**School of Computing, Edinburgh Napier University**

**Assessment Brief Pro Forma**

| | |
|---|---|
| **1. Module number** | SET10107 |
| **2. Module title** | Computational Intelligence |
| **3. Module leader** | Kevin Sim |
| **4. Tutor with responsibility for this Assessment** Student's first point of contact | Kevin Sim |
| **5. Assessment** | Report && Demonstration of Code |
| **6. Weighting** | 60% of overall module total: |
| **7. Size and/or time limits for assessment** | Report - 6 Pages |
| **8. Deadline of submission** | **Report** : Friday 12th April 2019 @ 15:00 Via Moodle<br><br>**Demo**: Monday 8th April 2019 in your usual Lab slot in the JKCC |
| **9. Arrangements for submission** | *Submit your report to Moodle by the deadline*: 12th April 2019 @ 15:00<br><br>*You **must** demonstrate your code during your usual practical slot on Monday 8th April 2019.* ***No demo no mark****.* |
| **10. Assessment Regulations** | All assessments are subject to the University Regulations**.** |
| **11. The requirements for the assessment** | Please see the attached document |
| **12. Special instructions** | See attached document |
| **13. Return of work** | Within 3 weeks of submission. |
| **14. Assessment criteria** | See attached document |

## Description

You are required to evolve weights for a MLP (Mutli-Layer Perceptron) artificial neural network that will be used to control a fleet of identical spacecraft. The objective is to evolve a single controller that is able to land the fleet safely regardless of their different starting positions and starting velocities.

Each spacecraft is equipped with three thrusters that can be independently switched on or off by the neural network. The neural network has three outputs, each controlling a separate thruster. The thrusters all have equal power and operate at full thrust when switched on. Thrusters are directed right, left and down and when activated apply a fixed force in the opposite direction.

Spacecraft can start at different locations near the centre of a 600×600 square universe. Each spacecraft can start with a different starting velocity (direction and speed). The spacecraft are subject to a fixed gravitational force that affects the spacecraft's trajectory.

To land safely a spacecraft has to navigate to a landing pad situated at the bottom centre of the universe. A successful landing requires a spacecraft to touchdown as gently as possible and as close to the centre of the landing site. Due to health and safety regulations each spacecraft must only land after using up as much of its fuel as possible.

You are provided with Java code consisting of:

- A simulation of the spacecraft and the simulated universe.
- A fitness function that gives a measure of success or failure.
- A neural network controller.
- A partially implemented evolutionary algorithm [1].
- Other useful classes as examples

The code supplied comes with three different fleets of spacecraft to allow you to train and test your implementations in as wide a range of conditions as possible.

- The ***training*** fleet contains eight spacecraft with different but fixed starting positions and velocities. These should be used to train you algorithm.
- The ***test*** set also contains eight spacecraft with fixed starting positions and velocities to allow you to verify the ability of the evolved controller to generalise to different conditions. These should be used to test your algorithm.
- A *random* fleet of eight spacecraft is also provided for experimental purposes. These should not be used for your final report.

[1] You are required to complete selection, crossover, mutation and replacement operators

- The random fleet of eight spacecraft are initialised at random positions and random velocities, suitably far away from the boundaries of the universe.

## Objective

The objective is to implement an evolutionary algorithm (EA) (or other suitable search technique) to evolve the weights of the supplied neural network controller. You can use the partially completed code provided and implement the missing evolutionary operators or implement your own search algorithm. As well as the Example Evolutionary Algorithm, a simple hill climber is provided as an example of how to extend the code.

The spacecraft simulation, fitness function and neural network are packaged into a jar library to prevent modification. The Classes outside the jar package can be modified where detailed in the code*. [Note that the jar library also contains the source code if you are interested in the implementation details but this **must not** be modified for the purposes of the coursework]*.

You are required to write a scientific report using the template provided (both Word and Latex templates are provided in ACM format). The report should detail any research you undertook while investigating the task and should provide details of your implementation. You should include a scientific analysis of your approach by conducting multiple runs on both training and test sets to show that your implementation can provide consistent results.

## Modifiable classes

***ExampleEvolutionaryAlgorithm:***: You are provided with a partially implemented EA. Methods for Initialisation and Mutation are provided. You are expected to add selection, crossover and replacement operators. You may also improve the existing mutation operator and add further operators if required. The code that should not be modified is clearly marked.

