8-2016

Analysis of artificial neural networks in the diagnosing of breast cancer using fine needle aspirates

Janette Vazquez
The University of Texas Rio Grande Valley

ANALYSIS OF ARTIFICIAL NEURAL NETWORKS IN THE DIAGNOSING OF BREAST CANCER USING FINE NEEDLE ASPIRATES

A Thesis

by

JANETTE VAZQUEZ

Submitted to the Graduate College of
The University of Texas Rio Grande Valley
In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2016

Major Subject: Computer Science

ANALYSIS OF ARTIFICIAL NEURAL NETWORKS IN THE DIAGNOSING OF BREAST CANCER USING FINE NEEDLE ASPIRATES

A Thesis
by
JANETTE VAZQUEZ

COMMITTEE MEMBERS

Dr. Laura Grabowski
Chair of Committee

Dr. Emmett Tomai
Committee Member

Dr. Dongchul Kim
Committee Member

August 2016

ABSTRACT

Vazquez, Janette, Analysis of Artificial Neural Networks in the Diagnosing of Breast Cancer using Fine Needle Aspirates. Master of Science (MS), August, 2016, 47 pp, 6 tables, 8 figures, references, 21 titles.

This thesis examines how Artificial Neural Networks can be used to classify a set of samples from a fine needle aspirate dataset. The dataset is composed of various different attributes, each of which are used to come to the conclusion as to whether a sample is benign or malignant. A Feedforward Neural Network was trained with the dataset using a Backpropagation training method. After training, the network performed a 10-fold cross validation to determine which model had the lowest error score. The data was looped through the model and the trained network classified the samples as either benign or malignant. Once classified, the overall accuracy, specificity and sensitivity were analyzed to measure performance. Three other neural networks were compared to the Feedforward Network to see how they performed. These three neural networks included a NEAT Neural Network, a Support Vector Machine, and a Radial Basis Function Neural Network.

DEDICATION

The completion of my master’s thesis could not have been possible without the love and support of my family. Special thanks to my mother, Maria, and my father, Fabian, for always inspiring and motivating me, as well as to my siblings for their support and love that came in various different forms.

ACKNOWLEDGEMENTS

The completion of my thesis would not have been possible without the advice from Dr. Grabowski, the chair of my committee, who I will always be grateful to for her mentoring, support, and patience as I completed the thesis process.

My thanks also go to my thesis committee members, Dr. Dongchul Kim and Dr. Emmett Tomai, for their participation, input and comments that helped ensure the quality of my intellectual work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION AND BACKGROUND

Introduction

Motivation

Breast cancer is a principle cause of death in women in developed countries – in fact, it is the second most fatal disease found in women worldwide. The estimated new cases of breast cancer for 2015 alone have been approximately 231,000, with deaths related to breast cancer estimated at 40,000. Current statistics show that 1 out of every 10 women will develop breast cancer in their lifetime, with the risk increasing as women age. Early detection and treatment of breast cancer is important and a major public health issue, especially since 90% of patients can be saved by early diagnosis (“What are the Key...”, 2015).

Cancer, aside from being one of the major causes of mortality in the world, is a complex and heterogeneous disease and research in the diagnosis and treatment of it has become one of the most important issues in the science community. In recent years, factors from different tests have been identified as possible indicators of disease progression in breast cancer and its prognosis but the analysis of this information by medical experts is prone to error or often cannot be analyzed fast enough. Radiologists usually study various features to determine between the two types of tumors – benign, a non-cancerous tumor that will not spread, or malignant, a cancerous tumor. Unfortunately, 10-30% of breast cancer lesions are usually misdiagnosed due to the limitations of human observers (Mahjabeen et al., 2012). False negatives result in malignant tumors that are misdiagnosed and later detection reduces the chance of survival. A false positive is something to be avoided as well since it results in unnecessary and costly surgical procedures that would not have needed to be undertaken in the first place. Identifying women at risk is an important strategy in reducing the number of women suffering from breast cancer and ensuring they get appropriate treatment early on.

Within the last decade, there have been major advancements in the methods that diagnose breast cancer that reduce the need of the human expert. Neural network based clinical support systems are beginning to be sought after to provide medical experts with a second opinion and removing the need for a biopsy or an excision and reducing unnecessary expenditures for patients. New innovations and methods in early detection as well as the prediction of recurrent cancer are all crucial for the survival of the patient. Recent use of Artificial Neural Networks has given accurate results for the diagnosis of breast cancer and the stage of the tumor (Mahjabeen et al., 2012). By continuing to improve on machine learning methods with higher accuracy rates for the diagnosis of breast cancer, these methods cannot only be implemented into diagnosing breast cancer, but can also expand into the diagnosis of other diseases.

Central Issues and Questions

Over the years, scientists have used machine learning techniques to address the problem of diagnosing breast cancer and analyzing the data available, but the problem has been difficult to solve since the data set has been relatively small and noisy. Detecting the probability of recurrence of cancer or identifying women at risk is important in getting treatment to women on time.

Mammography and ultrasonography continue to remain as the principal breast imaging tests for early detection, but both have limitations, have high costs, and usually require an experienced pathologist to read the results (Naguib, 1998). Mammography is sensitive but not specific at detecting breast cancer, resulting in close to 65% of cases being referred to surgical biopsy when they are actually benign. Smarter systems are needed in order to get rid of false positives and false negatives in order to prevent these unnecessary surgical procedures and reduce the cost (Lo, 1999).

Computer aided diagnosis has, for the last decade, been proposed for medical diagnosis, with fuzzy logic and Artificial Neural Networks forming the basis of these intelligent systems. There are several studies and published papers in which artificial intelligence has been used in the diagnosis as well as the prognosis and even recurrence probability of breast cancer. Different Artificial Neural Network architectures have been studied such as: Convolution Neural Networks, Radial Basis Networks, General Regression Neural Networks, Probabilistic Neural Networks, Backpropagation Neural Networks and hybrid with fuzzy logic (Mahjabeen et al., 2012).

In one study, a supervised Artificial Neural Network was used to classify breast lesions into either being malignant or benign. It processed computer cytology images, looking for four biomarkers and the relationships between them. In this study, the four biomarkers that were looked at were DNA ploidy, phase fraction, cell cycle distribution, and the state of steroid receptors. The accuracy of this trained neural network was 82.21%, higher than previously used methods such as logistic regression and establishing Artificial Neural Networks as a robust system for the diagnosis of breast cancer (Mahjabeen et al., 2012).

Two neural network approaches were proposed in a different study aimed at breast cancer diagnosis. The first approach was based off of evolutionary Artificial Neural Networks and involved a feed forward neural network evolved using an evolutionary programming algorithm. The weights and architectures were evolved in the same process, making the network either grow or shrink. The second approach was based off of neural network ensembles, a number of Feedforward Neural Networks trained at the same time to solve the breast cancer diagnosis problem cooperatively using a divide-and-conquer method. The evolutionary approach worked well for breast cancer, but has been found to be too computationally costly to turn these into larger neural networks using more data (Xin, 1999).

In another study that used Artificial Neural Networks as an aid for diagnosis, four models were implemented to aid in the diagnosis of the Wisconsin Breast Cancer Diagnosis (WBCD) Dataset. These four models were: Backpropagation Algorithm, Radial Basis Function Network, Learning Vector Quantization and a Competitive Learning Network. Although the Learning Vector appeared to show the best performance on the dataset used, the Backpropagation Algorithm showcased a higher generalizing capability (Janghel et al, 2010).

In a similar project created by Brittany Wenger in 2012, an artificial neural network was created to see if it would optimize and improve the success of breast cancer diagnosis through the use of fine needle aspirates. Wenger used three commercial neural networks along with the Wisconsin Breast Cancer Dataset (WBCD) and compared the results against her own custom neural network, which achieved a predictive success of 97.4% with a 99.1% sensitivity to malignancy (Wenger, 2012).

Based off of these studies, and various others it has been found that the use of Artificial Neural Networks increases the accuracy of most of the methods and reduces the need of a human expert, who may not be available at all times. A clinical support system based off of neural networks might aid in providing experts with a second opinion and reducing unnecessary expenditures for patients. Artificial Neural Networks, though, must be optimized according to the specific problem (Mahjabeen et al., 2012).

