Supervised Image Classification Using an Artificial Neural Network for Optical Digit Recognition and Diagnosis of Fine Needle Aspirates of Breast Cancer

Somil Govani

Abstract

Artificial Neural Networks (ANN) are machine learning models loosely based on the biological structure of the brain. In the past, they have shown success in approximating solutions for multivariate prediction and classification problems, including stock market analysis and the Travelling Salesman Problem (TSP). In this investigation, the goal was to build an extensible, multipurpose ANN for the analysis and classification of images. The neural network was built using an object-oriented approach in Python with an input layer, a hidden layer, and an output layer with variable numbers of neurons per each layer. A logistic sigmoid function was implemented in order to transform the propagated values, and the quasi-Newton Conjugate Gradient algorithm was implemented in order to perform batch gradient descent to minimize the cost of the ANN.

In order to test the viability of this neural network, it was applied for the recognition and classification of optical handwritten digits. An application was built in order to collect supervised pixel maps and images of optical digits, which were then used to train the neural network. Ultimately, the neural network was able to identify test samples of digits with an accuracy of 82.4% across 1560 trials; however, the ANN did show some signs of overfitting. Multiple techniques were employed to overcome overfitting including early stoppage and regularization, although it is hypothesized that an increased training sample size will diminish overfitting and increase accuracy.

Furthermore, this ANN was also applied for the categorical diagnosis of fine needle aspirates (FNA) of breast cancer tumors. Supervised biopsy data was acquired from the UCI Machine Learning Database consisting of 32 attributes compiled from each FNA image. The neural network was able to diagnose test samples as either malignant or benign with an accuracy of 97% across 715 trials with no signs of overfitting. Future endeavors include investigations into more applications as well as more convoluted, deep neural networks.

Table of Contents

1. Introduction

In computation, linear or limited variable problems often have straight-forward and low-cost approximate solutions. These problems often entail an algorithmic approach in which the scope of the solution is already well-defined. However, in numerous machine learning problems, the scope of the solution is not as well-defined, and the structural algorithmic approach must be replaced by an adaptable one instead. In this way, computers would be able to develop solutions that the developer hasn't already had the foresight to predict.

In particular, humans are generally both efficient and accurate in their approach to multivariate problems. For example, after having seen many examples of what a "tree" would look like in their lives, an individual would be able to observe the numerous attributes of any object and decide whether or not to classify it as a "tree." However, in contrast to the power of computation, human learning and evaluation speeds of new and foreign problems are relatively slow.

An Artificial Neural Network (ANN) is a machine learning model loosely based on the biological structure of the brain, aimed to emulate and expedite the learning process observed in humans and other animals. In the past, ANNs have shown considerable success in solving other multivariable learning problems, including stock market predictions, computer vision, and finding approximate solutions to the Travelling Salesman Problem (TSP). Neural networks have also shown success in developing intelligences that can perform image classification and have shown particular success in optical character recognition software. In general, the layout of a neural network is specific to the problem at hand. However, the design of a neural network is algorithmic in itself and can be made to be extensible across a discrete classification of problem sets.

In particular, ANNs success in supervised image classification offers a promising field of exploration. One potentially implementable form of this is with Optical Digit Recognition of handwritten characters in order to convert handwritten text into computer-type. This would allow computers to compile handwritten text into easy to access and searchable documents. Furthermore, ANNs would help remove human error, learning cost, and lack of efficiency in higher-caliber tasks. In particular, neural networks could help diagnose and analyze medical images including but not limited to biopsies, x-rays, MRIs, and CAT scans. With larger and larger sets of data, not only would this make diagnoses more accurate and efficient, but it also considerably lower the cost of diagnosis and make it more affordable. According to statistics from Cancer Research UK, more than 90% of women diagnosed with breast cancer at the earliest stage survive their disease for at least 5 years compared to around 15% for women diagnosed with the most advanced stage of disease. This could potentially save lives if terminable diseases such as cancer are able to be detected early.

1.1 Goal

The goal of this research was to develop a general-purpose, extensible artificial neural network for the supervised analysis and classification of images and to apply this ANN for

Optical Digit Classification and categorical diagnosis of fine needle aspirates of breast cancer tumors.

2. Methods

This project was programmed with an object-oriented approach using the Python programming language using the pygame, scipy, numpy, and pillow libraries.

2.1 Developing an Extensible ANN

The neural network framework. The first step of this project is to develop the extensible neural network, able to be applied to any number of classification problems. The canonical three-layered neural network was used (see **Figure 1** below), containing the input layer, hidden layer, and output layer. In this particular approach, only one hidden layer was implemented, as prior methods showed that deeper or more convoluted neural networks did not entirely benefit neural networks to a great extent in proportion to computational cost. The number of neurons in the input and output layers were constants dependent on the nature of the training data. The number of neurons in the hidden layer is also dependent on the training data, but is determined using iterative testing discussed later on. The value holding nodes in each layer and the connections between them will be referred to as neurons and synapses respectively in correlation to the terminology of brain structure.

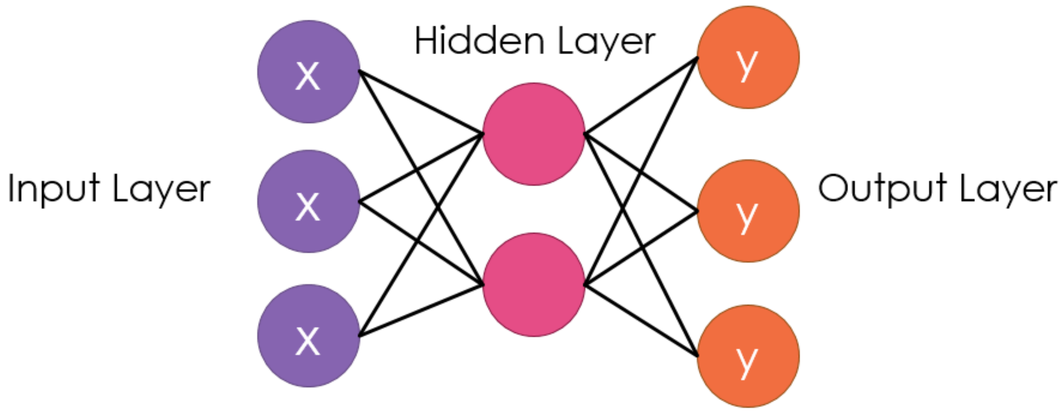


Figure 1. General framework for the three-layered neural network

This is the constructor used to initiate Neural_Network class as previously described:

```
class Neural_Network(object):
    #Intialize neural network object (requires length of square image)
    def __init__ (self, iLayer=2, oLayer=10, hLayer=10, Lambda=0):
        self.inputLayerSize= iLayer
        self.outputLayerSize = oLayer
        self.hiddenLayerSize = hLayer

        self.W1 = np.random.randn(self.inputLayerSize, self.hiddenLayerSize)
        self.W2 = np.random.randn(self.hiddenLayerSize, self.outputLayerSize)
        self.Lambda = Lambda
```

2.1.1 Forward propagation

ANNs operate on the basis of giving the inputs varying weights or biases in determining the nature of the outputs. This process occurs as illustrated in **Figure 2** below.

