

Reinforcement Learning

Data Science Specialization

Spring 2025

Jens Classen

Roskilde University

28.03./02.04.2025

1 Fundamentals

2 Q-Learning

3 Extensions and Variations

4 Summary

Fundamentals

Intelligent
Agents

Markov
Decision
Processes

The Bellman
Equations

Temporal-
Difference
Learning

Q-Learning

A Worked
Example

Extensions
and
Variations

SARSA

Deep Q
Networks

Summary

Fundamentals

Goal: Train agents to complete task that requires executing sequences of actions.

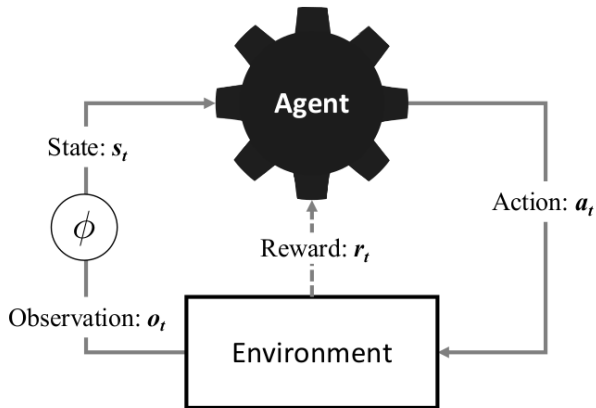


Figure 1: An agent behaving in an environment and the observation, reward, action cycle. The transition from observations of the environment to a state is shown by the state generation function, ϕ .

Fundamentals

Intelligent
AgentsMarkov
Decision
ProcessesThe Bellman
EquationsTemporal-
Difference
Learning

Q-Learning

A Worked
ExampleExtensions
and
VariationsSARSA
Deep Q
Networks

Summary

An **episode**:

$$(o_1, a_1, r_1), (o_2, a_2, r_2), (o_3, a_3, r_3), \dots, (o_e, a_e, r_e) \quad (1)$$

- o_t – observation at time t
- a_t – action executed at time t
- r_t – feedback (reward) obtained at time t

Instead of using observations directly, use a **state generation function** ϕ to turn them into states $s_t = \phi(o_t)$:

$$(s_1, a_1, r_1), (s_2, a_2, r_2), (s_3, a_3, r_3), \dots, (s_e, a_e, r_e) \quad (2)$$

In **fully observable** environments, we can identify observations with states!

Agent has goal to maximize **cumulative reward** or **return** of an episode:

$$G = r_t + r_{t+1} + r_{t+2} + r_{t+3} + \dots + r_e \quad (3)$$

Reward is often **delayed**!

👉 Design of a good reward function is an art.

Want to obtain a **policy** that lets agent decide its action.

Deterministic policy:

$$a_t = \pi(s_t) \quad (4)$$

Probabilistic policy:

$$P(A_t = a \mid S_t = s) = \pi(S_t = s) \quad (5)$$

Encoding Policies

- look-up table
- rule (set) to choose action (here)

Greedy action selection policy: choose action with highest immediate reward

☞ Not a good idea when rewards are delayed!

A **value function** returns the **expected cumulative reward** starting in state s_t following policy π :

$$V_{\pi}(s_t) = E_{\pi}[r_t + r_{t+1} + r_{t+2} + r_{t+3} + \dots + r_e \mid s_t] \quad (6)$$

An **action value function** returns the **expected cumulative reward** starting in state s_t **with action a_t** and afterwards following policy π :

$$Q_{\pi}(s_t, a_t) = E_{\pi}[r_t + r_{t+1} + r_{t+2} + r_{t+3} + \dots + r_e \mid s_t, a_t] \quad (7)$$

Typically, an immediate reward is valued more than future rewards (e.g., getting a gift of 100\$ now vs. in one year's time).

Often, a **discount factor** $\gamma \in [0, 1]$ is applied and one considers the **discounted return**.

$$G_\gamma = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots + \gamma^{e-t} r_e \quad (8)$$

Example: $\gamma = 0.1$:

$$G_{\gamma=0.1} = r_t + 0.1 \times r_{t+1} + 0.01 \times r_{t+2} + 0.001 \times r_{t+3} + \dots$$

Example: $\gamma = 0.9$:

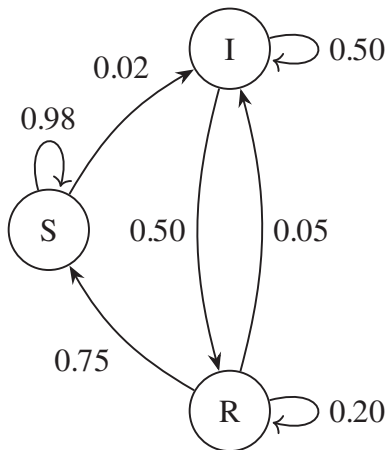
$$G_{\gamma=0.9} = r_t + 0.9 \times r_{t+1} + 0.81 \times r_{t+2} + 0.729 \times r_{t+3} + \dots$$

Discounted Return

Use discounted return in the action-value instead:

$$Q_{\pi}(s_t, a_t) = E_{\pi}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots + \gamma^{e-t} r_e \mid s_t, a_t] \quad (9)$$

Markov Process



(a) S-I-R Markov process

$$\mathcal{P} = \begin{matrix} & \begin{matrix} S & I & R \end{matrix} \\ \begin{matrix} S \\ I \\ R \end{matrix} & \begin{bmatrix} 0.98 & 0.02 & 0.00 \\ 0.00 & 0.50 & 0.50 \\ 0.75 & 0.05 & 0.20 \end{bmatrix} \end{matrix}$$

(b) S-I-R transition matrix

Figure 2: A simple Markov process to model the evolution of an infectious disease in individuals during an epidemic using the SUSCEPTIBLE-INFECTED-RECOVERED (S-I-R) model.

The Markov Assumption

Probability of transitioning to the next state does only depend on the current state!

