4-1 Policy Gradient

Describe your Policy Gradient model (1%)

這份作業使用Pong這個遊戲作為task，我們使用CNN的架構：

```
CNN2(
  (main): Sequential(
    (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(3, 3))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True)
    (2): ReLU()
    (3): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2))
    (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True)
    (5): ReLU()
    (6): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2))
    (7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
    (8): ReLU()
    (9): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2))
    (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
    (11): ReLU()
  )
  (fc1): Linear(in_features=512, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=3, bias=True)
)
```

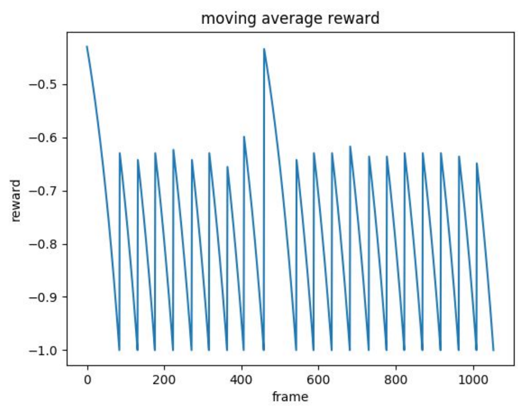
因為對Pong有一些基本了解，所以可以對每一個狀態的影像做一些簡化的預處理：

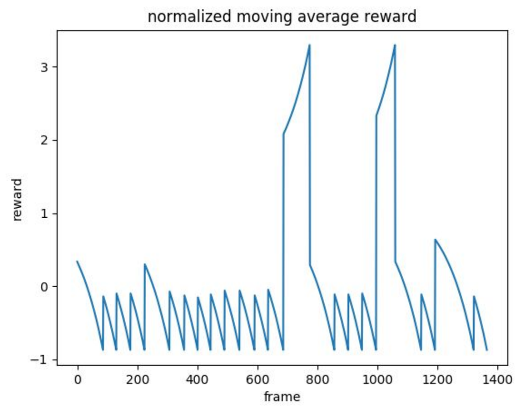
1. 將上方的計分表截去
2. 轉換成灰階
3. 將image size縮小為80x80的array
4. 將餵入model的image換成residual，定義為目前的frame減去前一個frame。

接著在Policy Gradient演算法的部份，我們實作了Reward normalization，然後gym預設的Action space有6個選項，但實際用command line一個一個去call env.step(0~5)發現其實只有3種，1=不動、2=向上、3=向下，因此我們將model的輸出定為3維。輸出之後再加1，是為了對應到environment的設定。

接著，因為一場episode裡每一次得分的事件應該是獨立的，所以只要有人得分，累積的reward應該要被清除，否則若是model得分之後馬上又失分，得分的動作可能會因為後續有失分而造成reward下降，但這兩件事不應該有關係，可能會造成model學習困難。

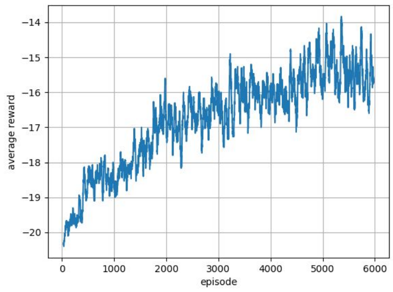
最後我們在初始化的部份採用Lecun Initialization，可以讓訓練速度更快。





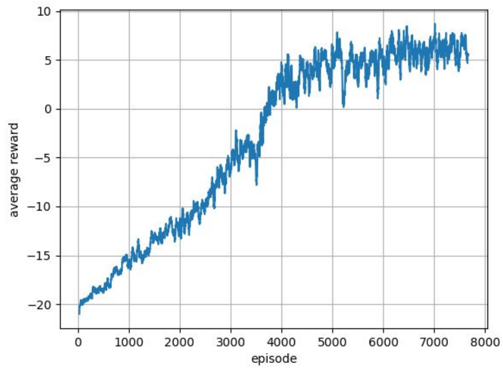
(1-a) moving average reward (1-b) normalized moving average reward

圖(1-a)是使用moving average計算的reward，每一次失分都可以將負reward往前面的動作傳遞。而圖(1-b)將reward normalized，算是比較標準的作法，但normalize的結果造成每一分剛開打時竟然有正reward，較不合理，對於這個task來說得分/失分的reward為正/負1，這樣的reward已經符合邏輯，因此認為這個task不必做normalization，下圖為normalize後的平均reward，訓練起來並沒有比較有效律。



。

Plot the learning curve to show the performance of your Policy Gradient on Pong (1%)
X-axis: episode | Y-axis: 每30個episode的平均reward（用moving average）



[Improvement]
Describe your tips for improvement (1%)
使用PPO

Learning curve (1%)

Compare to the vallina policy gradient (1%)

4-2 Deep Q Learning
Describe your DQN model (1%)

1. Hyperparameters
    a. Optimizer: RMSProp (lr = 2.5e-4)
    b. Criterior: MSELoss
    c. batch size: 32
    d. gamma: 0.99
    e. replay size: 1e4

