Artificial Intelligence

Artificial Neurons
Hands-on

POLITÉCNICO DE LEIRIA

Slides by Carlos Grilo & Rolando Miragaia

Keras

Keras (https://keras.io)

- Keras is a deep-learning framework for Python that provides a way to define and train almost any kind of deep-learning model

- Keras was initially developed for researchers, with the aim of enabling fast experimentation

Keras features

- It allows the same code to run seamlessly on CPU or GPU
- It has a user-friendly API that makes it easy to quickly prototype deep-learning models
- It has built-in support for convolutional networks, recurrent networks (for sequence processing), and any combination of both
- It supports arbitrary network architectures: multi-input or multi-output models, and so on

Keras stack

- Keras is a model-level library, providing high-level building blocks for developing deep-learning models
- It doesn’t handle low-level operations such as tensor manipulation and differentiation
- Instead, it runs on top of either the JAX, Tensorfow or PyTorch frameworks

Hands-on

Creating a Jupyter notebook on VS Code

- File -> New File -> Jupyter Notebook
- Name the notebook as
  `02_artificial_neurons_01_AND.ipynb`

Google Colab

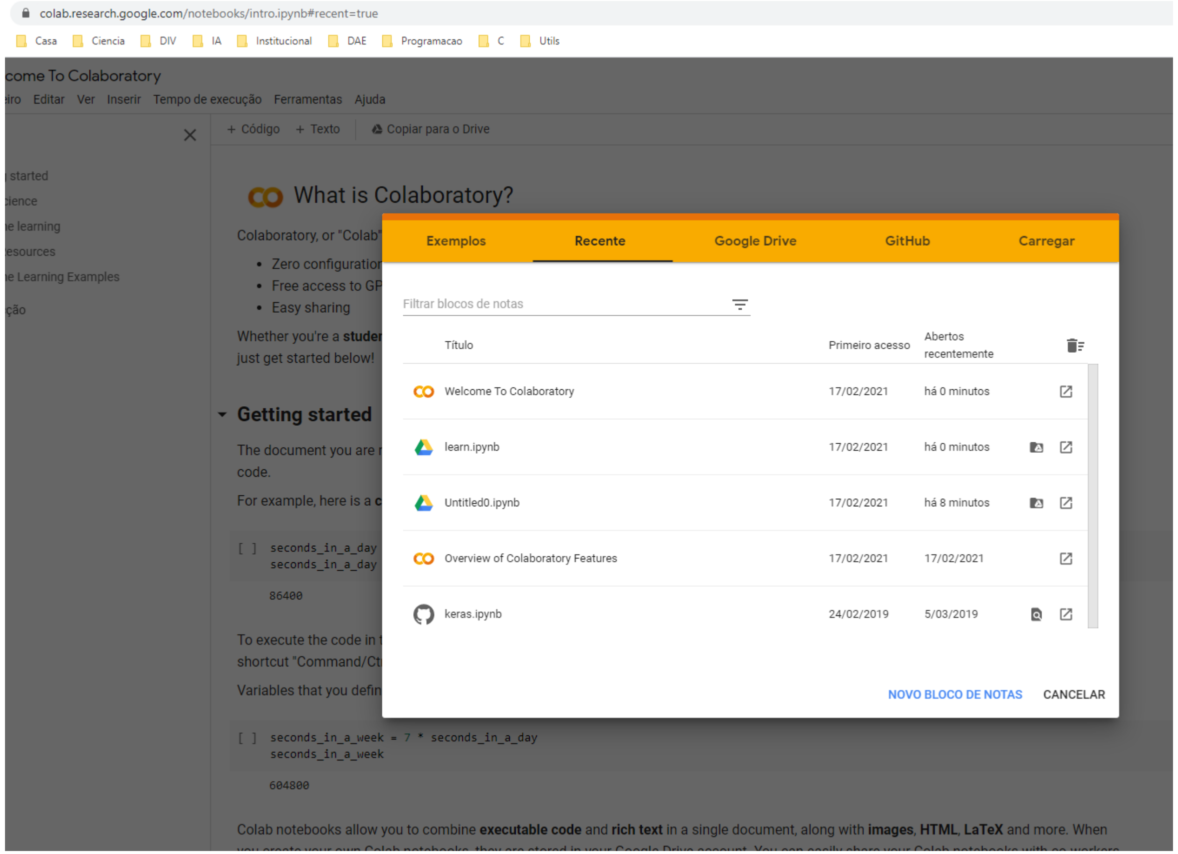
- Google Colab allows us to write and execute arbitrary Python code through the browser, and is especially well suited to machine learning, data analysis and education
- It is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs

