IMDB - Hypertuning

mchalama

2/7/2020

The IMDB dataset

Load IMDB dataset The IMDB dataset is preloaded in the Keras

```
library(keras)
library(tensorflow)
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------
--------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------------------
--------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(cowplot)
```

```
##
## ************************************************
```

```
## Note: As of version 1.0.0, cowplot does not change the
```

```
##   default ggplot2 theme anymore. To recover the previous
```

```
##   behavior, execute:
##   theme_set(theme_cowplot())
```

```
## ************************************************
```

```
imdb <- dataset_imdb(num_words = 10000)
c(c(train_data, train_labels), c(test_data, test_labels)) %<-% imdb
```

The argument num_words = 10000 means that we will only keep the top 10,000 most frequently occurring words in the training data. Rare words will be discarded

The variables train_data and test_data are lists of reviews, each review being a list of word indices (encoding a sequence of words). train_labels and test_labels are lists of 0s and 1s, where 0 stands for “negative” and 1 stands for “positive”:

Preparing the Data

```
vectorize_sequences <- function(sequences, dimension = 10000) {
  # Create an all-zero matrix of shape (len(sequences), dimension)
  results <- matrix(0, nrow = length(sequences), ncol = dimension)
  for (i in 1:length(sequences))
    # Sets specific indices of results[i] to 1s
    results[i, sequences[[i]]] <- 1
  results
}
# Our vectorized training data
x_train <- vectorize_sequences(train_data)
# Our vectorized test data
x_test <- vectorize_sequences(test_data)
# Our vectorized labels
y_train <- as.numeric(train_labels)
y_test <- as.numeric(test_labels)
set.seed(123)
val_indices <- sample(1:nrow(x_train),nrow(x_train)*0.40)
x_val <- x_train[val_indices,]
partial_x_train <- x_train[-val_indices,]
y_val <- y_train[val_indices]
partial_y_train <- y_train[-val_indices]
```

Building our network and Validating our Approach

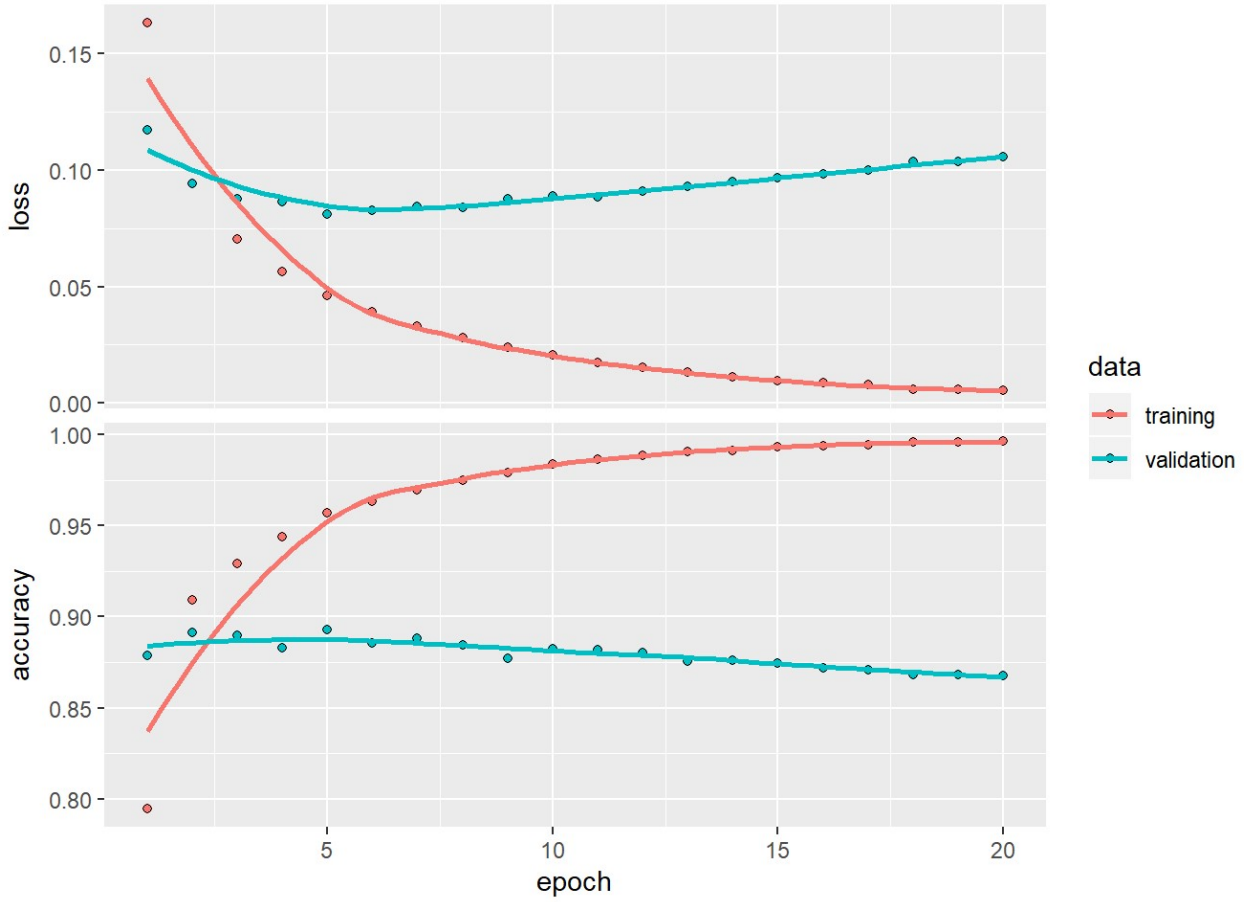
1-hidden layer network without any technique

1-hidden layer network with 16 units tanh activation mse loss function

```
model <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "tanh", input_shape = c(10000)) %>%
  layer_dense(units = 1, activation = "sigmoid")
model %>% compile(
  optimizer = "rmsprop",
  loss = "mse",
  metrics = c("accuracy")
)
history <- model %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
str(history)
```

```
## List of 2
##  $ params :List of 7
##   ..$ batch_size   : int 512
##   ..$ epochs       : int 20
##   ..$ steps        : num 30
##   ..$ samples      : int 15000
##   ..$ verbose      : int 0
##   ..$ do_validation: logi TRUE
##   ..$ metrics      : chr [1:4] "loss" "accuracy" "val_loss" "val_accuracy"
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 0.1632 0.0943 0.0703 0.0564 0.0461 ...
##   ..$ accuracy    : num [1:20] 0.795 0.909 0.929 0.944 0.957 ...
##   ..$ val_loss    : num [1:20] 0.1173 0.0943 0.0877 0.0866 0.0809 ...
##   ..$ val_accuracy: num [1:20] 0.879 0.891 0.89 0.883 0.893 ...
##  - attr(*, "class")= chr "keras_training_history"
```

```
plot(history)
```




```
model %>% fit(x_train, y_train, epochs = 3, batch_size = 512)
results <- model %>% evaluate(x_test, y_test)
results
```

```
## $loss
## [1] 0.1135264
##
## $accuracy
## [1] 0.86052
```

```
model %>% predict(x_test[1:10,])
```

```
##              [,1]
##  [1,] 3.000063e-02
##  [2,] 9.999934e-01
##  [3,] 9.907043e-01
##  [4,] 7.204456e-01
##  [5,] 9.963710e-01
##  [6,] 8.709746e-01
##  [7,] 9.999922e-01
##  [8,] 7.298589e-05
##  [9,] 9.976537e-01
## [10,] 9.977077e-01
```

This network begins to overfit after three epochs.

