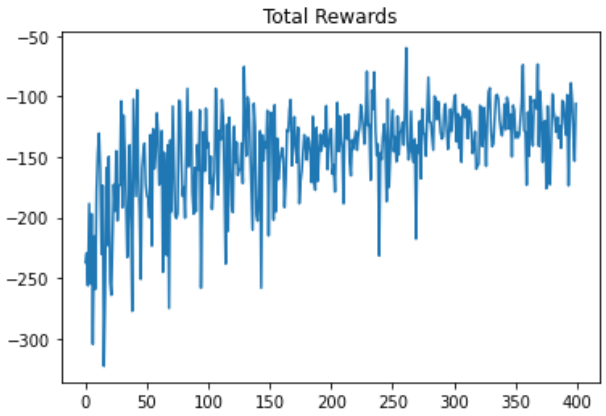
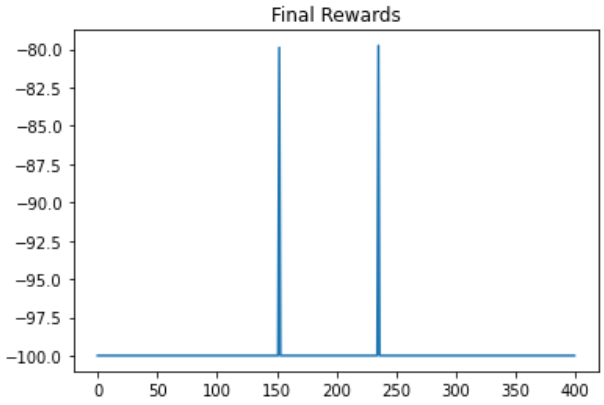
學號：b07901112　　系級：電機二　　姓名：劉聿珉

1. (20%) Policy Gradient 方法

   a. 請閱讀及跑過範例程式，並試著改進 reward 計算的方式。

      Total reward 為 -145.88239693995826

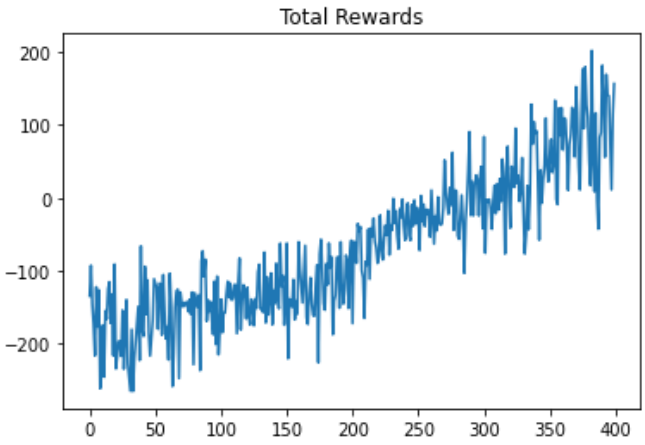


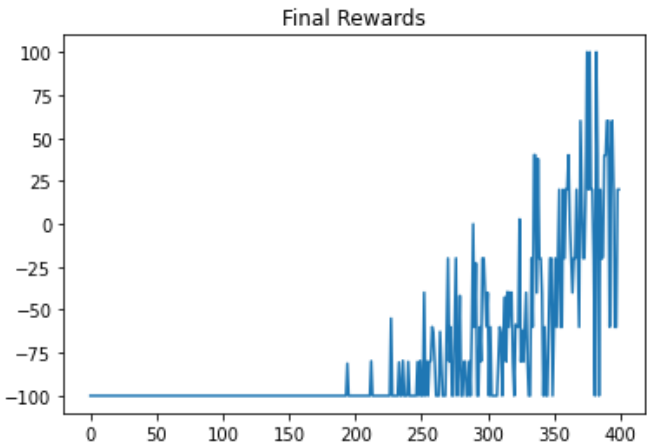


      從結果可以看到最後的 Total Rewards 大概都在-140 左右震盪，而 Final Rewards 則大部分都為-100 表示飛船經常墜毀。

b. 請說明你如何改進 reward 的算法，而不同的算法又如何影響訓練結果？

我是使用 discounted rewards 的方法 decay rate 為 0.98。最後測試的結果 Total rewards 為 75.73378 相較沒有使用 discounted rewards 的確進步很多而且 Final Rewards 也沒有一直貼在-100 上，表示飛船墜毀的機會的確有降低。



