Reinforcement learning II

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

- **We want to learn the control policy:** \( \pi : X \rightarrow A \)
- We see examples of \( x \) (but outputs \( a \) are not given)
- Instead of \( a \) we get a feedback (reinforcement, reward) from a critic quantifying how good the selected output was

- The reinforcements may not be deterministic
- **Goal:** find \( \pi : X \rightarrow A \) with the best expected reinforcements
Gambling example.

- **Game:** 3 different biased coins are tossed
  - The coin to be tossed is selected randomly from the three options and I always see which coin I am going to play next
  - I make bets on head or tail and I always wage $1
  - If I win I get $1, otherwise I lose my bet

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

- **A policy** $\pi : X \rightarrow A$

  Example: $\pi : \begin{array}{|c|c|} \hline \text{Coin1} & \text{head} \\ \text{Coin2} & \text{tail} \\ \text{Coin3} & \text{head} \\ \end{array}$
Gambling example

• **RL model:**
  – **Input:** $X$ – a coin chosen for the next toss,
  – **Action:** $A$ – choice of head or tail,
  – **Reinforcements:** $\{1, -1\}$
  – **A policy** $\pi$:
    
    \[
    \begin{array}{l|l}
    \text{Coin1} & \text{head} \\
    \text{Coin2} & \text{tail} \\
    \text{Coin3} & \text{head} \\
    \end{array}
    \]

• **Learning goal:** find $\pi : X \rightarrow A$

  maximizing future expected profits

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

  $\gamma$ 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 non-zero probability
  - **Objective:** reach the goal state in the shortest expected time
Agent navigation example

- **The RL model:**
  - **Input:** \( X \) – position of an agent
  - **Output:** \( A \) – a 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 20 \( \rightarrow left \)

- **Goal:** find the policy maximizing future expected rewards
  
  \[
  E\left(\sum_{t=0}^{\infty} \gamma^t r_t \right)
  \]
Objectives of RL learning

• **Objective:**
  Find a mapping \( \pi^* : X \rightarrow A \)
  That maximizes some combination of future reinforcements (rewards) received over time

• **Valuation models** (quantify how good the mapping is):
  – **Finite horizon model**
    \[
    E\left( \sum_{t=0}^{T} r_t \right) \quad \text{Time horizon: } T > 0
    \]
  – **Infinite horizon discounted model**
    \[
    E\left( \sum_{t=0}^{\infty} \gamma^t r_t \right) \quad \text{Discount factor: } 0 < \gamma < 1
    \]
  – **Average reward**
    \[
    \lim_{T \to \infty} \frac{1}{T} E\left( \sum_{t=0}^{T} r_t \right)
    \]
RL with immediate rewards

- **Expected reward**

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

- **Optimizing the expected reward**

\[
\max_{\pi} E(\sum_{t=0}^{\infty} \gamma^t r_t) = \max_{\pi} \sum_{t=0}^{\infty} \gamma^t E(r_t) = \max_{\pi} \sum_{t=0}^{\infty} \gamma^t R(\pi) = \max_{\pi} R(\pi)(\sum_{t=0}^{\infty} \gamma^t)
\]

\[
= (\sum_{t=0}^{\infty} \gamma^t) \max_{\pi} R(\pi)
\]

\[
\max_{\pi} R(\pi) = \max_{\pi} \sum_x R(x, \pi(x)) P(x) = \sum_x P(x)[\max_{\pi(x)} R(x, \pi(x))]
\]

- **Optimal strategy**: \( \pi^* : X \rightarrow A \)

\[
\pi^*(x) = \arg \max_a R(x, a)
\]
RL with immediate rewards

- **Problem:** In the RL framework we do not know $R(x, a)$
  - The expected reward for performing action $a$ at input $x$
- **Solution:**
  - For each input $x$ try different actions $a$
  - Estimate $R(x, a)$ using the average of observed rewards
    \[
    \tilde{R}(x, a) = \frac{1}{N_{x,a}} \sum_{i=1}^{N_{x,a}} r_{i}^{x,a}
    \]
  - Action choice $\pi(x) = \arg \max_a \tilde{R}(x, a)$
  - Accuracy of the estimate: statistics (Hoeffding’s bound)
    \[
    P\left(\left| \tilde{R}(x, a) - R(x, a) \right| \geq \varepsilon \right) \leq \exp\left[-\frac{2\varepsilon^2 N_{x,a}}{(r_{\max} - r_{\min})^2}\right] \leq \delta
    \]
  - Number of samples:
    \[
    N_{x,y} \geq \frac{(r_{\max} - r_{\min})^2}{2\varepsilon^2} \ln \frac{1}{\delta}
    \]
RL with immediate rewards

- **On-line (stochastic approximation)**
  - An alternative way to estimate $R(x, a)$

- **Idea:**
  - choose action $a$ for input $x$ and observe a reward $r^{x,a}$
  - Update an estimate

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

  $\alpha$ - a learning rate

- **Convergence property:** The approximation converges in the limit for an appropriate learning rate schedule.

- **Assume:** $\alpha(n(x, a))$ - is a learning rate for $n$th trial of $(x, a)$ pair

- Then the converge is assured if:

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

- **Uniform exploration**
  - 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
  $$p(a \mid x) = \frac{\varepsilon}{|A| - 1}$$

- **Boltzman exploration**
  - The action is chosen randomly but proportionally to its current expected reward estimate
  $$p(a \mid x) = \frac{\exp[\tilde{R}(x, a) / T]}{\sum_{a' \in A} \exp[\tilde{R}(x, a') / T]}$$

   $T$ – is temperature parameter. **What does it do?**
RL with delayed rewards

- **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
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

```
action_{t-1} -> state_{t-1} -> state_t
  |                          |
  | reward_{t-1}            |
```

CS 2750 Machine Learning
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
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

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

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 \( \pi : S \rightarrow A \)

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

Fixed point equation:

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

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

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

- For a finite state space— we get a set of linear equations
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** \( V \);; \( V \) is vector of values for all states

**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 or Bayesian parameter estimates
  - Use counts
    \[
    \tilde{P}(s'|s,a) = \frac{N_{s,a,s'}}{N_{s,a}} \quad N_{s,a} = \sum_{s' \in S} N_{s,a,s'}
    \]
- **Learning rewards**
  - Similar to learning with immediate rewards
    \[
    \tilde{R}(s,a) = \frac{1}{N_{s,a}} \sum_{i=1}^{N_{s,a}} r_{i}^{s,a}
    \]
- **Problem:** on-line update of the policy
  - would require us to solve the MDP after every update !!
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 \( a \) executed at \( s \)
Q-learning

The on-line update rule is applied repeatedly during 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 $(s,a)$
Exploration vs. Exploitation

• In the RL with the delayed rewards
  – At any point in time the learner has an estimate of $\hat{Q}(x, a)$ for any state action pair

• Dilemma:
  – Should the learner use the current best choice of action (exploitation)
    \[ \hat{\pi}(x) = \arg \max_{a \in A} \hat{Q}(x, a) \]
  – Or choose other action $a$ and further improve its estimate of $\hat{Q}(x, a)$ (exploration)

• Exploration/exploitation strategies
  – Uniform exploration
  – Boltzman 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) method**
  - Remedy of the wait $n$-steps problem
  - Partial back-up after every simulation step
    - Similar idea: weather forecast adjustment

Different versions of this idea have been implemented
RL successes

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

• Successful applications:
  – TD Gammon – learned to play backgammon on the championship level
  – Elevator control
  – Dynamic channel allocation in mobile telephony
  – Robot navigation in the environment