Module 11: Solutions to Recommended Exercises

TMA4268 Statistical Learning V2023

Daesoo Lee, Emma Skarstein, Kenneth Aase, Stefanie Muff

Department of Mathematical Sciences, NTNU

April 27, 2023

Problem 1

a)

It is a 4-4-4-3 feedforward 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 classification 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

1. Load and preprocess data

```
# load
boston_housing <- dataset_boston_housing()
x_train <- boston_housing$train$x
y_train <- boston_housing$train$y
x_test <- boston_housing$test$x
y_test <- boston_housing$test$y

# preprocess
mean <- apply(x_train, 2, mean)
std <- apply(x_train, 2, sd)
x_train <- scale(x_train, center = mean, scale = std)
x_test <- scale(x_test, center = mean, scale = std)
```

a)

2. Define the model

```
model_r <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu", input_shape = 13) %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1)

summary(model_r)
```

```
## Model: "sequential"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  dense_2 (Dense)                    (None, 64)                      896
##  dense_1 (Dense)                    (None, 32)                      2080
##  dense (Dense)                      (None, 1)                       33
## ================================================================================
## Total params: 3,009
## Trainable params: 3,009
## Non-trainable params: 0
## ________________________________________________________________________________
```

3. Compile

```
model_r %>% compile(
  loss = "mean_squared_error",
  optimizer = optimizer_adam(learning_rate = 0.001),  # adam is the most common optimizer for its robustness.
  metrics = c("mean_absolute_error")
)
```

4. Train the model

```
history <- model_r %>% fit(
  x_train, y_train,
  epochs = 100,
  batch_size = 32,
  validation_data = list(x_test, y_test)
)
```

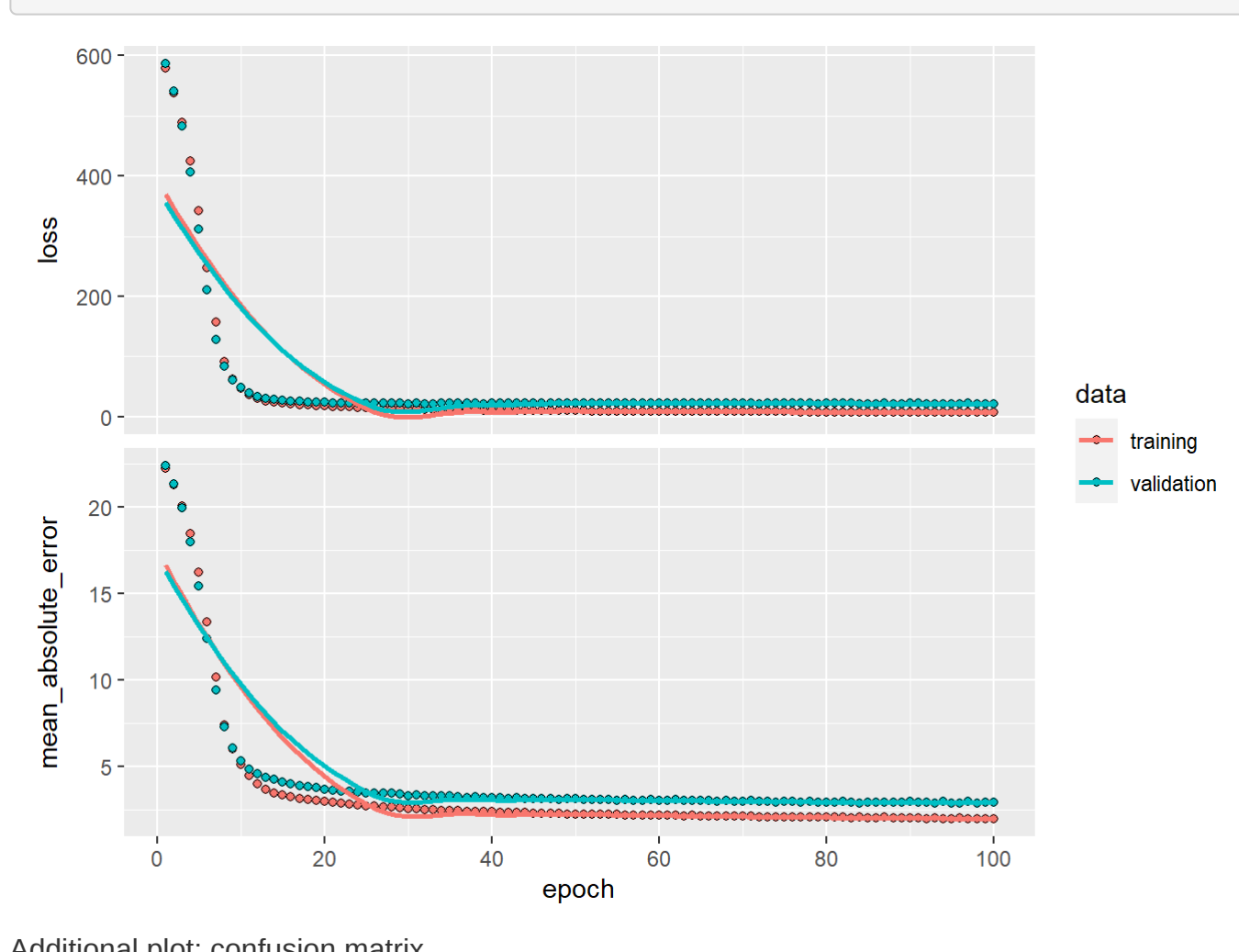
5. Test

```
scores <- model_r %>% evaluate(x_test, y_test, verbose = 0)

cat("Test loss (MSE):", scores[[1]], "\n",
    "Test mean absolute error (MAE):", scores[[2]], "\n")
```

```
## Test loss (MSE): 21.91999
##  Test mean absolute error (MAE): 2.906326
```

