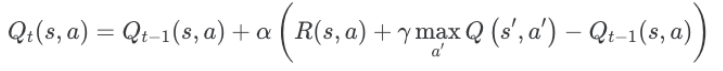
**Basic:**

1. **實作**

根據Q-Learning的方法，在每次的iteraion中，去更新我的q\_table。至於更新q\_table的關鍵是要知道如何選擇action，並根據得到的reward去更新q\_table。要完成q\_table的更新，需要完成以下步驟:

1. 透過choose\_action()這個function，根據當下的state，去選擇擁有最多q-value的action。再將action傳入env.step()這個agent，以得到observation及reward。
2. 再將得到的observation，丟入get\_state()，得到next\_state。

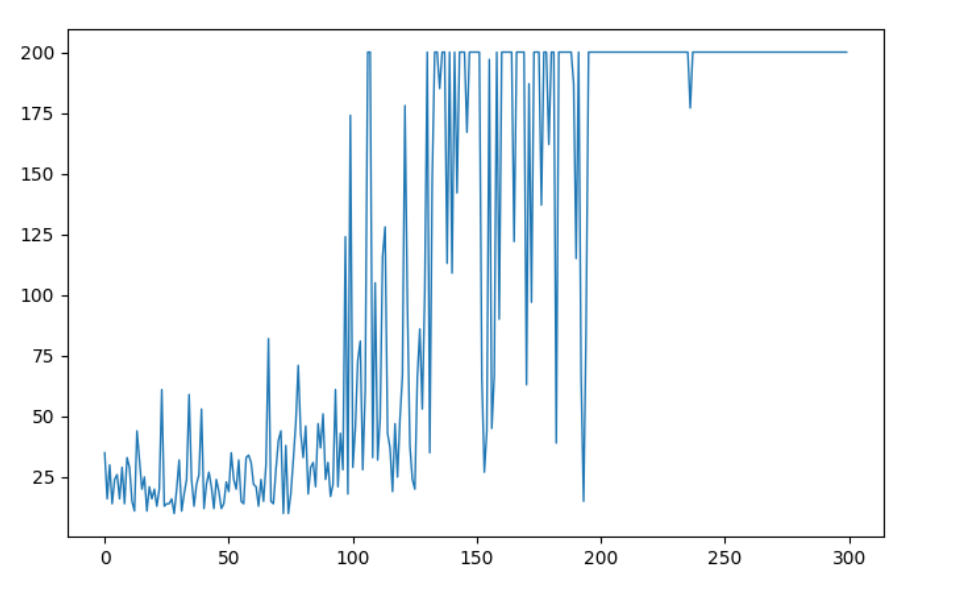
完成以上步驟，就可以用以下公式，去更新q\_table的值了。



另外一個重點是n\_bucket，因為state的四個元素分別是[position of cart, velocity of cart, angle of pole, rotation rate of pole]，然而這些直是連續的，所以就要透過n\_bucket把這些值變成discrete。

至於參數的設定，設定N\_EPISODES = 300, EPISODE\_LENGTH = 250，n\_buckets = (1, 1, 12, 10)，比較特別的是learning rate，template已經設定learning rate會隨著時間的增加而減少，最一開始試0.5，到最後會變成0.01。

以下是我的訓練過程，可以看到到後面，reward已經可以穩定維持在200了。



**Advance:**

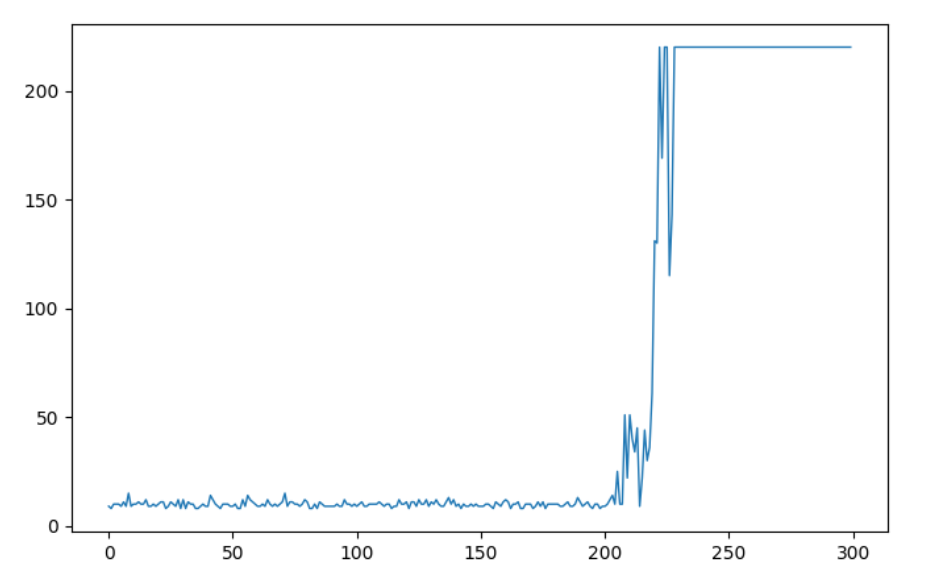
1. **實作**

q\_table 的缺點是如果 state 和 action 很多，table 也會需要很多空間。用 neural network 學習預測 Q-value 的話，就可以減少很多空間。

跟basic比較不一樣的是，我們在這邊是要訓練model去作出最好的選擇，所以Q-value的值就會變成是Y，也就是groundTruth，再訓練的過程中，會以Q-value為標準，希望model預測出來的數字可以越接近。至於model的Input 會是state(可以是連續值)，output 是這個 action 未來會帶來的總 reward。因此我們就可以透過reward，知道要選哪個action。

這個model的作法是先將前200比的結果[state, [action, reward], next\_state]當作是train data，直到收集完train data後，才開始learn。

以下是我的訓練過程，可以看到到後面，reward已經可以穩定維持在200以上了。



1. **遇到的困難:**
   1. 因為之前沒什麼接觸pytorch的經驗，跟tensorflow的模式比較不一樣，但我也透過這次的做也知道如何build一個pytorch的model。
   2. 再實作的過程中，我有發現template有一個Cheating part部分，會對model predict出一個結果進行一些處理。但照理來說，應該用model預測出來的結果可以了。所以我有嘗試過把那段code拿掉。結果訓練結果變得非常差，所以Cheating part應該是可以增加訓練結果的手段。