NeuralNetwork_Novikov

Vitalii Novikov

Table of contents

Libraries

```
library <- function(...) {suppressPackageStartupMessages(base::library(...))}
if (!require(caret)) install.packages("caret"); library(caret)
```

```
Loading required package: caret
```

```
Loading required package: ggplot2
```

```
Loading required package: lattice
```

```
if (!require(neuralnet)) install.packages("neuralnet"); library(neuralnet)
```

```
Loading required package: neuralnet
```

```
if (!require(mlbench)) install.packages("mlbench"); library(mlbench)
```

```
Loading required package: mlbench
```

```
if (!require(dplyr)) install.packages("dplyr"); library(dplyr)
```

```
Loading required package: dplyr


Attaching package: 'dplyr'

The following object is masked from 'package:neuralnet':

    compute

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
library(plyr)
```

Starting point

We would like to use Neural Network to predict the diagnosis (Benign or Malignant based on 10 features: - radius (mean of distances from center to points on the perimeter) - texture (standard deviation of gray-scale values) - perimeter - area - smoothness (local variation in radius lengths) - compactness (perimeter^2 / area - 1.0) - concavity (severity of concave portions of the contour) - concave points (number of concave portions of the contour) - symmetry - fractal dimension ("coastline approximation" - 1)

Data management

First the data is collected from package "mlbench".

```
data("BreastCancer", package = "mlbench")
BreastCancer |> head()
```

```
       Id Cl.thickness Cell.size Cell.shape Marg.adhesion Epith.c.size
1 1000025            5         1          1             1            2
2 1002945            5         4          4             5            7
3 1015425            3         1          1             1            2
4 1016277            6         8          8             1            3
5 1017023            4         1          1             3            2
6 1017122            8        10         10             8            7
  Bare.nuclei Bl.cromatin Normal.nucleoli Mitoses     Class
1           1           3               1       1    benign
2          10           3               2       1    benign
3           2           3               1       1    benign
4           4           3               7       1    benign
5           1           3               1       1    benign
6          10           9               7       1 malignant
```

But it is wrong dataset.

Get another:

```
dat <- read.csv("data.csv")
glimpse(dat)
```

```
Rows: 569
Columns: 33
$ id                      <int> 842302, 842517, 84300903, 84348301, 84358402, ~
$ diagnosis               <chr> "M", "M", "M", "M", "M", "M", "M", "M", "M", "~
$ radius_mean             <dbl> 17.990, 20.570, 19.690, 11.420, 20.290, 12.450~
$ texture_mean            <dbl> 10.38, 17.77, 21.25, 20.38, 14.34, 15.70, 19.9~
$ perimeter_mean          <dbl> 122.80, 132.90, 130.00, 77.58, 135.10, 82.57, ~
$ area_mean               <dbl> 1001.0, 1326.0, 1203.0, 386.1, 1297.0, 477.1, ~
$ smoothness_mean         <dbl> 0.11840, 0.08474, 0.10960, 0.14250, 0.10030, 0~
$ compactness_mean        <dbl> 0.27760, 0.07864, 0.15990, 0.28390, 0.13280, 0~
$ concavity_mean          <dbl> 0.30010, 0.08690, 0.19740, 0.24140, 0.19800, 0~
$ concave.points_mean     <dbl> 0.14710, 0.07017, 0.12790, 0.10520, 0.10430, 0~
$ symmetry_mean           <dbl> 0.2419, 0.1812, 0.2069, 0.2597, 0.1809, 0.2087~
$ fractal_dimension_mean  <dbl> 0.07871, 0.05667, 0.05999, 0.09744, 0.05883, 0~
$ radius_se               <dbl> 1.0950, 0.5435, 0.7456, 0.4956, 0.7572, 0.3345~
$ texture_se              <dbl> 0.9053, 0.7339, 0.7869, 1.1560, 0.7813, 0.8902~
$ perimeter_se            <dbl> 8.589, 3.398, 4.585, 3.445, 5.438, 2.217, 3.18~
$ area_se                 <dbl> 153.40, 74.08, 94.03, 27.23, 94.44, 27.19, 53.~
$ smoothness_se           <dbl> 0.006399, 0.005225, 0.006150, 0.009110, 0.0114~
$ compactness_se          <dbl> 0.049040, 0.013080, 0.040060, 0.074580, 0.0246~
$ concavity_se            <dbl> 0.05373, 0.01860, 0.03832, 0.05661, 0.05688, 0~
$ concave.points_se       <dbl> 0.015870, 0.013400, 0.020580, 0.018670, 0.0188~
$ symmetry_se             <dbl> 0.03003, 0.01389, 0.02250, 0.05963, 0.01756, 0~
$ fractal_dimension_se    <dbl> 0.006193, 0.003532, 0.004571, 0.009208, 0.0051~
$ radius_worst            <dbl> 25.38, 24.99, 23.57, 14.91, 22.54, 15.47, 22.8~
$ texture_worst           <dbl> 17.33, 23.41, 25.53, 26.50, 16.67, 23.75, 27.6~
$ perimeter_worst         <dbl> 184.60, 158.80, 152.50, 98.87, 152.20, 103.40,~
$ area_worst              <dbl> 2019.0, 1956.0, 1709.0, 567.7, 1575.0, 741.6, ~
$ smoothness_worst        <dbl> 0.1622, 0.1238, 0.1444, 0.2098, 0.1374, 0.1791~
$ compactness_worst       <dbl> 0.6656, 0.1866, 0.4245, 0.8663, 0.2050, 0.5249~
$ concavity_worst         <dbl> 0.71190, 0.24160, 0.45040, 0.68690, 0.40000, 0~
$ concave.points_worst    <dbl> 0.26540, 0.18600, 0.24300, 0.25750, 0.16250, 0~
$ symmetry_worst          <dbl> 0.4601, 0.2750, 0.3613, 0.6638, 0.2364, 0.3985~
$ fractal_dimension_worst <dbl> 0.11890, 0.08902, 0.08758, 0.17300, 0.07678, 0~
$ X                       <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
```

```
nrow(dat)
```

```
[1] 569
```

