SPARSE CODING LAB 5 SOLUTIONS

MAYUR MUDIGONDA

1. Sanger's Rule on Lines World

Sanger's rule basically updates the weight matrices thusly

$$\triangle w_{ij} = \eta y_i (x_j - \sum_{k=1}^{i} y_k w_{kj}) \tag{1}$$

A few things to recap about this model is that - (a) it is not neurally plausible because of the sequential nature of it's update steps. This is because the second neuron codes for what the first neuron has not accounted for and is thus harder to realize neuro-biologically. (b) What Sanger's rule codes for is equivalent to computing the principal components of the data. From the literature and class slides, we know that the principal components on natural scenes does not code all possible variances that we see in our data.

A WTA network would also not code sufficiently interesting features as evident in Figure[1]. The way to think about it perhaps is to go back to the k-means type analogy. The WTA network will basically create k-cliques that it thinks best represents the data over the space of all patches in natural scenes. The representative feature of each clique being it's centroid. This is not entirely meaningful as it would result in line-like weight spaces but it will not tease out all possible "types" of lines that we care about, i.e. horizontal, vertical and groups of these, etc.

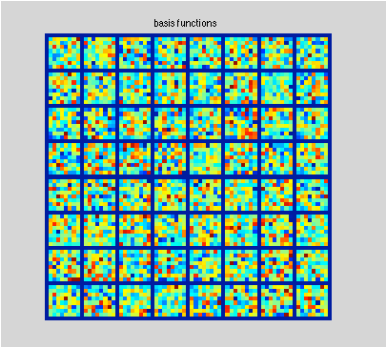


Figure 1. The weight's learnt (64 basis functions) using Sanger's rule.

2. Sparse Coding of Binary Patterns

Foldiak's model of sparse coding basically has three sets of parameters - feedforward weights - Q, lateral weights - W and thresholds - T for each output unit. We update these with the following rules

$$\triangle w_{ij} = -\alpha(y_i y_j - p^2) \tag{2}$$

$$\triangle q_{ij} = \beta y_i(x_j - q_{ij}) \tag{3}$$

$$\triangle t_i = \gamma(y_i - p) \tag{4}$$

Further, we set the value of $\triangle w_{ij} = 0$ when $i = j$

In Figure [2] we see that the system learns at least some of the possible line orientations. As we increase the codebook size to 16 (see Figure[3]) this behavior continues to scale but we also see combinations of lines in our dictionary. Finally, with 32 elements in our dictionary(see Figure[4]) we see that some elements try to learn pairs of lines but are conflicted with other dictionary elements that have learnt those lines.

My results do not show a converged solution, but tweaking the knobs of the system can result in cleaner solutions that some students in the class managed to do. With regards to the learning rate, the system should have a slower forward learning rate because we want the system to learn and update it's self but only after the system has realized what other neurons are coding for. This cannot be achieved if we have a higher forward learning rate.

3. Sparse coding on Natural Scenes

In this problem, we use the model defined by Rosell et al. (2008). The idea behind this model is that when we start with a random dictionary and penalize the system to have minimal activations through the dynamics defined in the assignment, we end up with a dictionary that represents the receptive fields that we see in the visual cortex (V1). LCA is one way to achieve sparsity, through local competition and dynamical systems.

Figure[5] shows a 256 element dictionary of sparse codes with $\lambda = 0.1$. Note how it differs from other sparse code books in the document. You can see more tiling of space and orientation in this code book. Further, you can see hints of more complicated patterns as well.

Figure[6] , Figure[7] , Figure[8],Figure[9] and Figure[10] shows a 64 element sparse code with varying values of λ, which is set at 0.01, 0.1, 0.2, 0.3 and 1 respectively. An interesting pattern that can be observed is that at smaller λ values there is more noise in the dictionary while at larger values the edges are more pronounced.

