HW5 R12922A12

Describe the Deep Q-Network (7%)

Deep Q-Network (DQN) is an advanced algorithm that integrates Q-learning with deep neural networks to enhance reinforcement learning. Here's a refined and improved version of your explanation:

Q-Learning in DQN

Q-learning is a type of model-free reinforcement learning algorithm that aims to learn a policy, which tells an agent what action to take under what circumstances. It does this by using a Q-value function which representing the expected cumulative reward of taking action a in state s and following the optimal policy thereafter.

Combining Q-Learning with Deep Neural Networks

DQN leverages a Convolutional Neural Network (CNN) to approximate the Q-value function. This network maps state-action pairs to their respective Q-values, effectively learning the relationship between states and actions. The steps are as follows:

Feature Extraction: The CNN extracts features from the state representation (such as images in game environments) to understand the important aspects that influence decision-making.

Q-Value Estimation: Using these features, the network predicts the Q-values for all possible actions in the given state. This allows the agent to evaluate the potential rewards of different actions.

Training the Network

The DQN algorithm updates the network based on experiences collected during the agent's interaction with the environment. The key steps include:

The advantages of DQN are as followed :

The use of experience replay and target networks significantly improves the convergence of the training process.

The deep neural network allows DQN to handle high-dimensional state spaces effectively, making it suitable for complex environments.

Describe the architecture of your PacmanActionCNN (7%)

In initialize part:

Construct 3 convolutional layers:

The first convolutional layer takes the input state with a specified number of channels (state_dim). It has 32 filters with a kernel size of 8 and a stride of 4.

The second convolutional layer has 64 filters, a kernel size of 4, and a stride of 2.

The third convolutional layer also has 64 filters, with a smaller kernel size of 3 and a stride of 1.

Construct 2 fully connected layers for advantage and 2 fully connected layers for value streams:

The first fully connected layer for the advantage stream, with 512 output features.

The first fully connected layer for the value stream, with 512 output features.

The second fully connected layer for the advantage stream, with the number of output features equal to the number of possible actions (action_dim).

The second fully connected layer for the value stream, with a single output feature representing the state value.

Initialize an ReLU activation function.

For Forward

The input state x passes through the three convolutional layers sequentially, each followed by a ReLU activation function.

The output of the final convolutional layer is flattened into a 1D vector using.

The flattened vector is split into two separate streams: the advantage stream and the value stream.

The advantage stream passes through self.fc1_adv followed by a ReLU activation, and then through self.fc2_adv to produce the advantage values (adv).

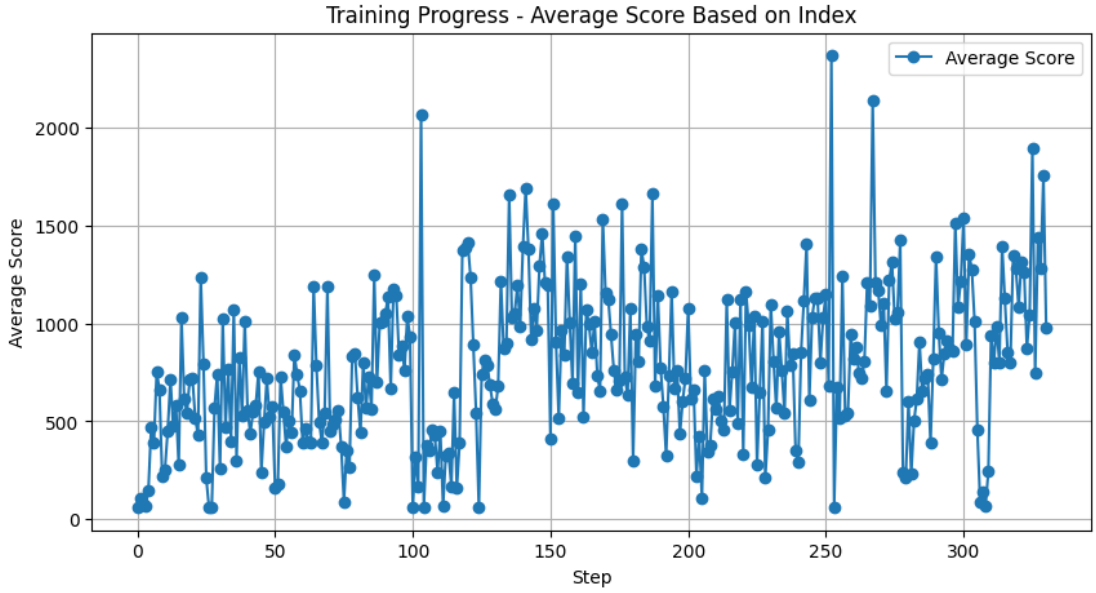
The value stream passes through self.fc1_val followed by a ReLU activation, and then through self.fc2_val to produce the value (val), which is expanded to match the shape of the advantage values.