This dataset include 569 observations. We will only use the "worst-features" for creating neural network.

Pick useful predictors

```
clean_data <- as_tibble(dat[c(2,23:32)])
clean_data <- mutate(clean_data,
  diagnosis = case_match(diagnosis, "M" ~ "malignant", "B"~ "benign")
   ) %>% mutate_if(is.character, as.factor)
#clean_data[['diagnosis']] <- as.factor(clean_data[['diagnosis']])
clean_data |> head(2)
```

```
# A tibble: 2 x 11
  diagnosis radius_worst texture_worst perimeter_worst area_worst
  <fct>            <dbl>         <dbl>           <dbl>      <dbl>
1 malignant         25.4          17.3            185.       2019
2 malignant         25.0          23.4            159.       1956
# i 6 more variables: smoothness_worst <dbl>, compactness_worst <dbl>,
#   concavity_worst <dbl>, concave.points_worst <dbl>, symmetry_worst <dbl>,
#   fractal_dimension_worst <dbl>
```

```
clean_data |> apply(2, function(x) sum(is.na(x)))
```

```
             diagnosis            radius_worst           texture_worst
                     0                       0                       0
       perimeter_worst              area_worst        smoothness_worst
                     0                       0                       0
     compactness_worst         concavity_worst    concave.points_worst
                     0                       0                       0
       symmetry_worst fractal_dimension_worst
                     0                       0
```

There is no missing data, good.

Normalize the data

All predictors have their own scales. We should perform min-max-normalization.

```
clean_data |> summary()
```

```
    diagnosis    radius_worst   texture_worst   perimeter_worst
 benign   :357   Min.   : 7.93   Min.   :12.02   Min.   : 50.41
 malignant:212   1st Qu.:13.01   1st Qu.:21.08   1st Qu.: 84.11
                 Median :14.97   Median :25.41   Median : 97.66
                 Mean   :16.27   Mean   :25.68   Mean   :107.26
                 3rd Qu.:18.79   3rd Qu.:29.72   3rd Qu.:125.40
                 Max.   :36.04   Max.   :49.54   Max.   :251.20
  area_worst     smoothness_worst  compactness_worst concavity_worst
 Min.   : 185.2   Min.   :0.07117   Min.   :0.02729   Min.   :0.0000
 1st Qu.: 515.3   1st Qu.:0.11660   1st Qu.:0.14720   1st Qu.:0.1145
 Median : 686.5   Median :0.13130   Median :0.21190   Median :0.2267
 Mean   : 880.6   Mean   :0.13237   Mean   :0.25427   Mean   :0.2722
 3rd Qu.:1084.0   3rd Qu.:0.14600   3rd Qu.:0.33910   3rd Qu.:0.3829
 Max.   :4254.0   Max.   :0.22260   Max.   :1.05800   Max.   :1.2520
 concave.points_worst symmetry_worst   fractal_dimension_worst
 Min.   :0.00000      Min.   :0.1565   Min.   :0.05504
 1st Qu.:0.06493      1st Qu.:0.2504   1st Qu.:0.07146
 Median :0.09993      Median :0.2822   Median :0.08004
 Mean   :0.11461      Mean   :0.2901   Mean   :0.08395
 3rd Qu.:0.16140      3rd Qu.:0.3179   3rd Qu.:0.09208
 Max.   :0.29100      Max.   :0.6638   Max.   :0.20750
```

```r
maxs <- apply(clean_data[-1], 2, max)
mins <- apply(clean_data[-1], 2, min)

scaled_data <- scale(clean_data[-1], center = mins, scale = maxs - mins) %>%
  cbind(clean_data[1])

summary(scaled_data)
```