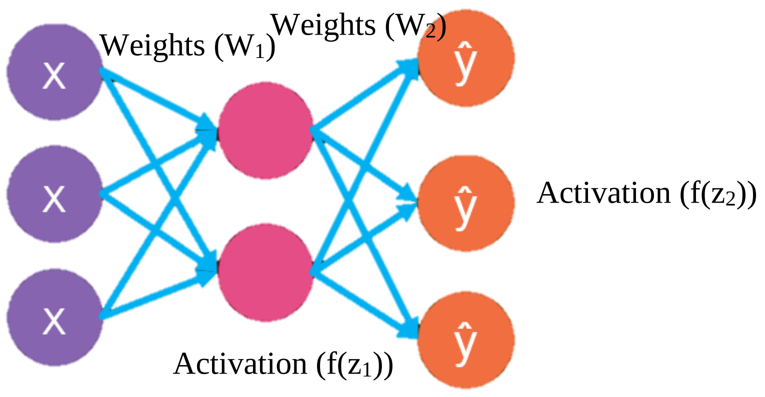


Figure 2. Forward propagation illustrated in the neural layout

Data begins in the input layer, where it is inputted. From here, a weight is applied to each input and the resulting values are summed at each neuron of the hidden layer. As it is seen in the above constructor, the number of weights between two layers is dependent on the number of neurons in each of the two endpoint layers because each neuron in the initial layer is connected to each neuron in the subsequent layer. Because these weighted inputs are being summed at each neuron of the hidden layer, it is useful to treat this operation as a matrix multiplication in order to simplify notation. At each subsequent neuron (after the input layer), an activation function is applied to the received value, which becomes the new input. This process is repeated until values reach the output layer and there are no more synapses to traverse. In the case of a simple ANN, as it is in the case of this implementation, this process is only repeated two fold, but an algorithmic implementation can easily be extended for deeper neural networks.

The general forward propagation model can be defined by the following series of equations, in which X is the input matrix, W_1 and W_2 are the weight matrices, f(z) is the activation function (which will be later discussed), and $\hat{y}$ is the predicted output:

$$z_1 = XW_1 \quad [1]$$

$$a = f(z_1) \quad [2]$$

$$z_2 = aW_2 \quad [3]$$

$$\hat{y} = f(z_2) \quad [4]$$

$$\hat{y} = f(f(XW_1)W_2) \qquad [5]$$

Equation 5 above is the composed version of its predecessors. These series of equations was used to implement the forward propagation method in code:

```
def forward(self, x):
        self.z2 = np.dot(x, self.W1)
        self.a2 = self.sigmoid(self.z2)
        self.z3 = np.dot(self.a2, self.W2)
        yHat = self.sigmoid(self.z3)
        return yHat
```

Sigmoid Activation Function. The role of the activation function in Equations 2 & 4 is to introduce non-linearity to the forward propagation of an ANN. In the absence of an activation function, any weighted ANN could easily be condensed to a single layer neural network that is linear in nature. However, the purpose of the activation function is to allow non-linearity so multivariate learning can occur. Based on this criteria, a the fairly elementary step function shown in **Figure 3** below would be appropriate:

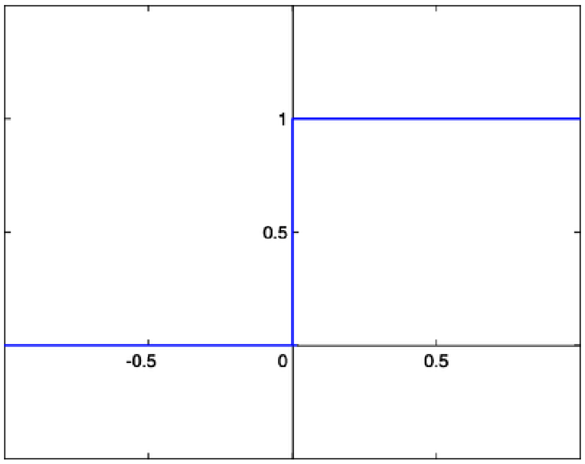


Figure 3. Step function in range 0 to 1

However, this function poses a potential issue because it is non-differentiable. Later on, it will be necessary to compute a gradient for optimization involving the activation function. In this case, it would be more useful to use a curved sigmoid function. For this implementation, the logistic sigmoid function was used (see **Figure 4** below for the graphical and mathematical representation of the function and its derivative), although other options such as hyperbolic tangent are popular ones.

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

$$f'(z) = \frac{e^{-z}}{(1 + e^{-z})^2}$$

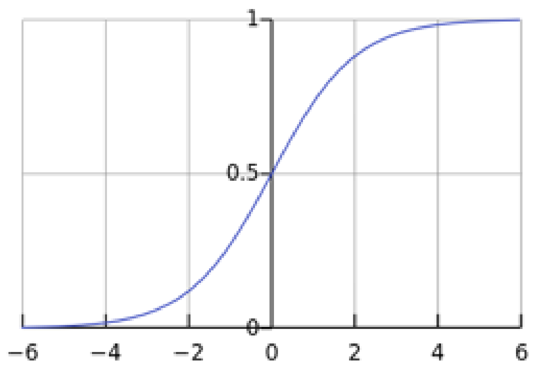


Figure 4. The logistic sigmoid activation function

2.1.2 Backwards Propagation

As it was evident in the initial constructor of the Neural_Network class, the weight matrices were initiated with random biases. As a result, the predicted outputs will be equally as random. Supervised backpropagation aims to tune these weights based on the supervised data used for training the network in order to minimize the error of the predicted outputs so as to "learn" how to classify the inputs. See **Figure 5** for the barebones model.

