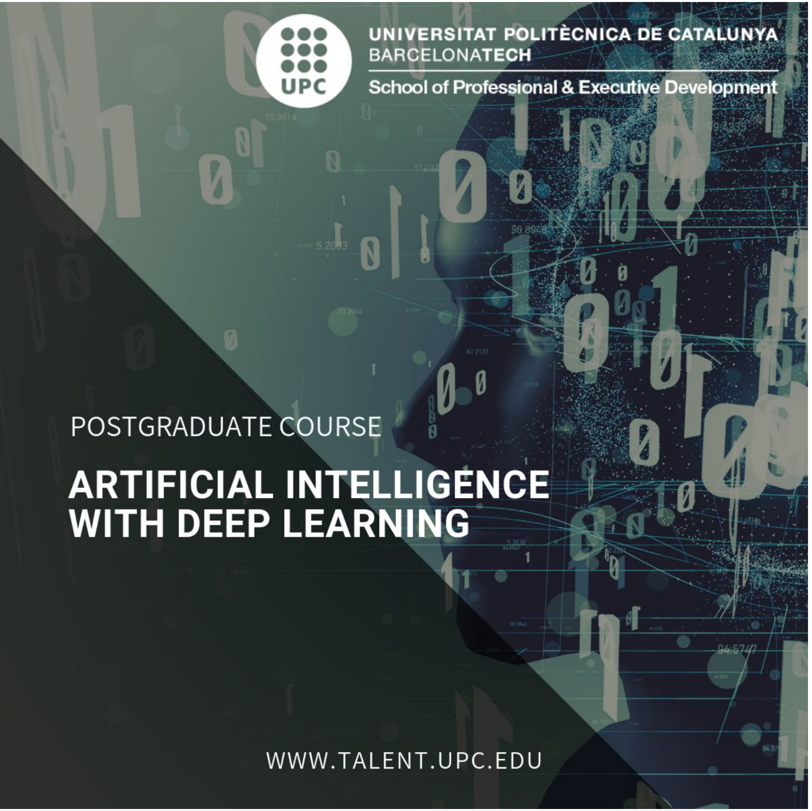


#DLUPC

Day 2 Lab 1

Linear Regression

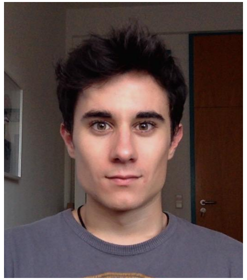
Xavier Giro-i-Nieto
xavier.giro@upc.edu

Associate Professor
Universitat Politècnica de Catalunya
Technical University of Catalonia

Daniel Fojo
dani.fojo@gmail.com

Research Engineer
Disney Research

Lab & Slides by



[Víctor Campos]