Here the validation Accuracy is 86% and loss of 12%. The result seems good. But let's try to improve the model by using other techniques like regularization and dropout

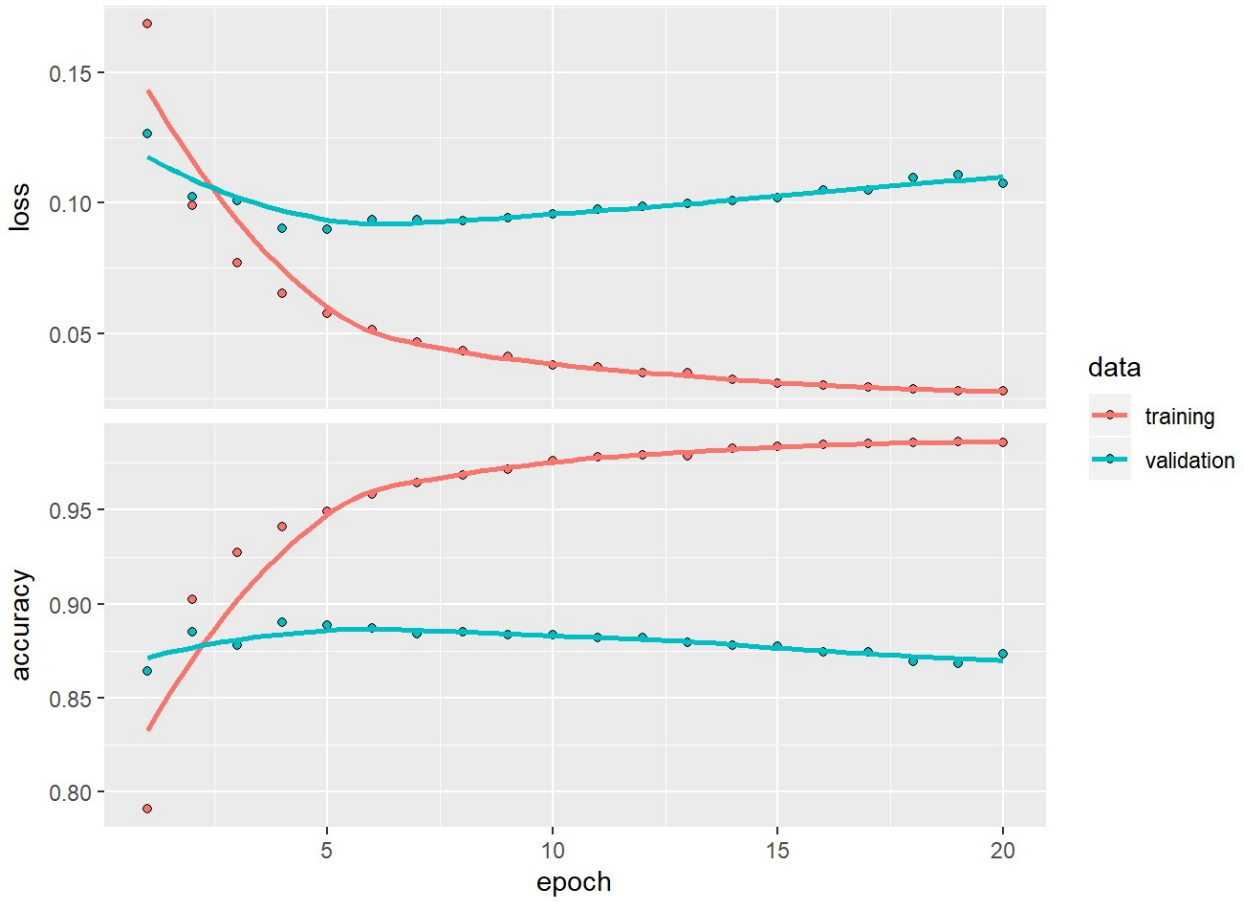
1- hidden layer network with regularization

1-hidden layer network with 16 units tanh activation mse loss function regularization technique

```
model <- keras_model_sequential() %>%
  layer_dense(units = 16,  kernel_regularizer = regularizer_l2(.0001),activation = "ta
nh", input_shape = c(10000)) %>%
  layer_dense(units = 1, activation = "sigmoid")
model %>% compile(
  optimizer = "rmsprop",
  loss = "mse",
  metrics = c("accuracy")
)
history <- model %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
str(history)
```

```
## List of 2
##  $ params :List of 7
##   ..$ batch_size   : int 512
##   ..$ epochs       : int 20
##   ..$ steps        : num 30
##   ..$ samples      : int 15000
##   ..$ verbose      : int 0
##   ..$ do_validation: logi TRUE
##   ..$ metrics      : chr [1:4] "loss" "accuracy" "val_loss" "val_accuracy"
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 0.1686 0.0991 0.0771 0.0651 0.0573 ...
##   ..$ accuracy    : num [1:20] 0.791 0.903 0.927 0.941 0.949 ...
##   ..$ val_loss    : num [1:20] 0.1264 0.1021 0.1008 0.0902 0.0898 ...
##   ..$ val_accuracy: num [1:20] 0.864 0.885 0.878 0.89 0.889 ...
##  - attr(*, "class")= chr "keras_training_history"
```

```
plot(history)
```




```
model %>% fit(x_train, y_train, epochs = 4, batch_size = 512)
results <- model %>% evaluate(x_test, y_test)
results
```

```
## $loss
## [1] 0.1162834
##
## $accuracy
## [1] 0.86148
```

```
model %>% predict(x_test[1:10,])
```

```
##             [,1]
##  [1,] 0.02977899
##  [2,] 0.99997854
##  [3,] 0.83844328
##  [4,] 0.62568808
##  [5,] 0.98647416
##  [6,] 0.75275356
##  [7,] 0.99997365
##  [8,] 0.00030756
##  [9,] 0.98417687
## [10,] 0.99333525
```

This network begins to overfit after four epochs

Here the validation Accuracy is 87% and loss of 11%. The result seems good. But the accuracy is lesser when compared to the previous one. We can try to improve the model by using both the techniques regularization and dropout

