Stat 8330 Neural Network R Lab

Chris Wikle

October 2020

In this lab we will experiment with a regression and classification problem with deep feed forward neural networks. There are several pacakges in r that can do this. I like the `ANN2` package because I like the options one has to choose from, and it is easy. It is not as flexible as keras or some of the other interfaces with tensorflow or pyTorch, but for feed forward models, it is definitely quite usable.

The Boston Housing Dataset We illustrate deep FFN regression with the well-used Boston Housing dataset (which we have used in the book labs). This is a very small dataset, but lets you experiment very quickly. The classifcation dataset below is VERY large and takes a while to fit.

```
library(ANN2)
bhouse <- read.table("housingdata")
X = bhouse[,1:13]
Y = bhouse[,14]

# make training and test samples 85% training, 15%  test
set.seed(1)
trainInds <- sample(1:nrow(X), round(nrow(X)*0.85))
valInds <- (1:nrow(X))[-trainInds]

Xtrain <- X[trainInds, ]
Ytrain <- Y[trainInds]

Xtest <- X[valInds, ]
Ytest <- Y[valInds]

# find mean and sd column-wise of training data (not surprisingily called
# a "z-transform" in the machine learning world)
XtrainMean <- apply(Xtrain,2,mean)
XtrainSd <- apply(Xtrain,2,sd)
Xtrain_stand <- sweep(sweep(Xtrain, 2L, XtrainMean), 2, XtrainSd, "/")
# check means and standard deviations
colMeans(Xtrain_stand)
```

```
##            V1            V2            V3            V4            V5
## -1.031959e-17 -3.472674e-17  9.688632e-17  2.969201e-18  2.079409e-16
##            V6            V7            V8            V9           V10
##  1.544388e-16 -4.649866e-17 -6.877573e-17  8.520316e-18 -3.244336e-17
##           V11           V12           V13
## -6.050070e-16 -1.276807e-16 -6.368856e-17
```

```
apply(Xtrain_stand, 2, sd)
```

```
##  V1  V2  V3  V4  V5  V6  V7  V8  V9 V10 V11 V12 V13
##   1   1   1   1   1   1   1   1   1   1   1   1   1
```

```r
# now, standardize the test data using the training means/sd; as
# expected, these don't have exactly mean 0 and sd 1
Xtest_stand <- sweep(sweep(Xtest, 2L, XtrainMean), 2, XtrainSd, "/")
colMeans(Xtest_stand)
```

```
##          V1          V2          V3          V4          V5          V6
##  0.05268874  0.14786171 -0.19932574 -0.07515721  0.01936599  0.07166221
##          V7          V8          V9         V10         V11         V12
## -0.04590013  0.11835796 -0.10723676 -0.16801418 -0.11114238  0.18457251
##         V13
## -0.08359955
```

```r
apply(Xtest_stand, 2, sd)
```

```
##        V1        V2        V3        V4        V5        V6        V7        V8
## 1.4210460 1.1675425 0.9339871 0.8680693 1.0411859 0.8895409 0.9777374 1.0807967
##        V9       V10       V11       V12       V13
## 0.9508794 0.9525441 1.0941657 0.7611498 0.9495329
```

