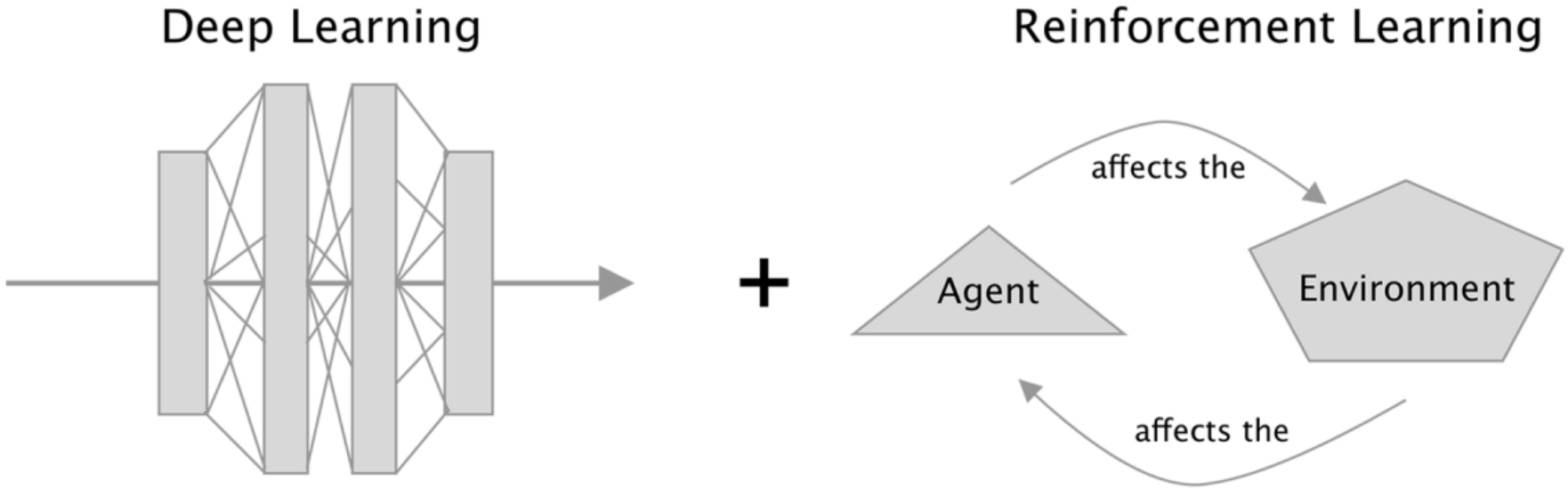
ECE 457A TUTORIAL 11: REINFORCEMENT LEARNING

