

Multi Layer Perceptron (MLP)

Reti Neurali Totalmente Connesse

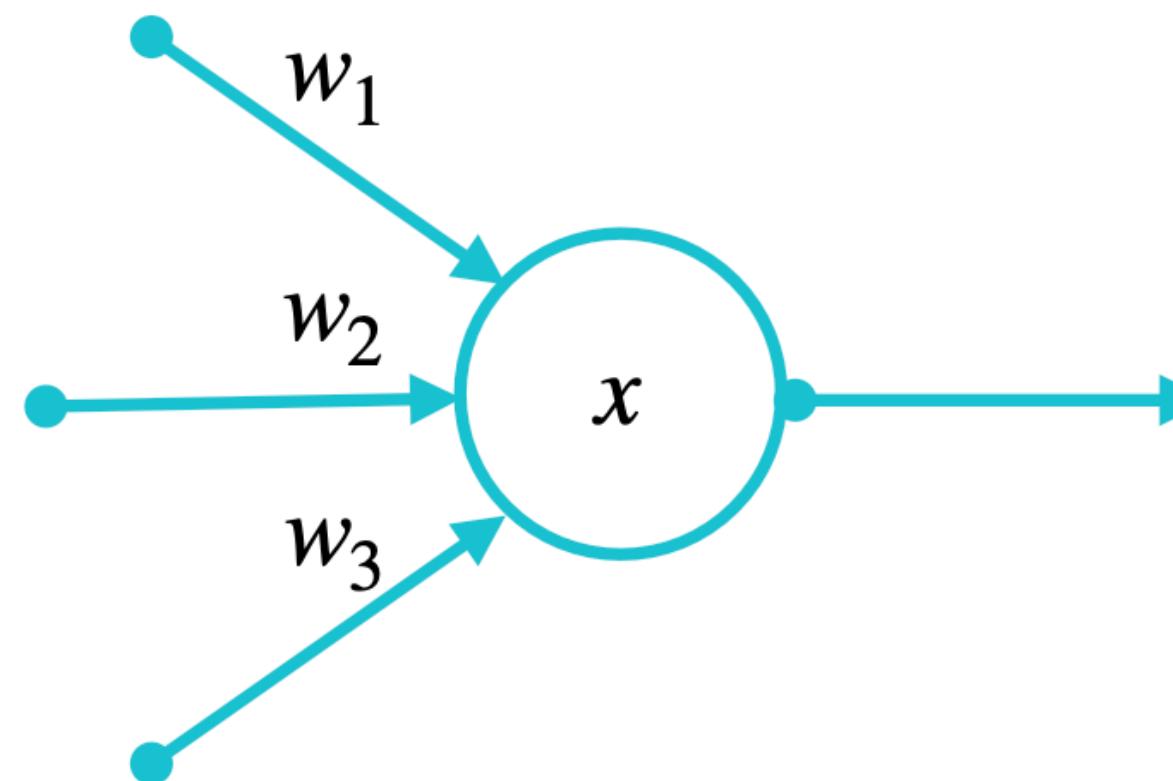
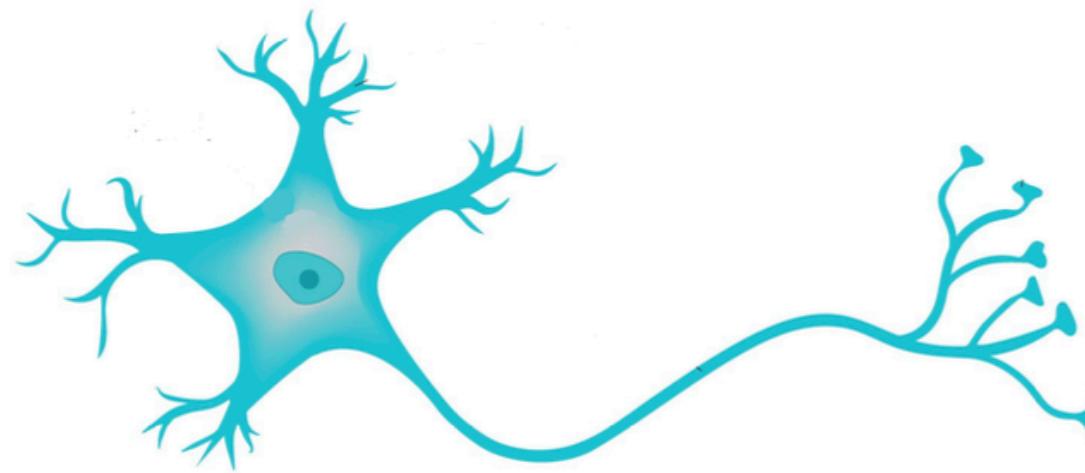
Laura Nenzi, Gloria Pietropolli, Gaia Saver



UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

Recap: Le Reti Neurali

Ogni neurone esegue una funzione semplice



STEP 1: Combinazione lineare del valore dei neuroni dello strato precedente, usando come coefficienti i pesi delle connessioni

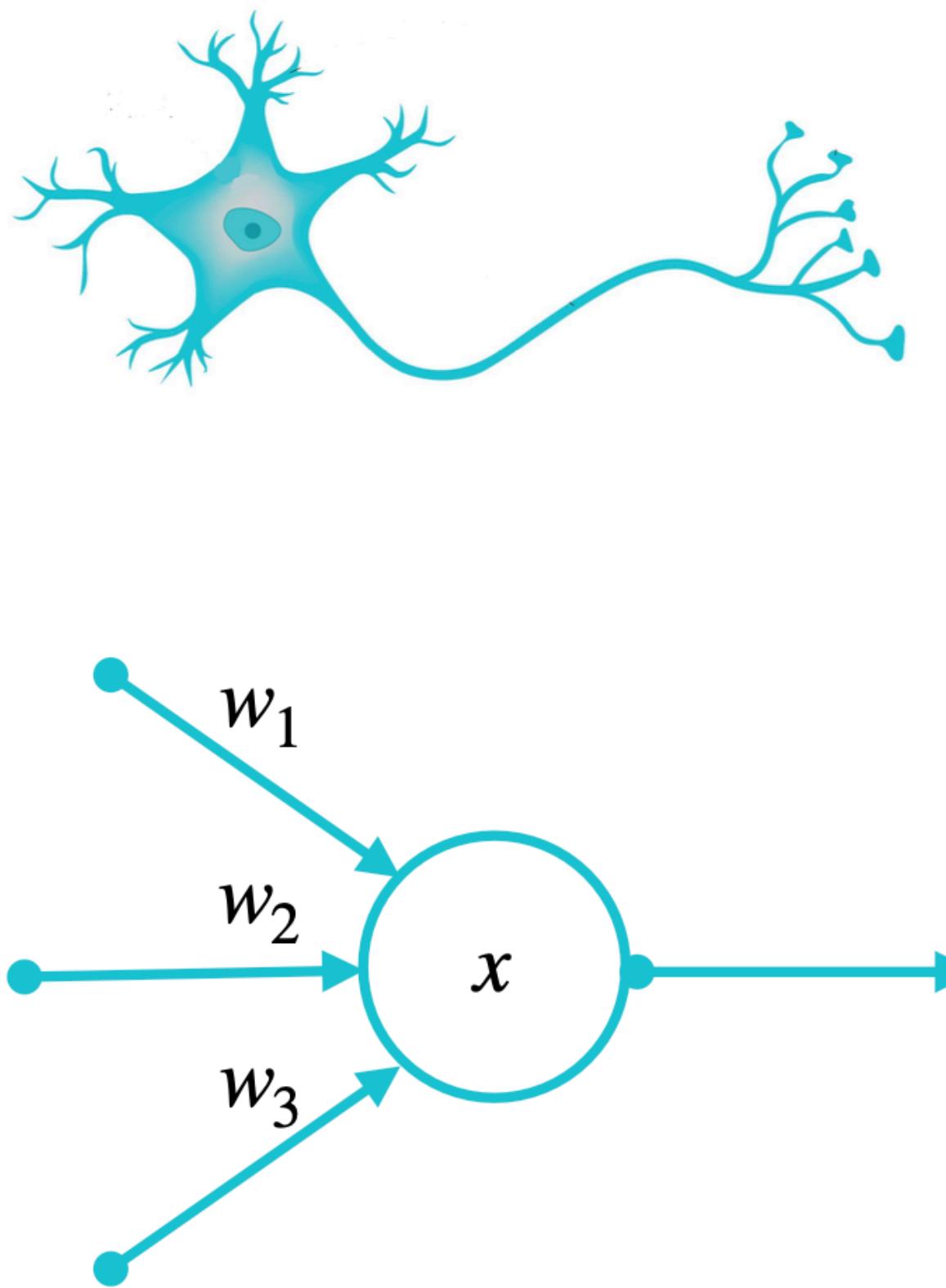
$$x = w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + b$$

STEP 2: Trasformare la combinazione lineare con una funzione non lineare (chiamata attivazione)

$$x = \sigma(w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + b)$$

Recap: Le Reti Neurali

Come funziona un singolo neurone?



Ogni neurone:

- Riceve un input numerico.
- Applica una funzione matematica.
- Produce un output.

Esempio:

- Input: 2, Peso: 0.5, Bias: 1.
- Output = funzione
 $(2 \times 0.5 + 1)$

PROVIAMO AD
IMPLEMENTARE IL
NOSTRO PRIMO
NEURONE

**Vogliamo iniziare a implementare i nostri
primi neuroni e le nostre prime reti neurali**

**Vogliamo iniziare a implementare i nostri
primi neuroni e le nostre prime reti neurali**

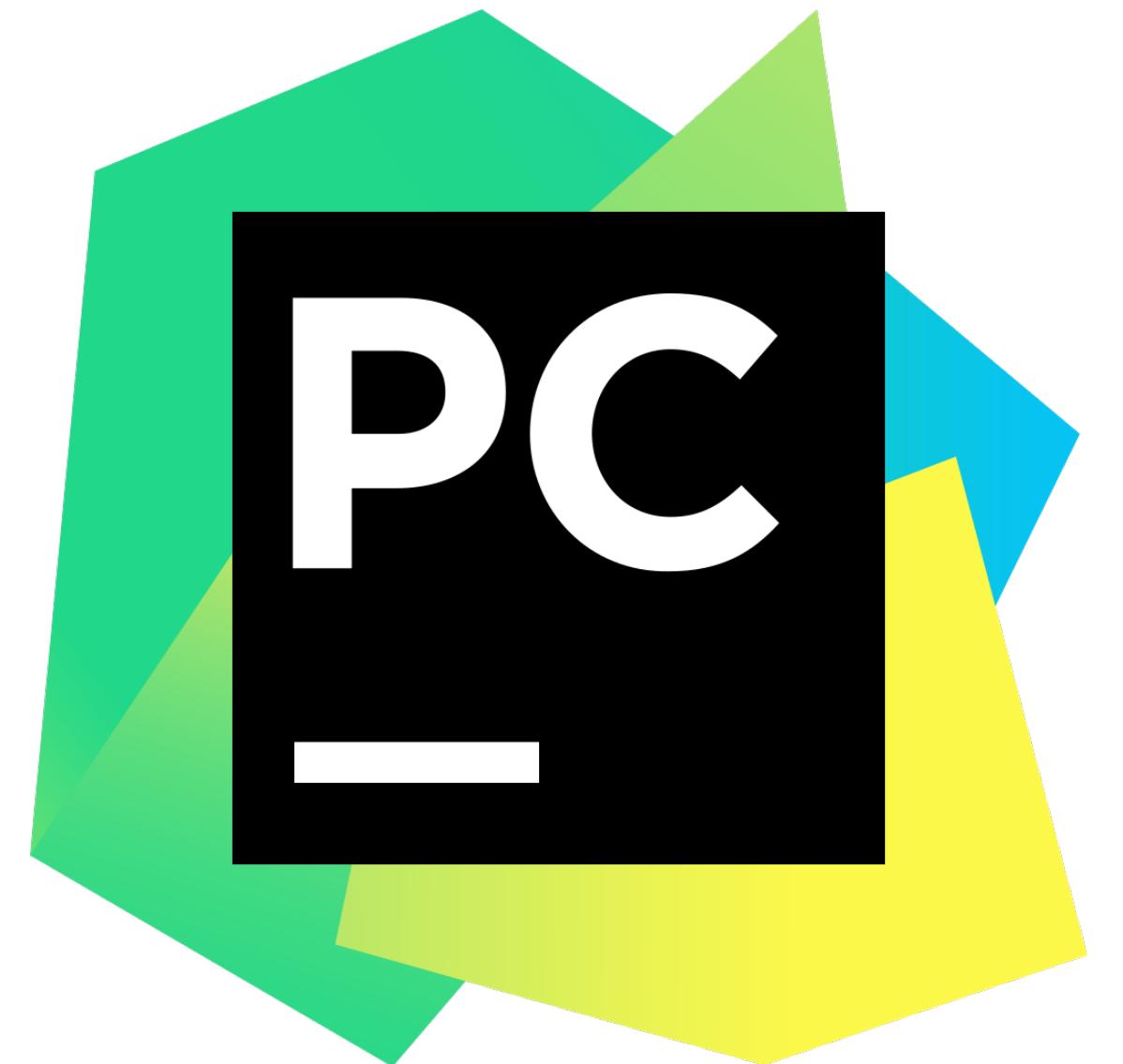
Quale è il modo migliore per farlo?

Pycharm

Come programmare con un IDE

Cos'è PyCharm?

- PyCharm è un ambiente di sviluppo integrato (IDE) per Python.
- Ti aiuta a:
 - Scrivere codice in modo **ordinato e veloce**.
 - **Trovare errori** prima di eseguire il programma.
 - Usare **strumenti avanzati** (debugging, completamento automatico).



Pycharm

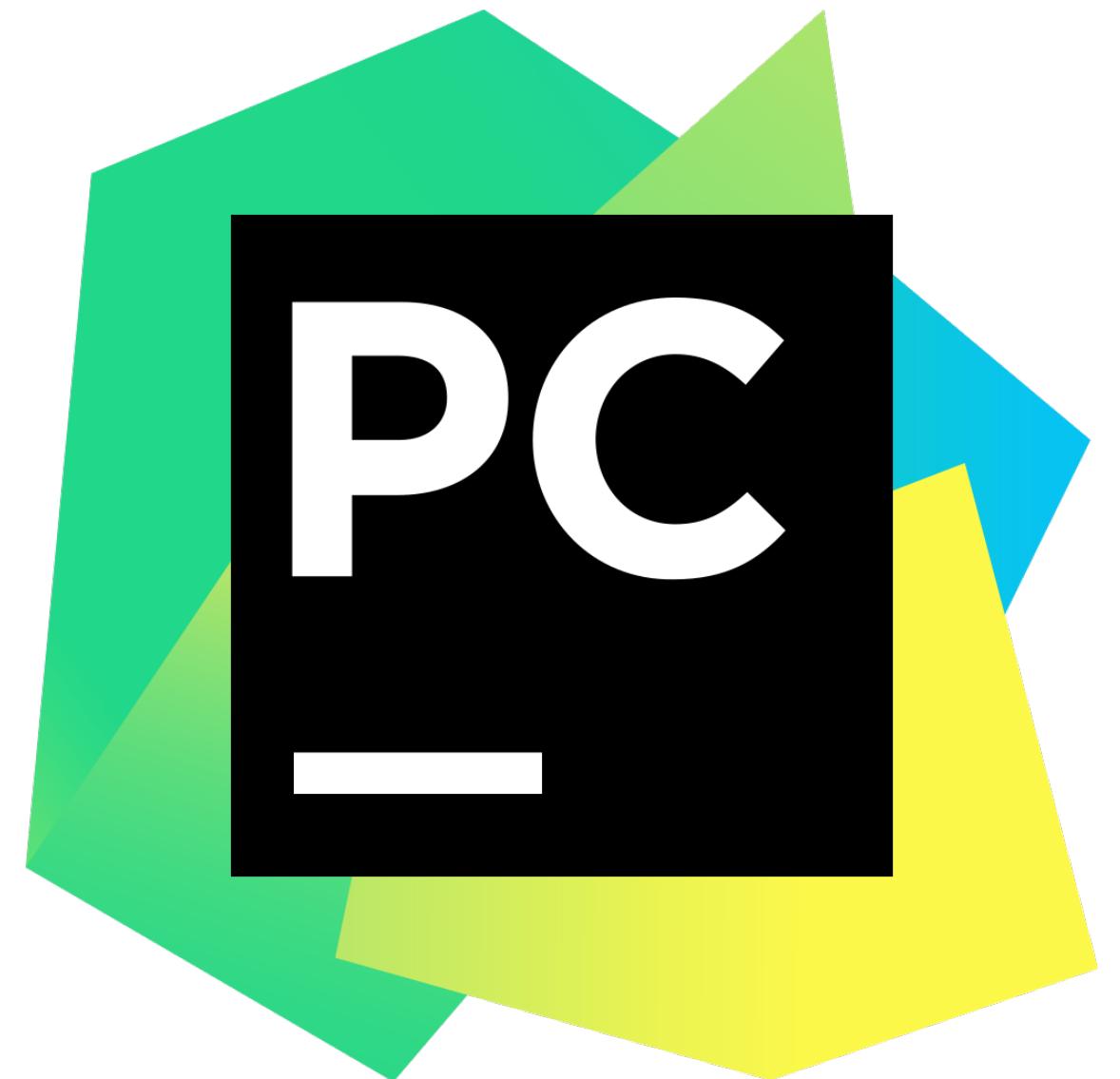
Come programmare con un IDE

Cos'è PyCharm?

- PyCharm è un ambiente di sviluppo integrato (IDE) per Python.
- Ti aiuta a:
 - Scrivere codice in modo **ordinato e veloce**.
 - **Trovare errori** prima di eseguire il programma.
 - Usare **strumenti avanzati** (debugging, completamento automatico).

Perché usare PyCharm?

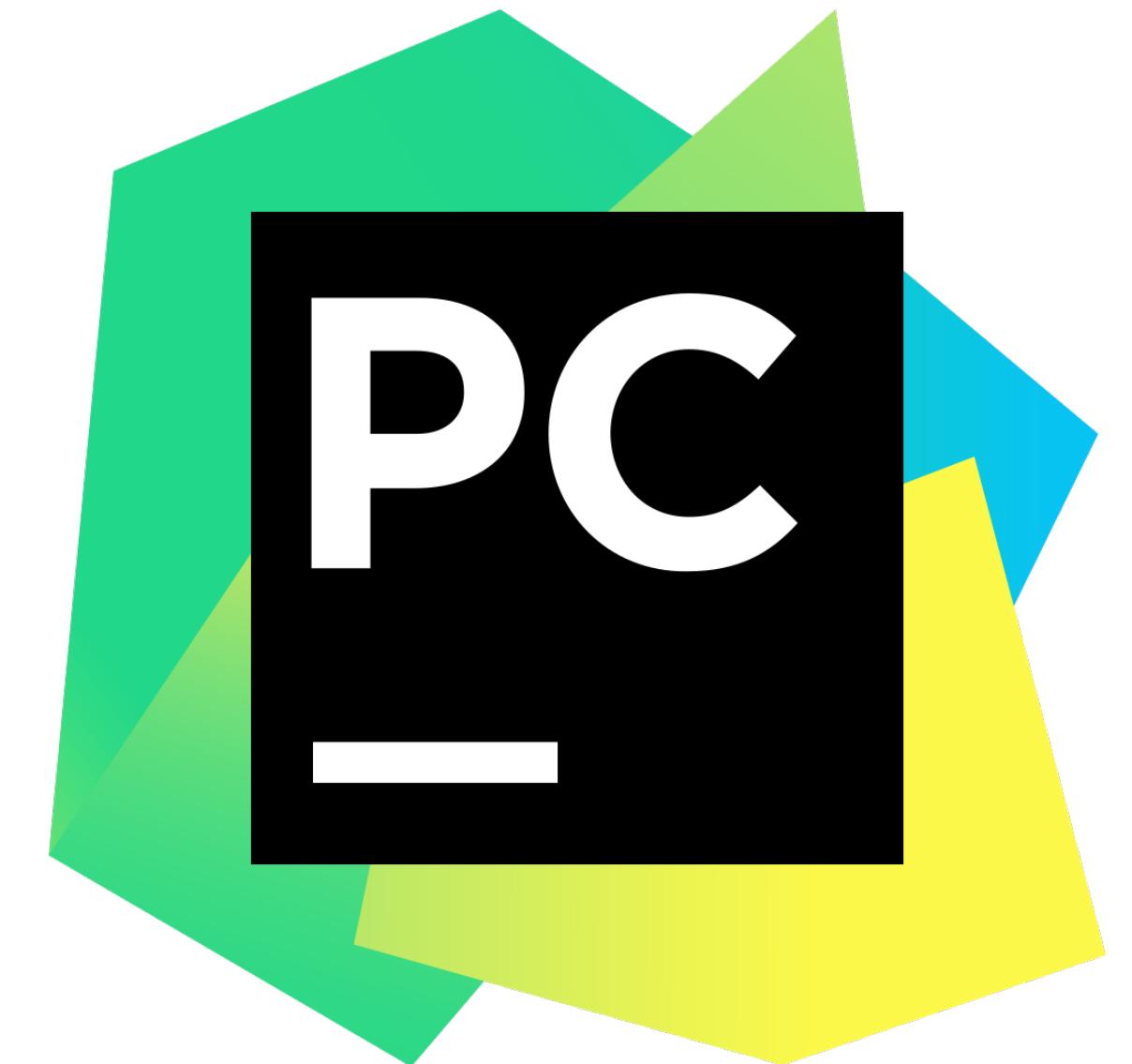
- Facilità d'uso: Interfaccia semplice per principianti.
- Supporto per Python: Strumenti dedicati per lavorare con Python.
- Organizzazione del progetto: Tutti i file e le librerie in un unico posto.



Pycharm

Creazione del primo progetto

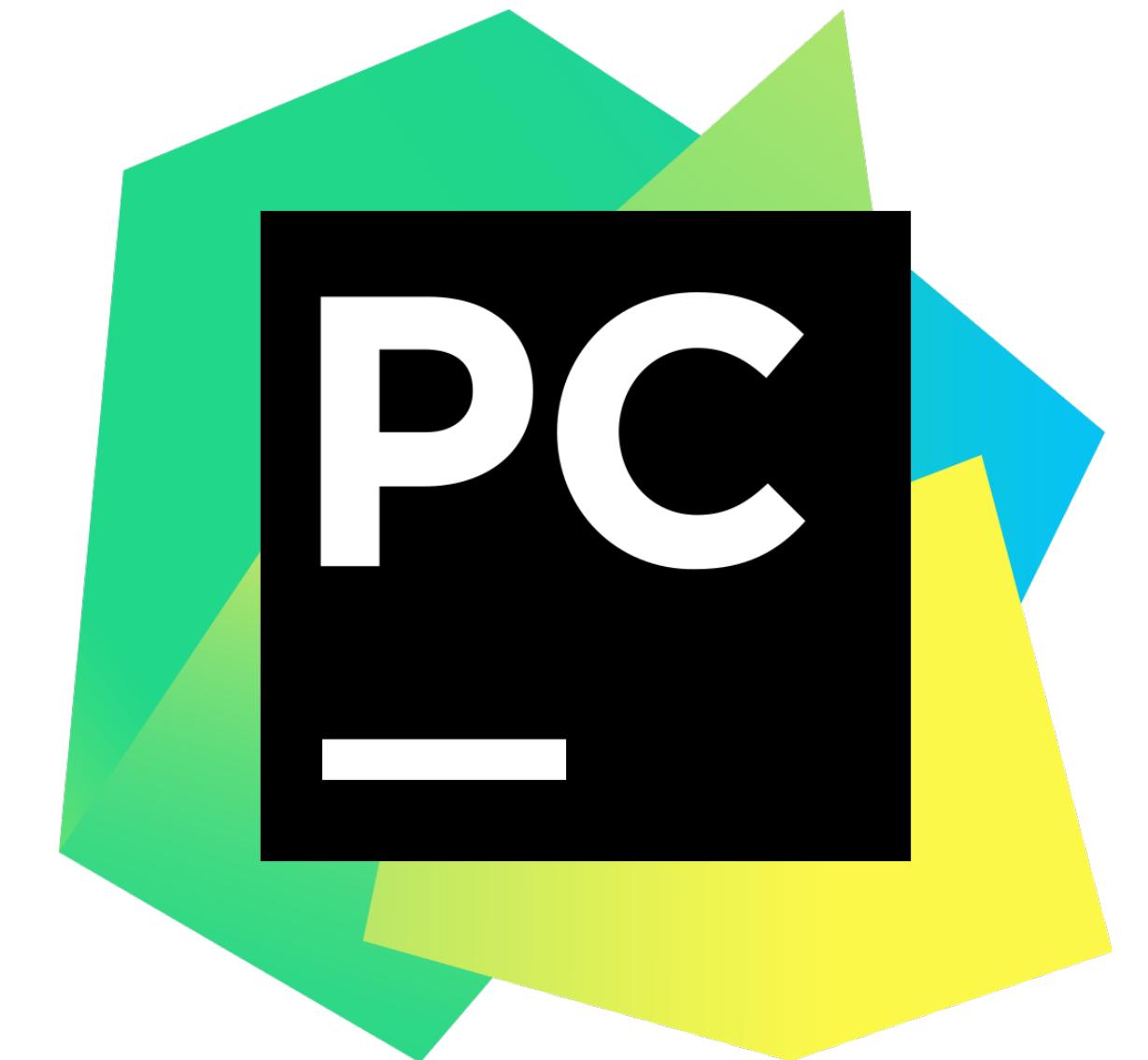
1. Apri PyCharm.
2. Clicca su **New Project**.
3. Scegli una directory (esempio: `reti_neurali`).
4. Configura un ambiente virtuale:
 1. Assicurati che sia selezionato "New Virtual Environment".
 2. Clicca su "Create".