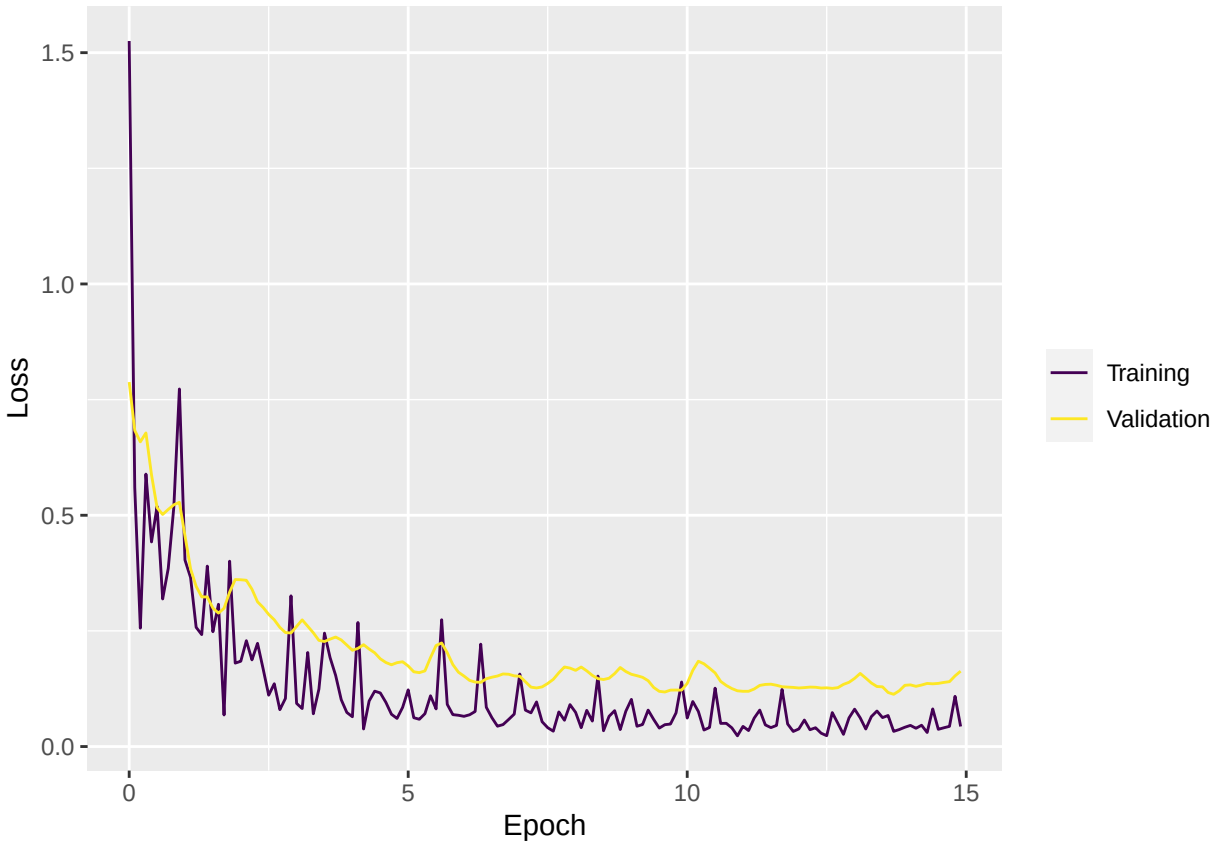
```r
# standardize the Y's
Ytrain_stand <- (Ytrain - mean(Ytrain))/sd(Ytrain)
Ytest_stand <- (Ytest - mean(Ytrain))/sd(Ytrain)

# Now, train the deep NN. Make sure you look at the help for the ``neuralnetwork''
# function to see the various options.  For regression, there are several loss
# functions you can choose, but the standard is squared error (of course). I just
# choose an architecture here (number of hidden layers, units, optimizer,
# activation function, etc.). You will definitely want to play with these. In
# this application, it is VERY fast, so easy to do.

NNbhouse <- neuralnetwork(X = Xtrain_stand, y = Ytrain_stand,regression = TRUE,
                    standardize = FALSE, loss.type = "squared",
                    hidden.layers =c(64,32,16),
                    optim.type = 'adam',n.epochs = 15,activ.functions = "relu",
                    learn.rates = 0.01, val.prop = 0.2)
```

```
## Artificial Neural Network:
##   Layer - 13 nodes - input
##   Layer - 64 nodes - relu
##   Layer - 32 nodes - relu
##   Layer - 16 nodes - relu
##   Layer - 1 nodes - linear
## With squared loss and Adam optimizer
## Training progress:
## [|--------------------------------------------------] 0% - Validation loss: 0.787943[|----------------
```

```r
# Plot the loss during training (also, the validation sample if val.prop ne 0)
plot(NNbhouse)
```




```
# summarize the NN
print(NNbhouse)
```

```
## Artificial Neural Network:
##   Layer - 13 nodes - input
##   Layer - 64 nodes - relu
##   Layer - 32 nodes - relu
##   Layer - 16 nodes - relu
##   Layer - 1 nodes - linear
## With squared loss and Adam optimizer
## Trained for 15 epochs
```

```
# Make predictions
y_pred <- predict(NNbhouse, newdata = Xtest_stand)

# MSE
MSE <- mean((Ytest_stand - y_pred$predictions)^2)
MSE
```

```
## [1] 0.2205127
```

```
#how does this MSE compare to multiple regression, or lasso, or ridge, etc.?
```

The Buzz Dataset (Source available at: https://www.kaggle.com/potamitis/wingbeats)

These data represent wingbeat recordings from 6 species of mosquito collected using an optical device. The inspiration was to create smart mosquito traps that can be used to monitor the abundance of different mosquito species.

