Reinforcement learning

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

- The reinforcements may not be deterministic
- **Goal:** find \( \pi : X \to A \) 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** $\pi : X \to A$

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

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** $\pi : \begin{array}{c|c}
\text{Coin1} & \text{head} \\
\text{Coin2} & \text{tail} \\
\text{Coin3} & \text{head} \\
\end{array}$

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

maximizing future expected profits

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

\[ \begin{array}{c}
\text{Position 1} \rightarrow \text{right} \\
\text{Position 2} \rightarrow \text{right} \\
\vdots \\
\text{Position 20} \rightarrow \text{left}
\end{array} \]

• 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:** \( \pi : X \rightarrow A \)

\[ \pi : \begin{cases}
\text{Position 1} \rightarrow \text{right} \\
\text{Position 2} \rightarrow \text{right} \\
\vdots \\
\text{Position 20} \rightarrow \text{left}
\end{cases} \]

• **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**
    \[
    \lim_{T \rightarrow \infty} \frac{1}{T} E \left( \sum_{t=0}^{T} r_t \right)
    \]

Exploration vs. Exploitation

- The (learner) actively interacts with the environment:
  - At the beginning the learner does not know anything about the environment
  - It gradually gains the experience and learns how to react to the environment

- **Dilemma (exploration-exploitation):**
  - After some number of steps, should I select the best current choice (**exploitation**) or try to learn more about the environment (**exploration**)?
  - **Exploitation** may involve the selection of a sub-optimal action and prevent the learning of the optimal choice
  - **Exploration** may spend too much time on trying bad currently suboptimal actions
Effects of actions on the environment

Effect of actions on the environment (next input $x$ to be seen)

• No effect, the distribution over possible $x$ is fixed; action consequences (rewards) are seen immediately,
• Otherwise, distribution of $x$ can change; the rewards related to the action can be seen with some delay.

Leads to two forms of reinforcement learning:

• Learning with immediate rewards
  – Gambling example
• Learning with delayed rewards
  – Agent navigation example; move choices affect the state of the environment (position changes), a big reward at the goal state is delayed

---

RL with immediate rewards

• 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: $A$ – head or tail bet
  – Reinforcements: $\{1, -1\}$
• Learning goal: find $\pi : X \rightarrow A$
  maximizing the future expected profits over time
  \[
  E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right) \quad \gamma \text{ a discount factor = present value of money}
  \]
RL with immediate rewards

- **Expected reward**
  \[ E(\sum_{t=0}^{\infty} \gamma^t r_t) \]  
  \( \gamma \) - a discount factor = present value of money

- **Immediate reward case:**
  - Reward for the choice becomes available immediately
  - Our choice does not affect environment and thus future rewards
  \[
  E\left(\sum_{t=0}^{\infty} \gamma^t r_t \right) = E\left(r_0\right) + E\left(\gamma r_1\right) + E\left(\gamma^2 r_2\right) + \ldots
  \]
  \( r_0, r_1, r_2 \ldots \) Rewards for every step
  - Expected one step reward for input \( x \) and the choice \( a \):
  \[ R(x, a) \]

---

RL with immediate rewards

**Immediate reward case:**

- Reward for the choice \( a \) becomes available immediately
- Expected reward for the input \( x \) and choice \( a \): \( R(x, a) \)
  - For the gambling problem it can be defined as:
  \[
  R(x, a_i) = \sum_j r(\omega_j | a_i, x) P(\omega_j | x, a_i)
  \]
  - \( \omega_j \) - a future outcome of the coin toss
  - Recall the definition of the expected loss
- **Expected one step reward for a strategy** \( \pi : X \rightarrow A \)
  \[
  R(\pi) = \sum_x R(x, \pi(x)) P(x)
  \]
  \( R(\pi) \) is the expected reward for \( r_0, r_1, r_2 \ldots \)
RL with immediate rewards

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

- **Optimizing the expected reward**
  \[
  \max_{\pi} E \left( \sum_{t=0}^{\infty} \gamma^t r_t \right) = \max_{\pi} \sum_{t=0}^{\infty} \gamma^t R(\pi) = \max_{\pi} \sum_{t=0}^{\infty} \gamma^t = \sum_{t=0}^{\infty} \gamma^t \\
  \max_{\pi} R(\pi) = \max_{\pi} \sum_{x} R(x, \pi(x)) P(x) = \sum_{x} P(x) \left[ \max_{\pi(x)} R(x, \pi(x)) \right]
  \]

**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
    
    $$
    \tilde{R}(x, a) = \frac{1}{N_{x,a}} \sum_{i=1}^{N_{x,a}} r_{i,x,a}^a
    $$
  - 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:
    
    $$
    N_{x,a} \geq \frac{(r_{\max} - r_{\min})^2}{2\varepsilon^2} \ln \frac{1}{\delta}
    $$

---

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

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

    $\alpha$ - 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

• In the RL framework
  – the (learner) actively interacts with the environment.
  – At any point in time it has an estimate of \( \tilde{R}(x, a) \) for any input action pair

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

• Different exploration/exploitation strategies exist

---

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 \( \frac{\varepsilon}{|A| - 1} \)

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

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

<table>
<thead>
<tr>
<th>A set of states</th>
<th>(S) ((X))</th>
<th>locations of a robot</th>
</tr>
</thead>
<tbody>
<tr>
<td>A set of actions</td>
<td>(A)</td>
<td>move actions</td>
</tr>
<tr>
<td>Transition model</td>
<td>(S \times A \times S \rightarrow [0,1])</td>
<td>where can I get with different moves</td>
</tr>
<tr>
<td>Reward model</td>
<td>(S \times A \times S \rightarrow \mathbb{R})</td>
<td>reward/cost for a transition</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

\[
E(\sum_{i=0}^{\infty} \gamma^i r_i)
\]

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:

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

- 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

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

Value function mapping form:

$$V^*(s) = (HV^*)(s)$$

• The optimal policy:

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

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
    \[ \tilde{P}(s'|s,a) = \frac{N_{s,a,s'}}{N_{s,a}} \]
    \[ N_{s,a} = \sum_{s' \in S} N_{s,a,s'} \]
- **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 an 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

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

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

  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
  - Boltzmann exploration
Q-learning speed-ups

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

![Diagram](image)

**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 has 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