Neural Networks

Preliminary Discussion

Here is an introduction on how to train models based on neural networks using the Keras library. Before running it, install the Keras library in RStudio.

a) Neural Networks with tidymodels We can train a Multilayer Perceptron using the tidymodels framework, which provides all the tools for parameter tuning and cross-validation. This is achieved by calling the `mlp(hidden_units, penalty, dropout, epochs)` function to define the model, where `hidden_units` is the number of neurons in the hidden layer, `penalty` is an L2 penalty constant (Ridge), `dropout` is the dropout rate used in the hidden layer, and `epochs` defines the number of iterations through the training dataset. We can set the neural network engine to the Keras package using `set_engine("keras")`.

```
library(tidymodels)
library(tensorflow)
library(reticulate)
library(keras)

use_condaenv("r-reticulate")

install_tensorflow()
```

```
##
## Installation complete.
```

```
# Initialize the neural network
neural_net <- mlp(hidden_units = 8,
  penalty = 0,
  dropout = 0.2,
  epoch = 100) %>%
  set_engine("keras") %>%
set_mode("classification")

# Train the model
trained_net <- neural_net %>% fit(Species ~ ., iris)
```

However, this approach only allows the use of shallow neural networks (not deep) with one hidden layer. Therefore, let's go over how to train deep neural networks using the Keras library.

b) Deep Neural Networks with Keras In this lesson, we will use the Keras library for deep learning with neural networks. These are the steps typically needed to take to define, train, and test our neural networks using this library:

1. Define the model you want to use by chaining the layers you want to assemble for your network using the pipe operator \textbf{\texttt{%>%}}. The input layer does not need to be explicitly defined. Some of the layers you can use are:

- **`layer_dense(units, activation, kernel_regularizer, input_shape)`:** Defines a layer for a dense neural network (Multilayer Perceptron). The **`units`** argument sets the number of neurons in this layer, **`activation`** defines the activation function used in this layer (e.g. **`relu`, `sigmoid`, `softmax`, `linear`**). The **`kernel_regularizer`** argument can take the result of the **`regularizer_l1(l1)`**, **`regularizer_l2(l2)`**, or **`regularizer_l1_l2(l1, l2)`** functions, which perform L2 (Ridge), L1 (Lasso), or both types of regularization, respectively, where L1 and L2 are the penalty values for each regression type.
- **`layer_dropout(rate)`**, where **`rate`** is the dropout rate. This layer applies dropout to the output of the previous layer.

2. Compile the model. At this point, we need to choose the loss function, the optimization method, and the metric used to display the results after each epoch. The command used here is defined as follows:

   - **`compile(loss, optimizer, metrics)`**, where **`loss`** is the loss function, chosen between the following options:
     - **`mean_squared_error`:** recommended for regression;
     - **`binary_crossentropy`:** recommended for binary classification;
     - **`categorical_crossentropy`:** recommended for classification problems with two or more outputs (requires one-hot encoding).
     - **`sparse_categorical_crossentropy`:** recommended for classification problems with two or more outputs (classes should be represented by integer values ranging from 0 to the number of classes subtracted by one; one-hot-encoding should not be used here).

   The **`optimizer`** parameter is used to choose the optimization algorithm. Common choices for this parameter include **`adam`, `SGD`, `RMSprop`, and `Adagrad`**. The **`metrics`** parameter is the metric used to evaluate the performance on the testing set. It can be chosen between **`accuracy`** and **`mean_squared_error`**.

3. Fit the model to the training data using the **`fit(model, x, y, epochs, batch_size, validation_split, verbose)`** command, where **`model`** is the compiled neural network model from the previous step, **`x`** is the data matrix without outputs, **`y`** is the output vector, **`epochs`** is the number of iterations over the entire training set, **`validation_split`** is the percentage of training data to use for validation, **`verbose`** determines whether to display messages as the learning process progresses, and **`batch_size`** is the number of observations used for each update.

