Math 6535 Homework1

Yingxue Su, Chunmei Sun

February 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.
So the number of classes is $r = 5$. And for each class $C_i, i = 1, 2, 3, 4, 5$, the number N_i of cases in C_i is 2300.

We want to train a neural network model by using this data set so that, if we have a new EEG record from a person, we will be able to identify which class the record belongs. So that we can tell whether he/she has seizure or tumor and whether the tumor locates where we record the EEG values. In order to do this, we divide this data set to a training set and a test set. The training set contains $11500 \times 0.7 = 8050$ cases and the test set contains $11500 \times 0.3 = 3450$ cases. The size of each class within the training set is $1623, 1621, 1614, 1581, 1611$. The size of each class within the test set is $677, 679, 686, 719, 689$. The proportions of each class within the the training set is $0.2016, 0.2014, 0.2005, 0.1964, 0.2001$. The proportions of each class within the test set is $0.1962, 0.1968, 0.1988, 0.2084, 0.1997$. These two proportions are quite similar. Figure 1 and figure 2 are the histograms of each class within training set and test set.

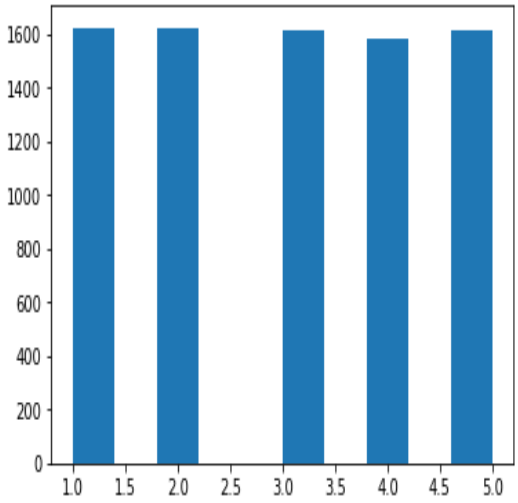

Figure 1: the histogram of the training set

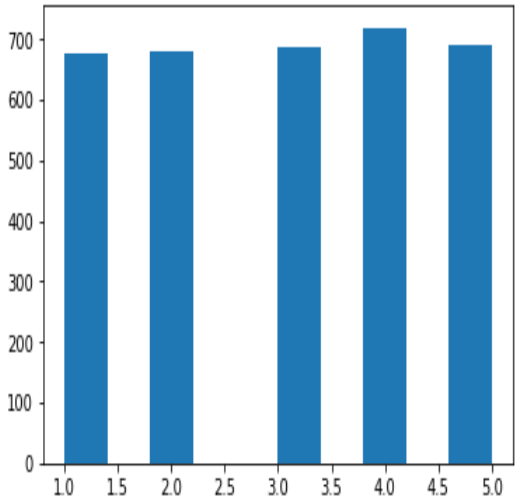

Figure 2: the histogram of the test set

2 Select 4 tentative sizes h for the hidden layer

Apply PCA analysis to the data set. Figure 3 shows the decreasing sequence of eigenvalues of the correlation matrix of the data set. $h95 = 39$, which means the first 39 biggest eigenvalues preserves 95% of the total sum of the eigenvalues. $h99 = 53$, which means the first 53 biggest eigenvalues preserves 99% of the

total sum of the eigenvalues.

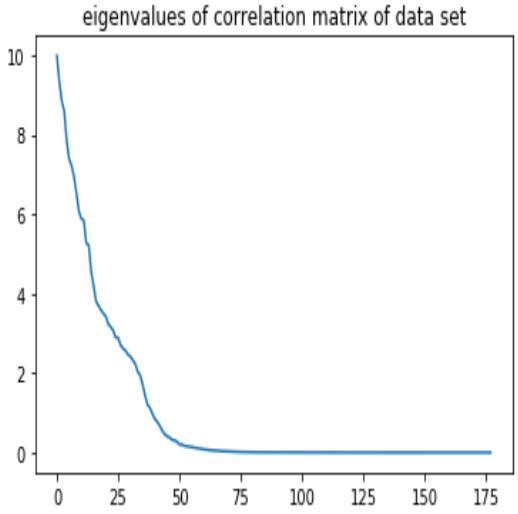

Figure 3: eigenvalues of the correlation matrix of the data set

Apply PCA analysis to the data set containing N_j cases which belongs to class j for $j = 1, 2, ..., 5$, respectively. And compute the smallest number S_j of eigenvalues preserving 99% of the total sum of the N_j eigenvalues for $j = 1, 2, ..., 5$. We get $S_1 = 53$, $S_2 = 47$, $S_3 = 51$, $S_4 = 62$ and $S_5 = 69$. Then $H99 = S_1 + S_2 + ... + S_5 = 282$ and $HH = 3 \times H99 = 846$.

3 Implement auto-encoding by gradient descent to minimize the MSE

3.1 Auto-encoding with the size of hidden layer $h < p_0 = 178$

For $h95 = 39$ and $h99 = 53$, they are both less than $p_0 = 178$. We choose $RELU$ function as our response function for these two cases. And we use tensorflow to implement auto encoder. 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 $training\_epochs = 1000$ and $global\_step$ is calculated by

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

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

And the following figure 4 shows how the leaning rate decreases during learning.

