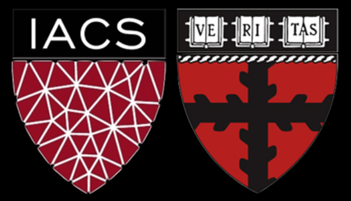
Lecture 33: Introduction to Reinforcement Learning 3

CS109B Data Science 2

Pavlos Protopapas, Mark Glickman, and Chris Tanner



Review: Policy Iteration vs Value Iteration

Includes: **policy evaluation** + **policy improvement**, and the two are repeated iteratively until policy converges.	Includes: **finding optimal value function** + one **policy extraction**. There is no repetition of the two because once the value function is optimal, then the policy out of it should also be optimal
It is based on the Bellman Expectation equation	It is based on the Bellman Optimality equation
The computation alternates between value and policy.	Each step gives a new value function. There is no explicit policy computed each step.
Every v from the loop corresponds to a valid policy π.	Each intermediate v may not correspond to any valid policy π.

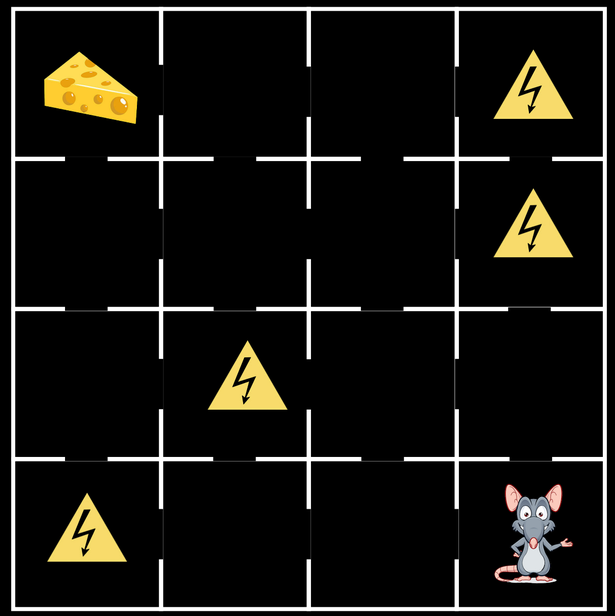
Contents

- Temporal Difference (TD) Learning
  - Motivation
  - Introduction
  - TD Prediction
  - TD Control
    - SARSA
- Q - Learning
- Deep Q - Learning

Contents

Until this point we have seen situations where we know the environment completely.

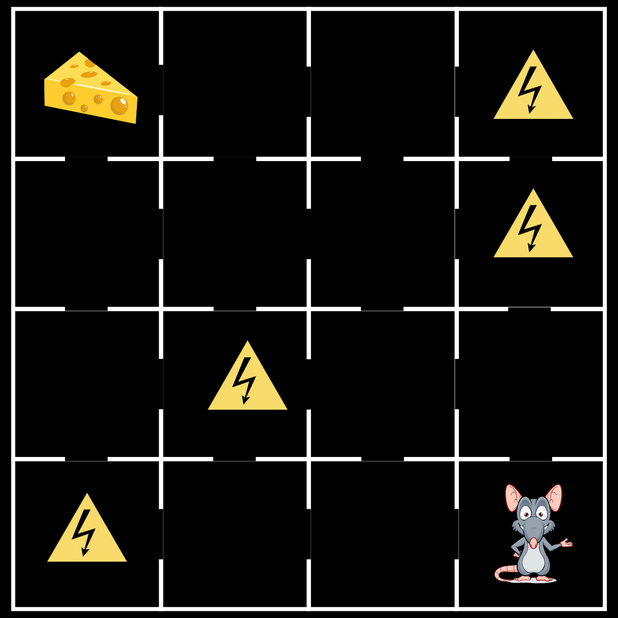
The mouse knew the states, the transition probability of the states and the rewards.



Until this point we have seen situations where we know the environment completely.

The mouse knew the states, the transition probability of the states and the rewards.

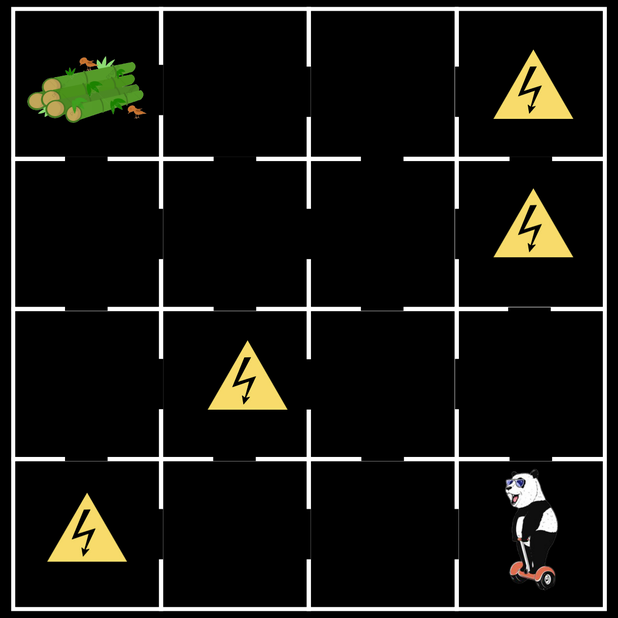
But, it is highly unlikely for an agent to know the entire Markov Decision Process (MDP).



Instead of a mouse, consider a Panda in a similar grid-like environment.

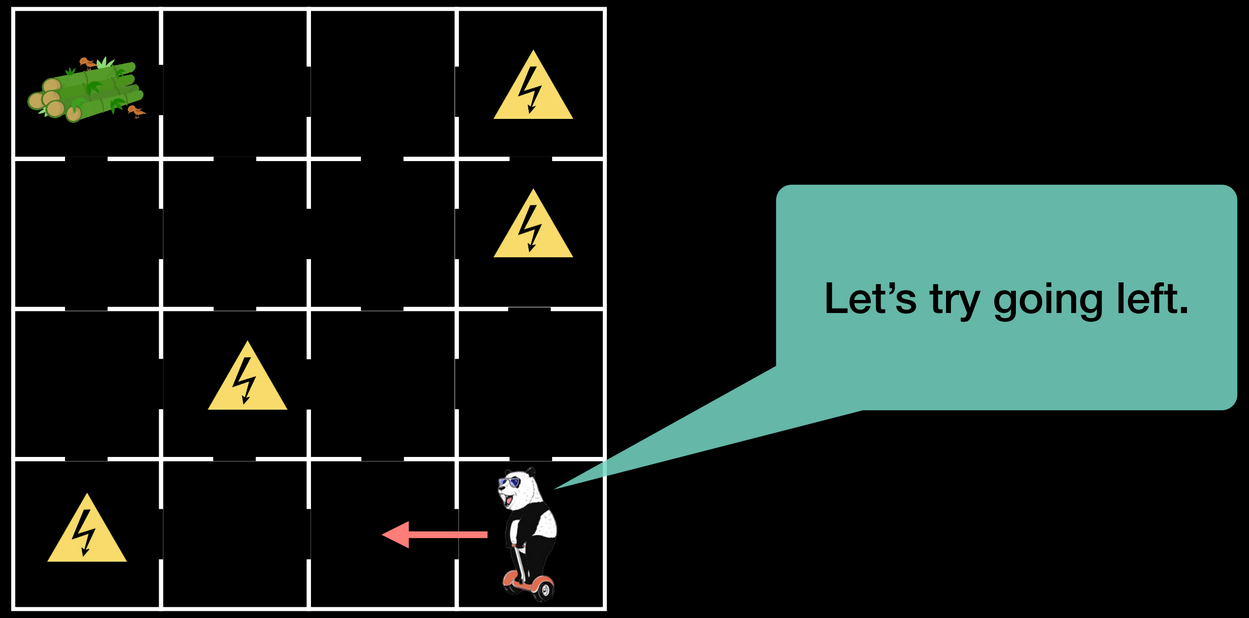
The goal remains the same, get the snack without getting an electric shock.

The difference is in the fact, that the Panda is smart but a tad-bit lazy, hence it will be using a hoverboard to navigate to the goal.



