Deep Reinforcement Learning

The goal of this assignment is for you to get hands-on experience with a basic version of the combination of reinforcement learning + neural network, which was used in DeepMind's paper on Atari.

Gridwall is a simple game described in the textbook: "Reinforcement Learning: An Introduction". The code and tutorial explaining it are available from http://outlace.com/Reinforcement-Learning-Part-3/. This implementation uses the Q-Learning algorithm and a neural network to find the best policy for the player to reach the goal, avoiding the wall and the pit.

Implementation

You will extend the game to support an interactive mode with two players: the computer (the trained Q-learning and neural network) vs. the user. The computer will control player 1 and the user will control player 2. The user uses the keyboard to select one of the four allowed moves in the game (up, down, left, right).

For the learning algorithm, the only change you will make is introducing the location of player 2 on the grid. You don't have to develop any strategy. The two players can share the same location on the grid. Player 1 does not keep track of whether player 2 has won or not.

Training

Train the learning algorithm using this modified problem definition (with player 2). The algorithm shows a "learned behavior" before 500 epochs. You can use the same number of epochs in the example provided: 1000 epochs, or stop after 700 epochs.

Furthermore, you will play an interactive game against the learning algorithm after each 100 epochs and finally after the training is complete. More on this below.

Your report

- Explain the strategy the learning algorithm seemed to follow in each of the four interactive games (if any).
- Comment on changes in the "strategy" of the learning algorithm across the four interactive games.

- Describe the architecture of the neural network used above (you only need to understand it, not modify it).
- Describe the role of the neural network in this implementation of the Q-learning algorithm.

Notes on Implementation

The neural network in this example is implemented using the Keras library. You can install Keras library via the command: pip install Keras.

A clarification: Unlike assignment 2, this implementation does not use or require SciKit/ Sklearn. However, it does require that you have Numpy installed.

Training time

The bad news

- Training 100 epochs takes around 1 hour, depending on your setup.

The good news

- Keras support storing the weights of a trained model (a neural network in this case) to a file, and later restoring the weights from file.

This means that you do not have to restart the training from scratch every time. Instead, you should extend the code provided to store the weights periodically (After every 100 epochs). To test the interactive mode, you can restore the weights stored after each number of epochs and then run the interactive mode using a particular weights file.

For Example: provided with the assignment is the network model and weights file after 400 epochs of training for the 1 player mode. You can restore this weight file and use it to run the example code* and get a feel of the performance of the learning algorithm in the 1 player mode.

* You can disable the training in this run, as the weights were previously trained.

How to save and restore Keras model and weights:

A good example can be found here:
http://machinelearningmastery.com/save-load-keras-deep-learning-models/
The neural network model (configurations) is saved to a JSON file. The weights are saved in file format H5. Now, if you will be saving the weights multiple times

throughout the training, you will need to save multiple versions of the weights file. Make sure to append a variable index to the file name, otherwise you will be overwriting the file every time. For example: you can append the epoch number to the file name. Also, do not save the weights after each epoch, this will slow the training down.

Bonus

Implement the Q-Learning only version of the problem. In the Q-Learning only version, the neural network part is replaced by a reward table or matrix. An example solution using a reward table for the BlackJack problem is available here:

https://github.com/outlace/outlace.github.io/blob/master/ipython-notebooks/rlpart2.ipynb

Groups

You may choose to work in groups [1], with a maximum of 3 members per group.

Submission

Deadline: 12am on Saturday 17th December. Online submission via drive or email submissions to csen1067@gmail.com

Please Submit one zip file that includes:

1. Python project file
2. Model and weights files
3. Report in pdf format

Please name the zip file using only your 3 id numbers, separated by hyphens. If you implemented the bonus part, add the word bonus to the name of the zip file.

[1] ***Thanks Amal et al. for the suggestion.**