Towards Explaining Neural Networks: Tools For Visualizing Activations and Parameters

Juan Puebla
University of Texas at El Paso

Follow this and additional works at: https://scholarworks.utep.edu/open_etd

Part of the Computer Sciences Commons

Recommended Citation
https://scholarworks.utep.edu/open_etd/4051

This is brought to you for free and open access by ScholarWorks@UTEP. It has been accepted for inclusion in Open Access Theses & Dissertations by an authorized administrator of ScholarWorks@UTEP. For more information, please contact lweber@utep.edu.
TOWARDS EXPLAINING NEURAL NETWORKS: TOOLS FOR VISUALIZING
ACTIVATIONS AND PARAMETERS

JUAN PUEBLA

Master’s Program in Computer Science

APPROVED:

______________________________
Martine Ceberio, Chair, Ph.D.

______________________________
Vladik Kreinovich, Ph.D.

______________________________
Christoph Lauter, Ph.D.

______________________________
Eric MacDonald, Ph.D.

______________________________
Stephen Crites, Ph.D.
Dean of the Graduate School
TOWARDS EXPLAINING NEURAL NETWORKS: TOOLS FOR VISUALIZING ACTIVATIONS AND PARAMETERS

by

JUAN PUEBLA, B.Sc.

THESIS
Presented to the Faculty of the Graduate School of
The University of Texas at El Paso
in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE

Department of Computer Science
THE UNIVERSITY OF TEXAS AT EL PASO
December 2023
Acknowledgments

I would like to extend my sincere thanks to Mayra L. Robles and Angel Coronel for helping me conduct the experiments in this research. Thanks also to Jorge L. Rodriguez Ortega and Elton Villa for collaborating with me on web app development. My appreciation to my mentors throughout my professional career, specifically James Calles and Jeanne Rickert. Lastly, I send my deepest appreciation to my committee.
Abstract

There is a growing number of applications using neural networks for making decisions. However, there is a general lack of understanding of how neural networks work. Neural networks have even been described as black boxes which has led to a lack of trust in artificially intelligent programs. To remedy this, explainable artificial intelligence has risen as a means to validate the decision-making processes and the results of computer programs that use artificial intelligence. The work in this master’s thesis is our contribution to explainable artificial intelligence, focusing on neural networks with the goal of helping users make more sense of the algorithms subsumed by the network. Our research deals with the visualization of node activations and weights within a neural network to see how data travels through the network to make decisions.

Additionally, when using neural networks, it is not clear which structure should be used or how the structure of the network influences its performance. Resulting in a network with unnecessary nodes or connections that contribute to its space and computational complexity. In this work, we look at the problem of identifying edges and nodes in the network that we can remove without compromising its overall performance. Some of the pruning techniques that we have explored are pruning never-activated nodes, mostly unactivated nodes, always-activated nodes, and $k$ nodes per layer. The results from pruning never activated and mostly unactivated nodes show similar accuracy to the original accuracy of the network, with an increase in accuracy after retraining. Pruning always activated nodes and $k$ nodes per layer results in comparable accuracy after some retraining.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents</td>
<td>v</td>
</tr>
<tr>
<td>List of Tables</td>
<td>vii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>viii</td>
</tr>
<tr>
<td><strong>Chapter</strong></td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2 BACKGROUND</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Background Notions</td>
<td>3</td>
</tr>
<tr>
<td>2.1.1 What is Artificial Intelligence?</td>
<td>3</td>
</tr>
<tr>
<td>2.1.2 What is Machine Learning?</td>
<td>4</td>
</tr>
<tr>
<td>2.1.3 Neural Networks</td>
<td>5</td>
</tr>
<tr>
<td>2.1.4 Training a Neural Network</td>
<td>9</td>
</tr>
<tr>
<td>2.1.5 Types of Neural Networks</td>
<td>10</td>
</tr>
<tr>
<td>2.1.6 Neural Network Applications</td>
<td>10</td>
</tr>
<tr>
<td>2.1.7 Pruning</td>
<td>11</td>
</tr>
<tr>
<td>2.1.8 Explainable AI</td>
<td>11</td>
</tr>
<tr>
<td>3 RELATED WORK</td>
<td>13</td>
</tr>
<tr>
<td>3.1 Visualization &amp; Tools</td>
<td>13</td>
</tr>
<tr>
<td>3.2 Pruning</td>
<td>16</td>
</tr>
<tr>
<td>4 CONTRIBUTION TO FULLY CONNECTED NEURAL NETWORK VISUALIZATION</td>
<td>18</td>
</tr>
<tr>
<td>4.1 Problem Description</td>
<td>18</td>
</tr>
<tr>
<td>4.1.1 Weights of the Edges</td>
<td>18</td>
</tr>
<tr>
<td>4.1.2 Activations of the Nodes</td>
<td>19</td>
</tr>
<tr>
<td>4.2 Visualization Solutions</td>
<td>20</td>
</tr>
</tbody>
</table>
4.2.1 Activation Heatmap ........................................... 20
4.2.2 Graph for Influential Weights ................................. 21
4.2.3 Weight Intensity Map ........................................... 22

5 CONTRIBUTION TO PRUNING TECHNIQUES ......................... 25
5.1 Pruning by Activation ............................................ 25
5.1.1 Experiment: Pruning Nodes that are Never Activated ........... 25
5.1.2 Experiment: Pruning Nodes that are Always Activated ........... 30
5.2 Pruning as Influenced by Weights ................................ 33
5.2.1 Experiment: Prune with threshold over all network ............. 33
5.2.2 Experiment: Prune \( k \) nodes per layer ....................... 35

6 EXPLORING FEATURE ARITHMETIC ................................. 38
6.1 Experiment: Visualizing parts of a digit .......................... 38
6.2 Experiment: What makes a seven a seven? ....................... 42

7 FULLY CONNECTED NEURAL NETWORK VISUALIZATION WEB APP . 53
7.1 Web Application .................................................. 53
7.2 Current Solution .................................................. 55

8 CONCLUSION AND FUTURE WORK ................................. 56
8.1 Conclusion ...................................................... 56
8.2 Future Work ..................................................... 56

References .......................................................... 58
Curriculum Vita ...................................................... 62
List of Tables

4.1 Example structure of a neural network with the number of parameters, i.e. weights, per layer. .......................... 19

5.1 Structure of Neural Networks. ............................ 28
5.2 Results from pruning nodes that are never activated. ................. 29
5.3 Results from pruning nodes that are mostly inactivated. Pruned nodes are selected based on a threshold $t$. ...................... 29
5.4 Results from pruning nodes that are always activated. ................. 32
5.5 Prune nodes based on a selected threshold. ........................ 34
5.6 Prune $k$ Nodes per Layer based on Max weight values .............. 36

6.1 Results for the feature arithmetic experiment on the image of a seven. The table displays the activation maps and classifications for images of a seven, a dash, and a slash. .......................... 41
6.2 Weight maps for nodes 0-5 in the first hidden layer of NN1. ........ 47
6.3 Weight maps for nodes 6-11 in the first hidden layer of NN1. ....... 48
6.4 Weight maps for nodes 12-15 in the first hidden layer of NN1. ....... 49
6.5 Results for the experiment on the basic characteristics of a seven. The table displays the activation maps and classifications for images of a seven and four cases of a decomposed seven. .......................... 51
List of Figures

2.1 Illustration of the graph of a Fully-Connected Feed-Forward Neural Network, with input layer nodes in yellow, hidden layer nodes in blue, and output layer nodes in green. Each node in each layer is connected to each node in the next layer. ................................................................. 6

2.2 Illustration of how the values passed from a previous layer come together through the weights and the result is filtered by the activation function $\sigma$. ................................. 9

3.1 Image of the CNN Explainer tool, showing how an image of an espresso is processed through a convolutional neural network. ................................. 14

3.2 Feed Forward Neural Network Visualization tool. The input is a grid of 9 values. There are 3 hidden layers and an output layer of 4 nodes. Activated nodes are colored green. ................................. 15

4.1 Heatmap graph showing the node non-activations, where the darker colored shade represents the least activated nodes. The neural network has eight hidden layers and 16 nodes per layer. The dataset is 10,000 examples. ................................. 21

4.2 Heatmap graph showing the activated nodes, where the darker colored shade represents the most activated nodes. The neural network has eight hidden layers and 16 nodes per layer. The scale shows 8000 because it does not include activations with a value of zero. ................................. 22

4.3 Box plot of hidden layer three from a NN with eight layers and 16 nodes per layer. The plot shows the distribution of the absolute value of the node weights. Nodes 2, 3, 13, and 14 have the lowest max and median, making them ideal for pruning. ................................. 23
4.4 Weight map of a node in the first hidden layer of a NN trained on MNIST dataset. The map shows the intensity of the weight for each pixel in the 28x28 pixel images. The darker areas are the features that the node is focusing on and the lighter areas are the features the node is ignoring.

6.1 Activation map for each digit class displays which nodes get activated and to what intensity. True Positive examples were used for the activation maps.

6.2 The overlayed activation maps of the dash and slash images.

6.3 Weight at x,y coordinates in the weight map corresponds to the pixel at x,y coordinates in the input image.

6.4 Weight map for node 0 in the first hidden layer of NN1. The areas circled in red are the features of the input image that the node is focusing on.

