A Hybrid Neural Network Model Based Reinforcement Learning Agent

Pengyi Gao[1,*], Chuanbo Chen[1], Kui Zhang[2], Yingsong Hu[1], and Dan Li[1]

[1] School of Computer Science and Technology, HuaZhong University of Science and Technology,WuHan 430074, China
[2] School of Mechanical, Aerospace and Civil Engineering, University of Manchester, Oxford Road, Manchester, M13 9PL, United Kingdom
Pengyi_gao@mail.hust.edu.cn

Abstract. In this work, a hybrid neural network model (HNNM) is proposed, which combines the advantages of genetic algorithm, multi-agents and reinforcement learning. In order to generate networks with few connections and high classification performance, HNNM could dynamically prune or add hidden neurons at different stages of the training process. Experimental results have shown to be better than those obtained by the most commonly used optimization techniques.

Keywords: Multi-agent, Reinforcement learning, Neural networks, Optimization, Genetic algorithm.

1 Introduction

It is one of the most important issues over many years for researchers how to design an optimal and adaptive architecture for artificial neural networks (ANNs), especially hidden neurons and connections weights. There have been many approaches to adjust hidden neurons of ANNs, such as constructive or pruning, constructive and pruning, evolutionary and hybrid algorithms [1,2,3,4,5], etc. But, all of these methods have a common weakness, i.e. not able to adjust automatically networks architecture.

In order to dynamically adjust hidden neurons, Islam presented an adaptive merging and growing algorithm (AMGA), which could autonomously merge and add hidden neurons of ANNs [6]. However, there are some problems in the algorithm, for example, significance of each hidden neuron depends on experience, and specified parameters, such as significance threshold, can not be modified automatically. It is also difficulty to compute the correlation coefficient between selected neuron and other neuron, Additionally, BP training algorithm is easy to be trapped into local minima [7]. Considering the factors in AMGA, a hybrid learning model based on reinforcement learning (RL) agent is proposed in this work, where the neural networks consisting of three layers are trained by NN agent using GA algorithm, while training parameters will be chosen by RL agent, which include significance of hidden neurons, epochs, errors, initial weights, and hidden neurons, etc.

* Corresponding author.

L. Zhang, J. Kwok, and B.-L. Lu (Eds.): ISNN 2010, Part I, LNCS 6063, pp. 436–443, 2010.

The paper is organized as follows. Section 2 introduces proposed hybrid neural network learning model, and describes the function of main agents. In section 3, learning algorithm for NN agent is described in detail. Section 4 offers an insight into learning algorithm of RL agent. Experimental results are discussed in section 5 and the conclusions and a look to future work is put forward in the last.

2 Architecture of Hybrid Learning Model

In the model, we design a multi-agent platform, which consists of two containers, main container and agent container, shown in Fig.1. Two agents, NN agent and RL agent live in containers which provide a run-time and the services for agents. NN agent is responsible of pruning or adding hidden neurons and modifying weights of ANN by GA. RL agent receives the reward of NN agent and adjusts training parameters to NN agent according to the reward values and expert knowledge. Agents cooperate with each other to finish a given tasks [8,9,10].

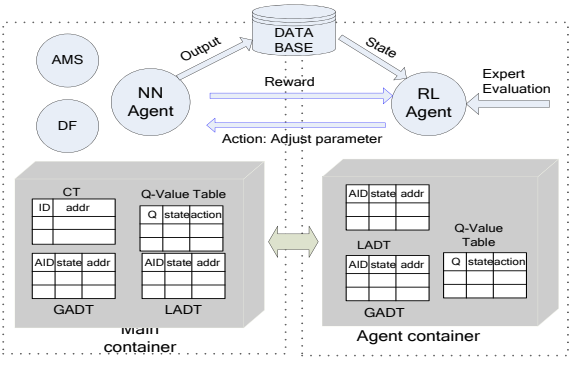

Fig. 1. The proposed learning model based multi-agent

The main container has the following special responsibilities:

- Managing the container table (CT).
- Managing agent descriptor table GADT and LADT.
- Managing Q-value table (QVT) for RL agents.

The AMS and the DF provide the agent management and white page service, and the default yellow page service of the platform, respectively.

3 Learning Algorithm for NN Agent

The aim of NN agent is to find a global best solution by GA algorithm. A solution vector is corresponding to an ANN, which consists of the number of neurons for each layer, ID of each hidden neuron, and their weights.

