# Keras

In this module we will introduce <u>Keras (https://keras.io/)</u>, a high level API for Neural Networks.

To be specific

- we will mostly restrict ourselves to the Keras Sequential model
- this will greatly simplify your learning and coding
- it will restrict the type of Deep Learning programs that you can write
  - but not a meaningful restriction for the simple programs that you will write in this course

### After we introduce the high level Keras API

- we will review the history of Deep Learning programming to see how we got here
- this will give you greater insight into what Keras does "under the covers"
  - appreciate history
  - aid your diagnostics

#### Note:

The code snippets in this notebook

• are illustrative: they are code fragments and will not actually execute in this notebook

#### Confusion warning:

- There are two similar but different packages that implement Keras
  - one built into TensorFlow (the one we will use)
  - a separate project

Later in this module we will explain the difference and why it's important to distinguish between them.

# The Keras Sequential Model

**Reference**: Getting started with the Keras Sequential Model (https://keras.io/getting-started/sequential-model-guide/)

Keras has two programming models

- Sequential
- Functional

We will start with the Sequential model

The Sequential model allows you to build Neural Networks (NN) that are composed of a sequence of layers

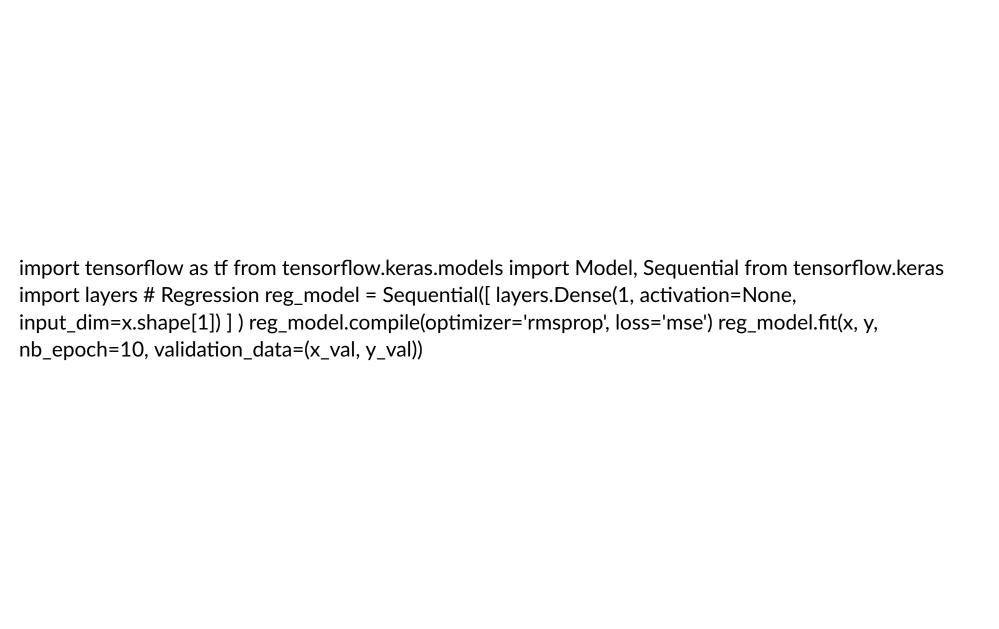
- just like our cartoon
- a very prevalent paradigm

