USER BULLETIN 7

Theoretical Basis of *ANN-Customize* ©: Customizable Artificial Neural Network Spreadsheet for General Use

1. INTRODUCTION

An Artificial Neural Network (ANN) is a mathematical tool that can learn the relationships between inputs and outputs based on a set of training data. ANNs can pick up on these relationships using weights and biases which mimic the behavior of synapses in the human brain. The process in which an ANN learns these relationships is known as training. Training an ANN is done using two different equation sets. The first one, known as the forward propagation, is used to calculate the ANN's guess at what the outputs should be based on the inputs. The second one, known as the backward propagation, is used to calculate a new set of weights and biases and to replace the old weights and biases to bring the ANN closer to the correct relationship. However, the creation and combination of these equations to make an ANN can be quite complex. In addition, ANNs are often only made for a single-use case. These barriers prevent the more prevent use of ANNs and hinders the abilities of those who wants to learn some of the benefits and limitations of ANNs.

This bulletin describes an Excel spreadsheet, *ANN-Customize*, that was developed to generate and train ANNs of any size. The purpose of *ANN-Customize* is to allow for an easy and intuitive way of creating ANNs while still giving as much control over the ANN as possible to the user. This allows the user to create and train ANNs and learn how the number of neurons, the layout of the neurons, and other variables affect the accuracy of the ANN. This tool is not just useful to help educate about how ANNs work but also to create and use an ANN for research and engineering applications. The theoretical basis and formulation being used by *ANN-Customize* to train ANNs are based on Almeida Jr. (2019), which uses a four-layer ANN. In order to allow *ANN-Customize* to have more layers or nodes, the closed forms of the equations developed for backward propagation are expanded in such a way that they could be scaled for any ANN. This document presents the expanded equations and discusses fundamental aspects of the *ANN-Customize*, which can be downloaded from this link.

This document is prepared by Andrew Coffman (third-year undergraduate student at the University of Toledo, OH, USA, as a part of a project supervised by Dr. Serhan Guner.

2. ANN-CUSTOMIZE LAYOUT

The first page that the user can see when opening *ANN-Customize* is the page where all of the inputs controlling the ANN are entered this page of *ANN-Customize* can be seen in **Figure 1**.

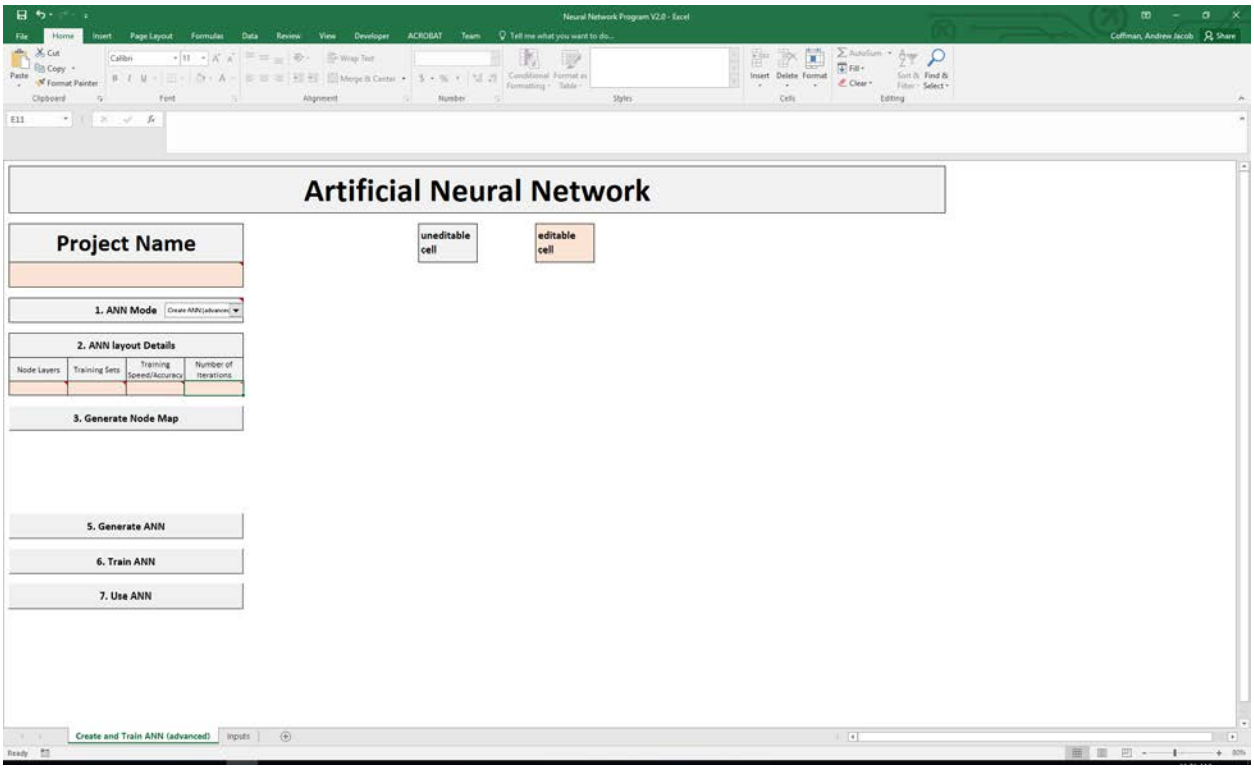


Figure 1 – The advanced training mode of *ANN-Customize.*

The user inputs on this page are as follows:

NL : Number of Node Layers [4 recommended]
DS : Number of Training Sets
η : Learning rate (between 0 and 1) [0.5 recommended]
I : Number of Iterations [2,000 recommended to start]
n_L : Number of neurons in layer L

After entering these variables and pressing the "Generate Node Map" button the user can also define the number of nodes within each layer and the number of inputs and outputs. For the simple ANN, these numbers are generated automatically based on the inputs and outputs tapering off from one to the other.

The second mode of the ANN is the Simple mode which is used for a quick way to debug parts of the program. This mode can be seen in **Figure 2**.

