Reinforcement Learning for traffic light control

Abhijit Mishra
Department of Information Technology,
IIIT Sonepat
Sonepat, India
abhijitmishra704@gmail.com

Abstract- **Inefficient traffic light control systems lead to congestion on roads causing various problems. In this paper, we propose a reinforcement learning method for controlling traffic lights in a wide range of scenarios. We create a crossroad environment whose variables can be altered to create multiple settings. We use the Q-learning algorithm to train our RL agent. We test our method in crossroad environments for moderate and low traffic congestion scenarios.**

Keywords- **Reinforcement learning, Q-learning, Traffic light control.**

I. INTRODUCTION

Traffic signal controllers generally work on electro-mechanical controllers that use dial timers with fixed, signalized intersection time plans. Recently more sophisticated systems are being used that use coordinated control. These systems still are inept in dealing with different scenarios encountered at crossroads. Congestion on urban roads has an adverse effect on urban economies. It also leads to an increase in trip delays and is fatal in emergency scenarios.

Reinforcement learning is an area of machine learning where an agent takes actions in an environment and receives rewards for desired behaviors and penalties for undesired ones. Reinforcement learning is being actively used to find solutions to the current traffic problems [1]. Traffic simulation environments such as SUMO(Simulation of Urban MObility) and CityFlow [2] have been built to train reinforcement learning models for traffic control. Deep RL models like IntelliLight [3] have been built for traffic light control.

In this paper, we present a simple crossroad environment whose parameters can be regulated to create a wide range of traffic scenarios. Further, we use the Q-learning [4] algorithm to train an RL agent in our crossroad environment. We train our agent in our simulated environment as it is difficult to collect and work on real-time traffic data for different scenarios. We also provide a method to implement our agent in real-life settings. Although our agent does not seem to discover the desired policy further analysis may lead to a better policy.

II. CROSSROAD ENVIRONMENT

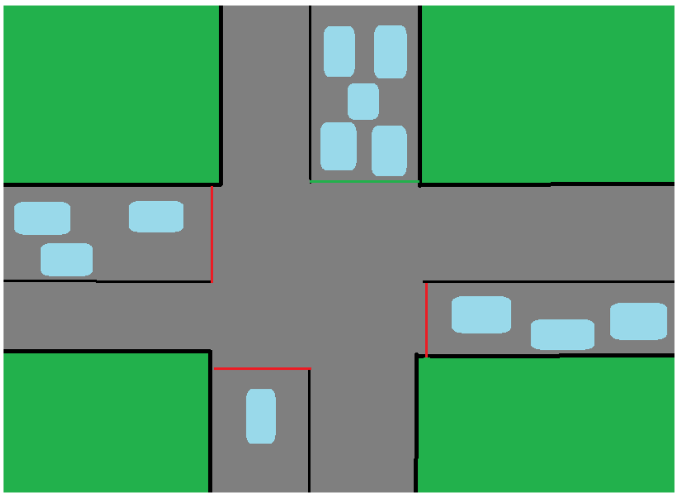


Fig. 1. Representation of Crossroad Environment

The crossroad environment has been built in accordance with OpenAI's gym class [5]. The environment is an intersection of four lanes with the state being the amount of traffic on each of the lanes. Traffic on each lane is a continuous observation space with a float value between 0 and 1 with 0 being an empty road and 1 being jam-packed with traffic. The environment's action space is discrete with 0 representing red light and 1 representing green light. To make the learning task episodic we take twenty steps in the environment.

A. REWARD

The reward is calculated at every step based on the agent's actions. Different rewards were tried to set up the optimal environment. Finally, the optimal reward was comprised of two attributes-

1. The difference between the current state and the future state. This accounts for the reward for the decrease in traffic caused by the agent's action.

