Level 1

1.

Agents are usually easier to learn from the game display (84x84 pixel image) rather than the console's RAM (128 byte array). This is because pixel images provide a more direct representation of the game environment and can be processed using convolutional neural networks (CNNs) to learn relevant features for decision making. Learning from the console's RAM can be more challenging, as the agent needs to extract meaningful information from non-visual data representations, which may require more complex data processing and feature engineering.

2.

The network's input layer in the starter code is a Conv2d layer, which takes an input tensor with the shape of the game's observation space (number of channels, height, width). This layer is responsible for processing the 84x84 pixel game display. The output layer is a Linear layer that outputs a tensor with the same length as the number of actions available in the game's action space. The output layer represents the Q-values associated with each action given the current input state. The agent will choose an action based on these Q-values.

3.

The purpose of the following lines is to implement the epsilon-greedy exploration strategy:

if random.random() > epsilon:

...

else:

action = random.randrange(self.env.action\_space.n)

This strategy helps balance exploration and exploitation. If a random number is greater than the epsilon value, the agent chooses the action with the highest Q-value (exploitation). If the random number is less than or equal to epsilon, the agent selects a random action (exploration). As training progresses, epsilon typically decreases, making the agent rely more on its learned knowledge and less on random exploration.

4.

To compute the Q-value and choose an action given a state, you can modify the code in the act function between lines 50-53 as follows:

if random.random() > epsilon:

state = Variable(torch.FloatTensor(np.float32(state)).unsqueeze(0), requires\_grad=True)

q\_values = self.forward(state)

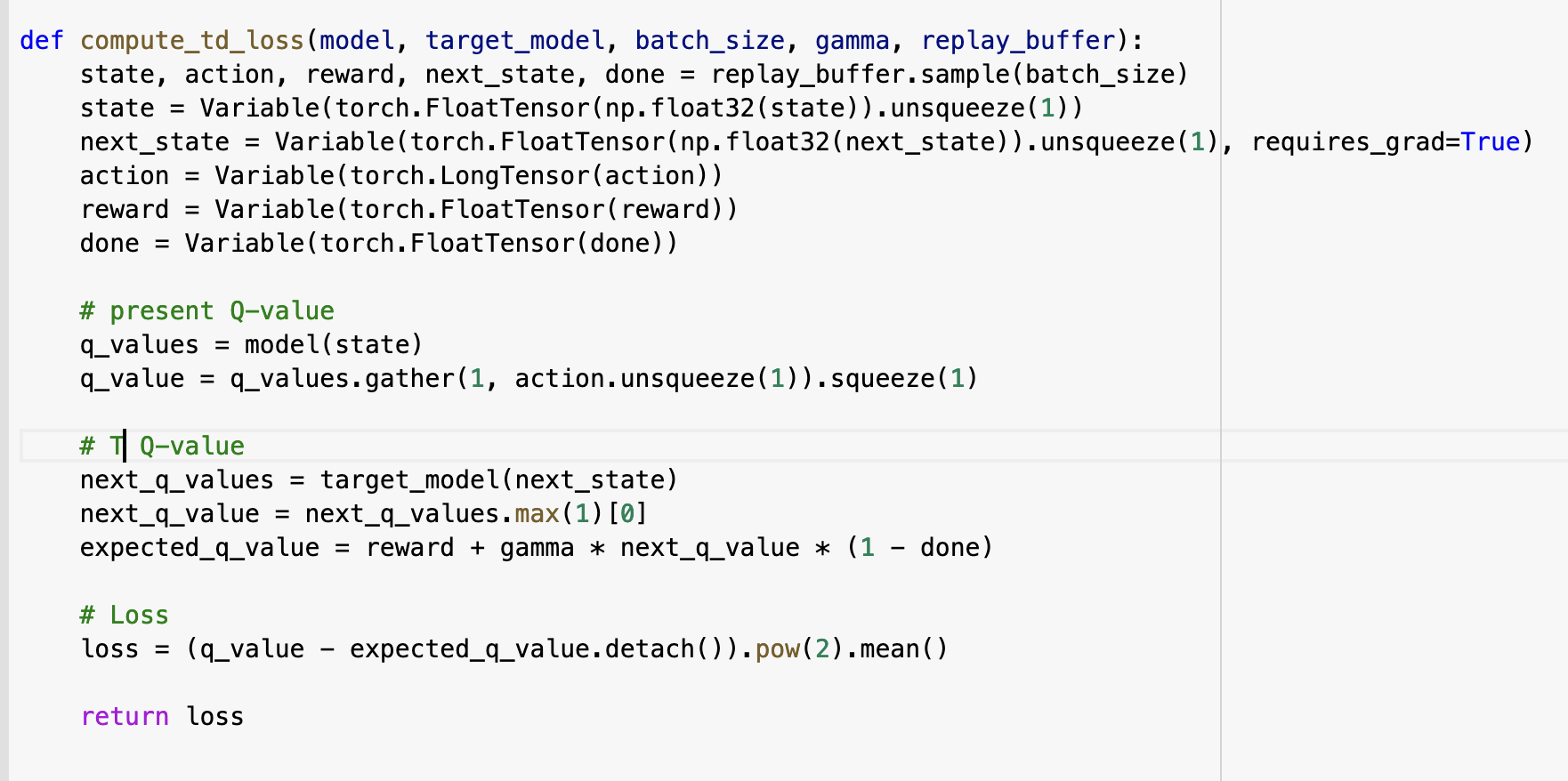
action = q\_values.max(1)[1].data[0]