4. To make predictions, use the **`predict(model, x)`** command, which takes the model and the data matrix **`x`** without the output **`y`**.

Here is an example with data from the iris dataset:

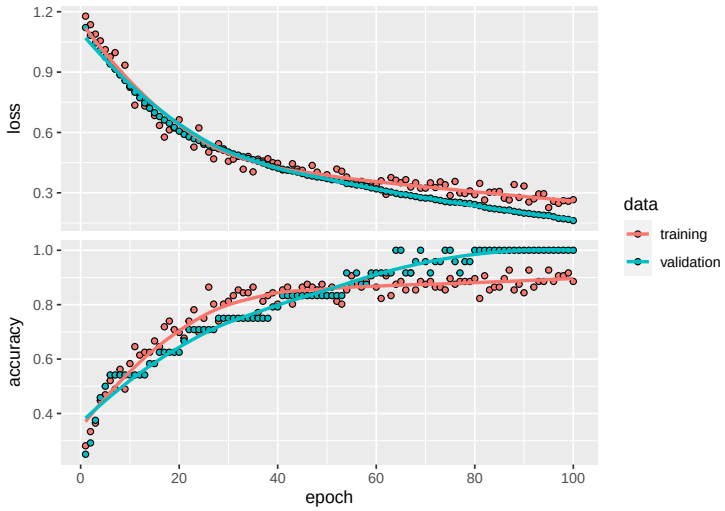
```
df <- as_tibble(iris)

# Split the dataset into training and testing
init_split <- initial_split(df,prop=0.8)
train_df <- training(init_split)
test_df <- testing(init_split)

# Recipe to scale and center the data
rec <- recipe(Species ~ ., train_df) %>%
step_center(all_predictors()) %>%
step_scale(all_predictors())

# Run the recipe on training data
rec_prep <- prep(rec, train_df)
```

```
train_df_prep <- rec_prep %>% juice()

# Load the keras library
library(keras)

# Disable GPU
Sys.setenv("CUDA_VISIBLE_DEVICES" = -1)

set.seed(1)

# Prepare the data

## X should be a matrix with real numbers
train_X <- train_df_prep %>% select(-Species) %>% as.matrix()

## Apply one-hot encoding to the Y target variable
train_Y <- (train_df_prep %>% pull(Species) %>% as.integer()-1) %>%
to_categorical()

# Create the model for the neural network
model <- keras_model_sequential() %>%
## Hidden layer with 8 neurons and ReLU as the activation function
layer_dense(units = 8, activation = 'relu', input_shape = ncol(train_X)) %>%
## Dropout layer with a rate of 0.2
layer_dropout(rate = 0.2) %>%
## Another hidden layer with 8 neurons and ReLU as the activation function
layer_dense(units = 8, activation = 'relu') %>%
## Dropout layer with a rate of 0.2
layer_dropout(rate = 0.2) %>%
## Output layer with 3 units and softmax as the activation function
layer_dense(units = 3, activation = 'softmax')
# Summary of the model
summary(model)
```

```
## Model: "sequential_1"
## _________________________________________________________________
##  Layer (type)                 Output Shape              Param #
## =================================================================
##  dense_5 (Dense)              (None, 8)                 40
##  dropout_2 (Dropout)          (None, 8)                 0
##  dense_4 (Dense)              (None, 8)                 72
##  dropout_1 (Dropout)          (None, 8)                 0
##  dense_3 (Dense)              (None, 3)                 27
## =================================================================
## Total params: 139
## Trainable params: 139
## Non-trainable params: 0
## _________________________________________________________________
```

```
# Compile the model

model %>%
compile(
```

