IoT Machine Learning

How to train a machine learning model?

By Anzhelika Kurnikova

IEEE Computer Society – Student Branch Chapter
University of South Florida
Tampa, FL, USA

21 November 2023

IEEE COMPUTER SOCIETY
Student Branch Chapter at the University of South Florida

UNIVERSITY of SOUTH FLORIDA
College of Engineering

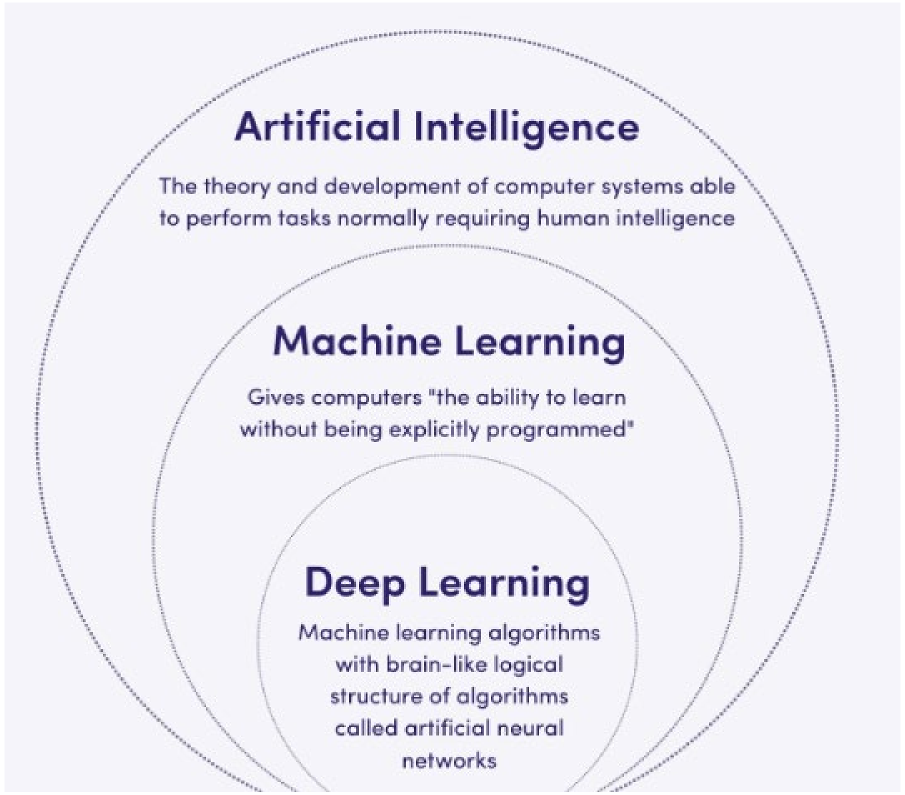
Agenda

Disclaimer: Machine learning and IoT is hard. I tried to simplify it.

- Overview of Machine Learning
- Overview of how to make a ML model using TensorFlow
- Coding
- Overview of IoT
- Setting up a Particle board (you get to keep it)
- Connecting ML model to our board

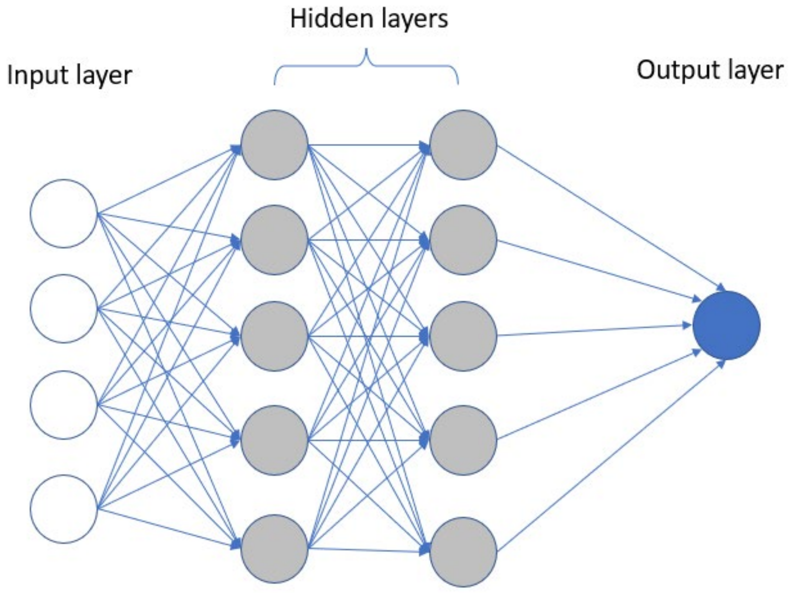
Machine Learning

What is AI and Machine learning?


Artificial Intelligence
The theory and development of computer systems able to perform tasks normally requiring human intelligence
Machine Learning
Gives computers "the ability to learn without being explicitly programmed"
Deep Learning
Machine learning algorithms with brain-like logical structure of algorithms called artificial neural networks

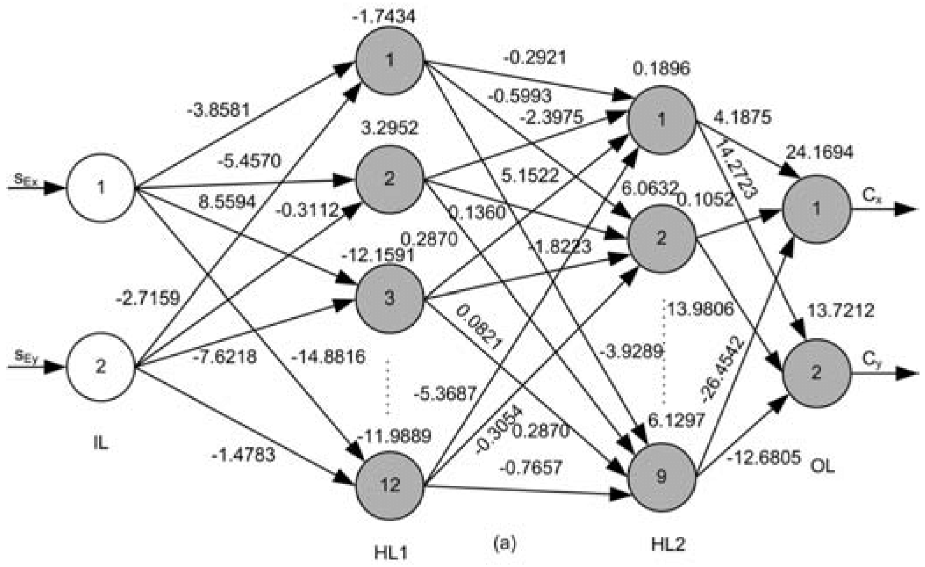
Neural Network

- Interconnected nodes called neurons that form layers
- Hidden layers use math to process information
- Each neuron:
  - gets data from other on
  - processes it using math (weighted function)
  - sends it to next layer



What is training of a neural network?

- Each node has its math function
- Training self-adjusts the weights in the math function to increase accuracy (programmer does not adjust the weights)



Advantages of neural networks over other machine learning models?

Neural Networks are more mathematically complex:

- Can understand complex patterns
- Scalable
- Can handle various types of data like pictures
- Better adapt to new data

How to make a machine learning model using TensorFlow

What is TensorFlow?

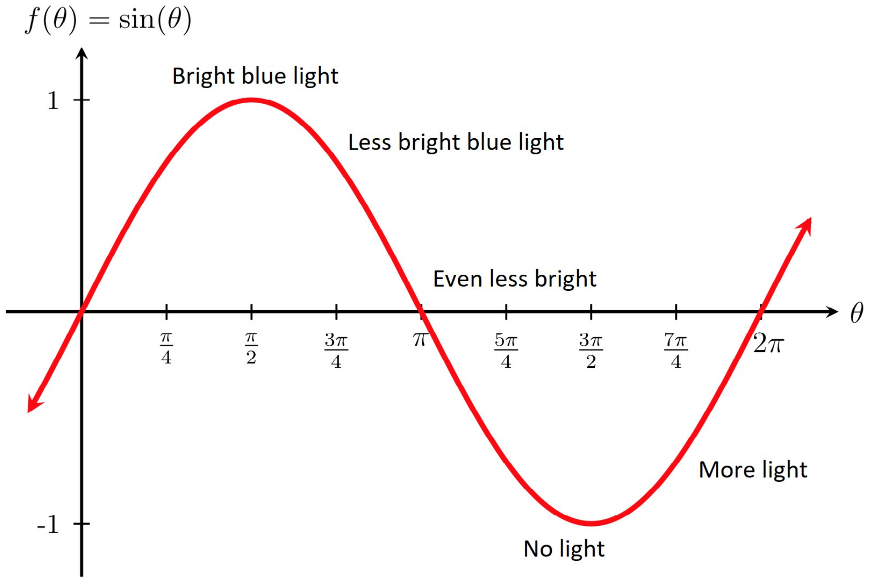
- A library for Python and C++ for machine learning and AI
- Developed by Google
- Focuses on training of deep learning neural networks

Why do we need it?

- inefficient to make ML model from scratch => take it from TensorFlow, PyTorch or scikit-learn
- similar to a character in computer game that you can customize

What we will make?

