EECE5644 Spring 2020 – Take Home Exam 4

Chenglong Lin
lin.chengl@husky.neu.edu
NU ID: 001024390

April 7, 2020

1 Question 1

1. **Neural Network Structure**
Below is the figure of the neural network I implemented:

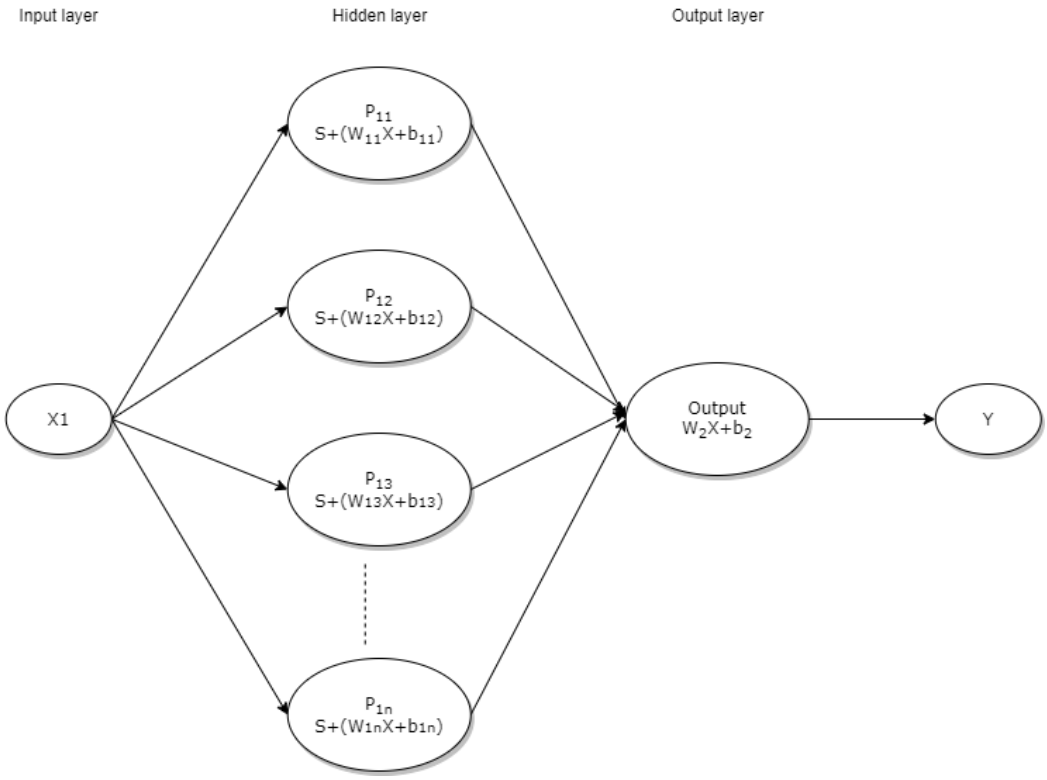

Figure 1: Neural network structure

In this neural network, it has 1 input layer, 1 fully connected hidden layer with variable number of perceptrons from the set [1,2,...,11,12], and 1 output layer. The hidden layer uses softplus activation (S+). The output layer uses linear activation with coefficient of 1. In figure 1, X1 is the input, the P1x's are perceptrons of the hidden layer, and the Y is the output. The weight and bias of each perceptron is label as W_{xx} and b_{xx}. The weights and bias are initialized to zeros.

2. **Forward Propagation and Prediction**

The forward propagation of the neural network is calulated as following:

$$\vec{X} = [\vec{1} \;\; \vec{X}1] \;\; \vec{Z} = \vec{X}(\vec{W}_1)^T \;\; \vec{H} = S + (\vec{Z}) \;\; \vec{H} = [1, \vec{H}_1]$$
$$\hat{Y} = \vec{H}(\vec{W}_2)^T$$

Where all variables above are matrix. By plug in all samples into $\vec{X}$, we can forward propagate all samples at once. For prediction of a single sample,

plug x1 into $\vec{X}$, forward propagate, and then $\hat{Y}$ would be the output.
The following is the mathematical expression of the softplus function/S+.

$$Softplus(x) = ln(1 + e^x)$$

3. **Loss function**
In this problem, I am using mean-square-error loss function because we are dealing with regression, not classification. The following is the loss function for mean-square-error loss [1].

$$J = \frac{1}{m}\sum_{i=1}^{m}(Y - \hat{Y})^2$$

where m is number of samples, Y is original value this model is trying to map to, and the $\hat{Y}$ is the value predicted by the model.
4. **Backward Propagation**
To implement the gradient descend, I need to implement the back propagation to compute the gradient of the loss function. The idea of back propagation is based on the chain rule. The following is the procedure on performing the back propagation [2].

$$\nabla_{W^l} J = \frac{1}{m}\sum_{i=1}^{m}\nabla_{W^l} J^i$$

Where l is layer number, $\nabla_{W^l} J^i$ is gradient of the loss function for the ith sample. Also:

$$\nabla_{W^2} J^i = 2(\beta^2)^T H(i,:) \qquad \beta^2 = \hat{Y}_i - Y_i$$
$$\nabla_{W^1} J^i = (\beta^1)^T X_i \qquad \beta^1 = (\beta^2 W^2(2:end)) * \sigma'(Z_i)$$

