

*****Default*****

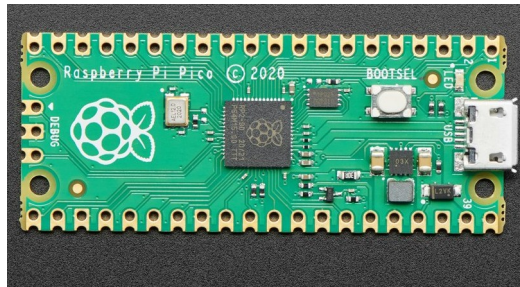
Raspberry Pi Pico RP2040 with TensorFlow Lite
Programming done on a Raspberry Pi4 and
Google Colaboratory
08/21/22

*****Default*****

Goals:

Goal 1.

To program a Raspberry Pi Pico RP2040 with TensorFlow Lite.



Additional information on process of compiling pico-tflmicro and testing can be found at

<https://github.com/develone/my-projects-docs/blob/master/pico/tensorflow.txt>

Steps to get the pico executable s

hello_world.elf, hello_world_test.elf & output_handler_test.elf

```
git clone git@github.com:develone/pico-tflmicro.git
```

```
cd pico-tflmicro
```

```
git clone git@github.com:develone/pico-sdk.git
```

```
cd pico-sdk/
```

```
git submodule update --init
```

```
cd ../
```

```
mkdir build
```

```
cd build
```

```
export PICO_SDK_PATH=./pico-sdk/
```

```
cmake -DPICO_BOARD=pico ..
```

```
make
```

Goal 2. To convert a TensorFlow model to a TensorFlow Lite model.

The command used to create model.cc was “xxd -i model.tflite > model.cc”