2. Q network

```
class DQN(nn.Module):
    def __init__(self):
        super(DQN, self).__init__()
        self.conv1 = nn.Conv2d(4, 16, kernel_size=5, stride=2)
        self.bn1 = nn.BatchNorm2d(16)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=5, stride=2)
        self.bn2 = nn.BatchNorm2d(32)
        self.conv3 = nn.Conv2d(32, 32, kernel_size=5, stride=2)
        self.bn3 = nn.BatchNorm2d(32)
        self.fc1 = nn.Linear(1568, 256)
        self.fc3 = nn.Linear(256, 4)
        self.conv1.reset_parameters()
        self.conv2.reset_parameters()
        self.conv3.reset_parameters()

    def forward(self, x, batch=1):
        x = F.relu(self.bn1(self.conv1(x)))
        x = F.relu(self.bn2(self.conv2(x)))
        x = F.relu(self.bn3(self.conv3(x)))
        x = x.view(batch, -1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc3(x)
        return x
```

3. Training process
    a. main loop (update target model every 2500 step)

```
replay_buffer = Replay2(self.max_buffer)
step = 0
for episode in range(self.episode):
    state = self.env.reset()
    state_pool, action_pool, next_state_pool, reward_pool = [], [], [], []
    state_pool.append(state)

    while True:
        step += 1
        action = self.play(state, step)
        next_state, reward, done ,_ = self.env.step(action)
        reward = max(-1.0, min(reward, 1.0))
        action_pool.append(action)
        reward_pool.append(reward)
        next_state_pool.append(next_state)

        if len(replay_buffer) >= self.max_buffer:
            if step % 4 == 0: # replay
                loss, Q = self.update_model(replay_buffer)
            if step % 2500 == 0: # update Q
                self.target.load_state_dict(self.Q.state_dict())
            if done:
                for i in range(len(state_pool)):
                    replay_buffer.push(state_pool[i], action_pool[i], next_state_pool[i], reward_pool[i])
                break
        else:
            if done:
                for i in range(len(state_pool)):
                    replay_buffer.push(state_pool[i], action_pool[i], next_state_pool[i], reward_pool[i])
                break
        state = next_state
        state_pool.append(state)
```

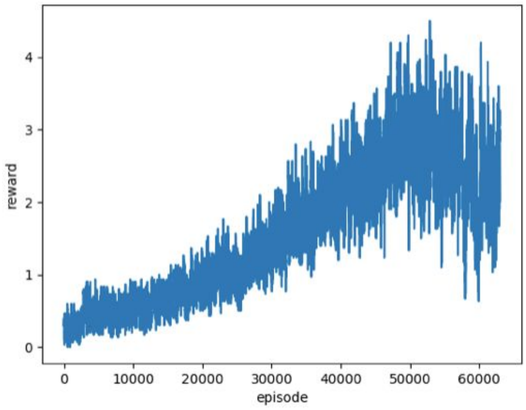
    b. play (self.epsilon = 1.0 as initial condition)

```
def play(self, obs, step):
    sample = random()
    if sample > self.epsilon - 0.9 * (step/10e6):
        obs = torch.from_numpy(obs).type(dtype).permute(2, 0, 1).unsqueeze(0)
        return self.Q(Variable(obs, volatile=True)).data.max(1)[1].cpu().numpy()[0]
    else:
        return randint(0,3)
```

c. replay/ update model (every 4 step)

```
def update_model(self, replay_buffer):
    transitions = replay_buffer.sample(self.batch)
    batch = Transition(*zip(*transitions))
    s = Variable(torch.from_numpy(np.asarray(batch.s)).type(dtype).permute(0,3,1,2)) #(32,4,84,84)
    a = Variable(torch.from_numpy(np.array(batch.a)).long()) #32
    r = Variable(torch.FloatTensor(batch.r)) #32
    s_next = Variable(torch.from_numpy(np.asarray(batch.next_s)).type(dtype).permute(0,3,1,2), volatile=True)
    if USE_CUDA:
        a, r = a.cuda(), r.cuda()
    current_Q = self.Q(s, self.batch).gather(1,a.unsqueeze(1)).squeeze() #(32,1)
    v_pred = self.Q(s_next, self.batch).max(1, keepdim= True)[1]
    next_max_Q = self.target(s_next, self.batch).gather(1, (v_pred)).detach().squeeze()
    next_max_Q.volatile = False
    target_Q = next_max_Q * 0.99 + r #32
    self.optimizer.zero_grad()
    d_error = self.criterion(current_Q, target_Q)
    d_error.backward()
    for param in self.Q.parameters():
        param.grad.data.clamp_(-1, 1)
    self.optimizer.step()
    return torch.mean(d_error.cpu()).data.numpy()[0], torch.mean(current_Q.cpu()).data.numpy()[0]
```

Plot the learning curve to show the performance of your Deep Q Learning on Breakout (1%)

	x軸：episode數 y軸：clipped reward (averaging last 30 episodes)

