Feed-Forward Neural Network using Keras and TensorFlow

Muhammad Tamjid Rahman

Contents

Loading R packages

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

1 Feed-Forward Neural Network using Keras and TensorFlow

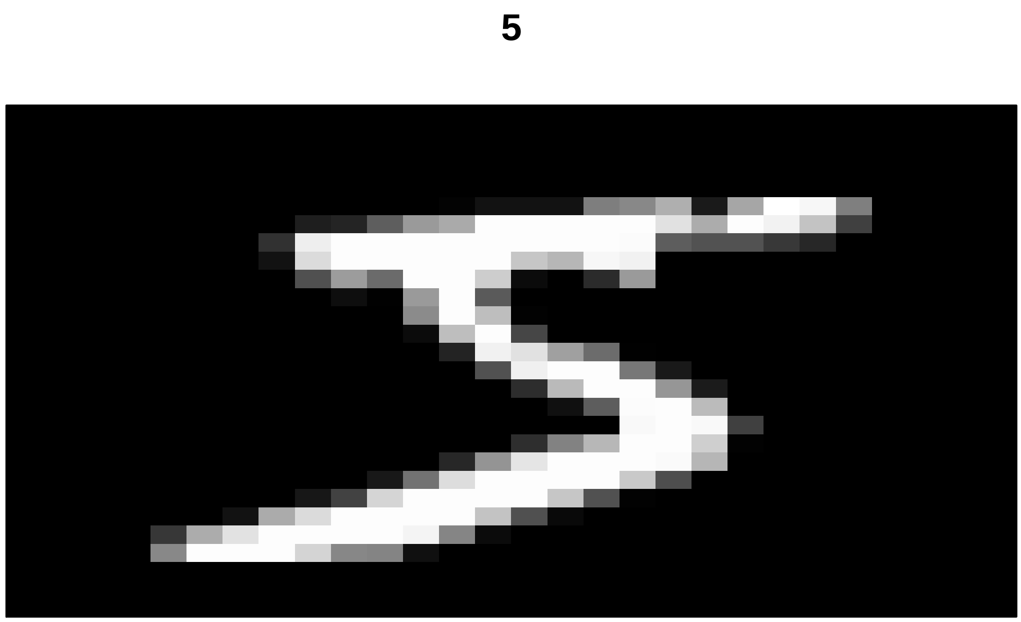
```
mnist <- dataset_mnist()
```

```
## Loaded Tensorflow version 2.7.0
```

```
mnist$train$x <- mnist$train$x/255
mnist$test$x <- mnist$test$x/255
```

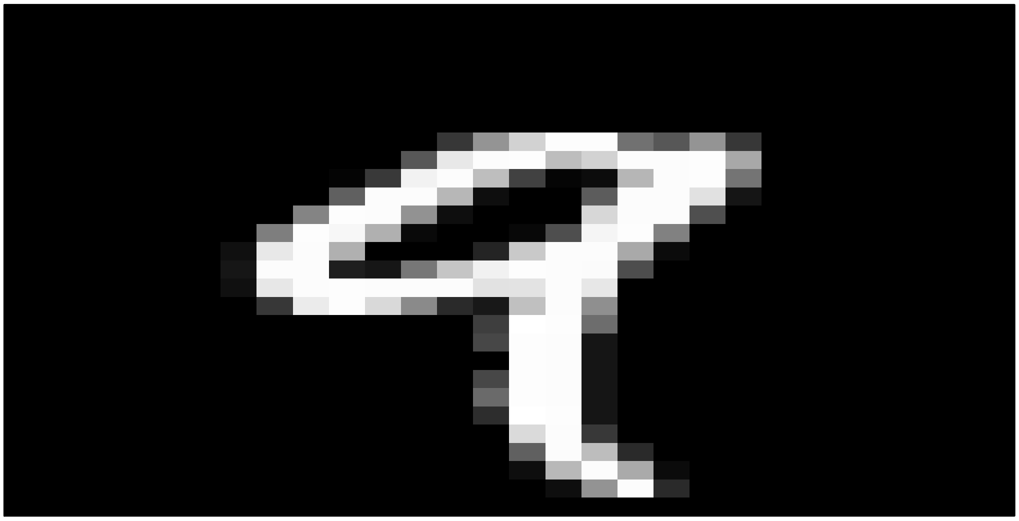
1 Visualizing digits

```
idx <- 1
im <- mnist$train$x[idx,,]
# Transpose the image
im <- t(apply(im, 2, rev))
image(1:28, 1:28, im, col=gray((0:255)/255), xlab = "", ylab = "",
xaxt='n', yaxt='n', main=paste(mnist$train$y[idx]))
```




```
idx <- 5
im <- mnist$train$x[idx,,]
# Transpose the image
im <- t(apply(im, 2, rev))
image(1:28, 1:28, im, col=gray((0:255)/255), xlab = "", ylab = "",
xaxt='n', yaxt='n', main=paste(mnist$train$y[idx]))
```

9



2 Training and test set size

Traing set

```
dim(as.data.frame( mnist$train))
```

```
## [1] 60000   785
```

```
object.size(mnist$train)
```

```
## 376560792 bytes
```

The traning set has 60000 observations with 785 variables and it is 376560792 bytes.

Test set

```
dim(as.data.frame( mnist$test))
```

```
## [1] 10000   785
```

```
object.size(mnist$test)
```

```
## 62760792 bytes
```

The test set has 10000 observations with 785 variables and it is 62760792 bytes.

3 Implementing a feed-forward neural network

a

```
model <- keras_model_sequential() %>%
    layer_flatten(input_shape = c(28, 28)) %>%
    layer_dense(units = 16, activation = "sigmoid") %>%
    layer_dense(10, activation = "softmax")

model %>%
    compile(
        loss = "sparse_categorical_crossentropy",
        optimizer = "adam",
        metrics = "accuracy"
    )

model %>%
    fit(
        x = mnist$train$x, y = mnist$train$y,
        epochs = 5,
        validation_split = 0.3,
        verbose = 2
    )
```

b

```
summary(model)
```

```
## Model: "sequential"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  flatten (Flatten)                  (None, 784)                     0
##
##  dense_1 (Dense)                    (None, 16)                      12560
##
##  dense (Dense)                      (None, 10)                      170
##
## ================================================================================
## Total params: 12,730
## Trainable params: 12,730
## Non-trainable params: 0
## ________________________________________________________________________________
```

There are total 12,730 parameters.

c

Layer(type) flatten_5 (Flatten) is the input layer and it has no parameter.

d

Layer(type) dense_10 (Dense) is the output layer and it has 170 parameters.

