SMM636 Machine Learning (PRD2 A 2019/20)

R exercises 9: Neural networks

DR. Rui Zhu

2020-02-27

In R exercise 9, you will know

- How to apply a single hidden layer neural network for classification
- How to use the `Keras` package
- How to use neural network for dimension reduction

Don't forget to change your working directory!

1 Data: the stock market data

This data set consists of percentage returns for the S& P 500 stock index over 1, 250 days, from the beginning of 2001 until the end of 2005. For each date, we have recorded the percentage returns for each of the five previous trading days, `Lag1` through `Lag5`. We have also recorded `Volume` (the number of shares traded on the previous day, in billions), `Today` (the percentage return on the date in question) and `Direction` (whether the market was Up or Down on this date).

We create a test set of observations from 2005.

2 Neural network for classification

The `nnet()` function in the package `nnet` can perform neural network with one hidden layer for classification or regression. Note that the labels input in `nnet()` should be one-hot vectors.

```
library(ISLR)
library(nnet)
attach(Smarket)
train=(Year<2005)
train.X= Smarket [train ,2:7]
train.Y=Direction[train]
test.X= Smarket [!train ,2:7]
test.Y=Direction[!train]
#fit a neural net with 10 hidden units
train.Y=class.ind(train.Y)
set.seed(2)
nnModel1=nnet(train.X, train.Y, size = 10, rang=0.1, maxit = 500,entropy=TRUE)
```

```
## # weights:  92
## initial  value 1385.023855
## iter  10 value 1379.178550
## iter  20 value 1346.030017
## iter  30 value 1313.375433
## iter  40 value 1294.991510
## iter  50 value 1269.664434
```

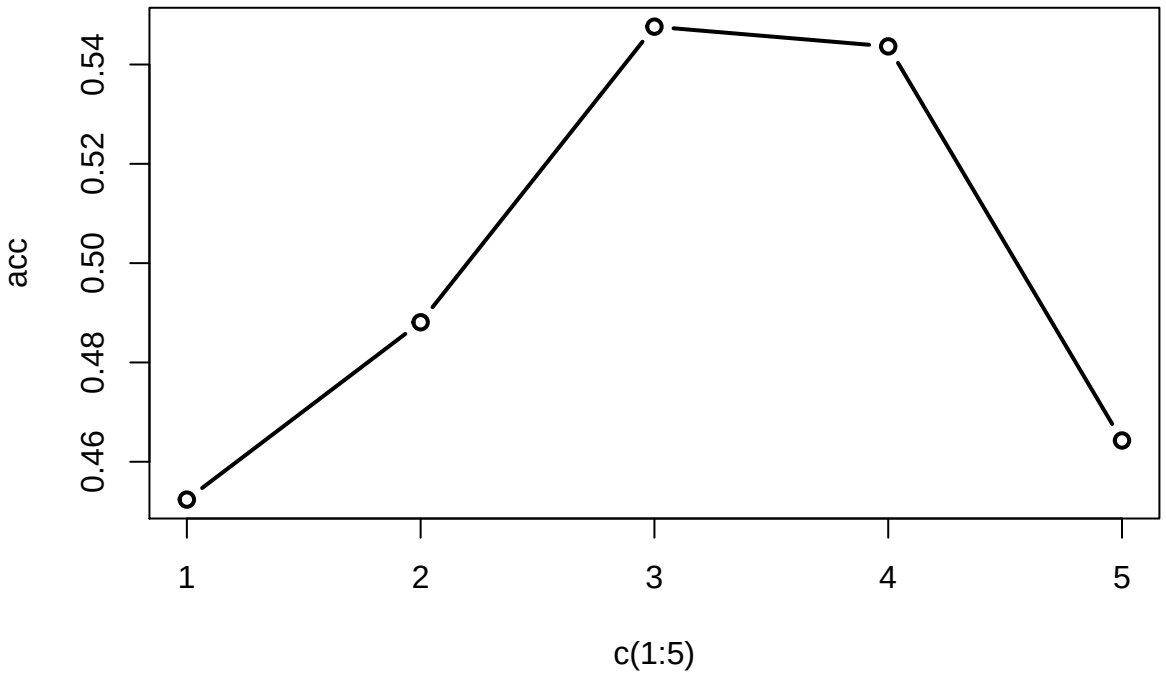
```
## iter  60 value 1257.267341
## iter  70 value 1236.269143
## iter  80 value 1213.628525
## iter  90 value 1190.390229
## iter 100 value 1170.549635
## iter 110 value 1149.707312
## iter 120 value 1124.578462
## iter 130 value 1115.631376
## iter 140 value 1112.796527
## iter 150 value 1112.192663
## iter 160 value 1112.083497
## iter 170 value 1112.052644
## final  value 1112.051706
## converged
```

```
pred1_prob=predict(nnModel1,test.X)
pred1=rep("Down",length(test.Y))
pred1[pred1_prob[,2]>0.5]="Up"
mean(test.Y==pred1)
```

```
## [1] 0.5039683
```

