Machine Learning Application Design using STM32 MCU's

DAY 4 : Training a Neural Network Part 2

Sponsored by

Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Group Chat’ by maximizing the chat widget in your dock.
- Submit questions for the lecturer using the Q&A widget. They will follow-up after the lecture portion concludes.

Course Sessions

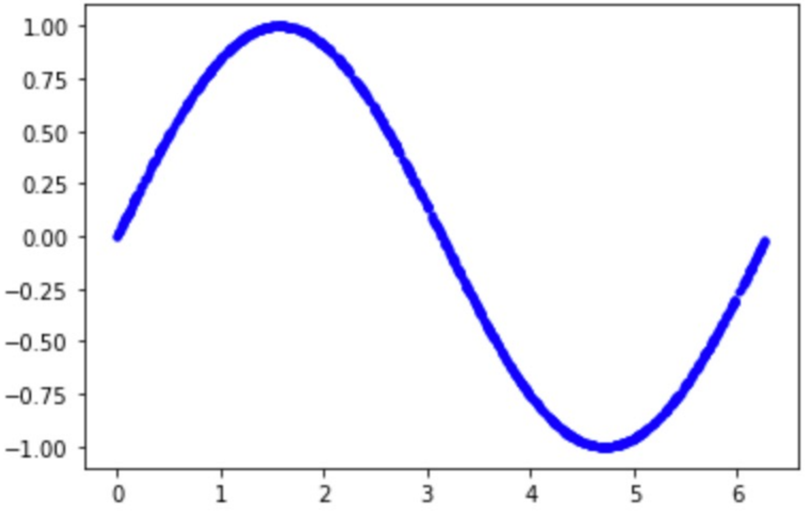
- Introduction to Machine Learning on MCU's
- Capturing, Cleaning and Digital Signal Processing Data
- Training a Neural Network Part 1
- **Training a Neural Network Part 2**
- Running an Inference on Target

TensorFlow Lite for Microcontrollers

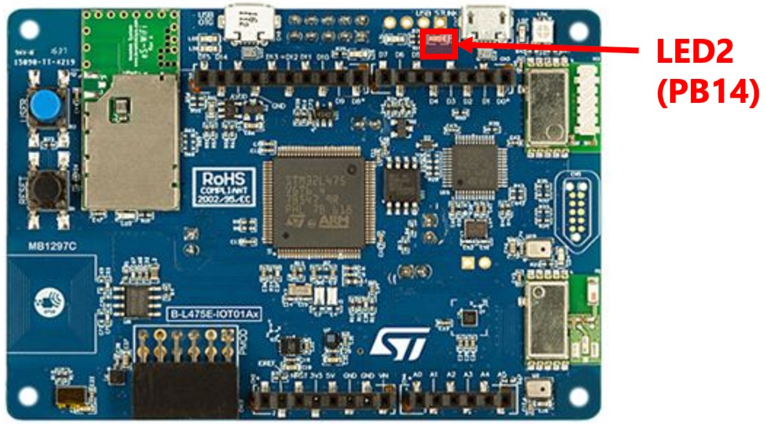
- Runs machine learning models on microcontrollers
- Core run-time is ~16kB
- Does not require an OS (can run baremetal)
- Written in C++ 11
- Several example cases already available:
  - Hello World
  - Keyword spotting (Micro speech)
  - Gesture detection (Magic wand)
  - Person detection (Image processing)

Hello World

- Shall demonstrate running a model
- Shall demonstrate controlling hardware (LED)



STM32L475 IoT Discovery Kit (B-L475E-IOT01A)



What is your idea of a good hello world program?

- Tests a simple hardware feature?
- Tests a simple software feature?
- Minimum demonstratable feature?
- Other

Option #1 - Tensorflow Lite for Microcontrollers

Can download Tensorflow and examples by cloning:

https://github.com/tensorflow/tensorflow

```
Projects — -zsh — 97×11
beningo@Jacobs-MacBook-Pro Projects % git clone https://github.com/tensorflow/tensorflow

Cloning into 'tensorflow'...
remote: Enumerating objects: 1134096, done.
remote: Counting objects: 100% (323/323), done.
remote: Compressing objects: 100% (220/220), done.
remote: Total 1134096 (delta 164), reused 169 (delta 103), pack-reused 1133773
Receiving objects: 100% (1134096/1134096), 667.32 MiB | 13.17 MiB/s, done.
Resolving deltas: 100% (924832/924832), done.
Updating files: 100% (24712/24712), done.
beningo@Jacobs-MacBook-Pro Projects %
```

Option #2 - Google Colab

Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with
- Zero configuration required
- Free access to GPUs
- Easy sharing

Often used in:
- Data science
- Machine learning
- etc

train_hello_world_model.ipynb

Google Colab training file

Train a Simple TensorFlow Lite for Microcontrollers model

This notebook demonstrates the process of training a 2.5 kB model using TensorFlow and converting it for use with TensorFlow Lite for Microcontrollers.

