Deep neutral networks – Joanna Wojtukiewicz

Joanna Wojtukiewicz

7 02 2021

Installation

1. Library installation

```
#install.packages('keras')
```

```
library(keras)
library(dplyr, quietly = TRUE, warn.conflicts = FALSE)
library(ggplot2, quietly = TRUE, warn.conflicts = FALSE)
```

```
install_keras()
```

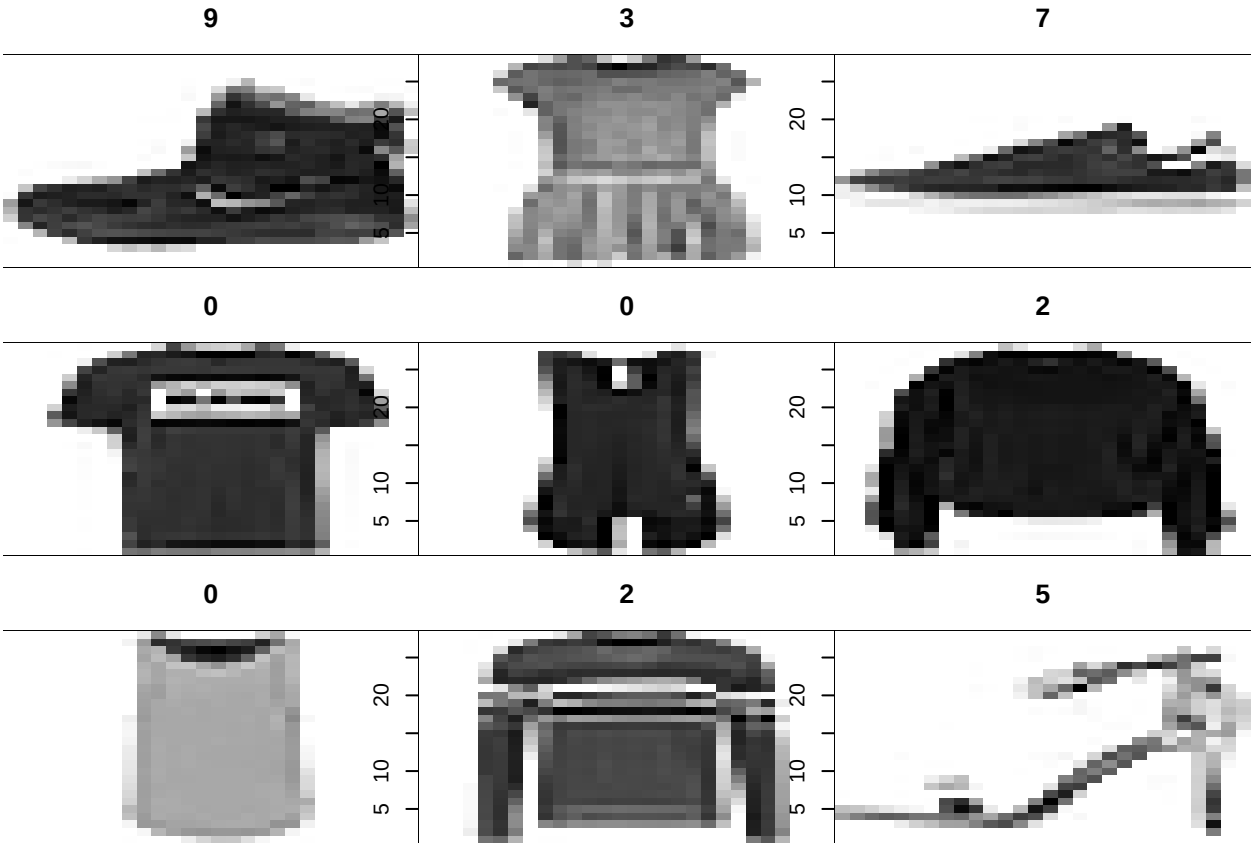
Fashion MNIST data set

Loading data set

```
fashion_mnist <- dataset_fashion_mnist()
c(x_train, y_train) %<-% fashion_mnist$train # Train set features, train set labels
c(x_test, y_test) %<-% fashion_mnist$test # Test set features, test set labels
```

