**HW2\_Policy\_Gradient Report**

Problem 1 .

The trick part is about the good initial parameter setting. The safe way is to initialize the variables with some random value like

**W1 = tf.Variable(tf.truncated\_normal(shape=[in\_dim, hidden\_dim], stddev=1.0))**

**b1 = tf.Variable(tf.zeros([hidden\_dim]))**

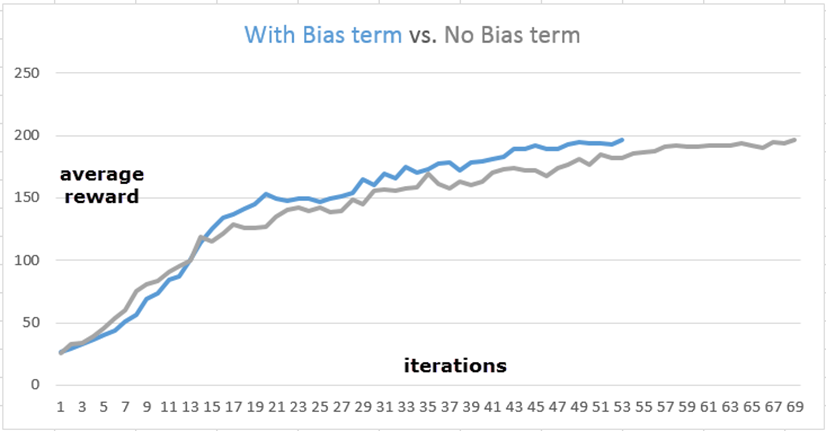
**W2 = tf.Variable(tf.truncated\_normal(shape=[hidden\_dim, out\_dim], stddev=1.0))**

**b2 = tf.Variable(tf.zeros([out\_dim]))**

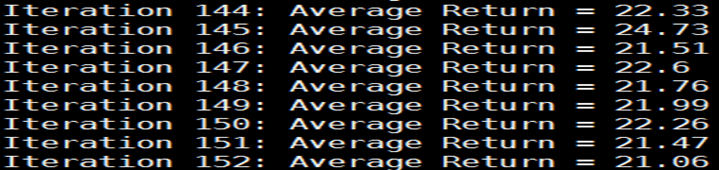
If variable is initialized with some random value, the bias term seems not so necessary for this RL training with two simple layer.

No Bias-term will need more iterations while still converge.

Here is the converge comparison between With Bias-term and No Bias-term



If variable is initialized with all zero value, then the iterations will become very long without any sign of converge.



The reason for the bad convergence is that the all zeros initial variable will let all path for episodes of one iterations just produce [0.5, 0.5] action probability. The training of this iteration will teach our policy network NOTHING and keep producing equal [0.5, 0.5] action probability. That’s the infinite bad loop.

So it’s quite important to initialize the variable with some value to let the policy learning something from beginning.

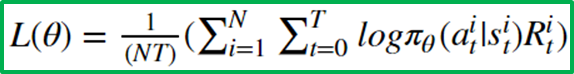
Another solution to reduce the iteration times, we use another method to initialize the weight

使用tf.random\_uniform(shape, -r, r), r = sqrt(6./(fan-in+fan-out))來初始化第一層的權重,使用tf.random\_uniform(shape, -r, r), r = 4 \* sqrt(6./(fan-in+fan-out))來初始化第二層的權重,使用tf.zeros(shape)來初始化兩層的bias,如此可使訓練到solved速度加快，五次實驗中有三次只要15個iterations，一次14個iterations，一次16個iterations，平均15個iterations。

Problem 2.

When dealing with surrogate loss of problem2, an interesting behavior is observed.

