The Genetic Algorithm and Network Reduction in Classification of Directions in Automatic Navigation of Wall-following Mobile Robot Using a Feed-forward Neuron Network

Jing Li

Research School of Computer Science, Australian National University

U6013763@anu.edu.au

Abstract. This paper focuses on the classification of directions in the automatic navigation of wall-following mobile robot, using a feed-forward neuron network of three layers trained by back-propagation. We utilize a relevant data set from UCI to do training and testing. We first utilize the genetic algorithm to select the most relevant features to optimize the performance of the classifier. Then we apply a network reduction technique to improve the network. The experiments demonstrate that the testing accuracy is over 85% while both consumed time and space are reduced. The results are better than results from another paper citing the same data set. Also, we show and analyze why different optimizers perform differently on this data set.

Keywords: Wall-following, mobile robot, genetic algorithm, network reduction, optimizer

1 Introduction

With the booming development and massive popularity of artificial intelligence (AI) techniques, mobile robot has become a promising technology which holds its importance in military, industry, and building of smart cities/homes. Global giants such as Amazon and Google find the business opportunities and are researching mobile robots, especially, autonomous mobile robots which mean they are capable of navigating in an uncontrolled environment without the need for physical or electro-mechanical guidance devices [1]. It is well-accepted that mobile robots are not static or stay still in some places, they keep moving for the purpose of performing the assigned tasks. However, due to complex terrain and varied topography in the real world, it is difficult for a mobile robot to make an optimal or correct decision in many situations. As a consequence, mobile robots are required to "learn" which exact direction to take as well as decide related future movements in real time. In reality, it is usually obstacles and walls that make the environments complex and hinder the continuous movements of mobile robots. In order to cope with the blocks, mobile robots take advantages of different kinds of sensors to calculate the distance from the obstacles or walls to its body from different angles. During navigation, the mobile robot maintains a safe distance from walls of obstacles, and these types of robots are called "wall following robots" or "wall followers" [2].

To help mobile robots "learn", we apply a feed-forward neuron network of three layers trained by back-propagation. However, it may exist some irrelevant and redundant features from a given dataset which do not contribute or decrease the accuracy of the classier [3]. And the performance of the classifier is sensitive to the choice of the features used to construct the classifier [4]. An exhaustive selection of features evaluates all combinations of features which require a large amount of computational work [3]. The genetic algorithm offers an attractive approach to find the most relevant features to represent the patterns to be classified based on the mechanics of natural genetics and biological evolution.

In addition, when constructing the neuron network, we are not able to determine the exact number of hidden units. Many works have demonstrated that we usually need more hidden neurons than required to gain a better performance in such neuron networks. Nevertheless, once the neuron network is trained successfully, we still want to remove excess hidden units which are meaningless.

In this work, we first apply genetic algorithms to optimize the performance of the neuron network by selecting the most relevant features. Then we remove unnecessary hidden units by comparing their distinctiveness [5]. Thus the neuron network is improved and able to acquire better performance in terms of both consumed time and space.

1.1 Problem statement

Given a set of distance information calculated by sensors from different angles, we should make the mobile robot automatically decide which direction to take. Input is the distance information calculated by sensors from different angles, while the output is the predicted next direction.

1.2 Data set

In this work, we choose Wall-Following Robot Navigation Data Set from UCI Machine Learning Repository [5]. This data set contains the raw values of the measurements of all 24 ultrasound sensors and the corresponding class label, i.e., which direction to move. Note that 24 ultrasound sensors are situated circularly around the waist of the robot. E.g., attribute US1 means distance information measured by ultrasound sensor at the front of the robot, while attribute US13 means distance information measured ultrasound sensor at the back of the robot.

We choose this data set for the following reasons:

– This data set is relatively complex with 5456 instances, 24 attributes (features) and 4 classes.
– In each attribute columns, the distance information is represented by real numbers without the hassle of tedious preprocessing of data set.
– There is no missing value in this dataset, so each instance is valid.
– Each class has a reasonable proportion of whole data set, as shown in table 1.

