Shaun Howard 9/17/15
EECS 484 PS3 smh150

PS3: Self Ordering Maps

To begin, I have tried many possibilities in terms of neighborhood functions, learning rates, and neighborhood radii. I have also experimented with several different implementations. I have gotten some good results and some bad results. Since there are so many combinations of functions and parameter values, I have simply sampled some of these choices for my experiments. I will now explain some of the aspects of my code.

First off, I did not use the "alphafnc". I had to alter the neighborhood influence in order to experiment and it was much easier to have this organized in one file rather than many. If I had more time, I would have made separate functions for each neighborhood influence that I tried, but given the time constraints and amount that I desired to experiment, I took the action that was most efficient.

I modified "find_closest_cluster" to be called "find_best_matching_cluster" since most literature I could find on SOM's referred to clusters as the "best matching units" or BMU's. I wanted to someone follow the published terms for these such ideas.

A brief explanation for each of the parameters of influence I have and there variations would be the following:

Iterations:

- "curr_iter" is the current epoch the algorithm is running on
- "MAX_NUM_ITERS" is the maximum number of iterations to execute
    - Usually between 50,000 and 1,000,000
    - I started out using 1,000,000 and tried various values
        - Eventually ended up using 100,000 due to time constraints

Find Best Matching Cluster:

- Assure that input vector dimensions match cluster feature vector dimensions by transposing input feature vector (1x64 to 64x1)
- Calculate and store the Euclidean distance from the randomly selected training feature set "invec" to each cluster in "clusters"
- Find the minimum of all stored distances and return its indices as "ibest,jbest"

Learning rate:

- Denoted as "curr_alpha"
- Set to 1 at iteration 0, varies based on type of decay afterwards
- MAX_ALPHA is the initial value for alpha and varies between 0.05 and 0.9 in experiments
- Variations include:
    - Exponential decay
        - `curr_alpha = MAX_ALPHA * exp(-curr_iter/MAX_NUM_ITERS);`
    - Linear decay

```
curr_alpha = 1/curr_iter;
```

- Power-series decay

```
curr_alpha =
MAX_ALPHA*((0.005/MAX_ALPHA)^(curr_iter/MAX_NUM_ITERS));
```

Neighborhood Radius:

- Denoted as "curr_radius"
- Initialized to "MAX_RADIUS," which is the initial value for the neighborhood radius
  - Decays based on variation used
- "MAX_RADIUS" was typically chosen as the max(nclustrows, nclustcols)/2
  - i.e. for 6x6 grid, this would evaluate to max radius of 3.
- Variations include:
  - Exponential decay

```
r_time_constant = MAX_NUM_ITERS/log(MAX_RADIUS);
curr_radius = MAX_RADIUS * exp(-curr_iter/r_time_constant);
```

  - Linear decay

```
curr_radius = MAX_RADIUS * (1 - curr_iter/MAX_NUM_ITERS);
```

Neighborhood Function:

- Calculated for each cluster
- Vector containing specific function values denoted as "neighborhood"
  - An nclustrows x nclustcols vector to store all neighborhood function outputs for cluster at i, j
- Several variations, most have the following logic structure
  - If cluster is the best matching unit, neighborhood(i, j) = 1
  - Else, calculate value based on variation type (listed next)
- Some of the variations of the neighborhood function I used were the following:
  - Bubble
    - This method is similar to cut Gaussian below, basically an all or nothing influence of the best matching unit on neighbors within the "curr_radius".
  - Gaussian

```
dist_from_bmu=(ictr - i)^2 + (jctr - j)^2;
neighborhood(i,j)=exp(-dist_from_bmu/(2*variance));
```

  - Epanechnikov

```
dist_from_bmu = pdist([ictr,jctr;i,j], 'euclidean');
neighborhood(i,j)=max(0,1-(curr_radius-dist_from_bmu)^2);
```

  - Cut Gaussian

```
dist_from_bmu = pdist([ictr,jctr;i,j], 'euclidean');
if dist_from_bmu < curr_radius
    neighborhood(i,j)=exp(-dist_from_bmu/(2*variance));
else
    neighborhood(i,j)=0;
end
```

Neighborhood Influence:

- Written in many different ways in literature, but I call the feature update "learning function" this
- This is used in the cluster updating phase
- The following code is an example of what I mean:
  - ```
    curr_alpha*neighborhood(i,j)*...
    (training_features-curr_features);
    ```
- Hence, it is the product of the learning rate and the neighborhood function and the difference between the randomly selected training features and the current cluster's features. Next, I will cover the feature update process which includes this.

Feature (Weight) Updating Phase:

- For each cluster, update its feature vector based on its grid location and the neighborhood influence on it at that time and location in space.
- The neighborhood influence, as previously discussed, is added to the current cluster's feature vector. Hence, the cluster's features are stored as "updated_features" with the new values, influenced by the best matching unit's features.
- "updated_features" is then normalized and the cluster is updated with the new features

The basic gist of the algorithm I have written for SOM is the following:

1. Initialize clusters to random values between 0 and 1
2. Normalize each training pattern (code provided)
3. While current iteration is less than maximum number of iterations
    a. Calculate the current radius
    b. Calculate the variance
    c. Calculate the learning rate
    d. Randomly select a feature set to train from
    e. Find the best matching cluster to the random training feature set
    f. For each cluster, calculate the neighborhood function and store output
    g. For each cluster, calculate and store its new features based on current time and location in space relative to the best matching cluster
        i. Normalize the new feature values and update the cluster with these values
    h. Increment iteration counter

Experiments:

As previously mentioned, I have experimented with several different types of variations. I will try my best to list and explain those I think were valuable. To come is a table of the experiments I performed, the variations of decay used, and the initial parameters given (to the best of my memory).
    ```

Trial #	Grid Size	# iters	Alpha(0)	Alpha type	Radius(0)	Radius type	Neighborhood type
1	6x6	1,000,000	0.1	Exp. decay	3	Exp. Decay	Bubble
2	6x6	500,000	0.05	Exp. Decay	3	Exp. Decay	Bubble
3	8x8	500,000	0.05	Linear Decay	3	Exp. Decay	Bubble
4	6x6	500,000	0.1	Exp. Decay	3	Exp. Decay	Gaussian
5	6x6	50,000	0.1	Exp. Decay	3	Exp. Decay	Gaussian
6	6x6	250,000	0.25	Exp. Decay	2	Exp. Decay	Gaussian
7	6x6	250,000	0.5	Linear Decay	3	Exp. Decay	Gaussian
8	6x6	115,000	0.5	Power Series Decay	3	Exp. Decay	Epanechnikov
9	6x6	100,000	0.5	Power Series Decay	3	Linear Decay	Cut Gaussian
10	8x8	100,000	0.5	Exp. Decay	3	Linear Decay	Cut Gaussian

The difference between 4 and 5 is that in 5 I have normalized feature values. I do this from experiment 4 on. The output graphics do not change very much, just the scale of the graphs changed.

Analysis:

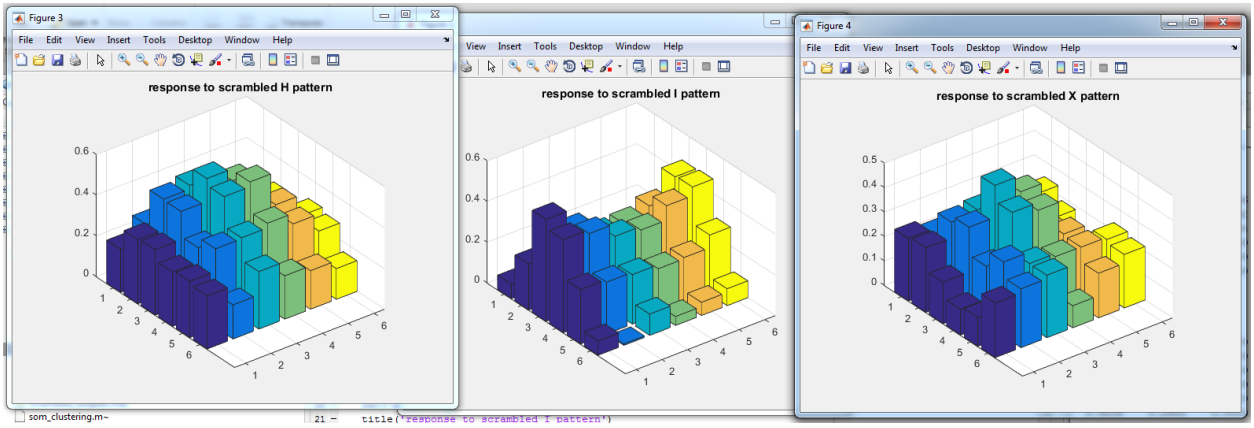
I have visualizations of the graphs for each trial. I will include those that I believe to be worthwhile.

These visualizations are on the next few pages.

Shaun Howard
EECS 484 PS3

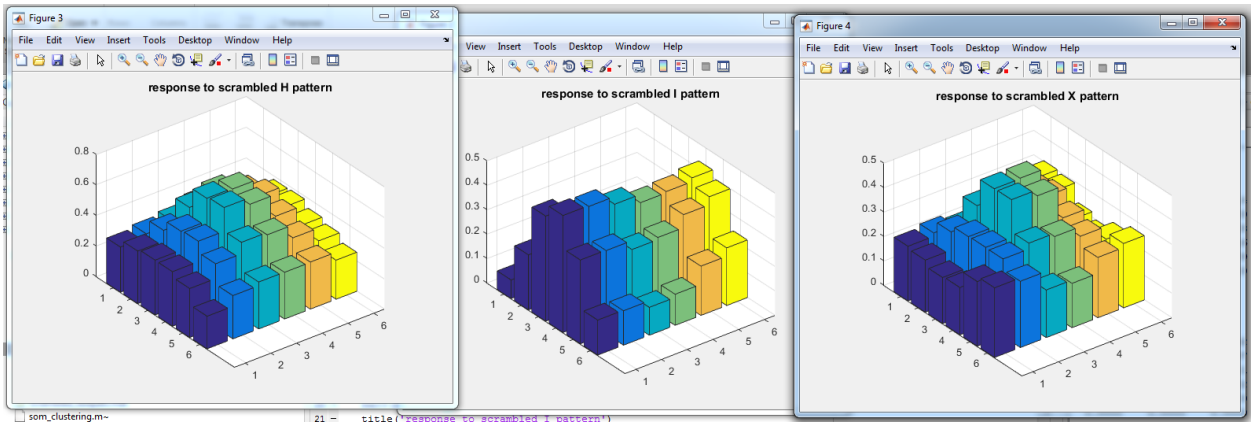
9/17/15
smh150

Trial 1:



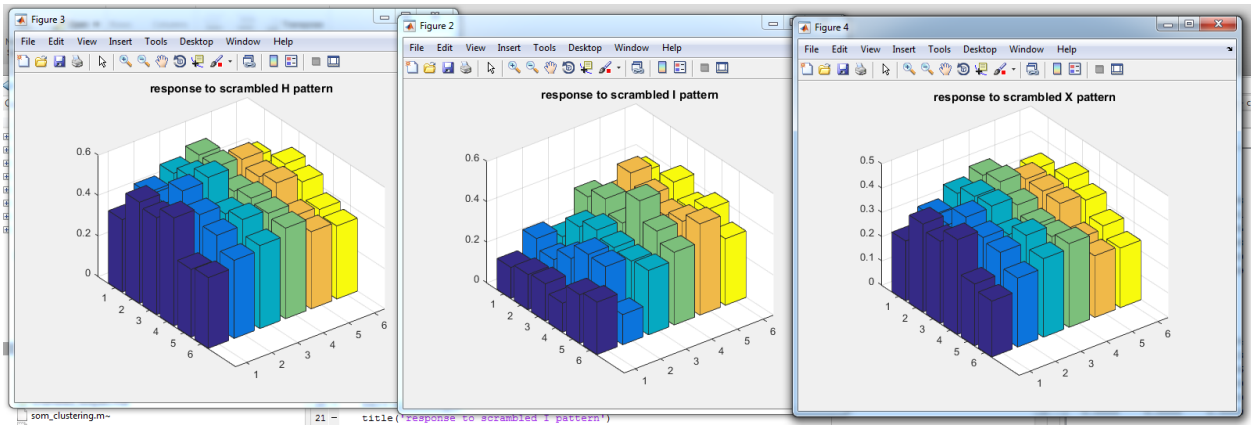
For what it was, my first trial went fairly well. After 1,000,000 iterations, it seems we can notice the I and some features of the H and X. Notice the "bubbly" nature of the distribution.

Trial 6:



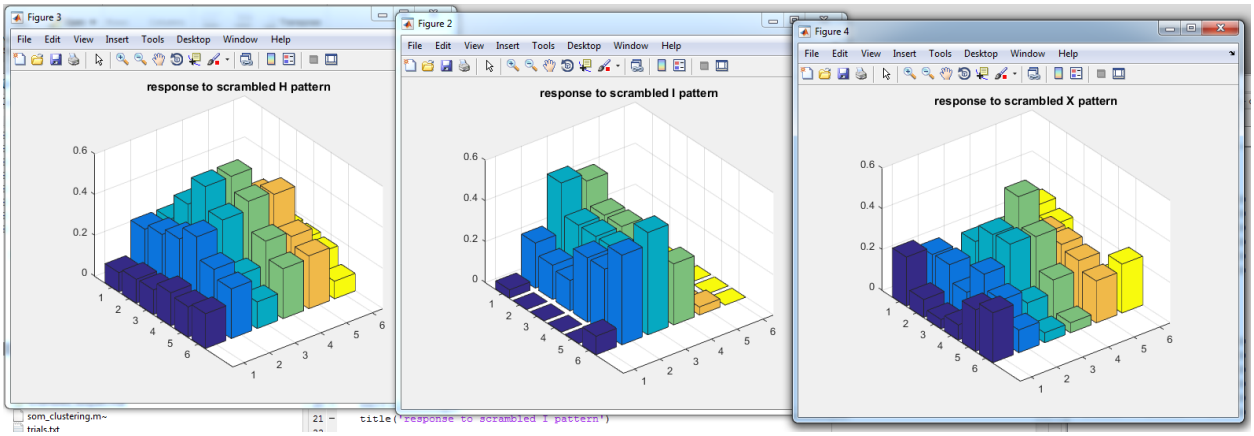
This trial was a more distinctive version of the first trial, but with a Gaussian neighborhood. The I and the X are looking better, but the H is hard to find.

Trial 8:



Here I chose to explore with new types of decay and neighborhood influence functions. The outputs look more determined for the X, but the others would probably benefit from more iterations or parameter tweaks.

Trial 9:

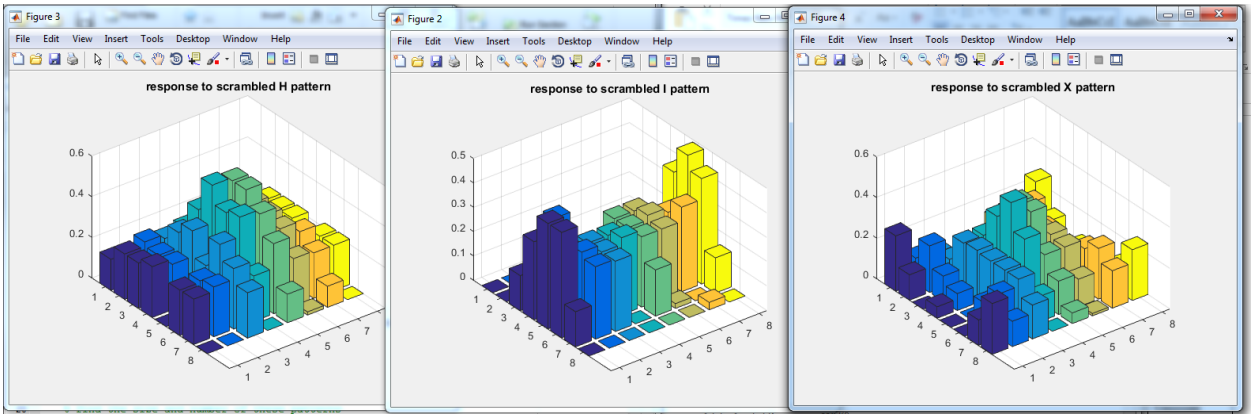


Now we are talking. We can start to see the outlines of the letters in these graphs. We can see the middle brace of the H and the sides being determined. The I is distinct. The X edges stick up along with the middle protruding, while the empty areas are closer to empty than previous trials.

Shaun Howard
EECS 484 PS3

9/17/15
smh150

Trial 10:



This, along with trial 9, was one of the best trials. It looks like the I is definitely there. The X again has edges and the middle defined, while the crosses would benefit from more iterations. The H looks better than most, but is still fuzzy.

Discussion:

Altogether, it seems like SOMs are good for determining patterns that are obvious, but if you mix the features and their locations up, the SOMs may only focus on one. This shows the competition between clusters. It seems as if the "I" was the easiest to depict, while the X was harder and the H was the hardest.

I think experimenting with values was fun and interesting, but I would still like to see better results. I think adding heuristics would make this run much faster and better, but it might be more complicated and specific per domain. We can see that Gaussian neighborhoods with an exponential decay of radius and learning rate seem to be most common and work best. The last few experiments with linear radius decay also went well. The cut Gaussian method is probably the best for neighborhood function.

Altogether, rounding out to 100,000 iterations and alpha of 0.5 seemed to work best. Initial radius was also good at half of the largest dimension of the grid. Overall, I think my results were good, but could have been better with more iterations and experimentation.