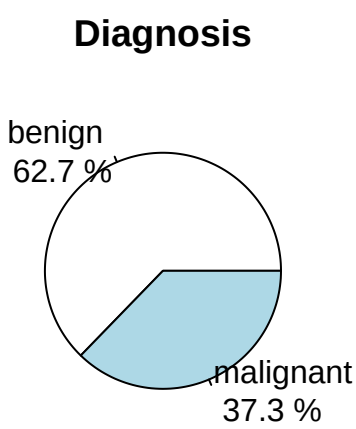
```
 radius_worst    texture_worst    perimeter_worst    area_worst
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000
 1st Qu.:0.1807   1st Qu.:0.2415   1st Qu.:0.1678   1st Qu.:0.08113
 Median :0.2504   Median :0.3569   Median :0.2353   Median :0.12321
 Mean   :0.2967   Mean   :0.3640   Mean   :0.2831   Mean   :0.17091
 3rd Qu.:0.3863   3rd Qu.:0.4717   3rd Qu.:0.3735   3rd Qu.:0.22090
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000
 smoothness_worst compactness_worst concavity_worst   concave.points_worst
 Min.   :0.0000   Min.   :0.0000    Min.   :0.00000   Min.   :0.0000
 1st Qu.:0.3000   1st Qu.:0.1163    1st Qu.:0.09145   1st Qu.:0.2231
 Median :0.3971   Median :0.1791    Median :0.18107   Median :0.3434
 Mean   :0.4041   Mean   :0.2202    Mean   :0.21740   Mean   :0.3938
```

```
 3rd Qu.:0.4942   3rd Qu.:0.3025    3rd Qu.:0.30583   3rd Qu.:0.5546
 Max.   :1.0000   Max.   :1.0000    Max.   :1.00000   Max.   :1.0000
 symmetry_worst   fractal_dimension_worst     diagnosis
 Min.   :0.0000   Min.   :0.0000          benign   :357
 1st Qu.:0.1851   1st Qu.:0.1077          malignant:212
 Median :0.2478   Median :0.1640
 Mean   :0.2633   Mean   :0.1896
 3rd Qu.:0.3182   3rd Qu.:0.2429
 Max.   :1.0000   Max.   :1.0000
```

Now we have all predictors in the same scale.

Balance of the data

```r
#custom function to create pie chart
pie_with_percentages <- function(diag_list, main = "Diagnosis", round = 1) {
  diag_table <- table(diag_list)
  diag_percentages <- prop.table(diag_table) * 100
  labels <- paste(names(diag_table), "\n", round(diag_percentages, round), "%")
  return(pie(diag_table, labels = labels, main = main))
}

pie_with_percentages(scaled_data$diagnosis)
```



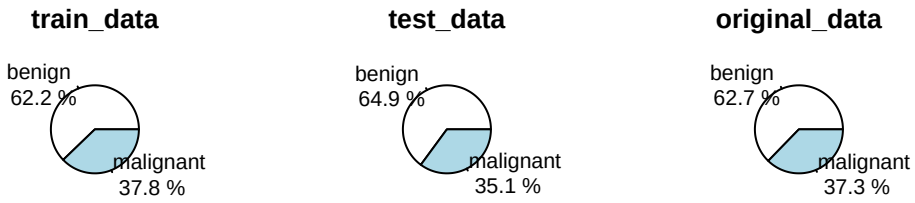
Our dataset do not show the perfect balance of diagnosis, but it is still not that bad as 90/10 or 99/1 split. Neural networks, by default, tend to favor the majority class. This means the model might become biased towards predicting "benign," potentially leading to poor performance in identifying "malignant" cases.

In the next step stratified data set will be created to minimize the potential bias.

