Lecture 36: Actor-critic deep reinforcement learning

CC-BY 4.0: copy at will, but cite the source
Mark Hasegawa-Johnson
April 2022


Outline

• Two approaches to solving an MDP
• Two approaches to deep reinforcement learning
• Combining the two, in order to solve the problems with either
Solving an MDP

Remember that, if you know $P(s'|s, a)$, you can solve for the optimum policy $\pi(s)$. This is done by solving Bellman’s equation:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s, a) U(s') \quad \forall s, s'$$

- Bellman’s equation is $N$ nonlinear equations in $N$ unknowns ($N$ is the number of states). In general, the only way to solve it is by exhaustively testing every possible policy ($O\{d^N\}$ computations where $d$ is the number of possible actions).
Two approaches to solving an MDP

We’ve learned two practical algorithms for solving an MDP:
1. Value Iteration: focuses on finding $U(s)$
2. Policy Iteration: focus on finding $\pi(s)$
Two approaches to solving an MDP

Value Iteration: focuses on finding $U(s)$

- Initialize with the value of a length-0 path: $U_0(s) = 0$
- Iterate by finding the best value of a length-$t$ path:

$$U_t(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U_{t-1}(s')$$
**Value Iteration**

\[ U_2(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s, a)U_1(s') \]

**\( U_2(s) (\gamma = 1) \)**

\[
\begin{array}{c|ccc}
| s' | s, down & s, up & s, left & s, right |
|-----|-----------|----------|---------------|---------------|
| \( s \) | \(-0.04\) | \(-0.04\) | \(+0.06\) | \(-0.14\) |
| \(-0.04\) | \(-0.04\) | \(-0.14\) | \(-0.04\) | \(-0.04\) |
| \(-0.04\) | \(-0.04\) | \(-0.14\) | \(-0.04\) | \(-0.04\) |
\end{array}
\]

**\( U_1(s) \)**

\[
\begin{array}{c|cc}
<table>
<thead>
<tr>
<th>s'</th>
<th>s, left &amp; s, right</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s )</td>
<td>(-0.04)</td>
</tr>
<tr>
<td>(-0.04)</td>
<td>(-0.04)</td>
</tr>
<tr>
<td>(-0.04)</td>
<td>(-0.04)</td>
</tr>
</tbody>
</table>
\end{array}
\]
**Value iteration**

Optimal utilities with discount factor 1
(Result of value iteration)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.812</td>
<td>0.868</td>
<td>0.918</td>
<td><strong>+1</strong></td>
</tr>
<tr>
<td>2</td>
<td>0.762</td>
<td><strong>-1</strong></td>
<td>0.660</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.705</td>
<td>0.655</td>
<td>0.611</td>
<td>0.388</td>
</tr>
</tbody>
</table>

Utility estimates

Number of iterations

Final policy
Two approaches to solving an MDP

Policy Iteration: focus on finding $\pi(s)$
- Initialize with a completely arbitrary initial policy, e.g.:
  $$\pi_0(s) = \text{Left}$$
- Iterate:
  - Policy evaluation: find out the value of each state under current policy:
    $$U^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s))U^\pi(s')$$
  - Policy improvement: change the action, in each state, to improve value:
    $$\pi(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s'} P(s'|s, a)U^\pi(s)$$
1. Policy Evaluation:

$$U^{\pi_0}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_0(s)) U^{\pi_0}(s')$$

...write it in matrix form:

$$
\begin{bmatrix}
U^{\pi_0}(1) \\
\vdots \\
U^{\pi_0}(N)
\end{bmatrix} =
\begin{bmatrix}
R(1) \\
\vdots \\
R(N)
\end{bmatrix} + \gamma
\begin{bmatrix}
P(1|1, \pi(1)) & \cdots & P(N|1, \pi(N)) \\
\vdots & \ddots & \vdots \\
P(1|N, \pi(1)) & \cdots & P(N|N, \pi(N))
\end{bmatrix}
\begin{bmatrix}
U^{\pi_0}(1) \\
\vdots \\
U^{\pi_0}(N)
\end{bmatrix}
$$

...and solve it:

$$U^{\pi_0}(s)(s):$$
2. Policy Improvement:

\[ U^{\pi_0}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_0(s)) U^{\pi_0}(s') \]

\[ \pi_1(s) = \arg\max_a \sum_{s'} P(s'|s, \pi_0(s)) U^{\pi_0}(s') \]
Outline

• Two approaches to solving an MDP
• Two approaches to deep reinforcement learning
• Combining the two, in order to solve the problems with either
Two approaches to deep reinforcement learning