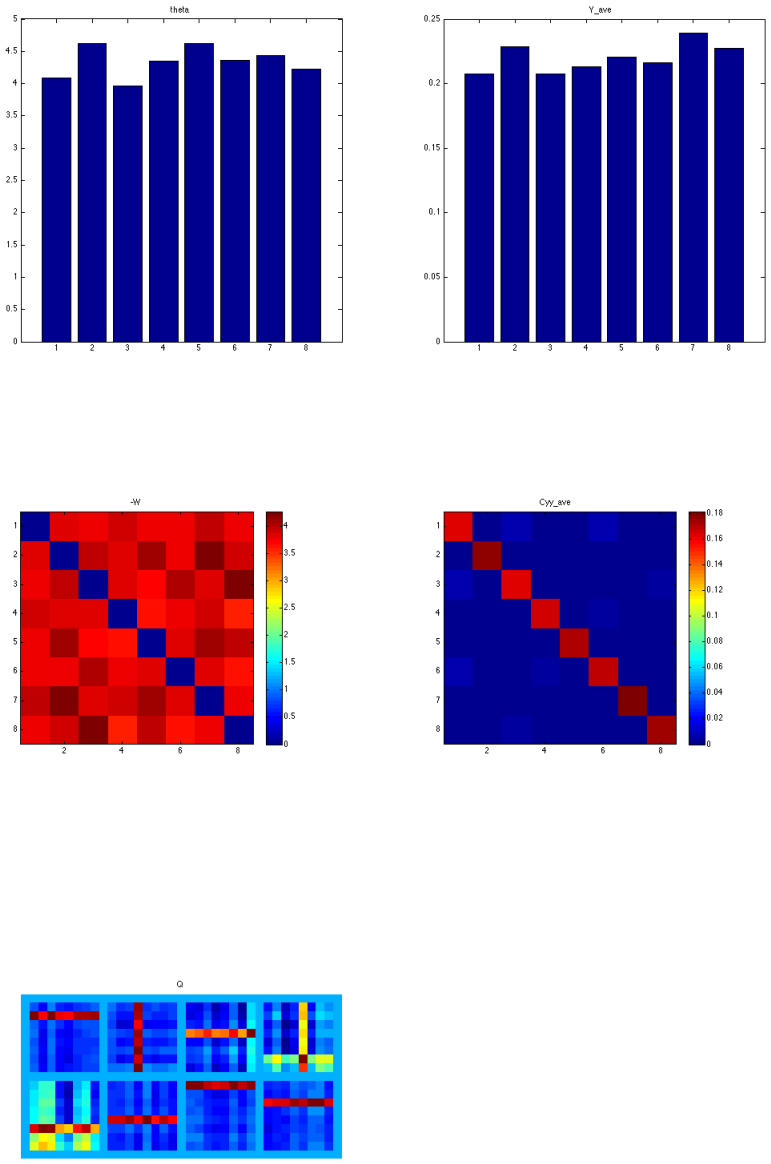


FIGURE 2. Foldiak's binary sparse coding using 8 dictionary elements. Top left, theta values. Top right, sparsity. Middle left, Weight (lateral). Middle right, $C_{yy}ave$. Bottom left, feedforward weights Q