Pycharm

Creazione del primo progetto

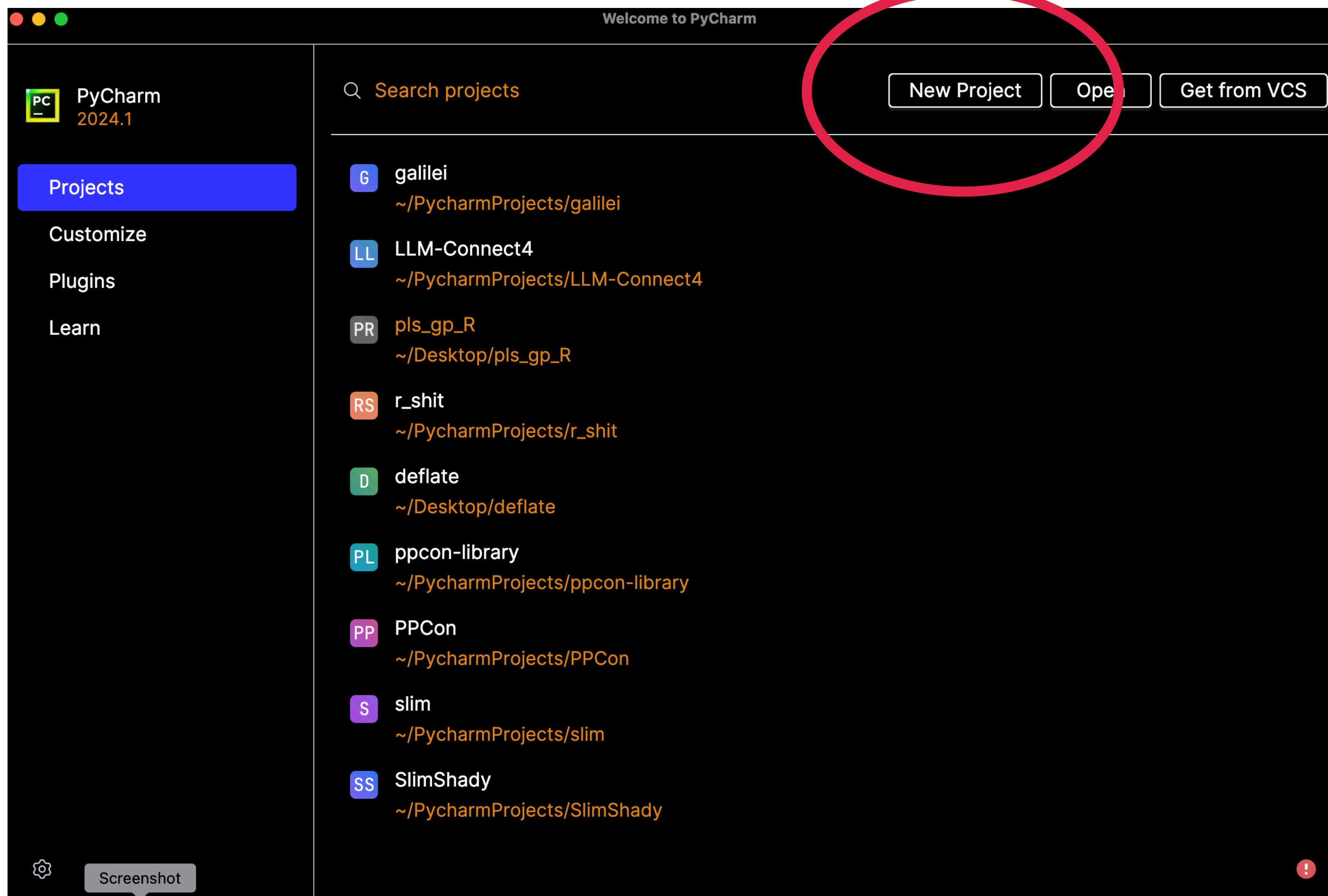
1. Apri PyCharm.
2. Clicca su **New Project**.
3. Scegli una directory (esempio: `reti_neurali`).
4. Configura un ambiente virtuale:
 1. Assicurati che sia selezionato "**New Virtual Environment**".
 2. Clicca su "Create".



Ambiente dove installeremo
tutte le librerie che servono a
compilare il nostro progetto

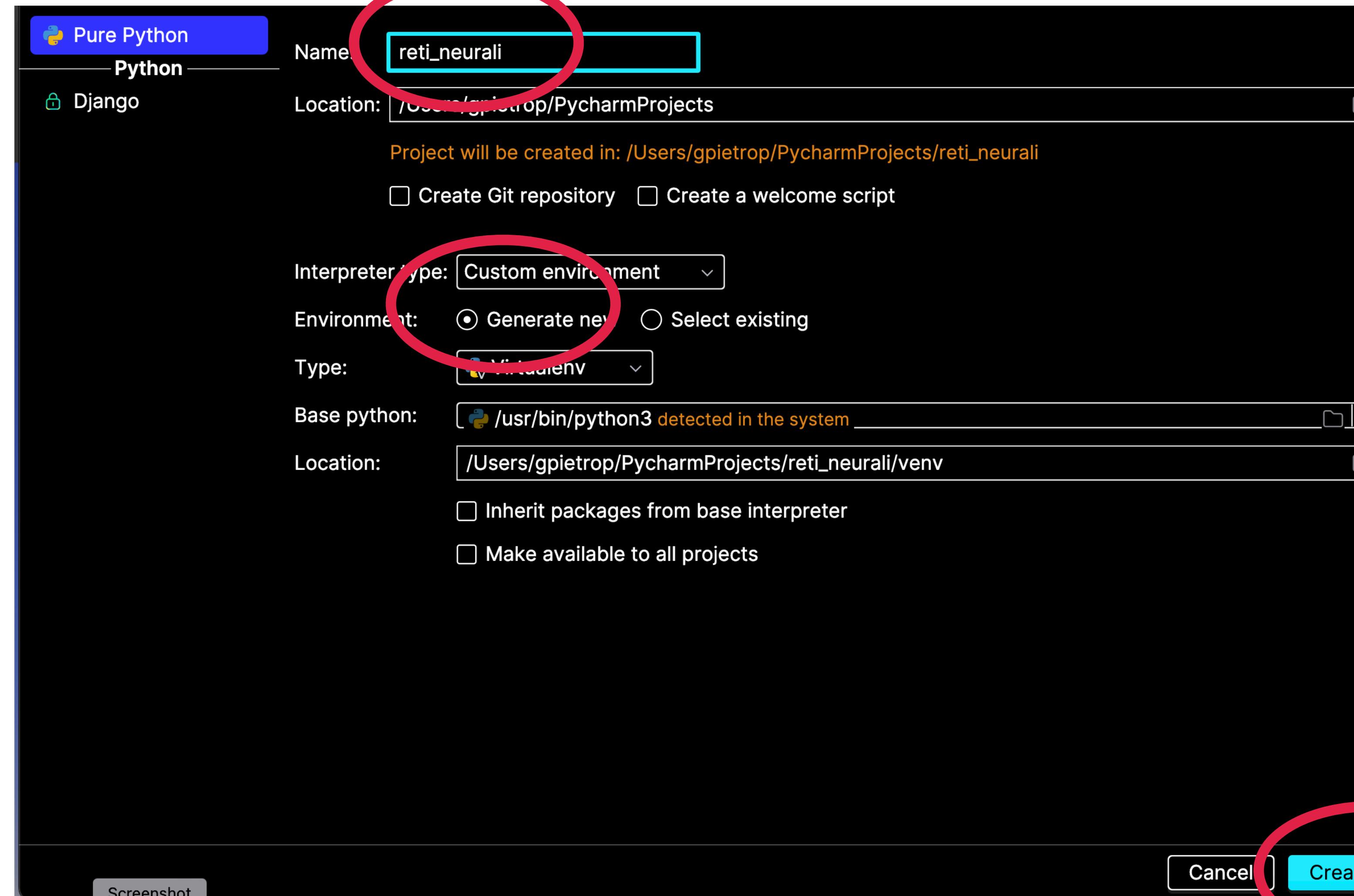
Pycharm

Creazione del primo progetto



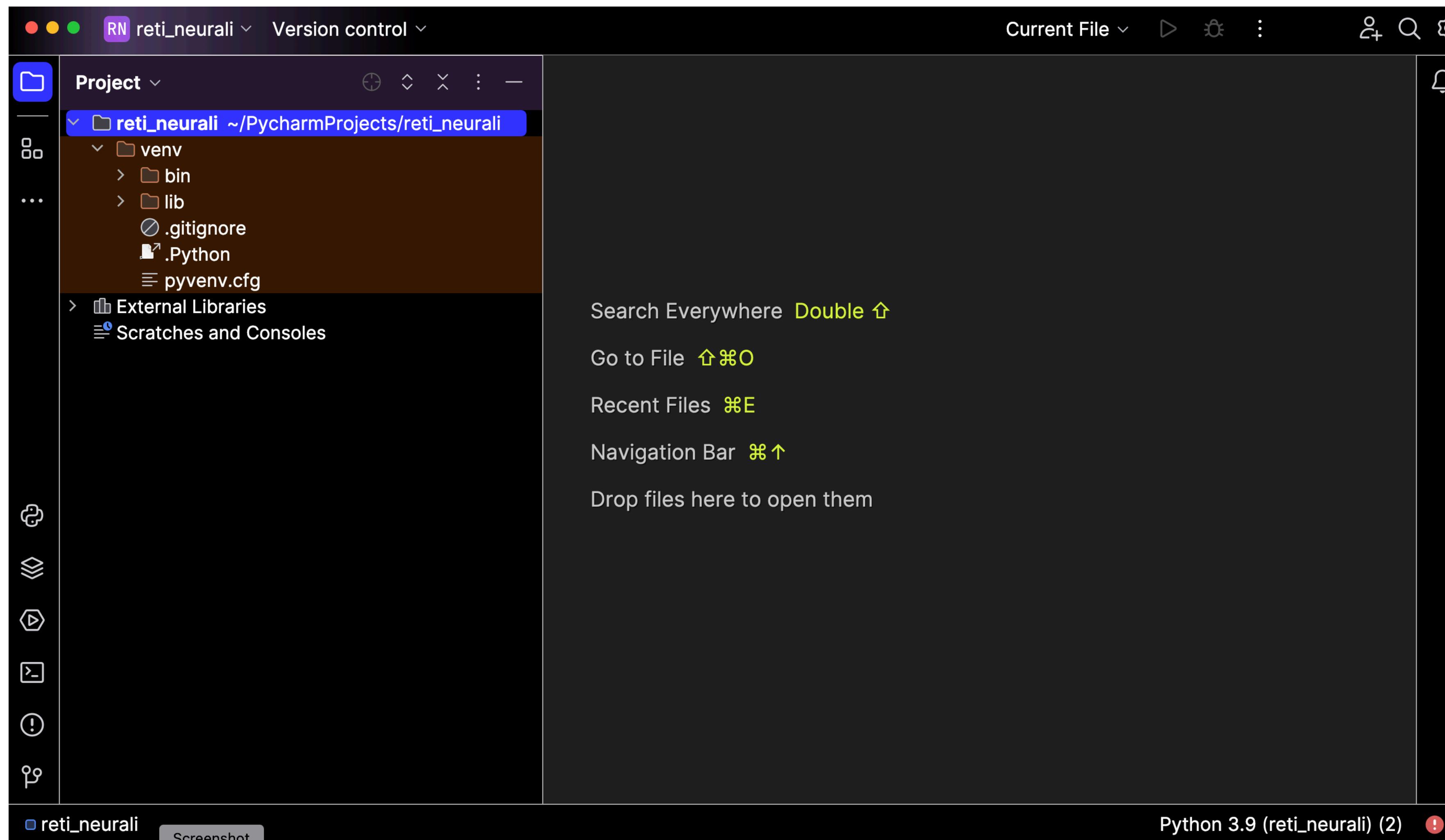
Pycharm

Creazione del primo progetto



Pycharm

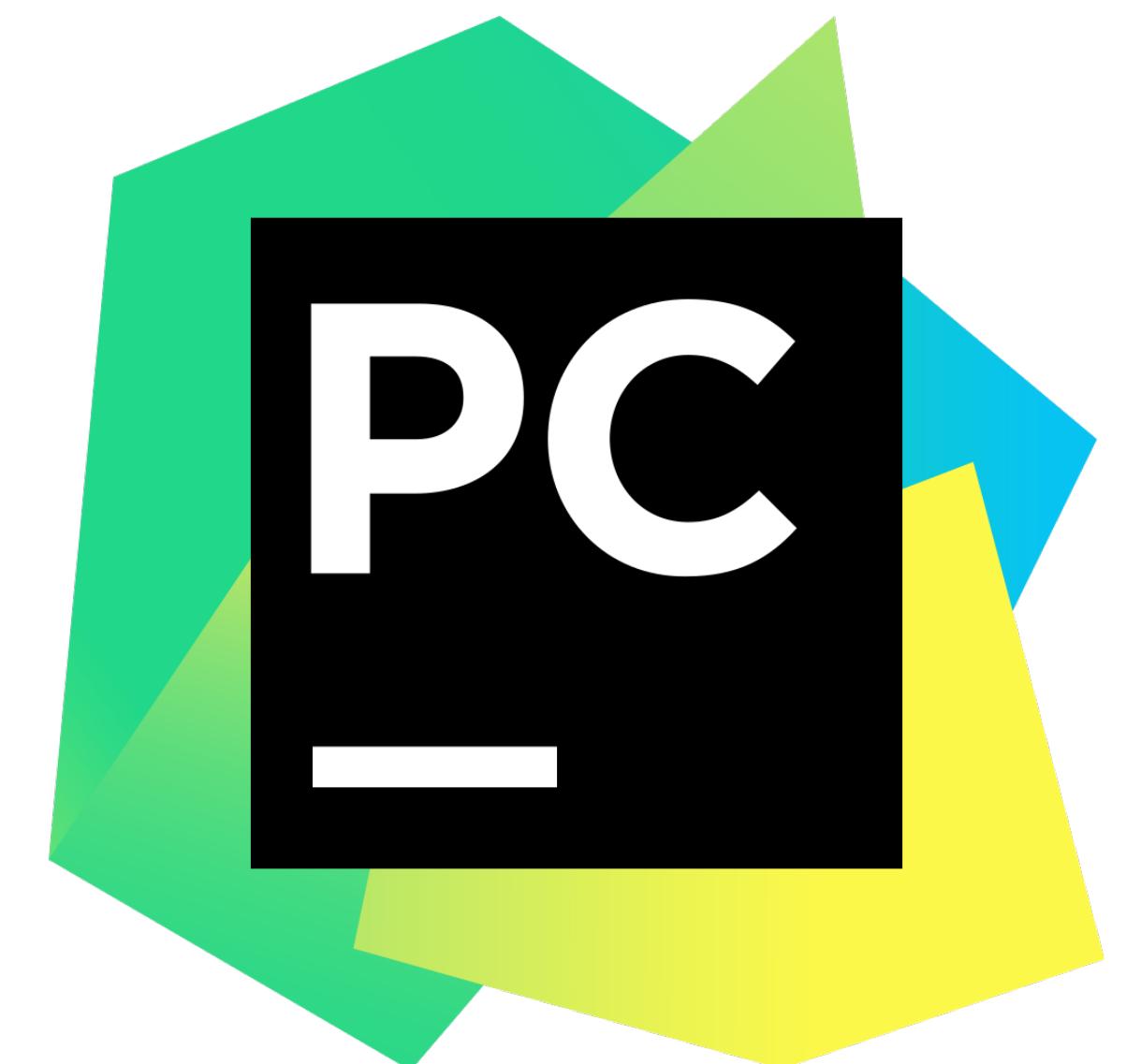
Creazione del primo progetto



Pycharm

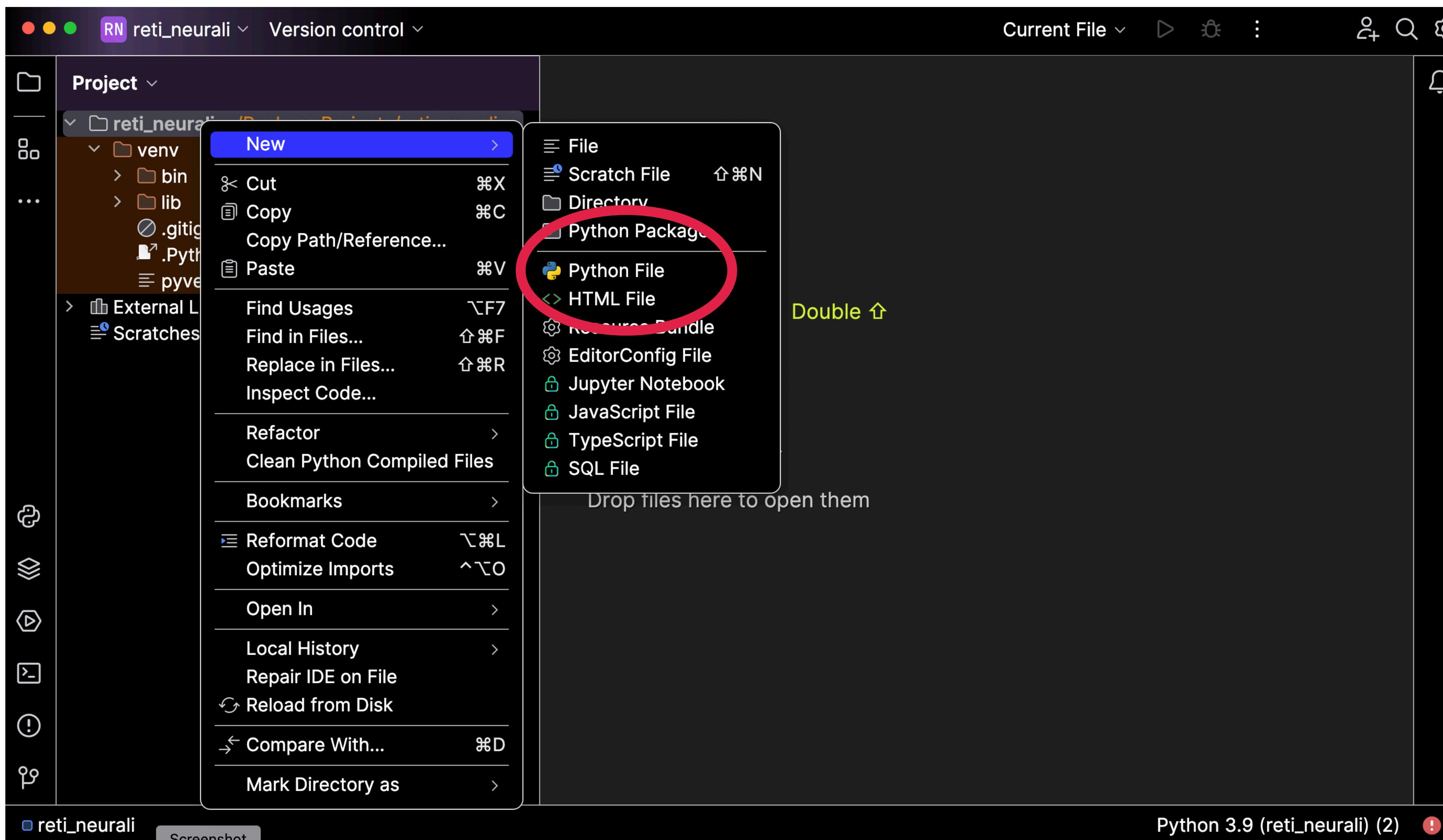
Creazione di un file python

1. Nella finestra del progetto, clicca con il tasto destro sulla cartella principale.
2. Seleziona New > Python File.
3. Dai un nome al file (esempio: `hello_world.py`).



Pycharm

Creazione di un file python



Pycharm

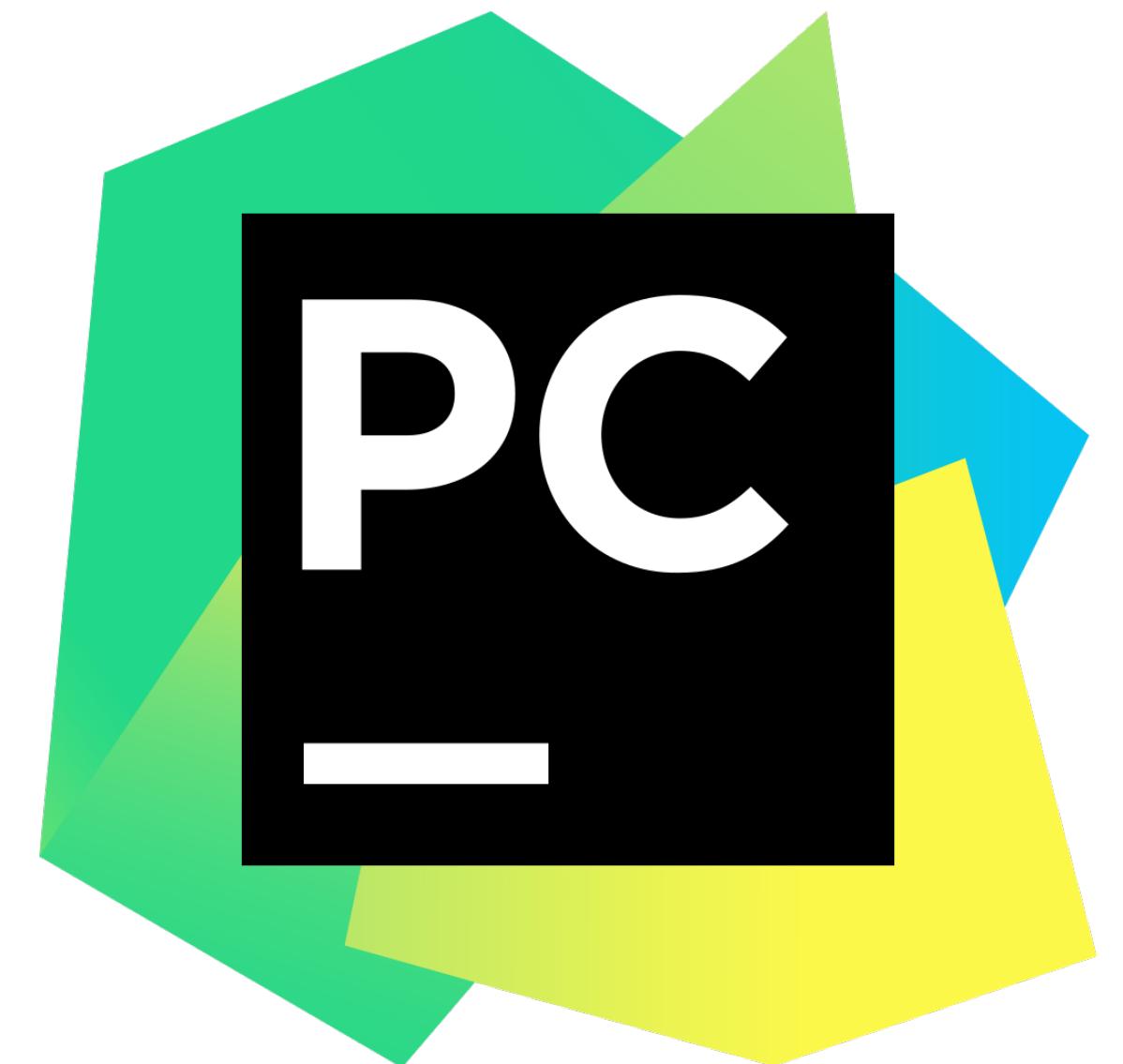
Il primo programma python

Scrivi il seguente codice nel file `hello_world.py`:

```
# Primo programma python  
print("Ciao studenti, benvenuti in PyCharm! ")
```

Per eseguirlo:

1. Clicca con il tasto destro sul file.
2. Seleziona Run 'hello_world'.



Pycharm

Creazione di un file python

The screenshot shows the PyCharm IDE interface. On the left is the Project tool window displaying a project named 'reti_neurali'. Inside the project, there is a 'venv' folder containing 'bin' and 'lib', along with files '.gitignore', '.Python', and 'pyvenv.cfg'. A file named 'hello_world.py' is selected in the list. The main editor window on the right contains the following Python code:

```
# Primo programma Python
print("Ciao studenti! Benvenuti in PyCharm!")
```

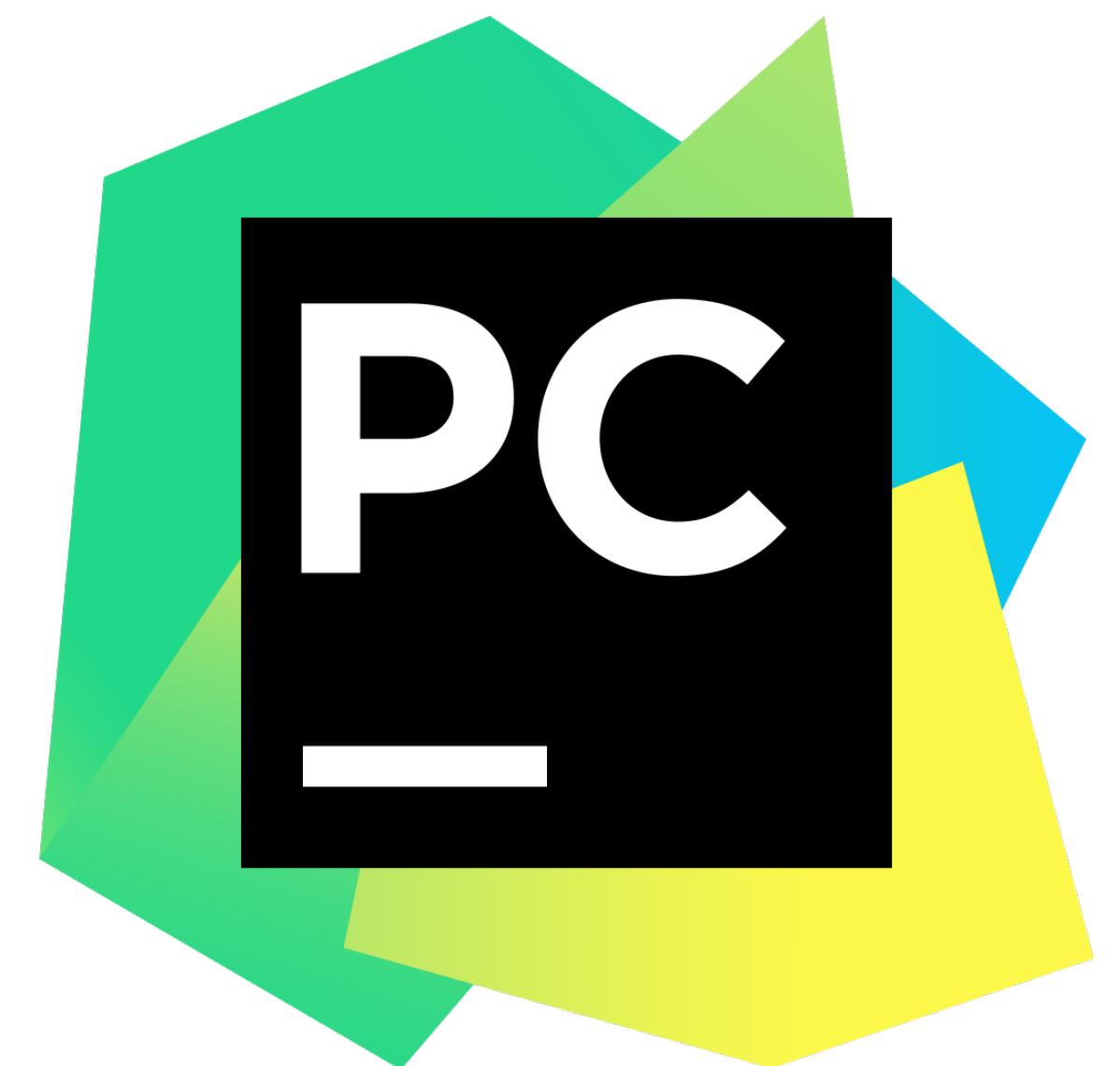
The status bar at the bottom indicates the file is named 'hello_world.py', the encoding is 'UTF-8', and the Python version is 'Python 3.9 (reti_neurali) (2)'. A red circle highlights the green run icon in the top right toolbar.

Con questo tasto possiamo
runnare il codice!

Pycharm

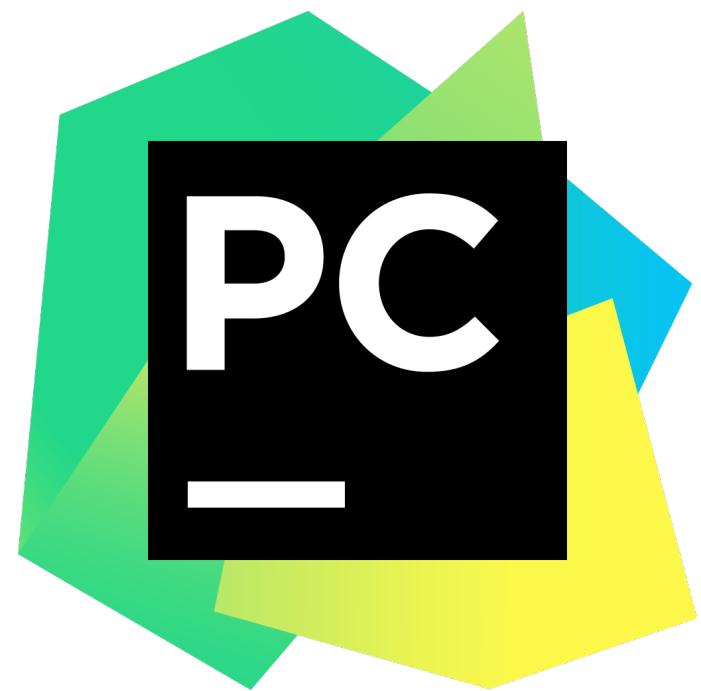
Esercizio Pratico

1. Crea un nuovo file chiamato `math_operations.py`.
2. Scrivi un programma che calcola la somma di due numeri:
3. Esegui il programma.