6.5 Weight map for node 6 in the first hidden layer of NN1. The areas circled in red are the features of the input image that the node is focusing on.

6.6 Weight map for node 14 in the first hidden layer of NN1. The areas circled in red are the features of the input image that the node is focusing on.

7.1 Initial prototype of the Feed-Forward Neural Network web application.
Chapter 1

INTRODUCTION

The rise of big data came from the wide availability of the internet, which created the largest online database [14], mobile computing through the continuous collection of our data via our smartphones, and more recently from social media where all our online social interactions are being recorded [3]. Data has been dubbed the new oil [25], and consequently, interest in Artificial Intelligence and Machine Learning has increased, including deep learning with neural networks.

Neural networks have been applied to various fields such as image recognition and natural language processing [30]. Generally, there still isn’t a good understanding of how neural networks function or how they should be structured. The research in this thesis focuses on the visualization of neural networks to gain a better understanding of how they work. With visualization, we can abstract the math behind neural networks. We see the network as a collection of neurons that activate like the neurons in our brains based on which neural networks were inspired [22].

This master’s work is an exploration of neural networks (NN), specifically fully connected neural networks (FCNN). Chapter 2 provides background notions on neural networks, including their history, what they are, their components, and their purpose. Chapter 3 reviews related work that has been done in the field of deep learning, specifically on the topics of visualization and pruning, which are most related to our work. Chapter 4 describes our contribution to the visualization of neural networks, concerning the activation of nodes and the weights of edges. Chapter 5 describes our contribution to pruning techniques of neural networks. We performed experiments that involved pruning nodes identified by the activation of nodes or by the importance of weights. Chapter 6 explores
neural network feature arithmetic, where we use visualization to understand how neural
networks process information and make decisions. In Chapter 7, we propose a fully con-
nected neural network web application to ease the use of neural networks for non-experts.
Conclusions and future directions are outlined in Chapter 8.
Chapter 2

BACKGROUND

2.1 Background Notions

2.1.1 What is Artificial Intelligence?

Intelligence is a difficult concept to define resulting in various definitions or, for some, making it undefinable. Intelligence can be defined as the ability to learn and apply knowledge [8]. Others would describe intelligence as the ability to innovate, to be curious, and to able to evolve over time [14]. With a range of definitions for intelligence, the definition of artificial intelligence (AI) is in a similar dilemma. The definition of AI may vary depending on who is posed this question. Some even argue that the word “intelligence” is the incorrect word to use for this field, thus implying that there is no such thing as Artificial Intelligence [14].

As AI stands now, there are two different approaches: a human-centered approach, which focuses on replicating human behavior, and a rational approach, which involves mathematics and engineering [22]. These two approaches can be further broken down by two criteria: thinking and acting, resulting in four different categories of AI [22]:

- Thinking Humanly: AI is defined as the automation of activities, such as decision-making, problem solving, and learning, which are typically associated with human thinking [6].

- Thinking Rationally: AI is defined as the study of the computations that make it possible to perceive, reason, and act [28].
• Acting Humanly: AI is defined as the art of creating machines that perform functions that require intelligence when people perform them [17].

• Acting Rationally: AI is defined as the study of the design of intelligent agents [19].

From all these definitions, one thing is certain, the study of Artificial Intelligence attempts to understand and create intelligent machines [22]. AI attempts to build machines that function autonomously in complex and changing environments [22]. This paper explores AI from a “thinking rationally” approach, specifically a machine learning model known as a Neural Network (NN).

### 2.1.2 What is Machine Learning?

Machine learning is a subfield of AI that involves the development and evaluation of algorithms that enable a computer to find functions based on a dataset [15]. A machine learning program is different from a typical computer program in that it is a general template, or model, with modifiable parameters [3]. When different values are assigned to these parameters, the program will exhibit different behavior [3]. The learning algorithm adjusts the parameters of the template by optimizing a performance criterion based on the data [3]. The goal is to fit the model to the data, to correctly predict new cases.

There are three categories of machine learning which include supervised, unsupervised, and reinforcement learning [15]:

• Supervised machine learning uses datasets where each example is labeled [15]. The label is the expected output or target value for the example [15]. In supervised learning, the computer reads a labeled dataset and learns a function that maps from input to output [22]. Two supervised machine learning methods are regression and classification [3]. The research in this paper deals with classification, a supervised learning method.

• Unsupervised machine learning is meant for datasets that do not have target values
in the dataset [15]. Unsupervised machine learning attempts to find associations between examples [3]. One method for unsupervised machine learning is clustering which attempts to find clusters or groups of examples [3].

- In reinforcement machine learning the computer program learns from a series of reinforcement [22], to take actions that maximize rewards and minimize penalty [3].

### 2.1.3 Neural Networks

The first recognized work of AI was achieved by Warren McCulloch and Walter Pitts in 1943 [22]. Their inspiration came from three sources [22]:

1. Basic physiology and the function of neurons in the brain;
2. Propositional logic; and
3. Alan Turing’s theory of computation.

The model they proposed was a network of connected artificial neurons, where each neuron can either be “on” or “off” [22]. A neuron will switch to “on” when it’s stimulated by a significant number of neighboring neurons [22]. Using this model, they showed that any computable function could be computed and any logical connectives could be implemented [22]. This was the beginning of the neural network.

A neural network (NN) is a computational model inspired by the structure of the human brain [15]. A NN is composed of nodes and weighted directed edges [22]. Each edge in the NN connects two nodes and is directed, meaning the information traveling between the two nodes flows in one direction [15]. The research in this paper focuses on fully connected feedforward neural networks, which have edges connected in only one direction forming a directed acyclic graph [22]. The fully connected feedforward NN is arranged into layers where each node in a layer only receives input from the previous layer’s nodes [22]. Figure 2.1 displays the structure of a fully-connected feed-forward NN.
Figure 2.1: Illustration of the graph of a Fully-Connected Feed-Forward Neural Network, with input layer nodes in yellow, hidden layer nodes in blue, and output layer nodes in green. Each node in each layer is connected to each node in the next layer.
The input layer is the information that is fed into the NN. The nodes in the input layer are considered sensing neurons and thus no processing of information occurs in this layer [15]. The hidden layers are the layers between the input and output layers. A NN with two or more hidden layers is called a deep learning neural network (DNN) [15]. The output layer produces the output of the NN.

When we describe neural networks, we describe them using their parameter values:

- Number of hidden layers
- Number of nodes in the input layer
- Number of nodes in the output layer
- Number of nodes per hidden layer
- Type of the activation function

Given a neural network of \((n - 1)\) hidden layers, we use the following notation:

- We use letter \(L\) to refer to the layers:
  
  - \(L_0\) is the input layer,
  
  - \(L_i\) is the \(i\)th layer, \(\forall i \in \{1, ..., n - 1\}\),
  
  - \(L_n\) is the output layer.

- In any given layer \(L_j\) (with \(j \in \{0, ..., n\}\)), we denote by:
  
  - \(N_j\) the number of nodes in the layer;
  
  - \(n_{j,k}\) each node in the layer, for \(k \in \{1, ..., N_j\}\)
  
  - \(a_{j,k}^{i}\) the value (activated value) at node \(n_{j,k}\), for \(k \in \{1, ..., N_j\}\)

- The weights on the edges from one layer \(L_j\) to the next \(L_{j+1}\), for \(j \in \{0, ..., n - 1\}\), are denoted by a matrix \(W_j\) of \(N_j\) rows and \(N_{j+1}\) rows, where

\[
W_j = (w_{k,l}^{j})_{k,l}, \text{ with } k \in \{1, ..., N_j\} \text{ and } l \in \{1, ..., N_{j+1}\}
\]
The biases on the nodes of Layer $L_j$ are denoted as a vector $B_j = (b_{j,k})_{j,k}$ where $k \in \{1, ..., N_j\}$.

The activation function at each node is usually denoted by $\sigma$. There are many different types of functions used as activation functions. Some of the most common ones are the following [4, 31]:

- ReLU:
  \[
  relu(z) = \max(0, z) \tag{2.1}
  \]

- Sigmoid:
  \[
  \sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.2}
  \]

- SoftMax:
  \[
  \sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{2.3}
  \]

- TanH:
  \[
  \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.4}
  \]

The activated value $a^{j,k}$ of a node $n_{j,k}$ is described by Equation (2.5) [22]:

\[
  a^{j,l} = \sigma \left( \sum_{k=1}^{N_{j-1}} w_{k,l}^{j-1} \cdot a^{j-1,k} + b_{j,l} \right) \tag{2.5}
  \]

Figure 2.1.3 illustrates the activation of a node, where each value $x_i$, out of a node of the previous layer, is multiplied by the corresponding weight, all these multiplied values are added to each other, and the result is processed through the activation function that acts as a filter of the incoming value [22]. Note that on this figure, the biases have been omitted.
2.1.4 Training a Neural Network

A NN is trained on a dataset to “learn” how to classify, i.e., make predictions on new examples of data. During training the NN predicts the label for an example, calculates the error i.e. how far the prediction is from the example’s actual label, and then fixes itself to reduce the error. Training a NN is an iterative process that involves forward propagation, calculating the total error, and then backpropagation.

