An Introduction to Model-Free Reinforcement Learning and the Q-learning Algorithm

ECE/CS 532
Summer 2019
Instructor: Barry Van Veen

Evan Palmer: epalmer4@wisc.edu
YeEun Lim: ylim52@wisc.edu
Aleksander Lesiewicz: lesiewicz@wisc.edu

Summary:

The machine learning topic that we have chosen to focus on is reinforcement learning. Reinforcement learning (RL), specifically model-free reinforcement learning (we will refer to this as MFRL in this activity), is an area of machine learning where the machine "learns" what action it should take based on trial and error. Reinforcement learning has many applications such as game playing and robotics.

In this activity we will introduce the following concepts: the concept of having a reward/punishment system, the basic terminology of RL, some common algorithms and equations used in MFRL, and a few practical applications of MFRL. You will first learn about the background and the above concepts. Then you will go through a warm-up activity designed as a self-check to make sure that you understand the basic concepts. Then you will go through a more thorough main activity focused on MFRL algorithms, specifically Q-learning.

At the end of this activity you should be able to do the following: define the terminology associated with RL, understand the underlying mathematics of MFRL. This includes the optimal value function and Q-learning update equation. You should also be able to get an idea about how to train RL agents with the Q-learning algorithm.

Background:

Introduction to Reinforcement Learning

Reinforcement learning is a subset of machine learning where an agent takes certain actions, given a certain state, in order to maximize a reward. This process can be visualized in Figure 1 below.

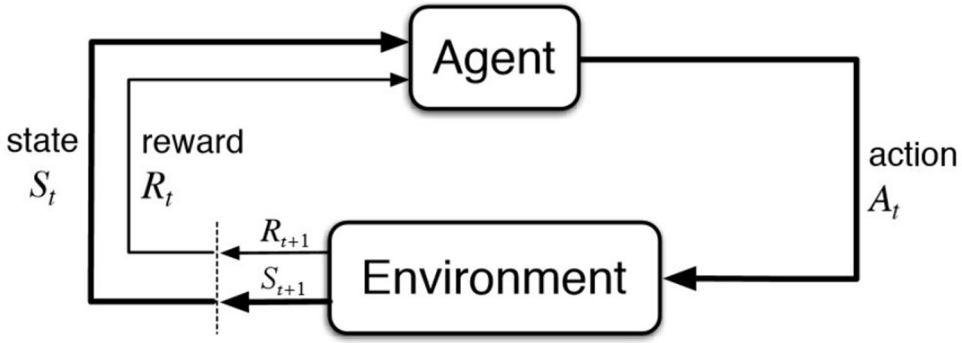


Figure 1: Reinforcement learning visualized (from towardsdatascience.com)

Two subcomponents were researched independently before creating modern reinforcement learning. These subcomponents are animal learning by trial and error and optimal control. The subcomponents came together in the late 1980s to create the field of modern reinforcement learning. Optimal control method came about in the late 1950s. Optimal control problems can be solved by an equation called the Bellman equation that was developed in the mid-1950s. These problems are addressed by methods using optimal return functions. Trial and error learning can go back as far as the 1850s. In terms of animal learning "reinforcement learning" is

related to Pavolv's dogs and their conditioned reflexes to a stimulus. This reinforcement learning was then transferred to computers in the 1940s, right around the time that development of artificial intelligence was beginning. The term "reinforcement learning" was first used in engineering literature in the 1960s. As reinforcement learning gained popularity, a subfield of neuroscience that uses the relationship between reinforcement learning algorithms and reinforcement learning in the nervous system was developed. From the study of the two subcomponents (animal learning by trial and error and optimal control), a computational method called temporal difference learning was developed to solve optimal control problems.

Reinforcement learning problems are derived from Markov Decision Processes (MDPs). An MDP is based on the Markov Property that states, "the future is independent of the past given the present". This means that all actions taken to get to the current state are no longer of concern to the agent.

In model-based RL (MBRL), the agent already knows all combinations of states and actions. The agent will learn the transition probability from the current state, given an action, to the next state that maximizes its reward. In this way, you can see that MBRL is closer to a supervised learning problem, since all possible transitions and actions are known. Hopefully, you can already see one issue with MBRL. If all possible states and actions in each state must be stored, then as the number of states and possible actions increase the amount of memory needed to store them increases as well. In this way, MBRL is not practical for large environments. Even though we will not be simulating large environments in this activity, we still have chosen to focus on model-free reinforcement learning (MFRL). MFRL differs from MBRL in that it does not require all the possible states and actions for each state to be stored. MFRL relies on the agent learning through trial and error.

