

UNIVERSITÀ DI BERGAMO

RELAZIONE PROGETTO

ARTIFICIAL INTELLIGENCE

---

# Reti neurali in python

---

*Autore:*

Dario SARDI

*Supervisore:*

Francesco TROVÒ

22 Aprile 2019



## Abstract

L'obiettivo del progetto è quello di creare da zero una rete neurale in python senza sfruttare librerie già esistenti.  
Si è creato dapprima un percettrone e successivamente una rete neurale con un solo hidden layer.

## 1 Percettrone

Per iniziare e prender pratica con eventuali librerie matematiche è stato creato un percettrone, un neurone in grado di compiere semplici scelte binarie. In quanto classificatore lineare il dataset per il percettrone consiste in una nuvola di punti posizionati randomicamente e pre-classificati in due categorie in base a una funzione lineare stabilita.

```
1 def function (x):
2     m=-1/3
3     c=0.5
4     return m*x+c
5
6 def genFunction (x,y):
7     if y>function(x): return 1
8     else: return -1
9
10
11
12 class point:
13     def __init__ (self ,x,y,b):
14         self.pos=[x,y,b]
15         self.group=genFunction(x,y)
```

In questo modo inizializzando un punto con posizione randomica, la sua appartenenza alle classi  $\{1,-1\}$  viene determinata dalla sua posizione relativa alla funzione lineare.

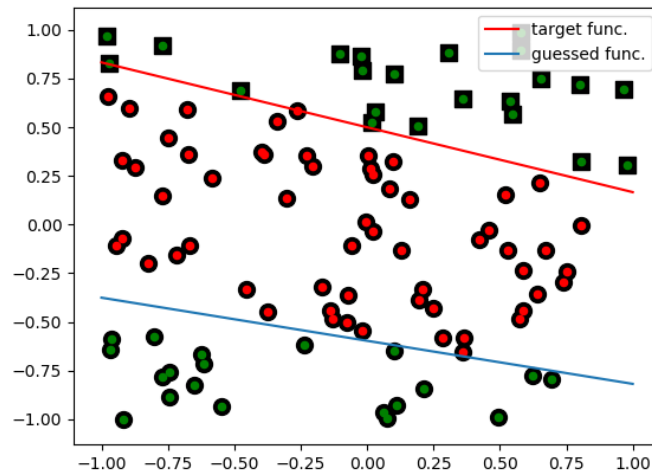


Figure 1: classificazione prima del training

Nella rappresentazione grafica (figura 1 ) le due classi son rappresentate con quadrati e cerchi colorati di rosso o verde se sono classificati rispettivamente in modo corretto o errato. La funzione effettiva di classificazione è la retta di colore rosso, in blu è presente quella stimata (inizialmente con pesi random). Il programma prosegue con dei cicli di training.

```
1 trainCycle (population , perc , 5 )
```

Per questo esempio il percettrone viene sottoposto a 5 cicli di training.

```
1 def TrainCycle (populationG , pa , number) :
2     for i in range (number) :
3         for i in range (50) :
4             dot = choice (population)
5             pa.train (dot.pos , dot.group)
```

La funzione **TrainCycle** seleziona 50 elementi randomici (funzione choice in python estrae casualmente da una collezione ) su 100 e li usa come train set.

```

1 def train (self ,inputs ,target):
2     guessed = self.guess(inputs)
3     error = target-guessed
4     for i in range(0,len(self.weights)):
5         self.weights[i] += error*inputs[i]*self.lr

```

La funzione di train del percettrone rivaluta i propri pesi secondo la formula:

$$\underline{w} = \underline{w} + \underline{err} * \underline{input} * \underline{learnRate}$$

L'output ottenuto dopo i cicli di training mostra una classificazione corretta.

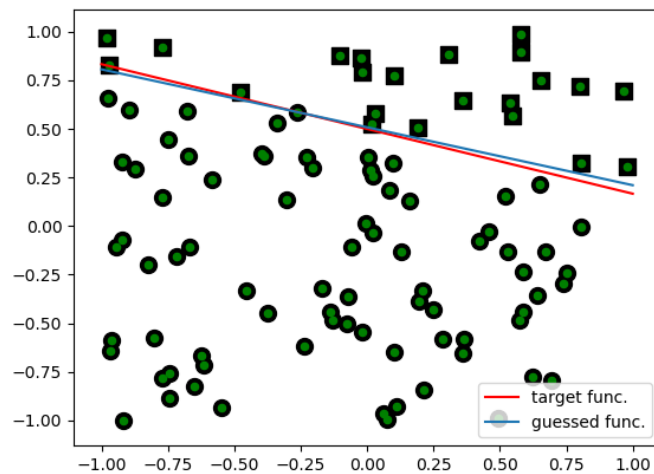


Figure 2: classificazione dopo il training

## 2 Doodle classifier

Lo scopo del progetto è stato ottenere un classificatore in grado di distinguere con una buona precisione a cosa somigliasse di più un disegno rispetto a altri riferimenti passati in precedenza.

