NEURAL NETWORKS AND REINFORCEMENT LEARNING

Abhijit Gosavi

Department of Engineering Management and Systems Engineering
Missouri University of Science and Technology
Rolla, MO 65409

Outline

What is Reinforcement Learning?

- Reinforcement Learning (RL) is a technique useful in solving control optimization problems.
- By control optimization, we mean the problem of recognizing the best action in every state visited by the system so as to optimize some objective function, e.g., the average reward per unit time and the total discounted reward over a given time horizon.
- Typically, RL is used when the system has a very large number of states ($>> 1000$) and has complex stochastic structure, which is not amenable to closed form analysis.
- When problems have a relative small number of states and the underlying random structure is relatively simple, one can use dynamic programming.

Q-Learning

- The central idea in Q-Learning is to recognize or learn the optimal action in every state visited by the system (also called the optimal policy) via trial and error.
- The trial and error mechanism can be implemented within the real-world system (commonly seen in robotics) or within a simulator (commonly seen in management science / industrial engineering).

Q-Learning: Working Mechanism

- The agent chooses an action, obtains feedback for that action, and uses the feedback to update its database.
- In its database, the agent keeps a so-called Q-factor for every state-action pair. When the feedback for selecting an action in a state is positive, the associated Q-factor's value is increased, while if the feedback is negative, the value is decreased.
- The feedback consists of the immediate revenue or reward plus the value of the next state.

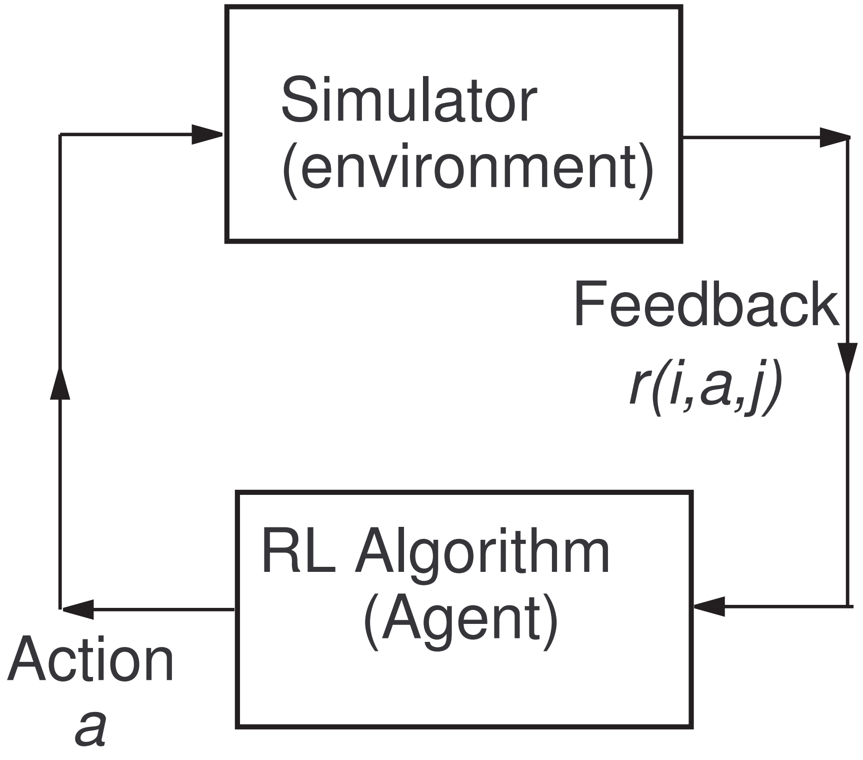

Figure 1: Trial and error mechanism of RL. The action selected by the RL agent is fed into the simulator. The simulator simulates the action, and the resultant feedback obtained is fed back into the knowledge-base (Q-factors) of the agent. The agent uses the RL algorithm to update its knowledge-base, becomes smarter in the process, and then selects a better action.

