DRL final project

第一組

楊晶宇 伍麒 常安彥

Outline

Episodic Backward Update

- Sample a whole episode from replay memory
- Propagate the value for entire transitions in backward manner
- Diffusion factor is set to overcome overestimate error

Sample a random episode $E = \{\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \boldsymbol{S'}\}$ from D, set $T = \text{length}(E)$
Generate a temporary target Q-table, $\tilde{Q} = \hat{Q}\left(\boldsymbol{S'}, \cdot; \boldsymbol{\theta}^{-}\right)$
Initialize the target vector $\boldsymbol{y} = \text{zeros}(T)$, $\boldsymbol{y}_T \leftarrow \boldsymbol{R}_T$
for $k = T - 1$ to 1 **do**
    $\tilde{Q}\left[\boldsymbol{A}_{k+1}, k\right] \leftarrow \beta \boldsymbol{y}_{k+1} + (1-\beta)\tilde{Q}\left[\boldsymbol{A}_{k+1}, k\right]$
    $\boldsymbol{y}_k \leftarrow \boldsymbol{R}_k + \gamma \max_a \tilde{Q}\left[a, k\right]$
end for
Perform a gradient descent step on $\left(\boldsymbol{y} - Q\left(\boldsymbol{S}, \boldsymbol{A}; \boldsymbol{\theta}\right)\right)^2$ with respect to $\boldsymbol{\theta}$

Implementation

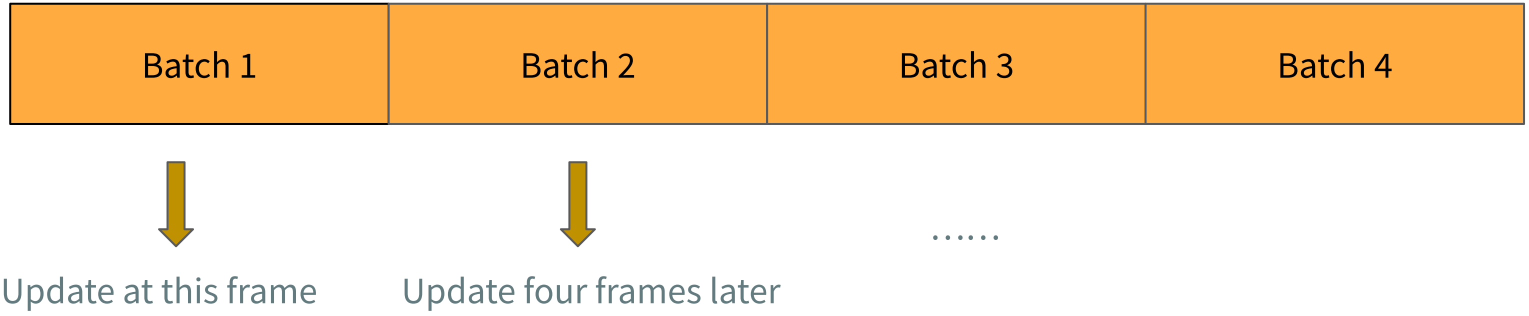
- Calculate Q_tilde with target network
- Generate y with Q_tilde, actions, Beta, discount and rewards

```
if cur_frame % 4 == 0:
    if batchnum == batchcount:
        states, actions, rewards, next_states, dones = buffer.sample()
        transition_length = len(actions)
        batchnum = ceil(transition_length / batch_size)
        batchcount = 1
        target_net.eval()
        _Q = target_net(next_states) #(sample_length, num_action)
        target_net.train()
        _Q[-1,:] = torch.as_tensor([0] * numOfAct)
        y = np.zeros(transition_length)
        y[-1] = rewards[-1]
        for i in range(transition_length - 2, -1, -1):
            _Q[i][actions[i+1]] = B * y[i+1] + (1 - B) * _Q[i][actions[i+1]]
            y[i] = rewards[i] + discount * torch.max(_Q[i])
```

Implementation

- Sample a batch size of transitions and update with optimizer
- Sample another batch size from the same trajectory until encountering the termination flag