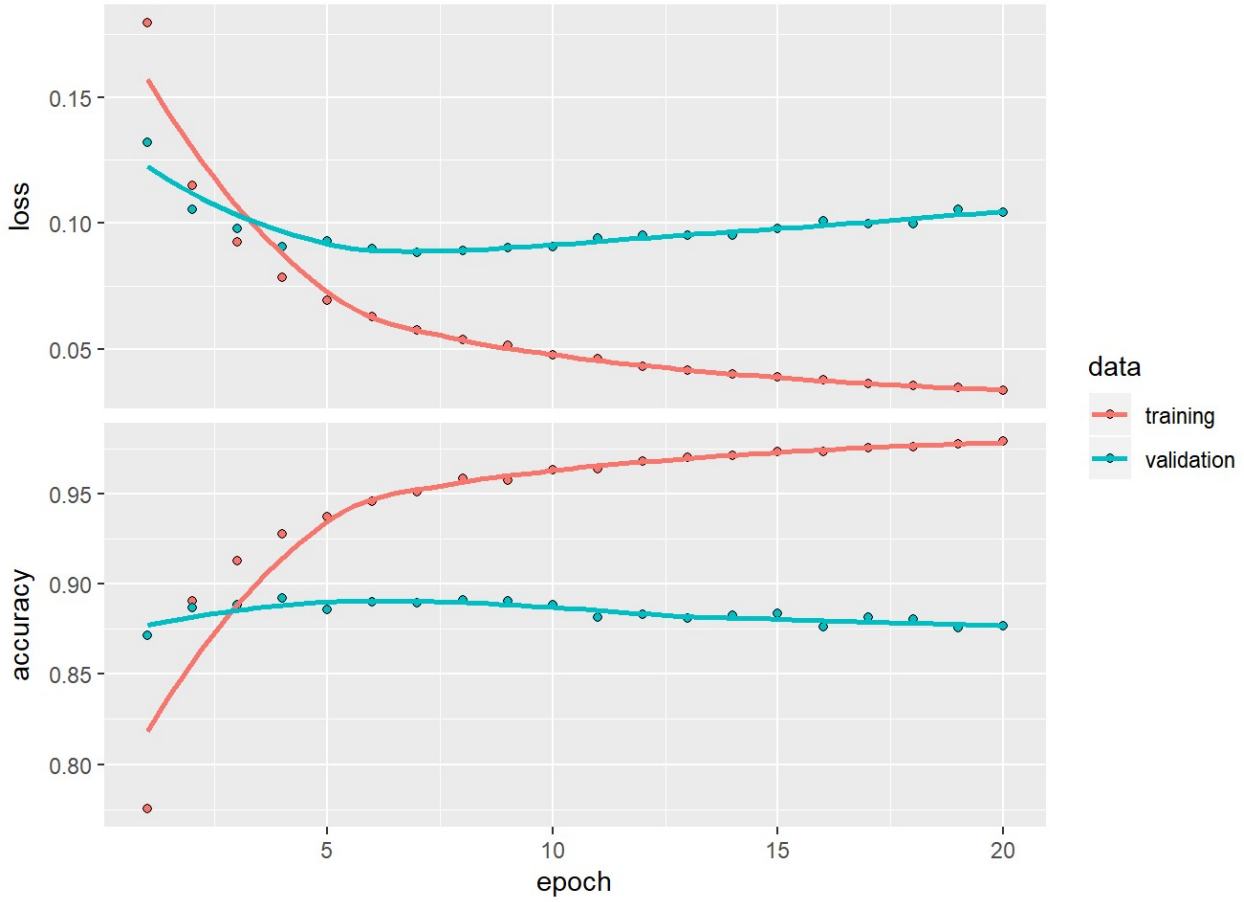
1- Hidden layer network with regularization and drop out

1-hidden layer network with 16 units tanh activation mse loss function regularization and dropout techniques

```
model <- keras_model_sequential() %>%
  layer_dense(units = 16,  kernel_regularizer = regularizer_l2(.0001),activation = "ta
nh", input_shape = c(10000)) %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 1, activation = "sigmoid")
model %>% compile(
  optimizer = "rmsprop",
  loss = "mse",
  metrics = c("accuracy")
)
history <- model %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
str(history)
```

```
## List of 2
##  $ params :List of 7
##   ..$ batch_size   : int 512
##   ..$ epochs       : int 20
##   ..$ steps        : num 30
##   ..$ samples      : int 15000
##   ..$ verbose      : int 0
##   ..$ do_validation: logi TRUE
##   ..$ metrics      : chr [1:4] "loss" "accuracy" "val_loss" "val_accuracy"
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 0.1793 0.1147 0.0926 0.0784 0.0694 ...
##   ..$ accuracy    : num [1:20] 0.775 0.891 0.913 0.927 0.937 ...
##   ..$ val_loss    : num [1:20] 0.132 0.1055 0.0978 0.0905 0.0928 ...
##   ..$ val_accuracy: num [1:20] 0.871 0.887 0.888 0.892 0.886 ...
##  - attr(*, "class")= chr "keras_training_history"
```

```
plot(history)
```




```
model %>% fit(x_train, y_train, epochs = 5, batch_size = 512)
results <- model %>% evaluate(x_test, y_test)
results
```

```
## $loss
## [1] 0.1062581
##
## $accuracy
## [1] 0.87224
```

```
model %>% predict(x_test[1:10,])
```

```
##             [,1]
##  [1,] 0.017152965
##  [2,] 0.999990523
##  [3,] 0.831163049
##  [4,] 0.881715298
##  [5,] 0.989483058
##  [6,] 0.834367096
##  [7,] 0.999946475
##  [8,] 0.000816077
##  [9,] 0.982040226
## [10,] 0.999282897
```

This network begins to overfit after five epochs

Here the validation Accuracy is 87% and loss of 13%. The result seems good. We can try to improve the model by using more hidden layers.

