MLDS HW3

B03705013 資管四 徐逸然

1. Describe your Policy Gradient & DQN model:

我參考的 Policy Gradient 教學這篇：https://github.com/keon/policy-gradient，我測試過其他不同的模型，例如增加一層 Conv2d，調整 Dense 的 neuron 數量至 256 等等，然而很神奇的只要改動一點就幾乎 train 不起來，因此最後我還是參照此篇的模型不做修改。

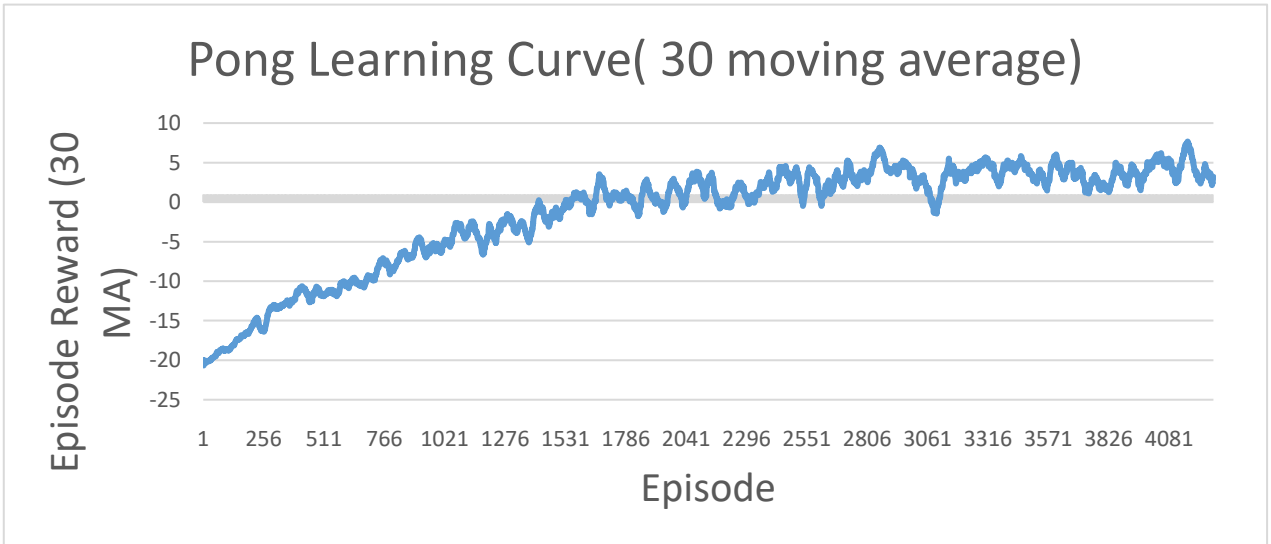
```
    model = Sequential()
    model.add(Convolution2D(32, 6, 6, subsample=(3, 3), input_shape=(80,80,1), border_mode='same',activation='relu', init='he_uniform'))
    model.add(Flatten())
    model.add(Dense(64, activation='relu', init='he_uniform'))
    model.add(Dense(32, activation='relu', init='he_uniform'))
    model.add(Dense(self.action_size, activation='softmax'))
    opt = Adam(lr=self.learning_rate)
    model.compile(loss='categorical_crossentropy', optimizer=opt)
    return model
```

