Homework3 Atari Game Playing Report

b03705027 鄭從德

1. Basic Performance (6%)

Policy Gradient Model on Pong:

a. Model

這個 model 是參考網路上別做出來的結果，只利用了一層的 conv2D，並且已經先把畫面中的重要部份切出來變成(80,80,1)的 input。

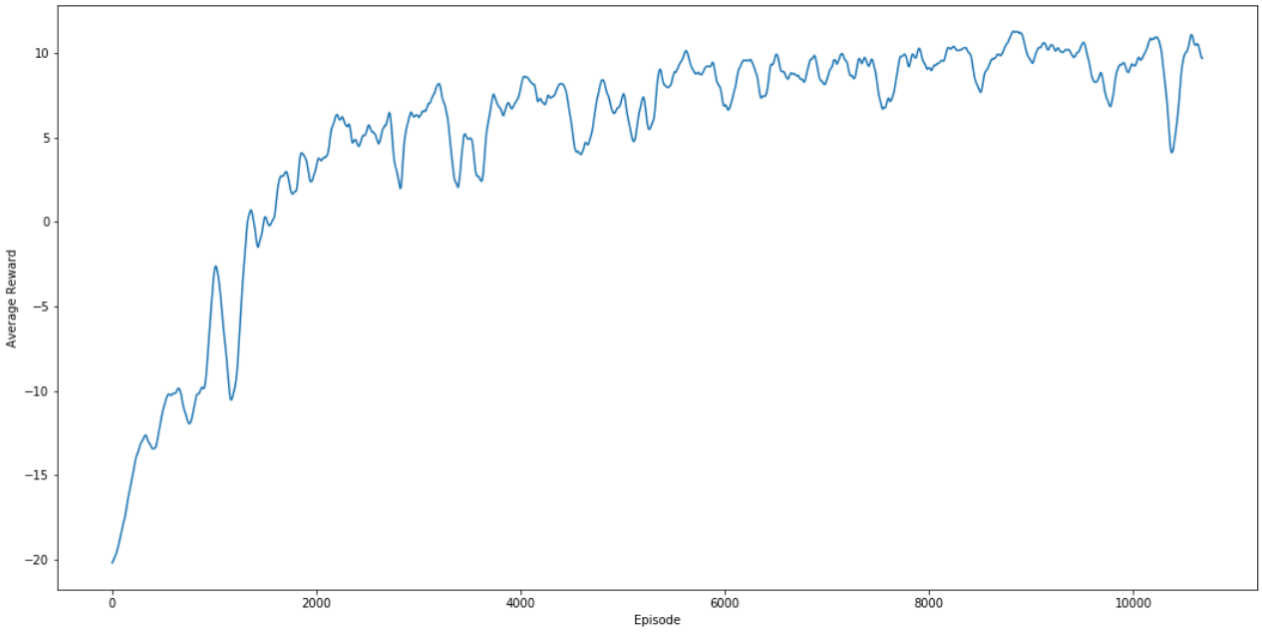
```
def build_model(self):
    model = Sequential()
    model.add(Convolution2D(32, (6, 6), activation="relu",padding="same",input_shape=(80, 80, 1),
    model.add(Flatten())
    model.add(Dense(64, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(32, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(self.action_size, activation='softmax'))
    opt = Adam(lr=self.learning_rate)
    model.compile(loss='categorical_crossentropy', optimizer=opt)

    return model
```

b. Best Parameters

gamma = 0.99, learning rate = 0.001

比較重要的只有這兩個 parameter，對於不同的 gamma 值，如果降低到 0.95 就會明顯的表現較差。

c. Learning Curve (100MA)：這個 model 還滿厲害的，在第 2000 個 episode 就可以跟電腦打平(0 分)，大概 train 到一萬個 episode 結果就收斂不再進步



d. Final Score: 7.833

DQN Model on Breakout:

a. Model：model 架構參考助教提供的參數，沒有啥特別：一共使用三層的 Conv2D 沒有 Maxpooling，最後接一個 kernel size = 512 的 dense

```
def build_network(self, num_actions):

    state = tf.placeholder("float", [None, 84, 84, 4])
    inputs = Input(shape=(resized_width, resized_height,agent_history_length))
    model = Convolution2D(32, (8, 8), stride=(4, 4), activation='relu', padding='same')(inputs)
    model = Convolution2D(64, (4, 4), stride=(2, 2), activation='relu', padding='same')(model)
    model = Convolution2D(64, (3, 3), stride=(1, 1), activation='relu', padding='same')(model)

    model = Flatten()(model)
    model = Dense(512, activation='relu')(model)
    q_values = Dense(num_actions, activation='linear')(model)
    m = Model(inputs, q_values)
    return state, m
```