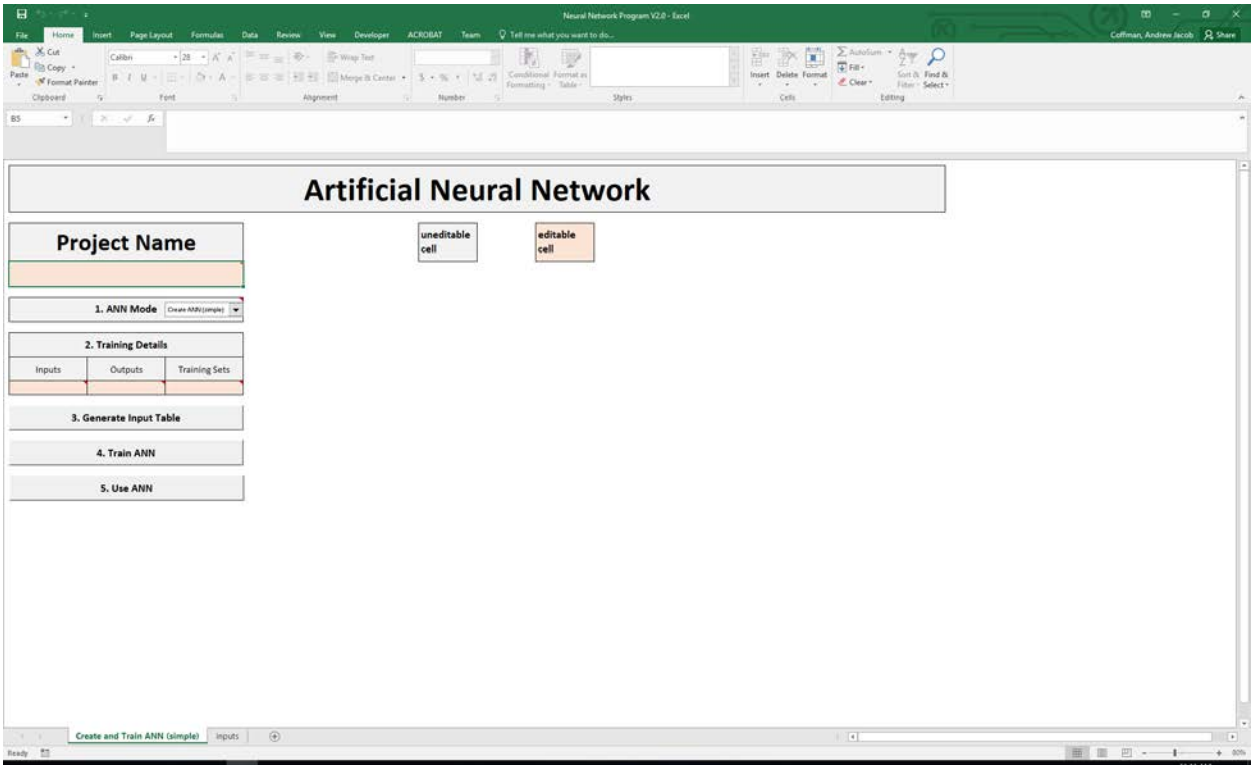


Figure 2 – The simple page of *ANN-Customize* meant for development.

In this mode, the user can easily modify the details of the training set that they will be using. The rest of the variables are handled within the code.

3. FORWARD PROPAGATION

The variables defined at the beginning of *ANN-Customize* are used throughout all the equations. The first use for these variables is within the creation of the equation set for forward propagation. Before explaining the equations being used, it is important to know their layout. **Figure 3** shows an ANN layout that could potentially exist with 4 layers. ANN layer one has 5 nodes; layer two has 3 nodes; layer three has 2 nodes; and the last layer has 1 node.

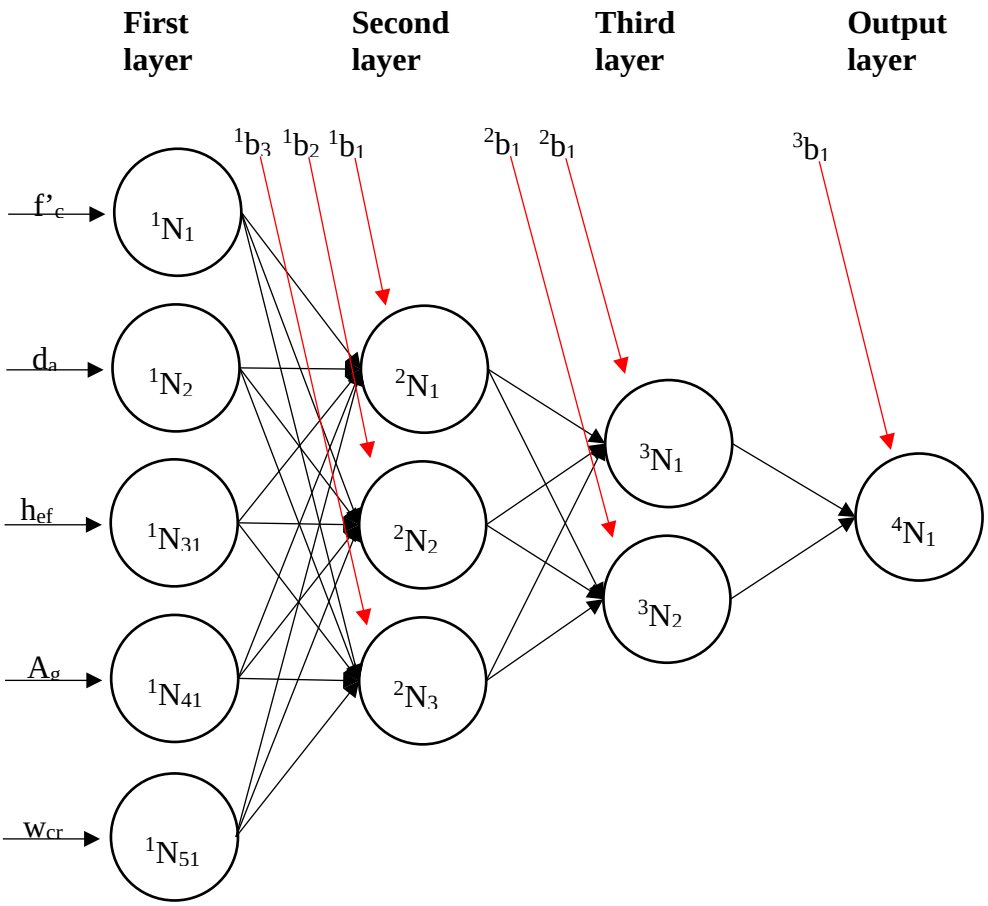


Figure 3 – ANN layout.

