Simplified Artificial Neural Network vs. In-built Neuralnet Function in R

Boysen Mutembwa
3083294
MacEwan University
STAT324: Computational Statistics
2021

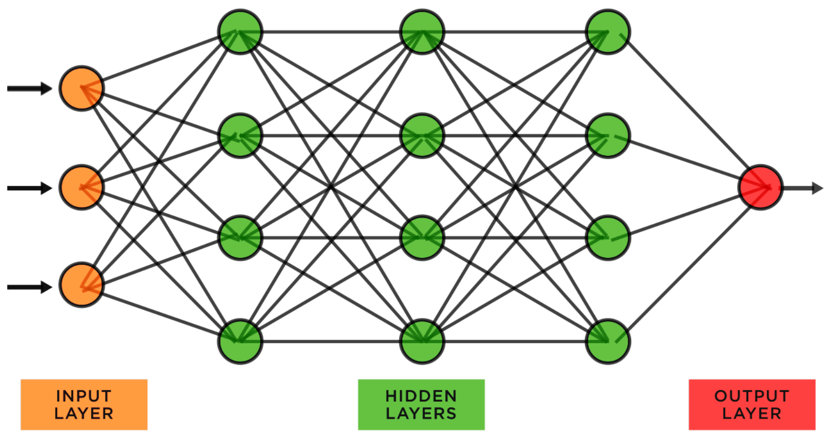
Abstract

In disciplines such as pattern recognition and game play, an artificial neural network (ANN) blends biological principles with modern statistical techniques to solve complicated problems. There are several methods in which ANNs might implement the essential concept of neuron analogues; in this report, we will build an artificial neural network from scratch and train it to a loan dataset to see if we can duplicate some of foundations of neural networks. We'll see whether the MSE of the neural network function and the programmed artificial neural network are comparable. The programmed ANN is a collection of multiple mathematical functions derived from Neurons, Weights, Backpropagation, and learning rate; the functions in the model have certain limits; for example, it can only train models with two hidden layers and can only utilize the sigmoid activation function. The purpose of this report is to demonstrate how to apply a combination of mathematical functions to construct an ANN model capable of forecasting output parameters and aiding in the best selection of machining parameters for process planning and optimization, in this case loan data. The ANN model did not perform well compared to the in-built function NeuralNet().

Introduction

There has been a 20-year gap between the conception of neural networks and the discovery of how to train them. It wasn't until 2010 that neural networks began winning contests that the notion of backpropagation was introduced in the 1960s. Neuronal networks have been on the increase since then because of their almost miraculous abilities, such as picture captioning, language translation, audio and video synthesis and more. Many of the most complex technology challenges, such as self-driving vehicles, risk assessment, fraud detection, and early cancer diagnosis, are now solved using neural networks (D.Kukieła et al , 2018).

Humans have long sought to replicate the functioning of the brain in order to improve lives. A general grasp of what is going on in the brain has been developed, despite the fact that we do not know all that is going on. Our best attempt has been to approximate it by using mathematical functions, which has shown to be beneficial. The human brain contains over 100 million neurons and 60 trillion synapses, which is a large number. Individual neurons like for humans cannot be reproduced by computers since they would outgrow computational units. Neural networks have been our most successful effort to create a computer function that is as near to human processing as possible (Education, 2020). Neural networks, sometimes referred to as artificial neural networks (ANNs), are a subset of machine learning that form the basis of deep learning techniques; their comprised of node layers, containing an input layer, one or more hidden layers, and an output layer (Q.Zhong et al., 2021). In contrast to other machine learning approaches, the interconnectedness of hundreds or thousands of neurons provides connections and outputs that typically surpass other methods.



