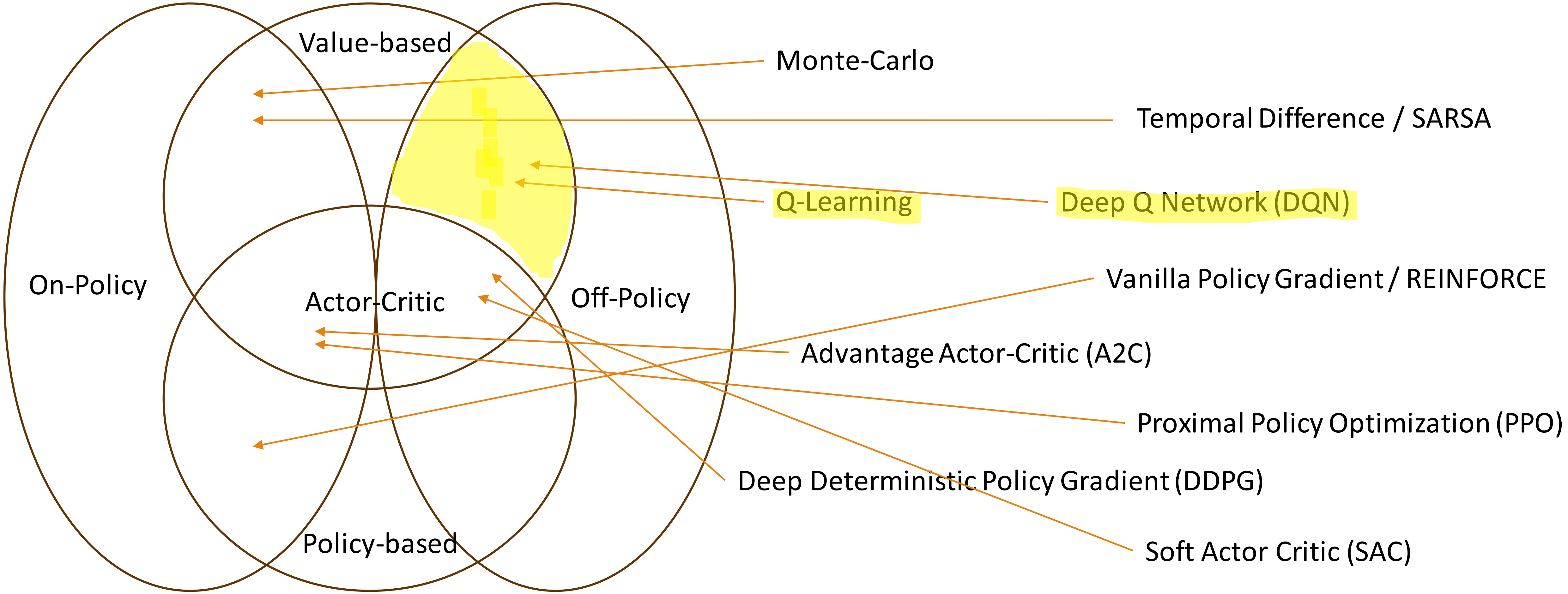
지능 시스템
Intelligent Systems

Lecture 4 – Q-Learning

Today’s Contents

- Off-Policy Algorithm
- Q-Learning
- Introduction to Deep Q Network (DQN)

Some Popular Algorithms



Value-based Algorithm

Example Algorithm for a Value-based Method:

1. Start with an **Initial Policy, $\boldsymbol{\pi_0}$,** that is random.

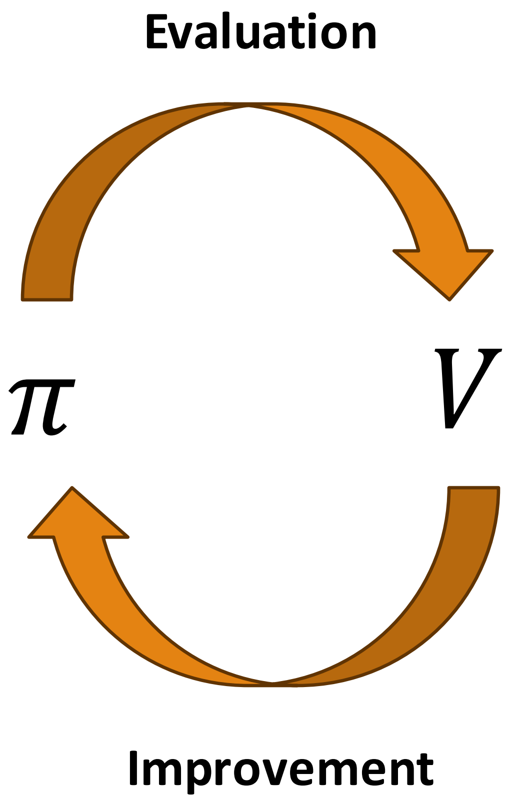
2. Collect a **Sample Trajectory**

3. Use the collected sample to **compute Values** of the states the agent has visited.

4. Update the policy;

$$\pi_1 \leftarrow UPDATE(\pi_0)$$

5. Repeat until convergence



On-Policy VS Off-Policy

On-Policy
- “Learn on the job”
- Learn about a policy from experiences sampled from that policy.

Off-Policy
- “Look over someone’s shoulder”
- Learn about a policy from experiences sampled from other policies.

Bellman Equation for Q

Bellman Equation of Q for one step bootstrapping

$$Q^{\pi}(s_t, a_t) = r_t + \mathbb{E}_{a_t \sim \pi(a_t|s_t), s_{t+1} \sim p(s_{t+1}|s_t, a_t)}[\gamma Q^{\pi}(s_{t+1}, a_{t+1})|s_t, a_t]$$

We can write a simplified version of the above Bellman equation as follows;

$$\mathrm{Q}(s_t, a_t) \cong r_t + \gamma \mathrm{Q}(s_{t+1}, a_{t+1})$$

Note that we have depreciated the expectation symbol since in the upcoming case, we are only considering a single step

SARSA – On-Policy Algorithm

$$Q(s_t, a_t) \cong r_t + \gamma Q\boxed{(s_{t+1}, a_{t+1})}$$

Next state

Next action

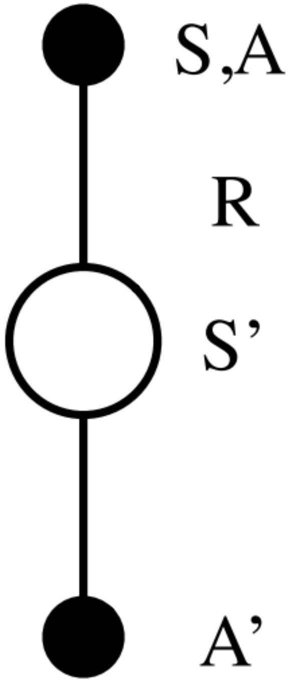
Data of 1 step

$$s_t, a_t, r_t, s_{t+1}$$

An usual 1 step sample is not enough!
We need the next action.

We can easily obtain the next action by using out policy!

We call the algorithm that uses this information a **SARSA**



SARSA Algorithm

SARSA Algorithm

1. Start with an **Initial Policy, $\boldsymbol{\pi_0}$**
2. Collect a **Sample Step**

$$s_t, a_t, r_t, s_{t+1}$$

3. Evaluate the current policy at state, s_{t+1}
4. Compute **Temporal Difference Target** at the state

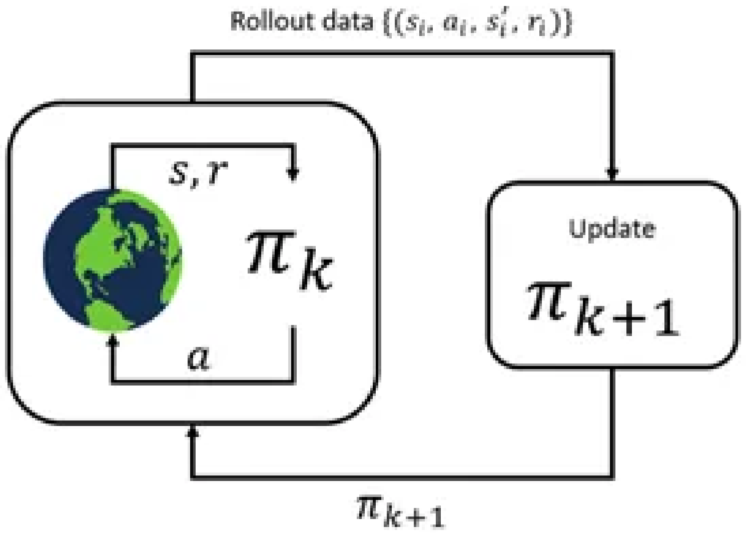
$$r_t + \gamma \mathrm{Q}(s_{t+1}, a_{t+1})$$

