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.

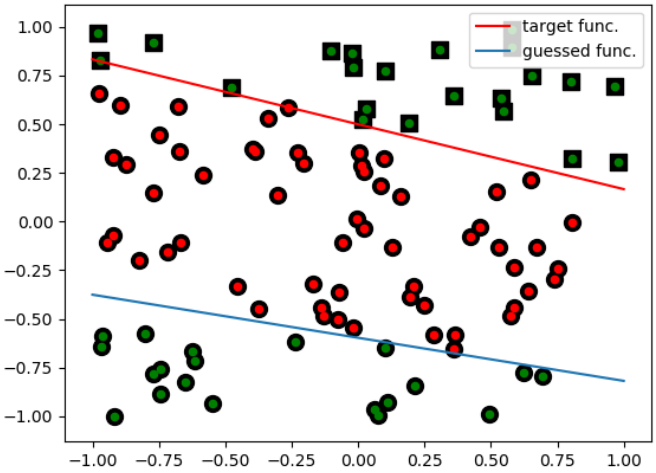

Figure 1: classificazione prima del training

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.

```
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:

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

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

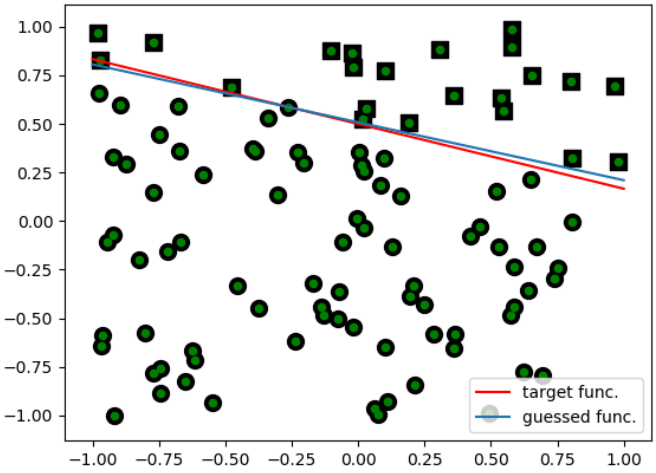

Figure 2: classificazione dopo il training

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

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

```
#funzione di feedforward
def ff(self):
    self.hidden = sigmoid(np.dot(self.weightsI, self.
        ↪ input)+self.bias_h)
    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

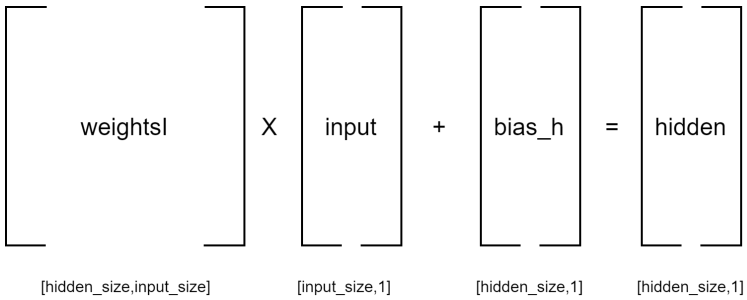

Figure 3: rappresentazione matriciale degli argomenti della classe usati nel calcolo dei valori dell'hidden layer