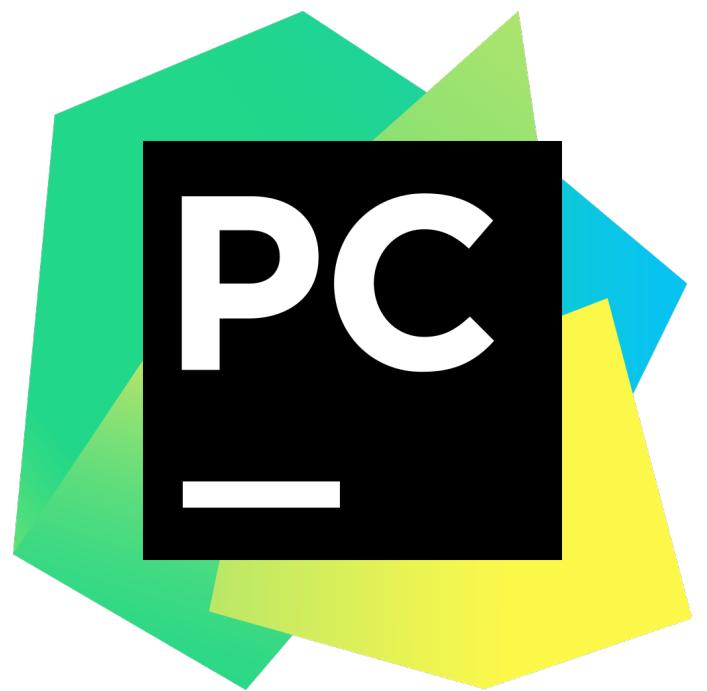


Ora sappiamo dove programmare

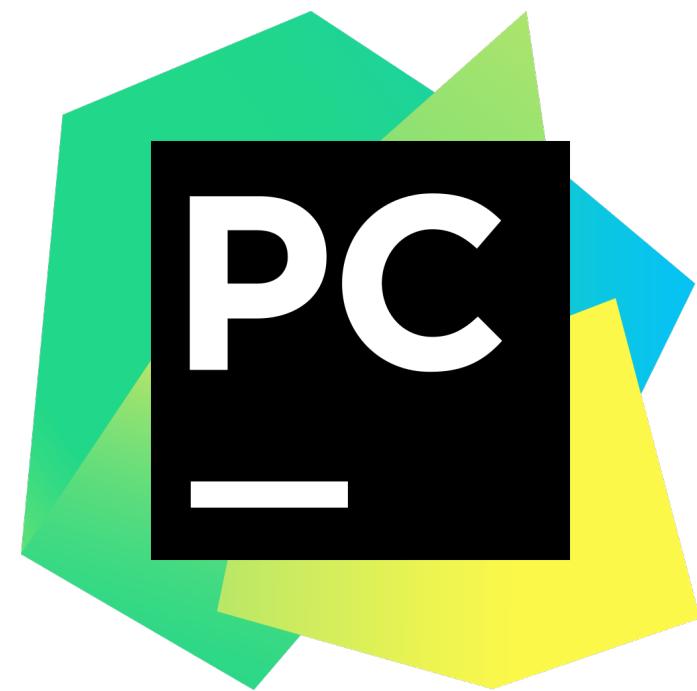




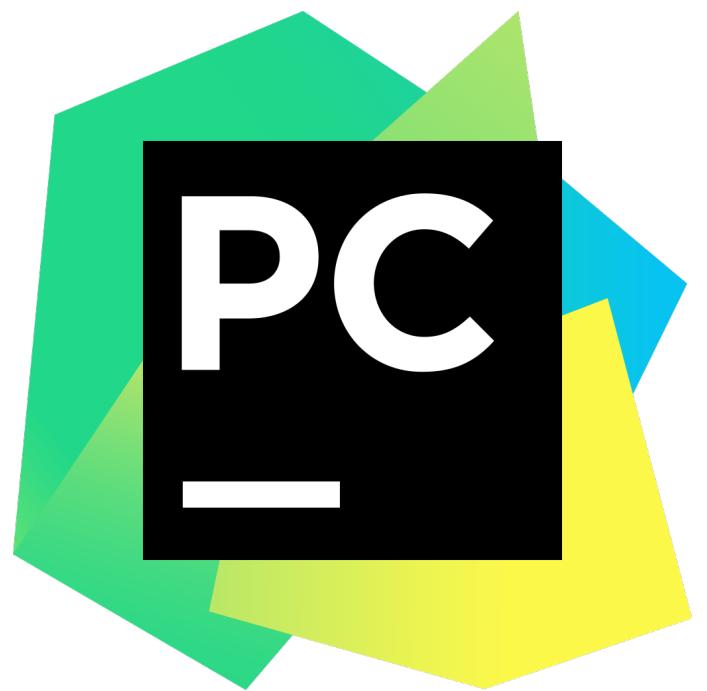
Ora sappiamo dove programmare



**Ma come facciamo se vogliamo usare il codice
python fornito durante le lezioni o condividere il
codice con i nostri amici?**



Ora sappiamo dove programmare



**Ma come facciamo se vogliamo usare il codice
python fornito durante le lezioni o condividere il
codice con i nostri amici?**

GitHub

GitHub

Cosa è Git?

Cos'è Git?

- Git è un sistema di controllo di versione.
- Ti permette di:
 - Tenere traccia delle modifiche al tuo codice.
 - Collaborare con altre persone.



GitHub

Cosa è Git?

Cos'è Git?

- Git è un sistema di controllo di versione.
- Ti permette di:
 - Tenere traccia delle modifiche al tuo codice.
 - Collaborare con altre persone.

Cos'è GitHub?

- GitHub è una piattaforma online per condividere codice usando Git.
- Funziona come un archivio dei tuoi progetti.
- Noi useremo GitHub per:
 - Scaricare i file .py del progetto.
 - Caricare i vostri file .py una volta completati.



GitHub

Cosa è Git?

The screenshot shows a GitHub profile for the user `gpietrop`. The profile features a large circular profile picture of a person standing on a white, layered rock formation. The user's name, **Gloria Pietropolli**, and handle, `gpietrop`, are displayed. Below the profile picture, it says "Researcher at the University of Trieste". A "Edit profile" button is visible.

The top navigation bar includes links for Overview, Repositories (52), Projects, Packages, and Stars (4). The search bar contains the placeholder "Type / to search".

The main content area is titled "Pinned" and displays five repositories:

- PreCorso-INF-unITS** (Public) - Directory con le slides utilizzate per il Precorso di Informatica @UniTS. Forked from DALabNOVA/slimg. 6 stars, 1 fork.
- slim** (Public) - Forked from DALabNOVA/slimg. slim (Semantic Learning algorithm based on inflate and Deflate Mutation) python library. Python.
- GSGP-GD** (Public) - Code for paper "Combining Geometric Semantic GP with Gradient-descent Optimization" - EuroGP 2022. Julia, 1 star.
- LLM-Connect4** (Public) - Code for paper "Large Language Model-based Test Case Generation for GP Agents" - GECCO 2024. Python.
- CAbEA** (Public) - Code for the paper "The Role of the Substrate in CA-based Evolutionary Algorithms" - GECCO 2024.
- ppcon** (Public) - ppcon: biogeochemical argo profile prediction with 1D convolutional networks. Python, 1 star.

A "Customize your pins" link is located in the top right corner of the pinned section.

Below the pinned section, there is a "302 contributions in the last year" chart showing activity over time. The chart uses a color scale where darker shades represent more frequent contributions. The x-axis shows months from Dec to Nov, and the y-axis shows days of the week (Mon, Wed, Fri). The chart indicates significant activity in January, February, May, June, July, August, September, October, and November, with a notable peak in late October.

At the bottom left, there is a "Screenshot /about-me" link. At the bottom right, there is a "Contribution settings" dropdown menu with a "2024" button.

GitHub

Accedere a una repository GitHub

Accedere a una repository su GitHub

1. Apri il tuo browser e vai su <https://github.com>.
2. Crea un account (se non ne hai già uno).
3. Una volta loggati, cerca la repository del progetto:
 - Esempio: <https://github.com/gpietrop/intro-mlp>.

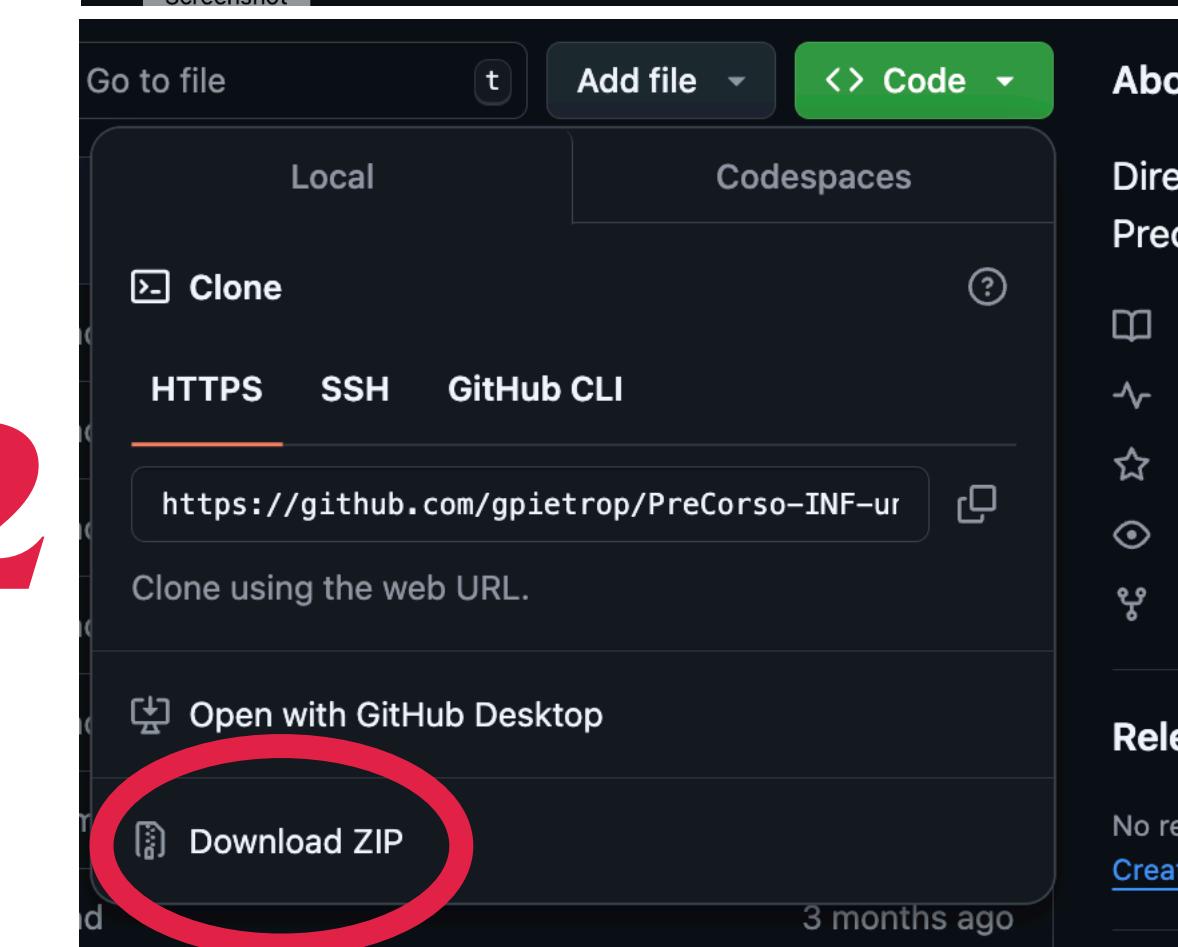
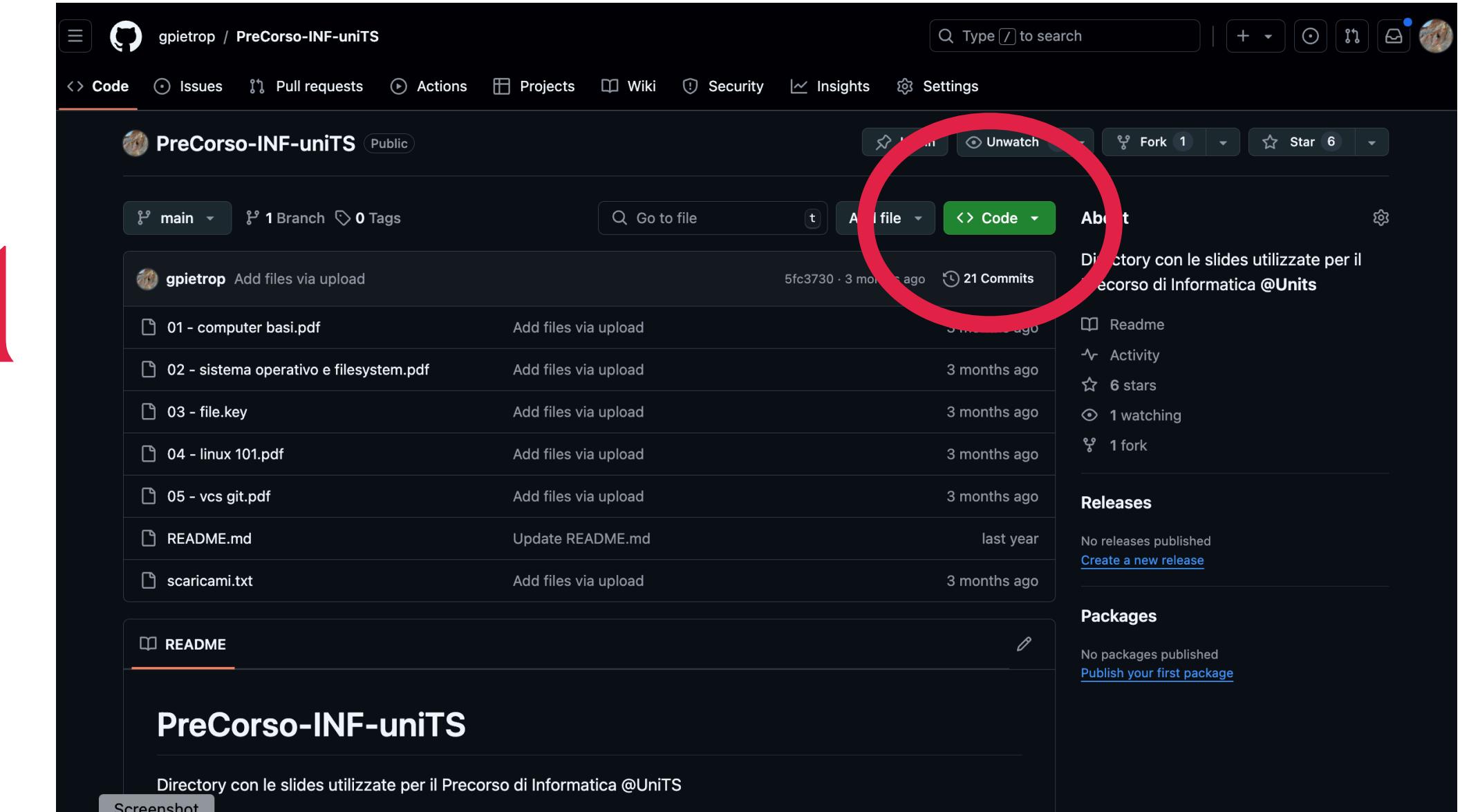


GitHub

Scaricare una Repository

Scaricare una repository

1. Vai sulla pagina della repository.
2. Clicca sul pulsante verde **Code**.
3. Seleziona **Download ZIP**.
4. Estrai il file ZIP sul tuo computer.



GitHub

Caricare una Repository

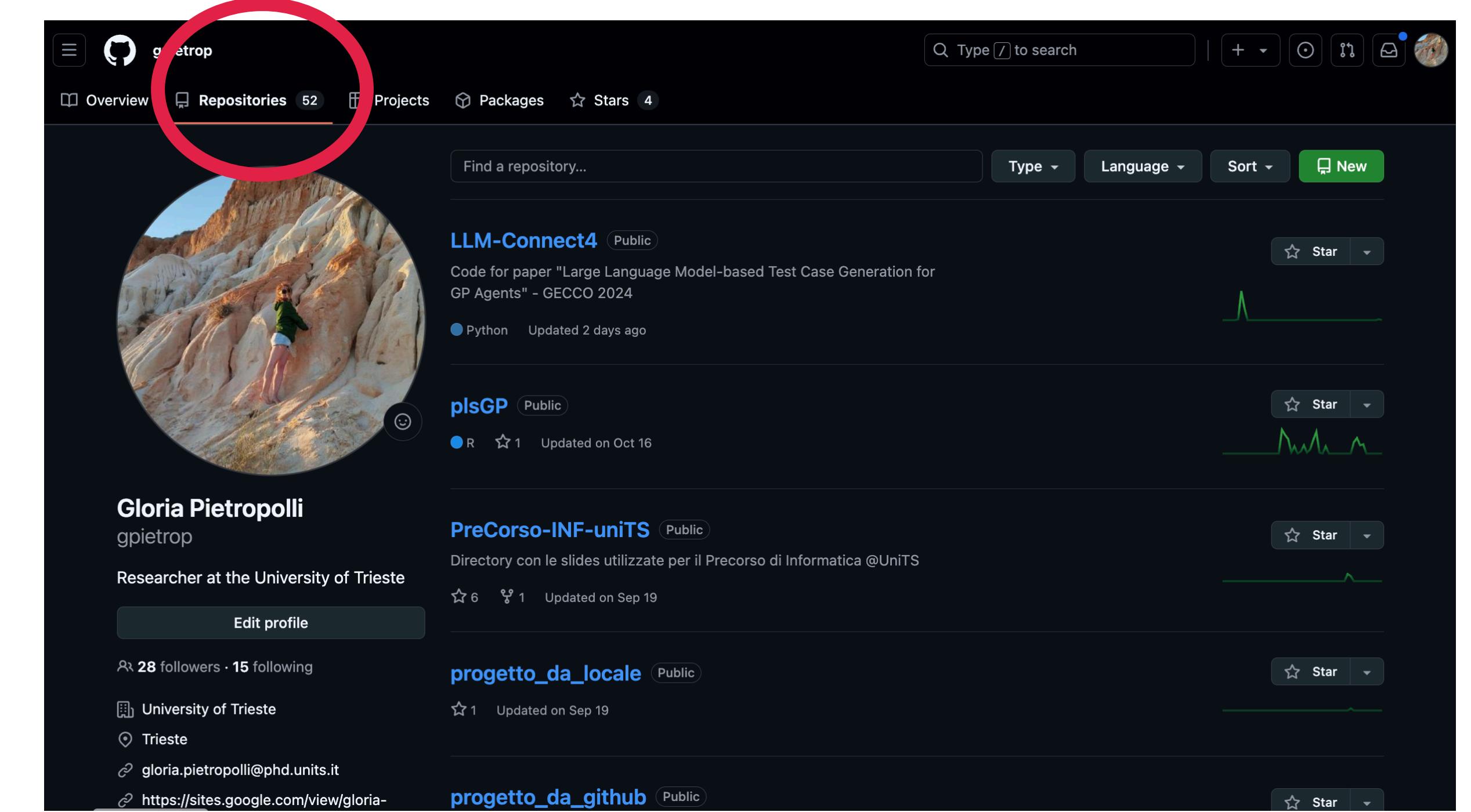
1. Entra nella finestra Repositories

2.

3.

4.

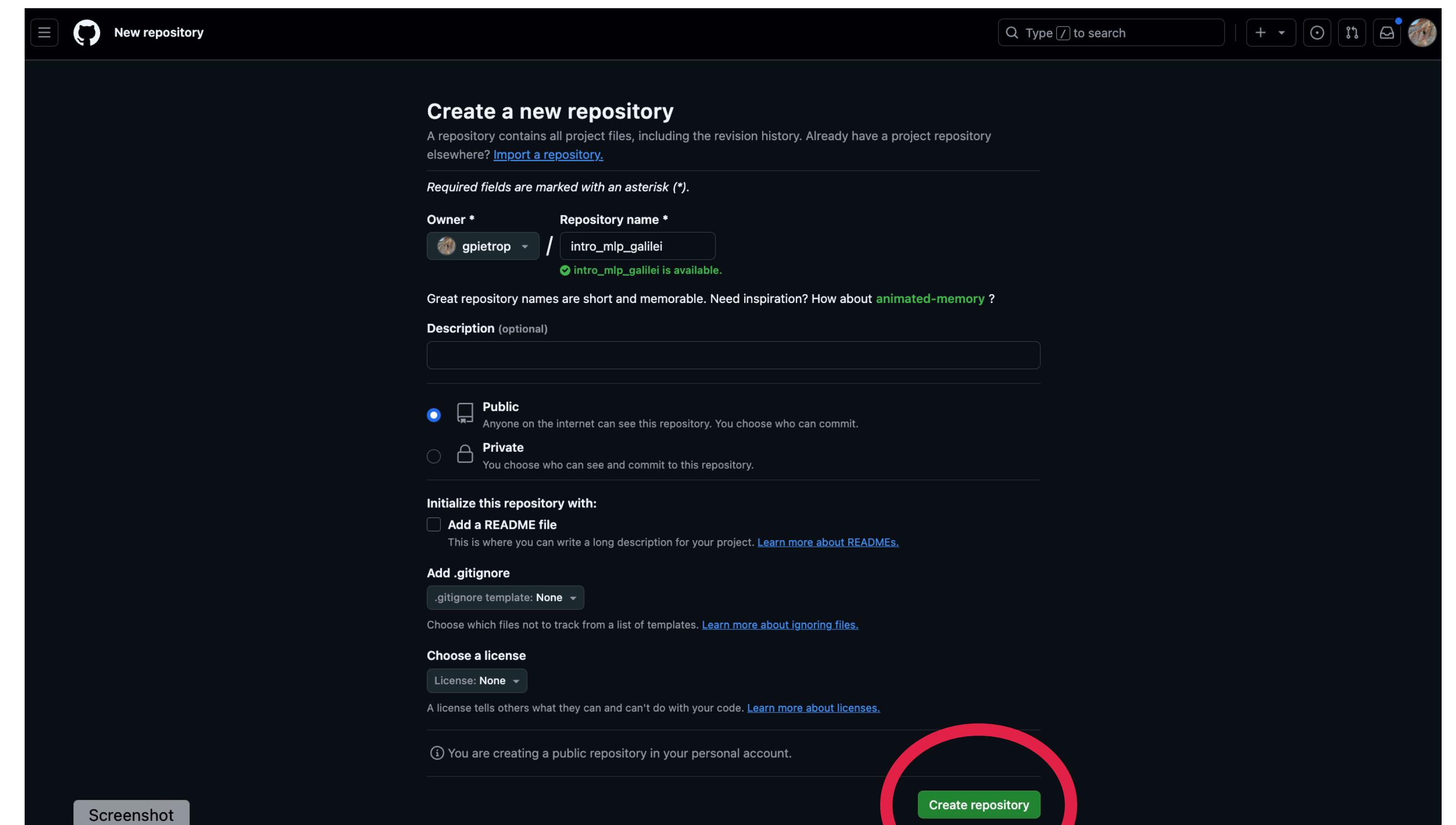
5.



GitHub

Caricare una Repository

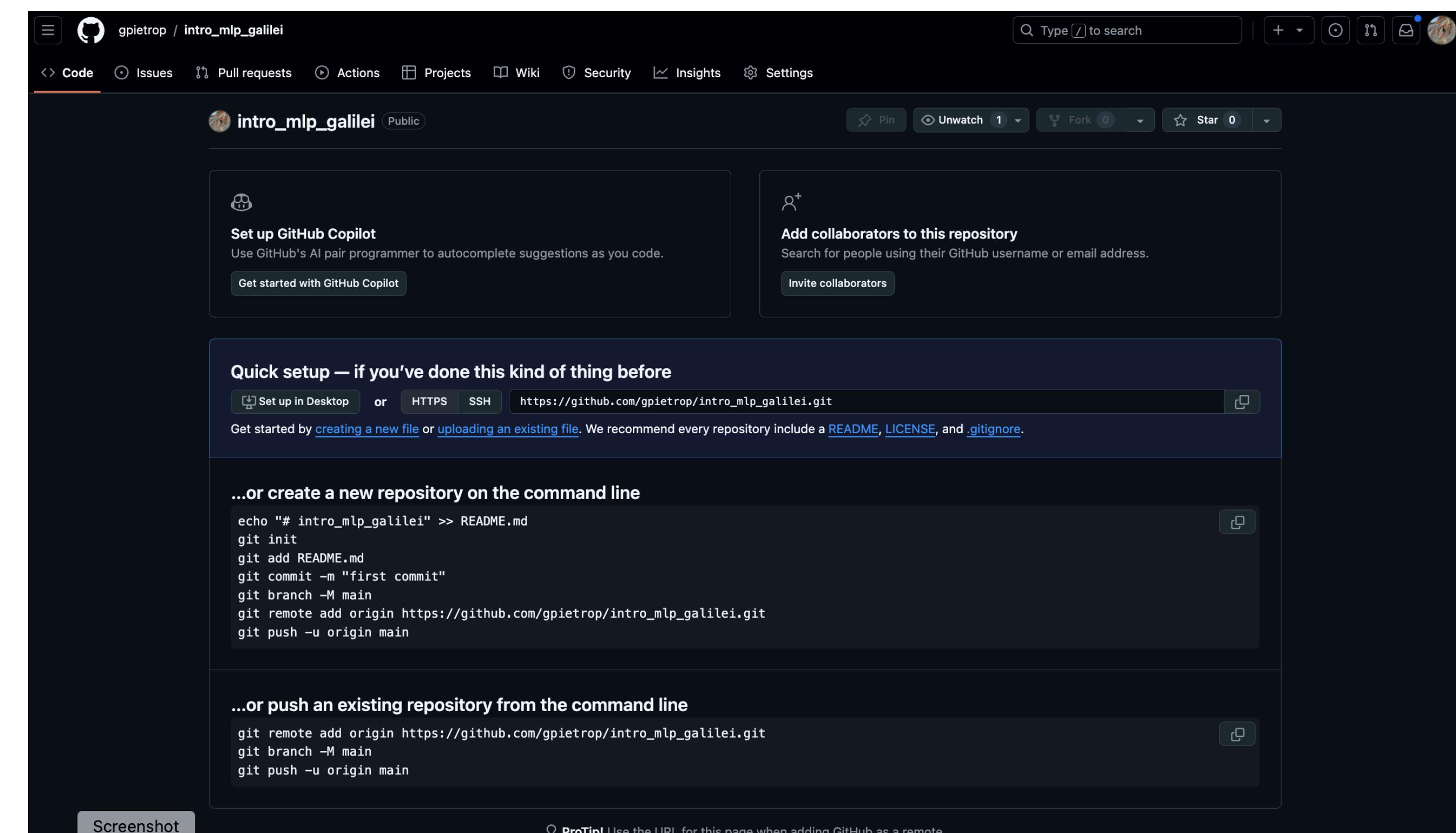
1. Entra nella finestra Repositories
2. Clicca new e crea una nuova repository
- 3.
- 4.
- 5.



