Reinforcement Learning Assignment 2 - Group 02

Divesh Manoj Bakshani
University at Buffalo
Buffalo, NY, 14215
`diveshma@buffalo.edu`

Vinay Kudari
University at Buffalo
Buffalo, NY, 14215
`vkudari@buffalo.edu`

Abstract

Making a Reinforcement Learning Environment and running a random agent on it for the purpose of learning OpenAI Gym Environments.

1 Description of Environments

There are mainly 2 environments that the project has been made in:

1. Cartpole-v1
2. Determinisic Maze environment

1.1 Cartpole-v1

The Cartpole -v1 environment was derived from the original Cartpole-v0 environment. There are 2 key differences in the 2 environments while keeping the original base code same, they are:

1. max_episode_steps=500 from 200
2. reward_threshold=475.0 from 195.0

Other than that the definition of the Cartpole-v1 as defined by OpenAI Gym (`https://gym.openai.com/envs/CartPole-v1/`) is:

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.

1.2 Deterministic Maze Solver Environment

The goal of the environment is to Solve maze by reaching goal state optimally avoiding forbidden states. They include out-of-bounds positions, walls and already visited states.

Actions are Up, Down, Left and Right

Agent position tuple (x, y) determines the current state. Environment with 16 states has been chosen for convenience but can be easily modified by passing a larger maze to GridEnv class. States with value of 1.0 are legal positions in which our agent can move, whilst others with a value of 0.0 are walls that our agent should avoid

Initially rewards were given according to the euclidean distance between agent and the goal, but policy was not able to converge in a limited time. Later, new dynamics were chosen and they seem to work well for the current task. The given pairs represent the appropriate reward dynamics, with key representing the current state type and value being the reward

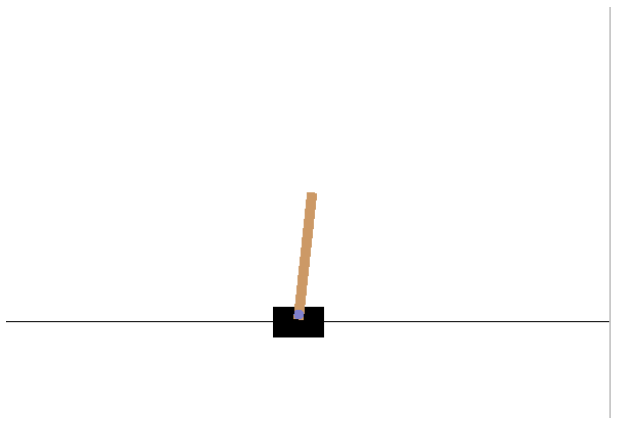


Figure 1: Cartpole-v1 Environment

1. Goal: +20.0
2. Wall: -5.0
3. Out of bounds: -0.75
4. Visited: -0.85
5. Legal Move: -0.05

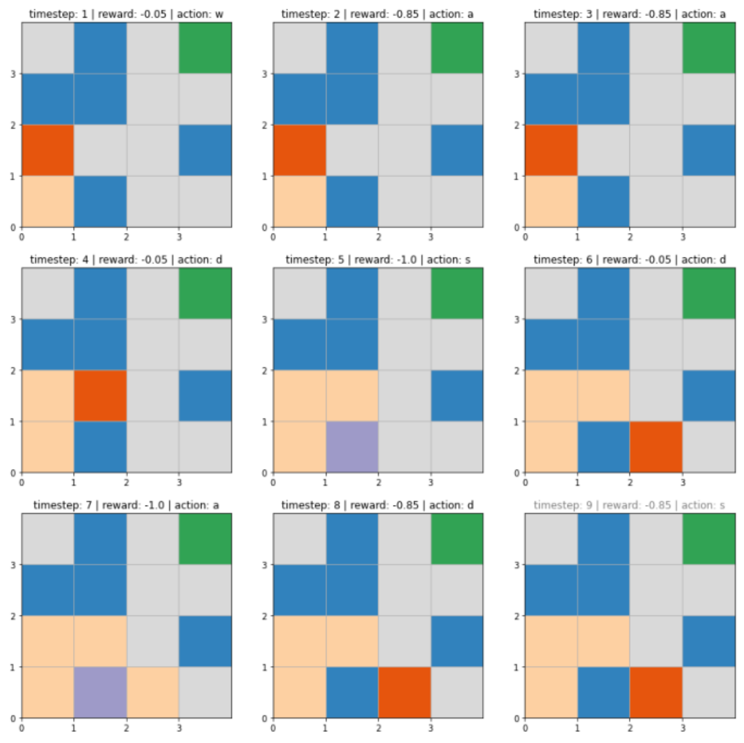

timestep: 1 | reward: -0.05 | action: w
timestep: 2 | reward: -0.85 | action: a
timestep: 3 | reward: -0.85 | action: a
timestep: 4 | reward: -0.05 | action: d
timestep: 5 | reward: -1.0 | action: s
timestep: 6 | reward: -0.05 | action: d
timestep: 7 | reward: -1.0 | action: a
timestep: 8 | reward: -0.85 | action: d
timestep: 9 | reward: -0.85 | action: s

Figure 2: Maze Solver Environment

2 Using Deep Q Network Learning

We used Deep Q Network Learning to solve the custom environment and have listed our results and observations in the following subsections.