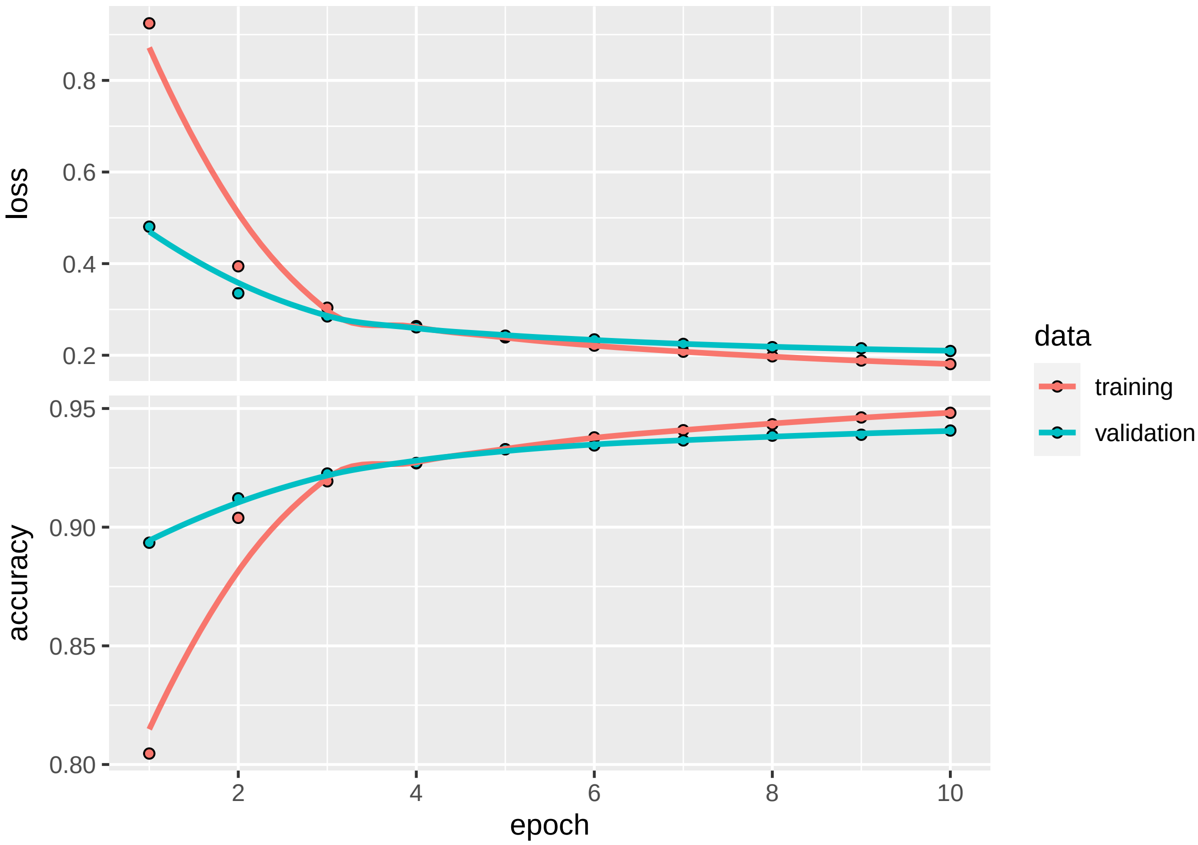
e

```
model %>%
    evaluate(mnist$test$x, mnist$test$y)#, verbose = 0)
```

```
##      loss  accuracy
## 0.2455706 0.9287000
```

The classification accuracy is 93%.

```
## `geom_smooth()` using formula 'y ~ x'
```



Confusion Matrix

```
pred<- model %>%
    predict((mnist$test$x))%>% k_argmax()
cm<-table(Predicted=as.numeric(pred), Actual=as.numeric(mnist$test$y))
cm
```

```
##          Actual
## Predicted    0    1    2    3    4    5    6    7    8    9
##         0  959    0   12    2    1   12   11    3    5   10
##         1    0 1108    1    2    1    4    3    9    5    6
##         2    0    4  950   14    8    2    8   22    5    1
##         3    2    0    6  923    0   22    1    7   10   10
##         4    1    0   11    1  928    9    8    8    5   26
##         5    8    1    1   17    0  802    9    0   12    4
```

```
##         6    6    3    7    1    5   11  912    0    6    0
##         7    2    2   10   16    4    4    1  958    6    9
##         8    2   17   31   26    5   21    5    1  912    9
##         9    0    0    3    8   30    5    0   20    8  934
```

```
FN=function(mat){
  mat_low_i=list()
  for (i in 1:(nrow(mat)-1)) {

    mat_low_i[i]=list(mat[row(mat) == (col(mat) + i)])
  }

  mat_low_i

  FN1=c()
  for (i in 1:(nrow(mat)-1)) {

    FN1[i]=sum(mat_low_i[[i]])
  }

  p=sum(FN1)

  return(FN=p)
}

FP=function(mat){
  mat_low_i=list()
  for (i in 1:(nrow(mat)-1)) {

    mat_low_i[i]=list(mat[row(mat) == (col(mat) - i)])
  }

  mat_low_i

  FP1=c()
  for (j in 1:(nrow(mat)-1)) {

    FP1[j]=sum(mat_low_i[[j]])
  }

  p=sum(FP1)

  return(FP=p)
}
```

```
FP=FP(cm)
FP
```

```
## [1] 308
```

```
FN=FN(cm)
FN
```

```
## [1] 306
```

```r
TPTN=sum(diag(cm))
TPTN
```

```
## [1] 9386
```

```r
accuracy=(TPTN)/(TPTN+FP+FN)
accuracy
```

```
## [1] 0.9386
```

```r
precision=TPTN/(TPTN+FP)
precision
```

```
## [1] 0.9682278
```

```r
recall=TPTN/(TPTN+FN)
recall
```

```
## [1] 0.9684276
```

Accuracy	Precision	Recall
93%	96%	97%

4

a

```r
model <- keras_model_sequential() %>%
    layer_flatten(input_shape = c(28, 28)) %>%
    layer_dense(units = 128, activation = "sigmoid") %>%
    layer_dense(10, activation = "softmax")
summary(model)
```

```
## Model: "sequential_2"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                      Param #
## ================================================================================
##  flatten_2 (Flatten)                (None, 784)                       0
##
##  dense_5 (Dense)                    (None, 128)                       100480
##
##  dense_4 (Dense)                    (None, 10)                        1290
##
## ================================================================================
## Total params: 101,770
## Trainable params: 101,770
## Non-trainable params: 0
## ________________________________________________________________________________
```

