10 Deep Learning

By: Udit (based on ISLR)

Fitting Linear & Lasso Models

The **pipe operator %>%** passes the previous term keras model sequential pipe as the first argument to the next function, and returns the result.

```
library(ISLR2)
library(ggplot2)
library(magrittr) # for pipe operator
library(keras)

names(Hitters)
```

```
##  [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"       "Walks"
##  [7] "Years"     "CAtBat"    "CHits"     "CHmRun"    "CRuns"     "CRBI"
## [13] "CWalks"    "League"    "Division"  "PutOuts"   "Assists"   "Errors"
## [19] "Salary"    "NewLeague"
```

```
summary(Hitters)
```

```
##      AtBat            Hits         HmRun            Runs
##  Min.   : 16.0   Min.   :  1   Min.   : 0.00   Min.   :  0.00
##  1st Qu.:255.2   1st Qu.: 64   1st Qu.: 4.00   1st Qu.: 30.25
##  Median :379.5   Median : 96   Median : 8.00   Median : 48.00
##  Mean   :380.9   Mean   :101   Mean   :10.77   Mean   : 50.91
##  3rd Qu.:512.0   3rd Qu.:137   3rd Qu.:16.00   3rd Qu.: 69.00
##  Max.   :687.0   Max.   :238   Max.   :40.00   Max.   :130.00
##
##       RBI             Walks            Years            CAtBat
##  Min.   :  0.00   Min.   :  0.00   Min.   : 1.000   Min.   :   19.0
##  1st Qu.: 28.00   1st Qu.: 22.00   1st Qu.: 4.000   1st Qu.:  816.8
##  Median : 44.00   Median : 35.00   Median : 6.000   Median : 1928.0
##  Mean   : 48.03   Mean   : 38.74   Mean   : 7.444   Mean   : 2648.7
##  3rd Qu.: 64.75   3rd Qu.: 53.00   3rd Qu.:11.000   3rd Qu.: 3924.2
##  Max.   :121.00   Max.   :105.00   Max.   :24.000   Max.   :14053.0
##
##      CHits            CHmRun           CRuns             CRBI
##  Min.   :   4.0   Min.   :  0.00   Min.   :   1.0   Min.   :   0.00
##  1st Qu.: 209.0   1st Qu.: 14.00   1st Qu.: 100.2   1st Qu.:  88.75
##  Median : 508.0   Median : 37.50   Median : 247.0   Median : 220.50
##  Mean   : 717.6   Mean   : 69.49   Mean   : 358.8   Mean   : 330.12
##  3rd Qu.:1059.2   3rd Qu.: 90.00   3rd Qu.: 526.2   3rd Qu.: 426.25
##  Max.   :4256.0   Max.   :548.00   Max.   :2165.0   Max.   :1659.00
##
##      CWalks          League  Division    PutOuts          Assists
##  Min.   :   0.00   A:175   E:157    Min.   :   0.0   Min.   :  0.0
##  1st Qu.:  67.25   N:147   W:165    1st Qu.: 109.2   1st Qu.:  7.0
##  Median : 170.50                    Median : 212.0   Median : 39.5
##  Mean   : 260.24                    Mean   : 288.9   Mean   :106.9
```

```
##  3rd Qu.: 339.25                    3rd Qu.: 325.0   3rd Qu.:166.0
##  Max.   :1566.00                    Max.   :1378.0   Max.   :492.0
##
##      Errors          Salary       NewLeague
##  Min.   : 0.00   Min.   :  67.5   A:176
##  1st Qu.: 3.00   1st Qu.: 190.0   N:146
##  Median : 6.00   Median : 425.0
##  Mean   : 8.04   Mean   : 535.9
##  3rd Qu.:11.00   3rd Qu.: 750.0
##  Max.   :32.00   Max.   :2460.0
##                  NA's   :59
```

```
hit.data = na.omit(Hitters)

# Split data into test and train
n = nrow(hit.data)
set.seed(13)
test = sample(1:n, n/3)

# Fitting linear model
ln.fit = lm(Salary~., data=hit.data[-test,])
ln.pred = predict(ln.fit, hit.data[test,])
sqrt(mean((ln.pred - hit.data$Salary[test])^2))  # RMSE = 341
```

```
## [1] 341.0237
```

```
# Fitting lasso using glmnet - need to create model matrix
library(glmnet)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.1-3
```

```
x = model.matrix(Salary~. -1 , data=hit.data) %>% scale()
y = hit.data$Salary

cvfit = cv.glmnet(x[-test,], y[-test], type.measure = "mae")
cvpred = predict(cvfit, x[test,], s = "lambda.min")
sqrt(mean((cvpred - hit.data$Salary[test])^2))  # RMSE = 359
```

```
## [1] 359.2246
```

Fitting Neural Network

The object modnn has a single hidden layer with 50 hidden units, and a ReLU activation function. It then has a dropout layer, in which a random 40% of the 50 activations from the previous layer are set to zero during each iteration of the stochastic gradient descent algorithm. Finally, the output layer has just one unit with no activation function, indicating that the model provides a single quantitative output.

units - dimensionality of output space.
input_shape - Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

```
# Creating a network and adding details - o/p is single quantitative output
modnn = keras_model_sequential() %>%
  layer_dense(units=50, activation="relu", input_shape = ncol(x)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 1)
```

```
## Loaded Tensorflow version 2.7.0
```

