Q1. Models

POLICY GRADIENT

PG的model會把每次的state, action拿去練習，像是如下：

$$(s_1, u_1, s_2, u_2, \ldots, s_H, u_H)$$

每次會有action動作對應的reward，將他一樣記錄下來，如下：

$$(s_1, u_1, r_1, s_2, u_2, r_2, \ldots, s_H, u_H, r_H)$$

有了這些資料，收集起來再回去update model的參數，就會有新的actor，新的環境、新的actor就會有新的reward，在同樣回去做update policy。
而reward越到後面會越遞減，Gamma就是discount rate

$$\mathbf{R}_t = \sum_{t`=t}^{T-1} \gamma^{t` - t} \mathbf{r}_{t`}$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

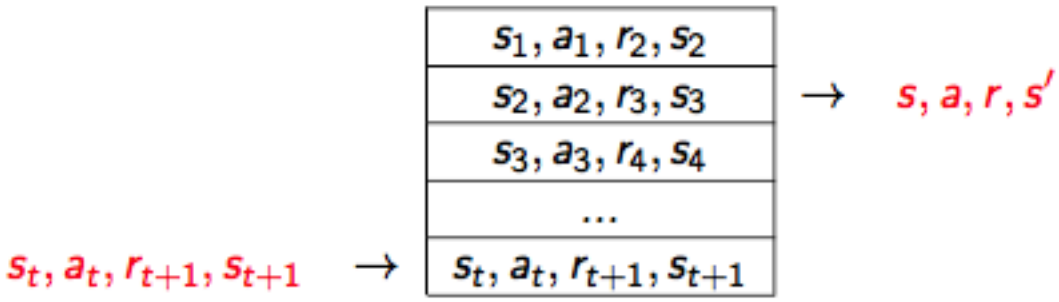
$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

一直重複下去，找到最好的policy。

REWARD的MOVING AVERAGE CURVE
可以看到前面一開始會是負的，後面慢慢到 > 0，一直到超過50

DEEP Q NETWORK

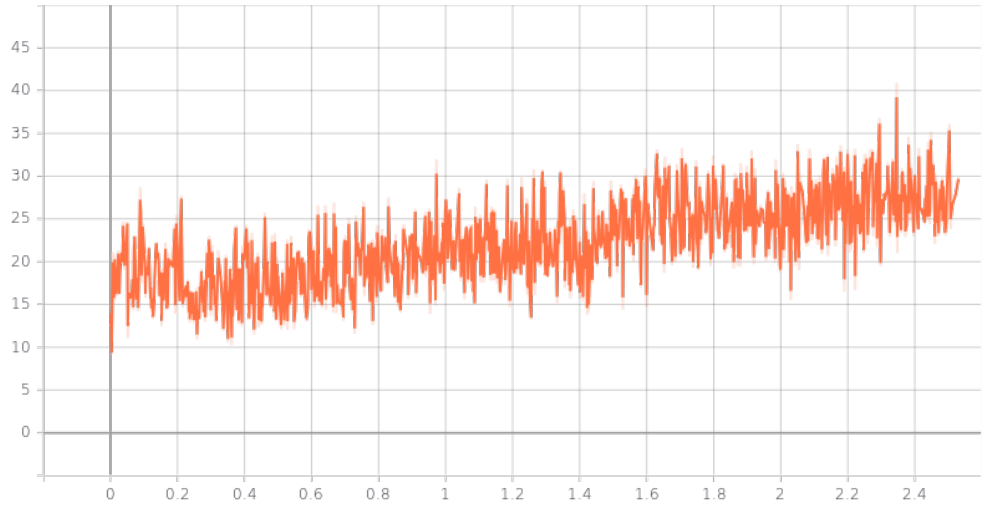
DQN一開始會先有一個 target_net & online_net來做預測及評估loss，首先依照目前的state，利用epsilon-greedy來決定是否要exploration，或是依循現在state來做action
會先把experience replay buffer填到10000個buffer後，會有 (S, A, S', R)，接著才會去做update



做update之前，先從replay buffer 中sample出一個batch，然後算出target_network的Q值，同時也算出online_network的Q值，再去算loss，我是用mse_loss來得到loss，
那數學是會是如下：

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}_{\text{target}} - Q(s, a; \theta_i)\right)^2$$

