Agent to Navigate and Collect Bananas in a Large, Square World.

September 6, 2024

---

1. Project Objective

The goal of this project was to train an agent to navigate and collect bananas in a large, square world. The agent received a reward of +1 for collecting yellow bananas and -1 for collecting blue bananas, with the objective of collecting as many yellow bananas as possible while avoiding blue ones.

2. Environment Details

The state space consisted of 37 dimensions, including the agent's velocity and ray-based perception of objects around its forward direction. The agent had to select from four discrete actions: moving forward, backward, turning left, or turning right. The task was episodic, and the agent was considered to have solved the environment if it achieved an average score of +13.0 over 100 consecutive episodes.

- **Simulation Environment**

The UnityAgents have utilized been API to simulate the environment, which significantly reduced the development time.

3. Model Architecture

a Deep Neural Network (DNN) was employed as a function approximator for the Q-function, also known as Deep Q-learning. The DNN architecture consisted of three linear layers: The first layer had an input dimension of 37 and an output dimension of 128. The second layer had an input dimension of 128 and an output dimension of 64. The third layer had an input dimension of 64 and an output dimension of 4, corresponding to the number of actions. Iused the ReLU activation function for each layer. The model was trained using Gradient Descent with the Adam optimizer to update the weights.

4. Agent Architecture

The agent employed in this project was a Q-learning agent, utilizing a 3-layered neural network as a function approximator. Q-learning is a renowned algorithm in reinforcement

learning that learns the action-value function for a given policy. A key advantage of Q-learning over SARSA is its ability to directly learn the optimal Q-value, rather than oscillating between evaluation and improvement.

- **Addressing Instability**

To mitigate the instability issues inherent in non-linear function approximators, two modifications has been implemented:

- **Experience Replay**: a buffer memory has been utilized, known as a replay buffer, to store experience tuples (comprising state, action, reward, and next state). The agent randomly samples experiences from this buffer, enabling it to learn from the same experience multiple times. This approach is particularly beneficial when encountering rare experiences.
- **Fixed Q-values**: To eliminate instability, a separate network with an identical architecture has been employed. The target network is updated slowly with the hyperparameter TAU, while the local network updates aggressively with each update (soft update).
- **Epsilon-Greedy Policy**

The learning algorithm incorporates an Epsilon-Greedy policy to select actions during training. Epsilon specifies the probability of selecting a random action instead of following the "best action" in the given state, thereby facilitating exploration-exploitation tradeoff.

- **Agent Class**

The agent class defines the agent's behavior, including how it acts given an action, learns from a time step, updates the network every UPDATE_EVERY time steps, and stores experiences in the memory buffer.

- **Replay Buffer Class**

To facilitate experience replay, a memory buffer class has been defined. An object of ReplayBuffer initializes a deque of maximum length BUFFER_SIZE to store experience tuples. It provides methods to store tuples and sample a set of tuples given a BATCH_SIZE.

5. DQN Algorithm

The DQN method encapsulates the DQN algorithm, which returns a list of scores for all episodes. The algorithm terminates when the reward value reaches or exceeds 15.0, indicating successful completion of the task. An epsilon decays 2 of 0.995 starting from an epsilon of 1.0 and ending at an epsilon of 0.01 has been use.

The best hyperparameters were:

BUFFER_SIZE = int(1e5)

BATCH_SIZE = 64

GAMMA = 0.99

TAU= 1e-3

LR= 5e-4

UPDATE_EVERY = 4

6. Result

An average score of 15.0 over the last 100 episodes was achieved at 685 episodes

```
In [10]: fig = plt.figure(figsize=(13, 10))
         ax = fig.add_subplot(111)
         plt.plot(np.arange(len(scores)), scores)
         plt.ylabel('Score')
         plt.xlabel('Episodes')
         plt.show()
```