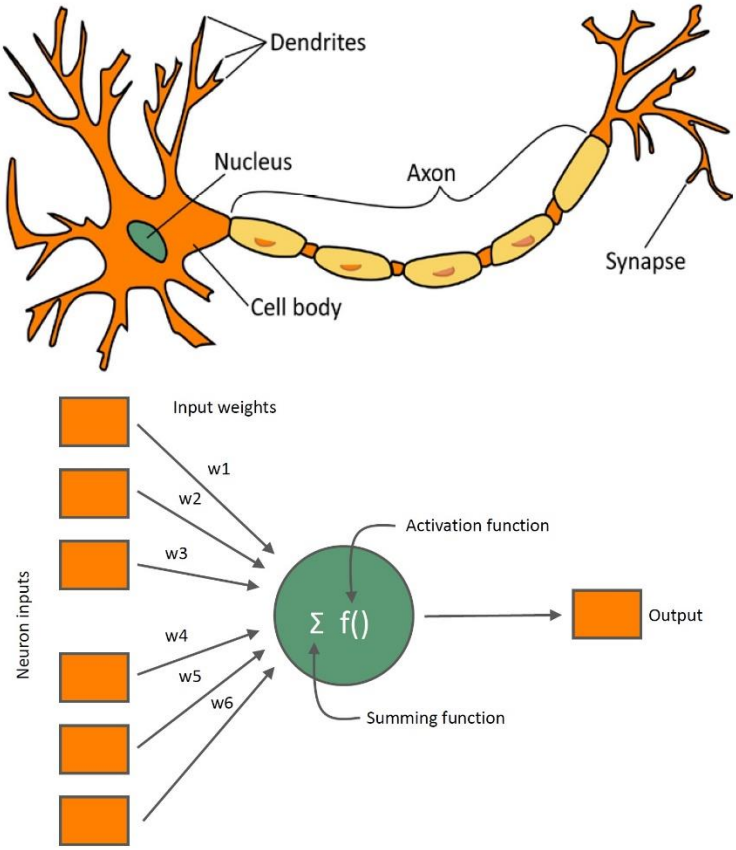
A PRIMER ON NEURAL NETWORKS

Contents

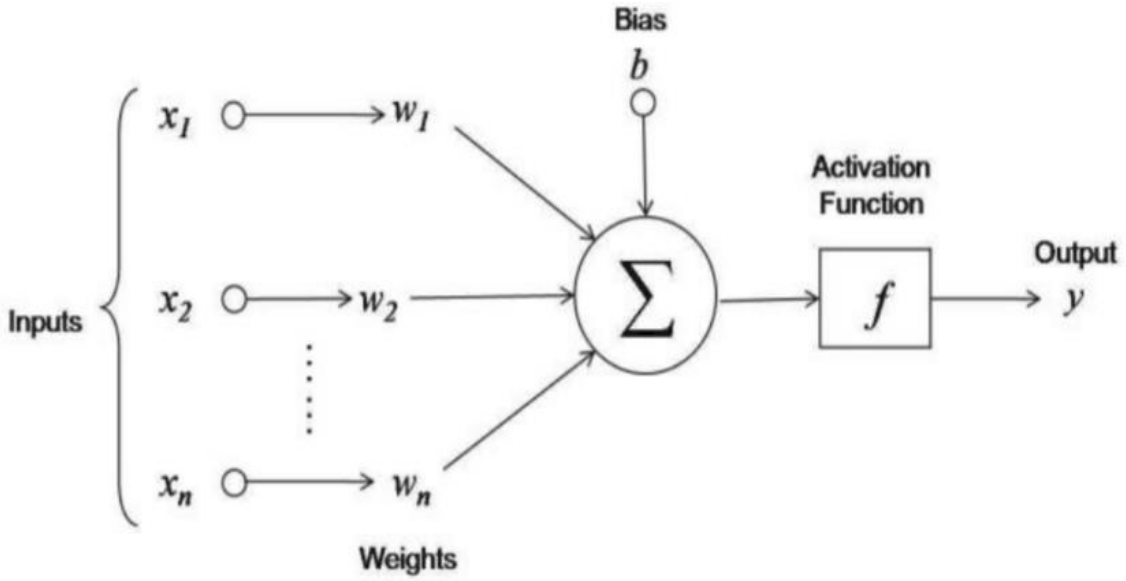
Introduction

- In 1943, Warren McCulloch and Walter Pitts developed the first mathematical model of a neuron.
- NN algorithms are inspired by the human brain to perform a particular task or functions.



- Neural networks perform computations through a process by learning.
- A neural network is a set of connected input/output units in which each connection has a weight associated with it.
- In the learning phase, the network learns by adjusting the weights to predict the correct class label of the given inputs.

