Machine Learning HW15 Report

學號：B06901045 系級：電機三 姓名：曹林熹

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。

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？

ANS：

- Rewards - Sample code Rewards

說明我的改進方法前，我先附上我的 Policy Gradient Network 當作我們的控制變因，下圖顯示了我們的 Network 以及 Optimizer 架構，這邊全部都是助教的 sample code，並沒有做任何更改。此外，我們使用了

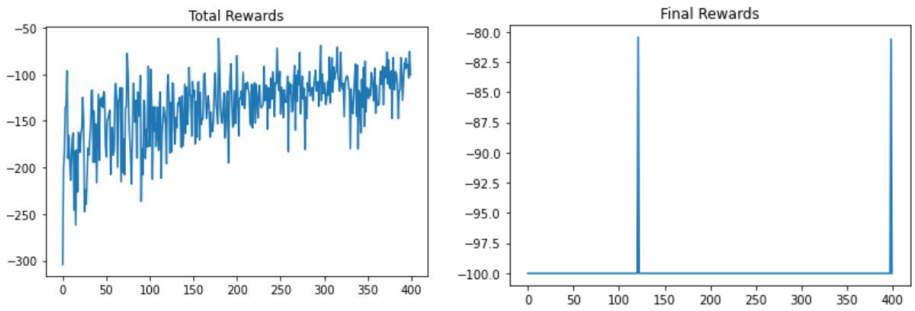
EPISODE_PER_BATCH = 5（代表每蒐集 5 個 episodes 更新一次 agent）
NUM_BATCH = 400 （代表總共更新 400 次）

```
PolicyGradientNetwork(
  (fc1): Linear(in_features=8, out_features=16, bias=True)
  (fc2): Linear(in_features=16, out_features=16, bias=True)
  (fc3): Linear(in_features=16, out_features=4, bias=True)
)
SGD (
Parameter Group 0
    dampening: 0
    lr: 0.001
    momentum: 0
    nesterov: False
    weight_decay: 0
)
```

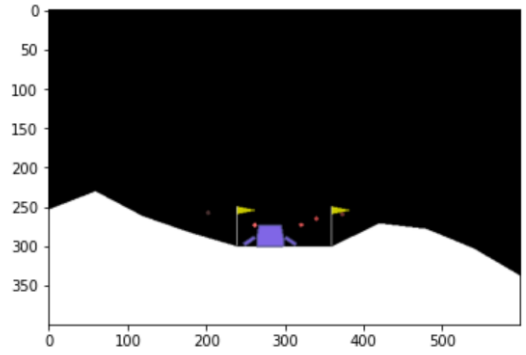
上面的參數固定住後，在我們的 training process 有寫出 reward 計算方式，在這裡把每一個 batch 計算五個 episode 後，對這個 batch 做正規化。

下圖給出了我們 training 的結果，可以看到 Total Rewards （左圖） 都是負的，趨勢上有逐漸愈來愈好，而在 Final Rewards （右圖） 上，有少量次在最後一個 action 表現得比較好，獲得比較高的分數。

Ps. （x-axis = num of episode, y-axis = total_reward）



最後，拿去我們的 test，不過要注意的是因為 action 是機率分布，所以在每一次 episode 的 total_reward 會有差異，此次 test 的 total_reward 如下圖所示。



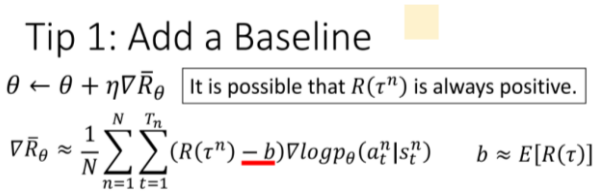
```
[18] print(total_reward)

     -24.36661415414649
```