In this thesis, I will use Artificial Neural Networks to diagnose breast cancer using instances taken from fine needle aspirates (FNA) of human breast tissue, each of which consist of nine measurements, and classify these samples as either benign or malignant in the hopes of achieving an accuracy rate higher than previous studies have achieved. The dataset will come from the Wisconsin Breast Cancer Dataset (WBCD) database, created by Dr. William H. Wolberg. To set it apart from previous research using these similar methods, I will be using different functions for the Neural Network and will analyze the data to see if it can be optimized for analysis through the ANN. The following questions are what I will focus on in my study.

What is the best Neural Network approach for diagnosing breast cancer using fine needle aspirates? In this paper I have decided that the use of a Multilayer Artificial Neural Network which uses the Backpropagation Algorithm to train weights will be used. Based off of previous research, this seems like the likeliest Artificial Neural Network architecture to yield the highest accuracy rate.

Are there other Neural Networks that may have better performance than a Feedforward Network? After analyzing the data and results, I plan to implement three other machine learning methods to compare and see which attains a higher performance. The other three machine learning methods I plan to implement include the Radial Basis Function, the NEAT Network, and a Support Vector Machine.

Background

Artificial Neural Networks

Artificial Neural Networks (ANNs) refer to learning models inspired by the biological neural networks of the brain. The study of this machine learning technique stemmed from ‘a desire to understand the basic functionality of the human brain’ (“Introduction to...”, 2000). Artificial Neural Networks are often represented as a system of interconnected neurons with weights that are adjusted as the network is trained, which makes neural networks adaptive and gives them the ability to learn. As shown in Figure 1, input values are inserted into an artificial neuron where they are multiplied by the weights and added to find out the weighted sum of the artificial neuron. Depending on what function is being used, this weighted sum is then inserted into a function so that a value can be given to the neuron.

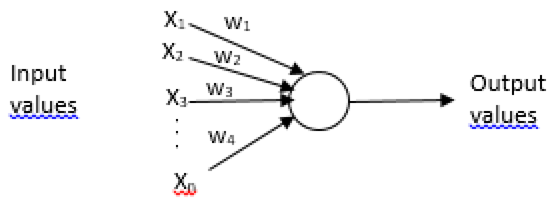

Figure 1. Example of a simplified artificial neuron showcasing the input value, output value, and the weights.

The drive behind developing an artificial neural network that simulates a biological neural network came about from a desire to understand how a human brain behaves. The first attempt at modeling this behavior of neurons was in the 1940’s and was hardware based with the prediction, by researchers, that the digital computer would eventually become available and incorporated into their model (“Introduction to..”, 2000).

The Perceptron Algorithm was first introduced by Rosenblatt in 1962 in a book called *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Perceptrons served as a simplified network that incorporated weights as a memory mechanism that allowed them to learn responses to stimuli in various types of experiments (Widrow, 1990).

In 1969, Marvin Minsky and Seymour Papert published a book titled *Perceptrons* that talked about the limitations of the perceptron Rosenblatt had previously developed. This resulted in a reduction in the amount of research that went into neural networks for nearly a decade until John Hopfield published a paper in 1982 on an associative neural network, often referred to as the Hopfield Network that reintroduced Artificial Neural Networks and their potential for success and applications ("Introduction to..", 2000). Since then, the field of Artificial Neural Networks in machine learning began to expand.

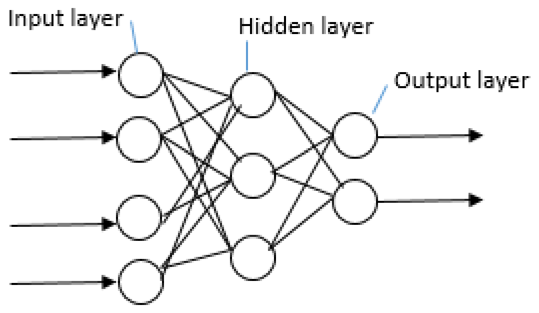

Figure 2. Example of a Feedforward Neural Network displaying the input layer, hidden layer, output layer, and the interconnectivity between the three layers.

Feedforward networks were soon introduced. These networks were termed as such since the input moves forward, or left to right, through the network towards the output layer, as shown in Figure 2. The first major addition to feed forward networks beyond Madeline, one of the earliest trainable layered neural networks with adaptive elements, was introduced by Paul Werbos when he developed a backpropagation algorithm that was published as his doctoral dissertation in 1974. His work remained unknown until another researcher rediscovered and incorporated the backpropagation algorithm into a report published in 1986 (Widrow, 1990).

Feedfoward networks today contain multiple layers, all of which are usually adaptive and adjust as a network is trained. Backpropagation networks are the best examples of multilayer networks and are considered as one of the most influential techniques in the field of neural networks due to the simplicity of the concept. Although many variations of the Feedforward algorithm are possible, Rumelhart's backpropagation technique is often used since, aside from its simplicity, it has proven to be effective (Widrow, 1990). The backpropagation technique differed from earlier architectures by using only differentiable functions on the elements and consists of a network of randomly assigned weights trained by propagating forward and generating outputs, which are then compared to target outputs to calculate error values at each node and adjust the weights in the network. This process is repeated until the weights are trained to generate an output similar to the target value.

The relationship between biological and artificial neurons has become more important, especially in recent years and, like other machine learning methods, Artificial Neural Networks can be used to solve a wide variety of tasks that may be harder to solve via other means ("Introduction to..", 2000).

Multilayer Neural Networks and Backpropagation Algorithm

In this project, Multilayer Neural Networks, such as Feedforward Neural Networks, will be used with a Backpropagation Algorithm for learning purposes. Multilayer networks are often used to learn tasks with a learning algorithm, such as backpropagation. There are many functions available that can be used as the activation function for calculating values at nodes in the layers but in this project, sigmoid functions will be used due to their similarities to the functions in perceptron units and differentiable properties. The activation function must be differentiable in order for the backpropagation algorithm to work correctly.

Functions in the nodes, or neurons, take the weights between nodes, often noted asw_{ij}, the weight between node i and node j, and add the weights of the nodes connected to it.

$$\sigma(S) = \frac{1}{1+e^{-S}}$$

The sigmoid function, as shown above, calculates the value of the weighted sum, S, and when the output values of the sigmoid function are plotted, the curve resembles a step function. The output values of this function will range between 0 and 1, each respectively symbolizing a neuron either firing or not firing (Colton, 2004).

Since the output for this network flows in only one direction, as shown in Figure 2, this type of network is referred to as a Feedforward Network. As explained before, weights are set on the connections between the nodes and sigmoid units exist in both the hidden layers and the output layer, which are found by using a sigmoid function on the weighted sums. The weighted sums are calculated by multiplying the weights (w) by the inputs (x), adding the different values that go into a node, $S = \sum_n x_i\, w_{ij}$, and using that value in the sigmoid function to find the node's output. Those nodes that are fired will result in a value closer to 1 while those not fired are usually closer to 0. Based off of these calculations, the nodes fired can be found and can be added and used as inputs to the next layer until the output layer is reached.

The Backpropagation Algorithm is then used to train the weights. As had been previously mentioned, w_{ij} is used to specify the weight between unit i and unit j. The change in weight, Δ_{ij}, is then calculated and added on to each weight after an example has been tried. To calculate the weight change, Δ_{ij}, target values must be specified that each output node should ideally produce. The input is sent through the network and the observed values can be recorded and compared to the target values. For each output unit, the error term is calculated using:

$$\delta O_k = O_k(E)(1 - O_k(E))(t_k(E) - O_k(E))$$

where O_k is the observed value of the output node and $t_k$is the target value.

The error terms for the output units are used to calculate the error terms for the hidden units, whose values are also written down since the weighted sum of each of the nodes in these layers also goes through the sigmoid function to obtain a value (Colton, 2004).

$$\delta O_k = h_k(E)(1 - h_k(E)) \sum_{i \in outputs} w_{ki}\, \delta o_i$$

The error term for every output unit is taken and multiplied by the weight of the hidden unit, h_k. These are then added and multiplied by the sum. Once all the error values are known, the weight changes, Δ_{ij}, can be calculated (Colton, 2004).

