Homework 2 for Deep Learning and Data Mining

Xiaoqian Chen

June 3, 2019

1 Data Set

In this homework, we use a new image classification data set called Fashion MNIST. This new data set contains images of various articles of clothing and accessories, such as shirts, bags, shoes, and other fashion items. I t is still a easy data , which do not require the professional background to understand data. The old data set is the USPS collection of handwritten digits. The study of digits is too easy and data size is small, which only has 1,100 examples for each class. For this task, Let 's try a little bit larger data size and more complicated images.

The Fashion MNIST training set contains 60,000 examples, and the test set contains 10,000 examples. Each example is a 28x28 gray scale image (just like the images in the original MNIST), associated with a label from 10 classes (t-shirts, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, and ankle boots). There are 6,000 examples for each class in the training set. Since we prefer to reduce the number of classes to the 3 largest classes, we choose three shoes items, including sandals, sneakers and ankle boots. The reduced training set size is 18,000 and the test set contains corresponding 3,000 example.

(a) sandals

(b) sneakers

(c) ankle boots

Figure 1: data examples

The images have been size-normalized and centered in a fixed-size image (28×28 pixels) with values range from 0 to 256, where 0 is black and 256 is white. For simplicity, each image has been flattened and converted to a 1×784 vector.
For output layer, there are 3 distinct classes (sandals, sneakers and ankle boots), presented by 3 independent standard basis vector $\{e_1, \ldots, e_3\}$, where $e_1 = [1, 0, 0]'$ stand for the first class "sandals" and so on.

Data set description		
	Training set	Test set
Set Size	18,000	3,000
Value Range of Input Descriptors	{0, ..., 256}	{0, ..., 256}
Size of each Input Descriptors	784	784
Size of CL1,CL2,CL3	(6,000 6,000, 6,000)	(1,000 1,000 1,000)

Table 1: Data set description

2 the MLP architecture of an Automatic Classifier with r =3 classes

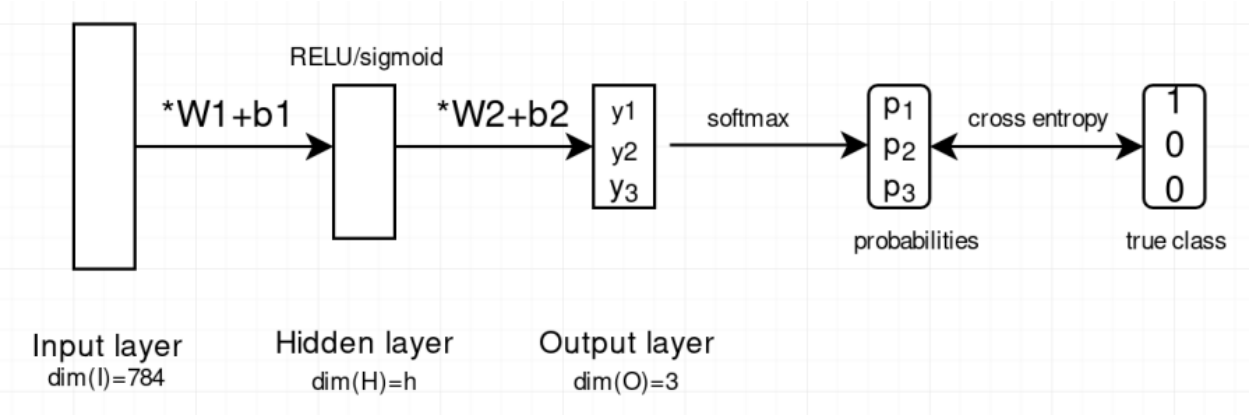

Figure 2: MLP architecture

Multilayer Perception Architecture for an auto-encoder is a 3 layers fully-connected network, having an input layer, an output layer and one hidden layers connecting them. Neurons between two adjacent layers are fully pairwise connected, but neurons within a single layer share no connections. There is only one hidden layer with dimension h. The dimension of the output layer is 3. The response function is applied for all neurons in the hidden layer, such as sigmoid function, RELU function. In the implement, we try both of them.

The softmax function applied to the output from the output layer, which lead to out final output. Assume we have $O_k = (y_1, y_2, y_3)$ from the output layer.There is a probability vector as output of softmax function, such that $OUT_k = (p_1, p_2, p_3)$, and $p_1 + p_2 + p_3 = 1$.

$$OUT_k = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} \frac{e^{y_1}}{e^{y_1}+e^{y_2}+e^{y_3}} \\ \frac{e^{y_2}}{e^{y_1}+e^{y_2}+e^{y_3}} \\ \frac{e^{y_3}}{e^{y_1}+e^{y_2}+e^{y_3}} \end{bmatrix} \tag{1}$$

The final classification of X_k is computed from OUT_k by identifying the class have the biggest probabilities.

$$classificationofX_k = argmax(OUT_k) \tag{2}$$

For example, assuming the final output is $OUT_K = (0.12, 0.81, 0.07)$, the classification will be 2, since the second probability $0.81 > 0.12 > 0.07$.

The goal of the MLP classifier is to obtain output probability vector OUT_k very close to the binary vector encoding the true class of the input vector, which is one of $\{(1,0,0),(0,1,0),(0,0,1)\}$. The cross entropy is used to measure the performance quality.

3 Select 2 tentative sizes h for the hidden layer

There are two alternative sizes for the hidden layer: h90 and hL. Firstly, estimate one small but plausible value for h, namely h = h90. Apply PCA analysis to the entire training data set, h90 is the smallest number of eigenvalues, that preserves 90% of the total sum of eigenvalues.

$$90\% = \frac{\sum_{i=1}^{h90} s_i}{\sum_{i=1}^{784} s_i} \tag{3}$$

Figure 3 shows the decreasing sequence of eigenvalues of the correlation matrix of the data set. h90 = 97, which means the first 97 biggest eigenvalues preserves 90% of the total sum of the eigenvalues.

