11155009 方品仁 HW2 Report

Cartpole V1

Introduction

This environment corresponds to the version of the cart-pole problem described by Barto, Sutton, and Anderson in “Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem”. A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum is placed upright on the cart and the goal is to balance the pole by applying forces in the left and right direction on the cart.

Experiments

The experiment is aimed to find out the performance between basic Q-learning and Deep Q-learning (DQN) on a continuous action space. For discrete action space, basic Q-learning with a simple Q-table can store all possible states and actions. While for continuous action space, it’s not an easy work. A simple way is discretization. Split the action space into multiples bins. Then for an action value, we can then categorize the action to the closest bin. The action can then become a discrete action space. The performance of this method is determined by the number of bins. The more bins we can have, the more accurate action we can choose. But there is a tradeoff between the performance and the required training time and memory consuming.

Another way to deal with continuous action space is DQN, replacing the basic Q-table with a neural network. As other comparisons between traditional models and neural network, DQN tend to have a better performance than basic Q-learning.

The experiment of this task is designed to compare the performance of basic Q-learning and DQN. And the performance and time consuming of different

number of bins in basic Q-learning.

Training

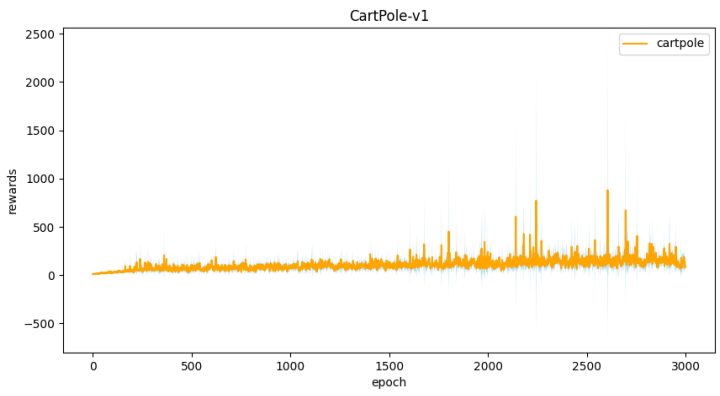
Basic Q-learning

- Number of bins: 7
- ε-greedy policy: Start with $\varepsilon = 0.95$
- Q-value update rule: $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma \cdot \text{amax}Q(s',a)]$
- Episode per run: 3000
- Decay: 0.045

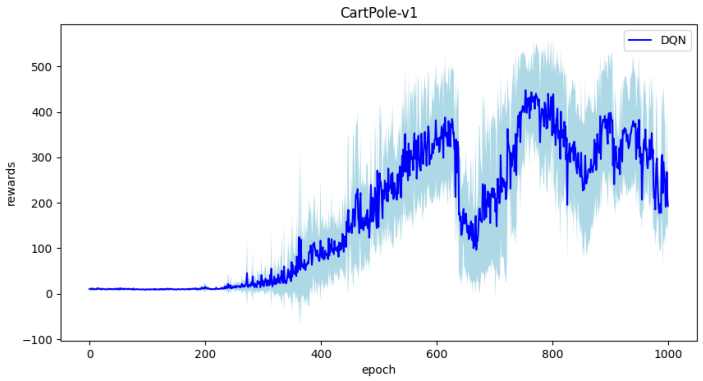
DQN

- Use experience replay and target network to stabilize the training process
- 2 hidden layers and outputs Q-values for each action
- Epsilon-Greedy policy is used to balance exploration and exploitation
- Optimizer: Adam optimizer
- Learning rate: 0.0002
- Epsilon: 0.95
- Gamma: 0.97
- Batch size: 32
- Replay buffer size: 10000
- Update target net for every 100 steps

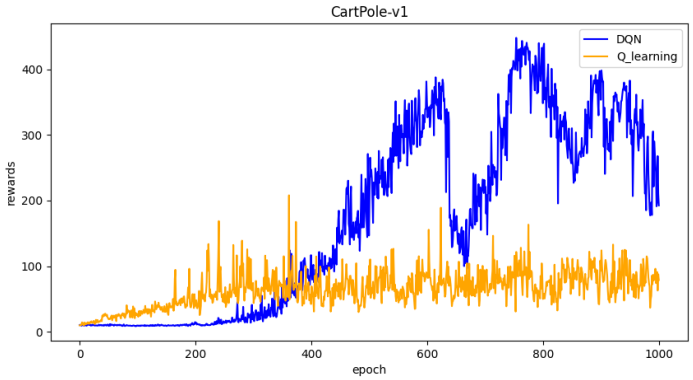
Result



(basic Q-learning)



(DQN)



(comparison)

	Basic Q-learning	DQN
Average score over tests for 5 times	146	500

Number of bins	7	10	15
score	146	124	100
Training time	25~30 secs	40~45 secs	90~200 secs

Out of my exception, the performance doesn't improve as the number of bins increases; while the time consuming does increase.

