COMP3702 Tutorial 10

Matt Choy | matthew.choy@uq.edu.au

Multi-Armed Bandits

- If we don’t know the system dynamics, should we take exploratory actions (that give us more information about the environment) or exploit current knowledge to perform as best as we can?
- Using a greedy policy (exploiting current knowledge), bad initial estimates in the first few cases can drive policy into a sub-optimal region, and never explore further
- Instead of acting according to a greedy policy, we act according to a sampling strategy that will explore (state, action) pairs until we get a “good” estimate of the value function

Multi-Armed Bandits

- If we don’t know the system dynamics, should we take exploratory actions (that give us more information about the environment) or exploit current knowledge to perform as best as we can?
- Using a greedy policy (exploiting current knowledge), bad initial estimates in the first few cases can drive policy into a sub-optimal region, and never explore further
- Instead of acting according to a greedy policy, we act according to a sampling strategy that will explore (state, action) pairs until we get a “good” estimate of the value function

- For Multi-Armed Bandits to work, few key assumptions must hold:
  - We have a choice of several arms (options)
  - Each arm pull (action) is independent of other arm pulls (actions)
  - Each arm has a fixed, yet unknown average payoff

Multi-Armed Bandits

- If we don’t know the system dynamics, should we take exploratory actions (that give us more information about the environment) or exploit current knowledge to perform as best as we can?
- Using a greedy policy (exploiting current knowledge), bad initial estimates in the first few cases can drive policy into a sub-optimal region, and never explore further
- Instead of acting according to a greedy policy, we act according to a sampling strategy that will explore (state, action) pairs until we get a “good” estimate of the value function

- For Multi-Armed Bandits to work, few key assumptions must hold:
  - We have a choice of several arms (options)
  - Each arm pull (action) is independent of other arm pulls (actions)
  - Each arm has a fixed, yet unknown average payoff
- How do we know which arm has the best average payoff?
- How do we maximise the sum of rewards over time?

ϵ-Greedy Strategy

- Choose a random action with probability ϵ, and choose the best action with probability $1 - \epsilon$

Exercise 9.1a - ϵ-Greedy Strategy

- Choose a random action with probability ϵ, and choose the best action with probability $1 - \epsilon$

Exercise 10.1. Consider a situation where an agent must decide between two actions, A_1 and A_2, but it does not know the distribution of rewards for either action. Unknown to the agent, selecting A_1 returns a random variable drawn from a normal distribution with mean $\mu = 3$ and standard deviation $\sigma = 1$; while selecting A_2 returns a random variable drawn from a Weibull distribution with shape parameter $a = 2$ and scale parameter $b = 2\sqrt{2}$.

a) In one instance of the MAB, the actions taken and rewards received for the first six trials are given in the table below:

Trial	Action	Reward
1	A_1	2.66
2	A_2	1.25
3	A_1	3.21
4	A_2	2.34
5	A_1	1.87
6	A_1	1.69

Using ϵ-greedy, which action is most likely to be chosen next?

Exercise 9.1a - ϵ-Greedy Strategy

- Choose a random action with probability ϵ, and choose the best action with probability $1 - \epsilon$

Exercise 10.1. Consider a situation where an agent must decide between two actions, A_1 and A_2, but it does not know the distribution of rewards for either action. Unknown to the agent, selecting A_1 returns a random variable drawn from a normal distribution with mean $\mu = 3$ and standard deviation $\sigma = 1$; while selecting A_2 returns a random variable drawn from a Weibull distribution with shape parameter $a = 2$ and scale parameter $b = 2\sqrt{2}$.

a) In one instance of the MAB, the actions taken and rewards received for the first six trials are given in the table below:

Trial	Action	Reward
1	A_1	2.66
2	A_2	1.25
3	A_1	3.21
4	A_2	2.34
5	A_1	1.87
6	A_1	1.69

Using ϵ-greedy, which action is most likely to be chosen next?

$\epsilon < 0.5$

Exercise 9.1a - ϵ-Greedy Strategy

- Choose a random action with probability ϵ, and choose the best action with probability $1 - \epsilon$

- Assuming that the agent wishes to maximise its reward, under the ϵ-greedy strategy, it will choose the action with the highest reward with probability $1 - \epsilon$, and a random action with probability $\epsilon < 0.5$

- We compute the estimated rewards:

$$\hat{v}_1 = \frac{2.66+3.21+1.87+1.69}{4} = 2.3757$$

Trial	Action	Reward
1	A_1	2.66
2	A_2	1.25
3	A_1	3.21
4	A_2	2.34
5	A_1	1.87
6	A_1	1.69

Exercise 9.1a - ϵ-Greedy Strategy

- Choose a random action with probability ϵ, and choose the best action with probability $1 - \epsilon$

- Assuming that the agent wishes to maximise its reward, under the ϵ-greedy strategy, it will choose the action with the highest reward with probability $1 - \epsilon$, and a random action with probability $\epsilon < 0.5$

- We compute the estimated rewards:

$$\hat{v}_1 = \frac{2.66+3.21+1.87+1.69}{4} = 2.3757$$

$$\hat{v}_2 = \frac{1.25+2.34}{2} = 1.7950$$

Trial	Action	Reward
1	A_1	2.66
2	A_2	1.25
3	A_1	3.21
4	A_2	2.34
5	A_1	1.87
6	A_1	1.69

Exercise 9.1a - ϵ-Greedy Strategy

- Choose a random action with probability ϵ, and choose the best action with probability $1 - \epsilon$

Trial	Action	Reward
1	A_1	2.66
2	A_2	1.25
3	A_1	3.21
4	A_2	2.34
5	A_1	1.87
6	A_1	1.69

- Assuming that the agent wishes to maximise its reward, under the ϵ-greedy strategy, it will choose the action with the highest reward with probability $1 - \epsilon$, and a random action with probability $\epsilon < 0.5$

- We compute the estimated rewards:

$$\hat{v}_1 = \frac{2.66+3.21+1.87+1.69}{4} = 2.3757$$

$$\hat{v}_2 = \frac{1.25+2.34}{2} = 1.7950$$

- Since we err on the side of exploitation and $\hat{v}_1 > \hat{v}_2$, A_1 is the action to be chosen next, with probability of at least $1 - \epsilon$

Exercise 9.1b – UCB Strategy

- Choose the action with the highest upper confidence bound.

b) Given the same sample information, now consider UCB1 with upper bounds given by:

$$UCB1_a = \hat{v}_a + \sqrt{\frac{C \ \ln(N)}{n_a}}$$

Set the tunable parameter C to 5. Using this UCB algorithm, which action is chosen next?

