Pillole di teoria

- MLP (Multi-Layer Perceptron)
- MNIST (Immagini di riconoscimento da 0 a 9)
    - Dati in frame con un certo numero di px e dimensioni

Codice

```
# Reti neurali che apprende da un DB
# scikit-learn --> DB per recupero (fetch) dal db MNIST

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

# 1. Esplorazione dei dati (f(x) - data = y (output = target))
import numpy as np
import matplotlib.pyplot as plt

y = mnist.target
X = mnist.data

# 2. Normalizzazione (gaussiana) -> Restringi ad un range (intervallo) ->
[0, 1]
X = X / 255

# Prendiamo un certo numero di campioni
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000],
y[60000:]

# 3. Addestramento
# Classificatore -> MLPClassifier (scikit-learn)
# Addestrarlo su (x,y) sapendo gli input nascosti, rendendolo casuale
(normalizzare)

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)

# Score -> Distanza tra data (x) e target (y)
print("Training set score: %f" % mlp.score(X_train, y_train))
print("Test set score: %f" % mlp.score(X_test, y_test))

# Restringiamo ad un certo numero di epoche = n.di iterazioni)

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

```python
mlp.fit(X_train, y_train)

# Accenno ad altri algoritmi di ottimizzazione (sgd)
# Stocastico -> Probabilità (aleatorio / random)
# Gradient descent -> Derivate parziali (Analisi 2)
# -> Derivata su x / Derivata su y -> Funzioni a più variabili

# 4. Analisi dell'overfitting (cambiando algoritmo)

# Dato un numero di epoche, cerchiamo di eseguire il classificatore e
controllare meglio lo score durante le iterazioni

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)

## Accenno alla libreria cv2 -> Immagini

# Immagini -> Matrice (m x n)
# [][][][]
# [][][][]
# [][][][]
# Array -> (1 x n) [][][][]

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)
```