# Energy Efficient Home Automation System using Machine Learning

Problem Statement: -

The task is to build a energy efficient system which regulates the use of energy of a place without compromising the comforts of an individual. The task was divided into two subtasks: -

1. Building a system for continuous pattern places like offices, institutes and schools.
2. B. Building a system for non-pattern places like homes.

Abstract:

The project was divided into two modules:

1. Corridors: Automating the lights of a corridor having a series of lights such that the intensity at every point in the corridor is above a minimum threshold.

Offices: Automating the lighting system of an office by pattern recognition i.e. finding a definite pattern in the activities of the person in the office with respect to time. The data hence obtained acts as the training set to predict the future activities. Example. If the timings of the office are from 9am to 6pm on week days, after training the result should be such that on next week days the lights switch on automatically at 9 am.

1. Using NLP: Home automation using NLP. In this the basic module is to get the voice control system in which the user can control all the electrical machineries with verbal commands. Example. If a person says, “Switch on the fan”, then the fan gets switched on by this system.

Simple intuition behind neural networks

Neural networks work in very similar manner.It takes several input, processes it through multiple neurons from multiple hidden layers and returns the result using an output layer. This result estimation process is technically known as “**Forward Propagation**“.

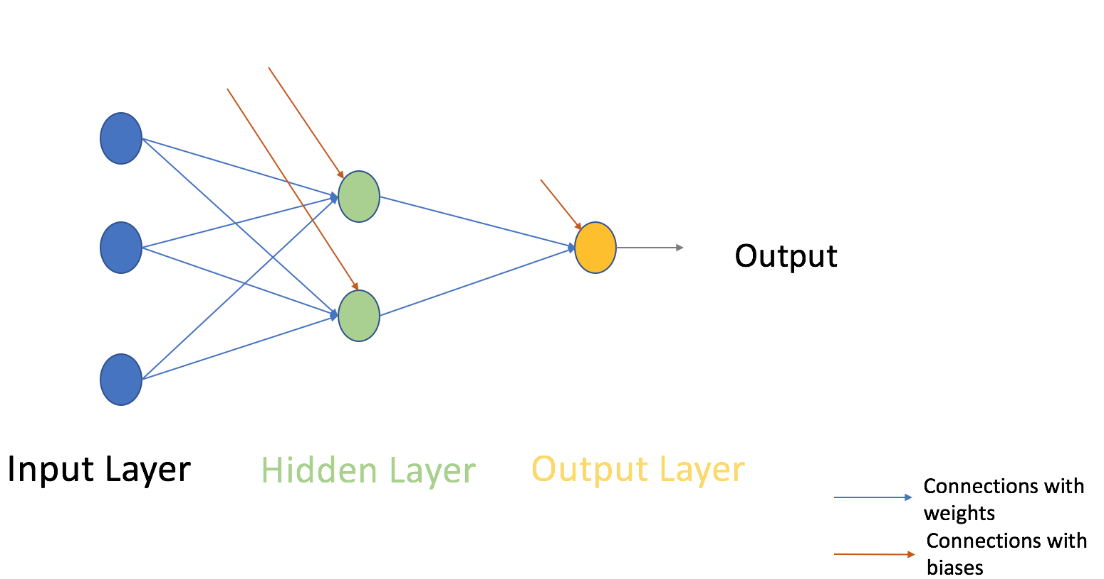
Next, we compare the result with actual output. The task is to make the output to neural network as close to actual (desired) output. Each of these neurons are contributing some error to final output. How do you reduce the error?

We try to minimize the value/ weight of neurons those are contributing more to the error and this happens while traveling back to the neurons of the neural network and finding where the error lies. This process is known as “**Backward Propagation**“.

In order to reduce these number of iterations to minimize the error, the neural networks use a common algorithm known as “Gradient Descent”, which helps to optimize the task quickly and efficiently.

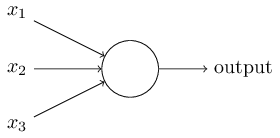
That’s it – this is how Neural network works!

