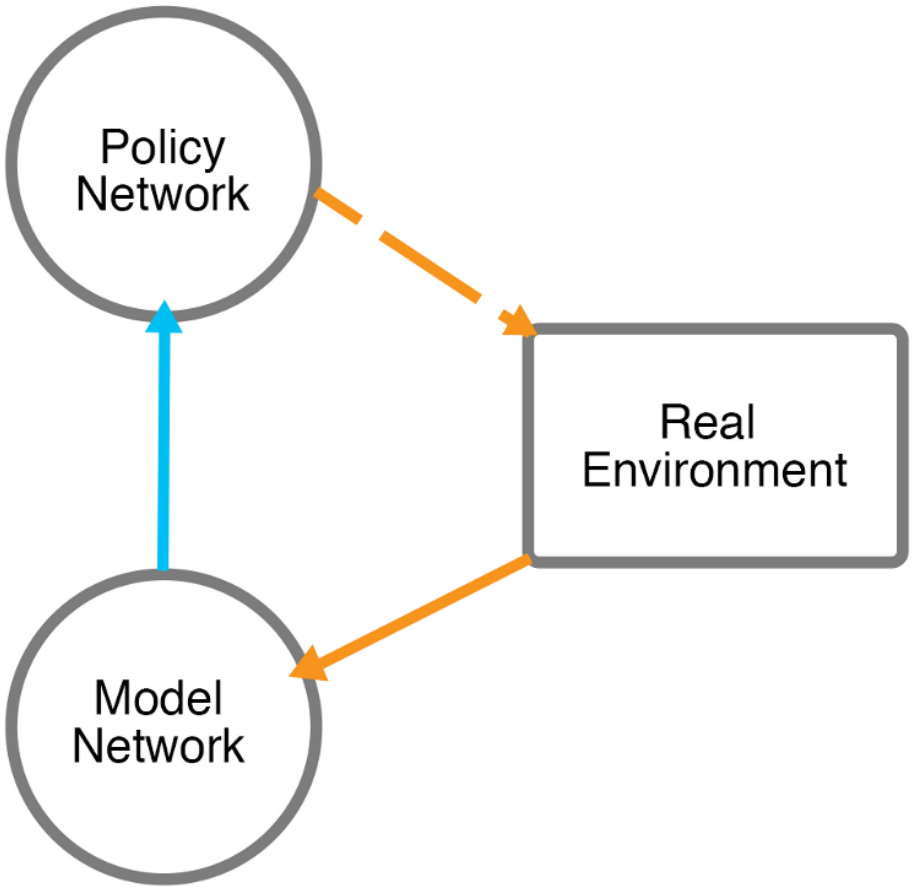
Simple Reinforcement Learning with Tensorflow: Part 3 – Model-Based RL

It has been a while since my last post in this series, where I showed how to design a policy-gradient reinforcement agent that could solve the CartPole task. In this tutorial, I would like to re-examine the CartPole problem, but this time introduce the concept of a model of the environment that the agent can use to improve it's performance.



(If you haven't read them already, here are links to the first and second tutorials in this series. Each tutorial builds on the last, so if you are new to reinforcement learning, I suggest reading through them chronologically.)

What is a model and why would we want to use one? In this case, a model is going to be a neural network that attempts to learn the dynamics of the real environment. For example, in the CartPole we would like a model to be able to predict the next position of the Cart given the previous position and an action. By learning an accurate model, we can train our agent using the model rather than requiring to use the real environment every time. While this may seem less useful when the real environment is itself a simulation, like in our CartPole task, it can have huge advantages when attempting to learn policies for acting in the physical world.

Unlike in computer simulations, physical environments take time to navigate, and the physical rules of the world prevent things like easy environment resets from being feasible. Instead, we can save time and energy by building a model of the environment. With such a model, an agent can 'imagine' what it might be like to move around the real environment, and we can train a policy on this imagined environment in addition to the real one. If we were given a good enough model of an environment, an agent could be trained entirely on that model, and even perform well when placed into a real environment for the first time.

How are we going to accomplish this in Tensorflow? As I mentioned above, we are going to be using a neural network that will learn the transition dynamics between a previous observation and action, and the expected new observation, reward, and done state. Our training procedure will involve switching between training our model using the real environment, and training our agent's policy using the model environment. By using this approach we will be able to learn a policy that allows our agent to solve the CartPole task without actually ever training the policy on the real environment! Read the iPython notebook below for the details on how this is done.

Since there are now two network involved, there are plenty of hyper-parameters to adjust in order to improve performance or efficiency. I encourage you to play with them in order to discover better means of combining the the models. In Part 4 I will be exploring how to utilize convolutional networks to learn representations of more complex environments, such as Atari games.