



Une école de l'IMT

Practice in Deep Learning TP 40

Attilio Fiandrotti

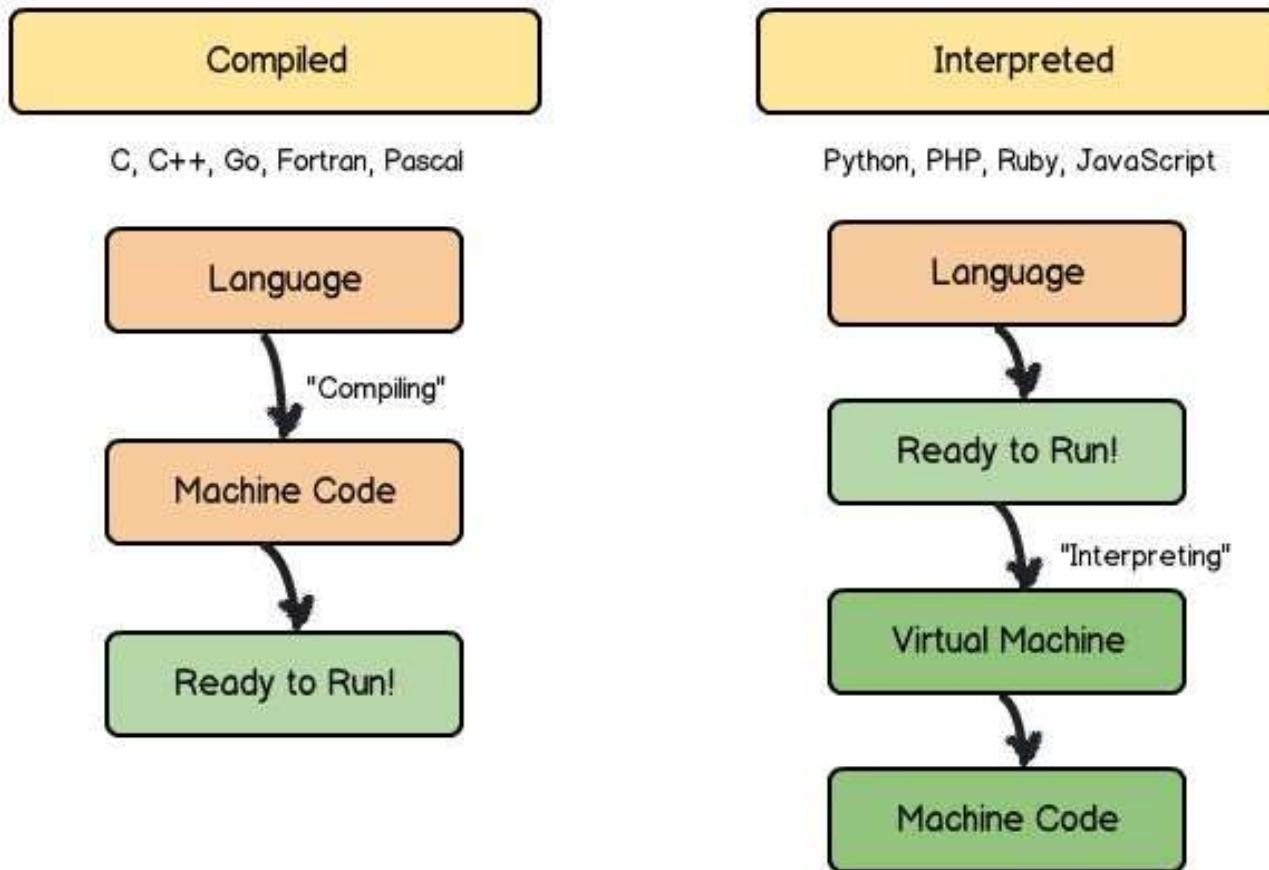
ATHENS - November 2025





Introduction to Python

Compiled vs Interpreted Languages



Python Versions and Syntax

■ Current version 3.x (PyTorch, Keras,

```
print ("Hello World!")
```

Old 2.x code not 100% compatible

```
print "Hello world!"
```

■ How do I know my python version /

```
attilio@debian:~$ python --version  
Python 3.5.3
```

```
attilio@debian:~$ which python  
/usr/bin/python
```



Indentation

■ Python requires indentation

```
if myIntVar == 10:  
    print ("First branch taken!")  
    if myStrVar == "Some text"  
        print ("Second branch taken!")
```

■ Indentation error example

```
>>> if myVar == 10:  
...     print ("Branch taken!")  
File "<stdin>", line 2  
    print ("Branch taken!")  
          ^  
IndentationError: expected an indented block
```

Variables Types

■ Weakly typized variables

```
# boolean variable  
>>> v=True  
>>> type(v)  
<type 'bool'>
```

```
# integer number  
>>> v=1  
>>> type(v)  
<type 'int'>
```

```
# floating point number  
>>> v=1.0  
>>> type(v)  
<type 'float'>
```

```
# text string  
>>> v="Hello World!"  
>>> type(v)  
<type 'str'>
```

```
# defining a list  
>>> list=['Rat','Cat','Dog']  
>>> type(list)  
<type 'list'>
```

```
# defining a list of lists  
>>> matrix=[[1,2,3],[4,5,6]]  
>>> type(matrix)  
<type 'list'>  
>>> len(matrix)  
2  
>>> len(matrix[0])  
3
```

```
# defining a dictionary  
>>> dictionary={'Rat': 10, True}  
>>> type(dictionary)  
<type 'dict'>
```

Variables by Reference

```
# creating list
>>> m=[1, 2, 3]

# creating second list pointer
>>> m2 = m

# modifying list via first pointer
>>> m[0][1] = -1
>>> m2
[-1, 2, 3]

# dropping first pointer
>>> del(m)
>>> type(m)
Traceback (most recent call last):
NameError: name 'm' is not defined

# list still accessible via second pointer
>>> type(m2)
<class 'list'>
```

Variables Cloning

■ See also *deepcopy()*

```
# original list
>>> list = [0, 1, 2]

# cloning the list
>>> new_list = list.copy()

# appending one element to the cloned list
>>> new_list.append(4)

# printing new and old list
>>> print('Old List: ', list)
Old List: [0, 1, 2]
>>> print('New List: ', new_list)
New List: [0, 1, 2, 4]
```

Conditional Branches

■ Mind the indentation!

```
>>> v=1
>>> if v == 0:
...     print("variable equal to 0")
... elif v == 1:
...     print("variable equal to 1")
... else:
...     print("variable value is " + str(v))
...
variable equal to 1
```

For Loops

■ The C / Java way

```
# define a list
>>> list=['a',-1,1.0]

# getting length of list
>>> length = len(list)

# Iterating the index
>>> for i in range(length):
... print(list[i])
a
-1
1.0

# same as 'for i in range(len(list))'
```



For Loops

■ The Python way

```
# define a list
>>> list=['a',-1,1.0]

# iterate through the list elements the python way
>>> for e in list:
...     print ("element " +str(e) + " is of type " + str(type(e)))
...
element a is of type <class 'str'>
element -1 is of type <class 'int'>
element 1.0 is of type <class 'float'>
```

While Loops

■ Also do ... while

```
# creating counter
>>> cnt = 5

# cycling through
>>> while cnt > 0:
...     print (cnt)
...     cnt = cnt - 1

...
5
4
3
2
1
```

Functions

- Positional arguments
- Keyword arguments

```
>>> def f(x, y=1, z=1):
...     return x + y + z
...
>>> print(f(1, 2, 3))
6
>>> print(f(1))
3
>>> print(f(1, 2))
4
>>> print(f(1, z=3))
5
>>> print(f(1, z=3, y=2))
6
>>> print(f(z=3, 1))
  File "<stdin>", line 1
SyntaxError: positional argument follows keyword argument
```

Importing modules

■ Importing entire libraries or subcomponents

```
>>> import numpy
>>> numpy.array([1, 2])
array([1, 2])
>>> import numpy as np
>>> np.array([1, 2])
array([1, 2])
>>> from numpy import array
>>> array([1, 2])
array([1, 2])
>>> from numpy import array as ar
>>> ar([1, 2])
array([1, 2])
```

The Anaconda Distribution

■ Problem: my OS has python 2.x, but I need 3.x

- Python 2.x needed by OS, have no root rights, ...

■ Solution: install Anaconda

- No need to be root (installed in user's home)

```
attilio@debian:~$ which python  
/home/attilio/anaconda3/bin/python
```

■ Manage libraries via *conda*

\$conda search [library name]

\$conda install [library name]



Conda Package Manager

```
attilio@debian:~$ conda -V  
conda 3.7.0
```

```
attilio@debian:~$ conda search keras  
Loading channels: done
```

#	Name	Version	Build	Channel
1	keras	1.1.1	py27_0	pkgs/free
2	keras	1.1.1	py34_0	pkgs/free
3	keras	1.1.1	py35_0	pkgs/free
4	keras	2.1.6	py27_0	pkgs/main
5	keras	2.1.6	py35_0	pkgs/main
6	keras	2.1.6	py36_0	pkgs/main

```
attilio@debian:~$ conda install [-c XYZ] keras[==2.1.6]=py27_0
```



Python Environments

■ Create the environment

```
attilio@debian:~$ conda create -n yourenvname python=x.x anaconda
```

■ Activate the environment

```
attilio@debian:~$ source activate yourenvname
```

■ Install packages in the environment

```
attilio@debian:~$ conda install -n yourenvname [package]
```

Conda Cheat Sheet



The Conda Cheat Sheet is a comprehensive guide to the command-line package and environment manager. It covers basic usage, environment management, finding packages, and managing multiple versions of Python.