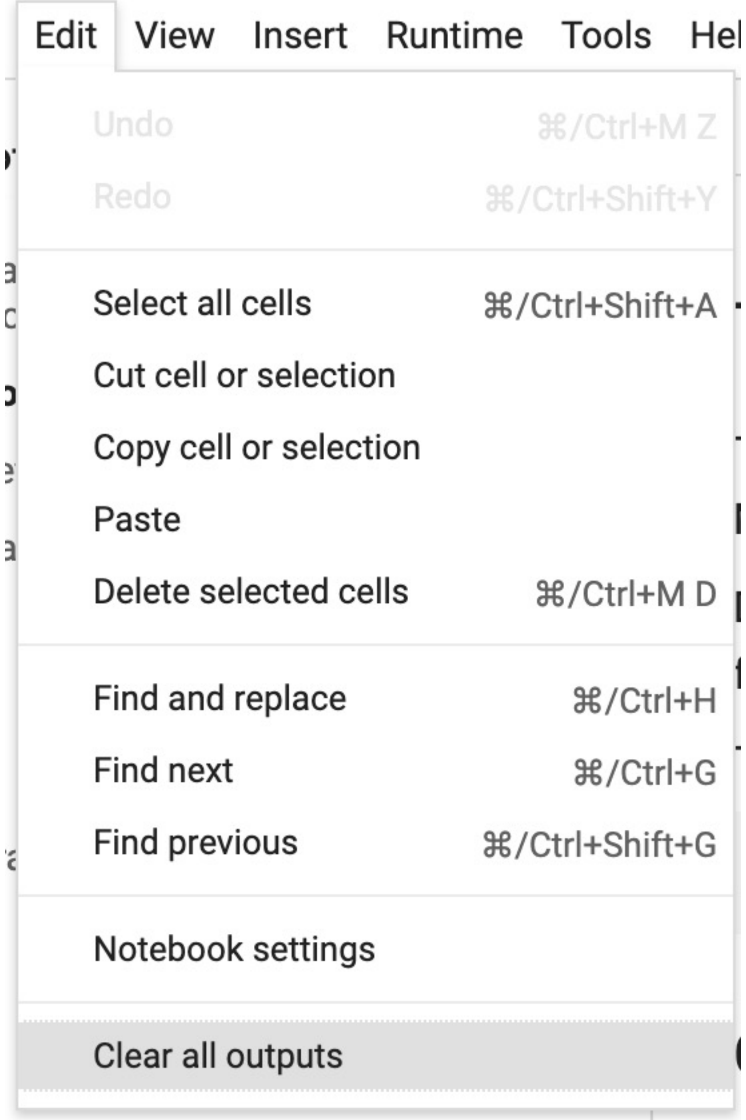
Deep learning networks learn to model patterns in underlying data. Here, we're going to train a network to model data generated by a sine function. This will result in a model that can take a value, `x`, and predict its sine, `y`.

The model created in this notebook is used in the hello_world example for TensorFlow Lite for MicroControllers.

Run in Google Colab   View source on GitHub

IMPORTANT!

Before you start to run the notebook, make sure that the notebook output has been cleared!



Will be attempting to run this notebook . . .

- Live during the session
- Later after the session
- Never, just listening in
- Other

