Artificial Neuron Network Hyperparameter Tuning by Evolutionary Algorithm and Pruning Technique

Hongbo Zhang

College of Engineering and Computer Science, Australian National University, Canberra, 2601, Australia
u6170245@anu.edu.au

Abstract. The hyperparameter tuning is a difficult problem in neuron network, especially in deep learning. In this work, evolutionary algorithm is used to tune the hyperparameters in an artificial neuron network. The results show that the evolutionary algorithm can give optimal hyperparameters, lead to a better prediction accuracy and reduce the overfitting. However, this method also suffers from some drawbacks. It is computational expensive, is not suitable for problem with stochastic nature and does not gaurantee to give a neuron network small in size. The last drawback can be overcome by pruning technique.

Keywords: neuron network · hyperparameter tuning · pruning · evolutionary algorithm.

1 Introduction

The deep neuron network (DNN) becomes main stream of artificial intelligence in recent years. However, it is very difficult to be trained for many reasons. One of the key reasons is that the hyperparameter selection is difficult and not obvious at all. Since in a typical DNN, such as Alexnet [10], it has more than 60 millions model parameters and several dozens of hyperparameters, including filter size, number of filters, strides, number of layers, dropout rate, learning rate, mini-batch size and so on. For such high dimensional hyperparameter space and such complex model, seaching for optimal hyperparameters usually become non-trivial.

Classical optimization methods usually involve first order or second order gradient and deploy gradient descent to find extreme points. However, for such complex and high dimensional optimization space, it will be usually trapped into local extreme points, rather than global extreme point. Consequently, more advanced hyperparameter tuning techniques are required. There are already some well-established techniques, including configuration selection approaches [4], adaptive configuration evaluation approaches [2], hybrid Bayesian optimization approaches [1], Gaussian process [11], Bayesian deep neuron network [8], their parallel counterparts and so on.

In this work, we will apply the evolutionary algorithm (EA) to hyperparameter tuning problem. In our artificial neuron network (ANN) with two fully connected layers, we select 7 hyperparameters to tune, including number of epoch, mini-batch size, learning rate, beta for moving average of gradient, beta for moving average of gradient square, weight decay and number of neurons in hidden layer. Among all the hyperparameters, number of hidden neurons plays a special role. Since other hyperparameters are all about the training algorithm, but number of hidden neuron is about the ANN structure itself. Consequently, we will pay more attention to this hyperparameter in this work. More details can be found in section 2.

Furthermore, pruning technique is also relevant to the number of hidden neurons. Therefore, we will also apply this techniques to study the effect of hidden neuron number. Especially, we will use distinctiveness approach [5] among other various pruning techniques, including pruning by relevance [12], by contribution [14], by sensitivitly [9], by badness [6] and so on, for its conceptual clearness.

In this work, we will use a simple all connected forward network with only one hidden layer. All units in hidden layer are connected to all neurons in input layer and output layer. There is no connection between input layer and output layer. The sigmoid non-linear activation function is used in both hidden layer and output layer. This simple network is used to study the cover type problem [3]. We train the neuron network by back-propagation [13]. Firstly, we deploy evolutionary algorithm to tune all 7 hyperparameters. Then, with the optimal hyperparameters obtained in previous step, we train an ANN and obatin the model parameters. After that, we apply pruning technique to the trained model to further reduce the size of hidden layer. Finally, we compare the results obtained above to previous work.

The structure of the paper is organised as following, Methods used in this work, including dataset preprocessing, details of evolutionary algorithm, and pruning by distinctiveness, is introduced in section 2. The results of this work, including hyperparameter tuning and pruning is shown in section 3. Discussion on the performance and issues of applying evolutionary algorithm to the problem hyperparameter tuning can be found in section 4. Finally, it is followed by a conclusion and future work perspective in section 5.

All the code, experimental results and various figures can be found in the supplimentary materials shipping with this paper.

