Introduzione al Machine Learning

Gianni Pantaleo

Dipartimento di Ingegneria dell'Informazione, DINFO
Università degli Studi di Firenze
Via S. Marta 3, 50139, Firenze, Italy
gianni.pantaleo@unifi.it

DISIT Lab
https://www.disit.org

MASTER: Big Data Analytics And Technologies For Management – *MaBiDa*
2022

Introduzione al Machine Learning - *Lineup*

1. *Introduzione*
2. *Machine Learning con TensorFlow per Python*
3. *Applicazioni*
4. *Deep Learning*

Introduzione al Machine Learning - *Lineup*

1. Introduzione ←

2. Machine Learning con TensorFlow per Python

3. Applicazioni

4. Deep Learning

1. Introduzione – *Machine Learning*: Alcune Definizioni

- « Machine Learning is a field of study that gives computers the ***ability to learn without being explicitly programmed***. » *(A. L. Samuel)*
  - ✓ *« A computer **can be programmed so that it will learn to play a better game** of checkers **than can be played by the person** who wrote the program ».*

    …e…
  - ✓ *« **Programming computers to learn from experience** should eventually eliminate the need for much of this detailed programming effort. »*

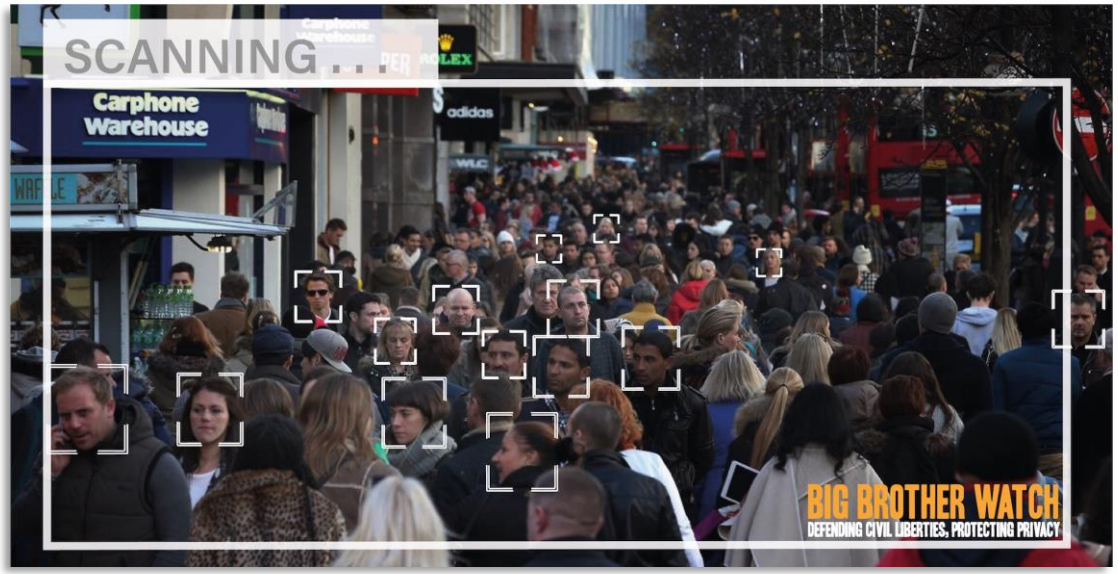
    (A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers", 1959)

- « A computer program is said to learn from ***experience E*** with respect to some class of ***tasks T*** and ***performance measure P***, if its performance at tasks in ***T***, as measured by ***P***, improves with experience ***E***. »

  (T. M. Mitchell, "Machine Learning", McGraw Hill, 1997)

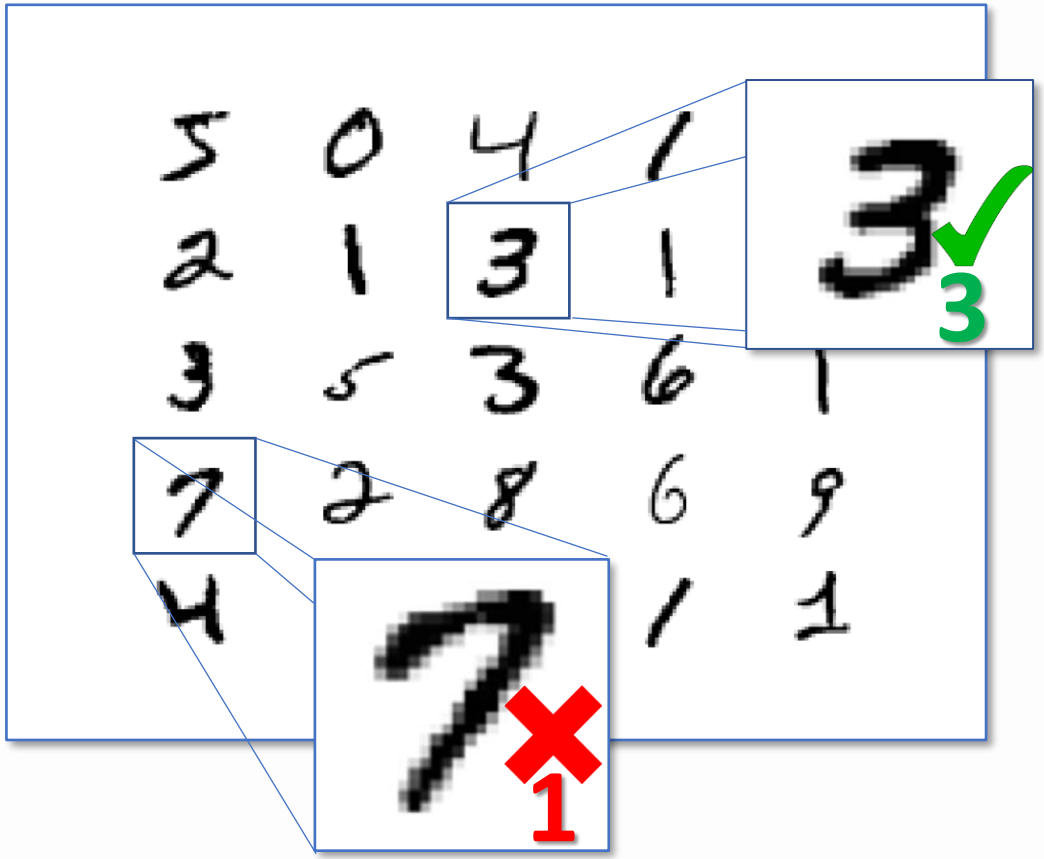
1. Introduzione – Applicazioni del *Machine Learning*

- Computer Vision
  - *Face/objects Recognition and Tracking*



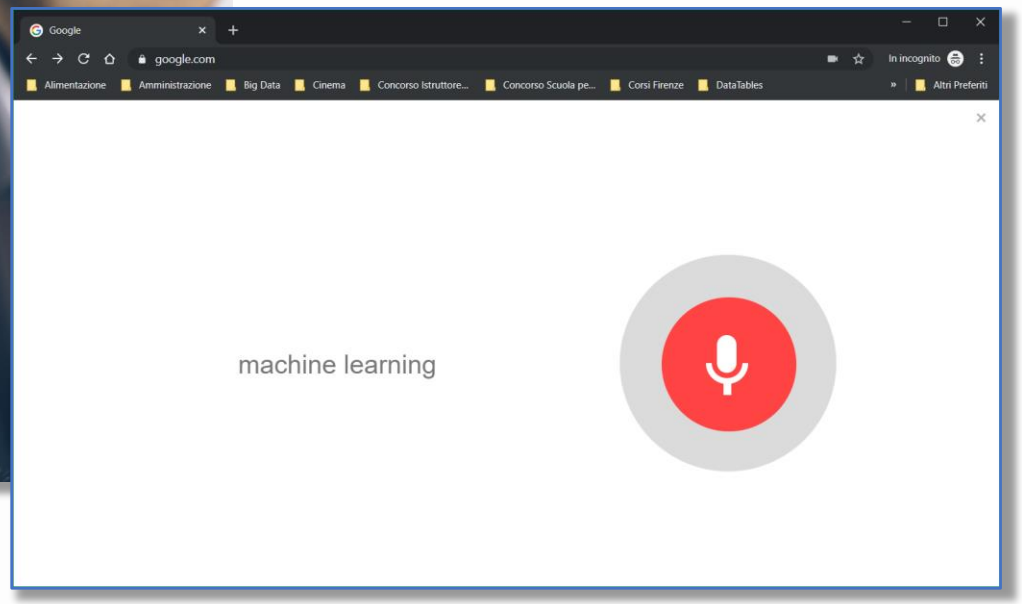
Fonte: https://www.wired.it/attualita/tech/2019/02/16/riconoscimento-facciale-cina-ungheria-liberta/

  - *Handwriting characters recognition*



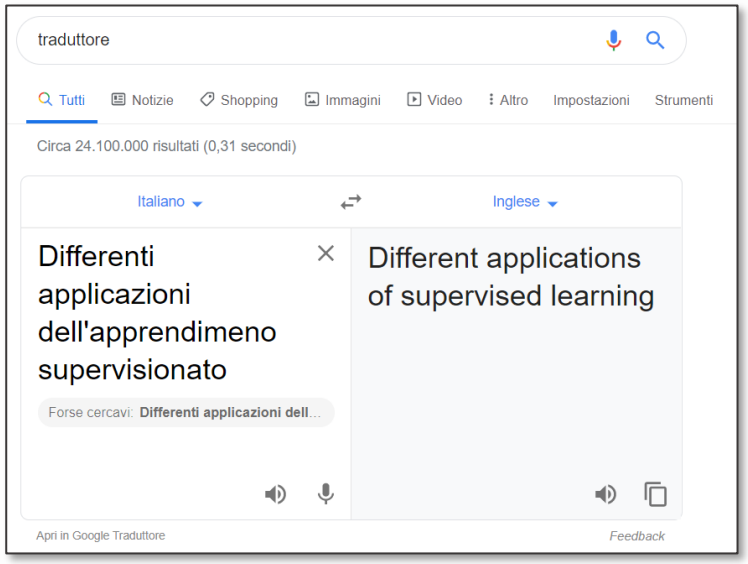
1. Introduzione – Applicazioni del *Machine Learning*

- Speech-to-Text / Voice Recognition

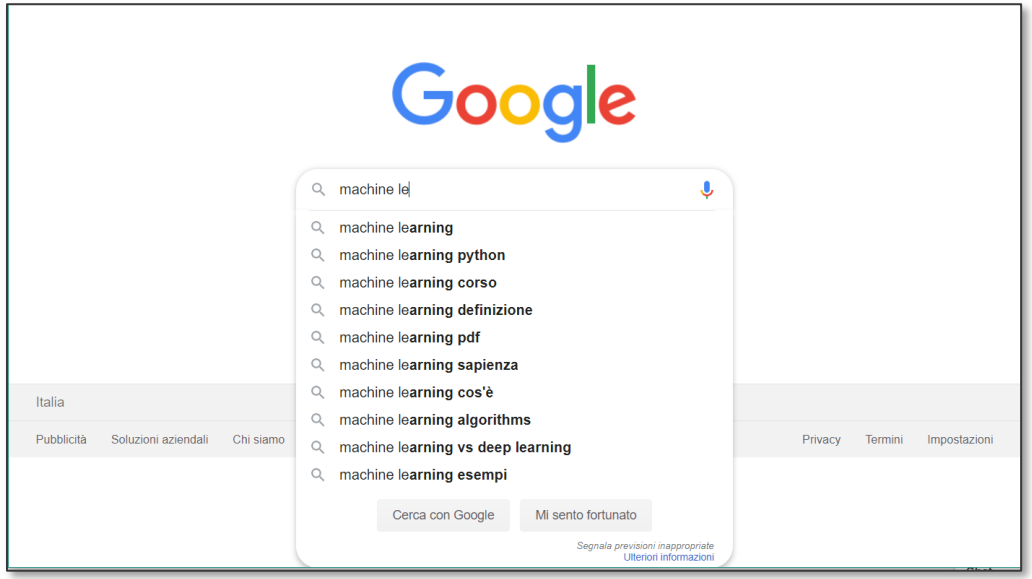


1. Introduzione – Applicazioni del *Machine Learning*

- Natural Language Processing
  - *Traduzioni automatiche*



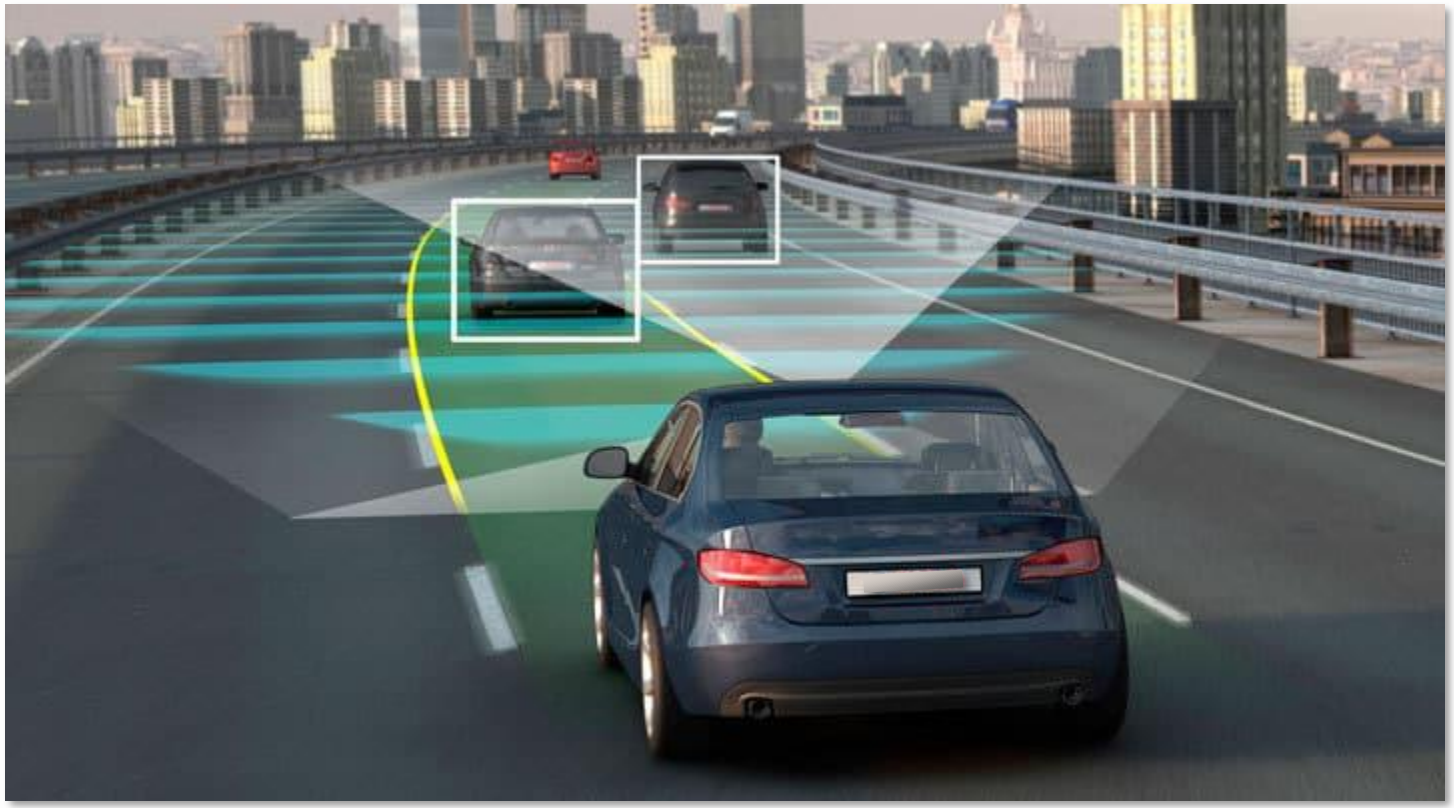
  - *Autocomplete*



  - *Text Summarization*
  - *Email Spam Filtering*

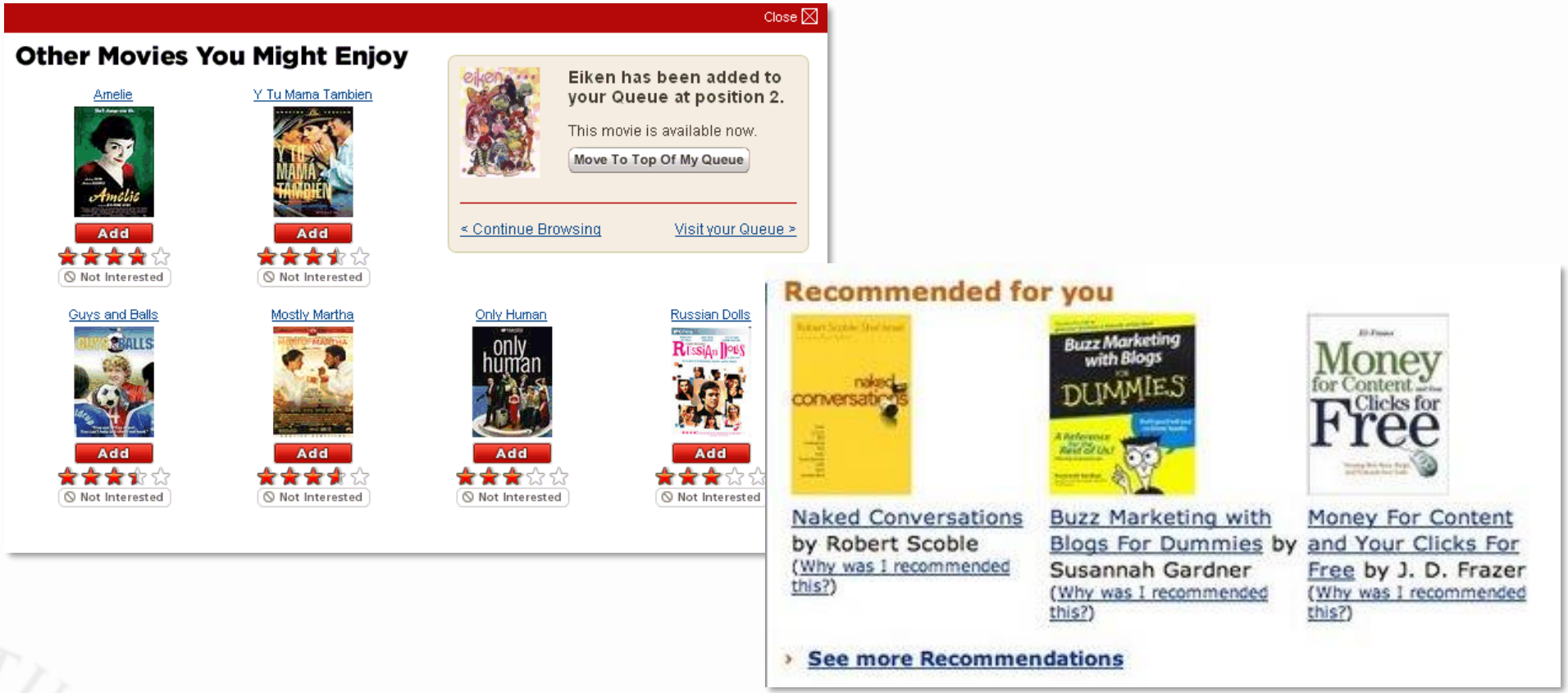
1. Introduzione – Applicazioni del *Machine Learning*

- Robotics, autonomous driving



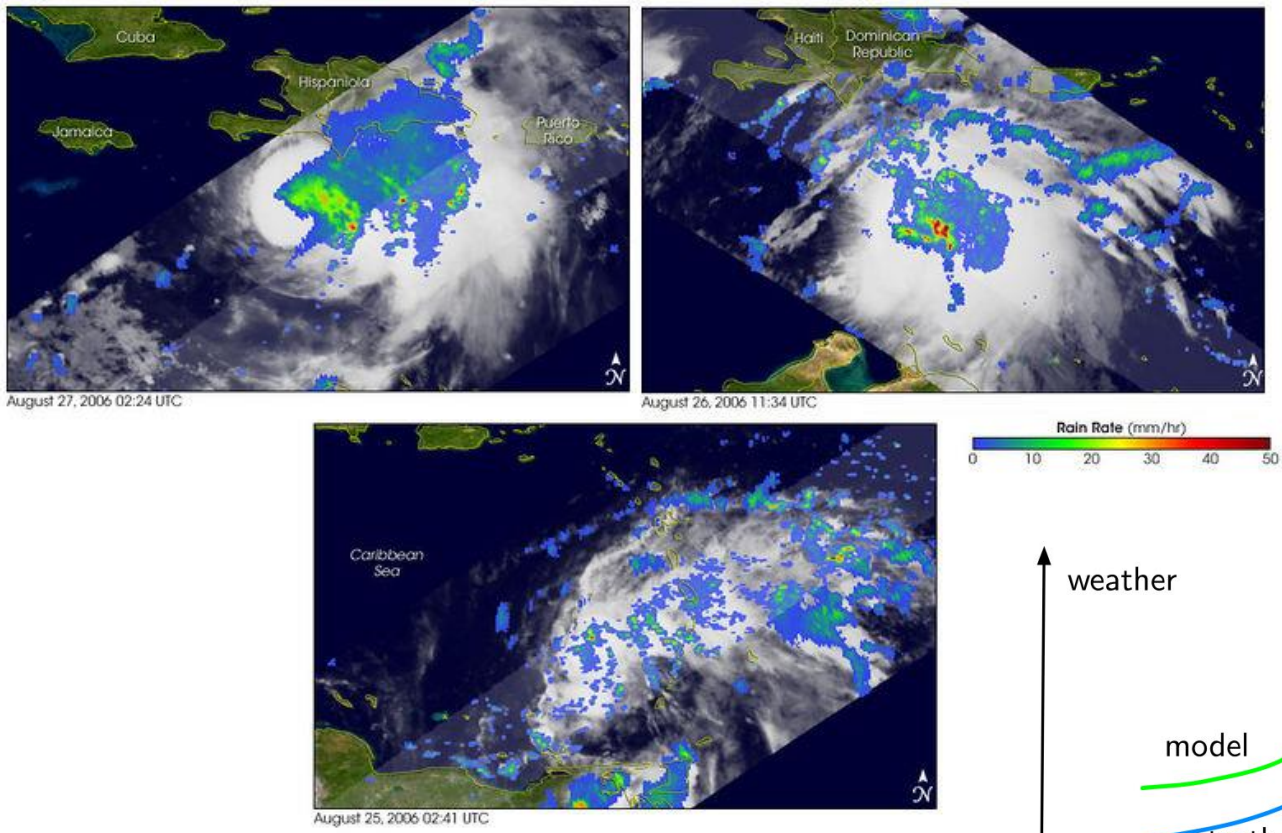
1. Introduzione – Applicazioni del *Machine Learning*

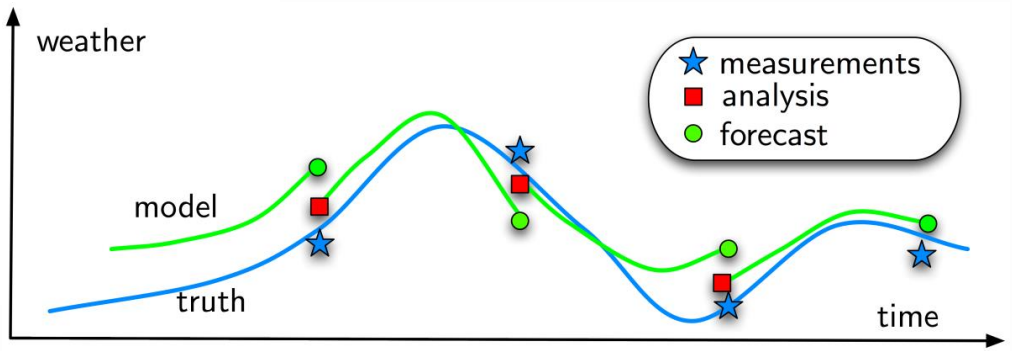
- Recommendation Systems



1. Introduzione – Applicazioni del *Machine Learning*

- Modelli predittivi





1. Introduzione – In quali ambiti usare il *Machine Learning*

- Stima di modelli e funzioni che mappano set ben definiti di dati in input con set ben definiti di dati in output.