One problem of using `nnet()`, or neural networks, is that we obtain different predictions with different initial values.

```
# get a plot of accuracy using different initial values
r=c(0.01,0.1,0.2,0.3,0.4)
acc=vector("numeric",length(r))
set.seed(2)
for(ii in 1:length(r)){
nnModel1=nnet(train.X, train.Y, size = 10, rang=r[ii], maxit = 500,entropy=TRUE)
pred1_prob=predict(nnModel1,test.X)
pred1=rep("Down",length(test.Y))
pred1[pred1_prob[,2]>0.5]="Up"
acc[ii]=mean(test.Y==pred1)
}
plot(c(1:5),acc,lty=1,lwd=2,type="b")
```



The predictions also change with different numbers of hidden units. Thus it is important to choose a proper number of units and a proper number of hidden layers in practice.

```
# get a plot of accuracy using different number of hidden units
r=c(2,3,5,7,9,10,15,20)
acc=vector("numeric",length(r))
set.seed(2)
for(ii in 1:length(r)){
  nnModel1=nnet(train.X, train.Y, size = r[ii], rang=0.01, maxit = 500,entropy=TRUE)
  pred1_prob=predict(nnModel1,test.X)
  pred1=rep("Down",length(test.Y))
  pred1[pred1_prob[,2]>0.5]="Up"
  acc[ii]=mean(test.Y==pred1)}
```