- A Machine Learning model using TensorFlow that given x, predicts the value of sin(x). Emphasis is on predicts, we will NOT do y = sin(x) in Python code. Model will itself come up with sin function.
- Set up a Particle Argon board to flash blue light with a brightness that follows a predicted function

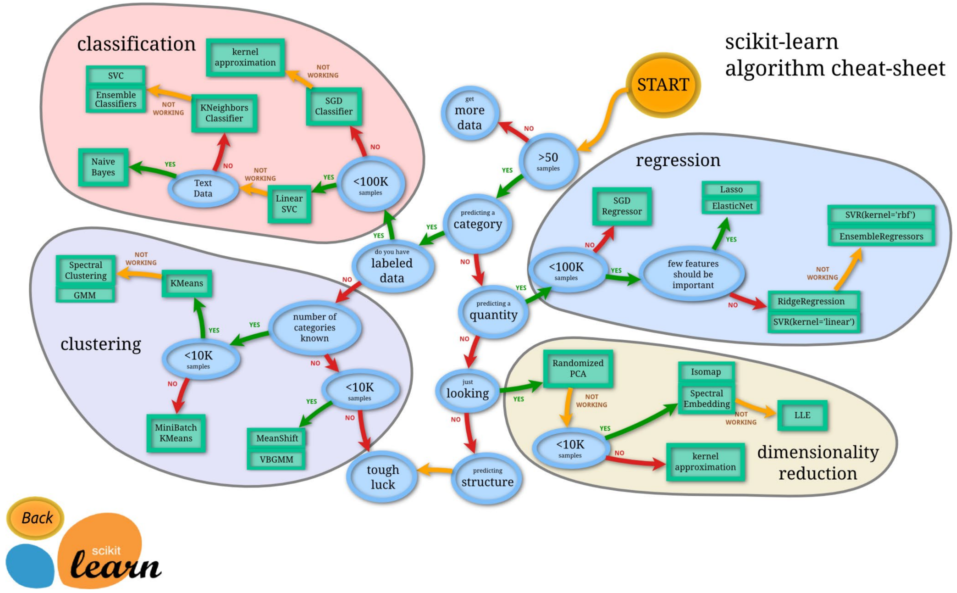


#1 Understand a problem you need to solve

- For each type of problem there is a certain model that works the best

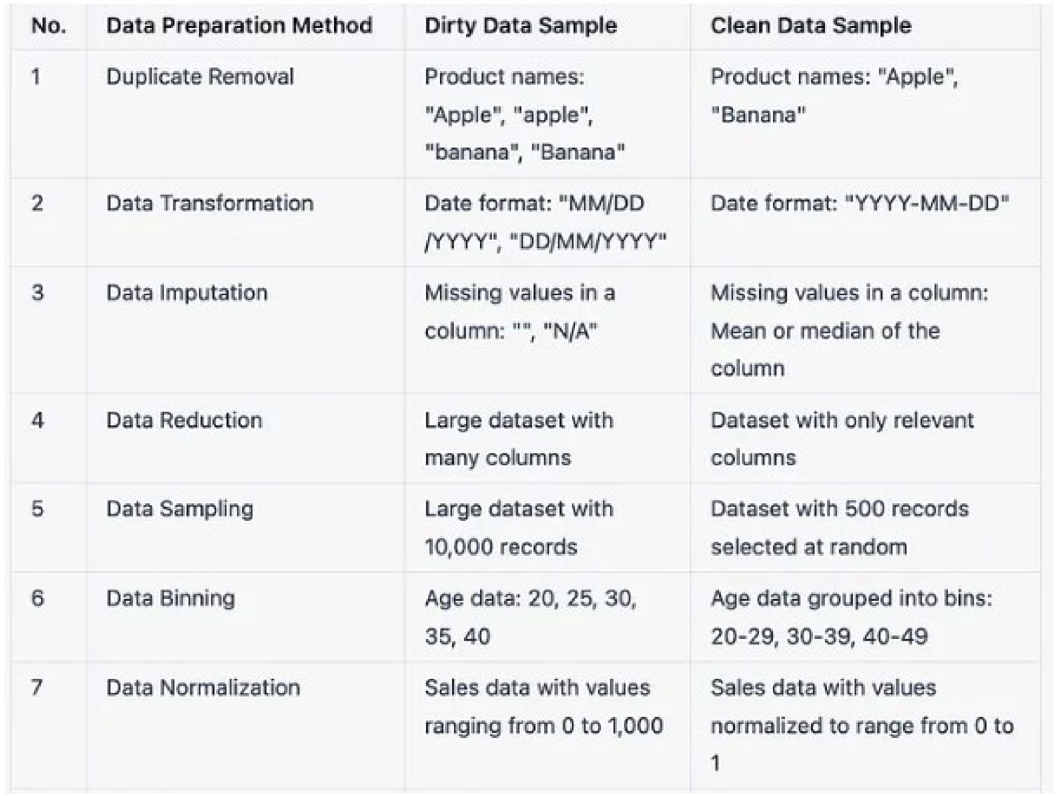
- Thought process is like in a computer game in which you decide what kind of character to choose (a mage or a knight or a gnome and etc.)

- All ML models have its strengths and weaknesses



#2 Collect and preprocess data

- To train (learn patterns) a model needs data
- Most cases data looks like a table
- Principle “Garbage in -> garbage out”
- Therefore, you need to clean data and handle missing value
- Entire profession – data engineer



No.	Data Preparation Method	Dirty Data Sample	Clean Data Sample
1	Duplicate Removal	Product names: "Apple", "apple", "banana", "Banana"	Product names: "Apple", "Banana"
2	Data Transformation	Date format: "MM/DD /YYYY", "DD/MM/YYYY"	Date format: "YYYY-MM-DD"
3	Data Imputation	Missing values in a column: "", "N/A"	Missing values in a column: Mean or median of the column
4	Data Reduction	Large dataset with many columns	Dataset with only relevant columns
5	Data Sampling	Large dataset with 10,000 records	Dataset with 500 records selected at random
6	Data Binning	Age data: 20, 25, 30, 35, 40	Age data grouped into bins: 20-29, 30-39, 40-49
7	Data Normalization	Sales data with values ranging from 0 to 1,000	Sales data with values normalized to range from 0 to 1

#3 Collect and preprocess data

Examples:

- Removing noise -> deleting unnecessary data
- Handling missing values -> delete the row or put there a mean/median of the column
- Dealing with Outliers -> outliers skew results, so delete or replace
- Normalizing data -> making 1-100 range into 0-1
- Feature engineering -> combine several columns into one. For example, width and length into a column area
- Ensuring consistency -> units, formats and etc.
- Handle duplicates

#3 Collect and preprocess data

Where to get data:

- If it is numbers, you can randomly generate them (like we will do today)
- Download a dataset from the internet
- If you work for real company, ask the company. For example, their clients data.

Data needs to get randomly split to:

- Training data
- Validation data
- Test data

In our case data is just randomly generated x and sin(x) values, so we do not need to preprocess them.

#4 Choose model architecture

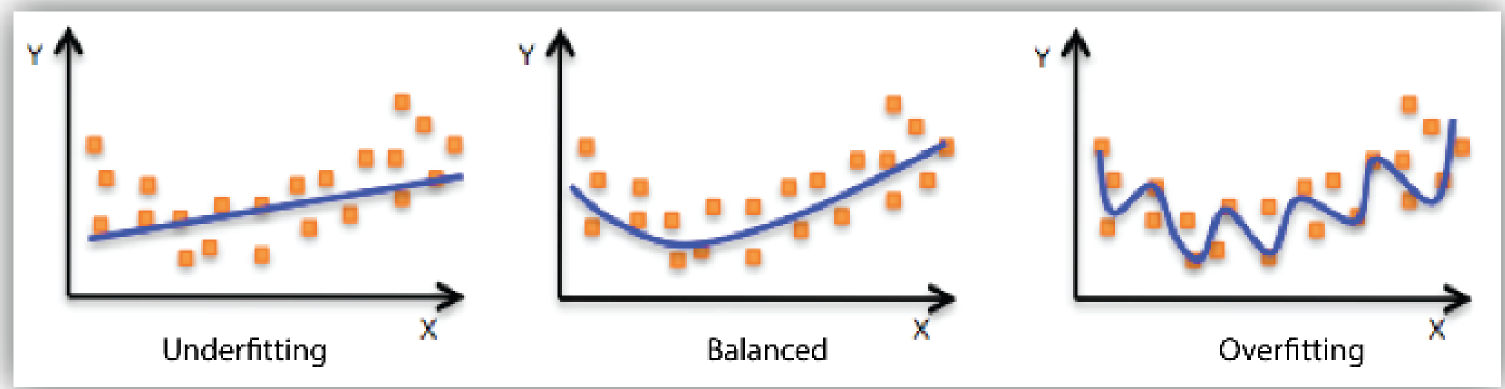
- Choose:
  - A type of model
  - Number of layers
    - More layers pluses:
      - Can potentially learn more complex patterns
      - Can reuse learned patterns
    - More layers minuses:
      - Your computer hardware needs to be strong enough
      - Neural network can overfit. Overfitting is when model adapts to training data so much that it performs really bad on the test data
  - Activation function – determines how output number of a neuron will look like
  - Connect the layers – set their order, which layer is first, which is second and etc.

