TensorFlow

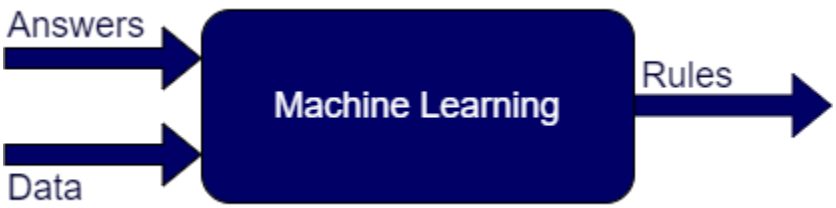
Before we dive into our TinyML applications' underlying mechanics, we must understand the software ecosystem that we will be using. The software that wraps around the models we train is just as important as the models themselves. Imagine working hard to train a tiny machine learning model that occupies only a few kilobytes of memory, only to realize that the TensorFlow framework used to run the model is itself several megabytes large.

To avoid such oversight, we first introduce the fundamental concepts around TensorFlow's usage versus TensorFlow Lite (and soon TensorFlow Lite Micro). We will be using TensorFlow for training the models. We will be using TensorFlow Lite for evaluation and deployment because it supports the optimizations we need. It has a minimal memory footprint, which is especially essential for deployment in mobile and embedded systems. Every application introduced later builds on the concepts, programming interfaces, and the mechanics behind designing and deploying TinyML models.

Recap of the Machine Learning Paradigm

Previously we were introduced to the essence of machine learning - a programming paradigm where rules are learned by a neural network.

Using TensorFlow we can create a neural network, compile it and then train it, with the training process, at a high level looking like this:



The neural network would randomly initialize, effectively making a guess to the rules that match the data to the answers, and then over time it would loop through measuring the accuracy and continually optimizing.

An example of this type of code is seen here:

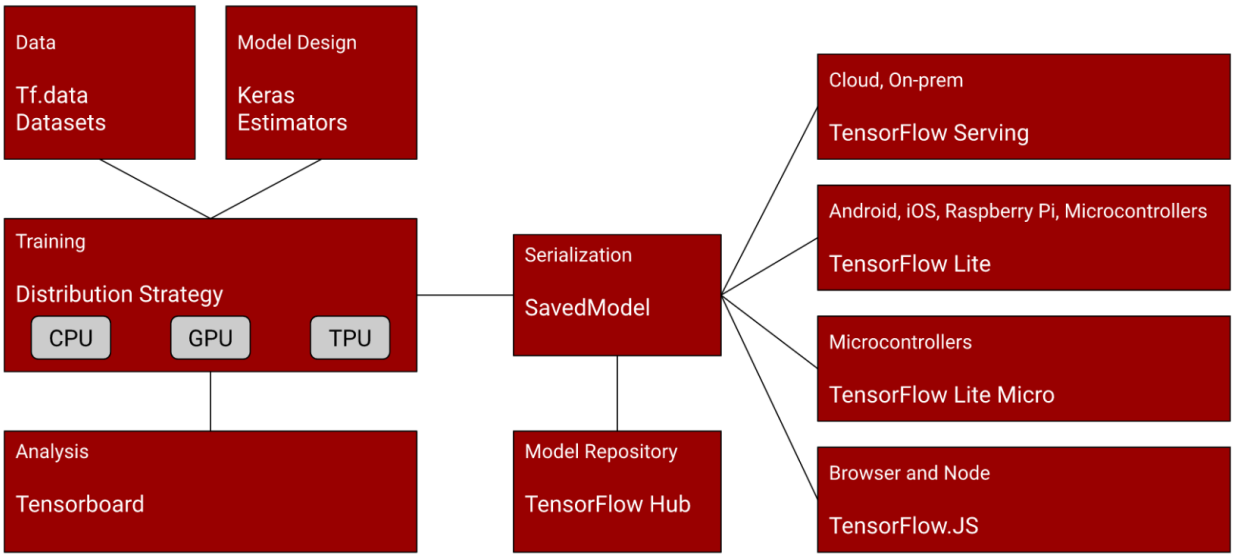
```
import tensorflow as tf
data = tf.keras.datasets.mnist

(training_images, training_labels), (val_images, val_labels) = data.load_data()
training_images  = training_images / 255.0
val_images = val_images / 255.0

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(28,28)),
                                    tf.keras.layers.Dense(20, activation=tf.nn.relu),
                                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(training_images, training_labels, epochs=20,
          validation_data=(val_images, val_labels))
```

The bottom line here is that we train a model, and we use this model in the future to get an inference based on the rules that it learned.

Model Training vs. Deployment

With TensorFlow there are a number of different ways we can deploy these models. So far we've been running inference within the colabs that we were training with. However, we can't share our model with other users this way.

The overall TensorFlow ecosystem can be represented using a diagram like this -- with the left side of the diagram showing the architecture and APIs that can be used for training models.



In the center is the architecture for saving a model, called TensorFlow SavedModel. You can learn more about it at:

https://www.tensorflow.org/guide/saved_model

On the right are the ways that models can be deployed.

- The standard TensorFlow models that we've been creating this far, trained without any post-training modification can be deployed to Cloud or on-Premises infrastructures via a technology called TensorFLow Serving, which gives us a REST interface to the model so inference can be executed on data that's passed to it, and it returns the results of the inference over HTTP. You can learn more about it at https://www.tensorflow.org/tfx/guide/serving
- TensorFlow Lite is a runtime that is optimized for smaller systems such as Android, iOS and embedded systems that run a variant of Linux, such as a Raspberry Pi. TensorFlow Lite also includes a suite of tools that help us convert and optimize our model for this runtime. https://www.tensorflow.org/lite
- TensorFlow Lite Micro is built on top of TensorFlow Lite and can be used to shrink oour model even further to work on microcontrollers and is a core enabling technology for TinyML. https://www.tensorflow.org/lite/microcontrollers
- TensorFlow.js provides a javascript-based library that can be used both for training models and running inference on them in JavaScript-based environments such as the Web Browsers or Node.js. https://www.tensorflow.org/js.