An artificial neural network is composed of three layers of neurons in its simplest form. Information is transmitted between layers in the same manner as it is in the human brain:

- *The input layer: the point of entry for data into the system.*
- *The hidden layer: data is processed*
- *The output layer: where the system decides what action to take next based on the data.*

Dendrites: Input to the cell (Approximated to the input)
Synopses: It is the weight assigned to the input (Approximated to the weight or slope)
Axon: Output to a neuron (Approximated to the final output)

A neural network's capacity to learn is a critical component. Not only is a neural network a complicated system, but it is also a complex adaptive system, which means that it may modify its internal structure in response to the information passing through it. Typically, this is accomplished by modifying weights; a weight is a parameter in a neural network that specifies how the hidden layers of the network translate incoming data . Each line in the figure above depicts a connection between neurons and denotes the channel for information to pass. Each connection has a weight, which is a numerical value that determines the strength of the signal between the neurons. There is no need to alter the weights if the network produces a satisfactory output. However, if the network produces substandard output, the system adjusts by adjusting the weights to enhance future outcomes (D.Kukieła, 2018).

The strength of neural networks is derived from the hidden layer. The variables used to create each node may be seen as a property of the node itself. The weights of each node in the hidden layer are linked to one of the input variables. It is analogous to coefficients in a regression model since each node is an expression of the combination of input variables. After that, a non-linear function is applied to the linear combination (J.Loy, 2019). A neural network is made up of layers and nodes, which are often shown graphically in the form of intricate network diagrams. It is possible to think of a feed forward neural network as a stacked logistic regression model when the activation function is utilised, making it easy for anybody who is acquainted with regression to understand (D.Kukieła, 2018). It is the non-linear component of the model that lends depth and uniqueness to the data so that non-linear patterns may be detected (J.Loy, 2019).

Applications of Artificial Neural Network

Artificial neural networks may be used to solve a variety of engineering and scientific challenges (D.Kukieła, 2018). The following application areas are possible:

i. *Curve fitting on a global scale:* The objective is to reconstruct the functional connection between variables (often real numbers) in a given system from a known set of meaningful values. These applications are as broad as possible and often include mapping processes that are difficult to describe conventionally.

ii. *Process supervision:* This application area is concerned with finding control actions capable of achieving specified quality, efficiency, and security standards. Among the many applications available, neural controllers are particularly useful in robots, aircraft, elevators, appliances, and satellites, to name a few.

iii. *Recognition/classification of patterns:* The objective is to map a particular input pattern (sample) to one of the previously specified classes, as in image, voice, or writing recognition. In this situation, the task at hand has a distinct and well-defined set of viable outcomes.
iv. *Clustering of data:* The objective in this case is to recognise and characterise the similarities and peculiarities of the various input patterns in order to facilitate their categorization (clustering). To name a few, applications involving automated class identification and data mining are examples.

v. *Forecasting system:* This system category's objective is to forecast the future values of a certain process by analysing multiple past samples seen in its domain. Among the well-known applications are those for time series prediction, stock market projection, and weather forecasting.

vi. *Optimization of the system:* The purpose is to reduce or maximise a cost function (objective) while adhering to any limitations that may exist in order to appropriately map a problem. The most significant optimization tasks that may benefit from artificial neural networks include restricted optimization issues, dynamic programming, and combinational optimization.

vii. *Memory associative:* The purpose is to reconstruct a proper pattern from unclear or faulty inner parts. Several instances include image processing, signal transfer, and recognition of written characters.

The main objective of this paper is to examine the mathematical functions used to generate artificial neural networks (ANNs), and to compare a simplified two-hidden-layer neural network to the built-in neuralnet() function in R. Typically, neural nets are supervised models, which means that the input data set contains labels or some dependent variable that we want to predict with the greatest accuracy possible (J.Loy, 2019). In this report we will use the dataset below:

The data set HMEQ reports characteristics and delinquency information for 5,960 home equity loans. A home equity loan is a loan where the obligor uses the equity of his or her home as the underlying collateral. The data set has the following characteristics:

- BAD: 1 = applicant defaulted on loan or seriously delinquent; 0 = applicant paid loan
- LOAN: Amount of the loan request
- MORTDUE: Amount due on existing mortgage
- VALUE: Value of current property
- REASON: DebtCon = debt consolidation; HomeImp = home improvement
- JOB: Occupational categories
- YOJ: Years at present job
- DEROG: Number of major derogatory reports
- DELINQ: Number of delinquent credit lines
- CLAGE: Age of oldest credit line in months
- NINQ: Number of recent credit inquiries
- CLNO: Number of credit lines
- DEBTINC: Debt-to-income ratio

The credit card department of a bank is looking to streamline the process of approving home equity lines of credit using automation. The Equal Credit Opportunity Act's guidelines for constructing an empirically developed and statistically sound credit scoring model shall be followed.

