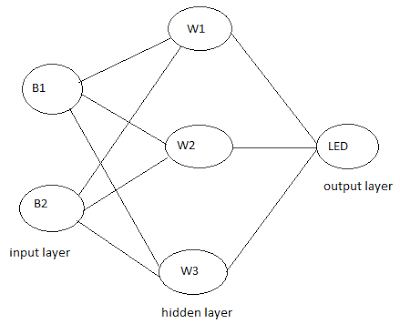
# **[Demo 32: Simple Machine Learning - Artificial neural network demo using Arduino ESP32](http://www.iotsharing.com/2017/09/simple-machine-learning-neural-network-demo-using-arduino-esp32.html)**

**1.Introduction**  
In this demo, I will make a simple Machine Learning - Artificial neural network system using Arduino ESP32. Certainly, I will not use ESP32 for training process; instead, i will use Python + numpy for training process. After training, I will use the result weights will be used by ESP32 for output calculation based on input.  
[Artificial neural network](https://en.wikipedia.org/wiki/Artificial_neural_network) (ANN) systems are inspired by the biological neural networks. These systems can learn to do tasks by considering examples, without task-specific programming.One of most famous methods for ANN training is Backpropagation. You can refer [here](https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/).  
In this demo: ESP32 with 2 buttons (B1 and B2) with possible states: pressed (0) or released (1) and there is 1 LED output with possible state state on (1) or off (0). The LED output can be set based on the state of input.

|  |  |  |
| --- | --- | --- |
| **B1** | **B2** | **LED** |
| pressed | pressed | off |
| pressed | released | on |
| released | pressed | on |
| released | released | off |

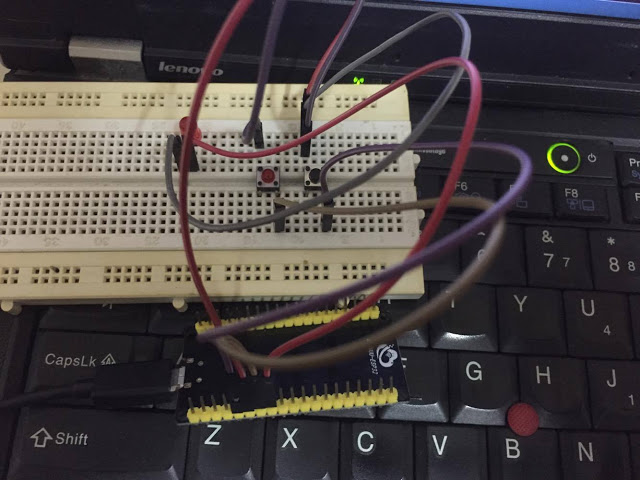
**Figure: state of LED based on state of buttons**

We will create an ANN to learn the table above. The network has structure:

[](https://1.bp.blogspot.com/-h_RE7QJTDNw/WbdMtClwV_I/AAAAAAAAEQ4/wRu37JsUDnsQPvUeLnf9u-nsiG_1SskbACLcBGAs/s1600/esp32-neuralnet.png)

**Figure: ANN structure of demo**

**2. Hardware**  
Here we set input pins as INPUT\_PULLUP so the schematic is simple and we can re-use the schematic in [Demo 21](http://www.iotsharing.com/2017/06/how-to-use-interrupt-in-arduino-esp32.html).