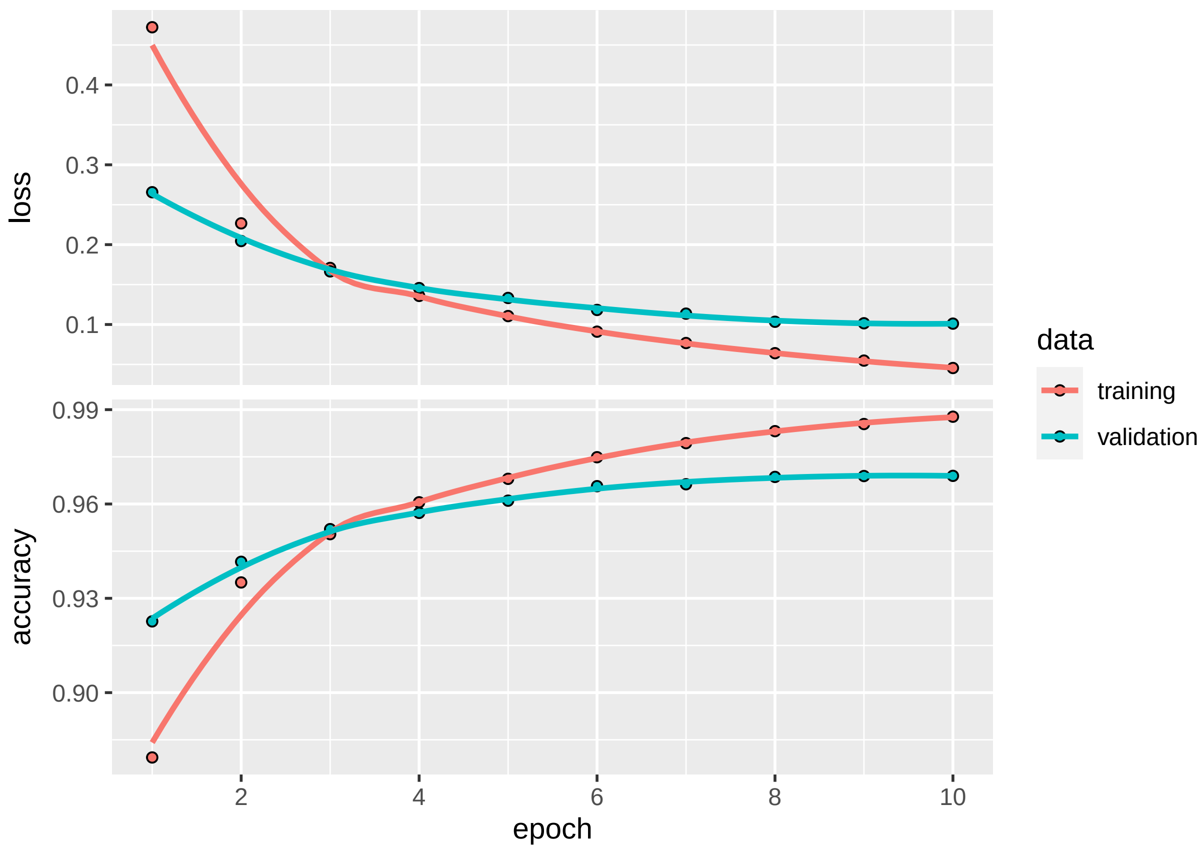
```r
model %>%
    compile(
        loss = "sparse_categorical_crossentropy",
        optimizer = "adam",
        metrics = "accuracy"
    )
```

```
history<-model %>%
    fit(
        x = mnist$train$x, y = mnist$train$y,
        epochs = 10,
        validation_split = 0.3,
        verbose = 2
    )

model %>%
    evaluate(mnist$test$x, mnist$test$y, verbose = 0)
```

```
##      loss  accuracy
## 0.0919233 0.9725000
```

```
plot(history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



b

```
model <- keras_model_sequential() %>%
    layer_flatten(input_shape = c(28, 28)) %>%
    layer_dense(units = 128, activation = "relu") %>%
    layer_dense(10, activation = "softmax")
summary(model)
```

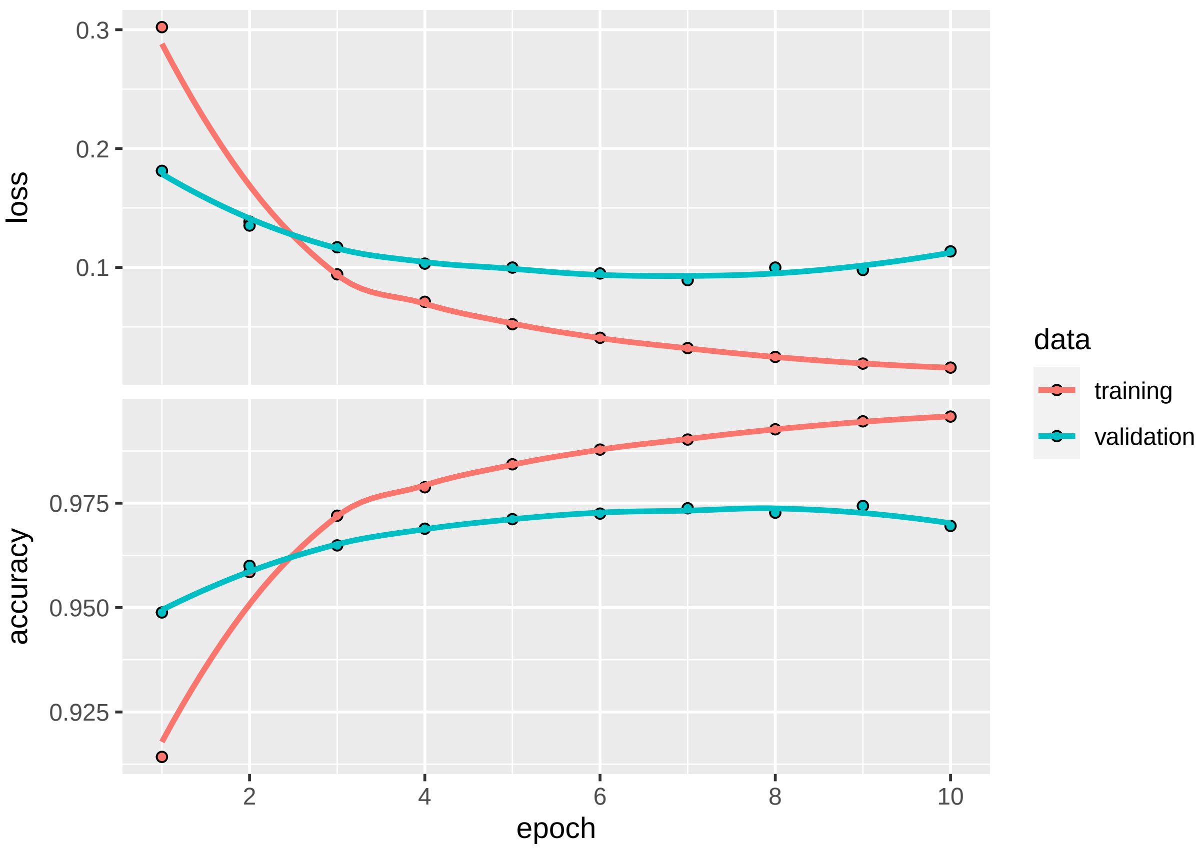
```
## Model: "sequential_3"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  flatten_3 (Flatten)                (None, 784)                     0
##
##  dense_7 (Dense)                    (None, 128)                     100480
##
##  dense_6 (Dense)                    (None, 10)                      1290
##
## ================================================================================
## Total params: 101,770
## Trainable params: 101,770
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
model %>%
    compile(
        loss = "sparse_categorical_crossentropy",
        optimizer = "adam",
        metrics = "accuracy"
    )

history<-model %>%
    fit(
        x = mnist$train$x, y = mnist$train$y,
        epochs = 10,
        validation_split = 0.3,
        verbose = 2
    )

model %>%
    evaluate(mnist$test$x, mnist$test$y, verbose = 0)
```

```
##      loss  accuracy
## 0.0899054 0.9753000
```

```
plot(history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



c

```
model <- keras_model_sequential() %>%
    layer_flatten(input_shape = c(28, 28)) %>%
    layer_dense(units = 128, activation = "relu") %>%
    layer_dense(10, activation = "softmax")
summary(model)
```

```
## Model: "sequential_4"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  flatten_4 (Flatten)                (None, 784)                     0
##
##  dense_9 (Dense)                    (None, 128)                     100480
##
##  dense_8 (Dense)                    (None, 10)                      1290
##
## ================================================================================
## Total params: 101,770
## Trainable params: 101,770
## Non-trainable params: 0
## ________________________________________________________________________________
```