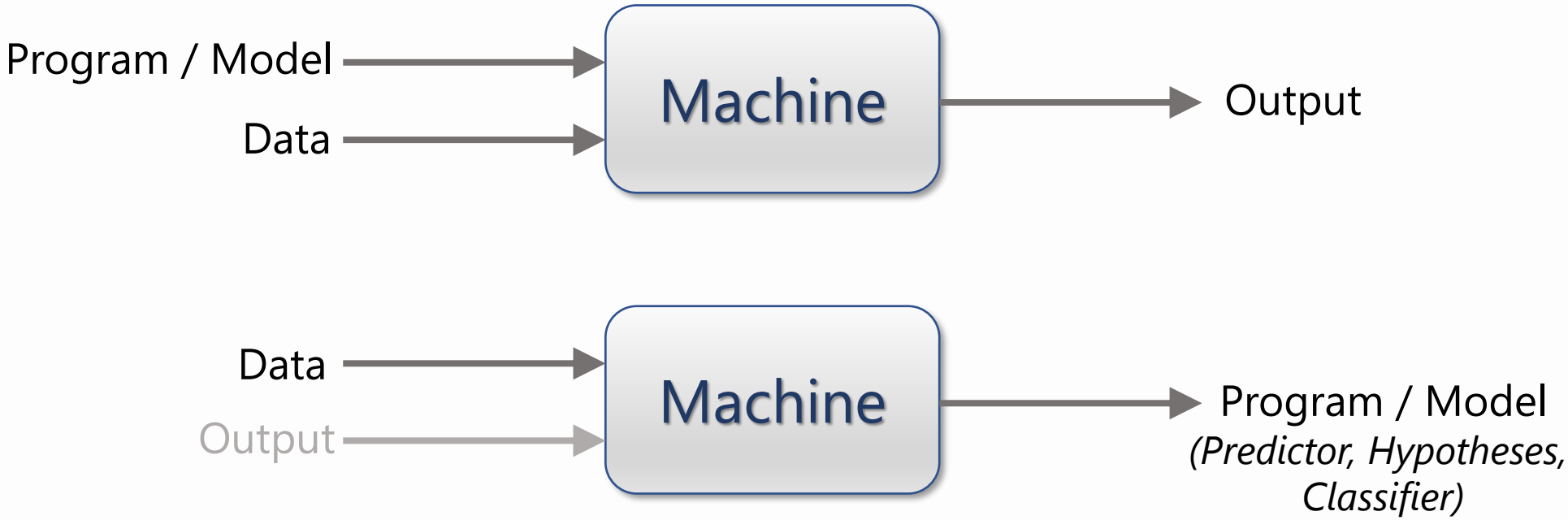
- Creazione di modelli predittivi agnostici, ricerca di pattern e correlazioni a partire unicamente dai dati.

- Stima di processi complessi di cui non si ha conoscenza esplicita e in cui non sono noti modelli a priori (ad es. l'automazione di processi cognitivi come *Speech recognition*, *Computer Vision*…).

- Produzione di modelli di classificazione quando si hanno a disposizione un numero molto elevato di dati (Big Data).

1. Introduzione – *Machine Learning* VS Computazione Tradizionale

Program complexity VS Data complexity



1. Introduzione – Il problema dell'Apprendimento

- Dati due spazi di variabili: ***input*** $\boldsymbol{x} = x_1, x_2, \dots, x_N$ e ***output*** $\boldsymbol{y} = y_1, y_2, \dots, y_N$
- Supponendo che esista una relazione f non nota (***target function***) tra $\boldsymbol{x}$ e $\boldsymbol{y}$:

$$\boldsymbol{y} = f(\boldsymbol{x})$$

- Attraverso l'algoritmo di apprendimento si vuole determinare una funzione h che approssimi al meglio la funzione target f:

$$\hat{\boldsymbol{y}} = h(\boldsymbol{x}) + \varepsilon$$

  dove ε è una misura dell'errore tra $\hat{\boldsymbol{y}}$ (valore predetto dal modello) e $\boldsymbol{y}$.

- L'obiettivo, dunque, è quello di minimizzare ε, in modo che la funzione h possa essere utilizzata per predire nuove istanze delle variabili di output a partire da variabili di input non osservati in precedenza.

1. Introduzione – Classificazione degli Algoritmi di Apprendimento

Classificazione dei processi di apprendimento attraverso cui stimare una funzione (non nota) f che descriva pattern e/o relazioni nascoste tra data set di input e output attesi:

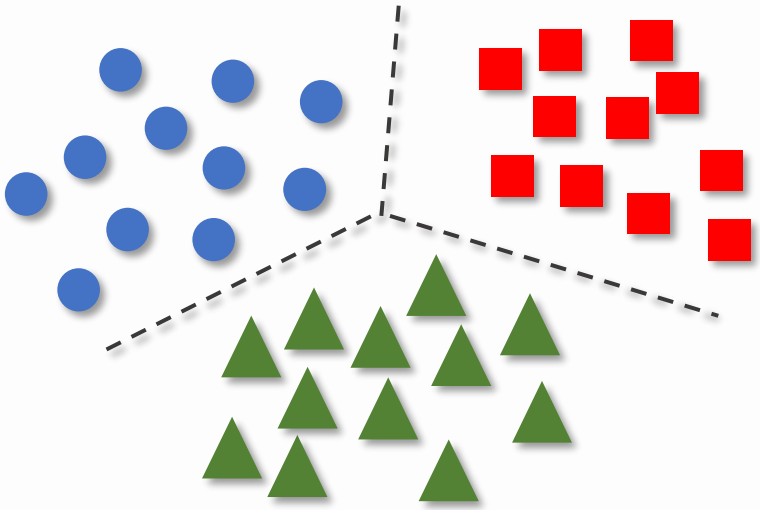
- **Supervised learning:** l'algoritmo produce un modello h (che approssima f secondo opportuni criteri di minimizzazione dell'errore) addestrato con un data set (*training* data set) annotato $\mathbb{T} = \{(x_i, y_i)\}_{i=1}^{N}$:

$$\boldsymbol{y} = h(\boldsymbol{x}) + \varepsilon$$

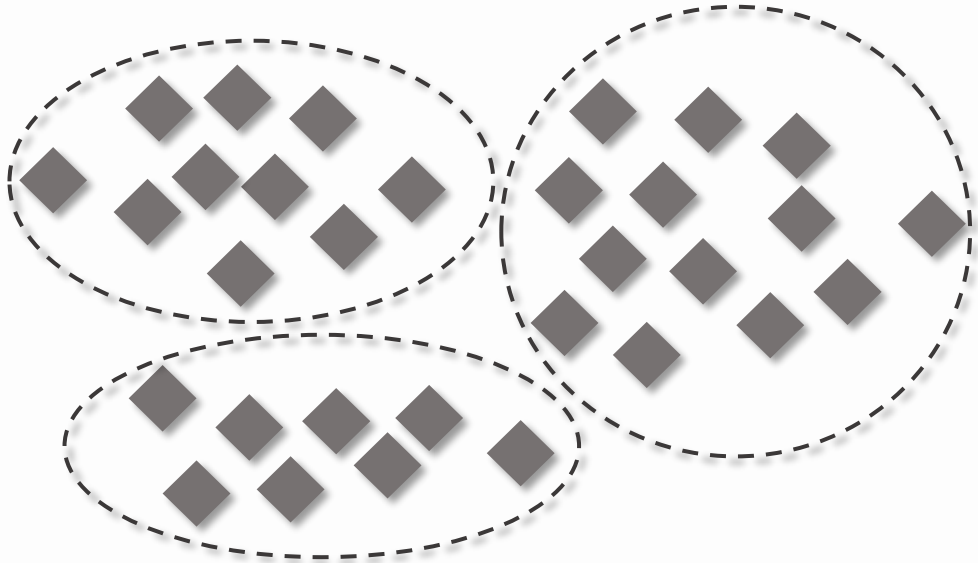
$$\boldsymbol{x} = x_1, x_2, \dots, x_N, \qquad \boldsymbol{y} = y_1, y_2, \dots, y_N$$

- **Unsupervised learning:** l'algoritmo riceve in input un set di dati non annotati $\{x_i\}_{i=1}^{N}$ con l'intento di trovare al suo interno pattern, strutture e relazioni per eseguire task di classificazione, raggruppamento ecc.
- **Semi-Supervised Learning:** l'algoritmo produce un modello sulla base di un data set in cui sono presenti una minor parte di dati annotati e una maggior parte di dati non annotati.
- **Reinforcement learning:** l'algoritmo produce una sequenza di decisioni basate sul feedback (*reward/punishment*) delle decisioni precedenti.

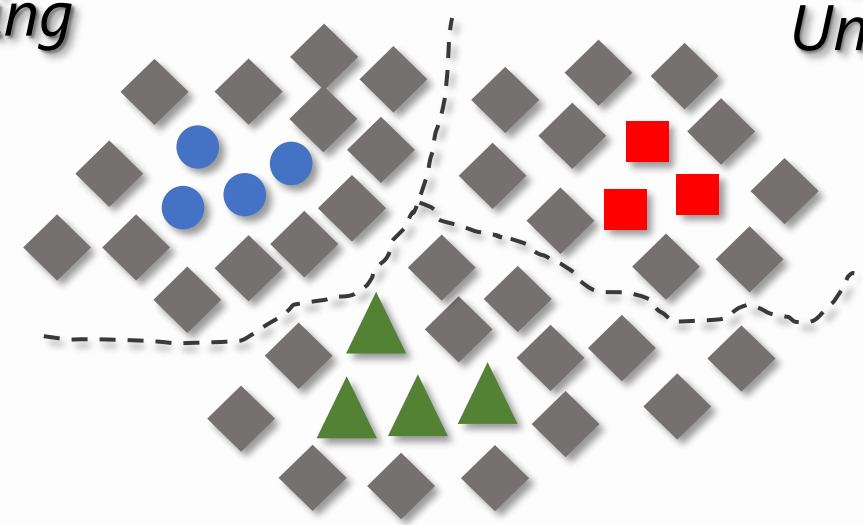
1. Introduzione – Classificazione degli Algoritmi di Apprendimento



Supervised Learning



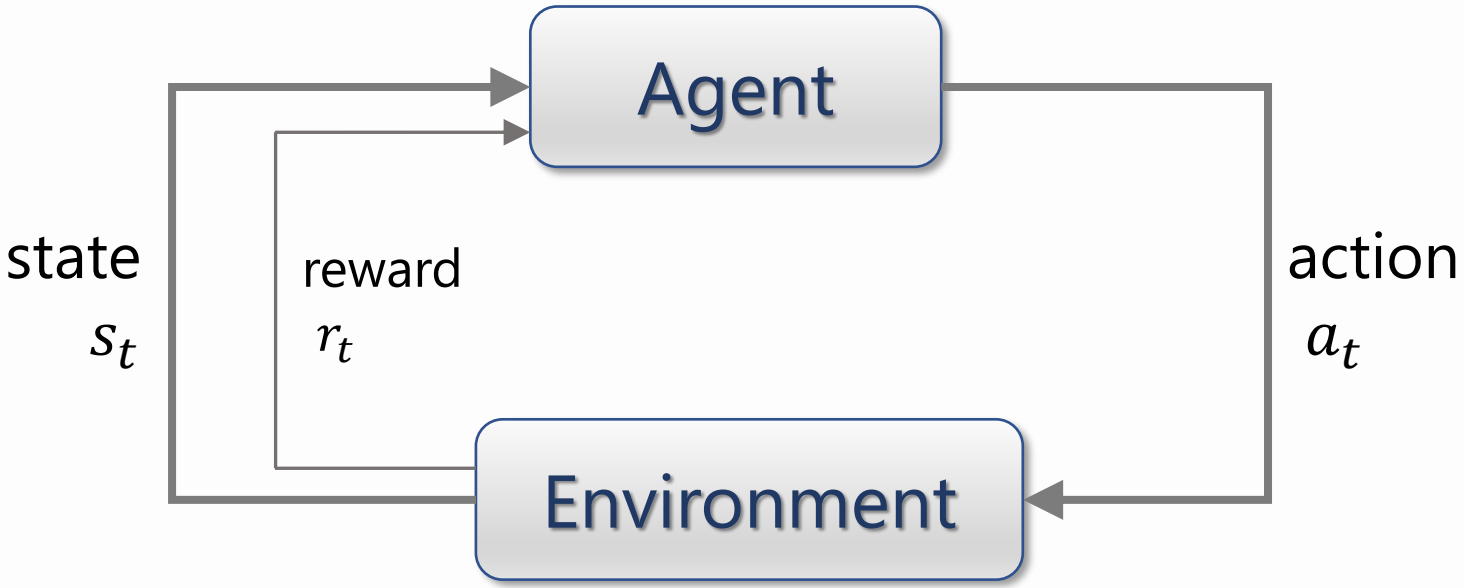
Unsupervised Learning



Semi-Supervised Learning

1. Introduzione – Classificazione degli Algoritmi di Apprendimento

Reinforcement Learning



1. Introduzione – Classificazione degli Algoritmi di Apprendimento

- Un tipico algoritmo di ***Reinforcement Learning*** prevede l'iterazione ciclica dei seguenti passi (fino al raggiungimento di una condizione finale):

1. Ad istanti discreti t $(t_0, t_1, \dots, t_N)$, denominati anche time steps o episodi, l'***Agent*** riceve informazioni che descrivono una rappresentazione dello stato (***state***) dell'ambiente (***Environment***) $S_t \in \mathbb{S}$ (dove $\mathbb{S}$ rappresenta un set finito di stati).
2. L'Agent utilizza tali informazioni sullo stato (S_t) per selezionare un'azione (***action***) $A_t \in \mathbb{A}(S_t)$, dove $\mathbb{A}(S_t)$ è l'insieme finito delle azioni disponibili per lo stato S_t.
3. L'Agent sceglie tale azione in base a una ***Policy*** π_t, che rappresenta l'azione scelta quando lo stato assume un certo valore, $S_t = s$. Tale policy può essere deterministica o stocastica.
4. All'istante successivo $t + 1$, l'Agent riceve una ricompensa (***Reward***) R_{t+1} come conseguenza dell'azione A_t. Inoltre, l'Agent riceve anche le informazioni sul nuovo stato S_{t+1}. Il ciclo così si ripete dal punto 1. fino al raggiungimento della condizione finale.

Introduzione al Machine Learning - *Lineup*

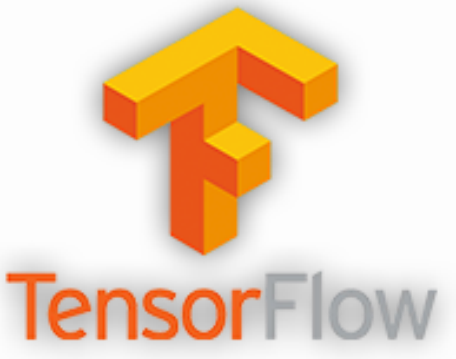
1. ***Introduzione***
2. ***Machine Learning con Python*** ←
3. *Applicazioni*
4. *Deep Learning*

2. Machine Learning con Python – Librerie

Librerie Python per Data Analytics e Machine Learning:

- **Pandas** – Libreria per la gestione e la trasformazione di dati; permette la gestione e manipolazione di set di dati di grandi dimensioni, serie temporali ecc.
- **NumPy** – Libreria per il calcolo matematico; ampia collezione di funzioni matematiche e gestione di array e matrici multidimensionali.
- **SciPy** – Libreria per il calcolo statistico e calcolo numerico.
- **Matplotlib** – Libreria 2D per la visualizzazione dei dati e produzione di grafici.
- **Scikit-Learn** – Libreria per il Machine Learning che offre l'implementazione di molti modelli di supervised e unsupervised learning.
- **TensorFlow, Keras** – Librerie e API per il Deep Learning e l'implementazione di modelli basati su Neural Networks

2. Machine Learning con Python – TensorFlow



- ***TensorFlow*** è una delle librerie free open source attualmente più utilizzate per Machine Learning e Deep Learning
- Sviluppata da Google (Google Brain) in ambito di ricerca e commerciale, alla base di molte applicazioni come riconoscimento vocale, virtual assistant, recommendation system…

- Il nome "TensorFlow" deriva dalle operazioni di calcolo computazionale eseguite dalle reti neurali su array di dati multidimensionali (***tensori***).
- Offre prestazioni elevate e supporto per varie piattaforme e architetture computazionali (CPU, GPU, TPU).
- Espone API per vari linguaggi, tra cui Python, C++, Java, Go, R…
- La versione 2 (introdotta nel Settembre 2019) offre miglioramenti nelle prestazioni, supporto multi-GPU e supporto TPU. Necessari alcuni accorgimenti per la retro-compatibilità con il codice scritto con le versioni 1.x

2. Machine Learning con Python – Scelta del Package Manager

Quale *Package Manager* scegliere per l'installazione di Python con TensorFlow?

pip VS Anaconda *(conda)*

- ***Pip*** è il tool di default per l'installazione di packages dal Python Package Index, ***PyPI***.

- ***Anaconda*** è una suite di packages e librerie (creata e distribuita da *Continuum Analytics*), molte delle quali utili in ambiti di Data Science.
- ***Conda*** è il package manager di Anaconda.

- ***Pip*** necessita come requisito l'installazione di un interprete ***Python***.

- ***Conda*** può installare l'interprete ***Python*** direttamente.

- Usando ***Pip*** è necessario creare dei virtual environment (***virtualenv***, ***venv***) per utilizzare versioni differenti di uno stesso package.

- ***Conda*** crea e gestisce automaticamente virtual environment, sostituendo ***virtualenv***.
- Oltre a Python, ***Conda*** supporta packages anche per altri linguaggi (Ruby, Java…)
- ***Anaconda*** può utilizzare anche ***Pip*** come package manager.

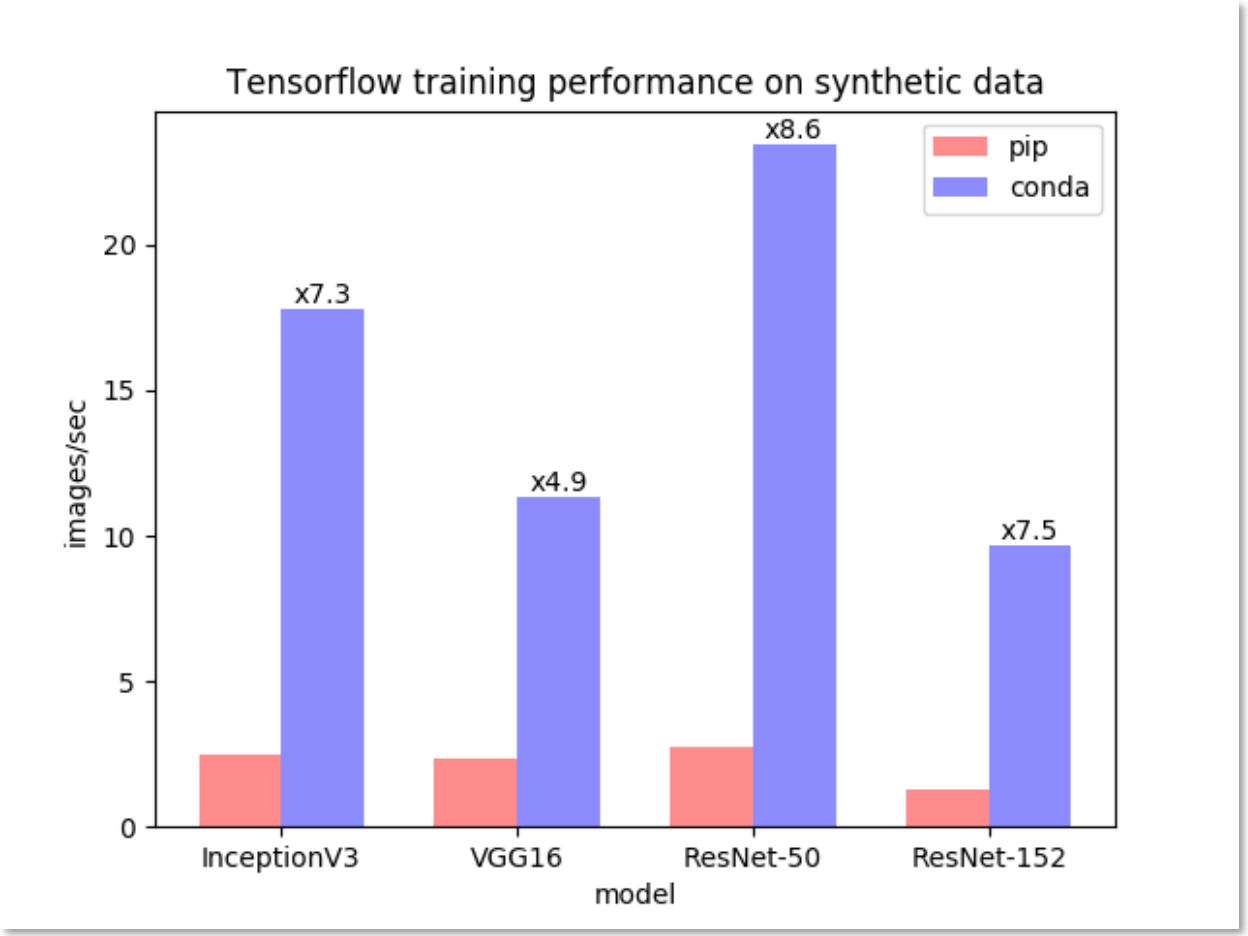
2. Machine Learning con Python – Scelta del Package Manager

pip VS Anaconda *(conda)*

~ Prestazioni con TensorFlow ~

Inception V3: modello di image classifier

VGG16: modello basato su *Convolutional Neural Networks* per Large Scale Image classification



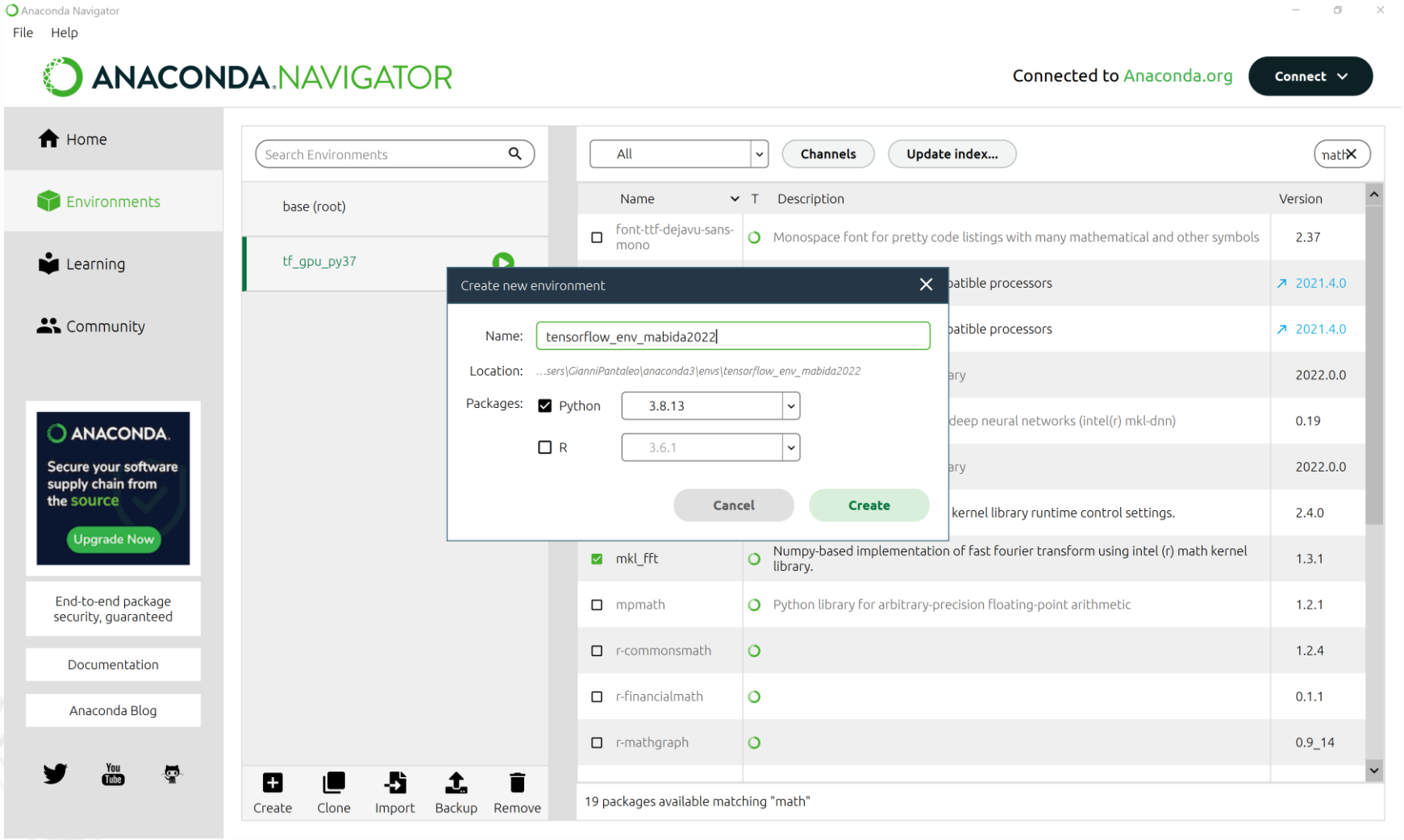
ResNet-50: 50-layers Residual Neural Network

ResNet-152: 152-layers Residual Neural Network

Fonte: https://www.anaconda.com/tensorflow-in-anaconda/

2. Machine Learning con Python – Ambiente di Sviluppo

- Creare un Virtual Environment con Anaconda Manager:



2. Machine Learning con Python – Ambiente di Sviluppo

- Creare un Virtual Environment da terminale:
- Da Anaconda Navigator, aprire un terminale conda, e digitare i seguenti comandi per creare ed attivare un nuovo virtual environment da riga di comando:

```
conda create -n tensorflow_env_mabida2022
conda activate tensorflow_env_mabida2022
```

- Per creare un environment con una specifica versione di Python:

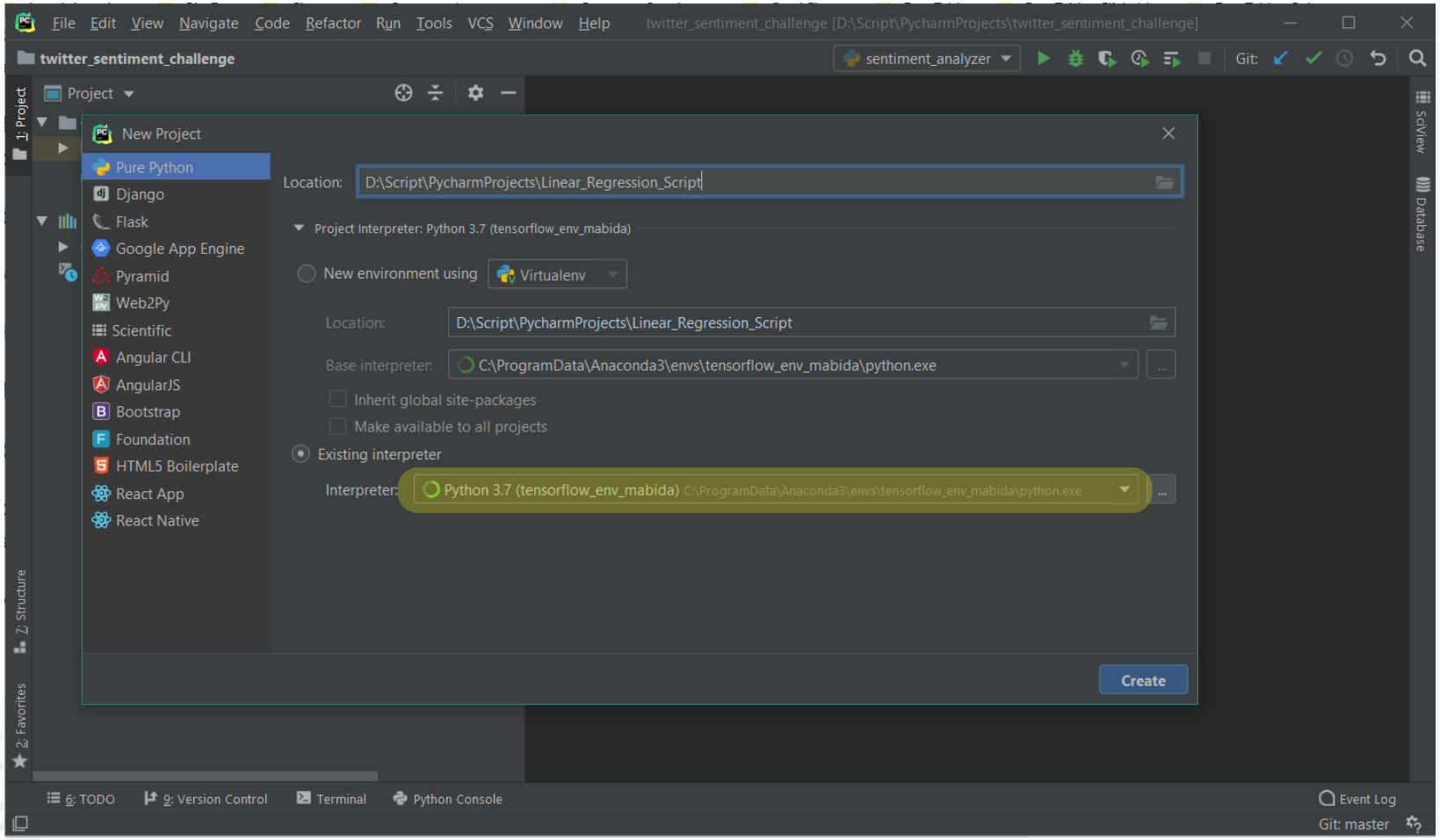
```
conda create -n myenv python=3.8
```

- Per creare un environment installando anche uno specific package:

```
conda create -n myenv tensorflow
```

2. Machine Learning con Python – Ambiente di Sviluppo

- Installare un ambiente di sviluppo IDE (***I**ntegrated **D**evelopment **E**nvironment*)
- Configurare il virtual environment e l'interprete Python (*conda*) all'interno dell'IDE



2. Machine Learning con Python – TensorFlow, GPU & CUDA

- Installazione di CUDA Toolkit e cuDNN (NVIDIA CUDA Toolkit Documentation):

  Windows: https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html

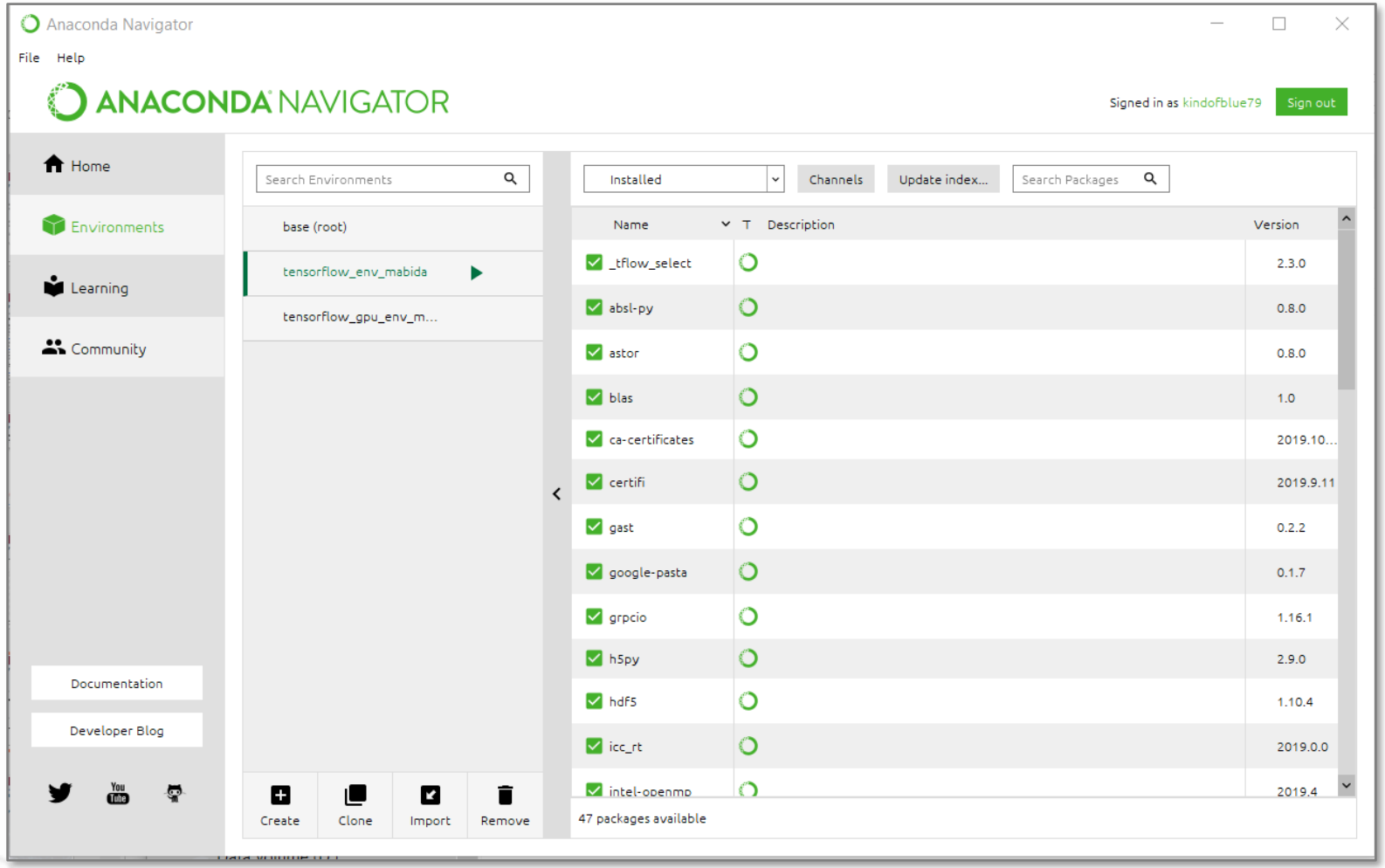
  Linux: https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html

  Mac OS: https://developer.download.nvidia.com/compute/cuda/10.1/Prod/docs/sidebar/CUDA_Installation_Guide_Mac.pdf

  1. Verificare se la GPU della scheda video è compatibile con CUDA: https://developer.nvidia.com/cuda-gpus
  2. Download e Installazione di CUDA Toolkit: https://developer.nvidia.com/cuda-downloads
  3. Download e Installazione di cuDNN (CUDA Deep Neural Network): https://developer.nvidia.com/cudnn

2. Machine Learning con Python – Virtual Environments

Visualizzare i virtual environment in Anaconda Navigator:



2. Machine Learning con Python – Importare ed usare TensorFlow

Importare ed usare TensorFlow in uno script Python

- All'inizio dello script, quando solitamente si caricano le librerie installate da importare:

```
import tensorflow as tf
```

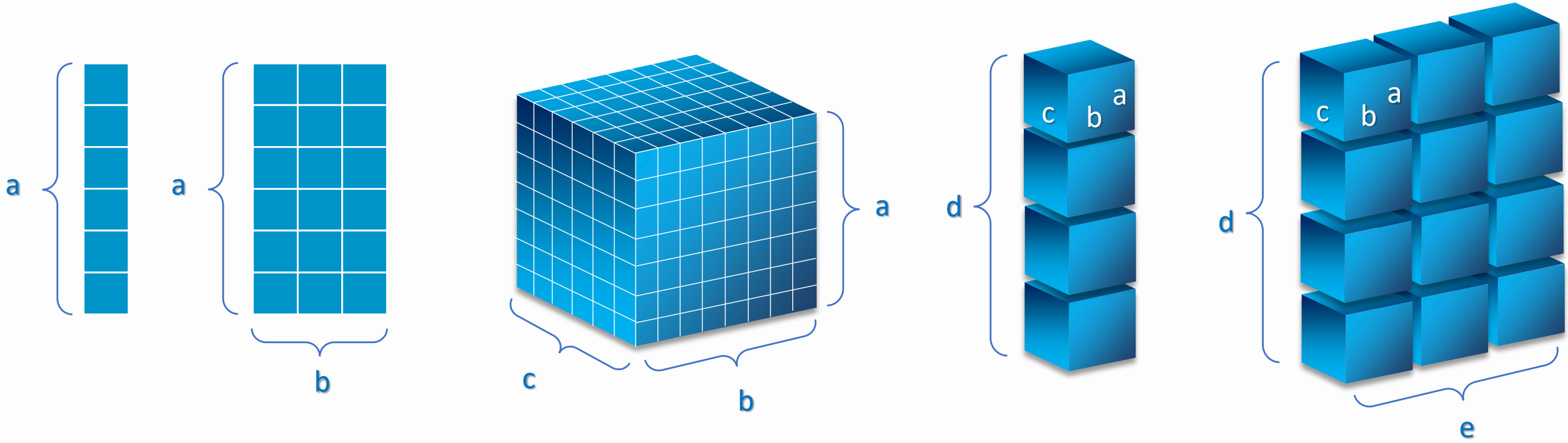
- Con la versione 2 di TensorFlow, si possono usare queste direttive per mantenere la retrocompabilità con gli script scritti con le versioni 1.x:

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

Questo approccio ha il vantaggio di non dover riscrivere il codice, ma contemporaneamente presenta lo svantaggio di non sfruttare le novità introdotte nella versione 2 (tra cui approccio computazionale più chiaro e semplice, monitoraggio e debug migliorati, migliore integrazione con molte librerie, ad esempio Keras,…):

2. Machine Learning con Python – TensorFlow Tensor

- Fisicamente, un tensore è una generalizzazione matematica di un array o matrice multidimensionale.



1D Tensor (Array) | 2D Tensor (Matrix) | 3D Tensor (Cube) | 4D Tensor (Array of Cubes) | 5D Tensor (Matrix of Cubes)

- In TensorFlow, il tensore è la principale entità computazionale, ed è rappresentato dall'oggetto `tf.Tensor`.

2. Machine Learning con Python – TensorFlow Tensor

Principali tipi di tensori

- In TensorFlow un tensore deve contenere elementi dello stesso tipo; il tipo deve essere sempre definito. Quindi un oggetto `tf.Tensor` è caratterizzato dalle seguenti proprietà:
  - un *data type* (float32, float64, int32, string…);
  - una *shape* (ovvero le dimensioni del tensore, che possono anche non essere specificate in fase di inizializzazione con l'uso del parametro **None**).
- Principali tipi di tensori:
  - ***Costanti***: rappresentano tensori a valori costanti; in TensorFlow sono definite tramite l'oggetto `tf.constant`;
  - ***Variabili***: rappresentano tensori a valori dinamici (ad es. i parametri di un modello); sono definite tramite l'oggetto `tf.Variable`;
  - ***Placeholder (TF1.x, non presente in TF2.x)***: rappresentano l'inizializzazione dei tensori che verranno successivamente popolati con i dati (generalmente usati per rappresentare i dati di input in un modello); sono definiti attraverso l'oggetto `tf.placeholder`.

2. Machine Learning con Python – TensorFlow v2: Costanti

Costanti `tf.constant`

https://www.tensorflow.org/api_docs/python/tf/constant

TF v2

```
tf.constant(
    value,
    dtype=None,
    shape=None,
    name='Const',
)
```

value: A constant value (or list) of type ***dtype***.

dtype: The type of the elements of the resulting tensor.

shape: Optional dimensions of resulting tensor.

name: Optional name for the tensor.

2. Machine Learning con Python – TensorFlow v2: Costanti

TF v2

```
import tensorflow as tf

# Inizializza le costanti a e b
a = tf.constant([1, 2, 3], name='constA')
b = tf.constant([3, 2, 1], name='constB')

# Esegue L'operazione nominandola 'Addizione'
result = tf.math.add(a, b, name='Addizione')
result2 = a+b;

# Visualizza il risultato in console
tf.print(result)
print('a + b = ', result)
print('a + b = ', result.numpy())
print('a + b = {}'.format(result))
print('a + b = {}'.format(result2))
```

```
Console ------------------------------------
a + b = [4 4 4]
```

2. Machine Learning con Python – TensorFlow v2: Variabili

Variabili `tf.Variable (class)` https://www.tensorflow.org/api_docs/python/tf/Variable

TF v2

```
Tf.Variable(
    initial_value=None,
    trainable=None,
    validate_shape=True,
    caching_device=None,
    name=None,
    variable_def=None,
    dtype=None,
    import_scope=None,
    constraint=None,
    synchronization=tf.VariableSynchronization.AUTO,
    aggregation=tf.compat.v1.VariableAggregation.NONE,
    shape=None
)
```

Esempi di instanziamento:

```
my_var = tf.Variable([1, 2, 3. 4, 5, 6], name='var1', shape=[2, 3])
```

Initial_value: A Tensor, or Python object convertible to a Tensor, which is the initial value for the Variable.

name: Optional name for the variable. Defaults to 'Variable'.

dtype: If set, initial_value will be converted to the given type. If None, either the datatype will be kept (if initial_value is a Tensor) or converted to Tensor.

shape: (optional) The shape of this variable. If None, the shape of initial_value will be used.

2. Machine Learning con Python – TensorFlow v2: Variabili

TF v2

```
# Import TensorFlow v2
import tensorflow as tf

# Inizializza la costante const
const = tf.constant(2.5, name="const")

# Crea le variabili a e b
a = tf.Variable(tf.ones([3]), name='a')
b = tf.Variable(tf.math.add(a, const))

# Definisce le operazioni
sum = tf.add(b, const, name='somma')  # b + const
mol = tf.multiply(sum, const, name='mol')  # sum * const

print("a = {}".format(a.read_value()))
print("b = ", b.numpy())
print("sum = {}".format(sum))
print("mol = {}".format(mol))
```

```
Console -------------------------------------
a = [1. 1. 1.]
b =  b =  [3.5 3.5 3.5]
sum = [6. 6. 6.]
mol = [15. 15. 15.]
```

2. Machine Learning con Python – Tensorflow Basic Operations

Tensorflow: Definizione delle principali operazioni

TensorFlow operator	*Description*
tf.math.add	x+y
tf.math.subtract	x-y
tf.math.multiply	x*y
tf.math.divide	x/y
tf.math.floormod	x % y
tf.math.abs	|x|
tf.math.negative	-x
tf.math.sign	sign(x)
tf.math.square	x*x

TensorFlow operator	*Description*
tf.math.round	round(x)
tf.math.sqrt	sqrt(x)
tf.math.pow	x^y
tf.math.exp	e^x
tf.math.log	log(x)
tf.math.maximum	max(x, y)
tf.math.minimum	min(x, y)
tf.math.cos	cos(x)
tf.math.sin	sin(x)

2. Machine Learning con Python – TensorFlow v2: Build Models

TF v2

```
# Import tensorflow v2
import tensorflow as tf

W = tf.Variable(tf.ones(shape=(2, 2)), name="W")
b = tf.Variable(tf.zeros(shape=2), name="b")

@tf.function
def linreg(x):
    return W * x + b

out = linreg([1, 0])
print("W * x + b = ", out.numpy())
```

```
Console ------------------------------
W * x + b =  [[1. 0.] [1. 0.]]
```

2. Machine Learning con Python – Tensorboard

Tensorboard: Visualizzare il grafo definito nel programma

- Per visualizzare il grafo con Tensorboard, occorre prima di tutto scrivere un file di log con le informazioni del grafo tramite la classe `tf.summary`:

```
writer = tf.summary.create_file_writer('./graph_log')
```

- A questo punto è necessario definire una funzione in cui includere le operazioni di cui si vuole visualizzare il grafo, e successivamente utilizzare la funzione `tf.summary.graph` per scrivere il grafo:

```
with writer.as_default():
    tf.summary.graph(myFunction.get_concrete_function().graph)
```

2. Machine Learning con Python – Tensorboard

Tensorboard: Visualizzare il grafo definito nel programma

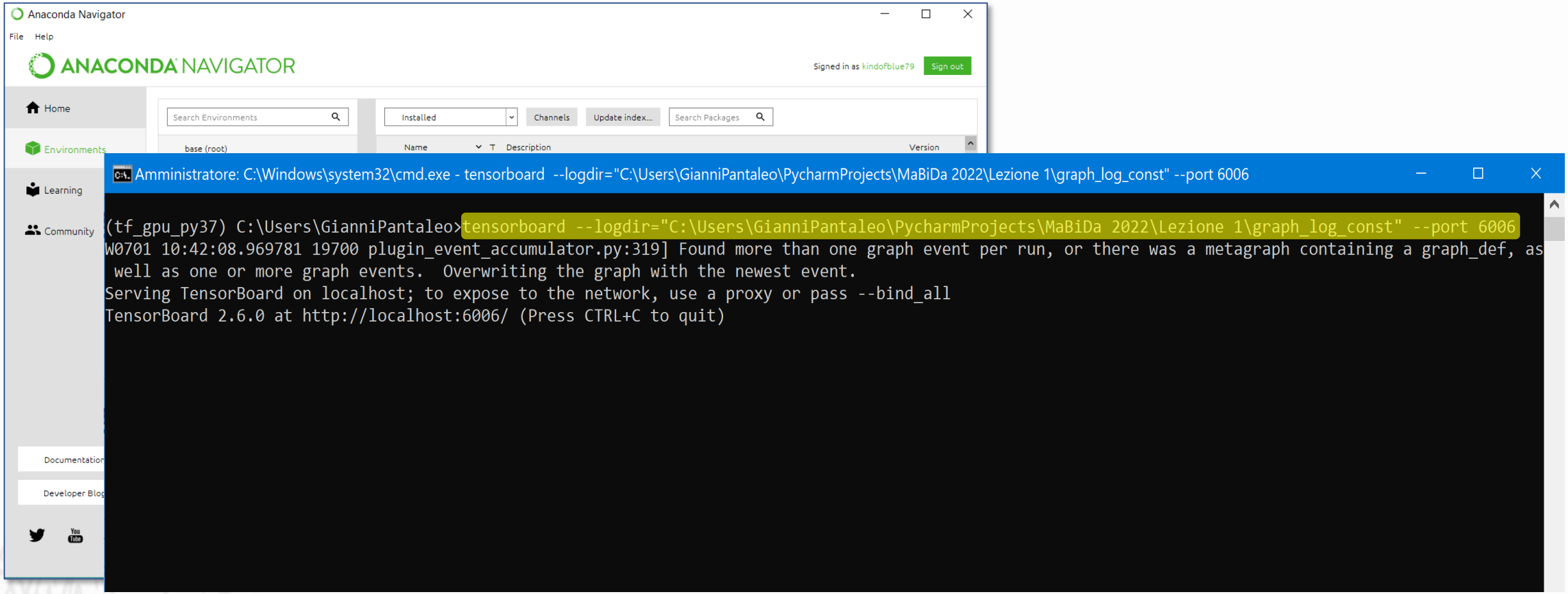
- Infine, aprire il terminale *conda* relativo al *virtual environment* su cui stiamo lavorando e digitare il comando:

```
tensorboard --logdir="<LOG_DIR_PATH>" --port 6006
```

dove `<LOG_DIR_PATH>` è il percorso della cartella di log definita nell'argomento di `tf.summary.create_file_writer`.

2. Machine Learning con Python – Tensorboard

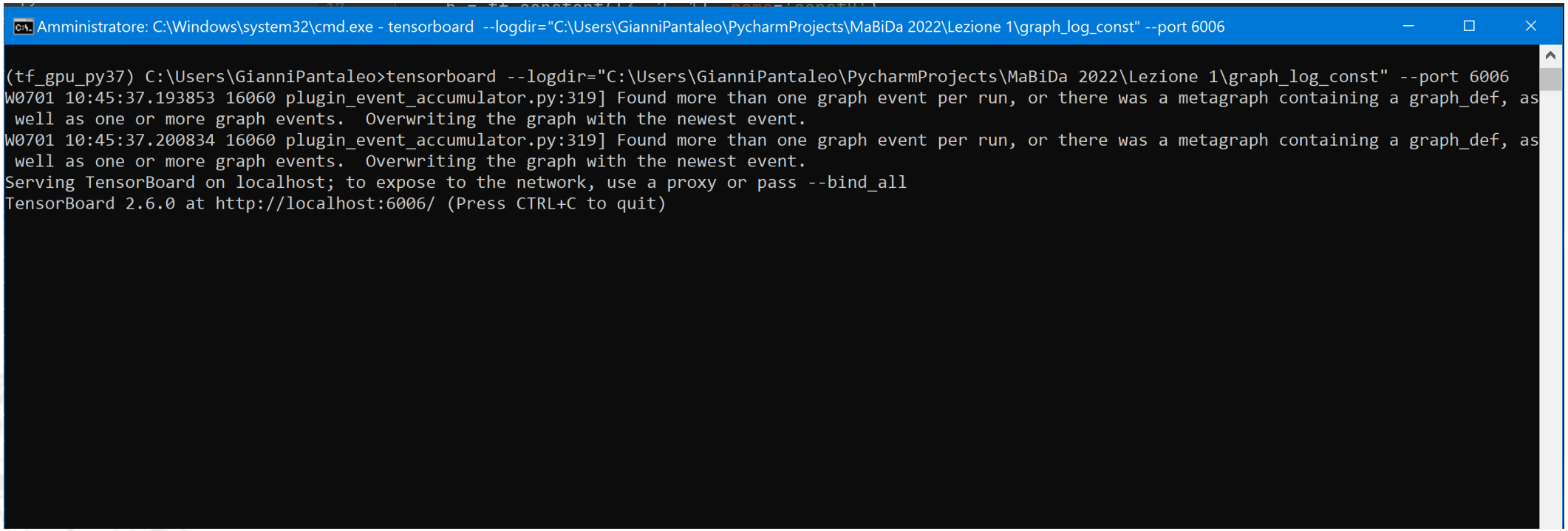
Tensorboard: Visualizzare il grafo definito nel programma



2. Machine Learning con Python – Tensorboard

Tensorboard: Visualizzare il grafo generato dal programma

Se si esegue più volte lo script scrivendo il grafo nella stessa cartella, la classe **`tf.summary`** non sovrascrive il file di log in cui viene salvato il grafo, e viene prodotto un warning sul terminale. E' necessario sovrascrivere o eliminare il file di log meno recenti:



2. Machine Learning con Python – TensorFlow v2: Tensorboard

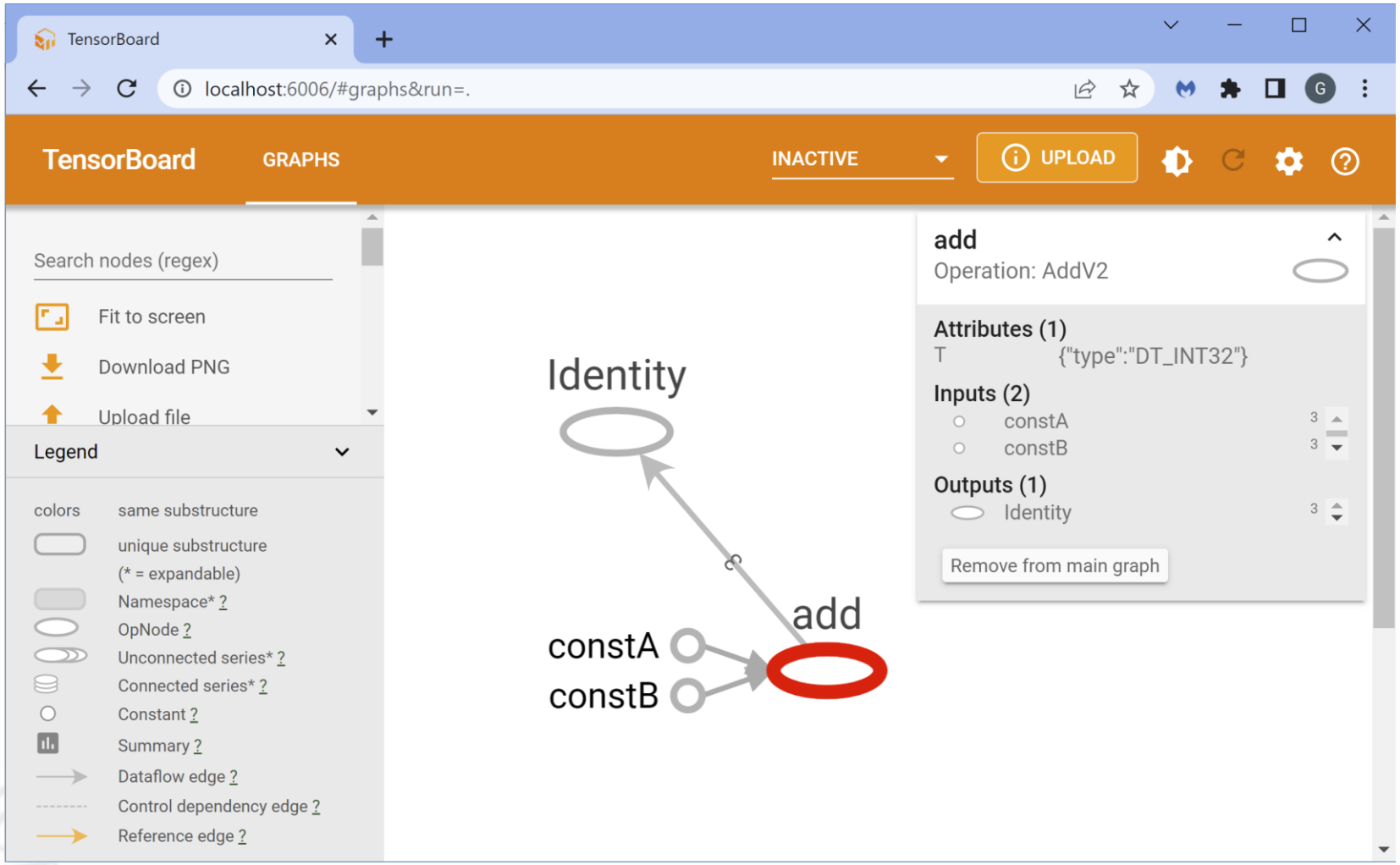
TF v2

```
import tensorflow as tf

@tf.function
def somma1():
    # Inizializza le costanti a e b
    a = tf.constant([1, 2, 3], name='constA')
    b = tf.constant([3, 2, 1], name='constB')
    result = tf.math.add(a, b, name='Addizione')
    return result

@tf.function
def somma2():
    # Inizializza le costanti a e b
    a = tf.constant([1, 2, 3], name='constA')
    b = tf.constant([3, 2, 1], name='constB')
    return a + b

writer = tf.summary.create_file_writer('./graph_log_const')

with writer.as_default():
    tf.summary.graph(somma2.get_concrete_function().graph)
    # tf.summary.graph(somma1.get_concrete_function().graph)
```

Scrittura del file di log in cui viene salvato il grafo definito nel programma

2. Machine Learning con Python – Tensorboard

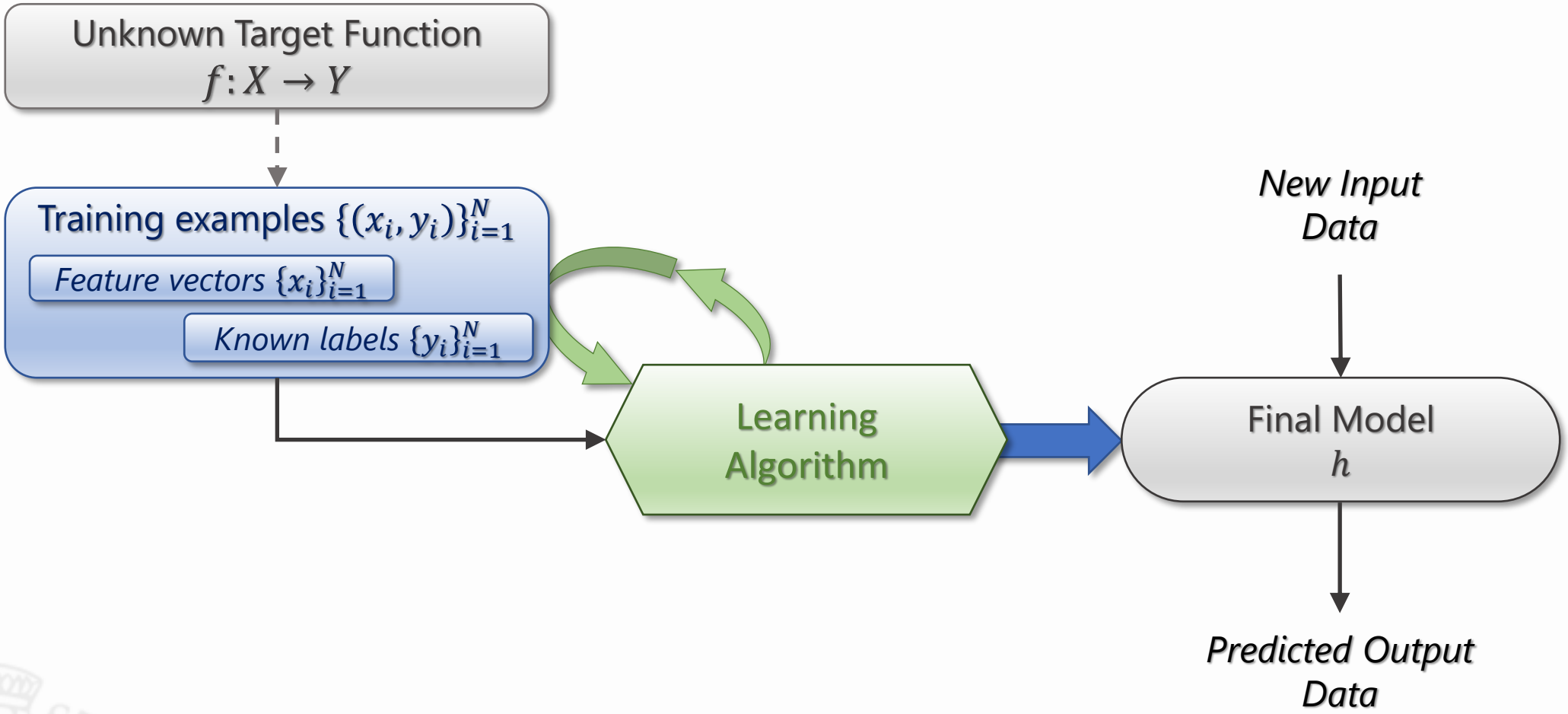
Tensorboard: Visualizzare il grafo definito nel programma http://localhost:6006/



Introduzione al Machine Learning - *Lineup*

1. *Introduzione*
2. *Machine Learning con Python*
3. *Applicazioni* ←
4. *Deep Learning*

3. Applicazioni: Supervised Learning – Modello



3. Applicazioni: Supervised Learning – Training, Validation, Test

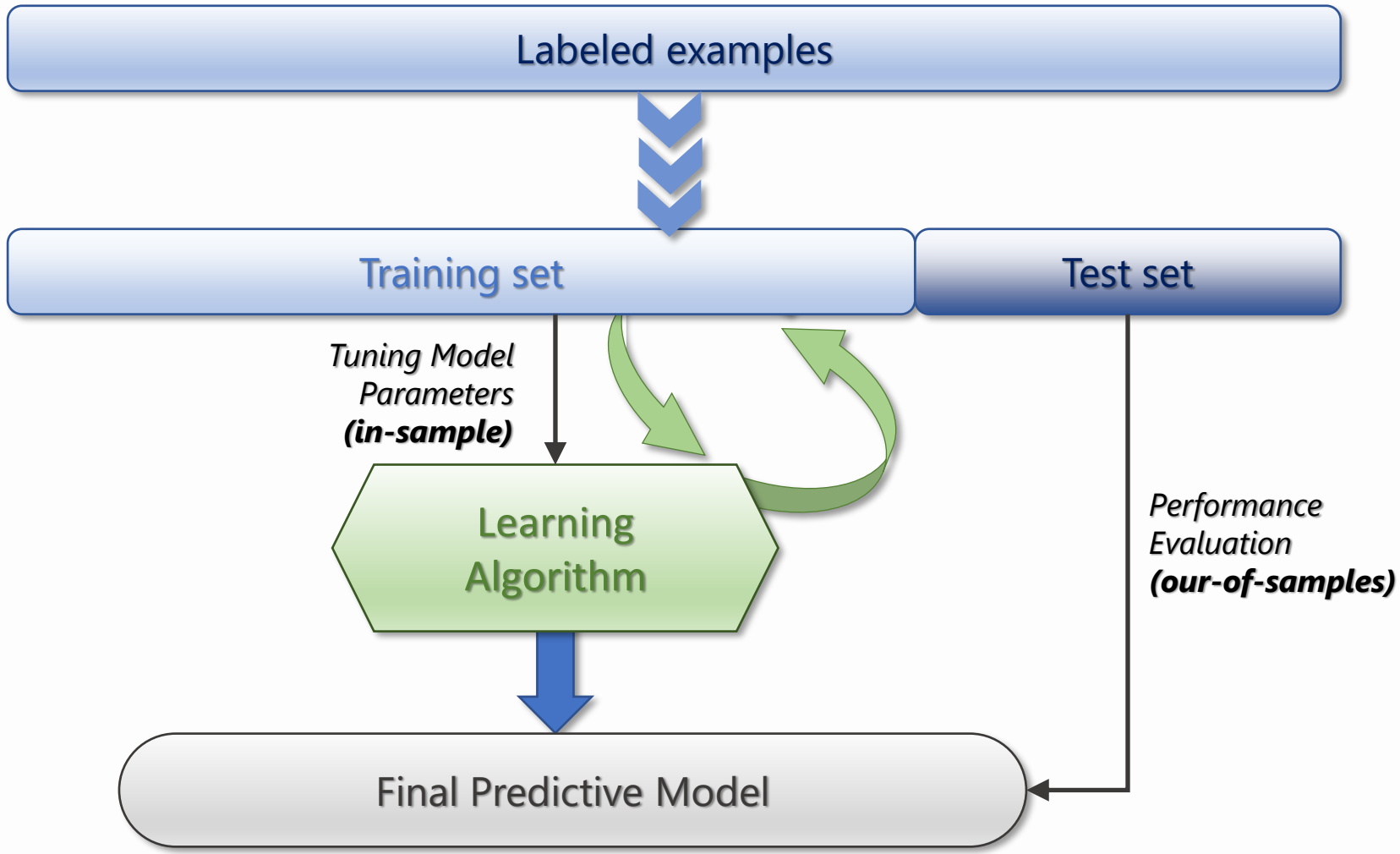
Data partition e fasi di un processo di Supervised Learning:

i. l'algoritmo di apprendimento viene addestrato (**training phase**) a produrre il modello predittivo o di classificazione attraverso un set di dati (*training data set*) annotati (*labeled*), ovvero il cui risultato (*expected output*) è noto;

ii. il modello viene poi validato (**validation phase**) su un data set apposito (*validation data set*), diverso dal data set di training, per ottimizzare parametri di alto livello (*hyperparameters*);

iii. infine, l'accuratezza predittiva del modello prodotto viene testata (**test phase**) su un data set diverso rispetto a quelli di training e validazione (*test data set*), formato da dati annotati o non annotati non precedentemente «visti» dal modello.

3. Applicazioni: Supervised Learning – Training, Validation, Test



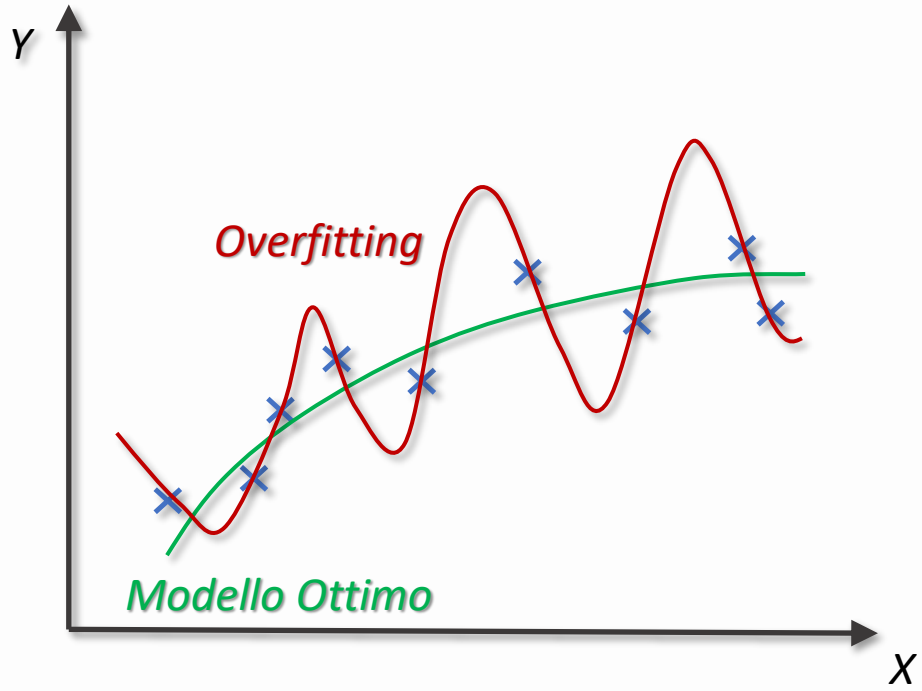
3. Applicazioni: Supervised Learning – Overfitting

Il problema dell'Overfitting

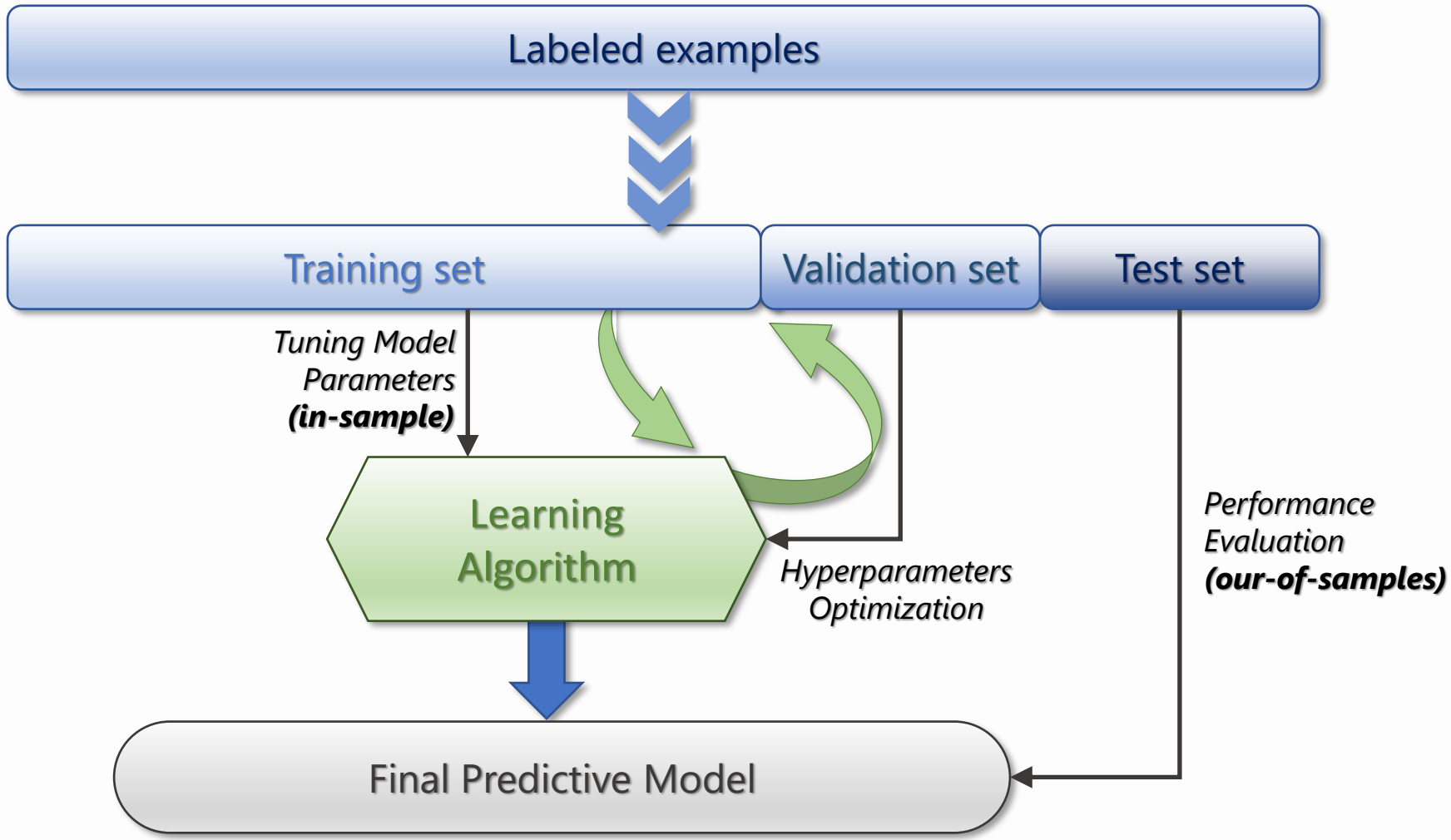
Aumentare la complessità del modello in modo da ottenere ottime performance (in termini di capacità predittiva) in fase di training spesso porta al fenomeno dell'***Overfitting***.

Questo si traduce in una diminuzione dell'errore del modello sul dataset di training ma in un aumento dell'errore sul dataset di test (il modello si comporta molto bene in fase di training ma non riesce a generalizzare in maniera soddisfacente su dati out-of-sample).

Ciò accade perché il modello, in questi casi, viene addestrato ad adattarsi anche il rumore, oltre che ai dati.



3. Applicazioni: Supervised Learning – Training, Validation, Test

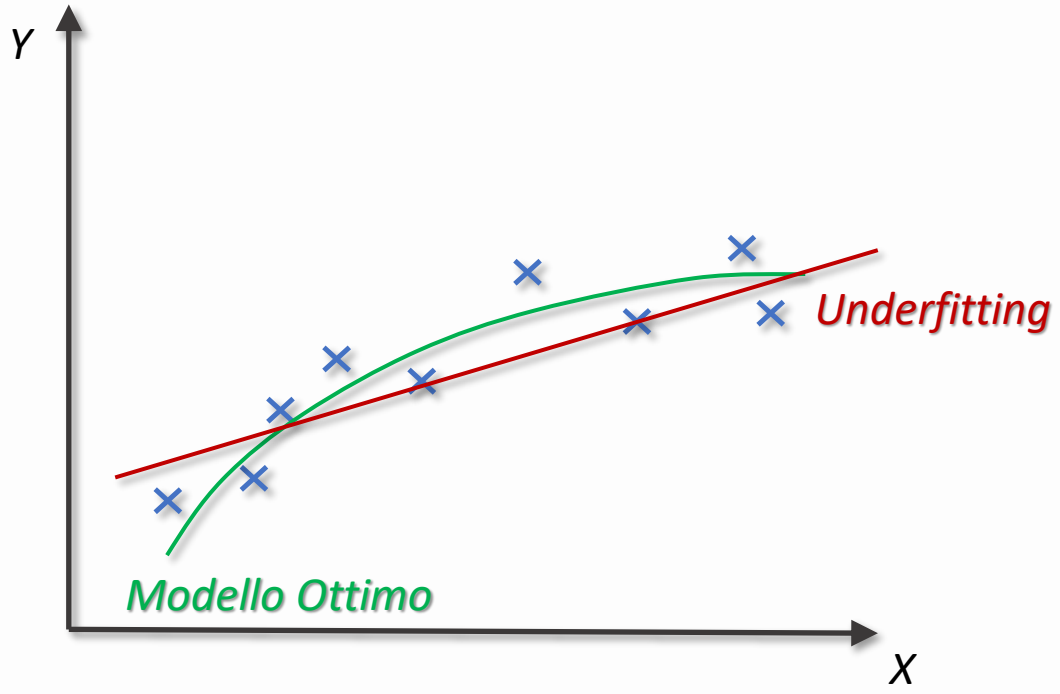


3. Applicazioni: Supervised Learning – Underfitting

Il problema dell'Underfitting

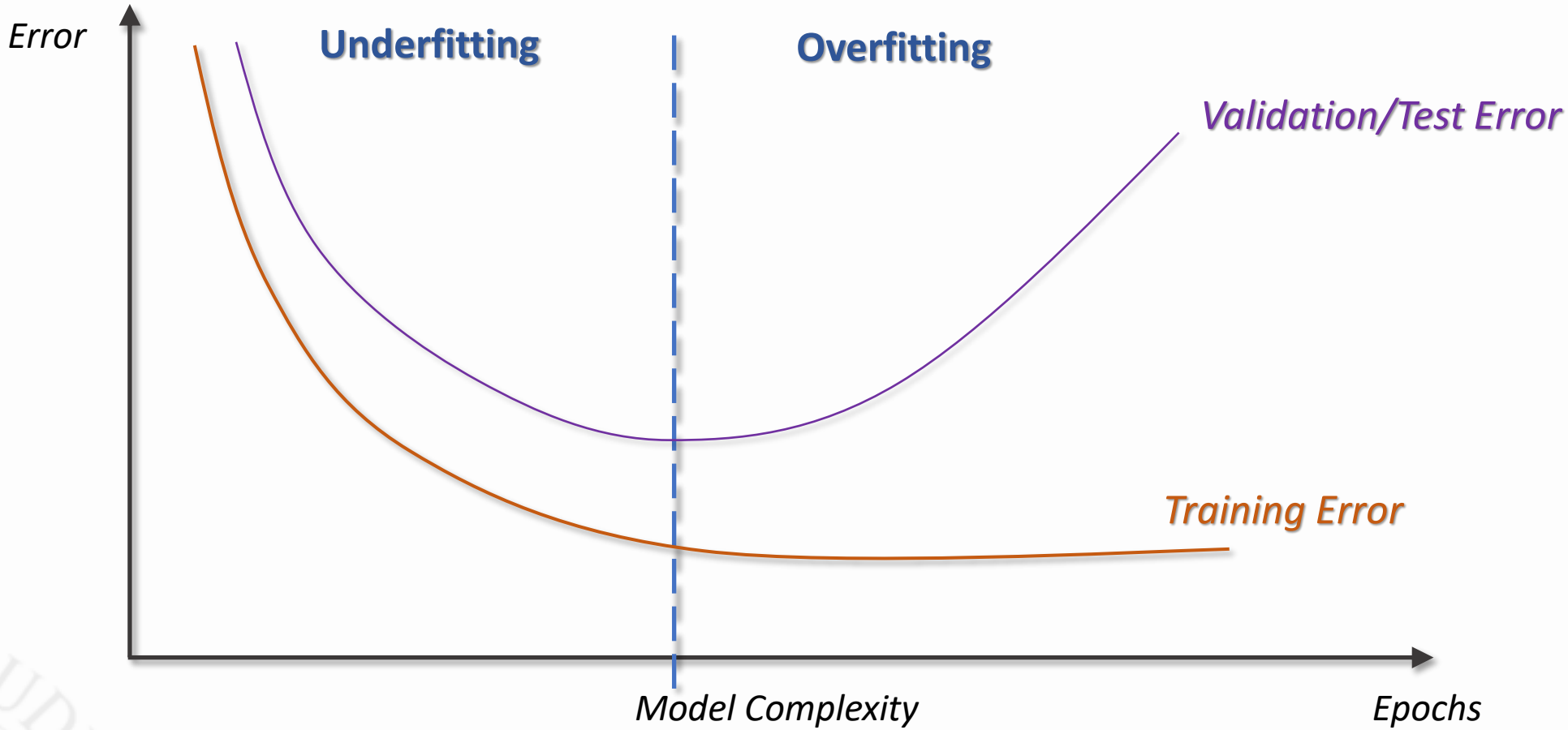
Il problema opposto all'Overfitting, ovvero l'***Underfitting***, si incontra quando il modello è troppo semplice (basso numero di parametri/iterazioni, pochi dati nel data set di training…)

In questo caso il modello non riesce ad apprendere pattern, trend dai dati e tipicamente non raggiunge prestazioni soddisfacenti (in termini di capacità predittive o di classificazione).