Basic Usage

Learn to use conda in 30 minutes of [DataCamp](#).

conda history

- Verify conda is installed, check version number: `conda --version`
- Update conda to the current version: `conda update conda`
- Install a package included in Anaconda: `conda install NAME_OF_PACKAGE`
- Run a package after install, example Spyder: `spyder`
- Update any installed program: `conda update NAME_OF_PROGRAM`
- Command line help: `conda --help`

*Must be installed and have a deployable command, usually `NAME_OF_PACKAGE`.

Using environments

- Create a new environment named py35, install Python 3.5: `conda create --name py35 python=3.5`
- Activate the new environment to use it: `conda activate py35`
- Get a list of all my environments, other environments are shown with *: `conda env list`
- Make exact copy of an environment: `conda create --clone py36 --name py36-1`
- List all packages and versions installed in active environment: `conda list`
- Set the history of each change to the current environment: `conda list --history`
- Restore environment to a previous revision: `conda install --revision 1`
- Save environment to a tar file: `conda list --export > ENVIRONMENT.tar`
- Delete an environment and everything in it: `conda env remove -n ENVIRONMENT`
- Deactivate the current environment: `conda deactivate`
- Create environment from a tar file: `tar -xvf ENVIRONMENT.tar`
- Stack command: create a new environment, name it h5py-1 and install the h5py package: `conda create --name h5py-1 h5py`

Finding conda packages

- Use conda to search for a package: `conda search NAME_OF_PACKAGE`
- See list of all packages in Anaconda: <https://repo.anaconda.com/webroot/packages.html>

Managing and updating packages

- Install a new package (Jupyter Notebook) in the active environment: `conda install --name jupyter`
- Run an installed package (Jupyter Notebook) in a different environment (locally): `conda install --name jupyter`
- Update a package in the current environment: `conda update NAME_OF_PACKAGE`
- Install a package (locally) from a specific channel (conda-forge): `conda install --channel conda-forge NAME_OF_PACKAGE`
- Install a package directly from PyPI into the current active environment using pip: `pip install NAME_OF_PACKAGE`
- Remove one or more packages (local, installed from a specific environment path): `conda remove --name NAME_OF_PACKAGE`

Managing multiple versions of Python

- Install different version of Python in a new environment named py34: `conda create --name py34 python=3.4`
- Switch to the new environment that has a different version of Python: `conda activate py34`
- List all the locations of all versions of Python that are currently in the path: `which python`
- NOTE: The first version of Python in the path will be executed.
- Show version information for the current active Python: `python --version`

Specifying version inclusive

Way to specify a pin-range version number for use with conda locate or conda install commands, and %conda% placeholder.

Constraint type	Specification	Result
From	<code>>=0.1.11</code>	0.0, 0.1, 0.1.1, 0.1.10 etc.
Exact	<code>=0.1.11</code>	0.1.11
Greater than or equal to	<code>>=0.1.11+</code>	0.0 or higher
OR	<code>>=0.1.11.1 0.1.11.2</code>	0.1.11.2
AND	<code>>=0.1.11&<0.1.12</code>	0.1.11.0 to 0.1.11.9

NOTE: Comma-separators must be used where you specify more than one or any of these characters: < > + =

More resources

- Free Community Support: groups.google.com/forum/#!forum/anaconda
- Online Documentation: conda.io/docs
- Commercial Resources: anaconda.com/commercial/
- Paid Support Options: anaconda.com/support/
- Anaconda Online Training Courses: anaconda.com/training/
- Anaconda Consulting Services: anaconda.com/consulting/

Follow us on Twitter [@AnacondaInc](#), and join the #AnacondaCafe. Connect with other users, like minded data scientists and developers while contributing to the open source movement. Visit [anaconda.com/community](#).

© 2017 Continuum Analytics, Inc. All rights reserved. continuum.io | jupyter.org | ipython.org | anaconda.com | datacamp.com | telecom-paris.fr

Introduction to Keras

The Keras Language

- ▶ High level framework for machine learning
 - ▶ *High-level* w.r.t. PyTorch, TensorFlow, etc.
- ▶ Several backends available
 - ▶ We will use the TensorFlow backend (Google)



The Keras Language

- ▶ High level framework for machine learning
 - ▶ *High-level* w.r.t. PyTorch, TensorFlow, etc.
- ▶ Several backends available
 - ▶ We will use the TensorFlow backend (Google)
- ▶ Main author François Chollet
 - ▶ Google employee, former ENSTA ParisTech alumn



Keras System Stack



Keras



TensorFlow

CUDDnn, CUBLAS, ...



CUDA

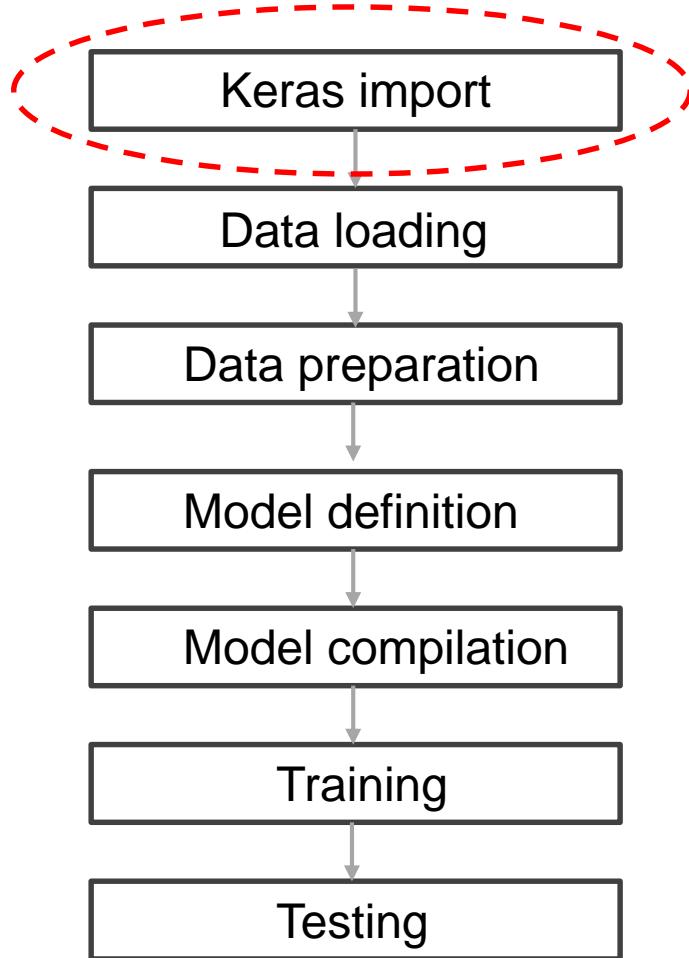


Kernel drivers

GPU

- ▶ Datasets loading
 - ▶ Popular datasets such as MNIST, etc available
 - ▶ Uses *scikit-learn* for synthetic data
- ▶ Defining network architectures
 - ▶ Non-sequential models supported
 - ▶ Pretrained deep models (AlexNet, ResNet)
- ▶ Training a network
 - ▶ Multiple optimizers available
 - ▶ One-line *fit()* function
- ▶ Visualizing data and results
 - ▶ *Relies on matplotlib* for visualization

Typical Keras Dataflow



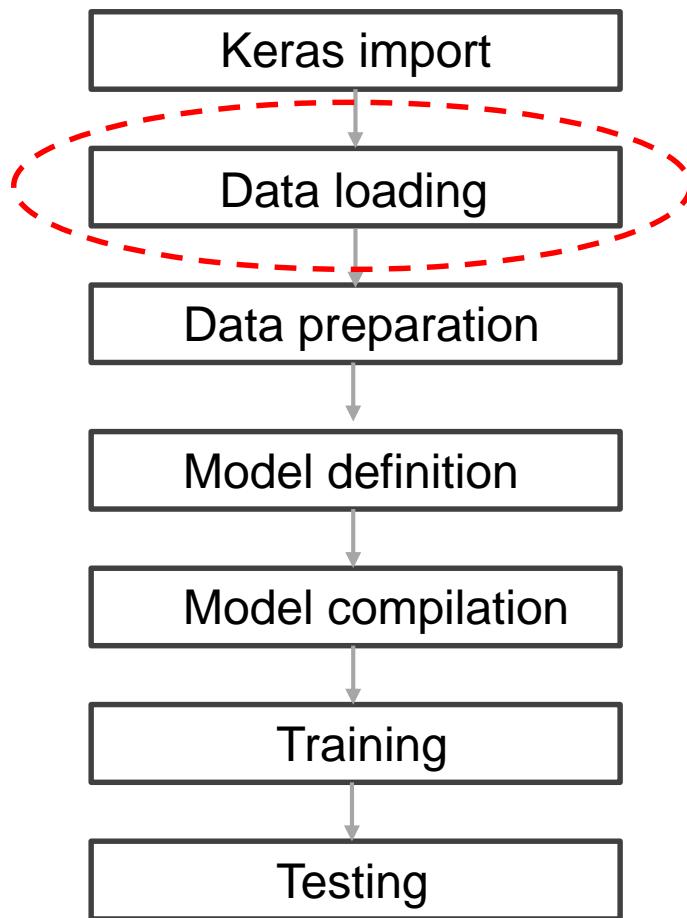
Module Import – Backend Ordering

- ▶ Make sure Keras uses the *TensorFlow* backend
- ▶ «*NHWC*» data ordering required for images
 - ▶ $N \rightarrow$ *image index in batch*
 - ▶ $H \rightarrow$ *image height*
 - ▶ $W \rightarrow$ *image width*
 - ▶ $C \rightarrow$ *image channel*

```
import keras
```

↳ Using TensorFlow backend.

