Image Classification using Reinforcement Learning

Talukder Asir Saadat

Abstract

This research project focuses on classifying images using the fundamentals of reinforcement learning. Although reinforcement learning is primarily used for sequential decision-making tasks to determine the optimal action from given states, in our case, the action involves selecting or classifying the correct image. I conducted an experiment on image classification using the MNIST dataset and provided an analysis of performance on DQN. The implementation can be seen in the link to my GitHub Repository

1 Introduction

Image classification is a fundamental task in computer vision. There are numerous studies on classification, such as digit classification using the MNIST dataset (Baldominos et al. 2019) and medical image (Li et al. 2014) classification. Traditional methods for image classification often rely on supervised learning techniques. However, reinforcement learning (RL) offers a promising alternative by framing the classification problem as a series of actions taken by an agent to maximize a reward.

Reinforcement learning, typically used for sequential decision-making tasks, can be adapted for image classification by treating the correct class selection as the optimal action. In this report, I explore the application of reinforcement learning to image classification using the MNIST dataset. Reinforcement learning (RL) has gained significant popularity and is commonly used in scenarios involving multiple states and agents that learn through different episodes. For example, RL has been successfully applied to playing Atari games and Pac-Man. With the help of RL and modern algorithms, agents learn the states or situations based on the rewards they receive. In my case, image classification is not a game, nor does it involve agents learning through living rewards or each episode. Instead, the RL agent receives a reward for correctly classifying an image. The input is the image, and the agent must take an action based on the given state. Each episode involves inputting the image and determining the action chosen by the agent. From there, it backpropagates and adjusts the weights accordingly.

For the experiment, I have utilized, Deep Q network(DQN)(Roderick et al. 2017) algorithm. I conducted experiments to evaluate the performance of various RL models and provide a comparative analysis of their accuracy and loss. My findings demonstrate the potential of reinforcement learning in image classification.

Corresponding author(s): Talukder Asir Saadat,
https://github.com/Asir-Saadat/Image-Classification-with-RL, asirsaadat@bubt.edu.bd

2 Background and Approach

2.1 Reinforcement Learning

2.1.1 Bellman Equation

The Bellman equation is a fundamental concept in reinforcement learning. We also call this the value-function. It is the cumulative reward that can be achieved from a particular state. The equation can be written as-

$$V(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right] \tag{1}$$

where:

- $V(s)$ is the value of state s,
- a represents the possible actions,
- $R(s,a)$ is the immediate reward received after taking action a in state s,
- γ is the discount factor,
- $P(s'|s,a)$ is the transition probability of moving to state s' from state s after taking action a.

It helps determine how good a given policy is by computing the expected rewards it achieves. We will see more of these during model implementation. This is actually state value function as it takes the maximum of value for each action. There is another value function called the action value function The expected cumulative reward starting from state s, taking action a, and then following policy. The equation can be written as-

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a') \tag{2}$$

2.1.2 Loss Function

In RL, the loss function is often symbolized as L. The loss function measures the difference between the predicted Q-values and the target Q-values. Predicted Q-value s is the Q-value estimated by the current policy for a given state-action pair. A common loss function used in RL is the mean squared error (MSE) loss. This can be shown as:

$$L = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)^2 \right] \tag{3}$$

2.1.3 Epsilon Greedy exploration

This approach balances exploration and exploitation. In this method, an agent selects a random action with probability ϵ(exploration) and chooses the action

with the highest estimated reward (Q-value) with probability $(1 - \epsilon)$ (exploitation). Initially, ϵ is set to a high value to encourage exploration of various actions and states. Over time, ϵ is gradually reduced, allowing the agent to increasingly exploit its learned knowledge to make optimal decisions.

2.1.4 Replay Buffer

The replay buffer stores the agent's experiences, including state, action, reward, and next state, in a buffer. During training, the DQN samples random mini-batches of experiences from the replay buffer to update the Q-values. This process helps break the correlation between consecutive experiences, leading to more stable and efficient learning.

2.2 Methodology

2.2.1 MNIST dataset

The MNIST dataset is a widely used benchmark in the field of machine learning and computer vision. It consists of 70,000 grayscale images of handwritten digits which are from 0 to 9. Each image is 28x28 pixels in size and is widely used for classifying tasks. For our research project, we will be using the MNIST dataset for evaluation.

