

#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

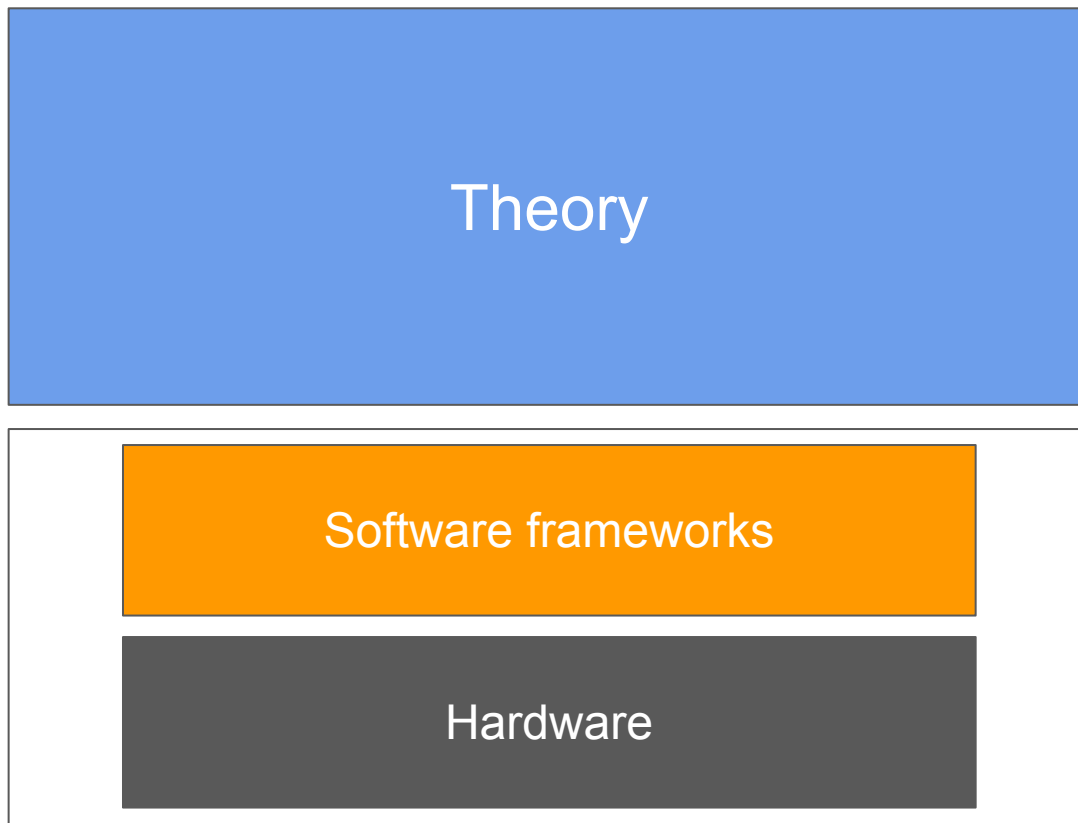


The diagram consists of two stacked rectangular boxes. The top box is blue and contains the word 'Theory' in white. The bottom box is yellow and contains the word 'Practice' in white. Both boxes have a thin black border.