(No knowledge of history before that required.)

$$P(S_{t+1} \mid S_t, S_{t-1}, S_{t-2}, \dots) = P(S_{t+1} \mid S_t) \quad (10)$$

Shorthand notation for **transitioning probability**:

$$P(s_1 \rightarrow s_2) = P(S_{t+1} = s_2 \mid S_t = s_1) \quad (11)$$

Can represent dynamics of Markov process by **transition matrix**:

$$\mathcal{P} = \begin{bmatrix} P(s_1 \rightarrow s_1) & P(s_1 \rightarrow s_2) & \dots & P(s_1 \rightarrow s_n) \\ P(s_2 \rightarrow s_1) & P(s_2 \rightarrow s_2) & \dots & P(s_2 \rightarrow s_n) \\ \vdots & \vdots & \ddots & \vdots \\ P(s_n \rightarrow s_1) & P(s_n \rightarrow s_2) & \dots & P(s_n \rightarrow s_n) \end{bmatrix}$$

Markov Decision Process

A **Markov decision process** (MDP) adds agent actions.

$$P(s_1 \xrightarrow{a} s_2) = P(S_{t+1} = s_2 \mid S_t = s_1, A_t = a) \quad (12)$$

$$R(s_1 \xrightarrow{a} s_2) = E(r_t \mid S_t = s_1, S_{t+1} = s_2, A_t = a) \quad (13)$$

Example: Game of TwentyTos

TwentyTos is a simplified form of “Black Jack”:

- cards are worth their number value, picture cards 10, ace 11
- goal: get higher value than opponent, but don't exceed 22 (*bust*)
- player and dealer are dealt two cards each
- player can see only one of dealer's cards
- player actions: *Twist* (get another card), until *Stick* (stop)
- afterwards:
 - dealer reveals second card
 - dealer deals themselves cards until they reach 17 or more, or go *bust*
- outcomes:
 - player wins \$2 if they have two aces
 - player wins \$1 if player's value doesn't exceed 22, and is higher than dealer's value
 - player wins \$1 if player's value doesn't exceed 22, and dealer goes bust
 - game is tied if both player and dealer have same value ≤ 22
 - otherwise, player loses \$1
- cards are dealt from an infinite deck

Example: Game of TwentyTws

Table 1: Some episodes of games played by the TwentyTws agent showing the cards dealt, as well as the states, actions, and rewards. Note that rewards are shown on the row indicating the action that led to them, not the state that followed that action.

Iter	Player Hand		Dealer Hand		State	Action	Reward
1	2♥ 7♣	(9)	8♥	(8)	PL-DH	<i>Twist</i>	0
2	2♥ 7♣ K♣	(19)	8♥	(8)	PH-DH	<i>Stick</i>	+1
3	2♥ 7♣ K♣	(19)	8♥ Q♦	(18)	WIN		
1	4♠ A♥	(15)	Q♥	(10)	PM-DH	<i>Twist</i>	-1
2	4♠ A♥ 9♣	(24)	Q♥	(10)	BUST		
1	2♦ 4♦	(6)	3♥	(3)	PL-DL	<i>Twist</i>	0
2	2♦ 4♦ 3♥	(9)	3♥	(3)	PL-DL	<i>Twist</i>	0
3	2♦ 4♦ 3♥ 6♣	(15)	3♥	(3)	PM-DL	<i>Twist</i>	0
4	2♦ 4♦ 3♥ 6♣ 6♦	(21)	3♥	(3)	PH-DL	<i>Stick</i>	0
5	2♦ 4♦ 3♥ 6♣ 6♦	(21)	3♥ 7♥ A♠	(21)	TIE		
1	Q♦ J♣	(20)	A♥	(11)	PH-DH	<i>Stick</i>	+1
2	Q♦ J♣	(20)	A♣ 5♣ Q♠	(26)	WIN		
1	A♦ A♥	(22)	2♥	(2)	PH-DL	<i>Stick</i>	+2
2	A♦ A♥	(22)	2♥	(2)	TWENTYTWO		

State Discretization

Discretize representation:

Player's Hand:

- Player Low (PL): 4–14
- Player Medium (PM): 15–18
- Player High (PH): 19–22

Dealer's Hand:

- Dealer Low (DL): 2–7
- Dealer High (DH): 8–11

Example MDP: TwentyTos

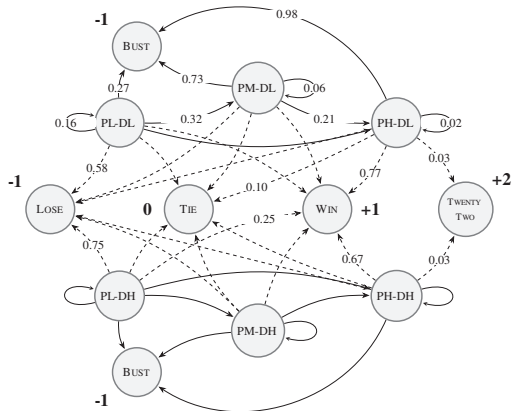


Figure 3: A Markov decision process representation for TwentyTos, a simplified version of the card game Blackjack.

Have one transition matrix for every action:

$$\mathcal{P}^{Twist} = \begin{bmatrix} P(\text{PL-DL} \xrightarrow{Twist} \text{PL-DL}) & P(\text{PL-DL} \xrightarrow{Twist} \text{PM-DL}) & \dots & P(\text{PL-DL} \xrightarrow{Twist} \text{TWENTYTWO}) \\ P(\text{PM-DL} \xrightarrow{Twist} \text{PL-DL}) & P(\text{PM-DL} \xrightarrow{Twist} \text{PM-DL}) & \dots & P(\text{PM-DL} \xrightarrow{Twist} \text{TWENTYTWO}) \\ \vdots & \vdots & \ddots & \vdots \\ P(\text{TWENTYTWO} \xrightarrow{Twist} \text{PL-DL}) & P(\text{TWENTYTWO} \xrightarrow{Twist} \text{PM-DL}) & \dots & P(\text{TWENTYTWO} \xrightarrow{Twist} \text{TWENTYTWO}) \end{bmatrix}$$

$$\mathcal{P}^{Stick} = \begin{bmatrix} P(\text{PL-DL} \xrightarrow{Stick} \text{PL-DL}) & P(\text{PL-DL} \xrightarrow{Stick} \text{PM-DL}) & \dots & P(\text{PL-DL} \xrightarrow{Stick} \text{TWENTYTWO}) \\ P(\text{PM-DL} \xrightarrow{Stick} \text{PL-DL}) & P(\text{PM-DL} \xrightarrow{Stick} \text{PM-DL}) & \dots & P(\text{PM-DL} \xrightarrow{Stick} \text{TWENTYTWO}) \\ \vdots & \vdots & \ddots & \vdots \\ P(\text{TWENTYTWO} \xrightarrow{Stick} \text{PL-DL}) & P(\text{TWENTYTWO} \xrightarrow{Stick} \text{PM-DL}) & \dots & P(\text{TWENTYTWO} \xrightarrow{Stick} \text{TWENTYTWO}) \end{bmatrix}$$