Multi Layer Perceptron and its basics



Just like atoms form the basics of any material on earth – the basic forming unit of a neural network is a perceptron. So, what is a perceptron?

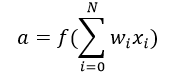
A perceptron can be understood as anything that takes multiple inputs and produces one output. For example, look at the image below.



Perceptron

The above structure takes three inputs and produces one output. The next logical question is what is the relationship between input and output? Let us start with basic ways and build on to find more complex ways.

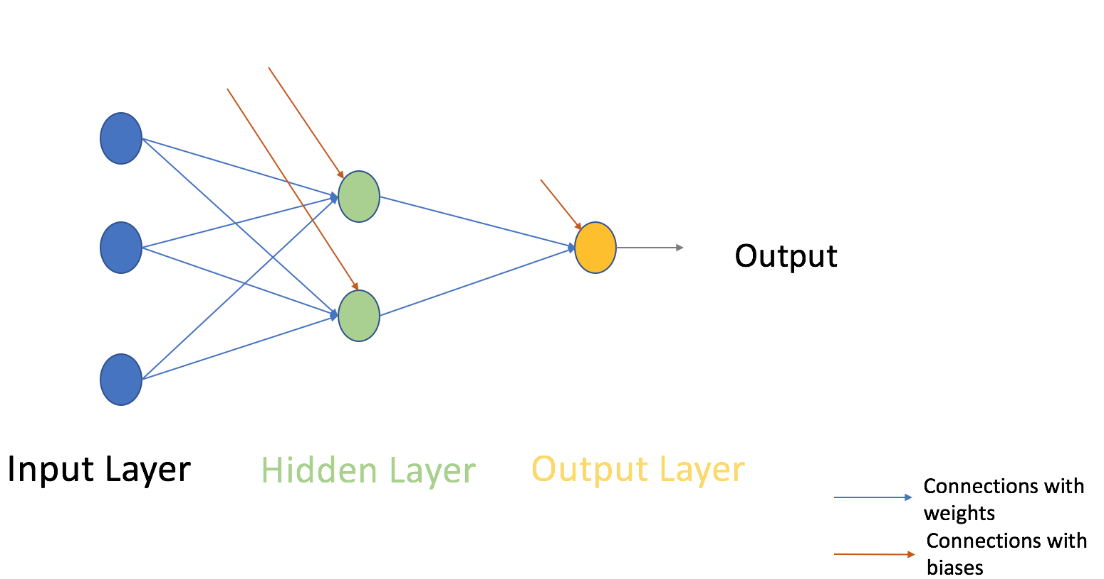
What is an activation function?

Activation Function takes the sum of weighted input (w1\*x1 + w2\*x2 + w3\*x3 + 1\*b) as an argument and return the output of the neuron.In above equation, we have represented 1 as x0 and b as w0.

The activation function is mostly used to make a non-linear transformation which allows us to fit nonlinear hypotheses or to estimate the complex functions. An example of an activation function is the ‘Sigmoid’ function.