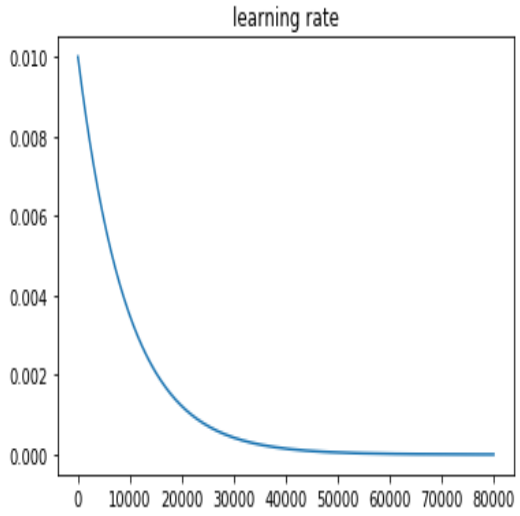

Figure 4: the decay of learning rate $\epsilon(n)$

We define the auto encoder with one hidden layer as:

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

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

We implement MSE loss function and gradient descent optimizer as following:

$$cost = tf.sqrt(tf.reduce\_mean(tf.losses.mean\_squared\_error(logits, Y)))$$

$$optimizer = tf.train.GradientDescentOptimizer(learning\_rate = learning\_rate).minimize(cost)$$

where $logits$ is the output we get from the auto encoder and Y is the correct output, which is just input.
For batch learning, we choose batch size $B = 100$. For every train, we generate 100 random number between 0 and 8050 by the following code:

$$randidx = np.random.randint(8050, size = batch\_size)$$

And these 100 numbers are the indexes of rows we choose from the training data set for this train. So we use $batch\_xs$ chosen as following for this train:

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

And for each epoch, we run the number of $total\_batch = [8050/B] = 80$ times training to approximately go through all the data in the training set once.
For $h95 = 39$, we run 1000 *epochs* and record the root of mean squared error $RMSE_n$, the norm of weight change $||W_{n+1} - W_n||$ and $||G_n||$ the norm of the gradient of MSE_n for every step $n = 1, ..., 8000$. We plot these three curves in figure 5,6,7.

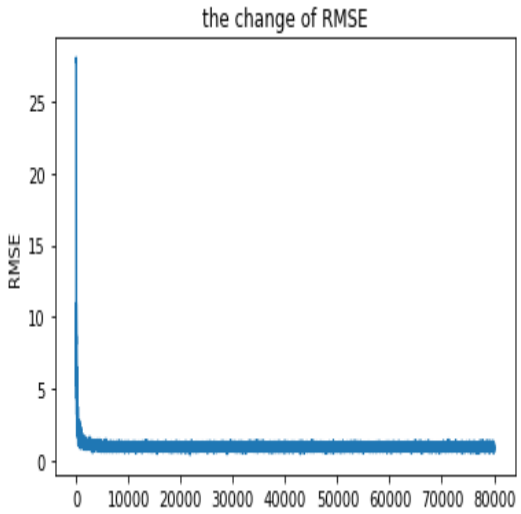

Figure 5: RMSE with $h95 = 39$

From the figure 5,6,7, we can see that $RMSE_n$, $||W_{n+1} - W_n||$ and $||G_n||$ all decrease very fast with certain amount fluctuations and stabilize around 0.991, 2.406×10^{-7} and 0.110 respectively. All these three figures become stable after around 30000 steps. The figure of $||G_n||$ bounces a bit more than the figure $||W_{n+1} - W_n||$ is because $\frac{1}{\epsilon(n)}$ becomes bigger as n grows.
And every 20 steps, we calculate the $RMSE$ for training set and test set. And plot these two curves in figure 8. We can see that the $RMSE$ on these two sets reduce very fast and stabilize around 1.41 on test set and 1.038 on traning set. And the $RMSE$ of test set is little bigger than the $RMSE$ of the training set, which can been seen more clearly in figure 9 with $total\_eoch = 100$. And there is no overfit/overlearning phenomenon happening. And we can stop the training when the performance on both sets stabilize with $n = 30000$.