Q-Learning: Feedback

- The immediate reward is denoted by $r(i,a,j)$, where i is the current state, a the action chosen in the current state, and j the next state.
- The value of any state is given by the maximum Q-factor in that state. Thus, if there are two actions in each state, the value of a state is the maximum of the two Q-factors for that state.
- In mathematical terms:

$$feedback = r(i,a,j) + \lambda \max_b Q(j,b),$$

where λ is the discount factor, which discounts the values of future states. Usually, $\lambda = 1/(1+R)$ where R is the rate of discounting.

Q-Learning: Algorithm

The core of the Q-Learning algorithm uses the following updating equation:

$$Q(i,a) \leftarrow [1-\alpha]Q(i,a) + \alpha\,[feedback]\,,$$

i.e.,

$$Q(i,a) \leftarrow [1-\alpha]Q(i,a) + \alpha\left[r(i,a,j) + \lambda \max_b Q(j,b)\right],$$

where α is the learning rate (or step size).

Q-Learning: Where Do We Need Neural Networks?

- When we have a very large number of state-action pairs, it is not feasible to store every Q-factor separately.
- Then, it makes sense to store the Q-factors for a given action within one neural network.
- When a Q-factor is needed, it is fetched from its neural network.
- When a Q-factor is to be updated, the new Q-factor is used to update the neural network itself.
- For any given action, $Q(i, a)$ is a function of i, the state. Hence, we will call it a Q-function in what follows.

Incremental or Batch?

- Neural networks are generally of two types: batch updating or incremental updating.
- The batch updating neural networks require all the data at once, while the incremental neural networks take one data piece at a time.
- For reinforcement learning, we need incremental neural networks since every time the agent receives feedback, we obtain a new piece of data that must be used to update some neural network.

Neurons and Backpropagation

- Neurons are used for fitting linear forms, e.g., $y = a + bi$ where i is the input (the state in our case). Also called adenaline rule or Widrow-Hoff rule.
- Backprop is used when the Q-factor is non-linear in i, which is usually the case. (Algorithm was invented by Paul Werbos in 1975).
- Backprop is a universal function approximator, and ideally should fit any Q-function!
- Neurons can also be used by fitting the Q-function in a piecewise manner, where a linear fit is introduced in every piece.

Algorithm for Incremental Neuron

Step 1: Initialize the weights of the neural network.

Step 2a: Compute the output o_p using

$$output = \sum_{j=0}^{k} w(j)x(j), \text{ where}$$

$w(j)$ is the jth weight of neuron and $x(j)$ is the jth input.

Step 2b: Update each $w(i)$ for $i = 0, 1, \ldots, k$ using:

$$w(i) \leftarrow w(i) + \mu[target - output]x(i),$$

where the target is the updated Q-factor.

Step 3: Increment *iter* by 1. If $iter < iter_{\max}$, return to Step 2.

Q-Learning combined with Neuron

We now discuss a simple example of Q-Learning coupled with a neuron using incremental updating on an MDP with two states and two actions.

Step 1. Initialize the weights of the neuron for action 1, i.e., $w(1,1)$ and $w(2,1)$, to small random numbers, and set the corresponding weights for action 2 to the same values. Set k, the number of state transitions, to 0. Start system simulation at any arbitrary state. Set $k_{\max}$ to a large number.

Step 2. Let the state be i. Simulate action a with a probability of $1/|\mathcal{A}(i)|$. Let the next state be j.

Step 3. Evaluate the Q-factor for state-action pair, (i,a), which we will call Q_{old}, using the following:

$$Q_{\text{old}} = w(1,a) + w(2,a)i.$$

Now evaluate the Q-factor for state j associated to each action, i.e.,

$$Q_{\text{next}}(1) = w(1,1) + w(2,1)j; \qquad Q_{\text{next}}(2) = w(1,2) + w(2,2)j.$$

$$\text{Now set } Q_{\text{next}} = \max\left\{Q_{\text{next}}(1), Q_{\text{next}}(2)\right\}.$$

Step 3a. Update the relevant Q-factor as follows (via Q-Learning).

$$Q_{\text{new}} \leftarrow (1-\alpha)Q_{\text{old}} + \alpha\left[r(i,a,j) + \lambda Q_{\text{next}}\right]. \tag{1}$$

Step 3b. The current step in turn may contain a number of steps and involves the neural network updating. Set $m = 0$, where m is the number of iterations used within the neural network. Set m_{max}, the maximum number of iterations for neuronal updating, to a suitable value (we will discuss this value below).

