CS-UY 6923: Lecture 14
Reinforcement Learning

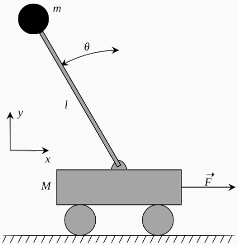
NYU Tandon School of Engineering, Prof. Christopher Musco

Today: Give flavor of the area and insight into one algorithm (Q-learning) which has been successful in recent years.

Basic setup:

- **Agent** interacts with **environment** over time $1, \ldots, t$.
- Takes repeated sequence of **actions**, $a_1, \ldots, a_t$ which effect the environment.
- **State** of the environment over time denoted $s_1, \ldots, s_t$.
- Earn **rewards** $r_1, \ldots, r_t$ depending on actions taken and states reached.
- Goal is to maximize reward over time.

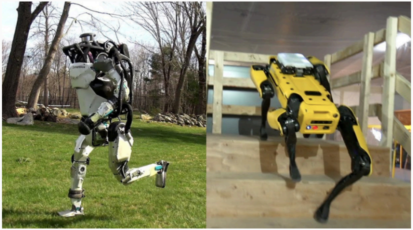
Classic inverted pendulum problem:



- **Agent:** Cart/software controlling cart.
- **State:** Position of the car, pendulum head, etc.
- **Actions:** Move cart left or move right.
- **Reward:** 1 for every time step that $|\theta| < 90°$ (pendulum is upright). 0 when $|\theta| = 90°$

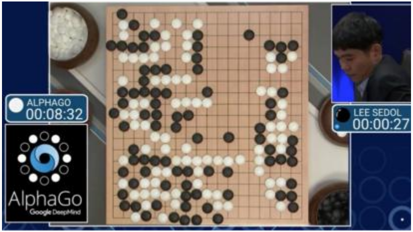
This problem has a long history in **Control Theory.** Other applications of classical control:

- Semi-autonomous vehicles (airplanes, helicopters, rockets, etc.)
- Industrial processes (e.g. controlling large chemical reactions)
- Robotics



control theory : reinforcement learning :: stats : machine learning

Strategy games, like Go:

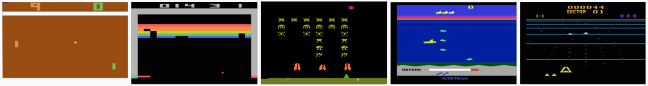


- **State:** Position of all pieces on board.
- **Actions:** Place new piece.
- **Reward:** 1 if in winning position at time t. 0 otherwise.

This is a sparse reward problem. Payoff only comes after many times steps, which makes the problem very challenging.

REINFORCEMENT LEARNING EXAMPLES

Video games, like classic Atari games:



- **State:** Raw pixels on the screen (sometimes there is also hidden state which can't be observed by the player).
- **Actions:** Actuate controller (up,down,left,right,click).
- **Reward:** 1 if point scored at time t.

Model problem as a **Markov Decision Process (MDP)**:

- $\mathcal{S}$: Set of all possible states. $|\mathcal{S}|$.
- $\mathcal{A}$: Set of all possible actions. $|\mathcal{A}|$.
- $\mathcal{R}$: Set of possible rewards. Could have $\mathcal{R} = \mathbb{R}$.
- **Reward function**
  $R(s, a) : \mathcal{S} \times \mathcal{A} \to$ probability distribution over $\mathcal{R}$. $r_t \sim R(s_t, a_t)$.
- **State transition function**
  $P(s, a) : \mathcal{S} \times \mathcal{A} \to$ probability distribution over $\mathcal{S}$. $s_{t+1} \sim P(s_t, a_t)$.

Why is this called a Markov decision process? What does the term Markov refer to?

Goal: Find a **policy** $\Pi : \mathcal{S} \rightarrow \mathcal{A}$ from states to actions which maximize expected cumulative reward.

- Start is state s_0.
- For $t = 0 \ldots, T$
  - $r_t \sim R(s_t, \Pi(s_t))$.
  - $s_{t+1} \sim P(s_t, \Pi(s_t))$.