Now consider that the hoverboard has been tampered with. So if the Panda wants to go Up, it does so with a certain probability p and goes in the other 3 directions, with a probability

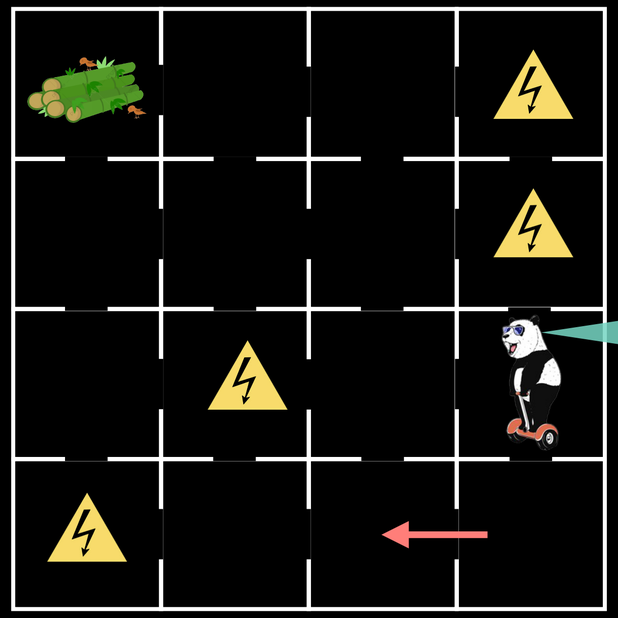
$$\frac{(1-p)}{3}.$$

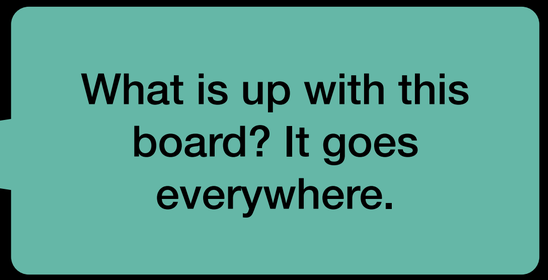


Now consider that the hoverboard has been tampered with. So if the Panda wants to go Up, it does so with a certain probability p and goes in the other 3 directions, with a probability

$$\frac{(1-p)}{3}.$$

The problem is the value of p is unknown.

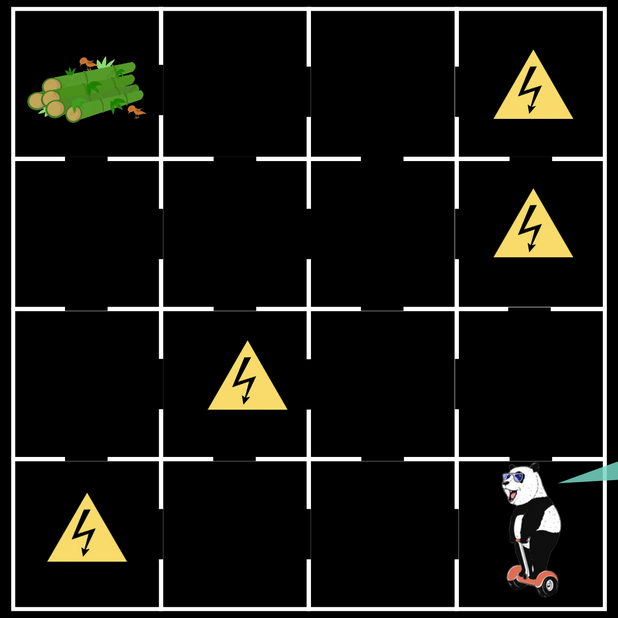




We cannot use dynamic programming here as its basis is that the transition probability P is known.

We need an alternative that works for model-free environments.

The simplest way is to just try the environment and learn from experience.



Fine by me, if you say I have infinite lives.

Temporal Difference Learning

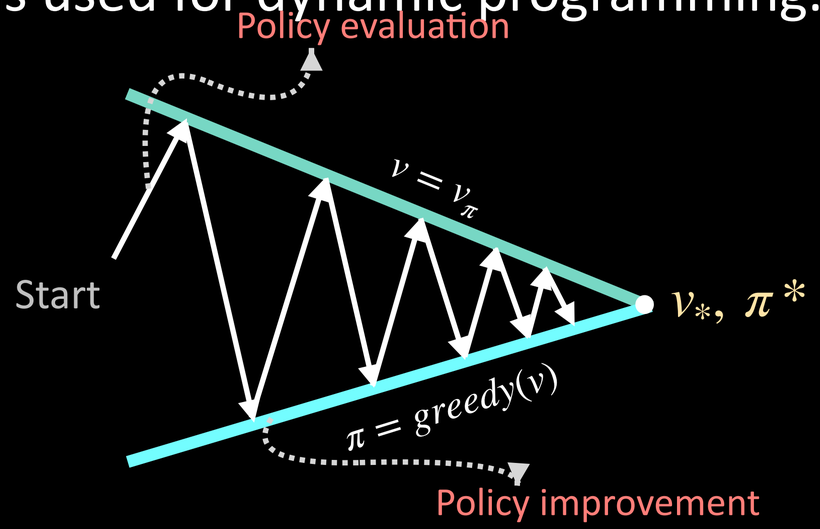
Contents

Temporal Difference (TD) Learning

Unlike dynamic programming, the TD methods learn directly from episodes of experience, as it is model-free.

Our aim now is to find the optimal policy, $\pi*$, given a policy π.

The TD methods use the Generalized Policy Iteration (GPI) - i.e. policy evaluation and policy improvement strategy which was used for dynamic programming.



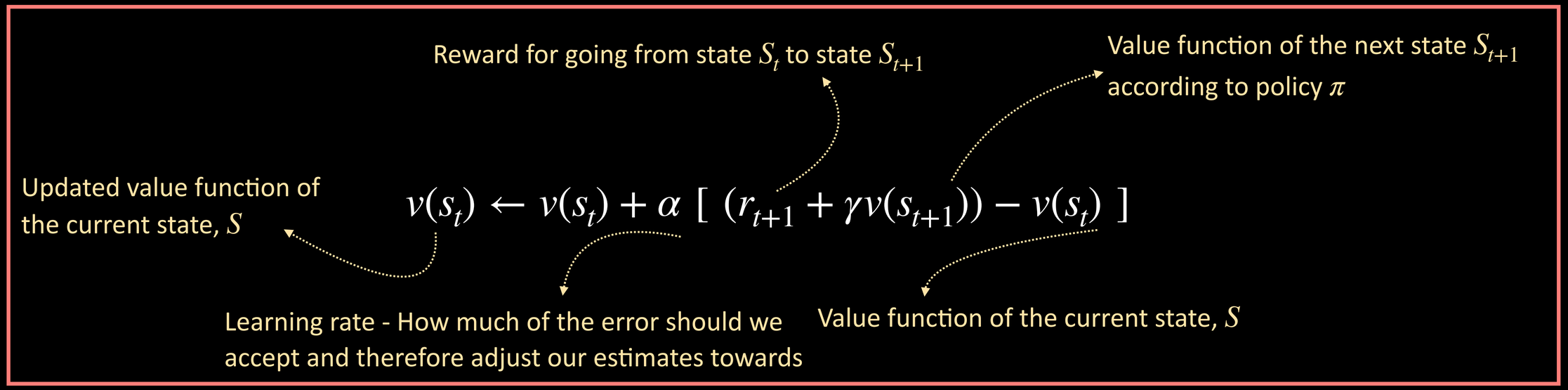
We first evaluate the given policy, π, and then improve greedily to get the optimal policy.

Contents

TD Prediction

TD Prediction is a policy evaluation method, that finds the value function given a policy, π.

For each state, s_t , we update the value function of the state, $v(s_t)$, immediately after transitioning to state s_{t+1}.



$r_{t+1} + \gamma v(s_{t+1})$ is the expected return based on the policy.

The term $(r_{t+1} + \gamma v(s_{t+1})) - v(s_t)$ is called the **TD error**.

PSUEDO CODE

INPUT - The policy π to be evaluated

Initialize $v(s)$, for all $s \in S$, arbitrarily except $v[terminal] = 0$

Loop for each episode:

Initialize s

Loop for each step of episode:

$a \leftarrow$ action given by π for s

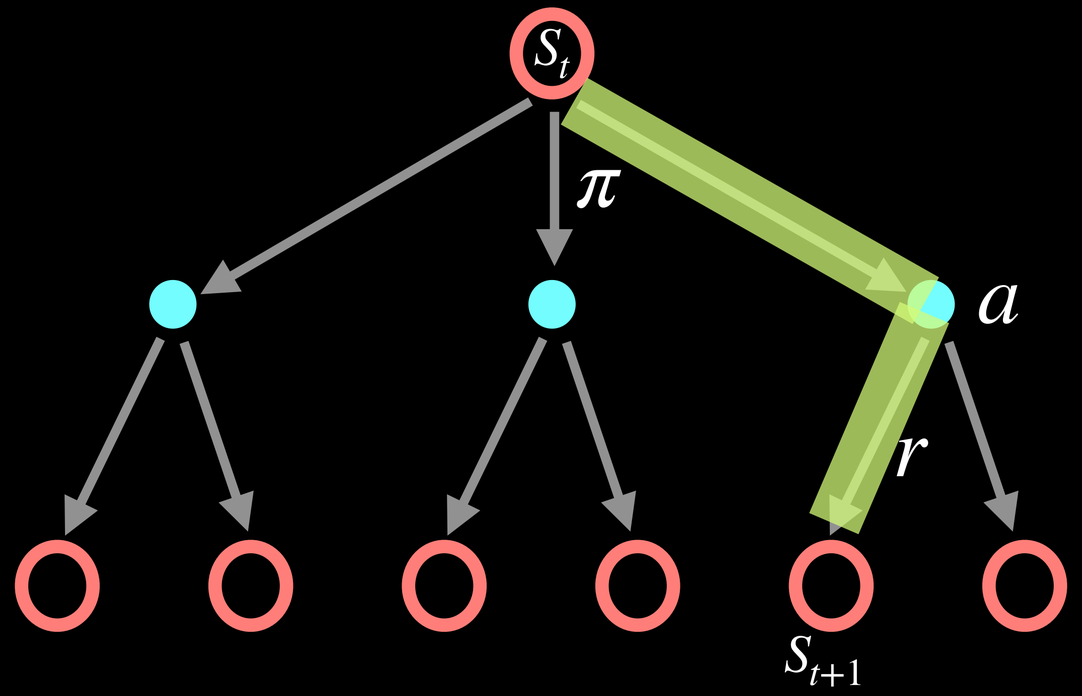
Take action a, observe r, s'

$v(s) \leftarrow v(s) + \alpha\ [\ (r + \gamma v(s')) - v(s)\ ]$

$s \leftarrow s'$

until s is terminal

Temporal Difference Backup



$$v(s_t) \leftarrow v(s_t) + \alpha \left[(r_{t+1} + \gamma v(s_{t+1})) - v(s_t) \right]$$

The TD methods can be used for non-episodic tasks as well.