The models we've developed will be based on data from recent loan applicants who were approved utilising the current loan underwriting method. The Mean Square Error (MSE) will be used to assess the models' accuracy.

Methodology

Implementation of the simplified Artificial neural network:

To make a simplified Artificial Neural Network we will have to apply a combination of mathematical functions and that have:

i. *Input signals* $(x_1, x_2, ..., x_n)$ are the signals or samples coming from the external environment and representing the values assumed by the variables of a particular application. The input signals are usually normalized in order to enhance the computational efficiency of learning algorithms.
ii. *Weights* $(w_1, w_2,..., w_n)$ are the values used to weight each one of the input variables, which enables the quantification of their relevance with respect to the functionality of the neuron.
iii. Linear aggregator gathers all input signals weighted by the synaptic weights to produce an activation voltage.
iv. *Bias* (*h*) is a variable used to specify the proper threshold that the result produced by the linear aggregator should have to generate a trigger value toward the neuron output.
v. *Activation potential* is the result produced by the difference between the linear aggregator and the activation threshold. If this value is positive, i.e. then the neuron produces an excitatory potential; otherwise, it will be inhibitory.
vi. *Activation function* whose goal is limiting the neuron output within a reasonable range of values, assumed by its own functional image (In this case the sigmoid)
vii. *Output signal (y)* consists of the final value produced by the neuron given a particular set of input signals and can also be used as input for other sequentially interconnected neurons.

The code which I translated to R code and some the explanation of the code was found in D.Kukieła & H.Kinsley (2018) Neural Networks from Scratch in Python book.

<u>Input layer</u>

First, we will simplify how the data gets into the Simplified Artificial neural network,
the X matrix is identical to the design matrix in a regression model, except that the first column has one for the bias term. I've labelled the first row β (Beta) differently to highlight that it's the bias parameter; the reason for the differing labelling is because the bias term is unaffected by the preceding layer; it is simply added to each layer. This might be interpreted as the node being

devoid of inbound connections, in contrast to the other nodes (Kinsley, 2018). The number of columns in W_1 denotes the number of nodes in the hidden layer.

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{13} \\ 1 & x_{21} & x_{22} & x_{23} \\ \vdots & & & \vdots \\ 1 & x_{n1} & x_{n2} & x_{n3} \end{bmatrix}; \quad W_1 = \begin{bmatrix} \beta_{01}^1 & \beta_{02}^1 & \beta_{03}^1 & \beta_{04}^1 \\ w_{11}^1 & w_{12}^1 & w_{13}^1 & w_{14}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 & w_{24}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 & w_{34}^1 \end{bmatrix};$$

$$W_2 = \begin{bmatrix} \beta_{01}^2 & \beta_{02}^2 \\ w_{11}^2 & w_{12}^2 \\ w_{21}^2 & w_{22}^2 \\ w_{31}^2 & w_{32}^2 \\ w_{41}^2 & w_{42}^2 \end{bmatrix}; \qquad \hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_{11} & \hat{y}_{12} \\ \hat{y}_{21} & \hat{y}_{22} \\ \vdots & \vdots \\ \hat{y}_{n1} & \hat{y}_{n2} \end{bmatrix}$$

The initialization of the weight matrices is the first stage. This is accomplished by doing random drawings from a uniform distribution distributed throughout the interval [-1,1].

$$W \sim U(-1,1)$$

Code:

```
weight.1 <- matrix(runif(ncol(X)*hidden.layers[1], -1, 1), nrow = ncol(X))
weight.2 <- matrix(runif((  hidden.layers[1]+1)*hidden.layers[2], -1, 1), nro
w = hidden.layers[1]+1)
weight.3 <- matrix(runif(( hidden.layers[2]+1)*ncol(y), -1, 1), nrow = hidden
.layers[2]+1)
```