Transition Matrices

Fundamentals

Intelligent Agents

Markov Decision Processes

The Bellman Equations

Temporal-Difference Learning

Q-Learning

A Worked Example

Extensions and Variations

SARSA

Deep Q Networks

Summary

$$\mathcal{P}^{Twist} =$$

	PL-DL	PM-DL	PH-DL	PL-DH	PM-DH	PH-DH	BUST	LOSE	TIE	WIN	TWENTYTWO
PL-DL	0.16	0.32	0.34	0.00	0.00	0.00	0.17	0.00	0.00	0.00	0.00
PM-DL	0.00	0.06	0.29	0.00	0.00	0.00	0.65	0.00	0.00	0.00	0.00
PH-DL	0.00	0.00	0.08	0.00	0.00	0.00	0.92	0.00	0.00	0.00	0.00
PL-DH	0.00	0.00	0.00	0.16	0.32	0.34	0.17	0.00	0.00	0.00	0.00
PM-DH	0.00	0.00	0.00	0.00	0.06	0.29	0.65	0.00	0.00	0.00	0.00
PH-DH	0.00	0.00	0.00	0.00	0.00	0.08	0.92	0.00	0.00	0.00	0.00
BUST	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
LOSE	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TIE	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
WIN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TWENTYTWO	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Transition Matrices

$$\mathcal{P}^{Stick} =$$

	PL-DL	PM-DL	PH-DL	PL-DH	PM-DH	PH-DH	BUST	LOSE	TIE	WIN	TWENTYTWO
PL-DL	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.68	0.00	0.32	0.00
PM-DL	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.58	0.06	0.36	0.00
PH-DL	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.10	0.68	0.03
PL-DH	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.81	0.00	0.19	0.00
PM-DH	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.73	0.06	0.21	0.00
PH-DH	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.22	0.15	0.60	0.03
BUST	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
LOSE	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TIE	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
WIN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TWENTYTWO	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Bellman Equations

Can restate the action-value function for MDPs with **infinite horizon**:

$$\begin{aligned} Q_{\pi}(s_t, a_t) &= E_{\pi} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots \mid s_t, a_t \right] \\ &= E_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t, a_t \right] \end{aligned} \quad (14)$$

$$= E_{\pi} \left[r_t + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t, a_t \right] \quad (15)$$

Bellman Equations

Reformulate to state **state transition** and **reward**:

$$Q_{\pi}(s_t, a_t) = \sum_{s_{t+1}} P(s_t \xrightarrow{a_t} s_{t+1}) \left[R(s_t \xrightarrow{a_t} s_{t+1}) + \gamma E_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_{t+1} \right] \right] \quad (16)$$

Reformulate to sum over expected returns of actions a_{t+1} :

$$Q_{\pi}(s_t, a_t) = \sum_{s_{t+1}} P(s_t \xrightarrow{a_t} s_{t+1}) \left[R(s_t \xrightarrow{a_t} s_{t+1}) + \gamma \sum_{a_{t+1}} E_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_{t+1}, a_{t+1} \right] \right] \quad (17)$$

Bellman Equation

$$Q_{\pi}(s_t, a_t) = \sum_{s_{t+1}} P(s_t \xrightarrow{a_t} s_{t+1}) \left[R(s_t \xrightarrow{a_t} s_{t+1}) + \gamma \sum_{a_{t+1}} \pi(s_{t+1}, a_{t+1}) Q_{\pi}(s_{t+1}, a_{t+1}) \right] \quad (18)$$

Bellman Optimality Equation (following *optimal* policy π_*)

$$Q_*(s_t, a_t) = \sum_{s_{t+1}} P(s_t \xrightarrow{a_t} s_{t+1}) \left[R(s_t \xrightarrow{a_t} s_{t+1}) + \gamma \max_{a_{t+1}} Q_*(s_{t+1}, a_{t+1}) \right] \quad (19)$$

Temporal-Difference Learning

- 👉 Solving Bellman equations **exactly** (e.g., using dynamic programming) is **computationally very expensive** and hence only possible for very small instances.
- 👉 Instead, we typically **approximate** a solution by an **iterative approach**.

Temporal Difference Learning

- store $Q_{\pi}(s_t, a_t)$ values in a table
- initialize values randomly (or: all zeros)
- update corresponding value after taking an action and observing reward

Example: Action-Value Table

Table 2: An action-value table for an agent trained to play the card game TwentyTws.

State	Action	Value	State	Action	Value	State	Action	Value
PL-DL	Twist	0.039	PH-DL	Twist	−0.666	PM-DH	Twist	−0.668
PL-DL	Stick	−0.623	PH-DL	Stick	0.940	PM-DH	Stick	−0.852
PM-DL	Twist	−0.597	PL-DH	Twist	−0.159	PH-DH	Twist	−0.883
PM-DL	Stick	−0.574	PL-DH	Stick	−0.379	PH-DH	Stick	0.391
BUST	Twist	0.000	TIE	Twist	0.000	WIN	Twist	0.000
BUST	Stick	0.000	TIE	Stick	0.000	WIN	Stick	0.000
LOSE	Twist	0.000				TWENTYTWO	Twist	0.000
LOSE	Stick	0.000				TWENTYTWO	Stick	0.000

Temporal-Difference Learning

Update rule:

$$Q_{\pi}(s_t, a_t) \leftarrow Q_{\pi}(s_t, a_t) + \alpha \underbrace{(G(s_t, a_t) - Q_{\pi}(s_t, a_t))}_{\text{difference between actual and expected returns}} \quad (20)$$

where:

- α is a **learning rate**
- $G(s_t, a_t)$ is the actual return after taking action a_t until end of episode

👉 Values will (slowly) converge to true values!

👉 Previous update rule can only be applied after completing an entire episode!

Bootstrapping: uses existing estimates of expected returns in the action-value table:

$$Q_{\pi}(s_t, a_t) \leftarrow Q_{\pi}(s_t, a_t) + \alpha \underbrace{(r_t + \gamma Q_{\pi}(s_{t+1}, a_{t+1}))}_{\text{actual return}} - \underbrace{Q_{\pi}(s_t, a_t)}_{\text{expected return}} \quad (21)$$

👉 Agent learns more quickly, but approach takes longer to converge!