victor.campos@bsc.es

PhD Candidate

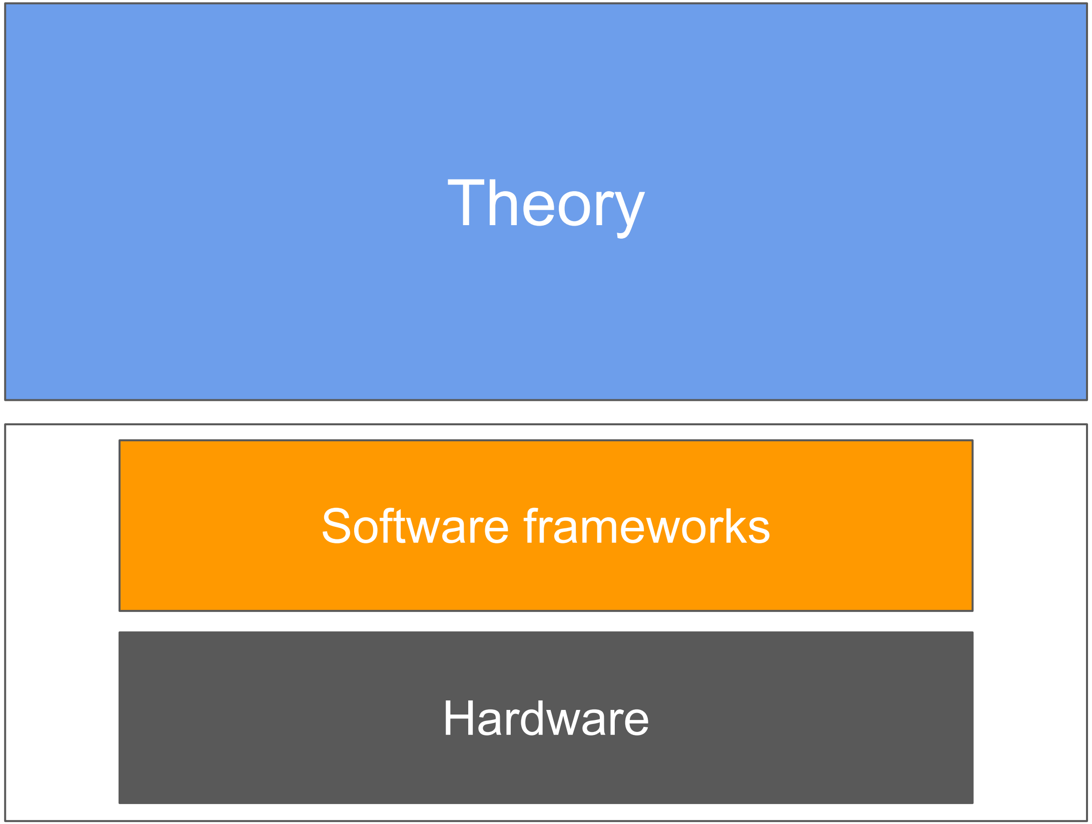
Barcelona Supercomputing Center

Today’s objective

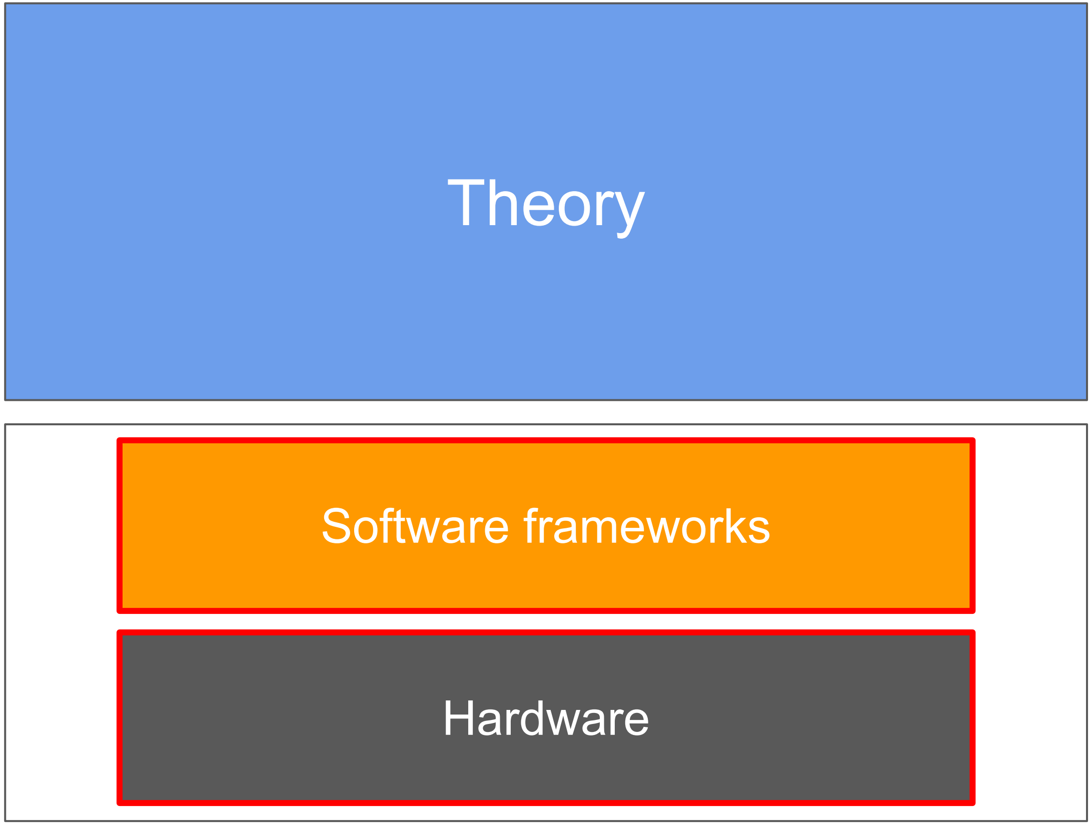
Theory

Practice

Today’s objective



Today's objective



Static vs dynamic graphs

There are many deep learning frameworks

TensorFlow

Caffe2

theano

Chainer

PYTORCH

K

mxnet

PaddlePaddle

Static vs dynamic graphs

We can classify frameworks in two main families

- Those based on static graphs follow a Define-**and**-Run strategy
- Those based on dynamic graphs follow a Define-**by**-Run strategy

Static vs dynamic graphs

Static graphs

- Two steps
  1. Define the complete graph
  2. Feed data and run the graph as many times as needed
- Some behaviors are difficult to implement, like graphs that vary depending on intermediate results
- It allows for aggressive optimizations that reduce memory and wall-clock time

