Reinforcement learning II

Milos Hauskrecht
milos@pitt.edu
5329 Sennott Square
Reinforcement learning

**Objective:** Learn how to act in the environment in order to maximize the reinforcement signal

- The selection of actions should depend on the input
- A policy \( \pi : X \rightarrow A \) maps inputs to actions
- **Goal:** find the optimal policy \( \pi : X \rightarrow A \) that gives the best expected reinforcements

**Example:** learn how to play games (AlphaGo)
Gambling example

• **Game:** 3 biased coins
  – The coin to be tossed is selected randomly from the three coin options. The agent always sees which coin is going to be played next. The agent makes a bet on either a head or a tail with a wage of $1. If after the coin toss, the outcome agrees with the bet, the agent wins $1, otherwise it loses $1

• **RL model:**
  – **Input:** \(X\) – a coin chosen for the next toss,
  – **Action:** \(A\) – choice of head or tail the agent bets on,
  – **Reinforcements:** \(\{1, -1\}\)

• **A policy** \(\pi : X \rightarrow A\)  
  **Example:** \(\pi : \begin{align*}
  1 & \rightarrow \text{head} \\
  2 & \rightarrow \text{tail} \\
  3 & \rightarrow \text{head}
\end{align*}\)
Gambling example

**RL model:**

- **Input:** $X$ – a coin chosen for the next toss,
- **Action:** $A$ – choice of head or tail the agent bets on,
- **Reinforcements:** $\{1, -1\}$
- **A policy** $\pi:

  \[
  \begin{array}{l}
  \text{Coin1} \rightarrow \text{head} \\
  \text{Coin2} \rightarrow \text{tail} \\
  \text{Coin3} \rightarrow \text{head}
  \end{array}
  \]

**State, action, reward trajectories**

<table>
<thead>
<tr>
<th></th>
<th>Step0</th>
<th>Step1</th>
<th>Step2</th>
<th>Step k</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>state</strong></td>
<td>Coin2</td>
<td>Coin1</td>
<td>Coin2</td>
<td>Coin1</td>
</tr>
<tr>
<td><strong>action</strong></td>
<td>Tail</td>
<td>Head</td>
<td>Tail</td>
<td>Head</td>
</tr>
<tr>
<td><strong>reward</strong></td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
**Gambling example**

**Learning goal:** find the optimal policy

\[ \pi^*: X \rightarrow A \]

\[ \pi^* : \begin{array}{c|c}
  \text{Coin1} & ? \\
  \text{Coin2} & ? \\
  \text{Coin3} & ? \\
\end{array} \]

that maximizes future expected rewards

\[ E\left( \sum_{t=0}^{T} \gamma^t r_t \right) \quad 0 \leq \gamma < 1 \]

a discount factor = present value of money
Agent navigation example

- **Agent navigation in the maze:**
  - 4 moves in compass directions
  - Effects of moves are stochastic – we may wind up in other than intended location with a non-zero probability
  - **Objective:** learn how to reach the goal state in the shortest expected time
Agent navigation example

- **The RL model:**
  - **Input:** $X$ – a position of an agent
  - **Output:** $A$ – the next move
  - **Reinforcements:** $R$
    - -1 for each move
    - +100 for reaching the goal
  - **A policy:** $\pi : X \rightarrow A$
    - $\pi :$
      - Position 1 $\rightarrow$ right
      - Position 2 $\rightarrow$ right
      - ... 
      - Position 25 $\rightarrow$ left

- **Goal:** find the policy maximizing future expected rewards
  \[
  E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right) \\
  \text{with } 0 \leq \gamma < 1
  \]
Agent navigation example

State, action reward trajectories

- policy

$$\pi : \begin{align*}
\text{Position 1} & \rightarrow \text{right} \\
\text{Position 2} & \rightarrow \text{right} \\
... & \\
\text{Position 25} & \rightarrow \text{left}
\end{align*}$$

