|  |
| --- |
| title: “Neural Network & the Bakken- What goes on Under the Hood?” |
| author: “Marshal Wigwe” |
| output: word\_document |

# Introduction

We currently live in the world of “big data” and several operators are beginning to apply data science tools for optimization of oil and gas production, among other things. About 88% of the 170 Bbbl. most likely estimate of the OOIP in the Bakken and Three Fork are located in six of the nineteen producing counties in North Dakota. These counties are Burke (10.04%), Divide (10.46%), Dunn (11.87%), McKenzie (21.51%), Mountrail (17.10%) and Williams (17.11%). The dataset used in this analysis contains 5755 wells, distributed in these counties as shown in Table 1. These wells were completed between 2008 and 2016, with at least one year of production recorded. The use of neural network as a predictive tool is common practice and most times, we tend to treat this tool as a “blackbox”. We provide the “box” some input parameters, and it spills out a result. However, it is important we understand what goes on under the hood, to open the “blackbox”. In the example shown here, we will predict the first twelve month oil production using well completion and production variables. We would discuss simple models, but it is important to understand the our dataset.

## Analysis Routine: Data Import

* To get started with this analysis, direct your R session to a dedicated working directory which should contain the bakken dataset. Remember to convert date variables to date. This was imported by read.csv function as factors. We would use several packages available in R.
* [bakken](./bakken_data.csv)
* import data into R and Preprocess the data

bakken = read.csv("bakken\_data.csv")  
  
library(lubridate)

##   
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':  
##   
## date

bakken$completionDate = mdy(bakken$completionDate)  
bakken$firstProdDate = mdy(bakken$firstProdDate)  
  
# Table 1: Distribution of wells by County and Formation  
addmargins(table(bakken$County, bakken$targetFormation))

##   
## BAKKEN THREE FORKS Sum  
## BURKE 87 51 138  
## DIVIDE 119 186 305  
## DUNN 658 365 1023  
## MCKENZIE 1185 678 1863  
## MOUNTRAIL 850 396 1246  
## WILLIAMS 843 337 1180  
## Sum 3742 2013 5755

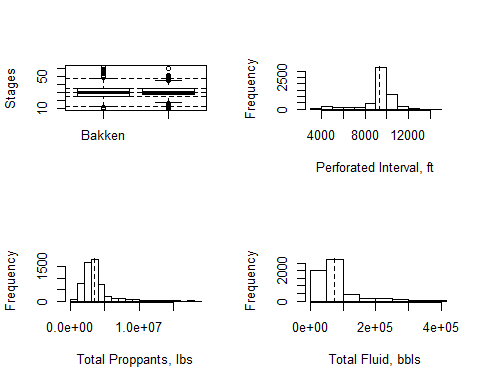
### Distribution of completion parameters

* Let’s take a look at the distribution of the number of stages, total pounds of proppant, total volume of fluid injected and the perforated interval typically used in frac jobs in the Bakken and Three Forks formations (fig. 1). The comparative boxplot shows the number of stages. We can observe that on average, operators are using the same application in both formations for the number of stages (30 stages). There is more variability in the bakken compared to the Three Forks. For the perforated interval, the histogram shows that most operators favor a perforated interval in the 8,000 ft. to 11,000 ft. range on the lateral. The distribution of total pounds of proppant used for the frac jobs is shown in the boxplot. Most frac jobs used less than 5 million pounds of total proppants.As we shall see later, of the 657 occurrences of application of more than 5 million pounds of total proppants, only 83 cases occurred prior to 2014 (Fig. 2). This indicates that the use of large pounds of proppants started becoming popular during the downturn. On average, 75,000 bbls of fluid and 3.5 million pounds of proppants were used for the 30 stage completion of a 9,300 ft. perforated interval between 2008 and 2016.

par(mfrow = c(2, 2))  
# Fig. 1a - Comparative boxplot for the number of frac stages between target formations  
with(bakken, boxplot(jitter(stage) ~ targetFormation, ylab = "Stages", names = c("Bakken", "Three Forks")))  
stats = c(13, 26, 30, 35, 48) # lower wisker, 1st, 2nd & 3rd qartiles and upper wisker  
abline(h = stats, lty = 2)  
  