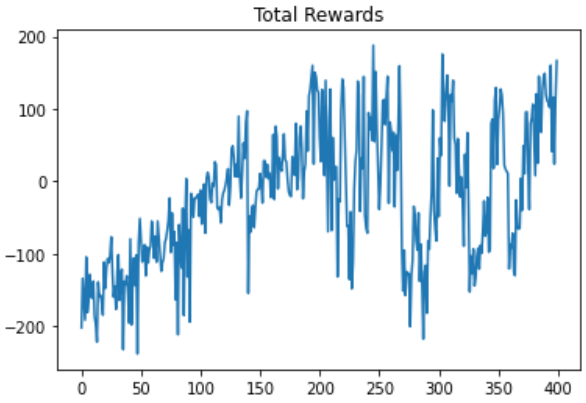


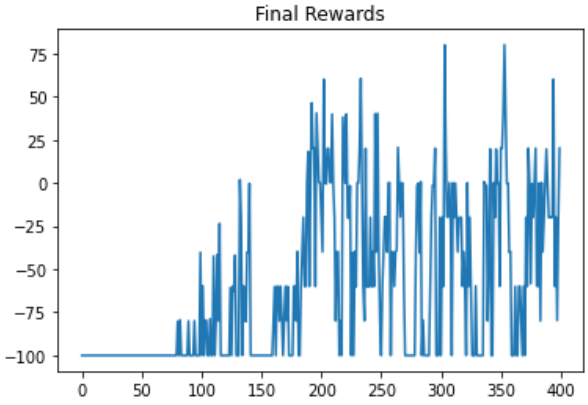
2. (30%) 試著修改與比較至少三項超參數（神經網路大小、一個 batch 中的回合數等），並說明你觀察到什麼。

這部分的改動我都是建立在 1(b)有做 discounted rewards 的基礎上下去改參數。

第一個我改變神經網路的大小，把他變寬詳細大小如下：

```
self.fc1 = nn.Linear(8, 128)
self.fc2 = nn.Linear(128, 64)
self.fc3 = nn.Linear(64, 4)
```

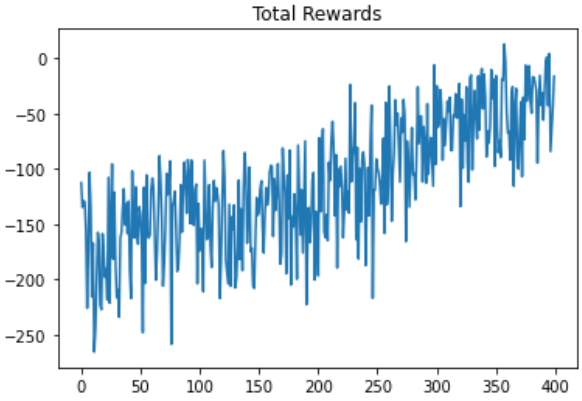


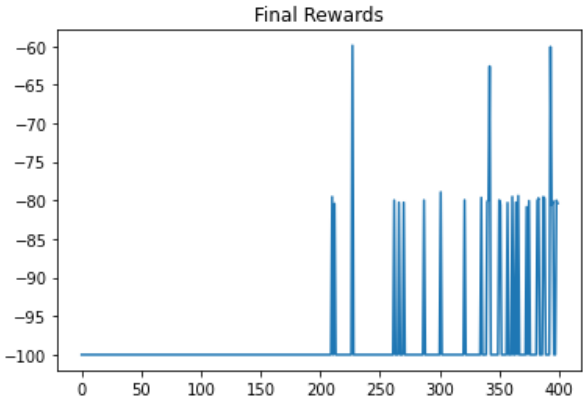


Total rewards = 101.43321

從圖中可以看到將神經元變大之後 Total reward 的振幅明顯變大許多，Final Rewards 的圖可以看到雖然不是-100 的值變多了(飛船墜毀情形有得到改善)可是最高值只有 75 左右，而且對非-100 的 final rewards 來說平均 Final Rewards 並沒有很高。

