1. Introduzione alle Reti Neurali e MLP

Cosa sono le Reti Neurali

Le reti neurali artificiali sono modelli computazionali ispirati al funzionamento del cervello umano, composti da unità di elaborazione interconnesse (neuroni) organizzate in strati. Questi modelli sono progettati per riconoscere pattern e relazioni complesse nei dati.

Multi-layer Perceptron (MLP)

Un Multi-layer Perceptron è un tipo di rete neurale feed-forward composta da almeno tre strati:

- **Strato di input**: riceve i dati originali
- **Strati nascosti** (uno o più): elaborano le informazioni
- **Strato di output**: produce la previsione finale

Ogni neurone in un MLP utilizza una funzione di attivazione non lineare (come ReLU, Sigmoid o Tanh) per trasformare i dati in ingresso e passarli allo strato successivo.

2. Il Dataset MNIST

MNIST è un database di riferimento nel campo del machine learning che contiene immagini di cifre scritte a mano (da 0 a 9). È ampiamente utilizzato per testare algoritmi di classificazione di immagini e riconoscimento di pattern.

Caratteristiche principali:

- 70.000 immagini in totale (60.000 per training, 10.000 per test)
- Ogni immagine è in scala di grigi 28×28 pixel (784 pixel totali)
- Valori dei pixel compresi tra 0 (nero) e 255 (bianco)

3. Analisi del Notebook

Caricamento del Dataset

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784')

print("Campi del dataset\n", mnist.keys())
print(mnist['DESCR'])
```

Questa sezione carica il dataset MNIST dalla libreria scikit-learn utilizzando la funzione `fetch_openml` e stampa la descrizione del dataset per comprenderne la struttura.

Esplorazione dei Dati

```
import numpy as np
import matplotlib.pyplot as plt

y = mnist.target
X = mnist.data
print(X.shape)  # 70000 campioni di 784 pixel = 28x28 in gradazioni di
grigio
```

Qui vengono estratti i dati (`X`) e le etichette (`y`) dal dataset. La forma di `X` è (70000, 784), indicando 70.000 immagini, ciascuna rappresentata da 784 pixel.

Visualizzazione di un Campione

```
plt.imshow(X.loc[0].values.reshape(28, 28), cmap='gray')
plt.title(f"Label: {y[0]}")
plt.axis("off")  # rimuove gli assi
plt.show()
```

Questo codice visualizza la prima immagine del dataset, ridimensionandola dalla forma piatta (784) alla forma originale (28×28) e mostrandola in scala di grigi.

Normalizzazione dei Dati

```
X = X / 255   # normalizziamo i dati ottenendo valori tutti compresi tra 0 e
1
```

La normalizzazione è una fase cruciale nel preprocessing dei dati per le reti neurali. Dividendo ogni valore per 255 (il valore massimo possibile), si ottengono valori compresi tra 0 e 1, facilitando la convergenza dell'algoritmo di apprendimento.

Separazione in Training e Test Set

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000],
y[60000:]
```

Come indicato nella descrizione, i primi 60.000 campioni sono usati per l'addestramento e i restanti 10.000 per il test.

Addestramento del Modello MLP

```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=20, verbose=True,
                    random_state=1, learning_rate_init=0.1)
mlp.fit(X_train, y_train)

print("Training set score: %f" % mlp.score(X_train, y_train))
print("Test set score: %f" % mlp.score(X_test, y_test))
```

Qui viene creato e addestrato un modello MLP con le seguenti caratteristiche:

- Un singolo strato nascosto con 50 neuroni (`hidden_layer_sizes=(50,)`)
- Massimo 20 epoche di addestramento (`max_iter=20`)
- Rapporto dettagliato durante l'addestramento (`verbose=True`)
- Tasso di apprendimento iniziale di 0.1 (`learning_rate_init=0.1`)

Dopo l'addestramento, il modello viene valutato sia sul training set che sul test set.

Ottimizzazione del Numero di Epoche

```
mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=5, verbose=True,
                    random_state=1, learning_rate_init=0.1)
mlp.fit(X_train, y_train)
```

Qui il notebook mostra come l'apprendimento possa arrestarsi prima del numero massimo di iterazioni se l'errore comincia ad aumentare (criterio di early stopping). Il parametro `max_iter` viene ridotto a 5 in base all'osservazione che l'errore raggiunge il minimo dopo 5 epoche.

Confronto tra Algoritmi di Ottimizzazione

```
mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=20, verbose=True,
                    random_state=1, learning_rate_init=.1, solver="sgd")
mlp.fit(X_train, y_train)
```

Questa sezione confronta l'algoritmo di ottimizzazione predefinito (Adam) con lo Stochastic Gradient Descent (SGD). Il parametro `solver="sgd"` specifica l'uso dell'algoritmo SGD.

