Project 2

Name: Kartik

NetId = kk1478

Course N0. = ECE-8493

Theory

Three strategies used in this project are:

Common function used in three of the processes are normalized, decode, newData.

Normalized():

- Reshape the image from [x,y,z] to [x*y,z]
- imgMean = column-wise mean
- Subtract the mean (Column-wise)

newData = eigen Vectors' * normalized'

Decode():

- Get the row-wise data = (v'*newData)'+imgMean(column-wise mean)
- Then reshape the data to image format i.e back to [x,y,z]

PCA using statistical batch method to compute and display the three PCs.

Algorithm:

- Normalized()
- Covariance of the normalized data
- Eigien value and vectors from the covariance matrix
- Sort the eigen vectors based on the eigen values
- Using the top n eigen vectors get back the data in original dimension by calling decode()

Files associated are: tradPca.m
Normalised.m
Decode.m

PCA using Hebbian Learning rule.(3 output neurons)

Algorithm:

- Initialize the weights of size (3,3)(normalized and uniformly distributed) with the help of init_weights function can be found in init_weights.m file.
- Get the normalised data using the normalized function discussed above.
- Calling the update weight method

```
LT = tril(y*yt);
a = y.*repmat(x,[1 pca])';
b = LT*W;
c = W+ lr/iterations*(a - b);
weights = c;
```

This the heart of the algorithm i.e code for updating the weights

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta\left(\mathbf{x}(n)\mathbf{x}^T(n)\mathbf{w}(n) - \mathbf{w}^T(n)\mathbf{x}(n)\mathbf{x}^T(n)\mathbf{w}(n)\mathbf{w}(n)\right)$$
$$= \mathbf{w}(n) + \eta\left(\mathbf{\Sigma} - \mathbf{w}^T(n)\mathbf{\Sigma}\mathbf{w}(n)\mathbf{I}\right)\mathbf{w}(n)$$

This equation is converted in the code shown above.

PCA using Hebbian Learning rule (1 output neurons)

Algorithm:

- Initialize the weights of size(1,3) with the help of init_wright function.
-

```
delW = lr*y*(x-(W*y));
weights = W+delW;
output = y;
```

As the output neurons is only one therefore x' = x and the weight update step is very simple

As there are is only one neuron and we want to find multiple PCs using this neuron we have to do some pre processing. Once the weights converges we will delete that PC from the input data with the help of equation given below.

New_Input = current_input*(I – VV')

Then out new input to the network will not contain that principal component. We will repeat this step till the desires PCs are found.

Results

Original Image

PCA using original statistical batch method

```
v =

    0.5360   -0.7914   -0.2939
    0.5860    0.0982    0.8043
    0.6077    0.6033   -0.5164
```

Eigen Vectors

```
1.0e+04 *

    2.0125
    0.0208
    0.0005
```

Eigen Values

Image using first PCA

Image using first 2 PCA

Root Mean Squared Error between original image and

PCA using Hebbian Learning rule.(3 output neurons)

0.5421	0.7685	−0.1660	101.4425
0.5839	−0.0843	0.0341	−0.7294
0.6043	−0.5666	0.1142	0.5934

Eigen Vector

Eigen Values

Using 1st PC

Using 2nd PC

PCA using Hebbian Learning rule (1 output neurons)

```
                                                ens =

-0.5421    -0.5839    -0.6043                  -101.4425
-0.5946    -0.5737    -0.5634                  -202.2726
-0.7979     0.1321     0.5881                    13.6061
```

Eigen Vector

Eigen Values

Using 1St PC

Using 2nd PC

Using 3rd PC

RMS
Error between each PC between statistical and habbin learning 3 output neuron

1st Pca immse = 1.7873e-05
2nd PC immse = 1.2785
3rd PC imsse = 0.3358

As we can see higher the PC is less the error is i.e habbin learning will start giving more error as we will increase the number of output neruons therefore habbin learning is useful for finding the less number of PC, we don't have to perform operations on the whole image.