Typical Keras Dataflow



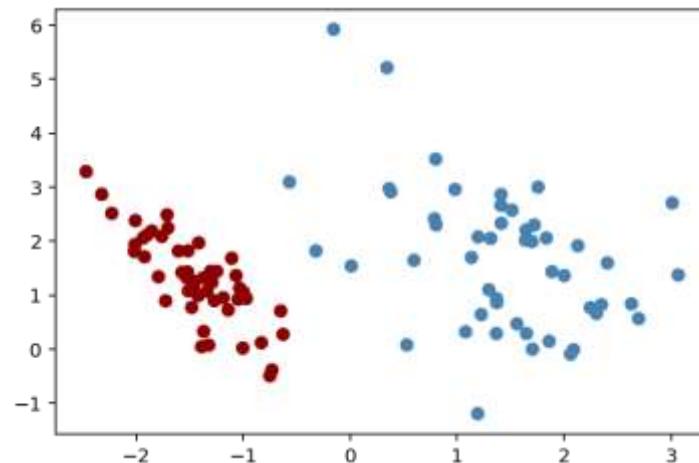
Data Generation

- ▶ Will use the *sklearn* backend

```
from sklearn import datasets
```

- ▶ Generate two Gaussian clusters of points

```
data, labels = datasets.make_classification(  
    n_samples=100,  
    n_features=2, n_informative=2, n_redundant=0,  
    n_classes=2, n_clusters_per_class=1,  
    class_sep=1.5, flip_y=0,  
    shuffle=True)
```



Data Generation

- ▶ Will use the *sklearn* backend

```
from sklearn import datasets, model_selection
```

- ▶ Generate two Gaussian clusters of points

```
data, labels = datasets.make_classification(  
    n_samples=100,  
    n_features=2, n_informative=2, n_redundant=0,  
    n_classes=2, n_clusters_per_class=1,  
    class_sep=1.5, flip_y=0,  
    shuffle=True)
```

- ▶ Separate 90% train and 90% test

```
train_data, test_data,  
train_labels, test_labels = model_selection.train_test_split(  
    data, labels,  
    test_size = 0.1)
```

Data Generation - Visualization

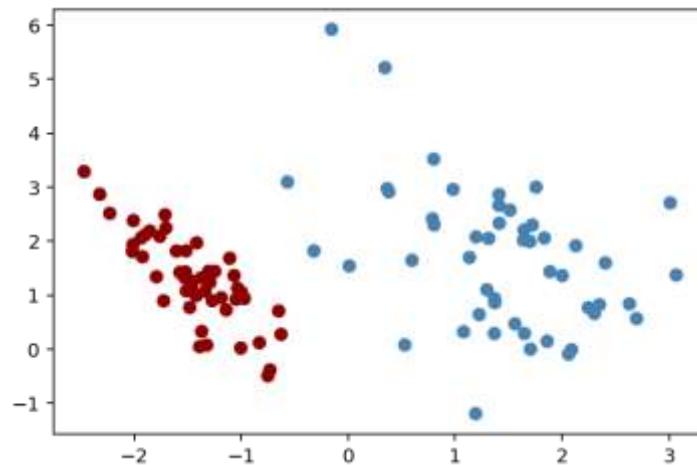
- ▶ Matlab-like plotting library

```
import matplotlib.pyplot as plt
```

- ▶ Plot the data

```
colors = ['steelblue' if label == 1 else 'darkred' for label in labels]
plt.scatter(x[:,0], x[:,1], color=colors)
plt.show()
```

```
Y.shape, X.shape  
(100,), (100, 2)
```



Dataset Loading - MNIST

- ▶ Keras has some popular datasets pre-packaged

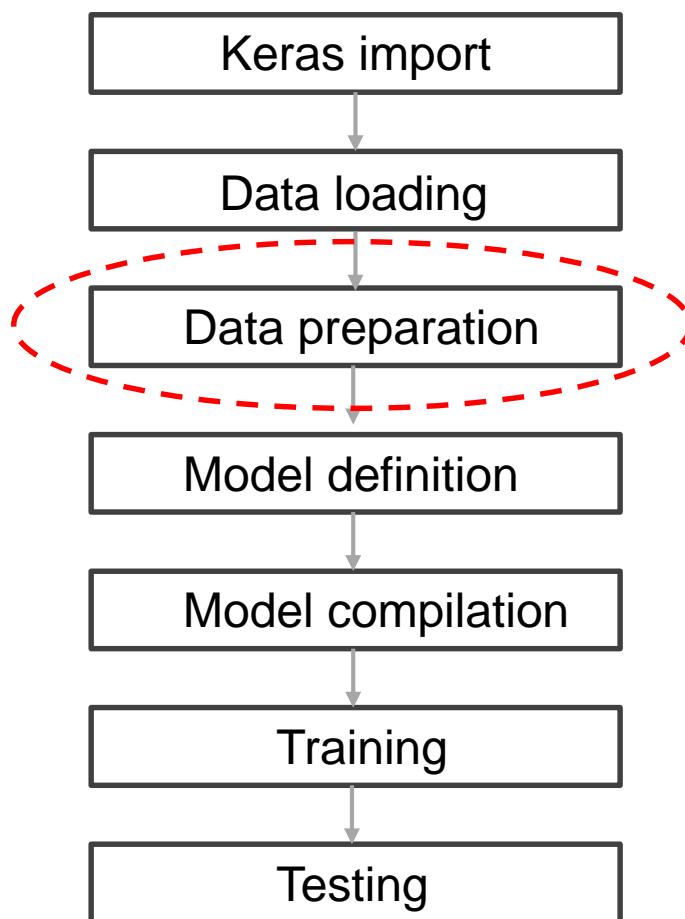
```
from keras.datasets import mnist
```

- ▶ Load images and labels into memory

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()  
  
print(y_train.shape)  
(60000,)  
print(x_train.shape)  
(60000, 28, 28)
```

- ▶ Train samples are a `numpy.ndarray` of `int8`
- ▶ N = 60000 samples
 - ▶ W x H = 28 x 28 px. samples grayscale

Typical Keras Dataflow



Data Preparation - Images

- ▶ Load images and labels into memory

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()  
  
print(x_train.shape)  
(60000, 28, 28)
```

- ▶ Samples are in (N,W,H) ordering
 - ▶ The TF backend requires (N,W,H,C) ordering
- ▶ Samples are 256-grayscale int8 arrays
 - ▶ Neural networks require float in input
- ▶ Samples are in 0-255 interval
 - ▶ Normalization desirable

Data Preparation - Images

- ▶ Let us exploit `numpy` builtins!
- ▶ Reshape the samples to NWHC order

```
x_train = numpy.reshape(x_train, newshape, order='c')
```

Data Preparation - Images

- ▶ Let us exploit `numpy` builtins!
- ▶ Reshape the samples to NWHC order

```
x_train = numpy.reshape(x_train, newshape, order='c')
```

- ▶ Recast the samples from `int8` to `float`

```
x_train = x_train.astype('float_32')
```

Data Preparation - Images

- ▶ Let us exploit `numpy` builtins!
- ▶ Reshape the samples to NWHC order

```
x_train = numpy.reshape(x_train, newshape, order='c')
```

- ▶ Recast the samples from `int8` to `float`

```
x_train = x_train.astype('float_32')
```

Data Preparation - Images

- ▶ Let us exploit `numpy` builtins!
- ▶ Reshape the samples to NWHC order

```
x_train = numpy.reshape(x_train, newshape, order='c')
```

- ▶ Recast the samples from `int8` to `float`

```
x_train = x_train.astype('float_32')
```

- ▶ Normalize to have zero-mean (and unit-std)

```
(x_train - x_train.mean())
x_train = -----
                    x_train.std()
```

Data Preparation - Images

- ▶ Let us exploit `numpy` builtins!
- ▶ Reshape the samples to NWHC order

```
x_train = numpy.reshape(x_train, newshape, order='c')
```

- ▶ Recast the samples from `int8` to `float`

```
x_train = x_train.astype('float_32')
```

- ▶ Normalize to have zero-mean (and unit-std)

```
(x_train - x_train.mean())
x_train = -----
                    x_train.std()
```

- ▶ What you do on train samples, do it also on test samples
 - ▶ Yet on train statistics!