Training the Model

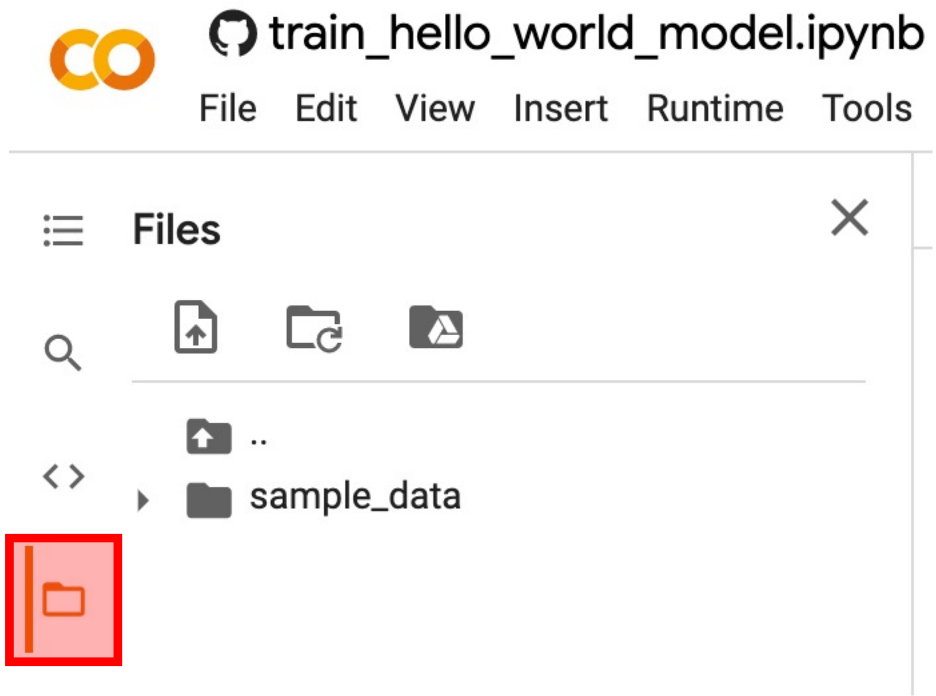
Configure Defaults

Run the cell

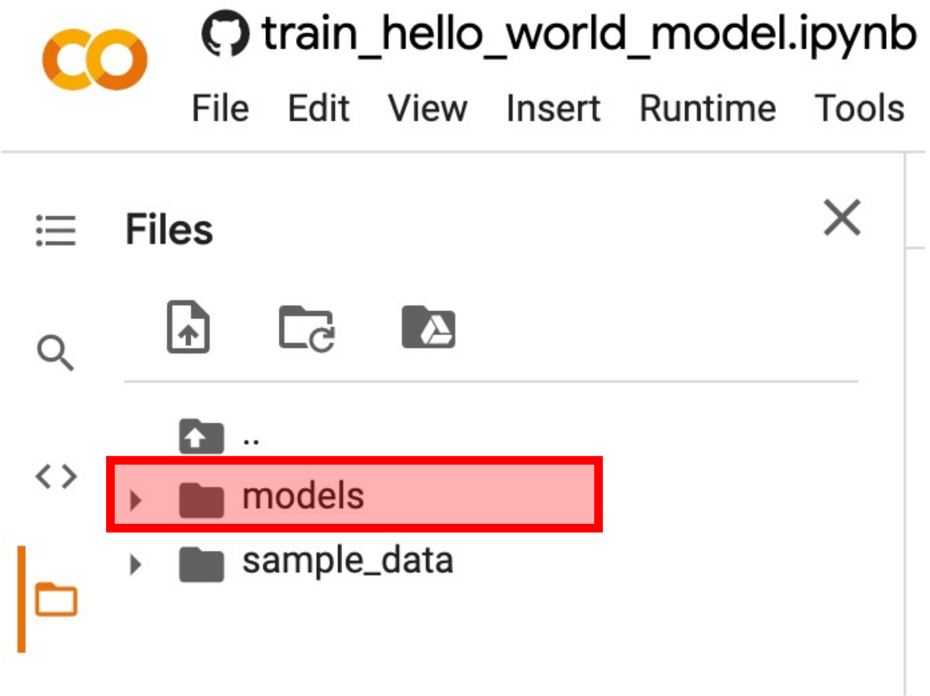
```
# Define paths to model files
import os
MODELS_DIR = 'models/'
if not os.path.exists(MODELS_DIR):
    os.mkdir(MODELS_DIR)
MODEL_TF = MODELS_DIR + 'model'
MODEL_NO_QUANT_TFLITE = MODELS_DIR + 'model_no_quant.tflite'
MODEL_TFLITE = MODELS_DIR + 'model.tflite'
MODEL_TFLITE_MICRO = MODELS_DIR + 'model.cc'
```

Training the Model

Before



After



Training the Model

Setup Environment

Install Dependencies

```
! pip install tensorflow==2.4.0
```

```
Collecting tensorflow==2.4.0
```

Import Dependencies

```
[ ] # TensorFlow is an open source machine learning library
    import tensorflow as tf

    # Keras is TensorFlow's high-level API for deep learning
    from tensorflow import keras
    # Numpy is a math library
    import numpy as np
    # Pandas is a data manipulation library
    import pandas as pd
    # Matplotlib is a graphing library
    import matplotlib.pyplot as plt
    # Math is Python's math library
    import math

    # Set seed for experiment reproducibility
    seed = 1
    np.random.seed(seed)
    tf.random.set_seed(seed)
```

Training the Model

1. Generate Data

The code in the following cell will generate a set of random `x` values, calculate their sine values, and display them on a graph.

```
[ ] # Number of sample datapoints
    SAMPLES = 1000

    # Generate a uniformly distributed set of random numbers in the range from
    # 0 to 2π, which covers a complete sine wave oscillation
    x_values = np.random.uniform(
        low=0, high=2*math.pi, size=SAMPLES).astype(np.float32)

    # Shuffle the values to guarantee they're not in order
    np.random.shuffle(x_values)

    # Calculate the corresponding sine values
    y_values = np.sin(x_values).astype(np.float32)

    # Plot our data. The 'b.' argument tells the library to print blue dots.
    plt.plot(x_values, y_values, 'b.')
    plt.show()
```

Training the Model

2. Add Noise

Since it was generated directly by the sine function, our data fits a nice, smooth curve.

