自己組織化写像

2016年6月24日

本日の内容

- 自己組織化写像について、プログラムsom.pyのコードに基づいて解説。

自己組織化写像

- 自己組織化写像（SOM）はニューラルネットワークの一種で、大脳皮質の視覚野をモデル化したもの。コホネンによって提案されたモデル。多次元のデータの可視化が可能。

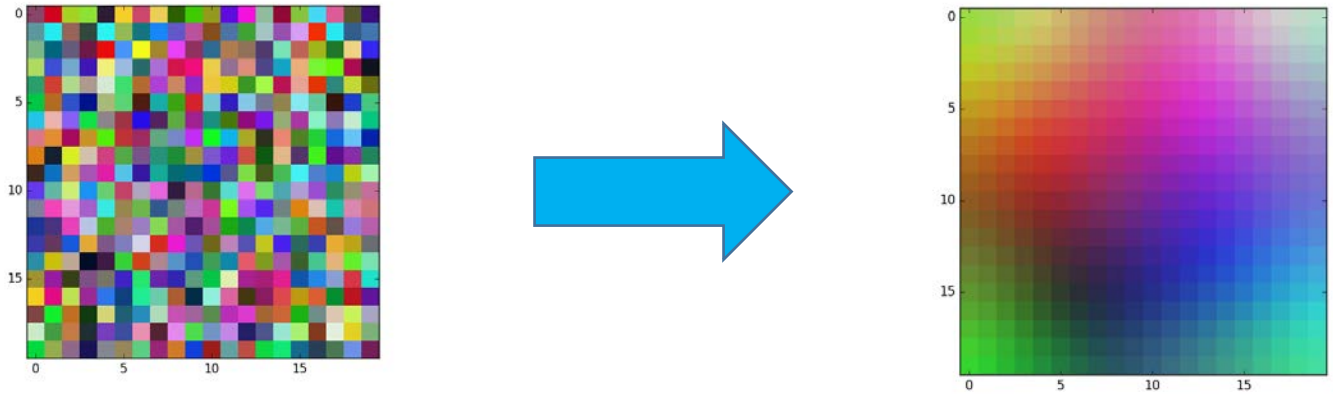
  https://ja.wikipedia.org/wiki/%E8%87%AA%E5%B7%B1%E7%B5%84%E7%B9%94%E5%8C%96%E5%86%99%E5%83%8F

som.py

- R，G，Bで表される色等のベクトル表記される多次元のデータを、マップ上の座標に近いもの同士で集めることが可能。
- 勝手に学習して分類してくれる。
- 色の場合の学習例がsom.py。

自己組織化写像（som.py）

- 実行終了まで少しかかる（final.pngが作られるまで待つ）。
- 最初はランダムだった配色が、回数を重ねるごとに少しずつランダム性が失われ、マス目がくっきりする。
- 似たような色が近くに並ぶが、これはコンピュータが元々類似色を知っていたわけではない



自己組織化写像（som.py）

- ある色を選択し、その色と一番近いものを図の中から探す。
  一番近かった色のマスとその周囲1マスに選択した色を混ぜる。
  この工程を繰り返すことで、図中の色の分布が似たような色で固まるようになる。
- デモ・参考
  http://gaya.jp/spiking_neuron/som.htm

コード解説について

- http://technocrat.hatenablog.com/entry/2015/02/12/014557
  上記のページにてパラメータ等の細かい設定の解説をしているので、このページとコードを照らし合わせて見ると理解がしやすい。

自己組織化写像（som.py）

```
import numpy as np
import matplotlib.pyplot as plt


"""
Global parameters
"""
N = 100 # linear size of 2D map
n_teacher = 10000 # # of teacher signal
np.random.seed(100)# test seed for random number
```

- numpyやmatplotlibの機能を使う。
- Nに2Dマップの長さを設定。
- n_teacher学習の回数を設定。
- numpyの機能での乱数生成設定。