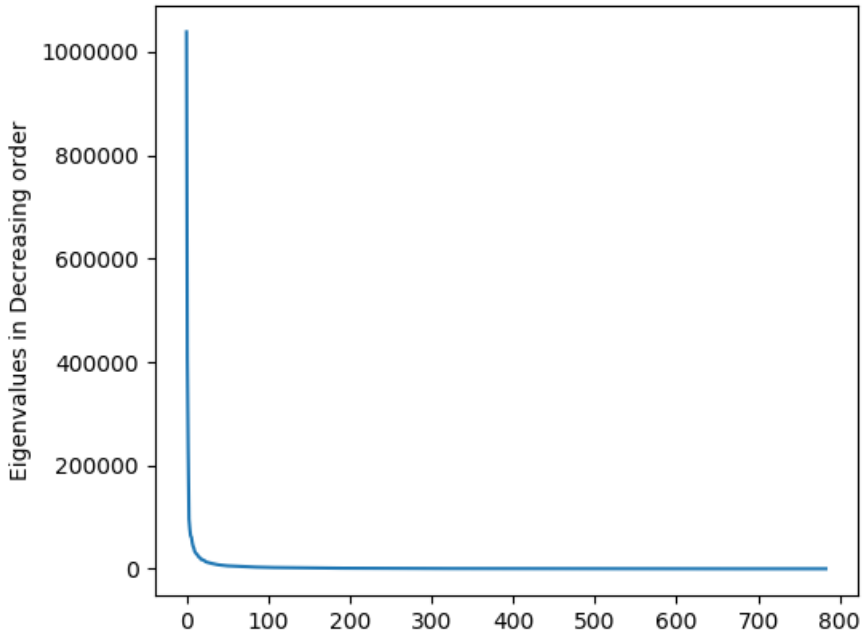

Figure 3: Eigenvalues in decreasing order

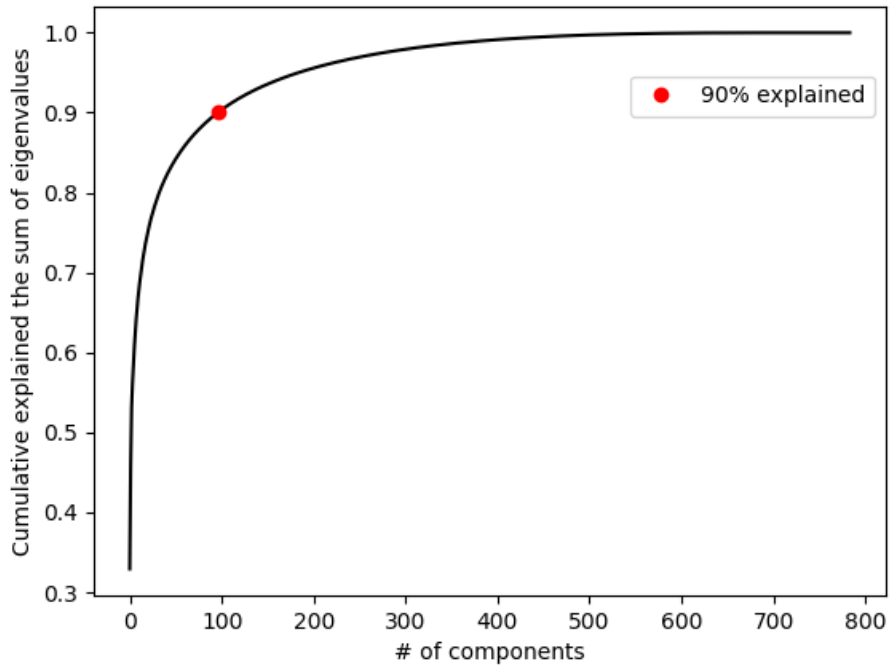

Figure 4: Cumulative Explained the sum of eigenvalues

Secondly, estimate one larger plausible value hL, which apply PCA analysis to data corresponding to 3 different data, get h90 for each class, then sum them up. To be specific, apply PCA analysis to the set of M_j input vectors corresponding to the class C_j, with j=1,2 3 to generate M_j eigenvalues in decreasing order, and compute the smallest number "U_j" of eigenvalues preserving 90% of the total sum of these M_j eigenvalues. We get U1=135 for first class, U2=79 for second class and U3=82 for the third class. Hence, $hL = U_1 + U_2 + U_3 = 135 + 79 + 82 = 296$.

4 implement automatic training for h= h90, h=hL

In this project, we train our neural network by Python3.5 with Tensorflow, which is an open-source machine learning library for research and production.
For initialization of the weights, we use the function

```
init=tf.random_normal().
```

The learning algorithm is use gradient descent to minimize Average CROSS ENTROPY error between computed and true values to get a adapted network. For the Average CROSS

ENTROPY error (ACRE), assume that the total number of input examples is N, for k_th example ($1 \leq k \leq N$), we have the final probability output OUT_k after learning and the true class vector $Target_k$ corresponding to the input.

$$ACRE = -\frac{1}{N}(\sum_{k=1}^{N} Target_k \cdot log(OUT_k)) \tag{4}$$

The sigmoid function and the RELU function are two response function , which applied in the hidden layer. We try both of them, the sigmoid have better performance ,we choose sigmoid.

$$sigmoid(x) = \frac{1}{1+e^{-x}} \tag{5}$$

$$RELU(x) = max\{0, x\} \tag{6}$$

```
fc1 = fc_layer(x, h, 'Hidden_layer', use_sigmoid=True)
output = fc_layer(fc1, n_classes, 'Output_layer',
                                        use_sigmoid=False)
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits
                        (labels=y, logits=output), name='loss')
optimizer = tf.train.GradientDescentOptimizer(learning_rate
                                    =learning_rate).minimize(loss)
```

We need terminologies like epochs, batch size, iterations only when the data is too big which happens all the time in machine learning and we cant pass all the data to the computer at once. So, to overcome this problem we need to divide the data into smaller sizes and give it to our computer one by one and update the weights of the neural networks at the end of every step to fit it to the data given. There are some optimal options for MLP, including batch learning, learning rate and early stopping strategy.

- epoch: one forward pass and one backward pass of all the training examples.
- weights initialization: initialization of the all weights and bias
- batch size: the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.
- learning rate: gradient descent step size
- early stopping: If the early shopping rule is applied,we mark minimal validation loss. Once validation loss go upward, we stop training.

And the following gure 5 shows how the leaning rate decreases during learning.

```
epochs = 700  # Total number of training epochs
batch_size = 200  # Training batch size
learning_rate = 0.0001 #
do_early_stopping = True # if apply early stopping learning
```

Learning quality:
For monitoring learning quality, define batch average cross-entropy error (BACRE), the gradient norm (G) and the quotient of gradient norm and weights norm(GW). After each batch, record these three indicator.

$$BACRE = \frac{1}{B}\sum_{k=1}^{B} Target_k \cdot log(OUT_k \tag{7}$$

$$GW = \frac{||W(n+1) - W(n)||}{||W(n)||} \tag{8}$$

$$G = \frac{1}{\sigma}||W(n+1) - W(n)|| \tag{9}$$

$$G2 = \frac{1}{\sigma\sqrt{D}}||W(n+1) - W(n)|| \tag{10}$$

where σ is the learning rate and D is the dimension of weights. Notice that Here in the code, loss function is ACRE.

```
grad_and_var = tf.train.AdamOptimizer(learning_rate=learning_rate,
                        name='Adam-op').compute_gradients(loss)
gradients = [x[0] for x in grad_and_var]
grad_norm = tf.global_norm(gradients).eval()
weights = [x[1] for x in grad_and_var]
w_norm = tf.global_norm(weights).eval()
G= grad_norm/learning_rate
G2 = grad_norm/(learning_rate*d)
GW_norm = grad_norm / w_norm
```

After each Epoch(m) , m= 1,2, ...compute the performance indicator percentage of correct classifications, which is defined as "accuracy" in the code, on the whole training set and the whole test set.

```
correct_prediction = tf.equal(tf.argmax(output, 1),
                        tf.argmax(y, 1), name='correct_pred')
accuracy = tf.reduce_mean(tf.cast(correct_prediction,
                        tf.float32), name='accuracy')
```

Also, select the best epoch Epoch(m*=best_so_far_epoch) after which the learning should be stopped to avoid overfit.

```
if do_early_stopping == True :
  if acc_test > best_so_far_acc:
                best_so_far_acc = acc_test
                best_so_far_epoch = epoch
```

After the last epoch Epoch(m*) compute and interpret the 3x3 confusion matrix of MLP(m*)

```
prediction = tf.argmax(output, axis=1, name='prediction')
true_label = tf.argmax(y, axis=1,name='true_label')
confusion_matrix = tf.confusion_matrix(labels=true_label,
                predictions= prediction, num_classes=n_classes)
```

For comparing the time of the learning with different options, we use function datetime to record the time length of training. start_time is put on the beginning of the session.

```
time_elapsed = datetime.now() - start_time
print('Time elapsed (hh:mm:ss.ms) {}'.format(time_elapsed))
```

5 Performance analysis

For the size of hidden layer h90 and hL, setting epochs=700, learning rate = 0.0001, Batch size =200. Then for each epoch, there are 1,8000/200=90 batches. And for each epoch, we run the number of total batches = 90 times training to approximately go through all the data in the training set once.

The entire learning with h = h90 =97, take time???. Since there are so many data 700*90=6300 fr each indicator, the curve will be not that unclear in that size. For plot, we only plot the four indicators in the last batch of epoch. Figure 5,6,7,8, are the curves for BACRE,G,G2, GW with h90=97, respectively. We can see that all these four gures reduce very fast in the beginning and become stable around values 0.1, 3000, 12 and 0.1 respectively. Since here we use a constant learning rate, it cause the gradients oscillate in the later part of learning. There is a reasonable solution is that we set a decreasing learning rate, for example exponentially decay function.