Stratified data set

Using 'sample' function

```
set.seed(230)
index <- sample(1:nrow(scaled_data),round(0.8*nrow(scaled_data)))
train_data <- scaled_data[index,]
test_data <- scaled_data[-index,]

layout(matrix(c(1,2,3), nrow = 1), respect = TRUE)
pie_with_percentages(train_data$diagnosis, main = "train_data")
pie_with_percentages(test_data$diagnosis, main = "test_data")
pie_with_percentages(clean_data$diagnosis, main = "original_data")
```

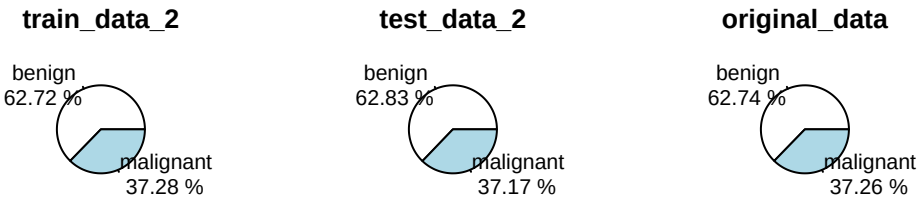



```
layout(1)
```

The test data show slightly lowest proportion of malignant then both train and original data, but it is still very close to actual picture.

Using caret library

```
set.seed(123)
index_2 <- createDataPartition(scaled_data$diagnosis, p = 0.8, list = FALSE)

train_data_2 <- scaled_data[index_2,]
test_data_2 <- scaled_data[-index_2,]

layout(matrix(c(1,2,3), nrow = 1), respect = TRUE)
pie_with_percentages(train_data_2$diagnosis, main = "train_data_2", round = 2)
pie_with_percentages(test_data_2$diagnosis, main = "test_data_2", round = 2)
pie_with_percentages(clean_data$diagnosis, main = "original_data", round = 2)
```



In this case we got really stratified train and test split of the data, because the proportions of classes are almost the same in each dataset. For further steps we will use this split of the data.

```
trainD <- train_data_2
testD <- test_data_2

split_description <- t(rbind(
  data.frame("trainD" = nrow(trainD), "testD" = nrow(testD)),
  c(round(nrow(trainD)/nrow(clean_data)*100,2),
    round(nrow(testD)/nrow(clean_data)*100,2))
))
colnames(split_description) <- c("nrow","percentage (%)")
split_description
```

```
       nrow percentage (%)
trainD  456          80.14
testD   113          19.86
```

The trainD include 456 rows, which is 80.14% of the original dataset.

Hyperparameters tuning

First NN

```
set.seed(1)
model1 <- neuralnet(diagnosis ~ . , data = trainD, hidden = c(5,3), linear.output=FALSE)
plot(model1)
```

As an example of NN the model with 2 hidden layers (with 5 and 3 neurons) were created. The black lines show the connections between each layer and the weights on each connection while the blue lines show the bias term added in each step.

Model evaluation

The most interesting thing to explore after creating classification model is confusion matrix. But in this step the real test data were used to evaluate the model. For further steps the validation data will be crated.

```
pred <- neuralnet::compute(model1, testD[, -11])$net.result

labels <- c("benign", "malignant")
prediction_label <- (data.frame(max.col(pred)) %>%
  mutate(pred=labels[max.col.pred.]))[2] %>%
  unlist()

table(testD$diagnosis, prediction_label)
```

```
           prediction_label
            benign malignant
  benign        69         2
  malignant      2        40
```

The confusion matrix shows that the model made 4 mistakes.

```
check = as.numeric(testD$diagnosis) == max.col(pred)
accuracy = (sum(check)/nrow(testD))*100
print(accuracy)
```

```
[1] 96.46018
```

Therefore the accuracy is about 96.5%.

But we would like to use **cross-entropy** as the error function. There is no ready-to-use *cross-entropy loss* function in neuralnet, therefore we should calculate loss manually.

```
true_one_hot <- testD["diagnosis"] %>%
  mutate(diagnosis = case_match(diagnosis,
    "malignant" ~ 0,
    "benign" ~ 1
))

cross_entropy <- -sum(true_one_hot * log(pred)) / nrow(testD)
cross_entropy
```

```
[1] 0.3530022
```

It is hard to interpret this number, because it is not a direct loss function like RSME. But we would like to minimize Cross Entropy when compare models.

