Practical 4: Reinforcement Learning

Writeup and code due 23:59 on Wednesday 6 May 2015

There is not a Kaggle component to this practical. However, your main deliverables are still largely the same: a 3-4 page PDF writeup that explains how you tackled the problems. There are two other differences with the previous practical: you are **not** allowed to use off-the-shelf planning/RL software such as PyBrain, and you should turn in all of your code as a zip file or gzipped tar file, with a README that explains how to run it.

As before, you will do this assignment in groups of three. You can seek partners via Piazza. Course staff can also help you find partners. Submit one writeup per team by the due date via Canvas.

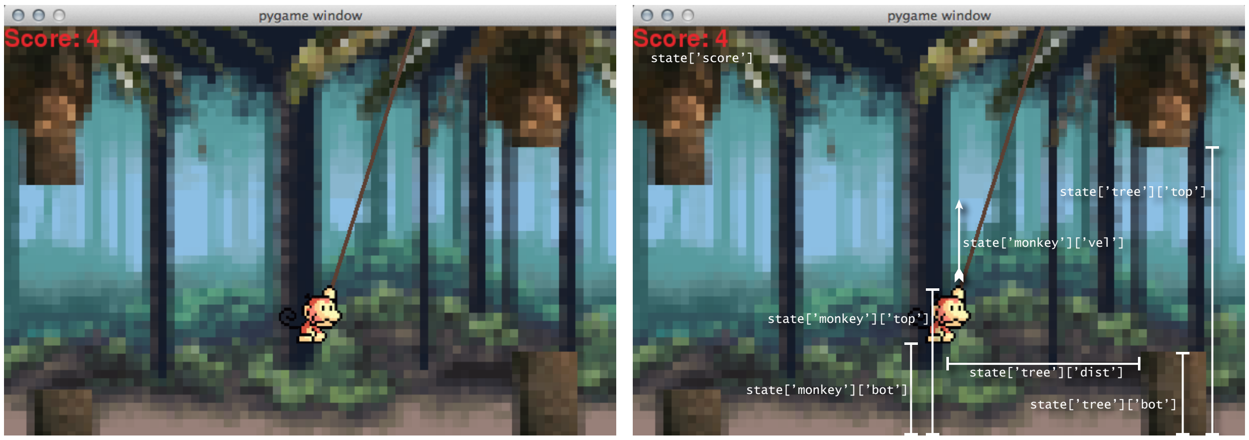
Last year, the mobile game *Flappy Bird* took the world by storm. After its discontinuation, iPhones with the game installed sold for thousands of dollars on eBay. In this practical, you'll be developing a reinforcement learning agent to play a similar game, *Swingy Monkey*. See screenshot in Figure 1a. In this game, you control a monkey that is trying to swing on vines and avoid tree trunks. You can either make him jump to a new vine, or have him swing down on the vine he's currently holding. You get points for successfully passing tree trunks without hitting them, falling off the bottom of the screen, or jumping off the top. There are some sources of randomness: the monkey's jumps are sometimes higher than others, the gaps in the trees vary vertically, and the distances between the trees are different. You can play the game directly by pushing a key on the keyboard to make the monkey jump. However, your objective in the practical will be to build an agent that *learns* to play on its own.

You'll be responsible for implementing two Python functions, `action_callback` and `reward_callback`. The reward callback will tell you what your reward was in the immediately previous time step:

- Reward of $+1$ for passing a tree trunk.
- Reward of -5 for hitting a tree trunk.
- Reward of -10 for falling off the bottom of the screen.
- Reward of -10 for jumping off the top of the screen.
- Reward of 0 otherwise.