Basically, the strict definition of surrogate loss is defined as



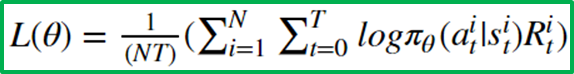
Let us call this formula as Strict Surrogate Loss

The corresponding code for above formula can be implemented as below

total\_time\_steps = tf.cast(tf.shape(self.\_advantages), tf.float32)

surr\_loss = -1 \* tf.reduce\_sum(tf.mul(log\_prob, self.\_advantages)) / total\_time\_steps

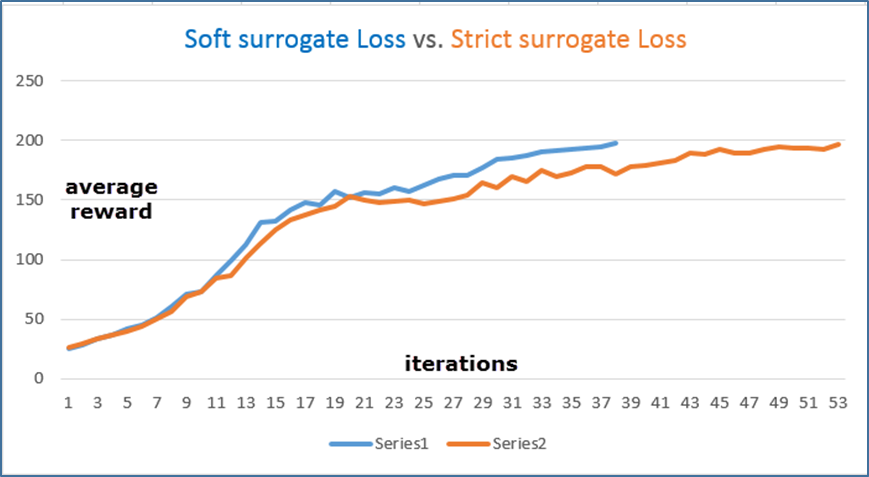
We want to understand what will happen if we chose another function to approximate above formula. The simplest way of one approximation is just remove the total time step term



Let us call this approximation as Soft Surrogate Loss

Interestingly, we found the policy network converged with less iteration numbers

This plot is confirmed with several repeat testing



Here is our conjecture about why this approximation helps for coverage:

The removal of 1/NT is similar to reweighting episode on their time scale. Mathematically, the reweighting will encourage the episode with long time play while ignore the episode with short time play.

Conceptually, that’s the reason we get better coverage at less iterations.

We try two methods reduce\_sum (the total loss) and reduce\_mean (the average loss)in our code.

Problem 3.

使用公式 = ,用一個for迴圈去寫,這邊設一個變數在迴圈中可以一直乘以discount\_rate再和reward相乘和前一個的結果相加.

Problem 4.

使用公式 A(s,a)=, 得到 a= r-b

Problem 5.

原先在觀察有無baseline的收斂速度情況時，兩者似乎沒有太大差別，收集各五次實驗至solved的平均回饋值(average\_returns)，計算其平均值與變異數，結果如下表格：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| baseline | with baseline | | | without baseline | | |
|  | iterations | mean | variance | iterations | mean | variance |
| exp1 | 15 | 99.60 | 4372.64 | 15 | 99.93 | 4338.30 |
| exp2 | 14 | 93.56 | 4068.66 | 15 | 97.09 | 4251.36 |
| exp3 | 15 | 99.74 | 4372.42 | 16 | 103.61 | 4657.96 |
| exp4 | 15 | 98.87 | 4332.32 | 14 | 92.68 | 4119.95 |
| exp5 | 15 | 91.62 | 3843.19 | 15 | 99.79 | 4520.27 |
| average | 14.8 | 96.678 | 4197.846 | 15 | 98.62 | 4377.568 |