Step 3b(i). Update the weights of the neuron associated to action a as follows:

$$w(1,a) \leftarrow w(1,a)+\mu(Q_{\text{new}}-Q_{\text{old}})1; \quad w(2,a) \leftarrow w(2,a)+\mu(Q_{\text{new}}-Q_{\text{old}})i. \tag{2}$$

Step 3b(ii). Increment m by 1. If $m < m_{\max}$, return to Step 3b(i); otherwise, go to Step 4.

Step 4. Increment k by 1. If $k < k_{\max}$, set $i \leftarrow j$; then go to Step 2. Otherwise, go to Step 5.

Step 5. The policy learned, $\hat{\mu}$, is virtually stored in the weights. To determine the action prescribed in a state i where $i \in \mathcal{S}$, compute the following:

$$\mu(i) \in \arg\max_{a \in \mathcal{A}(i)} \left[w(1,a) + w(2,a)i \right].$$

Some important remarks need to be made in regards to the algorithm above.

Remark 1. Note that the update in Equation (2) is the update used by an incremental neuron that seeks to store the Q-factors for a given action.

Remark 2. The step-size μ is the step size of the neuron, and it can be also be decayed with every iteration m

Backpropagation

- The algorithm is more complicated, since it involves multiple layers and the threshold functions.
- The algorithm requires significant tuning.
- Incremental version of the algorithm must be used.

Integration of neural networks with Q-Learning

- In some sense resembles integration of hardware (neural network) and software (Q-Learning).
- Integration has to be tightly controlled; otherwise results can be disappointing.
- Mixed success with backprop; Crites and Barto (1996; elevator case study), Das et al (1999; preventive maintenance used backprop with great success), and Sui (supply chains) have obtained great success, but some other failures reported (Sutton and Barto's 1998 textbook).
- Piecewise fitting of the function using neurons has also shown robust and stable behavior: Gosavi (2004; airline revenue management).

Toy Problem: Two states and Two actions

Table 1: The table shows Q-factors.

Method	$Q(1,1)$	$Q(1,2)$	$Q(2,1)$	$Q(2,2)$
Q-factor-value iteration	44.84	53.02	51.87	49.28
Q-Learning with $\alpha = 150/(300+n)$	44.40	52.97	51.84	46.63
Q-Learning with Neuron	43.90	51.90	51.54	49.26

Future Directions

- A great deal of current research in function approximation and RL is using regression, e.g., algorithms such as LSTD (least squares temporal differences)
- However, most exciting RL applications in robotics and neuro-science studies are still using neural networks.
- Support Vector Machines are yet another data mining tool that have seen some recent applications in RL.

References

D. Bertsekas and J. Tsitsiklis. *Neuro-dynamic programming*, Athena, 1996.

R. Crites and A. Barto. Improving elevator performance using reinforcement learning. *In Neural Information Processing Systems (NIPS)*. 1996.

T.K. Das, A. Gosavi, S. Mahadevan, and N. Marchalleck. Solving semi-Markov decision problems using average reward reinforcement learning. *Management Science*, 45(4):560574, 1999.

A. Gosavi. *Simulation-based optimization: Parametric optimization techniques and reinforcement learning*, Kluwer Academic Publishers, 2009.

A. Gosavi, N. Bandla, and T. K. Das. A reinforcement learning approach to a single leg airline revenue management problem with multiple fare classes and overbooking. *IIE Transactions* (Special Issue on Large-Scale Optimization edited by Suvrajeet Sen), 34(9):729742, 2002.

R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*, MIT Press, 1998.