Hidden layer

The first calculation is to construct the hidden layer

Mathematical function:

$$A_1 = XW_1$$
$$H = f(A_1)$$

The activation function is denoted by f, which is often the logistic function (sigmoid) or ReLU (Kinsley, 2018). There are several activation functions, and some are more effective with specific types of data. The Authors of "Neural Networks from Scratch in Python book" indicates that ReLU converges more rapidly (M.Hodnett, 2018). We will first create a function, a sigmoid activation function used to scale the output back to between 0 and 1, rendering it a logistic function:

Mathematical function:

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

Code:

```
sigmoid <- function(x) {
```

```
return(1/(1+exp(-x)))
}
```

We wrap the equation for new neurons with the activation, multiply summarization of the result of multiplying the weights and activations

Mathematical function:

$$h_{ij} = f\,(B+W1X1+W2X2+\ldots+WnXn)$$

The nodes in the hidden layer will function as input to the output layer. They may be thought of as created features from the input layer. This is something that a standard regression model cannot achieve; it is the strength of neural nets that they reveal hidden patterns in the data. As with the input layer and matrix X, a column of 1's must be added to H to account for the bias introduced by the connections between the hidden and output layers (Kinsley, 2018).

Mathematical function:

$$\bar{\mathbf{H}} = \begin{bmatrix} 1 & h_{11} & h_{12} & h_{13} & h_{14} \\ 1 & h_{21} & h_{22} & h_{23} & h_{24} \\ \vdots & & & & \vdots \\ 1 & h_{n1} & h_{n2} & h_{n3} & h_{n4} \end{bmatrix}$$

Outer layer

Mathematical function:

$$A_2 = \widetilde{H}W_1$$
$$\hat{y} = f(A_2)$$

Code:

```
hidden2 <- cbind(matrix(1, nrow = nrow(X)), sigmoid(hidden1 %*% weight.2))
y_hat <- sigmoid(hidden2 %*% weight.3)
```

Initially, the output will be imprecise. Adjust the weights in the direction that minimises the mistake. This is accomplished by backpropagation. To begin, the error function's derivative is determined (Wikipedia, 2019). Typically, this is the sum of squares or the standard deviation of the mean square error. It is irrelevant whatever component is chosen since the critical component is the difference between the real and forecasted values. When the weights are modified, the learning rate may take care of the constant parameters (M.Hodnett, 2018). Often, the sum of squares is multiplied by 1/2 for convenience, since it disappears when the derivative is calculated.

Mathematical function:

$$C = \frac{1}{2} \sum_{ij} (y_{ij} - \hat{y}_{ij})^2$$

Code:

```
mse[k] <- 1/nrow(y)*sum((y-y_hat)^2)
```

To alter the weights in the direction of the gradient, the partial derivative of the cost function with respect to the weights must be determined. This is accomplished via the application of the chain rule (M.Hodnett, 2018). The following table summarises the adjustment necessary between the hidden and output layers.

The errors are multiplied by the output gradient, which means that the weights are altered in the direction that minimises the error.

Mathematical function:

$$f'(x) = f(x)(1 - f(x))$$

Code:

```
d.sigmoid <- function(x) {
return(x*(1-x))
}
```

Mathematical function:

$$\delta_y = (\mathbf{y} - \hat{\mathbf{y}}) \cdot f'(\mathbf{A}_2)$$

Code:

```
hidden2_del <- y_hat_del %*% t(weight.3)*d.sigmoid(hidden2)
```

Similarly, we must locate the problem in the buried layer. We have no idea what the real values of the buried layer are since the model generates them (M.Hodnett, 2018). Rather than that, we utilise the output error to determine the needed modification in the hidden layer. The following code propagates this issue back to the hidden layer (Q.Zhong, 2021).

Mathematical function:

$$\delta_h = \delta_{\mathbf{y}} \mathbf{W}_2^T \cdot f'(\mathbf{A}_1)$$

Code:

```
hidden1_del <- hidden2_del[,-1] %*% t(weight.2)*d.sigmoid(hidden1)
```

Using these equations, the weight matrices are now updated as follows.