# Fig. 1b: Distribution of Perforated Interval for all Wells  
with(bakken, hist(perfInterval, main = "", xlab = "Perforated Interval, ft"))  
abline(v = mean(bakken$perfInterval), lty = 2 ) # average = 9,317 ft  
  
# Fig. 1c: Distribution of total proppant for all Wells  
with(bakken, hist(totalProp, main = "", xlab = "Total Proppants, lbs"))  
abline(v = mean(bakken$totalProp), lty = 2 ) # average = 3,511,481 lbs  
sum(bakken$totalProp > 5000000) # 656 wells with TP > 5MM lbs

## [1] 656

# Fig. 1d: Distribution of total fluid for all Wells  
with(bakken, hist(totalFluid, main = "", xlim = c(0, 400000), xlab = "Total Fluid, bbls"))  
abline(v = mean(bakken$totalFluid), lty = 2)



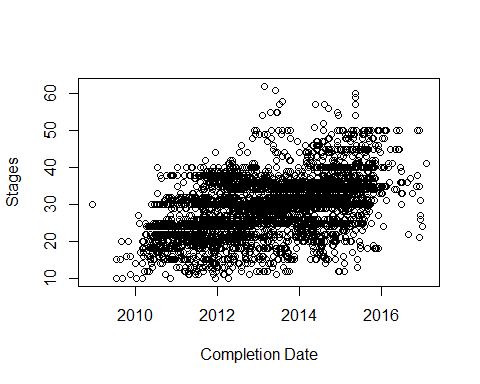
c(sum(bakken$totalFluid > 50000), sum(bakken$totalFluid > 100000), sum(bakken$totalFluid > 200000)) # 3738 949 308

## [1] 3738 949 308

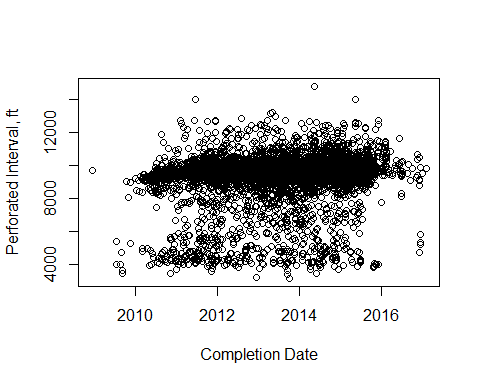
### How has completion parameters changed since 2008

* Variation of completion parameters with time from 2008 - 2016. Fig. 2a seems to suggest an increasing trend in the number of stages used in completions between this period. There does not appear to be a systematic change in the length of lateral and perforated interval since 2008 (Fig. 2b). However, we see an increasing tendency towards perforating the lateral in the 9,000 ft. - 11,000 ft. range. As mentioned previously, the use of more than 5MM lbs of proppants started becoming popular after 2014. Most of the completions prior to 2012 utilized less than 100,000 bbls of total fluid and like the case of total proppants, the use of more than 100,000 bbls of total fluid became increasingly popular from 2013 and well into the downturn (Fig. 2d). This tendency to use more proppants and fluid volume may have led operators to complete fewer wells as a way of cutting overall cost during the downturn (fig. 2e).

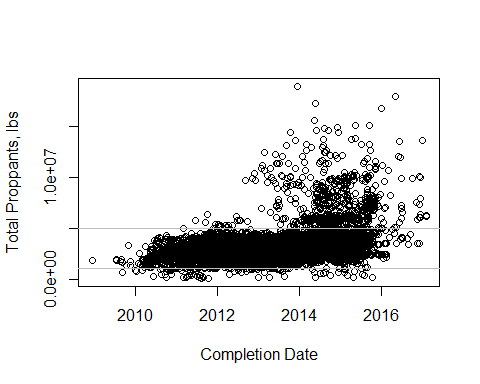
# fig. 2a -Variation of Frac stages used in completions with time  
with(bakken, plot(completionDate, stage, xlab = "Completion Date", ylab = "Stages"))