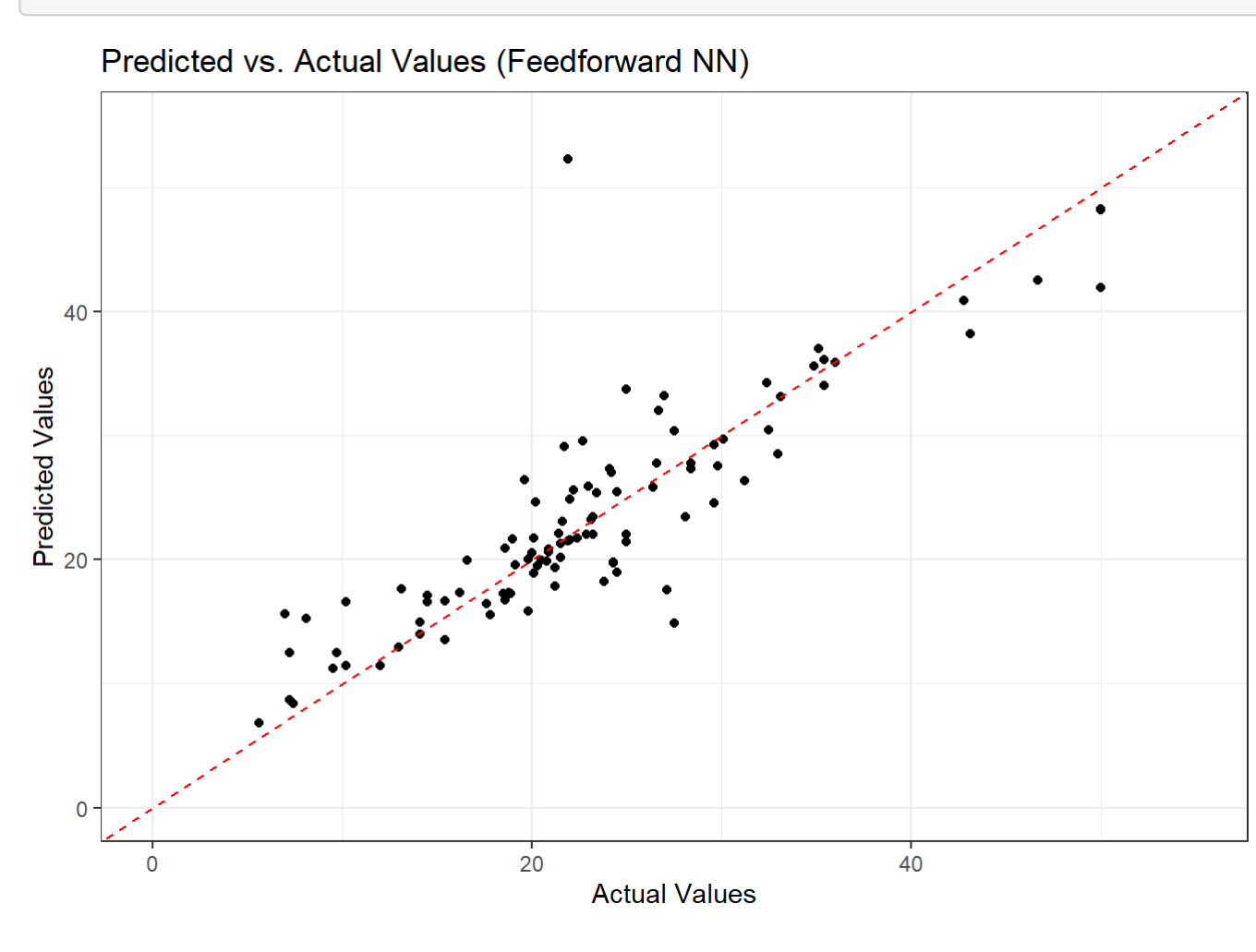
Plot training history

```
plot(history)
```



