STMicroelectronics SensorTile Tutorial: Introduction to Machine Learning for IoT

Table of Contents

1. Introduction to This Tutorial

Machine Learning will be a critical resource for every engineering discipline. Machine Learning combined with IoT offers the potential for remarkable advances in systems that learn and adapt to changing environments, arrival of unpredictable events, and the evolving needs for support of users of infrastructure, vehicles, medical instruments, and a vast diversity of other systems.

Internet of Things (IoT) provides access to sensing information from virtually any environment, vehicle, or individual. IoT devices also must learn and adapt with Machine Learning. However, many IoT devices must operate independently of other computing resources since they may be isolated, or constrained by size and cost.

EmbeddedML is a breakthrough developed entirely and independently by Charles Zaloom, a UCLA student and graduate of the Freshman IoT Curriculum. EmbeddedML provides a Machine Learning solution hosted and operating entirely on the STM32 processor platform and therefore entirely on the SensorTile.

Charles' technology is open source and supplied here:
https://github.com/merrick7/EmbeddedML-for-STM32

This Tutorial introduces EmbeddedML with an application for the solution of a problem with its Neural Network.

The following Tutorials will provide additional examples integrating SensorTile training and execution integrated with its motion sensors.

For more information regarding the SensorTile board, please open the following link.
www.st.com/sensortile

List of Required Equipment and Materials

1) 1x STMicroelectronics SensorTile kit.
2) 1x STMicroelectronics Nucleo Board.
3) 1x Personal Computer with two USB type-A inputs OR you must have a powered USB hub.
4) 1x USB 2.0 A-Male to Micro-B Cable (micro USB cable).
5) 1x USB 2.0 A-Male to Mini-B Cable (mini USB cable).
6) Network access to the Internet.

Prerequisite Tutorials

For this introduction to EmbeddedML, it is important that users have completed Tutorial one successfully.

It is recommended for the following Tutorials on EmbeddedML that users have completed Tutorials 1 through Tutorial 4.

Your instructor will ensure you have the required background regarding motion sensing, digital signal processing, and basic physics of acceleration, velocity, and displacement.

2. EmbeddedML: Machine Learning and ioT

- EmbeddedML is the latest generation of Machine Learning.
- A brief history of development of Machine Learning and Neural Networks includes four generations
- Generation 1:
    - Machine Learning appeared in research with investigators performing development of Machine Learning software on their local computing platforms.
- Generation 2:
    - Machine Learning applications expanded rapidly serving application from Internet search to applications in classifying medical images for disease detection.
    - The Machine Learning computation was performed on centralized server computing and datacenters. This is often referred to as the “cloud”
- Generation 3:
    - Machine Learning now includes the “edge” of the Internet network where individual IoT sensors and other devices provide data to the cloud where Machine Learning training and execution occur.
- Generation 4
    - As the number of IoT devices expands at an exponential growth rate, the available communication bandwidth and datacenter resources will be limited.
    - Independent, standalone, IoT devices must learn and act independently.
    - EmbeddedML is a step forward where the individual IoT device performs autonomous training (learning) and execution without reliance on any other resource.
    - EmbeddedML faced the development challenge of implementing Neural Network capability on a constrained, compact platform with limited resources. This is now solved and ready.

3. EmbeddedML XOR-AND Demonstration

This Tutorial provides an introduction to EmbeddedML that prepares the user for advanced development than can follow.

EmbeddedML will be demonstrated by developing a Neural Network that replicates the operation of a logic gate circuit including an Exclusive Or (XOR) and an AND gate as shown in Figure 1.

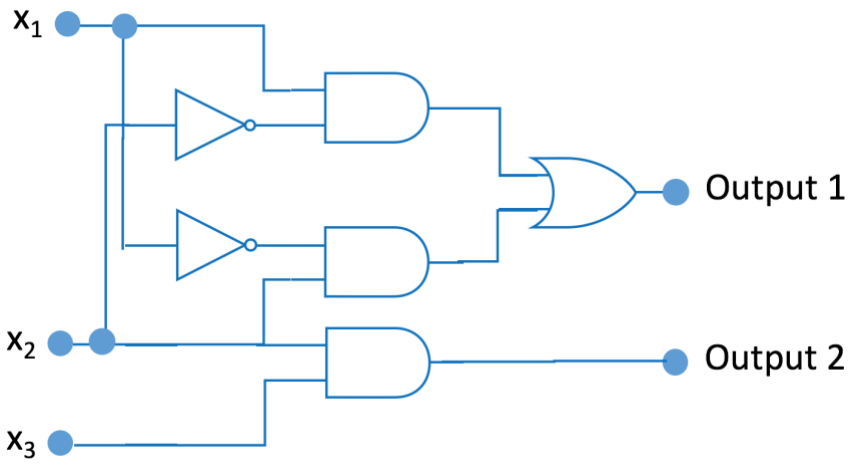


Figure 1. The Three Input - Two Output XOR-AND system.