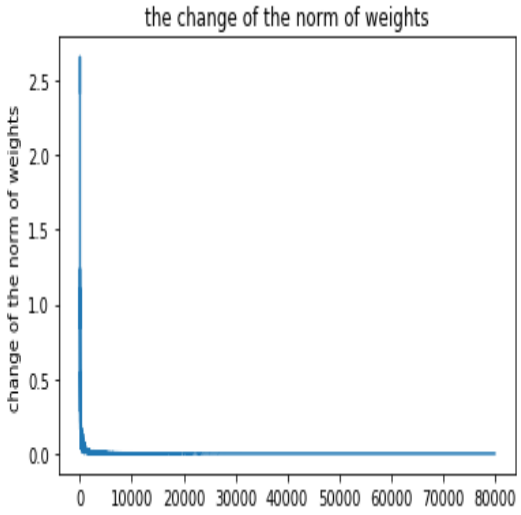

Figure 6: $||W_{n+1} - W_n||$ with $h95 = 39$

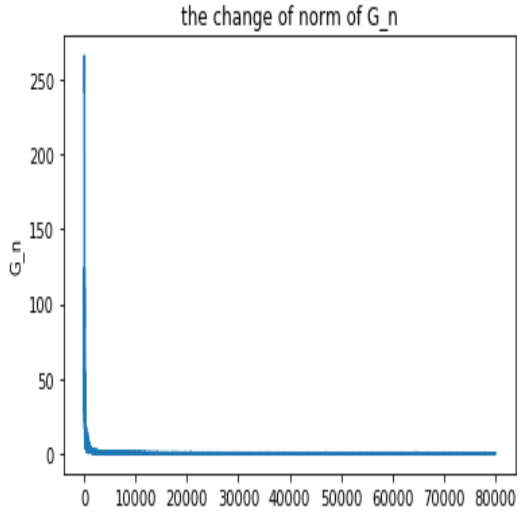

Figure 7: $||G_n||$ with $h95 = 39$

For $h99 = 53$, we use the exactly same function and parameters as when $h95 = 39$. And we plot $RMSE_n$, $||W_{n+1} - W_n||$ and $||G_n||$ in figure 10,11,12. From these 3 figures we can see that $RMSE_n$, $||W_{n+1} - W_n||$ and $||G_n||$ decrease very fast with certain amount fluctuations, similar as before when $h95 = 39$. These three curves stabilize around 1.209, 6.7×10^{-8} and 0.0306 respectively. And the figure of $||G_n||$ bounces a bit more than the figure $||W_{n+1} - W_n||$ is because $\frac{1}{\epsilon(n)}$

Figure 8: performance on training set and test set with 1000 total epochs and $h95 = 39$

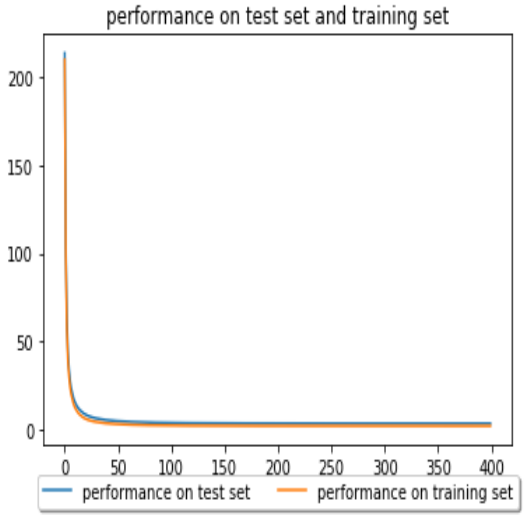

Figure 9: performance on training set and test set with 100 total epochs and $h95 = 39$

becomes bigger as n grows. In this situation, there is no overlearning happening either.
For every 20 steps, we calculate $RMSE$ for both training set and test set. And we plot these two curves in figure 13. We can see that the $RMSE$ of both sets

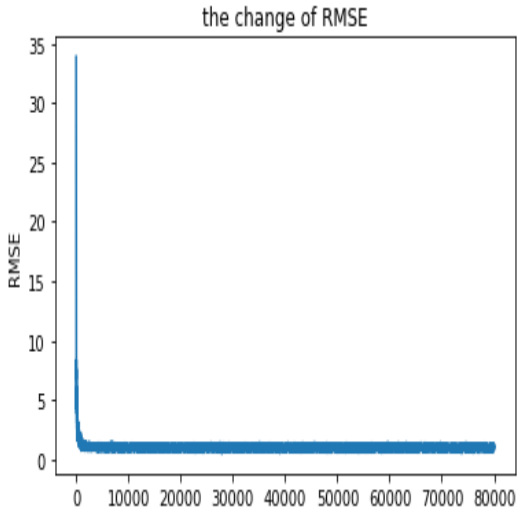

Figure 10: RMSE with $h99 = 53$

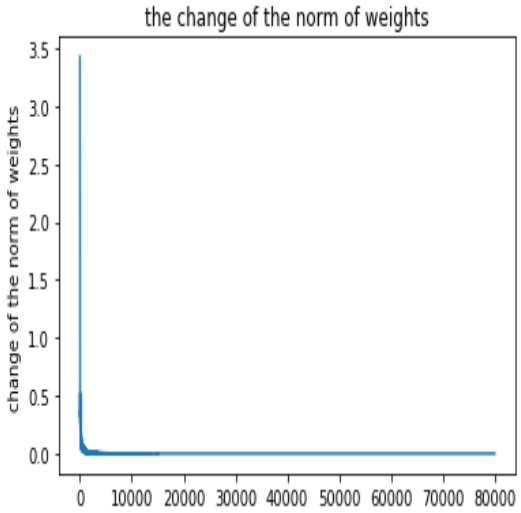

Figure 11: $||W_{n+1} - W_n||$ with $h99 = 53$

reduce very fast and stabilize around 1.43 on test set and stabilize around 1.038 on traning set. However, the $RMSE$ of the test set is a little bigger than the $RMSE$ of the training set. And there is no overlearning happening in this case. We can stop leaning when the loss function stabilizes with $n = 30000$.

