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 gia 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.

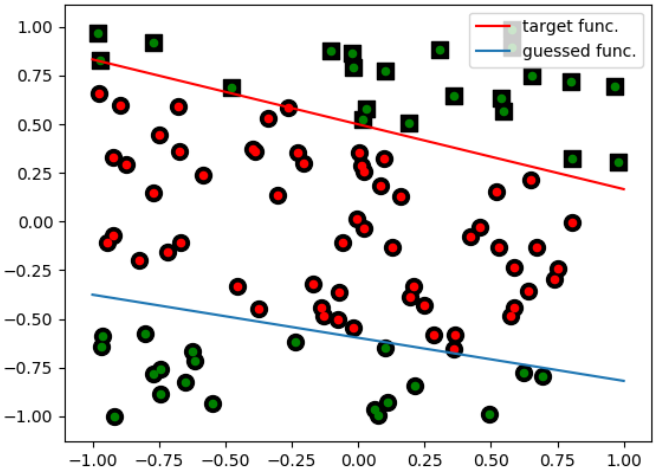

Figure 1: classificazione prima del training

Nella rappresentazione grafica (figura 1) le due classi son rappresentate con quadrati e cerchi colorati di rosso o verde se sono classificati rispettivamnente 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.

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

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

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

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

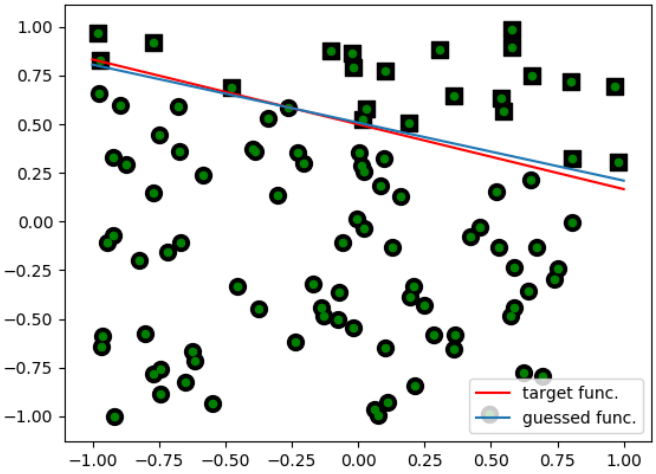

Figure 2: classificazione dopo il training

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.

2.1 Neural network

Il primo passo è stato creare una classe per la rete neurale dotata di un solo hidden layer.

```
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
6      self.weightsI = np.random.random(( hidden_size,input_size))
           ↪ *2-1
7      self.weightsO = np.random.random((out_size,hidden_size))*2-1
8      self.bias_h = np.random.random((hidden_size,1))*2-1
9      self.bias_o = np.random.random((out_size,1))*2-1
10     self.lr = 0.1
11     self.output = np.zeros(out_size)
```

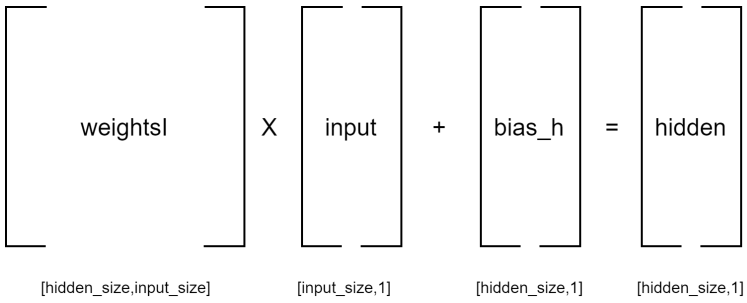

Figure 3: rappresentazione matriciale degli argomenti di classe 1

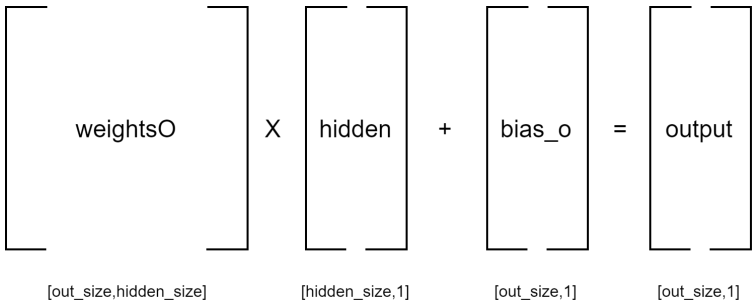

Figure 4: rappresentazione matriciale degli argomenti di classe 2