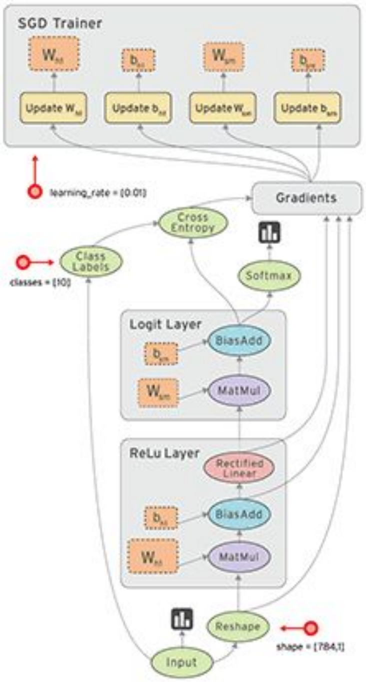
Static vs dynamic graphs

Dynamic graphs

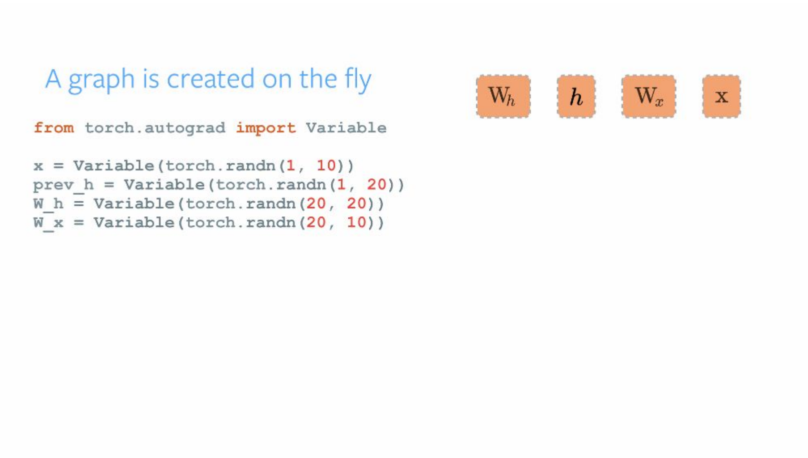
- Each element in the graph is created and run as soon as it is defined
- The following operations can depend on the latest results. This flexibility is helpful for if/else statements and loops.
- It allows for fewer optimizations than static graphs

Static vs dynamic graphs

Static (TensorFlow)



Dynamic (PyTorch)



Outline

1. Motivation
2. Static vs dynamic graphs
3. **Introduction to TensorFlow**
4. Introduction to Keras

Introduction to TensorFlow (TF)

Summary

- Deep learning framework developed by Google
- It uses static graphs
- It is important to distinguish between the graph definition and evaluation phases
- It is relatively low level, which makes it flexible… but some higher level APIs like Keras are useful for faster prototyping