Visualize the 9 first items from training set

```
par(mfcol = c(3, 3)) # Split the screen
par(mar = c(0, 0, 3, 0), xaxs = 'i', yaxs = 'i') # Margins
for (i in 1:9) {
  im <- x_train[i,,]
  im <- t(apply(im, 2, rev))
  image(1:28, 1:28, im, col = gray((255:0) / 255),
        xaxt = 'n', main = paste(y_train[i]))
}
```



Data prepare

To prepare the data for training we convert the 3-d arrays into matrices by reshaping width and height into a single dimension (28x28 images are flattened into length 784 vectors). Then, we convert the grayscale values from integers ranging between 0 to 255 into floating point values ranging between 0 and 1

```
# 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)
```

9 first encoded items from trainig set are below (the first column corresponds to zero, second to one, etc.):

```
head(y_train, 9)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    0    0    0    0    0    0    0    0     1
## [2,]    1    0    0    0    0    0    0    0    0     0
## [3,]    1    0    0    0    0    0    0    0    0     0
```

```
##  [4,]    0    0    0    1    0    0    0    0    0    0
##  [5,]    1    0    0    0    0    0    0    0    0    0
##  [6,]    0    0    1    0    0    0    0    0    0    0
##  [7,]    0    0    0    0    0    0    0    1    0    0
##  [8,]    0    0    1    0    0    0    0    0    0    0
##  [9,]    0    0    0    0    0    1    0    0    0    0
```

Defining the model

The core data structure of Keras is a model, a way to organize layers. The simplest type of model is the Sequential model, a linear stack of layers.

```
model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = 'relu', input_shape = 28 * 28) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 10, activation = 'softmax')
```

The input_shape argument to the first layer specifies the shape of the input data (a length 784 numeric vector representing a grayscale image). The final layer outputs a length 10 numeric vector (probabilities for each digit) using a softmax activation function. Softmax activation function takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers

$$f_i(\boldsymbol{z}) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}.$$

Summary the model

```
summary(model)
```

```
## Model: "sequential"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## dense_2 (Dense)                     (None, 256)                     200960
## ________________________________________________________________________________
## dropout_1 (Dropout)                 (None, 256)                     0
## ________________________________________________________________________________
## dense_1 (Dense)                     (None, 128)                     32896
## ________________________________________________________________________________
## dropout (Dropout)                   (None, 128)                     0
## ________________________________________________________________________________
## dense (Dense)                       (None, 10)                      1290
## ================================================================================
## Total params: 235,146
## Trainable params: 235,146
## Non-trainable params: 0
## ________________________________________________________________________________
```