GitHub

Caricare una Repository

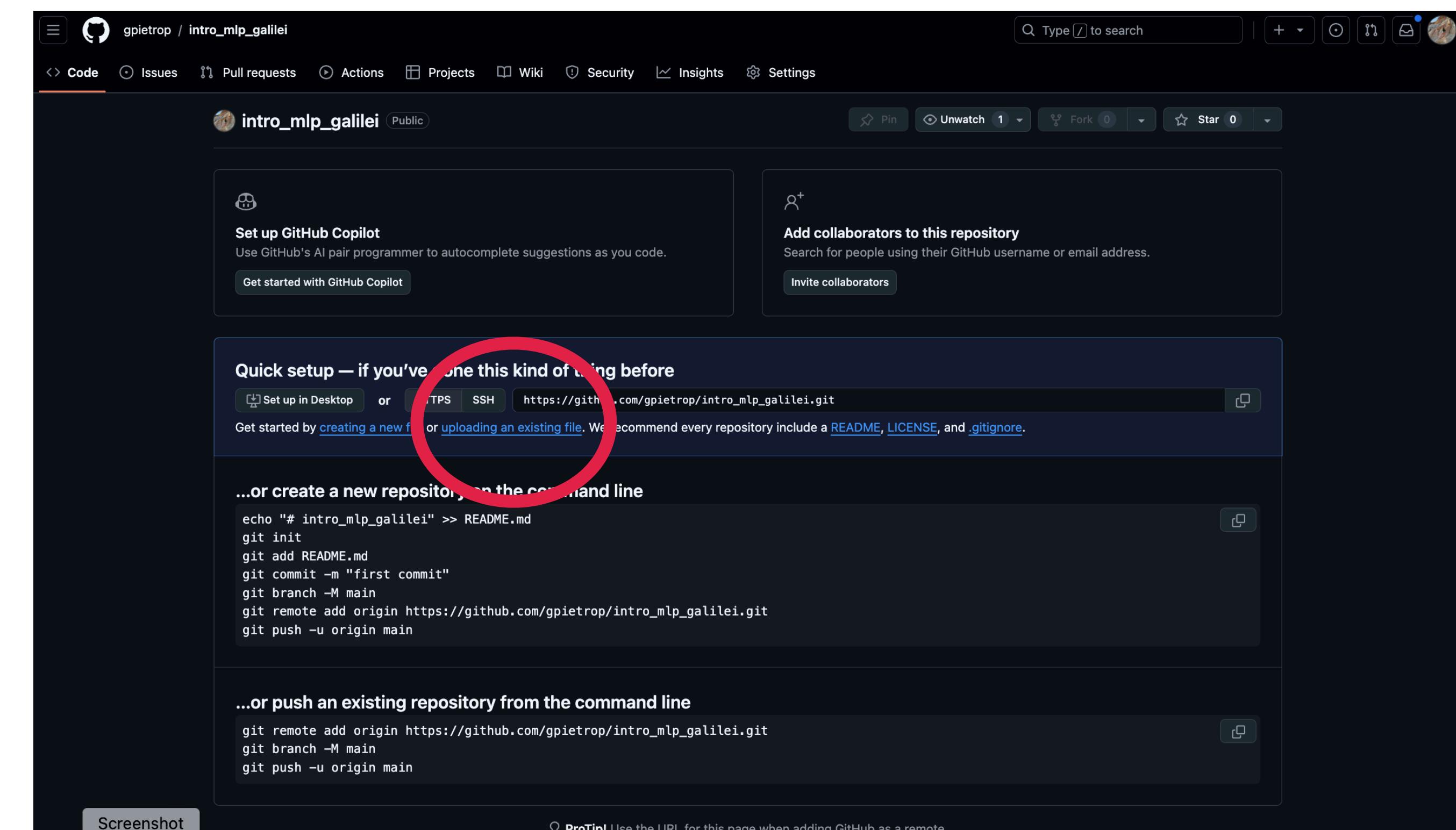
1. Entra nella finestra Repositories
2. Clicca new e crea una nuova repository
- 3.
- 4.
- 5.



GitHub

Caricare una Repository

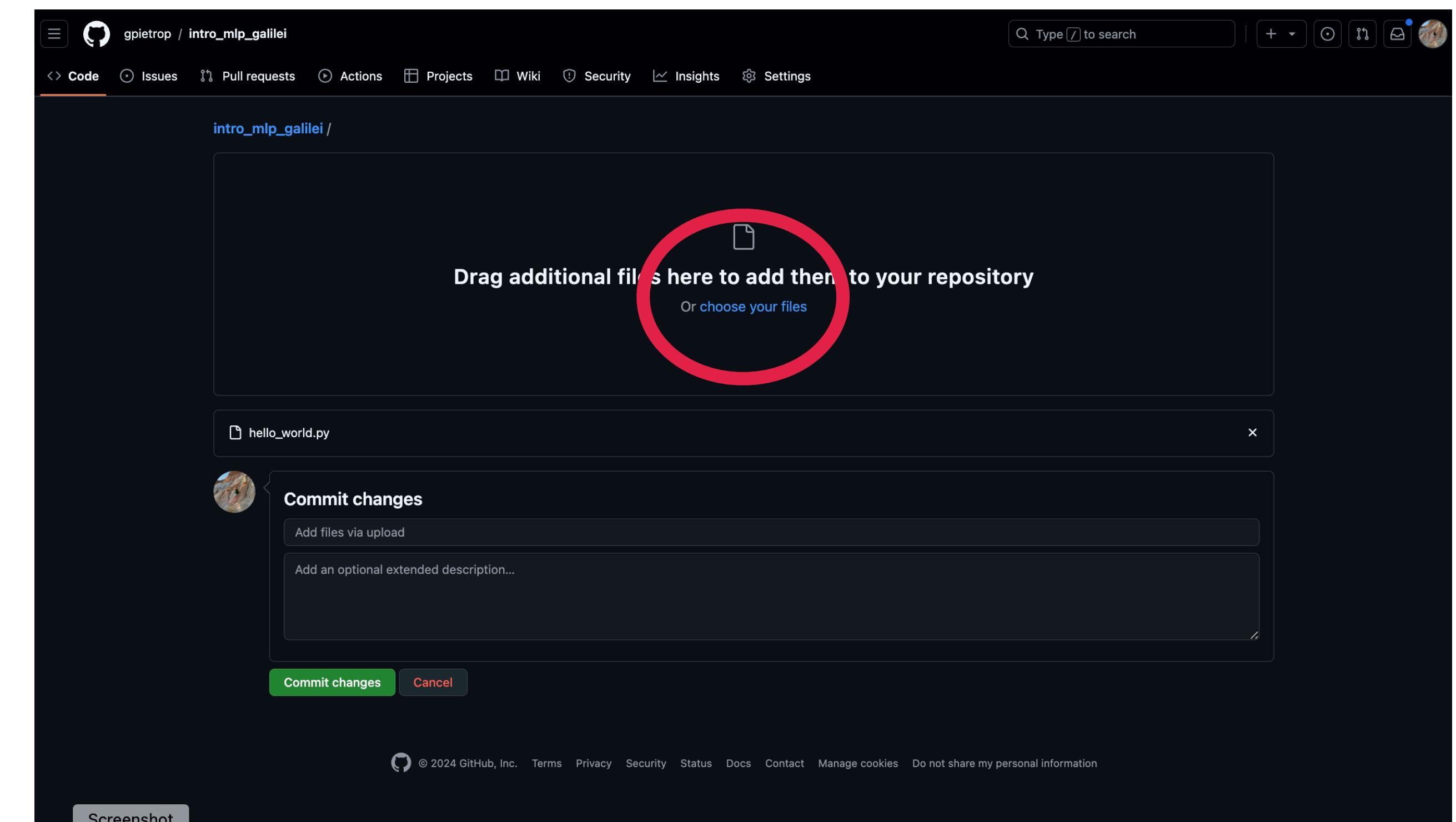
1. Entra nella finestra Repositories
2. Clicca **new** e crea una nuova repository
3. Clicca sul pulsante **Add file** e poi su **Upload files**.
- 4.
- 5.



GitHub

Caricare una Repository

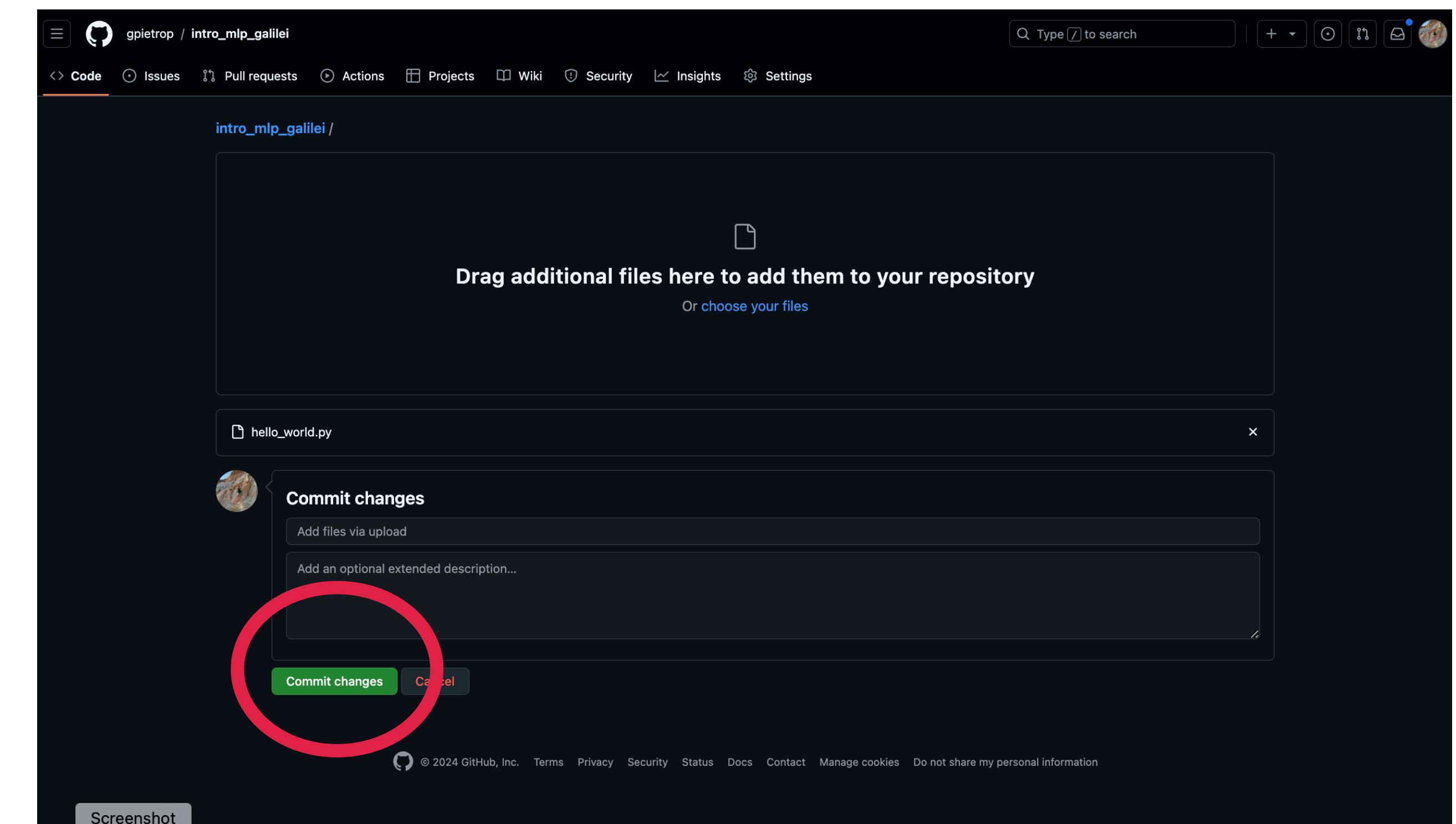
1. Entra nella finestra **Repositories**
2. Clicca **new** e crea una nuova repository
3. Clicca sul pulsante **Add file** e poi su **Upload files**.
4. Trascina i file .py nella finestra oppure clicca su **Choose your files**.
- 5.

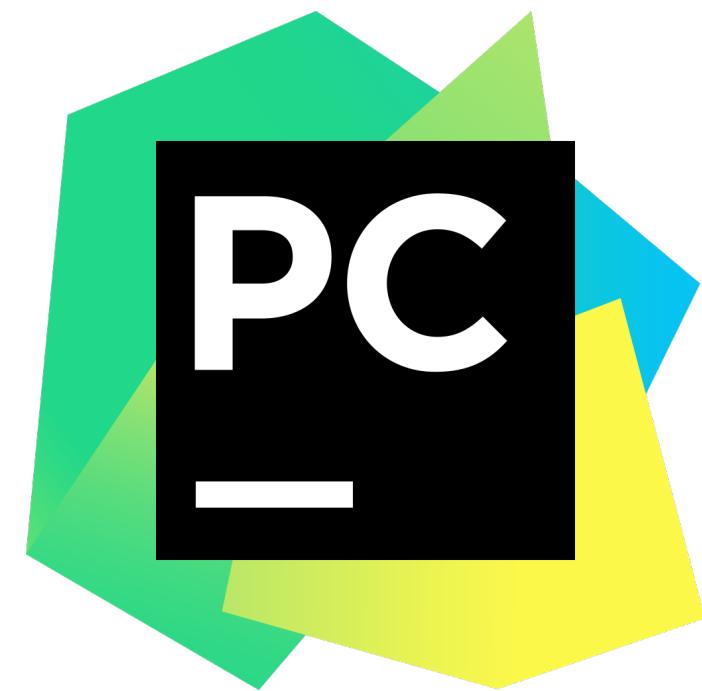


GitHub

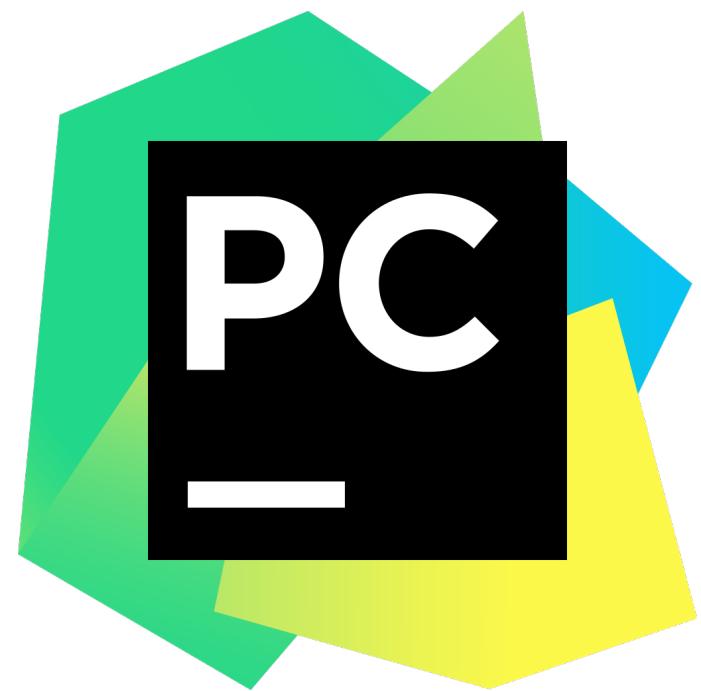
Caricare una Repository

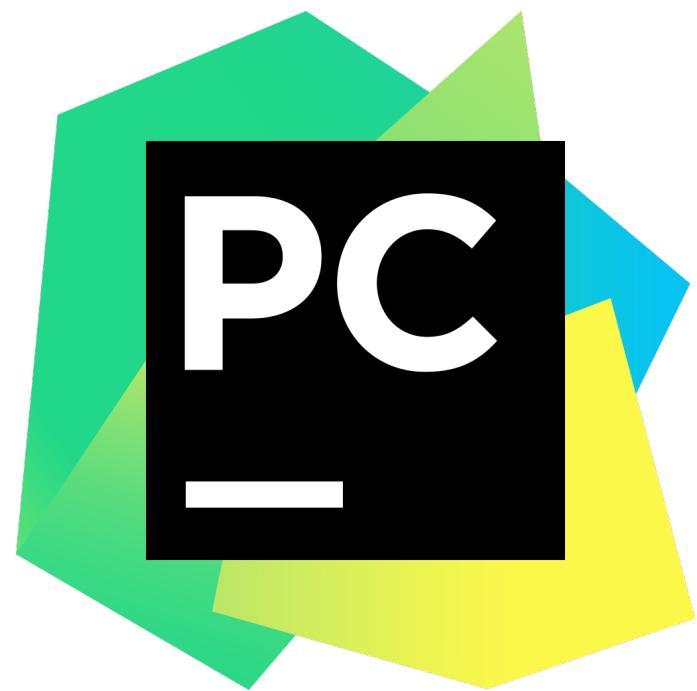
1. Entra nella finestra **Repositories**
2. Clicca **new** e crea una nuova repository
3. Clicca sul pulsante **Add file** e poi su **Upload files**.
4. Trascina i file .py nella finestra oppure clicca su **Choose your files**.
5. Clicca su **Commit changes** per salvare i cambiamenti eseguiti



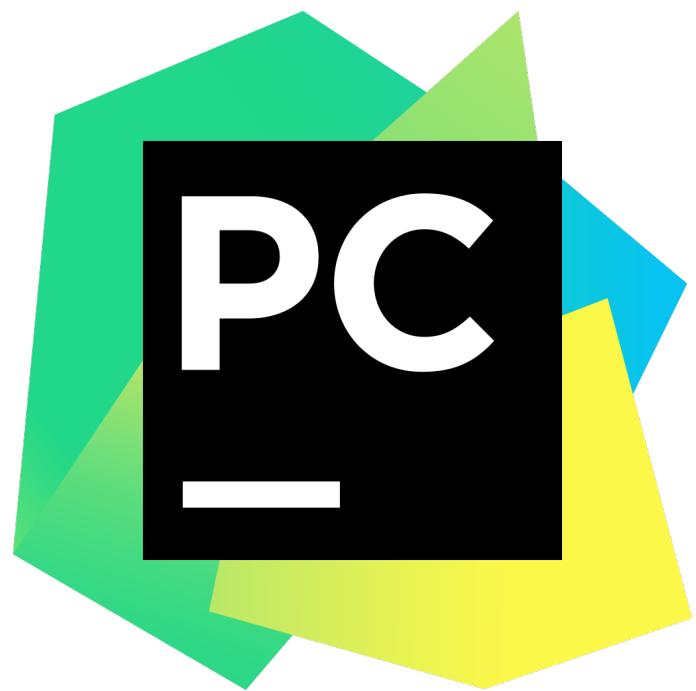


Ora sappiamo dove programmare



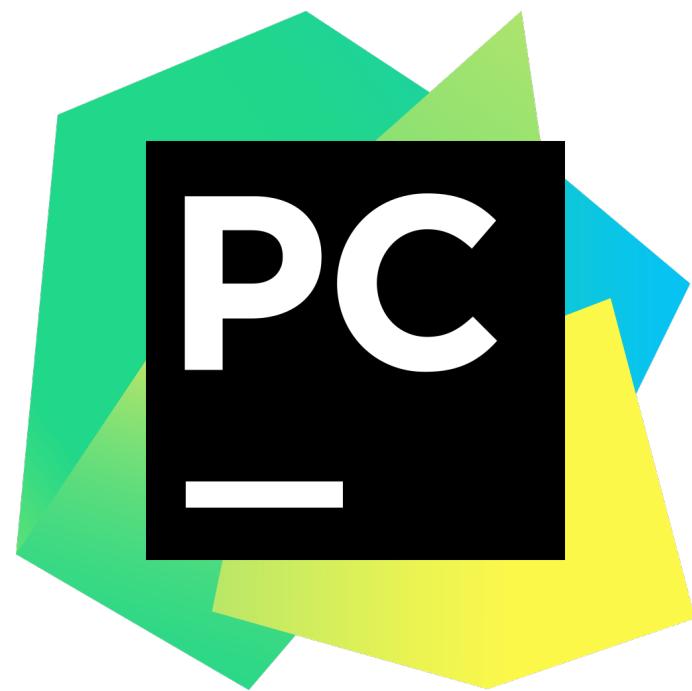


Ora sappiamo dove programmare



Ora sappiamo anche come
scaricare i file .py e come caricarli
se vogliamo condividerli





Ora sappiamo dove programmare

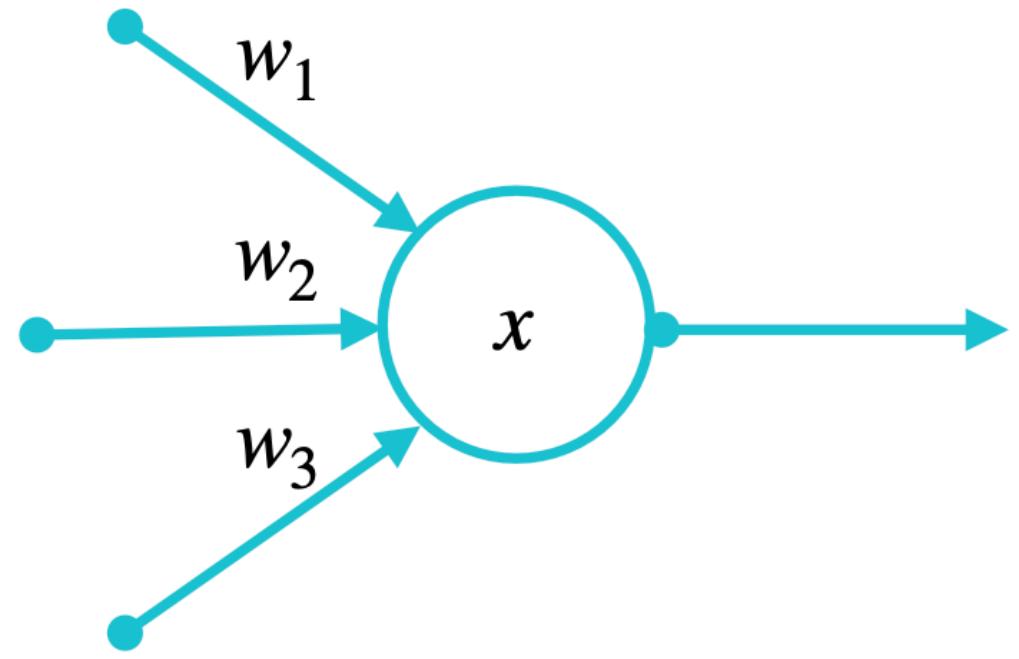
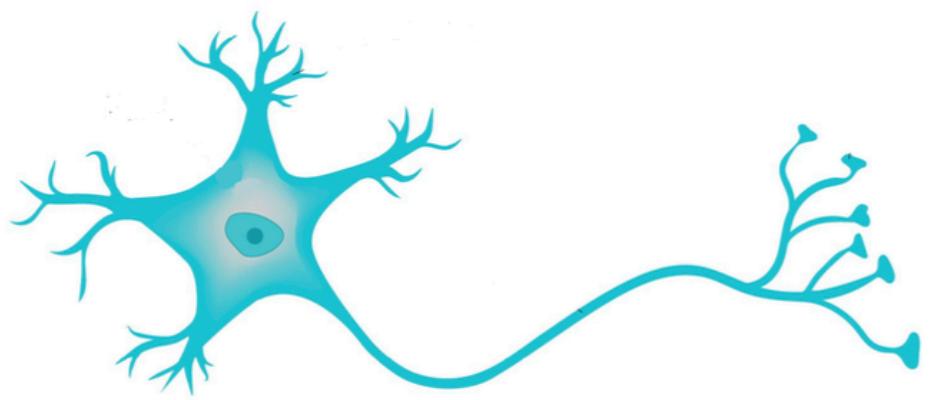


Ora sappiamo anche come
scaricare i file .py e come caricarli
se vogliamo condividerli



Non rimane che iniziare a programmare!

Dove eravamo rimasti?



Le Reti Neurali

Come funziona un singolo neurone?

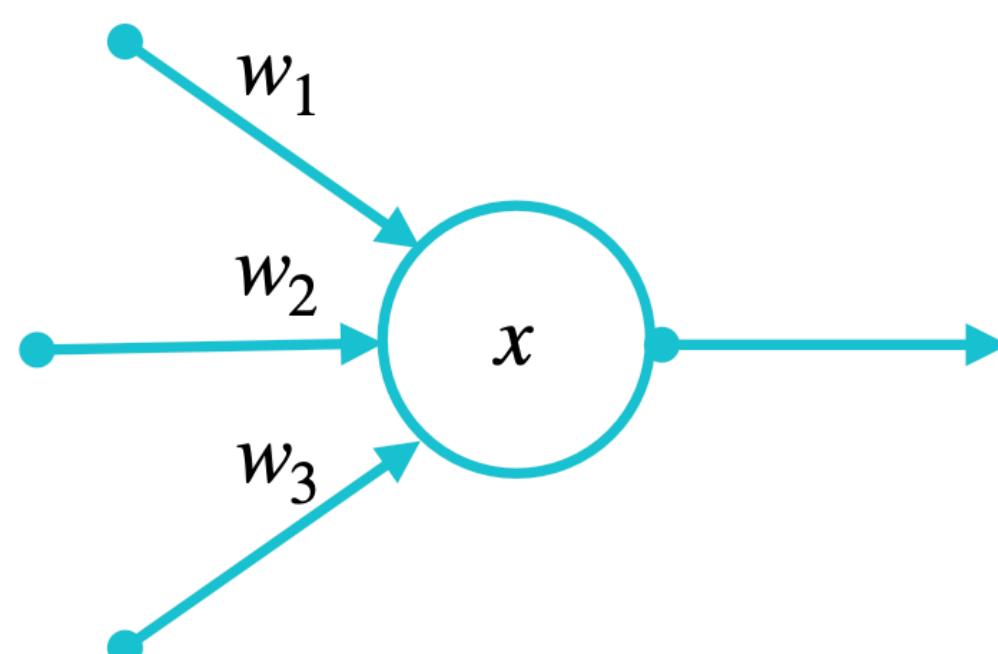
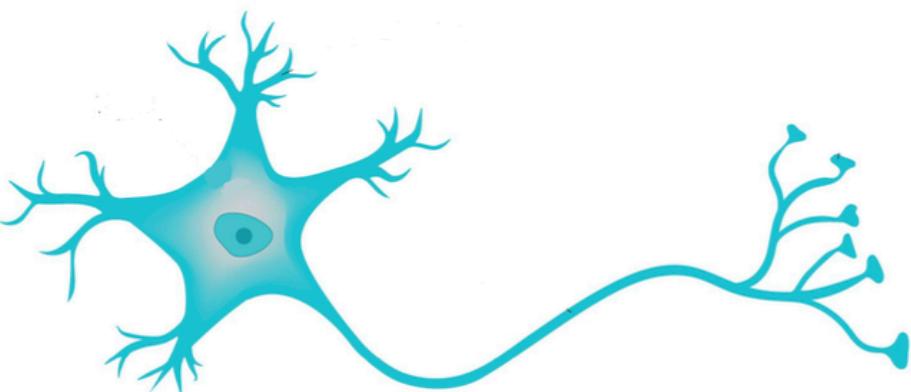
Ogni neurone:

- Riceve un input numerico.
- Applica una funzione matematica.
- Produce un output.

Esempio:

- Input: 2, Peso: 0.5, Bias: 1.
- Output = funzione
 $(2 \times 0.5 + 1)$

Dove eravamo rimasti?



Ogni neurone:

- Riceve un input numerico.
- Applica una funzione matematica.
- Produce un output.

Esempio:

- Input: 2, Peso: 0.5, Bias: 1.
- Output = funzione
 $(2 \times 0.5 + 1)$

Le Reti Neurali

Come funziona un singolo neurone?

