Învățare Automată - Laboratorul 7
Clasificare folosind rețele neurale

Tudor Berariu
tudor.berariu@gmail.com
Laboratorul AIMAS
Facultatea de Automatică și Calculatoare
Universitatea Politehnica București

5 aprilie 2016

1 Scopul laboratorului

Scopul laboratorului îl reprezintă înțelegerea și implementarea unei rețele neurale *feed-forward*, a algoritmilor *backpropagation* și *stochastic gradient descent*.

În cadrul laboratorului se vor clasifica imagini ce conțin cifre scrise de mână (setul de date MNIST[1]).

Rezolvarea cerințelor se va face pas cu pas urmând indicațiile și explicațiile primite.

2 Descrierea fișierelor

`transfer_functions.py` - conține implementări ale diferitelor funcții de transfer; este implementată funcția de transfer identitate; cerința 1 cere să se implementeze funcția logistică și tangenta hiperbolică;

`layer.py` - conține implementarea unui strat ce aplică o funcție de transfer non-lineară asupra unei combinații liniare a intrărilor; trebuie implementate funcțiile pentru fazele *înainte* (de calcul a ieșirilor) și *înapoi* (de propagare a erorilor).

`feed_forward.py` - conține implementarea unei rețele compusă din suprapunerea mai multor straturi;

`data_loader.py` - citește setul de date MNIST sub forma unor tensori;

`train.py` - script pentru antrenarea unei rețele pentru clasificare cifrelor scrise de mână din setul de date MNIST.

[1] `http://yann.lecun.com/exdb/mnist/`

3 Cerințe

3.1 Funcții de transfer

În fișierul `transfer_functions.py` implementați funcțiile `logistic` și `hyperbolic_tangent`. Ambele funcții primesc un array și întorc un array de aceeași dimensiune. Argumentul `derivate` indică dacă este cerută valoarea funcției de transfer într-un punct dat sau dacă este cerută valoarea derivatei.

```
In [1]: from transfer_functions import logistic
In [2]: import numpy as np
In [3]: x = np.array([0.2, 0.8])
In [4]: y = logistic(x)
In [5]: y
Out[5]: array([ 0.549834  ,  0.68997448])
In [6]: dy_dx = logistic(y, True)
In [7]: dy_dx
Out[7]: array([ 0.24751657,  0.2139097 ])
```

Testați implementarea celor două funcții.

```
$ python transfer_functions.py
Testing the identity transfer function ...
Identity transfer function implemented ok!
Testing the logistic transfer function ...
Logistic transfer function implemented ok!
Testing the hyperbolic tangent transfer function ...
Hyperbolic tangent transfer function implemented ok!
```

3.2 Straturi complet-connectate

În fișierul `layer.py` implementați funcțiile `forward` și `backward`.

Funcția `forward` calculează activările unui strat pe baza intrărilor:

$$a_j = \sum_{i=1} w_{ji} x_i + b_j \tag{1}$$

$$y_j = f(a_j) \tag{2}$$

Funcția `backward` calculează gradientul erorii în raport cu parametrii și gradientul erorii în raport cu intrările pe baza gradientului erorii în raport cu ieșirile stratului.

Fiecare dintre aceste calcule se face într-un singur rând folosind funcțiile rapide pentru tensori din `numpy`.

După implementare, testați corectitudinea:

```
$ python layer.py
Testing forward computation...
Forward computation implemented ok!
Testing backward computation...
    i. testing gradients w.r.t. the bias terms...
     OK
   ii. testing gradients w.r.t. the weights...
     OK
  iii. testing gradients w.r.t. the inputs...
     OK
Backward computation implemented ok!
```

3.3 Rețele cu mai multe straturi

În fișierul `FeedForward` implementați funcția `update_parameters`. Aceasta primește un singur argument, rata de învățare și actualizează parametrii fiecărui strat considerând că gradientul erorii în raport cu aceștia a fost calculat.

3.4 Antrenarea rețelei

În fișierul `train.py` implementați funcția `train_nn`. Aceasta trebuie să ia într-o ordine aleatoare fiecare exemplu din setul de învățare, să îl treacă prin rețea, să calculeze gradientul erorii și să actualizeze ponderile. Din când în când se vor calcula acuratețea pe setul de date de antrenare și acuratețea rețelei pe setul de date de test.

După implementare, testați că rețeaua învață să recunoască cele 10 cifre:

```
$ python train.py --learning_rate 0.001
(784 -> 300) -> (300 -> 10)
Train acc: 0.150400 ; Test acc: 0.132900
Train acc: 0.144000 ; Test acc: 0.127900
Train acc: 0.311400 ; Test acc: 0.304100
Train acc: 0.448400 ; Test acc: 0.431400
Train acc: 0.537200 ; Test acc: 0.530600
Train acc: 0.536800 ; Test acc: 0.531000
Train acc: 0.553600 ; Test acc: 0.545700
Train acc: 0.596000 ; Test acc: 0.592000
Train acc: 0.620800 ; Test acc: 0.617200
Train acc: 0.639400 ; Test acc: 0.626700
Train acc: 0.669400 ; Test acc: 0.664100
Train acc: 0.664400 ; Test acc: 0.652700
Train acc: 0.670400 ; Test acc: 0.665500
Train acc: 0.706600 ; Test acc: 0.712400
Train acc: 0.701400 ; Test acc: 0.697700
Train acc: 0.697200 ; Test acc: 0.691300
Train acc: 0.710000 ; Test acc: 0.710200
```