Învățare Automată - Laboratorul 9
Segmentarea și compresia imaginilor utilizând rețele Kohonen

Tudor Berariu
Laboratorul AIMAS
Facultatea de Automatică și Calculatoare

8 aprilie 2014

1 Scopul laboratorului

Scopul acestui laborator îl reprezintă înțelegerea și implementarea unei rețele Kohonen. Aceasta va fi aplicată pentru rezolvarea unei probleme de învățare nesupervizată: segmentarea imaginilor. Implementarea se va realiza în Matlab / Octave[1], Python[2] sau C++[3].

2 Rețele Kohonen

O rețea Kohonen este formată din N neuroni dispuși liniar, sub forma unei matrice sau, mai rar, în spații de dimensiuni mai mari. Această așezare permite identificarea vecinătății unui neuron, concept important în procesul de învățare. Scopul acestui tip de rețele este ca neuronii *învecinați* (apropiați) să răspundă unor semnale similare, iar perechi de neuroni mai *îndepărtați* să caracterizeze exemple mai puțin asemănătoare.

Antrenarea rețelei Kohonen corespunde unei segmentări a spațiului de intrare într-un număr de regiuni egal cu numărul de neuroni din rețea. Atunci

[1] este nevoie de pachetul `gnuplot-x11`
[2] este nevoie de pachetele `python-matplotlib` și `imagemagick`
[3] este nevoie de biblitecile `libjpeg` și `libgtkmm`

când un exemplu dintr-o astfel de regiune este transmis rețelei, neuronul corespunzător trebuie să prezinte nivelul de excitare maxim.

Nivelul de excitare al unui neuron este invers proporțional cu distanța euclidiană dintre ponderile sale și valorile exemplului dat la intrare. Practic, se va activa neuronul cel mai apropiat de semnalul de intrare.

Pentru antrenarea rețelei se folosește Algoritmul 1, unde rata de învățare η și vecinătatea ϕ sunt funcții ce depind de timp (numărul iterației).

În Algoritmul 1, $\mathbf{W}$ este o matrice de dimensiune $n_1 \times n_2 \times \ldots \times n_k \times d$ unde $n_1, n_2, \ldots, n_k$ sunt dimensiunile laticei de neuroni ($n_1 \cdot n_2 \cdot \ldots \cdot n_k = N$), iar d este dimensiunea spațiului de intrare.

Algoritmul 1 Antrenarea Rețelelor Kohonen

Intrări: spațiul de intrare $\mathbf{X}$, funcțiile η (rata de învățare), ϕ (vecinătate)
Ieșire: ponderile $\mathbf{W}$

1: $\mathbf{W} \longleftarrow random(0, 1)$
2: $t \longleftarrow 1$
3: **repetă**
4: se alege $\mathbf{x_i} \in \mathbf{X}$ aleator
5: $w_z \longleftarrow \underset{\mathbf{w} \in \mathbf{W}}{\text{argmin}}\ Distance(\mathbf{w}, \mathbf{x_i})$
6: **pentru toate** $\mathbf{w_j} \in \mathbf{W}$ **execută**
7: $\mathbf{w_j} \longleftarrow \mathbf{w_j} + \eta(t)\phi(w_z, t)(\mathbf{x_i} - \mathbf{w_j})$
8: **termină ciclu**
9: $t \longleftarrow t + 1$
10: **până când** algoritmul converge sau numărul maxim de iterații a fost atins

3 Segmentarea imaginilor

Pentru a rezolva sarcinile de laborator alegeți unul din următoarele limbaje de programare: Matlab / Octave, Python sau C++. Puteți implementa de la zero într-un alt limbaj la alegere, dar nu se recomandă acest lucru.

3.1 Cerința 1, de încălzire: Negativul unei imagini

Matlab / Octave

În arhiva laboratorului se găsește fișierul `negative.m`. Completați codul modificând valorile matricei `neg_pixels` din funcția `negative` pentru a calcula negativul imaginii date la intrare.

Pentru a verifica dacă totul funcționează bine, rulați în Matlab / Octave următoarea comandă:

```
>> negative('imgs/1.jpg')
```

Dacă ați implementat corect, pe ecran va apărea imaginea din Figura 1.

Python

În arhiva laboratorului se găsește fișierul `negative.py`. Completați codul modificând valorile listei `neg_pixels` din funcția `negative` pentru a calcula negativul imaginii date la intrare.

Pentru a verifica dacă totul funcționează bine, executați în consolă:

```
# chmod +x negative.py
# ./negative.py imgs/1.jpg
```

Dacă ați implementat corect, pe ecran va apărea imaginea din Figura 1.

C++

În arhiva laboratorului se găsește fișierul `negative.cc`. Completați codul pentru a calcula negativul imaginii date la intrare.

Pentru a verifica dacă implementarea este corectă, compilați și rulați `./negative imgs/1.jpg`.

```
$ make clean
$ make negative
$ ./negative imgs/1.jpg
```

Dacă ați implementat corect, pe ecran va apărea imaginea din Figura 1.