The **time horizon** T could be short (game with fixed number of steps), very long (stock investing), or infinite. Goal is to maximize:

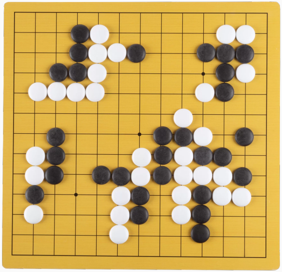
$$reward(\Pi) = \mathbb{E} \sum_{t=0}^{T} r_t$$

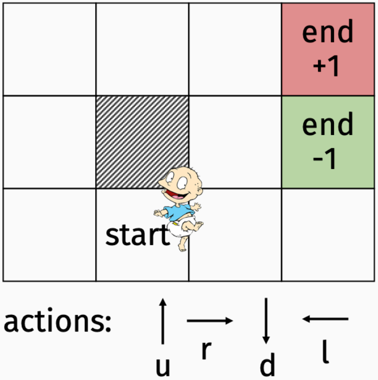
$[s_0, a_0, r_0], [s_1, a_1, r_1], \ldots, [s_t, a_t, r_t]$ is called a **trajectory** of the MDP under policy Π.[1]

[1] It is not a priori clear that a fixed policy makes sense. Maybe we could get better reward by changing the policy over time. We will discuss this shortly.

FLEXIBILITY OF MDPS

- Can be used to model time-varying environments. Just add time t to the state vector.
- Can be used to model games where actions have different effect if play in sequence (e.g. combo in a video game). Just add list of previous few actions to state.
- Can be used to model two-player games. Model adversary as part of the transition function.

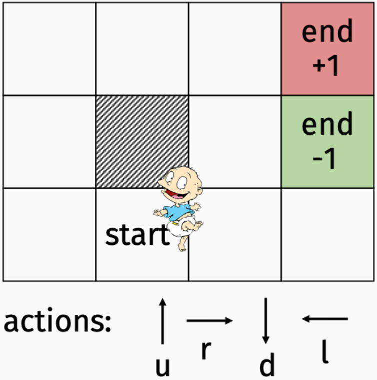




- $r_t = -.01$ if not at an end position. ± 1 if at end position.
- $P(s_t, a)$: 70% of the time move in the direction indicated by a. 30% of the time move in a random direction.

What is the optimal policy Π?

SIMPLE EXAMPLE: GRIDWORLD



- $r_t = -.5$ if not at an end position. ± 1 if at end position.
- $P(s_t, a)$: 70% of the time move in the direction indicated by a. 30% of the time move in a random direction.

What is the optimal policy Π?

For infinite or very long times horizon games (large T), we often introduce a **discount factor** γ and seek instead to take actions which minimize:

$$\mathbb{E} \sum_{t=0}^{T} \gamma^t r_t$$

where $r_t \sim R(s_t, \Pi(s_t))$ and $s_{t+1} \sim P(s_t, \Pi(s_t))$ as before.

$\gamma \to 1$: No discount. Standard MDP expected reward.

$\gamma \to 0$: Care about short term reward more.

From now on assume $T = \infty$. We can do this without loss of generality by adding a time parameter to state and moving into an "end state" with no additional rewards once the time hits T.

Value function: Measures the expected return if we start in state s and follow policy Π.

$$V^{\Pi}(s) = \mathbb{E}_{\Pi, s_0 = s} \sum_{t \geq 0} \gamma^t r_t$$

Let $\Pi_s^* = \arg\max V^{\Pi}(s)$. If we are in state s, at any point, we should always take action $\Pi_s^*(s)$.

Value function:

$$V^{\Pi}(s) = \mathbb{E}_{\Pi, s_0 = s} \sum_{t \geq 0} \gamma^t r_t$$

Claim: Let $\Pi_s^* = \arg\max V^{\Pi}(s)$. If we are in state s, at any point, we should always take action $\Pi_s^*(s)$.

Proof: Suppose we has already taken $j-1$ steps and seen trajectory $[s_0, a_0, r_0], \ldots, [s_j, a_j, r_j]$. Then our expected reward is:

$$\begin{aligned} & r_0 + \gamma r_1 + \ldots + \gamma^{j-1} r_{j-1} + \mathbb{E}_{\Pi} \sum_{t \geq j} \gamma^t r_j \\ &= r_0 + \gamma r_1 + \ldots + \gamma^{j-1} r_{j-1} + \gamma^j \mathbb{E}_{\Pi} \sum_{t \geq 0} \gamma^t r_{t+j} \\ &= r_0 + \gamma r_1 + \ldots + \gamma^j r_j + \gamma^j V^{\Pi}(s_j) \end{aligned}$$

Value function:

$$V^{\Pi}(s) = \mathbb{E}_{\Pi, s_0 = s} \sum_{t \geq 0} \gamma^t r_t$$

Claim: Let $\Pi_s^* = \arg\max V^{\Pi}(s)$. If we are in state s, at any point, we should always take action $\Pi_s^*(s)$.