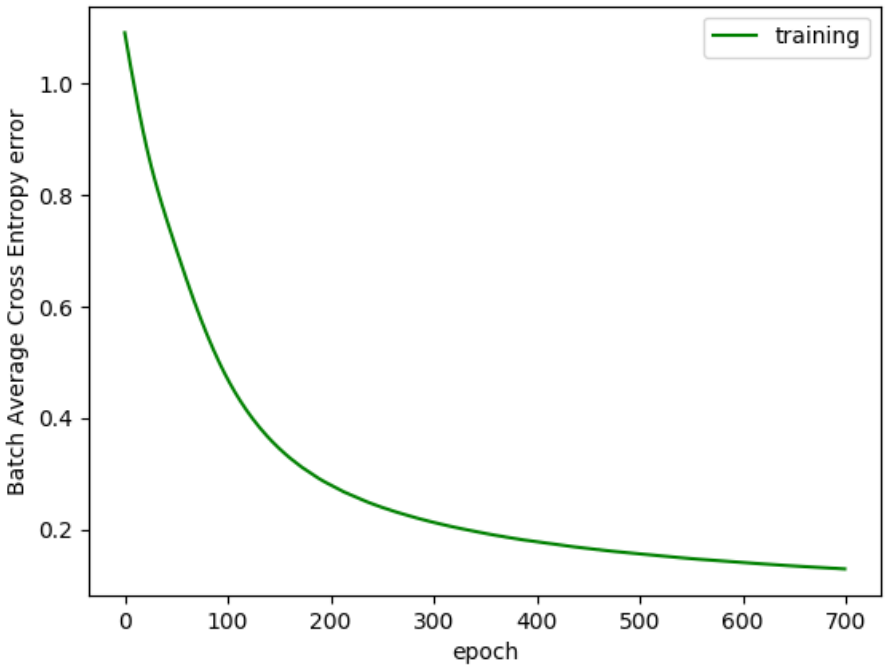

Figure 5: batch average cross-entropy error, with h90=97

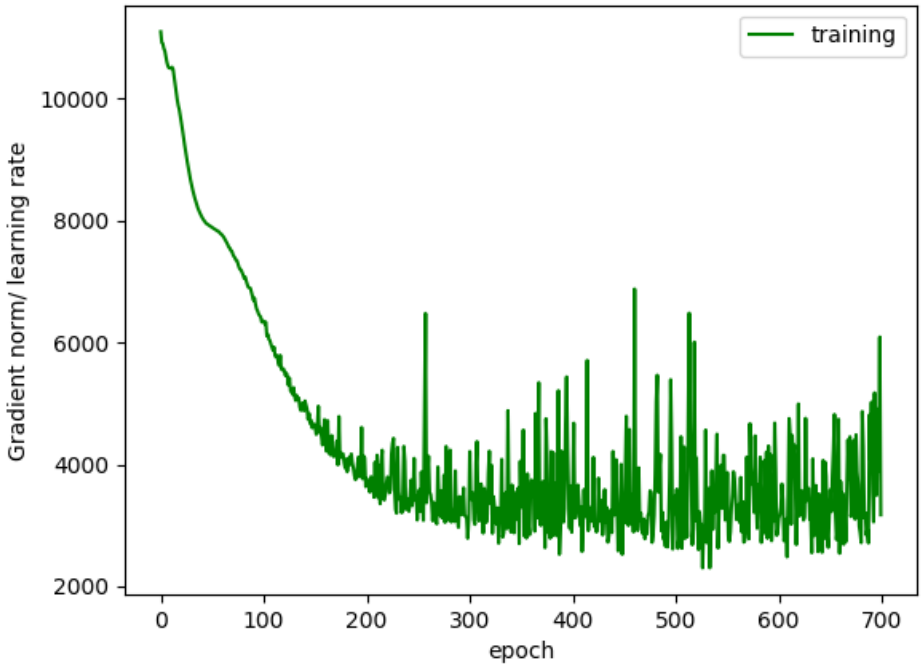

Figure 6: $||G||/learningrate$, with h90=97

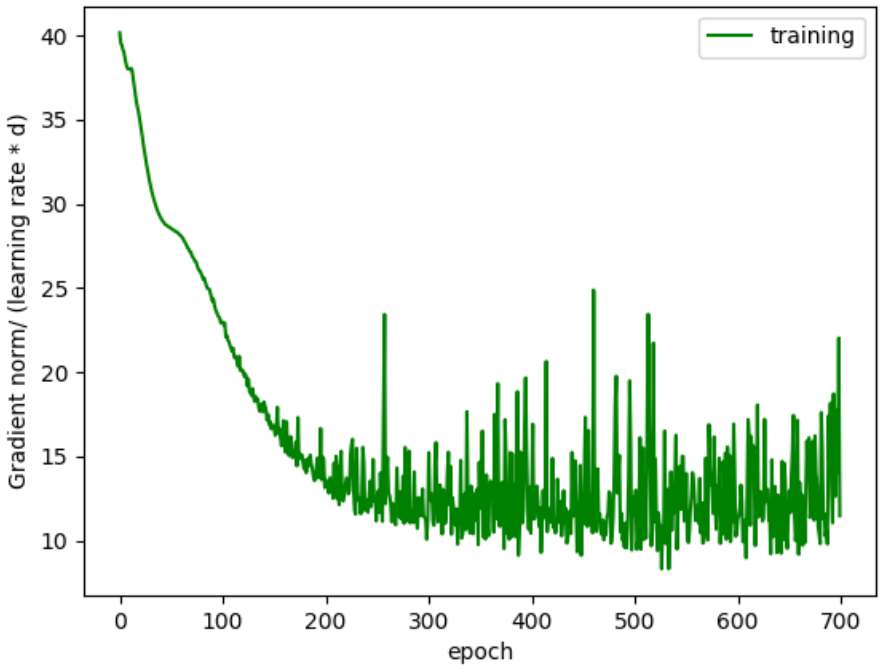

Figure 7: $||G||/(learningrate \times d)$, with h90=97

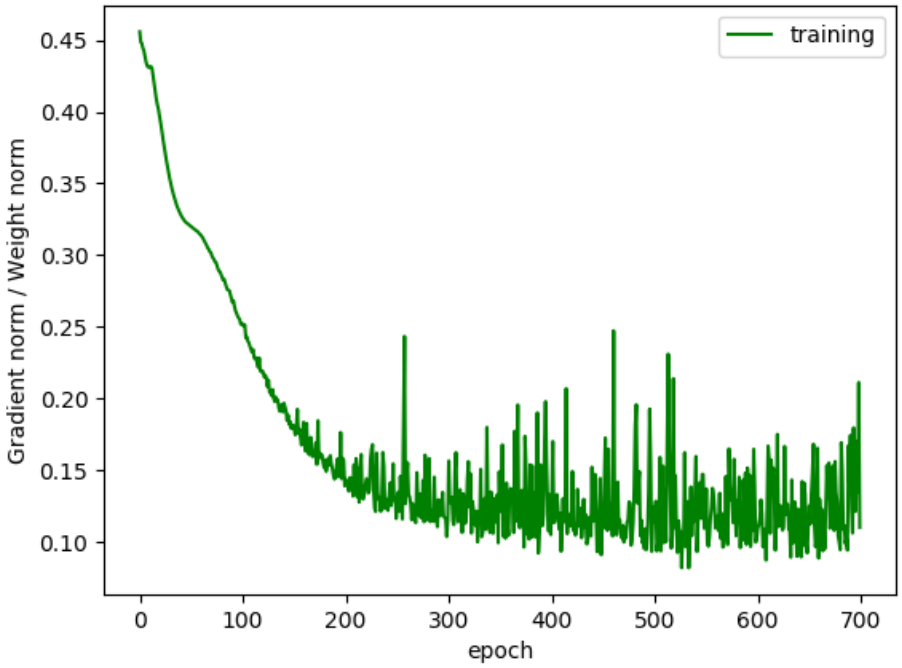

Figure 8: $||G||/||W||$, with h90=97

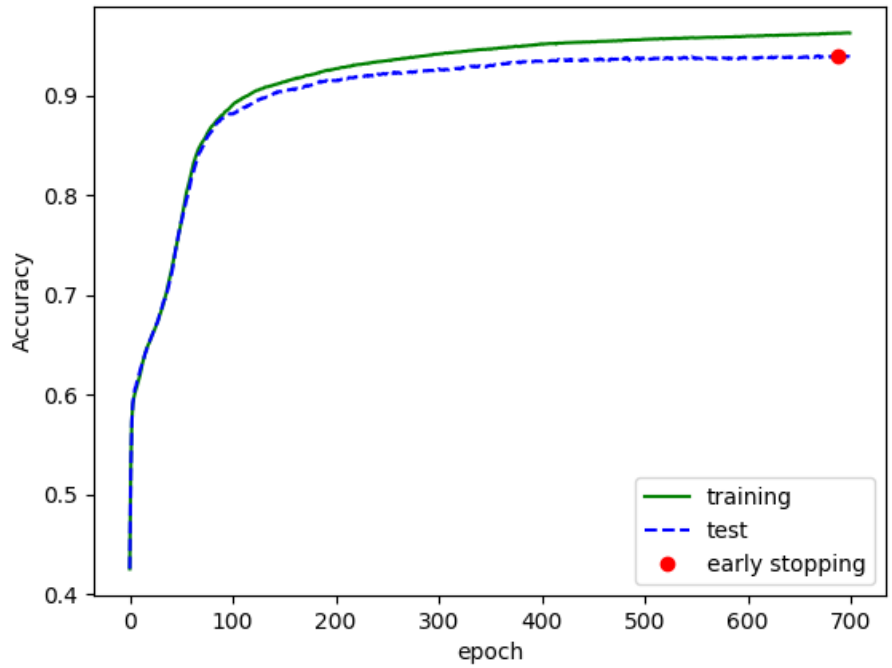

Figure 9: the percentage of correct classifications, with h90=97

Plot the the accuracy for training set and test set in Figure 9, that is the percentage of correct classifications. Early stopping learning rule give us the best epoch to stop, which is the epoch have the highest accuracy for the test set, m*=691. At this epoch, the accuracy of test set is 93.7%. Compute the 3x3 confusion matrix. The diagonal elements are close to 1, which means they have good prediction.
Confusion matrix of training set:

$$\begin{bmatrix} 0.9551 & 0.0265 & 0.0184 \\ 0.0132 & 0.9548 & 0.0320 \\ 0.0052 & 0.0264 & 0.9684 \end{bmatrix} \tag{11}$$

Confusion matrix of test set:

$$\begin{bmatrix} 0.9250 & 0.0410 & 0.0340 \\ 0.0240 & 0.9420 & 0.0340 \\ 0.0070 & 0.0380 & 0.9550 \end{bmatrix} \tag{12}$$

Run the same MLP with size of hidden layer h=hL=296. This learning takes time??? .
Figure 10,11,12,13 are the curves for BACRE,G,G2, GW with hL=296 respectively. We can see that, similarly as before, all these three gures reduce very fast in the beginning and become stable around values 0.15,2000,5 and 0.05. All of them are lower than the case with h90. Obviously, both weighted gradient norm are more stable.

