

# Deep neural networks – laboratory

Tomasz Górecki

Last update: 30.01.2021

## Installation

1. Library installation

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

2. The Keras R interface uses the [TensorFlow](#) backend engine by default. To install both the core Keras library as well as the TensorFlow backend:

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

```
install_keras()
```

## MNIST data set (Example – classification)

### Description

The MNIST dataset is included with Keras and can be accessed using the [dataset\\_mnist\(\)](#) function. The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. There have been a number of scientific papers on attempts to achieve the lowest error rate; one paper, using a hierarchical system of convolutional neural networks, manages to get an error rate on the MNIST database of 0.17%.

Type	Classifier	Error rate (%)
Linear classifier	Pairwise linear classifier	7.60
Non-linear classifier	40 PCA + quadratic classifier	3.30
Random Forest (RF)	Fast Unified Random Forests (RF-SRC)	2.80
Deep neural network (DNN)	2-layer 784-800-10	1.60
Boosted Stumps	Product of stumps on Haar features	0.87
Deep neural network (DNN)	2-layer 784-800-10	0.70
Support-vector machine (SVM)	Virtual SVM, deg-9 poly	0.56
K-Nearest Neighbors (KNN)	K-NN with non-linear deformation	0.52
Deep neural network (DNN)	6-layer 784-2500-2000-1500-1000-500-10	0.35
Convolutional neural network (CNN)	6-layer 784-40-80-500-1000-2000-10	0.31
Convolutional neural network (CNN)	6-layer 784-50-100-500-1000-10-10	0.27
Convolutional neural network (CNN)	Committee of 35 CNNs, 1-20-P-40-P-150-10	0.23
Convolutional neural network (CNN)	Committee of 5 CNNs, 6-layer 784-50-100-500-1000-10-10	0.21

Type	Classifier	Error rate (%)
Random Multimodel Deep Learning (RMDL)	10 NN-10 RNN - 10 CNN	0.18
Convolutional neural network (CNN)	Committee of 20 CNNs with Squeeze-and-Excitation Networks	0.17

