# CS-UY 4563: Lecture 24 Reinforcement Learning

NYU Tandon School of Engineering, Prof. Christopher Musco

# Supervised learning:

- **Decision trees.** Very effective model for problems with few features. Difficult to train, but heuristics work well in practice.
- **Boosting.** Approach for combining several "weak" models to obtain better overall accuracy than any one model alone.

# Unsupervised learning:

- Adversarial models. Modern alternative to auto-encoders that performs very well for lots of interesting problems, especially in generative ML.
- **eClustering.** Hugely important for data exploration and visualization.

### DATA CLUSTERING

# Iemportant unsupervised learning task:



Separate unlabeled data into natural clusters.

- Exploratory data analysis.
- Categorizing and grouping data.
- Visualizing data.

# Example application:

# Images of Cats.



Find sub-classes in your data which you did not know about. Helps you decide how to adjust features or improve data set for a supervised application.

#### DATA CLUSTERING



## k-center clustering:

- Choose centers  $\vec{c}_1, \ldots, \vec{c}_k \in \mathbb{R}^d$ .
- Assign data point  $\vec{x}$  to cluster *i* if  $\vec{c_i}$  is the "nearest" center.
- Can use any distance metric.

Given data points  $\vec{x}_1, \ldots, \vec{x}_n$  and distance metric  $\Delta(\vec{x}, \vec{c}) \rightarrow \mathbb{R}$ , choose  $\vec{c}_1, \ldots, \vec{c}_k$  to minimize:

$$Cost(\vec{c}_1,\ldots,\vec{c}_k) = \sum_{i=1}^n \min_j \Delta(\vec{x}_i,\vec{c}_j).$$

In general this could be a hard optimization problem.

### K-MEANS CLUSTERING

**Common choice:** Use Euclidean distance. I.e. set  $\Delta(\vec{x}, \vec{c}) = \|\vec{x} - \vec{c}\|_2^2$ .

- If k = 1, optimal choice for  $c_1$  is the centroid  $\frac{1}{n} \sum_{i=1}^{n} \vec{x}_n$ . For large k the problem is NP-hard.
- Can be solved efficiently in practice using optimization techniques known as **alternating minimization**. Called "Llyod's algorithm" when applied to *k*-means clustering.
- Euclidean *k*-means can only identify <u>linearly separable clusters</u>.



Today: Give flavor of the area and insight into <u>one</u> algorithm (Q-learning) which has been successful in recent years. Basic setup:<sup>1</sup>

- Agent interacts with environment over time 1, ..., 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.

<sup>&</sup>lt;sup>1</sup>Slide content adapted from: http://cs231n.stanford.edu/

### REINFORCEMENT LEARNING EXAMPLES

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 |θ| < 90° (pendulum is upright). 0 when |θ| = 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

### REINFORCEMENT LEARNING EXAMPLES

#### 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 <u>sparse reward problem</u>. 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):

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

Why is this called a <u>Markov</u> decision process? What does the term Markov refer to? **Goal:** Learn a **policy**  $\Pi : S \to A$  from states to actions which maximized expected cumulative reward.

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

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

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

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

#### SIMPLE EXAMPLE: GRIDWORLD



- $r_t = -.01$  if not at an end position.  $\pm 1$  if at end position.
- $P(s_t, a)$  : 50% of the time move in the direction indicated by a. 50% of the time move in a random direction.

### What is the optimal policy $\Pi$ ?

#### SIMPLE EXAMPLE: GRIDWORLD



- $r_t = -.5$  if not at an end position.  $\pm 1$  if at end position.
- $P(s_t, a)$  : 50% of the time move in the direction indicated by a. 50% of the time move in a random direction.

### What is the optimal policy $\Pi$ ?

For infinite or very long times horizon games (large *T*), we often introduce a **discount factor**  $\gamma$  and seek instead to find a policy  $\Pi$  which minimizes:



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

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*.

Two important definitions.

- Value function:  $V^{\Pi}(s) = \mathbb{E}_{\Pi,s_0=s} \sum_{t \ge 0} \gamma^t r_t$ . Measures the expected return if we start in state s and follow policy  $\Pi$ .
- 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  $\Pi$ .

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

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

### **BELLMAN EQUATION**

Q\* satisfies what's known as a Bellman equation:

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

Value Iteration: Used fixed point iteration to find Q\*:

- Initialize Q<sup>0</sup> (e.g. randomly).
- For *i* = 1, . . . , *z*:
  - $Q^{i} = \mathbb{E}_{s' \sim P(s,a)} \left[ R(s,a) + \gamma \max_{a'} Q^{i-1}(s',a') \right]$

Possible to prove that  $Q^i \to Q^*$  as  $i \to \infty$ .

#### Note that many details are involved in this computation.

Need to handle the expectations on the right hand side by randomly sampling trajectories from the MDP.

**Bigger 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:  $3^{(3\times3)} \approx 20,000$
- Chess:  $\approx 10^{43}$  (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  $\theta$ .

**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  $\theta$  is a small matrix:



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

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, ...
  - $a^* = \arg \max_a Q(s, a, \theta)$
  - $s_t \sim P(s_{t-1}, a^*)$

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

# Q-LEARNING W/ FUNCTION APPROXIMATION

# Find $\theta$ 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  $a, s \sim D$  for any distribution 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  $y = \mathbb{E}_{s' \sim P(s,a)} [R(s,a) + \gamma \max_{a'} Q(s',a',\theta)].$ 

# Q-LEARNING W/ FUNCTION APPROXIMATION

Minimize loss with gradient descent:

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

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  $\theta_0$
- For i = 0, 1, 2, ...
  - Choose random  $s, a \sim \mathcal{D}$ .
  - Set  $\theta_{i+1} = \theta_i \eta \cdot \nabla L(\theta_i, s, a)$

where  $\eta$  is a learning rate parameter.

# Q-LEARNING W/ FUNCTION APPROXIMATION

- Initialize  $\theta_0$
- For *i* = 0, 1, 2, ...
  - Choose random  $s, a \sim \mathcal{D}$ .
  - Set  $\theta_{i+1} = \theta_i \nabla L(\theta_i, s, a)$ .

What is the distribution  $\mathcal{D}$ ?

• Random play: Choose uniformly over reachable states + actions.

**Wasteful:** Seeks to approximate *Q*<sup>\*</sup> well in parts of the state-action space that don't actually matter for optimal play. Would require a ton of samples.

More common approach: Play according to current guess for optimal policy, with some random "off-policy" exploration. The  $\mathcal{D}$  is the distribution over states/actions results form this play. Note that  $\mathcal{D}$  changes over time...

# $\epsilon$ -greedy approach:

• Initialize s<sub>0</sub>.

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

Exploration-exploitation tradeoff. Increasing  $\epsilon$  = more **exploration**.

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