Change hidden level parameters

First, the 5 fold stratified data split is created.

```
set.seed(321)
folds <- createFolds(trainD$diagnosis, k = 5, list = TRUE, returnTrain = FALSE)
```

```
possible_levels <- c(1,2)
possible_neurons <- 2:5
validation_table <- data.frame(hidden_levels=integer(0),
                               l1_neurons=integer(0),
                               l2_neurons=integer(0),
                               ce1=numeric(0),
                               ce2=numeric(0),
                               ce3=numeric(0),
                               ce4=numeric(0),
                               ce5=numeric(0),
                               cross_entropy_avg=numeric(0),
                               accuracy_avg=numeric(0))

# 1 hidden level loop
for(n in possible_neurons){
```

```
  accuracy_list = numeric(0)
  cross_entropy_list = numeric(0)
  for(i in 1:5) {
    test_index <- folds[[i]]
    train_fold <- trainD[-test_index, ]
    test_fold <- trainD[test_index, ]
    set.seed(5)
    model_nn <- neuralnet(diagnosis ~ . , data = train_fold, hidden = n, linear.output=FALSE)
    pred <- neuralnet::compute(model_nn, test_fold[, -11])$net.result
    #ce
    true_one_hot <- test_fold["diagnosis"] %>%
    mutate(diagnosis = case_match(diagnosis,
        "malignant" ~ 0,
        "benign" ~ 1
    ))
    cross_entropy_list[i] <- -sum(true_one_hot * log(pred)) / nrow(test_fold)
    #ac
    check = as.numeric(test_fold$diagnosis) == max.col(pred)
    accuracy_list[i] <- (sum(check)/nrow(test_fold))*100
  }
  validation_table[nrow(validation_table) + 1, ] <- c(1, n, NA,
                                                      cross_entropy_list,
                                                      mean(cross_entropy_list),
                                                      mean(accuracy))
}
# 2 hidden levels loop
for(n1 in possible_neurons){
  for(n2 in possible_neurons){
    accuracy_list = numeric(0)
    cross_entropy_list = numeric(0)
    for(i in 1:5) {
      test_index <- folds[[i]]
      train_fold <- trainD[-test_index, ]
      test_fold <- trainD[test_index, ]
      set.seed(5)
      model_nn <- neuralnet(diagnosis ~ . , data = train_fold, hidden = c(n1,n2), linear.outp
      pred <- neuralnet::compute(model_nn, test_fold[, -11])$net.result
      #ce
      true_one_hot <- test_fold["diagnosis"] %>%
      mutate(diagnosis = case_match(diagnosis,
          "malignant" ~ 0,
          "benign" ~ 1
```

```
      ))
      cross_entropy_list[i] <- -sum(true_one_hot * log(pred)) / nrow(test_fold)
      #ac
      check = as.numeric(test_fold$diagnosis) == max.col(pred)
      accuracy_list[i] = (sum(check)/nrow(test_fold))*100
    }
    validation_table[nrow(validation_table) + 1, ] <- c(2, n1, n2,
                                                        cross_entropy_list,
                                                        mean(cross_entropy_list),
                                                        mean(accuracy_list))

  }
}

apply(validation_table,2 ,function(x) round(x,2))
```

```
   hidden_levels l1_neurons l2_neurons  ce1  ce2  ce3 ce4  ce5
1              1          2         NA 0.38 0.50 0.09   0 0.09
2              1          3         NA 0.16 0.36 0.07   0 0.15
3              1          4         NA 0.19 0.45 0.04   0 0.00
4              1          5         NA 0.17 0.34 0.12   0 0.01
5              2          2          2 0.16 0.11 0.05   0 0.00
6              2          2          3 0.12 0.49 0.05   0 0.12
7              2          2          4 0.15 0.64 0.10   0 0.00
8              2          2          5 0.23 0.11 0.04   0 0.04
9              2          3          2 0.13 1.48 0.13   0 0.05
10             2          3          3 0.00 1.10 0.12   0 0.05
11             2          3          4 0.22 0.50 0.02   0 0.18
12             2          3          5 0.30 0.51 0.13   0 0.00
13             2          4          2 0.03 0.06 0.01   0 0.11
14             2          4          3 0.13 0.27 0.04   0 0.07
15             2          4          4 0.00 0.07 0.05   0 0.04
16             2          4          5 0.21 0.50 0.02   0 0.07
17             2          5          2 0.17 0.07 0.02   0 0.11
18             2          5          3 0.04 0.21 0.08   0 0.15
19             2          5          4 0.12 0.54 0.11   0 0.05
20             2          5          5 0.03 0.32 0.08   0 0.08
   cross_entropy_avg accuracy_avg
1               0.21        96.46
2               0.15        96.46
3               0.14        96.46
4               0.13        96.46
5               0.07        96.93
```

```
6               0.16        96.27
7               0.18        96.93
8               0.08        96.93
9               0.36        96.27
10              0.25        96.71
11              0.19        97.37
12              0.19        96.27
13              0.04        96.49
14              0.10        97.37
15              0.03        96.05
16              0.16        96.93
17              0.07        96.93
18              0.10        96.49
19              0.16        96.49
20              0.10        96.71
```