Nel caso specifico sono stati utilizzati tre dataset contenenti disegni di nuvole, uccelli e della torre eiffel.

### 2.1 Neural network

Il primo passo è stato creare una classe per la rete neurale dotata di un solo hidden layer (*basic1NN/neuralNet.py*).

```
1 def __init__(self, input_size, hidden_size, out_size):
2     self.input = []
3     self.iS = int(input_size)
4     self.oS = int(out_size)
5     self.weightsI = np.random.random((hidden_size,
6         ↪ input_size))*2-1
7     self.weightsO = np.random.random((out_size,
8         ↪ hidden_size))*2-1
9     self.bias_h = np.random.random((hidden_size, 1))*2-1
10    self.bias_o = np.random.random((out_size, 1))*2-1
11    self.lr = 0.1
12    self.output = np.zeros(out_size)
```

Sono state implementate successivamente le funzioni di feed forward e di backward propagation

```

1 #funzione di feedforward
2 def ff(self):
3     self.hidden = sigmoid(np.dot(self.weightsI , self.
        ↪ input)+self.bias_h)
4     self.output = sigmoid(np.dot(self.weightsO , self.
        ↪ hidden)+self.bias_o)

```

Le operazioni effettuate sono rappresentati nella figura 3 e 4 escludendo la sigmoide usata come funzione di attivazione

$$\begin{array}{c} \left[ \begin{array}{c} \text{weightsI} \end{array} \right] \\ \text{[hidden\_size,input\_size]} \end{array} \times \begin{array}{c} \left[ \begin{array}{c} \text{input} \end{array} \right] \\ \text{[input\_size,1]} \end{array} + \begin{array}{c} \left[ \begin{array}{c} \text{bias\_h} \end{array} \right] \\ \text{[hidden\_size,1]} \end{array} = \begin{array}{c} \left[ \begin{array}{c} \text{hidden} \end{array} \right] \\ \text{[hidden\_size,1]} \end{array}$$

Figure 3: rappresentazione matriciale degli argomenti della classe usati nel calcolo dei valori dell'hidden layer

$$\begin{array}{c} \left[ \begin{array}{c} \text{weightsO} \end{array} \right] \\ \text{[out\_size,hidden\_size]} \end{array} \times \begin{array}{c} \left[ \begin{array}{c} \text{hidden} \end{array} \right] \\ \text{[hidden\_size,1]} \end{array} + \begin{array}{c} \left[ \begin{array}{c} \text{bias\_o} \end{array} \right] \\ \text{[out\_size,1]} \end{array} = \begin{array}{c} \left[ \begin{array}{c} \text{output} \end{array} \right] \\ \text{[out\_size,1]} \end{array}$$

Figure 4: rappresentazione matriciale degli argomenti della classe usati nel calcolo dell'output

```

1 #funzione di backward propagation
2 def backprop( self ):
3     out_err = self.y-self.output
4     gradiente_o = (sigmoidD(self.output)*out_err)*self.
        ↳ lr
5     delta_ho = np.dot(gradiente_o , self.hidden.T)
6     self.weightsO+=delta_ho
7     self.bias_o+=gradiente_o
8     gradiente_h = sigmoidD(self.hidden)*np.dot(self.
        ↳ weightsO.T, out_err)*self.lr
9     delta_ih = np.dot(gradiente_h , self.input.T)
10    self.weightsI += delta_ih
11    self.bias_h += gradiente_h

```

La funzione di backward utilizza la formula:

$$out\_err = target - output$$

$$\nabla_o = (\sigma'(output) * (out\_err)) * learningRate$$

$$\Delta_o = \nabla_o * hidden^T$$

$$weightsO = weightsO + \Delta_o$$

$$bias_o = bias_o + \nabla_o$$

$$\nabla_h = (\sigma'(hidden) * (weightsO^T * out\_err)) * learningRate$$

$$\Delta_i = \nabla_h * input^T$$

$$weightsI = weightsI + \Delta_i$$

$$bias_h = bias_h + \nabla_h$$

Per velocizzare l'esecuzione del codice sono state inserite delle funzioni per importare/esportare ( importPar(),export() ) i pesi (weightsI,weightsO,bias\_h,bias\_o)in modo da non dover fare training ogni volta prima di eseguire il codice.