model.cc

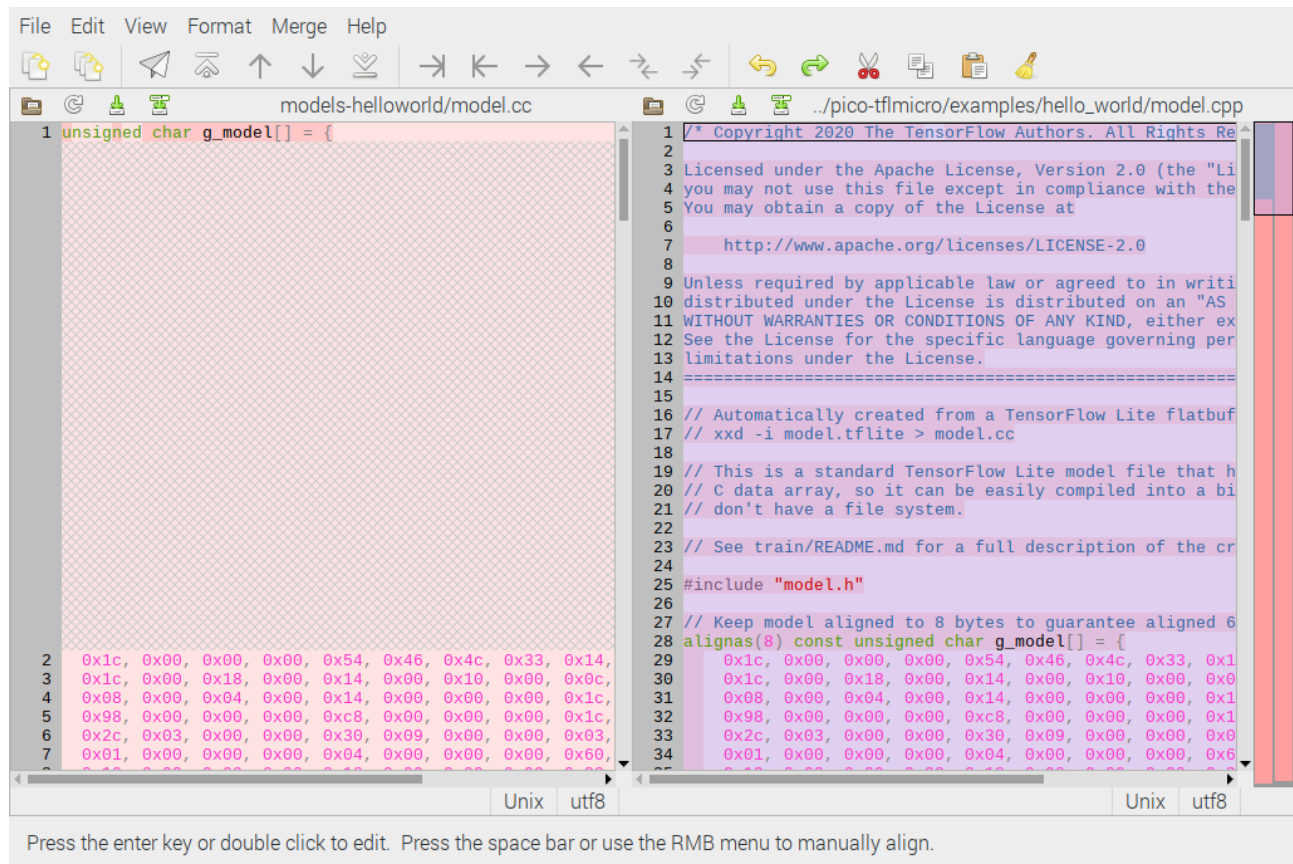
```
unsigned char g_model[] = {
```

model.cpp

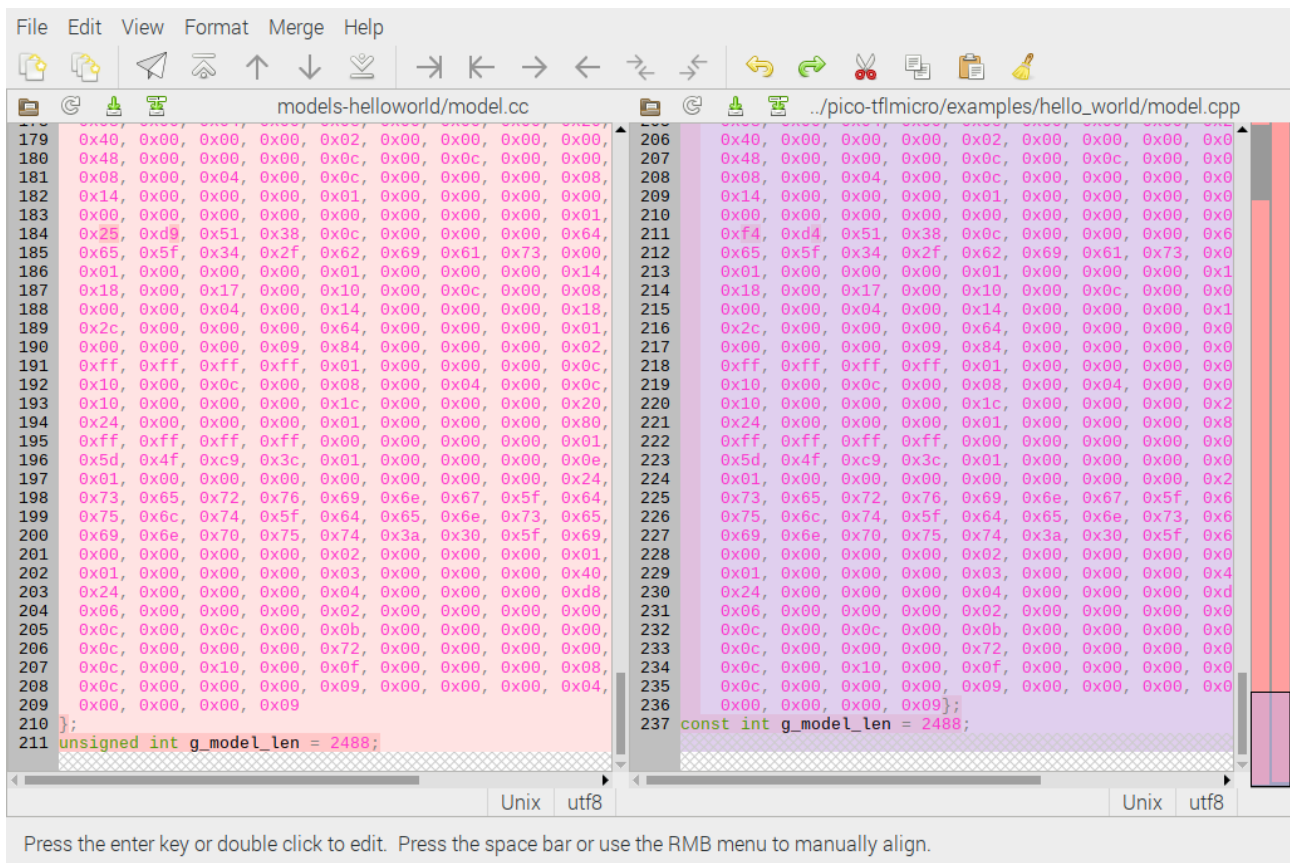
```
#include "model.h"
```

```
// Keep model aligned to 8 bytes to guarantee aligned 64-bit accesses.
```

```
alignas(8) const unsigned char g_model[] = {
```



Last of difference



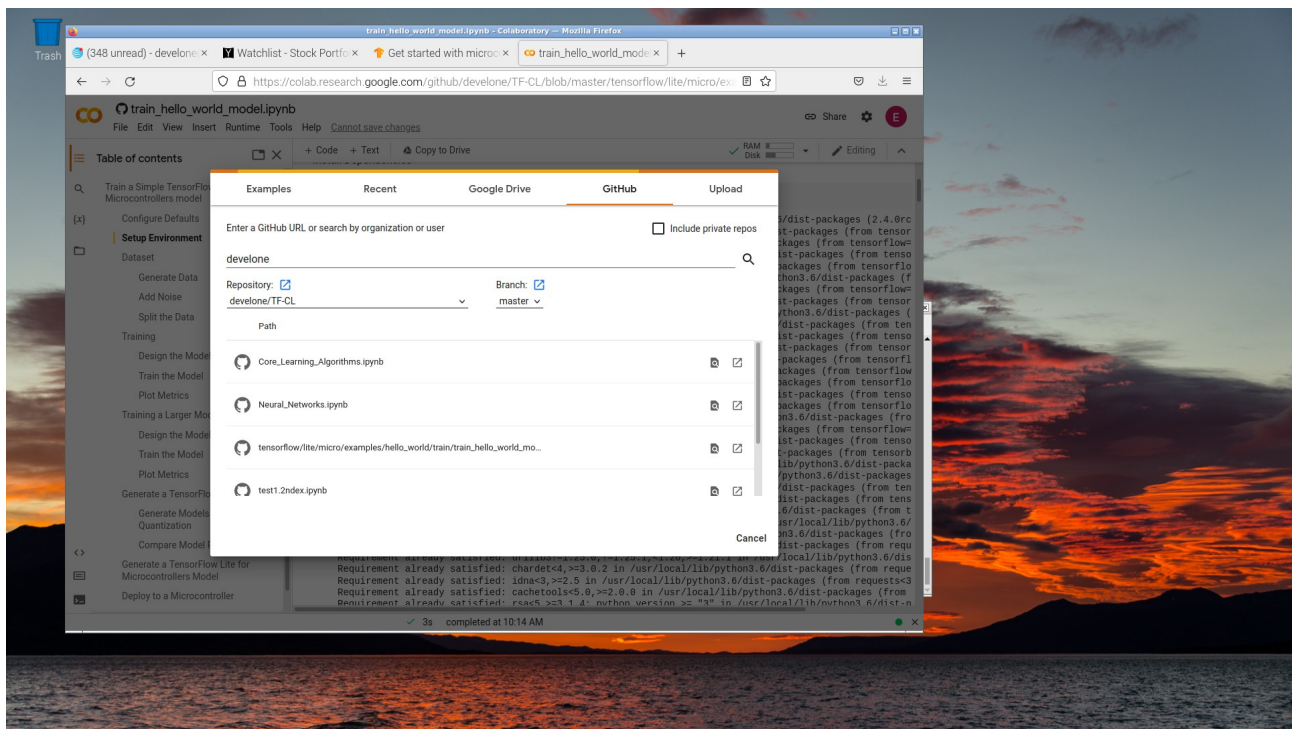
Testing using the xxd command to create the model.cc

```
devel@pi4-27:~/xx/TF-CL/models-helloworld $ xxd -i model.tflite > tt.cc
```

```
diff tt.cc model.cc
1c1
< unsigned char model_tflite[] = {
---
> unsigned char g_model[] = {
211c211
< unsigned int model_tflite_len = 2488;
---
> unsigned int g_model_len = 2488;
```