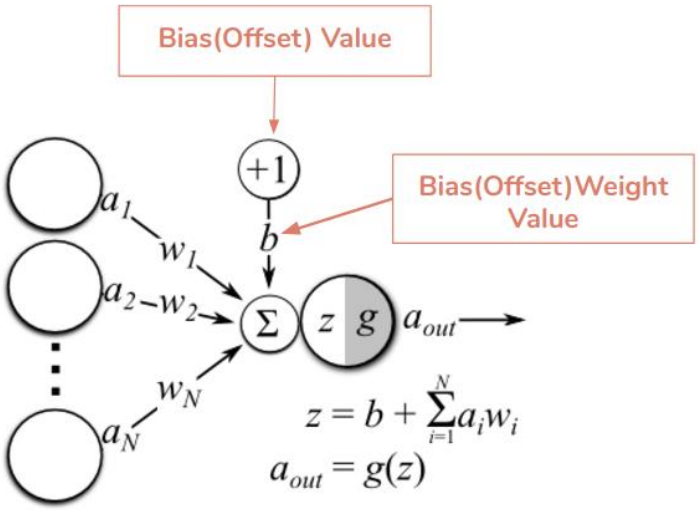
The Components



$$Y = \sum (weight * input) + bias$$

The Key Elements of an Artificial Neural Network

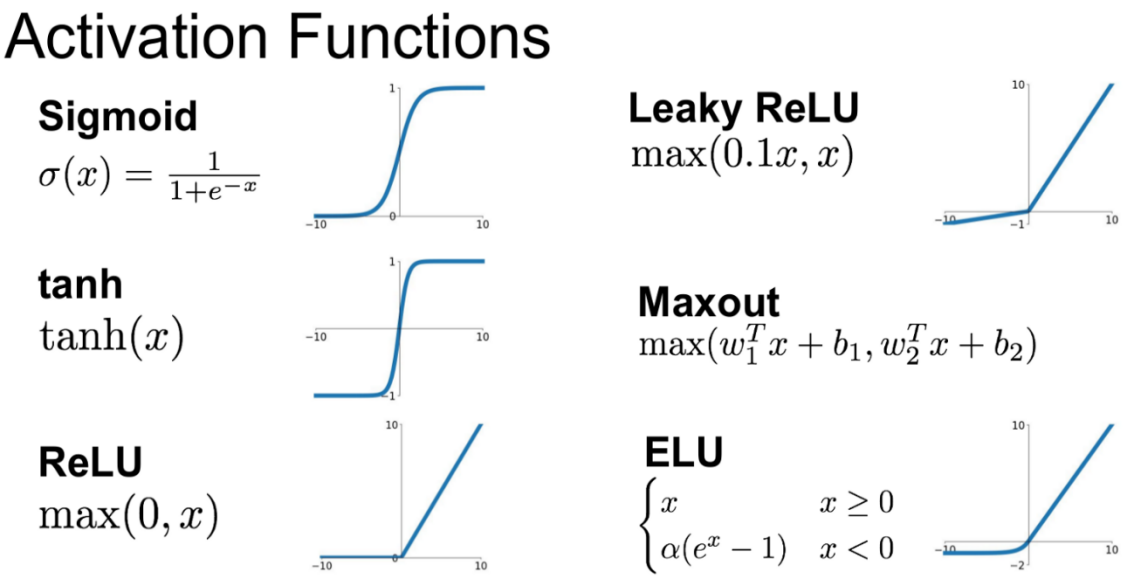
- The **input** variables x_1, x_2, …, x_n represent the neurons.
- The **weights/parameters/connections** of the respective inputs w_1, w_2, …, w_n represent the strength of the connection between units.



- The **biases** (b_n) which are summed with the weighted inputs to form the net inputs. Biases and weights are both adjustable parameters of the neuron. Parameters are adjusted using some learning rules.
- **Hyper-parameters** are the variables which determines the network structure such as the number of hidden units and the variables that determine how the network is trained (learning rate).
- **Activation functions** are designed to map inputs to outputs because the outputs of neurons can range from -∞ to +∞.
    - Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it.
    - The purpose of the activation function is to **introduce non-linearity** into the output of a neuron.
    - A neural network without an activation function is essentially just a linear regression model.
    - The activation function does the **non-linear transformation** to the input making it capable to learn and perform more complex tasks. Their main purpose is to

convert an input signal of a node in an ANN to an output signal. That output signal now is used as an input in the next layer in the stack.

- There are many activation functions. Here are a few examples. **ELU** stands for exponential linear unit and **ReLu** means Rectified Linear Unit. ReLu is the most frequently used activation function: It output 0 for negative values of x.



- **Layer** refers to **input** layer (initial data), **hidden** layer (intermediate layer between input and output layers where all computations are done), and **output** layer (the result for given inputs).

The Types of Artificial Neural Networks

There are two main types of artificial neural networks:

- Feedforward and
- Feedback.

- Feedforward neural network is a network which is not recursive. Neurons in this layer were only connected to neurons in the next layer, and they do not form a cycle. In feedforward, signals travel in only one direction towards the output layer.
- Feedback neural networks contain cycles. Signals travel in both directions by introducing loops in the network. The feedback cycles can cause the network's behavior to change over time based on its input. Feedback neural network are also known as recurrent neural networks.

