CSC3022H: Machine Learning

Practical Assignment:
Artificial Neural Networks and Reinforcement Learning

Department of Computer Science
University of Cape Town, South Africa

August 29, 2015

Overview

This practical assignment requires you to implement two machine learning algorithms to solve a simulated mine-sweeper task.

1. The Back-propagation algorithm (a *supervised learning* method for training Artificial Neural Networks). You have **3 weeks** to complete this part of the assignment **(Hand-in 28 September)**.

2. The Q-learning algorithm (a *reinforcement learning* method). You have **3 weeks** to complete the Q-learning algorithm **(Hand-in 19 October)**.

Simulation Framework

The machine learning algorithms are to control a set of autonomous mine sweeper agents in a two-dimensional world of mines, rocks and super-mines (figure 1). The exact parameters (mines, sweepers, iteration length (ticks), frame rate, etc.) for the simulation are set in the `CParams.ini` file. Super-mines are destroyed upon collision and re-created in the next iteration.

When a sweeper collides with the one of the rocks only the sweeper is destroyed. If a sweeper collides with a mine, the mine is removed from the field and the sweeper's statistics are updated.

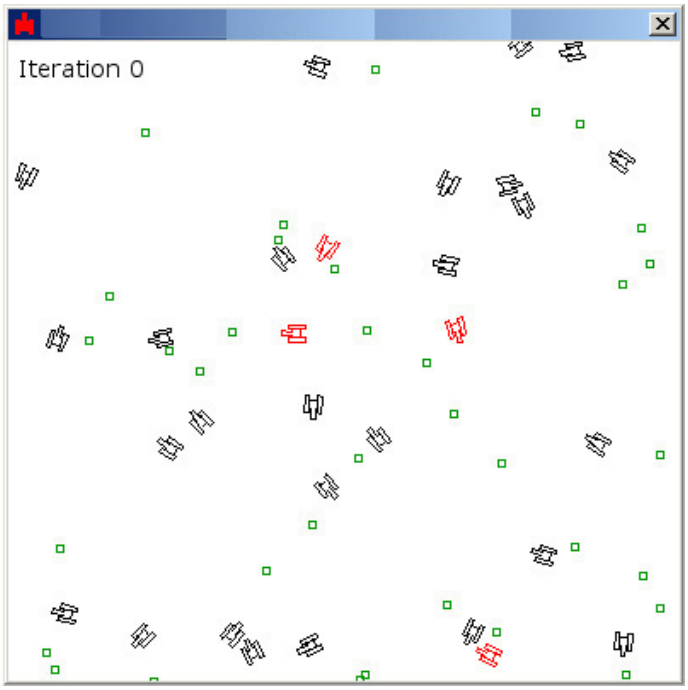

Figure 1: Simulation world consisting of minesweepers, mines and super-mines. The mines and sweepers are spawned at random positions within the world at the start of the simulation. The world wraps around on both the x and y axis (i.e. if the sweeper leaves the world on the window on the one side they reappear on the opposite end of the grid). The sweepers should ultimately learn to gather mines (green squares) and avoid rocks (black squares) and super-mines (red squares).

The framework to set up the world, rendering and update loop is supplied. It should not be necessary to modify the framework's core functionality (ie. the draw and update loops), aside from the two algorithm controllers and supporting methods. These are:

- `CBackPropController.cpp` and the supporting file `CNeuralNet.cpp`
- `CQLearningController.cpp`

Each of the controllers implements a carefully defined interface.

The `CBackPropController.cpp` and `CEAController.cpp` controllers are continuous environments and inherits from `CContController.cpp`.

`CQLearningController.cpp` is a discrete (grid) environment and inherits from `CDiscCollisionObject.cpp`.

The two controllers `CBackPropController.cpp`, `CQLearningController.cpp` override the following two methods:

- `virtual void Update(void);` This method must call the Update method of the parent controller.
- `virtual void InitializeLearningAlgorithm(void);`

You will need to fill in the details of the methods for each controller and supporting classes, paying special attention to the comments in each file. The driver file `main.cpp` allows you to easily switch between the two algorithm controller classes. You only need to modify the line after the includes section to one of the following:

typedef CBackPropController `PRAC_ALGORITHM`

//Backpropagation Algorithm

typedef CQLearningController `PRAC_ALGORITHM`

//Q − Learning Algorithm

The statistics of the current minesweepers are drawn by the PlotStats() function when the F-key is pressed. These graphs (depicting maximum and average number of mines gathered) will be drawn correctly when your mine sweepers learn to gather mines.

The framework uses the Windows `WIN32` windowing API and is bundled as a Visual Studio 2013 project. Both the Senior and the Games lab have copies of Visual Studio installed.

Submission Details

Important: At the end of each part you must submit the entire framework (excluding binaries) with your algorithm implementations, along with a short report (not more than 500 words) as a `ZIP` archive. Your code should be under `git` version control. The report should include:

1. Method pseudo-code and parameters. Be sure that the pseudo-code relates to task specific constraints and objectives.
2. Table of average results, including the average number of mines swept and number of agents destroyed, where 20 runs (for a learned behaviour) must be done for each controller method (i.e. back-propagation and reinforcement learning). Results must be averaged over these 20 runs.

