STMicroelectronics SensorTile Tutorial: IoT Machine Learning Motion Classification

Table of Contents

1. Introduction to This Tutorial

This Tutorial describes the development of a *self-learning* motion classification system. This harnesses the EmbeddedML system to enable a SensorTile to train on orientation states of the SensorTile. This system then autonomously recognizes orientation.

Tutorial 10 introduced EmbeddedML for the application of developing neural network classification systems that replicate the operation of a logical structure combining XOR and AND function gates.

This Tutorial will include acquisition of data from motion sensors for both training of a neural

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

It is important to have completed Tutorials 1 through Tutorial 4 and Tutorial 10.

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

2. Neural Network Motion Classification

In the previous Tutorial, EmbeddedML was be demonstrated by developing a Neural Network that replicates the operation of a logic gate circuit. For any set of input values, all output values were known.

There were three inputs creating eight possible combinations that each corresponded to a unique training instance. For each training instance, there was an output consisting of two states. The output values form the Ground Truth for each instance of training. The Neural Network was then trained on the combination of eight training instance and the known Ground Truth.

The input values corresponding to each training instance and its Ground Truth are referred to as *features* associated with the instance.

The objective for development of a Neural Network classifier is to enable classification of a correct output, for each instance. Training of the Neural Network requires development of features describing each instance and acquiring the Ground Truth output corresponding to each instance.

For Motion Classification, the computation of features for each instance is based on the processing of sensor signals that occur during the period of each instance.

For this first Tutorial on Motion Classification, signal features will include the acceleration values measured by the SensorTile sensor systems during the instance.

3. Neural Network Motion Classification Objective

The objective of this first application is to determine the change in Orientation of the SensorTile from an initial orientation to a final orientation.

For this motion classification application, measurement of the orientation of the SensorTile can be derived from measurement of the acceleration values for the X, Y, and Z axes exploiting the known acceleration due to gravity acting on the accelerometer. This method for measuring

orientation is accurate only if the accelerometer is stationary and not acted on by inertial acceleration induced by motion.

This application provides an ideal introduction to complete, autonomous Neural Network training and execution. The applications that follow this address challenges that are not directly solved by straightforward computation and rather rely entirely on the capability of machine learning on SensorTile.

4. SensorTile Accelerometer Sensors and Orientation Measurement

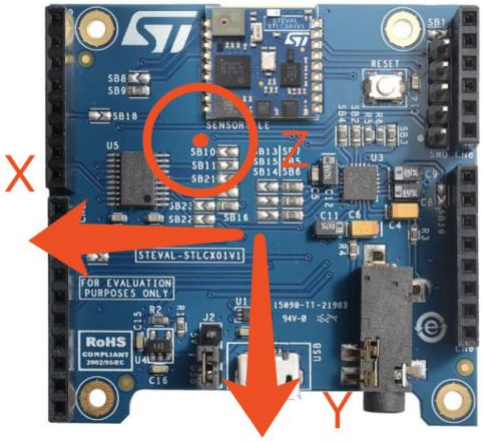


Figure 1. SensorTile Accelerometer Sensor Axis Orientation. Acceleration of the SensorTile in the direction indicated will be measured as a positive signal by the associated accelerometer.

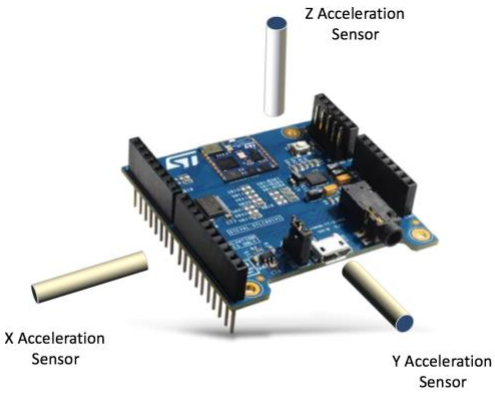


Figure 2. A perspective view of the SensorTile Accelerometer Sensor Axis Orientation. Acceleration of the SensorTile in the direction indicated will be measured as a positive signal by the associated accelerometer.

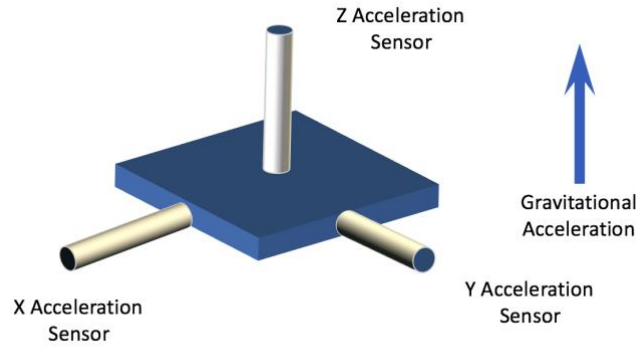


Figure 3. SensorTile with Z-Axis vertical and X and Y axes level and perpendicular with respect to the acceleration due to gravity. Here, the Z-axis accelerometer will produce a value of 1000 milli-g. However, the X and Y axis accelerometers will produce zero values since the projection of the gravitational acceleration vector on these axes is zero.

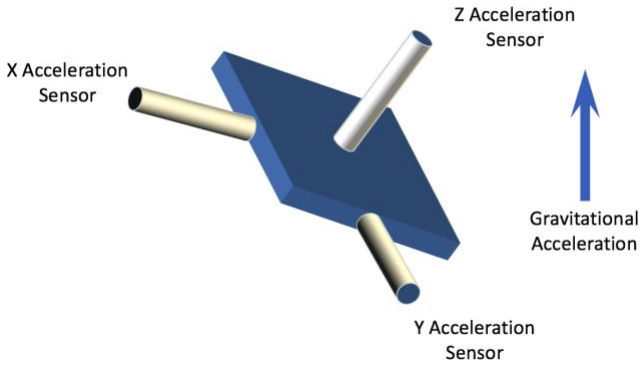


Figure 4. SensorTile with all axes at a non-zero angle with respect to vertical. All accelerometers will produce non-zero values since the projection of the gravitational acceleration vector on these axes are non-zero.

