人工智慧導論 Final Project by 葉乃瑄 B09902008

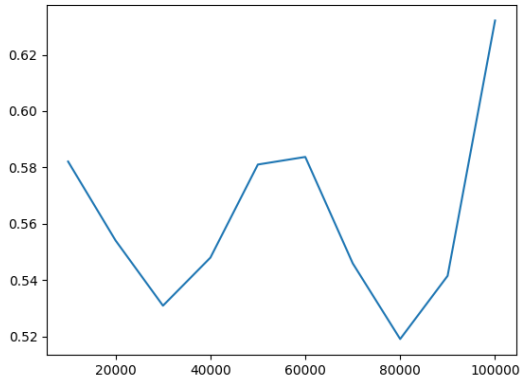
1 Card Encoding

在 AI 做決策的時候，我希望 AI 可以盡可能地擷取場上的資訊來做決定，而在眾多資訊當中，我認為手上的牌是最重要的，為了與之後的深度學習演算法銜接，我勢必得在一開始就把卡片轉成模型可以讀懂且有用的形式，因此我試了以下幾種方法

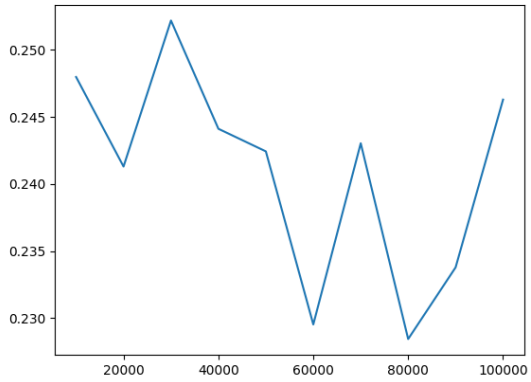
1.1 蒙地卡羅演算法

1.1.1 原理

為了對手上的牌做強度上的評估，我首先試了蒙地卡羅演算法，也就是隨機生成對手手上兩張牌跟場上還沒開過的牌，如果我的牌型大小大於對手的，則算我贏，統計隨機生成 1 萬次的結果後就可以對勝率有個精準的評估



不同次數蒙地卡羅預測結果的平均值



不同次數蒙地卡羅預測結果的標準差

1.1.2 優缺點

這樣的做法可以在省去探索所有對手跟場上可能的牌型 (最糟糕會有 P(50, 7) 種可能)，並且拿到還算準確的結果，但當缺點相當明顯，雖然我已經省去了相當多的時間，但跑了超過 1 萬次以上的蒙地卡羅演算法仍會非常耗時，且經過統計 (如上圖)，我發現蒙地卡羅非常不穩定，除了相同牌型不同次數預測的結果會差很多之外，同樣是 1 萬次預測，每次跑的結果都差很多，之後如果要將蒙地卡羅預測出來的數字當作 input 輸入模型，會造成很大的雜音，且太多次預測也會讓 training 非常耗時，因此我開始想用一個深度模型，其 input 跟 output 蒙地卡羅一樣是一副牌跟勝率，利用蒙地卡羅 1 萬次的結果當作目標來更新我的模型

1.2 CNN

1.2.1 Dataset

我把牌轉成 4x13 的 two-hot 陣列再將其向四周 padding 到 17x17，並且分成我手上的牌、場上的牌跟這兩者加起來共三種後 concat 在一起，變成 3x17x17 的陣列當作 CNN 的 input，接下來，生成 10 萬種數量 (2, 5, 6, 7 張) 跟牌都是隨機的牌型，將他們跑 1 萬次蒙地卡羅後的結果跟對應到的牌型記下來，這樣就有一個適合拿來 train 的 dataset(這個 script 跑了超過一天)

1.2.2 架構流程

我先用了三層卷基層將 input 從 3x17x17 打到 128x1x1，再丟一次 squeeze 讓維度變成 128，之後接三層 Linear 層讓維度降成 1，用 nn.Sigmoid 把值域限制在 0~1 之間，就可以代表模型預測出來的勝率，最後再跟蒙地卡羅 10000 次後的結果取 L1 Loss 來更新我的模型，Settings: optimizer = Adam，learning rate = 1e-4, batch size = 128

1.2.3 優缺點

模型預估出來的結果和蒙地卡羅平均只差了 0.02 左右，對比不同次數的蒙地卡羅之間預測的差距可以說是非常小的，而且此模型很快就收斂了，總計才 train 了 3 個小時左右，最令人驚訝的是，這個模型的預測時間大概跟 1000 次蒙地卡羅一樣快，但得到的卻是跟 10000 次蒙地卡羅一樣的結果，且相比蒙地卡羅，當我對一樣的牌型預測多次，模型的結果永遠都一樣，而蒙地卡羅卻是每次都不一樣，這樣的穩定性有助於接下來的 RL training，但缺點就是生成 dataset 非常花時間

2 Reinforcement Learning

做完了 Card Encoding, 我便開始著手寫 Reinforcement Learning 相關的演算法, 我最終的目的是建一個深度模型能藉由觀察遊戲中不同的狀態來學習, 到最後可以在不同的遊戲進程中做出最佳的決定，也就是讓 AI 自己決定每一回合應該要做什麼動作，而不是使用 rule-based 的方法由我來規定 AI 在哪些條件下該做哪些動作。
以下是我試不同 Reinforcement Learning 演算法的方法和結果

2.1 Double DQN

2.1.1 原理

DQN 是由 Q-learning 延伸而來的，因為課堂上有對 Q-learning 的講解，這裡就不再過多贅述，DQN 將原本 Q-learning 公式裡的 Q-function 從建立 Q-table 來得知不同 action 跟 state 下的 Q 值轉換成藉由模型來得知不同 state 下各個 action 的 Q 值，而

