Lab2 SOM Application Report

Yan Liu

Part 1– SOM Implementation

1. key points of the implementation

For D-1 SOM, I choose a typical procedure in initialization: sample initialization, where the weight vectors are initialized with random samples drawn from the input data set. Then I initialize two coefficients τ_1 and τ_2 which are used in decreasing learning rate and radius while iteration time increasing. Then do the iteration: Randomly pick an input vector from the training dataset. Get the winner (closest) neuron (Minimum Euclidean distance). Find neighbor neuron of the winner neuron (Judging by city-block distance). Update the weights of the winner and its neighbors. Decrease the learning rate and the radius.
I choose the learning algorithm provided in the slide to **decay both the learning rate and radius.** The learning rate decay rule is: $\eta(t) = \eta_0 \exp\left(-\frac{t}{\tau_1}\right)$, where t is the current iteration number. The radius decay rule is: $\sigma(t) = \sigma_0 exp\left(-\frac{t}{\tau_2}\right)$, where η_0 and σ_0 is the initial learning rate and radius. Then using a neighborhood kernel function and neighborhood size (radius) to **update the weights of a neuron then affect those nearby**. The updated rule function is $W_K = W_K + \eta(t) \cdot h_{ik}(t) \cdot (x_n - W_k)$,where neighborhood kernel function $h_{ik}(t) = exp\left(-\frac{d_{ik}^2}{2\sigma^2(t)}\right)$. Besides, the distance of two neurons is lattice distance that is city-block distance.
For D-2 SOM, it has a similar implementation method with D-1 SOM. The differences are that it requires a **grid matrix** to store the grid location of a neuron, and when the radius is less than 1, it's going to be different (when finding the neighbor neurons). Except for these, the implementation is the same as D-1 SOM.

2. The suitable hyper-parameters

In D-1 SOM. The suitable hyper-parameters for D-1 SOM is: lab_som(data, 20, 5000, 0.1, 10) where $\text{numNeurons} = 20$, $steps = 5000$, $learningRate = 0.1$, $radius = 10$.The result is shown in **Figure-1** below. Firstly, I tried different numbers of neurons from $10, 20, 30, \cdots 1000$ with other parameters fixed. Then I compare the visualization of them, finding that when neurons number is 20, it has a good convergence. Then I set the $startRadius = numNeurons/2 = 10$ to include all neurons. Next, because this algorithm gets input data at random from the training dataset, so we need a big value of training steps. I set training steps from 1000 to 10000, then do experiments, but find that the results are similar. Thinking about the number of training samples which is about 5000, so I choose 5000 as its parameter to randomize all training data as much as possible.

In D-2 SOM. The suitable hyper-parameters for D-2 SOM is lab_som2d (data, 10, 10, 15000, 0.1, 5), where grid_width=grid_height=10, steps=15000, learningRate=0.1, radius=5. Firstly, I tried different grid from $5 * 5, 6 * 6, \cdots 30 * 30$. Then I find 10*10 grid is stable and fast convergence when run it multiple times. Then I set the radius=5. Set the steps from 10000 to 100000, and find steps=15000 has less quantization error. The result is shown in **Figure-2** below.

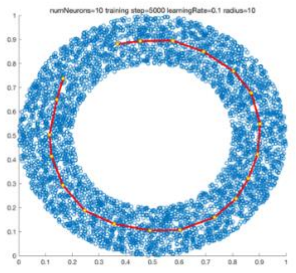


Figure-1 D-1 SOM result

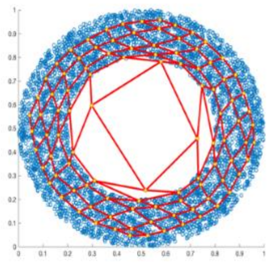


Figure-2 D-2 SOM result

Part 2– Image clustering with SOM

U-matrix representation of the SOM visualizes the distances between the neurons. The distance between the adjacent neurons is calculated and presented with different colorings between the adjacent nodes. A dark coloring between the neurons corresponds to a large distance. The **Figure-3** below is the U-matrix of my **first try**. The quality error is 2.8056 which is high. It shows many clusters, for example, the middle place of U-matrix is a large cluster and they share similar (blue) color that means these pictures have similar characteristics. However, it also shows that all the pictures are not well clustered because the color is basically similar or light (that means neurons are closely not separated, so pictures are not be clustered suitable).