Compile the model

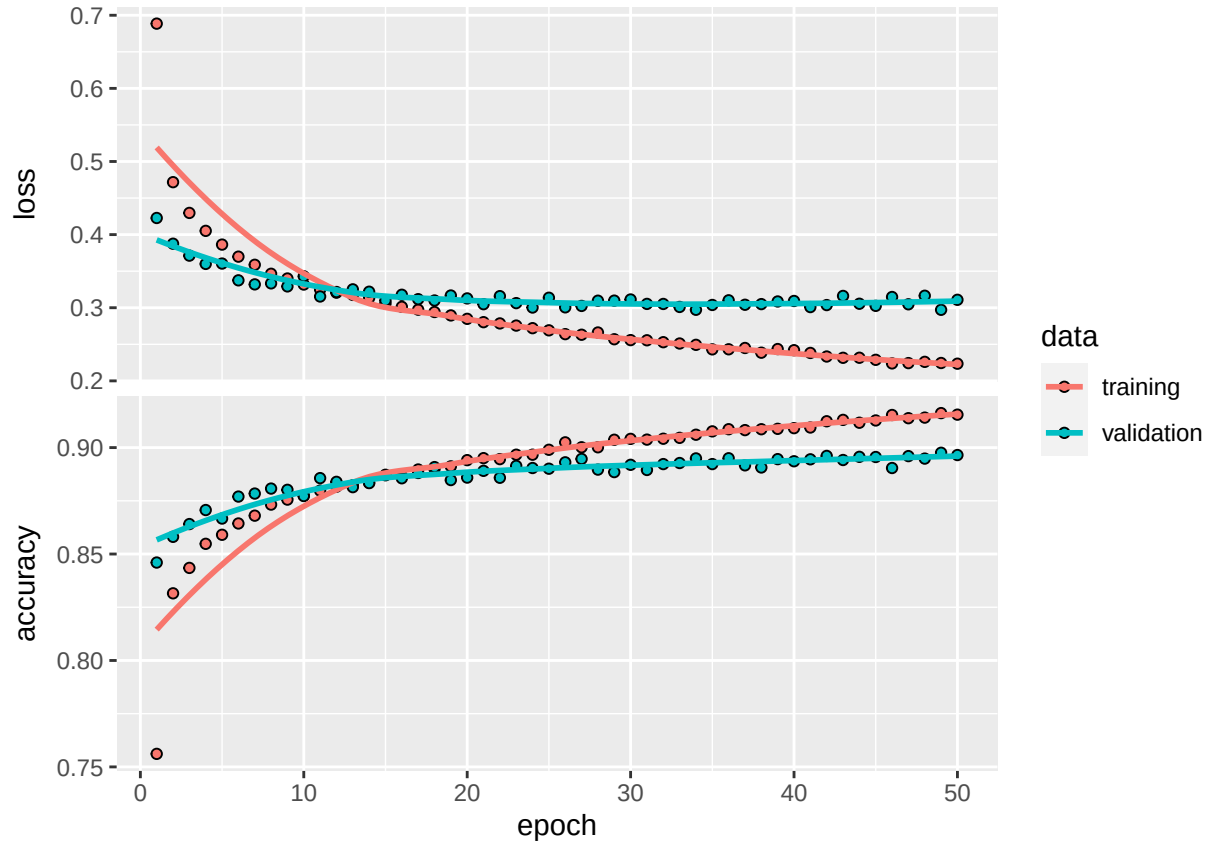
- Loss function – This measures how accurate the model is during training. We want to minimize this function to 'steer' the model in the right direction.
- Optimizer – This is how the model is updated based on the data it sees and its loss function.
- Metrics – Used to monitor the training and testing steps. The following example uses accuracy, the fraction of the digits that are correctly classified.

```
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_adam(),
  metrics = 'accuracy')
```

Training the model

```
model %>% fit(x_train,
              y_train,
              epochs = 50, # Number of epochs
              batch_size = 128, # Size of batch in single step
              validation_split = 0.2 # Percent of data in validation sets
) -> model_dnn
plot(model_dnn)
```

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



Evaluate the model

```
model %>% evaluate(x_train, y_train) # Evaluate the model's performance on the train data
```

```
##      loss  accuracy
## 0.1895796 0.9316334
```

