Lecture 15

Online Approximate Dynamic Programming

In the last chapter, we considered the case of offline learning of the value function via fitted Value iteration or fitted Q-Iteration. In this lecture, we will consider the case of online approximate dynamic programming.

15.1. Temporal Difference Learning

Let's start again with the discrete-time, discrete action scenario. We have the dynamic programming recursion given as

$$J^*(s) = \min_a \left[g(s,a) + J^*(s')\right]. \tag{15.1}$$

Similarly, for a fixed policy $\pi(s)$, we have

$$J^\pi(s) = g(s,a) + J^\pi(s'). \tag{15.2}$$

Now imagine we want to update an estimate of $\hat{J}^\pi$. One approach might be to use gradient descent. Consider the loss function

$$\mathcal{L} = \frac{1}{2}(J^\pi(s) - \hat{J}^\pi(s))^2. \tag{15.3}$$

The gradient descent update is given as

$$\hat{J}^{\pi,i+1}(s) = \hat{J}^{\pi,i}(s) - \alpha \frac{\partial \mathcal{L}}{\partial \hat{J}^{\pi,i}}. \tag{15.4}$$

where $\alpha > 0$. How do we estimate $\hat{J}$?

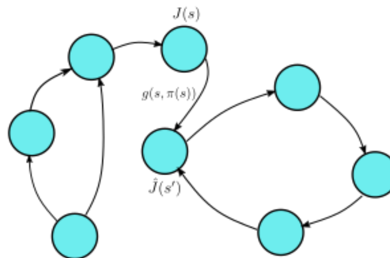

Figure 15.1: Bootstrapping on a Markov Decision Process.

© Joseph Moore, 2025

If we let

$$J^\pi(s) \approx g(s,a) + \gamma \hat{J}^\pi(s') \tag{15.5}$$

then

$$\mathcal{L} = \frac{1}{2}(g(s,a) + \gamma \hat{J}^\pi(s') - \hat{J}^\pi(s))^2 \tag{15.6}$$

which gives

$$\frac{\partial \mathcal{L}}{\partial \hat{J}^{\pi,i}} = -(g(s,a) + \gamma \hat{J}^\pi(s') - \hat{J}^\pi(s)). \tag{15.7}$$

$g + \gamma \hat{J}^{\pi'} - \hat{J}^\pi$ is known as the "temporal difference error". The final update is given as

$$\hat{J}^{\pi,i+1}(s) = \hat{J}^{\pi,i}(s) + \alpha(g(s,a) + \gamma \hat{J}^\pi(s') - \hat{J}^{\pi,i}(s)) \tag{15.8}$$

which can also be written as

$$\hat{J}^{\pi,i+1}(s) = (1-\alpha)\hat{J}^{\pi,i}(s) + \alpha(g(s,a) + \gamma \hat{J}^\pi(s')). \tag{15.9}$$

15.1.1 Data Inefficiency

One thing that we observe with "online" approximate dynamic programming methods is that it does not use data efficiently. Consider a policy guiding the robot on the grid world, where the policy first goes all the way right and then all the way down. The robot receives a cost of (-1) for transitioning into the goal region (4,1). For the first iteration, we have:

$$\begin{aligned}
\hat{J}^{\pi,i+1}(4,2) &= \hat{J}^{\pi,i}(4,2) + \alpha(g((4,2),(4,1)) + \gamma \hat{J}^\pi((4,1)) - \hat{J}^{\pi,i}((4,2)) && (15.10)\\
&= 0 + \alpha(-1 + 0 - 0) && (15.11)\\
&= -\alpha. && (15.12)
\end{aligned}$$

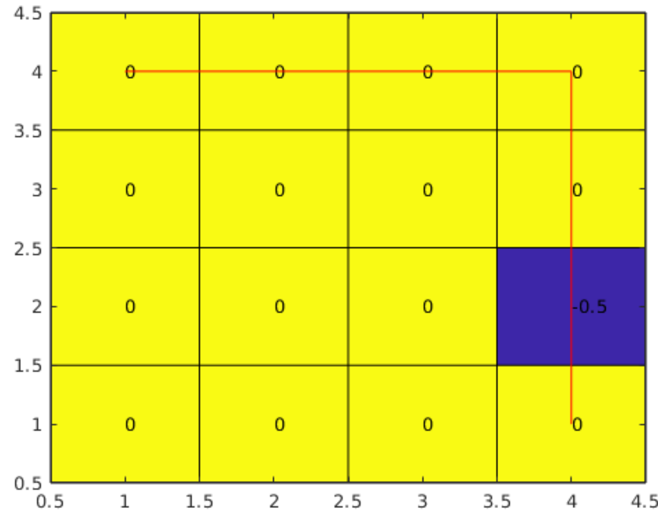

Figure 15.2: TD learning at Iteration 1

For the second iteration we have

$$\hat{J}^{\pi,i+1}(4,2) = \hat{J}^{\pi,i}(4,2) + \alpha\Big(g((4,2),(4,1)) + \gamma\hat{J}^{\pi}((4,1)) - \hat{J}^{\pi,i}((4,2))\Big) \tag{15.13}$$