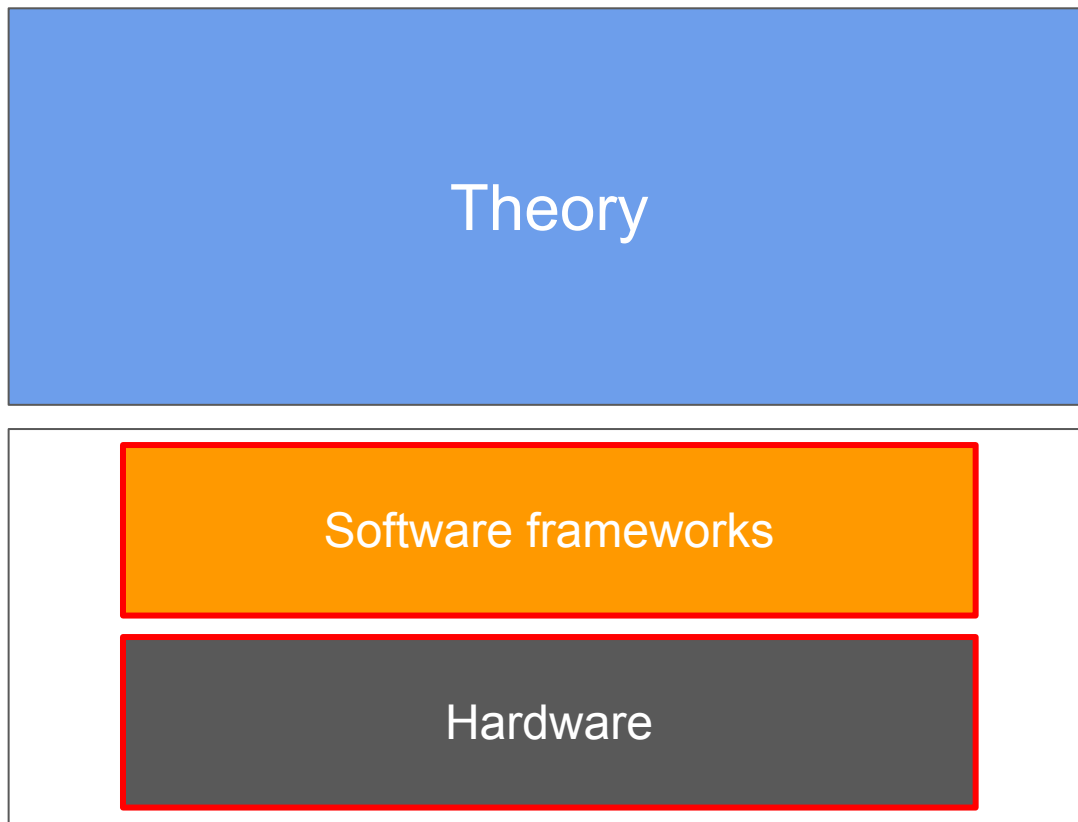
Theory

Practice

# Today's objective



# Today's objective



# Static vs dynamic graphs

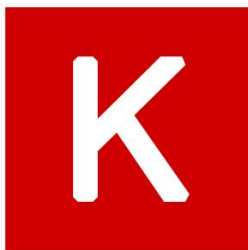
There are many deep learning frameworks



theano



PYTORCH



# 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



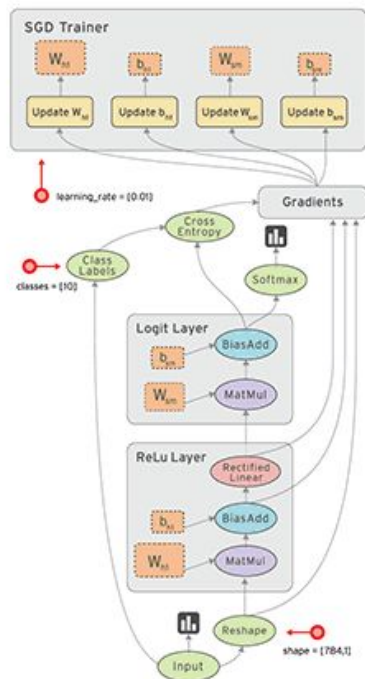
# Static vs dynamic graphs

## 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)

A graph is created on the fly

```
from torch.autograd import Variable
```

```
x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))
```



# 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

## Steps

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

```
1 import tensorflow as tf
2
3 # Define a placeholder that expects a vector of three floating-point values
4 x = tf.placeholder(tf.float32, shape=[3])
5
6 # Define a constant scalar value that scales the input
7 c = tf.constant(2.)
8
9 # Define an operation on both x and c
10 y = tf.square(c * x)
```

# Introduction to TensorFlow (TF)



## Steps

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

```
12 with tf.Session() as sess:  
13     # Initialize all variables (in this case, only c)  
14     sess.run(tf.global_variables_initializer())  
15
```