Table 1. Class distribution in sensor readings dataset

Direction	Number of samples	Proportion (%)
Move-Forward	2205	40.41%
Sharp-Right-Turn	2097	38.43%
Slight-Right-Turn	826	15.14%
Slight-Left-Turn	328	6.01%

1.3 Outline of the investigations

We utilize a feed-forward neuron network of three layers trained by back-propagation upon the chosen data set. First, we use historical loss, confusion matrix for training and testing to check whether the built neuron network performs well.

In the second place, we apply the genetic algorithm to select most relevant features. To check whether the genetic algorithm works in this problem, we compare the testing accuracies of using all features and using the selected features. In addition, we investigate the impact of the population size. Then we compare the training time with and without the genetic algorithm to see whether the training time is reduced.

Thirdly, we employ network reduction technique to improve the network. To check how the network is improved, we compare testing accuracies and confusion matrices before and after reduction. Since we do network reduction to prune useless units, it is taken for granted that the computing space is reduced. We compare testing time before and after reduction to see whether the consumed time is reduced. Then, we analyze the different performance of different optimizers on this data set, and try to explain why.

2 Methods

2.1 Data loading and processing

In the first place, we use data loader to read data. As mentioned in section 1.2, The values of 24 attributes are real numbers while the values of classes are strings, such as "Move-Forward". As a result, we need to do data processing and convert class targets from string values to numeric values, e.g., set "Move-Forward" as 1. Then we randomly split the whole data set into a training set (80%) and a testing set (20%). Because we usually need more data to do training.

2.2 Define and train a neural network

We build a neural network with one hidden layer. In the input layer, we set 24 neurons, representing the features of data set. In the output layer, we set 4 neurons, representing the classes. In the hidden layer, we set 50 neurons, using Sigmoid as the activation function.

In order to decide the number of hidden neurons, there are some empirically-derived rules-of-thumb, of these, the most commonly relied on is "the optimal size of the hidden layer is usually between the size of the input and size of the output layers" [7]. As a result, the optimal number of hidden neurons may be between 4 and 24. Consequently, we set the number of hidden neurons as 24.

We set batch size as the sample size. it may take a bit longer to compute the gradient for each step, but the training signal is less noisy.

We set learning rate as 0.1. Because after testing, we can find not only training is reliable, but also the final result converges.

We set the number of epochs as 500. Because after testing, we can find when we train neuron network with more than 500 epochs, the training accuracy increases very slowly. Also, when the number of epochs is 500, both training and testing accuracies are about 90%, thus no over-fitting exists.

The network will be trained with Adagrad algorithm as an optimizer, which dynamically incorporates knowledge of the geometry of the data observed in earlier iterations to perform more informative gradient-based learning [8]. We choose Adagrad as the optimizer, because after many testings, optimizer Adagrad performs best, and is most suitable for this data set. With optimizer Adagrad, the testing accuracies before and after reduction both approach 90%. By contrast, with optimizer RMSprop, we gain testing accuracies of less than 40% before and after reduction. In addition, with optimizer Rprop, although the testing accuracy before reduction can even hit 95%, the testing accuracy after reduction is decreased to about 65%. This part will be discussed in details in section 3.2.

We use the nn package to define our model as a sequence of layers. nn.Sequential is a module which contains other modules, and applies them in sequence to produce its output. Each linear module computes the output from the input using a linear function, and holds internal Variables for its weight and bias.

The performance of training will be evaluated using historical loss, and confusion matrix for training.

2.3 Feature Selection

We apply the genetic algorithm to do feature selection. We first randomly initialize the individuals in the population. To evaluate the fitness, we need to train the classifier with the training data of the selected features, and then evaluate the testing accuracy with the testing data of the selected features. According to the fitness, we select the individuals to do crossover to produce offspring. Then mutation is applied to introduce new genetic material into an existing individual.

Especially, to get the fitness of each individual (i.e., testing accuracy), we set the number of training epoch as 1 to save time. Nevertheless, with the final selected features after all generations, we retrain the classifier with 500 training epochs to get the accurate testing accuracy.

We set population size as 100 and number of generation as 10, because many experiments show that we can get a relatively high testing accuracy within a reasonable time with such settings.

We set the crossover rate as 0.8, because it should be a relatively high probability that couples will be picked for mating.

