Math 6535 Homework 2

Yingxue Su, Chunmei Sun

March 2019

1 Describe the data set and associated classification task

The original data set consists of 5 different folders, each with 2300 files, with each file representing a single person's brain activity for one second. Each file contains 178 descriptors which are the values of the EEG recording at a different point in time. These descriptors show the brain activity in one second. And the original 5 classes are as following:

Class 1: recording of seizure activity.
Class 2: they record the EEG where the tumor was located.
Class 3: they identify where the region of the tumor was in the brain and recording the EEG activity from the healthy brain area.
Class 4: eyes closed, which means when they were recording the EEG signal of the brain the patient had their eyes closed.
Class 5: eyes open, which means when they were recording the EEG signal of the brain the patient had their eyes open.

Now we want to change the number of classes to 3. We keep class 1 as before and name it class 3. Then we combine class 2 and class 3 as class 1 since they are both related to brain activity towards tumor. And combine class 4 and class 5 as class 2 since they are related to brain activity towards eye open or closed. Now the class becomes as following:
Class 1: they record the EEG where the tumor was located or they identify where the region of the tumor was in the brain and record the EEG from the healthy brain area.
Class 2: eyes open or eyes closed, which means whey they were recording the EEG signal of the brain the patient had their eyes open or closed.
Class 3: recording of seizure activity.

Then the size of class 1 is $siz1 = 4600$, the size of class 2 is $siz2 = 4600$ and the size of class 3 is $siz3 = 2300$. We choose the size of the training set as 8050 and the size of the test set as 3450. And generate the training set and test set as following:

$data\_train, data\_test, y\_train, y\_test = train\_test\_split(scaled\_data, y, test_size = 0.3)$

The size of each class within the training set is $3194, 3219, 1637$. The size of each class within the test set is $1405, 1371, 664$. The proportions of each class within the training set is $0.3968, 0.3999, 0.2033$. The proportions of each class within the test set is $0.4071, 0.3974, 0.1925$. These two proportions are quite similar.

2 Select 2 tentative sizes h for the hidden layer

Apply PCA analysis to the data set. Figure 1 shows the decreasing sequence of eigenvalues of the correlation matrix of the data set. $h90 = 33$, which means the first 33 biggest eigenvalues preserves 90% of the total sum of the eigenvalues.

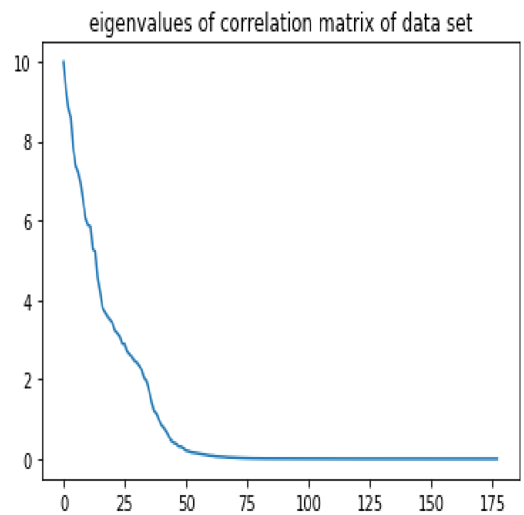


Figure 1: eigenvalues of the correlation matrix of the data set

Apply PCA analysis to the data set containing M_j cases which belongs to class j for $j = 1, 2, 3$, respectively. And compute the smallest number U_j of eigenvalues preserving 90% of the total sum of the M_j eigenvalues for $j = 1, 2, 3$. We get $U_1 = 17$, $U_2 = 31$, $U_3 = 34$. Then $hL = U_1 + U_2 + U_3 = 82$.

3 Implement automatic training

3.1 Automatic training with the size of hidden layer $h90 = 33$

For $h90 = 33$, we choose $RELU$ function as our response function. And we use tensorflow to implement MLP. Before we do the training, we first scale the data by using $scaler = preprocessing.StandardScaler()$.
For initialization of the weights, we use the function

$$tf.random\_normal()$$

in tensorflow to assign values to weights and biases. For the successive gradient descent step size $\epsilon(n)$, we use the function

$$learning\_rate = tf.train.exponential\_decay(initial\_learning\_rate = 0.01,$$

$$global\_step = global\_step, decay\_steps = training\_epochs, decay\_rate = 0.9)$$

where $training\_epochs = 5000$ and $global\_step$ is calculated by

$$global\_step = tf.Variable(0, trainable = False),$$

$$add\_global = global\_step.assign\_add(1).$$

And the following figure 2 shows how the leaning rate decreases during learning.

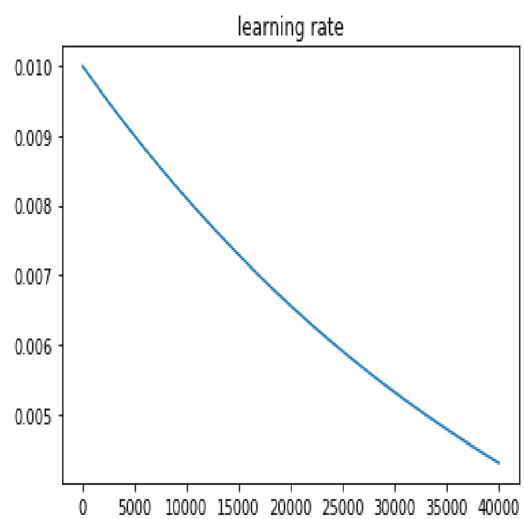


Figure 2: the decay of learning rate $\epsilon(n)$

We define the MLP with one hidden layer as:

$$layer\_1 = tf.nn.relu(tf.add(tf.matmul(x, weights['h']), biases['b']))$$

$$out\_layer = tf.add(tf.matmul(layer\_1, weights['out']), biases['out']).$$

We implement average cross entropy loss function and gradient descent optimizer as following:

$$loss1 = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(labels = Y1, logits = logits1))$$

$$optimizer1 = tf.train.GradientDescentOptimizer(learning\_rate = learning\_rate).minimize(loss1)$$

where $logits1$ is the output we get from the MLP and $Y1$ is the correct output. And command $tf.nn.softmax\_cross\_entropy\_with\_logits$ calculate the cross entropy between the correct distribution and the distribution from the softmax function of output from MLP.
And we define the accuracy as the percentage of correct classifications as following:

$$correct\_prediction = tf.equal(tf.argmax(logits1, 1), tf.argmax(Y1, 1))$$

$$accuracy1 = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))$$

Where $tf.argmax(logits1, 1)$ gives us the index of the largest value in vector $logits1$ and $tf.argmax(Y1, 1)$ gives us the index of the largest value in vector $Y1$. If these two index are same, $correct\_prediction = True$. If these two index are different, $correct\_prediction = False$. And $accuracy1$ calculate the mean of all $correct\_prediction$ for every case.

For batch learning, we choose batch size $B = 1000$. For every train, we generate 1000 random numbers between 0and 8050 as

$$randidx = np.random.randint(9660, size = batch\_size)$$

And these 1000 numbers are the indexes of rows we choose from the training data set for this train. So we use $batch\_xs$ and $batch\_ys$ chosen as following for this train:

$$batch\_xs = data\_train.iloc[randidx, :].$$

$$batch\_ys = y\_train.iloc[randidx, :]$$

We run 5000 total epochs and record batch average cross entropy $BACRE(n)$, weight $W(n)$ and gradient $G(n)$, which takes approximately 13 minutes. Then we plot $BACRE$, $||W(n+1) - W(n)||/||W(n)||$ and $||G_n||/d$ in figure 3,4,5. From the figure 3,4,5 we can see that $BACRE(n)$, $||W_{n+1} - W_n||/||W_n||$ and $||G_n||/d$ all decrease very fast with certain amount fluctuations and stabilize around 0.4998, 2.14×10^{-5} and 0.0049 respectively. $||G(n)||/d$ shows the average absolute value of the coordinates of vector $G(n)$. So the gradient reduce pretty well towards 0. Also we calculate the average accuracy on every batch and plot it in figure 6. We can see the accuracy increase with oscillation until around 0.80.

