Simple Neural Network

Hishma Malik

25/11/2021

Introduction

Neural networks have become a popular tool for analyzing datasets where the goal is to develop a complex prediction model which takes a set of input features and tries to predict the result of an outcome (or target) variable. These models work well in situations where the relationship between the input features and the outcome variable is highly nonlinear.

The basic structure of a neural network is as follows: 1) We assume a set of K input features. 2) We choose a number of hidden (unobserved) layers. 3) For each layer, we choose a number of nodes. 4) The inputs, layers, and outcome are connected by edges which have weights which need to be estimated. 5) Every node also has its own bias node that is used to help adjust the linear combinations to improve prediction (similar to an intercept term in linear regression). 6) Each node in each hidden layer contains a linear combination of values of previous nodes that then gets passed through the network to result in a final prediction.

Here we have an example of such a network for the palmerpenguins data, where we try to predict the sex of the penguin from bill length and body mass. I use this to create a set of functions step by step that will work with the data as a basis for creating a set of general functions.

My goal is to create a set of functions that will do all required tasks for ANY data set that contains a single outcome column and a set of possible input features.

Install packages

```
library(tinytex)
library(rmarkdown)
library(knitr)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.5     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.0.2     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tibble)
library(here)
```

```
## here() starts at /Users/hishmamalik/Desktop
```

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

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

```
library(magrittr)
```

```
##
## Attaching package: 'magrittr'
```

```
## The following object is masked from 'package:purrr':
##
##     set_names
```

```
## The following object is masked from 'package:tidyr':
##
##     extract
```

```
library(neuralnet)
```

```
##
## Attaching package: 'neuralnet'
```

```
## The following object is masked from 'package:dplyr':
##
##     compute
```

```
library(palmerpenguins)
```

Explore and explain data

The palmerpenguins package contains two datasets. One is called penguins, and is a simplified version of the raw data. This is what I will use as a foundation.

Penguins contains data for 344 penguins. There are 3 different species of penguins in this dataset, collected from 3 islands in the Palmer Archipelago, Antarctica.

```
head(penguins)
```

```
## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_~ body_mass_g sex
##   <fct>   <fct>           <dbl>         <dbl>            <int>       <int> <fct>
## 1 Adelie  Torge~           39.1          18.7              181        3750 male
## 2 Adelie  Torge~           39.5          17.4              186        3800 fema~
## 3 Adelie  Torge~           40.3          18                195        3250 fema~
## 4 Adelie  Torge~           NA            NA                 NA          NA <NA>
## 5 Adelie  Torge~           36.7          19.3              193        3450 fema~
## 6 Adelie  Torge~           39.3          20.6              190        3650 male
## # ... with 1 more variable: year <int>
```

Step 1

Given the tibble we start with, I will create a function that takes

1) A data frame or tibble
2) A length 1 character vector indicating the name of the outcome column in the dataset.
3) A character vector of unspecified length containing the names of the input features to be selected and scaled

and returns a new data set which contains a tibble containing only the outcome vector which should be renamed outcome and the scaled feature vectors, each of which has been scaled using the scale function.

Note:

The length of the vector of the hidden argument vector is the number of hidden layers. The value of each element of the hidden argument vector is the number of hidden nodes in that respective layer so hidden=c(a,b) means 2 hidden layers with a nodes in the first layer and b nodes in the second layer.

The two input features need to be scaled to have average value 0 and standard deviation 1 in order to use the neuralnet function to fit the models without lots of extra work.

```
first_function <- function(adf, outcome, inputs){

  result <- adf[, c(outcome, inputs)] %>% drop_na %>%
  mutate_at(~scale(.),.vars=vars(inputs)) %>% rename(outcome=sex)

  return(result)
}

# test on penguins data

penguins_example <- penguins %>% drop_na %>% # adjust data bc ours doesnt
  mutate(sex=ifelse(sex=="female",1,0))

test1 <- first_function(penguins_example, c("sex"),
                        c("body_mass_g", "bill_length_mm"))
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use ‘all_of(inputs)‘ instead of ‘inputs‘ to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

```
head(test1)
```

```
## # A tibble: 6 x 3
##   outcome body_mass_g[,1] bill_length_mm[,1]
##     <dbl>           <dbl>              <dbl>
## 1       0          -0.568             -0.895
## 2       1          -0.506             -0.822
## 3       1          -1.19              -0.675
## 4       1          -0.940             -1.33
## 5       0          -0.692             -0.858
## 6       1          -0.723             -0.931
```

I will use this for the rest of project.

Step 2

Since the neural network error generally tends to be under-estimated (because of optimization of weights) I will now split our data into a training sample and a test sample to evaluate the error independently from our estimated weights.

I will do this by writing a function to randomly split any data frame/tibble into training and test that will take two arguments: 1) The data frame or tibble 2) The percentage of the total number of rows that should be from training and returns a list which has two elements, one that is the Training data and the other is the Test data.

I will demonstrate that your function works by running it on the tibble that I generated in the last part with training fraction equal to 0.7.