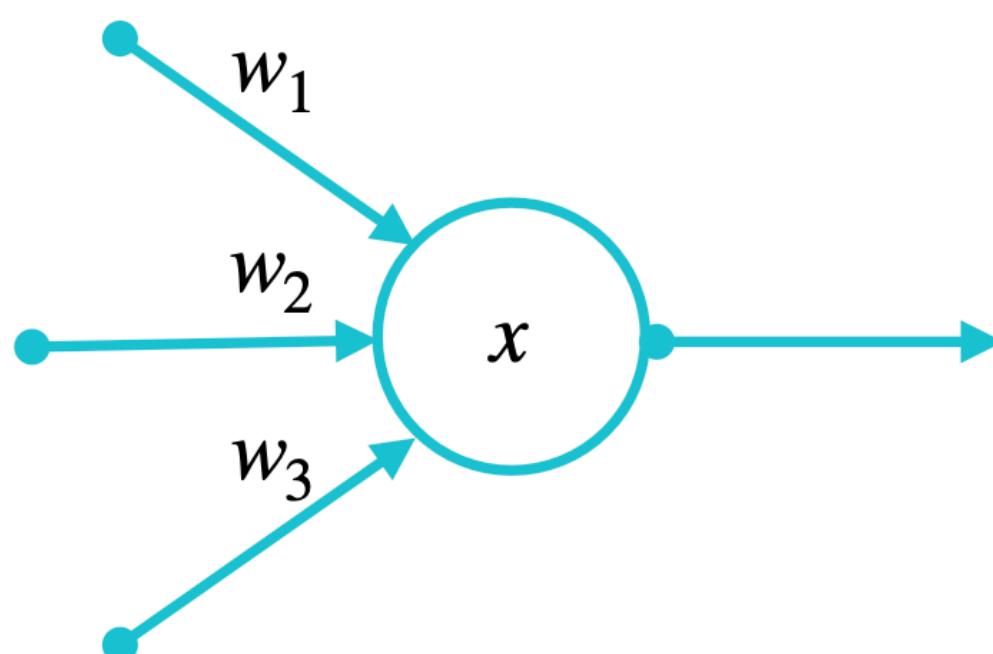
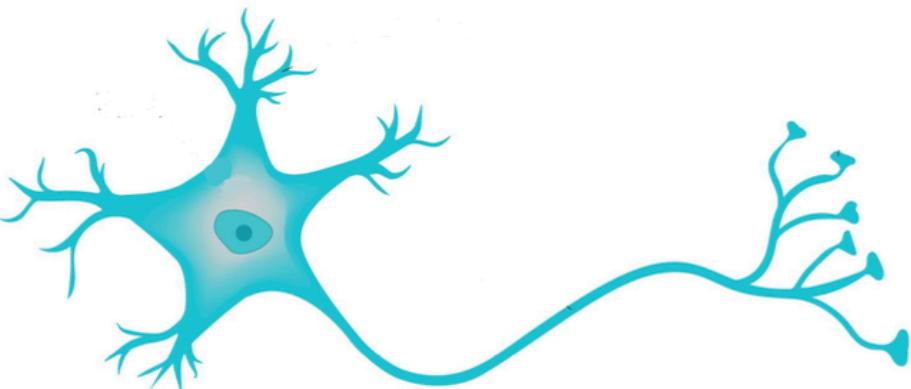
Proviamo a scrivere questa funzione in python usando Pycharm

Scriviamo una funzione
`simple_neuron(input_value, weight, bias):`

- **Input:** `input_value, weight, bias`
- **Output:** `input_value * weight + bias`

E testiamola!!

Dove eravamo rimasti?



Ogni neurone:

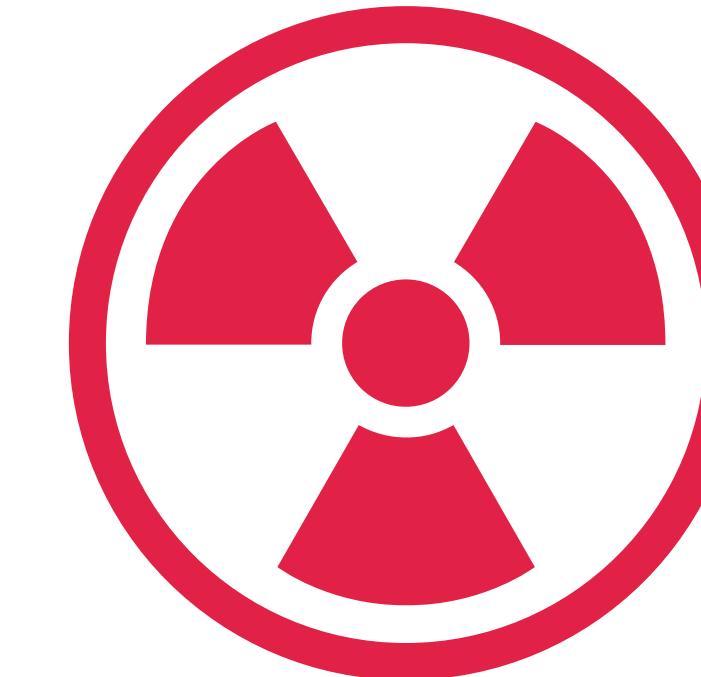
- Riceve un input numerico.
- Applica una funzione matematica.
- Produce un output.

Esempio:

- Input: 2, Peso: 0.5, Bias: 1.
- Output = funzione
 $(2 \times 0.5 + 1)$

Le Reti Neurali

Come funziona un singolo neurone?



LIMITAZIONE: un solo neurone non può risolvere problemi complessi

Esempio

XOR

Il problema XOR (o "porta logica esclusiva")

| INPUT 1 | INPUT 2 | OUTPUT |
|---------|---------|--------|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Input: Due valori 0 e 1

Output:

- 1 se i valori sono diversi.
- 0 se i valori sono uguali.

Esempio

XOR

Il problema XOR (o "porta logica esclusiva")

| INPUT 1 | INPUT 2 | OUTPUT |
|---------|---------|--------|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Input: Due valori 0 e 1

Output:

- 1 se i valori sono diversi.
- 0 se i valori sono uguali.

Se vogliamo usare un SOLO
nerurone, quanti input dobbiamo
avere nel neurone? Quanti output?

Esempio

XOR

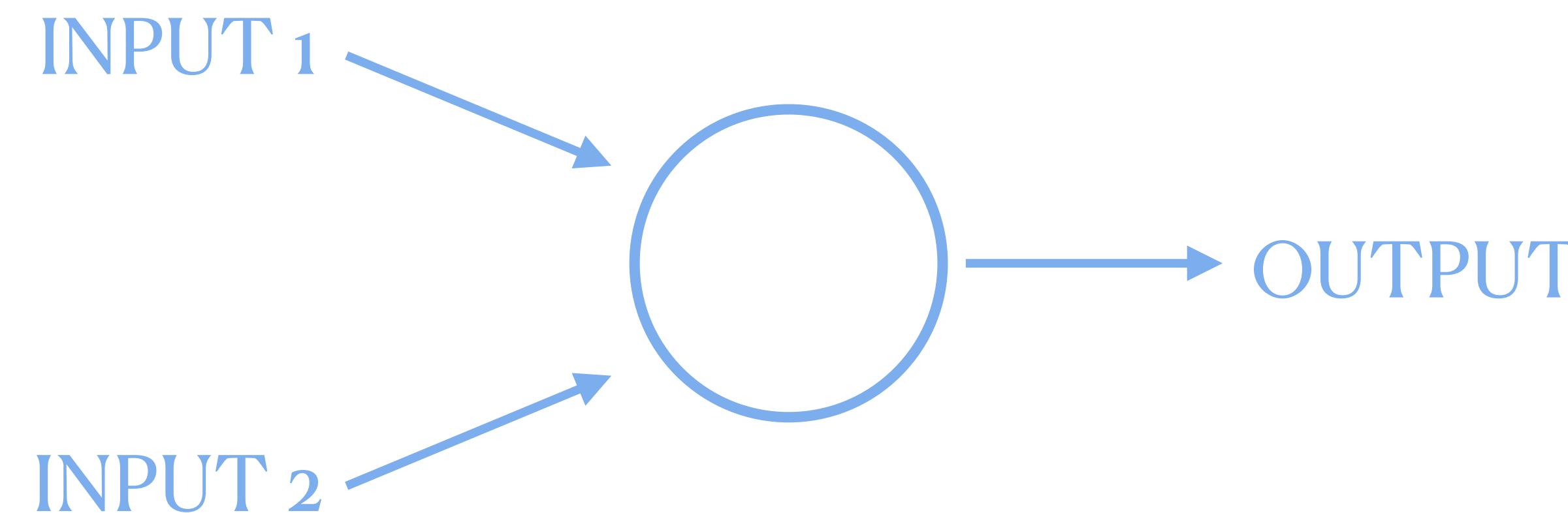
Il problema XOR (o "porta logica esclusiva")

| INPUT 1 | INPUT 2 | OUTPUT |
|---------|---------|--------|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Input: Due valori 0 e 1

Output:

- 1 se i valori sono diversi.
- 0 se i valori sono uguali.



Esempio

XOR

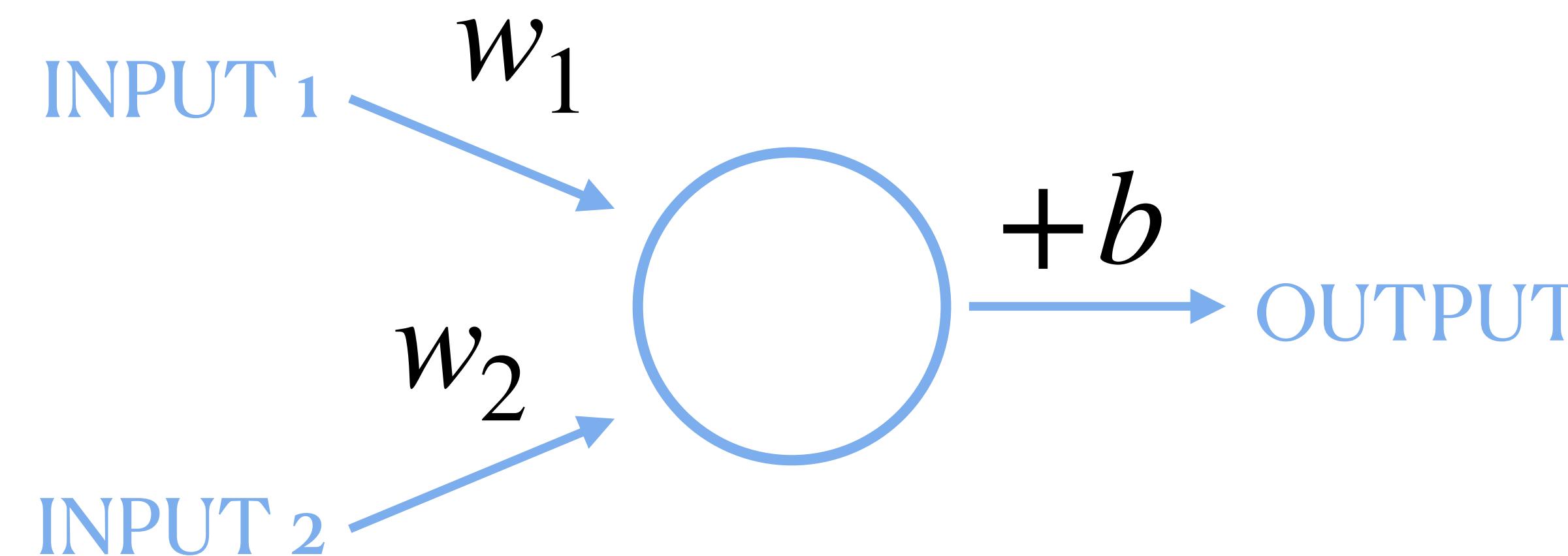
Il problema XOR (o "porta logica esclusiva")

| INPUT 1 | INPUT 2 | OUTPUT |
|---------|---------|--------|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Input: Due valori 0 e 1

Output:

- 1 se i valori sono diversi.
- 0 se i valori sono uguali.



Esempio

XOR

Il problema XOR (o "porta logica esclusiva")

| INPUT 1 | INPUT 2 | OUTPUT |
|---------|---------|--------|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Input: Due valori 0 e 1

Output:

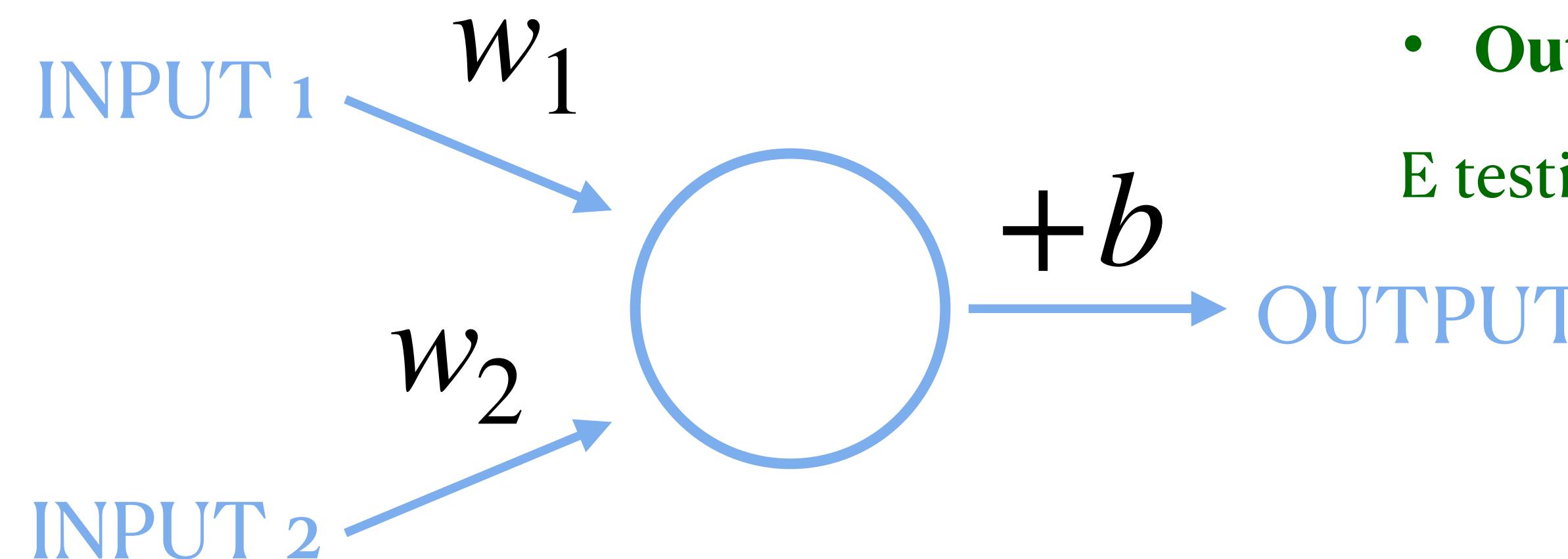
- 1 se i valori sono diversi.
- 0 se i valori sono uguali.

Proviamo a scrivere questa funzione in python usando Pycharm

Scriviamo una funzione xor_neuron(x1, x2):

- **Input:** due valori booleani x1, x2
- Dentro la funzione definiamo w1, w2 e b
- **Output:** $x1 * w1 + x2 * w2 + \text{bias}$

E testiamola!!



**Un neurone non basta a modellare problemi
complessi**

**Un neurone non basta a modellare problemi
complessi**

**E se noi considerassimo più neuroni che
comunicano tra loro?**

**Un neurone non basta a modellare problemi
complessi**

**E se noi considerassimo più neuroni che
comunicano tra loro?**

Rete Neurale

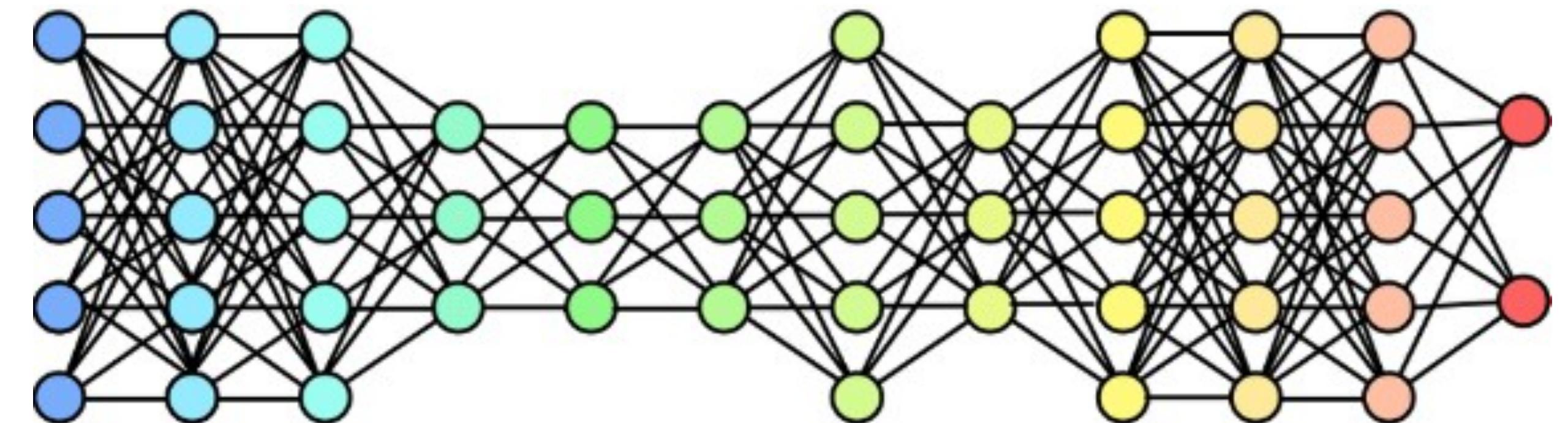
Addestrare una Rete Neurale

Ovvero trovare il valore dei suoi pesi w e termini di correzione b

Un neurone non può risolvere problemi complessi (es. XOR).

Per risolverli serve un Multi-Layer Perceptron (MLP):

- Strati intermedi (hidden layers) combinano neuroni.
- Le connessioni pesate permettono di rappresentare funzioni complesse.



I calcoli per produrre un output dato un input vanno in questa direzione



L'errore, ovvero la funzione di costo o perdita, viene calcolato in questa direzione



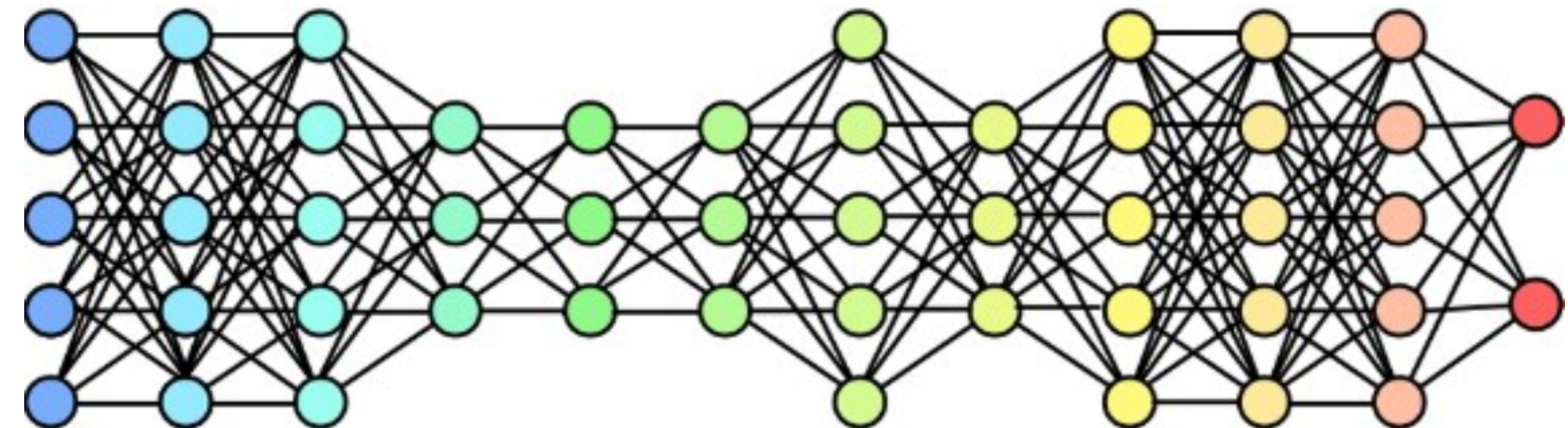
Addestrare una Rete Neurale

Ovvero trovare il valore dei suoi pesi w e termini di correzione b

Un neurone non può risolvere problemi complessi (es. XOR).

Per risolverli serve un Multi-Layer Perceptron (MLP):

- Strati intermedi (hidden layers) combinano neuroni.
- Le connessioni pesate permettono di rappresentare funzioni complesse.



**Ma ogni volta dobbiamo scrivere
“a mano” i pesi e i basi di tutti i
neuroni che vogliamo nella rete?**

**Per fortuna ci sono delle librerie che
possiamo usare facilmente che hanno
pronto e a disposizione quasi ogni tipo di
rete neurale**

**Per fortuna ci sono delle librerie che
possiamo usare facilmente che hanno
pronto e a disposizione quasi ogni tipo di
rete neurale**

Tensorflow

Tensorflow

Introduzione alla libreria per le Reti Neurali

Cos'è TensorFlow?

- **TensorFlow** è una libreria open source sviluppata da Google.
- Utilizzata per:
 - Creare e addestrare modelli di Machine Learning.
 - Costruire reti neurali complesse (come gli MLP).



TensorFlow

Tensorflow

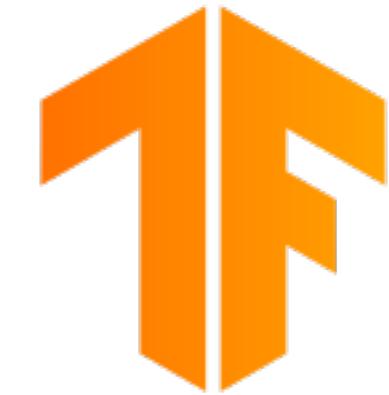
Introduzione alla libreria per le Reti Neurali

Cos'è TensorFlow?

- **TensorFlow** è una libreria open source sviluppata da Google.
- Utilizzata per:
 - Creare e addestrare modelli di Machine Learning.
 - Costruire reti neurali complesse (come gli MLP).

Perché usarlo?

- Offre molte funzioni **pronte all'uso**.
- È **ottimizzato** per performance elevate.



TensorFlow

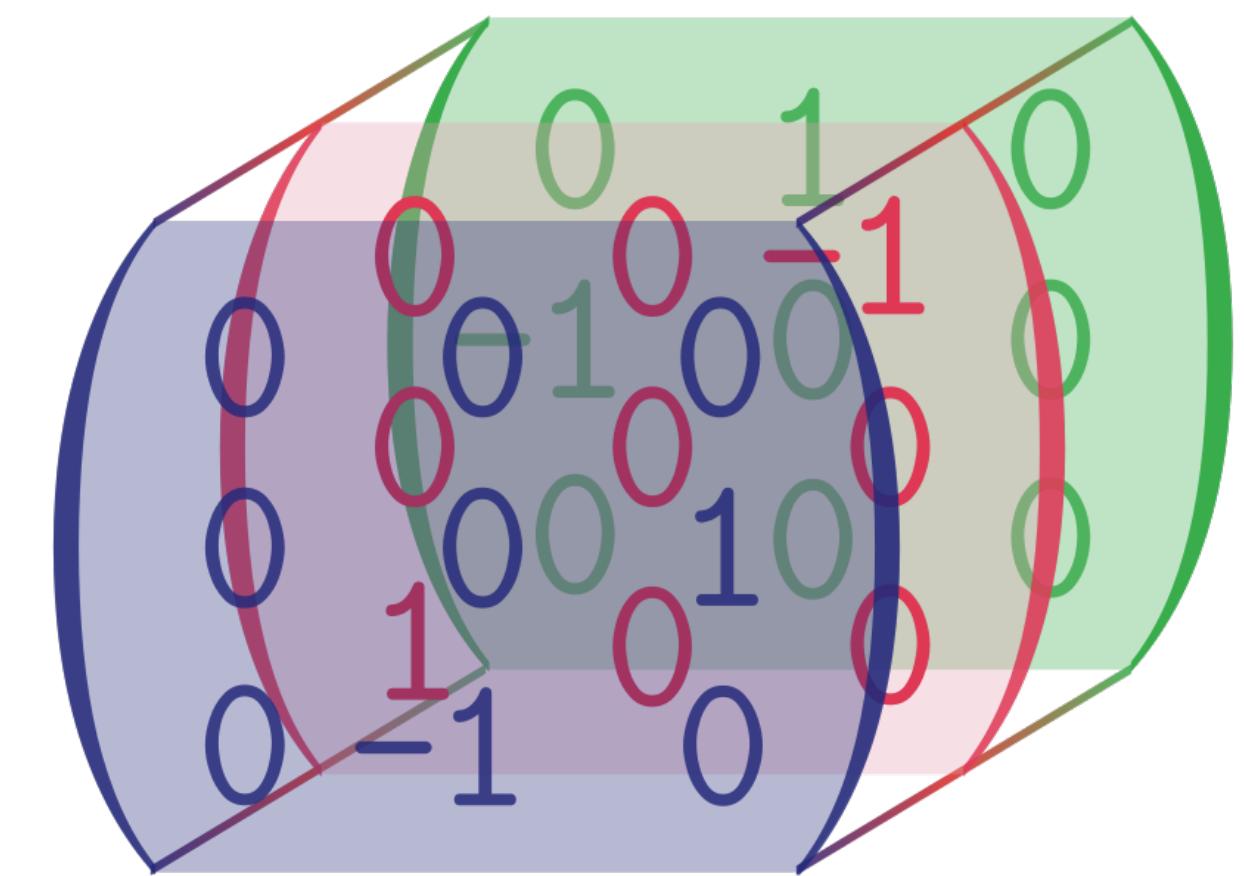
Tensorflow

Introduzione alla libreria per le Reti Neurali

Cos'è un Tensore?

- **Un tensore** è una struttura dati simile a un array.
- Viene usato per rappresentare dati (numeri, immagini, testo) nelle reti neurali.
- Esempi:
 - **Scalare**: un numero (es. 3).
 - **Vettore**: una lista di numeri (es. [1, 2, 3]).
 - **Matrice**: una tabella di numeri (es. [[1, 2], [3, 4]]).

$$\epsilon_{ijk} =$$



**Ma prima di procedere.. Cosa stiamo
facendo? Perchè lo stiamo facendo?**

**Ma prima di procedere.. Cosa stiamo
facendo? Perchè lo stiamo facendo?**

**Vogliamo costruire dei MODELLI per
fare delle predizioni**

**Ma prima di procedere.. Cosa stiamo
facendo? Perchè lo stiamo facendo?**

**Vogliamo costruire dei MODELLI per
fare delle predizioni**

**Prima di procedere, definiamo bene cosa
è un modello**

Perchè vogliamo costruire un modello?