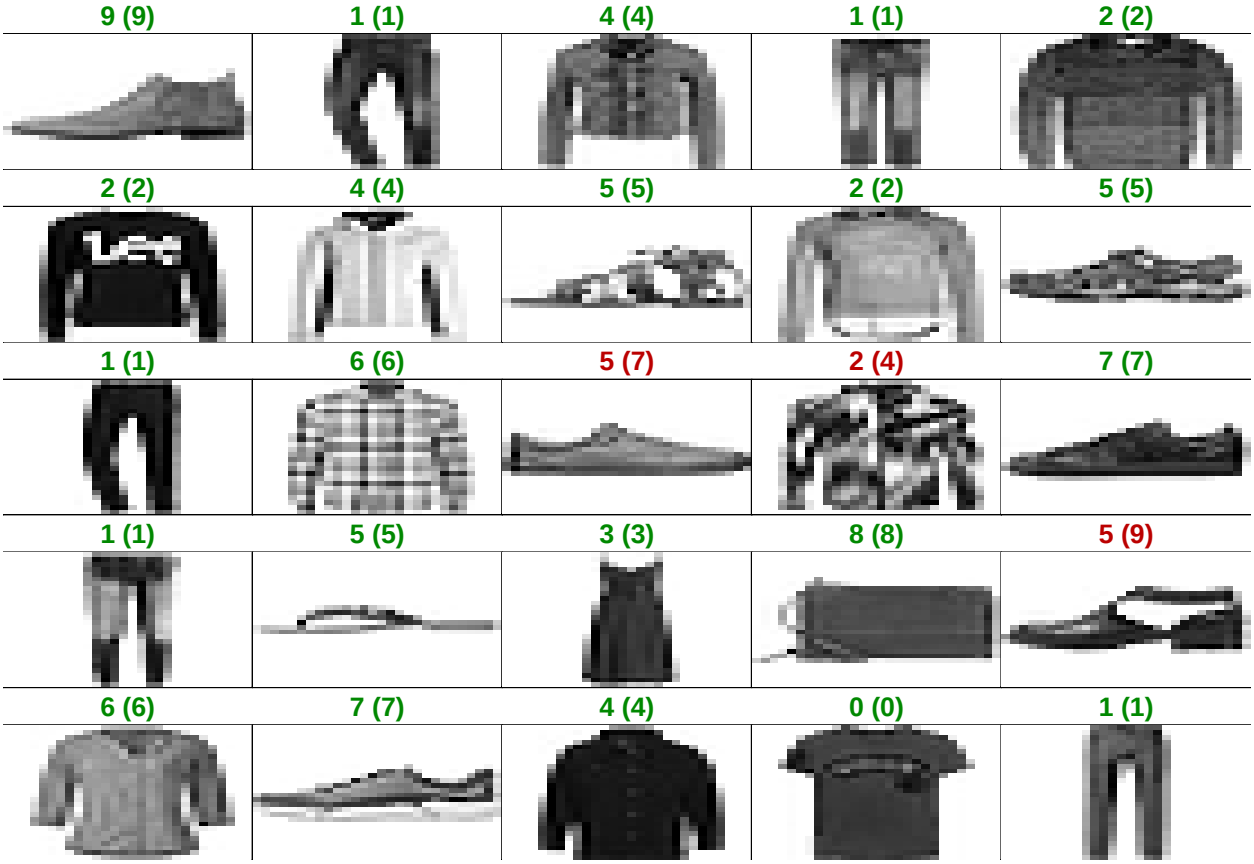
```
model %>% evaluate(x_test, y_test) # Evaluate the model's performance on the test data
```

```
##      loss  accuracy
## 0.3284694 0.8901000
```

Predictions

```
model %>% predict(x_test) -> predictions # Predicted probabilities on test data
model %>% predict_classes(x_test) -> predicted_items # Predicted digits on test data
```

