Deep network models and hyperparameter settings:

My deep neural networks I have used as the prediction model and target model are the one given in the code previously and the hyperparameters are the same as given in the program which implements a deep neural network model for Q-learning, which is used for reinforcement learning in an environment with discrete action spaces. The model takes in an input image of shape input shape (defaulting to (84, 84, 3)) and outputs a Q-value for each of the possible actions (size action size, defaulting to 4). The Q-value represents the expected future reward of taking that action in the current state.

The model architecture follows the one presented in the original Deep mind DQN paper, with a few simplifications assuming a single frame input. It consists of three convolutional layers with increasing depth and decreasing filter sizes, followed by a fully connected layer and a final output layer with linear activation. The convolutional layers use the ReLU activation function, while the fully connected layer uses the ReLU activation as well. The optimizer used for training is RMSprop with a specified learning rate.

Overall, this model is a standard deep neural network model used for Q-learning in reinforcement learning problems with image input.

This code implements a DQN agent for training a reinforcement learning agent to play Atari games using deep neural networks. The agent uses an epsilon-greedy exploration strategy to select actions and stores experiences (state, action, reward, next state, done) in a replay buffer.

The agent contains two Q-networks: a prediction model and a target model. The prediction model is used to select actions during training, and the target model is used to calculate the target Q-values for updating the prediction model. The weights of the target model are updated periodically to match those of the prediction model.

The Q-network architecture consists of three convolutional layers followed by two dense layers. The architecture is the same as the one used in the original Deep mind DQN paper, but with only one input frame instead of four. The model is compiled using the Huber loss function and the RMSprop optimizer with a learning rate of 0.001.

The hyperparameters for the agent are as follows:

state_shape: The shape of the state space for the Atari game, which is assumed to be a 2D RGB image of size (84, 84, 3).

action_size: The number of actions in the game, which is typically 4 for Atari games (up, down, left, right).

batch_size: The number of experiences to sample from the replay buffer during training.

gamma: The discount factor used to calculate the target Q-values for updating the prediction model.

learning_rate: The learning rate used by the Adam optimizer to update the prediction model weights.

epsilon: The initial value of the exploration rate used in the epsilon-greedy strategy for selecting actions.

epsilon_min: The minimum value of the exploration rate used in the epsilon-greedy strategy.

epsilon_decay: The factor by which the exploration rate is decreased after each training episode.

These are the hyperparameter values: batch_size=32, gamma=0.99, learning_rate=0.001, epsilon=1.0 (initial), epsilon_min=0.01, epsilon_decay=0.995

Overall, this code implements a basic DQN agent with a simple Q-network architecture for training a reinforcement learning agent to play Atari games.

DRL agents:

The DRL agent I implemented performed better than the previous action implementation but it is not the best performance than the previous works in this course.

In this particular case, the agent is implemented as a class called DQNAgent. The __init__ method initializes the agent with the necessary parameters, such as the environment, the state shape, the action size, the batch size, the discount factor (gamma), the learning rate, the exploration rate (epsilon), the minimum exploration rate, and the exploration rate decay. It also creates a replay buffer to store experiences, initializes neural networks for predicting Q values and updating target network weights, and compiles the prediction model with the mean squared error loss and the Adam optimizer.

The ***act method*** chooses an action using an epsilon-greedy exploration strategy. If a random number generated between 0 and 1 is less than the current exploration rate, the agent selects a random action. Otherwise, it uses the prediction model to select the action with the highest predicted Q value.

The ***remember method*** adds the experience to the replay buffer, which consists of the state, action, reward, next state, and done flag (indicating whether the episode is finished).

The ***train method*** samples a batch of experiences from the replay buffer and calculates the target Q values for the batch using the target model. It then updates the Q values for the batch in the prediction model and trains the prediction model on the batch of experiences.

The ***update_target_model method*** sets the weights of the target model to the weights of the prediction model.