Where σ' is the derivative of the softplus activation function:

$$\sigma'(z) = \frac{1}{1 + e^{-z}}$$

5. **Batch and Stochastic Gradient Descend**
After I figured out the backward propagation, I can perform gradient descend to minimize the loss function. The gradient descend basically has a loop that continues while the convergence criteria is not reached. Inside the loop, I perform a forward pass, then backward pass to calculate the gradients $\nabla_{W^2} J$ and $\nabla_{W^1} J$. The third step is then update the weight according

to the following formula:

$$W^l = W^l - \eta \nabla_{W^l} J$$

Where η is the learning rate, the learning rate follows the follow trend:

$$\eta = \frac{1}{k}$$

Where k is the number of iterations.
In this problem, I used batch gradient descend. The difference between batch gradient descend and stochastic gradient descend is that, batch gradient descend uses all samples to calculate the gradient at the back propagation stage, where stochastic gradient descend randomly chooses 1 sample and calculate the gradient base on this single sample. The stochastic gradient descend will result in a zig-zag curve that slowly minimize the loss function.

6. **Cross validation and model selection**

To select best number of perceptrons for the hidden layer, I used 10-fold cross validation to find the average validation-mean-square-error. The neural network model that has the minimum average validation-mean-square-error would be the best number of perceptrons to use. The 10-fold cross validation works by dividing the data into 10 partitions. In each experiment, 1 of the partition is left-out for validation, and the rest of the partitions are used for neural network training. After 10 experiments, the number of training-validation set combination has exhausted. We then average the validation-mean-square-error of each experiment to get the score for the neural network model. The following is an illustration of simpler 5-fold cross validation:

5-fold CV			Dataset		
Estimation 1	Test	Train	Train	Train	Train
Estimation 2	Train	Test	Train	Train	Train
Estimation 3	Train	Train	Test	Train	Train
Estimation 4	Train	Train	Train	Test	Train
Estimation 5	Train	Train	Train	Train	Test

Figure 2: 5-fold cross validation

7. **Result**

From the following figure, you can see that the best number of perceptrons to use in the hidden layer is 8 as it achieved the minimum validation-mean-square-error.

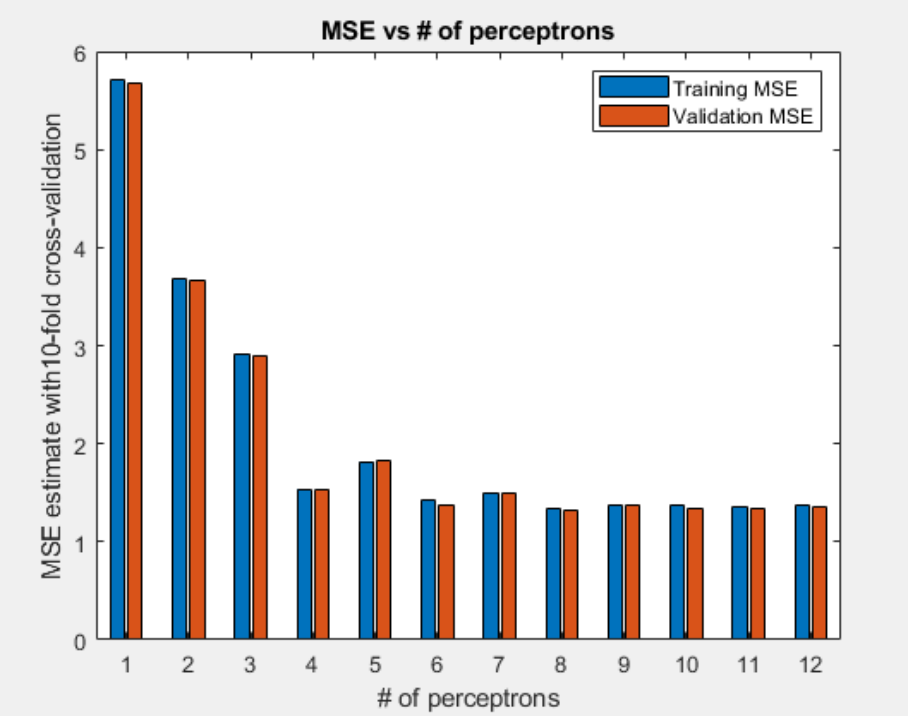

Figure 3: MSE vs Number of perceptrons

From the figure above, I found that number of perceptrons fewer than 8 underfits the model and forms a simpler function/classifier that may not best predict the result. Thus it has higher MSE. As the number of perceptrons grow, around at 8, the accuracy starts to saturate. Normally, as number of perceptrons grow, the model would overfit and thus do poorly on the test data, but in my result, it did not show that. I think this is the same phenomenon I observed in the problem 1 of the previous exam, where there aren't statistically significant difference between the parameters, so the accuracy wouldn't decrease [3].