#5 Compile the model

- Configure metrics (math functions) that measures how well the model performed
  - Loss function
  - Accuracy
  - Precision and Recall
  - F1 Score
  - MSE
  - R^2
  - And more…
- Configure optimizer that helps minimize the loss function = improve the model
  - Stochastic Gradient Descent (SGD)
  - Adaptive Moment Estimation (Adam)
  - And more…

#7 Train the model

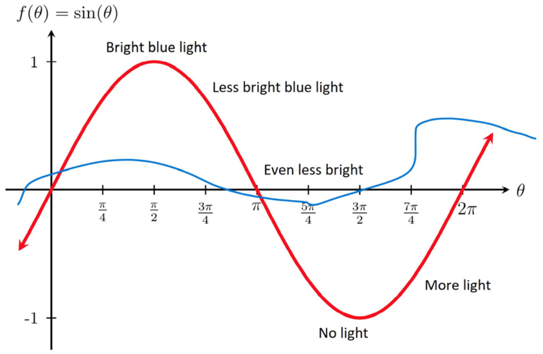
- Give the model data
- In a loop: model runs and adjusts weights in its neurons, trying to improve itself (compares the results with the output results a programmer gave)

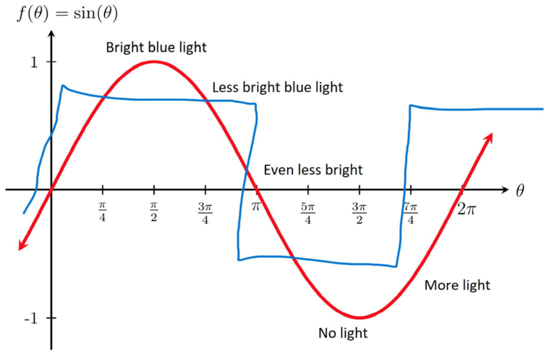


#7 Training in our example (sin function)

- In our case model could train something like that:
- Epoch – is when all training data gets passed through the neural network

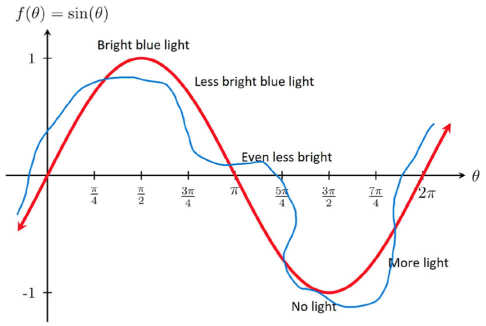
Epoch 1

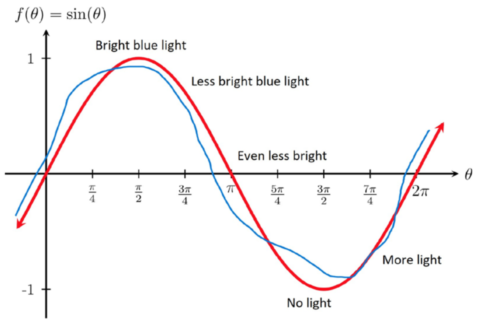




Epoch 2

Epoch 3





Epoch 4

#7 Evaluating the model

- Use the validation data (just part of your data that you took for validation) to evaluate the trained model
- If the model is performing bad, try:
  - Changing the models architecture
  - Changing the model compilation configuration
  - Take another type of model
  - Improve training data
  - Give up
- Do the step #7 until model improves
- Model is ready, use it

Takeaways about making a machine learning model

- Data is probably the most important part. If data is bad, no matter what you do machine learning model is bad.
- Machine learning model is like a computer game character, you get some general character and you customize it to a particular situation
- To be successful in machine learning, you need to know how to analyze and clean data, and know math like statistics, linear algebra and rarely calculus
- In our project today, we will use default optimizer, metric and etc, will not have to clean data, or to evaluate the model because our task is really simple

Coding

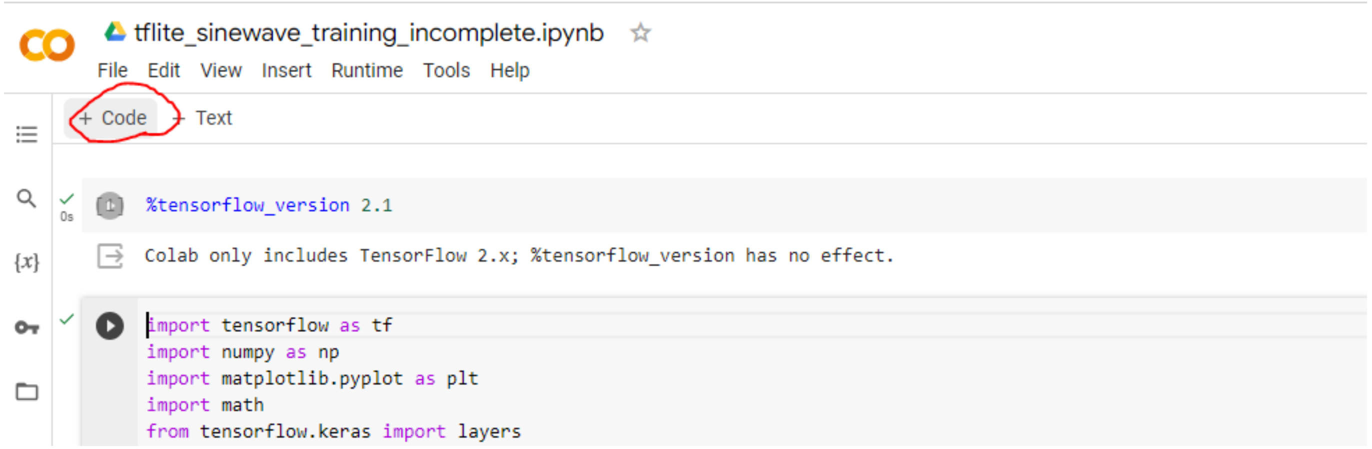
Set up Google Collab

https://colab.research.google.com/drive/1Km2gARpAb54Q3ZOd8CWrGtz3uzsH4IHh?usp=sharing

- Press “File”
- Press “Save a copy in drive”
- Now you can code with us (the file already has some premade code)

What is Google Collab?

- Google Collab is a platform for coding in Python for machine learning using Jupyter Notebook
- Jupyter notebook – a file that allows you to code, immediately visualize plots, and text. It is convenient for ML research.
- To make a code cell (gray rectangle) click Code button
- To run code in a cell, press Shift + Enter

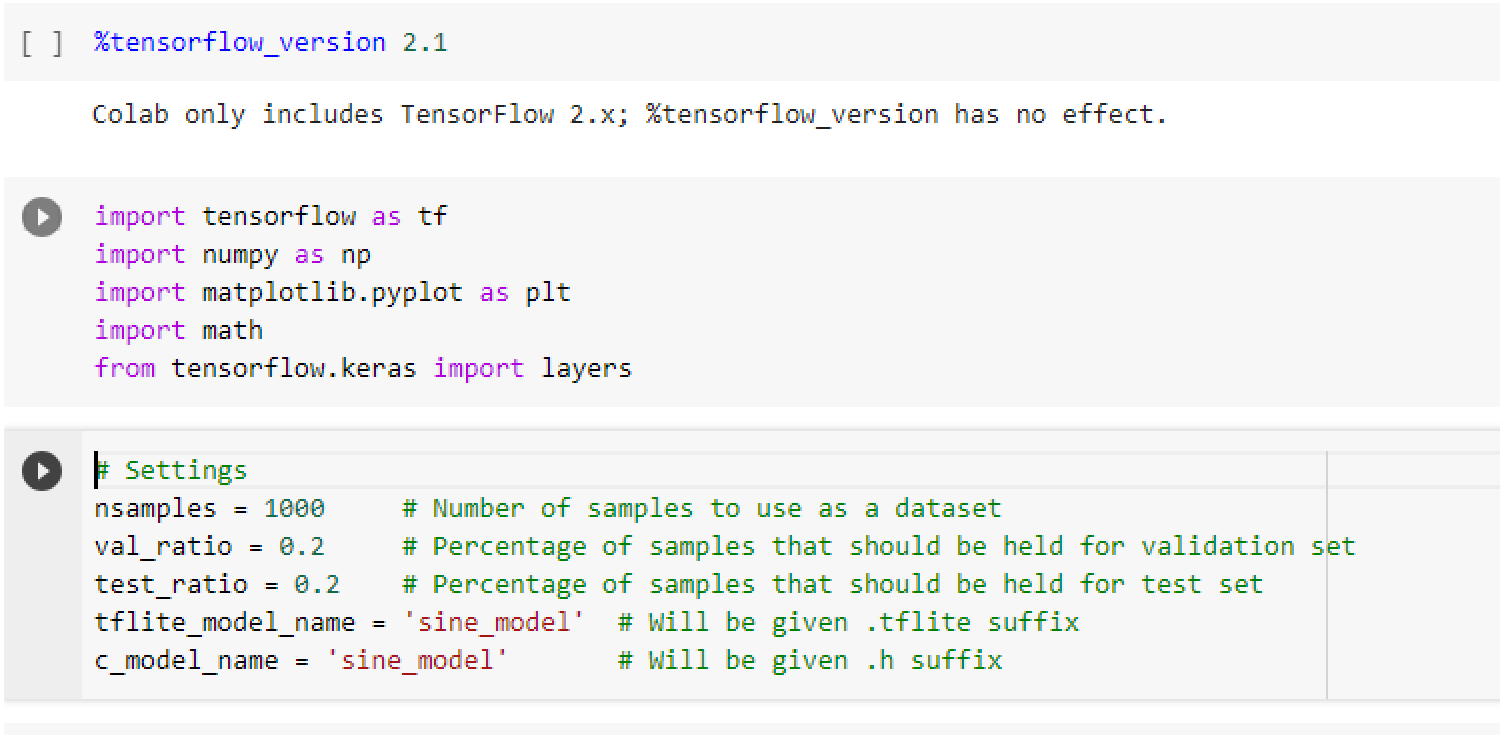

tflite_sinewave_training_incomplete.ipynb
File Edit View Insert Runtime Tools Help
+ Code
+ Text
0s

```
%tensorflow_version 2.1
```

```
Colab only includes TensorFlow 2.x; %tensorflow_version has no effect.
```

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import math
from tensorflow.keras import layers
```

Code part 1. Imports and settings. We premade.



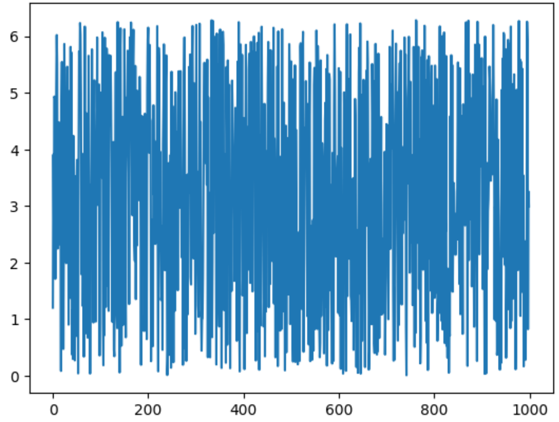
```
[ ] %tensorflow_version 2.1

