DEEP LEARNING LESSONS

Python for Deep Learning

Francesco Pugliese, PhD
Data Scientist at ISTAT
francesco.pugliese@istat.it

Python for Deep Learning

Why starting with Python?

- **Python** is a perfect choice for beginners to make your focus on in order to jump into the field of machine learning and data science. It is a minimalistic and intuitive language with a full-featured library line (also called frameworks) which significantly reduces the time required to get your first results.

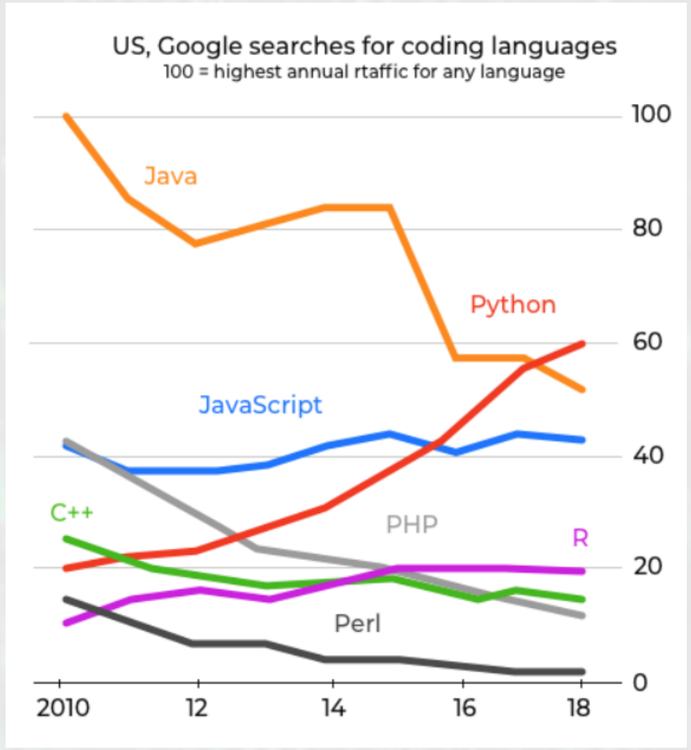
- **AI projects** differ from traditional software projects. The differences lie in the technology stack, the skills required for an AI-based project, and the necessity of deep research. To implement your AI aspirations, you should use a programming language that is stable, flexible, and has tools available. Python offers all of this, which is why we see lots of Python AI projects today.

- From development to deployment and maintenance, Python helps developers to be productive and confident about the software they are building.

Why starting with Python?

- **Python** is:

1. **Simple and consistent**
2. **A language with an extensive selection of libraries and frameworks**
3. **Platform independent**
4. **Has a great community and popularity**



The Ide Jupyter Notebook

- **What is Jupyter Notebook?**

- The **Jupyter Notebook** is an **open-source web application** that allows you to create and share documents that contain **live code, equations, visualizations and narrative text.** Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

- **According to some users opinions:**

1. Jupyter Notebooks are ususally in a love-and-hate relationship with users.

2. They are great if you see them once in a while. They are great for preparing workshops, presentations in which you show code. You can give every student a fresh, fully interactive learning environment. Sharing analysis sketches is trivial and beautiful, without any setup required by others.

3. However, **jupyter notebooks** live in their own environment, making sometimes it incredibly frustrating to reuse code or apply tools.

4. There is lots of hidden magic, leading to confusing and untraceable interactions with other libraries.

The Ide Spyder

- **What is Spyder?**

- **Spyder** is an open-source cross-platform **Integrated Development Environment (IDE)** for scientific programming in the Python Language.

- **Spyder** integrates with a number of prominent packages in the scientific Python stack, including NumPy, SciPy, Matplotlib, Pandas, IPython, SymPy and Cython, as well as other open source software. It is released under the MIT license.

- Initially created and developed by **Pierre Raybaut** in **2009**, since **2012** Spyder has been maintained and continuously improved by a team of scientific Python developers and the community.

- **Spyder** is extensible with first- and third-party plugins, including support for interactive tools for data inspection and embeds Python-specific **code quality** assurance and introspection instruments, such as Pyflakes.