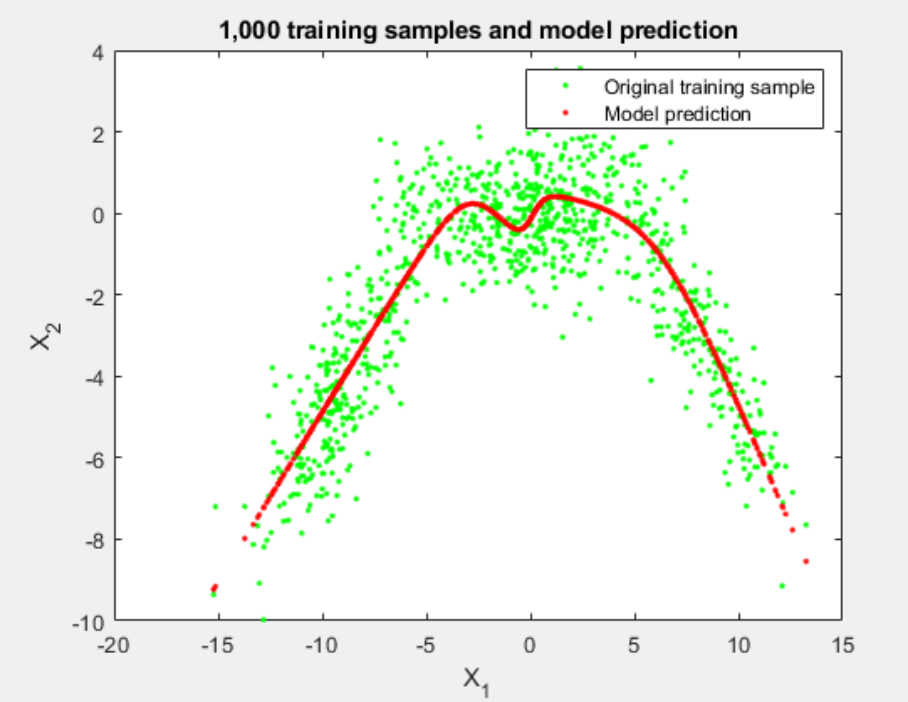

Figure 4: Model prediction on 1,000 training set

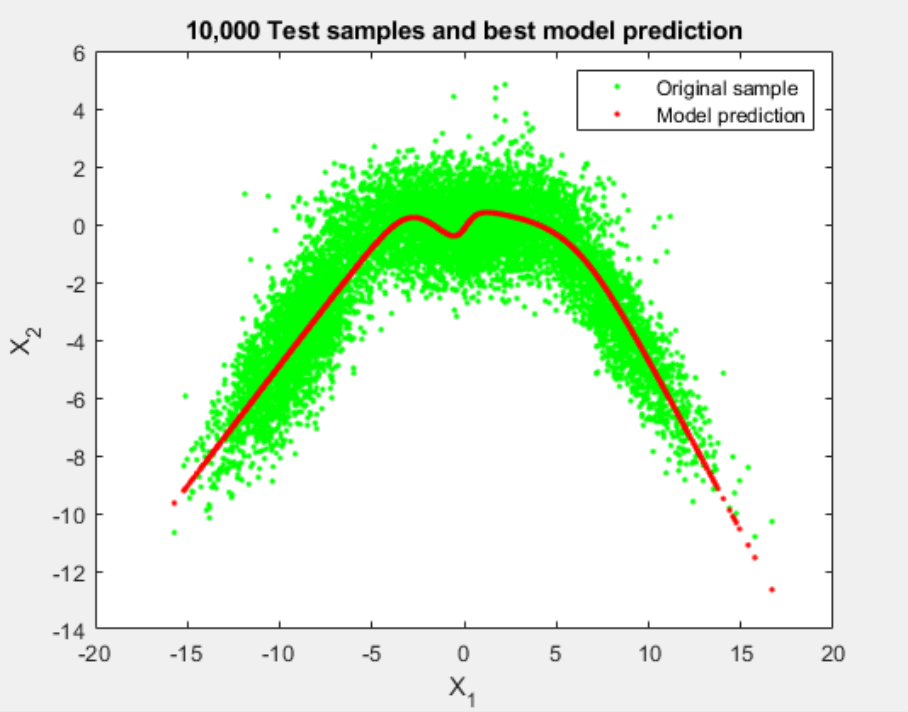

Figure 5: Model prediction on 10,000 testing set

The estimated mean squared error on training data is 1.304. The estimated mean squared error on testing data is 1.297.

2 Question 2

Support vector machine work by maximizing margin between classes using different type of boundaries. This boundary is determined by the kernel used in the SVM classification rule. Some common kernels are linear kernel, polynomial kernel, Gaussian radial basis function kernel, and hyperbolic tangent kernel [4]. In this problem, I am classifying nonlinear separable data, so I am using Gaussian radial basis function kernel.

1. **Gaussian RBF kernel**

The Gaussian RBF kernel is defined by following formula [4]:

$$k(\vec{x_i}, \vec{x_j}) = \varphi(\vec{x_i}) \cdot \varphi(\vec{x_j}) = e^{-\frac{(\vec{x_i}-\vec{x_j})^T(\vec{x_i}-\vec{x_j})}{2\sigma^2}}$$

Where x_i and x_j is sample i and j, σ is a constant hyper-parameter that controls the shape of the classification boundary, and is determined by using cross validation.