- Rewards - improved

在這部分，我主要修改了給予 reward 的計算方式，並且持續調整細節，更改內容主要有兩個 tips 如下：

a. Tip 1: Add a Baseline

此作法為把所有的 total_reward 減去一個 baseline，這樣的用意可以避免所有出來的 total_reward 都是正的，因為假如使整場遊戲 total_reward 是正的話，機器會就增加他出現的機率（反之負的話就減少），但因為可能很容易都是正，因此要加一個 baseline b。

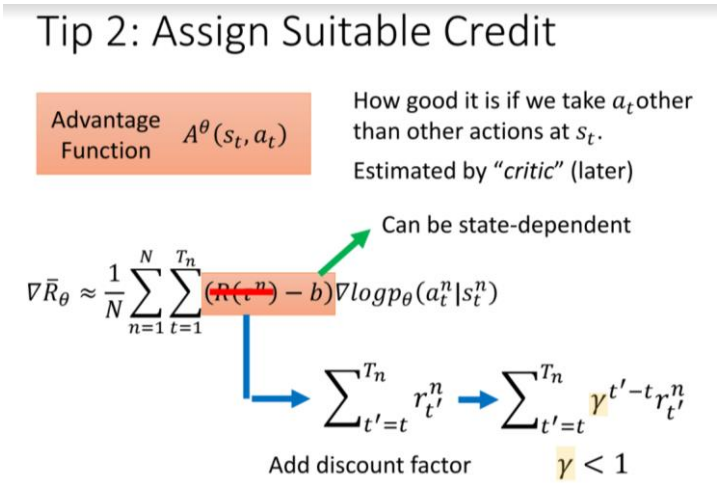


b. Tip 2: Assign Suitable Credit

計算 loss 時每個 episode 都是使用相同的 total_reward，那我們要改成在相同的 episode 下，也要使用不同的 total_reward，目的在於比較早出現的 action 會影響整個程序比較多，因此給予的 reward 也要比較大。改的地方主要有兩個：

1. 每個 episode 的期望值改成從時間 t 到結束 Tn 的 reward 加總

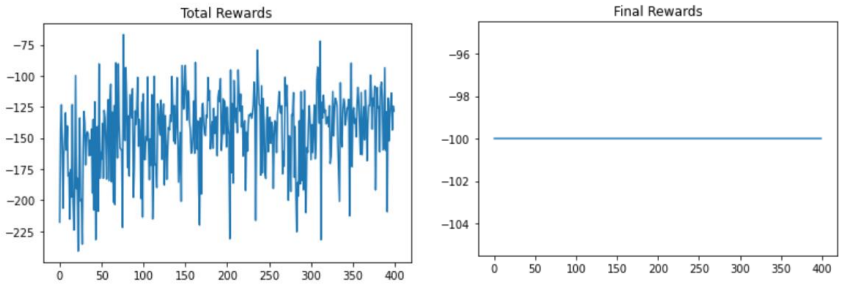
2. 多乘一個係數 γ，使愈前面 action 的 reward 較大



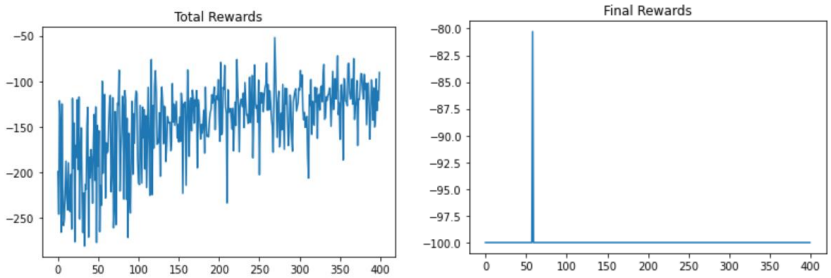
實作的方法可從下圖程式碼看到，tip_2 是一個 list 存了我們的 new_reward (gamma 設定 0.99)，並且所有元素同時剪去 tip_2 的平均值 (basline)。

