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

```
def getModel(self):

    model = Sequential()
    model.add(Conv2D(32, (8,8), strides=(4,4), padding="same"
                     , activation='relu', kernel_initializer='truncated_normal', input_shape=(80, 80, 1)))
    model.add(Conv2D(64, (4,4), strides=(2,2), padding="same"
                     , kernel_initializer='truncated_normal', activation='relu'))
    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(self.action_size, activation='softmax', kernel_initializer='he_uniform'))


    opt = Adam(lr=self.learning_rate)
    model.compile(loss='categorical_crossentropy', optimizer=opt)
    return model
```

■ 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。

$$\text{Sample random minibatch of transitions } \left(\phi_j, a_j, r_j, \phi_{j+1}\right) \text{ from } D$$

$$\text{Set } y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$$

j 是 minibatch 中的 index

r_j是 reward、a'是選擇的動作、γ是 gamma 值、Ø是 state

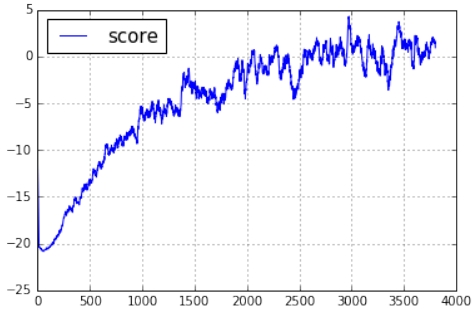
最後算完 y 之後成為 model 的 label，同時 input 為 state 放入 model 訓練

- 模型
  - RMSprop(lr=0.0001), gamma=0.05, epsilon_step=100000

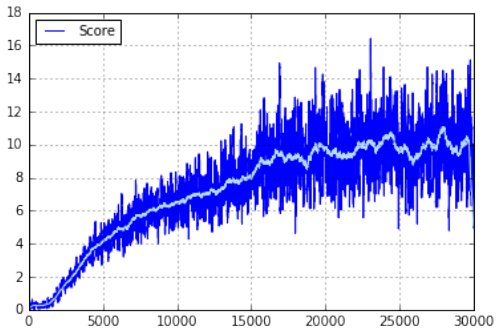
```
def _build_model(self):
    # Neural Net for Deep-Q learning Model
    model = Sequential()
    model.add(Conv2D(32, (8, 8), strides=(4, 4), activation='relu', input_shape=(84, 84, 4)))
    model.add(Conv2D(64, (4, 4), strides=(2, 2), activation='relu'))
    model.add(Conv2D(64, (3, 3), strides=(1, 1), activation='relu'))
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    if not self.dqn_duel:
        model.add(Dense(self.env.action_space.n))
    else:

        model.add(Dense(self.env.action_space.n + 1))
        model.add(Lambda(lambda a: K.expand_dims(a[:, 0], -1) + a[:, 1:] - K.mean(a[:, 1:], keepdims=True),
                         output_shape=(self.action_size,)))
    model.compile(loss="mse", optimizer=RMSprop(lr=self.learning_rate))
    return model
```

- Plot the learning curve to show the performance of your Policy Gradient on Pong

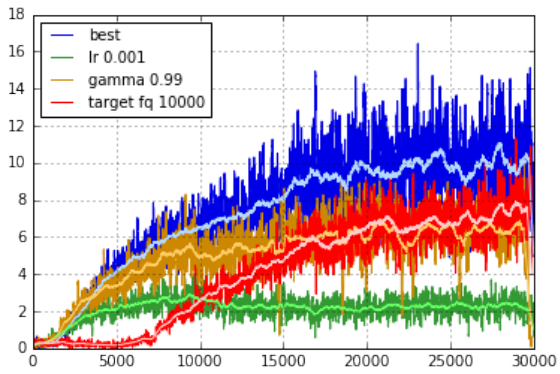


- Plot the learning curve to show the performance of your DQN on Breakout



- 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

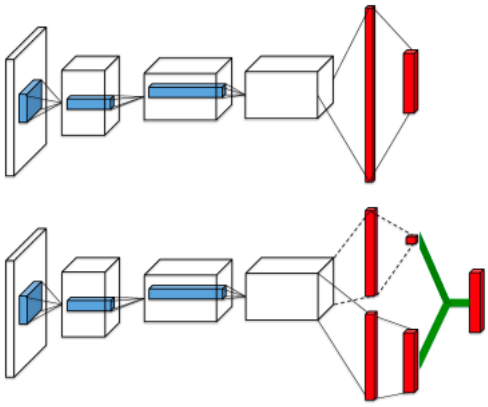


- Explain why you choose this hyperparameter and how it effect the results
  - Learning rate 0.001
    - 在 reinforcement learning 中，learning rate 的更動往往佔有非常重要得角色，所以我選擇更動此參數
    - 過大的 learning rate 會使得模型更動幅度太大，無法訓練
    - 適當的 lr 才能讓 model 有緩慢的進步
  - Target Q update frequency 10000
    - 挑選一個好的 Target Q 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

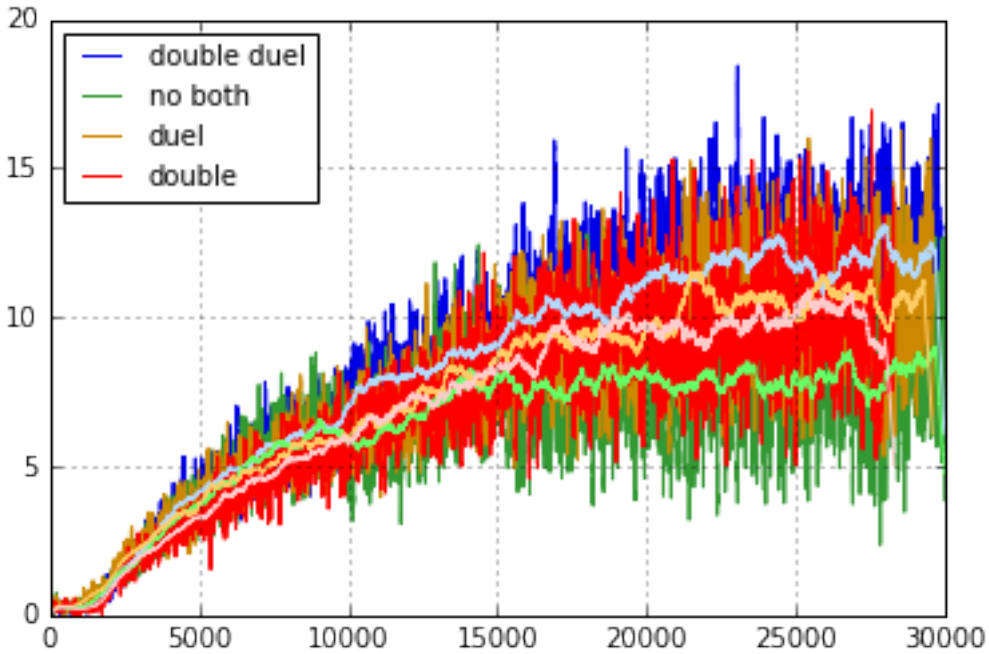
      $$Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a; \boldsymbol{\theta}_t), \boldsymbol{\theta}_t^-) .$$

      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