Somehow cross entropy for the 4th fold is 0

Recheck it:

```
validation_table$ce4
```

```
 [1] 2.550464e-05 2.147966e-05 1.177995e-05 9.935263e-06 5.066267e-06
 [6] 4.548479e-06 3.116685e-06 3.684743e-06 2.777155e-05 5.998682e-04
[11] 9.903698e-07 6.941158e-06 1.408572e-05 7.800080e-06 5.397184e-06
[16] 3.366158e-06 4.751372e-05 1.712628e-05 3.386463e-06 2.708440e-06
```

It is not 0 in most cases, but very close to zero.

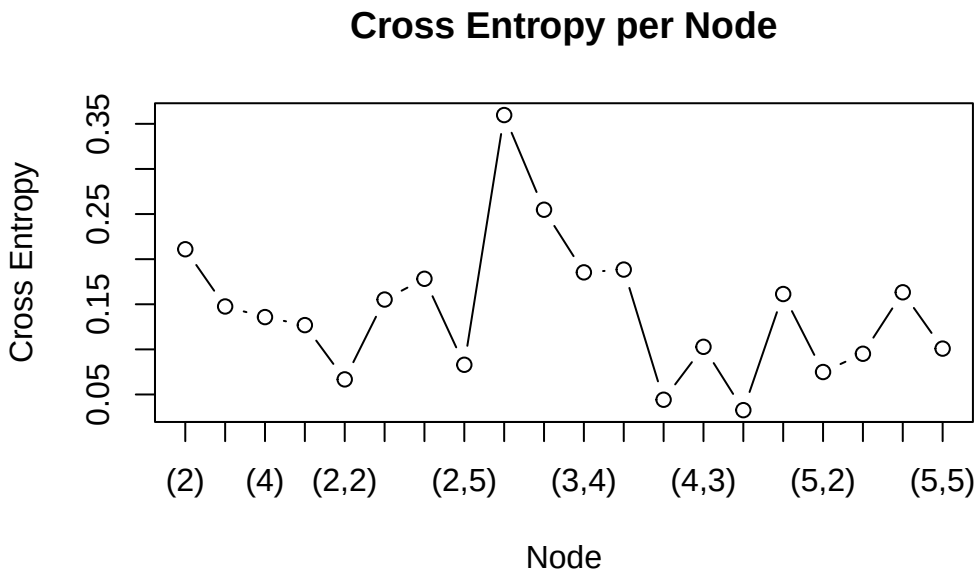
```
vt <- validation_table %>% mutate(
  node = case_when(
    hidden_levels==1 ~ paste0("(",l1_neurons,")"),
    hidden_levels==2 ~ paste0("(",l1_neurons,",",l2_neurons,")")
    )
)
vt$node <- factor(vt$node, levels = vt$node)
vt[c("node", "cross_entropy_avg", "accuracy_avg")] %>%
  sort_by(vt$cross_entropy_avg) %>%
  head() %>% knitr::kable()
```

	node	cross_entropy_avg	accuracy_avg
15	(4,4)	0.0326244	96.05351
13	(4,2)	0.0442029	96.49307
5	(2,2)	0.0667386	96.93024
17	(5,2)	0.0749148	96.93263
8	(2,5)	0.0829441	96.93024
18	(5,3)	0.0951407	96.49068

This table represents lowest cross_entropy_avg for models.