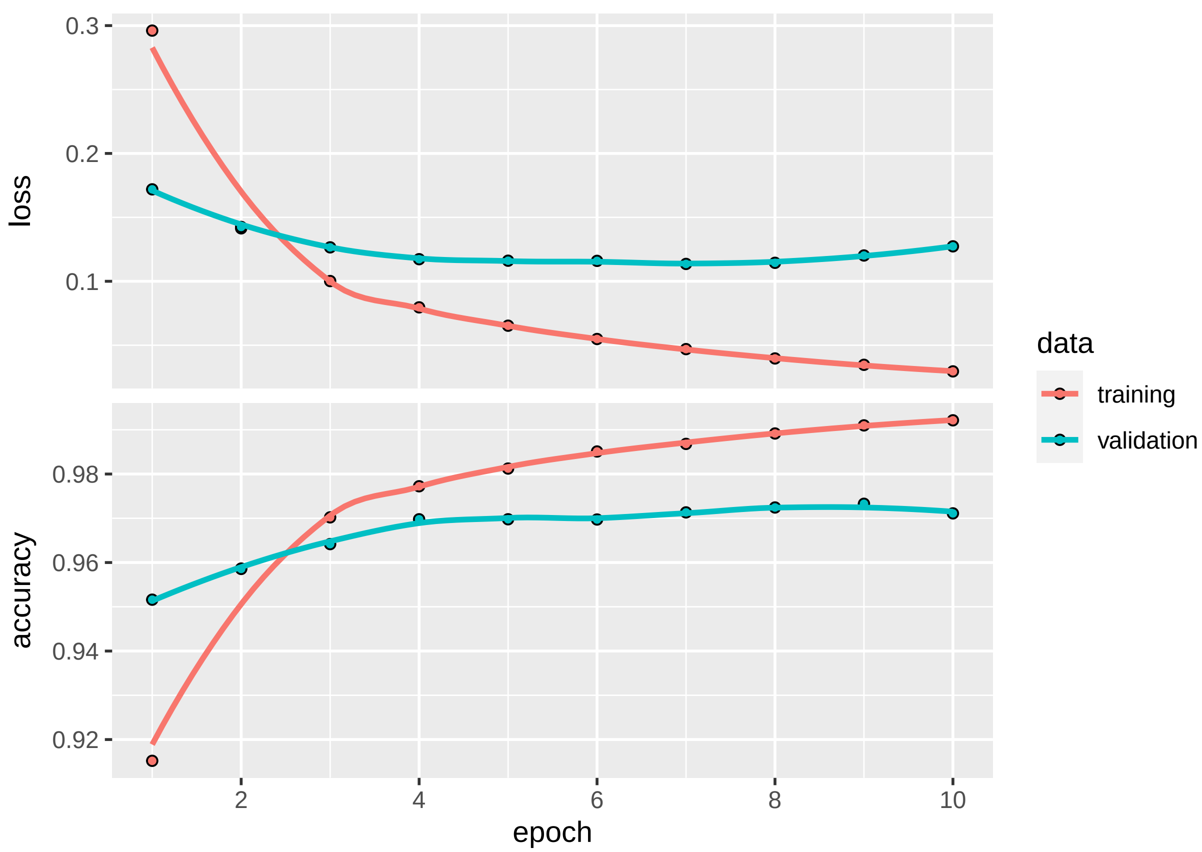
```
model %>%
    compile(
```

```
        loss = "sparse_categorical_crossentropy",
        optimizer = "RMSprop",
        metrics = "accuracy"
    )

history<-model %>%
    fit(
        x = mnist$train$x, y = mnist$train$y,
        epochs = 10,
        validation_split = 0.3,
        verbose = 2
    )

model %>%
    evaluate(mnist$test$x, mnist$test$y, verbose = 0)
```

```
##     loss accuracy
## 0.115161 0.975300
```

```
plot(history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



d

```
model <- keras_model_sequential() %>%
    layer_flatten(input_shape = c(28, 28)) %>%
    layer_dense(units = 128, activation = "relu") %>%
```

```r
    layer_dense(10, activation = "softmax")
summary(model)
```

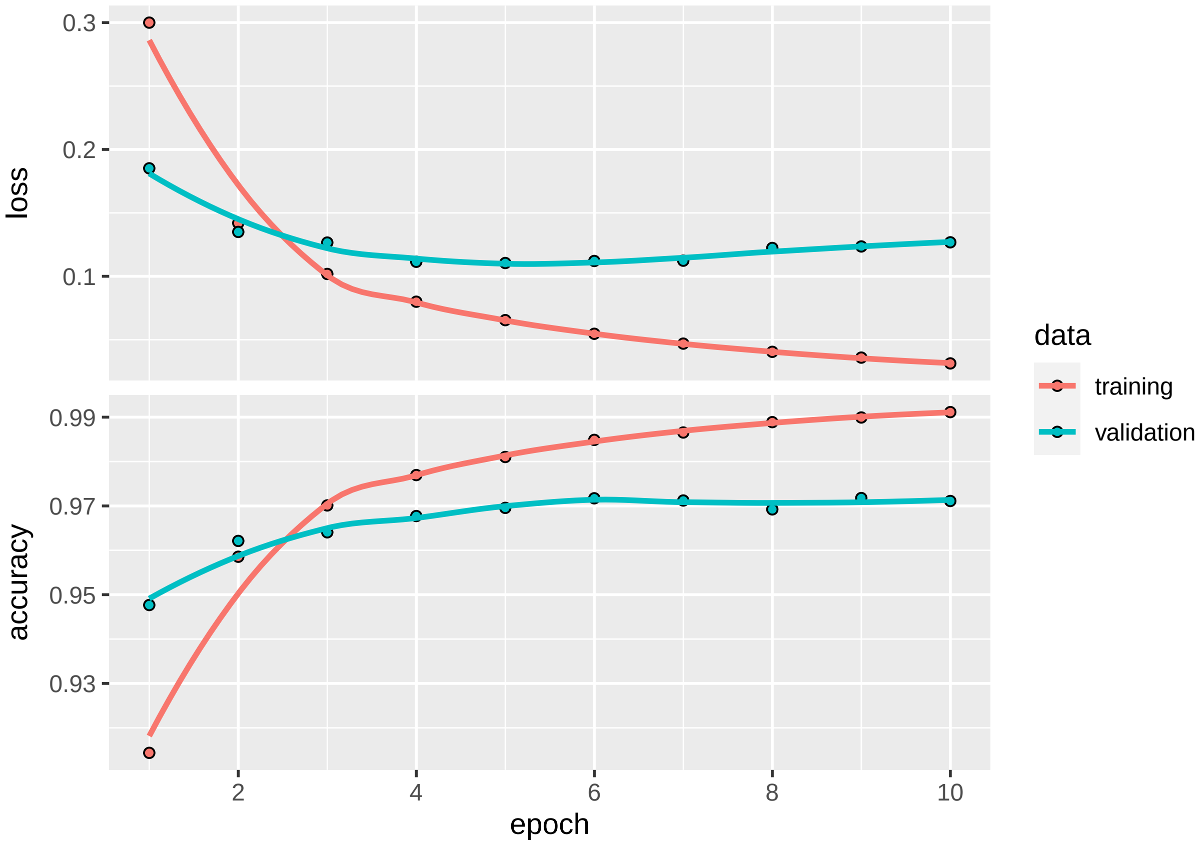
```
## Model: "sequential_5"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                      Param #
## ================================================================================
##  flatten_5 (Flatten)                (None, 784)                       0
##
##  dense_11 (Dense)                   (None, 128)                       100480
##
##  dense_10 (Dense)                   (None, 10)                        1290
##
## ================================================================================
## Total params: 101,770
## Trainable params: 101,770
## Non-trainable params: 0
## ________________________________________________________________________________
```