The model was saved to my github from

https://www.tensorflow.org/lite/microcontrollers/get_started_low_level



Loading the TensorFlow hello_world

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](#)

Configure Defaults

```
[ ] # 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'
```

Setup Environment

Install Dependencies

Setup Environment.

train_hello_world_model.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Table of contents

- Train a Simple TensorFlow Lite for Microcontrollers model
- Configure Defaults
- Setup Environment
- Dataset
- Training
- Training a Larger Model
- Generate a TensorFlow Lite Model
- Generate Models with or without Quantization
- Compare Model Performance
- Generate a TensorFlow Lite for Microcontrollers Model
- Deploy to a Microcontroller

Setup Environment

Install Dependencies

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

Building wheel for wrapt (setup.py) ... done
Created wheel for wrapt: filename=wrapt-1.12.1-cp37-cp37m-linux_x86_64.whl size=68716 sha256=7aa6d8...
Stored in directory: /root/.cache/pip/wheels/62/76/4c/aa25851149f3f6d9785f6c869387ad82b3fd37582fa81...
Successfully built wrapt
Installing collected packages: typing-extensions, numpy, grpcio, absl-py, wrapt, tensorflow-estimator
Attempting uninstall: typing-extensions
Found existing installation: typing-extensions 4.1.1
Uninstalling typing-extensions-4.1.1:
Successfully uninstalled typing-extensions-4.1.1
Attempting uninstall: numpy
Found existing installation: numpy 1.21.6
Uninstalling numpy-1.21.6:
Successfully uninstalled numpy-1.21.6
Attempting uninstall: grpcio
Found existing installation: grpcio 1.47.0
Uninstalling grpcio-1.47.0:
Successfully uninstalled grpcio-1.47.0
Attempting uninstall: absl-py
Found existing installation: absl-py 1.2.0
Uninstalling absl-py-1.2.0:
Successfully uninstalled absl-py-1.2.0
Attempting uninstall: wrapt
Found existing installation: wrapt 1.14.1
Uninstalling wrapt-1.14.1:
Successfully uninstalled wrapt-1.14.1
Attempting uninstall: tensorflow-estimator
Found existing installation: tensorflow-estimator 2.8.0
Uninstalling tensorflow-estimator-2.8.0:
Successfully uninstalled tensorflow-estimator-2.8.0

1m 27s completed at 10:21 AM

Import Dependencies

train_hello_world_model.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Table of contents

- Train a Simple TensorFlow Lite for Microcontrollers model
- Configure Defaults
- Setup Environment
- Dataset
- Training
- Training a Larger Model
- Generate a TensorFlow Lite Model
- Generate Models with or without Quantization
- Compare Model Performance
- Generate a TensorFlow Lite for Microcontrollers Model
- Deploy to a Microcontroller

