Neural network

Philippe Jacquet (based on the DataCamp tutorial “keras: Deep Learning in R”)

June 7, 2018

INSTALLATION: Here are some instructions for installing Keras with TensorFlow at the backend. The R keras is actually an interface to the Python Keras. In simple terms, this means that the keras R package allows you to enjoy the benefit of R programming while having access to the capabilities of the Python Keras package.

Run R in your terminal or launch RStudio.

You may need to install the “devtools” package (for installing packages from github):

```
install.packages('devtools')
```

Install the “keras” package:

```
devtools::install_github("rstudio/keras")
```

Load the “keras” package:

```
library(keras)
install_keras()
```

EXERCICE: We are going to build a neural network to predict the species a given iris plant belongs to. Try to go through all the steps and to understand what each step is doing.

Load the "keras" package and the "iris" dataset:

```
library(keras)
use_session_with_seed(3)
data(iris)
```

We will first do some exploratory data analysis. Let us take a look at the dataset:

```
head(iris, 3)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Widt
## 1          5.1         3.5          1.4         0.
## 2          4.9         3.0          1.4         0.
## 3          4.7         3.2          1.3         0.
```

```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   M
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1
##  Median :5.800   Median :3.000   Median :4.350   M
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   M
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   M
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
```

```
##
```

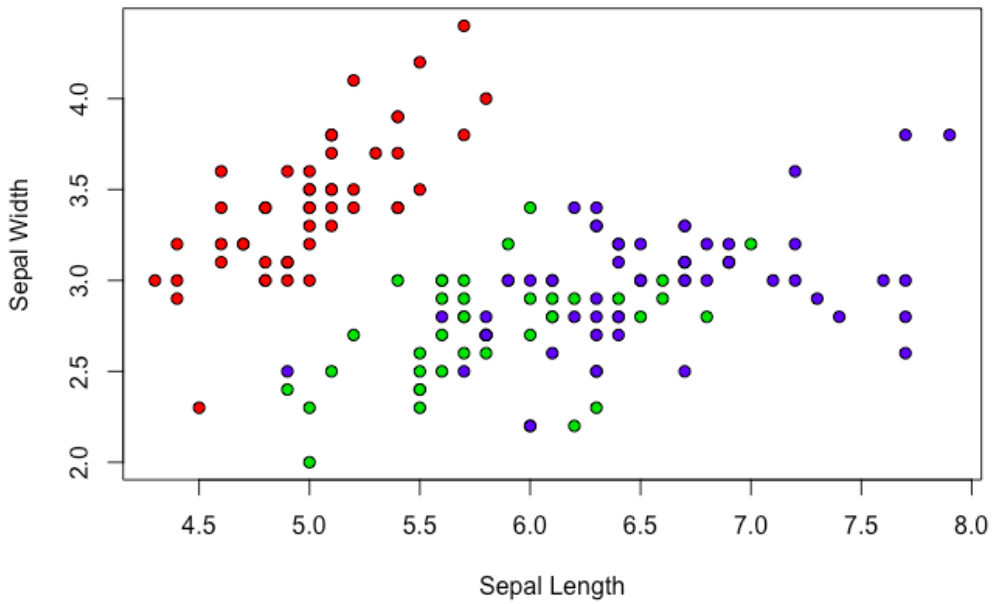
Plot a PCA:

```
#install.packages("ggplot2")
#install.packages("ggfortify")
library(ggfortify)
log.iris=log(iris[,1:4])
iris.pca=prcomp(log.iris,center=TRUE,scale.=TRUE)
autoplot(iris.pca,data=iris,colour='Species',main="PC
```

The PCA clearly shows three distinct groups corresponding to the three iris species. This gives us some hope that a neural network will be able to differentiate between the three specices based on the 4 measurements of the iris flowers. We also expect the setosa predictions to be very accurate, and maybe some confusion between versicolor and virginica.

