HW3 - Huihui Yang for data mining

1. Description of the data and the associated classification task
The data set is all about Statlog (Shuttle), and the dataset contains 9 attributes all of which are numerical. There are 5 classes, approximately 80% of the data belongs to class .
In summary,

- The dataset has 5 classes, which can be classified: C1, C2, C3, C4, C5.temp
- Total cases: there are 14500 in total: in details:

C1:	11478 cases, about 79.15%
C2:	16 cases, about 0.11%
C3:	41 cases, about 0.28%
C4:	2156 cases, about 14.86%
C5.temp:	809 cases, about 5.57%

- Further, there are 8 descriptors for each case as :

1. Rad Flow
2. Fpv Close
3. Fav Open
4. High
5. Bypass
6. Bpv Close
7. Bpv Open
8. Unknown

As we can see from the above information, the overall dataset is not that perfect for our analysis, it's better to do some change before the analysis. In details, there are very few cases in classes C2 and C3, it's better to neglect these 2 classes for the next analysis; the cases in C4 is about 15%, more or less we can use it; for C5.temp, the cases is just 5.57% of the whole dataset, it's better to inflate this class, I decide to inflate C5.temp 3 times, then there will be 2427 cases in C5.

After change the detailed information of the new data set is as followed:
New total cases: there are 3 classes, and 16061 cases in total.

C1:	11478 cases, about 71.47%
C4:	2156 cases, about 14.32%
C5:	2429 cases, about 15.11%

2. Select my train set, test set and the MLP architecture.
And I choose 13000 cases around 81.4% to be the training set, 3061 cases around 18.6% of the total dataset to be the test set, and.
In the training set, 13061 cases:

C1:	9300 cases, about 71.47%
C4:	1850 cases, about 14.32%
C5:	1850 cases, about 15.11%

In the training set, 3061 cases:

C1: 2161 cases, about 71.47%
C4: 445 cases, about 14.32%
C5: 455 cases, about 15.11%

For our new dataset still has 8 descriptors in the, thus p_0=8, and 3 classes, thus p_2=3,
Now let me explain how the input vector transform the output:
If I set the input node as x_i, hidden layer node as y_j, and output node as z_k, then all the input nodes will transform to the hidden nodes through the first weight and bias, as:

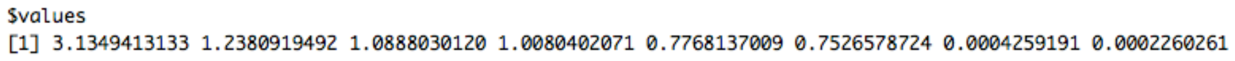
$$y_j=f(\sum_j \omega_{ij}x_i + b_i)$$

In details, as our input nodes are 13000 and we have 8 descriptions for the data, and when the hidden layer h=7, thus we will get the first weigh as 8*7 and bias as 1*8
Then we can get all the hidden layer nodes, after that the hidden layer nodes will transform to the output nodes through the second weigh and bias, as:

$$z_k=f(\sum_k \omega_{jk}x_i + b_j)$$

we will get the first weigh as 7*3 and bias as 1*3
By using this MLP, we can train our input to a 3 class output and get the calculated value.

3. Estimate the tentative sizes h for the hidden layer
1). Estimate h=s
Since in this dataset, the descriptors are all continuous, in this part, we still have 8 continuous descriptors. And the total number of the cases is 16061.
By applying the PCA method to analyze the dataset, first computing the correlation of the dataset, then compute its eigenvalues and eigenvectors, which are listed bellowing:
The eigenvalues of $S_1 \geq S_2 \geq \ldots \geq S_8$ =



```
$values
[1] 3.1349413133 1.2380919492 1.0888030120 1.0080402071 0.7768137009 0.7526578724 0.0004259191 0.0002260261
```

$$RAT_j = \frac{S_1 + S_2 + \ldots + S_j}{S_1 + S_2 + \ldots + S_8} =$$

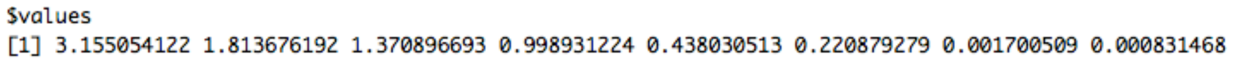
0.3919. 0.5466. 0.6827. 0.8087. 0.9058. 0.99992. 0.99997. 1.00000

To get $RAT_j \geq 0.95$, here the smallest number "s" of eigenvalues preserving 95% of the total sum of eigenvalues s=6 and $RAT_6 = 0.99992$
The intermediate plausible value s=6< h=S < p_0=8, thus we get h=S=7

2). Estimate h=S

<u>For C1 dataset,</u> similarly we apply the PCA analysis:

Eigenvalues are listed as

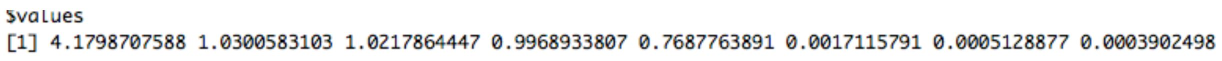


```
$values
[1] 3.155054122 1.813676192 1.370896693 0.998931224 0.438030513 0.220879279 0.001700509 0.000831468
```

$RAT_j =$ **0.3944 0.6211. 0.7924. 0.9173. 0.97207. 0.99968. 0.99990 1.00000**

Thus the chosen S1=5;

<u>For C4 dataset</u>, we apply the PCA analysis:
Eigenvalues are listed as:

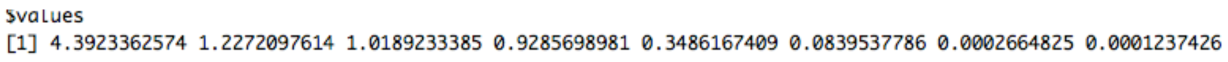


```
$values
[1] 4.1798707588 1.0300583103 1.0217864447 0.9968933807 0.7687763891 0.0017115791 0.0005128877 0.0003902498
```

$RAT_j =$ **0.5225. 0.6512. 0.7790. 0.9036. 0.9997. 0.99989. 0.99995. 1.00000**

Thus the chosen S4=5

<u>For C5 dataset</u>, we apply the PCA analysis:
Eigenvalues are listed as:



```
$values
[1] 4.3923362574 1.2272097614 1.0189233385 0.9285698981 0.3486167409 0.0839537786 0.0002664825 0.0001237426
```

$RAT_j =$ **0.549. 0.7024. 0.8298. 0.9459. 0.98946. 0.99995. 0.99998. 1.00000**

Thus the chosen S5=5
The defined h="S"=S1+S4+S5=5+5+5=15

3). Estimate h="SL"=2S=30

4. Implement learning by gradient descent
Here I use R to do the analysis, especially use the package of "Keras"
The loss function defined by mean squared error MSE can be written as:

$$MS_E = \frac{1}{N}\sum_{n=1}^{N} ||\hat{Z}_n - Z_n||^2,$$

$\hat{Z}_n$ is the computed output value, Z_n the true output value, N is the total number size for training.

For the training, I choose 13000 cases; and 3061 cases for the test set. To make sure that the frequency of each class C1,C2 and C3 is roughly equal to its frequency in the trading set, firstly, I use the random method to sample the whole dataset, which means that during the whole dataset, the cases have been randomly ordered, then I choose the first 13000 cases as the training set and the left 3061 cases as the test test, the cases in the test set is about 19% of the whole dataset.

In R, in order to perform the MLP learning, with the one can give more information and more flexible manipulation, I choose to use the “rstudio/keras” based on “Tensorflow‘, and in the library of “keras”, I can do more.
The first thing for the learning is to create the learning model. In our problem here, I need to establish a model with 3 layers: input layer with p0=8, hidden layer with h={7,15,30}, output layer with p2=3; at the same time, I use “kernel_initializer=“RandomNormal"” and “bias_initializer=“RandomNormal”option to initialize the weights and .
For the aim to get all the detailed information about the each iteration, there is no such option in the “Keras”, that makes the further study harder, so I have to **written a function to get all the detailed information of each iteration**, and then perform further learning. About the constant value, after several experiment I set to 0.2. epoch=50, batch size=300. Further, **I give two different ways, choosing different ways to set the learning rate to get the more accurate learning results.**
The first way is following the professor’s instructions as:
Select a batch size and a time dependent gain rate

$\epsilon_n = \frac{constant}{n}$,

where n is the iteration, then I can calculate the rMSE, Gn, and then give the corresponding plot.
The second way is to follow the method by using the “Keras” learning rate theory:
I need to “compile” the model: the chosen optimization way is stochastic gradient descent (SGD), with the “sgd”,

```
> optimizer_sgd(lr=0.1, decay=0.001/epochs, momentum=0.5)
```

I set the initial learning rate as “lr=0.1”, in order to get time dependent gain rate I set the decay as “decay=0.01/epochs”, for the computing method of keras is a little deferent, I could not set the time dependent grain rate as ϵ_n=constant/n. Here is the calculation of the time dependent gain rate:

$lr = \frac{lr_0}{kt}$, where lr and k are hyper parameters, t is the iteration number.

Further, $lr * = \frac{1}{1 + self.decay * self.iterations}$, this gain rate is good, much more complicated but results are good.
Just it will give some short backs that, I can not simply calculate the $G(n)$ by

$$||G_n|| = \frac{1}{\epsilon_n} * ||W_{(n+1)} - W_{(n)}||$$

I can not simply calculate Gn, thus in the back, even I have all the weights and still cannot easily calculate Gn.

Theoretically, we can choose a stop learning option such as in Pyhon Tensorflow “tf.gradients(stop_gradients=..) to **stop the learning**, in “keras”, here we can use “callback” function,

```
> callback_early_stopping(monitor = "loss",min_delta =0.001,mode = “min”)
```

Which can monitor the “loss”: quantity to be monitored

Mode: one of "auto", "min", "max". In `min` mode, training will stop when the quantity monitored has stopped decreasing; in `max` mode it will stop when the quantity monitored has stopped increasing; in `auto` mode, the direction is automatically inferred from the name of the monitored quantity.

min_delta: minimum change in the monitored quantity to qualify as an improvement, i.e. an absolute change of less than min_delta, will count as no improvement

Further I see the metric as **“accuracy” and loss=“mean_squared_error”** to Indicate the learning.

In order to get the detailed results for each epochs, I use the

> callback_csv_logger

Which can save the epoch, acc and loss with detailed information.