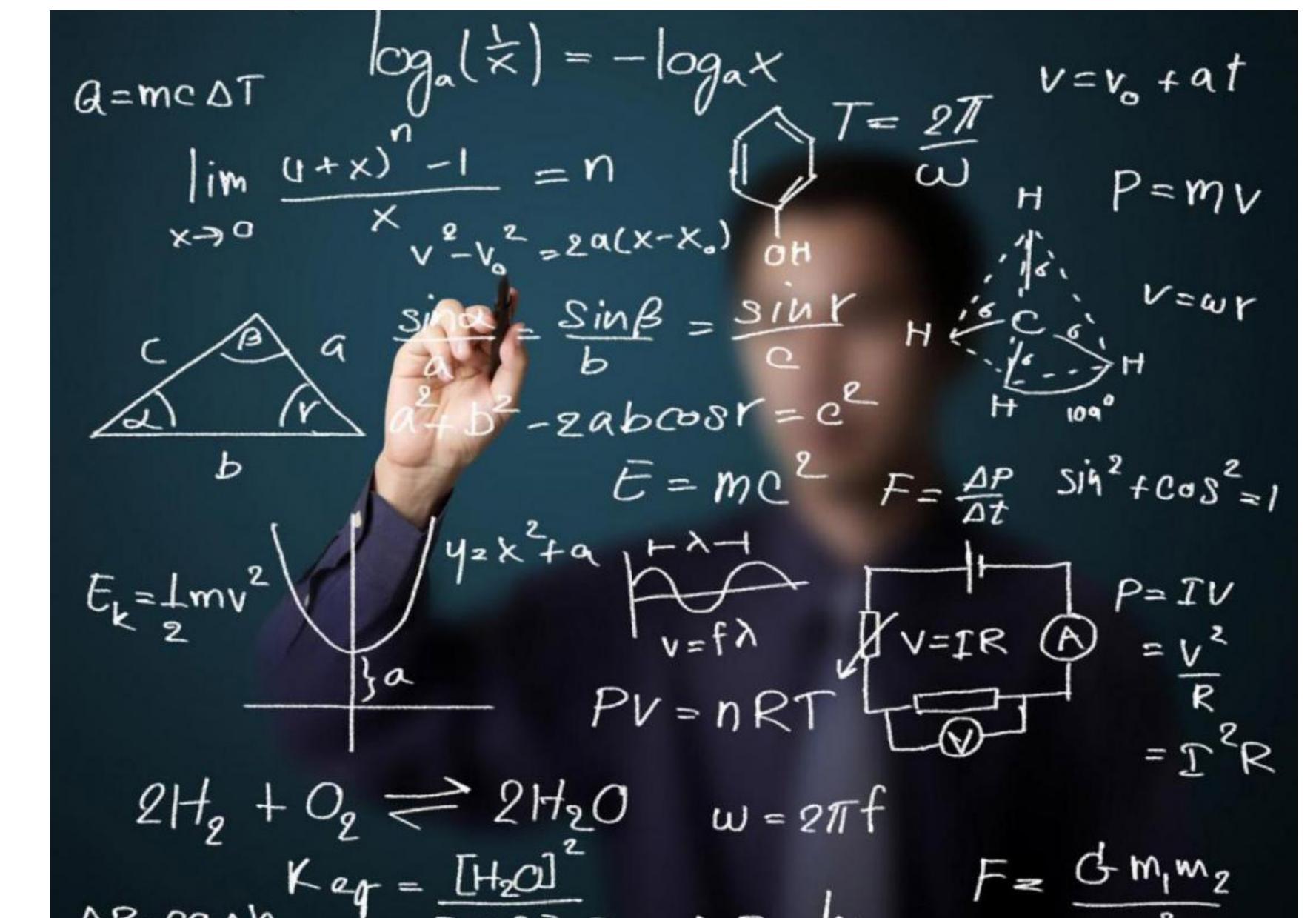
Cosa è un modello?

Perché ci serve un modello?

Problema: Non conosciamo la funzione esatta che collega input e output.

Soluzione: Addestrare un modello che:

- Osserva gli input X
- Impara a calcolare gli output Y
- Può generalizzare per predire output anche per dati mai visti.



Perchè vogliamo costruire un modello?

Cosa è un modello?

Perché ci serve un modello?

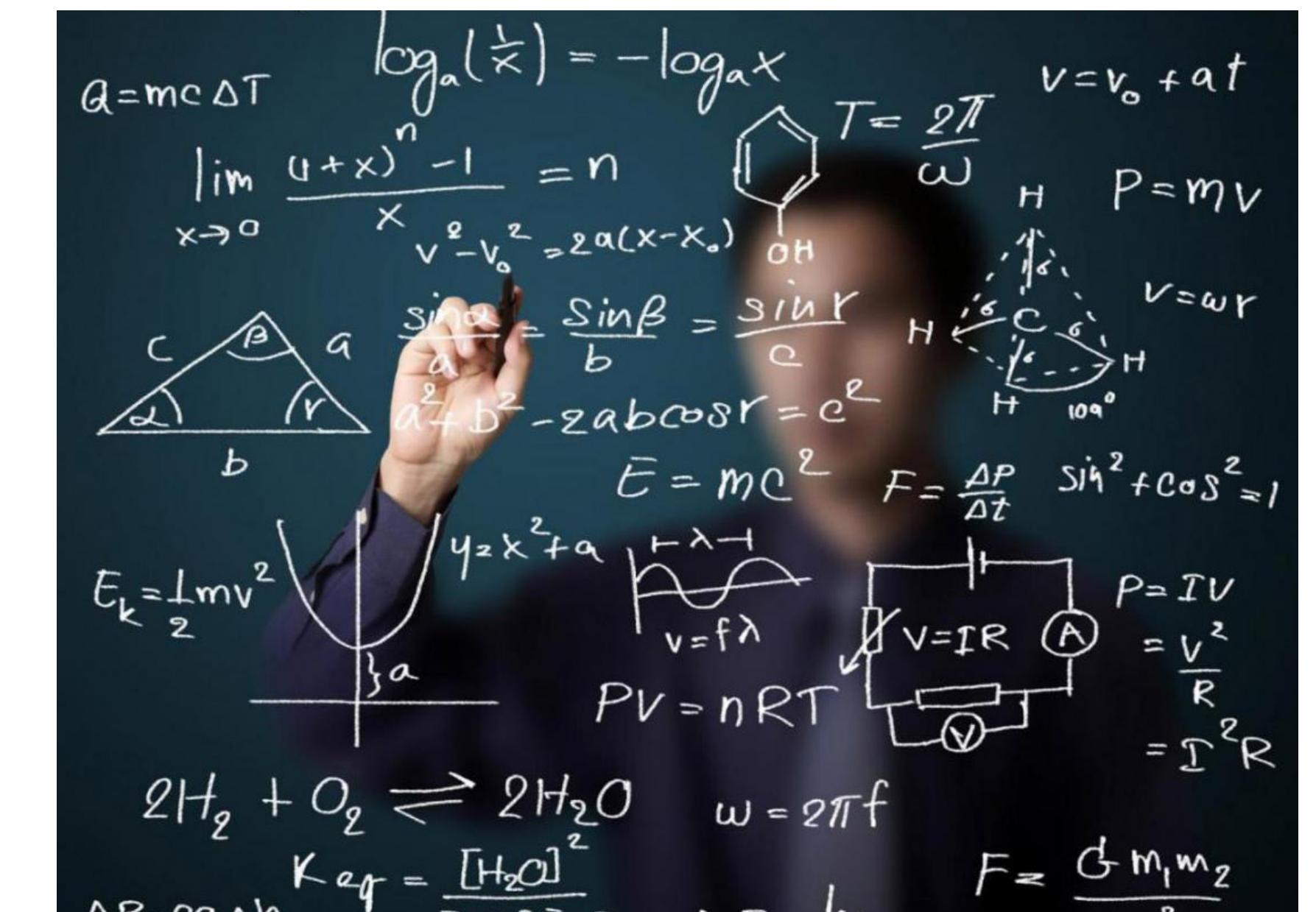
Problema: Non conosciamo la funzione esatta che collega input e output.

Soluzione: Addestrare un modello che:

- Osserva gli input X
- Impara a calcolare gli output Y
- Può generalizzare per predire output anche per dati mai visti.

Collegamento tra neuroni e modello: Come funziona un modello?

- **Input:** I dati che forniamo (ad esempio, valori numerici, immagini, testo).
- **Pesi:** Numeri che il modello regola durante l'apprendimento.
- **Output:** I valori che vogliamo predire.



Perchè vogliamo costruire un modello?

Cosa succede durante l'apprendimento?

Inizializzazione dei pesi:

- I pesi iniziano con valori casuali.
- Il modello non sa ancora come collegare input e output.

Calcolo dell'output:

- L'output viene calcolato moltiplicando gli input per i pesi e sommando un termine chiamato **bias**.
- Esempio per un neurone: $z = w_1 \cdot x_1 + w_2 \cdot x_2 + b$

Confronto con l'output reale:

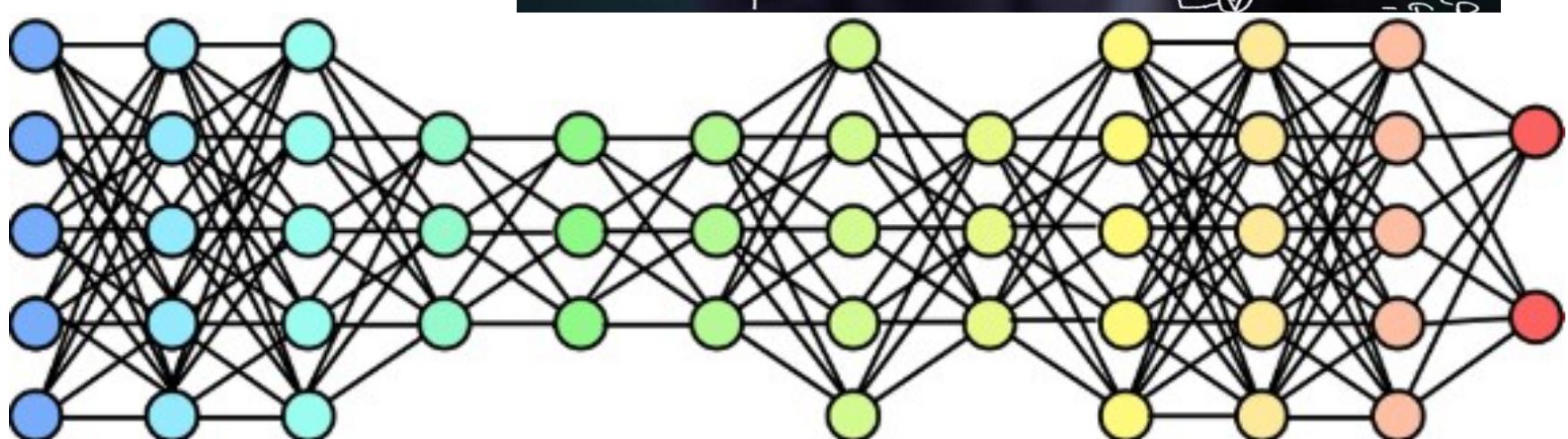
- Il modello calcola l'errore (differenza tra output predetto e reale) usando una **funzione di perdita** (ad esempio, MSE).

Aggiornamento dei pesi:

- L'algoritmo di ottimizzazione (es. Adam o SGD) aggiorna i pesi per ridurre l'errore.



$$Q = mc\Delta T \quad \log_a(\frac{1}{x}) = -\log_a x \quad v = v_0 + at$$
$$\lim_{x \rightarrow \infty} \frac{(1+x)^n - 1}{x} = n \quad T = \frac{2\pi}{\omega} \quad v = v_0 + at$$
$$v^2 - v_0^2 = 2\alpha(x - x_0) \quad H \quad P = mv$$
$$\sin \alpha = \frac{\text{opp}}{\text{hyp}} = \frac{\text{opp}}{c} \quad H \quad v = wr$$
$$c^2 = a^2 + b^2 - 2ab \cos \gamma \quad H \quad E = mc^2$$
$$E_k = \frac{1}{2}mv^2 \quad F = \frac{\Delta p}{\Delta t} \quad \sin^2 \theta + \cos^2 \theta = 1$$
$$PV = nRT \quad P = IV \quad P = \frac{V^2}{R}$$



Perchè vogliamo costruire un modello?

Cosa succede durante l'apprendimento?

Inizializzazione dei pesi:

- I pesi iniziano con valori casuali.
- Il modello non sa ancora come collegare input e output.

Calcolo dell'output:

- L'output viene calcolato moltiplicando gli input per i pesi e sommando un termine chiamato **bias**.
- Esempio per un neurone: $z = w_1 \cdot x_1 + w_2 \cdot x_2 + b$

Confronto con l'output reale:

- Il modello calcola l'errore (differenza tra output predetto e reale) usando una **funzione di perdita** (ad esempio, MSE).

Aggiornamento dei pesi:

- L'algoritmo di ottimizzazione (es. Adam o SGD) aggiorna i pesi per ridurre l'errore.

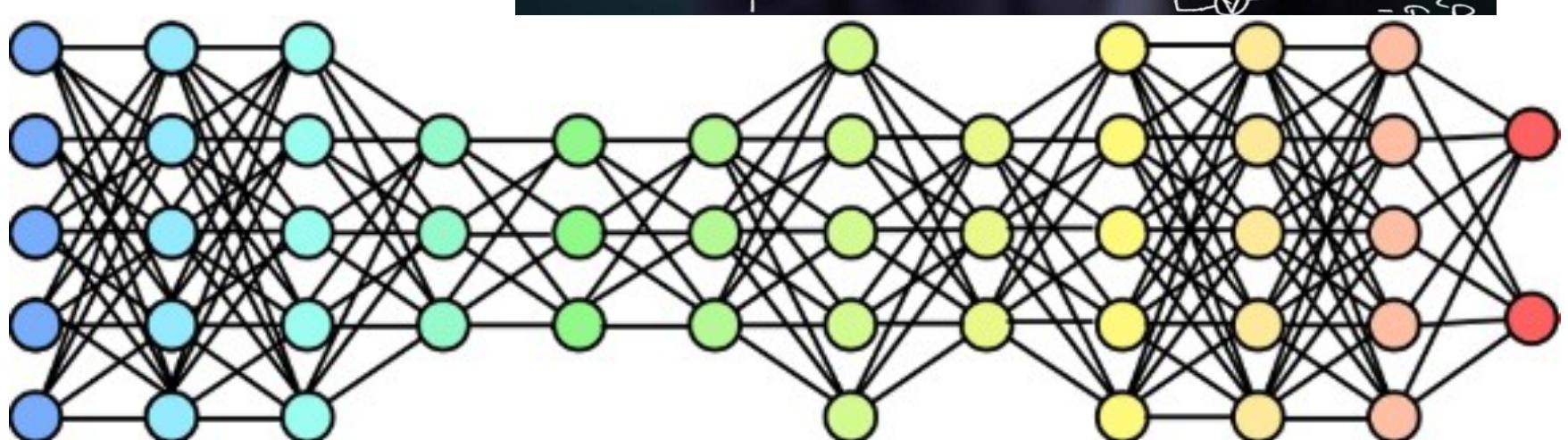
Il processo di apprendimento è iterativo:

- Il modello calcola l'output per ogni campione.
- Confronta l'output con il valore reale.
- Regola i pesi per migliorare la previsione.

Dopo molte iterazioni (**epoch**), il modello impara a fare previsioni accurate.



A chalkboard covered in mathematical handwriting. It includes:
 $a = mc\Delta T$, $\log_a(\frac{1}{x}) = -\log_a x$, $v = v_0 + at$, $T = \frac{2\pi}{\omega}$, $P = mv$,
 $\lim_{x \rightarrow \infty} \frac{(1+x)^n - 1}{x} = n$, $v^2 - v_0^2 = 2\alpha(x - x_0)$,
 $\sin \alpha = \frac{\text{opp}}{\text{hyp}}$, $\sin \beta = \frac{\text{opp}}{\text{hyp}}$, $\sin \gamma = \frac{\text{opp}}{\text{hyp}}$,
 $c^2 = a^2 + b^2 - 2ab \cos \gamma$, $E = mc^2$, $F = \frac{dp}{dt}$, $\sin^2 + \cos^2 = 1$,
 $E_k = \frac{1}{2}mv^2$, $y = x^2 + a$, $v = f\lambda$, $P = IV$,
 $PV = nRT$, $P = \frac{V^2}{R}$.



Perchè vogliamo costruire un modello?

Cosa succede durante l'apprendimento?

Inizializzazione dei pesi:

- I pesi iniziano con valori casuali.
- Il modello non sa ancora come collegare input e output.

Calcolo dell'output:

- L'output viene calcolato moltiplicando gli input per i pesi e sommando un termine chiamato **bias**.
- Esempio per un neurone: $z = w_1 \cdot x_1 + w_2 \cdot x_2 + b$

Confronto con l'output reale:

- Il modello calcola l'errore (differenza tra output predetto e reale) usando una **funzione di perdita** (ad esempio, MSE).

Aggiornamento dei pesi:

- L'algoritmo di ottimizzazione (es. Adam o SGD) aggiorna i pesi per ridurre l'errore.

Il processo di apprendimento è iterativo:

- Il modello calcola l'output per ogni campione.
- Confronta l'output con il valore reale.
- Regola i pesi per migliorare la previsione.

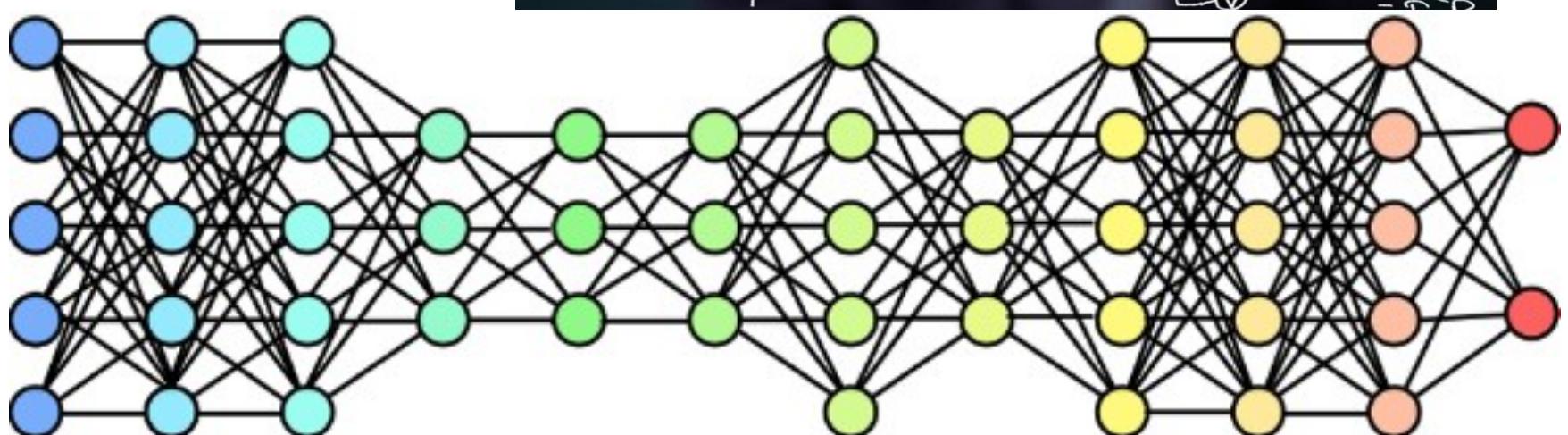
Dopo molte iterazioni (**epoch**), il modello impara a fare previsioni accurate.

Alla fine dell'addestramento:

- I pesi rappresentano la "conoscenza" del modello.
- Il modello può predire nuovi output dati nuovi input.



$$Q = mc\Delta T \quad \log_a(\frac{1}{x}) = -\log_a x \quad v = v_0 + at$$
$$\lim_{x \rightarrow \infty} \frac{(1+x)^n - 1}{x} = n \quad T = \frac{2\pi}{\omega} \quad v = v_0 + at$$
$$v^2 - v_0^2 = 2\alpha(x - x_0) \quad H \quad P = mv$$
$$\sin \alpha = \frac{\sin \beta}{b} = \frac{\sin \gamma}{c} \quad H \quad v = \omega r$$
$$c^2 = a^2 + b^2 - 2ab \cos \gamma \quad H \quad \alpha = 109^\circ$$
$$E = mc^2 \quad F = \frac{dp}{dt} \quad \sin^2 + \cos^2 = 1$$
$$E_k = \frac{1}{2}mv^2 \quad v = f\lambda \quad P = IV$$
$$PV = nRT \quad V = IR \quad P = \frac{V^2}{R}$$



Costruiamo il nostro primo modello MLP

Costruiamo il nostro primo modello MLP

**Per farlo dobbiamo innanzitutto trovare
dei dati che vogliamo predire... come???**

Costruiamo il nostro primo modello MLP

**Per farlo dobbiamo innanzitutto trovare
dei dati che vogliamo predire... come???**

**Usiamo una funzione matematica, e proviamo a vedere se
riusciamo a costruire un modello che predice le y date le x**

Approssimare una funzione con un MLP

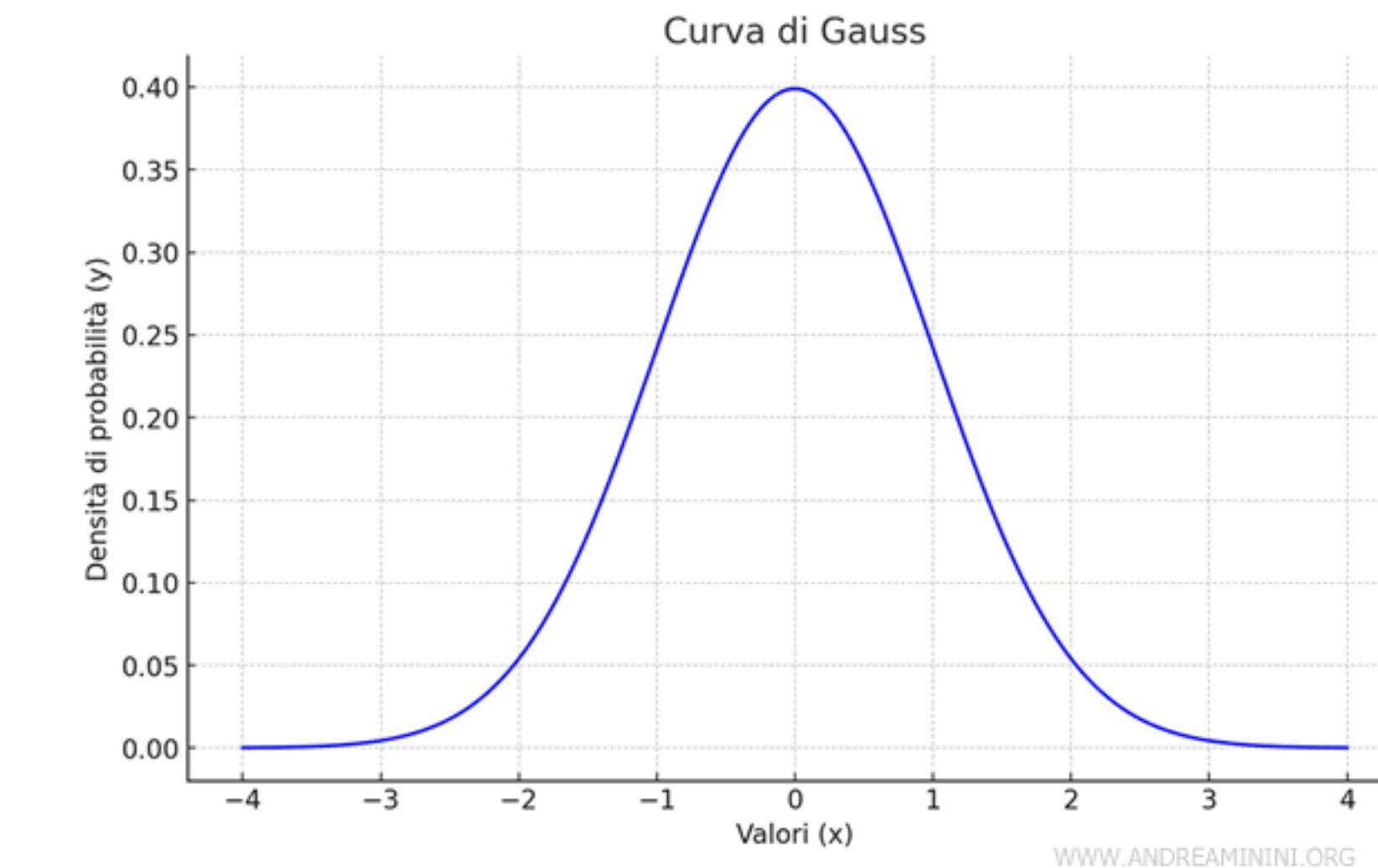
Generare i dati: costruiamo una funzione matematica con Python

La funzione **gaussiana** è definita come:

$$y = e^{-x^2}$$

Perché una funzione gaussiana?

- È **semplice** da calcolare ma **non lineare**, quindi interessante da modellare con un MLP.



Approssimare una funzione con un MLP

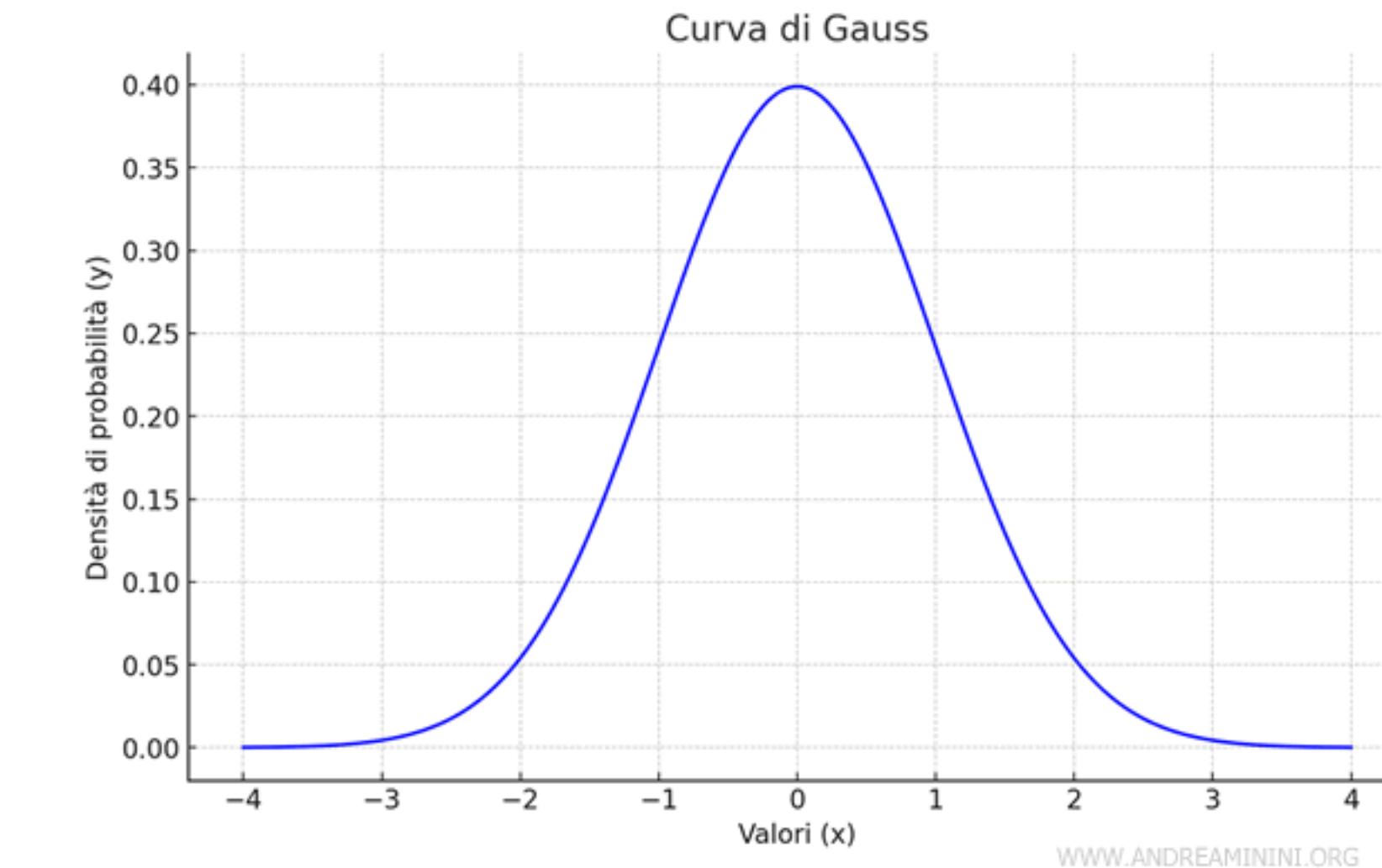
Generare i dati: costruiamo una funzione matematica con Python

La funzione **gaussiana** è definita come:

$$y = e^{-x^2}$$

Perché una funzione gaussiana?