<table>
<thead>
<tr>
<th>Step0</th>
<th>Step1</th>
<th>Step2</th>
<th>Step k</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>Pos1</td>
<td>Pos2</td>
<td>Pos3</td>
</tr>
<tr>
<td>action</td>
<td>Right</td>
<td>Right</td>
<td>Up</td>
</tr>
<tr>
<td>reward</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

G moves
RL with immediate rewards

- **Expected reward**

  \[ E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right) \quad 0 \leq \gamma < 1 \]

- **Immediate reward case:**
  - Reward depends only on \( x \) and the action choice
  - The action does not affect the environment and hence future inputs (states) and future rewards:

    \[ E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right) = E(r_0) + E(\gamma r_1) + E(\gamma^2 r_2) + \ldots \]

    \[ r_0, r_1, r_2 \ldots \] Rewards for every step of the game

  - Expected one step reward for input \( x \) (**coin to play next**) and the choice \( a \) : \( R(x, a) \)
RL with immediate rewards

- **Expected reward**

\[ E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right) = E(r_0) + E(\gamma r_1) + E(\gamma^2 r_2) + \ldots \]

- **Optimal strategy:**

\[ \pi^* : X \rightarrow A \]

\[ \pi^*(x) = \arg \max_a R(x, a) \]

\[ R(x, a) : \text{Expected one step reward for input } x \text{ (coin to play next) and the choice } a \]
RL with immediate rewards

The optimal choice assumes we know the expected reward \( R(x, a) \)

- Then: \( \pi^*(x) = \arg \max_a R(x, a) \)

Caveats

- We do not know the expected reward \( R(x, a) \)
  - We need to estimate it using \( \tilde{R}(x, a) \) from interaction
- We cannot determine the optimal policy if the estimate of the expected reward is not good
  - We need to try also actions that look suboptimal wrt the current estimates of \( \tilde{R}(x, a) \)
Estimating $R(x, a)$

- **Solution 1:**
  - For each input $x$ try different actions $a$
  - Estimate $R(x, a)$ using the average of observed rewards

$$\tilde{R}(x, a) = \frac{1}{N_{x,a}} \sum_{i=1}^{N_{x,a}} r_{i,x,a}$$

- **Solution 2:** online approximation
  - Updates an estimate after performing action $a$ in $x$ and observing the reward $r_{x,a}$

$$\tilde{R}(x, a)^{(i)} \leftarrow (1 - \alpha(i))\tilde{R}(x, a)^{(i-1)} + \alpha(i) r_{i,x,a}$$

$\alpha(i)$ - a learning rate
RL with immediate rewards

- At any step in time $i$ during the experiment we have estimates of expected rewards for each $(coin, action)$ pair:
  
  $\tilde{R}(coin1, head)^{(i)}$
  $\tilde{R}(coin1, tail)^{(i)}$
  $\tilde{R}(coin2, head)^{(i)}$
  $\tilde{R}(coin2, tail)^{(i)}$
  $\tilde{R}(coin3, head)^{(i)}$
  $\tilde{R}(coin3, tail)^{(i)}$

- Assume the next coin to play in step $(i+1)$ is coin 2 and we pick head as our bet. Then we update $\tilde{R}(coin2, head)^{(i+1)}$ using the observed reward and one of the update strategy above, and keep the reward estimates for the remaining $(coin, action)$ pairs unchanged, e.g. $\tilde{R}(coin2, tail)^{(i+1)} = \tilde{R}(coin2, tail)^{(i)}$
Exploration vs. Exploitation

• **Epsilon greedy exploration:**
  – Uses exploration parameter $0 \leq \varepsilon \leq 1$
  – Choose the “current” best choice with probability $1 - \varepsilon$
    \[
    \hat{\pi}(x) = \arg \max_{a \in A} \tilde{R}(x, a)
    \]
  – All other choices are selected with a uniform probability $\varepsilon / |A| - 1$

**Advantages:**
• Simple, easy to implement

**Disadvantages:**
• Exploration more appropriate at the beginning when we do not have good estimates of $\tilde{R}(x, a)$
• Exploitation more appropriate later when we have good estimates
Exploration vs. Exploitation