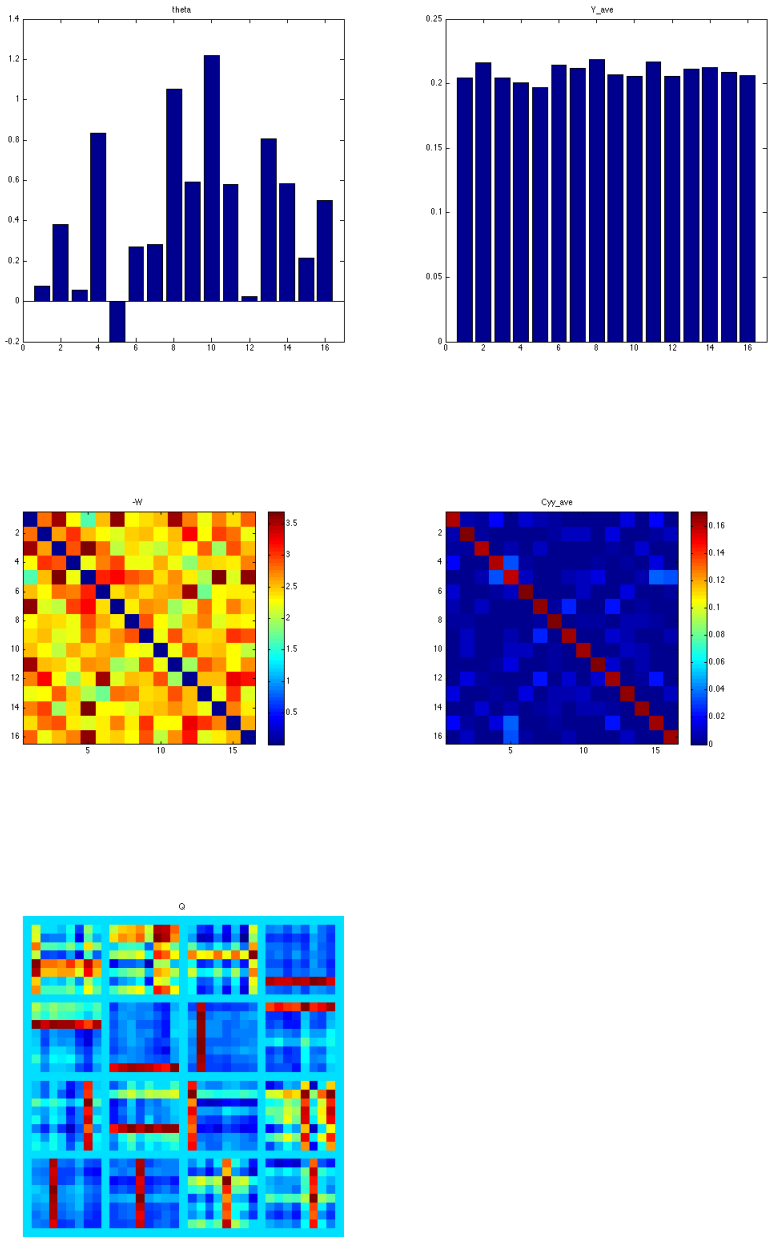


FIGURE 3. Foldiak's binary sparse coding using 16 dictionary elements. Top left, theta values. Top right, sparsity. Middle left, Weight (lateral). Middle right, $C_{yy}ave$. Bottom left, feedforward weights Q

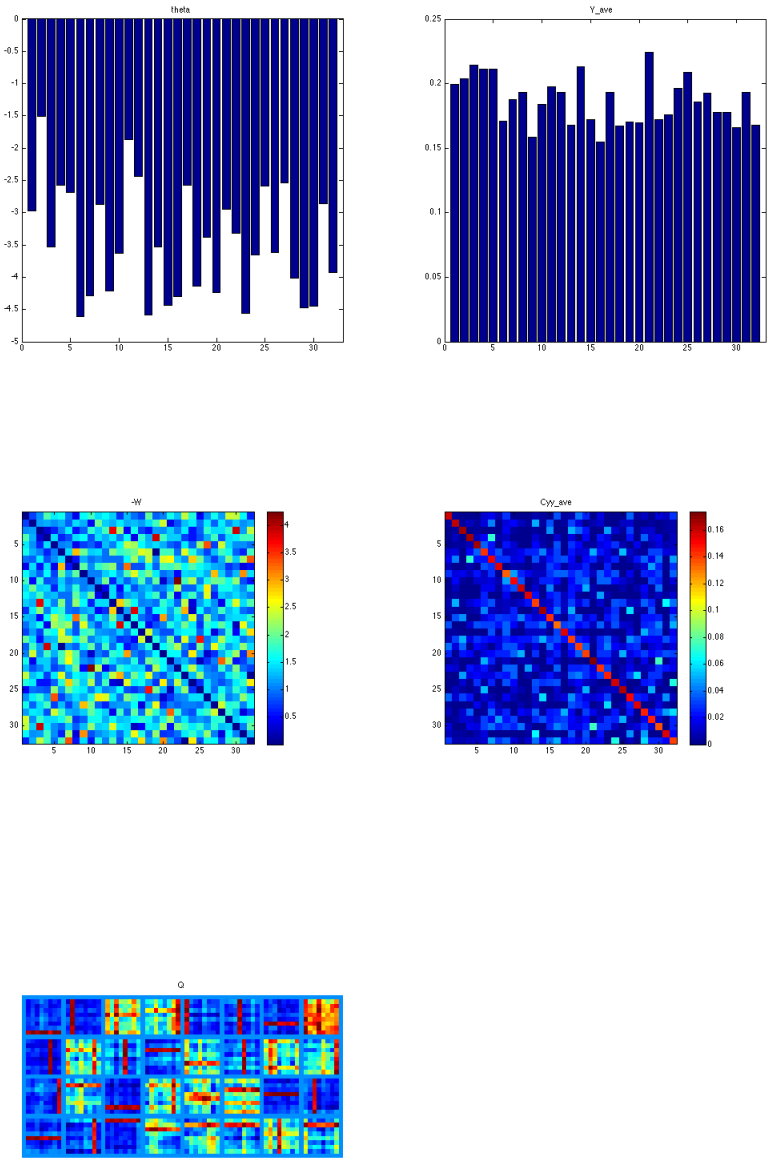


FIGURE 4. Foldiak's binary sparse coding using 32 dictionary elements. Top left, theta values. Top right, sparsity. Middle left, Weight (lateral). Middle right, $C_{yy}ave$. Bottom left, feedforward weights Q