Comparing the performance with hidden layer size $h95 = 39$ and performance

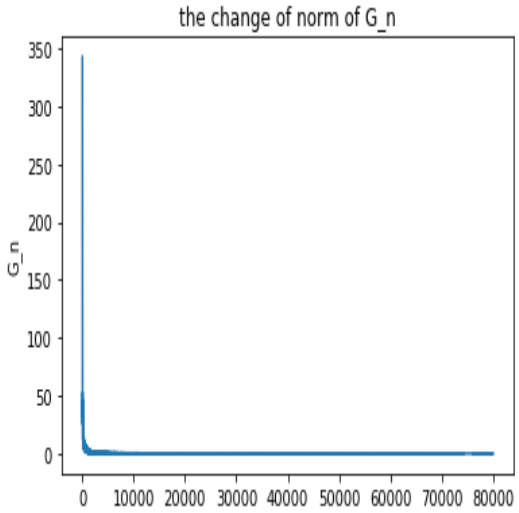

Figure 12: $||G_n||$ with $h99 = 53$

Figure 13: $||G_n||$ with $h99 = 53$

with hidden layer size $h99 = 53$. For $h99 = 53$, the $RMSE$ of the training set becomes stable around 1.04, the $RMSE$ of the test set becomes stable around 1.59. For $h95 = 39$, the $RMSE$ of the training set becomes stable around 1.02, the $RMSE$ of the test set becomes stable around 1.54. So for a smaller size hidden layer $h95 = 39$, the performance of the training set and test set is better than that with hidden layer size $h99 = 53$.

3.2 Auto-encoding with the size of hidden layer $h > p_0 = 178$

For $H99 = 282$ and $HH = 846$, we use $scaler = MinMaxScaler()$ to scale all the data between $[0, 1]$. Then we can use *sigmoid* function as response function. 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 $training\_epochs = 1000$ and $global\_step$ is calculated by

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

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

We define the auto encoder with one hidden layer as:

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

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

We define the cross entropy between probability distribution $[a, 1-a]$ and $[b, 1-b]$ as following:

$$crossentropy(a, b) = -a*tf.log(b)-(1-a)*tf.log(1-b)-b*tf.log(a)-(1-b)*tf.log(1-a)$$

We implement loss function and gradient descent optimizer as following:

$$cost = tf.reduce\_mean(tf.losses.mean\_squared\_error(logits, Y))+c*crossentropy(alpha, act)$$

$$optimizer = tf.train.GradientDescentOptimizer(learning\_rate = learning\_rate).minimize(cost)$$

where $logits$ is the output we get from the auto encoder and Y is the correct output, which is just input. $alpha = 0.01$ is the target average activity. act is the average activity on the hidden layer. $c = 5$ is a constant.

For batch learning, we choose batch size $B = 100$. For every train, we generate 100 random number between 0 and 8050 by the following code:

$$randidx = np.random.randint(8050, size = batch\_size)$$

And these 100 numbers are the indexes of rows we choose from the training data set for this train. So we use $batch\_xs$ chosen as following for this train:

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

And for each epoch, we run the number of $total\_batch = [8050/B] = 80$ times training to approximately go through all the data in the training set once.

Figure 14,15,16, are the curves for $RMSE_n$,$||W_{n+1} - W_n||$ and $||G_n||$ with $H99 = 282$, respectively. We can see that all these three figures reduce very fast in the beginning and become stable around valuse 0.175, 2.457×10^{-8} and 0.035 respectively. However, these curves oscillate much more during the process than the situation with $h95$ and $h99$. They all oscillate strongly between step 10000 to 30000. We think the oscillation is because we add the cross entropy term to the cost function and we are minimizing $MSE + c \times crossentropy$ as a whole term in stead of minimizing MSE only.

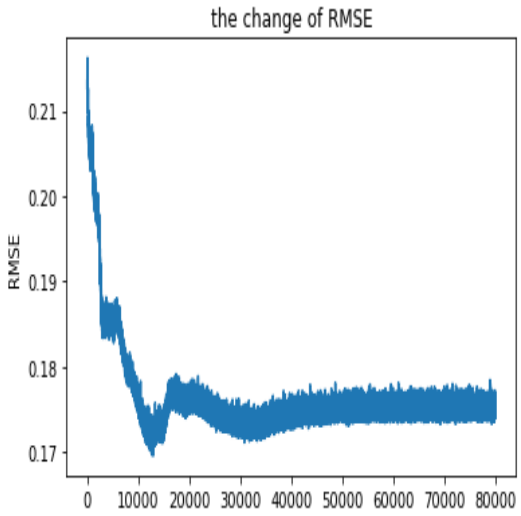

Figure 14: RMSE with $H99 = 282$

However, we can see the loss function and the cross entropy decrease pretty smoothly instead of oscillating strongly in figure 17,18. And both of these two curves become stable around values 2.3257 and 1.0468 around step 50000.

Since we add sparsity to the hidden layer, we also want the average activity in the hidden layer to become small towards the target $alpha = 0.01$. We plot the hidden layer average activity in figure 19. We can see the the average activity decreases and becomes stable around value 0.17 which is still quite bigger than 0.01.