Discussion

In this task, I learned a basic of Q-learning and basic DQN. It gave me a good experience to conduct on the job in Atari, Assault-v5. I learned how experience replay helps the network and how target network can stabilize the training process. And I learned that it is hard to train a DQN model because the data are not fixed as supervised learning. They are from a state after an action, which

means that I can barely send a great amount of data into the network.

Atari, Assault-v5

Introduction

You control a vehicle that can move sideways. A big mother ship circles overhead and continually deploys smaller drones. You must destroy these enemies and dodge their attacks.

Experiments

1. **Influence by replay buffer size**

   We know that small replay buffer size can cause overfitting to recent experience. Therefore, we tend to have a bigger replay buffer size like 10000. And my question is, will a more bigger buffer size lead to a better performance?

2. **Influence by different CNN structure**

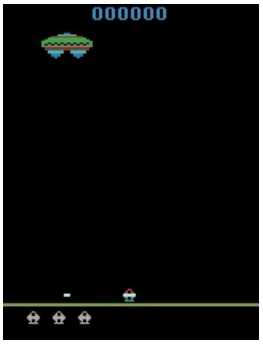
   [reference] In the paper, the authors mention using max pooling layers in their convolutional neural network (CNN) for a few key reasons. Despite initially not wanting to introduce translation invariance (since the position of game entities is important), they found that max pooling helped compress their large state space into a smaller vector of size 768. This was crucial for making the network more manageable and efficient.

   In the typical CNN in RL, max pooling is not recommended to be added into the model because they can reduce important spatial information that is crucial for decision-making. Since RL tasks, like game playing, often require precise understanding of the environment's state, including the position of objects, max pooling may hinder the agent's ability to make informed decisions. Retaining more detailed spatial features is typically preferred to improve the model's performance in such tasks.

3. **Influence by different training frame area**

   The picture below is a frame of the game. In my opinion, the upper part of the frame (the score part and the mother ship) and the lower part (the part below the green line) is not important for the agent. Therefore, I only use the

middle part to train the agent.



Training

- Replay buffer size: 10000
- Stack frames number: 4
- Preprocess of a frame: clip the upper part and the lower part, grayscale conversion, resizing to (84, 84), and normalize to [0, 1]
- Update target net for every 1000 steps
- ε-greedy policy for action selection
- Epsilon decay: exponential decay
- Loss function: Smooth L1 Loss
- Optimizer: RMSprop optimizer with a learning rate 0.00025
- Warm up steps: 10000 warm up steps for collecting experience
- Number of Episodes: ideally 10000 episodes
- Clipped Rewards: The rewards are clipped to the range [-1, 1] to prevent overly large updates and maintain stability.
- Evaluation during training: average reward of the last 50 episodes

Result

1. **Influence by replay buffer size**

   I tried the buffer size with 10000 and 50000. And the result was that the bigger buffer size, 50000, didn't significantly improved the performance. Instead, it slowed down the speed of training and caused the consuming of memory.

2. **Influence by different CNN structure**

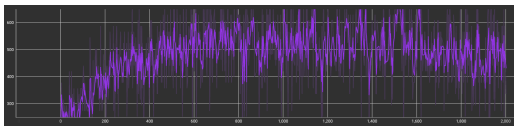
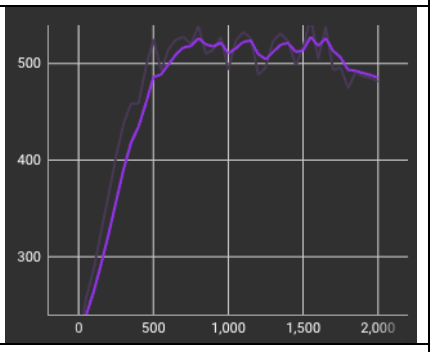
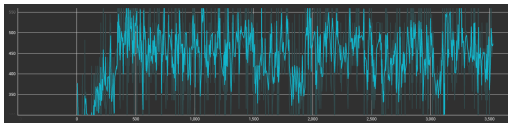
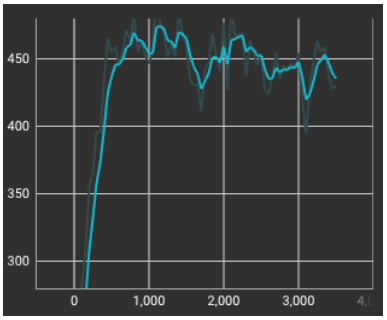
   The max-pooling layer doesn't really help the model be better. Therefore, I