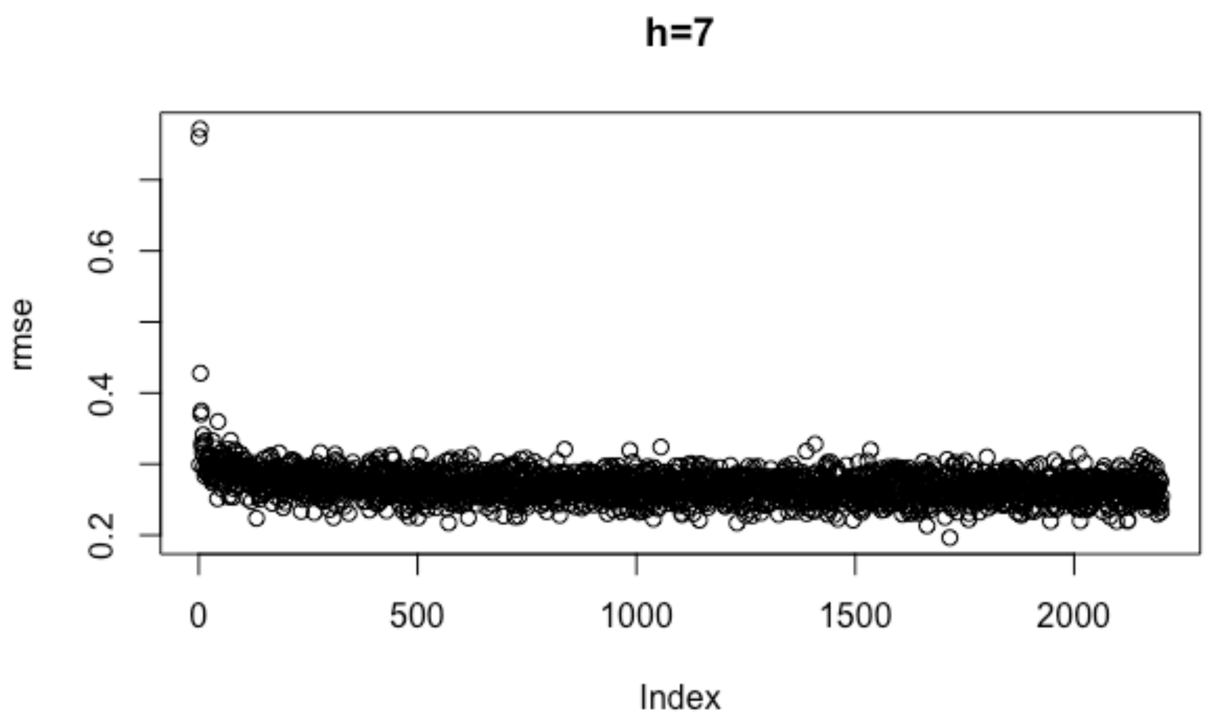
To get the weights and thresholds and each epochs, I use the code as:

> callback_model_checkpoint("filepath",monitor = “loss")

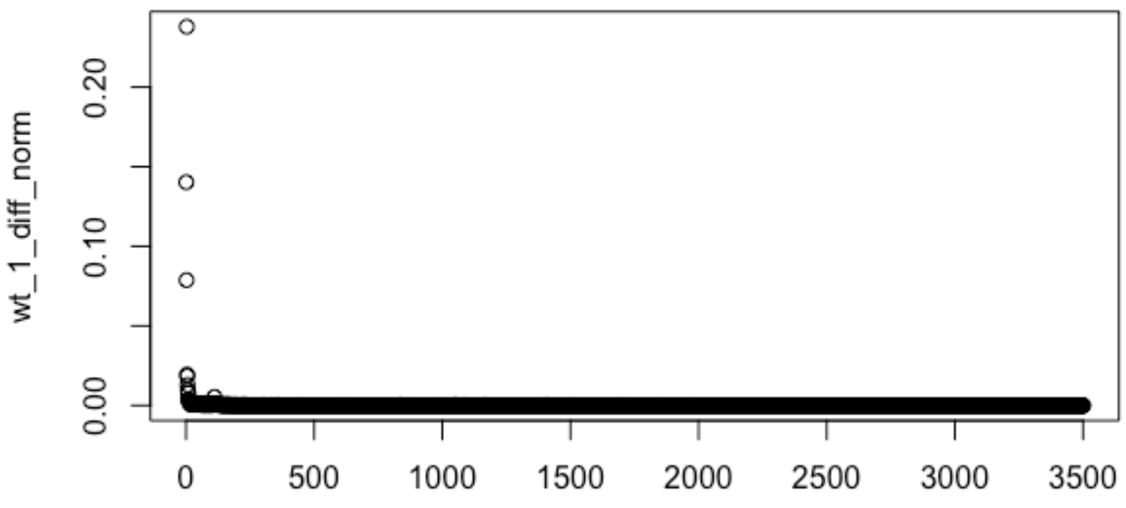
Which can save the model as each epochs, and we can further to load all the saved model during the learning and get all the weights and thresholds.

Just by using the function I written, this way I can get all the values about the MSE, weights and learning rate of each iteration. Bellowing is the graphs of the learning with epochs=50, batch size=300, h={7,15,30} separately, by using my function.

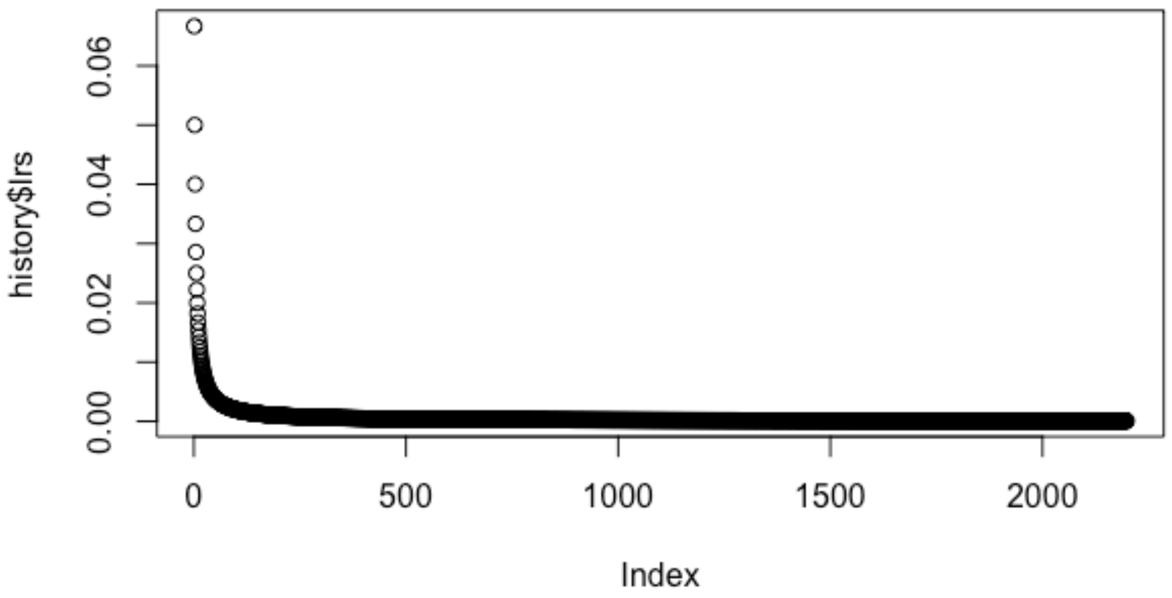
1).h=7, we can see that all the graphs are fine, just the learning rate decrease a little fast; the learning accuracy is good, about >0.8.