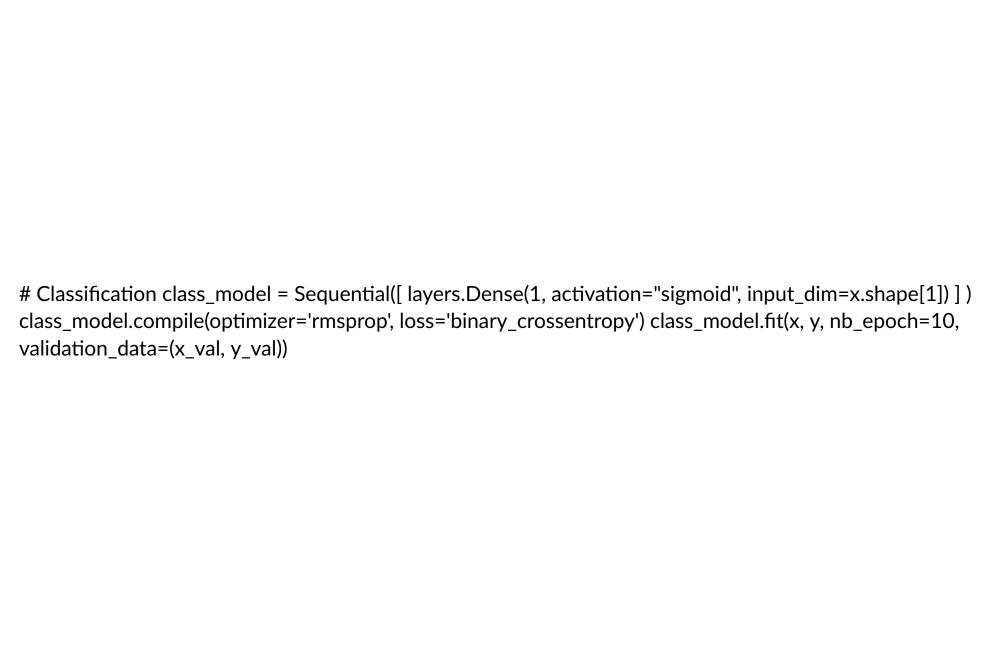
This will likely be sufficient in your initial studies

- but it restricts the architecture of the Neural Networks that you can build
- use the Functional API for full generality
  - but it might appear more complicated

Let's jump into some code.

Some old friends, in new clothing:



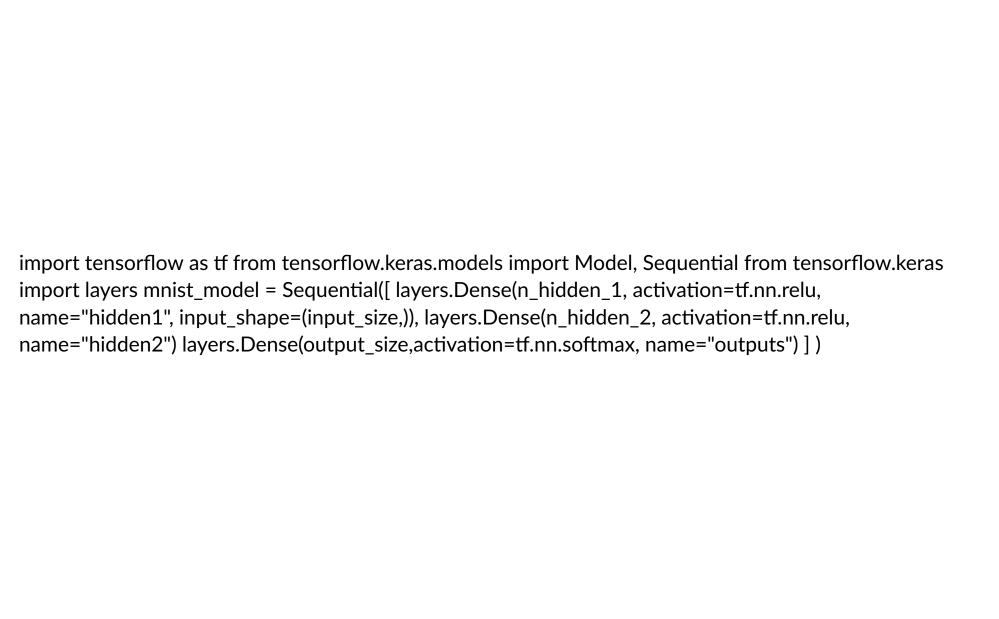


### TL;DR

- Both examples are a single layer
  - Dense, with 1 unit ("neuron")
- Regression example
  - No activation
  - MSE cost
- Binary classification example
  - Sigmoid activation
  - Binary cross entropy cost

Hopefully you get the idea.

Let's explore a slightly more complicted model.