   Reward $=\Sigma$(current_state-future_state) (1)

2. Penalty for causing heavy traffic on the crossroad. This was only applied in situations when the action included more than one green light and the sum of traffic on the lanes where green lights were shown was greater than the specified threshold. In our case, we defined the threshold to be 0.7 i.e. give a penalty if the sum of traffic on the lanes with green lights is greater than 0.7. A penalty of 5 units was applied.

   reward=reward-5 $\forall$ (Σ(action)>1 & Σ(action*current_state)>0.7) (2)

B.NEXT STATE

The next state is computed by taking into account the decrease in traffic based on the agent's actions and the increase in traffic because of new vehicles arriving. The amount of traffic increase and decrease is a hyper parameter that we can alter to create different scenarios. For new vehicles, we choose a random number between a range from a uniform distribution to increase the traffic in each direction. In our case, we decrease the traffic on lanes with a green light by 0.8

future_state[i] = future_state[i]-0.8 $\forall$ ((action[i]=1) & (current_state[i]>0.8)), i$\in${1,2,3,4}

future_state[i]= 0 $\forall$ ((action[i]=1) & (current_state[i]<=0.8)), i$\in${1,2,3,4} (3)

future_state[i]= future_state[i]+random.uniform(0.0, 0.1) $\forall$ i$\in${1,2,3,4}

future_state[i]=1 $\forall$ (future_state[i]>1),i$\in${1,2,3,4} (4)

III. THE AGENT

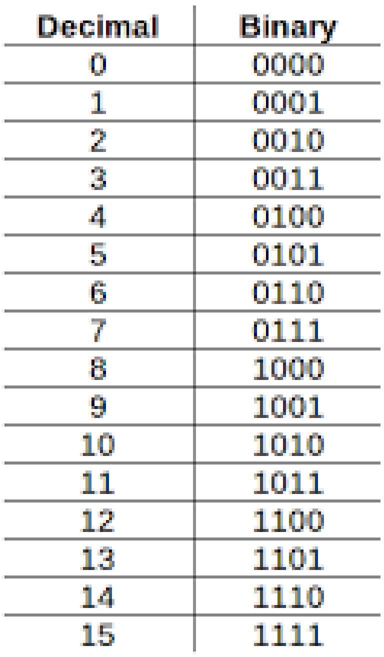
The agent decides which action to take based on the observations given from the environment. Thus the agent tries to find an optimal policy for a given environment.

A. VALUE FUNCTION

Nearly all RL algorithms are based on estimating value functions. Value functions give the expected return if you start in that state or state-action pair, and then act according to a particular policy henceforth.

The value function is a dense neural network that takes the environment's state(4*1 vector) as input. The input goes through a number of dense layers with rectified linear activation functions. In some networks we concatenate the output of these layers with the original input and pass the concatenated output through a dense layer with 16 units. Thus the output is a vector of 16 numbers, each number denoting the expected return for signalling 16 different combinations of signals.

The actions are a 4*1 vector with each element either 0 or 1. Therefore all the permutations of the actions can be defined as a 4 bit binary number which can be further represented as 16 integers.



Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Fig.2. Decimal to binary

Thus to find out the expected return for signalling green light in lane 2 and 3 and red in lane 1 and 4 we want the value corresponding to 0110 which is the 6th element of the value function. This is how the state action values were calculated. These values can then be used to train the agent by the Q-learning algorithm.

We use simple networks as well as network with skip connections to test our model.

