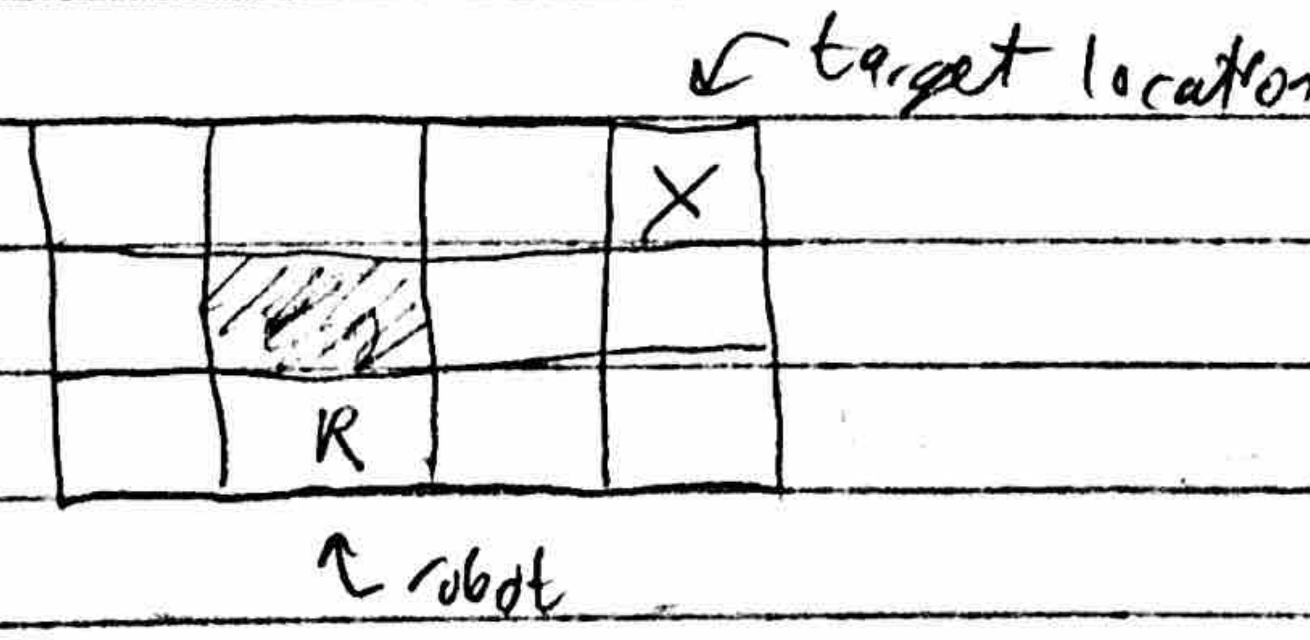
Lecture 11 – Reinforcement Learning

Today
- Markov Decision Processes (MDPs)
- Q-value Iteration
- Reinforcement Learning
- Exploration vs. Exploitation
- Deep Q-Networks (DQN)
- Final Review Competition!

Markov Decision Processes (MDPs)

Problem setup: We have a robot in a grid and we want it to go to a specific location.

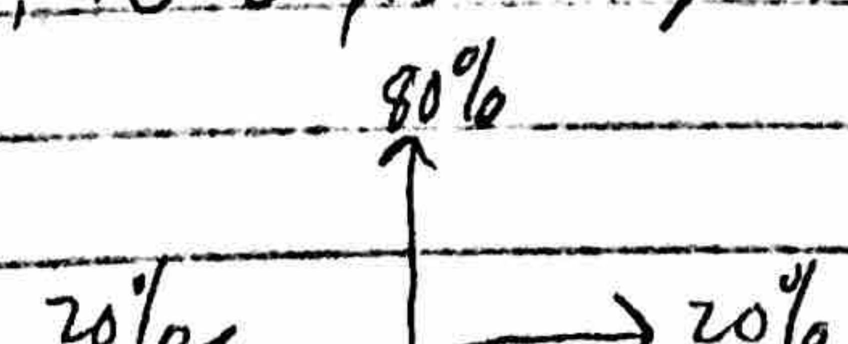


The robot's current location is its state.
The robot changes locations (states) by taking an action.

$$s \xrightarrow[\;]{\text{action } a} s'$$

current state → new state

However, the robot is not guaranteed to go where it's trying to go.
Ex, 80% probability it goes where it expects, 20% chance it doesn't.



We say that these probabilities are the transition probabilities.