Contents

- Temporal Difference (TD) Learning
  - Motivation
  - Introduction
  - TD Prediction
  - **TD Control**
    - SARSA
- Q - Learning
- Deep Q - Learning

TD Control

TD Control is a policy improvement method, that finds the optimal policy, $\pi*$, by acting greedily with the given value function based on a policy, π.

There is one major problem with this approach, if we use the state-value function, $v(s)$, then greedy improvement over $v(s)$ requires the model of MDP.

$$\pi'(s) = \underset{a \in A}{\operatorname{argmax}}\ r_{s,a} + p(\{s', r\} \mid s, a)\ v(s')$$

But, for a model-free environment, P is unknown.

Thus, we can only evaluate the model but not improve it.

Contents

- Temporal Difference (TD) Learning
  - Motivation
  - Introduction
  - TD Prediction
  - TD Control
    - **SARSA**
- Q - Learning
- Deep Q - Learning

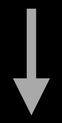
TD Control - SARSA (state–action–reward–state–action)

SOLUTION

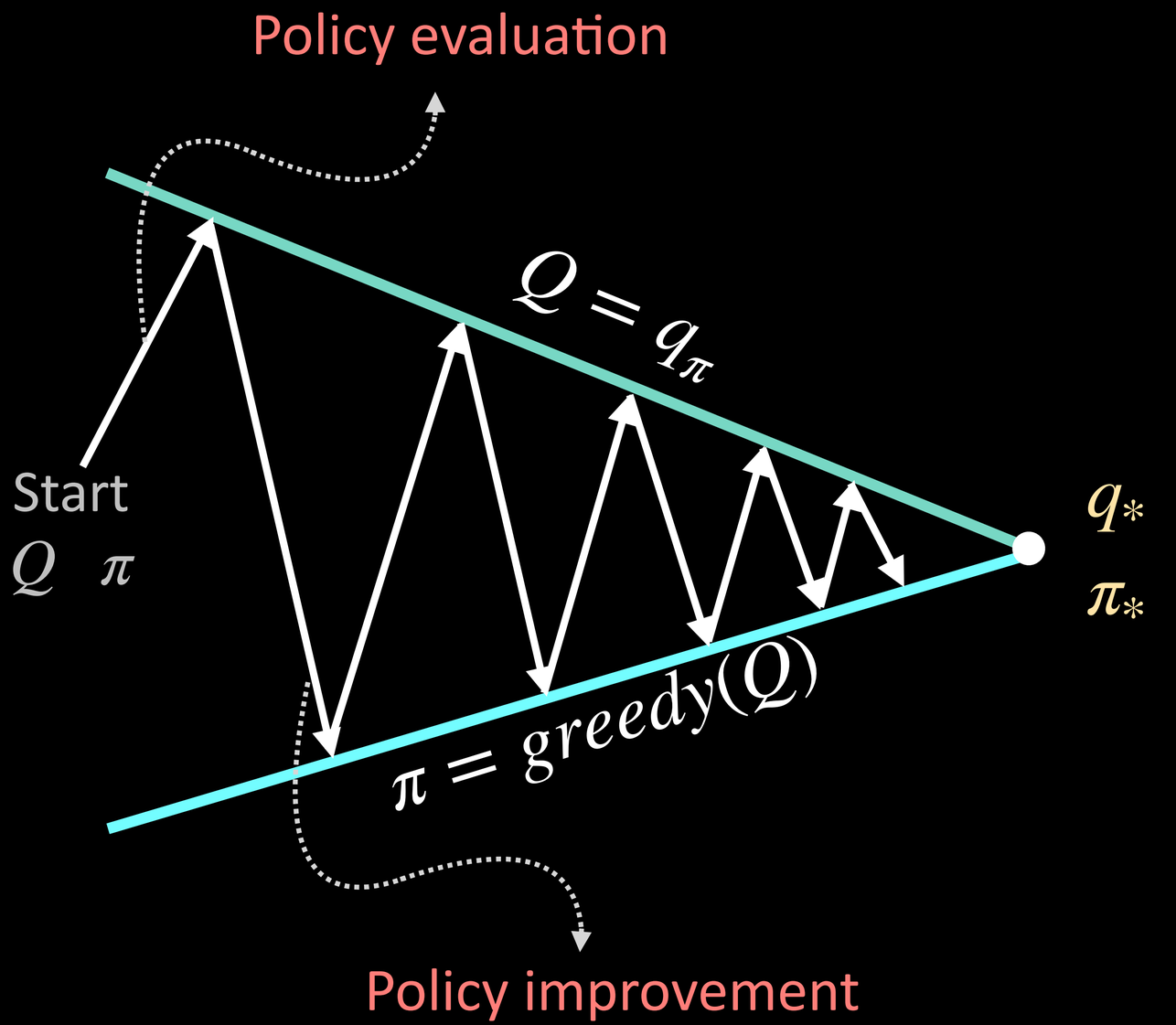
Use, the action-value function $Q(s, a)$ for both policy evaluation and improvement.

The steps remain the same, except we use the action-value instead of the state value.