The Ide Spyder

- **Pros and Cons**

PRO Free and open-source
Released under the MIT license.

PRO Graph plotting support
Spyder can plot graphs and provide the list of all variables.

PRO Relatively lightweight

PRO Powerful autocompletion
Spyder's autocomplete features are made possible by a library called rope which gives Spyder powerful autocompletion.

PRO Has support for Vim bindings via plugin support
Aside from being an open sourced, actively developed IDE, vim key-binding support is also available. If you remember Pydee - this is it, albeit with a new name.

PRO Helps you to use documentation

PRO Enables to write consistent code
Pylint integration enables to check the code for PEP8 style guide and detect errors.

PRO Has cross platform support - Linux, Mac, and even Windows
Spyder (formerly Pydee) has support for all of the major operating platforms - Linux, Mac, and even Windows.

PRO Intuitive interface

PRO Good GitHub project

PRO Excellent variable explorer
Dynamic variable explorer with editor and visualizer

PRO Completely Python

CON Not beautiful
The default theme is not beautiful. And there are not many themes.

CON The documentation is poor when it comes to debugging
Not a lot of information about debugging is available in the documentation.

CON Consumes a lot of memory
If you're working with large data, especially arrays, another IDE should be considered as spyder uses at least 200-300Mb of memory.

The Best Ide Ever: Eclipse + PyDev

- **Why Eclipse?**

1. Eclipse is a powerful IDE with many built in features.
2. Eclipse has an extensive marketplace for plugins.
3. It’s Open Source.
4. Easily sync projects to remote servers over SSH.
5. Nice Git integration.

- **What is PyDev?**

1. **PyDev** is a Python IDE for **Eclipse**, which may be used in **Python**, Jython and IronPython development.
2. It comes with many goodies such as:

- Django Integration
- Code completion
- Code Analysis
- Refactoring
- Remote Debugger
- Go to Definition
- Etc.

The Ide Eclipse

- **Pros and Cons**

PRO Feature rich including Django integration

Django Integration, Auto code completion, Multi language support, Integrated Python Debugging, code analysis, code templates ,smart indent, bracket matching, error markup, source control integration, code folding, UML Editing and viewing, and unit test integration.

PRO Can be extended to have additional features through plugins

Eclipse has a large and active community, which has resulted in a wide variety of plugins.

PRO Good integration of interactive python console even in debugging mode

PRO Handles non-Python components well

If your project includes non-Python pieces (ie. Javascript, CSS, UML, etc) Eclipse tends to provide much better support for these than IDEs focused exclusively on Python.

PRO Can use different environments for different projects

It's very handy for managing multiple projects that each use their own virtualenv or conda environment, you can assign a different interpreter to each project and they will handle things like code completion correctly.

PRO Good font rendering

Because Eclipse is based on SWT, it uses the native font rendering and thus looks better than other IDEs on some Linux systems, where the Java font rendering is not optimal.

PRO One of the few that can debug Jython code

It handles both Jython and CPython.

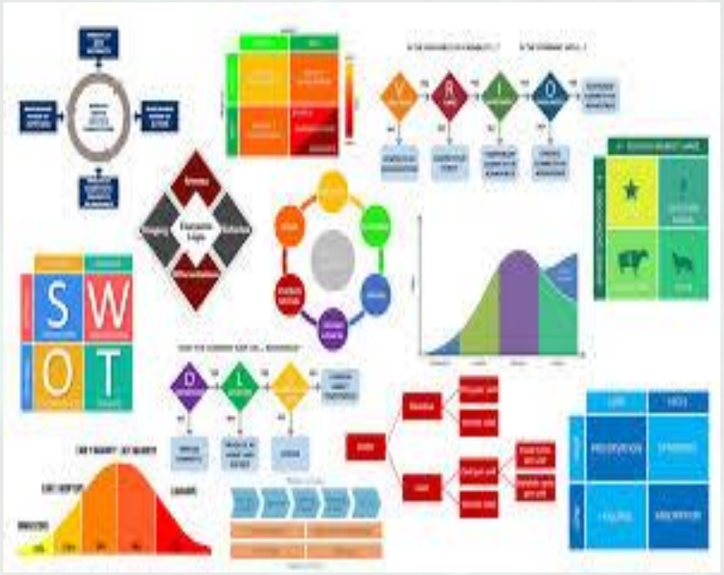
CON Just plugin for python, not full editor

It useful only if you use python as additional (secondary) language in your project on Eclipse

CON Plugins can be unstable

Though there are plenty of plugins to choose from, they aren't always reliable. Some aren't maintained, bug fixes can be slow, and you may need to download plugins from multiple sources.