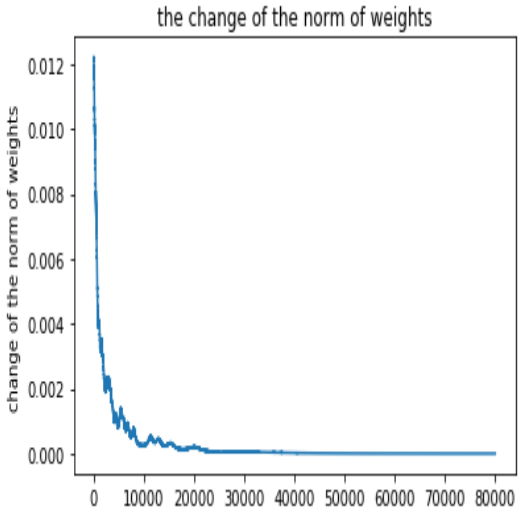

Figure 15: $||W_{n+1} - W_n||$ with $H99 = 282$

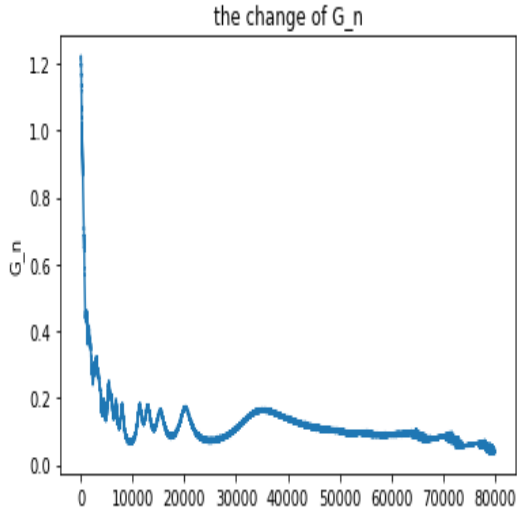

Figure 16: $||G_n||$ with $H99 = 282$

For every 20 steps, we calculate the loss function and the RMSE on training set and test set and plot these two figures in figure 20,21. We can see two curves of loss function and RMSE decrease almost exactly same. However, the loss on the test set are still a little bit higher than the loss on the training set, which is normal. And the RMSE of the test set stabilizes with a lower value than that of the training set, which is not expected. The loss and RMSE on the test

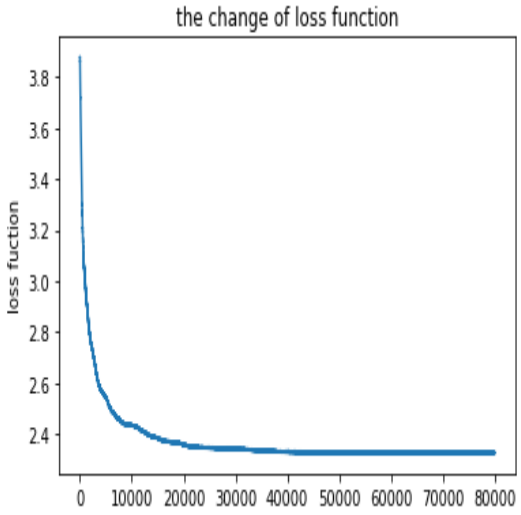

Figure 17: loss function with $H99 = 282$

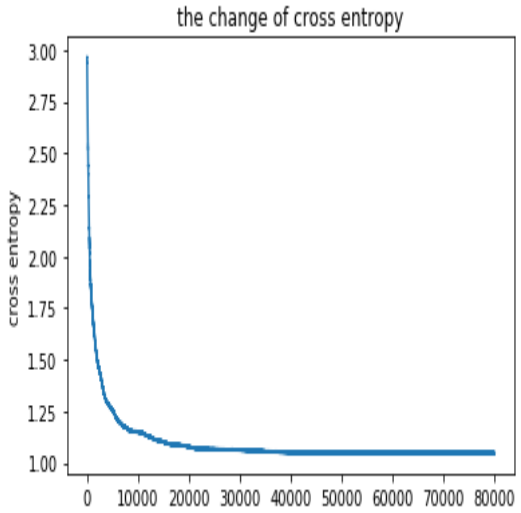

Figure 18: cross entropy with $H99 = 282$

set stabilizes around 5.411 and 0.4139. The loss and RMSE on the training set stabilizes around 5.409 and 0.4141. And there is no overlearning happening in this case. We can stop the learning when the loss function stabilizes with $n = 50000$.

Figure 22,23,24, are the curves for $RMSE_n$,$||W_{n+1}-W_n||$ and $||G_n||$ with $HH =$

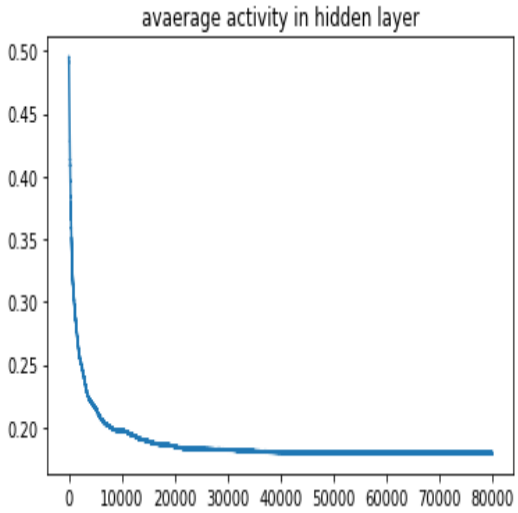

Figure 19: average activity in hidden layer with $H99 = 282$

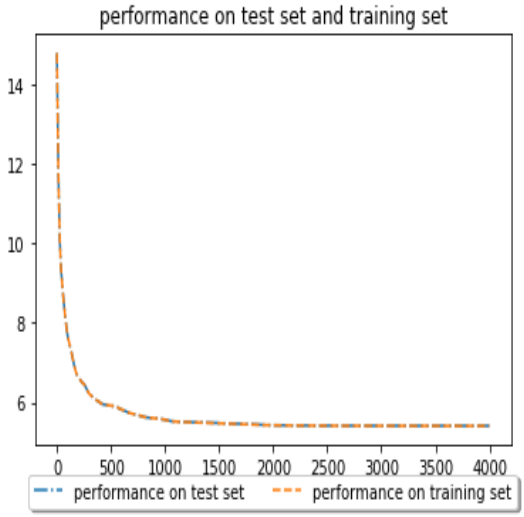

Figure 20: loss function on training set and test set with $H99 = 282$

846, respectively. We can see that, similarly as before, all these three figures reduce very fast in the beginning and become stable around values 0.1964,1.32×10^{-7}, 0.0764 respectively. $||W_{n+1} - W_n||$ becomes very close to 0 at the end. However, $RMSE$ increased twice during the process. And it oscillates even more than the situation with $H99$ after it stabilizes.