During forward propagation, data is fed into the input layer and then it travels through the network to generate an output [24]. The total error is then calculated by taking the difference between the NN’s predicted output and the target output. Finally, the weights of the network are updated during backpropagation. Backpropagation is the process of traversing back through the network, from the output layer back to the input layer, while updating the weights using an optimization method [24]. Ultimately, training a NN is searching for the most optimal set of weights that produce the best accuracy.
2.1.5 Types of Neural Networks

There are different types of neural networks. Some of the more popular types of neural networks are fully connected neural networks, convolutional neural networks, and recurrent neural networks [15].

A fully connected neural network has layers of nodes where each node is connected to all nodes in the next layer [15]. A fully connected neural network is a type of feed-forward neural network, a network that does not contain loops, i.e. all the connections are directed edges that go from the input to the output layer [15]. The research in this master’s thesis utilizes fully connected neural networks.

A convolutional neural network (CNN) uses a specific type of layer called a convolutional layer [4]. A convolution is a mathematical operation on two or more functions [4]. In a CNN, the convolutional layers are not fully connected to the previous layers, i.e. they are sparsely connected [4]. The convolutional layer contains a kernel or filter matrix that strides over the input to extract different features [2].

A recurrent neural network (RNN) contains a recurrent loop which is a connection or edge that forms a directed cycle [15, 29]. Information flows in a loop, where the output of a node is fed back to the node in the next iteration [15]. RNNs were designed for processing sequential data such as a sentence or a time series [29]. The main idea is that the recurrent loop gives the RNN a memory that allows it to process each new input in the context of the previous data input [15].

2.1.6 Neural Network Applications

A neural network classifies a data point. When a NN is trained it is able to make a prediction on a new example of data that it receives. NN automates the classification process to some level of accuracy. Computer scientists do not need to develop and maintain an algorithm that classifies data, the NN is the algorithm.

Neural networks are used for the classification of data, for example pattern recognition
(PR) [1]. Common features in a set of data can define a pattern, where a pattern is a set of items that are similar [1]. Examples of a pattern can be a handwritten word or digit, a breed of dog, or a speech signal. Recognition in pattern recognition refers to the task of identifying a piece of data [1].

Neural networks have been applied to a wide range of problems, such as handwritten character classification, speech classification, and medical diagnosis [1]. An example of a neural network application is identifying different types of non-small cell lung cancers from computerized tomography (CT) images [23]. Another example of a neural network application is detecting if a patient has a brain tumor by classifying a magnetic resonance imaging (MRI) image [16].

2.1.7 Pruning

Deep neural networks are extremely effective models for pattern recognition tasks, however, they suffer from high computational complexity the more layers and nodes they have [18]. Thus, it is highly desirable to reduce the network size in order to decrease the number of parameters, preferably using a network reduction technique that doesn’t cause significant performance degradation [18]. Pruning is the process of removing node(s) from the neural network in order to reduce the number of parameters i.e. weights. By reducing the number of weights the NN has less calculations to perform and thus it is computationally faster. This paper will go over several pruning techniques that improve the computational efficiency of an NN without a significant reduction in the network’s accuracy.

2.1.8 Explainable AI

There is a lack of trust in AI programs due to their automation, autonomy, and black-box nature [11]. Thus, some companies are hesitant to use AI in their systems. Explainable AI (XAI) deals with AI technologies that provide an understandable explanation of the AI’s decision-making processes and classification results [11]. With XAI, people are able to
build trust in artificial intelligence because it enables them to validate the inner workings of artificial intelligence [11].

The work in this master’s thesis is our contribution to explainable AI. Our research deals with the visualization of node activations and weights within a neural network to give a non-expert the ability to see how neural networks make decisions. This will be further explored in Chapter 4 Contribution to Fully Connected Neural Network Visualization, Chapter 5 Contribution to Pruning Techniques, and Chapter 6 Exploring Feature Arithmetic.
Chapter 3

RELATED WORK

3.1 Visualization & Tools

The weight values and output of nodes of a fully connected NN can be obtained, but it is difficult to analyze that amount of data. By visualizing the NN as a graph with activated nodes we can quickly find what nodes in the NN are most important. If a specific node is never activated, it could mean that the NN could function without this specific node. Thus, the NN could be optimized, resulting in reduced memory usage and faster NN. Additionally, with visualization, non-technical users can see what’s happening in the network without having to understand the math behind it.

An example of a current solution to this problem is, e.g., the CNN Explainer (Figure 3.1). This tool focuses on visualizing convolutional neural networks (CNNs) as a learning tool. It is described as an interactive visualization tool designed for non-experts to learn and examine CNNs [27]. It helps users to understand the underlying components of CNNs and examine the interactions between low-level mathematical operations and high-level model structures [27]. CNN explainer is meant to teach: it isn’t meant for further evaluation of neural networks or to find opportunities for optimization. The tool is limited to classifying images on a pre-defined pre-trained CNN model, which is reasonable because the tool is meant to teach.

The application Feed Forward Neural Network Visualization is a tool that visualizes the activations of nodes in a feedforward neural network [26]. The tool is a source code project, written in Java, that needs to be run in an integrated development environment. It contains configuration files that allow the user to define the structure of the neural network,
Figure 3.1: Image of the CNN Explainer tool, showing how an image of an espresso is processed through a convolutional neural network.

including the number of layers and nodes per layer [26]. The tool functions by generating random inputs and feeding the inputs one by one to the network until an output node is activated [26]. The main purpose of the tool is to visualize node activations, there are no additional features that the tool can perform.

Another visualization solution is described in the paper Visualization System for Evolutionary Neural Networks for Deep Learning. The paper’s focus is to visualize the search space of a genetic algorithm during the neural network training process [7]. Genetic algorithms are one solution to train neural networks, by trying different parameter values to find an optimal solution [7]. The visualization approach in this paper can be used to help
users understand and improve genetic algorithms used in neural networks.

As in the previous solution, the paper *Visualizations of the Training Process of Neural Networks* describes another visualization solution of the training process of a neural network. The proposed solution visualizes the parameters or weights of the neural network after each training iteration [5]. The research uses different data transformations for the visualizations, e.g. *dif* which calculates the difference between iterations [5]. One of the visualization solutions shows the training path from the start of training to the convergence point. The authors do mention a limitation, their visualizations will require a large amount of memory space for large neural networks because of all the parameters, i.e. weights, for each iteration [5].
3.2 Pruning

As stated previously, deep neural networks are effective machine learning models for pattern recognition but their computational complexity increases the deeper the network is [18]. Pruning is an effective technique of deleting nodes to reduce the network’s size and thus lowering the network’s computational complexity [18]. Pruning allows neural networks to be used on a wider range of platforms, such as low-performance computing devices like mobile and embedded devices [18].

In the article *DNN Pruning with Principle Component Analysis and Connection Importance Estimation*, the authors’ pruning approach is to remove redundant and unnecessary nodes [21]. They propose a two-step pruning technique based on Principal Component Analysis (PCA) and the Distribution Invariant Relative Importance Estimation (DIRIE) metric [21]. The first step is to apply PCA, which transforms a set of N-dimensional variables into a coordinate system that is ordered by variance [21]. PCA will expose similar or redundant nodes that can be removed. The second step is to use DIRIE which measures the relative importance of each node’s connection [21]. A node’s connection, or edge, is considered unimportant if the absolute value of its weight is small compared to the average absolute value of all the node’s incoming edges’ weights [21]. The connections that were identified as unimportant are then removed. Finally, the network is retrained. Their proposed pruning technique finds the optimized network in two steps and achieves similar results to existing iterative pruning techniques [21].

In the paper *Learning Both Weights and Connections for Efficient Neural Networks*, the authors propose a three-step pruning technique that removes redundant edges [12]. The first step involves training the neural network. In the second step, unimportant connections, edges with weights lower than a threshold, are pruned [12]. The final step is to retrain the neural network to recover accuracy [12]. Their technique is an iterative process, repeating the pruning and retraining steps to further reduce the network’s parameters [12]. Their experiment demonstrated a huge decrease in network size, from 61 million parameters to
6.7 million parameters, without accuracy loss [12].

Another pruning technique, as described in *A Novel Layerwise Pruning Method for Model Reduction of Fully Connected Deep Neural Networks*, is to prune nodes with strongly correlated activations [18]. The correlated nodes are identified using the mean and covariance per layer [18]. Covariance measures the distribution of data in relation to the mean [13]. The authors’ pruning technique results in a reduction in network size with minimal performance loss.

Besides pruning, another technique for neural network parameter reduction is a low-rank representation of layers [20]. The paper *Data-Driven Low-Rank Neural Network Compression* proposes a Data-Driven Low-rank (DDL) method which finds the lowest rank approximation for each fully connected layer to compress the network [20]. The method in this paper requires no retraining and results in high parameter reduction with minimal accuracy loss.
Chapter 4

CONTRIBUTION TO FULLY CONNECTED NEURAL NETWORK VISUALIZATION

4.1 Problem Description

A growing number of applications rely on Neural Networks. However, we do not have a good understanding of how a NN makes decisions. To better understand how a NN works, we need to collect the parameters of the NN as well as the information that travels through the network. In other words, we need to look at the weights of the edges and the activations of the nodes. We can obtain the weight and activation values of a fully connected NN, but analyzing that amount of data is difficult. With visualization, we can see how data travels through the NN and identify which nodes in the NN are the most important.