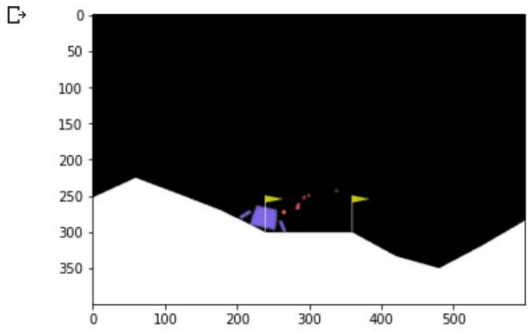
```
if done:
    for i in range(len(tip_2_temp)):
        tip_2.append((gamma**i)*sum(tip_2_temp[i:])) # implement tip_2

    final_rewards.append(reward)
    total_rewards.append(total_reward)
    # rewards.append(np.full(total_step, total_reward))  # 設定同一個 episode 每個 action 的 reward 都是 total reward (same rewards)
    # 產生 dim = 1*total_step， value = total_reward 之 array

    tip_2 = np.array(tip_2)
    rewards.append(tip_2 - np.mean(tip_2))  # implement tip_1 + tip_2
```

綜合兩個方法並實作後，下圖給出了我們 training 的結果，可以看到 total rewards （左圖） 都是負的，我感覺沒有愈來愈好，而在 Final Rewards （右圖） 上，都是 -100。拿去 test 後可以得到 -97.28 成績，但是很納悶為何並沒有將 total_reward 提升，因此我又做了一些調整。



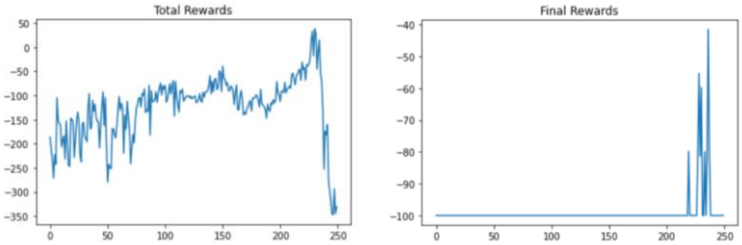
我把剛剛的 baseline 從 Mean 換成 -100，結果發現 training process 有起色，Total Rewards 有明顯愈來愈好的趨勢，但是 Final Rewards 仍然差不多；而在 test 的部分，最終拿到了 0.22 的成績，比 sample code 還要好。



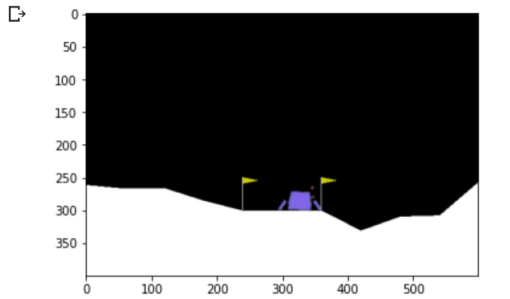


```
[86] print(total_reward)

     0.2234179957221869
```

更進一步，我把上述實作的模型加入 SGD momentum = 0.9 重新訓練，將訓練次數改成 250 次。在 RL 訓練時我認為每次的分數都落差很大，加入了 momentum 的話，有可能很幸運地跳到更好的 maximum，但也有可能不小心又脫離了，反而使訓練更好的模型更慘（如下圖）。因此我在訓練的時候，將每次 epoch 都記錄當下的 total_reward，如果他比最好的 total_reward 高，我就把這個模型參數存下來，這樣一來就可以將表現最好的模型存下來了。



最後，拿去我們的 test，可以看到飛船安全降落在旗幟內，此次 test 的 total_reward 如下圖所示，有高的成績。



```
[9] print(total_reward)

19.69272582251729
```