So, there is a single optimal policy Π^* which simultaneously maximizes $V^{\Pi}(s)$ for all s. I.e. $\Pi_1^* = \Pi_2^* = \ldots = \Pi_{|S|}^* = \Pi^*$. We do not need to change the policy over time to maximize expected reward.

Goal in RL is to find this optimal policy Π^*.

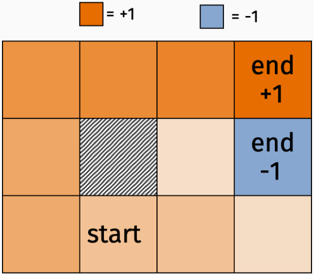
Full information: We know $\mathcal{S}$, $\mathcal{A}$, the transition function P and reward function R. The optimal policy can Π^* can be found via dynamic programming. Sometimes called "planning" problem.

Reinforcement Learning setting: We do not know P or R, but we can repeatedly play the MDP, running whatever policy we like.

Let $V^*(s) = V^{\Pi^*}(s)$. In the full information setting, if we knew V^* we can easily find the optimal policy Π:

$$\Pi^*(s) = \arg\max_a \sum_{s',r} \Pr(s', r \mid s, a) V^*(s')$$

Let $V^*(s) = V^{\Pi^*}(s)$. In the full information setting, if we knew V^* we can easily find the optimal policy Π:



$$\Pi^*(s) = \arg\max_a \sum_{s',r} \cdot \Pr(s', r \mid s, a)[r + \gamma V^*(s')]$$

$V^*(s)$ satisfies what is called a Bellman equation:

$$V^*(s) = \max_a \sum_{s',r} \cdot \Pr(s', r \mid s, a)[r + \gamma V^*(s')]$$

Run a fixed point iteration to find V^*:

- Start with initial guess V^0.
- For $i = 1, \ldots, z$:
  - For $s \in \mathcal{S}$:
    - $V^i(s) = \max_a \sum_{s',r} \cdot \Pr(s', r \mid s, a)[r + \gamma V^{i-1}(s')]$

Can be shown to converge in roughly $z = \frac{1}{1-\gamma}$ iterations. What is the computational cost of each iteration?

Full information: We know $\mathcal{S}$, $\mathcal{A}$, the transition function P and reward function R.

Reinforcement Learning setting: We do not know P or R, but we can repeatedly play the MDP, running whatever policy we like.

- **Model-based** RL methods essentially try to learn P and R very accurately and then find Π^* via dynamic programming. Require a lot of samples of the MDP.

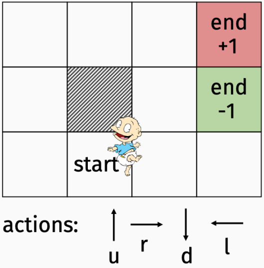
How many parameters do we need to learn if we hope to learn P and R?

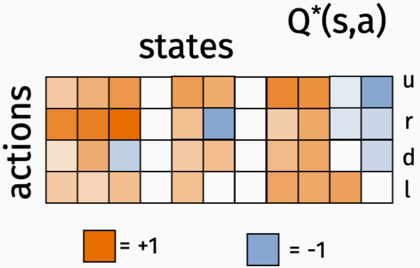
- **Model-free** RL methods try to learn Π^* without necessarily obtaining an accurate model of the world – i.e. without learning P and R.

Another important function:

- Q**-function:** $Q^{\Pi}(s,a) = \mathbb{E}_{\Pi, s_0 = s, a_0 = a} \sum_{t \geq 0} \gamma^t r_t$. Measures the expected return if we start in state s, play action a, and then follow policy Π.

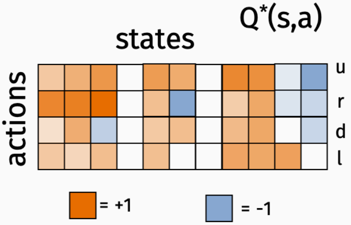
$$Q^*(s,a) = \max_{\Pi} Q^{\Pi}(s,a) = Q^{\Pi^*}(s,a).$$





$$Q^*(s,a) = \max_{\Pi} \mathbb{E}_{\Pi, s_0=s, a_0=a} \sum_{t\geq 0} \gamma^t r_t.$$

If we knew the function Q^*, we would immediately know an optimal policy. Whenever we're in state s, we should always play action $a^* = \arg\max_a Q^*(s,a)$.



Q has more parameters than V, but you can use it to determine an optimal policy without knowing transition probabilities.

