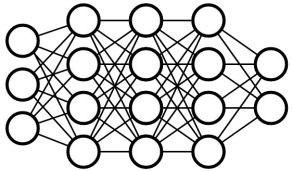
Intro to Machine Learning with Keras

--

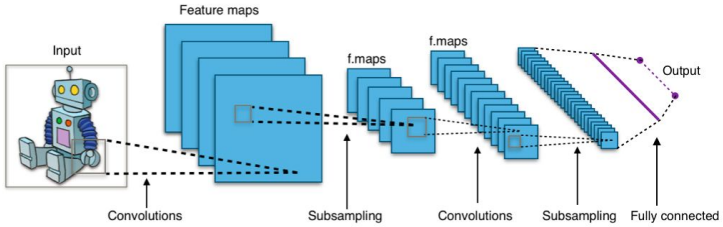
Federico Errica
Ph.D. Student
Dept. of Computer Science, University of Pisa
Mail: federico.errica@phd.unipi.it
Website: http://pages.di.unipi.it/errica

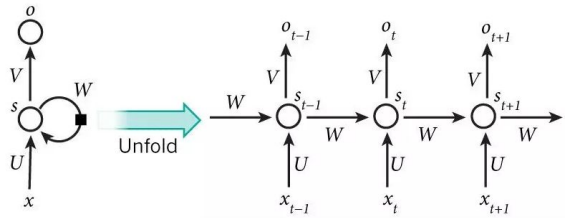
Computational Intelligence
and Machine Learning Group

Lecture Outline



- Keras 101 (TensorFlow 2.0 backend)
- Split your data 101
- **Learn via examples:**
  - Linear Model / Multi-Layer Perceptron
  - Neural Autoencoders for anomaly detection
  - Convolutional Neural Networks for image classification
  - Recurrent Networks for time-series prediction





About the Lab

- The code of the exercises will remain publicly available
  https://github.com/diningphil/Intro_Keras
- If you have doubts, **please interrupt me! I'll do my best to answer**
- **Acknowledgments:** Francesco Crecchi and Daniele Castellana
- **Do try** this at home ;)

In theory, **theory** and **practice** are the **same**.
In **practice**, **they are not**.

(supposedly) Jan L. A. van de Snepscheut

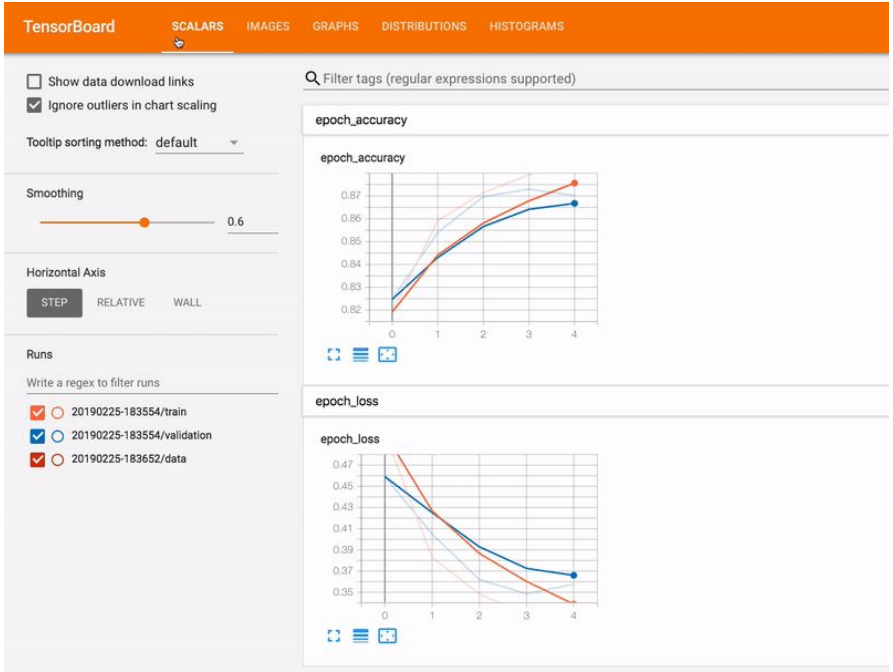
TensorFlow 2.0

- A Machine Learning framework by Google
  - 2015 → over 100M downloads in 2020
- **Production vs Prototyping**
  - Static optimization for faster training/inference
  - Eager execution in TF 2.0
    - **details later**
  - Can be quite complex to learn at first
    - Change of coding paradigm
  - Less intuitive than **PyTorch**