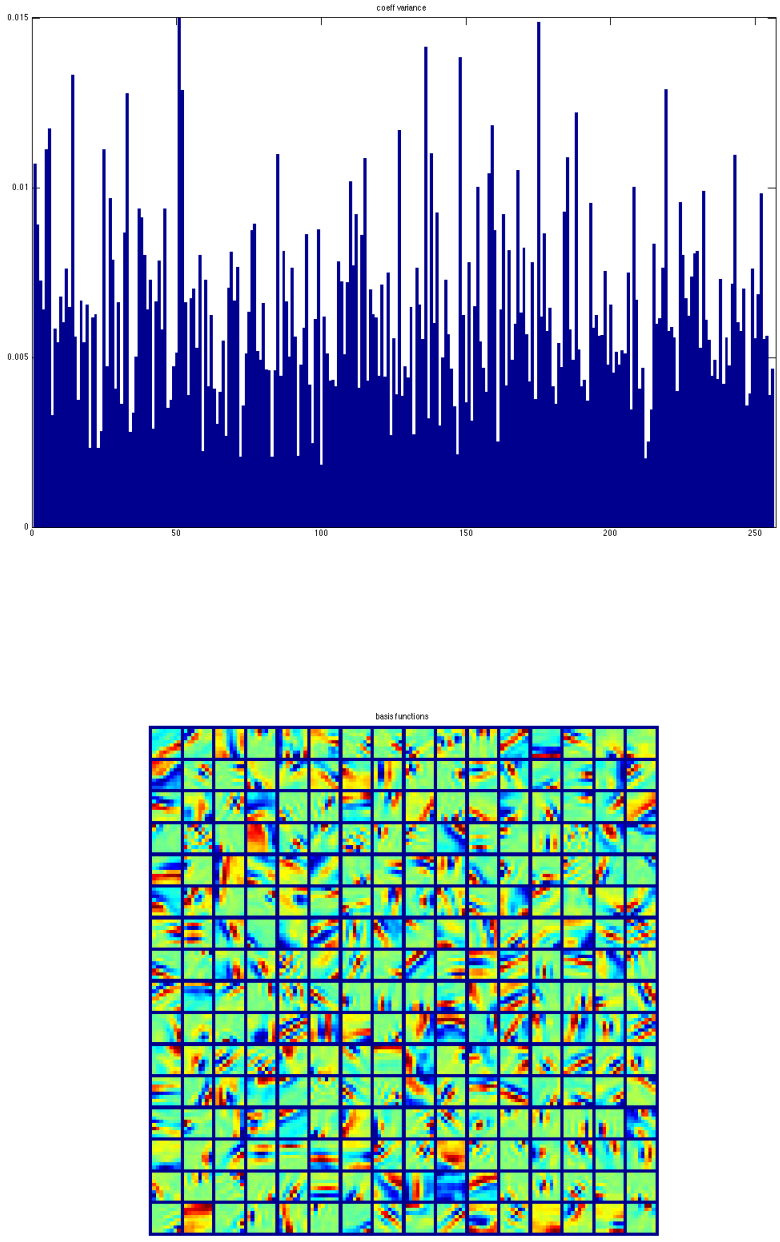


FIGURE 5. top, activations of code book. bottom, 256 element codebook on 8 x 8 patches

HW 5

Bharath Hariharan

October 14, 2012

1 Q1

1. Figure 1 shows the learnt weights using PCA (Sanger's rule) and WTA. PCA gives mostly noise, with nothing really related to the structure in the data, while WTA, although it captures something like lines, doesn't learn to separate the vertical and horizontal lines. The failure of both these methods is because their modeling assumptions are inconsistent with what the data actually is. The data is produced by picking up a small number of horizontal lines, a small number of vertical lines, and putting these together. WTA on the other hand makes the assumption that each data point is just a small perturbation away from some basis element; it has no notion of capturing combinations of the basis elements and it therefore needs to capture a separate cluster center for each set of lines chosen. PCA does capture the fact that each data point is a linear combination of the basis elements, but it does not capture the fact that each data point is constrained to choose only a few basis elements. As such the basis elements are all mushed up combinations of lines.

