NEURAL NETWORKS

As an introduction, we'll tackle a prediction task with a continuous variable. We'll reproduce research from the field of cement and concrete manufacturing that seeks to model the compressive strength of concrete using it's age and content. This dataset was donated to the UCI Machine Learning Repository by Professor I-Cheng Yeh from Chung-Hua University in Taiwan. This tutorial was adapted from that of Brett Lantz in Machine Learning with R (Packt Publishing 2015).

The following predictor variables, all measured in kg per cubic meter of mix (aside from age which is measured in days), are used to model the compressive strength of concrete as measured in MPa:

- Cement
- Blast Furnace Slag
- Fly Ash
- Water
- Superplasticizer
- Coarse Aggregate
- Fine Aggregate
- Age

This dataset has 1030 observations. All of the variables are numeric, taking positive values. The scales will vary quite a bit since some ingredients are more prevalent than others.

```
> load("concrete.Rdata")
> str(concrete)
```

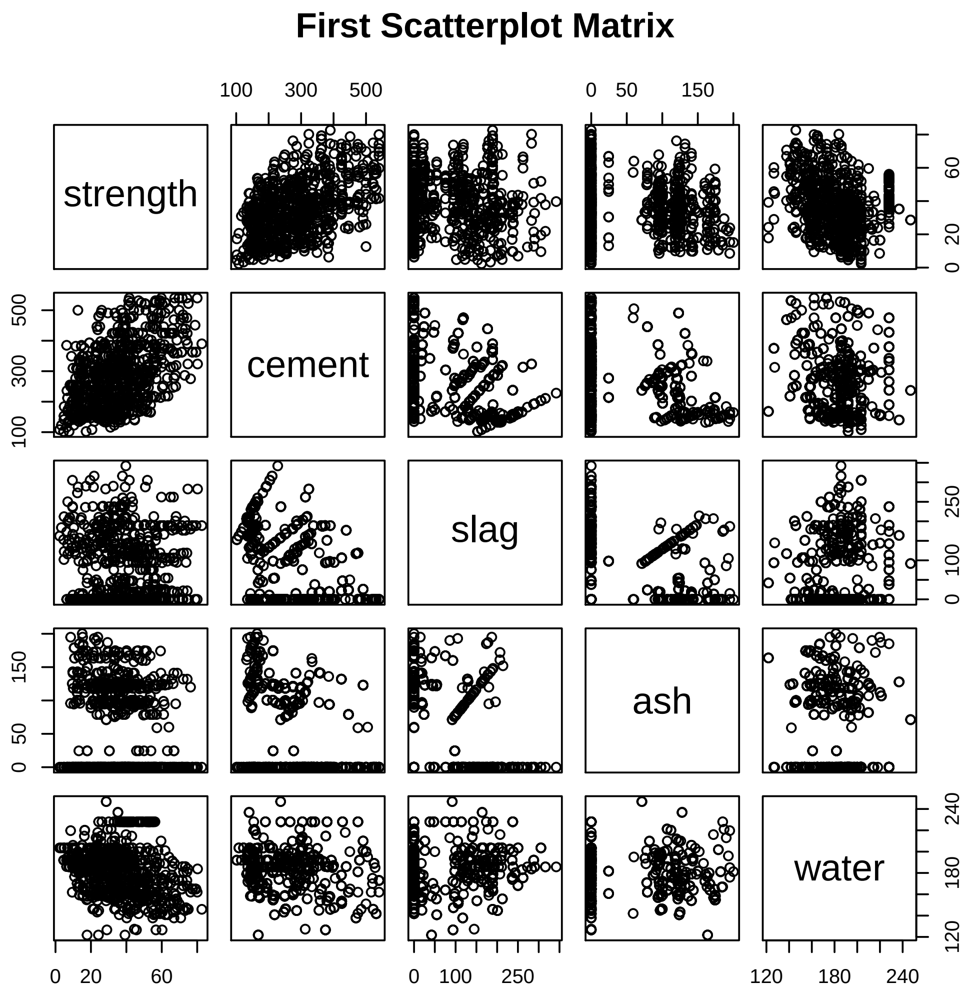
```
'data.frame':        1030 obs. of  9 variables:
 $ cement      : num  141 169 250 266 155 ...
 $ slag        : num  212 42.2 0 114 183.4 ...
 $ ash         : num  0 124.3 95.7 0 0 ...
 $ water       : num  204 158 187 228 193 ...
 $ superplastic: num  0 10.8 5.5 0 9.1 0 0 6.4 0 9 ...
 $ coarseagg   : num  972 1081 957 932 1047 ...
 $ fineagg     : num  748 796 861 670 697 ...
 $ age         : int  28 14 28 28 28 90 7 56 28 28 ...
 $ strength    : num  29.9 23.5 29.2 45.9 18.3 ...
```

