Implementation and Optimisation of Neural Networks on Letter Recognition Dataset

Liwei Hou

Research School of Computer Science, Australian National University
u6343089@anu.edu.au

Abstract. This paper studied on the implementation of neural network on the Letter Pattern Recognition Data set, and used Heuristic Pattern Reduction [2] and genetic algorithm for feature selection to optimise the model. As results, the Heuristic Pattern Reduction succeeded in reducing computational costs while keeping the model's generalisation ability from decrease. On the contrast, due to the nature of the data, genetic algorithm for feature selection played a limited role and exposed its limitations. This paper also tested many parameters affecting the prediction ability of neural networks through the principle of control variables, and successfully made the model perform better than the Holland-style adaptive classifier systems introduced by Frey and Slate [5].

Keywords: Neuron network, letter recognition, heuristic pattern reduction, genetic algorithm

1 Introduction

This paper is aimed to implement an artificial neural network, then use optimisation strategies to improve the model and compare it with other related papers to reflect the advantages and disadvantages of neural networks and the optimisation methods used.

In this paper, a classification problem on Letter Recognition Data Set from the UCI website was chosen [1]. The problem to solved is to identify capital letters in the English alphabet from lots of black-and-white rectangular pixel. Each of these 20000 images were randomly distorted from 20 different fonts of English capital letters and converted into 16 numerical attributes. This dataset is chosen because of its moderate size, which means the training time can be controlled within an acceptable range, and the number of instances is large enough for the training of neural networks. In addition, the attributes are also appropriate because of its moderate size for implementing a genetic algorithm to optimise the feature selection, and its numerical nature eliminates the need for complicated pre-processing. Besides, choosing this dataset also has practical meaning. Letter recognition is considered as an important technique in pattern recognition, artificial intelligence and computer vision, which can be used in Human Machine Verification, handwriting recognition, document scanning and many other fields.

In the following work, I will use genetic algorithm to optimise the feature selection, then implement an artificial neural network to fit the data and use heuristic pattern reduced introduced by Gedeon and Bowden [2] and other parameter adjustment methods to improve it. Then it will be compared with models from other papers based on the same dataset and discuss whether the model implemented in this paper is better or not.

2 Method

2.1 Data Preprocessing

The raw data is ideal, because all the 16 numeric features have been scaled into a range from 0 to 15 and there are no missing attribute values. As for the distribution of *letter*, the target variable to be classified, as Fig. 1 shown, the 26 classes are in a balanced distribution, which means there are no necessary to do oversampling or undersampling. Besides, the numerical nature of all 16 attributes makes encoding and decoding unnecessary. Therefore, I performed standardisation on the data, which is a basic preprocessing method that subtracts each value of dimensions of the data from the average of that dimension and then dividing it by the standard deviation of the dimension, thereby normalising each dimension to the same scale. This formula is given by:

$$z = \frac{x - \overline{x}}{S} , \tag{1}$$

where $\overline{x}$ is the mean of the dimension, S is the standard deviation of the dimension and $S \neq 0$. Then headlines were added to the data for a clear view. After that, *letter*, the target categorical variable, was converted into numeric according to alphabetical order. For example, A was converted into 0, and Z was converted into 25.

After pre-processing, the data was randomly separated into a 80% training dataset and a 20% testing dataset.

2.2 Implement the Neural Network

In this paper, PyTorch 4.0 has been used to implement neural network on Python 3.6. When the operating environment supports, *CUDA* will be used to perform Pytorch Tensors and models on NVDIA GPU for a shorter training time. Some code that will be reused, such as partitioning data into X and Y are encapsulated