[](https://2.bp.blogspot.com/-RKqNsdDngxg/WbfSuTU19oI/AAAAAAAAERY/S7VE_rsCaakK71bjD8oRmsxnD8NxjlB0ACLcBGAs/s1600/esp32-neuralnetwork.jpg)

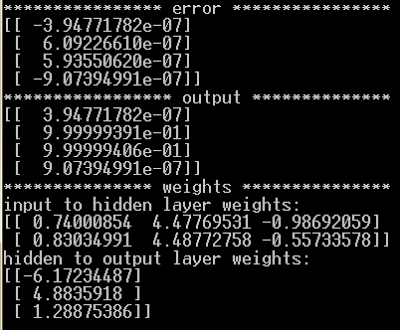
**Figure: hardware connections**

[ESP32 GIO12 - BUTTON 1 - GND]

[ESP32 GIO13 - BUTTON 2 - GND]

[ESP32 GIO14 - LED - GND] **3. Software**  
**3.1 ANN is implemented by Python + numpy**  
I used the code [here](http://python3.codes/neural-network-python-part-1-sigmoid-function-gradient-descent-backpropagation/).

|  |
| --- |
| import numpy as np    epochs = 10000 # Number of iterations  inputLayerSize, hiddenLayerSize, outputLayerSize = 2, 3, 1  L = 0.1 # learning rate    X = np.array([[0,0], [0,1], [1,0], [1,1]]) # Buttons states array  Y = np.array([ [0], [1], [1], [0]]) # LED states array    def sigmoid (x): return 1/(1 + np.exp(-x)) # activation function  # weights on layer inputs  Wh = np.random.uniform(size=(inputLayerSize, hiddenLayerSize))  Wz = np.random.uniform(size=(hiddenLayerSize,outputLayerSize))    for i in range(epochs):    H = sigmoid(np.dot(X, Wh)) # calculate forward part  Z = np.dot(H,Wz) #  E = Y - Z # calculate error  dZ = E \* L # delta Z  Wz += H.T.dot(dZ) # calculate backpropagation part  dH = dZ.dot(Wz.T) \* sigmoid\_deriv(H) #  Wh += X.T.dot(dH) # update hidden layer weights  print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* error \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print(E)  print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* output \*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print(Z)  print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* weights \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("input to hidden layer weights: ")  print(Wh)  print("hidden to output layer weights: ")  print(Wz) |

[](https://1.bp.blogspot.com/-WT8Xc9ZftaY/WbdOtuw3UxI/AAAAAAAAERI/0t_Pc4ZdRCwdfN_d7qJmkwIcBZd8AD7OgCLcBGAs/s1600/esp32-neuralnet-2.png)

**Figure: training process and weights output**

**3.2 Arduino ESP32**  
The Arduino code just do the forward part to calculate the output when getting the inputs.

|  |
| --- |
| #include <math.h>  #define B1 12  #define B2 13  #define LED 14  int X[1][2] = {{1,0}};  /\*these matrices was calculated by python \*/  float W1[2][3] = {{0.74000854, 4.47769531, -0.98692059},  {0.83034991, 4.48772758, -0.55733578}};  float W2[3][1] = {{-6.17234487},  {4.8835918},  {1.28875386}};  float Wo1[1][3];  float sum = 0;  float Y = 0;  /\*sigmoid function\*/  float sigmoid (float x)  {  return 1/(1 + exp(-x));  }  void setup()  {  Serial.begin(115200);  pinMode(B1, INPUT\_PULLUP);  pinMode(B2, INPUT\_PULLUP);  pinMode(LED, OUTPUT);  digitalWrite(LED, LOW);  }  void loop()  {  X[0][0] = digitalRead(B1);  X[0][1] = digitalRead(B2);  printf("B1 = %d, B2 = %d\n", X[0][0], X[0][1]);    /\* calculate forward part based on weights \*/  //hidden layer  for(int i=0; i<1; i++)  {  for(int j=0;j <3; j++)  {  for(int k=0; k<2; k++)  {  sum += X[i][k]\*W1[k][j];  }  Wo1[i][j] = sigmoid(sum);  sum = 0;  }  }  //output layer  for(int i=0; i<1; i++)  {  for(int j=0;j <1; j++)  {  for(int k=0; k<3; k++)  {  Y += Wo1[i][k]\*W2[k][j];  }  }  }  printf("Y = %f\n", (Y));  Y = round(Y);  digitalWrite(LED, int(Y));  Y = 0;  delay(1000);  } |