Additional plot: confusion matrix

```
predictions <- model_r %>% predict(x_test)
plot_df <- data.frame(Predicted = predictions, Actual = y_test)
ggplot(plot_df, aes(x = Actual, y = Predicted)) +
  geom_point() +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  theme_bw() +
  xlab("Actual Values") +
  ylab("Predicted Values") +
  ggtitle("Predicted vs. Actual Values (Feedforward NN)") +
  xlim(0, 55) +
  ylim(0, 55)
```



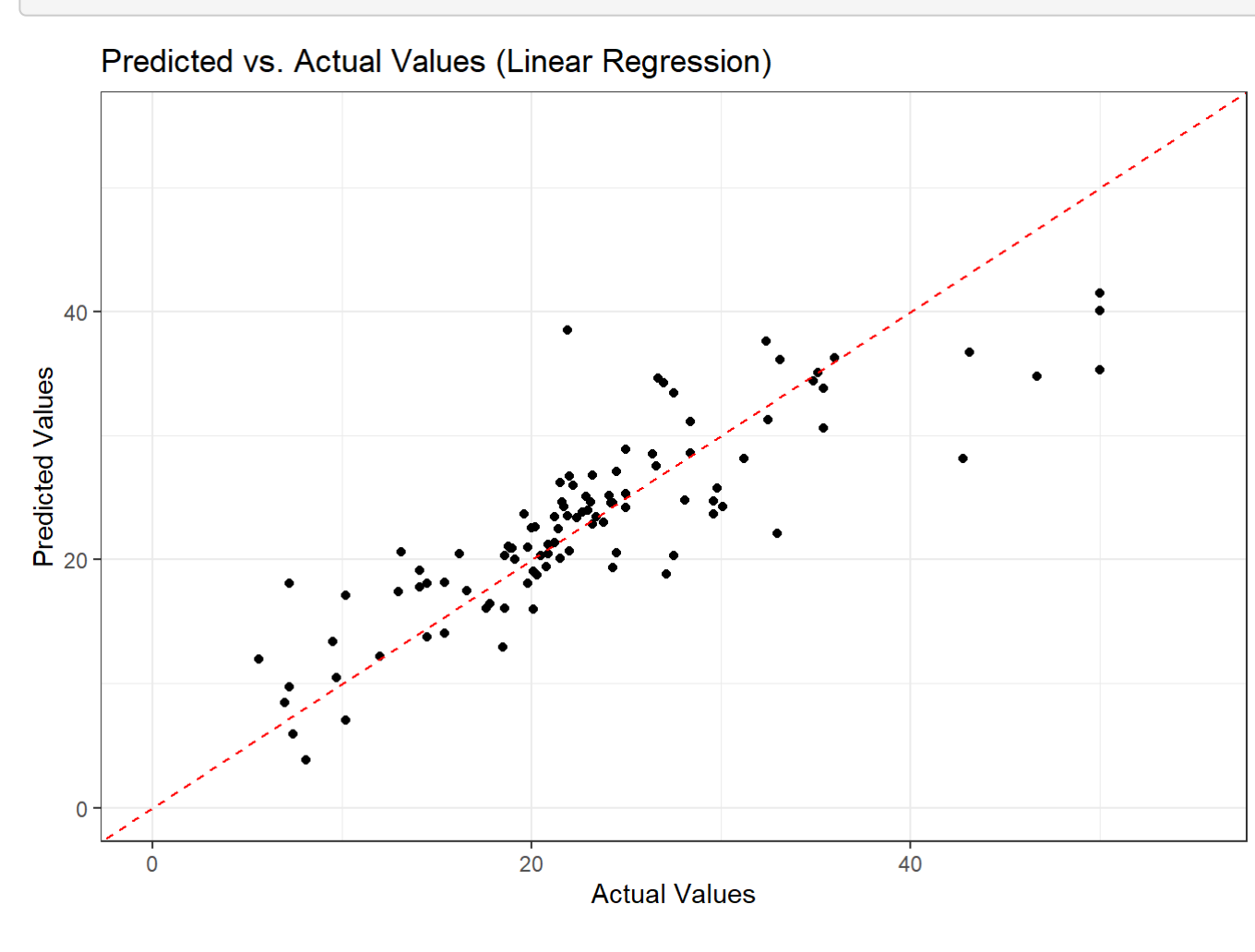
b)