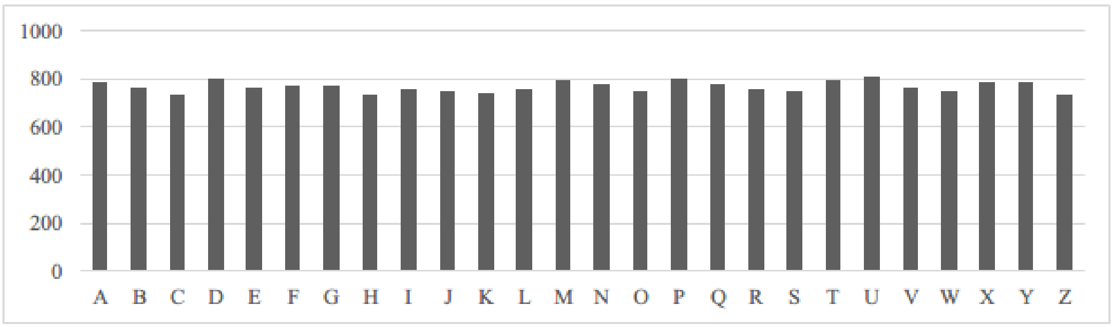

Fig. 1: Class Distribution of Dataset, the abscissa represents classes while the ordinate represents the number of each class.

into functions to save time and effort and reduce the possibility of bugs. For the main part of the code, since the model used for prediction is only trained once, it does not need to be encapsulated.

More complex artificial neural networks than that used in genetic algorithm for optimisation were trained for a better performance. This neural network is a multi-layer feed-forward network with back-propagation, in which units between two adjacent layers are connected to each other, without backward, horizontal, or multi-layer connection. The units of the previous layer and that of the next layer are connected by Linear function. In order to find the parameters that optimise the model, a large number of different combinations of neurons and hidden layer numbers, optimisers, activation functions, learning rates, and updating epochs were tested. After each parameter adjustment, firstly stratified 10-fold cross-validation will be performed to ensure that the model is not overfitted, then a model will be trained with full training set and run prediction on the test set, which will be the final result of this model.

2.3 Genetic Algorithm

For machine learning, the selection of features is significant. On the one hand, selecting features that irrelevant or have limited association with the target variable will increase the noise that the model learning from. On the other hand, Selecting features that have a large or even linear correlation with the target will largely improve the prediction accuracy of the model. However, datasets often contain a large number of features, making it impractical to test every possible combination of features through exhaustive search. In contrast, a better approach is to use genetic algorithms for feature selection.

Genetic algorithm is an evolutionary algorithm inspired by the theory of natural selection. This algorithm is actually a random search algorithm that generates solutions to optimisation problems through bio-inspired mutation, crossover, and selection operations. In this paper, genetic algorithm is used to find the optimise features for training. In addiction, the python package deap has been used to implement the it.

The flowchart of the algorithm is shown in Fig. 3. First, the selection of the 16 features will be encoded as 16-bit chromosomes as Fig. 2 shown, in which each bit corresponds to whether a feature is selected or not. Then the population will be randomly initialised by the Bernoulli distribution with probability p=0.5. After that, the population will be evaluated by training a simple neural network using the feature selection expressed by chromosome, where the fitness value will be the accuracy of the model. If the generation of breeding population has reached a predetermined threshold, the genetic algorithm will be terminated, and the individuals with the highest fitness in the current generation will be chosen, then the features selected by its chromosomes will be considered as the optimal feature selection. On the other hand, if the threshold has not been satisfied, the algorithm will proceed to the next step, crossover. This operation is to exchange some genes between a couple of chromosomes using the function *cxOrdered*, which is a simulation of chromosome exchange and recombination in biology. The next step is mutation, which is randomly generated according to a predefined probability. When a mutation occurs, the *mutShuffleIndexes* function will be used to shuffle individual features and then return them into the chromosome. After that, another evaluation will be held, where the *selRoulette* function will be used to randomly select individuals and select the one with the highest fitness value. After selection, the algorithm will return to the termination step, and then cycle in the order of crossover, mutation, and selection, until the threshold satisfied.

In order to reduce the computational complexity of the algorithm, during the fitness values evaluation, a very simple neural network structure (with a hidden layer and 5 hidden neurons) will be used and 30% of training set will be extracted hierarchically by applying the heuristic pattern reduction described later. Besides, training will be terminated under 500 generations because there is no need to fully train the model.