```
# split df into training and test

sec_function <- function(adf,perc){

  # in case percentage is not divided by 100 by user

  if(perc>1){perc <- (perc/100)}
  else{perc <-perc}

  # split the data
  split_adf <- sample(c("Training", "Test"), prob = c(perc, (1-perc)),
                      replace=T, size=nrow(adf))

  # create training data and test data

  training_sample <- adf %>% filter(split_adf=="Training")

  testing_sample <- adf %>% filter(split_adf=="Test")


  # return list with two elements (training and test data)
  return(list(training_sample, testing_sample))

}
```

```
# test on penguins data
test2 <- sec_function(test1, 0.7 ) # also works if you input 70%
test2
```

```
## [[1]]
## # A tibble: 223 x 3
##    outcome body_mass_g[,1] bill_length_mm[,1]
##      <dbl>           <dbl>              <dbl>
##  1       0        -0.568              -0.895
##  2       1        -0.506              -0.822
##  3       1        -0.723              -0.931
##  4       0        -0.506              -0.986
##  5       0         0.240              -1.72
##  6       1        -0.940              -0.968
##  7       0         0.364              -0.273
##  8       0        -0.00876             0.367
##  9       1        -1.00               -1.13
## 10       1        -0.506              -1.48
## # ... with 213 more rows
##
## [[2]]
## # A tibble: 110 x 3
##    outcome body_mass_g[,1] bill_length_mm[,1]
##      <dbl>           <dbl>              <dbl>
##  1       1          -1.19               -0.675
##  2       1          -0.940              -1.33
##  3       0          -0.692              -0.858
##  4       0           0.581              -0.876
##  5       1          -1.25               -0.529
##  6       1          -0.630              -1.35
##  7       1          -1.10               -1.75
##  8       0          -0.754              -1.15
##  9       1          -0.506              -1.59
## 10       1          -1.19               -0.822
## # ... with 100 more rows
```

PART C : Fit the neural network

Now I can fit the neural network to the training data and compute predictions and average squared error for the training data.

I will create a function that takes: 1) A data frame or tibble formatted as above (with a column named outcome and other columns that are all scaled feature vectors) 2) A vector of integers that can be used as the hidden argument to the neuralnet function, i.e. a list of numbers of nodes of the hidden layers of a neural network

to return a neuralnet object that is the result of running the neuralnet function on the data frame/tibble with the hidden nodes specified from the second argument and the following other arguments: linear.output = FALSE,act.fct="logistic" and using the outcome variable as the outcome in the formula argument. I will show that my function works by running it on the Training Data that I created above.

Note: Setting linear.output=FALSE and using the logistic function (argument of act.funct) converts the output of the neural network into a value between 0 and 1 which can be interpreted as a probability.

```
third_function <- function(adf, vec){


  result <- neuralnet(outcome~.,linear.output = FALSE,
            act.fct="logistic",data=adf, hidden=vec)
}

# training sample is first element of list from b
test3 <- third_function(test2[[1]], c(2,2))


# fitted neural network on test
plot(test3)
```

Explanation

Above, we can see the structure of the network when there are two hidden layers and two hidden nodes per layer. The weights of the edges are in black, the blue edges and nodes are the bias terms. The weights and bias terms are chosen to minimize the sum of squared errors which is sum from i=1 to i= no.of units of the euation: (yi - yihat !)^2 where yi is the binary gender value for the penguin in row i of the data and y"i is the predicted probability from the neural network that this penguin in row i has yi = 1. Other errors are possible in neuralnet , but I consider just the default is for this project.

Note: A better measure is the average error per observation, which I will compute later.

PART D

Now I will compute the predictions and error for the test data. I will createa function that takes:

1) A neuralnet object
2) A data frame/tibble containing Training Data
3) A data frame/tibble containing Test Data

and returns a vector containing the average training squared error and the average test squared error using the neuralnet object to find the predictions.

```
fourth_function <- function(nnobj, df_train, df_test){

  # use predict function on neural net object for training
  train_predict <- predict(nnobj,newdata=df_train)

  Training_Error <- df_train %>%
    mutate(train_error_sq=(outcome-train_predict)^2) %>%
    summarize(Avg1=mean(train_error_sq))

  # use predict function on neural net object for testing
  test_predict <- predict(nnobj,newdata=df_test)

  Testing_Error <- df_test %>% mutate(test_error_sq=(outcome-test_predict)^2) %>%
   summarize(Avg2 = mean(test_error_sq))
```

```
  return(c(Training_Error, Testing_Error))
  #Training_Error
  #Testing_Error
}

# from part b
training_tibble <- test2[[1]]
testing_tibble <- test2[[2]]

test4 <- fourth_function(test3, training_tibble, testing_tibble)
test4
```

```
## $Avg1
## [1] 0.1015088
##
## $Avg2
## [1] 0.1280932
```

Integrate

e. Write a function that takes the following arguments:
f. A data frame or tibble

ii. A length 1 character vector indicating the name of the outcome column in the dataset.
iii. A character vector of unspecified length containing the names of the input features to be selected and scaled.
iv. The percentage of the total number of rows in the data/frame or tibble that should be used in the training data.

and returns a tibble where each row contains the Average Training and Average Test squared error for fitting a two-layer neural network at all possible combinations of numbers of hidden nodes at each layer (1 through 3). Your returned tibble should look like this:

as_tibble(expand.grid(`First layer`=c(1,2,3),`Second layer`=c(1,2,3), `Training Error`=NA, `Test error`=NA))

where the NA's are replaced with the values for your runs. Hint: You can use the expand.grid function above to create a data.frame/tibble that you can iterate over the functions from parts (a) through (d) over the strategies that we have learned in class. Anything that works is acceptable (no need to optimize the speed). Demonstrate that your function works by running it on the penguins_example tibble from the Background section for the two features used in the Background, body_mass_g and bill_length_mm .