As described in previous lectures, the relationship between input values and output Ground Truth values is most important.

X_1	X_2	X_3	Output 1 (X_1 XOR X_2)	Output 2 (X_2 AND X_3)
0	0	0	0	0
0	1	1	1	1
1	0	1	1	0
1	1	1	0	1
0	0	1	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	0

Table 1. The Truth Table showing Input values and Ground Truth Output values for the XOR-AND system.

4. The Neural Network Topology

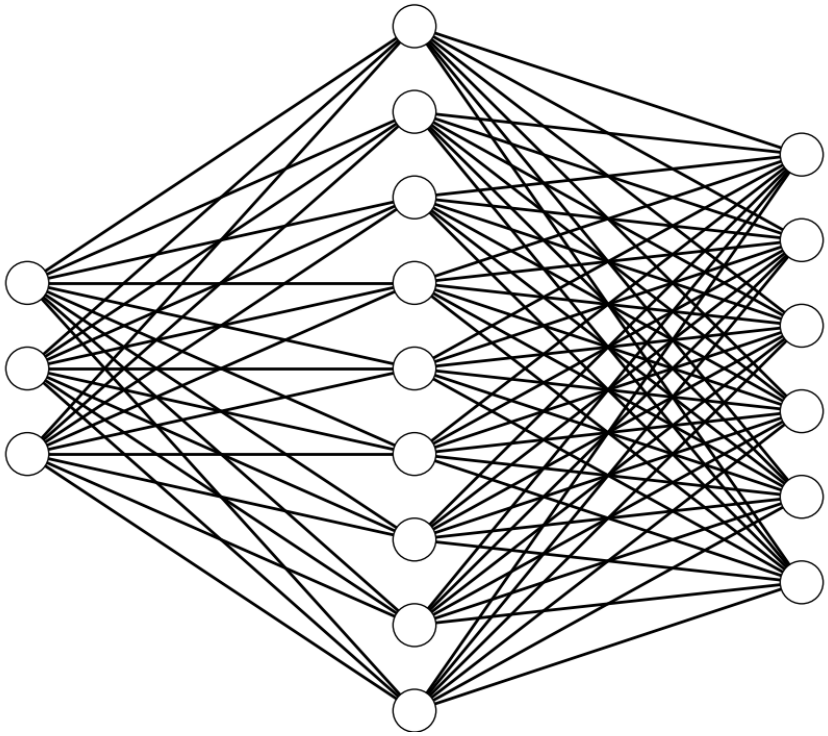


Figure 2. The Neural Network topology with 3 Inputs, one Hidden Layer with 9 neurons, and 6 neurons in the Output layer.

5. The Neuron Activation Function

The EmbeddedML applications for XOR-AND and others will apply the Rectified Linear Unit (ReLU) activation function. This is defined for Weights, w_j, neuron inputs, x_j, neuron bias, b, and neuron output $f(y)$. The response of the ReLU activation function is shown in Figure 3.

$$f(y) = \max(0, y)$$

and where

$$y = \left(\sum_{j=1}^{N} w_j x_j \right) - b$$

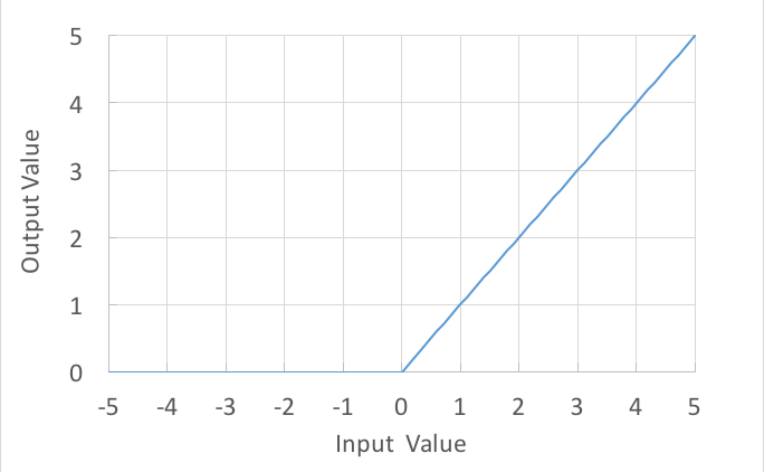


Figure 3. The Rectified Linear Unit ReLU activation function.

The ReLU activation function is computationally simple and thus improves throughput. Finally, its output value of zero for negative input removes the contribution from many neurons for negative input value, further reducing computation requirements.