Tensorboard

Interactive visualization

- Training logs
  - Train/val/test
- Model's graph
- Project embeddings in 2D
- Histograms of weights, biases
- Images, audio and text



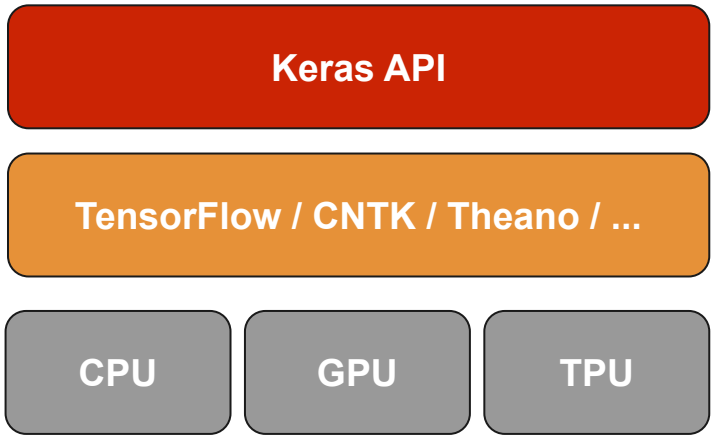
https://www.tensorflow.org/tensorboard

Keras 101

Credit goes to F. Crecchi

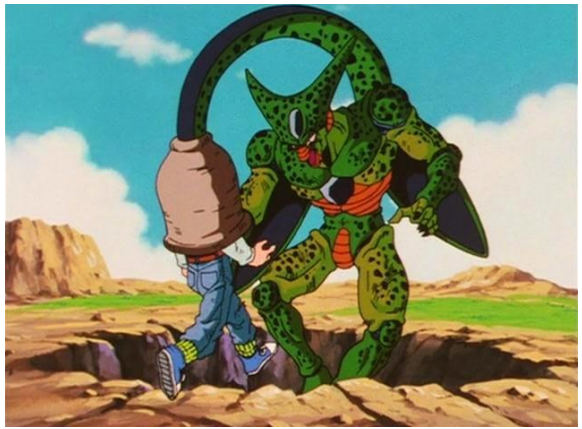
- Minimalist, highly-modular neural network library written in python
- Supports **TensorFlow**/Theano and CNTK
- **Easy and fast prototyping**
  - User-friendly
  - Modular
    - Pre-built layers, optimizers, etc..
  - Easy-extensibility
    - Also good for doing research!

- We will rely on **TensorFlow 2.0**



Keras and TensorFlow 2.0

- **Keras merged into TF 2.0 now**
  - **tf.keras**
  - Used in our Lab
  - Supported in **Colab**!
- We'll cover the **very basics**
- **Quickstart for experts**



It’s poll time!

Is anyone NOT familiar with NumPy + tensor indexing?

Tensors

- Generalization of the concept of vectors and matrices to **higher dimensional spaces**
- When using Keras, it is **fundamental** to know what tensor **shapes** you are working with!

Scalar
0-d tensor

$$0$$

Vector
1-d tensor

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Matrix
2-d tensor

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

Tensor
2x2x2 tensor

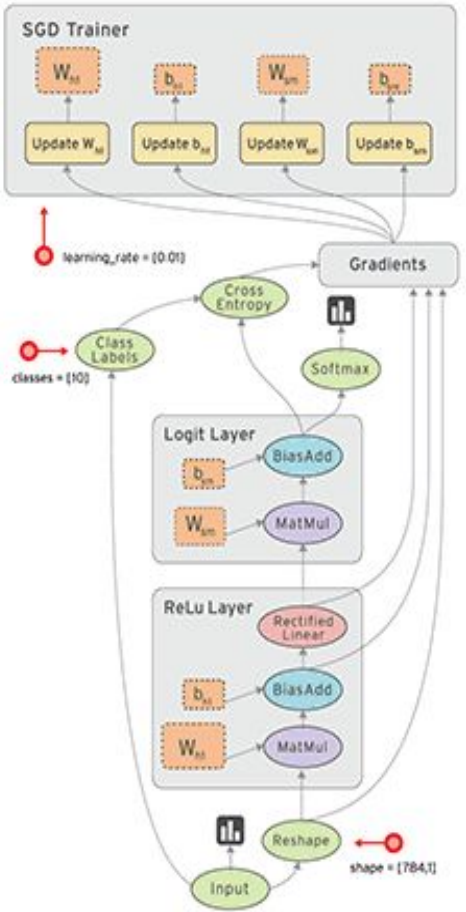
$$\left(\begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 3 & 4 \\ 7 & 8 \end{bmatrix}\right)$$