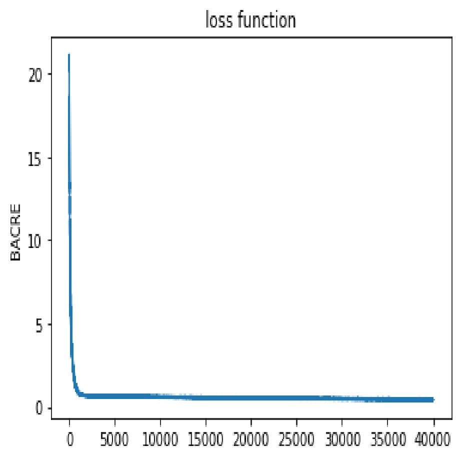


Figure 3: BACRE with $h90 = 33$

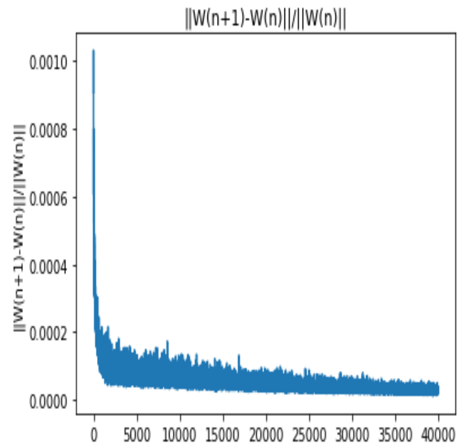


Figure 4: $||W_{n+1} - W_n||/||W_n||$ with $h90 = 33$

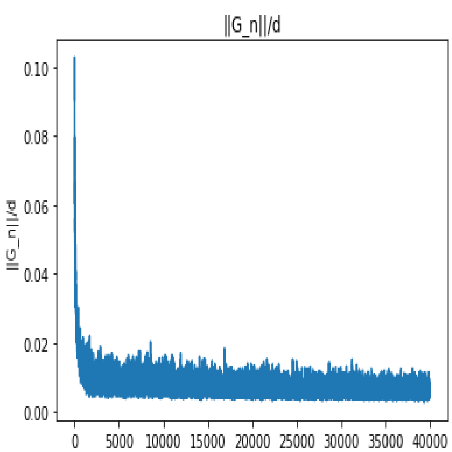


Figure 5: $||G(n)||/d$ with $h90 = 33$

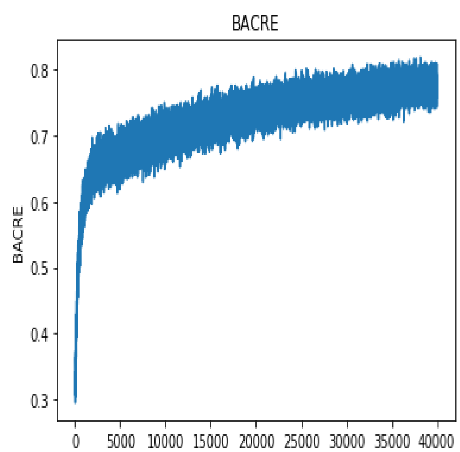


Figure 6: batch accuracy with $h90 = 33$

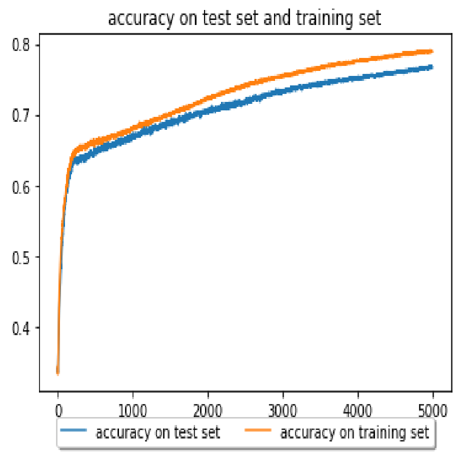


Figure 7: accuracy on test set and training set with $h90 = 33$

Figure 8: loss function on test set and traning set with $h90 = 33$

And after every epoch, we calculate the *accuracy* and *loss* for training set and test set. And plot these curves in figure 7,8. The accuracy on the training set and test set stabilize around 0.78 and 0.75, which are not very high. The loss function on the training set and test set stabilize around 0.49 and 0.54. And there is no overlearning happening in this case. we can just stop the training after 5000 epochs.

After finishing training, we compute the confusion matrix by using:

$confusion\_matrix = tf.confusion\_matrix(tf.argmax(logits1, 1), tf.argmax(Y1, 1))$

The confusion matrix on the training set is:

$$C\_M_{training} = \begin{bmatrix} 2438 & 835 & 132 \\ 686 & 2355 & 52 \\ 72 & 18 & 1462 \end{bmatrix}$$

where $C\_M_{training}(i, j)$ means the number of cases in the training set whose real class is i and predicted class is j. And the confusion matrix on the test set is:

$$C\_M_{test} = \begin{bmatrix} 1042 & 409 & 56 \\ 328 & 975 & 37 \\ 34 & 8 & 561 \end{bmatrix}$$

where $C\_M_{test}(i, j)$ means the number of cases in the test set whose real class is i and predicted class is j. We can see that the the mistake happens mostly when the MLP has to classify between class 1 and class 2. And the MLP can identify the difference between class 1 and class 3 and the difference between class 2 and class 3 better.

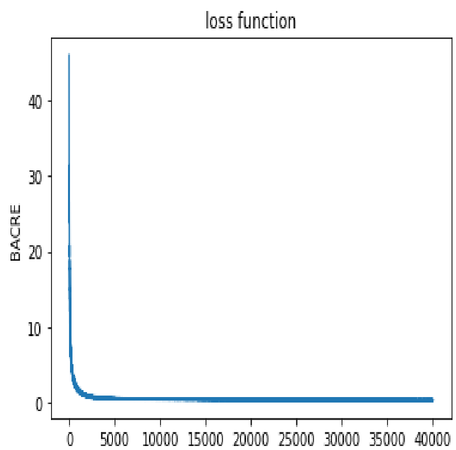


Figure 9: BACRE with $hL = 82$

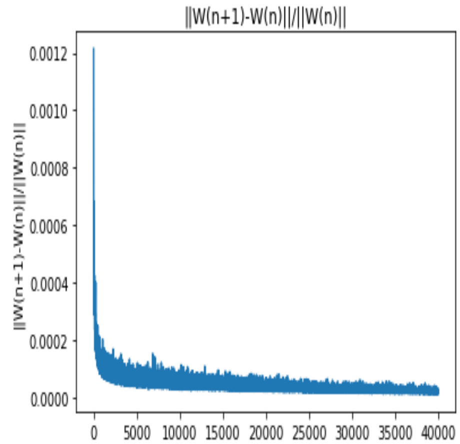


Figure 10: $||W_{n+1} - W_n||/||W_n||$ with $hL = 82$

3.2 Automatic training with the size of hidden layer $hL = 82$

For hidden layer size $hL = 82$, we keep all other part of the program same as before. And we run 5000 epochs for this situation too, which takes approximately 15 minutes. Record batch average cross entropy $BACRE(n)$, weight $W(n)$ and gradient $G(n)$. Then we plot $BACRE$, $||W(n+1) - W(n)||/||W(n)||$ and $||G_n||/d$ in figure 9,10,11. From the figure 9,10,11 we can see that $BACRE(n)$, $||W_{n+1} - W_n||/||W_n||$ and $||G_n||/d$ all decrease very fast with certain amount fluctuations and stabilize around 0.2785, 1.69×10^{-5} and 0.0039 respectively. $||G(n)||/d$ shows the average absolute value of the coordinates of vector $G(n)$. So the gradient reduce pretty well towards 0 too. Also we calculate the average accuracy on every batch and plot it in figure 12. We can see the accuracy increase with oscillation until around 0.885, which is better than the accuracy with $h90 = 33$.