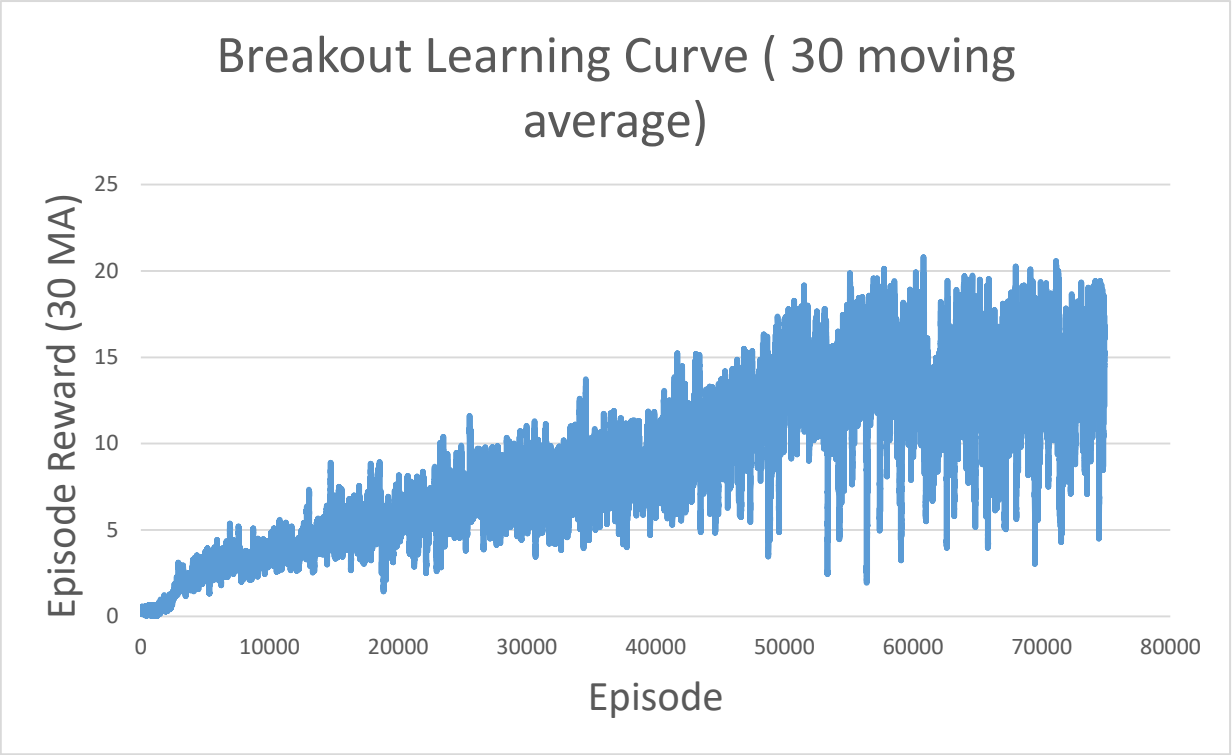
DQN 的模型比較複雜，我參考的是 https://github.com/coreylynch/async-rl 這篇。原本我採用另一篇 tutorial 的簡易模型，只有一個 network 負責訓練與 predict，做出來的結果僅能達到 15 分左右(test_dqn)，爬了許多篇之後才決定改這篇的方法。在我的 DQN 模型中有兩個 network, Q_network 與 target_Q_network，進行 Experience Replay 時，每個 Label 中 Q(S,A)那項是由 target_Q_network 預測出來的，而 fit 的時候則是 fit Q_network。每隔一段 timestep 才會用 Q_network 訓練得到的 weight 更新 target_Q_network 的 weight。Q_network 有兩個 input，state 與 action，根據一個 state 預測三種 action 的 reward 值，再 dot 給定的 action(one-hot vector)，假設預測的 reward 是[1,2,3]， 給定的 action 是[0,1,0]，Q_network output 就是 2。target_Q_network 則只需要 input state，並根據這個 state 預測三種 action 分別的 reward。可以發現若 Q_network 省去最後一個 dot 的步驟，則跟 target_Q_network 一模一樣，因此我用 build_model 這個函式來生成兩個 network 相同的部分，build_graph 則是定義所有會用到的變數以及 loss 與 optimizers 等等。

```python
def build_network(self, num_actions, w, h, c):
    with tf.device("/gpu:0"):
        state = tf.placeholder("float", [None, w, h, c])
        inputs = Input(shape=(w, h, c))
        model = Convolution2D(32, (3,3), activation='relu', padding='same')(inputs)
        model = MaxPool2D((2,2))(model)
        model = Convolution2D(64, (3,3), activation='relu', padding='same')(model)
        model = MaxPool2D((2,2))(model)
        model = Flatten()(model)
        model = Dense(512, activation='relu')(model)
        q_values = Dense(num_actions, activation='linear')(model)
        m = Model(inputs, q_values)
        # print(m.summary())
    return state, m
```

```python
def build_graph(self, num_actions):
    # Create shared deep q network
    s, q_network = self.build_network(num_actions, 84, 84, 4)
    network_params = q_network.trainable_weights
    q_values = q_network(s)

    # Create shared target network
    st, target_q_network = self.build_network(num_actions, 4, 84, 84)
    target_network_params = target_q_network.trainable_weights
    target_q_values = target_q_network(st)

    # Op for periodically updating target network with online network weights
    reset_target_network_params = [target_network_params[i].assign(network_params[i]) for i in range(len(target_network_params))]

    # Define cost and gradient update op
    a = tf.placeholder("float", [None, num_actions])
    y = tf.placeholder("float", [None])
    action_q_values = tf.reduce_sum(q_values * a, reduction_indices=1)
    cost = tf.reduce_mean(tf.square(y - action_q_values))
    optimizer = tf.train.AdamOptimizer(0.001)
    grad_update = optimizer.minimize(cost, var_list=network_params)

    graph_ops = {"s" : s,
                 "q_values" : q_values,
                 "st" : st,
                 "target_q_values" : target_q_values,
                 "reset_target_network_params" : reset_target_network_params,
                 "a" : a,
                 "y" : y,
                 "grad_update" : grad_update}
    return graph_ops
```

