Logic Gate Classification by Using Multilayer Artificial Neural Network in C Programming Language

Emre Erdin

Software Engineering

Istanbul Aydin University

Istanbul, Turkey

emre@emreerdin.com

Abstract—**Logic Gate Classification by Using Multilayer Artificial Neural Network in C Programming Language is a study in which libraries which ease implementation of artificial neural network algorithm were not used and the implementation process has been done from scratch. The main aim of the algorithm is to solve XOR Logic Gate problem. The program has been implemented in a way that firstly weights of the neurons are initialized randomly between 0 and 1, then the algorithm takes its inputs and generates outputs by using Sigmoid Activation Function and Gradient Descent algorithm. By feed forwarding, the program calculates its inputs and weights, sends it to the activation function, then the output goes through hidden layer and output layer. The resultant output is compared with expected output. Then the error is calculated and back propagated to the neurons by applying Chain Rule. For 10.000 epochs and 0.1 learning rate, the accuracy score of the artificial neural network is closely %96.**

Keywords—Artificial Neural Network, ANN, Machine Learning, Data Mining.

I. Introduction

Logic Gate Classification by Using Multilayer Artificial Neural Network in C Programming Language is a study in which artificial neural network structure is implemented in computer-based system to solve a specific logic gate result classification problem. The aim is to determine the output of any logic gate considering its inputs. For any logic gate such as OR, AND, NAND, XOR, NOR and others, the program responds in the accurate way as soon as correct input and output sets are provided since it is based on classification and uses labeled data. The algorithm generates one input layer with two input neurons for taking two inputs of the logic gate such as 0 and 1, then one hidden layer with two hidden layer neurons for providing non-linearity and finally one output layer with one output neuron. The predefined input set is provided to the system for forward feeding, then the given inputs are multiplied by their weights provided by the neuron and then the output is sent to the hidden layer. Each hidden layer neuron takes the output and multiplies this activated value with the respective output neuron weights and activates it. After the activation of this multiplication, the calculated output and the expected output are taken into the calculation for error which is also known as loss. The value of the error is derivatized by applying Gradient Descent algorithm. Then the calculated function and the result of delta is back propagated to the neurons in order to adjust the weights and the bias. The program completes its epochs and prints the result after the number of epochs has been made by it.

II. Methodology

A. Declaration of Neural Layers and Neurons

Declaration of neural layers and neurons have been done by using either one dimensional array or two-dimensional array in C programming language. The initial values of the layers and the neurons have been set as null.

B. Value Assignment of Weights, Bias and Data Set

Values of the weights and the biases were completed by using a function that was declared in the program. The function called *InitiliazeWeights generates random values between 0 and 1 for assignment purposes of the weights. Then the bias is set as null first to initially calculate after the back propagation and dataset is set as below.*

C. Feed Forwarding

In order to calculate an output for the given inputs, the program takes a random input and after multiplying it by the value of its related weight, it sums up all the weighted sums and sends it to the activation function of each hidden layer.

Each *Input Layer* neuron takes the inputs provided by the dataset. Then the inputs given to the input layer neurons are multiplied by the weights that the system randomly generates in the first step. Then the multiplied input values by weights added together to get the *Weighted Total*.
Then, the weighted total is sent to the Sigmoid Activation Function to get the value between 0 and 1.

The selection of Sigmoid Activation Function over others is because Sigmoid Activation Function provides a probability for a prediction and yields between 0 and 1.

D. Error Calculation

Error calculation is the next step which is completed after all inputs and weights are multiplied, summed and sent to the output layer. Output layer takes the final value and acts it as the predicted output. Then calculates the loss by using *Mean Squared Error* function.

E. Back Propogation

After the error calculation, the result is sent into a function in order to have the slope of the function. The slope of the function is used in order to determine the optimal point for the weights which need to be adjusted. In order to get the slope of the result function, by using Chain Rule, the derivative of the activation function was taken to determine the delta value of the error.

The delta calculation is done for both the output layer and the hidden layer neurons. After the delta calculation, the value calculated is used to adjust the weights by using the following formula
Where learning rate is the value of the step size which is taken to find the optimal point on the Gradient Descent curve.

III. Codıng

A. Programming Language and Libraries

In this project, C programming language was used to implement the neural network algorithm for the classification of the logic gate result. For fast computing and compiling purposes, C programming language was chosen and for libraries standart libraries which C programming language provides were used. Since *Stdio.h* is the header library of any C program, mostly for input and output purposes, it was also included in the project. *Stdlib.h* library is used to include functions such as memory allocation, process control and conversion operations of data which is also known as *typecasting*. In order to achieve mathematical operations such as being able to use exponential function value, *Math.h* library was included and finally to be able to generate random variables *Time.h* library is used.

B. Algorithm

The main logic behind the algorithm works in a way that
the program declares the neuron number of each layer by using #define syntax. Then we declare each layer by using the defined lengths as each index represents a neuron itself. Then by using a function that generates random values between zero and one, we initialize our weights. Then we assign these weights to the hidden layer

connections and output layer connections. After that iteration number is requested from the user in order to provide feed forward and back propagation. By taking the iteration number, the program also requests another input for the learning rate which determines the step size of the gradient descent. After the interaction of the user to the system, the model starts its training by randomly selecting input rows and calculating the *Weighted Total* to send it to the activation function. Once the first activation function that is included in the hidden layer receives the weighted total, the value is sent to another function to calculate the Sigmoid value of the calculated value. After this operation, again another activation for the output layer is done by using the activated value and finally the back propagation starts.

IV. Conclusıon

This project was beyond the limits of my knowledge for artificial neural network mathematics and implementation in C. Yet, at the end, it was much more obvious that how a neural network algorithm must be planned, implemented, trained and for accuracy score increasing purposes different parameters are set. I really appreciate to be a part of this process and thankful for any contributor who has helped me so far.