- https://research.google.com/colaboratory/faq.html

Google Colab – creating a notebook

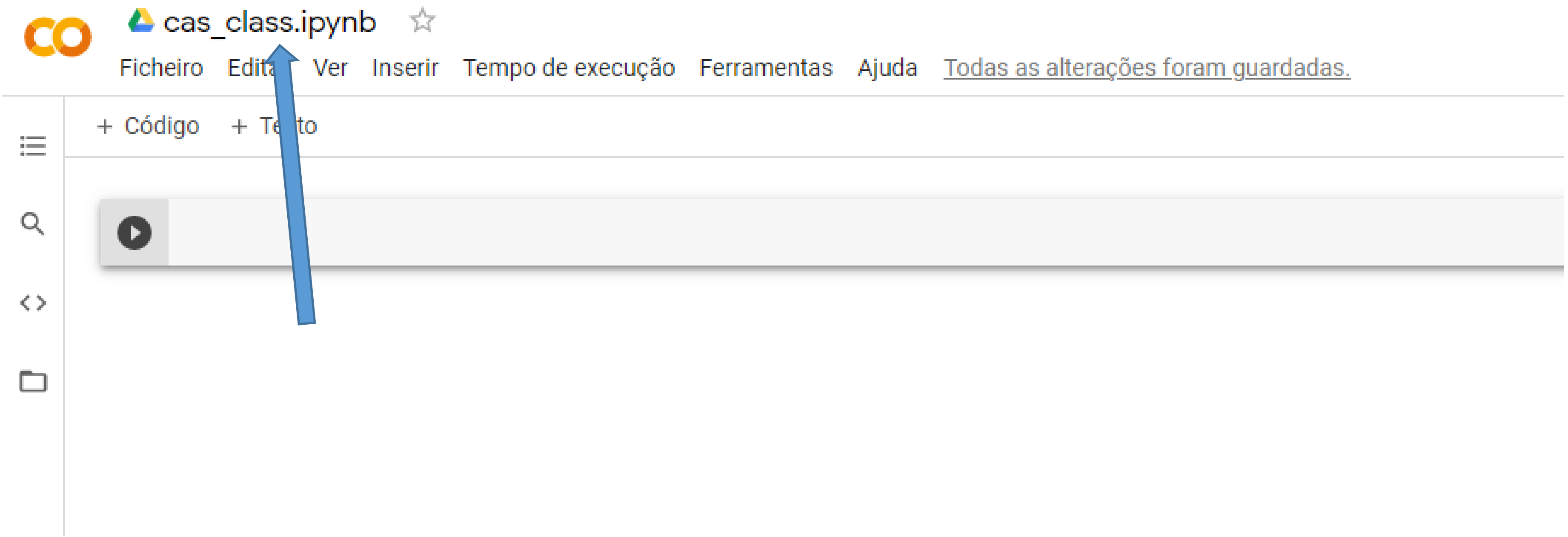
- Go to Google Colab (https://colab.research.google.com) using your Google account

Click on
"NOVO BLOCO DE NOTAS" (PT)
or
"NEW NOTEBOOK" (EN)



Changing the name of the notebook

- Change the name of the notebook to
  `02_artificial_neurons_01_AND.ipynb`



- We are now ready to start working…

AND logic function

- We will start by exemplifying the creation and training of a one-unit neural network that should work as an AND logic function

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Importing classes

- First, we need to import the libraries and classes we will use:

```
import tensorflow as tf
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
```

- tensorflow is a machine learning library
- numpy is a math library
- "… `as something`" allows us to use a shorter name

Defining the dataset

```
# Defining the dataset
training_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], "float32")
target_data = np.array([[0], [0], [0], [1]], "float32")
```

- training_data: array containing all possible input vectors
- target_data: array containing all target values for each input vector

Defining the model

```
# Defining the model
model = Sequential()
model.add(Dense(1, input_dim=2, activation='sigmoid'))
```

One unit

Two inputs

Activation function

Just one layer

- Sequential(): our network will be a sequence of neuron layers (just 1, in this case)
- Dense(…) -> defines the type of layer. More on this later

Defining the model

```python
# Defining the model
model = Sequential()
model.add(Dense(1, input_dim=2, activation='sigmoid'))
```

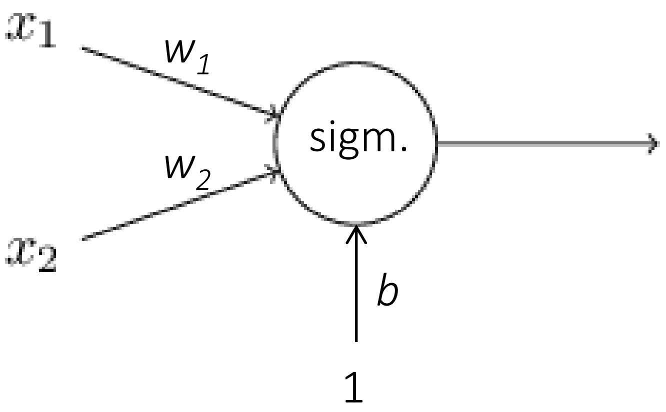
One unit

Two inputs

Activation function

Just one layer

Our model is something like this



Compiling the model

```
#Compile the model
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.25),
              loss='MSE',
              metrics=['accuracy'])
```