自己組織化写像（som.pyのmain関数（1/2））

```
def main():
    # initialize node vectors
    nodes = np.random.rand(N,N,3)# node array. each node has 3-dim weight vector
    #initial out put
    #TODO; make out put function to simplify here
    plt.imshow(nodes, interpolation='none')
    plt.savefig("init.png")
    """
    """ Learning """
```

- nodesにN×Nの個のランダムな3値（R,G,B）を持ったノード生成。
- それをmatplotlibの機能で図にする。N×Nマスで、それぞれのマスが3値に応じた色を塗った状態の図となる。
- この初期状態の図をinit.pngという名前のファイルにして出力。

自己組織化写像（som.pyのmain関数（2/2））

```
# teacher signal
teachers = np.random.rand(n_teacher,3)
for i in range(n_teacher):
    train(nodes, teachers, i)
    # intermediate out put
    if i%1000 ==0 or i< 100: #out put for i<100 or each 1000 iteration
        plt.imshow(nodes, interpolation='none')
        plt.savefig(str(i)+".png")
#output
plt.imshow(nodes, interpolation='none')
plt.savefig("final.png")
```

- teachersに学習回数分のランダムな3値（R,G,B）を用意する。
- teachersの長さ（学習回数分）だけfor文を回し、trainで学習を行う。
- その後、もしiが1000の倍数か100未満だったら、その状態での図をファイル出力する。（デフォルトだとiが0～99,1000,2000,…10000の時に出力している。）
- 設定した回数終わったら最終状態を画像出力する。

自己組織化写像（som.pyのtrain関数（1/2））

```
def train(nodes, teachers, i):
    bmu = best_matching_unit(nodes, teachers[i])
    #print bmu
    for x in range(N):
        for y in range(N):
            c = np.array([x,y])# coordinate of unit
            d = np.linalg.norm(c-bmu)
            L = learning_ratio(i)
            S = learning_radius(i,d)
            for z in range(3): #TODO clear up using numpy function
                nodes[x,y,z] += L*S*(teachers[i,z] - nodes[x,y,z])
```

- bmuにbest_matching_unitの結果を入れる（teachers[i]に最も近い色）。
- 図のマップの長さだけfor文を回し（２重）、１マスずつ見ていく。
- マスごとに更新式　$W_{t+1}(\vec{x}) = W_t(\vec{x}) + \Theta(\vec{x}, t)L(t)(V_t - W_t(\vec{x}))$

  に従い更新する。（teachersに近い色周辺のマスに色を混ぜる意味合い）
  c,d,L,Sで計算の材料を準備している。L×Sは、学習が進むにつれ、更新するノードの範囲（半径）が小さくなっていくことを意味する。

自己組織化写像（som.pyのtrain関数（2/2））

```
def train(nodes, teachers, i):
    bmu = best_matching_unit(nodes, teachers[i])
    #print bmu
    for x in range(N):
        for y in range(N):
            c = np.array([x,y])# coordinate of unit
            d = np.linalg.norm(c-bmu)
            L = learning_ratio(i)
            S = learning_radius(i,d)
            for z in range(3): #TODO clear up using numpy function
                nodes[x,y,z] += L*S*(teachers[i,z] - nodes[x,y,z])
```

- arrayはベクトル生成。
- linalg.normは、ベクトルの長さ（ノルム）を計算。
- dでは、cを用いて各マスの位置とbmuの位置の間の距離を求めている。
- L,Sについては、該当する関数にて。

自己組織化写像
(som.pyのbest_matching_unit関数)

```
def best_matching_unit(nodes, teacher):
    #compute all norms (square)
    #TODO simplify using numpy function
    norms = np.zeros((N,N))
    for i in range(N):
        for j in range(N):
            for k in range(3):
                norms[i,j] += (nodes[i,j,k] - teacher[k])**2
    #then, choose the minimum one
    bmu = np.argmin(norms) #argment with minimum element
    # argmin returns just flatten, serial index,
    # so convert it using unravel_index
    return np.unravel_index(bmu,(N,N))
```

- 各ノード毎に、色それぞれについて(kのfor文)teacherとの距離を加えている。
- argminは、要素中の最小値を取るため、一番近いもの。
- teacherの色に一番近い色のマスを探し、返す。(unravel_indexにより、最小値のインデックスが求まる。)

