Sep 28, 2016 · 4 min read

Simple Reinforcement Learning with Tensorflow Part 1.5: Contextual Bandits

(Note: This post is designed as an additional tutorial to act as a bridge between Parts 1 & 2.)

In Part 1 of my Simple RL series, we introduced the field of Reinforcement Learning, and I demonstrated how to build an agent which can solve the multi-armed bandit problem. In that situation, there are no environmental states, and the agent must simply learn to choose which action is best to take. Without a given state state, the best action at any moment is also the best action always. Part 2 establishes the full Reinforcement Learning problem in which there are environmental states, new states depend on previous actions, and rewards can be delayed over time.

There is actually a set of problems in-between the stateless situation and the full RL problem. I want to provide an example of such a problem, and show how to solve it. My hope is that those entirely new to RL can benefit from being introduced to each element of the full

formulation step by step. Specifically, in this post I want to show how to solve problems in which there are states, but they aren't determined by the previous states or actions. Additionally, we won't be considering delayed rewards. All of that comes in Part 2. This simplified way of posing the RL problem is referred to as the Contextual Bandit.

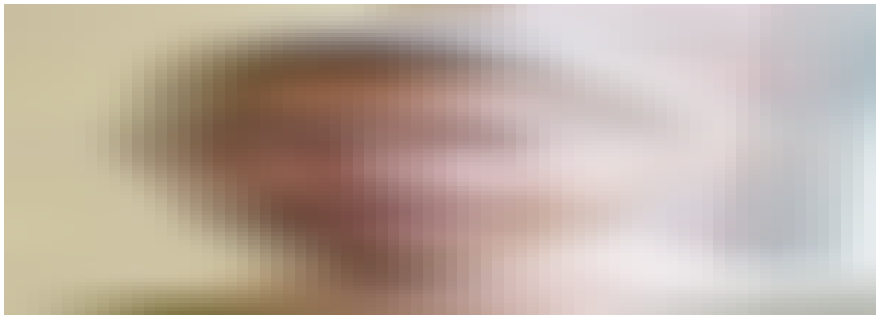
Above: Multi-armed bandit problem, where only action effect reward. Middle: Contextual bandit problem, where state and action effect reward. Bottom: Full RL problem, where action effects state, and rewards may be delayed in time.

Contextual Bandit

In the original multi-armed bandit problem discussed in Part 1, there is only a single bandit, which can be thought of as like a slot-machine. The range of actions available to the agent consist of pulling one of multiple arms of the bandit. By doing so, a reward of +1 or -1 is received at different rates. The problem is considered solved if the agent learns to always choose the arm that most often provides a positive reward. In such a case, we can design an agent that completely ignores the state of the environment, since for all intents and purposes, there is only ever a single, unchanging state.

Contextual Bandits introduce the concept of the *state*. The state consists of a description of the environment that the agent can use to take more informed actions. In our problem, instead of a single bandit, there can now be multiple bandits. The state of the environment tells us which bandit we are dealing with, and the goal of the agent is to learn the best action not just for a single bandit, but for any number of them. Since each bandit will have different reward probabilities for each arm, our agent will need to learn to condition its action on the state of the

environment. Unless it does this, it won't achieve the maximum reward possible over time. In order to accomplish this, we will be building a single-layer neural network in Tensorflow that takes a state and produces an action. By using a policy-gradient update method, we can have the network learn to take actions that maximize its reward. Below is the iPython notebook walking through the tutorial.



Hopefully you've found this tutorial helpful in giving an intuition of how reinforcement learning agents can learn to solve problems of varying complexity and interactivity. If you've mastered this problem, you are ready to explore the full problem where time and actions matter in Part 2 and beyond of this series.

. . .

If this post has been valuable to you, please consider *donating* to help support future tutorials, articles, and implementations. Any contribution is greatly appreciated!

More from my Simple Reinforcement Learning with Tensorflow series:

1. *Part 0—Q-Learning Agents*
2. *Part 1—Two-Armed Bandit*
3. **Part 1.5—Contextual Bandits**
4. *Part 2—Policy-Based Agents*
5. *Part 3—Model-Based RL*
6. *Part 4—Deep Q-Networks and Beyond*

7. *Part 5—Visualizing an Agent's Thoughts and Actions*
8. *Part 6—Partial Observability and Deep Recurrent Q-Networks*
9. *Part 7—Action-Selection Strategies for Exploration*
10. *Part 8—Asynchronous Actor-Critic Agents (A3C)*

. . .

If you'd like to follow my work on Deep Learning, AI, and Cognitive Science, follow me on Medium @Arthur Juliani, or on twitter @awjliani.