**Reinforcement Learning**

**(CAP 6629)**

**Logo, company name

Description automatically generated**

**Report**

**On**

**Project - 2**

**Submitted To Submitted By**



Prof. Xingnan Zhong **Name:** Neelima Rajawat

Dept. EECS MS in Artificial Intelligence

**CAP 6629: Reinforcement Learning**

**Fall 2023**

**Course project 2**

Due: 10/31/2023 (Tuesday), 11:59PM

Submission: Two files (one report in .pdf and one .ipynb/code).

Please follow the project report guidelines and submit the report with setup,

results and analysis.

In project 1, you may realize that when you have a large grid world maze setup, it takes a long time for the agent to learn a value table. One way to eliminate this challenge is to use neural

networks to approximate the value function. There are two options provided below and you may choose either one to implement.

1. Based on your results in project 1, you can choose to build a neural network (or deep

neural network) to approximate your obtained Q or V table.

1. You can design another complex grid world example and develop the Q-learning (or

deep Q-learning) method based on that.

Either way, you are using a neural network to generate your Q or V value so that you can guide

the agent to move to achieve the goal.

**Report suggestions:**

1. Design your own grid world example and describe it at the beginning of the report.
2. Define your states, actions, and rewards.
3. Design and implement your Q network.
4. Provide the pseudo code in the report.
5. Show the convergence process of mean square error (objective function) and the weights trajectories.

**Q - Learning with Neural Network**

**Approximation in a Grid World**

**1.Introduction**

Reinforcement learning (RL) is a branch of **machine learning** that focuses on training computers to make optimal decisions by interacting with their environment. Instead of being given explicit instructions, the computer learns through trial and error: by exploring the environment and receiving rewards or punishments for its actions.

Q-learning is**a model-free reinforcement learning algorithm to learn the value of an action in a particular state.**It does not require a model of the environment (hence "model-free"), and it can handle problems with stochastic transitions and rewards without requiring adaptations.

The objective of this report is to explore the application of Q-learning, a fundamental reinforcement learning technique, in a grid world environment. Q-learning enables an agent to learn optimal actions for states by iteratively updating a Q-table. To handle larger and more complex state spaces, we extend Q-learning to leverage a neural network for Q-value approximation. We present a grid world scenario as an illustrative example to showcase this approach.

**1.1 Overview of Deep Q Networks**

**[a]. Q-table can handle simple problems with few states**

Q Learning builds a Q-table of State-Action values, with dimension *(s, a)*, where *s* is the number of states and *a* is the number of actions. Fundamentally, a Q-table maps state and action pairs to a Q-value.

A white rectangular object with black text

Description automatically generated

**Figure 1. Q Learning looks up state-action pairs in a Q table**

However, in a real-world scenario, the number of states could be huge, making it computationally intractable to build a table.

**[b]. Use a Q-Function for real-world problems:**

To address this limitation, we use a Q-function rather than a Q-table, which achieves the same result of mapping state and action pairs to a Q value.

A black text on a white background

Description automatically generated

**Fig 2. A state-action function is required to handle real-world scenarios with a large state space**

**[c]. Neural Nets are the best Function Approximators**

Since neural networks are excellent at modeling complex functions, we can use a neural network, which we call a Deep Q Network, to estimate this Q function.This function maps a state to the Q values of all the actions that can be taken from that state.

A close-up of a computer code

Description automatically generated

It learns the network’s parameters (weights) such that it can output the Optimal Q values.

The underlying principle of a Deep Q Network is very similar to the Q Learning algorithm. It starts with arbitrary Q-value estimates and explores the environment using the ε-greedy policy. And at its core, it uses the same notion of dual actions, a current action with a current Q-value and a target action with a target Q-value, for its update logic to improve its Q-value estimates.

**DQN Architecture Components**

The DQN architecture has two neural nets, the Q network and the Target networks, and a component called Experience Replay. The Q network is the agent that is trained to produce the Optimal State-Action value.

**Experience Replay** interacts with the environment to generate data to train the Q Network.

A diagram of a diagram of a diagram

Description automatically generated

**Fig 4. DQN Architecture Components**

The Q Network is a standard neural network architecture and could be as simple as a linear network with a couple of hidden layers if your state can be represented via a set of numeric variables. Or if your state data is represented as images or text, you might use a regular CNN or RNN architecture.

The Target network is identical to the Q network.