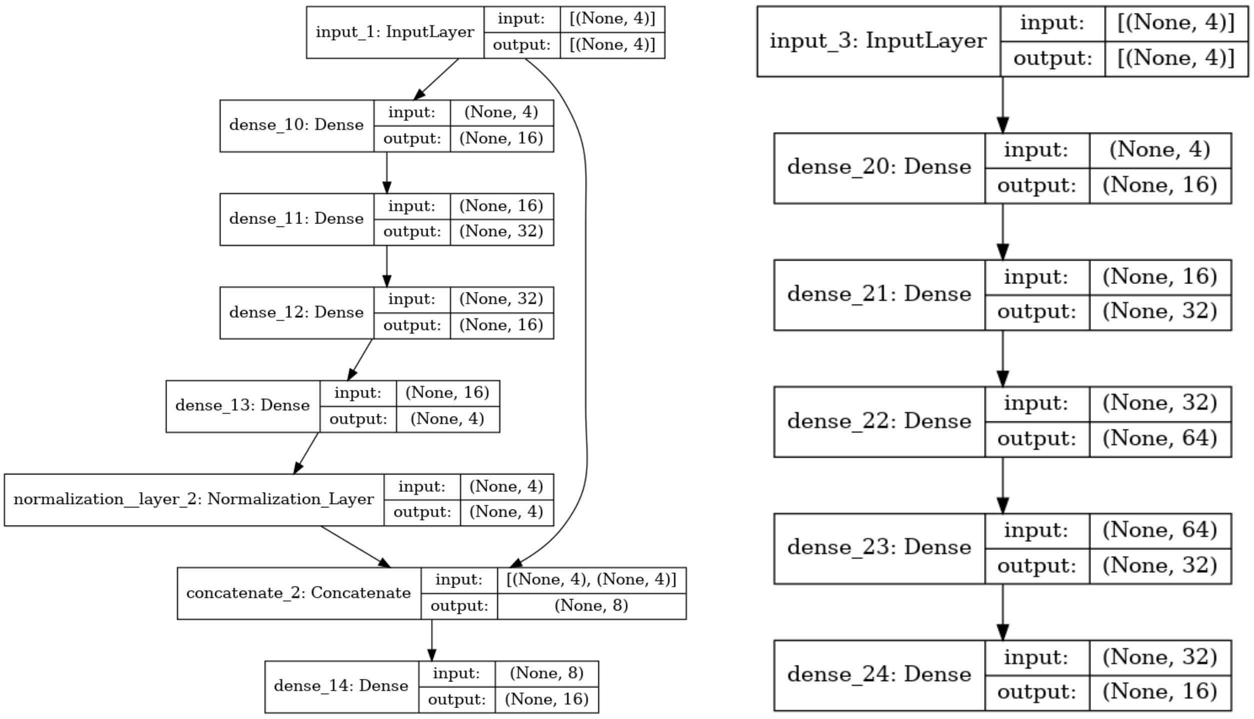

input_1: InputLayer | input: [(None, 4)] | output: [(None, 4)]
dense_10: Dense | input: (None, 4) | output: (None, 16)
dense_11: Dense | input: (None, 16) | output: (None, 32)
dense_12: Dense | input: (None, 32) | output: (None, 16)
dense_13: Dense | input: (None, 16) | output: (None, 4)
normalization__layer_2: Normalization_Layer | input: (None, 4) | output: (None, 4)
concatenate_2: Concatenate | input: [(None, 4), (None, 4)] | output: (None, 8)
dense_14: Dense | input: (None, 8) | output: (None, 16)
input_3: InputLayer | input: [(None, 4)] | output: [(None, 4)]
dense_20: Dense | input: (None, 4) | output: (None, 16)
dense_21: Dense | input: (None, 16) | output: (None, 32)
dense_22: Dense | input: (None, 32) | output: (None, 64)
dense_23: Dense | input: (None, 64) | output: (None, 32)
dense_24: Dense | input: (None, 32) | output: (None, 16)

Fig.3. Value Function Networks

B. THE POLICY

The policy used is epsilon greedy. We use this policy to balance the exploration exploitation tradeoff. The policy explores with ε probability(chooses a random action), and exploits for most of the time(chooses the best action i.e. action with highest expected return).

C. TRAINING

The agent is trained via Q-learning algorithm for function approximation case [6]. Q-learning is a is a values-based model-free reinforcement learning algorithm. For stable training, we use experience replay [7] and a target buffer.

To stop action values from oscillating or diverging when using a large buffer of all our past experience we use a replay buffer. The replay buffer consists of a collection of experience tuples (State, Action, Reward, Next State, done). These tuples are added to the buffer as we gain experience from the environment. As new tuples get added the older ones are pushed out. We take a mini-batch randomly from this tuple to train our agent.