$$\pi'(s) = \underset{a \in A}{\text{argmax}}\ Q(s, a)$$

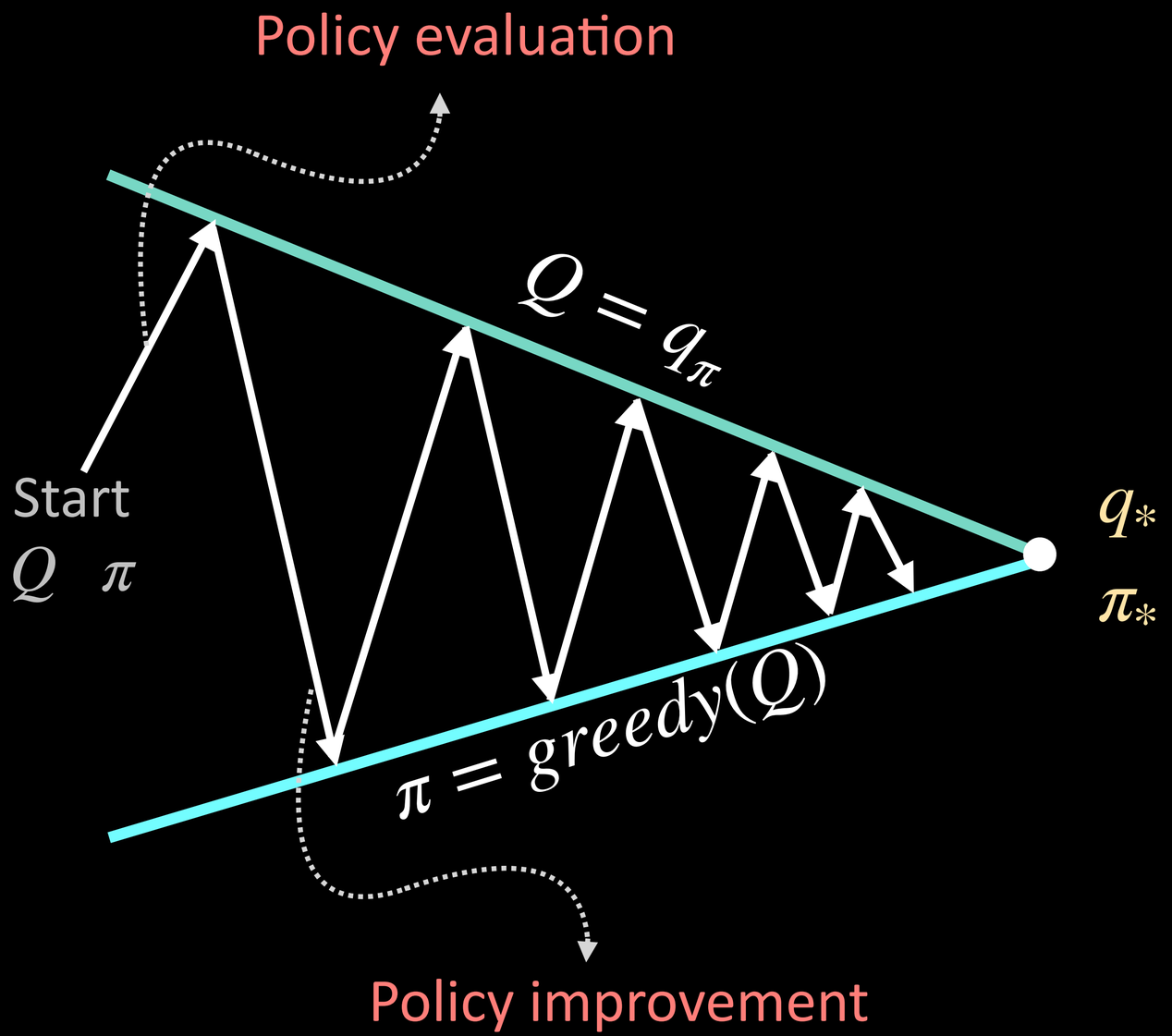


Policy Improvement



POLICY EVALUATION with Q

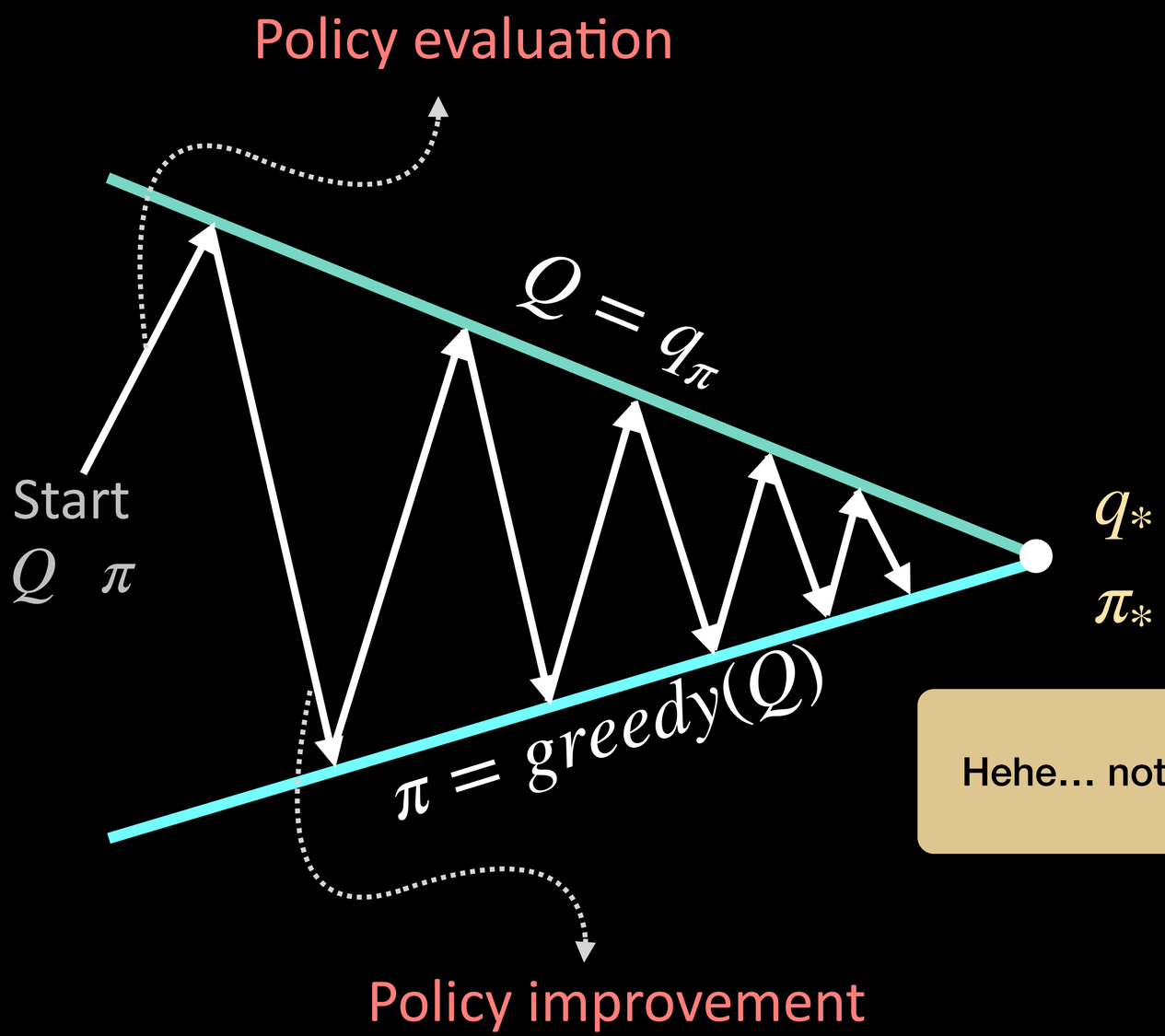
POLICY IMPROVEMENT with $greedy(Q)$



POLICY EVALUATION with Q

POLICY IMPROVEMENT with $greedy(Q)$

Finally, there is a way for me to figure out the best route to reach my bamboo.



POLICY EVALUATION with Q

POLICY IMPROVEMENT with $greedy(Q)$

Finally, there is a way for me to figure out the best route to reach my bamboo.

Hehe… not so fast

One major issue with this greedy approach is that, for a deterministic policy, we may never visit some state-action pairs.

Hence, the action-value function will not be updated, because of which the policy will not improve with experience.

One major issue with this greedy approach is that, for a deterministic policy, we may never visit some state-action pairs.

Hence, the action-value function will not be updated, because of which the policy will not improve with experience.

SOLUTION

One major issue with this greedy approach is that, for a deterministic policy, we may never visit some state-action pairs.

Hence, the action-value function will not be updated, because of which the policy will not improve with experience.

SOLUTION

Use exploration!

One major issue with this greedy approach is that, for a deterministic policy, we may never visit some state-action pairs.

Hence, the action-value function will not be updated, because of which the policy will not improve with experience.

SOLUTION

Use exploration!

With probability $1 - \epsilon$ choose the greedy action.

With probability ϵ choose a random action.

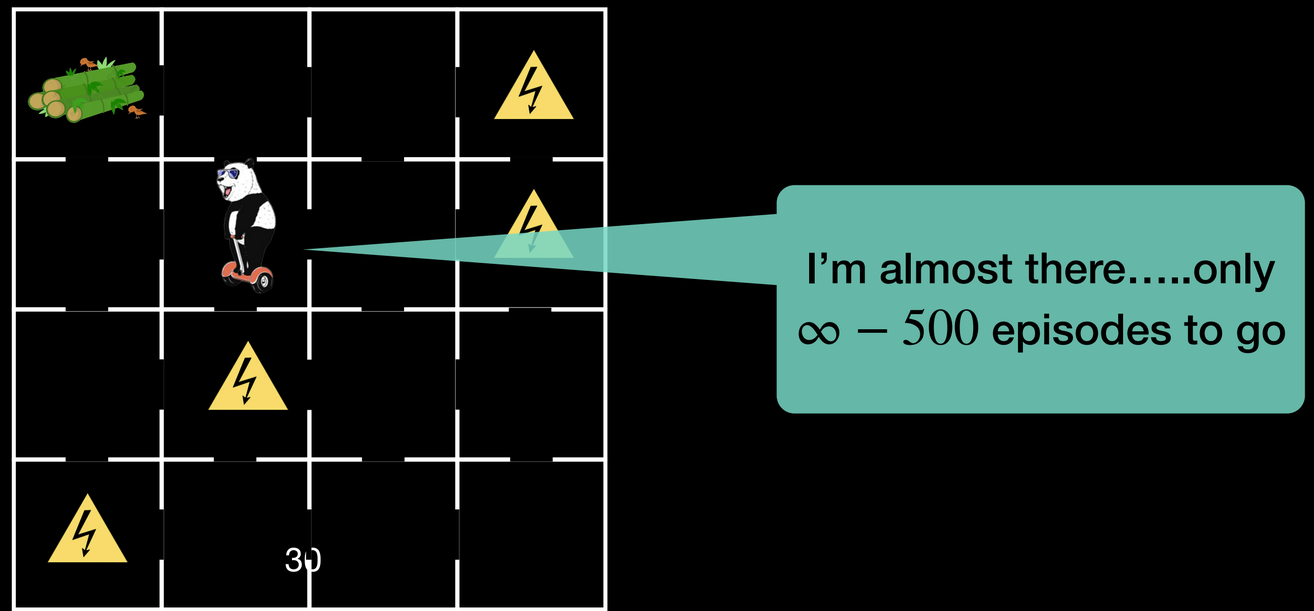
ϵ- GREEDY ALGORITHM

On-policy TD Learning

In on-policy TD learning, the agent learns the value of the policy π, that is used to make the decisions.