We set mutation rate as 0.002 which is low, because mutation may cause bad results.

2.4 Network Reduction

We do network reduction based on distinctiveness between hidden units. The distinctiveness of hidden units is determined from the unit output activation vector over the pattern presentation set. That is, for each hidden unit we construct a vector of the same dimensionality as the number of patterns in the training set, each component of the vector corresponding to the output activation of the unit [5]. This vector represents the functionality of the hidden unit in (input) pattern space. To normalize vector angle calculations, we move the origin to (0.5, 0.5).

Then we determine the similarity between two vectors (i.e., two hidden neurons) by calculating the angle between two vectors. If the obtained angle is less than $15°$, we can say these two neurons are sufficiently similar As a result, one of them needs to be removed. After that we add the weight vector of the moved unit to the weight vector of the unit which remains. If the obtained angle is larger than $165°$, we can say these two neurons have complementary effects, so both of them needs to be removed. The above processing removes excess hidden units, and no retraining is demanded.

2.5 Network testing and result evaluation

In the first place, we use the genetic algorithm to select most relevant features, then compare the performance of the network using all features and using the selected features by genetic algorithm. The result demonstrates how genetic algorithm influences both testing accuracy and training time.

Secondly, we pass testing data to the built neural network both with and without reduction to compare testing accuracies. Then we compare the generated confusion matrices related to built network with or without reduction. At last, we compare their testing time. After reduction, if the testing accuracy is similar, or just decreases a little, while testing time is significantly reduced, we can say the applied network reduction technique is practical and successfully improves the neuron network in terms of consumed time and space.

3 Result and Discussion

3.1 Performance Evaluation of Genetic Algorithm

	First test	Second test	Third test
Testing accuracy without genetic algorithm (%)	88.24	88.28	88.12
Testing accuracy with genetic algorithm (%)	85.67	83.89	84.14

Table 2. Comparison with and without genetic algorithm

To evaluate the performance of the genetic algorithm, we conducted three experiments with the same settings and parameters because the results are not deterministic.

It can be seen that applying the genetic algorithm makes the testing accuracy decrease, i.e., the experiments show that the genetic algorithm does not work in this problem. It is because the genetic algorithm is used to remove irrelevant and useless features from the data set to improve accuracy. In this problem, the 24 features represent the measurements of all 24 ultrasound sensors situated circularly around the waist of the robot. It is clear that all of 24 features are important to decide the next direction of the robot. The number of features is 24 which is relatively small, and there are no irrelevant and useless features. Since the genetic algorithm is a stochastic method, it is hard to get the exact optimal solution, i.e., keeping all features.

On the contrast, in the design of segmenters of medical images or of remotely sensed aerial images, the initial set of candidate features often consists of over 100 features [9]. Such a large number of features often include many garbage features. The genetic algorithm plays an important role in improving classification accuracy in such scenario.

The Impact of Population Size Population size in genetic algorithms needs to be large enough to initialize with a rich set of solutions [8]. However, a large population size may lead to a large amount of computational work, and it is hard to decide the optimal population size. Figure 3.1 demonstrates the performance of the classifier using the genetic algorithm with the population size of 10, 20, 40, 60, 80 and 100. Other settings and parameters are fixed such as the number of generations.

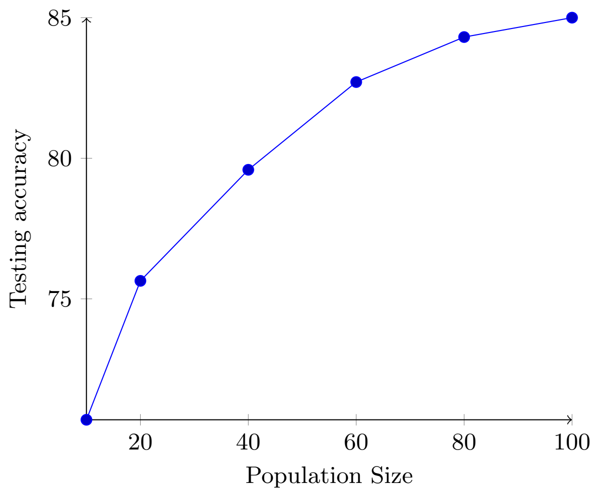

Fig. 1. Accuracy comparison

It can be seen that the higher testing accuracy is achieved with the larger population size. Since the number of generations is the same, with the greater population size, chance is greater that the population will contain a better feature selection. On the flip side, it is clear that when the population size is small, usually more generations are needed to get a better result.