2. Figure 2 shows the learnt weights using Foldiak's network for 8,16 and 32 outputs. With 16 outputs, the lines pop out effortlessly. What is reassuring is that even with 8 or 32 outputs, the lines do pop out. With

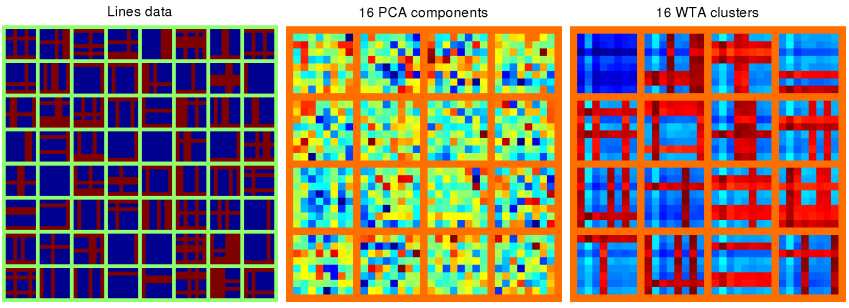


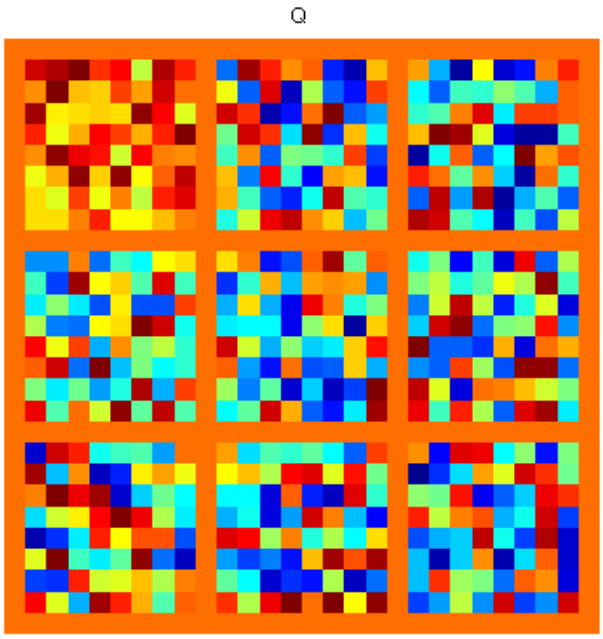
Figure 1: From left to right: Sample lines data, 16PCA components and 16 WTA clusters

Bill Sprague
VS265 Lab 5
Oct. 15, 2012

Sparse Coding of Binary Patterns

A) Sanger's Rule

Trying to code this data with a set of linear neurons using Sanger's rule won't find the higher order structure. The figure below shows the attempt with 16 neurons, and it appears to find no structure at all. This was run on 9000 iterations of 100 data points each. Although it's possible it failed to converge in that time, Sanger's rule is ill-equipped to find the causes of the data (i.e. randomly positioned lines) because it is limited to 2-point statistics. Finding the pattern of lines requires at least 3 points to be certain of collinear correlations. WTA would not fair much better, because it will tend to find clusters of line combinations that look mostly like weird amalgamations of several input patterns.



B) Foldiak's Network

The figure below shows the results of training on Foldiak's network of anti-Hebbian horizontal connections, W, feed-forward Hebbian connections, Q, and a nonlinear threshold, theta. The learning rates for these were .1, .01, and .1 respectively for 4000 iterations of 100 images each. The learning rate for the feed-forward connections should be lower than the others so that the feed-forward connections are nearly constant on the time-scale the other connections are learning on. The learning rules are:

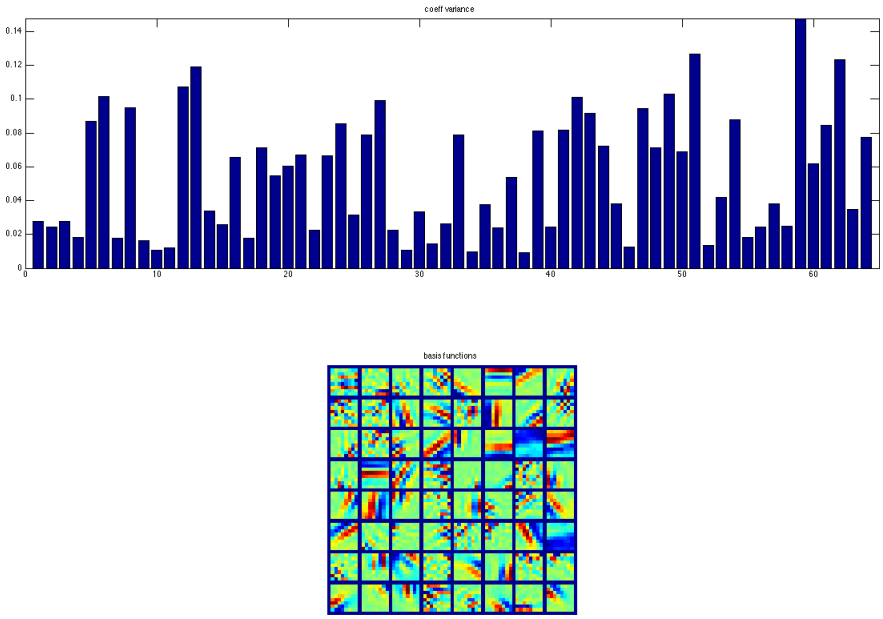


FIGURE 6. top, activations of code book. bottom, 64 element codebook on 8 x 8 patches, and lambda value = 0.01

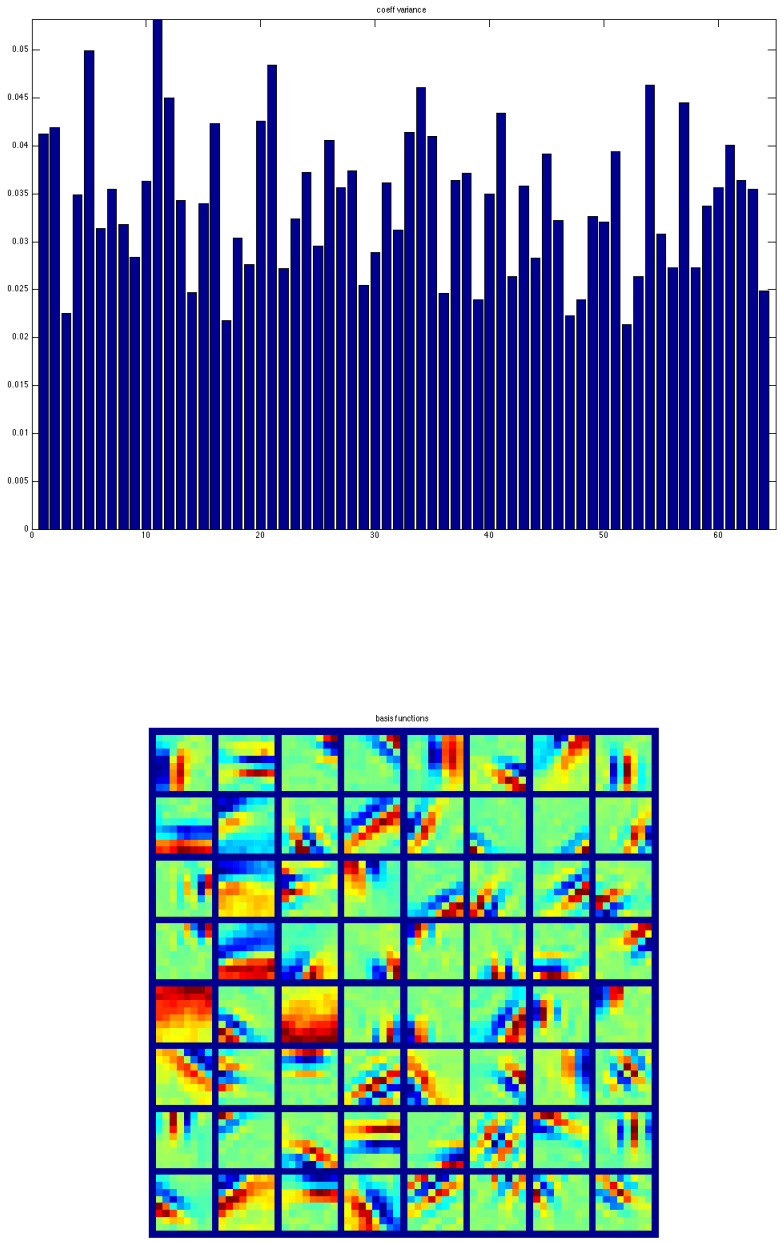


FIGURE 7. top, activations of code book. bottom, 64 element codebook on 8 x 8 patches and lambda value = 0.1