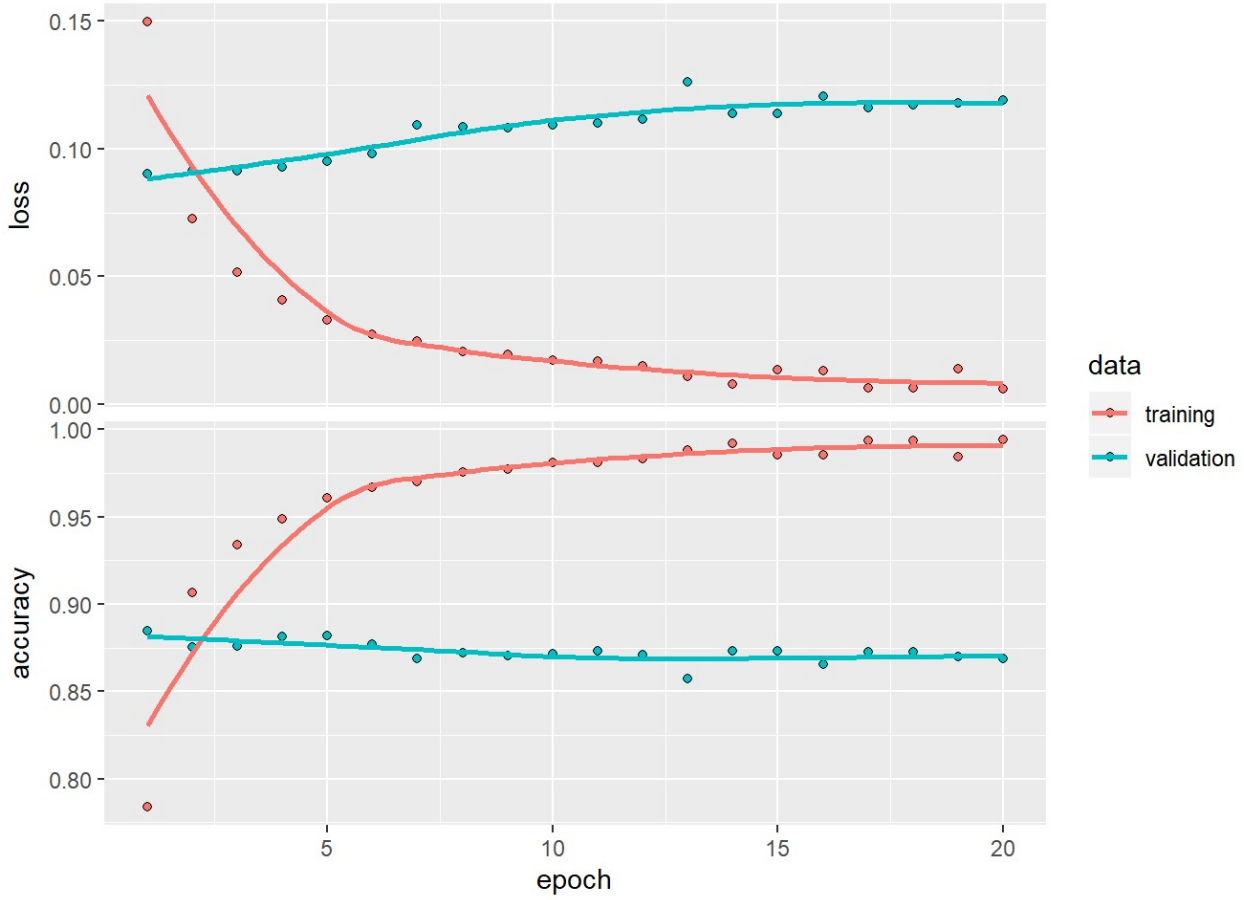
3- Hidden layer network without regualrization and dropout

3-hidden layer network with 64, 32 and 16 units tanh activation mse loss function

```
model <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "tanh", input_shape = c(10000)) %>%
  layer_dense(units = 32, activation = "tanh") %>%
  layer_dense(units = 16, activation = "tanh") %>%
  layer_dense(units = 1, activation = "sigmoid")
model %>% compile(
  optimizer = "rmsprop",
  loss = "mse",
  metrics = c("accuracy")
)
history <- model %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
str(history)
```

```
## List of 2
##  $ params :List of 7
##   ..$ batch_size   : int 512
##   ..$ epochs       : int 20
##   ..$ steps        : num 30
##   ..$ samples      : int 15000
##   ..$ verbose      : int 0
##   ..$ do_validation: logi TRUE
##   ..$ metrics      : chr [1:4] "loss" "accuracy" "val_loss" "val_accuracy"
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 0.1495 0.0725 0.0517 0.041 0.0329 ...
##   ..$ accuracy    : num [1:20] 0.784 0.906 0.934 0.949 0.961 ...
##   ..$ val_loss    : num [1:20] 0.0901 0.0912 0.0913 0.093 0.0949 ...
##   ..$ val_accuracy: num [1:20] 0.885 0.875 0.876 0.881 0.882 ...
##  - attr(*, "class")= chr "keras_training_history"
```

```
plot(history)
```




```
model %>% fit(x_train, y_train, epochs = 2, batch_size = 512)
results <- model %>% evaluate(x_test, y_test)
results
```

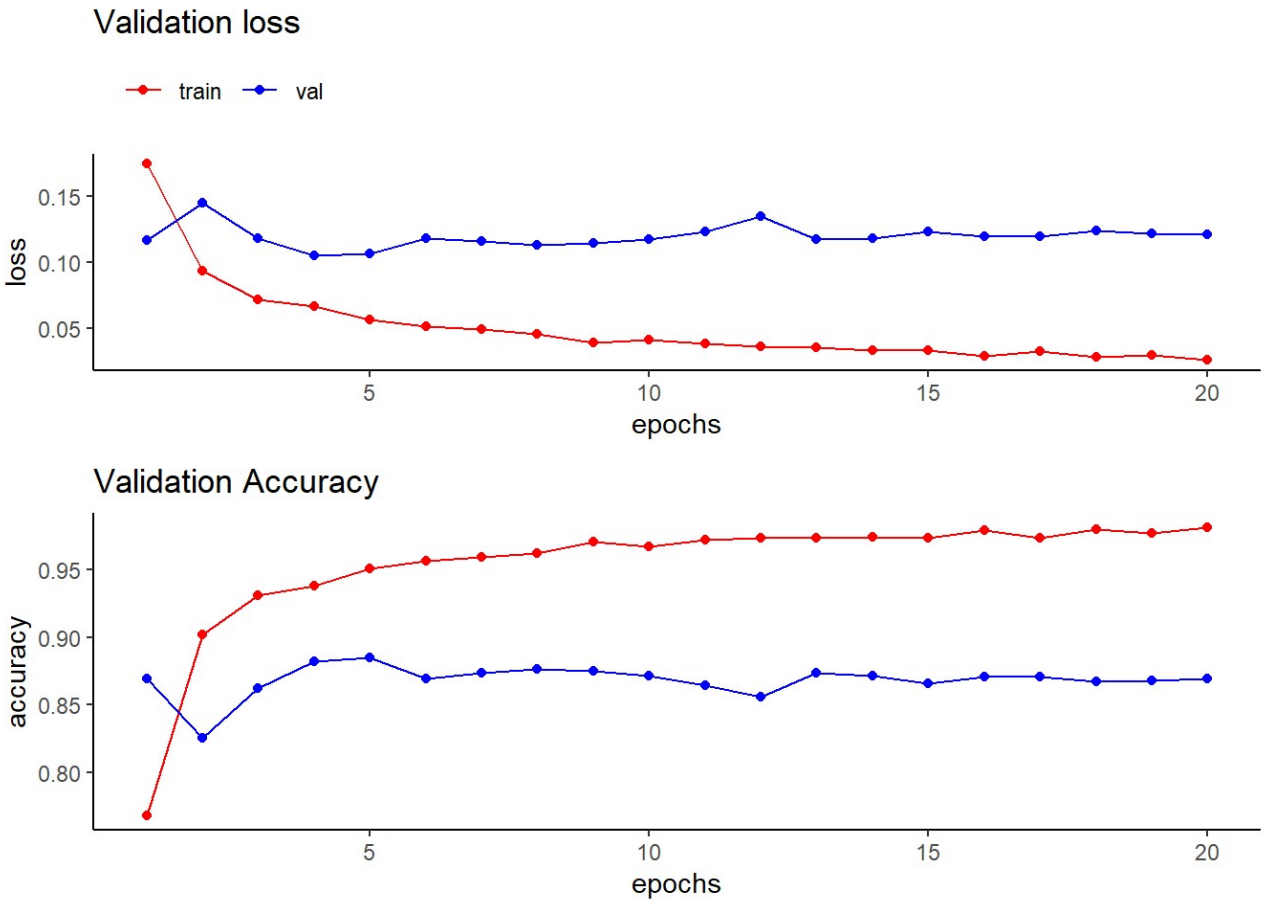
```
## $loss
## [1] 0.1132613
##
## $accuracy
## [1] 0.85636
```

```
model %>% predict(x_test[1:10,])
```

```
##              [,1]
##  [1,] 0.050998688
##  [2,] 0.990540326
##  [3,] 0.988667548
##  [4,] 0.930186510
##  [5,] 0.987824738
##  [6,] 0.935481727
##  [7,] 0.990623951
##  [8,] 0.009410083
##  [9,] 0.989233017
## [10,] 0.989485979
```