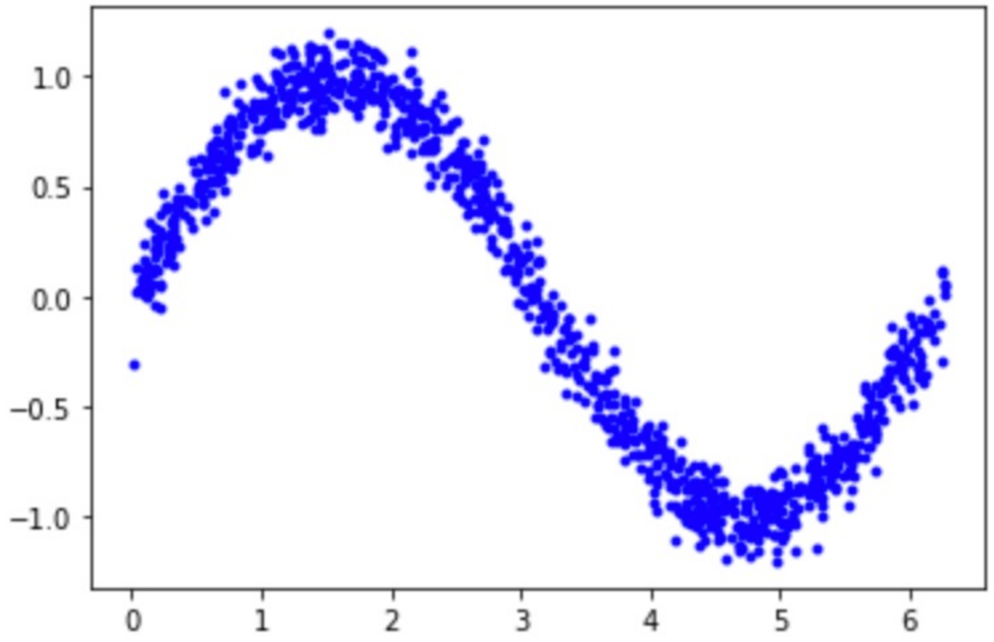
However, machine learning models are good at extracting underlying meaning from messy, real world data. To demonstrate this, we can add some noise to our data to approximate something more life-like.

In the following cell, we'll add some random noise to each value, then draw a new graph:

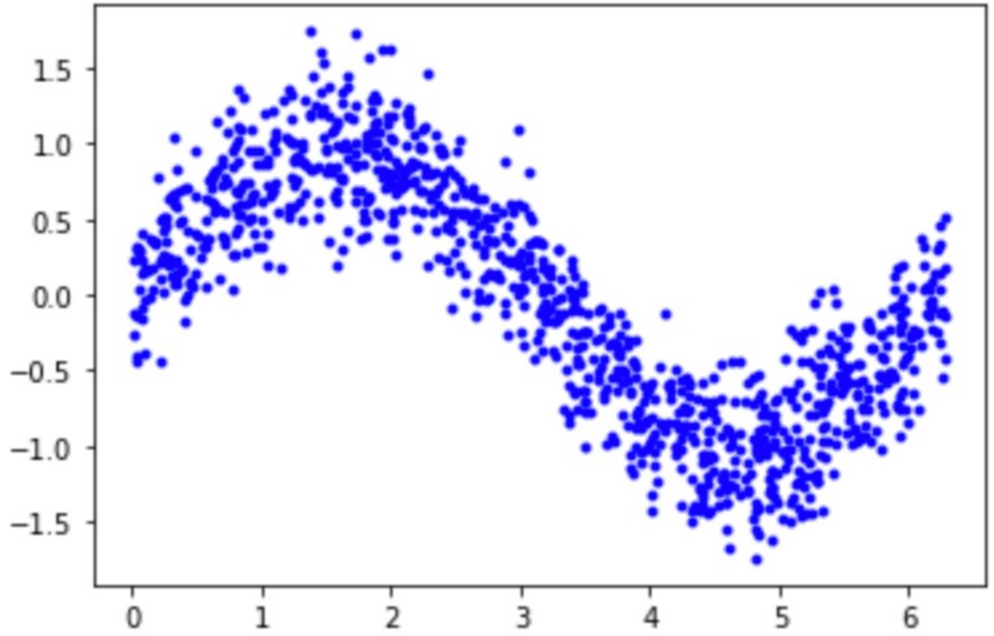
```
[ ] # Add a small random number to each y value
    y_values += 0.1 * np.random.randn(*y_values.shape)

    # Plot our data
    plt.plot(x_values, y_values, 'b.')
    plt.show()
```