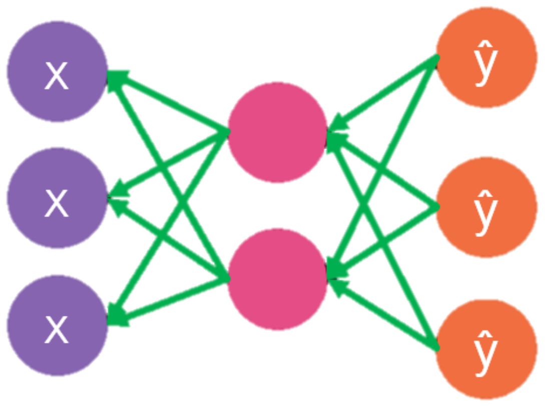


Figure 5. Backwards propagation model

Cross Entropy Cost Function. In order to tune the biases accordingly, the error of the predicted outputs must first be quantified. One approach to this problem is to calculate the variance of each predicted output like so (where N is the total number of training data):

$$\frac{1}{N}\sum(y - \hat{y})^2 \qquad [6]$$

However, although this approach works considerably well for continuous predictions, it falls short in supervised classification for a number of reasons. This is so because in classification problems, the outputs more likely represent Booleans of 0 and 1 or -1 and 1 using 1-of-N encoding (will be discussed later). In this sense, each datum can only have one correct classification, however the mean-squared cost function above puts a disproportionate emphasis on the error of the incorrect classifications whereas it is most important to note the error on the correct classifications. Thus, the better solution for supervised categorical problems such as image classification is the cross-entropy cost function:

$$J = -\frac{1}{N}\sum_{i=0}^{n} y_n \ln(\hat{y}_n) + (1-y_n)\ln(1-\hat{y}_n) \qquad [7]$$

The first term in this summation computes the error on the output neuron that has been supervised as the ‘correct’ neuron for classification (the neurons supervised with 1s). The second term does the same for the neurons that have been deemed ‘incorrect.’ However, in the second term, the cross entropy function takes the natural log of 1 minus the predicted output and thus minimizes the significance of the incorrect output neurons as this is a single-classification problem. The code for the cross entropy function is like so (ignore the regularization constant for now):

```
def cost(self, x, y, outPut=False, test=False):
    self.yHat = self.forward(x)
    if outPut:
        print self.yHat
    J = (-1.0/len(x)) * sum(sum(y * np.log(self.yHat)
        + (1-y)*np.log(1-self.yHat)))
    #Regularizes cost function to prevent overfitting
    regularize = (self.Lambda/2.0/len(x)) * (sum(sum(self.W1**2))
        + sum(sum(self.W2**2)))
    if test:
        regularize = 0
    return J + regularize
```

Computing the Gradient. In order to minimize the multivariate cost function, the following gradient must be considered:

$$\nabla J = \left(\frac{\partial J}{\partial W_1}, \frac{\partial J}{\partial W_2}\right) \qquad [8]$$

This gradient can be extended to include all weight matrices up to and including W_n, where n is the number of hidden layers in the neural network plus one. In order to compute this gradient, the partial derivative of the cost function in respect to the weights must be computed. It is more straight forward to begin with the derivative in respect to W_2, as per the *backwards* propagation model.

Begin with the cross entropy cost previously defined in Equation 7 and take the derivative in respect to the W_2 matrix. (The range of the summation is temporarily ignored for readability).

$$J = -\frac{1}{N}\sum y_n \ln(\hat{y}_n) + (1-y_n)\ln(1-\hat{y}_n) \qquad [7]$$

$$\frac{\partial J}{\partial W_2} = -\frac{1}{N}\sum \frac{y}{\hat{y}} * \frac{\partial \hat{y}}{\partial W_2} + \frac{1-y}{1-\hat{y}} * \frac{-\partial \hat{y}}{\partial W_2} \qquad [9]$$

Factor out the partial derivative of y-hat in respect to W_2 and apply the chain rule of calculus and the equation relationships defined in Equations 1-4 to arrive at:

$$\frac{\partial J}{\partial W_2} = -\frac{1}{N}\sum \left[\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right] \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{\partial W_2} \qquad [10]$$

The next step is to is to evaluate the internal partial derivatives in Equation 10 using Equations 1-4 and simplify the expression within the brackets.

$$\frac{\partial J}{\partial W_2} = -\frac{1}{N}\sum \left[\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right] f'(z_2)a$$

$$\frac{\partial J}{\partial W_2} = -\frac{1}{N}\sum \left[\frac{y-\hat{y}}{\hat{y}(1-\hat{y})}\right] f'(z_2)\, a \qquad [11]$$

From here, using the definition of $\hat{y}$ given in Equation 4 and the following identity:

$$f'(z) = \frac{-e^{-z}}{(1+e^{-z})^2} = \frac{1}{(1+e^{-z})} * \left(1 - \frac{1}{1+e^{-z}}\right) = f(z)(1-f(z)) \qquad [12]$$

It is evident that Equation 11 will simplify to the following expression like so

$$\frac{\partial J}{\partial W_2} = \frac{1}{N}\sum (\hat{y} - y)\, a \qquad [13]$$

Although the subscript *n* and range of the summation was temporarily omitted for readability during this derivation, it is important to note that y and $\hat{y}$ represent matrices of all their respective terms rather than singular values. Thus, the summation can be accounted for by using matrix multiplication by multiplying by a transposed version of the matrix *a* by the difference between matrices y-hat and y.

$$\frac{\partial J}{\partial W_2} = a^T \cdot \frac{\hat{y} - y}{N} \qquad [14]$$

Thus, Equation 14 yields the final expression for the partial derivative of cost in respect to the second matrices of weights.

Computing a similar derivative for weights closer to the input layer of the neural network (in this case W_1 being the only one to fit this criteria) follows an almost algorithmically similar derivation. The steps are the same except for the application of the chain rule in Equation 10, which requires further regression through Equations 1-4 in order to break it down appropriately, resulting in this equation:

$$\frac{\partial J}{\partial W_1} = -\frac{1}{N}\sum\left[\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right]\frac{\partial \hat{y}}{\partial z_2}\frac{\partial z_2}{\partial a}\frac{\partial a}{\partial z_1}\frac{\partial z_1}{\partial W_2} \qquad [15]$$

Finally, using identical simplification methods as per the prior derivation, the summation must once again be accounted for through transposed matrix multiplication.

$$\frac{\partial J}{\partial W_1} = -\frac{1}{N}\sum\left[\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right] f'(z_2) * W_2 * f'(z_1) * X \qquad [16]$$

$$\frac{\partial J}{\partial W_1} = X^T \cdot \frac{(\hat{y}-y)}{N} \cdot {W_2}^T \cdot f'(z_1) \qquad [17]$$

Thus, Equation 17 yields the final derivation for the partial derivative of the cost function in respect to the second set of weights. In more convoluted networks (more hidden layers), it is plain to see that a similar pattern persists. Thus, a more algorithmic/recursive approach may be used in order to back propagate through multiple hidden layers in the case of a deeper neural network. Nonetheless, the gradient of the cost of the neural network is computed with the following function:

```
#Compute derivative of cost function
    def costPrime(self, x, y):

        self.yHat = self.forward(x)
        backError2 = (y-self.yHat)/(-float(len(x)))
        dJdW2 = np.dot(self.a2.transpose(), backError2)
            + (self.Lambda*self.W2)/(len(x))
        backError1 = np.dot(backError2, self.W2.transpose())
            * self.sigmoidPrime(self.z2)
        dJdW1 = np.dot(x.transpose(), backError1)
            + (self.Lambda*sum(sum(self.W1)))/(len(x))
        return dJdW1, dJdW2
```

Minimizing cost using Batch Gradient Descent and the Conjugate Gradient Algorithm. In order to minimize the cost (error) of the neural network's predicted outputs, the computed gradient is used to update the weight matrices in the negative direction. This batch gradient descent method will approximately approach a minimum. According to prior investigations into batch gradient descent, the minima approached in this algorithm are generally absolute at very large variable counts. In this neural network implementation, the scipy library is used in order to minimize the cost function. The previously computed gradients pose a significantly lesser computational cost than if an iterative derivative approximation approach was taken. In particular, the Conjugate Gradient algorithm was used in order to minimize the cost function. The Conjugate Gradient method is a quasi-newton algorithm often used to minimize gradients with large sets of variables.