• **Boltzmann exploration**
  – The action is chosen randomly but proportionally to its current expected reward estimate
  – Can be tuned with a temperature parameter T to promote exploration or exploitation

• Probability of choosing action \(a\)

\[
p(a \mid x) = \frac{\exp[\tilde{R}(x, a)/T]}{\sum_{a'\in A} \exp[\tilde{R}(x, a')/T]}
\]

• **Effect of T:**
  – For high values of \(T\), \(p(a \mid x)\) is uniformly distributed for all actions
  – For low values of \(T\), \(p(a \mid x)\) of the action with the highest value of \(\tilde{R}(x, a)\) is approaching 1
RL with delayed rewards

A more general reinforcement learning model

• Agent navigation in the Maze:
  – 4 moves in compass directions
  – Effects of moves are stochastic – we may wind up in other than intended location with non-zero probability
  – Objective: reach the goal state in the shortest time
Agent navigation example

**State, action reward trajectories**

- **policy**

\[ \pi : \begin{array}{c|c}
\text{Position 1} & \text{right} \\
\text{Position 2} & \text{right} \\
... & \\
\text{Position 25} & \text{left} \\
\end{array} \]

<table>
<thead>
<tr>
<th>state</th>
<th>Step 0</th>
<th>Step 1</th>
<th>Step 2</th>
<th>Step k</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Pos1</td>
<td>Pos2</td>
<td>Pos3</td>
<td>Pos15</td>
</tr>
<tr>
<td>Right</td>
<td>Right</td>
<td>Up</td>
<td>Up</td>
<td>Up</td>
</tr>
<tr>
<td>reward</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

moves
Learning with delayed rewards

- Actions, in addition to immediate rewards affect the next state of the environment and thus indirectly also future rewards
- We need a model to represent environment changes
- The model we use is called **Markov decision process (MDP)**
  - Frequently used in AI, OR, control theory
  - **Markov assumption**: next state depends on the previous state and action, and not states (actions) in the past
Markov decision process

Formal definition: 4-tuple \((S, A, T, R)\)

- A set of states \(S\) \((X)\) locations of a robot
- A set of actions \(A\) move actions
- Transition model \(S \times A \times S \rightarrow [0,1]\) where can I get with different moves
- Reward model \(S \times A \times S \rightarrow \mathbb{R}\) reward/cost for a transition
Markov decision process

**Formal definition:** 4-tuple $(S, A, T, R)$

**Transition model $T$:** For each action define a probability of reaching the next state from the current state

**Example:** 3 states

<table>
<thead>
<tr>
<th>Current state</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>2</td>
<td>0.1</td>
<td>0.5</td>
<td>0.4</td>
</tr>
<tr>
<td>3</td>
<td>0.3</td>
<td>0.5</td>
<td>0.2</td>
</tr>
</tbody>
</table>
Markov decision process

Formal definition: 4-tuple \((S, A, T, R)\)

Reward model \(R\): For each action define a reward associated with the transition

Example: 3 states

<table>
<thead>
<tr>
<th>Current state</th>
<th>Next state</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
MDP problem

• We want to find the best policy $\pi^* : S \rightarrow A$

• Value function ($V$) for a policy, quantifies the goodness of a policy through, e.g. infinite horizon, discounted model

$$E(\sum_{t=0}^{\infty} \gamma^t r_t)$$

It:
1. combines future rewards over a trajectory
2. combines rewards for multiple trajectories (through expectation-based measures)
Value of a policy for MDP

• Assume a fixed policy \( \pi : S \to A \)

• How to compute the value of a policy under infinite horizon discounted model?