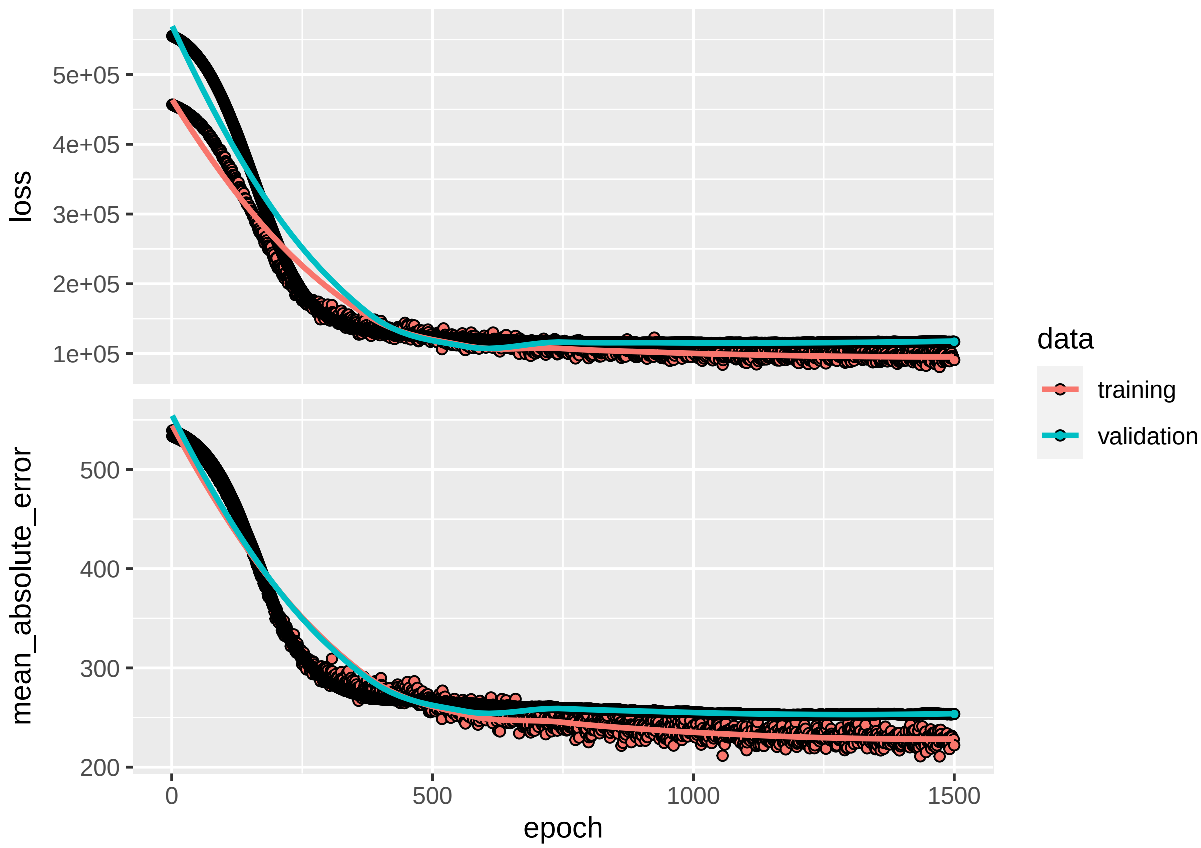
```r
summary(modnn)
```

```
## Model: "sequential"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  dense_1 (Dense)                    (None, 50)                      1050
##
##  dropout (Dropout)                  (None, 50)                      0
##
##  dense (Dense)                      (None, 1)                       51
##
## ================================================================================
## Total params: 1,101
## Trainable params: 1,101
## Non-trainable params: 0
## ________________________________________________________________________________
```

```r
# Add details on fitting algorithm (compile passes the info to python instance)
modnn %>% compile(loss="mse", optimizer = optimizer_rmsprop(),
                  metrics = list("mean_absolute_error"))

# Fit the model - 2 parameters (epochs and batch_size)
history = modnn %>% fit(x[-test,], y[-test], epochs=1500, batch_size=32,
                        validation_data = list(x[test,], y[test]))
plot(history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



```
npred = predict(modnn, x[test,])
sqrt(mean((npred - hit.data$Salary[test])^2))  # RMSE = 339
```

```
## [1] 342.1589
```

Fitting Neural Network - MNIST Digit Data

There are 60,000 images in the training data and 10,000 in the test data. The images are 28x28, and stored as a 3D array.

Neural networks are somewhat sensitive to the scale of the inputs. Here the inputs are eight-bit grayscale values between 0 and 255, so we scale to the unit interval.

```
mnist = dataset_mnist()
x_train = mnist$train$x
y_train = mnist$train$y
dim(x_train)
```

```
## [1] 60000    28    28
```

```
x_test = mnist$test$x
y_test = mnist$test$y
dim(x_test)
```

```
## [1] 10000    28    28
```

```
get_matrix = function(x){
  array_reshape(x, c(nrow(x), 28*28))
}

x_train = get_matrix(x_train)
x_test  = get_matrix(x_test)

# example
to_categorical(head(y_train), 10)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    0    0    0    0    1    0    0    0     0
## [2,]    1    0    0    0    0    0    0    0    0     0
## [3,]    0    0    0    0    1    0    0    0    0     0
## [4,]    0    1    0    0    0    0    0    0    0     0
## [5,]    0    0    0    0    0    0    0    0    0     1
## [6,]    0    0    1    0    0    0    0    0    0     0
```

```
y_train = to_categorical(y_train, 10) #convert to categorical w/ 10 classes
y_test  = to_categorical(y_test, 10)


# scaling - Neural networks are sensitive to scale
x_train = x_train/255
x_test  = x_test/255

# Create a Neural Network model
modelnn <- keras_model_sequential()
modelnn %>%
  layer_dense(units = 256, activation = 'relu', input_shape = c(784)) %>%
```