Basic Terminology

Before anything else is covered, it is important to go over the basic terminology associated with reinforcement learning. Common terms can be seen below:

Environment: the "place" where the agent can act. This can be a game engine or maybe a room where a robot learns how to complete basic human tasks.

Agent: takes actions to change the state in an environment. Can be thought of as the "thing" that is learning.

State (S): Current situation that is returned by the environment. In the context of game playing this could mean the specific frame on the screen.

Action (A): All possible moves the agent can take in a certain state. If you think about a robot in a room, possible actions could include walk forward, jump, or stand still.

Reward (R): Immediate return that is sent back by the environment to evaluate the last action. In RL this is represented by a positive or negative number.

Policy (π): The specific strategy that the agent uses to determine its next action based on the current state. Think about the robot in a room. If it is facing a wall an inch in front of it an effective policy would not have it walk forward.

Discount Factor (γ): Used to weigh immediate rewards against future rewards.

Learning Rate (α): A measure of how quickly the agent will learn the environment.

Value (V): Expected long-term return based on the current state. The discount factor is included. This is different than R which is short-term.

Q-value or Action-value (Q): Similar to V, except it takes the action into account as opposed to V that only accounts for the current state.

Value function: Maps from states to real numbers. The value of a state signifies the long-term reward that the agent can earn from starting from that state and making actions based on a policy. RL agents learn policies indirectly by learning value functions.

Model-Free Reinforcement Learning

Model-free reinforcement learning can be more easily understood by thinking about a young child. The child takes certain actions and receives a certain reward for each action they take. From the reward they receive they can develop an idea about whether they should do that again or not. It is good to note that the reward can either be good or bad. For example, think about a young child. One environment might be the kitchen in the house that the child lives in. If the child saw a lit candle in the kitchen and tried to touch the flame they would get burned. The action in this case would be the child moving their hand toward the flame. The reward in this case is negative. Another possible environment might be the park. If the child picks up trash on the ground and throws it away, then the action would be throwing away the trash. They could also receive a positive reward like extra play time. It should be noted that only the environments, actions, and rewards were mentioned in these two scenarios in order to keep the examples general.

The agent goes through the same process when it is taking actions. It will be in a certain environment and it will take certain actions in certain states. Based on those actions it will receive a reward. In the case of the agent, the reward would just be a negative or positive value added to the cumulative reward. The mathematical concepts of MDP will not specifically be covered, but it is important to relate the given mathematical concepts to an MDP because that is where they are derived from.

In reality, MFRL is not as simple as we introduced above. If you think about the scenario with the child touching the lit candle, the reward is not as simple as positive or negative. The child may bring their hand close to the candle and notice that it feels warm. This could be a positive reward. Then at a certain point the warm changes to burn, and the reward becomes negative. If we decide to break up the distance from the child to the candle in several discrete points, then we can think of the different distances from the candle as different states. For simplicity, in each state the child will have three options, move their hand closer to the candle flame, pull their hand

away from the candle flame, or keep their hand in the same location. As the distances get closer to the candle, up to a certain point, the reward will increase. Once the point of warmth is passed and it turns into the burn region then the rewards will be subtracted. The child must employ a policy to maximize its reward, in this case warmth. As stated in the terminology, the policy is indirectly learned from the value function. The child will learn the policy by learning the maximum reward is achieved by bringing their hand directly to the warm area and keeping there until they get too warm. On the first few attempts the child may stay too far away from the candle and receive no warmth, or they may keep moving toward the candle until they get burned. In both cases the child would not be maximizing the reward. After some attempts, the child might move toward the candle and then move away too soon. After a while, the child would learn how to move directly to the warm area and keep their hand there until they received enough warmth.

