Gautham Rajesh
4/20/2020

Report

Project Information

The project involved working in a Unity environment where the agent had to collect yellow bananas while avoiding the blue bananas. The agent got a reward of +1 for collecting a yellow banana while the reward was -1 for collecting a blue banana. Here are the values for the state and action spaces:

- State space: 37 dimensions (velocity, perception of objects in front of the agent, etc.)
- Action space: 4 dimensions (moving forward/backward/left/right)

The environment is considered solved when the agent has achieved an average reward of +13 over 100 episodes.

Learning Algorithm

The learning algorithm I chose was a Deep Q-Network. Essentially, the algorithm tries to find the optimal action-value policy, like other reinforcement learning algorithms. However, instead of having the model learn only after an episode is complete, a deep-Q-network would allow the agent learn after every action taken. This is possible because this learning algorithm uses the maximum value of the Q-table based on the next state and the current action. The equation below sums it up:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

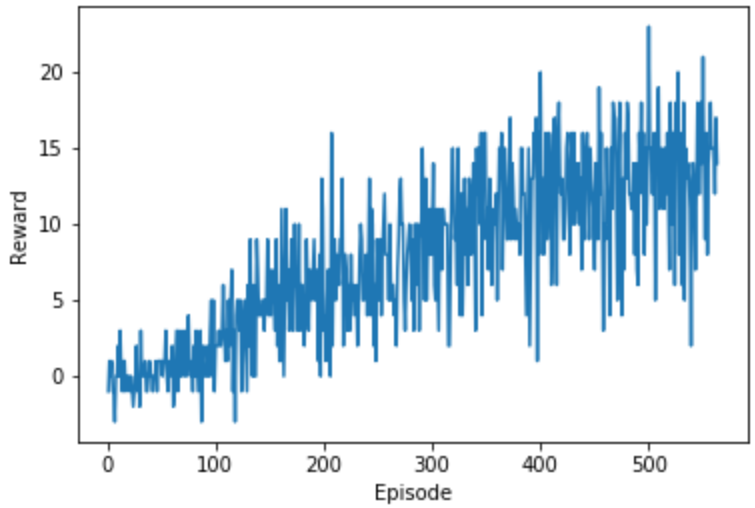
Here, we also see an alpha value and a discount factor applied to this calculated reward to provide some sort of regularization in the calculation. This calculation would be simple enough if the state and action spaces were discrete and small; however, in this case, the state space is relatively large with 37 dimensions. Thus, deep-Q-networks harness the power of neural networks to simplify this calculation—the network takes in states as its input, and outputs the actions that should be taken. The parameters for this network are trained with a mean square error loss function, where the error is calculated from the difference between the calculated state-action policy and the optimal state-action policy. Here are the hyperparameters I used for my network:

- 2 hidden layers
- 64 hidden units
- ReLU activation on all layers except the output layer

- Adam optimizer with a learning rate of 0.0005

For the other parameters, I set my discount rate to 0.99, epsilon_start to 1, epsilon_decay to 0.995, and epsilon_min to 0.1.

Results



The model reached an average score of 13.03 in 463 episodes.

Future Work

Two ways this model could be improved include implementing a Double DQN and prioritized experience replay. The Double DQN essentially relies on two networks: one that chooses the best action to take based on the current state, and one that evaluates that action. This prevents an overreliance on the max operator for calculating potential reward for a given state-action pair. The first model would be the one that has its parameters updated, while its performance is being checked by the second model. Prioritized experience replay builds on top of the existing experience replay system in DQN's, where the model relies on experiences (state-action pairs and their corresponding rewards) to shape its decision-making on which action to pick from a given state. *Prioritized* experience replay relies on modifying the TD (temporal difference) error between the predicted reward and the actual reward gained by the agent by a small weight. This allows more experiences, even those with relatively low TD error, to be used by the model during training. This could reduce the chance of overfitting to the experience samples with higher TD error (the ones picked more often).