Indexing and Broadcasting

- Each dimension of a tensor can be **indexed** → sub-tensor
  - Usual square bracket notation: my_tensor[:10, : , 2:5]
  - You can filter on the basis of **boolean arrays**
    - my_tensor[: , **bool_filter**, :]
- Broadcasting allows you to forget about replicating data across dimensions
  - e.g., elem-wise multiplication between 100x10x32 and 100x1x32 tensors
  - **Always check the shape of your tensors**

Data-Flow Graph

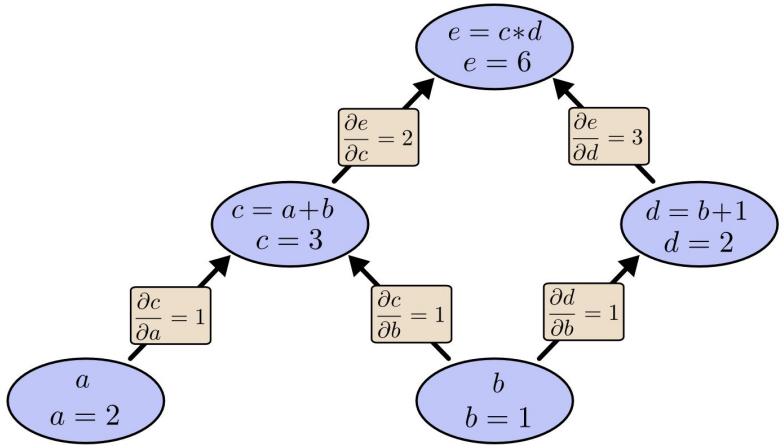
- Model for **parallel computing**
- Benefits:
  - Parallel/distributed execution
  - Portability
  - **Auto-differentiation (!)**
  - **Clear separation** model and logic
- Graphs can be static or dynamic
  - Lazy vs eager execution



Auto-differentiation!

$$e = (a+b) * (b+1)$$

$$\begin{cases} c=a+b \\ d=b+1 \\ e=c*d \end{cases}$$



Useful (Sub-)Packages

import tensorflow as tf

- Datasets → tf.keras.**datasets**
  - MNIST → tf.keras.datasets.mnist
- Data creation/management → **tf.data**
  - Dataset utilities → tf.data.Dataset
- Layers → tf.keras.**layers**
- Loss functions → tf.keras.**losses**
- Metrics → tf.keras.**metrics**
- Optimizers → tf.keras.**optimizers**
- Regularizers → tf.keras.**regularizers**
- Tensorboard → tf.keras.**callbacks**
- Save/Load → tf.keras.**callbacks**

Wait! I want to use a GPU!

```
import tensorflow as tf

try:
        # Specify a valid GPU device
        with tf.device('/device:GPU:0'):  # “with” ensures that GPU resources are freed
                a = tf.constant([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
                b = tf.constant([[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]])
                c = tf.matmul(a, b)
except RuntimeError as e:
        print(e)
```

Three API Styles

- **Sequential Model** (70+% of use cases)
  - **Dead simple!**
  - Only for single-input, single output, sequential layer stacks

- **Functional API** (95% of use cases)
  - **Functions of functions!**
  - Multi-input multi-output arbitrary graph topologies

- **Model subclassing**
  - **Maximum flexibility!**

Three API Styles

```
import keras
from keras import layers
# Sequential
model = keras.Sequential()
model.add(layers.Dense(20, activation='relu', input_shape=(10, )))
model.add(layers.Dense(20, activation='relu')
model.add(layers.Dense(10, activation='softmax'))

# Functional
inputs = keras.Input(shape=(10,))
x = layers.Dense(20, activation='relu')(inputs)
x = layers.Dense(20, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)
model = keras.Model(inputs, outputs)
```