And after every epoch, we calculate the *accuracy* and *loss* for training set and test set. And plot these curves in figure 13,14. The accuracy on the training set and test set stabilize around 0.88 and 0.82, which are a little better than the situation with $h90 = 33$. The loss function on the training set and test set stabilize around 0.2943 and 0.5598. And there is no overlearning happening in this case. we can just stop the training after 5000 epochs.
Similarly, after finishing training, we compute the confusion matrix by using:

$$confusion\_matrix = tf.confusion\_matrix(tf.argmax(logits1, 1), tf.argmax(Y1, 1))$$

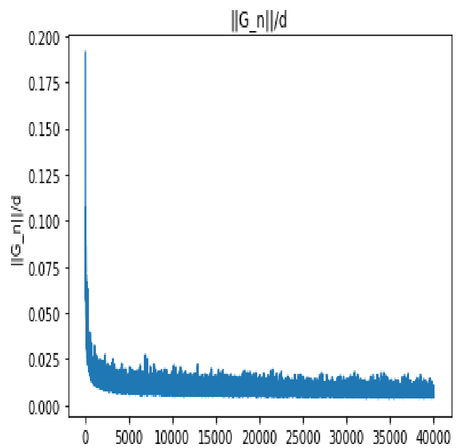


Figure 11: $||G(n)||/d$ with $hL = 82$

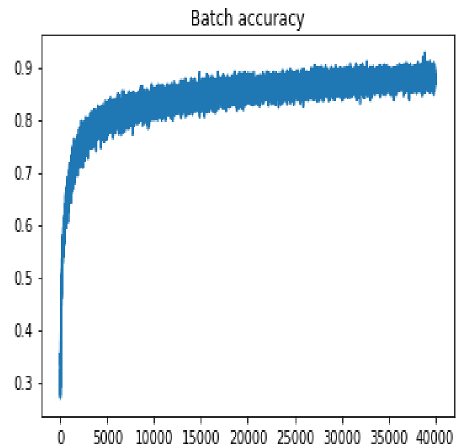


Figure 12: batch accuracy with $hL = 82$

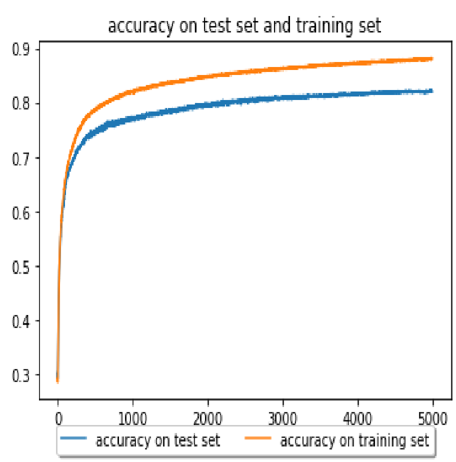


Figure 13: accuracy on test set and training set with $hL = 82$

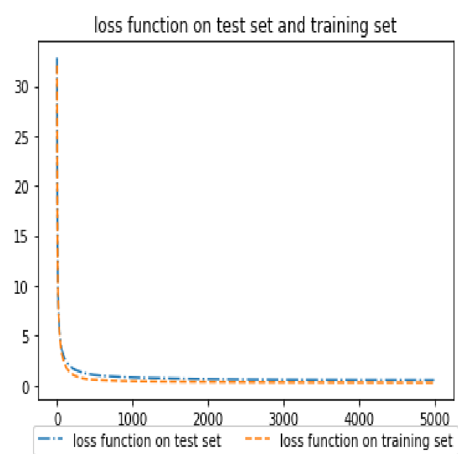


Figure 14: loss function on test set and training set with $hL = 82$

The confusion matrix on the training set is:

$$C\_M_{training} = \begin{bmatrix} 2826 & 498 & 36 \\ 374 & 2718 & 15 \\ 20 & 16 & 1547 \end{bmatrix}$$

where $C\_M_{training}(i,j)$ means the number of cases in the training set whose real class is i and predicted class is j. And the confusion matrix on the test set is:

$$C\_M_{test} = \begin{bmatrix} 1152 & 266 & 59 \\ 191 & 1074 & 40 \\ 37 & 28 & 603 \end{bmatrix}$$

where $C\_M_{test}(i,j)$ means the number of cases in the test set whose real class is i and predicted class is j. We can see that the the mistake still happens mostly when the MLP has to classify between class 1 and class 2. And the MLP can identify the difference between class 1 and class 3 and the difference between class 2 and class 3 better. However, with $hL = 82$, the MLP performs better when it has to classify cases of class 1 and class 2. Also it performs better when classifying between class 2 and class3 and between class 1 and class 3.

4 Impact of various learning options

4.1 batch size

If we change batch size to 100 instead of using 1000 and use $hL = 82$, $||W(n+1) - W(n)||/||W(n)||$, $||G(n)||/d$ and $BARCE$ stabilize around 2.19×10^{-5}, 0.01,and 0.31 respectively(figure 15,16,17), which are similar to those when $batch\_size = 1000$. However, we can see that these values oscillates more in this situation. And the accuracy on training set and test set reduce to 0.81 and 0.77 respectively(figure 18). And the loss function on the training set and test set increase to 0.42 and 0.65 respectively(figure 19). So the MLP trained with a larger batch size works a bit better in this situation.

4.2 Learning rate

If we reduce the learning rate faster as in figure 20 and use $hL = 82$ for hidden layer, $||W(n+1) - W(n)||/||W(n)||$, $||G(n)||/d$ and $BARCE$ stabilize around 1.84×10^{-7}, 0.0047,and 0.42 respectively, which are similar to those when we use previous learning rate. However, if we plot the accuracy of correct prediction and loss function on the training set and test set in figure 21,22, we can see that the accuracy on training set and test set stabilize around 0.81 and 0.77, which are lower. And loss function on the training set and test set grow a little bit to 0.43 and 0.58. So the MLP with previous learning rate which reduce more slowly works better for classification.