$P(s'|s,a)$ — $T(s,a,s')$ = probability the robot ends up in state s' after taking action a in state s

11-2

<u>Utility function</u>: Sum of rewards across all states

$$U(s_0, s_1, s_2, \ldots) = R(s_0) + R(s_1) + R(s_2) + \ldots$$

We also need to measure how well the robot is performing the task of navigating through the grid.
The robot's performance will be measured with a <u>reward function</u>.
The reward function can either be a function just of the current state $R(s)$, or it can be a function of the current state, action, and next state $R(s,a,s')$.

Ex. $R(s) = \begin{cases} 1 & \text{if } s = \text{target state (location)} \\ 0 & \text{otherwise} \end{cases}$

? What will robot do with above reward function?

<u>Problem</u>: If we use the above reward function, then there's no penalty for taking infinitely long to reach the target. Our utility function is 1 no matter how long we take.

<u>Idea</u>: Prioritize reward sooner rather than later.

? How? Use <u>discounted future reward</u> as our utility function.

$$U(s_0, s_1, s_2, \ldots) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots, \quad 0 \le \gamma < 1$$

Since $0 \le \gamma < 1$, the total utility is larger when rewards are accumulated sooner.
The robot will maximize its utility by getting to the target as quickly as possible.

Finally, we need a method for the robot to choose an action in each state.

<u>Policy</u>: A function π which specifies an action for each state.

Now we have all the components to define a markov decision process.

MDP:
- a set of states S
- a set of actions A
- a transition probability function $T(s,a,s') = p(s' \mid s,a)$
- a reward function $R(s,a,s')$ or $R(s)$

Goal: Find the optimal policy π^* which maximizes the discounted future reward for the MDP

How?

Q-Value Iteration

Idea: Iteratively estimate the expected utility of performing each action in each state.

Notation:

- $V^*(s)$ = Value (expected utility) of starting in state s and acting optimally thereafter
- $Q^*(s,a)$ = expected utility of starting in state s, taking action a, and acting optimally thereafter
- $\pi^*(s)$ = the optimal action we should take in state s to maximize the expected utility

Relations:

- $V^*(s) = \max_a Q^*(s,a) = Q^*(s, \pi^*(s))$
- $Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$

$$= \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q(s',a') \right]$$

- $\pi^*(s) = \underset{a}{\operatorname{argmax}}\ Q^*(s,a)$

If we can learn the optimal Q-values Q^*, then we can compute the optimal policy π^*.

We will estimate Q^* with the Q-value iteration algorithm.

Since each Q-value just depends on the Q-value of the next state, we'll recursively update the Q-values to reflect the reward from farther and farther states.

11-4

Q-Value Iteration Algorithm
- Start with $Q_0(s,a)=0$ for all $s\in S$, $a\in A$
- For $i=1,2,\ldots$ until convergence:

$$Q_{i+1}(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

Problem: The above relies on us knowing the transition probabilities T and the reward function R. In reality, our robot doesn't know these apriori. The robot can only move around in the grid and observe states and collect rewards.

How can we still learn an optimal policy without knowing T or R.

Reinforcement Learning
We will implicitly discover T and R by sampling transitions s, a, s' and collecting rewards $R(s,a,s')$. (We don't know the function R but we can still collect rewards from individual samples.)

We can then use these samples to update our estimates of $Q(s,a)$. We don't want to totally replace our old $Q(s,a)$ based on a single sample, so we instead compute an exponential moving average.

Q-Learning
$$Q(s,a) \leftarrow (1-\alpha)\, Q(s,a) + \alpha\left[R(s,a,s') + \gamma \max_{a'} Q(s',a')\right]$$
$$= Q(s,a) + \alpha\left[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q^*(s,a)\right]$$

Old values of $Q(s,a)$ will slowly fade away with time since they are multiplied by $(1-\alpha)$ on every step.

Exploration vs. Exploitation
How do we select the actions a for the samples s, a, s' in the above Q-Learning algorithm?

We could follow our estimate of the optimal policy $\pi(s) = \arg\max_a Q(s,a)$. But early on our estimates of $Q(s,a)$ will be bad and we may choose bad actions

Alternatively, we could randomly sample actions. This guarantees that we explore a variety of actions for different states, but then we lose the opportunity to exploit what we've learned about good actions in $Q(s,a)$.

This is the exploration vs. exploitation tradeoff.

Idea: At first explore while $Q(s,a)$ is bad. Then shift to exploitation as $Q(s,a)$ improves.