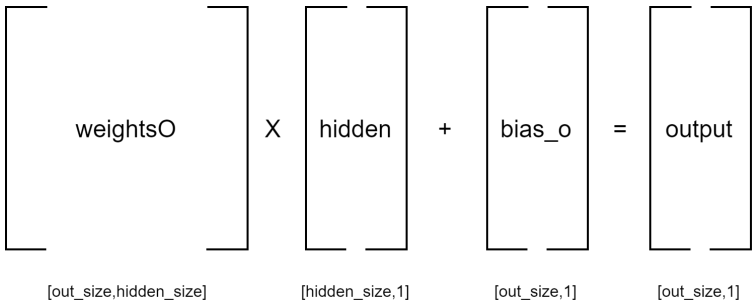

Figure 4: rappresentazione matriciale degli argomenti della classe usati nel calcolo dell'output

```
#funzione di backward propagation
def backprop(self):
    out_err = self.y-self.output
    gradiente_o = (sigmoidD(self.output)*out_err)*self.
        ↪ lr
    delta_ho = np.dot(gradiente_o,self.hidden.T)
    self.weightsO+=delta_ho
    self.bias_o+=gradiente_o
    gradiente_h = sigmoidD(self.hidden)*np.dot(self.
        ↪ weightsO.T,out_err)*self.lr
    delta_ih = np.dot(gradiente_h , self.input.T)
    self.weightsI += delta_ih
    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 gia pronta si puo 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,[
        ↪ splitClouds])
6     ...
```

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 basic1LNN.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).

```
if TRAIN:
    trainingSet = []
    print("generating training set")
    for i in range(splitBirds-1):
        x=np.reshape(TrainBirds[i]/255.0,(784,1))
        y=np.reshape(np.array([1,0,0]),(3,1))
        trainingSet.append([x,y])
    for i in range(splitClouds-1):
        x=np.reshape(TrainClouds[i]/255.0,(784,1))
        y=np.reshape(np.array([0,1,0]),(3,1))
        trainingSet.append([x,y])
    for i in range(splitEiffel-1):
        x=np.reshape(TrainEiffel[i]/255.0,(784,1))
        y=np.reshape(np.array([0,0,1]),(3,1))
        trainingSet.append([x,y])
    r = random.SystemRandom()
    r.shuffle(trainingSet)
    for el in trainingSet:
        oracle.trainOnce(el[0],el[1])
    oracle.export()
else:
    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 Testing

Dopo la fase di training si puo valutare il rendimento della rete testandola sulla seconda parte dei dataset divisi in precedenza.

```
if TEST:
    for i in range(TestBirds.shape[0]):
            ans=oracle.answer(np.reshape(TestBirds[i
               ↪ ]/255.0,(784,1)))
            if max(ans[0],ans[1],ans[2])==ans[0]:
                    continue
            else:
                    #print(ans)
                    wrong+=1
    print(str(int(100*wrong/i))+" percento di errori su
       ↪  "+str(i)+" uccelli")
    ...
    #ripetere su ogni dataset.
```

generando un output testuale