2.2 Training

In order for any neural network framework to carry out the previously explored supervised backwards propagation method, supervised training data is necessary. In conjunction with training data, backpropagation of neural networks is referred to aptly as *training*, as this is the stage where the ANN ‘learns’ to classify its data set, essentially by trial and error. For this stage, the following train method was implemented into the Trainer class, and it implements the batch gradient descent using Conjugate Gradient that was aforementioned.

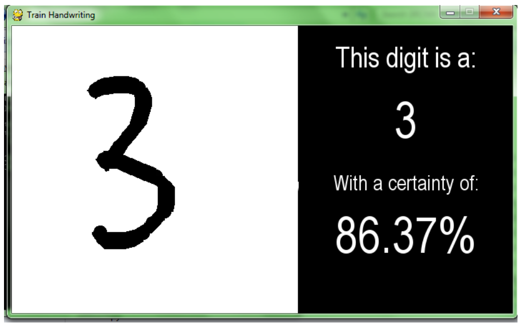
```
def train(self, trainX, trainY, testX, testY):
        #Make an internal variable for the callback function:
        self.X = trainX
        self.y = trainY

        self.testX = testX
        self.testY = testY

        #Make empty list to store training costs:
        self.J = []
        self.testJ = []

        params0 = self.N.getParams()

        options = {'maxiter': 100, 'disp' : True}

        #Minimize cost function using computed gradient method
        _res = optimize.minimize(self.costFunctionWrapper, params0, jac=True,
method='CG',
                                 args=(trainX, trainY), options=options,
callback=self.callbackF)
        self.N.setParams(_res.x)
        self.optimizationResults = _res
```

As a proof of concept to the now built ANN, the following two training sets were implemented and tested.

2.2.1 Optical Digit Recognition

Optical digit recognition is an open problem due to its potential benefits in eBook digitization and computer interaction with handwritten text. In this application, the ANN is trained with supervised pixel maps of optical digit samples of the numbers from 0-9.

Gathering pixel map data. Rather than using available databases containing compiled pixel maps of various handwritten digits, an application was developed in order to both train the neural network and to forward propagate predicted outputs using the currently gathered pixel maps. The application was developed using pygame and can be seen in **Figure 6** below.



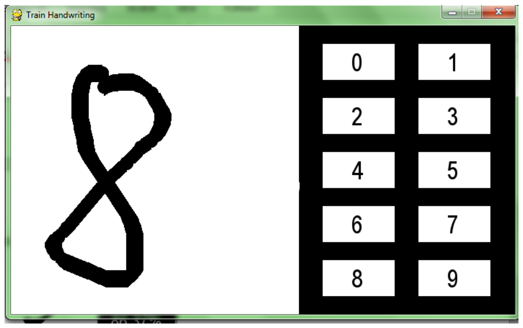


Figure 6. Pygame Application to Train (Optical) Handwriting

The left-hand image in **Figure 6** shows an already trained network analyzing the character drawn on the white canvas using the mouse. The right-hand image in **Figure 6** shows a user training the neural network by supplying it with a drawn digit on the canvas along with a supervision through the digit selection pad on the right side of the window. When gathering the digit training data, it is important to note that it was only gathered from one source (rather than a database or multiple sources) in order to maximize experimental controls.

Parsing the digit image data. When collecting the pixel maps of the images, it is necessary to maintain a standard aspect ratio and pixel size in order to train a singular neural network. In order to accomplish this, a screenshot of the image on the canvas was saved temporarily. From here, the PIL (Python Image Library) was used in order to crop the image to its minimum height and width and scale it to 16 x 16 pixels. Then the image was converted into a pixel map in which each black pixel was represented by a 1 and each white pixel was represented by a 0. The pixel map was stored as a one-dimensional list. Each image's supervised output data was stored as another tuple (derived from the file structure) and was a 10-tuple in which all indexes held a 0 except for the index of the correct output which was a 1. See the complete code for the implementation.

Sample data file organization. Each pixel map was stored in a separate text file as a string-tuple of length 256 (area of a 16 x 16 pixel map). Rather than have the supervision be stored within this tuple, it is preserved in the file system by its parent folder. Thus, the data folder will contain 10 folders, one for each digit 0-9.

Integrating the Neural Network and setting hyperparameters. Based on the format of the input data, the initialized Neural_Network object will have an input layer of size 256 neurons (one for each pixel) and an output layer of 10 neurons (one for each digit). The hidden layer size is currently arbitrarily set, but will be optimized with data analysis later on. Finally, a parsing method was created in order to retrieve the data from the sample storage file system and convert it into matrices that are in usable formats for the Neural_Network class. See the complete code for this implementation. The results of the ANN's output yielded a tuple of 10 predicted outputs.

2.2.2 Binary Diagnosis of Fine Needle Aspirates of Breast Tumors

The second application that this neural network framework was applied to was for the diagnosis of breast cancer biopsies, particularly fine needle aspirates of breast tumors. Image data was acquired from the UCI Machine Learning Database, and it consisted of analyzed data from raw images of biopsies. Particularly, instead of raw images, this data set consisted of 30 attributes derived algorithmically from the raw biopsies. The 30 attributes were computed as the mean, standard error, and max-mean (mean of top 3) of the following ten criteria:

```
a) radius (mean of distances from center to points on the perimeter)
b) texture (standard deviation of gray-scale values)
c) perimeter
d) area
e) smoothness (local variation in radius lengths)
f) compactness (perimeter^2 / area - 1.0)
g) concavity (severity of concave portions of the contour)
h) concave points (number of concave portions of the contour)
i) symmetry
j) fractal dimension ("coastline approximation" - 1)
```

In addition to this, each data tuple was supervised with diagnosis of the tumor where ‘M’ meant malignant and ‘B’ meant benign.