Training the Model

0.1

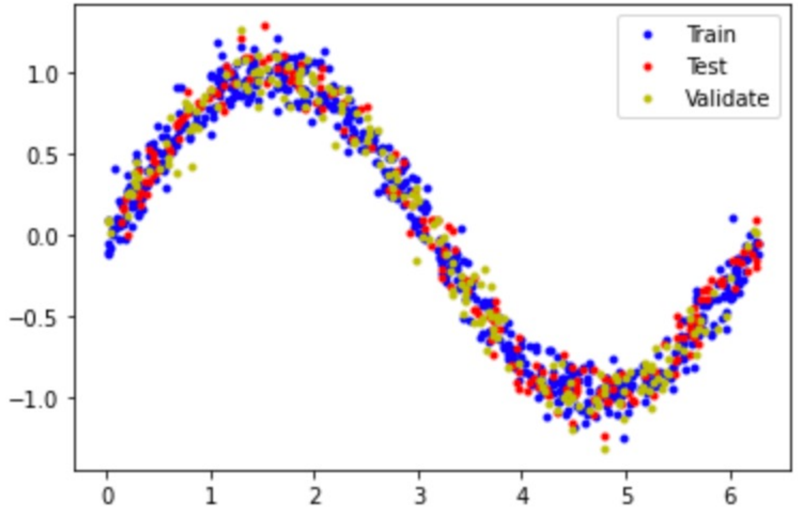


0.3



Training the Model

```
[ ] # We'll use 60% of our data for training and 20% for testing. The remaining 20%
    # will be used for validation. Calculate the indices of each section.
    TRAIN_SPLIT =  int(0.6 * SAMPLES)
    TEST_SPLIT = int(0.2 * SAMPLES + TRAIN_SPLIT)

    # Use np.split to chop our data into three parts.
    # The second argument to np.split is an array of indices where the data will be
    # split. We provide two indices, so the data will be divided into three chunks.
    x_train, x_test, x_validate = np.split(x_values, [TRAIN_SPLIT, TEST_SPLIT])
    y_train, y_test, y_validate = np.split(y_values, [TRAIN_SPLIT, TEST_SPLIT])

    # Double check that our splits add up correctly
    assert (x_train.size + x_validate.size + x_test.size) ==  SAMPLES

    # Plot the data in each partition in different colors:
    plt.plot(x_train, y_train, 'b.', label="Train")
    plt.plot(x_test, y_test, 'r.', label="Test")
    plt.plot(x_validate, y_validate, 'y.', label="Validate")
    plt.legend()
    plt.show()
```



Training the Model

Note: To learn more about how neural networks function, you can explore the Learn TensorFlow codelabs.

The code in the following cell defines our model using Keras, TensorFlow's high-level API for creating deep learning networks. Once the network is defined, we *compile* it, specifying parameters that determine how it will be trained:

```
[ ] # We'll use Keras to create a simple model architecture
    model_1 = tf.keras.Sequential()

    # First layer takes a scalar input and feeds it through 8 "neurons". The
    # neurons decide whether to activate based on the 'relu' activation function.
    model_1.add(keras.layers.Dense(8, activation='relu', input_shape=(1,)))

    # Final layer is a single neuron, since we want to output a single value
    model_1.add(keras.layers.Dense(1))

    # Compile the model using the standard 'adam' optimizer and the mean squared error or 'mse' loss function for regression.
    model_1.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

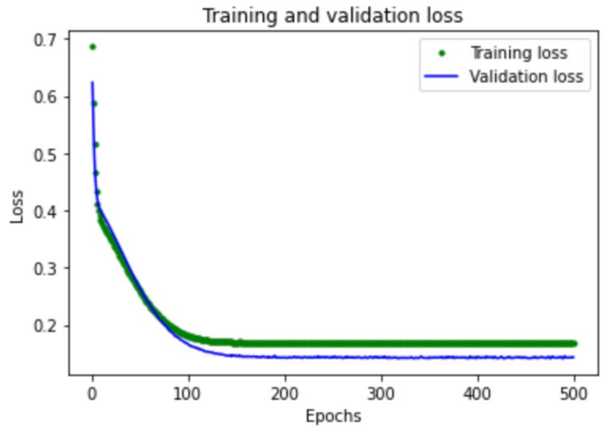
Training the Model

```
[ ] # Train the model on our training data while validating on our validation set
    history_1 = model_1.fit(x_train, y_train, epochs=500, batch_size=64,
                            validation_data=(x_validate, y_validate))
```

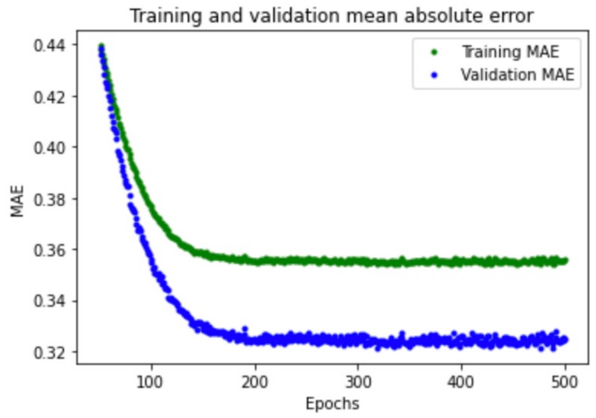
```
Epoch 494/500
10/10 [==============================] - 0s 6ms/step - loss: 0.1721 - mae: 0.3546 - val_loss: 0.1424 - val_mae: 0.3238
Epoch 495/500
10/10 [==============================] - 0s 6ms/step - loss: 0.1809 - mae: 0.3705 - val_loss: 0.1426 - val_mae: 0.3240
Epoch 496/500
10/10 [==============================] - 0s 7ms/step - loss: 0.1595 - mae: 0.3459 - val_loss: 0.1424 - val_mae: 0.3238
Epoch 497/500
10/10 [==============================] - 0s 7ms/step - loss: 0.1726 - mae: 0.3576 - val_loss: 0.1431 - val_mae: 0.3247
Epoch 498/500
10/10 [==============================] - 0s 7ms/step - loss: 0.1676 - mae: 0.3575 - val_loss: 0.1425 - val_mae: 0.3239
Epoch 499/500
10/10 [==============================] - 0s 7ms/step - loss: 0.1697 - mae: 0.3566 - val_loss: 0.1437 - val_mae: 0.3256
Epoch 500/500
10/10 [==============================] - 0s 6ms/step - loss: 0.1684 - mae: 0.3548 - val_loss: 0.1431 - val_mae: 0.3248
```

Training the Model