**ExampleHillClimber:** A simple hill climber that works on a single solution. This provides an example of how to extend the neural network and implement your own search algorithm should you choose to do so.

**Parameters:** The code that should not be modified is clearly marked. You are free to modify the parameters relating to the EA in this code. You can adjust the number of Hidden nodes in the Neural Network. The number of input and output nodes should not be altered.

Two further classes are provided to assist implementation.

**StartGui** opens a graphical interface that allows you to train and test your implementation. The user interface allows you to get a feel for the implementation but runs slower than the console application.

**StartNoGui** includes an example of how the code can be executed without the user interface. This is useful for conducting multiple runs for your reports.

You may also Extend the NeuralNetwork Class with your own implementation. An example of how to do this is provided in ExampleHillClimber. To enable your extended class to run from the GUI you need to change the neuralNetworkClass variable in the Parameters class. Various other methods are provided to ease implementation. See the code for more details

A simple User Interface is provided to allow you to get a feel for the application. It is recommended that you conduct your final experiments without the graphical interface.

**TODO**

You are supplied with a basic framework in Java that implements some of the basic functionality required to set up a neural network and implements a skeleton EA. At minimum you must:

- Add missing evolutionary operators (selection, crossover, replacement etc.)
- Evaluate the performance of your algorithm(s) on the training and test scenarios given
- Report the fitness on both the training and test sets averaged over at least 10 runs of your algorithm.

In addition, you may wish to:

- Conduct a parameter exploration to tune the algorithm you have designed
- Investigate different activation functions for the neural network
- Investigate the number of nodes that should be in the hidden layer
- Redesign the evolutionary algorithm
- Investigate the role of different operators

You must NOT

- Alter ANY of the code supplied in the jar libraries

You CAN

- Rewrite the code in a different language if you prefer - however, it is your responsibility to ensure that you encode the neural network and the fitness function correctly.

Read the following very carefully:

You can be as creative as you wish in designing the evolutionary algorithm – the aim is simply minimise the average fitness for the training and test sets provided

A small prize will be given to the student that finds the best network during the demo (although this will not affect your marks).

1. The **maximum number of function evaluations that your algorithm can run for is 20,000**. You can use more or less during your investigation but for your final report, results using a maximum of 20,000 function evaluations should be reported.

2. The purpose of the coursework is to test your understanding of neural networks and evolutionary algorithms. You **do not** get any credit for parallelizing the algorithm, running in multiple threads etc.

There are two parts to the hand-in: a **demo** and a **report**

## Demo

At the demo, you will need to demonstrate your code, and be able to explain what you did.

***The demo does not carry a mark – however, if the demo is not completed satisfactorily, then no mark will be awarded for the report.***

## Report

You need to write a report detailing what you did, and presenting the results of your experiments. It should have a **maximum length of 6 pages and be written using either the Latex or Word template supplied (ACM conference format). Marks will be deducted if you exceed the page limit.**

The report should cover:

1. **Approach (20 marks)**

   Describe the approach you took to solving the problem. This section should cover:
   - Any background reading you did to inform your design
   - A description of each of the new operators you added
   - A description of the EA you used
   - A description of the neural network parameters used

   You should justify your design and choices in each case, giving reasons for your choices.

2. **Experiments & Analysis (25 marks)**

   Describe any experiments that you undertook in order to investigate the performance of your algorithm and to tune any parameters. Be careful to give sufficient detail that another reader could reproduce your experiments. Use graphs or tables to summarise your results - do not simply give the result of every experiment you ran.

   Comment on any trends or observations. Marks will reflect the quality of the experimental design, presentation and analysis of your results.

## 3. Conclusions (10 marks)

Include a single table that specifies for each function (training and test sets):

- the mean fitness obtained from at least 10 runs of your algorithm
- the parameters used to obtain this result
- the activation function used by the neural network

Comment on the success (or not) of your approach, reflecting on decisions made such as the choice of operators or parameter values and noting any particularly interesting observations made during your experiments

## 4. Future Work (5 marks)

Explain what you could do in the future to improve the algorithm. This might include taking a completely different approach, or modifying your current one.

