28 November, 2016

Neural Networks

Course 8: Reinforcement Learning

Overview

- What is reinforcement learning
- Basic Reinforcement Learning
- Q - Learning
- Reinforcement Learning using Neural Networks
- Questions

What is reinforcement learning?

What is reinforcement learning?

- Reinforcement learning is learning what to do--how to map situations to actions--so as to maximize a numerical reward signal
  - The learner is not told which actions to take
  - He receives rewards or penalties for each actions he takes
  - Through trial and error he must find a strategy that maximizes the reward
  - It is not specific to neural nets, but neural nets can be used in this kind of learning

- Differences to supervised learning:
  - No labeled data
  - Feedback may be delayed
  - The agent affects the environment

What is reinforcement learning?

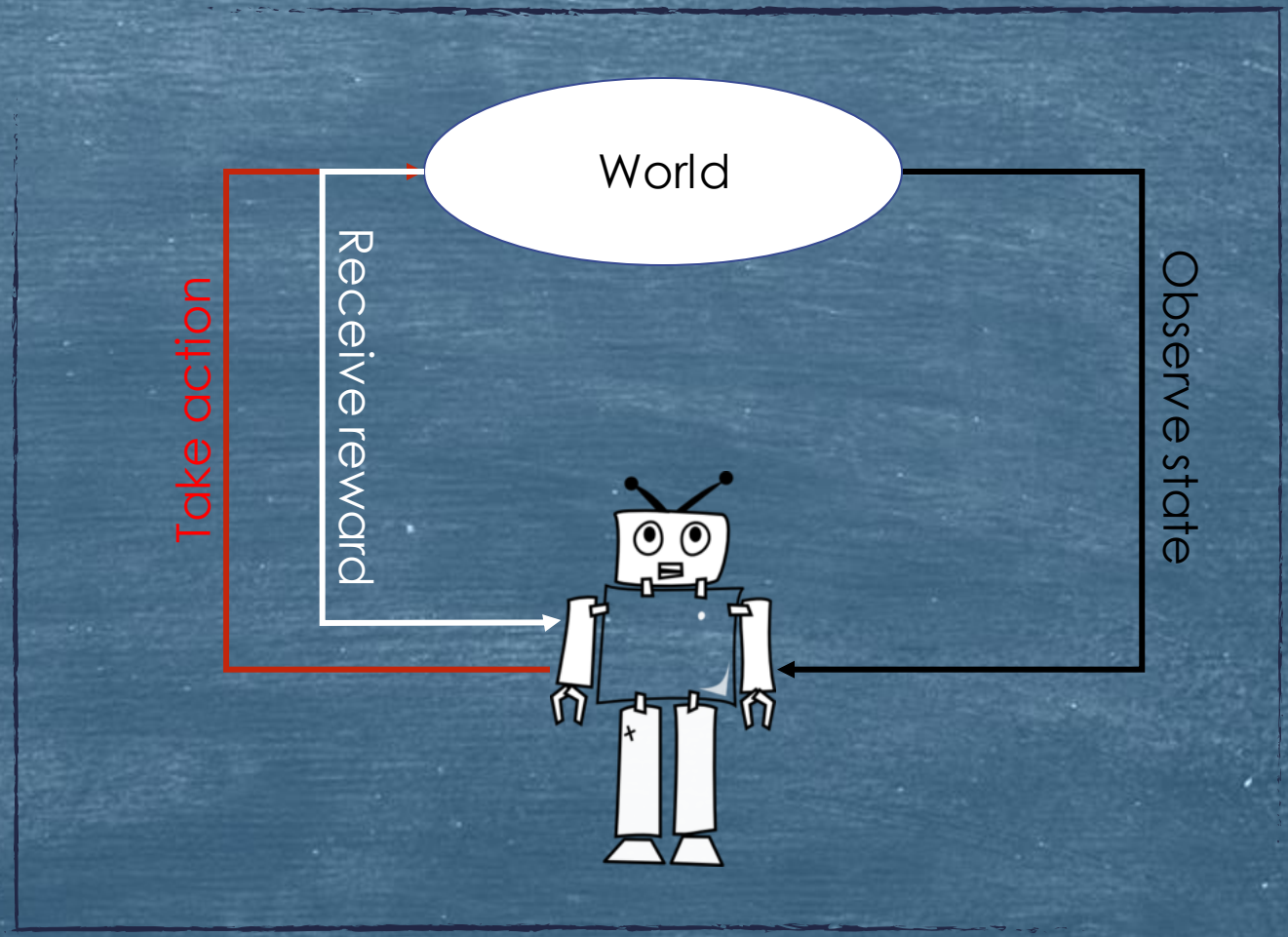
- Examples of reinforcement learning:
  - Make a robot walk
  - Play Atari Games
  - Beat world champion at Backgammon game
  - Invest in the stock market
  - Make an helicopter perform stunts

Basic Reinforcement Learning

The agent does the following steps:

- Observes the environment
- Takes an action
- Receive reward
- Learn
- Repeat

The purpose is to find a policy that maximizes the total reward over the lifetime of the agent



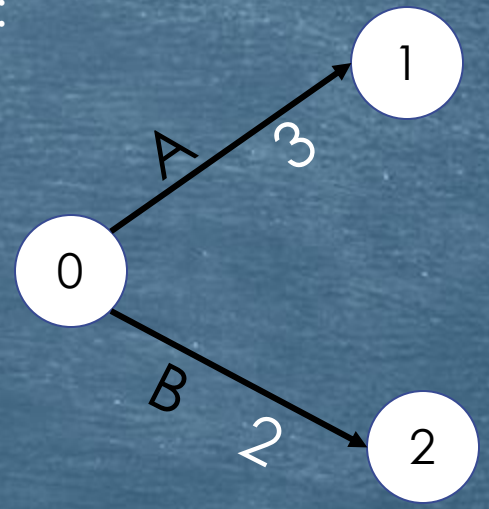
Basic Reinforcement Learning

- Markov Decision Process: Making a single decision:

Agent is in state 0

Action A, takes him to State 1, reward is 3

Action B, takes him to State 2, reward is 2



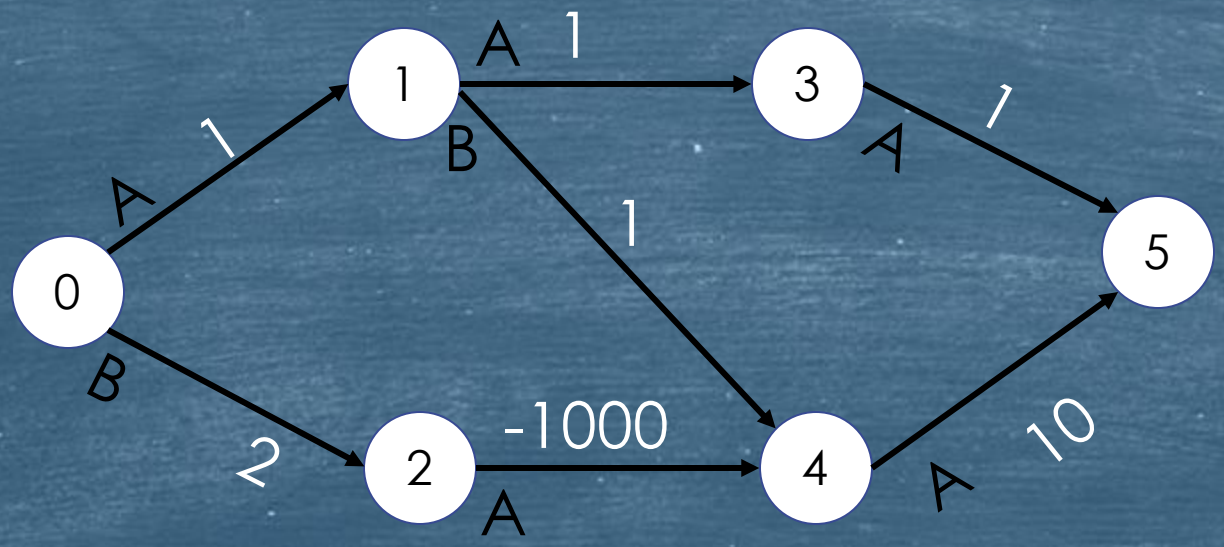
If Goal is to maximize reward, than the answer is simple:

Take action with the highest reward (B)

- State 0 has a value of 3:
  - Sum of rewards resulted from taking the best actions starting from state 0

Basic Reinforcement Learning

- Markov Decision Process: Making multiple decisions:
- This can be generalized. Every action affects subsequent actions.