After a number of generations, the genetic algorithm will decide the optimal feature selection. In the later work of this paper, these optimal features will be applied to the training of more complex neural networks.

2.4 Heuristic Pattern Reduction

Heuristic pattern reduction has been used to improve the performance of the model. According to Gedeon and Bowden [2], heuristic pattern reduction is a method that can raise the performance of the model and decrease

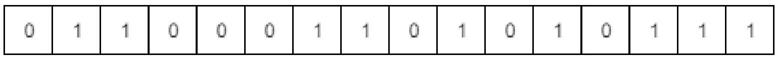

Fig. 2: The structure of the chromosome is a 16-bit list, where each bit represents a gene. The expression of each gene is determined by the value of the bit. 0 represents no expression and 1 represents to be expressed.

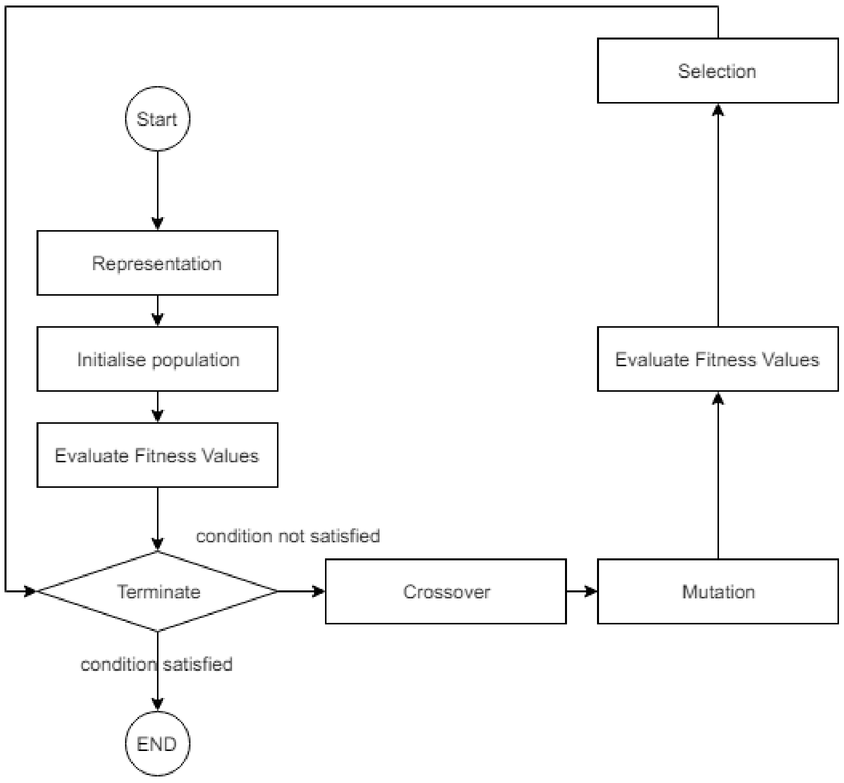

Fig. 3: The flowchart of genetic algorithm for feature selection.

the training time by reducing the size of the training dataset. In this paper, the specific implementation method is to keep the overall characteristics of the data set by using stratified sampling of *letter* by 10% less each time.

2.5 Evaluation Methods

Stratified 10-Fold Cross Validation has been used to validate the model, which is a validating method that divides training set into 10 folds and uses 1 of them for testing and the other 9 for training each time to train a neural network model, which means there are 10 model that will be trained each using 90% of the training set. Different from ordinary cross-validation, stratified cross-validation uses stratified sampling to partition the validation set. This allows the partitioned validation set and training set to retain the characteristic of the original data set as much as possible and minimise the error between the prediction results on the validation set and the actual performance of the model. In the process of cross-validation, the test set will be completely independent of cross-validation, which means that the model will never contact the test set during the entire training process. This ensures that the trained model will not overfit the test set, and thereby ensures the prediction result on the test set will be reliable.