The raw dataset consists of .wav files of male and female mosquitoes representing 3 genera and 6 species:

Aedes aegypti (0), *Aedes albopictus* (1), *Anophales arabiensis* (2), *Anophales gambiae*(3), *Culex pipiens* (4), *Culex quinquefasciatus* (5). Raw audio files (<1s long) have been converted into smoothed periodograms (a spectral representation of the audio time series). This is what makes up the x data given below. We seek to classify the mosquito species given these audio periodograms. Note, we have 12,000 periodograms (samples) from the individual audio files, and each has 2,500 values (features) that describe the smoothed periodogram. We will use 90% of the data for training and 10% for validation.

In this application, we will use the `ANN2` package for classification. Note, this is a much larger data set than we typically work with in this class, so be prepared for the training to take quite a long time.

```
library(ANN2)

load("~/Box Sync/Courses/Stat8085deeplearning/Example1_buzz/BuzzRDataFiles/buzz_X1.RData")
load("~/Box Sync/Courses/Stat8085deeplearning/Example1_buzz/BuzzRDataFiles/buzz_X2.RData")
load("~/Box Sync/Courses/Stat8085deeplearning/Example1_buzz/BuzzRDataFiles/buzz_X3.RData")
load("~/Box Sync/Courses/Stat8085deeplearning/Example1_buzz/BuzzRDataFiles/buzz_X4.RData")
load("~/Box Sync/Courses/Stat8085deeplearning/Example1_buzz/BuzzRDataFiles/buzz_X5.RData")
load("~/Box Sync/Courses/Stat8085deeplearning/Example1_buzz/BuzzRDataFiles/buzz_X6.RData")
load("~/Box Sync/Courses/Stat8085deeplearning/Example1_buzz/BuzzRDataFiles/buzz_Y.RData")

X <- rbind(X1, X2, X3, X4, X5, X6)
str(X)
```

```
##  num [1:12000, 1:2500] 15448907 4980965 7850658 5318554 1888559 ...
```

```
str(Y)
```

```
##  num [1:12000] 0 0 0 0 0 0 0 0 0 0 ...
```

```
unique(Y)
```

```
## [1] 0 1 2 3 4 5
```

```
# select training (90%) and validation samples (10%) randomly
set.seed(1)
trainInds <- sample(1:nrow(X), round(nrow(X)*0.9))
valInds <- (1:nrow(X))[-trainInds]

Xtrain <- X[trainInds, ]
Ytrain_classes <- Y[trainInds]

Xtest <- X[valInds, ]
Ytest_classes <- Y[valInds]

# rescale function to scale training X's between 0-1 (alternatively, you
#  could standardize as we usually do by dividing by the standard deviation);
# Here is a little function to scale between 0-1; note, we have to apply the same #transformation that

rescaleCols <- function(rowX, colMins, colMaxs)
{
  r <- (rowX - colMins)/(colMaxs - colMins)
  r[is.nan(r)] <- 0
  return(r)
}

colMinsX <- apply(Xtrain, 2, min)
colMaxsX <- apply(Xtrain, 2, max)
```

```r
Xtrain_scaled <- t(apply(Xtrain, 1, rescaleCols, colMinsX, colMaxsX))
Xtest_scaled <- t(apply(Xtest, 1, rescaleCols, colMinsX, colMaxsX))

# it is good to look to see how the scaling worked - especially on
# the test set
range(Xtrain_scaled)
```

```
## [1] 0 1
```

```r
range(Xtest_scaled)
```

```
## [1] -0.0008748488  2.0461963448
```