    Colab only includes TensorFlow 2.x; %tensorflow_version has no effect.
```

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import math
from tensorflow.keras import layers
```

```
# Settings
nsamples = 1000     # Number of samples to use as a dataset
val_ratio = 0.2     # Percentage of samples that should be held for validation set
test_ratio = 0.2    # Percentage of samples that should be held for test set
tflite_model_name = 'sine_model'  # Will be given .tflite suffix
c_model_name = 'sine_model'       # Will be given .h suffix
```

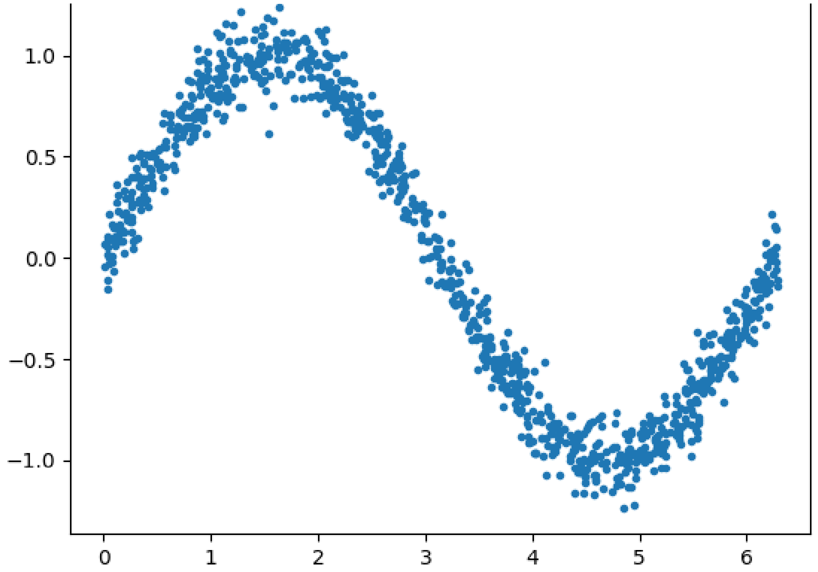
Code part 2. Generate input data to feed the model

```
# Generate some random samples
np.random.seed(1234)
x_values = np.random.uniform(low=0, high=(2 * math.pi), size=nsamples)
plt.plot(x_values)
```

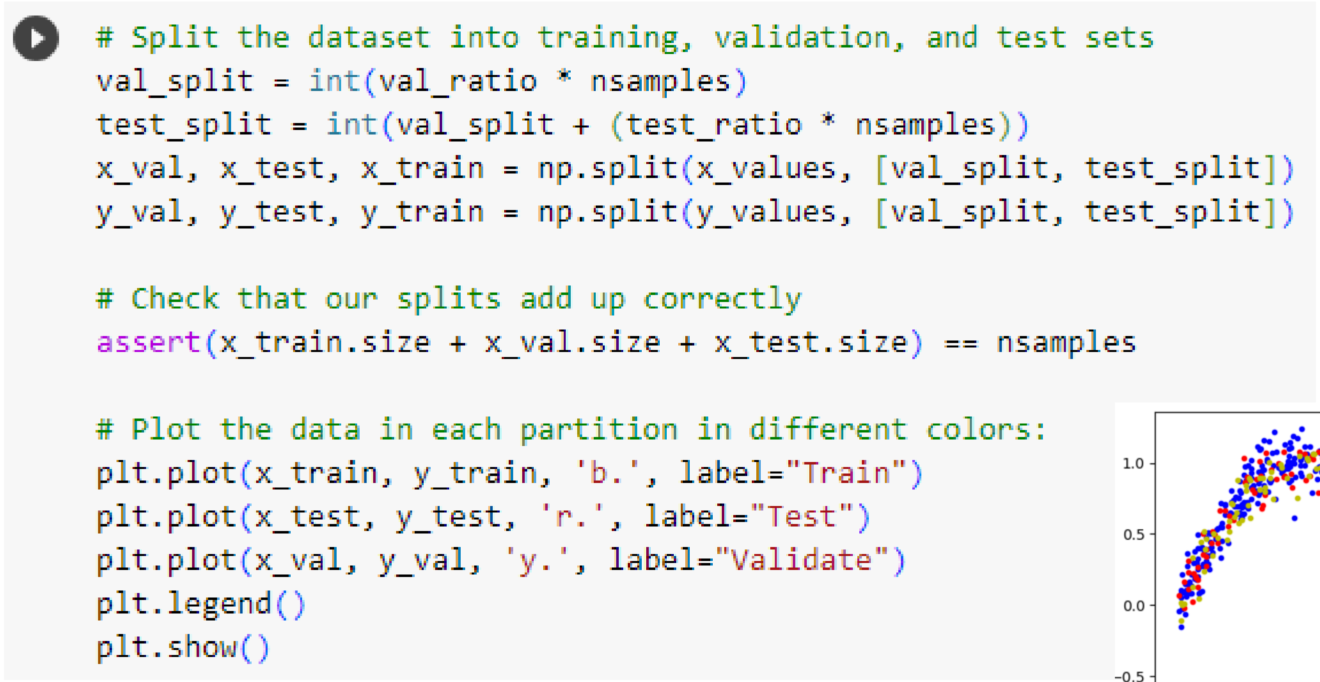


Code part 3. Generate output data, so model can compare to it

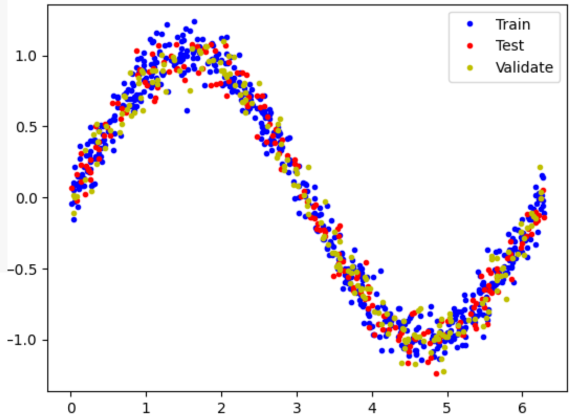
```
# Create a noisy sinewave with these values
y_values = np.sin(x_values) + (0.1 * np.random.randn(x_values.shape[0]))
plt.plot(x_values, y_values, '.')
```



Code part 4. Split the data into training, validation and test data



```
# Split the dataset into training, validation, and test sets
val_split = int(val_ratio * nsamples)
test_split = int(val_split + (test_ratio * nsamples))
x_val, x_test, x_train = np.split(x_values, [val_split, test_split])
y_val, y_test, y_train = np.split(y_values, [val_split, test_split])

# Check that our splits add up correctly
assert(x_train.size + x_val.size + x_test.size) == nsamples

# 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_val, y_val, 'y.', label="Validate")
plt.legend()
plt.show()
```



Code part 5. Architect the model.

```
# Create a model
model = tf.keras.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(1,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1))
```

```
# View model
model.summary()
```

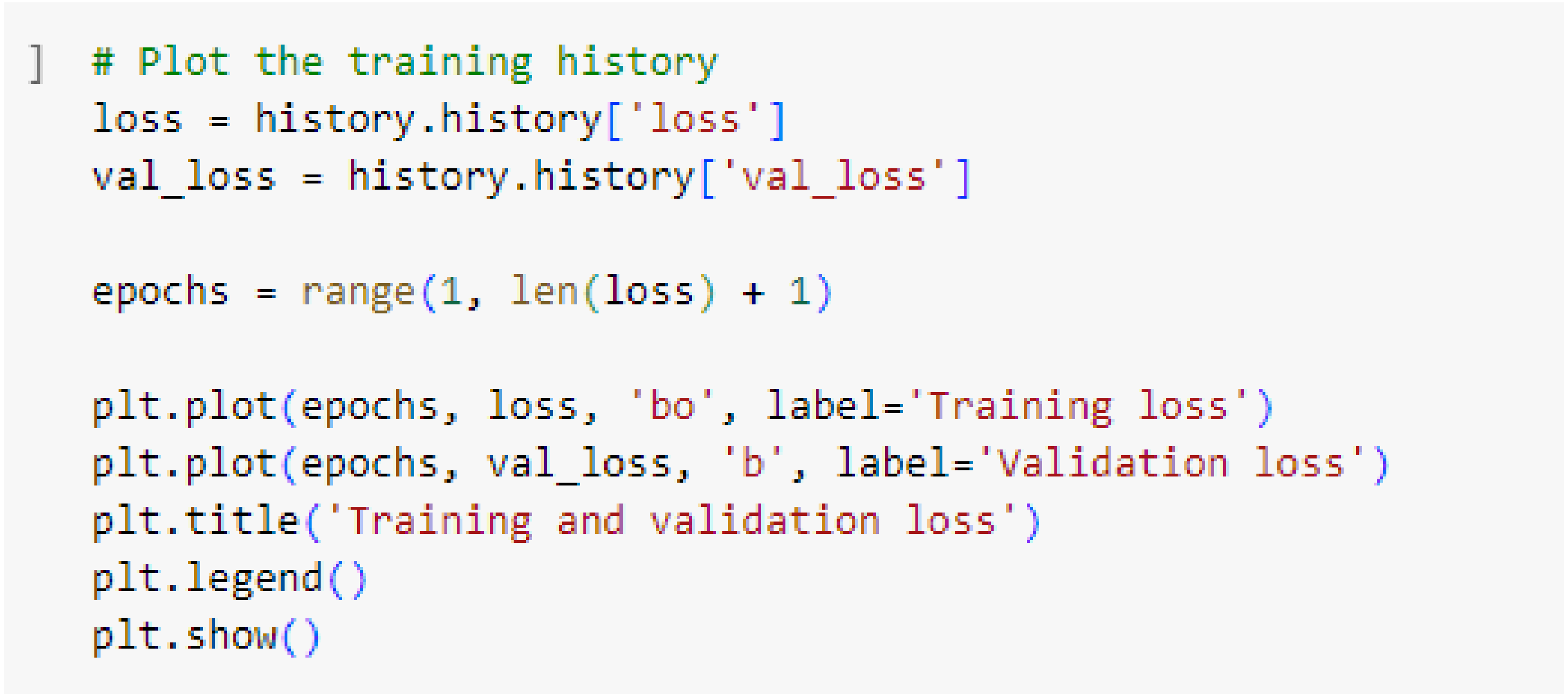
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                32

 dense_1 (Dense)             (None, 16)                272

 dense_2 (Dense)             (None, 1)                 17

=================================================================
Total params: 321 (1.25 KB)
Trainable params: 321 (1.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Code part 6. Compile the model and train it.

```
# Add optimizer, loss function, and metrics to model and compile it
model.compile(optimizer='rmsprop', loss='mae', metrics=['mae'])
```