The action callback will take in a dictionary that describes the current state of the game and you will use it to return an action in the next time step. This will be a binary action, where 0 means to swing downward and 1 means to jump up. The dictionary you get for the state looks like this:

```
{ 'score': <current score>,
  'tree': { 'dist': <pixels to next tree trunk>,
           'top': <height of top of tree trunk gap>,
           'bot': <height of bottom of tree trunk gap> },
```



(a) SwingyMonkey Screenshot　　(b) SwingyMonkey State

Figure 1: (a) Screenshot of the Swingy Monkey game. (b) Interpretations of various pieces of the state dictionary.

```
 'monkey': { 'vel': <current monkey y-axis speed>,
             'top': <height of top of monkey>,
             'bot': <height of bottom of monkey> }}
```

All of the units here (except score) will be in screen pixels. Figure 1b shows these graphically. There are multiple challenges here. First, the state space is very large – effectively continuous. You'll need to figure out how to handle this. One strategy might be to use some kind of function approximation, such as a neural network, to represent the value function or the Q-function. Another strategy – one that worked well for me – is to discretize the position space into bins. Second, you don't know the dynamics, so you'll need to use a reinforcement learning approach, rather than a standard MDP solving approach. I got a pretty good monkey policy with Q-Learning.

Your task is to use reinforcement learning to find a policy for the monkey that can navigate the trees. The implementation of the game itself is in file `SwingyMonkey.py`, along with a few files in the `res/` directory. A file called `stub.py` is provided to give you an idea of how you might go about setting up a learner that interacts with the game. I also posted a YouTube video of my Q-Learner at http://youtu.be/l4QjPr1uCac. You can see that it figures out a reasonable policy in a few dozen iterations. You should explain how you decided to solve the problem, what decisions you made, and what issues you encountered along the way. As in the other practicals, provide evidence where necessary to explain your decisions. You should not use off-the-shelf RL tools like PyBrain to solve this. Turn in your code.

Questions and Answers

What should I turn in via Canvas? The main deliverable of this practical is a three-to-four page typewritten document in PDF format that describes the work you did. You will also turn in the code that you write, as a zip file or a gzipped tar file. Concretely, you should turn in:

- A 3-4 page PDF writeup that shows your results and explains your approach to Swingy Monkey. This may include figures, tables, math, references, or whatever else is necessary for you to communicate to us how you worked through the problem. The page limit and format is not strict; use common sense.
- A zipped or gzipped folder containing your warm-up code and a README file explaining how to run it.

How will my work be assessed? This practical is intended to be a realistic representation of what it is like to tackle a problem in the real world with machine learning. As such, there is no single correct answer and you will be expected to think critically about how to solve it, execute and iterate your approach, and describe your solution. The upshot of this open-endedness is that you will have a lot of flexibility in how you tackle the problem. You can focus on methods that we discuss in class, or you can use this as an opportunity to learn about approaches for which we do not have time or scope. It is your responsibility to make it clear in your writeup that you did not simply download and run code that you found somewhere online.

You will be assessed on a scale of 20 points, divided evenly into four categories:

1. **Effort:** Did you thoughtfully tackle the problem? Did you iterate through methods and ideas to find a solution? Did you explore several methods, perhaps going beyond those we discussed in class? Did you think hard about your approach, or just try random things?
2. **Technical Approach:** Did you make tuning and configuration decisions using quantitative assessment? Did you compare your approach to reasonable baselines? Did you dive deeply into the methods or just try off-the-shelf tools with default settings?
3. **Explanation:** Do you explain not just what you did, but your thought process for your approach? Do you present evidence for your conclusions in the form of figures and tables? Do you provide references to resources your used? Do you clearly explain and label the figures in your report?
4. **Execution:** Did your methods give reasonable performance?

What language should I code in? For this practical, you'll need to use Python, as that's the language the game is written in.

Can I use {scikit-learn | pylearn | torch | shogun | other ML library}? You can use these tools, but not blindly. You are expected to show a deep understanding of the methods we study in the course, and your writeup will be where you demonstrate this. You should implement the MDP and reinforcement learning parts of this practical yourself.

These practicals do not have conceptual questions. How will I get practice for the midterms? We will provide practice problems and solutions in section. You should work through these to help learn the material and prepare for the exams. They will not be a part of your grade.

Can I have an extension? There are no extensions. There are no exceptions, so plan ahead.

Changelog

This format for assignments is somewhat experimental and so we may need to tweak things slightly over time. In order to be transparent about this, a changelog is provided below.

- **v1.0** – 21:45 on 18 April 2015