RMS between Original image and the image generated with PCs

Between Statistcal 1st PC	69.9698
Between statistical 1st and 2nd PC	1.8209
Habbin learning with 3 output neurons and 1st PC.	69.897
Habbin learning with 3 output neurons and 1st and 2nd PC	5.485
Habbin learning with 1 output neurons and 1st PC.	69.8966
Habbin learning with 1 output neurons and 1st&2nd pc	1.0964e+03
Habbin learning with 1 output neurons and all pc	1.0175e+03

Habbin learning with 1 output nuron gave very intersting result as per my code. The 1st and 2nd PCs were very similart and the 3rd PC was very similar to the 2nd PC of the stastical method. So if we want first 3 PCs with habbin learning with one output neuron we have to run the algorithm 6 times.

Below are the algoritm arranged in the basis of runtime from fastest to slowest.

Stastical method < habbin learning 1 neuron < habbin learning with 3 neurons

Code

```
function [W_new] = init_weights(pcs,ev)

W_init = normrnd(0,0.5,[pcs,ev]);
W_new = W_init./vecnorm(W_init,2,2)

end
```

```
function [im,rowData] = decode(v,newData,imgMean,x,y,z)
    disp(size(v))
    rowData = (v'*newData)'+imgMean;
    im = reshape(rowData,[x,y,z]);
    im = uint8(im);
    imshow(im);
end
```

```
function [norm,imgMean,x,y,z] = normalised()

rawImg = double(imread('kk1478.jpeg'));

[x,y,z] = size(rawImg);
img = reshape(rawImg,[x*y,z]);

imgMean = mean(img,1);

norm = img-imgMean;

end
```

For habbin learning with one output neuron

```
[norm1,imgMean,x,y,z] = normalised();
norm = norm1;
eta = 1e-9;

pcs = 1;

[W_new] = init_weights(pcs,3);
evi = 0;
out = 0;
evs = [];
ens = [];
itr = [15,200,150]
prevEV = 0;
for j = 1:3
    for i = 1:itr(j)
        for idx = 1:x*y
            [W_new,output] = updateWOne(eta,norm(idx,:) , W_new, 1,pcs);
        end
    end
    rd = (W_new'*W_new);
    rd = rd + prevEV;
    rd = eye - rd;
    norm = norm*rd;
    prevEV = rd+prevEV;
    evi = W_new;
    out = output;
    evs = [evs;evi];
    ens = [ens; out];
    disp(j)
end

pcs = 1
newData = evs(1:pcs,:)*norm1';
[im2,rowData] = decode(evs(1:pcs,:),newData,imgMean,x,y,z);

imshow(im2);
```

```
[norm,imgMean,x,y,z] = normalised();

co = cov(norm);

[v,n] = eig(co);

[d n] = sort(diag(n),'descend');

v = v(:,n);

pcs = 2;
newData = v(:,1:pcs)'*norm';
[im,ro] = decode(v(:,1:pcs)',newData,imgMean,x,y,z);
```

Update weight for habbin learning with 3 output neuron

```
function [weights,output] = updateW(lr, x, W, iterations,pca)
    y = W*x.';
    y = reshape(y,[pca,1]);
    yt = reshape(y,[1,pca]);
    LT = tril(y*yt);
    a = y.*repmat(x,[pca 1]);
    b = LT*W;
    weights = W+ lr/iterations*(a - b);
    output = y;
end
```

Update weight for habbin learning with one output neuron

```
function [weights,output] = updateWOne(lr, x, W, iterations,pca)
    y = W*x.';
    delW = lr*y*(x-(W*y));
    weights = W+delW;
    output = y;
end
```

```
[norm,imgMean,x,y,z] = normalised();

eta = 1e-9;

pcs = 3;

[W_new] = init_weights(pcs,3);
for i = 1:1000
    for idx = 1:x*y
        [W_new,output] = updateW(eta, norm(idx,:), W_new, 1,pcs);
    end
end
pcs = 3
newData = W_new(1:pcs,:)*norm';
[im1,rowData] = decode(W_new(1:pcs,:),newData,imgMean,x,y,z);

imshow(im1);
```