Q^* also satisfies a Bellman equation:

$$Q^*(s,a) = \mathbb{E}[R(s,a)] + \gamma \mathbb{E}_{s' \sim P(s,a)} \max_{a'} Q^*(s',a').$$

Bellman equation:

$$Q^*(s, a) = \mathbb{E}[R(s, a)] + \gamma \mathbb{E}_{s' \sim P(s,a)} \max_{a'} Q^*(s', a').$$

Again use <u>fixed point</u> iteration to find Q^*. Let Q^{i-1} be our current guess for Q^* and suppose we are at some state s, a.

$$Q^i(s, a) = \mathbb{E}[R(s, a)] + \gamma \mathbb{E}_{s' \sim P(s,a)} \max_{a'} Q^{i-1}(s', a')$$

In reality, drop expectations and use a learning rate α

$$Q^i(s, a) = (1 - \alpha)Q^i(s, a) + \alpha \left(R(s, a) + \gamma \max_{a'} Q^{i-1}(s', a') \right)$$

How do we choose states s and a to make the update for? In principal you can do anything you want! E.g. choose some policy Π and run:

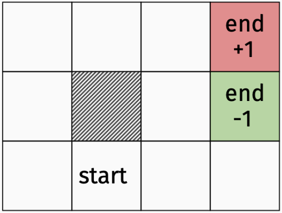
- Initialize Q^0 (e.g. all zeros)
- Start at s, play action $a = \Pi(s)$, observe reward $R(s, a)$.
- For $i = 1, \ldots, z$
  - $Q^i(s, a) = (1 - \alpha)Q^i(s, a) + \alpha \left(R(s, a) + \gamma \max_{a'} Q^{i-1}(s', a')\right)$
  - $s \leftarrow P(s, a)$
  - $a \leftarrow \Pi(s)$

  (restart if we reach a terminating state)

Q-learning is considered an **off-policy** RL method because it runs a policy Π that is not necessarily related to its current guess for an optimal policy, which in this case would be $\Pi(s) = \max_a Q^i(s, a)$ at time i.

Q-learning always converge to Q^* as long as we follow a policy Π that visits every start (s, a) with non-zero probability. Very mild condition, but exact choice of Π matters a lot for convergence speed.

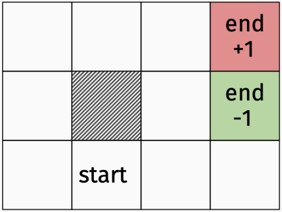
- **Random:** At state s, choose a random action a.
- **Greedy:** At state s, choose $\arg\max_a Q^i(s, a)$. I.e. the current guess for the best action.



Random can be wasteful. Spend time improving parts of Q that aren't relevant to optimal play. **Greedy** can cause you to zero in on a locally optimal policy without learning new strategies.

Possible choices for Π:

- **Random:** At state s, choose a random action a.
- **Greedy:** At state s, choose $\arg\max_a Q^i(s, a)$. I.e. the current guess for the best action.
- **ϵ-Greedy:** At state s, choose $\arg\max_a Q^i(s, a)$ with probability $1 - \epsilon$ and a random action with probability ϵ.



Exploration-exploitation tradeoff. Increasing ϵ = more **exploration**.

Another issue: Even writing down Q^* is intractable... This is a function over $|\mathcal{S}||\mathcal{A}|$ possible inputs. Even for relatively simple games, $|\mathcal{S}|$ is gigantic...

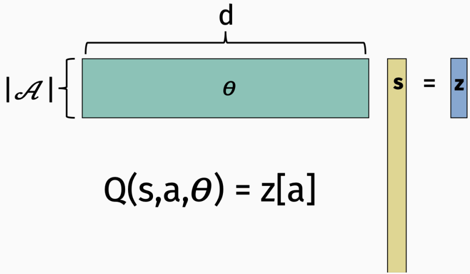
Back of the envelope calculations:

- **Tic-tac-toe:** $3^{(3\times 3)} \approx 20,000$
- **Chess:** $\approx 10^{43} < 28^{64}$ (due to Claude Shannon).
- **Go:** $3^{(19\times 19)} \approx 10^{171}$.
- **Atari:** $128^{(210\times 160)} \approx 10^{71,000}$.

Number of atoms in the universe: $\approx 10^{82}$.