2. **Support vector machine optimization**

The SVM optimization is done by maximizing the following formula. This formula is derived using Wolfe dual. The detailed derivation can be found on the video on the Youtube playlist, Aug 09 [4].

$$Max(\sum_{i=1}^{n} c_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i c_i k(\vec{x_i}, \vec{x_j}) y_j c_j)$$

Where y_i is label for sample i, y_j is label for sample j, and n is number of samples.

This maximization problem is a dual maximization problem, and it can be solve by quadratic programming algorithms [4]. After it's solved, we can obtain the value of c_i.

We can also add a slack variable s to the optimization problem's constrain for overlapping data cluster. This slack variable has a constant called box constrain C that determines how much slack is allowed. This constant is determined using cross validation.

3. **SVM with Gaussian RBF kernel classification rule**

The classification vector is the following:

$$\vec{w} = \sum_{i=1}^{n} c_i y_i \varepsilon(\vec{x_i})$$

To use this vector to make classification, simply plug in the sample and compare the result with 0:

$$w \lesseqgtr 0| < 0 : D = 0; > 0 : D = 1$$

4. **Cross validation**

To select the best hyper-parameters C and σ, I used 10 fold cross validation to find the minimum validation-probability-of-error. The hyper-parameter combination that has minimum average validation probability of error is the best combination. The 10-fold cross validation works by dividing the data into 10 partitions. In each experiment, 1 of the partition is left-out for validation, and the rest of the partitions are used for SVM training. After 10 experiments, the number of training-validation set combination has exhausted. We then average the validation-probability-of-error of each experiment to get the score for the SVM model.

5. **Result**
The best C parameter is 100,000. The best σ parameter is 8.254. After using these hyper-parameters and the 1,000 training data to train the SVM, I was able to achieve 0 percent probability of error on the 10,000 testing data. This may be because the testing data are well separated, or the SVM may be over-fitting with this parameter.

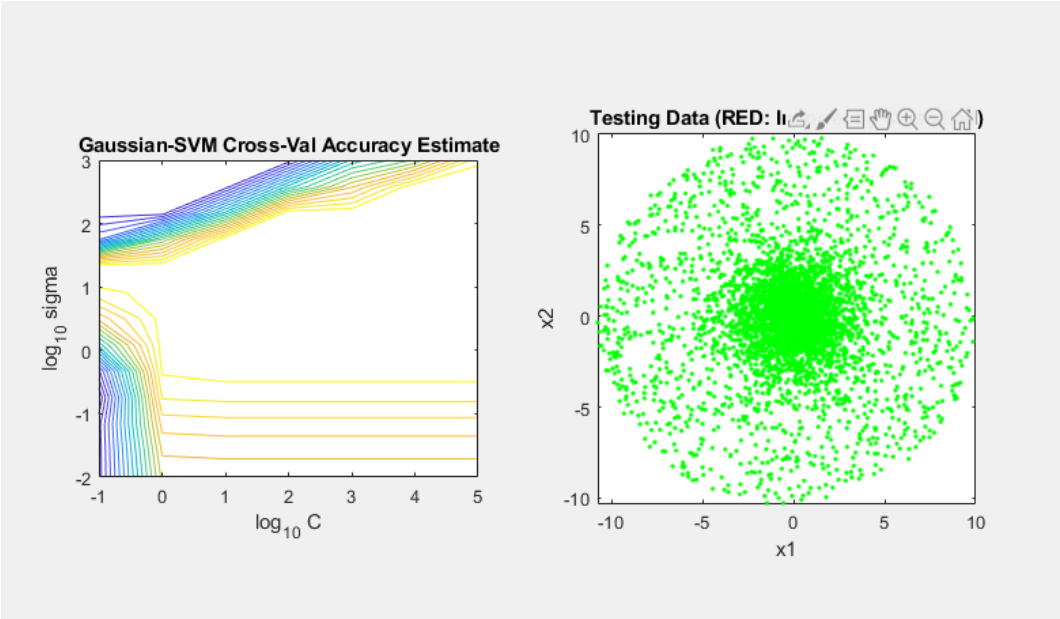

Figure 6: SVM best parameter test accuracy

3 Question 3

1. **Data Prepossessing**
To segment the images using GMM-based clustering, we treat each pixels as a data sample. Because it's a colored image, each data sample/pixel would contain 5 pieces of information. They are row index of the pixel, column index, red value, green value, and blue value. Once the image data was read into a 5 by number of pixels array, the values in the array is normalize into interval of [0,1].
2. **GMM fitting**
To fit processed data into GMM model with n components, it's similar to what we did in exam 2 question 3, where we used EM algorithm to maximize

the log likelihood. The process of GMM fitting is as following. First, assuming each components are uniformly selected (same priors), it randomly pick n samples as the mean/centroid of each clusters/components. Second, it assigned the rest of samples to the nearby centroid, then it calculate the variance/Sigma of each component based on these clusters.

```
alpha = ones(1,M)/M; % assume gaussian components are uniformly selected
Converged = 0; % Not converged at the beginning
while ~Converged %if not converged after max ite, reinitialize
    shuffledIndices = randperm(size(Dtrain,2)); % randperm?, N is number of samples drawn from
    mu = Dtrain(:,shuffledIndices(1:M)); % pick M random samples as initial mean estimates
    [~,assignedCentroidLabels] = min(pdist2(mu',Dtrain'),[],1); % assign each sample to the ne
    for m = 1:M % use sample covariances of initial assignments as initial covariance estimate
        Sigma(:,:,m) = cov(Dtrain(:,find(assignedCentroidLabels==m))') + regWeight*eye(d,d);
    end
```