Basic Reinforcement Learning

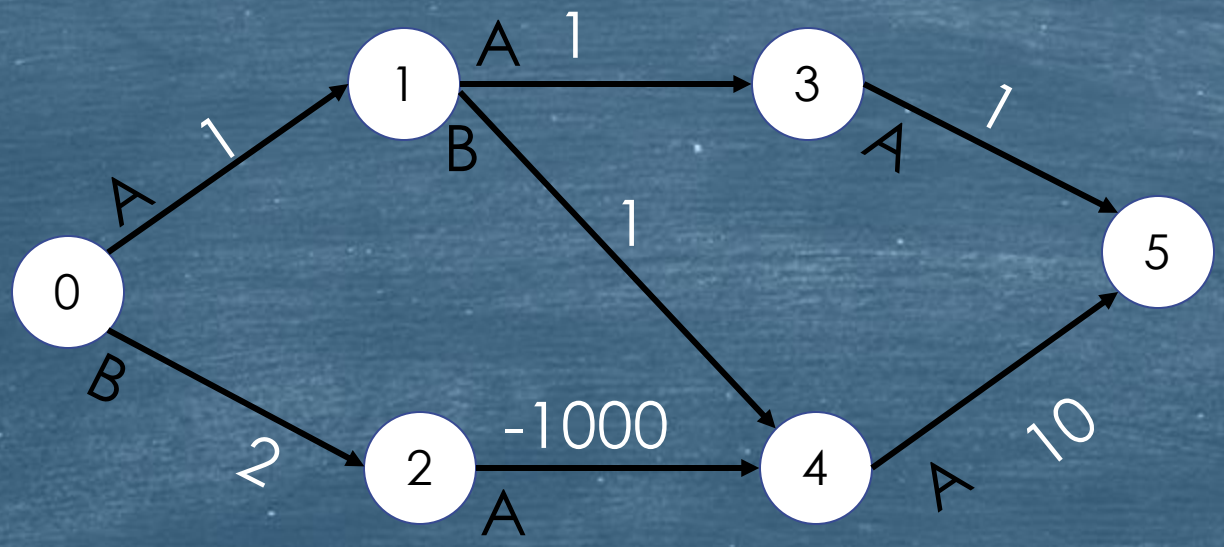
A Markov Decision Process is:

- A set of states, $S = \{s_1, s_2, \dots, s_n\}$
- A set of actions, $A = \{a_1, a_2, \dots a_m\}$
- A reward function, $R = S \times A \times S \to \mathbb{R}$
- A transition function: $P_{ij}^{a} = P(s_{t+1} = j | s_t = i, a_t = a)$

- We want to learn a policy (π) that will tell us what action to take in which state: $\pi: S \to A$ in order to maximize the sum of rewards over the lifetime of the agent

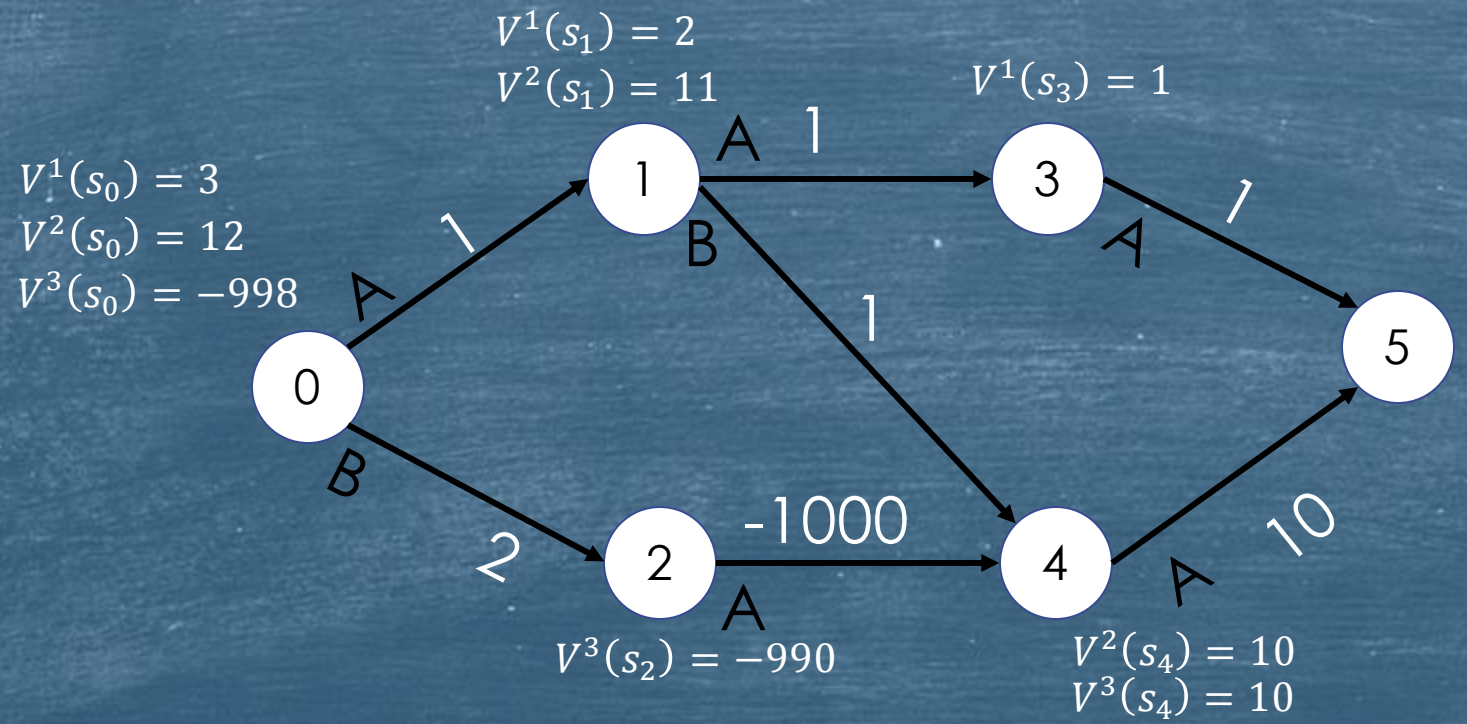
Basic Reinforcement Learning

- There are three policies:
  - $0 \rightarrow 1 \rightarrow 3 \rightarrow 5$. Reward=$1 + 1 + 1 = 3$
  - $0 \rightarrow 1 \rightarrow 4 \rightarrow 5$. Reward=$1 + 1 + 10 = 12$
  - $0 \rightarrow 2 \rightarrow 4 \rightarrow 5$. Reward=$2 - 1000 + 10 = -998$