Figure 3 also shows the weights as the arrows connecting the nodes and the biases are also shown in red. To get a better idea, a single neuron that is not in the first layer is explained below where:

${}^{L}x_i$: Input of neuron *i* in layer *L*
${}^{L}y_i$: Output of neuron *i* in layer *L*

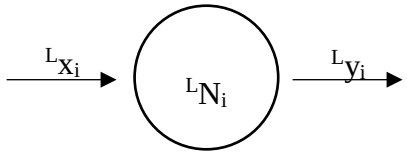


Figure 4 – Input and output of neuron *i* at layer *L*.

Figure 4 shows a single neuron with all the inputs combined using **Eq. 3** and giving the result of **Eq. 4** to the next set of nodes.

For L = 1,

$$^{L}x_i = I_i \quad (1)$$

$$^{L}y_i = f(^{L}x_i) \quad (2)$$

For 1 < L < OL

$$^{L}x_i = \sum_{k-1}^{L-1 n}(^{L-1}W_{ki}{}^{L-1}y_k) + {}^{L}b_i \quad (3)$$

The sigmoid is used as the activation function (Eq. 3). This function is often preferred due to some of its characteristics, such as: being continuous and smooth, assuming values between 0 and 1, and having a higher slope around the origin and decreasing slope far from it. These features make the sigmoid an appropriate function for the NN.

$$^{L}y_i = \frac{1}{1+e^{-{}^{L}x_i}} \quad (4)$$

4. NORMALIZATION

Due to how the sigmoid function only operates between the values of 0 and 1, all of the inputs and outputs had to be normalized in order to work with the sigmoid. The equation for this normalization is as follows:

$$I = 0.8 * \frac{UI - UI_{min}}{UI_{max} - UI_{min}} + 0.1 \quad (5)$$

where
I : normalized input
UI : user-defined input

The same equation is used in order to normalize all the outputs for the ANN as well.

5. BACKWARD PROPAGATION

5.1. Weight Error

The equation for the change in the weights used for backward propagation is a derivative of the total error with respect to the weight of interest (**Eq. 6**).

$$\delta^{L}w_{ij} = \frac{\partial E}{\partial {}^{L}w_{ij}} \quad (6)$$

where:
$\delta^{L}w_{ij}$: the amount that the weight between neurons *i* and *j* of layer L needs to change to make the ANN more accurate
${}^{L}w_{ij}$: the weight between neurons *i* and *j* of layer L
E : the total error of the ANN

The explicit form in Almeida Jr. (2019) covers up to four layers; however, in order to go beyond four layers, the equation would have to either be derived from the total error or a new equation based on the patterns seen when the total error is derived would need to be formed. This new equation is what is used within the *ANN-Customize*. For simplicities sake, this equation will be split up into equation segments and covered by pseudo-code. Before explaining the pseudo-code, an understanding of the patterns discovered is needed. In order to explain these patterns, an equation showing what a more complicated derivative of the total error with respect to a weight in layer one would look like will be explained. (**Eq. 7**)

$$\delta^{0}w_{0,0} = -\left(\mathrm{T}_{0,0} - \mathrm{N}_{3,0}\right) * \mathrm{N}_{3,0} * \left(1 - \mathrm{N}_{3,0}\right) * {}^{2}\mathrm{W}_{0,0} * \mathrm{N}_{2,0} * \left(1 - \mathrm{N}_{2,0}\right) * {}^{1}\mathrm{W}_{0,0} * \mathrm{N}_{1,0} * \left(1 - \mathrm{N}_{1,0}\right) * \mathrm{N}_{0,0} - \left(\mathrm{T}_{0,0} - \mathrm{N}_{3,0}\right) * \mathrm{N}_{3,0} * \left(1 - \mathrm{N}_{3,0}\right) * {}^{2}\mathrm{W}_{1,0} * \mathrm{N}_{2,1} * \left(1 - \mathrm{N}_{2,1}\right) * {}^{1}\mathrm{W}_{0,1} * \mathrm{N}_{1,0} * \left(1 - \mathrm{N}_{1,0}\right) * \mathrm{N}_{0,0} - \left(\mathrm{T}_{0,1} - \mathrm{N}_{3,1}\right) * \mathrm{N}_{3,1} * \left(1 - \mathrm{N}_{3,1}\right) * {}^{2}\mathrm{W}_{0,1} * \mathrm{N}_{2,0} * \left(1 - \mathrm{N}_{2,0}\right) * {}^{1}\mathrm{W}_{0,0} * \mathrm{N}_{1,0} * \left(1 - \mathrm{N}_{1,0}\right) * \mathrm{N}_{0,0} - \left(\mathrm{T}_{0,1} - \mathrm{N}_{3,1}\right) * \mathrm{N}_{3,1} * \left(1 - \mathrm{N}_{3,1}\right) * {}^{2}\mathrm{W}_{1,1} * \mathrm{N}_{2,1} * \left(1 - \mathrm{N}_{2,1}\right) * {}^{1}\mathrm{W}_{0,1} * \mathrm{N}_{1,0} * \left(1 - \mathrm{N}_{1,0}\right) * \mathrm{N}_{0,0} \quad (7)$$

where:
NL is equal to 4
n_L for all layers is 2
N : array holding results of the calculations from the forward propagation process
T : array holding all of the training data target outputs for all data sets

The equation has two parts that repeat with only changes to the nodes or weights that they reference. These two parts are as follows:

$$-\left(\mathrm{T}_{\mathrm{DS,CO}} - \mathrm{N}_{\mathrm{NL,CO}}\right) * \mathrm{N}_{\mathrm{NL,CO}} * \left(1 - \mathrm{N}_{\mathrm{NL,CO}}\right) \quad (8)$$

where:
CO : current node in the output layer
DS : data set of the target value from the test data sets

$$^{L}\mathrm{W}_{i,j} * \mathrm{N}_{\mathrm{L,i}} * \left(1 - \mathrm{N}_{\mathrm{L,i}}\right) \quad (9)$$

In order to use this information, a pattern should be found in how those references change. This pattern can also be seen in **Eq. 7**. The equation starts with a node in the output and takes a path from that node to the weight that is being updated. If there is more than one path from this output, it takes each different path. If there is more than one node in the output layer, it does this for every node in that layer. With those two patterns in mind, the pseudo-code segments can be explained. Within the pseudo-code in **Figure 5**, the two loops allow for the loop through all the nodes within the output layer and for all the ability to take a path from the output to the weight being changed. This segment of the equation also checks to see if the layer of the weight is within the last three layers since the equations within these layers apply the patterns slightly differently than how they are applied for the rest of the layers.