In addiction, confusion matrix has been used in evaluating the model. In the matrix, the columns represent the instances in actual classes, and the rows represent the instances in predicted classes. For each class, the number of times the class was correctly classified will be distributed on the diagonal of the matrix, the number of times other classes are erroneously labelled to this class will be distributed in other positions on the row. The number of misclassification of this class to other classes will be distributed in other positions on the column. The significance of this matrix is clearly reflecting the model's performance in predicting different classes, and thus enabling observations on which categories are difficultly distinguishable for model.

Cross entropy loss has also been used in evaluation. The meaning of this value is to describe the distance between the model and the ideal model. In Pytorch [3], this function is defined as:

$$loss(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right) . \tag{2}$$

Comparing with accuracy or the classification error, which is given by:

$$classification\ error = \frac{count\ of\ error}{count\ of\ all\ items} , \tag{3}$$

cross-entropy loss can provide a more accurate evaluation of model quality during training.

3 Results and Discussion

3.1 Genetic Algorithm

Different combinations of population sizes and breeding generations were examined as illustrated in Tab. 1. Theoretically, the genetic algorithm will converge to several optimal values when there are enough generations. In this result, the optimal feature selection is all 16 features, which is explainable because the features of this data set are all derived from the key features of the letter images (for example, positions and mean edge count), which makes each feature have an important contribution on letter recognition.

Generations	Population	Optimal Chromosome	Accuracy
20	10	1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1	94.89%
20	10	1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1	89.76%
30	10	1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1	93.95%
20	30	1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1	94.92%
50	10	1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1	95.03%
50	30	1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1	94.96%
50	1	0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1	45.52%
50	5	0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0	75.79%

Table 1: Results of genetic algorithm using different generations and population settings

Interestingly, the experimental results also reflects the instability of the genetic algorithm. As shown in the table, under the same generation and population settings (20, 10), due to the different population initialisations and the large number of random variations and crossovers in the evolutionary process, the final presented results may vary greatly, indicating that the genetic algorithm is easy to fall into local optimum. The solution is to increase the number of populations and generations, and use larger mutation probabilities. However, while increasing the size of the algorithm, the computational complexity of the algorithm will greatly increase, which makes it too expensive to simulate an extremely large-scale population breeding to obtain the optimal selection. Therefore, balancing the size of the algorithm and the robustness of the algorithm is a significant issue. In the actual test, when the population is set to 10 and the generation is 50, and the mutation probability of 0.1 is used, the probability of generating a global optimum is high enough.

3.2 Adjust Parameters

Hidden Layers and Neurons. Different combinations of layers and neurons have been tested. As Fig. 4 shown, the three curves have similar trends, and the results are very close at high neuron numbers, indicating that a single-layer neural network running on this data is sufficient for saturation training effects. In addiction, The curves are approaching the 100% with decreasing slopes, which makes it considerable to take the number of neurons slightly higher than the corners of the curve in order to balance the training time and prediction accuracy of the model, because after this corner, the effect of increasing the number of neurons and hidden layers is limited while the computational complexity growing more rapidly. Therefore, the best combination is one hidden layer and 128 hidden neurons.

Activation Functions. As a classic activation function for machine learning, sigmoid was intuitively considered first, which is defined as:

$$\sigma(x) = \frac{1}{1+e^{-x}} \ . \tag{4}$$

Although this function has advantages such as the easy calculation of its derivation, there are three major drawbacks that make it less practical. Firstly, in its derivation, the input will be compressed to 1/4 of the original, which will lead to gradient vanishing, in other words, when the number of hidden layers increases, the derivative of the back layers will be very close to 0, which makes the back propagation algorithm difficult to converge. Secondly, the output is not zero-centred, which means the derivatives and inputs are always either both positive or both negative, leading to a slow convergence of the back propagation algorithm. Thirdly, the exponentiation in sigmoid is time consuming.

