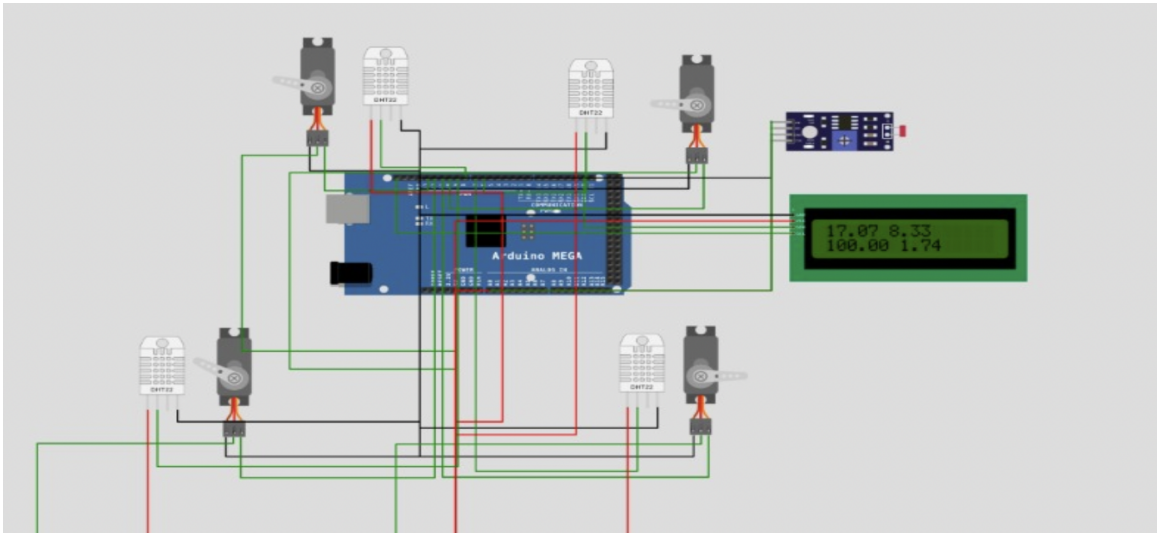
Indian Institute of Technology Kanpur
Introduction to IoT and Its Industrial Applications (CS698T)
Assignment 2

Arjun Singh (21111402, arjuns21@iitk.ac.in)
Gajender Sharma (21111028, gajenders21@iitk.ac.in)

November 2021

1 COMPONENTS

This project is simulated on Arduino Mega board on wokwi platform for processing the instructions. There is a Multi Layer Perceptron (MLP) Model deployed, which has been trained on a dataset of humidity and temperature values before to the simulation. For sensing the surrounding, DHT22 is used. Water flow is controlled by Servo motor. The LDR sensor detects if it's a Day or Night and the LCD 16x2 displays the amount of waterflow needed being supplied to the farm.



2 SENSORS USED

- **LDR - A photoresistor**

  A photoresistor or light-dependent resistor(LDR) is light sensitive device most often used to indicate the presence or absence of light, or to measure the light intensity. In the dark, their resistance is very high, sometimes up to 1mega ohms, but when the LDR sensor is exposed to light, the resistance drops dramatically, even down to a few ohms, depending on the light intensity. LDRs have a sensitivity that varies with the wavelength of the light applied and are nonlinear devices. The sensor has 4 pins namely VCC, GND, DO and AO. It can give both digital and analog reading depending on which pin among DO and AO is used for recording the readings.The concept involved in a photoresistor is that if incident light exceeds a certain frequency, photons absorbed by the semiconductor give bound electrons enough energy to jump into the conduction band. The resulting free electrons conduct electricity, thereby lowering resistance.

- **DHT22**

  The DHT22 is a basic digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and gives out a digital signal on the data pin, no analog input pins needed. It has a VCC pin(5v positive pin), SDA pin(signal line goes to a digital pin), NC(not connected) and GND(ground -ve line).

- **Servo Motor**

  A servomotor is a rotary actuator or linear actuator that allows for precise control of angular or linear position, velocity and acceleration. It consists of a suitable motor coupled to a sensor for position feedback. It also requires a relatively sophisticated controller, often a dedicated module designed specifically for use with servomotors. It has 3 pins – ground pin, v+ pin and PWM pin(signal pin). We can turn its arm by 180, this can be done in a looping manner or just once.

