Introduction

Neurodegenerative diseases are common diseases that affect the cognitive abilities of those afflicted. Usually, this type of disease affects the structure of neurons, rendering them less useful, and kills neurons. Diseases such as Parkinson's or Alzheimer's are the result of a gradual degeneration or loss of neurons. In hopes of obtaining a better understanding of neurodegenerative diseases, we have based ourselves on a system that is designed to imitate neurons, the artificial neural network (ANN). We hope to model the effects of neurodegenerative diseases through looking at the effects of deleting nodes on an artificial neural network.

An artificial neural network is a computing system that is based on the biological neural networks inside living beings' brains. The artificial neural network provides a system where multiple algorithms work together to "learn" tasks such as pattern recognition. In an artificial neural network, the basic elements imitate real neurons. The basic structure of an ANN consists of layers of nodes each interconnected with each other through a certain weight and connected to a bias. The first layer is an input layer and the last layer is an output layer. In between, there are usually hidden layers that serve to transform the function of the input so that it can be used by the output layer. The node in an ANN acts like the body of a neuron, it receives information, processes it and according to the value obtained decides the degree at which it will be fired and if it will be fired at all. Inside of the nodes, to transform data, there is an activation function to introduce non-linearity into the system, which allows the system to transform the information it receives. There are a lot of popular types of activation functions such as TanH, sigmoid and ReLU. All the nodes are linked to each other with a certain strength defined by a certain value named the weight. On all nodes, there is a bias attached. The bias allows the system to function more efficiently, insures that not all of the input are 0, and increases possibilities in the model. Weights and bias are adjusted so that in the right answer can be transmitted at maximum signal.

To pass information from one neuron to the next, the neural network uses a process named forward feed.

$$x_n^l = \sum_k w_{kn}^l \, O_k^{l-1}$$

$$x_n^l = b_n^l + \sum_k w_{kn}^l \, O_k^{l-1}$$

$$O_n^l = \varepsilon(x_n^l)$$

First, the node calculates the sum all the weights (w) connected to the current layer (o) multiplied by the values of the nodes of the previous values. k represents the number of nodes in each previous layer i.e. the number of nodes the current node is connected to. n indicates which node it is within the layer. After calculating this sum, we add the bias to this sum. Finally, we pass this sum through an activation function, in our case, the sigmoid function ε to transform the function. The sigmoid function with the formula ($\frac{1}{1+e^{-x}}$) normalizes the values to between 0 and 1, providing an indication

of the degree at which the neurons are firing. 0 being not firing at all and 1 being firing at maximum.

For the nodes to learn the answer that they are supposed to obtain, an optimization method named backward propagation is used. The backward propagation adjusts weights and biases so that, at the output stage, the weight for the right answer is overall greatest as it will produce the least error down the error gradient.

To process the error gradient, we first calculate the error or the cost function by this formula. t represents the expected value for the current node j in the layer J and k represents the nodes in the previous layer K.

$$E = \frac{1}{2}\sum_{k \in K}(\mathcal{O}_j - t_j)^2$$

Then, we use the gradient descent technique to find the minimum where the error is smallest. To do this, we take the partial derivative of each variable in this cost function according to the weights try to find the minimum of the function, in other words where the error will be closest to 0. The following equations are the derivative of the previous function in accordance to the weights. t represents the previous layer and j represents the current layer.

For an output layer node $k \in K$

$$\frac{\partial E}{\partial W_{jk}} = \mathcal{O}_j \delta_k$$

where

$$\delta_k = \mathcal{O}_k(1 - \mathcal{O}_k)(\mathcal{O}_k - t_k)$$

For a hidden layer node $j \in J$

$$\frac{\partial E}{\partial W_{ij}} = \mathcal{O}_i \delta_j$$

where

$$\delta_j = \mathcal{O}_j(1 - \mathcal{O}_j)\sum_{k \in K} \delta_k W_{jk}$$

As for the biases represented here by θ, they are treated as the same as weights as they function similar to an additional weight. However, the equation for the bias applies to any layer. Thus, to update the biases and the weights, the following formulas are used. The η represents the learning rate, in other words, how fast the code is processed.