Trial	Action	Reward
1	A_1	2.66
2	A_2	1.25
3	A_1	3.21
4	A_2	2.34
5	A_1	1.87
6	A_1	1.69

Exercise 9.1b – UCB Strategy

- Choose the action with the highest upper confidence bound.

$$UCB1_a = \hat{v}_a + \sqrt{\left(\frac{C \ln(N)}{n_a}\right)}$$

- We know that C=5, N=6, $n_{a1} = 4, n_{a2} = 2$

From the previous question, we know that $\hat{v}_1 = 2.3757, \hat{v}_2 = 1.7950$

Trial	Action	Reward
1	A_1	2.66
2	A_2	1.25
3	A_1	3.21
4	A_2	2.34
5	A_1	1.87
6	A_1	1.69

Exercise 9.1b – UCB Strategy

- Choose the action with the highest upper confidence bound.

$$UCB1_a = \hat{v}_a + \sqrt{\left(\frac{C \ln(N)}{n_a}\right)}$$

- We know that C=5, N=6, $n_{a1} = 4, n_{a2} = 2$

From the previous question, we know that $\hat{v}_1 = 2.3757, \hat{v}_2 = 1.7950$

Trial	Action	Reward
1	A_1	2.66
2	A_2	1.25
3	A_1	3.21
4	A_2	2.34
5	A_1	1.87
6	A_1	1.69

- For action A_1:

$$UCB1_{a1} = 2.3757 + \sqrt{\frac{5 \times \ln(6)}{4}} = 2.3757 + 1.4965 = 3.8541$$

- For action A_2

$$UCB1_{a2} = 1.7950 + \sqrt{\frac{5 \times \ln 6}{2}} = 1.7950 + 2.116 = 3.9115$$

Exercise 9.1b – UCB Strategy

- Choose the action with the highest upper confidence bound.

$$UCB1_a = \hat{v}_a + \sqrt{\left(\frac{C \ln(N)}{n_a}\right)}$$

- We know that C=5, N=6, $n_{a1} = 4, n_{a2} = 2$

From the previous question, we know that $\hat{v}_1 = 2.3757, \hat{v}_2 = 1.7950$

Trial	Action	Reward
1	A_1	2.66
2	A_2	1.25
3	A_1	3.21
4	A_2	2.34
5	A_1	1.87
6	A_1	1.69

- For action A_1:

$$UCB1_{a1} = 2.3757 + \sqrt{\frac{5 \times \ln(6)}{4}} = 2.3757 + 1.4965 = 3.8541$$

- For action A_2

$$UCB1_{a2} = 1.7950 + \sqrt{\frac{5 \times \ln 6}{2}} = 1.7950 + 2.116 = 3.9115$$

Confidence Interval

Exercise 9.1b – UCB Strategy

- Choose the action with the highest upper confidence bound.

$$UCB1_a = \hat{v}_a + \sqrt{\left(\frac{C \ln(N)}{n_a}\right)}$$

- We know that C=5, N=6, $n_{a1} = 4, n_{a2} = 2$

From the previous question, we know that $\hat{v}_1 = 2.3757, \hat{v}_2 = 1.7950$

Trial	Action	Reward
1	A_1	2.66
2	A_2	1.25
3	A_1	3.21
4	A_2	2.34
5	A_1	1.87
6	A_1	1.69

- For action A_1:

$$UCB1_{a1} = 2.3757 + \sqrt{\frac{5 \times \ln(6)}{4}} = 2.3757 + 1.4965 = 3.8541$$

- For action A_2

$$UCB1_{a2} = 1.7950 + \sqrt{\frac{5 \times \ln 6}{2}} = 1.7950 + 2.116 = 3.9115$$

Confidence Interval

Exercise 9.1b – UCB Strategy

- For action A_1:

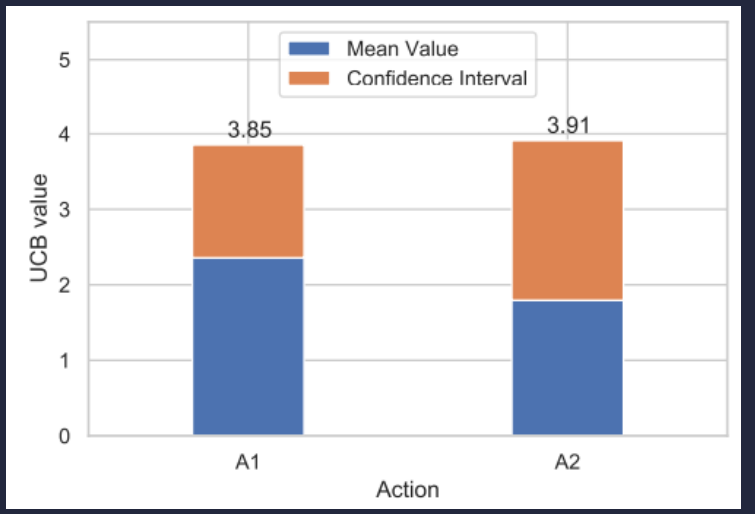
$$UCB1_{a1} = 2.3757 + \sqrt{\frac{5 \times \ln(6)}{4}} = 2.3757 + 1.4965 = 3.8541$$

- For action A_2

$$UCB1_{a2} = 1.7950 + \sqrt{\frac{5 \times \ln 6}{2}} = 1.7950 + 2.116 = 3.9115$$

- Action A_2 is chosen next as it has the greater upper confidence bound.
- A_2 has been sampled fewer times than A_1 - greater uncertainty in estimate, reflected in larger interval size.

Trial	Action	Reward
1	A_1	2.66
2	A_2	1.25
3	A_1	3.21
4	A_2	2.34
5	A_1	1.87
6	A_1	1.69



Exercise 9.1b – UCB Strategy

- For action A_1:

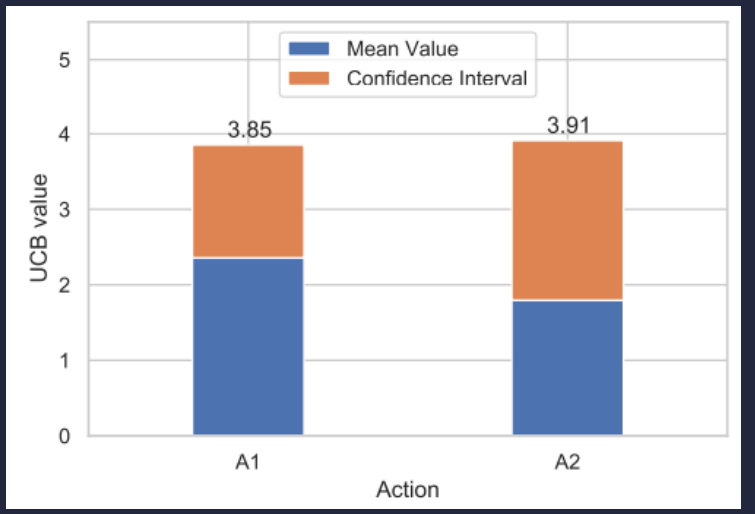
$$UCB1_{a1} = 2.3757 + \sqrt{\frac{5 \times \ln(6)}{4}} = 2.3757 + 1.4965 = 3.8541$$

- For action A_2

$$UCB1_{a2} = 1.7950 + \sqrt{\frac{5 \times \ln 6}{2}} = 1.7950 + 2.116 = 3.9115$$

- Action A_2 is chosen next as it has the greater upper confidence bound.
- A_2 has been sampled fewer times than A_1 - greater uncertainty in estimate, reflected in larger interval size.

