

# Advanced School in Artificial Intelligence

## Introduction to Neural Networks

**Ing. Zese Riccardo**  
**[riccardo.zese@unife.it](mailto:riccardo.zese@unife.it)**

*Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021*



**Università  
degli Studi  
di Ferrara**

## Outline

- Introduction to Python
- Introduction to Neural Networks
- Convolutional NN
- Recurrent NN
- Autoencoders and self supervised learning

## Sources

- These slides are taken from:

- Intel Nervana AI Academy Instructional Content

Intel Legal Notices and Disclaimers

This presentation is for informational purposes only. INTEL MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at [intel.com](http://intel.com).

This sample source code is released under the [Intel Sample Source Code License Agreement](#).

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

\*Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.

- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

[https://www.deeplearningbook.org/lecture\\_slides.html](https://www.deeplearningbook.org/lecture_slides.html)

- Chapter “Neural Networks” of “A Course in Machine Learning” by Hal Daume III.

<http://ciml.info/>

- Some parts from “CS231n: Convolutional Neural Networks for Visual Recognition”, Stanford University

<http://cs231n.stanford.edu/>

## Outline

- Introduction to Python
- Introduction to Neural Networks
- Convolutional NN
- Recurrent NN
- Autoencoders and self supervised learning

## NN Frameworks

- There are many frameworks available for working with NNs.
  - Tensorflow
  - CNTK
  - Theano
  - PyTorch
  - Caffe
  - ...



## NN Frameworks

- An interesting library is Keras (<https://keras.io/>).
- From the website:

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

## Keras

- Contains a set of classes that abstracts the underlying NN library giving a unified interface to the systems Tensorflow, CNTK, and Theano.
- In practice, you can use Keras to define a network and run it using one of the three systems above without changing anything in the code.
- Written in Python, allows users to configure complicated models directly in Python
- Uses either CPU or GPU for computation
- Uses numpy data structures and a similar command structure to scikit-learn (model.fit , model.predict, etc.)

## Keras and Tensorflow



- Recently, Tensorflow adopted Keras as its official frontend to simplify the definition and the management of the code.
  - It contains keras as a submodule.
- Before version 2, users could choose whether to use the original Tensorflow interface and functions or use the Keras module
- From version 2, the actual one, Tensorflow has been redesigned to use Keras only. Old code that does not use Keras cannot be executed if a previous version of Tensorflow is not installed.



## Tensorflow

- In this course we will use Tensorflow from version 2, therefore we will use the Keras API (<https://www.tensorflow.org/>).
- To install Tensorflow simply run the command  
**`pip install tensorflow`**

## Should I use directly Keras or Tensorflow's implementation?

- Keras is an API that works with different NN engines. This means that it contains the mapping from its methods to those of the different libraries.
- Keras defines common methods while it is possible to have functionalities from the underlying libraries which are not mapped in Keras (because they are specific of only one library for example)

## Should I use directly Keras or Tensorflow's implementation?

- Tensorflow's implementation of Keras contains only the parts that work with Tensorflow's methods and methods for those functions not available in other libraries that could implement optimizations to improve performance.
- The choice is yours and depends on what you want to do:
  - If you want to work only with Tensorflow you can use directly its implementation.
  - If you are willing to install and test other libraries (CNTK and Theano) it is worth installing Keras and using it.
  - In general, unless you need to work with very specific methods, you won't see any difference in using directly Keras or using it through Tensorflow.

## Other libraries

- CNTK: <https://docs.microsoft.com/cognitive-toolkit/>
- Theano: <http://deeplearning.net/software/theano/>
- PyTorch: <https://pytorch.org/>
- Caffe: <https://caffe.berkeleyvision.org/>
- Each of them come with tutorials, examples and vast documentation.
- Feel free to download and test them.
- Feel free to use them in your exam project.

## Tensorflow

```
import tensorflow as tf

x_train, y_train, (x_test, y_test) = tf.keras.datasets.mnist.load_data()

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test, y_test)
```



## Tensorflow

```
import tensorflow as tf

x_train, y_train, (x_test, y_test) = tf.keras.datasets.mnist.load_data()

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test, y_test)
```

Downloads the MNIST dataset and loads the dataset dividing it in training and test sets



## Tensorflow

```
import tensorflow as tf
```

```
x_train, y_train, (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

```
model = tf.keras.models.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28, 28)),  
    tf.keras.layers.Dense(512, activation=tf.nn.relu),  
    tf.keras.layers.Dropout(0.2),  
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)  
])
```

Definition of the model

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',  
              metrics=['accuracy'])  
  
model.fit(x_train, y_train, epochs=5)  
  
model.evaluate(x_test, y_test)
```

## Tensorflow

```
import tensorflow as tf

x_train, y_train, (x_test, y_test) = tf.keras.datasets.mnist.load_data()

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test, y_test)
```

Definition of the optimizer, loss function  
and metrics to consider and print.

## Tensorflow

```
import tensorflow as tf

x_train, y_train, (x_test, y_test) = tf.keras.datasets.mnist.load_data()

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Training of the net. We don't have to compute loss or any other metric. The fit method will do the work for us.

## Tensorflow

```
import tensorflow as tf

x_train, y_train, (x_test, y_test) = tf.keras.datasets.mnist.load_data()

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])


model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test, y_test)
```

The test set is used to evaluate the trained model.

## Typical Command Structure in Tensorflow

- Build the structure of your network.
- Compile the model, specifying your loss function, metrics, and optimizer (which includes the learning rate).
- Fit the model on your training data (specifying batch size, number of epochs)
- Predict on new data
- Evaluate your results 

## Building the model

- Keras provides two approaches for building the structure of your model:
  - **Sequential Model**: it allows a linear stack of layers – simpler and more convenient if the model has this form
  - **Functional API**: more detailed and complex, but it allows more complicated architectures
- We will focus on the Sequential Model.



## Way to scale input

- Normalization

- Linear scaling to interval [0,1]

$$x_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

- Linear scaling to interval [-1,1]

$$x_i = 2 \left( \frac{x_i - \bar{x}}{x_{max} - x_{min}} \right) - 1$$

- Adds numerical stability and inputs on the same scale → **Good Practice!**

## Way to scale input



- Standardization (making variable approx. std. normal)

$$x_i = \frac{x_i - \bar{x}}{\sigma}$$

$\bar{x}$  = mean

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$$

## Standardization vs Normalization


- Standardizing the features is important
  - if we are comparing measurements that have different units (e.g. feet, kilometers, and hours) that, in turn, may mean the variables have different scales,
  - it is also a general requirement for many machine learning algorithms → if we consider gradient descent certain weights may update faster than others since the feature values  $x_j$  play a role in the weight updates
- Normalizing is important
  - in image processing, where pixel intensities have to be normalized to fit within a certain range (i.e., 0 to 255 for the RGB color range).
  - typical neural network algorithm require data that on a 0-1 scale.

## Standardization vs Normalization

- Standardization and Normalization are created to achieve a similar target, i.e., build features that have similar ranges to each other.
  - Widely used in data analysis to help the data scientists to get some clue out of the raw data.



## Batching Terminology

- An **Epoch** refers to a single pass through all of the training data. 
- In full batch gradient descent, there would be one step taken per epoch.
- In SGD / Online learning, there would be  $n$  steps taken per epoch ( $n$  = training set size)
- In **Minibatch** there would be  $(n/\text{batch size})$  steps taken per epoch
- When training, it is common to refer to the number of epochs needed for the model to be “trained”.