自己組織化写像
（som.pyのneighbourhood関数）

```
def neighbourhood(t):#neighbourhood radious
    halflife = float(n_teacher/4) #for testing
    initial  = float(N/2)
    return initial*np.exp(-t/halflife)
```

$$\sigma(t) = \sigma_0 \exp\left(-\frac{t}{\lambda}\right)$$

- learning_radius関数の計算材料を提供する。
- tは現在の学習ステップ（計算時間）
- λ（halftime,各自設定するパラメータ）は時定数で、今回は10000/4と設定している。（10000はn_teacherの数に依存）
- σ0（initial,各自設定するパラメータ）は、今回は20/2と設定している。（20はNの数に依存）

自己組織化写像
(som.pyのlearning_ratio関数)

```
def learning_ratio(t):
    halflife = float(n_teacher/4) #for testing
    initial  = 0.1
    return initial*np.exp(-t/halflife)
```

$$L(t) = L_0 \exp\left(-\frac{t}{\lambda}\right)$$

- tは現在の学習ステップ(計算時間)
- L0(各自設定するパラメータ)は0.1と設定している。
- λ (halftime,各自設定するパラメータ)は時定数で、今回は10000/4と設定している。
- tが大きくなるにつれ、だんだん小さくなる。

自己組織化写像
（som.pyのlearning_radius関数）

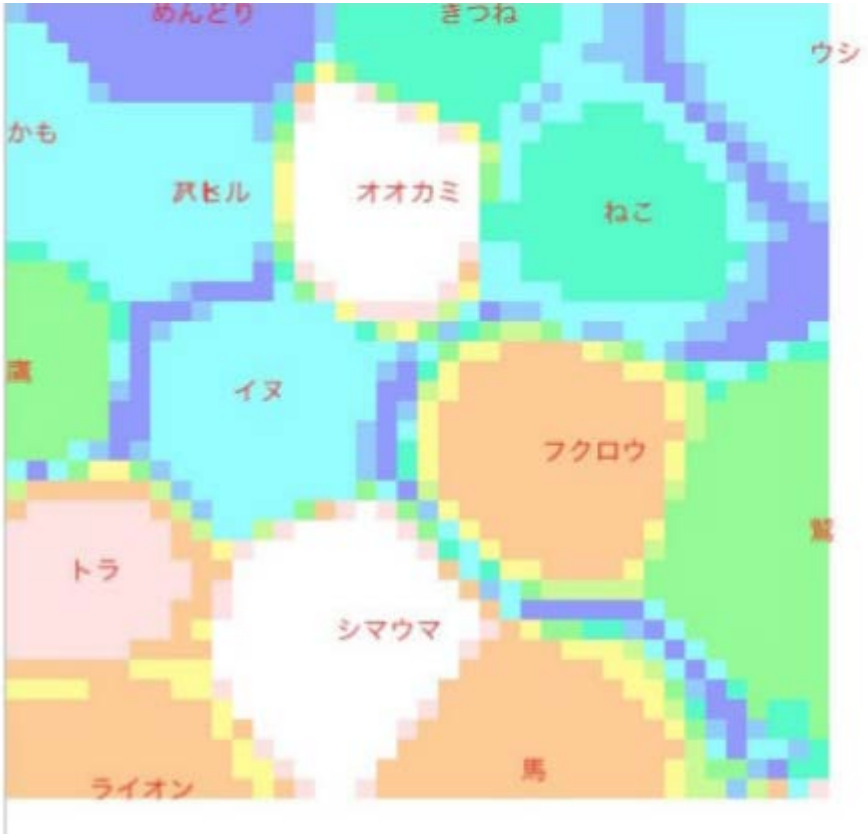
```
def learning_radius(t, d):
    # d is distance from BMU
    s = neighbourhood(t)
    return np.exp(-d**2/(2*s**2))
```

$$\Theta(\vec{x}, t) = \exp\left(-\frac{d^2}{2\sigma^2(t)}\right)$$

- sは下の式のσ(t)に該当し、neighbourhoodにて値を得る。
- dは現在みているマスと、teachers[i]の色と最も近かった色のマスとの距離。
- ノード更新を範囲（半径）を求める。

