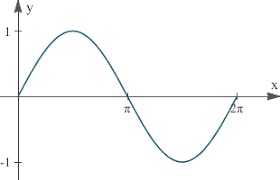
We are going to train a neural network using TensorFflow and implement it in a microcontroller. Our neural network is going to predict the sin(x). Using the same procedure we can predict differents outputs with the right data.

An Artificial Neural Network is defined in the wikipedia as "computing systems vaguely inspired by the [biological neural networks](https://en.wikipedia.org/wiki/Biological_neural_network) that constitute animal [brains](https://en.wikipedia.org/wiki/Brain). Such systems "learn" to perform tasks by considering examples, generally without being programmed with task-specific rules.

So the first part is that we have to teach the NN how a sin(x) function is.



We use TensorFlow in colab, here is the link <https://colab.research.google.com/drive/1ABDULCjzvNZJ6TwHpTvAJnKeyM-_kfPR>

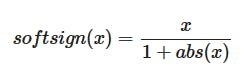
We need data for training so we get x, y pairs for training:



And x, y pairs for test our NN:



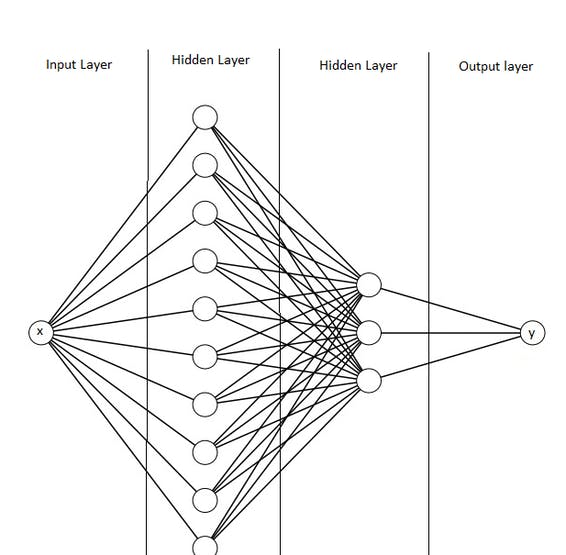
The basic unit of computation in a neural network is the **neuron** or node It receives input from some other nodes, or from an external source and computes an output. Each input has an associated weight (a), which is assigned on the basis of its relative importance to other inputs plus constant, called bias (b). The node applies a non linear function to the weighted sum of its inputs called activation function. In our case we use the softsign function as activation function.



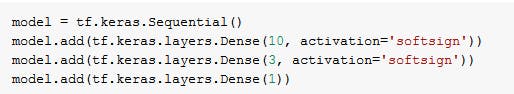
We now define a simple Neural Network, with nodes arranged in layers. Nodes from adjacent layers have **connections** or **edges** between them. All these connections have **weights** associated with them. We use four layers, one input layer, two hidden layers and an output layer.

* **Input Nodes –** The Input nodes provide information from the outside world to the network and are together referred to as the “Input Layer”. No computation is performed in any of the Input nodes.
* **Hidden Nodes –**The Hidden nodes have no direct connection with the outside world (hence the name “hidden”). They perform computations and transfer information from the input nodes to the output nodes. A collection of hidden nodes forms a “Hidden Layer”. It can have zero or multiple Hidden Layers.
* **Output Nodes –**The Output nodes are collectively referred to as the “Output Layer” and are responsible for computations and transferring information from the network to the outside world.

We have a node in the input layer, 10 nodes in the first hidden layer, 3 nodes in the second hidden layer and one node in the output layer. This is the graph:

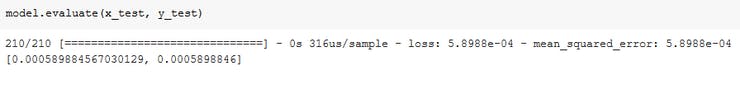


In TensorFlow is defined as:

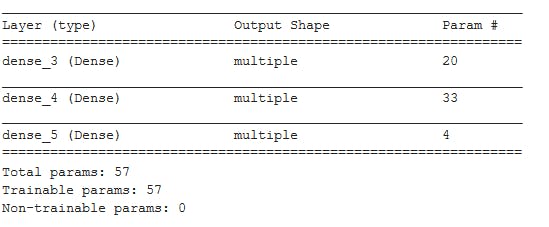


The we train the model and record a video of the training, you can see the video here