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。

ANS：

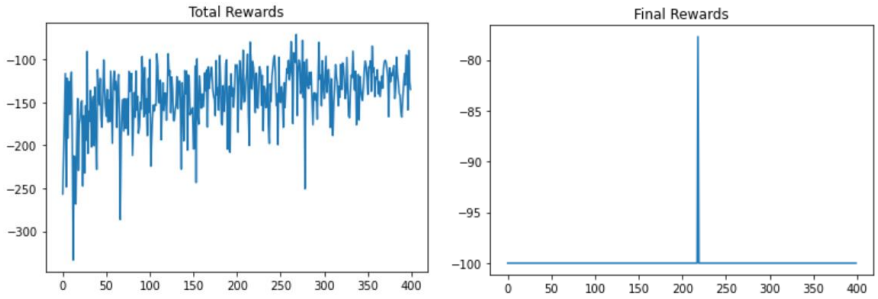
更改超參數前，先了解“超參數”與“參數”的差別，最大的差異為是否由訓練階段學習，參數是訓練模型時學習出的，譬如權重 w 與偏差值 b ；超參數則是則是由人為給定，例如神經網路的層數、損失函數、卷積核的大小、學習率等等。以下的改進方法由更改超參數，並且給予以下四種方法分別講述，我們的控制組為第一題的助教 sample code，比較不同操作變因的 total_reward 差異。

- 改變神經網路大小

  在這裡我的操作變因是 NN 的架構，給出新的架構如下圖所示。更改內容將層數多加一層，並且 in_features 與 out_features 的數量有更動。

```
PolicyGradientNetwork(
  (fc1): Linear(in_features=8, out_features=16, bias=True)
  (fc2): Linear(in_features=16, out_features=32, bias=True)
  (fc3): Linear(in_features=32, out_features=16, bias=True)
  (fc4): Linear(in_features=16, out_features=4, bias=True)
)
```

下圖給出了我們 training 的結果，可以看到 Total Rewards （左圖）與對照組比起來表現較差，趨勢上有逐漸愈來愈好，但是上升速度較慢；而在 Final Rewards （右圖） 上，只有一次獲得較好分數。

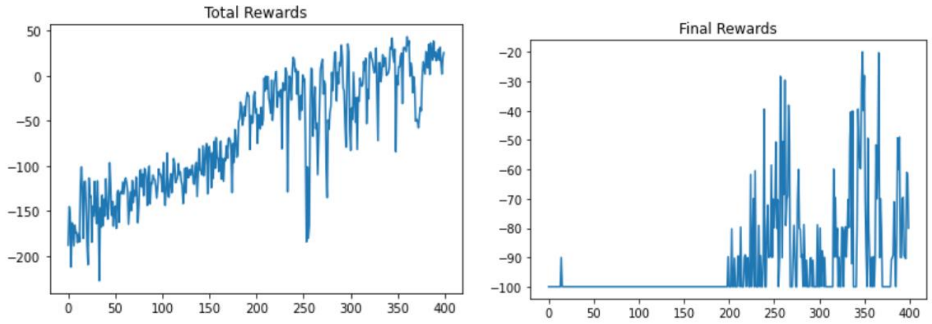


此次 test 的 total_reward 如下圖所示，可見改變這個架構得到的成效比較差，從 training 的過程就可以看出來。

```
print(total_reward)
-154.96778384228887
```

- 改變一個 batch 中的回合數（EPISODE_PER_BATCH）

原本的 EPISODE_PER_BATCH = 5，此處我改成 10 看看結果比較。可以看到 Total Rewards （左圖） 從負的變成正的；而在 Final Rewards （右圖），最後的分數不再都是 -100，產生有高有低的狀況。總體而言，可以看到 batch = 10 得到的效果比較好，因為每一次訓練的資料比較多的關係。

