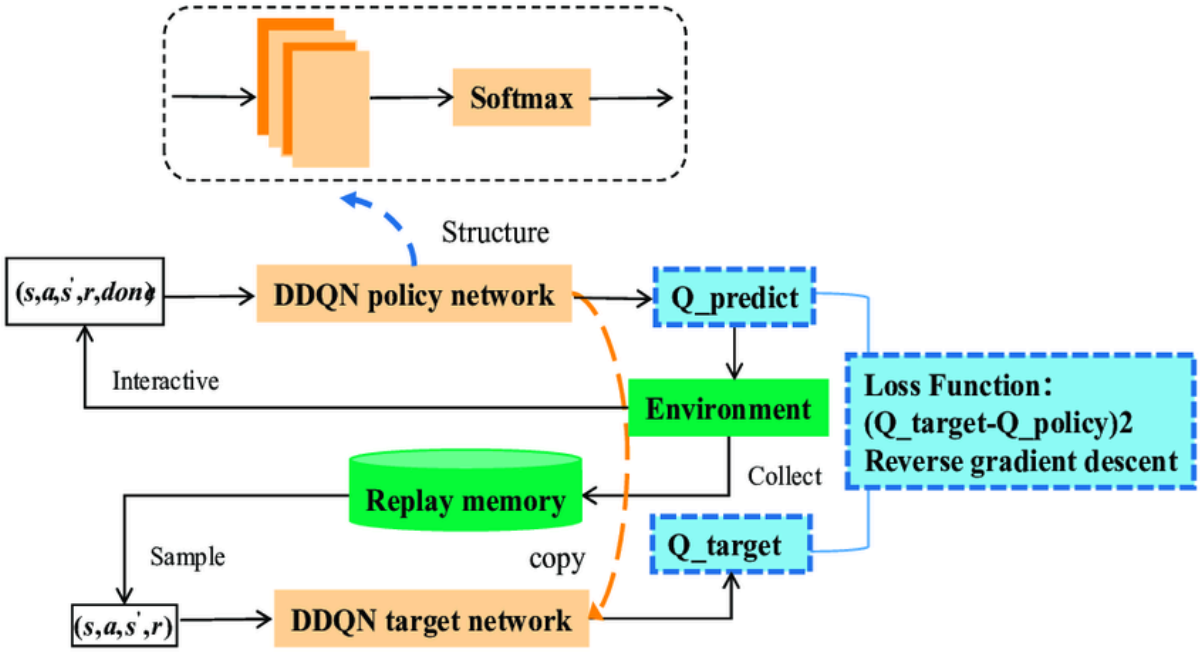
Reinforce learning hw2

1. Double DQN implement

Double Training method

- Double DQN 架構:



→ 在初始化狀態經過policy network後，預測出一個最有最大Q value的action

→ 這些Q network預測的經驗並且得到的環境新的reward會放入Replay memory中

→ target network會隨機sample replay memory之中的資料並且進行target network的訓練，並會進行td target的預測。

→ 根據target network/ policy network loss function 計算，update回 policy network , target network的參數中

- 根據policy/target network參數更新方式根據以下公式:

$$y_i = r_t + \gamma Q_{\theta^-}\left(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a')\right)$$

$$\theta = \theta - \alpha \frac{1}{N} \sum_i \nabla_\theta Q(s_i, a_i)(y_i - Q(s_i, a_i))$$

→ Q_{θ^-} 為target network ，Q_θ為policy network
→Q_θ 先計算出新的最state所有action中對應最大Q value值的action →a'
→ a' state(t+1)放入target network中， +reward變為td_target
→ td error定義為td_target-$Q(s_i, a_i)$ 使用td error 更新target network,policy network 參數

→ 有嘗試使用Duel DQN 結果沒有明顯提升，但是改變reward設定方式，及加上priotied memory 後training效果有明顯改善

2. 使用Priortized memory 優化結果

- 會將動作之後的結果，放入memeory中，在training時，會取random 取batch size 數量的data來更新q-network。
- Priortized memory為使用td_error normalized結果，作為在random取batch size 數量的data時的機率，td_error越高，其取到的機率會越高，這樣可以使參數在更新時，主要會去更新td target越大的state。

3. environment wrapper 方式

- 主要分成以下幾個動作:
  - 使用nes_py.wrappers JoypadSpace模組，將來簡化環境，及動作空間
  - 下採樣環境將pixel 數量較多的rgb環境變為灰皆，並將環境視野變小
  - 跳過幾個frame 的step變成一次step結果，代表一次training，時，會考量連續四個動作的相關性
  - 客製化環境reward結果

4. 程式架構

- environmet 初始化 如上述wrapper方式

- Agent net define:
  - 主要架構為pytorch Sequential model模型
  - 有3層捲基層及兩層linear構成

- Agent 主要包括四個函數:
  - choose action
    - Explore:random 選個方向的值
    - 若不是在exploration模式下，會根據目前state放入policy network中，選定有最高值action
  - cache memory

        - 會將環境執行action後的結果，放入agengt memory中，並根據td error做priorty排序。
    - update agent parameter
        - 會更新agent 目前step步數，及eplison值等參數。
    - learn: td error 計算，
        - 會將環境執行action後的next state,reward，進行td target，及td error的計算，並更新network

- start training: 每個episode重複以下動作直到done，或拿到flag:
    - update agent parameter
    - choose action
    - 環境執行action
    - cache memory
    - learn

- start testing:每個episode重複以下動作直到done，或拿到flag:
    - choose action
    - 計算step reward

5. 在training 時嘗試的其他優化方式

- 在exporation時，將complex movement中，向上,右即向右的機率，設定較高，
  → 發現他在一開始因exploration速率會比較快，reward會較快上升，但是在收斂到大約1000多時，整體的reward跟原本沒有差太多

- 在training時，會發現agent在某些較高的水管，其會一直無法跳過，因此嘗試在state,next state若相關係數連續個step大於0.9時，要強制切換到random explore的模式，且向右, 向上的動作機率提高。
  → 發現因為背景會就算他在原地位置沒有變化，背景也時常會一直改變，像是第一關中會一直有移動的香菇，因此這個連續相關性的偵測無法很精準。

- 降低exploration機率在每個episode下降率，控制explration 機率的參數eplison，原本是在每次episode都乘以0.99975，但是發現，即使這樣還是會下降太快，後來改用以下公式來控制exploration參數，效果會好很多
  → eplison = 0.01 + (1- episode) * exp(-1 * ((episode + 1) / 100))

- 改變envirnment reward計算方式:
    - 原本reward 加上此action產生的新的分數，跟產生的分數，可以更加獎勵具有分數增加的動作

- reward整以除以40，增加模型穩定性
  → 可以發現改成這樣的reward計算，可以大幅增加training效果

6. training結果

→ 第一關可以順利過關
→第二關大概到中途會失敗