Exploration and Exploitation

Exploration and Exploitation refer to the tradeoff between having the agent learn "too well" and learn "too poorly". Exploration is the idea that the agent should learn as much as it can by exploring the environment. Exploitation is that idea that the agent should exploit what it already knows about the environment to maximize its reward. For example, if the agent is working its way through the environment and it finds a reward of 1, then the next episode (can be thought of as a "trial") the agent will want to go back to that reward. This is productive, but what if there is another reward in the environment that has a value of 10? If the agent only exploits what it already knows about the environment it may not find the higher reward. On the other hand, if the agent only explores the environment then it will not be consistent in maximizing the total reward. This is a crucial tradeoff in RL. The agent must explore the world while also exploiting the rewards that it knows are there. The way that MFRL algorithms commonly solve this problem is with so called epsilon-greedy (ε-greedy) policy. In simple terms, this means that the agent will take the "best" action with a higher probability (set by ε). At each state there is a smaller probability that the agent will take a random action so at times the agent will "explore" the environment.

On-policy and Off-policy Reinforcement Learning

On-policy algorithms can be used only when the policy acting, and the policy being learned should be the same. Off-policy does not have to be the same in policy. To be specific, on-policy acts and learns with one policy. On the other hand, off-policy learns independently of the policy it currently acts on. This means that the policy does not stay the same throughout the learning process. In an example such as Q-learning, the policy is updated with the optimal policy for each iteration.

Mathematical Concepts

A key idea in RL is the idea that the value of a state is a measure of how "good" the state is. The value of a certain state, following a certain policy, is given by the value function

$$V^{\pi}(s) = E\left(\sum_{i\geq 0} \gamma^{i} r_{t+i}\right), \forall s \in SS.$$

In this equation *E* is the expectation value, *r* is the reward, *s* is the state, and *SS* is the state space (all possible states). Expanded out this equation gives

$$V^{\pi}(s) = E(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots), \forall s \in SS.$$

From the above equation it becomes clear that the discount factor weighs immediate and future results in the value function. The optimal value function gives the highest value for all states. Written in equation form, it looks like

$$V^{*}(s) = \max_{\pi} V^{\pi}(s), \forall s \in SS.$$

In other words, the optimal value function is given by the policy that maximizes the value function for a given state. Likewise, the optimal policy is given by

$$\pi^{*} = \operatorname{argmax}_{\pi} V^{\pi}(s), \forall s \in SS.$$

This simply means that the optimal policy is the policy that gives the optimal value function.

As mentioned above, MFRL algorithms commonly use ε-greedy policy. In terms of pseudo-code, the policy is as follows:

1. Let *r* be a random number between 0 and 1, inclusively
2. If $r > \varepsilon$, take random action, for some specified ε
3. Else, take action with best return (Q)

Moving Forward

As mentioned above, RL is a much more complex field and it would take much longer than this small activity to cover all of the aspects of it. This activity is meant to scratch the surface of the seemingly endless topic of RL.

In the next section, two MFRL algorithms will be covered. Q-learning and State-Action-Reward-State-Action (SARSA) are two such algorithms. These two will be covered because they are two of the most basic MFRL algorithms. For the sake of keeping this activity under an hour and a half, we will not be going into any deep RL concepts that involve neural networks.

Q-learning

Q-learning is an off-policy MFRL algorithm that is based on maximizing the Q-value. In this way, it learns the optimal MDP policy. It is a type of learning called Temporal Difference (TD) learning. TD learning means that the reward is calculated at each state instead of only once at the very end. The optimal value function can be written in terms of the optimal Q-function

$$V^{*}(s) = \max_{a} Q^{*}(s, a), \forall s \in SS.$$

The optimal policy can also be written this way,

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s,a)\,, \forall s \in SS.$$

This means that the optimal policy is determined from the action that yields the optimal Q-function. The main equation in Q-learning is called the Bellman equation. In terms of Q, it is as follows:

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a'),$$

In this equation, *s* is the current state, *a* is the action, *s'* is the next state, *a'* is a possible next future action. In simple terms, the *r* represents the immediate reward and the second term represents the future reward. From this equation, we can converge to the optimal Q iteratively

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha(r_{t+1} + \gamma \max_{a} Q_t(s_{t+1},a) - Q_t(s_t,a_t)).$$

In words, this means that given a current state s_t, and action a_t, find the one action *a* out of all possible actions in the next state s_{t+1} that gives the maximum Q. Pseudo-code for the Q-learning algorithm is as follows:

1. Initialize
2. For each episode
    a. Initialize *s* (starting state)
    b. Choose a_t from s_t using the policy (ε-greedy)
    c. Take action, get r_{t+1} and s_{t+1} (next state)
    d. $Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha(r_{t+1} + \gamma \max_{a} Q_t(s_{t+1},a) - Q_t(s_t,a_t))$
    e. Make $s_t = s_{t+1}$ (current state is now previous next state)
    f. If s_t is terminal state, end
    g. Else go to (2)

For simplicity of working examples by hand, we are going to simplify the update equation to

$$Q(s_t,a_t) = R(s_t,a_t) + \gamma \max_{a} Q(s_{t+1},a).$$

This means, when we are working examples by hand we will only worry about the reward for a given action in a certain state $R(s_t,a_t)$, the discount factor, and the maximum reward of the next state for all actions. This will come in handy in the next part when we work an example by hand.