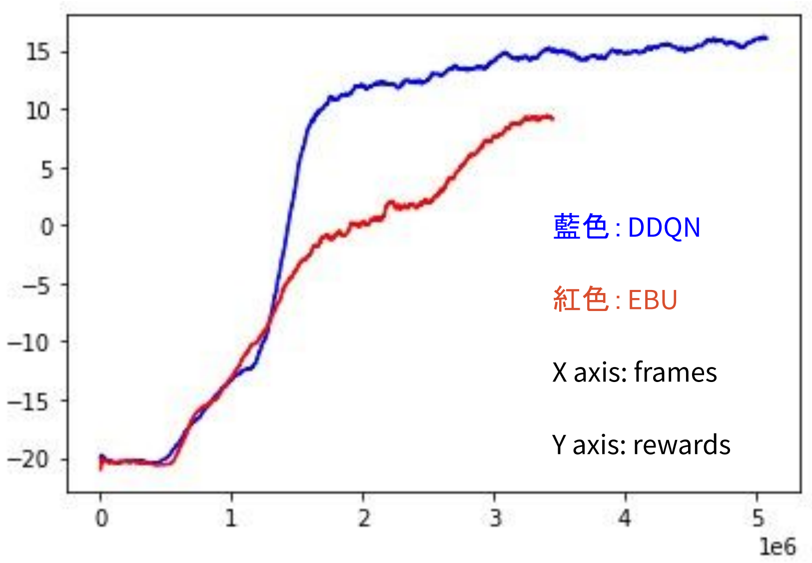
Whole episode from buffer: a trajectory



Results

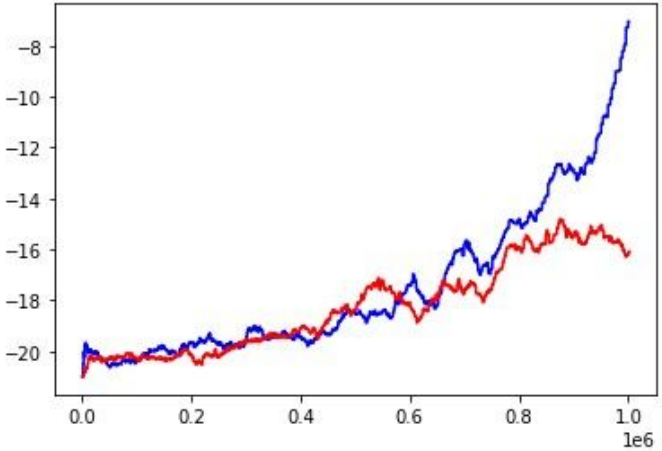
- Adam with low learning rate: 1e-5
- β = 0.5

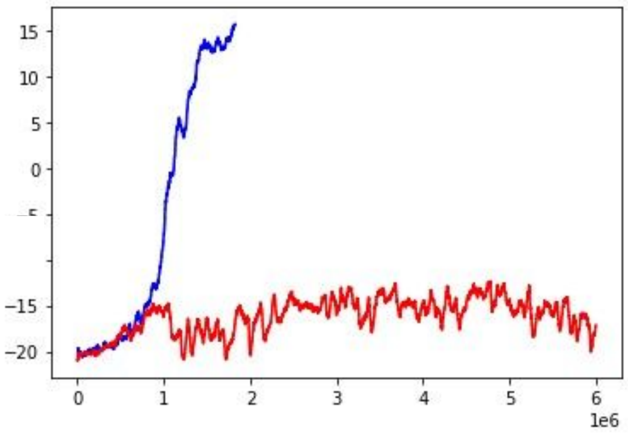
- Discuss :
  - 在低lr的時候EBU是能夠train得起來的
  - 在前1.5e6的時候表現有稍微超過DDQN，只是後面就輸掉了



Results

- RMSprop with higher learning rate: 2.5e-4
- β = 1
- Trajectory saved with life loss, not termination
- Discuss
    - 前100萬個frame看起來競爭力都很夠
    - 100萬之後就會開始震盪上不去
    - Agent的表現很不穩定，有時候last 25 eps的平均很低，但是測試單局的分數又很高

