Massachusetts Institute of Technology

6.038 Artificial Intelligence in Practice

Problem Set 2B: Thursday, February 12, 1998
DUE: Thursday, February 19, 1998

1 Files

All the files for this assignment are in `/mit/6.038/ps2/src/`.

2 Background

The objective of this assignment is to get you thinking about machine learning in problems where actions are called for and the "right" answer is not necessarily known ahead of time. While we are at it, we'll review some simple concepts from search and game playing.

2.1 Search

Look at the code in `nn-robots.c`, which is an extended version of the `robots.c` file from Part A. As before, we've given you the code for `pick_move_by_search`. However, we have only given you an outline for the key functions `evaluate_move` and `static_score`. The idea is to look ahead a specified number of moves (`lookahead_depth`) and select the current move that leads to the best final board position. The comments in the file give you some hints on behavior. The biggest hint is that we only took out about 20 lines of code.

Problem 1: You are to finish this code to try to get performance similar to (or better) to the one we have provided you. Hand in ONLY the definitions of `evaluate_move` and `static_score`. If you get stuck on this, proceed to the next parts; they are independent of this one.

2.2 Supervised Learning

We can try to learn to play the game by training a neural net to play the way some "expert" plays. We can do this by recording a large set of games played by an expert and using that for training samples. We have two sorts of experts available: you and the search-based player. A file containing all the games you saved when you did Part A is available as `part-A-games`. We have also saved 1000 games played by the computer, with default settings, in `search-games-1K`. We have also saved a smaller file with 100 games `search-games-100`. All of these saved games were somewhat abbreviated, they went only 10 moves at most, since within ten moves these games are mostly decided and this way we could get more variety. By the way, the computer "won," i.e. survived 10 moves, 809 of these 1000 games.

You can see in `train-nn.c` a simple interface for the `backprop.c` code from last assignment. The file `netinput.c` has code to do two different encodings of the board, one is the board itself (100 input units) and the other uses the relative position of the robots to the player.

To train a network, you can use `train-nn` as follows:

```
train-nn GameDataFile(input) NeuralNetFile(output) [num_hidden] [num_epochs]
        [learning rate] [momentum] [NN_input_encoding]
```

The `GameDataFile` is one of the files produced by saving games. The `NeuralNetFile` will be a stored neural net as in Problem Set 1; if an existing file is given here, training proceeds from where it left off. `num_hidden` is the number of hidden units in the net, which defaults to 10. `learning rate` and `momentum` default to 0.3 as in Problem Set 1. The `NN_input_encoding` chooses the board encoding; the default is 0, which is the "whole board" encoding but specifying 1 will choose the "relative position" encoding.

While running, the code produces output lines as follows:

```
Epoch 326  1717.450 1913 errors out of 7741 patterns
Epoch 327  1715.993 1911 errors out of 7741 patterns
```

The first number is a sum of the "deltas" as in Problem Set 1. The next number shows how many times in one epoch the current net chose a move different from the "correct" move.

To have the computer use a trained net to play, use the `-u NeuralNetFile` option for `robots`.

We have given you a number of pre-trained nets whose names are `search-Xh-Yg.net`, where X indicates the number of hidden units and Y indicates the number of games used in training. We have also trained a net with your saved games from Part A, these are called `part-A-Xh.net`, where X is the number of hidden units.

Problem 2: Experiment with setting the parameters `num_hidden`, `learning_rate`, `momentum` and `NN_input_encoding`. Try to find the best setting to train a net using the 100 stored search games. If you have a fast machine, you can try using the 1K stored games for some of your runs (this will be much slower). In the interest of saving you time, you don't have to run these training sessions to convergence, but run them at least 100 epochs each, so you get a feeling where it's going. Report and try to explain your observations. Hand in only the final `Epoch x...` line of any runs.

Problem 3: Use the stored nets we provided you to get a feeling for the level of play (how many times they win out of 500 games, for example) of the various nets. Remember to set `-d 0` when trying these experiments.

2.3 Reinforcement Learning

An alternative strategy is to let the machine learn by observing its own play and the results. This is the approach we called *reinforcement* learning in class.

Problem 4: Implement the TD method of reinforcement learning. For this, you will need to provide code for the function `pick_move_by_TD` in `nn-robots.c`. The paper by Tesauro is fairly explicit about how this can be done; it is remarkably simple. Hand in just the code for this function.

Problem 5: You can either use your implementation or ours to do this part. Let's compare the effect of numbers of hidden units. Try the following for 10 and 20 units. Plot the progress of learning by having the machine play the following numbers of games in sequence: 100, 1000, 5000, 10000, while recording how many wins there were during each training episode. That is, execute the following sequence:

```
robots -t td-10.net -h 10 -g 100 -d 0
robots -t td-10.net -h 10 -g 1000 -d 0
robots -t td-10.net -h 10 -g 5000 -d 0
robots -t td-10.net -h 10 -g 10000 -d 0
```

and a similar sequence using 20 hidden units. Record after each run the percentage of wins. Note that training here is **cumulative** when you use the same net. Hand in your data and compare performace with the search code and the supervised neural net training. Briefly, discuss your conclusions.