else:

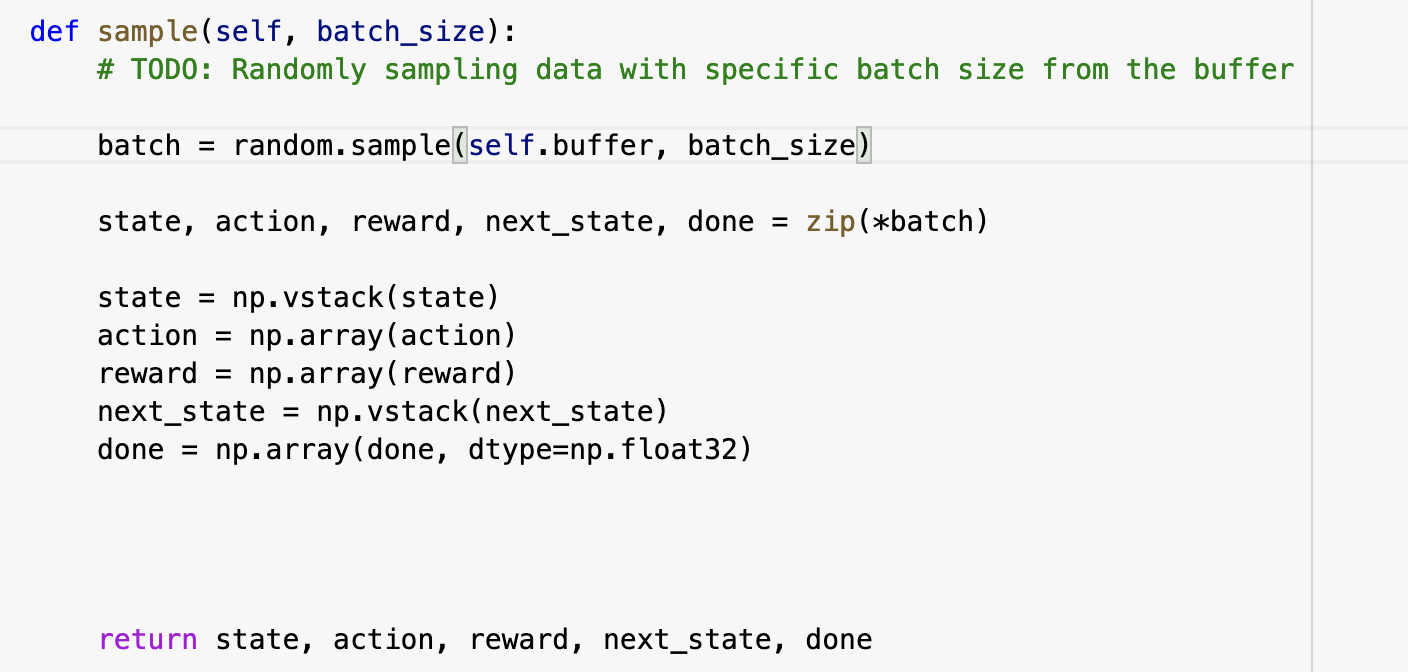
action = random.randrange(self.env.action\_space.n)