```python
# Draw a graph of the loss, which is the distance between
# the predicted and actual values during training and validation.
train_loss = history_1.history['loss']
val_loss = history_1.history['val_loss']

epochs = range(1, len(train_loss) + 1)

plt.plot(epochs, train_loss, 'g.', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



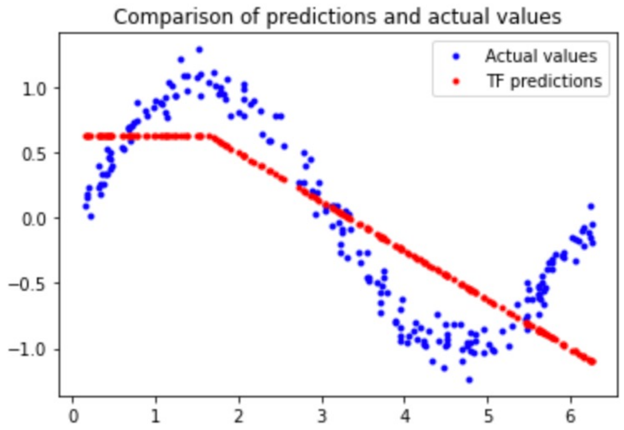

```python
plt.clf()

# Draw a graph of mean absolute error, which is another way of
# measuring the amount of error in the prediction.
train_mae = history_1.history['mae']
val_mae = history_1.history['val_mae']

plt.plot(epochs[SKIP:], train_mae[SKIP:], 'g.', label='Training MAE')
plt.plot(epochs[SKIP:], val_mae[SKIP:], 'b.', label='Validation MAE')
plt.title('Training and validation mean absolute error')
plt.xlabel('Epochs')
plt.ylabel('MAE')
plt.legend()
plt.show()
```



Training the Model

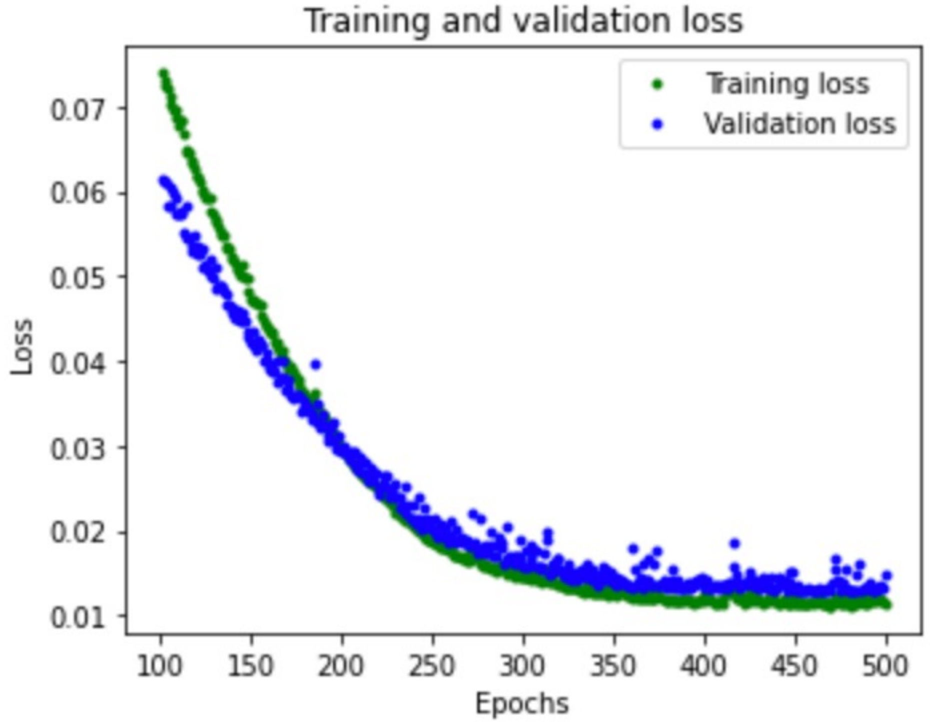
```
# Calculate and print the loss on our test dataset
test_loss, test_mae = model_1.evaluate(x_test, y_test)

# Make predictions based on our test dataset
y_test_pred = model_1.predict(x_test)

# Graph the predictions against the actual values
plt.clf()
plt.title('Comparison of predictions and actual values')
plt.plot(x_test, y_test, 'b.', label='Actual values')
plt.plot(x_test, y_test_pred, 'r.', label='TF predictions')
plt.legend()
plt.show()
```