This network begins to overfit after two epochs
Here the validation Accuracy is 86% and loss of 11%. The result seems good. We can try to improve the model by using regularization with the same hidden layers.
3-layer network with regualrization
3-hidden layer network with 64, 32 and 16 units
tanh activation
mse loss function
regularization technique

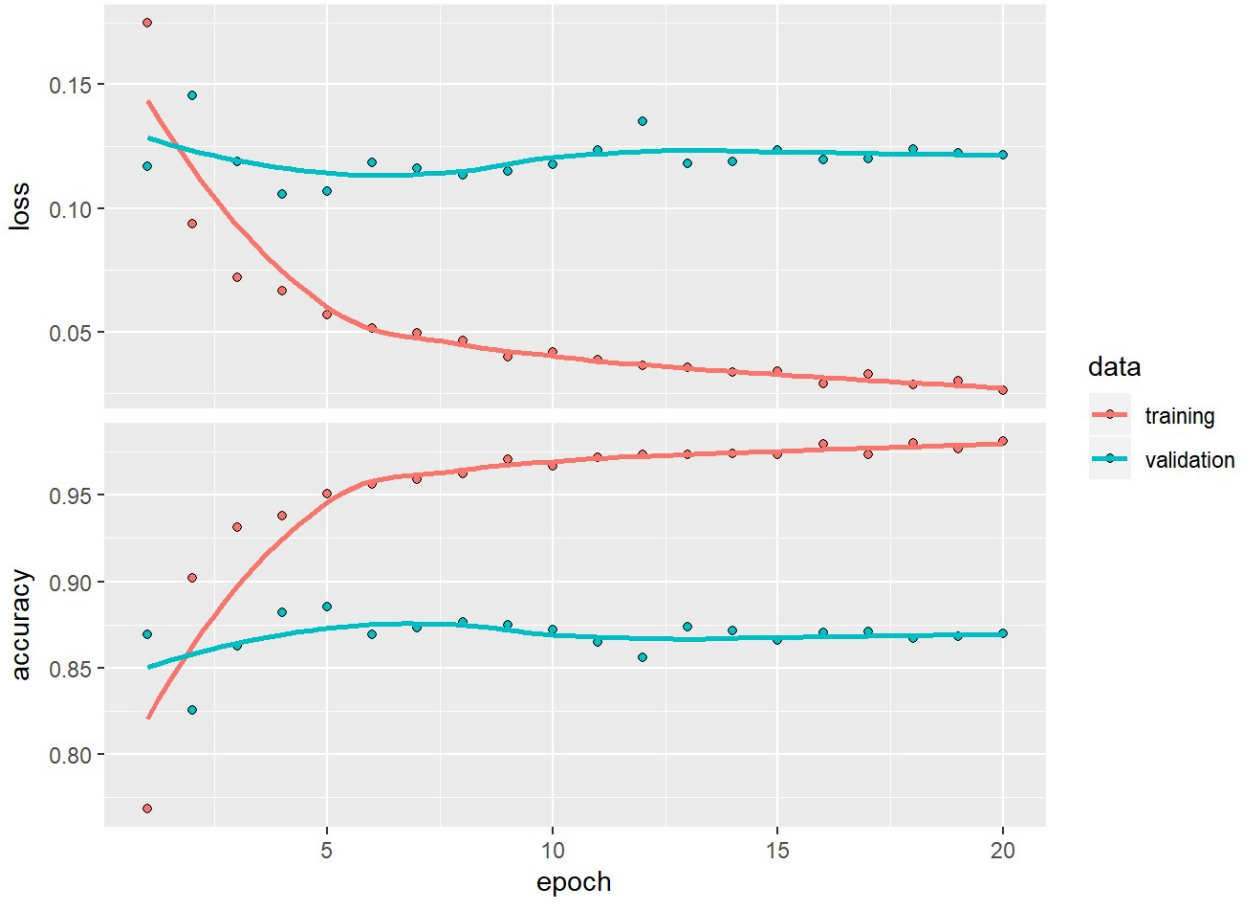
```r
model <- keras_model_sequential() %>%
  layer_dense(units = 64, kernel_regularizer = regularizer_l2(.0001), activation = "tanh", input_shape = c(10000)) %>%
  layer_dense(units = 32, kernel_regularizer = regularizer_l2(.0001), activation = "tanh") %>%
  layer_dense(units = 16, kernel_regularizer = regularizer_l2(.0001), activation = "tanh") %>%
  layer_dense(units = 1, activation = "sigmoid")
model %>% compile(
  optimizer = "rmsprop",
  loss = "mse",
  metrics = c("accuracy")
)
history <- model %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
hist1 <- as.data.frame(history$metrics)
names(hist1) <- c("train-loss","train-accuracy","val_loss","val_accuracy")
hist1 <- hist1 %>% mutate(epochs = 1:n()) %>% gather("split","values",-epochs) %>% separate(split,c("split","metric")) %>% spread(metric,values)
g1<- ggplot(hist1,aes(x=epochs,y=loss,color=split)) + geom_point() + geom_line() + theme_classic() + ggtitle("Validation loss") + theme(legend.position = "top",legend.justification = "left",legend.title = element_blank()) +scale_color_manual(values = c("red","blue"))
g2<- ggplot(hist1,aes(x=epochs,y=accuracy,color=split)) + geom_point(show.legend = F) + geom_line(show.legend = F) + theme_classic() + ggtitle("Validation Accuracy") + theme(legend.position = "top",legend.justification = "left",legend.title = element_blank()) + scale_color_manual(values = c("red","blue"))
plot_grid(g1,g2,nrow=2)
```




```
str(history)
```

```
## List of 2
##  $ params :List of 7
##   ..$ batch_size   : int 512
##   ..$ epochs       : int 20
##   ..$ steps        : num 30
##   ..$ samples      : int 15000
##   ..$ verbose      : int 0
##   ..$ do_validation: logi TRUE
##   ..$ metrics      : chr [1:4] "loss" "accuracy" "val_loss" "val_accuracy"
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 0.1748 0.0935 0.0718 0.0667 0.057 ...
##   ..$ accuracy    : num [1:20] 0.768 0.902 0.931 0.938 0.951 ...
##   ..$ val_loss    : num [1:20] 0.117 0.146 0.119 0.105 0.107 ...
##   ..$ val_accuracy: num [1:20] 0.869 0.826 0.863 0.882 0.885 ...
##  - attr(*, "class")= chr "keras_training_history"
```

```
plot(history)
```




```
model %>% fit(x_train, y_train, epochs = 3, batch_size = 512)
results <- model %>% evaluate(x_test, y_test)
results
```

```
## $loss
## [1] 0.1170135
##
## $accuracy
## [1] 0.8674
```

```
model %>% predict(x_test[1:10,])
```

```
##             [,1]
##  [1,] 0.019235015
##  [2,] 0.997906446
##  [3,] 0.977522314
##  [4,] 0.974916816
##  [5,] 0.989317298
##  [6,] 0.942523301
##  [7,] 0.997870624
##  [8,] 0.006730497
##  [9,] 0.985813320
## [10,] 0.997355223
```