Exploration and Exploitation

If we always take the currently best action, we might never find actions that lead to good outcomes at a later stage.

👉 We need to balance **exploration** and **exploitation**.

ϵ -greedy action selection policy

- With probability ϵ , perform a **random** action.
- With probability $1 - \epsilon$, perform the **best** current action.

Often one has a **behaviour policy** to be used during learning, and a **target policy** that is used after deployment.

Model-Free Learning

Note: Temporal difference learning is **model-free**: It works without knowing anything about the dynamics of the environment (transition probabilities etc.).

Q-Learning

We use a slight variant of the previous update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right) \quad (22)$$

Intuition

Q -value for s_t and a_t should increase if a_t

- gives immediate positive reward, OR
- leads to a state where we expect high future return

Pseudocode description of the Q-learning algorithm for off-policy temporal-difference learning.

Require: a behavior policy, π , that chooses actions

Require: an action-value function Q that performs a lookup into an action-value table with entries for every possible action, a , and state, s

Require: a learning rate, α , a discount-rate, γ , and a number of episodes to perform

- 1: initialize all entries in the action-value table to random values (except for terminal states which receive a value of 0)
- 2: **for** each episode **do**
- 3: reset s_t to the initial agent state
- 4: **repeat**
- 5: select an action, a_t , based on policy, π , current state, s_t , and action-value function, Q
- 6: take action a_t observing reward, r_t , and new state s_{t+1}
- 7: update the record in the action-value table for the action, a_t , just taken in the last state, s_t , using:

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

- 8: let $s_t = s_{t+1}$
- 9: **until** agent reaches a terminal state
- 10: **end for**

Off-Policy Learning

Q-learning uses **off-policy** learning: It learns an optimal (target) policy, but uses a different (behaviour) policy to choose actions during learning.

Popular Behaviour Policies

- ϵ -greedy policy (see previous slide)
- **Boltzmann** action selection: build **softmax** distribution over actions

$$\pi(s_t, a_t) = \frac{e^{Q(s_t, a_t)}}{\sum_a e^{Q(s_t, a)}}$$

Result after 100,000 episodes of Q-learning for TwentyTwos game:

State	Action	Value	State	Action	Value	State	Action	Value
PL-DL	<i>Twist</i>	0.039	PH-DL	<i>Twist</i>	-0.666	PM-DH	<i>Twist</i>	-0.668
PL-DL	<i>Stick</i>	-0.623	PH-DL	<i>Stick</i>	0.940	PM-DH	<i>Stick</i>	-0.852
PM-DL	<i>Twist</i>	-0.597	PL-DH	<i>Twist</i>	-0.159	PH-DH	<i>Twist</i>	-0.883
PM-DL	<i>Stick</i>	-0.574	PL-DH	<i>Stick</i>	-0.379	PH-DH	<i>Stick</i>	0.391
BUST	<i>Twist</i>	0.000	TIE	<i>Twist</i>	0.000	WIN	<i>Twist</i>	0.000
BUST	<i>Stick</i>	0.000	TIE	<i>Stick</i>	0.000	WIN	<i>Stick</i>	0.000
LOSE	<i>Twist</i>	0.000				TWENTYTWO	<i>Twist</i>	0.000
LOSE	<i>Stick</i>	0.000				TWENTYTWO	<i>Stick</i>	0.000

With this policy, player will earn $\$198.24 \pm 24$ on average out of playing 1000 hands.

A random player *loses* $\$197 \pm 25$ on average!

Grid World Example

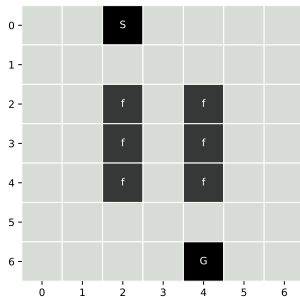


Figure 4: A simple grid world. The start position is annotated with an *S* and the goal with a *G*. The squares marked *f* denote fire, which is very damaging to an agent.

Rewards: -1 per move, +50 for reaching *G*, -20 for stepping on *f*.

Parameters: $\epsilon = 0.1$, $\alpha = 0.2$, $\gamma = 0.9$ (recommended values)

Grid World Example

Table 3: A portion of the action-value table for the grid world example at its first initialization.

State	Action	Value	State	Action	Value	State	Action	Value
0-0	<i>up</i>	0.933		...		6-2	...	
0-0	<i>down</i>	-0.119	2-0	<i>left</i>	-0.691	6-2	<i>right</i>	0.201
0-0	<i>left</i>	-0.985	2-0	<i>right</i>	0.668	6-3	<i>up</i>	-0.588
0-0	<i>right</i>	0.822	2-1	<i>up</i>	-0.918	6-3	<i>down</i>	0.038
0-1	<i>up</i>	0.879	2-1	<i>down</i>	-0.228	6-3	<i>left</i>	0.859
0-1	<i>down</i>	0.164	2-1	<i>left</i>	-0.301	6-3	<i>right</i>	-0.085
0-1	<i>left</i>	0.343	2-1	<i>right</i>	-0.317	6-4	<i>up</i>	0.000
0-1	<i>right</i>	-0.832	2-2	<i>up</i>	0.633	6-4	<i>down</i>	0.000
0-2	<i>up</i>	0.223	2-2	<i>down</i>	-0.048	6-4	<i>left</i>	0.000
0-2	<i>down</i>	0.582	2-2	<i>left</i>	0.566	6-4	<i>right</i>	0.000
0-2	<i>left</i>	0.672	2-2	<i>right</i>	-0.058	6-5	<i>up</i>	0.321
0-2	<i>right</i>	0.084	2-3	<i>up</i>	0.635	6-5	<i>down</i>	-0.793
0-3	<i>up</i>	-0.308	2-3	<i>down</i>	0.763	6-5	<i>left</i>	-0.267
0-3	<i>down</i>	0.247	2-3	<i>left</i>	-0.121	6-5	<i>right</i>	0.588
0-3	<i>left</i>	0.963	2-3	<i>right</i>	0.562	6-6	<i>up</i>	-0.870
0-3	<i>right</i>	0.455	2-4	<i>up</i>	0.629	6-6	<i>down</i>	-0.720
0-4	<i>up</i>	-0.634	2-4	<i>down</i>	-0.409	6-6	<i>left</i>	0.811
		6-6	<i>right</i>	0.176

Grid World Example

Fundamentals

Intelligent
AgentsMarkov
Decision
ProcessesThe Bellman
EquationsTemporal-
Difference
Learning