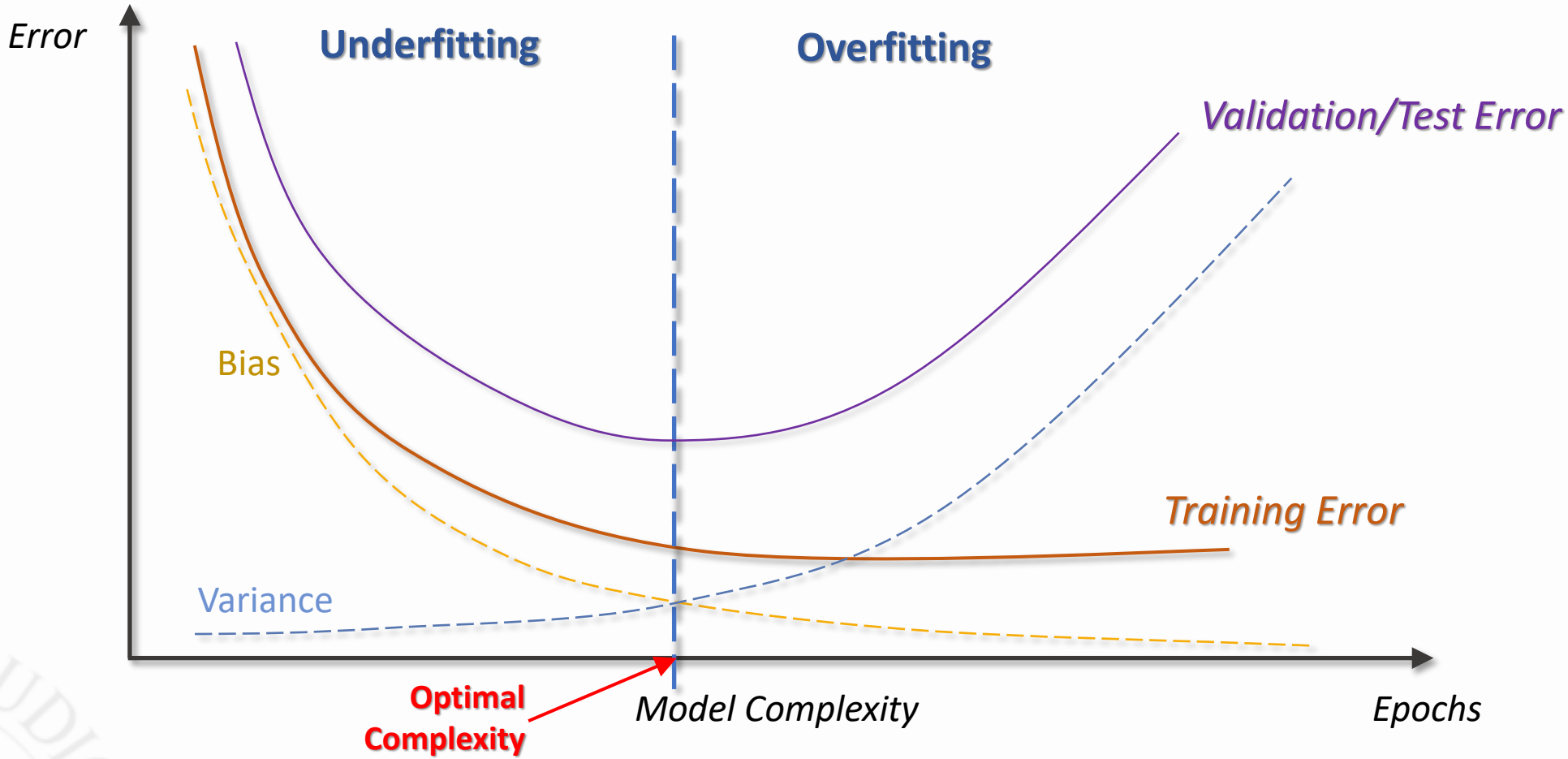
3. Supervised Learning – Tradeoff tra Underfitting e Overfitting

Underfitting, Overfitting – Errore di Training ed Errore di Test



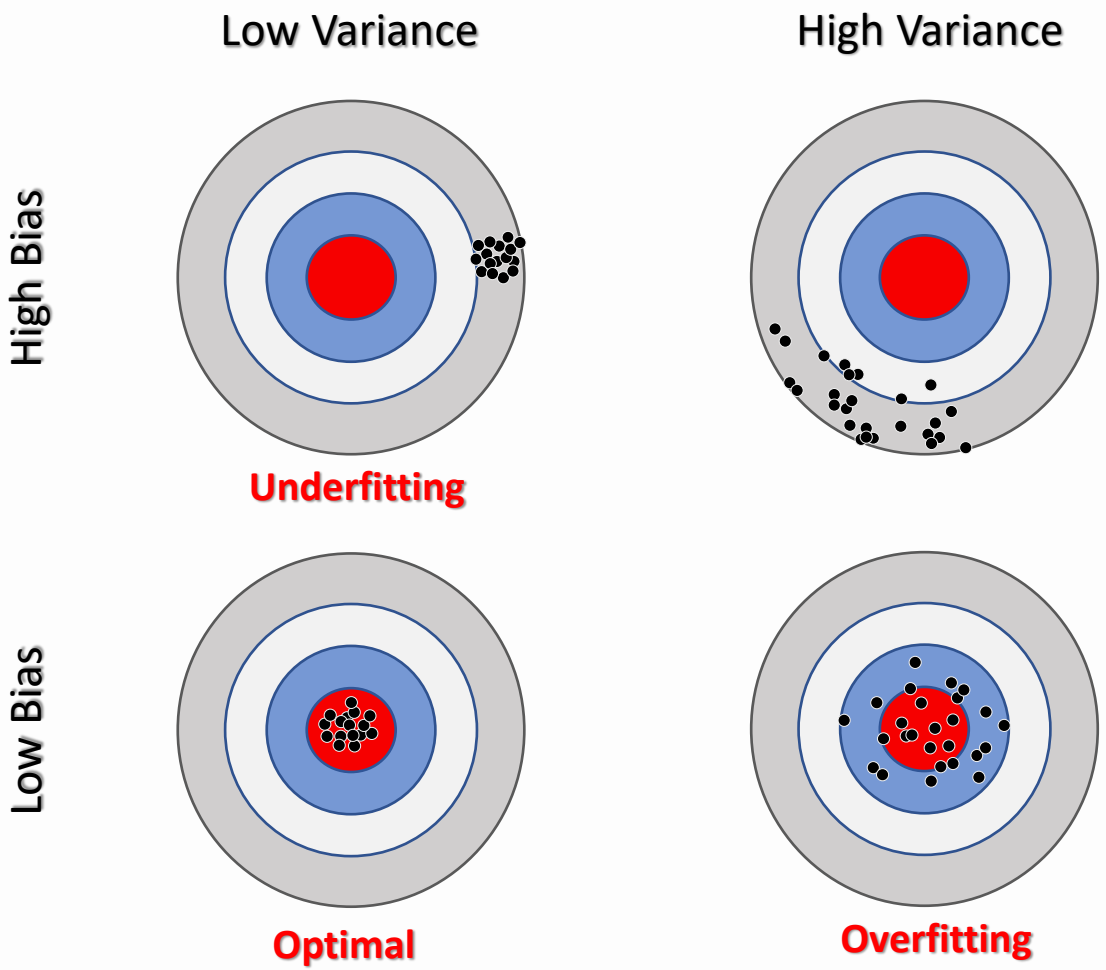
3. Supervised Learning – Tradeoff tra Bias e Varianza

Underfitting, Overfitting – Trade-off tra Bias e Varianza



3. Supervised Learning –Tradeoff tra Bias e Varianza

Underfitting, Overfitting – Trade-off tra Bias e Varianza



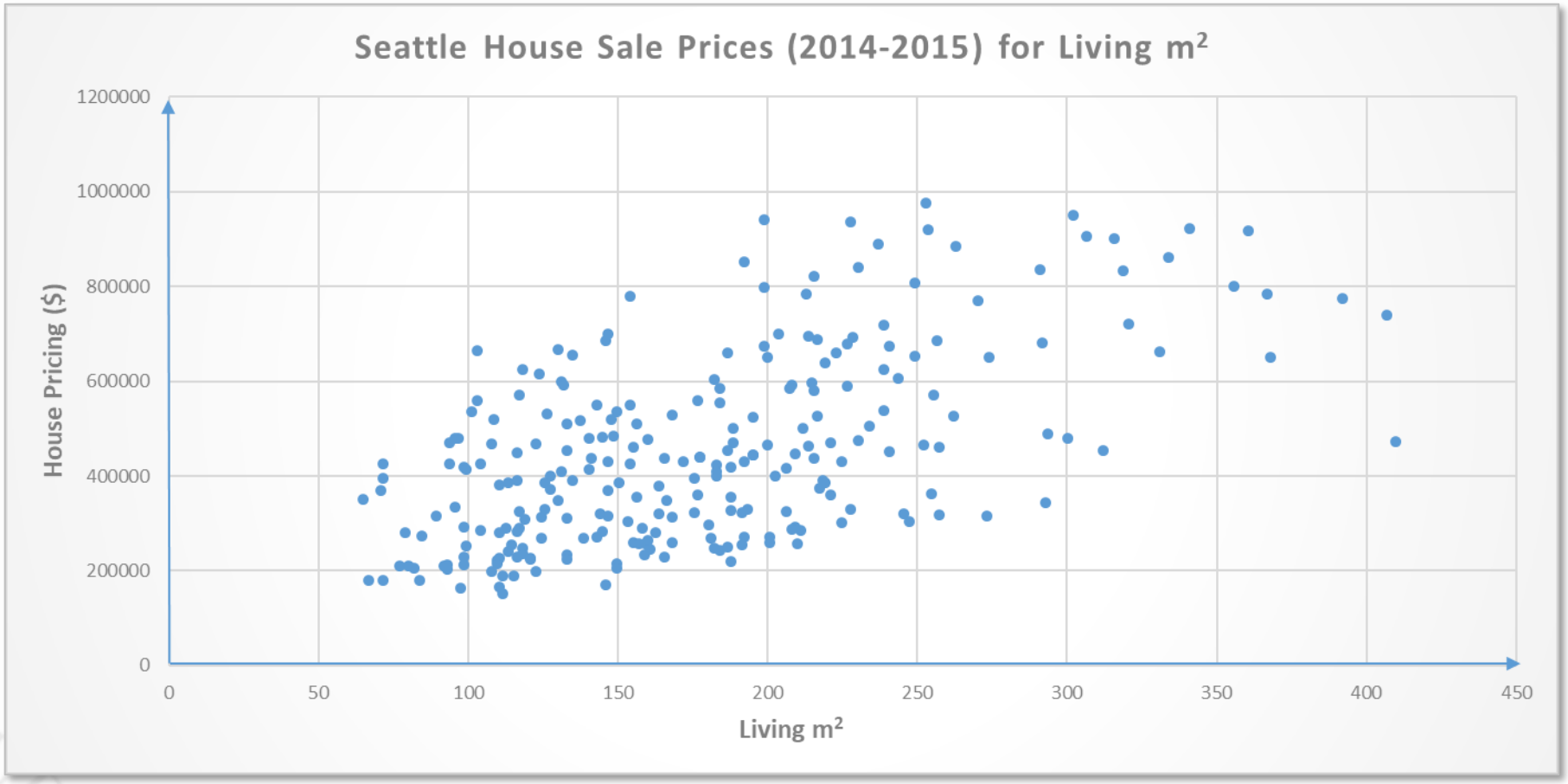
3. Supervised Learning – Metodi

Principali metodi e approcci:

- **Regressione:** La variabile da predire è di tipo continuo.
  - ***Linear/Polynomial Regression***
  - ***Decision Trees Regression***
  - ***Random Forest Regression***
  - ***Neural Networks***
- **Classificazione:** La variabile da predire è di tipo discreto (un numero finito di classi, binary-class / multi-class…)
  - ***Logistic Regression***
  - ***Naïve Bayes***
  - ***Support Vector Machine (SVM)***
  - ***K Nearest Neighbor***
  - ***Decision Trees Classification***
  - ***Random Forest Classification***
  - ***Neural Networks***

3. Applicazioni: Supervised Learning – Regressione Lineare

Regressione Lineare



Dataset from Kaggle: https://www.kaggle.com/swathiachath/kc-housesales-data/download

1. Introduzione – *Machine Learning*: Alcune Definizioni

- « Machine Learning is a field of study that gives computers the ***ability to learn without being explicitly programmed***. » *(A. L. Samuel)*
  - ✓ *« A computer **can be programmed so that it will learn to play a better game** of checkers **than can be played by the person** who wrote the program ».*

  ...e...
  - ✓ *« **Programming computers to learn from experience** should eventually eliminate the need for much of this detailed programming effort. »*

  (A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers", 1959)

- « A computer program is said to learn from ***experience E*** with respect to some class of ***tasks T*** and ***performance measure P***, if its performance at tasks in ***T***, as measured by ***P***, improves with experience ***E***. »

  (T. M. Mitchell, "Machine Learning", McGraw Hill, 1997)

3. Applicazioni: Supervised Learning – Regressione Lineare

Regressione Lineare

- **Experience *E*** : Dato un set osservabile di dati annotati $\mathbb{T} = \{(x_i, y_i)\}_{i=1}^{N}$, dove $\boldsymbol{x} = x_1, x_2, \dots, x_N$ è un vettore di *features*, e $\boldsymbol{y} = y_1, y_2, \dots, y_N$ un vettore di *labels* note ($\boldsymbol{x} \in \mathbb{R}$, $\boldsymbol{y} \in \mathbb{R}$);

- **Task *T*** : stimare tramite apprendimento la funzione non nota f: $y = f(\boldsymbol{x})$ attraverso un modello

$$h_{\boldsymbol{w},b}: \hat{\boldsymbol{y}} = \boldsymbol{w}\boldsymbol{x} + \boldsymbol{b}$$

  che approssimi f, per predire valori non noti $\hat{\boldsymbol{y}}$ partendo da dati non precedentemente annotati;

- **Performance *P*** : minimizzare l'errore di predizione (errore quadratico medio, ***MSE***):

$$MSE = cf = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{y}_i - (\boldsymbol{w}x_i + \boldsymbol{b})\right)^2$$

Cost Function

3. Applicazioni: Supervised Learning – Gradient Descent

Gradient Descent

- L'algoritmo *Gradient Descent* consente a un modello di apprendere iterativamente la «direzione» che il modello dovrebbe seguire al fine di minimizzare la funzione di costo;

$$cf = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{y}_i - (\boldsymbol{w}x_i + \boldsymbol{b})\right)^2$$

- Per minimizzare la funzione di costo, l'algoritmo *Gradient Descent* utilizza il calcolo delle derivate parziali rispetto a $\boldsymbol{w}$ e $\boldsymbol{b}$:

$$\frac{\partial cf}{\partial \boldsymbol{w}} = \frac{1}{N}\sum_{i=1}^{N} -2x_i\left(\hat{y}_i - (\boldsymbol{w}x_i + \boldsymbol{b})\right)$$

$$\frac{\partial cf}{\partial \boldsymbol{b}} = \frac{1}{N}\sum_{i=1}^{N} -2\left(\hat{y}_i - (\boldsymbol{w}x_i + \boldsymbol{b})\right)$$

3. Applicazioni: Supervised Learning – Gradient Descent

Gradient Descent: Implementazione dell'algoritmo

1. Alla prima iterazione si inizializzano nella maniera più opportuna i valori dei coefficienti e dell'intercetta. Si definisce inoltre un valore *Learning Rate* L per controllare la variazione di $\boldsymbol{w}$ e $\boldsymbol{b}$.

2. Si calcolano le derivate parziali della funzione di costo rispetto a $\boldsymbol{w}$ e $\boldsymbol{b}$:

$$\frac{\partial cf}{\partial \boldsymbol{w}} = \frac{1}{N}\sum_{i=1}^{N} -2x_i\left(\hat{y}_i - (\boldsymbol{w}x_i + \boldsymbol{b})\right) \qquad \frac{\partial cf}{\partial b} = \frac{1}{N}\sum_{i=1}^{N} -2\left(\hat{y}_i - (\boldsymbol{w}x_i + \boldsymbol{b})\right)$$

3. Si aggiornano i valori di $\boldsymbol{w}$ e $\boldsymbol{b}$:

Back Propagation

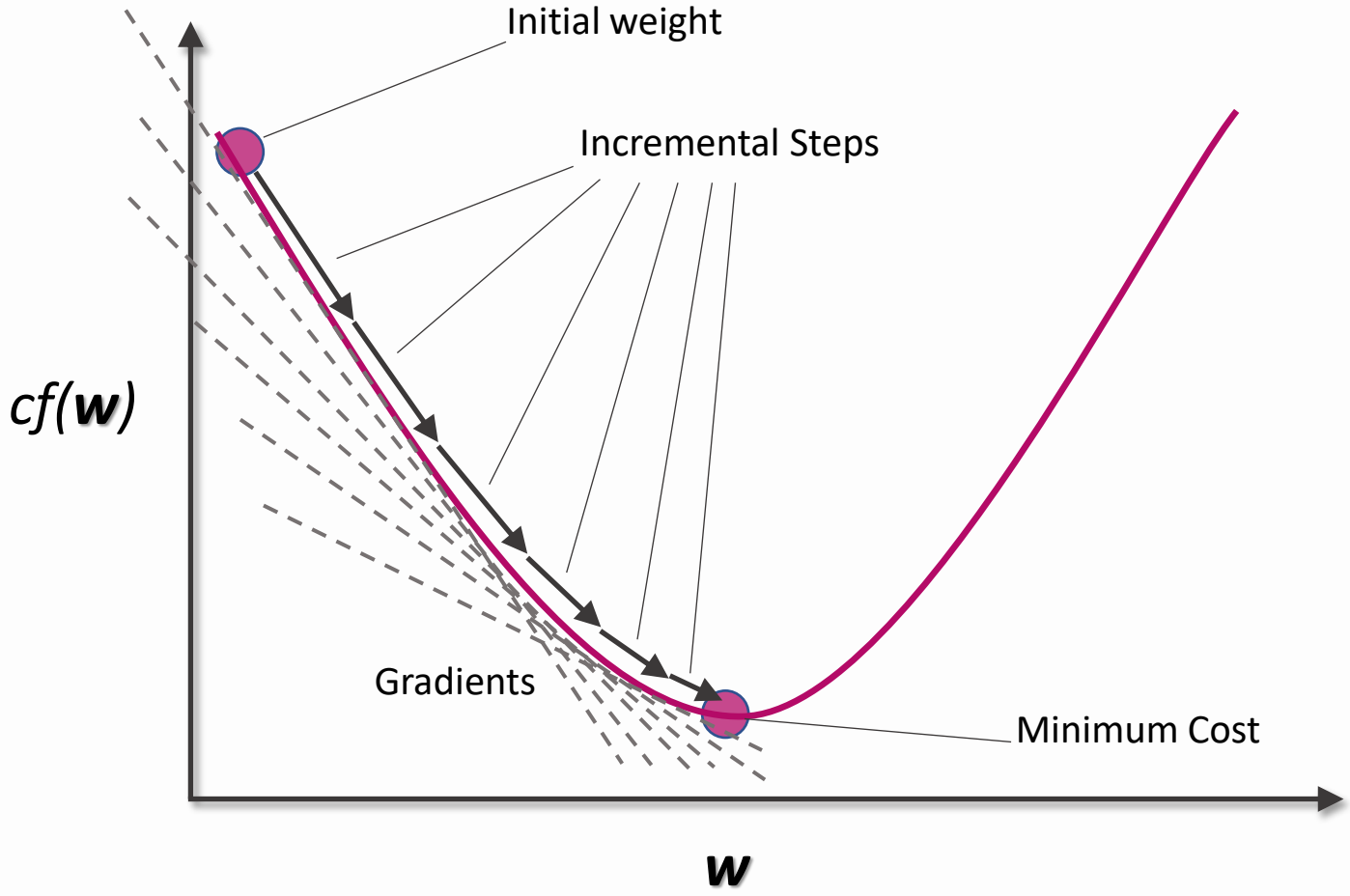
$$\boldsymbol{w} = \boldsymbol{w} - L\frac{\partial cf}{\partial \boldsymbol{w}} \qquad \boldsymbol{b} = \boldsymbol{b} - L\frac{\partial cf}{\partial \boldsymbol{b}}$$

4. Si calcola la nuova funzione di costo con i valori aggiornati di $\boldsymbol{w}$ e $\boldsymbol{b}$, e si procede per iterazioni successive finché il valore della funzione di costo raggiunge il valore desiderato (oppure terminano le iterazioni previste).

5. I valori di $\boldsymbol{w}$ e $\boldsymbol{b}$ per cui la funzione di costo viene minimizzata rappresentano i valori ottimi per il modello predittivo.

3. Applicazioni: Supervised Learning – Gradient Descent

Gradient Descent



3. Applicazioni: Supervised Learning – Gradient Descent

Stochastic Gradient Descent (SGD) e Mini-batch Gradient Descent

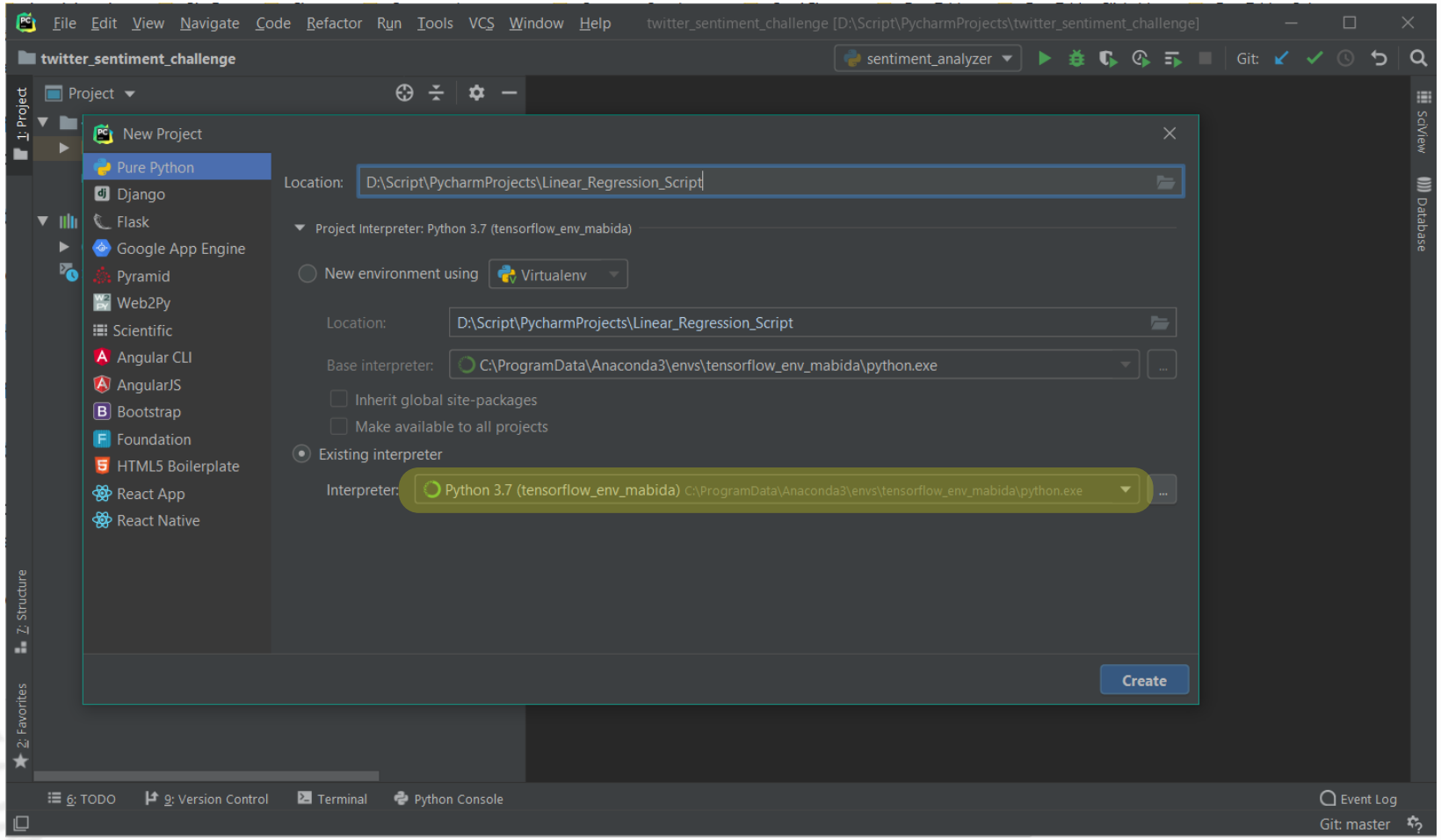
- L'implementazione tradizionale del *Gradient Descent* viene eseguita sull'intero dataset di training (*Batch Gradient Descent*), e può diventare computazionalmente troppo costosa con l'aumentare delle dimensioni del training dataset.
- L'algoritmo *Stochastic Gradient Descent (SGD)* riduce la computazione del *Batch Gradient Descent* su un singolo campione (selezionato in maniera random ad ogni iterazione); come operazione preliminare si esegue un *random shuffle* tra i campioni. Il cammino verso il minimo locale è in questo caso più rumoroso, ma i tempi di training si accorciano notevolmente.
- Un trade-off tra Batch GD e Stochastic GD è il *mini-Batch Gradient Descent*, che consiste nell'iterare su un sottoinsieme $\mathbb{B}$ del dataset di training $\mathbb{T}$:

$$\frac{\partial cf}{\partial \boldsymbol{w}} = \frac{1}{N}\sum_{j=1}^{m} -2x_j\left(\hat{y}_j - \left(\boldsymbol{w}x_j + \boldsymbol{b}\right)\right) \qquad \frac{\partial cf}{\partial \boldsymbol{b}} = \frac{1}{N}\sum_{j=1}^{m} -2\left(\hat{y}_j - \left(\boldsymbol{w}x_j + \boldsymbol{b}\right)\right)$$

$$\mathbb{B} = \{(x_j, y_j)\}_{j=1}^{m} \subset \mathbb{T} = \{(x_i, y_i)\}_{i=1}^{N}$$