2 Methods

Before discussing the methods in details, we should make clear some terminology used in this work.

Model parameter The model parameters refers to the parameters to be trained in ANN, such as filter value, weights, bias and so on. In this work, it is the weights and bias.

Hyperparameter The hyperparameter is the parameters of ANN architecture and parameters of training algorithms, such as learning rate, momentum, size of minibatch, filter size, stride, dropout rate, number of hidden neurons and so on. In this work, we will select 7 hyperparameters, including number of epoch (EP), mini-batch size (BS), learning rate (LR), beta for moving average of gradient (BG), beta for moving average of gradient square (BG2), weight decay (WD) and number of neurons in hidden layer (NH).

Metaparameter The metaparameter is the parameters used in EA, such as population size, number of generation, chromosome length, crossover rate, mutation rate, dynamical crossover coefficient and so on. We name it metaparameter to distinguish from hyperparameter in ANN. In this work, we only focus on 4 metaparameters, including population size (PS), number of generation (NG), crossover rate (CR) and mutation rate (MR).

2.1 Dataset and Data Preprocessing

Dataset Description We select cover type dataset contributed by Blackard and Dennis [3]. The dataset can be downloaded at UCI's website [1].

This data set include 54 features and 7 categories. It is about to predict the plant cover type of an small area give the area's elevation, aspect, slope, distances to hydrology/roadways/firepoint, hill shade, soil type and a given wildness region.

We choose this dataset since it's relative large in size among all dataset available at UCI which will facilitate learning process. Furthermore, it contains complex input feature, which provide lots of space for encoding scheme. It contains rich feature type, including continuous, cyclic and categories type.

The statistical overview of the data set can be found at UCI's website, or in Blackard's paper [3], and hence skiped here.

Dataset Encoding A complex and sophisticated input feature encoding scheme is discussed in our previous work [7]. However, in that work, the author showed that a more sophisticated encoding scheme will not improve the prediction of ANN. Furthermore, the scope of this work is to study the hyperparameter tuning problem, rather than input feature encoding problem. Consequently, here we adopt the most naive preprocessing method, in which we linearly squash all the input feature to the range $[0, 1]$. In this case, all the input data will fall in the sensitive range of sigmoid function and make the learning more efficiency. In the linearly squash, we don't consider the variance of input data and the nature of input data, e.g., cyclic, indexed, categories or continous. The target category is encoded by an integer ranging from 0 to 6.

ANN Dataset: Training Set, Validation Set and Testing Set The *covtype* dataset contains 581012 patterns. We randomly choose 290506 of them (50%) as training set, 145253 (25%) as testing set and 145253 (25%) as validation set.

I have confirmed that the targets in these sets have nearly indentical distribution by plotting their histogram.

The training set is used in training the neuron network.

The testing set is used to tell the performance of a trained model.

The validation set is used in determining the hyperparameters. For example, if we want to know which learning rate should be used in back-propagation, we will perform serveral experiments with different learning rate, and select the one gives best accuracy in validation set. Consequently, validation set is ***not*** used to tell performance of a trained model. It is different from a testing set, and is a part of training by itself. Especially, in this work, the accuracy of validation set is used as the fitness function in EA.

EA Dataset: Training Set and Validation Set In principle, ANN dataset alone is enough for this work. However, due to the heavy computational load required by EA, using ANN dataset (which use 100% of all *covtype*) in EA training is very expensive and beyond the scope of my computational power. In fact, training by EA with full dataset will cost more than 1 day for a single EA run with 20 generations and 20 populations.

Consequently, we will use a smaller dataset, EA dataset, which is a subset of ANN dataset in EA training to determine the metaparameters. To be specifically, we randomly select 29050 patterns (5% of total) from ANN

[1] https://archive.ics.uci.edu/ml/datasets/Covertype

training set to form the EA training set, and randomly select 14525 data (2.5% of total) from EA validation set to form the EA validation set. Notice that we don't require the testing set in EA training.

