PREDICTING POKER HAND'S STRENGTH WITH ARTIFICIAL NEURAL NETWORKS

Gökay Dişken

Adana Bilim ve Teknoloji Üniversitesi
gdisken@adanabtu.edu.tr

ABSTRACT

In this study, artificial neural networks are used to classify five cards drawn from a standard deck of 52 by suits and ranks of each card. After determining each cards value, system measures strength of the hand according to the poker game. MATLAB's Neural Networks Toolbox is used to design, train and test the network. The data used in this study is taken from UCI data archive page. Gradient descent method, scaled conjugate gradient method and resilient back-propagation methods are selected as training methods. Each method is used several times for different numbers of neurons, different learning rates, different maximum epoch numbers, and different transfer functions of output layer. Purelin and tansig functions are selected for output layer to see the effect of the functions on results. Simulations showed that tansig function is more suitable for this data. Training results for each training methods are compared. Resilient back-propagation method achieved over 90% correct classification values, while gradient descent with momentum method remained around 50%. Test results are very similar to training values, so we did not test all trained networks to save time and space but only select some of them to test with test data which is never used in training part.

1. Introduction

Artificial neural networks (ANN) are classifiers based on the human brain's working system. The neurons in the brain are mimicked by perceptrons in the artificial neural networks. In the brain, neurons are connected to many others and each neuron collets inputs from all other neurons connected to it and if a threshold value is reached, it signals to all the neurons it is connected to [1]. ANN uses the same method which makes it distinguish from other conventional programming methods. In most case, three layers of neurons are hired. One layer is the input layer and one layer is the output layer. Other layer is between them and called hidden layer. Mostly we know how many input and output neuron we need based on our data. We determine the number of neurons in the hidden layer by some kind of trial and error sense. We also have to account that small numbers of neurons may not be sufficient to network's proper learning and large numbers of neurons may cause overlearning and more computation time.

In this work, artificial neural networks are used to classify five cards from a standard deck of 52 by poker rules. Data for training and testing the designed networks can be found at UCI dataset page [2], a similar data set is used in [3] and also in [4] for a tutorial. The networks are designed with the aid of MATLAB's Neural Networks Toolbox. This paper is organized as following, in the second section basics of poker game and adjusting the dataset for MATLAB implementation is discussed, in the third section gradient descent method, scaled conjugate gradient method and resilient back-propagation method are described, in the fourth section simulation results are given and in the last section obtained results are discussed.

2. Basics of poker game, pre- and postprocessing the data

Poker is a family of card games involving betting and individual play, where the winner is determined by the ranks and combinations of their cards, some of which may remain hidden until the end of the game. The players do not have any information about components' hands. That's why the game has a bluff option which means a player acts like he has the best hand and force the others to fold so he can win the pot which may not be the case if all players bet until the last call. Since this project is not interested in actions like betting, folding and calling, we will only deal with the names of poker hands which, in fact, will be the outputs of the network system. The inputs of the system are ranks and suits of each card. In a hand we have five different cards so we will have ten inputs. We also have ten outputs because we have ten different strength levels. These are described in detail below.

The strength of a hand is increased by including multiple cards of the same rank or by all five cards being from the same suit or by the five cards forming a consecutive series. The Ace can be the lowest or the highest ranked card in series (the lowest if the series is A-2-3-4-5, the highest if the series is 10-J-Q-K-A). From the weakest to the most powerful, hand names are "high card", "one pair", "two pairs", "three of a kind", "straight", "flush", "full house", "four of a kind", "straight flush", "royal flush". High card means there is no multiple cards in the hand, value of the hand is the highest ranked card. One pair means, there is one pair of equal ranked cards. Two pairs mean we have two pairs of equal ranks. Three of a kind is three equal ranks. Straight means five cards are sequentially ranked without a gap. Flush is the name of a hand consisting five cards with the same suit. Full house is a

pair with three of a kind (ranks of the pair and other three are different). Four of a kind is four equal ranks. Straight flush is combination of straight and flush. Royal flush is the always winning hand with Ace, King, Queen, Jack, Ten with flush. The neural network system is going to classify the hand according to the data, then give one of the classes above as a result, depending on the cards ranks and suits.

