Module 11: Solutions to Recommended Exercises

TMA4268 Statistical Learning V2022

Emma Skarstein, Daesoo Lee, Stefanie Muff
Department of Mathematical Sciences, NTNU

April 4, 2021

Problem 1

a)

It is a 4-4-4-3 feedforard neural network with an extra bias node in both the input and the two hidden layers. It can be written in the following form

$$y_c(\mathbf{x}) = \phi_o(\beta_{0c} + \sum_{m=1}^{4} \beta_{mc} z_m) = \phi_o(\beta_{0c} + \sum_{m=1}^{4} \beta_{mc} \phi_{h*}(\gamma_{0m} + \sum_{l=1}^{4} \gamma_{lm} \phi_h(\alpha_{0l} + \sum_{j=1}^{4} \alpha_{jl} x_j))).$$

b)

It is not clear wheter the network has 3 input nodes, or 2 input nodes plus one bias node (both would lead to the same representation). The hidden layer has 4 nodes, but no bias node, and the output layer consists of two nodes. This can be used for regression with two responses. If we have a classifiation problem with two classes then we usually use only one output node, but is is possible to use softmax activation for two classes, but that is very uncommon. Remember that for a binary outcome, we would usually only use one output node that encodes for the probability to be in one of the two classes.

c)

When the hidden layer has a linear activation the model is only linear in the original covariates, so adding the extra hidden layer will not add non-linearity to the model. The feedforward model may find latent structure in the data in the hidden layer. In general, however, we would then recommend to directly use logistic regression, because you then end up with a model that is easier to interpret.

d)

This is possible because the neural network is fitted using iterative methods. But, there is not one unique solutions here, and the network will benefit greatly by adding some sort of regulariztion, like weight decay and early stopping.

Problem 2

a)

This is a feedforward network with 10 input nodes plus a bias node, a hidden layer with 5 nodes plus a bias node, and a single node in the output layer. The hidden layer has a ReLU activiation function, whereas the output layer has a linear activation function.

The number of the estimated parameters are $(10+1)*5+(5+1)=61$.

b)

Feedforward network with two hidden layers. Input layer has 4 nodes and no bias term, the first hidden layer has 10 nodes and ReLU activation and a bias node, the second hidden layer has 5 nodes plus a bias node and ReLU activiation. One node in output layer with sigmoid activiation.

The number of estimated parameters are $4*10+(10+1)*5+(5+1)=101$.

c)

In module 7 we had an additive model of non-linear function, and interactions would be added manually (i.e., explicitly). Each coefficient estimated would be rather easy to interpret. For neural nets we know that with one hidden layer and squashing type activation we can fit any function (regression), but may need many nodes - and then the interpretation might not be so easy. Interactions are automatically handled with the non-linear function of sums.

Problem 3

```
library(ElemStatLearn)
train_data=zip.train[,-1]
train_labels=factor(zip.train[,1])
test_data=zip.test[,-1]
test_labels=factor(zip.test[,1])
mean <- apply(train_data, 2, mean)
std <- apply(train_data, 2, sd)
train_data <- scale(train_data, center = mean, scale = std)
test_data <- scale(test_data, center = mean, scale = std)
```

a)

b)

5 hidden nodes: $257*5+6*10$=1345 parameters

```
library(nnet)
zipnnet5<- nnet(train_labels~., data=train_data,size=5,MaxNWts=3000,maxit=5000)
summary(zipnnet5)
pred=predict(zipnnet5,newdata=test_data,type="class")
library(caret)
confusionMatrix(factor(pred),test_labels)
```