Therefore, when we determines the metaparameter and hyperparameter, we will use the smaller dataset (EA dataset). When we already know the hyperparameter from EA training, we will use the full dataset (ANN dataset) to estimate the actually prediction power of the ANN with given hyperparameters.

2.2 ANN and Its Hyperparameters

A simple all connected forward network with only one hidden layer is used in this work. All units in hidden layer are connected to all neurons in input layer and output layer. There is no connection between input layer and output layer. The sigmoid non-linear activation function is used in both hidden layer and output layer. We use cross entropy loss function and ADAM optimizer. The ANN is trained by back-propagation and mini-batch gradient descent method.

We will select 7 hyperparameters. They are [EP, BS, LR, BG, BG2, WD, NH], which represent the number of epoch (EP), mini-batch size (BS), learning rate (LR), beta for moving average of gradient (BG), beta for moving average of gradient square (BG2), weight decay (WD) and number of neurons in hidden layer (NH), respectively.

2.3 Evolutionary Algorithm to Tune Hyperparameters

Binary Representation Due to the limitation of computational power, I can't use a more flexible binary encoding of chromosome to represent the 7 hyperparameters.

As a result of trading off between the limited computational time and size of searching space, we have to discrete the searching space by grid, and encode them with a limited number of bits.

The bits used to encode different hyperparameters are shown in the Tab. 1. Hence, we will use a chromosome with 21 bits to represent a set of hyperparameters, without much flexible. With 21 bits, if one uses brute-force searching over all the $2^{21} \sim 2 \times 10^6$ possiblities, and under a conservative assumption that training each ANN will cost 10 seconds, it will cost 242 days, which is infeasible.

Table 1. The number of bits used to encode different hyperparameters.

hyperparameter	EP	BS	LR	BG	BG2	WD	NH
number of bits	4	3	3	2	2	3	4

EP: 4 bits. From previous study [7], EP = 20 is already coverged for *covtype* problem. Hence we only take 16 possible values, from 10 to 40, with step size of 2. So the decimal value of EP can be calculated from binary representation by

$$EP = 20 + \text{binaryToInteger}(\text{4 bits binary})$$

Due to the computational cost, we did not use more bits encoding scheme. Since in the random searching in EA, the large EP will cost so much computational time (typically 10 mins on my laptop for training one two layer fully connected ANN), especially when the batch size is small.

BS: 3 bits. From previous study [7], BS is optimal around 300. With this knowledge, we will grid the BS by 8 possible values: $[100, 300, 600, 900, 1200, 1800, 2400, 3000]$. So the decimal value of BS can be calculated from binary representation by

$$BS = BS\_Array[\text{binaryToInteger}(\text{4 bits binary})]$$

Furthermore, the grid starts at 100, since a too small batch size will cost so much computation time. And the grid ends at 3000, since a too large batch size will lose the power of mini-batch learning. Indeed, from previous study [7], when the batch size is around 30'000, the prediction ability reduce by more than 10%. Again, it is trade-off between computational cost and greater searching space.

LR: 3 bits. From previous study [7], LR is optimal around 0.01, so we select learning rate from 8 possible values: $[0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5]$. The decimal value of LR can be obtained from its binary representation by a formula similar to that of BS.

BG: 2 bits. Since we have no priori knowledge of optimal BG, we select beta for computing moving average of gradient from following 4 values: $[0.99.0.9, 0.8, 0.5]$, which represents averaging the gradient over most recent $[100, 10, 5, 2]$ values, respectively. The decimal value of BG can be obtained from its binary representation by a formula similar to that of BS.

BG2: 2 bits. Since we also have no priori knowledge of optimal BG2 as well, We select beta for computing moving average of gradient square from following 4 values: $[0.999, 0.99, 0.9, 0.8]$, which represents averaging the gradient square over most recent $[1000, 100, 10, 5]$ values, respectively. The decimal value of BG2 can be obtained from its binary representation by a formula similar to that of BS.