2.2.2 Convolutional Neural Network

For my project, I have a CNN for DQN which is shown in Figure:1. The network consists of two convolutional layers followed by max-pooling layers. The first convolutional layer applies 32 filters of size 3x3 with ReLU activation and same padding, followed by a max-pooling layer that reduces the spatial dimensions by half. The second convolutional layer applies 64 filters of size 3x3 with ReLU activation and same padding, followed by another max-pooling layer. The output from the convolutional layers is then flattened into a one-dimensional vector by the Flatten layer, which serves as the input to the fully connected layers. The first fully connected layer has 512 units with ReLU activation, and the final fully connected layer has 10 units, corresponding to the 10 possible digit classes (0-9).

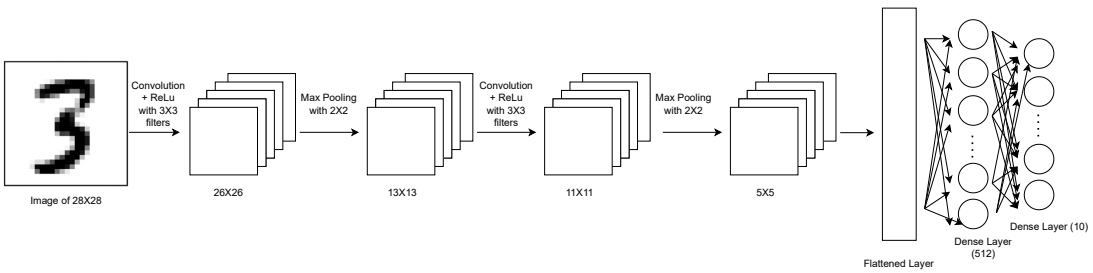

Figure 1: The Convolutional Neural Network

2.2.3 Trainig Loss and Accuracy

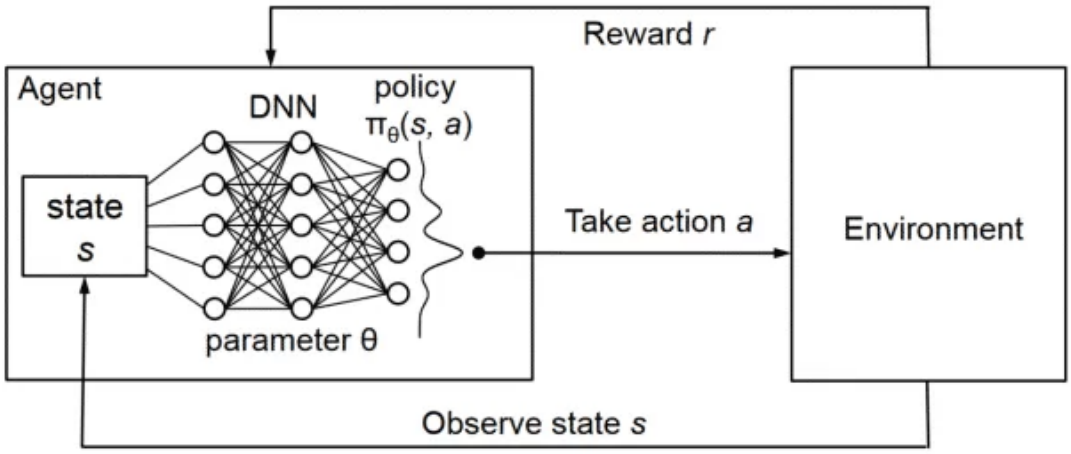

Figure 2: The Deep Q-Network

DQN uses a neural network to approximate the Q-value function, which estimates the expected cumulative reward for taking a specific action in a given state. The network is trained using a loss function that minimizes the difference between the predicted Q-values and the target Q-values. DQN employs techniques such as experience replay, where past experiences are stored and sampled randomly during training, and target networks, which stabilize the learning process by providing fixed Q-value targets for a certain number of steps.

3 Experimental Setup

The entire experiment was conducted on Google Colab using a T4 GPU. The experiment consisted of 10 epochs, during which the training loss and accuracy were calculated for each epoch. As previously mentioned, I utilized the Deep Q-Network (DQN) model for this experiment and evaluated its performance. To evaluate the model's performance, I utilized accuracy as the primary metric. There are several hyperparameters that need to be considered for DQN. The learning rate of the Adam optimizer is 0.001. In RL, an agent typically begins with exploration before shifting to exploitation. Initially, the exploration rate, denoted by epsilon value ϵ is set high to encourage the agent to explore various actions. For my experiment, the starting epsilon value was set to 1.0. As the agent gains more experience, it needs to focus on taking optimal actions, which correspond to the highest Q-values. Therefore, the epsilon value gradually decreases over time. In this experiment, the epsilon decay rate was set to 0.995, with a minimum epsilon value of 0.01. The epsilon value decreases after training each batch of data, allowing the agent to transition from exploration to exploitation effectively.