The above took some time to run, the results were:

```
> zipnnet5<- nnet(train_labels~., data=train_data,size=5,MaxNWts=3000,maxit=5000)
iter2960 value 864.566658
final  value 864.561810
converged
> summary(zipnnet5)
a 256-5-10 network with 1345 weights
options were - softmax modelling
   b->h1   i1->h1   i2->h1   i3->h1   i4->h1   i5->h1   i6->h1   i7->h1   i8->h1   i9->h1  i10->h1  i11-
  -49.27     9.15     1.24    21.03    -2.82    17.97     4.63    11.60    -4.31     2.28    -4.57    -
 i19->h1  i20->h1  i21->h1  i22->h1  i23->h1  i24->h1  i25->h1  i26->h1  i27->h1  i28->h1  i29->h1  i30-
> confusionMatrix(factor(pred),test_labels)
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2   3   4   5   6   7   8   9
         0 324   0   6   5   3   9   7   0   3   0
         1   1 245   7   0   1   0   0   0   7   5
         2   4   7 148   8  12   5  11   0   3   0
         3   2   0   6 128   4  10   0   4   5   3
         4   4   1   4   0 152   1   1   9   2   4
         5   1   0   2  18   1 117   5   0   7   1
         6  21   3  11   0   6   1 146   0   3   0
         7   0   1   6   4   5   1   0 122   9   5
         8   2   2   7   3   6  15   0   1 113   3
         9   0   5   1   0  10   1   0  11  14 156

Overall Statistics

               Accuracy : 0.8226
                 95% CI : (0.8052, 0.8391)
```

Problem 4: Deep Learning with Keras

a)

```
library(keras)
mnist = dataset_mnist()
x_train = mnist$train$x
y_train = mnist$train$y
x_test = mnist$test$x
y_test = mnist$test$y


# reshape
x_train = array_reshape(x_train, c(nrow(x_train), 28*28))
x_test = array_reshape(x_test, c(nrow(x_test), 28*28))
# rescale
```

```
x_train = x_train / 255
x_test = x_test / 255

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
dim(x_test)
```

```
## [1] 10000   784
```

```
model = keras_model_sequential() %>%
  layer_dense(units = 8, activation = 'relu', input_shape = c(28*28)) %>%
  layer_dense(units = 8, activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax')
```

1. Define the model The identity function is used for regression problems.

```
model %>% compile(
optimizer = "rmsprop",
loss = "categorical_crossentropy", metrics = c("accuracy")
)
```