Q-Learning

A Worked
ExampleExtensions
and
Variations

SARSA

Deep Q
Networks

Summary

$$\begin{aligned} Q(0-3, left) &\leftarrow Q(0-3, left) + \alpha \times (R(0-3, left) + \gamma \times Q(0-2, left) - Q(0-3, left)) \\ &0.963 + 0.2 \times (-1 + 0.9 \times 0.672 - 0.963) \\ &0.691 \end{aligned}$$

Grid World Example

Fundamentals

Intelligent
AgentsMarkov
Decision
ProcessesThe Bellman
EquationsTemporal-
Difference
Learning

Q-Learning

A Worked
ExampleExtensions
and
Variations

SARSA

Deep Q
Networks

Summary

$$\begin{aligned} Q(6-5, left) &\leftarrow Q(6-5, left) + \alpha \times (R(6-5, left) + \gamma \times Q(6-4, up) - Q(6-5, left)) \\ &\quad - 0.267 + 0.2 \times (50 + 0.9 \times 0 - -0.267) \\ &\quad 9.786 \end{aligned}$$

Grid World Example

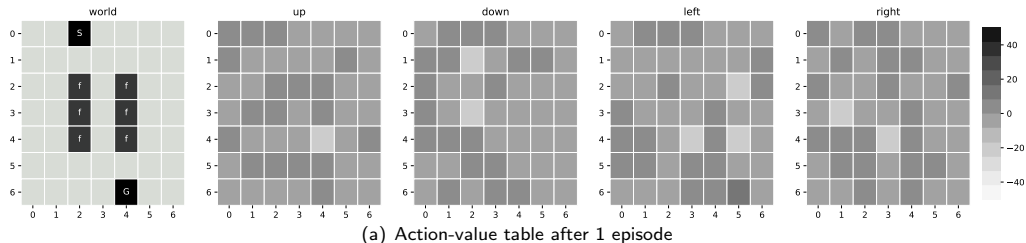


Figure 5: (a)–(c) The evolution of the entries in the action-value table over episodes of Q-learning off-policy temporal-difference learning across the grid world. (d) The cumulative reward earned from each episode. (e) An illustration of the target policy learned by the agent after 350 episodes. The arrows show the direction with the highest entry in the action-value table for each state. (f) The path the agent will take from the start state to the goal state when greedily following the target policy.

Grid World Example

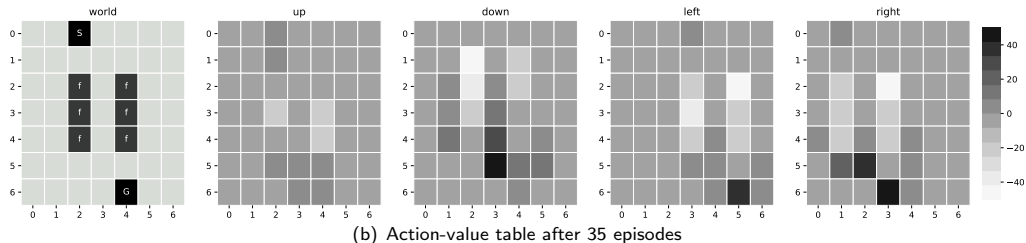


Figure 6: (a)–(c) The evolution of the entries in the action-value table over episodes of Q-learning off-policy temporal-difference learning across the grid world. (d) The cumulative reward earned from each episode. (e) An illustration of the target policy learned by the agent after 350 episodes. The arrows show the direction with the highest entry in the action-value table for each state. (f) The path the agent will take from the start state to the goal state when greedily following the target policy.

Grid World Example

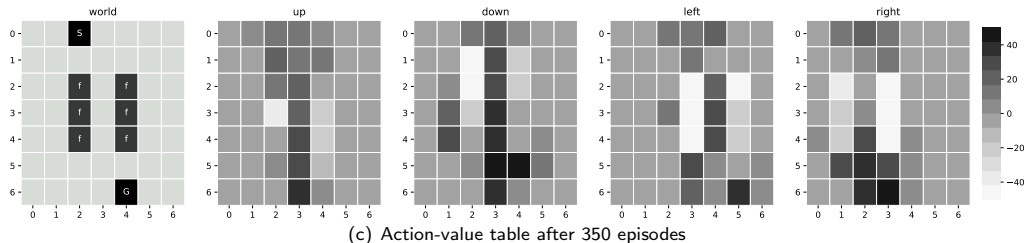
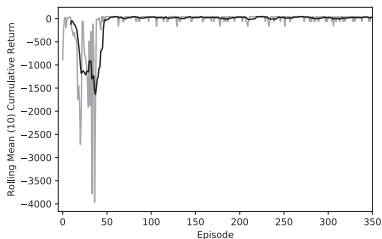
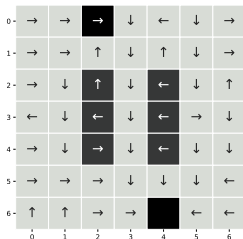


Figure 7: (a)–(c) The evolution of the entries in the action-value table over episodes of Q-learning off-policy temporal-difference learning across the grid world. (d) The cumulative reward earned from each episode. (e) An illustration of the target policy learned by the agent after 350 episodes. The arrows show the direction with the highest entry in the action-value table for each state. (f) The path the agent will take from the start state to the goal state when greedily following the target policy.

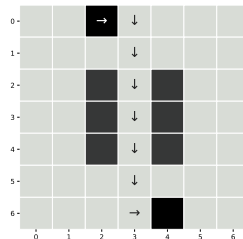
Grid World Example



(d) Cumulative Reward



(e) Policy



(f) Offline Path

Figure 8: (a)–(c) The evolution of the entries in the action-value table over episodes of Q-learning off-policy temporal-difference learning across the grid world. (d) The cumulative reward earned from each episode. (e) An illustration of the target policy learned by the agent after 350 episodes. The arrows show the direction with the highest entry in the action-value table for each state. (f) The path the agent will take from the start state to the goal state when greedily following the target policy.

Grid World Example

Table 4: A portion of the action-value table for the grid world example after 350 episodes of Q-learning have elapsed.