WD: 3 bits. Similar to BG and BG2, there is no priori knowledge about this parameter, so we make a very coarse search over a large range, namely $[0, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1]$ The decimal value of WD can be obtained from its binary representation by a formula similar to that of BS.

NH: 4 bits. Since we are interested in this hyperparameter and will compare it with that obtained from the approach of purning, therefore, we will make it a fine grid which spans over a large range of values. To be more specifically, NH will range from 10 to 160 with a step size of 10. So the decimal value of NH can be calculated from binary representation by

$$NH = 10 + \text{binaryToInteger}(4 \text{ bits binary})$$

Fitness Function For each given chromosome with 21 bits long, we can decode it to get a set of 7 hyperparameters from the encoding scheme described in previous sub-sub-section. With the set of hyperparameters, we can train a two layer fully connected ANN with 54 input neurons and 7 output neurons on EA training set (5% of total data), and obtain a trained ANN model. Using this ANN model, we can make prediction on the EA validation set (2.5% of total data). Therefore, we can calculate the accuracy of validation set from the prediction. The accuracy is taken as the target function in EA.

We use a naive definition of fitness function, which is simply substract the accuracy by the minimum accuracy in a population and add a small positive real number to make sure every term is non-zero positive. This non-zero positive function will be used in the selection later.

Selection In selection step, we use the bias selection. For each chromosome in a population, it can be selected as parent of next generation with following probability

$$p(i) = \frac{f(i)}{\sum_i^{\text{POPSIZE}} f(i)}$$

where f is fitness of each chromosome.

Crossover and Mutation In reproducing step, we use a uniform crossover algorithm, in which each chromosome has a probability of CR (crossover rate) to crossover with another randomly chosen chromosome with uniform probability. Each point in a chromosome has $0.5 \times$ CR to crossover with the corresponding point on the another randomly chosen chromosome

We use the most naive static mutation algorithm, in which each point on each chromosome will have a constant probability of MR (mutation rate) to flip its bit value. The probability MR will not evolve with time.

Training EA In each generation, the fitness function is calculated for each chromosome in the generation, and according to the fitness function, we select some chromosomes to be the parents of the next generation. After crossover and mutation, we get next generation. This process is looped until termination criterion is satisfied.

All POPSIZE chromosomes in the initial generation is generated randomly, so that they are distributed across the hyperparameter space uniformly.

Hall of Fame. The best chromosome, who has largest accuracy, is recorded to Hall of Fame.

Termination Criterion. In this work, the most naive termination criterion is used. The evolution will be terminated after a fixed number of generation.

Optimal Hyperparameter. After the termination of EA training, the chromosome in Hall of Fame with largest accuracy will be chosen as the optimal chromosome, and its corresponding hyperparameters will be chosen as the optimal hyperparameters.

2.4 Metaparameter Selection and EA Learning Curve

In this work, we only focus on 4 metaparameters, including population size (PS), number of generation (NG), crossover rate (CR) and mutation rate (MR).

Since the limited by computational power, we cannot make an extensive search over the metaparameter space. Therefore, we will use some well-known guidelines to reduce the amount of searching.

Population Size and Number of Generation In order to select an appropriate population size and number of generation which balance the computational cost and searching more hyperparameter space, we run the EA training with two different set of metaparameters.

In both sets, we fix the crossover rate to be 0.8 and mutation rate to be 0.05, the number of generation to be 50. In the first set of metaparameter, we take the population size as 10, while in the second set of metaparameter, we take the population size as 20.

The learning curve of these two EA training is shown in Fig 1. In the learning curve, we plot the mean accuracy and the maximum accuracy of each generation.

For both learning curves, there are lots of fluctuations and large variance. This is due to the large crossover rate and mutation rate we selected, as well as the stochastic nature of ANN.