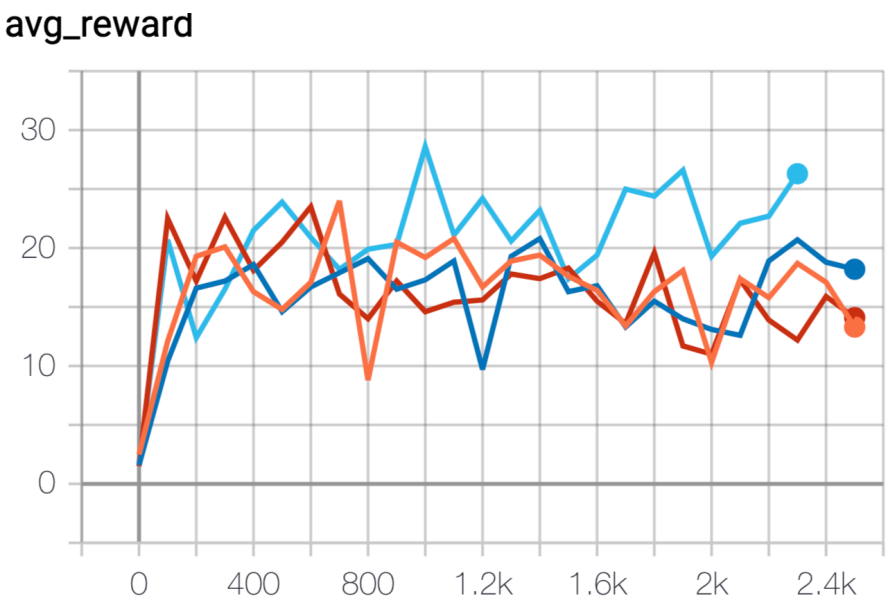
其中中間就是target算出來的Q值再去和我們算出來的Q值去做loss
接著做update，而update會每1000 steps去做target的update



最後我們可以看到REWARD會是一直上升的，大概到整個TRAIN完會落在大概 [20, 30]

Q2. Hyperparameters of DQN

選了DISCOUNT RATE當作實驗對象，也就是GAMMA
先用了原本預設的0.99，後面依序用了0.999999, 0.999, 0.95，如圖：



橘色: 0.99　深藍色: 0.999999 紅色: 0.999　　淺藍色: 0.95
大致上可以看出GAMMA越大，拿到的REWARD比較小。
比如深藍色跟紅色其實都比原本的橘色0.99還要來得差不多，有時會更差。
而小很多的0.95，淺藍色在中前期就開始比其他還高，後面都拿到25以上的REWARD

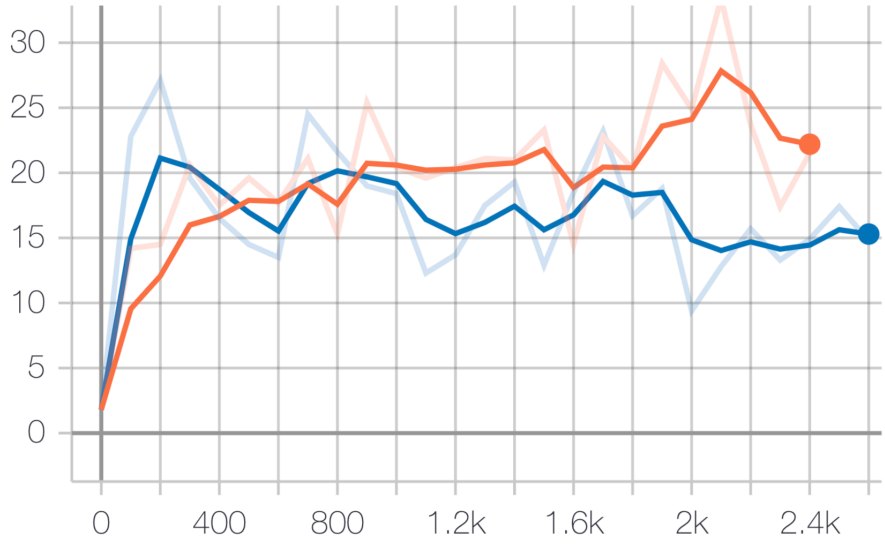
選discount rate當實驗對象是因為，我是同樣在小精靈上面來玩。想說通常玩這遊戲應該會需要考慮在做這個動作時，未來的情況可能是滿重要的，比如走進一個出口很少的角落之類的（個人猜想），所以想說discount rate比較大，後面動作不要折太多的話，會不會比較好。結果discount rate很大看起來不會差太多，反而是discount rate小，一次就折滿多的的話，會比其他來得好，跟我想得相反。