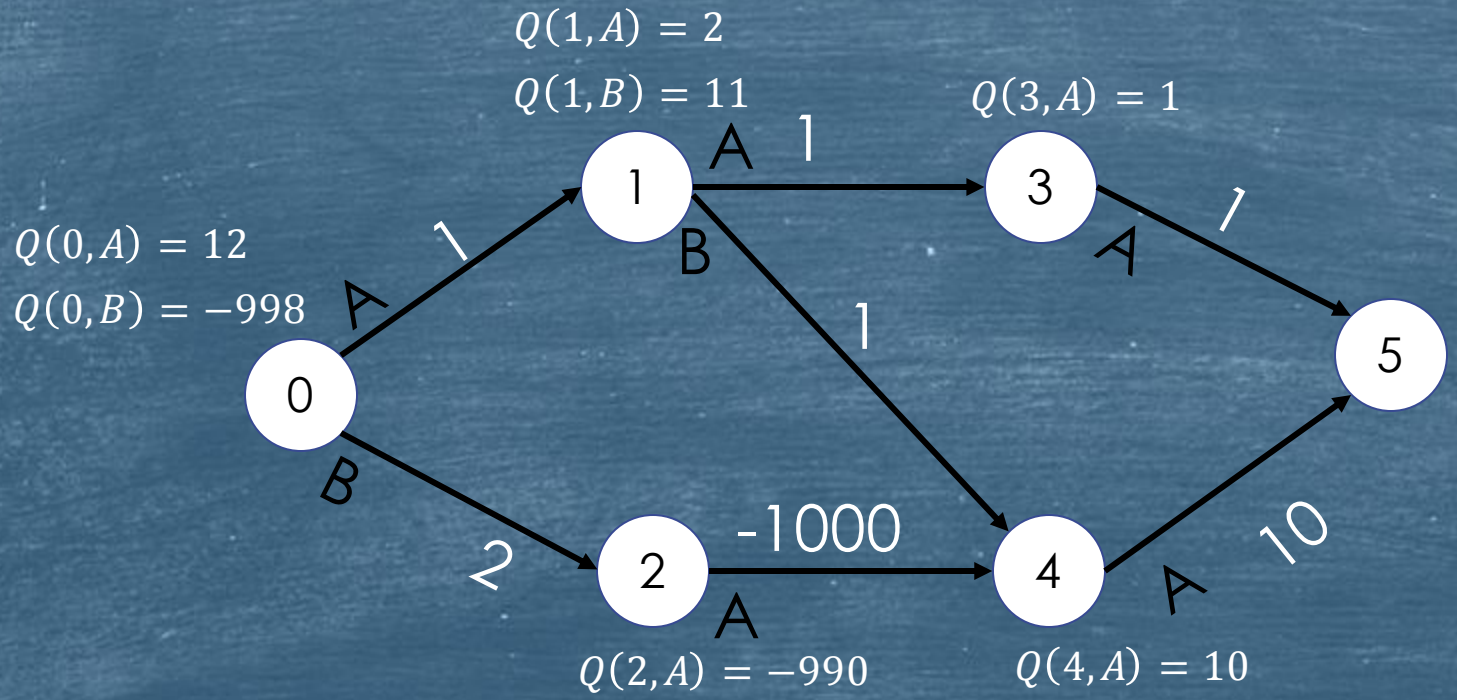
Basic Reinforcement Learning

- Value Function: Associates a value with each state for a policy (How good it is to run policy π from state s)



Basic Reinforcement Learning

- Value Function: We can define value without specifying the policy:
  - Specify the value of taking action a from state s and then performing optimally



Basic Reinforcement Learning

So, we have two ways of defining the value functions:

- $V^{\pi}(s) = R(s, \pi(s), s') + V^{\pi}(s')$
- $Q(s, a) = R(s, a, s') + \max_{a} Q(s', a')$

- Both tell us the same thing: Best reward + the best I can do for the next state

Basic Reinforcement Learning

If we have the value function, then it is easy to find the policy

- $\pi(s) = \arg\max_a(R(s,a,s') + V^{\pi}(s'))$
- $\pi(s) = \arg\max_a Q(s,a)$

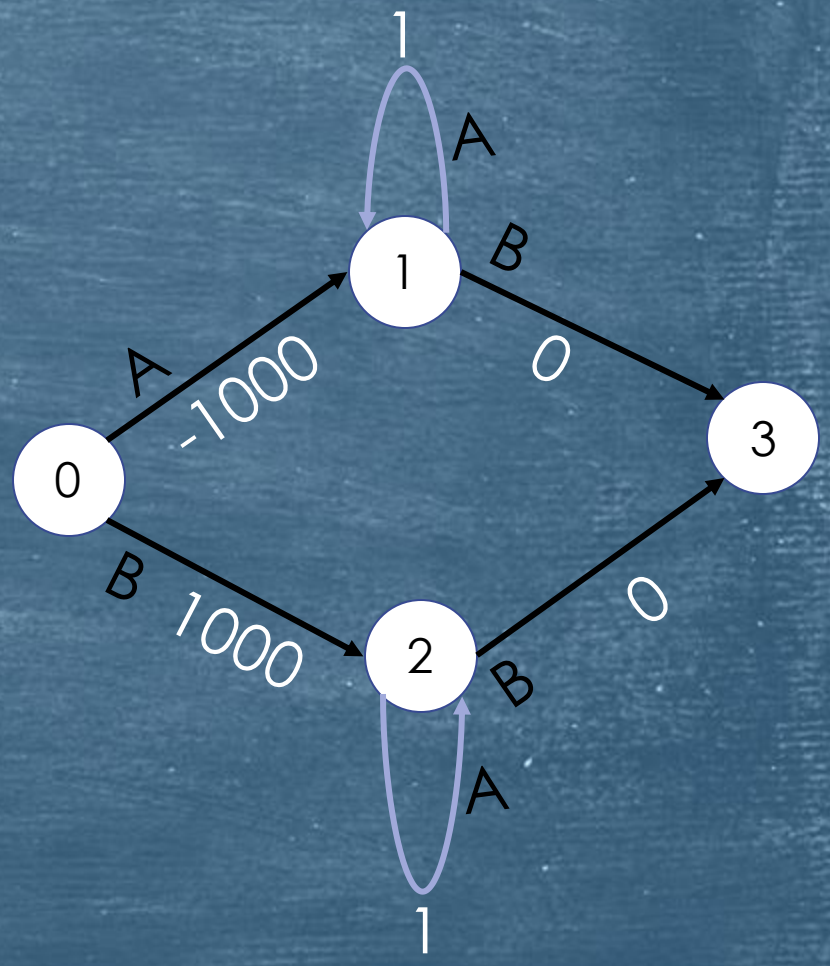
So, the easiest way to find the policy is to first find the value function

Basic Reinforcement Learning

More complex MDPs:

In this case, the number of steps is unlimited
The value of states 1 or 2 can be infinite