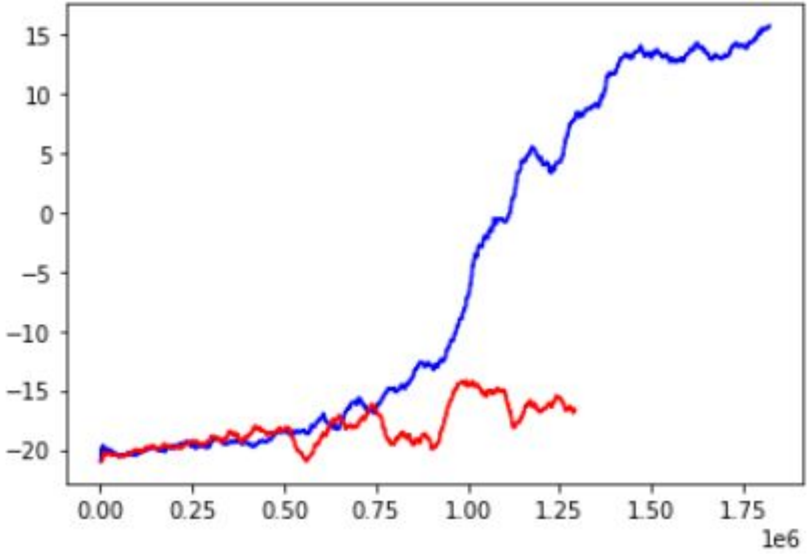


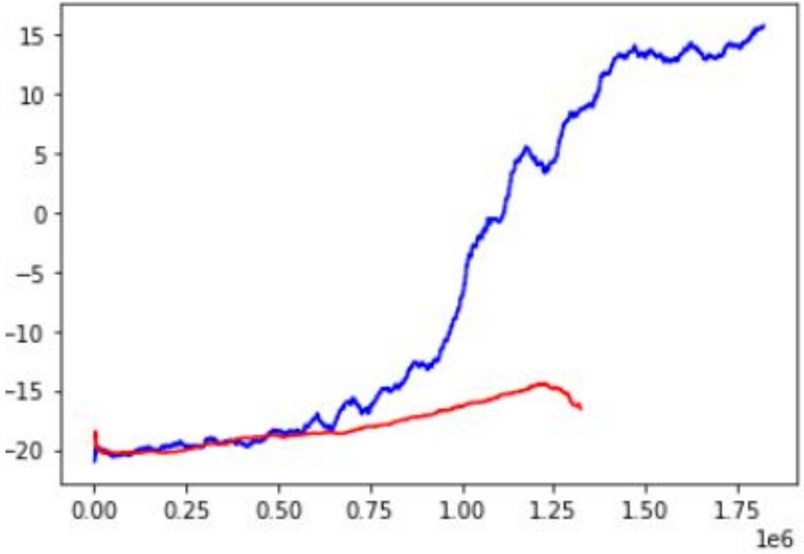



```
Episode: 2880/5000, Epsilon: 0.100, Loss: 1.1884658, Return: -18.60
current episode reward:  21.0
```