27-Nov-2023

Danial Sadrian Zadeh,
Department of Electrical and Computer Engineering

UNIVERSITY OF WATERLOO | FACULTY OF ENGINEERING

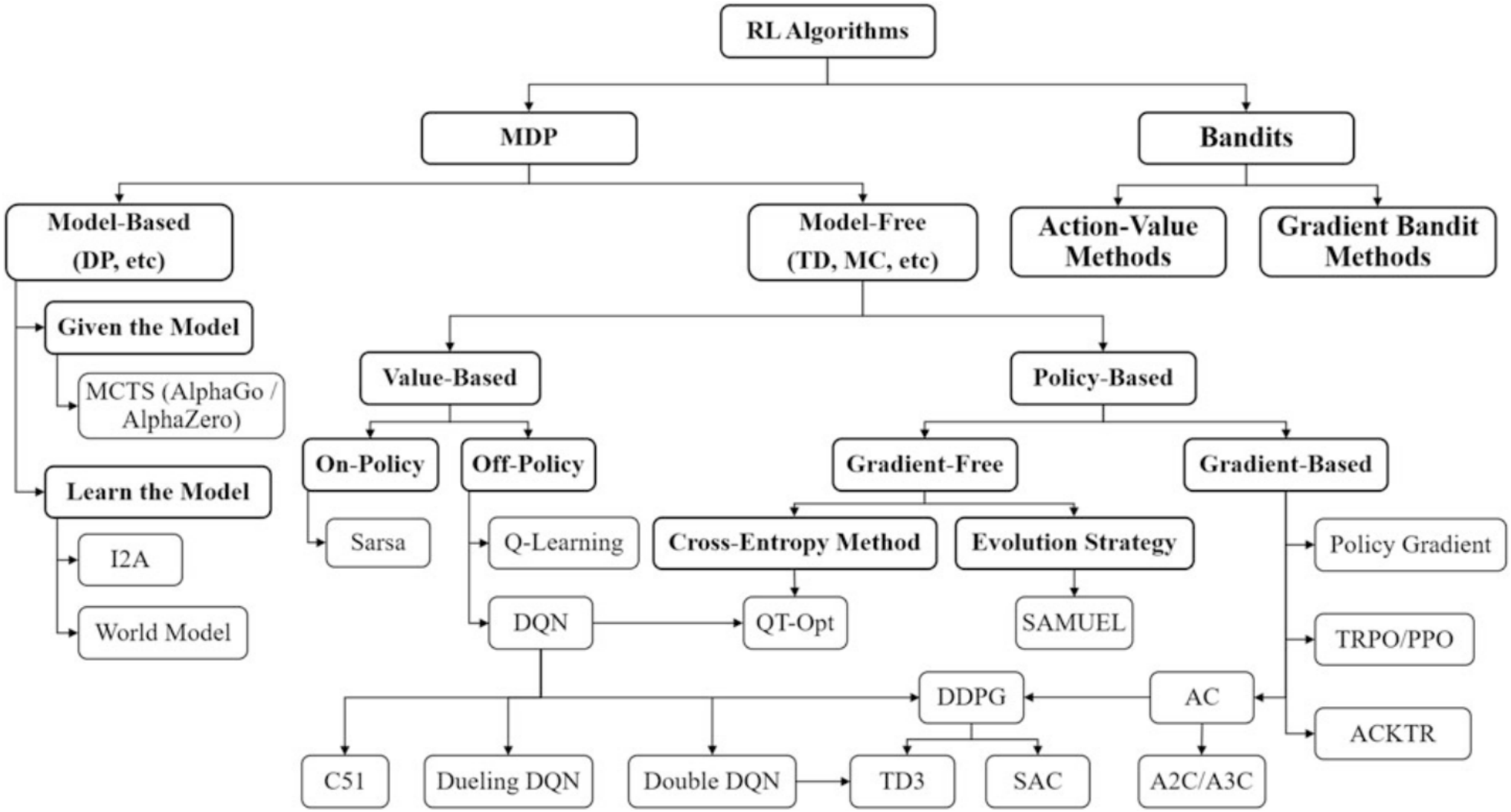
Introduction to Deep Reinforcement Learning (DRL)



=

DEEP REINFORCEMENT LEARNING

Summary of RL Algorithms



Deep Q-Network (DQN)

- In this tutorial, we only focus of Deep Q-Network (DQN) algorithm.
  - Q-learning can struggle with situations when the count of the observable set of states is very large.
  - Q-learning uses a table to represent the state-action values (Q-values). Each entry in the table corresponds to the expected cumulative reward of taking a particular action in a particular state.
  - DQN algorithm is a model-free, online, off-policy reinforcement learning method.
  - It employs a neural network (NN) to approximate Q-values rather than using a table. This allows DQN to handle high-dimensional state spaces, making it applicable to a broader range of problems.

Deep Q-Network (DQN)

- In this tutorial, we only focus of Deep Q-Network (DQN) algorithm.
  - DQN can handle both discrete and continuous observation spaces, making it more versatile than traditional Q-learning.
  - The NN plays the role of Q function.

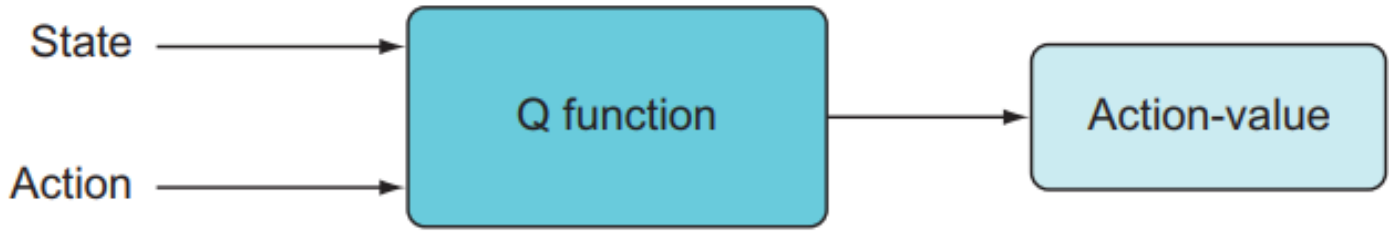


Figure 3.2 The Q function could be any function that accepts a state and action and returns the value (expected rewards) of taking that action given that state.

Deep Q-Network (DQN)

- In this tutorial, we only focus of Deep Q-Network (DQN) algorithm.
  - Why → Universal Approximation Theorem
    - A feedforward NN with a single hidden layer containing a sufficient number of neurons (nodes) can approximate any continuous function on a compact subset of the input space to arbitrary accuracy. This means that, in theory, an NN with the right architecture and a large enough number of neurons can approximate any function.

