Reinforcement Learning with Modular Neural Networks for Control

Charles W. Anderson Zhaohui Hong

Department of Computer Science
Colorado State University
Fort Collins, CO 80523
anderson@cs.colostate.edu

Abstract

Reinforcement learning methods can be applied to control problems with the objective of optimizing the value of a function over time. They have been used to train single neural networks that learn solutions to whole tasks. Jacobs and Jordan [5] have shown that a set of expert networks combined via a gating network can more quickly learn tasks that can be decomposed. Even the decomposition can be learned. Inspired by Boyan's work of modular neural networks for learning with temporal-difference methods [4], we modify the reinforcement learning algorithm called Q-Learning to train a modular neural network to solve a control problem. The resulting algorithm is demonstrated on the classical pole-balancing problem. The advantage of such a method is that it makes it possible to deal with complex dynamic control problem effectively by using task decomposition and competitive learning.

1 Introduction

Neural networks have been applied in a number of ways to the problem of learning to control a system [3]. Usually there is a single network in a system. The performance of the system depends on many factors, such as the structure and the size of the problem to which it is applied, the amount of training data, the type of neurons in the network, and so on. Larger networks are capable of learning more complex functions, but generally require more training experience. This prompted the development of a modular network architecture to automatically decompose a problem and train multiple, smaller networks on sub-problems. Modular networks have been primarily studied in the supervised-learning paradigm. Here we develop a modular network form of a reinforcement learning algorithm, the Q-learning algorithm, and apply it to the simulated pole-balancing problem.

Reinforcement learning is a direct learning method in which the performance of the learning agent is evaluated based on a single scalar reinforcement signal. The objective is to determine an optimal policy which can determine the actions that the agent is going to take given each state.

Q-learning is a family of reinforcement learning algorithms initially developed by Watkins [6]. The prevalence of these algorithm is partially due to the existence of convergence proofs [7]. In Q-learning, the predicted long term cumulative reinforcement, called the Q-value, is a function of actions as well as input states. The Q-function acts as an evaluation function that predicts the discounted cumulative reinforcement. The action-selection policy is based on the Q-value and the adjustment of the reinforcement prediction is based on the temporal difference error in the Q-value. The Q-network learns the optimal Q-function that maps each action-state pair to the discounted cumulative reinforcement. The control policy simply selects the action that leads to the maximum Q-value for the current state.

Modular nets consist of several single networks and a *gating network*, which determines how much of each network's output is applied to the final output. The single network is also called an *expert network*. Expert networks and the gating network work together to learn to divide a task into several subtasks that are functionally independent. The gating network mediates the competition of each expert network and allocates distinct networks to learn each task. This modular architecture avoids learning a fixed global control policy, which may not be good for the whole control task under every different operation point.

Figure 1 shows a typical architecture of modular networks. There are two types of modular networks. The difference resides in the design of the gating network. For the first type of network a human decomposes the problem. A prior knowledge is hard-coded

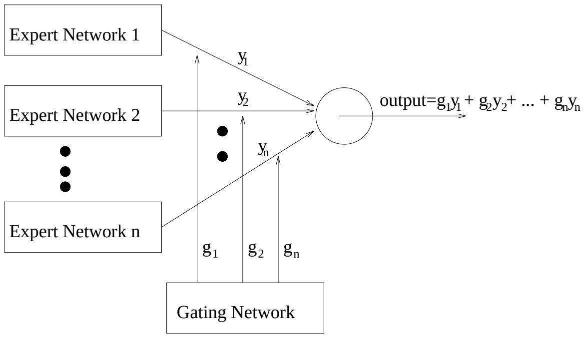

Figure 1: A Modular Connectionist Architecture

into the gating network so the gating network can coordinate the outputs of the expert networks without being trained. The second type of network consists of several expert networks and a gating network that learns to control the final output of the whole network. This network starts training from scratch and doesn't need any a priori knowledge.

Jacobs and Jordan [5] studied the modular network architecture in a "multiple payload robotics" task. They tested four architectures: a single network, a modular architecture, a modular architecture with a share network, and a constrained modular architecture with a share network, respectively. They concluded that faster learning speed over the single network can be achieved by developing a piecewise control strategy for each sub-network.

Boyan [4] developed a modular neural network for learning game strategies. He applied his "Designer Domain Decomposition" and "Meta-Pi" architectures to the Tic-Tac-Toe and Backgammon games. Both architectures used temporal-difference methods.