# Introduction to TensorFlow (TF)



## Steps

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

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

# Introduction to TensorFlow (TF)



## 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



## Small example

```
1 import tensorflow as tf
2
3 # Define a placeholder that expects a vector of three floating-point values
4 x = tf.placeholder(tf.float32, shape=[3])
5
6 # Define a constant scalar value that scales the input
7 c = tf.constant(2.)
8
9 # Define an operation on both x and c
10 y = tf.square(c * x)
11
12 with tf.Session() as sess:
13     # Initialize all variables (in this case, only c)
14     sess.run(tf.global_variables_initializer())
15
16     # Feeding a value changes the result that is returned when you evaluate `y`.
17     print(sess.run(y, {x: [1.0, 2.0, 3.0]})) # => "[4.0, 16.0, 36.0]"
18     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](#)

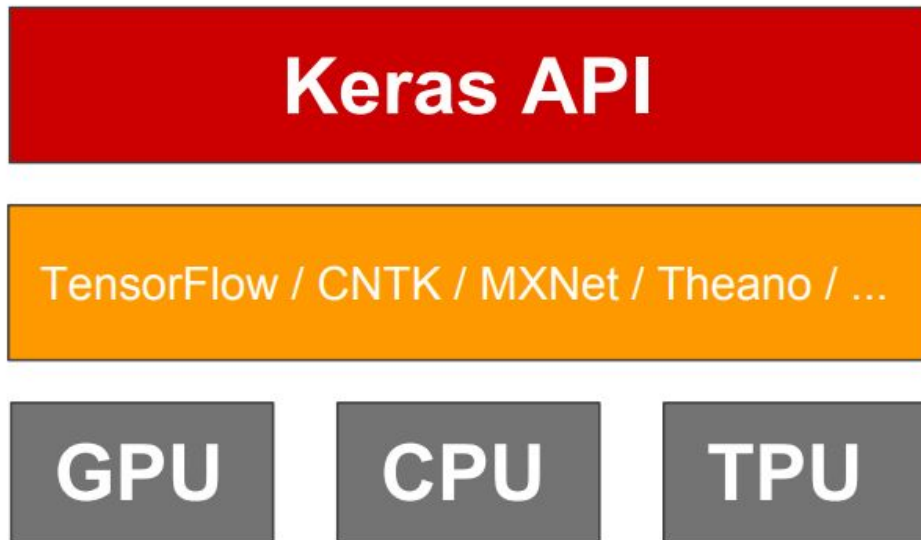


## Summary

- 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



## 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



## 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



## The Sequential API