Data Preparation - Images

- ▶ Labels (classes) are encoded as integers [0-9]

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()  
  
print(y_train.shape)  
(60000,)
```

- ▶ One-hot encoding required (multiclass problem)
 - ▶ Use `to_categorical()`

```
from keras.utils import to_categorical  
y_train_oh = to_categorical(y_train)  
  
print(y_train_oh.shape)  
(60000, 10, 2)  
  
print(train_labels.shape)  
[[1. 0.] [1. 0.] [1. 0.] [1. 0.] [1. 0.] [0. 1.] [1. 0.] [1. 0.] [1. 0.] [1. 0.]]
```

Data Loading - Generating Data

- ▶ Transform your labels to *one-hot* encoding first

```
from keras.utils.np_utils import to_categorical  
  
data, labels = datasets.make_classification(  
    n_samples=5, n_features=2, n_classes=2, [...])  
  
labels_oh = to_categorical(labels)
```

```
print(data)  
array([[ 3.34195848, -2.02906319],  
       [ 0.12096805,  2.27640523],  
       [ 2.02600963, -0.62195723],  
       [ 1.91840064, -2.19699255],  
       [ 3.47918424, -0.43343625]])
```

```
print(labels)  
array([0,  
      1,  
      0,  
      0,  
      1])
```

```
print(labels_oh)  
array([[1, 0],  
       [0, 1],  
       [1, 0],  
       [1, 0],  
       [0, 1]])
```

Data Preparation - Augmentation

- ▶ Import ImageDataGenerator

```
from tf.keras.preprocessing.image import ImageDataGenerator
```

- ▶ Create proper ImageDataGenerator instance

```
myDataGen = ImageDataGenerator(  
    rotation_range=0,  
    width_shift_range=0.1,  
    height_shift_range=0.1,  
    horizontal_flip=True,  
    vertical_flip=False  
)
```

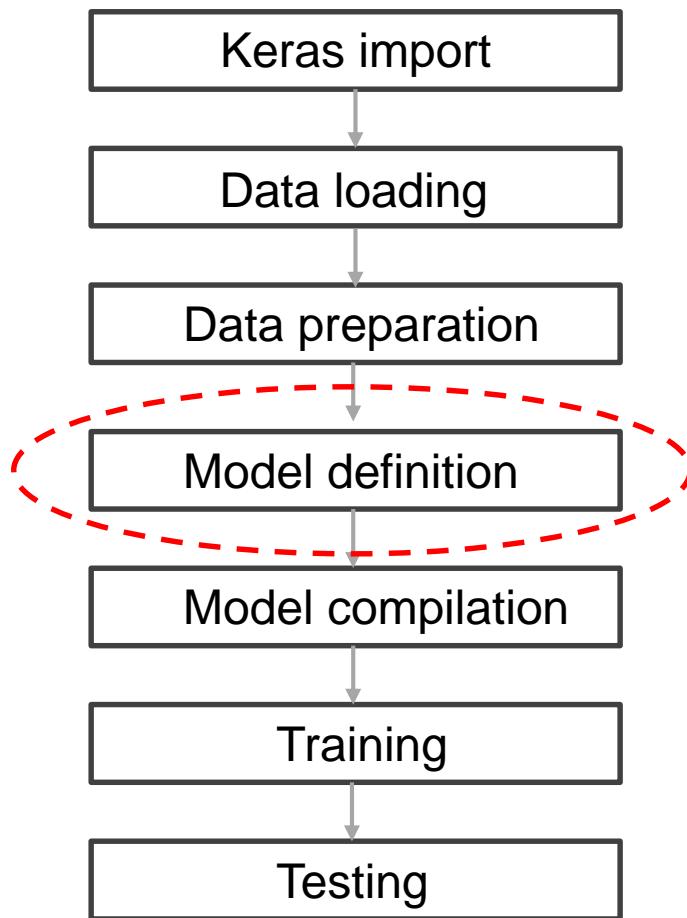
- ▶ Can optionally normalize data for you

- ▶ Must fit generator to your data if normalization used!

- ▶ Further steps required during training

- ▶ Detailed later

Typical Keras Dataflow



Model definition – Binary Classifier

- ▶ Import keras modules

```
from keras.models import Sequential  
from keras.layers import Input, Dense, Activation
```

- ▶ Use the *sequential API*

```
model = Sequential()
```



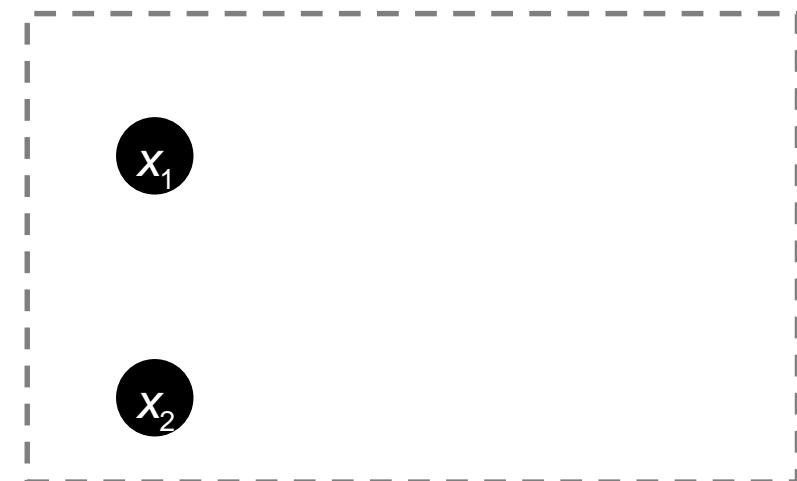
Model definition – Binary Classifier

- ▶ Import keras modules

```
from keras.models import Sequential  
from keras.layers import Input, Dense, Activation
```

- ▶ Use the *sequential API*

```
model = Sequential()  
model.add(Input(shape=(2,)))
```



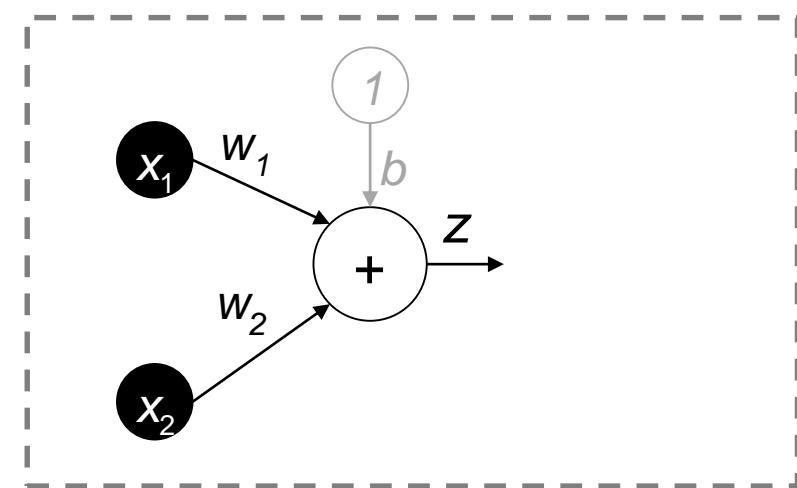
Model definition – Binary Classifier

- ▶ Import keras modules

```
from keras.models import Sequential  
from keras.layers import Input, Dense, Activation
```

- ▶ Use the *sequential API*

```
model = Sequential()  
model.add(Input(shape=(2,)))  
model.add(Dense(units=1))
```



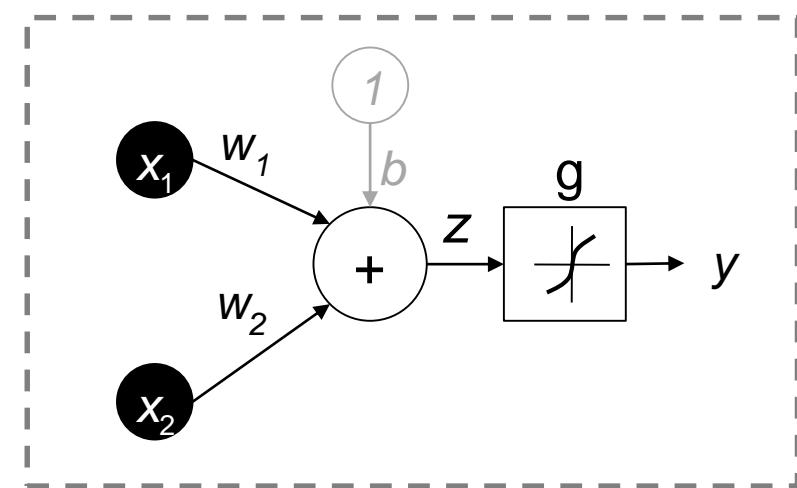
Model definition – Binary Classifier

- ▶ Import keras modules

```
from keras.models import Sequential  
from keras.layers import Input, Dense, Activation
```

- ▶ Use the *sequential API*

