CSE 586A Problem Set 4

Name: Jin Yang
UID: 467527

1. Auto-encoder

1.1 Problem Description

The goal of this problem is to train an auto-encoder network with aligned faces. After training this model, I can apply this model to reconstruct the training data with different sizes of latent layers and aadd regularity with different weights.

(a) Apply trained auto-encoder to recover the original aligned faces with a full size of hidden layers without regularity. Plot the recovered results.

(b) Apply trained auto-encoder to recover the original aligned faces with 1%, 3% and 10% of hidden layers respectively without regularity. Plot the recovered results and record the reconstruction error.

(c) Apply trained auto-encoder to recover the original aligned faces with a full size of hidden layers with regularity with different weights $\omega = \{0.1,0.2,0.3,0.4,0.5\}$. Plot the results and record the reconstruction error.

1.2 Solution

An auto-encoder us a neural network which can be used as a tool to learn deep neural networks. Training an auto-encoder is unsupervised because no labeled data is used to train the auto-encoder. The training process is based on the optimization of a cost function, which measures the error between the input x and its reconstruction at the output $\hat{x}$.

When training an auto-encoder, it is likely to make the sparsity regularizer small by increasing the values of the weights $w^{(l)}$ and decreasing the values of $z^{(l)}$. The method to prevent it from happening is to add a regularization term, L2 regularization term, on the weights to the cost function.

$$L_2 = \frac{1}{2}\sum_{l}^{L}\sum_{j}^{n}\sum_{i}^{k}(\omega_{ji}^{l})^2$$

Where L is the number of hidden layers, n is the number of observations or examples, and k is the number of variables in the training data.

The cost function for training a auto-encoder is an adjusted mean squared error, and the method to compute the mean squared error is shown below.

$$E = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}(x_{kn} - \hat{x}_{kn})^2$$

The method to calculate an adjusted mean square error is shown below.

$$E = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}(x_{kn} - \hat{x}_{kn})^2 + \lambda * \Omega_{weights} + \beta * \Omega_{sparsity}$$

Where $\lambda * \Omega_{weights}$ is the L2 regularization term, and λ is the coefficient for the L2 regularization term. $\beta * \Omega_{sparsity}$ is the sparsity regularization term, and β is the coefficient for the sparsity regularization term.

1.3 Experimental Details

The number of hidden layers that I set is 136, which means that the full size of hidden layer is 136, since the dimension of input training data is 136×1; based on the value of the full size of hidden layers, I can get the value of 1%, 3% and 10% of hidden layers, and then recover the faces through these auto-encoders.

The regularity that I choose is L2 Regularity, and when I train auto-encoder, I set the parameter, “regularity”, in auto-encoder function to “L2WeightRegularization’. I also set the weight of regularity to 0.1, 0.2, 0.3, 0.4, 0.5 respectively to reconstruct faces and calculate the reconstruction errors.

When I reconstruct and recover faces, it is necessary to calculate the reconstruction error. To achieve this goal, I use function “mse” to compute the mean squared error, which can be regraded as the value of reconstruction error.

1.4 Experimental Results

The original faces are shown in Figure 1 as below.