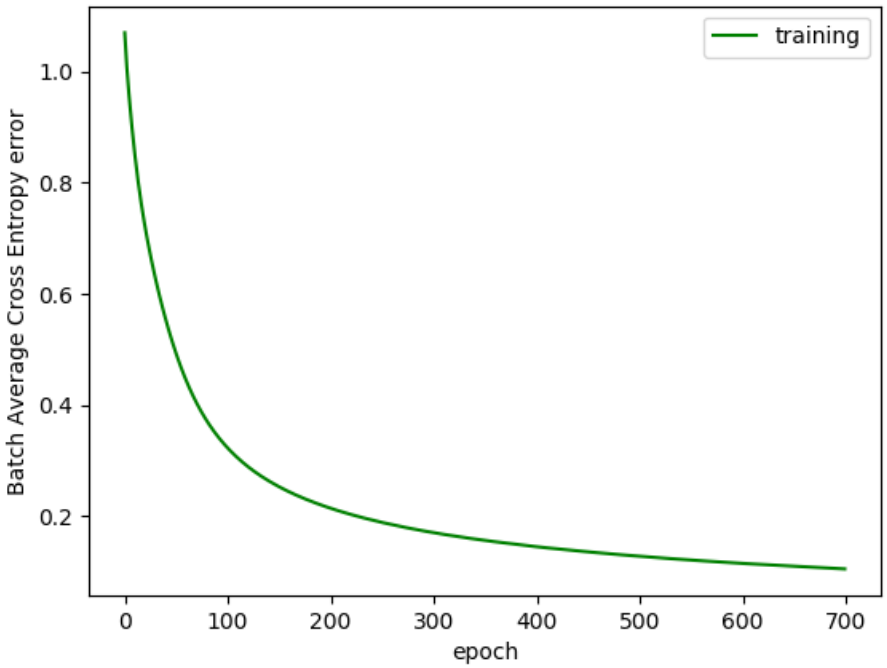

Figure 10: batch average cross-entropy error, with hL=296

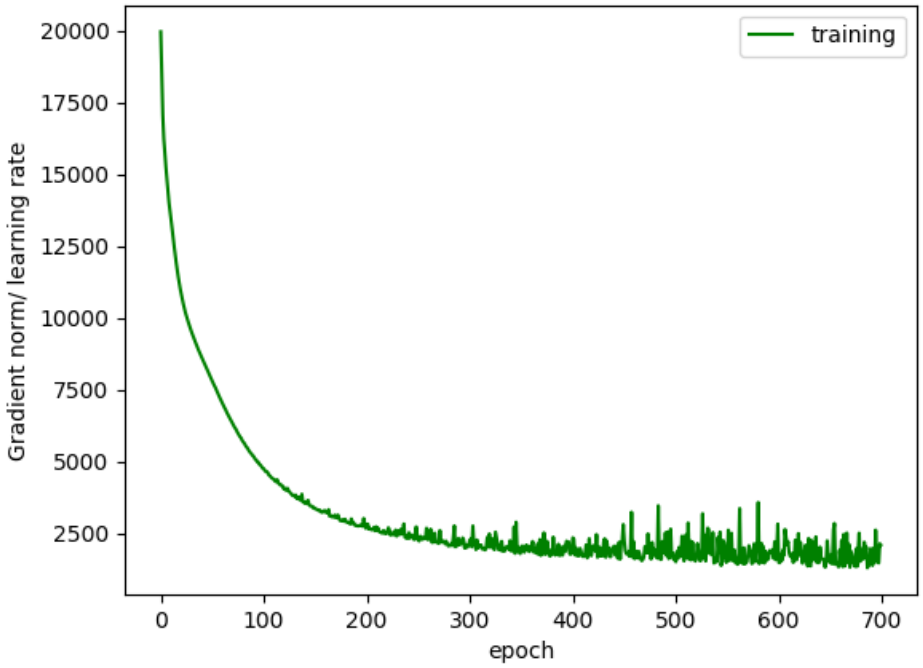

Figure 11: $||G||/learningrate$, with hL=296

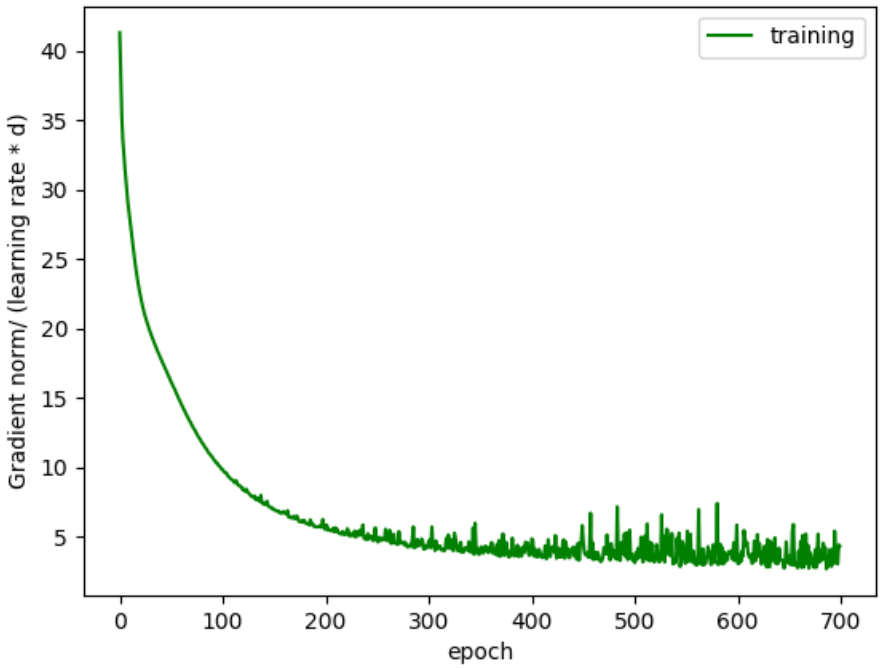

Figure 12: $||G||/(learningrate \times d)$, with hL=296

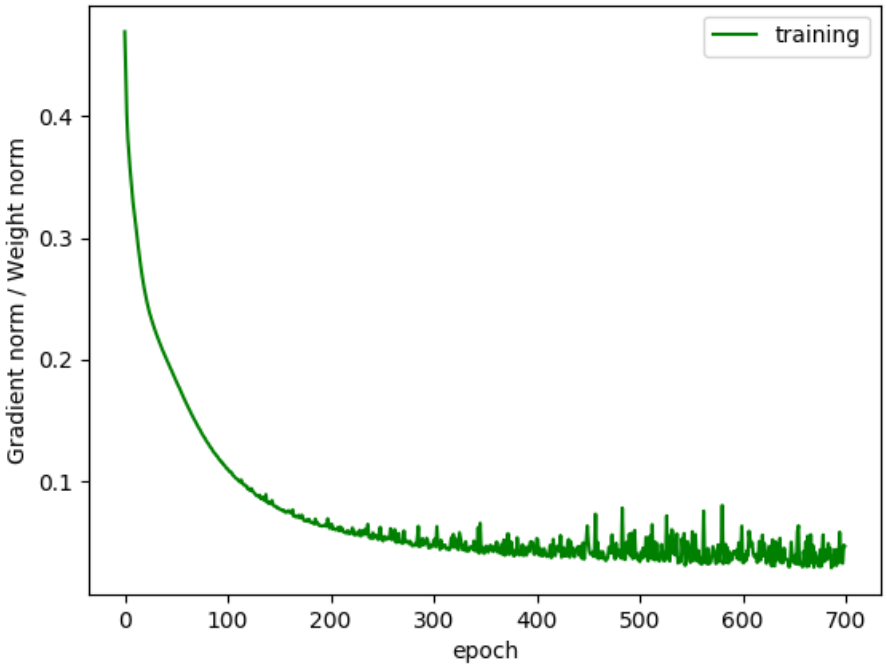

Figure 13: $||G||/||W||$, with hL=296

Plot the curve for the percentage of correct classifications for training set and test set in Figure 14.