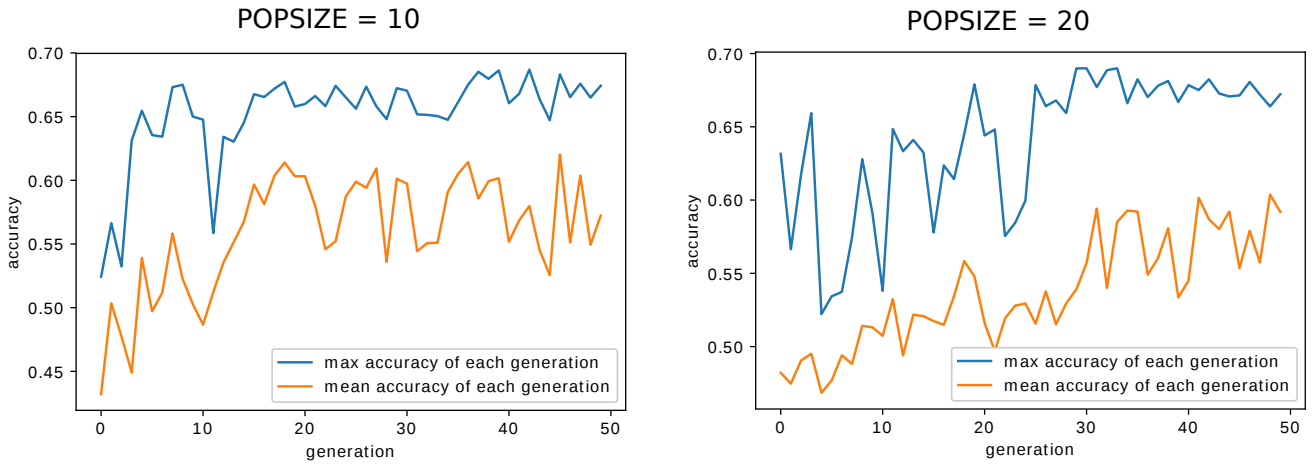

Fig. 1. Two EA training with population size 10 and 20, respectively. For both training, the crossover rate, mutation rate and number of generation are taken as 0.8, 0.05 and 50, respectively.

The case of population size being 10 is on the left side of Fig. 1. From it we can read that although the maximum accuracy curve still increases on average after 20 generation, however, its increasing rate is very low. As a trade-off between the computational cost and searching power, we can regard that the curves are converged after 20 generations.

The case of population size being 20 is on the right side of Fig. 1. From it we can read that the maximum accuracy curve will converged after 30 generation. This training is very expensive, it takes more than 4 hours on a laptop with i5 CPU.

Furthermore, the largest accuracy found in the case of population size = 10 is 68.7%, while that found in the case of population size = 20 is 69.0%, which are almost the same. Again, for the sake of computational cost, we regard that population size = 10 and number of generation = 20 is enough for the EA to find the optimal hyperparameters.

In the remaining part of this work, we will set population size = 10 and number of generation = 20. From a general guidance, this population size and number of generation is too small. However, this set of value balance the computational cost and searching power.

Mutation Rate. We take the mutation rate as the inverse of chromosome length, $1/L$. Since in our encoding scheme, there are 21 bits in a chromosome, as a result, we take MR = 0.05. It means for each chromosome, one bit will be mutated on average.

Crossover Rate. In order to find a good crossover rate, we perform three experiments with different value of crossover rate (CR), CR = $[0.1, 0.50.8]$.

The results of experiments are shown in Tab 2. From the results, the large crossover rate will give a better results. That is because we only run a very small amount of generation, and a large crossover rate which favors exploration will be more efficiency in searching the hyperparameter space in this case.

Actually, with smaller CR, it will require more generations for the learning curves to be converged, which also increase the computational cost.