The data we get from the UCI page includes two sets of data, one for training and one for testing the trained network. Before training the network we have to adjust the data for MATLAB. First we load the data as a 25010 x 11 sized matrix. The last column consists our targets, in other words the strength levels. So we separate that column to a new variable. The other columns represent the ranks and the suits of five cards. Another thing to do before the training is translate the output values to ones and zeros. Normally, output values are numbers from 0 to 9 where 0 means the weakest hand and 9 means the powerful hand. We need our output that shows the strength of the hand to be 1 and all the other outputs to be 0. This way we can easily understand the strength of the hand at a glance. These operations are done with a few steps of codes.

After training the network and testing the calculated weights, we have to compare the network outputs with our targets. Again to do this comparison we write a small program. It transforms the biggest value in each column to 1 and all other values to 0. Our target also has one 1 in each column so if we multiply these two matrix, only the correctly predicted values will be one and all wrong predictions will be zero. After counting how many ones we have, we can divide this number by our sample size then we get the percentage of true prediction.

3. Training Methods

We chose three methods for training the network. One of them is the gradient descent method with momentum. The command for this method in MATLAB is "traingdm". This method uses a back-propagation type neural network. The system tries to find a global minimum along the steepest vector of the error surface with the help of partial derivatives [5]. Back-propagation learning is a widely used algorithm for supervised learning with multi-layered feed-forward networks [6]. Learning rate parameter has very important effect in this method. If it is selected too small, large iterations will be needed to reach a sufficient solution. On the other hand, if it is selected a large value, calculations will tend to oscillate between some values and therefore prevent the error to fall below a certain value. A way to solve this problem is defining a momentum parameter, a parameter like the learning rate. This new parameter scales the influence of the previous step on the current step. This way, the convergence of the error function is increased.

The second method is scaled conjugate gradient method. The momentum parameter does not exist in this method. Instead of it, this method has a scale parameter. The scale parameter is used to increase the convergence of the system. This parameter determines a change in the weigths, instead of using fixed weigths. [7].

Other method is the resilient back-propagation method. This method performs a direct adaptation of the weight step based on local gradient information [6]. The biggest difference of this technique from other adaptation techniques is that it takes only the sign of the partial derivation, not the magnitude, and for each weight if there is a sign change comparing to the last iteration, the update value is multiplied by 0.5. If the sign is the same with the previous step, the update value is multiplied by 1.2. These values are chosen by the authors after experimenting results of different values. The resilient back-propagation method is compared with 2 other methods in the literature, Levenberg-Marquardt and conjugate gradient, and the results showed that the resilient back-propagation has the best accuracy in testing [8]. MATLAB command for this method is "trainrp".

4. Simulation Results

The network is designed with 10, 30 and 50 neurons in the hidden layer. Transfer function of the hidden layer is selected as "tansig" function for all designs. However, output layer's transfer function is changed in designs to see the effect of the transfer function on the neural network. For the gradient descent method, the momentum parameter is fixed at 0.7. Learning rate value is also selected 0.2 and 0.02 to compare the results of a big and a small learning rate values. Maximum epoch is one of the most important parameters. 1000, 2000 and 3000 maximum epoch values are used to limit the iteration. Mean Squared Error (MSE) goal is defined as 0.01, but none of the designs reached this limit. Validation check and minimum gradient parameters are fixed at 1000 and 1e-10, respectively. The designed networks are trained with the same parameters four times, because of the random nature of the weights at the start, different results are expected. The results given here are the best values obtained from a train session.

Table 1 shows the training results for different neuron sizes. Training function is gradient descent method for this one. Table 2 shows the same results of the same parameters except the learning rate. This parameter is changed to 0.2 in this one. Table 3 to Table 5 shows the effect of different iteration limits for different neuron numbers. All of them used the gradient descent method.