```r
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 10, activation = 'softmax')
summary(modelnn)
```

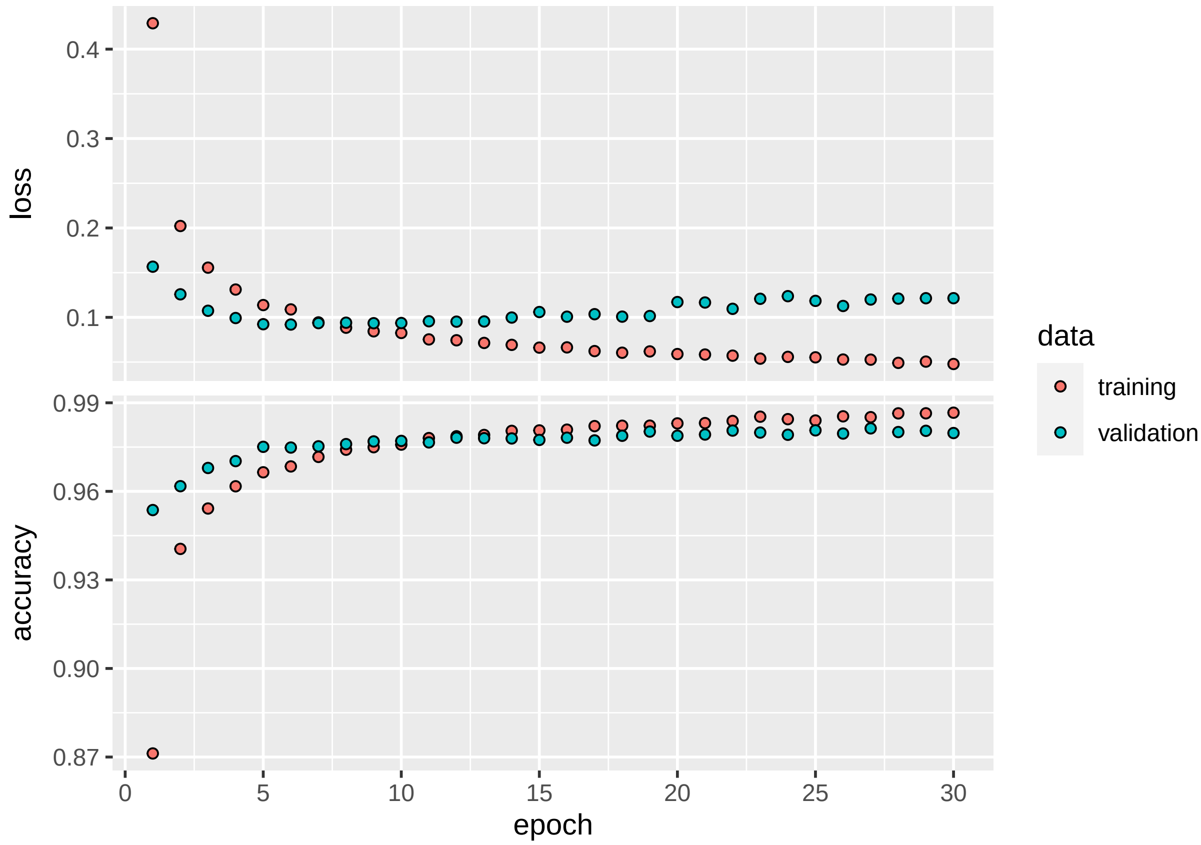
```
## Model: "sequential_1"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                      Param #
## ================================================================================
##  dense_4 (Dense)                    (None, 256)                       200960
##
##  dropout_2 (Dropout)                (None, 256)                       0
##
##  dense_3 (Dense)                    (None, 128)                       32896
##
##  dropout_1 (Dropout)                (None, 128)                       0
##
##  dense_2 (Dense)                    (None, 10)                        1290
##
## ================================================================================
## Total params: 235,146
## Trainable params: 235,146
## Non-trainable params: 0
## ________________________________________________________________________________
```

```r
# Details for fitting
modelnn %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)

# Fit the model
system.time(
history <- modelnn %>% fit(x_train, y_train, epochs=30, batch_size=128,
                           validation_split=0.2))
```

```
##    user  system elapsed
##  175.38    5.16   36.83
```

```r
plot(history, smooth=FALSE)
```




```
# Calculate accuracy
accu = function(pred, truth) {mean(drop(pred)== drop(truth))}

ypred = modelnn %>% predict(x_test) %>% k_argmax()
accu(ypred$numpy(), mnist$test$y)  #98%
```

```
## [1] 0.9814
```

```
# Fitting a single layer model
modellr = keras_model_sequential() %>% layer_dense(input_shape=784, units=10,
                                                   activation="softmax")
summary(modellr)
```

```
## Model: "sequential_2"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                     Param #
## ================================================================================
##  dense_5 (Dense)                    (None, 10)                       7850
##
## ================================================================================
## Total params: 7,850
## Trainable params: 7,850
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
modellr %>% compile(loss="categorical_crossentropy",
                    optimizer=optimizer_rmsprop(),
                    metrics=c("accuracy"))
modellr %>% fit(x_train, y_train, epochs=30, batch_size=128, validation_split=0.2)
```

```
ypred = modellr %>% predict(x_test) %>% k_argmax()
accu(ypred$numpy(), mnist$test$y)  #90%
```

```
## [1] 0.9273
```

CNN - Convolutional Neural Network

The array of 50,000 training images has 4 dimensions: each color image is represented as a set of 3 channels, each of which consists of 32x32 8bit pixels.