Parsing the tumor data and hyperparameters. This image data was parsed from one data file containing 357 benign cases and 212 malignant cases as string-tuples of length 32 (one additional element was an unused ID). Thus, the Neural_Network class was initialized with 30 neurons in the input layer and only one neuron in the output layer (with a 1 for malignant and a 0 for benign).

2.3 Testing

In order to test the accuracy of the neural network, the sample is divided into a training set and a testing set. Approximately 80% of the sample data is allocated to the training set, and the remaining 20% is allocated to the testing set. This process is applied for each sample set – particularly, for both the Optical Digit Recognition and the Breast Biopsy Diagnosis data sets.

Cost versus Minimization Iterations for Testing and Training Set. In order to track the learning progress of the neural network, the cost function is called on both the training set and the testing set for each iteration of the cost minimization training function; however, only the training set it used for training the ANN. The resulting testing and training cost data is collected against the number of iterations on the same coordinate-axis in order to the show the variability in learning progress for the two sets. Hypothetically, both costs should decrease at the same rate if the testing set is a perfect sample of the training set; however, it is more likely that the testing set cost will decrease at a slightly slower rate than the training set cost with a finite sample size.

Output Selection. Upon forward propagation, the ANN forward propagates a predicted output list, each index representing the propensity of the real output to be at that index, or the propensity that the index’s real output is a 1 (the higher the number the more likely that index should hold a 1). Thus, the index with the highest propensity is chosen as the output categorization, with the assumption that any given data can only fall into one category. Furthermore, the certainty of the neural network is a pseudo-certainty attribute calculated by the following formula (where n is the length of the list of predicted outputs):

$$Psuedo - Certainty = \frac{\max(\hat{y})}{\sum_{i=0}^{n} \hat{y}_i} \qquad [17]$$

Equation 17 takes into account the propensity of all of the predicted outputs (by summing them in the denominator) in order to find the weighted propensity of the maximum index. This offers a more useful metric than simply the raw propensities because it will take into account other relatively high propensities surrounding the maximum. For the case of the Breast Biopsies, Equation 17 will always be equivalent to 1 because the predicted output vector only has a length of 1. Thus, this metric is less useful for shorter smaller predicted output vectors and raw propensities may offer a more useful metric.

Percent Accuracy. The percent accuracy simply represents the fraction of correctly chosen categorizations out of the total number of categorizations. Once the ANN is fully trained, the testing sample is forward-propagated and the total number of correct outputs is calculated.

2.4 Data Analysis Dependent Factors

2.4.1 Accounting for Overfitting

Overfitting occurs when the ANN is trained to fit the training data so closely that the neural network begins to fail to predict correct outputs for testing data or other untrained data. This project employed multiple methods to overcome overfitting.

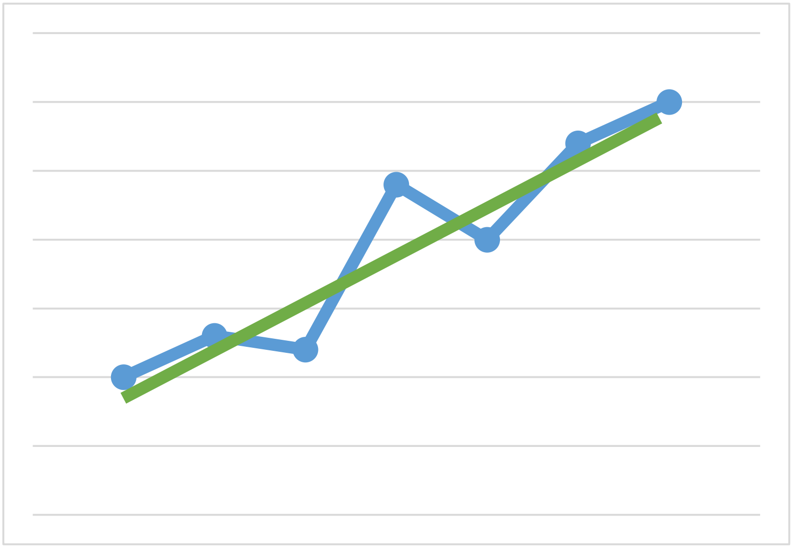


Figure 6. Overfitting Illustrated

As it is evident in **Figure 6** above, the blue curve illustrates the neural network overfitting the training data very closely. In reality, it is pretty evident that the green line offers a better approximate solution to training the neural network as it will better fit new data in the long run.

Early Stoppage. One method employed in stopping overfitting is to stop the minimization of the neural network before overfitting can occur. Overfitting can be located by analyzing the cost versus iterations graph of the testing and training data costs. The point at which the cost of the testing data begins to go up is approximately where overfitting begins. Thus, it may be useful to stop the minimization function at this point, before overfitting can occur.

Regularization Constant. One of the reasons overfitting occurs is because there are no constraints on how great the weights of the ANN can be. As a result, the neural network is allowed to tune the weights to very large values, which will fit the training data almost perfectly, but will sacrifice accuracy with testing and untrained data. As a result, the following constant is added to the cost function where lambda is some regularization constant, *n* is the sample size of the data, and W is the weight matrix.

$$\frac{\lambda}{2n}\sum {W_j}^2 \qquad [18]$$

By adding this regularization constant to the cost function, it will penalize the neural network for weights that are too great. This will in effect minimize the cost of the neural network at a point before the weights become too great. In effect, this has a similar (yet more automated result) to that of Early Stoppage techniques.

Additional Data. Ultimately, the most definitive way to account for overfitting is by increasing the sample size of the training data. At large sample sizes, even neural networks that may be overfitting the training sample will yield a good approximate solution for untrained data. In order to assert this claim, percent accuracies were collected against varying training sample sizes with a constant testing sample.

2.4.2 Determining Hidden Hyperparameters

The hyperparameter of the hidden layer is a relatively arbitrary value that may vary for any number of neural networks. Although no canonical method exists to determine the hyperparameter of the hidden layer given other attributes of the neural network, a data analytical approach may yield an approximately optimal number of neurons in the hidden layer. As a result, the percent accuracy of the neural network was plotted against the number of neurons in the hidden layer to determine an approximate trend for the hidden hyperparameter and to determine an optimal hyperparameter.