The value functions are updated using results from executing actions determined by some policy. These policies are usually "soft" and non-deterministic.

The meaning of "soft" in this sense, is that it ensures there is always an element of exploration to the policy.



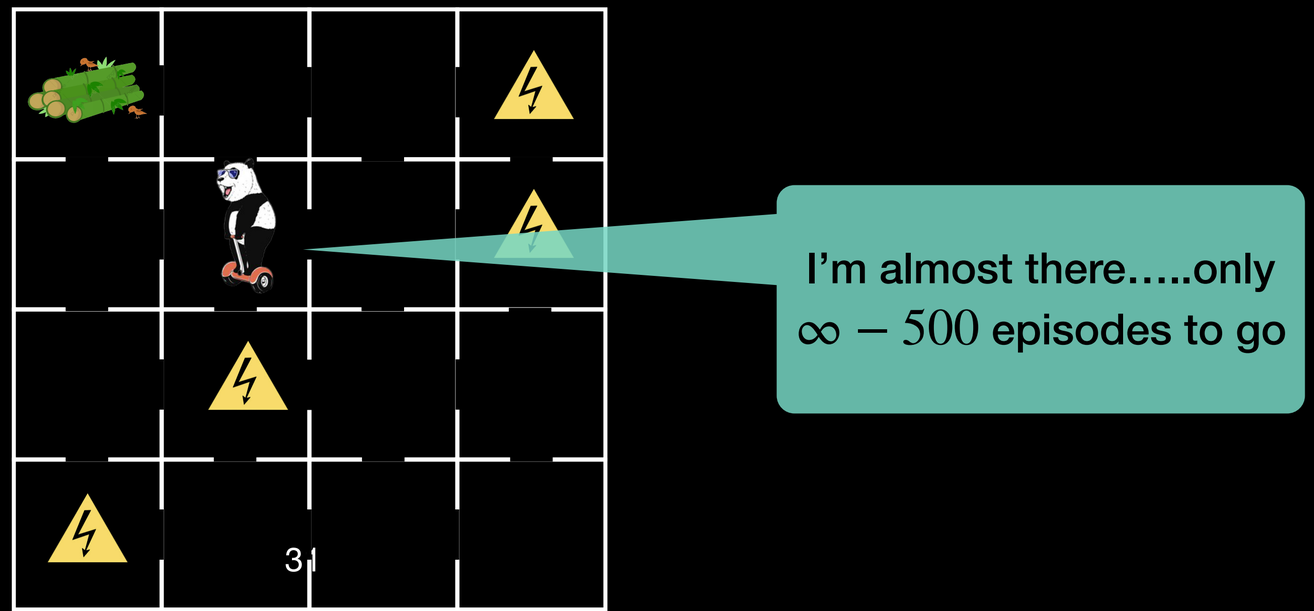
On-policy TD Learning

In on-policy TD learning, the agent learns the value of the policy π, that is used to make the decisions.

The value functions are updated using results from executing actions determined by some policy. These policies are usually "soft" and non-deterministic.

The meaning of "soft" in this sense, is that it ensures there is always an element of exploration to the policy.

SARSA is an example of On-policy learning

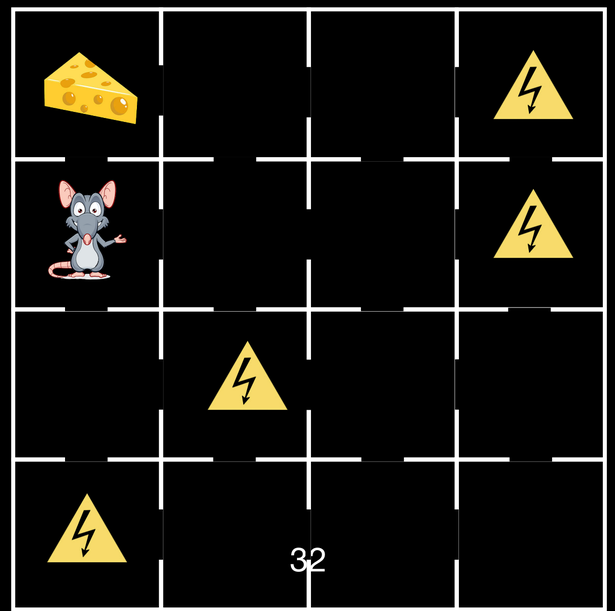


Off-policy TD Learning

Off-Policy methods can learn different policies for behavior and estimation.

These algorithms can update the estimated value functions using hypothetical actions, those which have not actually been tried.

An agent trained using an off-policy method may end up learning tactics that it did not necessarily exhibit during the learning phase.



Contents

- Temporal Difference (TD) Learning
  - Motivation
  - Introduction
  - TD Prediction
  - TD Control
    - SARSA
- **Q - Learning**
- Deep Q - Learning

Q - Learning

Q - Learning

Q-Learning is an Off-Policy algorithm for Temporal Difference learning.

Q-Learning learns the optimal policy even when actions are selected according to a more exploratory or even random policy.

Q - Learning

Q-Learning is an Off-Policy algorithm for Temporal Difference learning.

Q-Learning learns the optimal policy even when actions are selected according to a more exploratory or even random policy.

Here, we have 2 policies, a current policy, called behavior policy, μ and a target policy π, that we want to update.

INTUITION

Given that you are in a state s_t ,

Choose an action $a_{t+1} \sim \mu(\cdot \mid s_t)$

Choose an action $a' \sim \pi(\cdot \mid s_t)$

Update $q(S_t, A_t)$ towards the alternative action, target π

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a q(s_{t+1}, a') - q(s_t, a_t) \right]$$

POLICY EVALUATION

PSUEDO CODE

Initialize $q(s,a), \forall\ s \in S,\ a \in A(s)$ arbitrarily except $q[terminal, \cdot\ ] = 0$

Loop for each episode:

Initialize s

Loop for each step of episode:

Action according to behavior policy μ

$a \leftarrow$ action for s given by policy derived from q for s

Take action a, observe r, s'

$$q(s,a) \leftarrow q(s,a) + \alpha\ [\ (r + \gamma \max_{a'}\ q(s',a')) - q(s,a)\ ]$$

Update Q estimate with the sample data according to greedy policy for action selection.

$s \leftarrow s'$

until s is terminal

This can be visualised as a table, with rows as the state and columns as all possible action. Each cell represents the action-value corresponding to that state and action. This is called the Q-table.

Q - Table

The output of Q - learning is a Q - table with rows as the state and columns as all possible action. Each cell represents the action-value corresponding to that state and action.

The agent then refers to this “look-up” table to take an action given it is in a state s_t.

ACTION

STATE	a_1	a_2	a_3
s_1	10	-2.2	3
s_2	-0.6	-7	4
s_3	5	6.3	0.1
s_4	-9.5	13.4	-4
....	....	....	....
s_n	8.6	-1.5	7

Q - Table

The output of Q - learning is a Q - table with rows as the state and columns as all possible action. Each cell represents the action-value corresponding to that state and action.

The agent then refers to this “look-up” table to take an action given it is in a state s_t.

STATE	ACTION		
	a_1	a_2	a_3
s_1	10	-2.2	3
s_2	-0.6	-7	4
s_3	5	6.3	0.1
s_4	-9.5	13.4	-4
....	....	....	....
s_n	8.6	-1.5	7

Consider the agent is in state s_3. Now, the agent refers to this Q-table to take an action.