Figure 1 shows the orientation of the accelerometer axes relative to the SensorTile itself and its carrier board. Note the orientation of the board and its USB connector in interpreting signals. Figure 2 displays a perspective view. Figure 3 shows the SensorTile oriented with its Z-axis accelerometer axis vertical. Acceleration acting on the SensorTile is the combination of acceleration due to gravity and acceleration due to motion and change in velocity. However, if the SensorTile is motionless, then the only acceleration acting is that due to gravity. The orientation of Figure 4 will produce a gravitational acceleration aligned with the Z-axis accelerometer producing a value of 1g and appearing as 1000 milli-g. However, since the X- and Y-Axis accelerometer sensors are oriented perpendicular to gravity, their output is zero.

In the case of the orientation measurement, the SensorTile will be rotated to a position similar to that of Figure 4. Now, each axis will produce a non-zero value equal to the projection of gravitational acceleration onto the respective axis.

5. Feature Data Acquisition for Motion Classification

The first step in Motion Classification is the acquisition of acceleration values that create features. This applies the methods introduced in Tutorial 3 to measure acceleration.

The feature for this application will measure the difference in acceleration between an initial state where the user holds the SensorTile in a starting orientation, and a final state where the user rotates the SensorTile and holds it in a final orientation.

The initial state will occur with the SensorTile held level (or nearly level) and the final state will occur with the SensorTile tilted about a direction by an angle of about 45 degrees, for example.

Now, the features required for determining orientation include only the X and Y axis accelerations. Here, the projection of the gravitational acceleration vector on the sensor X and Y axis accelerometer axes provides information on the angles defining the orientation of the SensorTile.

The classification features will be derived then from these accelerations.

The features, a_x and a_y, are the differences between the X and Y axis accelerations, respectively, between a final acceleration value when motion is complete and an initial value.

$$a_x = a_{final-x} - a_{initial-x}$$
$$a_y = a_{final-y} - a_{initial-y}$$
$$a_z = 0$$

Since no information is required from Z-Axis acceleration, this is set to zero.

6. Feature Computation for Motion Classification

The same Neural Network as included in Tutorial 10 is applied here. This has a topology of 3 input neurons, 9 hidden layer neurons, and 6 output neurons. Therefore, three features are required to be supplied to the three input neurons.

The features must now be normalized before being presented to the Neural Network classifier.

This normalization will ensure that all values corresponding to the X and Y axis acceleration values lie within the range of -1.0 to +1.0. Normalized feature values, F_i , will be derived from acceleration values, a_i , for $i = x \; or \; y$ with:

$$F_1 = \frac{a_x}{\sqrt{{a_x}^2 + {a_y}^2}}$$
$$F_2 = \frac{a_y}{\sqrt{{a_x}^2 + {a_y}^2}}$$

$$F_3 = 0$$

The third feature is not required for distinguishing the multiple orientations. This is set to zero for this system.

7. Acquisition of Training Data Features

The Neural Network training process is shown in Figure 5. This starts with a notification to the user (via both an LED illumination and a text message) to orient the SensorTile in its initial state and then in its final state. The process then continues with a measurement of feature values that correspond to training Ground Truth.

There will be six instances of orientation and therefore six sets of Ground Truth values.

After acquisition of feature values, these are normalized by the Softmax function. The process continues until all training is complete for each of the six instances.

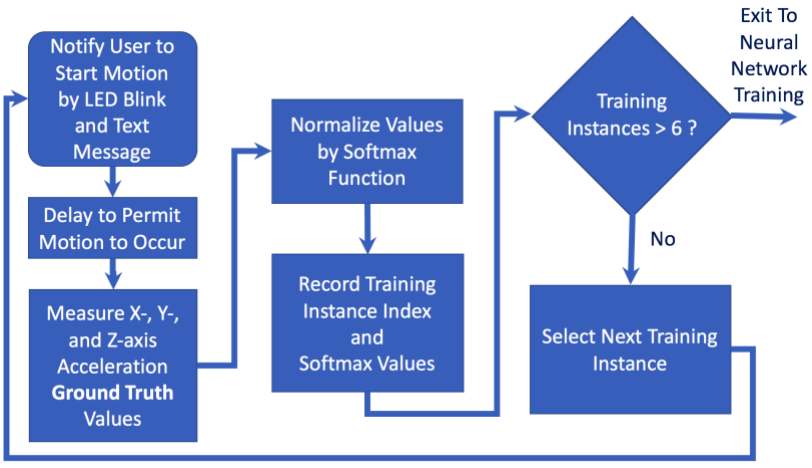


Figure 5. The data acquisition and feature computation process. This system executes for 6 training instances and then exits to the Neural Network training process.

8. Neural Network Training for EmbeddedML

The same Neural Network as included in Tutorial 10 is applied here. This has a topology of 3 input neurons, 9 hidden layer neurons, and 6 output neurons.

9. Neural Network Training for EmbeddedML

After completion of acquisition and normalization of features, the Neural Network training may start. The EmbeddedML system follows the process of Figure 6.