**High-level DQN Workflow**

The DQN gets trained over multiple time steps over many episodes. It goes through a sequence of operations in each time step:

A diagram of a network

Description automatically generated

These operations are performed in each time-step:

**Phase 1:**

A diagram of a network

Description automatically generated

**Experience Replay** selects an ε-greedy action from the current state, executes it in the environment, and gets back a reward and the next state.

A diagram of a process

Description automatically generated

It saves this observation as a sample of training data.

A diagram of a system

Description automatically generated

**Phase 2:**

A diagram of a network

Description automatically generated

**Q Network predicts Q-value**

All prior Experience Replay observations are saved as training data. We now take a random batch of samples from this training data, so that it contains a mix of older and more recent samples.

This batch of training data is then inputted to both networks. The Q network takes the current state and action from each data sample and predicts the Q value for that particular action. This is the ‘Predicted Q Value’.

A diagram of a process

Description automatically generated

**Target Network predicts Target Q-value**

The Target network takes the next state from each data sample and predicts the best Q value out of all actions that can be taken from that state. This is the ‘Target Q Value’.

A diagram of a process

Description automatically generated

**Compute Loss and Train Q Network**

The Predicted Q Value, Target Q Value, and the observed reward from the data sample is used to compute the Loss to train the Q Network. The Target Network is not trained.

A diagram of a process

Description automatically generated

**2. Grid World Example**

In our designed grid world, we consider a 6x6 grid. The agent starts at the top-left cell, denoted as 'S,' and must navigate to the bottom-right cell, marked as 'G.' Certain cells in the grid are obstructed ('O'), presenting navigational challenges for the agent.

**3. States, Actions, and Rewards**

**States**

The state space consists of the agent's current position, defined by its row and column in the grid.

**Actions**

The agent can take four actions: UP, DOWN, LEFT, and RIGHT, allowing it to move in the respective directions.

**Rewards**

Moving into an empty cell incurs a penalty of -1.

Reaching the goal ('G') results in a significant positive reward of +10.

Colliding with an obstacle ('O') imposes a large negative reward of -100.

**4. Q-Network Design and Implementation**

To approximate Q-values efficiently, we implemented a neural network-based Q-network. The network consists of an input layer representing the state, a hidden layer with 24 units, and an output layer with 4 units corresponding to the available actions. The network is trained to minimize the mean squared error (MSE) loss between predicted Q-values and target Q-values.

A diagram of a network

Description automatically generated

**Figure 1: Sample NN Format**

As shown in the above diagram, it works this way,

1. **Input Layer:** The first layer that receives input data and passes it to the neural network. It takes the current state as input.
2. **Hidden Layers:** Intermediate layers that perform computations and feature extraction. They are called "hidden" because their calculations are not directly observed. Perform computations and feature extraction to estimate Q-values.
3. **Output Layer:** The final layer that produces the network's predictions or results based on the information processed in the hidden layers. The number of neurons in the output layer depends on the task. Provides Q-values for each possible action. The action with the highest Q-value is chosen by the agent. These layers enable the agent to learn and make decisions in reinforcement learning tasks.

A grid with blue squares and black dots

Description automatically generated

**Figure 2: Grid World**

For implementation, decide to go with 6X6 grid, as shown in figure 2.

In this grid world, are at positions:

Obstacle: [(1, 2), (2, 3), (3, 1), (3, 4), (4, 1), (4, 4)]

Start: [0, 0]

Goal: [ 5, 5]

**5. Pseudo Code**

**Step1: Define hyperparameters**

NUM\_EPISODES = 100, MAX\_STEPS = 100, EXPLORATION\_PROB = 0.2, GRID\_WIDTH = 6, GRID\_HEIGHT = 6, NUM\_ACTIONS = 4, LEARNING\_RATE = 0.1, DISCOUNT\_FACTOR = 0.9

**Step2: Define the grid world and obstacle states**

grid = create\_empty\_grid(GRID\_WIDTH, GRID\_HEIGHT)

OBSTACLE\_STATES = [(1, 2), (2, 3), (3, 1), (3, 4), (4, 1), (4, 4)]

set\_obstacle\_rewards(grid, OBSTACLE\_STATES)

set\_goal\_and\_start\_states(grid)

ACTIONS = ['UP', 'DOWN', 'LEFT', 'RIGHT']

**Step3: Initialize the Q-network with random weights**

model = initialize\_q\_network(GRID\_WIDTH, GRID\_HEIGHT, NUM\_ACTIONS)

**Step4: Q-learning algorithm using neural network approximation**

for episode in range(NUM\_EPISODES):

state = get\_start\_state()

total\_reward = 0

for step in range(MAX\_STEPS):

