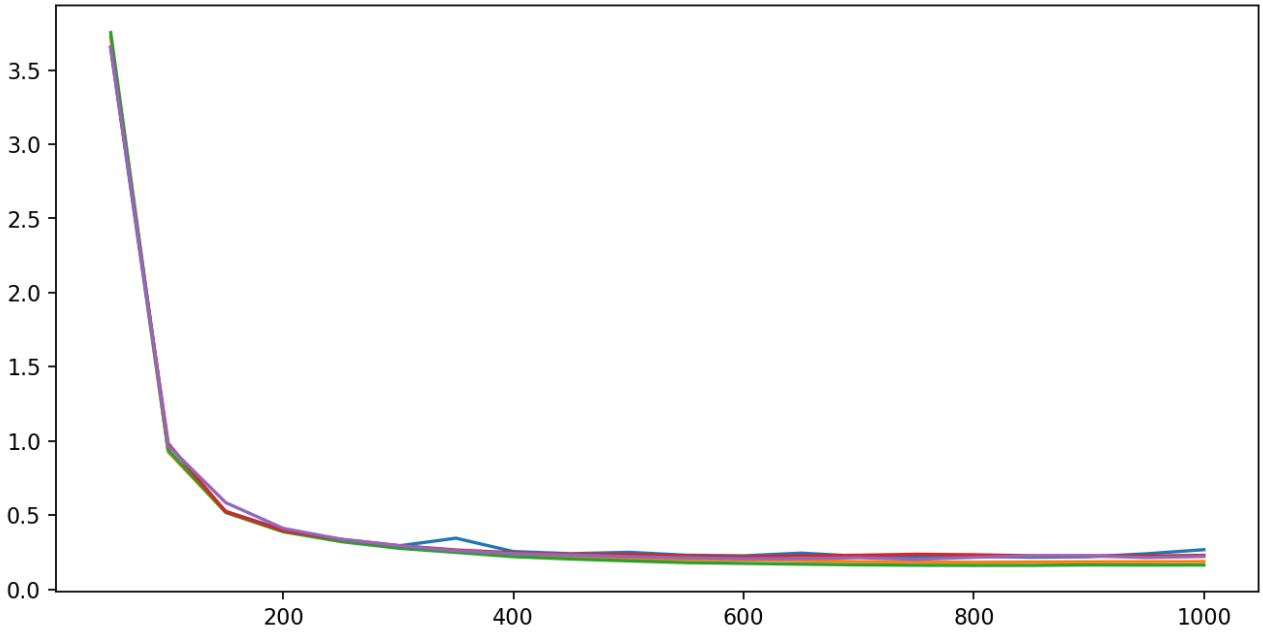

Fig. 6. Loss for validation set with 5 folds (epochs = 1000)

Table 4. Description of test

Average_loss	Variance_loss	Average_accuracy
0.1889	0.0003	94.77%

From Fig.6 we can see that for some dataset, the value of loss marginally increases, which means overfitting happens. Fortunately, the overfitting is not huge so the accuracy does not decrease too much. The low variance of loss shows that our model is stable.

5.3 Network reduction

As shown in Table 5, pruning cause some damge to the neural network's performance, although the number of pruned neurons only takes up about 5-10% of total hidden neurons.
The possible reason that this method doesn't work here is that the although some outputs of neurons are similar, or complementary, these neurons still have a necessary contribution to the classification.
Another probable reason is that other neurons may be able to substitute pruned neurons, but further training is required.
Therefore, we retrain the network after pruning for 100 epochs and then we find that the performance increases dramatically and sometimes become even better than the network before pruning.
On the other hand, before pruning, for some dataset, the model is overfitting, while after pruning the accuracy of the network still increases during training. Therefore, pruning increases the generalization of the model.

Table 5. Comparison of accuracy for test set between network before and after pruning

Number of pruned neurons	19	10	23	8	9
Before pruning	94.55%	94.50%	94.20%	95.70%	93.85%
After pruning	85.05%	92.40%	80.25%	90.65%	67.20%
Retrained	94.50%	94.75%	95.00%	95.80%	94.25%

5.4 Related work and compare

Dietterich and Bakiri used the same dataset [2]. The major difference is that in their paper, they used conjugate gradient algorithm to update the weight, and they introduce an encoder named error-correcting to encode the output.
As shown in Table 6. Compared to their model, our model has limited ability on small size dataset.
One possible reason is that we are using Rprop while they are using conjugate gradient algorithm. On small dataset using Rprop is easier to overfit.
Another likely reason is that we are using one-hot encoder, while they are using the error-correcting encoder. Compared with error-correcting encoder, the output matrix of the one-hot encoder is more sparse, which requires more training data.

Table 6. Comparison of accuracy between two models

Training set size	200	500	1000	10000
Our model	50%	66%	67%	93%
D&B model	60%	75%	82%	92%

6 Conclusion

In this paper, we have shown the process of feature selection based on genetic algorithm, training and evaluating a neural network, and reduction of a neural network. We firstly pre-process the dataset by normalization and one-hot encoding and use genetic algorithm to select appropriate features for our neural network. Next, we construct a neural network and choose sigmoid as activation function and cross-entropy as loss function based on literature. We also empirically determine a two-layer structure with 256 hidden neurons and 1000 training epochs. Our network has average 94.77% accuracy over cross-validation. Then, we prune the hidden neurons that have similar or contradictory functionality. We found that although the pruning damages the neural network, after few epochs re-training, the pruned neural network performs better than the original neural network. Finally, we compared our model with Dietterich & Bakiri's model which uses the same dataset. We found that our model works better when training with a large dataset, However Dietterich & Bakiri's model has better performance with small size dataset.

7 Future work

Due to the limitation of the performance of our computers, we only use genetic algorithm in our feature selection step. However, as a good optimization problem solver, we believe that genetic algorithm can also be used in the setting of the hyperparameters in neural networks. Moreover, nowadays deep learning is more and more popular these days, I believe the approach of network reduction can also be used to optimize the structure of deep neural networks.

8 References

1. Ciresan, D. C., Meier, U., Gambardella, L. M., & Schmidhuber, J. (2011, September). Convolutional neural network committees for handwritten character classification. In Document Analysis and Recognition (ICDAR), 2011 International Conference on (pp. 1135-1139). IEEE.
2. Dietterich, T. G., & Bakiri, G. (1994). Solving multiclass learning problems via error-correcting output codes. Journal of artificial intelligence research, 2, 263-286.
3. Engelbrecht, A. P. (2006). Fundamentals of computational swarm intelligence. John Wiley & Sons.
4. Frey, P. W., & Slate, D. J. (1991). Letter recognition using Holland-style adaptive classifiers. Machine learning, 6(2), 161-182.
5. Gedeon, T.D. and Harris, D. (1991) "Network Reduction Techniques," Proceedings International Conference on Neural Networks Methodologies and Applications, AMSE, San Diego, vol. 1: 119-126.
6. MacLeod, C. (2010). An introduction to practical neural networks and genetic algorithms for engineers and scientists. *Scotland: The Robert Gordon University*.
7. Nasrabadi, N. M. (2007). Pattern recognition and machine learning. *Journal of electronic imaging*, *16*(4), 049901.

8. Sharma, N., & Gedeon, T. (2013, April). Hybrid genetic algorithms for stress recognition in reading. In *European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics* (pp. 117-128). Springer, Berlin, Heidelberg.
9. David J. Slate Letter Image Recognition Data, https://archive.ics.uci.edu/ml/datasets/letter+recognition