Comparison to a Linear Regression Model

```
# Fit a linear regression model
linear_model <- lm(y_train ~ ., data = as.data.frame(cbind(x_train, y_train)))

# Make predictions on the test set
predictions <- predict(linear_model, as.data.frame(x_test))

# Calculate the mean squared error and mean absolute error
mse <- mean((y_test - predictions)^2)
mae <- mean(abs(y_test - predictions))

cat("=== [Feedforward Neural Network] === \n", "Test loss (MSE):", scores[[1]],
    "\n",
    "Test mean absolute error (MAE):", scores[[2]],
    "\n",
    "=================================== \n\n",
    "=== [Linear Regression] === \n",
    "Test loss (MSE):", mse, "\n",
    "Test mean absolute error (MAE):", mae, "\n",
    "==========================\n\n")
```

```
## === [Feedforward Neural Network] ===
##  Test loss (MSE): 21.91999
##  Test mean absolute error (MAE): 2.906326
##  ===================================
##
##  === [Linear Regression] ===
##  Test loss (MSE): 23.1956
##  Test mean absolute error (MAE): 3.464186
##  ==========================
```

```
plot_df <- data.frame(Predicted = predictions, Actual = y_test)
ggplot(plot_df, aes(x = Actual, y = Predicted)) +
  geom_point() +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  theme_bw() +
  xlab("Actual Values") +
  ylab("Predicted Values") +
  ggtitle("Predicted vs. Actual Values (Linear Regression)") +
  xlim(0, 55) +
  ylim(0, 55)
```



c)

- The feedforward neural network (FNN) demonstrates superior performance compared to the linear model. However, the FNN comes with reduced interpretability and increased complexity. As a result, some may prefer the simpler and more interpretable linear model.

Problem 4: Convolutional Neural Network (CNN)

Problem 4.1: Image Classification with CNN

1. Load and preprocess data

```
cifar10 <- dataset_cifar10()
x_train <- cifar10$train$x / 255
y_train <- to_categorical(cifar10$train$y, num_classes = 10)
x_test <- cifar10$test$x / 255
y_test <- to_categorical(cifar10$test$y, num_classes = 10)
```

a)

2. Define the model

```
model_c <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu", input_shape = c(32, 32, 3)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 10, activation = "softmax")

summary(model_c)
```

```
## Model: "sequential_1"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  conv2d_1 (Conv2D)                  (None, 30, 30, 32)              896
##  max_pooling2d_1 (MaxPooling2D)     (None, 15, 15, 32)              0
##  conv2d (Conv2D)                    (None, 13, 13, 64)              18496
##  max_pooling2d (MaxPooling2D)       (None, 6, 6, 64)                0
##  flatten (Flatten)                  (None, 2304)                    0
##  dense_4 (Dense)                    (None, 64)                      147520
##  dense_3 (Dense)                    (None, 10)                      650
## ================================================================================
## Total params: 167,562
## Trainable params: 167,562
## Non-trainable params: 0
## ________________________________________________________________________________
```

3. Compile

```
model_c %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer_adam(learning_rate = 0.001),  # adam is the most common optimizer for its robustness.
  metrics = c("accuracy")
)
```

4. Train the model

```
history <- model_c %>% fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 64,
  validation_data = list(x_test, y_test)
)
```