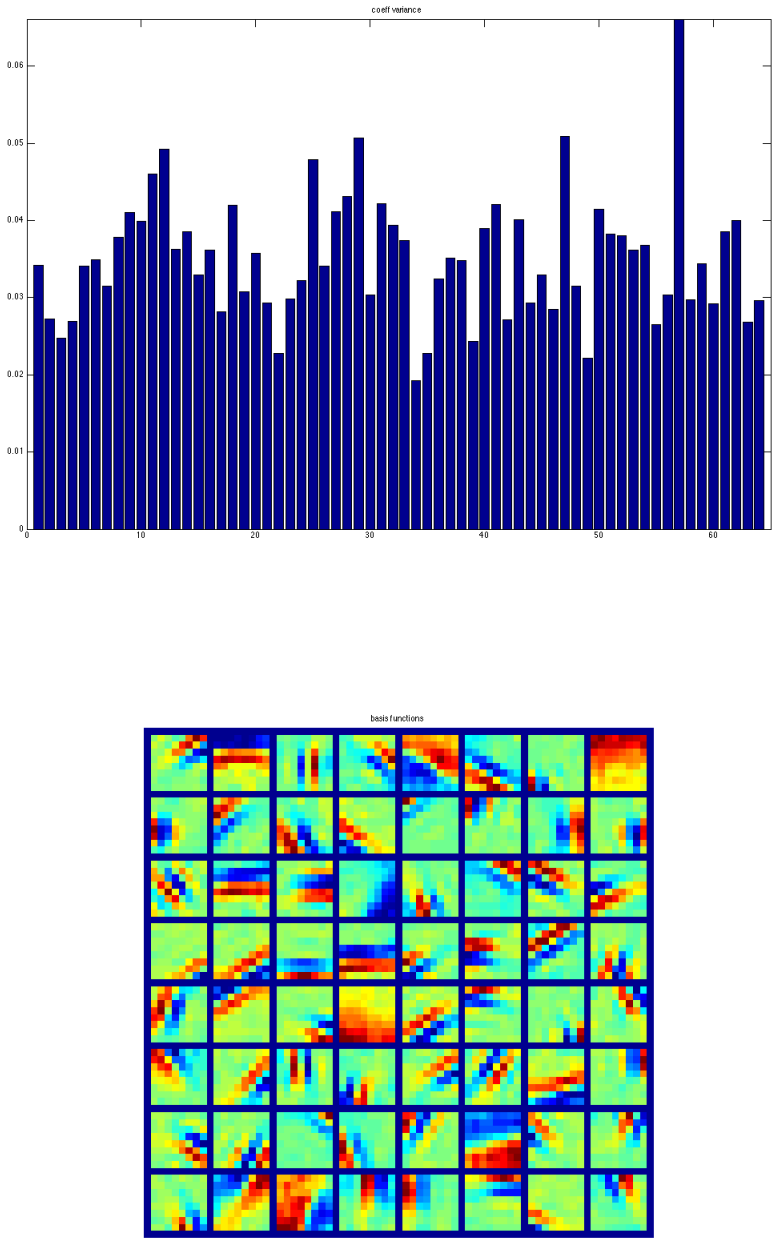


FIGURE 8. top, activations of code book. bottom, 64 element codebook on 8 x 8 patches, and lambda value = 0.2