$$= -\alpha + \alpha(-1 + 0 - (-\alpha)) \tag{15.14}$$

$$= -2\alpha + \alpha^2 \tag{15.15}$$

and

$$\hat{J}^{\pi,i+1}(4,3) = \hat{J}^{\pi,i}(4,3) + \alpha\Big(g((4,3),(4,2)) + \gamma\hat{J}^{\pi}((4,2)) - \hat{J}^{\pi,i}((4,3))\Big) \tag{15.16}$$

$$= 0 + \alpha(0 + \gamma(-\alpha) - 0) \tag{15.17}$$

$$= -\gamma\alpha^2. \tag{15.18}$$

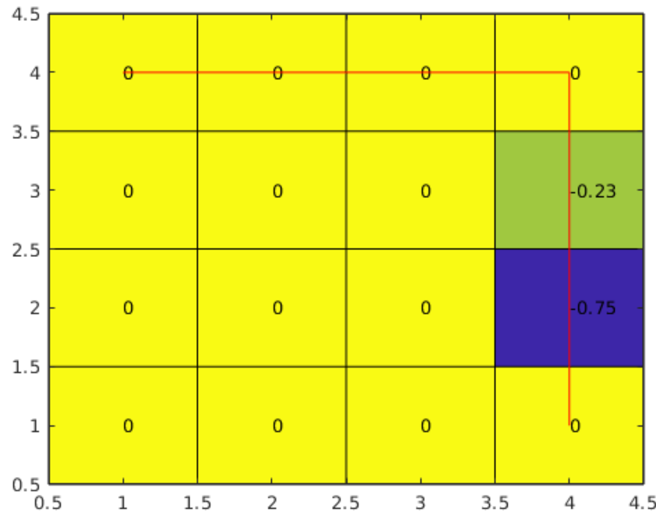

Figure 15.3: TD learning at Iteration 2

15.2. Monte Carlo Evaluation

Another means of estimating the cost-to-go is using a Monte Carlo Rollout:

$$\hat{J}^{\pi}_n(s_0) = \sum_{k=0}^{\infty} \gamma^k g(s_k, \pi(a_k)). \tag{15.19}$$

If the system (or policy) is stochastic we can estimate

$$\mathbb{E}[J^{\pi}(s_0)] = \frac{1}{N}\sum_{n=1}^{N} \hat{J}^{\pi}_n(s_0). \tag{15.20}$$

In general, there can be an advantage to combining the idea of Monte-Carlo Evaluation with Bootstrapping. For instance, consider

$$\hat{J}^{\pi}(s_k) = g(s_k, \pi(s_k)) + \gamma g(s_{k+1}, \pi(s_{k+1})) + \gamma^2 \hat{J}^{\pi}(s_{k+2}) \tag{15.21}$$

or more generally

$$\hat{J}^{N}(s_k) = \sum_{n=0}^{N-1} \gamma^n g(s_{k+n}, \pi(s_{k+n})) + \gamma^N \hat{J}^{\pi}(s_{k+N}). \tag{15.22}$$

© Joseph Moore, 2025

15.3. TD-λ

To select the N value to determine the “roll-out” horizon, Richard Sutton proposed the TD-λ algorithm. He proposed

$$J^{\lambda}(s_k) = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}J^n, \tag{15.23}$$

where $\lambda \in [0,1]$. If we use the notion of the elligibility trace, we have

$$e_{k+1}(s_k) = \begin{cases} \gamma\lambda e_k(s_k) & \text{if} \quad s_{k+1} \neq s_k \\ \gamma\lambda e_k(s_k) + 1 & \text{if} \quad s_{k+1} = s_k \end{cases} \tag{15.24}$$

where $\vec{e}(0) = 0$ and

$$\vec{\hat{J}}^{\pi,i+1} = \vec{\hat{J}}^{\pi,i} + \alpha\Big(g(s_k,\pi(s_k)) + \gamma\hat{J}^{\pi,i}(s_{k+1}) - \hat{J}^{\pi,i}(s_k)\Big)\vec{e}_k. \tag{15.25}$$

15.4. SARSA

SARSA or (state-action-reward-state-action) extends TD-learning by employing Q-factors. Remember

$$Q^{\pi}(s_k,a_k) = g(s_k,a_k) + \sum_{k=1}^{\infty}\gamma^k g(s_k,\pi(s_k)). \tag{15.26}$$

Given the loss function

$$\mathcal{L} = \frac{1}{2}(Q^{\pi}(s,a) - \hat{Q}^{\pi}(s,a))^2 \tag{15.27}$$

$$= \frac{1}{2}(g(s,a) + \gamma\hat{Q}^{\pi}(s') - \hat{Q}^{\pi}(s))^2 \tag{15.28}$$

the gradient is

$$\frac{\partial\mathcal{L}}{\partial\hat{Q}^{\pi}} = -(g(s,a) + \gamma\hat{Q}^{\pi,i}(s',\pi(s')) - \hat{Q}^{\pi,i}(s,a)). \tag{15.29}$$

