REINFORCEMENT LEARNING USING DQN

For this project, I was required to score at least **200** rewards for the **CART POLE** game. To achieve this, I used **DQN** (Deep Q Networks), which are basically neural networks for implementing the reinforcement learning. In the network, I used two fully connected layers with **300** neurons in each layer. The output layer outputs policy values for each possible action that can be taken for this game, that is, moving the cart right or left represented by 1 and 0 respectively.

After the output layer gives the policy values, I apply softmax activation function to output the probabilities for various actions that can be taken. To train the policy model, we must play the game for a certain number of episodes so that the training data can be generated. Each episode can be played for a maximum of **1000** timesteps. The rewards collected during each such episode are fed into a function called **"update_network_parameters ()"** along with the log of probabilities with which the possible actions were taken. This function calculates the loss and sends it to a function called **"update_weights ()"** which propagates the loss backwards to calculate gradient and update the model weights. In this way, our model learns to suggest optimal actions that must be taken to play the game efficiently. The model automatically stops the training when it has learned to score specified number of rewards.

When we run the code, it asks the user if he wants to use pre-trained model or train a new one. When we run the code for the first time, it asks to enter the following parameters:

- **num_episodes:** Here, we must specify the number of episodes foe which we wish to train the model. I trained it for **1500** episodes.
- **target_avg_reward:** This is to specify what is the average target average reward that we wish to achieve. I set it to **200.** Also, the average rewards are calculated over a span of **50** episodes.
- **verbose:** This can be set **True or False** depending on whether we want to see the rewards collected during the training process. By default, it has been set to **True.**
- **learning_rate:** This is parameter that controls the rate which our model learns. I have set it to **0.001.**

Using the above set of parameters, the model trained for **1000** episodes and it was able to achieve the desired target in **980** episodes. Along with this, I also plotted the rewards earned during each episode and average number of rewards over a span of **50** episodes. This helps in visualizing the model's progress.

PLEASE NOTE that if the user chooses to use the pre-trained model and the file does not exist in the directory, then the model would flash the message that the model does not exist and the user is required to train the model first. But, if the model exists, then it would simply load the model and ask the user to input the number of episodes for which he would like to play the game.