2.1 Replay Buffer

When we solve environments using methods like Deep Q Network Learning, **Catastrophic Forgetting** can occur.

Here while learning a new set/task of actions, the neural networks tend to learn the new task while catastrophically forgetting previous ones.

If we solved the environment using just a single(last) experience the variance in the 2 expriences could be very high and the agent may never converge.

To solve this issue we create a DEQUE(Double ended Queue) to store a replay buffer of memory where it stores the whole experience tuple to solve the environment on a different 'frozen' state of networks. The system stores the data discovered for [state, action, reward, next_state] - typically in a large table. This is used as raw data to feed into the neural network.

The advantange mainly is that we are able to make better use of the DQN Algorithm and past experiences to train the agent.

2.2 Target Network

The Q Network's weights get updated at each time step, and this improves the Q values of the prediction over the long term. However since the network and it's weights are the same it could change the direction of the target Q values. We would end up 'chasing' a moving target.

By using a Q Network that doesn't get trained as often we can ensure to 'freeze' a target value at least for a short amount of time. The target network does get updated since they are the final predictions, howeveer not as often as the policy network.

2.3 Representing the Q function along with it's weight

To implement DQN which is basically Q Learning in a Deep Learning Network we need to associate a weight with each Q value and hence it is represented as a function of (states and weight)

3 Results after applying DQN

After applying DQN over 2000 episodes we got the environment to converge. Following are the result graphs

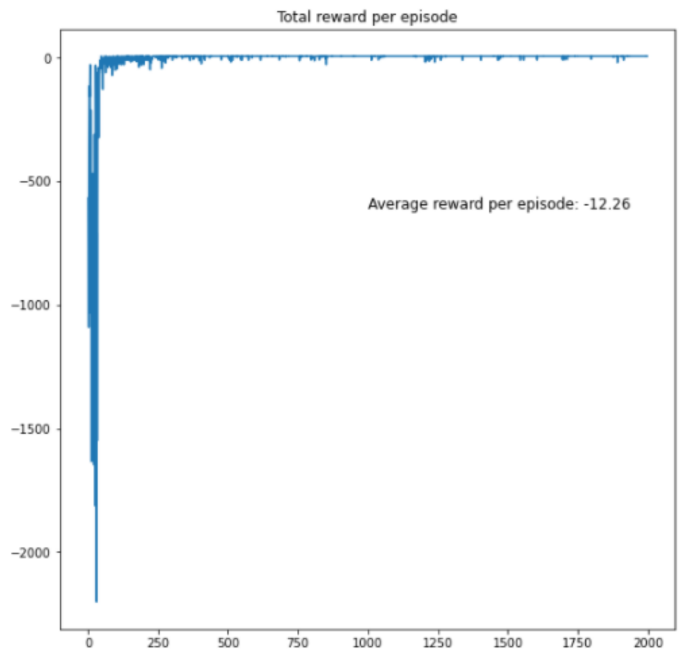


Figure 3: Total Reward Per episode for Maze Environment

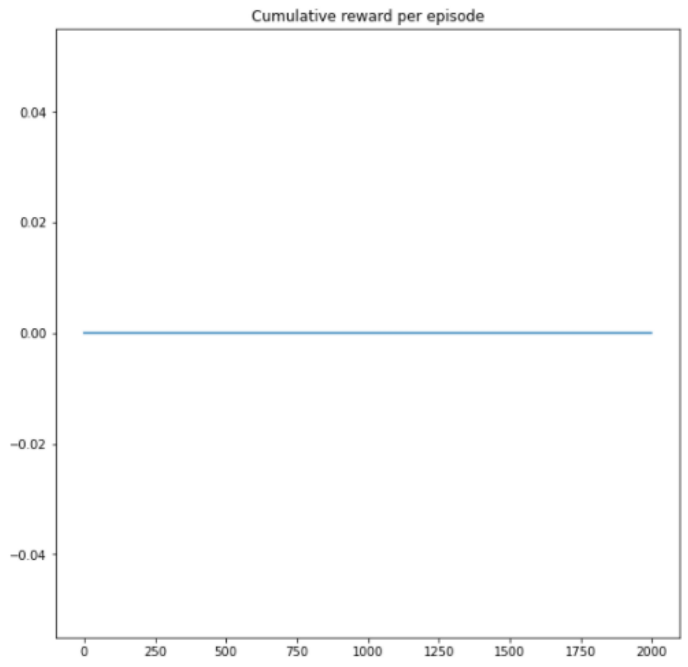


Figure 4: Cumulative Reward per Episode for Maze Environment

4 Hyperparameters

Following image is the hyperparameter values used to make the Maze Solver environment converge