```
#cifar = dataset_cifar100()
setwd("C:/Users/uditg/Documents/R scripts")
cifar = readRDS("cifar100_object")
names(cifar)
```

```
## [1] "train" "test"
```

```
x_train <- cifar$train$x
g_train <- cifar$train$y
x_test  <- cifar$test$x
g_test  <- cifar$test$y
dim(x_train)
```

```
## [1] 50000    32    32     3
```

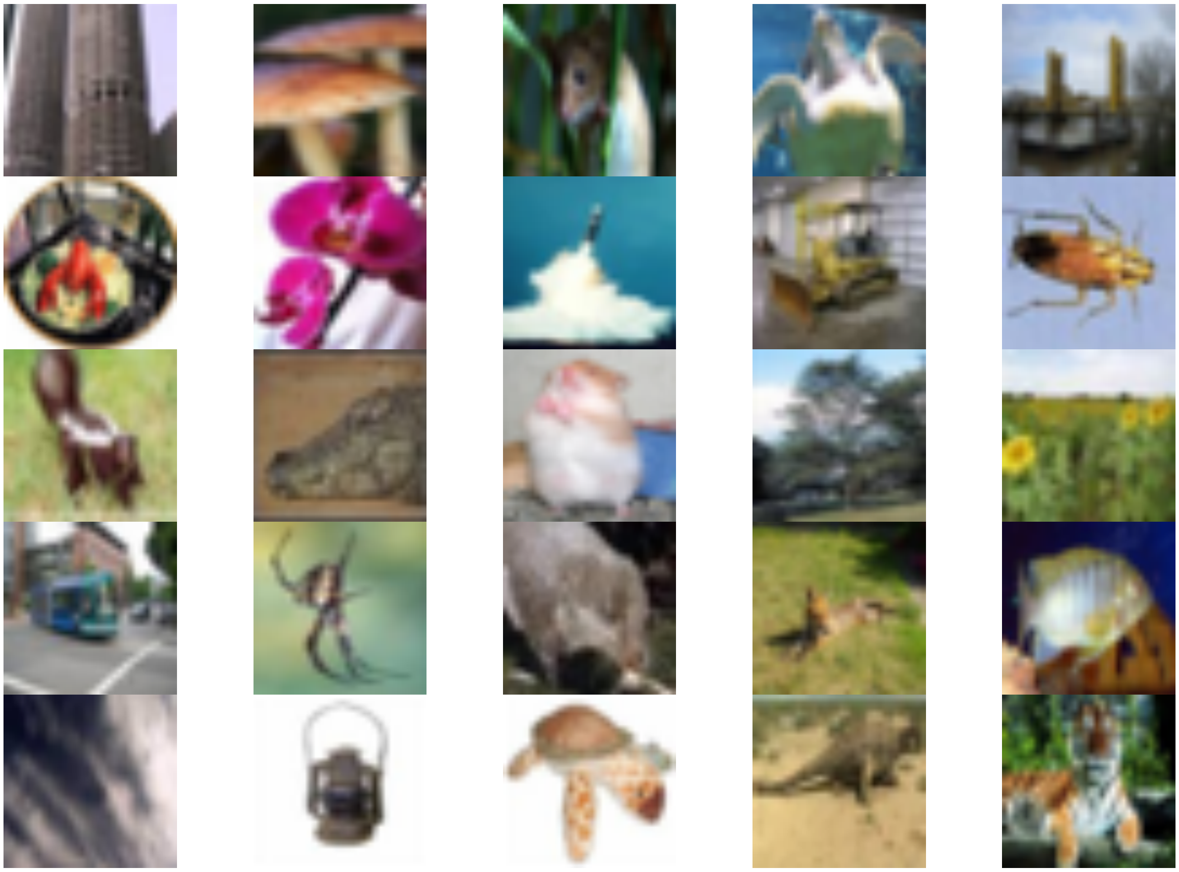
```
range(x_train[1,,,1])
```

```
## [1]  13 255
```

```
x_train = x_train/255
x_test  = x_test/255
y_train = to_categorical(g_train,100)
dim(y_train)
```

```
## [1] 50000   100
```

```
# plotting sample images
library(jpeg)
par(mar=c(0,0,0,0), mfrow=c(5,5))
index = sample(seq(50000),25)
for (i in index) plot(as.raster(x_train[i,,,]))
```



```r
# Moderately sized CNN model
modelcnn = keras_model_sequential() %>%
  layer_conv_2d(filters=32, kernel_size=c(3,3),
                padding = "same", activation="relu",
                input_shape = c(32, 32, 3)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters=64, kernel_size=c(3,3),
                padding="same", activation="relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters=128, kernel_size=c(3,3),
                padding="same", activation="relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters=256, kernel_size=c(3,3),
                padding="same", activation="relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_flatten() %>%
  layer_dropout(rate=0.5) %>%
  layer_dense(units=512, activation="relu") %>%
  layer_dense(units=100, activation="softmax")
summary(modelcnn)
```

```
## Model: "sequential_3"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                     Param #
## ================================================================================
##  conv2d_3 (Conv2D)                  (None, 32, 32, 32)               896
##
##  max_pooling2d_3 (MaxPooling2D)     (None, 16, 16, 32)               0
##
##  conv2d_2 (Conv2D)                  (None, 16, 16, 64)               18496
```