More formally, let ε be the probability of randomly selecting an action and $1-\varepsilon$ be the probability of following the current estimate of the policy $\pi(s) = \arg\max_a Q(s,a)$

We'll start with $\varepsilon = 1$ and we'll decay ε as we perform updates of $Q(s,a)$.

$$\varepsilon \text{ large} \rightarrow \varepsilon \text{ small}$$
$$\text{exploration} \rightarrow \text{exploitation}$$

Deep Q-Networks (DQN)

Problem: What if the number of states is very large?

Ex. Playing an Atari game with 84×84 pixel grayscale image with 256 gray levels, and our state is the last 4 frames.

$\#\text{states} = 256^{84 \times 84 \times 4} \approx 10^{67,970} > \#$ atoms in universe

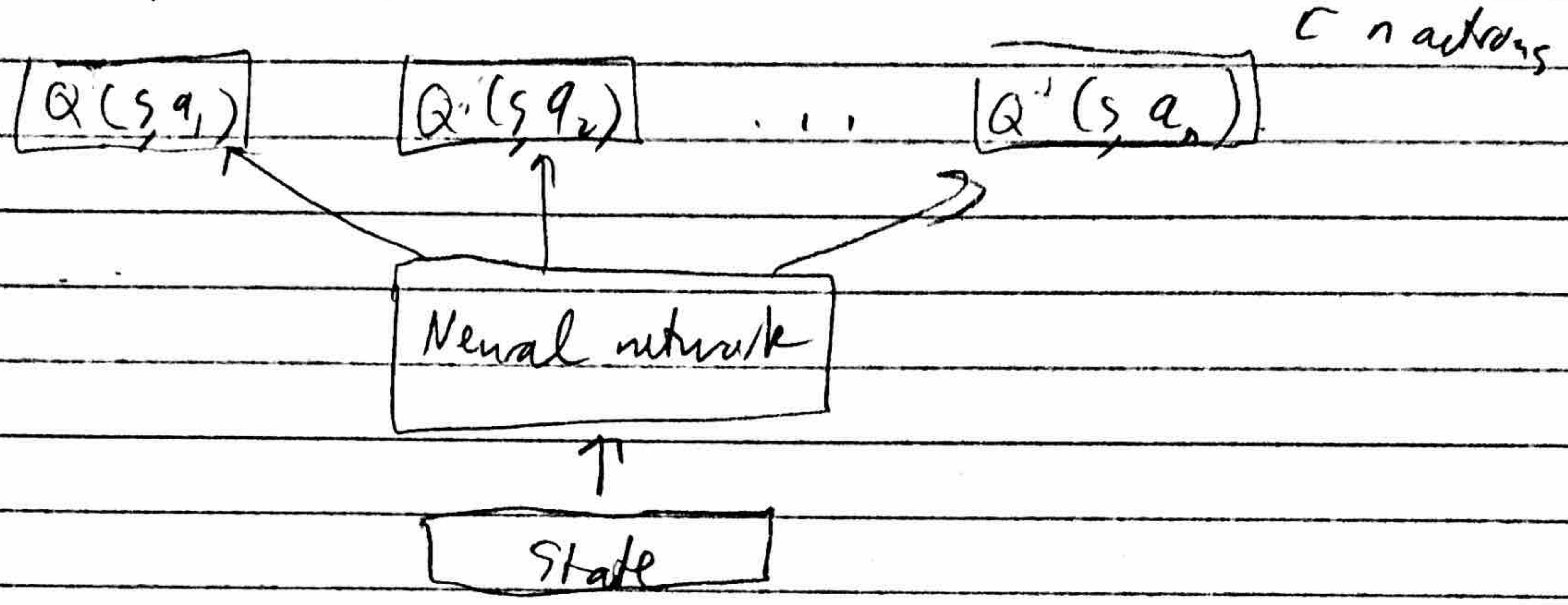
Clearly we cannot compute the Q-value for every state and action.

How can we predict the Q-values for states we haven't seen (but which might have features similar to states we have seen)?

Neural networks!

11-6

We train a neural network to predict the Q-value for each possible action given the state.



Prediction = $Q(s,a)$

Target = $R(s,a,s') + \max_{a'} Q(s',a')$

$L = \frac{1}{2}(\text{target} - \text{prediction})^2$ (mean squared error)

<u>DQN</u> (simplified)

Alternately:

- Use the network to predict Q-values, use the Q-values to determine the policy π, and play the game according to π (with some ε exploration). Save the experiences s,a,s' and rewards $R(s,a,s')$.
- Use the experiences to train the network to produce better Q-values.

<u>Final</u> Review Competition – see slides