Training time analysis The training time with and without the genetic algorithm is shown as below:

	First test	Second test	Third test
Training time without genetic algorithm (seconds)	63.6	66.7	62.5
Training time with genetic algorithm (seconds)	54.8	55.9	53.3

Table 3. Training time comparison

From the table, it can be seen that the training time is reduced because only a subset of features is utilized when using the genetic algorithm. Thus less training time is needed because the size of the training data set is reduced.

3.2 Performance Evaluation of Network Reduction

The Impact of Optimizer As mentioned in section 2.2, we compared all optimizers and found Adagrad most suitable for this data set. In this section, we take optimizer Adagrad, RMSprop and Rprop as examples to demonstrate their differences in performance. As the results are not deterministic, we do the analysis with three sets of testing statistics. In addition, we show the number of removed units in each testing. Recall that

	First test	Second test	Third test
Testing accuracy without reduction (%)	88.24	88.28	88.12
Testing accuracy with reduction (%)	88.05	87.64	85.59
Number of removed units	2	3	3

Table 4. Adagrad as optimizer

	First test	Second test	Third test
Testing accuracy without reduction (%)	39.43	39.98	39.89
Testing accuracy with reduction (%)	15.38	7.34	13.65
Number of removed units	23	23	23

Table 5. RMSprop as optimizer

	First test	Second test	Third test
Testing accuracy without reduction (%)	95.15	94.48	94.92
Testing accuracy with reduction (%)	62.57	78.75	74.89
Number of removed units	5	6	7

Table 6. Rprop as optimizer

we set the numbers of initial hidden units as 24. In each testing, other parts of the configuration are the same, such as the number of epochs, and learning rate.

From above tables, we can first find the common point that testing accuracy usually decreases after reduction. Then, usually the more we remove hidden units, the lower testing accuracy becomes.

By comparisons among above three tables, we are able to find reasons why optimizer Adagrad performs best while optimizer RMSprop performs worst. First, optimizer Adagrad only removes a few hidden units, which means, with optimizer Adagrad, functionalities of hidden units in input pattern space varies a lot. That is to say, only a few of hidden units have similar or complementary effects. On the contrary, with optimizer RMSprop, 49 out of 50 hidden units are pruned. This is only 3 testing sets, but we can say, with optimizer RMSprop, most hidden units work similarly or in an opposite way. This is also the reason why testing accuracy without reduction is not high with optimizer RMSprop in this data set.

Historical loss and confusion matrix analysis The historical loss and confusion matrix for training the network is plotted as below. We use optimizer Adagrad as the optimizer.

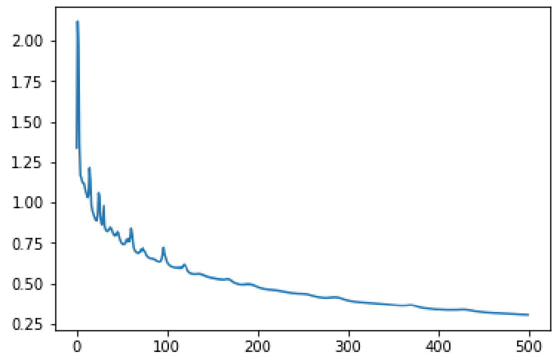

Fig. 2. Historical loss

$$\begin{bmatrix} 1541 & 73 & 161 & 14 \\ 86 & 604 & 7 & 0 \\ 63 & 7 & 1571 & 1 \\ 10 & 0 & 13 & 230 \end{bmatrix}$$

Fig. 3. Confusion matrix for training

We can find the training is reliable. The loss keeps decreasing as a whole, and finally the result converges. The spikes in the historical loss exist due to the local optimum. The confusion matrix also shows the result is relatively accurate.