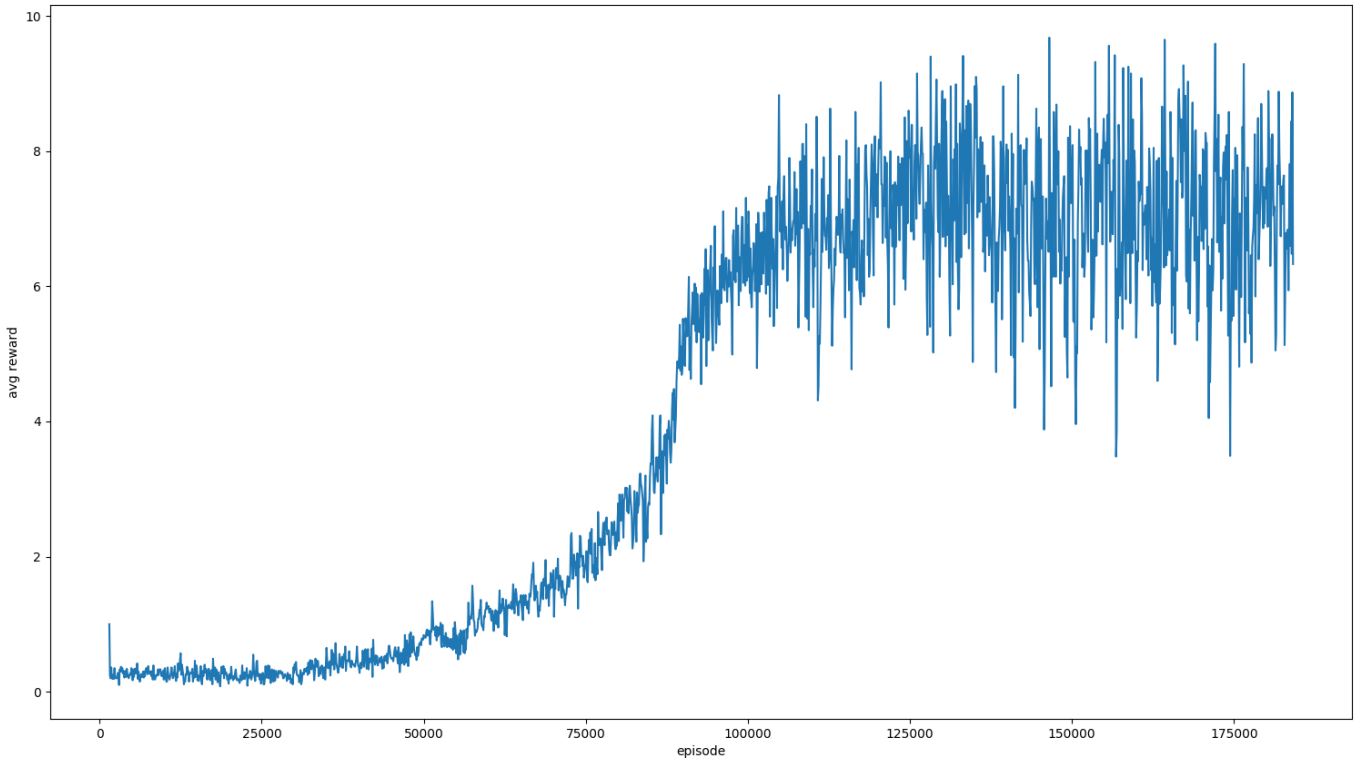
b. Best Parameters

learning rate: 0.00025, gamma: 0.99

target network update frequency: 10000

online network frequency: 4

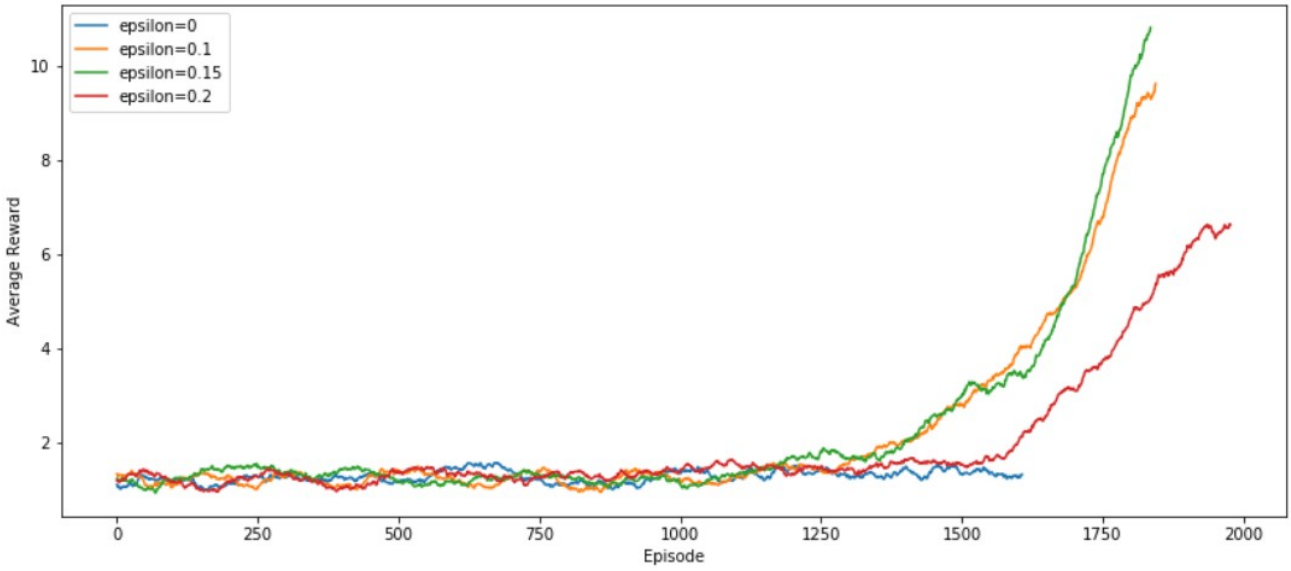
c. Learning Curve (100MA – clipped reward): 可以看出這個 model 在 clipped reward 到七點多之後就沒有明顯進步。



d. Final Score: 52.79

Experimenting with DQN hyperparameters (4%)

在這一題的實驗中，我做了 min_epsilon 的測試，也就是一樣的 decay rate，讓最終的 epsilon 停留在不同的地方。因為在網路上查了許多資料，看到大部分的人都建議直接把 min epsilon 設為 0.1，但心裡總覺得 0.1 這個數字不是很合理，所以就試著做這個實驗。下圖中，四條線分別是:從頭到尾沒有 epsilon、decay 到 0.1、decay 到 0.15、decay 到 0.2。(100MA)



從結果中我們可以看出來，從頭到尾都沒有 epsilon 的效果明顯最差。這是因為沒有 epsilon 的話，我們的 agent 並不會隨機亂走，因此可能完全不會 explore 新的玩法（從來就沒有發現）。