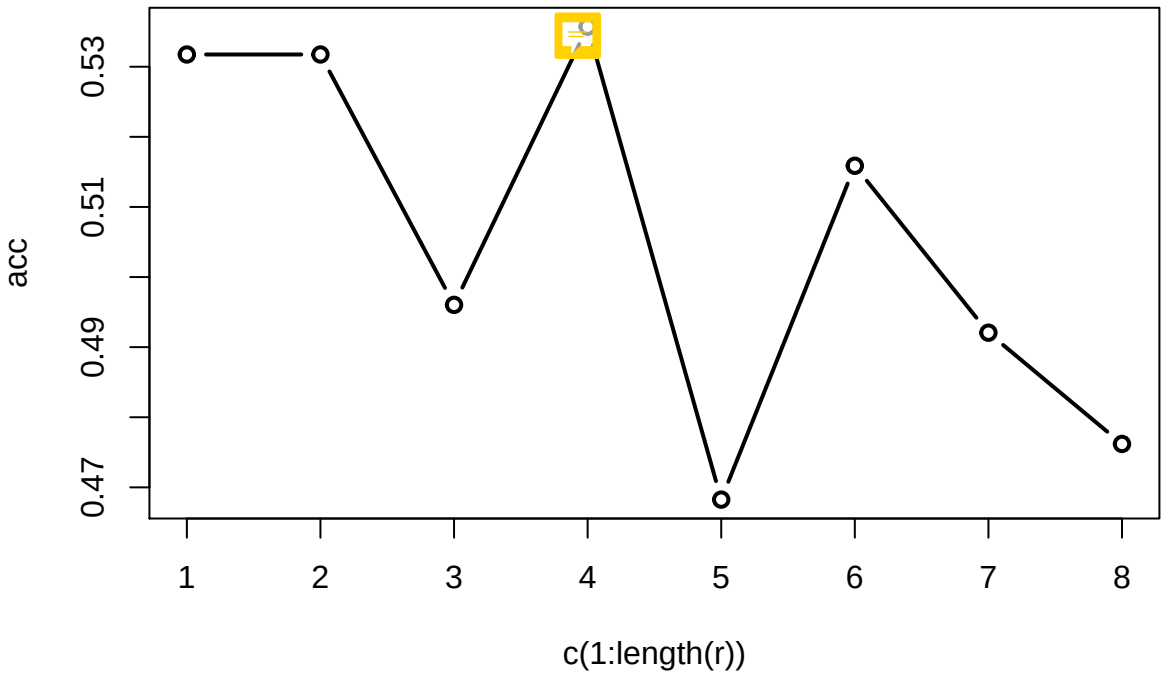
```
## # weights:  20
## initial  value 1383.672952
## iter  10 value 1381.351596
## iter  20 value 1373.562162
## iter  30 value 1368.988878
## iter  40 value 1360.559804
## iter  50 value 1358.706273
## iter  60 value 1357.844909
## iter  70 value 1357.334335
## iter  80 value 1357.197600
## iter  90 value 1357.177847
## iter  90 value 1357.177846
## final  value 1357.177846
## converged
## # weights:  29
## initial  value 1383.410544
## iter  10 value 1379.989361
## iter  20 value 1368.964399
## iter  30 value 1355.637915
```

```
## iter  40 value 1336.455435
## iter  50 value 1329.502418
## iter  60 value 1327.758297
## final  value 1327.734054
## converged
## # weights:  47
## initial  value 1383.499606
## iter  10 value 1379.970472
## iter  20 value 1363.047321
## iter  30 value 1352.296294
## iter  40 value 1346.644047
## iter  50 value 1339.148400
## iter  60 value 1327.419150
## iter  70 value 1308.826830
## iter  80 value 1292.294160
## iter  90 value 1287.579184
## iter 100 value 1287.022237
## final  value 1287.021609
## converged
## # weights:  65
## initial  value 1383.569257
## iter  10 value 1379.876102
## iter  20 value 1364.399303
## iter  30 value 1342.850143
## iter  40 value 1318.956922
## iter  50 value 1306.509486
## iter  60 value 1295.241701
## iter  70 value 1283.886932
## iter  80 value 1282.177029
## iter  90 value 1281.266543
## iter 100 value 1281.229212
## final  value 1281.229134
## converged
## # weights:  83
## initial  value 1383.708672
## iter  10 value 1380.751369
## iter  20 value 1374.503150
## iter  30 value 1347.510793
## iter  40 value 1323.767685
## iter  50 value 1312.239708
## iter  60 value 1297.072231
## iter  70 value 1276.311803
## iter  80 value 1256.051436
## iter  90 value 1237.425549
## iter 100 value 1233.816849
## iter 110 value 1233.097657
## iter 120 value 1232.931367
## iter 130 value 1232.783079
## iter 140 value 1232.362133
## iter 150 value 1231.788174
## iter 160 value 1231.203361
## iter 170 value 1231.144690
## iter 180 value 1231.132791
## iter 190 value 1231.128368
```

```
## iter 200 value 1231.126488
## iter 210 value 1231.123444
## iter 220 value 1231.120021
## iter 230 value 1231.080067
## iter 240 value 1230.903866
## iter 250 value 1230.826418
## iter 260 value 1230.772154
## iter 270 value 1230.765616
## iter 280 value 1230.757698
## iter 290 value 1230.751785
## iter 300 value 1230.445568
## iter 310 value 1230.441590
## iter 320 value 1230.437059
## iter 330 value 1230.431492
## iter 340 value 1230.423225
## iter 350 value 1230.418486
## iter 360 value 