6. Creating and Initializing a Neural Network

The definition of the Neural Network required data structures is provided by the Designer system.

For all of the projects ahead, the same Neural Network topology of 3 input neurons, 9 hidden layer neurons, and 6 output neurons will be implemented as shown in Figure 2.

The EmbeddedML source code defining the Neural Network follows below.

Definitions for data structures and data structure elements include:

1) weights[81]
    a. The array of neuron weights for each of the synapse connections between the 3 input neurons and 9 hidden layer neurons (a total of 27) and 9 hidden layer neurons and 6 output layer neurons (a total of 54). This results in 81 weights.
2) dedw[81]
    a. The array of values of the derivative of output error with respect to weight.
3) bias[15];
    a. The bias values for each of the 15 neurons
4) network_topology[3] = { 3, 9, 6 };
    a. The network topology definition.
5) output[6]
    a. The output value of output layer neurons
6) x[3]
    a. The input value of input layer neurons
7) y[6]
    a. Ground Truth output values
8) net.eta
    a. Neural Network gradient descent learning rate for weight value training update
9) net.beta
    a. Neural Network gradient descent learning rate for bias value training update
10) net.alpha
    a. The Momentum Coefficient determines the scaling of learning rate with each Epoch. Momentum is a process that reduces (slows) learning rate as the number of Epochs increasing. This benefits accuracy of learning.

11) net.output_activation_function = &relu;
    a. Selection of the ReLU activation function at output neurons
12) net.hidden_activation_function = &relu;
    a. Selection of the ReLU activation function at input neurons

The EmbeddedML source code defining the Neural Network follows below.

```
//---EMBEDDED ANN---

        float weights_initial[81] = { 0.680700, 0.324900, 0.607300, 0.365800, 0.693000,
                                      0.527200, 0.754400, 0.287800, 0.592300, 0.570900, 0.644000,
                                      0.416500, 0.249200, 0.704200, 0.598700, 0.250300, 0.632700,
                                      0.372900, 0.684000, 0.661200, 0.230300, 0.516900, 0.770900,
                                      0.315700, 0.756000, 0.293300, 0.509900, 0.627800, 0.781600,
                                      0.733500, 0.509700, 0.382600, 0.551200, 0.326700, 0.781000,
                                      0.563300, 0.297900, 0.714900, 0.257900, 0.682100, 0.596700,
                                      0.467200, 0.339300, 0.533600, 0.548500, 0.374500, 0.722800,
                                      0.209100, 0.619400, 0.635700, 0.300100, 0.715300, 0.670800,
                                      0.794400, 0.766800, 0.349000, 0.412400, 0.619600, 0.353000,
                                      0.690300, 0.772200, 0.666600, 0.254900, 0.402400, 0.780100,
                                      0.285300, 0.697700, 0.540800, 0.222800, 0.693300, 0.229800,
                                      0.698100, 0.463500, 0.201300, 0.786500, 0.581400, 0.706300,
                                      0.653600, 0.542500, 0.766900, 0.411500 };

        float dedw[81];
        float bias[15];
        unsigned int network_topology[3] = { 3, 9, 6 };
        float output[6];
        float x[3];
        float y[6];

        for (i = 0; i < 15; i++){
                bias[i] = 0.5;
        }
        for (i = 0; i < 6; i++){
                output[i] = 0.0;
        }
        for (i = 0; i < 81; i++){
                weights[i] = weights_initial[i];
                dedw[i] = 0.0;
        }

        ANN net;
        net.weights = weights;
        net.dedw = dedw;
        net.bias = bias;
        net.topology = network_topology;
        net.n_layers = 3;
        net.n_weights = 81;
        net.n_bias = 15;
        net.output = output;

        //OPTIONS
        net.eta = 0.25;    //Learning Rate
        net.beta = 0.01;   //Bias Learning Rate
        net.alpha = 0.25;  //Momentum Coefficient
        net.output_activation_function = &relu;
        net.hidden_activation_function = &relu;;
```

```
/*
* Initialize neural network
*/

init_ann(&net);

//---------------------
```

7. Creating Input Values

- Neural network input values are defined next
- These input values are each associated with Ground Truth output values as in Table 1.

```
/*
* XOR-AND Input Values corresponding to output Ground Truth values
* The neural network includes 3 input neurons
*
*/

float x0[3] = { 0.0, 0.0, 0.0 };  // Corresponds to output 0 0
float x1[3] = { 0.0, 1.0, 0.0 };  // Corresponds to output 1 0
float x2[3] = { 0.0, 1.0, 1.0 };  // Corresponds to output 1 1
float x3[3] = { 1.0, 1.0, 1.0 };  // Corresponds to output 0 1
float x4[3] = { 0.0, 0.0, 1.0 };  // Corresponds to output 0 0
float x5[3] = { 1.0, 0.0, 0.0 };  // Corresponds to output 1 0
float x6[3] = { 1.0, 0.0, 1.0 };  // Corresponds to output 1 0
float x7[3] = { 1.0, 1.0, 0.0 };  // Corresponds to output 0 0
```