```
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)
```





## The Functional API

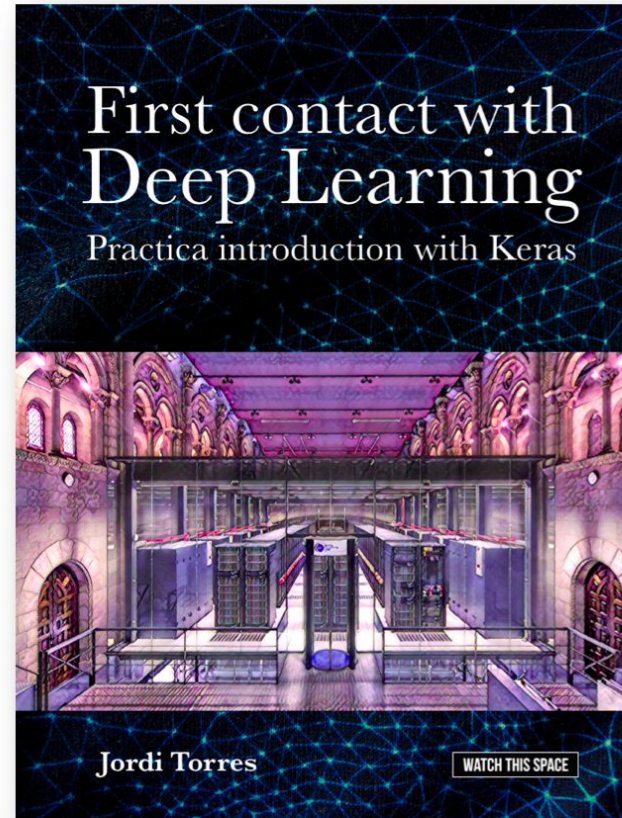
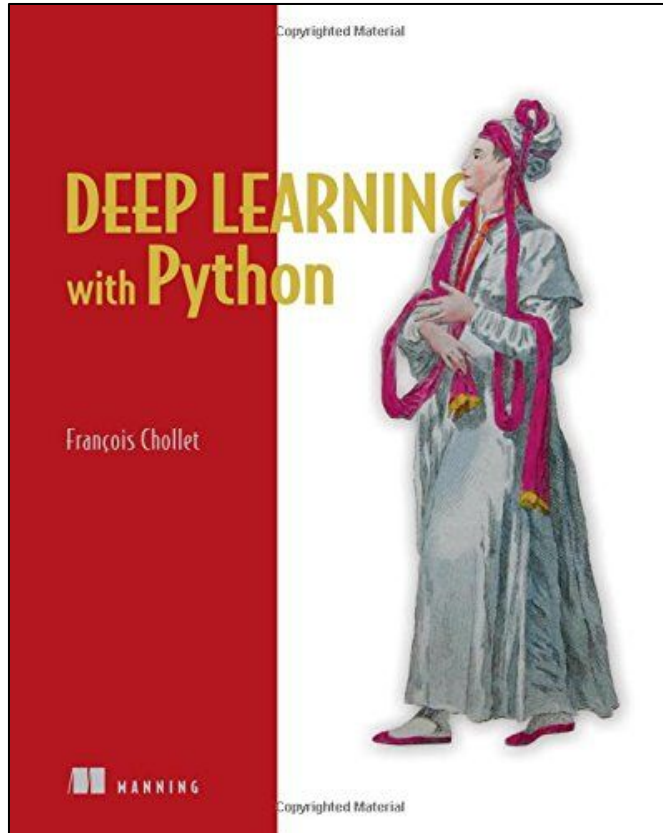
```
import keras
from keras import layers