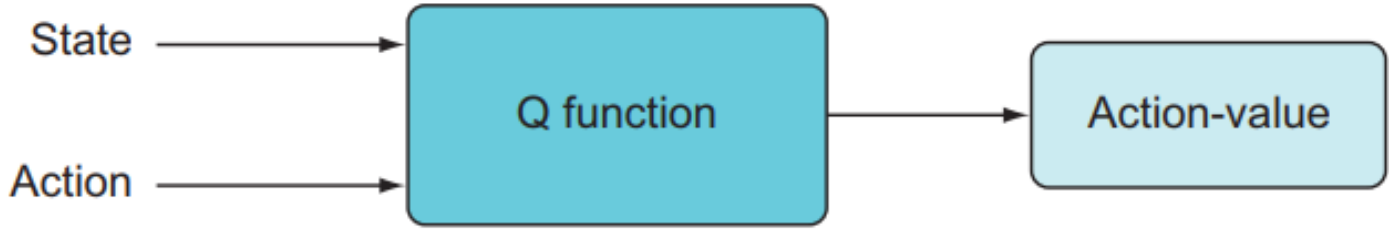


Figure 3.2 The Q function could be any function that accepts a state and action and returns the value (expected rewards) of taking that action given that state.

DQN Algorithm

- We train a function approximator, such as an NN with parameters θ, to estimate the Q-values. This can be done by minimizing the following loss function at each step i.

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \rho(.)} \left[(y_i - Q(s, a; \theta_i))^2 \right] \text{ where } y_i = r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})$$

- Here, y_i is called the TD target, and $y_i - Q$ is called the TD error. ρ represents the behavior distribution, the distribution over transitions $\{s, a, r, s'\}$ collected from the environment.

DQN Algorithm

1. We set up a `for` loop for the number of epochs.
2. In the loop, we set up a `while` loop (while the game is in progress).
3. We run the Q-network forward.
4. We're using an epsilon-greedy implementation, so at time t with probability ε we will choose a random action. With probability $1 - \varepsilon$, we will choose the action associated with the highest Q value from our neural network.
5. Take action a as determined in the preceding step, and observe the new state s′ and reward r_{t+1}.
6. Run the network forward using s′. Store the highest Q value, which we'll call max Q.
7. Our target value for training the network is $r_{t+1} + \gamma * \max Q_A(S_{t+1})$, where γ (gamma) is a parameter between 0 and 1. If after taking action a_t the game is over, there is no legitimate s_{t+1}, so $\gamma * \max Q_A(S_{t+1})$ is not valid and we can set it to 0. The target becomes just r_{t+1}.
8. Given that we have four outputs and we only want to update (i.e., train) the output associated with the action we just took, our target output vector is the same as the output vector from the first run, except we change the one output associated with our action to the result we computed using the Q-learning formula.
9. Train the model on this one sample. Then repeat steps 2–9.

Gridworld Problem

- Let us apply DQN to the Gridworld problem.
- Goal
  - Train an NN to play the game from scratch.
- Design
  - Any move that does not win the game receives a reward of -1.
  - The winning move receives a reward of +10.
  - The losing move receives a reward of -10.

Gridworld Problem

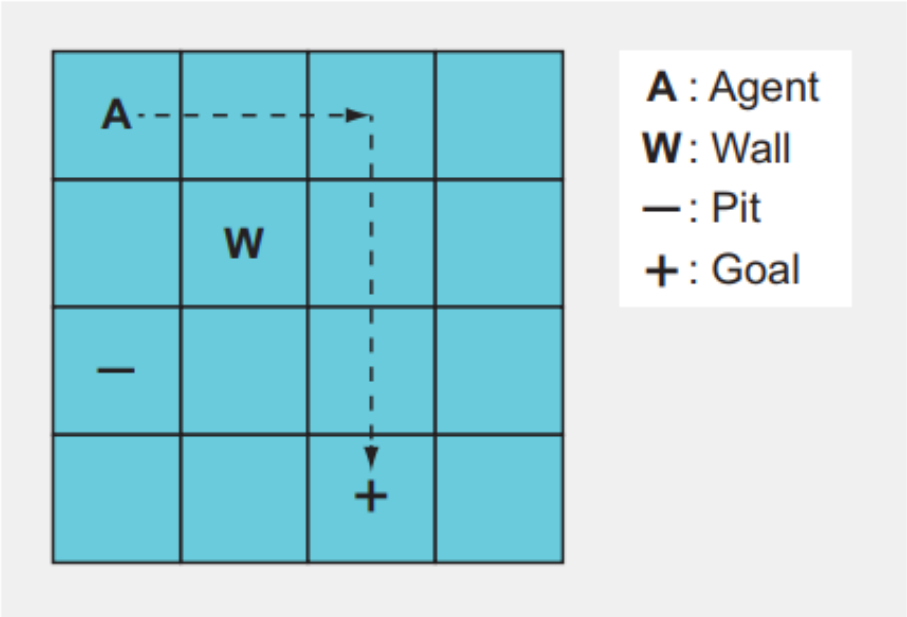


Figure 3.1 This is a simple Gridworld game setup. The agent (A) must navigate along the shortest path to the goal tile (+) and avoid falling into the pit (–).