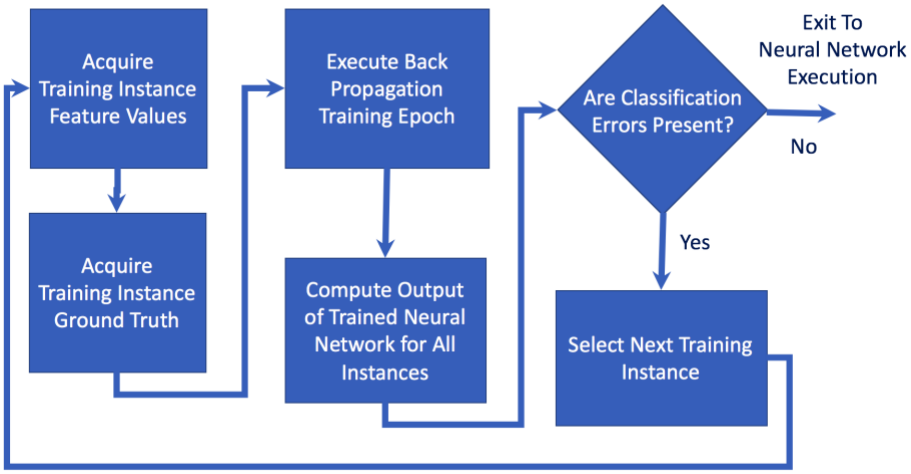


Figure 6. The Neural Network training process executing with the features and Ground Truth obtained from the data acquisition process of Figure 5.

10. Neural Network Execution

Finally, Neural Network execution proceeds. Here, as in the acquisition of training data, the user is notified to complete an orientation action. Then, feature values are computed by normalization. These are then presented to the Neural Network execution system.

The Neural Network execution system proceeds by forward propagation computation to determine output classifications based on input values and the trained Neural Network.

The user is notified of the predicted result by both a text message and an LED blink pattern.

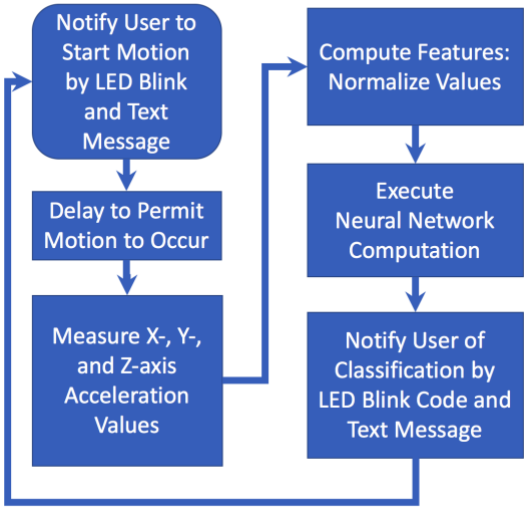


Figure 7. The Neural Network execution process operating on data acquired and with the Neural Network trained by the process of Figure 6.

11. EmbeddedML Motion Classification Software Systems

It is important to review the application software components now.

First, acceleration values are returned as arguments of the getAccel function. It takes an argument corresponding to a pointer address to the accelerometer system handler data structure.

```
void getAccel(void *handle, int *xyz) {
	uint8_t id;
	SensorAxes_t acceleration;
	uint8_t status;

	BSP_ACCELERO_Get_Instance(handle, &id);
	BSP_ACCELERO_IsInitialized(handle, &status);

	if (status == 1) {
		if (BSP_ACCELERO_Get_Axes(handle, &acceleration) == COMPONENT_ERROR) {
			acceleration.AXIS_X = 0;
			acceleration.AXIS_Y = 0;
			acceleration.AXIS_Z = 0;
		}
		xyz[0] = (int) acceleration.AXIS_X;
		xyz[1] = (int) acceleration.AXIS_Y;
		xyz[2] = (int) acceleration.AXIS_Z;
	}
}
```

Second, the notification to the user and acquisition of features is supplied by the Feature_Extraction_State_0 () function.

```
void Feature_Extraction_State_0(void *handle, int * ttt_1, int * ttt_2,
		int * ttt_3, int * ttt_mag_scale) {

	int ttt[3];
	int ttt_initial[3];
	int axis_index;
	float accel_mag;
	char  msg[128];

	/*
	* Acquire acceleration values prior to motion
	*/
	getAccel(handle, ttt_initial);

	sprintf(msg, "\r\nStart Motion to new Orientation and hold while LED On");
	CDC_Fill_Buffer((uint8_t *) msg, strlen(msg));
	BSP_LED_On(LED1);
```

```
        HAL_Delay(3000);

        /*
        * Acquire final state acceleration values after motion
        * Notify user of motion complete
        */
        getAccel(handle, ttt);
        sprintf(msg, "\r\n Motion Complete");
        CDC_Fill_Buffer((uint8_t *) msg, strlen(msg));
        BSP_LED_Off(LED1);

        /*
        * Compute acceleration magnitude including only X and Y axes
        */

        accel_mag = 0;
                for (axis_index = 0; axis_index < 3; axis_index++) {
                   accel_mag = accel_mag + pow((ttt[axis_index] - ttt_initial[axis_index]), 2);
                }
        accel_mag = sqrt(accel_mag);

        /*
        * Compute feature values including only X and Y axes and
        * setting Z-axis feature to zero
        */
        *ttt_1 = ttt[0] - ttt_initial[0];
        *ttt_2 = ttt[1] - ttt_initial[1];
        *ttt_3 = 0;
        *ttt_mag_scale = (int)(accel_mag);

        return;
}
```

Please refer to Figure 5 while reviewing this function.