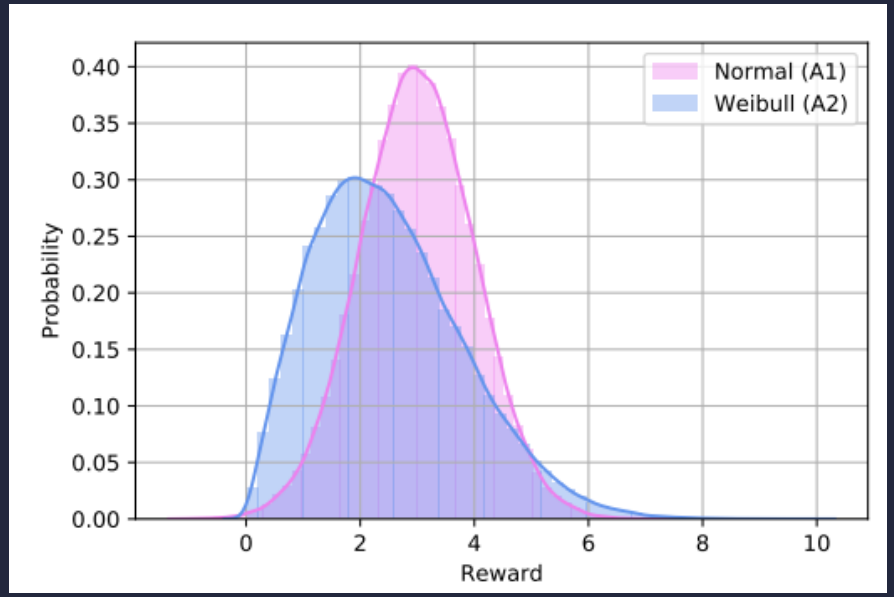
Trial	Action	Reward
1	A_1	2.66
2	A_2	1.25
3	A_1	3.21
4	A_2	2.34
5	A_1	1.87
6	A_1	1.69



Exercise 9.1c – Probability Distributions

c) Plot the distributions of rewards from each arm. *Note: the support code provides classes for arms with normal and Weibull random rewards, which also show you how to sample from these distributions.* **Question**: If the agent wishes to maximise its cumulative reward over time and knew these distributions, which would be the optimal arm to pull?

- The mean of the normal distribution is given as $\mu = 3$
- The mean of the Weibull distribution is given as $2\sqrt{\frac{\pi}{2}} \approx 2.506$ in this case.



Exercise 9.1d – MAB Algorithm Comparison

d) Set up a MAB instance with two arms described above, and consider the ϵ-greedy exploration strategy with random sampling parameter set to $\epsilon = 0.1$, and the UCB bound as described in b) above. For each strategy, plot their cumulative rewards over 1000 arm trials in an MAB instance. **Questions**: Which performs better initially? Which performs better in the long run?

Exercise 9.1d – MAB Algorithm Comparison

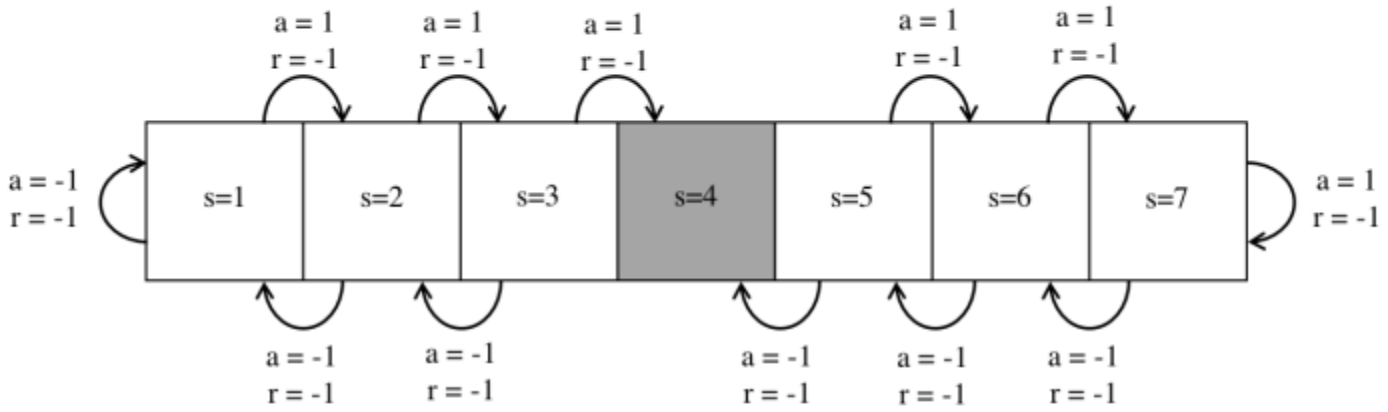
d) Set up a MAB instance with two arms described above, and consider the ϵ-greedy exploration strategy with random sampling parameter set to $\epsilon = 0.1$, and the UCB bound as described in b) above. For each strategy, plot their cumulative rewards over 1000 arm trials in an MAB instance. **Questions**: Which performs better initially? Which performs better in the long run?

- Difficult to say which algorithm performs better initially – dependent on implementation and random realisation of rewards.

- In the long run, expect UCB1 algorithm to outperform ϵ-Greedy
  - ϵ-Greedy continues to sample random arms with a fixed probability ϵ even after the mean reward estimates have been converged to close to their true values
  - In contrast, UCB1 reduces the chance of exploring away from the highest-value arm by adjusting the confidence interval based on the number of samples taken.

Exercise 9.2

Exercise 10.2. Consider the chain MDP shown in the figure below[2]. The MDP is episodic and deterministic with $n = 7$ states arranged in a line.

