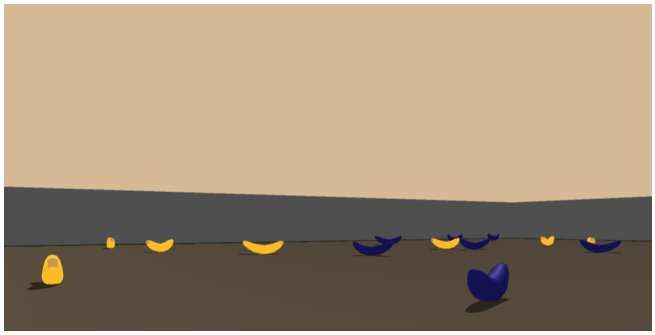
Deep Q-Learning Agent in a 3D Unity Environment

Benjamin Li

May 6, 2020

The intent of this project is to train an agent to maximize its score in a 3D Unity game environment. To do this, a simple neural network with 3 hidden-layers was used to implement **Q-Learning**. By the 400th training episode, the agent's average score for 100 consecutive trials beat the pass-benchmark of 13.

To use a neural net, facets of Q-Learning, such as the **Bellman Equation** are adapted to new forms (i.e. loss function)—enabling efficient RL in this environment with a 37 dimensional state-space and 4-dimensional action space.

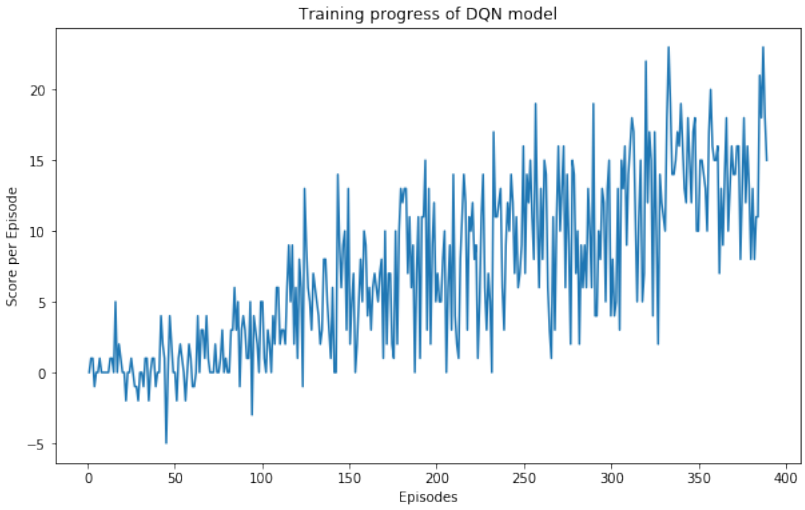


+1 for collecting a yellow banana, and -1 for collecting a blue banana.

1 Model Architecture

- Input Layer takes a 1x37 state-vector
- 3 hidden layers with 128/64/32 units using **ReL activation**
- Output layer produces 1x4 vector consisting of the Q-value of each action

Hyper-parameters were selected based on [1][2]: $\alpha = 5e-4$, $\epsilon_0 = 1.0$, $\epsilon_{min} = 0.01$, $\epsilon_{decay} = 0.99$. The learning-rate (α) determines the impact of each *training example* and ϵ related parameters concern exploration of environment dynamics (choosing a random action) versus exploiting the current optimal solution.



1.1 Next Steps

This RL model uses state-vectors with relatively comprehensive data (i.e. velocity). A natural next step is to create a model that takes raw pixels (i.e. snapshot of agen't POV in game at each time-step), like in [2]. In this case, the algorithm may stay the same, but certain image preprocessing techniques such as converting to grey-scale are necessary for efficiency; and, hyperparamters may need adjustment. [2]

2 References

1. Udacity internal material on Deep RL
2. "Human-level control through deep reinforcement learning"
3. blangwallner/Udacity-Deep-Reinforcement-Learning-ND—Project-1—Navigation