```
model = Sequential()  
model.add(Input(shape=(2,)))  
model.add(Dense(units=1))  
model.add(Activation('sigmoid'))
```



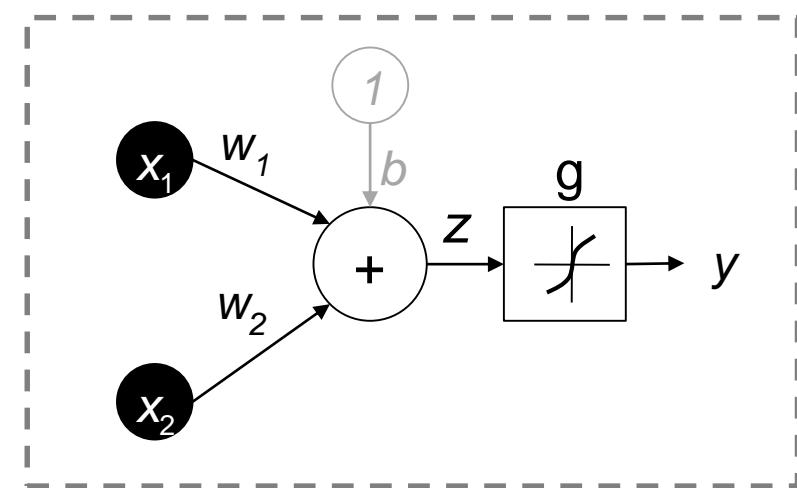
Model definition – Binary Classifier

- ▶ Import keras modules

```
from keras.models import Sequential  
from keras.layers import Input, Dense, Activation
```

- ▶ Use the *sequential API*

```
model = Sequential()  
model.add(Input(shape=(2,)))  
model.add(Dense(units=1,  
               activation='sigmoid'))
```



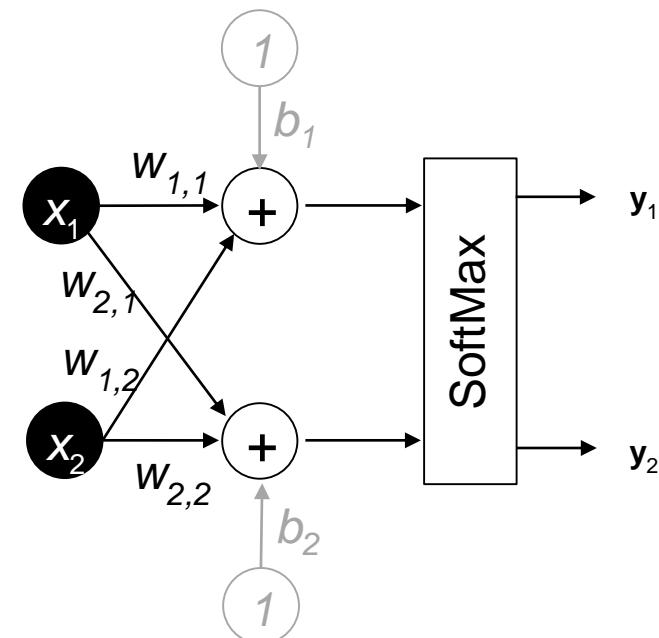
Model definition – Multiclass Classifier

- ▶ Import keras modules

```
from keras.models import Sequential  
from keras.layers import Input, Dense, Activation
```

- ▶ Use the *sequential API*

```
model = Sequential()  
model.add(Input(shape=(2,)))  
model.add(Dense(units=2,  
               activation='softmax'))
```



Model definition – Convolutional Layers

▶ Import keras modules

```
from keras.models import Conv2D, MaxPooling2D
```

▶ Convolutional layer with sigmoid activation and MaxPooling

```
model = Sequential()
model.add(Input(w,h,c))
model.add(Conv2D(filters=6,
                 kernel_size=(5,5),
                 padding='same',
                 data_format='channels_last'))
model.add(Activation('sigmoid'))
model.add(MaxPooling2D(pool_size=2))
```

▶ Serialize the feature maps into feature vectors

```
model.add(Flatten())
```

Model definition - Regularizers

- ▶ Implemented as layer parameters

```
from keras import regularizers
```

- ▶ L1 and L2 norm regularizers commonly used
- ▶ Apply individually to each layer

```
model = Sequential()  
  
model.add(Dense(1000,  
                kernel_regularizer=regularizers.l2(0.01),  
                bias_regularizer=regularizers.l2(0.01)))  
  
model.add(Activation(...))  
model.add(Dense(10, ...))  
model.add(Activation('softmax'))
```

Model definition - Dropout

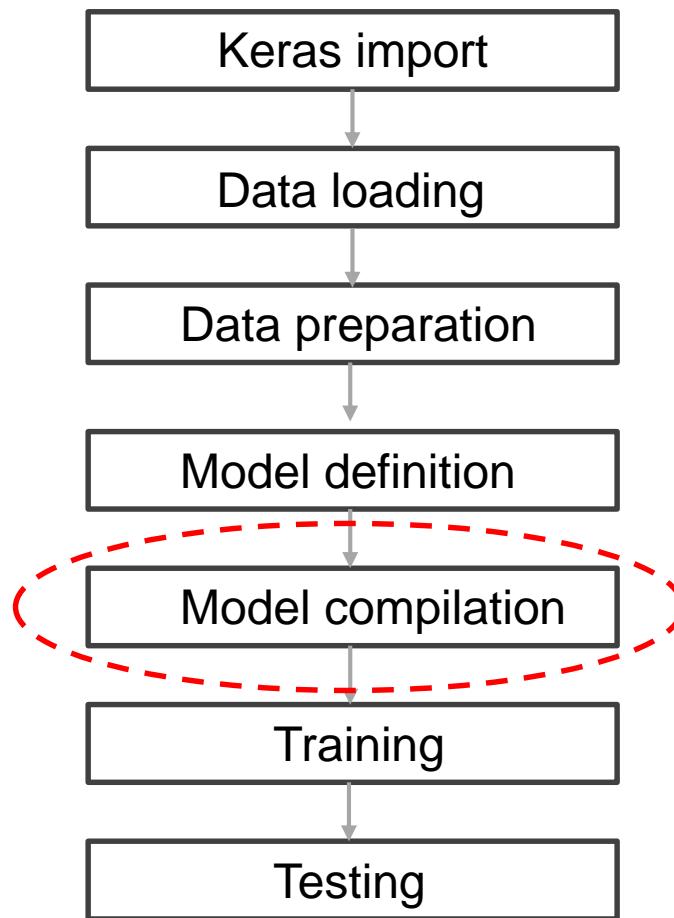
- ▶ Implemented as a layer

```
from keras.models import Dropout  
keras.layers.Dropout(rate, noise_shape=None, seed=None)
```

- ▶ Usually useful before most parametrized layer

```
model = Sequential()  
model.add(Input(...))  
model.add(Conv2D(...))  
model.add(Activation(...))  
model.add(MaxPooling2D(...))  
model.add(Flatten())  
  
model.add(Dropout(rate = 0.5)  
  
model.add(Dense(1000))  
model.add(Activation(...))  
model.add(Dense(10, ...))  
model.add(Activation('softmax'))
```

Typical Keras Dataflow



Model compilation – Binary Classifier

- ▶ Define an optimizer

```
myOpt = tf.keras.optimizers.SGD(learning_rate=0.01, decay=10e-6)
```

- ▶ Define loss (and performance) metric

```
model.compile(optimizer=myOpt, loss='binary_crossentropy',  
              metrics=['accuracy'])
```

- ▶ After compiling the model, visualize it

```
model.summary()
```

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 1)	3
Total params: 3		

Model compilation – Multiclass Classifier

- ▶ Define an optimizer

```
myOpt = tf.keras.optimizers.SGD(learning_rate=0.01, decay=10e-6)
```

- ▶ Define loss (and performance) metric

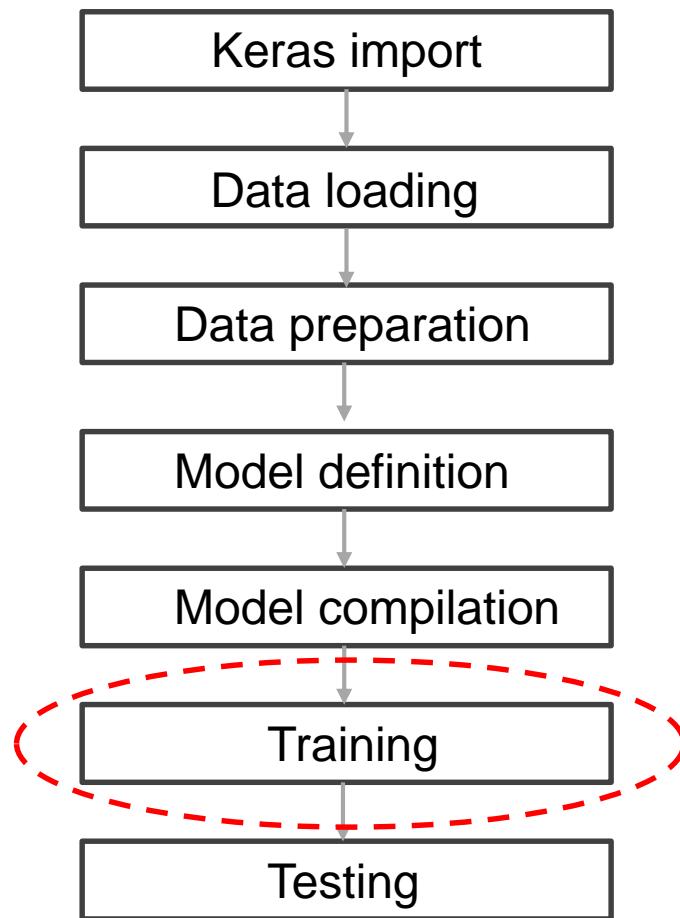
```
model.compile(optimizer=myOpt, loss='categorical_crossentropy', [...])
```

- ▶ After compiling the model, visualize it