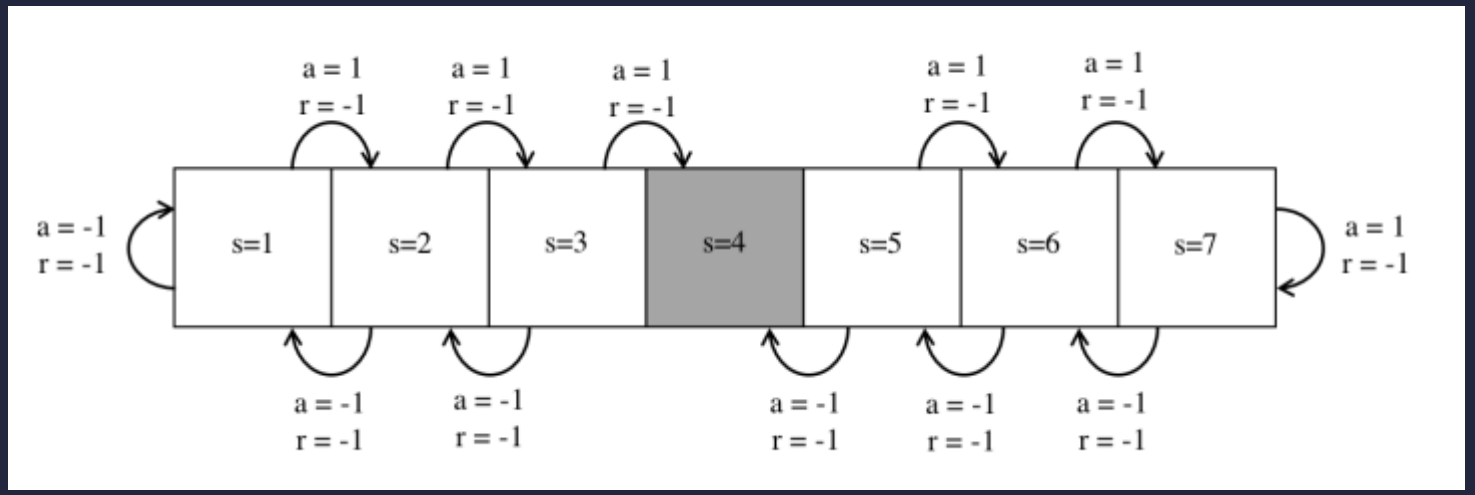


The possible actions are $a \in \{-1, 1\}$, and the transition function is deterministic such that $s' = s + a$. As an exception, taking $a = -1$ from $s = 1$ results in $s' = 1$, and taking $a = 1$ from $s = 7$ results in $s' = 7$.

The goal state is $s = 4$ (shaded in grey), and any action within the goal state ends the episode with reward of $r = 0$. From all other states, any action incurs a reward of $r = -1$. Let the discount factor, $\gamma = 1$.

a) Write an expression for the Value of each state, $V^*(s)$

Exercise 9.2a



Task: Write an expression for the value of each state, $V^*(s)$

- State 4 has a value of 0

Moving from State 4 -> State 3 and State 4 -> State 5 incurs a reward of -1

- State 3 has a value of -1
- State 5 has a value of -1

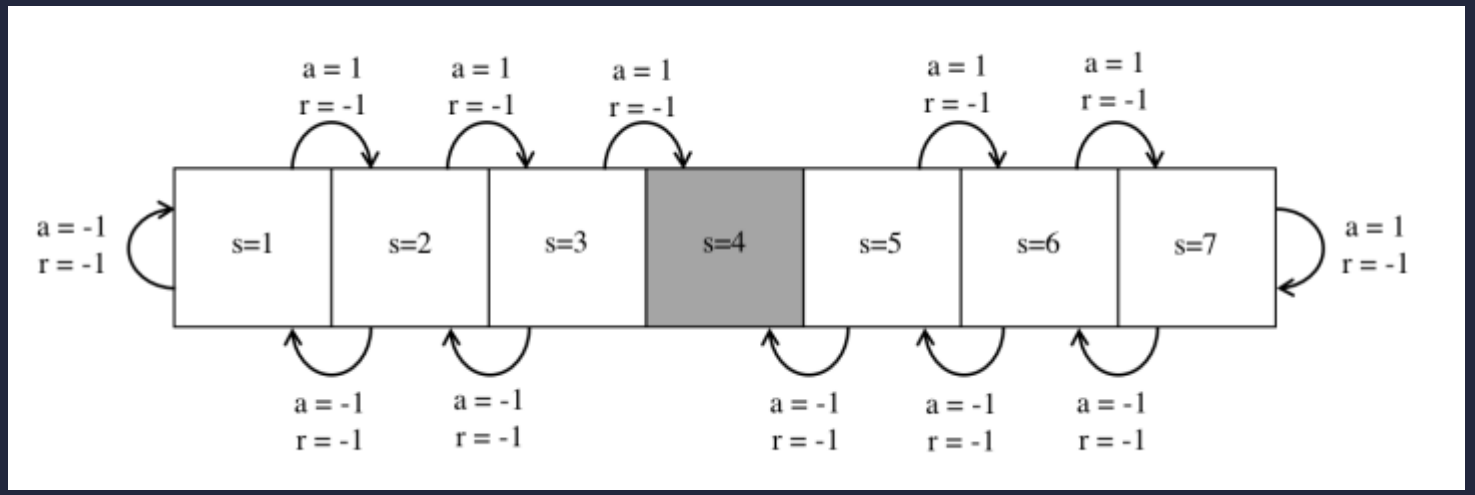
Moving from State 3 -> State 2 and State 5 -> State 6 incurs a reward of -1

- State 2 has a value of -2
- State 6 has a value of -2

Moving from State 2 -> State 1 and State 6 -> State 7 incurs a reward of -1

- State 1 has a value of -3
- State 7 has a value of -3

Exercise 9.2a



Task: Write an expression for the value of each state, $V^*(s)$

- State 4 has a value of 0

Moving from State 4 -> State 3 and State 4 -> State 5 incurs a reward of -1

- State 3 has a value of -1
- State 5 has a value of -1

Moving from State 3 -> State 2 and State 5 -> State 6 incurs a reward of -1

- State 2 has a value of -2
- State 6 has a value of -2

Moving from State 2 -> State 1 and State 6 -> State 7 incurs a reward of -1

- State 1 has a value of -3
- State 7 has a value of -3

$$V^*(s) = -|s - 4|$$

Exercise 9.2b

b) Suppose we observe the following episode:

state	action	reward
s_3	-1	-1
s_2	1	-1
s_3	1	-1
s_4	1	0

Perform tabular Q-learning on this chain MDP, determining what values the Q-function attains if we initialise the Q-values to 0 and replay the experience in the table once. Use a learning rate $\alpha = 0.5$, and update the values for $Q(3, -1)$, $Q(2, 1)$, $Q(3, 1)$.