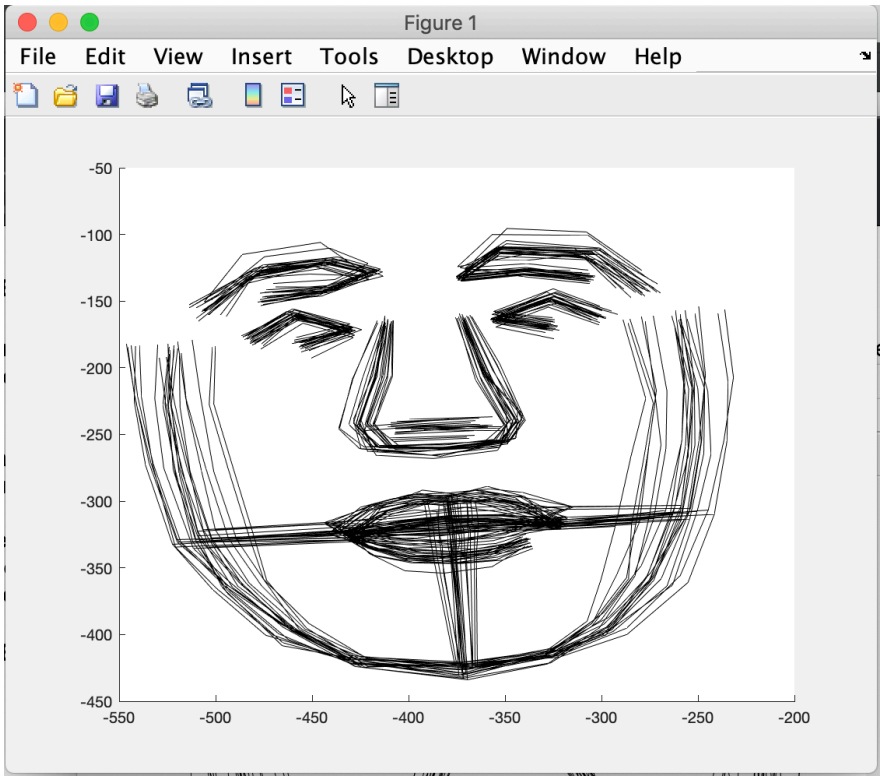

Figure 1. Original faces

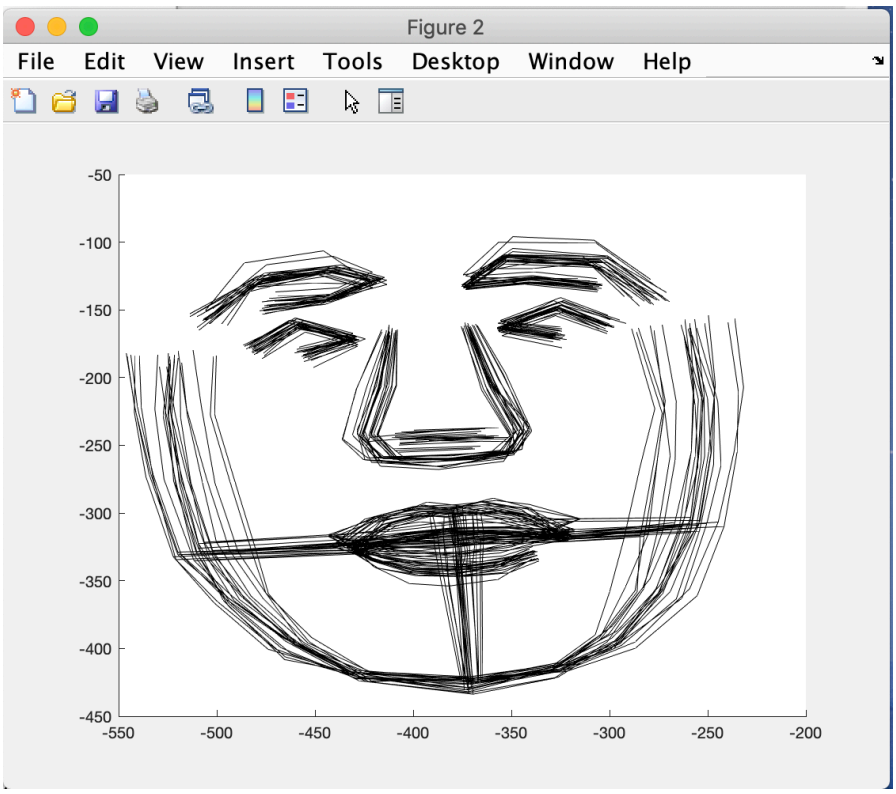

Figure 2. The reconstructed faces by AE with the full size of hidden layer

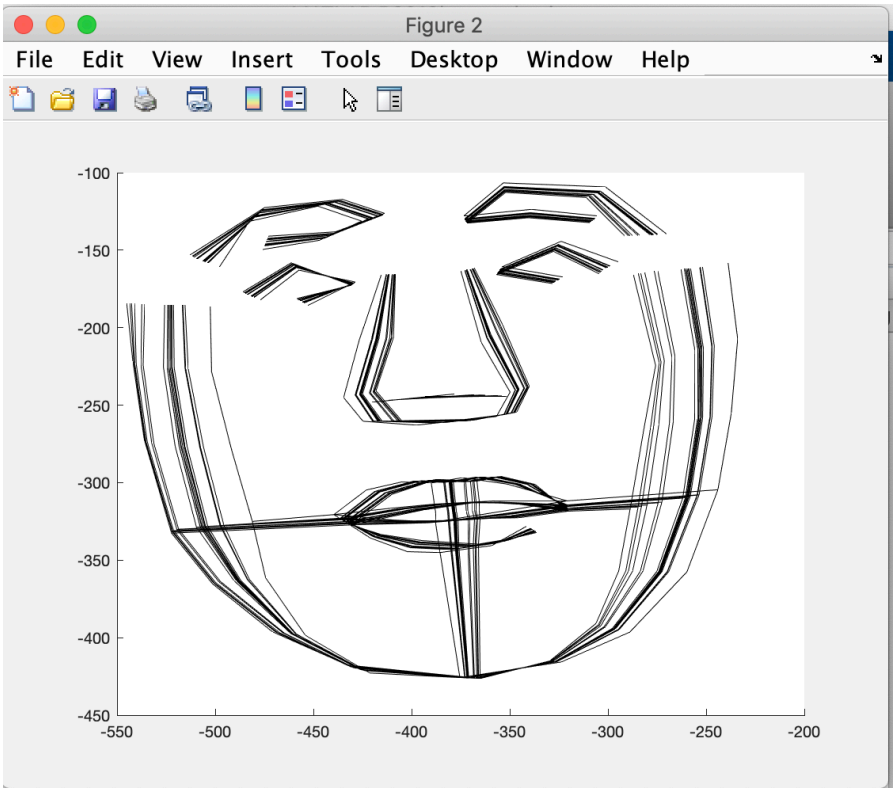

Figure 3. The reconstructed faces by AE with 1% of the full size of hidden layer