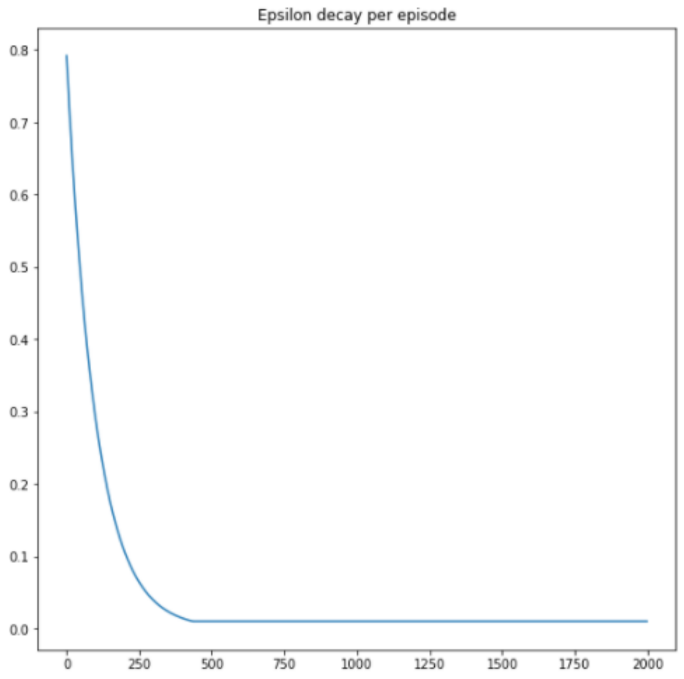


Figure 5: Epsilon decay for maze environment

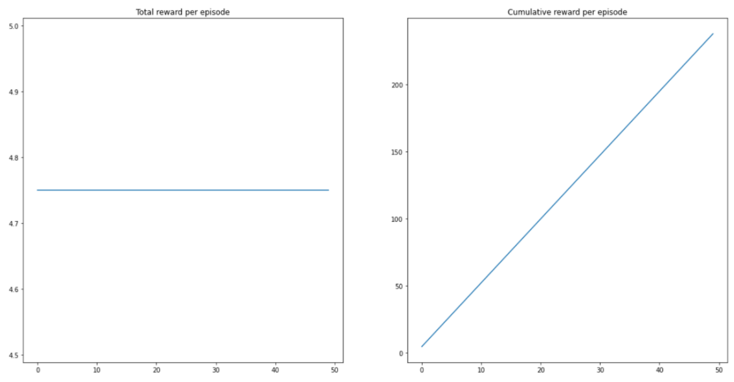
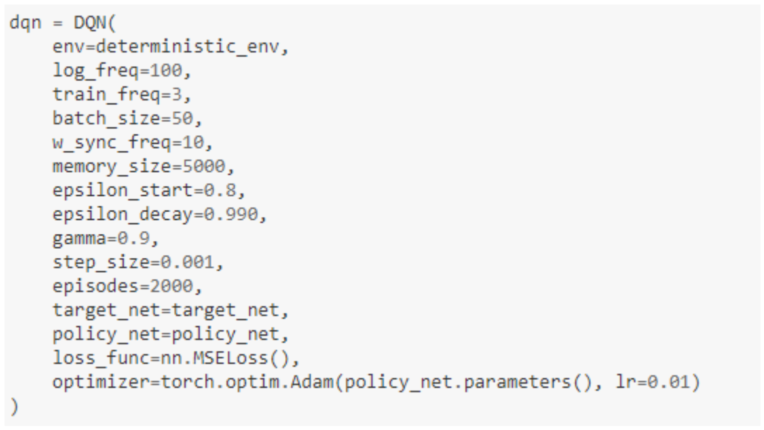


Figure 6: Evaluation results of the Maze Solver environment over 10 episodes



```
dqn = DQN(
    env=deterministic_env,
    log_freq=100,
    train_freq=3,
    batch_size=50,
    w_sync_freq=10,
    memory_size=5000,
    epsilon_start=0.8,
    epsilon_decay=0.990,
    gamma=0.9,
    step_size=0.001,
    episodes=2000,
    target_net=target_net,
    policy_net=policy_net,
    loss_func=nn.MSELoss(),
    optimizer=torch.optim.Adam(policy_net.parameters(), lr=0.01)
)
```

Figure 7: Hyperparameters used for the Maze Solver environment