Setup Environment

Import Dependencies

```
[3] # 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

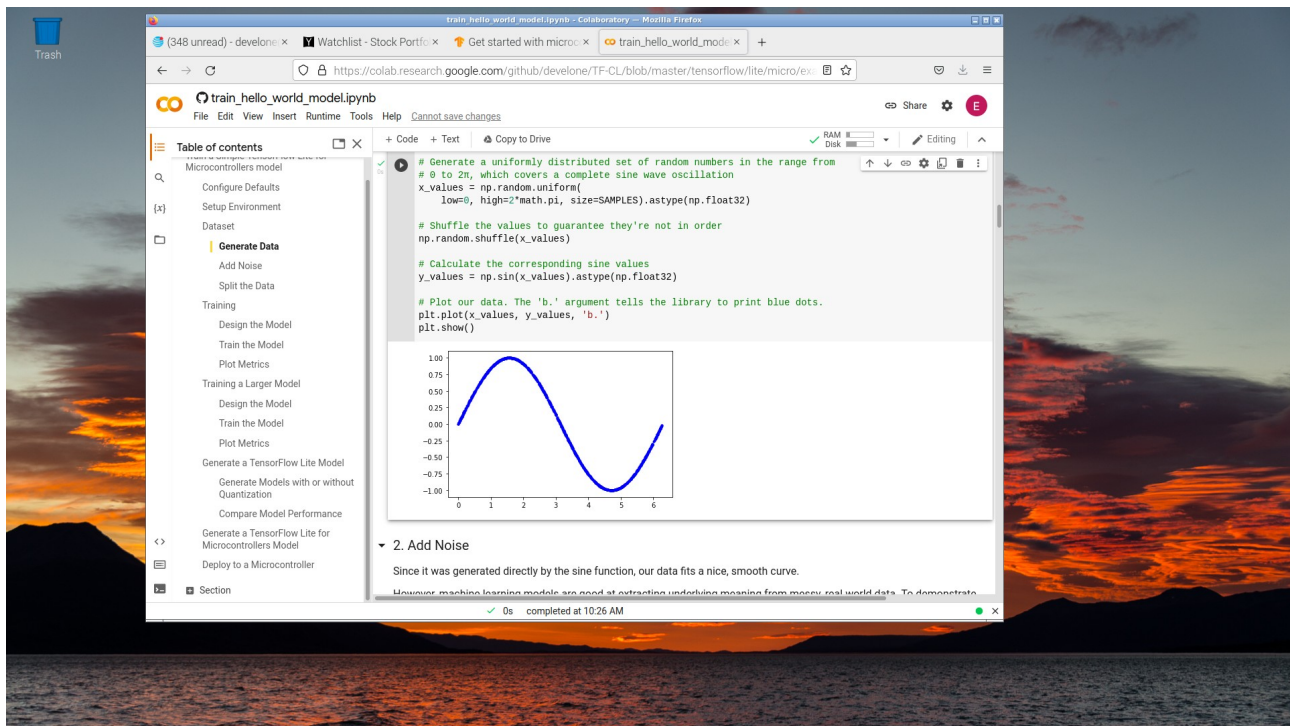
# Math is Python's math library
import math

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

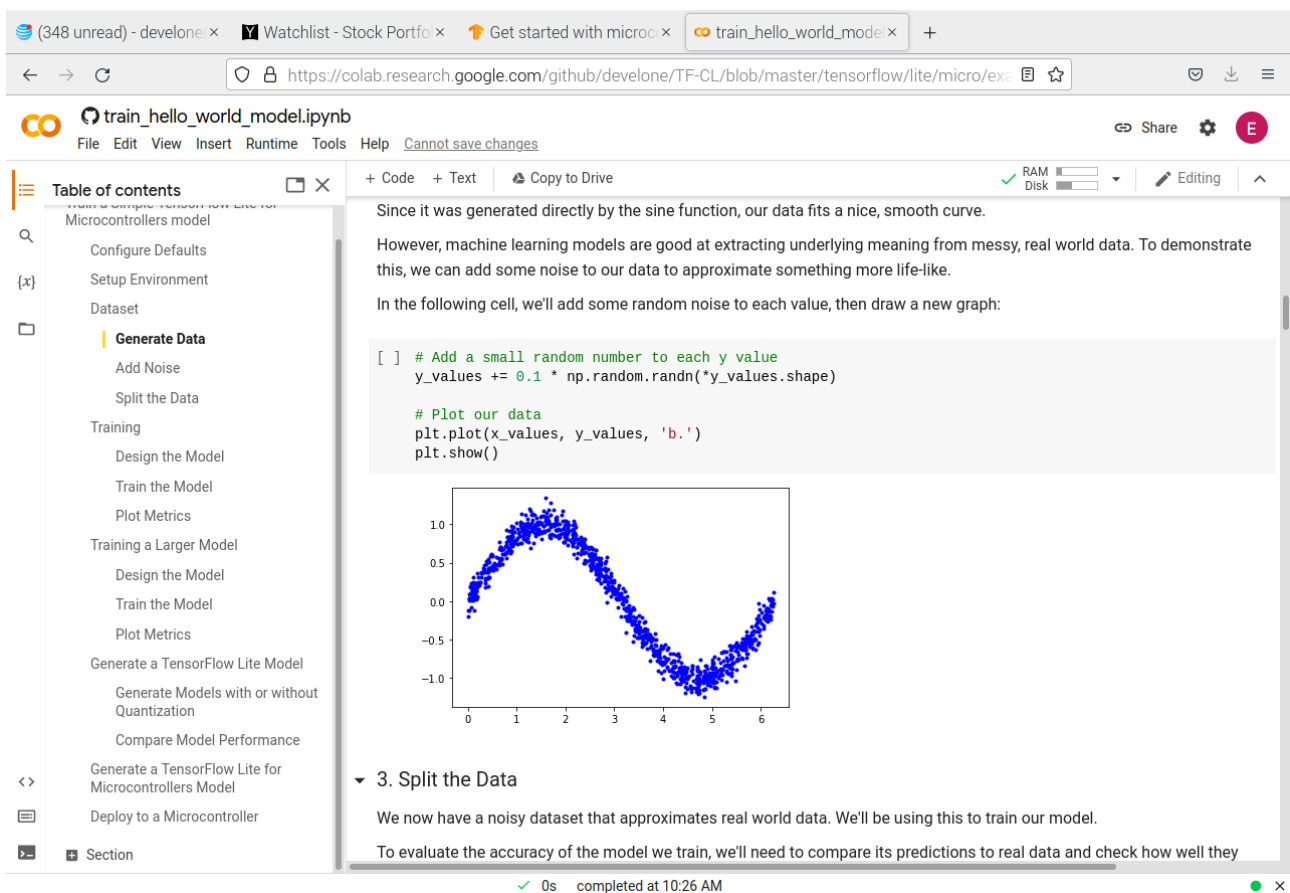
Dataset

1m 27s completed at 10:21 AM

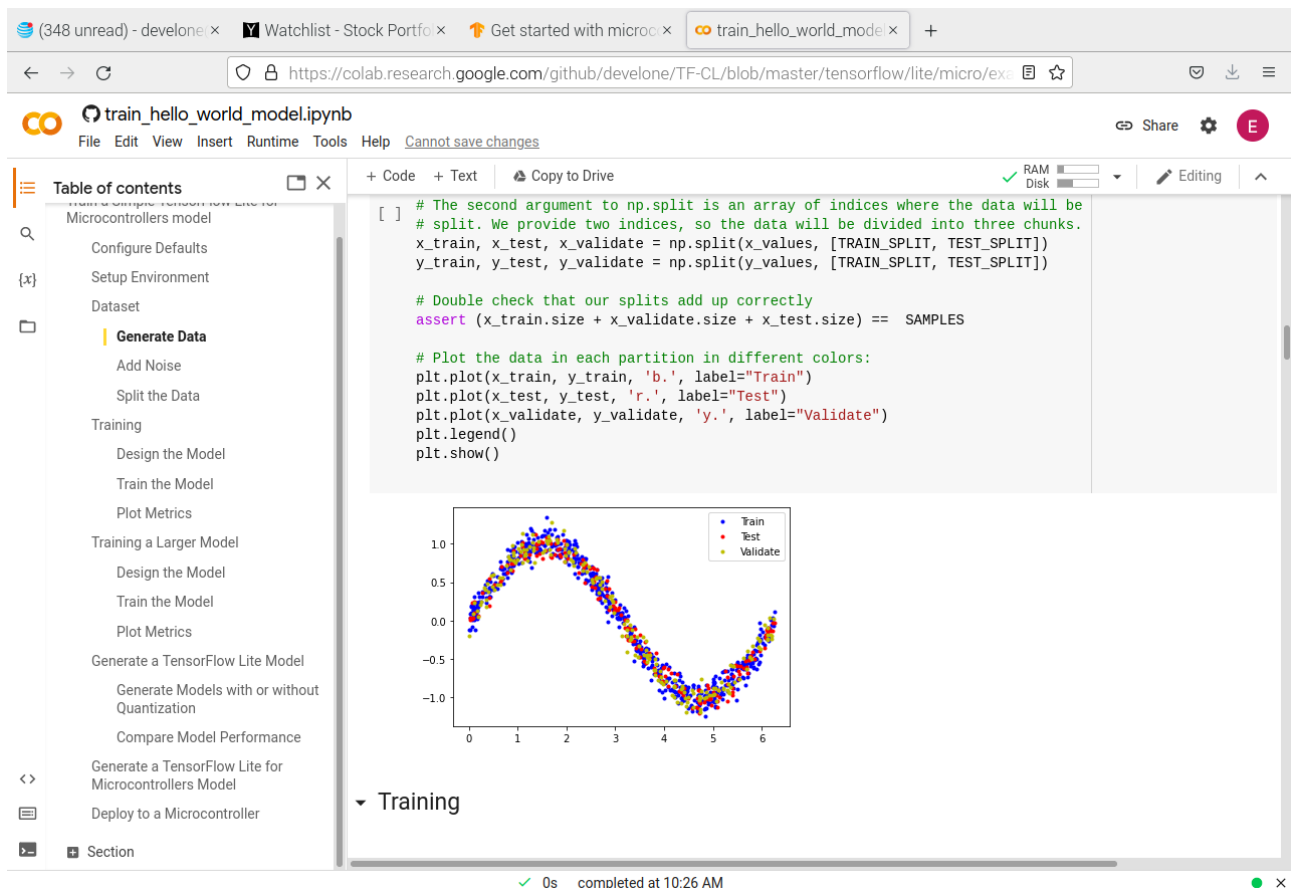
Generate Data



Add Noise



Split Data



Design the model

1. Design the Model

We're going to build a simple neural network model that will take an input value (in this case, `x`) and use it to predict a numeric output value (the sine of `x`). This type of problem is called a `_regression_`. It will use `_layers_` of `_neurons_` to attempt to learn any patterns underlying the training data, so it can make predictions.

To begin with, we'll define two layers. The first layer takes a single input (our `x` value) and runs it through 8 neurons. Based on this input, each neuron will become `_activated_` to a certain degree based on its internal state (its `_weight_` and `_bias_` values). A neuron's degree of activation is expressed as a number.

The activation numbers from our first layer will be fed as inputs to our second layer, which is a single neuron. It will apply its own weights and bias to these inputs and calculate its own activation, which will be output as our `y` value.

****Note:**** To learn more about how neural networks function, you can explore the [Learn TensorFlow](<https://codelabs.developers.google.com/codelabs/tensorflow-lab1-helloworld>) codelabs.

The code in the following cell defines our model using [Keras](<https://www.tensorflow.org/guide/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:

train_hello_world_model.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Table of contents

- Microcontrollers model
 - Configure Defaults
 - Setup Environment
 - Dataset
 - Generate Data
 - Add Noise
 - Split the Data
 - Training
 - Design the Model**
 - Train the Model
 - Plot Metrics
 - Training a Larger Model
 - Design the Model
 - Train the Model
 - Plot Metrics
 - Generate a TensorFlow Lite Model
 - Generate Models with or without Quantization
 - Compare Model Performance
 - Generate a TensorFlow Lite for Microcontrollers Model
 - Deploy to a Microcontroller

Section

Code Text Copy to Drive

RAM Disk

Editing

this input, each neuron will become *activated* to a certain degree based on its internal state (its *weight*). A neuron's degree of activation is expressed as a number.