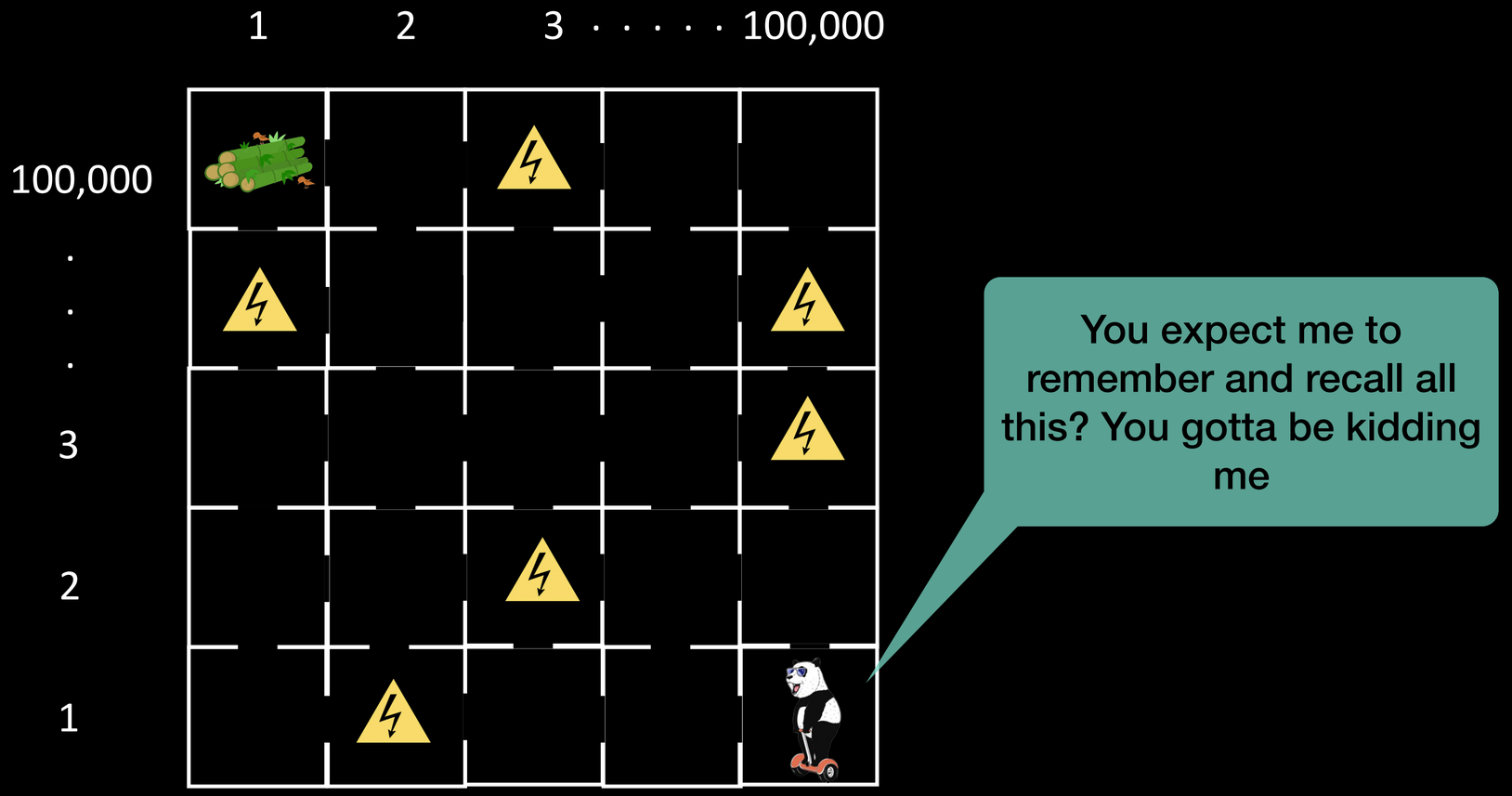
The Q-values associated to state s_3 are 5, 6.3 and 0.1. The agent takes the **argmax** of these values, which gives a_2 as the appropriate action.

Contents

- Temporal Difference (TD) Learning
  - Motivation
  - Introduction
  - TD Prediction
  - TD Control
    - SARSA
- Q - Learning
- **Deep Q - Learning**

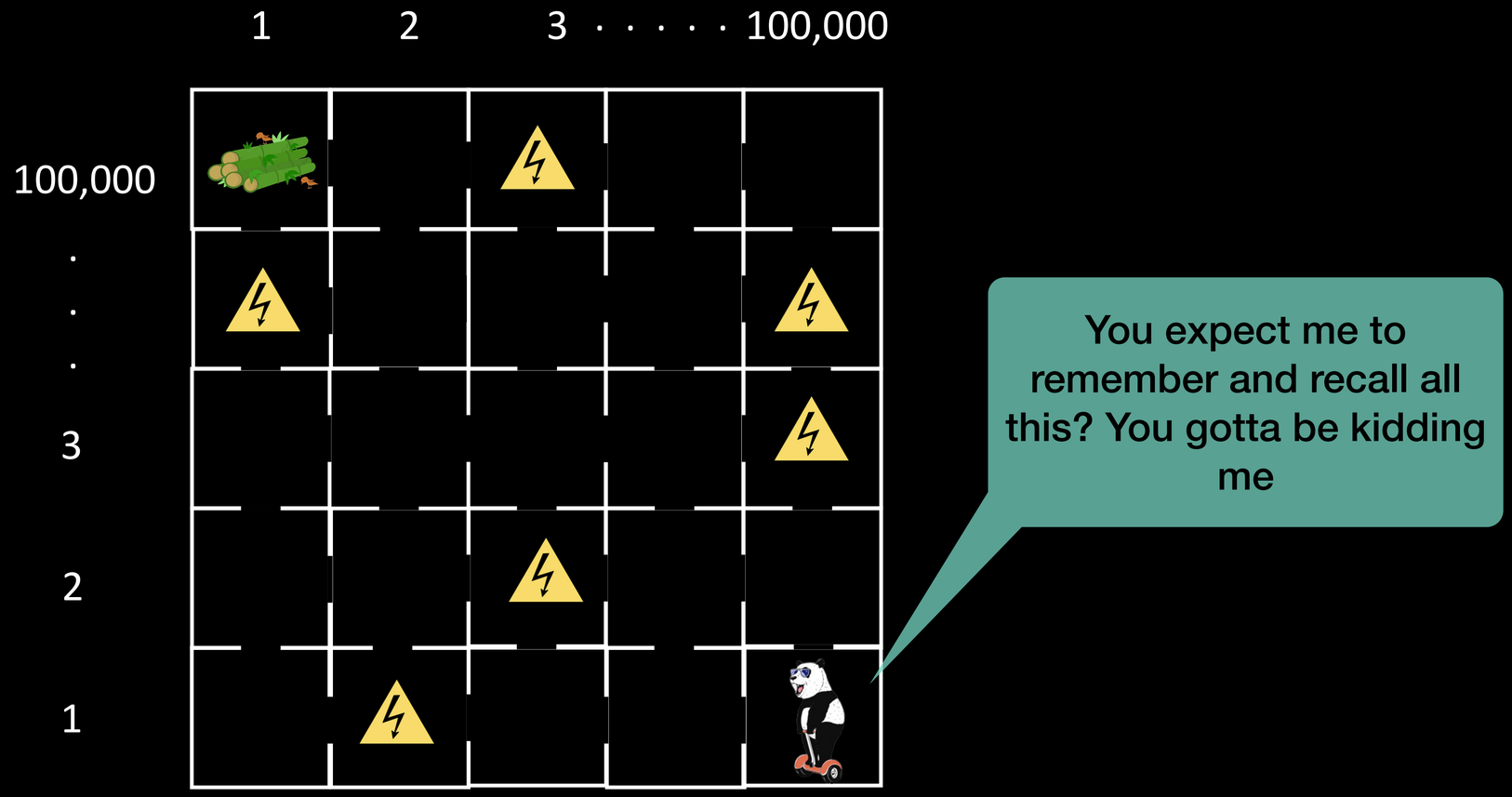
Deep Q Network (DQN)

For many problems, it is impractical to represent the Q-function as a table containing values for each combination of s and a.



For many problems, it is impractical to represent the Q-function as a table containing values for each combination of s and a.

Instead, we train a function approximator, such as a neural network with parameters θ to estimate the Q-values $Q(s, a; \theta) \approx Q^*(s, a)$

