Test Report

Full name:	Endri Kastrati
student id:	24064580
Instructor:	ABM Russel

1.0 Introduction

One of the most important aspects of the project life cycle management is the testing phase. Human brains are prone to errors which, depending on the circumstances, might result in catastrophic consequences for the company, organisation, government or user that is deploying the software/project. This report provides a detailed explanation and walk-through of the software tests that were performed during the implementation of the computer science project titled “A deep learning approach to thyroid disease classification”. The project had three deliverables:

- The design of an artificial neural network library
- The design of a unix application programming interface that interacts with the library.
- The design of a minimalist web application that interacts with the command line program.

The first two project deliverables were implemented using the C programming language and the GNU scientific library while the third one was created entirely using the nodejs environment and web sockets. The C programming language is very old and it does not have sophisticated unit testing libraries and most existing C testing frameworks are quite difficult to setup and follow due to poor documentation. Also given the nature of artificial neural networks, the only valid and proper form of functionality validation is the black-box testing technique. The usage of implicit testing methods will help us prove the correctness of the library and consequently of the command line program and web interface. The following sections will elaborate on the unit and system testing techniques that have been deployed during the project life cycle and the software that has been written to perform those tests.

2.0 Unit Testing

The design of the artificial neural network library was broken down into five modules and a main file. It was very important in this project to understand the concept of ANNs, the abstract components of the ANNs and how to map them to mathematical objects. Though no testing framework was used, the modules and data structures were thoroughly tested during their implementation phases using the standard assertion library of the c language, namely “<assert.h>” and manually written main programs that would test the constructors, the getters / setters and the deconstructors of the data structures. The main programs that would test the data structures, would also be tested using the valgrind memory checker to detect any memory leaks, buffer overflows and double free corruptions. Having initialised a local git repository, these tests can be accessed via git checkouts since every module completion has been recorded via commits. By navigating to previous commits, the instructor can review the manually written main files that perform tests on the corresponding data structures. It is worth mentioning, that because of the high levels of abstraction and careful planning and design of the ANN library, not many risks manifested during the implementation of the modules. The library also employs a few clever tricks that make it extremely hard to write proper unit tests, such as function pointers and type casting of matrix views into matrix pointers and vise versa and so on. Most procedures have explicit assertions that check the parameters and if any invalidity is triggered the program terminates immediately and prints into the standard error stream the failed assertion. These mechanisms were quite useful during the implementation phase of the modules. Since neural networks encode a generalizer within their synaptic weights it is really hard to device appropriate unit tests. Of course, we can prove the correctness using implicit techniques which will be elaborated extensive in the next section.

3.0 Black-box Testing

Convergence of mean square error

A proper way to validate the functionality of the ANN library and the command line program, is to observer the difference between the synaptic weights of consecutive training epochs. By executing the examples that have been included in the “params.txt” file it can be observed that indeed, the loss decreases with each training epoch and that the mean square error value convergences to a global/local minimum. This implicitly indicates that the synaptic weights are constantly being adjusted with each epoch such that the mean square error value is minimised. From this perspective, it seems that the back-propagation is working correctly.

Universal function approximators

A more formal way of proving the correctness of the ANN library and the command line program is to gather empirical evidence regarding curve-fitting applications and demonstrate that it holds the properties of a universal function approximator. For this project, three octave scripts were designed that would run the command line program will three different data-sets that describe the qudratic function, the semicircle function and the sin function, all three with limited ranges of course. To run the experiments the instructor must first make sure to compile the source code such that the command line program “neuralnet” is generated and then navigate into the “datasets” directory of the computer science project folder and execute the following commands:

```
$ octave –no-gui
> quadratic
> circlef
> sinf
```

Quadratic approximator

The “quadratic.m” file trains a neural network model that approximates the quadratic function. The results of the qudratic approximation test are show in the following image:

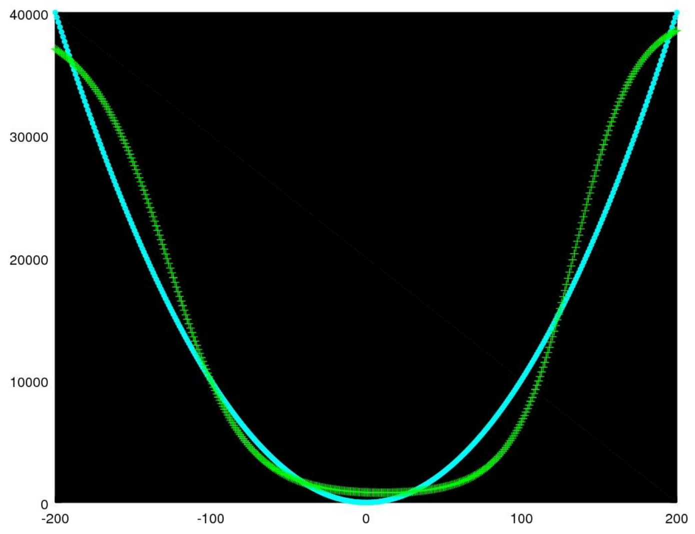

Illustration 1: blue represents the quadratic function, green represents the quadratic approximator

By running the “quadratic.m” file repetitively, a very good quadratic approximator can be obtained with just a single hidden neural layer that contains 10 neurons at most. Of course the qudratic function is not that complicated and does not reinforce our confidence regarding the correctness of the library, so we have to demonstrate that it can approximate more complicated functions.