For weights w_{ij} between the input unit and hidden unit, and for the weight between the hidden unit and output unit, we add on:

$\Delta_{ij} = n\delta H_j x_i$, for the weight between the input unit and hidden unit and

$\Delta_{ij} = n\delta O_j h_i(E)$, for the weight between the hidden unit and output unit.

The change in weights $\Delta_{ij}$is added to the previous weights and the new weights are then kept and, in the next iteration are tweaked even further. As the network runs through all of the training examples, often referred to as epochs, the weights are trained and the values change until a network is achieved that, when tested, produces ideal results based off the data that is being analyzed.

CHAPTER II

METHODS

Experimental Setup

This project was inspired by previous experiments that dealt with Artificial Neural Networks and how effective they were in the diagnosis of breast cancer. In these experiments, researchers either created their own Artificial Neural Networks and compared them against currently available neural network software (Wenger, 2012) or compared different various machine learning methods against each other to see which would be the best at classifying the dataset used (Osareh, 2010).

In a study by Osarch and Shadgar, two datasets, one of fine needle aspirate breast lesions and another comprised of gene microarrays, were used to compare three supervised learning algorithms: k-Nearest Neighbors algorithm, Support Vector Machine and Probabilistic Neural Network (Osareh, 2010). The three machine learning algorithms were applied to diagnose breast cancer using the top three ranked features of dataset the fine needle aspirate dataset and to evaluate the prognosis of breast cancer in the second dataset. The performance of these three classifiers were then evaluated into three separate categories - sensitivity, specificity, and overall accuracy - and compared to see which of the three produced the best overall accuracies (Osareh, 2010). I will be conducting a similar experiment but with a focus on a Feedforward Neural Network with Backpropagation training methods. Using this machine learning method, I will see how well it can classify the fine needle aspirate dataset as well as compare it against three other machine learning techniques: Radial Basis Function Networks, NEAT Networks, and Support Vector Machines.

Encog

For this experiment I will be using Encog, a machine learning framework made available for Java, .Net and C++. Encog supports a variety of machine learning methods, advanced algorithms, and includes classes that aid in normalizing and processing data. The first version of Encog was created by Jeff Heaton in 2008 and has since grown due to an active community that has provided improvements and fixes to the libraries (Heaton, 2015).

One of the main goals of Encog, and reasons as to why I chose to use it, was to allow users to switch between different neural networks with ease. Different machine learning models, algorithms and training methods were created to be interchangeable. This allows for a programmer to experiment with various machine learning models and creates a good way to allow for finding a model that best fits the data to be classified or predicted (Heaton, 2015).

Structure

Encog uses the same structure as a typical neural network, with an input pattern that gets accepted and an output pattern that is then returned. The input layer is the first layer in the neural network and contains a specific amount of neurons (usually one neuron for every attribute the neural network will use, though some attributes may need more than one neuron).

The output layer is the final layer of the neural network and is provided after all the previous layers are processed. Hidden layers are inserted between the input and the output layers and can take on more complex structures, depending on the type of neural network that will be used. Hidden layers are used to provide a layer of neurons that can apply a function to try and produce the expected output of a certain input at a better accuracy.

Neural network programming in Encog is the same as in any other program - first, the input and output must be defined and the amount of neurons must be specified. Hidden layers are defined once the programming problem has been figured out, though the challenge is to avoid creating a hidden structure that becomes either too complex or too simple. A structure that is too complex will take too long to train and a hidden layer that is too simple will not help the problem (Heaton, 2015).

In the Feedforward network I used, the hidden layer had neurons equal to the amount of input and output neurons times 1.5. A good starting point, as described by Heaton, is a single hidden layer with a number of neurons equal to twice the input layer, with the amount of neurons in the hidden layer either increasing or decreasing depending on the performance of the network (Heaton, 2015). The amount of hidden layers has also been one of much debate, though research has so far shown that a second hidden layer (or more) are rarely any help to the overall network.

Data

Encog provides a variety of classes for analyzing, normalizing and importing datasets since, as previously stated, its main function is to be able to fit various machine learning methods. Among these, MLData, MLDataPair and MLDataSet are included, which were used in the neural network code of my program. MLDataSet is used to create new datasets while the MLData class is used to define array-like data that will be used by the Encog software. The BasicMLData class implements the MLData interface and provides a memory-based data holder for the neural network data that is returned from the network's output layer once the input has been processed by the network and which is deleted at the end of the training and testing. These classes take both arrays and .CSV files which must then be normalized depending on the activation function to be used (Heaton, 2015).

Normalizing Data

Encog's neural networks accept floating point numbers as their inputs, within the range of -1 to +1 or 0 to +1, depending on the choice of activation function since some activation functions have only a positive range while others have both a positive and negative range.

The sigmoid activation function only has a range of positive numbers, while the hyperbolic tangent activation function has a range of positive and negative numbers. Although the most common case is the hyperbolic tangent activation function for Encog, for my experiment I decided to use the sigmoid activation function (Heaton, 2015).

Encog provides built in classes to both normalize and denormalize data as it is trained and tested. Encog also has an Analyst class that can be used to normalize data already stored in a csv file and denormalize it before the output is shown, as well as provide some statistical information such as the mean, standard deviation and low and high numbers of a dataset.

Activation Sigmoid

For my experiment, the activation sigmoid function will be used which means the output expected is a positive number and the inputs will be normalized to fall between the ranges of 0 to +1. The equation for the activation sigmoid function can be seen in the following equation.

$$f(x) = \frac{1}{1+e^{-x}}$$

If negative numbers were to be found in the input, the activation sigmoid function would move the negative numbers into the positive range. This can be seen in Figure 3 which shows the graph of a sigmoid function.

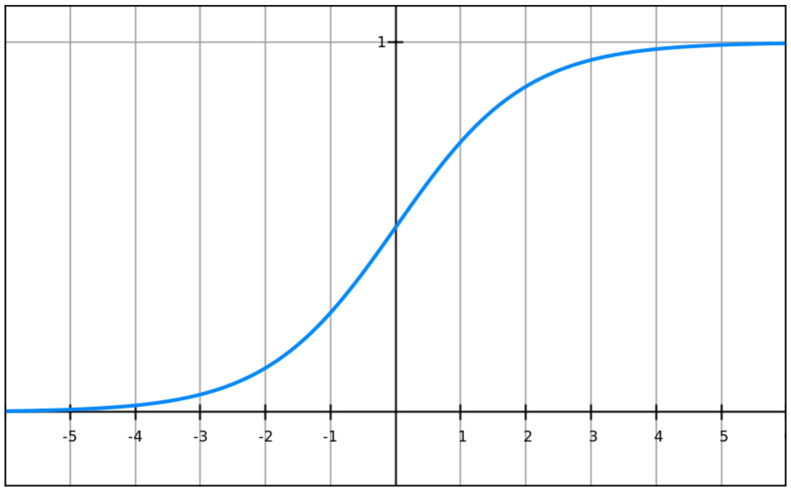

Figure 3. Graph of a sigmoid function.

Propagation Training

The propagation training algorithm to be used in Encog uses supervised training, which means that a training set of inputs and ideal outputs is given to the training algorithm. The training algorithm then goes through various iterations to improve the neural network's error rate, the difference between the actual output from the neural network and the ideal output in the training data. Each iteration loops through the same training data and a change to the weights is calculated. At the end of each iteration, Encog updates the weight matrix values. For each training iteration, a two pass process is executed: a forward pass and a backward pass – that are then looped over all of the training elements (Heaton, 2015).

The forward pass presents the data to the neural network as it would if no training had occurred and the algorithm calculates the error between the output and the ideal output. The backward pass then starts from the output layer and works its way back towards the input layer. It examines the differences between the ideal and actual outputs from each of the neurons, changing the weights as it goes at each neuron.