```
model.summary()
```

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 2)	6
Total params: 6		

Typical Keras Dataflow



Training

► High-level `fit()` function

```
history = model.fit(x=train_data[firstSample:lastSample],  
                     y=train_labels[firstSample:lastSample],  
                     validation_data=(test_data, test_labels),  
                     epochs=10,  
                     batch_size=32,  
                     shuffle=True,  
                     verbose=0|1|2)
```

► Set `verbose=1` to visualize this output

```
Epoch 1/100  
100/100 [=====] - 0s 2ms/step - loss: 0.9530 - acc: 0.5500  
Epoch 2/100  
100/100 [=====] - 0s 94us/step - loss: 0.6889 - acc: 0.5700  
  
...  
  
Epoch 99/100  
100/100 [=====] - 0s 70us/step - loss: 0.0371 - acc: 1.0000  
Epoch 100/100  
100/100 [=====] - 0s 67us/step - loss: 0.0369 - acc: 1.0000
```

► Train/Validation loss/accuracy logged in history

Training with data generators – fit_generator()

- ▶ High-level fit_generator() function

```
history = model.fit_generator(  
    myDataGen.flow(train_data, train_labels, batch_size=32),  
    steps_per_epoch = train_images.shape[0] / 32,  
    validation_data=(test_data, test_labels),  
    epochs=10,  
    shuffle=True,  
    verbose=1,  
    workers=1  
)
```

- ▶ Requires setting up a DataGenerator
 - ▶ required if you have a custom dataset
- ▶ Also validation_data could be augmented in principle
 - ▶ Typically only train data augmented

Training with data generators – train_on_batch()

► Manually iterate over train/test batches

```
for e in range(numEpochs):
    # Loop over training samples in batches of 32 samples
    trainLoss = 0; trainBatches = 0
    for x_batch, Y_batch in
        datagenTrain.flow(train_data, train_labels, batch_size=32):
        trainLoss += model.train_on_batch(x_batch, Y_batch)
        trainBatches += 1
        if trainBatches >= len(x_train) / 32:
            break
    # Loop over testing samples in batches of 32 samples
    testLoss = 0; testBatches = 0
    for x_batch, Y_batch in
        datagenTest.flow(test_data, test_labels, batch_size=32):
        testLoss += model.test_on_batch(x_batch, Y_batch)
        testBatches += 1
        if testBatches >= len(x_test) / batchSize:
            break

    print ('Epoch' + str(e) +
          ' trainLoss ' + str(trainLoss/trainBatches) +
          ' testLoss ' + str(testLoss/testBatches) )
```

Training - Monitoring

► Monitor the CPU usage via htop

```
File Modifica Visualizza Cerca Terminale Aiuto

1 [|||] 26.7% 13 [|||] 17.1% 25 [ ] 0.0% 37 [ ] 0.0%
2 [|||] 6.3% 14 [|||] 6.5% 26 [ ] 0.0% 38 [ ] 0.0%
3 [ ] 1.3% 15 [|||] 25.3% 27 [|||] 0.7% 39 [|||] 0.7%
4 [|||] 3.3% 16 [|||] 3.9% 28 [ ] 0.0% 40 [ ] 1.3%
5 [ ] 0.6% 17 [|||] 9.9% 29 [|||||] 37.9% 41 [ ] 0.0%
6 [ ] 0.0% 18 [ ] 2.6% 30 [|||||] 29.6% 42 [|||] 6.0%
7 [ ] 0.0% 19 [ ] 0.0% 31 [ ] 0.0% 43 [|||] 20.4%
8 [|||] 0.7% 20 [|||] 2.6% 32 [|||] 3.3% 44 [|||] 7.9%
9 [|||] 16.9% 21 [ ] 0.0% 33 [ ] 0.0% 45 [|||] 5.9%
10 [ ] 0.7% 22 [ ] 0.0% 34 [ ] 0.0% 46 [|||] 7.2%
11 [ ] 2.0% 23 [ ] 0.0% 35 [ ] 0.0% 47 [ ] 0.0%
12 [ ] 0.0% 24 [ ] 0.0% 36 [ ] 0.0% 48 [|||] 0.7%
Mem[|||||||||||||] 4.02G/126G Tasks: 81, 148 thr; 3 running
Swp[ ] 142M/128G Load average: 2.60 2.54 2.32
Uptime: 31 days, 13:43:06

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
12564 vsuser 20 0 25.2G 2674M 442M R 67.4 2.1 28:38.25 python vssr.py in_folder/
12624 vsuser 20 0 25.2G 2674M 442M S 2.0 2.1 1:20.90 python vssr.py in_folder/
41916 abmessaou 20 0 15356 3992 2472 S 0.7 0.0 1h07:27 watch -n 1 nvidia-smi
18814 attilio 20 0 27092 4628 3544 R 2.0 0.0 0:01.22 htop
31480 vsuser 20 0 32004 2320 1316 S 0.0 0.0 13:35.47 SCREEN -S Terminal Nour
12621 vsuser 20 0 25.2G 2674M 442M S 0.0 2.1 0:02.86 python vssr.py in_folder/
1998 root 20 0 17308 432 364 S 0.0 0.0 0.0 8:31.73 /usr/bin/nvidia-persisten
36214 abmessaou 20 0 103M 6068 5060 S 0.0 0.0 2:03.63 sshd: abmessaoudi@pts/2
1 root 20 0 220M 5332 3552 S 0.0 0.0 0.0 2:08.96 /sbin/init
893 root 19 -1 187M 68652 57000 S 0.0 0.1 12:40.65 /lib/systemd/systemd-jour
920 root 20 0 97708 0 0 S 0.0 0.0 0:00.00 /sbin/lvmetad -f
F1Help F2Setup F3Search F4Filter F5Tree F6SortByF7Nice -F8Nice +F9Kill F10Quit
```

Training - Monitoring

► Monitor the GPU usage via nvidia-smi

```
File Modifica Visualizza Cerca Terminale Aiuto
Every 1.0s: nvidia-smi                                         sun: Mon May 27 23:10:20 2019

Mon May 27 23:10:20 2019
+-----+
| NVIDIA-SMI 410.79      Driver Version: 410.79      CUDA Version: 10.0 |
+-----+
| GPU  Name Persistence-M| Bus-Id     Disp.A  Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M. |
|-----+-----+-----+-----+-----+-----+-----+-----+
| 0  GeForce RTX 208... On  00000000:18:00.0 Off |          N/A |
| 28%   27C    P8    14W / 175W |    10MiB / 10989MiB | 0%       Default |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1  GeForce RTX 208... On  00000000:3B:00.0 Off |          N/A |
| 28%   28C    P8    18W / 175W |    10MiB / 10989MiB | 0%       Default |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 2  GeForce RTX 208... On  00000000:86:00.0 Off |          N/A |
| 39%   66C    P2    169W / 175W |  2771MiB / 10989MiB | 87%       Default |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 3  GeForce RTX 208... On  00000000:AF:00.0 Off |          N/A |
| 28%   33C    P8    8W / 175W |    10MiB / 10989MiB | 0%       Default |
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
| Processes:                               GPU Memory |
| GPU  PID  Type  Process name             Usage    |
|-----+-----+-----+-----+
| 2    12564  C    python                  2761MiB |
+-----+
```

Training - Analysis

▶ Using matplotlib again

```
import matplotlib.pyplot as plt
```

▶ Plot the loss graph

```
plt.figure()
plt.subplot(211)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.ylabel('Loss')
plt.xlim(left=1, right=10)
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()

