6CCS3ML1 (Machine Learning)

Practical 08: Reinforcement Learning

(Version 1.1)

1 Overview

This practical explores ideas from Lecture 8 on reinforcement learning. You will write some simple bandit learners for two different problems. One is the test case that was covered in Lecture 8. The other is a simple version of Pacman (the same you used in Coursework 1).

Note that Pacman will again be used for the second coursework, so it is very much in your interests to spend some time working on the Pacman part of the practical this week.

2 Simple bandit problem

This part of the practical is to replicate the experiment on simple bandit learning that I showed you in the lecture. To help you out, there is a skeleton program on KEATS. So, download:

```
bandit-starter.py
```

Here the creation of each bandit problem is carried out in the class `bandit_problem`. Creating an instance of `bandit_problem` generates a problem with a number of actions specified by `num_actions`. Each action has a payoff that is picked from a normal distribution with mean 0 and variance 1 (which means the standard deviation is also 1). The `provide_payoff` method of `bandit_problem` allows the main program to carry out an action by passing an integer between 0 and `num_actions - 1`. The method the returns a payoff.

The code in `bandit-starter.py` selects actions randomly, and remembers which action is played and what the payoff for that action is. It runs `plays` times with a given `bandit_problem` and repeats this experiment `iterations` times. It then computes and plots the average payoff and the number of times the optimal action was selected. (In fact what it does right now is to identify the "optimal action" at random.)

Run the code and see what it generates.

Your job is turn this into code that will generate the plots I showed in the Lecture. To do that you should:

1. Write code that keeps track of the average reward for each action.
2. Given the average reward for each action, implement the ϵ-greedy method for action selection.

   This will complete the main part of the exercise and should give you the "Average reward" graph.
3. Write code that figures out which is the optimal action by looking at the real payoffs for each action (the value that is called `q_star` in the code) and use that to plot the "Percent optimal action" graph.

Note that `starter_bandit.py` is set up for 20 iterations and 10 plays. To get results that look like the ones from the lecture, you will have to run over 1000 plays. The results in the lecture used 2000 iterations and took quite a long time to run. You can get away with many fewer iterations (around 100 works fine for me) and get decent results a lot quicker.

3 Back to Pacman

As you already know, the AI module at UC Berkeley has some really nice resources in Python, including a version of Pacman which we will use to explore reinforcement learning.

You should:

1. Download:

   ```
   reinforcement.zip
   ```

   from KEATS.

   This is a copy of the code from UC Berkeley, but a slightly different set of files from the ones we used earlier in the module. So be sure to download it rather than just using a copy of what you had from before.

2. Save that file to your computer.

3. Unzip the archive.

   This will create a folder `reinforcement`

From the command line you operate this just as before. Switching to the folder `reinforcement` and typing:

```
python pacman.py
```

will open up a window that looks like that in Figure 1 and allows you to play the game using the arrow keys.

4 Pacman bandits

Now you get to use your reinforcement learning skills on Pacman.

You can, as you know, run the Pacman game with software controlling the Pacman character. For example, try:

```
python pacman.py -p LeftTurnAgent
```

This runs the game with Pacman controlled by an instance of the class `LeftTurnAgent` (defined in `pacmanAgents.py`.

You can run Pacman with lots of optional (command line) arguments. For a list, run:

```
python pacman.py -h
```

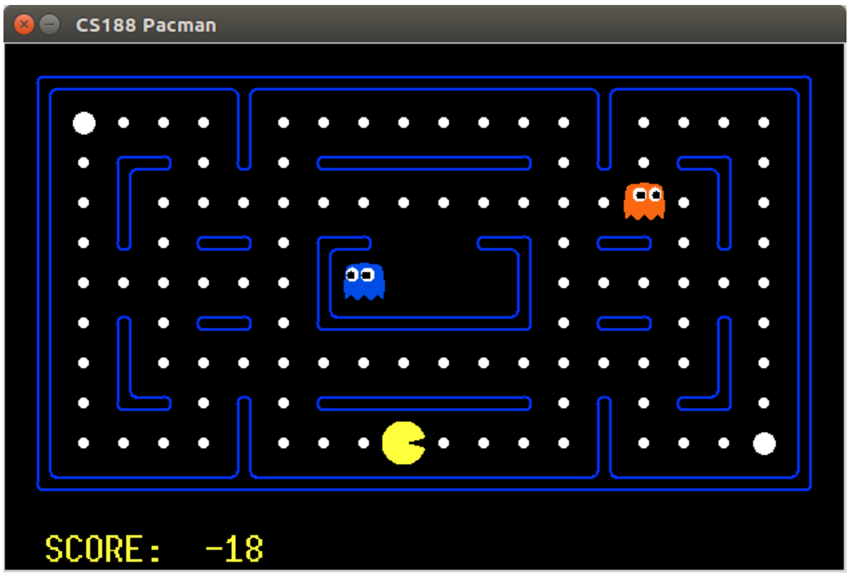

Figure 1: Pacman

Helpful for testing are the ability to run the game multiple times:

```
python pacman.py -p LeftTurnAgent -n 2
```

and to run the game with a smaller board:

```
python pacman.py -p LeftTurnAgent --layout smallClassic
```

or a minimal board:

```
python pacman.py -p LeftTurnAgent --layout smallGrid
```

Now, if you call Pacman with the following:

```
python pacman.py -p SomeAgent
```

it searches through all the Python files in the same directory that have a name that ends in `Agents.py` for the class `SomeAgent`, and then has an instance of that class play Pacman.

Now:

1. Look at the various Pacman agents in the file:

   `pacmanAgents.py`

   and try running them.

2. Download the sample agents that I wrote and run them. They are in the file:

   `sampleAgents.py`

   on KEATS.

   These illustrate a number of the features of the Pacman code, including how to extract the legal moves in a given state, how to make a move, and how to get the current score. They don't show you anything that isn't in `pacmanAgents`, but they are a bit better documented.

3. Take the ϵ-greedy code that you wrote to answer the questions in Section 2 and use it to create a `BanditAgent` class in `sampleAgents.py` which uses an ϵ-greedy strategy to move around the Pacman world while learning the value of each action. It is possible to write `BanditAgent` using only the bits of the Pacman API that are illustrated in `sampleAgents.py` (that is all I used), but feel free to use other elements if you want (and if you can figure them out).[1]

Don't expect your ϵ-greedy agent to work very well. (Mine consistently got stuck in a corner, though not always the same corner.) That is because a simple bandit just learns which action is best, assuming that the actions are always played in the same state. That assumption is not good in Pacman. Rather we need to learn which action to take *in a particular state*. You will see some good methods for doing this in Reinforcement Learning 2, and you will then implement one of these in the coursework.

5 More

If you want to try other things:

1. Now go back to the simple e-greedy learner from Section 2 and make it a softmax learner.
2. Write a new version of the `BanditAgent` for Pacman which uses softmax.

6 Version list

- Version 1.0, March 8th 2020.
- Version 1.1, February 9th 2021.

[1]Note that for this lab and the coursework, you are not restricted to using the code in `api.py` to access the game. If you have been working with Pacman since `6ccs3ain`, feel free to dance around the room — you now have unfettered access to the game.