Machine Learning Methods

Machine learning methods in Encog descend from the MLMethod class. The three most important are MLRegression, MLClassification and MLClustering, though for the purposes of my experiment I will be focused on classification.

To train a neural network in Encog, the MLDataSet object is constructed with the inputs and expected outputs. The data is called from a CSV file and then normalized so that it can be used by the activation function. Encog will automatically detect which the correct normalization type based on the activation function is chosen in the last step. For model validation, 30% of the data is specified to be held back and 70% is used for training. The training type and machine learning type is then selected, and a 10-fold cross validated technique is used to choose the model with the best validation score before a test is run with the chosen model. More detail on the Encog program will be described in the code section.

Dataset

For this project, I used a publicly available dataset, the Breast Cancer Wisconsin (BCW) dataset from the UCI Machine Learning Repository. The dataset consists of fine needle aspirates of breast lesions and contains 699 specimens, with samples arriving periodically over three years. The dataset grew out of a desire by Dr. William H Wolberg to accurately diagnose masses based solely on Fine Needle Aspiration (FNA) samples and was originally obtained from the University of Wisconsin Hospitals, Madison (Wolberg, 1992).

He identified nine visually assessed nuclear features of an FNA sample (Figure 4) considered relevant to diagnosis and gave each attribute a value between 1 and 10 depending on the state, 1 corresponding to a normal state and 10 to an anaplastic state. The following attributes were the ones identified: clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, and mitoses. Analysis showed that the nine characteristics differed significantly between benign and malignant samples. The sample code number was removed when running the data since it does not have an effect on the outcome. Sixteen instances of missing values occur, but were able to be replaced using Encog.

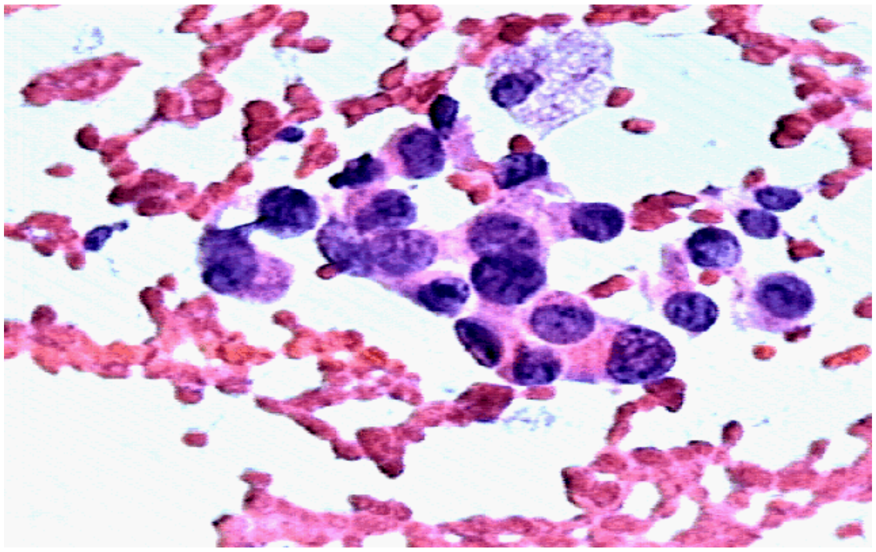

Figure 4. Fine Needle Aspirate sample analyzed and given attributes for BCW dataset (Wolberg, 1992)

The dataset contains 241 positive samples (malignant) and 458 negative samples (benign). For analysis purposes, the dataset was converted into a .CSV file to be read by the program. The first column of ID’s was removed, since it was irrelevant. The rows with missing values were also removed to get more accurate results.

Classification Methods

The main focus of this experiment were Artificial Neural Networks using Feedforward learning methods with Backpropagation training on a dataset of fine needle aspirate samples, classifying them into either benign or malignant depending on their attributes. Three other classification methods were also used in this experiment solely to compare how Feedforward Networks stacked up against them. These include Support Vector Machines, a NEAT Neural Network, and a Radial Basis Function Network.

Support Vector Machines

Support vector machines (SVM) were proposed by Vapnik et al., based on a statistical learning theory which was effectively used as an algorithm to deal with classification and regression problems (Osareh, 2010). The SVM method is essentially a discriminative classifier defined by a separating hyper plane. This method relies on labeled training data, since it is a supervised learning method, which is preprocessed to represent patterns. These patterns are then separated by the hyper plane to categorize new examples. The goal of training a SVM is to find the optimal separating hyper plane that maximizes the margin of the training data, as shown in Figure 5. The larger the margin the better the generalization of the classifier is usually expected to be.

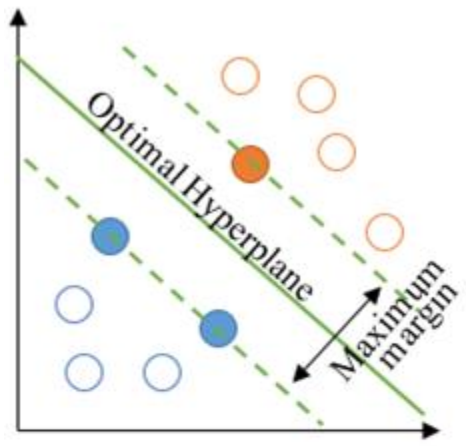

Figure 5. Optimal hyper plane at the maximum distance from both training patterns that a Support Vector Machine aims to achieve.

The hyper plane is constructed as (Osareh, 2010):

$$f(x) = < w, x > + b$$

where x is the feature vector, w is the vector that is perpendicular to the hyper plane and b specifies the offset from the beginning of the coordinate system. SVM's tend to be less prone to overfitting than other methods (Duda, 2001).

A simple method of training a SVM is based on a modification of the perceptron training rule, which requires updating the weight vector by an amount proportional to any misclassified pattern ("Introduction to Support..", 2016).

Radial Basis Function Network

A Radial Basis Function (RBF) network is an artificial neural network that uses radial basis functions as activation functions. RBF networks were first formulated by Broomhead and Lowe in 1988 (Orr, 1996). RBF networks can be used for function approximation, time series prediction, system control, and classification (Orr, 1996).

Radial Basis Function networks usually have three layers: an input layer, a hidden layer with an RBF function, and an output layer. Functions that end on the distance from a center vector are radially symmetric about that vector, hence the name radial basis function. In the basic form, all inputs are connected to each hidden neuron (Bullinaria, 2004).

RBF Networks are essentially a two-layer Feedforward networks with hidden nodes that implement a set of radial basis functions, such as a Gaussian Function, a Multi-Quadratic Function, or even a Cubic or Linear Function. The output nodes implement a linear summation function similar as the Multilayer Perceptron.

RBF networks are typically trained using a two-step algorithm. In the first step, the center vectors of the RBF function in the hidden layer are chosen, which means the weights from the input layer to the hidden layer are determined, and then the weights from the hidden to the output layer are determined in the second step (Bullinaria, 2004).

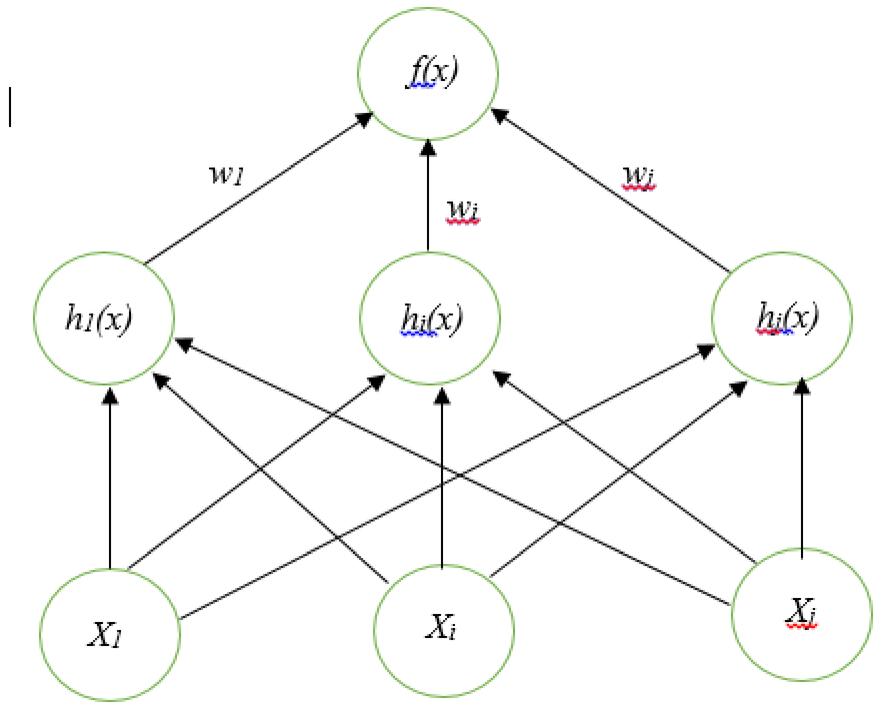

Figure 6. Shows a traditional radial basis function network. Each of the components of vector x feed towards the m basis functions, whose outputs are linearly combined with weights into the network output f(x) (Bullinaria, 2004).