State	Action	Value	State	Action	Value	State	Action	Value
0-0	<i>up</i>	-1.627		
0-0	<i>down</i>	-1.255	2-0	<i>left</i>	-1.583	6-2	<i>right</i>	40.190
0-0	<i>left</i>	-1.655	2-0	<i>right</i>	-1.217	6-3	<i>up</i>	34.375
0-0	<i>right</i>	-1.000	2-1	<i>up</i>	-1.493	6-3	<i>down</i>	40.206
0-1	<i>up</i>	1.302	2-1	<i>down</i>	4.132	6-3	<i>left</i>	24.784
0-1	<i>down</i>	-1.900	2-1	<i>left</i>	-1.643	6-3	<i>right</i>	50.000
0-1	<i>left</i>	-1.900	2-1	<i>right</i>	-36.301	6-4	<i>up</i>	0.000
0-1	<i>right</i>	15.173	2-2	<i>up</i>	13.247	6-4	<i>down</i>	0.000
0-2	<i>up</i>	13.299	2-2	<i>down</i>	-46.862	6-4	<i>left</i>	0.000
0-2	<i>down</i>	12.009	2-2	<i>left</i>	-0.858	6-4	<i>right</i>	0.000
0-2	<i>left</i>	8.858	2-2	<i>right</i>	-1.157	6-5	<i>up</i>	-0.353
0-2	<i>right</i>	18.698	2-3	<i>up</i>	16.973	6-5	<i>down</i>	-0.793
0-3	<i>up</i>	13.921	2-3	<i>down</i>	29.366	6-5	<i>left</i>	36.823
0-3	<i>down</i>	21.886	2-3	<i>left</i>	-88.492	6-5	<i>right</i>	-0.342
0-3	<i>left</i>	15.900	2-3	<i>right</i>	-77.447	6-6	<i>up</i>	-0.870
0-3	<i>right</i>	13.846	2-4	<i>up</i>	-1.016	6-6	<i>down</i>	-0.720
0-4	<i>up</i>	1.637	2-4	<i>down</i>	-20.255	6-6	<i>left</i>	1.008
		6-6	<i>right</i>	-0.802

Extensions and Variations

Fundamentals

Intelligent
AgentsMarkov
Decision
ProcessesThe Bellman
EquationsTemporal-
Difference
Learning

Q-Learning

A Worked
ExampleExtensions
and
Variations

SARSA

Deep Q
Networks

Summary

Pseudocode description of the **SARSA** algorithm for on-policy temporal-difference learning.

Require: a behavior policy, π , that chooses actions

Require: an action-value function Q that performs a lookup into an action-value table with entries for every possible action, a , and state, s

Require: a learning rate, α , a discount-rate, γ , and a number of episodes to perform

1: initialize all entries in the action-value table to random values (except for terminal states which receive a value of 0)

2: **for** each episode **do**

3: reset s_t to the initial agent state

4: select an action, a_t , based on policy, π , current state, s_t , and action-value function, Q

5: **repeat**

6: take action a_t observing reward, r_t , and new state, s_{t+1}

7: select the next action, a_{t+1} , based on policy, π , new state, s_{t+1} , and action-value function, Q

8: update the record in the action-value table for the action, a_t , just taken in the last state, s_t , using:

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

9: let $s_t = s_{t+1}$ and $a_t = a_{t+1}$

10: **until** agent reaches terminal state

11: **end for**

Fundamentals

Intelligent
AgentsMarkov
Decision
ProcessesThe Bellman
EquationsTemporal-
Difference
Learning

Q-Learning

A Worked
ExampleExtensions
and
Variations

SARSA

Deep Q
Networks

Summary

Difference: SARSA uses **on-policy** temporal difference learning.

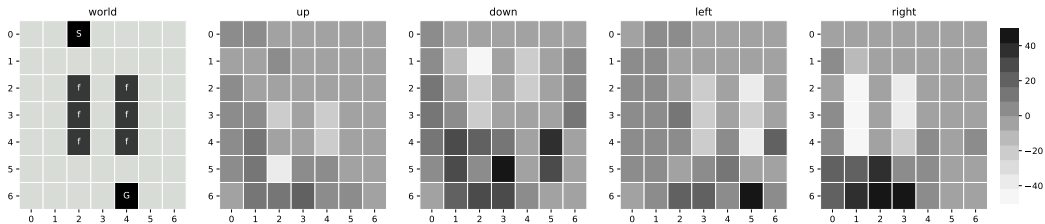
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha (r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

(Notice that here we use action a_{t+1} as chosen by policy π , and *not* the (currently) optimal one.)

Grid World Example

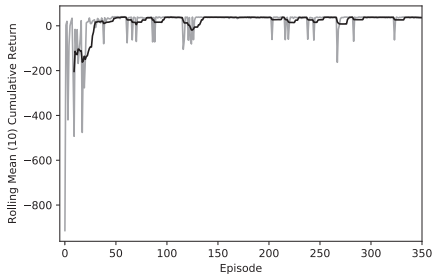
$$\begin{aligned} Q(0-3, left) &\leftarrow Q(0-3, left) + \alpha \times (R(0-3, left) + \gamma \times Q(0-2, down) - Q(0-3, left)) \\ &0.963 + 0.2 \times (-1 + 0.9 \times 0.582 - 0.963) \\ &0.675 \end{aligned}$$

Grid World Example

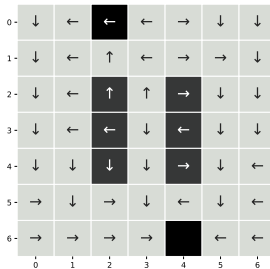


(a) Action-value table after 350 episodes

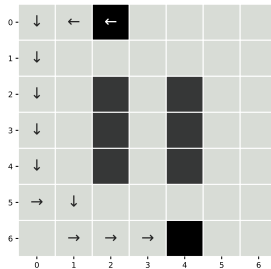
Grid World Example



(c) Cumulative Reward



(d) Policy



(e) Offline Path

Fundamentals

Intelligent
Agents

Markov
Decision
Processes

The Bellman
Equations

Temporal-
Difference
Learning

Q-Learning

A Worked
Example

Extensions and Variations

SARSA

Deep Q
Networks

Summary

Agents trained using SARSA tend to learn more **conservative** strategies.

Reason: Estimation of expected return includes poor (e.g., random) actions.

For many applications, a **tabular** representation of Q values is infeasible!

Reason: **too many states!**

Example: Lunar Lander Game

- Task: land spacecraft on lunar surface.
- Actions: thrusters for left, right, and up direction
- States: e.g. represented by
 - x, y position
 - x, y velocity
 - angular velocity
 - altitude
 - slope of landing pad

Even if we discretize state characteristics to 5 levels (very low, low, medium, high, very high), we get $5^5 = 3125$ states times 4 actions, i.e. 12,500 entries!