Three API Styles

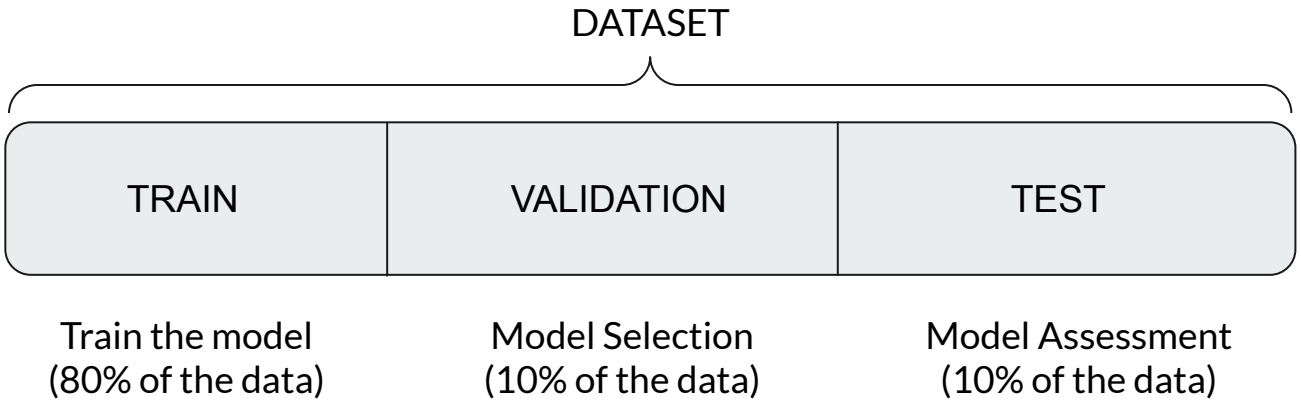
```
# Model subclassing
class MyModel(keras.Model):
     def __init__(self):
          super(MyModel, self).__init__()
          self.dense1 = layers.Dense(20, activation='relu')
          self.dense2 = layers.Dense(20, activation='relu')
          self.dense3 = layers.Dense(10, activation='softmax')

     def call(self, inputs):
          x = self.dense1(inputs)
          x = self.dense2(x)
          return self.dense3(x)
model = MyModel()
model.fit(x, y, epochs=10, batch_size=32)
```

Remember:

Use the right tool (API) for the right job!

Split your data 101



Called **Hold-Out** Technique

Model Selection

- Process that finds the **“best”** hyper-parameters configuration for your model using the **VALIDATION** set
- “Best” according to some performance metric
- Two possible ways to do that:
    - **Grid Search:** Define possible values for each hyper-parameter and try all possible configuration
    - **Random Search:** fix range of value for each hyper-parameters and try several random configurations.

Golden Rule (a MUST)

Never

ever

EVER

(do a PhD)

USE THE TEST SET FOR MODEL SELECTION

THE TEST SET IS USED ONLY ONCE!
YOU CANNOT REPEAT THE EXPERIMENT
IF YOU "DO NOT LIKE" THE TEST RESULTS!

Seriously..

1) That makes the difference between making *your boss* 😡 (when things **do not work "as expected" in production**) or 😇

2) Bringing ***biased*** results to the table does ***not*** help anyone

3) The test set is the "oracle" of your model.. you do not want to kill the oracle because you don't like the answer.

4) Once you have your answer….