- **LCD display**

  We have used 16x2 dimensions LCD display screen. It is able to display 16x2 characters i.e. 16 characters on 2 lines (including white characters and special characters). VCC, GND, SDA, SCL. SDA is the serial data pin and SCL is the clock pin.

3 Working of Irrigation system

We are supposed to build four independent units of irrigation system all of which will contain a sensors and a Actuator namely Servo motor and DHT22. The control flow of the project is :

- **Inputs:** We read the data of the LDR sensor which is used for telling the luminous intensity falling upon it , we have set the threshold value to 50 Lux , if lux intensity is below Threshold ,it will predict Night and shuts off the water flow and will not perform any operation and if lux intenstiy is greater than 50 lux then it will show Day time and will start the further operation.

- During Day time , Irrigation system starts sensing temperature and humidity values from four sensor units using the DHT22 sensor in the surroundings using the in-built function ***readTemperature() and readHumidity()***

- **Processing of sensed values :**Then this data is fed to the forward pass algorithm, whose weights have been learned on trained Multi Layered Perceptron model()()() which returns the waterflow% that needs to be supplied to the farm's four servo motors. The amount of waterflow is returned in percentage that we are displaying on the 16x2 LCD, then we have converted this percentage into degrees using the following formula:

$$Servo\ motor\ angle = \frac{water\ percentage * 179}{100}$$

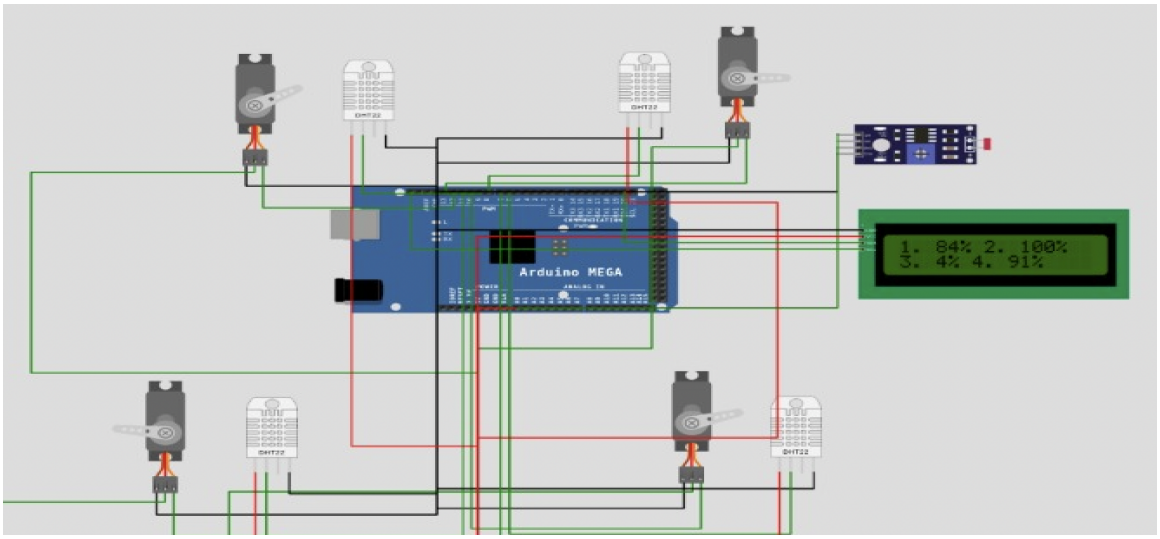
  This gives us the value by which a particular servo motor was supposed to open in order to irrigate the farm which is done using the function ***servo.write(Degrees to turn motor);***

- Each of the four units are independent, so changing the sensor values of Dht22 for a particular servo motor will affect the waterflow rate of that particular servo motor.