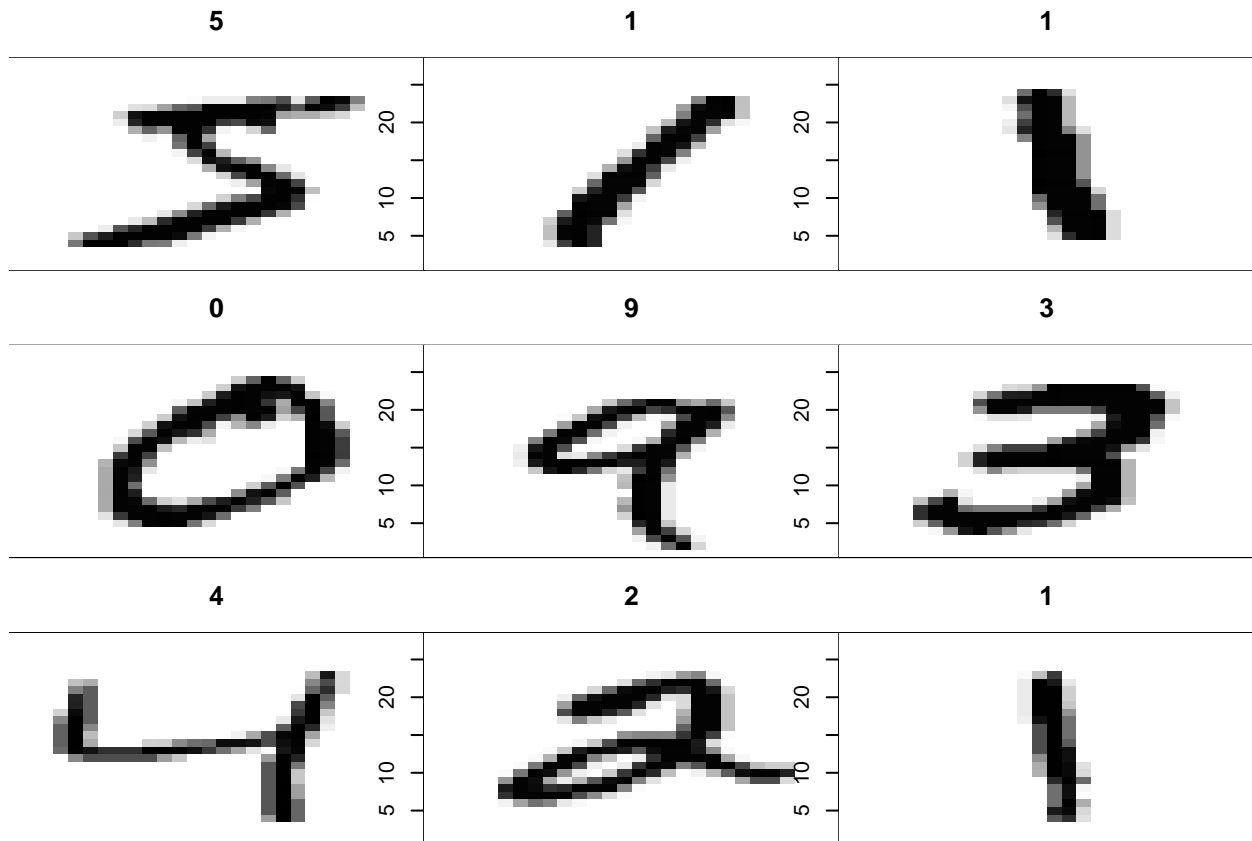
### Loading data set

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

The x data is a 3-d array (images, width, height) of grayscale values. The y data is an integer vector with values ranging from 0 to 9.

### Visualize the 9 first digits 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
```

To prepare this data for training we one-hot encode the vectors into binary class matrices using the Keras `to_categorical()` function. One hot encoding is a vector representation where all elements of the vector are 0 except one, which has 1 as its value (assigning 1 to working feature and 0's to other idle features).

```
y_train <- to_categorical(y_train, 10)
y_test  <- to_categorical(y_test,  10)
```

9 first encoded digits 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    1    0    0    0    0
## [2,]    1    0    0    0    0    0    0    0    0    0
## [3,]    0    0    0    0    1    0    0    0    0    0
```