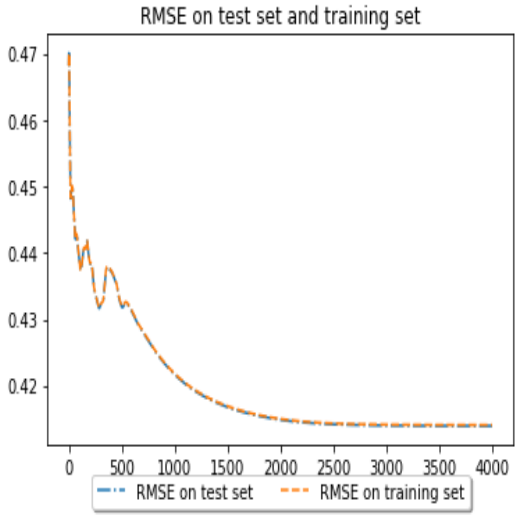

Figure 21: RMSE on training set and test set with $H99 = 282$

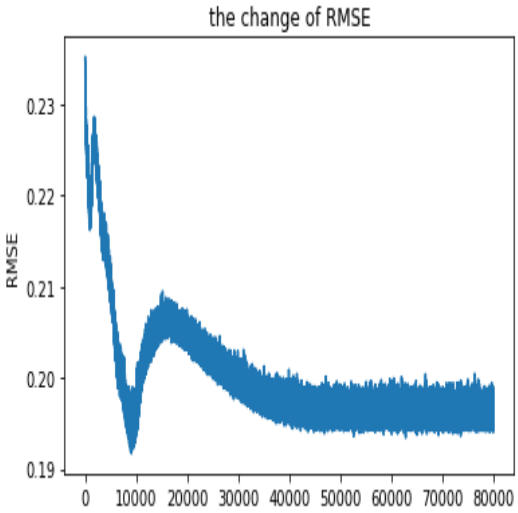

Figure 22: RMSE with $HH = 846$

Similarly as before, we can see the loss function and the cross entropy decrease pretty smoothly in figure 25,26. And both of these two curves become stable around step 40000. And the cross entropy becomes stable around value 1.28. Loss function becomes stable around the value 2.47.

For $HH = 846$, we also want the average activity in the hidden layer to become

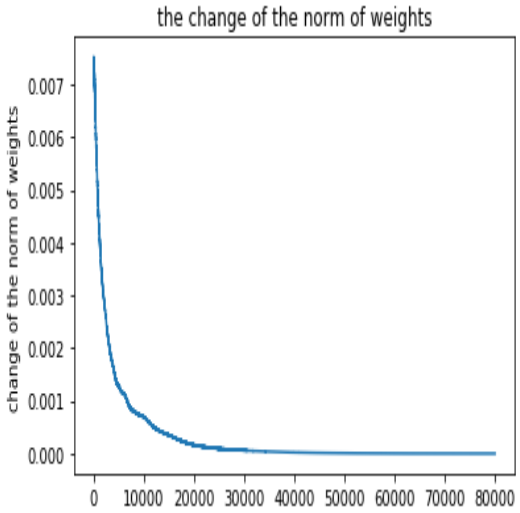

Figure 23: $||W_{n+1} - W_n||$ with $HH = 846$

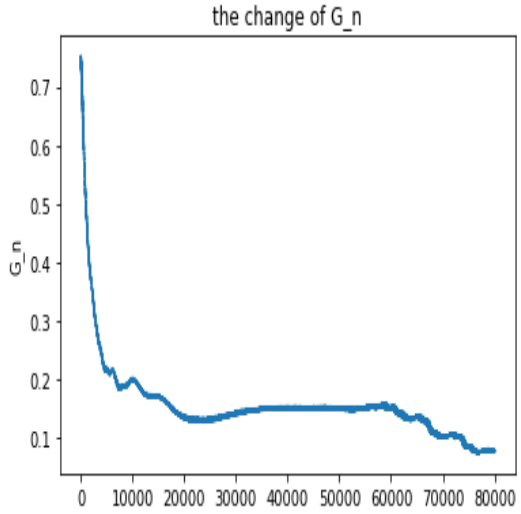

Figure 24: $||G_n||$ with $HH = 846$

small towards target $alpha = 0.01$. We plot the hidden layer average activity in figure 27. We can see the the average activity becomes stable around value 0.22 which is even bigger than the situation with $H99 = 846$.