8. Train and Test the Neural Network

- Training of the Neural Network begins with the generation of Ground Truth data.
- This is completed with the generate_xorand(x,y,i) function
    - This function accepts arguments corresponding to pointers to the x and y arrays (input and Ground Truth output values).
    - This returns the values in this array.
    - Each time this function is called, a random index between 0 and 7 is selected and the input and Ground Truth values are returned.

- The Neural Network is then trained with execution of train_ann(&net,x,y);
  - This accepts arguments including the address of the net data structure defining the Neural Network as well as input and Ground Truth arrays,
- After training of the Neural Network, execution may occur with run_ann(&net, x0);
  - This accepts arguments including the address of the net data structure defining the Neural Network as well as input values.
  - The returned output value may then be examined and compared against Ground Truth.
- Printing of output values via the USB serial port is provided by
  - Output_Error(int size, ANN *net, float * ground_truth, float *error)
  - This accepts arguments including the number of output neurons (6), the address of the net data structure defining the Neural Network as well as Ground Truth values.
  - This returns the Error value and also prints output values.

```
/*
* Initiate train and test cycles
*/

for (i = 0; i < N_EPOCH; i++) {

        /*
        * Compute input and output ground truth
        */

        generate_xorand(x, y, i);

        /*
        * Train network on input and ground truth
        */

        train_ann(&net, x, y);

        /*
        * After completion of a number of epochs, N_REPORT
        * perform neural network test execution, display
        * resulting outputs and output error
        */

        if (i % N_REPORT == 0 || i == 0) {
                sprintf(epochOut, "\n\rEpoch: %i", i);
                CDC_Fill_Buffer((uint8_t *) epochOut,
      strlen(epochOut));

                net_error = 0;

                /*
                * Execute trained network and compute Output Error
                * with Ground Truth supplied in ground_truth array
                */

                run_ann(&net, x0);
                ground_truth[0] = 0.0;
                ground_truth[1] = 0.0;
                Output_Error(2, &net, ground_truth, &error);
                net_error = net_error + error;

                run_ann(&net, x1);
                ground_truth[0] = 1.0;
                ground_truth[1] = 0.0;
              Output_Error(2, &net, ground_truth, &error);
```

```
                    net_error = net_error + error;

                    run_ann(&net, x2);
                    ground_truth[0] = 1.0;
                    ground_truth[1] = 1.0;
                    Output_Error(2, &net, ground_truth, &error);
                    net_error = net_error + error;

                    run_ann(&net, x3);
                    ground_truth[0] = 0.0;
                    ground_truth[1] = 1.0;
                    Output_Error(2, &net, ground_truth, &error);
                    net_error = net_error + error;

                    run_ann(&net, x4);
                    ground_truth[0] = 0.0;
                    ground_truth[1] = 0.0;
                    Output_Error(2, &net, ground_truth, &error);
                    net_error = net_error + error;

                    run_ann(&net, x5);
                    ground_truth[0] = 1.0;
                    ground_truth[1] = 0.0;
                    Output_Error(2, &net, ground_truth, &error);
                    net_error = net_error + error;

                    run_ann(&net, x6);
                    ground_truth[0] = 1.0;
                    ground_truth[1] = 0.0;
                    Output_Error(2, &net, ground_truth, &error);
                    net_error = net_error + error;

                    run_ann(&net, x7);
                    ground_truth[0] = 0.0;
                    ground_truth[1] = 1.0;
                    Output_Error(2, &net, ground_truth, &error);
                    net_error = net_error + error;

                    floatToInt(net_error, &d1, &d2, 5);
                    sprintf(Report_Message, "\r\nTotal Mean Squared
    Error: %d.%05d", (int)d1, (int)d2);
                    CDC_Fill_Buffer((uint8_t *) Report_Message,
          strlen(Report_Message));
                    }
}

sprintf(Report_Message, "\n\rTrain and Test Complete at Epoch:
                      %d\n", i);
```

```
        CDC_Fill_Buffer((uint8_t *) Report_Message,
                        strlen(Report_Message));
        /*
        * Delay of 5 seconds to permit user to view results
        */

        HAL_Delay(5000);
}
```