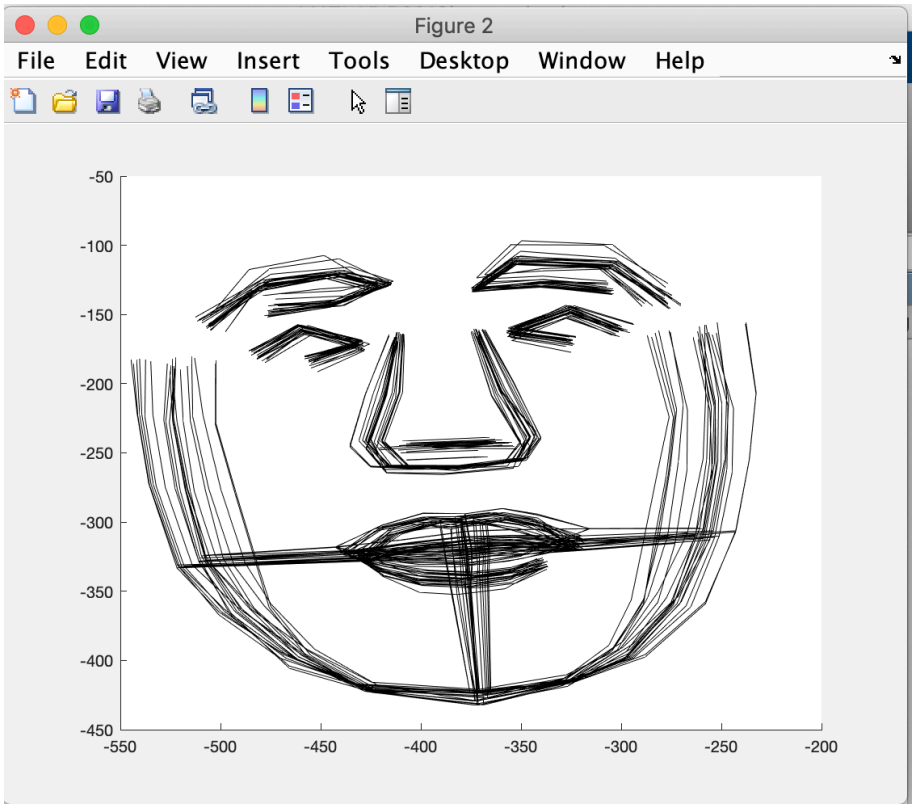

Figure 4. The reconstructed faces by AE with 3% of the full size of hidden layer

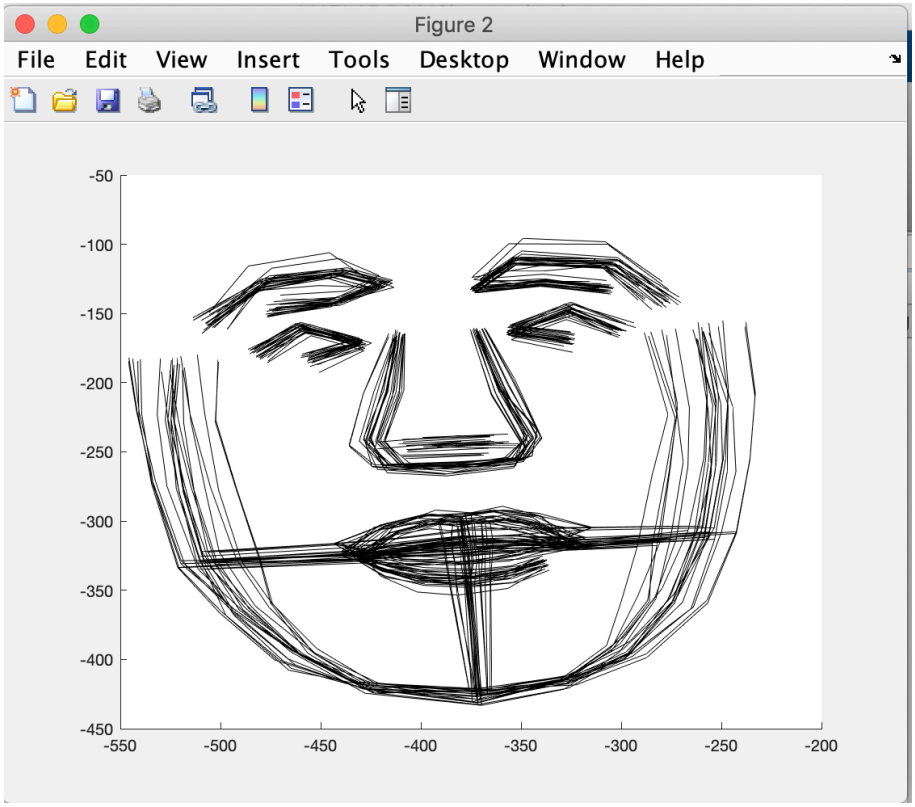

Figure 5. The reconstructed faces by AE with 10% of the full size of hidden layer