This code computes the Q-values for the given state by passing the state through the model's forward pass (q\_values = self.forward(state)). Then, it selects the action corresponding to the maximum Q-value (action = q\_values.max(1)[1].data[0]).

Level 2

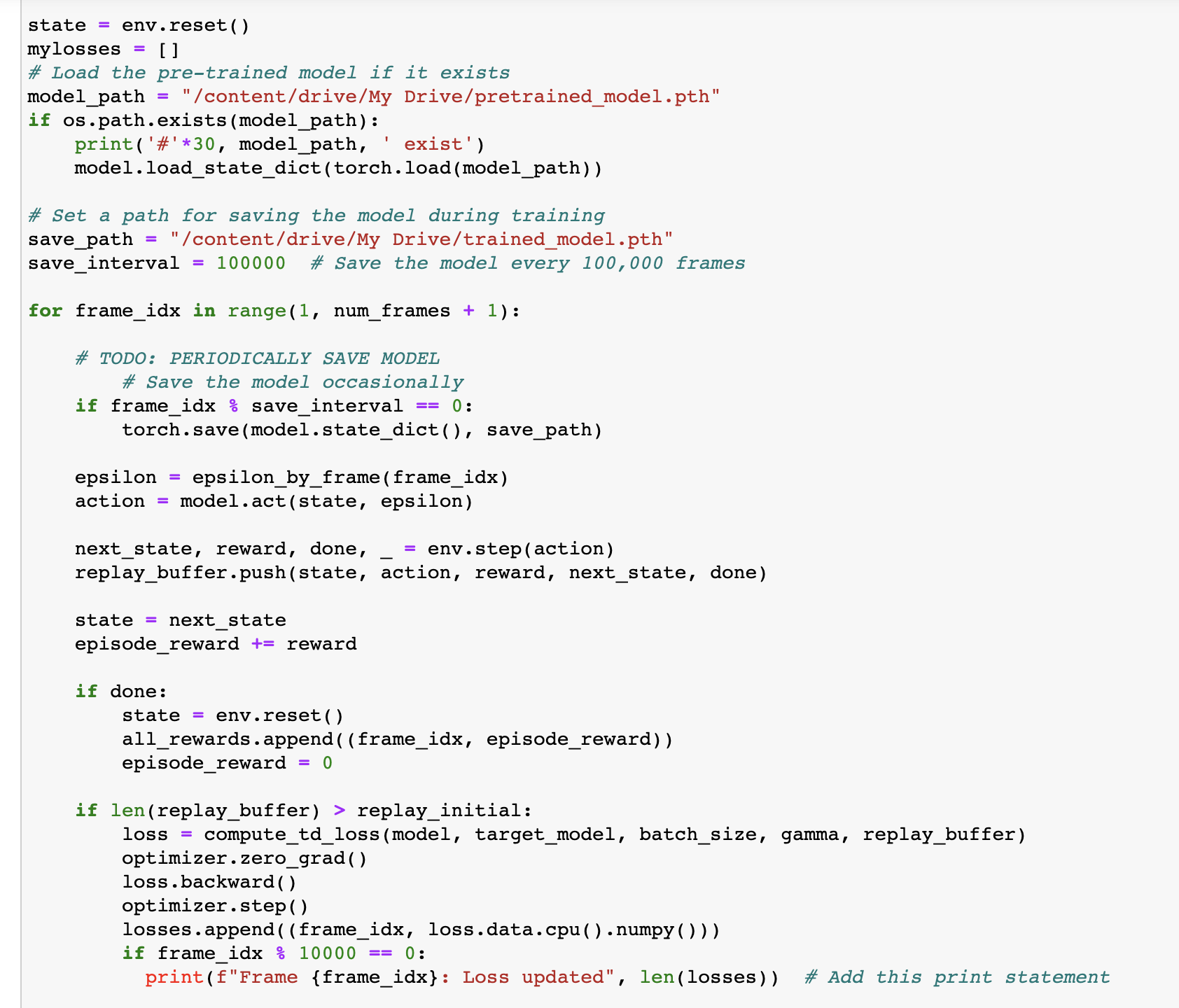


Level 3



Level 4

4.1



4.2

pasted-image.tiff

4.3

Yes , I train it on colab( I pay for pro)

Bonus Level

Q1：There are two parts to the bonus level to get extra credit. One is a more programming intensive portion about explanation, and the other is a conceptual question about your model’s behavior.

1.

In this project, judging whether the model is overfitting requires observing the performance of the model during training. If the model performs well on the training set, but performs poorly on the test set or the actual game environment, it may be overfitting. One of the ways to judge whether a model is overfitting is to divide the training process of the model into a training phase and a validation phase, and then compare the losses and rewards of these two phases. If the training loss keeps decreasing, but the validation loss no longer decreases or even increases, then there may be overfitting. In the context of reinforcement learning, overfitting means that the agent overfits to a specific state of the training environment without learning to generalize to new, unknown states.

2.

Overfitting is the phenomenon in which a machine learning model performs well on training data but performs poorly on new, unseen data. Overfitting indicates that the model is overly complex, capturing the noise in the training data rather than the true underlying patterns. In the context of reinforcement learning (RL), overfitting can manifest itself as an agent achieving high scores in a training environment but failing to perform to the same level in an actual game environment. This is because the agent over-adapts to the specific state and dynamics of the training environment during the training phase, and fails to generalize in the new environment.