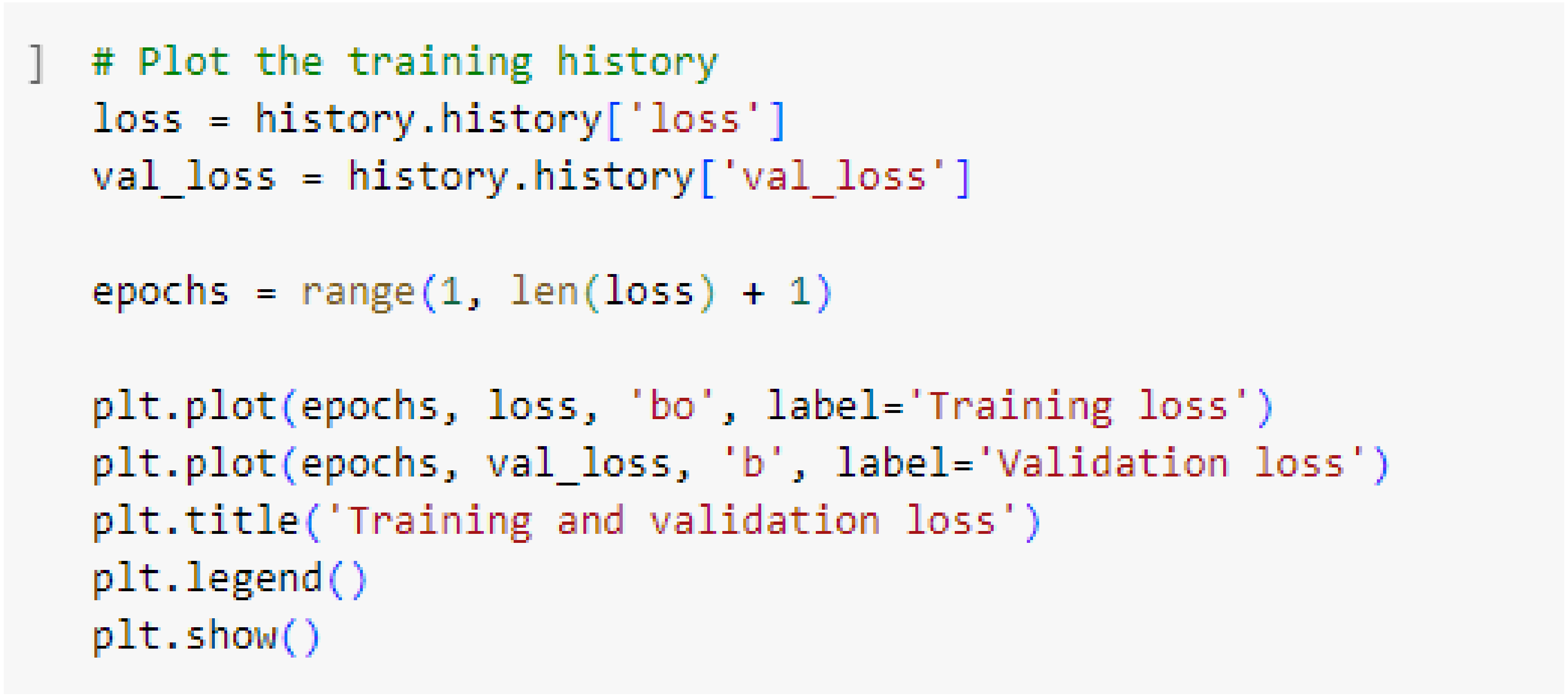
```
# Train model
history = model.fit(x_train,
                    y_train,
                    epochs=500,
                    batch_size=100,
                    validation_data=(x_val, y_val))
```

Not required code we premade for you (don't code it now)



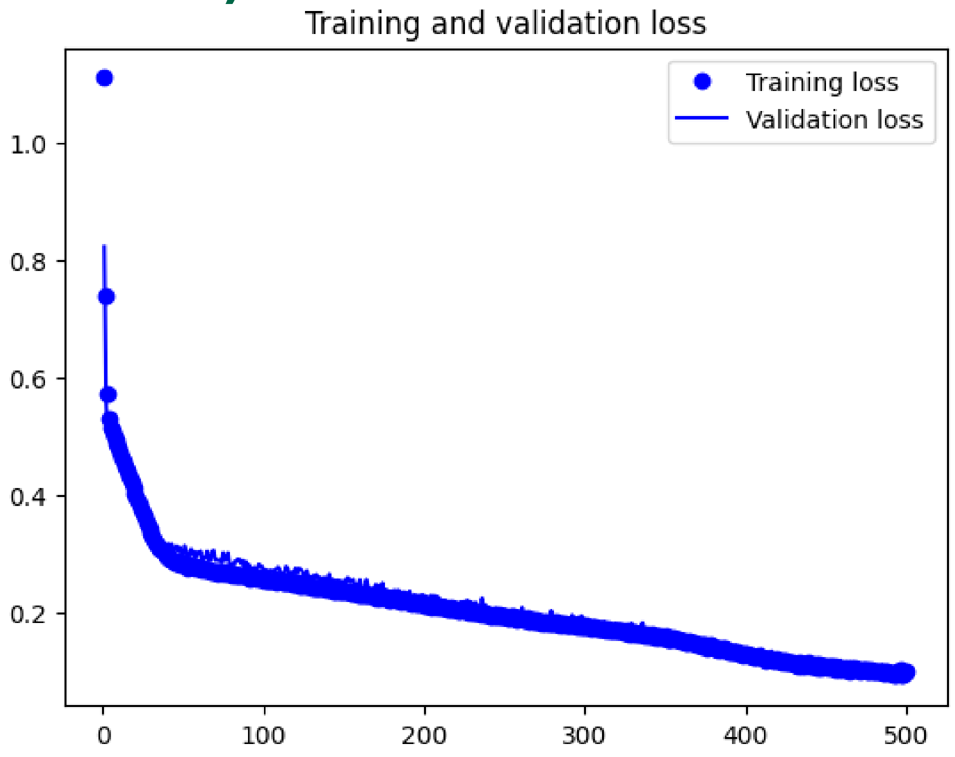
```
]  # Plot the training history
   loss = history.history['loss']
   val_loss = history.history['val_loss']

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

   plt.plot(epochs, loss, 'bo', label='Training loss')
   plt.plot(epochs, val_loss, 'b', label='Validation loss')
   plt.title('Training and validation loss')
   plt.legend()
   plt.show()
```

Not required code we premade for you (don’t code it now)

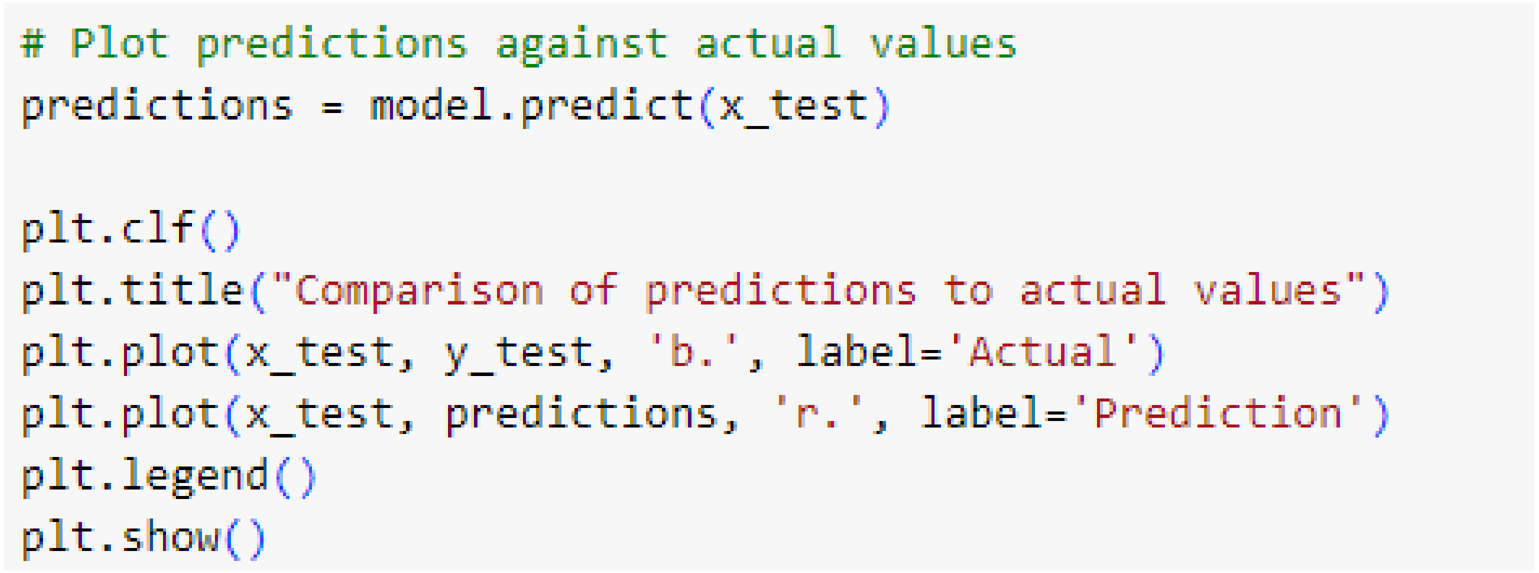


```
]   # Plot the training history
    loss = history.history['loss']
    val_loss = history.history['val_loss']

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

    plt.plot(epochs, loss, 'bo', label='Training loss')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()
    plt.show()
```