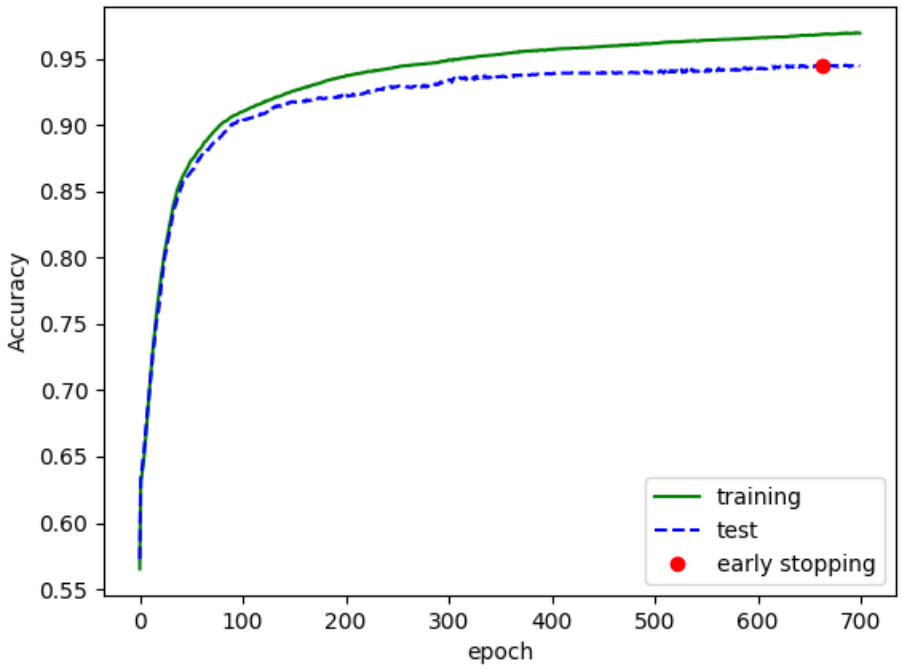

Figure 14: the percentage of correct classifications, with hL=296

The Early stopping learning rule also give us the best epoch to stop, m*=663. The best rate accuracy for the test set is 94.5%, which is better than h90. From test set 3x3 confusion matrix, the diagonal elements are larger than 93%. In the wrong prediction examples, the largest one is 0.036, which is the third true label(angle boots), but the network give wrong prediction the second label(sneakers). The network are good in distinguish angle boots(3rd label) with sandals(1st label).
Confusion matrix of training set:

$$\begin{bmatrix} 0.9690 & 0.0205 & 0.0105 \\ 0.0122 & 0.9660 & 0.0218 \\ 0.0040 & 0.0248 & 0.9712 \end{bmatrix} \tag{13}$$

Confusion matrix of test set:

$$\begin{bmatrix} 0.9310 & 0.038 & 0.0310 \\ 0.0220 & 0.9470 & 0.0310 \\ 0.0060 & 0.0360 & 0.9580 \end{bmatrix} \tag{14}$$

6 Impact of various learning options

- Weight Initialization:

In the Weight Initialization, we do not know what the final value of every weight should be in the trained network, but with proper data normalization it is reasonable to assume that approximately half of the weights will be positive and half of them will be negative. A reasonable-sounding idea then might be to set all the initial weights to zero, which we expect to be the best guess in expectation. This turns out to be a mistake, because if every neuron in the network computes the same output, then they will also all compute the same gradients during back propagation and undergo the exact same parameter updates. In other words, there is no source of asymmetry between neurons if their weights are initialized to be the same. Therefore, we still want the weights to be very close to zero, but as we have argued above, not identically zero. In our project, we try the 4 kind of initialization of weights with tf.initializers. Different weight initialization have relatively little impact on the final performance in practice.

- Batch Size:

Larger batch size means more "accurate" gradients, in the sense of them being closer to the true gradient one would get from whole-batch (processing the entire training set at every step) training. This leads to lower variance in the gradients. But it would be slow due to the number of iteration need per example. On the other side, a small batch size is often more efficient computationally. There would be huge noise in the update since the gradient update direction is only reliant on small part of data point. It currently seems that with deep networks it's preferable to take many small steps than to take fewer larger ones. And it bring the stochastic factor into training, which may lead to lower loss. Some online experience is about (50 1000).

For exploring the effort of batch size, we choose h90 = 97, hidden layer size with batch size = 2000 instead of 200.
In the Figure 15,16,17. ACRE, G2 and GW stabilize around and ,which are both bigger than the same case with small batch size 200(Figure 5,7,8). G2 and GW curve are more smooth. It's more accurate since the variance drops when you average something. and more stable gradients.

Also, it is slow due to the number of iteration need per example. It takes longer than the case of batch size =200, with the same epochs=200. It take 44 minutes and 13 seconds to finish learning, close to the previous case 48 minutes. After 700 epochs, training accuracy rate is 89.3% and test accuracy rate is 89.0%. The learning performance is not that good with the same time. But the accuracy rate is still increasing. Hence, big batch size is more accurate but slow.

