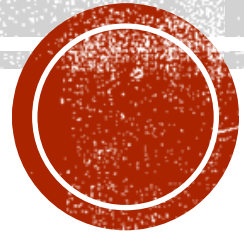
TENSORFLOW & KERAS

Dr. Brian Mc Ginley



TENSORFLOW

- TensorFlow is an open-source symbolic maths library that is popular in Machine Learning, especially for Neural Networks
- Developed by GoogleBrain, Google's AI/Deep learning division
- Note that the implementation of TensorFlow that we see on GitHub is open-source. Google maintains its own internal version.
- It is not the first package. What is unique about TensorFlow is how quickly it has gained so much market share. It is the dominant framework for NNs

SOME ALTERNATIVES

- Theano
- Caffe
- Torch
  - and there are others
- What they all have in common is they are designed to complete fast mathematics problems, in particular linear algebra routines for arrays (multiplication of matrices in particular), which is the backbone of Machine Learning techniques

KERAS

- Keras is an open-source deep learning library written in Python.
- TensorFlow/Theano/PyTorch:
  - Developing deep learning models using mathematical libraries like TensorFlow, Theano, and PyTorch was cumbersome, requiring tens or even hundreds of lines of code to achieve the simplest tasks.
  - The focus of these libraries was on research, flexibility, and speed of execution, not ease of use.
- Note: Keras is an API designed for human beings, not machines. Keras offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear & actionable error messages. It also has extensive documentation and developer guides.

KERAS

- The Keras API is clean and simple, allowing standard deep learning models to be defined, fit, and evaluated in just a few lines of code.
- A secondary reason Keras took-off was because it allowed you to use any one among the range of popular deep learning mathematical libraries as the backend (e.g. used to perform the computation), such as TensorFlow, Theano, and later, CNTK. This allowed the power of these libraries to be harnessed (e.g. GPUs) with a very clean and simple interface.

TENSORFLOW 2

- In 2019 TensorFlow 2 was released by Google with a lot of changes to the API, some fundamental changes and the code is mostly not compatible
  - Be careful when looking up tutorials, which version of TensorFlow are they made for.
  - https://www.datasciencecentral.com/profiles/blogs/tensorflow-1-x-vs-2-x-summary-of-changes shows a summary of changes.
- In short
  - TensorFlow 2 tries to be more ”Pythonic” and easy to use
  - By default, it uses Eager execution (to be more consistent with how Python usually works), whereas `v1` used Sessions to run.
  - Has built in support for Keras
    - This is usually referred to as `tf.keras`
  - For building models so much less complex

TF.KERAS

- Commonly the Keras API integrated with TensorFlow is referred to as tf.keras. This is to distinguish it from the standalone Keras open-source project that supports other backends and it follows the usual way of importing

```
import tensorflow as tf
tf.keras ....
```

- Quote (from Keras homepage): "At this time, we recommend that Keras users who use multi-backend Keras with the TensorFlow backend switch to tf.keras in TensorFlow 2.0. tf.keras is better maintained and has better integration with TensorFlow features (eager execution, distribution support and other)."

TF.KERAS

- See PyImageSearch (https://www.pyimagesearch.com/2019/10/21/keras-vs-tf-keras-whats-the-difference-in-tensorflow-2-0/) for a comparison and pay note to:

- Quote: "Keras v2.3.0 is the first release of Keras that brings keras in sync with tf.keras. It will be the last major release to support backends other than TensorFlow (i.e., Theano, CNTK, etc.). And most importantly, deep learning practitioners should start moving to TensorFlow 2.0 and the tf.keras package"

INSTALL TENSORFLOW

- It does not come as default through Anaconda so you will need to install it using conda/pip or whatever method you use.
- Your scripts should always verify the installation and version number to ensure it is installed correctly and the version is what you expect.

```
import tensorflow as tf
print(tf.__version__)
```

- Often you will get warning messages when using the API about what your hardware supports but it was not configured for use. This could be things like CPU instructions/GPU instructions.
- They are for information and will not prevent the code from running. However, some instructions can speed up the Math that the library does. GPUs are especially efficient at Matrix math.

INSTALL TENSORFLOW

- https://www.tensorflow.org/install/pip has instructions for installing it through pip.
- https://docs.anaconda.com/anaconda/user-guide/tasks/tensorflow/ has instructions for installing it through conda.
- If you want it GPU accelerated, you will need CUDA if you've an Nvidia GPU and instructions are given there too.

```
import tensorflow as tf
if tf.test.gpu_device_name():
    print('Default GPU Device:
          {}'.format(tf.test.gpu_device_name()))

else:
    print("Please install GPU version of TF")
```

GOOGLE COLAB

- A great way to try TensorFlow (and other Python projects) is by using Google collab. It allows you to access GPU resources which can greatly speed up your calculations
  - Google Colab: https://colab.research.google.com/
- Free to use, you will be in a queue for some things, but if you don't have a decent GPU in your own machine can be very useful

LONGER EXAMPLE

- Let's use LinearRegression as a basic example to look at building a model and to demonstrate how to use TensorFlow
- In short, say we have a sample of data, each item in the sample has 1 feature (x) and you wish to create a model:

$$\hat{y} = \alpha + \beta \mathbf{x}$$

- or if you have multiple features ($\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$) then the model looks like:

$$\hat{y} = \alpha + \beta_1 \mathbf{x}_1 + \beta_2 \mathbf{x}_2 + \ldots + \beta_m \mathbf{x}_m$$