Frameworks for Deep Learning

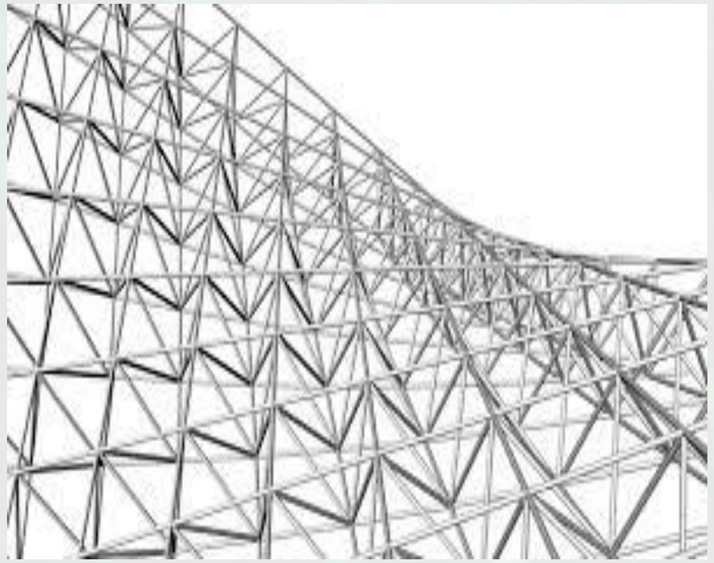


Keras is an higher-level interface for Theano (which works as backend). Keras displays a more intuitive set of abstractions that make it easy to configure neural networks regardless of the backend scientific computing library.

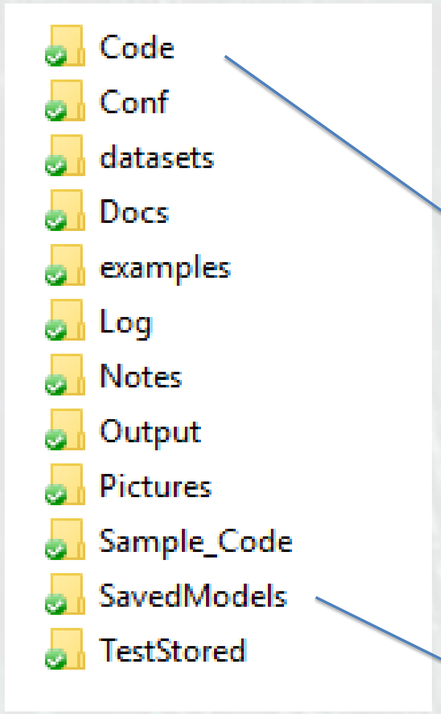
TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and also used for machine learning applications such as neural networks. It is used for both research \and production at Google.

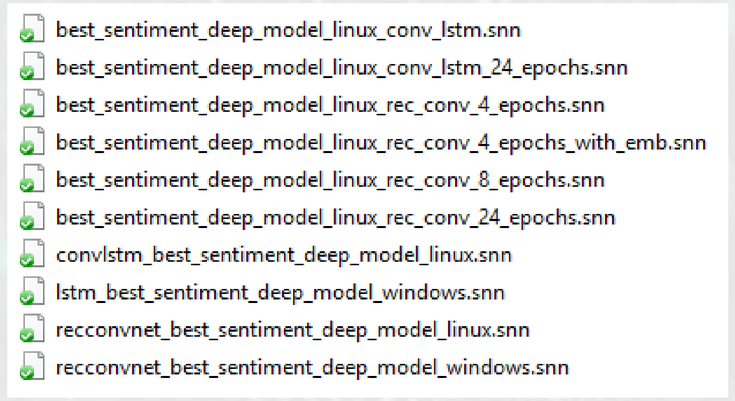
PyTorch is an open-source machine learning library for Python, derived from Torch, used for applications such as natural language processing. It is primarily developed by Facebook’s artificial-intelligence research group, and Uber’s "Pyro" software for probabilistic programming is built on it.