The activation numbers from our first layer will be fed as inputs to our second layer, which is a single neuron. It will apply its own weights and bias to these inputs and calculate its own activation, which will be output as our *y* value.

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 func
model_1.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

2. Train the Model

Once we've defined the model, we can use our data to *train* it. Training involves passing an *x* value into the neural network, checking how far the network's output deviates from the expected *y* value, and adjusting the neurons' weights and biases so that the output is more likely to be correct the next time.

Training runs this process on the full dataset multiple times, and each full run-through is known as an *epoch*. The number of epochs to run during training is a parameter we can set.

0s completed at 10:26 AM

Train the model

train_hello_world_model.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Table of contents

- Microcontrollers model
 - Configure Defaults
 - Setup Environment
 - Dataset
 - Generate Data
 - Add Noise
 - Split the Data
 - Training
 - Design the Model
 - Train the Model**
 - Plot Metrics
 - Training a Larger Model
 - Design the Model
 - Train the Model
 - Plot Metrics
 - Generate a TensorFlow Lite Model
 - Generate Models with or without Quantization
 - Compare Model Performance
 - Generate a TensorFlow Lite for Microcontrollers Model
 - Deploy to a Microcontroller

Section

Code Text Copy to Drive

RAM Disk

Editing

2. Train the Model

Once we've defined the model, we can use our data to *train* it. Training involves passing an *x* value into the neural network, checking how far the network's output deviates from the expected *y* value, and adjusting the neurons' weights and biases so that the output is more likely to be correct the next time.

Training runs this process on the full dataset multiple times, and each full run-through is known as an *epoch*. The number of epochs to run during training is a parameter we can set.

During each epoch, data is run through the network in multiple *batches*. Each batch, several pieces of data are passed into the network, producing output values. These outputs' correctness is measured in aggregate and the network's weights and biases are adjusted accordingly, once per batch. The *batch size* is also a parameter we can set.

The code in the following cell uses the *x* and *y* values from our training data to train the model. It runs for 500 *epochs*, with 64 pieces of data in each *batch*. We also pass in some data for *validation*. As you will see when you run the cell, training can take a while to complete:

```
# 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))

10/10 [=====] - 0s 5ms/step - loss: 0.3458 - mae: 0.5042 - val_loss: 0.3492
Epoch 27/500
10/10 [=====] - 0s 5ms/step - loss: 0.3163 - mae: 0.4764 - val_loss: 0.3459
Epoch 28/500
10/10 [=====] - 0s 5ms/step - loss: 0.3441 - mae: 0.5018 - val_loss: 0.3427
Epoch 29/500
10/10 [=====] - 0s 5ms/step - loss: 0.3062 - mae: 0.4705 - val_loss: 0.3395
Epoch 30/500
10/10 [=====] - 0s 5ms/step - loss: 0.3202 - mae: 0.4808 - val_loss: 0.3362
Epoch 31/500
10/10 [=====] - 0s 5ms/step - loss: 0.3313 - mae: 0.4919 - val_loss: 0.3330
Epoch 32/500
10/10 [=====] - 0s 18ms/step - loss: 0.3028 - mae: 0.4682 - val_loss: 0.3297
```

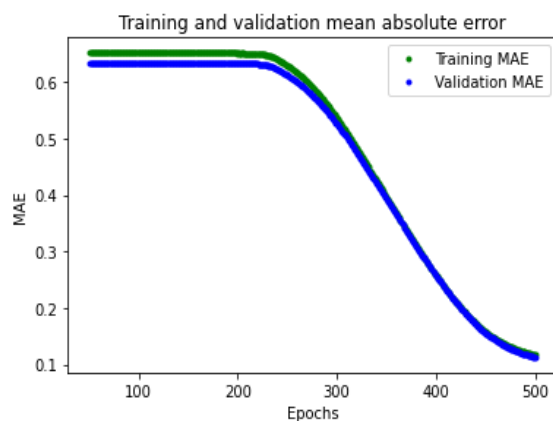
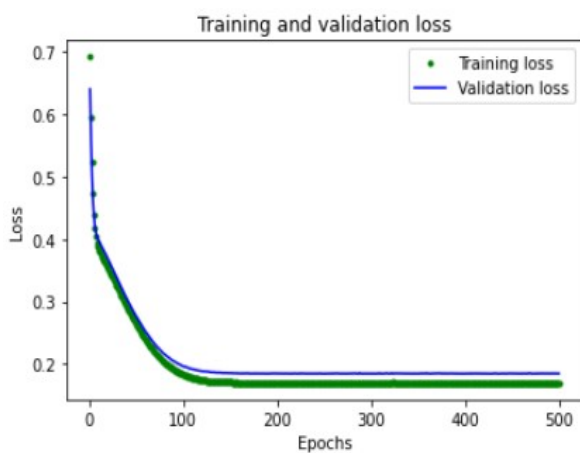
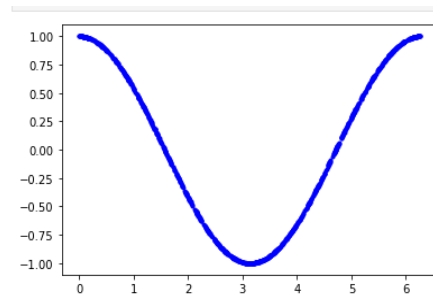
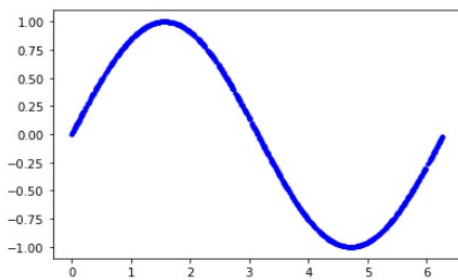
0s completed at 10:33 AM

Wanted to determine if making a change in TensorFlow at Google Corroboratory would result in a TensorFlow Lite run on Raspberry Pi Pico.

Changing the line `y_values = np.sin(x_values).astype(np.float32)` on left to `y_values = np.cos(x_values).astype(np.float32)` image on the right.

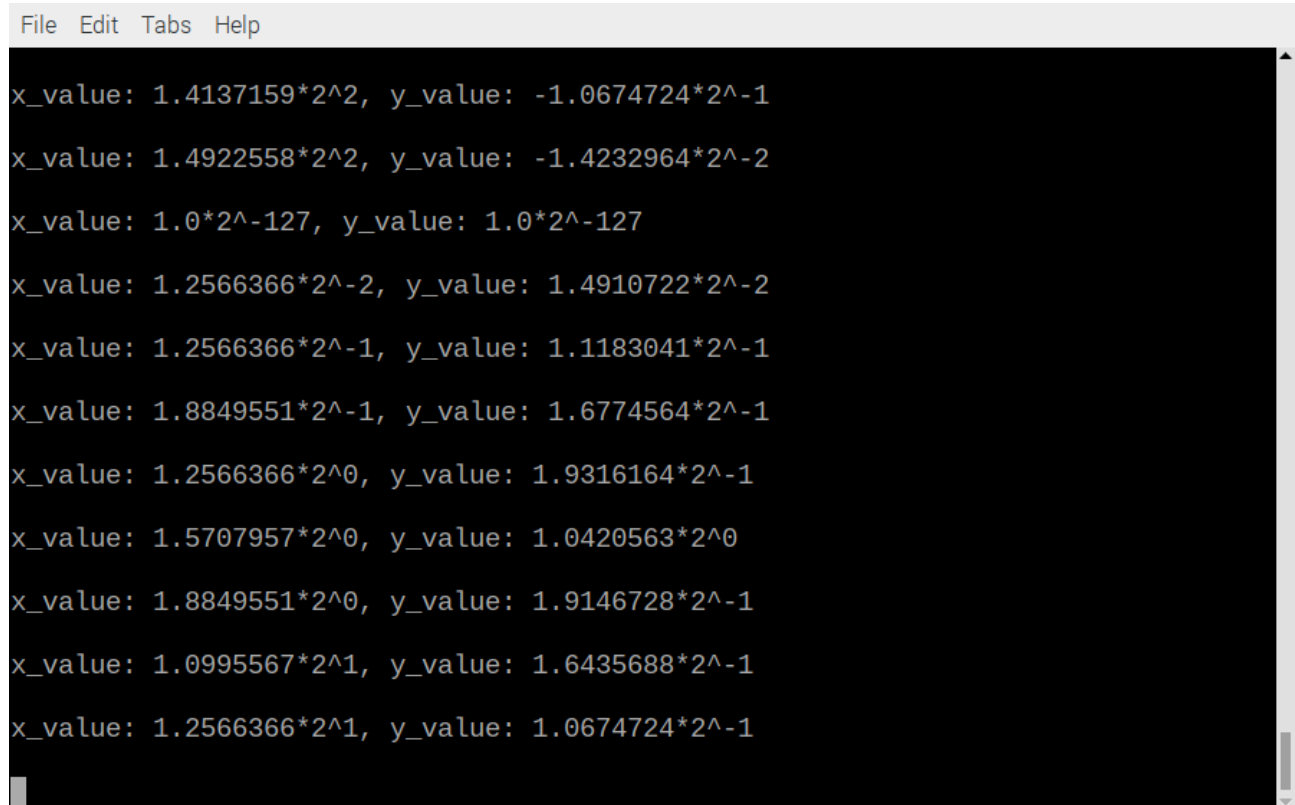
The original was saved in `model-sin.cpp`

This created a new `model.cpp` which was saved as `model-cos.cpp`