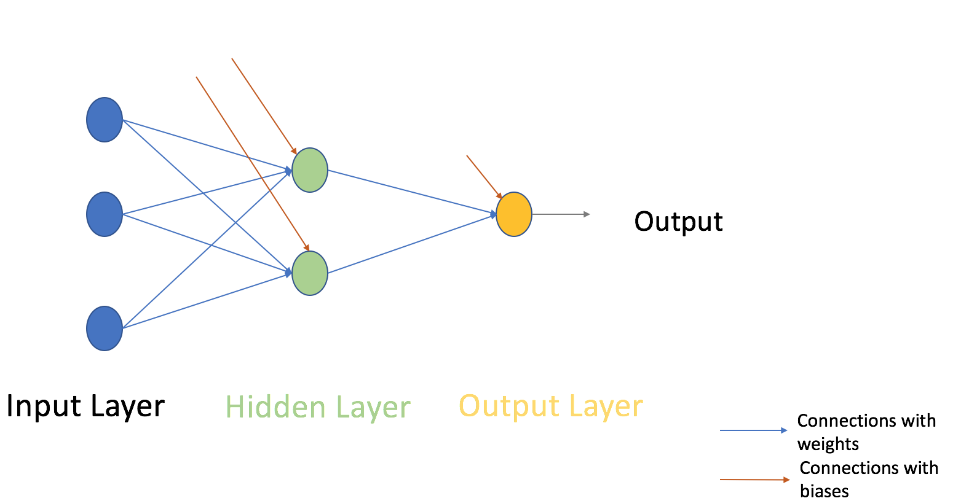
Multi-layer perceptron

Now, let’s move on to next part of **Multi-Layer** Perceptron. So far, we have seen just a single layer consisting of 3 input nodes i.e x1, x2 and x3 and an output layer consisting of a single neuron. But, for practical purposes, the single-layer network can do only so much. An MLP consists of multiple layers called **Hidden Layers** stacked in between the **Input Layer** and the **Output Layer** as shown below.



The image above shows just a single hidden layer in green but in practice can contain multiple hidden layers. Another point to remember in case of an MLP is that all the layers are fully connected i.e every node in a layer(except the input and the output layer) is connected to every node in the previous layer and the following layer.

Steps involved in Neural Network methodology



Let’s look at the step by step building methodology of Neural Network (MLP with one hidden layer, similar to above-shown architecture). At the output layer, we have only one neuron as we are solving a binary classification problem (predict 0 or 1). We could also have two neurons for predicting each of both classes.

First look at the broad steps:

0.) We take input and output

* X as an input matrix
* y as an output matrix

1.) We initialize weights and biases with random values (This is one time initiation. In the next iteration, we will use updated weights, and biases). Let us define:

* wh as weight matrix to the hidden layer
* bh as bias matrix to the hidden layer
* wout as weight matrix to the output layer
* bout as bias matrix to the output layer

2.) We take matrix dot product of input and weights assigned to edges between the input and hidden layer then add biases of the hidden layer neurons to respective inputs, this is known as linear transformation:

3) Perform non-linear transformation using an activation function (Sigmoid). Sigmoid will return the output as 1/(1 + exp(-x)).

4.) Perform a linear transformation on hidden layer activation (take matrix dot product with weights and add a bias of the output layer neuron) then apply an activation function (again used sigmoid, but you can use any other activation function depending upon your task) to predict the output

**All above steps are known as “Forward Propagation “**

5.) Compare prediction with actual output and calculate the gradient of error (Actual – Predicted). Error is the mean square loss = ((Y-t)^2)/2

6.) Compute the slope/ gradient of hidden and output layer neurons ( To compute the slope, we calculate the derivatives of non-linear activations x at each layer for each neuron). Gradient of sigmoid can be returned as x \* (1 – x).

7.) Compute change factor(delta) at output layer, dependent on the gradient of error multiplied by the slope of output layer activation

8.) At this step, the error will propagate back into the network which means error at hidden layer. For this, we will take the dot product of output layer delta with weight parameters of edges between the hidden and output layer (wout.T).

9.) Compute change factor(delta) at hidden layer, multiply the error at hidden layer with slope of hidden layer activation

10.) Update weights at the output and hidden layer: The weights in the network can be updated from the errors calculated for training example(s).

learning\_rate: The amount that weights are updated is controlled by a configuration parameter called the learning rate)

11.) Update biases at the output and hidden layer: The biases in the network can be updated from the aggregated errors at that neuron.

* bias at output\_layer =bias at output\_layer + sum of delta of output\_layer at row-wise \* learning\_rate
* bias at hidden\_layer =bias at hidden\_layer + sum of delta of output\_layer at row-wise \* learning\_rate

**Steps from 5 to 11 are known as “Backward Propagation “**

One forward and backward propagation iteration is considered as one training cycle. When do we train second time then update weights and biases are used for forward propagation.

Above, we have updated the weight and biases for hidden and output layer and we have used full batch gradient descent algorithm.