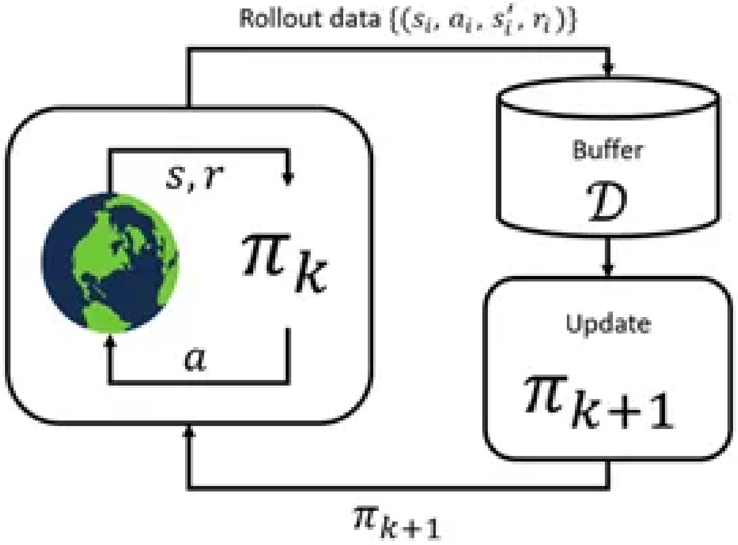
5. Update the **Q function**

$$\mathrm{Q}(s_t, a_t) \leftarrow \mathrm{Q}(s_t, a_t) + \alpha\big(r_t + \gamma \mathrm{Q}(s_{t+1}, a_{t+1}) - \mathrm{Q}(s_t, a_t)\big)$$

6. Update the ***ε*-greedy policy**
7. Repeat until convergence

Off-Policy Algorithm Overview





Off-Policy algorithms can re-use rollout data generated by different policies.

It has improved sample efficiency compared to On-Policy algorithms.

Behavior and Target Policy

Behavior Policy: Policy that selects & performs an action that influences the environment and collect samples

Target Policy: Policy that is used only when computing the target value.

SARSA Algorithm – Policies

SARSA Algorithm

1. Start with an **Initial Policy, $\boldsymbol{\pi_0}$**
2. Collect a **Sample Step**

$$s_t, a_t, r_t, s_{t+1}$$

3. Evaluate the current policy at state, s_{t+1}
4. Compute **Temporal Difference Target** at the state

$$r_t + \gamma Q(s_{t+1}, a_{t+1})$$

5. Update the **Q function**

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\big)$$

6. Update the **$\boldsymbol{\varepsilon}$-greedy policy**
7. Repeat until convergence

Behavior Policy:
Chooses a_t and **perform** it in the environment.

Target Policy:
Chooses a_{t+1} for the need of **computing the TD target**. The action is not performed in the environment.

Off-Policy

The algorithm is **On-Policy** if
Behavior Policy = Target Policy

The algorithm is **Off-Policy** if
Behavior Policy $\neq$ Target Policy

Off-Policy Advantages:

- Target can be computed by previous policies of the agent or different policy of other agents
  - ❑ Re-use past experiences
  - ❑ Sample Efficient
- Increased Exploration
  - ❑ Act greedy, but explore with the target!
- Re-evaluate past experiences
  - ❑ When confronted with similar states from the past, it can re-evaluate the past experiences

Q-Learning Target

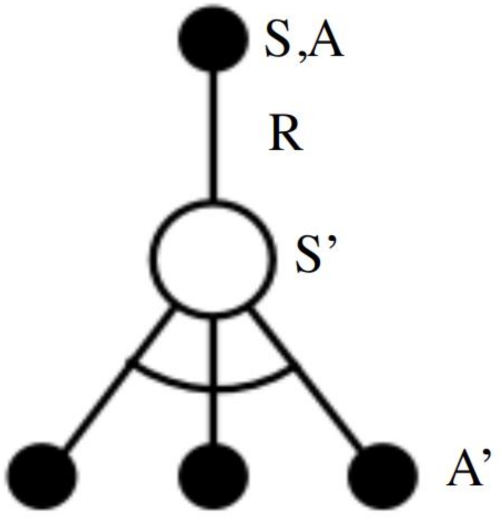
$$Q(s_t, a_t) \cong r_t + \gamma \boxed{Q(s_{t+1}, a^*)} \qquad \text{where, } a^* = \underset{a}{\operatorname{argmax}}\, Q(s_{t+1}, a)$$

Next state

Alternative successor action

$$Q(s_t, a_t) \cong r_t + \gamma Q(s_{t+1}, a^*) = r_t + \gamma \boxed{\max_a Q(s_{t+1}, a)}$$

Q-Learning takes the best possible value of Q as the target.