Optimizer 改成 Adam：

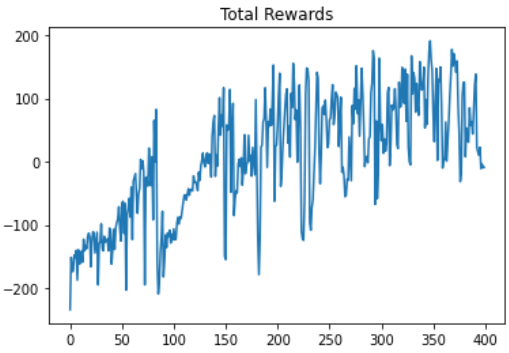


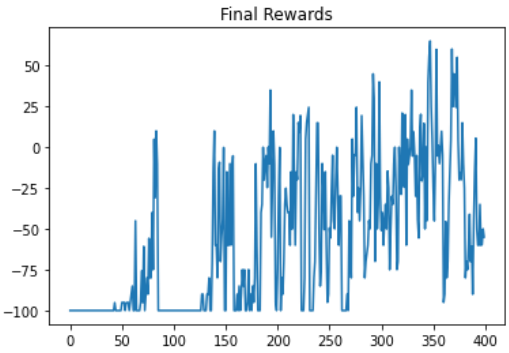


Total Reward：-29.59370455428829

Adam 相較於 SGD 收斂的比較快，且當 lr 大的時候比較不會震盪，但是在最後收斂時效果可能不比 SGD 好。從結果來看，光只把 optimizer 改成 Adam 的效果非常不好，不僅 Total rewards 下降許多 Final rewards 也糟很多，可見在這次的作業之中使用 Adam 並不是一個好的方法。

把 EPISODE_PER_BATCH 改成 20(原本 5)：





Total rewards = 87.21121

增加採樣次數，比較有機會出現特殊的情形還有動作。Final reward 的表現雖然不是-100 的值變多了，可是對非-100 的 final rewards 來說平均 Final Rewards 並沒有很高。

從結果可以看到除了 optimizer 不能用 Adam 之外，把神經元加大還有增加 EPISODE_PER_BATCH 都可以讓 final rewards 有比較好的行為，total reward 也有上升一些，但是 final rewards 還是沒辦法達到 100，所以之後有進行一些策略補足這部分的問題(下面會再討論)。

3. (20%) Actor-Critic 方法

   a. 請同學們從 REINFORCE with baseline、Q Actor-Critic、A2C 等眾多方法中擇一實作。

      我實作的是 REINFORCE with baseline，有額外建立一個 network 當作 Critic 結構如下：

```
class BaselineNetwork(nn.Module):
  def __init__(self):
    super().__init__()
    self.fc1 = nn.Linear(8, 64)
    self.fc2 = nn.Linear(64, 32)
    self.fc3 = nn.Linear(32, 1)
  def forward(self, state):
    hid = torch.relu(self.fc1(state))
    hid = torch.relu(self.fc2(hid))
    return self.fc3(hid)
```

      EPISODE_PER_BATCH=10
      NUM_BATCH = 400
      有使用 discounted reward(跟 1b 一樣)