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

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

## 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

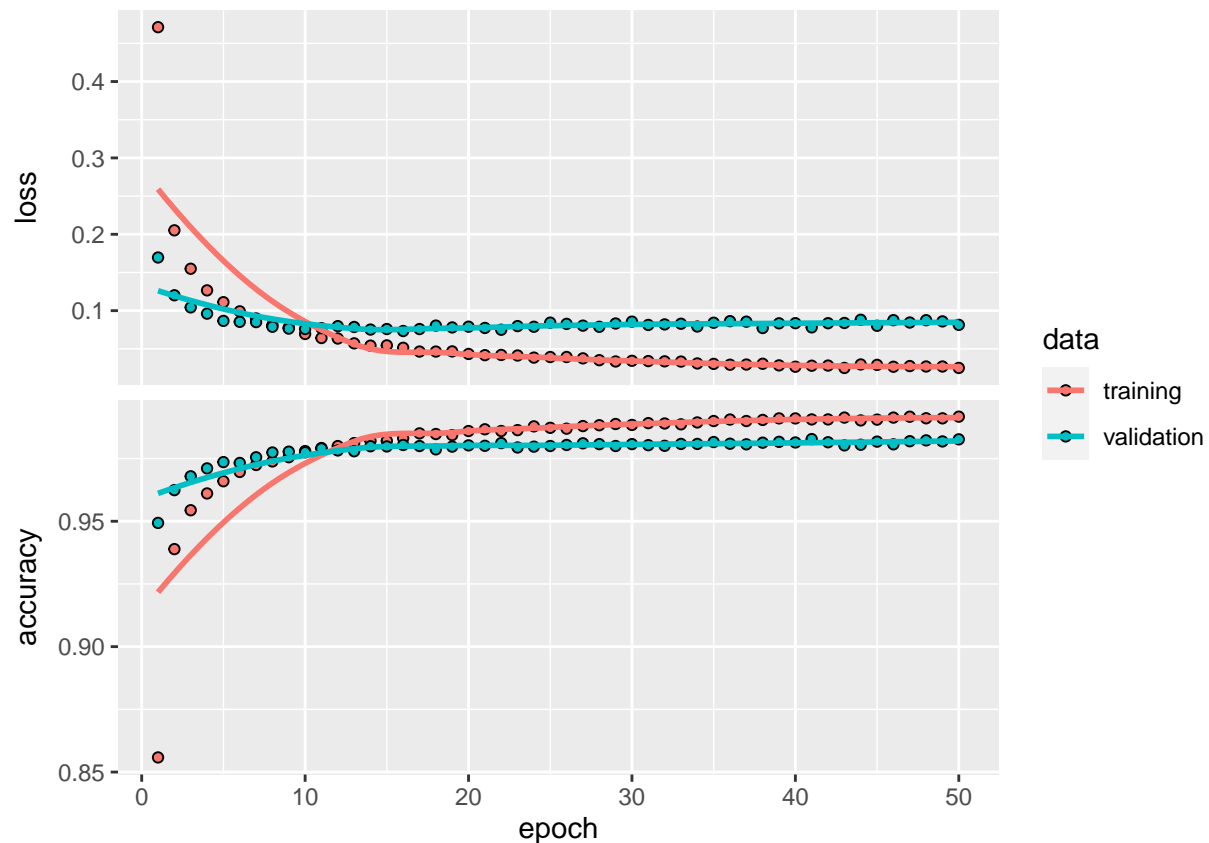
- Loss function – This measures how accurate the model is during training. We want to minimize this function to ‘steer’ the model in the right direction.
- Optimizer – This is how the model is updated based on the data it sees and its loss function.
- Metrics – Used to monitor the training and testing 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  
## [1] 0.01767  
##  
## $accuracy  
## [1] 0.9962
```

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

