**B09902135 賴豐彰**

# Utilities

因為德州撲克的規則牌型比較較為複雜，還有一些需要估價的地方，因此，我先是寫出一個parse 牌型的util。

### 估價牌型

因為每個牌型有不同的強度，因此我們可以針對現在的手牌加上台面上翻開的牌去做評分，然而因為有些牌型（例如：同花、高牌）都會連續比較五張牌的大小導致有點麻煩。但比到總共五張牌的機率其實非常少，因此其實經過評估之後只紀錄最大的兩張牌。將其最大的兩張牌與牌型所代表的數字加起來就可以（算是有效率的）比較牌型的大小了。

### 估價手牌

因為一開始上網路上搜尋德州撲克的技巧時有發現其實起手的手牌也蠻重要的，因此利用網路上所查的手牌分數去做起手（pre-flop street）的決定，但套用在first approaches (Monte Carlo)的時候發現並沒有很好的效果，所以沒有派上用場

# Zeroth Approaches: Smart Strategy

在隨機的對戰裡面有發現baseline有時候的戰術其實很精妙，因為我們知道我們的小盲大盲分別的數目，所以我們只要一到達特定的錢（例如 $> 1150$ ）那我們就可以棄牌到底且贏得遊戲。利用這樣的戰術，可以作到所有baseline勝率達到 70% 但因為覺得這不是一個好方法，所以變把他拿來作為其中一種訓練的AI。

# First Approaches: Monte Carlo

因為可以寫出估價的函數，因此第一個其實還算簡單的方式便是利用monde carlo法，我們只要利用手中的牌，然後隨機 sample 對手的牌以及community cards，就可以算出大致的Win rate (因為沒辦法全部跑完)，如此一來，sample 10000 次，就可以依照這個Win Rate 去做決定。 一開始，我的方法是如果 $Winrate > 50\%$ 就可以call，其餘則棄牌，然後就發現其實大多數時間（ 90% ）都是棄牌，等於白白送錢。

後來就有想到一個策略是，如果沒有人raise 且 Win Rate 很低的時候，原本會棄牌，但此時因為不用在多錢，所以可以繼續看牌（不虧） 經過改進之後，贏過medium baseline勝率大約為 50% ，不過會發現有些時候會輸特別多，我把其紀錄下來，會發現輸的部份大多都是baseline AI 在拿到好牌的時候下注更多，讓我們就算拿到一個不錯的牌（Win Rate較高），也會因為下太多而導致後面追不上，因此這裡設定了一個threshold 來防止自己拿到好牌之後開始被心理戰。

雖然調完之後可以過medium baseline，strong baseline 也可以有約 50% 的勝率，不過這個AI有很大一部份是「工人智慧」，基本上這樣也已經快到上限了（後面的微調也都幾乎是為了strong baseline而去做一些不太有實質意義的調整），所以我改往下一步，也就是Reinforce Learning 的部份下去做。

# Second Approaches: DQN

發現蒙地卡羅有其上限之後，開始上網搜尋其他deep learning poker的策略，就先找到[2]，然後就會發現其實可以利用一些在 *round_state* 可以拿到的資訊變成一個tensor，但若是關於Street 的階段我們沒辦法量化，於是這裡使用one hot encoding，在手牌的部份，因為不確定怎麼做，於是參考[2]的方式，使用multi-hot encoding，因為這樣可以保有花色之間有距離對應關係，在點數上也有，而不會是單純的一個 $1 \sim 52$ 的比較沒有相對關係的數字。利用所有數值都可以做成一個Tensor之後我們接下來就可以說明我的model的架構

### Model 架構

首先我取出16個feature，包含遊戲的各個資訊，然後這個16的tensor 經過幾層Linear + ReLU 變成128長的Tensor 另外card 用手上的2張牌加上community cards，以及全部加起來的multi-hot encoding，加上一點pad變成4*16*16的tensor，然後最後塞幾層Convolution 變成 128*1*1 squeeze 變成128 與剛剛的attribute們concat 起來變成長度 256，再經過幾層Linear 層，打到一個長度為6的機率分佈，選出最高的變成我們的選擇（我先預設有幾個動作，fold, call, 幾種raise）

### DDQN 架構

我們創造兩個target model 跟 eval model 兩個， 先玩很多場遊戲，然後玩到數量足夠多之後，就從這些數據之中取出一個batch然後丟進兩個model裡面，利用兩個的差距算loss， 等到學夠一段時間之後就把target 的model 換成已經train一下子的，這樣繼續下去。 此外，為了不要過於專注於跟一個model 打（會有針對的策略的問題），所以這裡在train的時候會每一個回合random 選一個AI來打，其中也包含Approach 1中的蒙地卡羅，（但後來發現蒙地卡羅沒有實際幫助，所以捨棄，詳細見Comparison）