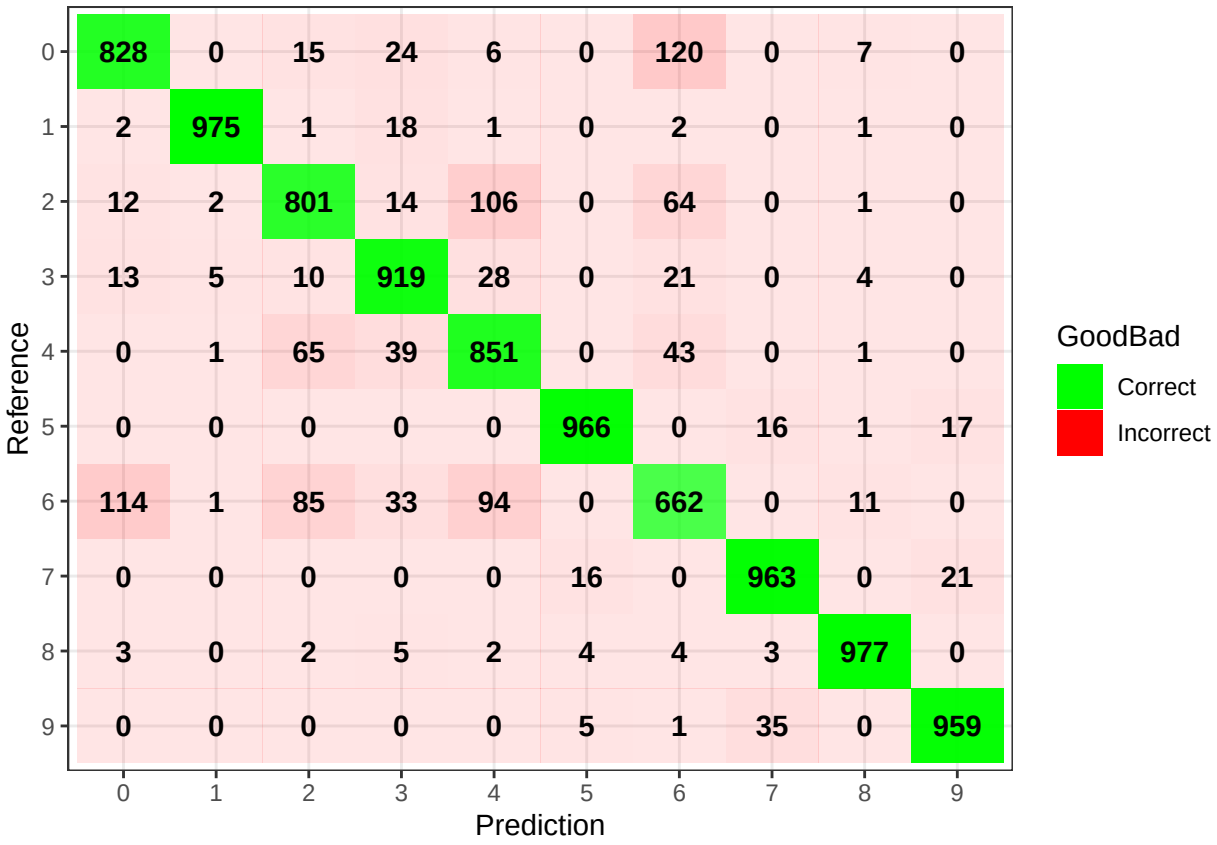
```
par(mfcol = c(5, 5))
par(mar = c(0, 0, 1.5, 0), xaxs = 'i', yaxs = 'i')
for (i in 1:25) {
  img <- fashion_mnist$test$x[i, , ]
  img <- t(apply(img, 2, rev))
  if (predicted_items[i] == fashion_mnist$test$y[i]) {
    color <- '#008800'
  } else {
    color <- '#bb0000'
  }
  image(1:28, 1:28, img, col = gray((255:0) / 255), xaxt = 'n', yaxt = 'n',
        main = paste0(predicted_items[i], ' (',
                      fashion_mnist$test$y[i], ')'),
        col.main = color)
}
```



Confusion matrix

```
data.frame(table(predicted_items, fashion_mnist$test$y)) %>%
  setNames(c('Prediction', 'Reference', 'Freq')) %>%
  mutate(GoodBad = ifelse(Prediction == Reference, 'Correct', 'Incorrect')) -> conf_table

conf_table %>%
  ggplot(aes(y = Reference, x = Prediction, fill = GoodBad, alpha = Freq)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = 0.5, fontface  = 'bold', alpha = 1) +
  scale_fill_manual(values = c(Correct = 'green', Incorrect = 'red')) +
  guides(alpha = FALSE) +
  theme_bw() +
  ylim(rev(levels(conf_table$Reference)))
```


Reference
Prediction
0 1 2 3 4 5 6 7 8 9
828 0 15 24 6 0 120 0 7 0
2 975 1 18 1 0 2 0 1 0
12 2 801 14 106 0 64 0 1 0
13 5 10 919 28 0 21 0 4 0
0 1 65 39 851 0 43 0 1 0
0 0 0 0 0 966 0 16 1 17
114 1 85 33 94 0 662 0 11 0
0 0 0 0 0 16 0 963 0 21
3 0 2 5 2 4 4 3 977 0
0 0 0 0 0 5 1 35 0 959
GoodBad
Correct
Incorrect