## Important detail

- **TOTAL MARKS: 60**
- **Hand-in date: Friday 12th April 2019 @ 15:00 Via Moodle**
- **Demo: Monday 8th April in your usual Lab slot in the JKCC**

## Guidelines for Marking Scheme

**40-50%:**

- Basic operators implemented; evolution demonstrated for training set, possibly high values of average fitness however; little or no evidence of parameter tuning; justification of choices or reference to literature; weak presentation of results; critical analysis lacking or very limited

**50-60%**

- Basic operators implemented; evolution demonstrated and reasonable values of average fitness; limited parameter tuning; some attempt to justify choices demonstrating understanding; adequate presentation of results; limited critical analysis

**60-70%**

- Good choice of operators implemented; evolution demonstrated for all networks, good values of fitness on training and test sets; parameter tuning; good attempt justify choices demonstrating understanding and limited references to literature; good presentation of results; some critical analysis

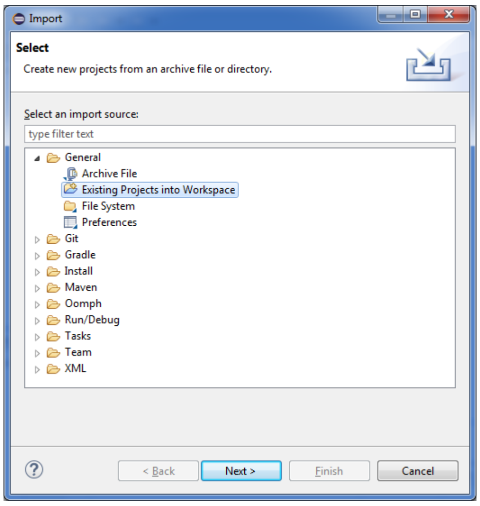
**70% +**

- Carefully designed operators that clearly demonstrate understanding of problem and course; related to relevant literature; low values for average fitness; thorough parameter tuning conducted; presentation of results shows insight into trends etc; in-depth critical assessment of results.

## Appendix

### Instructions for opening code

Using a new eclipse Workspace select import Existing Project from the file menu and then select the archive (zip) you downloaded from Moodle. The zip will be extracted to your Workspace



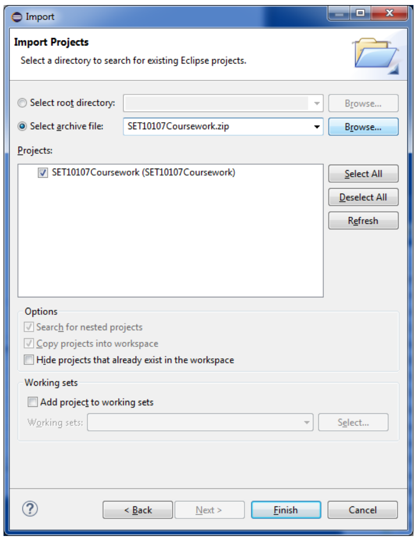
Import

**Select**

Create new projects from an archive file or directory.

Select an import source:

- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - Preferences
- Git
- Gradle
- Install
- Maven
- Oomph
- Run/Debug
- Tasks
- Team
- XML



Import

**Import Projects**

Select a directory to search for existing Eclipse projects.

( ) Select root directory:

(•) Select archive file: SET10107Coursework.zip

Projects:

[✓] SET10107Coursework (SET10107Coursework)