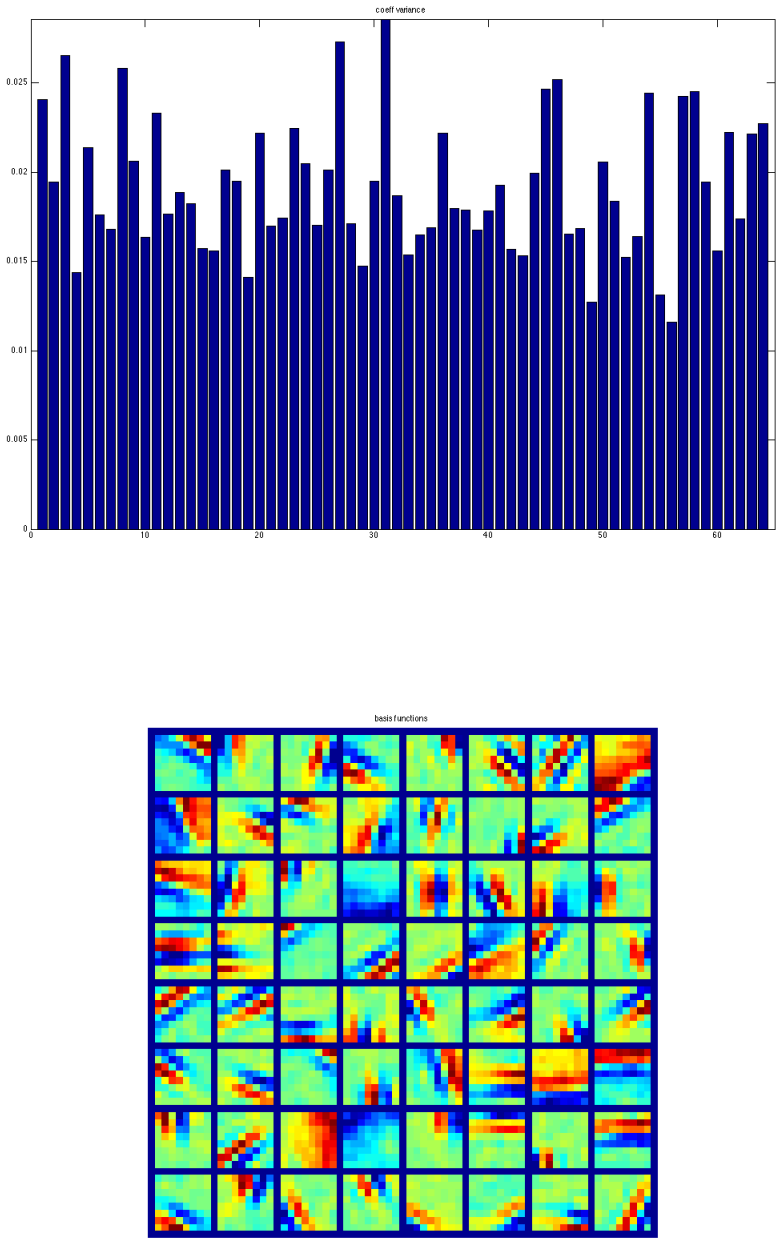


FIGURE 9. top, activations of code book. bottom, 64 element codebook on 8 x 8 patches, and lambda value = 0.3

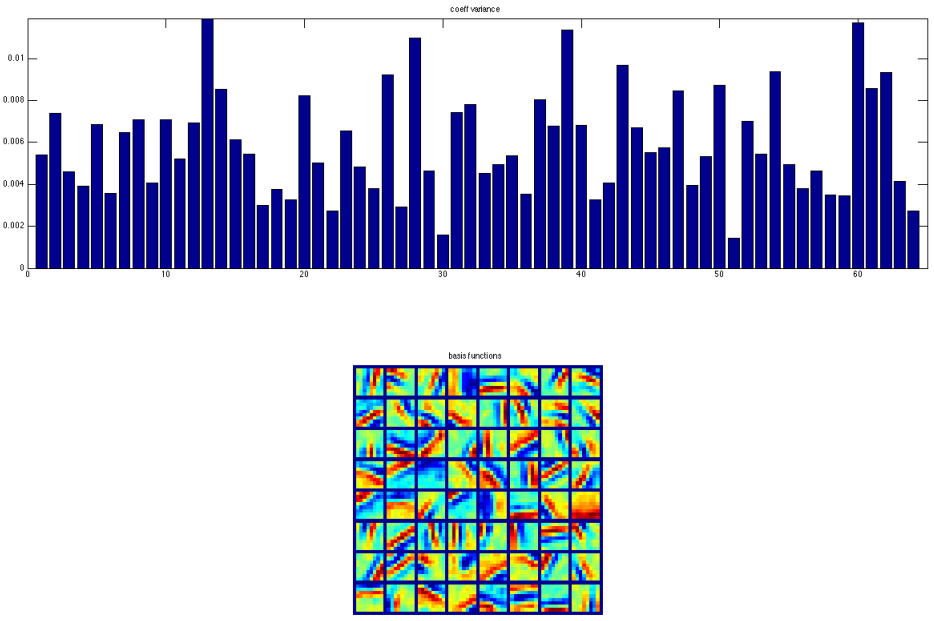


FIGURE 10. top, activations of code book. bottom, 64 element codebook on 8 x 8 patches, and lambda value = 1