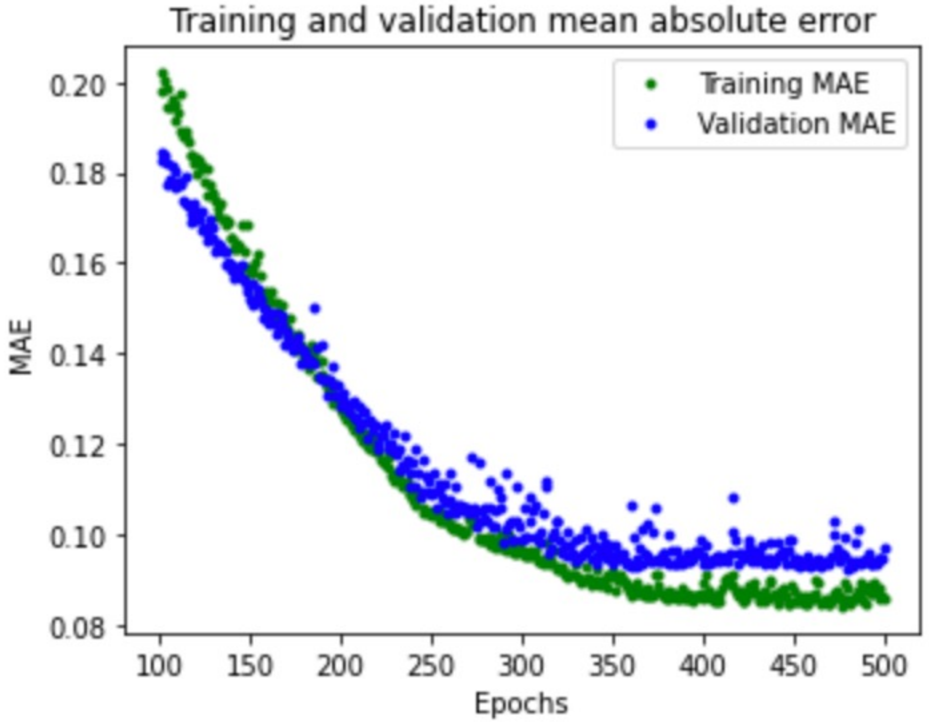
```
7/7 [==============================] - 0s 2ms/step - loss: 0.1935 - mae: 0.3768
```



Training the Model

Update the Model and run again



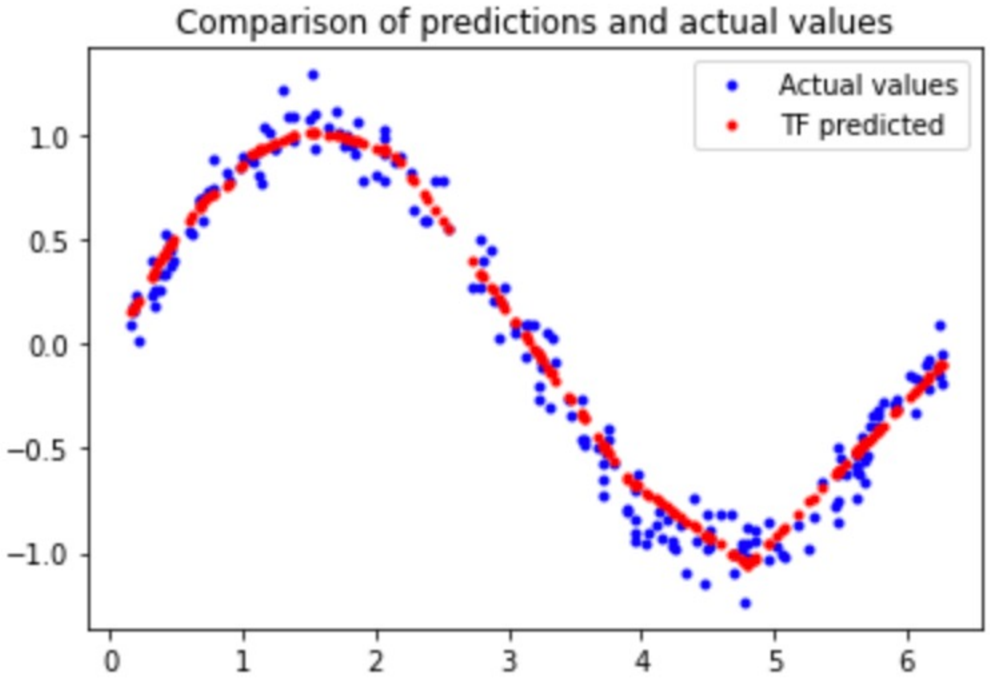


Training the Model

```
# Calculate and print the loss on our test dataset
test_loss, test_mae = model.evaluate(x_test, y_test)

# Make predictions based on our test dataset
y_test_pred = model.predict(x_test)

# Graph the predictions against the actual values
plt.clf()
plt.title('Comparison of predictions and actual values')
plt.plot(x_test, y_test, 'b.', label='Actual values')
plt.plot(x_test, y_test_pred, 'r.', label='TF predicted')
plt.legend()
plt.show()

model_1.save(MODEL_TF+"model.h5")
```