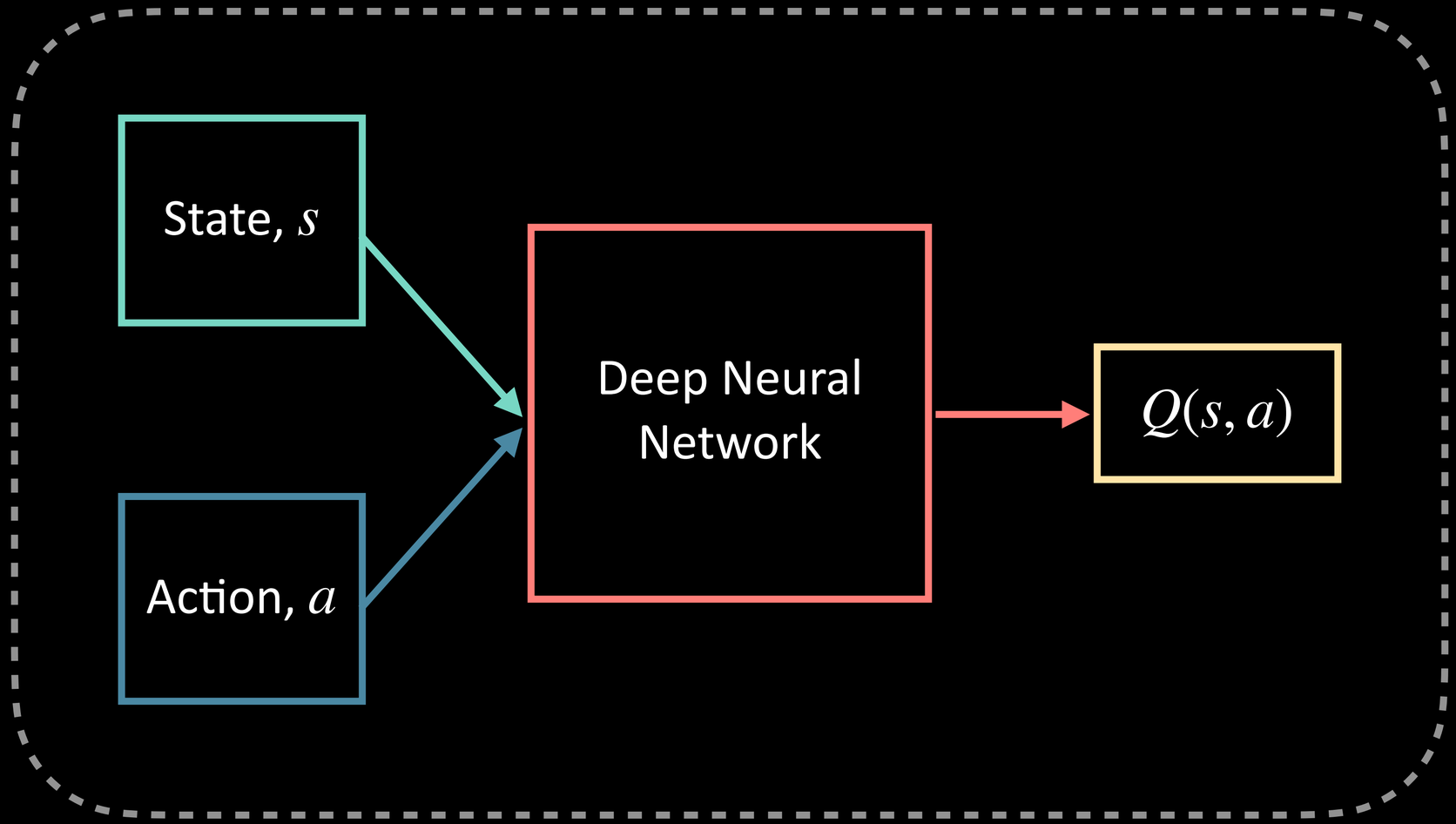


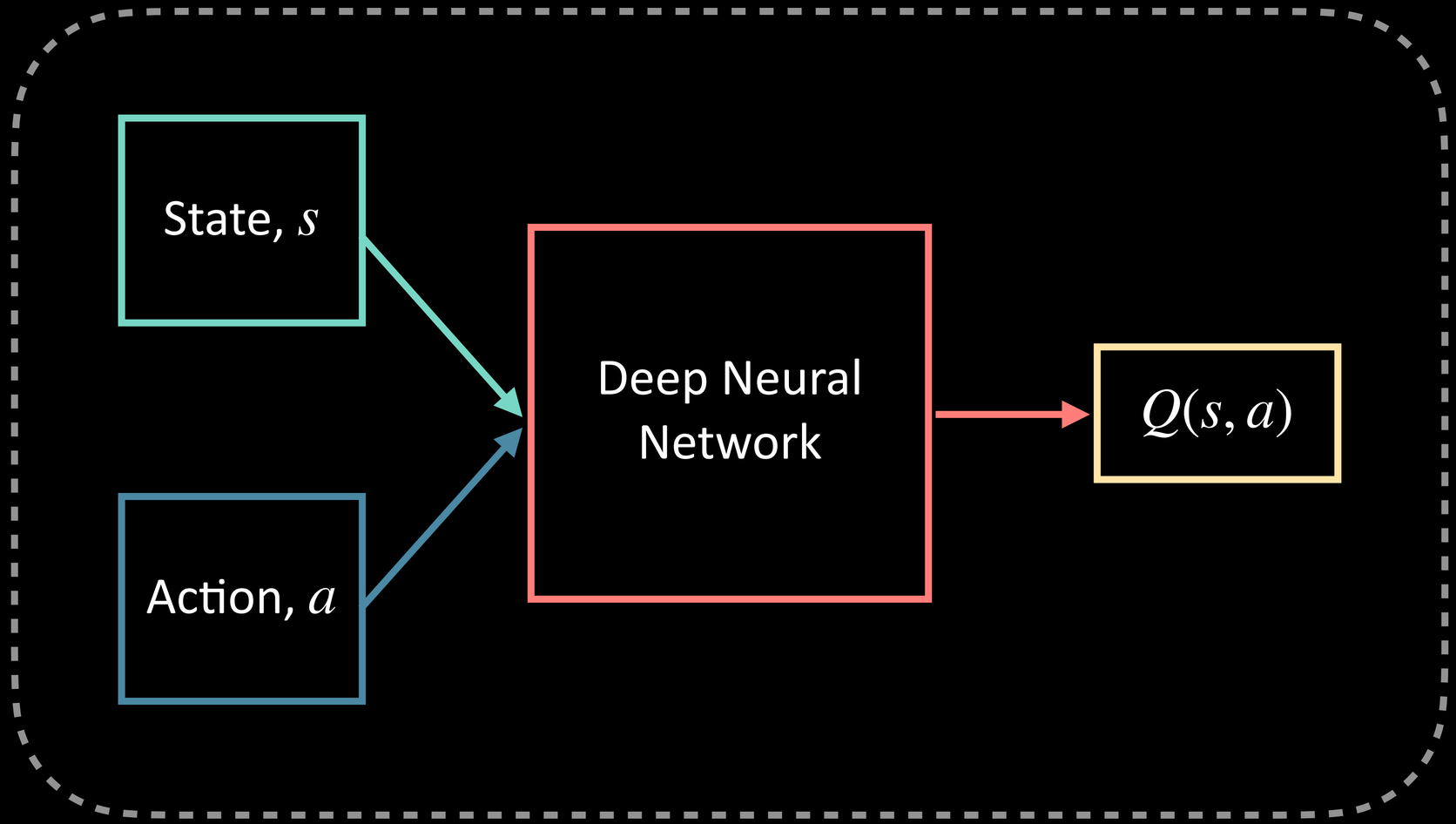
The DQN (Deep Q-Network) algorithm was developed by DeepMind in 2015.

It was able to solve a wide range of Atari games (some to superhuman level) by combining reinforcement learning and deep neural networks at scale.

The algorithm was developed by enhancing a classic RL algorithm called Q-Learning with deep neural networks and a technique called *experience replay*.

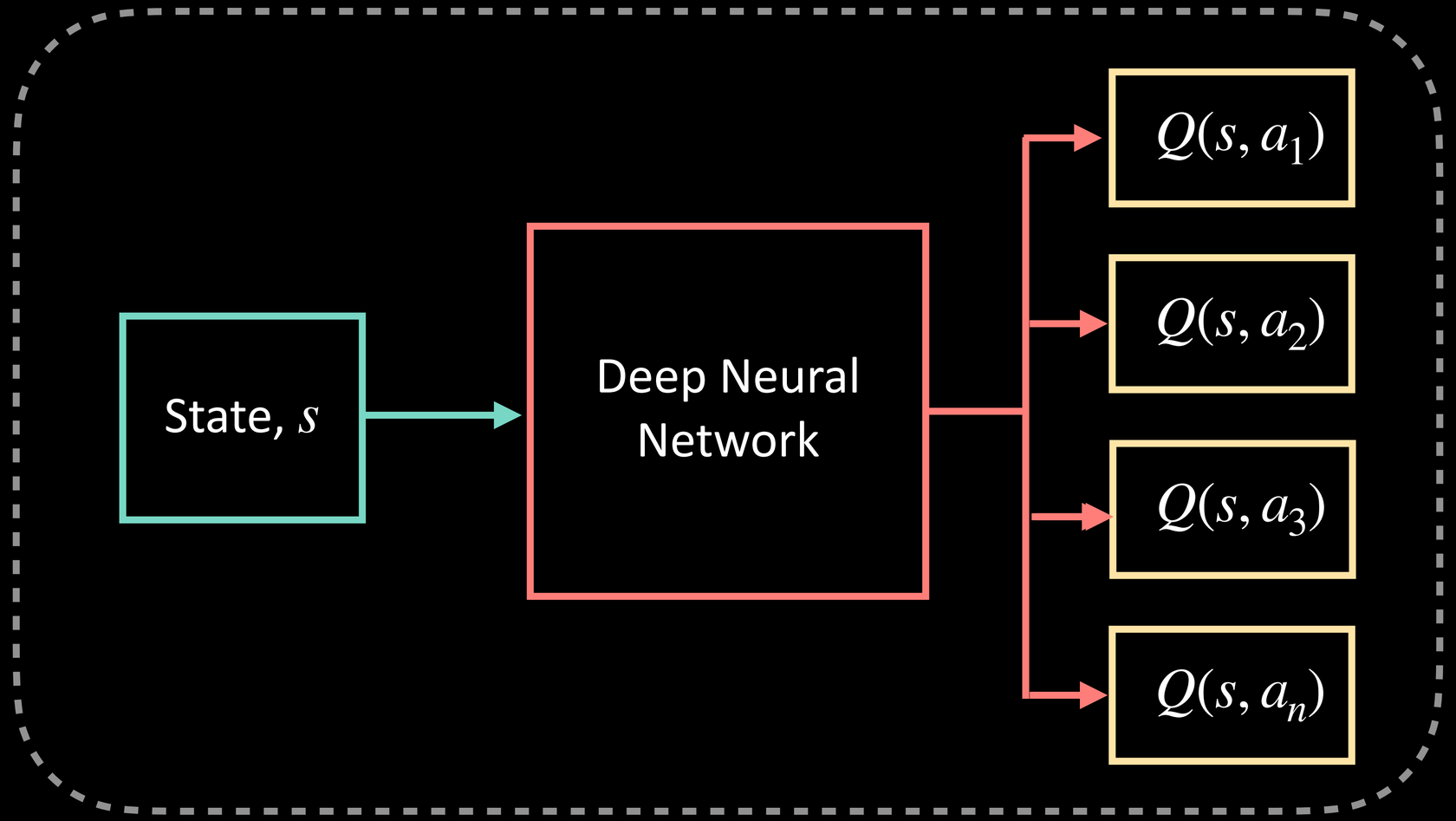
DQN replaces the standard Q - table by a Deep Neural Network which maps environment states to actions.