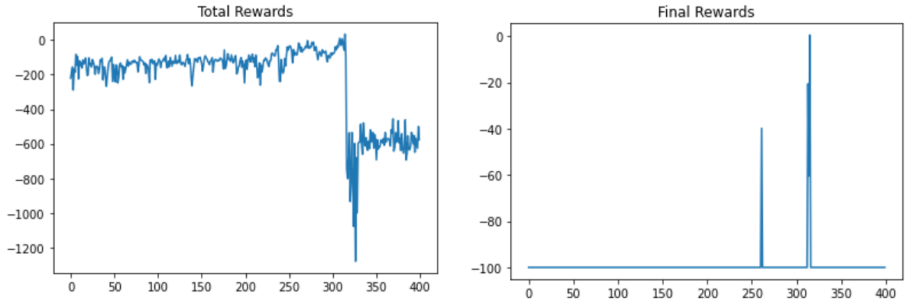


此次 test 的 total_reward 如下圖所示，居然飆漲到了 98.97 的分數，比在 training process 任何時刻都高很多，可見機器在玩的時候可以有很厲害的爆發性發展。

```
[ ]  print(total_reward)

     98.9712692584564
```

- 更改 SGD learning rate

  原本的 SGD learning rate = 0.001，此處我改成 0.005 看看結果比較。可以看到 Total Rewards （左圖） 比之前的不穩定，但是有好有壞；而在 Final Rewards （右圖）也是有好有壞狀況。個人認為把 lr 調大很像是賭一把的概念，有時機器可以變成很強，但是也會不小心離開區域最佳值。因此記得要記錄每次的 total_reward，如果當下的 total_reward 比最好的 total_reward 好的話，我們存下此模型參數，可以看到在 Total Rewards 圖中，並不是訓練愈久表現愈好。我自己也有另外將 lr 調成 0.05，但是因為 lr 太大造成模型爛掉，就不多探討。

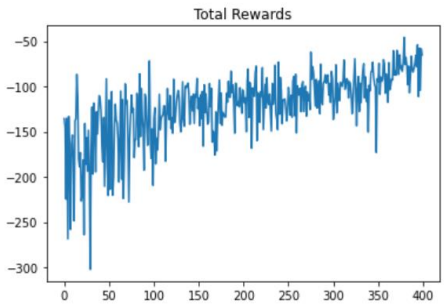


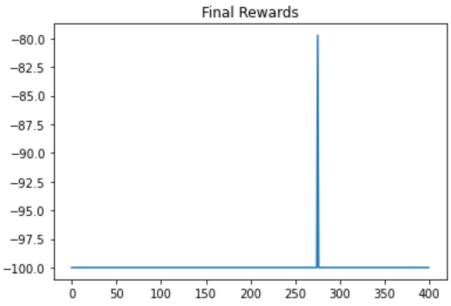
此次 test 的 total_reward 如下圖所示。

```
[9]  print(total_reward)

     37.45724566858735
```

- 更改 optimizers 為 Adam

  原本的 optimizer = SGD with lr (0.001)，此處我改成 optimizer = Adam with lr (0.001) 看看結果比較。可以看到 Total Rewards （左圖） 與之前差不多，皆是逐漸上升且在 -50 ~ -100 徘徊；而在 Final Rewards （右圖），只有一處未獲得 -100。





此次 test 的 total_reward 如下圖所示。

```
print(total_reward)
-78.34021793733965
```

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。**

b. **請說明你的實做與前者（Policy Gradient）的差異。**

ANS：

a. 這邊我使用了 Advantage Actor-Critic (A2C) 來實作，以下我將逐步介紹我實作的方法以及改善。

Step_1. 建構 actor-critic 與 optimizer：

下圖第一張給出了我的 actor-critic 架構，兩者結構差不多，不一樣的在於 activation function 與最後的 output layer 數量。Actor 有四個是因為最後給出四個動作，並且最後通過 softmax 產生類似機率分布的輸出；而 Critic 給出了我們的 state_value，是一個數值。