Figura 1: Negativul imaginii `1.jpg`

3.2 Taskul 2: Rata de învățare

În al doilea pas trebuie să calculați rata de învățare. Pentru început, implementați o descreștere liniară a ratei de învățare de la 0.75 la 0.1.

După ce terminați toate task-urile, puteți încerca alte limite (superioară și inferioară) pentru rata de învățare, precum și variații pătratice, exponențiale, etc. pentru a vedea cum influențează rezultatul segmentării.

Matlab / Octave

Deschideți fișierul `learning_rate.m` și modificați corpul funcției `learning_rate` astfel încât să întoarcă valoarea ratei de învățare în funcție de numărul iterației curente și numărul total de iterații.

Pentru o verificare rapidă a codului rulați în Matlab / Octave:

```
>> plot_learning_rate
```

Python

Deschideți fișierul `learning_rate.py` și modificați corpul funcției `learning_rate` astfel încât să întoarcă valoarea ratei de învățare în funcție de numărul iterației curente și numărul total de iterații.

Pentur o verificare rapidă a codului, rulați în Python:

```
# chmod +x plot_learning_rate.py
# ./plot_learning_rate.py
```

C++

Implementați funcția `double learning_rate(int, int)` din fișierul `learning_rate.cc` astfel încât să întoarcă valoarea ratei de învățare în funcție de numărul iterației curente și numărul total de iterații.

```
make clean
make print_learning_rate
./print_learning_rate | gnuplot -e "plot '-' with lines" --persist
```

3.3 Exercițiul 3: Raza vecinătății

În cadrul exercițiului 3 trebuie să modificați funcția `radius` astfel încât să întoarcă valoarea razei vecinătății în funcție de numărul iterației, dar și de dimensiunile rețelei Kohonen.

Pentru început, implementați o descreștere liniară a razei vecinătății de la $\frac{\max(width,height)}{2}$ către valoarea 0 (vecinătatea unui neuron nu conține alți neuroni în afară de acesta).

După ce implementați tot algoritmul încercați valori de start mai mici pentru rază și descreșteri mai rapide și observați cum variază rezultatele.

Matlab / Octave

În arhivă se găsește fișierul `radius.m`. Completați aici implementarea funcției `radius`. Pentru a verifica rezolvarea, rulați în Matlab / Octave:

```
>> plot_radius
```

3.3.1 Python

Între sursele Pythongăsiți fișierul `radius.py`. Modificați corpul funcției astfel încât să calculeze rata vecinătății conform indicațiilor de mai sus. Pentru verificare executați:

```
$ chmod +x plot_radius.py
$ ./plot_radius.py
```

3.3.2 C++

În fișierul `radius.cc` completați codul funcției `double radius(int iter_no, int iter_count, int height, int width)` astfel încât să calculeze raza vecinătății conform indicațiilor de mai sus. Verificați-vă astfel:

```
$ make clean
$ make print_radius
$ ./print_radius | gnuplot -e "plot '-' with lines" --persist
```

3.4 Exercițiul 4: Vecinătatea

Vecinătatea reprezintă mulțimea acelor neuroni ale căror ponderi vor fi actualizate într-un ciclu. Mulțimea cuprinde neuronul *câștigător* și neuronii aflați la o distanță mai mică decât raza vecinătății (vezi taskul 3).

Modificați funcția `neighbourhood(y, x, radius, height, width)` astfel încât să întoarcă o matrice de dimensiunea rețelei Kohonen care să aibă valoarea 1 pentru neuronii din interiorul vecinătății și zero pentru ceilalți.

Parametrii funcției sunt:

- `y` - coordonata y (linia) a neuronului câștigător (centrul vecinătății)
- `x` - coordonata x (coloana) a neuronului câștigător (centrul vecinătății)
- `radius` - valoarea razei
- `height` - înălțimtea laticei (bidimensionale) de neuroni
- `width` - lățimea laticei (bidimensionale) de neuroni

După terminarea tuturor exercițiilor, vă puteți întoarce la funcția neighbourhood pentru a experimenta cu valori zecimale din intervalul $[0, 1]$ (1 în centru și valori ce descresc pe măsură ce distanța față de centru crește).

Matlab / Octave

Deschideți fișierul `neighbourhood.m` și implementați funcția urmând indicațiile date.

Pentru verificare, introduceți în Matlab / Octave:

```
octave:1> neighbourhood(4,4,3,7,7)
ans =

   0   0   0   1   0   0   0
   0   1   1   1   1   1   0
   0   1   1   1   1   1   0
   1   1   1   1   1   1   1
   0   1   1   1   1   1   0
   0   1   1   1   1   1   0
   0   0   0   1   0   0   0

octave:2> neighbourhood(6,5,3,7,7)
ans =

   0   0   0   0   0   0   0
   0   0   0   0   0   0   0
   0   0   0   0   1   0   0
   0   0   1   1   1   1   1
   0   0   1   1   1   1   1
   0   1   1   1   1   1   1
   0   0   1   1   1   1   1
```