All policies with non-zero reward cycle have infinite value



Basic Reinforcement Learning

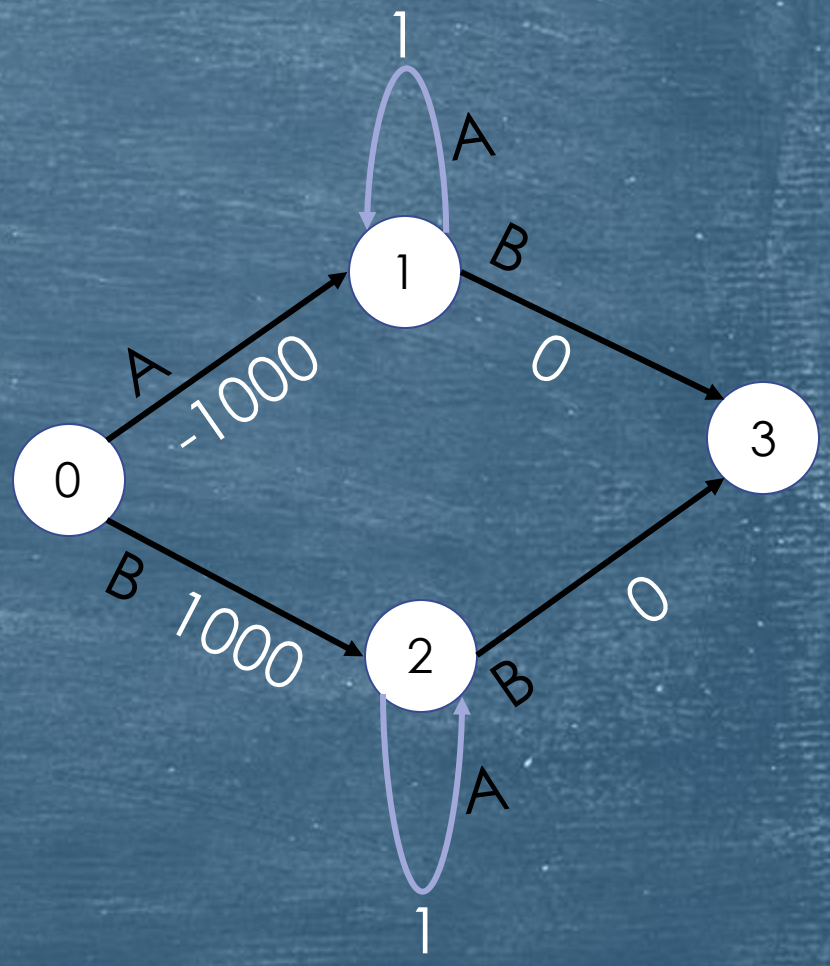
More complex MDPs:

We introduce a new term, $\gamma \in [0,1)$, called discount factor.

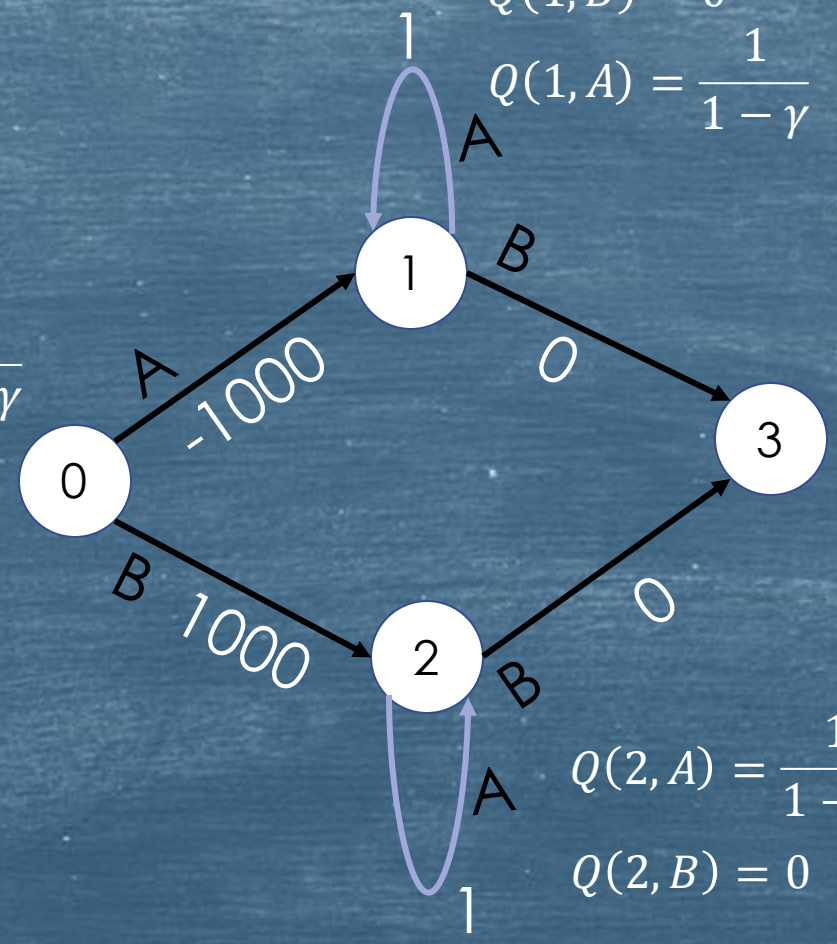
$$V^{\pi}(s) = R(s, \pi(s), s') + \gamma V^{\pi}(s')$$

$$Q(s,a) = R(s,a,s') + \gamma max_a \, Q(s',a')$$

- The purpose of γ is to avoid infinity, but it can also be considered as a term that measures how important is a reward in the future vs now



Basic Reinforcement Learning



$Q(0, A) = -1000 + \frac{\gamma}{1-\gamma}$

$Q(0, B) = 1000 + \frac{\gamma}{1-\gamma}$

$Q(1, B) = 0$

$Q(1, A) = \frac{1}{1-\gamma}$

$Q(2, A) = \frac{1}{1-\gamma}$

$Q(2, B) = 0$

$Q(1, A) = 1 + \gamma Q(1, A)$

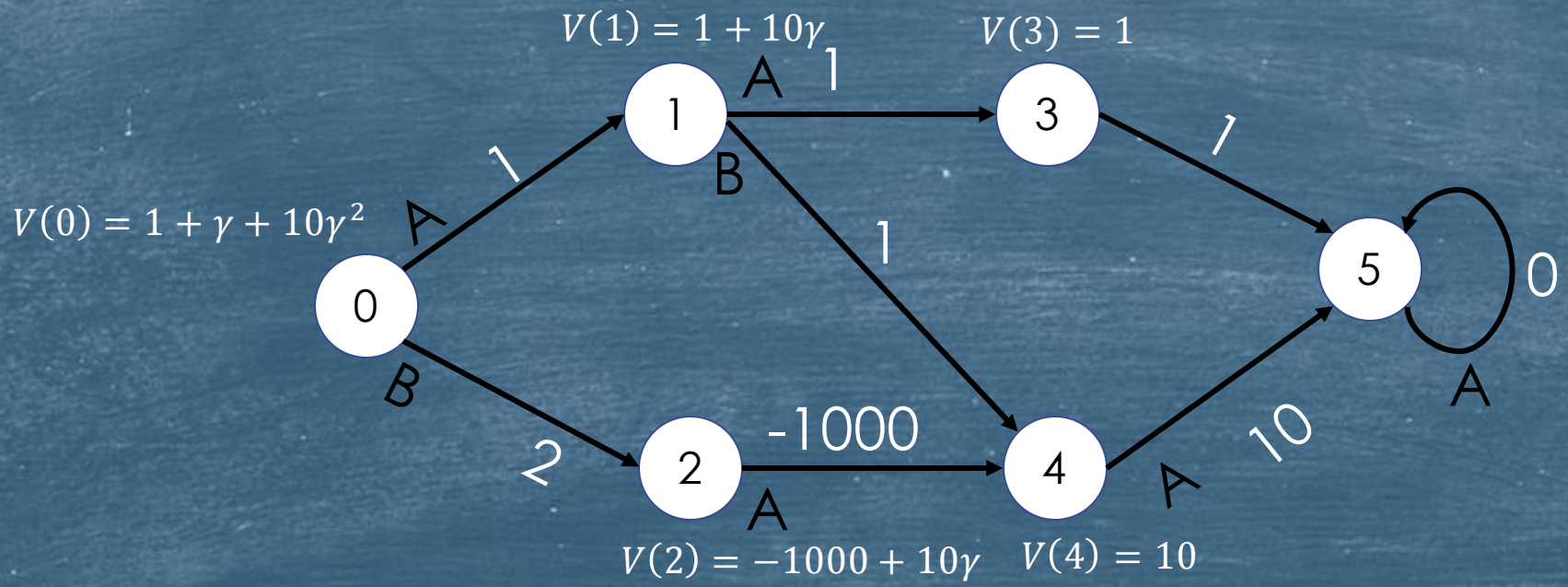
$\rightarrow Q(1, A) = \frac{1}{1-\gamma}$

Optimal policy:

$\pi(0) = B$
$\pi(1) = A$
$\pi(2) = A$

Basic Reinforcement Learning

- If the MDP model is complete, we can compute the optimal value function directly, through dynamic programming (recursive)



Basic Reinforcement Learning

- What happens when we don't have a complete MDP:
  - We know the states and actions
  - We don't have the system model (no transition function or reward function)
- We are allowed to sample from the MDP.