```
## $loss  
## [1] 0.07446  
##  
## $accuracy  
## [1] 0.9841
```
























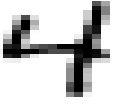
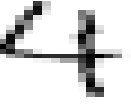
It turns out, the accuracy on the test dataset is a little less than the accuracy on the training dataset. This gap between training accuracy and test accuracy is an example of overfitting. Overfitting is when a machine learning model performs worse on new data than on their training data

## Predictions

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

A prediction is an array of 10 numbers. These describe the ‘confidence’ of the model that the image corresponds to each of the 10 different digits. Let’s plot several images with their predictions. Correct prediction labels are green and incorrect prediction labels are red.

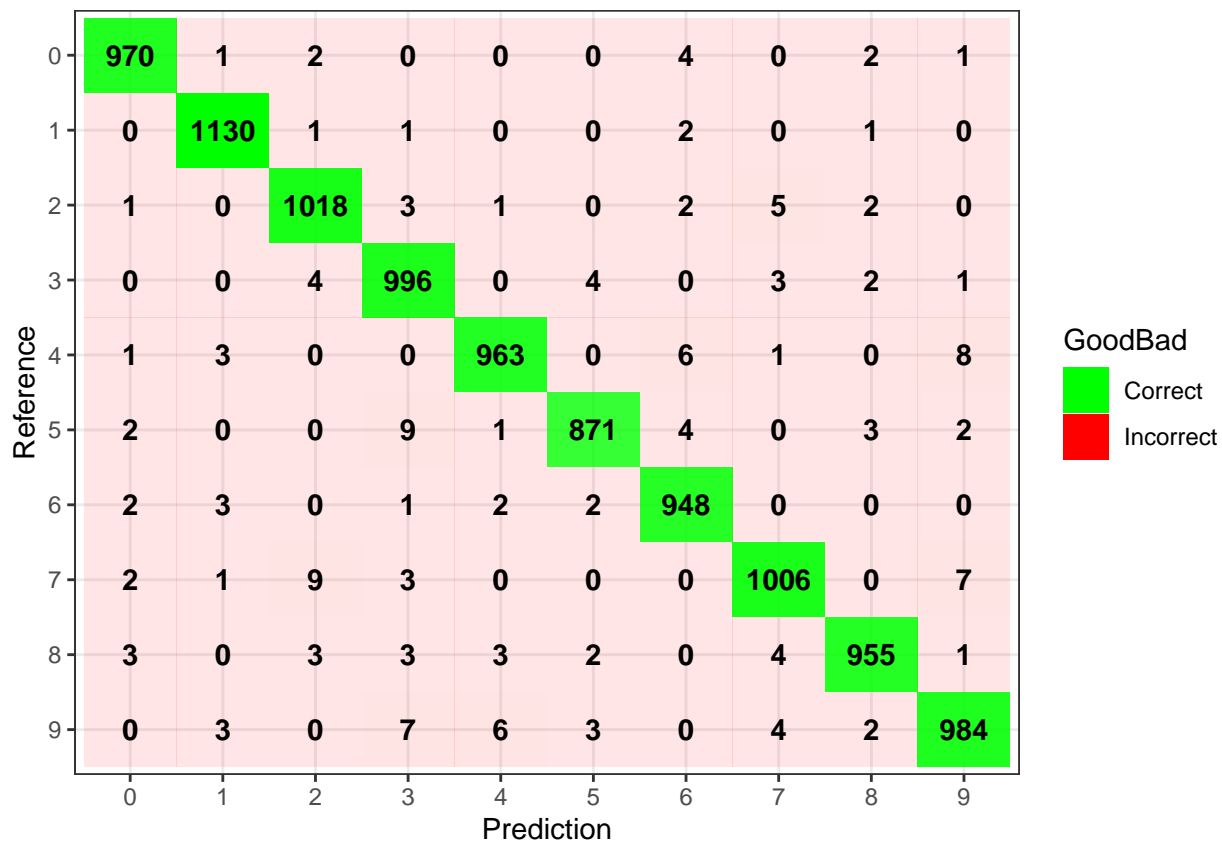
```
par(mfcol = c(5, 5))  
par(mar = c(0, 0, 1.5, 0), xaxs = 'i', yaxs = 'i')  
for (i in 1:25) {  
  img <- mnist$test$x[i, , ]  
  img <- t(apply(img, 2, rev))  
  if (predicted_digits[i] == mnist$test$y[i]) {  
    color <- '#008800'  
  } else {  
    color <- '#bb0000'  
  }  
  image(1:28, 1:28,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

7 (7)	1 (1)	0 (0)	5 (5)	9 (9)
				
2 (2)	4 (4)	6 (6)	9 (9)	6 (6)
				
1 (1)	9 (9)	9 (9)	7 (7)	6 (6)
				
0 (0)	5 (5)	0 (0)	3 (3)	5 (5)
				
4 (4)	9 (9)	1 (1)	4 (4)	4 (4)
				

