Homework 5 Report

B02902039　資工四　李奕皓

- **Explain why DQN algorithm need these functions and how you implement them:**

  - **Experience Replay**
    將每次玩遊戲的 experience (state transition) 都存起來，稱作 replay memory，然後每個 training step 都從 replay memory 中隨機 sample 出一些 data 做為一個 batch 當作 input（而不是直接用最近的 experience），這樣可以去掉 training data 前後的相關性以獲得更好的效果。實作見 Snippet 1。

  - **Target Network**
    用 2 組 Q-network，一組是每個 training step 都會被 optimize 的 main network，另外一組是平時是固定的 target network。按照某個固定週期將 main network 中 trainable variables 的值 assign 到 target network，每次 optimize main network 時，都是根據 target network 的 Q-value，由於 target network 在大部分時候都是固定的，這樣的 training 方式會比較穩定。實作見 Snippet 2。

  - **Epsilon Greedy**
    在玩遊戲的時候每個 step 有 ε 的機率會隨機選一個 action，而不是用 DQN 認為最好的 action，因為 DQN 自己認為最好的 action 不一定是最好的，所以需要時不時嘗試別的 action，才可能會發現另外一個更好的 action 並修正判斷。實作見 Snippet 1。

  - **Clip Reward**
    將 reward 限制在 -1 與 1 之間，可以避免 Q-value 太大，也可以讓 train 的過程更加穩定。實作見 Snippet 1。

Snippet 1:

```
# Choose next action
if evalWithEpsilon is None:
    epsilon = max(.1, 1.0 - 0.9 * environment.getStepNumber() / 1e6)
else:
    epsilon = evalWithEpsilon

if state is None or random.random() > (1 - epsilon):
    action = random.randrange(environment.getNumActions())
else:
    screens = np.reshape(state.getScreens(), (1, 84, 84, 4))
    action = dqn.inference(screens)

# Make the move
oldState = state
reward, state, isTerminal = environment.step(action)

# Record experience in replay memory and train
if isTraining and oldState is not None:
```

```
    clippedReward = min(1, max(-1, reward))
    replayMemory.addSample(
        replay.Sample(oldState, action, clippedReward, state, isTerminal))

    if (environment.getStepNumber() > args.observation_steps and
            environment.getEpisodeStepNumber() % 4 == 0):
        batch = replayMemory.drawBatch(32)
        dqn.train(batch, environment.getStepNumber())
```

Snippet 2:

```
def train(self, batch, stepNumber):
    x2 = [b.state2.getScreens() for b in batch]
    y2 = self.y_target.eval(feed_dict={self.x_target: x2}, session=self.sess)

    x = [b.state1.getScreens() for b in batch]
    a = np.zeros((len(batch), self.numActions))
    y_ = np.zeros(len(batch))

    for i in range(0, len(batch)):
        a[i, batch[i].action] = 1
        if batch[i].terminal:
            y_[i] = batch[i].reward
        else:
            y_[i] = batch[i].reward + gamma * np.max(y2[i])

    self.train_step.run(feed_dict={
        self.x: x,
        self.a: a,
        self.y_: y_
    }, session=self.sess)

    if stepNumber % self.targetModelUpdateFrequency == 0:
        self.sess.run(self.update_target)
```

- **If a game can perform two actions at the same time, how will you solve this problem using DQN algorithm?**
  我們可以將每一種可能的 action 組合都列舉出來。
  例如本來的 action set 是{no-op, ↑, ↓, ←, →, A, B}，但是今天 A 鍵可以與方向鍵一起按的話，那新的 action set 就會是{no-op, ↑, ↓, ←, →, A, B, ↑+A, ↓+A, ←+A, →+A}，如此一來就可以沿用本來只能做一個 action 的 DQN。