NEAT Network

The NeuroEvolution of Augmenting Topologies (NEAT) is a genetic algorithm developed in 2002 by Ken Stanley at the University of Texas at Austin. NEAT works by altering the weight parameters and structures of the networks in an attempt to find a balance between the fitness of evolved solutions and their diversity. In other words, it relieves the neural network programmer of trying to find the optimal structure of a neural network's hidden layer (Heaton, 2015). By tracking genes with history markers to allow crossover among topologies, applying speciation to preserve innovations, and developing topologies incrementally from simple initial structures it arrives at an effective network more quickly than other neuro-evolutionary techniques and reinforcement learning methods (Stanley, 2002).

A NEAT neural network has input neurons and output neurons, much like a perceptron-like feed-forward network. A NEAT network begins with one input and output layer, as can be seen in Figure 7, and, as evolution progresses through discrete steps, the complexity of the network grows, either by inserting new neurons into a connection path or by creating a new connection between unconnected neurons, as can be seen in Figure 8. These connections inside of the NEAT neural network can be feedforward, recurrent, or self-connected (Heaton, 2015).

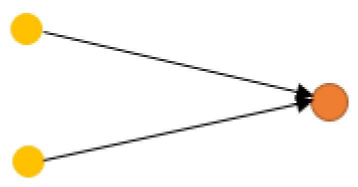

Figure 7. Shows the NEAT network before evolution.

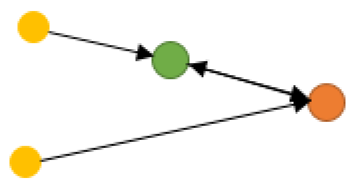

Figure 8. Shows how a NEAT network grows the complexity of its network by inserting a new neuron and creating new connections.

There is very little difference between the code needed to use the NEAT Neural Network and Feedforward Network in Encog since, as was stated previously, the core objective of Encog is to allow interchangeability between different machine learning methods.

Experimental Design

For this experiment, I decided to utilize the Netbeans IDE and Java language to write the neural network program using the Encog libraries. The libraries were imported as a .jar file into the project folder, and the main application was then coded using the different libraries available by calling the different methods and classes.

Libraries

The following are the Encog libraries that were imported and called in the java neural network project.

```
import org.encog.ConsoleStatusReportable;

import org.encog.Encog;

import org.encog.ml.MLRegression;

import org.encog.ml.data.MLData;

import org.encog.ml.data.versatile.NormalizationHelper;

import org.encog.ml.data.versatile.VersatileMLDataSet;

import
org.encog.ml.data.versatile.columns.ColumnDefinition;

import org.encog.ml.data.versatile.columns.ColumnType;

import org.encog.ml.data.versatile.sources.CSVDataSource;

import
org.encog.ml.data.versatile.sources.VersatileDataSource;
```

```
import org.encog.ml.factory.MLMethodFactory;
import org.encog.ml.model.EncogModel;
import org.encog.util.csv.CSVFormat;
import org.encog.util.csv.ReadCSV;
import org.encog.util.simple.EncogUtility;
```

All of the files imported were used in the java program and were called in order to ensure functionality. For example the ConsoleStatusReportable() class was imported to allow the output reports to go to the console (Heaton, 2015).

The MLData() class is used to define an array of data and the NormalizationHelper() class is used to perform the normalizations on methods trained as a versatile dataset, which is why the VersatileMLDataSet() is used. This class supports several advanced features, such as reading and normalizing directly from a CSV file and is also necessary if the EncogModel() class is used. The CSVDataSource() class allows for a CSV file to serve as a source for the VersatileDataSet(), while the ColumnDefinition() and ColumnType() classes serve to define which column and what type of data is in the column. The CSVFormat() class is used to distinguish between the separator, whether it is a decimal comma or a decimal point, while the ReadCSV() class reads and parses a CSV file (Heaton, 2015).

The MLMethodFactory() class is used to easily interchange between various machine learning such as Bayesian Neural Networks, Feedforward Neural Networks, Neat Neural Networks, Probabilistic Neural Networks, Radial Basis Function Neural Networks, and Support Vector Machines. The EncogModel() class is what allows the interchange between different neural networks to occur and also aids in normalizing the data, depending on the activation function that corresponds to a particular neural network (Heaton, 2015).

Importing the Dataset

The following code starts off the program by importing the CSV file and setting it as our source.

```
public class BCClassification {

    public void run(String[] args)

      try {

          File dbcFile = new File ("dbcdata2.csv")

          VersatileDataSource source = new

          CSVDataSource(dbcFile, false,

          CSVFormat.DECIMAL_POINT);

          VersatileMLDataSet bcdata = new

          VersatileMLDataSet(source);
```

As can be seen from the code, the project starts off by importing the csv file and making sure the program reads it using the VersatileDataSource() class. This class indicates the name of the file, whether there are any boolean headers, and the csv format. The VersatileMLDataSet() object is defined and used to load our dataset and make it available for the MLMethodFactory() class to use later on.

Specifying the Dataset

```
     bcdata.defineSourceColumn("clump thickness", 0,
     ColumnType.continuous;

     bcdata.defineSourceColumn("uniformity of cell size", 1,
     ColumnType.continuous);

     bcdata.defineSourceColumn("uniformity of cell shape", 2,
     ColumnType.continuous);
```

```
bcdata.defineSourceColumn("marginal adhesion", 3,
ColumnType.continuous);

bcdata.defineSourceColumn("single epithelial cell size", 4,
ColumnType.continuous);

bcdata.defineSourceColumn("bare nuclei", 5,
ColumnType.continuous);

bcdata.defineSourceColumn("bland chromatin", 6,
ColumnType.continuous);

bcdata.defineSourceColumn("normal nucleoli", 7,
ColumnType.continuous);

bcdata.defineSourceColumn("mitoses", 8,
ColumnType.continuous);

ColumnDefinition outputC =
bcdata.defineSourceColumn("classification", 9,
ColumnType.nominal);
```

In this next part, each column in the dataset is given a name and index, since they did not have a column header in the CSV file. As can be seen, we start off the index at 0. The defineSourceColumn() contains the name to be given to the column, the index of the column, and the column type. Four variations exist that could be assigned to our column type: continuous, ignore, nominal, or ordinal. For this code, continuous was chosen since it is used for continuous non-discrete values, similar to a double value in Java, and nominal for our output column, which specifies a discrete nominal and is used to specify a class.

Analyzing the Dataset

The analyze() method reads the entire file and determines the minimum, maximum, mean and standard deviations for each column. These statistics can be useful for both normalization and for replacing missing values.

```
bcdata.analyze();
```

Declaring the Machine Learning Method

```
bcdata.defineSingleOutputOthersInput(outputC);
 EncogModel network = new EncogModel(bcdata);
network.selectMethod(bcdata,
MLMethodFactory.TYPE_FEEDFORWARD);
 network.setReport(new ConsoleStatusReportable());
```

This next part specifies the model type that is to be used. It starts by specifying what the output column is, hence why the output column earlier was distinguished in the code when the columns were defined. EncogModel() is designed to allow the interchange between different model types and automatically normalizes data. It is designed to work with VersatileMLDataSet() only. The selectMethod() class specifies what the data is and is followed by what type of machine learning type we plan on using. Different types are available to switch between, such as NEAT Neural Network (TYPE_NEAT), Probabilistic Neural Networks (TYPE_PNN), Support Vector Machines (TYPE_SVM), and RBF Neural Networks (TYPE_RBF). ConsoleStatusReportable() sends any output to the console and reports on current status of the method (Heaton, 2015).