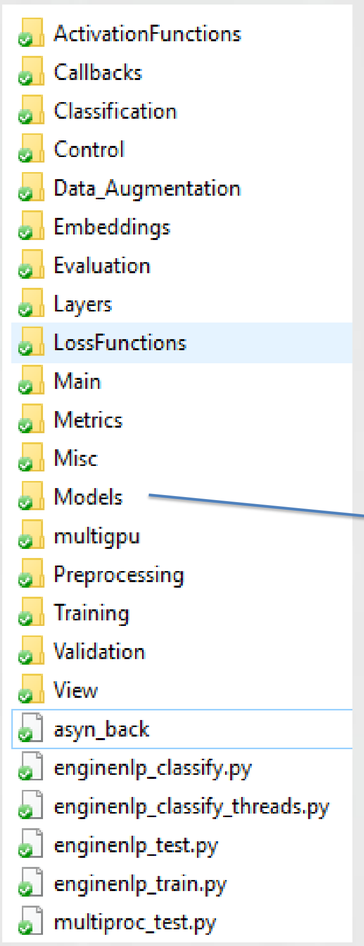
Anatomy of an advanced Python Machine Learning Application

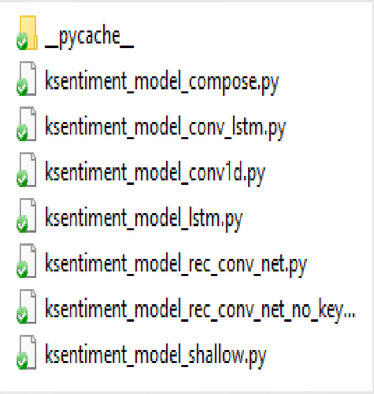


Python for Deep Learning
Francesco Pugliese









Keras: The Python Deep Learning Library

- **Keras** is an higher-level API to build neural networks Quickly. It is written in Python and works in Python.
- **Keras** works on top of **Tensorflow, CNTK** or **Theano** accordingly. It was developed keeping in mind prototyping and fast experimentation.
- **Keras** is able to go from the idea to the result with the least possible delay, which is the key to doing both good research and value business.
- We can adopt **Keras** in the cases we need:

  1) Enabling fast prototyping by means of friendliness, modularity and extensibility
  2) Supporting both Convolutional Networks and Recurrent Networks, as well as combinations of the two.
  3) Running seamlessly on **CPU** and **GPU**

Keras: The Python Deep Learning Library

READ THE DOCUMENTATION AT keras.io

Keras is compatible with Python 2.7 / 3.6.

- The core data structure of **Keras** is a model, namely a way to manage layers easily. The simplest type of model can be implemented by the class ***Sequential***, which enables to build a model as a linear stack of layers.

- For more complex topologies of neural networks, implementing parallel layers for instance, we need the ***Keras Functional API***, enabling to build arbitrary graphs of layers

- Example of implementation of **Sequential Model:**

```
from keras.models import Sequential
model = Sequential()
```

Getting started: 30 seconds to Keras Library

- Stacking layers is easy by the method .add of the class Sequential.

```
from keras.layers import Dense
model = Sequential()
model.add(Dense(units = 64, activation = ‘relu’, input_dim = 100))
model.add(Dense(units = 10, activation = ‘softmax’)
```

- After the model definition, it is necessary to configure the learning process with the method **.compile**

```
model.compile(loss=‘categorical_crossentropy’,
              optimizer=‘sgd’, metrics = [‘accuracy’])
```

Getting started: 30 seconds to Keras Library

- After compiling the neural model it is possible to iterate on training data in batches by using **.fit** method.

```
model.fit(x_train, y_train, epochs = 5, batch_size = 32)
```