Options

[✓] Search for nested projects

[✓] Copy projects into workspace

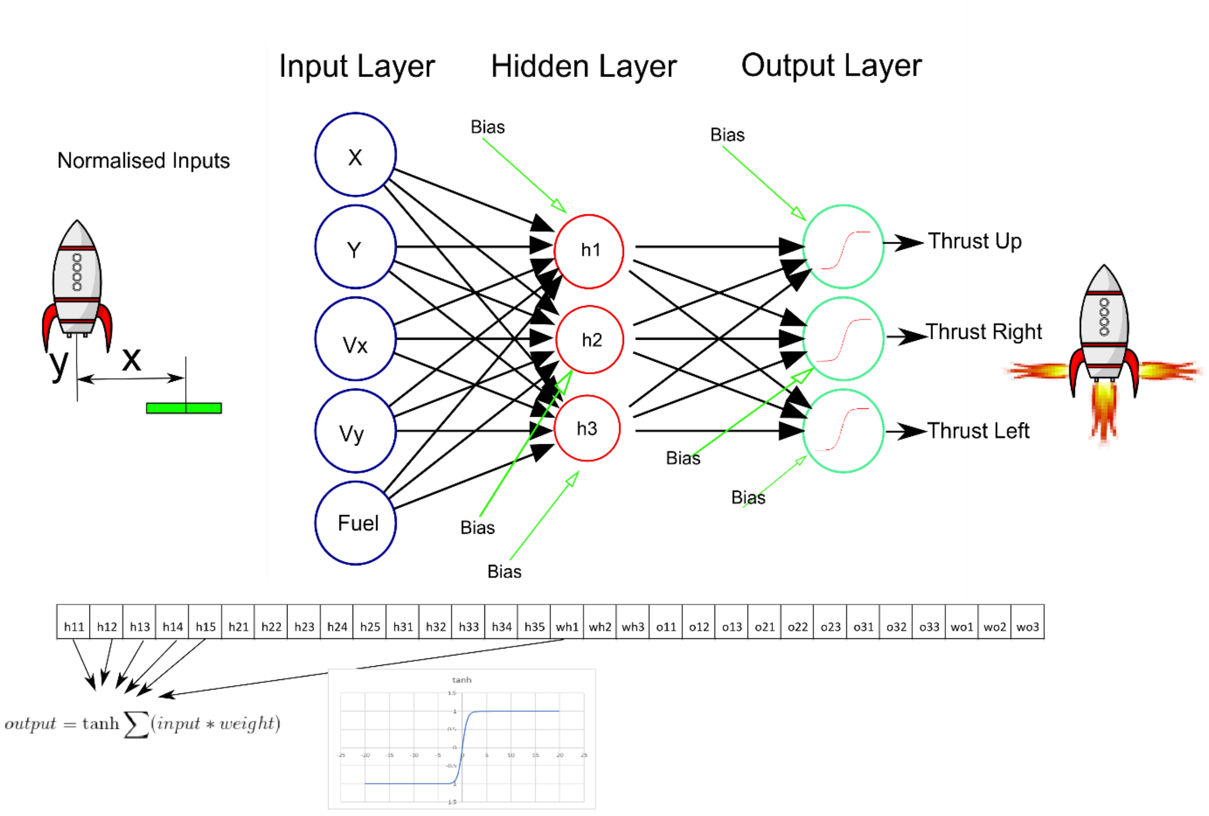
[ ] Hide projects that already exist in the workspace

Working sets

[ ] Add project to working sets

Working sets:

### Code Overview



$$output = \tanh \sum (input * weight)$$

*Figure 1: Code Overview*

Figure 1 gives an overview of the code. The neural network has 5 inputs representing the x and y coordinates of the spaceship, the horizontal and vertical velocities and remaining fuel. All inputs are normalised in the range 0 – 1;

The number of hidden nodes is set in the Parameters file. If you change this from outside the Parameters class (programmatically) then you should use the *setHidden(int num)* method provided in Parameters. Changing the number of Hidden Nodes changes the number of weights in the chromosome.

The chromosome represents the weights on the connections of the Neural Network. In the example shown with 3 hidden nodes the first 5 weights represent the weights between the 5 inputs and hidden node 1. The next 5 weights represent the weights between the 5 inputs and hidden node 2. The next 5 weights represent the weights between the 5 inputs and hidden node 3. The next 3 weights represent the bias weights of each of the hidden nodes. This sequence is repeated for the output layer.

There are three outputs, each controlling a different thruster. If the output from an output neuron > 0 then the thruster is turned on. If the output is <= 0 the thruster is turned off. If you decide to modify the activation function then you should make sure that it operates to these criteria.