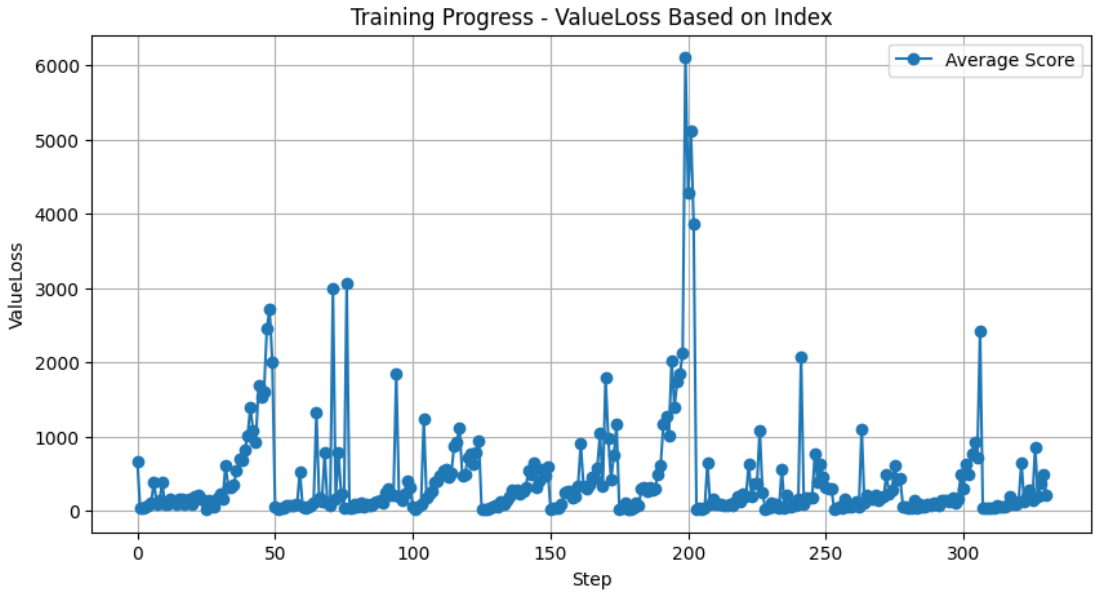
The final output is computed by combining the advantage and value streams:

State value + advantage value of this action – the mean of advantate value with other action.

The final combined values are returned as the output, representing the Q-values for each possible action given the input state.

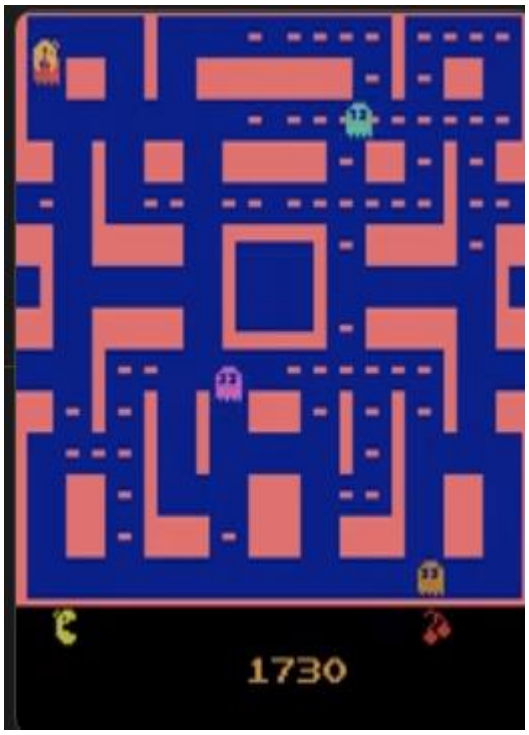
Plot your training curve, including both loss and rewards. (3%)





Show screenshots from your evaluation video (3%)


1710


1730


```
  logger.warn(
The score of the agent:  1750.0
```