REGRESSION PROCESS

- The process for regression usually follows the following steps

1. Formulating the problem
   - Selecting a tentative form of the model (linear, polynomial etc.)
   - Selecting the features
2. Fit the model
   - Use the data we know the correct answer to and find the best fit for the parameters with that data.
   - So that the predictions at the end are close to what the actual values were for each x.
3. Model validation
   - Evaluate how it performs and the stability of it

MODEL LIFE-CYCLE

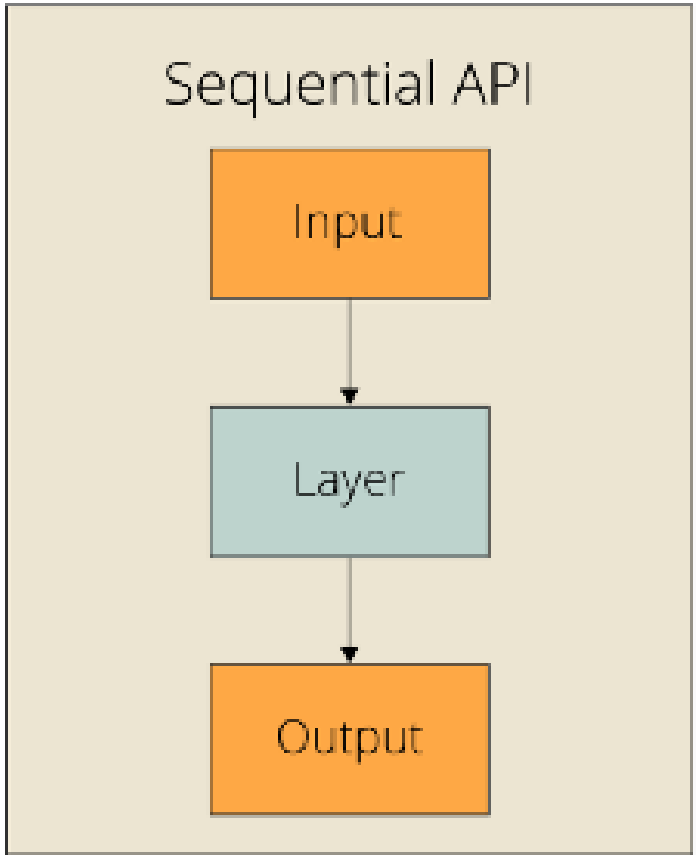
- Previous slide is fine for any ML process. ”Selecting tentative form” could be select a neural network. In terms of how this is programmed, using tf.keras has the following process:

1. Define the model
2. Compile the model
3. Fit the model
4. Evaluate the model
5. Make predictions

DEFINE THE MODEL

- There are two ways to define the model
  1. The sequential model
  2. Functional API
- The sequential model is what will be demonstrated
  - A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.
  - The picture shows one ”layer” but we could have multiple sequential layers in it, one thing follows the other.

```
model = tf.keras.Sequential(...)
# array of " layers"
```



COMPILE THE MODEL

- Compiling the model requires that you first select a loss (error/cost) function that you want to optimise, such as mean squared error or cross-entropy. You want parameters that give you the lowest error.
- It also requires that you select an algorithm to perform the optimization procedure, typically stochastic gradient descent, or a modern variation, such as Adam.
- The algorithm is typically an iterative method that will keep adjusting the weights until a minimum error is found.

```
model.compile (
   optimizer=tf.optimizers.Adam( learning_rate=0.1 ) ,
   loss='mean_absolute_error')
```

FIT THE MODEL

- Fitting the model means putting the data you have in and figuring out the ”best” parameters for that data.

```
model.fit(X, y, epochs =500, verbose =0)
```

- where X is the array storing all the samples and their features and y is the vector storing the known response for each feature
- Fitting is where the longest amount of time takes. In a neural network, it can take weeks depending on the size of the dataset, the complexity of the model and the CPU/GPU power you’ve thrown at it. It also may require a lot of memory.

VALIDATION SET

- If we want to have a validation set to monitor at the end of each Epoch, we can add

```
model.fit(X, y, epochs =500, verbose =0, validation_split =0.2)
```

- and you will get the val_loss and val_accuracy if you have specified these.
- Warning:
  - When using this for a validation set, 20% of the end of the data is just sliced away (no shuffling or randomisation!).
  - If your data is ordered information, you are going to get very poor results
  - If you have previously used train_test_split(), then the data will be randomized, so ok to use.

EVALUATE THE MODEL

- Evaluating is much faster than fitting, as the model is not changed but it is proportional to how much data you are using to evaluate

```
loss = model.evaluate(X, y, verbose =0)
```

PREDICT

- And the whole point of everything! Being able to make predictions on previously unseen data.
- In terms of code, it is quite straightforward. Say X_{new} is a vector of data (i.e. it has all the features) then

```
yPred = model.predict(Xnew)
```

- and it gives you the prediction. This is true whether we are using regression or classification (seeing if something is a picture of a cat/dog). You can also have X_{new} to be a matrix where each row is one sample.