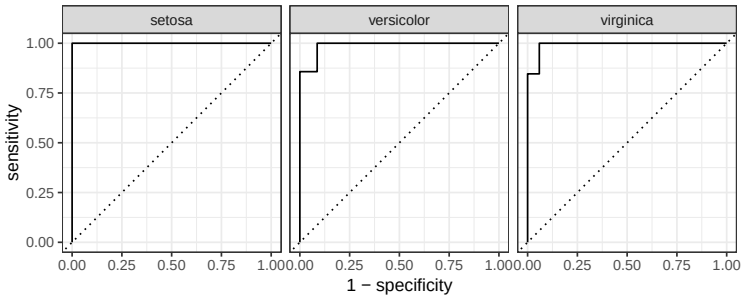
```
loss = "categorical_crossentropy", # Use cross-entropy for a problem with multiple classes
optimizer = "adam",
metrics = "accuracy"
)

# Fit the model to the testing data

history <- model %>%
fit(
x = train_X,
y = train_Y,
epochs = 100,
batch_size = 5,
validation_split = 0.2,
verbose = 2
)

# Visualize the training process
plot(history)
```




```
# Evaluate the results using testing data

# Prepare the testing set
test_df_prep <- rec_prep %>% bake(new_data = test_df)
test_X <- test_df_prep %>% select(-Species) %>% as.matrix()
test_Y <- (test_df_prep %>% pull(Species) %>% as.integer()-1) %>%
to_categorical()

# Calculate the accuracy score for the testing set
model %>% evaluate(test_X, test_Y)
```

```
##      loss  accuracy
## 0.1816162 0.9000000
```

```r
# Making predictions
predictions <- model %>% predict(test_X) %>% as_tibble()

# New column with class labels
class_labels <- levels(test_df$Species)
colnames(predictions) <- paste0(".pred_", class_labels)


# Prepare a table for evaluation using Yardstick
test_pred <- predictions %>%
bind_cols(test_df_prep %>% select(Species)) %>%
mutate( .pred = class_labels[pmap_int(as.list(predictions),
function(...)which.max(c(...)))]) %>%
mutate( .pred = factor(.pred,levels=class_labels))

# a) Plot the ROC curve
roc_curve(test_pred,starts_with(".pred_"),truth = Species) %>%
autoplot(curva_roc)
```




```r
# b) Create a confusion matrix
conf_mat(test_pred, Species, .pred)
```

```
##             Truth
## Prediction   setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0          6         2
##   virginica       0          1        11
```

```r
# c) Evaluate the accuracy, precision, and recall metrics
accuracy(test_pred, Species, .pred)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass       0.9
```

```
precision(test_pred, Species, .pred)
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision macro          0.889
```

```
recall(test_pred, Species, .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 recall  macro          0.901
```