1230.402144
## iter 370 value 1230.375423
## iter 380 value 1230.371331
## iter 390 value 1230.364976
## iter 400 value 1230.341055
## iter 410 value 1230.263932
## iter 420 value 1229.949685
## iter 430 value 1229.431480
## iter 440 value 1229.248028
## iter 450 value 1229.124793
## iter 460 value 1229.006540
## iter 470 value 1228.870886
## iter 480 value 1228.809830
## iter 490 value 1228.645553
## iter 500 value 1228.542090
## final  value 1228.542090
## stopped after 500 iterations
## # weights:  92
## initial  value 1383.664618
## iter  10 value 1380.994657
## iter  20 value 1371.490670
## iter  30 value 1334.595754
## iter  40 value 1283.633740
## iter  50 value 1264.732677
## iter  60 value 1252.117703
## iter  70 value 1241.463172
## iter  80 value 1230.363608
## iter  90 value 1213.342512
## iter 100 value 1199.254036
## iter 110 value 1192.254250
## iter 120 value 1188.350305
## iter 130 value 1186.556763
## iter 140 value 1186.250327
## iter 150 value 1186.117467
## iter 160 value 1186.106495
## iter 170 value 1186.097401
## iter 180 value 1186.096215
## final  value 1186.095921
```

```
## converged
## # weights:  137
## initial  value 1383.546434
## iter  10 value 1381.353825
## iter  20 value 1370.092665
## iter  30 value 1343.347792
## iter  40 value 1278.824956
## iter  50 value 1254.091399
## iter  60 value 1220.866186
## iter  70 value 1190.805391
## iter  80 value 1175.422413
## iter  90 value 1160.967944
## iter 100 value 1148.022544
## iter 110 value 1142.762847
## iter 120 value 1135.025276
## iter 130 value 1122.592052
## iter 140 value 1096.358606
## iter 150 value 1075.287131
## iter 160 value 1071.495065
## iter 170 value 1070.422750
## iter 180 value 1070.294646
## iter 190 value 1070.266898
## final  value 1070.266840
## converged
## # weights:  182
## initial  value 1383.399083
## iter  10 value 1380.582637
## iter  20 value 1370.813464
## iter  30 value 1343.163459
## iter  40 value 1287.894810
## iter  50 value 1211.053864
## iter  60 value 1166.763872
## iter  70 value 1124.554966
## iter  80 value 1087.631385
## iter  90 value 1050.875880
## iter 100 value 1023.519385
## iter 110 value 997.409725
## iter 120 value 981.550324
## iter 130 value 959.183278
## iter 140 value 934.706134
## iter 150 value 908.757290
## iter 160 value 887.946818
## iter 170 value 881.338476
## iter 180 value 878.056528
## iter 190 value 877.575572
## iter 200 value 877.537084
## iter 210 value 877.533494
## iter 210 value 877.533487
## iter 210 value 877.533487
## final  value 877.533487
## converged
```

```
plot(c(1:length(r)),acc,lty=1,lwd=2,type="b")
```