Semi-circle approximator

The “circlef.m” file trains a neural network model that approximates the semi-circle function. The results of the semi-circle approximation test are show in the following image:

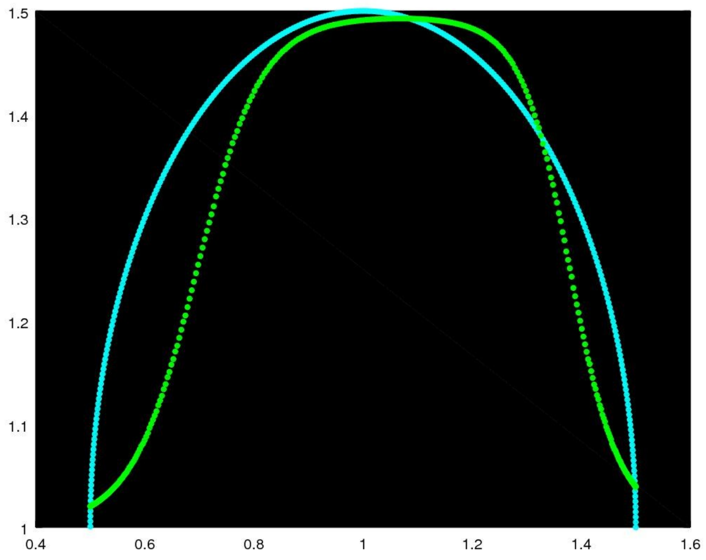

Illustration 2: blue represents the semi-circle function, green represents the semi-circle approximator

By running the “circlef.m” file repetitively, a very good semi-circle approximator can be obtained with just a single hidden neural layer that contains five neurons at most. We are getting closer to validating the correctness of the library as we have already approximated two functions, namely the quadratic and semi-circle functions.

Sin approximator

The “sinf.m” file trains a neural network model that approximates the sin function. The results of the sin approximation test are show in the following image:

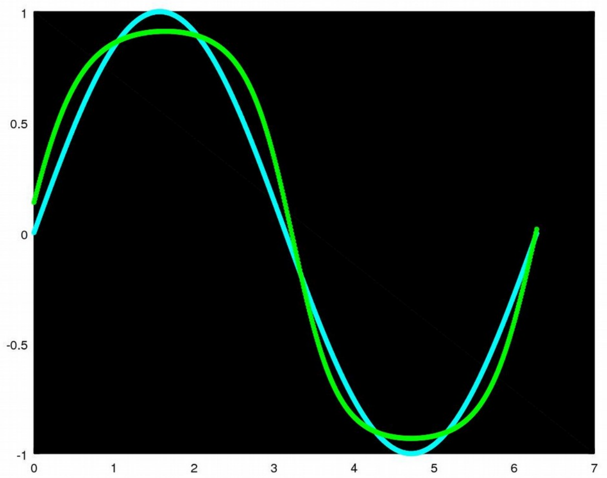

Illustration 3: blue represents sin function, green represents sin approximator

By running the “sinf.m” file repetitively, a very good sin approximator is obtained with just a single hidden neural layer that contains 12 neurons at most. By gathering empirical evidence, we have shown that the ANN library holds the properties of a universal function approximator and hence must be correct. In fact, this is the only formal and mathematically appropriate way that we can prove the correctness of the training process, namely by demonstrating the function approximation universality property of multi-layer feed-forward neural networks.

4.0 Stress Testing

The library was designed to be highly efficient and fault resilient. This was achieved, by taking advantage the manual memory management capabilities of the C programming language. What this implies, is that the command line program will never crash during a training or deployment process that it can sustain in terms of memory resources and CPU power. Thereby, the ANN library is only limited by the computer architecture that is hosting the library.

5.0 Testing via Resubstitution

Due to time constraints, the implementation of a cross-validation technique was not possible. Instead, a simple resubstituion test was used to validate whether the synaptic weights of the neural network model had indeed encoded a generaliser. By running the examples that are included in the “params.txt” file it can be observed that indeed a generaliser is obtained with a great returned accuracy when tested on the training data-set. Of course a cross-validation technique would have been more useful and a better performance evalution tool.

6.0 Integration Testing

By running the web application program located in the “webapp” folder, we demonstrate how the ANN library and the command line program can properly and efficiently interact with different environments and systems. While the command line program is meant for the computer scientist, APIs and interfaces can be built that interact with it by abstracting away all the technical and unnecessary details.

Conclusion

To sum up, the testing phase is one of the most important aspects of the project life cycle and this report presented the different types of testing methodologies that were used to validate the functionality of the three project deliverables. Through the gathering of empirical evidence we have demonstrated the correctness of the ANN library and have shown that the command line program is working as expected without any malfuctions or misbehaviours. At this point, the computer science project is ready for deployment.