Mathematical function:

$$\begin{aligned}\mathbf{W}_2^{t+1} &= \mathbf{W}_2^t + \gamma\bar{\mathbf{H}}^T \cdot \delta_y \\ \mathbf{W}_1^{t+1} &= \mathbf{W}_1^t + \gamma\mathbf{X}^T \cdot \delta_h\end{aligned}$$

Code:

```
W3 <- weight.3 + l*t(hidden2) %*% y_hat_del
W2 <- weight.2 + l*t(hidden1) %*% hidden2_del[,-1]
W1 <- weight.1 + l*t(X) %*% hidden1_del[,-1]
```

where γ is the learning rate. We add the new weights to the old weight and these steps are then repeated until convergence or the maximum number of iterations has been reached.

To do this, the difference between the output predicted values and the real observed values is propagated back through the model using backpropagation. The weights are modified in a way that minimises the output inaccuracy (M.Hodnett, 2018). This is repeated until a specified level of tolerance or the maximum number of allowable repetitions is reached. From there we can output the mean square error, when the function is called with the right parameters.

Implementation of the in-built neuralnet () function:

Where we will compare our simplified artificial neural networks using the neuralnet () function. Oftentimes, while working with neural nets, effort is required to fine-tune the parameters, such as the number of hidden layers, the size of the hidden layers, the learning rate, and so on, in order to reach the optimum outcome; here we will not do as much tweaking need to make a good model:

Code:

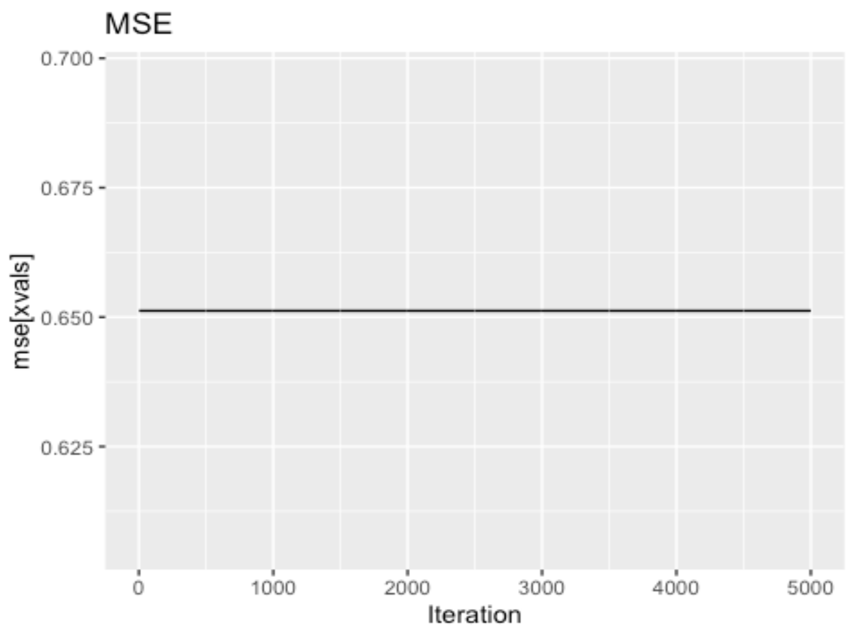
```
nn.loan <- neuralnet(BAD ~ ., data = df.clean,  hidden = c(6,6) , linear.outp
ut = F hidden = hidden.layers,algorithm = "backprop", learningrate = lr, act.
fct = "logistic")
```

Hidden layers 6 by 6 and a learning rate of 0.02 are used in this case.
Both conditions were used for both models.