Normalizing the Data

The following line normalizes the data and allocates the memory to hold the normalized data. Due to the sigmoid function, the data should fall between 0 and +1.

```
bcdata.normalize();
network.holdBackValidation(0.3, true, 1001);
```

The next line holds back some of the data in our CSV file for final validation. It shows that 30% of our data is being held for testing, while the other 70% is used for training. The ‘true’ stands for boolean shuffle, and the last number indicates the seed for random generation.

Cross-validating Data

```
network.selectTrainingType(bcdata);
MLRegression chooseBest =
(MLRegression)network.crossvalidate(10, true);
```

This part of the code uses a ten fold cross-validated training and returns the best method found out of the ten. The ‘10’ indicates the amount of folds, and ‘true’ means that shuffling will occur. Cross validation is a model validation technique for assessing results of statistical analysis. It is primarily used in prediction with the goal of defining a dataset to ‘test’ the model in the training phase and limit problems of overfitting. The original sample is randomly partitioned into different sized samples. Of the different subsamples, only a single subsample is retained as validation data for testing the model. The cross validation process is then repeated k times with each of the k subsamples used exactly once as the validation data. 10 fold is commonly used but in general k remains an unfixed parameter.

```
System.out.println("Training error: " +
 EncogUtility.calculateRegressionError(chooseBest,
 network.getTrainingDataset()));

System.out.println("Validation error: " +
 EncogUtility.calculateRegressionError(chooseBest,
 network.getValidationDataset()));

NormalizationHelper normdata = bcdata.getNormHelper();

System.out.println(normdata.toString());
```

This is used to call the normalization parameters and display them to the console. Although not necessary, the mean, standard deviation, and highs and lows could be useful when analyzing the dataset. The validation error and training error are also displayed onto the console window.

Training and Testing

```
ReadCSV csv = new ReadCSV(dbcFile, false,
CSVFormat.DECIMAL_POINT);

String[] line = new String[9];

MLData input = normdata.allocateInputVector();

int count=0;

int correct=0;

int ben=0;

int mal=0;

int totalben=0;

int totalmal=0;

   while(csv.next()) {

     StringBuilder results = new StringBuilder();

     line[0] = csv.get(0);
```

```
line[1] = csv.get(1);

line[2] = csv.get(2);

line[3] = csv.get(3);

line[4] = csv.get(4);

line[5] = csv.get(5);

line[6] = csv.get(6);

line[7] = csv.get(7);

line[8] = csv.get(8);

String correctOutput = csv.get(9);

normdata.normalizeInputVector(line,input.getData(),fal
se);

MLData output = chooseBest.compute(input);

String predictedOutput=
normdata.denormalizeOutputVectorToString(output)[0];

results.append(Arrays.toString(line));

results.append(" -> predicted: ");

results.append(predictedOutput);

results.append("(correct: ");

results.append(correctOutput);

results.append(")\r");

System.out.println(results.toString());

Count++;

int m = Integer.parseInt(correctOutput);

int b = Integer.parseInt(predictedOutput);

if (b == m){

correct++;
}

if (4==m && b==m){

mal++;

}

if (2==m && b==m){

     ben++;

     }

     if (4==b){

     totalmal++;

     }

     if (2==b){

     totalben++;

     }

     double percentage =
((double)correct/(double)count)*100;

     System.out.println("Total Tested: "+count);

     System.out.println("Total Correct: "+correct);

     System.out.println("Overall accuracy:
"+percentage+"%");

     System.out.println("Sensitivity:"+(double)mal/(do
uble)totalmal* 100+"%");

     System.out.println("Specificity:"+(double)ben/(do
uble)totalben*100+"%");
```

This next part opens the CSV file and loops through the data, predicting the output of each sample and using the best model and normalization helper. The normalization helper allows to normalize and denormalize the data as it loops through so the dataset values viewed on the console are not the normalized ones but the ones from the original data file. The total number of rows (or loops) tested is then displayed in the console (under Total Tested) along with whether the data produced the correct result or not. Overall accuracy of the classification, sensitivity, and specificity are also shown and were placed into the program to easily grab the data. Sensitivity, in this case, refers to the total amount of correctly classified malignant (4) samples over the total amount of malignant samples there actually were. Specificity refers to the total amount of samples classified as benign samples (2) over the actual amount of benign samples that existed within the dataset.

```
dbcFile.delete();
Encog.getInstance().shutdown();
```

Finally, the program deletes the dbcdata file that had been saved to memory (the normalized data and denormalized data) and shuts down.

Neural Network Architecture

The architecture of the Feedforward Network used in this program can be found in the FeedForwardConfig.java file. From here, I changed the activation function from TANH (indicating the hyperbolic tangent activation function) to SIGMOID (for the sigmoid activation function), and the normalization numbers to go from 0 to 1. Other possibilities included changing the amount of hidden neurons as well as layers, though I decided to leave the amount of hidden neurons at the suggested number of the sum of output and input neurons multiplied by 1.5. Previous tests showed that too low an amount of neurons did not aid in obtaining a higher accuracy for classification and a higher amount of neurons made the network too slow to perform well, resulting in more iterations than were needed. Momentum was already set at 0.7. The training method was changed as well from resilient propagation to backpropagation training method.

```
public String suggestModelArchitecture(VersatileMLDataSet
dataset) {

      int inputColumns =
dataset.getNormHelper().getInputColumns().size();

      int outputColumns =
dataset.getNormHelper().getOutputColumns().size();

      int hiddenCount = (int)
((double)(inputColumns+outputColumns) * 1.5);

      StringBuilder result = new StringBuilder();

      result.append("?:B->SIGMOID->");

      result.append(hiddenCount);

      result.append(":B->SIGMOID->?");

      return result.toString();
```

The code shows the Encog code for the Feedforward Network Configuration and how the basic architecture of the neural model is structured, with the input and output columns grabbed from the CSV file, while the hidden layer contains 1.5 times that of the input and output neurons combined. It also shows how the activation functions are being specified and called.

Results

The results for the following four networks, Feedforward networks, NEAT Networks, Support Vector Machine, and RBF Networks, were gathered from the console. The overall accuracy was taken by calculating the total amount of rows correctly predicted over the total amount of rows tested.

$$Overall\ accuracy\ =\ Total\ Correct/Total\ Tested$$

Specificity and sensitivity were also used to evaluate the performance of the classifiers and were defined as follows:

$$Sensitivity\ =\ Predicted\ Malignant/Correct\ Malignant$$

$$Specificity\ = Predicted\ Benign/Correct\ Benign$$

Sensitivity could also be described as the total positives over the total positives and false negatives, while specificity could be described as the total negatives over the total negatives and false positives.