### Other Details

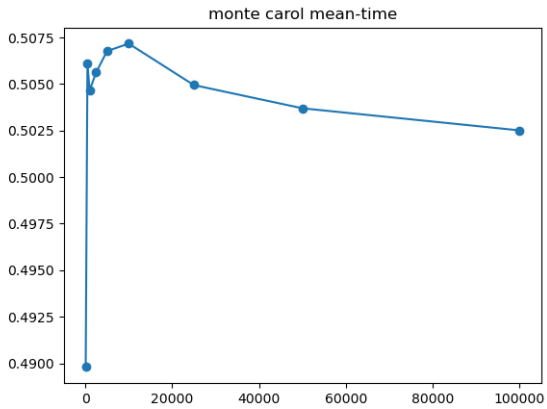
除了上面說的架構以外，我還做了一個會step的random，也就是最一開始會是全部random，最終會走到部份（ 15% )random，因為要確保model可以學到東西，所以要多一些隨機性，才可以多學幾個方面，這也是ADA裡的作業P Game有學到的，如果 100% 都是靠model的話反而會造成其實很好打贏，因此需要增加一定的隨機性。

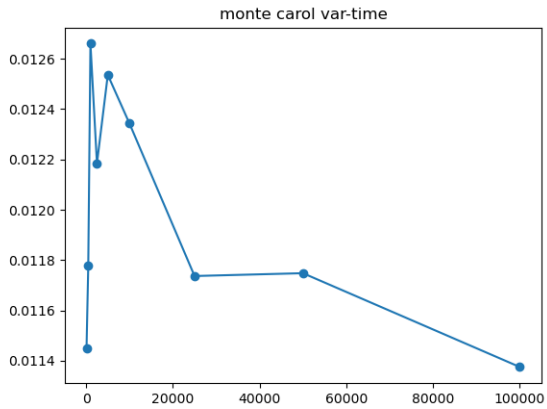
也在DDQN 上面延伸priority DQN，也就是讓memory依照l1loss分配選到的機率，讓差更多（loss更高）的人更容易丟到model裡去fit，讓對 baseline 0,1,2,3 的勝率從約50%提高至 48%, 72%, 64%, 92% 值得一提的是，當我train完之後加上Approach 0 的判斷式之後，勝率增加到 54%, 84%, 72%, 96% (但train之前不加判斷式的原因是因為想讓model explore 更多的 path)

## Comparison

### 蒙地卡羅測試

因為有一個attribute 其實是拿蒙地卡羅測試一個手牌跟community cards的勝率，然後因為這個東西如果離線打表的話會需要 $C_9^{52}$ 大小的空間，所以我們如果要做的更好勢必得train一個model，那要評估我們model的dataset 是否是正確的，就必須就要測試蒙地卡羅的準確性才能造出一個好的dataset，我使用不同次數的蒙地卡羅跑100次（因為發現10次的浮動有點大，所以跑100次做比較），做出一個評估他win rate要跑多少次蒙地卡羅才會足夠穩定，也就是和更遠的蒙地卡羅相較不遠（這裡假設蒙地卡羅越多次越精準）利用mean/variance來看是否是正常的 x軸為跑的次數, y軸為預估出的win rate的mean/variance

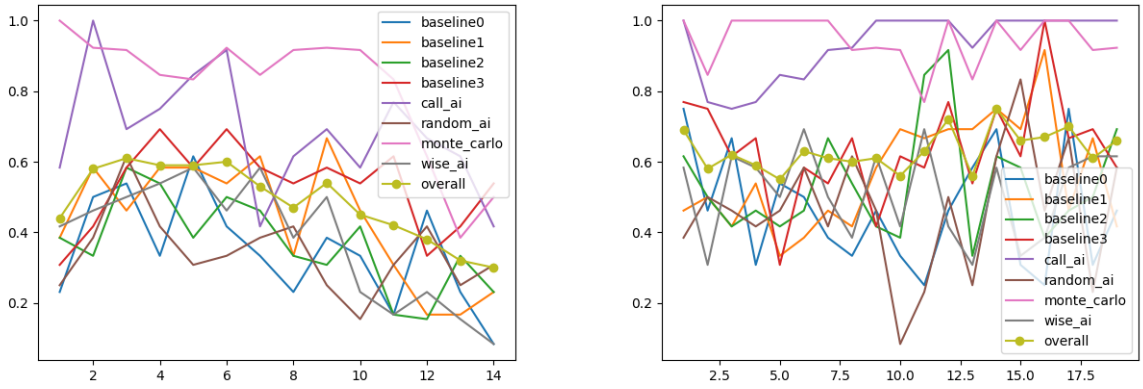


因為我們可以假設蒙地卡羅在跑越多次的時候越精準，看左圖可以發現蒙地卡羅只要在約10000次就可以精準到與100000次差 0.005 也就是 0.5% ，於是繼續做更多次下去的需求不高。然後看右圖，可以看見其實標準差並沒有變低，我推測這是說明蒙地卡羅的上限就到 $Var(winrate) \sim 0.01$ ，雖然很高，但看起來沒有辦法解決，嘗試想要train 一個給定牌就可以給出勝率的model，但是l1loss一直停留在Winrate 10% 所以就放棄了。

