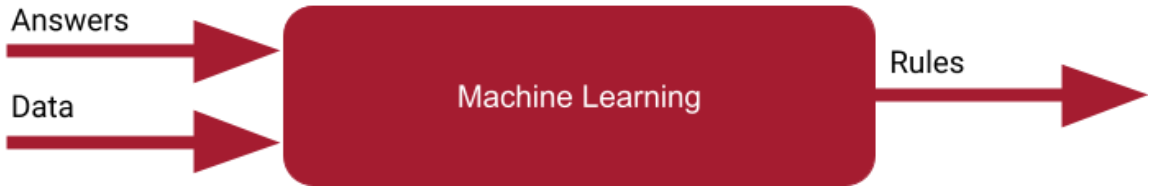
# TensorFlow: Where We Left Off

Before we dive into our TinyML applications' underlying mechanics, you 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.

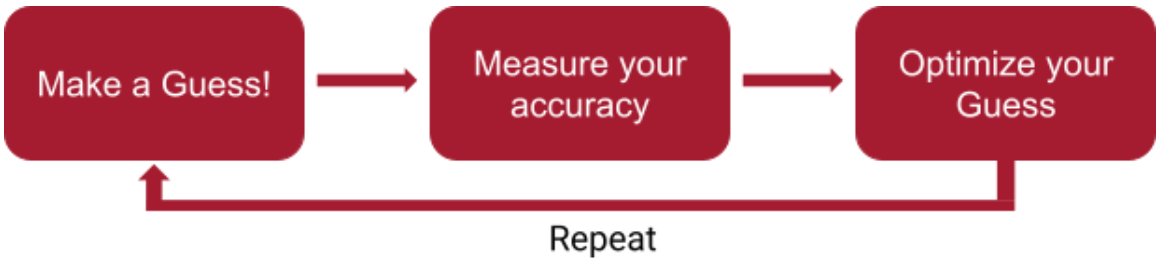
To avoid such oversight, we first introduce the fundamental concepts around TensorFlow's usage versus TensorFlow Lite (and soon TensorFlow Lite Micro). We will be using TensorFlow for training the models. We will be using TensorFlow Lite for evaluation and deployment because it supports the optimizations we need. It has a minimal memory footprint, which is especially essential for deployment in mobile and embedded systems. To this end, in the following two modules, Laurence and I are going to first introduce the concepts, programming interfaces, and the mechanics behind them before diving deep into the applications. Every application reuses these fundamental building blocks.

## Recap of the Machine Learning Paradigm

In the previous course you had an introduction to machine learning, exploring how it is a new programming paradigm that changes the programming paradigm. Instead of creating rules explicitly with a programming language, the rules are learned by a neural network.




Using TensorFlow you could 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:

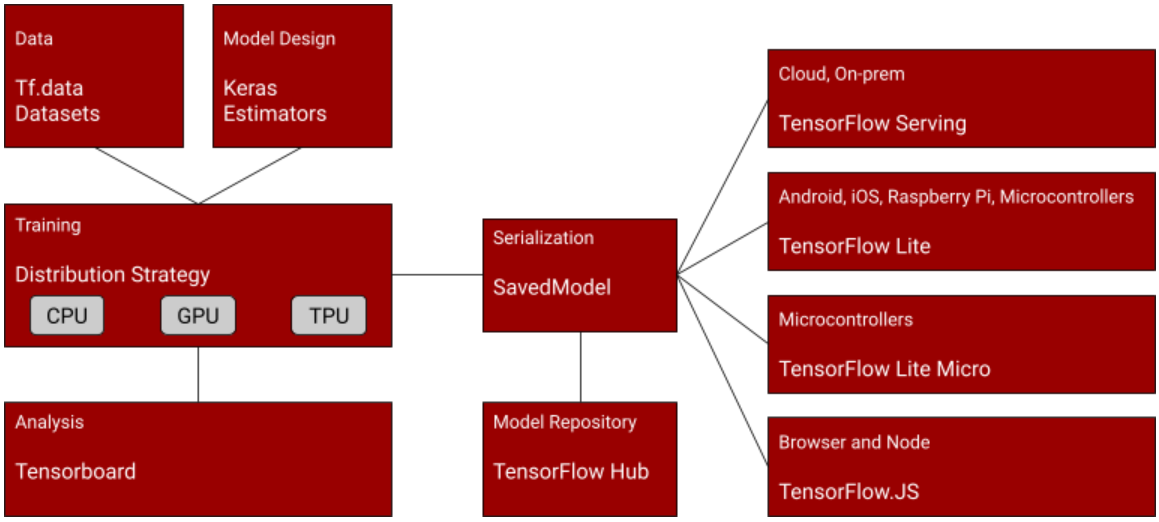
```
import tensorflow as tf
data = tf.keras.datasets.mnist

(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))
```

With the bottom line being that instead of creating a program or an app, you train a *model*, and you use this model in 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 you can deploy these models. As you've been studying this specialization, you've been running inference within the colabs that you were training with. However, you can't share your model with other users that 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 you'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 can give you 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. You'll be exploring that over the next few videos. TensorFlow Lite also includes  a suite of tools that help you *convert* and *optimize* your model for this runtime. https://www.tensorflow.org/lite
- TensorFlow Lite Micro, which you'll explore later in this course, is built on top of TensorFlow Lite and can be used to shrink your 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

Let's now explore TensorFlow Lite, so you can see how TensorFlow models can be used on smaller devices on our way to eventually deploying TinyML to microcontrollers in Course 3. Over the rest of this lesson we'll be exploring models that will generally run on devices such as Android and iOS phones or tablets. The concepts you learn here will be used as you dig deeper into creating even smaller models to run on tiny devices like microcontrollers.