This provides the update law

$$\hat{Q}^{\pi,i+1}(s,a) = \hat{Q}^{\pi,i}(s,a) + \alpha\Big(g(s,a) + \gamma\hat{Q}^{\pi,i}(s',\pi(s')) - \hat{Q}^{\pi,i}(s,a)\Big). \tag{15.30}$$

15.5. Q-Learning

Q-learning attempts to learn the *optimal* action-value function from online data

$$\hat{Q}^{*,i+1} = \hat{Q}^{*,i} + \alpha\Big(g(s,a) + \gamma\min_{a'\in\mathbb{A}}\hat{Q}^{*,i}(s',a') - \hat{Q}^{*,i}(s,a)\Big). \tag{15.31}$$

This formulation is important, as it allows for us to do off-policy learning. That is, we can learn about the optimal policy while following a non-optimal different policy.

© Joseph Moore, 2025

15.6. Fitted Q-Learning

Suppose we want apply Q-Learning in the case of an approximate parameterized Q-function, Q_θ. The loss function is given as

$$\mathcal{L} = \frac{1}{2}(Q(x,a) - Q_\theta(x,a))^2 \tag{15.32}$$

$$= \frac{1}{2}(g(x,a) + \gamma \min_{a' \in U} Q_\theta(x', a') - Q_\theta(x,a))^2 \tag{15.33}$$

$$\frac{\partial \mathcal{L}}{\partial \theta} = \left(g(x,a) + \min_{a' \in U} Q_\theta(x', a') - Q_\theta(x,a)\right)\left(\frac{\gamma \partial Q_\theta(x', a^*)}{\partial \theta} - \frac{\partial Q_\theta(x,a)}{\partial \theta}\right) \tag{15.34}$$

where $a = \arg\max_a Q_\theta(x', a')$ and

$$\theta^{i+1} = \theta^i - \alpha \frac{\partial \mathcal{L}}{\partial \theta}. \tag{15.35}$$

For most formulations, $\min_{a' \in U} Q_\theta(x', a')$ is assumed to be constant so that

$$\frac{\partial \mathcal{L}}{\partial \theta} = \left(g(x,a) + \min_{a' \in U} Q_\theta(x', a') - Q_\theta(x,a)\right)\left(-\frac{\partial Q_\theta(x,a)}{\partial \theta}\right). \tag{15.36}$$

15.7. ϵ-Greedy Policy

To select execute a policy that allows for a trade-off between exploration and exploitation, we can leverage a policy known as the ϵ-greedy policy.

1. With probability 1-ϵ, choose $a = \arg\min_{a \in \mathbb{A}} \hat{Q}(s,a)$
2. With probabilty ϵ, choose a random action.

Often you choose an exponentially decaying ϵ, where $\epsilon \in [0,1]$.

15.8. Deep Q-Learning

One of the most common reinforcement learning algorithms today is known as Deep Q-Learning.

There are a couple of special design decisions that underly Deep Q-Learning. In someways, Deep Q-Learning starts to feel a bit like fitted Q-iteration.

1. The network architecture is leverages the notion of discrete actions.

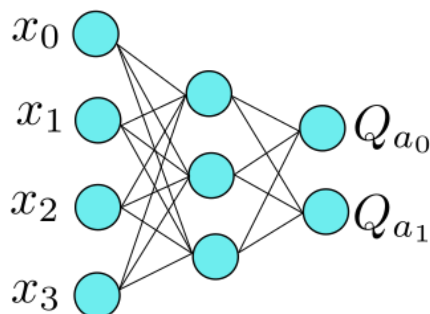

Figure 15.4: The Neural Network Architecture used for Q-Learning.

© Joseph Moore, 2025

2. The algorithm uses two neural networks, a “target” network (Q_t) and a “main” network (Q_m). The main network is updated via supervised learning with Q_t^d, while the target network is updated more slowly, often by copying over the weights from the main network every so often. The updates are:

$$Q_t^d = Q_t + \alpha \left(g(x,a) + \gamma \min_{a' \in \mathbb{A}} Q_t(x',a') - Q_t(x,a) \right) \tag{15.37}$$

and

$$\hat{\theta} = \arg\min_{\theta} \sum_i (Q_m(x_i,u_i,\theta) - Q_t^d(x_i,u_i))^2. \tag{15.38}$$

3. Train in batches, via experience replay, to speed up training. Often the algorithms randomly sample some history of state-action pairs to update the Q-function.

Bibliography

[1] Russ Tedrake. *Underactuated Robotics*. 2023.

[2] Dimitri Bertsekas. *Reinforcement Learning and Optimal Control*. Athena Scientific, 2019.

[3] Drew Bagnell, Byron Boots, and Sanjiban Choudhury. *Modern Adaptive Control and Reinforcement Learning*. 2022.

© Joseph Moore, 2025