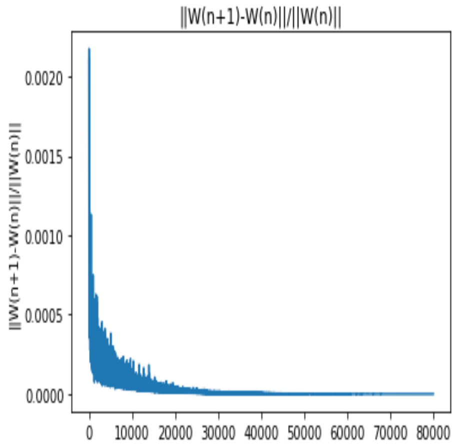


Figure 15: $||W_{n+1} - W_n||/||W_n||$ with $batch\_size = 100$

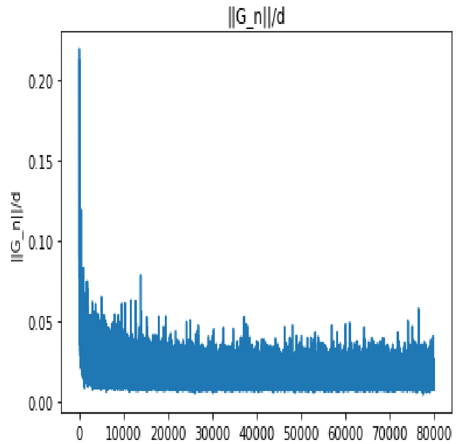


Figure 16: $||G(n)||/d$ with $batch\_size = 100$

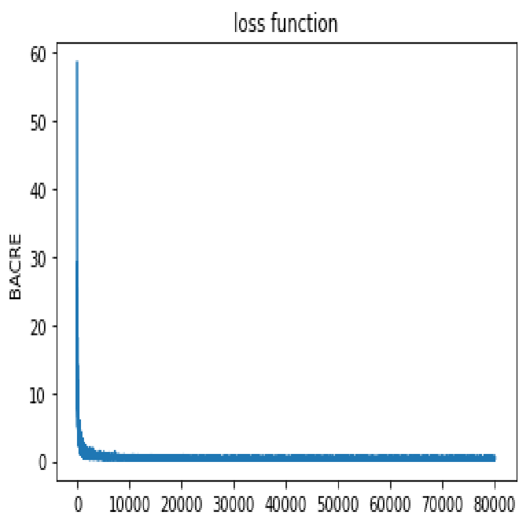
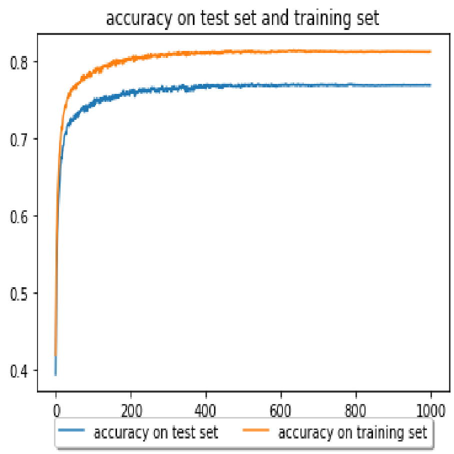


Figure 17: BACRE with $batch\_size = 100$

4.3 Hidden layer size

We use $h90 = 33$ and $hL = 82$ to train the MLP for classification. Compare the result in section 3.1 and section 3.2. We can see that the MLP with $hL = 82$ has a better performance, since the accuracy on the training set and test set are higher and the loss function on the training set and test set are lower. Also, from the confusion matrix we can see that MLP with $hL = 82$ makes less mistakes especially when classifying cases from class 1 and 2.



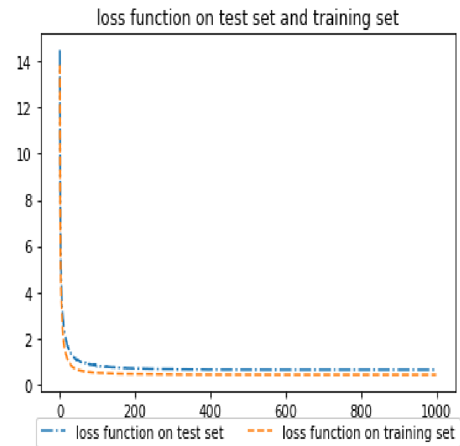


Figure 18: accuracy on test set and training set with $batch\_size = 100$

Figure 19: loss function on test set and training set with $batch\_size = 100$

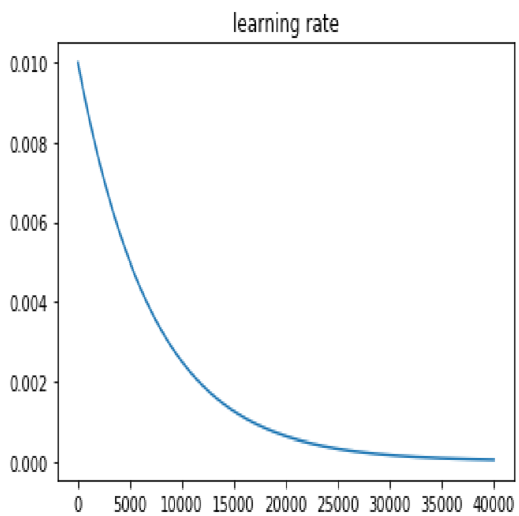


Figure 20: New learning rate

4.4 Initialization of the weights

Before we use standard normal distribution to assign values to weights and biases. Now we use normal distribution with $mean = 0$ and $std = 0.1$ to assign values to weights and biases. And we still use $hL = 82$ run 5000 epochs. Then we can see that $||W(n+1) - W(n)||/||W(n)||$, $||G(n)||/d$ and $BARCE$ stabilize around 4.38×10^{-7}, 0.0013 and 0.27 respectively, which are similar as those when using standard normal distribution. However, the accuracy on the training set