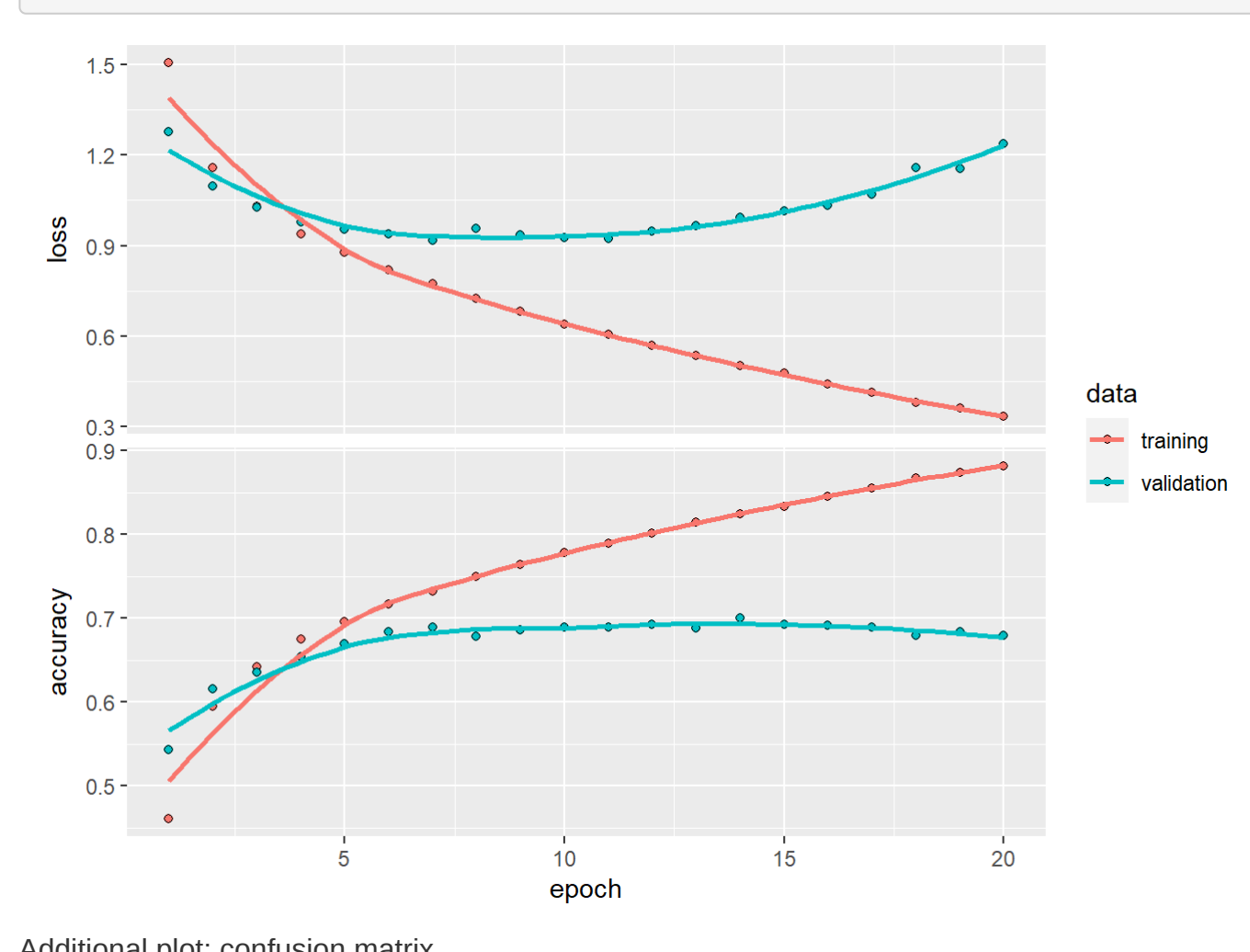
5. Test

```
scores <- model_c %>% evaluate(x_test, y_test, verbose = 0)

cat("Test loss:", scores[[1]], "\n",
    "Test accuracy:", scores[[2]], "\n")
```

```
## Test loss: 1.237937
##  Test accuracy: 0.6794
```

Plot training history

```
plot(history)
```



Additional plot: confusion matrix

```
library(caret)
predictions <- model_c %>% predict(x_test)%>% k_argmax()
y_true <- cifar10$test$y
confusion_matrix <- confusionMatrix(factor(as.vector(predictions)), factor(y_true))
print(confusion_matrix$table)
```

```
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 725  32  56  30  23  21   8  20  59  42
##          1  19 768   6  18   7   7  12   7  33  64
##          2  77  14 614  84  85  84  66  45  23  16
##          3  13   4  54 401  51 122  55  33   8  12
##          4  16   5  76  85 623  47  60  59   8   4
##          5   9   3  68 214  55 613  42  55   7   8
##          6   5   6  42  44  44  15 696   4   4   4
##          7   9   3  39  52  84  62  12 738   7  18
##          8  89  35  24  30  16  12  25  10 822  38
##          9  38 130  21  42  12  17  24  29  29 794
```

b)

The exact misclassification rate should be slightly different for each run. A misclassification rate is calculated as (number of misclassified samples / total number of samples).