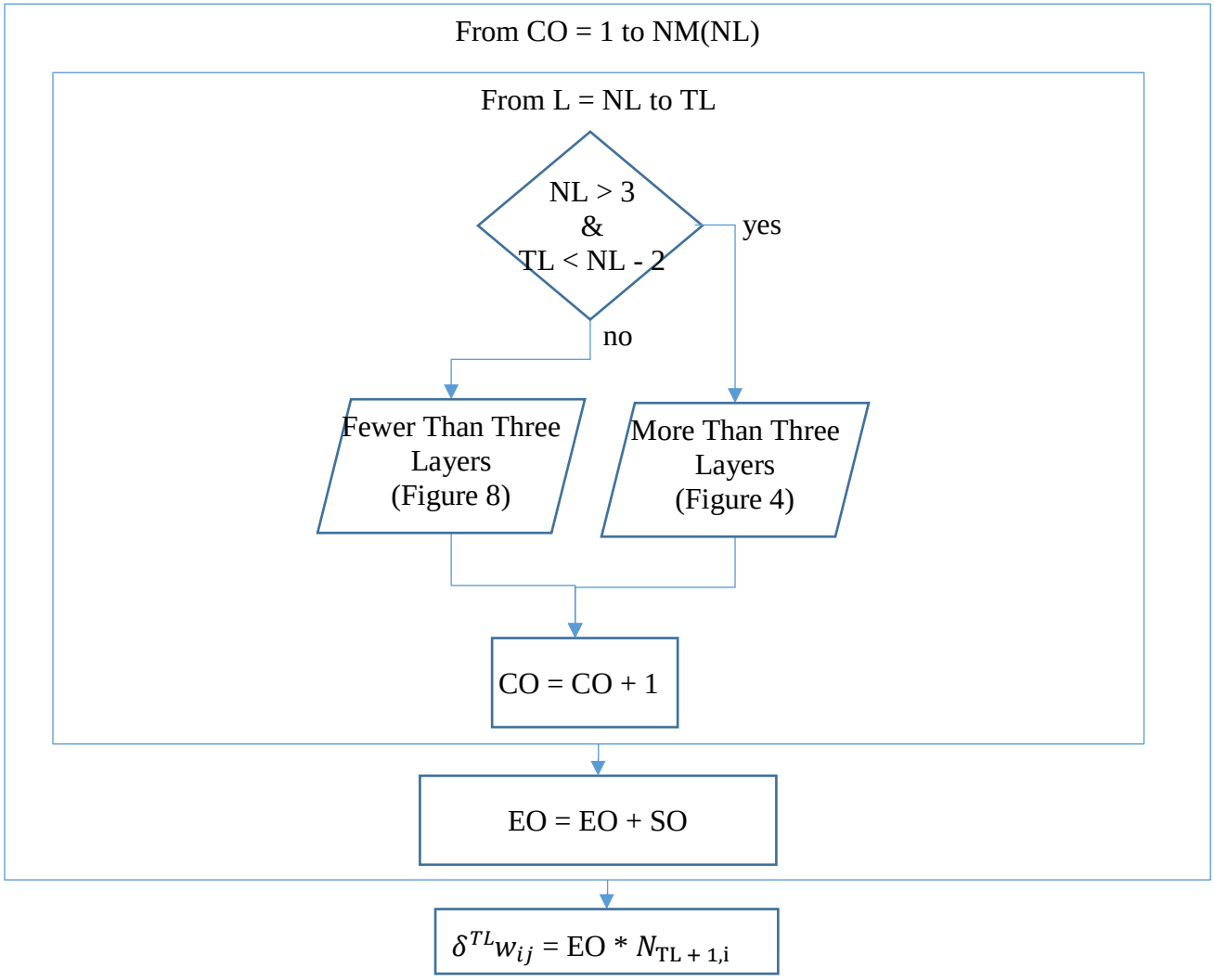


Figure 5 – The final combination of the weight error equation.

where:
CO : current node in the output layer
NM : array defining the shape of the ANN
EO : output variable from all of the equation parts
TL : the layer that the weight being checked is on
L : the current layer in the loop

If the layer that the weight is on is not within the last three layers, then the code in **Figure 6** runs.

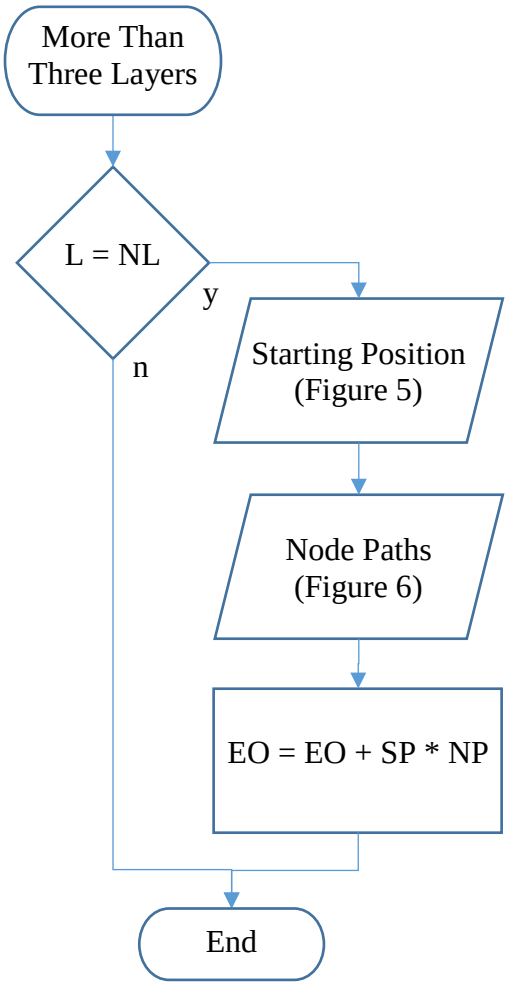


Figure 6 – Combines the equation parts for weights more than 3 layers from the output layer.