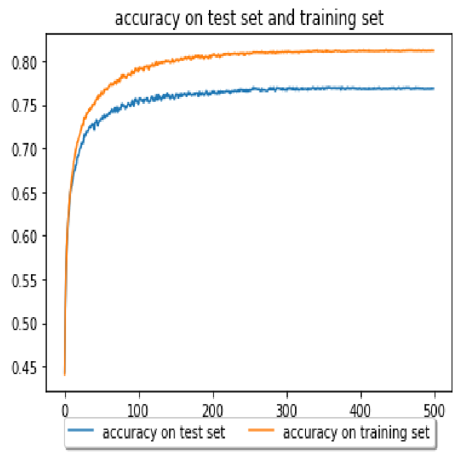


Figure 21: accuracy on test set and training set with new learning rate

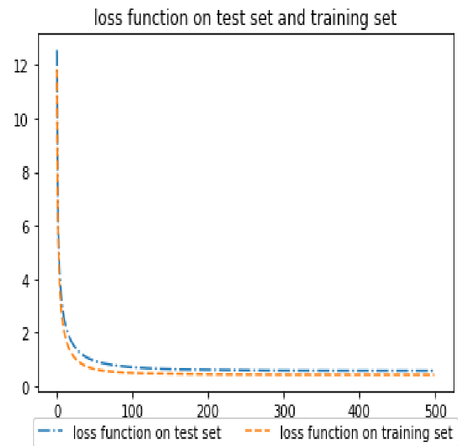


Figure 22: loss function on test set and training set with new learning rate

and test set increased to 0.91 and 0.88 respectively(figure 23). And the loss function on the training set and test set reduced to 0.2817 and 0.3435(figure 24). So by using normal distribution with $mean = 0$ and $std = 0.1$ to assign initial values to weights and biases, MLP performs better.

4.5 MLP with auto encoder

In homework 1, we built four auto encoders. And the auto encoder with $h99 = 53$ performs the best. We add the auto encoder with $h99 = 53$ in front of the MLP hidden layer $hL = 82$. Then we can see that $||W(n+1)-W(n)||/||W(n)||$, $||G(n)||/d$ and $BARCE$ stabilize around 0.0081, 0.027 and 0.46 respectively, which are a bit bigger than those without auto encoder. Also we plot the accuracy of the correct prediction and loss function on the training set and test set in figure 25,26. The accuracy on the training set and test set stabilize around 0.80 and 0.76 which are a bit lower than those without auto encoder. And the curves oscillate more. The loss function on the training set and test set stabilize around 0.45 and 0.62 which are a bit bigger than those without auto encoder. So the MLP with the auto encoder does not seem to perform better.

5 Analysis of the hidden layer behavior after completion of automatic learning

Because the MLP with hidden layer $hL = 82$ has a better performance, $h^* = 82$. We perform a PCA analysis on the hidden layer activity vectors $H(1), H(2), ..., H(8050)$. The eigenvalues of the hidden layer activity is in figure 27. And the smallest number of eigenvalues preserving 90% of the total sum of eigenvalues is 48, which