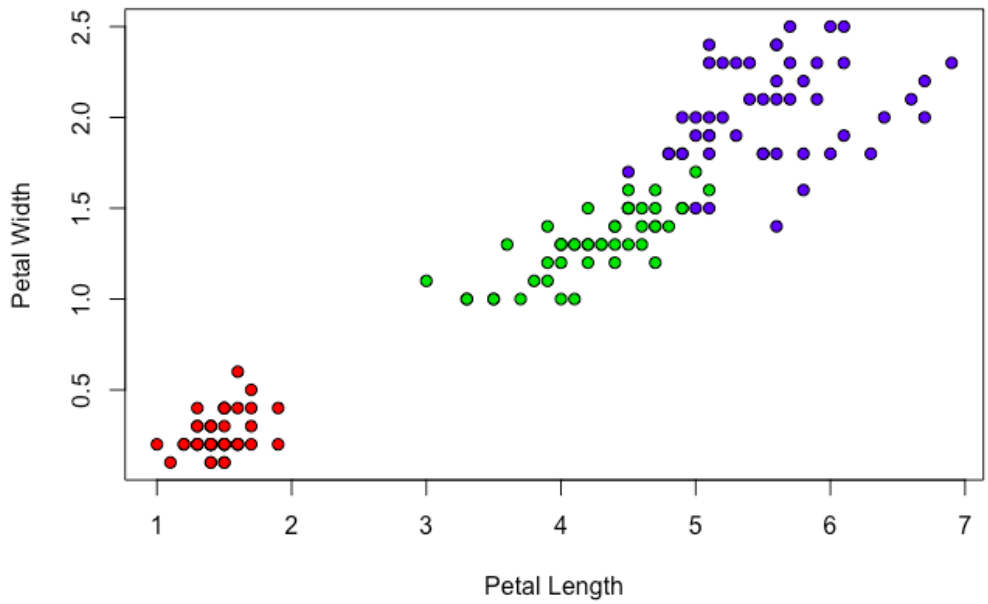
Plot Sepal.Width versus Sepal.Length:

```
plot(iris$Sepal.Length,iris$Sepal.Width,pch=21,bg=c("
```



Plot Petal.Width versus Petal.Length:

```
plot(iris$Petal.Length,iris$Petal.Width,pch=21,bg=c("
```



We see in the above plot that using only the Petal.Length and Petal.Width variables, one should be able to classify the plants by species. This observation will be confirmed later when using a decision tree.

Turn the “iris” dataset into a matrix:

```
iris[,5]=as.numeric(iris[,5])-1
iris=as.matrix(iris)
dimnames(iris)=NULL
```

Split the “iris” dataset into training and test datasets (the validation set will be included in the train set later on):

```
set.seed(2)
ind=sample(2,nrow(iris),replace=TRUE,prob=c(0.80,0.20

iris.training=iris[ind==1,1:4]
iris.test=iris[ind==2,1:4]
```

```
iris.trainingtarget=iris[ind==1,5]
iris.testtarget=iris[ind==2,5]

iris.trainLabels=to_categorical(iris.trainingtarget)
iris.testLabels=to_categorical(iris.testtarget)
```

Initialize a sequential model:

```
model=keras_model_sequential()
```

Add layers to the model with the “Glorot normal initializer” for the weights, a bias input node x0=1 and a bias hidden node z0=1:

```
model %>%
    layer_dense(input_shape=c(4),units=8,activation='
    layer_dense(units=3,activation='softmax',kernel_i
```

Print a summary of the model:

```
summary(model)
```

```
## _________________________________________________________________
## Layer (type)                 Output Shape
## =================================================================
## dense_1 (Dense)              (None, 8)
## _________________________________________________________________
## dense_2 (Dense)              (None, 3)
## =================================================================
## Total params: 67
## Trainable params: 67
## Non-trainable params: 0
## _________________________________________________________________
```

There are 5x8=40 links between the input and the hidden layer (4 initial nodes + 1 bias node x0=1, and 8 nodes in the hidden layer). There are 9x3=27 links between the hidden layer and the outer layer (9 nodes in the hidden layer including the bias z0=1, and 3 nodes at the output layer). Altogether there there 67 weight parameters to fit.