For every 20 steps, we calculate the loss function and RMSE on training set and test set and plot these two figures in figure 28,29. We can see two curves of loss

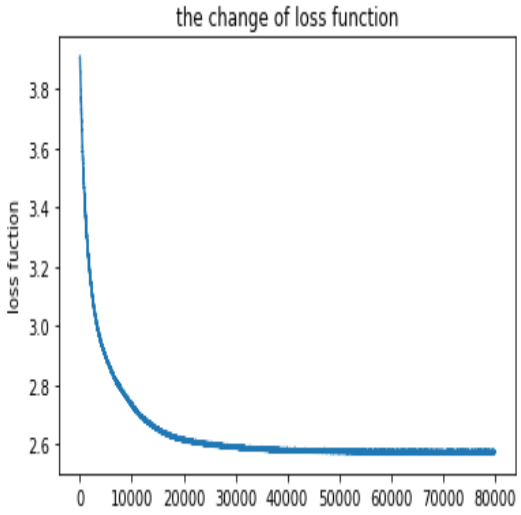

Figure 25: loss function with $HH = 846$

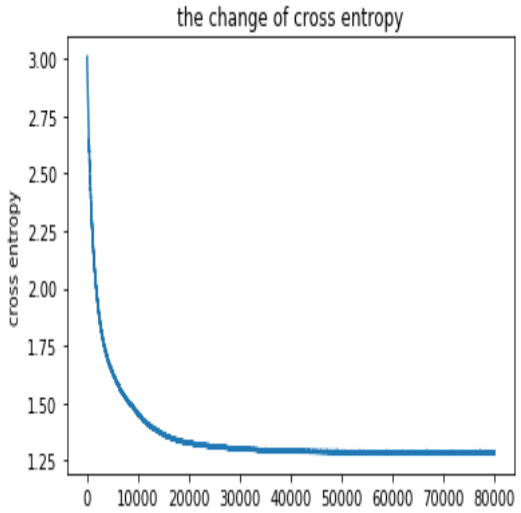

Figure 26: cross entropy with $HH = 846$

functions and two curves of RMSE decrease almost exactly same again. And the loss and RMSE on the test set are still a little bit higher than the loss on the training set. The loss and the RMSE on the test set stabilizes around 6.626 and 0.4552. The loss and the RMSE on the training set stabilizes around 6.613 and 0.4550. And there is no overlearning happening in this situation. We can stop the learning when the loss function stabilizes with $n = 50000$.

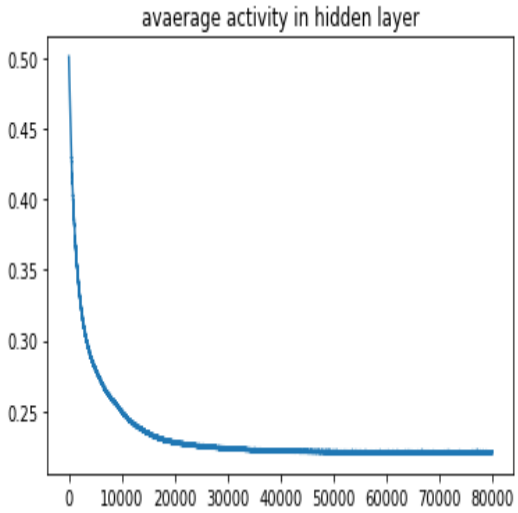

Figure 27: average activity in hidden layer with $HH = 846$

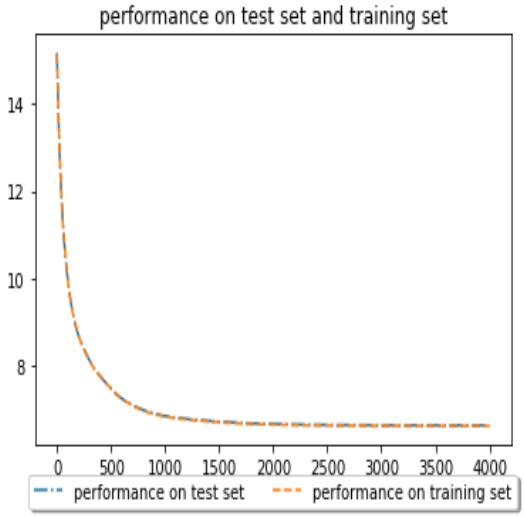

Figure 28: loss function on training set and test set with $HH = 846$

4 Impact of various learning options

4.1 Batch size

If we use $h95 = 39$ as hidden layer size with $batch\_size = 1000$ instead of 100. $||W_{n+1} - W_n||$, $RMSE$ stabilize around 0.00039 and 1.055 which are both bigger

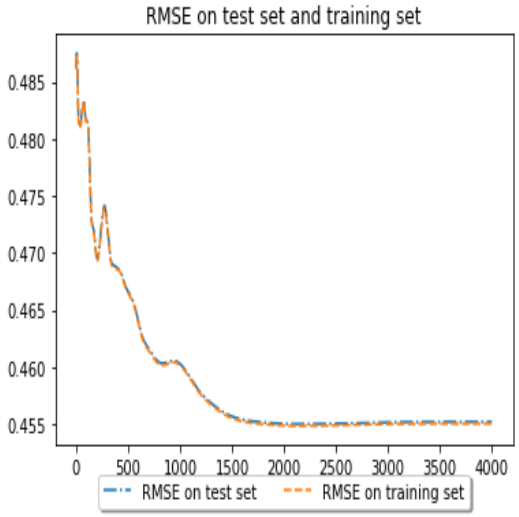

Figure 29: RMSE on training set and test set with $HH = 846$

than those values with $batch\_size = 100$. And the $RMSE$ on the training set and test set increased a little bit to 1.045 and 1.578 instead of 1.038 and 1.413 with $batch\_size = 100$. So smaller batch size works better for our data.