Exercise 01

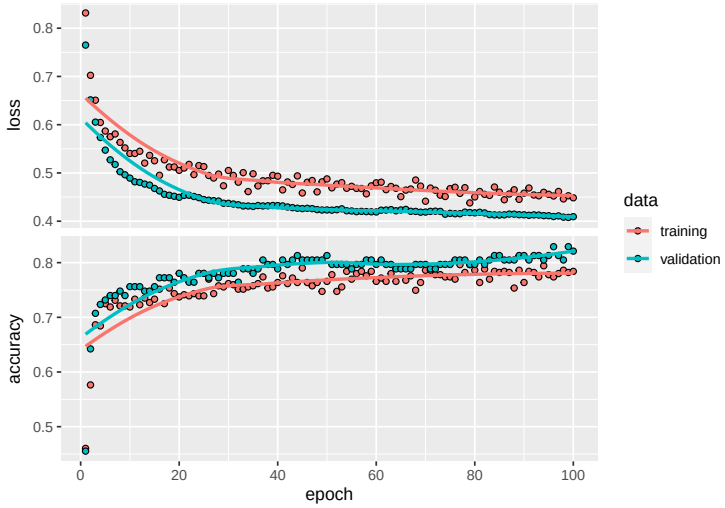
Repeat the steps from the above tutorial to solve the classification problem for the Pima Indians Diabetes dataset from the mlbench library. Build a Multilayer Perceptron to find the best solution.

```
# Apply the above process to predict Diabetes
library(mlbench)
data(PimaIndiansDiabetes)
df <- as_tibble(PimaIndiansDiabetes)

# Split the dataset into training and testing sets
init_split <- initial_split(df,prop=0.8)
train_df <- training(init_split)
test_df <- testing(init_split)

# Set up the recipe
rec <- recipe(diabetes ~ ., train_df) %>%
step_center(all_predictors()) %>%
step_scale(all_predictors())

# Run the recipe on training data
rec_prep <- prep(rec, train_df)
train_df_prep <- rec_prep %>% juice()


Sys.setenv("CUDA_VISIBLE_DEVICES" = -1)

set.seed(1)

# Prepare the data
train_X <- train_df_prep %>% select(-diabetes) %>% as.matrix()
train_Y <- (train_df_prep %>% pull(diabetes) %>% as.integer()-1) %>%
to_categorical()

# Create the model
model <- keras_model_sequential() %>%
layer_dense(units = 8, activation = 'relu', input_shape = ncol(train_X)) %>%
```

```
layer_dropout(rate = 0.2) %>%
layer_dense(units = 8, activation = 'relu') %>%
layer_dropout(rate = 0.2) %>%
# Changing to 2 neurons on the output layer and using softmax as the activation function
layer_dense(units = 2, activation = 'softmax')
summary(model)
```

```
## Model: "sequential_1"
## _________________________________________________________________
##  Layer (type)                 Output Shape              Param #
## =================================================================
##  dense_5 (Dense)              (None, 8)                 72
##  dropout_2 (Dropout)          (None, 8)                 0
##  dense_4 (Dense)              (None, 8)                 72
##  dropout_1 (Dropout)          (None, 8)                 0
##  dense_3 (Dense)              (None, 2)                 18
## =================================================================
## Total params: 162
## Trainable params: 162
## Non-trainable params: 0
## _________________________________________________________________
```

```
# Compile the model
model %>%
compile(
loss = "categorical_crossentropy",
optimizer = "adam",
metrics = "accuracy"
)


# Fit the model to the testing set
history <- model %>%
fit(
x = train_X,
y = train_Y,
epochs = 100,
batch_size = 5,
validation_split = 0.2,
verbose = 2
)

plot(history)
```




```
# Evaluate the results using testing data
test_df_prep <- rec_prep %>% bake(new_data = test_df)
test_X <- test_df_prep %>% select(-diabetes) %>% as.matrix()
test_Y <- (test_df_prep %>% pull(diabetes) %>% as.integer()-1) %>%
to_categorical()

model %>% evaluate(test_X, test_Y)
```

```
##      loss  accuracy
## 0.5101292 0.6818182
```

```
predictions <- model %>% predict(test_X) %>% as_tibble()

class_labels <- levels(test_df$diabetes)
colnames(predictions) <- paste0(".pred_",class_labels)

test_pred <- predictions %>%
bind_cols(test_df_prep %>% select(diabetes)) %>%
mutate( .pred = class_labels[pmap_int(as.list(predictions),
function(...)which.max(c(...)))]) %>%
mutate( .pred = factor(.pred, levels=class_labels))

# Confusion matrix
conf_mat(test_pred, diabetes, .pred)
```

```
##           Truth
## Prediction neg pos
##        neg  67  33
##        pos  16  38
```

```
# Performance metrics
accuracy(test_pred, diabetes, .pred)
```

```
## # A tibble: 1 x 3
```

```
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.682
```

```
precision(test_pred, diabetes, .pred)
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary          0.67
```

```
recall(test_pred, diabetes, .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 recall  binary         0.807
```

Exercise 02

In the list on Parameter Tuning, we used a linear model to fit the Salaries data from the car library. The observed results were not very good, and the error rate (measured by RMSE - Root Mean Squared Error) was quite high. In this exercise, we will use a neural network (from the Keras library) to solve this regression problem. We will use the metric 'MAE' (Mean Absolute Error). Can you create a neural network with a low error rate?

Hint 1: Note that 'rank,' 'discipline,' and 'sex' are categorical variables. Create a recipe to add dummy variables (you can use One-Hot-Encoding here if you prefer).