The Pro and Cons of Artificial Neural Networks

- Neural networks are more flexible and can be used with both regression and classification problems.
- Neural networks are good for nonlinear datasets with a large number of inputs such as images.
- Neural networks can work with any number of inputs and layers.
- Neural networks have the numerical strength that can perform jobs in parallel.
- There can represent an alternative algorithm to SVM, Decision Tree and Regression. They can provide better performance.
- However, a neural network is much more of a black box, requires more time for development and more computation power.
- Neural networks require more data than other Machine Learning algorithms.

- NNs can be used only with numerical inputs and non-missing value datasets.
- A well-known neural network researcher said *"A neural network is the second best way to solve any problem. The best way is to actually understand the problem."* John Denker

Applications

ANN's wonderful properties offer many applications such as:

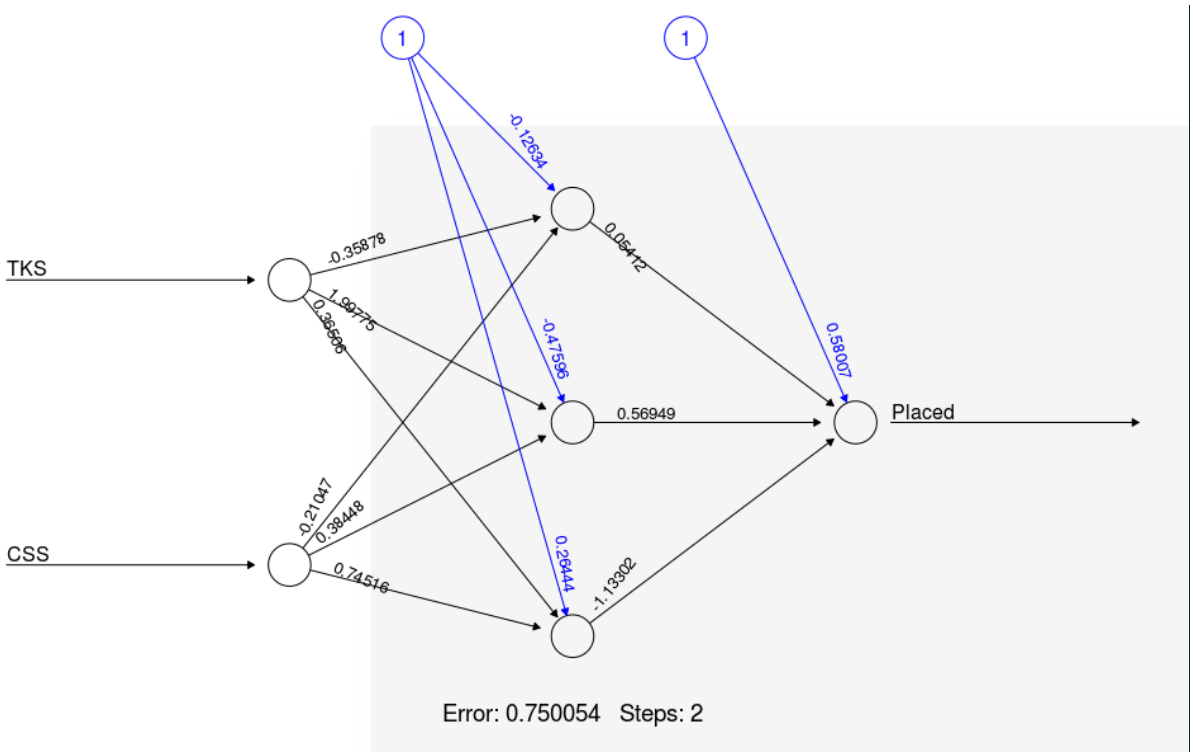
- **Pattern Recognition:** neural networks are very suitable for pattern recognition problems such as facial recognition, object detection, fingerprint recognition, etc.
- **Anomaly Detection:** neural networks are good at pattern detection and they can easily detect the unusual patterns that do not fit in general patterns.
- **Time Series Prediction:** Neural networks can be used to predict time series problems such as stock price, and weather forecasts, among others.
- **Natural Language Processing:** Neural networks offer a wide range of applications in Natural Language Processing (NLP) tasks such as text classification, Named Entity Recognition (NER), Part-of-Speech (POS) Tagging, speech recognition, and spell checking.

Using R to Understand the Structure of Neural Networks

Example 1

```
library(neuralnet)

# Creating training data set with TKS as technical knowledge
# score and CSS as communication skill score
TKS=c(20,10,30,20,80,30)
CSS=c(90,20,40,50,50,80)
Placed=c(1,0,0,0,1,1)

# Here, you will combine multiple columns or features into a
# Single set of data
df=data.frame(TKS,CSS,Placed)

# Fit neural network
nn=neuralnet(Placed~TKS+CSS,data=df, hidden=3,act.fct =
"logistic",linear.output = TRUE)

# Plot neural network
plot(nn)

# Creating test set
TKS=c(30,40,85)
CSS=c(85,50,40)
test=data.frame(TKS,CSS)

# Prediction using neural network
Predict=compute(nn,test)
Predict$net.result
          [,1]
[1,] 1.0336055
[2,] 0.4267056
[3,] 0.9069714

# Converting probabilities into binary classes setting threshold
# level 0.5
prob <- Predict$net.result
```

```
pred <- ifelse(prob>0.5, 1, 0)
pred
     [,1]
[1,]    1
[2,]    0
[3,]    1
```