Support Vector Machines

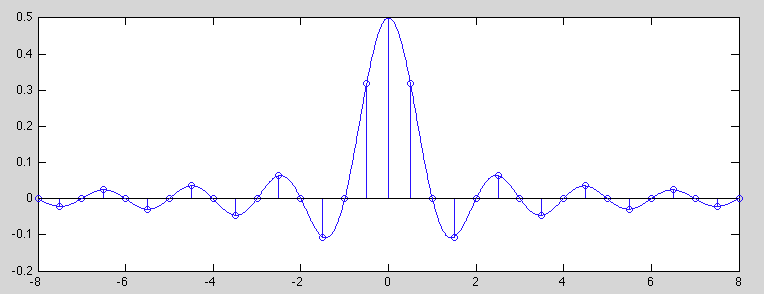
Support Vector Machines(SVM) is an example of a Supervised Learning Algorithm, its mostly used in classification and regression analysis.

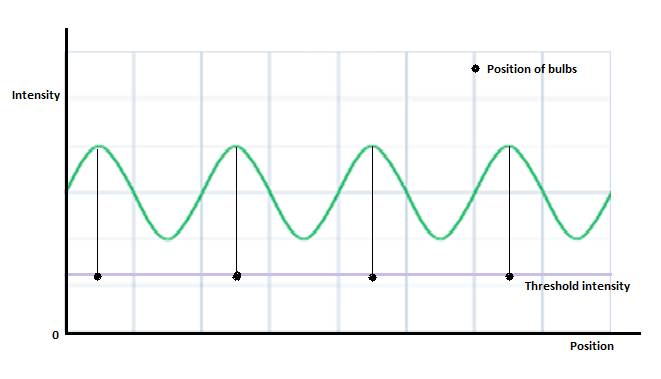
There are several algorithms present under the classification of SVM, names of a few of them are linear, polynomial, RBF which is also called Gaussian, however for this project polynomial regression can be used .

Before explaining polynomial regression, an example of linear regression will be taken. Based on your inputs (X) there is an output(Y). To find the relationship between Y and X, the gradient of the graph is found which is denoted as m and the y-intercept is also found which is denoted as c. The relationship between the inputs(X) and the outputs(Y) is given by the classical relationship Y=mX+c. In linear regression the inputs(x) and the outputs(y) are given to the program and the program tries to find the best slope for which the error is minimum.

Now moving onto polynomial regression similar to linear regression the inputs(X) and the outputs(Y) are given, however the equation is modified to fit higher powers of X, this allows for curves to be modelled into the equation. Y=m1X+m2X^2+m3X^3+mnx^n.

Approach:





The graph between the position and the corresponding intensity is plotted. On the same, the line of the threshold intensity is plotted. The green curve represents the superposition of intensities of individual bulbs. The basic idea is to obtain the intensity approximately equal to the threshold intensity at every point.

Model 1:

Input: The graph for intensity at every point as shown earlier.

Neural Network 1: The intensity at each point is the training set for the same. Its output is the status of each bulb whether on or off.

The same can be done using a circuitry which detects if the bulb is on or off using X rated capacitor which converts 220V to 5V.

Neural Network 2: The output of the 1st neural network is fed as input to 2nd neural network. By adjusting weights of different layers, the output is obtained such that intensity at each point is just above threshold intensity.

Flaw: It needed high amount of computation to train according to dataset. It also had the vulnerability of not being very efficient in the approach.

Model 2:

Input: Intensity array of all sensors at different time, time at different instances and state of window i.e. open or close (0 or 1)

Output: Binary array depicting status of lights (0 or 1) such that intensity at each point is above and nearest to threshold.

Training: Each combination of output on or off states of lights will be set for a day. The intensity at each point will be obtained by sensors. This will be the training set for the four-layer neural network.

Flaw: Training is needed for 365 days because of seasonal change and random environment changes. Training this large data is not feasible hence this model was discarded or needed future changes.

Model 3 (Using SVM):

The independent variable is the location array of the bulbs, and the dependent variable is the light intensity at that point. Using polynomial regression a graph is plotted between the input and output. For the system to be energy efficient the area under the graph(curve) should be as small as possible and at the same time should be above a threshold value. The threshold value is the minimum light intensity to be present so that the visibility of the area is not hindered. All the points are checked against the threshold value and based off that a few calculations , the output of the program gives which lights based on their position should be on or off .

