Building a simple neural network using Keras and Tensorflow

JSM 2018: Poster 181 - Classroom Demonstration: Deep Learning for Classification and Regression, Introduction to GPU Computing

Eric A. Suess

Department Of Statistics and Biostatistics

CSU East Bay

eric.suess@csueastbay.edu

Thank you

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

Building a simple neural network using Keras and Tensorflow

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.

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:

```r
# 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

```r
library("keras")
suppressMessages(library("tidyverse"))
```

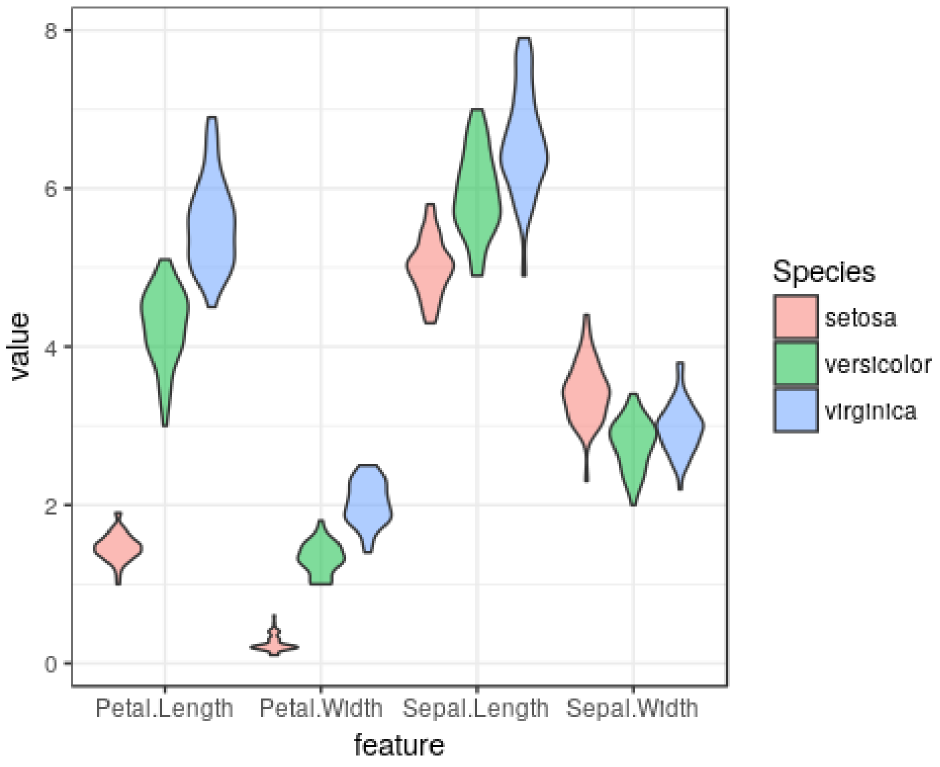
Artificial Neural Network Using the Iris Data Set

Right, let's get to it!

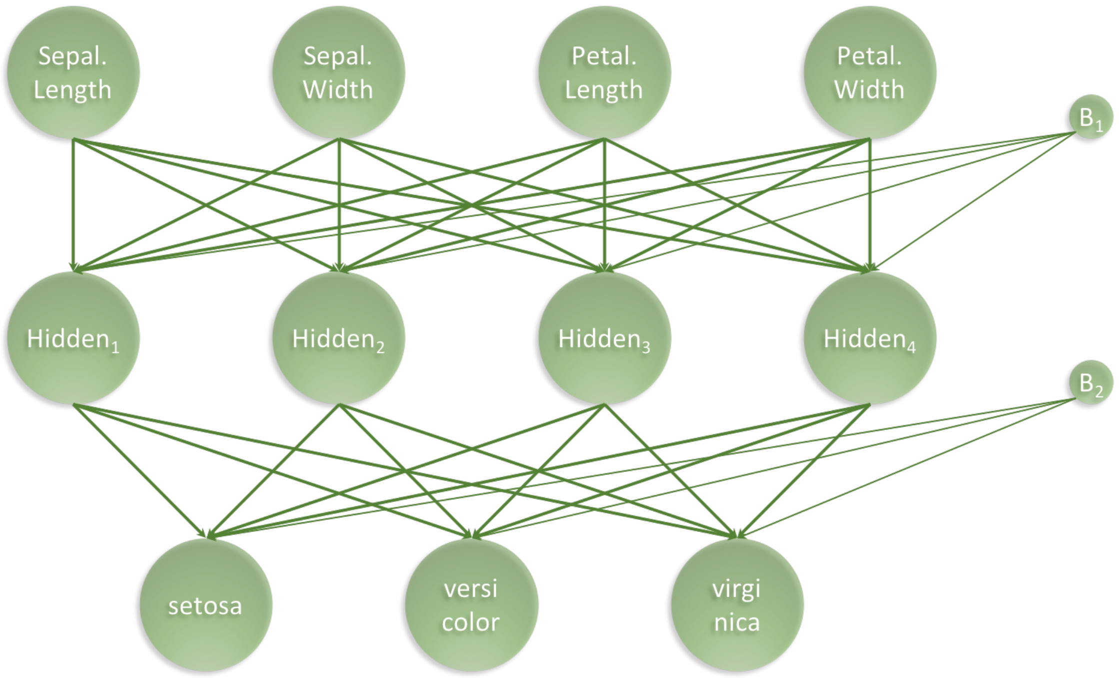
Data

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:

```r
iris %>% as_tibble %>% gather(feature, 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. $4 \cdot 4 + 4 + 4 \cdot 3 + 3 = 35$

Prepare data

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

```r
set.seed(265509)
nn_dat <- iris %>% as_tibble %>%
  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(3)
```

```
## # A tibble: 3 x 5
##   sepal_length sepal_width petal_length petal_width 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.
```

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:

```
test_fraction   <- 0.20
n_total_samples <- nrow(nn_dat)
n_train_samples <- ceiling((1 - test_fraction) * n_total_samples)
train_indices   <- sample(n_total_samples, n_train_samples)
n_test_samples  <- n_total_samples - n_train_samples
test_indices    <- setdiff(seq(1, n_train_samples), train_indices)
```

Based on the indices, we can now create training and test data

```
x_train <- nn_dat %>% select(-class_label) %>% as.matrix %>% .
[train_indices,]
y_train <- nn_dat %>% pull(class_label) %>% .[train_indices] %>%
to_categorical(3)
x_test  <- nn_dat %>% select(-class_label) %>% as.matrix %>% .
[test_indices,]
y_test  <- nn_dat %>% pull(class_label) %>% .[test_indices] %>%
to_categorical(3)
```

Set Architecture

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

```
model <- keras_model_sequential()
model %>%
  layer_dense(units = 4, activation = 'relu', input_shape = 4) %>%
  layer_dense(units = 3, activation = 'softmax')
model %>% summary
```

```
##
___________________________________________________________________________
## Layer (type)                 Output Shape              Param #
##
===========================================================================
## dense_1 (Dense)              (None, 4)                 20
##
___________________________________________________________________________
## dense_2 (Dense)              (None, 3)                 15
##
===========================================================================
## Total params: 35
## Trainable params: 35
## Non-trainable params: 0
##
```

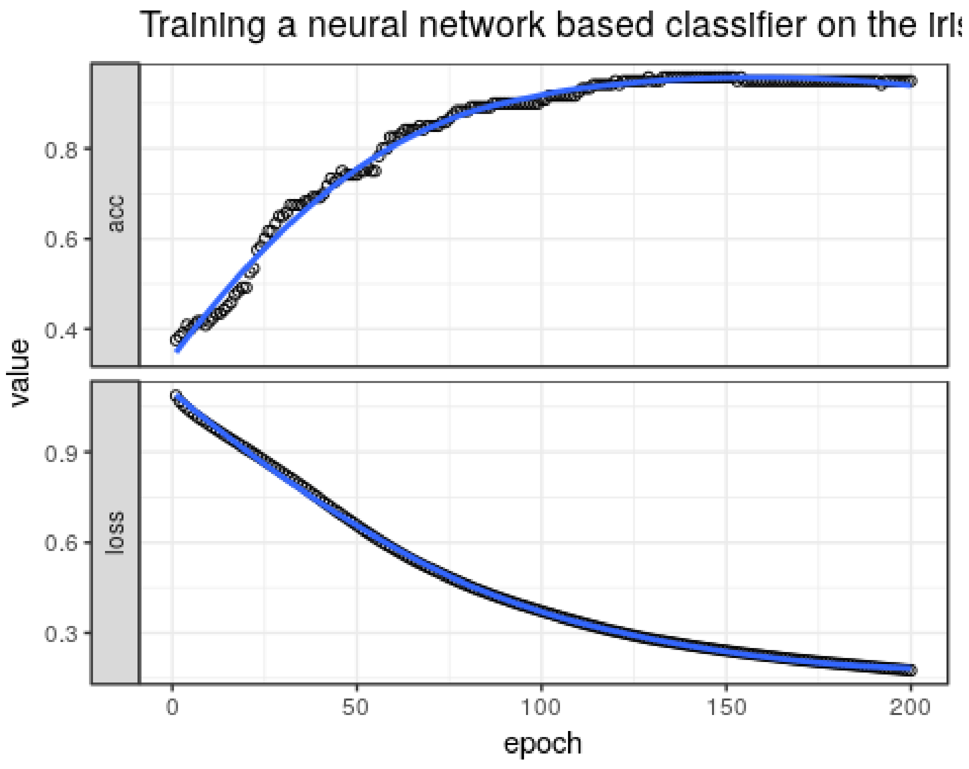
Next, the architecture set in the model needs to be compiled:

```
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 progres in the history object:

```
history <- model %>% fit(
  x = x_train, y = y_train,
  epochs = 200,
  batch_size = 20,
  validation_split = 0
)
plot(history) +
  ggtitle("Training a neural network based classifier on the iris data set") +
  theme_bw()
```



Evaluate Network Performance

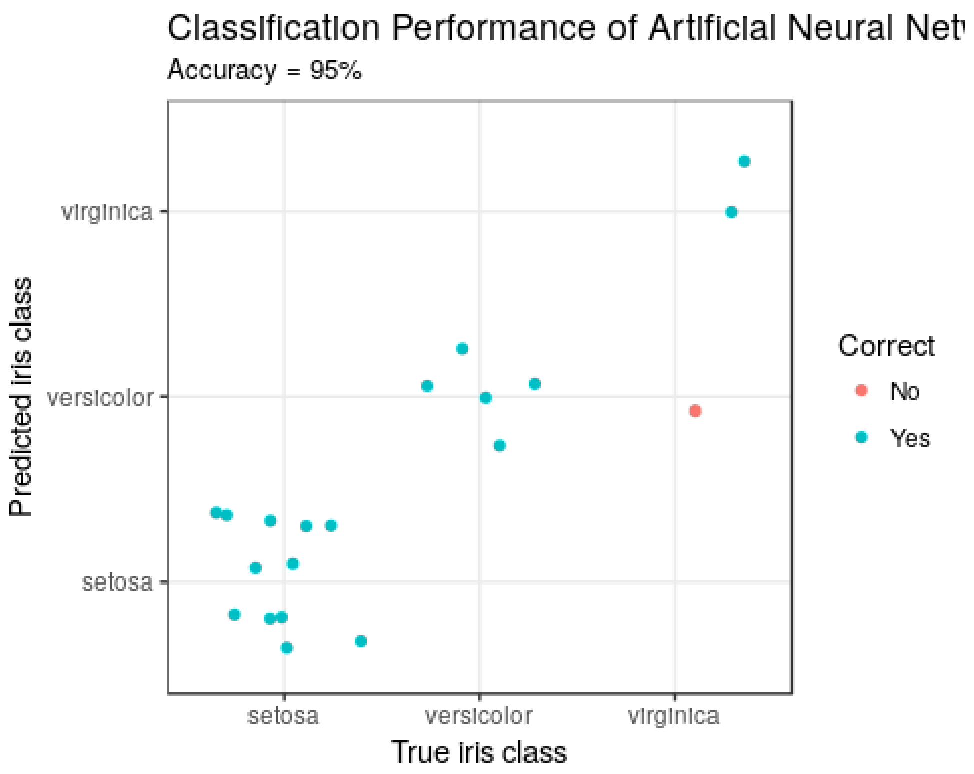
The final performance can be obtained like so:

```
perf <- model %>% evaluate(x_test, y_test)
print(perf)

## $loss
## [1] 0.1339914
##
## $acc
## [1] 0.95

classes <- iris %>% as_tibble %>% pull(Species) %>% unique
y_pred  <- model %>% predict_classes(x_test)
y_true  <- nn_dat %>% pull(class_label) %>% .[test_indices]

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$acc,3)*100,"%")) +
  xlab(label = "True iris class") +
  ylab(label = "Predicted iris class")
```



```r
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:  20 
## 
## 
##              | actual 
##    predicted |         0 |         1 |         2 | Row Total | 
## -------------|-----------|-----------|-----------|-----------|
##            0 |        12 |         0 |         0 |        12 | 
##              |     1.000 |     0.000 |     0.000 |           | 
## -------------|-----------|-----------|-----------|-----------|
##            1 |         0 |         5 |         1 |         6 | 
##              |     0.000 |     1.000 |     0.333 |           | 
## -------------|-----------|-----------|-----------|-----------|
##            2 |         0 |         0 |         2 |         2 | 
##              |     0.000 |     0.000 |     0.667 |           | 
## -------------|-----------|-----------|-----------|-----------|
## Column Total |        12 |         5 |         3 |        20 | 
##              |     0.600 |     0.250 |     0.150 |           | 
## -------------|-----------|-----------|-----------|-----------|
## 
## 
```

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 awsome!!!