Hint 2: Since this is a regression problem, use a single unit in the output layer with the 'linear' activation function (i.e., no activation function). Also, when compiling the model, use the 'mean_squared_error' loss function and 'MAE' (Mean Absolute Error) as the metric.

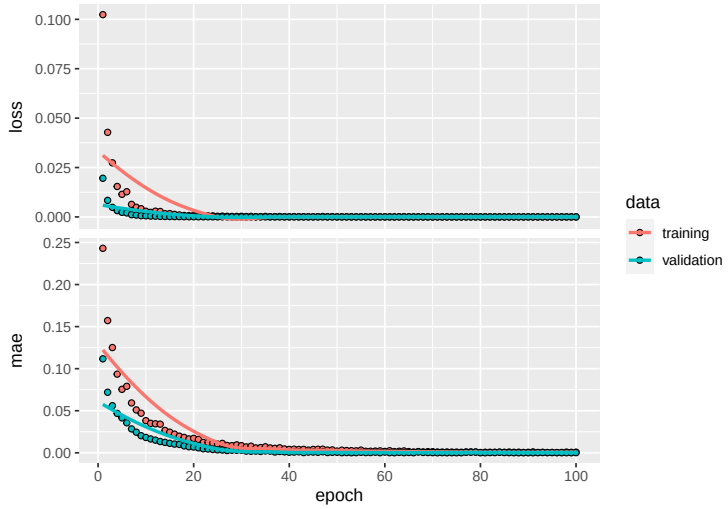
```
## Repeat the process to predict "Salary"
library(car)
df <- as_tibble(Salaries)

# Split the dataset into training and testing sets
init_split <- initial_split(df, prop=0.8)
train_df <- training(init_split)
test_df <- testing(init_split)


# Create the recipe
rec <- recipe(salary ~ ., df) %>%
step_scale(all_predictors(),-all_nominal()) %>%
step_center(all_predictors(),-all_nominal()) %>%
# Turn categorical features into dummy variables
step_dummy(all_nominal())

# Run the recipe on training data
rec_prep <- prep(rec, train_df)
```

```
train_df_prep <- rec_prep %>% juice()

Sys.setenv("CUDA_VISIBLE_DEVICES" = -1)
set.seed(1)

# Prepare the data
train_X <- train_df_prep %>% select(-salary) %>% as.matrix()
train_Y <- (train_df_prep %>% pull(salary) %>% as.integer()-1) %>%
to_categorical()

# Create the model
model <- keras_model_sequential() %>%
layer_dense(units = 8, activation = 'relu', input_shape = ncol(train_X)) %>%
layer_dropout(rate = 0.2) %>%
layer_dense(units = 8, activation = 'relu') %>%
layer_dropout(rate = 0.2) %>%
# Changing to 1 neuron on the output layer and passing 'linear' as the activation function
layer_dense(units = 1, activation = 'linear')
summary(model)
```

```
## Model: "sequential"
## _________________________________________________________________________
##  Layer (type)                   Output Shape                Param #
## =========================================================================
##  dense_2 (Dense)                (None, 8)                   56
##  dropout_1 (Dropout)            (None, 8)                   0
##  dense_1 (Dense)                (None, 8)                   72
##  dropout (Dropout)              (None, 8)                   0
##  dense (Dense)                  (None, 1)                   9
## =========================================================================
## Total params: 137
## Trainable params: 137
## Non-trainable params: 0
## _________________________________________________________________________
```

```
# Compile the model
model %>%
compile(
loss = "mean_squared_error",
optimizer = "adam",
metrics = "mae"
)

# Fit the model to the testing set
history <- model %>%
fit(
x = train_X,
y = train_Y,
epochs = 100,
batch_size = 5,
validation_split = 0.2,
verbose = 2
)
```

```
plot(history)
```




```
# Evaluate the results using testing data
test_df_prep <- rec_prep %>% bake(new_data = test_df)
test_X <- test_df_prep %>% select(-salary) %>% as.matrix()
test_Y <- (test_df_prep %>% pull(salary) %>% as.integer()-1)

model %>% evaluate(test_X, test_Y)
```

```
##         loss          mae
## 1.349098e+10 1.135783e+05
```

```
predictions <- model %>% predict(test_X) %>% as_tibble()
```

Exercise 03

This exercise is a tutorial for Keras in an image classification problem using a Multilayer Perceptron. Note: convolutional neural networks are generally recommended for this type of problem. They will be covered in the next exercise.