3 Neural network in Keras

`Keras` is a nice package that can be used to efficiently design a network with a user-specified architecture. It can also produce nice graphs showing the training process. However, installing the package may take some time.

```
install.packages("devtools")
devtools::install_github("rstudio/keras")
library(keras)
install_keras()
```

After running the last line, you may see an error message. To solve it, you need to open Terminal in MacOS or cmd in Windows and type in the lines showing in the error message. Then re-run the last line. It takes some time to complete the installation.

The `Keras` documentation can be found in https://keras.io/models/sequential/. There's also a short tutorial on fashion images classification in https://keras.rstudio.com/articles/tutorial_basic_classification.html.

Let's build a simple network with one single hidden layer in `Keras`. Similarly to `nnet`, we have to transform the labels to one-hot vectors. We specific the model using three layers (input layer, hidden layer and output layer). `%>%` is used to connect the commands. The `summary` command shows the details of each layer and the total number of parameters to be trained.

We first get training and test data as follows.

```
library(keras)
###################
##training data
train=(Year<2005)
train.X= Smarket [train ,2:7]
train.Y=Direction[train]
test.X= Smarket [!train ,2:7]
```

```
test.Y=Direction[!train]
######transform labels to one hot vectors
train.Y=ifelse(train.Y=="Up",0,1)
test.Y=ifelse(test.Y=="Up",0,1)
train.Y=to_categorical(train.Y, 2)
test.Y=to_categorical(test.Y, 2)
```

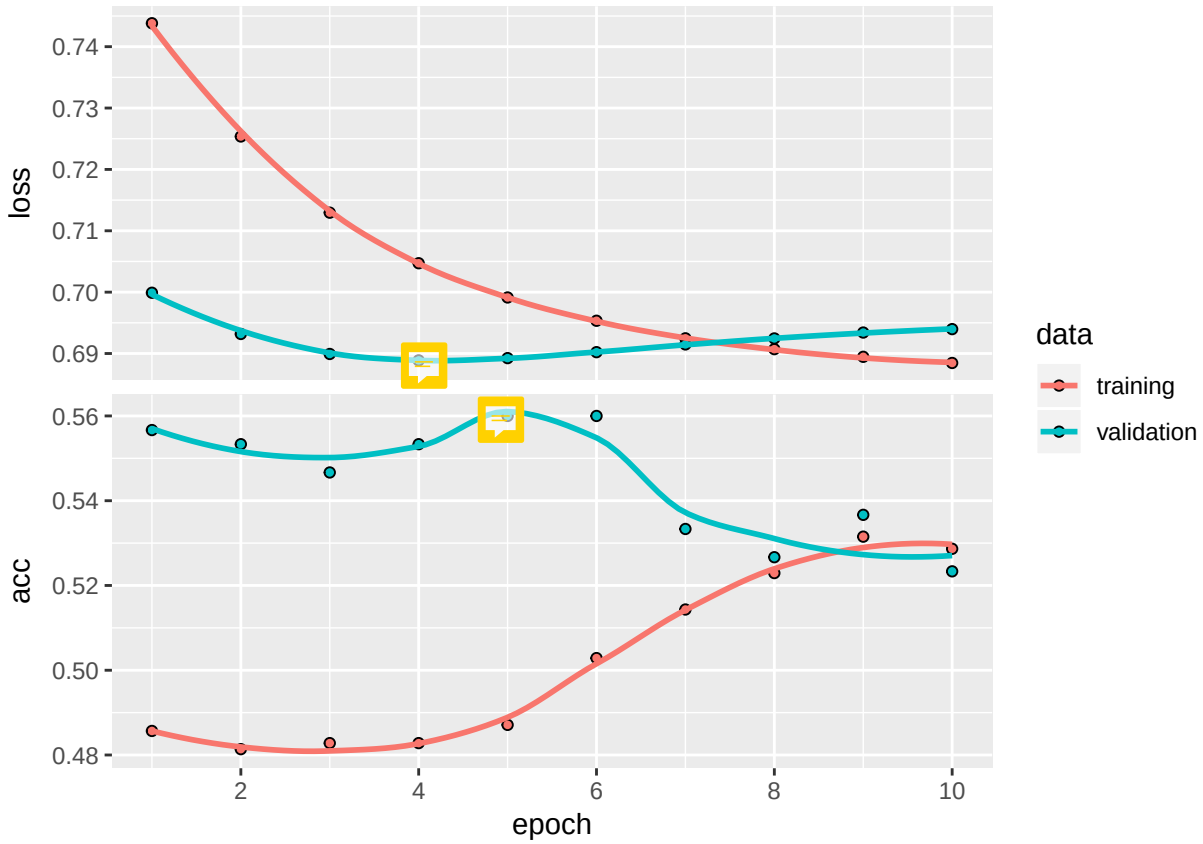
We now build a neural network with one hidden layer.

```
######set up keras model
model <- keras_model_sequential()
model %>%
  layer_dense(units = 6, activation = 'tanh', input_shape = ncol(train.X)) %>%
  layer_dense(units = 2, activation = 'softmax')
summary(model)
```

```
## ___________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## ===========================================================================
## dense (Dense)                    (None, 6)                     42
## ___________________________________________________________________________
## dense_1 (Dense)                  (None, 2)                     14
## ===========================================================================
## Total params: 56
## Trainable params: 56
## Non-trainable params: 0
## ___________________________________________________________________________
```