Since there is more than one path that the weight from the node in the output layer, the loop for handling that path will be done in the "Node Paths" code. This means that the loop from **Figure 5** running through all the Layers is not needed so that this segment of code ensures that the code running all possible paths will only be run once. The "Starting Position" segment of code is the first part of the equation that loops (**Eq. 3**) and can be seen in **Figure 7**, where SP is the starting position of output. The more involved part of the process is the "Nodes Paths" code in **Figure 8**.

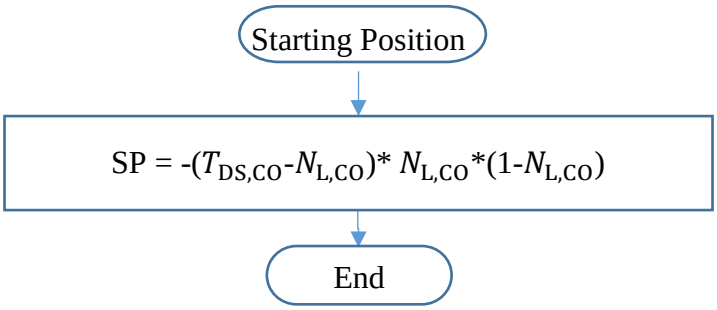


Figure 7 – Outputs of **Eq. 3** for the current output in the loop.

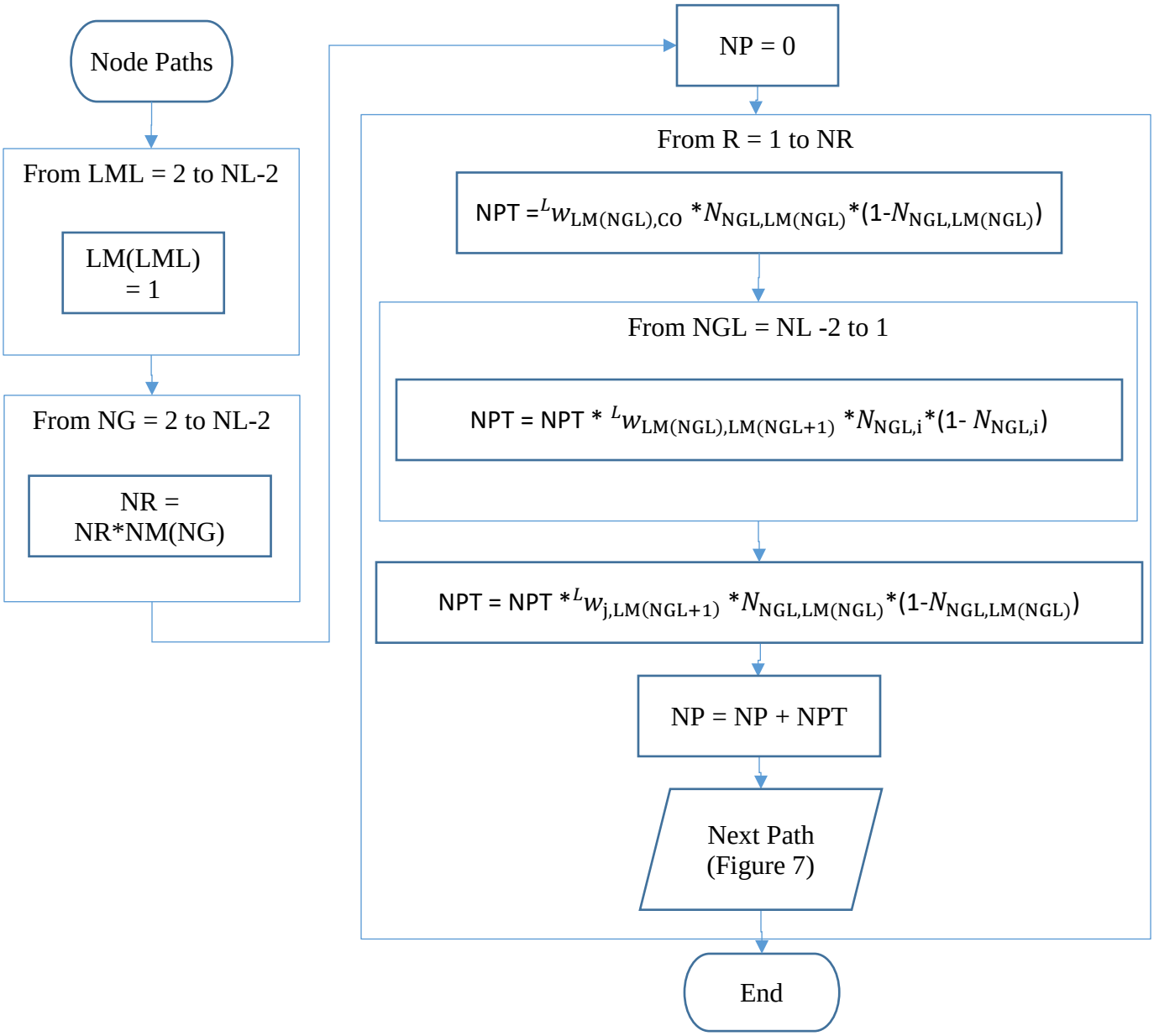


Figure 8 – Combines all possible paths from the output to the weight being changed.

where:
LM : array defining the path that the program is looking at from the output to the weight that is being changed
LML : the layer within the ANN that is corresponding to the node position being stored within
NR : The number of paths that the ANN will need to take through the ANN
NPT : temporary holder of the equation being output
NP : starting position output

The code first defines the path that the code will take from the output to the layer that the weight is on. This is done using the LML loop. The code then calculates how many times it needs to loop in order to allow every path in-between these two points using the NG loop. Now that everything is set up the code initializes the final output of the code and then in the loop, it handles the three different cases of how **Eq. 4** repeats in the total equation. These three cases are the layer directly after the output layer, the layer directly before the layer of the weight being changed, and all the layers in-between the other two cases. Afterwards, the output from these three cases combined is added onto the final output of this section of code and the next path for the loop to take is defined. This is done using the code segment "Next Path" as shown in **Figure 9**.