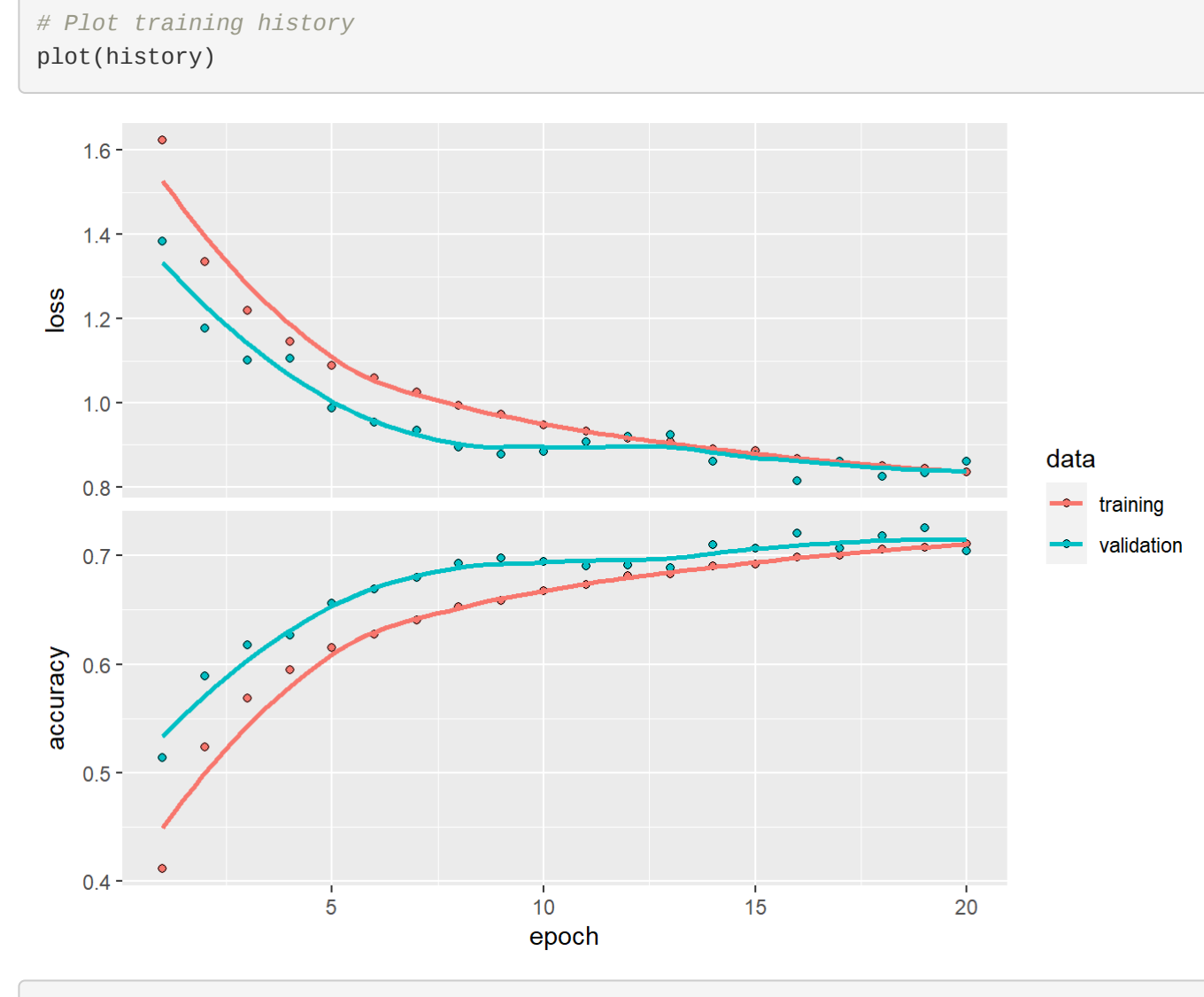
Problem 4.2: Improving the test accuarcy with data augmentation techniques

```
# 1) Load and preprocess data
cifar10 <- dataset_cifar10()
x_train <- cifar10$train$x / 255
y_train <- to_categorical(cifar10$train$y, num_classes = 10)
x_test <- cifar10$test$x / 255
y_test <- to_categorical(cifar10$test$y, num_classes = 10)

# 2) Define the model
model_ca <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu", input_shape = c(32, 32, 3)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 10, activation = "softmax")

# 3) Compile
model_ca %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer_adam(learning_rate = 0.001),
  metrics = c("accuracy")
)

# 4) Data augmentation
datagen <- image_data_generator(
  rotation_range = 10,
  width_shift_range = 0.1,
  height_shift_range = 0.1,
  horizontal_flip = TRUE
)

# Compute the data generator internal statistics
datagen %>% fit_image_data_generator(x_train)

# 5) Train the model with data augmentation
batch_size = 64
train_generator <- flow_images_from_data(x = x_train, y = y_train, generator = datagen, batch_size = batch_size)

history <- model_ca %>% fit_generator(
  generator = train_generator,
  steps_per_epoch = as.integer(nrow(x_train) / batch_size),
  epochs = 20,
  validation_data = list(x_test, y_test)
)
```

```
# Test
scores <- model_ca %>% evaluate(x_test, y_test, verbose = 0)

cat("Test loss:", scores[[1]], "\n",
    "Test accuracy:", scores[[2]], "\n")
```

```
## Test loss: 0.860364
##  Test accuracy: 0.7045
```

```
# Plot training history
plot(history)
```




```
# Additional plot: confusion matrix

predictions <- model_ca %>% predict(x_test)%>% k_argmax()
y_true <- cifar10$test$y
confusion_matrix <- confusionMatrix(factor(as.vector(predictions)), factor(y_true))
print(confusion_matrix$table)
```

```
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 722   9  51  10   9  10   2  10  44  11
##          1  16 804   6  11   2   8   2   2  18  22
##          2  47   2 487  46  36  38  17  13   9   4
##          3  10   3  38 448  28 172  21  22   4   2
##          4  25   1 108  72 623  63  13  35   7   6
##          5   1   2  27  71   1 467   5  19   1   1
##          6  20  16 188 206 154 107 918  30  12   8
##          7  18   4  39  64 130  94   5 828   7  11
##          8  73  29  23  23  11  11   6   6 831  18
##          9  68 130  33  49   6  30  11  35  67 917
```

a)

