# Reinforcement Learning

## Part II

Some slides were adapted/taken from various sources, including Prof. Andrew Ng's Coursera Lectures, Stanford University, Prof. Kilian Q. Weinberger's lectures on Machine Learning, Cornell University, Prof. Sudeshna Sarkar's Lecture on Machine Learning, IIT Kharagpur, Prof. Bing Liu's lecture, University of Illinois at Chicago (UIC), CS231n: Convolutional Neural Networks for Visual Recognition lectures, Stanford University, Dr. Luis Serrano, Prof. Alexander Ihler and many more. We thankfully acknowledge them. Students are requested to use this material for their study only and **NOT** to distribute it.

# Value Functions

Now with the MDP in place, we have a description of the environment but still we don't know how the agent should act in this environment.

The rule we impose on the agent is that it must act in a way to maximize the collected rewards.

But to be able to do that, the agent should have something to estimate the position or state it is currently in.

Consider again the labyrinth, if the agent is few steps from the exit, his position has much more value than a position at the center of the labyrinth.

We call this is to estimate the Value of the position or the Value of the state.

Once we know the value of each state, we can figure out what is the best way to act (simply by following the state that with the highest value).

# State Value Function

Still we need to figure out a way to quantify the value of each state. This is done by a function called Sate-Value Function:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S}$$

What is important to learn about this equation is that the value of each state is the average of discounted future rewards.

If we look closely at the equation we see that ***v(s)*** is expressed in terms of immediate rewards ***r*** and the value of neighboring states ***v(s')***.

# State Value Function

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S}$$

1. When at a state ***s***, consider all possible actions, then,

2. for each action ***a*** get the probability ***p(s', r | s, a)*** of getting immediate reward ***r***, and moving to neighboring state ***s'***, knowing that we are at state ***s*** and we performed action ***a***.

3. Add the reward ***r*** to the discounted value of neighbor state ***s'*** given by ***γ*v(s')***. Multiply the result with ***p(s', r | s, a)***, and we get ***p(s', r | s, a)*[r+γ*v(s')]***.

4. Repeat steps 2 and 3 for all states **s'** neighboring ***s,*** as well as all the possible rewards ***r***, and compute the sum of the results. Now we have ***Sum( p(s', r | s, a)*[r+γ*v(s')])*** over all ***s'*** and ***r.***

5. This will give the average of discounted future rewards when taking only one action ***a***. However there are multiple actions to be taken so we have to average over all actions. To do so we multiply by probability ***π(a|s)*** that action ***a*** be performed at state ***s***, and we sum over all possible actions in that state.

# Action Value Function

•It suffices to think that when we are at a state we have a probability to take some action that might lead us to different states with different rewards.

•So the value of our state is the average of all discounted rewards that we might get when performing all of the possible actions at the current state.

•Now we have a function that gives us the value of each state. It tells us how good is to be at each one of them.

•Of course we still have to make the computation in order to get a number that represents the value of that state.

•***v(s)*** tells how good to be in state ***s***, but it does not tell how good to perform action ***a*** while in state ***s***. This is the purpose of **the Action-Value function**:

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right]$$

# Action Value Function

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big]$$

The explication is simple, we take ***v(s)*** and instead of performing all actions, we decide to perform only one action.

We no more ask ourselves what is the probability of using action ***a*** over all possible actions, but we say we will use action ***a***.

So the sum of ***π(a|s)*** does not apply anymore.

With this logic we reduce ***v(s)*** from averaging over all possible actions to simply using one selected action, we denote this as ***q(s,a)***.

# Action Value Function

Notice that ***q(s,a)*** checks the value of the action at state ***s*** while the values ***v(s')*** of neighboring states are kept unchanged. So in this sense it checks how good this action is in the current state while keeping everything else the same.

It is also easy to notice that ***v(s)*** is the weighted average of ***q(s,a)*** over all possible actions at state ***s***

$$v_\pi(s) = \sum_a \pi(a|s)\, q_\pi(s,a)$$

# Policy

The act of selecting an action at each state is called "policy" and is denoted as $\pi$.

- some policies are better than others due to the selection of actions over others in one or more states.
- It is important to note that a policy $\pi$ is better than $\pi'$ if all $v(s)$ under $\pi$ are greater or equal to all $v(s)$ under $\pi'$.
- It follows that in order to **maximize the collected rewards** we have to find the best possible policy, called optimal policy and denoted $\pi^*$.

# Optimal Value Functions and Policy

An optimal value state function ***v*(s)*** is a function that gives the maximum value at each state among all policies:

$$v_*(s) = \max_\pi v_\pi(s), \quad \text{for all } s \in \mathcal{S} \quad \text{where}$$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S}$$

Similarly an optimal action state function ***q*(s)*** is the function that gives the maximum ***q*** value at each state among all policies:

$$q_*(s, a) = \max_\pi q_\pi(s, a), \quad \text{for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}(s)$$

it follows that

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a) \quad \text{where } q_\pi(s, a) = \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big]$$

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma v_*(s')\big]$$

# Optimal Policy

Notice that ***v(s)*** is the average of values produced by all actions, while ***v*(s)*** is the maximum value that one action can produce among all other actions.

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right]$$

Since ***q(s,a)*** is the value produced by a specific action ***a*** at state ***s***, ***q*(s,a)*** is the value produced by a specific action ***a*** at state ***s***, while selecting maximum ***q*** values in the other states.

Earlier we said that a policy $\pi$ is better than another policy $\pi'$ if it produces higher or equal ***v(s)*** value at each state:

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

# Optimal Policy

It follows that the optimal policy $\pi^*$ produces, at each state, higher or equal ***v(s)*** values than any other policy $\pi$.

$$\pi_* \geq \pi \quad \text{if } v_{\pi_*}(s) \geq v_\pi(s), \forall s \; \forall \pi$$

This means that any ***v(s)*** and ***q(s,a)*** following the optimal policy $\pi^*$ are equal to ***v*(s)*** and ***q*(s,a)*** respectively.