In the remaining work, we will choose CR = 0.8 to facilitate exploration and get a better result.

2.5 Distinctiveness Pruning

We adopt the distinctiveness pruning approach [5], since its conceptual clearness.

Table 2. The accuracy obtained by three EA training with different crossover rate CR = [0.1, 0.50.8]. The other metaparameters are population size = 10, number of generation = 20, mutation rate = 0.05.

crossover rate	0.1	0.5	0.8
best accuracy	65.6%	62.9%	68.4%

In this approach, we first normalise the output of each hidden neuron to the range $[-0.5, 0.5]$ and represent it in pattern space, i,e, in a vector with each element is the hidden neuron output for a certain input pattern. Then we calculate the angle between neuron pairs. Using this angle, we can tell whether the hidden neuron pair is similar to or complementary to each other.

There are many pruning categories associating with distinctiveness. However, in this work, we only focus on the following three categories.

Neurons with Constant Output. In the pattern space representation, a hidden neuron's output is a vector with dimension of pattern number. If all the elements in the output vector have the same value (within a given thresholds, of course), then this hidden neuron can be safely removed. For compensating its removal, we add its weight to the bias. Upon removal of this neuron, we also remove the bias associated with it in input layer.

Similarity Neuron Pairs. If the output vector of two hidden neurons are similar to each other (angle is smaller than a thresholds), then these two neurons have similar funcationality, hence one of them can be removed. For compensating its removal, we add its weight to its couterpart in the pair. Upon removal of this neuron pair, we also remove the bias associated with it in input layer.

Complementary Neuron Paris. If the output vector of two hidden neurons are complementary to each other (angle is larger than a thresholds), then these two neurons have similar funcationality, but in opposite phase. They cancel the other when are conveyed to next layer. Hence, both of them can be safely removed, and no weight/bias adjustment in this layer is required. Upon removal of this neuron, we also remove the bias associated with it in input layer.

3 Results

From subsection 2.4 Metaparameter Selection, we determine a good set of metaparameter by serveral experiments. With this set of metaparameters, we can run the EA with EA dataset (training set is 5% of total data, and validation set is 2.5% of total data), and obtain the optimal hyperparameters, which are shown in Tab 3.

Table 3. The optimal hyperparameters obtained by evolutionary algorithm.

hyperparameter	EP	LR	BG	BG2	WD	BS	NH
optimal value	30	0.05	0.99	0.999	10^{-6}	100	120

In the following, we will use this set of hyperparameter.

Train an ANN and Prediction of ANN on Full Dataset We train an ANN with above hyperparameters on ANN dataset (full dataset). The learning curve of ANN training is shown in Fig. 2. As it shows, the learning is already converged.

The prediction accuracy on the ANN testing set is 65.7%, and on the ANN validation set is 66.1%, which is even lower than the prediction accuracy of EA dataset (using only 5% data rather than full data), which is 68.4%. That is because we use EA dataset in evolutionary algorithm to determine the optimal hyperparameters. Obviously, from our result, the hyperparameter obtained in this way is the optimal ones for EA dataset (5% of all data), rather than ANN dataset (full dataset).

However, it is too expensive to use full data in hyperparameter tuning by EA.

Pruning the Trained ANN The number of hidden layer determined by EA is 120. However, we can perform pruning to reduce it.

The result of pruning and its comparison with the original network is shown in Tab. 4

It was surprised to find out 85% of hidden neuron can be pruned out without any affection on the prediction accuracies. Especially, there are 84 constant neurons which are pruned out. Obviously, according to our result, although NH = 120 is a good one, it is not necessary the smallest one. Since the EA algorithm only guarantees that the optimal hyperparameter is found with enough generations and population size. It doesn't guarantee that the found optimal hyperparameter is small or large at all.