Example Hand Calculation

To solve a Q-learning problem by hand, we use matrices. A reward matrix contains the reward information for all actions for all states. The rows of the matrix represent the states and the columns of the matrix represent actions (moving to a different state). For example, the picture below shows a possible "state diagram" with reward values. For this example, let's say that the discount factor is 0.9. Also, we will define state 3 as the terminal state.

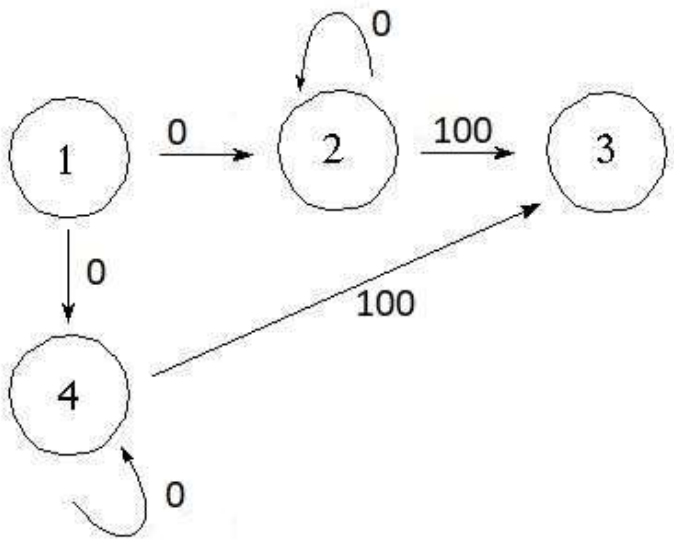


Figure 2: Example “state diagram”

In this case the reward matrix would look like

$$R = \begin{pmatrix} -1 & 0 & -1 & 0 \\ -1 & 0 & 100 & -1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & 100 & -1 \end{pmatrix}$$

As you can see, the impossible state transitions are signified with a value of negative one. The Q-matrix keeps track of the taken actions and their reward. When the problem is initialized the Q-matrix is filled with zeros

$$Q = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

As the agent works through the environment, the Q-matrix is updates. Note that the entries of the Q-matrix represent the same state-action pair that the entries in the reward matrix do. The important difference is that the upon initialization, the agent doesn’t know the rewards yet.

Assume that the starting state is state 2. We will begin by choosing the action to take in state 2. The possible actions are go to state 2 or go to state 3. Since we don’t know which one is better the choice is random whether our random number is greater than ε or not. Let’s say by random chance we choose to transition to state 3. This is where the agent will learn the reward. The update equation becomes

$$Q(2,3) = R(2,3) + 0.9 \max_a Q(3,a) = 100 + 0.9\max(Q(2,1), Q(2,2), Q(2,3)) = 100 + 0 = 100.$$

There are a few things to notice here. First, the reward for the state action pair can be looked at by the entry of the reward matrix, $R(2,3)$ is the second row and third column of the reward matrix. Also, note that there was no maximum value in the Q-matrix to account for. This is because the agent has not learned what to do in state 3 yet so the Q-values are all zero. The Q-matrix would be updated

$$Q = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

In this way, we can see that the agent has learned what action to take. The process would continue for as many episodes as specified. The next time the agent works through the environment, it would know that when it is in state 2 it should go to state 3 instead of taking a random action.

State-Action-Reward-State-Action (SARSA)

State-Action-Reward-State-Action (SARSA) is an algorithm for learning an MDP policy that is very similar to Q-learning. The only difference between SARSA and Q-learning is that SARSA is an on-policy algorithm. The update policy for SARSA is

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\big(r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)\big).$$

Notice that the policy does not change based on the what the optimal policy would be. Similarly, the pseudo-code for SARSA is the same as Q-learning except with the different update equation. Q-learning will be focused on in this activity, but it is still important to recognize the difference between Q-learning and SARSA as well as see how another MFRL algorithm functions.