Introduction to TensorFlow (TF)

Steps

1. **Define** the graph, including its inputs and operations. This will not use any resources nor instantiate any variables.

```
import tensorflow as tf

# Define a placeholder that expects a vector of three floating-point values
x = tf.placeholder(tf.float32, shape=[3])

# Define a constant scalar value that scales the input
c = tf.constant(2.)

# Define an operation on both x and c
y = tf.square(c * x)
```

Introduction to TensorFlow (TF)

TensorFlow

Steps

2. Create a **session**, which will be assigned resources to run the graph (CPU, GPU, memory)

3. Initialize the variables in the graph

```
with tf.Session() as sess:
    # Initialize all variables (in this case, only c)
    sess.run(tf.global_variables_initializer())
```

Introduction to TensorFlow (TF)

Steps

4. **Run** the graph as many times as needed

```
# Feeding a value changes the result that is returned when you evaluate `y`.
print(sess.run(y, {x: [1.0, 2.0, 3.0]}))  # => "[4.0, 16.0, 36.0]"
print(sess.run(y, {x: [0.0, 0.0, 5.0]}))  # => "[0.0, 0.0, 100.0]"
```

Introduction to TensorFlow (TF)

TensorFlow

Steps

1. **Define** the graph, including its inputs and operations. This will not use any resources nor instantiate any variables.

2. Create a **session**, which will be assigned resources to run the graph (CPU, GPU, memory)

3. **Initialize** the variables in the graph

4. **Run** the graph as many times as needed

Introduction to TensorFlow (TF)

TensorFlow

Small example

```python
import tensorflow as tf

# Define a placeholder that expects a vector of three floating-point values
x = tf.placeholder(tf.float32, shape=[3])

# Define a constant scalar value that scales the input
c = tf.constant(2.)

# Define an operation on both x and c
y = tf.square(c * x)

with tf.Session() as sess:
    # Initialize all variables (in this case, only c)
    sess.run(tf.global_variables_initializer())

    # Feeding a value changes the result that is returned when you evaluate `y`.
    print(sess.run(y, {x: [1.0, 2.0, 3.0]}))  # => "[4.0, 16.0, 36.0]"
    print(sess.run(y, {x: [0.0, 0.0, 5.0]}))  # => "[0.0, 0.0, 100.0]"
```

Outline

1. Motivation
2. Static vs dynamic graphs
3. Introduction to TensorFlow
4. **Introduction to Keras**

Introduction to Keras

High level API created by François Chollet (Google Brain) to simplify the definition of deep learning models

NIPS 2016

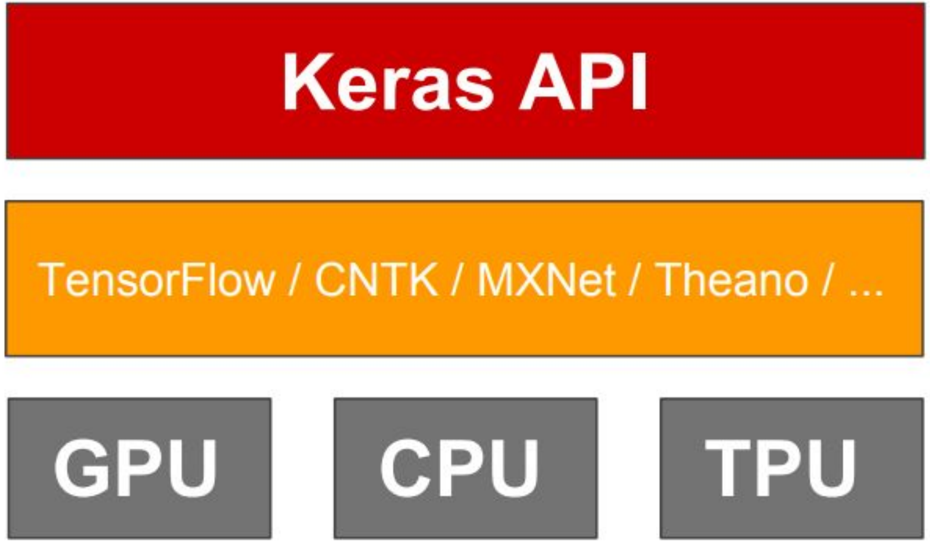
CVPR 2017

Introduction to Keras

Summary

- It originally worked on top of Theano and TensorFlow, but it was merged into the core of TensorFlow
- It is less flexible than using lower level frameworks. If more flexibility is needed, pieces of TensorFlow code can be used.