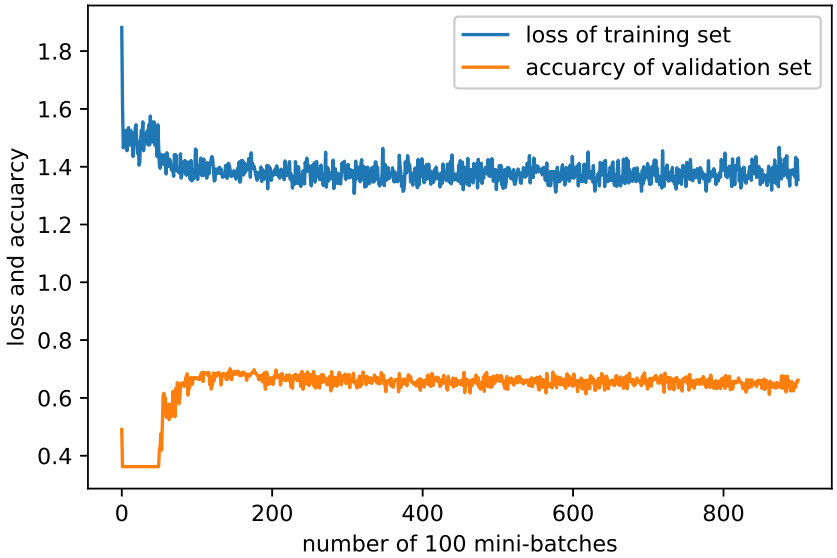

Fig. 2. Learning curve of ANN training.

Table 4. Comparison between Pruned and Original Network.

	Full Net	Pruned
training accuracy	65.7%	66.5%
validation accuracy	66.1%	66.8%
testing accuracy	65.7%	66.5%
hidden neurons num	120	18

If we want to take the largeness of hyperparameter into account, we should modify the fitness function, so that the norm of hyperparameters is included.

Comparison between This Work and My Previous Work In the previous work [7], we selected the hyperparameters by running a small number of ANN with different hyperparameters.

The hyperparameters used in [7] shown in Tab 5. With this set of hyperparameters, the accuracy of training set is 67.5%, the accuracy of validation set is 71.1%, the accuracy of testing set is 63.6% [7]. Obviously, the training set accuracy and validation set accuracy of previous work are greater than those of this work (65.7% and 66.1%), however, the testing accuracy is smaller (65.7% in this work). This means the hyperparameters selected by EA suffers ***less overfitting*** than that in previous work by the most naive method.

Table 5. The optimal hyperparameters used in previous work [7].

hyperparameter	EP	LR	BG	BG2	WD	BS	NH
value used in [7]	20	0.01	0.9	0.999	0	290	100

Comparison between This Work and Other's Work Blackard and his collabrators [3] also studies the covertype dataset in 2000. In their work, the prediction accuracy of two layer ANN is 70.58%.

They trained their network with momentum gradient descent method with a MSE loss function. They tried several combination of number of input features, number of hidden neurons, learning rate and momentum rate. From these experiments, they claimed that 54 (all) input features, 120 hidden neuron, learning rate 0.05 and momentum 0.5 is the optimal for the covtype problem. With these settings, the accuracy of testing data is 70.58%.

The accuracy of testing data is 3.8% higher than that in this work. However, due to the limitation on computational power, we tune the hyperparameters by EA on a smaller dataset (5% of total data), however, if we have enough computational power and apply EA on the full dataset, the resulting performance will be improved.

4 Discussion

As shown in this work, one can indeed obtain a good set of hyperparameters by evolutionary algorithm. Furthermore, the hyperparameter selected by EA algorithm will suffer from less overfitting.

However, this method also suffers lots of drawbacks.

The most obviously one is its expense. Since EA is essentially a random algorithm to search the hyperparameter space, similar to other random algorithm, it usually converges slowly, hence cost lots of computational time. Especially, the ANN is expensive by iteself. Due to its expensiveness, we have to use a much smaller sub-dataset in EA, nevertheless, the hyperparameter obtained in this way is only optimal ones for that smaller sub-dataset, rather than the full dataset. When we apply this set of hyperparameter to the full dataset, it will give worse prediction.

