Building a simple neural network using Keras and Tensorflow - Updated

**Update:** The original code has been updated to use the *tidymodels* init\_split() function, rather than using the indicies method which originally used setdiff, which now may have a conflict between base R and the tidyverse.

Thank you

A big thank you to Leon Jessen for posting his code on github.

[Building a simple neural network using Keras and Tensorflow](https://github.com/leonjessen/keras_tensorflow_on_iris/blob/master/README.md)

I have forked his project on github and put his code into an R Notebook so we can run it in class.

### Motivation

The following is a minimal example for building your first simple artificial neural network using Keras and TensorFlow for R.

[TensorFlow for R by Rstudio lives here](https://tensorflow.rstudio.com/keras/).

### Gettings started - Install Keras and TensorFlow for R

You can install the Keras for R package from CRAN as follows:

# install.packages("keras")

TensorFlow is the default backend engine. TensorFlow and Keras can be installed as follows:

# library(keras)
# install\_keras()

Naturally, we will also need **Tidyverse**.

# Install from CRAN
# install.packages("tidyverse")

# Or the development version from GitHub
# install.packages("devtools")
# devtools::install\_github("hadley/tidyverse")

Once installed, we simply load the libraries.

library("keras")
suppressMessages(library("tidyverse"))

### Artificial Neural Network Using the Iris Data Set

Right, let’s get to it!

### Data

The famous (Fisher’s or Anderson’s) *iris* data set contains a total of 150 observations of 4 input features *Sepal.Length*, *Sepal.Width*, *Petal.Length* and *Petal.Width* and 3 output classes *setosa* *versicolor* and *virginica*, with 50 observations in each class. The distributions of the feature values looks like so:

iris\_tib <- as\_tibble(iris)
iris\_tib

## # A tibble: 150 x 5
## Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## <dbl> <dbl> <dbl> <dbl> <fct>
## 1 5.1 3.5 1.4 0.2 setosa
## 2 4.9 3 1.4 0.2 setosa
## 3 4.7 3.2 1.3 0.2 setosa
## 4 4.6 3.1 1.5 0.2 setosa
## 5 5 3.6 1.4 0.2 setosa
## 6 5.4 3.9 1.7 0.4 setosa
## 7 4.6 3.4 1.4 0.3 setosa
## 8 5 3.4 1.5 0.2 setosa
## 9 4.4 2.9 1.4 0.2 setosa
## 10 4.9 3.1 1.5 0.1 setosa
## # … with 140 more rows

iris\_tib %>% pivot\_longer(names\_to = "feature", values\_to = "value", -Species) %>%
 ggplot(aes(x = feature, y = value, fill = Species)) +
 geom\_violin(alpha = 0.5, scale = "width") +
 theme\_bw()



Our aim is to connect the 4 input features to the correct output class using an artificial neural network. For this task, we have chosen the following simple architecture with one input layer with 4 neurons (one for each feature), one hidden layer with 4 neurons and one output layer with 3 neurons (one for each class), all fully connected.



architecture\_visualisation.png

Our artificial neural network will have a total of 35 parameters: 4 for each input neuron connected to the hidden layer, plus an additional 4 for the associated first bias neuron and 3 for each of the hidden neurons connected to the output layer, plus an additional 3 for the associated second bias neuron, i.e. $4×4+4+4×3+3=35$

### Prepare data

We start with slightly wrangling the iris data set by renaming and scaling the features and converting character labels to numeric.

set.seed(265509)
nn\_dat <- iris\_tib %>%
 mutate(sepal\_length = scale(Sepal.Length),
 sepal\_width = scale(Sepal.Width),
 petal\_length = scale(Petal.Length),
 petal\_width = scale(Petal.Width),
 class\_label = as.numeric(Species) - 1) %>%
 select(sepal\_length, sepal\_width, petal\_length, petal\_width, class\_label)

nn\_dat %>% head()

## # A tibble: 6 x 5
## sepal\_length[,1] sepal\_width[,1] petal\_length[,1] petal\_width[,1] class\_label
## <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 -0.898 1.02 -1.34 -1.31 0
## 2 -1.14 -0.132 -1.34 -1.31 0
## 3 -1.38 0.327 -1.39 -1.31 0
## 4 -1.50 0.0979 -1.28 -1.31 0
## 5 -1.02 1.25 -1.34 -1.31 0
## 6 -0.535 1.93 -1.17 -1.05 0

Then, we create indices for splitting the iris data into a training and a test data set. We set aside 20% of the data for testing.

library(tidymodels)

## ── Attaching packages ────────────────────────────────────── tidymodels 0.1.2 ──

## ✓ broom 0.7.5 ✓ recipes 0.1.15
## ✓ dials 0.0.9 ✓ rsample 0.0.9
## ✓ infer 0.5.4 ✓ tune 0.1.2
## ✓ modeldata 0.1.0.9000 ✓ workflows 0.2.1
## ✓ parsnip 0.1.5 ✓ yardstick 0.0.7