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)
```

# Recommended books

K





# Outline

1. **Linear regression**
2. Introduction to Google Colab

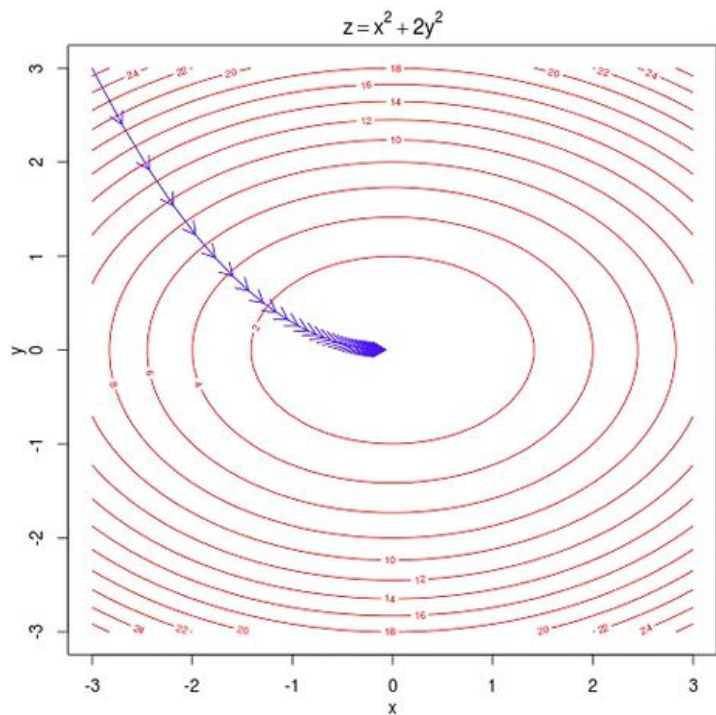
# Linear regression

- 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

The screenshot shows the Google Colaboratory web interface. At the top, there's a header with the Google Colab logo and the text "Hello, Colaboratory". Below this is a navigation bar with options like "Archivo", "Editar", "Vista", "Insertar", "Entorno de ejecución", "Herramientas", and "Ayuda". On the right side of the header, there are buttons for "COMPARTIR", "CONECTAR", and "EDICIÓN".

The main content area is divided into two parts. On the left, there's a sidebar with a "Índice" (Index) section containing links to "Welcome to Colaboratory!", "Local runtime support", "Python 3", "TensorFlow execution", "Visualization", "Forms", and "Examples". Below this is a "SECCIÓN" (Section) button.

The right part of the main content area displays the "Welcome to Colaboratory!" message. It states that Colaboratory is a Google research project for disseminating machine learning education and research, and that it's a Jupyter notebook environment. It also mentions that notebooks are stored in Google Drive and can be shared. Below this, there's a section for "Local runtime support" and a section for "Python 3" which lists two bullet points about choosing between Python 2 and Python 3, and changing the language associated with a notebook.

At the bottom of the main content area, there's a code cell with the following code:

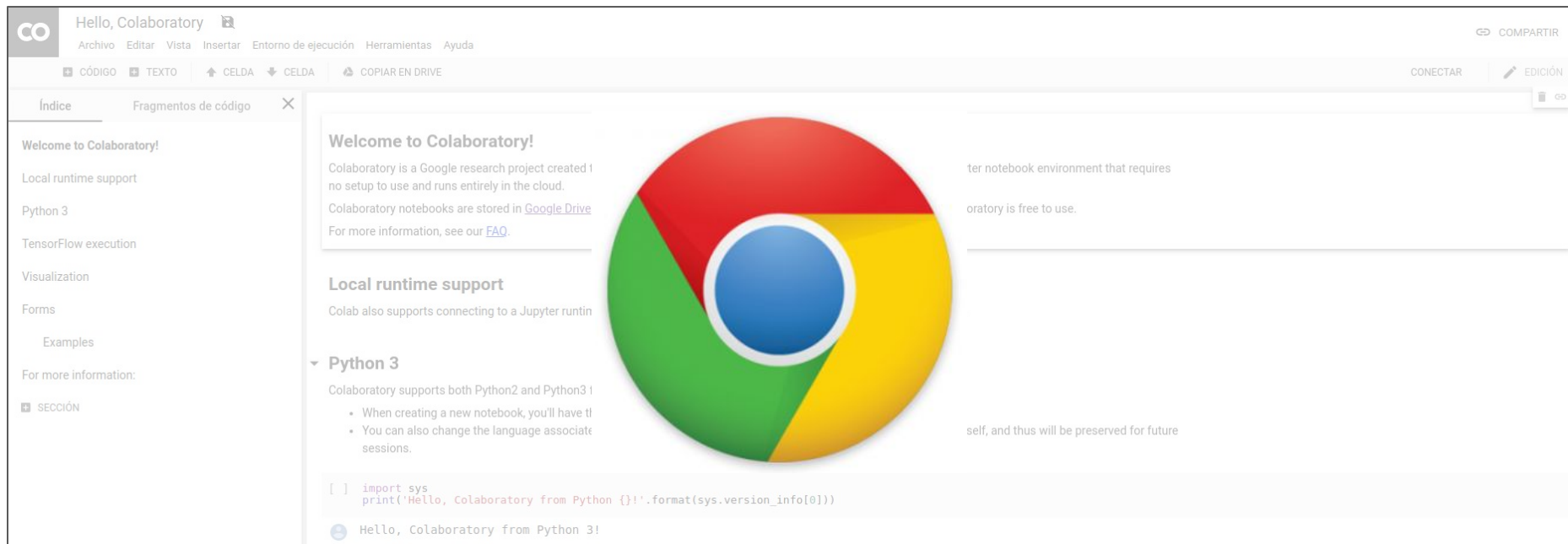
```
[ ] import sys
print('Hello, Colaboratory from Python {}'.format(sys.version_info[0]))
```

Below the code cell, the output is displayed: "Hello, Colaboratory from Python 3!".

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



# Google Colab



The screenshot displays the Google Colaboratory web interface. At the top, the header includes the 'co' logo, the title 'Hello, Colaboratory', and a menu with options: Archivo, Editar, Vista, Insertar, Entorno de ejecución, Herramientas, and Ayuda. On the right, there are links for 'COMPARTIR' and 'CONECTAR', and a button for 'EDICIÓN'. Below the header, a toolbar shows icons for 'CÓDIGO', 'TEXTO', 'CELDA', and 'COPIAR EN DRIVE'. The left sidebar contains a table of contents with links to 'Índice', 'Fragmentos de código', 'Welcome to Colaboratory!', 'Local runtime support', 'Python 3', 'TensorFlow execution', 'Visualization', 'Forms', 'Examples', and 'For more information:'. The main content area features a large, colorful circular logo in the center. To the left of the logo, the 'Welcome to Colaboratory!' section explains that Colaboratory is a Google research project that runs entirely in the cloud and that notebooks are stored in Google Drive. Below this, the 'Local runtime support' section mentions connecting to a Jupyter runtime. The 'Python 3' section lists two bullet points: 'When creating a new notebook, you'll have to...' and 'You can also change the language associated with the notebook for future sessions.' To the right of the logo, there is a text box that says 'ter notebook environment that requires' and 'oratory is free to use.' At the bottom, a code cell is shown with the following Python code: 