Feedforward Networks

The Feedforward Network was the first Neural Network that I tested. As previously stated, for this neural network I used the sigmoid activation function for the layers, an input of 9 neurons, an output of 1 neuron, and only one hidden neuron layer with fifteen neurons. A momentum of 0.7 was used as well to avoid overfitting. 10-cross fold validation was also used to select the best model from among those trained. The following is an example of what the console displayed.

```
1/10 : Fold #1

1/10 : Fold #1/10: Iteration #1, Training Error: 1.95338433,
Validation Error: 1.88607197

1/10 : Fold #1/10: Iteration #2, Training Error: 0.65475619,
Validation Error: 1.88607197

1/10 : Fold #1/10: Iteration #3, Training Error: 0.50751008,
Validation Error: 1.88607197

...

10/10 : Fold #10/10: Iteration #41, Training Error: 0.07384639,
Validation Error: 0.05168873

10/10 : Fold #10/10: Iteration #42, Training Error: 0.07332578,
Validation Error: 0.06249483

10/10 : Cross-validated score:0.11516842228163807

Training error: 0.05999059427124292

Validation error: 0.09192842693193816

...

[2, 1, 1, 1, 2, 1, 1, 1, 1] -> predicted: 2(correct: 2)

[5, 10, 10, 3, 7, 3, 8, 10, 2] -> predicted: 4(correct: 4)

[4, 8, 6, 4, 3, 4, 10, 6, 1] -> predicted: 4(correct: 4)

[4, 8, 8, 5, 4, 5, 10, 4, 1] -> predicted: 4(correct: 4)

Total Tested: 683

Total Correct: 671

Overall accuracy: 98.24304538799414%

Sensitivity: 95.95141700404858%

Specificity: 99.54128440366972%
```

In the output, the number of iterations each fold takes, as well as the cross validated score with the final training and validation error were displayed on the console. The crossfold validation is useful since it compares the model against data it was not trained with, giving a more accurate result as to how the network might work when presented with new data. Once the crossfold validation was done and a model was chosen, the testing began on the data and went through each row, showing the denormalized data for each row, as well as the predicted and correct results.

Overall, the total correct out of the total amount of rows tested resulted in an accuracy of 98.24%. The sensitivity was also recorded and had a percentage of 95.95% and a specificity of 99.54%. The cross validated score was 0.12, with a training error of 0.06 and a validation error of 0.09.

As can be seen in the table, the each fold has a different amount of iterations that it took for it to achieve a goal training and validation error.

Table 1. The iterations, training error values, and validation error values for each of the ten cross - fold validations for Feedforward Neural Networks.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Iterations	24	48	18	18	24	30	24	18	12	42
Training Error	0.08	0.06	0.09	0.08	0.07	0.07	0.06	0.08	0.09	0.07
Validation Error	0.11	0.02	0.08	0.12	0.12	0.09	0.17	0.21	0.18	0.06

SVM Results

Much like the Feedforward Neural Network results, the SVM results were displayed on the console except without iterations. Each fold gave a training error and validation error, and at the end a cross-validated score, total training error and validation error were given.

```
10/10 : Cross-validated score:0.04297112462006079
Training error: 0.04384133611691023
Validation error: 0.049019607843137254
```

The different rows and the predicted output versus the correct output were also displayed with the following results:

```
Overall accuracy: 95.46120058565154%
Sensitivity: 96.84684684684684%
Specificity: 94.79392624728851%
```

As can be seen by the output, the overall accuracy was lower than that of the Feedforward Neural Network's result. Support Vector Machines attained an overall accuracy of 95.46%, with a sensitivity of 96.85% and a specificity of 94.79%.

Table 2. The training error values and validation error values for the ten cross-fold validation of Support Vector Machine Network.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Training error	0.04	0.04	0.04	0.05	0.05	0.05	0.04	0.05	0.04	0.05
Validation error	0.09	0.09	0.04	0.0	0.02	0.02	0.09	0.02	0.06	0.0

RBF Network Results

The Radial Basis Function Network went through several iterations at each fold. This algorithm obtained the following results.

```
10/10: Cross-validated score:0.1855936294354464
Training error: 0.11573472522050637
Validation error: 0.12460479104242392
Overall accuracy: 93.1185944363104%
Sensitivity: 98.0%
Specificity: 91.09730848861284%
```

The RBF Network obtained an overall accuracy of 93.12%, the lowest out of the four different methods, though it did achieve a sensitivity of 98.0%. The RBF Network also went through the highest amount of iterations, as can be seen in Table 3.

Table 3. The iterations, training error value and validation error values for each of the ten cross-fold validations for the Radial Basis Function Network.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Iterations	36	54	138	24	48	36	30	54	126	36
Training error	0.21	0.19	0.15	0.19	0.21	0.11	0.18	0.16	0.16	0.19
Validation error	0.23	0.20	0.20	0.18	0.24	0.10	0.15	0.18	0.18	0.20

NEAT Results

For the NEAT network, a low amount of iterations were needed to achieve the training error and validation error goals (Table 3), though the overall accuracy was still below that of the Feedforward Networks at 96.63%.

```
10/10 : Cross-validated score:0.043723725771640534
Training error: 0.035645061169585435
Validation error: 0.05620576258222121
Overall accuracy: 96.63250366032212%
Sensitivity: 93.54838709677419%
Specificity: 98.39080459770115%
```

The cross-validated score was one of the lowest, though, as was the total error and the validation error for the training model, indicating that with a new dataset, this method might outperform the Feedforward Neural Networks.

Table 4. The iterations, training error values and validation error values for the ten cross-fold validation of the NEAT Neural Network

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Iterations	25	13	19	13	19	25	13	31	25	13
Training Error	0.04	0.05	0.04	0.03	0.03	0.03	0.04	0.04	0.04	0.03
Validation Error	0.06	0.02	0.05	0.06	0.02	0.02	0.12	0.01	0.05	0.03

Comparison

Table 5. The overall accuracy, specificity and sensitivity of the four Neural Networks: Feedforward Network, NEAT Network, RBF Network, and SVM Network. Overall accuracy was taken by dividing the total correct over the total tested. The specificity was calculated by dividing the amount of correct benign predicted over the total amount of benign predicted, and the sensitivity was calculated as the total amount of correct malignant classifications over the amount of malignant predictions.

	Overall Accuracy	**Specificity**	**Sensitivity**
Feedforward	98.24%	99.54%	95.95%
NEAT	96.63%	98.39%	93.55%
RBF Network	93.12%	91.10%	98.0%
SVM	95.46%	94.79%	96.85%

As can be seen in Table 5, Feedforward Networks achieved the best overall accuracy at 98.24%, with a 99.54% specificity and a 95.95% sensitivity. Although the results were high on specificity, sensitivity and the data that falls into that category could be analyzed in order to see what is making it be lower than the specificity. NEAT Networks performed second best, with a 96.63% accuracy, a 98.39% specificity, and a 93.55% sensitivity. It is possible that with some adjustments to the algorithm in the code, this network could perform better.

The Radial Basis Function Network performed at an overall accuracy of 93.12%, with a high sensitivity of 98% but rather low specificity of 91.10%. The low specificity indicates that a large amount of benign samples are being classified as malignant. Further analysis of the data and algorithm used for the RBF Network in Encog could provide a higher specificity rate. The Support Vector Machine Network performed at a 95.46% overall accuracy, with a specificity and sensitivity that did not deviate too much from each other. Similar to the previous two networks, analysis of the algorithm and methods used for this method in Encog, as well as going through the data could allow some insight into how to get a higher accuracy for this method.

Table 6. The cross-validated scores, total error values, and validation error values of the best training model from each of the four networks: Feedforward Neural Network, NEAT Neural Network, RBF Network and SVM Network.

	Cross-Validated Score	**Total Error**	**Validation Error**
Feedforward	0.12	0.06	0.09
NEAT	0.04	0.04	0.05
RBF Network	0.18	0.12	0.12
SVM	0.04	0.04	0.06

In this table, the cross-validated scores are compared between the different networks. Cross-validated scores are useful in being able to tell what networks may get a higher accuracy when tested with new data since the data used after training is data that has not previously been introduced to the networks. The total error and the validation error are also shown. As can be seen, the RBF Network, also the network with the lowest overall accuracy, got the worst cross-validated score, total error, and validation error. The SVM Network did slightly better than the NEAT network, though they were close in terms of which had the better cross-validated score.

The Feedforward Network seems to not have performed as well during training as it did during testing, which poses the question of how well it would do with new data. Further testing would be required to see whether the network would perform just as well or worse with new data and an analysis as to why the cross-validated score and the validation error came out low would have to be done.