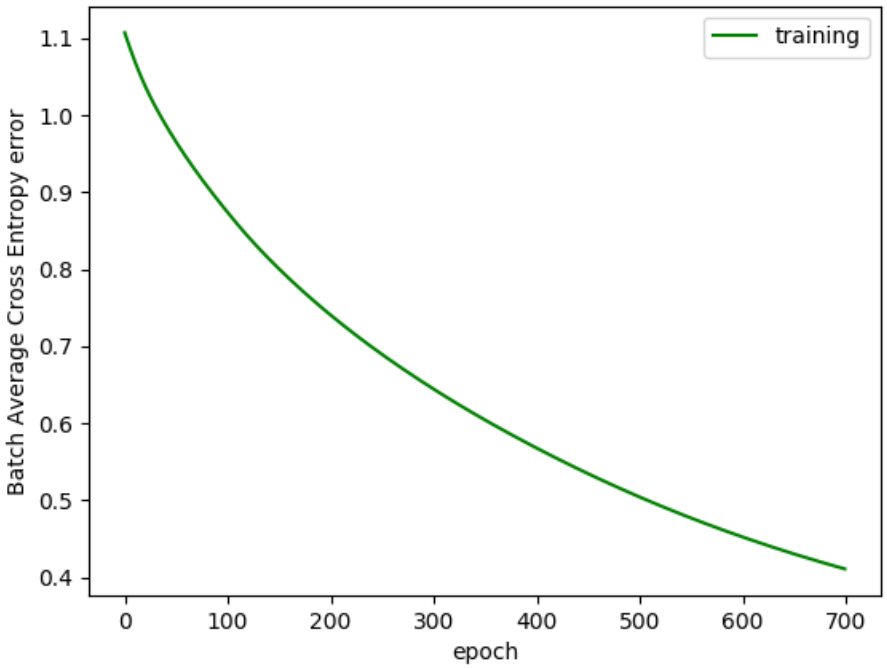

Figure 15: batch average cross-entropy error, with B=2000

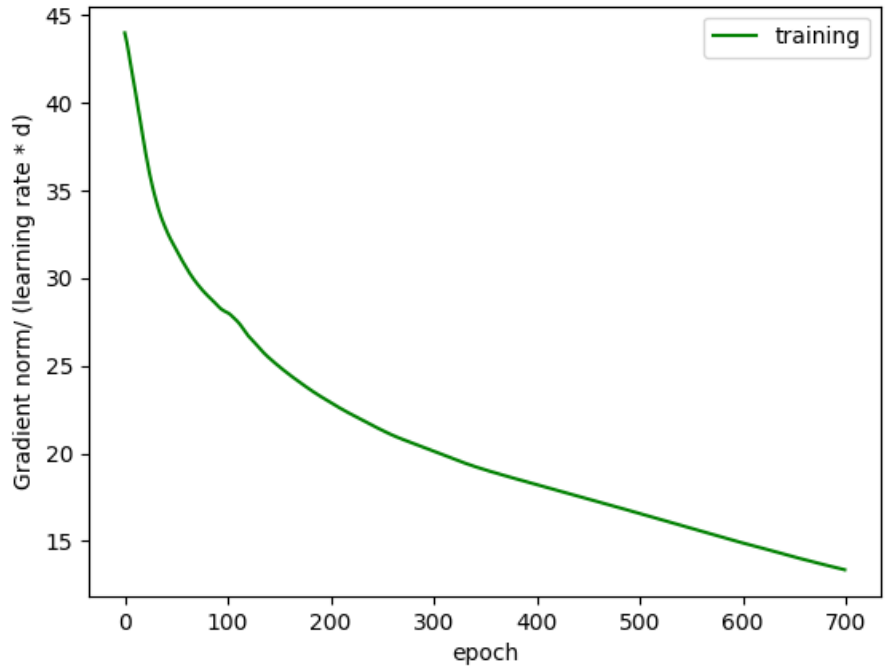

Figure 16: $||G||/(learningrate \times d)$, with B=2000

b

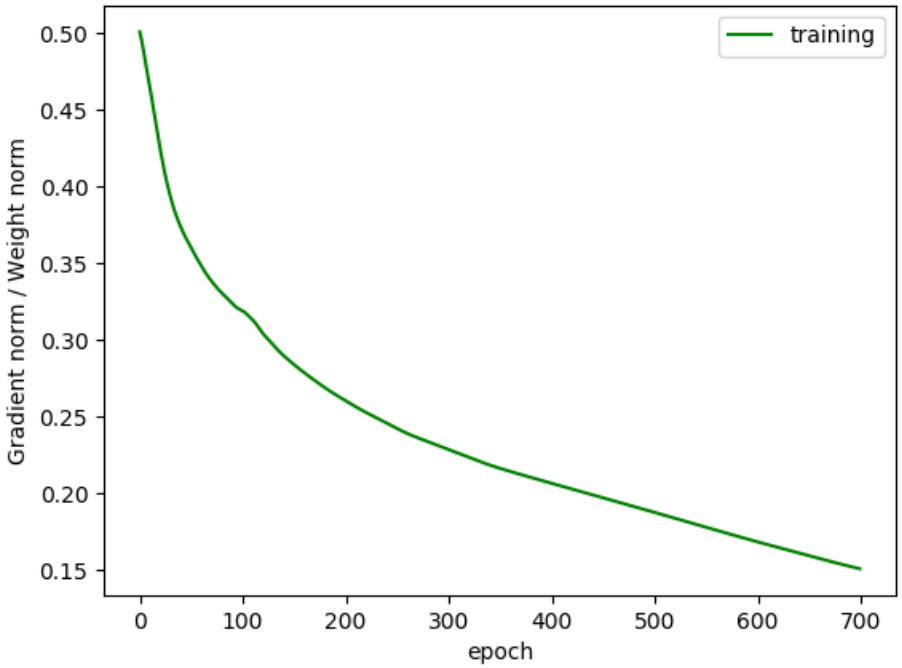

Figure 17: $||G||/||W||$, with B=2000

b

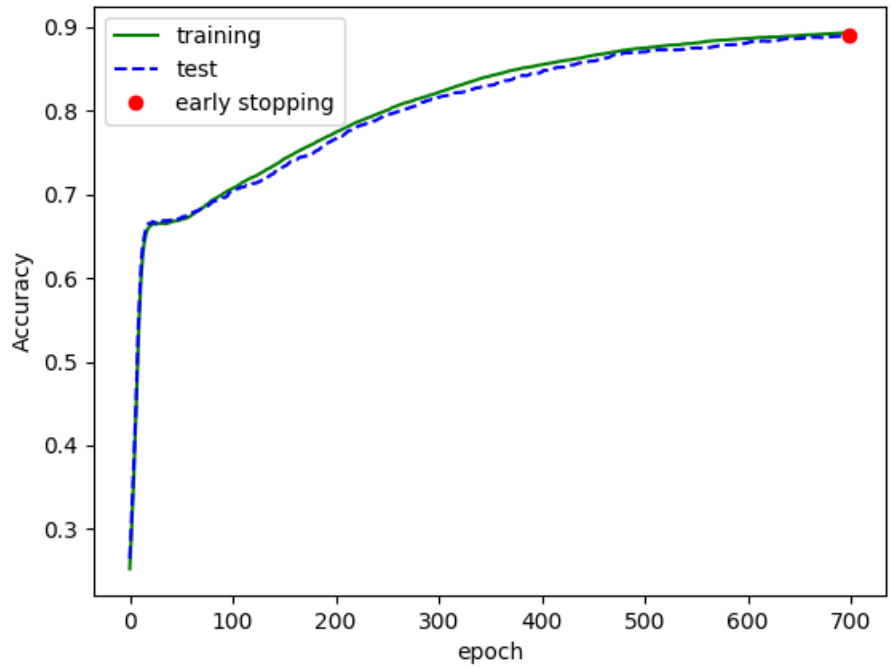

Figure 18: the percentage of correct classifications, with B=2000

- Learning rate / gradient descent step size

Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient. The lower the value, the slower we travel along the downward slope. While this might be a good idea (using a low learning rate) in terms of making sure that we do not miss any local minimal, it could also mean that well be taking a long time to converge, especially if we get stuck on a plateau region. Conversely, if you specify a learning rate that is too large, the next point will perpetually bounce haphazardly across the bottom of the well like a quantum mechanics experiment gone horribly wrong.

If setting learning rate =0.001 rather than 0.0001, with hidden layer size h90 = 97. ACRE, G2 and GW stabilize around becomes stable much faster in gure 19,20,21. And the Accuracy on the training set and test set increase to 99.6% and 95.8% in figure 22. It is very impressive!!! The early stopping learning rule tell that the best accuracy after 418 epochs , which takes only about 28 minutes. Hence, in the previous network, we take too small learning rate, the network learn too slow.

Combine with the implement of learning rate in homework 1, we get the experience that RELU usually work with small learning rate. We reset the RELU function as response function in the hidden layer, learning rate =0.000001 give us similar performance. If we choose sigmoid function, the learning rate would be relatively big (0.001 in homework2).