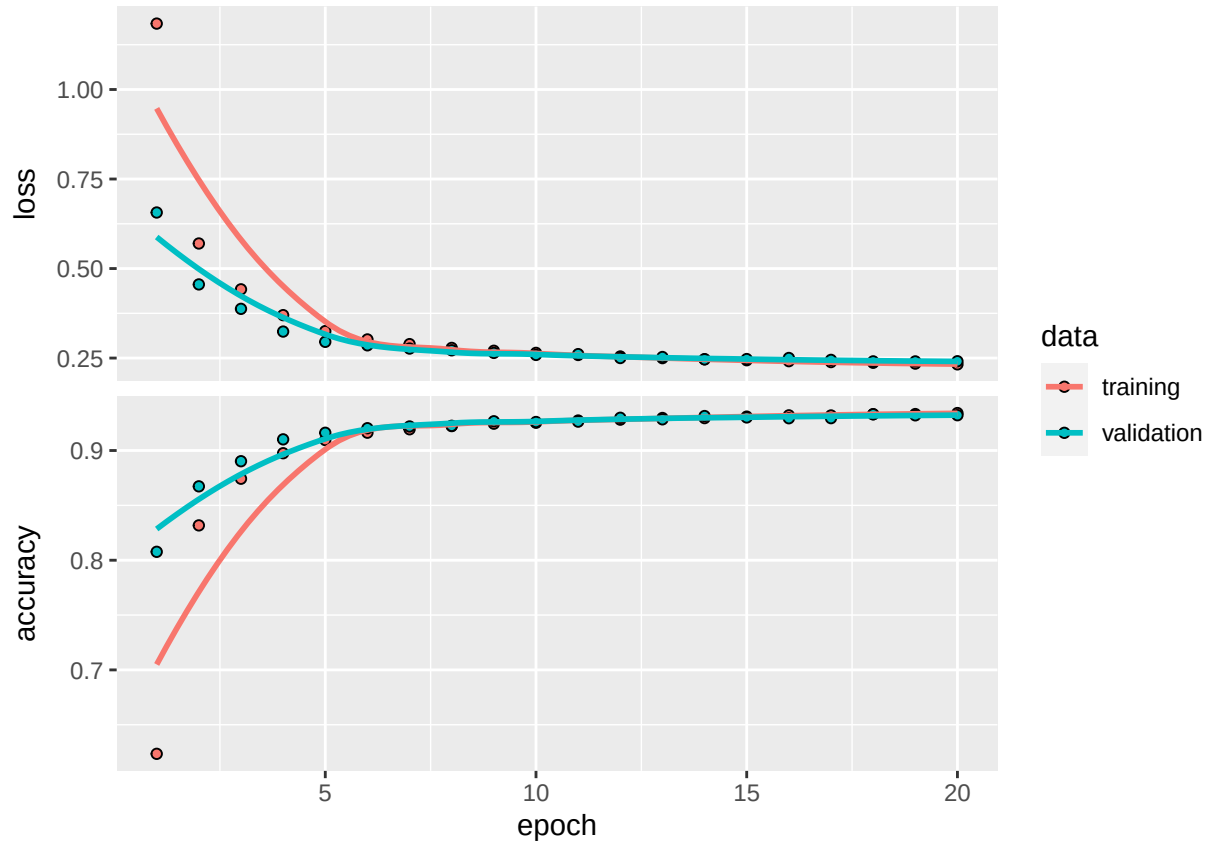
2. Compile

```
history = model %>% fit(x_train, y_train,
epochs = 20,
batch_size = 128,
validation_split = 0.2)

str(history)
```

3. Train

```
## List of 2
##  $ params :List of 3
##   ..$ verbose: int 1
##   ..$ epochs : int 20
##   ..$ steps  : int 375
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 1.184 0.57 0.442 0.37 0.325 ...
##   ..$ accuracy    : num [1:20] 0.623 0.832 0.874 0.898 0.91 ...
##   ..$ val_loss    : num [1:20] 0.656 0.456 0.387 0.324 0.295 ...
##   ..$ val_accuracy: num [1:20] 0.808 0.867 0.89 0.91 0.916 ...
##  - attr(*, "class")= chr "keras_training_history"
```

```
plot(history)
```




```
model %>% evaluate(x_test,y_test)
```

```
##      loss  accuracy
## 0.2564999 0.9314000
```