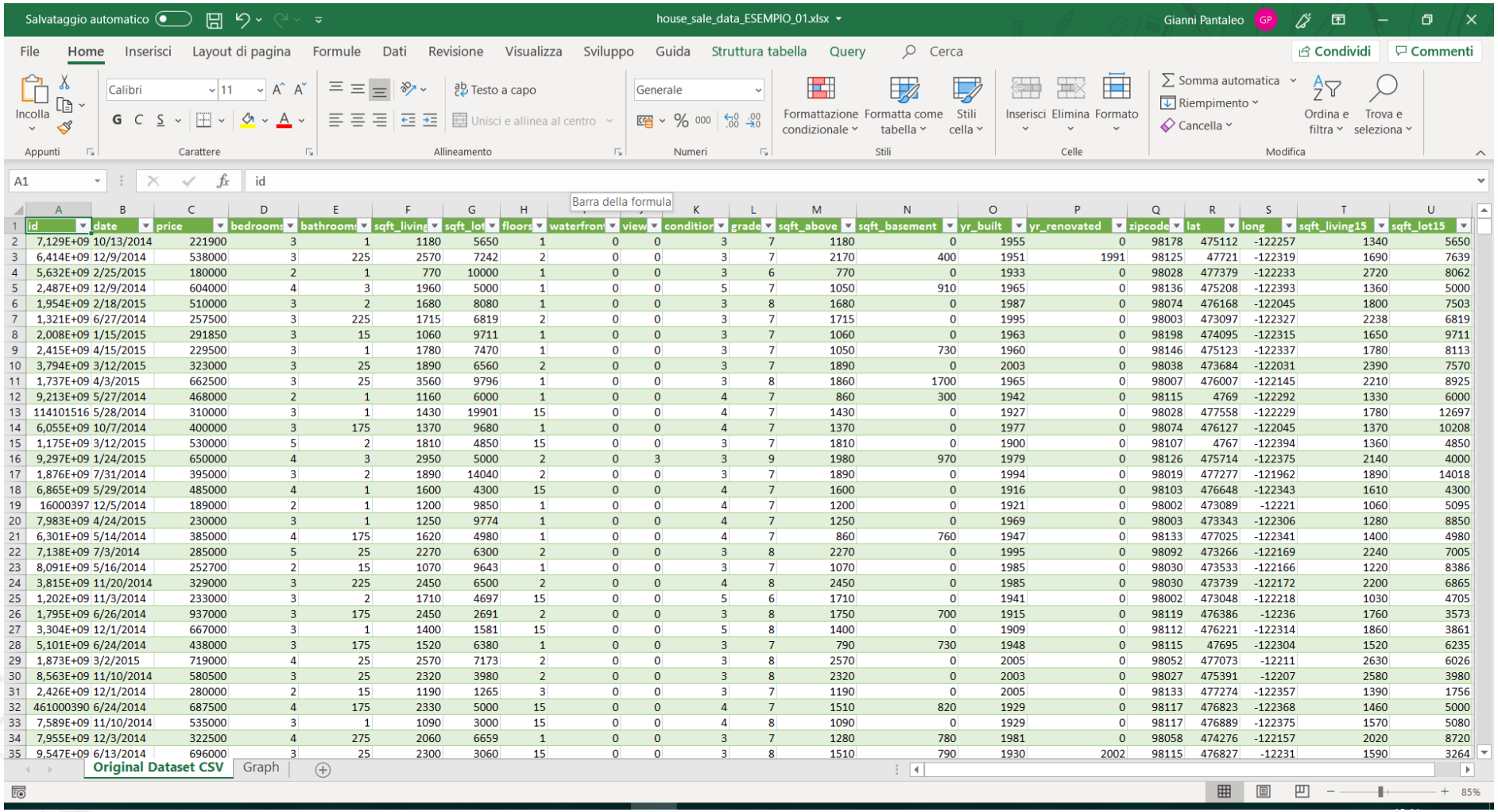
3. Applicazioni: Supervised Learning – Lavorare con Python

- Avviare l'ambiente di sviluppo (**IDE**)
- Configurare il virtual environment e l'interprete Python (***conda***) all'interno dell'IDE



3. Applicazioni: Supervised Learning – Preparazione dei Dati

- Preparare i dati:



id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
7,129E+09	10/13/2014	221900	3	1	1180	5650	1	0	0	3	7	1180	0	1955	0	98178	475112	-122257	1340	5650
6,414E+09	12/9/2014	538000	3	225	2570	7242	2	0	0	3	7	2170	400	1951	1991	98125	47721	-122319	1690	7639
5,632E+09	2/25/2015	180000	2	1	770	10000	1	0	0	3	6	770	0	1933	0	98028	477379	-122233	2720	8062
2,487E+09	12/9/2014	604000	4	3	1960	5000	1	0	0	5	7	1050	910	1965	0	98136	475208	-122393	1360	5000
1,954E+09	2/18/2015	510000	3	2	1680	8080	1	0	0	3	8	1680	0	1987	0	98074	476168	-122045	1800	7503
1,321E+09	6/27/2014	257500	3	225	1715	6819	2	0	0	3	7	1715	0	1995	0	98003	473097	-122327	2238	6819
2,008E+09	1/15/2015	291850	3	15	1060	9711	1	0	0	3	7	1060	0	1963	0	98198	474095	-122315	1650	9711
2,415E+09	4/15/2015	229500	3	1	1780	7470	1	0	0	3	7	1050	730	1960	0	98146	475123	-122337	1780	8113
3,794E+09	3/12/2015	323000	3	25	1890	6560	2	0	0	3	7	1890	0	2003	0	98038	473684	-122031	2390	7570
1,737E+09	4/3/2015	662500	3	25	3560	9796	1	0	0	3	8	1860	1700	1965	0	98007	476007	-122145	2210	8925
9,213E+09	5/27/2014	468000	2	1	1160	6000	1	0	0	4	7	860	300	1942	0	98115	4769	-122292	1330	6000
114101516	5/28/2014	310000	3	1	1430	19901	15	0	0	4	7	1430	0	1927	0	98028	477558	-122229	1780	12697
6,055E+09	10/7/2014	400000	3	175	1370	9680	1	0	0	4	7	1370	0	1977	0	98074	476127	-122045	1370	10208
1,175E+09	3/12/2015	530000	5	2	1810	4850	15	0	0	3	7	1810	0	1900	0	98107	4767	-122394	1360	4850
9,297E+09	1/24/2015	650000	4	3	2950	5000	2	0	3	3	9	1980	970	1979	0	98126	475714	-122375	2140	4000
1,876E+09	7/31/2014	395000	3	2	1890	14040	2	0	0	3	7	1890	0	1994	0	98019	477277	-121962	1890	14018
6,865E+09	5/29/2014	485000	4	1	1600	4300	15	0	0	4	7	1600	0	1916	0	98103	476648	-122343	1610	4300
16000397	12/5/2014	189000	2	1	1200	9850	1	0	0	4	7	1200	0	1921	0	98002	473089	-12221	1060	5095
7,983E+09	4/24/2015	230000	3	1	1250	9774	1	0	0	4	7	1250	0	1969	0	98003	473343	-122306	1280	8850
6,301E+09	5/14/2014	385000	4	175	1620	4980	1	0	0	4	7	860	760	1947	0	98133	477025	-122341	1400	4980
7,138E+09	7/3/2014	285000	5	25	2270	6300	2	0	0	3	8	2270	0	1995	0	98092	473266	-122169	2240	7005
8,091E+09	5/16/2014	252700	2	15	1070	9643	1	0	0	3	7	1070	0	1985	0	98030	473533	-122166	1220	8386
3,815E+09	11/20/2014	329000	3	225	2450	6500	2	0	0	4	8	2450	0	1985	0	98030	473739	-122172	2200	6865
1,202E+09	11/3/2014	233000	3	2	1710	4697	15	0	0	5	6	1710	0	1941	0	98002	473048	-122218	1030	4705
1,795E+09	6/26/2014	937000	3	175	2450	2691	2	0	0	3	8	1750	700	1915	0	98119	476386	-12236	1760	3573
3,304E+09	12/1/2014	667000	3	1	1400	1581	15	0	0	5	8	1400	0	1909	0	98112	476221	-122314	1860	3861
5,101E+09	6/24/2014	438000	3	175	1520	6380	1	0	0	3	7	790	730	1948	0	98115	47695	-122304	1520	6235
1,873E+09	3/2/2015	719000	4	25	2570	7173	2	0	0	3	8	2570	0	2005	0	98052	477073	-12211	2630	6026
8,563E+09	11/10/2014	580500	3	25	2320	3980	2	0	0	3	8	2320	0	2003	0	98027	475391	-12207	2580	3980
2,426E+09	12/1/2014	280000	2	15	1190	1265	3	0	0	3	7	1190	0	2005	0	98133	477274	-122357	1390	1756
461000390	6/24/2014	687500	4	175	2330	5000	15	0	0	4	7	1510	820	1929	0	98117	476823	-122368	1460	5000
7,589E+09	11/10/2014	535000	3	1	1090	3000	15	0	0	4	8	1090	0	1929	0	98117	476889	-122375	1570	5080
7,955E+09	12/3/2014	322500	4	275	2060	6659	1	0	0	3	7	1280	780	1981	0	98058	474276	-122157	2020	8720
9,547E+09	6/13/2014	696000	3	25	2300	3060	15	0	0	3	8	1510	790	1930	2002	98115	476827	-12231	1590	3264

3. Applicazioni: Supervised Learning – Preparazione dei Dati

- Preparare i dati:

Salvataggio automatico — house_sale_data_ESEMPIO_01.xlsx

File Home Inserisci Layout di pagina Formule Dati Revisione Visualizza Sviluppo Guida Struttura tabella Query

SOMMA | =G2*0,092903

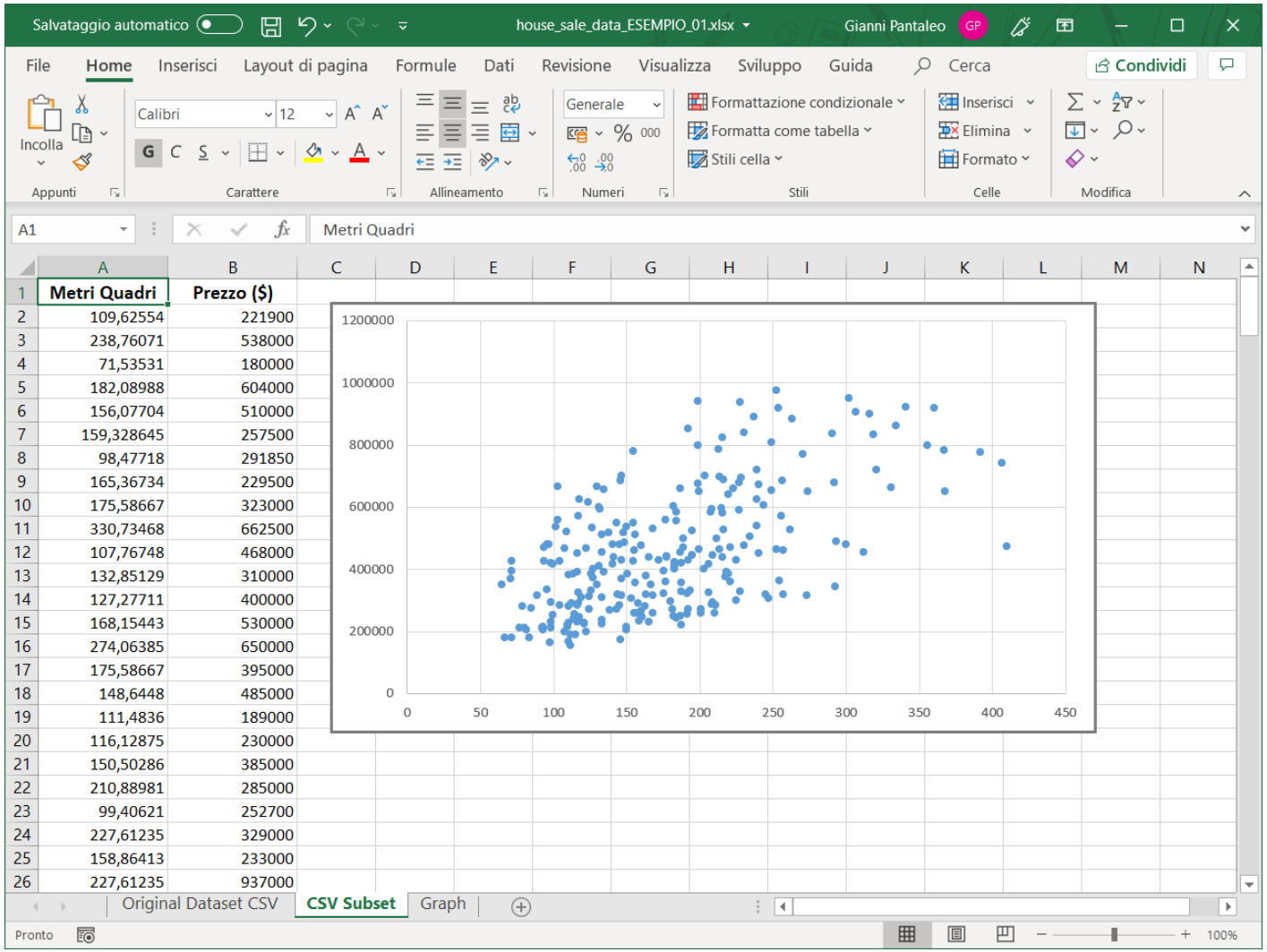
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
1	id	date	metri_quadri	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat
2	7129300520	10/13/2014	=G2*0,092903	221900	3	1	1180	5650	1	0	0	3	7	1180	0	1955	0	98178	47
3	6414100192	12/9/2014	238,76071	538000	3	225	2570	7242	2	0	0	3	7	2170	400	1951	1991	98125	4
4	5631500400	2/25/2015	71,53531	180000	2	1	770	10000	1	0	0	3	6	770	0	1933	0	98028	47
5	2487200875	12/9/2014	182,08988	604000	4	3	1960	5000	1	0	0	5	7	1050	910	1965	0	98136	47
6	1954400510	2/18/2015	156,07704	510000	3	2	1680	8080	1	0	0	3	8	1680	0	1987	0	98074	47
7	1321400060	6/27/2014	159,328645	257500	3	225	1715	6819	2	0	0	3	7	1715	0	1995	0	98003	47
8	2008000270	1/15/2015	98,47718	291850	3	15	1060	9711	1	0	0	3	7	1060	0	1963	0	98198	47
9	2414600126	4/15/2015	165,36734	229500	3	1	1780	7470	1	0	0	3	7	1050	730	1960	0	98146	47
10	3793500160	3/12/2015	175,58667	323000	3	25	1890	6560	2	0	0	3	7	1890	0	2003	0	98038	47
11	1736800520	4/3/2015	330,73468	662500	3	25	3560	9796	1	0	0	3	8	1860	1700	1965	0	98007	47
12	9212900260	5/27/2014	107,76748	468000	2	1	1160	6000	1	0	0	4	7	860	300	1942	0	98115	
13	114101516	5/28/2014	132,85129	310000	3	1	1430	19901	15	0	0	4	7	1430	0	1927	0	98028	47
14	6054650070	10/7/2014	127,27711	400000	3	175	1370	9680	1	0	0	4	7	1370	0	1977	0	98074	47
15	1175000570	3/12/2015	168,15443	530000	5	2	1810	4850	15	0	0	3	7	1810	0	1900	0	98107	
16	9297300055	1/24/2015	274,06385	650000	4	3	2950	5000	2	0	3	3	9	1980	970	1979	0	98126	47
17	1875500060	7/31/2014	175,58667	395000	3	2	1890	14040	2	0	0	3	7	1890	0	1994	0	98019	47
18	6865200140	5/29/2014	148,6448	485000	4	1	1600	4300	15	0	0	4	7	1600	0	1916	0	98103	47
19	16000397	12/5/2014	111,4836	189000	2	1	1200	9850	1	0	0	4	7	1200	0	1921	0	98002	47
20	7983200060	4/24/2015	116,12875	230000	3	1	1250	9774	1	0	0	4	7	1250	0	1969	0	98003	47
21	6300500875	5/14/2014	150,50286	385000	4	175	1620	4980	1	0	0	4	7	860	760	1947	0	98133	47
22	7137970340	7/3/2014	210,88981	285000	5	25	2270	6300	2	0	0	3	8	2270	0	1995	0	98092	47
23	8091400200	5/16/2014	99,40621	252700	2	15	1070	9643	1	0	0	3	7	1070	0	1985	0	98030	47
24	3814700200	11/20/2014	227,61235	329000	3	225	2450	6500	2	0	0	4	8	2450	0	1985	0	98030	47
25	1202000200	11/3/2014	158,86413	233000	3	2	1710	4697	15	0	0	5	6	1710	0	1941	0	98002	47
26	1794500383	6/26/2014	227,61235	937000	3	175	2450	2691	2	0	0	3	8	1750	700	1915	0	98119	47
27	3303700376	12/1/2014	130,0642	667000	3	1	1400	1581	15	0	0	5	8	1400	0	1909	0	98112	47
28	5101402488	6/24/2014	141,21256	438000	3	175	1520	6380	1	0	0	3	7	790	730	1948	0	98115	4
29	1873100390	3/2/2015	238,76071	719000	4	25	2570	7173	2	0	0	3	8	2570	0	2005	0	98052	47

Original Dataset CSV | Graph

Modifica

3. Applicazioni: Supervised Learning – Preparazione dei Dati

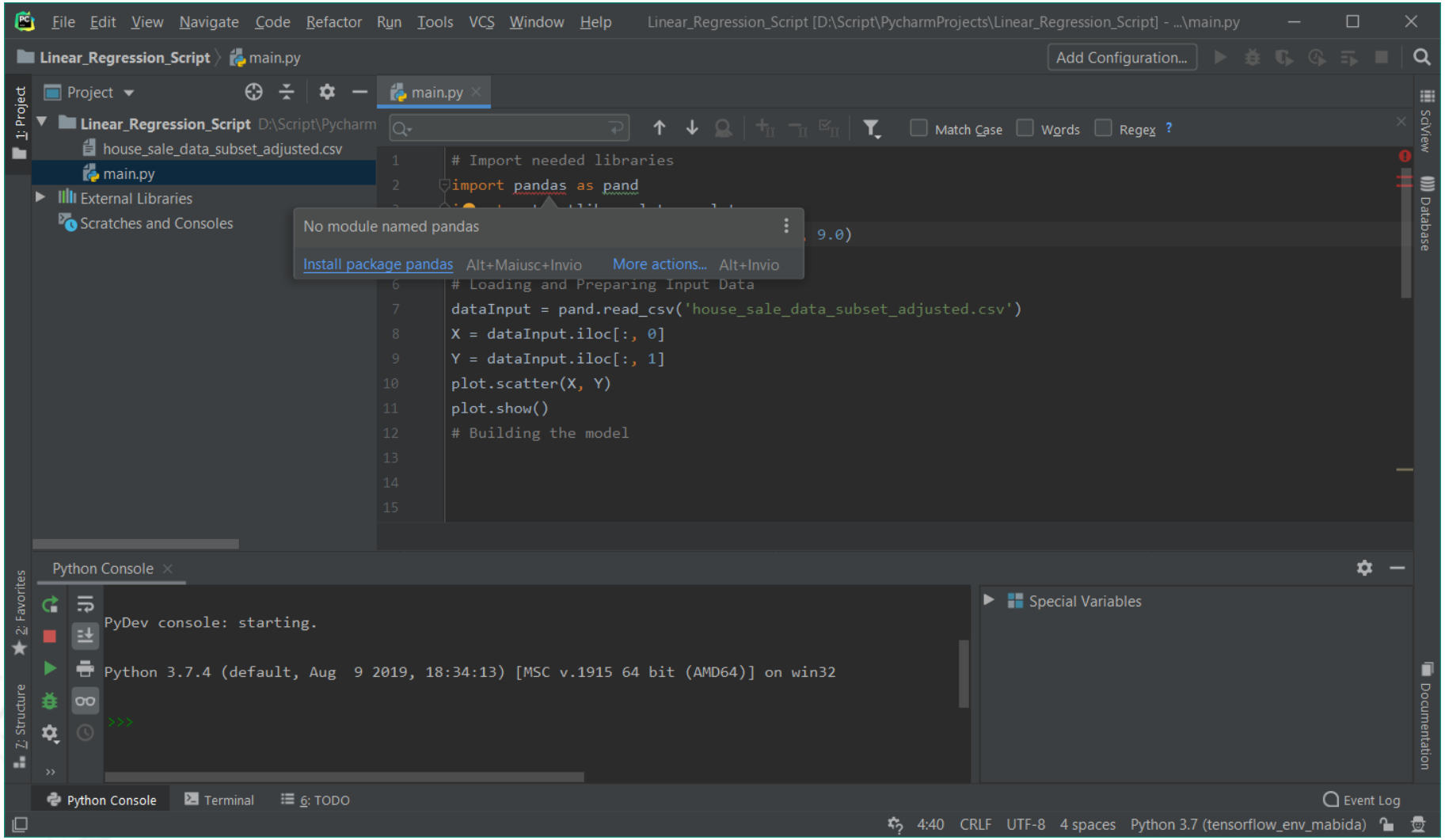
- Preparare i dati:



Metri Quadri	Prezzo ($)
109,62554	221900
238,76071	538000
71,53531	180000
182,08988	604000
156,07704	510000
159,328645	257500
98,47718	291850
165,36734	229500
175,58667	323000
330,73468	662500
107,76748	468000
132,85129	310000
127,27711	400000
168,15443	530000
274,06385	650000
175,58667	395000
148,6448	485000
111,4836	189000
116,12875	230000
150,50286	385000
210,88981	285000
99,40621	252700
227,61235	329000
158,86413	233000
227,61235	937000

3. Applicazioni: Supervised Learning – Lavorare con Python

- Installare le eventuali dipendenze mancanti (direttamente dall'IDE o da Anaconda Manager):



Introduzione al Machine Learning - *Lineup*

1. *Introduzione*
2. *Machine Learning con TensorFlow per Python*
3. *Applicazioni*
4. *Deep Learning* ←

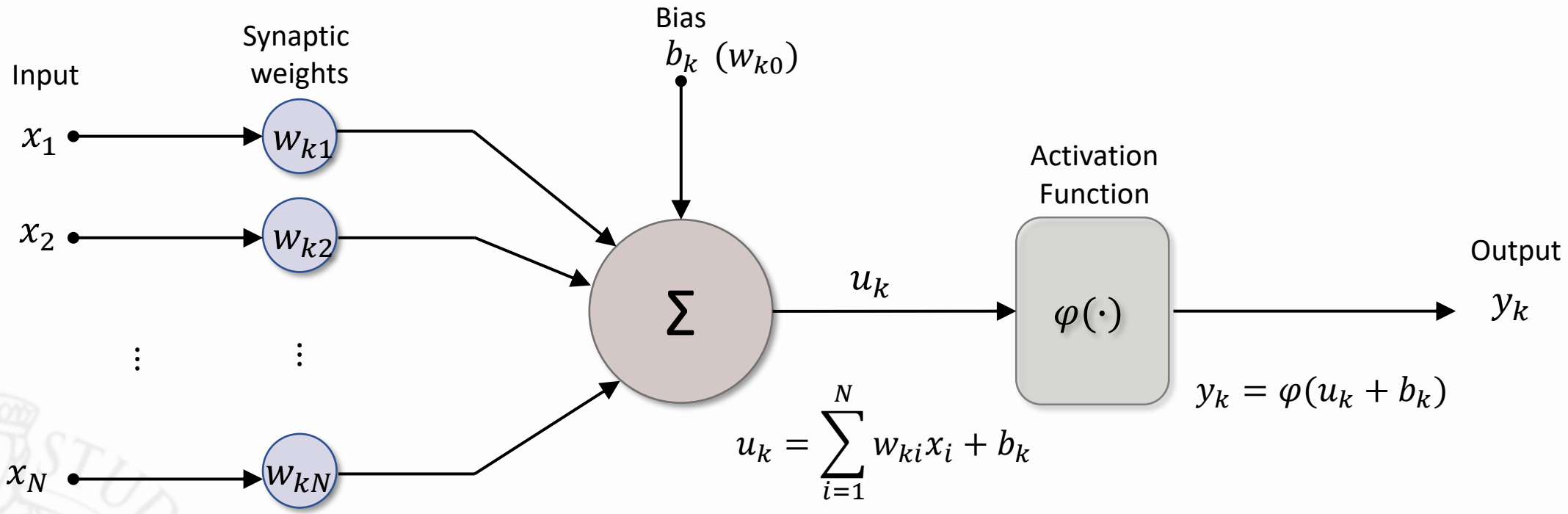
4. Deep Learning – Introduzione

- Il *Deep Learning* è una modalità di apprendimento basata su reti neurali (*Artificial Neural Networks* – *ANN*), le quali posso ricercare pattern, effettuare predizioni, esplicitare relazioni ed associazioni esistenti tra un insieme di coppie di dati di input e di output.

- La terminologia «*Deep Learning*» si basa sul concetto di profondità di una rete neurale, che è associato al numero di strati interconnessi (*layers*) di celle neurali di cui essa è composta. Tale parametro è rappresentativo quindi della complessità della rete neurale.

