CS 440/ECE448 Lecture 32: Model-Based Reinforcement Learning

Mark Hasegawa-Johnson, 4/2024
These slides are in the public domain.

By Nicolas P. Rougier - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=29327040
Outline

• Reinforcement learning
• Model-based learning
• On-policy vs. Off-policy learning
• Exploration vs. Exploitation
Review: Markov Decision Process

• MDP defined by states, actions, transition model, reward function
• The “solution” to an MDP is the policy: what do you do when you’re in any given state
• The Bellman equation tells the utility of any given state, and incidentally, also tells you the optimum policy. The Bellman equation is N nonlinear equations in N unknowns (the policy), therefore it can’t be solved in closed form.

• Value iteration:
  • At the beginning of the (i+1)’st iteration, each state’s value is based on looking ahead i steps in time
  • ... so finding the best action = optimize based on (i+1)-step lookahead

• Policy iteration:
  • Find the utilities that result from the current policy,
  • Improve the current policy
Reinforcement learning: Basic scheme

But what if you don’t know \( P(s'|s, a) \) or \( r(s) \)?
Answer: “learning by doing” (a.k.a. reinforcement learning).

In each time step:
• Take some action
• Observe the outcome of the action: successor state and reward
• Update some internal representation of the environment and policy
Key problems
1. What should you learn?

• Model-based learning: \( P(s'|s, a) \) is estimated using a neural network or probability table, then use value iteration or policy iteration to find the best policy

• Q-learning: \( Q(s, a) \), the quality of action \( a \), is estimated using a neural network or a table of numbers, and directly specifies the best action

• Policy learning: \( \pi(s) \), the policy, is directly estimated using a neural network or a table
Key problems, 2. In which order should you study the states?

- Real-time learning
  - In state $s_t$, try action $a_t$, see what reward $r_t$ state $s_{t+1}$ results, and immediately update your estimates of $r(s_t)$ and $P(s_{t+1}|s_t, a_t)$

- Experience replay buffer
  - In state $s_t$, try action $a_t$, see what reward $r_t$ state $s_{t+1}$ results, and store the tuple $(s_t, a_t, r_t, s_{t+1})$ in an experience replay buffer
  - When the experience replay buffer is full, learn by drawing samples from it according to some criterion that optimizes the rate at which you learn
Key problems, 3. Which actions should you perform while learning?

• On-policy vs. Off-policy learning:
  • On-policy: For each $\pi(s)$, try it, and learn $P(s'|s, a = \pi(s))$
  • Off-policy: Try to learn the values of all possible actions

• Exploration vs. Exploitation
  • Exploration: try actions at random, to see what happens
  • Exploitation: try to act optimally (to maximize value)
Example of model-based reinforcement learning: Theseus the Mouse

Claude Shannon and Theseus the Mouse. Public domain image, Bell Labs.

https://www.youtube.com/watch?v=_9_AEVQ_p74
Model-based reinforcement learning: Theseus’ strategy

Learning phase:

• At each position in the maze \((s)\),
  • For every possible action \(a \in \{\text{Forward, Left, Right, Back}\}\):
    • If the action succeeded in changing the state \((s' \neq s)\), then set
      \[ P(s'|s, a) = 1 \]
    • If not, set \(P(s'|s, a) = 0\) for all \(s' \neq s\)

Once you’ve learned the maze, then compute the best policy \((\pi(s))\) using Value Iteration.

• If \(P(s'|s, a) \in \{0, 1\}\), Value Iteration = BFS
Outline

• Reinforcement learning
• Model-based learning
• On-policy vs. Off-policy learning
• Exploration vs. Exploitation
On-policy learning: Laplace smoothing

• Let’s keep a table of numbers, $N(s, a, s')$, telling how many times action $a$ in state $s$ led to next-state $s'$

• At time $t$, in state $s_t$, choose action $a_t$, observe $r_t$ and $s_{t+1}$, update:

$$N(s_t, a_t, s_{t+1}) += 1$$

$$P(s_{t+1}|s_t, a_t) = \frac{N(s_t, a_t, s_{t+1}) + \lambda}{\sum_{s'\in S} N(s_t, a_t, s') + \lambda |S|}$$

• **On-policy learning**: we only update $P(s_{t+1}|s_t, a_t)$ corresponding to the action that we performed. We don’t learn anything about other actions.
On-policy learning: Neural network

• Estimate the probability table using a softmax:

\[
P(s'|s, a) = \frac{\exp(s^TW_ats')}{\sum_{s'' \in S} \exp(s'^TW_ats'')}
\]

• At time \(t\), in state \(s_t\), choose action \(a_t\), observe \(s_{t+1}\), update:

\[
W_{a_t} \leftarrow W_{a_t} + \eta \nabla_{W_{a_t}} \ln P(s_{t+1}|s_t, a_t)
\]

• **On-policy learning**: we only update \(P(s_{t+1}|s_t, a_t)\) corresponding to the action that we performed. We don’t learn anything about other actions.
Off-policy learning: Neural network

- Estimate the probability table using a softmax:

\[
P(s' | s, a) = \frac{\exp([s^T, a^T]Ws')}{\sum_{s'', \in S} \exp([s^T, a^T]Ws'')}
\]

- At time \(t\), in state \(s_t\), choose action \(a_t\), observe \(s_{t+1}\), update:

\[
W \leftarrow W + \eta \nabla_W \ln P(s_{t+1} | s_t, a_t)
\]

- **Off-policy learning**: By updating \(W\), we modify \(P(s' | s, a)\) for all actions, not just for the one that we performed.
Benefits of On-policy vs. Off-policy learning

• Off-policy learning can converge more quickly because we update $P(s'|s, a)$ for all actions, not just for the one that we performed.