Considering the code and the dataset provided below, build a neural network that classifies images, then train this neural network and evaluate the model's accuracy. Train it for 5 epochs and plot a graph for both the loss vs. epoch and the accuracy score vs. epoch. Before running the experiments, ensure that the Keras library is installed.

```
library(keras)

mnist <- dataset_mnist()
mnist$train$x <- mnist$train$x/255
mnist$test$x <- mnist$test$x/255

model <- keras_model_sequential() %>%
layer_flatten(input_shape = c(28, 28)) %>%
```

```
# Hidden layer with 128 neurons and ReLU as the activation function
layer_dense(units = 128, activation = "relu") %>%
# Dropout layer with a rate of 0.2
layer_dropout(rate = 0.2) %>%
# Output layer with 10 neurons and softmax as the activation function
layer_dense(units = 10, activation = "softmax")
summary(model)
```

```
## Model: "sequential_1"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  flatten (Flatten)                  (None, 784)                     0
##  dense_4 (Dense)                    (None, 128)                     100480
##  dropout_2 (Dropout)                (None, 128)                     0
##  dense_3 (Dense)                    (None, 10)                      1290
## =========================================================================
## Total params: 101,770
## Trainable params: 101,770
## Non-trainable params: 0
## _________________________________________________________________________
```

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

# Fit the model to the testing set
model %>%
fit(
x = mnist$train$x, y = mnist$train$y,
epochs = 5,
validation_split = 0.3,
verbose = 2
)

# Make predictions
predictions <- predict(model, mnist$test$x)
head(predictions, 2)
```

```
##              [,1]         [,2]         [,3]         [,4]         [,5]
## [1,] 4.099951e-07 5.038867e-09 3.331366e-06 3.799899e-04 6.036021e-11
## [2,] 2.333100e-07 3.689172e-04 9.995874e-01 3.580976e-05 5.529931e-12
##              [,6]         [,7]         [,8]         [,9]        [,10]
## [1,] 4.437537e-07 2.367015e-11 9.996055e-01 7.160029e-07 9.592445e-06
## [2,] 1.138145e-06 1.895188e-07 1.001544e-11 6.320225e-06 7.101601e-12
```

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

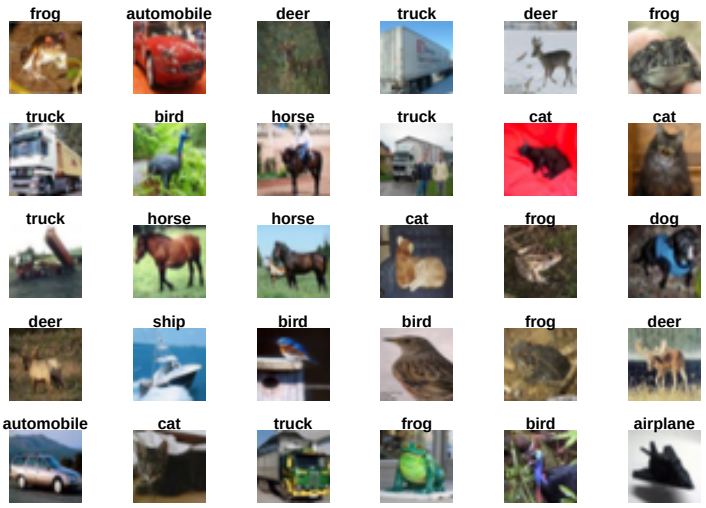
```
##       loss   accuracy
## 0.08039267 0.97579998
```

Exercise 04

This exercise is a tutorial on how to solve an image classification problem using convolutional neural networks from the Keras library.

For this problem, we will load data from the CIFAR-10 dataset, containing 60,000 colored images divided into 10 classes (6,000 images per class). The dataset is split into 50,000 images for training and 10,000 for testing. Implement and train a model that correctly classifies the images. Train it for 10 epochs and plot a graph for Loss vs. Epoch and Accuracy vs. Epoch.