Then, you may examine the TrainOrientation() function. An excerpt here is shown below.

```
        sprintf(msg1, "\r\n\r\n\r\nTraining Start in 2 seconds ..");
        CDC_Fill_Buffer((uint8_t *) msg1, strlen(msg1));
        BSP_LED_Off(LED1);
        HAL_Delay(2000);

        num_train_data_cycles = 1;

        for (k = 0; k < num_train_data_cycles; k++) {
                for (i = 0; i < 6; i++) {

                        sprintf(msg1, "\r\nMove to Start Position - Wait for LED On");
                        CDC_Fill_Buffer((uint8_t *) msg1, strlen(msg1));
                        HAL_Delay(START_POSITION_INTERVAL);

                        switch (i) {
                        HAL_Delay(1000);

                case 0:
                        sprintf(msg1, "\r\nMove to Orientation 1 on LED On");
                        CDC_Fill_Buffer((uint8_t *) msg1, strlen(msg1));

                        Feature_Extraction_State_0(handle, &ttt_1, &ttt_2, &ttt_3,
                                                   &ttt_mag_scale);

                        break;
```

Observe that there will be 6 training instances (enumerated 0 through 5). For each instance, the Feature_Extraction_State_0() function acts to measure Ground Truth.

After each instance is complete, normalization occurs and a training dataset is computed.

```
sprintf(msg1, "\r\nAccel %i\t\%i\%i", ttt_1, ttt_2, ttt_3);
CDC_Fill_Buffer((uint8_t *) msg1, strlen(msg1));

XYZ[0] = (float) ttt_1;
XYZ[1] = (float) ttt_2;
XYZ[2] = (float) ttt_3;

motion_softmax(net->topology[0], XYZ, xyz);

training_dataset[i][k][0] = xyz[0];
training_dataset[i][k][1] = xyz[1];
training_dataset[i][k][2] = xyz[2];
```

The normalization is computed by motion_softmax, below.

```
void motion_softmax(int size, float *x, float *y) {
        float norm;
        norm = sqrt((x[0] * x[0]) + (x[1] * x[1]) + (x[2] * x[2]));
        y[0] = x[0] / norm;
        y[1] = x[1] / norm;
        y[2] = x[2] / norm;
}
```

12. EmbeddedML Motion Neural Network Training

The Neural Network is trained as the next step in the Train_Orientation() function.

A further excerpt is shown below.

The Neural Network defined by the data structure, net, training Ground Truth data, and an indicator of the specific Orientation instance corresponding to Ground Truth

The assignment of an Orientation to Ground Truth is defined in the_Motion_N arrays, as will be shown below. These arrays assign values to each Output Neuron. The Neural Network prediction appears as a value for each of the six Output Neurons.

This sequence is shown below

```
/*
```

```
* Enter NN training
*/

float _Motion_1[6] = { 1.0, 0.0, 0.0, 0.0, 0.0, 0.0 };
float _Motion_2[6] = { 0.0, 1.0, 0.0, 0.0, 0.0, 0.0 };
float _Motion_3[6] = { 0.0, 0.0, 1.0, 0.0, 0.0, 0.0 };
float _Motion_4[6] = { 0.0, 0.0, 0.0, 1.0, 0.0, 0.0 };
float _Motion_5[6] = { 0.0, 0.0, 0.0, 0.0, 1.0, 0.0 };
float _Motion_6[6] = { 0.0, 0.0, 0.0, 0.0, 0.0, 1.0 };

sprintf(msg1, "\r\n\r\nTraining Start\r\n");
CDC_Fill_Buffer((uint8_t *) msg1, strlen(msg1));

for (k = 0; k < num_train_data_cycles; k++) {

        i = 0;
        while (i < training_cycles) {
                for (j = 0; j < 6; j++) {

                        for (n = 0; n < 3; n++) {
                                training_data[j][n] = training_dataset[j][k][n];
                        }

                        if ((i % 20 == 0 && i < 100) || i % 100 == 0) {
                                char print_train_time[128];
                                sprintf(print_train_time,
                                                "\r\n\r\nTraining Epochs: %d\r\n", i);
                                CDC_Fill_Buffer((uint8_t *) print_train_time,
                                                strlen(print_train_time));

                                LED_Code_Blink(0);

                                net_error = 0;
                                for (m = 0; m < 6; m++) {
                                        run_ann(net, training_dataset[m][k]);
                                        printOutput_ANN(net, m, &error);
                                        if (error == 1) {
                                                net_error = 1;
                                        }
                                }

                                sprintf(msg1, "\r\nError State: %i\r\n",
                                                net_error);
                                CDC_Fill_Buffer((uint8_t *) msg1, strlen(msg1));
                                if (net_error == 0) {
                                        return;
                                }

                        }

                        switch (j) {

                        case 0:
                                train_ann(net, training_dataset[j][k], _Motion_1);
                                break;
                        case 1:
                                train_ann(net, training_dataset[j][k], _Motion_2);
                                break;
                        case 2:
                                train_ann(net, training_dataset[j][k], _Motion_3);
                                break;
                        case 3:
                                train_ann(net, training_dataset[j][k], _Motion_4);
                                break;
                        case 4:
                                train_ann(net, training_dataset[j][k], _Motion_5);
                                break;
```

```
                                        case 5:
                                                train_ann(net, training_dataset[j][k], _Motion_6);
                                                break;
                                        default:
                                                break;
                                        }                                                       i++;
                                        HAL_Delay(5);
                                }

                        }
```

Finally, the Neural Network, described by the net data structure, is tested for accuracy by executing the Neural Network with test data in this sequence shown above:

```
for (m = 0; m < 6; m++) {
        run_ann(net, training_dataset[m][k]);
        printOutput_ANN(net, m, &error);
        if (error == 1) {
                net_error = 1;
        }
}
```

This sequence selects each training data set instance included in the array training_data[][] and assigns these to a three element array, test_NN[ ].