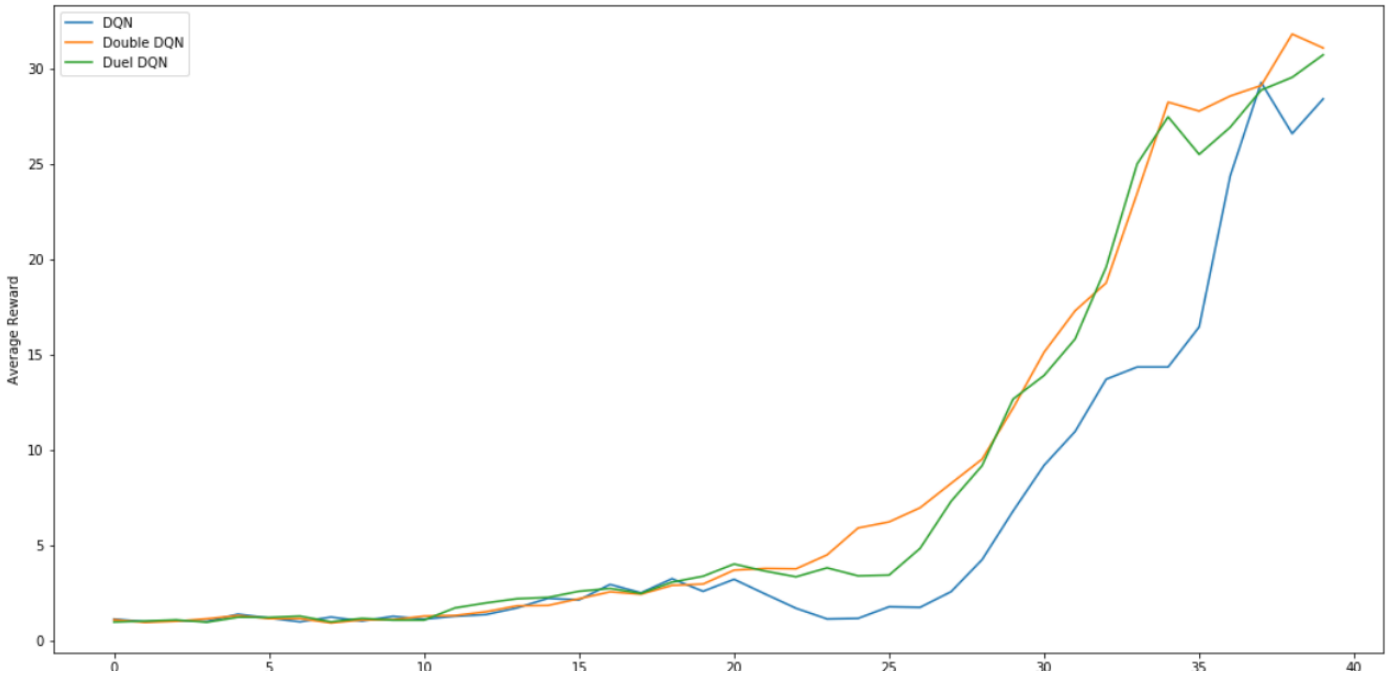
剩下三條線當中，讓我比較驚訝的是 min_epsilon=0.15 竟然有比較好的 performance。因為在網路上看大部分的 paper 還有範例，都是拿最終 decay 到 0.1 當作一個標準，沒想到實做出來他並不是最優秀的。

我們可以比較容易的解釋為何 min_epsilon=0.2 會炸爛，因為有五分之一的步伐亂走就感覺已經十分亂來了。但我原本預期做較小的 epsilon 可以得到比較好的結果(比較乖乖走)，但實際做出來竟然是界於 0.1 到 0.2 之間的 0.15 表現最佳，可能代表在剛開始分數爆衝的階段，0.15 這個數字十分幸運的平衡了「探索新的走法」以及「正確的選擇」。

Bonus

Bonus 1. Breakout: DQN vs Double DQN vs Dueling DQN

實際測試 double dqn 以及 dueling network 在 breakout 上面的表現差異，將 average reward(100MA) – 100 episode 製圖如下。可以看出，double DQN 還有 dueling network 表現不相上下，但是都明顯較一般的 DQN 來的好。