Test Environments

To verify learned behaviour, agent controllers trained by each learning method must be tested on the following environments. For each environment, be sure to record the number of mines gathered and minesweepers destroyed.

Environment 1: Initialise the environment with 30 minesweepers, 40 mines and 10 super-mines.

Environment 2: Initialise the environment with 30 minesweepers, 25 mines and 25 super-mines.

Environment 3: Initialise the environment with 30 minesweepers, 10 mines and 40 super-mines.

NOTE: For each environment, one simulation can run for only 2000 iterations (ticks). At the end of the simulation, mine-sweeper task performance is measured by the number of mines gathered, and the number of mine-sweepers remaining.

For your learning method you can run as many trial simulations as is necessary for the method to learn the required behavior.

Part 1: Implement an Artificial Neural Network with Back-Propagation

The back-propagation algorithm is a classic supervised learning algorithm, based on gradient descent. You can refer to Chapter 18 of *Russell and Norvig (2009) Artificial Intelligence: A Modern Approach*[1] or Chapter 4 of *Mitchell (1997) Machine Learning*[2] for a detailed description.

We have given you a basic controller class which will read the supplied training data file `training_data.txt`. In this file the possible dot products between the nearest rock, super-mine and mine are enumerated and neural network responses are specified as *turn towards* or *turn away*. This is only a simple solution. You can modify the training data and controller to accommodate the distance between the sweeper and the mine in order to speed it up or slow it down should you wish. A simple `Numpy` script have been supplied as `data_gen.py`.

Your primary objective is to fill out the stripped down Neural Network class, `CNeuralNet.cpp`. You may modify the structure of the class if you wish, but the algorithm essentially comprises of 4 steps and method definitions and structure have been defined for you.

[1] http://aima.cs.berkeley.edu/
[2] http://www.cs.cmu.edu/~tom/mlbook.html

BACK-PROPAGATION ALGORITHM:
DO
  for all training examples
  propagate forward
  get classification
  compute the error
  propagate the error backwards from the output layer to the input layer
UNTIL the neural net classifies the examples accurately

You should define and allocate the memory needed to store the network weights for a single hidden layer fully connected network. You can come up with more complex schemes, but this one will likely be the simplest to implement. If successful the Mean Squared Error will decrease as your network trains. You may have to adjust the learning rate and parameters to suit your needs. You may find the sigmoid function and its first derivative useful:

$$S(t) := \frac{1}{1 + e^{-t}}$$
$$\dot{S}(t) = S(t)(1 - S(t))$$

HAND-IN DATE: Monday, 28 September at 10:00am

Part 2: Implement the Q-learning Algorithm

The Q-learning algorithm is a reinforcement learning approach. For a detailed description of the algorithm refer to the paper *Watkins, Christopher JCH, and Peter Dayan. "Q-learning." Machine learning 8. 3-4 (1992): 279-292* [3]. There are also many other resources available online.

The algorithm keeps a table containing all possible states and actions, and progressively updates the table according to the reward function. Since all possible state-action pairs have to be tracked it is easy to see why the world must consist of a finite number of grid positions. You can assume that each mine-sweeper can only move up, down, left and right, but that they must move every step of the update cycle. The sweepers can for instance be rewarded for completing the task at hand (clearing the field) and penalized for finding nothing. This will culminate in exploratory behavior. You can decide on the exact details, learning rates and discount factors to use, but comment on your choices and results in your report.

Q-LEARNING ALGORITHM:
DO
  Observe the current state
  Select the action (in the current state) with the highest historic return
  Move the agent to the new state (based on the action selected)
  Observe the new state
  Update the table of state-action pairs accordingly
UNTIL satisfactory convergent behavior / goal state is reached

HINT: You may find keeping a separate table for each mine-sweeper useful when implementing the algorithm for a set of sweepers. You can assume that the experiment can run for an as long as it takes the agents (or a subset) to develop some useful behaviour.

HAND-IN DATE: Monday, 19 October at 10:00am

---

[3] You can use the library *ezproxy* tool http://ezproxy.uct.ac.za/ to download most articles from sites such as IEEE Explore when you're off campus.

PLEASE NOTE:

1. You must submit a working Visual Studio 2013 Solution or a Make-file to compile your program. If it does not compile a 50% penalty will apply.

2. If you add additional files and or have any compilation instructions you must provide a README file explaining what each file submitted does and how it fits into the program as a whole. The README file should not explain any theory that you have used. These will be used by the tutors if they encounter any problems.

3. Your learning method description and motivation should be compiled as a PDF document and be no more than 500 words.

4. Pseudo-code for your minesweeper learning method should appear in the same PDF document.

5. Do not hand in any binary files.

6. Please ensure that the ZIP archive of your git repo works and is not corrupt (you can check this by trying to extract the contents of your archive - make this a habit!). Corrupt or non-working archives will not be marked - no exceptions! Please do not use other formats like WinRAR / 7zip / etc.

7. A 10% penalty per day will be incurred for all late submissions. No hand-ins will be accepted if later than 7 days.

8. This is NOT teamwork. Do not copy. All code submitted must be your own. Copying is punishable by 0 and can cause a blotch on your academic record. Scripts will be used to check that code submitted is unique.