Then, run_ann() is executed with with an argument of the now trained Neural Network, the motion identifier associated with the measured features, and the training_dataset[m][k]array.

The next step, is the evaluation of Neural Network prediction and reliability of prediction.

13. EmbeddedML Motion Classification Performance

The determination of the Neural Network prediction and prediction accuracy is performed by the function printOutput_ANN(net, m, &error)

This called with the integer index, m, indicating the motion instance and also the address of the error variable that will return the reported error. This function will also print performance characteristics.

Now, the Neural Network prediction is important. However, equally important is an assessment of the prediction reliability.

If the Neural Network training has been successful, and if a test example matches a training example, then one of the six Output Neurons will produce a value near unity while others produce values near zero.

However, if training were not successful, or if a test example corresponds to a motion for which the Neural Network was not trained, then all of the Output Neuron values will be small.

Therefore, it is important to assess accuracy. This is performed by multiple methods.

First, if no Output Neuron exhibits a value greater than a threshold value (here set to 0.1) then no prediction is returned and the condition of insufficient or incorrect training should be notified to the user.

Second, an inaccurate classification may be indicated by the occurrence where two Output Neuron values are nearly equal and also yet greater than the values of others. This corresponds to an instance where accuracy and reliability of prediction are reduced. Neither of the two selections is a certain prediction and the condition of insufficient or incorrect training should be notified to the user.

Third, classification accuracy may be indicated by a statistical measure of the significance of the value of the selected, greatest Output Neuron value, in comparison to others. This is classification metric, referred to here as a Z-Score.

The Z-Score of a value in comparison to a set of values, is the ratio of the difference between the value and the mean of values, to the standard deviation of all values.

First, the mean of the output values is

$$\mu = \frac{1}{N}\sum_{n=1}^{N} Output(n)$$

The Population Standard Deviation of the output values is

$$\sigma = \sqrt{\frac{1}{N}\sum_{n=1}^{N} (Output(n) - \mu)^2}$$

Now, the Z – Score for a specific output value is

$$Z(i) = \frac{Output(i) - \mu}{\sigma}$$

The Z-Score is a measure of the difference between the output value and the mean of all values in units of the standard deviation.

If the difference is large and the standard deviation small, this indicates that a given output is statistically significant relative to others.

However, if there was large variation in the distribution of output values, then the standard deviation will be large. This reduces the Z-Score.

The Z-Score is compared with a threshold. If the Z-Score is less than the threshold, a classification error is indicated. Otherwise, this Neural Network classification is indicated as successful.

The printOutput_ANN(net, m, &error) function performs a series of computations. The steps include:

1) Determination of the Neural Network prediction by selection of the Output Neuron providing the largest value of the six Output Neurons.
2) Determination of the Output Neuron showing the next greatest value – the variable point is set to this value.
3) Computation of the mean value of all outputs
4) Computation of the mean value of all outputs ***other*** than the greatest
5) Computation of the Root Mean Square value of all outputs for purposes of computation of the Standard Deviation
6) Computation of the Z-Score.

The following sequence appears for selection of the largest output value, identified by index value, loc, and value, point.

```
count = 0;
mean_output = 0;
for (i = 0; i < net->topology[net->n_layers - 1]; i++) {
        mean_output = mean_output + (net->output[i]);
        if (net->output[i] > point && net->output[i] > 0.1) {
                point = net->output[i];
                loc = i;
        }
        count++;
}
```

The following sequence appears for selection of the next largest output value of next_max,

```
next_max = 0;
for (i = 0; i < net->topology[net->n_layers - 1]; i++) {
        if (i == loc) {
                continue;
        }
        if (net->output[i] > next_max && net->output[i] > 0.1) {
                next_max = net->output[i];
        }
}
```

Mean output is computed

```
mean_output = (mean_output) / (count);
```

The Population Standard Deviation (rms_output) is computed

```
rms_output = 0;

for (i = 0; i < net->topology[net->n_layers - 1]; i++) {
        rms_output = rms_output + pow((net->output[i] - mean_output), 2);
}
```

Finally, Z-Score is computed

```
rms_output = sqrt(rms_output / count);
if (rms_output != 0) {
        classification_metric = (point - mean_output) / rms_output;
} else {
        classification_metric = 0;
}
```

Results are then printed and also error conditions are evaluated and reported.

This test determines if the selected Output Neuron value matches the supplied ground truth, provided as the input_state argument.

```
if (loc != input_state) {
        *error = 1;
        sprintf(dataOut, "\t Classification Error");
        CDC_Fill_Buffer((uint8_t *) dataOut, strlen(dataOut));
}
```

The second test determines whether in the event that the Output Neuron value matches the supplied ground truth that:

1) The Z-Score is less than a threshold
2) The ratio of the maximum value Output Neuron to the next maximum value Output Neuron is less than a threshold.

```
if ((loc == input_state) &&
                ((classification_metric < CLASSIFICATION_ACC_THRESHOLD) ||
                ((point / next_max) < CLASSIFICATION_DISC_THRESHOLD))) {
        *error = 1;
        sprintf(dataOut, "\t Classification Accuracy Limit");
        CDC_Fill_Buffer((uint8_t *) dataOut, strlen(dataOut));
}
```

Training occurs over all instances for a maximum of 2000 epochs. The results of training are presented to the user at each of 100 epochs.

14. EmbeddedML Motion Classification Operation

After training is complete, the EmbeddedML system is now capable of classification of unknown inputs and predicting actual orientation.