Q3. Improvements of DQN

DOUBLE-DQN

DDQN & DQN 差別在於Q值的計算
把原本的DQN改成了Double-DQN，可以用來解決DQN時常高估Q值的現象，因為DQN會直接取最大的Q值，通常會高估而有偏差。實作上來說除了Q值的計算，其他地方都相同。
DDQN會先找出最大Q值的那個Action，然後用那個動作在target_network裡面計算Q值才去做loss，而不是像DQN直接拿出最大的Q

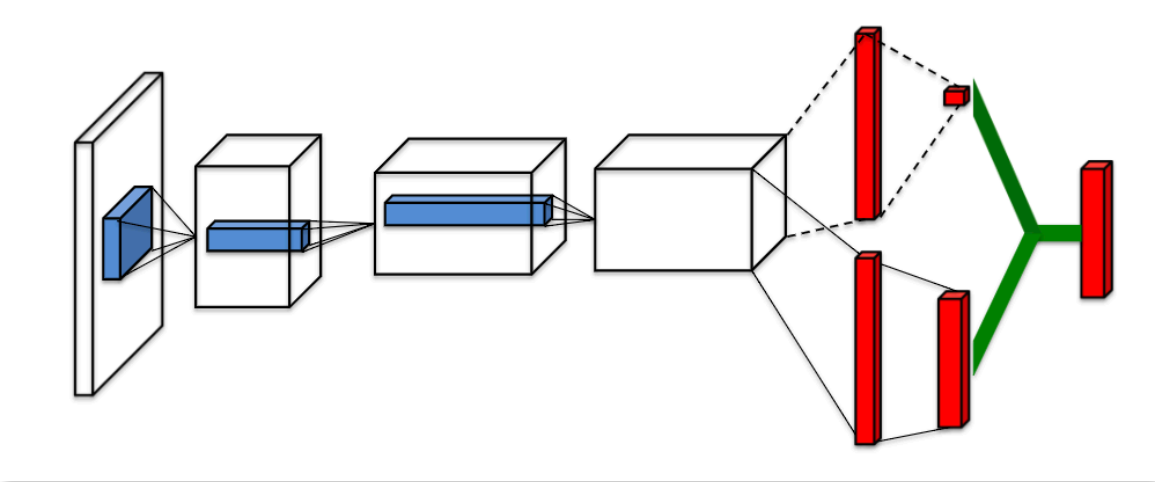
Double Deep Q-network



橘色: DDQN 藍色: DQN
結果如上，到後面DDQN總是比DQN好，DDQN之所以會表現得比DQN還好，因為解決了DQN高估Q值的，畢竟他就是直接去找MAX的Q值

DUELING-DQN

Dueling-dqn 多了一個value function來看現在state，也就是會多對state做評估

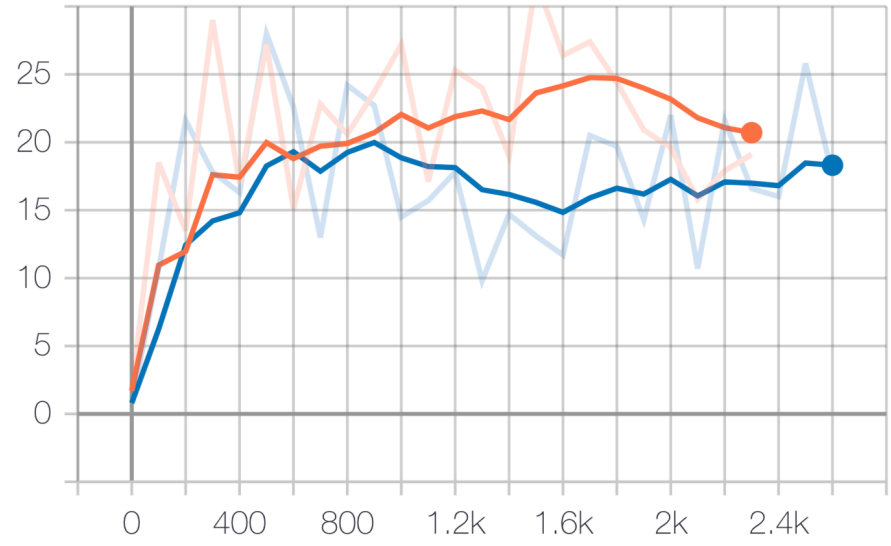


如上圖，最後的Q值從VALUE 和 ADVANTAGE而來

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha),$$

如上，會多考慮一個value，而他只會看state

Dueling-DQN



橘色：DUELING-DQN　　藍色：DQN

可以看到dueling-dqn普遍是比dqn來的好，顯示了如果多看現在state的狀況，可以提昇reward，可見多看環境或許是個值得學習的。