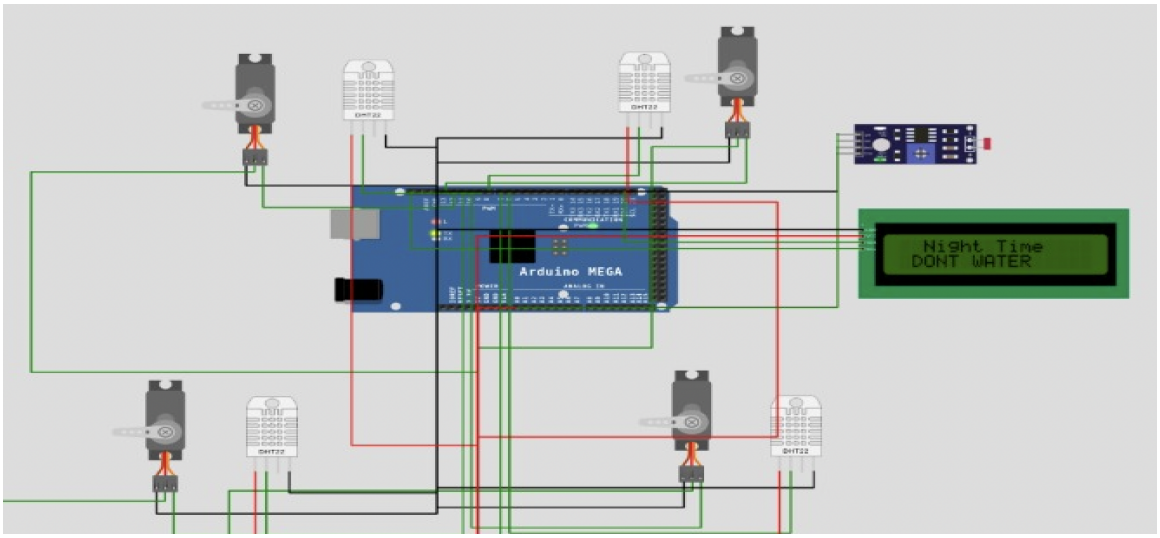
- **Output:** The output of the ML model of waterflow percentage for four servo motors will be dispayed on the LCD screen 16X2, then the angle for the servo motor is calculated using percentage of waterflow returned by ML model.

4 Snapshots of Simulation and Night time

4.1 During Day Time



4.2 During Night Time



5 Machine Learning Model Used

We have Developed a two-layer perceptron machine learning model from scratch (i.e. input layer (with 2 nodes), 2 hidden layers (with 3 and 4 nodes respectively) , and output layer (with a single node)) to predict the servo motor's signal. The activation function that we

have used is ReLu(Rectified Linear Unit), for **1000** epochs and the number of neurons in the first hidden layer as 3 and in the second hidden layer as 4 and output layer with 1 neuron.

5.1 MultiLayer Perceptron Architecture

A multilayer perceptron (MLP) is a deep, artificial neural network. It is composed of more than one perceptron. They are composed of an input layer to receive the signal, an output layer that makes a decision or prediction about the input, and in between those two, an arbitrary number of hidden layers that are the true computational engine of the MLP. MLPs with one hidden layer are capable of approximating any continuous function.

The input layer receives the input signal to be processed. The required task such as prediction and classification is performed by the output layer. An arbitrary number of hidden layers that are placed in between the input and output layer are the true computational engine of the MLP. Similar to a feed forward network in a MLP the data flows in the forward direction from input to output layer. The neurons in the MLP are trained with the back propagation learning algorithm. MLPs are designed to approximate any continuous function and can solve problems which are not linearly separable. The major use cases of MLP are pattern classification, recognition, prediction and approximation.

5.2 Working of MultiLayer Perceptron

- Just as with the perceptron, the inputs are pushed forward through the MLP by taking the dot product of the input with the weights that exist between the input layer and the hidden layer. This dot product yields a value at the hidden layer. We do not push this value forward as we would with a perceptron though.
- MLPs utilize activation functions at each of their calculated layers. There are many activation functions to discuss: rectified linear units (ReLU), sigmoid function, tanh. Push the calculated output at the current layer through any of these activation functions.
- Once the calculated output at the hidden layer has been pushed through the activation function(ReLU), push it to the next layer in the MLP by taking the dot product with the corresponding weights.
- Repeat steps two and three until the output layer is reached.
- At the output layer, the calculations will either be used for a backpropagation algorithm that corresponds to the activation function that was selected for the MLP (in the case of training) or a decision will be made based on the output (in the case of testing).