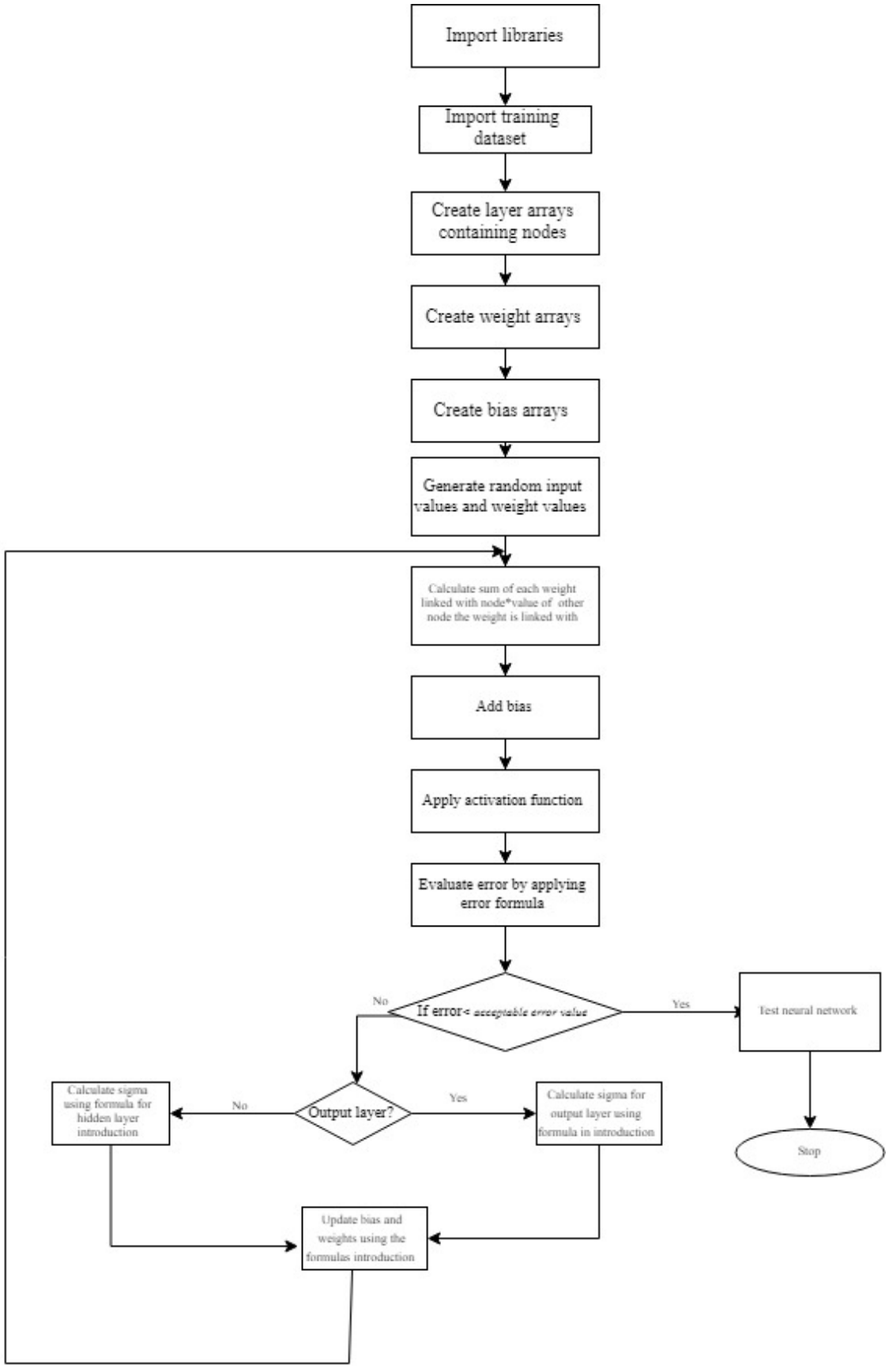
$$\Delta W = -\eta \delta_\ell \mathcal{O}_{\ell-1}$$
$$\Delta \theta = \eta \delta_\ell$$

$$W + \Delta W \to W$$
$$\theta + \Delta \theta \to \theta$$

This process is repeated until sigma (the error) reaches an acceptable error value near 0.

Procedure

To imitate neurons and their workings, we are going to construct an artificial neural network with Java. Then, through the process of supervised learning, our system is going to learn number recognition. We are going to provide our program with the MNIST data set which consists of 60 000 samples of handwriting for training and 10 000 samples for testing. Through the continuous process of forward feed and backpropagation, the program will learn to recognize numbers. We are going to calculate how accurate our system is by the percentage of times the system recognizes the numbers wrongly. Then, like neurodegenerative diseases that destroy neurons, we are going to delete a few nodes, which, in an artificial neural network imitates neurons. Finally, we will take the same number of tests as we did with the original sample and we are going to look at the effects produced on the output of the network. We are also going to calculate the percentage of errors made by the program after the deletion of a given number of nodes.


Import libraries
Import training dataset
Create layer arrays containing nodes
Create weight arrays
Create bias arrays
Generate random input values and weight values
Calculate sum of each weight linked with node*value of other node the weight is linked with
Add bias
Apply activation function
Evaluate error by applying error formula
If error< acceptable error value
No
Yes
Test neural network
Stop
Output layer?
No
Yes
Calculate sigma using formula for hidden layer introduction
Calculate sigma for output layer using formula in introduction
Update bias and weights using the formulas introduction