```r
model %>%
    compile(
        loss = "sparse_categorical_crossentropy",
        optimizer = "RMSprop",
        metrics = "accuracy"
    )

history<-model %>%
    fit(
        x = mnist$train$x, y = mnist$train$y,
        epochs = 10,
        validation_split = 0.3,
        verbose = 2
    )

model %>%
    evaluate(mnist$test$x, mnist$test$y, verbose = 0)
```

```
##      loss  accuracy
## 0.1051387 0.9755000
```

```r
plot(history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



For early stopping regularization I would use kerasR built in function 'EarlyStopping' to avoid being overfitted. This stop training if the improvement(accuracy, loss) of the model for validation has stopped.

e

```
model <- keras_model_sequential() %>%
    layer_flatten(input_shape = c(28, 28)) %>%
    layer_dense(units = 128, activation = "relu") %>%
    layer_dense(units = 128, activation = "relu") %>% #2nd layer
    layer_dense(10, activation = "softmax")
summary(model)
```

```
## Model: "sequential_6"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  flatten_6 (Flatten)                (None, 784)                     0
##
##  dense_14 (Dense)                   (None, 128)                     100480
##
##  dense_13 (Dense)                   (None, 128)                     16512
##
##  dense_12 (Dense)                   (None, 10)                      1290
##
## ================================================================================
## Total params: 118,282
## Trainable params: 118,282
```

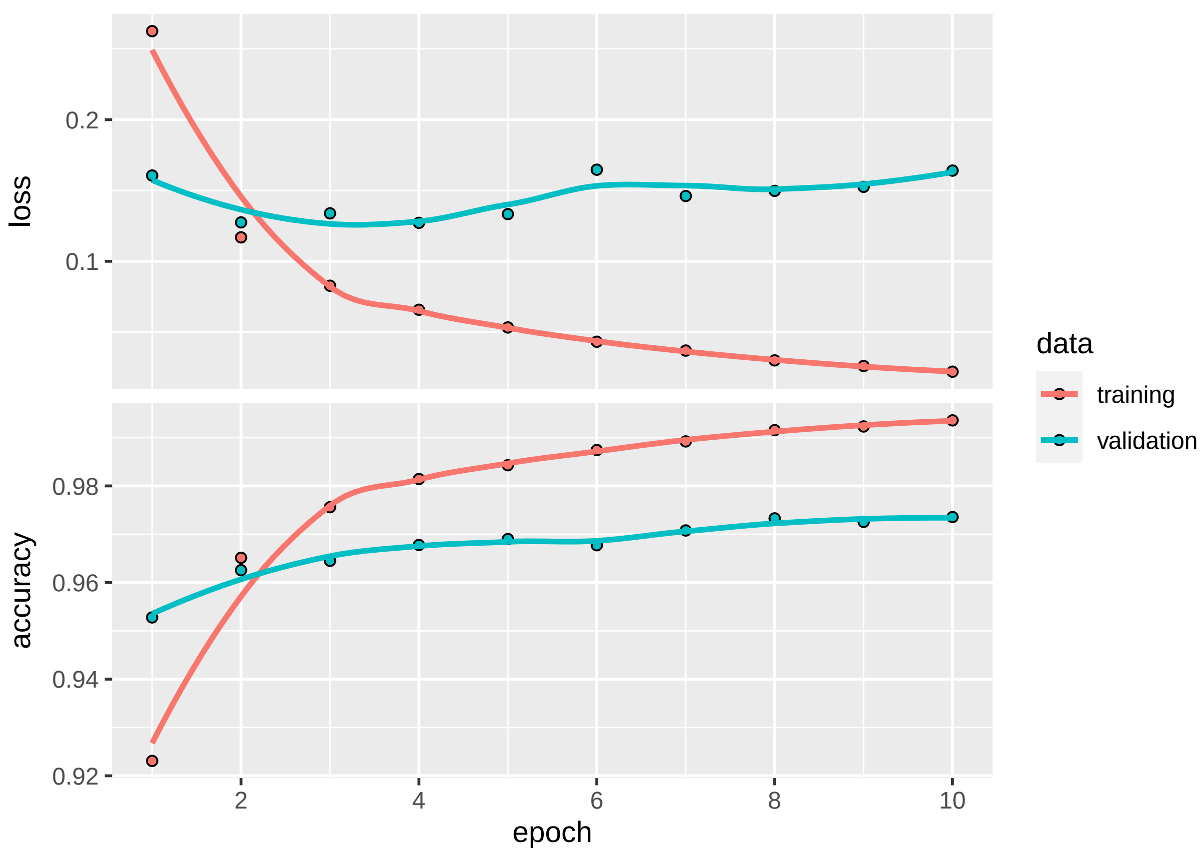
```
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
model %>%
    compile(
        loss = "sparse_categorical_crossentropy",
        optimizer = "RMSprop",
        metrics = "accuracy"
    )

history<-model %>%
    fit(
        x = mnist$train$x, y = mnist$train$y,
        epochs = 10,
        validation_split = 0.3,
        verbose = 2
    )

model %>%
    evaluate(mnist$test$x, mnist$test$y, verbose = 0)
```

```
##      loss  accuracy
## 0.1210637 0.9785000
```

```
plot(history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



There are total 118,282 parameters in the model.

f