Q-Learning Algorithm

Q-Learning Algorithm

1. Start with an **Initial Policy, $\boldsymbol{\pi_0}$**
2. Collect a **Sample Step via $\boldsymbol{\varepsilon}$-greedy policy, use the Bahavior Policy when selecting the action**

$$s_t, a_t, r_t, s_{t+1}$$

3. Using the Target Policy, compute $a^* = \underset{a}{\operatorname{argmax}}\, Q(s_{t+1}, a)$
4. Compute **Temporal Difference Target** at the state

$$r_t + \gamma \mathrm{Q}(s_{t+1}, a^*)$$

5. Update the **Q function**

$$\mathrm{Q}(s_t, a_t) \leftarrow \mathrm{Q}(s_t, a_t) + \alpha\big(r_t + \gamma \mathrm{Q}(s_{t+1}, a^*) - \mathrm{Q}(s_t, a_t)\big)$$

6. Update the **Behavior $\boldsymbol{\varepsilon}$-greedy policy**
7. Repeat until convergence

SARSA VS Q-Learning

SARSA:
Can take **any action** when computing the target.

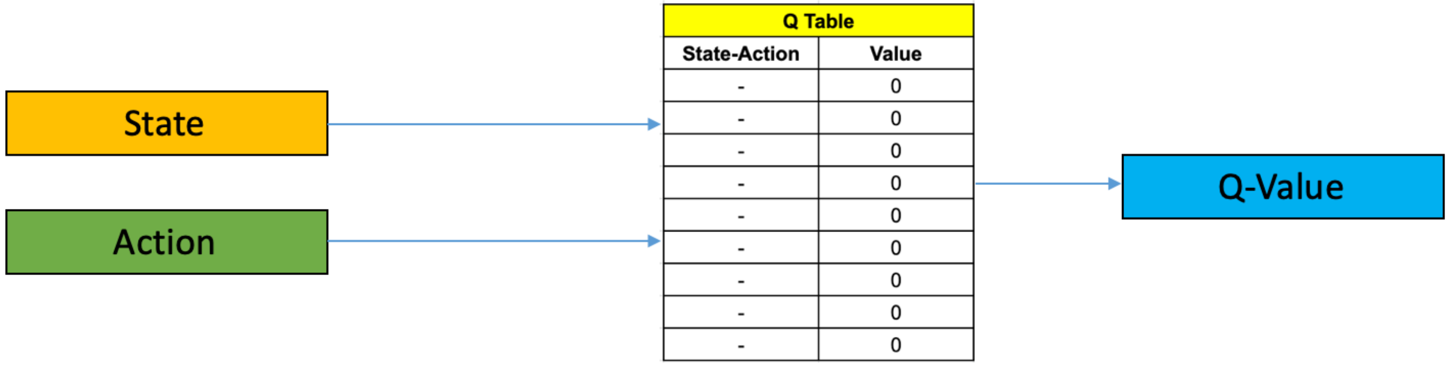
R = -1	R = -1	R = -1	R = -1
R =	R		R =
R =	R = -1	R = -1	R =
Sta	R = -100	R = -100	R = 10

Q-Learning:
Only consider the **optimal action** when computing the target.

R = -1	R = -1	R = -1	R = -1
R = -1	R = -1	R = -1	R = -1
R =	R		R =
Sta	R = -100	R = -100	R = 10

Introduction to Deep Q-Network (DQN)