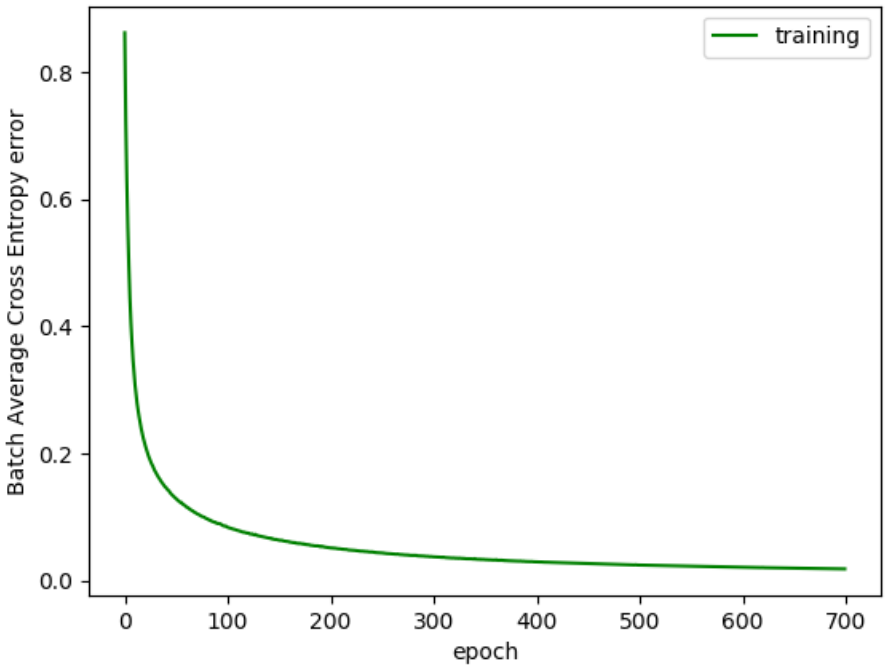

Figure 19: average cross-entropy error, with learning rate =0.001

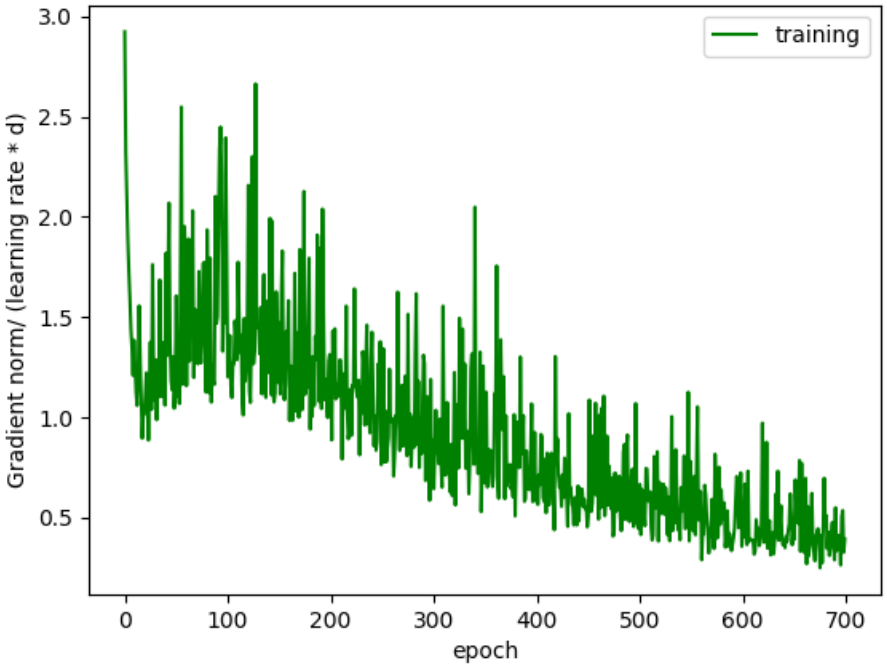

Figure 20: $||G||/(learningrate \times d)$, with learning rate=0.001

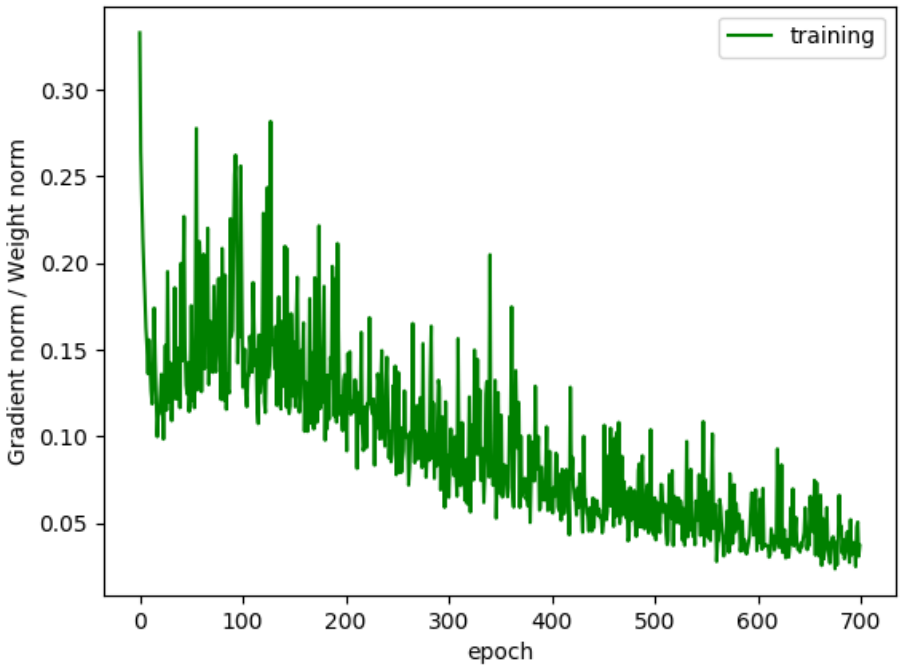

Figure 21: $||G||/||W||$, with learning rate=0.001

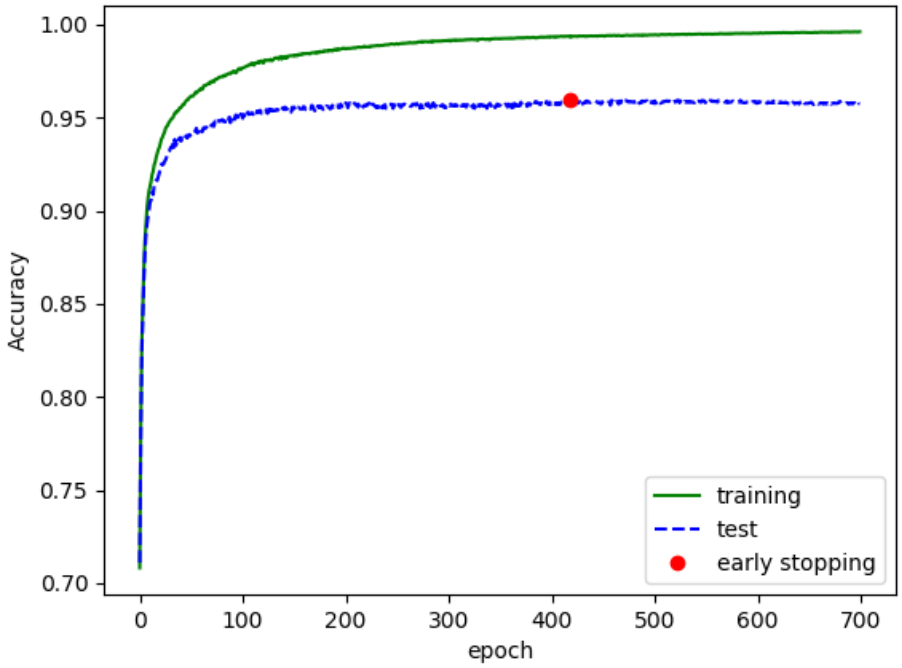

Figure 22: the percentage of correct classifications, with learning rate=0.001

- Dimension of the hidden layer

Compare the learning with h90= 97 and hL=296, we can see that ACRE, G2 and GW have smaller stabilized values with hL. And the accuracy on the test set and training set with h90 are smaller than those with hL. So training with hL=296 seems to work better.

7 Analysis of the hidden layer behaviour after completion of automatic learning

According to the above performance evaluations, we have the best of the two values h90 and hL, h*=hL=296. And the best epochs m*=418, mark the corresponding MLP*= MLP(m*). Base on this network MLP*, do the analysis of the hidden layer behaviour.

The hidden layer activity vectors H(1) ...H(k)... H(N) is generated by all the training inputs X(k). N= 1,8000, is the size of input. So H is in the size of N*h=1,8000*296. From the result of the PCA analysis on the hidden layer activity vectors. 252 of components could explain 99% of the sum of the eigenvalues, which is close to the total size h=296, which mean most of hidden neurons play an important role in the learning.

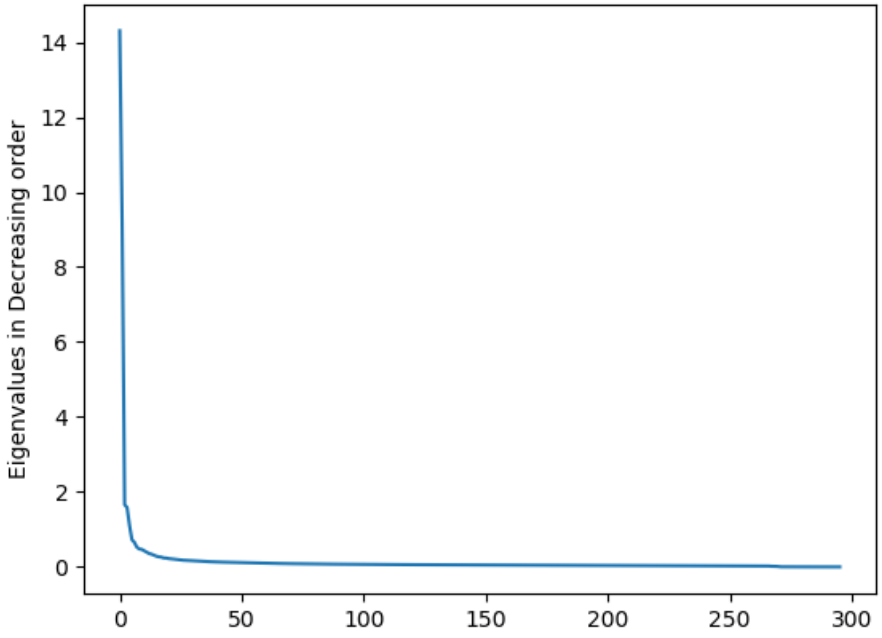

Figure 23: Eigenvalues in decreasing order

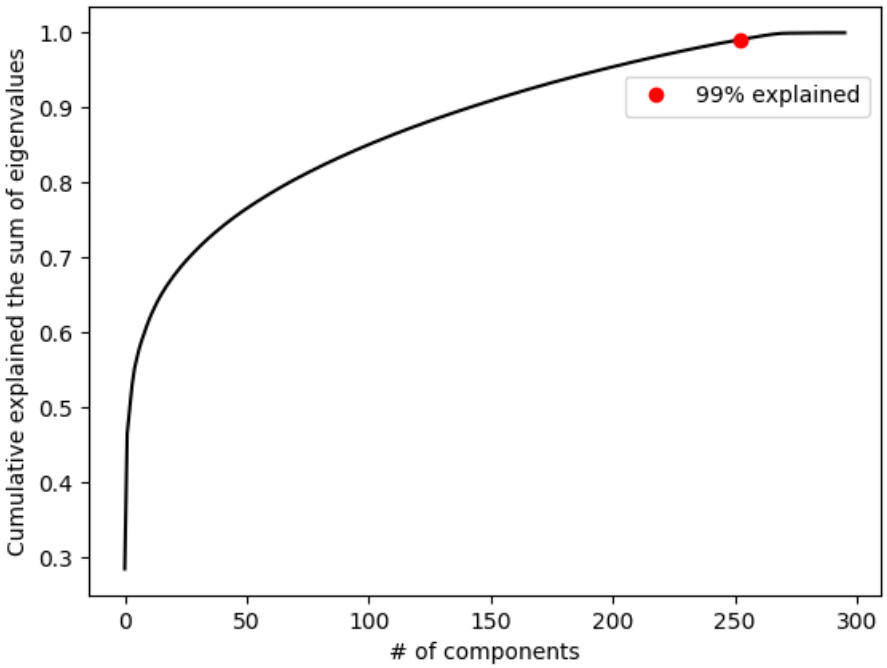

Figure 24: Cumulative Explained the sum of eigenvalues

Define the average hidden neurons activity profiles $PROF_j$, j=1,2,3 is the average of the H(k) over all cases case(k) belonging to class j. Because we apply sigmoid function as the response function in the hidden layer, all value of hidden neurons range between 0 and 1.