Also: unlikely to visit all states during training!

Example: Lunar Lander

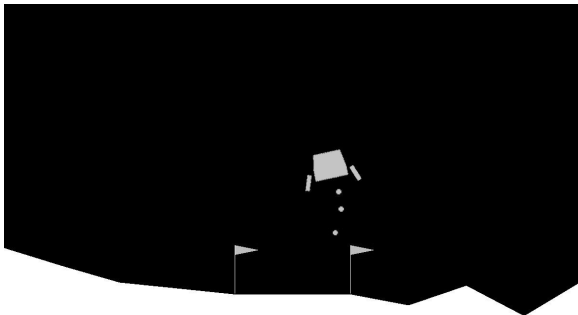


Figure 9: The Lunar Lander environment. The aim of the game is to control the spaceship starting from the top of the world and attempting to land on the landing pad.

State-Action Value Approximation

Idea: **approximate method** that **generalizes** from the observed state-action pairs.

We can use **predictive models** for that:

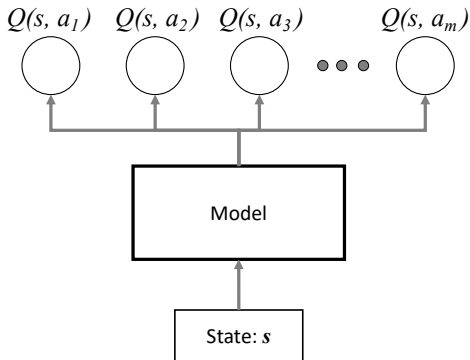


Figure 10: Framing the action-value function as a prediction problem.

State-Action Value Approximation

Approximate Q function by predictive model:

$$\mathbb{M}(s_t, a_t) \approx Q_\pi(s_t, a_t) \quad (23)$$

For **neural networks**, we can output the values for all actions at the same time in the output layer:

$$\mathbb{M}(s_t) \approx Q_\pi(s_t, a_t) \quad (24)$$

There is a strong link between temporal difference learning and gradient descent:

- Temporal difference learning is based on **error** between predicted and actual return.
- Neural network learning with gradient descent is based on **loss function**.

Loss Function:

$$\mathcal{L}(Q_{\mathbb{M}\mathbf{W}}(s_t)) = (t_i - Q_{\mathbb{M}\mathbf{W}}(s_t, a_t))^2 \quad (25)$$

$$= \left(r_t + \gamma \max_{a_{t+1}} Q_{\mathbb{M}\mathbf{W}}(s_{t+1}, a_{t+1}) - Q_{\mathbb{M}\mathbf{W}}(s_t, a_t) \right)^2 \quad (26)$$

\mathbf{W} : network weights t_i : actual return (target feature value)

Gradient:

$$\frac{\partial \mathcal{L}(Q_{\mathbb{M}\mathbf{W}}(s_t, a_t))}{\partial \mathbf{W}} = \left(r_t + \gamma \max_{a_{t+1}} Q_{\mathbb{M}\mathbf{W}}(s_{t+1}, a_{t+1}) - Q_{\mathbb{M}\mathbf{W}}(s_t, a_t) \right) \frac{\partial Q_{\mathbb{M}\mathbf{W}}(s_t, a_t)}{\partial \mathbf{W}} \quad (27)$$

Pseudocode description of the **naive neural Q-learning** algorithm.

- 1: initialize weights, \mathbf{W} , in action-value function network, $Q_{\mathbb{M}}$, to random values
- 2: **for** each episode **do**
- 3: reset s_t to the initial agent state
- 4: **repeat**
- 5: select action, a_t , based on policy, π , the current state, s_t , and action-value network output, $Q_{\mathbb{M}}(s_t, a_t)$
- 6: take action a_t and observing reward, r_t , and new state, s_{t+1}
- 7: generate a target feature
$$t = r_t + \gamma \max_{a_{t+1}} Q_{\mathbb{M}}(s_{t+1}, a_{t+1})$$
- 8: perform an iteration of stochastic gradient descent using a single training instance $\langle s_t, t \rangle$
- 9: **until** agent reaches terminal state
- 10: **end for**

Naive Neural Q-Learning: Problems

Problems with Naive Neural Q-Learning

- Gradient Descent only work when training instances are independent (shuffled), but the states and actions visited during an episode depend on one another!
 - 👉 Approach becomes stuck in local minimum.
- Same network is used for training and generating targets for training!
 - 👉 Rapid changes may cause the network to oscillate.

Deep Q Network Learning

The **deep Q network** (DQN) algorithm addresses these issues by using **experience replay** and **network freezing**.

Experience Replay

After each action:


Add the observed $\langle s_t, a_t, r_t, s_{t+1} \rangle$ tuple in the **replay memory**.

Select random sample of b instances from the replay memory and perform an iteration of **mini-batch gradient descent**, where target features are created as in naive neural Q-learning.

Use two different networks:

The **action-value behaviour network** is used to predict the values of actions and making decisions.

The **action-value target network** is used to predict the values of subsequent actions in subsequent states when generating target features. It is **frozen** (remains unchanged) for a number of C steps, and then replaced by a copy of the action-value behaviour network.

 more stable, faster conversion

DQN Algorithm

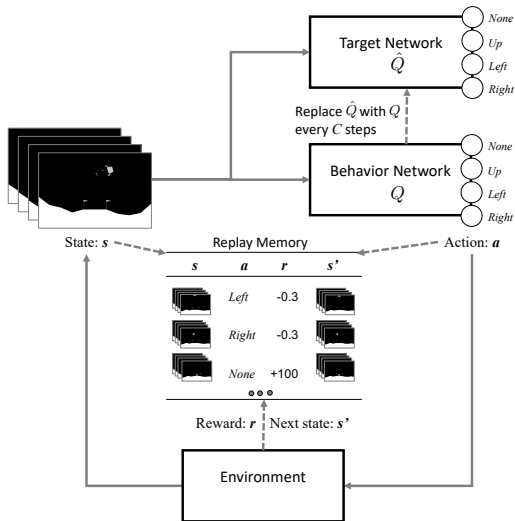


Figure 11: An illustration of the DQN algorithm including experience replay and target network freezing.

Pseudocode description of the **deep Q network** (DQN) algorithm.