As mentioned previously, we will be using the Bellman equation. In our scenario, each image input is treated as an individual episode. In reinforcement learning (RL), we typically focus on the terminal value of the episode and do not consider future value functions or Q-value functions. In this context, the terminal

value refers to determining whether the image classification is correct or incorrect, and we only take the reward into account. Although we have set the gamma (γ) hyperparameter to 0.99, it will not be utilized in our case. As for the replay buffer, I have used a double-ended queue with a maximum capacity of 50,000, allowing it to hold up to 50,000 experiences which was taken from the github (MicroSoft 2019).

4 Experimental Results

As shown in Fig. 3, it demonstrates the training loss and accuracy. It is clear that the model performs the worst for the first epoch as the test accuracy for the first batch of images is only 12.50%. But gradually, the model tends to learn more and more, finishing of test accuracy of batch 9 with 98.64%.

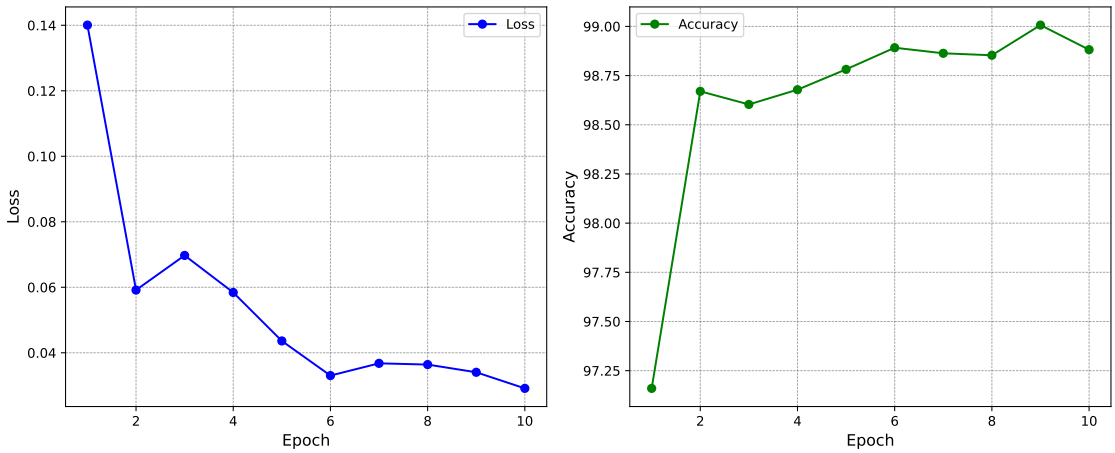

Figure 3: Training Loss and Accuracy

5 Discussion and Conclusion

I have experimented with the MNIST dataset using the DQN algorithm and it performs quite well, as shown in Fig. 3. The primary issue I encountered is that the model behaves similarly to a regular CNN. The true essence of reinforcement learning (RL) is not fully realized for several reasons. Firstly, choosing an action is similar to predicting a class, and there are no longer distinct episodes. Each episode involves inputting an image and obtaining an output. Also, the Bellman equation does not apply here, as there are no future rewards to consider as each input state is a terminal state.

However, there are instances where it retains certain properties. For example, I have used two neural networks: the primary DQN and the target DQN. The target DQN is updated less frequently and serves as a stable reference for the primary network to learn from. This approach helps stabilize the learning process by providing consistent targets and preventing the primary network from chasing a moving target.

References

Baldominos, Alejandro, Yago Saez & Pedro Isasi. 2019. A survey of handwritten character recognition with mnist and emnist. *Applied Sciences* 9(15). 3169.

Li, Qing, Weidong Cai, Xiaogang Wang, Yun Zhou, David Dagan Feng & Mei Chen. 2014. Medical image classification with convolutional neural network. In *2014 13th international conference on control automation robotics & vision (icarcv)*, 844–848. IEEE.

MicroSoft. 2019. Tqdn. https://github.com/microsoft/tdqn/blob/master/tdqn/replay.py.

Roderick, Melrose, James MacGlashan & Stefanie Tellex. 2017. Implementing the deep q-network. *arXiv preprint arXiv:1711.07478* .