

ONLINE SUPPLEMENT TO Reinforcement Learning: A Tutorial Survey and Recent Advances

INFORMS JOURNAL ON COMPUTING

Abhijit Gosavi

219 Engineering Management

Department of Engineering Management and Systems Engineering

Missouri University of Science and Technology, Rolla, MO 65409

gosavia@mst.edu

Regular Markov chains: For a *regular* Markov chain, there exists a finite value for k , where $k > 0$ and k is an integer, such that if $\mathbf{P}$ denotes the one-step TPM of the Markov chain concerned, then every element of $\mathbf{P}^k$ is strictly greater than 0. (See Grinstead and Snell, 1997) for an in-depth discussion.)

Q -value methods for SMDPs: For continuous-time SMDPs, in which the time of transition has an exponential distribution (Bertsekas, 1995), Bradtke and Duff (1995) presented the following updating rule for the Q -values, which can be used in place of Step 4 in Figure 6 of main text:

$$Q(i, a) \leftarrow (1 - \mu)Q(i, a) + \mu \left[\frac{1 - e^{-Rt(i,a,j)}}{R} r(i, a, j) + e^{-Rt(i,a,j)} \max_{b \in \mathcal{A}(j)} Q(j, b) \right],$$

where $t(i, a, j)$ denotes the time spent in the transition from i to j under action a . For the more general case, the following two updating rules were proposed in Gosavi (2003) (pgs 245-247). The first applies to Q -Learning, as described in Figure 6 of main text, with Step 4 replaced by:

$$Q(i, a) \leftarrow (1 - \mu)Q(i, a) + \mu \left[r(i, a, j) + e^{-Rt(i,a,j)} \max_{b \in \mathcal{A}(j)} Q(j, b) \right],$$

and the second to Q - P -Learning, as described in Figure 9 of main text, with Step 2b replaced by:

$$Q(i, a) \leftarrow (1 - \mu)Q(i, a) + \mu \left[r(i, a, j) + e^{-Rt(i,a,j)} Q \left(j, \arg \max_{b \in \mathcal{A}(j)} P(j, b) \right) \right].$$

Model building: The idea of building a model via *straightforward counting* can be formalized as follows. Let $N(i, a)$ denote the number of times action a is tried in state i , and let $N_j(i, a)$ denote the number of times the selection of action a in state i leads the system to state j in the next decision-making epoch. All of these counters are initialized to 0 in the beginning. In a model-based algorithm, one keeps estimates of the value function, $h(i)$, and the expected immediate reward, $\bar{r}(i, a)$, all of which are initialized to 0 in the beginning. When the system transitions to state j after action a is selected in state i , the following updates are carried out in the order given below:

$$N(i, a) \leftarrow N(i, a) + 1, \quad N_j(i, a) \leftarrow N_j(i, a) + 1$$

$$\tilde{p}(i, a, l) \leftarrow N_l(i, a)/N(i, a), \text{ for all } l \in \mathcal{S} \text{ and } \bar{r}(i, a) \leftarrow \bar{r}(i, a) + \frac{[r(i, a, j) - \bar{r}(i, a)]}{N(i, a)}.$$

Then, the value function, h , of DP is updated with estimates of the TPs and expected immediate rewards. The main updating rule for discounting from Barto et al (1995) (see Tadepalli and Ok (1998) for average reward) is:

$$h(i) \leftarrow \max_{u \in \mathcal{A}(i)} \left[\bar{r}(i, u) + \gamma \sum_{l \in \mathcal{S}} \tilde{p}(i, u, l) h(l) \right].$$

Neural networks: A back-propagation neural network has at least three layers, the input layer, the hidden layer, and an output layer, and associated with each layer is a finite number of nodes. See Figure 1 in online supplement. The unique output node predicts the value of the Q -value for a given state. Usually, a separate net is used for each action. Two classes of weights are used for representation: weights from the input layer to the hidden layer and weights from the hidden layer to the output layer. These weights are gradually adjusted in an *incremental* manner as data become available, i.e., every time a Q -value is updated.