# Fig. 2b - Variations in the Perforated Intervals Used by Operators for Completion  
with(bakken, plot(completionDate, perfInterval, xlab = "Completion Date", ylab = "Perforated Interval, ft"))



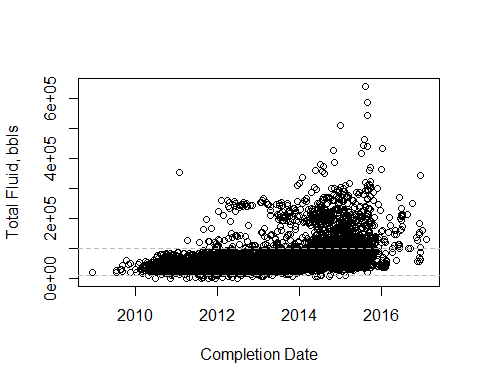
# Fig. 2c - Variations in total pounds of proppants Used by Operators for Completion  
with(bakken, plot(completionDate, totalProp, xlab = "Completion Date", ylab = "Total Proppants, lbs"))  
abline(h = c(1000000, 5000000), col = "grey")



sum((bakken$totalProp > 5000000)& (year(bakken$completionDate) > 2013))

## [1] 574

# Fig. 2d - Variations in total fluid Used by Operators for Completion  
with(bakken, plot(completionDate, totalFluid, xlab = "Completion Date", ylab = "Total Fluid, bbls"))  
abline(h = c(100000, 10000), lty = 2, col = "grey")



sum((bakken$totalFluid > 100000)& (year(bakken$completionDate) > 2013))

## [1] 780

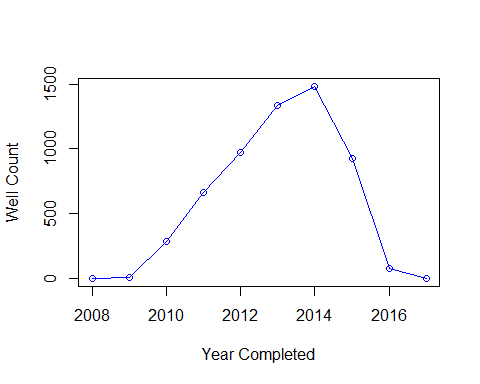
sum((bakken$totalFluid > 20000)&(bakken$totalFluid < 200000)) # 92% of wells in this range.

## [1] 5311

### Looking at other well parameters

* Aggregate the number of wells completed by year.

bakken$wellcount = 1  
a<-aggregate(wellcount ~ year(completionDate), bakken, sum)  
# Fig. 2e: Variation in well count with Year Completed  
plot(a$`year(completionDate)`, a$wellcount, type='o', col='blue', xlab = "Year Completed", ylab = "Well Count")

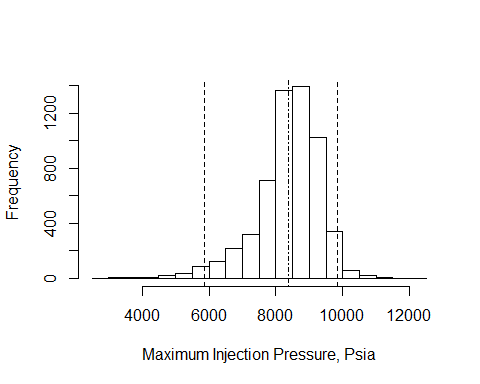


* Looking at injection rate and pressure

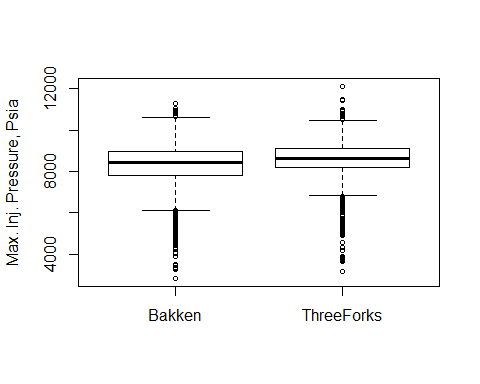
# Fig. 3a: Distribution of Maximum Injection Pressure  
with(bakken, hist(maxInjPres, main = "", xlab = "Maximum Injection Pressure, Psia"))  
c(min(bakken$maxInjPres), mean(bakken$maxInjPres), max(bakken$maxInjPres)) # 2836, 8366, 12112