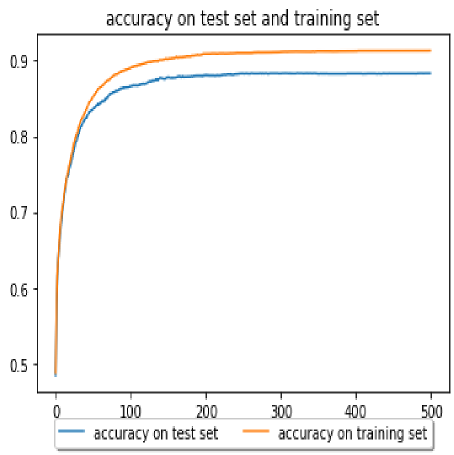


Figure 23: accuracy on test set and training set with std=0.1

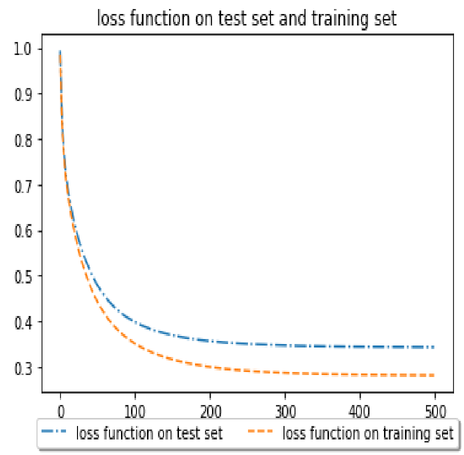


Figure 24: loss function on test set and training set with std=0.1

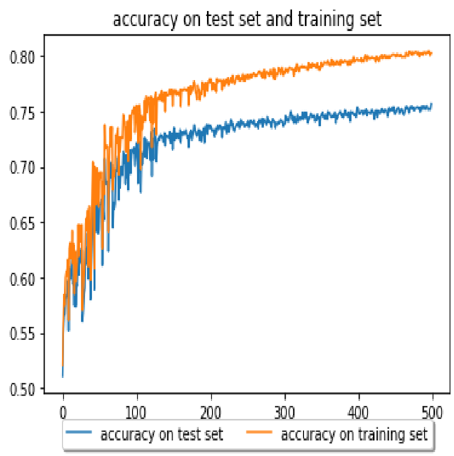


Figure 25: accuracy on test set and training set with auto encoder

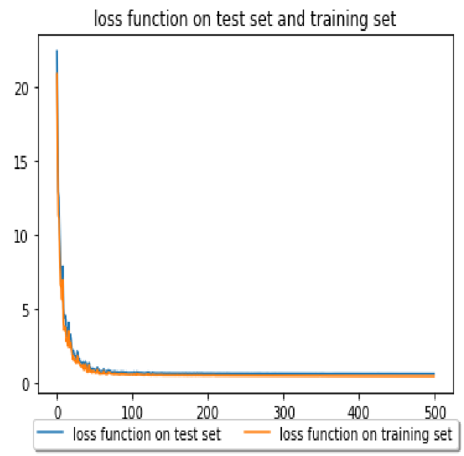


Figure 26: loss function on test set and training set with auto encoder

means there is some dependency among the neurons on the hidden layer. And we can probably reduce the size of hidden layer and get the similar performance.

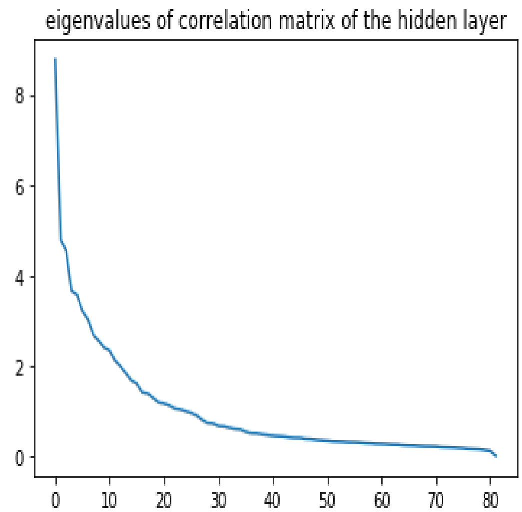


Figure 27: eigenvalues of correlation matrix of the hidden layer

Let $PROFj$ be the average hidden neurons activity over all cases belonging to class j. And we plot these three curves in figure 28,29,30. We can see that PROF1 and PROF2 have similar value range of each coordinate. However, PROF3 has much bigger value range of each coordinate. That could be the reason why the MLP has a difficult time to classify between class 1 and class 2.

We want to look for the hidden neurons which achieve best differentiation between class 1 versus class 2. So we compare PROF1 and PROF2 in figure 31. And plot $PROF1 - PROF2$ in figure 32. We can see that hidden neurons 2,5,39,63,66 which have large positive values act strongly to cases of class 1 but not class 2 and neurons 8,12,26,29,52 which have small negative values act strongly to cases of class 2 but not class 1.

We do the same thing for class 1 and class 3. Plot $PROF1vsPROF3$ in figure 33. Plot $PROF1 - PROF3$ in figure 34. We can see that coordinates of PROF3 have much bigger values than those of PROF1, which means all the neurons act much more strong to cases in class 3 than those in class1. That explains MLP can classify cases from class 1 and class 3 well. Since the the coordinates in PROF3 are always bigger than those in PROF1, we only focus on the smallest values in $PROF1 - PROF3$. And we can see that neurons 3,7,18,19,37,39,42,54,59,60,64,78 which have small negative values act strongly to class 3 but not class 1.

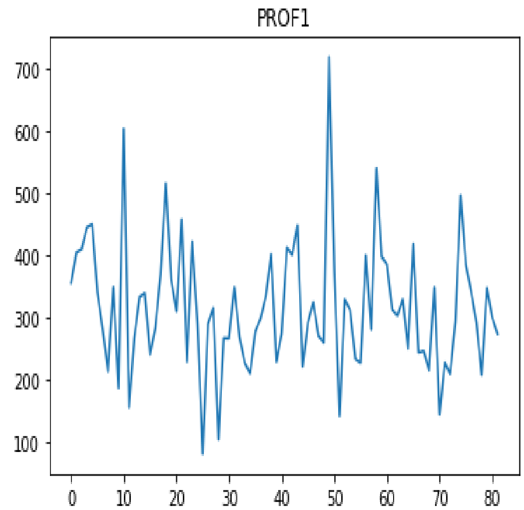


Figure 28: PROF1

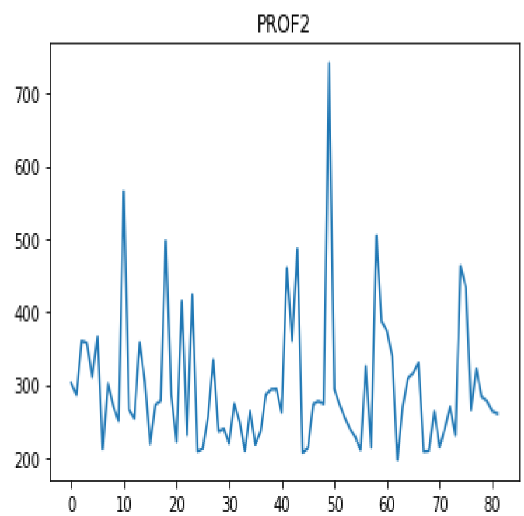


Figure 29: PROF2

We do the same thing for class 2 and class 3. Plot $PROF2 vs PROF3$ in figure 35. Plot $PROF2 - PROF3$ in figure 36. Similarly, we can see that coordinates of PROF3 have much bigger values than those of PROF2, which means all the neurons act much more strongly to the cases from class 3 than that from class 2. That explains MLP can classify cases from class 2 and class 3 well.