Exercise 9.2b

state	action	reward
s_3	-1	-1
s_2	1	-1
s_3	1	-1
s_4	1	0

From the lecture on Q-Learning, we know that given a tuple (s, a, r, s'), we can use the update equation to perform Q-Learning.

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \times \max_{a' \in \{-1,1\}} Q(s',a') - Q(s,a))$$

Using this equation, with hyperparameters $\alpha = 0.5, \gamma = 1$, we have:

$$Q(3,-1) \leftarrow 0 + 0.5(-1 + \max_{a' \in \{-1,1\}} Q(2,a') = 0.5(-1+0) = -0.5$$

$$Q(2,1) \leftarrow 0 + 0.5(-1 + \max_{a' \in \{-1,1\}} Q(3,a')) = 0.5(-1+0) = -0.5$$

$$Q(3,1) \leftarrow 0 + 0.5(-1 + \max_{a' \in \{-1,1\}} Q(4,a') = 0.5(-1+0) = -0.5$$

Exercise 9.3a

a) List the dimensions of complexity for the Gridworld reinforcement learning environment on the following page:
Modularity, Planning horizon, Representation, Computational limits, Learning, Sensing uncertainty, Effect uncertainty, Preference, Number of agents, Interaction.

Exercise 9.3a

a) List the dimensions of complexity for the Gridworld reinforcement learning environment on the following page:
Modularity, Planning horizon, Representation, Computational limits, Learning, Sensing uncertainty, Effect uncertainty, Preference, Number of agents, Interaction.

- Modularity - Flat
- Planning Horizon - Indefinite Stage / Infinite Stage
- Representation - States and Features
- Computational Limits - Perfect Rationality
- Learning - Knowledge is Learned
- Sensing Uncertainty - Fully Observable
- Effecting Uncertainty - Stochastic
- Preference - Complex Preferences
- Number of Agents - Single Agent
- Interaction - Online

Exercise 9.3a

a) List the dimensions of complexity for the Gridworld reinforcement learning environment on the following page:
Modularity, Planning horizon, Representation, Computational limits, Learning, Sensing uncertainty, Effect uncertainty, Preference, Number of agents, Interaction.

- Modularity - Flat
- Planning Horizon - Indefinite Stage / Infinite Stage
- Representation - States and Features
- Computational Limits - Perfect Rationality
- Learning - Knowledge is Learned
- Sensing Uncertainty - Fully Observable
- Effecting Uncertainty - Stochastic
- Preference - Complex Preferences
- Number of Agents - Single Agent
- Interaction - Online

Exercise 9.3b

b) In RL, what is the connection between the environment's state transition function and the exploration policy used by the agent?

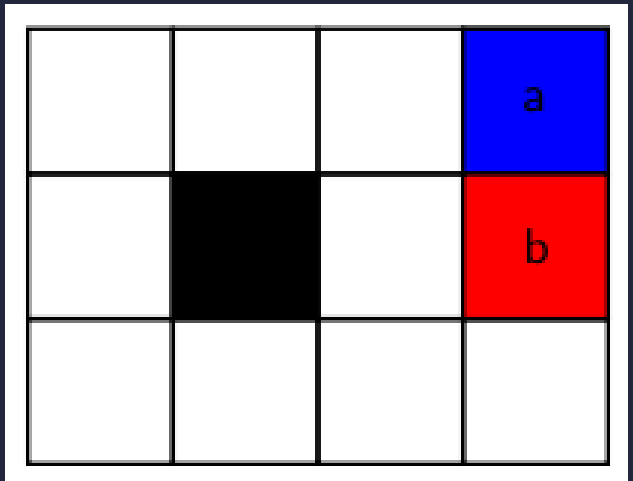
In Reinforcement learning problems, when the transition function is unknown / uncertain, we should have a higher exploration term to learn information about the environment. As it becomes more certain, we should lower the exploration term (similar to UCB, and "Annealing Epsilon-Greedy" in the supporting code, where ϵ is reduced over time).

Exercise 9.3b

b) In RL, what is the connection between the environment's state transition function and the exploration policy used by the agent?

In Reinforcement learning problems, when the transition function is unknown / uncertain, we should have a higher exploration term to learn information about the environment. As it becomes more certain, we should lower the exploration term (similar to UCB, and "Annealing Epsilon-Greedy" in the supporting code, where ϵ is reduced over time).

Exercise 10.4



States in this environment are the positions on the tiles. The world is bounded by a boundary wall, and there is one obstacle, at [1,1] (using python or zero indexing starting from the bottom left corner). In addition, there are two terminal states, indicated by the coloured squares.

Actions and Transitions: In this world, an agent can generally choose to move in four directions — *up*, *down*, *left* and *right*. However, the agent moves successfully with only $p = 0.8$, and moves perpendicular to its chosen direction with $p = 0.1$ in each perpendicular direction. If it hits a wall or obstacle, the agent stays where it is (i.e. no collision cost). In addition, once the agent arrives on a coloured square with a value, it has only one special action available to it; that is, to *exit* the environment.

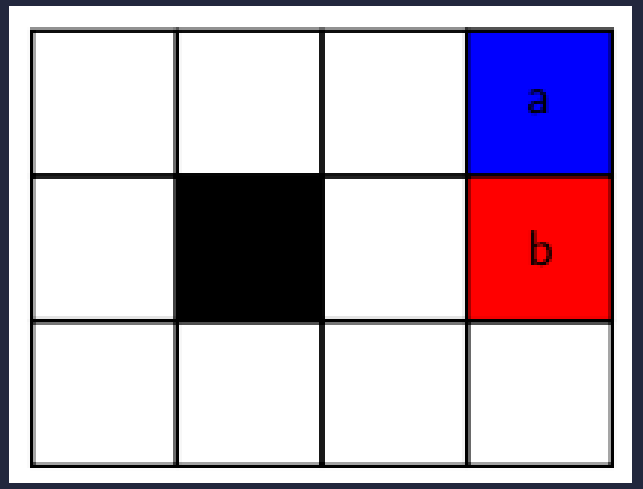
Rewards: The values stated on the coloured squares are the reward for *exiting* the square and the environment, so the reward is not repeatedly earned; that is, $R([3, 2], exit) = a$, and $R([3, 1], exit) = b$. All other states have 0 reward.

Discount factor: $\gamma = 0.9$.

Exercise 10.4. For the grid world above, develop a simulator for the state-action-reward-state loop.