下圖第二張給出了我們的 actor-critic optimizer，我一開始實作採用 SGD with lr = 0.001, momentum = 0.9，不過經過實作後發現會有些缺陷，因此我之後改成 Adam with lr = 0.005，詳細內容會在後面講述。

```
Actor(
  (actor): Sequential(
    (0): Linear(in_features=8, out_features=16, bias=True)
    (1): Tanh()
    (2): Linear(in_features=16, out_features=16, bias=True)
    (3): Tanh()
    (4): Linear(in_features=16, out_features=4, bias=True)
    (5): Softmax(dim=None)
  )
)
Critic(
  (critic): Sequential(
    (0): Linear(in_features=8, out_features=16, bias=True)
    (1): ReLU()
    (2): Linear(in_features=16, out_features=16, bias=True)
    (3): ReLU()
    (4): Linear(in_features=16, out_features=1, bias=True)
  )
)
```

```
SGD (
Parameter Group 0
    dampening: 0
    lr: 0.001
    momentum: 0.9
    nesterov: False
    weight_decay: 0
)
SGD (
Parameter Group 0
    dampening: 0
    lr: 0.001
    momentum: 0.9
    nesterov: False
    weight_decay: 0
)
```

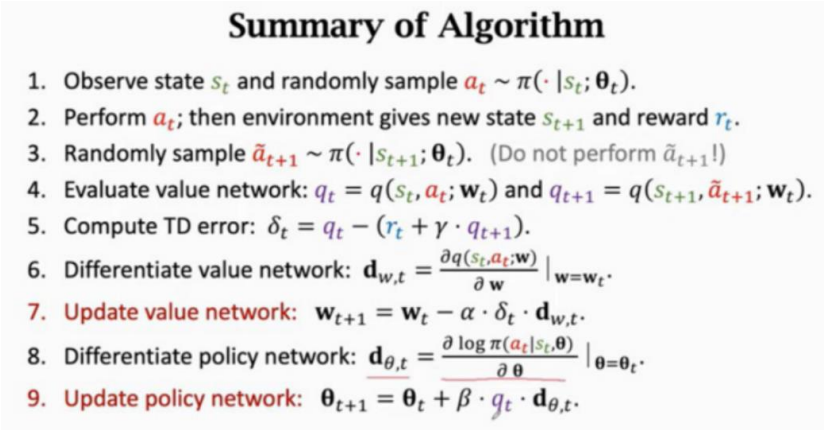
Step_2. 定義超參數：

我給出了我超參數如下。其中較特別的是 best，是決定是否要存 model 的 threshold，當 avg_total_reward 大於 best 我們會存下來，並且將 best 更新成此 avg_total_reward，因此最後存下的 model 有著訓練過程中最大的 avg_total_reward；而 max_steps 代表每次 episode 不超過 200 steps，以免花許多訓練時間。

```
NUM_BATCH = 300         # 訓練 300 次
EPISODE_PER_BATCH = 10  # batch = 10
gamma = 0.99            # 設定 gamma (discount factor)
best = -100000          # save model initial threshold
max_steps = 200         # step max threshold
```

Step_3. 訓練 Actor-Critic：

下圖簡單給出了我們的演算法，主要就是先讓 actor 在狀態 s 產生行為 a，並且更新到下一個狀態 s’ ，此時 critic 針對 s、s’ 給出兩個 state_value，藉由兩個 state_value 與 reward 產生 TD error，最後再使用此 loss 更新 actor-critic 參數。