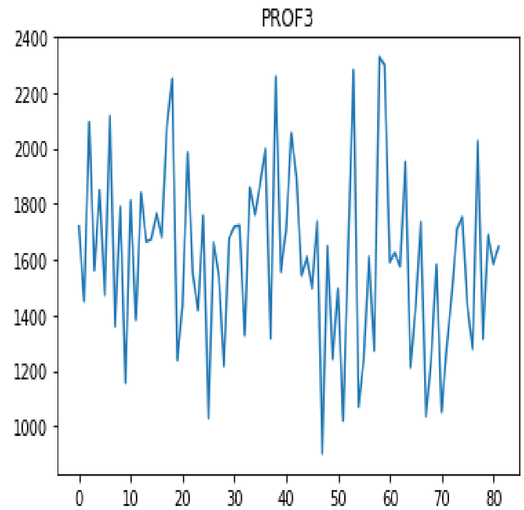


Figure 30: PROF3

Since the the coordinates in PROF3 are always bigger than those in PROF2, we only focus on the smallest values in $PROF2 - PROF3$. And we can see that neurons 3,7,18,19,34,36,37,39,54,59,60,64,78 which have small negative values act strongly to class 3 but not class 2.

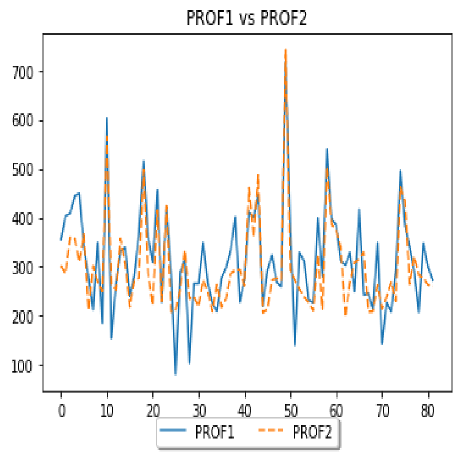


Figure 31: PROF1 vs PROF2

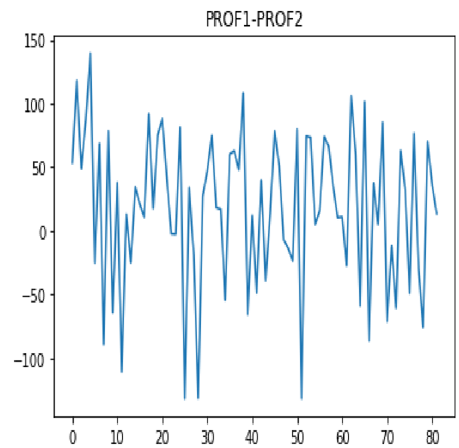


Figure 32: PROF1-PROF2

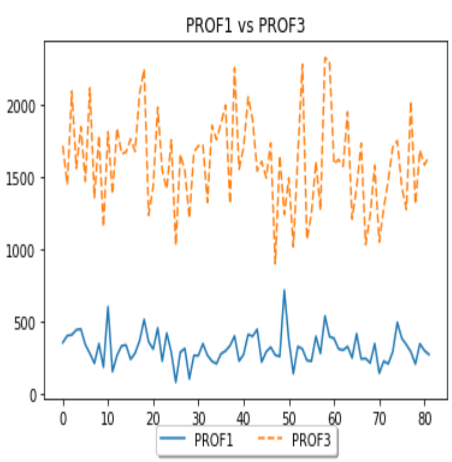


Figure 33: PROF1 vs PROF3

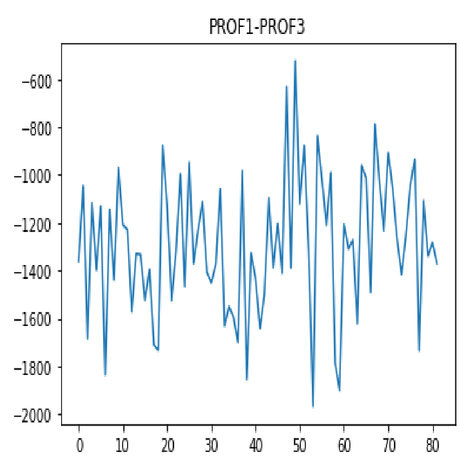


Figure 34: PROF1-PROF3

Figure 35: PROF1 vs PROF3

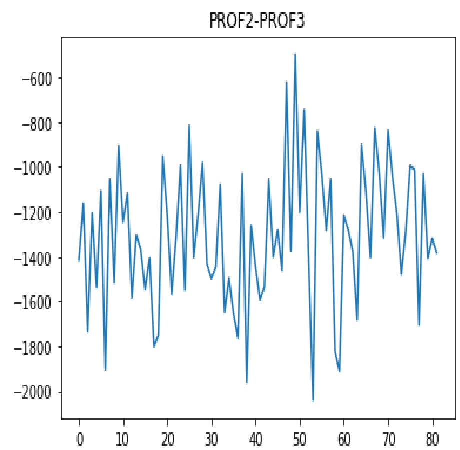


Figure 36: PROF1-PROF3