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](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"))

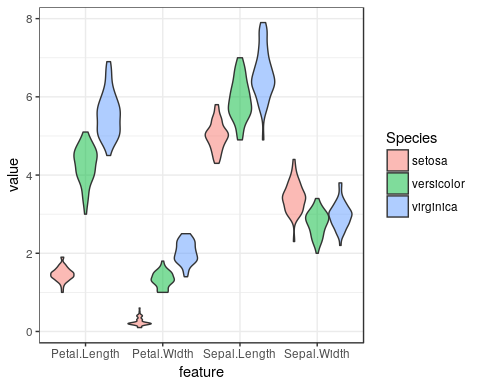
## Artificial Neural Network Using the Iris Data Set

Right, let's get to it!

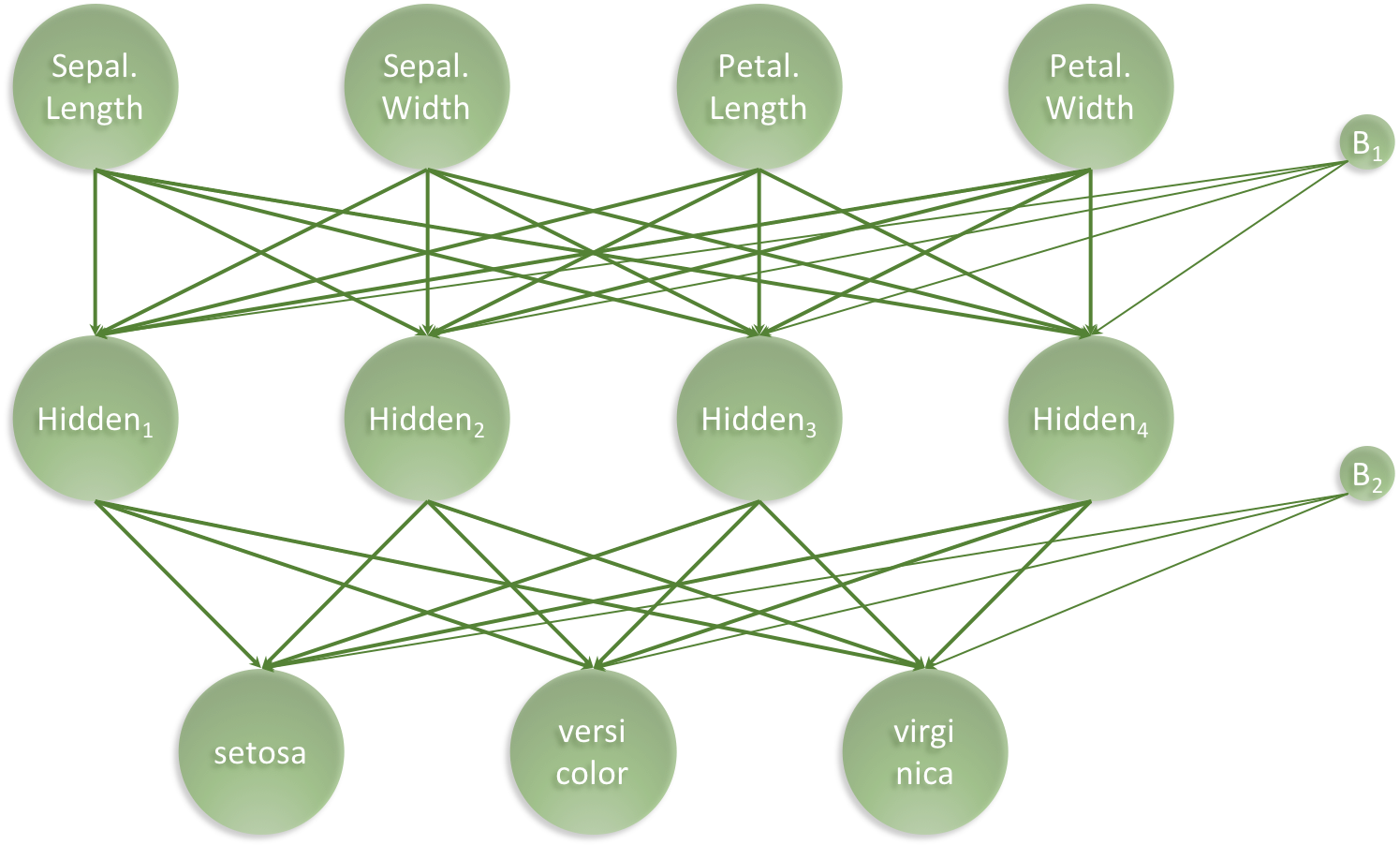
### Data

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 %>% 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.

### 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 %>% 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  
## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

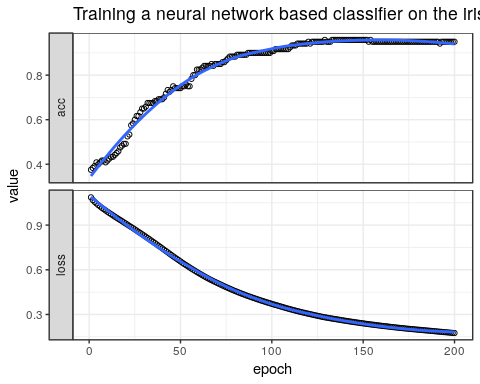
Next, the architecture set in the model needs to be compiled:

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()



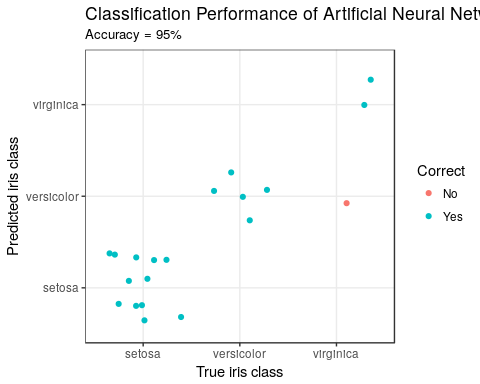
### Evaluate Network Performance

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")



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!!!**