- È **semplice** da calcolare ma **non lineare**, quindi interessante da modellare con un MLP.



```
def gaussiana(x):
    return np.exp(-x * x)

# Generazione dei dati
x_gauss = np.linspace(-5, stop: 5, num: 1000)
y_gauss = gaussiana(x_gauss)
```

Abbiamo bisogno di due tipi di dati

- 1. Dati da cui MLP impara la funzione (**train**)**
- 2. Dati che il modello non ha mai visto per testare se ha imparato (**test**)**

Abbiamo bisogno di due tipi di dati

1. Dati da cui MLP impara la funzione (**train**)
2. Dati che il modello non ha mai visto per testare se ha imparato (**test**)

Dividiamo i dati che abbiamo in train e test

Anche qua, invece che “fare a mano”, abbiamo già una libreria python che fa al caso nostro: **sklearn**

Approssimare una funzione con un MLP

Prepariamo i dati per addestrare il modello: division in train e test

Domanda: Perché dividiamo i dati in train e test?

- Per valutare quanto bene il modello generalizza su dati mai visti.
- Dati di train: Usati per addestrare il modello e aggiornare i pesi.
- Dati di test: Usati per verificare le prestazioni finali.

Obiettivo: Preparare i dati per addestrare un modello.

- Usiamo `train_test_split` da `sklearn` per dividere i dati

Approssimare una funzione con un MLP

Prepariamo i dati per addestrare il modello: division in train e test

Domanda: Perché dividiamo i dati in train e test?

- Per valutare quanto bene il modello generalizza su dati mai visti.
- Dati di train: Usati per addestrare il modello e aggiornare i pesi.
- Dati di test: Usati per verificare le prestazioni finali.

```
16 # Divisione in train e test
17 x_train_gauss, x_test_gauss, y_train_gauss, y_test_gauss = train_test_split(
18     *arrays: x_gauss, y_gauss, test_size=0.2, shuffle=True
19 )
```

Obiettivo: Preparare i dati per addestrare un modello.

- Usiamo `train_test_split` da `sklearn` per dividere i dati

**Vogliamo visualizzare i dati che abbiamo
generato e anche la loro divisione in train
e test**

Vogliamo visualizzare i dati che abbiamo generato e anche la loro divisione in train e test

Anche qua, invece che “fare a mano”, abbiamo già una libreria python che fa al caso nostro: `matplotlib`

Approssimare una funzione con un MLP

Prepariamo i dati per addestrare il modello: division in train e test

Domanda: Perché dividiamo i dati in train e test?

- Per valutare quanto bene il modello generalizza su dati mai visti.
- Dati di train: Usati per addestrare il modello e aggiornare i pesi.
- Dati di test: Usati per verificare le prestazioni finali.

```
16 # Divisione in train e test
17 x_train_gauss, x_test_gauss, y_train_gauss, y_test_gauss = train_test_split(
18     *arrays: x_gauss, y_gauss, test_size=0.2, shuffle=True
19 )
```

Obiettivo: Preparare i dati per addestrare un modello.

- Usiamo `train_test_split` da `sklearn` per dividere i dati

Visualizzazione dei punti di train e test:

- Punti rossi: Train (usati per addestrare il modello).
- Punti gialli: Test (usati per valutare il modello).

Approssimare una funzione con un MLP

Prepariamo i dati per addestrare il modello: division in train e test

Domanda: Perché dividiamo i dati in train e test?

- Per valutare quanto bene il modello generalizza su dati mai visti.
- Dati di train: Usati per addestrare il modello e aggiornare i pesi.
- Dati di test: Usati per verificare le prestazioni finali.

```
16 # Divisione in train e test
17 x_train_gauss, x_test_gauss, y_train_gauss, y_test_gauss = train_test_split(
18     *arrays: x_gauss, y_gauss, test_size=0.2, shuffle=True
19 )
```

Obiettivo: Preparare i dati per addestrare un modello.

- Usiamo `train_test_split` da `sklearn` per dividere i dati

Visualizzazione dei punti di train e test:

- Punti rossi: Train (usati per addestrare il modello).
- Punti gialli: Test (usati per valutare il modello).

```
21 # Visualizzazione dei dati
22 plt.plot(*args: x_gauss, y_gauss, color='gray', label='Gaussiana')
23 plt.scatter(x_train_gauss, y_train_gauss, color='red', label='Train Points', s=5)
24 plt.scatter(x_test_gauss, y_test_gauss, color='yellow', label='Test Points', s=5)
25 plt.legend()
26 plt.tight_layout()
27 plt.show()
```

Ora vogliamo definire la rete neurale, per farlo serve:

- 1. Decidere l'architettura della rete (quanti neuroni in input, quanti nello strato intermedio, quanti in quello finale)**
- 2. Definire una funzione di attivazione**

Ora vogliamo definire la rete neurale, per farlo serve:

1. Decidere l'architettura della rete (quanti neuroni in input, quanti nello strato intermedio, quanti in quello finale)
2. Definire una funzione di attivazione

Approssimare una funzione con un MLP

Funzioni di Attivazione

Definizione: Sono trasformazioni matematiche applicate al risultato di un neurone.

Obiettivo:

- Aggiungere **non linearità** al modello.
- Permettere alla rete di apprendere funzioni complesse.

Dove si usano?

Dopo ogni neurone in un layer nascosto o nell'output.

Approssimare una funzione con un MLP

Funzioni di Attivazione

Definizione: Sono trasformazioni matematiche applicate al risultato di un neurone.

Obiettivo:

- Aggiungere **non linearità** al modello.
- Permettere alla rete di apprendere funzioni complesse.

Dove si usano?

Dopo ogni neurone in un layer nascosto o nell'output.

2. **ReLU (Rectified Linear Unit):** $f(x) = \max(0, x)$

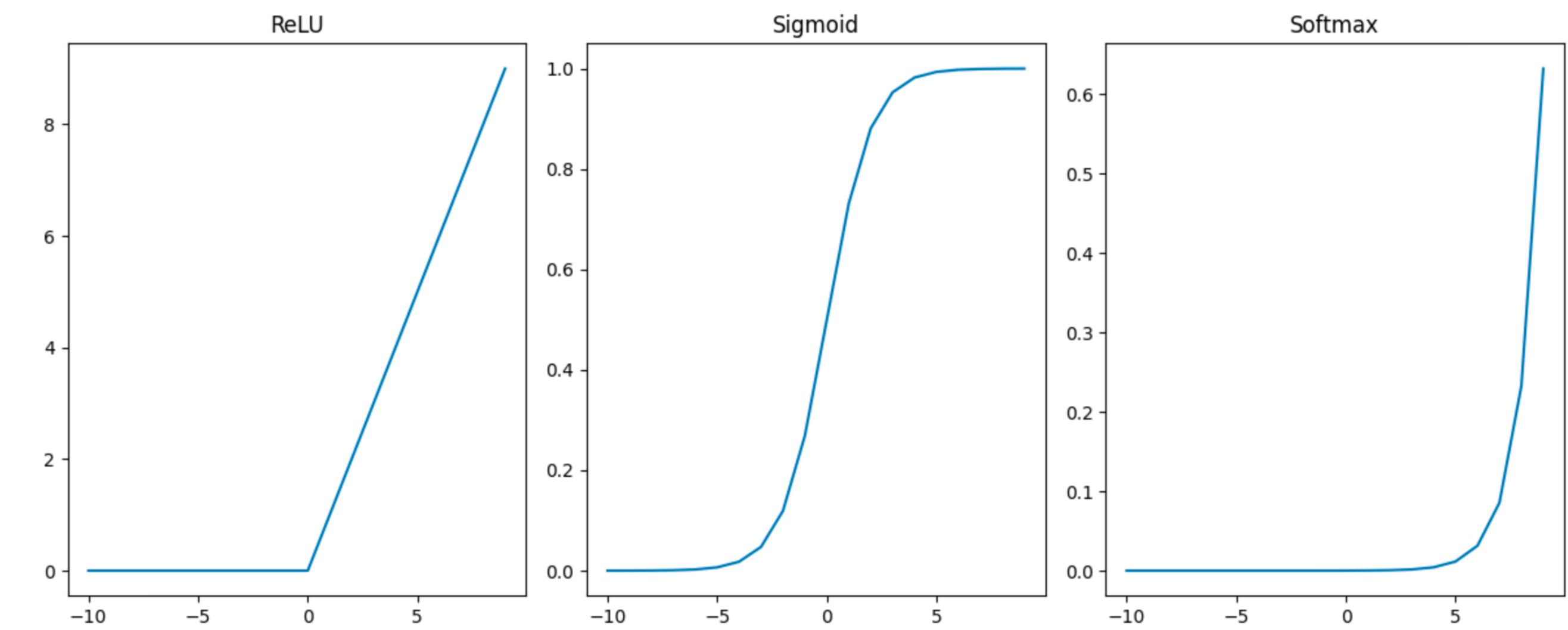
- In TensorFlow: activation='relu'.

3. **Sigmoid:** $f(x) = \frac{1}{1 + e^{-x}}$

- In TensorFlow: activation='sigmoid'.

4. **Softmax:** $f(x) = \frac{e^{x_j}}{\sum_j e^{x_j}}$

- In TensorFlow: activation='softmax'.



Ora vogliamo definire la rete neurale, per farlo serve:

- 1. Decidere l'architettura della rete (quanti neuroni in input, quanti nello strato intermedio, quanti in quello finale)**
- 2. Definire una funzione di attivazione**

Approssimare una funzione con un MLP

Creazione di un modello con Tensorflow

Struttura del modello:

- **Input:** Valore di x
- **Hidden layer:** Due layer intermedi con 10 neuroni
- **Output:** valore predetto di y

Di quanti input abbiamo bisogno? Di quanti output?

Approssimare una funzione con un MLP

Creazione di un modello con Tensorflow

Struttura del modello:

- **Input:** Valore di x
 - **Hidden layer:** Due layer intermedi con 10 neuroni
 - **Output:** valore predetto di y
-
- **Dense(10, activation='relu')**: Aggiunge 2 neuroni con funzione di attivazione ReLU.
 - **input_shape=(1,)**: Specifica che l'input ha una sola caratteristica (valori x).
 - **Dense(1)**: Output layer con un solo neurone.

Di quanti input abbiamo bisogno? Di quanti output?

```
29 # Creazione del modello
30 gauss_mlp = tf.keras.models.Sequential([
31     tf.keras.layers.Dense(10, activation='relu', input_shape=(1,)), # Strato di input e primo hidden
32     tf.keras.layers.Dense(units=10, activation='relu'), # Secondo hidden layer
33     tf.keras.layers.Dense(1) # Strato di output
34 ])
```

Ora sappiamo che i pesi sono inizializzati in maniera casuale

Ora sappiamo che i pesi sono inizializzati in maniera casuale

**Dobbiamo TRAINARE il modello (=aggiornare i pesi) affinché
diventi un BUON modello**

Ora sappiamo che i pesi sono inizializzati in maniera casuale

**Dobbiamo TRAINARE il modello (=aggiornare i pesi) affinché
diventi un BUON modello**

Per farlo ci servono alcuni ingredienti, come abbiamo visto:

- Che ottimizzatore usare per aggiornare i pesi?**
- Che funzione usare per misurare l'errore?**

Ora sappiamo che i pesi sono inizializzati in maniera casuale

**Dobbiamo TRAINARE il modello (=aggiornare i pesi) affinché
diventi un BUON modello**

Per farlo ci servono alcuni ingredienti, come abbiamo visto:

- **Che ottimizzatore usare per aggiornare i pesi?**
- **Che funzione usare per misurare l'errore?**

Approssimare una funzione con un MLP

Ottimizzatori

Definizione: Algoritmi che aggiornano i pesi del modello per minimizzare la funzione di perdita.

- Obiettivo:
 - Migliorare le previsioni del modello.
 - Trovare i pesi ottimali più velocemente.
- Tipi principali di ottimizzatori
 - Stochastic Gradient Descent (SGD):
 - Adam (Adaptive Moment Estimation):
 - RMSprop

Approssimare una funzione con un MLP

Ottimizzatori

Definizione: Algoritmi che aggiornano i pesi del modello per minimizzare la funzione di perdita.

- Obiettivo:
 - Migliorare le previsioni del modello.
 - Trovare i pesi ottimali più velocemente.
- Tipi principali di ottimizzatori
 - Stochastic Gradient Descent (SGD):
 - Adam (Adaptive Moment Estimation):
 - RMSprop

Ogni ottimizzatore lavora con un learning rate, che decide con quale “velocità” vengono aggiornati i pesi

Approssimare una funzione con un MLP

Learning Rate

Cos'è il Learning Rate?

- A. È il tasso con cui l'ottimizzatore aggiorna i pesi del modello durante l'addestramento.
- B. Controlla quanto grandi o piccoli sono i passi che il modello compie per minimizzare la funzione di perdita.

Approssimare una funzione con un MLP

Learning Rate

Cos'è il Learning Rate?

- A. È il tasso con cui l'ottimizzatore aggiorna i pesi del modello durante l'addestramento.
- B. Controlla quanto grandi o piccoli sono i passi che il modello compie per minimizzare la funzione di perdita.

- **Learning Rate troppo alto:**

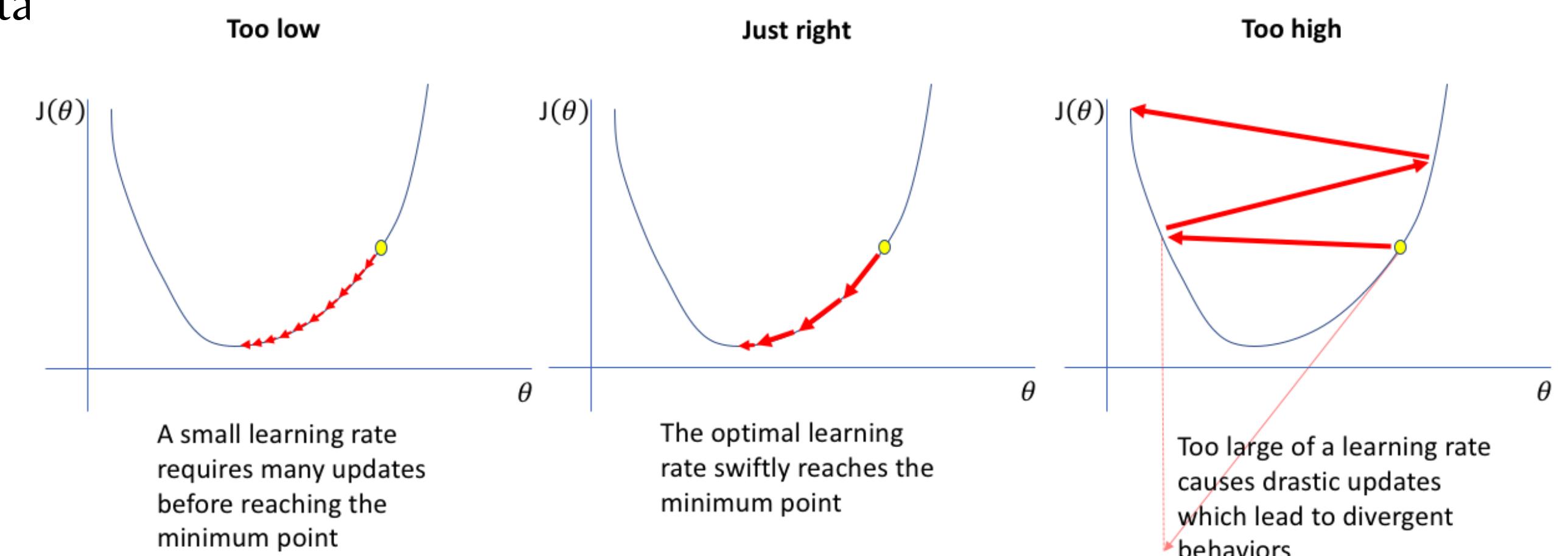
- Aggiornamenti dei pesi molto grandi.
- Il modello potrebbe saltare il minimo della funzione di perdita
- Risultato: Mancata convergenza

- **Learning Rate troppo basso:**

- Aggiornamenti dei pesi molto piccoli.
- Il modello impiega troppo tempo per convergere.
- Risultato: L'addestramento è inefficiente e costoso.

- **Learning Rate ottimale:**

- Aggiornamenti bilanciati.
- Il modello convergerà verso il minimo in un tempo ragionevole.



Ora sappiamo che i pesi sono inizializzati in maniera casuale

**Dobbiamo TRAINARE il modello (=aggiornare i pesi) affinché
diventi un BUON modello**

Per farlo ci servono alcuni ingredienti, come abbiamo visto:

- Che ottimizzatore usare per aggiornare i pesi?**
- Che funzione usare per misurare l'errore?**

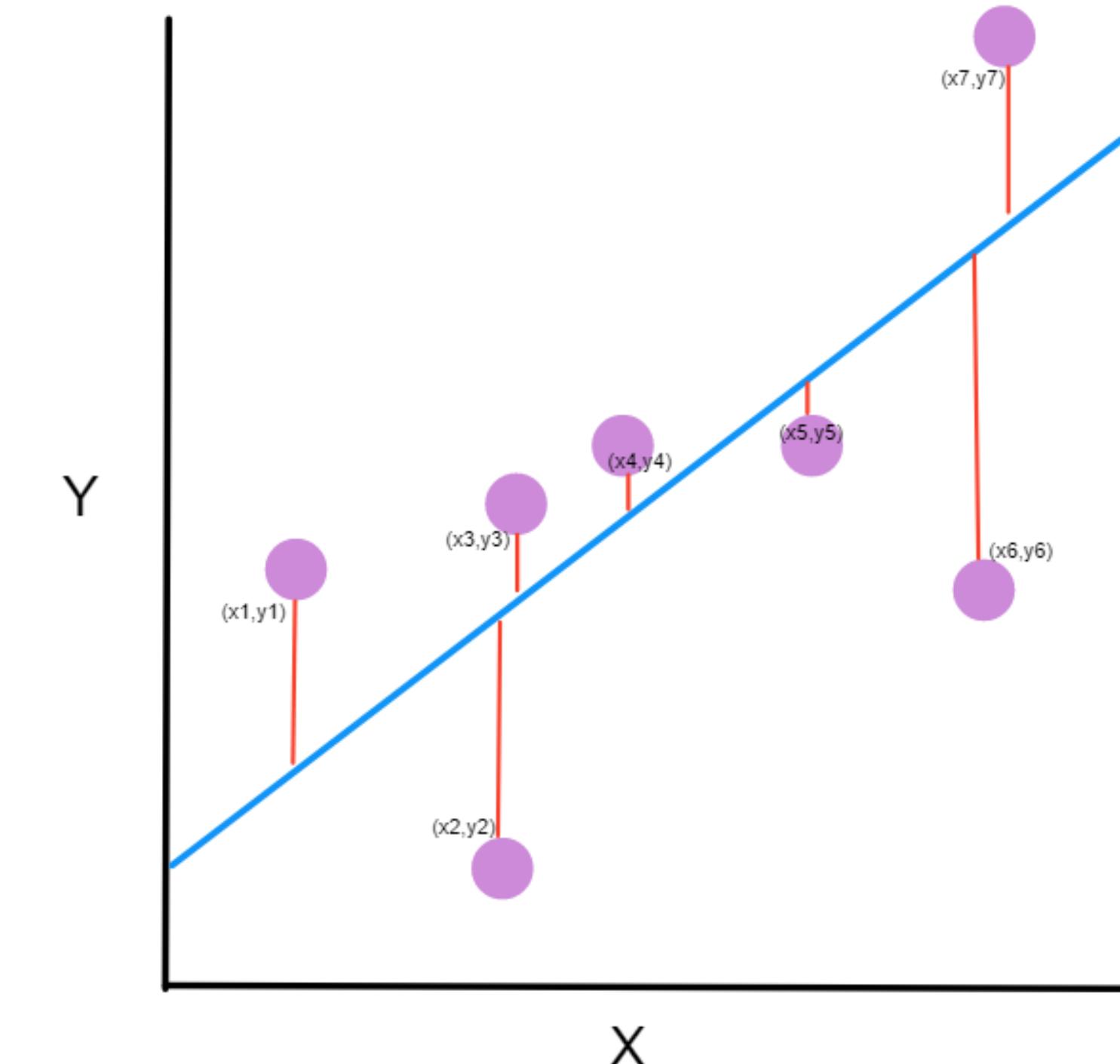
Approssimare una funzione con un MLP

Funzione di errore

Definizione: Una misura che calcola la media degli errori quadrati tra i valori previsti dal modello (\hat{y}) e i valori reali (y).

Formula: $MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$

- n : Numero di campioni.
- y_i : Valore reale del campione i
- \hat{y}_i : Valore previsto dal modello per il campione i



Approssimare una funzione con un MLP

Compilazione ed addestramento del modello

Codice per compilare il modello

```
40 # Compilazione del modello
41 lr = 0.01
42 gauss_mlp.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
43                     loss="mean_squared_error",
44                     metrics=['mean_squared_error'])
```

1. **Ottimizzatore:** Adam, per aggiornare i pesi del modello.
2. **Learning Rate:** 0.01, passo con cui si muove l'ottimizzatore
3. **Funzione di perdita:** mean_squared_error, per misurare la distanza tra predizioni e valori reali.

Approssimare una funzione con un MLP

Compilazione ed addestramento del modello

Codice per compilare il modello

```
40 # Compilazione del modello
41 lr = 0.01
42 gauss_mlp.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
43                     loss="mean_squared_error",
44                     metrics=['mean_squared_error'])
```

Codice per addestrare il modello

```
46 # Addestramento del modello
47 n_epochs = 150
48 batch_size = 64
49 history = gauss_mlp.fit(
50     x_train_gauss,
51     y_train_gauss,
52     validation_split=0.2,
53     epochs=n_epochs,
54     batch_size=batch_size,
55     verbose=1
56 )
```

1. **Ottimizzatore:** Adam, per aggiornare i pesi del modello.
2. **Learning Rate:** 0.01, passo con cui si muove l'ottimizzatore
3. **Funzione di perdita:** mean_squared_error, per misurare la distanza tra predizioni e valori reali.

1. **Validation split:** 20% dei dati di train vengono usati per valutare il modello durante l'addestramento.
2. **Epochs:** Numero di iterazioni sui dati di train.
3. **Batch size:** Quanti campioni vengono elaborati insieme.

Ora abbiamo trainato il modello

Ora abbiamo trainato il modello

Ma sarà un buon modello? Verifichiamolo!

Approssimare una funzione con un MLP

Valutazione e test del modello

Obiettivo: Verificare quanto bene il modello approssima la funzione su dati mai visti.

Perché valutare?

- Per capire se il modello sta sovradattando i dati di train o se generalizza bene.

```
58 # Valutazione del modello
59 test_loss = gauss_mlp.evaluate(x_test_gauss, y_test_gauss, verbose=0)
60 print(f"Loss sul set di test: {test_loss}")
```

**Ok, non è molto chiaro.. forse meglio
visualizzare graficamente le nostre
previsioni!!!!**

Approssimare una funzione con un MLP

Visualizziamo i risultati

Obiettivo: Confrontare le predizioni del modello con i valori reali.

Cosa mostra il grafico?

- Punti rossi: Valori predetti dal modello.
- Punti grigi: Valori reali della funzione gaussiana.

```
65 # Visualizzazione delle predizioni
66 plt.scatter(x_test_gauss, y_pred_gauss, label='Predictions', marker='x', color='red')
67 plt.scatter(x_test_gauss, y_test_gauss, label='Ground Truth', marker='o', s=15, color='gray')
68 plt.legend()
69 plt.tight_layout()
70 plt.show()
```

Ok, siamo stati bravi.. Ma è perchè abbiamo avuto fortuna o perchè abbiamo costruito un buon modello?

Ok, siamo stati bravi.. Ma è perchè abbiamo avuto fortuna o perchè abbiamo costruito un buon modello?

Proviamo a cambiare qualcosa e vediamo se funziona ancora:

- 1. Diminuiamo il numero di epoche di training a 10**
- 2. Diminuiamo il numero di neuroni negli HIDDEN LAYER**

Ok, siamo stati bravi.. Ma è perchè abbiamo avuto fortuna o perchè abbiamo costruito un buon modello?

Proviamo a cambiare qualcosa e vediamo se funziona ancora:

- 1. Diminuiamo il numero di epoche di training a 10**
- 2. Diminuiamo il numero di neuroni negli HIDDEN LAYER**

Prima di farlo, ragioniamo.. quali sono le conseguenze intuitivamente?

Ora abbiamo visto come usare un MLP per la REGRESSIONE

Ora abbiamo visto come usare un MLP per la REGRESSIONE

Facciamo insieme un esempio per la CLASSIFICAZIONE

Ora abbiamo visto come usare un MLP per la REGRESSIONE

Facciamo insieme un esempio per la CLASSIFICAZIONE

Creiamo un problema di classificazione binaria dove MLP deve distinguere due classi di cerchi (rosso e blu)

Come iniziamo?

Creiamo insieme le due classi di cerchi e visualizziamole

MLP Per la classificazione

```
8 # Generazione di cerchi non concentrici
9 x_circles, y_circles = make_circles(n_samples=100, noise=0.1)
10
11 # Visualizzazione
12 plt.scatter(x_circles[np.where(y_circles == 0)[0], 0], x_circles[np.where(y_circles == 0)[0], 1],
13               color='red', label='Classe 0')
14 plt.scatter(x_circles[np.where(y_circles == 1)[0], 0], x_circles[np.where(y_circles == 1)[0], 1],
15               color='blue', label='Classe 1')
16 plt.xlabel(r'$x_1$')
17 plt.ylabel(r'$x_2$')
18 plt.legend()
19 plt.tight_layout()
20 plt.show()
```

Poi?

Dividiamo i dati in train e test

MLP Per la classificazione

```
22 # Divisione in train e test
23 x_train_circles, x_test_circles, y_train_circles, y_test_circles = train_test_split(
24     *arrays: x_circles, y_circles, test_size=0.2, shuffle=True
25 )
26
27 # Preprocessing dei dati
28 y_train_circles = y_train_circles.reshape(-1, 1)
29 y_test_circles = y_test_circles.reshape(-1, 1)
```

Poi?

Poi?

Esatto, dobbiamo **creare il modello**, ma stiamo attenti:

- I dati di **input** avranno **due features**, per le due coordinate x_1, x_2 che definiscono le coordinate del punto

Poi?

Esatto, dobbiamo **creare il modello**, ma stiamo attenti:

- I dati di **input** avranno **due features**, per le due coordinate x_1, x_2 che definiscono le coordinate del punto
- L'**output** sarà una **probabilità** che può essere ZERO o UNO (rappresenta l'appartenenza a una delle due classi)

Poi?

Esatto, dobbiamo **creare il modello**, ma stiamo attenti:

- I dati di **input** avranno **due features**, per le due coordinate x_1, x_2 che definiscono le coordinate del punto
- L'**output** sarà una **probabilità** che può essere ZERO o UNO (rappresenta l'appartenenza a una delle due classi)
- Serve una **funzione di attivazione** nel layer finale che assicuri che l'output sia forzato ad essere ZERO o UNO

Creiamo il modello

MLP Per la classificazione

```
31 # Creazione del modello
32 mlp_circles = tf.keras.models.Sequential([
33     tf.keras.layers.Dense(10, input_shape=(2,), activation='relu'), # Primo hidden layer
34     tf.keras.layers.Dense(units: 10, activation='relu'), # Secondo hidden layer
35     tf.keras.layers.Dense(units: 1, activation='sigmoid') # Strato di output per classificazione binaria
36 ])
```

Poi?