## [1] 2836.000 8366.275 12112.000

ucl = quantile(bakken$maxInjPres, 0.975); lcl = quantile(bakken$maxInjPres, 0.025)  
abline(v = c(ucl, lcl, mean(bakken$maxInjPres)), lty = c(2, 2, 4))



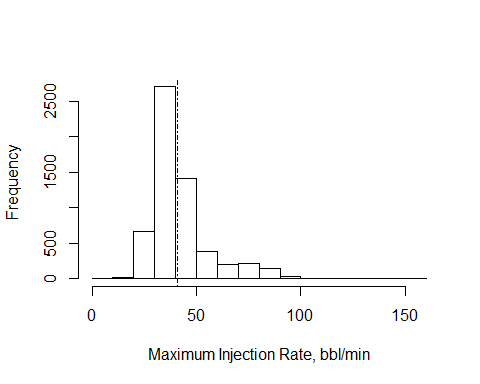
# Fig. 3b: Distribution of Maximum Injection Pressure by Formation  
with(bakken, boxplot(maxInjPres ~ targetFormation, ylab = "Max. Inj. Pressure, Psia", cex = 0.7, names = c("Bakken", "ThreeForks")))



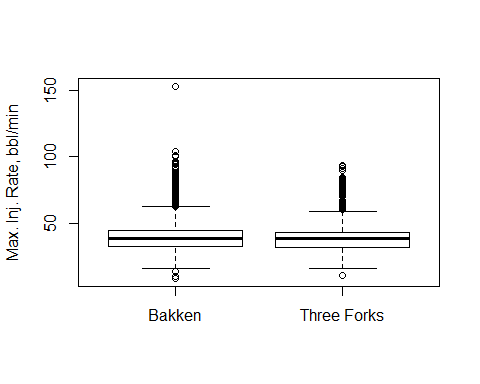
# Fig. 3c: Distribution of Maximum Injection Rate  
with(bakken, hist(maxInjRate, main = "", xlab = "Maximum Injection Rate, bbl/min"))  
c(min(bakken$maxInjRate), mean(bakken$maxInjRate), max(bakken$maxInjRate)) # 8.1, 41, 153.4 ucl = quantile(bakken$maxInjRate, 0.975); lcl = quantile(bakken$maxInjRate, 0.025)

## [1] 8.10000 41.00137 153.40000

abline(v = c(ucl, lcl, mean(bakken$maxInjRate)), lty = c(2, 2, 4))

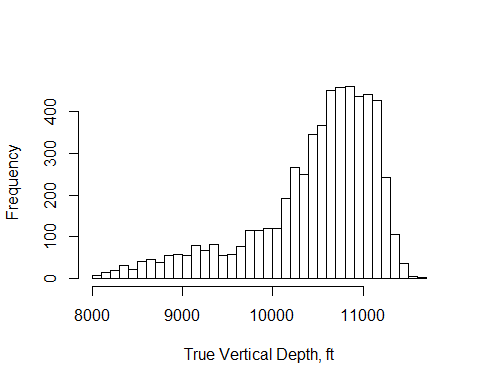


# Fig. 3d: Distribution of Maximum Injection Rate by Formation  
with(bakken, boxplot(maxInjRate ~ targetFormation, ylab = "Max. Inj. Rate, bbl/min", names = c("Bakken", "Three Forks")))