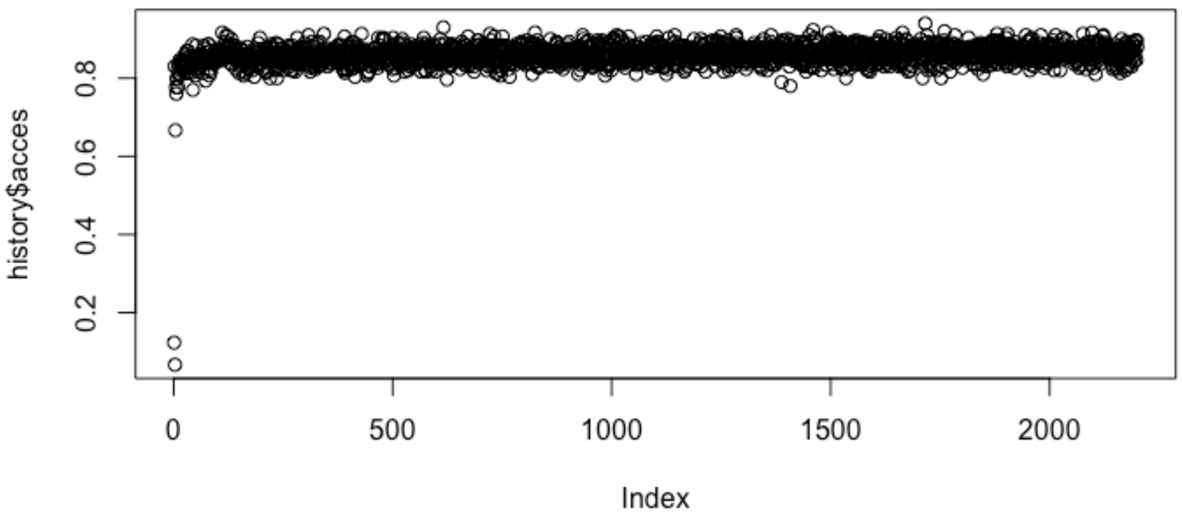
h=7


wt_1_diff_norm
0.00
0.10
0.20
0
500
1000
1500
2000
2500
3000
3500

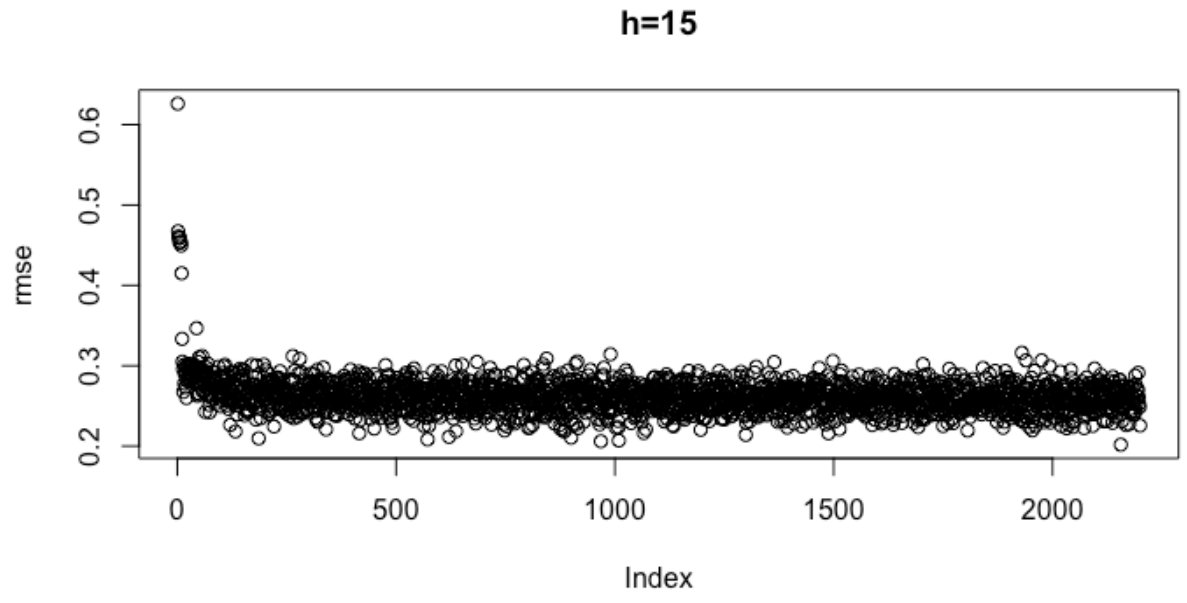
h=7

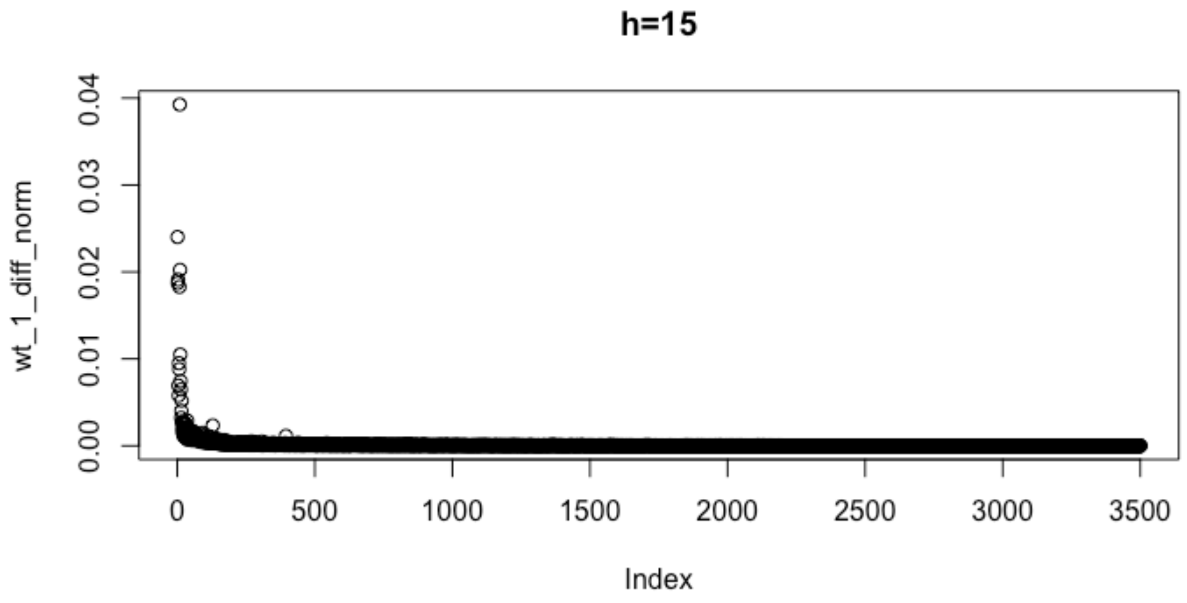

history$lrs
0.00
0.02
0.04
0.06
0
500
1000
1500
2000
Index

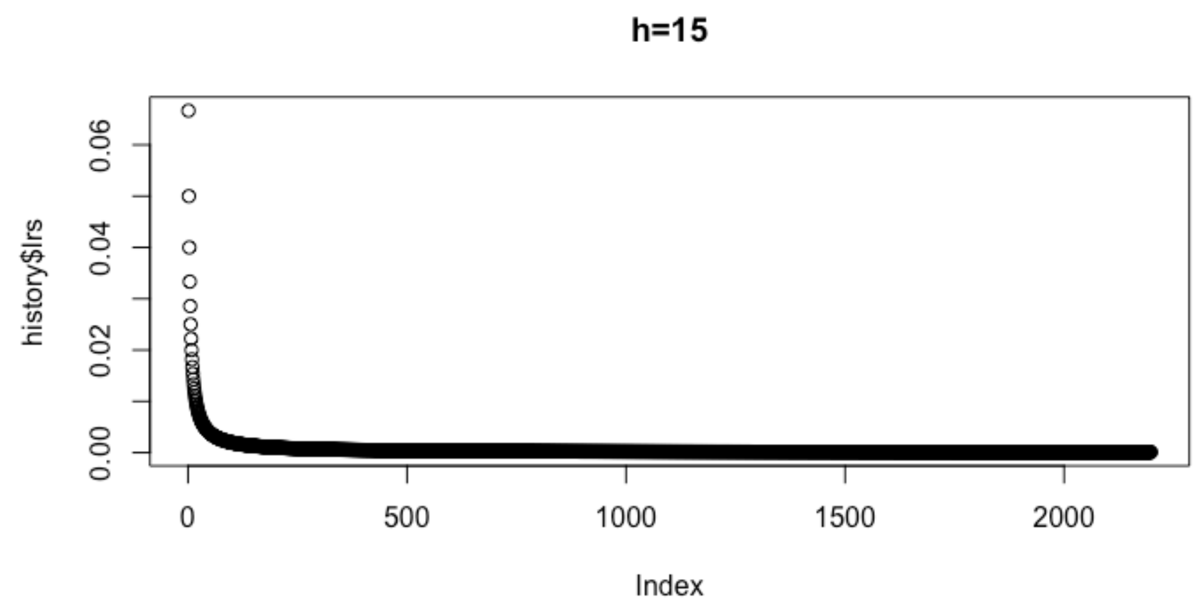
h=7

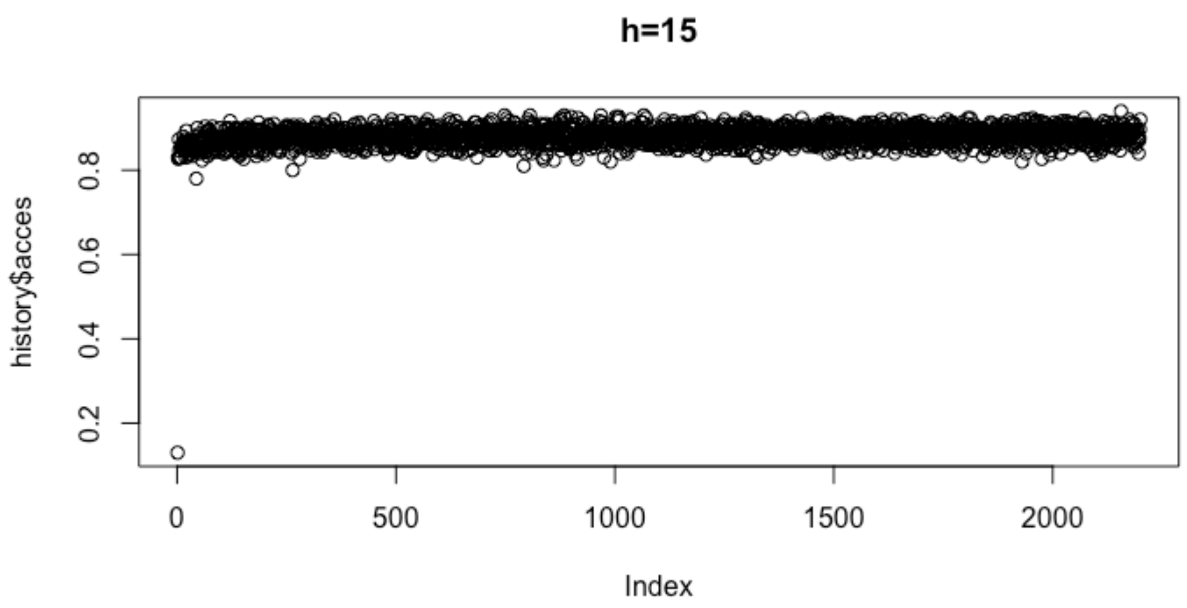

history$acces
0.2
0.4
0.6
0.8
0
500
1000
1500
2000
Index

2). h=15, we can see that all the graphs are fine, just the learning rate decrease a little fast; the learning accuracy is good, about >0.8 too.