1. Increased size of the training dataset: Data augmentation allows for the creation of new training examples from the existing ones, which increases the size of the training dataset. A larger dataset helps in building more robust machine learning models that are less likely to overfit to the training data.
2. Improved generalization: By augmenting the training data, the model is exposed to more diverse examples, which helps it to generalize better to new, unseen data.
3. Increased model performance: Data augmentation can improve the performance of the model by reducing overfitting, especially in cases where the original dataset is small.
4. Cost-effectiveness: Data augmentation can be a cost-effective way of creating new training data, especially when collecting new data is expensive or time-consuming.
5. Reduced bias: Data augmentation can help to reduce bias in the dataset by balancing the class distribution, which is particularly important in cases where the original dataset is imbalanced.
6. Robustness to input variations: Data augmentation can make the model more robust to input variations such as rotation, scaling, and translation, which is useful in applications such as object recognition and natural language processing.

Problem 5: Univariate Time Series Classification with CNN

1. Load and preprocess data

```
# load the Wafer dataset
train <- read.delim("dataset/Wafer/Wafer_TRAIN.tsv", header = FALSE, sep = "\t")
test <- read.delim("dataset/Wafer/Wafer_TEST.tsv", header = FALSE, sep = "\t")

# the first column in `train` and `test` contains label info.
# therefore we separate them into `x` and `y`.
x_train <- train[,2:dim(train)[2]]
y_train <- clip(train[,1], 0, 1) #- 1
y_train <- to_categorical(y_train)
x_test <- test[,2:dim(test)[2]]
y_test <- clip(test[,1], 0, 1) #- 1
y_test <- to_categorical(y_test)

# create a channel dimension so that `x` has dimension of (batch, channel, length)
x_train <- array(as.matrix(x_train), dim = c(nrow(x_train), ncol(x_train), 1))
x_test <- array(as.matrix(x_test), dim = c(nrow(x_test), ncol(x_test), 1))

# preprocess
# The provided dataset has already been preprocessed, therefore no need for it.
```

a)

2. Define the model

```
# Define the model
model_1dc <- keras_model_sequential() %>%
    layer_conv_1d(filters = 16, kernel_size = 8, activation = "relu", input_shape = c(dim(x_train)[2], 1)) %>%
    layer_max_pooling_1d(pool_size = 2) %>%
    layer_conv_1d(filters = 32, kernel_size = 5, activation = "relu") %>%
    layer_max_pooling_1d(pool_size = 2) %>%
    layer_conv_1d(filters = 64, kernel_size = 3, activation = "relu") %>%
    layer_max_pooling_1d(pool_size = 2) %>%
    layer_flatten() %>%
    layer_dense(units = 2, activation = "softmax")

summary(model_1dc)
```

```
## Model: "sequential_3"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  conv1d_2 (Conv1D)                  (None, 145, 16)                 144
##  max_pooling1d_2 (MaxPooling1D)     (None, 72, 16)                  0
##  conv1d_1 (Conv1D)                  (None, 68, 32)                  2592
##  max_pooling1d_1 (MaxPooling1D)     (None, 34, 32)                  0
##  conv1d (Conv1D)                    (None, 32, 64)                  6208
##  max_pooling1d (MaxPooling1D)       (None, 16, 64)                  0
##  flatten_2 (Flatten)                (None, 1024)                    0
##  dense_7 (Dense)                    (None, 2)                       2050
## ================================================================================
## Total params: 10,994
## Trainable params: 10,994
## Non-trainable params: 0
## ________________________________________________________________________________
```

3. Compile

```
model_1dc %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer_adam(learning_rate = 0.001),
  metrics = c("accuracy")
)
```