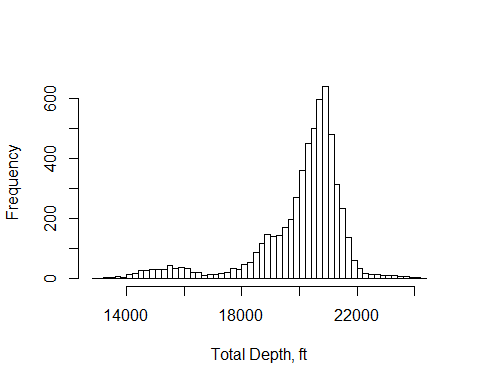


* Looking at TVD, TD, Peak Oil and 12 month oil

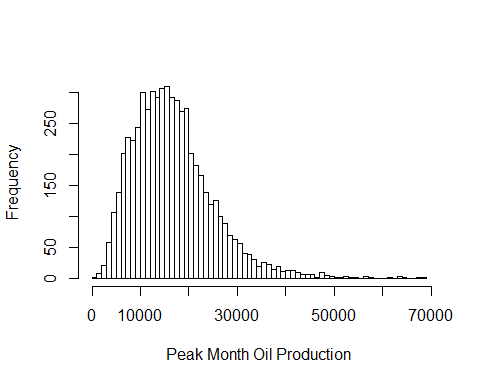
# Fig. 4a: Distribution of True Vertical Depth, ft  
with(bakken, hist(TVD, main = "", breaks = 50, xlab = "True Vertical Depth, ft"))



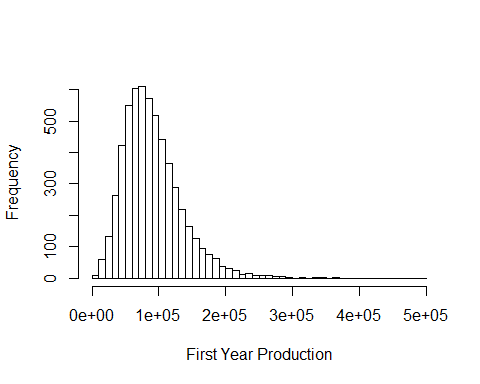
# Fig. 4b: Distribution of Total Depth, ft  
with(bakken, hist(totalDepth, main = "", breaks = 50, xlab = "Total Depth, ft"))



# Fig. 4c: Distribution of Peak Month Oil Production  
with(bakken, hist(peakOil, main = "", breaks = 50, xlab = "Peak Month Oil Production"))



# Fig. 4d: Distribution of First Year Cumulative Oil Production  
with(bakken, hist(First12Oil, main = "", breaks = 50, xlab = "First Year Production"))



mean(bakken$First12Oil) # 91,441 bbls

## [1] 91441.25

## Application of Neural Network

With the on-going growth in the world of “big data”, several data science techniques are currently being used to evaluate the performance of oil and gas wells. Among commonly used techniques are linear and multiple regression, multivariate analysis, decision trees, and the family of AI, which includes neural network, deep learning, etc. Neural network regression model belongs to a class of functions called universal approximators, just like polynomial and Fourier functions. These functions can come really close in the estimation of the true mean function if correctly applied. NN is used to estimate the conditional mean functions that are highly non-linear. Hence, NN regression aims to approximate the true (unknown) regression function , using a general non-linear function, say . NN regression uses the logistic function as activation functions. Iterative techniques that are similar to those used to obtain maximum likelihood, are used to estimate the parameters of the model.

### Prediction of Well Performance using Neural Network - One Node

* Application of Neural Network to model First Year Production using one hidden layer with one node. NN uses numeric data for analysis, hence there is the need to subset our data to extract the variables that are important to our analysis. These variables are measured in different units, as a result, it is good practice to standardize these. We would need to re-scale the final result back to the original scale.

data = bakken[, c("First12Oil", "totalDepth", "TVD", "perfInterval", "stage", "maxInjRate",   
 "maxInjPres", "totalProp", "totalFluid", "peakOil")]  
  
library(neuralnet)   
set.seed(12345)  
max\_data <- apply(data, 2, max); min\_data <- apply(data, 2, min)  
data\_scaled <- scale(data, center = min\_data, scale = max\_data - min\_data)   
# Sampling from scaled data  
n\_train = 0.7 # Using 70-30 rule  
idx = sample(1:nrow(data), round(n\_train\*nrow(data)))  
train\_data <- as.data.frame(data\_scaled[idx,]) # for training the model  
test\_data <- as.data.frame(data\_scaled[-idx,]) # for testing the model  
  