3.1 Algorithm Procedure

The developed algorithm for NN agent consists of the following steps:

Step 1: Generate randomly an initial population, which consists of variable length real-valued chromosomes, and each chromosome is corresponding to an ANN. Initialize the number of hidden neurons H and all connection weights at random. Set training epoch μ_i=0, i = 1, 2,. . .,H, for each hidden neuron h_i.
Step 2: The evolutionary process begins by applying the genetic operators (fitness, selection, crossover and mutation).
Step 3: Compute the objective function, that is sum of squared error (SSE) for each trained ANN and assign a probability for each solution according to SSE.
Step 4: Store the ANN architecture with the lowest objective value.
Step 5: Select a pair parent to perform crossover by the way of binary tournament [11], in order to reproduce the offspring that will be reinserted into the population replacing the solution with the smallest fitness.
Step 6: If the best offspring ANN has a lowest SSE, replace the last stored ANN.
Step 7: Generate randomly a new offspring by mutation operator in order to avoid being trapped into local minima.
Step 8: Execute steps 2–7 again until GA converges or maximum number of generations is reached.
Step 9: Compare SSE for the current best solution with the previous. If larger than the previous, then restore the previous network.
Step10: Increase epoch: $\mu_i = \mu_i + \tau$.
Step11: Save the current solution and the SSE into Database.
Step12: If termination criteria are met, stop the procedure and output the best.
Step13: Select the neurons, which significance η_i received from RL agent is less than threshold η_{th}. If not, go to the Step 16.
Step14: Compute the correlation between the selected hidden neurons and other hidden neurons in the network based on the output of hidden neurons.
Step15: If the pairs are found, then merge them and generate a new population, go to step 2 and retrain the networks.
Step16: If the neuron addition criterion is not satisfied, go to the Step 11.
Step17: Add one neuron to the hidden layer by splitting an existing neuron. The splitting produces two neurons from one neuron. Generate a new population and go to the Step 2 and retrain the networks.

3.2 Objective Function Definition

Objective function is defined by classification error TER, expressed as follows [7]:

$$TER = \frac{100}{\#P} \sum_{y \in P} \varepsilon(y) \tag{1}$$

Where #P is the number of patterns, $\varepsilon(y)$ is the error for the pattern y, which is 0 when true class of the pattern y is same as the desired class, otherwise, 1.

3.3 Correlation Coefficient Definition

Correlation measure can be used to measure linear relationship between variables, and it can also be used to find the correlation between different hidden neurons in the ANN. The correlation coefficient between $\mathbf{h}_i$ and $\mathbf{h}_j$ is defined as [12]:

$$P_d = \frac{cov\left(\mathrm{h}_i, \mathrm{h}_j\right)}{\sqrt{var(\mathrm{h}_i)var(\mathrm{h}_j)}} \tag{2}$$

Where cov is the covariance and var the variance, $\mathbf{h}_i$ and $\mathbf{h}_j$ are the output vector of a selected hidden neuron i and another unselected hidden neuron j, respectively in the training set. Actually, the coefficient is the cosine between output vectors.

3.4 Neuron Merging Criterion

If the correlation between neuron h_a and h_b is greater than the threshold, NN agent merges the two neurons into a new neuron h_m. The weights of h_m are assigned as

$$w_{mi} = (w_{ai} + w_{bi}) / 2 \tag{3}$$

$$w_{jm} = w_{ja} + w_{jb} \tag{4}$$

where w_{ai} and w_{bi} are the connection weights from ith input neuron to h_a and h_b, respectively, while w_{ja} and w_{jb} are the connection weights to jth output, respectively. W_{mi} and w_{jm} are the ith input and jth output connection weights of h_m, respectively [6].

3.5 Neuron Addition Criterion

The weights of the new two neurons created by splitting an existing neuron are calculated as follows:

$$w^1 = (1+\alpha)w \tag{5}$$

$$w^2 = -\alpha w \tag{6}$$

where $\mathbf{w}$ represents the weight vector of the existing neuron, and $\mathbf{w}^1$ and $\mathbf{w}^2$ are the weight vectors of the new neurons. α is a small mutation parameter.

3.6 Termination Criterion

The procedure is terminated when one of the criteria is met as follows:

① The number of successive iterations without any improvement of the objective function value is equal to the maximum.
② The validation error increases for T consecutive times.
③ Specified number of generations is reached.

4 Learning Algorithm for RL Agent

RL agent can complete the task to adjust automatically networks architecture through cooperation with NN agent. The learning process includes two main stages. First, RL agent observes the output of NN agent, and the state of solutions from Database, and explores an action that can yield the maximum reward, based on the former experience associated with the current observation and accumulated reinforcement (reward). Next, the agent takes those actions.