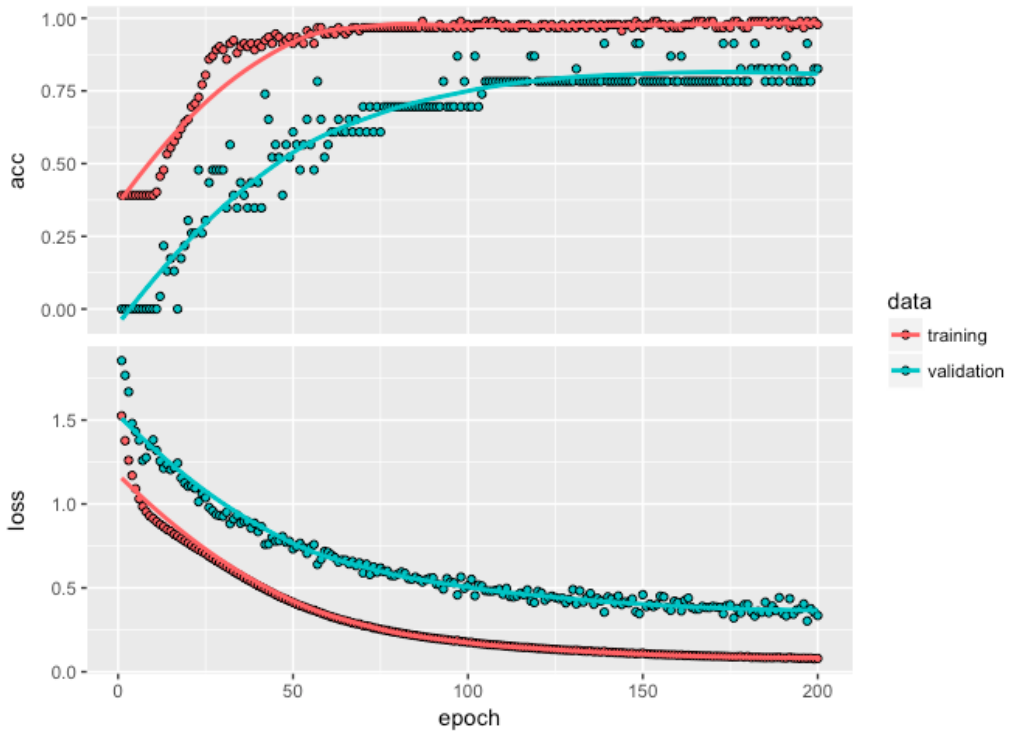
Question: Try to draw by hands the corresponding neural network.

Compile and fit the model:

```
model %>% compile(loss='categorical_crossentropy',opt
history=model %>% fit(iris.training,iris.trainLabels,
```

Visualize the model training history:

```
plot(history)
```



We see that at 200 epochs there is no sign of over-fitting, so we will take this value (if you try larger values of epochs, you will see no significant decrease in the loss function). Now that your model is created, compiled and has been fitted to the data, it is time to actually use your model to predict the labels for your test set "iris.test":

```
predicted.classes=model %>% predict_classes(iris.test
table(iris.testtarget,predicted.classes)
```

```
##                predicted.classes
## iris.testtarget  0  1  2
##               0 14  0  0
##               1  0  9  0
##               2  0  3  9
```

The above confusion matrix shows that the neural network has made only 3 mistakes in 35 predictions. As expected there is some confusion between versicolor and virginica. Let us now compute the loss and accuracy values:

```
score=model %>% evaluate(iris.test,iris.testLabels)
print(score)
```

```
## $loss
## [1] 0.1490622
##
## $acc
## [1] 0.9142857
```

Questions: (0) Play around with different hyperparameter values [activation ('sigmoid' or 'tanh'), kernel_initializer ("glorot_uniform"), use_bias (FALSE), epochs, batch size, optimizer ("sgd")] in the hope that your model will perform better. We may want to look at https://keras.rstudio.com/. (1)

Try adding more hidden units to the model. (2) Try adding another layer to your model.

Solution 1:

```
model=keras_model_sequential()

model %>%
    layer_dense(input_shape=c(4),units=28,activation=
    layer_dense(units=3,activation='softmax',kernel_i

model %>% compile(loss='categorical_crossentropy',opt

model %>% fit(iris.training,iris.trainLabels,epochs=2

score=model %>% evaluate(iris.test,iris.testLabels)

print(score)
```

```
## $loss
## [1] 0.1408264
##
## $acc
## [1] 0.9428571
```

Solution 2:

```
model=keras_model_sequential()

model %>%
    layer_dense(input_shape=c(4),units=28,activation=
    layer_dense(units=5,activation='relu',kernel_init
    layer_dense(units=3,activation='softmax',kernel_i

model %>% compile(loss='categorical_crossentropy',opt

model %>% fit(iris.training,iris.trainLabels,epochs=2
```

```
score=model %>% evaluate(iris.test,iris.testLabels)

print(score)
```

```
## $loss
## [1] 0.1918294
##
## $acc
## [1] 0.9428571
```