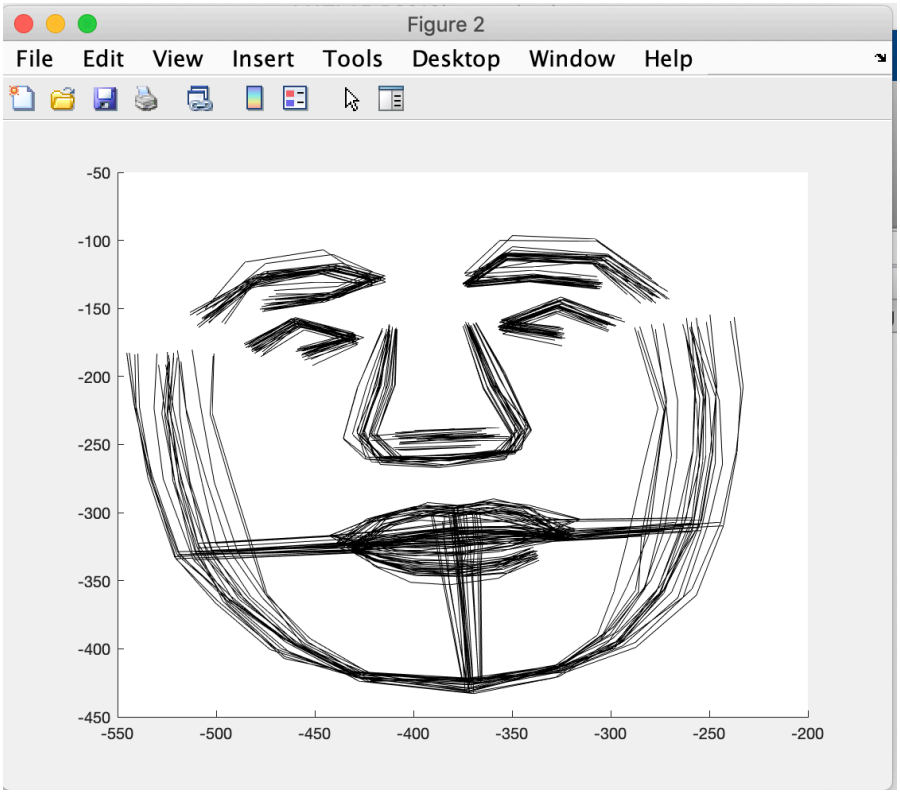

Figure 6. The reconstructed faces by AE with regularity with weights 0.1

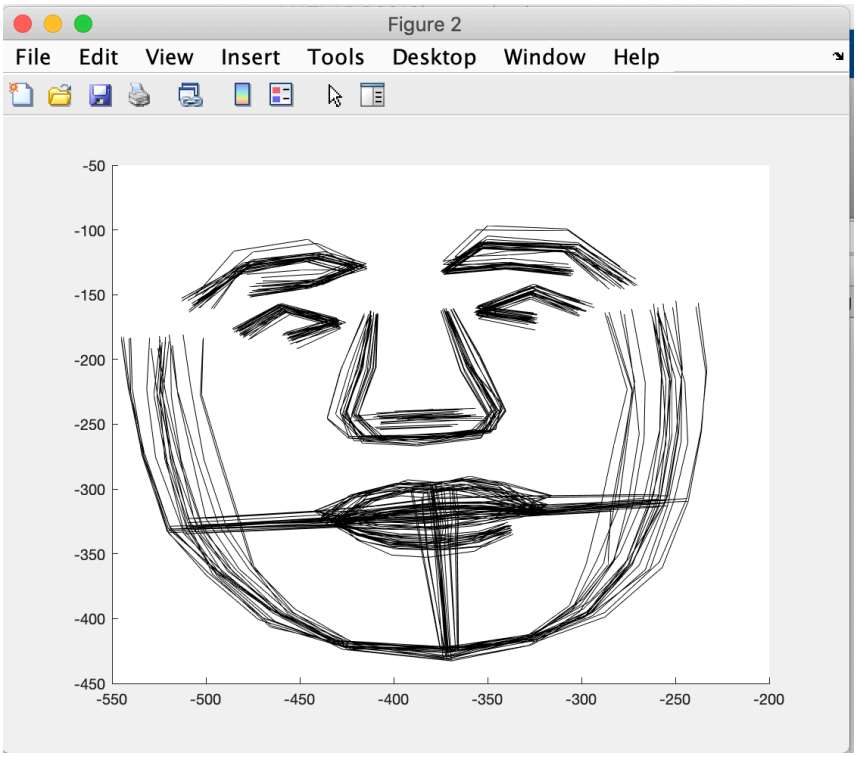

Figure 7. The reconstructed faces by AE with regularity with weights 0.2

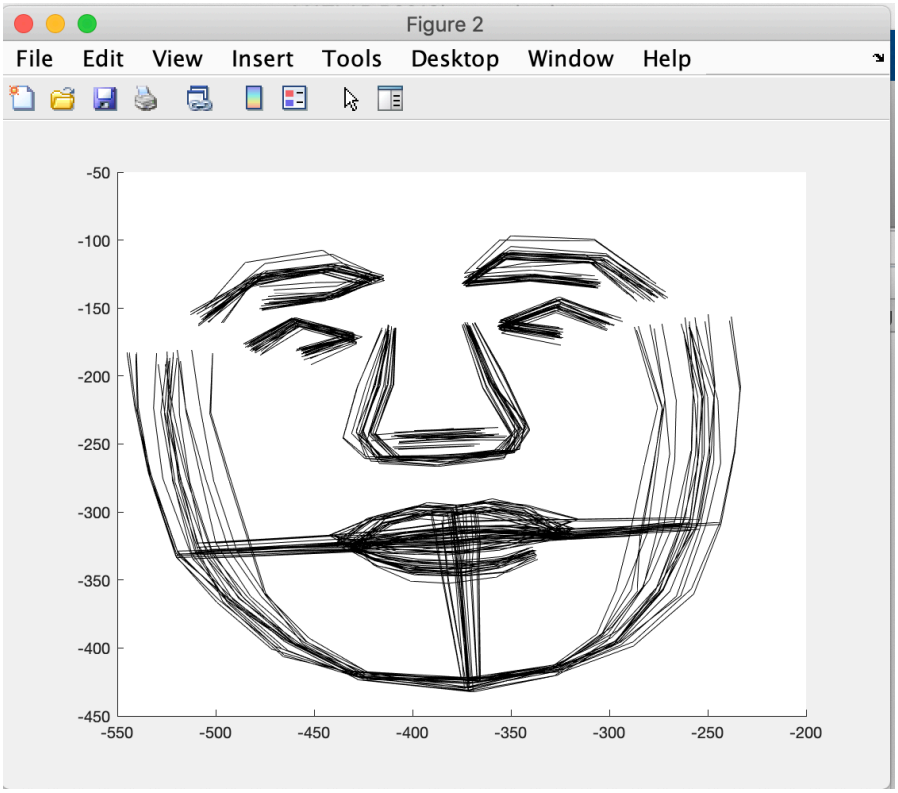

Figure 8. The reconstructed faces by AE with regularity with weights 0.3

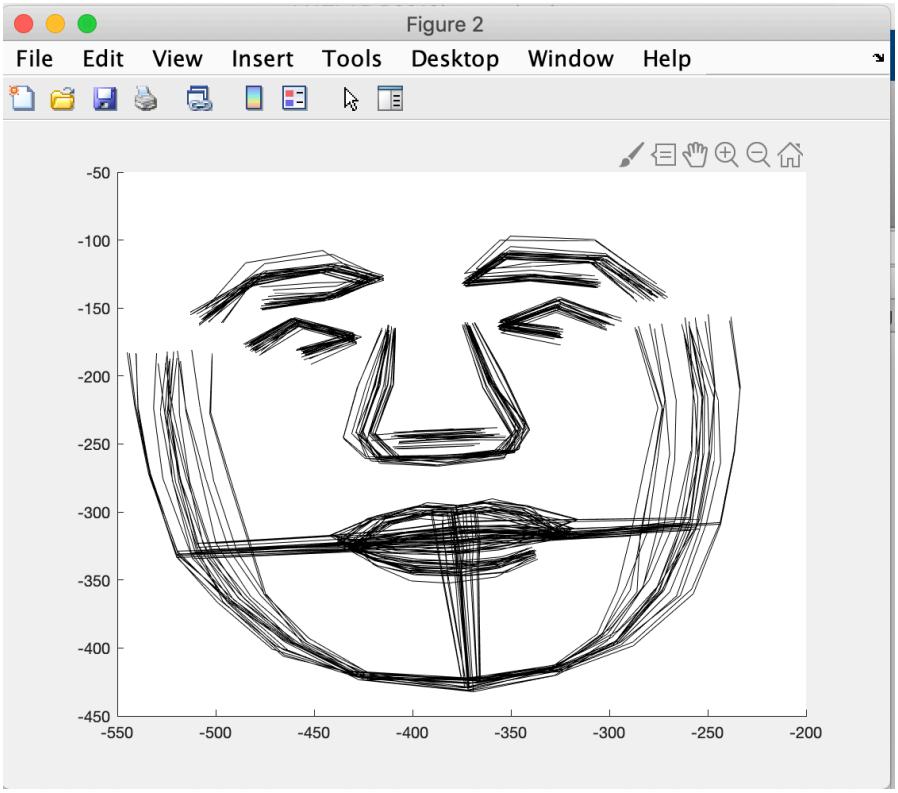

Figure 9. The reconstructed faces by AE with regularity with weights 0.4

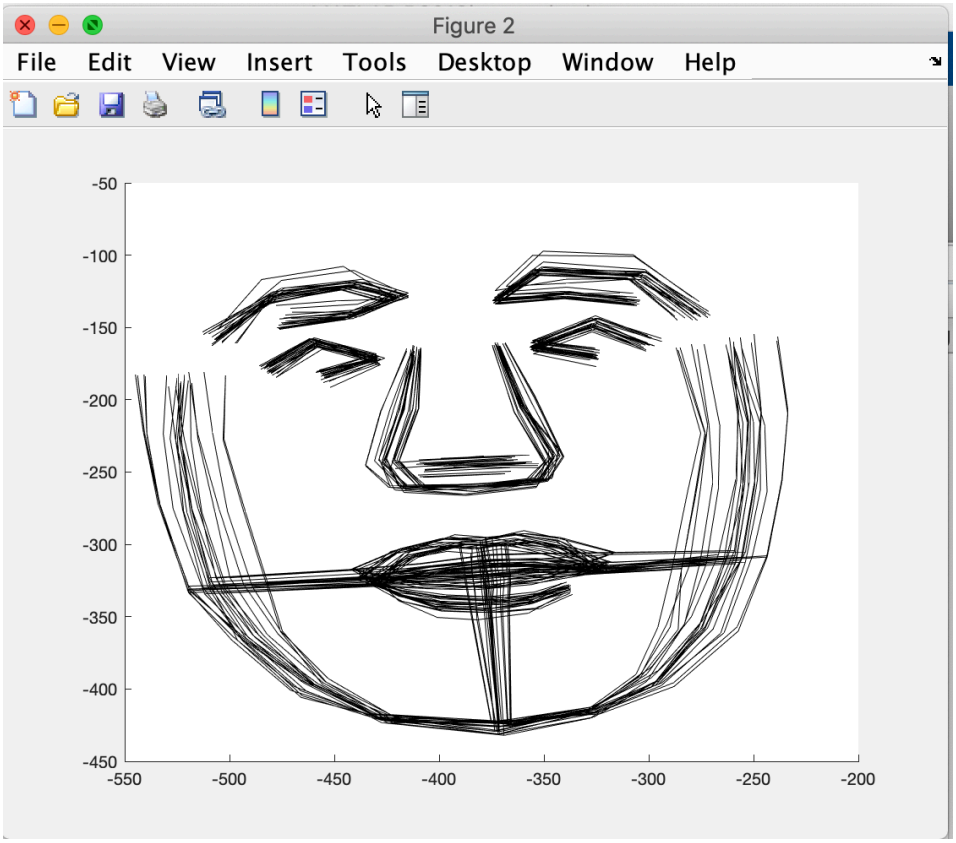

Figure 10. The reconstructed faces by AE with regularity with weights 0.5

The reconstruction is shown in Table 1.

Table 1. Mean Squared Error with Different AE

Auto-encoder	Mean Squared Error
The full size of hidden layers without regularity	0.0036
1% of the full size of hidden layers without regularity	8.8266
3% of the full size of hidden layers without regularity	2.2156
10% of the full size of hidden layers without regularity	0.2337
The full size of hidden layers with regularity with weights 0.1	0.1445
The full size of hidden layers with regularity with weights 0.2	0.2783
The full size of hidden layers with regularity with weights 0.3	0.3894