Figure 7: Initializing the GMM model

Thirdly, it evaluates each data sample in each Gaussian cluster using multivariate Gaussian conditional PDF formula to see how likely each data sample belong to the cluster. This way, the new prior value, mean, sigma value of each Gaussian cluster can be found.

```
for l = 1:M
    temp(l,:) = repmat(alpha(l),1,size(Dtrain,2)).*evalGaussian(Dtrain,mu(:,l),Sigma(:,:,l)); %prior * pdf
end
plgivenx = temp./sum(temp,1); % sum of each column
alphaNew = mean(plgivenx,2); % mean of each row, new priors
w = plgivenx./repmat(sum(plgivenx,2),1,size(Dtrain,2));
muNew = Dtrain*w';
for l = 1:M
    v = Dtrain-repmat(muNew(:,l),1,size(Dtrain,2));
    u = repmat(w(l,:),d,1).*v;
    SigmaNew(:,:,l) = u*v' + regWeight*eye(d,d); % adding a small regularization term
end
```

Figure 8: Find new Gaussian cluster

Lastly, after certain iterations, once the change of prior, mean, sigma value are below the threshold, the GMM model has converged.

```
Dalpha = sum(abs(alphaNew-alpha'));
Dmu = sum(sum(abs(muNew-mu)));
DSigma = sum(sum(abs(abs(SigmaNew-Sigma))));
Converged = ((Dalpha+Dmu+DSigma)<delta); % Check if converged
```

Figure 9: Convergence criterion

Another way of fitting GMM model is to use fitgmdist() package in Matlab.

3. **Sample labeling**

To assign a cluster label to each samples once the GMM is fitted. I used the following MAP-classification rule. The component with the largest posterior probability is the cluster label [5].

$$l_i = argmax_k(\alpha_k g(x_i|\mu_k, \Sigma_k))$$

Where x_i is the i sample, k is the component number, l_i is the label of the i sample, and $g(x_i|\mu_k, \Sigma_k)$ is the multivariate Gaussian conditional PDF:

$$g(x|\mu, \Sigma) = (2\pi)^{\frac{-k}{2}} det(\Sigma)^{\frac{-1}{2}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

4. **Cross validation and Pixel shuffle**

To select best number of clusters in the image, I used 10-fold cross validation to find the average validation-log-likelihood. The GMM model that has the max average validation-log-likelihood would be the best number of clusters that best segments the image. The 10-fold cross validation works by dividing the data into 10 partitions. In each experiment, 1 of the partition is left-out for validation, and the rest of the partitions are used for GMM fitting. After 10 experiments, the number of training-validation set combination has exhausted. We then average the validation-log-likelihood of each experiment to get the score for this GMM model. The following is formula for calculating the validation-log-likelihood:

$$\sum_{i}^{N} ln(\sum_{i}^{N} \alpha_i * g(x|\mu_i, \Sigma_i))$$

Where x is validation data samples, α is prior, $g(x|\mu_i, \Sigma_i)$ is Gaussian PDF, and N is number of Gaussian components. This formula came from Professor's EMforGMM code [6].

In this problem, because I am reading each pixel row by row, one by one,

the data isn't drawn from random. If I just partition the data into 10 folds, data within each folds would come from same region of the image. So before I partition the data, I generated shuffle indexes for each pixels, and partition the shuffle indexes into 10 folds instead. This allows me to have data from different region of the image across different partitions.

5. **Result**

The following is the GMM based segmentation of the image with K=2 components:

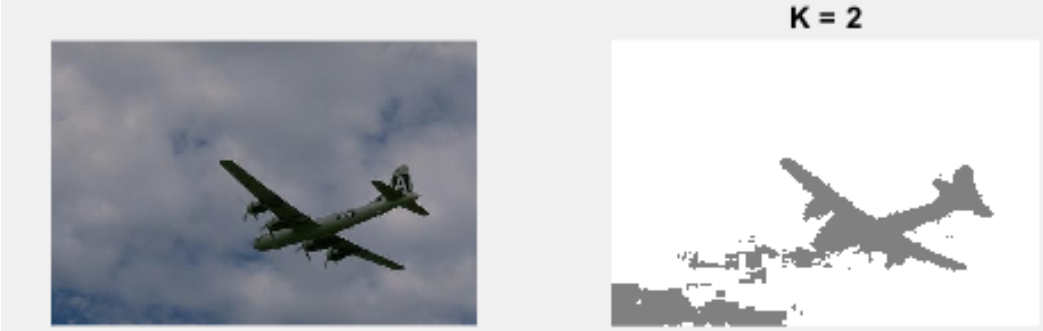

Figure 10: Airplane K=2

Figure 11: Bird K=2

For the airplane image, the best number of components is 8, as the

validation log likelihood peaked at 8. You can see that the GMM successfully segment the dark lower deck of the airplane, each white cloud patch, and the blue sky.