Not required code we premade for you (don't code it now)

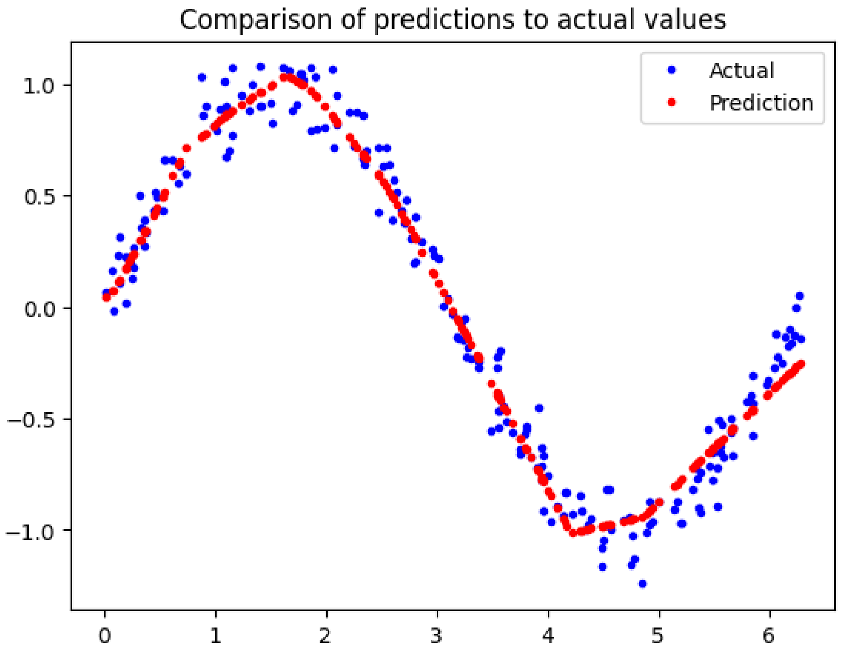


Not required code we premade for you (don’t code it now)

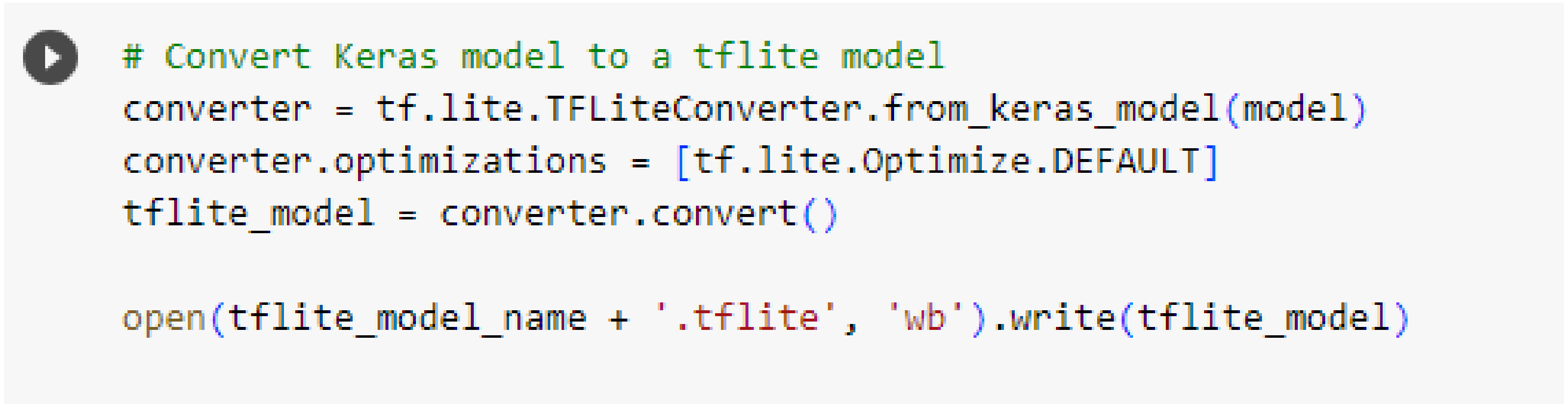


```python
# Plot predictions against actual values
predictions = model.predict(x_test)

plt.clf()
plt.title("Comparison of predictions to actual values")
plt.plot(x_test, y_test, 'b.', label='Actual')
plt.plot(x_test, predictions, 'r.', label='Prediction')
plt.legend()
plt.show()
```

Not required code we premade for you (don't code it now)

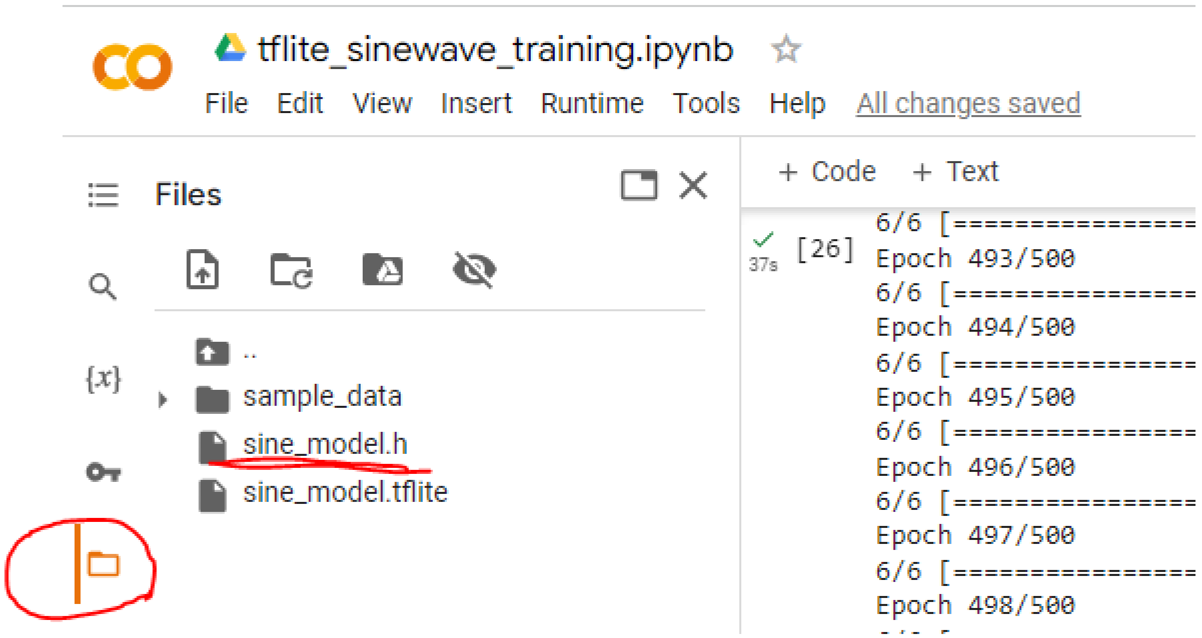


Not required code we premade for you (don’t code it now)



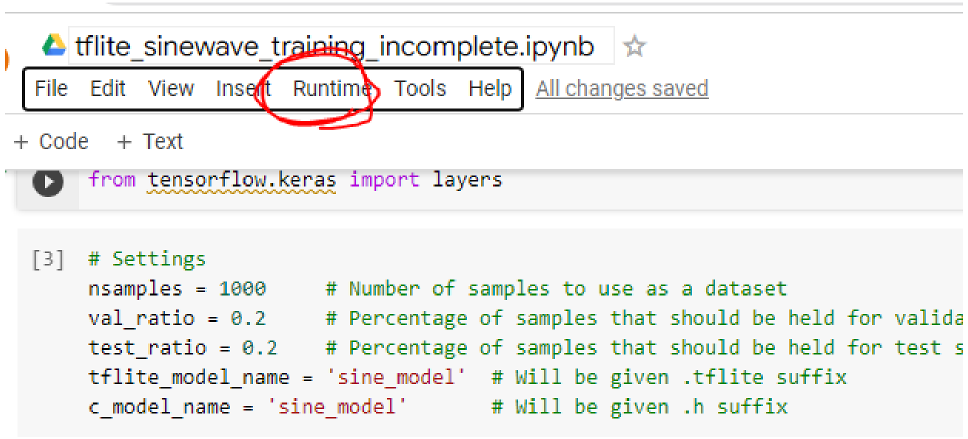
```
# Convert Keras model to a tflite model
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model = converter.convert()

open(tflite_model_name + '.tflite', 'wb').write(tflite_model)
```

Not required code we premade for you (don’t code it now)



```
6/6 [================
Epoch 493/500
6/6 [================
Epoch 494/500
6/6 [================
Epoch 495/500
6/6 [================
Epoch 496/500
6/6 [================
Epoch 497/500
6/6 [================
Epoch 498/500
```

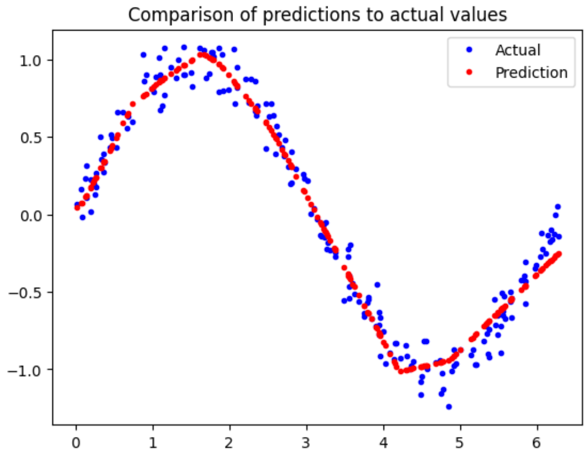
How to find that you repeated the code correctly?


tflite_sinewave_training_incomplete.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

```
from tensorflow.keras import layers
```

```
[3] # Settings
    nsamples = 1000     # Number of samples to use as a dataset
    val_ratio = 0.2     # Percentage of samples that should be held for valida
    test_ratio = 0.2    # Percentage of samples that should be held for test s
    tflite_model_name = 'sine_model'  # Will be given .tflite suffix
    c_model_name = 'sine_model'       # Will be given .h suffix
```

- Press "Runtime",
- Then "Run all".