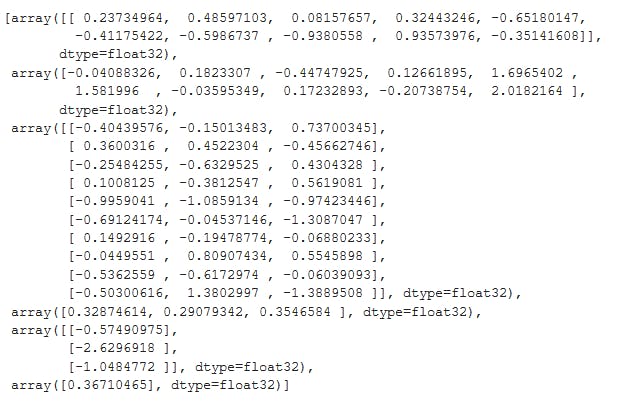
We evaluate the model, and the error is very low.



In order to implement the model in the microcontroller we need the architecture, that we define before of three layers and the weights os each node.



We have three layers as we expect and these are the weights:

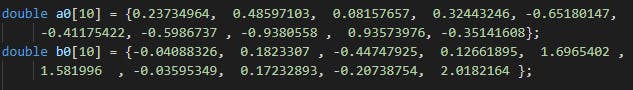


The input layer has only one input the x we want to calculate y = sin(x).

The first hidden layer has 10 nodes, each node performs the operation:

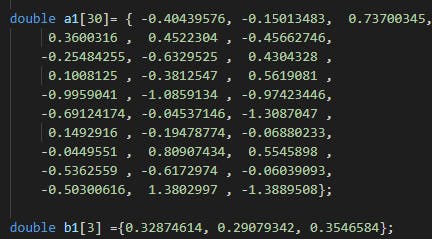


So the first array has 10 elements, an "a" for each node, and the second arrary of 10 elements are the "b".



The second hidden layer has 3 nodes with 10 inputs, so we have an array 3x10 for the "a" one for each node, and array of 3 for the "b".





The output layer is a node with 3 inputs so we have 3 "a" and one "b".

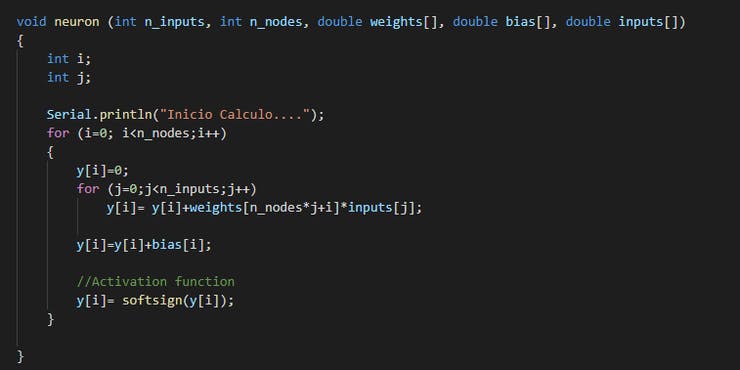




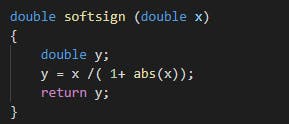
I use platformio + vs code. For MT3620 you need to install the platform support from here <https://github.com/Wiz-IO/platform-azure>

Now we can implement it in the microcontroller. I have implemented it in the M7 core of the MT3620 board, and in an ESP32.

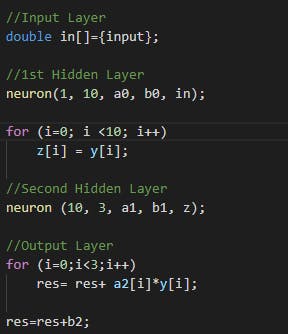
We only need two functions one for the layer, we pass number of inputs for each node, number of nodes in the layer, weights, bias and the inputs:



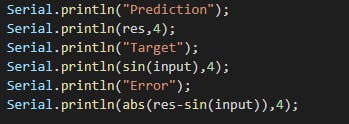
And one for the activation function, the softsign function:



We ask for an input using the serial port, then we pass it to the first hidden layer, the result is passed to the second hidden layer and the output to the output layer:



Finally we compare the prediction with the output of the sin function.



The same code could be used in an ESP32 and with the same results.