Flaws:

Unlike neural networks, where the weights get updated, in this model the same number of calculations have to be made to decide which lights are going to be on and off

After a few lights turn off , the updated graph might give another set of lights to turn on and off ,visually a few lights may turn on and off within seconds and it would take time before the light output comes to stable point .

Model 4(Final Model):

Input of the network consists of 6 neurons, intensity values obtained by ldr sensors, 2 placed outside the window of the lobby indication atmospheric conditions and 4 placed in between 5 led’s

Output: Pattern of 0’s and 1’s indicating the status of each led

According to the intensity values at points in the lobby and the atmospheric conditions, the network learns the pattern of led.

While training the input will be ldr sensor values of the windows and intensity equal to threshold for the middle 4 sensors. The output will be pattern of led’s such that the intensity is approximately equal to the threshold.

Training dataset:

Data of ldr sensor was collected every 15 mins for each and every combination of led (2^5=32 combinations) using microcontroller Arduino uno. The data was saved as an excel file using PLX-DAQ software

**Harnish Rajput (Work):**

In the starting stage I learned Neural Network and implemented it. Than after, I started

thinking on the lobby model. I proposed a in general model of the lobby.

Model:

Input: Time, sensors values and bit array of whether the particular window/door is open or close.

Output: Which combination of light lead to this intensity (light bit array).

Data: Data will be taken for whole day by applying a particular combination of light. And this will be done for all combinations.

Flaw: It won’t be effective for random atmospheric changes.

In whole December I learned NLP (Natural language processing). From starting of January I was assigned myself the work of making a miniature lobby with implementation of automation (of lights) by neural networks. Then, I helped to make the hardware of lobby. I with my colleagues took the readings upto 15 th January. But this data had error, so we again took the readings at 22 nd January.

After that I tried a lot to reduce the error. The following I tried:

1. I normalised data

2. I changed the size of hidden layer

3. I changed the number of hidden layer from 0 to 3

4. I tried to use the series of Neural Network for reducing error. But that also didn’t work that good. But was better than previous models.

Finally, I get realised that this was the data error. I was training it for all combination of lights. But the fact was the combination of light which was giving the middle intensities (4 sensors) less than 700 and more than 950 was of no use. Rather, this extra data was creating the trouble in training because huge range of data. Thus, I again took the readings and this time I filtered it with the above logic and this worked. I also closed one side while taking readings this time. Thus, biasness in intensities was also functioning the same output. So, here I am with 51% of accuracy with 5 bit comparison and 93% with one bit tolerance.

**Mansi’s Documentation**:

Problems with dataset:

1. In the first attempt, there was a sensor with continuously gave intensity equal to 1023, which indicated the ldr sensor to have reached maximum limit. Hence some reading were taken again.

2. In case of the last two ldr sensors, on changing the reversing of the lobby, the intensity values did not get reversed.

3. The intensity obtained in the middle 4 ldr sensors was in the range 850-1000, while the threshold should be almost 750. Such readings were not available. Hence the network learnt something which was not fulfilling the goal. Hence useful readings were extracted from the set of available readings and the network was then trained with this dataset.

Problems in training and optimisation methods used:

1. For training the parameters like learning rate, number of epochs and number of hidden layer neurons were varied from 0.3 to 0.0003, 800 to 100000 and 2 to 20 respectively. The code was not generalised hence number of hidden layers could not be changed.

2. Computation of efficiency: We may consider the test case containing 5 outputs as correct only if all 5 inputs are correct or compute the efficiency bit wise. Using method 1, bit wise efficiency was increased from 40% to 66%. Yet the efficiency of all 5 bits remained zero.

3. While training the sigmoid function gave overflow error. Since the intensity values were in the range of 900 to 1000, exp(-x) reached an infinite value and hence could not be computed. Hence the input values were normalised by a factor of 1000.