State, s
Action, a
Deep Neural Network
$Q(s, a)$

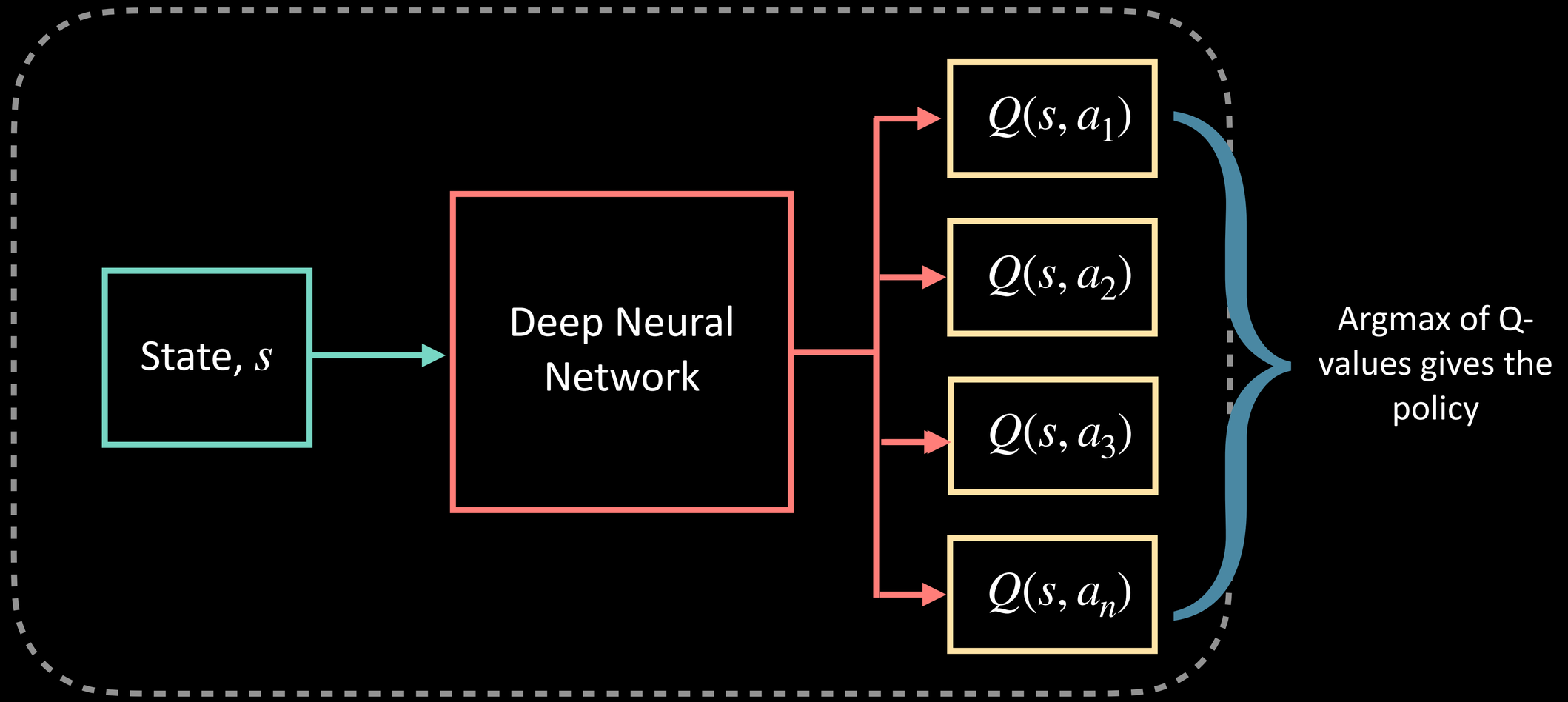


There is one problem with this, if we want to update the policy, then we would like to try all the possible actions.

So, for n actions, we would have to run this network n times.



An alternative is to have a network that takes as input only the state and returns the Q-value for each action.



An alternative is to have a network that takes as input only the state and returns the Q-value for each action.

DQN: Training

To train the network we need some data.

This data is got by executing the random policy in the environment for a few steps, recording the actions, states and the return.

DQN: Training

To train the network we need some data.

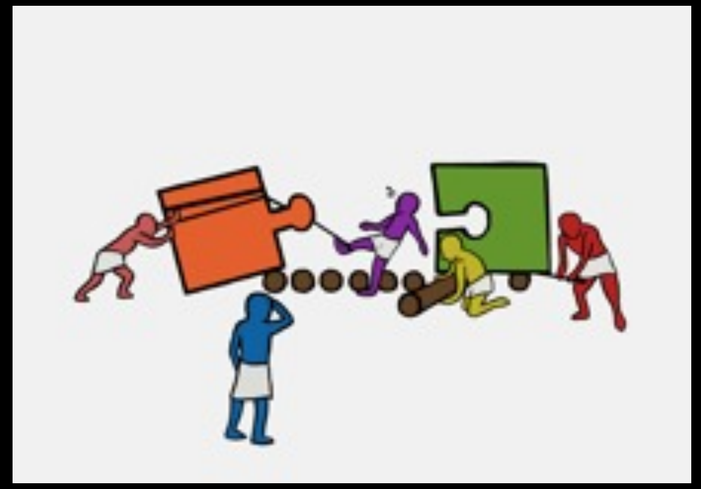
This data is got by executing the random policy in the environment for a few steps, recording the actions, states and the return.

The loss of the neural network is a simple MSE of the following form:

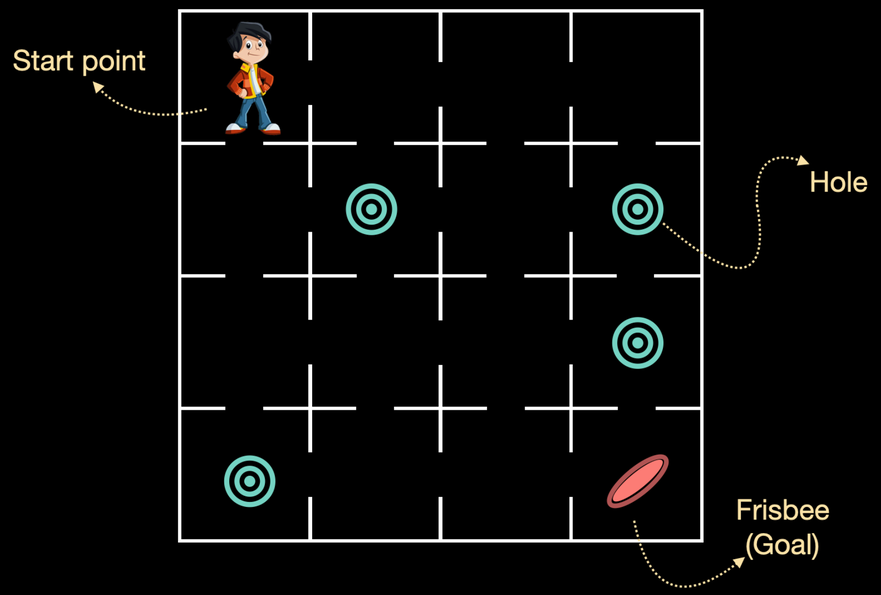
$$\mathscr{L} = \mathbb{E}\left[\left\| \underbrace{(r + \gamma \max_{a'} Q(s', a'))}_{\text{TARGET}} - \underbrace{Q(s, a)}_{\text{PREDICTION}} \right\|^2 \right]$$

Q-Loss

Exercise: Q-Learning using DQN



The aim of this exercise is to find the Q-value given an environment. For this, we will be using a pre-defined environment by OpenAI Gym called FrozenLake-v0.



We will get the Q-value of the environment by implementing a Deep Q Network.