*All optimal policies achieve the optimal value function,*
$v_{\pi_*}(s) = v_*(s)$
*All optimal policies achieve the optimal action-value function,*
$q_{\pi_*}(s, a) = q_*(s, a)$

# How to find the optimal policy $\pi^*$ ?

So the question that remains is, how to find the optimal policy $\pi^*$ ?
The optimal policy should always return the action that produces ***q*(s,a)***

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\text{argmax}}\ q_*(s,a) \\ 0 & \textit{otherwise} \end{cases}$$

- There is always a deterministic optimal policy for any MDP
- If we know ***q*(s,a)*** it will be very easy to find $\pi^*$.

# MDP Example: Grid World

actions = {

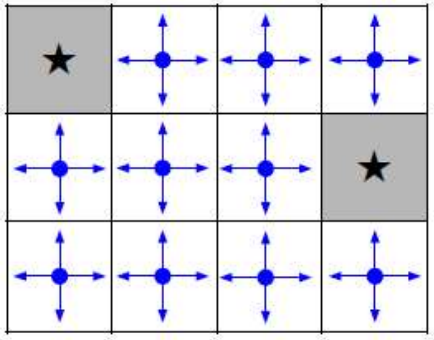
1. right
2. left
3. up
4. down

}

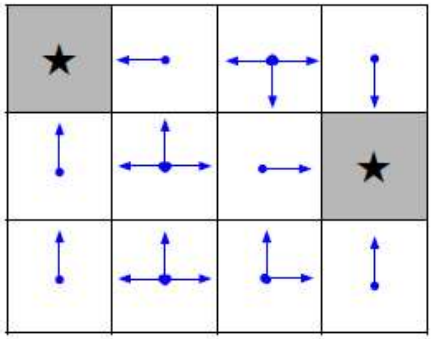

Set a negative "reward" for each transition (e.g. $r = -1$)

**Objective:** reach one of terminal states (greyed out) in least number of actions

# MDP Example: Grid World



Random Policy



Optimal Policy

# Optimal Policy

Note that the optimal policy is not unique, and this can be easily verified.

Suppose an agent in the Labyrinth has arrived to a state with 3 possible moves (or actions), go left, go right, or go forward. Suppose going forward leads the agent to a dead end, while going left or right will lead the agent to the exit using the same number of steps.

It is clear that there are two optimal policies (not only one) at this state, one that instructs the agent to go left and the other instructs it to go right.

# Markov Decision Process

- At time step t=0, environment samples initial state $s_0 \sim p(s_0)$
- Then, for t=0 until done:
  - Agent selects action $a_t$
  - Environment samples reward $r_t \sim R( \cdot \mid s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P( \cdot \mid s_t, a_t)$
  - Agent receives reward $r_t$ and next state $s_{t+1}$

- A policy $\pi$ is a function from S to A that specifies what action to take in each state
- **Objective**: find policy $\pi^*$ that maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$

# The optimal policy $\pi^*$

We want to find optimal policy $\pi^*$ that maximizes the sum of rewards.

How do we handle the randomness (initial state, transition probability…)?

Maximize the **expected sum of rewards!**

Formally: $\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi\right]$ with $s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$

As we see earlier

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\text{argmax}}\ q_*(s, a) \\ 0 & \textit{otherwise} \end{cases}$$

# Value function and Q-value function

Following a policy produces sample trajectories (or paths) $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$

How good is a state?

The **value function** at state s, is the expected cumulative reward from following the policy from state s:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

As we see earlier

$$v_{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r | s, a)\Big[r + \gamma v_{\pi}(s')\Big], \quad \text{for all } s \in \mathcal{S}$$

How good is a state-action pair?

The **Q-value function** at state s and action a, is the expected cumulative reward from taking action a in state s and then following the policy:

$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

As we see earlier

$$q_{\pi}(s, a) = \sum_{s',r} p(s', r | s, a)\Big[r + \gamma v_{\pi}(s')\Big]$$

# Bellman equation

The optimal Q-value function Q* is the maximum expected cumulative reward achievable from a given (state, action) pair:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

Q* satisfies the following **Bellman equation**:

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

**Intuition:** if the optimal state-action values for the next time-step Q*(s',a') are known, then the optimal strategy is to take the action that maximizes the expected value of $r + \gamma Q^*(s',a')$

As we see earlier

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right]$$

The optimal policy $\pi^*$ corresponds to taking the best action in any state as specified by Q*

# Solving for the optimal policy

**Value iteration** algorithm: Use Bellman equation as an iterative update

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right]$$

$Q_i$ will converge to Q* as i -> infinity

What's the problem with this?
Not scalable. Must compute Q(s,a) for every state-action pair. If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!

Solution: use a function approximator to estimate Q(s,a). E.g. a neural network!

# Solving for the optimal policy: Q-learning

Q-learning: Use a function approximator to estimate the action-value function

$$Q(s, a; \theta) \approx Q^*(s, a)$$

function parameters (weights)

If the function approximator is a deep neural network => **deep q-learning**!

# Solving for the optimal policy: Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

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

Forward Pass

Loss function: $L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[ (y_i - Q(s,a;\theta_i))^2 \right]$

where $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | s,a \right]$

Iteratively try to make the Q-value close to the target value ($y_i$) it should have, if Q-function corresponds to optimal Q* (and optimal policy $\pi^*$)

Backward Pass

Gradient update (with respect to Q-function parameters θ):

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)) \nabla_{\theta_i} Q(s,a;\theta_i) \right]$$

Best of Luck…