- Alternatively, we can train the model on each batch manually, by means of **.train_on_batch**.

```
model.train_on_batch(x_batch, y_batch)
```

- Then we need to evaluate our model performance on a test set for example.

```
loss_and_metrics = model.evaluate(x_test, y_test, batch_size = 128)
```


- **NOTE:** in case of evaluation or prediction the batch size can be any size, and it does not affect the accuracy of the model such as in the training stage. In test phase batch_size depends exclusively on the size of GPU. Batch_size hyperparameter so can scale up the test stage according to the computational

Getting started: 30 seconds to Keras Library

- At this point, we can adopt the trained model in order to predict categorical variables (classification) or continuous variables (regression). For this purpose, we need the ***.predict*** method of the class **Sequential**.

```
classes = model.predict(x_test, batch_size = 128)
```

- This way, it is simple and fast to build a neural chatbot or a question-answer system, an image classification model, a text classifier or any other model.

- The idea behind keras is that Deep Learning implementations are not painful anymore.

Keras Sequential Model

The `Sequential` model is a linear stack of layers.

You can create a `Sequential` model by passing a list of layer instances to the constructor:

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

model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

You can also simply add layers via the `.add()` method:

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

Keras Sequential Model

Python for Deep Learning
Francesco Pugliese

Compilation

Before training a model, you need to configure the learning process, which is done via the `compile` method. It receives three arguments:

- An optimizer. This could be the string identifier of an existing optimizer (such as `rmsprop` or `adagrad`), or an instance of the `Optimizer` class. See: optimizers.
- A loss function. This is the objective that the model will try to minimize. It can be the string identifier of an existing loss function (such as `categorical_crossentropy` or `mse`), or it can be an objective function. See: losses.
- A list of metrics. For any classification problem you will want to set this to `metrics=['accuracy']`. A metric could be the string identifier of an existing metric or a custom metric function.

```
# For a multi-class classification problem
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# For a binary classification problem
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# For a mean squared error regression problem
model.compile(optimizer='rmsprop',
              loss='mse')

# For custom metrics
import keras.backend as K

def mean_pred(y_true, y_pred):
    return K.mean(y_pred)

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

Keras Sequential Model

Python for Deep Learning
Francesco Pugliese

Training

Keras models are trained on Numpy arrays of input data and labels. For training a model, you will typically use the `fit` function. Read its documentation here.

```
# For a single-input model with 2 classes (binary classification):

model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Generate dummy data
import numpy as np
data = np.random.random((1000, 100))
labels = np.random.randint(2, size=(1000, 1))

# Train the model, iterating on the data in batches of 32 samples
model.fit(data, labels, epochs=10, batch_size=32)
```

```
# For a single-input model with 10 classes (categorical classification):

model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Generate dummy data
import numpy as np
data = np.random.random((1000, 100))
labels = np.random.randint(10, size=(1000, 1))

# Convert labels to categorical one-hot encoding
one_hot_labels = keras.utils.to_categorical(labels, num_classes=10)

# Train the model, iterating on the data in batches of 32 samples
model.fit(data, one_hot_labels, epochs=10, batch_size=32)
```

Keras Sequential Model

Python for Deep Learning
Francesco Pugliese

Multilayer Perceptron (MLP) for multi-class softmax classification:

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD

# Generate dummy data
import numpy as np
x_train = np.random.random((1000, 20))
y_train = keras.utils.to_categorical(np.random.randint(10, size=(1000, 1)), num_classes=10)
x_test = np.random.random((100, 20))
y_test = keras.utils.to_categorical(np.random.randint(10, size=(100, 1)), num_classes=10)

model = Sequential()
# Dense(64) is a fully-connected layer with 64 hidden units.
# in the first layer, you must specify the expected input data shape:
# here, 20-dimensional vectors.
model.add(Dense(64, activation='relu', input_dim=20))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=['accuracy'])

model.fit(x_train, y_train,
          epochs=20,
          batch_size=128)
score = model.evaluate(x_test, y_test, batch_size=128)
```

DEEP LEARNING LESSONS

Python for Deep Learning

Francesco Pugliese, PhD
Data Scientist at ISTAT
francesco.pugliese@istat.it

Thank You