This defines a NN with three layers

• we will explain the layers in detail later

To use the model, you first need to "compile" it

metrics = [ "acc" ] mnist\_model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=metrics)

"Compiling" is quite significant as we will demonstrate later

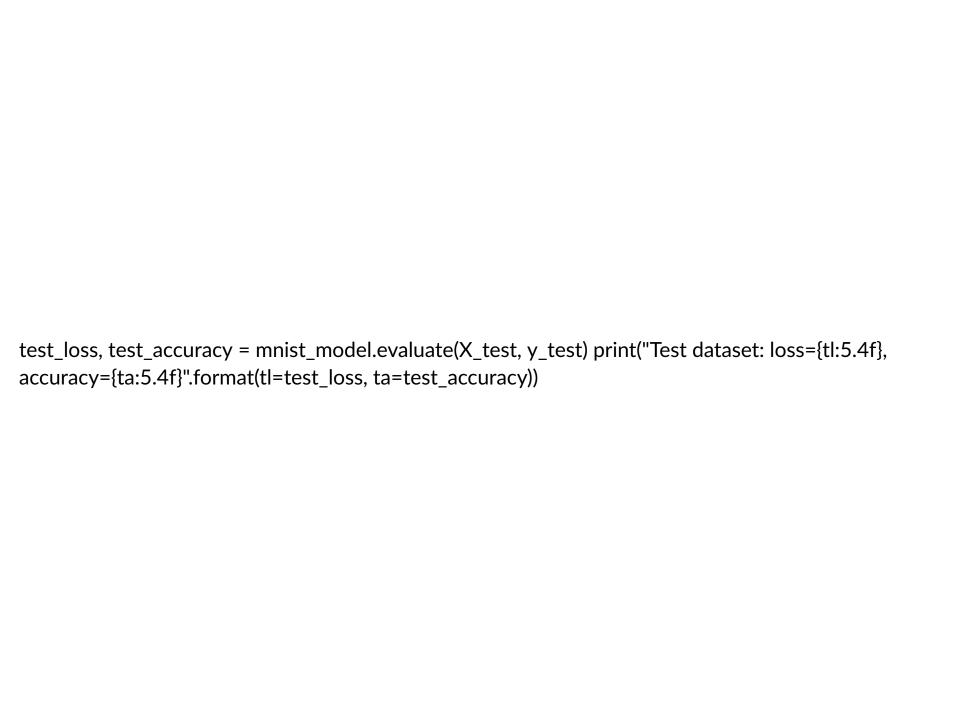
• For now: it is where you define the Cost/Loss function

Next, just as in sklearn: you "fit" the model to the training data.

history = mnist\_model.fit(X\_train, y\_train, epochs=n\_epochs, batch\_size=batch\_size, validation\_data= (X\_valid, y\_valid), shuffle=True)

Once the model is fit, you can predict, just like sklearn.

Here we evaluate the model on the Test dataset.



### The idea is quite simple

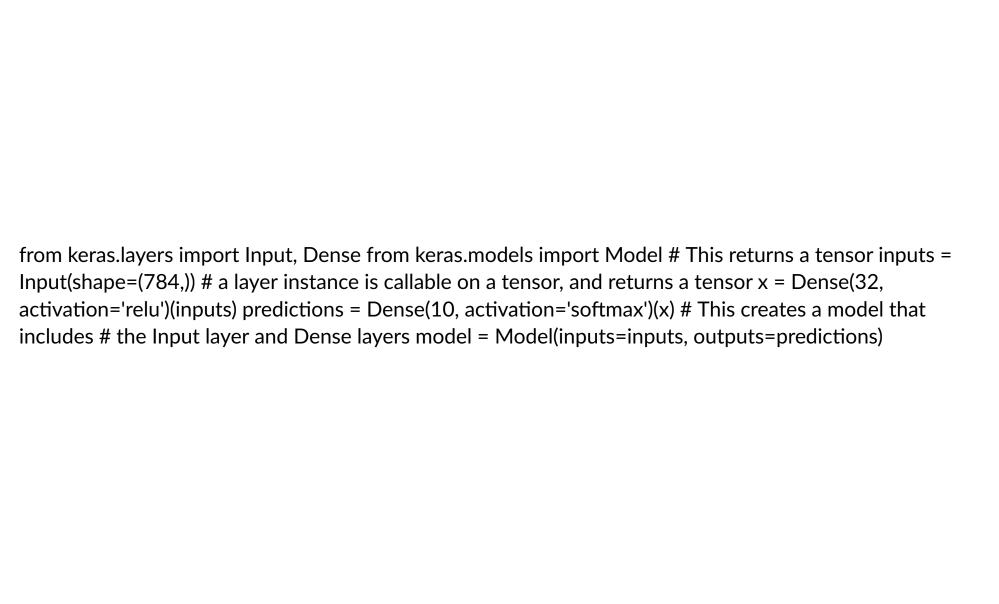
- Keras Sequential implements an sklearn-like API
  - define a model
  - fit the model
  - predict

We have glossed over a lot of details

- What does each layer do?
- Why do we need to "compile"?
  - and why does it need an optimizer?

