

Învățare Automată - Laboratorul 9

Segmentarea și compresia imaginilor utilizând rețele Kohonen

Tudor Berariu
Laboratorul AIMAS
Facultatea de Automatică și Calculatoare

11 aprilie 2016

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.

2 Rețele Kohonen

O rețea Kohonen este formată din N neuroni dispuși pe o latice. Această așezare permite identificarea vecinătății unui neuron, concept important în procesul de învățare. Scopul acestui tip de rețele este ca neuronii *învecinați* (aproiați) 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 excitație maxim.

Nivelul de excitație 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 $n_1 \times n_2 \times \dots \times n_k \times d$ unde n_1, n_2, \dots, n_k sunt dimensiunile lăței de neuroni ($n_1 \cdot n_2 \cdot \dots \cdot n_k = N$), 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: $\mathbf{W} \leftarrow \text{random}(0, 1)$
 - 2: $t \leftarrow 1$
 - 3: **repetă**
 - 4: se alege $\mathbf{x}_i \in \mathbf{X}$ aleator
 - 5: $w_z \leftarrow \underset{\mathbf{w} \in \mathbf{W}}{\text{argmin}} \text{Distance}(\mathbf{w}, \mathbf{x}_i)$
 - 6: **pentru toate** $\mathbf{w}_j \in \mathbf{W}$ **execută**
 - 7: $\mathbf{w}_j \leftarrow \mathbf{w}_j + \eta(t)\phi(w_z, t)(\mathbf{x}_i - \mathbf{w}_j)$
 - 8: **termină ciclu**
 - 9: $t \leftarrow t + 1$
 - 10: **până când** algoritmul converge sau numărul maxim de iterații a fost atins
-

3 Segmentarea imaginilor

3.1 Cerința 1, de încălzire: Negativul unei imagini

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.



Figura 1: Negativul imaginii 1.jpg

3.2 Taskul 2: Rata de învățare

În al doilea pas trebuie calculată rata de învățare pentru agloritmul de antrenare. 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.

În fișierul `learning_rate.py` modificați funcția `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 Python:

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

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 $\frac{\max(\text{width}, \text{height})}{2}$ 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.

3.3.1 Python

Între sursele [Python](#) găsiți 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
```

3.4 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` - înălțimea lăței (bidimensionale) de neuroni
- `width` - lățimea lăței (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).

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]]

```

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.py`.

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

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

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