Let's take an 80-20 split of the data for training and validation and then explore some preliminary relationships with the target variable.

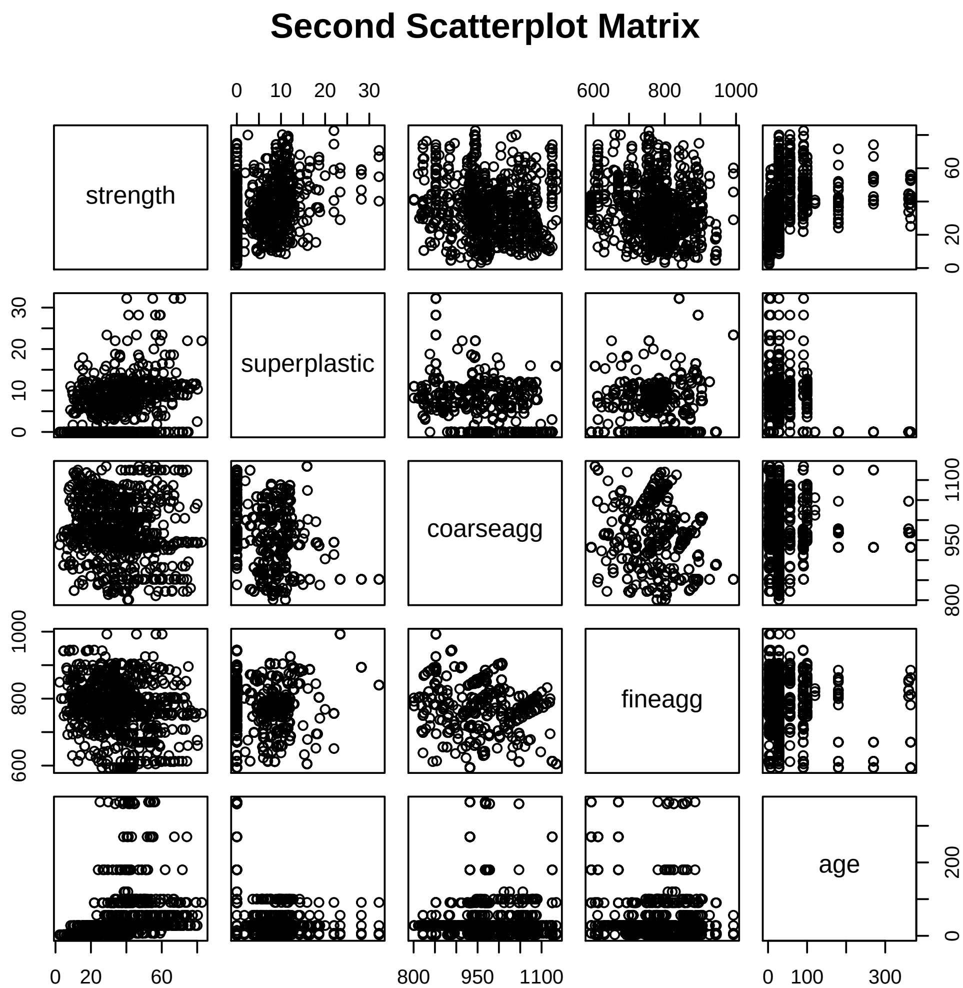
```
> TrainInd = sample(c(T,F),size=1030, replace=T,prob=c(0.8,0.2))
> train=concrete[TrainInd,]
> valid=concrete[!TrainInd,]
```