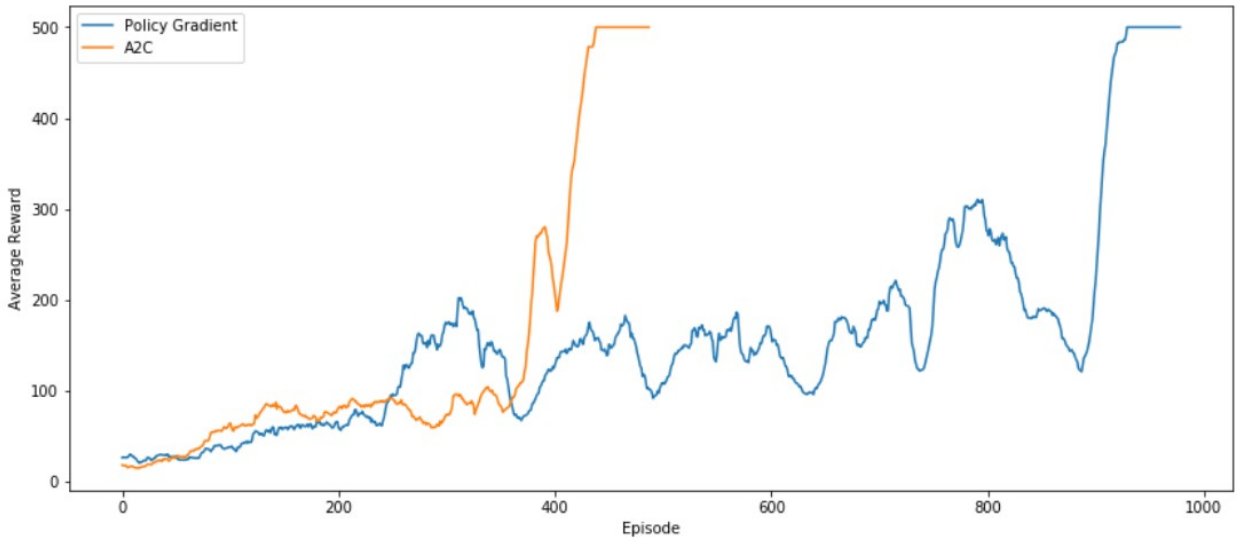


可能就如同老師上課所說的，DQN network 最大的問題就是 over estimate Q value。因為傳統的 DQN 當中 r_t + max Q(S_{t+1} , a)中，Q(S_{t+1} , a)中每一個值都有可能被高估或低估，這樣的情況下往往會不斷選擇到被高估的 Q 值，所以 Learn 出來的值就會一直被高估。利用 double DQN 的方法，可以比較公平的利用一個 Q’來決定最終要加上的 Q’。這樣被 Q 高估的值，到 Q’可能可以予以修正、而 Q’會高估的值不一定會被 Q 選到，所以大大降低了高估 Q 值的機率。

Dueling 中，透過改變最後的 output 架構，讓最後的 Q(s,a) → A(s,a) + v(s)，這樣解決了許多我一開始對於 RL 的疑慮。首先，尤其是 atari game 裡面，許多的 action 都是多餘的，但是傳統的方式中 action 是唯一間接觀察 reward 的手段，所以儘管有許多不重要的 action，也只能繼續告訴你的 model 在這些情況下你要怎麼做這些不重要的 move。但是 duel network 裡面，把整個分數切成了看 action 和 state 的 Advantage 加上只看 state 的 V，這樣在許多情況下可以注重觀察 V 的狀態，因此得到更好的表現。

Bonus 2. A2C Performance on Cart-Pole

我實做了利用 a2c 在最簡單的 atari 遊戲：cart-pole 上的表現差異比較。在環境中 reward 代表平衡桿可以撐多久，所以到 500 設為一個上線結束。將 reward-episode 繪製出來如下圖：可以看出下圖的橘線(A2C)很明顯的叫快速達到 500 分，所利用的時間(episode 數)只需要一般 policy gradient 的一半。



Advantage Actor-Critic 的方法中，我們學 policy 這件事情中，我們利用 critic 估計來的 expected reward （V），作為 objective function 的一部分

$$\text{Advantage function: } r_t^n - \left(V^\pi(s_t^n) - V^\pi(s_{t+1}^n)\right)$$

這樣這個 advantage funtion 做出來就不再只是 reward 大小，而是正或負，到表我們應該增加或減少選擇這個 action 的機率。

參考資料：

breakout:

https://github.com/rlcode/reinforcement-learning

https://keon.io/deep-q-learning

pg:

https://github.com/mrahtz/tensorflow-rl-pong