LSTM network in R, In this tutorial, we are going to discuss Recurrent Neural Networks. Recurrent Neural Networks are very useful for solving sequence of numbers-related issues.

The major applications involved in the sequence of numbers are text classification, time series prediction, frames in videos, DNA sequences Speech recognition problems, etc..

A special type of Recurrent Neural network is LSTM Networks.

LSTM networks are very popular and handy.

**What is mean by LSTM?**

LSTM stands for long short-term memory.

LSTM network helps to overcome gradient problems and makes it possible to capture long-term dependencies in the sequence of words or integers.

In this tutorial, we are using the internet movie database (IMDB). This database contains sentiments of movie reviews like 25000 positive reviews and 25000 negative reviews.

**Load library**

library(keras)

library(tensorflow)

use\_condaenv("keras-tf", required = T)

**Getting Data**

imdb <- dataset\_imdb(num\_words = 500)

These datasets are already pre processed so no need to clean the datasets.

c(c(train\_x, train\_y), c(test\_x, test\_y)) %<-% imdb

length(train\_x); length(test\_x)

train\_x and test\_x contains integer values

train\_y and test\_y contains labels (0 & 1).

0 represent the negative sentiment and 1 represent positive sentiment in the movie review

table(train\_y)

train\_y

    0     1

12500 12500

table(test\_y)

test\_y

    0     1

12500 12500

This indicates that our dataset is balanced.

Words in the movie review are represented by unique integers and each integer is assigned by overall frequency in the dataset. Customer review can extract from the below command.

train\_x[[10]]

[1]   1  14  20  47 111 439   2  19  12  15 166  12 216 125  40   6 364 352   2   2  39 294  11  22 396  13  28   8 202  12   2  23  94

[34]   2 151 111 211 469   4  20  13 258   2   2   2  12  16  38  78  33 211  15  12  16   2  63  93  12   6 253 106  10  10  48 335 267

[67]  18   6 364   2   2  20  19   6   2   7   2 189   5   6   2   7   2   2  95   2   6   2   7   2   2  49 369 120   5  28  49 253  10

[100]  10  13   2  19  85   2  15   4 481   9  55  78   2   9 375   8   2   8   2  76   7   4  58   5   4   2   9 243   7  43  50

Before doing any further analysis we need to make ensure the length of the movie reviews are equal, The current dataset has different length this can overcome based on padding process.

**Padding sequences**

train\_x <- pad\_sequences(train\_x, maxlen = 90)

num [1:25000, 1:90] 14 2 360 2 13 0 26 11 6 13 ...

test\_x <- pad\_sequences(test\_x, maxlen = 90)

num [1:25000, 1:90] 0 2 30 8 10 20 2 2 2 50 ...

Now all the train\_x and test\_x integers restricted to 90 only. So padding removed all extra integers.

Now you can examine **train\_x[10,]** customer review again

[1]  13 258   2   2   2  12  16  38  78  33 211  15  12  16   2  63  93  12   6 253 106  10  10  48 335 267  18   6 364   2   2  20  19   6

[35]   2   7   2 189   5   6   2   7   2   2  95   2   6   2   7   2   2  49 369 120   5  28  49 253  10  10  13   2  19  85   2  15   4 481

[69]   9  55  78   2   9 375   8   2   8   2  76   7   4  58   5   4   2   9 243   7  43  50

If the dataset contains fewer number integers suppose 60 integers remaining 30 integers that is 0 will be added automatically.

**Model**

Initiate model with keras function kera\_model\_sequantiall and embedded the recurrent neural network layers.

model <- keras\_model\_sequential()

model %>%

  layer\_embedding(input\_dim = 500, output\_dim = 32) %>%

  layer\_simple\_rnn(units = 32) %>%

  layer\_dense(units = 1, activation = "sigmoid")

activation we used sigmoid function that is very useful for interpretation purposes.