Number of parameters

```
str(model)
```

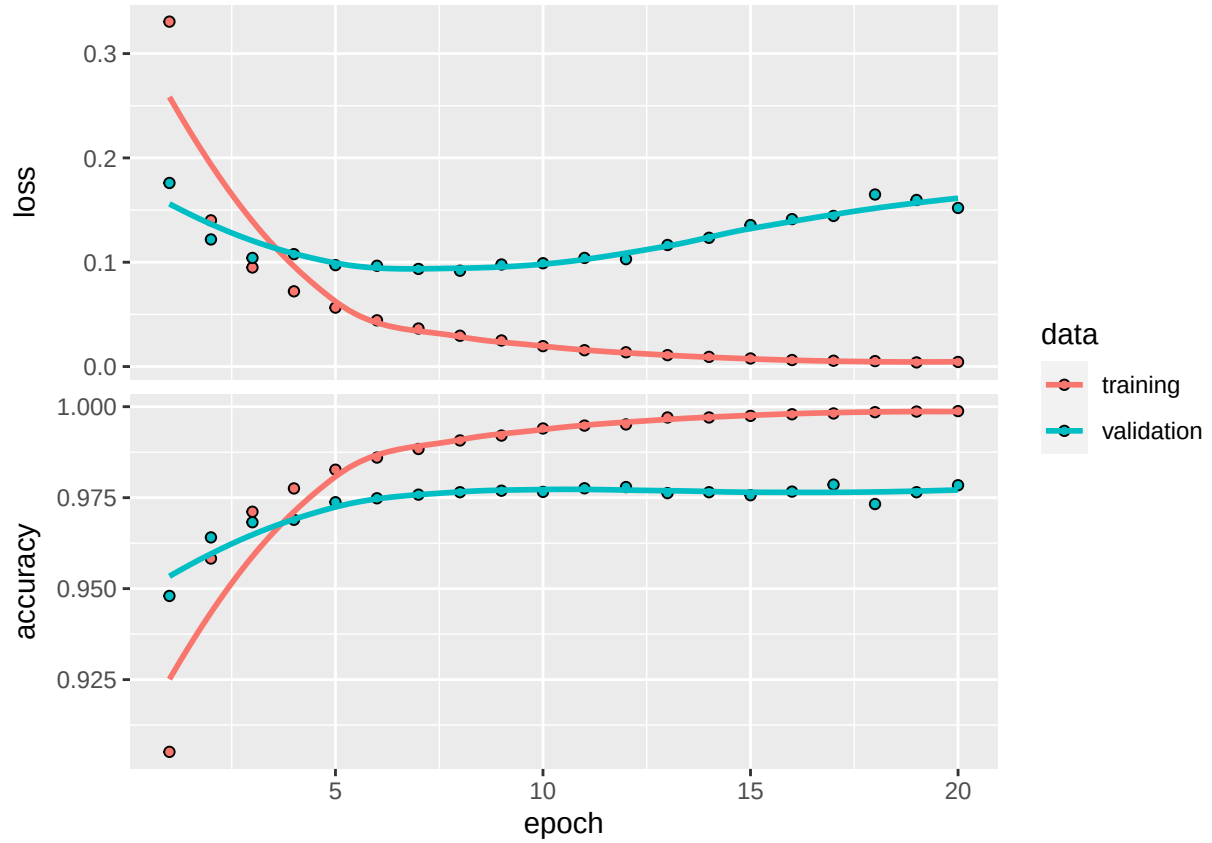
```
## Model: "sequential"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  dense_2 (Dense)                    (None, 8)                       6280
##
##  dense_1 (Dense)                    (None, 8)                       72
##
##  dense (Dense)                      (None, 10)                      90
##
## =========================================================================
## Total params: 6,442
## Trainable params: 6,442
## Non-trainable params: 0
## _________________________________________________________________________
```

b)

```
model = keras_model_sequential() %>%
  layer_dense(units = 128, activation = 'relu', input_shape = c(28*28)) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax')

model %>% compile(
optimizer = "rmsprop",
loss = "categorical_crossentropy", metrics = c("accuracy")
)

history = model %>% fit(x_train, y_train,
epochs = 20,
batch_size = 128,
validation_split = 0.2)

str(history)
```

```
## List of 2
##  $ params :List of 3
##   ..$ verbose: int 1
##   ..$ epochs : int 20
##   ..$ steps  : int 375
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 0.3306 0.1402 0.095 0.0721 0.0565 ...
##   ..$ accuracy    : num [1:20] 0.905 0.958 0.971 0.978 0.983 ...
##   ..$ val_loss    : num [1:20] 0.176 0.1218 0.1041 0.1078 0.0972 ...
##   ..$ val_accuracy: num [1:20] 0.948 0.964 0.968 0.969 0.974 ...
##  - attr(*, "class")= chr "keras_training_history"
```

```
plot(history)
```




```
model %>% evaluate(x_test,y_test)
```

```
##      loss  accuracy
## 0.1169891 0.9780000
```

We see that the validation loss reach its minimum after 5-10 epochs and raises again afterwards. This larger network thus begins overfitting after the first few epochs.

