ADL HW3: Game Playing

R06725035 陳廷易

Basic Performance

Policy Gradient model

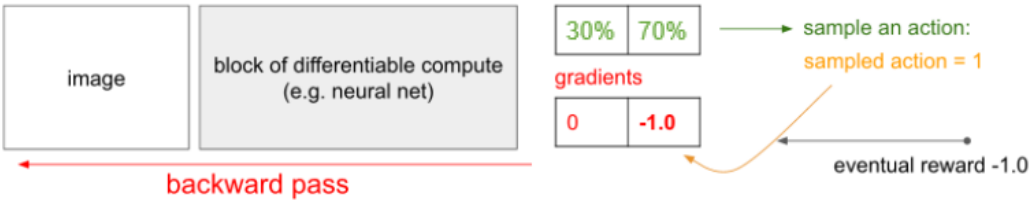
1. 影像前處理：將所得影像進行資料前處理，如移除背景、移除顏色、縮減取樣等等。
2. 利用神經網路計算往上或往下移動的機率。
3. 從機率分布中取樣決定 agent 要往上移動或是往下移動。
4. 如果一局結束後，依照是己方漏接球或是對方漏接球得知這局是贏或輸。
5. 如果有一方獲得了 21 分表示 episode 結束，則將結果利用 backpropagation 計算 weights 的 gradients。
6. 歷經十個 episodes 以後，將 gradient 加總並將 weight 朝該 gradient 方向移動。
7. 不斷重複此流程直到通過 baseline。

- Optimizer = rmsprop
- Activation function = relu
- Hidden layer neuron = 512
- Batch size = 16
- Learning rate = 1e-4
- Gamma = 0.99
- Decay rate = 0.99

首先為了使學習更穩定，依據助教所給的 document 僅挑出向上移動與向下移動兩個 action，其他重覆與無用的動作便捨棄，以加速學習。在 model 的部分，會將所餵入的畫面先通過捲積層，再利用全連通層作為 hidden layer，最後將會輸出兩動作的機率分布。依據 softmax 所給予的機

率決定要向上或向下移動，做該動作的值設為 1 其餘為 0，接下來將沒做動作的機率乘上負號，做動作的機率以 1 減掉，並依據從環境中獲得的最終 reward 乘上經處理後的機率，再將此與 learning rate 相乘並加回上次的機率分布作為下次的 Y。最後以新的畫面與新的 Y 一起放入 model，不斷重複此過程。

機率越高的 action 在面對該 frame 就越容易取樣到，然而在做該動作時尚不知此 action 是好或是壞，但沒關係待此輪結束以後會獲得+1 或-1 reward，再將此 scalar 作為該 action 的 gradient 進行 backprop。意即 stochastic policy 會鼓勵獲得好結果的 sampled action，而會不鼓勵獲得壞結果的 sampled action。



DQN model

- Environment step =
- Epsilon start = 1
- Epsilon end = 0.05
- Exploration step = 1000000
- Batch size = 32
- Experience replay size = 250000
- Learning start steps = 30000
- Target network update frequency = 10000
- Gamma = 0.99
- Optimizer = RMSprop
- Learning rate = 1e-4
- 1st Conv2d : out channels=32, kernel size=8, stride=4, activation=relu
- 2nd Conv2d : out channels=64, kernel size=4, stride=2, activation=relu
- 3rd Conv2d : out channels=64, kernel size=3, stride=1, activation=relu
- 4th fully-connected : neurons=512, activation=relu
- 5th fully-connected : neurons=(action size=4)

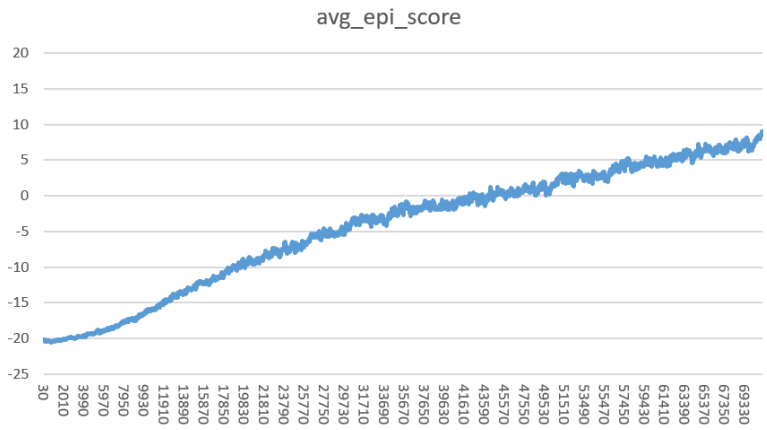
此 model 採用以下機制：

1. **Epsilon-greedy policy**：利用隨機產生 0~1 的數與 epsilon 值做比較決定是否要進行 exploration，因為 DQN 自己認為最好的 action 未必真的是最好的，因此有時需要多加常識，才可發現更好的 action 以修正判斷。
2. **Experience replay**：因在 predict 的時候會採用一個件小的參數乘上由近到遠的 reward，因此將先前遇過的 state 重新拿出來德以使其不會過度依賴於近幾次的畫面，使原本有前後關係的畫面變為監督式學習所需要的隨機獨立分布。
3. **2 networks**：target network 可以視為 policy network 的克隆，只是訓練時僅會不斷更新 policy network，經一定更新次數後才會更新 target network，兩者分開更新使訓練更加穩定。