CHAPTER III

CONCLUSION AND FUTURE WORK

Conclusion

In this thesis I focused on investigating how Feedforward Networks worked and how well they would be at classifying the fine needle aspirate dataset. Using the Encog library, I was able to build a program that allowed me to run various tests, as well as compare the Feedforward Network to three other type of networks: Support Vector Machine, Neat Neural Networks, and RBF Neural Networks (although these were left as is without any customization). First, the program was built using the Encog libraries, the dataset was imported, normalized, and trained. Although the dataset could have been normalized easily seeing as the numbers ranged from one to ten already, it was interesting to see how the Encog software normalized the dataset (though the range had to be changed from -1 to +1 to 0 to +1 since the activation functions were changed). Once the program was created and initiated, the amount of iterations it took for the dataset in to train was noticeably lower compared to other softwares that I had tested, such as Neuroph, a Java neural network framework. Encog required an average of about thirty iterations per fold. Crossfold validation was used in this experiment to aid in choosing the best model for testing. Results at the end showed Feedforward Neural Networks having the best overall accuracy at 98.24%, but with a crossfold validation score of 0.12, lower than NEAT networks and SVM Networks.

The experiments also showed how Feedforward Neural Networks compared against SVM, Neat Networks, and RBF Networks. Whether it was due to the dataset that was used, or the fact that the Feedforward Neural Network was altered from the basic Encog library, the Feedforward Neural Network consistently managed better results than the other three methods in terms of classification.

Future Work

The focus of this project was to get some insight into how Artificial Neural Networks work, how they analyzed data, and how accurate they were at predicting the ideal output for certain data. I was able to observe how a Feedforward Network performed, as well as how it performed against three other networks although more in depth research is necessary into the other three methods since I barely touched the surface on understanding how they work.

Network Customization

Continuing to test the Feedforward Neural Network, such as how different learning rates affect the outcome, how different layers affect the output, and how different amount of neurons may affect the output would be something to continue to research. Changing these as well as the maximum error rate may have an impact on the total accuracy and could prove interesting results and better understanding of neural networks, at least in terms of how many layers may actually be useful for a Feedforward Network and how many hidden neurons should a hidden layer have.

Machine Learning Methods

Comparing the Feedforward Network to other networks, but more in detail would also be of value. Not just using a basic library for the other three, but actually creating and customizing the other three neural networks to best fit the data would create more competitive neural networks and may even show that a different type of Machine Learning Method is better suited for this type of classification than a Feedforward Neural Network.

Image Recognition

As was stated previously in the paper, the dataset came from analysis of fine needle aspirate samples. These analysis were done from samples of cells under a microscope that a cytologist had to examine. Cytologists can be costly, and being able to automate this analysis would aid the fine needle aspirate tests and bring down their cost even further. Implementing not only Encog, but a library geared towards image recognition, such as Caffe or Torch7 would allow to see whether a different software may be more suitable for this type of task.

This work is only a step into Artificial Neural Networks and how they work, as well as the research being done to understand neural networks better and how to utilize them. Here I performed experiments with Feedforward Neural Networks to better understand how to implement neural networks and how they work, as well as how accurate they were at classification. Comparisons were done to see whether Feedforward Networks really were the most suitable method for this type of dataset. Further research and future investigations are needed to continue to explore neural networks, their implementation, and how to best utilize them.

REFERENCES

Bullinaria, John A. (2004). Radial Basis Function Networks: An Introduction [Pdf]. Retrieved from http://www.cs.bham.ac.uk/~jxb/NN/l12.pdf.

Colton, Simon. "Multi-Layer Artificial Neural Networks." N.p., 2004. Web. 04 Aug. 2015.

Duda, R. O., Hart, P. E., & Stork, D. G. (2000). *Pattern Classification* (2nd ed.). New York, US: John Wiley & Sons.

Heaton, J. (2015). Encog: Library of Interchangeable Machine Learning Models for Java and C#. *Journal of Machine Learning Research*, 16, 1243-1247.

"Introduction to artificial neural networks," *Electronic Technology Directions to the Year 2000, 1995. Proceedings.* , vol., no., pp.36,62, 23-25 May 1995.

Janghel, R.R.; Shukla, A.; Tiwari, R.; Kala, R., "Breast cancer diagnosis using Artificial Neural Network models," Information Sciences and Interaction Sciences (ICIS), 2010 3rd International Conference on , vol., no., pp.89,94, 23-25 June 2010.

Lo, J.Y.; Floyd, C.E., Jr., "Application of artificial neural networks for diagnosis of breast cancer," *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on* , vol.3, no., pp.,1759 Vol. 3, 1999.

Mahjabeen Mirza Beg and Monika Jain, "An Analysis of the methods employed for Breast Cancer Diagnosis," International Journal of Research in Computer Science, vol. 2, no. 3, pp. 25-29, 2012.

Mumtaz, K.; Sheriff, S.A.; Duraiswamy, K., "Evaluation of three neural network models using Wisconsin breast cancer database," *Control, Automation, Communication and Energy Conservation, 2009. INCACEC 2009. 2009 International Conference on* , vol., no., pp.1,7, 4-6 June 2009.

Naguib, R.N.G.; Mat-Sakim, H.A.; Lakshmi, M.S.; Wadehra, V.; Lennard, T.W.J.; Bhatavdekar, J.; Sherbet, G.V., "Logistic regression in the analysis of image cytometric data of fine-needle aspirated cells of breast cancer patients-a comparison with artificial neural networks," *Engineering in Medicine and Biology Society, 1998. Proceedings of the 20th Annual International Conference of the IEEE* , vol.2, no., pp.982,985 vol.2, 29 Oct-1 Nov 1998.

OpenCV DevTeam. (n.d.). Introduction to Support Vector Machines. Retrieved July 10, 2016, from http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction _to_svm.html

Orr, M. J. (1996). Introduction to radial basis function networks.

Osareh, A.; Shadgar, B., "Machine learning techniques to diagnose breast cancer," *Health Informatics and Bioinformatics (HIBIT), 2010 5th International Symposium on* , vol., no., pp.114,120, 20-22 April 2010.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, *10*(2), 99-127.

Street, W. N., Olvi L. Mangasarian, and William H. Wolberg. "Machine Learning for Cancer Diagnosis and Prognosis." *Machine Learning for Cancer Diagnosis and Prognosis*. N.p., n.d. Web. 04 Aug. 2015.

"What Are the Key Statistics about Breast Cancer?" *Cancer.org*. American Cancer Society, 10 June 2015. Web. 4 Aug. 2015.

Widrow, B.; Lehr, M.A., "30 years of adaptive neural networks: perceptron, Madaline, and backpropagation," *Proceedings of the IEEE*, vol.78, no.9, pp.1415, 1442, Sep 1990.

Xin Yao; Yong Liu, "Neural networks for breast cancer diagnosis," *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol.3, no., pp., 1767 Vol. 3, 1999.

Zevarac, Zoran, Ivan Goloskokovic, Jon Tait, Laura Carter-Greaves, Aidan Morgan, and Valentin Steinhauer. *Java Neural Network Framework Neuroph*. Computer software. *Java Neural Network Framework Neuroph*. Vers. 2.9. N.p., n.d. Web. 04 Aug. 2015.

Wenger, Brittany. https://sites.google.com/a/googlesciencefair.com/science-fair-2012-project-64a91af142a459cfb486ed5cb05f803b2eb41354-1333130785-87/home.

Wolberg, William H. "Breast Cancer Wisconsin (Original) Data Set." *UCI Machine Learning Repository*. UCI, 15 July 1992. Web. 4 Aug. 2015. <https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Original)>.

BIOGRAPHICAL SKETCH

Janette Vazquez was born December 1988 in Mission, Texas. She attended Leo Marcell Elementary from August 1994 to May 2001. From August 2001 to May 2003 she attended Kenneth White Jr High in Mission, Texas, and then went on to attend the High School for Health Professionals (Med High) from August 2003 to May 2007. In August of 2007 she enrolled at the University of Texas-Pan American (UTPA) and in 2011 she received her Bachelor of Science in Biology, with a minor in Chemistry. In August 2013 she once again attended the University of Texas-Pan American (UTPA) to complete a Master’s of Science in Computer Science, during which time UTPA transitioned into the University of Texas Rio Grande Valley (UTRGV). She earned her Master’s of Science in Computer Science in 2016. She currently resides at 4003 Zulley St, Mission, TX 78573.