Iris Classification con Rete Neurale

Davide Ferrara

Luglio 2025

1 Introduzione

Il problema della classificazione dei fiori *Iris* consiste nell'identificare correttamente la specie di un fiore a partire da alcune sue caratteristiche misurabili. Questo compito richiede la capacità di riconoscere schemi complessi nei dati.

Per risolvere questo problema si utilizza una rete neurale a più strati (Multi-Layer Perceptron), un modello in grado, tramite un processo di addestramento, di apprendere le relazioni tra le caratteristiche dei fiori e le rispettive specie di appartenenza.

L'obiettivo del progetto è esplorare come questo modello, una volta addestrato, riesca a generalizzare su nuovi dati e a classificare correttamente anche i fiori mai visti prima.

2 Il Dataset

I fiori considerati appartengono alla specie *Iris* e sono descritti da quattro attributi:

- Lunghezza del sepalo
- Larghezza del sepalo
- Lunghezza del petalo
- Larghezza del petalo

Le tre specie di fiori analizzate sono:

- **Iris Setosa**: la specie più piccola, altezza tra 30 e 50 cm.
- **Iris Versicolor**: specie intermedia, altezza tra 50 e 80 cm.
- **Iris Virginica**: la specie più alta, altezza tra 60 e 100 cm.

Il dataset utilizzato è composto da 150 righe nel seguente formato:

5.1,3.5,1.4,0.2,Iris-setosa

La suddivisione dei dati è la seguente:

- **Training set**: 150 righe (50 per ciascuna specie)
- **Test set**: 30 righe (10 per ciascuna specie)

La divisione è stata effettuata per utilizzare un approccio di apprendimento supervisionato.

3 Il Modello: Perceptrone Multistrato

Per affrontare il problema è stato utilizzato un *Perceptrone Multistrato* (Multi-Layer Perceptron), un tipo di rete neurale artificiale ispirato al funzionamento dei neuroni biologici.

Un Perceptrone semplice elabora input e genera output, ma non è in grado di gestire problemi non linearmente separabili. L'estensione multistrato, invece, permette di affrontare problemi complessi grazie alla presenza di uno o più strati nascosti.

La struttura utilizzata è la seguente:

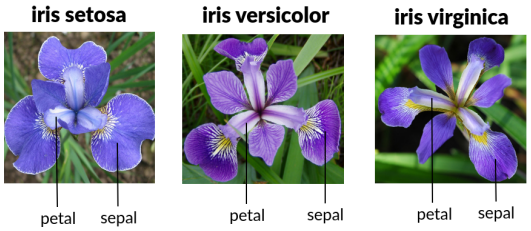

Figure 1: Esempio di fiore del genere Iris

- **Strato di Input**: 4 neuroni, uno per ciascun attributo del fiore.
- **Strato Nascosto**: 8 neuroni per elaborare le informazioni e apprendere le relazioni complesse.
- **Strato di Output**: 3 neuroni, uno per ciascuna specie da classificare.

4 Implementazione della Rete Neurale

L'implementazione della rete è contenuta nel file `NeuralNetwork.py`.

4.1 Costruttore della Rete

```
class NeuralNetwork:
    def __init__(
        self,
        num_hidden,
        num_outputs,
        dataset_path,
        num_features,
        epochs,
        learning_rate,
        error_threshold,
        momentum,
        test_dataset_path=None,
    ):

    # Parametri istanza classe
    dataset_path = "dataset/iris.data"
    test_dataset_path = "dataset/test_set.data"
    num_features = 4
    num_hidden = 16
    num_outputs = 3
    epoch = 2000
    learning_rate = 0.0001
    error_threshold = 1e-5
    momentum = 0.9
```

I parametri principali sono:

- **num_features**: Numero di attributi del fiore (4).
- **num_hidden**: Numero di neuroni nello strato nascosto (8).
- **num_outputs**: Numero di neuroni di output (3 classi).
- **epoch**: Massimo numero di epoche di addestramento (2000).
- **learning_rate**: Tasso di apprendimento (0.0001).

- **error_threshold**: Soglia di errore per terminare anticipatamente l'addestramento.
- **momentum**: Fattore che aiuta a stabilizzare l'aggiornamento dei pesi.

4.2 Inizializzazione dei pesi e bias

I pesi vengono inizializzati con un valore compreso tra **-1 e 1**, mentre tutti i bias a **0**.

```
def init_weights(self, row, col):
    return np.random.uniform(-1, 1, (row, col))

def init_bias(self, row):
    return np.zeros((row, 1))
```

4.3 Feed Forward

Il processo di **Feed Forward** rappresenta il passaggio in cui i dati in ingresso vengono elaborati dalla rete neurale per produrre un output. Durante questa fase, gli input attraversano i vari strati della rete, subendo trasformazioni attraverso operazioni matematiche come prodotti matriciali e funzioni di attivazione.

```
def feed_forward(self, input):
    self.hidden = np.dot(self.weights_ih, input) + self.bias_h
    self.hidden = self.relu(self.hidden)

    self.outputs = np.dot(self.weights_ho, self.hidden) + self.bias_o
    self.outputs = self.softmax(self.outputs)

    return self.outputs
```

4.4 Training della rete

La fase di **Training** rappresenta il processo di apprendimento della rete neurale, in cui i pesi e i bias vengono progressivamente aggiornati per ridurre l'errore tra le previsioni della rete e i risultati attesi.