Applications of Reinforcement Learning:

Game Playing

One common application of reinforcement learning is game playing. Game playing is essentially training an agent to learn how to master a certain game. In game playing, the environment would be the game engine. For simplicity, consider the game Pong. If you are unfamiliar with Pong, it is the game with a ball and two paddles, one on either side of the screen. The paddles can only move up and down and if the ball gets past a player's paddle then the opposing player would score a point. It is essentially a very basic ping-pong game. If you still have no idea what I'm talking about, an example of the Pong screen can be seen below in Figure 3.

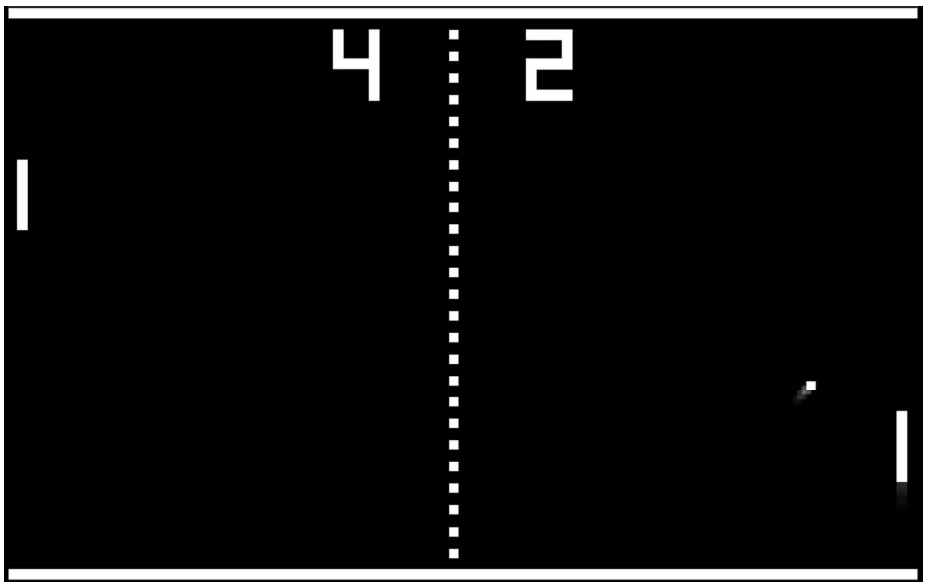


Figure 3: A Pong game in progress (from https://teachingkidstocode.io/sphero-pong/)

In this scenario, the possible actions for every state would be move up or down. The current state would be the specific frame. The reward could be based on actions taken when the ball is travelling back toward the agent's paddle. The agent would receive a positive reward when the action taken moves the paddle toward the ball and it would receive a negative reward for missing the ball. After many trials, the agent would learn how to move the paddle to hit the ball.

Robotics

Another common application of RL is in robotics. Even the simplest of robots must employ an RL algorithm with many actions per state. For example, something as simple as grabbing a block would involve many actions. In each state, the possible actions would involve moving specific joints in the fingers and arms. The reward could be based on whether the robot grabbed the block or not.

Ethical Considerations of Reinforcement Learning

Reinforcement learning with computers is connected to the reinforcement learning animals can participate in. Animals can make conscious decisions, but there still may be ethical issues when regarding reinforcement learning with technology. Reinforcement learning can be connected to industry, video games, research, and much more. When looking at the possibilities of reinforcement learning, it is important to consider ethics. As more artificial intelligence is developed, these machines will have a greater impact on the lives of humans. How these agents interact with humans has the potential to be largely dependent on reinforcement learning, which may cause ethical debate. As artificial intelligence has more capabilities how these agents are programmed should be up for debate. If reinforcement learning is used to program an "ethical" acting robot, there must be a standard for ethics. We must consider the possible negative side effects to this programming approach. Not every human on this earth has the same ethical standards. This can cause problems for different cultures around the world because our world is so culturally diverse. There must be a committee of culturally diverse people to set standards in

training the agents. There is not a single person or culture that should be given the power to decide what is a positive reward and what is a negative reward. There are many other ethical considerations that must be dealt with before reinforcement learning can be successfully implemented into society.

Warm-up Activity:

1. In RL, what is the term referring to the specific strategy that the agent uses to determine its next action based on the current state?