1.1 Q-Learning for Modular Networks

Figure 1 shows several expert networks in the modular architecture and one gating network. The expert network architectures are equivalent, and in fact can be used as stand-alone networks trained to learn the whole task. When used as part of a bigger modular network, each competes to learn a sub-task instead of learning the whole task. In Figure 1, $y_i, i = 1, 2, \ldots, n$ denotes the outputs of expert networks. The gating network has the same number of output units as the number of expert networks. The variables $g_i, i = 1, 2, \ldots, n$ denote the outputs of each output unit of the gating network. The values of g_i are non-negative and sum to one. The output of the entire network is determined by

$$output = \sum_{i=0}^{n} g_i y_i$$

In this way, the gating network determines how much each expert network should contribute to the final output.

The weights of a neural network being trained via Q-learning are modified so as to maximize the discounted cumulative reinforcement in the future:

$$V_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

where V_t is the discounted cumulative reinforcement, r_t is the reinforcement received after the state transition from t to $t+1$, γ is a discounted factor adjusting the importance of long term consequences of actions. During learning, $\max\{Q(y,k)|k \in actions\}$ is used as an approximation to the discounted cumulative reinforcement V_t.

When given a state x and an action a, the system goes into a new state y and gets feedback reinforcement r from the environment. The Q-function can be learned by the following steps:

1. input vector is fed into expert networks and the gating network.

2. the gating network selects ith expert network for that particular input state according to the pre-coded output value.

3. let u be the current value of $Q(x,a)$ output by the ith expert network;

4. let u' be the ith expert network's predicted value $r + \gamma \max\{Q(y,k)|k \in actions\}$;

5. update the weights of the ith expert network to improve Q-function by back-propagating the temporal difference error $u' - u$.

When the gating network is not fixed, the modular network is able to learn both a decomposition of the whole task and the control of each sub-task. The gating network with trainable weights takes the same input as what the expert networks take. The last layer of the network computes the weighted sums of the outputs of the hidden units. The weighted sum of the jth unit is denoted as s_j:

$$s_j = \sum_{i=1}^{m} x_i w_{ji}$$

where m is the number of hidden units, and w_{ji} is the weight connecting output node j and hidden unit output x_i. Because the outputs of gating network have to sum to one, the softmax activation function is used in the second layer of the network to meet this constraint. The ith output node is denoted as g_i:

$$g_i = \frac{e^{s_i}}{\sum_{j=1}^{n} e^{s_j}}$$

where n is the number of output units.

During the training, the weights of the expert networks and the gating network are updated at the same time using the backpropagation with TD-error. The Q-function is learned and the discounted cumulative reinforcement in the future is maximized. In our experiments, the expert networks used radial basis functions in the hidden units in order to compare directly to the results of Anderson [2].

2 Experiments and Results

The pole-balancing problem is a classic example of an inherently unstable system. It involves a pole hinged to the top of a wheeled cart which can move along a track of limited length. The system is modelled by two differential equations, taken from Anderson [1]. The neural network receives a performance feedback, which indicates failure when the pole falls past 12 degrees from vertical and when the cart hits the bounds of the track, and a four-component vector as the current state including the velocities and positions from the pole-cart system. The pole is said to be balanced if it does not fail within 10,000 steps. After each failure either due to that the pole falls to a degree greater than required or due to the cart hits the horizontal boundary, the pole and the cart is reset to the original position in which pole is straight up and the cart is in the middle of the track.

The performance is judged by the average number of failures before the pole balanced. The smaller it is, the better the performance is. Averages are taken from a total of 30 runs. Each run follows the procedure of initializing the networks, taking the pole/cart system state from input vector, predicting the action that needs to take, applying the action, getting next pole/cart system state, and updating the network. Each run ends up with the pole/cart system is balanced. The only difference among these runs is that the initial states of networks are different because of the difference seed values for the random number generator.

For the fixed gating network, we tested networks with two and with four expert networks and four partitioning methods: partitioning the state space according to the position of the cart, the velocity of the cart, the position of the pole, and the velocity of the pole. The results of a typical two-expert experiment are the following. The everage number of failures before balancing is 2433, 2364, 3548, and 4507 for partitions based on cart position, velocity, pole position, and pole velocity, respectively. The average number of failed runs for these four partition methods is 0, 0, 5, and 21. The average number of failures before balancing does not count the failed runs. From these results, we see that the performance of the network depends on the methods of task decomposition. The decomposition based on the cart's position and velocity are significantly better than on the pole's position and velocity.