9. Generate Training Data

This function generates that data to be applied to training of EmbeddedML.

```
void generate_xorand(float *x, float *y, int i) {
        int k;

        k = rand() % 8;

        switch (k) {
        case 0:
                x[0] = 0.0;
                x[1] = 0.0;
                x[2] = 1.0;
                y[0] = 0.0;
                y[1] = 0.0;
                break;
        case 1:
                x[0] = 0.0;
                x[1] = 1.0;
                x[2] = 1.0;
                y[0] = 1.0;
                y[1] = 1.0;
                break;
        case 2:
                x[0] = 1.0;
                x[1] = 0.0;
                x[2] = 1.0;
                y[0] = 1.0;
                y[1] = 0.0;
                break;
        case 3:
                x[0] = 1.0;
                x[1] = 1.0;
                x[2] = 1.0;
                y[0] = 0.0;
```

```
            y[1] = 1.0;
            break;
        case 4:
            x[0] = 0.0;
            x[1] = 0.0;
            x[2] = 0.0;
            y[0] = 0.0;
            y[1] = 0.0;
            break;
        case 5:
            x[0] = 0.0;
            x[1] = 1.0;
            x[2] = 0.0;
            y[0] = 1.0;
            y[1] = 0.0;
            break;
        case 6:
            x[0] = 1.0;
            x[1] = 0.0;
            x[2] = 0.0;
            y[0] = 1.0;
            y[1] = 0.0;
            break;
        case 7:
            x[0] = 1.0;
            x[1] = 1.0;
            x[2] = 0.0;
            y[0] = 0.0;
            y[1] = 0.0;
            break;
```

```
        default:
                x[0] = 0.0;
                x[1] = 0.0;
                x[2] = 1.0;
                y[0] = 0.0;
                y[1] = 0.0;
                break;
        }

        /*
        * The XOR-AND system supplies three input values and
        * two Ground Truth output values.
        *
        * Since 6 input neurons form the neural network, the
        * remaining four Ground Truth values are set to zero.
        *
        */

        y[2] = 0.0;
        y[3] = 0.0;
        y[4] = 0.0;
        y[5] = 0.0;
}
```

10. Compute and Print Neural Network Outputs and Output Error

This function will be used to compute Neural Network output classification values and their error with respect to ground truth.

```
void Output_Error(int size, ANN *net, float * ground_truth, float *error) {

        int i;
        int32_t d1, d2;
        char msg[128];

        /*
        * Compute error as mean squared difference between
        * output and Ground Truth
        */

        *error = 0;
```

```
        for (i = 0; i < size; i++) {
                *error = *error
                                + (net->output[i] - ground_truth[i]) * (net->output[i] - ground_truth[i]);
        }
        *error = *error / size;

        floatToInt(net->output[0], &d1, &d2, 3);
        sprintf(msg, "\n\rOutput: %d.%03d", (int) d1, (int) d2);
        CDC_Fill_Buffer((uint8_t *) msg, strlen(msg));
        floatToInt(net->output[1], &d1, &d2, 3);
        sprintf(msg, "\t%d.%03d", (int) d1, (int) d2);
        CDC_Fill_Buffer((uint8_t *) msg, strlen(msg));
}
```

11. EmbeddedML Demonstration

The progress of EmbeddedML Neural Network training may be observed.

This is demonstrated by recording the Neural Network output for a set of inputs and computing and displaying output values and output error. Output error is expected to decrease as each training epoch is completed.

An example is shown here. For Inputs of {0, 1, 0} corresponding to Ground Truth outputs of 1, 0, the training and execution of the Neural Network is shown in Figure 4. The output Error is also shown in Figure 5.