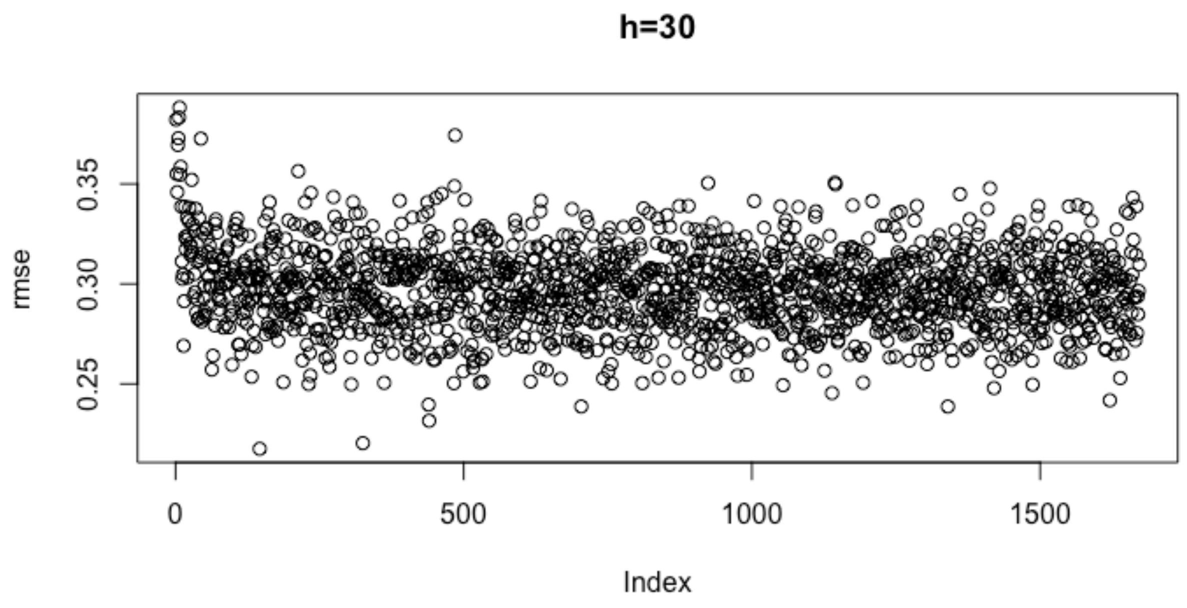


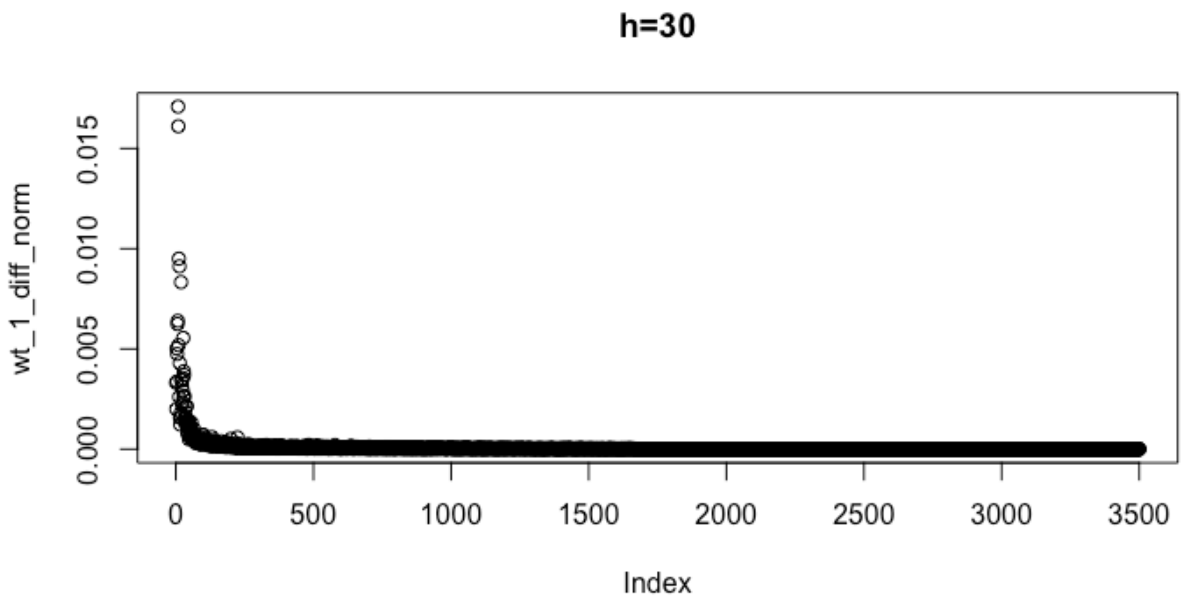


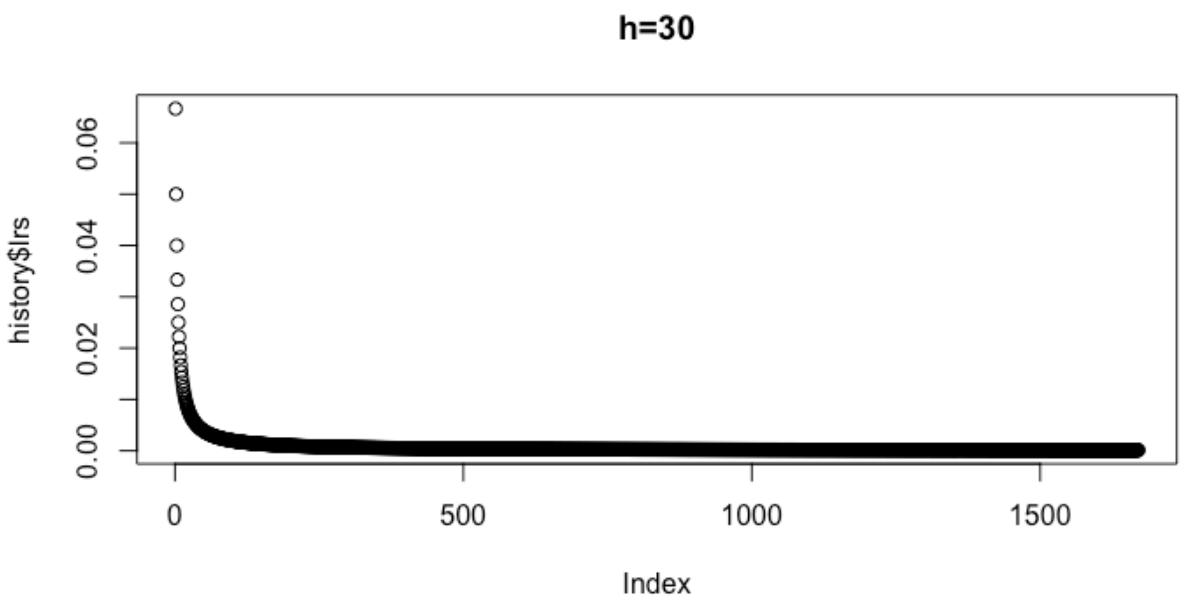

h=15
history$lrs
0.00
0.02
0.04
0.06
0
500
1000
1500
2000
Index

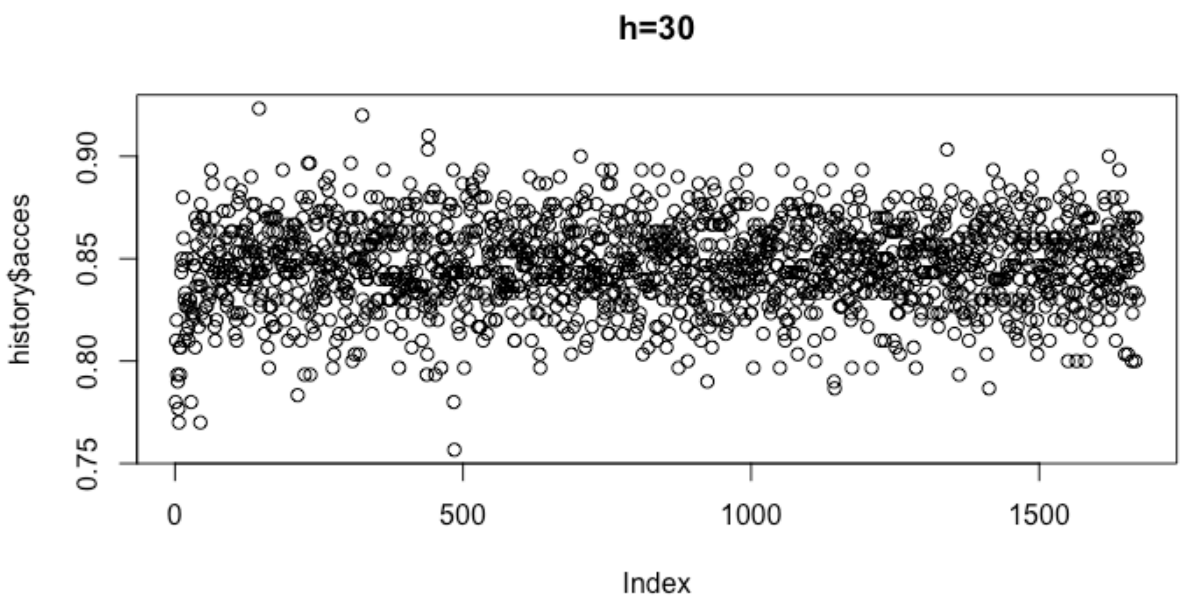

h=15
history$acces
0.2
0.4
0.6
0.8
0
500
1000
1500
2000
Index

3).h=30,we can see that all the graphs are fine, just the learning rate decrease a little fast; the learning accuracy is good, about >0.8 too.