2. What is the term referring to how the agents weighs immediate rewards against future rewards?

3. What is the difference between model-free reinforcement learning and supervised learning?

4. What is the difference between on-policy and off-policy?

5. Classify the two demonstrated algorithms as either on-policy or off-policy.

Main Activity:

1. Suppose you are in a state *s* and there are 4 possible actions. The action rewards are as follows

$$Q(s, a_1) = 5, Q(s, a_2) = -1, Q(s, a_3) = 10, Q(s, a_4) = 0$$

   a. Using the optimal policy function, determine the optimal policy.
   b. Using the optimal value function, determine the optimal Q, Q^*.

2. Suppose you are given the following reward matrix and Q-matrix. Assume the discount factor is 0.9.

$$R = \begin{pmatrix} -1 & 100 & 0 \\ 0 & 100 & 0 \\ 0 & 100 & 0 \end{pmatrix}, Q = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 100 & 0 \end{pmatrix}.$$

   a. Which state is the terminal state?
   b. What are actions that are not allowed in this environment?
   c. Draw the "state-diagram" given the reward matrix.
   d. Based on the Q-matrix, assuming one iteration has been done, what can you say about the starting state and the action taken in that state?
   e. Why is Q(3,2) not a higher value (for example 190)?

3. You are given the following "maze" to navigate an agent through where the rewards of each state are shown. The only valid actions are to move to a state directly next to the

current state (no diagonal movement). Also, assume that the states are numbered going through row by row. So state 1 is the start, state 2 is the first row second column,…, state 8 is the third row second column, state 9 is the third row third column.

start		1
5	-10	10
2		

Assume you always start in the state labeled "start".

a. If the agent is under an exploitation only policy, what would its first action be after many episodes?
b. Can we say the same about its first action if it uses ε-greedy policy?
c. If the agent is initialized at the starting point and the discount factor is 0.9, find reward given by the update equation for transitioning to state 4. Assume that the reward for transitioning to from start to state 4 are known and the rewards for all actions in state 4 are known. *Hint:* First find the reward matrix and initialize the Q-matrix (these will be 9x9 matrices).

4. In your own words, explain how ε-greedy policy ensures a balance between exploration and exploitation. *Hint*: think about what happens as the number of episode gets large.

5. A certain environment has 5 states with transition reward values as follows: 1 to 2: 0, 1 to 3: 0, 2 to 1: 0, 2 to 4: 0, 3 to 1: 10, 3 to 4: 8, 3 to 5: 100, 4 to 2: 0, 4 to 3: 0, 4 to 5: 100, 5 to 3: 0, 5 to 4: 0, 5 to 5: 100.
    a. What state will be the termination state?
    b. Draw the corresponding "state-diagram". Clearly label all states and transition reward values.
    c. Define the reward matrix R.
    d. Assume the starting state is state 3, $\varepsilon = 0.8$, and the random number $r = 0.9$. Also, assume the Q-matrix is equal to the reward matrix. Show the iteration of the Q-learning algorithm and determine the action taken, the next state, and reward Q. For random action generation use some fair random method of your choice.

References:

GeeksforGeeks, “Basics of Reinforcement Learning,” *GeeksforGeeks*, 25-Apr-2018. [Online]. Available: https://www.geeksforgeeks.org/what-is-reinforcement-learning/. [Accessed: 08-Aug-2019].

E. Brunskill, “Stanford CS234: Reinforcement Learning | Winter 2019 | Lecture 1 - Introduction,” *YouTube*, 29-Mar-2019. [Online]. Available: https://www.youtube.com/watch?v=FgzM3zpZ55o. [Accessed: 08-Aug-2019].

S. Huang, “Introduction to Various Reinforcement Learning Algorithms. Part I (Q-Learning, SARSA, DQN, DDPG),” *Medium*, 16-Sep-2018. [Online]. Available: https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287. [Accessed: 08-Aug-2019].

S. Zychlinski,“The Complete Reinforcement Learning Dictionary,” *Medium*, 28-Feb-2019. [Online]. Available: https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e. [Accessed: 08-Aug-2019].

UMass, “Glossary of Terminology in Reinforcement Learning”. [Online]. Available: http://www-anw.cs.umass.edu/rlr/terms.html. [Accessed: 08-Aug-2019].