First let's check out a couple scatter plot matrices containing the target.

```
> pairs(strength~cement+slag+ash+water,data=train,
+    main="First Scatterplot Matrix")
```




```
> pairs(strength~superplastic+coarseagg+fineagg+age,data=train,
+    main="Second Scatterplot Matrix")
```



There certainly seem to be some relationships between input variables, and some variables appear to have a relationship with the target, but overall it is a little difficult to tell whether or not we will be able to make a good model for the compressive strength using these inputs. Let's try a simple linear model with all of the input variables and see how it performs.

```
> linear = lm(strength~cement+slag+ash+water+superplastic+coarseagg+fineagg+age,data=train)
> summary(linear)
```

```
Call:
lm(formula = strength ~ cement + slag + ash + water + superplastic +
    coarseagg + fineagg + age, data = train)

Residuals:
    Min      1Q  Median      3Q     Max
-28.724  -6.266   0.600   6.588  32.882

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  -2.945291  29.675529  -0.099   0.9210
cement        0.115508   0.009437  12.239  < 2e-16 ***
slag          0.100233   0.011394   8.797  < 2e-16 ***
ash           0.086760   0.013987   6.203 8.80e-10 ***
```

```
water        -0.178346   0.045034  -3.960 8.14e-05 ***
superplastic  0.263093   0.104180   2.525   0.0117 *
coarseagg     0.011785   0.010465   1.126   0.2604
fineagg       0.010877   0.011956   0.910   0.3632
age           0.106462   0.005740  18.549  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.38 on 818 degrees of freedom
Multiple R-squared:  0.6149,       Adjusted R-squared:  0.6111
F-statistic: 163.3 on 8 and 818 DF,  p-value: < 2.2e-16
```

Most of the variables appear to have a significant (linear) relationship with compressive strength, but the overall performance of the model leaves much to be desired (adjusted R^2 0.6). Let's check the MAPE on our validation data.

```
> linearPred = predict(linear, valid)
> linearRsq = cor(linearPred,valid$strength)
> linearMAPE = mean((linearPred-valid$strength)/valid$strength)
> cat("linear regression R-squared:", linearRsq)
```

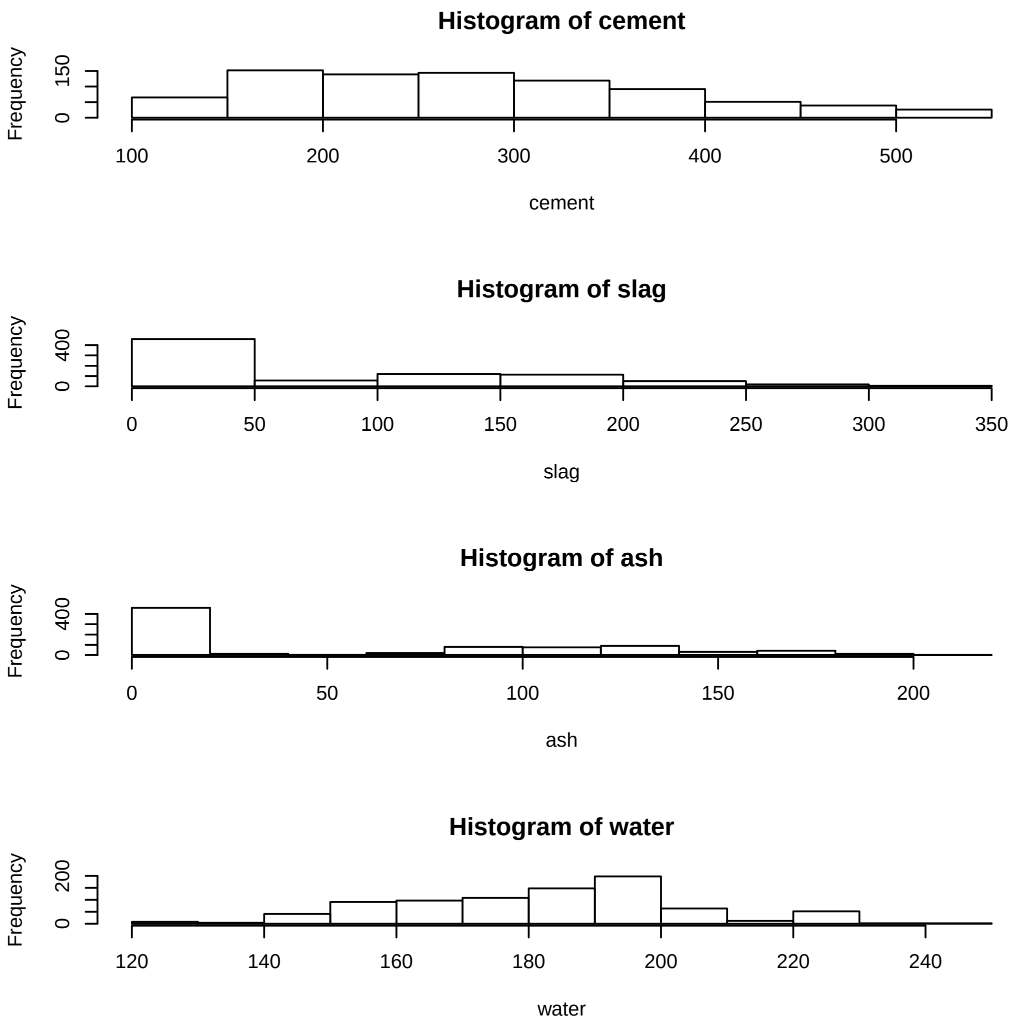
```
linear regression R-squared: 0.7820822
```

```
> cat("linear regression MAPE:",linearMAPE)
```