3. Data

3.1 Optical Digit Recognition [Data]

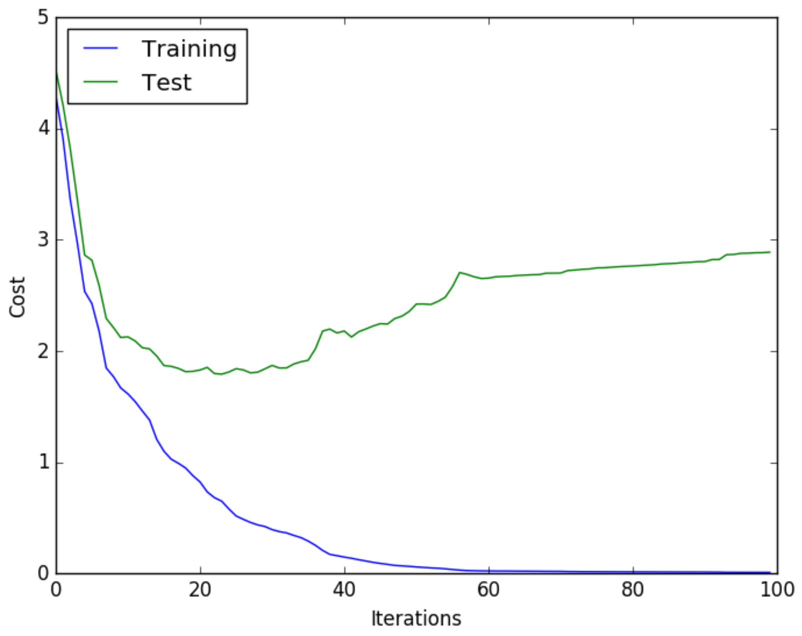


Figure 7. Cost vs Minimization Iterations of Test/Training Data (100 Iterations)

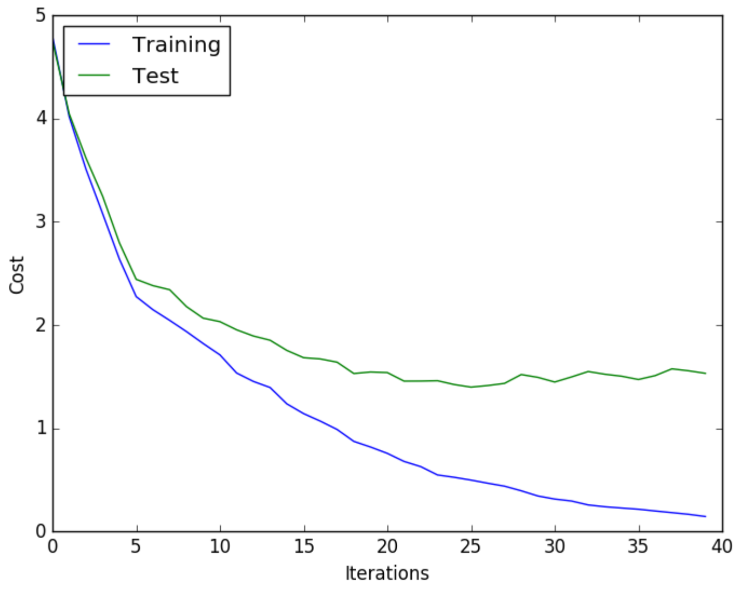


Figure 8. Cost vs Minimization Iterations of Test/Training Data (40 Iterations) (Early Stoppage)

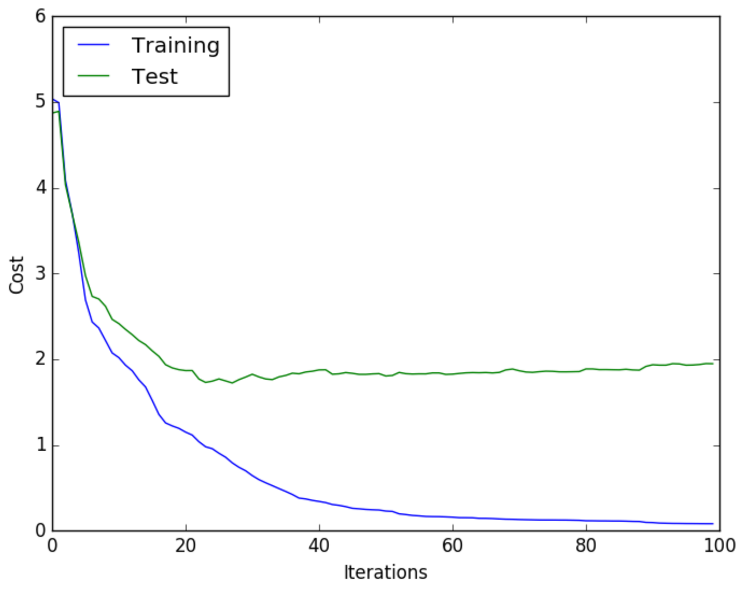


Figure 9. Cost vs Minimization Iterations of Test/Training Data (100 Iterations, Regularization Constant of .1)

	Number Correct	Size of Test Sample	Percent Accurate
Trial 1	210	260	0.807692
Trial 2	201	260	0.773077
Trial 3	212	260	0.815385
Trial 4	222	260	0.853846
Trial 5	217	260	0.834615
Trial 6	224	260	0.861538
TOTAL	**1286**	**1560**	**0.824359**