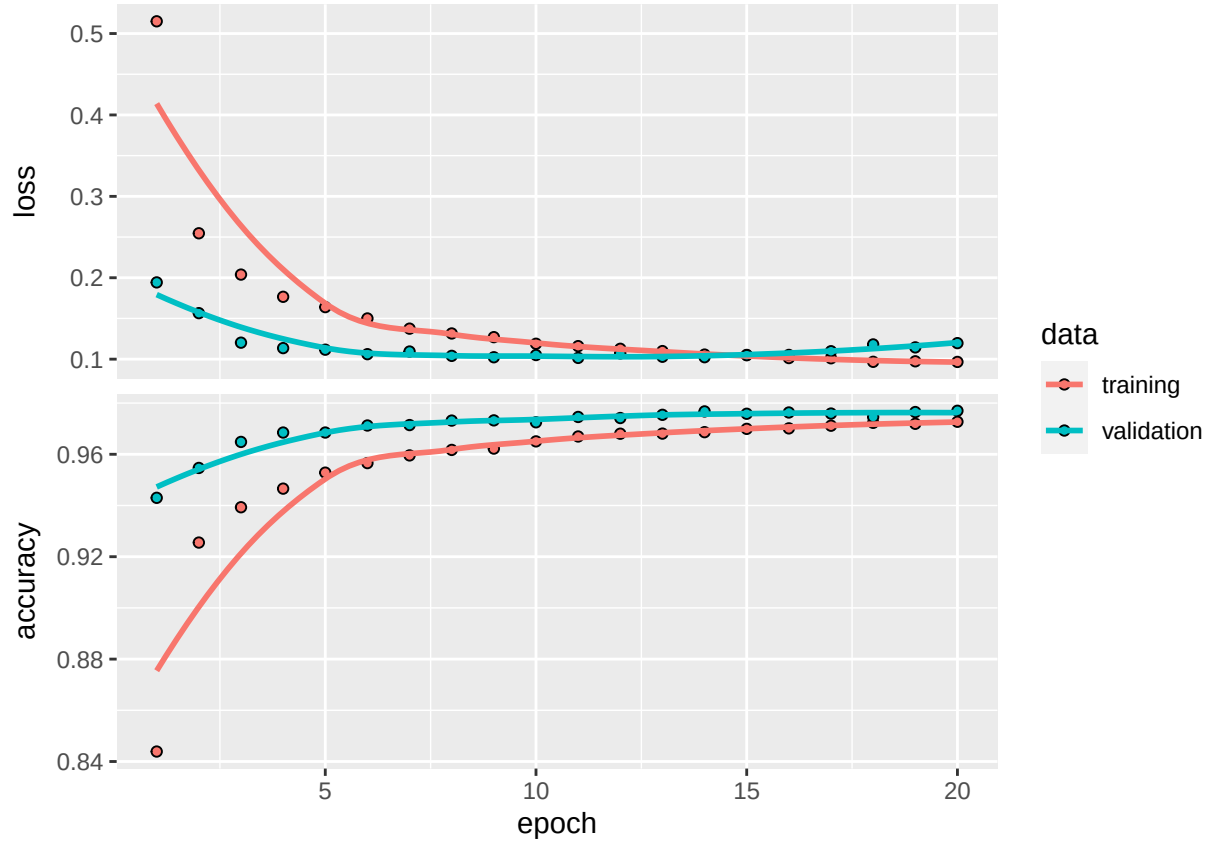
c)

```
model = keras_model_sequential() %>%
  layer_dense(units = 128, activation = 'relu', input_shape = c(28*28)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 10, activation = 'softmax')

model %>% compile(
optimizer = "rmsprop",
loss = "categorical_crossentropy", metrics = c("accuracy")
)

history = model %>% fit(x_train, y_train,
```

```
epochs = 20,
batch_size = 128,
validation_split = 0.2)

str(history)
```

```
## List of 2
##  $ params :List of 3
##   ..$ verbose: int 1
##   ..$ epochs : int 20
##   ..$ steps  : int 375
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 0.515 0.255 0.204 0.177 0.164 ...
##   ..$ accuracy    : num [1:20] 0.844 0.926 0.939 0.947 0.953 ...
##   ..$ val_loss    : num [1:20] 0.194 0.157 0.12 0.114 0.112 ...
##   ..$ val_accuracy: num [1:20] 0.943 0.955 0.965 0.969 0.969 ...
##  - attr(*, "class")= chr "keras_training_history"
```

```
plot(history)
```




```
model %>% evaluate(x_test,y_test)
```