4.1.1 Weights of the Edges

Let us consider the following scenario. We train a NN on the Modified National Institute of Standards and Technology (MNIST) dataset [9]. The neural network is fully connected with eight hidden layers and 16 nodes per layer. Each image in the MNIST dataset is 28 by 28 pixels [9]. Therefore, each image has 784 input values, one for each pixel. The neural network will have 14,336 weights, or parameters, in the hidden layers. Table 4.1 shows the parameter breakdown by layer. The number of weights alone would be complex to analyze.
without the assistance of a computer program.

<table>
<thead>
<tr>
<th>Layer Number</th>
<th>Layer Type</th>
<th>Nodes</th>
<th>Trainable Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Input</td>
<td>784</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Hidden</td>
<td>16</td>
<td>12544</td>
</tr>
<tr>
<td>2</td>
<td>Hidden</td>
<td>16</td>
<td>256</td>
</tr>
<tr>
<td>3</td>
<td>Hidden</td>
<td>16</td>
<td>256</td>
</tr>
<tr>
<td>4</td>
<td>Hidden</td>
<td>16</td>
<td>256</td>
</tr>
<tr>
<td>5</td>
<td>Hidden</td>
<td>16</td>
<td>256</td>
</tr>
<tr>
<td>6</td>
<td>Hidden</td>
<td>16</td>
<td>256</td>
</tr>
<tr>
<td>7</td>
<td>Hidden</td>
<td>16</td>
<td>256</td>
</tr>
<tr>
<td>8</td>
<td>Hidden</td>
<td>16</td>
<td>256</td>
</tr>
<tr>
<td>9</td>
<td>Output</td>
<td>10</td>
<td>160</td>
</tr>
</tbody>
</table>

Total Trainable Parameters | 14496

Table 4.1: Example structure of a neural network with the number of parameters, i.e. weights, per layer.

4.1.2 Activations of the Nodes

Let us continue with the previous scenario. The neural network has to be tested after it has been trained. The MNIST test dataset has 10,000 test images. Each example in the test dataset is fed into the trained neural network to predict its classification, i.e., for each image in the MNIST test dataset, predict which digit the image is. The node activation values are collected for each example in the test dataset. A dataset of 10,000 images fed into a NN of 8 hidden layers and 16 nodes per layer will generate 1,280,000 activation values. The volume of activation values is too immense for a human to make sense of the data. Thus,
visualization of the activation values is preferred to easily see what is happening inside the NN.

4.2 Visualization Solutions

4.2.1 Activation Heatmap

We propose visualizing a Fully Connected NN as a heatmap graph highlighting important and unimportant nodes. With visualization, non-technical users can see what is happening inside the network without understanding the math behind it. A heatmap is ideal to effectively visualize the intensity of a node’s activation or non-activation. Using a color gradient and based on the shade of color, it is easy to see the most activated nodes and the least activated nodes.

One application is to use a heatmap on the activation values to identify the least activated nodes, see Figure 4.1. To achieve this, we count the number of occurrences where each node’s activation value is zero. The result is a matrix or 2-D array, where the dimensions of the matrix are the number of layers by the number of nodes. In the matrix, the rows represent the layers, and the columns represent the nodes. The matrix is visualized as a heatmap, where the darker shade is the less active nodes, and the lighter shade is the more active nodes.

Alternatively, we can use a heatmap to identify the most activated nodes, see Figure 4.2. In this case, activation is when the node activation value is greater than or equal to the node’s non-zero mean of its activation values. Equation 4.1 calculates the non-zero mean of a node’s activation values, where \( m \) is the number of activation values for the given node. We selected the non-zero mean equation because the NNs in this research use the ReLU function, producing many zeros. The significant number of zeros skew the average, making it smaller. The threshold for activation was too sensitive, so we decided not to include the zeros in the average.
Figure 4.1: Heatmap graph showing the node non-activations, where the darker colored shade represents the least activated nodes. The neural network has eight hidden layers and 16 nodes per layer. The dataset is 10,000 examples.

\[
NodeNonZeroMean = \frac{\sum_{i=1}^{m} v_i}{|\{i : v_i > 0\}|}
\]  

4.2.2 Graph for Influential Weights

A box plot helps visualize the distribution of data, including the data’s minimum, maximum, and median [10]. A box plot is a good choice for summarizing the weight values of a NN. We can see the distribution of each node’s weight values for a given layer with a box plot graph, including each node’s minimum, maximum, and median weight values.

If we are concerned with the activation of nodes, based on the activation equation 2.5,
larger weights will yield the most influence. Therefore, less influential nodes will be nodes whose max weight is closer to zero. Figure 4.3 demonstrates how we can use a boxplot to visualize the weight values. Nodes with the lowest maximum and median are prime candidates for pruning.

4.2.3 Weight Intensity Map

To better understand the information flow of a NN, it is important to see what features the NN is focusing on. Recall that in the activation equation 2.5, the higher the weight value the more influence the input has on the output activation of the node. We visualized the
Figure 4.3: Box plot of hidden layer three from a NN with eight layers and 16 nodes per layer. The plot shows the distribution of the absolute value of the node weights. Nodes 2, 3, 13, and 14 have the lowest max and median, making them ideal for pruning.

weight values as a heat map to show the input areas that are most important. This view is especially useful in the first hidden layer because it allows us to see what areas of the input image the node is focusing on.

Figure 4.4 is the map of a node that displays the magnitude of the weight for each pixel in the input image. The darker areas are the features the node is focusing on and the lighter areas are the features the node is ignoring. This example node was trained on the MNIST dataset and is looking for features of the digit seven (the dash and the slash).
Figure 4.4: Weight map of a node in the first hidden layer of a NN trained on MNIST dataset. The map shows the intensity of the weight for each pixel in the 28x28 pixel images. The darker areas are the features that the node is focusing on and the lighter areas are the features the node is ignoring.
CONTRIBUTION TO PRUNING TECHNIQUES

Pruning, as discussed in Chapter 3, is critical to the time and space complexity of a neural network: “What edges and nodes can be removed without compromising the overall performance of the network?” Pruning is an after-the-fact process as without any indication of how the network will perform, it is hard to predict exactly how many nodes per layer and which connections between these nodes are needed.

We have also recalled that pruning is often considered expensive. In this chapter, we propose pruning approaches that are effective and not expensive beyond the usual cost of training a network. We first explore pruning driven by activation (in Section 5.1), which can be easily visualized (see Chapter 4) and operated by a non-expert. We then consider weight values as indicators of the importance of the information traveling through them (in Section 5.2). We propose to prune nodes whose incoming or outgoing edges’ weights are insignificant. We describe our experimental approach below and report and analyze our results.

5.1 Pruning by Activation

5.1.1 Experiment: Pruning Nodes that are Never Activated

Once a neural network is trained, we observe that the information incoming to some nodes is filtered out by the activation function. It could be, for instance, that negative values are
passed and reduced to 0 through a ReLU activation function. This leads to question if these nodes are even important to the NN because mathematically they are not contributing to the NN. The following experiment explores this question.

Often, literature on pruning claims that it is expensive to identify nodes that need to be pruned [21]. What we claim is that it is not expensive the way we do it. How do we do it? We identify the activation at the same time as we evaluate the performance of the network.

What do we mean by “non-activated”?

Let us recall that the activation of a node is the value resulting from computing the weighted sum of the node’s inputs and then applying an activation function on the sum, see Equation (2.5). We evidently considered activation values of 0 to indicate that nodes were not activated.

A non-activated node has an activation value of zero and will have no impact on the next node it is connected to because zero multiplied with any weight value will always result in zero. It’s as if the node does not exist. Thus, the hypothesis is that non-activated nodes can be pruned from the NN without any performance degradation, i.e. loss of accuracy, because a non-activated node contributes a value of zero in the weighted sum. Theoretically, the neural network will produce the same result if the non-activated node is present or not present in the NN.

We also studied the effect of node removal given a threshold. We considered near-zero activation values to indicate that a node may not be critical in the network’s decision-making.

Pruning method

The pruning technique used in this paper does not remove or delete nodes from the NN. The “prune” technique in this paper is best described as disabling the node. The node is disabled by setting the weight values of the edges connected to the node to zero. By setting
the weights of the edges to zero the node does not propagate any information to the next layer. Theoretically, the math of the NN with the disabled node is the same as the math of the NN if the node was removed.

The approach of disabling a node was chosen because it’s similar to removing the node without the complications of dealing with ragged 2-D arrays. The only downside to this approach is that we don’t get the computational benefit of removing the node. However, we are only concerned with pruning techniques, we are not concerned with the computational speed or memory space after pruning.

Our experiment

We used the MNIST dataset to check our hypothesis. We trained 5 neural networks. These networks are described in Table 5.1. The ReLU activation function was used for the hidden layer nodes. The output layer has ten nodes because there are ten different classes, one for each digit (0-9). The hidden layers all have the same number of nodes to make visualization easier.