Training the Model

```
7/7 [==============================] - 0s 2ms/step - loss: 0.0120 - mae: 0.0891
```



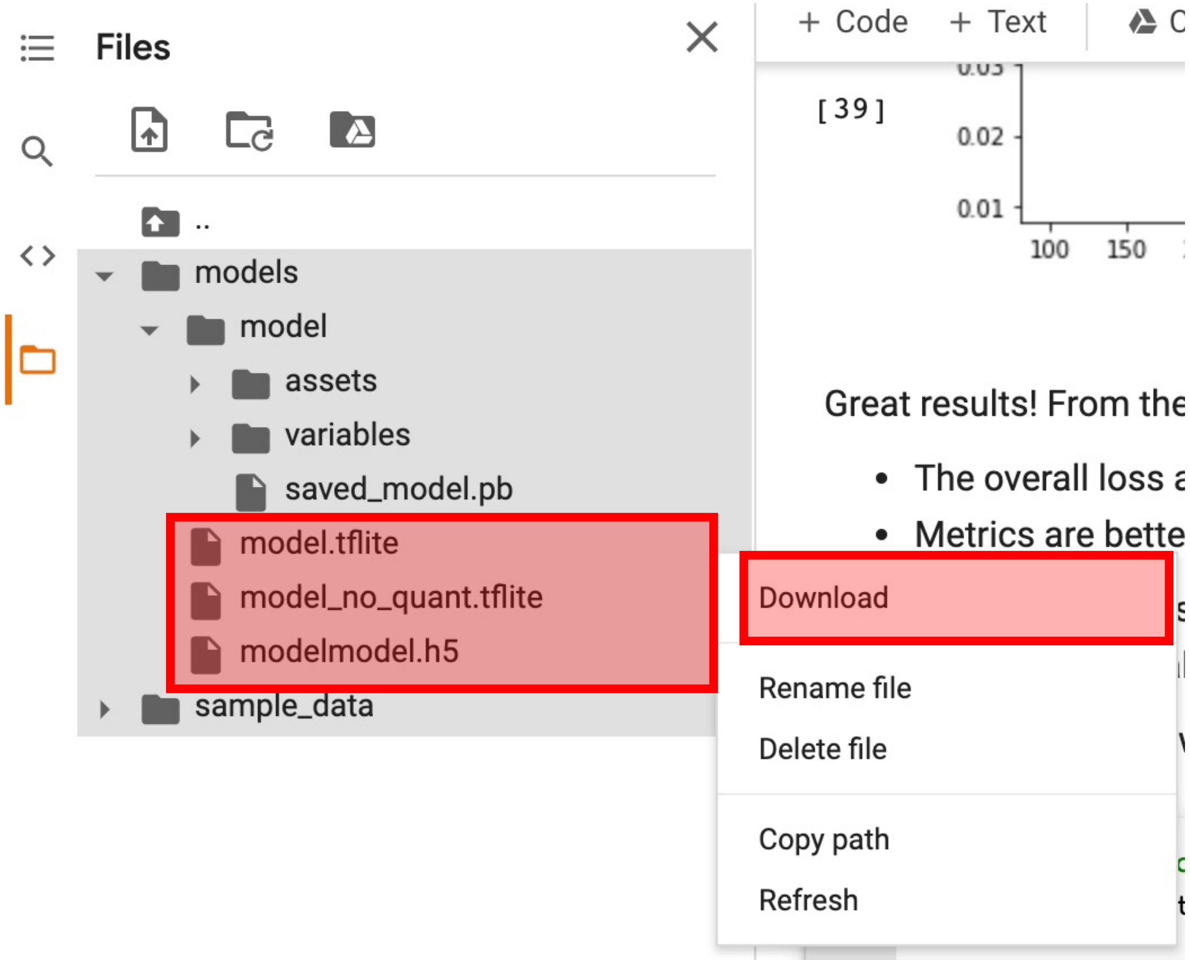
Generate a TensorFlow Lite Model

- Run through and run the blocks in this section.
- Make sure you read up on what is being done and why.

Model	Loss/MSE
TensorFlow	0.0120
TensorFlow Lite	0.0120
TensorFlow Lite Quantized	0.0134

Model	Size	
TensorFlow	4096 bytes	
TensorFlow Lite	2788 bytes	(reduced by 1308 bytes)
TensorFlow Lite Quantized	2488 bytes	(reduced by 300 bytes)

Save your models!



Which ML model seems to best fit the MCU?
- TensorFlow
- TensorFlow Lite
- TensorFlow lite quantized

Continuing Education Center

Sponsored By

Thank you for attending

Please consider the resources below:

- www.beningo.com
  - Blog, White Papers, Courses
  - Embedded Bytes Newsletter
    - http://bit.ly/1BAHYXm

From www.beningo.com under

- Blog > CEC – Machine Learning Application Design using STM32 MCUs

Thank You

Sponsored by