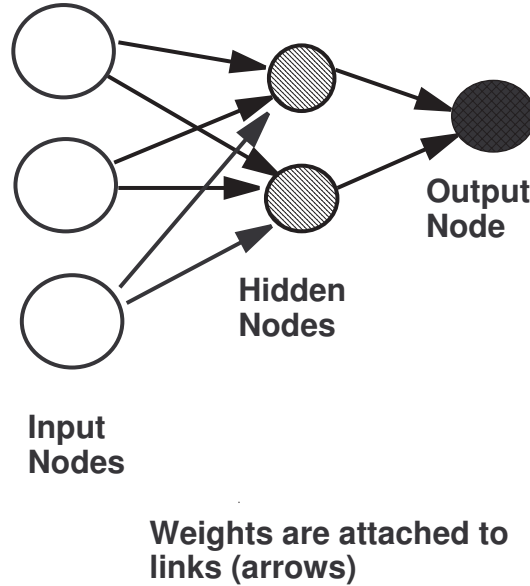


Figure 1: A schematic view of a multi-layer neural network with three input nodes, two hidden nodes, and one output node.

Let $W_1(j, h)$ denote the weights from the j th node in the input layer to the h th node in the hidden layer, and let $W_2(h)$ denote the weight from the h th hidden node to the output node. Let n denote the number of input nodes, H the number of hidden nodes, and let Q_{new} be the *updated* (new) value for the Q -value. All the weights are initialized to very small values before the RL algorithm starts operation. Every time a Q -value is updated by the RL algorithm, the following “sweep” of computations is carried out to update the weights in the neural network (to accommodate the change). Each sweep consists of the following four steps and the number of sweeps required per iteration of the RL algorithm has to be determined by trial and error with respect to the task at hand.

Step 1. Compute $v(h)$ for $h = 1, 2, \dots, H$, using the following function:

$$v(h) \leftarrow \frac{1}{1 + e^{-v^*(h)}}, \text{ where } v^*(h) \leftarrow \sum_{j=1}^n W_1(j, h)x(j),$$

in which $x(j)$ denotes the j th state variable of the Q -value being updated.

Step 2. Now, compute

$$Q_{old} \leftarrow \sum_{h=1}^H W_2(h)v(h).$$

Step 3. For each $h = 1, 2, \dots, H$ and each $j = 1, 2, \dots, n$, update the weights as follows:

$$W_1(j, h) \leftarrow W_1(j, h) + \mu_{NET}(Q_{new} - Q_{old})W_2(h)v(h)(1 - v(h))x(j),$$

where μ_{NET} is the learning rate of the neural network.

Step 4. For each $h = 1, 2, \dots, H$, update

$$W_2(h) \leftarrow W_2(h) + \mu_{NET}(Q_{new} - Q_{old})v(h).$$

Kernel methods: An example that assigns a distance-based weight to every Q -value would work as follows: Employ for action a , the function $\tilde{w}(i, l, a)$, which, for all given points i and l in the state space, is defined as:

$$\tilde{w}(i, l, a) = D\left(\frac{|i - l|}{\omega_a}\right), \text{ in which}$$

ω_a is a smoothing parameter that has to be tuned with respect to the task at hand, and

$$D(s) = 3/4(1 - s^2) \text{ if } |s| \leq 1 \text{ and } D(s) = 0 \text{ otherwise.}$$

The Q -value at any state l and for action a in such a representation is computed as:

$$Q(l, a) = \frac{\sum_i \tilde{w}(i, l, a)Q(i, a)}{\sum_i \tilde{w}(i, l, a)}, \text{ where the summation is over all the exemplar points.}$$

References:

- A.G. Barto, S.J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81-138, 1995.
- D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena, MA, 1995.
- S.J. Bradtke and M. Duff. Reinforcement learning methods for continuous-time Markov decision problems. In *Advances in Neural Information Processing Systems* 7. MIT Press, Cambridge, MA, USA, 1995.
- A. Gosavi. *Simulation-based Optimization: Parametric Optimization Techniques and Reinforcement Learning*. Kluwer Academic, Boston, MA, 2003.
- C.M. Grinstead and J.L. Snell. *Introduction to Probability*. American Mathematical Society, Providence, RI, USA, 1997.
- P. Tadepalli and D. Ok. Model-based Average Reward Reinforcement Learning Algorithms. *Artificial Intelligence*, 100:177-224, 1998.