We must perform actions in order to generate experiences (s,a,r,s'). Based on a (large) number of experiences we can approximate the transition function and the reward function

This is called reinforcement learning

Basic Reinforcement Learning

- Monte Carlo Method:
  - The way the learning is performed is called policy iteration and contains two steps:

    1. Policy Evaluation: Start with a policy π and estimate the Q(s,a) values
    2. Policy Improvement: $\pi(s) = \arg\max_a Q(s,a)$, where Q values can be obtained by using: $Q(s,a) = R(s,a,s') + \gamma \max_a \hat{Q}(s',a')$

    Repeat until there is no change in policy

Basic Reinforcement Learning

- Monte Carlo Method:

Of course, the main problem is computing $Q(s,a) = R(s,a,s') + \gamma \max_a Q(s',a')$ since we don't have $\max_a Q(s',a')$

These can be estimated. More exact, the Monte Carlo Method is as following

1. Initialize Q(s,a) with some values (0)
2. Generate an episode starting from a state s_0 until finish according to a policy π
3. Store in a table the average values for each of the Q(s,a) from the episode.
   (as episodes increase the average will be a good approximation to the real value)
4. Repeat from 2.

Basic Reinforcement Learning

- Monte Carlo Method:

  Since the policy π is computed greedy with respect to the current value function, each change in Q(s,a) will produce a new policy, until the optimal policy is obtained

  However, all the states and all the actions need to be visited (many times) in order to have the correct Q(s,a) for every combination of state and action.

  A solution:

  1. Generate an episode from each state/action combination
  2. From time to time, don't follow the optimal policy

Basic Reinforcement Learning

- Example: Black Jack Game

  Rules of blackjack:
  - There is a dealer and a player
  - Each player is dealt 2 cards. Face-up. The dealer is delt two cards, one face-up, one face-down
  - The goal is to get the sum of your cards, as close to 21 as possible
  - Cards value(2-9 their number, J, Q, K -10 points)
  - An ace can be considered a 1 or a 10
  - You have two actions possible: stay or hit
  - The dealer always follows this policy: hit until cards sum 17 or more then stay
  - The player closer to 21 wins. The one over 21, loses

Basic Reinforcement Learning

What is the state:

- Sum of the player's cards
- If the players has an usable ace
- Sum of the dealer's cards
- If the dealer has an usable ace

What are the actions:

- Deal
- Stay

What are the rewards:

- Win: 1
- Lose: -1
- Draw: 0

Basic Reinforcement Learning

- Example: Black Jack Game

We will learn the best strategy (policy), by running the blackjack many times and averaging the results for each state/action combination

The states/actions (Q(s,a)) will be stored in a table (python dictionary)

Basic Reinforcement Learning

- Example: Black Jack Game

  Let's say that the game is in the following state:

  The sum of the player's card is 18

  The card that the dealer shows is 9

  This is a difficult position, since:

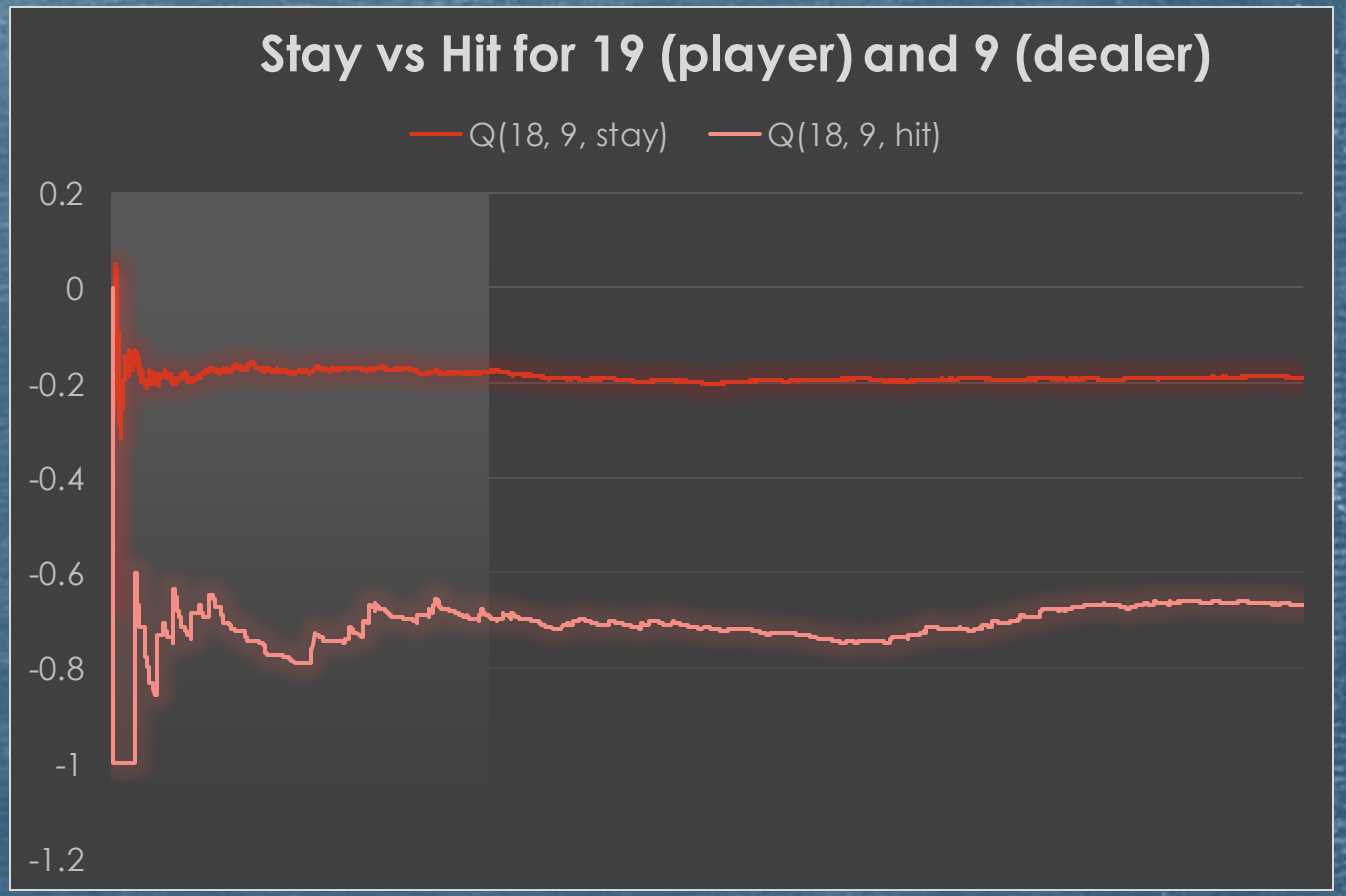
  There is a high probability that the other card of the dealer is 10 points