Another drawback is that the ANN itself is a stochastic one. Even for the same hyperparameters and the same input patterns, one will get different results for different training, since the initial value of model parameters are assigned randomly. Furthermore, the mini-batch training makes the ANN more stochastic. If we watch carefully enough, we can find out that the EA learning curve fluctuates and has a large variation. In this case, the hyperparameter space will be highly degenerated due to the fluctuation. In another word, the error bar of accuracy predicted by the model with different hyperparameter will be overlapped with each other. In this sense, many different hyperparameter combinations will give almost same accuracy. In this case, it seems that it is non-sense to make such expensive search.

Furthermore, the EA algorithm doesn't gaurantee the found optimal hyperparameter is small/large in value. For example, the NH (number of hidden neuron) found by EA algorithm is 120, which has lots of redundant. Consequently, we can use other techniques, such as pruning, to reduce this hyperparameter to make the neuron network more efficient.

5 Future Work and Conclusion

Future Work. In this work, only the most naive evolutionary algorithm is applied to the hyperparameter tuning problem, and shows its power on this problem. However, there are still lots of more advanced evolutionary algorithm to explore, such as improving the termination criterion, the selection algorithm, the crossover and mutation algorithm, using tournament training and so on.

Conclusion In this work, we apply the EA to solve the problem of hyperparameter tuning. The results show that one can indeed obtain a good hyperparameter combination with evolutionary algorithm. It gives a good prediction on accuracy and reduce overfitting. However, it also suffers from some problem, including too expensive, not suitable to the problem with too much noise (like ANN trained by mini-batch.).

References

1. A, Klein S, F., et al: "fast bayesian optimization of machine learning hyperparameters on large datasets". In: AISTATS (2017)
2. A, G., L, K.: "efficient multi-start strategies for local search algorithms". In: JAIR. vol. 41 (2011)
3. Blackard, J.A., Dean., D.J.: "comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables.". In: Computers and Electronics in Agriculture. pp. 131–151. 24(3) (2000)
4. F, Hutter H, H., K, L.B.: sequential model-based optimization for general algorithm configuration. In: Proc. of LION-5 (2011)
5. Gedeon, T., Harris, D.: network reduction techniques,. In: Proc. Int. Conf. on Neural Networks Methodologies and Applications. pp. 119–126. vol. 1 (1991)
6. Hagiwara, M.: novel back propagation algorithm for reduction of hidden units and acceleration of convergence using artificial selection. In: IJCNN. pp. 625–630. vol. 1 (1990)
7. HB, Z.: "neuron network reduction by input feature encoding and hidden neuron pruning". In: ABCs2018 (2018)
8. J, Springenberg A, K., et al: "bayesian optimization with robust bayesian neural networks". In: NIPS (2016)
9. Karnin, E.: a simple procedure for pruning back-propagation trained neural networks,. In: IEEE Transactions on Neural Networks. pp. 239–242. vol. 1 (1990)
10. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1. pp. 1097–1105. NIPS'12, Curran Associates Inc., USA (2012), http://dl.acm.org/citation.cfm?id=2999134.2999257
11. Lzaro-Gredilla J, Q.C., et al: "sparse spectrum gaussian process regression". In: JMLR (2010)
12. Mozer, M., Smolenski, P.: using relevance to reduce network size automatically,. In: Connection Science. pp. 3–16. vol. 1 (1989)
13. Rumelhart, DE Hinton, G., Williams, R.: learning internal representations by error propagation,. In: Parallel distributed processing. vol. 1 (1986)
14. Sanger, D.: contribution analysis: a technique for assigning responsibilities to hidden units in connectionist networks. In: Connection Science. pp. 115–138. vol. 1 (1989)