The full size of hidden layers with regularity with weights 0.4	0.5065
The full size of hidden layers with regularity with weights 0.5	0.6123

1.5 Discussion

The larger the hidden layer size is, the less reconstruction error I will get, which means that the difference between original data and reconstructed data will be smaller when I use more hidden layers to train the auto-encoder.

If I add a regularization term during the process of training the auto-encoder, the reconstruction error will be larger than that if I train the auto-encoder without adding any regularization term. This means that if I add a regularization term when I train an auto-encoder, the difference between original data and reconstructed data will be larger. Furthermore, with the increase of weights, the mean squared error will increase.

2. Regression

2.1 Problem Description

The goal of this problem is to train a classier that will predict an unknown class label from a new data point. The Bayesian logistic regression model is shown below.

$$Y \sim Ber(\frac{1}{1+e^{-X^T\beta}})$$

$$\beta \sim N(0,\sigma^2 I)$$

The solution is to drive and implement a maximum a posterior Bayesian inference on β based on this model.

2.2 Solution

(a) The formula for the unnormalized posterior of $\beta \mid Y$, i.e.

$$p(\beta \mid y; x, \sigma) \propto \prod_{i=1}^{n} p(y_i \mid \beta; x_i) p(\beta; \sigma)$$

The details of the formula for normalized posterior is shown below.

$$p(y_i \mid \beta; x_i) = (\frac{1}{1+e^{-x_i^T\beta}})^{y_i}(1 - \frac{1}{1+e^{-y_i^T\beta}})^{1-x_i}$$

Since the variable Y follows Bernoulli distribution $Y \sim Ber(\frac{1}{1+e^{-X^T\beta}})$

$$p(\beta; \sigma) = \frac{1}{\sqrt{2\pi\sigma^2 I}} e^{-\frac{\beta^2}{2\sigma^2}}$$

Since the variable β follows standard normal distribution $\beta \sim N(0,\sigma^2 I)$

$$p(\beta \mid y; x, \sigma) = \prod_{i=1}^{n} p(y_i \mid \beta; x_i) p(\beta; \sigma) = \prod_{i=1}^{n} (\frac{1}{1+e^{-x_i^T\beta}})^{y_i}(1 - \frac{1}{1+e^{-x_i^T\beta}})^{1-y_i} \frac{1}{\sqrt{2\pi\sigma^2 I}} e^{-\frac{\beta^2}{2\sigma^2}}$$

(b) Prove that this posterior is proportional to $exp(-U(\beta))$, where

$$U(\beta) = \sum_{i=1}^{n} (1-y_i) x_i^T \beta + log(1+e^{-x_i^T\beta}) + \frac{1}{2\sigma^2} \parallel \beta \parallel^2$$