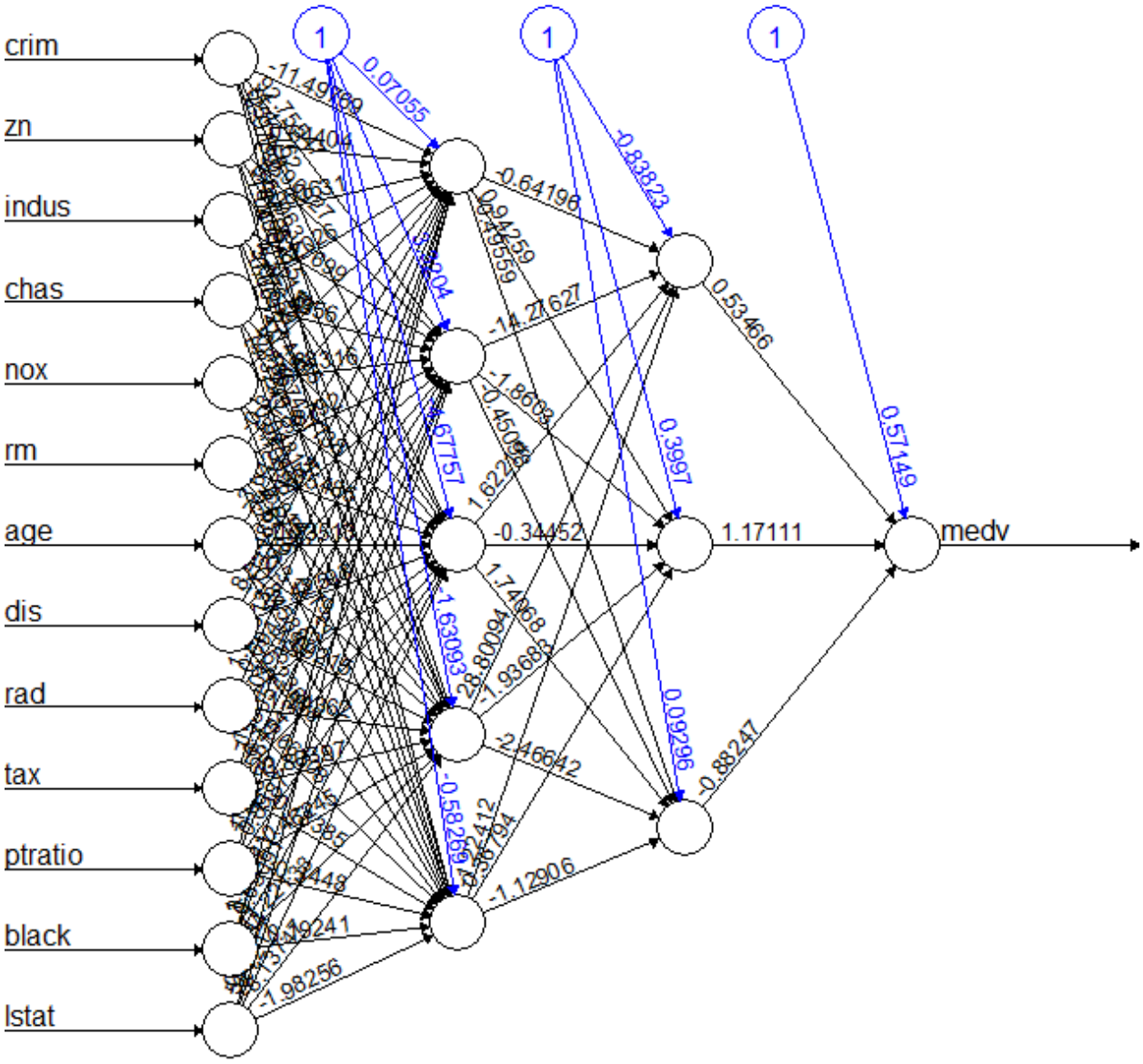
Example 2

Let us use the Boston dataset in the MASS package. The Boston dataset is a collection of data about housing values in the suburbs of Boston. The 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 for data problems
apply(data,2,function(x) sum(is.na(x)))
    crim       zn    indus     chas      nox       rm      age      dis      rad      tax
       0        0        0        0        0        0        0        0        0        0
ptratio    black    lstat     medv
       0        0        0        0
```

```
index <- sample(1:nrow(data),round(0.75*nrow(data)))
train <- data[index,]
test <- data[-index,]

# Use a Generalized Linear Model
lm.fit <- glm(medv~., data=train)
summary(lm.fit)
Call:
glm(formula = medv ~ ., data = train)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-15.2113   -2.5587   -0.6552    1.8275   29.7110