```
##
##  max_pooling2d_2 (MaxPooling2D)     (None, 8, 8, 64)              0
##
##  conv2d_1 (Conv2D)                  (None, 8, 8, 128)             73856
##
##  max_pooling2d_1 (MaxPooling2D)     (None, 4, 4, 128)             0
##
##  conv2d (Conv2D)                    (None, 4, 4, 256)             295168
##
##  max_pooling2d (MaxPooling2D)       (None, 2, 2, 256)             0
##
##  flatten (Flatten)                  (None, 1024)                  0
##
##  dropout_3 (Dropout)                (None, 1024)                  0
##
##  dense_7 (Dense)                    (None, 512)                   524800
##
##  dense_6 (Dense)                    (None, 100)                   51300
##
## ================================================================================
## Total params: 964,516
## Trainable params: 964,516
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
modelcnn %>% compile(loss="categorical_crossentropy", optimizer=optimizer_rmsprop(),
                  metrics=c("accuracy"))
history = modelcnn %>% fit(x_train, y_train, epochs=30,
                        batch_size=128, validation_split=0.2)
ypred = modelcnn%>% predict(x_test) %>% k_argmax()
accu(ypred$numpy(), g_test)  #45%
```

```
## [1] 0.4353
```

CNN Pretrained Models - ImageNet

```
img_location = "CNN_images"
image_names = list.files(img_location)
num_images = length(image_names)
x = array(dim=c(num_images, 224, 224, 3))
for(i in 1:num_images){
  img_path = paste(img_location, image_names[i], sep="/")
  img = image_load(img_path, target_size=c(224,224))
  x[i,,,] = image_to_array(img)
}
x = imagenet_preprocess_input(x)

modelpre = application_resnet50(weights="imagenet")
#summary(modelpre)
pred6 = modelpre %>% predict(x) %>% imagenet_decode_predictions(top=3)
names(pred6) = image_names
print(pred6)
```

```
## $car_internet.jfif
##   class_name class_description      score
## 1  n03770679           minivan 0.47357273
```

```
## 2  n03796401        moving_van 0.11269771
## 3  n03594945              jeep 0.06929874
##
## $cat_internet.jfif
##   class_name class_description      score
## 1  n02124075      Egyptian_cat 0.53495884
## 2  n02123045             tabby 0.11732875
## 3  n02123159         tiger_cat 0.08018519
##
## $elephant_internet.jfif
##   class_name class_description      score
## 1  n02504458  African_elephant 0.53504777
## 2  n02437312     Arabian_camel 0.40940914
## 3  n01871265            tusker 0.03513662
##
## $flamingo.jpg
##   class_name class_description       score
## 1  n02007558          flamingo 0.926349938
## 2  n02006656         spoonbill 0.071699433
## 3  n02002556       white_stork 0.001228211
##
## $guitar_internet.jfif
##   class_name class_description       score
## 1  n02676566   acoustic_guitar 0.858463466
## 2  n03272010   electric_guitar 0.138924599
## 3  n02787622             banjo 0.002450667
##
## $hawk.jpg
##   class_name class_description     score
## 1  n03388043          fountain 0.2788653
## 2  n03532672              hook 0.1785543
## 3  n03804744              nail 0.1080727
##
## $hawk_cropped.jpeg
##   class_name class_description      score
## 1  n01608432              kite 0.72270924
## 2  n01622779    great_grey_owl 0.08182573
## 3  n01532829       house_finch 0.04218878
##
## $huey.jpg
##   class_name            class_description      score
## 1  n02097474              Tibetan_terrier 0.50929672
## 2  n02098413                        Lhasa 0.42209941
## 3  n02098105 soft-coated_wheaten_terrier 0.01695856
##
## $kitty.jpg
##   class_name    class_description      score
## 1  n02105641 Old_English_sheepdog 0.83265990
## 2  n02086240             Shih-Tzu 0.04513895
## 3  n03223299              doormat 0.03299776
##
## $mtn_internet.jfif
##   class_name class_description       score
## 1  n09193705               alp 0.961790085
## 2  n09468604            valley 0.029649865
## 3  n03792972     mountain_tent 0.003976547
##
## $weaver.jpg
##   class_name class_description      score
```

```
## 1  n01843065           jacamar 0.49795389
## 2  n01818515             macaw 0.22193316
## 3  n02494079   squirrel_monkey 0.04287858
```

IMDb Document Classification

Using: 1. Logistic regression w/ Lasso regularization 2. Bag-of-words model 3. **RNN (handles vector embedding too)**

Logistic regression w/ Lasso regularization We score each document for the presence or absence of each of the words in a language dictionary - in this case an English dictionary. If the dictionary contains M words, that means for each document we create a binary feature vector of length M, and score a 1 for every word present, and 0 otherwise.

```
max_features = 10000
imdb = dataset_imdb(num_words=max_features)
max(unlist(x_train))
```

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

```
c(c(x_train, y_train), c(x_test, y_test)) %<-% imdb

x_train[[1]][1:12]
```

```
##  [1]    1   14   22   16   43  530  973 1622 1385   65  458 4468
```

```
word_index <- dataset_imdb_word_index()

# Text codes are off by 3 because of adjustments, made explicitly below
word = names(word_index)  #words in the dictionary
idx  = unlist(word_index, use.names=FALSE) #paired values
word = c("<PAD>","<START>","<UNK>","<UNUSED>",word) #appending values
idx  = c(0:3, idx+3)

decode_review = function(text){
  words= word[match(text, idx, 2)] #returns 2 when no match
  paste(words, collapse = " ")
}

decode_review(x_train[[1]][1:12])
```

```
## [1] "<START> this film was just brilliant casting location scenery story direction everyone's"
```

```
y_train[1]  # 1 = Good
```

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

```
decode_review(x_train[[3]][1:12])
```

```
## [1] "<START> this has to be one of the worst films of the"
```

```
y_train[3]  # 0 = Bad
```

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

```r
# One-hot encoding
library(Matrix)
one_hot <- function(sequences, dimension){
  # create 'i' - row #
  seqlen = sapply(sequences, length)
  n = length(seqlen)
  row.ind = rep(1:n, seqlen)