Structure of a classic Q estimator



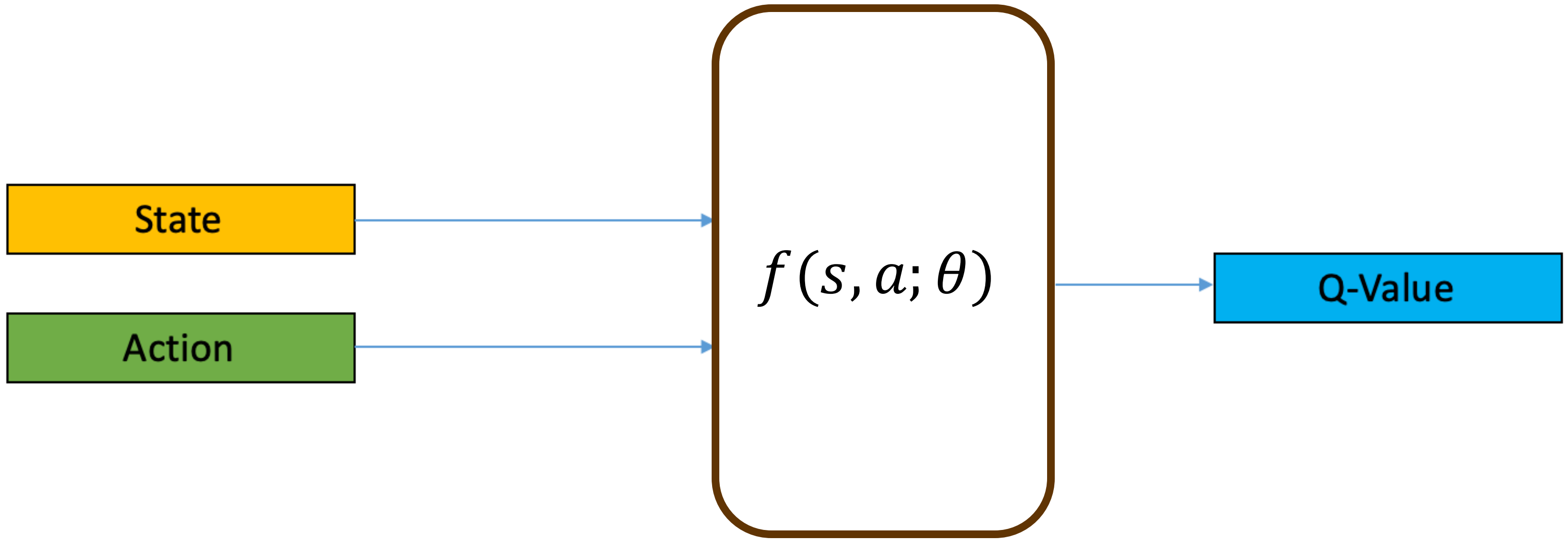
Q Learning

Q table example

Q	Dealer showing 6									
	12	13	14	15	16	17	18	19	20	21
Hit	0.9	0.8	0.7	0.7	0.6	0	0	0	0	0
Stick	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1

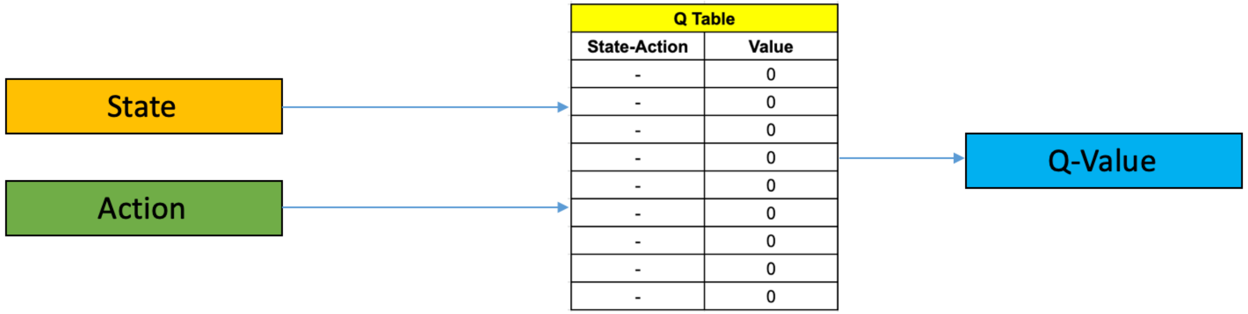
Q	Dealer showing 10									
	12	13	14	15	16	17	18	19	20	21
Hit	0.95	0.95	0.95	0.9	0.8	0.7	0.5	0.4	0.3	0
Stick	-0.95	-0.9	-0.9	-0.8	-0.7	0.2	0.5	0.7	0.8	1