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  31.111702   5.459811   5.698 2.49e-08 ***
crim         -0.111372   0.033256  -3.349 0.000895 ***
zn            0.042633   0.014307   2.980 0.003077 **
indus         0.001483   0.067455   0.022 0.982473
chas          1.756844   0.981087   1.791 0.074166 .
nox         -18.184847   4.471572  -4.067 5.84e-05 ***
rm            4.760341   0.480472   9.908  < 2e-16 ***
age          -0.013439   0.014101  -0.953 0.341190
dis          -1.553748   0.218929  -7.097 6.65e-12 ***
rad           0.288181   0.072017   4.002 7.62e-05 ***
tax          -0.013739   0.004060  -3.384 0.000791 ***
ptratio      -0.947549   0.140120  -6.762 5.38e-11 ***
black         0.009502   0.002901   3.276 0.001154 **
lstat        -0.388902   0.059733  -6.511 2.47e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 20.23806)

    Null deviance: 32463.5  on 379  degrees of freedom
Residual deviance:  7407.1  on 366  degrees of freedom
AIC: 2237

Number of Fisher Scoring iterations: 2
```

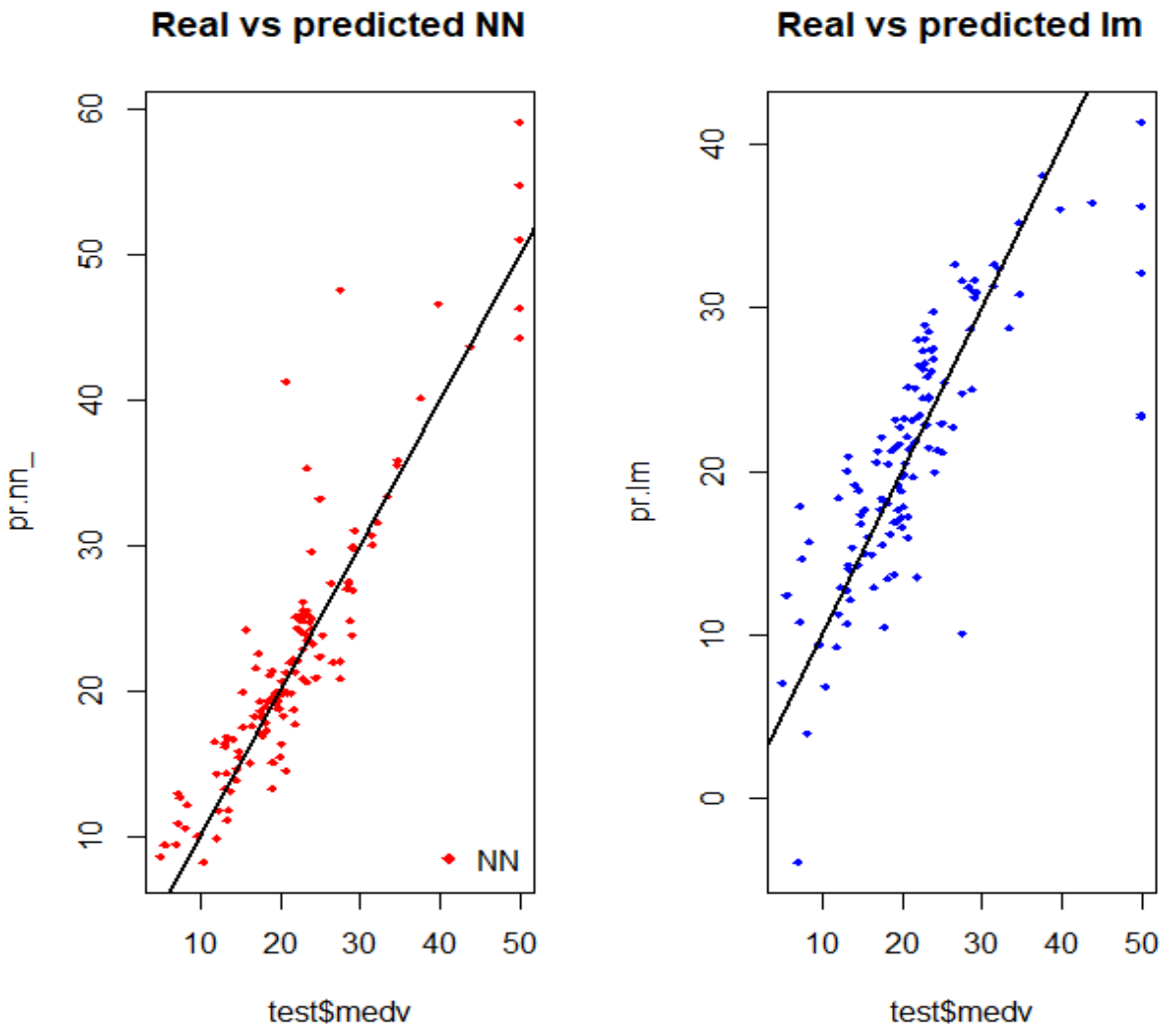
```
pr.lm <- predict(lm.fit,test)
MSE.lm <- sum((pr.lm - test$medv)^2)/nrow(test)
maxs <- apply(data, 2, max)
mins <- apply(data, 2, min)
scaled <- as.data.frame(scale(data, center = mins, scale = maxs
- mins))
train_ <- scaled[index,]
test_ <- scaled[-index,]