```
# Load the data
cifar <- dataset_cifar10()
class_names <- c('airplane', 'automobile', 'bird', 'cat', 'deer',
'dog', 'frog', 'horse', 'ship', 'truck')
index <- 1:30

# Load data from the CIFAR-10
par(mfcol = c(5,6), mar = rep(1, 4), oma = rep(0.2, 4))
cifar$train$x[index,,,] %>%
purrr::array_tree(1) %>%
purrr::set_names(class_names[cifar$train$y[index] + 1]) %>%
purrr::map(as.raster, max = 255) %>%
purrr::iwalk(~{plot(.x); title(.y)})
```

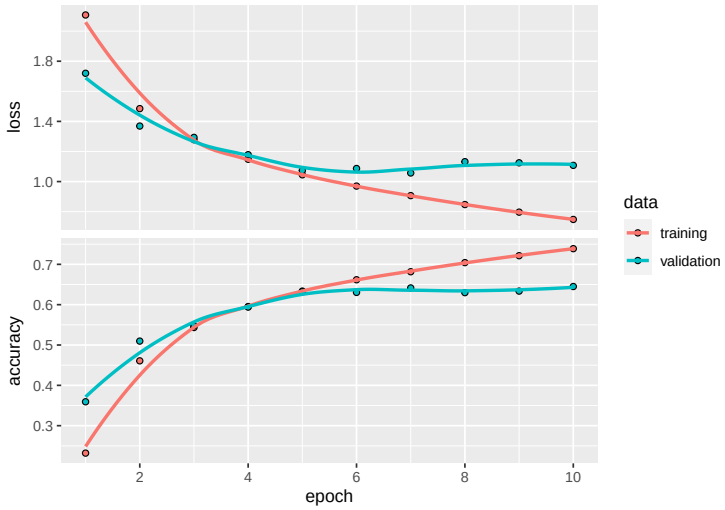



```
# Define the keras model
model <- keras_model_sequential() %>%
layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "relu",
input_shape = c(32,32,3)) %>%
layer_max_pooling_2d(pool_size = c(2,2)) %>%
layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
layer_max_pooling_2d(pool_size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

```
layer_dense(units = 64, activation = "relu") %>%
layer_dense(units = 10, activation = "softmax")
summary(model)
```

```
## Model: "sequential_4"
## _________________________________________________________________
##  Layer (type)                 Output Shape              Param #
## =================================================================
##  conv2d_2 (Conv2D)            (None, 30, 30, 32)        896
##  max_pooling2d_1 (MaxPooling2D  (None, 15, 15, 32)      0
##  )
##  conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
##  max_pooling2d (MaxPooling2D)  (None, 6, 6, 64)         0
##  conv2d (Conv2D)              (None, 4, 4, 64)          36928
##  flatten_1 (Flatten)          (None, 1024)              0
##  dense_12 (Dense)             (None, 64)                65600
##  dense_11 (Dense)             (None, 10)                650
## =================================================================
## Total params: 122,570
## Trainable params: 122,570
## Non-trainable params: 0
## _________________________________________________________________
```

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

# Fit the model on training data
history <- model %>%
fit(
x = cifar$train$x,
y = cifar$train$y,
epochs = 10,
validation_data = unname(cifar$test),
verbose = 2
)

plot(history)
```


loss
1.8
1.4
1.0
accuracy
0.7
0.6
0.5
0.4
0.3
2
4
6
8
10
epoch
data
training
validation