- If on the bottom, you see this red and blue plot (could be slightly different), your code is correct.

If it is not, don't worry. In next step, we will give you the C code.

IoT

Internet of Things

Network of physical objects or “things” that are embedded with sensors, software, and other technologies to collect and exchange data with other devices and systems over the internet.

Examples of IoT Devices

- Fitness tracking devices
- Hospital call buttons

- Smart light bulbs
- Smoke Detectors (w/ internet)

- Automatic traffic lights
- Car computers (w/ internet)

IoT Systems

Microcontrollers:

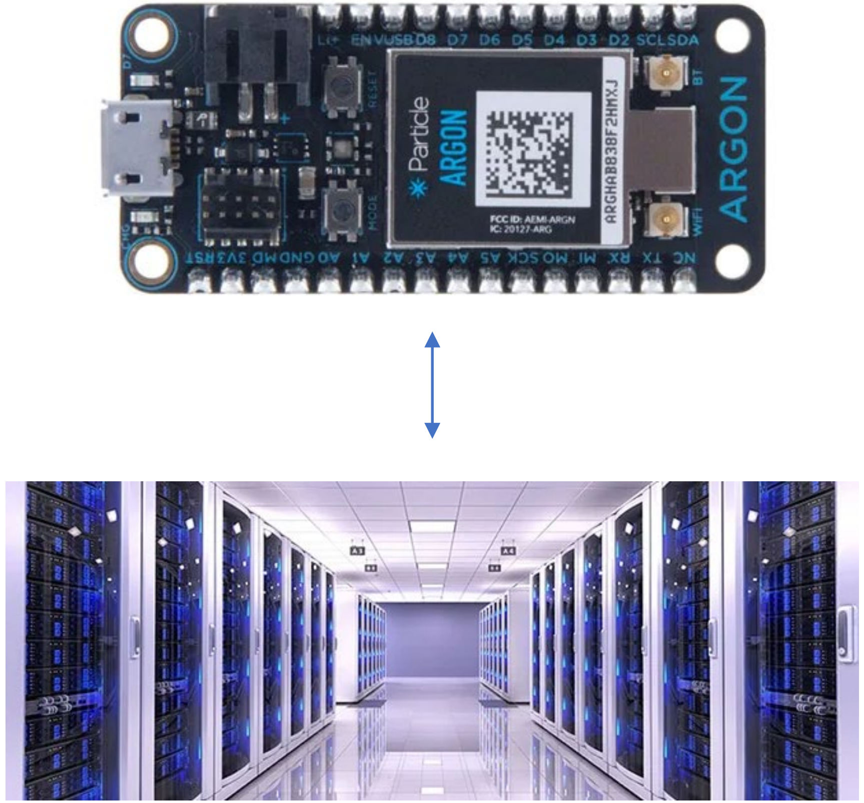
- Nordic Semiconductor nRF
- Microchip Atmega

Development Boards:

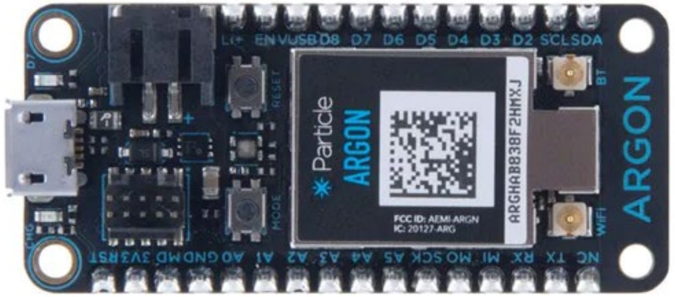
- Particle Argon
- Arduino BLE

Platforms:

- Particle IO
- Amazon Web Services
- Cisco IoT



Get the Particle Argon board



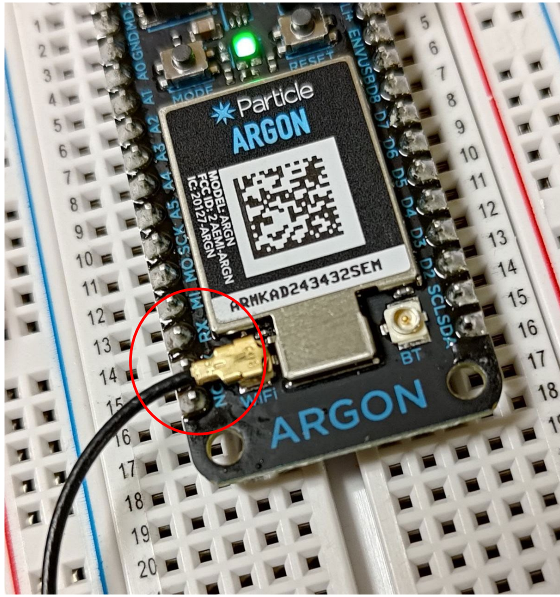
- Cool board, like Arduino but more advanced
- You get to keep it
- Please take care of it

Getting Ready

1. docs.particle.io/quickstart/argon/
2. “Set up your Argon”
3. “Get Started”
4. Make Account (required)

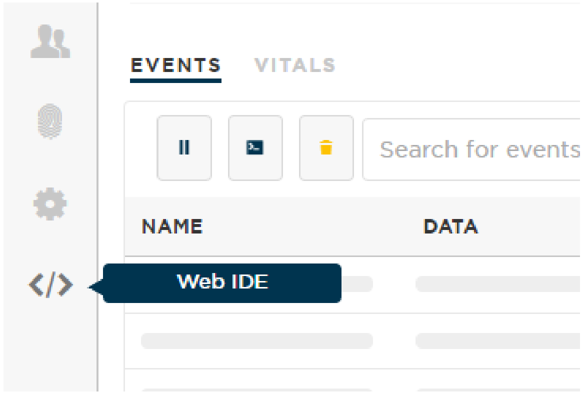
Updating

1. Attach antenna to Argon board “Wi-Fi” port
2. “Start setting up my device”
3. Attach board with white USB cable to laptop
4. “Select Device,” pick device, and “connect”
5. “Continue,” pick device, and “connect”
6. “Continue” and “Update Device”



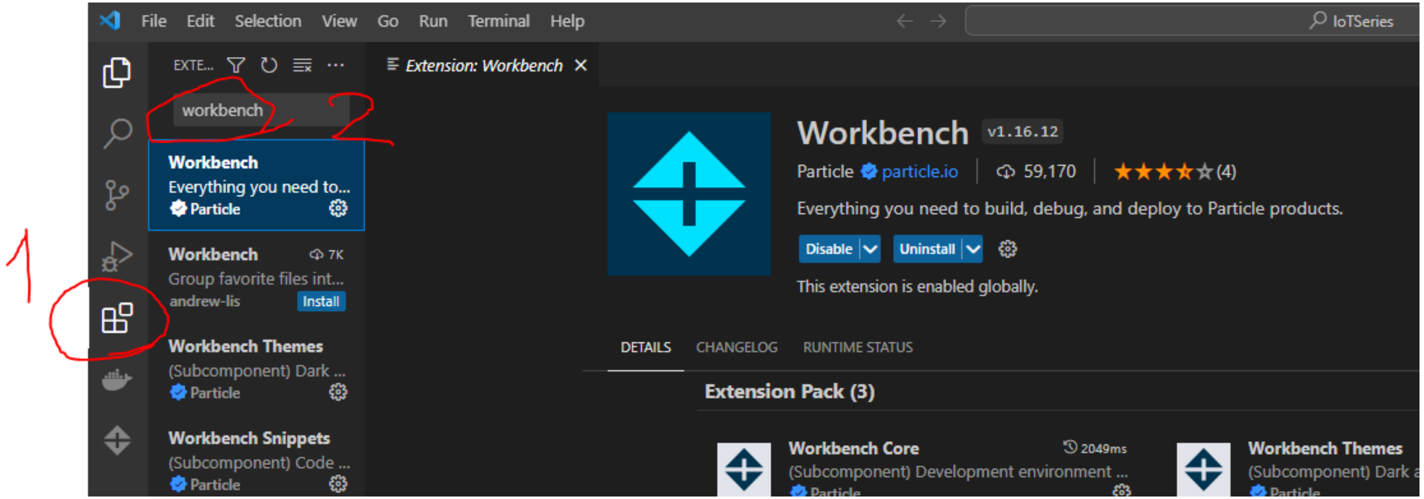
Registering Device

1. “Or create a new product” and give a name
2. “Add to product” and gave a device name
3. “Name Device”
4. Choose Wi-Fi network (Not school Wi-Fi, try phone hotspot)
5. Activate Device (if it did not work, try again)
    1. “Go to Console”
    2. Open Web IDE



Setting Up VS Code

1. Download VS Code https://code.visualstudio.com/download
2. Go to Extensions
3. Install Workbench – needed to work with the board



Workbench
Everything you need to...
Particle

Workbench 7K
Group favorite files int...
andrew-lis Install

Workbench Themes
(Subcomponent) Dark ...
Particle

Workbench Snippets
(Subcomponent) Code ...
Particle

Workbench v1.16.12
Particle particle.io | 59,170 | ★★★★☆ (4)
Everything you need to build, debug, and deploy to Particle products.
Disable Uninstall
This extension is enabled globally.