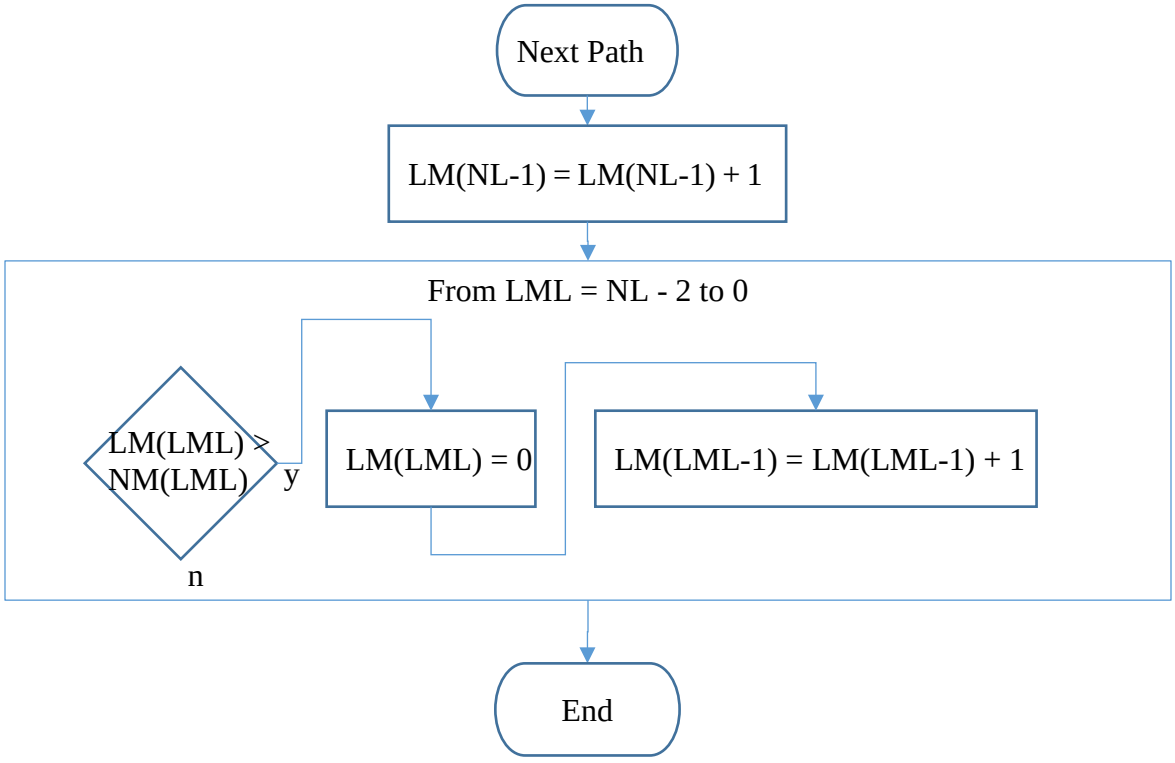


Figure 9 – Checks the node position for layer NL-1 ensuring that it is not larger than the number of nodes in that layer.

In order to change the path the code is taking through the ANN, the position in the layer directly before the output within the LM array is incremented and all of the values held within the array are checked to see if they are greater than the number of nodes within that layer. Ff they are greater, the value is set back to the first node in the layer and the layer before it in the array is incriminated.

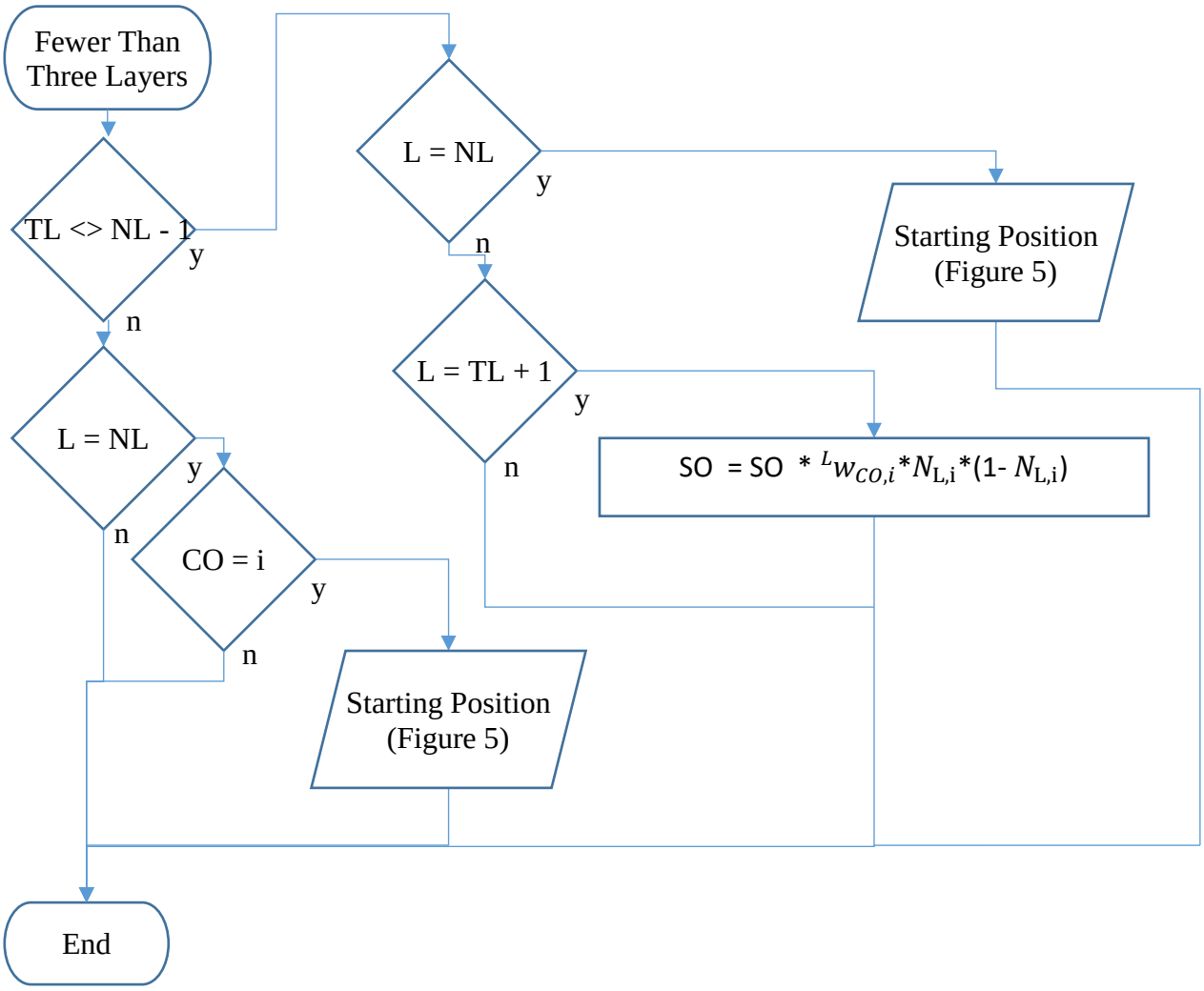


Figure 10 – The combination of equation parts for weights within the last three layers of an ANN.