  But if the player chooses to hit, it is a high probability that he will go over 21

Basic Reinforcement Learning

Observe how the rewards stabilize over time

So, the best actions to take is to stay, since it is less likely to lose (-0.2) than if he would hit (-0.5)



Basic Reinforcement Learning

In the same way, all Q-values can be computed for every possible combination and a proper decision can be taken

	1		2		3		4		5		6		7		8		9		10	
	stay	hit	stay	hit	stay	hit	stay	hit	stay	hit	stay	hit	stay	hit	stay	hit	stay	hit	stay	hit
11	-0.76	-0.12	-0.29	0.21	-0.24	0.23	-0.21	0.26	-0.19	0.29	-0.15	0.31	-0.53	0.27	-0.5	0.2	-0.6	0.13	-0.58	0.04
12	-0.78	-0.54	-0.29	-0.3	-0.24	-0.25	-0.21	-0.22	-0.16	-0.22	-0.14	-0.19	-0.44	-0.24	-0.49	-0.29	-0.59	-0.36	-0.59	-0.43
13	-0.77	-0.57	-0.29	-0.32	-0.26	-0.32	-0.21	-0.31	-0.17	-0.25	-0.16	-0.28	-0.47	-0.29	-0.52	-0.34	-0.52	-0.41	-0.57	-0.48
14	-0.78	-0.59	-0.29	-0.33	-0.25	-0.35	-0.21	-0.35	-0.17	-0.31	-0.15	-0.31	-0.5	-0.35	-0.49	-0.39	-0.52	-0.45	-0.58	-0.51
15	-0.8	-0.63	-0.29	-0.41	-0.25	-0.42	-0.21	-0.38	-0.17	-0.42	-0.17	-0.39	-0.47	-0.4	-0.5	-0.44	-0.56	-0.48	-0.58	-0.55
16	-0.76	-0.65	-0.3	-0.5	-0.25	-0.46	-0.21	-0.45	-0.17	-0.43	-0.15	-0.43	-0.46	-0.43	-0.54	-0.47	-0.56	-0.52	-0.57	-0.59
17	-0.64	-0.67	-0.15	-0.55	-0.12	-0.58	-0.08	-0.51	-0.04	-0.53	0.01	-0.5	-0.1	-0.5	-0.39	-0.54	-0.43	-0.58	-0.47	-0.64
18	-0.38	-0.72	0.12	-0.63	0.15	-0.59	0.17	-0.62	0.2	-0.62	0.29	-0.63	0.39	-0.59	0.11	-0.63	-0.19	-0.62	-0.24	-0.67
19	-0.12	-0.79	0.39	-0.74	0.41	-0.75	0.42	-0.73	0.45	-0.73	0.5	-0.7	0.61	-0.72	0.59	-0.73	0.29	-0.7	-0.02	-0.75
20	0.15	-0.89	0.64	-0.85	0.64	-0.85	0.66	-0.85	0.67	-0.85	0.7	-0.84	0.77	-0.87	0.79	-0.86	0.75	-0.85	0.43	-0.86

Basic Reinforcement Learning

- Disadvantages for Monte Carlo:
  - The algorithm needs to reach the final reward in order to update all the rewards)
  - If the rewards have high variance, the convergence is slow
  - The number of states and actions must be small so they can fit in a table
  - The algorithm needs to evaluate a long trajectory just to update the starting state-action pair

Q-Learning

Q-Learning

- As with Monte-Carlo, Q-learning tries to predict the value for every Q(s,a) in order to be able to compute:

$$Q(s,a) = R(s,a,s') + \gamma \max_a Q(s',a')$$

- Let's suppose that we have all the predictions in a tables Q. We don't know how correct are those but it's the only ones we have.

  We'll note $Q^p(s,a)$ the prediction of $Q(s,a)$

- The idea of Q-learning is to first take an action, see the reward and correct the old prediction

Q-Learning

- As stated earlier, the γ factor tells us how important is the reward in the future vs now.

$$Q(s,a) = R(s,a,s') + \gamma \max_a Q(s',a')$$

Let's suppose that $\gamma = 0$. So, we are only interested in immediate rewards.

When we have to take an action; we look of all $\mathrm{Q^p}(s,a)$ values from our prediction table(we only have predictions) and take the corresponding action that takes as to a new state s' and gives us a reward $R(s,a,s')$

Q-Learning

Since we've got our immediate reward, we can compare it to our prediction and adjust the prediction.

$$Q^p(s,a) = Q^p(s,a) + \alpha(\, R(s,a,s') - Q^p(s,a))$$

Where $\alpha \in [0,1]$ is the learning rate

The learning rate tell us how quickly we want to incorporate the new date in our prediction:

- if $\alpha = 1$ then we want our prediction to be equal to the Reward
- if $\alpha \in (0,1)$ then we want more rewards in order to better estimate the prediction

Q-Learning

Let us incorporate the γ factor into the equation.

$$Q^p(s,a) = Q^p(s,a) + \alpha(R(s,a,s') + \gamma \max_a Q^p(s',a') - Q^p(s,a))$$

In the long run, $Q^p(s,a)$ will be equal to $R(s,a,s') + \gamma \max_a Q(s',a')$

So, we use a prediction of a later quantity ($Q^s(s',a')$) to update a current prediction.

This may seem strange since neither of these predictions are accurate.

But, in the course of the algorithm, later predictions tend to become accurate sooner then earlier ones. Se, there tends to be an overall error reduction as learning proceeds.

Reinforcement Learning using Neural Networks

Reinforcement Learning using Neural Networks

Until now we've been considering $Q(s, a)$ as a table.

However, this is not always practical.

For example, DeepMind used the pixels from the last 4 screen to represent a state.

Even though they have resized the screen to 84x84 and converted to grayscale (256 levels), the amount of states available is $256^{84x84x4}$

That is $\approx 10^{67970}$ rows in the Q table. More than the atoms in the universe

Reinforcement Learning using Neural Networks

This is where Neural Networks come in.

Neural Networks are good at approximating values.

So, we'll use neural networks to approximate Q(s,a).

So, like in the case of Atari games, there won't be much of a problem if a state is approximated (i.e. states that are different only by a bunch of pixels could have the same value)

But how do we train the network? What is our target value?

Reinforcement Learning using Neural Networks

In the case of Q-learning:

$$Q\ \ (s,a) = Q\ \ (s,a) + \alpha(R(s,a,s') + \gamma \max_{a} Q\ \ (s',a') - Q\ \ (s,a))$$

We've arrived at this equation by wanting to make $Q(s,a)$ be more like $R(s,a,s') + \gamma \max_{a'} Q(s',a')$

So, we'll train our neural network using this target value

Reinforcement Learning using Neural Networks

So, at each state s:

1. Predict all Q(s,a) for every action

2. Take action a where Q(s,a) is largest and arrive at state s' and receive reward $R(s,a,s')$

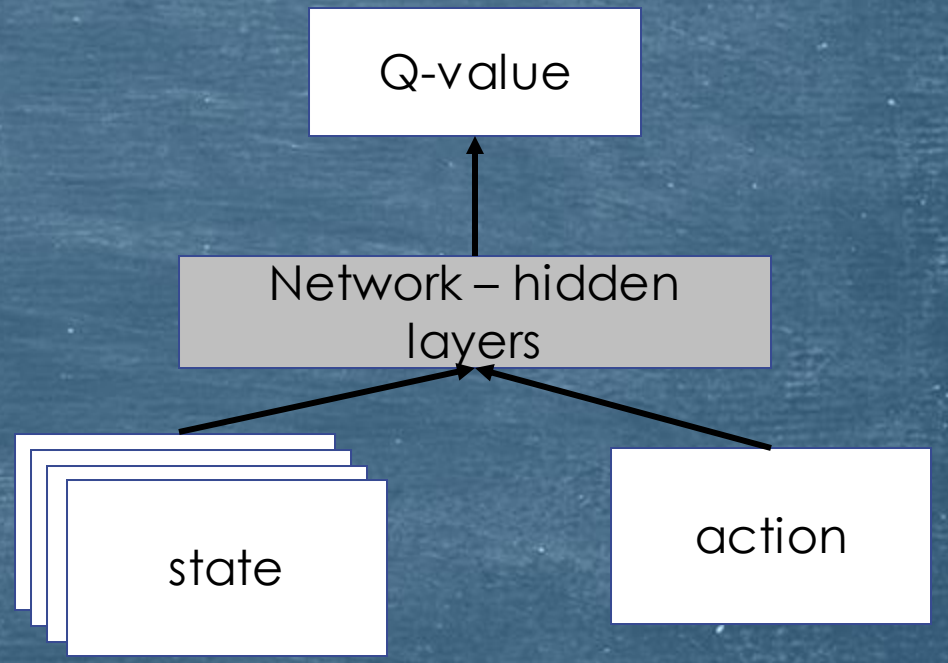
3. Compute Target Value:

   3'. Predict all Q(s',a') for every a' available in state s' using the neural network

   3''. Compute $\max_{a'} Q(s',a')$

   3''. Compute $y_{sa} = R(s,a,s') + \gamma \max_{a'} Q(s',a')$

4. Train the neural network using the previous state(s) and the previous action(a) as input and y_{sa} as output

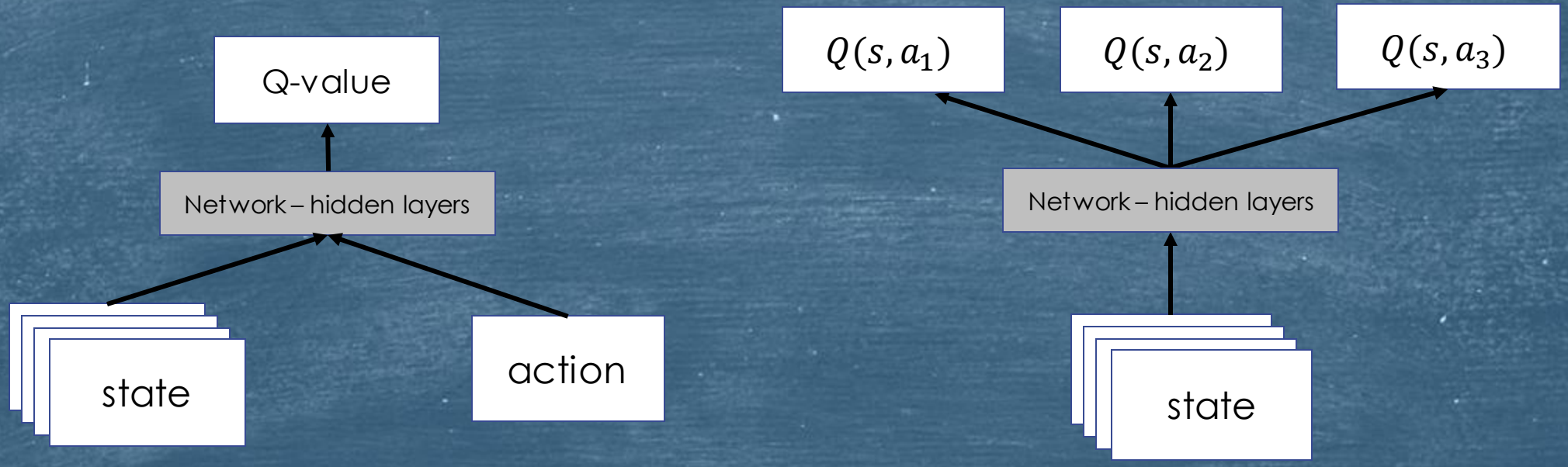


Reinforcement Learning using Neural Networks

We can improve our neural network learning, by modifying the output value

Instead of doing a forward pass for each state/action combination, we modify our network to have an output for each action

This way, we only need a forward pass for each action

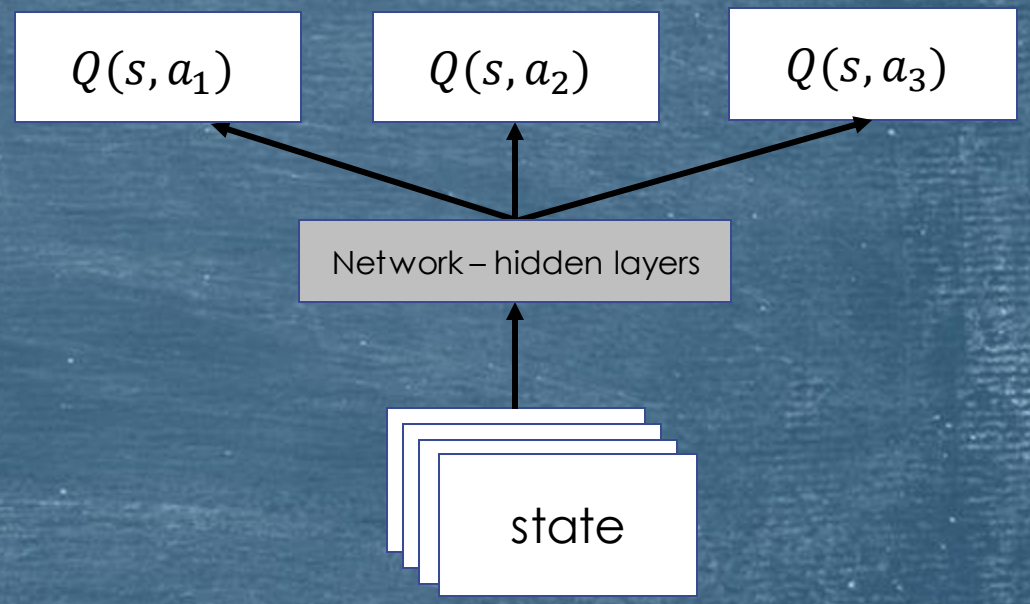


Reinforcement Learning using Neural Networks

Since the network architecture has changed, the target value must be changed.

So, our target value, y_s, must have the same number of elements as actions possible in the state s

Since we want to change the Q value only for the action we took, the target y value must be



$$y_s = [Q(s, a_1), Q(s, a_2), \ldots R(s, a_i, s') + \gamma \max_{a'} Q(s', a'), \ldots Q(s, a_n)]$$

Reinforcement Learning using Neural Networks

Some tips to make reinforcement learning work better with neural networks:

1. Use experience replay

   Why?

   Consecutive states are highly correlated. Usually, there is just a slight modification between two states.

   If these kind of states are used to train the neural network, than the network won't generalize well. The network might also forget what it had previously learn

   How?

   Use a buffer that stores as many records (s,a,r,s') as possible and sample from the buffer

Reinforcement Learning using Neural Networks

Some tips to make reinforcement learning work better with neural networks:

2. Use ϵ – greedy exploration

   Why?

   if we stop exploring, we might get stuck in local minimum

   Some states or actions might never be visited which means that we will satisfy with the first optimal policy

   How?

   Consider ϵ a probability of taking a random action.