Table 1. Percent Accuracy Table for 6 Test Samples

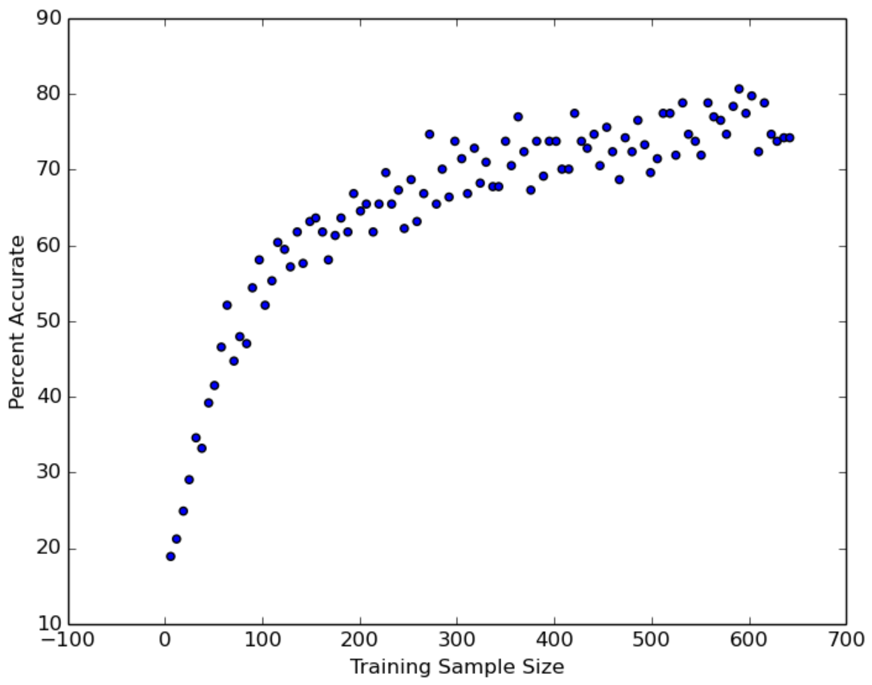


Figure 10. Percent Accuracy vs Training Sample Size

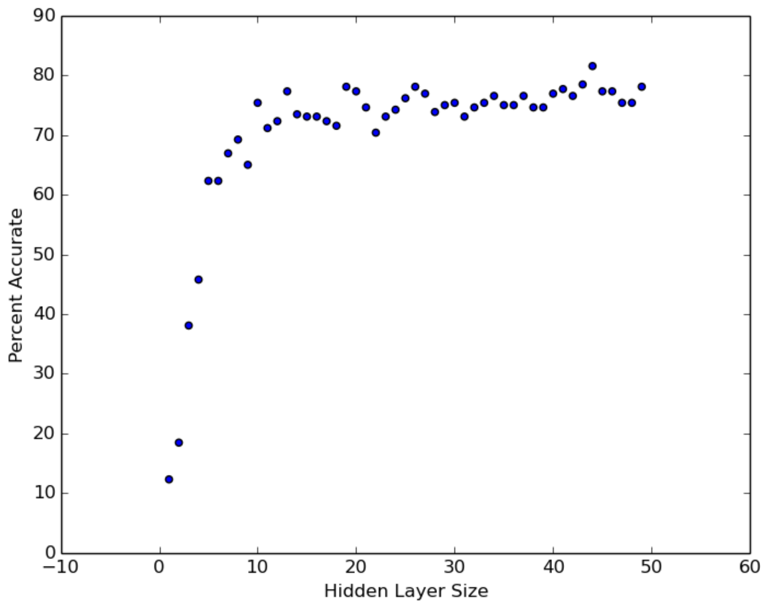


Figure 11. Percent Accuracy versus Hidden Layer Size (Neurons)

3.2 Diagnosis of Fine Needle Aspirates of Breast Tumors [Data]

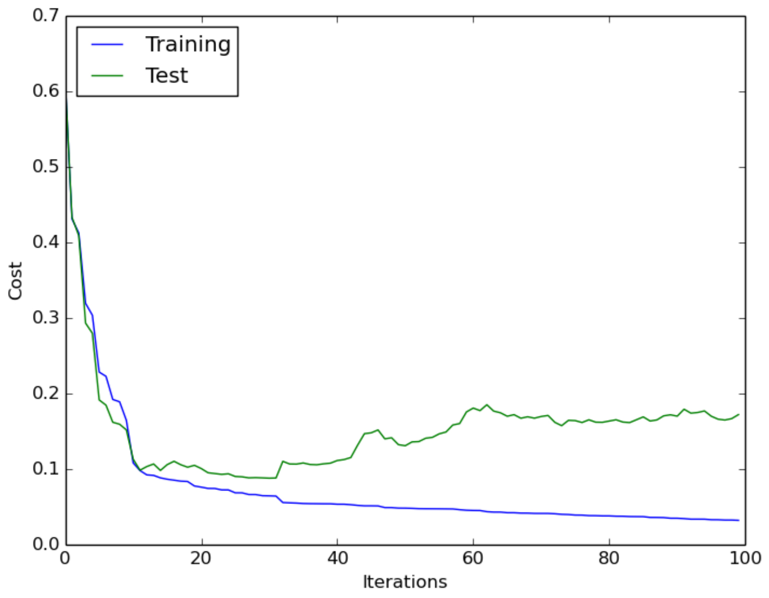


Figure 12. Cost vs Minimization Iterations of Test/Training Data (100 Iterations)

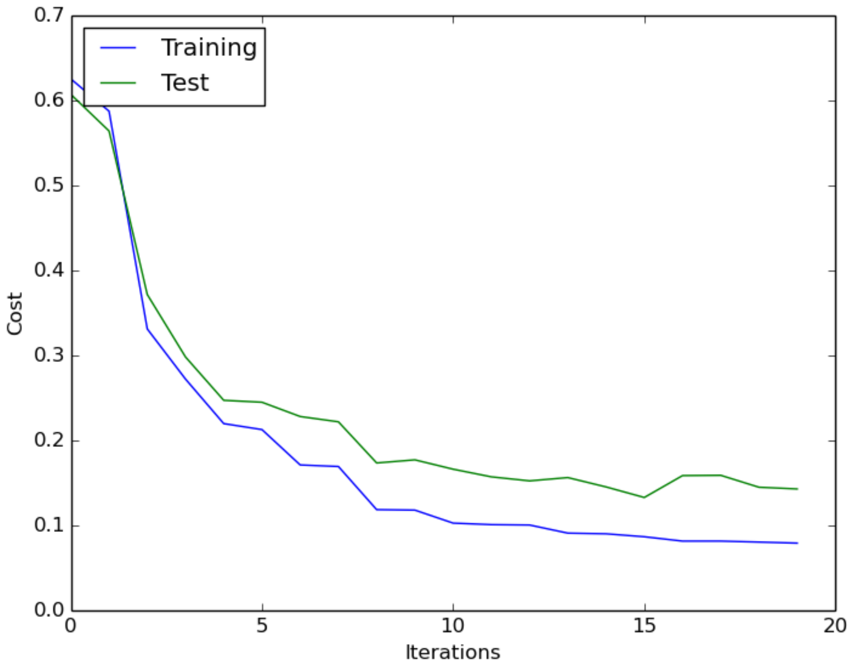


Figure 13. Cost vs Minimization Iterations of Test/Training Data (20 Iterations) [Early Stoppage]

	Number Correct	Size of Test Sample	Percent Accurate
Trial 1	140	143	0.979020979
Trial 2	138	143	0.965034965
Trial 3	136	143	0.951048951
Trial 4	138	143	0.965034965
Trial 5	139	143	0.972027972
TOTAL	**691**	**715**	**0.966433566**