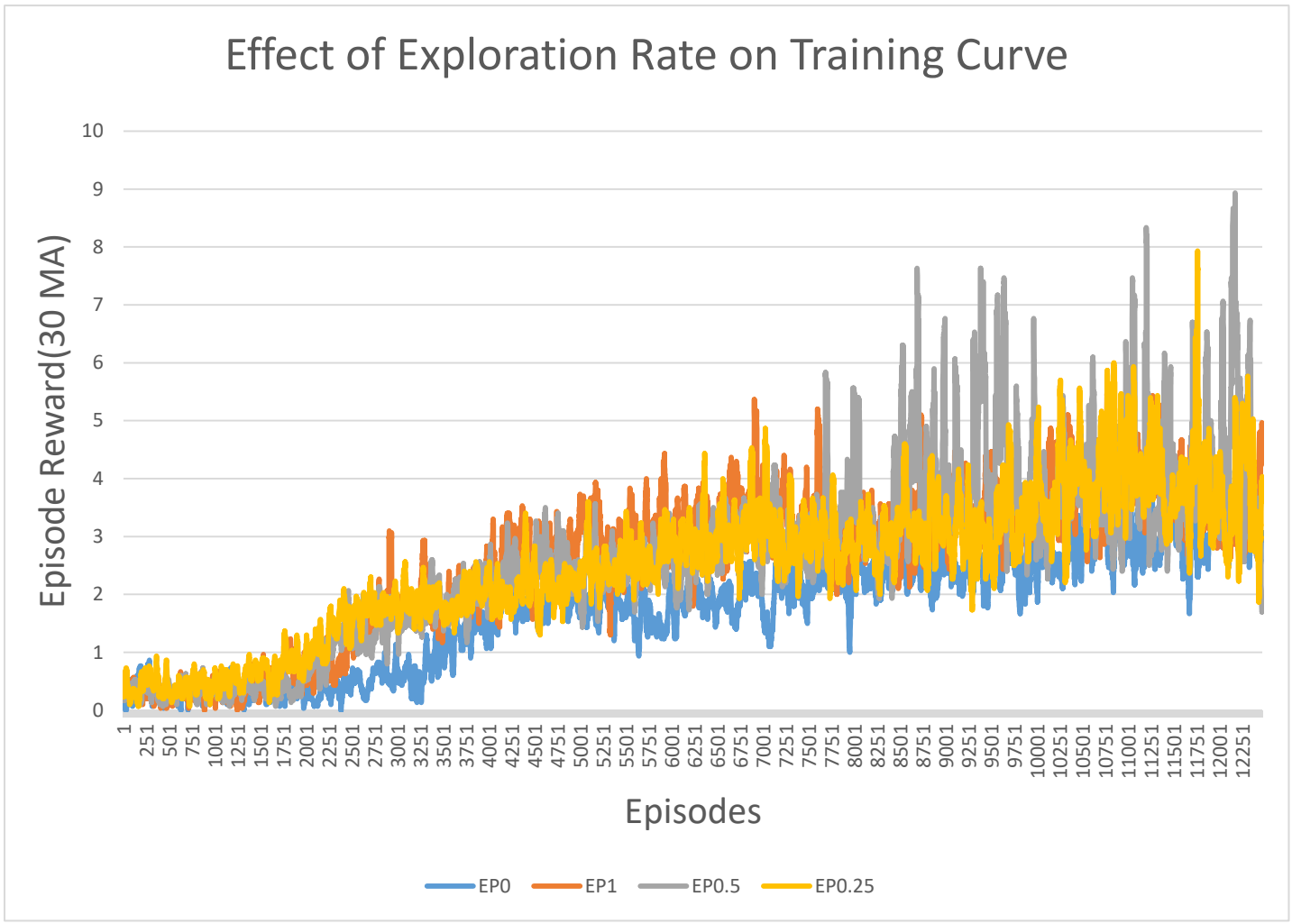
2. Learning Curve Plots :





3. Experimenting with DQN hyperparameters

一開始聽教授講解 DQN 時我就對 exploration rate 很有興趣，在模型初期還沒訓練好之前，就盡量讓他隨機走，等到開始有學到一些東西之後才逐漸降低 exploration rate，也就是相信模型的 output 是正確的選擇。以下是我針對四種不同的 Exploration Rate 做測試，EP 0 代表從頭到尾皆不做隨機 action，EP 1, EP 0.5, EP 0.25 則是一開始分別由 Exploration rate 1, 0.5, 0.25 開始每 episode decay 0.995 至 0.0001。做了約 12000 個 episode 之後我們可以發現 EP 0 的效果明顯最差，尤其是前期訓練的部分 Model 在前期還沒辦法正確 predict 出正確 action 時，會不斷預測出挑錯誤的行為而使 memory 裡儲存的 history 中有一堆都是沒有意義的，例 initial model 的時候如果”靜止”的 action 剛好 predict 的值比較高一些，則 model 不管怎樣都會 predict 出靜止，而使 memory 中大多數的 state 都對應到”靜止” 。要突破這個困境最好的辦法就是在前期隨機作出 action，而根據實驗的結果 Explore Rate 1, 0.5 與 0.25 做出來的效果不會差太多，也就是說前期只要有一定機率讓模型去冒險，就能夠使 training 的速度加快。


Effect of Exploration Rate on Training Curve
Episode Reward(30 MA)
10
9
8
7
6
5
4
3
2
1
0
1
251
501
751
1001
1251
1501
1751
2001
2251
2501
2751
3001
3251
3501
3751
4001
4251
4501
4751
5001
5251
5501
5751
6001
6251
6501
6751
7001
7251
7501
7751
8001
8251
8501
8751
9001
9251
9501
9751
10001
10251
10501
10751
11001
11251
11501
11751
12001
12251
Episodes
EP0
EP1
EP0.5
EP0.25