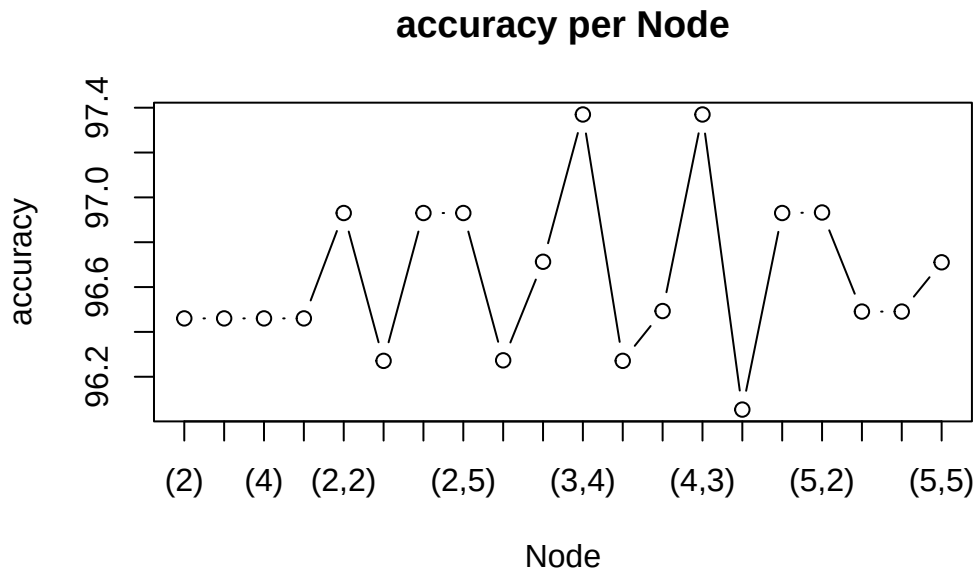
The neural network with hyperparametres (4,4) shows the lowest cross entropy loss. This model will be used for further steps.

```
plot(1:length(vt$node), vt$cross_entropy_avg, type = "b",
     xlab = "Node", ylab = "Cross Entropy",
     main = "Cross Entropy per Node", xaxt = "n")

axis(1, at = 1:length(vt$node), labels = vt$node)
```



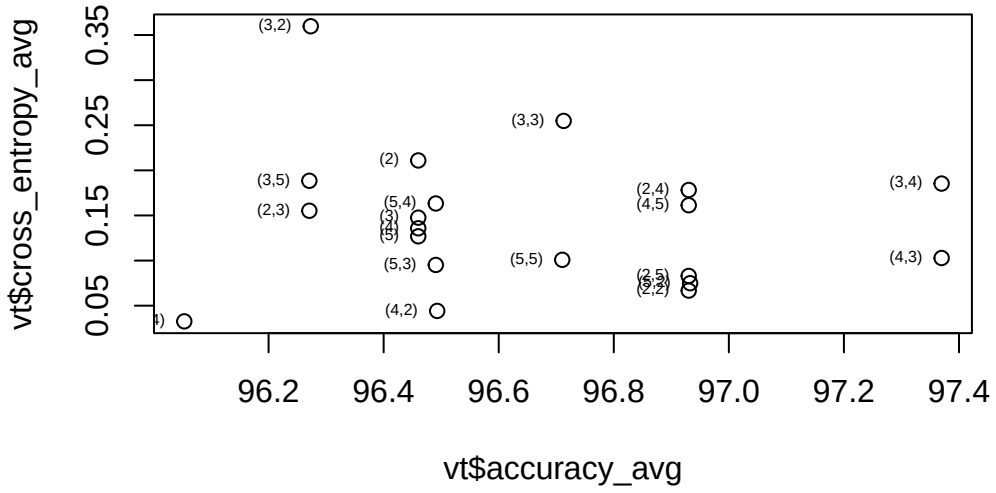
There is no special pattern in cross_entropy_avg when the number of neurons increases.

```
plot(1:length(vt$node), vt$accuracy_avg, type = "b",
     xlab = "Node", ylab = "accuracy",
     main = "accuracy per Node", xaxt = "n")

axis(1, at = 1:length(vt$node), labels = vt$node)
```



The accuracy flactuates between 96 and 97 in most cases, but for (3,4) and (4,3) it becomes larger then 97, whereas (4,4) nn shows the lowest accuracy within networks (96.05).

```
plot(x=vt$accuracy_avg, y=vt$cross_entropy_avg)
text(x = vt$accuracy_avg, y = vt$cross_entropy_avg, labels = vt$node, pos = 2, cex = 0.5)
```



Another possible neural network, that could be good choice for using is nn with 2 hidden layers and (4,3) neurons on these layers, because it is the closest to right bottom corner, which means that the model shows good balance of accuracy and loss function.

But we would like to choose the model with the lowest cross entropy loss, so the (4,4) is a winning model.

Final model

Train & test model

Now the whole trainD set is used for training.

```
set.seed(2)
nn <- neuralnet(diagnosis ~ . , data = testD, hidden = c(4,4), linear.output=FALSE)
pred <- neuralnet::compute(nn, testD[, -11])$net.result
#ce
true_one_hot <- testD["diagnosis"] %>%
mutate(diagnosis = case_match(diagnosis,
    "malignant" ~ 0,
    "benign" ~ 1
))
```

```
cross_entropy <- -sum(true_one_hot * log(pred)) / nrow(testD)
#ac
check = as.numeric(testD$diagnosis) == max.col(pred)
accuracy = (sum(check)/nrow(testD))*100

# confusion matrix
labels <- c("benign", "malignant")
prediction_label <- (data.frame(max.col(pred)) %>%
    mutate(pred=labels[max.col.pred.]))[2] %>%
    unlist()

table(testD$diagnosis, prediction_label)
```

```
           prediction_label
            benign malignant
  benign        71         0
  malignant      0        42
```