- In particular, you should develop a `Grid.apply_move()` function that takes the current state and actions as inputs and updates the state while returning the reward for the state-action-state transition.
- The provided code includes `Grid.player_x` and `Grid.player_y` variables that represent the state of the player on the (x, y) grid.
- You should make use of the provided code, especially the `stoch_action()` and `attempt_move()` methods.
- Include a way to restart the agent in a random map location, avoiding obstacles, after it exits the environment.

Exercise 10.4



For this GridWorld, the environment randomness is based on action noise.

- An action is chosen (this is the "*nominal action*") and with some probability, a different action is selected *("performed action*")

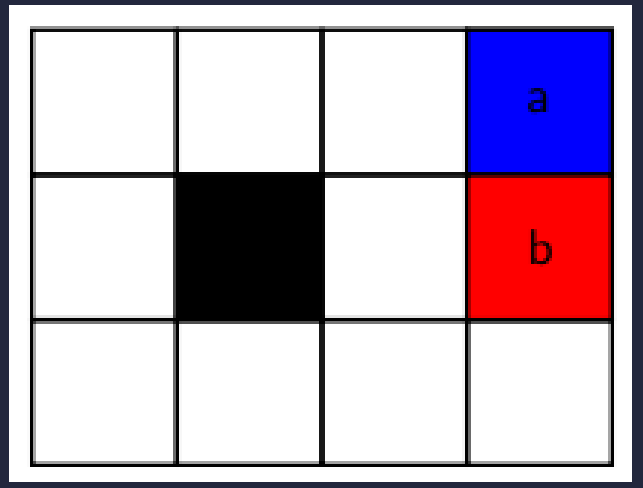
- Because of this, it makes sense to split the environment dynamics into a deterministic component attempt_move(…) and a stochastic (non-deterministic) component stoch_action(…)

Exercise 10.4

```
def apply_move(self, s, a):
    # handle special cases
    if s in REWARDS.keys():
        # go to the exit state
        next_state = EXIT_STATE
        reward = REWARDS[s]
        return next_state, reward
    elif s == EXIT_STATE:
        # go to a new random state (end of episode)
        next_state = random.choice(self.states)
        reward = 0
        return next_state, reward

    # choose a random true action
    r = random.random()
    cumulative_prob = 0
    action = None
    for k, v in self.stoch_action(a).items():
        cumulative_prob += v
        if r < cumulative_prob:
            action = k
            break

    # apply true action
    next_state = self.attempt_move(s, action)
    reward = 0
    return next_state, reward
```

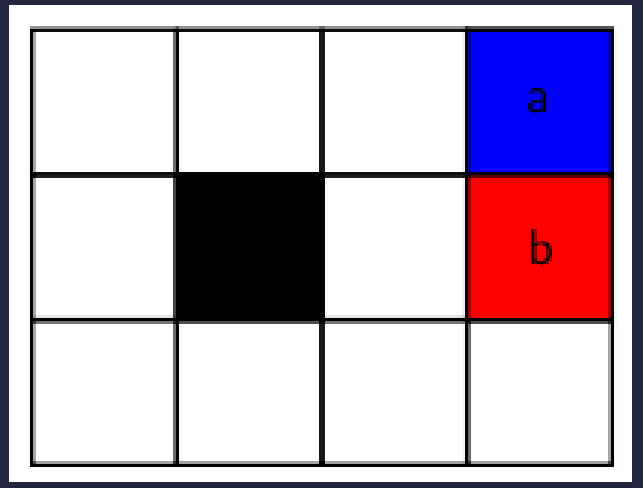


Exercise 10.4

```
def apply_move(self, s, a):
    # handle special cases
    if s in REWARDS.keys():
        # go to the exit state
        next_state = EXIT_STATE
        reward = REWARDS[s]
        return next_state, reward
    elif s == EXIT_STATE:
        # go to a new random state (end of episode)
        next_state = random.choice(self.states)
        reward = 0
        return next_state, reward

    # choose a random true action
    r = random.random()
    cumulative_prob = 0
    action = None
    for k, v in self.stoch_action(a).items():
        cumulative_prob += v
        if r < cumulative_prob:
            action = k
            break

    # apply true action
    next_state = self.attempt_move(s, action)
    reward = 0
    return next_state, reward
```




```
def stoch_action(self, a):
    # Stochastic actions probability distributions
    if a == RIGHT:
        return {RIGHT: self.p, UP: (1-self.p)/2, DOWN: (1-self.p)/2}
    elif a == UP:
        return {UP: self.p, LEFT: (1-self.p)/2, RIGHT: (1-self.p)/2}
    elif a == LEFT:
        return {LEFT: self.p, UP: (1-self.p)/2, DOWN: (1-self.p)/2}
    elif a == DOWN:
        return {DOWN: self.p, LEFT: (1-self.p)/2, RIGHT: (1-self.p)/2}
```

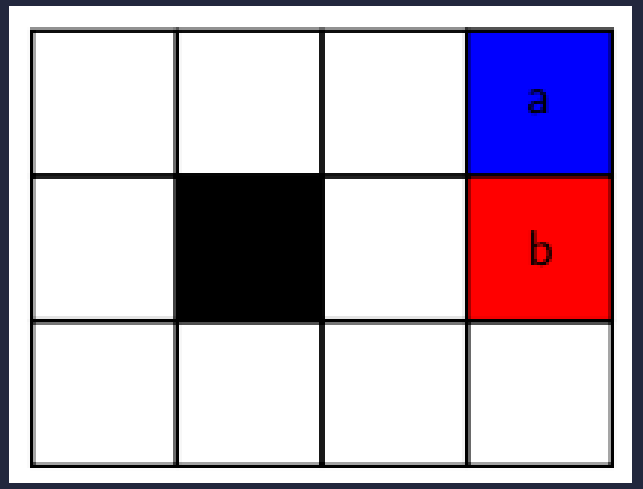
Exercise 10.5

Exercise 10.5. Using your simulator, implement Q-learning for this gridworld problem.

a) First, write a function to choose an action given the Q-value estimates. Incorporate your agent's exploration strategy in this function.

b) Then write a `next_iteration()` function, which will probe the simulator to receive a reward and a new state, and use this to update your agent's Q-values.