然而，將iterations次數設為50、200次，即即使solved亦不停止，將有無baseline做各三次實驗結果收集，得到12組平均回饋值資料，設定：

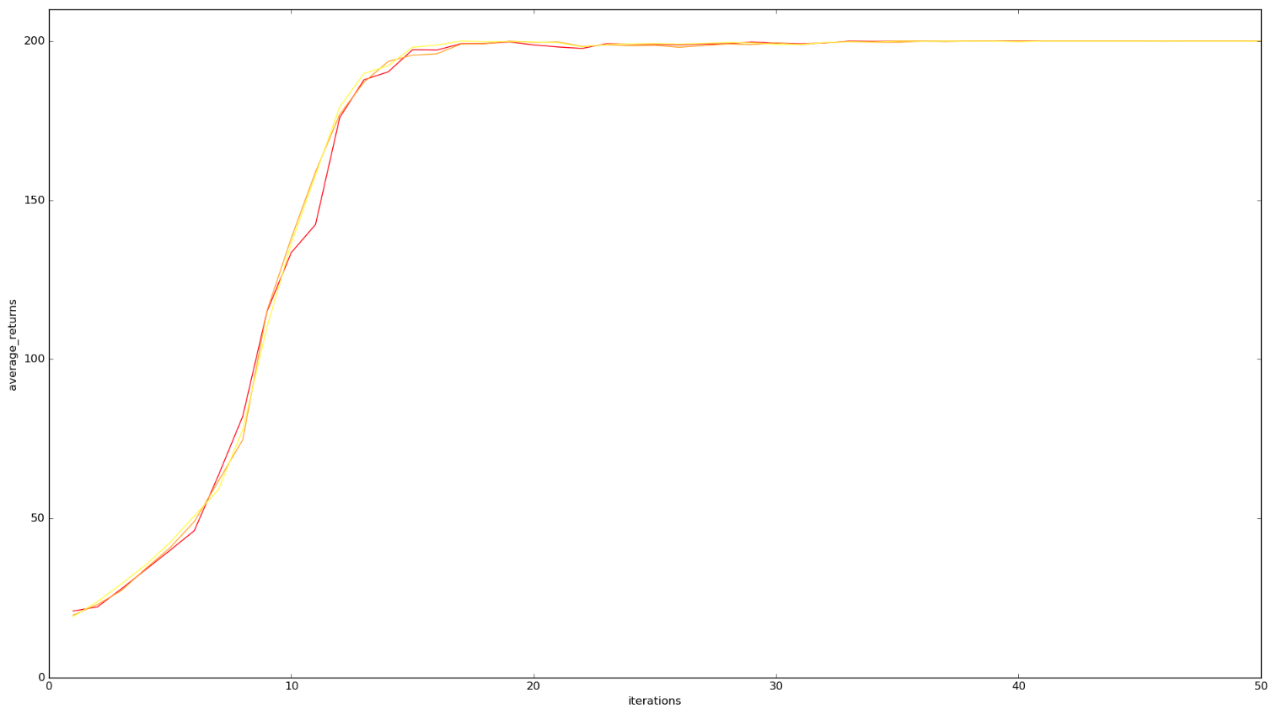
x軸為iterations次數，

y軸為在該iterations的average\_returns，

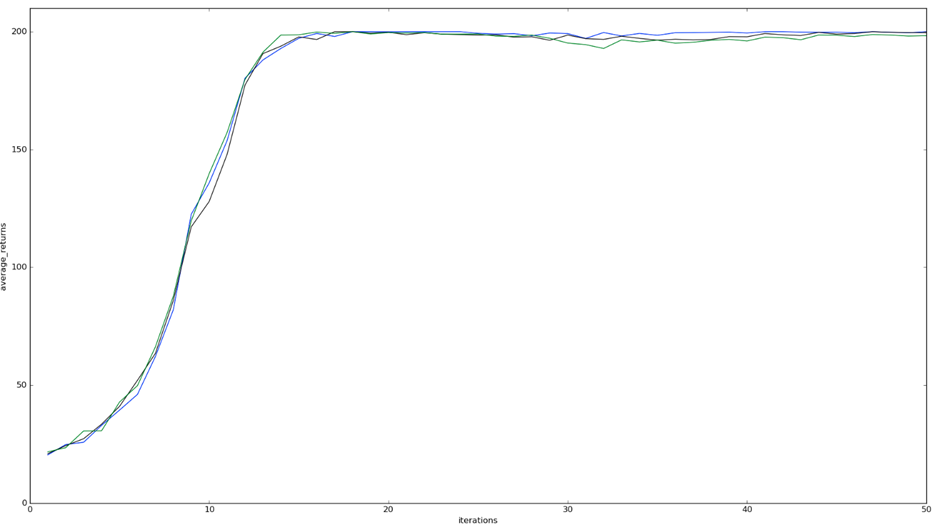
有baseline的三次實驗以紅、橘、黃色線表示，

沒有baseline的實驗以藍、黑、綠色線表示。

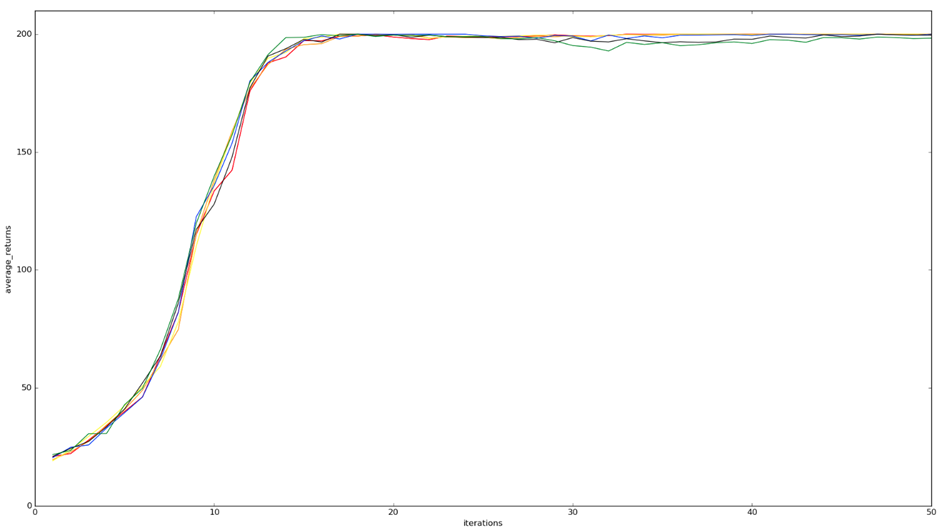
50次iterations，有baseline



50次iterations，沒有baseline

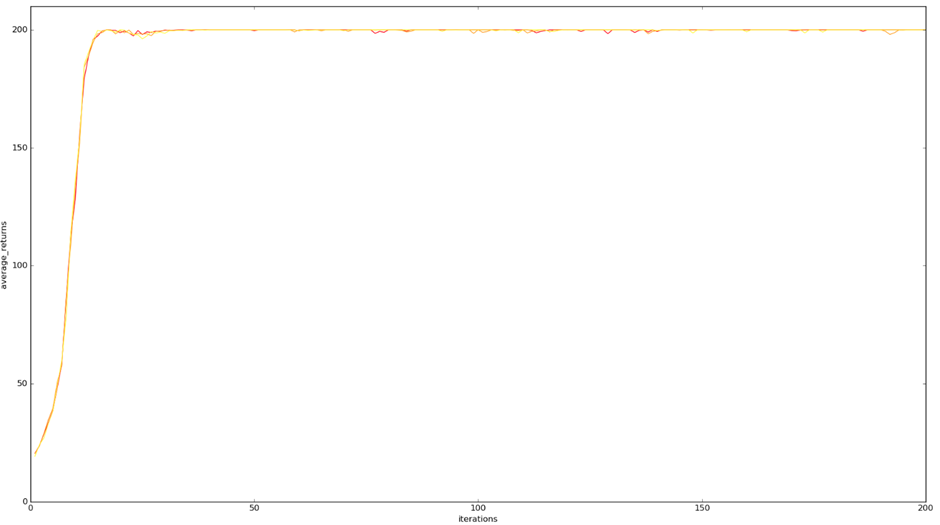


50次iterations，有無baseline兩種情況畫在一起