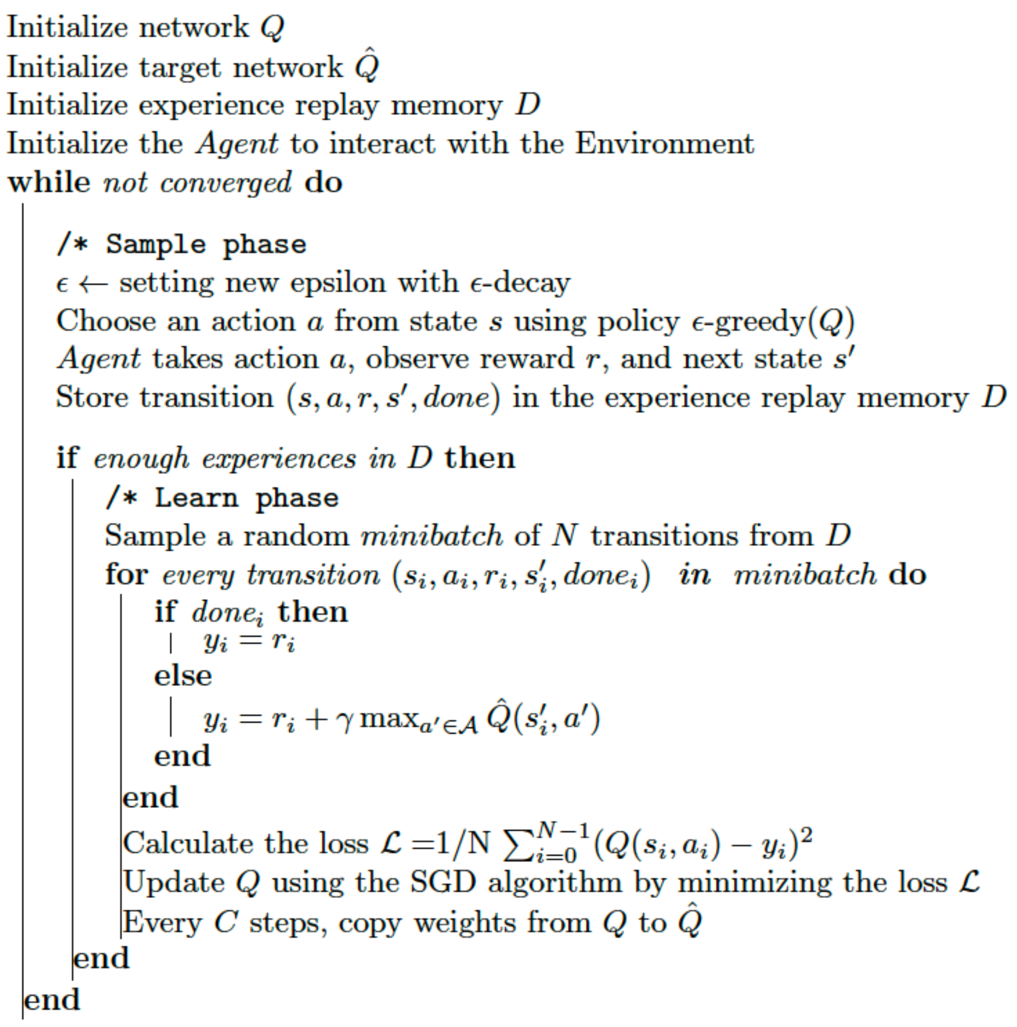
As in Q-Learning we update a guess with a guess. This process could adversely effect our training by forming harmful correlations. Thus to prevent this we create a copy of our neural network called the target network. The values of this network are used in backpropagation to update the main network. The target network is then synchronized periodically with the main network.

The Q-learning algorithm used is-

$$L(\theta^{(i)})=((reward+\gamma *max_{a'}(Q(s',a'))-Q(s,a))^{2})/2 \tag{5}$$

$$\theta^{(i+1)}=\theta^{(i)}-\alpha\nabla L(\theta^{(i)}) \tag{6}$$

Here $L(\theta^{(i)})$ is the loss for our optimization problem, and $\theta^{(i)}$ are the weights of the value function network. In our case we use the Adam optimization.



```
Initialize network $Q$
Initialize target network $\hat{Q}$
Initialize experience replay memory $D$
Initialize the $Agent$ to interact with the Environment
while $not\ converged$ do
    /* Sample phase
    $\epsilon \leftarrow$ setting new epsilon with $\epsilon$-decay
    Choose an action $a$ from state $s$ using policy $\epsilon$-greedy($Q$)
    $Agent$ takes action $a$, observe reward $r$, and next state $s'$
    Store transition $(s, a, r, s', done)$ in the experience replay memory $D$

    if $enough\ experiences\ in\ D$ then
        /* Learn phase
        Sample a random $minibatch$ of $N$ transitions from $D$
        for $every\ transition$ $(s_i, a_i, r_i, s'_i, done_i)$ $in$ $minibatch$ do
            if $done_i$ then
                $y_i = r_i$
            else
                $y_i = r_i + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s'_i, a')$
            end
        end
        Calculate the loss $\mathcal{L}$ =1/N $\sum_{i=0}^{N-1}(Q(s_i, a_i) - y_i)^2$
        Update $Q$ using the SGD algorithm by minimizing the loss $\mathcal{L}$
        Every $C$ steps, copy weights from $Q$ to $\hat{Q}$
    end
end
```

Fig.4. Pseudocode for the Q-learning algorithm using experience replay and target buffer
Image Source- Jordi Torres, "Deep Q-Network (DQN)-II", Aug 16, 2020.
Accessed via- https://miro.medium.com/max/875/1*lSMJQIIYY7pEeC-fTSHf3Q.png

IV EXPERIMENTS

The experiments were performed in two environments one for low traffic and the other for medium traffic congestion scenarios. To view, all the different experiments go to- Traffic light control environment results
Note- Sum of rewards in figures is batch sum.