The confusion matrices for testing with and without reduction is also shown as below.

$$\begin{bmatrix} 343 & 26 & 42 & 5 \\ 23 & 105 & 1 & 0 \\ 25 & 9 & 421 & 0 \\ 6 & 0 & 5 & 64 \end{bmatrix}$$

Confusion matrix for testing without reduction

$$\begin{bmatrix} 314 & 32 & 66 & 4 \\ 15 & 111 & 3 & 0 \\ 20 & 7 & 428 & 0 \\ 3 & 0 & 8 & 64 \end{bmatrix}$$

Confusion matrix for testing with reduction

Fig. 4. Confusion matrices

Combining these two matrices, it is obvious that network reduction affects testing accuracy at a very small scale.

Space and time reduction analysis Because we utilize a network reduction technique to prune some unnecessary hidden units. It is clear that the computing space is reduced.

Then the testing time with and without reduction is shown as below. In each testing, we set optimizer as Adagrad, and the configurations are the same.

	First test	Second test	Third test
Testing time without reduction (seconds)	0.001159	0.001136	0.001688
Testing time with reduction (seconds)	0.000777	0.000796	0.000842

Table 7. Testing time comparison

From above table, although each testing time is less than 0.002 seconds, we can still say testing time is reduced significantly.

3.3 Comparison with a published paper

As the genetic algorithm cannot improve the testing accuracy in this problem, we compare our results using network reduction with results in [10]. T. Dash. et al. proposes an attempt to control the navigation of a wall following mobile robot using a hybrid metaheuristic which is a combination of two different population-based heuristics such as Gravitational Search (GS) and Particle Swarm Optimization (PSO). They evaluate the performance of adopted Artificial Neuron Network (ANN) model based on accuracy and time required for the model to be trained and tested including cross-validation.

Note that they used three data sets: *sensor_readings_2.data*, *sensor_readings_4.data*, *sensor_readings_24.data*. We only make use of *sensor_readings_24.data* because the other two data sets have too few attributes. As a result, we only compare results with their results in *sensor_readings_24.data*.

In addition, their results are associated with not only different numbers of hidden neurons but also numbers of the population which are defined in their paper. For simplicity, we assume each of their results as a certain test regardless of numbers of hidden neurons and numbers of the population which is hard to calculate. There are 6 results in their paper, thus we collect 6 sets of testing statistics and plot their comparisons in terms of accuracy. In each testing, we set optimizer as Adagrad, and the configurations are the same.

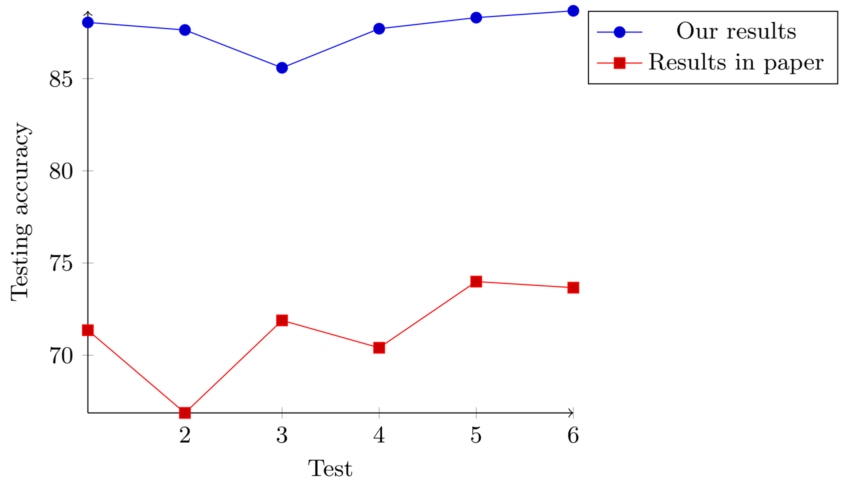

Fig. 5. Accuracy comparison

From above figure, we can find our results are always better than the results in the paper. It is because they propose a hybrid metaheuristic based algorithm to solve the problem. The algorithm is essentially a heuristic algorithm, which aims to find an approximate solution when classic methods fail to find any exact solution. In other words, the heuristic algorithm produces a solution in a reasonable time frame that is good enough for solving the problem, but not very accurate.