Python

Implementați funcția în fișierul `neighbourhood.py` și testați astfel:

```
 $ chmod +x neighbourhood.py
 $ ./neighbourhood.py 4 4 3 7 7
[[0, 0, 0, 1, 0, 0, 0],
 [0, 1, 1, 1, 1, 1, 0],
 [0, 1, 1, 1, 1, 1, 0],
 [1, 1, 1, 1, 1, 1, 1],
 [0, 1, 1, 1, 1, 1, 0],
 [0, 1, 1, 1, 1, 1, 0],
 [0, 0, 0, 1, 0, 0, 0]]
 $ ./neighbourhood.py 6 5 3 7 7
[[0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0],
```

```
 [0, 0, 0, 0, 1, 0, 0],
 [0, 0, 1, 1, 1, 1, 1],
 [0, 0, 1, 1, 1, 1, 1],
 [0, 1, 1, 1, 1, 1, 1],
 [0, 0, 1, 1, 1, 1, 1]]
```

C++

Implementați funcția în fișierul `neighbourhood.cc`. Pentru a vă testa implementarea (Atenție, se indexează de la 1):

```
$ make clean
$ make print_neighbourhood
$ ./print_neighbourhood 4 4 3 7 7
0 0 0 1 0 0 0
0 1 1 1 1 1 0
0 1 1 1 1 1 0
1 1 1 1 1 1 1
0 1 1 1 1 1 0
0 1 1 1 1 1 0
0 0 0 1 0 0 0
$ ./print_neighbourhood 6 5 3 7 7
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 1 0 0
0 0 1 1 1 1 1
0 0 1 1 1 1 1
0 1 1 1 1 1 1
0 0 1 1 1 1 1
```

3.5 Taskul 5: Antrenarea rețelei Kohonen

Completați corpul funcției `som_segmentation` pentru a calcula ponderile neuronilor conform Algoritmului 1. Folosiți funcțiile implementate anterior. Parametrul n reprezintă lungimea matricei pătratice de neuroni ($N = n^2$).

W este o matrice de dimensiune $n \times n \times 3$, adică va conține n^2 valori RGB.

Implementarea se va face în fișierul `som_segmentation.m` dacă lucrați în Matlab / Octave, în fișierul `som_segmentation.py` dacă lucrați în Pythonsau în fișierul `som_segmentation.cc` dacă alegeți C++.

3.6 Taskul 6: Segmentarea imaginii

Completați funcția `som_segmentation` pentru a construi o imagine modificată pornind de la cea originală și înlocuind fiecare pixel cu valorile neuronului cel mai apropiat (distanță euclidiană; după antrenarea rețelei).

În final, imaginea nouă (`seg_pixels`) trebuie să conțină doar valori egale cu ponderile neuronilor din rețeaua antrenată.

Salvați imaginea pe disc adăugând la numele fișierului original sufixul `_seg` (de exemplu `1_seg.jpg`).

Un posibil rezultat folosind o rețea cu 9 neuroni este în Figura 2.

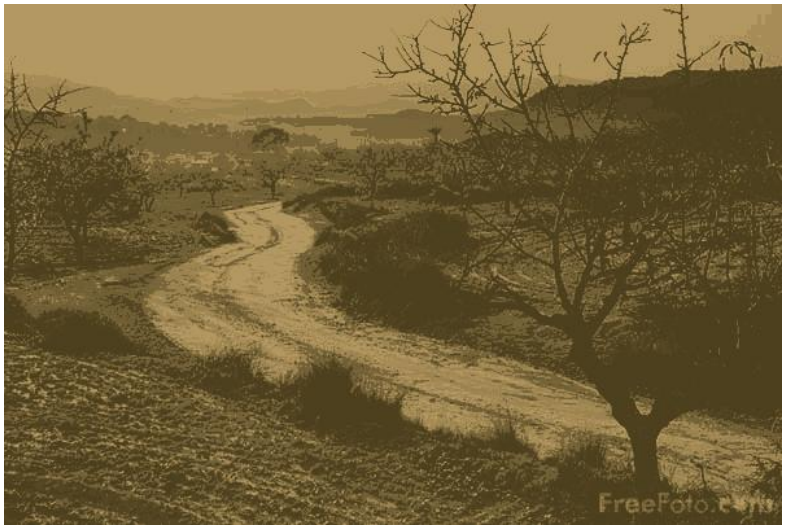

Figura 2: O posibilă segmentare a imaginii `1.jpg`

Matlab / Octave

Rezolvarea se va face, la fel ca la Exercițiul 5, în fișierul `som_segmentation.m`. Rulați apoi:

```
> som_segmentaion('imgs/3.jpg', 4)
```

Python

Rezolvarea se va face, la fel ca la Exercițiul 5, în fișierul `som_segmentation.py`. Rulați apoi:

```
$ ./som_segmentation imgs/3.jpg 4
```

C++

Rezolvarea se va face, la fel ca la Exercițiul 5, în fișierul `som_segmentation.cc`. Rulați apoi:

```
$ make clean
$ make som_segmentation
$ ./som_segmentation imgs/3.jpg 4
```