National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Faculty of Computing

CS 368: Reinforcement Learning

Lab 10: Value Function Approximation

(Deep Q Networks)

Date: 29 November 2024

Time: 02:00 PM – 5:00 PM

Instructor: Dr. Zuhair Zafar

Lab 10: Value Function Approximation (Deep Q Networks)

Objectives

Given a simulated environment, improve the policy so as to maximize the sum of rewards (returns) using Deep Q networks.

Tools/Software Requirement:

Google Colab, Python, Gymnasium Library, PyTorch

Introduction

Deep Q-Networks (DQN) is a reinforcement learning algorithm that combines **Q-learning**, a popular method for solving decision-making problems, with **deep neural networks**, enabling it to handle complex, high-dimensional state spaces. It was introduced by DeepMind in 2013 and gained prominence due to its success in learning to play Atari games directly from pixel inputs.

The key idea behind DQN is to use a neural network to approximate the Q-value function Q(s, a), which represents the expected cumulative reward of taking an action aaa in a state sss and following the optimal policy thereafter.

Deep Q Networks

Core Components of DQN

1. **Q-Learning Recap**
   Q-learning is a model-free, off-policy reinforcement learning algorithm that updates the Q-value function using the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a))$$

where:

- r: reward for taking action aaa in state s,
- γ: discount factor, balancing immediate and future rewards,
- s′: next state,
- α: learning rate.

2. **Deep Q-Network**
   In DQN, a neural network $Q(s,a,\theta)$ parameterized by θ is used to approximate $Q(s,a)$. Instead of storing Q-values in a table, the network predicts Q-values for all actions given a state.

3. **Replay Buffer**
   DQN uses a replay buffer to store the agent's experiences (s, a, r, s'). During training, mini-batches of experiences are sampled randomly from the buffer. This:

- Breaks correlations between consecutive experiences,
- Stabilizes training,
- Improves data efficiency.

4. **Target Network**

   DQN employs a separate target network $Q\ (s, a, \theta^-)$, which is periodically updated with the weights of the main network $Q\ (s, a, \theta)$. This reduces instability caused by the network chasing its own predictions during updates.

The DQN Algorithm

1. **Initialize**:
   - Replay buffer D,
   - Q-network $Q\ (s, a, \theta)$ with random weights θ,
   - Target network $Q\ (s, a, \theta^-)$, with weights $\theta^- = \theta$.
2. **For each episode**:
   - **Reset the environment** to an initial state s.
   - **For each step within the episode**:
     1. **Select an action a**:
        - Use an $\epsilon\ -greedy$ policy:
          - With probability ϵ, choose a random action (exploration),
          - Otherwise, choose a $= \underset{a' \in A}{\text{argmax}}\, Q(s, a, \theta)$ (exploitation).
     2. **Execute the action** a, observe reward r and next state s′.
     3. **Store transition** (s, a, r, s') in the replay buffer D.
     4. **Sample a mini-batch** of transitions (s_i, a_i, r_i, s_i') from D.
     5. **Compute the target**:
        $$y_i = r_i + \gamma \max_{a' \in A} Q(s_i', a', \theta^-)$$
     6. **Update the Q-network**:
        - Perform a gradient descent step to minimize the loss:
          $$L(\theta) = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i, \theta))^2$$
          where N is the mini-batch size.
     7. **Periodically update the target network**:
        - Set $\theta^- \leftarrow \theta$
3. **Repeat** until convergence or sufficient performance is achieved.

DQN Algorithm Pseudocode:

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ — Experience Replay
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$ — Fixed Q-Target
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Advantages of DQN

- Capable of handling high-dimensional input spaces, such as images.
- Requires no prior knowledge of the environment's dynamics.
- Proven to solve challenging problems, like playing video games at or above human levels.

Lab Tasks

- Discount factor, gamma, $\gamma = 0.99$
- Exploration factor, epsilon, $\varepsilon = 0.2$
- Number of Episodes = 2500
- Batch Size = 32
- Replay Memory Size = 100,000

1. **Solve the Acrobat v1 problem given in the Gymnasium library using the Deep Q network algorithm. You must use deep neural network to find optimal Q values. Plot the returns for every 100 episodes and the MSE (network loss).**

Hint: If you're looking for guidance on coding the deep learning architecture and understanding its integration with Reinforcement Learning, consider exploring the following resource: Johnnycode8's Gym Solutions on GitHub.
This repository provides practical examples and implementations of reinforcement learning solutions for OpenAI Gym environments, including deep learning models. It can serve as an

excellent starting point for understanding how to build, train, and apply reinforcement learning algorithms.

Deliverable:
Please submit your notebook on LMS before the deadline **(5th December 2024, 11:59pm).**

Lab Rubrics

Assessment	**Does not meet expectation** **(1/2 marks)**	**Meets expectation** **(3/4 marks)**	**Exceeds expectation** **(5 marks)**
Software Problem Realization **(CLO1, PLO1)**	The student struggles to formulate the problem as RL and does not apply Deep Q Network algorithm to solve it. There is a lack of understanding of the problem's requirements and no attempt to find the optimal policy effectively.	The student formulates the problem as RL with some guidance, applies Deep Q network algorithm with hints, and shows it's working. However, the approach might not be fully optimized or lacks a thorough justification.	The student independently formulates the given problem as RL, applies Deep Q network algorithm without guidance, and effectively finds an optimal policy. The approach is fully optimized and can be applied to different similar problems.
Software Tool Usage **(CLO4, PLO5)**	Code has syntax errors, and the implementation of the Deep Q network algorithm is incorrect or incomplete. The code is not modular and lacks comments for readability and reuse. The student shows limited ability to use gymnasium / pytorch library functions where required.	The student has implemented the Deep Q networks algorithm correctly for the given problem with minor mistakes. The code is mostly correct in terms of syntax and functionality but might not be optimized or well-structured for reuse. Some documentation is provided. The student also shows working knowledge of gymnasium and Pytorch library where required.	The student has implemented the Deep Q network algorithm efficiently and correctly. The code is clean, modular, well-documented, and follows best practices for reproducibility and reuse. The student demonstrates full command of the gymnasium and pytorch library and its functions.