4. Method of training the network: I trained the model both batch wise and single input wise. Taking single input, the output in the test case remained same as the last data trained. Taking single input, the maximum efficiency gained was 60%. I am yet to try using mini batch training.

5. Change in learning rate: Even after reducing the learning rate to 0.0003, there was overshooting in the network. Further the learning rate was kept variable i.e. after every 1000 iterations the learning rate was halved.

6. Initialising weights: Instead of randomly initialising the weights and biases every time we train, the weights obtained after some iterations were saved to a new csv file and for next training the same weights were used.

7. After trying all above techniques, the maximum efficiency of the network for initial data (1214 set) was obtained as 66%.

8. For the new dataset obtained, data of 400 was kept for training and remaining 83 was kept as testing. Upon using the above techniques for the new dataset, a maximum efficiency of 88%was obtained.

**Vasu’s Documentation**

What is Natural Language Processing?

Natural Language Processing is a field of computer science specifically a sub branch of Artificial Intelligence. Through its programming a computer is tasked to understand a language, its words, the grammar, its implicature as there are several words present in any language which

have different meanings when used in a sentence for example, “I wore an orange shirt.” Is different from “Today I had orange juice.”

To start it off before Natural Language Processing can be done, tokenization has to be done on the string which is being passed . Tokenization is where the string is broken up into its individual components(words), similar to way Humans look at a sentence as a group of words arranged in a specific sequence. To apply tokenisation there are functions already available in python and in java . In python it can be imported from the nltk module, and there is a ready

made function called spilt () and for tokenization to take place the delimiter (split the words based on) has to bet set to ‘ ‘ (whitespaces).

After tokenization has been done, the tokenized sentence can be then compared then against a dictionary to figure out what the user has sent. There are several ways to go about this, one is using a decision tree, the other is by using a parser which then identifies the verb , noun , adjective using some previous training. For this project since there is a limited vocabulary there was no need for a parser . A parser would be helpful when dealing with a wide range of vocabulary( where the user input cannot be predicted ) . Since there is a limited vocabulary the list of available nouns,verbs and compound nouns were defined explicitly such as the only two verbs needed were ,”On” and “Off”, the nouns which were required were ,”Light/s” and similarly “Fan/s”.

To figure out what verbs, nouns and compound nouns are present . The intersection between the explicitly defined lists and the tokenized are checked and based on these intersections the command is given whether to turn on or off the lights and fan . The length of these intersections are then found as it is computationally ‘faster’ to compare numbers than to compare strings, therefore the length of these intersections are then found . To indexes are checked in cases where the words “On” and “Off” are there and the words “Light” and “Fan” are there. The index of these words are taken and the absolute distance between the words “Light” and “On” and “Off” are taken . If the distance between “Light” and “On” is greater than the distance between the words ,”Light” and “Off” then the command which should be given to the lights are to turn them off .

Problems faced:

In the beginning when strings were compared it used to jump through the conditions and give a wrong command . For example when the command, “ Turn off both the light and fan” it used to identify the nouns , and verbs properly but it used to give the command ,”Turn off the fan”. To solve this the lengths were calculated and the comparison was done in between numbers .

**Works(Sourabh Varshney):**

At the beginning, we were trying to learn as much as I can in the field of machine learning. As it is still an unexplored one, it took time for me to recognize and learn all these algorithms. I learned them and used them for some basic projects like Xor, Xnor and few more.

After learning neural networks, I wrote the code in python(it took time as I was not an expert). I ran it on some use cases.

Now at this point, task was assigned to teams. I was in the Natural language processing Team. I took the part where it was required to process the data and interface it with the app.

In the data processing, I made the CFG for the task and then parsed the text. Now, from the parsed sentence, I identified the main terms and according to that , I changed the state of variables(Idea which was left for completion later). But that was done in python, but app is written in Java. So, we need to make an interface between the two. I tried to use jython library but there was a bug in jython because of which I could not use it. I tried kivy but in kivy, we need to have every code in python. So, I dropped the idea. Then, I converted the python code to Java and corresponding output was stored in some variables, which was further transferred to micro controller through bluetooth and H-305 and hence the further process was carried out.