Training neural networks process:

1. Define neural network structure, see Table 5.1.

2. Train the neural network on the MNIST dataset.

3. Save the trained model.
Table 5.1: Structure of Neural Networks.

<table>
<thead>
<tr>
<th>NN</th>
<th>Layers</th>
<th>Nodes</th>
<th>Activation Function</th>
<th>Nodes</th>
<th>Activation Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>16</td>
<td>ReLU</td>
<td>10</td>
<td>Softmax</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>32</td>
<td>ReLU</td>
<td>10</td>
<td>Softmax</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>16</td>
<td>ReLU</td>
<td>10</td>
<td>Softmax</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>32</td>
<td>ReLU</td>
<td>10</td>
<td>Softmax</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>128</td>
<td>ReLU</td>
<td>10</td>
<td>Softmax</td>
</tr>
</tbody>
</table>

The experiment was performed as follows:

1. Test the NN and collect all the node output activation values.

2. Record the NN performance metrics.

3. Count the zero value node activation data, i.e. for each node count the number of times the activation value was zero.

4. Prune the nodes that are non-activated given some threshold.

5. Test the NN and record the NN performance metrics.

6. Retrain and test the NN. (optional)

Results

Below, we report the results we obtained:
Table 5.2: Results from pruning nodes that are never activated.

<table>
<thead>
<tr>
<th>NN</th>
<th>Original Accuracy</th>
<th>Pruned Nodes</th>
<th>Pruned Nodes %</th>
<th>Accuracy After Pruning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>94.64</td>
<td>17</td>
<td>13.28</td>
<td>94.64</td>
</tr>
<tr>
<td>2</td>
<td>95.60</td>
<td>14</td>
<td>5.47</td>
<td>95.60</td>
</tr>
<tr>
<td>3</td>
<td>93.37</td>
<td>19</td>
<td>7.42</td>
<td>93.37</td>
</tr>
<tr>
<td>4</td>
<td>95.22</td>
<td>47</td>
<td>9.18</td>
<td>95.22</td>
</tr>
<tr>
<td>5</td>
<td>97.56</td>
<td>10</td>
<td>1.95</td>
<td>97.56</td>
</tr>
</tbody>
</table>

Table 5.3: Results from pruning nodes that are mostly inactivated. Pruned nodes are selected based on a threshold $t$.

<table>
<thead>
<tr>
<th>NN</th>
<th>Original Accuracy</th>
<th>$t$</th>
<th>Pruned Nodes</th>
<th>Pruned Nodes %</th>
<th>Accuracy After Pruning</th>
<th>Accuracy after Retraining</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>94.64</td>
<td>0.9</td>
<td>27</td>
<td>21.09</td>
<td>94.62</td>
<td>94.88</td>
</tr>
<tr>
<td>2</td>
<td>95.60</td>
<td>0.9</td>
<td>50</td>
<td>19.53</td>
<td>95.57</td>
<td>95.70</td>
</tr>
<tr>
<td>3</td>
<td>93.37</td>
<td>0.9</td>
<td>37</td>
<td>14.45</td>
<td>93.06</td>
<td>94.42</td>
</tr>
<tr>
<td>4</td>
<td>95.22</td>
<td>0.9</td>
<td>93</td>
<td>18.16</td>
<td>90.67</td>
<td>95.60</td>
</tr>
<tr>
<td>5</td>
<td>97.56</td>
<td>0.9</td>
<td>54</td>
<td>10.55</td>
<td>97.56</td>
<td>97.44</td>
</tr>
</tbody>
</table>

Table 5.2 shows the results from pruning nodes that are never activated, i.e. nodes that always output a value of zero. We can see that our hypothesis is correct. The results show that the accuracy does not change when the never activated nodes are pruned. Therefore, non-activated nodes can be pruned from the NN without any performance degradation. For example, NN4 has an original accuracy of 95.22%, we pruned 47 nodes that were never activated and the accuracy remained the same at 95.22%.

Table 5.3 shows the results from pruning nodes that are mostly inactivated using a
threshold of 0.90. In this case, mostly inactivated nodes are nodes that are inactivated at least 90% of the time. The results show that this is a good approach for pruning since the accuracy after pruning and after retraining is comparable to the original accuracy. For example, in NN1 27 nodes were pruned, the accuracy after pruning only decreased by 0.02%, and accuracy increased 0.24% after retraining.

5.1.2 Experiment: Pruning Nodes that are Always Activated

Further, in considering activation, we posited that nodes that are always activated may not be meaningful either, and we ran experiments where we also removed these nodes.

What does it mean for a node to be “always activated”? A node is considered “always activated” when the node is always “on,” regardless of the input data fed to the neural network. An always-activated node is a node where all its activation values are non-zero. In other words, the difference between a node’s maximum and minimum activation values is smaller than the maximum activation value.

We propose Equation (5.1), where $A_{i,j}$ is the activation values for node $i$ in layer $j$, to find the always-activated nodes. The equation takes the difference between the max and min activation values and divides the difference by the max value. If the result is 0, the node is always activated, where the activation values for the node are always the same. If the result is less than 1, the node is always activated, where the activation values are always non-zero and within some range. If the result is 1, the node has been activated and not activated, meaning the activation values have been zero and non-zero.

Equation for the importance of a node. Only applies to nodes with a non-zero max activation value:

$$\text{NodeImportance}(i, j) = \frac{\max(A_{i,j}) - \min(A_{i,j})}{\max(A_{i,j})}$$ (5.1)

$$\text{AlwaysActivatedNode}(i, j) = (\text{NodeImportance}(i, j) < 1)$$ (5.2)
We hypothesize that if a node is always activated with the same activation value, the node is not essential to the network. In other words, regardless of the input class, the node always transmits the same information to the next layer. The node does not differentiate between classes and thus provides nothing to the classification of the input.

Additionally, in cases where all the node activation values are non-zero and fall within some range, we hypothesize that the smaller the activation range, the less important the node is.

**Our experiment**

In this experiment, we find always-activated nodes and prune them to test our hypothesis. The experiment method is as follows:

1. Prune never-activated nodes to prevent the weights for these nodes from changing during retraining. We want the NN to refrain from accidentally starting to rely on these nodes.

2. Calculate the importance of each hidden layer node using Equation (5.1).

3. Identify the always-activated nodes where all the activation values are the same.

4. Identify the always-activated nodes where the activation values fall within some range.

5. Prune the nodes found in step 3. Prune the nodes found in Step 4 if no nodes were found in Step 3.

6. Test the NN and record the NN performance.

7. Retrain the NN.

8. Test the NN and record the NN performance.
Results

The experiment results are shown in Table 5.4.

<table>
<thead>
<tr>
<th>NN</th>
<th>Original Accuracy</th>
<th>Pruned Nodes (never activated, always activated)</th>
<th>Accuracy After Pruning</th>
<th>Retraining Epoch Count</th>
<th>Accuracy after Retraining</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>94.64</td>
<td>17, 1</td>
<td>84.41</td>
<td>14</td>
<td>94.72</td>
</tr>
<tr>
<td>2</td>
<td>95.66</td>
<td>13, 5</td>
<td>87.04</td>
<td>28</td>
<td>95.73</td>
</tr>
<tr>
<td>3</td>
<td>94.94</td>
<td>19, 2</td>
<td>82.13</td>
<td>32</td>
<td>94.99</td>
</tr>
<tr>
<td>4</td>
<td>96.09</td>
<td>31, 8</td>
<td>83.51</td>
<td>31</td>
<td>96.34</td>
</tr>
<tr>
<td>5</td>
<td>97.47</td>
<td>13, 4</td>
<td>95.02</td>
<td>33</td>
<td>97.38</td>
</tr>
</tbody>
</table>

Table 5.4: Results from pruning nodes that are always activated.

Neural network one in Table 5.4 did not have an always-activated node where all the activation values were the same. However, it did have one always-activated node where all the activation values were within some range. The always-activated node had an importance score of 0.978 per Equation (5.1) on a scale of 0 to 1, meaning the range of the activation values was wide. This node can be considered less important than most nodes, but the wide range means it does plenty of work in the NN. The importance of this node is seen when the node is pruned because the accuracy drops by approximately 10%. However, the other nodes quickly make up for the missing node after retraining the NN for 14 epochs.

The results show that always-activated nodes with a range of activation values can be pruned from the NN with some retraining. The training is minimal, meaning the other nodes can do the work of the always-activated nodes.
5.2 Pruning as Influenced by Weights

Beyond the activation values of nodes, we looked into how weights of incoming edges to a given node N were playing a role in the significance of N.

**Influential weights**

Based on the node activation equation (2.5), it can be seen that the value of the weights defines the magnitude of the input to the node’s activation function. In this research, the activation function used for the hidden layer nodes is the ReLU activation function. This means that the node’s activation value will either be 0 or the weighted sum of the inputs, whichever is greater. From this understanding, we can hypothesize that nodes with smaller weights will produce a smaller output value and thus are less influential to the NN.

