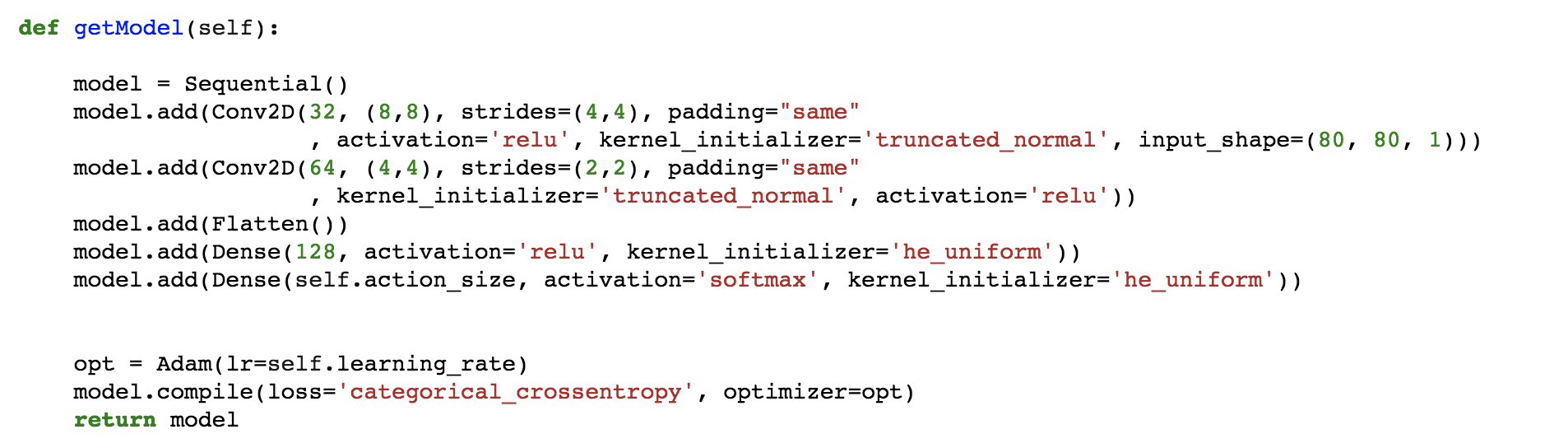
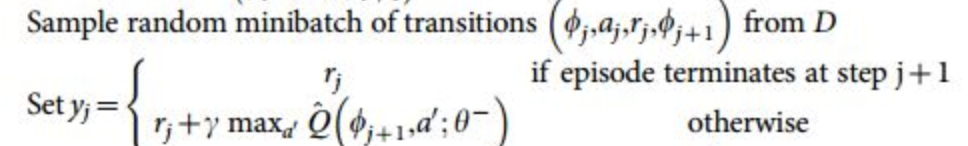
Report

R06922074 吳柏威

* Describe your Policy Gradient & DQN model
  + Policy Gradient
    - 前處理
      * 首先將每次的observation經過preprocess處理，將畫面進行灰階化之後並且將影像壓縮，從(210, 160, 3)壓縮成(80, 80, 1)，這樣子可以有效的降低model的大小，降低圖片的複雜度，有效提升training的速度與品質。
      * 接著在training或是testing時，丟入model的input都是目前的state（已經過前處理的畫面）減去上一個state，讓model只關注於那些有移動的地方，像是球以及敵我雙方的位置與軌跡，藉由擷取這些特徵，可以讓pong學的更好
    - 紀錄
      * 在訓練時我們會有4個array去記錄每場之中所有的state（目前state與上一個state相減的畫面，下同）、決定每次動作時得到的reward、每一次state輸入model之後得出每一個動作的機率分佈、以及gradient。gradient是在該state輸入model之後得出每一個動作的機率分佈prob輸出之後，並經過該機率分佈隨機選擇出的動作進行one hot並且與該機率分佈相檢所獲得的梯度。
    - 訓練
      * 接著我們將每次個動作的reward進行discount以及疊加的動作，讓一個動作當下雖然沒reward，但是根據之後的移動我們可以計算出一個動作之後到結果預期的reward是多少，而discount的動作是為了要讓遊戲盡快贏得越快越好，避免鬼打牆情況出現，接著進行normalize的動作，避免突然巨大的回饋影響了整個學習的過程。
      * 最後我們將rewards乘上先前的gradient以及learning rate，這就是我們最後所要算的loss，之後與prob相加之後成為keras model的label，而input為每一場的所有states。
    - 模型
      * Adam(lr=0.0001), gamma=0.99
      * 因為在pong遊戲中123和045動作相同，所以只輸出動作123，可以讓收斂速度大幅增加，訓練更快