This occurs in the Accel_Sensor_Handler() function. This function takes the arguments of a pointer address to the handler data structure associated with accelerometer measurement. This also includes the pointer address to the net data structure describing the complete Neural Network. This also returns a value, prev_loc, that corresponds to the actual prediction.

Focusing on the code segment that enables execution, the first step, shown below, is the call to the Feature_Extraction_State_0() function returning data from one orientation change.

This is followed by motion_softmax() normalization.

Finally, run_ann() executes forward propagation computation based on the newly trained Neural Network.

Finally, the identification of the most likely prediction of the Neural Network is made. Specifically, the six output neuron values appear as net->output[i] where i ranges from 0 to 5 including the six values. The greatest of these values corresponds to the prediction of the correct classification among the six possibilities.

Given a value for this prediction, loc, the user notification may occur including blinking of the LED with an identifying code and printing of this value.

```
Feature_Extraction_State_0(handle, &ttt_1, &ttt_2, &ttt_3,
                        &ttt_mag_scale);

XYZ[0] = (float) ttt_1;
XYZ[1] = (float) ttt_2;
XYZ[2] = (float) ttt_3;

motion_softmax(net->topology[0], XYZ, xyz);

run_ann(net, xyz);

for (i = 0; i < net->topology[net->n_layers - 1]; i++) {
        if (net->output[i] > point && net->output[i] > 0.1) {
                point = net->output[i];
                loc = i;
        }
}
```

15. EmbeddedML Motion Classification Project

As in other EmbeddedML projects, this project, is based on an extension of the DataLog application to include EmbeddedML.

SensorTile EmbeddedML System Installation

1. **Download the project for Tutorial 11 from the Google Drive**

   https://drive.google.com/open?id=1IWkwmeVwvyZyM03Y37d-2B6hD2ejfpgM

   and decompress it to your working directory.

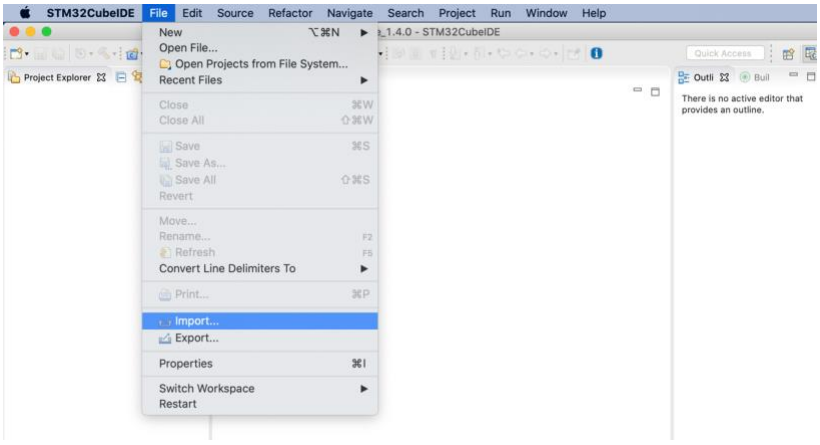
   Note: Please download this project above. Please do not use the Projects from previous Tutorials.

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

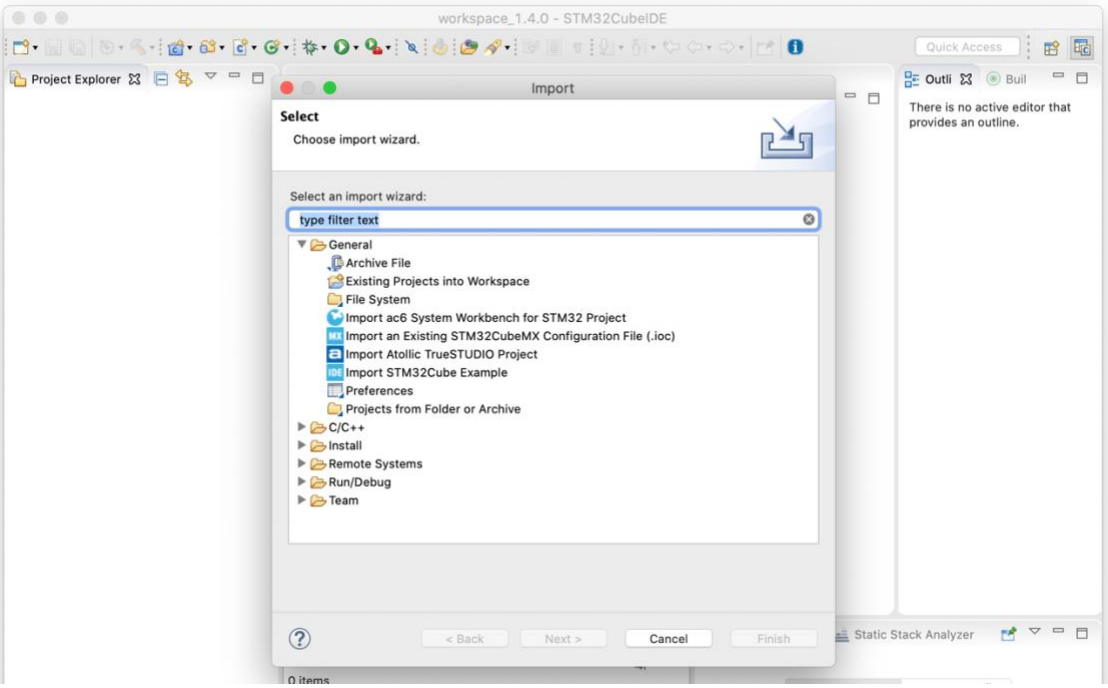
3. Once the IDE is open, first remove your current project.