A fixed point equation:

\[
V^\pi (s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi(s))V^\pi (s')
\]

- For a finite state space— we get a set of linear equations

\[
v = r + Uv \quad \Rightarrow \quad v = (I - U)^{-1} r
\]
Optimal policy

• **The value of the optimal policy**

\[
V^*(s) = \max_{a \in A} \left[ R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s') \right]
\]

- expected one step reward for the first action
- expected discounted reward for following the opt. policy for the rest of the steps

Value function mapping form:

\[
V^*(s) = (HV^*) (s)
\]

• **The optimal policy**: \( \pi^* : S \rightarrow A \)

\[
\pi^*(s) = \arg \max_{a \in A} \left[ R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s') \right]
\]
Computing optimal policy

**Dynamic programming:**  **Value iteration:**
- computes the optimal value function first then the policy
- iterative approximation
- converges to the optimal value function

**Value iteration** (\(\varepsilon\))

**initialize**  
\[ V \]

**repeat**

set  
\[ V' \leftarrow V \]

set  
\[ V \leftarrow HV' \]

until  
\[ \|V' - V\|_\infty \leq \varepsilon \]

**output**  
\[ \pi^*(s) = \arg\max_{a \in A} \left[ R(s, a) + \gamma \sum_{s' \in S} P(s' | s, a)V(s') \right] \]
Reinforcement learning of optimal policies

• **In the RL framework we do not know the MDP model !!!**

• **Goal**: learn the optimal policy

\[ \pi^* : S \rightarrow A \]

• **Two basic approaches:**
  – **Model based learning**
    • Learn the MDP model (probabilities, rewards) first
    • Solve the MDP afterwards
  – **Model-free learning**
    • Learn how to act directly
    • No need to learn the parameters of the MDP
  – A number of clones of the two in the literature
Model-based learning

• We need to learn transition probabilities and rewards

• Learning of probabilities
  – ML parameter estimates
  – Use counts
    \[
    \tilde{P}(s'|s,a) = \frac{N_{s,a,s'}}{N_{s,a}}
    \]
    \[
    N_{s,a} = \sum_{s' \in S} N_{s,a,s'}
    \]

• Learning rewards
  – Similar to learning with immediate rewards
    \[
    \tilde{R}(s,a) = \frac{1}{N_{s,a}} \sum_{i=1}^{N_{s,a}} r_i^{s,a}
    \]
    or the online solution

• Problem: changes in the probabilities and reward estimates would require us to solve an MDP from scratch! (after every action and reward seen)
Model free learning

• **Motivation:** value function update (value iteration):

\[
V^*(s) \leftarrow \max_{a \in A} \left[ R(s, a) + \gamma \sum_{s' \in S} P(s' | s, a)V^*(s') \right]
\]

• Let

\[
Q(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s' | s, a)V^*(s')
\]

• Then \( V^*(s) \leftarrow \max_{a \in A} Q(s, a) \)

• Note that the update can be defined purely in terms of Q-functions

\[
Q(s, a) \leftarrow R(s, a) + \gamma \sum_{s' \in S} P(s' | s, a) \max_{a'} Q(s', a')
\]
Q-learning

- **Q-learning** uses the Q-value update idea
  - **But** relies on a stochastic (on-line, sample by sample) update

\[ Q(s, a) \leftarrow R(s, a) + \gamma \sum_{s' \in S} P(s' | s, a) \max_{a'} Q(s', a') \]

is replaced with

\[ \hat{Q}(s, a) \leftarrow (1 - \alpha) \hat{Q}(s, a) + \alpha \left( r(s, a) + \gamma \max_{a'} \hat{Q}(s', a') \right) \]

- \( r(s, a) \) - reward received from the environment after performing an action \( a \) in state \( s \)
- \( s' \) - new state reached after action \( a \)
- \( \alpha \) - learning rate, a function of \( N_{s,a} \)
  - a number of times \( a \) has been executed at \( s \)
Q-function updates in Q-learning

- At any step in time \( i \) during the experiment we have estimates of Q functions for each \((state, action)\) pair:

\[
\tilde{Q}(\text{position1, up})^{(i)} \\
\tilde{Q}(\text{position1, left})^{(i)} \\
\tilde{Q}(\text{position1, right})^{(i)} \\
\tilde{Q}(\text{position1, down})^{(i)} \\
\tilde{Q}(\text{position2, up})^{(i)} \\
\cdots
\]

- Assume the current state is \textit{position 1} and we pick \textit{up} action to be performed next.
- After we observe the reward, we update \( \tilde{Q}(\text{position1, up}) \), and keep the Q function estimates for the remaining \((state, action)\) pairs unchanged.
Q-learning

The on-line update rule is applied repeatedly during the direct interaction with an environment.