In Table 6, a comparison between three methods for 10 neurons is given as a reference before investigate the other parameters' effects. As seen in the table, trainrp and trainscg methods are given better results than the traingdm method. To understand the transfer function's effect, we change the output layers' functions from purelin to tansig. From the shapes of each function, we expect to have better results with tansig function as it suppresses the output value which is very useful for our data. The results given in Table 7 verified us. The tansig function is much better than the purelin function with both 0.2 and 0.02 learning rate values. If we increase the iteration limit, the results are even getting better (Table 8). The trainscg method did not get the same benefits from tansig function as the results can be seen from Table 9.

Table 1: Training results for different neuron numbers.

Neuron number	**Learning rate**	**Max. Epoch**	**Transfer function**	**Overall results**	**MSE results**
10	0.02	1000	purelin	%42.2	0.115
30	0.02	1000	purelin	%33.2	0.178
50	0.02	1000	purelin	%28.8	0.249

Table 2: Effect of the change in learning rate value.

Neuron number	**Max. epoch**	**Learning rate**	**Transfer function**	**Overall results**	**MSE results**
10	1000	0.2	purelin	%50.3	0.059
30	1000	0.2	purelin	%48.4	0.074
50	1000	0.2	purelin	%47.7	0.086

Table 3: Effect of different epoch limits for 10 neurons.

Neuron number	**Max. epoch**	**Learning rate**	**Overall results**	**MSE results**
10	1000	0.2	%50.3	0.059
10	2000	0.2	%51.3	0.057
10	3000	0.2	%51.0	0.056

Table 4: Effect of different epoch limits for 30 neurons.

Neuron number	**Max. epoch**	**Learning rate**	**Overall results**	**MSE results**
30	1000	0.2	%48.4	0.074

30	2000	0.2	%51.1	0.061
30	3000	0.2	%51.1	0.060

Table 5: Effect of different epoch limits for 50 neurons.

Neuron number	Max. epoch	Learning rate	Overall results	MSE results
50	1000	0.2	%47.7	0.086
50	2000	0.2	%50.7	0.067
50	3000	0.2	%52.7	0.062

Table 6: comparison of three methods for 10 neurons

Neuron number	Train func.	Learning rate	Max. epoch	Overall results	MSE results
10	trainrp	0.02	1000	%57.3	0.053
10	trainrp	0.2	1000	%55.1	0.054
10	traingdm	0.2	1000	%50.3	0.059
10	traingdm	0.02	1000	%42.2	0.115
10	trainscg	0.2	1000	%56.5	0.055
10	trainscg	0.02	1000	%58.9	0.052

Table 7: Effect of transfer function for resilient back-propagation method for 1000 iteration.

Neuron number	Transfer func.	Max epoch	Learning rate	Overall results	MSE results
30	purelin	1000	0.02	%55.0	0.054
30	tansig	1000	0.02	%76.3	0.036

30	purelin	1000	0.2	%55.1	0.054
30	tansig	1000	0.2	%65.5	0.047
50	purelin	1000	0.02	%54.8	0.055
50	tansig	1000	0.02	%85.0	0.028
50	purelin	1000	0.2	%55.1	0.054
50	tansig	1000	0.2	%78.6	0.037

Table 8: Effect of transfer function for resilient back-propagation method for 2000 iteration.

Neuron number	Transfer func.	Max epoch	Learning rate	Overall results	MSE results
10	purelin	2000	0.2	%55.0	0.055
10	tansig	2000	0.2	%58.0	0.052
30	purelin	2000	0.2	%57.9	0.052
30	tansig	2000	0.2	%76.2	0.036
50	purelin	2000	0.2	%57.4	0.053
50	tansig	2000	0.2	%92.4	0.010

Table 9: Effect of transfer function for scaled conjugate gradient method.