• ...However, off-policy learning might converge to the wrong answer! In the limit, we might be guessing the results of actions we *never* perform!

• Limiting ourselves to on-policy learning usually slows convergence but makes it more stable.
Outline

- Reinforcement learning
- Model-based learning
- On-policy vs. Off-policy learning
- Exploration vs. Exploitation
Exploration vs. Exploitation

**Exploration:** take a new action with unknown consequences
- **Pros:**
  - Get a more accurate model of the environment
  - Discover higher-reward states than the ones found so far
- **Cons:**
  - When you’re exploring, you’re not maximizing your utility
  - Something bad might happen

**Exploitation:** go with the best strategy found so far
- **Pros:**
  - Maximize reward as reflected in the current utility estimates
  - Avoid bad stuff
- **Cons:**
  - Might also prevent you from discovering the true optimal strategy

“Search represents a core feature of cognition:”
* Exploration versus exploitation in space, mind, and society.*
How to trade off exploration vs. exploitation

**Epsilon-first strategy:** when you reach state $s$, check how many times you’ve tested each of its available actions.

- **Explore for the first $N_{first}$ trials:** If the least-explored action has been tested fewer than $N_{first}$ times, then perform that action ($N_{first}$ is an integer).
- **Exploit thereafter:** Once you’ve finished exploring, start exploiting (work to maximize your personal utility).

**Epsilon-greedy strategy:** in every state, every time, forever,

- **With probability $\epsilon$, Explore:** choose any action, uniformly at random.
- **With probability $(1 - \epsilon)$, Exploit:** choose the action with the highest expected utility, according to your current estimates.
- Guarantee: $P(s'|s, a)$ converges to its true value as #trials $\rightarrow \infty$. 
The epsilon-first strategy

The “epsilon-first” strategy tries every action $N_{first} = \frac{1}{\epsilon}$ times, where $\epsilon$ is the desired modeling precision. For example, if we want $|\hat{P}(s'|s,a) - P(s'|s,a)| < 0.1$ ... then we might set $N_{first} = 10$.*

* We can never guarantee that $|\hat{P}(s'|s,a) - P(s'|s,a)| < \epsilon$ with 100% confidence, but using $1/\epsilon$ trials is enough to be pretty confident. If you’ve taken ECE 313 or CS 361, you should be able to work out the relationship more precisely.
The epsilon-first strategy

As you wander through the maze, you reach some state, $s$.

- If there is any action, $a$, for which $N(s, a) < 1/\epsilon$, then try that action.
- If not, then use value iteration (with the current estimates of $P(s'|s, a)$) to decide what is the best action to take.
The epsilon-first strategy

As you wander through the maze, you reach some state, $s$.

• If there is any action, $a$, for which $N(s,a) < 1/\epsilon$, then **explore** (= try the action, to see what it does).

• If not, then **exploit** your knowledge (choose the action that, according to your model, will lead to the highest utility).
The epsilon-greedy strategy

Regardless of how few times or how many times you’ve been in state $s$: generate a uniform random number, $z \in (0,1)$.

- If $z \leq \epsilon$, then **explore**. Choose an action, $a$, uniformly at random, and try it. See what $s'$ results. Increment $N(s, a)$ and $N(s, a, s')$.
  - This happens with probability $\epsilon$.

- If $z > \epsilon$, then **exploit**. Use value iteration or policy iteration to figure out the best action in the current state, then do that action.
  - This happens with probability $1 - \epsilon$. 
Quiz

Try the quiz!

https://us.prairielearn.com/pl/course_instance/147925/assessment/2414902
Compare: Epsilon-first and Epsilon-greedy

For both: \( P(s'|s, a) \approx \frac{N(s, a, s_t)}{N(s, a)} \)

**Advantages of Epsilon-first:**

- In the beginning, when \( P(s'|s, a) \) is still inaccurate, we just try things at random (explore).
- We can choose the level of precision that’s “enough” for us. When \( P(s'|s, a) \) reaches that point, we stop exploring, and instead, we focus on getting the biggest rewards possible (exploit).

**Advantages of Epsilon-greedy:**

- Gradually, over a series of many experiments, \( N(s, a) \to \infty \)
- Therefore, as the number of experiments gets large,

\[ |\hat{P} (s'|s, a) - P(s'|s, a)| \to 0 \]
Outline

• Reinforcement learning

• Model-based learning

\[ P(s_{t+1}|s_t, a_t) = \frac{N(s_t, a_t, s_{t+1}) + \lambda}{\sum_{s' \in S} N(s_t, a_t, s') + \lambda|S|} \]

• On-policy vs. Off-policy learning

\[
W_{a_t} \leftarrow W_{a_t} + \eta \nabla_{W_{a_t}} \ln P(s_{t+1}|s_t, a_t) \\
W \leftarrow W + \eta \nabla_{W} \ln P(s_{t+1}|s_t, a_t)
\]

• Exploration vs. Exploitation

  • Epsilon-first: \( N_{first} = \frac{1}{\epsilon} \)
  • Epsilon-greedy: If \( z \leq \epsilon \), for random number \( z \in (0,1) \), then explore