We then compile the model and fit it using the training data. During the training process, we can see that the function generates a plot showing the losses and accuracies on the training set and the validation set.

```
######compile the model
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)
######fit the model
set.seed(2)
history <- model %>% fit(
  as.matrix(train.X), train.Y,
  epochs = 10,
  validation_split = 0.3
)
plot(history)
```



The last `plot()` function shows the fitted lines of training and validation losses and accuracies.

We can then evaluate the model on the test set

```
######get accuracy on test data
model %>% evaluate(as.matrix(test.X), test.Y)
```

```
## $loss
## [1] 0.6834705
##
## $acc
## [1] 0.5634921
```

3.0.0.1 Exercise

Build a neural network with two hidden-layers for classification.

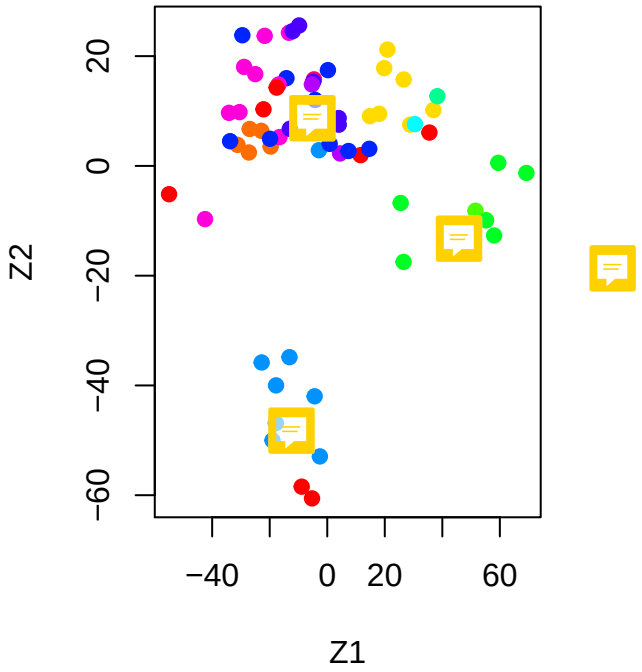
4 Neural networks for dimension reduction

In this section, we will use the gene expression data in `ISLR` package.

```
library(ISLR)
######get gene data
nci.labs=NCI60$labs
nci.data=NCI60$data
```

We first find the principal components of the data based on PCA and get the PC plots. We label each instance with their known class labels, i.e. the samples from the same class have the same color.

```
####################################
######PCA
pr.out=prcomp(nci.data, scale=TRUE)
######PC plots
Cols=function(vec){
  cols=rainbow(length(unique(vec)))
  return(cols[as.numeric(as.factor(vec))])
}
par(mfrow=c(1,2))
plot(pr.out$x[,1:2], col=Cols(nci.labs), pch=19,
       xlab="Z1",ylab="Z2")
```



We then use a neural network with one hidden layer and linear activations functions for dimension reduction. The point is to set the input and output layers with the same size.

