**Problem1**

For problem1, 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]))**

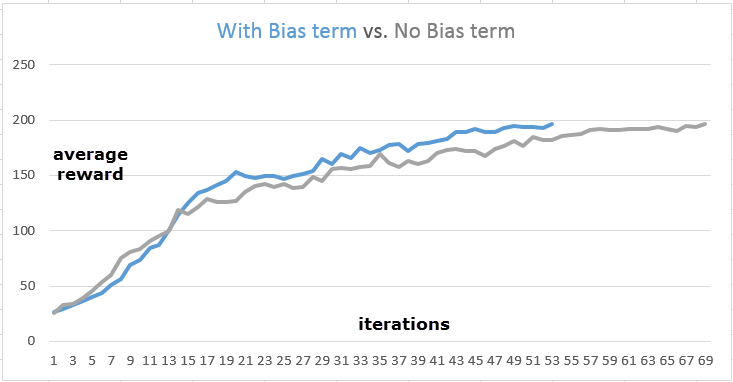
**W2 = tf.Variable(tf.truncated\_normal(shape=[hidden\_dim, out\_dim], stddev=1.0))**

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

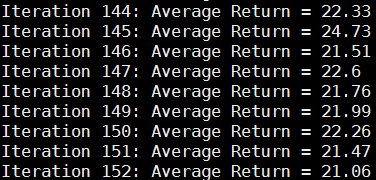
If variable is initialized with some random value, the bias term seems not so necessary for this RL training with two simple layer.

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.

**Problem2**

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

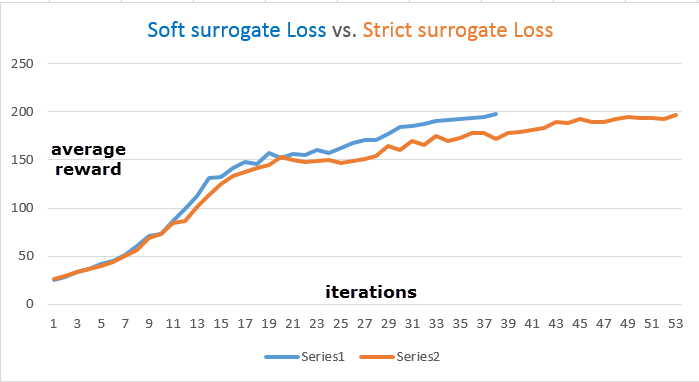
We want to understand what will happen if we chose another function to approximate above formula. The simplest way of one approximation is just remove the total time step term



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