```
model <- keras_model_sequential() %>%
    layer_flatten(input_shape = c(28, 28)) %>%
    layer_dense(units = 128, activation = "relu") %>%
    layer_dropout(0.2) %>%
    layer_dense(units = 128, activation = "relu") %>%
    layer_dropout(0.5) %>%
    layer_dense(10, activation = "softmax")
summary(model)
```

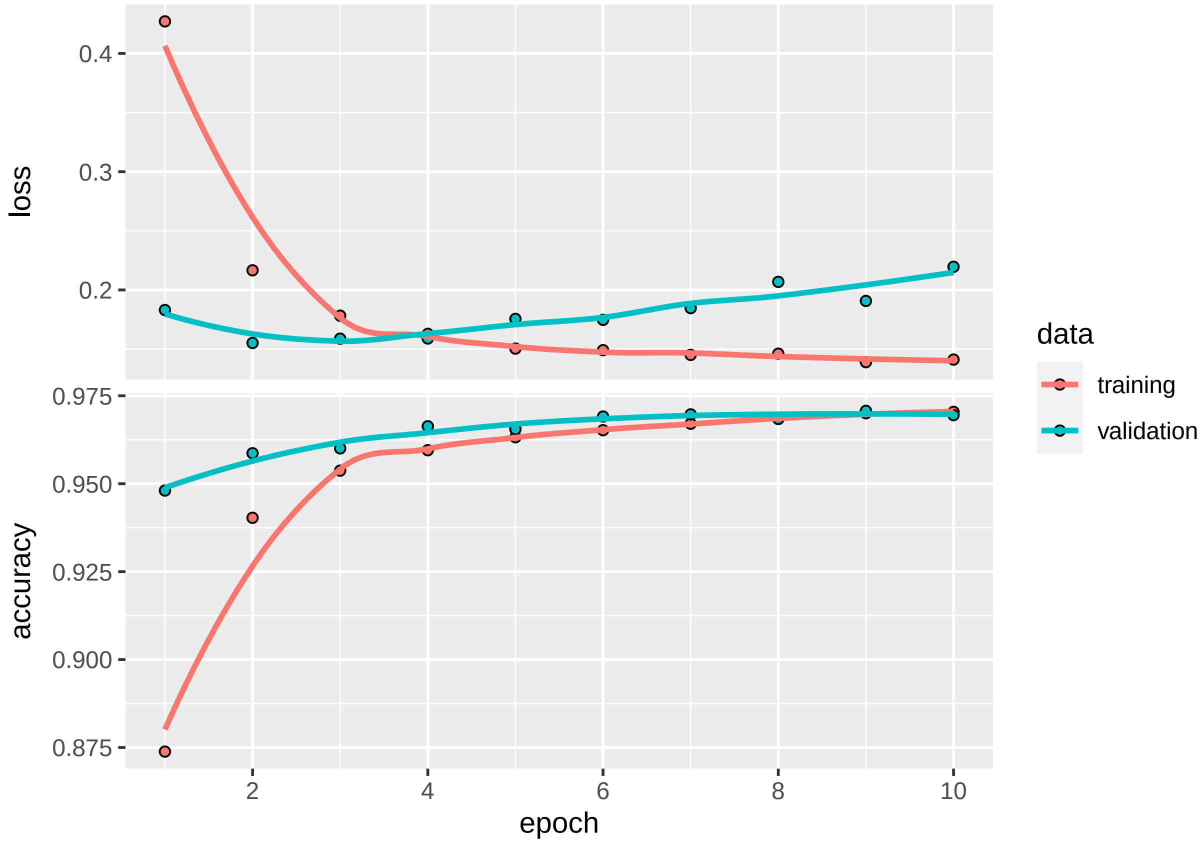
```
## Model: "sequential_7"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  flatten_7 (Flatten)                (None, 784)                     0
##
##  dense_17 (Dense)                   (None, 128)                     100480
##
##  dropout_1 (Dropout)                (None, 128)                     0
##
##  dense_16 (Dense)                   (None, 128)                     16512
##
##  dropout (Dropout)                  (None, 128)                     0
##
##  dense_15 (Dense)                   (None, 10)                      1290
##
## ================================================================================
## Total params: 118,282
## Trainable params: 118,282
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
model %>%
    compile(
        loss = "sparse_categorical_crossentropy",
        optimizer = "RMSprop",
        metrics = "accuracy"
    )

history<-model %>%
    fit(
        x = mnist$train$x, y = mnist$train$y,
        epochs = 10,
        validation_split = 0.3,
        verbose = 2
    )

model %>%
    evaluate(mnist$test$x, mnist$test$y, verbose = 0)
```

```
##      loss  accuracy
## 0.1797494 0.9724000
```

```r
plot(history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



g

```r
model <- keras_model_sequential() %>%
    layer_flatten(input_shape = c(28, 28)) %>%
    layer_dense(units = 128, activation = "relu" ) %>%
    layer_batch_normalization() %>%

    layer_dense(units = 128, activation = "relu") %>%
    layer_batch_normalization() %>%

    layer_dense(10, activation = "softmax")
summary(model)
```

```
## Model: "sequential_8"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                     Param #
## ================================================================================
##  flatten_8 (Flatten)                (None, 784)                      0
##
##  dense_20 (Dense)                   (None, 128)                      100480
##
##  batch_normalization_1 (BatchNormal  (None, 128)                     512
```

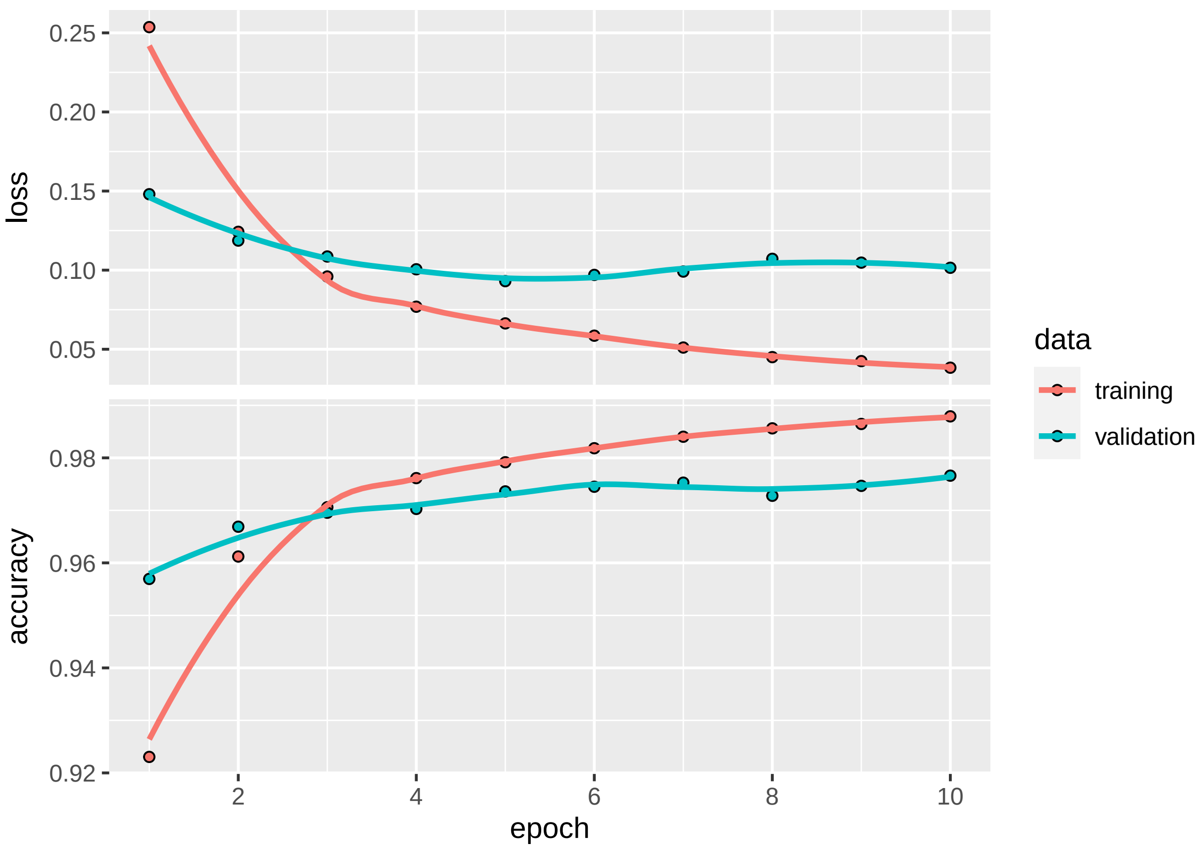
```
##  ization)
##
##  dense_19 (Dense)                   (None, 128)                     16512
##
##  batch_normalization (BatchNormaliz  (None, 128)                    512
##  ation)
##
##  dense_18 (Dense)                   (None, 10)                      1290
##
## ================================================================================
## Total params: 119,306
## Trainable params: 118,794
## Non-trainable params: 512
## ________________________________________________________________________________
```

```r
model %>%
    compile(
        loss = "sparse_categorical_crossentropy",
        optimizer = "RMSprop",
        metrics = "accuracy"
    )

history<-model %>%
    fit(
        x = mnist$train$x, y = mnist$train$y,
        epochs = 10,
        validation_split = 0.3,
        verbose = 2
    )

model %>%
    evaluate(mnist$test$x, mnist$test$y, verbose = 0)
```

```
##       loss   accuracy
## 0.09347585 0.97570002
```

```r
plot(history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



5 My model