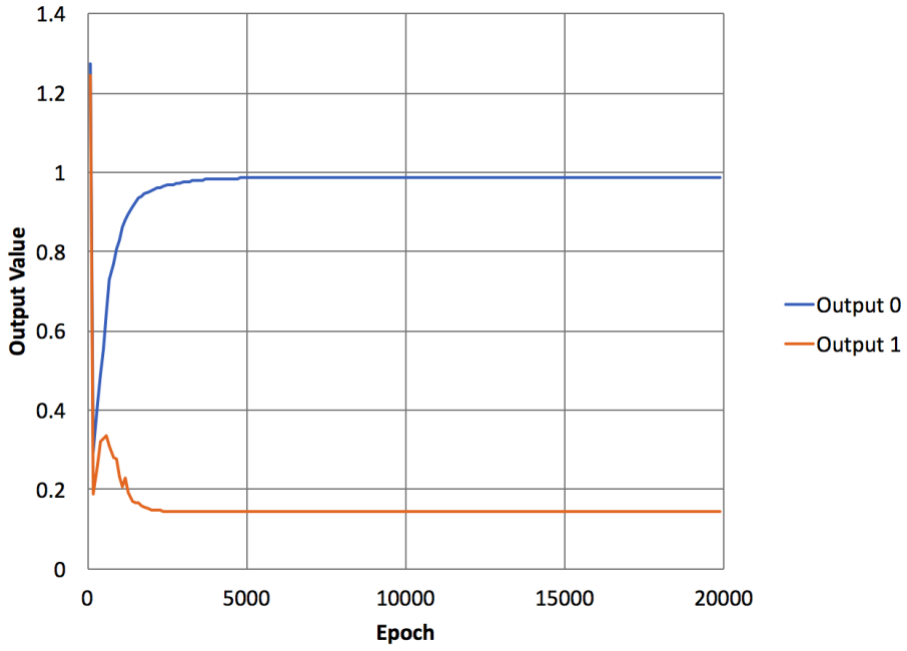


Figure 4. Output values of the Neural Network evolving towards Ground Truth with Training Epochs from 0 to 20,000.

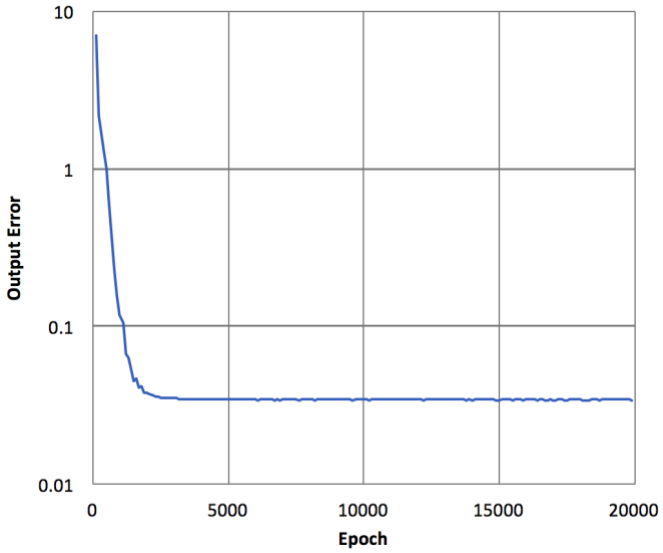


Figure 5. Output Error of the Neural Network evolving from 0 to 20,000.

12. EmbeddedML Application Installation

2.1 SensorTile EmbeddedML System Installation

1. **Download the project from this google drive**

https://drive.google.com/open?id=1gPmLUmteEw3IsTPzK6du0umJpy1FH96S

and decompress it to your working directory.

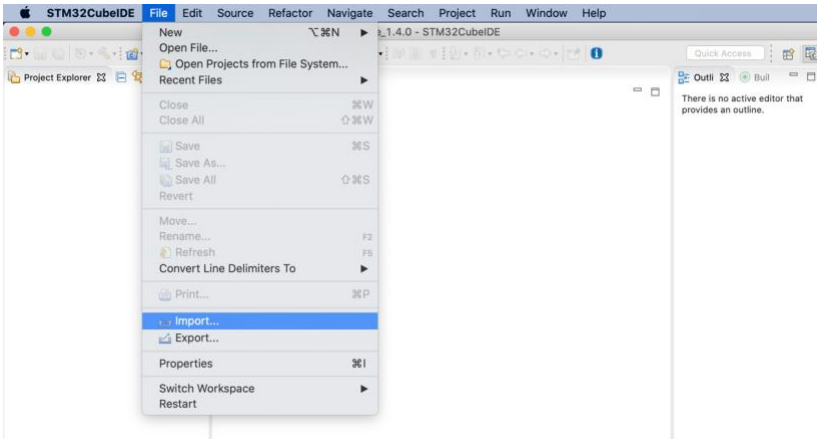
Note: Please download this project above. Please do not use the Projects from previous Tutorials.