* DQN
  + 前處理
    - 助教已經做好前處理了
  + 紀錄
    - 在訓練時不斷的將目前的狀態、移動所獲得的獎勵、model所predict出的移動、以及下一個state和遊戲是否結束給記錄至memory中，memory是一個deque，當塞入的東西超過了，就會將最早塞入的給移除。
    - 在遊戲的過程中，我設定經過4個state就會更新一次online model，以及每1000個state更新一次 target model，從online model複製參數過去。如果不使用fix target Q的技術，會在學習上不能好好的根據reward去更新Ｑ值，導致整個網路無法訓練。
  + 訓練
    - 在訓練時我們會根據memory隨機抽樣產生一組minibatch，我設定為32。接著依照Q learning的公式進行更新。

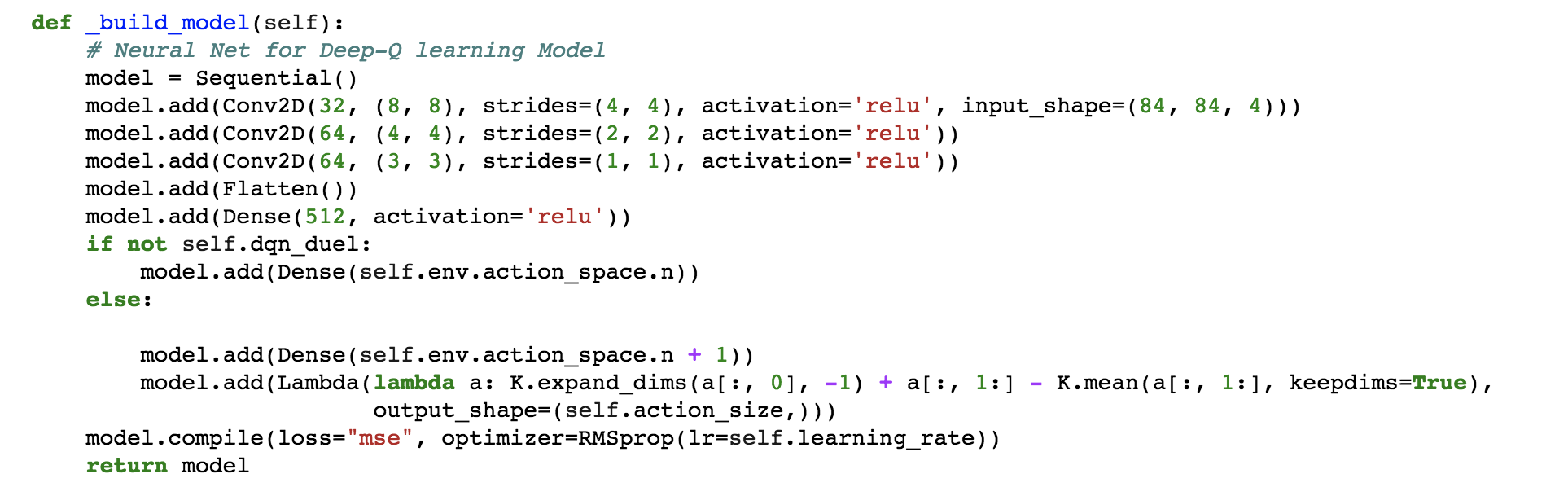


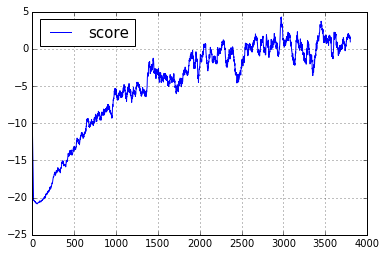
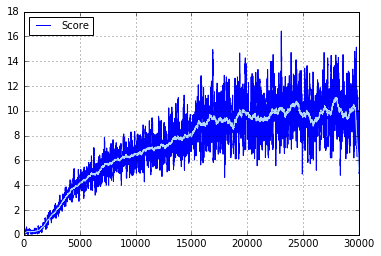
j是minibatch中的index