Gridworld Problem

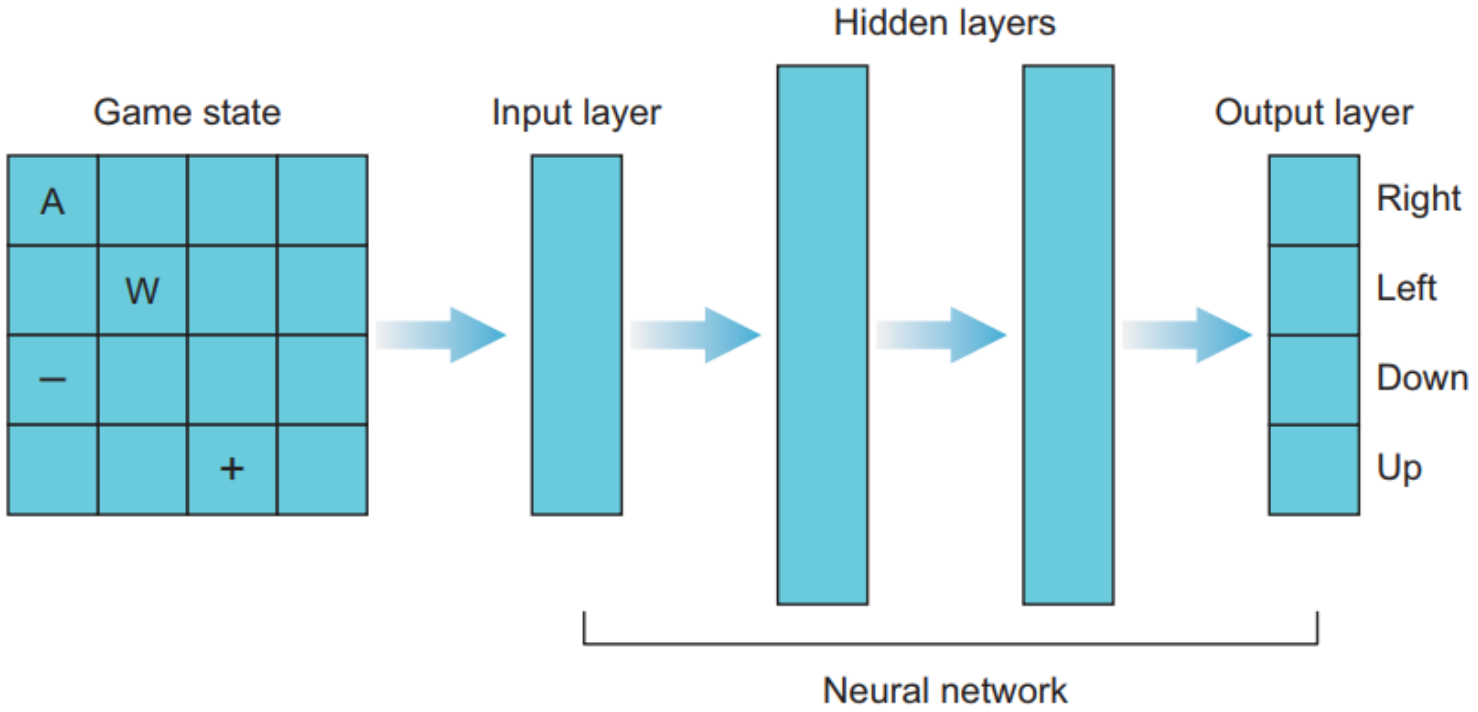


Figure 3.8 The neural network model we will use to play Gridworld. The model has an input layer that can accept a 64-length game state vector, some hidden layers (we use one, but two are depicted for generality), and an output layer that produces a 4-length vector of Q values for each action, given the state.

References

1. B. Brown and A. Zai, Deep Reinforcement Learning in Action. Manning Publication Co., 2020. Codes: https://github.com/DeepReinforcementLearning/DeepReinforcementLearningInAction
2. H. Dong, Z. Ding, and S. Zhang, Deep Reinforcement Learning. Springer Nature, 2020.
3. “Introduction to RL and Deep Q Networks,” TensorFlow, Sep. 26, 2023. [Online]. Available: https://www.tensorflow.org/agents/tutorials/0_intro_rl