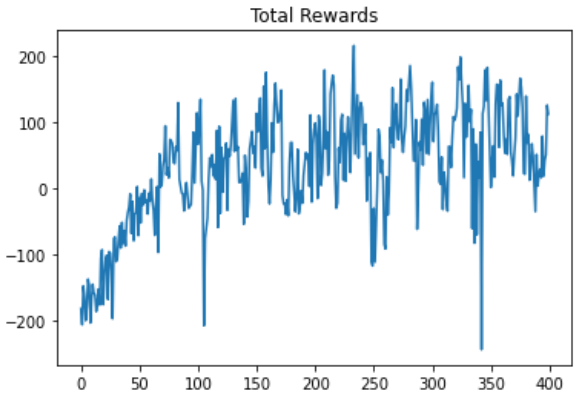
      PolicyGradientNetwork 的架構：

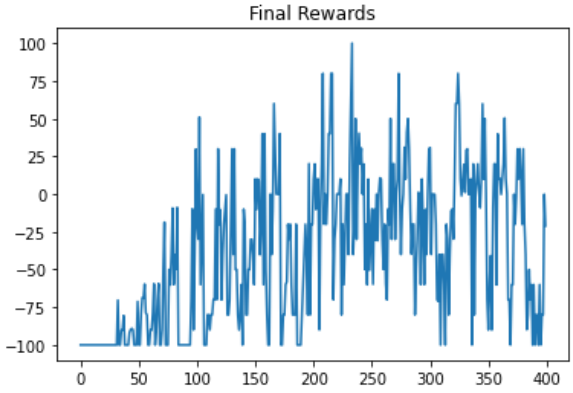
```
class PolicyGradientNetwork(nn.Module):

    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(8, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 4)

    def forward(self, state):
        hid = torch.tanh(self.fc1(state))
        hid = torch.tanh(self.fc2(hid))
        return F.softmax(self.fc3(hid), dim=-1)
```

實驗結果：





Total rewards = 115.74834

b. 請說明你的實做與前者（Policy Gradient）的差異。

從結果可以看到 Total Rewards 還是在 100 上下震盪，Total rewards 有變高一些，飛船降落的情形也有好上許多，非-100 的 Final Rewards 有大幅增加。但是總體來說大致上的圖形還是沒有太大的改變，飛船還是沒辦法穩定有效的順利降落，可以再更進步。我認為改進方法應該是可以外加一些條件去限制飛船的行為，現在的飛船可能會瘋狂噴氣去解決不會墜落的問題或是學到一些奇怪的技巧去避免受到處罰，這樣的結果並不是我們樂見的，我們希望飛船就是乖乖的降落得到獎勵，而不是僅滿足於避免受到懲罰。

4. (30%) 具體比較（數據、作圖）以上幾種方法有何差異，也請說明其各自的優缺點為何。

經過上面的實驗可以發現使用 discounted rewards 可以大增加飛船降落的成功率並且也可以使 total rewards 上升許多，但是僅僅增加 EPISODE_PER_BATCH 或是神經元的寬度只能讓 Final rewards 的曲線盡量

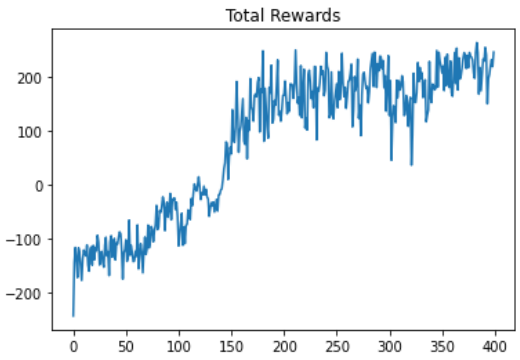
不要貼在-100 上面，但是高度並沒有得到提升，並且對於 total rewards 並沒有太多正面的影響，只有上升一點點而已，同時也會增加 total rewards 震盪的幅度。使用 REINFORCE with baseline 加上一些超參數的改動變可以稍稍改進 final rewards 的問題。而在這次的實作之中 optimizer 並不適合使用 Adam(將會是一場災難)。

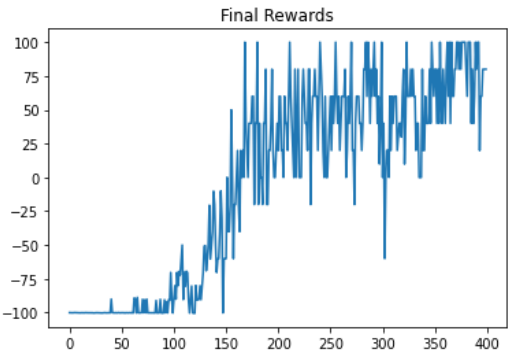
有鑑於我們的飛船還是沒辦法穩定下降，跟同學討論之後我們多做了一些人為的限制使他的表現變得更好，並且將資料再重 train 一遍方便討論。

我們為了不讓飛船亂噴，所以我們會強迫飛船下降，當高度大於 0.3 且垂直速率高於-0.1 時，施加-0.3 的 reward，且當飛船已經著地之後若他還是亂噴我們便會給他負的 reward