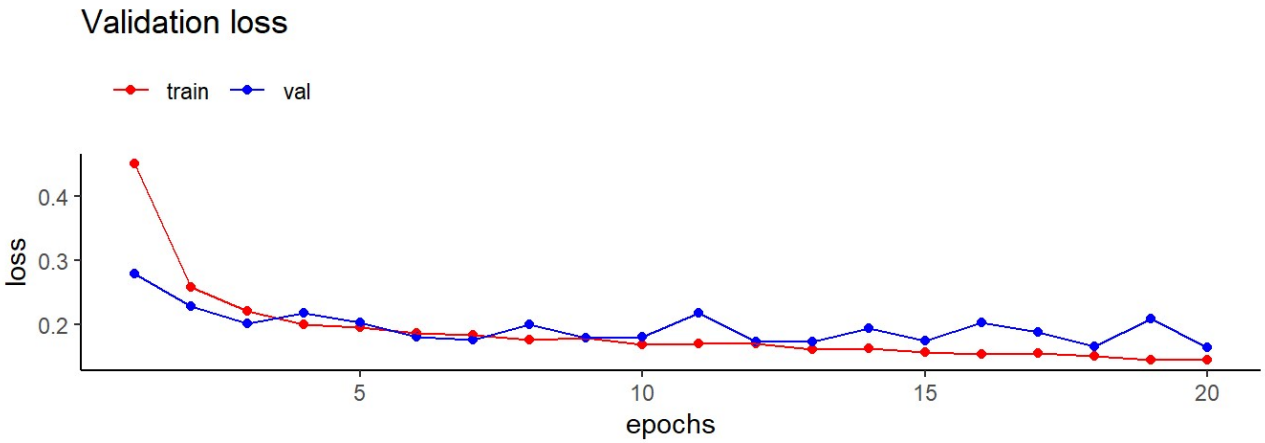
This network begins to overfit after one epoch

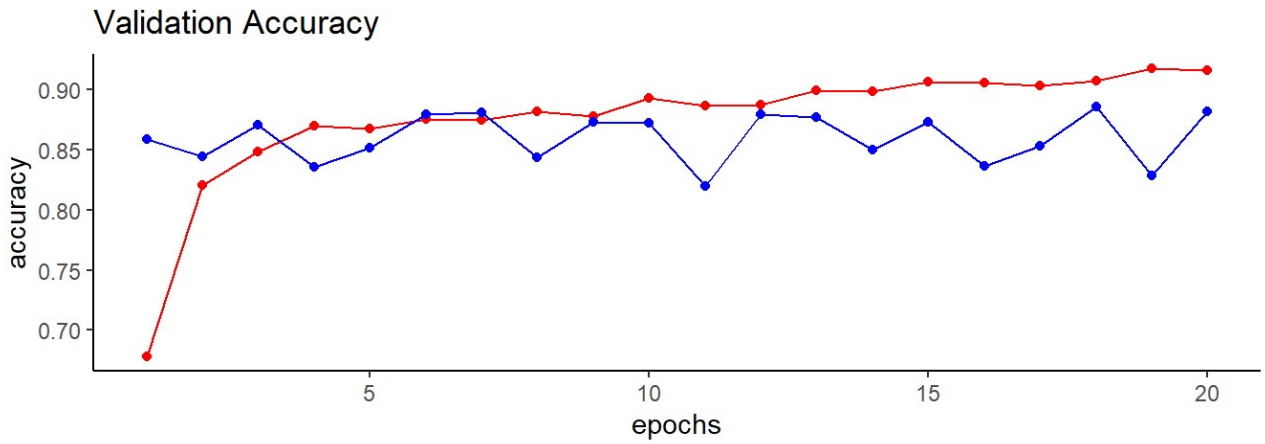
Here the validation Accuracy is 85% and loss of 13%. The result seems good. But the accuracy is less when compared to the previous one. We can try to improve the model by using both regularization and dropout.