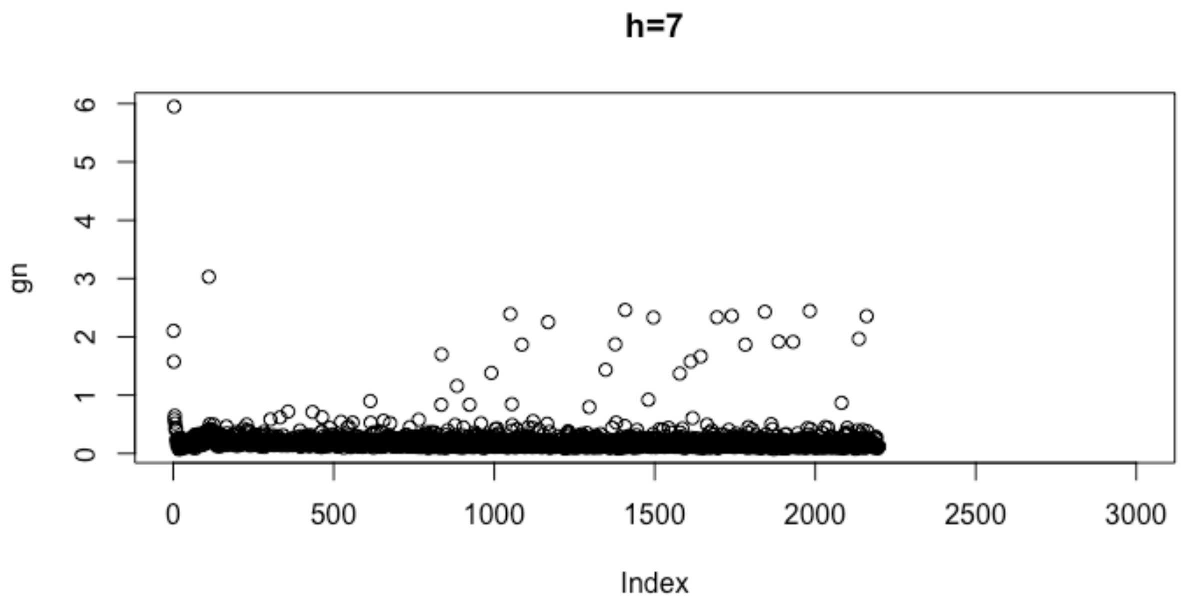

h=30
history$lrs
0.00
0.02
0.04
0.06
0
500
1000
1500
Index

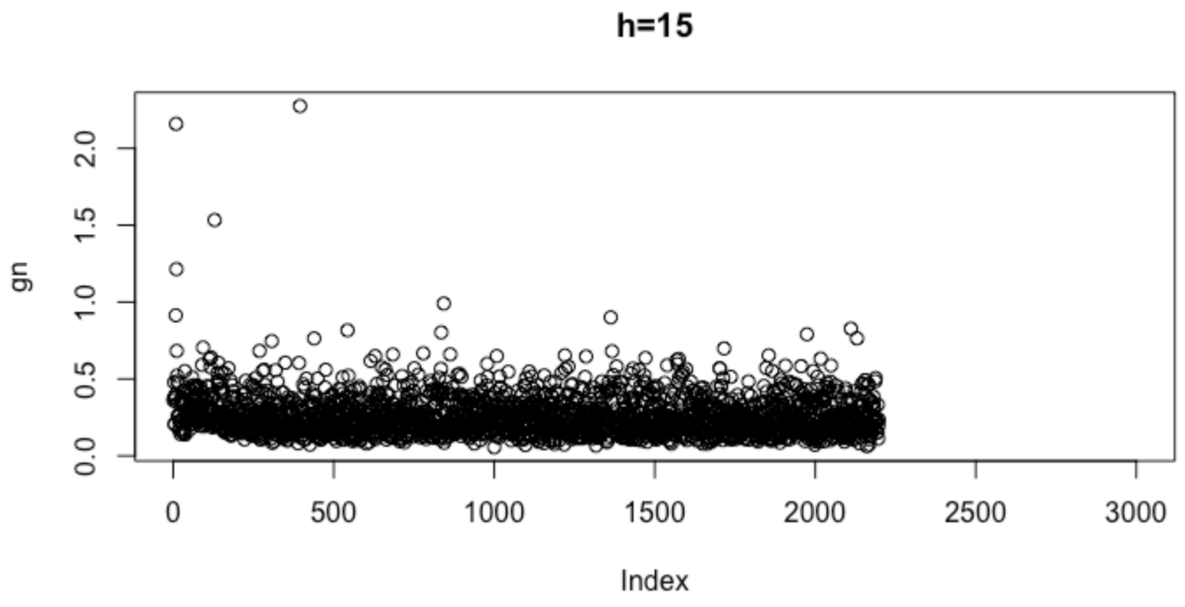

h=30
history$acces
0.75
0.80
0.85
0.90
0
500
1000
1500
Index

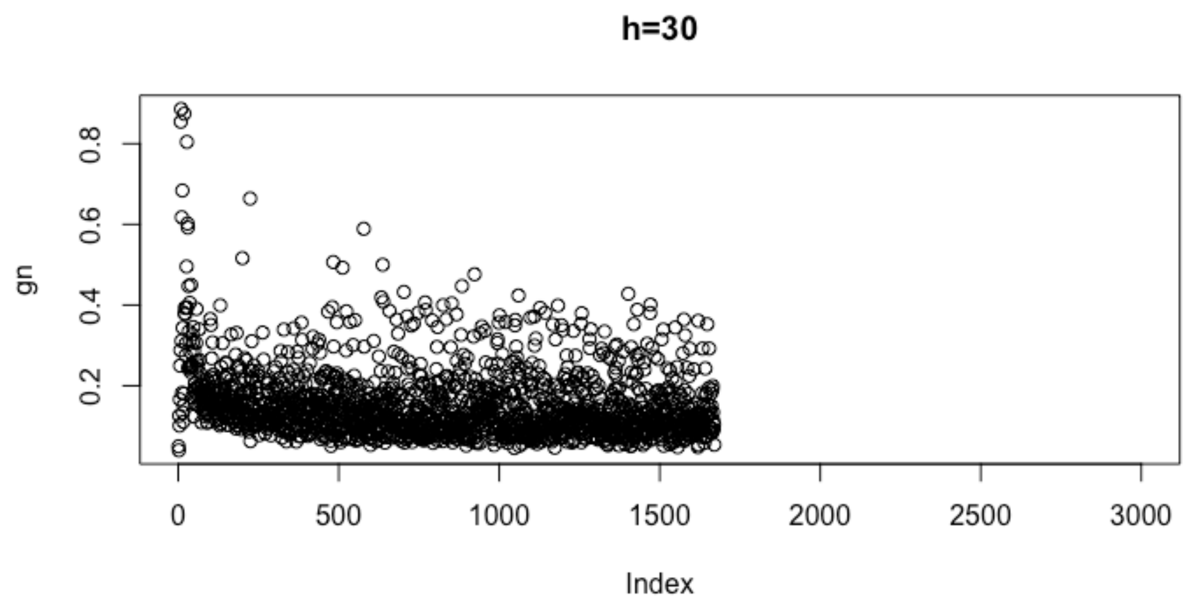
5.

After learning is stopped, the terminal weights was stored for further calculation.

For the different computing method of the time dependent gain rate, even I know in theoretical how to calculate the Gn, but it is not easy to get the results, so I do not give the Gn -n graph and the histogram of Gn. For the “keras” learning is good, Gn almost be 0.







By using the final weights and bias, I calculate the percentage of the cases which were truly of class Cm and which which were classified into class Cn by MLP; Note that for the training set data and the test set data used here is the one without enlarge, they are the original ones.
After each training I get two confusion matrices, one for the training set and another is for the test set. And then to compute the standard error which can be estimated by

:$\sigma = \frac{\sqrt{f * (1 - f)}}{size}$, thus can also get the σ_{train} and σ_{test}, further can also get the global

standard error by σ_{global}=$\sqrt{\sigma_{train}^2 + \sigma_{test}^2}$, for the “keras” learning gives pretty good results, all most 99% for the computing of class Ci-Ci, I just give the ci-ci percentage.
h=7
The training set confusion matrices and σ_{train} is as:
confusion(i,i)=almost 1, standard error almost 0
$f_{train.c1-c1}$=0.9208333,$f_{train.c2-c2}$=0.8515,$f_{train.c3-c3}$=0.9986667

The testing set confusion matrices and σ_{test} are as:
$f_{test.c1-c1}$=0.9144494,$f_{test.c2-c2}$=0.8444535,$f_{test.c3-c3}$=0.998772

h=15
The training set confusion matrices and σ_{train} is as:

confusion(i,i)=almost 1, standard error almost 0

$f_{train.c1-c1}$=0.9214167, $f_{train.c2-c2}$=0.8714167, $f_{train.c3-c3}$=0.96125

The testing set confusion matrices and σ_{test} are as:

$f_{test.c1-c1}$=0.9148588, $f_{test.c2-c2}$=0.8702415, $f_{test.c3-c3}$=0.9676627

h=30

The training set confusion matrices and σ_{train} are as:

confusion(i,i)=almost 1

$f_{train.c1-c1}$=0.8855833, $f_{train.c2-c2}$=0.8515833, $f_{train.c3-c3}$=0.9635833

The testing set confusion matrices and σ_{test} are as:

$f_{test.c1-c1}$=0.8829308 $f_{test.c2-c2}$=0.8456815, $f_{test.c3-c3}$= 0.9611134

Overall the percentages is really good.

5. Impact of various learning options

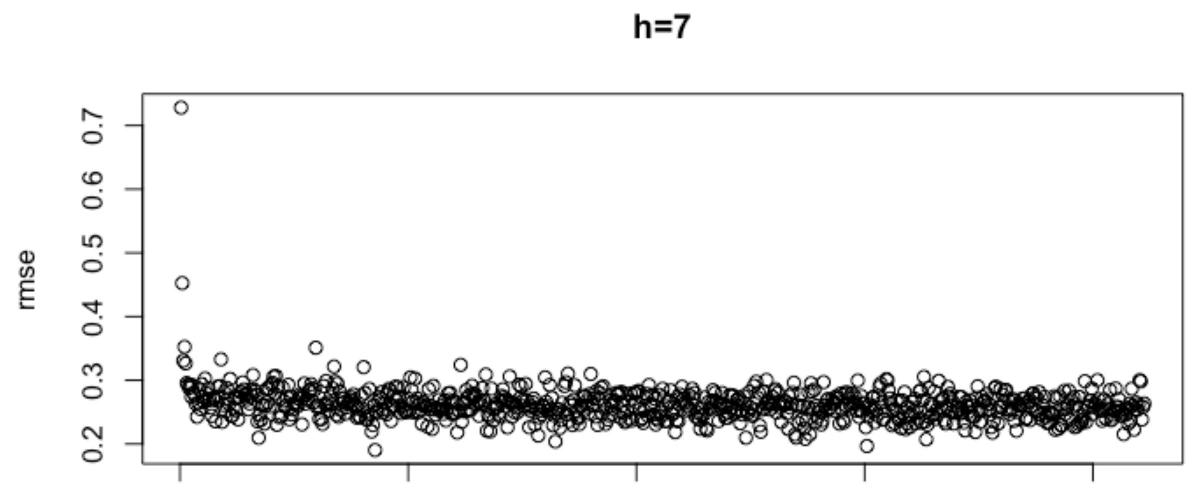
There are is not doubt that with small batch size, it will give more iteration, and will give better learning results, otherwise big batch size will give worse learning results, here in my learning method, I set the batch size to 300, the learning accuracy is pretty good for all the 3 different hidden layers; the first method I used the method