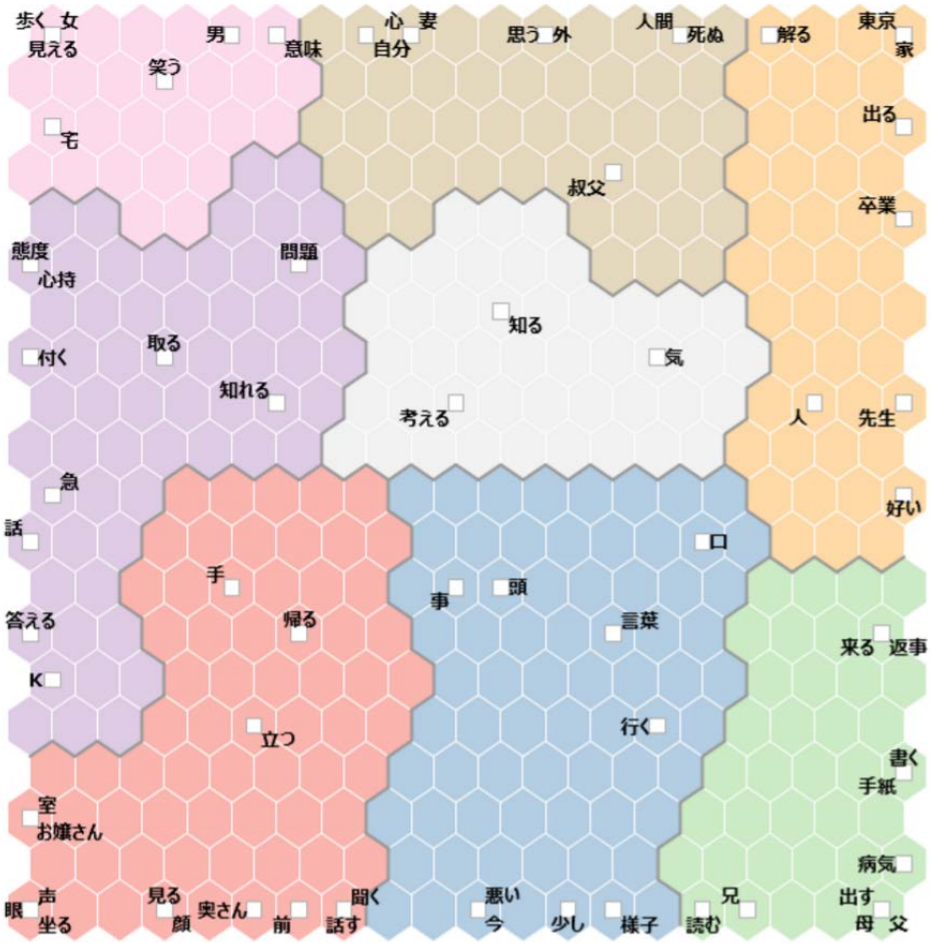
応用例

- http://www.pirika.com/JP/ChemInfo/SOMdoubutu.html



応用例

- http://khc.sourceforge.net/scr_r.html



演習

- Nやn_teacherの値を変えるとどう変化するか。
- どんなことに応用できるか考えてみる。
- プログラムは現在、ランダム生成した色での学習をしているが、実際に何か検証する際はランダムでない用意したデータを使う。

Further reading

- Hybrid Kohonen self-organizing map

https://en.wikipedia.org/wiki/Hybrid_Kohonen_self-organizing_map

Hybrid Kohonen SOM has been used in weather prediction and especially in forecasting stock prices, which has made a challenging task considerably easier. It is fast and efficient with less classification error, hence is a better predictor, when compared to Kohonen SOM and backpropagation networks.

今後

- 7/1　誤差逆伝搬法に触れる。
- 7/8,15　最終発表