The needed simulation time in their paper is shown as below,

	1st test	2ed test	3rd test	4th test	5th test	6th test
Simulation time (seconds)	4647.7253	9246.5584	13869.0084	4647.7253	7784.9553	15423.8735

Table 8. Simulation time in the paper

As there are 5456 instances in this data set, it needs relatively more time to run. It can be seen that the simulation time in their paper is very long while our whole process of training, reduction, and testing only needs far less than 1 minute. It is because their algorithm has a relatively large time complexity.

4 Conclusion and Future Work

4.1 Conclusion

Using this data set, we analyzed historical loss, confusion matrix for training and testing, and found the built neuron network performed well and got a good result.

In this paper, we found that the genetic algorithm did not improve the testing accuracy in this problem because the number of features is relatively small and all of them are useful. However, the training time was reduced adopting the genetic algorithm. With the same number of generations, we found that greater population size usually lead to better classification accuracy.

We did the optimizer analysis, and found different choices of optimizers could lead to very different results and performance. Also if a neuron network consists of too many hidden units of similar or complementary functionaries, the network may perform badly.

We found that after reduction, accuracy usually decreases in a certain scale. Through testing on the data set, we proved the applied network reduction technique is practical and promising. In this data set, our result shows after reduction, accuracy only decreases a little and is over 85% while time complexity and space complexity are significantly reduced.

By comparison with the results from another paper, we found that using a neuron work to train on a training set, then predicting the actual results may perform better than a specific algorithm, in terms of both accuracy and consumed time.

4.2 Future work

In this paper, since the impact of the population size in the genetic algorithm was investigated, we would like to do researches on how to decide the settings and parameters of genetic algorithms such as the crossover rate and mutation rate. It helps to improve the accuracy while reduce the computational work [9].

With respect to the optimizer analysis, we would like to do some studies on how to devise a best-suitable optimizer based on the applied data set. Because we have found different optimizers perform differently on a given data set. In other words, for a specific data set, if all available optimizers perform bad, we desire to devise a new optimizer which performs well. The new optimizer could be a mixture of several available optimizers or a brand new one. I think it would be interesting.

In addition, we would like to survey more on the network reduction techniques. We want to find a method which could promise the minimum decrease of accuracy while maximizing the reduction of time and space complexity.

Considering the mobile robot problem, this paper only considers the static walls and obstacles. In future work, we can put mobile robots in more complex environments. E.g. in a street with many people, a mobile robot needs to avoid bumping into people who keep moving.

5 The References Section

References

1. X. Ma and Y. Huang, Research on Mobile Cloud Robotics based on Cloud Computing, *Proceedings of the 2016 International Forum on Management, Education and Information Technology Application*, 2016.
2. M. Katsev, A. Yershova, B. Tovar, R. Ghrist and S.M. LaValle, Mapping and pursuit-evasion strategies for a simple wall-following robot, *IEEE Transactions on Robotics*, 2011.
3. F. Gmez and A. Quesada, Genetic algorithms for feature selection in Data Analytics, *Artelnics*, 2018.
4. J. YangVasant and V. Honavar, Feature Subset Selection Using a Genetic Algorithm. *The Springer International Series in Engineering and Computer Science*, vol 453, 1998.

5. T.D. Gedeon, D. Harris, Network Reduction Techniques, Proceedings International Conference on Neural Networks Methodologies and Applications, *AMSE*, San Diego, vol. 1: 119-126, 1991.
6. A. Frank, A. Asuncion et al., UCI machine learning repository, [http://archive.ics.uci.edu/ml], Irvine, CA, 2010.
7. J.T. Heaton, Introduction to Neural Networks for Java, *Heaton Research Inc.*, 2008.
8. J. Duchi, E. Hazan, Y. Singer, Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, *Journal of Machine Learning Research*, 2011.
9. K. Mineichi, and J. Sklansky. Comparison of Algorithms that Select Features for Pattern Classifiers. *Pattern recognition* 33.1: 25-41, 2000.
10. T. Dash, R.R. Swain, T. Nayak, Automatic navigation of wall-following mobile robot using a hybrid metaheuristic assisted neural network, *Data Science*, 2017.