Equation (5.3) is used to find the importance of a node, which is the max value of the input edges weights. In Equation (5.3), \( W_{i,j} \) is the set of weight values of the input edges for node \( i \) in layer \( j \). The higher the weight value, the more influential the node is; the lower the weight value, the less influential a node is. The importance or influence of a node is directly proportional to the values of the node’s weights.

\[
\text{Importance}(i, j) = \max(W_{i,j})
\]

(5.3)

5.2.1 Experiment: Prune with threshold over all network

In this experiment, we use the weight values of the incoming edges as an indication of the importance of a node. Using Equation (5.3), we find the importance of each node in the network. The least essential nodes are the nodes with the smaller max value of their weights. We hypothesize that the least essential nodes can be pruned from the NN since they have less influence on the network.
Our experiment

We selected a threshold $t$ and applied it to all the node importance values. The least essential nodes are the nodes below the chosen threshold. Once the least essential nodes of the NN are identified, they are pruned. The experiment method is as follows:

1. Calculate the importance of each hidden layer node.
2. Define a threshold to apply to the entire NN.
3. Find nodes that are below the threshold.
4. Prune the identified nodes.
5. Retrain the NN.

Results

<table>
<thead>
<tr>
<th>NN</th>
<th>Original Accuracy</th>
<th>t</th>
<th>Pruned Nodes</th>
<th>Accuracy After Pruning</th>
<th>Retraining Epoch Count</th>
<th>Accuracy after Retraining</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>94.64</td>
<td>0.30</td>
<td>8</td>
<td>28.66</td>
<td>55</td>
<td>93.03</td>
</tr>
<tr>
<td>1</td>
<td>94.64</td>
<td>0.40</td>
<td>20</td>
<td>15.86</td>
<td>53</td>
<td>56.31</td>
</tr>
<tr>
<td>2</td>
<td>95.66</td>
<td>0.20</td>
<td>19</td>
<td>37.96</td>
<td>46</td>
<td>94.85</td>
</tr>
<tr>
<td>2</td>
<td>95.66</td>
<td>0.30</td>
<td>66</td>
<td>9.15</td>
<td>59</td>
<td>19.85</td>
</tr>
<tr>
<td>3</td>
<td>94.94</td>
<td>0.40</td>
<td>17</td>
<td>58.72</td>
<td>32</td>
<td>94.36</td>
</tr>
<tr>
<td>3</td>
<td>94.94</td>
<td>0.50</td>
<td>97</td>
<td>12.74</td>
<td>60</td>
<td>86.99</td>
</tr>
<tr>
<td>4</td>
<td>96.09</td>
<td>0.30</td>
<td>58</td>
<td>91.45</td>
<td>34</td>
<td>95.95</td>
</tr>
<tr>
<td>4</td>
<td>96.09</td>
<td>0.40</td>
<td>307</td>
<td>10.89</td>
<td>60</td>
<td>88.60</td>
</tr>
<tr>
<td>5</td>
<td>97.47</td>
<td>0.15</td>
<td>79</td>
<td>50.18</td>
<td>16</td>
<td>97.13</td>
</tr>
<tr>
<td>5</td>
<td>97.47</td>
<td>0.20</td>
<td>263</td>
<td>17.59</td>
<td>60</td>
<td>90.46</td>
</tr>
</tbody>
</table>

Table 5.5: Prune nodes based on a selected threshold.
Table 5.5 shows the results obtained from this experiment. The results show a significant decrease in the NN accuracy performance after pruning. Even after retraining, the accuracy of the NN is not recovered. The problem is that this pruning approach prunes most of the nodes in the first hidden layer.

Most nodes in the first hidden layer are identified as the least essential because their importance score, as defined by Equation (5.3), is the lowest compared to all the hidden layer nodes. For example, in NN1 with \( t = 0.4 \), 14 of the 16 nodes in the first hidden layer are pruned, producing an accuracy of 56.31% after retraining. With many of the first hidden layer nodes missing, the NN ignores most of the input data, resulting in a loss of NN accuracy. We conclude that a threshold should not be applied equally to all nodes in the NN.

5.2.2 Experiment: Prune \( k \) nodes per layer

From the previous experiment, we learned that most of the smaller weights are in the first hidden layer of the NN. Pruning nodes in the first hidden layer greatly affects the NN’s performance, as seen in the previous experiment. Thus, it’s unfair to treat all nodes in the NN as equal and use a threshold across all of them. We should look at nodes on a layer-by-layer basis and only compare nodes to other nodes in their respective layer.

Our experiment

In this experiment, we prune \( k \) number of nodes per layer with the smallest weights. The hypothesis is that for a given layer, nodes with smaller weight values are the least important nodes for that layer. This is assumed because the smaller weights produce a smaller activation value per Equation (2.5), i.e. these nodes are doing the least amount of work in the layer.

The experiment method is as follows:

1. Calculate the importance of each hidden layer node.
2. Identify the $k$ least important nodes per layer.

3. Prune the identified nodes.

4. Retrain the NN.

Results

<table>
<thead>
<tr>
<th>NN</th>
<th>Original Accuracy</th>
<th>$k$</th>
<th>Pruned Nodes</th>
<th>Accuracy After Pruning</th>
<th>Retraining Epoch Count</th>
<th>Accuracy after Retraining</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>94.64</td>
<td>2</td>
<td>16</td>
<td>61.42</td>
<td>102</td>
<td>94.73</td>
</tr>
<tr>
<td>1</td>
<td>94.64</td>
<td>3</td>
<td>24</td>
<td>47.54</td>
<td>35</td>
<td>94.44</td>
</tr>
<tr>
<td>1</td>
<td>94.64</td>
<td>4</td>
<td>32</td>
<td>41.21</td>
<td>136</td>
<td>94.42</td>
</tr>
<tr>
<td>2</td>
<td>95.66</td>
<td>4</td>
<td>32</td>
<td>93.70</td>
<td>30</td>
<td>95.35</td>
</tr>
<tr>
<td>2</td>
<td>95.66</td>
<td>8</td>
<td>64</td>
<td>86.79</td>
<td>30</td>
<td>95.40</td>
</tr>
<tr>
<td>2</td>
<td>95.66</td>
<td>16</td>
<td>128</td>
<td>38.91</td>
<td>37</td>
<td>94.64</td>
</tr>
<tr>
<td>3</td>
<td>94.94</td>
<td>3</td>
<td>48</td>
<td>42.51</td>
<td>127</td>
<td>94.59</td>
</tr>
<tr>
<td>3</td>
<td>94.94</td>
<td>4</td>
<td>64</td>
<td>34.04</td>
<td>114</td>
<td>94.13</td>
</tr>
<tr>
<td>3</td>
<td>94.94</td>
<td>8</td>
<td>128</td>
<td>12.56</td>
<td>40</td>
<td>89.99</td>
</tr>
<tr>
<td>4</td>
<td>96.09</td>
<td>4</td>
<td>64</td>
<td>93.10</td>
<td>38</td>
<td>96.01</td>
</tr>
<tr>
<td>4</td>
<td>96.09</td>
<td>8</td>
<td>128</td>
<td>62.17</td>
<td>16</td>
<td>95.56</td>
</tr>
<tr>
<td>4</td>
<td>96.09</td>
<td>16</td>
<td>256</td>
<td>12.20</td>
<td>74</td>
<td>94.87</td>
</tr>
<tr>
<td>5</td>
<td>97.47</td>
<td>19</td>
<td>76</td>
<td>96.80</td>
<td>14</td>
<td>97.43</td>
</tr>
<tr>
<td>5</td>
<td>97.47</td>
<td>32</td>
<td>128</td>
<td>90.10</td>
<td>16</td>
<td>97.48</td>
</tr>
<tr>
<td>5</td>
<td>97.47</td>
<td>64</td>
<td>256</td>
<td>36.78</td>
<td>17</td>
<td>97.08</td>
</tr>
</tbody>
</table>

Table 5.6: Prune $k$ Nodes per Layer based on Max weight values

Table 5.6 shows the results obtained from this experiment. The results from pruning $k$ nodes per layer show comparable performance to the original accuracy after some retraining. The
NN’s performance after pruning and retraining saw a minimal reduction in accuracy. For example, NN 3 has a total of 256 hidden layer nodes and we pruned 25 percent of nodes (64 nodes) which resulted in a 0.81% decrease in accuracy. Additionally, there were two instances where the accuracy saw a very slight increase. In NN1 we pruned 16 out of 128 hidden layer nodes and accuracy increased by 0.09%. In NN5 we pruned 128 out of 512 hidden layer nodes and accuracy increased by 0.01%.

When the results are compared to the previous experiment results, it can be seen that equal pruning on each layer is a better approach than applying a threshold across the entire NN, leading us to believe that nodes should only be compared to other nodes within their layer. This is understandable because in a fully connected neural network, the input data to each layer is different but the input data to each node in a given layer is the same.

The results of this experiment are promising. We can take the experiment further and attempt to find distinct \( k \) values for each layer. In other words, instead of uniformly pruning \( k \) nodes per layer, we can prune different numbers of nodes per layer based on some criteria. Lastly, we should avoid over-pruning nodes in the first hidden layer, since the first hidden layer is the layer that receives the input data. When we prune in the first hidden layer, we risk ignoring some features of the input data.
Chapter 6

EXPLORING FEATURE ARITHMETIC

6.1 Experiment: Visualizing parts of a digit

To gain a better understanding of how neural networks make decisions, we need to see how information travels through the neural network. We wanted to explore what would happen if a neural network is given an image that it’s not exactly trained for. We questioned what would happen if a neural network is given just a part of an image, i.e. one feature of the whole image. Would the neural network still classify the same as the whole image? Would the neural network activate the same set of nodes or a subset of nodes that were activated when the network is given the whole image? The following experiment explores these questions.