On the other hand, unlike sigmoid, ReLU (Rectified Linear Units), which is defined by:

$$\mathrm{ReLU} = \max(0, x) \ , \tag{5}$$

solves the problem of the gradient vanishing, which improves the convergence speed of the back-propagation algorithm. At the same time, it does not require exponentiation, which largely reduces its computational complexity. However, this function also has disadvantages. Firstly, similarly to sigmoid, its output is also not zero-centred. Secondly, this function has a dead ReLU problem, that is, some neurons may never be activated, resulting in the corresponding parameters will never be updated. This problem is due to parameter initialisation or too high learning rate.

Therefore, ReLU (Rectified Linear Units) was used instead of sigmoid.

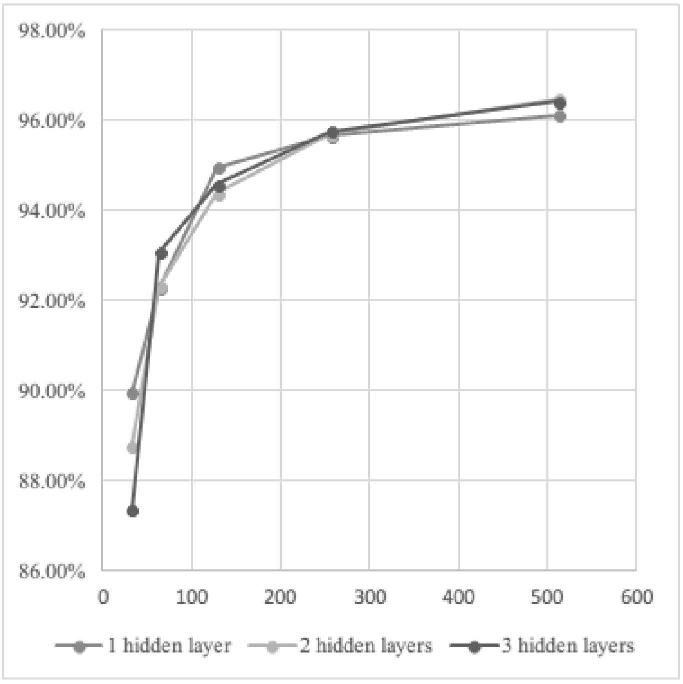

Fig. 4: Relationship between accuracy and number of hidden layers and neurons, the abscissa represents hidden units in each hidden layer, and the ordinate represents the accuracy of predictions using corresponding model, the three fold lines respectively represent one, two and three hidden layers.

Optimisers. Many optimisation algorithm were tested, including SGD (Stochastic Gradient Descent), RMSprop (Resilient Backpropagation), Adadelta, Adam (Adaptive Moment Estimation) and Adamax. Fig. 5 shown illustrates the performance of models trained with different optimisers, from which it can be found that models with Adamax, Adam and RMSprop have higher accuracy on the testing set, among which Adam has the lowest cross-entropy loss. In addiction, this optimiser also minimised the training time. Therefore, Adam is the best performing one among the candidate optimisers.

The good performance of Adam is closely related to its nature. It is an algorithm for "first-order gradient-based optimisation of stochastic objective functions, based on adaptive estimates of lower-order moments" [4]. This optimiser has high computational efficiency, little memory requirements, no variance on diagonal rescaling of the gradient, and is suitable for issues with large data.

