

Keras

In this module we will introduce Keras (<https://keras.io/>), 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 *fragments* of a larger [notebook](#) ([DNN TensorFlow example.ipynb](#)).

- are illustrative: will not actually execute in this notebook but will in the complete 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

TL;DR

YES

- `import tensorflow as tf`
 `tf.keras.layers.Dense(...)`
- `from tensorflow import keras`
 `keras.layers.Dense(...)`

NO

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

If you want to know the details, visit this [notebook](#)
([Tensorflow Keras Archaeology.ipynb#tensorflow.keras-vs-keras-\(Confusion-alert\)](#))

The Keras Sequential Model

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


```
import tensorflow as tf from tensorflow.keras.models import Model, Sequential from tensorflow.keras
import layers # Regression reg_model = Sequential([ layers.Dense(1, activation=None,
input_shape=X[0].shape) ] ) reg_model.compile(optimizer='rmsprop', loss='mse') reg_model.fit(x, y,
nb_epoch=10, validation_data=(x_val, y_val ))
```

- A model uses the Sequential architecture
- A sequence (implemented as an array) of layers
 - Single element array
 - Consisting of a Dense (Fully connected) layer
 - with 1 output
 - No activation
 - Implements Regression
- Loss is mse

```
# Classification
class_model = Sequential([ layers.Dense(1, activation="sigmoid", input_shape=X[0].shape) ]
) class_model.compile(optimizer='rmsprop', loss='binary_crossentropy') class_model.fit(x, y, nb_epoch=10,
validation_data=(x_val, y_val))
```

- A model uses the `Sequential` architecture
- A sequence (implemented as an array) of layers
 - Single element array
 - Consisting of a `Dense` (Fully connected) layer
 - with 1 output: binary classification
 - sigmoid activation
 - Implements Classification
- Loss is `binary_crossentropy`

TL;DR

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

Hopefully you get the idea.

Let's explore a slightly more complicated model.

```
import tensorflow as tf from tensorflow.keras.models import Model, Sequential from tensorflow.keras
import layers mnist_model = Sequential([ layers.Dense(n_hidden_1, activation=tf.nn.relu, name="hidden1",
input_shape=X[0].shape), layers.Dense(n_hidden_2, activation=tf.nn.relu, name="hidden2")
layers.Dense(output_size,activation=tf.nn.softmax, name="outputs") ])
```

- A model uses the `Sequential` architecture
- A sequence (implemented as an array) of layers
 - 3 layers (3 element array)
 - 2 Dense layers
 - with varying number of outputs: `n_hidden_1`, `n_hidden_2`
 - `relu` activation
 - A Dense layer implementing Multinomial Classification
 - number of outputs equal to number of classes
 - `softmax` activation

- The first two layers "transform" the input
- The "head" layer implements Multinomial Classification

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 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.

```
test_loss, test_accuracy = mnist_model.evaluate(X_test, y_test) print("Test dataset: loss={tl:5.4f}, accuracy={ta:5.4f}".format(tl=test_loss, ta=test_accuracy))
```

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 ?

We will explain some of these details as part of our explanation of the code in our first example notebook (last section of this module).

Sequential model summary

- A Sequential model consists of an array of layers
 - The array may be constructed in several ways
 - By passing it to the `Sequential` constructor
 - By appending one layer at a time to an existing model
- ```
model.add(Dense(10, activation="relu"))
```

The first layer of a Sequential model is special.

**Best practice** is to provide an indication of the size of an example as part of the first layer's definition

- Usually the `input_shape` argument (but sometimes different)

```
Sequential([Dense(10, activation="relu",
input_shape=X[0].shape])
```

- where `X` is the matrix of training examples
  - so `X[0].shape` is the shape of the first (and hopefully each) example

It is **not** necessary (but still a good idea) to specify the input shape. If you do not

- You can't examine the model, e.g.

```
model.summary()
```

- until **after** you first call it
  - the shape of the input will be inferred from the first example with which the model is called

## Technical note

The number of weights of the first layer depends on the shape of an input example.

- If you provide the shape: it is known immediately
- If you do not provide the shape, it will not be known until the model is first run
  - inferred from first example

The number of weights of layers subsequent to the first can be inferred from the previous layer.

# The Keras Functional Model

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

```
from keras.layers import Input, Dense from keras.models import Model # This returns a tensor inputs =
Input(shape=(784,)) # a layer instance is callable on a tensor, and returns a tensor x = Dense(32,
activation='relu')(inputs) predictions = Dense(10, activation='softmax')(x) # This creates a model that
includes # the Input layer and Dense layers model = Model(inputs=inputs, outputs=predictions)
```

## 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 its 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)

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy']) model.fit(data, labels) # starts training
```



As a beginner, you will probably exclusively use the Sequential model.

Keep the Functional API in the back of your mind.

# Let's code !

Lets see a working notebook.

Two options

- Run on your local machine: [DNN Tensorflow example Notebook local \(DNN TensorFlow example.ipynb\)](#) (local)
  - Tensorflow version 2+ only !
- Run on Google Colab: [DNN Tensorflow example Notebook from github \(https://colab.research.google.com/github/kenperry-public/ML\\_Spring\\_2023/blob/master/DNN\\_TensorFlow\\_example.ipynb\)](#) (Colab)

```
In [1]: print("Done")
```

Done