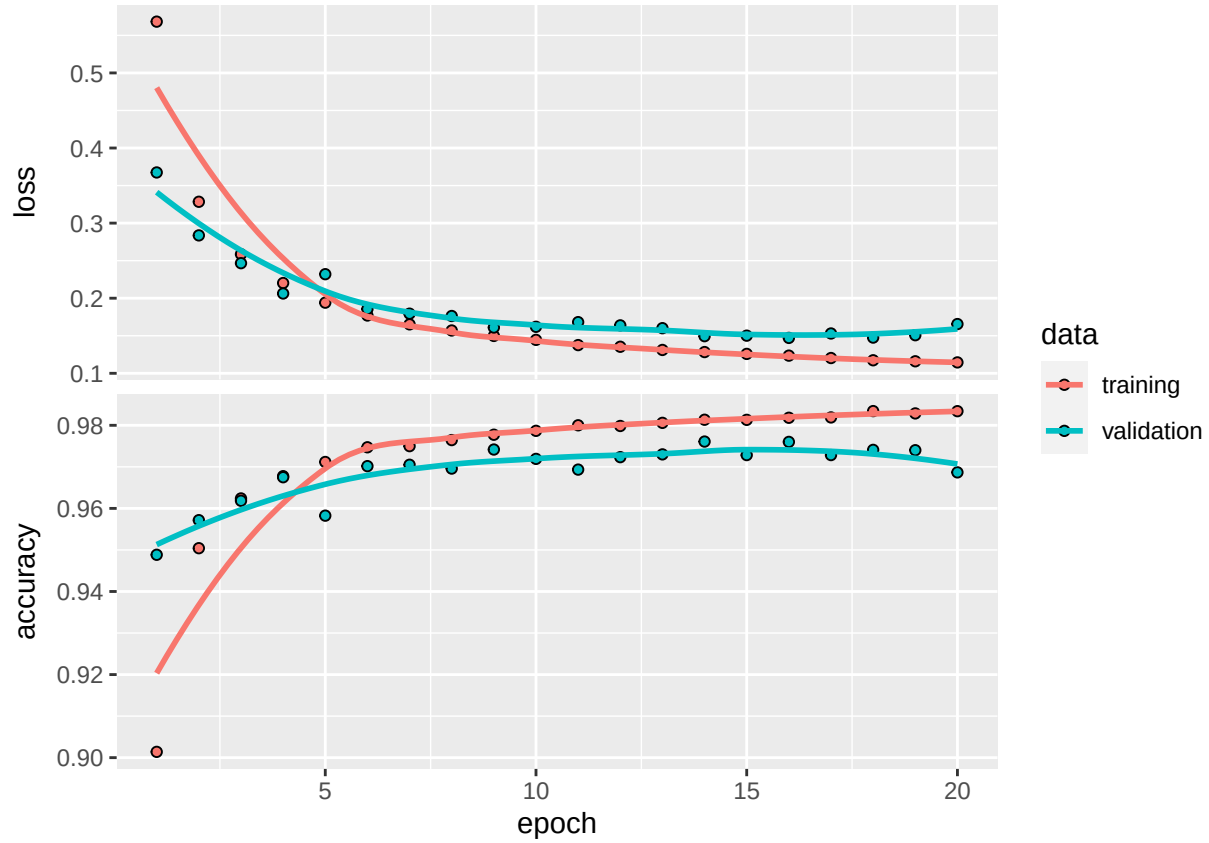
```
##      loss  accuracy
## 0.1027257 0.9761000
```