  # create 'j' - column #
  col.ind = unlist(sequences)

  #i,j specify location of non-zero elements
  sparseMatrix(i=row.ind, j=col.ind, dims=c(n, dimension))
}


x_train_1h = one_hot(x_train, 10000)
x_test_1h = one_hot(x_test, 10000)
dim(x_train_1h)
```

```
## [1] 25000 10000
```

```r
nnzero(x_train_1h)/(25000*10000) #only 1.3% contains non-zero value (i.e. 1)
```

```
## [1] 0.01316987
```

First we fit a lasso logistic regression model using glmnet() on the training data, and evaluate its performance on the validation data. Finally, we plot the accuracy, acclmv, as a function of the shrinkage parameter, λ.

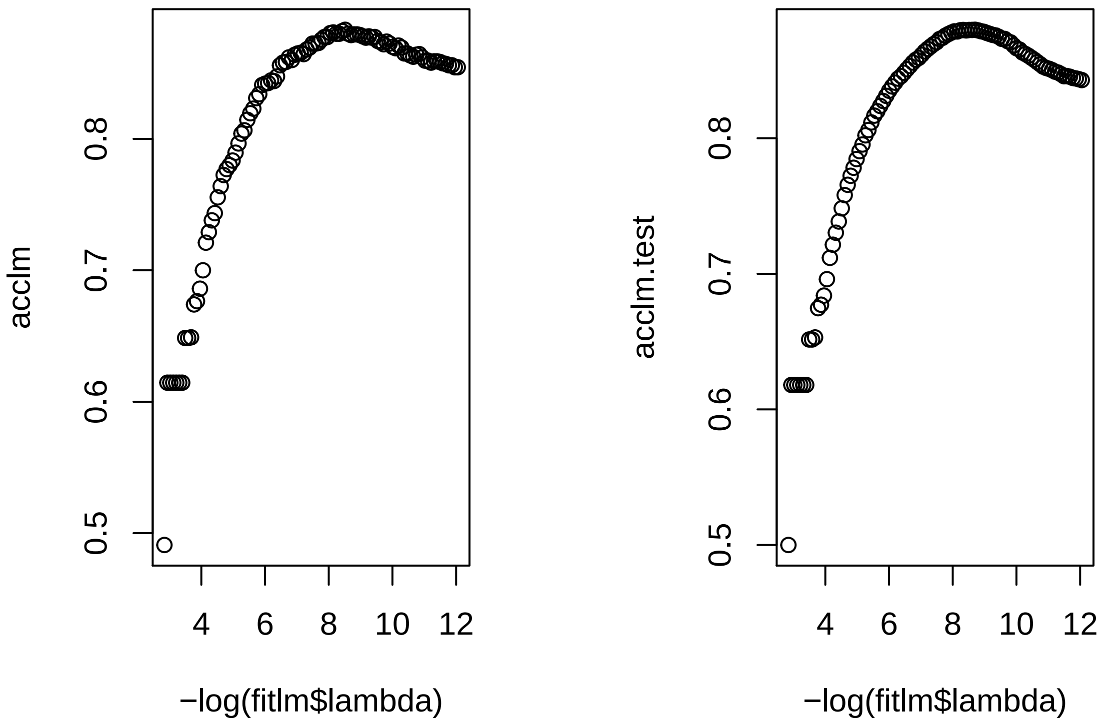
```r
library(glmnet)

#Validation set of 2000... 23000 for training
set.seed(3)
ival = sample(seq(along=y_train), 2000)

fitlm = glmnet(x_train_1h[-ival,], y_train[-ival], family = "binomial",
               standardize = FALSE) #alpha =1 by default, Lasso
classlm = predict(fitlm, x_train_1h[ival,]) > 0
acclm = apply(classlm, 2, accu, y_train[ival]>0)  #y_train is argument to accu

par(mar=c(4,4,4,4), mfrow=c(1,2))
plot(-log(fitlm$lambda), acclm)