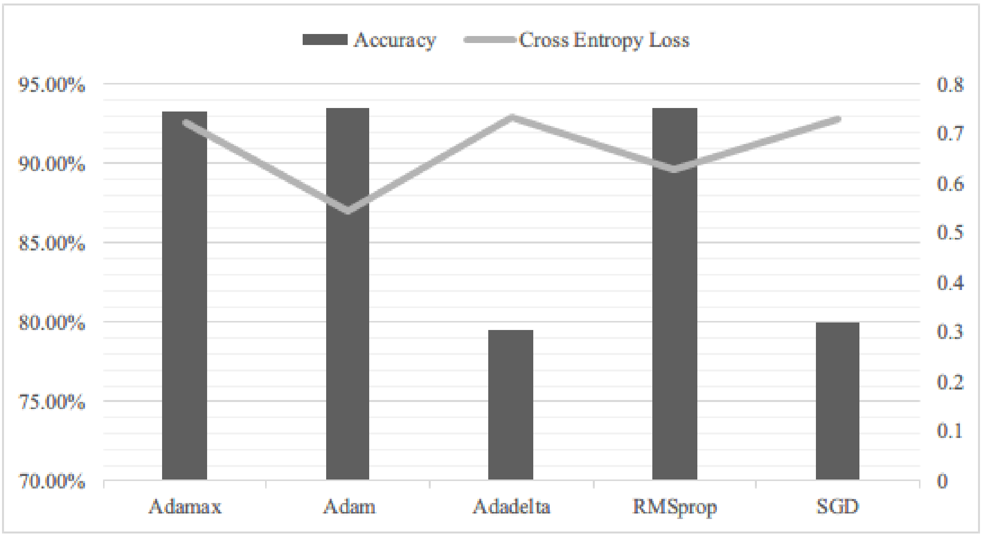

Fig. 5: Accuracy and Cross-entropy Loss of Models using different optimisers. The abscissa represents the optimiser, the main ordinate represents the accuracy, the secondary ordinate represents the the cross entropy loss.

Learning Rates and Updating Epochs. High learning rates can reduce the time cost of the calculation. However, it may cause the optimization algorithm to fall into a local minimum. Therefore, a self-decay learning rate is necessary because it can reduce the time for the gradient to approach the optimal value, while increasing the flexibility of learning when approaching the optimal value. Because Adamax features self-decay, manual adjustment on decay parameters is not necessary. After testing several initial learning rate, 0.01 has been chosen.

The update epoch number changes with other parameters. When the loss no longer decreases, the training should be terminated to avoid overfitting and reduce the time cost.

3.3 Heuristic Pattern Reduction

Fig. 6 shown is the result of using heuristic pattern reduction. According to the figure, before reducing to 50% of training set, as the patterns participating in training hierarchically decreases, the decrease on accuracy of model prediction is limited, and the increase on cross-entropy loss is also within a limited range. After the model reduced to less than 50% of training set, the change in accuracy and cross-entropy loss reaches a relatively significant scale. This shows that when the patterns are reduced to half, the generalisation ability of the model is still not significantly reduced. Therefore, it is considerable to hierarchically halve the patterns in order to reduce computational costs.

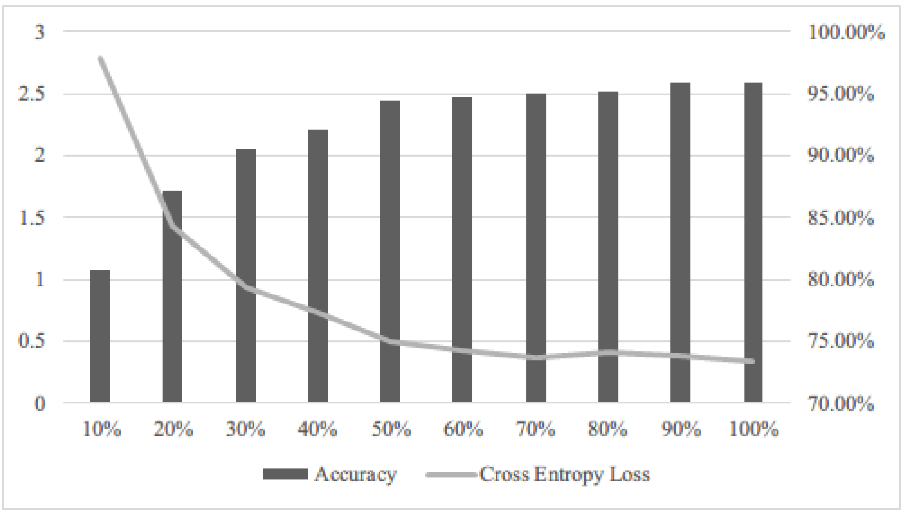

Fig. 6: Accuracy and Cross-entropy Loss of Models after applying Heuristic Pattern Reduction. The abscissa represents the proportion of data sampled by stratification, the main ordinate represents the the cross entropy loss, the secondary ordinate represents the accuracy