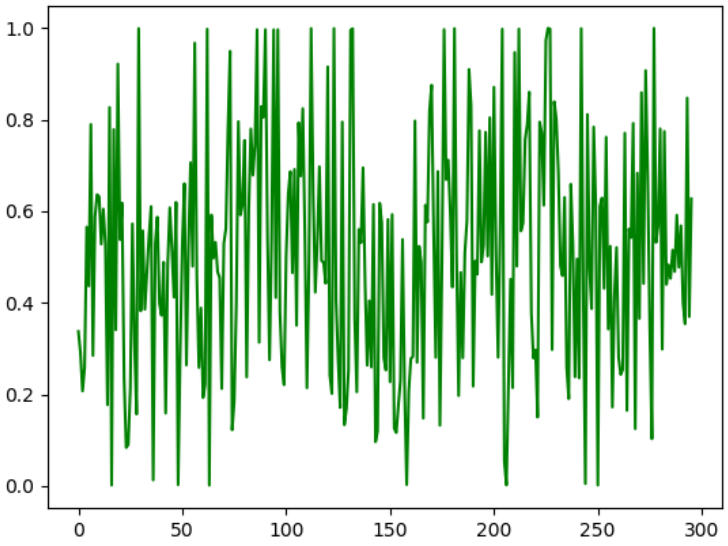

Figure 25: PROF1

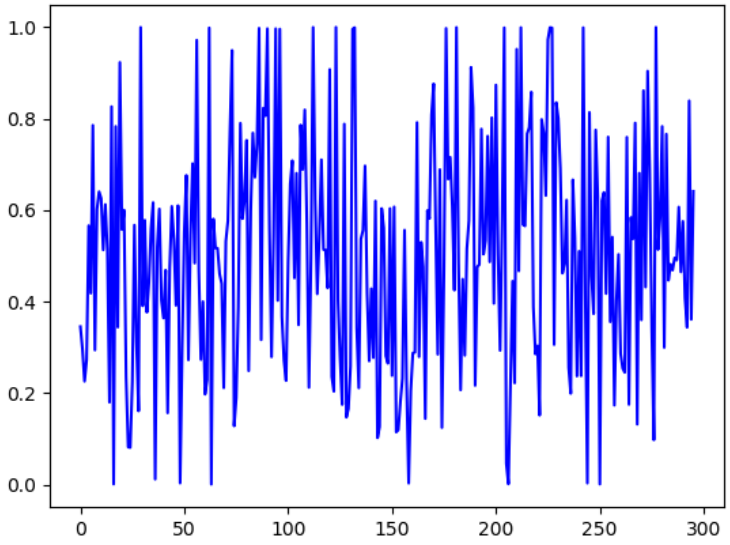

Figure 26: PROF2

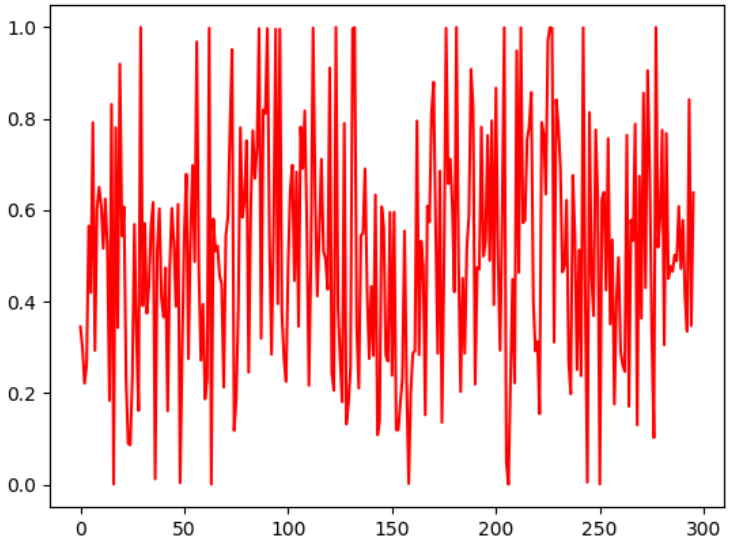

Figure 27: PROF3

In order to get the hidden neurons which achieve best DIFFERENTIATION between a pair of classes, we calculate the absolute difference between the pair of classes, then use argmax function to the hidden neurons ,which give the largest absolute differentiation.

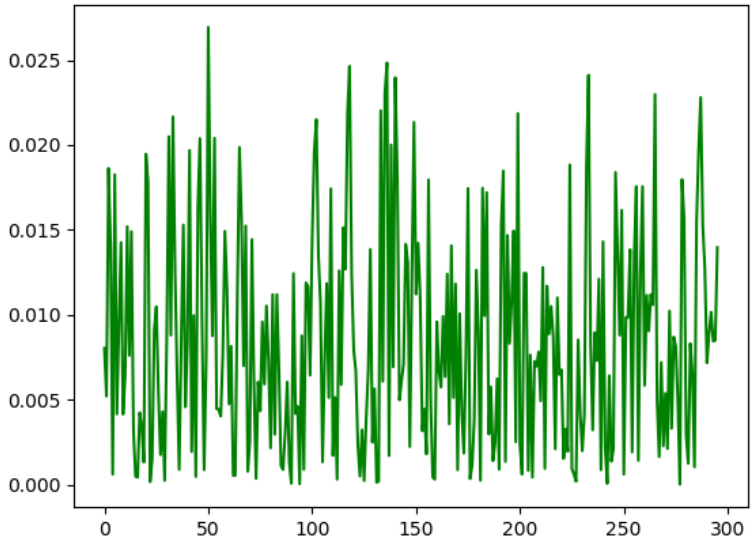

Figure 28: $|PROF1 - PROF2|$

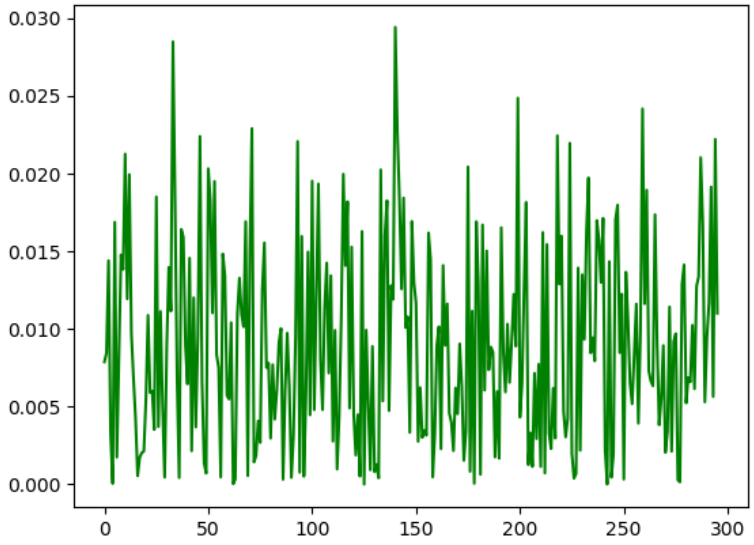

Figure 29: $|PROF1 - PROF3|$

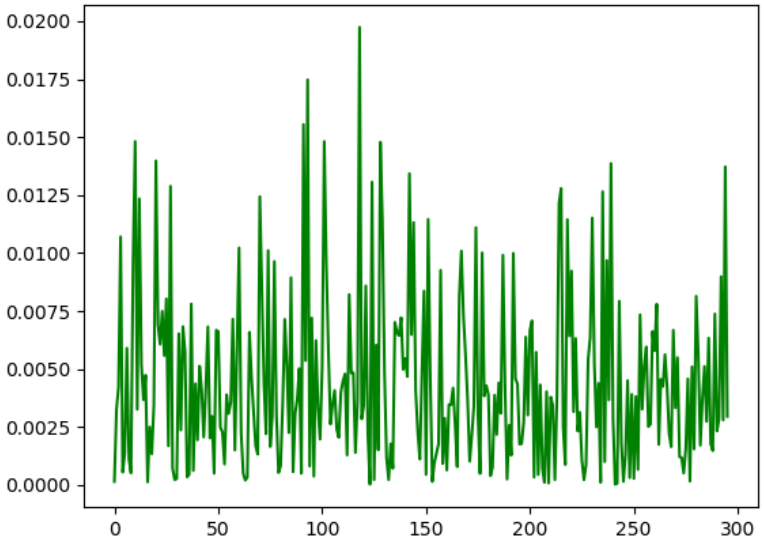

Figure 30: $|PROF2 - PROF3|$

Hence, we have result as following:

```
the hidden neurons which achieve best DIFFERENTIATION between class C1 versus C2: 50
the hidden neurons which achieve best DIFFERENTIATION between class C1 versus C3: 140
the hidden neurons which achieve best DIFFERENTIATION between class C2 versus C3: 118
```

Figure 31: the hidden neurons which achieve best DIFFERENTIATION