A. FOR LOW TRAFFIC CONGESTION ENVIRONMENT-

State at start of episode ∈ $[0.1, 0.3]$
New vehicles in each lane at each step ∈ $[0.0, 0.1]$

Network Architecture-
Input->Dense(16)->Dense(32)->Dense(16)->Dense(4)->Normalize->Concatenate(Input,Normalize)->Dense(16)

Epsilon=0.2

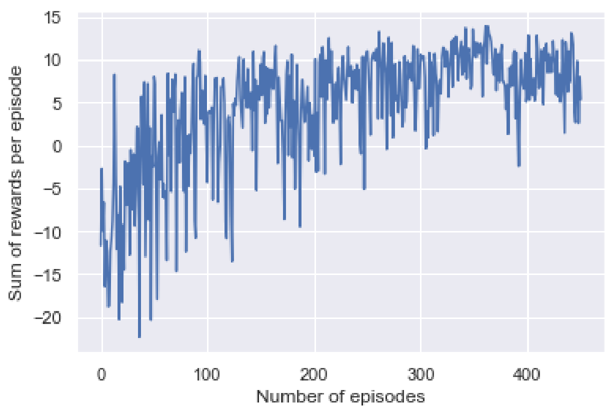


Fig.5. List of rewards for the given architecture

Network Architecture-
Input->Dense(16)->Dense(32)->Dense(64)->Dense(32)->Dense(16)

Epsilon=0.2

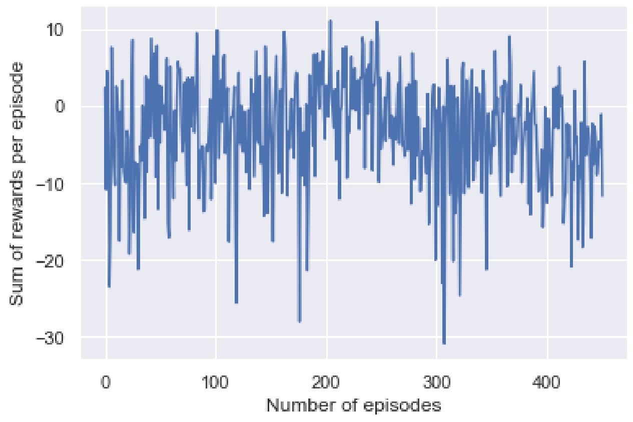


Fig.6. List of rewards for the above architecture

Epsilon=0.1

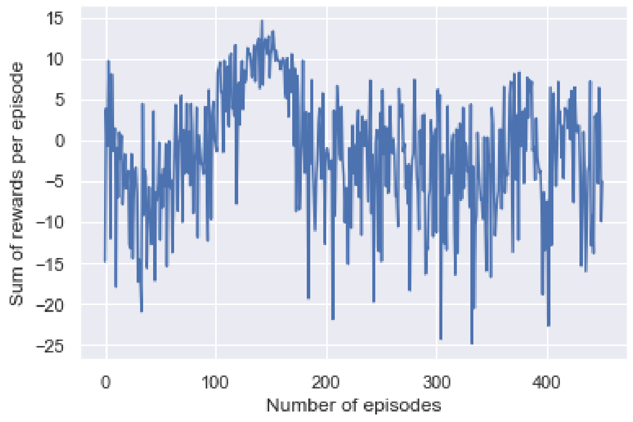


Fig.7. List of rewards for the above architecture

B. FOR MEDIUM TRAFFIC CONGESTION ENVIRONMENT-

State at start of episode $\in [0.4, 0.6]$
New vehicles in each lane at each step $\in [0.0, 0.2]$

Network Architecture-
Input->Dense(16)->Dense(32)->Dense(16)->Dense(4)->Normalize->Concatenate(Input,Normalize)->Dense(16)