Double DQN 則是改進了原本算 Q-function 的方式，藉由兩個模型來生成 $Q(s_t, a_t)$(eval net 負責) 跟 $Q(s_{t+1}, a')$(target net 負責)，但只有 eval net 會隨時更新，經過一定的時間後再把 target net 跟 eval net 做同步

$$Q(s_t, a_t) = r_t + \gamma \max_{a'} Q(s_{t+1}, a')$$

這樣的做法可以讓模型的穩定度提高，原本 DQN 兩次的 Q-function 都是由同一個模型去計算，因為模型隨時都會變化，而 TD-Error 又是由模型自己算出來的，會有高估 Q 值的問題，所以像這樣一邊隨時更新、一邊固定的方式去計算 Loss 的話，會讓出來的 Loss 更加穩定

2.1.2 架構

在德州撲克中雖然看似一回合只有 3 種不同 action(fold, call, raise) 可以做，但根據 raise 的金額事實上會有無限多種 action，所以我先將 action 用離散的方式表示成五種，棄牌、跟牌、加注最小值、加注最大值的一半跟 all-in，這樣模型只需要輸出這其中一個動作就可以了，input 的部分有分兩種，一種是上面提到的把卡牌變成 3x17x17, 另外一種是 Card encoding 後的勝率與場上可以看到的資訊 (總計 16 個)，卡牌的部分會經過 3 次卷基層變成 128x1x1，剩下的會經過兩層 Linear 層變成 128，將這兩者接起來變成 256 後丟入 3 層 Linear 層後變成 5，就可以代表對 5 種 action 預測的 Q 值, training 的方式是跑 3000 場左右的遊戲, 每次從 baseline0, 1, 2, 3 隨機挑一個人當對手, 並記錄最近 100 場的勝率來觀察模型是不是收斂了, 過程中每次要 declare_action 的時候就把當前的 state 跟上一個 state 和 action 配對起來存到 memory 裡面，當 memory 滿了之後，每次做動作就開始從裡面隨機拿出 batch size 個狀態計算 TD-Error，拿這個 Loss 去更新我的模型，而途中的 reward 則是由輸贏的錢除大盲來決定的，除了每一個 round 最後一個狀態之外其他都是 0，Settings: optimizer = Adam，learning rate = 1e-4, batch size = 128, memory capacity = 5000

2.2 Dueling Double DQN

2.2.1 原理

dueling 的概念是從分析 Q-function 而來的，主要是將原本的 Q-function 分成 Advantage(A) 跟以前一樣計算各個動作的 Q 值跟 Value(V) 用來評估現在 state 的重要程度

$$Q(s, a) = A(s, a) + V(s)$$

這樣的做法會使得模型更快收斂，因為大部分的 state 其實都是不重要的，經過 Value 的計算後就可以讓不重要的 state 對模型的影響比較小，Settings: optimizer = Adam, learning rate = 1e-4, batch size = 128, memory capacity = 5000

2.2.2 架構

這邊的架構只需要保留原本 DDQN 的 code 當作 Advantage，再另外多建兩層的 linear，讓原本從 256 打到 128 的 input 從 128 再打到 64 後打到 1，就可以當作 Value 的計算，之後再將 Advantage 跟 Value 出來的結果相加就會是真正的Q值了

2.3 Prioritized Replay Dueling Double DQN

2.3.1 原理

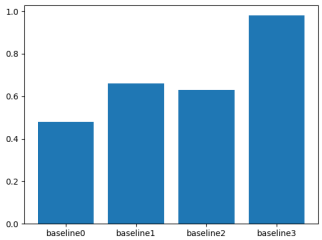
由於原本從 memory 隨機取 batch size 個當做 training set，但這樣會沒有考慮到事實上每個 state 是有差距的，我不應該是平均地拿 data 出來，而是應該以當時算出來的 Loss 來做評估，Loss 越大的情況就代表我越需要去看他，因為比較大的 Loss 更新模型會更有效率，Settings: optimizer = Adam，learning rate = 1e-4, batch size = 128, memory capacity = 5000

2.3.2 架構

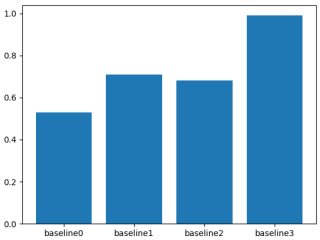
Prioritized 的部分我使用了 Sum Tree 來記錄每個存檔的優先度，也就是 TD-Error，如果被分到的優先度越高，代表有越大的機率被選到，而每當新的存檔要進來時因為還沒有計算過 TD-Error，所以會被分配到最大值，讓之後模型可以有很高的機率去計算他的 TD-Error 後再重新分配被選到的機率

2.4 結果

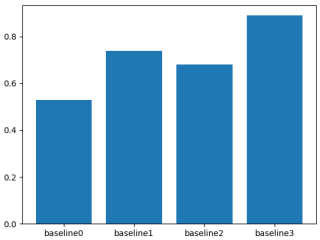
我統計了以上三種方法，可以看到一步步改良後的 DQN 勝率是有在提升的，不過還是維持在 7 成、7 成、9 成上下



Double DQN



Dueling Double DQN



Prioritized Replay

3 Conclusion

我最後使用的是 train 了 12 個小時的 Prioritized Replay Dueling Double DQN，雖然很想嘗試建完 Rainbow DQN，但礙於時間的關係，只能使用這三種方法改進 DQN