4.2 Learning rate

If the learning rate decrease faster with hidden layer size $h95 = 39$ in figure 30, $||W_{n+1} - W_n||$,$||G_n||$ and $RMSE$ becomes stable much faster in figure 31,32,33. However, the $RMSE$ stabilize around 1.355 which is bigger than the previous value 0.991. And the RMSE on the training set and test set increases to 1.641 and 2.396. So with bigger gradient descent step size, the network learns faster. However, the network does not learn as well.

4.3 dimension h of hidden layer

Compare the learning with $h95 = 39$ and $h99 = 53$, we can see that $||W_{n+1} - W_n||$, $RMSE$ and $||G_n||$ have smaller stabilized values with $h95$. And the $RMSE$ on the test set and training set with $h95$ are smaller than those with $h99$. So training with $h95 = 39$ seems to work better.

Compare the learning with $H99$ and HH, we can see that $||W_{n+1} - W_n||$, $RMSE$, $||G_n||$, $average\_activity$ and $cross\_entropy$ have smaller stabilized value with $H99$. And the $RMSE$ and loss function on the test set and training set with $H99$ are smaller than those with HH. So training with $H99$ seems to work better with sparsity.

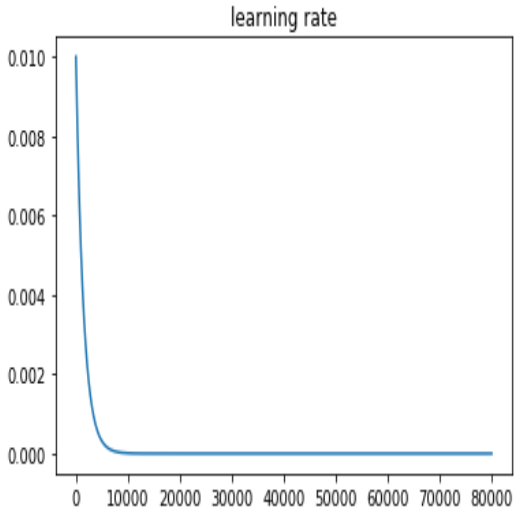

Figure 30: New learning rate

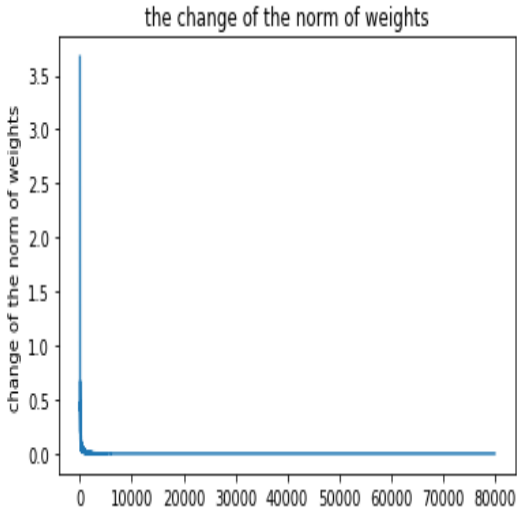

Figure 31: $||W_{n+1} - W_n||$ with new learning rate

Compare the learning with $H99$ and $h95$, $||W_{n+1} - W_n||$, $||G_n||$ and $RMSE$ seem to have smaller stabilized value with $H99$. And for training with $H99$, the $RMSE$ on the test set and training set are smaller. So with hidden layer size $H99 = 282$, the auto encoder seems to work the best.

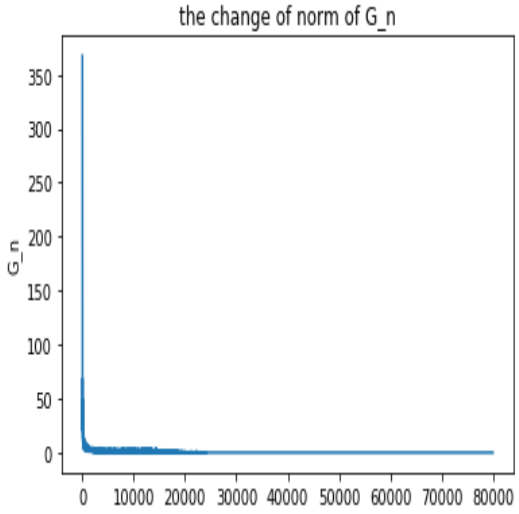

Figure 32: $||G_n||$ with new learning rate

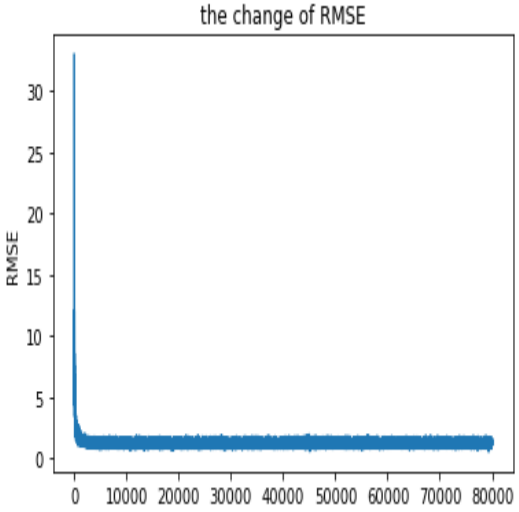

Figure 33: RMSE with new learning rate