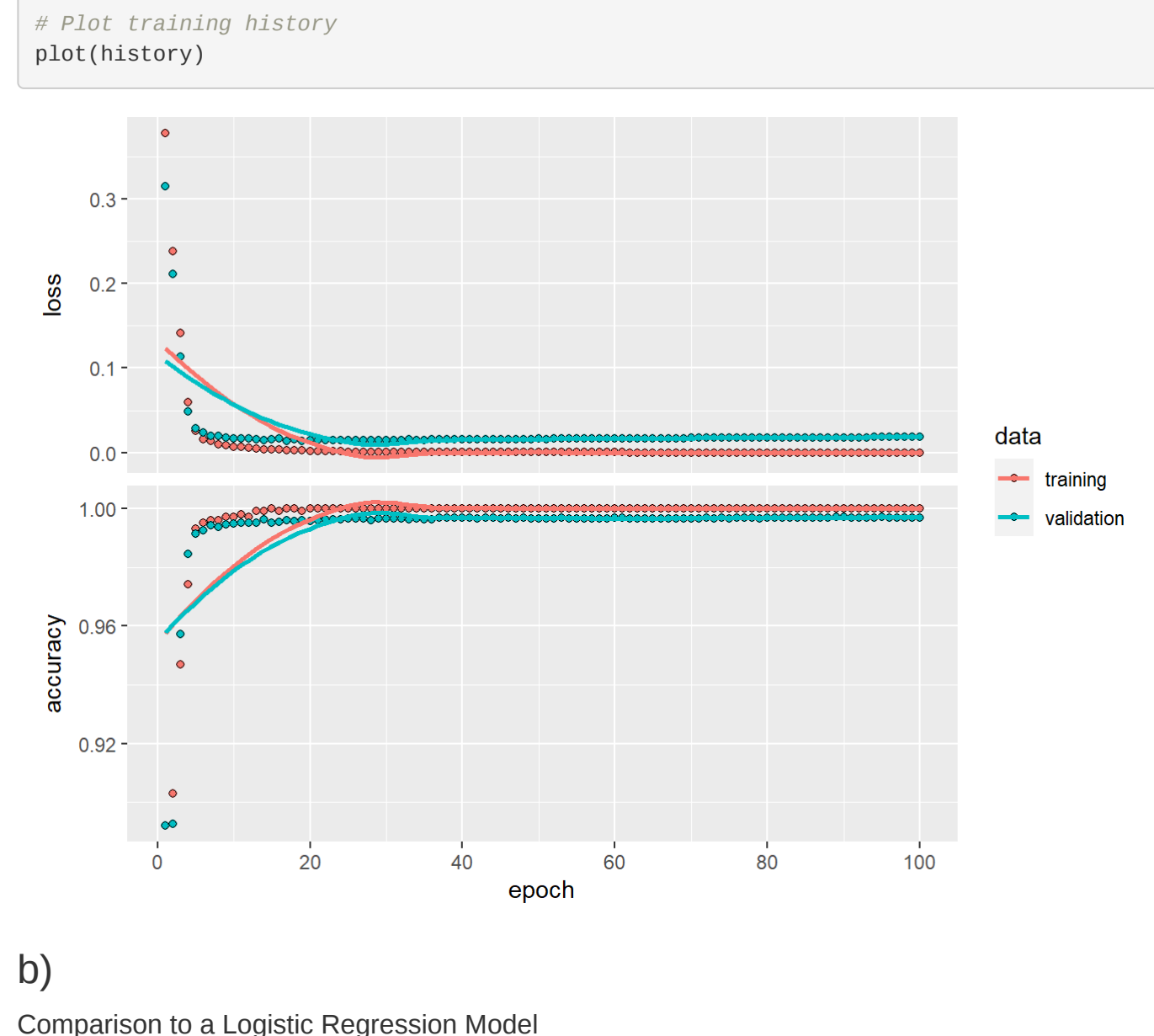
4. Train the model

```
history <- model_1dc %>% fit(
  x_train, y_train,
  epochs = 100,
  batch_size = 64,
  validation_data = list(x_test, y_test)
)
```

```
# Test
scores <- model_1dc %>% evaluate(x_test, y_test, verbose = 0)

cat("Test loss:", scores[[1]], "\n",
    "Test accuracy:", scores[[2]], "\n")
```

```
## Test loss: 0.01831522
##  Test accuracy: 0.9969176
```

```
# Plot training history
plot(history)
```



b)

Comparison to a Logistic Regression Model

```
# dataset
x_train <- train[,2:dim(train)[2]]
y_train <- clip(train[,1], 0, 1)
x_test <- test[,2:dim(test)[2]]
y_test <- clip(test[,1], 0, 1)

# Fit a linear regression model
#linear_model <- lm(y_train ~ ., data = as.data.frame(cbind(x_train, y_train)))
logit_reg <- glm(y_train ~ ., data = as.data.frame(cbind(x_train, y_train)), family = "binomial")

# Make predictions on the test set
predictions <- predict(logit_reg, newdata = x_test, type = "response")
predictions <- as.integer(predictions > 0.5)  # cutoff = 0.5
predictions <- as.factor(predictions)

result <- confusionMatrix(predictions, as.factor(y_test))

cat("=== [1D CNN] === \n", "Test accuracy:", scores[[2]],
    "\n",
    "=========================== \n\n",
    "=== [Logistic Regression] === \n",
    "Test accuracy:", result$overall["Accuracy"], "\n",
    "============================\n\n")
```

```
## === [1D CNN] ===
##  Test accuracy: 0.9969176
##  ===========================
##
##  === [Logistic Regression] ===
##  Test accuracy: 0.9018494
##  ============================
```