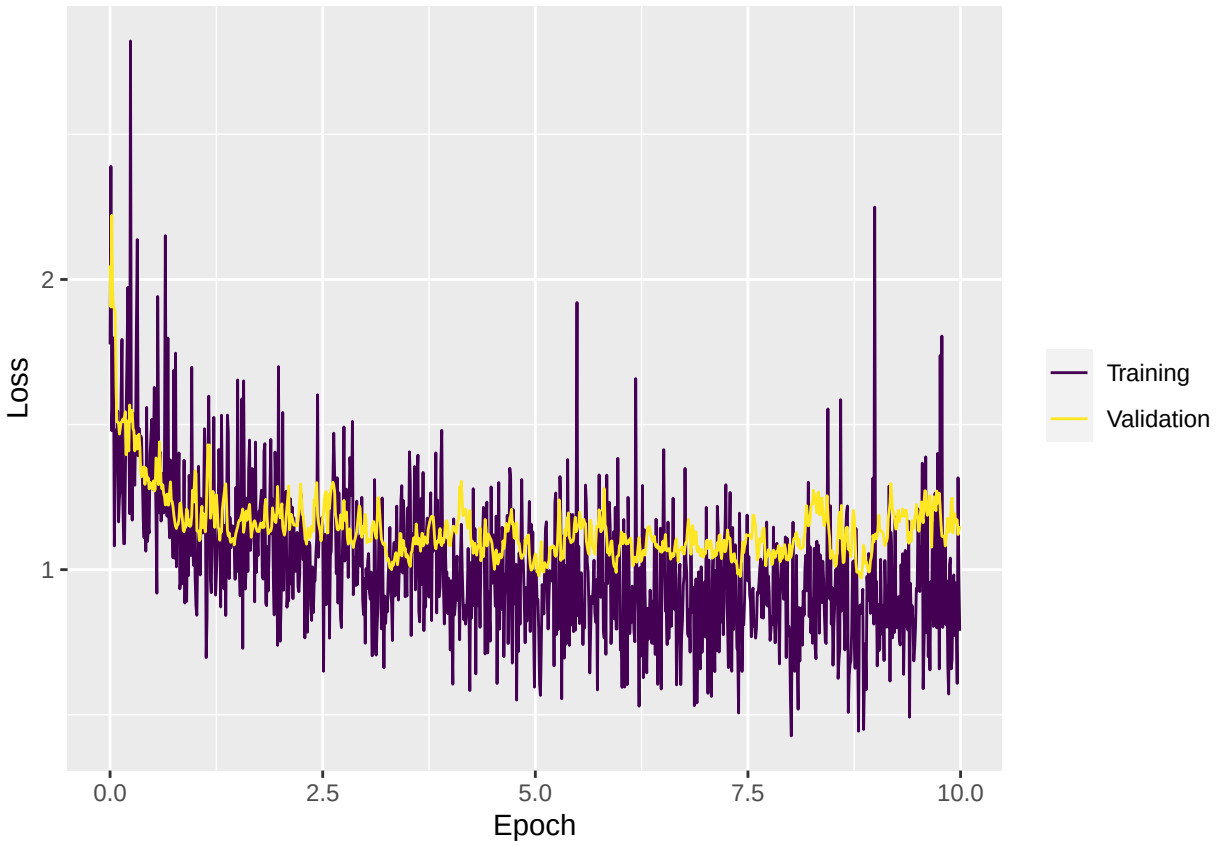
```r
# convert the responses to factor variables
Ytrain <- as.factor(Ytrain_classes)
Ytest <- as.factor(Ytest_classes)


# Now, train the deep NN. Make sure you look at the help for the ``neuralnetwork''
# function to see the various options.  For classification, it uses the ``log''
# loss, which is cross-entropy. Note, in this problem, it can take a LONG time
# to train if you have a lot of hidden layers or hidden units.  Also, letting the
# proportion of training data used for tracing the loss on the validation set,
# val.prop be > 0 adds quite a bit more time, but it is useful to do to see
# the convergence better (in my opinion). But, if you want to play with options,
# probably best to set it to 0.

NN <- neuralnetwork(X = Xtrain_scaled, y = Ytrain, hidden.layers = c(64,32,32),
                    optim.type = 'adam',n.epochs = 10,activ.functions = "relu",
                    learn.rates = 0.01, val.prop = 0.05)
```

```
## Artificial Neural Network:
##   Layer - 2500 nodes - input
##   Layer - 64 nodes - relu
##   Layer - 32 nodes - relu
##   Layer - 32 nodes - relu
##   Layer - 6 nodes - softmax
## With log loss and Adam optimizer
## Training progress:
## [|--------------------------------------------------] 0% - Validation loss: 2.04816[|----------------
```

```r
# Plot the loss during training (also, the validation sample if val.prop ne 0)
plot(NN)
```




```
# Make predictions
y_pred <- predict(NN, newdata = Xtest_scaled)

# Confusion matrix
tabl_pred <- table(truth = Ytest, fitted = y_pred$predictions)
tabl_pred
```

```
##       fitted
## truth   0   1   2   3   4   5
##     0 132   5   9   1  28  16
##     1  31  95  26  19  12   3
##     2  16   3  91  37  34  20
##     3  13   3  47 105  17   0
##     4  39   2   6   2 180   3
##     5  27   2  10   0   4 162
```

```
# Accuracy
sum(diag(tabl_pred))/length(Ytest)
```

```
## [1] 0.6375
```

I highly recommend that you play around with this. Check the sensitivity to the optimizer, the activation functions, number of hidden levels/units, learning rate, etc.