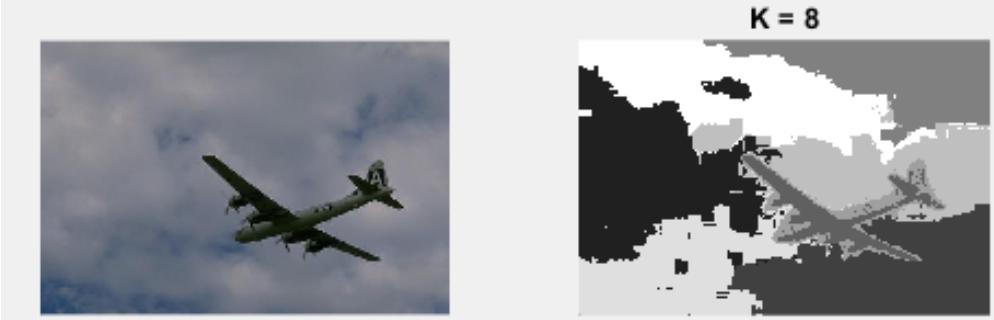

Figure 12: Airplane best K=8

After 8, the validation log likelihood starts to saturate, and from the image it generated, you can see the GMM cluster is starting to over-fit.

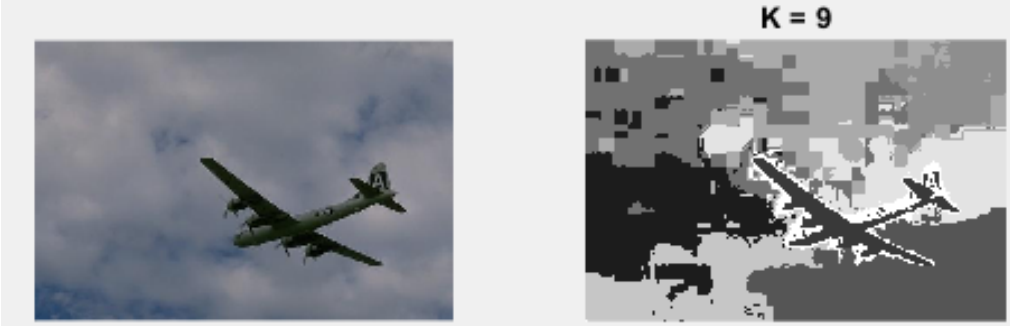

Figure 13: Airplane K=9

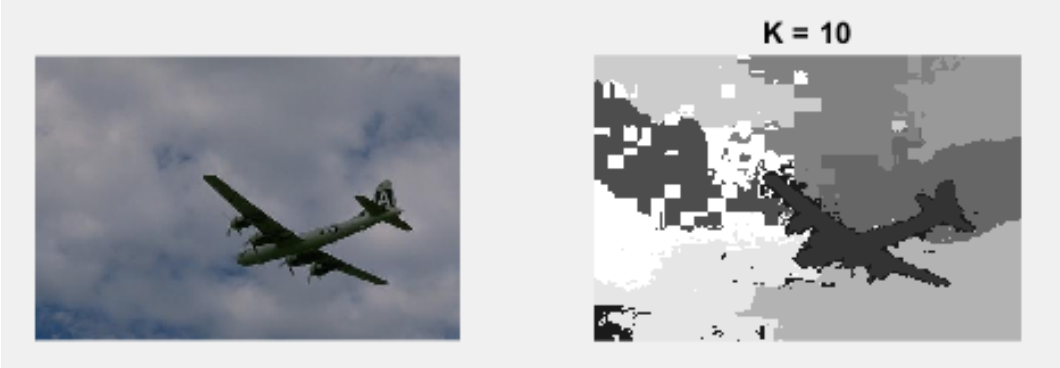

Figure 14: Airplane K=10

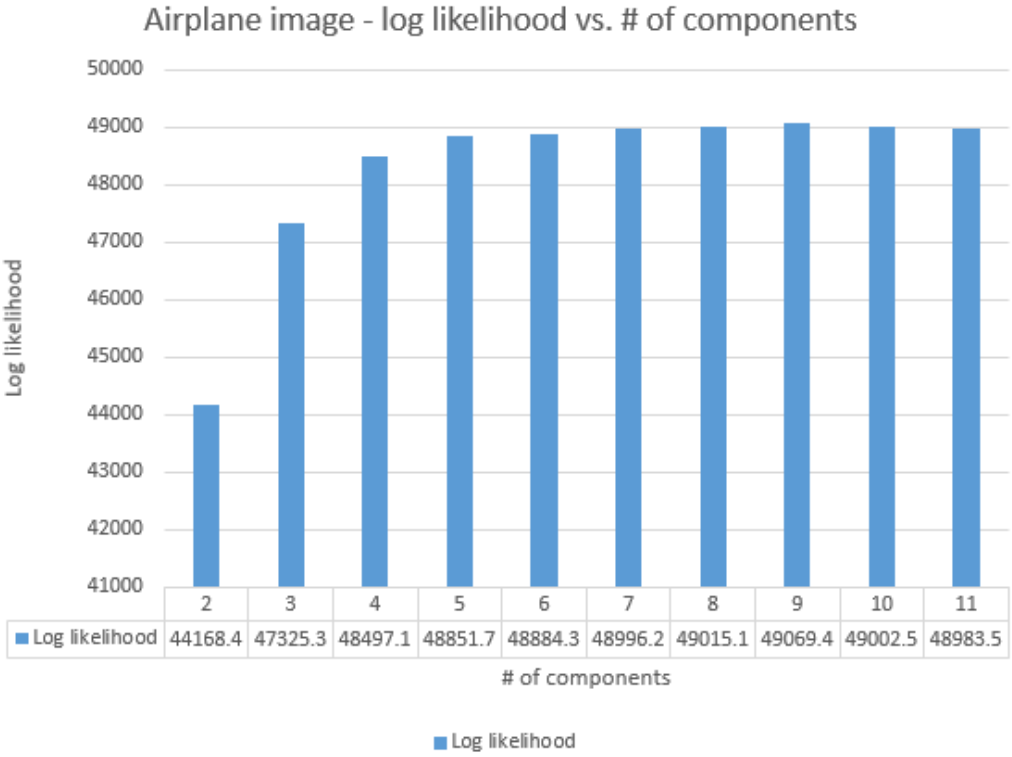

Figure 15: Airplane log likelihood