Other Results

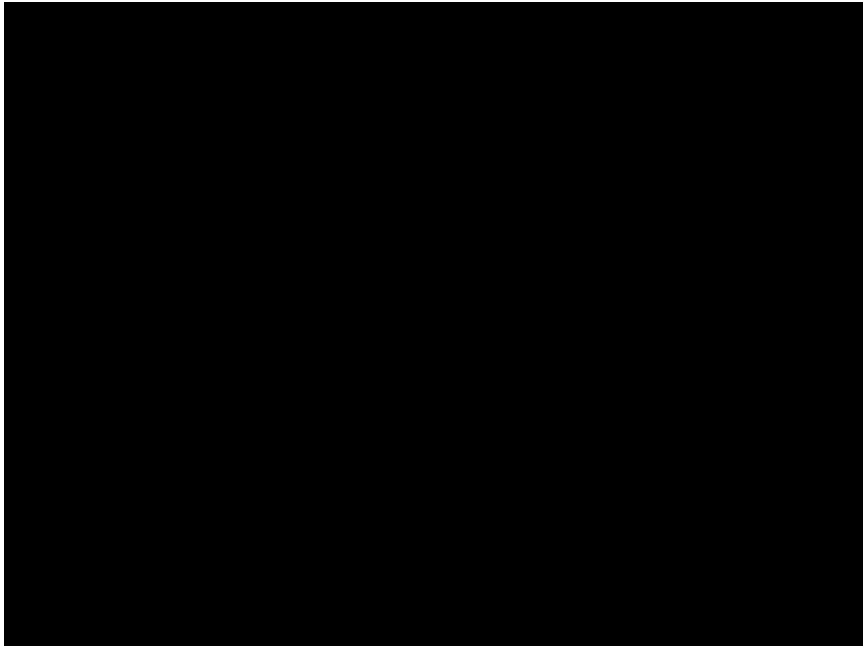
- Trying different parameters



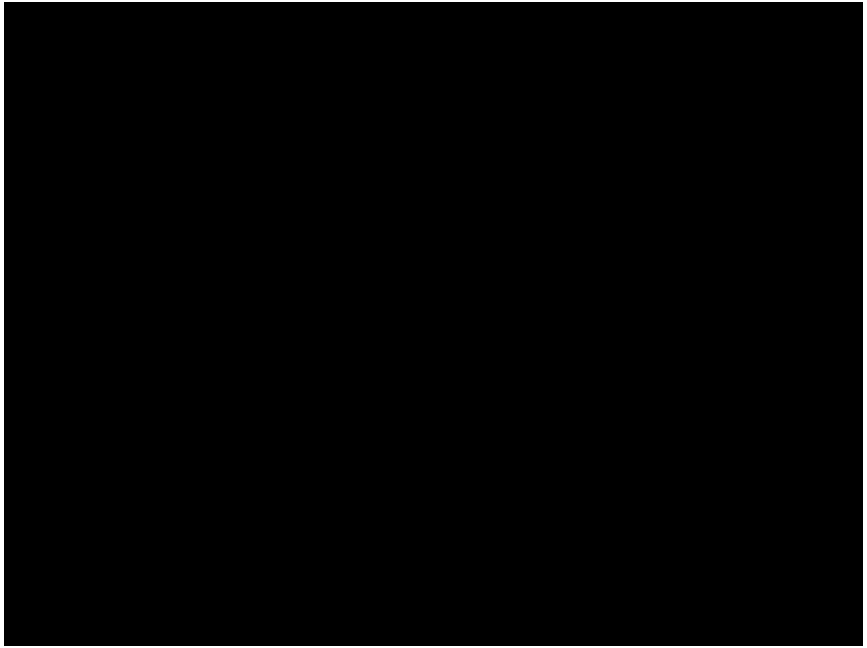


Behavior

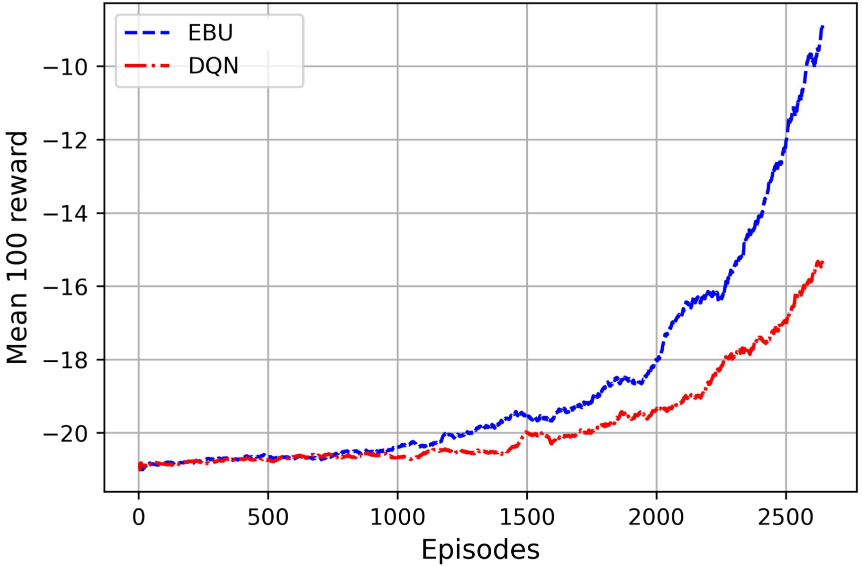
- DDQN



- EBU



Final experiment



Conclusion

$$\tilde{Q}\left[\mathbf{A}_{k+1}, k\right] \leftarrow \beta \boldsymbol{y}_{k+1} + (1-\beta)\tilde{Q}\left[\mathbf{A}_{k+1}, k\right]$$

- Diffusion factor設為一致的效果在Pong的環境中的表現較DQN差
- adaptive版本的表現比其他model好(用不同的β去train很多個target network，再選表現較好的network去update)，如果是固定的β去做training，model能train到的程度感覺很吃運氣，很容易因為sample到的trajectory不夠好讓loss重新掉回好幾個eps前的表現
- 會說sample到的trajectory影響對model影響很大是因為我們有把agent evaluation的video 定期拿出來看，常常原本agent有學會接球，但過了幾個eps後又變成不會接發球，直接卡在最上方或最下方，我們推測是因為可能取到太多次發球在同一邊的trajectory，造成model在遊戲一開始的對策一直變動

Thanks for Listening!