## The Keras Functional Model

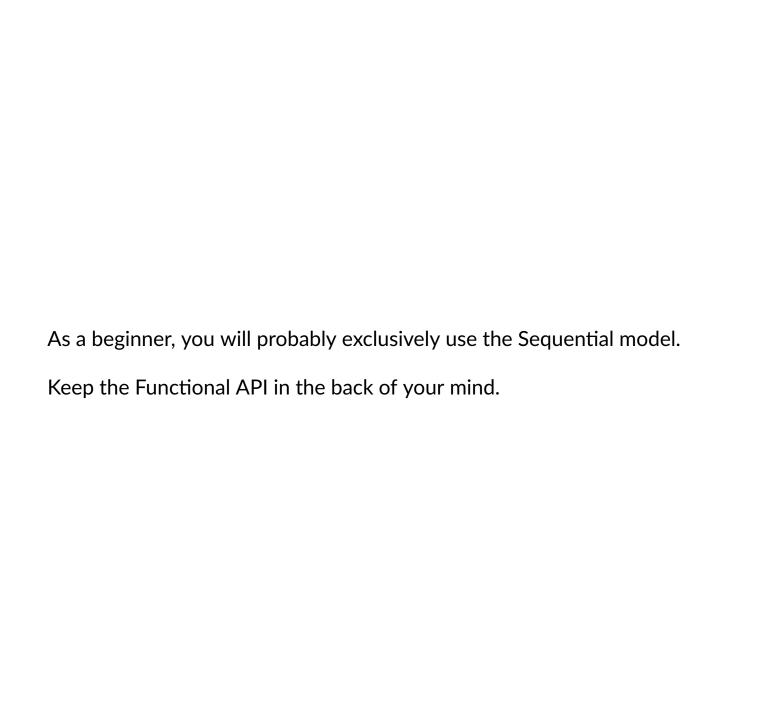
- More verbose than Sequential
- Also more flexible
  - you can define more complex computation graphs (multiple inputs/outputs, shared layers)



#### Highlights:

- Manually invoke a single layer at a time
  - Passing as input the output of the prior layer.
- You must define an Input layer (placeholder for the input/define it's shape)
  - Sequential uses the input\_shape= parameter to the first layer
- You "wrap" the graph into a "model" by a Model statement
  - looks like a function definition
    - names the input and output formal parameters
  - a Model acts just like a layer (but with internals that you create)





# **Archaelogy**

Before Keras there was a layer API.

Before the layer API, there was raw TensorFlow.

We will perform a deeper dive to uncover the multiple layers.



- The Deep Learning area evolves very quickly
- Papers/blog posts/books become outdated very quickly
  - The code in the Geron book is **not** exactly what we would use today
  - And the code we show today will not be that which is recommended a year from now

You will constantly encounter "older" code (without necessarily knowing it)

- It is still very useful
- You must understand why it is superfiically different in order to appreciate the important parts

# **Detour: DNN Tensorflow notebook on Colab**

We will now switch to a notebook running on Google Colab

- will perform the archaelogical dig
- maybe talk about Colab too

<u>DNN Tensorflow example Notebook from github</u>
(<a href="https://colab.research.google.com/github/kenperry-">https://colab.research.google.com/github/kenperry-</a>
public/ML Fall 2019/blob/master/DNN TensorFlow example.ipynb)

We will return to this notebook to clarify the difference between similar but different versions of Keras.

# Back from the detour: summary

Keras is not tightly integrated into TensorFlow (even more so in TensorFlow 2.0)

This cleans up a rather unruly TensorFlow eco-system that resulted in similar functionality in multiple places.

This can make it very confusing for someone new to TensorFlow.

There are lots of examples on the web written using various similar-looking packages.

I'll try to point out potential sources of confusion. Beware!