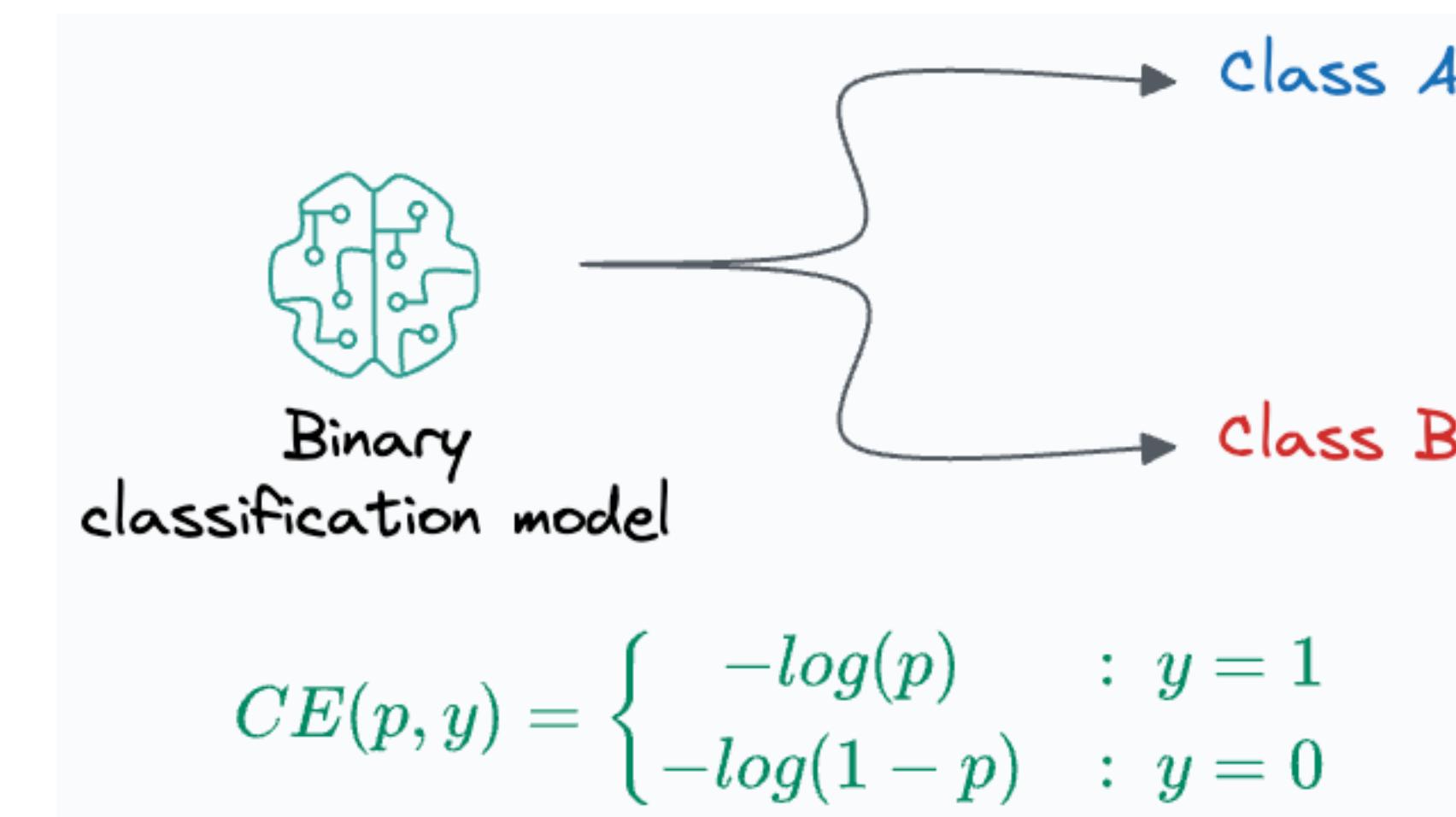
Poi?

Esatto, dobbiamo **compilare il modello**, ma stiamo attenti:

Poi?

Esatto, dobbiamo **compilare il modello**, ma stiamo attenti:

- Come funzione di errore non andrà più bene la RMSE ma serve una funzione che calcoli l'errore di assegnazione di un dato a una classe piuttosto che un'altra
- **Binary Cross Entropy**



Compiliamo il modello

MLP Per la classificazione

```
38 # Compilazione del modello
39 lr = 0.01
40 mlp_circles.compile(
41     optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
42     loss='binary_crossentropy', # Funzione di perdita per classificazione binaria
43     metrics=['accuracy']
44 )
```

Poi?

Addestriamo il modello

MLP Per la classificazione

```
46 # Addestramento del modello  
47 n_epochs = 100  
48 batch_size = 64  
49 history = mlp_circles.fit(  
50     x_train_circles,  
51     y_train_circles,  
52     validation_split=0.2,  
53     epochs=n_epochs,  
54     batch_size=batch_size,  
55     verbose=1  
56 )
```

Poi?

Poi?

Esatto, dobbiamo **valutare il modello (sui dati di test)**:

- Invece di visualizzare i dati come sulla gaussiana introduciamo due metriche nuove per valutare le performances del nostro modello
 - Grafico della loss sul train e sul test (va bene sia per regressione che classificazione)
 - Confusion Matric (va bene solo per classificazione)

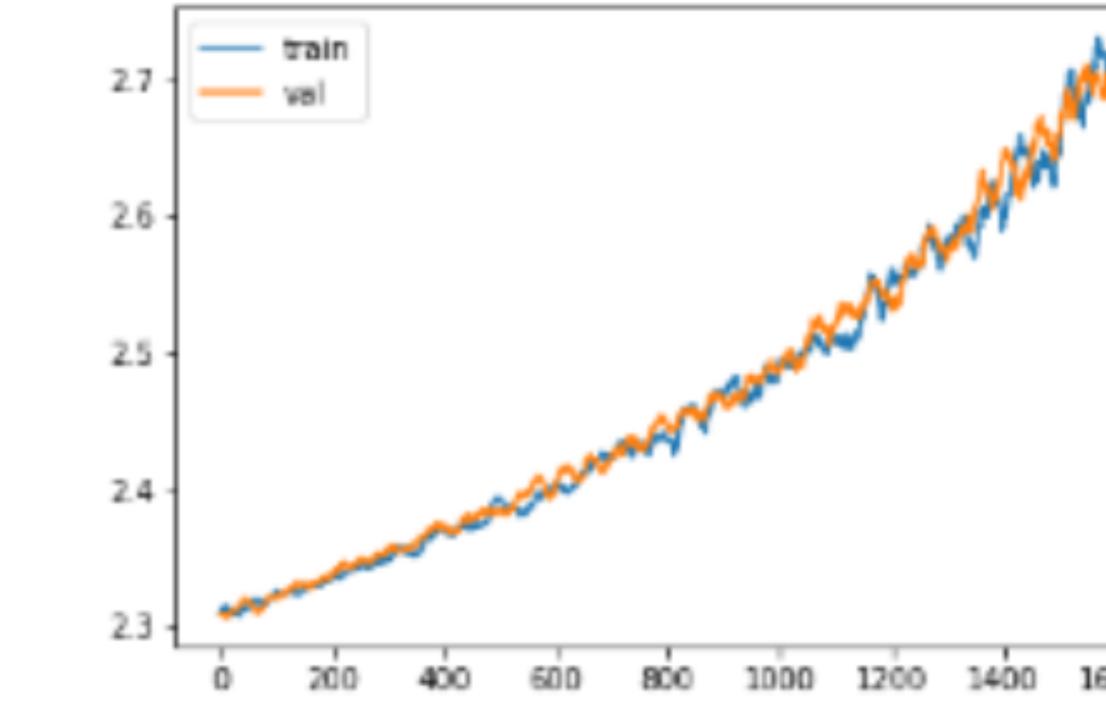
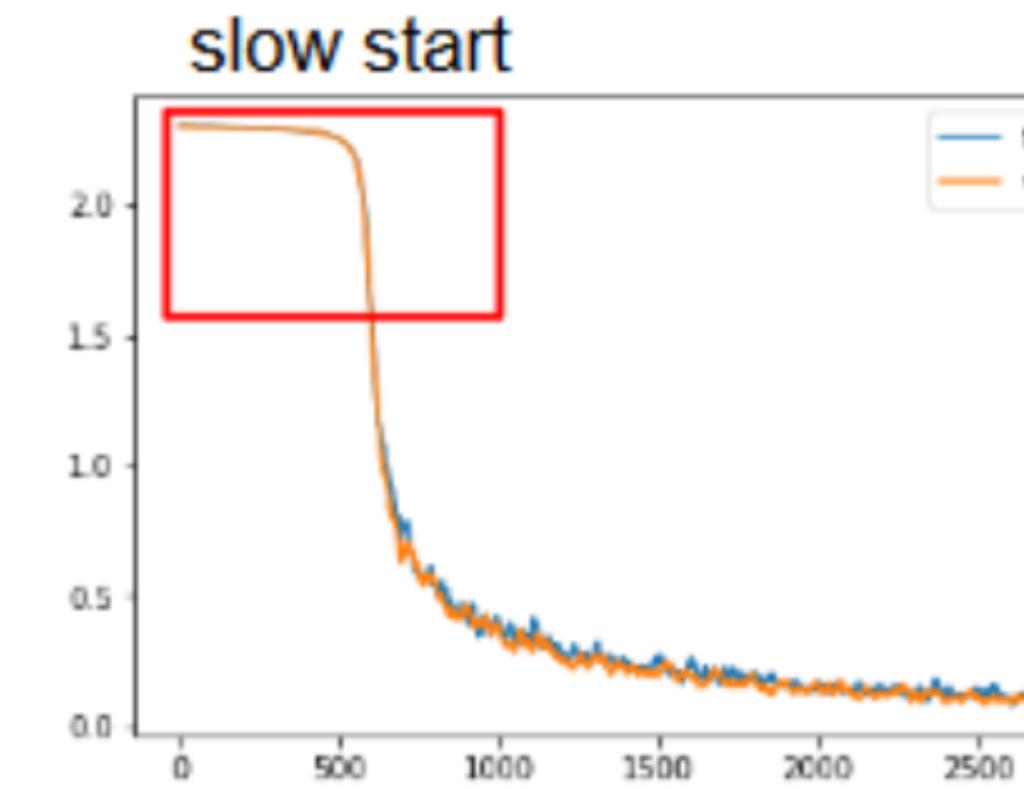
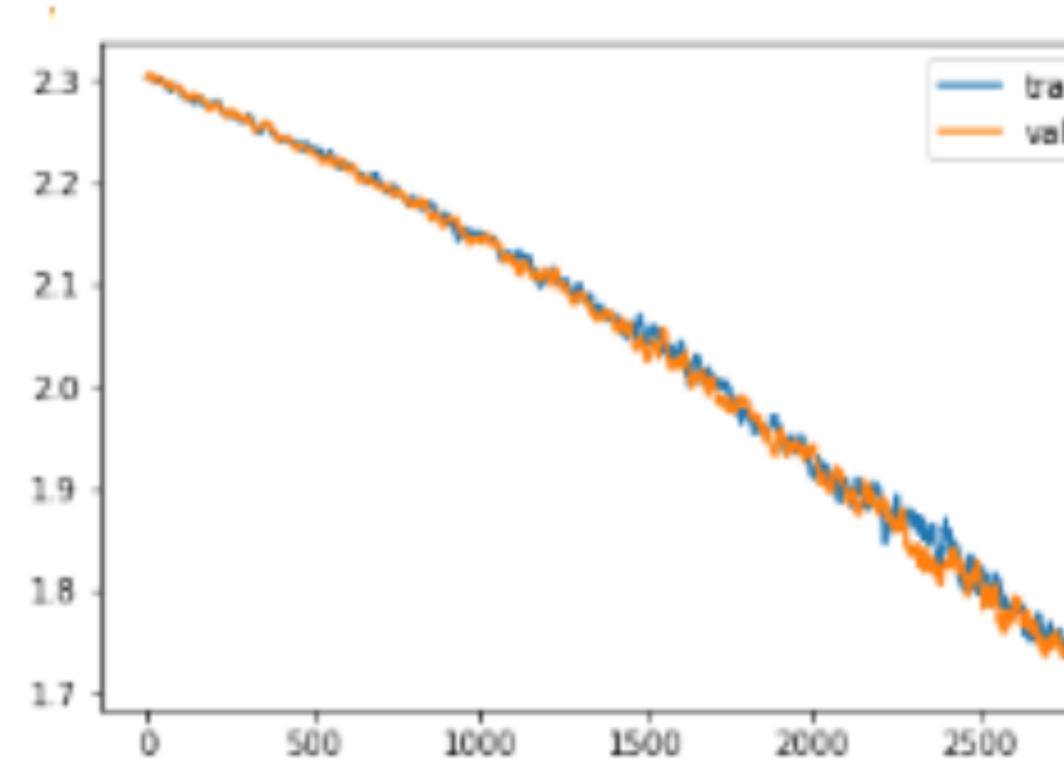
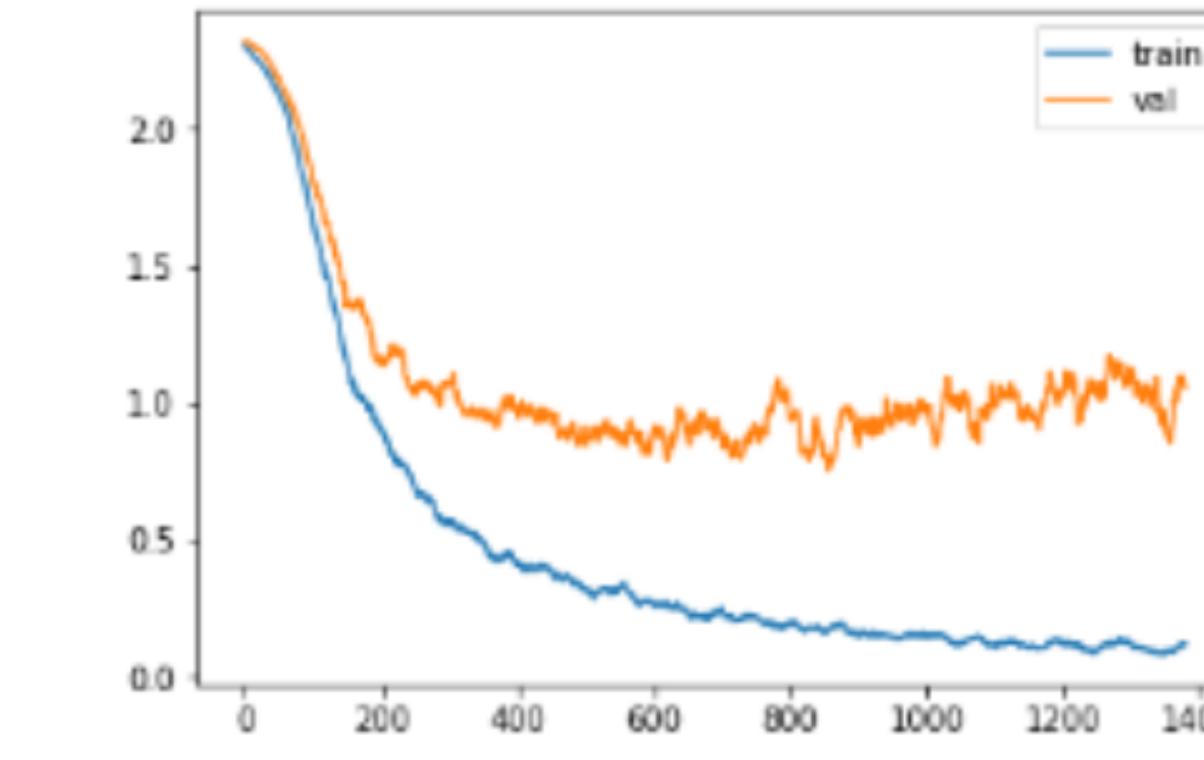
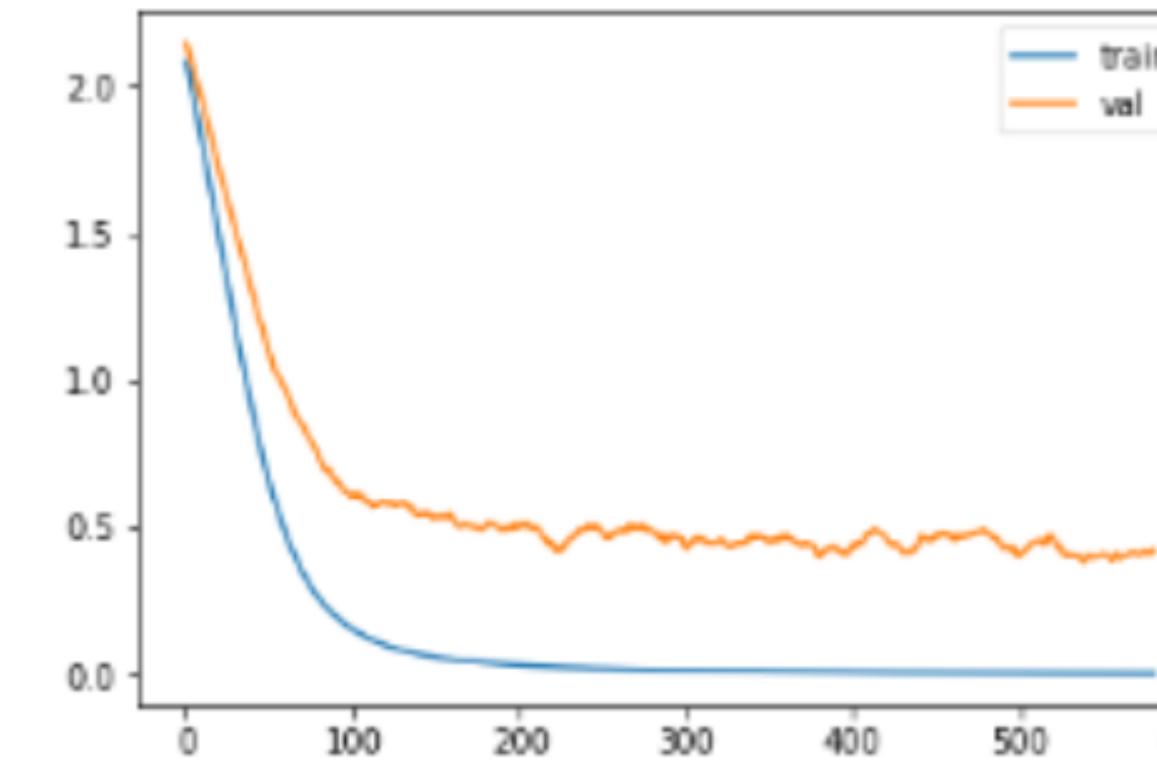
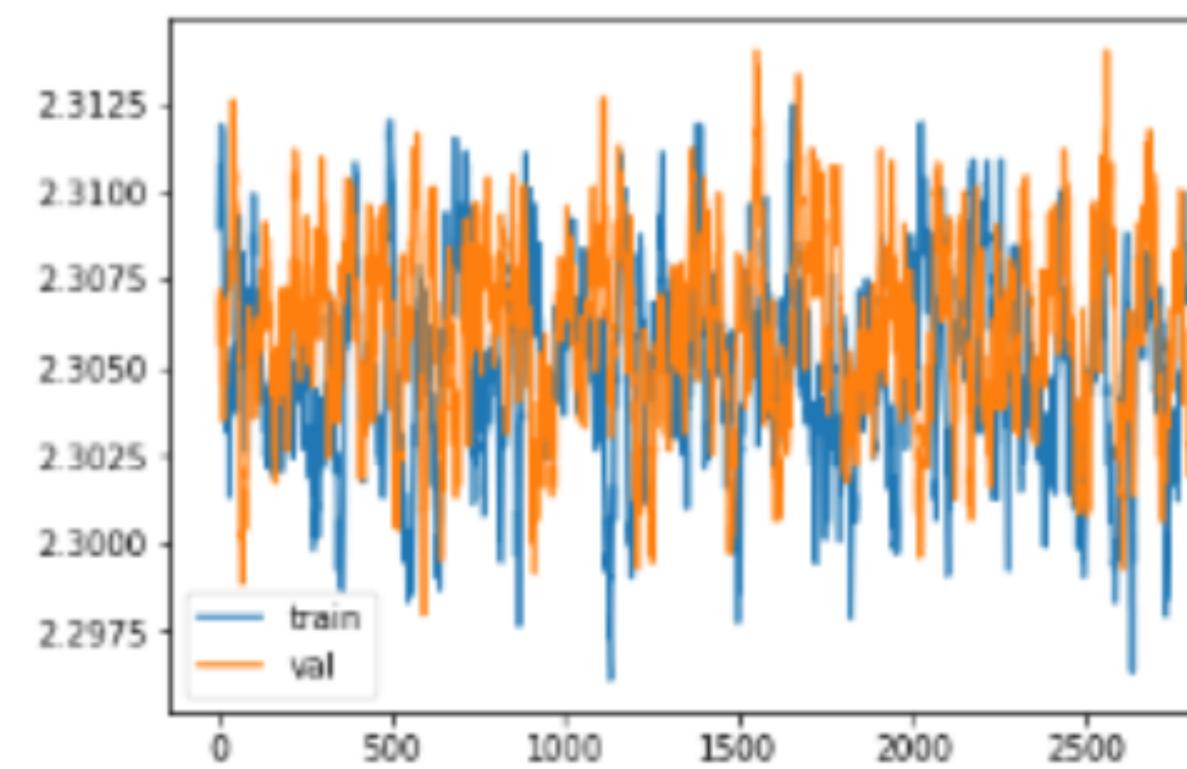
Poi?

Esatto, dobbiamo **valutare il modello (sui dati di test)**:

- Invece di visualizzare i dati come sulla gaussiana introduciamo due metriche nuove per valutare le performances del nostro modello
 - Grafico della loss sul train e sul test (va bene sia per regressione che classificazione)
 - Confusion Matric (va bene solo per classificazione)

Grafico di train e test loss

Una nuova metrica per capire se il nostro modello ha imparato



Valutiamo il modello: grafico dell'errore

MLP Per la classificazione

```
58 # Plot della perdita (Training e Validation)
59 train_loss = history.history['loss']
60 val_loss = history.history['val_loss']
61 epochs = range(1, len(train_loss) + 1)
62
63 plt.figure(figsize=(8, 6))
64 plt.plot(*args: epochs, train_loss, 'r-', label='Training Loss')
65 plt.plot(*args: epochs, val_loss, 'b--', label='Validation Loss')
66 plt.title('Andamento della Loss durante l\'addestramento')
67 plt.xlabel('Epoche')
68 plt.ylabel('Loss')
69 plt.legend()
70 plt.tight_layout()
71 plt.show()
```

Poi?

Esatto, dobbiamo **valutare il modello** (sui dati di test):

- Invece di visualizzare i dati come sulla gaussiana ne approfittiamo per introdurre due metriche nuove per valutare le performances del nostro modello
 - Grafico della loss sul train e sul test (va bene sia per regressione che classificazione)
 - Confusion Matric (va bene solo per classificazione)

Confusion Matrix

Una nuova metrica per capire se il nostro modello ha imparato

| | | PREDICTED | |
|--------|----------|-------------------------|-------------------------|
| | | NEGATIVE | POSITIVE |
| ACTUAL | NEGATIVE | TRUE NEGATIVES (TN) | FALSE POSITIVES (FP) |
| | POSITIVE | FALSE NEGATIVES (FN) | TRUE POSITIVES (TP) |

Valutiamo il modello: confusion matrix

MLP Per la classificazione

```
6  from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay  
  
79 # Predizioni sul test set  
80 y_pred_circles_probabilities = mlp_circles.predict(x_test_circles)  
81 y_pred_circles = (y_pred_circles_probabilities >= 0.5).astype(int)  
82  
83 # Matrice di confusione  
84 conf_matrix = confusion_matrix(y_test_circles, y_pred_circles)  
85 ConfusionMatrixDisplay(conf_matrix, display_labels=['Rosso', 'Blu']).plot(colorbar=False)  
86 plt.title("Matrice di confusione")  
87 plt.show()
```

Funziona il nostro modello? Cosa ci dicono le metriche che abbiamo analizzato?

**Funziona il nostro modello? Cosa ci dicono le metriche che
abbiamo analizzato?**

Come possiamo migliorare il nostro modello?

Ma quindi.. un MLP può fare tutto? Mettiamolo alla prova

Usiamo un dataset ancora più difficile



MNIST Dataset:

- Immagini delle **9 cifre** scritte da persone diverse
- **Obiettivo:** creare un modello in grado di classificare correttamente la cifra visualizzata
- Dataset disponibile su **tensorflow**

Caricamento e preprocessing di MNIST dataset

MLP Per la classificazione

```
6 # Caricamento e preprocessamento del dataset MNIST
7 (x_train_mnist, y_train_mnist), (x_test_mnist, y_test_mnist) = tf.keras.datasets.mnist.load_data()
8
9 # One-hot encoding delle etichette
10 y_train_mnist_enc = tf.keras.utils.to_categorical(y_train_mnist)
11 y_test_mnist_enc = tf.keras.utils.to_categorical(y_test_mnist)
12
13 # Normalizzazione dei pixel delle immagini
14 x_train_mnist_norm = x_train_mnist.astype(np.float32) / 255.0
15 x_test_mnist_norm = x_test_mnist.astype(np.float32) / 255.0
```

Creazione del modello MLP

MLP Per la classificazione

```
17 # Creazione del modello MLP
18 mnist_mlp = tf.keras.models.Sequential([
19     tf.keras.layers.Flatten(input_shape=(28, 28)), # Flatten per convertire l'immagine in un vettore
20     tf.keras.layers.Dense(units=128, activation='relu'), # Primo hidden layer
21     tf.keras.layers.Dropout(0.2), # Dropout per regolarizzazione
22     tf.keras.layers.Dense(units=10, activation='softmax') # Strato di output per classificazione multicl
23 ])
```

Compilazione del modello MLP

MLP Per la classificazione

```
25 # Compilazione del modello
26 lr = 0.001
27 loss_fn = tf.keras.losses.CategoricalCrossentropy()
28 mnist_mlp.compile(
29     optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
30     loss=loss_fn,
31     metrics=['accuracy']
32 )
```

Addestramento del modello MLP

MLP Per la classificazione

```
34 # Addestramento del modello
35 n_epochs = 25
36 mnist_mlp_history = mnist_mlp.fit(
37     x_train_mnist_norm, y_train_mnist_enc,
38     validation_split=0.2,
39     epochs=n_epochs,
40     verbose=1
41 )
```

Valutiamo il modello: grafico dell'errore

MLP Per la classificazione

```
43 # Plot delle perdite di training e validazione
44 train_loss = mnist_mlp_history.history['loss']
45 val_loss = mnist_mlp_history.history['val_loss']
46 epochs = range(1, len(train_loss) + 1)
47
48 plt.figure(figsize=(8, 6))
49 plt.plot(*args: epochs, train_loss, 'r-', label='Training Loss')
50 plt.plot(*args: epochs, val_loss, 'b--', label='Validation Loss')
51 plt.title('Andamento della Loss durante l\'addestramento')
52 plt.xlabel('Epoche')
53 plt.ylabel('Loss')
54 plt.legend()
55 plt.tight_layout()
56 plt.show()
```

Valutiamo il modello: confusion matrix

MLP Per la classificazione

```
62 # Predizioni sul test set
63 y_pred_mnist_prob = mnist_mlp.predict(x_test_mnist_norm)
64 y_pred_mnist = np.argmax(y_pred_mnist_prob, axis=1)
65
66 # Matrice di confusione
67 conf_matrix = confusion_matrix(y_test_mnist, y_pred_mnist)
68 ConfusionMatrixDisplay(conf_matrix, display_labels=np.arange(10)).plot(colorbar=True, cmap='viridis')
69 plt.title("Matrice di Confusione (MNIST)")
70 plt.show()
```