A typical four-expert experiment resulted in only 17 runs balancing the pole and other 13 runs don't balance. The average number of failures before balance in the 17 successful runs is 4323, not counting the unsuccessful runs. This result shows that performance is not improved as the task is decomposed into more sub-tasks.

In experiments with a trainable gating network, our goal was to see whether the number of expert networks impact the performance of the whole network and how the gating network allocate the expert networks to the various input state spaces over different runs. Table 1 shows the performance of the network with various numbers of expert networks. The performance is almost the same for all the networks, which contradicts our expectations. We expected that the performance would increase as the number of experts increases. However, we found that a four expert networks with a trainable gating network generally performs better than the one just configured with a fixed gating network.

Table 1: Training with trainable gating network for different numbers of expert networks.

number of expert networks	2	4	8
avg. number of failures	2398.8	2364.97	2596.23
number of failed runs	0	0	0

Training with a trainable gating network results in a smaller average number of failures, and a smaller number of unsuccessful runs during the total of 30 runs. The trainable gating network helped the expert networks perform better than a fixed gating network. Table 2, which shows the training results under two same learning rate sets (in each pair, the first item

Table 2: Fixed Gating Network vs. Trainable Gating Network

	learning rate 1	learning rate 2
fixed	4323.12, 13	3199.75, 6
trainable	2405.00, 2	2430.33, 0

represents the average number of failures before balance, the second item represents the number of unsuccessful runs), also confirms this conclusion. This is encouraging because it shows that the gating network does learn the decomposition of the task and how to assign different expert network to respond to various state spaces.

In order to watch how the the gating network assigns different expert networks to various input state space, we investigated the outputs of the gating network. We found that out of 30 runs, 13 resulted in 1 expert network being allocated, 14 resulted in 2 networks allocated, 3 resulted in 3 networks, and no runs resulted in 4 networks. So sometimes the modular network just runs like a single network. Other expert networks are not in use. But there are still 14 runs in which there are two networks allocated. In most of these runs, the cart's velocity was used to allocate the expert networks, which matches the results that we got from the training with 2 expert networks and a fixed gating network. One works on the positive cart's velocity and another works on the negative cart's velocity. There are three runs that allocate three expert networks. No run allocates all four expert networks. Other expert networks actually are not in use.

3 Discussion and Conclusion

Compared with the training on the same problem with single neural network, the results show that the modular neural network does not surpass the performance of the single networks. With the same values of β, β_h, and β_σ, an experiment with the single network succeeded in balancing the pole/cart system for 29 runs and the average number of failure before balance is 2532 without counting the failure run.

This work has shown that Q-learning can be adapted for a modular network architecture and applied to dynamic control problems. However, our preliminary results show that the modular networks do not perform better that a single network architecture. This might be related to the symmetry of the pole-balancing problem, which results in a fairly simple, single-network solution. For example, the control signal may be negated when the cart or the pole goes from left to right. The pole-balancing problem is not a typical control problem in which different local control models are required. A global control model can perform as well as two local control models.

Possible extensions of this research are to try different activation functions for the hidden layer in the expert networks, to test more parameter-value combinations, and to apply such an architecture and learning algorithm to a dynamic problem with a more complex input state space.

References

[1] C. W. Anderson. Strategy learning with multilayer connectionist representations. Technical Report TR87-509.3, GTE Laboratories, Waltham, MA, 1987. Revision of article that was published in Proceedings of the Fourth International Workshop on Machine Learning, pp. 103–114, June, 1987.

[2] Charles W. Anderson. Q-learning with hidden-unit restarting. In Stephen Jose Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 81–88. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[3] A. G. Barto. Connectionist learning for control: An overview. In W. T. Miller, R. S. Sutton, and P. J. Werbos, editors, *Neural Networks for Control*, chapter 1, pages 5–58. MIT Press, Cambridge, MA, 1990.

[4] Justin A. Boyan. Modular neural networks for learning context-dependend game strategies. Master's thesis, University of Cambridge, 1992.

[5] R. A. Jacobs and M. I. Jordan. A modular connectionist architecture for learning piecewise control strategies. In *Proceedings of the 1991 American Control Conference*, 1991.

[6] C. Watkins. *Learning with Delayed Rewards.* PhD thesis, Cambridge University Psychology Department, 1989.

[7] Whitley, Dominic, Das, and Anderson. Genetic reinforcement learning for neurocontrol problems. Technical Report CS92-102, CSU, 1992.