## Confusion matrix

```
data.frame(table(predicted_digits, 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)))
```



## Boston Housing Prices data set (Example – regression)

### Description

Each record in the database describes a Boston suburb or town. The data was drawn from the Boston Standard Metropolitan Statistical Area (SMSA) in 1970. It has 506 total examples that are split between 404 training examples and 102 test examples. The attributes are defined as follows: 1. CRIM: per capita crime rate by town 2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft. 3. INDUS: proportion of non-retail business acres per town 4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise) 5. NOX: nitric oxides concentration (parts per 10 million) 6. RM: average number of rooms per dwelling 7. AGE: proportion of owner-occupied units built prior to 1940 8. DIS: weighted distances to five Boston employment centers 9. RAD: index of accessibility to radial highways 10. TAX: full-value property-tax rate per \$10,000 11. PTRATIO: pupil-teacher ratio by town 12. B:  $1000(B_k - 0.63)^2$  where  $B_k$  is the proportion of blacks by town 13. LSTAT: % lower status of the population 14. MEDV: Median value of owner-occupied homes in \$1000s

Each one of these input data features is stored using a different scale. Some features are represented by a proportion between 0 and 1, other features are ranges between 1 and 12, some are ranges between 0 and 100, and so on.

### Loading data

```
boston_housing <- dataset_boston_housing()
```



```
c(train_data, train_labels) %<-% boston_housing$train
c(test_data, test_labels) %<-% boston_housing$test
```

```
column_names <- c('CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
                  'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT')
as_tibble(train_data) %>%
  setNames(column_names) %>%
  head() %>%
  knitr::kable('latex', align = rep('c', 10))
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if `.`name_repair` is
## Using compatibility `.`name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
1.2325	0.0	8.14	0	0.538	6.142	91.7	3.977	4	307	21.0	396.9	18.72
0.0218	82.5	2.03	0	0.415	7.610	15.7	6.270	2	348	14.7	395.4	3.11
4.8982	0.0	18.10	0	0.631	4.970	100.0	1.333	24	666	20.2	375.5	3.26
0.0396	0.0	5.19	0	0.515	6.037	34.5	5.985	5	224	20.2	396.9	8.01
3.6931	0.0	18.10	0	0.713	6.376	88.4	2.567	24	666	20.2	391.4	14.65
0.2839	0.0	7.38	0	0.493	5.708	74.3	4.721	5	287	19.6	391.1	11.74

```
train_labels[1:10] # The labels are the house prices in thousands of dollars.
```

```
## [1] 15.2 42.3 50.0 21.1 17.7 18.5 11.3 15.6 15.6 14.4
```

## Prepare data

It's recommended to normalize features that use different scales and ranges. Although the model might converge without feature normalization, it makes training more difficult, and it makes the resulting model more dependant on the choice of units used in the input.

```
# Normalize training data
train_data <- scale(train_data)

# Test data is not used when calculating the mean and std.
# Use means and standard deviations from training set to normalize test set
col_means_train <- attr(train_data, "scaled:center")
col_stddevs_train <- attr(train_data, "scaled:scale")
test_data <- scale(test_data, center = col_means_train, scale = col_stddevs_train)
```

## Defining the model

```
model <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu",
              input_shape = dim(train_data)[2]) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 1)
```

## Summary the model

```
summary(model)

## Model: "sequential_1"
## -----
## Layer (type)                Output Shape          Param #
## -----
## dense_5 (Dense)             (None, 64)            896
## -----
## dense_4 (Dense)             (None, 64)            4160
## -----
## dense_3 (Dense)             (None, 1)             65
## -----
## Total params: 5,121
## Trainable params: 5,121
## Non-trainable params: 0
## -----
```

## Compile the model