From part (a), the posterior is shown below

$$p(\beta \mid y; x, \sigma) = \prod_{i=1}^{n} (\frac{1}{1+e^{-x_i^T\beta}})^{y_i}(1 - \frac{1}{1+e^{-x_i^T\beta}})^{1-y_i} \frac{1}{\sqrt{2\pi\sigma^2 I}} e^{-\frac{\beta^2}{2\sigma^2}}$$

$$= \prod_{i=1}^{n} \frac{1}{(1+e^{-x_i^T\beta})^{y_i}} \frac{(e^{-x_i^T\beta})^{1-y_i}}{(1+e^{-x_i^T\beta})^{1-y_i}} \frac{1}{\sqrt{2\pi\sigma^2 I}} e^{-\frac{\beta^2}{2\sigma^2}}$$ Fraction simplification

$$= \frac{1}{\sqrt{2\pi\sigma^2 I}} \prod_{i=1}^{n} \frac{1}{1+e^{-x_i^T\beta}} e^{(-x_i^T\beta)(1-y_i)-\frac{\beta^2}{2\sigma^2}}$$ Change the format

$$= \frac{1}{\sqrt{2\pi\sigma^2 I}} e^{\sum_{i=1}^{n} log(\frac{1}{1+e^{-x_i^T\beta}})} e^{\sum_{i=1}^{n}(-x_i^T\beta)(1-y_i)-\frac{\beta^2}{2\sigma^2}}$$ Property of logarithm

$$= \frac{1}{\sqrt{2\pi\sigma^2 I}} e^{\sum_{i=1}^{n} -log(1+e^{-x_i^T\beta})-(x_i^T\beta)(1-y_i)-\frac{\beta^2}{2\sigma^2}}$$ Property of exponent

$$= \frac{1}{\sqrt{2\pi\sigma^2 I}} e^{-\sum_{i=1}^{n} log(1+e^{-x_i^T\beta})+(x_i^T\beta)(1-y_i)+\frac{\beta^2}{2\sigma^2}}$$

Therefore, the posterior $p(\beta \mid y; x, \sigma)$ is proportional to $exp(-U(\beta))$

(c) Then I implement maximum a posterior MAP to infer β. The method to infer β is to compute the minimal value of β through gradient descent algorithm. I compute the gradient of β first from $U(\beta)$

$$U(\beta) = \sum_{i=1}^{n} (1-y_i)x_i^T\beta + log(1+e^{-x_i^T\beta}) + \frac{1}{2\sigma^2} \parallel \beta \parallel^2$$

$$\frac{\partial U(\beta)}{\partial x} = \sum_{i=1}^{n} (1-y_i)x_i^T + \frac{1}{1+e^{-x_i^T\beta}}(e^{-x_i^T\beta})(-x_i^T) + \frac{1}{\sigma^2} \parallel \beta \parallel$$ Calculate derivative

$$= \sum_{i=1}^{n} (1-y_i)x_i^T + \frac{(e^{-x_i^T\beta})(-x_i^T)(e^{x_i^T\beta})}{(1+e^{-x_i^T\beta})(e^{x_i^T\beta})} + \frac{1}{\sigma^2} \parallel \beta \parallel$$ Change format

$$= \sum_{i=1}^{n} (1-y_i)x_i^T - \frac{x_i^T}{1+e^{x_i^T\beta}} + \frac{1}{\sigma^2} \parallel \beta \parallel$$

2.3 Experimental Details

I read in data from the text file “iris.txt” file through implementing “Import Data” function. I generate a function to load in data as a table, named “importfile.m”. Since the data which “Species” is “setosa” will not be used in this experiment, I read data from 52nd rows to 151st row. I acquire total 100 rows data, including 50 rows for “versicolor” and 50 rows for “virginica”. Then I use the first 30 rows for each species as training data and leave out the last 20 rows for each species as testing data.

The data format of “Species” is string, so it is necessary to transform string data type to other data type, such as integer, to ensure the successful implementation of prediction. Therefore, I label the species “versicolor” as 1, and label the species “virginica” as 0.

After I read in data as a table from the text file, I read in data from the table as arrays. To be specific, I read in the first 30 rows for species, “versicolor”, into the array “x_train1” and the last 20 rows for this species into the array “x_test1”; similarly, I read in the first 30 rows for species, “virginica”, into the array “x_train2” and the last 20 rows for this species into the array “x_test2”. Therefore, the total number of training data is 60 and the total number of testing data is 40.

After reading in all data, I preprocess these data and prepare for the implementation of gradient descent. I add a constant term, a column of 1’s to my X matrix. Before implementing gradient descent algorithm to calculate the minimal value for β, I set various values for learning rate ϵ and variance σ^2.