The confusion matrix shows that the model performed with flying colors, because there is no false negative and false positive predictions

```
print(paste0("Accuracy: ",accuracy,"%"))
```

```
[1] "Accuracy: 100%"
```

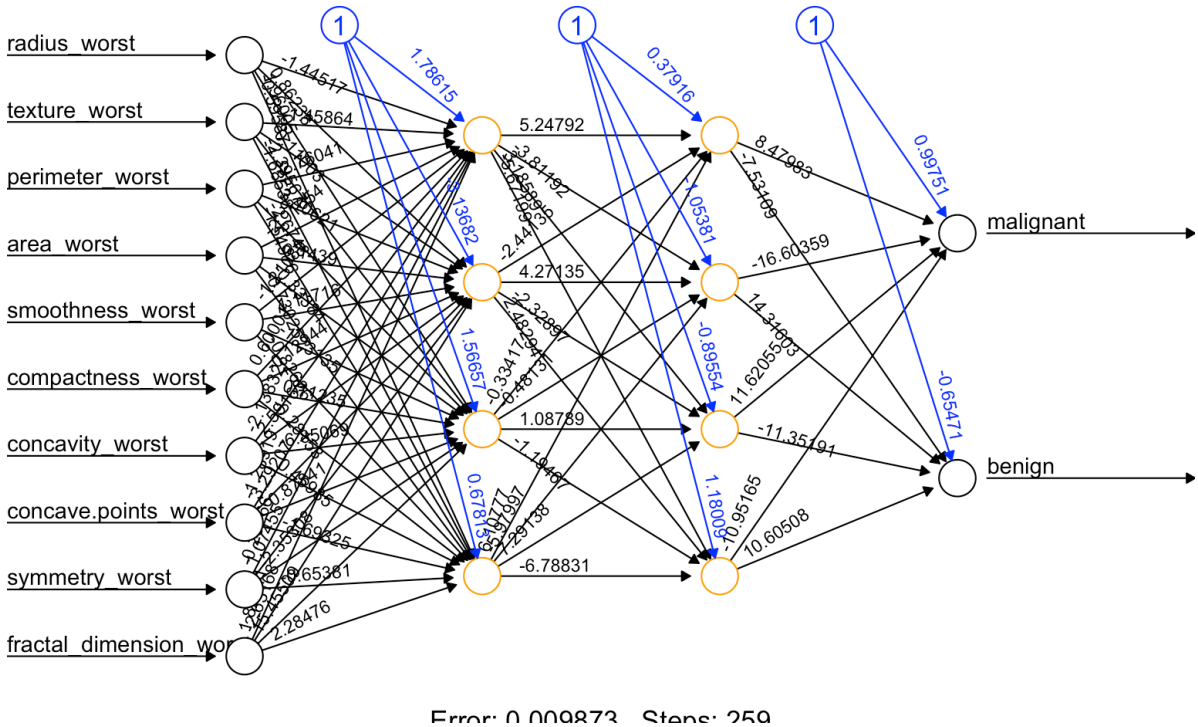
```
print(paste0("Cross Entropy: ",round(cross_entropy,4)))
```

```
[1] "Cross Entropy: 0.0013"
```

So, the accuracy is 100%, while the loss is very comparing to models that were on previous steps.

Model vizualization

```
png("neuralnet_plot.png")
plot(nn, col.hidden = "orange", fontsize = 10, information.pos = 0.25)
knitr::include_graphics("neuralnet_plot.png")
```



The black lines show the connections between each layer and the weights on each connection, while the blue lines show the bias term added at each step. The black neurons on the left are inputs, and the black neurons on the right are outputs. Orange neurons are part of the hidden layers.

Problem of overfitting

Overfitting means a neural network performs well on training data, poorly on unseen data. It memorizes, not generalizes. The detection of it can be via comparing training and validation performance, because a large gap between them indicates overfitting. Some ways to prevent overfitting: - Early stopping (of course with monitoring validation loss) - Regularization (L1/L2), but it reduces model complexity - Dropinng randomly disables neurons - Expandding the dataset to enhance the variety of training data.

There is no problem with overfitting in the final model, but sometimes neural networks tend to be overfited especially if the data is not balanced. In this case the balance of resulting class is 37/63, which is not so bad. Also we used stratified split to perform as good as possible.