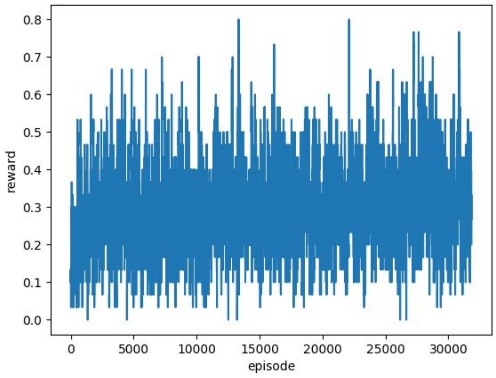
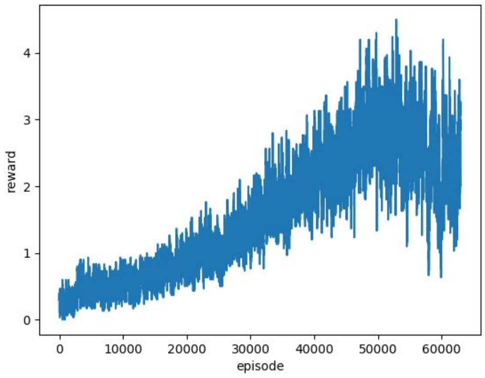
Implement 1 improvement method on page 6
Describe your tips for improvement (1%)

我們採用了double DQN的模式下去train，基本上只是在產生target Q的過程改成由Q network本身下去預測要執行的action。

```
current_Q = self.Q(s, self.batch).gather(1,a.unsqueeze(1)).squeeze() #(32,1)
v_pred = self.Q(s_next, self.batch).max(1, keepdim= True)[1]
next_max_Q = self.target(s_next, self.batch).gather(1, (v_pred)).detach().squeeze()
next_max_Q.volatile = False
target_Q = next_max_Q * 0.99 + r #32
self.optimizer.zero_grad()
d_error = self.criterion(current_Q, target_Q)
```

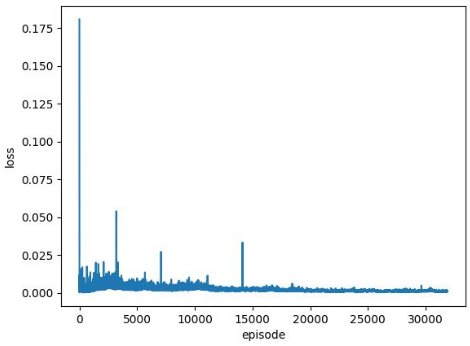
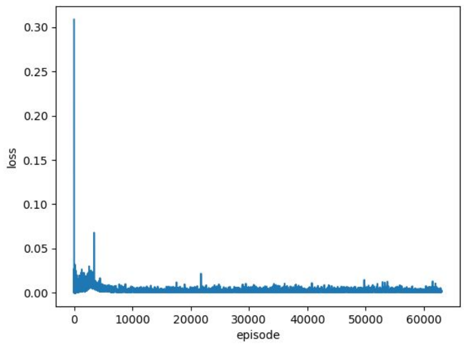
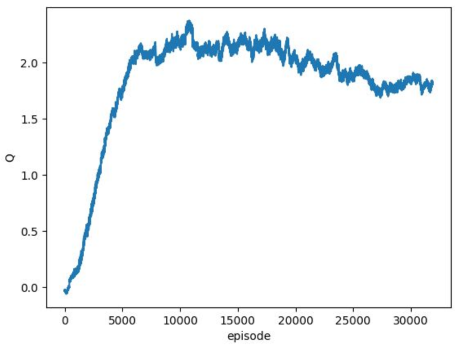
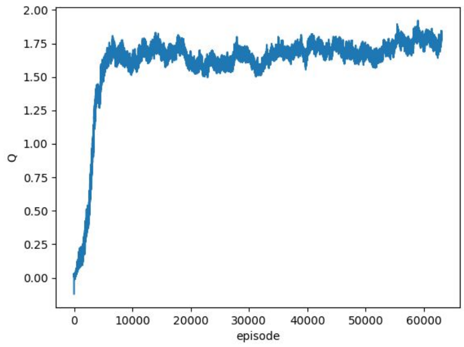
Learning curve (1%)

Naive DQN	Double DQN

Compare to origin Deep Q Learning(1%)

learning curve （上題所示）顯示在這個模型中，naive DQN幾乎沒有學到東西，DDQN的表現則相對較好。Loss表現則差不了太多，

	Naive DQN	Double DQN
loss		
Q		

4-3 Actor Critic

Describe your actor-critic model on Pong and Breakout (2%)

Plot the learning curve and compare with 4-1 and 4-2 to show the performance of your actor-critic model on Pong & Breakout (2%)
X-axis: number of time steps
Y-axis: average reward in last 100 episodes

Reproduce 1 improvement method of actor-critic (Allow any resource)
Describe the method (1%)

Plot the learning curve and compare with 4-1 and 4-2, 4-3 to show the performance of your improvement (1%)