3-layer network with regualrization

3-hidden layer network with 64, 32 and 16 units relu activation mse loss function regularization technique

```
model <- keras_model_sequential() %>%
  layer_dense(units = 64, kernel_regularizer = regularizer_l1(.0001), activation = "re
lu", input_shape = c(10000)) %>%
  layer_dense(units = 32, kernel_regularizer = regularizer_l1(.0001), activation = "re
lu") %>%
  layer_dense(units = 16, kernel_regularizer = regularizer_l1(.0001), activation = "re
lu") %>%
  layer_dense(units = 1, activation = "sigmoid")
model %>% compile(
  optimizer = "rmsprop",
  loss = "mse",
  metrics = c("accuracy")
)
history <- model %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
hist1 <- as.data.frame(history$metrics)
names(hist1) <- c("train-loss","train-accuracy","val_loss","val_accuracy")
hist1 <- hist1 %>% mutate(epochs = 1:n()) %>% gather("split","values",-epochs) %>% sep
arate(split,c("split","metric")) %>% spread(metric,values)
g1<- ggplot(hist1,aes(x=epochs,y=loss,color=split)) + geom_point() + geom_line() + the
me_classic() + ggtitle("Validation loss") + theme(legend.position = "top",legend.justi
fication = "left",legend.title = element_blank()) +scale_color_manual(values = c("re
d","blue"))
g2<- ggplot(hist1,aes(x=epochs,y=accuracy,color=split)) + geom_point(show.legend = F)
+ geom_line(show.legend = F) + theme_classic() + ggtitle("Validation Accuracy") + them
e(legend.position = "top",legend.justification = "left",legend.title = element_blank
()) + scale_color_manual(values = c("red","blue"))
plot_grid(g1,g2,nrow=2)
```

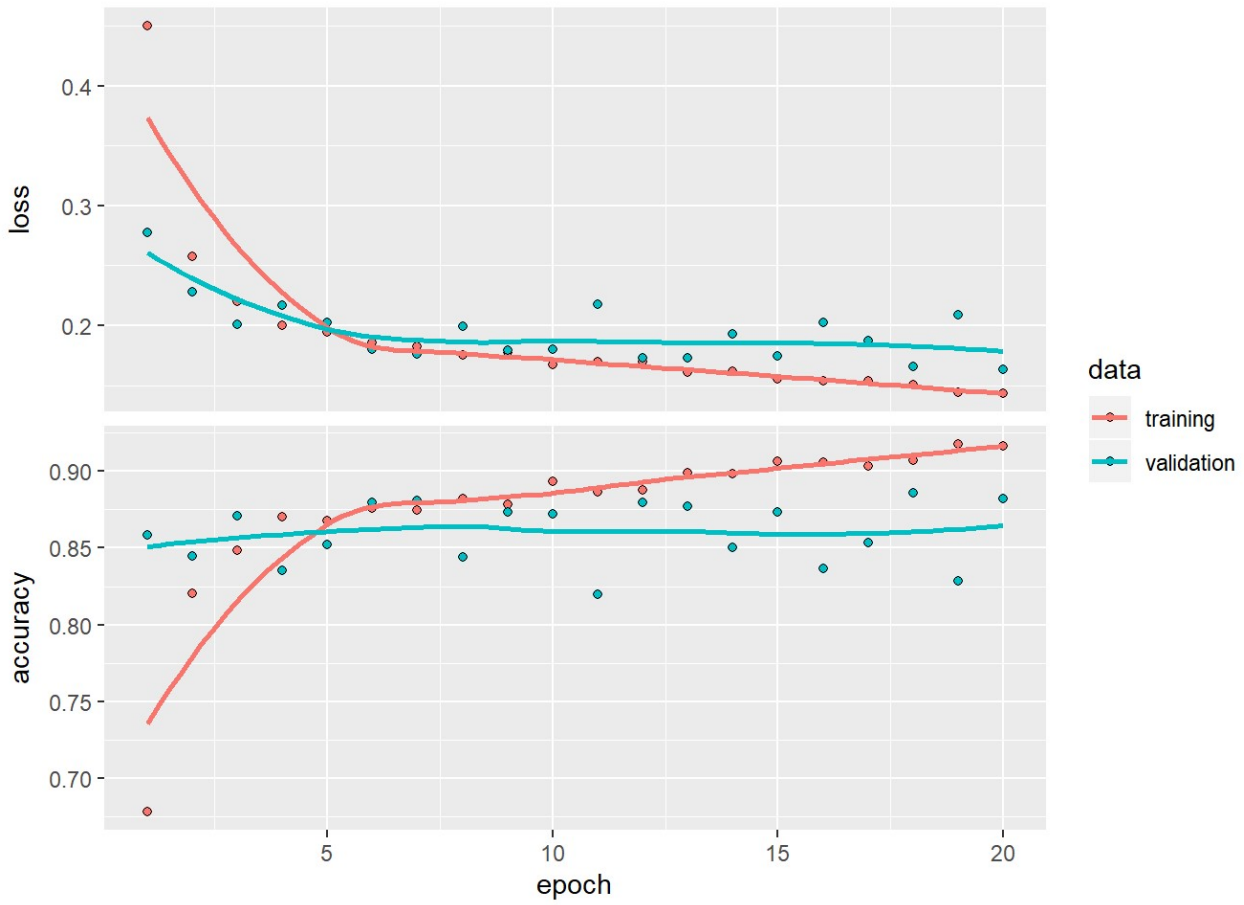





```
str(history)
```

```
## List of 2
##  $ params :List of 7
##   ..$ batch_size   : int 512
##   ..$ epochs       : int 20
##   ..$ steps        : num 30
##   ..$ samples      : int 15000
##   ..$ verbose      : int 0
##   ..$ do_validation: logi TRUE
##   ..$ metrics      : chr [1:4] "loss" "accuracy" "val_loss" "val_accuracy"
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 0.45 0.258 0.22 0.2 0.194 ...
##   ..$ accuracy    : num [1:20] 0.678 0.82 0.848 0.87 0.868 ...
##   ..$ val_loss    : num [1:20] 0.278 0.228 0.201 0.217 0.202 ...
##   ..$ val_accuracy: num [1:20] 0.859 0.844 0.871 0.835 0.852 ...
##  - attr(*, "class")= chr "keras_training_history"
```

```
plot(history)
```




```
model %>% fit(x_train, y_train, epochs = 9, batch_size = 512)
results <- model %>% evaluate(x_test, y_test)
results
```

```
## $loss
## [1] 0.1677572
##
## $accuracy
## [1] 0.86888
```

```
model %>% predict(x_test[1:10,])
```

```
##             [,1]
##  [1,] 0.10935858
##  [2,] 0.99951720
##  [3,] 0.99208450
##  [4,] 0.97979951
##  [5,] 0.97504914
##  [6,] 0.97740805
##  [7,] 0.99546695
##  [8,] 0.04269981
##  [9,] 0.97577697
## [10,] 0.99539113
```