```
openocd -f interface/raspberrypi-swd.cfg -f target/rp2040.cfg -c "program
examples/hello_world/hello_world.elf verify reset exit"
```

```
minicom myusb0
```



A screenshot of a terminal window with a menu bar (File, Edit, Tabs, Help) and a dark background. It displays a list of 12 pairs of values, each consisting of an x_value and a y_value separated by a comma. The values are in scientific notation using powers of 2.

```
x_value: 1.4137159*2^2, y_value: -1.0674724*2^-1
x_value: 1.4922558*2^2, y_value: -1.4232964*2^-2
x_value: 1.0*2^-127, y_value: 1.0*2^-127
x_value: 1.2566366*2^-2, y_value: 1.4910722*2^-2
x_value: 1.2566366*2^-1, y_value: 1.1183041*2^-1
x_value: 1.8849551*2^-1, y_value: 1.6774564*2^-1
x_value: 1.2566366*2^0, y_value: 1.9316164*2^-1
x_value: 1.5707957*2^0, y_value: 1.0420563*2^0
x_value: 1.8849551*2^0, y_value: 1.9146728*2^-1
x_value: 1.0995567*2^1, y_value: 1.6435688*2^-1
x_value: 1.2566366*2^1, y_value: 1.0674724*2^-1
```