At the beginning, RL agent has nothing about the effect that actions may make. It discovers the actions, such as modifying the significance of hidden neuron according to an explored optimal policy which may bring the highest reward. After a while, the agent gradually takes those actions and then observes the results from NN agent. In fact, the reward can be defined objectively based on the results, or gained subjectively by directly receiving evaluation from the interactive expert.

State definition. Define the states s as a vector consisting of architecture of networks and modified hidden neurons.

Action definition. Define the actions a as a vector consisting of modified significance of hidden neuron, epoch and GA parameters such as population size.

Reward definition. Define the reward r as a vector by:

$$r_i = \frac{1}{TER_i} \tag{7}$$

Where r_i is the reward received at neuron i, TER_i is classification error associated with neuron i, it means that TER_i is evaluated according to its influences on training results when neuron i is merged or added into the ANN.

Reward updating criterion. RL agent uses Q-learning algorithm to learn the optimal action policy. The action value function Q(s,a) in this case satisfies equation as follows:

$$\mathrm{Q(s,a)} \leftarrow \mathrm{Q(s,a)} + \alpha(\mathrm{r} + \gamma(\max_{a'} \mathrm{Q(s',a')} - \mathrm{Q(s,a)}) \tag{8}$$

Where r is a received reward value, α is a learning rate, and γ is the discounting factor. In order to simplify computation, the policy update is performed based on a method called reinforcement comparison [13], which is updated by

$$\overline{r}_i(k+1) = \overline{r}_i(k) + \alpha(r_i(k) - \overline{r}_i(k)) \tag{9}$$

Where α $(0 < \alpha \leq 1)$ is a learning rate, and $\overline{r}_i(k)$ is reference reward of $r_i(k)$ in general. Then update the policy only if the received reward $r_i(k) > \overline{r}_i(k)$

5 Experimental Results

5.1 Testing Effectiveness of Algorithm

In this experiment, in order to evaluate the effectiveness of the proposed method, an empirical study is carried out on Diabetes data sets, which has eight inputs, two outputs, and 768 examples. In the training phase for the classifiers, 768 samples will be randomly divided into two subsets, training set with the 60% of samples and testing set with the remaining 40%.

All input are transformed within the range from 0 to 1. Initial population size is set 20, crossover rate is 0.9, and mutation rate is 0.1.

Figure 3 shows results including TER, and the best number of hidden neurons. TER is less when number of hidden neurons is 5, 10 and 11, respectively.

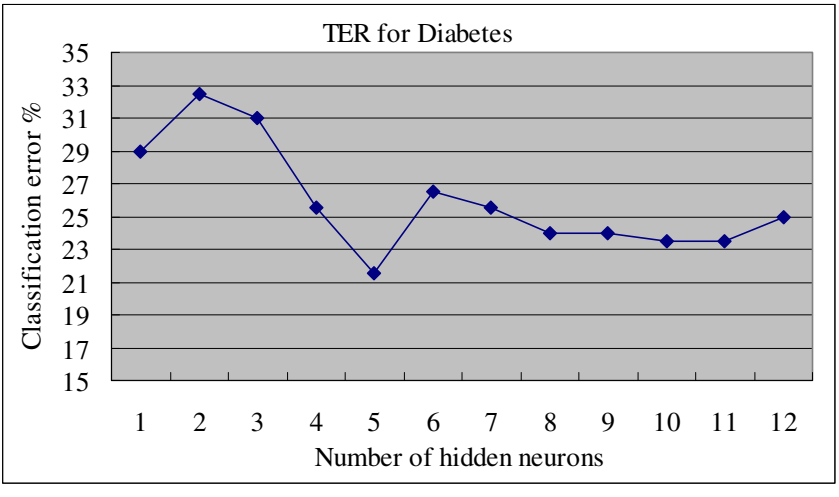

Fig. 2. Compute classification error for different hidden neurons. When the number of hidden neurons is 5, classification error is 21.5%, it is best solution.

5.2 Comparing with Other Algorithms

In order to compare effectively HNNM with other algorithms, we select adaptive merging and growing algorithm (AMGA), basic constructive algorithm (BCA), basic constructive–pruning algorithm (BCPA) and Ludermir algorithm[14] as candidates, also, we will take same experimental methodologies as possible.

Table 1 shows the results of HNNM, AMGA,BCA,BCPA and Ludermir over 50 independent runs for Diabetes data set. It can be seen that HNNM has the smallest number of hidden neurons and number of epochs, while BCP is largest in term of number of hidden neurons. With respect to TER, HNNM takes the lowest.