Here the validation curve looks better and we might increase the number of epochs.

```
model = keras_model_sequential() %>%
  layer_dense(units = 128, activation = 'relu',
              input_shape = c(28*28),kernel_regularizer = regularizer_l2(l = 0.001)) %>%
  layer_dense(units = 128, activation = 'relu',
              kernel_regularizer = regularizer_l2(l = 0.001)) %>%
  layer_dense(units = 10, activation = 'softmax')

model %>% compile(
optimizer = "rmsprop",
loss = "categorical_crossentropy", metrics = c("accuracy")
)

history = model %>% fit(x_train, y_train,
epochs = 20,
batch_size = 128,
validation_split = 0.2)

str(history)
```

```
## List of 2
##  $ params :List of 3
##   ..$ verbose: int 1
##   ..$ epochs : int 20
##   ..$ steps  : int 375
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 0.568 0.328 0.259 0.22 0.194 ...
##   ..$ accuracy    : num [1:20] 0.901 0.95 0.962 0.968 0.971 ...
##   ..$ val_loss    : num [1:20] 0.367 0.284 0.247 0.206 0.232 ...
##   ..$ val_accuracy: num [1:20] 0.949 0.957 0.962 0.967 0.958 ...
##  - attr(*, "class")= chr "keras_training_history"
```

```
plot(history)
```




```
model %>% evaluate(x_test,y_test)
```

```
##      loss  accuracy
## 0.1551789 0.9705000
```

d)

```
# re-define `x_train`, ..., `y_test` to keep them as 2-dimensional.
mnist = dataset_mnist()
x_train = mnist$train$x
y_train = mnist$train$y
x_test = mnist$test$x
y_test = mnist$test$y

# rescale
x_train = x_train / 255
x_test = x_test / 255

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