Epsilon=0.1

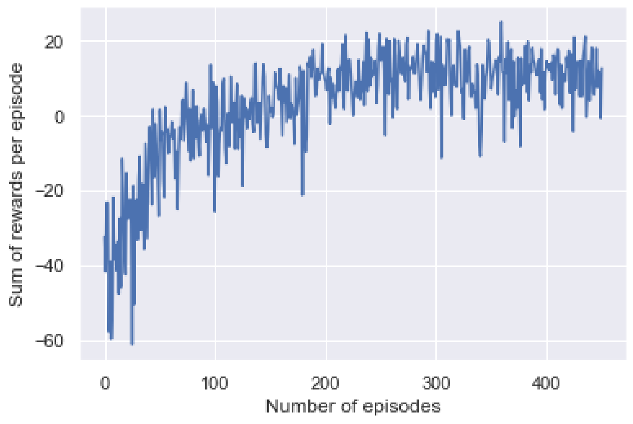


Fig.8. List of rewards for the above architecture

Network Architecture-
Input->Dense(16)->Dense(32)->Dense(64)->Dense(32)->Dense(16)

reward=action // reward for signalling green light

reward=reward-15 ∀ ((action)>1 & (action*current_state)>0.7) // penalty for high traffic on crossroad

reward-=(np.sum(self.state)/2) // penalty for sum of future states

Epsilon=0.1

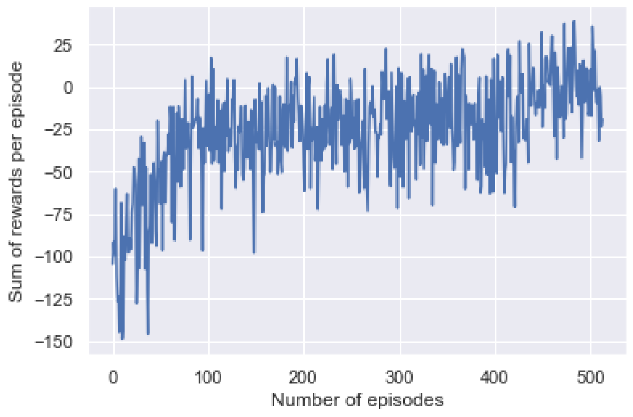


Fig.9. List of rewards for the above architecture

In most of the cases, the rewards increase and then stagnate at some value. Thus the learning stops and the model isn't able to learn the optimal policy for our environment.

V CONCLUSION

In this paper, we approached the problem of traffic signal control from a reinforcement learning method. We built a crossroad environment that could be used for a wide range of scenarios. We trained several models based on the Q-learning algorithm. Although we could not find optimal policies further investigation may lead to better results. Our project had a few limitations like the crossroad environment had a single intersection compared to the real-world road network. All the training and testing were done in a simulated environment which is way simpler compared to the real world scenario. In the future, we could address these issues.

REFERENCES

[1] Z. Li, C. Xu, and G. Zhang, "A Deep Reinforcement Learning Approach for Traffic Signal Control Optimization," *arXiv*, 2021, doi: 10.48550/arxiv.2107.06115.

[2] H. Zhang *et al.*, "CityFlow: A Multi-Agent Reinforcement Learning Environment for Large Scale City Traffic Scenario," in *The World Wide Web Conference on - WWW '19*, New York, New York, USA, May 2019, pp. 3620–3624, doi: 10.1145/3308558.3314139.

[3] H. Wei, G. Zheng, H. Yao, and Z. Li, "Intellilight: A reinforcement learning approach for intelligent traffic light control," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD '18*, New York, New York, USA, Aug. 2018, pp. 2496–2505, doi: 10.1145/3219819.3220096.

[4] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-Learning Algorithms: A Comprehensive Classification and Applications," *IEEE Access*, vol. 7, pp. 133653–133667, 2019, doi: 10.1109/ACCESS.2019.2941229.

[5] "Environment Creation - Gym Documentation." https://www.gymlibrary.ml/content/environment_creation/ (accessed May 16, 2022).

[6] "Going Deeper Into Reinforcement Learning: Understanding Q-Learning and Linear Function Approximation." https://danieltakeshi.github.io/2016/10/31/going-deeper-into-reinforcement-learning-understanding-q-learning-and-linear-function-approximation/ (accessed May 16, 2022).

[7] Long-Ji Lin, "Reinforcement learning for robots using neural networks | Guide books." https://dl.acm.org/doi/10.5555/168871 (accessed May 16, 2022).