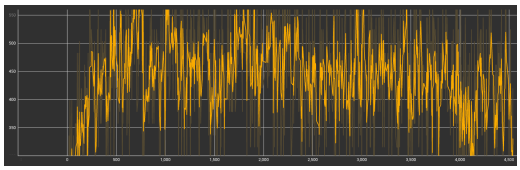
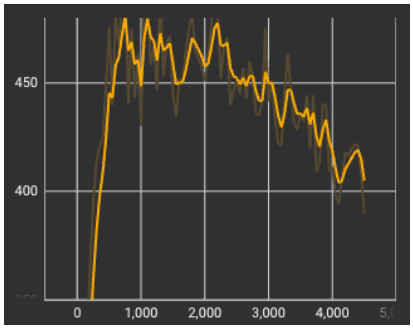
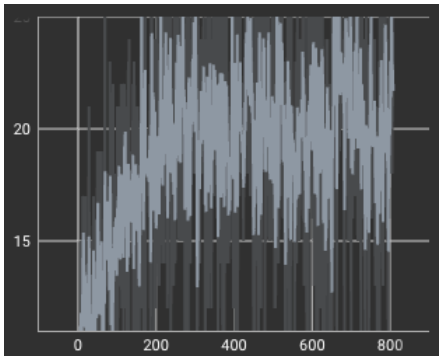
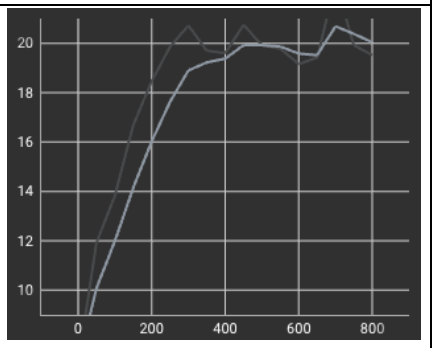
think the key to the better performance in the paper lies not just in the use of max pooling layers, but in a combination of factors. These include experience replay for stabilizing training, careful hyperparameter tuning to optimize learning, effective preprocessing by converting frames to grayscale and binary, and a well-designed CNN architecture to capture relevant features. Additionally, stacking consecutive frames provides crucial temporal context, and the appropriate discount factor helps balance short-term and long-term rewards. These elements, along with careful network initialization and gradient descent techniques, enable the model to successfully tackle the complexity of the Atari *Assault* game, resulting in superior performance compared to traditional Q-learning methods.

3. **Influence by different training frame area**

Cropping the upper part and the lower part of a frame and training only the remaining parts do help the training process. The reward increases by about 100 in the first 1000 episodes. And the numbers of episodes required to reach the max reward is reduced.

The result of this experiment shows that focusing the most meaningful part to a human of a frame leads to a great improvement.

	Training reward	Avg reward of 50 episodes
clipped area		
Whole area		

50000 buffer size		
Max-pooling layer	 (clipped reward)	 (clipped reward)

	Clipped area	Whole area	50000 buffer size	Max-pooling layer
Average testing reward	402	314	322	290

Discussion

Due to the limited hardware resource, it is hard to train a good DQN. I tried to run my code on Google Collab, but because the use of replay buffer, after trained for 20 episodes, the RAM is out of usage on collab. Therefore, I must train my model on my own laptop. It took almost a day to train for 1000 episodes. To save time for more experiments, I had to stop the training process once I found the training reward didn't increase and maybe started to decrease. This leads to uncomplete training process. The result was that the score was not good enough compared to other people's models.

Future Works

By reading the figure of training reward, I think that the agent stuck at a local

optimal solution. To solve this problem, there are some methods I can try to work on:

1. Reset the epsilon for every 500 episodes. This can have the agent explore more when the training process runs more.
2. Use Prioritized Experience Replay. I think as the training process runs, the agent learns old experience. So using Prioritized Experience Replay can have the agent learn to focus on the experience with a greater TD-error.
3. Add a learning scheduler.

Video Link

- cartpole
- Assault-v5

Reference

- CS229 Project Final Report Deep Q-Learning on Arcade Game Assault
- The Importance of Experience Replay Buffer Size in Deep Reinforcement Learning