• Deep Q learning: train a network to estimate $Q(s,a)$
  – Like value iteration: we focus on $Q(s,a)$, which is closely related to $U(s)$
  – Big problem: $Q(s,a)$ is very noisy, needs lots of smoothing

• Imitation learning: train a network to imitate a human being
  – Like policy iteration: focus directly on estimating $\pi(s)$
  – Big problem: the only way to train this is by imitating a human!
Train the neural network weights in order to minimize the mean-squared error:

\[ \mathcal{L} = \frac{1}{2} E[(f(\vec{s}, \vec{a}) - Q_{\text{local}}(\vec{s}, \vec{a}))^2] \]

\( Q_{\text{local}}(\vec{s}, \vec{a}) \) is the estimated value of the current action:

\[ Q_{\text{local}}(\vec{s}_t, \vec{a}_t) = R_t(\vec{s}_t) + \gamma \max_{\vec{a}_t} f(\vec{s}_{t+1}, \vec{a}') \]
Imitation Learning

If we have $|A|$ possible actions, $1 \leq a \leq |A|$, we could train the network to learn a hidden layer $h(s)$ so that:

$$\pi_a(s) = \frac{\exp(w_a^T h(s))}{\sum_{k=1}^{|A|} \exp(w_k^T h(s))} = P(A = a|S = s)$$

Meaning “the probability that the best action is $a$.”
Two approaches to deep reinforcement learning

• Deep Q learning: train a network to estimate $Q(s,a)$
  – Like value iteration: we focus on $Q(s,a)$, which is closely related to $U(s)$
  – Big problem: $Q(s,a)$ is very noisy, needs lots of smoothing

• Imitation learning: train a network to imitate a human being
  – Like policy iteration: focus directly on estimating $\pi(s)$
  – Big problem: the only way to train this is by imitating a human!
Outline

• Two approaches to solving an MDP
• Two approaches to deep reinforcement learning
• Combining the two, in order to solve the problems with either
The Actor-Critic Algorithm

- Deep Q-learning gives us a network $Q(s,a)$ which is very noisy, so we don’t really want to trust it.
- A policy network can directly estimate $\pi(s)$. The only problem is that we have no way to train it, unless we imitate human behavior.
Actor-critic algorithm

So let’s train two neural nets!

- $Q_t(s, a)$ is the **critic**, and is trained according to the deep Q-learning algorithm (MMSE).

- $\pi_a(s)$ is the **actor**, and is trained to satisfy the critic.
The Actor-Critic Algorithm

Main idea:
• The **actor** is a policy network that decides what action to perform:
  \[ \pi_a(s) = \text{Probability that } a \text{ is the best action in state } s \]
• The **critic** is a deep Q-learning network that estimates the quality of that action (\(Q(s, a)\)).
  \[ Q(s, a) = \text{Expected sum of future rewards if } (s, a) \]
• The critic is noisy, so they don’t get to decide the action. Instead, we only use the critic to help us to train the actor.
The Actor-Critic Algorithm

\[ \pi_a(s) = \text{Probability that } a \text{ is the best action in state } s \]
\[ Q(s, a) = \text{Expected sum of future rewards if } (s, a) \]

• The critic is noisy, so they don’t get to decide the action. Instead, we only use the critic to help us to train the actor.

\[ \mathcal{L} = - \sum_a \pi_a(s) Q(s, a) \]

• The training loss = negative expected sum of future rewards given action \( a \), averaged over the probability with which the actor chooses action \( a \).
The Actor-Critic Algorithm: Forward-Prop

\[ \pi_a(s) = P(a|s) \]

\[ Q_t(s, a) \]
The Actor-Critic Algorithm: Back-Prop

\[ L_{\text{actor}} = -\sum_a \pi_a(s) Q_t(s,a) \]

\[ L_{\text{critic}} = \frac{1}{2} (Q_t(\tilde{s}, \tilde{a}) - Q_{\text{local}}(\tilde{s}, \tilde{a}) )^2 \]
Asynchronous advantage actor-critic (A3C)

TORCS car racing simulation video

Overview: All of the Model-Free Reinforcement Learning Algorithms You’ve Learned

• Policy learning: learn $\pi(s)$ directly
  – Imitation learning

• Q-learning: learn $Q(s, a) = R(s) + \gamma \sum_{s'} P(s'|s, a) U(s')$
  – Table-based: TD, SARSA
  – Deep Q-learning

• Actor-Critic: learn both