```
3 percento di errori su 33392 uccelli
9 percento di errori su 30066 nuvole
3 percento di errori su 33700 eiffel
```

ottenendo un come risultato un errore inferiore al 10% su tutte le classi (tenendo conto che il database non risulta filtrato e dunque contiene immagini "rumorose").

Nella figura 5 si puo notare come su 20 reti neurali sottoposte a training l'error rate non abbia mai superato il 10%.

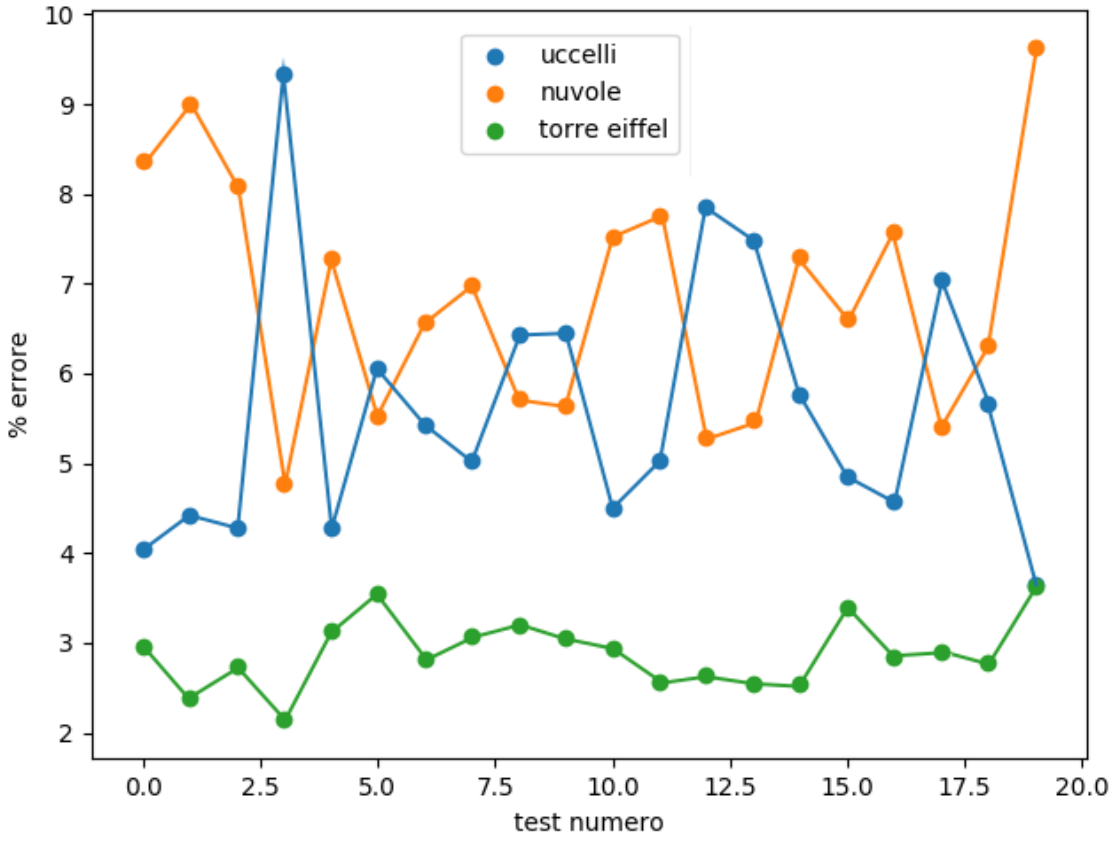

Figure 5: errori percentuali su 20 reti neurali

2.3 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.

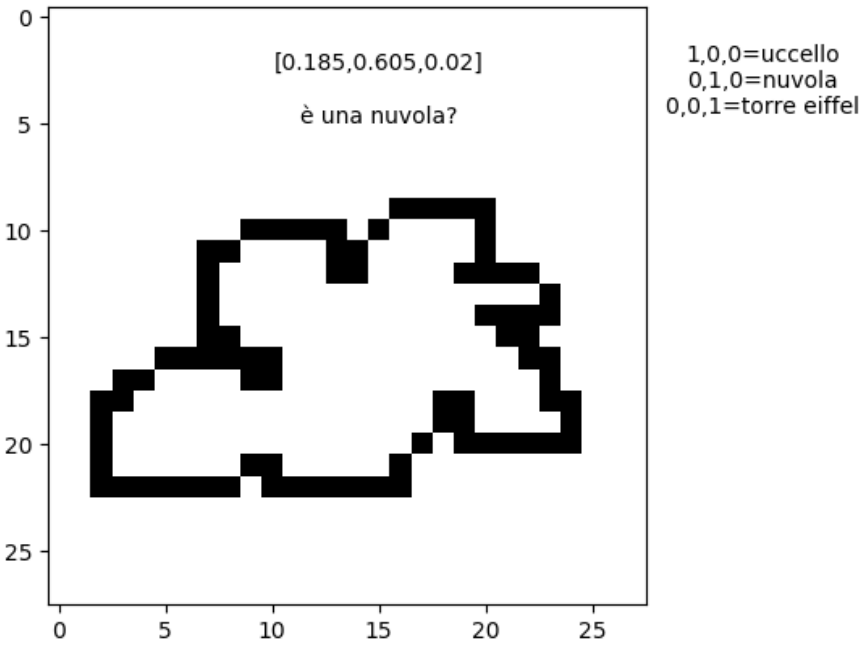

Figure 6: output post esecuzione

References

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