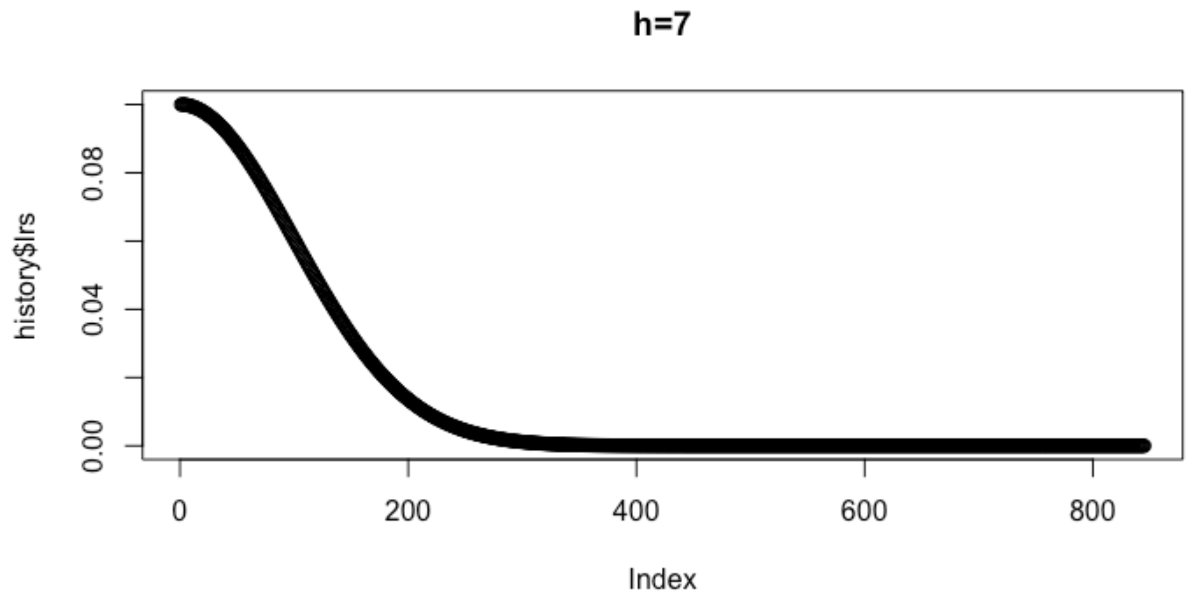
$\epsilon_n = \frac{constant}{n}$,

I do not think it gives a very good learning rate, it decrease a little sharp, but the second way I want to show here is by using the "Keras". I can initialize the weights and bias by different method, but the key of the learning accuracy is not that related to the initialized learning weights and bias; further more, the gradient descent step size is very important, we can not set the learning rate decrease to fast, that's why we need to find a very appropriate learning rate, set it to decrease but not decrease that much. Here I set lr0=0.1, decay=0.0001, momentum=0.5, and $lr = \frac{lr_0}{kt}$, where lr and k are hyper parameters, t is the iteration number,

$$lr * = \frac{1}{1 + self.decay * self.iterations};$$

Here I also give the results by using this learning rate, seem list the learning rate is much better, slowly, it doest not decrease that sharply. Here I give h=7 as the example to illustrate that:





at last, I tried 3 different hidden layer h={7,15,30}, from the accurate and the learning percentage, I can not see much difference, meaning that the hidden layer does not impact the learning that much by both the above two ways to set the learning rate.

6 PCA analysis of hidden layer

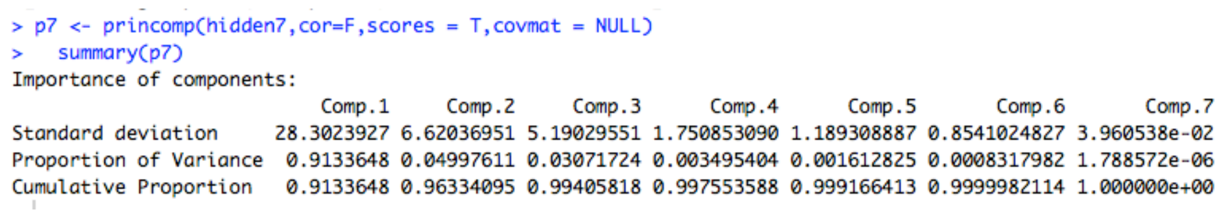
For each one of the hidden layers, the related PCA analysis is as followed:

h=7

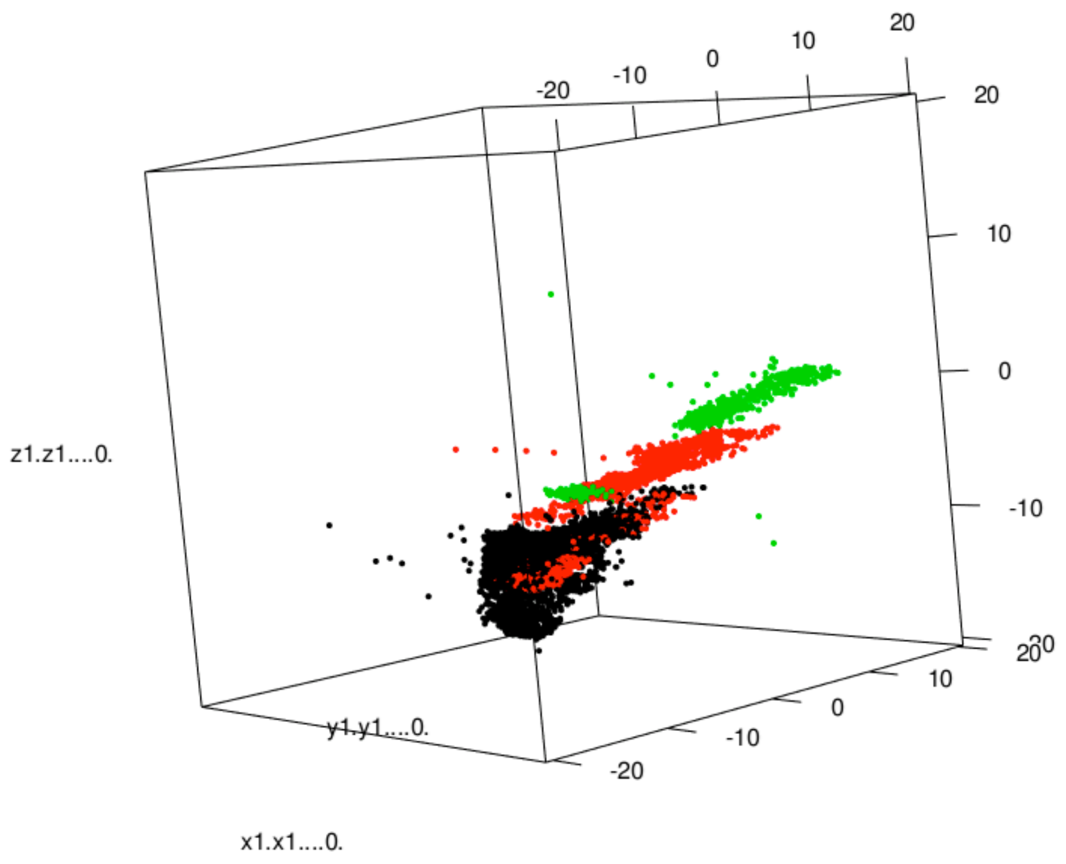
Bellowing is the PCA summary, if using 90% cutoff, we can choose 2 dimension,

After using the first 3 eigenvectors of the matrix Cova, and project all the vector of the hidden layer notes on the coordinate, we get the 3D plot, which is as bellowing.

we can see that for h=7 the separation is good.

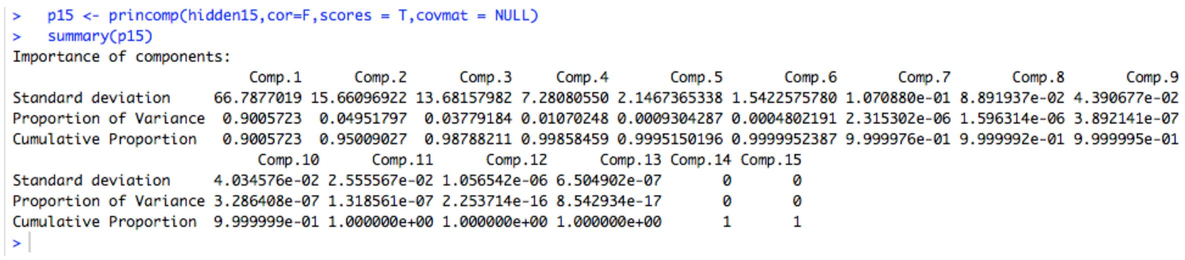


```
> p7 <- princomp(hidden7,cor=F,scores = T,covmat = NULL)
>   summary(p7)
Importance of components:
                          Comp.1     Comp.2     Comp.3      Comp.4      Comp.5       Comp.6       Comp.7
Standard deviation     28.3023927 6.62036951 5.19029551 1.750853090 1.189308887 0.8541024827 3.960538e-02
Proportion of Variance  0.9133648 0.04997611 0.03071724 0.003495404 0.001612825 0.0008317982 1.788572e-06
Cumulative Proportion   0.9133648 0.96334095 0.99405818 0.997553588 0.999166413 0.9999982114 1.000000e+00
```



h=15

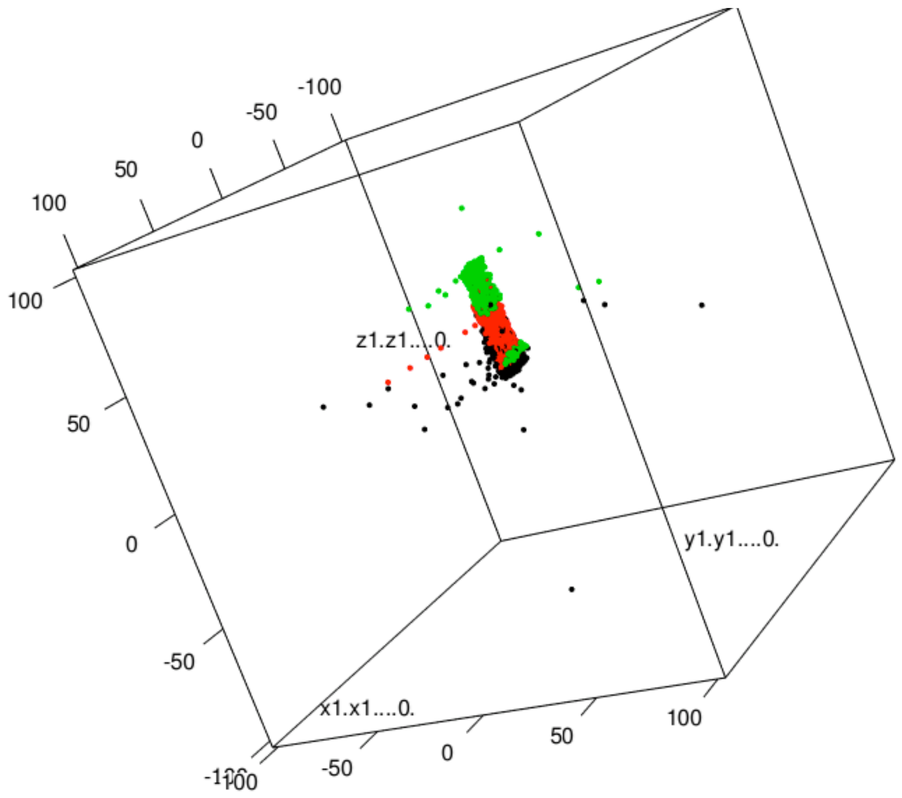
Bellowing is the PCA summary, if using 90% cutoff, we can also choose 2 dimension,



```
>   p15 <- princomp(hidden15,cor=F,scores = T,covmat = NULL)
>   summary(p15)
Importance of components:
                           Comp.1      Comp.2      Comp.3     Comp.4       Comp.5       Comp.6       Comp.7       Comp.8       Comp.9
Standard deviation     66.7877019 15.66096922 13.68157982 7.28080550 2.1467365338 1.5422575780 1.070880e-01 8.891937e-02 4.390677e-02
Proportion of Variance  0.9005723  0.04951797  0.03779184 0.01070248 0.0009304287 0.0004802191 2.315302e-06 1.596314e-06 3.892141e-07
Cumulative Proportion   0.9005723  0.95009027  0.98788211 0.99858459 0.9995150196 0.9999952387 9.999976e-01 9.999992e-01 9.999995e-01
                            Comp.10      Comp.11      Comp.12      Comp.13 Comp.14 Comp.15
Standard deviation     4.034576e-02 2.555567e-02 1.056542e-06 6.504902e-07       0       0
Proportion of Variance 3.286408e-07 1.318561e-07 2.253714e-16 8.542934e-17       0       0
Cumulative Proportion  9.999999e-01 1.000000e+00 1.000000e+00 1.000000e+00       1       1
> 
```