```
[ ] import sys
print('Hello, Colaboratory from Python {}'.format(sys.version_info[0]))
```

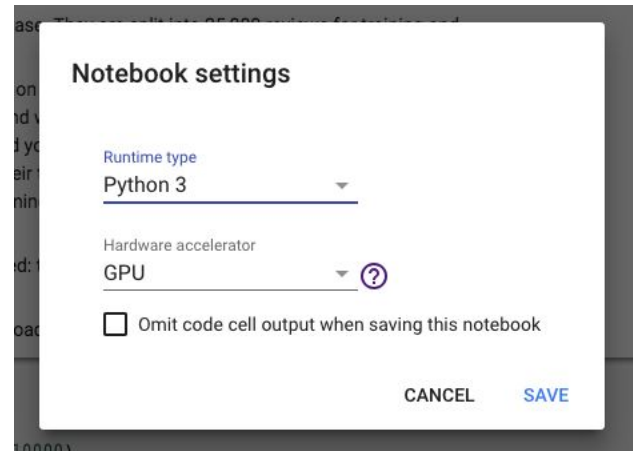
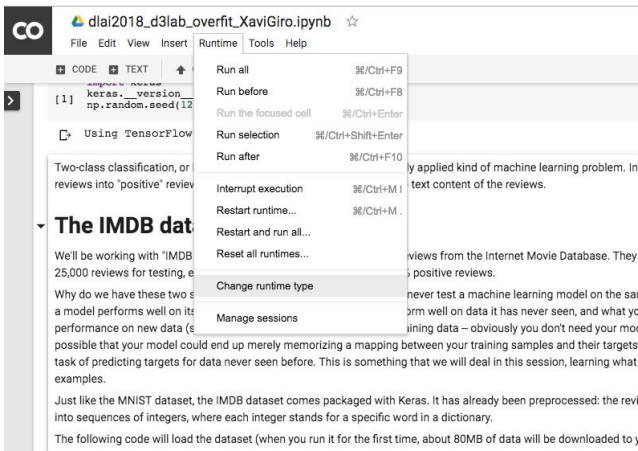
 Below the code, a small icon and the text 'Hello, Colaboratory from Python 3!' are visible.

<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](mailto: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  $\frac{1}{2}$  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

## Undergradese

What undergrads ask vs. what they're REALLY asking

"Is it going to be an open book exam?"

Translation: "I don't have to actually memorize anything, do I?"

"Hmm, what do you mean by that?"

Translation: "What's the answer so we can all go home."

"Are you going to have office hours today?"

Translation: "Can I do my homework in your office?"

"Can i get an extension?"

Translation: "Can you re-arrange your life around mine?"

"Is this going to be on the test?"

Translation: "Tell us what's going to be on the test."

"Is grading going to be curved?"

Translation: "Can I do a mediocre job and still get an A?"

JORGE CHAM © 2008



WWW.PHDCOMICS.COM