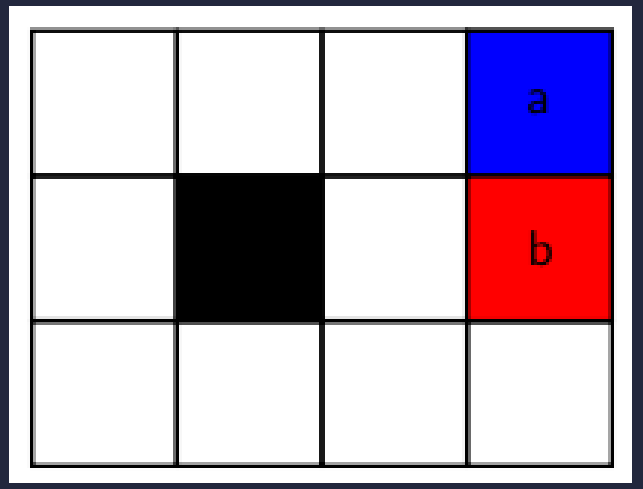
- Be sure to include an exploration strategy
- See the provided code.



- Often better to write the Q-learning algorithm as two separate parts:
  - Choosing action based on exploration strategy
  - Choosing optimal action based on Q-values

Exercise 10.5a

- Algorithm for choosing action from optimal action

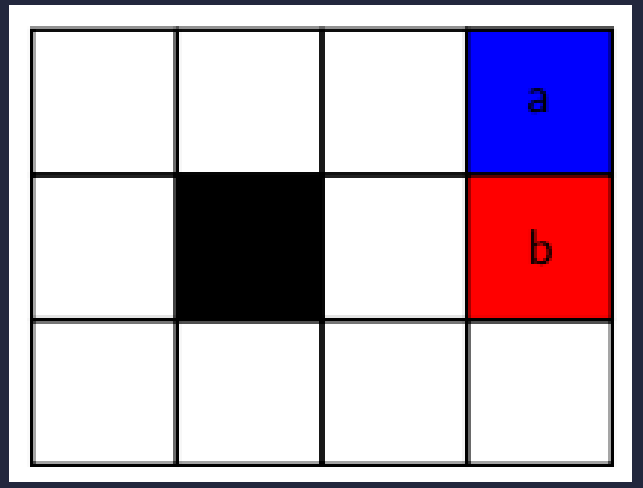



```
def select_action(self, state):
    # choose the action with the highest Q-value for the given state
    best_q = -math.inf
    best_a = None
    for a in ACTIONS:
        if ((state, a) in self.q_values.keys() and
                self.q_values[(state, a)] > best_q):
            best_q = self.q_values[(state, a)]
            best_a = a

    if best_a is None:
        return random.choice(ACTIONS)
    else:
        return best_a
```

Exercise 10.5

- Often better to write the Q-learning algorithm as two separate parts:
  - Choosing action based on exploration strategy
  - Choosing optimal action based on Q-values



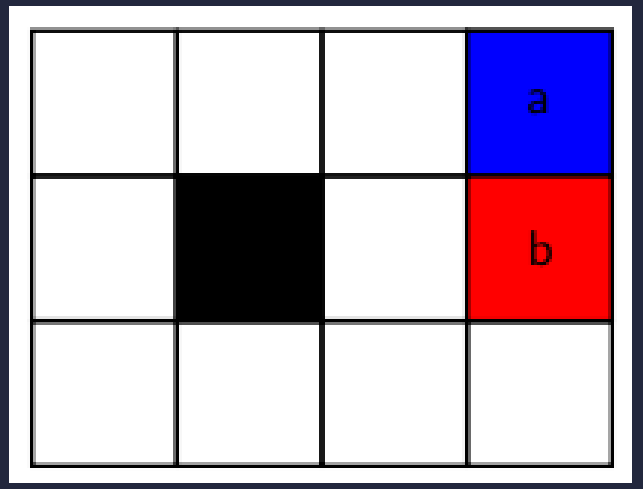

```
best_q = -math.inf
best_a = None

for a in ACTIONS:
    if ((self.persistent_state, a) in self.q_values.keys() and
            self.q_values[(self.persistent_state, a)] > best_q):
    best_q = self.q_values[(self.persistent_state, a)]
    best_a = a

    if best_a is None or random.random() < self.EPSILON: # exploration
        action = random.choice(ACTIONS)
    else:  # exploitation
        action = best_a
```

Exercise 10.5

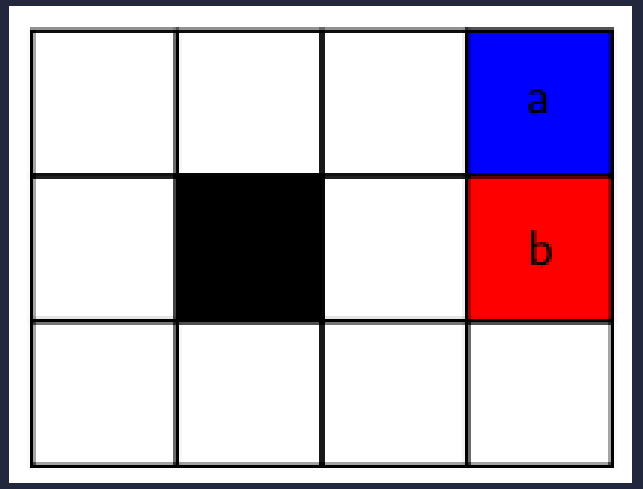
- Often better to write the Q-learning algorithm as two separate parts:
  - Choosing action based on exploration strategy
  - Choosing optimal action based on Q-values




```
next_state, reward = self.grid.apply_move(self.persistent_state, action)
```

Exercise 10.5

- Often better to write the Q-learning algorithm as two separate parts:
  - Choosing action based on exploration strategy
  - Choosing optimal action based on Q-values




```
# ===== update value table =====
# Q(s,a) <-- Q(s,a) + alpha * (temporal difference)
# Q(s,a) <-- Q(s,a) + alpha * (target - Q(s, a))
# target = r + gamma * max_{a' in A} Q(s', a')
# compute target
best_q1 = -math.inf
best_a1 = None
for a1 in ACTIONS:
    if ((next_state, a1) in self.q_values.keys() and
            self.q_values[(next_state, a1)] > best_q1):
        best_q1 = self.q_values[(next_state, a1)]
        best_a1 = a1
if best_a1 is None or next_state == EXIT_STATE:
    best_q1 = 0
target = reward + (self.grid.discount * best_q1)
if (self.persistent_state, action) in self.q_values:
    old_q = self.q_values[(self.persistent_state, action)]
else:
    old_q = 0
self.q_values[(self.persistent_state, action)] = old_q + (self.ALPHA * (target - old_q))

# move to next state
self.persistent_state = next_state
```

Exercise 10.6

Exercise 10.6. In reinforcement learning problems, often the goal is to learn the value function V^π from episodes of experience under policy π. Recall that the *return* is defined as the total discounted reward: $G_t = R_{t+1} + \gamma R_{t+2} + \ldots \gamma^{T-1} R_T$ and the value function $V^\pi(s) = E[G_t \mid S_t = s]$, is the expected return.

a) Compare the state value ($V(S_t)$) update formula for Monte Carlo vs Temporal Difference (TD) reinforcement learning with a 1-step lookahead, i.e. TD(0).

b) Compare the update formula for the state value, $V(S_t)$, in TD learning vs the update formula for $Q(s, a)$ in Q-learning.

c) Consider Q-learning with linear value function approximation, where:

$$Q(s,a) = \sum_i^n f_i(s,a) w_i = w_1 f_1(s,a) + w_2 f_2(s,1) + \cdots + w_n f_n(s,a).$$

What does the update function for Q-learning with linear q-functions become?