### DQN reward Test

因為這個遊戲針對於每個action 是sparse 的，然而DQN在這裡有兩種方式，一種是可以是sparse的reward （沒有的就是0），一種則是全部action的reward都會是最終獲得的reward，那我把兩者在train的winrate都作圖變成以下兩張圖




其中左圖為sparse的reward，右圖為全部action的reward都會是最終獲得的reward，可以發現，右圖的表現相對穩定，所以選擇右圖的策略，又可以發現，蒙地卡羅只要被learn過其實winrate會變得十分低，因此不適合被放在train的對方model裡

## Result

如上文所提到的，利用Priority Double DQN 去做前期隨機後面學習的Model，最後也決定選用這個作為繳交的agent，原因除了表現較穩定以外，也不該只單純看打baseline的勝率，更多的是因為其他agent（Approach 0 & 1）很有可能因為簡單的策略被摸透而變得不那麼厲害，在Approach 1 & 2的對打上發現DQN的邏輯比較不好摸透，因此最後選擇其作為繳交的agent 詳細的code 也放上GitHub 了不過因為不確定有沒有人會延後繳交所以可能過一下才會make public

## Additional notes

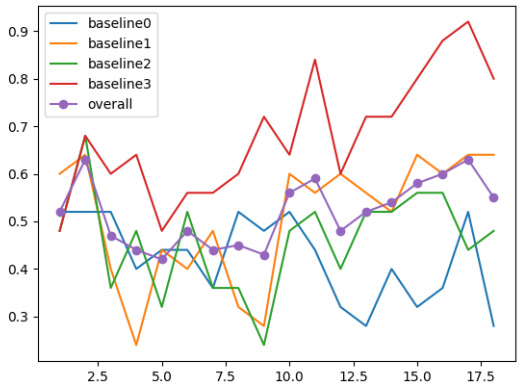
### Strange Behaviour

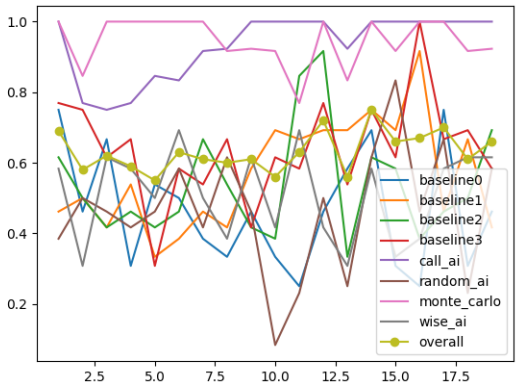
在途中有發現一些有趣的東西， 例如當我們想要raise 高一點，但是baseline 卻會做出一些invalid 的操作， 因此送錢給我們，推測是在在當下雙方都有不錯的牌型，因而都想要raise，然而baseline卻有一個raise的上限，因而造成Invalid的操作，被判fold。

```
"me" declared "raise:10"
"baseline" declared "raise:247.5"
"me" declared "raise:485.0"
"baseline" declared "raise:247.5"
"['me']" won the round 1 (stack = {'baseline': 742.5, 'me': 1257.0})
```

### Strange Win Rate

在train model 的時候，會發現一些很奇怪的事。




可以發現每當baseline1, baseline2, baseline3 勝率上升時，baseline 0的勝率會大降，包含但不限於baseline 0 對上其他三個，任何一個變高的時候另外三個勝率都會掉下去。

## References

[1] https://www.moshike.com/a/1951.html

[2] https://github.com/EvgenyKashin/TensorPoker

[3] https://blog.csdn.net/weixin_40759186/article/details/87524192

[4] https://en.wikipedia.org/wiki/Q-learning

[5] https://pytorch.org/

[6] https://zhuanlan.zhihu.com/p/142973528

[7] https://stats.stackexchange.com/questions/350451

[8] https://ithelp.ithome.com.tw/articles/10251599

[9] https://zhuanlan.zhihu.com/p/37685044

[10] https://hackmd.io/@shaoeChen/Bywb8YLKS/