Învățare Automată
Laboratorul 9: **Segmentarea și compresia imaginilor utilizând rețele Kohonen**

Tudor Berariu
Laboratorul AIMAS
Facultatea de Automatică și Calculatoare

15 aprilie 2013

1 Introducere

În cadrul acestui laborator se vor implementa în Matlab / Octave sau Python[1] **rețele Kohonen** pentru rezolvarea unei probleme de învățare nesupervizată: segmentarea imaginilor.

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

[1] este nevoie de pachetul `python-matplotlib`

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

3.1 Taskul 1: De încălzire

Descărcați arhiva laboratorului de pe `curs.cs` și deschideți pentru editare fișierul `negative.m` dacă vreți să lucrați în Matlab / Octave sau fișierul `negative.py` dacă lucrați în Python.

Completați codul (modificând valorile matricei / listei `neg_pixels` din funcția `negative`) pentru a calcula negativul imaginii date la intrare.

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

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

... sau în Python executați în consolă:

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

Pe ecran va apărea imaginea din Figura 1.

Figura 1: Negativul imaginii `1.jpg`

3.2 Taskul 2: Rata de învățare

Deschideți fișierul `learning_rate.m` dacă lucrați în Matlab / Octave sau `learning_rate.py` pentru Python. 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 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.

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

```
>> plot_learning_rate
```

... sau în Python:

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

3.3 Taskul 3: Raza vecinătății

Deschideți fișierul `radius.m` dacă lucrați în Matlab / Octave sau `radius.py` dacă lucrați în Python. Modificați corpul funcției astfel încât să întoarcă valoarea ratei de învățare 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.

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

```
>> plot_radius
```

... sau în Python:

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

3.4 Taskul 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).

Deschideți fișierul `neighbourhood.m` dacă lucrați în Maltab / Octave sau `neighbourhood.py` dacă lucrați în Python și modificați funcția `neighbourhood` 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:

- `x` - coordonata x a neuronului câștigător (centrul vecinătății)
- `y` - coordonata y a neuronului câștigător (centrul vecinătății)
- `radius` - valoarea razei

- `width` - lățimea laticei (bidimensionale) de neuroni
- `height` - înălțimtea laticei (bidimensionale) de neuroni

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(5,6,3,7,7)
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
```

... sau, similar, în Python:

```
 # 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 5 6 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]]
```

După finalizarea tuturor task-urilor, puteți experimenta cu valori zecimale din intervalul $[0, 1]$ (1 în centru și valori ce descresc pe măsură ce distanța față de centru crește).

3.5 Taskul 5: Antrenarea rețelei Kohonen

Deschideți fișierul `som_segmentation.m` dacă lucrați în Matlab / Octave sau fișierul `som_segmentation.py` dacă lucrați în Python. Modificați funcția `som_segmentation` pentru a calcula ponderile neuronilor conform Algoritmului 1. Folosiți funcțiile implementate anterior. Parametrul n reprezintă lungimea matricei de neuroni ($N = n^2$).

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

3.6 Taskul 6: Segmentarea imaginii

Modificați finalul funcției `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).

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`