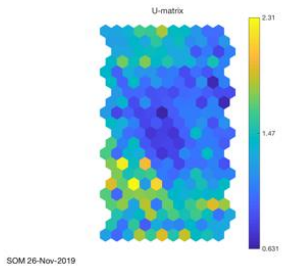
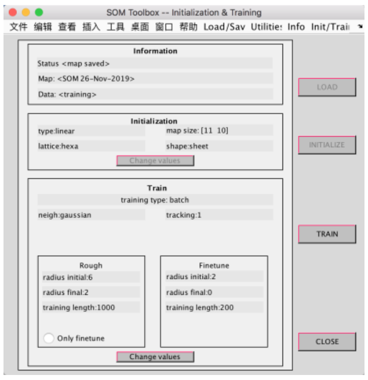


Figure-3 First try

Then I do a lot of experiments, try different parameters and use som_quality(som, training) command to caculate the mean quantization error. Finally, I find a **best parameters setting**, which is shown in **Figure-4** below. The mean quantization error is 1.2979 (the lowest I can find). The U-matrix result of this setting is shown in **Figure-5** below. And when I check it with lab_showsimilar, the classification is very clear, for example, tree pictures, car pictures and shell pictures are all well matched to their clusters.


SOM Toolbox -- Initialization & Training
文件 编辑 查看 插入 工具 桌面 窗口 帮助 Load/Sav Utilitie: Info Init/Trai
LOAD
INITIALIZE
TRAIN
CLOSE
Change values
Change values

Information
Status <map saved>
Map: <SOM 26-Nov-2019>
Data: <training>

Initialization
type:linear
map size: [11 10]
lattice:hexa
shape:sheet

Train
training type: batch
neigh:gaussian
tracking:1

Rough
radius initial:6
radius final:2
training length:1000
Only finetune

Finetune
radius initial:2
radius final:0
training length:200

Figure-4 best parameters setting

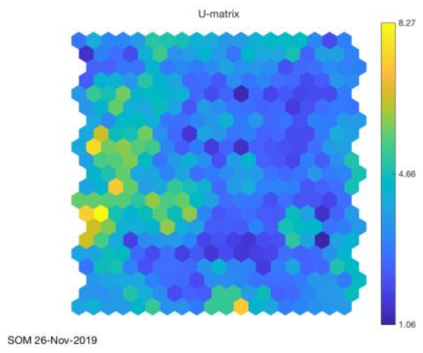
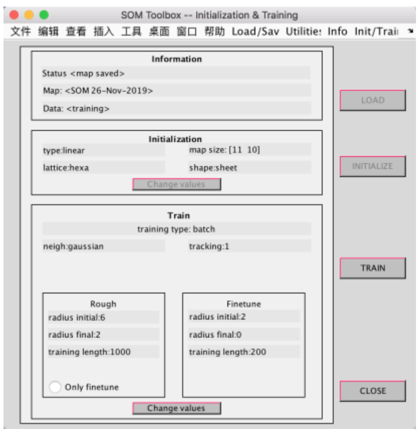


Figure-5 U-matrix of best parameters setting

Part 3– Bonus marks

In this part, I complete the missing code for lab_features.m which are RGB histogram and RGB area. Looking at the comments and code, I find that the function needs five parameters and it missing parameters f4 and f5. The f4 is the fHistogramRGB and f5 is the fAreaRGB. Then we can imitate grey histogram to complete RGB histogram and imitate AreaGrey to complete AreaRGB.
For RGB Histogram. First of all, I define the colors by RGB format[0-255,0-255,0-255,] named rgb_bucket which contains 8 types of color. Therefore, the fHistogramRGB is a 1*8*1 matrix and I create such matrix to store the result. Then I iterate the input image and each pixel's proximity to each bucket is measured. To implement this, I calculate the minimum distance between rgb_pixel and rgb_bucket, in this way, we can find which color it belongs to. And then we can increment the count of that bucket. Finally, we can get the return value: fHistogramRGB.
For RGB Area, I divided the image up into a grid. Then calculate the average color of each grid cell. This is very similar to Greyscale Area.
The reason it performs better is that, by adding rgb histogram and area, the test image is converted into training data with more pixel details that means most of the information of the image is preserved. So when we do SOM, there are more features to be used to help the picture clustered.
Then I try test implement for several times and find the results are better than that with default code given in part 2. The test parameters and U-matrix result are shown below.


SOM Toolbox -- Initialization & Training

Information
Status <map saved>
Map: <SOM 26-Nov-2019>
Data: <training>

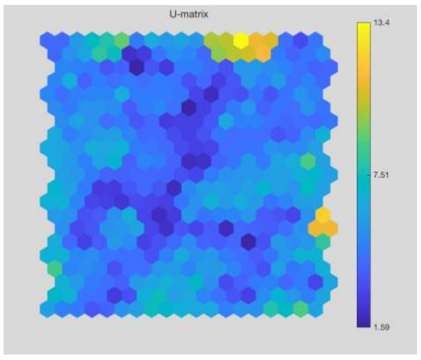
Initialization
type:linear
map size: [11 10]
lattice:hexa
shape:sheet

Train
training type: batch
neigh:gaussian
tracking:1

Rough
radius initial:6
radius final:2
training length:1000
Only finetune

Finetune
radius initial:2
radius final:0
training length:200

文件 编辑 查看 插入 工具 桌面 窗口 帮助 Load/Sav Utilitie: Info Init/Trai
Change values
Change values
LOAD
INITIALIZE
TRAIN
CLOSE


U-matrix