# Load the 'neuralnet' library
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)
plot(nn)
pr.nn <- compute(nn,test_[,1:13])
pr.nn_ <- pr.nn$net.result*(max(data$medv)-min(data$medv))+
min(data$medv)
test.r <- (test_$medv)*(max(data$medv)- min(data$medv))+
min(data$medv)
MSE.nn <- sum((test.r - pr.nn_)^2)/nrow(test_)
print(paste(MSE.lm,MSE.nn))
[1] "31.2630222372615 16.4595537665717"
```



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. The bias can be thought of as the intercept of a linear model. The net is essentially a black box so we cannot say that much about the fitting, the weights and the model. Suffice to say that the training algorithm has converged and therefore the model is ready to be used.

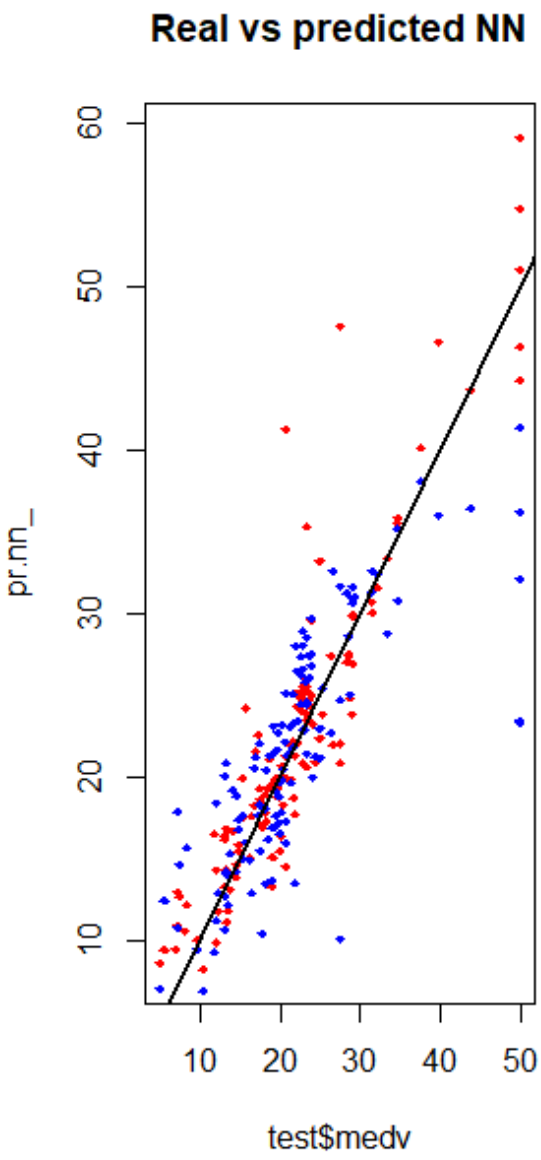
```
par(mfrow=c(1,2))
```

```
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, cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='LM',pch=18,col='blue', bty='n',
cex=.95)
```




```
plot(test$medv,pr.nn_,col='red',main='Real vs predicted
NN',pch=18,cex=0.7)
points(test$medv,pr.lm,col='blue',pch=18,cex=0.7)
abline(0,1,lwd=2)
```

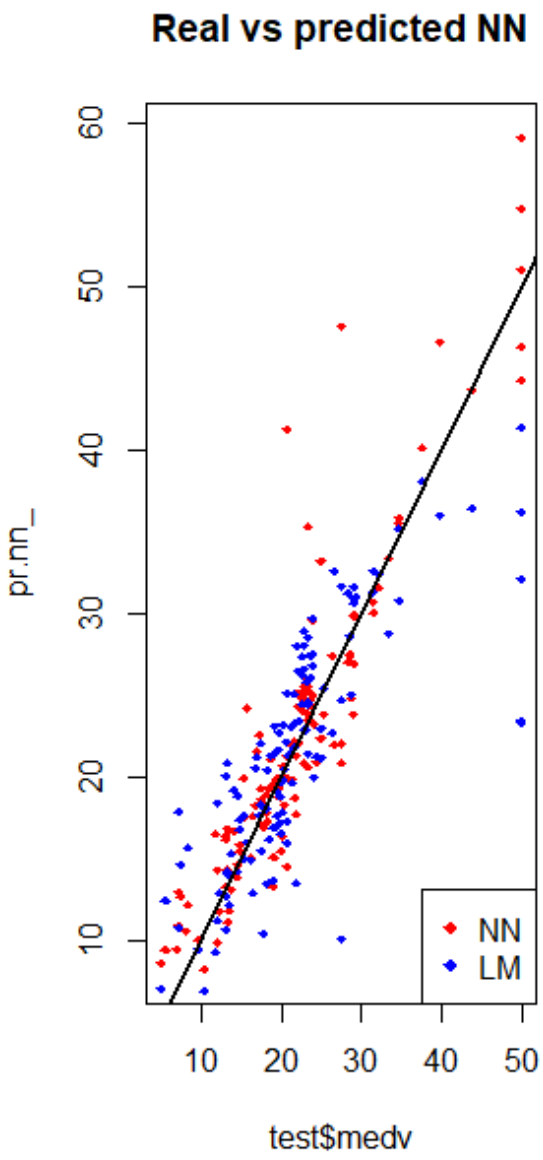
```
legend('bottomright',legend=c('NN','LM'),pch=18,col=c('red','blu
e'))
```

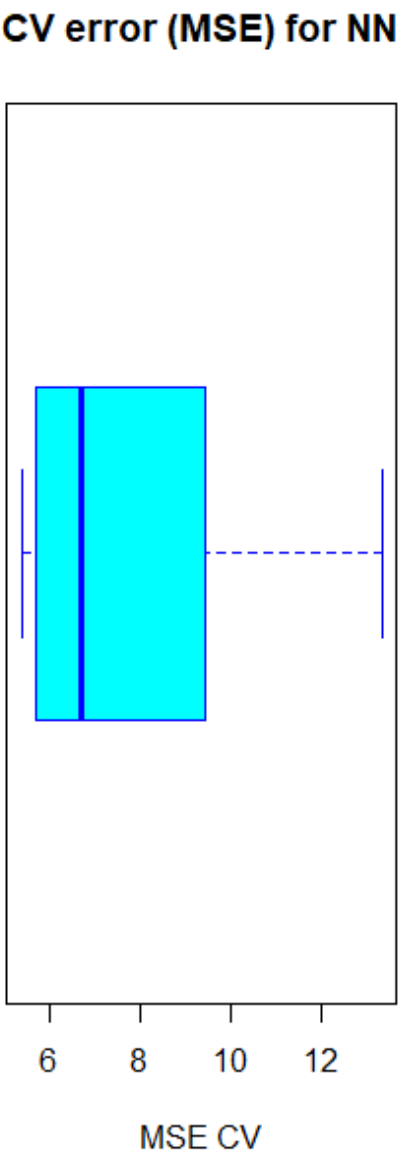



