AI Programming (IT-3105) Project Module # 6:
Deep Learning for Game Playing

Purpose:

- Design a game-playing agent using a neural network trained via supervised learning.
- Gain further experience with Theano for configuring and running neural networks.
- Appreciate the pivotal role of representations in neural network applications.

1 Assignment Overview

You will use supervised learning to train an artificial neural network (ANN) to play the game of 2048, based on case data that you generate by either a) playing the game yourself and recording each situation-action pair, or b) using the situation-action pairs generated by your Minimax or Expectimax algorithm earlier in the course. The real challenge in this assignment is devising an effective preprocessor of the board situation (a.k.a. state) that will enable the ANN to learn anything significant. Though your grade is partially based on the quality of play of this ANN agent, the 2048 performance requirement lies well below that of the earlier assignment.

2 Introduction

Recent advances in Deep Learning have not only yielded impressive classifiers for machine-learning benchmarks, such as MNIST, but have also produced talented game-playing agents in a wide range of domains such as Tetris, Checkers, and a whole handful of classic Atari games (see the Materials section of the course web page for reference articles). These successes have involved relatively advanced combinations of AI techniques, somewhat beyond the one-month time scales of IT-3105 course projects.

However, your previous experience with both Theano and the game of 2048 provides an opportune starting point for a brief journey into the exciting world of ANN game playing. Assuming that your code for these earlier modules was written with modularity and reusability in mind, the current module should only require a page or two of code, essentially a bridge between those two earlier modules: the output of one becomes the input of the other.

The devilish detail in all this involves that bridge and how to properly transform your many 2048 cases into useable ANN training data. It may not be as easy as it appears on the surface.

3 Basic Design

The core of your system is an ANN, as sketched in Figure 1, that takes board states as input and produces move choices as output. This should be implemented in Theano (or something of similar power). As in the previous module, you will need to experiment to find the proper topology (e.g. number and size of hidden layers) along with an effective choice of activation function(s), learning rate, momentum, etc.

In this module, however, two factors that were obvious in the previous module have become more of a challenge:

- the data set, and
- the proper preprocessing of that data.

3.1 Data Set Selection

You may choose to use a data set generated by a human player, since that may more consistently exhibit a particular strategy, even if it may have a lower ultimate performance level. That consistency could be essential in training the ANN, which will need to find some patterns in the data. Conversely, the cases generated by Minimax (Expectimax) may have an advanced *logic of their own* that the ANN could eventually learn and exploit to produce 2048 (or greater) tiles with ease. For this module, it is fine to borrow a case set from another group if you are dissatisfied with your own data.

3.2 Preprocessing of Board States

Preprocessing poses the greatest problem. Although some of the ANN game-playing successes mentioned earlier take in raw board states (i.e., pixel images), it is far from obvious that a raw array of integers houses enough accessible information for an ANN. You may need to massage these states with some scaling and/or more advanced interpretation. In the latter case, you would define salient features of a board state (such as the presence of the high tile in a corner or the degree to which a snake or other gradient has formed) and then re-represent each state in terms of those features, which would then serve as the ANN's inputs.

This is where the *knowledge engineering* of Good Old-Fashioned AI (GOFAI) meets the statistical brute force of bottom-up, deep learning, where the art of representation design meets the science of efficient number crunching. Good luck!

3.3 Visualization

The same 2048 game viewer used for the earlier module should be reused here, so that the skill of your ANN can be easily evaluated. For the report, you will need to take a few snapshots of the game to illustrate the intelligence (or lack thereof) of your ANN.