```
# define a CNN model
```

```
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 8, kernel_size = c(3,3), activation = "relu",
                input_shape = c(28,28,1)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 16, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 22, kernel_size = c(3,3), activation = "relu") %>%
  layer_flatten() %>%
  layer_dense(units = 10, activation = "softmax")
summary(model)
```

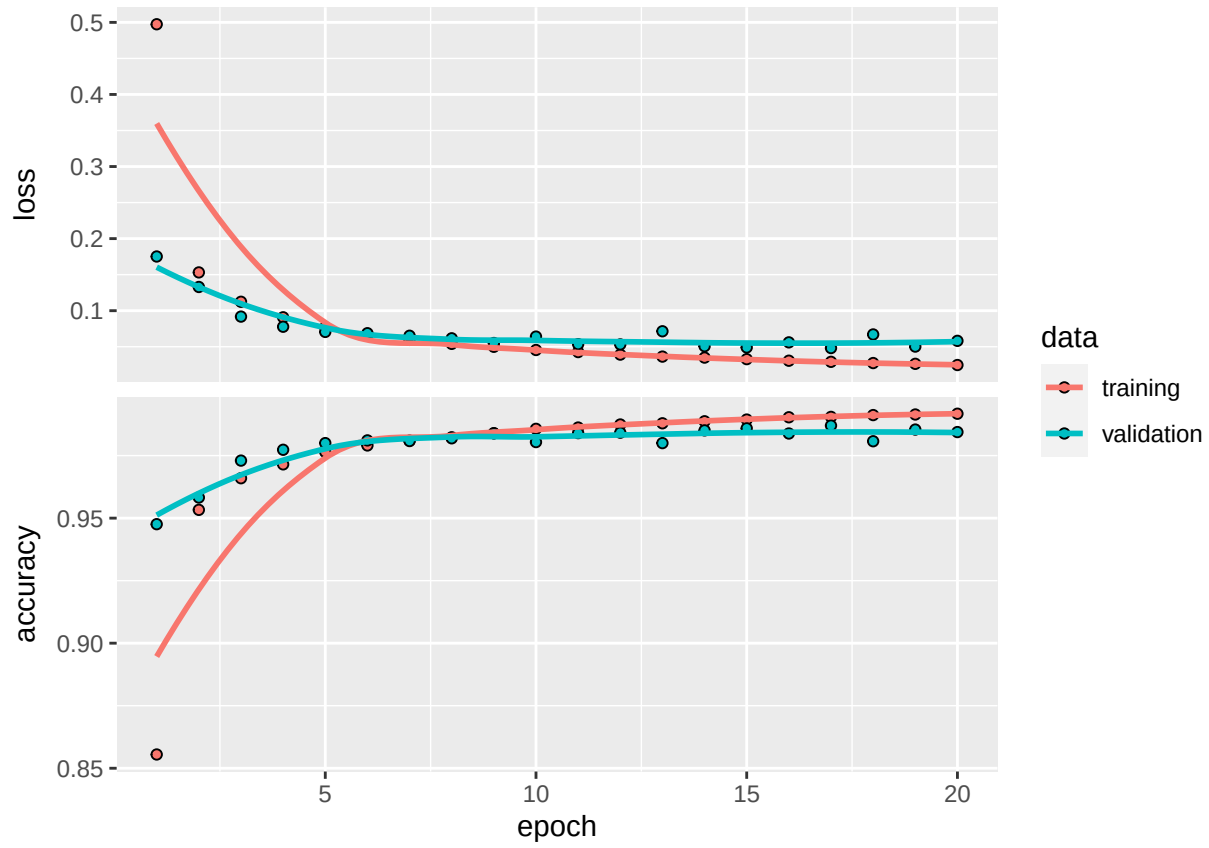
```
## Model: "sequential_4"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  conv2d_2 (Conv2D)                  (None, 26, 26, 8)               80
##
##  max_pooling2d_1 (MaxPooling2D)     (None, 13, 13, 8)               0
##
##  conv2d_1 (Conv2D)                  (None, 11, 11, 16)              1168
##
##  max_pooling2d (MaxPooling2D)       (None, 5, 5, 16)                0
##
##  conv2d (Conv2D)                    (None, 3, 3, 22)                3190
##
##  flatten (Flatten)                  (None, 198)                     0
##
##  dense_12 (Dense)                   (None, 10)                      1990
##
## ================================================================================
## Total params: 6,428
## Trainable params: 6,428
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
model %>% compile(
optimizer = "rmsprop",
loss = "categorical_crossentropy", metrics = c("accuracy")
)

history = model %>% fit(x_train, y_train,
epochs = 20,
batch_size = 128,
validation_split = 0.2)

str(history)
```

```
## List of 2
##  $ params :List of 3
##   ..$ verbose: int 1
##   ..$ epochs : int 20
##   ..$ steps  : int 375
```

```
##  $ metrics:List of 4
##   ..$ loss        : num [1:20] 0.4974 0.1531 0.1124 0.0908 0.077 ...
##   ..$ accuracy    : num [1:20] 0.855 0.953 0.966 0.971 0.976 ...
##   ..$ val_loss    : num [1:20] 0.1752 0.1329 0.0918 0.0777 0.0706 ...
##   ..$ val_accuracy: num [1:20] 0.948 0.958 0.973 0.977 0.98 ...
##  - attr(*, "class")= chr "keras_training_history"
```

```
plot(history)
```




```
model %>% evaluate(x_test,y_test)
```

```
##       loss   accuracy
## 0.04648211 0.98509997
```

The CNN model results in 98.7% test accuracy while the DNN model with the same model size results in 92.4% test accuracy. Hence, the CNN model is shown to be more effective in processing the image data.