3.4 The Final Parameters and Performance

The final model is a one-hidden-layer neural network with 128 hidden neurons, using Adam as optimiser and ReLU as activation function, trained on a 50% training set with a learning rate of 0.01 for 10000 epochs, with feature selection completed by genetic algorithm. This model finally performed an accuracy of 95.69% and a cross-entropy loss of 0.413 on testing set. The average accuracy on the 10-fold cross-validation is 95.03%, which indicates the model did not overfit the data.

3.5 Comparison with other paper

Frey and Slate used Holland-style adaptive classifier systems to perform machine learning on this data set [3]. According to the authors, the best accuracy of their models was 81.6%, which is a worse result comparing with the neural network model implemented in this paper. The most likely reason of this difference is the algorithm. According to the authors, the principle of Holland-style adaptive classifier systems is creating a list of condition-action rules and parallelly applying it to messages which represent whether specific features exist in the current environment. This algorithm is largely different from neural networks, which can be a reason of the disparity of performance.

4 Conclusions and Future Work

This paper constructed a neural network model on Letter Recognition Data Set and improved it with appropriate methods, including parameter adjustment, heuristic pattern reduction and genetic algorithm for feature selection. In the steps of optimising the model by comparing multiple methods, the principle of control variables has been strictly followed. Because of the relatively large size of data, this paper reasonably balanced the training time and the generalisation ability of the model.

Through stratified sampling of target variables, the heuristic pattern reduction largely reduced the training time of the model while keeping the generalisation ability of the model from decreasing, which proved the advantage of this method. As for the genetic algorithm for feature selection, it did not improve the accuracy of the model because the optimal feature selection on the data set is selecting all 16 features. This is due to the fact that each feature of the data is largely relevant to the target variable. The experiment also shows the limitations of the genetic algorithm, such as easy to fall into local optimum and high computational cost. This paper also compared with Frey and Slate's Holland-style adaptive classifier systems on the same dataset. As a result, this paper performed better on the data set due to the advantages of artificial neural network algorithm.

Theoretically, the continued expansion of the size of neural networks by increasing the number of hidden layers and neurons and training more generations until overfitting could continue to improve prediction accuracy. However, because the prediction accuracy of the model is already high, and the size of the model is already close to saturation, continuing to expand the scale will make the cost of computation more expensive. Therefore, in the future work, using other more complicated data, such as image data or other noisy, unprocessed data is a direction that can be considered, these data can provide more space to continue researching on optimisations on neural networks. Moreover, there are some optimisation methods for neural networks that are not covered in this paper, such as pruning and cascading network construction can be considered.

References

1. Slate, D.: UCI Machine Learning Repository: Letter Recognition Data Set. `https://archive.ics.uci.edu/ml/datasets/letter+recognition` (2018)
2. Gedeon, T.D., Bowden, T.G.: Heuristic Pattern Reduction, International Joint Conf. on Neural Networks, Beijing, 2, 449-453. (1992)
3. Pytorch Documentation, `https://pytorch.org/docs/stable/index.html#` (2018)
4. Kingma, D. P., Ba, J.: Adam: A Method for Stochastic Optimization. CoRR, abs/1412.6980. `http://arxiv.org/abs/1412.6980` (2014)
5. Frey, P., Slate, D.: Letter recognition using Holland-style adaptive classifiers. Machine Learning, 6(2), 161-182. `http://dx.doi.org/10.1007/bf00114162` (1991)