實驗結果如下：

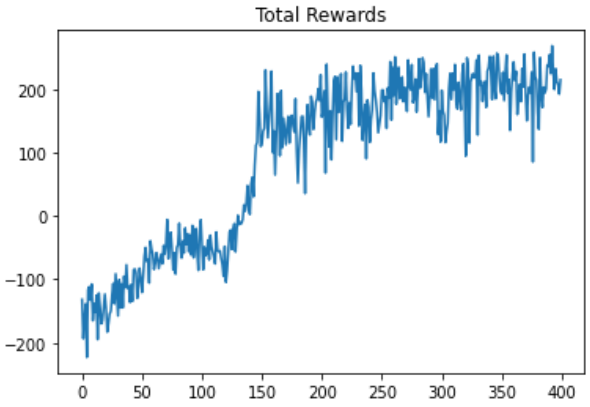
有使用 REINFORCE with baseline：

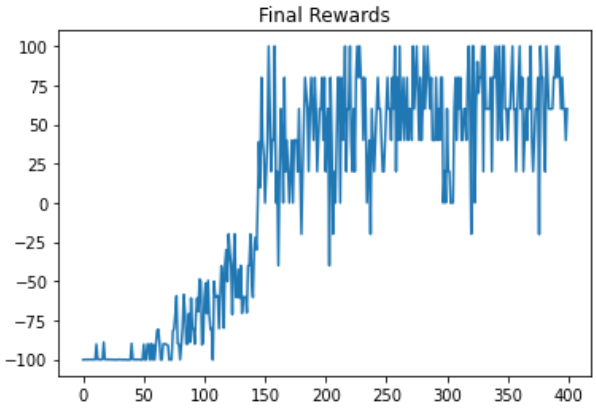




Total reward = 245.94243

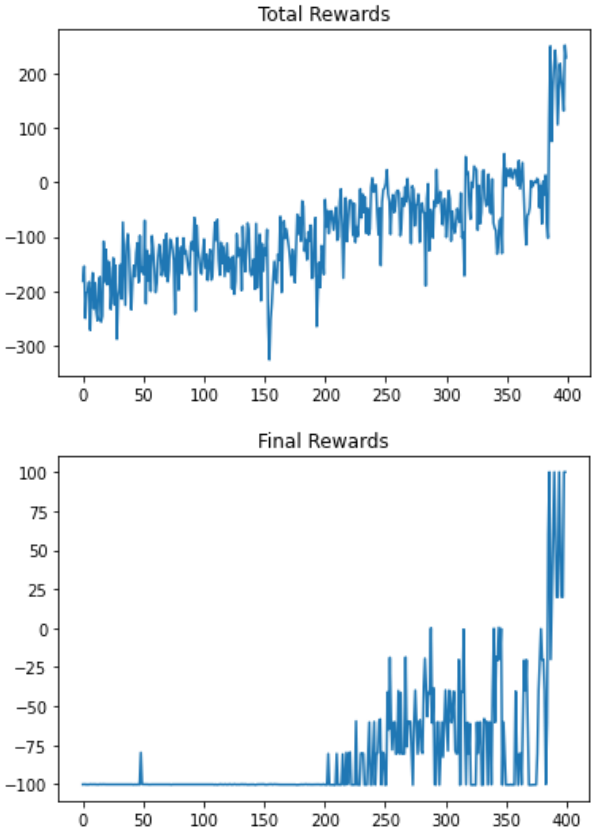
更改超參數(我的 optimizer 使用 SGD 因為之前的實驗結果顯示 Adam 成效不彰)：





Total reward = 210.64538

Policy gradient：



Total reward = 163.00861

從圖表裡面可以看到不管是 total reward 還是 final reward 他們的成效都比之前好上不少，total reward 可以停在差不多 200 左右的地方而 final reward 也可以到達 100。不管從哪角度來說，actor-critic 都完勝 policy gradient，不僅 total rewards 可以收斂在更高的點，final reward 也有更高的機率可以拿到 100 分，在實際測試後還發現它可以停在更中間的點，著實聰明。

至於有跟改超參數以及沒有更改超參數的比較，我們可以看到有更改參數的 total rewards.的收斂速度比較快也收斂在比較高的位置 Final reward 表現也明顯好很多。

從以上的經驗可以知道要求機器學習到一些人類覺得很直覺的東西其實並沒有那麼容易，因為他們有可能只會為了避免被處罰而忽略了遊戲的目的，所以適時的增加一點聰明的訓練策略會使整體的訓練成果有很大的進步，甚至會比更改參數以及架構還要有用一些。