```
# Load the 'boot' library
library(boot)
set.seed(200)
lm.fit <- glm(medv~.,data=data)
cv.glm(data,lm.fit,K=10)$delta[1]
```

```
[1] 23.17094
```

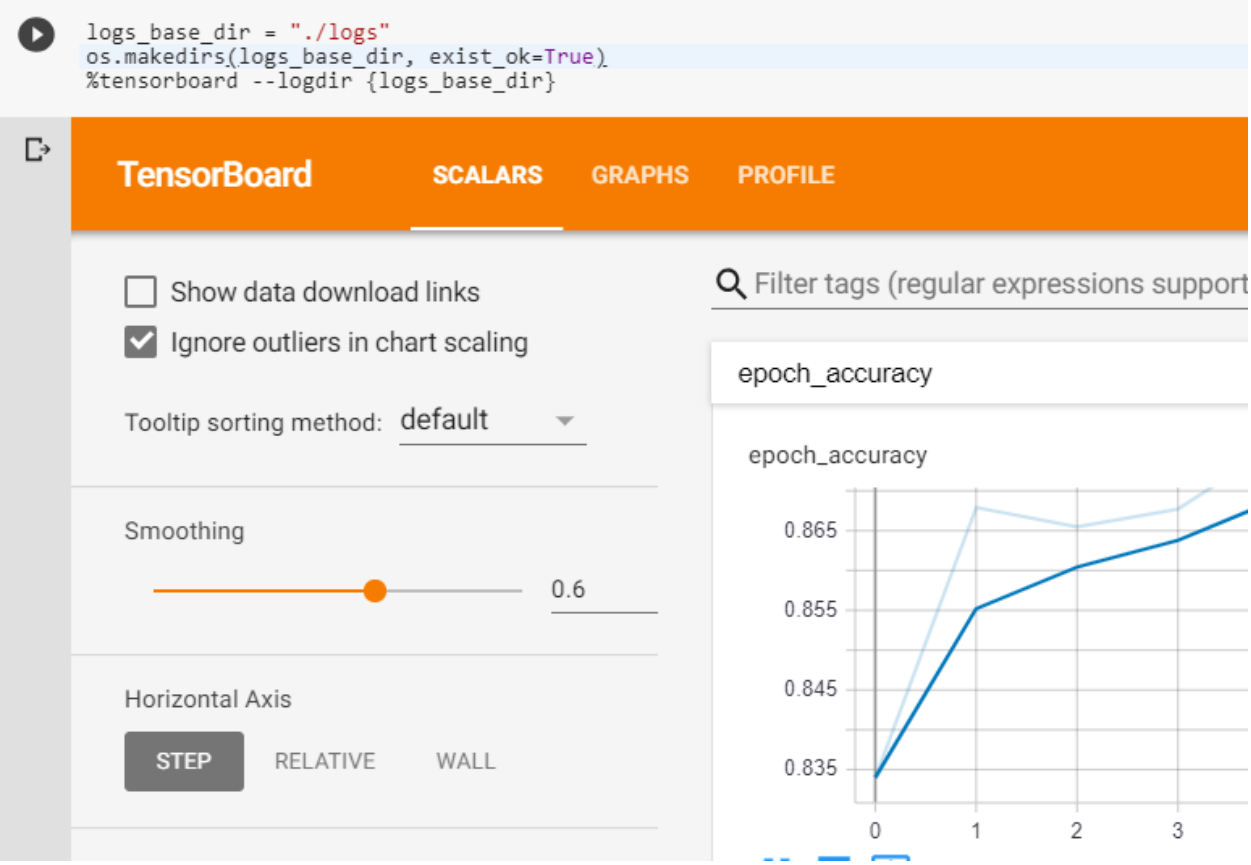
```
set.seed(450)
cv.error <- NULL
k <- 10