Finally, the pseudo-code seen in **Figure 10** is only used if the weight that is being adjusted is within the last three layers of the ANN. This section of code uses the loop from **Figure 5** that runs through all the layers in the ANN to combine all its segments. There are three situations that the code checks for. The first is: if the weight that is being changed runs between the layer before the outputs and the output layer in which case it will go down the left side of **Figure 10**. The second is: if the weight is not connecting the node in the output layer to the layer before, but the layer

within the loop is currently at the output layer in which case the code will run down the right side of **Figure 10**. The final case is if the weight is not connecting the node in the output layer and is on the layer directly before the weight that is being changed in which case the code will run down the middle of **Figure 10**.

5.2. Bias Error

In order to fully update the ANN, the biases also need to be updated. The equation to find the error within the biases to update them is almost identical to that of the weights with one main difference. This difference is that it does not get multiplied by the node's value that it ends on; the equation for the bias error can then be covered by changing how the equation parts combine in **Figure 11**.

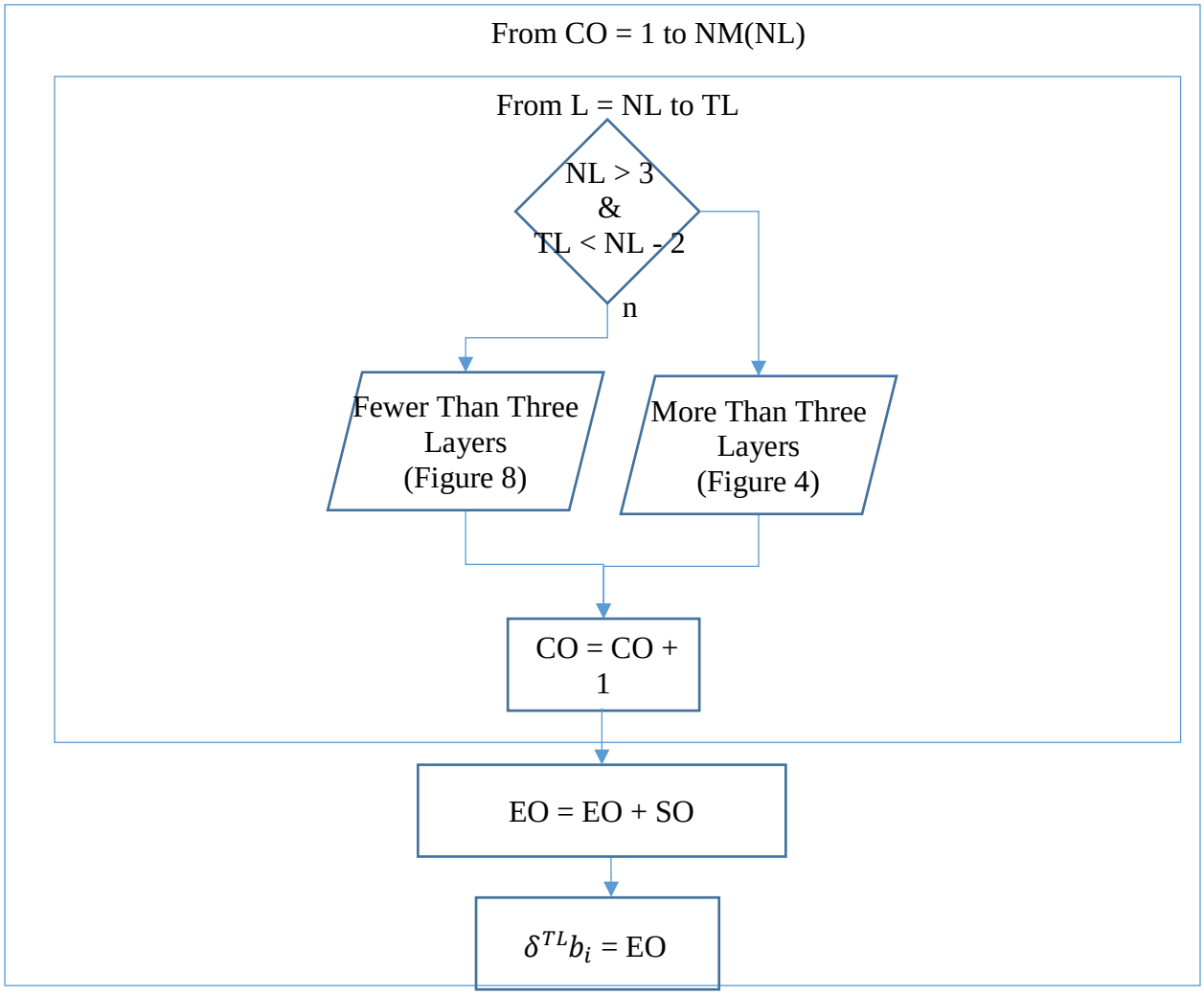


Figure 11 – The final combination of the bias error equation.

5.3. New Weights and Biases

Now that the error of the weights and biases is known, in order to find the new weight or bias, the error needs to be subtracted from the old weight or bias. Since the individual errors do not account for all of the weights and biases, if the error was applied to all of the weights and biases all at once, it would overshoot its target. In order to prevent this, the learning rate variable (η) defined in *ANN-Customize* is multiplied onto the error. The equation for a new weight is given in **Eq. 10** and the equation for a new bias is given in **Eq. 11**.