In this experiment, we took an image of a digit from the MNIST dataset and broke the digit down into its parts. We created separate images of the parts and fed the images to a trained neural network. Then we observed the activated nodes and created an activation map for the images of the parts. Finally, compared the activation maps of the parts with the activation map of the whole digit. We looked for similarities in the activation maps, checking if the same nodes are activated.

What is feature arithmetic?

The main idea is that the activation maps of the parts when added together will equal the activation map of the whole. We hypothesize that when we overlay the activation maps
Figure 6.1: Activation map for each digit class displays which nodes get activated and to what intensity. True Positive examples were used for the activation maps.
of the parts, the resulting activation map will bear some similarity to the activation map of the whole. The overlayed activation map will not be exactly the same as the activation map of the whole, but it will be similar.

**Our experiment**

This experiment used NN1 from table 5.1 and an image of a seven from the MNIST dataset. The experiment method is as follows:

1. Generate an activation map for each digit class, see Figure 6.1, using only True Positive examples, i.e. examples that were correctly classified.

2. Take an image of a digit and break it down into two parts to create two new images. Resulting in three images (whole, part1, and part2), see table 6.1.

3. Feed the three images to the NN and record the node activation values.

4. Generate an activation map for each image.

**Results**

Table 6.1 shows the results obtained from this experiment. The table displays the input images, the activation map of the network given the input images, and the classification of the image as a probability. The whole image seven was classified as a seven with 99.9% probability. The dash part of the seven was classified as a seven with 97.2% probability. The slash part of the seven was classified as a nine with 36.4% probability and as a seven with 16.8% probability.

From the results, we see that a subset of the activated nodes from the whole digit seven are activated with the dash image. Additionally, the neural network classifies the dash as a seven with 97.2% probability. The neural network NN1 considers the dash as a significant feature of the digit seven. The slash by itself is a less significant feature, the network believes it could be a nine or a seven.
<table>
<thead>
<tr>
<th>Name</th>
<th>Image</th>
<th>Activation Map</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td><img src="image1" alt="Activation Map for 7" /></td>
<td><img src="image1" alt="Activation Map for 7" /></td>
<td>7: 0.999</td>
</tr>
<tr>
<td>dash</td>
<td><img src="image2" alt="Activation Map for dash" /></td>
<td><img src="image2" alt="Activation Map for dash" /></td>
<td>7: 0.972</td>
</tr>
<tr>
<td>slash</td>
<td><img src="image3" alt="Activation Map for slash" /></td>
<td><img src="image3" alt="Activation Map for slash" /></td>
<td>9: 0.364, 7: 0.168</td>
</tr>
</tbody>
</table>

Table 6.1: Results for the feature arithmetic experiment on the image of a seven. The table displays the activation maps and classifications for images of a seven, a dash, and a slash.
Figure 6.2: The overlayed activation maps of the dash and slash images. The
overlayed activation map has some similarities to the activation map of the whole digit
seven. One notable difference is the lack of activation of nodes zero and six in the first
layer of the overlayed activation map. This means that there is some feature that the nodes
zero and six, in the first layer, are looking for but is missing in the dash and slash images.

The existence of feature arithmetic is inconclusive. We must be able to see what features
the first layer is focusing on in order to activate the correct nodes throughout the network.
From the activation map of the whole digit seven we gather that there must be some
significant features or characteristics that the first layer is focusing on. Characteristics of
the digit seven that activate nodes 0, 6, and 14. Theoretically, if we activate these nodes
simultaneously, the activations should pass through the network and output a seven.

6.2 Experiment: What makes a seven a seven?

Following the previous experiment, we want to see if it’s possible to activate specific nodes
in the first layer that will classify, or output, a desired class. In other words, what are
the features needed to activate a specific class? In this experiment, we look for the basic characteristics that make a digit a seven, according to a network trained on the MNIST dataset.

**Weight Intensity Maps**

To perform this experiment we need to see what are the features of interest for the nodes in the first hidden layer. A weight intensity map visualizes the weight values as a heatmap to show the most important input areas. The higher the weight value, the more influence the input has on the node’s output activation value, see Equation 2.5. Therefore, we are looking for areas with weights of higher value which are represented by the darker areas in the weight intensity map. Tables 6.2, 6.3, and 6.4 contain the weight intensity maps for the nodes in the first hidden layer of NN1 from Table 5.1. The weight intensity maps demonstrate which areas of the input image the specific node is focusing on. Each map is different which means that each node is looking for different features in the input image.

Each weight intensity map from the first hidden layer consists of 784 weights, represented as cells. Each cell corresponds to one pixel from a 28 × 28 pixel image from the MNIST dataset. In the weight intensity maps, the coordinates for a weight cell are the same as the coordinates for the pixel that the weight belongs to. In other words, each pixel has an assigned weight, and the weight corresponds to the same coordinates as the pixel from the input image, see Figure 6.3.
Figure 6.3: Weight at x,y coordinates in the weight map corresponds to the pixel at x,y coordinates in the input image.

The activation map for the digit class seven, shown in Figure 6.1, illustrates that nodes 0, 6, and 14 are the activated nodes in the first hidden layer. The importance of a node is seen by the node’s shade, node 14 is the darkest, and node 0 is the lightest of the activated nodes. When the nodes are ordered by their importance, they are 14, 6, and 0. Node 14 is looking for the top dash in the input image, as seen in Figure 6.6. Node 6 looks at three different areas of the input image, the top where the dash is supposed to be, the middle part, and the base part of the image, as seen in Figure 6.5. Node 0 is looking for the top dash and the slash of a seven simultaneously in the image, as seen in Figure 6.4.
Figure 6.4: Weight map for node 0 in the first hidden layer of NN1. The areas circled in red are the features of the input image that the node is focusing on.

Figure 6.5: Weight map for node 6 in the first hidden layer of NN1. The areas circled in red are the features of the input image that the node is focusing on.
Our experiment

This experiment used NN1 from Table 5.1. The experiment method is as follows:

1. Generate the weight intensity maps for each node in the first layer of the NN, see Tables 6.2, 6.3, and 6.4.

2. Using Figure 6.1, identify which nodes are activated in the first layer for the digit class.

3. For each activated node, identify which region in the weight maps the node focuses on, i.e. the region in the weight map where it’s the darkest.

4. Create a new image for that digit that only includes the darkest areas in the weight maps.

5. Feed the image to the NN and record the node activation values.

6. Generate an activation map for the image.
Table 6.2: Weight maps for nodes 0-5 in the first hidden layer of NN1.
Table 6.3: Weight maps for nodes 6-11 in the first hidden layer of NN1.
Table 6.4: Weight maps for nodes 12-15 in the first hidden layer of NN1.
Results

Table 6.5 shows the results obtained from this experiment. The table displays the input images, the activation map of the network given the input images, and the classification of the image as a probability. We came up with four cases of a decomposed seven using the weight intensity maps of nodes zero and 14. The weight intensity map of node six, Figure 6.5, was not utilized when creating the images of a decomposed seven because node six is more complex. Node six according to Figure 6.1 is used to classify images of the digits three, four, five, seven, eight, and nine. Since node six is used to classify a wide range of nodes, we only focused on nodes zero and 14 to reduce the complexity.

Case 1 in Table 6.5 is meant to activate node zero. Node zero’s weight intensity map shown in Figure 6.4, illustrates that the node is looking for the top dash with a small gap and a separate slash that does not reach the bottom of the image. Given the weight intensity map for node zero, we designed case 1 input image to have a top dash with a gap and a slash that does not reach the bottom of the image. The result is, the neural network classifies the image as a seven with a 98.8% probability. Visually, case 1 image is not a seven but it has characteristics of a seven. Leading us to conclude that the neural network does not look at the entire digit, it is only looking for specific qualities of a digit.

Case 2 in Table 6.5 is meant to activate node fourteen. Node 14’s weight intensity map shown in Figure 6.6, shows that the node is only looking for the top part of the seven or the top dash. Case 2 image is similar to case 1 image but without the slash part of the seven. When the case 2 image was fed into the network it was classified as a seven with a 97.1% probability. Case 2 image did not activate node zero in the first layer, see case 2 activation map in Table 6.5, because the input image does not have a slash. Case 2 image only activated node 14 in the first layer, which is explainable because node 14 looks for the top part of the digit seven. We conclude that our initial assumption, that node 14 is the most important node in classifying a seven, is correct because case 2 image resulted in a high probability of 97.1% that the image is a seven. Additionally, we conclude that NN1 only needs to see the top part of the seven to classify the image as a seven. In other words,
<table>
<thead>
<tr>
<th>Name</th>
<th>Image</th>
<th>Activation Map</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td><img src="image1" alt="Activation Map for 7" /></td>
<td><img src="image2" alt="Activation Map for 7" /></td>
<td>7: 0.999</td>
</tr>
<tr>
<td>case 1</td>
<td><img src="image3" alt="Activation Map for Decomposed 7 (case 1)" /></td>
<td><img src="image4" alt="Activation Map for Decomposed 7 (case 1)" /></td>
<td>7: 0.988</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9: 0.009</td>
</tr>
<tr>
<td>case 2</td>
<td><img src="image5" alt="Activation Map for Decomposed 7 (case 2)" /></td>
<td><img src="image6" alt="Activation Map for Decomposed 7 (case 2)" /></td>
<td>7: 0.971</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9: 0.018</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3: 0.005</td>
</tr>
<tr>
<td>case 3</td>
<td><img src="image7" alt="Activation Map for Decomposed 7 (case 3)" /></td>
<td><img src="image8" alt="Activation Map for Decomposed 7 (case 3)" /></td>
<td>3: 0.461</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7: 0.287</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2: 0.162</td>
</tr>
<tr>
<td>case 4</td>
<td><img src="image9" alt="Activation Map for Decomposed 7 (case 4)" /></td>
<td><img src="image10" alt="Activation Map for Decomposed 7 (case 4)" /></td>
<td>7: 0.885</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9: 0.057</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3: 0.019</td>
</tr>
</tbody>
</table>

Table 6.5: Results for the experiment on the basic characteristics of a seven. The table displays the activation maps and classifications for images of a seven and four cases of a decomposed seven.
the network sees the top part of the digit seven as the main identifier of the digit seven.