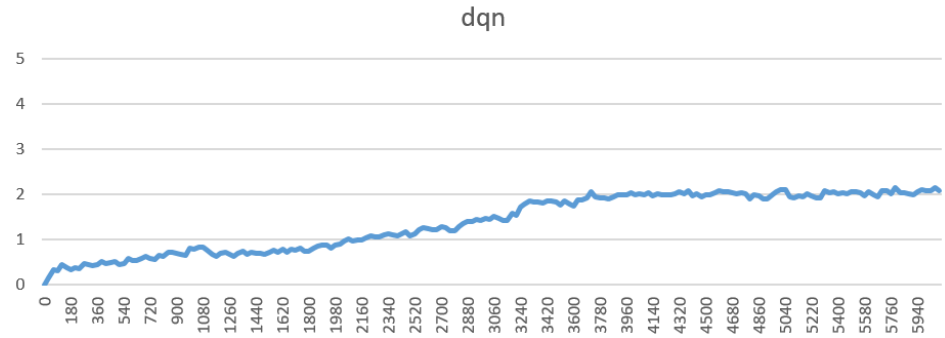
搭配助教所給予的 wrapper 環境，已將前處理部分完成，將所觀察到的圖像放入 history 當中。依據當時的 epsilon 值決定要否進行隨機動作，若否則將 history 餵入 value model 中 predict，依據最大的 Q 值決定 agent 要做的動作。環境會回傳 next state 近來，也將之 append 至 history 當中，取代 history 原先最舊的 frame 成為 next history。接下來將原先的 history, 所做的 action, 所得到的 reward, 新的 next history 存進 replay memory 當中。在訓練時，使用 minibatch 的方式每次抓 batch size 大小的資料，將連續的 training sample 相似性打破，避免對某一情況特別在行，其他表現卻過差。此外，為了使 training 更穩定，除了使用 q network 外也使用 target network，在 agent 端可以用來預測下個 action 會帶來的 reward，target network 會產生目標的 q 值用以計算 loss，平時參數是固定的；而 q network 在 training 時每一步都會發生偏移，更新一定數量後才更新 target network 的參數，利用 target network 所產生的學習目標以便 q network 在下個周期更新。

$$L(w) = \mathbb{E}[(r + \gamma max_{a'} Q(s', a', w^-) - Q(s, a, w))^2]$$

Learning curve of Policy Gradient on Pong



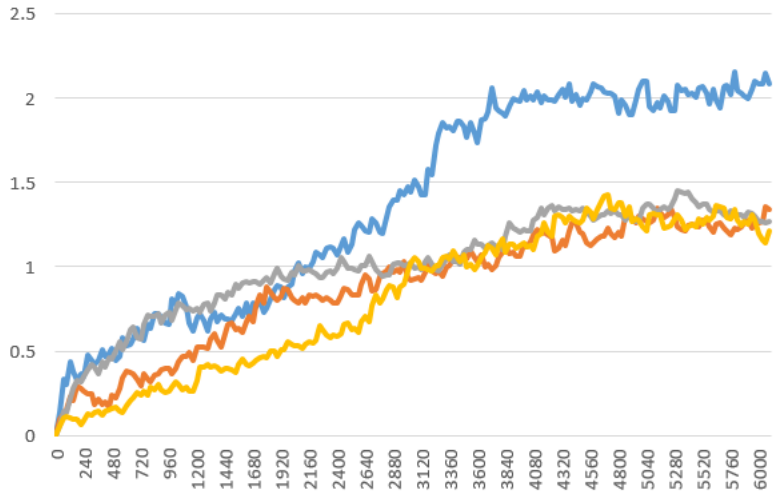
Learning curve of DQN on Breakout



Experimenting with DQN hyperparameters

因時間較趕的緣故，僅選前 6000 episodes 來作實驗。藍色線為原始的 network 架構。灰色線為僅有捲積層沒有其他 hidden layer 的結果，network 因而比較難學成。灰色線 memory 僅為原始的一半，因此可能 replay 的效果比較差。紅色線的 exploration 部署僅有原始的一半，因此可能會忽略比較多的有用訊息。