```
model %>% compile(loss = 'mse',
  optimizer = optimizer_adam(),
  metrics = list('mean_absolute_error'))
```

- Mean Squared Error (MSE) is a common loss function used for regression problems (different than classification problems):

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2,$$

where  $y_i$  are true values,  $\hat{y}_i$  are predicted values and  $n$  is a sample size.

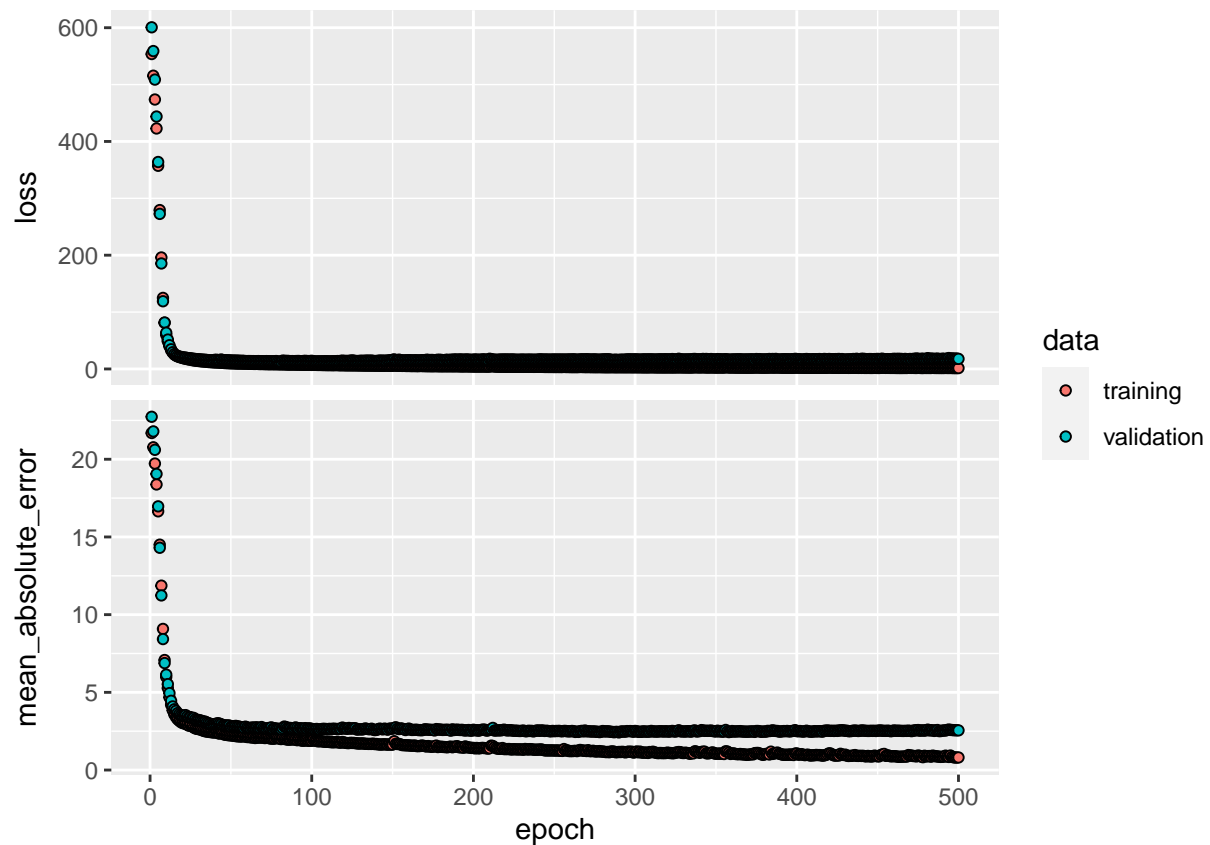
- Similarly, evaluation metrics used for regression differ from classification. A common regression metric is Mean Absolute Error (MAE):

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|.$$

## Training the model

```
model %>% fit(
  train_data,
  train_labels,
  epochs = 500,
  validation_split = 0.2) -> model_reg
```

```
plot(model_reg, smooth = FALSE)
```



This graph shows little improvement in the model after about 200 epochs.

### Evaluate the model

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

## $loss
## [1] 4.794
##
## $mean_absolute_error
## [1] 1.167

model %>% evaluate(test_data, test_labels) # Evaluate the model's performance on the test data

## $loss
## [1] 20.46
##
## $mean_absolute_error
## [1] 2.905
```

### Predictions

```
model %>% predict(test_data) -> predictions # Predicted prices on test data
```