## 2.2 Main file

Nel file principale ( *./googleDoodle.py* ) se la rete neurale non fosse già pronta si può impostare la variabile TRAIN per inizializzare il dataset e eseguire le funzioni di train.

### 2.2.1 Training

Viene effettuato il training su 3 differenti dataset di disegni. Le immagini sono 28x28 pixel e sono salvati in file di numpy (libreria di python).

```
1 if TRAIN:
2     clouds=np.load("dataset/cloud.npy")
3     ...
4     splitClouds=int(120265*0.75)
5     TrainClouds,TestClouds = np.split(clouds,[
6         ↪ splitClouds])
7     ...
```

Il dataset delle nuvole ad esempio viene caricato e diviso al 75% (120265 è il numero totale di samples nel dataset delle nuvole) in due per il training e testing rispettivamente.

```
1 import basicLNN.neuralNet as nn
2 oracle = nn.NeuralNet(784,64,3)
```

Viene creata una rete neurale con un input di 784 (pixel totali dell'immagine), 64 nodi nell'hidden layer e un output di 3 ( per i 3 dataset utilizzati).



```

1  if TRAIN:
2      trainingSet = []
3      print("generating training set")
4      for i in range(splitBirds-1):
5          x=np.reshape(TrainBirds[i]/255.0,(784,1))
6          y=np.reshape(np.array([1,0,0]),(3,1))
7          trainingSet.append([x,y])
8      for i in range(splitClouds-1):
9          x=np.reshape(TrainClouds[i]/255.0,(784,1))
10         y=np.reshape(np.array([0,1,0]),(3,1))
11         trainingSet.append([x,y])
12     for i in range(splitEiffel-1):
13         x=np.reshape(TrainEiffel[i]/255.0,(784,1))
14         y=np.reshape(np.array([0,0,1]),(3,1))
15         trainingSet.append([x,y])
16     r = random.SystemRandom()
17     r.shuffle(trainingSet)
18     for el in trainingSet:
19         oracle.trainOnce(el[0], el[1])
20     oracle.export()
21 else:
22     oracle.importPar()

```

Le immagini del train set vengono quindi prese singolarmente, convertite in vettori con valori da 0 a 1, associati alla terna corrispondente all'immagine (e.g. 0,1,0 per i disegni di nuvole) e inseriti in un database unico che verrà poi mescolato per avere un training uniforme.

### 2.2.2 Guessing

Il programma prende in input il file `myInput.png` posizionato nella stessa cartella di `googleDoodle.py`, lo converte in un formato compatibile e tramite il metodo `answer()` della classe `NeuralNetwork` ottiene la classificazione.

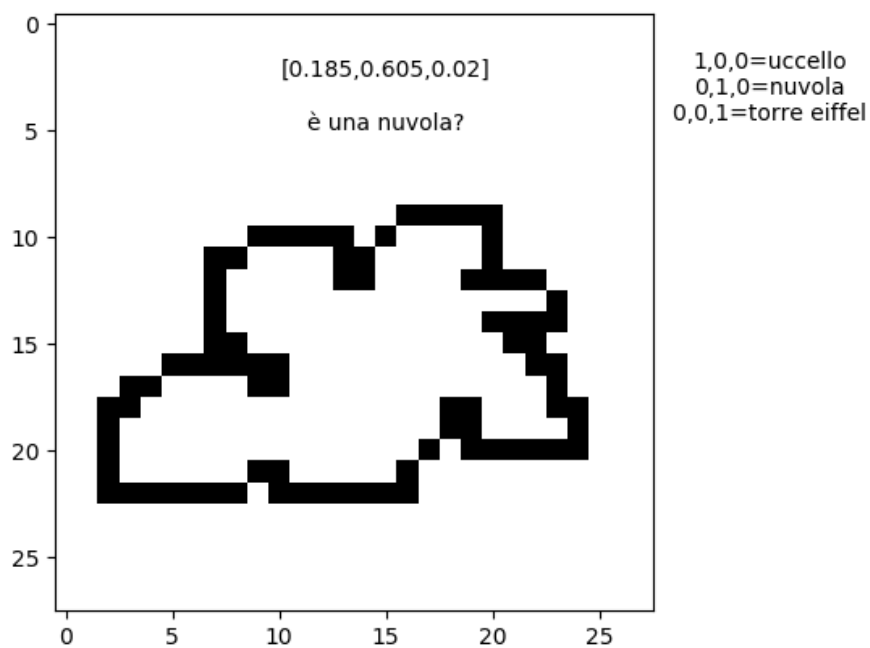


Figure 5: output post esecuzione

## References

- [1] Dataset di google:  
<https://github.com/googlecreativelab/quickdraw-dataset>