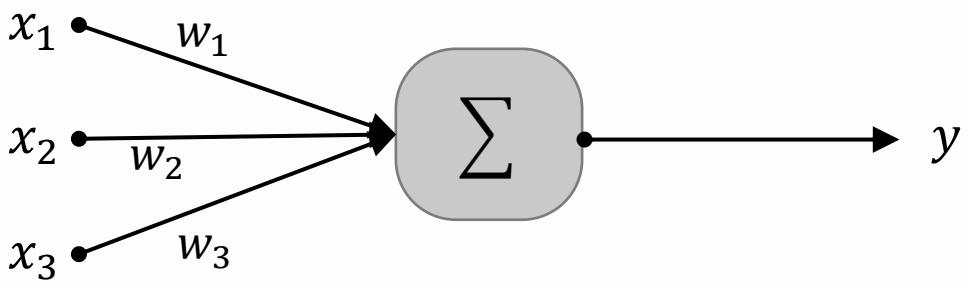
4. Deep Learning – Modello del neurone: perceptron

- L'unità base di information processing è il ***perceptron***. Una rete neurale è formato da uno (single-layer perceptron) o più (multi-layer perceptron) strati (layers) di neuroni, interconnessi tramite collegamenti caratterizzati da pesi (***synaptics***).
- Ciascun neurone riceve in ingresso degli input x_i, li combina linearmente e li propaga attraverso una funzione di attivazione.



4. Deep Learning – Single-Layer Perceptron

- Single Layer Perceptron ANN:



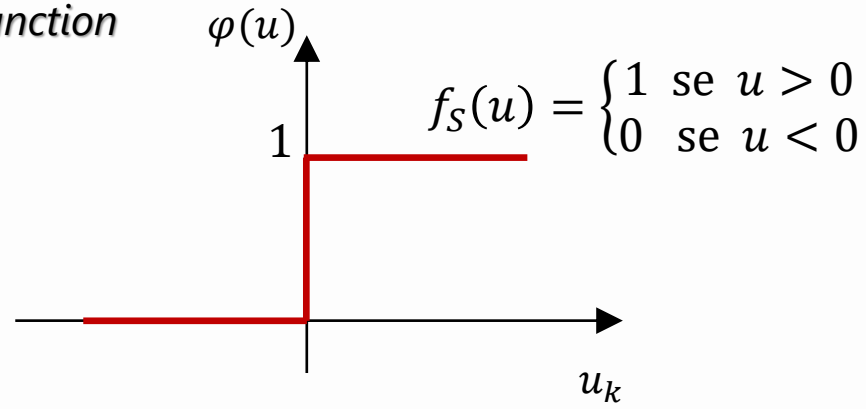
$$y = \begin{cases} 1 & \text{se } \sum_i w_i x_i > th \\ 0 & \text{se } \sum_i w_i x_i < th \end{cases} \quad \Rightarrow \quad y = \begin{cases} 1 & \text{se } \sum_i w_i x_i + b > 0 \\ 0 & \text{se } \sum_i w_i x_i + b < 0 \end{cases}$$

- Con questa funzione di attivazione (step-function, funzione gradino, o funzione di Heaviside), può accadere che modeste variazioni degli input producano ampie variazioni dell'output (e questo può propagarsi con effetti inattesi in reti più profonde)

4. Deep Learning – Activation Functions

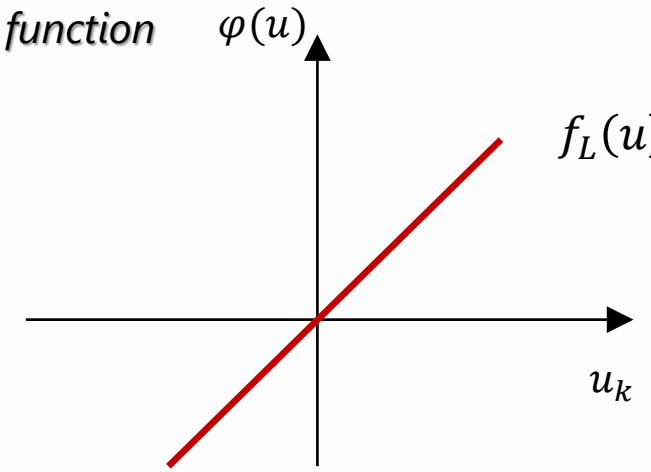
- Differenti tipologie di *activation function*:

(a) Step-function



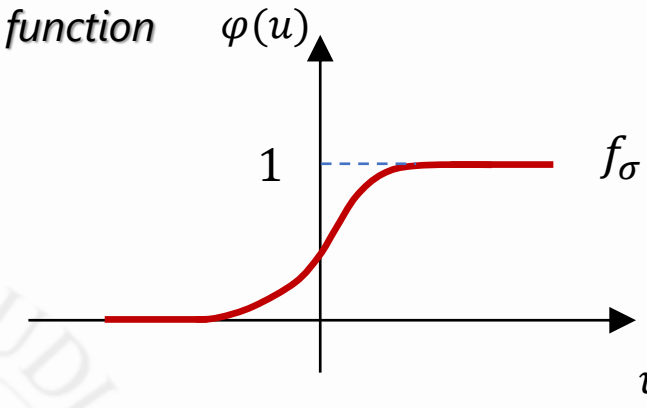
$$f_S(u) = \begin{cases} 1 & \text{se } u > 0 \\ 0 & \text{se } u < 0 \end{cases}$$

(b) Linear function



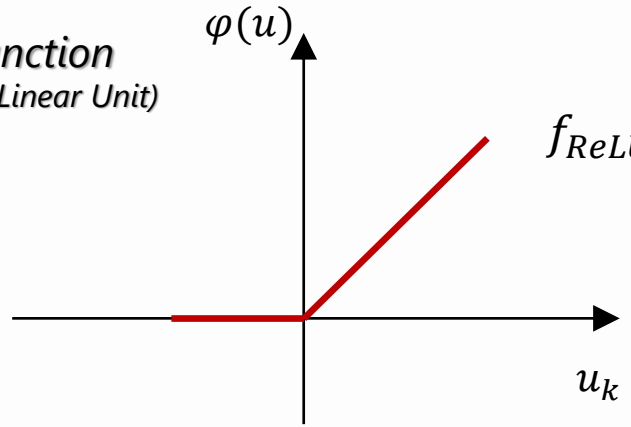
$$f_L(u) = u$$

(c) Sigmoid function



$$f_\sigma(u) = \frac{1}{1 + e^{-u}}$$

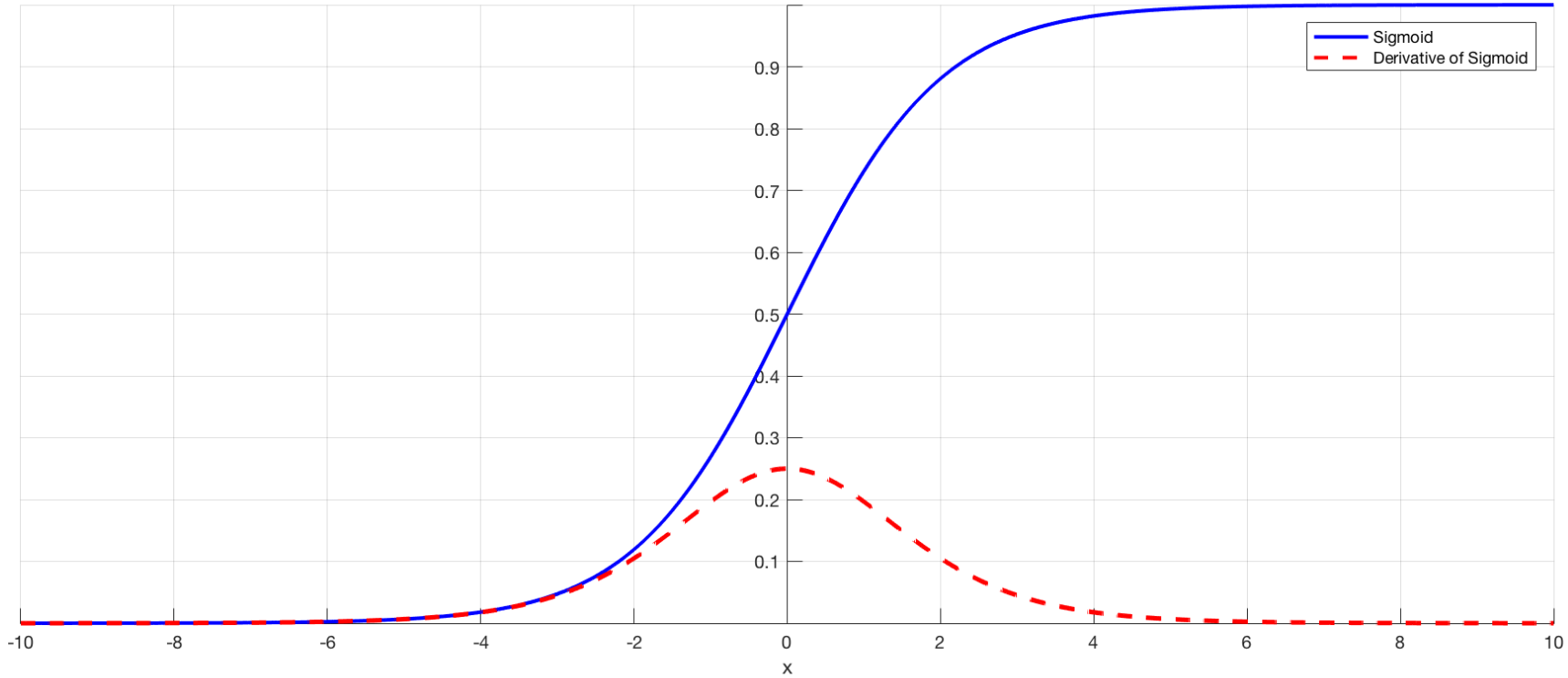
(d) ReLU function
(Rectified Linear Unit)



$$f_{ReLU}(u) = max\{0, u\}$$

4. Deep Learning – Funzioni di Attivazione

- Implicazioni della scelta della funzione di attivazione per l'inizializzazione dei pesi e dei bias:



4. Gradient Descent

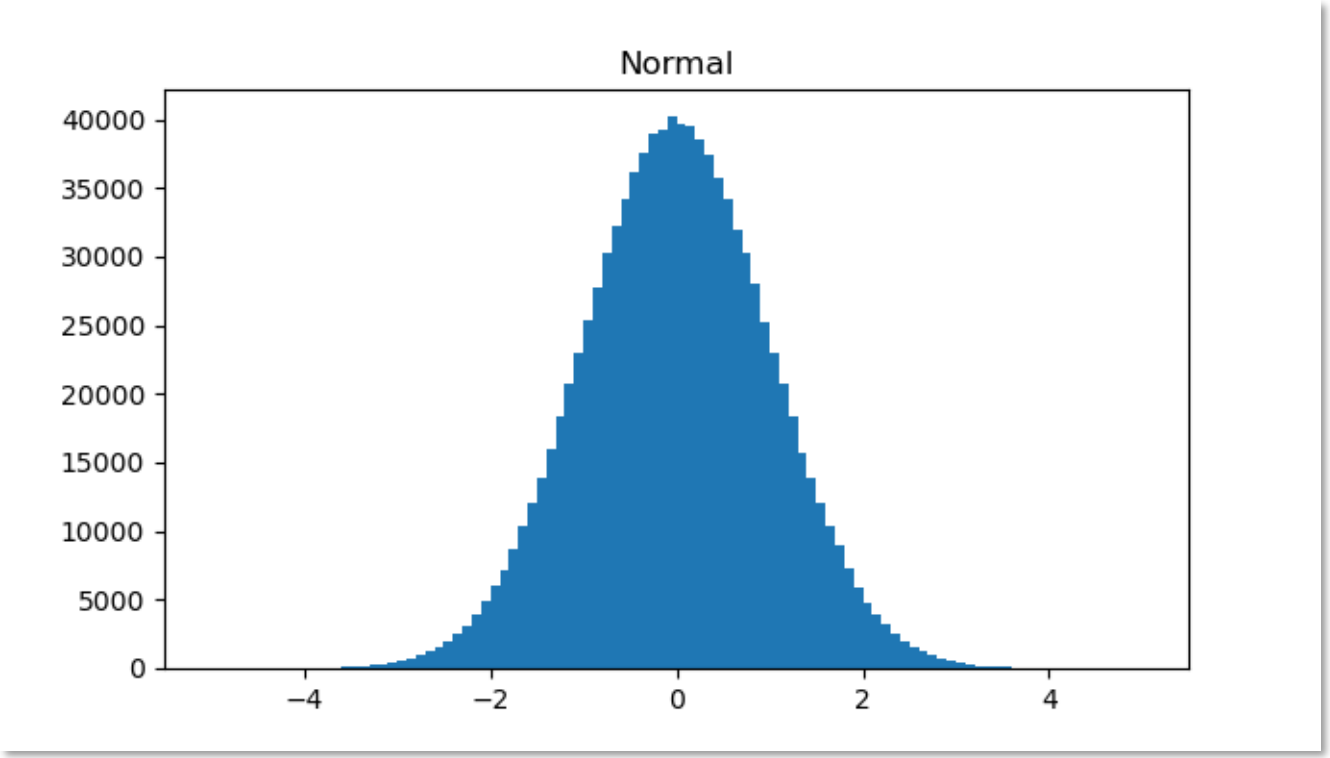
$$cf = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{y}_i - A(\boldsymbol{w}x_i + \boldsymbol{b})\right)^2$$

$$\frac{\partial cf}{\partial \boldsymbol{w}} = \frac{1}{N}\sum_{i=1}^{N} -2x_i\left(\hat{y}_i - A(\boldsymbol{w}x_i + \boldsymbol{b})\right) \qquad \frac{\partial cf}{\partial b} = \frac{1}{N}\sum_{i=1}^{N} -2\left(\hat{y}_i - A(\boldsymbol{w}x_i + \boldsymbol{b})\right)$$

$$\boldsymbol{w} = \boldsymbol{w} - L\frac{\partial cf}{\partial \boldsymbol{w}} \qquad \boldsymbol{b} = \boldsymbol{b} - L\frac{\partial cf}{\partial \boldsymbol{b}}$$

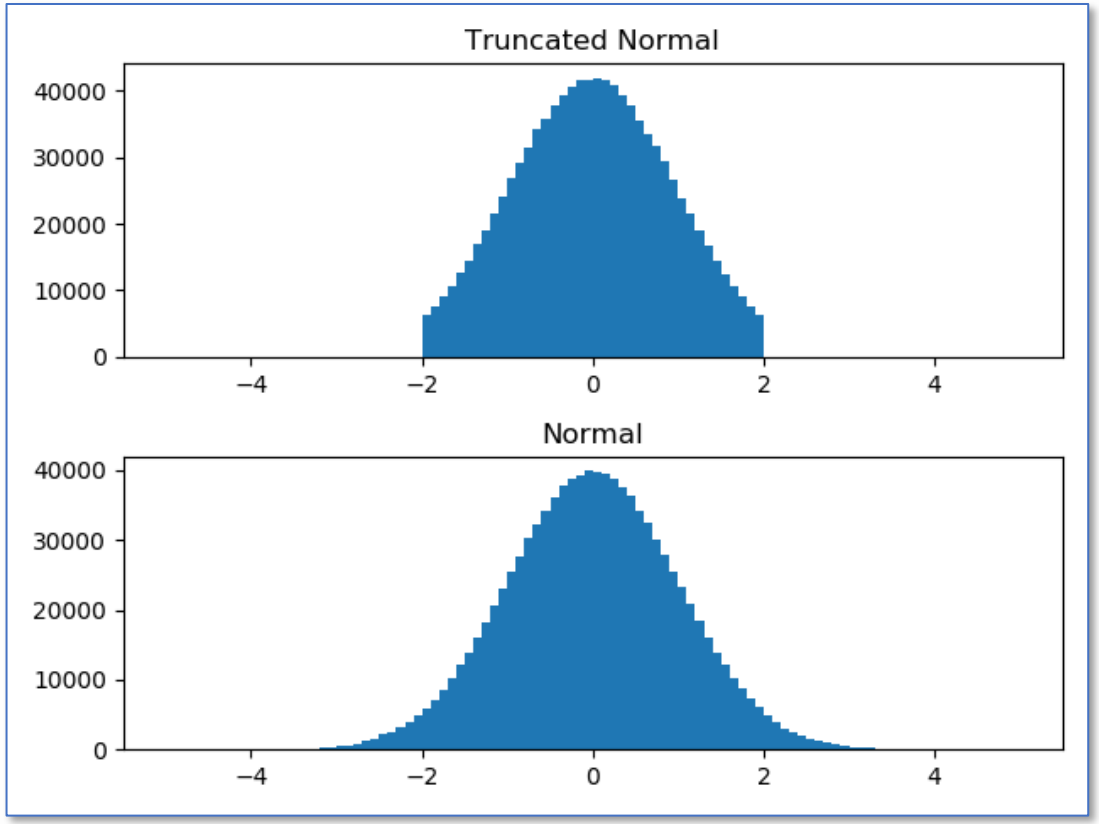
4. Deep Learning – Inizializzazione Pesi e Bias: tf.random_normal VS tf.truncated_normal

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

import matplotlib.pyplot as plt

n = 1000000
random_norm = tf.random_normal((n,))
with tf.Session() as sess:
    w = sess.run(random_norm)

plt.figure()
plt.title("Normal Distribution")
plt.hist(w, 100, (-5, 5));
plt.show()
```



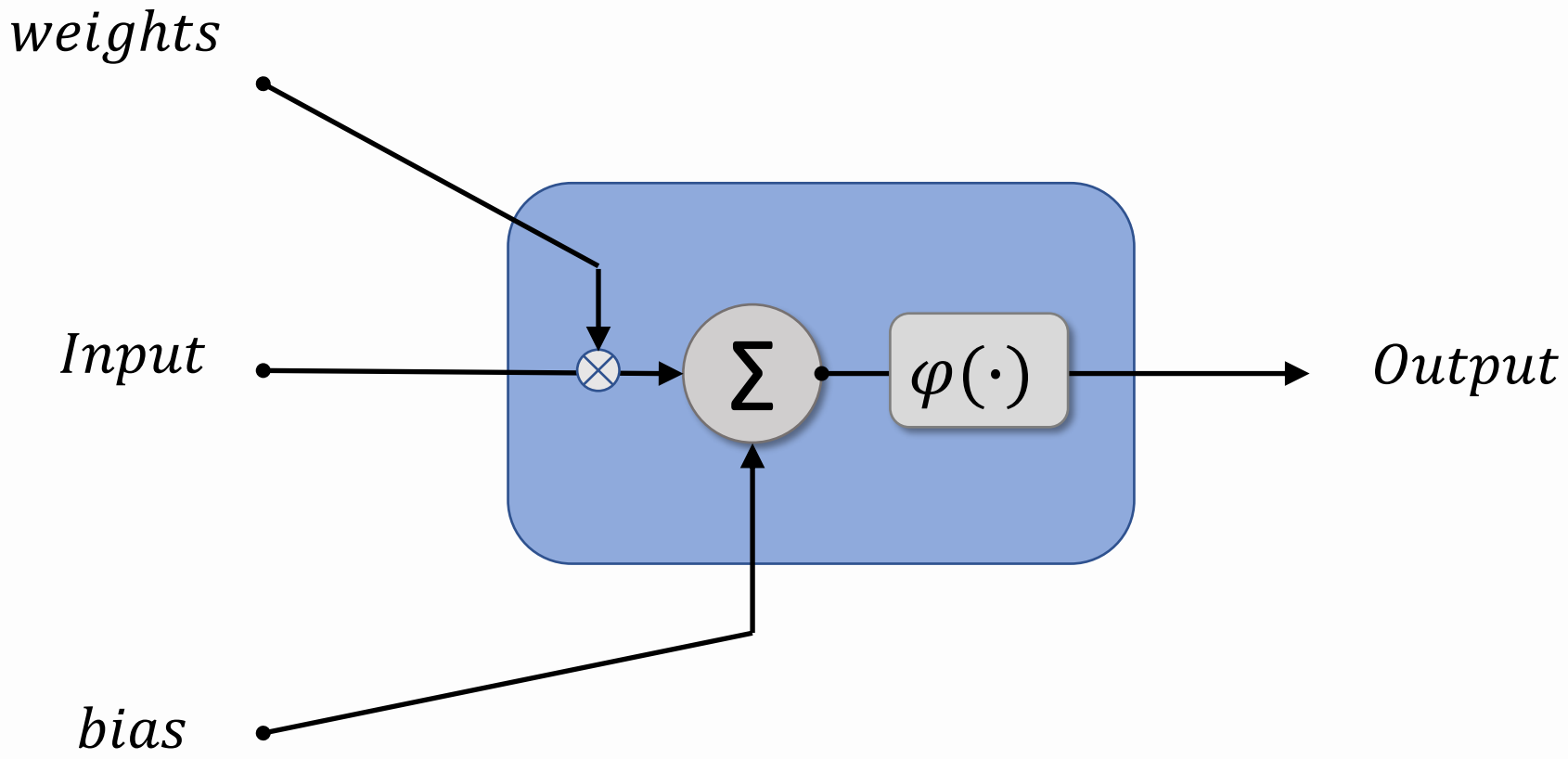
4. Deep Learning – Inizializzazione Pesi e Bias: tf.random_normal VS tf.truncated_normal

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

import matplotlib.pyplot as plt

n = 1000000
trunc_norm = tf.truncated_normal((n,))
random_norm = tf.random_normal((n,))
with tf.Session() as sess:
    a, b = sess.run([trunc_norm, random_norm])

fig, subplt = plt.subplots(2)
subplt[0].set_title("Truncated Normal")
subplt[0].hist(a, 100, (-5, 5));
subplt[1].set_title("Normal")
subplt[1].hist(b, 100, (-5, 5));
plt.show()
```



4. Deep Learning – Differenti Tipologie di Reti Neurali

- Rappresentazione di uno strato (layer) a singola cella (perceptron o neurone) di tipo generico:

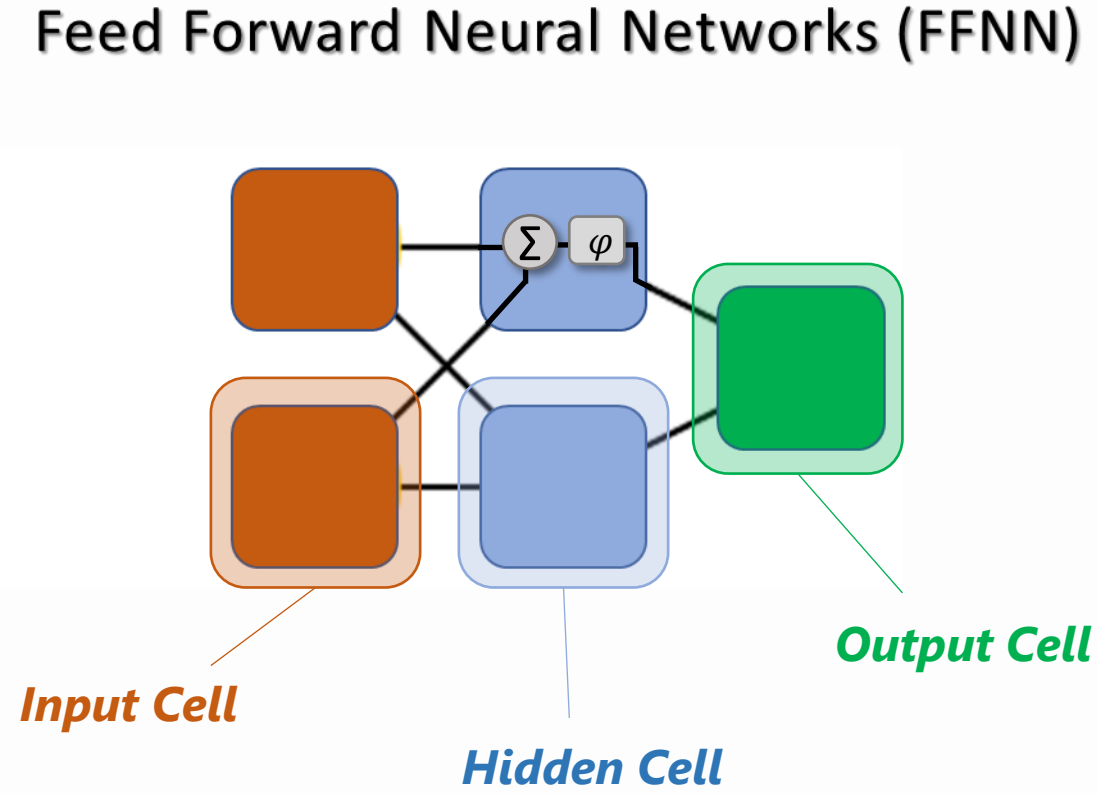


4. Deep Learning – Topologia e Modelli di Neural Networks

Topologie e Modelli di Neural Networks

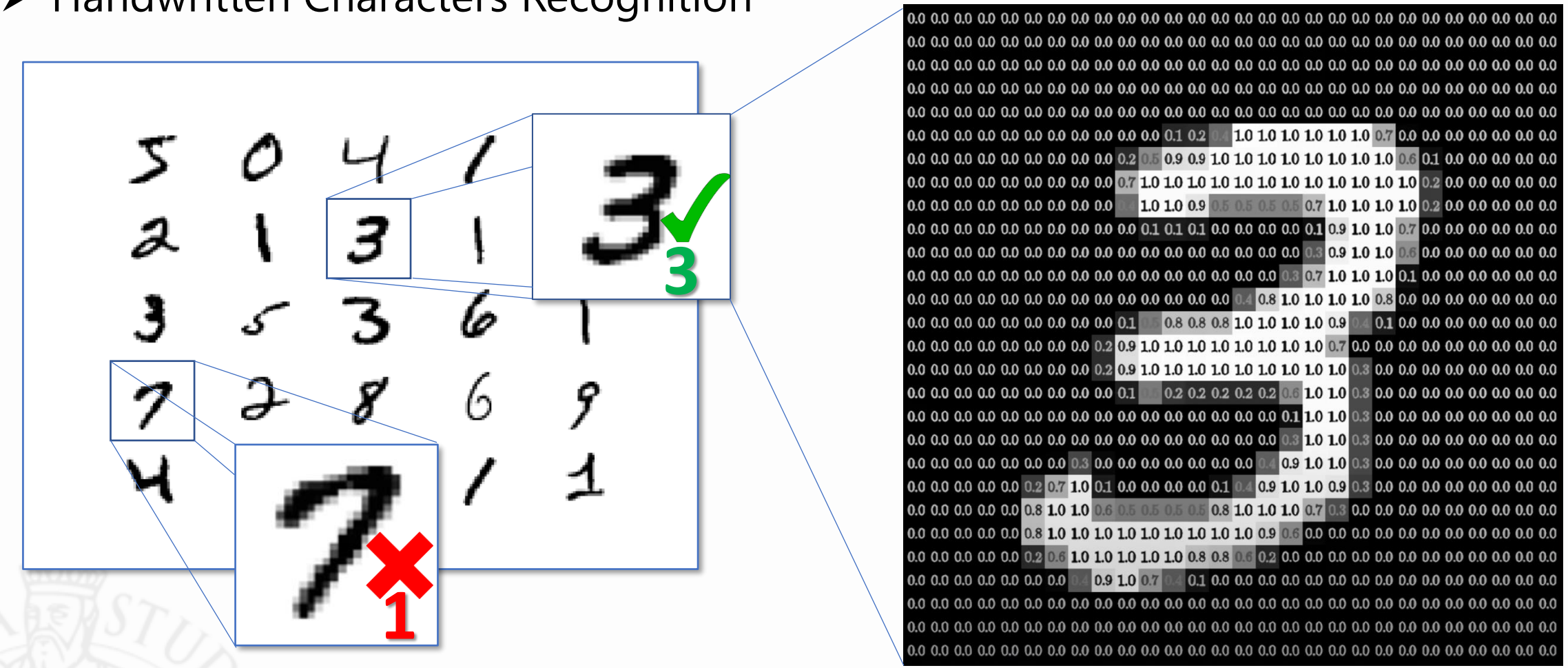
4. Deep Learning – Multi-Layer Perceptron: Feed Forward Network

- È la tipologia di rete neurale più semplice dopo il Single-Layer Perceptron, di cui ne è un'estensione (chiamata infatti anche Multi-Layer Perceptron, MLP). Prevede un layer di *input*, un layer di *output*, e uno (o più) layers intermedi (*hidden layers*), in genere densamente connessi tra loro (*fully connected layers*). Il nome deriva dal fatto che l'informazione viene propagata in avanti dal layer di input al layer di output (anche se nel training sono previste tecniche di ottimizzazione tramite backpropagation).