2. Open the STM32CubeIDE, as instructed in the Tutorial 1**.** Select the same workspace as in Tutorial 1.

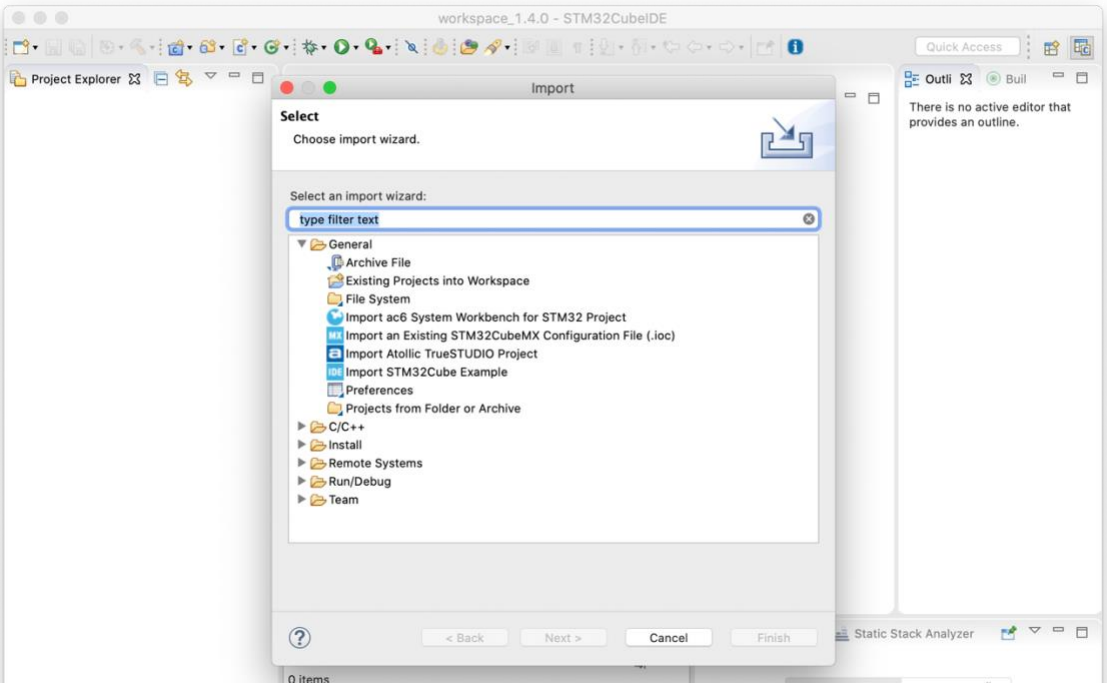
3. Once the IDE is open, first remove your current project.