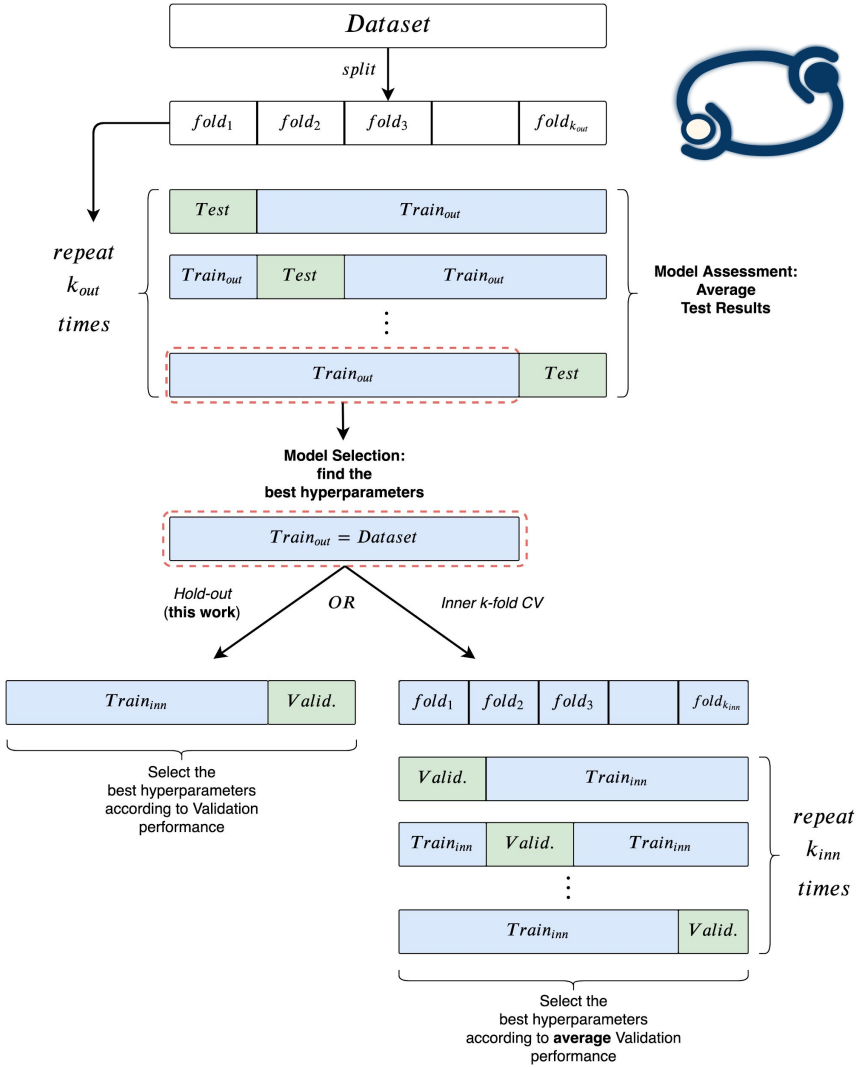
More complex splits

- **External** K-fold Cross Validation
  - For Model **Assessment**

- **Internal** K-fold Cross Validation
  - For Model **Selection**

or..

- **Internal** Hold-out train/validation split
  - For Model **Selection**

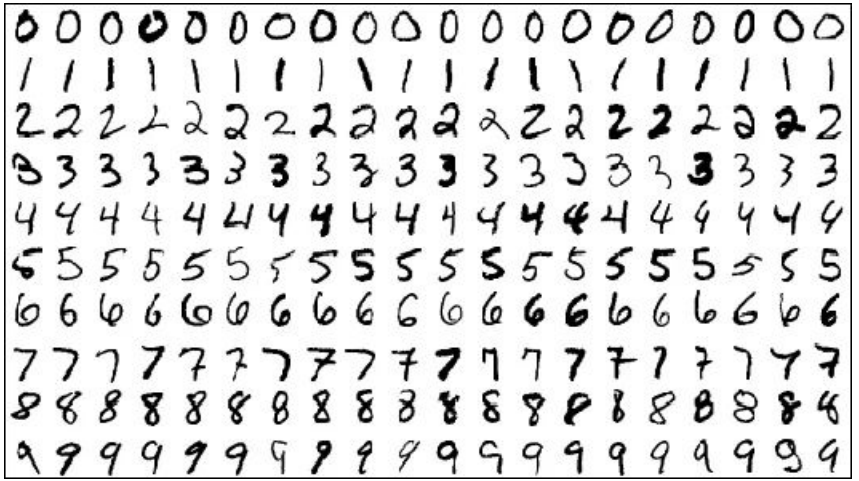


Shall we start training our machine? ;)

Hands-on!

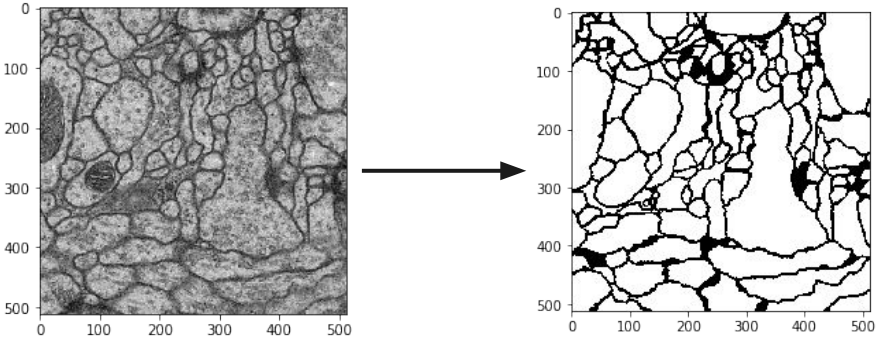
Our data: MNIST

- ML “Hello World” problem
- Labeled handwritten digits dataset
- **Goal**: obtain better and better performance on the task with models of increasing complexity



Our data: Membrane

- Used for Image Segmentation
- **Goal**: train a Convolutional Network to do image segmentation!



10-minute break?

Upcoming: **Coding Lab Practice**

Questions?

References

Keras Documentation:

1. https://www.tensorflow.org/api_docs/python/tf/keras
2. https://keras.io/guides/

   Again: We will use the Keras library inside TensorFlow 2.0.

Keras Tutorials: https://www.tensorflow.org/guide/keras

Colab: https://colab.research.google.com/

Let's open a Notebook / Colab

Just type **jupyter notebook** in your terminal
(with the environment activated)

and create a **Python3 notebook** using the "New" button,
or open one of the notebooks in the repo

- **Interactive** execution of Python code
- **Alternative: open the Github Lessons in Colab**