4. Then, import the new Tutorial 11

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

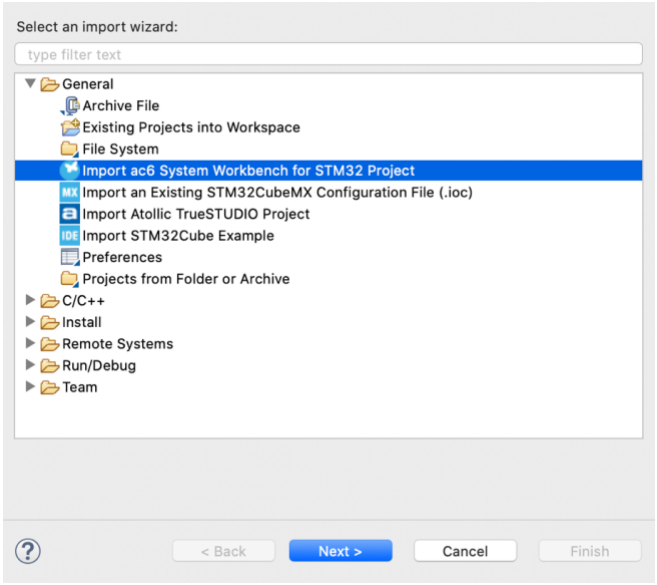


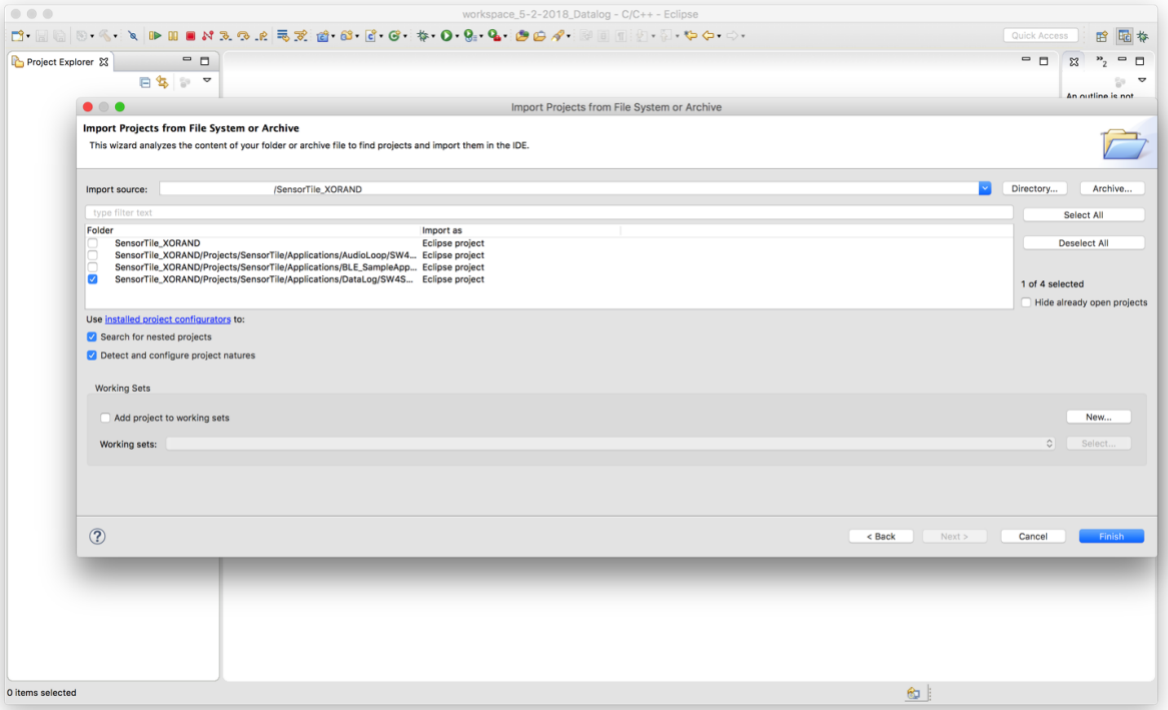
6. Then, Select General



7. Select Import ac6 System Workbench for STM32 Project

NOTE THIS SELECTION IS CRITICAL





8. Then, select Project > Clean
9. Then, select Project > Build All
10. Then select Run > Debug As > STM32 Cortex-M C/C++

16. Assignment Part 1

With an initial state of the SensorTile level, train this system to predict six motions that correspond to tilting the SensorTile to about a 45 degree angle about the X axis (as in Figure 8(a)), about the Y axis, and at an angle formed by a line bisecting the X and Y axes as shown in Figure 8(b).

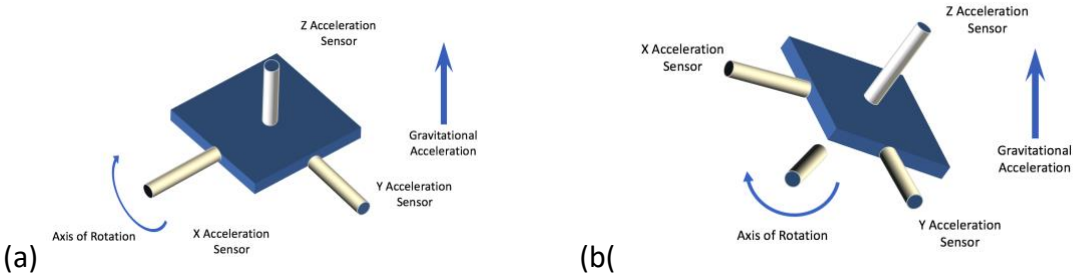


Figure 8. Example of Orientation change induced by rotation about X-Axis shown in (a) or about a line bisecting the X and Y axes shown in (b).