4. Then, import the new Tutorial 10

5. Now, start the STM32CubeIDE Application. Select File > Import

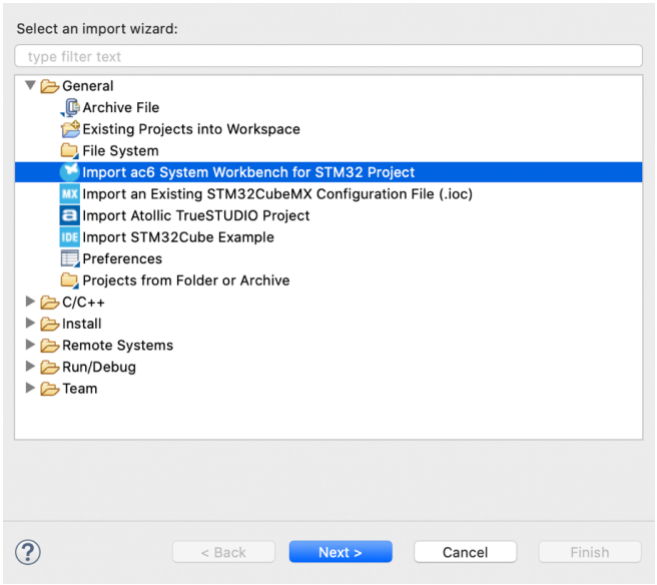


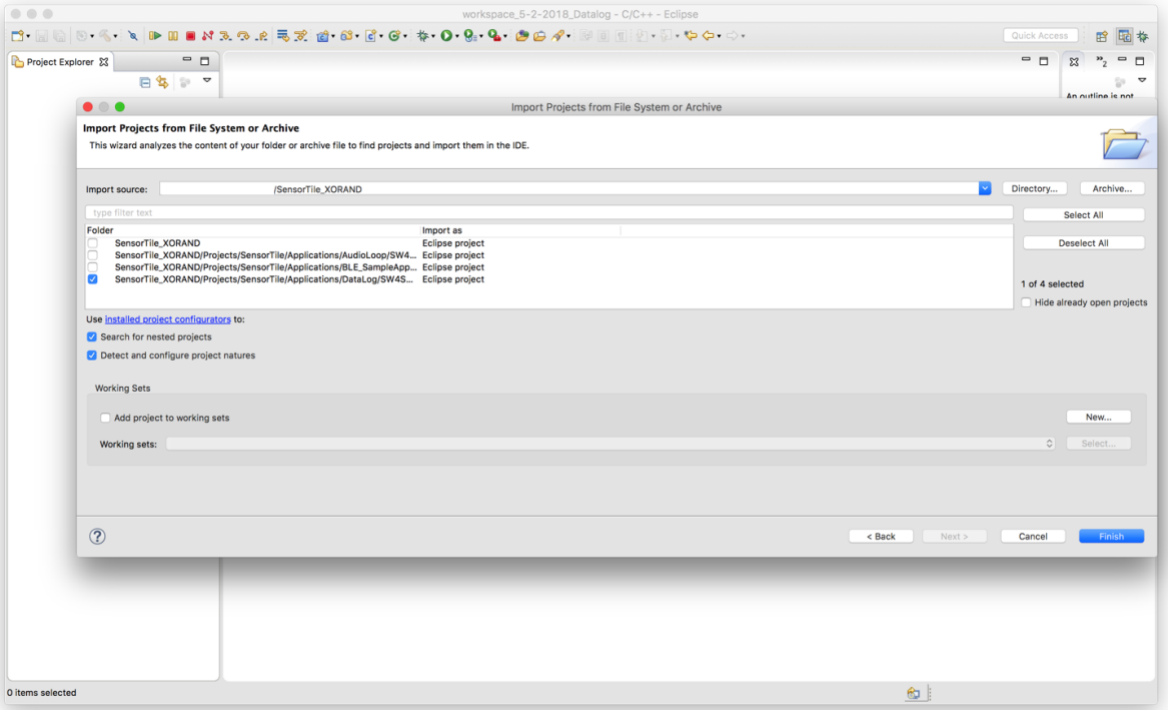
6. Then, Select General



7. Select Import ac6 System Workbench for STM32 Project

NOTE THIS SELECTION IS CRITICAL


Select an import wizard:
type filter text
General
Archive File
Existing Projects into Workspace
File System
Import ac6 System Workbench for STM32 Project
Import an Existing STM32CubeMX Configuration File (.ioc)
Import Atollic TrueSTUDIO Project
Import STM32Cube Example
Preferences
Projects from Folder or Archive
C/C++
Install
Remote Systems
Run/Debug
Team
< Back
Next >
Cancel
Finish



8. Then, select Project > Clean

9. Then, select Project > Build All

10. Then select Run > Debug As > STM32 Cortex-M C/C++

13. EmbeddedML Demonstration with SensorTile

The first lines of output will appear as below. The EmbeddedML system will continue to train for 200 Epochs and then execute and compute output values and Output Error. The outputs will evolve towards their Ground Truth values and the Error will fall towards zero.

```
Embedded ML XOR-AND System

Train and Test Start in 5 seconds

Epoch: 0
Output: 0.613 0.532
Output: 1.273 1.239
Output: 1.468 1.387
Output: 1.508 1.418
Output: 1.236 1.185
Output: 1.269 1.201
Output: 1.450 1.366
Output: 1.508 1.418
```

Figure 6. Output of the XOR-AND Neural Network application.

14. EmbeddedML Assignment Description

The following exercise will begin to provide experience with EmbeddedML

Consider the XOR-XOR system shown in Figure 7. Clearly, this is quite similar to the XOR-AND system. Find its Ground Truth table and modify the XOR-AND project to enable training and execution of a Neural Network replicating the response of the XOR-XOR system.

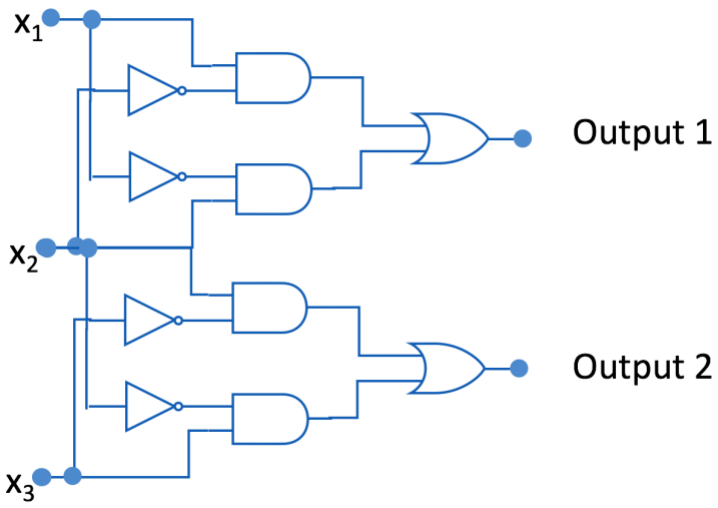


Figure 7. The XOR-XOR system with two XOR gate constructions operating on three inputs and presenting two outputs.

The development of this new system will require modifications to the function, generate_xorand() to account for the change in ground truth values for each input and also to the definition of ground truth appearing elsewhere in this system associated with system training.

15. EmbeddedML Assignment Submission

For assignment submission, obtain a screen capture of these items:

1) The source code segment showing the new function replacing generate_xorand()

2) The source code segment showing the updated ground truth values appearing in training and evaluation of error.

3) The output similar to that of Figure 6.