```
model <- keras_model_sequential() %>%
    layer_flatten(input_shape = c(28, 28)) %>%
    layer_dropout(0.2) %>%
    layer_dense(units = 128, activation = "relu" ) %>%
    layer_dropout(0.5) %>%
    layer_batch_normalization() %>%
    layer_dense(units = 256, activation = "relu") %>%
    layer_batch_normalization() %>%
    layer_dense(units = 128, activation = "relu") %>%


    layer_dense(10, activation = "softmax")
summary(model)
```

```
## Model: "sequential_9"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                      Param #
## ================================================================================
##  flatten_9 (Flatten)                (None, 784)                       0
##
##  dropout_3 (Dropout)                (None, 784)                       0
##
##  dense_24 (Dense)                   (None, 128)                       100480
##
```

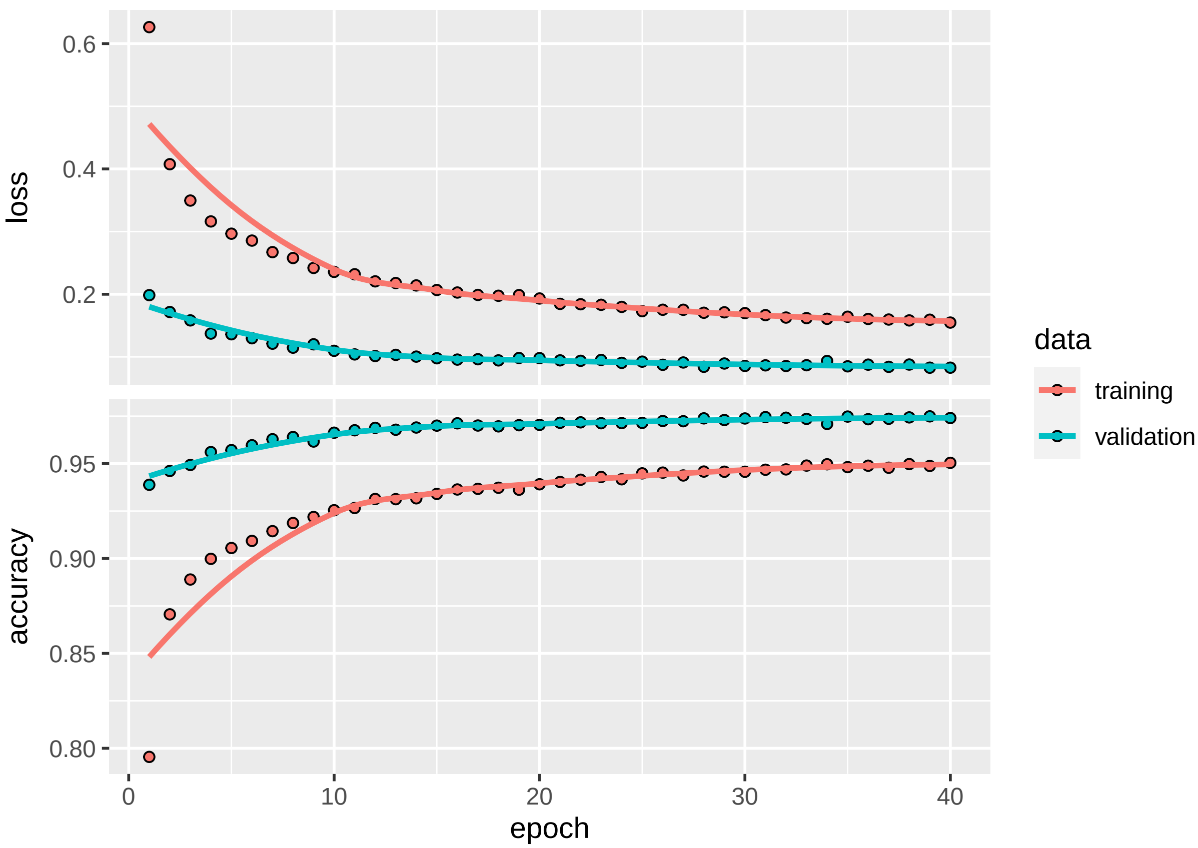
```
##  dropout_2 (Dropout)                (None, 128)                     0
##
##  batch_normalization_3 (BatchNormal  (None, 128)                    512
##  ization)
##
##  dense_23 (Dense)                   (None, 256)                     33024
##
##  batch_normalization_2 (BatchNormal  (None, 256)                    1024
##  ization)
##
##  dense_22 (Dense)                   (None, 128)                     32896
##
##  dense_21 (Dense)                   (None, 10)                      1290
##
## ================================================================================
## Total params: 169,226
## Trainable params: 168,458
## Non-trainable params: 768
## ________________________________________________________________________________
```

```
model %>%
    compile(
        loss = "sparse_categorical_crossentropy",
        optimizer = "adam",
        metrics = "accuracy"
    )

history<-model %>%
    fit(
        x = mnist$train$x, y = mnist$train$y,
        epochs = 40,
        validation_split = 0.3,
        verbose = 2
    )

model %>%
    evaluate(mnist$test$x, mnist$test$y, verbose = 0)
```

```
##       loss   accuracy
## 0.07251313 0.97970003
```

```
plot(history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



In my model, 3 hidden layer were used with the same activation function reLU. Batch normalized and dropout of layer were used in 2 hidden layers. After 30 epochs accuracy and loss seem to be stabled. The accuracy is 98%. Validation set accuracy is close to 95%. So, the model is well fitted, not overfitted.

6 Two digits that the network has classified incorrectly

```
library(tidyverse)
```

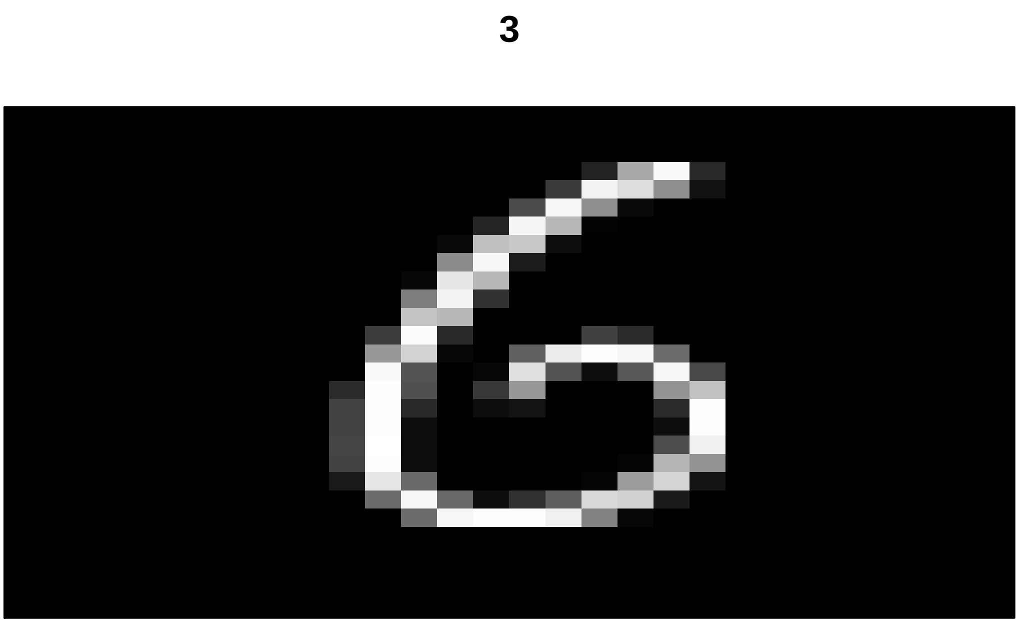
```
pred<- model %>%
    predict((mnist$test$x))%>% k_argmax()
```

```
df<-cbind(1:length(mnist$test$y),mnist$test$y,as.numeric(pred))

subset(df, df[,2] !=df[,3])[1:2,]
```