UNSW Sydney, “Reinforcement Learning,” *Reinforcement Learning - Algorithms*. [Online]. Available: https://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html. [Accessed: 08-Aug-2019].

Mathworks, “rlQAgentOptions,” *Q-Learning Agents - MATLAB & Simulink*. [Online]. Available: https://www.mathworks.com/help/reinforcement-learning/ug/q-agents.html. [Accessed: 08-Aug-2019].

R. Sutton, A. Barto “PSYCH 209: Neural Network Models of Cognition: Principles and Applications,” *Stanford University*. [Online]. Available: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf. [Accessed: 08-Aug-2019].

A. Violante, “Simple Reinforcement Learning: Q-learning,” *Medium*, 01-Jul-2019. [Online]. Available: https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56. [Accessed: 08-Aug-2019].

F. Woergoetter and B. Porr, “Reinforcement learning,” *Scholarpedia*. [Online]. Available: http://www.scholarpedia.org/article/Reinforcement_learning#Background_and_History. [Accessed: 08-Aug-2019].

“Q-Learning .*;” *A Painless Q-Learning Tutorial*. [Online]. Available: http://mnemstudio.org/path-finding-q-learning-tutorial.htm. [Accessed: 08-Aug-2019].

S. Ravichandiran, “Hands-On Reinforcement Learning with Python,” *O'Reilly | Safari*. [Online]. Available: https://www.oreilly.com/library/view/hands-on-reinforcement-learning/9781788836524/0c14fb24-1926-4cc3-8bf6-818cae23bde2.xhtml. [Accessed: 08-Aug-2019].

Appendix:

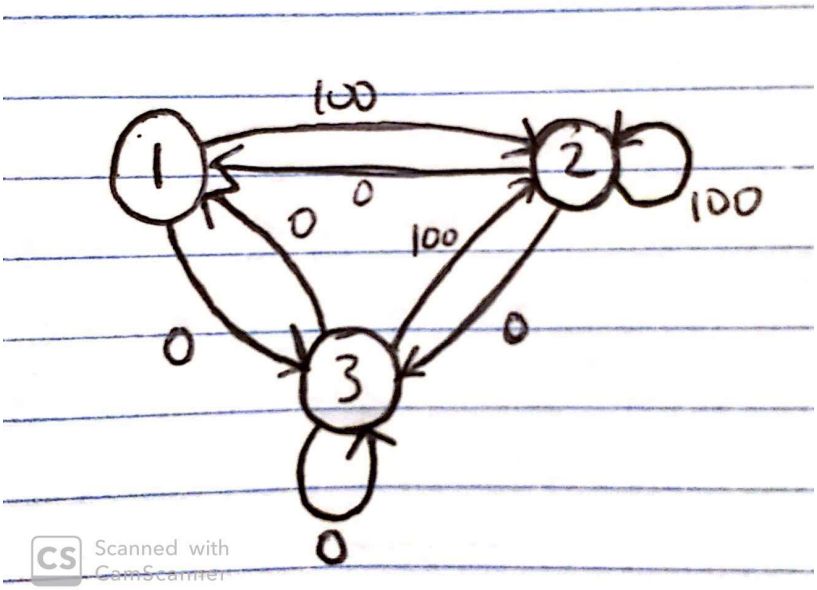
Warm-up Activity Solutions

1. Policy
2. Discount Factor
3. MFRL does not have training data but instead uses iterative methods to maximize reward through trial and error
4. On-policy uses the same policy, off-policy uses different policies
5. On-policy: SARSA, off-policy: Q-learning

Main Activity Solutions

1. a) From the optimal policy function, the optimal Q-value is obtained by taking action a_3 in state s
   b) From the optimal value function, the optimal Q-value (highest value) is 10

2. a) The terminal state is state 2
   b) The transition from state 1 to state 1 is not allowed
   c)



d) From the Q-matrix we can see that on the first iteration, the agent started in state 3 and randomly chose to go to state 2 where it observed the reward of 100.
e) The reward is not higher because even though the action that yields the highest reward in state 2 is the action that stays in state 2, the agent has not learned this yet so it only knows the reward that it observed.