  Training of the Model :

  - training is done in the 2 phases :

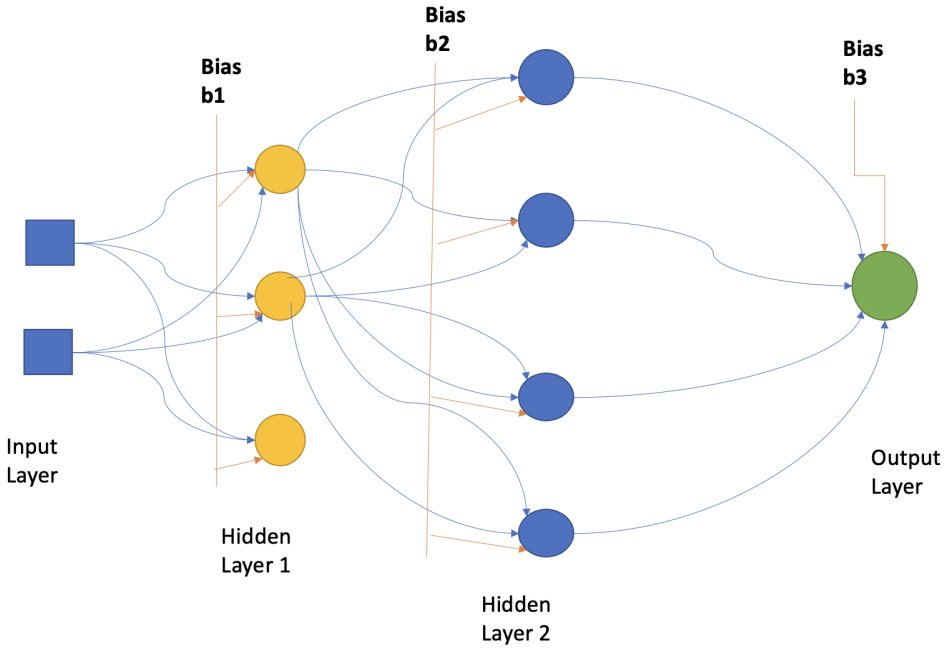


Figure 1: Architecture of Our MLP Model with 2 Hidden Layers

  - ∗ **forward Propagation :** As mentioned above, inputs are pushed to the different layers one by one, and after the activation produced by that layer is pushed to layers above it, until the output layer is reached. At the output layer, Error is calculated.
  - ∗ **Backward Propagation :** Once the error is calculated, we do a backward propagation, in which, error travels backward in the model. By calculating, how error is with respect to the particular weight, we perform gradient descent using learning rate and update the weights.
- This two phases are run until our model converges.

5.3 Parameters Used in ML Model

- **Input Layer Size:** The input layer size used is 2 one is for humidity and other one is for temperature values.
- **Hidden Layer Size:** The total number of neurons used are 3 nodes in first hidden layer and 4 neurons has been used in second layer.
- **Epochs:** The Total number of Iterations Multi layer perceptron run is **1000**.
- **Activation Function: ReLU()**

- **Weights and Bias :**
  - **Between input layer and Hidden layer 1** : Since input layer contains 2 nodes and hidden layer 1 contains 3 nodes , weight matrix was of size 2x3, and bias matrix was of size 3x1.

    Final Weights and bias are as follows :

    W1 :

    [[-1.33812703, -2.07168414, 3.10868101]

    [4.04486389, 0.82929414, -0.29572624]]

    b1 :

    [[0.11331626, 0.11603411, 0.07995422]]
  - **Between Hidden layer 1 and Hidden layer 2** : Since Hidden layer 1 contains 3 nodes and hidden layer 2 contains 4 nodes, weight matrix was of size 3x4, and bias matrix was of size 4x1.