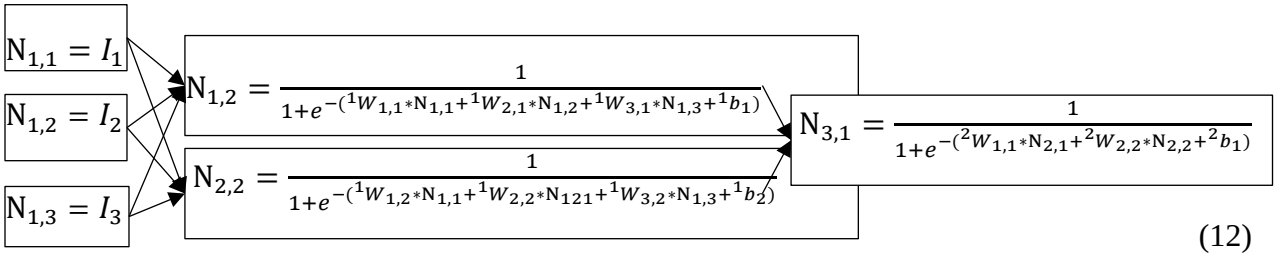
$$^{L}\mathrm{W}_{i,j} - \delta\,^{L}\mathrm{W}_{i,j} * \eta \tag{10}$$

$$^{L}\mathrm{b}_{i} - \delta\,^{L}\mathrm{b}_{i} * \eta \tag{11}$$

6. SAMPLE ANN EQUATIONS

As an example of how these equations work together, a 3-input, 1-output, and 3-layer ANN is formulated below.

6.1 Forward Propagation



(12)

6.2 Backward Propagation

6.2.1 Weights

$$\delta^{1}W_{1,1} = (-(\mathrm{T}_1 - \mathrm{N}_{3,1}) * \mathrm{N}_{3,1} * (1 - \mathrm{N}_{3,1}) * (^{2}\mathrm{W}_{1,1} * \mathrm{N}_{2,1} * (1 - \mathrm{N}_{2,1})) * \mathrm{N}_{1,1} \tag{13}$$

$$\delta^{1}W_{1,2} = (-(\mathrm{T}_1 - \mathrm{N}_{3,1}) * \mathrm{N}_{3,1} * (1 - \mathrm{N}_{3,1}) * (^{2}\mathrm{W}_{2,1} * \mathrm{N}_{2,2} * (1 - \mathrm{N}_{2,2})) * \mathrm{N}_{1,1} \tag{14}$$

$$\delta^{1}W_{2,1} = (-(\mathrm{T}_1 - \mathrm{N}_{3,1}) * \mathrm{N}_{3,1} * (1 - \mathrm{N}_{3,1}) * (^{2}\mathrm{W}_{1,1} * \mathrm{N}_{2,1} * (1 - \mathrm{N}_{2,1})) * \mathrm{N}_{1,2} \tag{15}$$

$$\delta^{1}W_{2,2} = (-(\mathrm{T}_1 - \mathrm{N}_{3,1}) * \mathrm{N}_{3,1} * (1 - \mathrm{N}_{3,1}) * (^{2}\mathrm{W}_{2,1} * \mathrm{N}_{2,2} * (1 - \mathrm{N}_{2,2})) * \mathrm{N}_{1,2} \tag{16}$$

$$\delta^{1}W_{3,1} = (-(\mathrm{T}_1 - \mathrm{N}_{3,1}) * \mathrm{N}_{3,1} * (1 - \mathrm{N}_{3,1}) * (^{2}\mathrm{W}_{1,1} * \mathrm{N}_{2,1} * (1 - \mathrm{N}_{2,1})) * \mathrm{N}_{1,3} \tag{17}$$

$$\delta^1 W_{3,2} = (-(\mathrm{T}_1 - \mathrm{N}_{3,1}) * \mathrm{N}_{3,1} * (1 - \mathrm{N}_{3,1}) * ({}^2\mathrm{W}_{2,1} * \mathrm{N}_{2,2} * (1 - \mathrm{N}_{2,2})) * \mathrm{N}_{1,3} \quad (18)$$

$$\delta^2 W_{1,1} = (-(\mathrm{T}_1 - \mathrm{N}_{3,1}) * \mathrm{N}_{3,1} * (1 - \mathrm{N}_{3,1}) * \mathrm{N}_{2,1} \quad (19)$$

$$\delta^2 W_{2,1} = (-(\mathrm{T}_1 - \mathrm{N}_{3,1}) * \mathrm{N}_{3,1} * (1 - \mathrm{N}_{3,1}) * \mathrm{N}_{2,2} \quad (20)$$

6.2.2 Biases

$$\delta^1 B_1 = (-(\mathrm{T}_1 - \mathrm{N}_{3,1}) * \mathrm{N}_{3,1} * (1 - \mathrm{N}_{3,1}) * ({}^2\mathrm{W}_{1,1} * \mathrm{N}_{2,1} * (1 - \mathrm{N}_{2,1})) \quad (21)$$

$$\delta^1 B_2 = (-(\mathrm{T}_1 - \mathrm{N}_{3,1}) * \mathrm{N}_{3,1} * (1 - \mathrm{N}_{3,1}) * ({}^2\mathrm{W}_{2,1} * \mathrm{N}_{2,2} * (1 - \mathrm{N}_{2,2})) \quad (22)$$

$$\delta^1 B_1 = (-(\mathrm{T}_1 - \mathrm{N}_{3,1}) * \mathrm{N}_{3,1} * (1 - \mathrm{N}_{3,1}) \quad (23)$$

7. PERCENT ERROR EQUATIONS

The percent that the calculated output is diverged from the target output for each data set is calculated using **Eq. 24**.

$$E_n = 100 * \frac{|N_{L,n} - T_n|}{T_n} \quad (24)$$

where:
n : output number
L : total number of node layers
E : percent error

For the total percent error of the ANN, the average of all the percent errors are taken with the top 20% of the values removed and the bottom 20% of the values removed. This ensures that any outliers that exist in the data set will be removed.

8. REFERENCES

[1] Almeida Jr, S. A. (2019), "Modeling of Concrete Anchors Supporting Non-Structural Components Subjected to Strong Wind and Adverse Environmental Conditions," MS Thesis, Department of Civil and Environmental Engineering, University of Toledo, Ohio, U.S.A. 105 pp. 68-79. <web link>