## ── Conflicts ───────────────────────────────────────── tidymodels\_conflicts() ──
## x scales::discard() masks purrr::discard()
## x dplyr::filter() masks stats::filter()
## x recipes::fixed() masks stringr::fixed()
## x yardstick::get\_weights() masks keras::get\_weights()
## x dplyr::lag() masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step() masks stats::step()

set.seed(364)
n <- nrow(nn\_dat)
n

## [1] 150

iris\_parts <- nn\_dat %>%
 initial\_split(prop = 0.8)

train <- iris\_parts %>%
 training()

test <- iris\_parts %>%
 testing()

list(train, test) %>%
 map\_int(nrow)

## [1] 121 29

n\_total\_samples <- nrow(nn\_dat)

n\_train\_samples <- nrow(train)

n\_test\_samples <- nrow(test)

### Create training and test data

**Note** that the functions in the keras package are expecting the data to be in a matrix object and not a tibble. So as.matrix is added at the end of each line.

x\_train <- train %>% select(-class\_label) %>% as.matrix()
y\_train <- train %>% select(class\_label) %>% as.matrix() %>% to\_categorical()

x\_test <- test %>% select(-class\_label) %>% as.matrix()
y\_test <- test %>% select(class\_label) %>% as.matrix() %>% to\_categorical()

dim(y\_train)

## [1] 121 3

dim(y\_test)

## [1] 29 3

### Set Architecture

With the data in place, we now set the architecture of our neural network.

model <- keras\_model\_sequential()
model %>%
 layer\_dense(units = 4, activation = 'relu', input\_shape = 4) %>%
 layer\_dense(units = 3, activation = 'softmax')
model %>% summary

## Model: "sequential"
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
## Layer (type) Output Shape Param #
## ================================================================================
## dense\_1 (Dense) (None, 4) 20
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
## dense (Dense) (None, 3) 15
## ================================================================================
## Total params: 35
## Trainable params: 35
## Non-trainable params: 0
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Next, the architecture set in the model needs to be compiled.

model %>% compile(
 loss = 'categorical\_crossentropy',
 optimizer = optimizer\_rmsprop(),
 metrics = c('accuracy')
)

### Train the Artificial Neural Network

Lastly we fit the model and save the training progress in the *history* object.

**Try** changing the *validation\_split* from 0 to 0.2 to see the *validation\_loss*.

history <- model %>% fit(
 x = x\_train, y = y\_train,
 epochs = 200,
 batch\_size = 20,
 validation\_split = 0.2
)

plot(history) +
 ggtitle("Training a neural network based classifier on the iris data set") +
 theme\_bw()

## `geom\_smooth()` using formula 'y ~ x'



### Evaluate Network Performance

The final performance can be obtained like so.

perf <- model %>% evaluate(x\_test, y\_test)
print(perf)

## loss accuracy
## 0.3694951 0.8275862

For the next plot the predicted and true values need to be in a vector. Note that the true values need to be unlisted before putting them into a numeric vector.

classes <- iris %>% pull(Species) %>% unique()
y\_pred <- model %>% predict\_classes(x\_test)
y\_true <- test %>% select(class\_label) %>% unlist() %>% as.numeric()

tibble(y\_true = classes[y\_true + 1], y\_pred = classes[y\_pred + 1],
 Correct = ifelse(y\_true == y\_pred, "Yes", "No") %>% factor) %>%
 ggplot(aes(x = y\_true, y = y\_pred, colour = Correct)) +
 geom\_jitter() +
 theme\_bw() +
 ggtitle(label = "Classification Performance of Artificial Neural Network",
 subtitle = str\_c("Accuracy = ",round(perf[2],3)\*100,"%")) +
 xlab(label = "True iris class") +
 ylab(label = "Predicted iris class")



library(gmodels)

CrossTable(y\_pred, y\_true,
 prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
 dnn = c('predicted', 'actual'))

##
##
## Cell Contents
## |-------------------------|
## | N |
## | N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table: 29
##
##
## | actual
## predicted | 0 | 1 | 2 | Row Total |
## -------------|-----------|-----------|-----------|-----------|
## 0 | 8 | 0 | 0 | 8 |
## | 1.000 | 0.000 | 0.000 | |
## -------------|-----------|-----------|-----------|-----------|
## 1 | 0 | 12 | 5 | 17 |
## | 0.000 | 1.000 | 0.556 | |
## -------------|-----------|-----------|-----------|-----------|
## 2 | 0 | 0 | 4 | 4 |
## | 0.000 | 0.000 | 0.444 | |
## -------------|-----------|-----------|-----------|-----------|
## Column Total | 8 | 12 | 9 | 29 |
## | 0.276 | 0.414 | 0.310 | |
## -------------|-----------|-----------|-----------|-----------|
##
##

### Conclusion

I hope this illustrated just how easy it is to get started building artificial neural network using Keras and TensorFlow in R. With relative ease, we created a 3-class predictor with an accuracy of 100%. This was a basic minimal example. The network can be expanded to create Deep Learning networks and also the entire TensorFlow API is available.

Enjoy and Happy Learning!

Leon

**Thanks again Leon, this was awesome!!!**