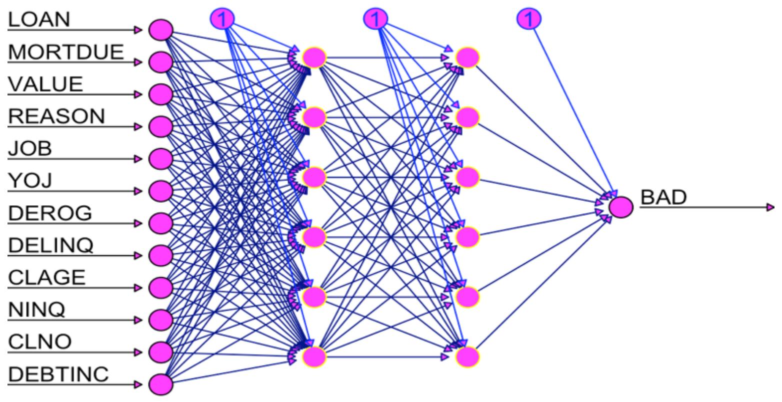
Results

After fitting the simplified ANNs and the built-in neuralnet () function in r, I discovered that the MSE for the Simplified ANNs was 0.6512207 after executing my simplified ANNs function with a learning rate of 0.02, 5000 iterations, and 6 by 6 hidden layers.

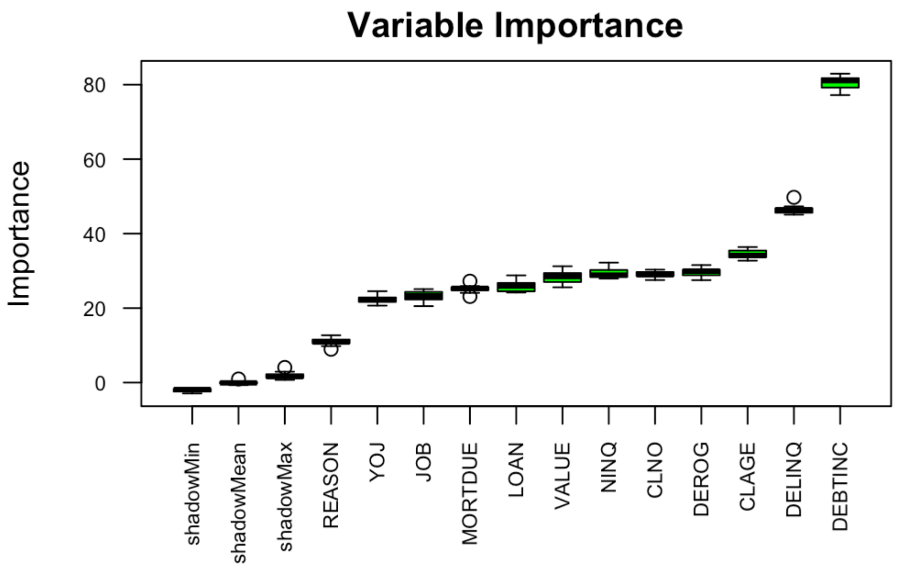
```
X.loan <- as.matrix(df.n)
Y.Bad <- model.matrix(~ BAD- 1, data = df.n)
out.loan <- NNet(X.loan, Y.Bad, hidden.layers= c(6, 6) , n = 5000, l = 0.02)
```



Which is not that good, especially when comparing to the in-built neuralnet () function that had 0.0856.



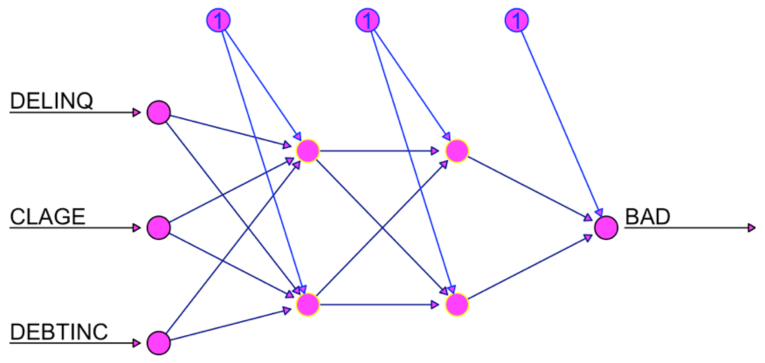
From the results I decide to use a Boruta package, "Boruta algorithm is a wrapper built around the random forest classification algorithm. It tries to capture all the important, interesting features you might have in your dataset with respect to an outcome variable" ().



Took the top three variables in the loan dataset as determined by the Boruta algorithm. After that, I chose to normalise the variables, since standardising scores makes it simpler to compare them, even if they were assessed using different scales. Additionally, it simplifies the reading of data and assures that all variables contribute to a scale when combined.