<u>Demystify the TensorFlow APIs (https://medium.com/google-developer-experts/demystify-the-tensorflow-apis-57d2b0b8b6c0)</u> summarizes it well

- tf.layers is going away in TensorFlow 2.0
  - tf.keras is recommended going forward
  - Do not use
- <u>Estimators (https://www.tensorflow.org/guide/estimators)</u> (tf.Estimator)
  - Estimators are sometimes called "models in a box"; somewhat similar to sklearn
    - pre-canned high-level models (like Classifiers) rather than low-level tf.keras.layers (like Dense) from which it is built
    - convenient interface to <u>Datasets for Estimators</u>
       (<a href="https://www.tensorflow.org/guide/datasets">https://www.tensorflow.org/guide/datasets</a> for estimators)
      - no need to create own mini-batches, etc.
  - You can achieve quite a bit of the convenience using Keras, so we will skip Estimators.

- Low-level TensorFlow
  - great for learning
  - better to rely on pre-defined layers when possible

#### And our own observations

- tf.contrib
  - this was a name-space created to enable users to contribute useful packages.
  - some of these packages may have made their way into the core, or been integrated elsewhere
    - tf.contrib.learn.Estimator is the obsolete version of tf.Estimator
  - eliminated from TensorFlow 2.0
    - avoid
- <u>Datasets API (https://www.tensorflow.org/guide/datasets)</u>
  - an API to handle large datasets, in memory-

We will focus on two styles or packages in our course

- tf.keras
  - this is the future, as it will be tightly integrated into TensorFlow 2.0
- tf.layers modules (e.g., tf.layers.dense)
  - used only to be compatible with the Geron book.
  - it is slightly lower level than Keras

# TensorFlow 2.0

TensorFlow 2.0 is a new version of TensorFlow that is in "beta".

It makes a very important changes from the current version

eager execution becomes the default

This is a very good thing

 non-eager execution is probably the most confusing part of TensorFlow to beginners So why aren't we using it?

- Most of the code you will find in papers/the Web requires non-eager execution
- Still in beta

I strongly recommend that you stick with a version of TensorFlow < 2.0

# tensorflow.keras vs keras (Confusion alert)

#### TL;DR

#### YES

```
• import tensorflow as tf tf.keras.layers.Dense(...)
```

 from tensorflow import keras keras.layers.Dense(...)

#### NO

```
• import keras keras.layers.Dense( ... )
```

Technically speaking: Keras is an API -- a specification -- not a library.

- TensorFlow has implemented this specification as a submodule of the TensorFlow module:
  - tensorflow.keras
- There is a separate Keras project and module: keras
  - that supports multiple "backends", including TensorFlow
  - Cannot run Python versions > 3.6 (one backend isn't cooperating)

#### This is not just a legal difference

they are separate modules that do very similar things

#### This may get confusing

- The <u>TensorFlow docs for Keras (https://www.tensorflow.org/guide/keras)</u> refers to TensorFlow's implementation of the API
  - used as from tensorflow import keras
  - this is what we will use!
  - other syntactic forms to use: tf.keras...
- The <u>Keras docs (https://keras.io/)</u> refers to the abstract Keras API and keras module
  - used as import keras as keras