Summary



Slide credit: François Chollet

Introduction to Keras

Steps

1. **Define** the model by specifying a sequence of high level operations (e.g. convolutions, poolings)

2. **Compile** the model

3. **Fit** the model to minimize some loss function

Introduction to Keras

Three API styles

- The Sequential Model
  - Dead simple
  - Only for single-input, single-output, sequential layer stacks
  - Good for 70+% of use cases
- The functional API
  - Like playing with Lego bricks
  - Multi-input, multi-output, arbitrary static graph topologies
  - Good for 95% of use cases
- Model subclassing
  - Maximum flexibility
  - Larger potential error surface

Slide credit: François Chollet [source]

Introduction to Keras

The Sequential API

```python
import keras
from keras import layers

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'))

model.fit(x, y, epochs=10, batch_size=32)
```

Slide credit: François Chollet [source]

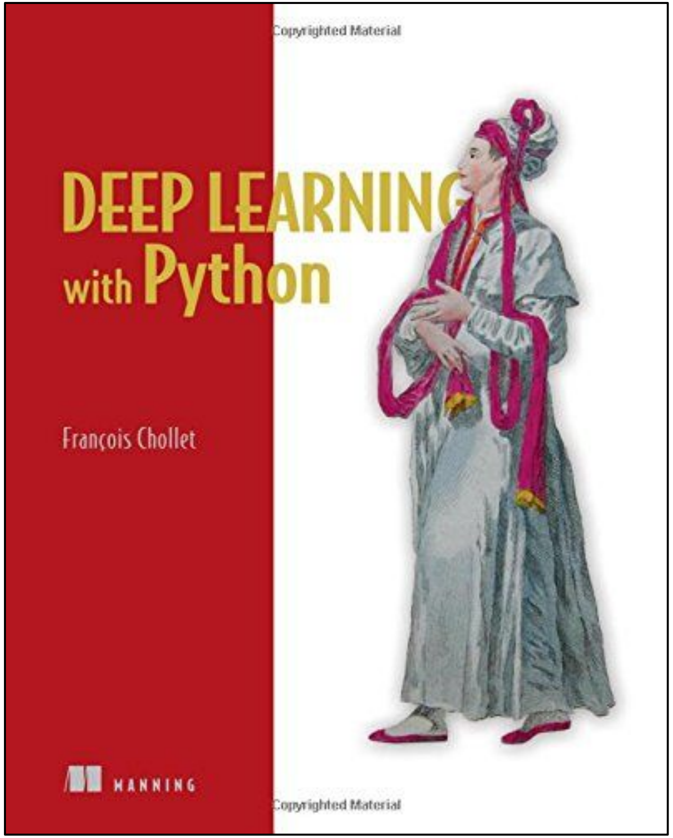
Introduction to Keras

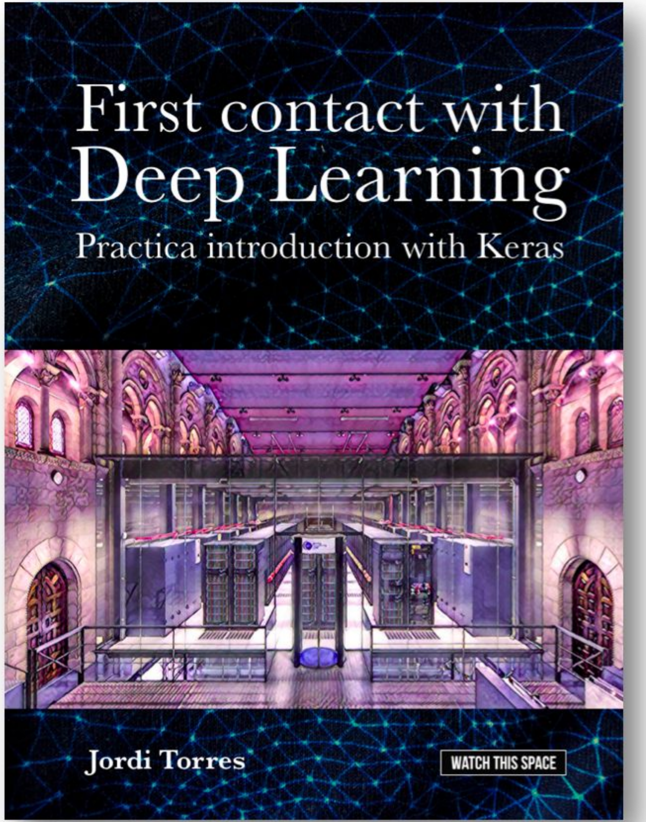
The Functional API

```
import keras
from keras import layers

inputs = keras.Input(shape=(10,))
x = layers.Dense(20, activation='relu')(x)
x = layers.Dense(20, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)

model = keras.Model(inputs, outputs)
model.fit(x, y, epochs=10, batch_size=32)
```


Slide credit: François Chollet [source]

Recommended books





Outline

1. **Linear regression**

2. Introduction to Google Colab

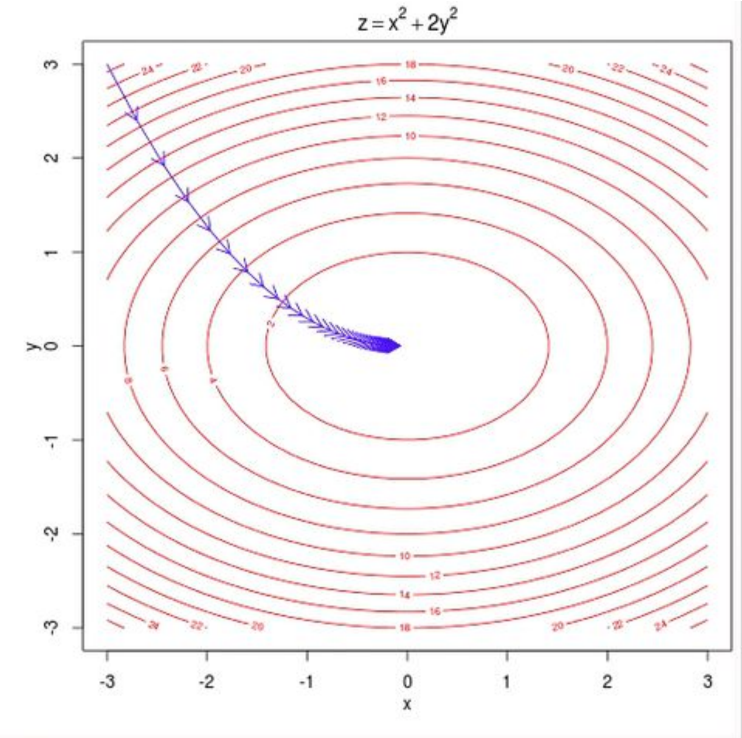
Linear regression

- We are given a set of (x, y) tuples
- We want to approximate $\hat{y} = W \cdot x + b$
- We will fit the model parameters, W and b, by minimizing the Mean Squared Error between the predictions and the real values:

$$MSE = |y - \hat{y}|^2 = |y - W \cdot x - b|^2$$

- The parameters will be updated using a variant of Stochastic Gradient Descent

Stochastic Gradient Descent

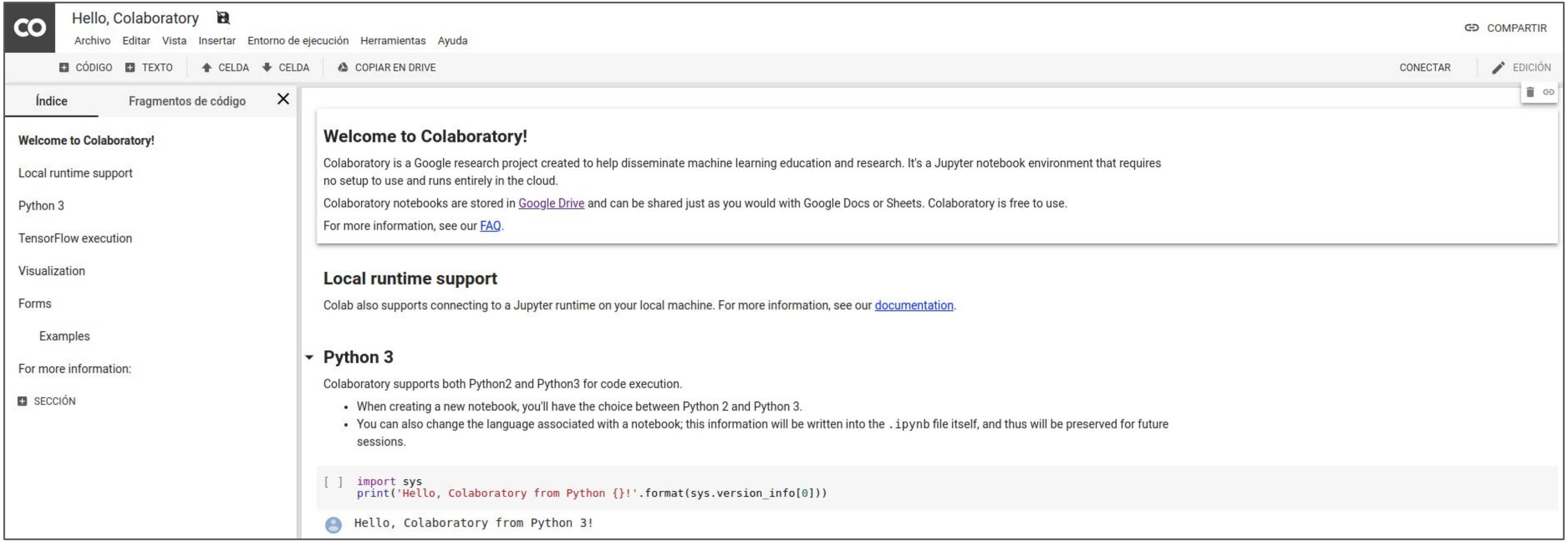


Outline

1. Linear regression

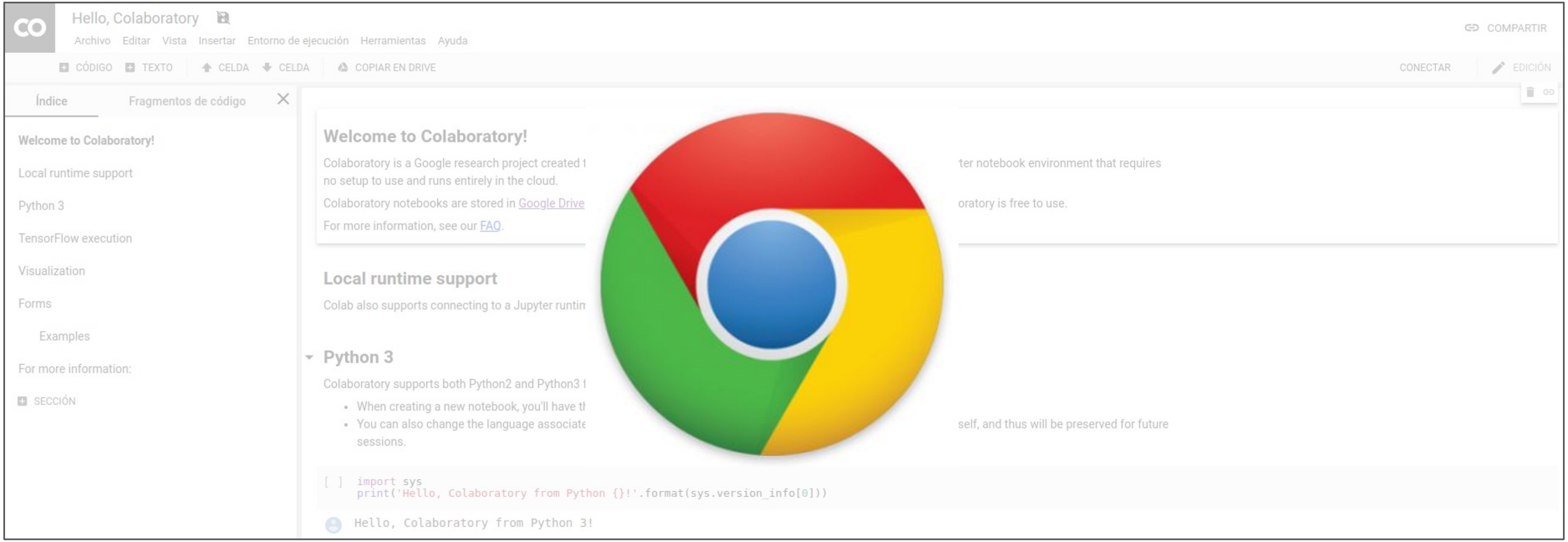
2. **Introduction to Google Colab**

Google Colab



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

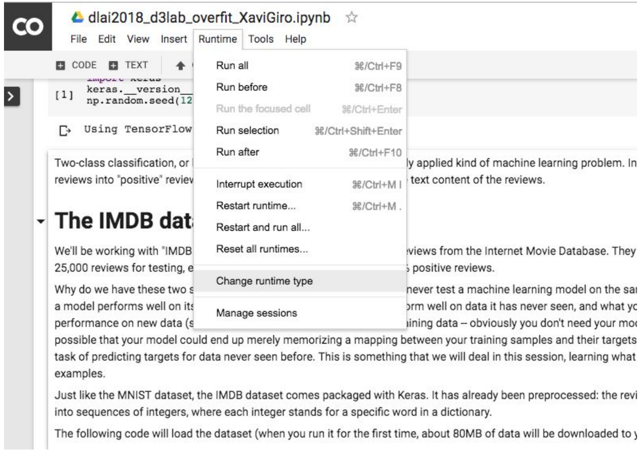
Google Colab

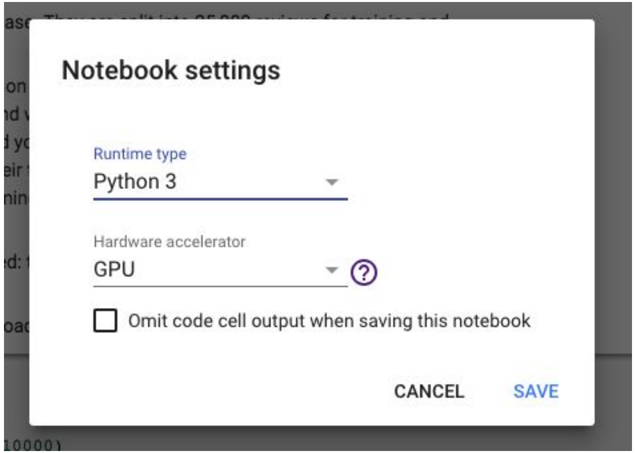


https://colab.research.google.com/

Google Colab

1. Download the two notebooks (TensorFlow & Keras) of this lab session
2. Login to a Google account: yours or aidlupc2019@gmail.com (talentcenter)
3. Copy/move it to your Google drive folder
4. From there, open it with Colab
5. Change runtime type to work with GPU! Your trainings will be much faster :)





Code comments

1. MSE loss is defined with ½ factor

```
# Mean squared error
cost = tf.reduce_sum(tf.pow(pred-Y, 2))/(2*n_samples)
```

```
# Fit all training data
for epoch in range(100):
    for (x, y) in zip(train_X, train_Y):
        sess.run(optimizer, feed_dict={X: x, Y: y})
```

Code comments

1. Python Zip allows simultaneous operations between lists.

```
# Fit all training data
for epoch in range(100):
    for (x, y) in zip(train_X, train_Y):
        sess.run(optimizer, feed_dict={X: x, Y: y})
```

Final Questions