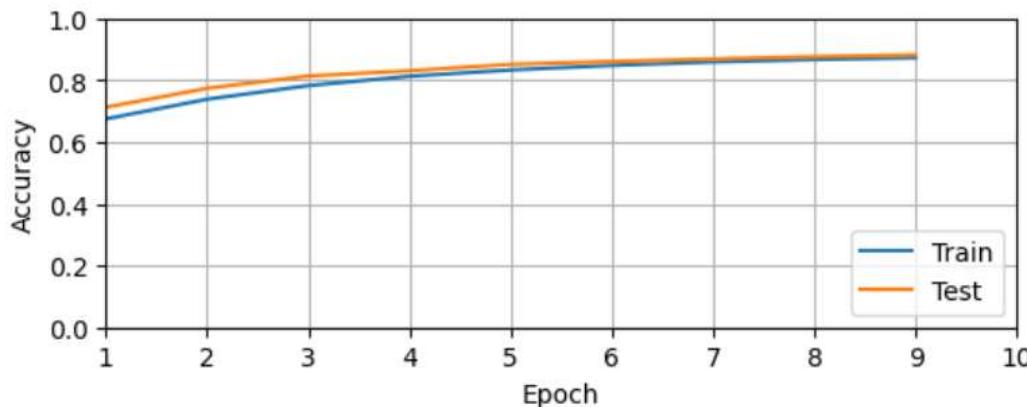
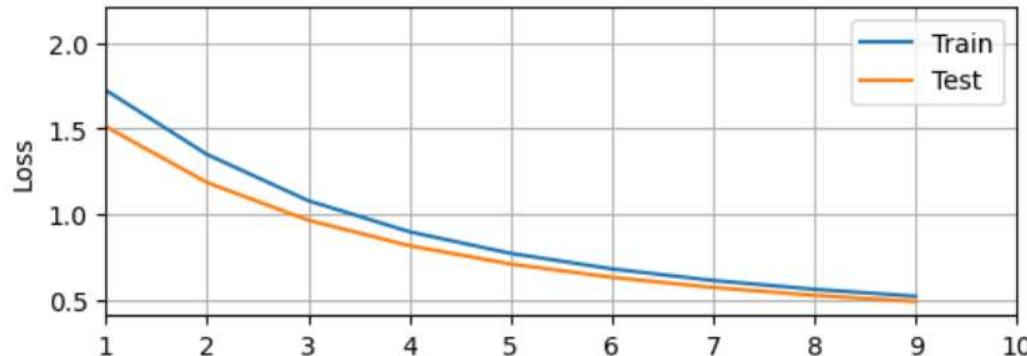
plt.subplot(212)
plt.grid(visible=True)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.ylabel('Accuracy')
plt.xlim(left=1, right=10); plt.ylim(top=1, bottom=0)
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='lower right')
plt.show()
```

Training - Analysis

- ▶ Using matplotlib again

```
import matplotlib.pyplot as plt
```

- ▶ Plot the loss graph



Training – Confusion Matrix

- ▶ Another courtesy of sklearn

```
from sklearn.metrics import confusion_matrix
```

- ▶ Which class is likely confused with which

```
predictions = model.predict(test_images)
matrix = confusion_matrix(test_labels.argmax(axis=1),
                           predictions.argmax(axis=1))
print (matrix)
```

```
[[ 950    0    1    5    1    7    8    1    5    2]
 [  0 1109    2    4    0    2    3    0   15    0]
 [  7    3  943   17   18    2   11   12   12    7]
 [  3    3   14  927    1   22    2   10   22    6]
 [  1    3    4    1  891    0   35    0    4   43]
 [ 12    5    5   54   14  733   16    6   34   13]
 [ 14    4    1    2   15    5  913    0    4    0]
 [  1   19   38    7    7    1    0  918    4   33]
 [ 11    0    5   41   11   20   11    8  853   14]
 [  7    7    9    7   30    7    2   23   13  904]]
```

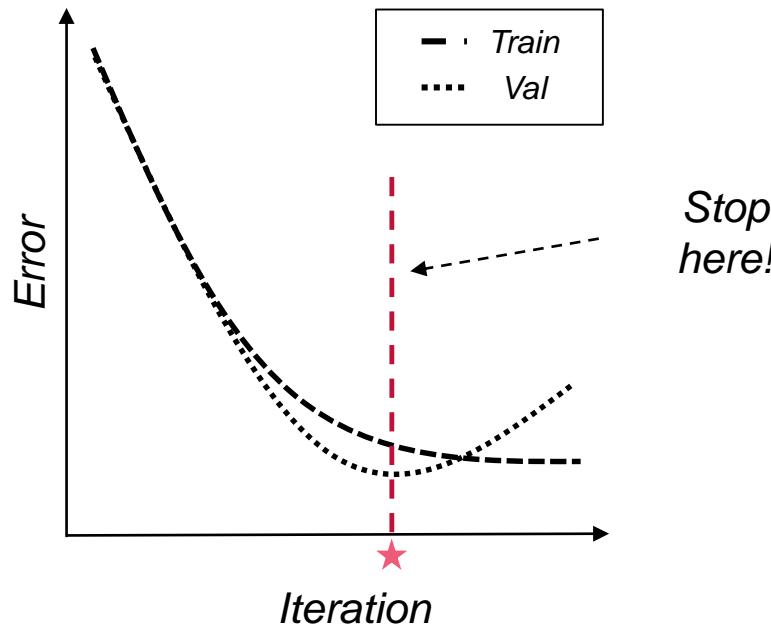
Training – Early Stop

- ▶ Useful functions for a variety of purposes

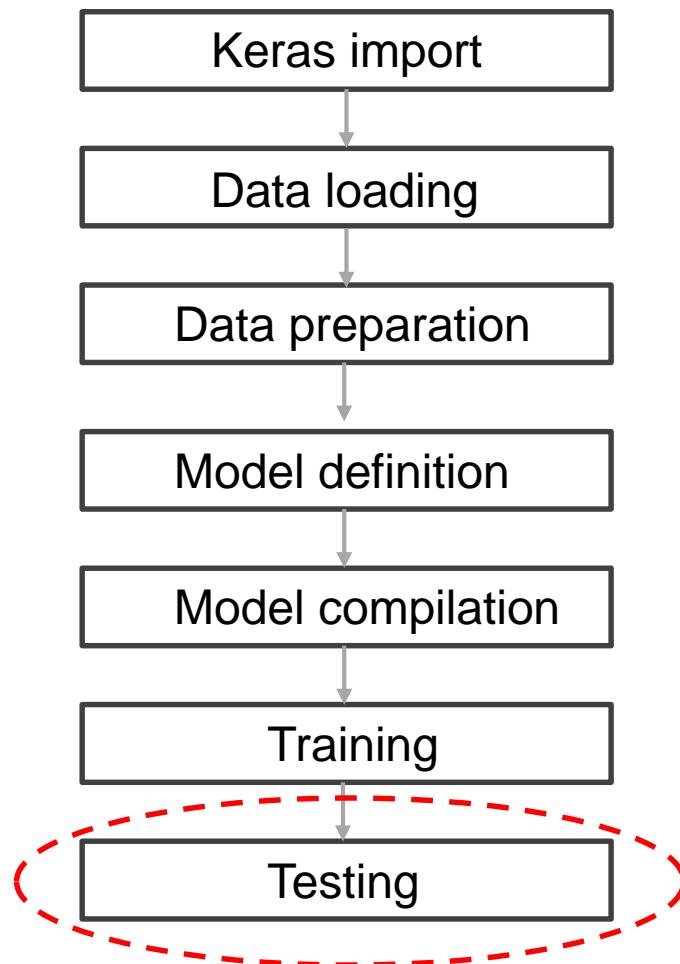
```
from keras.callbacks import EarlyStopping
```

- ▶ Ends training when validation loss stops decreasing

```
early_stopping = EarlyStopping(monitor='val_loss', patience=2)
model.fit(..., callbacks=[early_stopping])
```



Typical Keras Dataflow



Testing

▶ Left-out samples

```
score = model.evaluate(  
    data_test[firstSample:lastSample],  
    labels_test[firstSample:lastSample]  
)
```

Model Saving / Loading

- ▶ Let us save the trained model (topology + params)

```
import os
model_name = 'trained_model.h5'
model.save(model_path)
print('Saved trained model at ' + os.getcwd() + '/' + model_path)
del(model)
```

- ▶ And let us load the trained model later on