**Compile Model**

model %>% compile(optimizer = "rmsprop",

                  loss = "binary\_crossentropy",

                  metrics = c("acc"))

**Fit model**

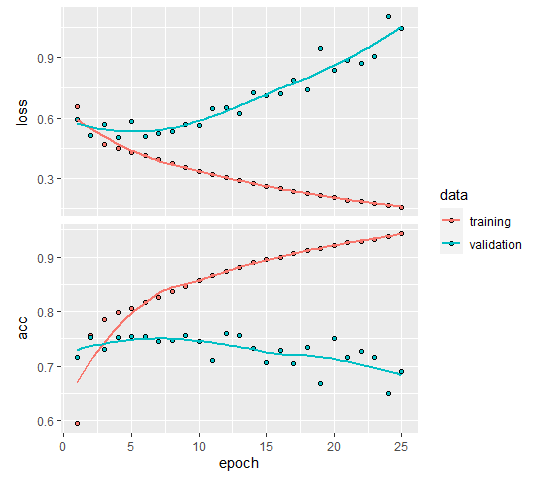
history <- model %>% fit(train\_x, train\_y,

                         epochs = 25,

                         batch\_size = 128,

                         validation\_split = 0.2)

plot(history)



validation\_split indictes 20% of the dataset used for validation purposes.

The top one is for loss and the second one is for accuracy, now you can see validation dataset loss is increasing and accuracy is decreasing from a certain epoch onwards. So this because of overfitting.

**Model Prediction**

model %>% evaluate(train\_x, train\_y)

loss       acc

0.3644736 0.8765600

pred <- model %>%

predict\_classes(train\_x)

table(Predicted=pred, Actual=imdb$train$y)

Actual Predicted

0     1

0 11503  2089

1   997 10411

model %>% evaluate(test\_x, test\_y)

loss      acc

1.032544 0.687720

pred1 <- model %>%

predict\_classes(test\_x)

table(Predicted=pred1, Actual=imdb$test$y)

Actual Predicted

0    1

0 9203 4510

1 3297 7990

In the training dataset, we got 87% of accuracy and it falls into 68% in the test dataset.

So improvement required in the model for better prediction.

You can make some changes in the model

model %>%

  layer\_embedding(input\_dim = 500, output\_dim = 32) %>%

  layer\_simple\_rnn(units = 32,return\_sequences = TRUE,activation = 'relu') %>%

  layer\_simple\_rnn(units = 32,return\_sequences = TRUE,activation = 'relu') %>%

  layer\_simple\_rnn(units = 32) %>%

  layer\_dense(units = 1, activation = "sigmoid")

In the above model instead of 1 layer, we used 3 layers, return sequences mentioned as TRUE and relu activation function used. Other changes we can do in padding. In the current model we used 90 instead of that we can find out the average customer review and the same can use for padding.

z<-NULL

for(i in 1:250000){z[i]<-print(length(train\_x[[i]]))}

summary(z)

Min. 1st Qu.  Median    Mean 3rd Qu.    Max.

   11.0   130.0   178.0   238.7   291.0  2494.0

Median is coming 178 and mean is 238 and we use some middle number for padding like 200.

**Padding sequences**

train\_x <- pad\_sequences(train\_x, maxlen = 200)

test\_x <- pad\_sequences(test\_x, maxlen = 200)

Rerun the model and check the accuracy again.

model %>% evaluate(train\_x, train\_y)

    loss       acc

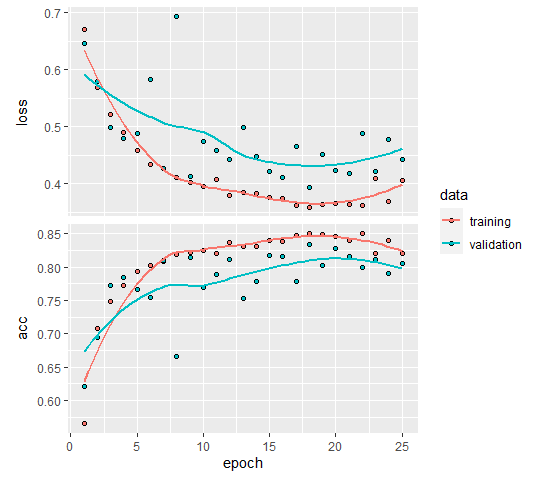
0.3733827 0.8421200

The train dataset accuracy is 84% earlier it was 87%

model %>% evaluate(test\_x, test\_y)

    loss       acc

0.4351899 0.8114400



Test data set accuracy significantly improved from 68% to 81%.

Now you can check with simple LSTM model for better prediction

**LSTM Network in R**

model %>%

  layer\_embedding(input\_dim = 500, output\_dim = 32) %>%

  layer\_lstm(units = 32,return\_sequences = TRUE) %>%

  layer\_lstm(units = 32,return\_sequences = TRUE) %>%

  layer\_lstm(units = 32) %>%

  layer\_dense(units = 1, activation = "sigmoid")

When you are using LSTM model try the optimizer “**adam**” for better prediction.