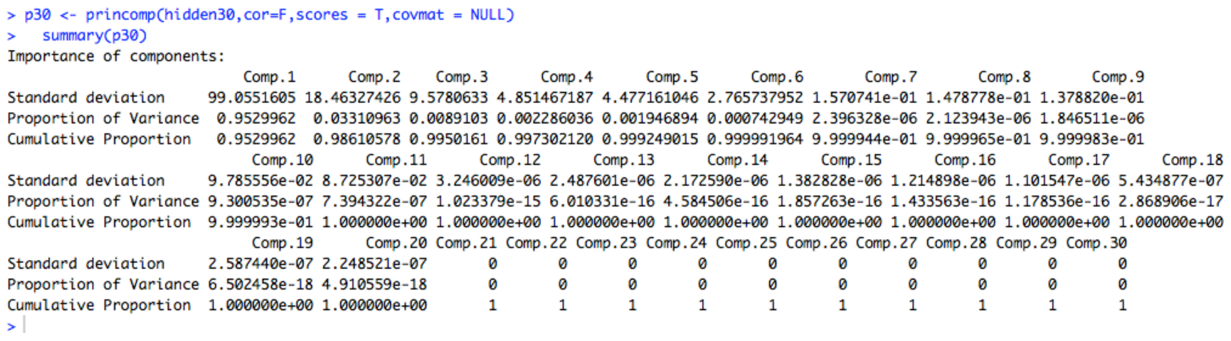
After using the first 3 eigenvectors of the matrix Cova, and project all the vector of the hidden layer notes on the coordinate, we get the 3D plot, which is as bellowing.

we can see that for h=15 the separation is not bad.



h=15

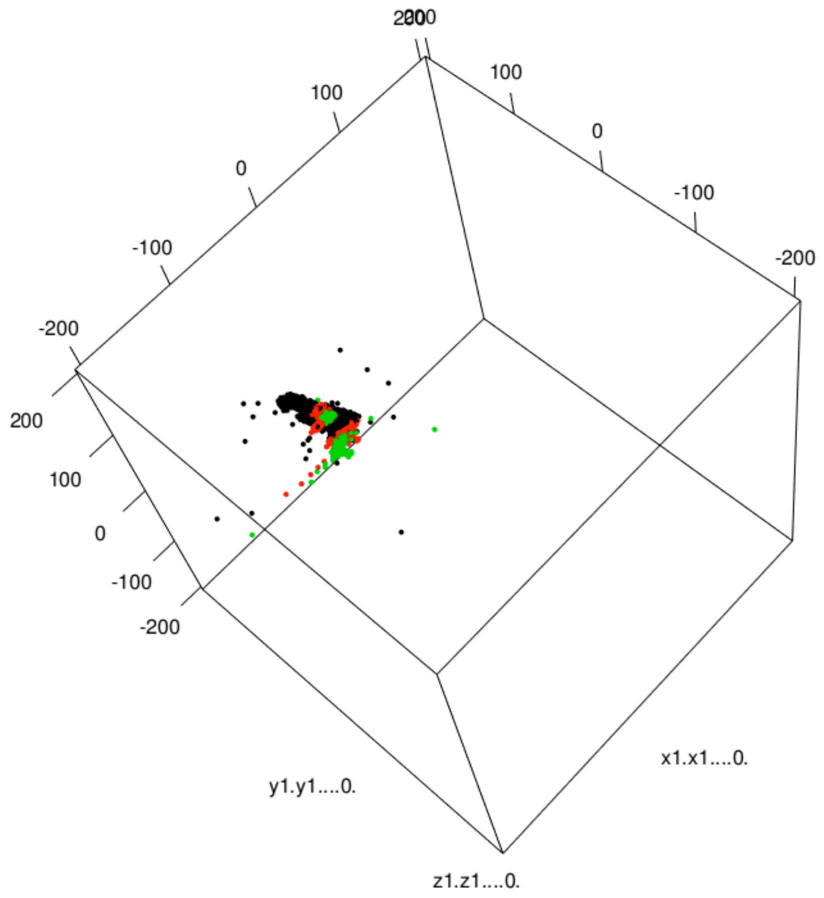
Bellowing is the PCA summary, if using 90% cutoff, we can also choose 2 dimension,



```
> p30 <- princomp(hidden30,cor=F,scores = T,covmat = NULL)
>   summary(p30)
Importance of components:
                           Comp.1      Comp.2    Comp.3      Comp.4      Comp.5      Comp.6       Comp.7       Comp.8       Comp.9
Standard deviation     99.0551605 18.46327426 9.5780633 4.851467187 4.477161046 2.765737952 1.570741e-01 1.478778e-01 1.378820e-01
Proportion of Variance  0.9529962  0.03310963 0.0089103 0.002286036 0.001946894 0.000742949 2.396328e-06 2.123943e-06 1.846511e-06
Cumulative Proportion   0.9529962  0.98610578 0.9950161 0.997302120 0.999249015 0.999991964 9.999944e-01 9.999965e-01 9.999983e-01
                            Comp.10      Comp.11      Comp.12      Comp.13      Comp.14      Comp.15      Comp.16      Comp.17      Comp.18
Standard deviation     9.785556e-02 8.725307e-02 3.246009e-06 2.487601e-06 2.172590e-06 1.382828e-06 1.214898e-06 1.101547e-06 5.434877e-07
Proportion of Variance 9.300535e-07 7.394322e-07 1.023379e-15 6.010331e-16 4.584506e-16 1.857263e-16 1.433563e-16 1.178536e-16 2.868906e-17
Cumulative Proportion  9.999993e-01 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00
                            Comp.19      Comp.20 Comp.21 Comp.22 Comp.23 Comp.24 Comp.25 Comp.26 Comp.27 Comp.28 Comp.29 Comp.30
Standard deviation     2.587440e-07 2.248521e-07       0       0       0       0       0       0       0       0       0       0
Proportion of Variance 6.502458e-18 4.910559e-18       0       0       0       0       0       0       0       0       0       0
Cumulative Proportion  1.000000e+00 1.000000e+00       1       1       1       1       1       1       1       1       1       1
> 
```

After using the first 3 eigenvectors of the matrix Cova, and project all the vector of the hidden layer notes on the coordinate, we get the 3D plot, which is as bellowing.

we can see that for h=30the separation is not that bad.



R CODE:

```
rm(list = ls())
#1. prepare the R
install.packages("drat", repos="https://cran.rstudio.com")
drat:::addRepo("dmlc")
install.packages("mxnet")
install.packages("devtools")
devtools::install_github("rstudio/keras")
library(keras)
install_keras()

#2. Prepare the data and do some change of the data
shuttle <- read.csv(file.choose())
c1 <- shuttle[shuttle$Classes=="1",]
c2 <- shuttle[shuttle$Classes=="2",]
c3 <- shuttle[shuttle$Classes=="3",]
c4 <- shuttle[shuttle$Classes=="4",]
c5.temp <- shuttle[shuttle$Classes=="5",]
c5 <- rbind(c5.temp,c5.temp)
c5 <- rbind(c5,c5.temp)
sh <- rbind(c1,c4,c5)
sh.cm <- sh <- rbind(c1,c4,c5.temp)

# get hidden layer h=s
pca.sh <- sh[,2:9]
pca.c <- cor(pca.sh)
pca.e <- eigen(pca.c)
pca.e
scale(as.matrix(pca.sh))%*%pca.e$vectors
e.pca <- pca.e$values
# by pca code
p <- prcomp(pca.sh,scale. = T) #p <- prcomp(pca.seed,retx=T,center=T,scale. = T) # a little
different with prcomp(pca.seed)
```

```
summary(p)  # this can check with the ratj answer, they are the same
predict(p)  # the same results as by hand scale()
p2 <- princomp(pca.sh)

# get  hidden layer h=S1
pca.c1 <- c1[,2:9]
pca.c.c1 <- cor(pca.c1)
pca.e.c1 <- eigen(pca.c.c1)
pca.e.c1
scale(as.matrix(pca.c1))%*%pca.e.c1$vectors
e.pca.c1 <- pca.e.c1$values
# by PCA code
p.c1 <- prcomp(pca.c1,scale. = T) #p <- prcomp(pca.seed,retx=T,center=T,scale. = T) # a little
different with prcomp(pca.seed)
summary(p.c1)  # this can check with the ratj answer, they are the same
predict(p.c1)  # the same results as by hand scale()
p2.c1 <- princomp(pca.c1)

# get hidden layer h=S4
pca.c4 <- c4[,2:9]
pca.c.c4 <- cor(pca.c4)
pca.e.c4 <- eigen(pca.c.c4)
pca.e.c4
scale(as.matrix(pca.c4))%*%pca.e.c4$vectors
e.pca.c4 <- pca.e.c4$values
# by pca code
p.c4 <- prcomp(pca.c4,scale. = T) #p <- prcomp(pca.seed,retx=T,center=T,scale. = T) # a little
different with prcomp(pca.seed)
summary(p.c4)  # this can check with the ratj answer, they are the same
predict(p.c4)  # the same results as by hand scale()
p2.c4 <- princomp(pca.c4)

# get h=S4
```

```
pca.c5 <- c5[,2:9]
pca.c.c5 <- cor(pca.c5)
pca.e.c5 <- eigen(pca.c.c5)
pca.e.c5
scale(as.matrix(pca.c5))%*%pca.e.c5$vectors
e.pca.c5 <- pca.e.c5$values
# by pca code
p.c5 <- prcomp(pca.c5,scale. = T) #p <- prcomp(pca.seed,retx=T,center=T,scale. = T) # a little
different with prcomp(pca.seed)
summary(p.c5)  # this can check with the ratj answer, they are the same
predict(p.c5)  # the same results as by hand scale()
p2.c5 <- princomp(pca.c5)

# inflate the data
sh <- sh[sample(1:nrow(sh),length(1:nrow(sh))),1:ncol(sh)]
shtr.temp <- sh[1:13000,]
shte.temp <- sh[13001:16061,]
train_x <- shtr.temp[,2:9]
train_x <-as.matrix(train_x)
train_y <- shtr.temp[,10]
test_x <- shte.temp[,2:9]
test_x <- as.matrix(test_x)
test_y <- shte.temp[,10]
shtr.temp <- sh[1:13000,]
shte.temp <- sh[13001:16061,]
train_x <- shtr.temp[,2:9]
train_x <- as.matrix(train_x)
train_y.temp1 <- shtr.temp[,10]
test_x <- shte.temp[,2:9]
test_x <-as.matrix(test_x)
test_y.temp2 <- shte.temp[,10]
train_y.temp2 <-to_categorical(train_y.temp1,6)
test_y.temp2<-to_categorical(test_y.temp2,6)
```

```
train_y <- train_y.temp2[,-1]
train_y <- train_y[,-2]
train_y <- train_y[,-2]
test_y <-test_y.temp2[,-1]
test_y <- test_y[,-2]
test_y <- test_y[,-2]

#for confusion matrix calculating set data
sh.cm <- sh.cm[sample(1:nrow(sh.cm),length(1:nrow(sh.cm))),1:ncol(sh.cm)]
shtr.temp.cm <- sh.cm[1:12000,]
shte.temp.cm <- sh.cm[12001:14443,]
train_x.cm <- shtr.temp.cm[,2:9]
train_x.cm <-as.matrix(train_x.cm)
train_y.cm <- shtr.temp.cm[,10]
test_x.cm <- shte.temp.cm[,2:9]
test_x.cm <- as.matrix(test_x.cm)
test_y.cm <- shte.temp.cm[,10]
shtr.temp.cm <- sh.cm[1:12000,]
shte.temp.cm <- sh.cm[12001:14443,]
train_x.cm <- shtr.temp.cm[,2:9]
train_x.cm <- as.matrix(train_x.cm)
train_y.temp1.cm <- shtr.temp.cm[,10]
test_x.cm <- shte.temp.cm[,2:9]
test_x.cm <-as.matrix(test_x.cm)
test_y.temp2.cm <- shte.temp.cm[,10]
train_y.temp2.cm <-to_categorical(train_y.temp1.cm,6)
test_y.temp2.cm<-to_categorical(test_y.temp2.cm,6)
train_y.cm <- train_y.temp2.cm[,-1]
train_y.cm <- train_y.cm[,-2]
train_y.cm <- train_y.cm[,-2]
test_y.cm <-test_y.temp2.cm[,-1]
test_y.cm <- test_y.cm[,-2]
test_y.cm <- test_y.cm[,-2]
```