    Final Weights and bias are as follows :

    W2 :

    [[1.86370592, -0.9328917, -2.21483562, 1.50763159]

    [1.2746869, 0.81197727, 0.80514356, 1.31346308]

    [2.0495405, 0.34053685, 2.16277428, 0.93998669]]

    b2 :

    [[0.11353248, 0.09667483, 0.09231333, 0.11184735]]
  - **Between Hidden layer 2 and Output Layer** : Since Hidden layer 2 contains 4 nodes and Output layer contains 1 node, weight matrix was of size 4x1, and bias matrix was of size [1].

    Final Weights and bias are as follows :

    W3 :

    [[2.96574043]

    [-0.13227896]

    [-3.6796874]

    [2.32967535]]

    b3 :

    [0.1072818]

The Rectified Linear Unit is the most commonly used activation function in deep learning models. The function returns 0 if it receives any negative input, but for any positive value x it returns that value back. So it can be written as f(x)=max(0,x).

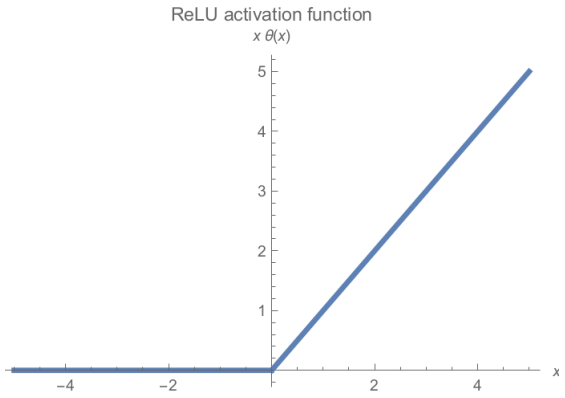
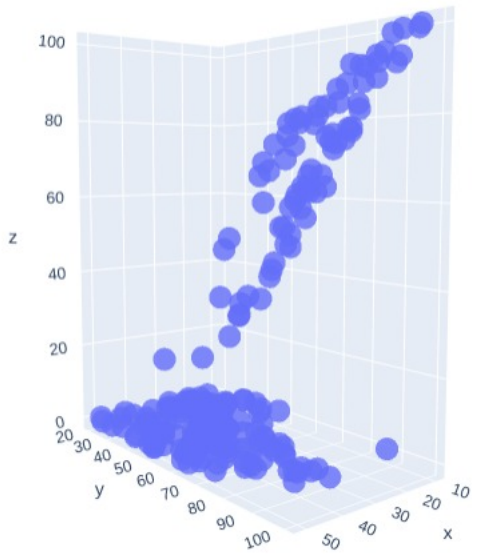