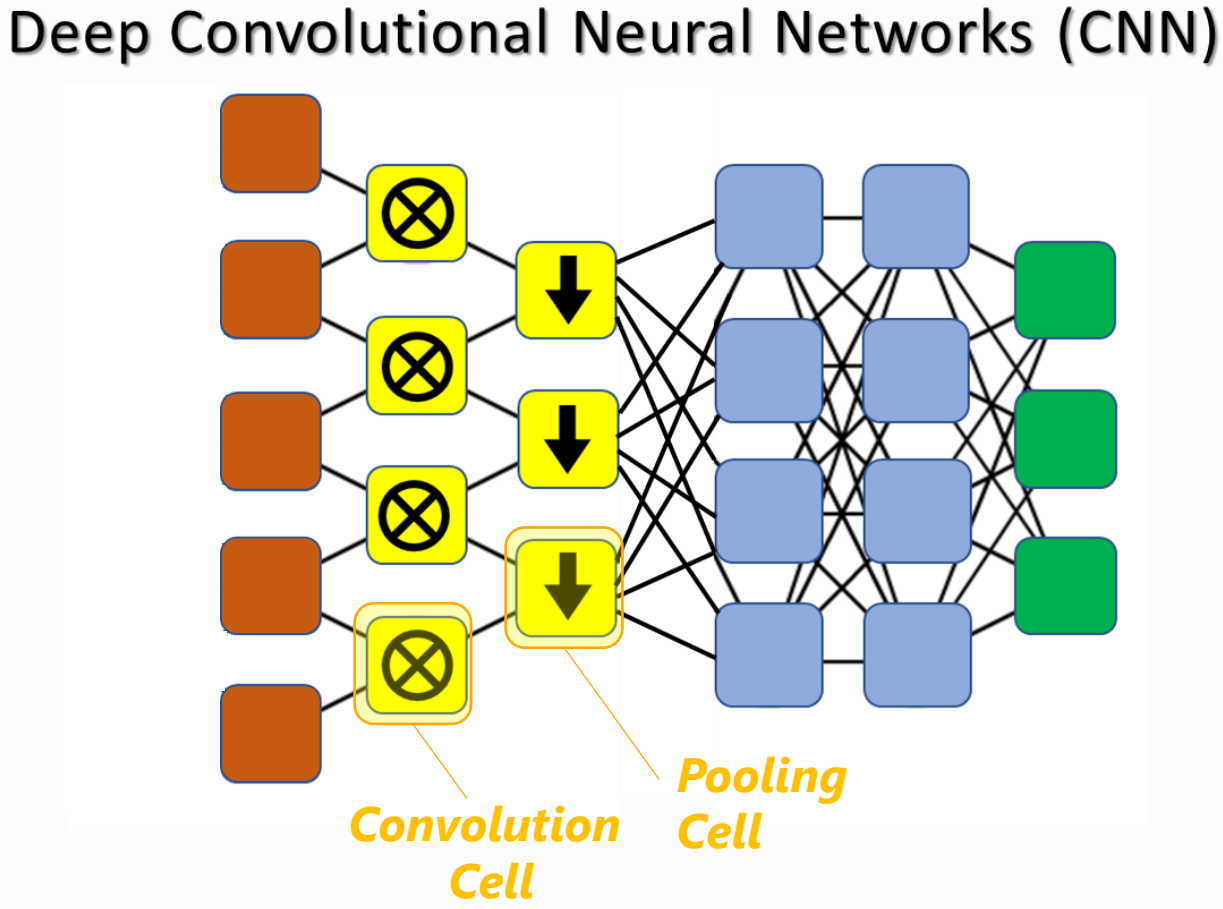
1. Introduzione – Applicazioni del *Machine Learning*

- Handwritten Characters Recognition



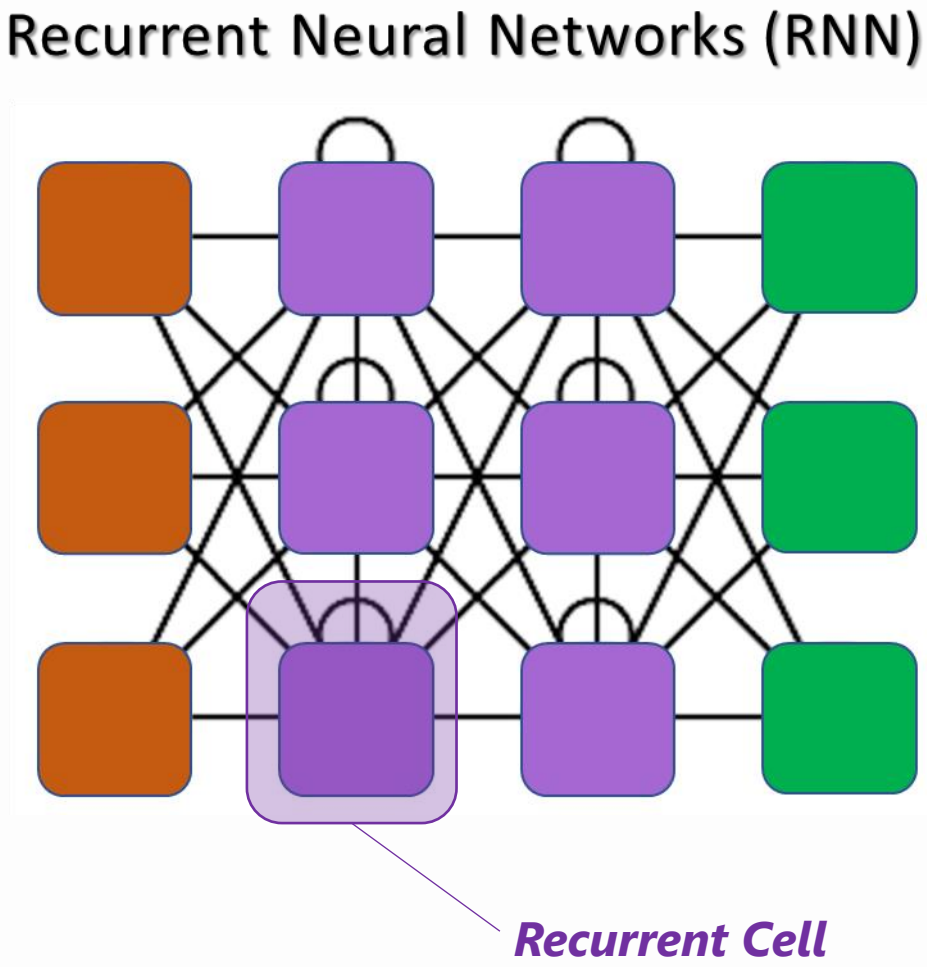
4. Deep Learning – Convolutional Neural Networks (CNN)

- Le *Convolutional Neural Networks* sono spesso usate nell'ambito della classificazione di features in immagini e video. Nel caso siano applicate ad un dataset di immagini, ad esempio, un filtro mobile (*Kernel*) scorre sull'immagine, eseguendo un'operazione di convoluzione tra i valori del filtro e i valori dell'area di immagine selezionata (*Convolution* Layer) per l'estrazione di features. Successivamente, un layer di *Pooling* esegue un downsampling dell'immagine per ridurre i parametri e la computazione. Eventualmente vengono applicati altri layer di convoluzione e pooling in reti a convoluzione profonde (*Deep CNN*) per l'estrazione di features più complesse.



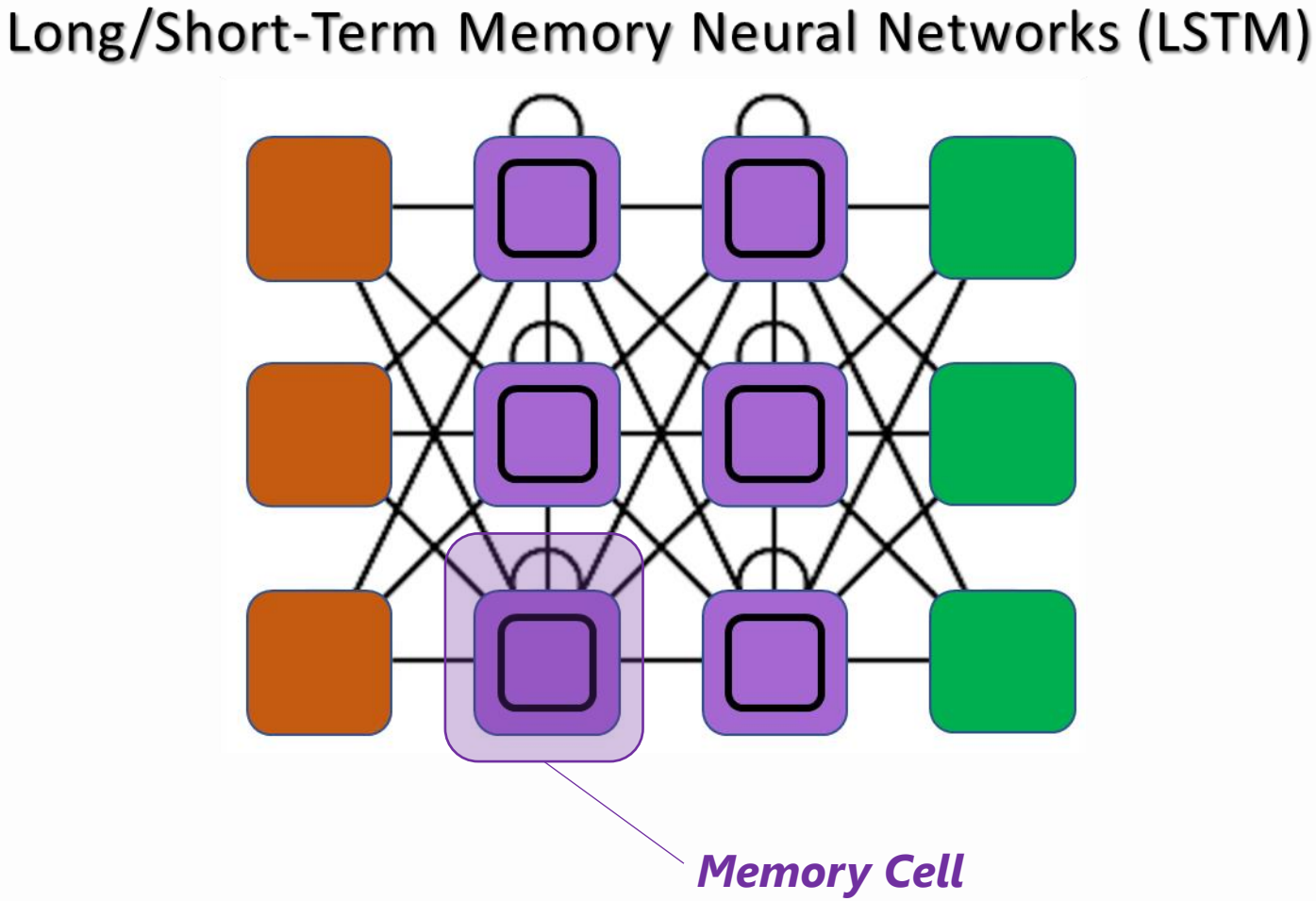
4. Deep Learning – Recurrent Neural Networks

- In questa categoria di reti neurali, i valori in input ad ogni layer non sono rappresentati solamente dagli output del layer precedente, ma anche dagli output dello stesso layer in istanti precedenti. La rete in questo caso conserva memoria del suo stato per aumentare la capacità di apprendimento.
- Il principale difetto di questo tipo di NN è la propensione agli effetti di *Gradient Vanishing* e *Gradient Exploding*.



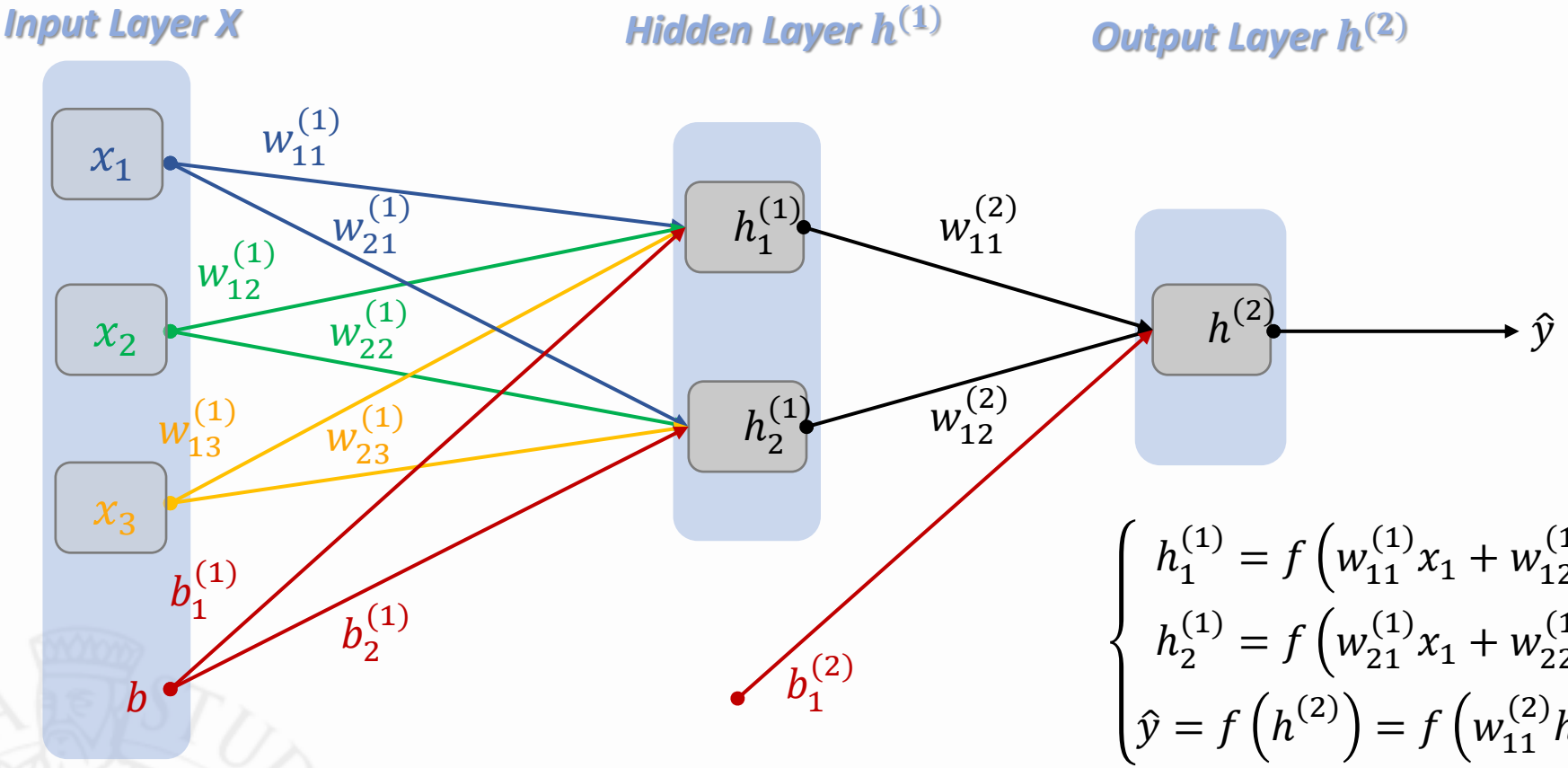
4. Deep Learning – Long/Short-Term Neural Networks (LSTM)

- Le reti *Long/Short-Term Memory (LTSM)* sono un tipo di RNN in cui ogni neurone è costituito da una cella di memoria e tre gate: *input*, *output* e *forget*. Il gate di *input* determina la quantità di informazioni del layer precedente da memorizzare nella cella corrente. Il gate di *output* determina la quantità di informazione che il layer successivo riceverà. Il gate *forget* determina il rate con cui la cella corrente resetta il proprio stato (dimenticando quindi il contesto precedente).



4. Deep Learning – Architettura e Topologia di una Feed Forward Neural Network (FFNN)

- Feed Forward Propagation:



$$\begin{cases} h_1^{(1)} = f\left(w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3 + b_1^{(1)}\right) \\ h_2^{(1)} = f\left(w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3 + b_2^{(1)}\right) \\ \hat{y} = f\left(h^{(2)}\right) = f\left(w_{11}^{(2)}h_1^{(2)} + w_{12}^{(2)}h_2^{(2)} + b_1^{(2)}\right) \end{cases}$$

4. Deep Learning – Feed Forward Neural Network (FFNN) in Python

```
import numpy as np

w1 = np.random.uniform(0, 1, size=(2, 3))
w2 = np.random.uniform(0, 1, size=2)
b1 = np.random.uniform(0, 1, size=2)
b2 = np.random.uniform(0, 1, size=1)

def f(x):
    return 1 / (1 + np.exp(-x))

def forward_NN (n_layers, x, w, b):
    for l in range(n_layers):
        if l == 0:
            node_in = x
        else:
            node_in = h
        z = w[l].dot(node_in) + b[l]
        h = f(z)
        if l == 0:
            print('h1_2 = ', h[0], 'h2_2 = ', h[1])
    return h
```

```
w = [w1, w2]
b = [b1, b2]

x = [1., 2., 3.]

h3 = forward_NN(2, x, w, b)
print('h1_3 = ', h3[0])
```

$$W^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{pmatrix}$$

$$W^{(2)} = \begin{pmatrix} w_{11}^{(2)} & w_{12}^{(2)} \end{pmatrix}$$

$$b^{(1)} = \begin{pmatrix} b_1^{(1)} & b_2^{(1)} \end{pmatrix}$$

$$b^{(2)} = b_1^{(2)}$$

```
Console ----------------------------------------
h1_2 =  0.9769464396533163 h2_2 =  0.970194796360446
h1_3 =  0.7834781501970273
```

4. Deep Learning – Dimensionamento degli Iperparametri

- *layer di input*: il numero di neuroni è pari alla dimensione degli input (features)
- *layer di output*: il numero di neuroni è pari alla dimensione dell'output atteso (classi)
- *Hidden Layer*: non è sempre possibile determinare analiticamente il numero di neuroni per l' hidden layer, nonché il numero di hidden layers. Il metodo migliore è quello di usare algoritmi di *cross-validation* iterativi che determinano dinamicamente il numero ottimale di neuroni per ciascun hidden layer, ed eventualmente anche il numero di hidden layers, con l'obiettivo di minimizzare l'errore di predizione nella fase di test.

  Altri approcci empirici (*Rule of Thumb*):

  - "A rule of thumb is for the size of this [hidden] layer to be somewhere between the input layer size ... and the output layer size ..." [Blum, A. (1992), Neural Networks in C++, NY: Wiley]
  - "To calculate the number of hidden nodes we use a general rule of: (Number of inputs + outputs) * (2/3)" [from the FAQ for a commercial neural network software company].
  - "You will never require more than twice the number of hidden units as you have inputs in a Multi-Layer Perceptron with one hidden layer" [Swingler, K. (1996), Applying Neural Networks: A Practical Guide, London: Academic Press]

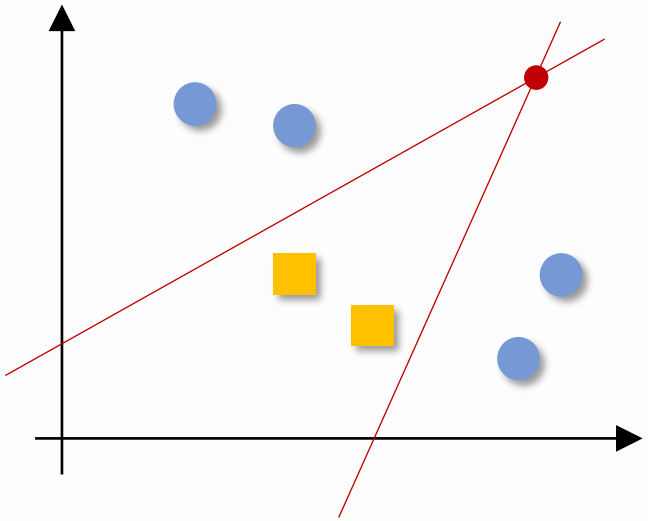
4. Deep Learning – Dimensionamento del numero di hidden layers

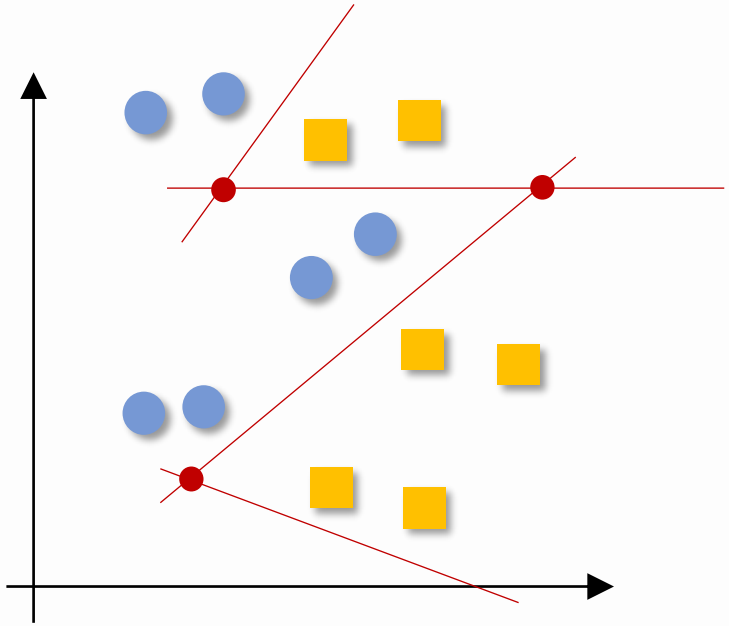
- Generalmente una rete neurale non richiede *hidden layers* se i dati sono linearmente separabili. Un singolo hidden layer viene considerato sufficiente nella prima letteratura per la maggior parte dei problemi: "*any continuous function can be represented by a neural network that has only one hidden layer with exactly 2n+1 nodes, where n is the number of input nodes*" (Hect-Nielsen, 1987).

- Un numero maggiore di *hidden layers* può essere utilizzato per modellare comportamenti non lineari o descrivere features più complesse, e anche a ridurre il numero di neuroni totali con cui verrebbe implementata l'architettura con un singolo hidden layer.

4. Deep Learning – Dimensionamento del numero di hidden layers

- Partendo dall'idea che una rete neurale (multi-layer perceptron) può essere rappresentata come una combinazione di più single-layer perceptron, ognuno dei quali si comporta come un classificatore lineare, è possibile derivare un approccio con cui dimensionare il numero di *hidden layers*, ad esempio in un problema di classificazione.





- Per quanto riguarda la determinazione di altri parametri e iperparametri della rete (numero di neuroni, numero di epochs su cui fare l'addestramento ecc.) spesso si ricorre a tecniche iterative di ottimizzazione, che tendono a massimizzare l'accuratezza di predizione/classificazione o a minimizzare l'errore.