# Build model using Neural Network on train data  
n = names(train\_data)  
y = "First12Oil" # Variable of interest. Should be character  
f <- as.formula(paste(y, "~", paste(n[!n %in% y], collapse = " + ")))  
fit\_net = neuralnet(f, data = train\_data, hidden = 1)  
plot(fit\_net)  
  
# Test/Validate the model in two ways  
# 1- by the compute method in R  
predicted\_data <- compute(fit\_net, test\_data[, !n %in% y])  
pred\_data = predicted\_data$net.result\*(max\_data[y] - min\_data[y]) + min\_data[y]

* Now let’s compute the predicted values “by hand”:

# 2- by opening the "blackbox"  
b10 = fit\_net$weights[[1]][[1]][1,1]; b11 = fit\_net$weights[[1]][[1]][2,1]  
b12 = fit\_net$weights[[1]][[1]][3,1]; b13 = fit\_net$weights[[1]][[1]][4,1]  
b14 = fit\_net$weights[[1]][[1]][5,1]; b15 = fit\_net$weights[[1]][[1]][6,1]  
b16 = fit\_net$weights[[1]][[1]][7,1]; b17 = fit\_net$weights[[1]][[1]][8,1]  
b18 = fit\_net$weights[[1]][[1]][9,1]; b19 = fit\_net$weights[[1]][[1]][10,1]  
  
# Linear functions  
h1 = b10 + b11\*test\_data$totalDepth + b12\*test\_data$TVD + b13\*test\_data$perfInterval + b14\*test\_data$stage + b15\*test\_data$maxInjRate + b16\*test\_data$maxInjPres + b17\*test\_data$totalProp + b18\*test\_data$totalFluid + b19\*test\_data$peakOil   
#Nodes  
N1 = 1/(1 + exp(-h1))  
  
# Node weights  
g0 = fit\_net$weights[[1]][[2]][1,1]  
g1 = fit\_net$weights[[1]][[2]][2,1]  
  
#Final prediction  
g\_y\_x = g0 + g1\*N1   
g\_y\_x = g\_y\_x\*(max\_data[y] - min\_data[y]) + min\_data[y]  
  
# Comparing the prediction using compute and "by nand" yield similar result  
sum ( (pred\_data - g\_y\_x)^2 ) # Should be approx. zero within computer error.

## [1] 0

### Mathematical equations

Depending on the number of neurons/nodes in each hidden layer, we have as many logistic functions added together. To understand what goes on under the hood, from the plot of the model shown above for one hidden layer and one node, the NN uses one logistic function . With two nodes, we add a second logistic function and so on. here, is a linear function of the nine variables in the model.

The result of the analysis above can be summarized in equation form as follows:

Putting the NN model in equation form makes the picture shown in the plot clearer. The values shown on the connecting lines and to the left of the node are the coefficients/weights () of the variables, while the values shown to the right of the node are the

* With two and three nodes, we add second and third logistic functions as shown below for three nodes. The linear functions , and are different functions.

### Evaluation of Well Performance using Neural Network - three nodes.

set.seed(12345)  
max\_data <- apply(data, 2, max)   
min\_data <- apply(data, 2, min)  
data\_scaled <- scale(data, center = min\_data, scale = max\_data - min\_data)   
# Sampling from scaled data  
n\_train = 0.7 # Using 70-30 rule  
idx = sample(1:nrow(data), round(n\_train\*nrow(data)))  
train\_data <- as.data.frame(data\_scaled[idx,]) # for training the model  
test\_data <- as.data.frame(data\_scaled[-idx,]) # for testing the model  
  
# Build model using Neural Network on train data  
n = names(train\_data)  
y = "First12Oil" # Variable of interest. Should be character  
f <- as.formula(paste(y, "~", paste(n[!n %in% y], collapse = " + ")))  
fit\_net = neuralnet(f, data = train\_data, hidden = 3)  
plot(fit\_net)

# Test/Validate the model in two ways  
# 1- by the compute method in R  
predicted\_data <- compute(fit\_net, test\_data[, !n %in% y])  
pred\_data = predicted\_data$net.result\*(max\_data[y] - min\_data[y]) + min\_data[y]  
  