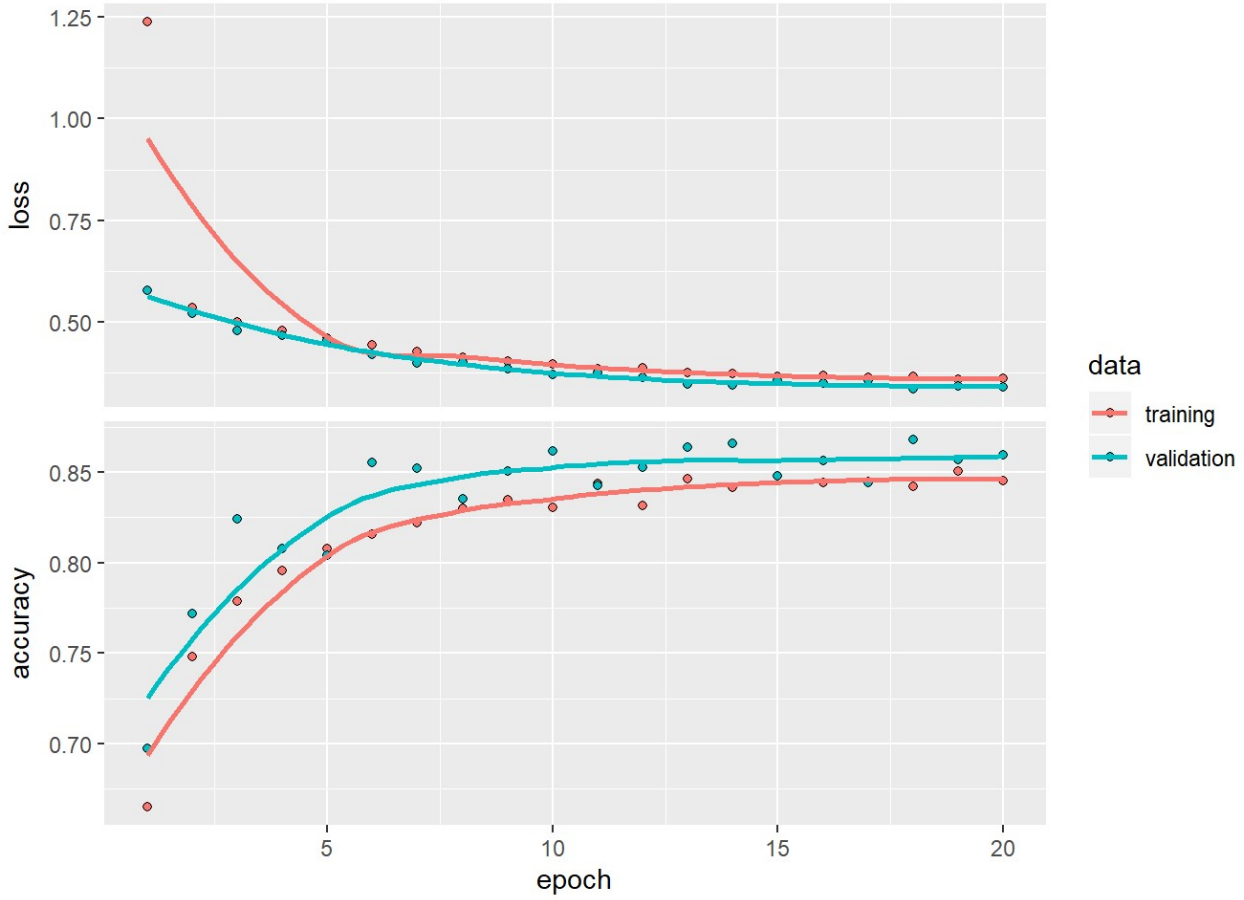
3-Hidden layer network with regualrization and dropout

3-hidden layer network with 64, 32 and 16 units tanh activation mse loss function regularization and dropout

```
model <- keras_model_sequential() %>%
  layer_dense(units = 32, kernel_regularizer = regularizer_l1(.001), activation = "tan
h", input_shape = c(10000)) %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 16, kernel_regularizer = regularizer_l1(.001), activation = "tan
h") %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 16, kernel_regularizer = regularizer_l1(.001), activation = "tan
h") %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 1, activation = "sigmoid")
model %>% compile(
  optimizer = "rmsprop",
  loss = "mse",
  metrics = c("accuracy")
)
history <- model %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
str(history)
```

```
## List of 2
##  $ params :List of 7
##   ..$ batch_size   : int 512
##   ..$ epochs       : int 20
##   ..$ steps        : num 30
##   ..$ samples      : int 15000
##   ..$ verbose      : int 0
##   ..$ do_validation: logi TRUE
##   ..$ metrics      : chr [1:4] "loss" "accuracy" "val_loss" "val_accuracy"
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 1.238 0.536 0.499 0.479 0.46 ...
##   ..$ accuracy    : num [1:20] 0.665 0.748 0.779 0.796 0.808 ...
##   ..$ val_loss    : num [1:20] 0.577 0.522 0.48 0.467 0.456 ...
##   ..$ val_accuracy: num [1:20] 0.697 0.772 0.824 0.808 0.804 ...
##  - attr(*, "class")= chr "keras_training_history"
```

```
plot(history)
```




```
model %>% fit(x_train, y_train, epochs = 8, batch_size = 512)
results <- model %>% evaluate(x_test, y_test)
results
```

```
## $loss
## [1] 0.3375593
##
## $accuracy
## [1] 0.85784
```

```
model %>% predict(x_test[1:10,])
```

```
##             [,1]
##  [1,] 0.34870070
##  [2,] 0.99767882
##  [3,] 0.92687941
##  [4,] 0.92731953
##  [5,] 0.94188291
##  [6,] 0.74238586
##  [7,] 0.97718638
##  [8,] 0.05360094
##  [9,] 0.94230551
## [10,] 0.93442941
```

This network begins to overfit after two epochs.

Here the validation Accuracy is 88% and loss of 1o%. The result seems pretty good.This is the best among all other models.
To prevent overfitting, stop training after one epoch. Let?s train a new network from scratch for two epochs and then evaluate it on the test data.

```
model <- keras_model_sequential() %>%
  layer_dense(units = 64, kernel_regularizer = regularizer_l1(.0001), activation = "ta
nh", input_shape = c(10000)) %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 32, kernel_regularizer = regularizer_l1(.0001), activation = "ta
nh") %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 16, kernel_regularizer = regularizer_l1(.0001), activation = "ta
nh") %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 1, activation = "sigmoid")
model %>% compile(
  optimizer = "rmsprop",
  loss = "mse",
  metrics = c("accuracy")
)
```

Our fairly naive approach achieves an accuracy of 88%.

Using a trained network to generate predictions on new data

After having trained a network, you will want to use it in a practical setting. You can generate the likelihood of reviews being positive by using the predict method:

```
model %>% fit(x_train, y_train, epochs = 8, batch_size = 512)
results <- model %>% evaluate(x_test, y_test)
results
```

```
## $loss
## [1] 0.1980056
##
## $accuracy
## [1] 0.87756
```

```
model %>% predict(x_test[1:10,])
```

```
##             [,1]
##  [1,] 0.13323021
##  [2,] 0.99577683
##  [3,] 0.95903200
##  [4,] 0.82515955
##  [5,] 0.97869259
##  [6,] 0.95580763
##  [7,] 0.99291402
##  [8,] 0.00904277
##  [9,] 0.98411775
## [10,] 0.97526997
```

As you can see, the network is very confident for some samples (0.99 or more, or 0.02 or less) but less confident for others.