COSC470 Assignment 3

Due: October 2, 2018
Worth: 25% of the mark for COSC470

September 14, 2018

Tasks

1. Implement in Python using Reinforcement Learning (RL) an agent with convolutional neural network (CNN) policy to play the Frozen Lake game.
2. Write a report showing/explaining the architecture of your policy network, the training method and the results.

The Frozen Lake Game

The Frozen Lake Game is a single player game that takes place on a 4x4 grid.

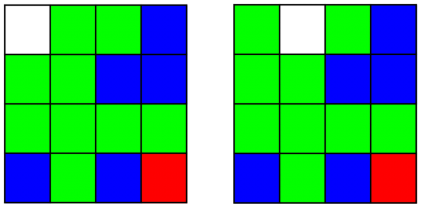

Figure 1: Two consecutive states of the Frozen Lake Game

The rules of the game are as follows:

- the player starts in the top left corner of the grid;
- the player has four actions to choose from - go N, E, S, or W;
- there is a small probability that the player slips sideways on the ice;
- heading in the direction off the board edge when standing on the edge square results in the player bouncing back to (and thus remaining on) that edge square;

- the game terminates when the player reaches the goal square (win) or walks into a water square (loss) or makes a 100 moves without reaching a terminal state (loss);

The configuration of the board changes from game to game such that:

- the goal will be in one of the three corners of the board (excluding the player's starting corner);
- the number of water holes and their locations changes for new game (episode);
- a path over the ice squares between player's starting square and the goal square is guaranteed to exist.

The board on the left of Figure **??** shows the starting state of a possible board configuration of the game. The square with the player is shown in white, the goal square in red, the water hole squares in blue and the ice squares in green. The figure on the right shows the second state, after the player has chosen to move E (and didn't slip sideways).

Starter code

A python implementation of the Frozen Lake Game environment (`frozenlakegame.py`) and a sample agent (`agent.py`) are provided. Sample agent code demonstrates how to use the environment:

- a game is started with a call to the environment's reset method, returning the starting state (and a new random board)
- the state is a 4x4x3 numpy array providing the information about the configuration of the board and player's location on it – it's essentially a 4x4 pixel colour image with information encoded exactly as shown in Figure **??**;
- the agent executes an action by a call to the environment's step method, passing the action as a value between 0 and 3 (corresponding to four directions to head in) – the method returns the resulting state and corresponding reward; the only thing that can (but doesn't have to) change in a given instance of a game is the player's location; the reward for going into a water hole is -1, and the reward for reaching the goal square is +1
- the agent can check if a terminal state has been reached by a call to the environment's terminal method; an environment that has reached a terminal state needs to be reset, so that new game can be played.

The provided environment also comes with a visualisation method that shows the game state – see the comments in `agent.py` for information on other changes you can make to the environment upon instantiation.

Your agent

Make a copy of `agent.py` and name your agent `<your name>_agent.py` – this way, when I am marking, I will be able to place all your agents inside the same folder and using the same environment file. The sample agent plays the game by picking actions at random – you can see it in action by running `agent.py`. Your job is to create a policy using a CNN and train it using RL technique(s).

You are not allowed to use pathfinding algorithms in order to get your agent to the goal state. The objective of this assignment is to use RL and have the agent learn the policy without being "told" the rules of the game.

To hand in

1. Your code that can run in Python 3.6 under cosc470 anaconda environment (the environment is configured on machines in Lab B). If you're working at home, it might be a good idea to create a clone of that environment in order to match the version of Tensorflow, etc. The code should be commented and clean.
2. Your report as a pdf.

Hints:

- If your training takes a long time, it might be a good idea to submit alongside your code a file with saved configuration of your trained policy, which can be loaded into your agent by setting a flag in the script.
- You are encouraged to use Tensorflow for implementation of your CNN.
- Your CNN might not need to be very big and complicated – smaller networks will train faster and the state is represented by a pretty small (4x4 pixel) image that hopefully will not require big models that take forever to train.
- You may (but don't have to) implement an additional model for a value network to be used for training.
- Negative results (as in, submissions with agents that don't learn to play the game very well) can still be given a good mark. In such case, rather than focusing on explaining the details of your final model that works, you can provide a good overview of what you have done, what choices you made about various parts of the RL, and show changes in the results along the way.