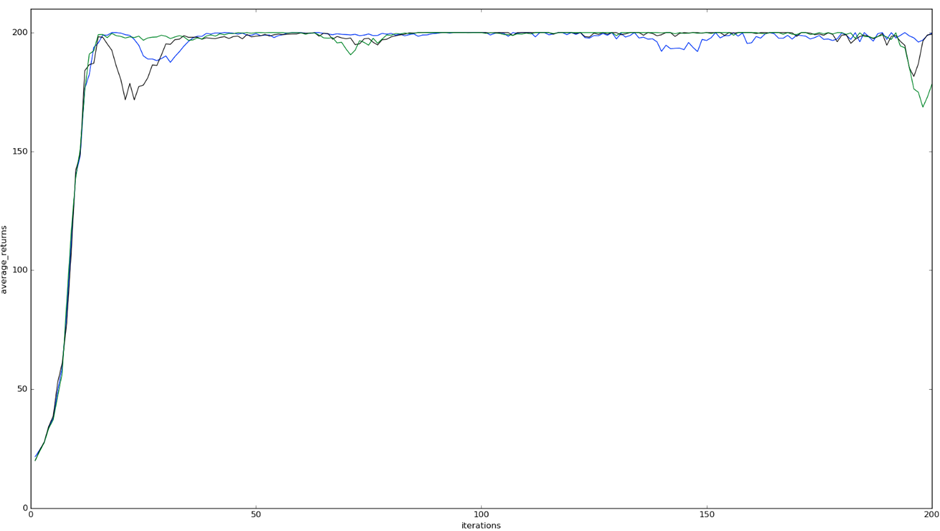


可以觀察出在大約30次iterations時，沒有baseline的線段，相較於有baseline的線段，有向下掉的情況，但又回升。以下為再各作三次200次iterations實驗的結果。

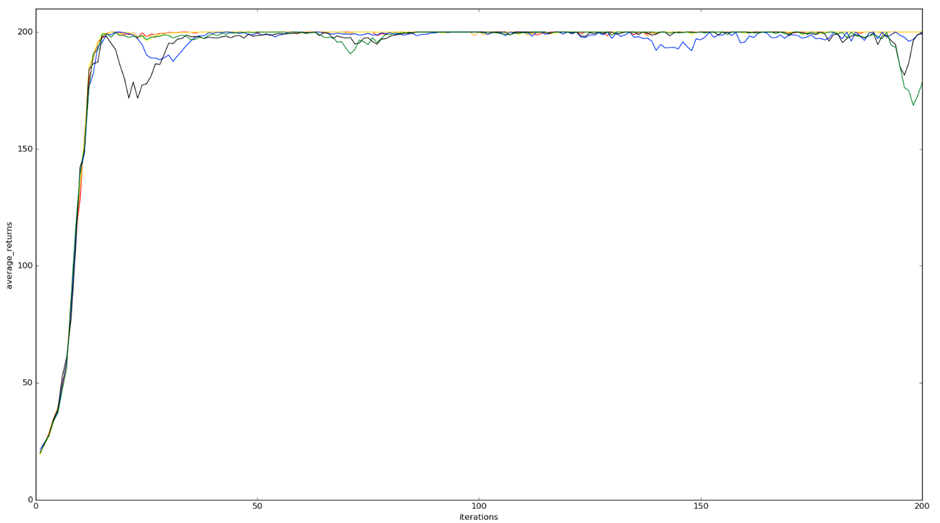
200次iterations，有baseline



200次iterations，沒有baseline



200次iterations，有無baseline兩種情況畫在一起



可以觀察出，在沒有baseline的情況，平均回饋值有波動的傾向（大約於iterations 25, 75, 150, 200），而有baseline的情況則一直保持平穩。

就此實驗結果而言，有無使用baseline，對solved速度並無明顯的影響，然而對於保持平均回饋值的穩定（solved後的變異數較小），使用baseline則有明顯的幫助。

也許，在沒有使用baseline的情況，若在solved之前，平均回饋值發生向下波動，就會影響訓練到solved的時間，然而在這次實驗中並無觀察出如此情況。

Problem 6.

列出下列理由：

1. 做標準化後，可想像為促進大約一半的actions、亦抵制大約一半的actions。

2. 我們使用neural network為線性，對advantage做標準化可以較佳結果。

3. 數學上可將此解釋為控制policy gradient estimator的變異數。

4. 因為神經網路會經過linear跟non-linear,而linear的部份若值過於大(沒有normalize),會造成non-linear時的變化不明顯,舉例,若non-linear的function為Relu, ,若沒有normalize則x的值過大,指數趨近於零,則non-linear結果趨近於1.