- DELINQ: Number of delinquent credit lines
- CLAGE: Age of oldest credit line in months
- DEBTINC: Debt-to-income ratio
- BAD: 1 = applicant defaulted on loan or seriously delinquent; 0 = applicant paid loan

Using the variables BAD, DELINQ, CLAGE and DEBTINC to fit the simplified ANNs and the built-in neuralnet () function in R, by reducing the number of hidden layers to 2 by 2 for both models. I discovered that the MSE for the Simplified ANNs improved drastically from 0.6512207 to 0.111. Which is the closest the MSE has been to the in-built neuralnet () function MSE of 0.0912



Discussion

Results show that using in-built neuralnet() package produces superior results, with predicted variables that are more closely related to the desired observed variables. After a few tweaks, the Artificial Neural Network was almost as good as the in-built neuralnet() function. When the model, simplified Artificial Neural network, had too many variables with a big variation, an adjustment was made to remove them from the simplified model. It didn't matter how many hidden layers I used to reduce the Mean Square Error for the simplified model; received the same results. There was no proof that the hidden layers impacted the Mean Square Error. Due to GPU-induced uncertainty, a specific artificial neural network has an improved ability to learn (A.Krizhevsky, 2014). Using a CPU instead of a GPU, the learning error is larger than when training the same number of epochs using a GPU, demonstrating that the GPU plays a distinct function in the learning process than merely improving computing speed (A.Krizhevsky, 2014). Another advanced technique that was not used to improve the model's performance was pruning, which is the process of reducing the number of nodes in a network (but not the number of layers) in order to improve computation and resolution performance. These solutions aim to eliminate nodes from the network during training by identifying those that would have a negligible effect on network performance (Z.Liu, 2018). As can be shown, creating a neural network using mathematical functions has precedence. As this may be very beneficial for creating neural networks that process particular data, such as COVID-19 data or predicting when a natural disaster is about to happen; changing the weights or the hidden layer to process data especially for the purpose of the research needed at hand can result in new discoveries and better decision-making.

Future Work

A considerable degree of pattern recognition-like abilities, which are required for pattern identification and decision-making, is provided by artificial neural networks (ANNs). These ANNs are robust classifiers that can generalise and make choices from large, complicated input data sets. Artificial neural networks (ANNs) are the finest when it comes to solving nonlinear problems. A system is easy to understand if the function used to describe it is linear. Looking more into other forms of neural network like Convolution Neural Networks (CNN) and Recurrent Neural Networks (RNN), to make more specified models that could give further understanding optimizing decision making. For example, fitting a model that predicts the nature of the market or a model that is used to analyze handwritten character

References

A.Krizhevsky, I.Sutskever, R.Salakhutdinov, N.Srivastava, G.Hinton.(2014) Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15.

Education, I. C. (2020, August 17). What Are Neural Networks?. What are Neural Networks? IBM. https://www.ibm.com/cloud/learn/neural-networks.

J.Loy. (2019). Neural Network Projects with Python : The Ultimate Guide to Using Python to Explore the True Power of Neural Networks Through Six Projects. Packt Publishing.

D.Kukieła , H.Kinsley (2018). Neural Networks from Scratch in Python (1st ed.). Copyright © 2020 Harrison Kinsley.

M.Hodnett, J.Wiley. (2018). R Deep Learning Essentials : A Step-by-step Guide to Mathematics Of Artificial Neural Networks

Q.Zhong, Z.Zhang, Q.Qiu, & X.Cheng. (2021). Roulette: A Pruning Framework to Train a Sparse Neural Network From Scratch. IEEE Access, 9, 51134–51145. https://doi.org/10.1109/ACCESS.2021.3065406

Wikipedia. (2019, August 18). Mathematics of artificial neural networks - Wikipedia. https://en.wikipedia.org/wiki/Mathematics_of_artificial_neural_networks.

Z.Liu, M.Sun, T.Zhou, G.Huang, and T.Darrell.(2018) Rethinking the value of network pruning. arXiv preprint arXiv:1810.05270. URL https://arxiv.org/ab s/1810.05270.