Function Approximator



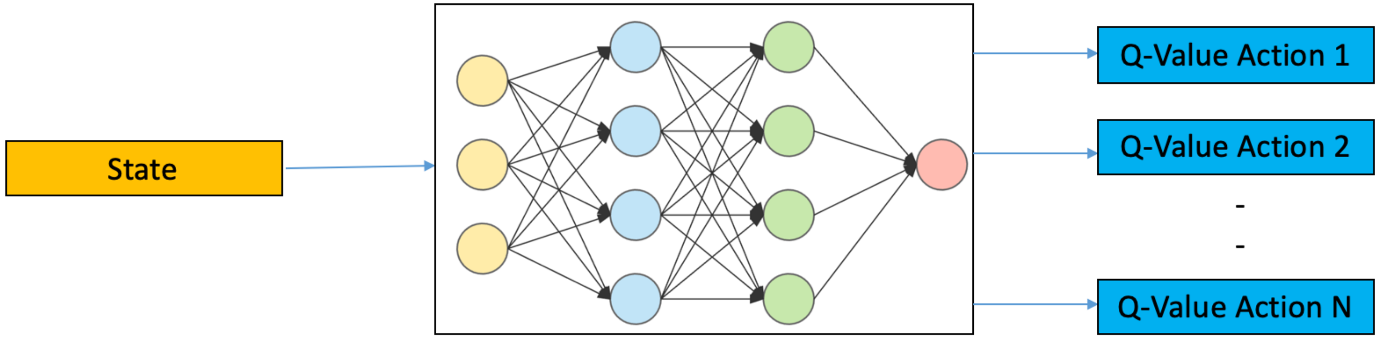
We need a well-designed function approximator that can compute the Q-values from the given inputs and its parameters, θ

Deep Q-Network



Q Table	
State-Action	Value
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0

Q Learning



Deep Q Learning

Next Lecture

Deep Reinforcement Learning

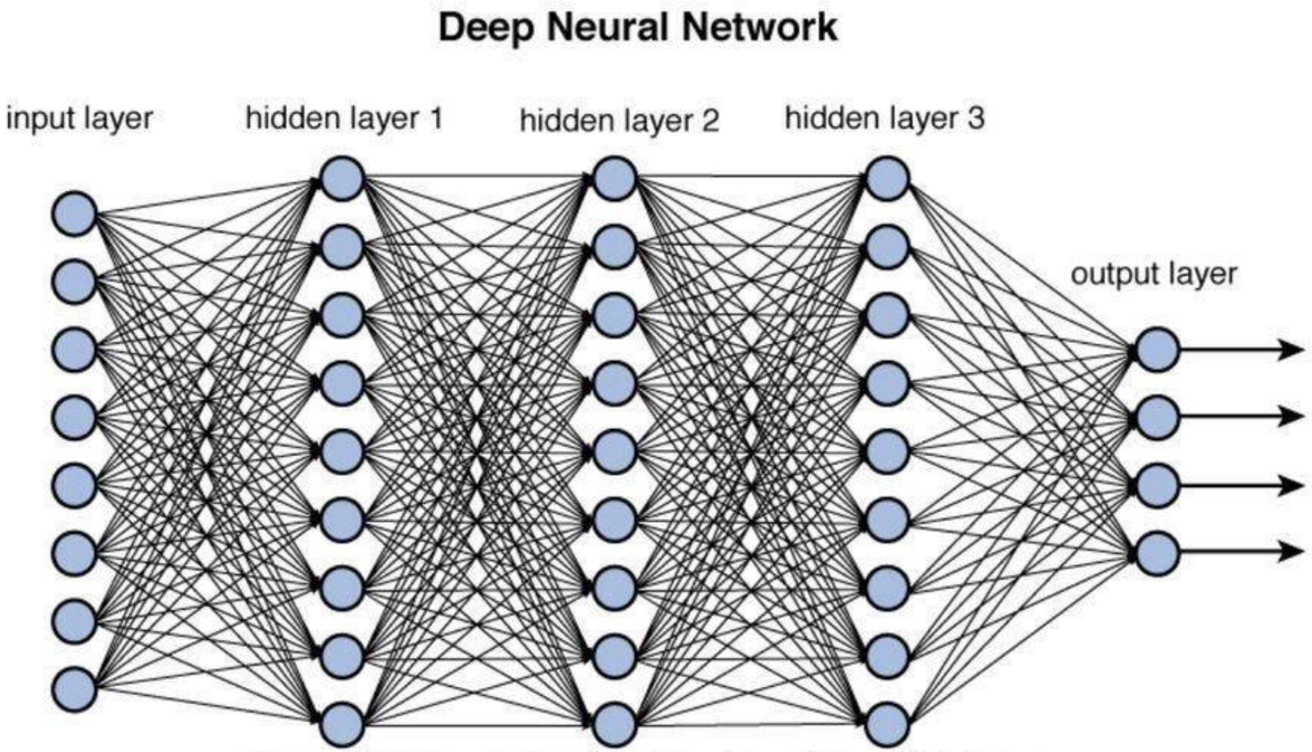


Figure 12.2 Deep network architecture with multiple layers.