# performance on test data: ~85%
classlm.test = predict(fitlm, x_test_1h) > 0
acclm.test = apply(classlm.test, 2, accu, y_test>0)
plot(-log(fitlm$lambda), acclm.test)
```



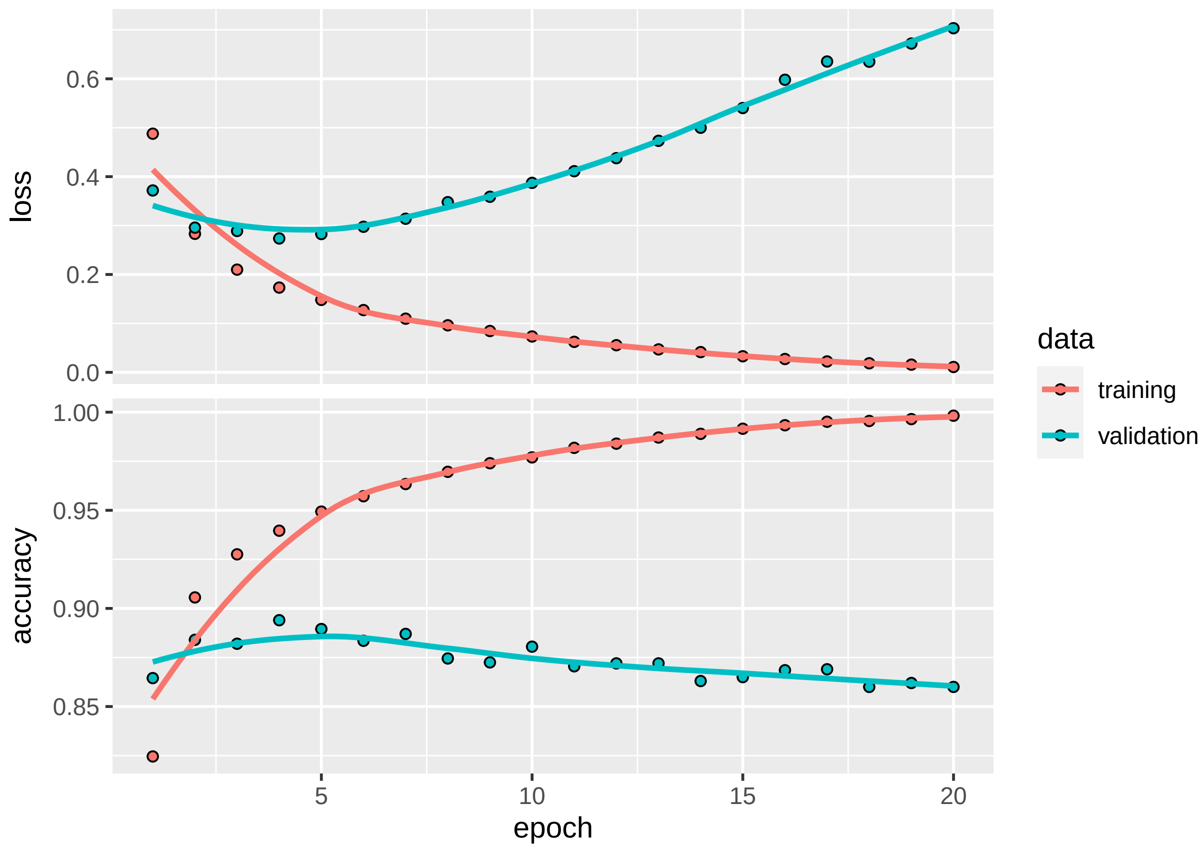
Bag-of-words model

Next we fit a fully-connected neural network with two hidden layers, each with 16 units and ReLU activation

```
modelbow = keras_model_sequential() %>%
  layer_dense(units=16, activation="relu", input_shape=c(10000)) %>%
  layer_dense(units=16, activation="relu") %>%
  layer_dense(units=1, activation="sigmoid")

modelbow %>% compile(optimizer="rmsprop", loss="binary_crossentropy",
                     metrics=c("accuracy"))
history = modelbow %>% fit(x_train_1h[-ival,], y_train[-ival],
                           epochs=20, batch_size=512,
                           validation_data=list(x_train_1h[ival,],y_train[ival]))
plot(history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



RNN - Recurrent Neural Network Sentiment analysis with IMDb data.

```
wc = sapply(x_train, length)
median(wc)      #median word count 178
```

```
## [1] 178
```

```
mean(wc<=500)   #92% of reviews have <500 words
```

```
## [1] 0.91568
```

```
# keeps last 500 words/ adds padding in the beginning
x_train <- pad_sequences(x_train, maxlen=500)
x_test  <- pad_sequences(x_test , maxlen=500)
dim(x_train); dim(x_test)
```

```
## [1] 25000   500
```