DETAILS CHANGELOG RUNTIME STATUS

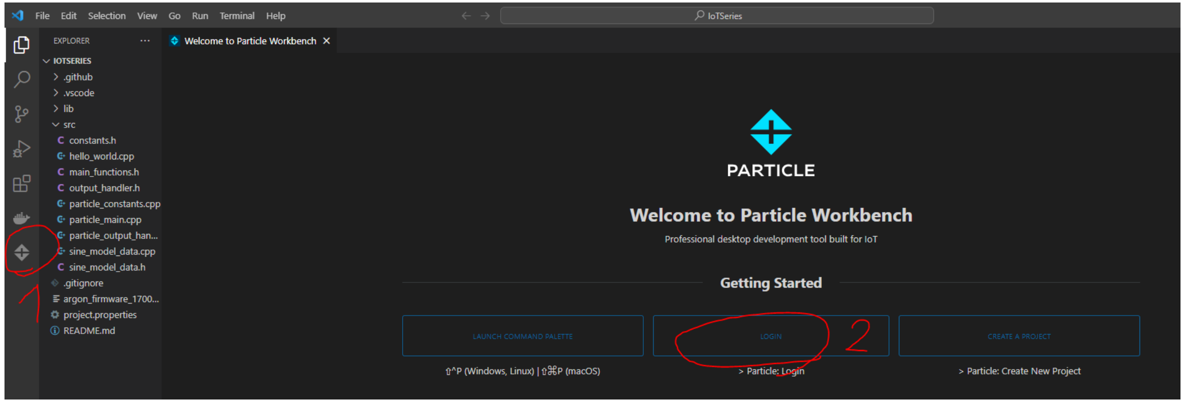
Extension Pack (3)

Workbench Core 2049ms
(Subcomponent) Development environment ...
Particle

Workbench Themes
(Subcomponent) Dark a
Particle

Login into Particle

1. Press on Particle Workbench on the left
2. Click “login”, “login with username” and login using your Particle credentials
3. When login will ask for a code, just press Enter



Create Project

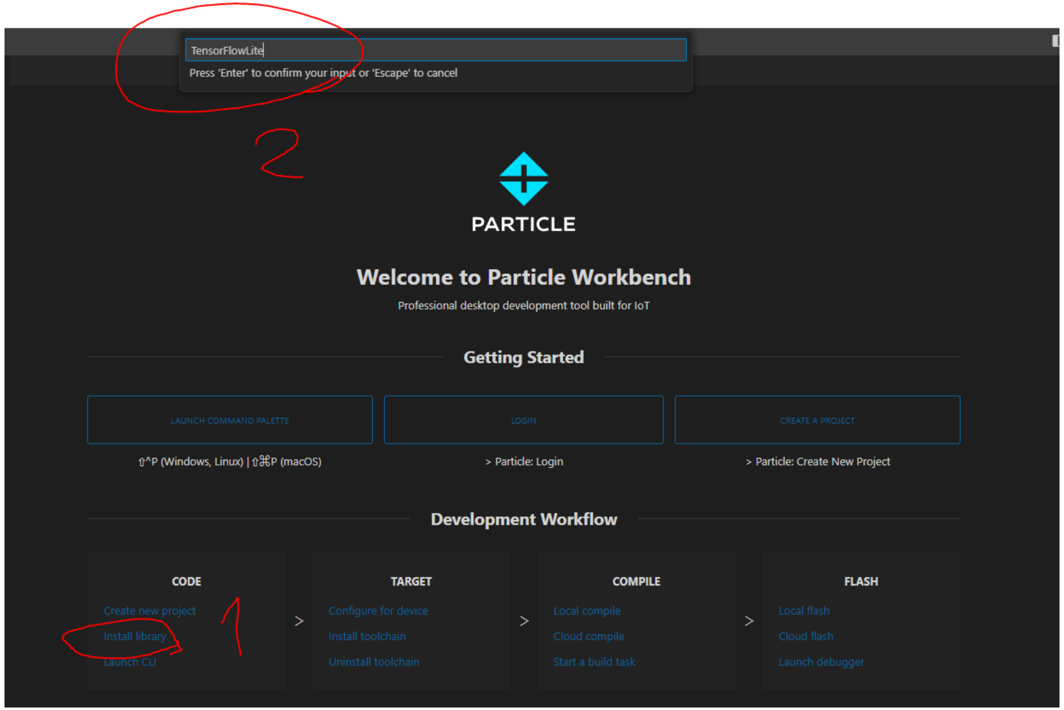
1. Press on Particle Workbench on the left
2. Click “Create a project”
3. Replace src folder with src folder from our Github. Why?

Because we made a C code that actually allows the Python ML model to run on IoT.

https://github.com/USF-IEEE-Computer-Society/IoT-worshop-ML

Install Library

1. Press on Particle Workbench on the left
2. Click “Install Library” and choose “TensorFlowLite”



TensorFlowLite

Press 'Enter' to confirm your input or 'Escape' to cancel

2

PARTICLE

Welcome to Particle Workbench

Professional desktop development tool built for IoT

Getting Started

LAUNCH COMMAND PALETTE	LOGIN	CREATE A PROJECT
⇧^P (Windows, Linux) | ⇧⌘P (macOS)	> Particle: Login	> Particle: Create New Project

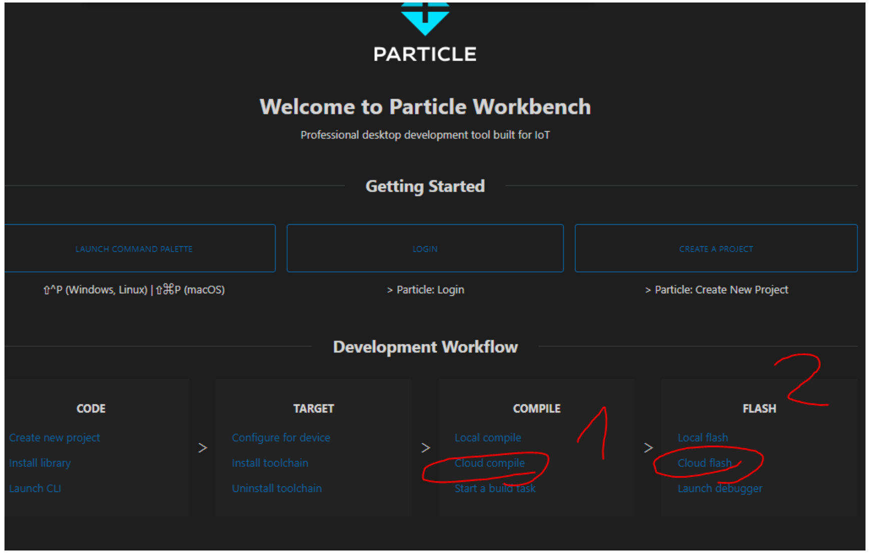
Development Workflow

CODE	TARGET	COMPILE	FLASH
Create new project	Configure for device	Local compile	Local flash
Install library	Install toolchain	Cloud compile	Cloud flash
Launch CLI	Uninstall toolchain	Start a build task	Launch debugger

1

Compile and Flash the Code

1. Press on Particle Workbench on the left
2. Click "Cloud Compile"
3. When compiling is done, click "Cloud Flash"
4. After a bit, your board should start flashing with blue gradually


PARTICLE

Welcome to Particle Workbench

Professional desktop development tool built for IoT

Getting Started

LAUNCH COMMAND PALETTE — ⇧^P (Windows, Linux) | ⇧⌘P (macOS)

LOGIN — > Particle: Login

CREATE A PROJECT — > Particle: Create New Project

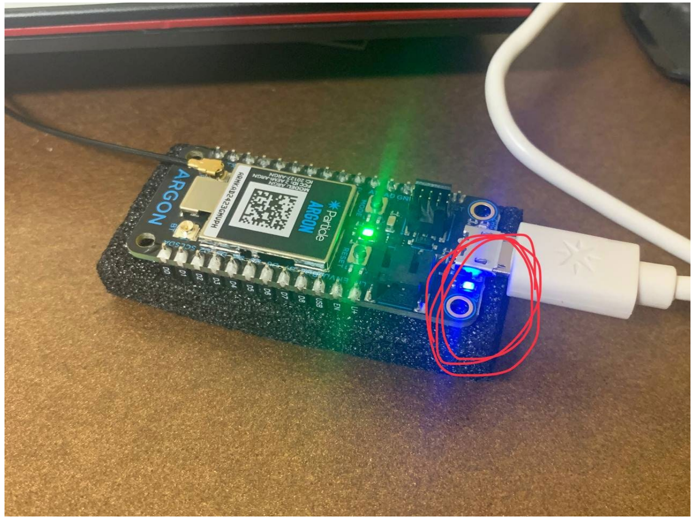
Development Workflow

CODE: Create new project, Install library, Launch CLI

TARGET: Configure for device, Install toolchain, Uninstall toolchain

COMPILE: Local compile, Cloud compile, Start a build task

FLASH: Local flash, Cloud flash, Launch debugger



Conclusion

Conclusion

We learned:

- Basics of machine learning and IoT
- Coded our own Machine Learning model that predicts sin function
- Uploaded this model on our IoT board and ran it on it
- If you liked the event, come to our events in Spring and we will work more with IoT and boards like today
- If you did not like the event, come to our events anyway. There is another Tech Lead as well.
- Special thanks to Samir Ahmed – Vice President and Liam Osman – Hardware Tech Lead for helping

Questions?

IEEE COMPUTER SOCIETY

Student Branch Chapter at the University of South Florida

UNIVERSITY of SOUTH FLORIDA

College of Engineering