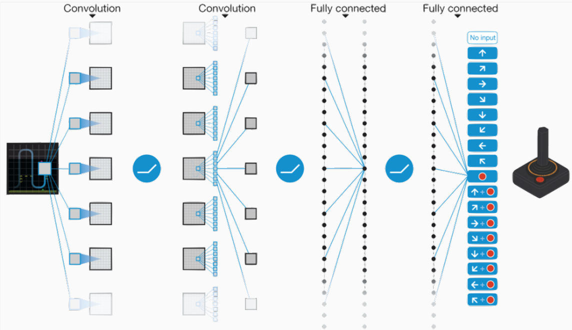
Learn a **simpler** function $Q(s, a, \theta) \approx Q^*(s, a)$ parameterized by a small number of parameters θ.

Example: Suppose our state can be represented by a vector in $\mathbb{R}^d$ and our action a by an integer in $1, \ldots, |\mathcal{A}|$. We could use a linear function where θ is a small matrix:



Learn a **simpler** function $Q(s, a, \theta) \approx Q^*(s, a)$ parameterized by a small number of parameters θ.

Example: Could also use a (deep) neural network.



DeepMind: "Human-level control through deep reinforcement learning", Nature 2015.

If $Q(s, a, \theta)$ is a good approximation to $Q^*(s, a)$ then we have an approximately optimal policy: $\tilde{\Pi}^*(s) = \arg\max_a Q(s, a, \theta)$.

- Start in state s_0.
- For $t = 1, 2, \ldots$
  - $a^* = \arg\max_a Q(s, a, \theta)$
  - $s_t \sim P(s_{t-1}, a^*)$

How do we find an optimal θ? If we knew $Q^*(s, a)$ could use supervised learning, but the true Q function is infeasible to compute.

Find θ which satisfies the Bellman equation:

$$Q^*(s,a) = \mathbb{E}_{s'\sim P(s,a)}\left[R(s,a) + \gamma \max_{a'} Q^*(s',a')\right]$$

$$Q(s,a,\theta) \approx \mathbb{E}_{s'\sim P(s,a)}\left[R(s,a) + \gamma \max_{a'} Q(s,a,\theta)\right].$$

Should be true for all a, s. Should also be true for $a, s \sim \mathcal{D}$ for any distribution $\mathcal{D}$:

$$\mathbb{E}_{s,a\sim\mathcal{D}} Q(s,a,\theta) \approx \mathbb{E}_{s,a\sim\mathcal{D}}\mathbb{E}_{s'\sim P(s,a)}\left[R(s,a) + \gamma \max_{a'} Q(s,a,\theta)\right].$$

Loss function:

$$L(\theta) = \mathbb{E}_{s,a\sim\mathcal{D}}\left(y - Q(s,a,\theta)\right)^2$$

where $y = \mathbb{E}_{s'\sim P(s,a)}\left[R(s,a) + \gamma \max_{a'} Q(s',a',\theta)\right]$.

Minimize loss with **gradient descent**:

$$\nabla L(\theta) = \mathbb{E}_{s,a\sim\mathcal{D}}\left[-2\nabla Q(s,a,\theta)\cdot[y - Q(s,a,\theta)]\right]$$

In practice use stochastic gradient:

$$\nabla L(\theta, s, a) = -2\cdot\nabla Q(s,a,\theta)\cdot\left[R(s,a) + \gamma\max_{a'} Q(s',a',\theta) - Q(s,a,\theta)\right]$$

- Initialize θ_0
- For $i = 0, 1, 2, \ldots$
    - Run policy Π to obtain s, a and $s' \sim P(s,a)$
    - Set $\theta_{i+1} = \theta_i - \eta\cdot\nabla L(\theta_i, s, a)$

η is a learning rate parameter.

Again, the choice of Π matters a lot. **Random play** can be wastefully, putting effort into approximating Q^* well in parts of the state-action space that don't actually matter for optimal play. ϵ-greedy approach is much more common:

- Initialize s_0.
- For $t = 0, 1, 2, \ldots,$
  - $a_i = \begin{cases} \arg\max_a Q(s_t, a, \theta_{curr}) & \text{with probabilty } (1-\epsilon) \\ \text{random action} & \text{with probabilty } \epsilon \end{cases}$

REFERENCES

Lots of other details we don't have time for! References:

- Original DeepMind Atari paper: `https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf`, which is very readable.
- Stanford lecture video: `https://www.youtube.com/watch?v=lvoHnicueoE` and slides: `http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture14.pdf`

Important concept we did not cover: **experience replay**.

https://www.youtube.com/watch?v=V1eYniJ0Rnk

THANKS!

- Don’t forget about the last problem set!
- I will release a study document for the exam and also schedule and extra office hours for next week.