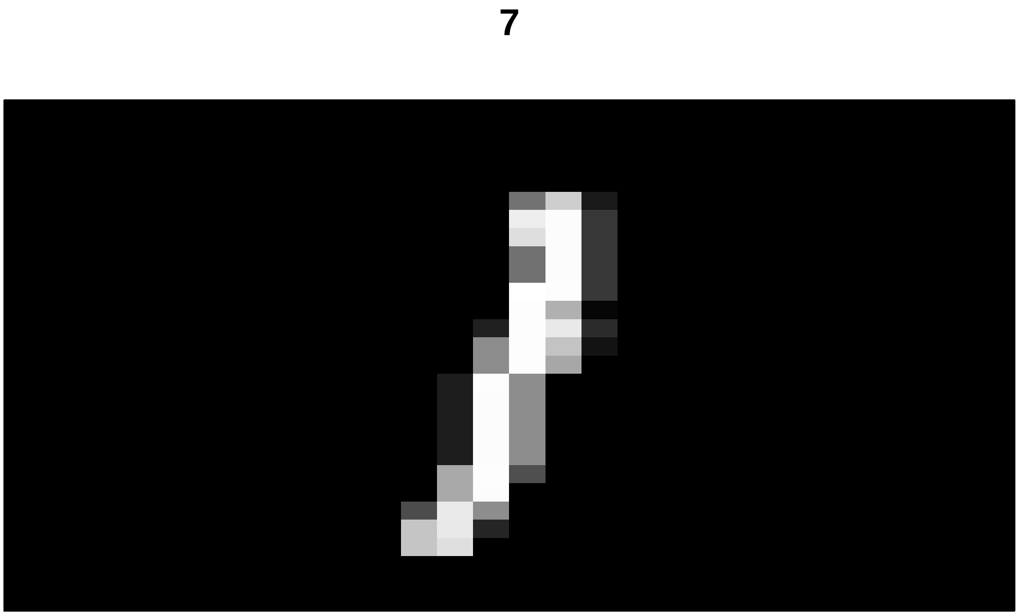
```
##      [,1] [,2] [,3]
## [1,]   19    3    8
## [2,]  152    9    8
```

Image with index 19, 125 were miss-classified. 19th image is 3 but predicted as 8 and 19th image is 7 but predicted as 4.

```
idx <- 19
im <- mnist$train$x[idx,,]
# Transpose the image
im <- t(apply(im, 2, rev))
image(1:28, 1:28, im, col=gray((0:255)/255), xlab = "", ylab = "",
xaxt='n', yaxt='n', main=paste(mnist$test$y[idx]))
```



```
idx <- 125
im <- mnist$train$x[idx,,]
# Transpose the image
im <- t(apply(im, 2, rev))
image(1:28, 1:28, im, col=gray((0:255)/255), xlab = "", ylab = "",
xaxt='n', yaxt='n', main=paste(mnist$test$y[idx]))
```



After visualizing we see that both 19th and 125th were labeling incorrectly. This can be one of the reasons of miss-classification.

7

Confusion Matrix

```
cm1<-table(Predicted=as.numeric(pred), Actual=as.numeric(mnist$test$y))
cm1
```

```
##          Actual
## Predicted    0    1    2    3    4    5    6    7    8    9
##         0  973    0    4    0    1    2    5    2    5    1
##         1    0 1132    3    1    0    0    2    8    1    5
##         2    0    0 1009    3    1    0    0    8    4    0
##         3    0    0    2  986    0    6    1    1    5    5
##         4    0    0    3    0  961    1    2    0    4    8
##         5    1    1    0    5    0  872    3    0    3    8
##         6    0    1    1    0    7    6  941    0    1    0
##         7    1    0    7    7    1    1    0 1003    4    4
##         8    2    1    3    5    2    3    4    0  945    3
##         9    3    0    0    3    9    1    0    6    2  975
```

```
FP=FP(cm1)
FP
```

```
## [1] 115
```

```r
FN=FN(cm1)
FN
```

```
## [1] 88
```

```r
TPTN=sum(diag(cm1))
TPTN
```

```
## [1] 9797
```

```r
accuracy=(TPTN)/(TPTN+FP+FN)
accuracy
```

```
## [1] 0.9797
```

```r
precision=TPTN/(TPTN+FP)
precision
```

```
## [1] 0.9883979
```

```r
recall=TPTN/(TPTN+FN)
recall
```

```
## [1] 0.9910976
```

Accuracy	Precision	Recall
98%	99%	99%

We gtt better accuracy than the validation set.

2 A simple neural network

A simple neural network function is going to be implemented.

```r
mini_net<- function(X, W, c, w, b){
w1 <- X %*% W + c
a1 <- matrix(NA,nrow(X),ncol(W ))
for(i in 1:nrow(X)){

  for(j in 1:ncol(W)){
    a1[i,j] <- max(0,w1[i,j]) # activation function
  }
}
return(y1=a1%*%w+b)
}
```

```r
W <- matrix(1, nrow = 2, ncol = 2)
c <- matrix(c(0, -1), ncol = 2, nrow = 4, byrow = TRUE)
X <- matrix(c(0,0,1,1,0,1,0,1), ncol = 2)
w <- matrix(c(1, -2), ncol = 1)
b <- 0
mini_net(X, W, c, w, b)
```

```
##      [,1]
## [1,]    0
## [2,]    1
```

```
## [3,]    1
## [4,]    0
```

```
##      [,1]
## [1,]  0.0
## [2,]  0.9
## [3,]  0.9
## [4,]  0.2
```

2

Changing the value W_{11} to 0.

```
W[1,1]=0
mini_net(X, W, c, w, b)
```

```
##      [,1]
## [1,]    0
## [2,]    1
## [3,]    0
## [4,]   -1
```

The above is the result we got.

3

For the neural network we used reLU as activation function. So, the output function is relu$(X\% * \%W + c)\% * \%w\% + b$. For simple problem this network might be reasonable but in reallity not. It's better to use more complex network. We could have used sigmoid instead of reLU.

4 Implementing a mean squared error loss function

```
W[1,1]=1
mini_net_loss=function(y, X, W, c, w, b){
  y1=mini_net(X, W, c, w, b)
  MSE=mean((y1-y)^2)
  return(MSE)
}
y <- c(0,1,1,0)
mini_net_loss(y, X, W, c, w, b)
```

```
## [1] 0
```

```
mini_net_loss(y, X, 0.9*W, c, w, b)
```

```
## [1] 0.015
```

5

```
W[1,1]=0
mini_net_loss(y, X, W, c, w, b)
```

```
## [1] 0.5
```

The value of the loss function is 0.5 .