For the bird image, the best number of components is 12, as the validation log likelihood peaked at 12. You can see the GMM successfully segment

the dark corners, shades on the branches, and the shades on the bird.

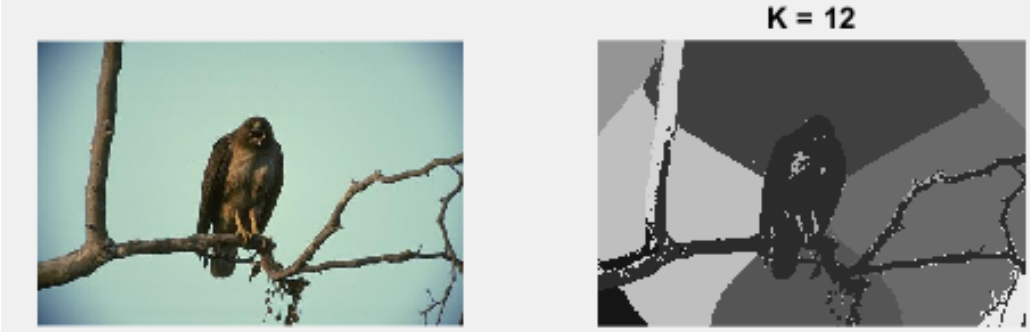

Figure 16: Bird best K=12

After 12, the validation log likelihood starts to saturate, and from the image it generated, you can see the GMM cluster is starting to over-fit.