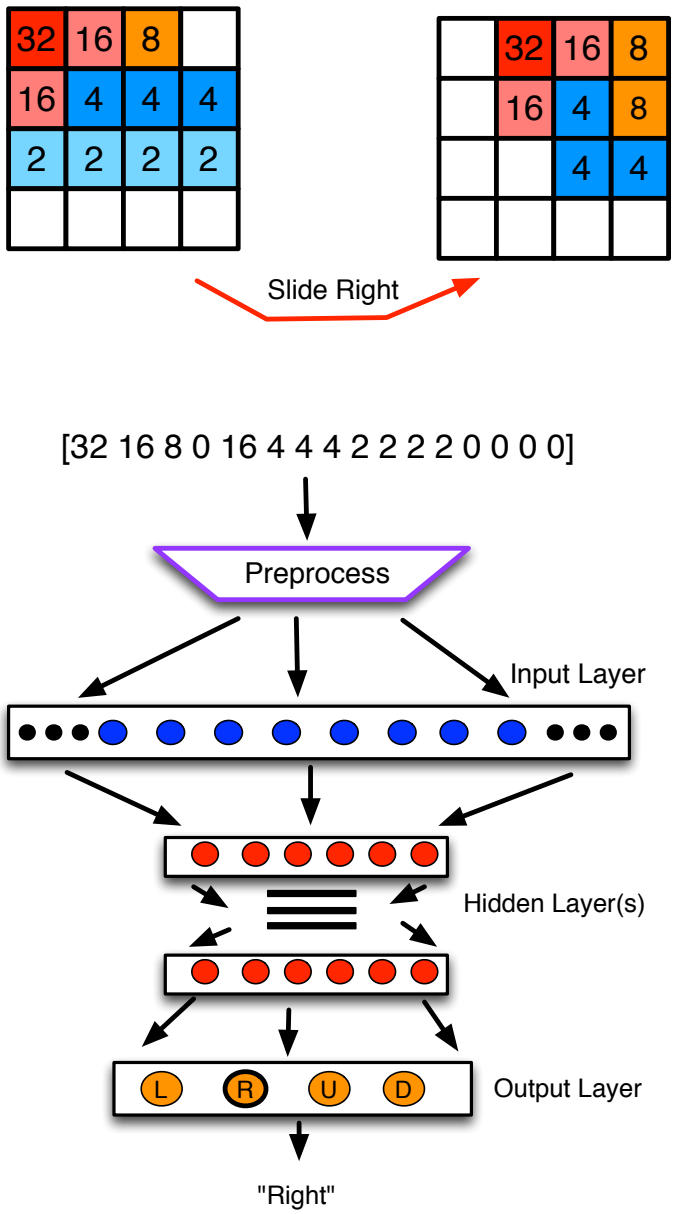

Figure 1: The basic framework for training the neural network to play 2048. (Top) Moves chosen in response to particular game states constitute case pairs: (state, move). (Middle) The state is preprocessed and then entered into the input neurons of the network. (Bottom) After feedforward propagation of signals, the output node with the highest activation level (thick border) constitutes the network's action choice. Through backpropgation training, that choice should eventually match that of the case pair.

3.4 Performance Evaluation

Your ANN's performance will be compared to that of a random player over the course of 100 games (50 for each type). For each set of 50 games, the highest tile (at game's end) will be recorded. The two 50-element lists will then be analyzed and compared to assess the statistical significance of your ANN's *better* performance. To get any (of the 7) demonstration points for this module, your ANN must have a higher average tile over the 50 runs than does the random player (over 50 runs). Assuming a better average, the two-tailed p value returned by Welch's t-test will determine your performance score via the following formula:

$$s = \lceil -\log_{10}(p) \rceil \tag{1}$$

The score will be rounded up to an integer value in the range [0,7]. For more on Welch's T-test, check out the Python function scipy.stats.ttest_ind (with the *equal_var* flag set to False).

For the actual demonstration, you will need to use the file **ai2048demo.py** (which will be provided ahead of time) and to include it in your main ANN file using:

import ai2048demo

To use this module to calculate a performance score, you must do the following (at the demo):

- Perform 50 runs of the ANN-based player and 50 runs of the random player to produce two 50-element lists of high-tile values, Lr (for the random player) and La (for the ANN player).
- Make the following call:

  ai2048demo.welch(Lr,La)

  where Lr MUST be the first argument, and La the second.
- Print the result returned by *welch*.

You are sending both 50-element lists to *welch*, which will return a string containing the two-tailed p value along with your performance score.

4 Deliverables

1. A report (of length no greater than 4 pages) that includes the following:
   - A **discussion** of the choices made in designing your ANN, particularly those involving topology, activation function, learning rate, momentum, etc. *Justify* your choices; don't just state them **(2 points)**.
   - A description of TWO different representations (i.e., two forms of preprocessing) that were attempted, and a comparison of the results achieved by each. These results should involve enough runs (e.g. 50) and analysis to statistically justify the choice of one representation over the other. Use Welch's T-test (see function scipy.stats.ttest_ind with the *equal_var* flag set to False) to verify that one representation beats the other. The random variable to sample from each 2048 run should be the high tile at run's end (rather than, say, the number of moves or sum total of all tiles). **(4 points)**

- An analysis of your best ANN playing one game of 2048. This should highlight a few points in the game where it made smart decisions, and others where it appeared to make unwise moves (unless, of course, your ANN plays perfectly). **(2 points)**

2. A demonstration of your system that includes training, testing, and a comparison between the trained ANN and a random player, each playing 50 games (as described above). **(7 points)**

Important! At the demo session, you may be asked to run a few different ANNs that differ in their topology (i.e., number and size of hidden layers). Be sure that you can run your system using different types of networks **without editing your source code**. Your system needs to be general enough to accept a simple topology description (that specifies the number and size of hidden layers in whatever format you choose) and to build a fully-functioning network from it. Failure to satisfy this simple requirement can entail a loss of points.

4.1 Other Practical Information

A zip file containing your report along with the commented code must be uploaded to It's Learning prior to the demo session in which this project module is evaluated. You will not get explicit credit for the code, but it is crucial that we have the code online in the event that you decide to register a formal complaint about your grade (for the entire course).

The 15 total points for this module are 15 of the 100 points that are available for the entire semester.

The due date for this module is the 3rd demo session.