Il procedimento prevede:

1. Per ogni epoca (iterazione):
   - Si esegue il `feed_forward` per ottenere le previsioni della rete sui dati di training.
   - Si calcola l'errore tra le previsioni e i valori attesi, utilizzando la funzione `mean_squared_error`.
   - Si applica la retropropagazione dell'errore (`backprop`) per aggiornare i pesi e i bias della rete.
   - Si registra l'errore relativo al training set.
2. Se è presente un set di test, si calcola l'errore anche su questo, per monitorare la capacità della rete di generalizzare.

```
def train(self, args=""):
    for epoch in range(self.epochs):
        a = self.feed_forward(self.X.T)
        y = self.targets.T

        self.backprop(a, y)
        error = self.mean_squared_error(a, y)
        self.train_loss.append(error)

        if self.test_dataset_path is not None:
            a_test = self.feed_forward(self.X_test.T)
```

```
            y_test = self.targets_test.T
            error_test = self.mean_squared_error(a_test, y_test)
            self.test_loss.append(error_test)
        ...
```

4.5 Salvataggio e caricamento del modello

Tramite il modulo `pickle` è possibile serializzare l'intero modello allenato, per importarlo successivamente in un altro programma Python.

```
@staticmethod
def load(model_path):
    with open(model_path, "rb") as f:
        nn = pickle.load(f)
    return nn

def save(self, model_name="model.pkl"):
    with open(model_name, "wb") as f:
        print(f"Model has been saved as: {model_name}!")
        pickle.dump(self, f)
```

4.6 One-hot Encoding e Output della rete

Per confrontare l'output della rete con i dati reali è stato utilizzato il metodo *one-hot encoding*, che trasforma le classi testuali in vettori numerici binari.

Esempio di mappatura:

```
Hot Vector = {
    Iris-Setosa: [1, 0, 0],
    Iris-Versicolor: [0, 1, 0],
    Iris-Virginica: [0, 0, 1]
}
```

Esempio di output della rete:

```
Array of probs: [0.99994 0.00006 0.0]
Prediction: Iris-Setosa con confidenza del 99.99%
```

5 Risultati

Di seguito vengono mostrati i grafici principali ottenuti durante l'addestramento:

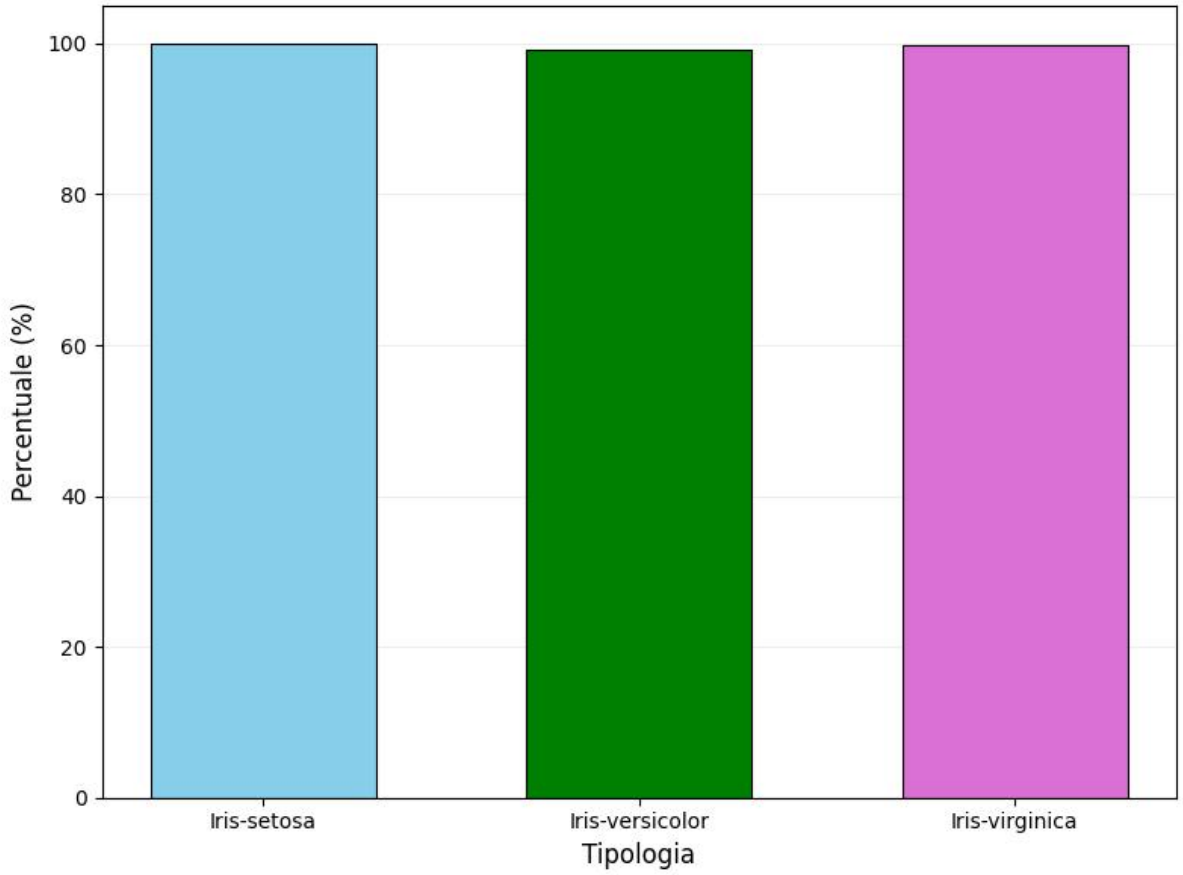

Figure 2: Grafico delle confidenze per ciascuna classe di fiore

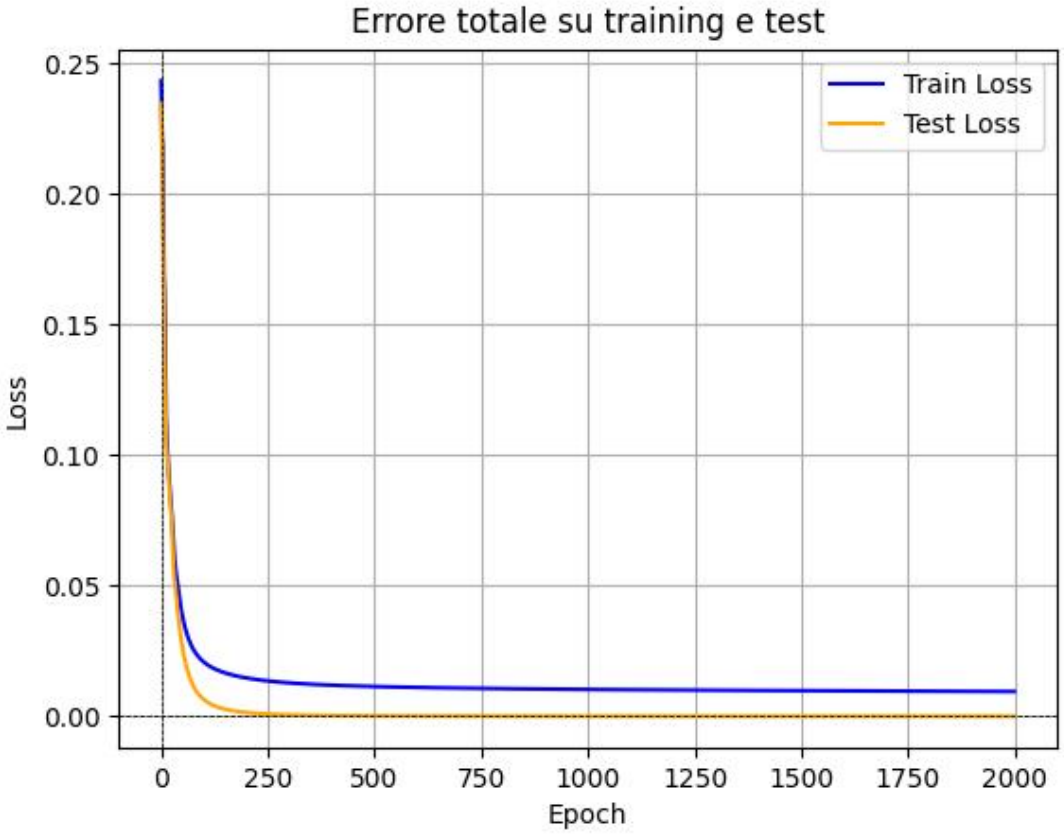

Figure 3: Andamento dell'errore su training set e test set

6 Conclusioni

Il codice sviluppato consente di addestrare efficacemente un Perceptrone Multistrato per la classificazione dei fiori Iris.

L'analisi dei risultati mostra:

- Accuratezza vicina al 100% su tutte le classi.
- Rapida discesa dell'errore nelle prime epoche, seguito da stabilizzazione.
- Assenza di fenomeni evidenti di overfitting.

Nel complesso, il modello si è dimostrato adeguato alla risoluzione del problema, confermando la validità dell'approccio basato su reti neurali per compiti di classificazione non lineari.