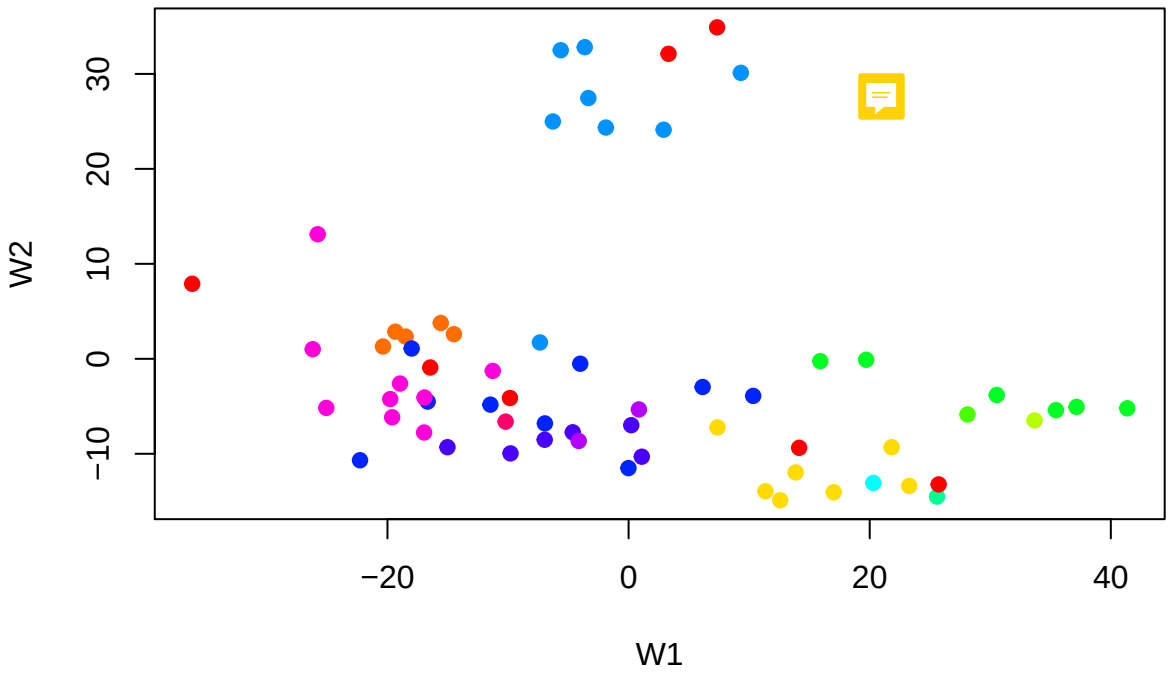
```
#################################################################
######Neural network with one hidden layer
model <- keras_model_sequential()
model %>%
  layer_dense(units = 2, activation = 'linear', input_shape = ncol(nci.data),name = "subspace") %>%
  layer_dense(units = ncol(nci.data), activation = 'linear')
summary(model)
```

```
## ________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ========================================================================
## subspace (Dense)                    (None, 2)                       13662
## ________________________________________________________________________
## dense_2 (Dense)                     (None, 6830)                    20490
## ========================================================================
## Total params: 34,152
## Trainable params: 34,152
## Non-trainable params: 0
```

```
## ------------------------------------------------------------------------
model %>% compile(
  loss = 'mean_squared_error',
  optimizer = "adam"
)
history <- model %>% fit(
  as.matrix(nci.data), as.matrix(nci.data),
  epochs = 80
)
```

We can then get the middle hidden layer out and project the training instances to the subspace spanned by the two hidden layers.

```
################################################################
######Neural network with one hidden layer
subspace <- keras_model(inputs = model$input, outputs = get_layer(model, "subspace")$output)
projection1 <- predict(subspace, nci.data)
plot(projection1, col=Cols(nci.labs), pch=19,
     xlab="W1",ylab="W2")
```



We obtain the following two graphs for PCA (on the left) and neural network (on the right). It seems that the two subspaces are quite similar. Here we only set `epochs = 80`, but you can set it to a larger number to further reduce the reconstruction error and see the outputs.

4.0.0.1 Exercise

We can also use neural network to do nonlinear dimension reduction. It usually takes much longer time to reduce the reconstruction error (usually need a large `epochs`). Try different numbers of hidden layers and hidden units and draw the projection plots. Can you see any difference compared with the linear one?