Lab Sheet - 9

Neural Network

Foundations of Data Science

CS F320

I Semester 2021-22

---

1. Introduction

A neural network is a model characterized by an activation function, which is used by interconnected information processing units to transform input into output. A neural network has always been compared to human nervous system. Information in passed through interconnected units analogous to information passage through neurons in humans. The first layer of the neural network receives the raw input, processes it and passes the processed information to the hidden layers. The hidden layer passes the information to the last layer, which produces the output. The advantage of neural network is that it is adaptive in nature. It learns from the information provided, i.e. trains itself from the data, which has a known outcome and optimizes its weights for a better prediction in situations with unknown outcome.

The neuralnet package requires an all numeric input data.frame / matrix. Control the hidden layers with hidden and it can be a vector for multiple hidden layers.

We are going to use the Boston dataset in the MASS package. The Boston dataset is a collection of data about housing values in the suburbs of Boston. Our goal is to predict the median value of owner-occupied homes (medv) using all the other continuous variables available.

```
set.seed(500)
library(MASS)
data <- Boston
```

Check that no datapoint is missing, otherwise need to fix the dataset.

```
apply(data,2,function(x) sum(is.na(x)))
```

Now randomly splitting the data into a train and a test set, then we fit a linear regression model and test it on the test set

```
index <- sample(1:nrow(data),round(0.75*nrow(data)))
train <- data[index,]
test <- data[-index,]
lm.fit <- glm(medv~., data=train)
summary(lm.fit)
pr.lm <- predict(lm.fit,test)
MSE.lm <- sum((pr.lm - test$medv)^2)/nrow(test)
```

Normalize your data before training a neural network

```
maxs <- apply(data, 2, max)
mins <- apply(data, 2, min)
scaled <- as.data.frame(scale(data, center = mins, scale = maxs - mins
))
train_ <- scaled[index,]
test_ <- scaled[-index,]
```

Usually, one hidden layer is enough for a vast numbers of applications. As far as the number of neurons is concerned, it should be between the input layer size and the output layer size, usually 2/3 of the input size. At least in my brief experience testing again and again is the best solution since there is no guarantee that any of these rules will fit your model best. Since this is a toy example, we are going to use 2 hidden layers with this configuration: 13:5:3:1. The input layer has 13 inputs, the two hidden layers have 5 and 3 neurons and the output layer has, of course, a single output since we are doing regression.

```
library(neuralnet)
n <- names(train_)
f <- as.formula(paste("medv ~", paste(n[!n %in% "medv"], collapse = "
+ ")))
nn <- neuralnet(f,data=train_,hidden=c(5,3),linear.output=T)
```

A couple of notes:

- For some reason the formula y~ is not accepted in the neuralnet() function. You need to first write the formula and then pass it as an argument in the fitting function.
- The hiden argument accepts a vector with the number of neurons for each hidden layer, while the argument linear.output is used to specify whether we want to do regression linear.output = TRUE or classification linear.output = FALSE

The neuralnet package provides a nice tool to plot the model:

```
plot(nn)
```

Predict the values for the test set and calculate the MSE.

```
pr.nn <- compute(nn,test_[,1:13])
pr.nn_ <- pr.nn$net.result*(max(data$medv)-min(data$medv))+min(data$me
dv)
test.r <- (test_$medv)*(max(data$medv)-min(data$medv))+min(data$medv)
MSE.nn <- sum((test.r - pr.nn_)^2)/nrow(test_)
```

Compare the two MSEs (neural network and linear model)

```
print(paste(MSE.lm,MSE.nn))
```

Visually analyses the performance of the network and the linear model on the test set.

```
par(mfrow=c(1,2))
```

```
plot(test$medv,pr.nn_,col='red',main='Real vs predicted NN',pch=18,cex
=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='NN',pch=18,col='red', bty='n')
plot(test$medv,pr.lm,col='blue',main='Real vs predicted lm',pch=18, ce
x=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='LM',pch=18,col='blue', bty='n', cex=.95)
```

2. Cross validation

Cross validation is another very important step of building predictive models. While there are different kind of cross validation methods, the basic idea is repeating the following process a number of time:
train-test split

- Do the train-test split
- Fit the model to the train set
- Test the model on the test set
- Calculate the prediction error
- Repeat the process K times

Then by calculating the average error we can get a grasp of how the model is doing.

Exercise:

- Try multiple hidden layers and compare the results (start with one hidden layer then increase it by one and check how it affects the MSE).
- Write code in R for backpropagation algorithm for the example explained in the class**.**