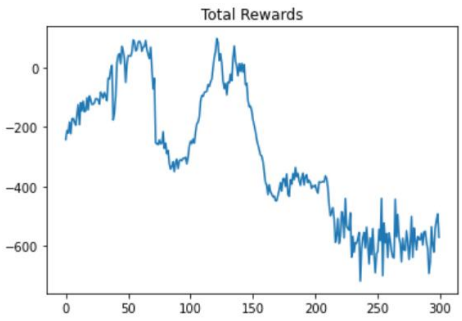


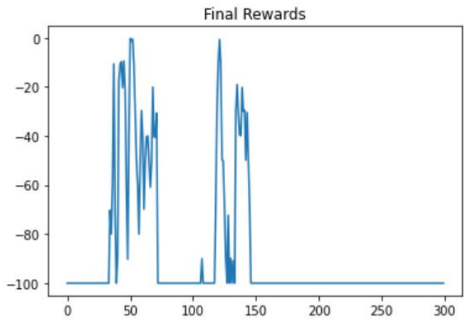
下圖的 code 定義了我們的 advantage function，也就是 TD_error。我發現 advantage_function 也需要經過正規化才可讓模型訓練起來，正規化方式與助教的 sample code 相同。最後，更新 actor-critic 的算法就如同上述演算法。

```
# target values are calculated backward
# it's super important to handle correctly done states,
# for those cases we want our to target to be equal to the reward only
q_val = state_value_final

# 用最後的 q_val 推到前面方法
for i in range(total_step-1, -1, -1):
    q_val = rewards[i] + gamma*q_val*(1.0-dones[i])
    q_vals[i] = q_val # store values from the end to the beginning

advantage = torch.Tensor(q_vals) - values
advantages.append(advantage.detach().numpy())
break
```

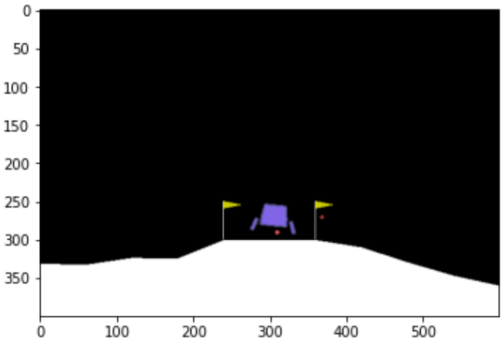
Step_4. 觀察 A2C-SGD 結果：

首先，我儲存的 model 在訓練中有 avg_total_reward = 156.5 的成績，然而從下圖的趨勢圖可以觀察到在 epoch = 0~100 與 100~125 是有訓練起來的區間，愈往後面反而跳出區域最佳值，使模型訓練到爛掉，會有這樣的原因是因為 SGD 的 lr 並不會隨著 epoch 變小，並且又有 momentum 的加持，使得很容易離開我們要的區域值。而在 avg_final_reward 上，一樣也是前半段有比較好的成績。





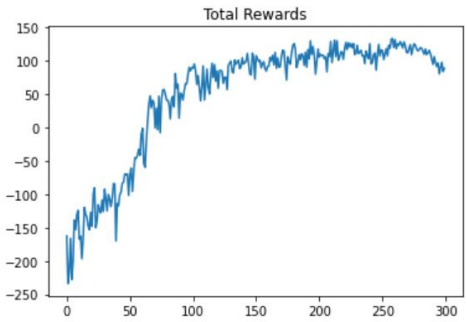
最後拿到我們的 test，可以看到成績如下圖。

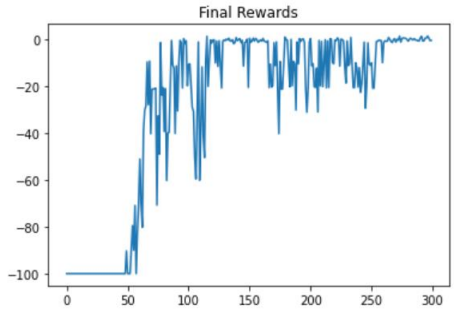



```
[15] print(total_reward)

     107.1376849692344
```