#3. I wrote the function to get all the information about each iteration and also with the code by using Keras learning rate calculation.

```
library(keras)
# define custom callback class
LossHistory <- R6::R6Class("LossHistory",
                           inherit = KerasCallback,
                           public = list(
                           losses = NULL,
                           acces = NULL,
                           weights = NULL,
                           lrs = NULL,
                           lr = NULL,

                           #on_batch_begin=function(batch, logs=list()){
                            # iteration = k_get_value(self$model$optimizer$iterations)
                             #print(iteration)
                            # if(iteration == 0) {
                               #self$lr=k_get_value(self$model$optimizer$lr)
                             #  }
                             #else {
                             #  tmp = self$lr/(iteration+1)
                            #   k_set_value(self$model$optimizer$lr, tmp)
                             #}
                          # },

                           on_batch_end = function(batch, logs = list()) {
                             iteration = k_get_value(self$model$optimizer$iterations)
                             decay = k_get_value(self$model$optimizer$decay)
                             self$losses = c(self$losses, logs[["loss"]])
                             self$acces = c(self$acces, logs[["acc"]])
                             self$weights = c(self$weights, self$model$get_weights())
                             if (iteration > 3) {
                              # tmp = self$lrs[iteration-1]/(1.0+decay*iteration)
                              # self$lrs = c(self$lrs, tmp)
                              self$lrs = c(self$lrs, k_get_value(self$model$optimizer$lr))
                              }
                            else {
                              #self$lrs = c(self$lrs,k_get_value(self$model$optimizer$lr))
                              self$lrs = k_get_value(self$model$optimizer$lr)
                              }
                          }
                        ))

h<- c(30) # example as h=30
for (i in 1:1) {
  model <- keras_model_sequential() %>%
              layer_dense(units = h[i], activation = 'relu', input_shape = c(8),
kernel_initializer="RandomNormal",
              bias_initializer="RandomNormal") %>%
    layer_dense(units = 3, activation = 'softmax')

  epochs = 50
  sgd = optimizer_sgd(lr=0.2, decay=0.000, momentum=0)

  model %>% compile(
    optimizer = sgd,
    loss = "mean_squared_error",
    metrics = c("accuracy")
  )
  #fitting the model on the training dataset
  #stopping =callback_early_stopping(monitor = "loss",min_delta =1e-5,mode = "min")

  log_name<-sprintf('/Users/haiyangwang/Desktop/check_h_%02d/training.log',h[i])
  stopping =callback_early_stopping(monitor = "loss",min_delta =0.00001,mode = "min")
  history <- LossHistory$new()
  csv_logger = callback_csv_logger(log_name)

  history_simple <- model %>% fit(train_x, train_y, epochs = epochs, batch_size = 300,callbacks
= c(stopping,csv_logger,history))

  model %>% evaluate(train_x, train_y, epochs = epochs, batch_size = 300)

 # read in the history weights
  wt_1_diff_norm<-rep(0,3500)
  for (j in 1:3500) {
      weight_current <- unlist(history$weights[((j-1)*4+1):(j*4)])
      #hist(weight_current)

      if (j>1) {
        wt_1_cur=matrix((weight_current))
        wt_1_fml=matrix((weight_formal))
        wt_1_diff = wt_1_cur-wt_1_fml
        wt_1_diff_norm[j-1]=norm(data.matrix(wt_1_diff),type='F')
      }
      weight_formal=weight_current
    }

  }



  lrs=matrix(unlist(history$lrs))

  #h_current <- read.csv(file.choose(TRUE))
  h_current <- read.csv(log_name)
  title<-paste0('hidden layer h=',h[i])
  plot(h_current$epoch,sqrt(h_current$loss),xlab = "n",ylab = "RMSE",title(title))
  plot(h_current$epoch,h_current$acc,xlab = "n",ylab = "acc",title(title),type = "b")
  plot(wt_1_diff_norm,main = "delta weight h=7")  # delta weight
  plot(seq(1:100),seq(1:100)/0.001 *wt_1_diff_norm,main = "Gn h=7")



  #get confusion matrix for train
  train_pred<- model %>% predict(train_x.cm, batch_size = 100)
  train_pred_y = round(train_pred)
  train_cm <- matrix(0,3,3)
  for (i in 1:3)
  {train_cm[i,i]=sum(train_y.cm[,i]==train_pred_y[,i])/length(train_y.cm[,1])}
  # get sd error for train
  train_sd.error <-matrix(0,3,3)
  for (i in i:3)
  {train_sd.error[i,i] <- sqrt((1-train_cm[i,i])*train_cm[i,i]/(length(train_y[,1])))}

  # get confusion matrix for test
  test_pred<- model %>% predict(test_x.cm, batch_size = 100)
  test_pred_y = round(test_pred)
  test_cm <- matrix(0,3,3)
  for (i in 1:3)
  {test_cm[i,i]=sum(test_y.cm[,i]==test_pred_y[,i])/length(test_y.cm[,1])}
  # get sd error for
  test_sd.error <-matrix(0,3,3)
  for (i in i:3)
  {test_sd.error[i,i] <- sqrt((1-test_cm[i,i])*test_cm[i,i]/(length(test_y[,1])))}

# plot all the calculated data information
  plot(history$losses)
  rmse <- sqrt(history$losses)
  plot(rmse)
  title(main="h=7")
  plot(history$lrs)
  title(main="h=7")
```

```
 plot(wt_1_diff_norm)
 title(main="h=7")
 gn <- wt_1_diff_norm[1:3000]/lrs[1:3000]
 plot(gn)
 title(main="h=7")
 plot(history$acces)
 title(main="h=7")

#4. PCA analysis of the hidden layer and get the 3D plot
 hidden30.tem <- train_x %*% (matrix(unlist(keras::get_weights(model)[1]),8,30))
 hidden30.tem2<- matrix(rep(unlist(keras::get_weights(model)[2]),13000),13000,30)
 hidden30 <- hidden30.tem+hidden30.tem2
 p7 <- princomp(hidden7,cor=F,scores = T,covmat = NULL)
 summary(p7)
 p15 <- princomp(hidden15,cor=F,scores = T,covmat = NULL)
 summary(p15)
 p30 <- princomp(hidden30,cor=F,scores = T,covmat = NULL)
 summary(p30)
 w <- p30$loadings  #w[,i]%*%w[,i]=1 and w[,i]%*%w[,j]=0,orthonormal basis

 x<- y<- z<-c(rep(0,13000))
 for (i in 1:13000)
 {
   x[i] <- hidden30[i,]%*%w[,1]
   y[i] <- hidden30[i,]%*%w[,2]
   z[i] <- hidden30[i,]%*%w[,3]
 }

 x1=x*train_y[,1]
 y1=y*train_y[,1]
 z1=z*train_y[,1]
 x2=x*train_y[,2]
 y2=y*train_y[,2]
```

```
z2=z*train_y[,2]
x3=x*train_y[,3]
y3=y*train_y[,3]
z3=z*train_y[,3]
xyz1 <- data.frame(x1[x1!=0],y1[y1!=0],z1[z1!=0])
xyz2 <- data.frame(x2[x2!=0],y2[y2!=0],z2[z2!=0])
xyz3 <- data.frame(x3[x3!=0],y3[y3!=0],z3[z3!=0])
library(rgl)
plot3d(xyz1,col = 1,xlim = c(-100,100),ylim  = c(-100,100),zlim  = c(-100,100))
plot3d(xyz2,col = 2,xlim =  c(-100,100),ylim  = c(-100,100),zlim  = c(-100,100), add=TRUE)
plot3d(xyz3,col = 3,xlim =  c(-100,100),ylim  = c(-100,100),zlim  = c(-100,100), add=TRUE)
```