```
from keras.models import load_model
model = load_model('trained_model.h5')
```

- ▶ Save/load only the parameters

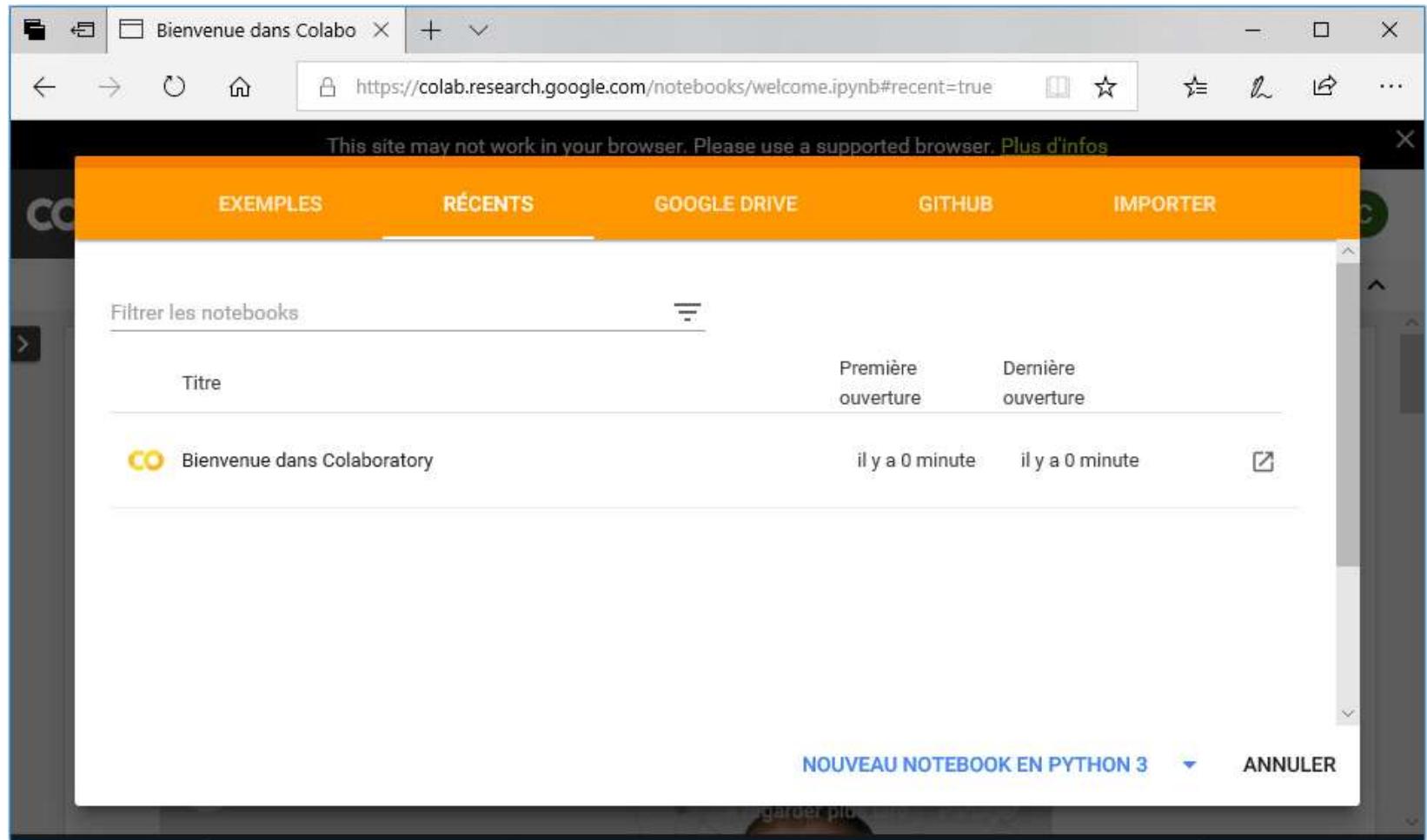
```
model.save_weights('my_model_weights.h5')

[...]

model.load_weights('my_model_weights.h5', by_name=True)
```

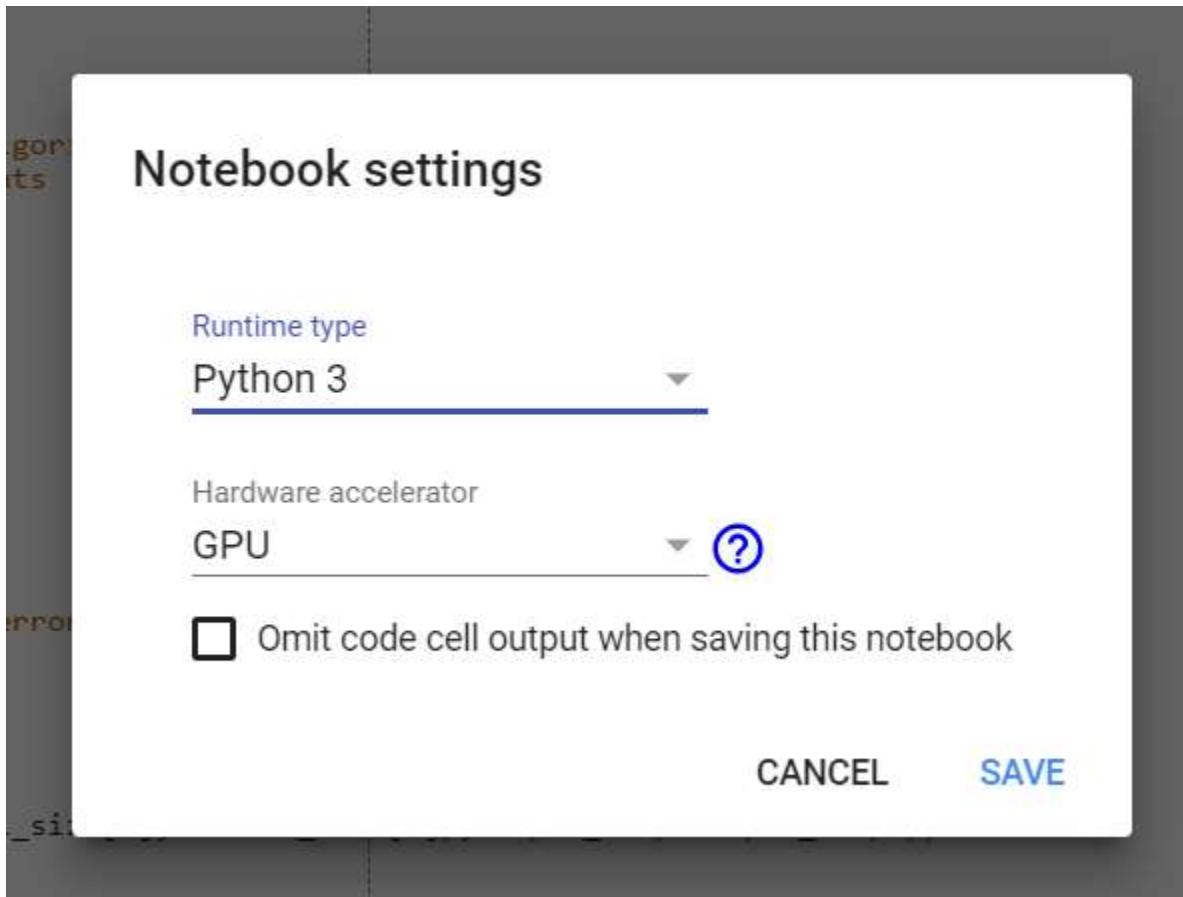
Google Colab

► Jupiter-like development environment



Google Colab

- ▶ If available, select a GPU-enabled VM («*runtime*»)
 - ▶ *Menu - Runtime – Change runtime type - GPU*



Colab Diagnostics

- ▶ Check which GPU you have available
 - ▶ «!» means system-level command (*bash*)

```
[ ] !nvidia-smi
```

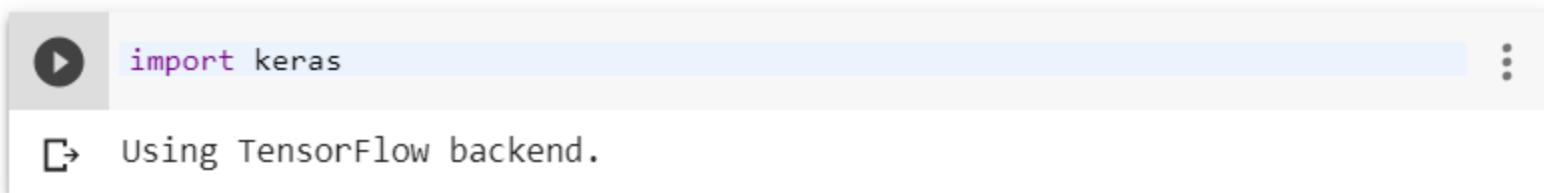
```
↳ Wed Nov 21 13:29:39 2018
```

NVIDIA-SMI 396.44				Driver Version: 396.44		
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr. ECC
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.
0	Tesla K80	Off	00000000:00:04.0	off	0	
N/A	35C	P0	68W / 149W	649MiB / 11441MiB	0%	Default

Processes:				GPU Memory
GPU	PID	Type	Process name	Usage

Colab Diagnostics

- ▶ Make sure Keras uses the *TensorFlow* backend
 - ▶ «*NHWC*» *data ordering required*

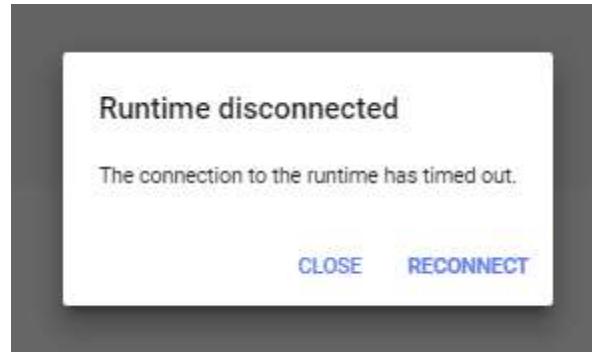


```
import keras
```

↳ Using TensorFlow backend.

Preliminaries

- ▶ Do not let the terminal inactive to avoid disconnection
 - ▶ Loss of all session variables and data on filesystem



Exercises

1. Generate two linearly separable classes of points and train a simple sigmoid classifier
2. Generate two non linearly separable classes of points and find the simplest FCN capable to separate them

```
data, labels = datasets.make_moons(n_samples=1000, noise=0.05, random_state=0)
```

3. Find the projection of the input data to a linearly separable space using the above trained architecture