是reward、a’是選擇的動作、是gamma值、是state

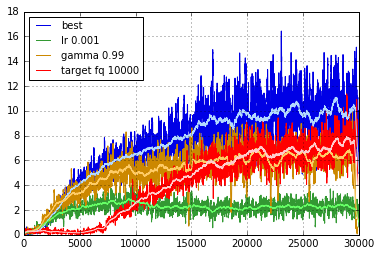
最後算完y之後成為model的label，同時input為state放入model訓練

* + 模型
    - RMSprop(lr=0.0001), gamma=0.05, epsilon\_step=100000



* Plot the learning curve to show the performance of your Policy Gradient on Pong  
  
* Plot the learning curve to show the performance of your DQN on Breakout  
  
* Plot all four learning curves in the same graph

以下是以dqn為例，best為 lr = 0.0001, gamma=0.95, update target frequency=1000，並且以此基礎進行以下更動

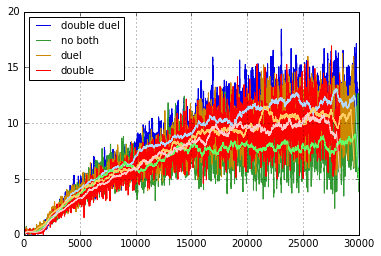


* Explain why you choose this hyperparameter and how it effect the results
  + Learning rate 0.001
    - 在reinforcement learning中，learning rate的更動往往佔有非常重要得角色，所以我選擇更動此參數
    - 過大的learning rate會使得模型更動幅度太大，無法訓練
    - 適當的lr才能讓model有緩慢的進步
  + Target Q update frequency 10000
    - 挑選一個好的Target Q update frequency也很重要，影響了訓練的速度與品質，所以我選此參數
    - 我發現Target Q update frequency調大之後，在約6000左右的episode才score才開始有進步，而且進步的幅度比調成1000還慢
  + gamma 0.99
    - 好的gamma可以使得模型具有遠見，不會短視近利。同時不容易讓遊戲進入鬼打牆情況發生，所以我選擇此參數
    - 當gamma調成0.99時，我發現訓練後期分數並不如gamma為0.95的模型，我認為是因為小的gamma可以讓模型能夠在更短的時間內得到更多分數，模型可以訓練得更快更好。
* Implement at least two improvements to DQN (p.9) and describe why they can improve the performance
  + Implement內容寫在agent\_dir/agent\_dqn中
  + 可以利用\_\_init\_\_ function改動self.dqn\_double以及self.dqn\_duel來選擇是否使用
  + Double DQN
    - 我們將Q learning的公式從以上的公式改為



也就是我們先把下一個state丟入online Q中求出分數最好的動作，接著我們在把該動作丟入target Q中求出Q值

* 因為我們的神經網絡預測Qmax本來就有誤差，每次也向著最大誤差的改進Q现实神經網絡，就是因為這個Qmax導致了overestimate。所以Double DQN的想法就是引入另一個神經網絡來打消一些最大誤差的影響。
* Duel DQN
  + 我們將原本的model改為以下架構  
    
  + 我們將最後面一層分成兩個部分，一個是value function以及advantage function，value function是接收state之後預期的Q值，而advantage function是輸出每一種動作可以創造出的價值，分開來之後接著在串接起來並且最後dense成每個動作的Q值。
  + 利用Duel DQN可以大幅提升學習效果以及收斂速度。
* Plot a graph to compare and analyze the results with and without the improvements



當我們double還有duel都沒有使用時效果最差

使用double dqn可以避免overestimate情形發生，效果比都沒有好一點

使用duel dqn可以有效的收斂Q value，效果又比加了double更好

最後兩個都加效果最好同時也可以通過作業的baseline

* Implement other advanced RL method, describe what it is and why it is better
  + - * 我implement的是A2C, Code在agent\_dir/agent\_pg\_a2c.py中
      * 在訓練時我們使用了兩個model，actor以及critic，critic負責學習再輸入state之後，他需要輸出的是這個state預期的reward，而actor則是在輸入state之後輸出每個動作的機率分佈，在訓練actor時，我們會使用advantage function去update actor的參數  
        在update critic的時候，我們是使用l2 loss的方式計算predict value以及reward的差距，而actor則是利用advantage去update，當reward大於value時，則該動作的機率會被提升，反之則會降低
      * 他結合了dqn(critic)以及pg(actor)兩種優點，不僅提升了學習效率，同時也可以訓練出在連續動作中可以選取合適的動作。
      * 圖表：

可以看出訓練曲線提升得更快，訓練的效果也比pg好很多