With an initial state other than level (that may be 45 degrees or 90 degrees relative to level) define 6 orientation changes and show that these can be predicted.

Note that due to the normalization of acceleration values, it may be possible to detect the polarity of an orientation change in angle, but not its magnitude. While a rotation direction may be resolved, a rotation magnitude corresponding to a rotation of 45 degrees may not be distinguished from a rotation of 60 degrees, for example.

There is a first video showing experimental training of six orientations is here:

https://youtu.be/f9SOp7ltbGk

(Please note that it is *not required to separate the Nucleo and SensorTile. Instead, use them as they have been in previous Tutorials).*

Please carefully note how motion is performed as commanded by the LED blink illumination.

Please carefully note the 6 different orientation actions.

Then, please also observe that after completion of the sixth motion, training occurs. The LED flashes appear at the end of each 500 training epochs. When these cease the system is trained.

Now, there is a second video here that displays the errors that occur when training is incorrect.

https://youtu.be/yN52LY1CJ60

Here, the first orientation motion is repeated twice, and this labeled as two different orientations. Note that training epochs continue through 2000 epochs and that classification shows errors.

It is very important at this stage of development to gain experience in Neural Network training and execution that may *fail.*

Submit a screen capture of the terminal output showing classification results. This will appear similar to the screen shown in Figure 9. The tests may be performed in any order.

Specifically, try training the SensorTile system with a set of instances where the motions in two instances are not different, but are the same. This will lead to a fault.

How is this revealed?

Also, if motions are not precise, the SensorTIle training and execution phases will be unpredictable.

Please observe this carefully.

It is as important to understand when Machine Learning fails as when it is successful.

17. Assignment Part 2

Now, this system measures only X and Y axis acceleration values and computes features that are related to Orientation angle.

However, this system may not discriminate between Orientations that are executed with the SensorTile oriented upright with its Z-Axis vertical, or oriented downward with its Z-Axis directed downward.

Modify this system to include the Z-Axis acceleration value in the feature computation, resulting in three features.

Demonstrate that it is possible to now train for classification of Orientation changes that include the SensorTile orientation with upward and downward Z-Axis.

Submit a screen capture showing the source code modification.

Submit a screen capture of the terminal output showing classification results. This will appear similar to the screen shown in Figure 9. The tests may be performed in any order.

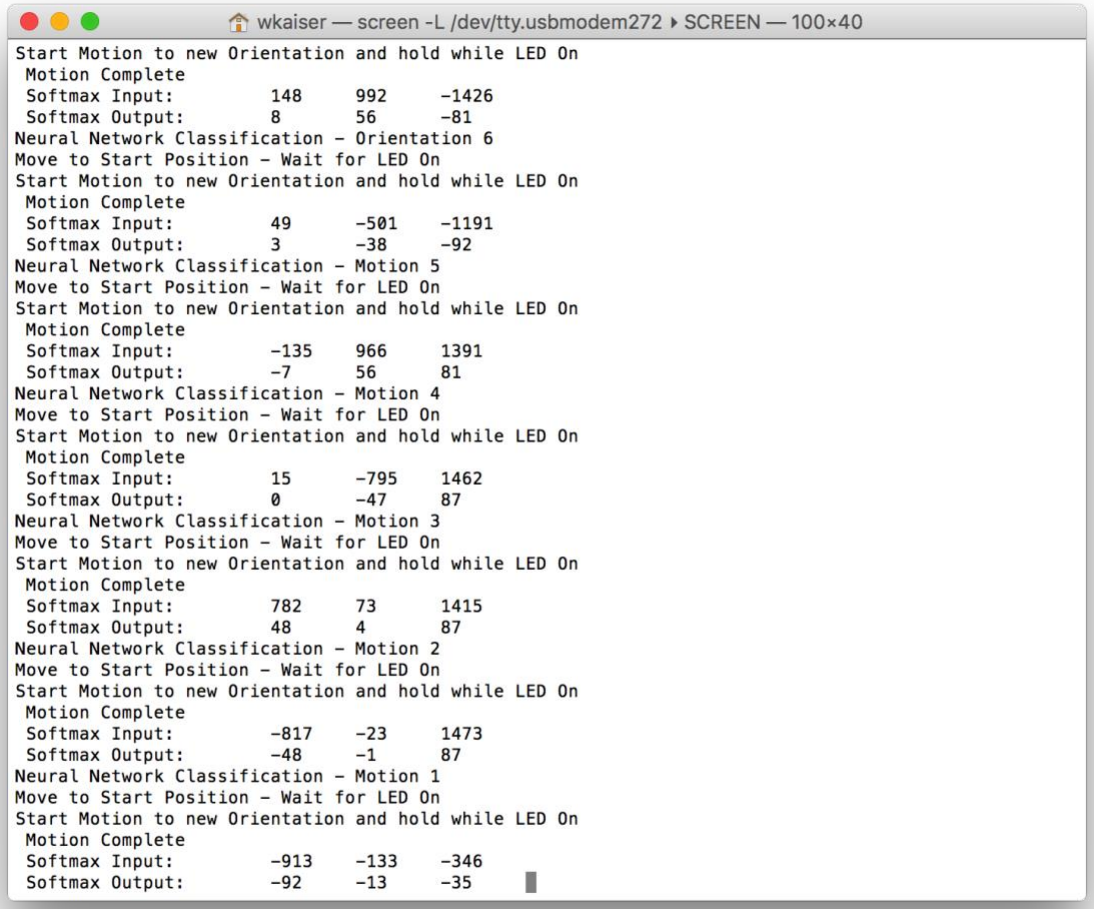


Figure 9. Example output of the EmbeddedML Orientation Classification system.