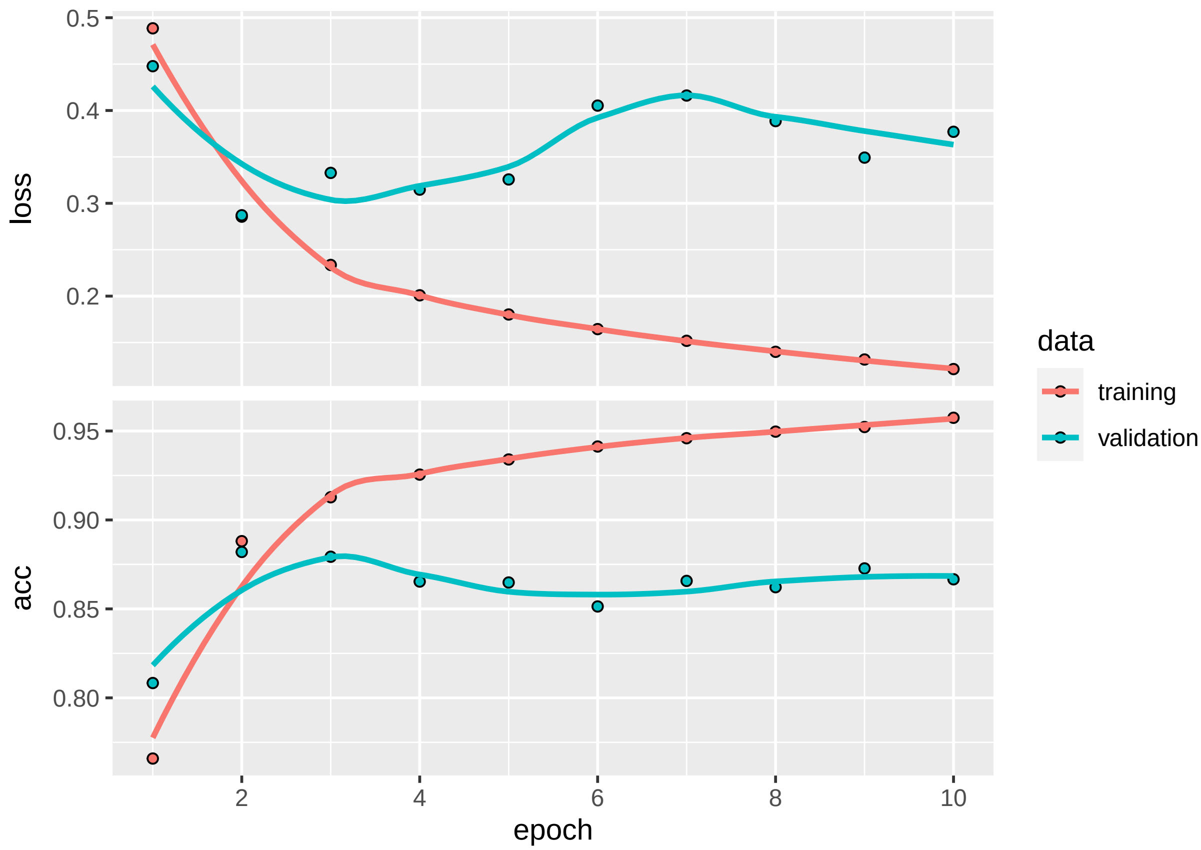
```
## [1] 25000   500
```

At this stage, each of the 500 words in the document is represented using an integer corresponding to the location of that word in the 10,000-word dictionary. The **first layer** of the RNN is an embedding layer of size 32, which will be learned during training. This layer **one-hot encodes each document as a matrix of dimension 500×10,000**, and then maps these 10,000 dimensions down to 32.

```r
modelrnn = keras_model_sequential() %>%
  layer_embedding(input_dim = 10000, output_dim = 32) %>%
  layer_lstm(units=32) %>%
  layer_dense(units=1, activation="sigmoid")

modelrnn %>% compile(optimizer="rmsprop", loss="binary_crossentropy",
                     metrics=c("acc"))
history = modelrnn %>% fit(x_train, y_train, epochs=10,
                           batch_size=128,
                           validation_data=list(x_test, y_test))
plot(history)
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```r
predy = predict(modelrnn, x_test) > 0.5
mean(abs(y_test == as.numeric(predy)))
```

```
## [1] 0.86664
```

Time Series Prediction

1. AR model
2. **RNN**

```r
library(dplyr)
```

AR model

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
xdata=data.matrix(NYSE[,c("DJ_return", "log_volume", "log_volatility")])
istrain = NYSE[,"train"]  # True/ False
xdata = scale(xdata)

volume = data.frame(logvol = xdata[,"log_volume"],
                    L1 = lag(xdata,1), L2 = lag(xdata,2),
                    L3 = lag(xdata,3), L4 = lag(xdata,4), L5 = lag(xdata,5))
colnames(volume) = c("logvol", "L1.ret", "L1.volm", "L1.vol", "L2.ret", "L2.volm",
                     "L2.vol", "L3.ret", "L3.volm", "L3.vol", "L4.ret", "L4.volm",
                     "L4.vol", "L5.ret", "L5.volm", "L5.vol")

volume = volume[-(1:5),]
istrain = istrain[-(1:5)]

#Fitting Linear AR model
arfit = lm(logvol ~., data=volume[istrain,])
arpred= predict(arfit, volume[!istrain,])
summary(arfit)$r.squared
```

```
## [1] 0.570715
```

```
# R-squared on test data ~41%
1 - mean((arpred - volume[!istrain, "logvol"])^2)*(1/var(volume[!istrain, "logvol"]))
```

```
## [1] 0.413223
```

```
volume.wk = data.frame(day=NYSE[-(1:5),"day_of_week"], volume)
arfit.wk = lm(logvol~., data=volume.wk[istrain,])
arpred.wk = predict(arfit.wk, volume.wk[!istrain,])

# R-squared on test data ~46%
1 - mean((arpred.wk - volume[!istrain, "logvol"])^2)*(1/var(volume[!istrain, "logvol"]))
```

```
## [1] 0.4598616
```

RNN - Recurrent Neural Network Need to reshape the data, for RNN since it expects a sequence of L = 5 feature vectors for each observation.

Two forms of dropout for the units feeding into the hidden layer. The first is for the input sequence feeding into this layer, and the second is for the previous hidden units feeding into the layer. The output layer has a single unit for the response.

```r
n = nrow(volume)
xrnn = data.matrix(volume[,-1])
xrnn = array(xrnn, c(n, 3, 5))
dim(xrnn)
```

```
## [1] 6046    3    5
```

```r
xrnn = xrnn[,,5:1]  # reversing order of lag data... index 1 is back
xrnn = aperm(xrnn, c(1,3,2)) # Transpose an array by permuting its dimensions
dim(xrnn)
```

```
## [1] 6046    5    3
```

```r
modelrnn = keras_model_sequential() %>%
  layer_simple_rnn(units=12, input_shape = list(5,3), dropout=0.1,
                   recurrent_dropout = 0.1) %>%
  layer_dense(units=1)

modelrnn %>% compile(optimizer=optimizer_rmsprop(), loss="mse")

history = modelrnn %>% fit(xrnn[istrain,,], volume[istrain,"logvol"],
                           batch_size = 64, epochs = 200,
            validation_data=list(xrnn[!istrain,,], volume[!istrain, "logvol"]))

kpred = predict(modelrnn, xrnn[!istrain,,])

#40%
1 - mean((kpred-volume[!istrain, "logvol"])^2)/var(volume[!istrain, "logvol"])
```

```
## [1] 0.4122657
```

```r
# Fitting linear AR using RNN framework
modelt = keras_model_sequential() %>% layer_flatten(input_shape=c(5,3)) %>%
  layer_dense(units=1)
modelt %>% compile(optimizer=optimizer_rmsprop(), loss="mse")

history = modelt %>% fit(xrnn[istrain,,], volume[istrain,"logvol"],
                           batch_size = 64, epochs = 50,
            validation_data=list(xrnn[!istrain,,], volume[!istrain, "logvol"]))
kpred = predict(modelt, xrnn[!istrain,,])

#41%
1 - mean((kpred-volume[!istrain, "logvol"])^2)/var(volume[!istrain, "logvol"])
```

```
## [1] 0.3997312
```