Table 1. The result of testing Diabetes Data set

Algorithm	Number of hidden neurons	TER	Number of epochs
HNNM	5.05	21.55	420
AMGA	4.14	21.97	390
BCA	5.96	26.04	467
BCPA	5.56	26.22	501
Ludermir	4.53	25.87	--

5.3 Discussion

From the experimental results, HNNM takes better performance than other algorithms in terms of TER. There are three reasons. One is that the neural networks are trained by GA, which can help to find a global best solution, and avoid procedure being trapped into a local minimum. Next is that HNNM uses an adaptive strategy like AMGA, and can dynamically prune or add the hidden neurons, while, other algorithms use a fixed modification strategy. Last is that training parameters is taken by RL agent. Unlike AMGA, in HNNM, RL agent evaluates significance of hidden neurons according to previous training results, feedback of current training and expert's knowledge. It isn't necessary to compute the significance according to empirical formula. HNNM is easy to be suitable for all classification problems.

6 Conclusion

This paper has present a hybrid neural network learning model based on reinforcement learning agents. In this model, NN agent is responsible of training the neural networks by using GA algorithm, while RL agent is responsible of determining the parameters of the neural networks by reinforcement learning algorithm, such as significance of hidden neurons, epochs, errors, initial weights, and hidden neurons, etc. There are two main contributions in this paper. First, due to being trained by GA, it's easy that HNNM can find a better global solution. Second, significance of the hidden neurons that will be merged may not be computed by formula, but determined by RL agent according to the evaluation at their influences on training results (through receiving reward and observing state of ANN), so that HNNM has the ability to generalize the training algorithm of ANN. A series of simulations are provided to demonstrate these.

References

1. Tsai, J., Chou, J., Liu, T.: Tuning the structure and parameters of a neural network by using hybrid Taguchi-genetic algorithm. IEEE Trans. Neural Netw. 17, 69–80 (2006)
2. Teoh, E.J., Tan, K.C., Xiang, C.: Estimating the number of hidden neurons in a feedforward network using the singular value decomposition. IEEE Trans. Neural Netw. 17, 1623–1629 (2006)
3. Islam, M., Sattar, A., Amin, F., Yao, X., Murase, K.: A New Constructive Algorithm for Architectural and Functional Adaptation of Artificial Neural Networks. IEEE Trans. on Systems, Man and Cybernetics—Part B: Cybernetics 39, 1590–1605 (2009)

4. Goh, C.K., Teoh, E.J., Tan, K.C.: Hybrid Multiobjective Evolutionary Design for Artificial Neural Networks. IEEE Trans. Neural Netw. 19, 1531–1547 (2008)
5. Hamid, B., Mohamad, R.M.: A learning automata-based algorithm for determination of the number of hidden units for three-layer neural networks. International Journal of Systems Science 40, 101–118 (2009)
6. Islam, M., Sattar, A., Amin, F., Yao, X., Murase, K.: A New Adaptive Merging and Growing Algorithm for Designing Artificial Neural Networks. IEEE Trans. on Systems, Man and Cybernetics—Part B: Cybernetics 39, 705–718 (2009)
7. Gao, P.Y., Chen, C.B., Qin, S., Hu, Y.S.: An Optimization Method for Neural Network Based on GA and TS Algorithm. In: 2nd International Conference on Computer and Automation Engineering. IEEE Press, New York (2010)
8. Farhang, S., Hamid, R.T., Magdy, M.M.A.S.: A reinforcement agent for object segmentation in ultrasound images. Expert Systems with Applications 35, 772–780 (2008)
9. Ronnie, W., Robert, W.: 2009 Special Issue: Representation in dynamical agents. Neural Networks 22, 258–266 (2009)
10. Tan, A.H.: Self-organizing neural architecture for reinforcement learning. In: Wang, J., Yi, Z., Żurada, J.M., Lu, B.-L., Yin, H. (eds.) ISNN 2006. LNCS, vol. 3971, pp. 470–475. Springer, Heidelberg (2006)
11. Benardos, P.G., Vosniakos, G.C.: Optimizing feedforward artificial neural network architecture. Engineering Application of Artificial Intelligence 20, 365–382 (2007)
12. Zhang, K., Andrew, B., Gu, F.S., Yu, H., Li, S.: A hybrid model with a weighted voting scheme for feature selection in machinery condition monitoring. In: Proc. of 3rd IEEE International Conference on Automation Science and Engineering, pp. 424–429. IEEE Press, New York (2007)
13. Sasakawa, T., Hu, J.L., Hirasawa, K.: A Brainlike Learning System with Supervised, Unsupervised, and Reinforcement Learning. Electrical Engineering in Japan 162, 32–38 (2008)
14. Ludermir, T.B., Akio, Y., Cleber, Z.: An optimization methodology for neural network weights and architectures. IEEE Trans. Neural Netw. 17, 1452–1457 (2006)