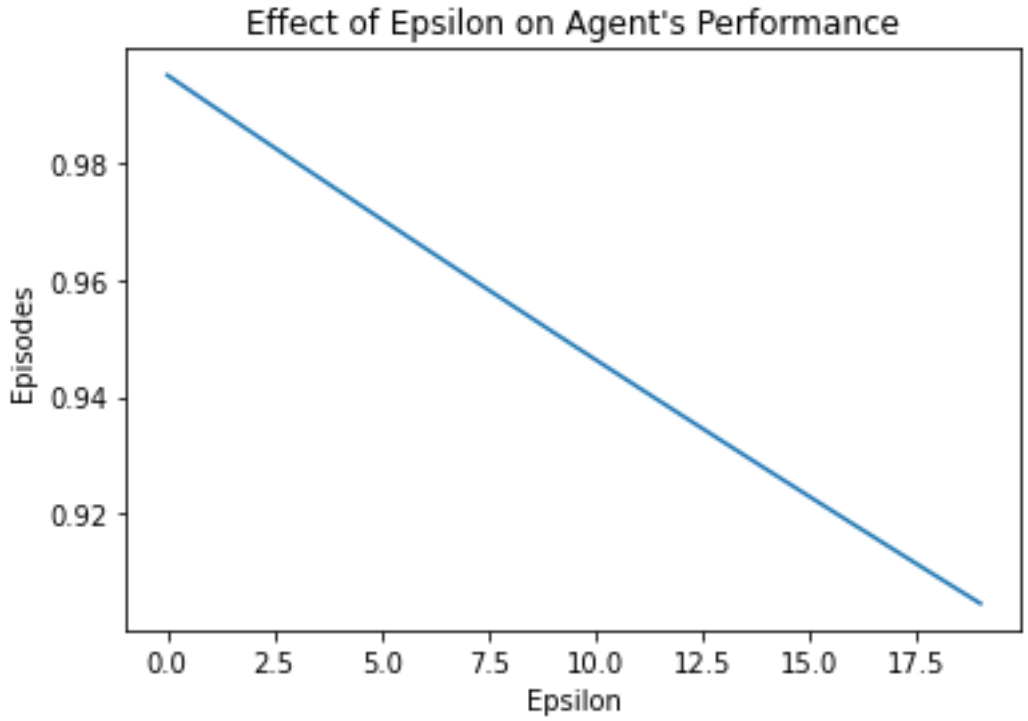
The ***decrease_epsilon method*** updates the exploration rate by multiplying the current exploration rate by the exploration rate decay factor.

At first, I was able to train for only 2 episodes since I was figuring out what to do it was quite hard to figure out. The reason for that was the action function I was doing before I didn't pass the environment in the class DQNAgent. It wasn't able to choose the best action but after passing the environment inside the action function I was able to train for twenty episodes.

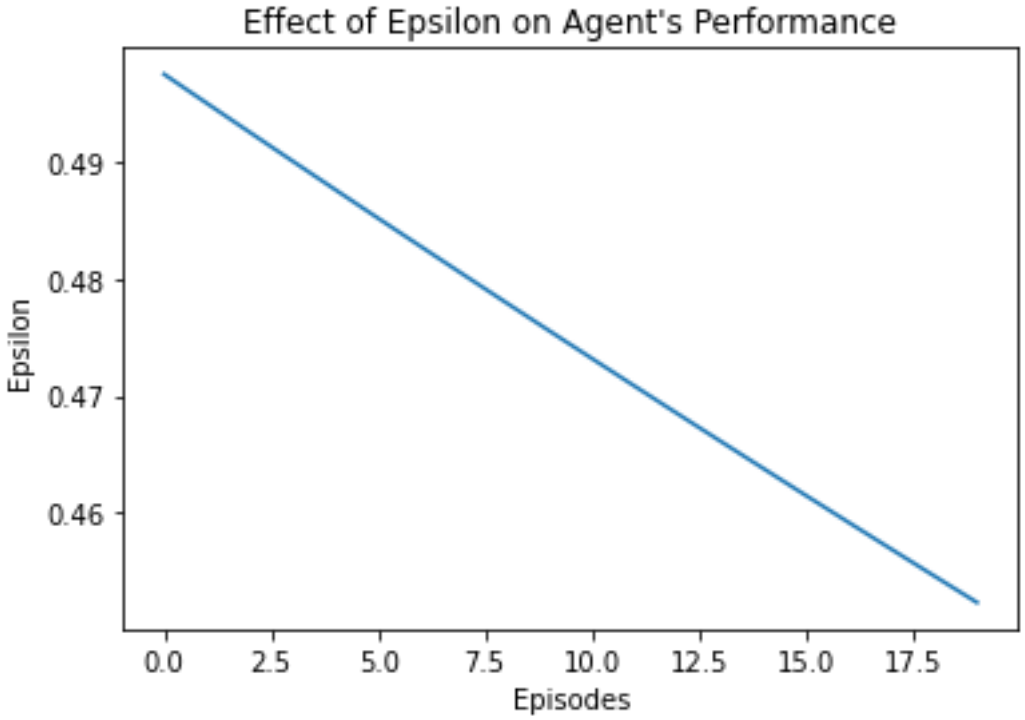
To train 20 episodes it took 20-30 mins with the collab's GPU. Since it allots only few hours in GPU, I was not be able to go more episodes but if it trains for more episodes, I think it might perform better.

Observations: The DRL agent tries to predict the ball movement and moves to get there but it is not reaching at the needed time.

Epsilon = 1.0:



Epsilon = 0.5:



General reflections:

The difficulty of this assignment was the toughest among all as everyone one knows. My rating for this assignment on a scale of 1-10 was 10/10. But I learned a lot from this course going forward towards a better path in the future. If I try next time, I would try to start debugging the code before starting my DQNAgent because I implemented the code and tried to debug what is the shape of the inputs and because of that a lot of time was taken. Especially, the GPU runs out before I try more things it was hard to do.