After training Bayesian logistic regression with training data, I use estimated β and apply this model to predict labels for testing data. When I get predictions for all testing data, I notice that these results are positive and negative float numbers which lie in the range (-1,1). Since the initial labels that I set for two species, “versicolor” and “virginica”, is 1 and 0, it is optimal to transform these results and ensure that transformed results lie in the range [0,1]. The way to achieve this goal is to apply sigmoid function. Sigmoid function that I use is presented below.

$$f(x) = \frac{1}{1 + e^{-x}}$$

Because of the properties of sigmoid function, the value of accurate prediction for species, “versicolor” is larger than 0.5, so the false prediction for this species is smaller than 0.5. Similarly, the value of sure prediction for species, “ virginica”, is smaller than 0.5, so the false prediction for this species is larger than 0.5. I create a variable “total_error_class” to store the total number of false prediction and a variable “error_rate” to store the rate of false prediction. In this situation, if the value of prediction for species, “versicolor”, is smaller than 0.5, the model make a false prediction for species “versicolor”, and the value of “total_error_class” will increase by 1; if the value of prediction for species, “virginica”, is larger than 0.5, the model make a false prediction for species, “virginica”, and the value of “total_error_class” should also increase by 1.

I calculate the value of energy to check whether the energy function converges, and then I can acquire information about whether the value of β decrease with the process of gradient descent. In this experiment, the energy can be calculated as followed.

$$E = logp(\beta \mid y; x, \sigma) = \sum_{i=1}^{n} (1 - y_i) x_i^T \beta + log(1 + e^{-x_i^T \beta}) + \frac{1}{2\sigma^2} \parallel \beta \parallel^2$$

After calculating the energy in every iteration, I plot the energy to check whether energy function converges.

2.4 Experimental Results

I set the value of learning rate ϵ to values, 1, 0.1, 0.01, 0.001 respectively and estimate the average error rate; I also set the value of variance σ^2 to 1, 10, 100, 1000, and then estimate the average error rate. The results is recorded in Table 2 and Table 3 below.

Table 2. Estimated total error number

	1	10	100	1000
1	20	20	20	7
0.1	20	20	9	2
0.01	20	12	2	1
0.001	20	3	2	2

Table 3. Estimated error rate

	1	10	100	1000
1	0.5000	0.5000	0.5000	0.1750
0.1	0.5000	0.5000	0.2250	0.0500
0.01	0.5000	0.3000	0.0500	0.0250
0.001	0.5000	0.0750	0.0500	0.0500

The energy function is plotted in Figure 11.

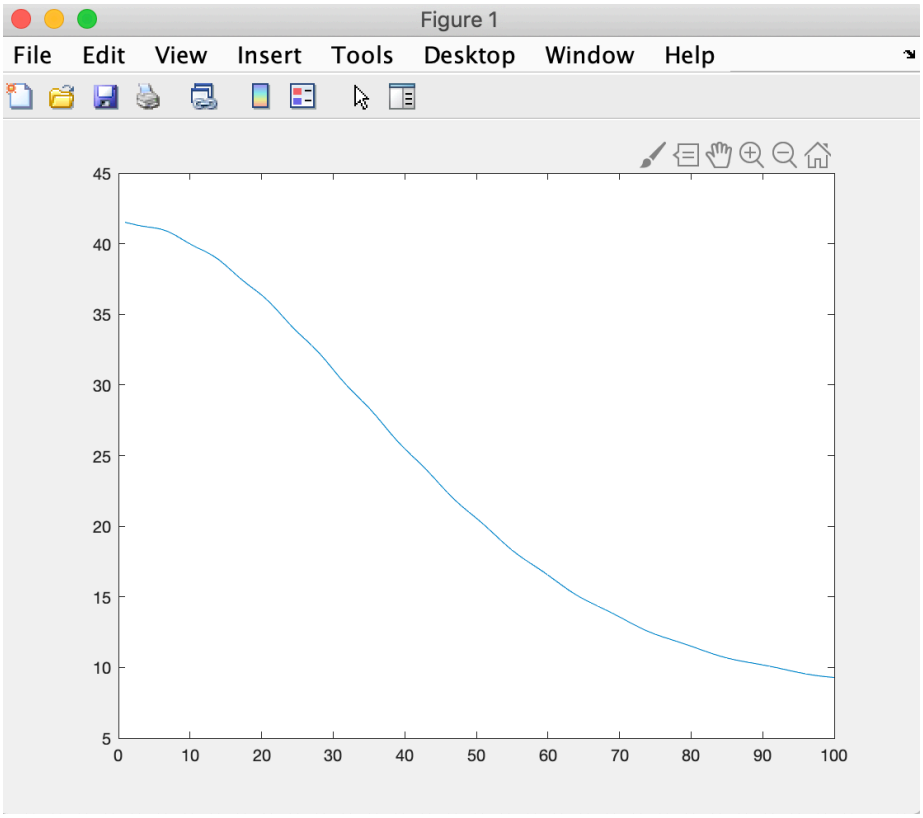

Figure 11. Energy Function

2.5 Discussion

With the increase of variance σ^2, the value of Estimated total error number and estimated error rate will decrease. If the value of learning rate ϵ is too large or too small, it is impossible for the model to get an optimal performance. The reason is that if the learning rate ϵ is too large, the gradient will bounce around the optimal value; therefore, it is unlikely to reach the optimal value through implementing gradient descent algorithm; on the contrary, if the learning rate ϵ is too small, the gradient cannot reach to the optimal value through a specific number of iterations (the number of iterations in this experiment is 100) when I implement gradient descent algorithm.

Therefore, the value of learning rate ϵ and variance σ^2 that I choose is 0.01 and 1000 respectively.