- loss: how the network will be able to measure its performance on the training data, and thus how it will be able to steer itself in the right direction
- optimizer: the mechanism through which the network will update itself based on the data it sees and its loss function

$$MSE = \frac{1}{n}\sum_x (\underset{\text{desired}}{y(x)} - \underset{\text{predicted}}{a})^2$$

- metrics: the metric used to assess the network's performance
  - accuracy: the fraction of the samples that were correctly classified

Training the model

```
# Training the model on the dataset
model.fit(training_data, target_data, epochs=100, batch_size=1)
```

```
Epoch 1/100 4/4 [==============================] - 0s 4ms/step - loss: 0.2902 - accuracy: 0.7500
Epoch 2/100 4/4 [==============================] - 0s 3ms/step - loss: 0.2839 - accuracy: 0.7500
Epoch 3/100 4/4 [==============================] - 0s 3ms/step - loss: 0.2771 - accuracy: 0.7500
Epoch 4/100 4/4 [==============================] - 0s 3ms/step - loss: 0.2720 - accuracy: 0.7500
…
Epoch 97/100 4/4 [==============================] - 0s 3ms/step - loss: 0.0722 - accuracy: 1.0000
Epoch 98/100 4/4 [==============================] - 0s 3ms/step - loss: 0.0715 - accuracy: 1.0000
Epoch 99/100 4/4 [==============================] - 0s 3ms/step - loss: 0.0708 - accuracy: 1.0000
Epoch 100/100 4/4 [==============================] - 0s 4ms/step - loss: 0.0701 - accuracy: 1.0000
```

- epochs: number of times the training data is presented to the neural network
- batch_size: the periodicity of weights' update (1 means that the weights are updated after the presentation of each input)

Assessing the network's performance

```
scores = model.evaluate(training_data, target_data)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
print (model.predict(training_data).round())
```

```
1/1 [==============================] - 0s 25ms/step - loss: 0.0661 - accuracy: 1.0000

accuracy: 100.00%
[[0.]
 [0.]
 [0.]
 [1.]]
```

- Notice that in this case we are testing the model with the training dataset
- In real situations a different set is used for this: the test set

Viewing the weights

```
model.weights
```

```
[<tf.Variable 'dense_10/kernel:0' shape=(2, 1) dtype=float32, numpy=
 array([[1.9860712],
        [1.9591248]], dtype=float32)>,
 <tf.Variable 'dense_10/bias:0' shape=(1,) dtype=float32, numpy=array([-3.096141], dtype=float32)>]
```

Saving and loading the model with VS Code

```
model.save("02_artificial_neurons_01_AND_model.h5")

--------------

from tensorflow import keras

loaded_model = keras.models.load_model(
    '02_artificial_neurons_01_AND_model.h5')

print (loaded_model.predict(training_data).round())
```

Saving and loading the model with Colab

```
from google.colab import drive
drive.mount('/content/drive')
--------------
model.save('/content/drive/MyDrive/models/
02_artificial_neurons_01_AND_model.h5')
--------------
from tensorflow import keras

loaded_model = keras.models.load_model(
    '/content/drive/MyDrive/models/
02_artificial_neurons_01_AND_model.h5')
print (loaded_model.predict(training_data).round())
```

Defining the dataset in a file

Creating a copy of the previous notebook

- Create a copy of the previous notebook and name it

  `02_artificial_neurons_02_AND_dataset_file.ipynb`

Defining the dataset

- Real datasets are usually defined in files, for example, csv files
- Create the `02_artificial_neurons_02_AND_dataset.csv` file with the following content:

```
0, 0, 0
0, 1, 0
1, 0, 0
1, 1, 1
```

- Each line represents one sample and has *c* columns (3, in this case)
- The first *c-1* columns represent the sample inputs
- Column *c* represents the target output for the sample

Uploading the dataset with Colab

- Now, instead of code cell

```
# Defining the dataset
training_data = np.array([[0, 0], [0, 1], [1, 0],[ 1, 1]], "float32")
target_data = np.array([[0], [0], [0], [1]], "float32")
```

we use

```
# Uploading the dataset file
from google.colab import files
uploaded = files.upload()
```

and then…

Loading the dataset with Colab/VS Code

- … and then

```
# Loading the dataset
dataset = np.loadtxt(
           '02_artificial_neurons_02_AND_dataset.csv', delimiter=',')
# We will split the array into two arrays by selecting subsets of
columns using the standard NumPy slice operator “:”.
# The following line selects the first 2 columns, from index 0 to
index 2 via the slice 0:2.
training_data = dataset[:, 0:2]
# selects the output column (the 3rd variable) via index 2.
target_data = dataset[:, 2]
```

Visualizing data

```
#Visualizing training data
print(training_data)
print(training_data.shape)
print(len(training_data))

#Visualizing target data
print(target_data)
print(target_data.shape)
print(len(target_data))
```

- Add this cell next to the previous one and run it
- Then, redo the steps needed to build the model (defining, compiling, training and assessing)

Exercises

- Try to train the model using different learning rate values
- Train one-unit neural networks that should work as
  - an OR logic function
  - a NAND logic function
  - an XOR logic function (what happens here? why?)