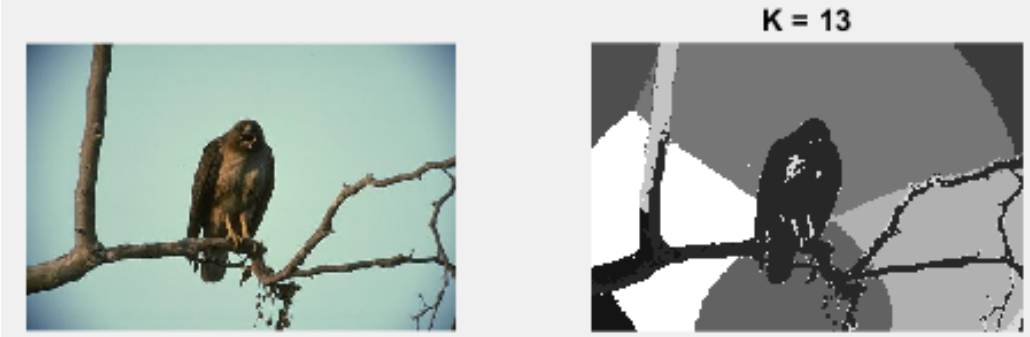

Figure 17: Bird K=13

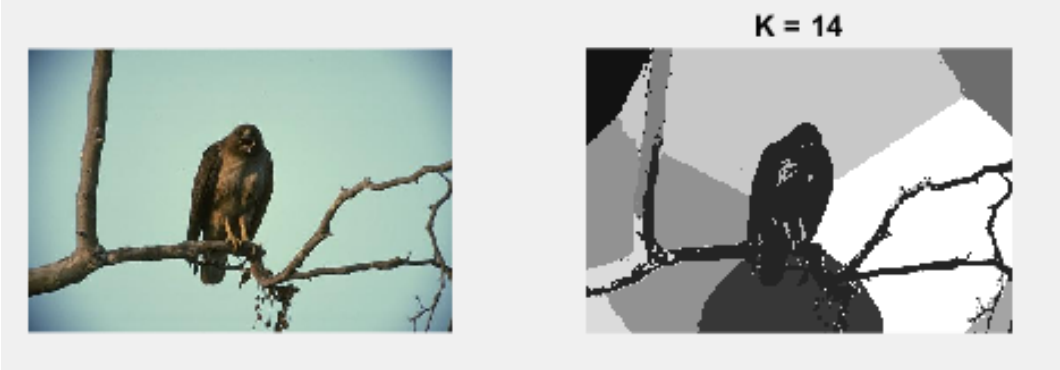

Figure 18: Bird K=14

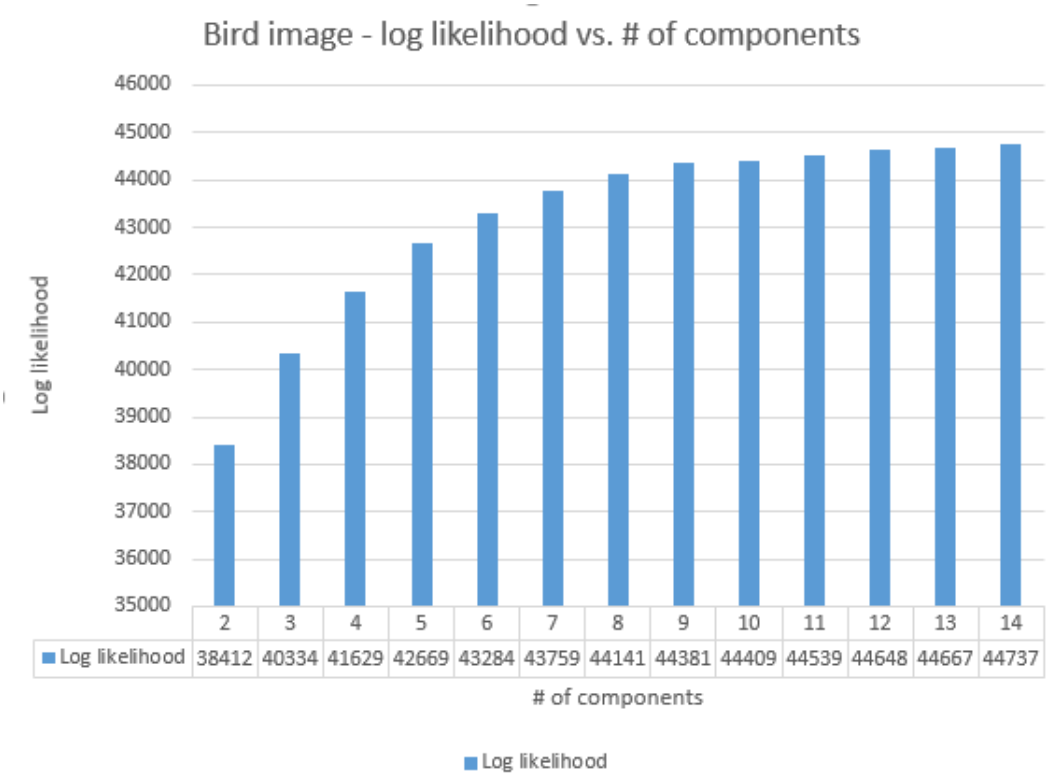

Figure 19: Bird log likelihood

4 References, Citations, and Acknowledgment

[1] C.-F. Wang, “Finding the Cost Function of Neural Networks,” Medium,25-Oct-2018. [Online]. Available: https://towardsdatascience.com/step-by-step-the-math-behind-neural-networks-490dc1f3cfd9. [Accessed: 09-Apr-2020].

[2] C.-F. Wang, “Calculating Gradient Descent Manually,” Medium, 25-Oct-2018. [Online]. Available: https://towardsdatascience.com/calculating-gradient-descent-manually-6d9bee09aa0b. [Accessed: 09-Apr-2020].

[3] ”Discrepancy is Q1 results”, Northeastern University, Mar 23, 2020. [Accessed: 09-Apr-2020]. [Online]. Available: Husky email.

[4] Youtube playlist video, EECE5644 - 2018 Summer2 - Thu, Aug 09. [Online]. Available: https://www.youtube.com/watch ?v=hkpu7S4kmB8list=PLIhapq-9C0GMZD8wFTwiCEfO8VLS8vNs4index=23. [Accessed: 09-Apr-2020]. Northeastern University, Boston, MA, 2020.

[5] Youtube playlist video, EECE5644 - 2018 Summer2 - Thu, Aug 07. [Online]. Available: https://www.youtube.com/watch ?v=CcvvCKzw7Qklist=PLIhapq-9C0GMZD8wFTwiCEfO8VLS8vNs4index=21. [Accessed: 09-Apr-2020]. Northeastern University, Boston, MA, 2020.

[6] EECE56442020sprSharedFolder/Code/EMforGMM, Northeastern University, Feb 24, 2020. [Accessed: 09-Apr-2020]. [Online]. Available: https://drive .google.com/drive/folders/1BJidoDDb6ifRs48MjPyGAj1RD3VxXvJi?usp=sharing

5 Git repo

https://github.com/MicroRAsus/Machine-Learning-Pattern-Recognition-HW4