The fitness function is the average fitness over 8 simulations. Each simulation in the training and test sets has a different but fixed starting position and velocity. The fitness of an individual simulation is given by

***Fitness = (D + V + F) / 3***

Where

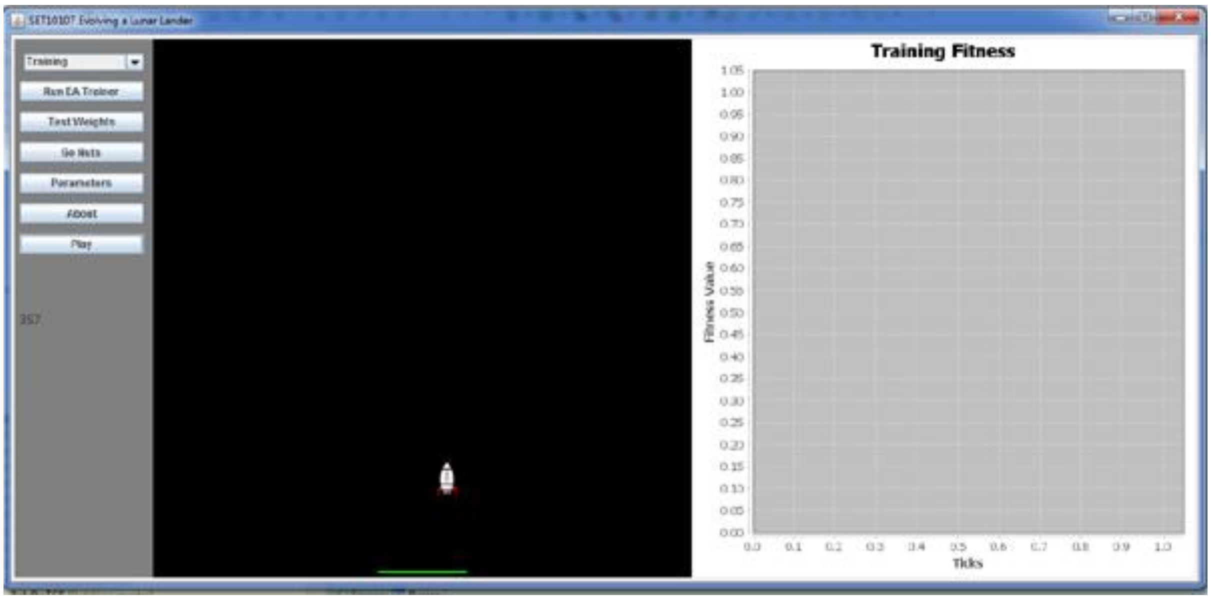
- **D** is the normalised distance from the landing position at the end of a run
- **V** is the normalised velocity at the end of a run
- **F** is the amount of remaining Fuel at the end of a run (normalised)

The optimal fitness is zero but beware that if you manage to get close to this on the training set then you have most likely found a network that has overfitted to solve the training set but will provide poor results on the test set.

## GUI

A user interface is provided for experimental purposes, you are encouraged to conduct your experiments on your final design programmatically. An example of how to do this is provided in class *StartNoGui*

To start the gui run the StartGui class



- The **Combo Box** allows you to set the current data set. For your report you should train on the training set and test on the test set only.
- Button **Run EA Trainer** starts the Neural network Trainer specified by the static variable

  *Parameters.neuralNetworkClass*

  Note that Parameters must be set prior to starting training. At the end of training the best weights will be saved to disk using the syntax xxxxxxxxx-N.txt. Where xxxxxxxxxx is the current system timestamp and N specifies the number of Hidden nodes. This file also contains the fitness at the end of training and the parameters that the algorithm was started with.

- When training, the graph will show you the current fitness Vs the number of calls to the GUI. This may be different to the number of evaluations used to terminate your code. For example using the supplied EA there are two offspring and two fitness evaluations for each generation but the GUI is only called once. The EA will run for 20000 evaluations but the graph will only show 10000. It is currently not possible to stop the training procedure without quitting the application.

- Button **Test Weights** allows you to load a set of weights from disk. You can select any data set to evaluate your weights on but for your final reports you should be testing on the test set only.

- Button **Go Nuts** starts 180 spacecraft at 2 degree intervals with equal magnitude velocities acting in the direction determined by the line from the centre to the starting position of each aircraft (radial pattern). This is provided for “fun” and should not distract you from the task of training your network.

- Button **Parameters** shows a pop up window displaying the current parameters

- Button **About** shows information about the Application

- Button **Play** allows you to take control of a single randomly positioned spacecraft. Use the Up, Right and Left arrow keys to control the spacecraft. The current fuel level is displayed underneath the Play Button.

If a spacecraft lands too quickly, exits the universe or misses the landing pad then the image will change to reflect this. This makes no difference to the average fitness that you should detail in your report.