Table 2. Percent Accuracy Table for 5 Different Test Samples

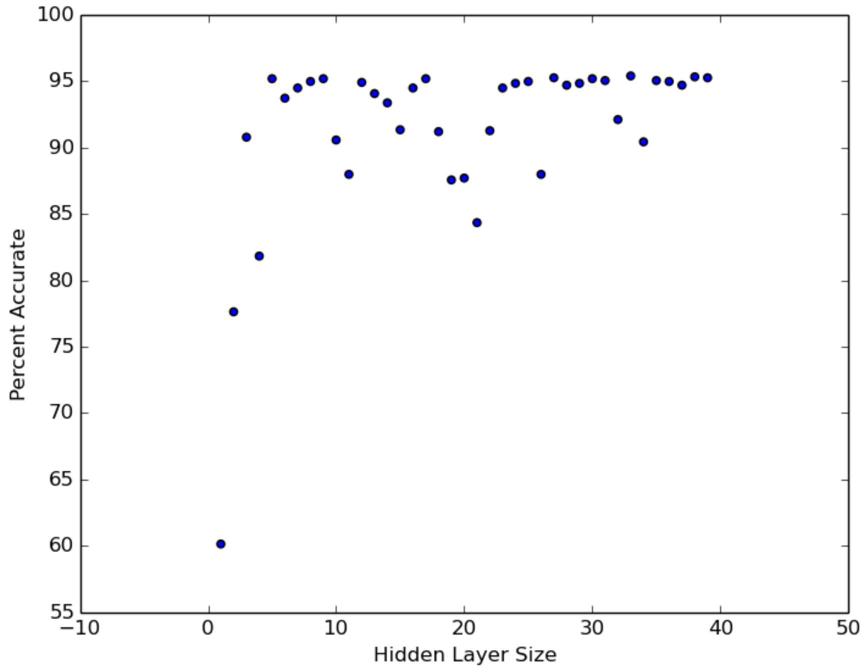


Figure 14. Percent Accuracy vs Hidden Layer Size

4. Discussion

Overall, the Artificial Neural Network framework built in this project illustrated a large degree of success. For one, it was highly extensible in that it proves its capability to be applied to any number of supervised image classification problems. This is proven by its successful application to both optical digit recognition as well as categorical diagnosis of fine needle aspirate biopsies of breast cancer tumors.

In the application to optical digit recognition, testing against part of the unsupervised sample yielded an accuracy of approximately 82.4% (see **Table 1**). This is a fairly accurate result; however, any lack of accuracy may likely be attributed to the neural network's conjugate gradient minimization algorithm overfitting the training data, as it is evident in **Figure 7** since the test sample's cost begins to take on an upward slope. The strategies employed to avoid this issue include early stoppage and adding a regularization constant term to the cost function. As it is evident in **Figure 8**, Early Stoppage did indeed decrease the terminal cost of the test samples substantially as it was able to end the training before overfitting could begin. Additionally, using the regularization constant for the same number of iterations as in **Figure 7** also decreased the terminal cost of the test samples as it is apparent in **Figure 9**. Although these strategies showed some success in alleviating the effects of overfitting, it is likely that overcoming overfitting will drastically decrease and accuracy will increase as the training sample size increases. This is illustrated by **Figure 10**, which shows that percent accuracy and sample size show a logarithmically growing proportionality. Another source of error may be due to the non-uniform distribution of samples across various categories, which would skew the weights of the neural network. A potential fix for this would be to implement bootstrap aggregation, an algorithm which evenly weights each category regardless of the non-uniformity of the distribution. **Figure 11** shows that after approximately 10 neurons, the size of the hidden layer did not significantly increase the percent accuracy, thus the optimal hidden layer was chosen to be between 10-20 neurons.

Furthermore, in the application for breast tumor biopsy diagnosis, testing against part of the unsupervised sample yielded an accuracy of 97% (see **Table 2**). As it is evident in **Figure 12** there are relatively small signs of overfitting at 100 iterations of minimization. Similarly, **Figure 13** shows that Early Stoppage at 20 iterations does indeed decrease the terminal cost of the test sample and results in almost no overfitting. This negates the value of using a regularization constant. This is a very accurate result; although miniscule, any lack of accuracy may likely be attributed to outliers in the test data set not accounted for when training, incomplete minimization of the cost function, and image inconsistencies. Similarly, an increase in the training data's sample size would likely increase the accuracy of the neural network. **Figure 14** shows that after approximately 10 neurons, the size of the hidden layer did not significantly increase the percent accuracy, thus the hidden layer was chosen to be between 10-20 neurons.

Future implications of this project include investigation into deeper, more convoluted neural networks. In this project, only one hidden layer was implemented into the neural network, but it may be useful to recursively extend the current backpropagation algorithm to include a larger number of hidden layers. Additionally, it may be useful to collect more training data or apply this ANN to more image classification problems in order to fine-tune it even further. Furthermore, it may also be useful to extend this neural network to continuous or unsupervised problems in the future and optimize approximate solutions to stock market analysis and the Travelling Salesman Problem (and other NP problems).

5. Conclusion

An extensible Artificial Neural Network (ANN) was created with the purpose of classifying images. The cost of the ANN was minimized using the Cross Entropy cost function for discrete categorization along with the Conjugate Gradient algorithm to conduct batch gradient descent. The resulting ANN design was able to classify handwritten digits as well as diagnose fine needle aspirates of breast cancer using data gathered manually and from the UCI Machine Learning Database with a relatively high percent accuracy; the optical digit recognition yielded a percent accuracy of 82.4% and the breast cancer diagnosis yielded a percent accuracy of 97%. Furthermore, the ANN's ability to be applied to multiple classification problems are a testament to its extensibility.

Moreover, a systematic approach was used in order to account for overfitting that occurred during the training of the sample data. In both test cases, some initial overfitting did occur. As a result, Early Stoppage and Regularization were employed in order minimize overfitting. It is hypothesized and backed by the data collected that an increase in the training sample size will ultimately overcome overfitting. In the future, it may be useful to extend this ANN to more practical applications, including stock market analysis and NP problems. It may also be interesting to compare these results to deeper or convolutional neural networks.

6. References

[1] Cernazanu-Glavan, Cosmin, and Stefan Holban. *Segmentation of bone structure in X-ray images using convolutional neural network*. Timosoara: Politehnica University of Timisoara, n.d. Digital file.

[2] Hagan, Martin T. *Neural Network Design*. N.p.: Oklahoma State, n.d. Print.

[3] Jenker, Andrej. "Introduction to Artificial Neural Networks." *Artificial Neural Networks - Methodological Advances and Biomedical Applications*. Ed. Kenji Suzuki. N.p.: InTech, 2011. N. pag. Digital file.

[4] Jiang, J. *Medical image analysis with artificial neural networks*. N.p.: Computerized Medical Imaging and Graphics, 2010. Print.

[5] Le Cun, Y. *Handwritten Character Recognition Using Neural Network Architectures*. Holmdel: AT&T Bell Laboratories, 1990. Print.

[6] Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[7] Nielsen, Michael A., "Neural Networks and Deep Learning", Determination Press, 2015

[8] Rojas, Raul. *Neural Networks: A Systematic Introduction*. Berlin: Springer, 1996. Digital file.

[9] Russell, Stuart J., Peter Norvig, and Ernest Davis. *Artificial Intelligence: A Modern Approach*. 3rd ed. Upper Saddle River: Prentice Hall, 2010. Print.

[10] Zhang, Q. J. *Neural Network Structures for RF and Microwave Applications*. Orlando: IEEE AP-S Antennas and Propagation Int. Symp., 1999. Digital file.