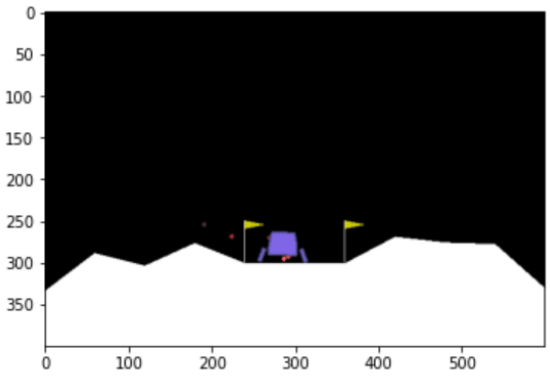
Step_5. 觀察 A2C-Adam 結果：

由於前面訓練的過程中十分不穩定，因此我將 optimizer 改成 Adam with lr = 0.005，可以使 lr 愈來愈小，因此當我們找到比較好的區域最佳值時不會不小心離開它。沒想到整個 avg_total_reward 表現十分穩定，隨著 epoch 上升，reward 也隨之增加，可見在訓練後期 lr 真的不能太大。而在 avg_final_reward 上，從 epoch = 50 開始後已經脫離 -100 了，表現十分亮眼。





最後拿去 test，有著下圖的成績。可以發現其實使用 SGD 與 Adam 在最後的成績是差不多的，只是 Adam 的訓練過程比較穩定。




```
[12] print(total_reward)

     108.11029979599292
```

b. 經過實作後發現，我認為確實使用 A2C 方法比 Policy Gradient 還要有效，相同超參數，avg_total_reward 還是 A2C 獲勝。此外，我發現 A2C 訓練速度快，在前半部就可以將模型訓練好，使用 Adam 將 lr 遞減可以使模型的 avg_total_reward 逼近最大化，同時保持模型的訓練穩定度。相較若使用 SGD 很容易造成模型訓練不穩定，這方面比 Policy Gradient 使用大 lr 還要嚴重。

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。

ANS：

我們針對下列四種方法探討他們的差異與優缺點，數據與作圖的部分都在前幾題有顯示出來：

- 範例程式的做法

  這裡算是我們的 baseline method，因為是評斷基準我就不多贅述，從圖上可以看到隨著 epoch 數增加，model 趨勢上有訓練愈來愈好。

  優點的話就是簡單實作；缺點的話就是出來的結果是負數，但是我們希望結果要愈大愈好，因此成效上仍有不足。

- Policy Gradient (REINFORCE algorithm) 方法

  在這邊我們增加了 tip_1 與 tip_2 method，與 sample code 最大的差異是如果將 baseline 使用 Mean 的話，模型會訓練不起來，而且每個 epoch 的 total_reward 差比較大。改用定值 baseline 可以使模型最後有比較好的效果。以上述做法為基底加入了 momentum 後，可以看到模型訓練效果更好，但是很容易就離開區域最佳值就掉進另一個比較差的區域最佳值，因此在這裡我了解了訓練次數並不代表就是最好，不像是 sample code 會慢慢穩定到一個區域最佳值。

  優點的話就是表現結果較好（就算是沒有使用 momentum)；缺點的話就是要如何正確選擇 baseline 還需要經驗法則，每個 case 都會有他們適合的 baseline，此外計算量也會比較大。

- 超參數選擇造成的差異

  因為改變模型參數與改變 optimizer 的結果沒有與 sample code 有很大差異，因此這裡主要探討改變 batch 與 learning rate。

  首先我們探討 batch，顯著發現調大 batch 會影響到表現結果，而且從訓練圖上可以看到不太容易從很好的 tota_reward 掉到很差的（比較穩定)，而且在 final_reward 上也可以創造出比較好的成績，然而該如何選到合適的 batch 仍然要多實驗幾次。

  改 batch 的優點就是表現結果較好，訓練過程穩定；缺點的話就是要如何正確選擇 batch 也需要經驗法則，運算過程較久。

  接著我們探討 learning rate，發現調大 learning rate 會影響到表現結果，但是曲線很不穩定。

  改 learning rate 的優點就是表現結果較好，因為大 lr 可以使學習速度較快；缺點的話就是訓練過程不穩定，要記得存最好的模型參數，不然有可能不小心從好的值掉進不好的 local maximum。

- Actor-Critic （擇一實作） 方法 (optional)

  A2C with Adam 的優點就是表現結果較好、模型訓練穩定；缺點的話就是訓練過程若不定義 max_steps threshold 會花很多時間訓練，因此表現與時間花費需要權衡。