Neuron number	Transfer func.	Learning rate	Max epoch	Overall results	MSE results
10	purelin	0.2	1000	%56.5	0.055
10	purelin	0.02	1000	%58.9	0.052
10	tansig	0.2	1000	%58.2	0.052
10	tansig	0.02	1000	%50.0	0.073

To increase the poor results of Table 9, more neurons are hired. When 50 neurons are used the results gets better but the validation check limit can take the systems performance down if it is not selected properly. Table 10 shows this limitation. Note that this table shows only the best values. When trained with 100 validation limit value, the networks training stops too early. One can see the superior results of the training without reaching the validation limit as the results get high as 92.4%. However, the validation

limit value of 1000 is not even enough for our data because in some cases the limit is reached and the results dropped down to 48.9% (for 0.2 learning rate an 5000 epoch) and 59.8% (for 0.02 learning rate and 5000 epoch).

Table 10: Effect of transfer function for scaled conjugate gradient method.

Neuron number	Validation limit	Learning rate	Max epoch	Overall results	Train stop criteria
50	100	0.2	1000	%56.3	Valid. Limit reached
50	1000	0.2	1000	%73.1	Max. epoch reached
50	1000	0.2	5000	%92.3	Max. epoch reached
50	1000	0.02	5000	%92.4	Max. epoch reached

Table 11 shows the test results for 50 neurons and 2000 iteration. Test results are similar to train results.

Table 11: Test results for various networks.

Neuron number	Train method	Learning rate	Max epoch	Best train results	Test results
50	trainrp	0.02	2000	92.4%	91.85%
50	trainscg	0.02	2000	88.7%	87.49%
50	trainrp	0.2	2000	92.4%	91.89%
50	trainscg	0.2	2000	68.3%	64.73%

5. Results and discussion

In this study, artificial neural networks are used to classify poker hands by their strength levels which is a result of the ranks and suits of the cards in the hand. MATLAB's neural network toolbox is employed to design, train and test the networks. Feedforward type networks are designed with several different parameters such as transfer functions, neuron numbers in the hidden layer, iteration limit, validation check limit. Three different training functions are used. Best results are obtained with resilient back-propagation method with 50 hidden neurons, 92.4% for training, 91.85% for testing. Scaled conjugate gradient method also gives good results with 50 hidden neurons but requires more iteration than the resilient back-propagation. Gradient descent with momentum method gives similar results with other 2 methods when only 10 hidden neurons are used. Increasing the neuron numbers and changing the transfer function of the output neurons from purelin to tansig gives very good results for trainrp and trainscg methods but the traingdm method can not compete with these methods under this new conditions.

An important result from this study is that the effect of the validation limit value on the results. Table 10 tells us the small validation limit criteria could stop the training early and gives poor results. Even the network have the potential to reach over 90% training results as seen in the table, the validation limit stopped the training when the results just reached around 50-60% values.

The training and testing results showed that for this kind of data, resilient back-propagation method gives the best results among the selected methods. Different data sets may require different designs and it has to be noted that maybe we can further increase the results but there will be a more complex design or more hidden neurons or we need to increase the maximum iteration time, these will all make it longer for computer to give a solution because there will be more calculations.

6. References

1. HOPFIELD, John J. Artificial neural networks. Circuits and Devices Magazine, IEEE, 1988, 4.5: 3-10.

2. http://archive.ics.uci.edu/ml/datasets/Poker+Hand

3. R. Cattral, F. Oppacher, D. Deugo. Evolutionary Data Mining with Automatic Rule Generalization. Recent Advances in Computers, Computing and Communications, pp.296-300, WSEAS Press, 2002.

4. http://neuroph.sourceforge.net/tutorials/PredictingPokerhands/Predicting%20poker%20hands%20with%20neural%20networks.htm

5. HAYKIN, Simon. Neural networks: a comprehensive foundation. Prentice Hall PTR, 1994.

6. RIEDMILLER, Martin; BRAUN, Heinrich. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In: Neural Networks, 1993., IEEE International Conference on. IEEE, 1993. p. 586-591.

7. MØLLER, Martin Fodslette. A scaled conjugate gradient algorithm for fast supervised learning. Neural networks, 1993, 6.4: 525-533.

8. KISI, Özgür; UNCUOGLU, Erdal. Comparison of three back-propagation training algorithms for two case studies. Indian journal of engineering & materials sciences, 2005, 12.5: 434-442.