```
clear
```

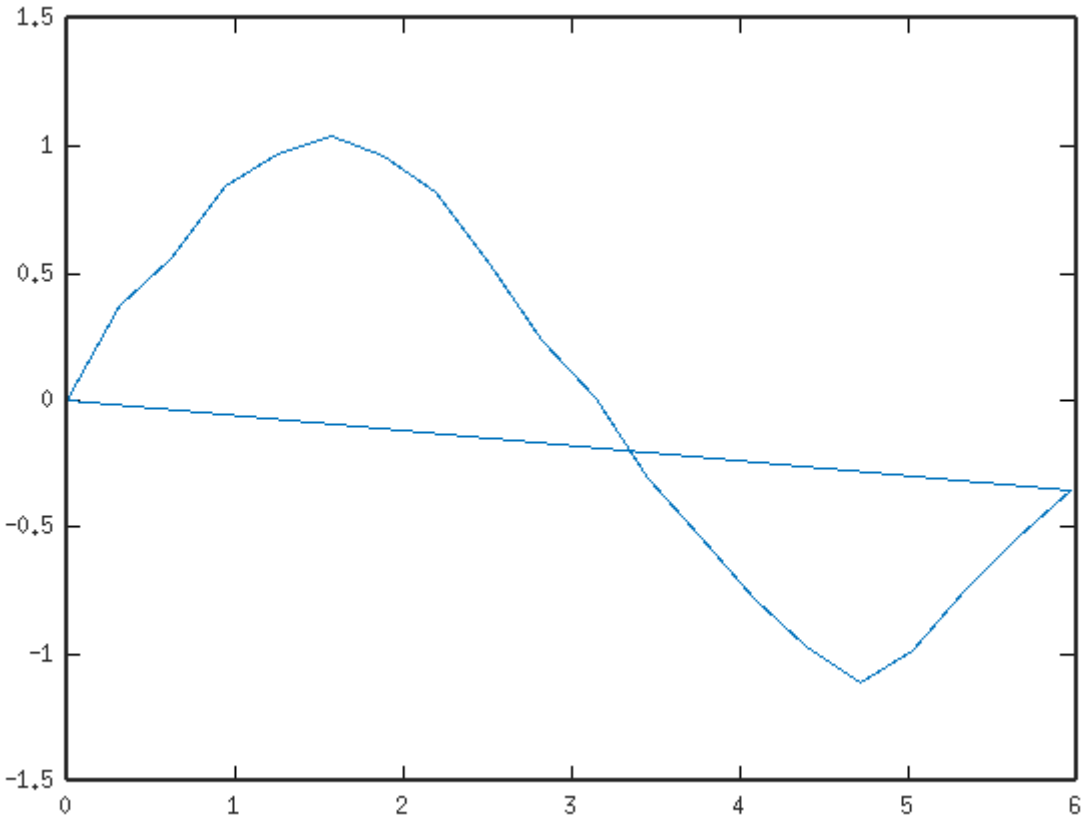
```
close all
```

```
x=[1.0*2^-127,1.2566366*2^-2,1.2566366*2^-1,1.8849551*2^-1,1.2566366*2^0,1.5707957*2^0,1.8849551*2^0, 1.0995567*2^1, 1.2566366*2^1,
1.4137159*2^1, 1.5707957*2^1, 1.7278753*2^1, 1.8849551*2^1, 1.0210171*2^2,
1.0995567*2^2, 1.1780966*2^2, 1.2566366*2^2, 1.3351763*2^2, 1.4137159*2^2,
1.4922558*2^2, 1.0*2^-127];
y=[1.0230449*2^0, 1.9795505*2^-1,1.5803134*2^-1, 1.197711*2^-1, 1.2309807*2^-2,
1.0646323*2^-5, -1.0646323*2^-2, -1.1311719*2^-1, -1.6634879*2^-1, -1.913011*2^-1, -
1.9961854*2^-1, -1.8631063*2^-1, -1.6135832*2^-1, -1.1145368*2^-1, -1.0646323*2^-2,
1.0*2^-127, 1.2975204*2^-2, 1.2143458*2^-1, 1.5969484*2^-1, 1.913011*2^-1,
1.0230449*2^0];

yy=[ 1.0*2^-127, 1.4910722*2^-2, 1.1183041*2^-1, 1.6774564*2^-1, 1.9316164*2^-1,
1.0420563*2^0, 1.9146728*2^-1, 1.6435688*2^-1, 1.0674724*2^-1, 1.8977287*2^-3,
1.0844163*2^-7, -1.2199684*2^-2, -1.0674724*2^-1, -1.5588485*2^-1, -1.9316164*2^-1, -
1.1098324*2^0, -1.9655047*2^-1, -1.4910722*2^-1, -1.0674724*2^-1, -1.4232964*2^-2,
1.0*2^-127];
figure
```

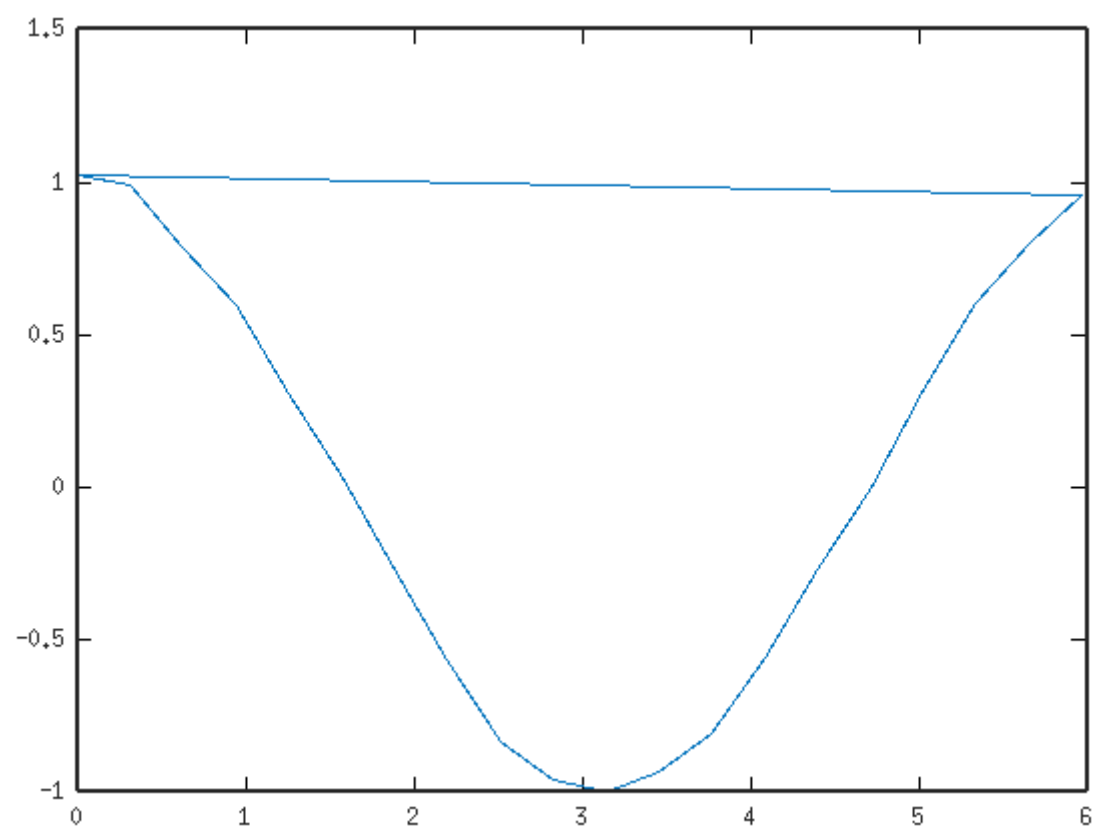
```
plot(x,yy)
```

```
figure  
plot(x,y)
```



5.65663, 0.267050

```
sin
```



6.05045, 0.0256640

XX
COS