### tensorflow.keras

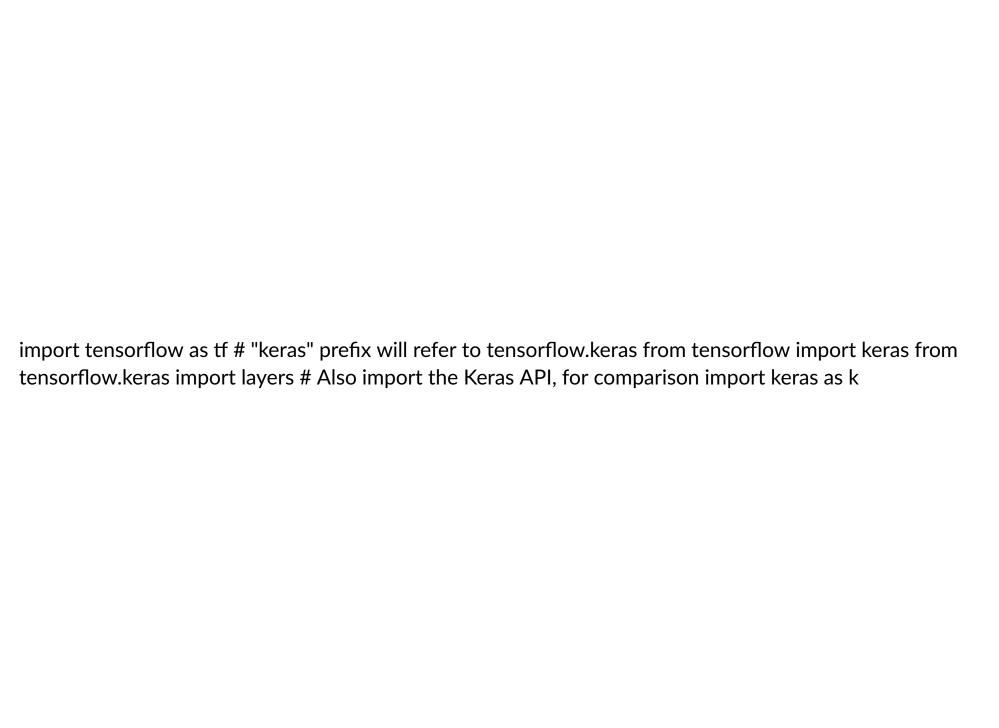
<u>Guidance from TensorFlow team (https://medium.com/tensorflow/standardizing-on-keras-guidance-on-high-level-apis-in-tensorflow-2-0-bad2b04c819a)</u>

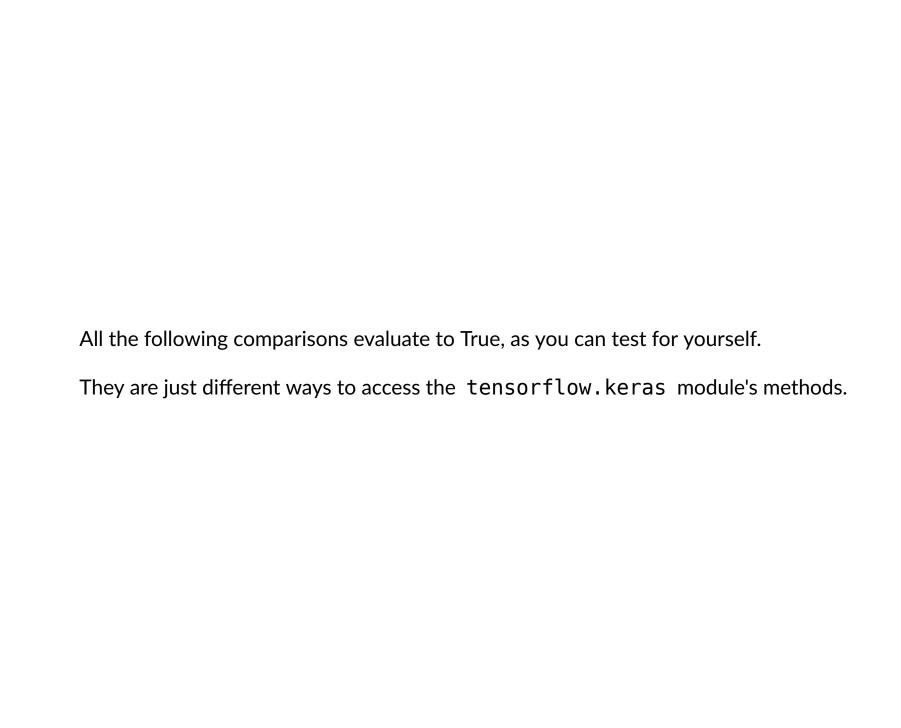
- tf.keras is an implementation of the Keras API
  - with enhancements
    - eager execution
  - integrated into TensorFlow ecosystem
    - ∘ tf.data

import tensorflow as tf Dense = tf.keras.layers.Dense model = tf.keras.Sequential() model.add(layers.Dense(64, activation='relu'))

# Technical point: showing the difference between tensorflow.keras and keras

Here we demonstrate that although the two modules implement the same methods, the are different methods





tf.keras.layers.Dense == keras.layers.Dense tf.keras.layers.Dense == layers.Dense

But the following is **not** True because they come from different packages:

- one from tensorflow.keras
- one from keras

tf.keras.layers.Dense == k.layers.Dense