# 2- by opening the "blackbox"  
b10 = fit\_net$weights[[1]][[1]][1,1]; b11 = fit\_net$weights[[1]][[1]][2,1]  
b12 = fit\_net$weights[[1]][[1]][3,1]; b13 = fit\_net$weights[[1]][[1]][4,1]  
b14 = fit\_net$weights[[1]][[1]][5,1]; b15 = fit\_net$weights[[1]][[1]][6,1]  
b16 = fit\_net$weights[[1]][[1]][7,1]; b17 = fit\_net$weights[[1]][[1]][8,1]  
b18 = fit\_net$weights[[1]][[1]][9,1]; b19 = fit\_net$weights[[1]][[1]][10,1]  
  
b20 = fit\_net$weights[[1]][[1]][1,2]; b21 = fit\_net$weights[[1]][[1]][2,2]  
b22 = fit\_net$weights[[1]][[1]][3,2]; b23 = fit\_net$weights[[1]][[1]][4,2]  
b24 = fit\_net$weights[[1]][[1]][5,2]; b25 = fit\_net$weights[[1]][[1]][6,2]  
b26 = fit\_net$weights[[1]][[1]][7,2]; b27 = fit\_net$weights[[1]][[1]][8,2]  
b28 = fit\_net$weights[[1]][[1]][9,2]; b29 = fit\_net$weights[[1]][[1]][10,2]  
  
b30 = fit\_net$weights[[1]][[1]][1,3]; b31 = fit\_net$weights[[1]][[1]][2,3]  
b32 = fit\_net$weights[[1]][[1]][3,3]; b33 = fit\_net$weights[[1]][[1]][4,3]  
b34 = fit\_net$weights[[1]][[1]][5,3]; b35 = fit\_net$weights[[1]][[1]][6,3]  
b36 = fit\_net$weights[[1]][[1]][7,3]; b37 = fit\_net$weights[[1]][[1]][8,3]  
b38 = fit\_net$weights[[1]][[1]][9,3]; b39 = fit\_net$weights[[1]][[1]][10,3]  
  
# Linear functions  
h1 = b10 + b11\*test\_data$totalDepth + b12\*test\_data$TVD + b13\*test\_data$perfInterval + b14\*test\_data$stage + b15\*test\_data$maxInjRate + b16\*test\_data$maxInjPres + b17\*test\_data$totalProp + b18\*test\_data$totalFluid + b19\*test\_data$peakOil   
h2 = b20 + b21\*test\_data$totalDepth + b22\*test\_data$TVD + b23\*test\_data$perfInterval + b24\*test\_data$stage + b25\*test\_data$maxInjRate + b26\*test\_data$maxInjPres + b27\*test\_data$totalProp + b28\*test\_data$totalFluid + b29\*test\_data$peakOil  
  
h3 = b30 + b31\*test\_data$totalDepth + b32\*test\_data$TVD + b33\*test\_data$perfInterval + b34\*test\_data$stage + b35\*test\_data$maxInjRate + b36\*test\_data$maxInjPres + b37\*test\_data$totalProp + b38\*test\_data$totalFluid + b39\*test\_data$peakOil  
  
#Nodes  
N1 = 1/(1 + exp(-h1))  
N2 = 1/(1 + exp(-h2))  
N3 = 1/(1 + exp(-h3))  
  
# Node weights  
g0 = fit\_net$weights[[1]][[2]][1,1]  
g1 = fit\_net$weights[[1]][[2]][2,1]  
g2 = fit\_net$weights[[1]][[2]][3,1]  
g3 = fit\_net$weights[[1]][[2]][4,1]  
  
#Final prediction  
g\_y\_x = g0 + g1\*N1 + g2\*N2 + g3\*N3  
g\_y\_x = g\_y\_x\*(max\_data[y] - min\_data[y]) + min\_data[y]  
  
# Comparing the prediction using compute and "by nand" yield similar result  
sum ( (pred\_data - g\_y\_x)^2 )# Should be approx. zero within computer error.

## [1] 0

### Put the Solution in Mathematical form