To determine whether a model is overfitting requires careful observation of losses and rewards during training. If the training loss keeps decreasing, but the validation loss stops decreasing or starts increasing, this may indicate that the model is overfitting. In our model, we need to observe the loss and reward during training and compare it to the performance in the actual game environment. If the training and validation performance are similar and better, then the model is not overfitting. Poor performance in real game environments may indicate model overfitting. In this case, we need to take measures to alleviate overfitting, such as using more complex strategies, increasing the exploration rate, using regularization techniques, etc.

Q2

1.

In order to collect 512 features of 1000 randomly drawn frames and related auxiliary information, we first need to extract these features from the trained deep learning model. Auxiliary information can include the position of the agent, the position of the ball, etc., which may be of great significance to our understanding of the decision-making process of the model.

2.

After collecting features and auxiliary information, we can project these features into 2D or 3D space using dimensionality reduction techniques such as PCA, CCA, MDS, ISOMAP or LLE. This will help us visualize and analyze these features in order to understand how the model makes decisions based on the input state. After drawing the projections, we can color-code the projections according to the collected auxiliary information, so that it is easy to observe the performance of the model on these information.

3.

After analyzing the projections, we can draw conclusions about what the model has learned. These conclusions are not necessarily positive, and it is also possible that we find that the model does not fully consider auxiliary information. For example, if the projections show clear clusters or patterns related to auxiliary information such as agent positions, we can consider the model to have learned to make decisions based on this information. If there are no obvious patterns in the projections, then we may conclude that the model is not making good use of side information.

By plotting the projections and color-coding them according to auxiliary information, we can better understand how the model utilizes the input features when making decisions. These observations and conclusions will help us improve the model for better performance. Also, including the plotted projections and conclusions in the report helps us earn extra bonus points.