   In most of the cases, we will take the action that gives as the biggest reward. However, with ϵ probability we will take a leap of faith

Example

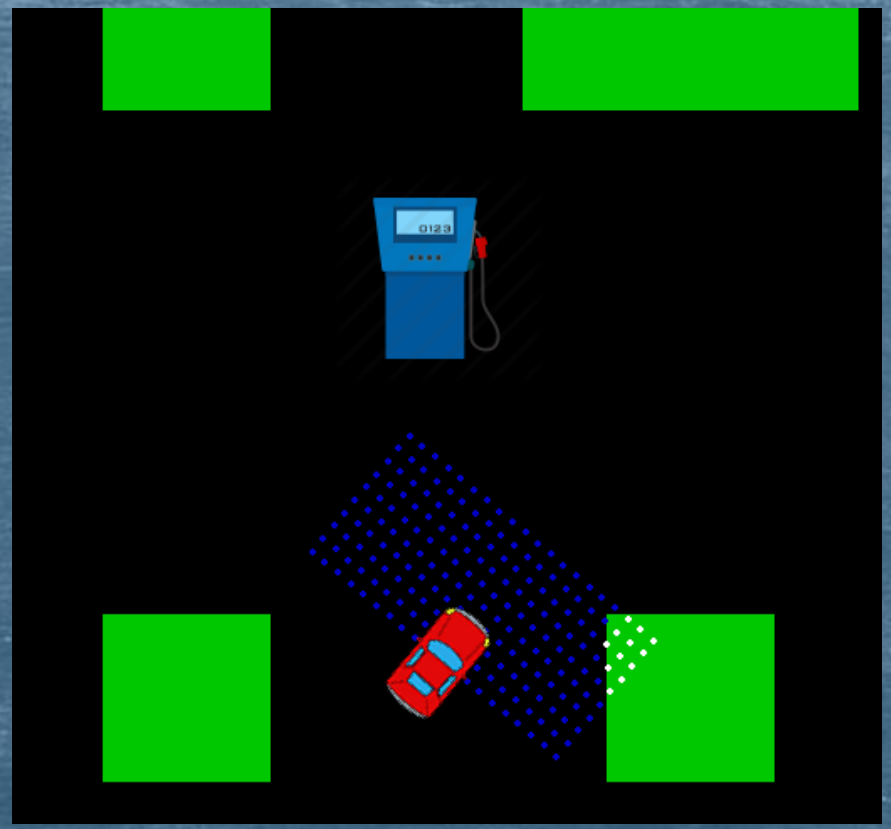
Example: Train a car to avoid obstacles and reach to the gas station

States: the car only knows what is in front and on its side.

It does that through 20 sensors that each can detect an object that is at most 10 units away

(it does not know where the gas station is) or how the map looks like

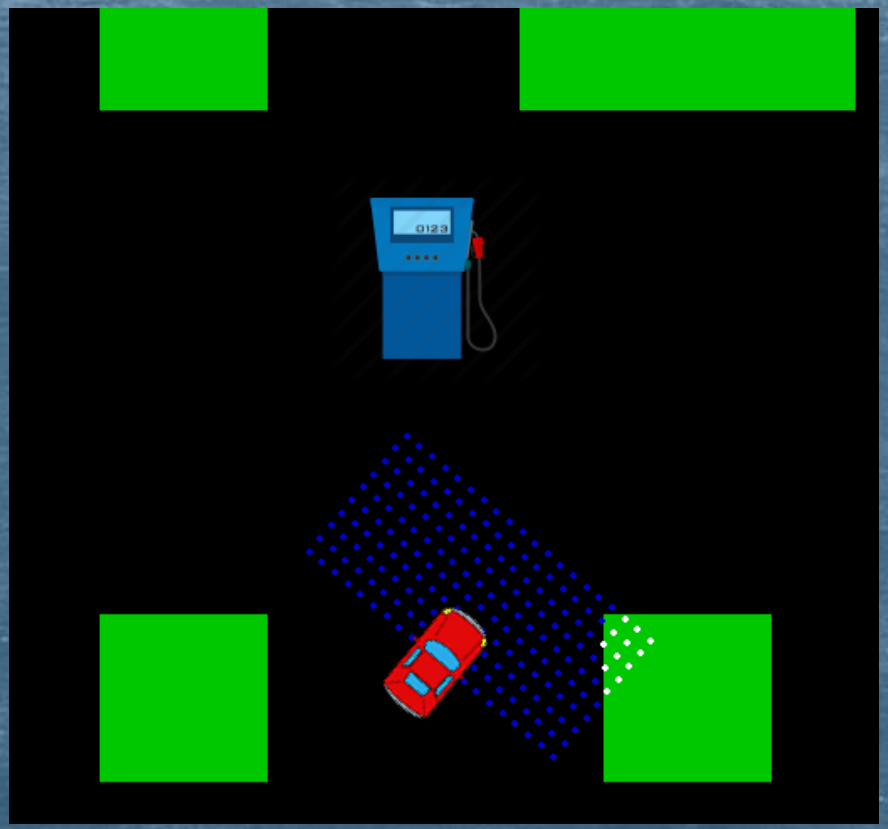
(This is more realistic)



Example

Actions:

The car always moves. It can, however, decide to move left , right or do nothing (move forward)



Example

- The rewards:

The car will get a big reward for reaching the gas station: 10000 and a big penalty for hitting an obstacle (-20000).

For the other states, the car will receive the following reward:

$$-2 + \frac{\sum_i s_i}{2}$$

Where $s_i \in [-10,10]$. A sensor will receive a negative value if it detects an obstacle and a positive value if it detects the gas station

The value of -2 is used to make the car search for the gas station.

Without it, the car will continue to go in circles, since it is safer.

Example

- Hyper-parameters:

  $\gamma = 0.9$

  This is clearly a problem where a feature reward is much more important than an immediate one. So γ must be closer to 1

  $\epsilon = variable$. The algorithm starts with a value of 1 and slowly decreases (with 0,00001 on every move) until it reaches 0.1

  So, at the beginning most of the moves will be random (this is in order to collect as many records as possible)

  Towards the end, the randomness part of the algorithm will almost disappear

Example

- Hyper-parameters:

  size of replay buffer: 50000

  This means that the buffer will store 50000 records (s,a,r,s').

  At each move, the car will first try a random move or a predicted one (based on ϵ) and then store the new resulted record in the replay buffer. If there are more than 50000 elements, the oldest one will be removed

  At each move, a batch of 100 elements, randomly selected from the replay buffer will be used to train the neural network.

Example

- Neural Network:

  The Neural network has 3 layers:

  input: 20

  hidden: 150

  output: 3

  The activation of the first 2 layers is rectifier linear, while the output is linear

  The learning is performed using RMSprop algorithm

  The batch size is 100.

90000 iterations were used to train the network.

Questions & Discussion

References

- "Reinforcement Learning: An Introduction", Richard S. Sutton , Andrew G. Barto, MIT Press, Cambridge, MA, 2017
- https://www.nervanasys.com/demystifying-deep-reinforcement-learning/
- https://gym.openai.com
- https://en.wikipedia.org/wiki/Q-learning
- https://medium.com/@harvitronix/using-reinforcement-learning-in-python-to-teach-a-virtual-car-to-avoid-obstacles-6e782cc7d4c6#.kp2idsahl
- http://outlace.com/Reinforcement-Learning-Part-3/
- https://www.youtube.com/watch?v=yNeSFbE1jdY&spfreload=10
- http://www.mcgovern-fagg.org/amy/courses/cs5033_fall2007/Lundgaard_McKee.pdf
- http://cs231n.stanford.edu/reports2016/121_Report.pdf