Exercise 10.6a

Exercise 10.6. In reinforcement learning problems, often the goal is to learn the value function V^{π} from episodes of experience under policy π. Recall that the *return* is defined as the total discounted reward: $G_t = R_{t+1} + \gamma R_{t+2} + \ldots \gamma^{T-1} R_T$ and the value function $V^{\pi}(s) = E[G_t \mid S_t = s]$, is the expected return.

a) Compare the state value ($V(S_t)$) update formula for Monte Carlo vs Temporal Difference (TD) reinforcement learning with a 1-step lookahead, i.e. TD(0).

- In the Monte Carlo and Temporal Difference formulas, we have the following equation, however the "TARGET" differs in each:

$$V(S_t) \leftarrow V(S_t) + \alpha[TARGET - V(S_t)]$$

- In the Monte Carlo Reinforcement learning algorithm, we use the true final reward returned, G_t from the completion of the episode to update the values for each state

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

Exercise 10.6a

Exercise 10.6. In reinforcement learning problems, often the goal is to learn the value function V^{π} from episodes of experience under policy π. Recall that the *return* is defined as the total discounted reward: $G_t = R_{t+1} + \gamma R_{t+2} + \ldots \gamma^{T-1} R_T$ and the value function $V^{\pi}(s) = E[G_t \mid S_t = s]$, is the expected return.

a) Compare the state value ($V(S_t)$) update formula for Monte Carlo vs Temporal Difference (TD) reinforcement learning with a 1-step lookahead, i.e. TD(0).

- In the Monte Carlo and Temporal Difference formulas, we have the following equation, however the "TARGET" differs in each:

$$V(S_t) \leftarrow V(S_t) + \alpha[TARGET - V(S_t)]$$

- In the Monte Carlo Reinforcement learning algorithm, we use the true final reward returned, G_t from the completion of the episode to update the values for each state

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

- In TD(0) learning, we don't need to wait until the end of the episode to update $V(S_t)$, and instead uses a one-step look-ahead based on current estimates.
- That is, at time $t + 1$, the temporal difference method uses the observed reward R_{t+1} and immediately forms a TD target $R_{t+1+\gamma}V(S_{t+1})$, updating $V(S_t)$ with the TD error.
- Therefore, the TD(0) update equation is defined as

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Exercise 10.6b

b) Compare the update formula for the state value, $V(S_t)$, in TD learning vs the update formula for $Q(s,a)$ in Q-learning.

- Note: Q-Learning is an implementation of Temporal Difference learning.
- For Q-learning, we replace $V(S_t)$ with $Q(s,a)$
- In addition to this, the value at the next step is estimated from the optimal policy, where we use the action that maximises the Q-value $V(S_{t+1})$ becomes $\max_{a'} Q(s',a')$

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R_{t+1} + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Exercise 10.6c

c) Consider Q-learning with linear value function approximation, where:

$$Q(s,a) = \sum_{i}^{n} f_i(s,a)w_i = w_1 f_1(s,a) + w_2 f_2(s,1) + \cdots + w_n f_n(s,a).$$

What does the update function for Q-learning with linear q-functions become?

- In linear Value Function Approximations, updates to $Q(s,a)$ are replaced by updates to the weights using the following formula:

$$w_i \leftarrow w_i + \alpha\delta f_i(s,a)$$

  - $\delta = \left[R_{t+1} + \gamma \max_{a'} Q_{\overline{w}}(s',a') - Q_{\overline{w}}(s,a)\right]$
  - $Q_{\overline{w}}$ is the Q-table indexed by features
- This is effectively adjusting the feature weights w_i to reduce the difference between the sampled value and the estimated expected value.
- Instead of learning the Q-table, we are learning weights which are used to infer what the Q-Value for a particular (state, action) pair
  - We hope that learning the weights of features will allow us to generalise the value of states that we haven't encountered before.
- Q-value is computed each time from the sum of weights, rather than being stored in a table.
- This is essentially a single-layer linear Deep Neural Network

Exercise 10.7

Consider a reinforcement learning problem in the game of Pacman with linear value function approximation and Q-learning. The actions available to the agent are 'Up', 'Down', 'Left', 'Right'. Assume we observe an 'Up' action where Pacman's next state would result in being eaten and receiving a reward of -500, and that we are using two linear features to represent $Q(s, a)$: a feature representing the proximity to food, f_{food}, and the other representing the proximity to ghosts, f_{ghost}.

Using linear function approximation, update the weights, using a learning rate $\alpha = \frac{1}{250}$, given the following:

$$Q(s, a) = 4.0 f_{food}(s, a) - 1.0 f_{ghost}(s, a)$$

$$\begin{aligned} f_{food}(s, Up) &= 0.5 \\ f_{ghost}(s, Up) &= 1.0 \\ Q(s, a) &= +1 \\ R_{t+1} = R(s, a, s') &= -500 \end{aligned}$$

Exercise 10.7

$$Q(s,a) = 4.0 f_{food}(s,a) - 1.0 f_{ghost}(s,a)$$

$$\begin{aligned} f_{food}(s, Up) &= 0.5 \\ f_{ghost}(s, Up) &= 1.0 \\ Q(s,a) &= +1 \\ R_{t+1} = R(s,a,s') &= -500 \end{aligned}$$

- $\delta = -501$ (difference / temporal difference)
- $\delta = \left[R_{t+1} + \gamma \max_{a'} Q_{\overline{w}}(s', a') - Q_{\overline{w}}(s,a)\right] = -500 + 0 - 1$
- $w_i \leftarrow w_i + \alpha\gamma f_i(s,a)$
  - $w_{food} \leftarrow 4.0 + \alpha[-501] \times 0.5$
  - $w_{ghost} \leftarrow -1.0 + \alpha[-501] \times 1.0$
- Therefore, the updated weights in the Q-function are:

$$Q(s,a) = 3.0 \times f_{food}(s,a) - 3.0 \times f_{ghost}(s,a)$$

  - More precisely, $w_{food} = w_{ghost} = 3.0$