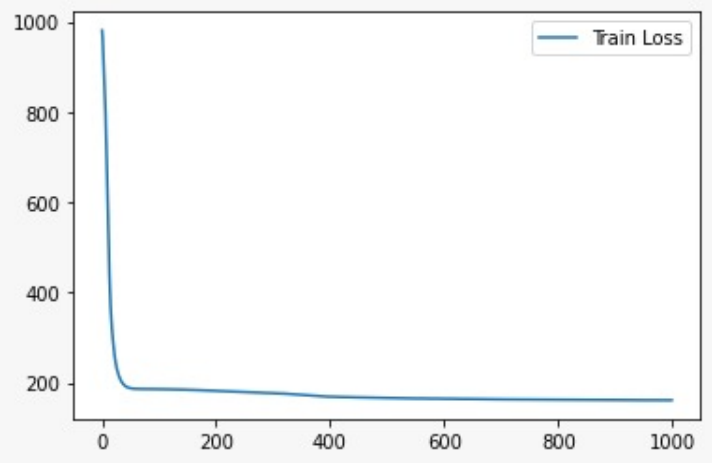
Figure 2: ReLu function

5.4 Performance of Model

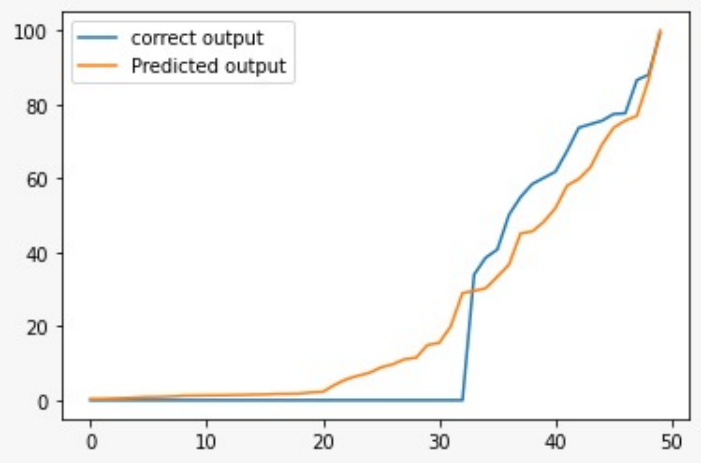
5.4.1 Interactive 3D graph to visualize data



5.4.2 Train Loss



5.4.3 Visualizing how our model performs with Test Data



5.5 Instructions to Run ML Model

- Open and extract the **21111402_21111028.zip** file.
- Find the jupyter notebook with format(.ipynb) in it.
- Run all the notebook cells.

6 Programming

6.1 Libraries Used

- Numpy

- Pandas
- Matplotlib

6.2 Defined functions in ML Model

- We developed our ***ML model from Scratch***, including functions for forward propagation and back propagation.
- **forward:** This Function accepts the value of Humidity and Temperature and performs the forward pass using weights of the Model.
- **backward:** This Function accepts the value of Humidity and Temperature, expected values and learning rate. This function performs the backward propagation to update our weights using Gradient Descent
- **Summary:** This function is used to print the Size of Model Parameters.
- **Return weights:** This Function prints the value of weights and bias, so that they can be incorporated into the Simulator.

6.3 Defined functions in simulation

- **initi() :** It has been used to initialize the different variables , optimal weight obtained from ML model and initializing the bias obtained from ML model.
- **forward_pass() :** It is similar to forward pass in the ML model. Here we are performing dot product on the activations of previous layer to the current layer. the output is then passed through relu function and passed on to next layers until output layer is reached. It takes argument as Temperature and humidity, and calculates the predicted value for all the 4 sensors nodes in a single forward pass only.
- **initactivations() :** It is used to set all the activations as zero. It is defined under loop() method, because after every forward pass, we need to set value of activation matrix as Zero.

6.4 Setup

Setup section in simulator is used to Run one time and used to initialize different input devices. Here we are initializing LCD screen, the DHT22 humidity and temperature sensor and the Servo motors with appropriate connection to Arduino Mega board. Here we have also initialized our model weights and bias values.

6.5 Loop

This is the section where we give instructions which we want to run continuously in loop by the devices. Here we retrieve the values from various sensors and take appropriate actions. All the actions, like printing on LCD screen and telling servo motors the angle to which they have to move, is done under this method.

7 Result

We computed the our performance metric based on two methods , one of them is coefficient of Determination and other is Root mean squared.

- R^2 **Coefficient of Determination** : We have used R^2 score to determine the accuracy of the predictions of the our ML model. The best possible score that can be obtained is 1.0 and usually minimum value is 0.0 and it can be negative as well (when the model is arbitrarily worse). And in case of a constant model which always predicts the same value of y, disregarding the input features, would get a R^2 score of 0.0.

  The R^2 score is defined as,

  $$R^2 = 1 - \frac{RSS(Residual\ sum\ of\ squares)}{TSS(Total\ sum\ of\ squares)}$$

  where RSS is the residual sum of squares and TSS is the total sum of squares given by,

  $$RSS = \sum ((y\_actual - y\_pred)^2)$$

  $$TSS = \sum ((y\_actual - y\_actual.mean())^2)$$

- **Root Mean Squared Error** : The mean_squared_error function computes mean square error, a risk metric corresponding to the expected value of the squared (quadratic) error or loss.

  RMSE is defined as,

  $$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (y\_actual - y\_pred)^2}{N}}$$

Final Performance Values of our ML model is in below Table

Performance Metric	Scores
R^2	0.7542
RMSE	16.176

8 Link To Our Simulation of The Assignment

https://wokwi.com/arduino/projects/313994911532712512

9 References

- https://deepai.org/machine-learning-glossary-and-terms/multilayer-perceptron
- https://docs.wokwi.com/
- https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141
- https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/