Analisi dell'Overfitting

```
start = 3   # numero iniziale di epoche
stop = 30   # numero finale di epoche
passo = 2   # passo incrementale

vEpochs = np.arange(start, stop+1, passo)
```

```
yTrain = []  # lista dell'accuratezza sui dati di train per disegnare un
grafico
yTest = []   # lista dell'accuratezza sui dati di test per disegnare un
grafico

for e in vEpochs:
    mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=e,
                        random_state=1, learning_rate_init=.1, solver="sgd")
    mlp.fit(X_train, y_train)
    accTrain = mlp.score(X_train, y_train)
    yTrain.append(accTrain)
    accTest = mlp.score(X_test, y_test)
    yTest.append(accTest)
```

Questo ciclo addestra modelli con diversi numeri di epoche (da 3 a 30) e registra l'accuratezza sia sul training set che sul test set per ogni modello.

```
plt.plot(vEpochs, yTrain, c='b', label='Train')
plt.plot(vEpochs, yTest, c='g', label='Test')
plt.xlabel("epoche", fontsize=15)
plt.ylabel("accuratezza", fontsize=15)
plt.tick_params(axis='both', labelsize='15')
plt.legend(fontsize=12)
```

Il grafico risultante mostra l'andamento dell'accuratezza al variare del numero di epoche. È possibile identificare il numero ottimale di epoche come il punto in cui l'accuratezza sul test set inizia a diminuire mentre quella sul training set continua ad aumentare (segno di overfitting).

Test del Modello su Nuove Immagini

```
import cv2

# carica l'immagine in scala di grigi e ridimensionala a 28x28
image = cv2.imread("3.png", cv2.IMREAD_GRAYSCALE)
image = cv2.resize(image, (28, 28))
# normalizza i valori
image = image / 255
# converte l'immagine in un vettore 1x784
x_sample = image.flatten().reshape(1, -1)

y_sample = mlp.predict(x_sample)
print(y_sample)
```

L'ultima parte del notebook mostra come utilizzare il modello addestrato per classificare una nuova immagine. L'immagine viene:

1. Caricata in scala di grigi
2. Ridimensionata a 28×28 pixel
3. Normalizzata (divisa per 255)
4. Appiattita in un vettore 1×784
5. Sottoposta al modello per la previsione

4. Concetti Chiave di Machine Learning

Overfitting

Si verifica quando un modello impara troppo bene i dati di addestramento, memorizzandoli invece di generalizzare. Si riconosce quando l'accuratezza sul training set continua a migliorare mentre quella sul test set peggiora.

Algoritmi di Ottimizzazione

- **Adam (Adaptive Moment Estimation)**: algoritmo che combina il momentum e la normalizzazione adattiva del learning rate. È spesso l'opzione predefinita perché efficiente e facile da configurare.
- **SGD (Stochastic Gradient Descent)**: aggiorna i parametri del modello un campione alla volta, permettendo aggiornamenti più frequenti e potenzialmente una convergenza più rapida.

Normalizzazione

Processo di riscalamento dei dati in un intervallo standard (spesso [0,1]). È importante per:

- Accelerare la convergenza durante l'addestramento
- Prevenire che alcune caratteristiche dominino altre a causa della loro scala
- Migliorare la stabilità numerica

Validation Set

Anche se non esplicitamente utilizzato nel notebook, sarebbe una buona pratica separare un validation set dal training set per poter ottimizzare gli iperparametri senza "contaminare" il test set.

5. Possibili Estensioni dell'Esercizio

Architetture Più Complesse

Sperimentare con diverse architetture di rete:

- Più strati nascosti
- Diverso numero di neuroni per strato

- Diverse funzioni di attivazione

Regolarizzazione

Introdurre tecniche di regolarizzazione per combattere l'overfitting:

- L1 o L2 (parametri `alpha` e `penalty` in MLPClassifier)
- Dropout (non direttamente disponibile in scikit-learn, ma implementabile con altre librerie come TensorFlow o PyTorch)

Ottimizzazione degli Iperparametri

Utilizzare tecniche di ricerca sistematica come:

- Grid Search
- Random Search
- Cross-validation

Visualizzazione e Interpretazione

- Visualizzare la matrice di confusione per identificare classi problematiche
- Visualizzare i pesi appresi per capire quali caratteristiche sono più importanti
- Implementare tecniche di interpretabilità come LIME o SHAP

6. Relazione con il CSV "marziani"

Il file CSV menzionato nella cartella "marziani" probabilmente contiene un dataset diverso per un esercizio simile. L'approccio generale sarebbe simile:

1. Caricare e analizzare il dataset
2. Preprocessare i dati (normalizzazione, codifica delle features categoriche)
3. Dividere in training e test set
4. Addestrare un modello MLP
5. Valutare le performance
6. Ottimizzare gli iperparametri

La differenza principale sarebbe nella natura dei dati e potenzialmente nella definizione del problema (classificazione o regressione).

Conclusioni

Questo notebook fornisce un'ottima introduzione alle reti neurali artificiali e alla loro implementazione pratica utilizzando scikit-learn. Gli studenti imparano non solo a costruire e addestrare un modello, ma anche a valutarne le prestazioni e a riconoscere problemi comuni come l'overfitting.

Le competenze acquisite possono essere applicate a una vasta gamma di problemi di machine learning, modificando l'architettura della rete e adattando il preprocessing dei dati alle specifiche caratteristiche del problema affrontato.