Project 3 - Problem 2: Reinforcement Learning

Yufei Hu (404944367)

February 14, 2018

1 Introduction

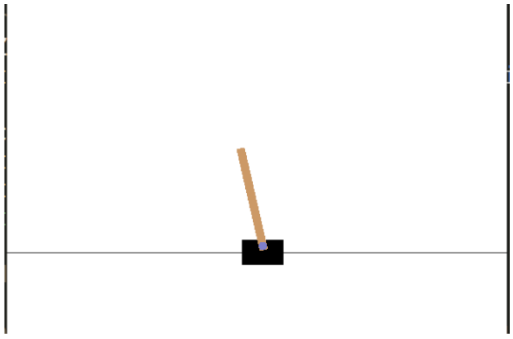

Figure 1: OpenAI gym CartPole-v0

In this problem, I implemented two classic Reinforcement Learning techniques using deep neural network as function approximation. The environment is 'CartPole-v0' from OpenAI gym. CartPole or 'Inverse Pendulum' as shown in Figure 1 is a classic optimal control problem. It is such an easy game that the agent only need to balance the pole on top of the little cart as long as possible. I have implemented a Reinforcement Learning agent which can keep balancing the pole.

2 Implementation

2.1 Deep Q-network:

In this section, I implemented Q-Learning with neural network as function approximation. The formulation of Q-Learning is shown as Equation 1. Besides Q-learning, I also implemented Experience Replay and ε-greedy to make the whole algorithm work.

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \tag{1}$$

where γ is the discounted factor.

2.2 Input:

'CartPole-v0' environment in OpenAI Gym only provides a tuple (state, action, reward, next state, done) as feedback. The observed state is a vector describing current condition of the cart including position, speed, acceleration etc. It does not provide image as observation. I have trained two models: the first one taking images as input and the second one taking states as input.

The architecture of the neural network for the first model is concluded as follows:

Table 1: First model neural network architecture

Block	Parameters
Conv - Batch Norm - ReLU	filter size = $16 \times 3 \times 5 \times 5$, stride = 2
Conv - Batch Norm - ReLU	filter size = $32 \times 16 \times 5 \times 5$, stride = 2
Conv - Batch Norm - ReLU	filter size = $32 \times 32 \times 5 \times 5$, stride = 2
FC	filter size = 448×2

The architecture of the neural network for the second model is concluded as follows:

Table 2: Second model neural network architecture

Block	Parameters
FC	filter size = 4×256
FC	filter size = 256×2

The reason for choosing a rather small architecture for the second model is because the input to the second model is much simpler than the one for the first model, thus it requires much less model capacity to fit the input data. Since the input to the first model is an image, Convolutional layers are used in the first model for staying consistent to such input format.

The hyperparameters I used for training both networks are: number of epochs is 210, batch size is 64, and the learning rate is 0.001.

3 Results

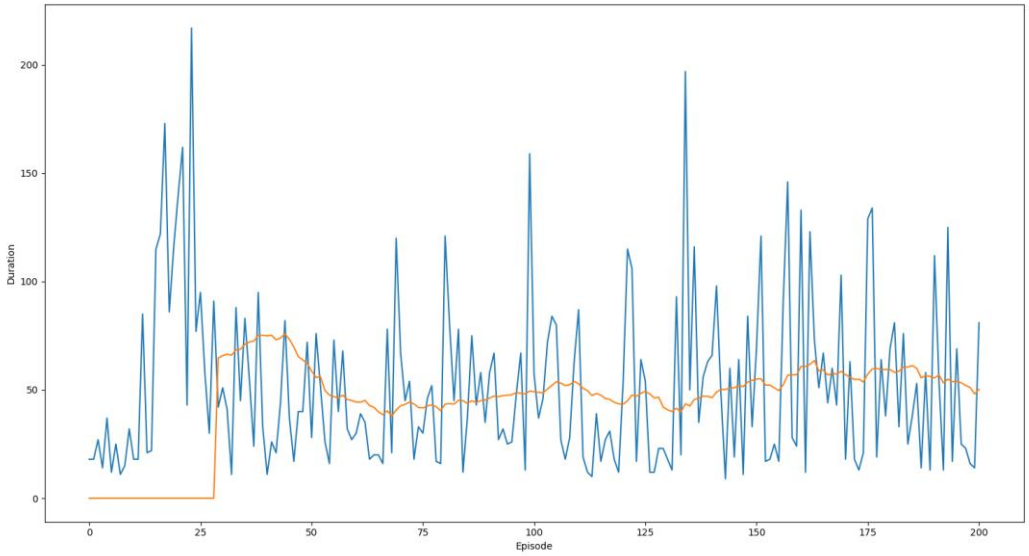

Figure 2: Reward (blue) and the average for the last 30 rewards (yellow) for image input model

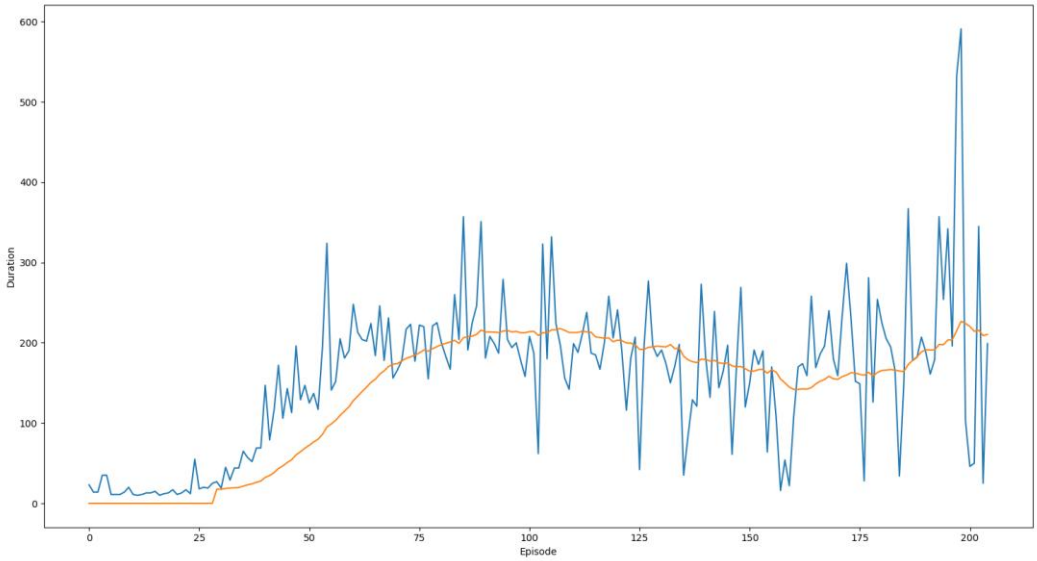

Figure 3: Reward (blue) and the average for the last 30 rewards (yellow) for state input model

As seen in the two figures above, the state input model performs much better than the image input model. The former one has an average reward of 200 while the latter one only has an average reward of 50. The reason for this performance discrepancy is the image input model has to derive certain critical features including velocity and acceleration indirectly from the input image by itself while state input model directly fetch those important information from the environment.

Another observation is that the image input model trains much slower than state input model. The reason is straightforward as the input for the former model has to be rendered during each training iteration while the latter model has no such concern. Another major reason is because the function approximation neural network for the first model has deeper and more complicated layers than the second model. This model complexity could also slow down the training speed.