- 1: initialize replay memory \mathcal{D} with N steps based on random actions
- 2: initialize weights, \mathbf{W} in behavior action-value function network, $Q_{\mathbf{M}}$, to random values
- 3: initialize weights, $\widehat{\mathbf{W}}$ in target action-value function network, $\widehat{Q}_{\mathbf{M}}$ to \mathbf{W}
- 4: **for** each episode **do**
- 5: reset s_t to the initial agent state
- 6: **repeat**
- 7: select action, a_t , based on agent's policy, π , the current state, s_t , and behavior network output, $Q_{\mathbf{M}}(s_t, a_t)$
- 8: take action a_t and observe the resulting reward, r_t , and new state, s_{t+1}
- 9: add tuple $\langle s = s_t, a = a_t, r = r_t, s' = s_{t+1} \rangle$ as a new instance in \mathcal{D}
- 10: randomly select a mini-batch of b instances from \mathcal{D} to give \mathcal{D}_b
- 11: generate target feature values for each instance, $\langle s_i, a_i, r_i, s'_i \rangle$ in \mathcal{D}_b as:

$$t_i = r_i + \gamma \max_{a'} \widehat{Q}_{\mathbf{M}}(s'_i, a')$$

- 12: perform an iteration of mini-batch gradient descent using \mathcal{D}_b
- 13: every C steps let $\widehat{Q}_{\mathbf{M}} = Q_{\mathbf{M}}$
- 14: **until** agent reaches terminal state
- 15: **end for**

Deep Q Network Learning

In principle, any network architecture could be used with DQN.

The algorithm was first proposed for learning to play video games with **screenshots** from the game as input. Images were handled using **convolutional neural networks**.

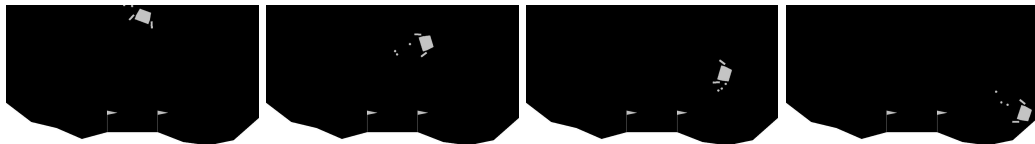
Note that a single, static screenshot does not provide the full state of the game. The environment is hence not **fully observable**, and the **Markov assumption** does not hold.

As a solution, a short “stack” of the last k screenshots is used to determine the state (e.g., $k = 4$). This is an example of the **state generation function** from earlier.

Example: Lunar Lander

- 4 actions: *None, Up, Left, Right*
- rewards: +100 for landing, -100 for crashing, +10 for each leg touching ground gently, -0.3 per thruster action
- convolutional neural network as action-value network
 - images scaled to 84x84 resolution
 - convolutional layers with 32, 64, and 64 units
 - filter sizes 8x8 (stride 4), 4x4 (stride 3), 3x3 (stride 1)
 - rectified linear activation functions in all hidden units
 - final hidden layer to flatten outputs of previous layer (512 fully connected units with rectified linear activations)
 - output layer fully connected with 4 outputs (one per action) using linear activations
- ϵ greedy behaviour policy with **linear annealing** (changing ϵ over time from 0.9 to 0.05)
- replay memory size: 50,000
- action-value function network replaced every 10,000 steps

Lunar Lander Example



(a) Poor performance in the Lunar Lander environment early in the learning process.

Figure 12: (a) Frames from an episode early in the training process in which the agent performs poorly. (b) Frames from an episode near the end of the learning process where the agent is starting to be very effective. (c) Changing episode returns during DQN training. The gray line shows a 50-episode moving average to better highlight the trend.

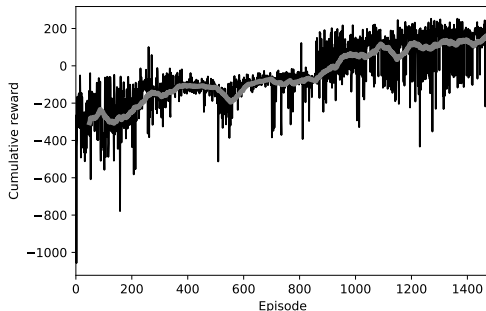
Lunar Lander Example



(b) Good performance in the Lunar Lander environment after 30,000 learning steps.

Figure 13: (a) Frames from an episode early in the training process in which the agent performs poorly. (b) Frames from an episode near the end of the learning process where the agent is starting to be very effective. (c) Changing episode returns during DQN training. The gray line shows a 50-episode moving average to better highlight the trend.

Lunar Lander Example



(c) Changing return during training.

Figure 14: (a) Frames from an episode early in the training process in which the agent performs poorly. (b) Frames from an episode near the end of the learning process where the agent is starting to be very effective. (c) Changing episode returns during DQN training. The gray line shows a 50-episode moving average to better highlight the trend.

Summary

- In a reinforcement learning scenario an **agent** inhabiting an **environment** attempts to achieve a **goal** by taking a sequence of **actions** to move it between **states**.
- On completion of each action the agent receives an immediate scalar **reward** indicating whether the outcome of the action was positive or negative and to what degree.
- To choose which action to take in a given state the agent uses a **policy**.
- Policies rely on being able to assess the **expected return** of taking an action in a particular state, and an **action-value function** is used to calculate this.

Fundamentals

Intelligent
Agents

Markov
Decision
Processes

The Bellman
Equations

Temporal-
Difference
Learning

Q-Learning

A Worked
Example

Extensions and Variations


SARSA

Deep Q
Networks

Summary

- The learning approaches described here are **value-based** and **model-free**.
- **Temporal-difference learning**, and its **Q-learning** (off-policy) and **SARSA** (on-policy) variants, is an important tabular methods for reinforcement learning.
- **Deep Q networks** are an approximate approach to temporal difference learning based on deep neural networks.
- One overarching point about reinforcement learning that is worth mentioning is that it comes at the cost of hugely increased computation.

Reading

- Mitchell, T. M. (1997). *Machine Learning* (Vol. 1). McGraw-Hill New York. Chapter 13.
- Russell S. J. & Norvig P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson. Chapter 22.
- Kelleher, Mac Namee, B., & D'Arcy, A. (2020). *Fundamentals of Machine Learning for Predictive Data Analytics Algorithms, Worked Examples, and Case Studies*. MIT Press. Chapter 11.
- Sutton, R. S. & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.  <http://incompleteideas.net/book/the-book-2nd.html>