Q-learning
initialize $Q(s,a) = 0$ for all $s,a$ pairs
observe current state $s$
repeat
    select action $a$; use some exploration/exploitation schedule
    receive reward $r$
    observe next state $s'$
    update $Q(s,a) \leftarrow (1 - \alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$
    set $s$ to $s'$
end repeat
Q-learning convergence

The **Q-learning is guaranteed to converge** to the optimal Q-values under the following conditions:

- Every state is visited and every action in that state is tried infinite number of times
  - This is assured via exploration/exploitation schedule
- The sequence of learning rates for each $Q(s,a)$ satisfies:

  1. $\sum_{i=1}^{\infty} \alpha(i) = \infty$
  2. $\sum_{i=1}^{\infty} \alpha(i)^2 < \infty$

  $\alpha(n(s,a))$ - is the learning rate for the $n$th trial of $(s,a)$
RL with delayed rewards

The optimal choice \( \pi^*(s) = \arg \max_a Q(s, a) \)

- much like what we had for the immediate rewards
  \( \pi^*(x) = \arg \max_a R(x, a) \)

**RL Learning**

- Instead of exact values of \( Q(s, a) \) we use \( \hat{Q}(s, a) \)

\[
\hat{Q}(s, a) \leftarrow (1 - \alpha) \hat{Q}(s, a) + \alpha \left( r(s, a) + \gamma \max_{a'} \hat{Q}(s', a') \right)
\]

- Since we have only estimates of \( \hat{Q}(s, a) \)
  - We need to try also actions that look suboptimal wrt the current estimates
  - **Exploration/exploitation strategies**
    - Epsilon greedy exploration
    - Boltzmann exploration
Q-learning speed-ups

• The basic Q-learning rule updates may propagate distant (delayed) rewards very slowly

Example:

• **Goal:** a high reward state
• To make the correct decision we need all Q-values for the current position to be good
• **Problem:**
  – in each run we back-propagate values only ‘one-step’ back. It takes multiple trials to back-propagate values multiple steps.
Q-learning speed-ups

- **Remedy:** Backup values for a larger number of steps

Rewards from applying the policy

\[ q_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \]

We can substitute (immediate rewards with n-step rewards):

\[ q_t^n = \sum_{i=0}^{n} \gamma^i r_{t+i} + \gamma^{n+1} \max_{a'} Q_{t+n} (s', a') \]

Postpone the update for \( n \) steps and update with a longer trajectory rewards

\[ Q_{t+n+1} (s, a) \leftarrow Q_{t+n} (s, a) + \alpha \left( q_t^n - Q_{t+n} (s, a) \right) \]

**Problems:**
- larger variance
- exploration/exploitation switching
- wait \( n \) steps to update
Q-learning speed-ups

• One step vs. n-step backup

Problems with n-step backups:
- larger variance
- exploration/exploitation switching
- wait n steps to update
Q-learning speed-ups

- **Temporal difference: TD(\(\lambda\)) method**
  - Remedy of the wait n-steps problem
  - Partial back-up after every simulation step
    - Similar idea: weather forecast adjustment

Implemented with *eligibility traces*
RL successes

- Reinforcement learning is relatively simple
  - On-line techniques can track non-stationary environments and adapt to its changes

- Successful applications:
  - Deep Mind’s AlphaGo (Alpha Zero)
  - TD Gammon – learned to play backgammon on the championship level
  - Elevator control
  - Dynamic channel allocation in mobile telephony
  - Robot navigation in the environment