3. a) After many iterations its first action would be to go one step down because it sees the immediate reward which it will always go for
   b) No, if it uses ε-greedy policy it would have some chance to take a random action so it would have the possibility to explore the environment
   c)

$$R = \begin{pmatrix} -1 & 0 & -1 & 5 & -1 & -1 & -1 & -1 & -1 \\ 0 & -1 & 1 & -1 & -10 & -1 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 & -1 & 10 & -1 & -1 & -1 \\ 0 & -1 & -1 & -1 & -10 & -1 & 2 & -1 & -1 \\ -1 & 0 & -1 & 5 & -1 & 10 & -1 & 0 & -1 \\ -1 & -1 & 1 & -1 & -10 & -1 & -1 & -1 & 0 \\ -1 & -1 & -1 & 5 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & -1 & -10 & -1 & 2 & -1 & 0 \\ -1 & -1 & -1 & -1 & -1 & 10 & -1 & 0 & -1 \end{pmatrix}$$

   It was specified that the agent will transition from state 1 to state 4. The update equation would be

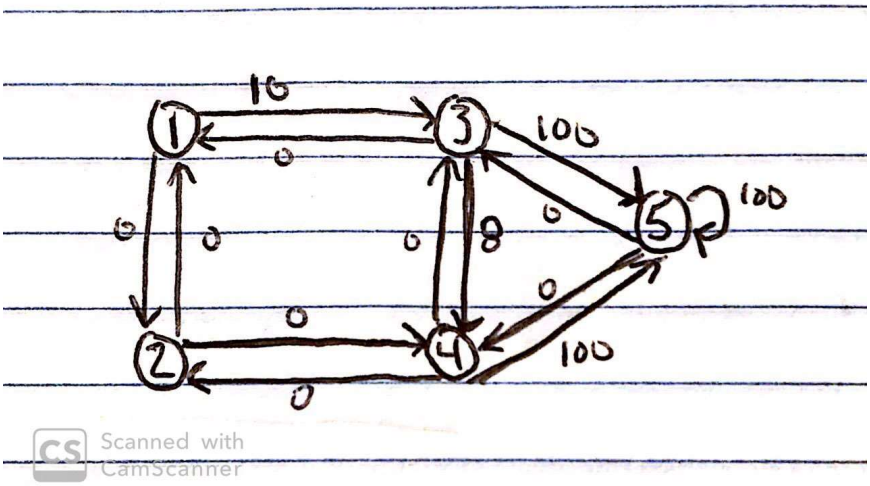
$$Q(1,4) = R(1,4) + 0.9 \max_a Q(4,a)$$

   This gives

$$5 + 0.9 \max\big(Q(4,1), Q(4,5), Q(4,7)\big) = 5 + 0.9 \max(0, -10, 2) = 5 + 0.9(2) = 6.8$$

4. The ε-greedy policy ensures that the agent will not always exploit what it knows about the environment. As the number of episodes gets larger the chance that the agent will take random actions gets larger so over a longer period of time the agent will be able to exploit what it knows about the environment while still exploring. At any given state there is a small chance that the agent takes a random action so if the total number of actions taken is large then the agent will exploit what it knows about the environment most of the time but as it is exploiting it will take some random actions as well so it can better explore the environment.

5. a) State 5 will be the termination state
   b)



NOTE: THE TRANSITION FROM 1 TO 3 AND THE TRANSITION FROM 3 TO 1 ARE FLIPPED AROUND, 3 TO 1 SHOULD BE 10 AND 1 TO 3 SHOULD BE 0

c)

$$R = \begin{pmatrix} -1 & 0 & 0 & -1 & -1 \\ 0 & -1 & -1 & 0 & -1 \\ 10 & -1 & -1 & 8 & 100 \\ -1 & 0 & 0 & -1 & 100 \\ -1 & -1 & 0 & 0 & 100 \end{pmatrix}$$

d) There are three possible transitions. Depending on how you determined your action, the agent could either transition from state 3 to state 1, from state 3 to state 4, or from state 3 to state 5. All updates will be shown below.

From state 3 to 1:

$$Q(3,1) = R(3,1) + 0.9 \max_a\big(Q(1,3), Q(1,2)\big) = 10 + 0.9(0) = 10$$

From 3 to 4:

$$Q(3,4) = R(3,4) + 0.9 \max_a(Q(\,4,2), Q(4,3), Q(4,5)) = 8 + 0.9\max\,(0,0,100) = 98$$

From 3 to 5:

$$Q(3,5) = R(3,5) + 0.9 \max_a\big(Q(5,3), Q(5,4), Q(5,5)\big) = 100 + 0.9\max(0,0,100)$$
$$= 190$$