**4.1 Select an action using epsilon-greedy strategy**

action = select\_action(state, model, EXPLORATION\_PROB)

**4.2 Perform the action and observe the new state and reward**

new\_state, reward = take\_action(state, action, grid)

total\_reward += reward

**4.3 Update the Q-value using the Q-network and Bellman equation**

q\_values\_current = model.predict(state)

q\_values\_next = model.predict(new\_state)

q\_values\_current[action] = q\_values\_current[action] + LEARNING\_RATE \* (reward + DISCOUNT\_FACTOR \* max(q\_values\_next) - q\_values\_current[action])

**4.4 Update the Q-network with the new Q-values**

model.update\_q\_values(state, q\_values\_current)

state = new\_state

**4.5 Check if reached the goal or obstacle**

if is\_goal\_state(new\_state):

break

**Step5: Update the exploration probability**

EXPLORATION\_PROB = update\_exploration\_prob(episode)

**Step6: Calculate the Mean Squared Error (MSE) for this episode**

mse = calculate\_mse(q\_values\_current)

record\_mse\_for\_episode(mse)

**Step7: Display episode and step count**

print(f"Episode {episode + 1}/{NUM\_EPISODES}, Steps: {step + 1}, MSE: {mse:.2f}")

**Step8: Display the learned Q-values using the neural network**

display\_learned\_q\_values(model)

**Step9: Plot the Mean Squared Error (MSE) curve**

plot\_mse\_curve()

**6. Results**

After running the code, for 6x6 grid, we finally get the values.

A screenshot of a grid

Description automatically generated

**Figure 2: Grid World**

After that, want to bring attention over the final Q-values and V-values over the grid world, which is shown as below:

A screenshot of a grid

Description automatically generated

**Figure 3: Grid World**

**7. Convergence Process**

The convergence process was visualized by monitoring the mean squared error (MSE) of the Q-network during training. Additionally, we tracked the trajectories of weights in the neural network throughout the training process. These visualizations provide insights into the network's learning progress and convergence to an optimal Q-value function.

A graph with a line drawn on it

Description automatically generated

**Figure 4: MSE vs Episode**

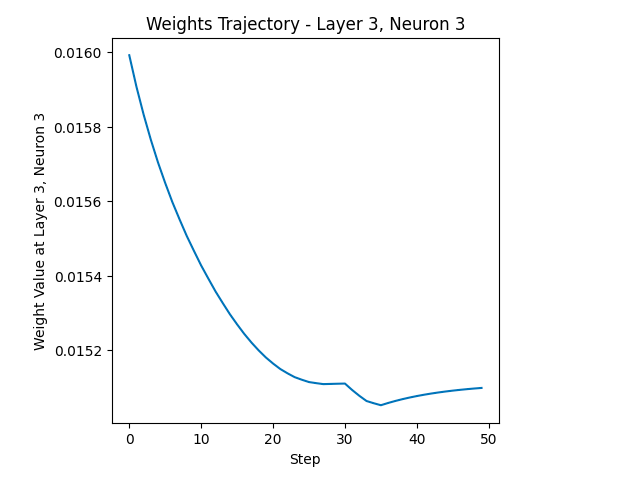
As we can see in the above plot, mean squared error (MSE) is decreasing over the time.

**Weight Trajectories:** refer the next page.

A graph with a line

Description automatically generated

**Figure 5: Weight Trajectory for Layer 1, Neuron 2**

****

**Figure 6: Weight Trajectory for Layer 3, Neuron 3**

For all layers and neurons weight trajectories, please refer the code part**.**

**8. Conclusion**

The integration of neural network approximation in Q-learning allows for more efficient and scalable solutions in reinforcement learning problems. This report showcased the application of Q-learning with a neural network in a grid world scenario, paving the way for further exploration and experimentation in the domain of reinforcement learning.

Therefore, this Q-learning implementation using a neural network successfully trained an agent to find an optimal path in a grid world. It demonstrated the power of reinforcement learning techniques in solving navigation problems and can be further extended and customized for more complex scenarios. This approach provides a foundation for building intelligent agents capable of making decisions and solving problems in a wide range of environments.

**Reference**

1. <https://www.geeksforgeeks.org/deep-q-learning/>
2. <https://en.wikipedia.org/wiki/Reinforcement_learning>
3. <https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b>
4. Bowen Baker, Otkrist Gupta, Nikhil Naik & Ramesh Raskar. DESIGNING NEURAL NETWORK ARCHITECTURES USING REINFORCEMENT LEARNING Under review as a conference paper at ICLR 2017