Case 3 in Table 6.5 is a variation of the dash example in Table 6.1. The dash gets classified as a seven with 97.2% probability. Once the dash loses some of its area, as in case three, the neural network loses confidence in its classification. Case 3 is classified as a three with 46.1% probability or a seven with 28.7% probability. The activated nodes in case three, see case 3 activation map in Table 6.5, represent a subset of the activated nodes in the activation map for digit three in Figure 6.1. This means case 3 activated the path in the network that classifies the image as a three.

Case 4 in Table 6.5 is a variation of case 2 but with a wider gap and keeping the vertical part of the top-right corner of the seven. Case 4 is classified as a seven with 88.5% probability. The classification probability decrease from case 2 to case 4 is evident, considering that case 4 has less area than case two.

From this experiment, we concluded that it is possible to activate specific nodes in the first hidden layer that will output a desired class in the output layer. Based on the weight intensity maps, we can extract the characteristics that the node is looking for. If we trigger an activation on a node in the first hidden layer, it will cause a chain reaction through the network and output the desired class to some probability.
Chapter 7

FULLY CONNECTED NEURAL NETWORK VISUALIZATION WEB APP

The research in the paper is meant for non-experts, software engineers, and data scientists who are interested in learning about neural networks. We want this research to be accessible to anyone who is interested in machine learning.

7.1 Web Application

We propose a web application that incorporates the experiments from this research to allow a user to run the experiments themselves. The application will allow the user to define the structure of the neural network, activation functions, training parameters, thresholds, and datasets. The application will generate the same visualization outputs shown in this research paper (master’s work), allowing the users to see the intensity of the parameters and how information travels through the neural network. Figure 7.1 displays the first prototype of the proposed tool.

The web application is a work in progress and is in the early stages of development. The source code is hosted on github, in the following URL https://github.com/jlpuebla/FCNN-Visualization-Tool. The following technologies are being used to develop the tool: Svelte, HTML, JavaScript, TypeScript, CSS, TensorFlow, and mpld3.
Figure 7.1: Initial prototype of the Feed-Forward Neural Network web application.
7.2 Current Solution

If someone is interested in running the experiments from this Master's Thesis work, they can access the source code written for the experiments. The programs are Google colab notebooks (ipynb files) and they are available on GitHub at the following URL https://github.com/jlpuebla/FCNN-Visualization-Experiments. The programs use the following technologies: TensorFlow, Keras, pandas, Matplotlib, and seaborn. The programs are fully functional and easy to run. However, the programs are not accessible to the average person, non-experts, or non-computer scientists, which is why a web application with a user interface is suggested.
Chapter 8

CONCLUSION AND FUTURE WORK

8.1 Conclusion

This master’s work was an exploration of fully connected feedforward neural networks through visualization and is our contribution to the explainable AI space. In the first half of this research, we explored pruning techniques and in the second half, we explored feature arithmetic and how information travels through the network.

The pruning experiments demonstrated that a neural network’s accuracy does not change when we prune nodes that were never activated. Also, the accuracy after pruning nodes that were mostly unactivated, with or without retraining, is comparable to the original network’s accuracy. The research also shows that always-activated nodes can be safely pruned with some retraining afterward. We also saw that pruning by layer is more effective than pruning across the entire network given a threshold. Lastly, we learned that if we trigger an activation on a node in the first hidden layer it will cause a chain reaction through the network. We can get a specific output when we identify what features the network is looking for in the first hidden layer.

8.2 Future Work

In regards to pruning, the following ideas are possible opportunities to take this research further. The technique to prune $k$ nodes per layer is effective. However, we could take
this approach further by finding the best $k$ value to use on a given network. Also, we can find a method to calculate the most appropriate $k$ value for each layer. Lastly, we should conduct experiments to see the importance of the first hidden layer and if we should avoid pruning from this layer. When we pruned many of the first hidden layer nodes, we saw a significant loss in the network’s accuracy which was not recovered even with retraining. We hypothesize that pruning from the first hidden layer results in the network ignoring parts of the input data which could be important for the decision-making process.
References


[12] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both Weights and Connections for Efficient Neural Networks. 2015.


[14] Luc Julia. There is no such thing as Artificial Intelligence. LP publisher, France, 2020.


EDUCATION

University of Texas at El Paso
Master of Science in Computer Science
Thesis: Towards Explaining Neural Networks: Tools for Visualizing Activations and Parameters

University of Texas at El Paso
Bachelor of Science in Electrical Engineering

RESEARCH EXPERIENCE

University of Texas at El Paso
Graduate Researcher with Dr. Martine Ceberio
Conducted experiments on fully connect feedforward neural networks involving pruning techniques and visualization of node activations and edge weights.

SKILLS

• Experienced in all aspects of the software development lifecycle, including software requirements, software design, software testing, and software maintenance. Experienced in agile software development methodology.

• Proficient in Python, Java, C++, SQL, C, Pascal, and Verilog/VHDL programming languages.

• Web development with HTML, CSS, JavaScript, Spring, and jQuery.

• Adept in software testing, including unit testing, integration testing, and regression testing. Testing experience using Junit, Jenkins, Mockito, Postman, and Jest.
• Experienced in Machine Learning methods such as K-NN, Naive Bayes, Decision Tree, Random Forest, Linear Regression, and Neural Networks. Programming with TensorFlow, Scikit-learn (sklearn), Keras, Pandas, NumPy, Matplotlib, and seaborn.

TEACHING EXPERIENCE

University of Texas at El Paso
El Paso, TX
Master Teaching Assistant, CS3 Data Structures
Fall 2023
Supported and guided students on their laboratory assignments, by answering their questions, helping them debug their programs, and providing feedback. Tutored students and hosted review sessions.

PROFESSIONAL EXPERIENCE

Sam’s Club
Dallas, TX
Software Engineer Intern – Sam’s Club Technology - Fresh
Summer 2023
Developed web application Café App using React, Next.js, Tailwind, and Jest.

Sam’s Club
Dallas, TX
Software Engineer Intern – Sam’s Club Technology - Replenishment
Summer 2022
Application Programming Interface (API) development for web application Replenishment Insights. API tested using Postman and wrote unit test cases. Developed and tested queries with Azure portal query editor.

Texas Instruments
Dallas, TX
Software Engineer - Test Technology Group - User Support Package (USP) 2015-2019
Supported Teradyne/Eagle Test Systems (ETS) Eagle Vision User Support Package (EVUSP) for ETS364 and ETS200. Resolved 20+ high-level tickets per major software release, including factory top issues, new features, process changes, and bug fixes. Trusted and relied upon by TI Test factories to provide software support and remote debugging. Developed key
features, including Automated Test Equipment (ATE) Inventory, Test Statistical Process Control (SPC) Rules, and Handler driver support for turret handlers. Increased EVUSP testing coverage by 34%. Spearheaded the implementation of regression testing for USPs. Trained engineers on how to create regression test cases. Participated in Universal USP development, including the development of Test SPC Rules code and unit testing. As USP and SPC expert, reviewed and approved pull requests for new unit test cases.

Texas Instruments
Dallas, TX
Test Engineer - Test Technology Group – Factory Test Solutions (FTS) 2013-2014
Solid State Relay (SSR) Modules project manager. In charge of finding device candidates using Test Overall Equipment Effectiveness (OEE) data. Delegating project tasks with TI Factories. Test solution conversion for analog-to-digital converter device resulting in Final Test (FT) yield improvement by +30%. Developed ATE Inventory Data program (generates RITdb file) for J750 tester.

Texas Instruments
Dallas, TX
Software Engineer - Test Technology and Product Engineering Group 2011-2012
Developed Bin Stability Tool, a web based application that generates bin yield reports using bin data from Test Ware (TW) databases. Application was written in Java, SQL, HTML, CSS, and JavaScript. Developed YieldGard application as proof of concept. Application executed at midnight and notified user by email if their test program/device yield was low. The email included a yield report. Assembly Checker (aka One Setup) checkout on Very Low Cost Tester (VLCT) in TI Mexico. Yield analysis on post-assembly semiconductor devices.