library(plyr)
pbar <- create_progress_bar('text')
pbar$init(k)
for(i in 1:k){
     index <- sample(1:nrow(data),round(0.9*nrow(data)))
     train.cv <- scaled[index,]
     test.cv <- scaled[-index,]
     nn<-neuralnet(f,data=train.cv,hidden=c(5,2),linear.output=)
     pr.nn <- compute(nn,test.cv[,1:13])
     pr.nn <- pr.nn$net.result*(max(data$medv)-min(data$medv))+
min(data$medv)
     test.cv.r <- (test.cv$medv)*(max(data$medv)-
     min(data$medv))+min(data$medv)
     cv.error[i] <- sum((test.cv.r - pr.nn)^2)/nrow(test.cv)
     pbar$step()
 }

mean(cv.error)
[1] 7.641292

cv.error
 [1] 13.331937  7.099840  6.580337  5.697609  6.841745  5.771481
10.751406  5.384253
 [9]  9.452109  5.502201
```

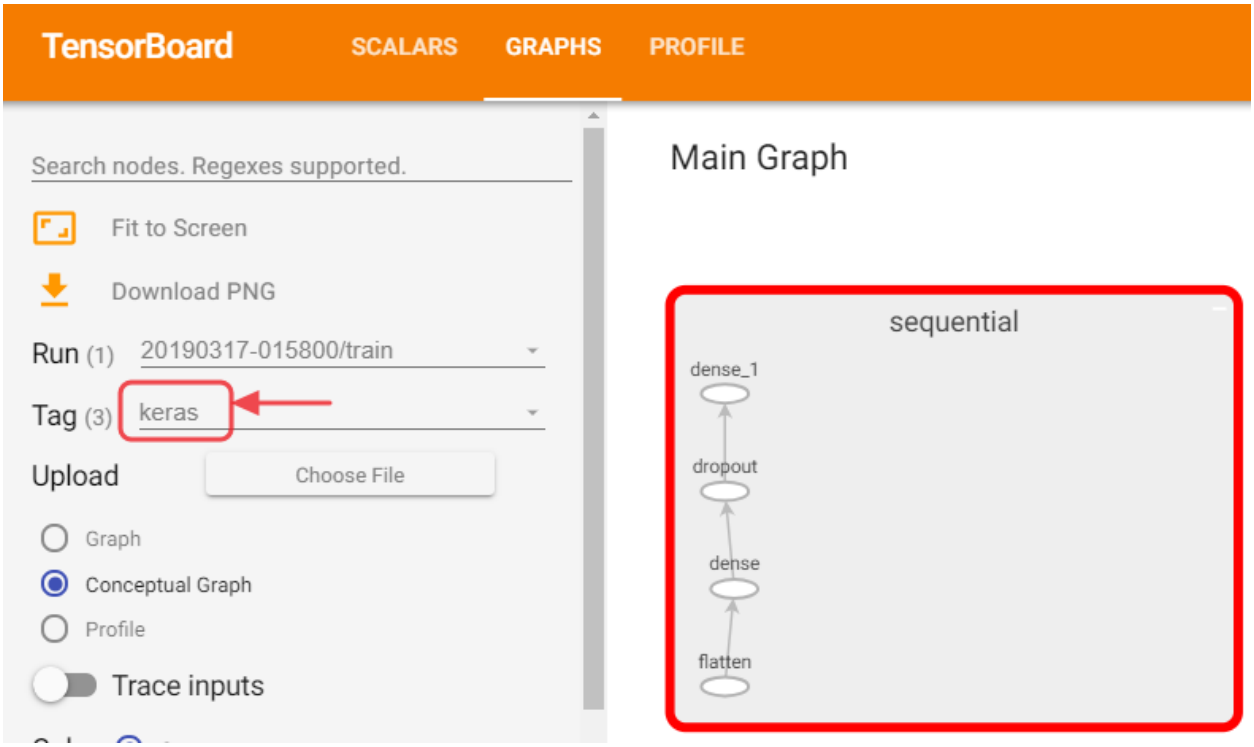

Real vs predicted NN
pr.nn_
test$medv
NN
LM
10
20
30
40
50
60


CV error (MSE) for NN
MSE CV
6
8
10
12

Visualization with Tensorflow 2

The tensorboard with Tensorflow 2 allows analysts to track the performance of the model.




TensorBoard
SCALARS
GRAPHS
PROFILE
Search nodes. Regexes supported.
Fit to Screen
Download PNG
Run (1)
20190317-015800/train
Tag (3)
keras
Upload
Choose File
Graph
Conceptual Graph
Profile
Trace inputs
Main Graph
sequential
dense_1
dropout
dense
flatten