Four learning curves



Bonus

Implement other advanced RL method

1. Asynchronous Advantage Actor-Critic (A3C): (code:Pong_A3C.py)

原需要蒐集許多Γ來進行計算，但改為由 critic 的 value function 估計出來的 reward 來進行 policy gradient 的 update。也就是利用 critic 所給的估值作為該怎麼微調 policy 變化的指引，也就是 advantage function 會指引在看到此 state 時做某動作的機率要上升或下降。Actor 學 policy 而 critic 學 value function，參數亦可共享。此外，也會將 output entropy 作為 policy 的 regulation，希望能多做探索。而 asynchronous 利用 worker 機制可各自進行計算再更新 global model，達到平行訓練而大幅提升學習效率，提高收斂性。

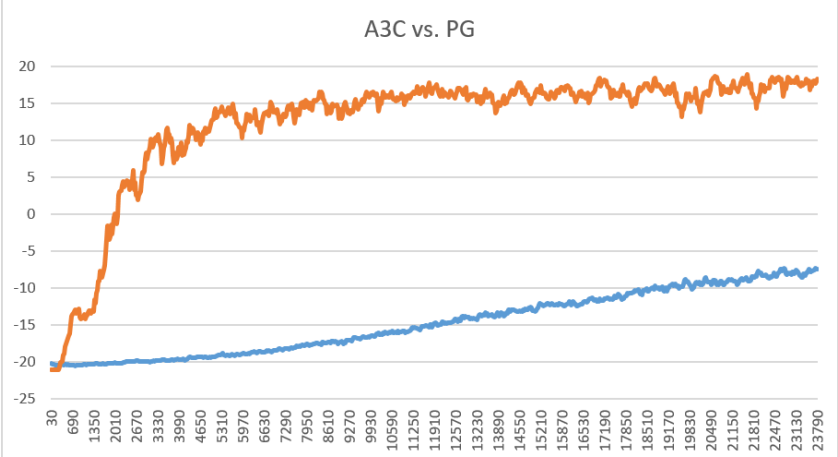
在實作部分，使用 actor network 及 critic network。actor 利用 state 與 action 決定 gradient ascent 的方向；依據 critic 所產生的資訊告訴 actor 該方向對不對。Critic 則負責學習 state value，計算 TD error 在利用此 error 評斷這一步是否有帶來比

較好的結果，也就是 advantage。

$$\theta^{\pi'} \leftarrow \theta^{\pi} + \eta \nabla \mathcal{R}(\theta^{\pi})$$

$$\nabla \mathcal{R}(\theta^{\pi}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p(a_t^n \mid s_t^n, \theta^{\pi})$$

evaluated by critic



Improvements to DQN

1. **Double Q-Learning: (code: ddqn.py)**

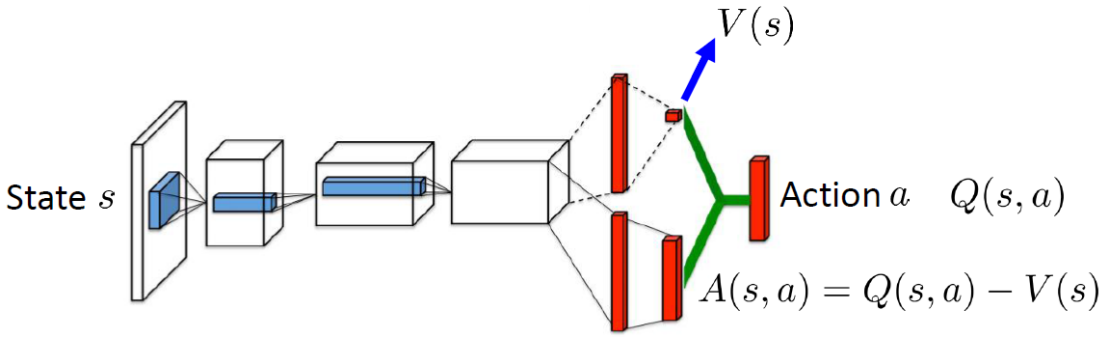
一般的 DQN 是選擇此 state 下可以使 Q 值最大的動作，然而此種做法可能會選到一些高估的 action，而其實該 estimate 出來的 value 可能與真實的 reward value 不同，但一般 DQN 會傾向選高估的動作因此較不準確。

而 Double DQN 多加一層 Q 來 model 變成兩個不同的 Q network，一個拿來選擇 action 另一個拿來估計現在的 value，兩個分開就不會是同一組參數也就較能避免高估。實作方法就是將中間的 action 置換成從新的 q 拿，而非原本舊的 current 的拿。

$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \gamma Q(s', \arg\max_{a'} Q(s', a', w), w^{-}) - Q(s, a, w)\right)^2\right]$$

2. **Dueling Network: (code: dueling.py)b**

因將原先的 Q-network 拆為兩個 channel 分開學，一個負責學這個 state 可以得到多少期望值，另一個則負責估計這個 state 可以得到多少額外的 benefit，如此便會比原先直接學 Q 還容易。


$V(s)$
State s
Action a
$Q(s,a)$
$A(s,a) = Q(s,a) - V(s)$