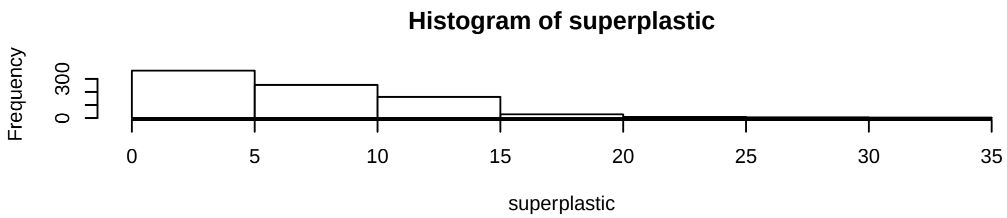
```
linear regression MAPE: 0.1092112
```

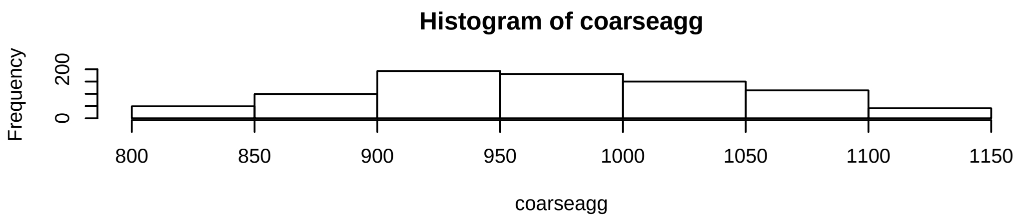
We probably desire a model with less MAPE than 30% if we're predicting strength of concrete to build bridges! Let's see if we can improve upon this MAPE using Neural Networks. For this tutorial, we'll use the neuralnet package. If you recall, neural networks work best when the input data is standardized to a narrow range near zero. Since our data takes positive values up to 1000, we'll standardize the data. Let's first check the distribution of the variables to see if statistical standardization seems appropriate.

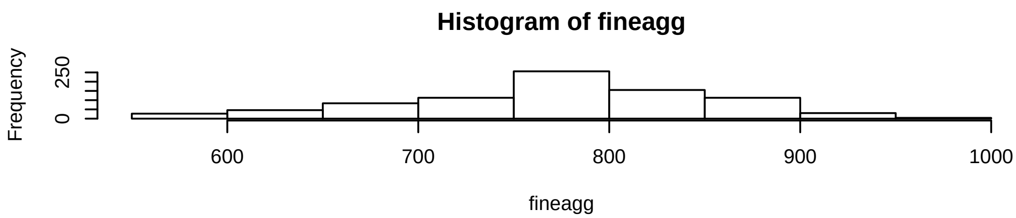
```
> # 3 figures arranged in 3 rows and 1 column
> attach(train)
> par(mfrow=c(4,1))
> hist(cement)
> hist(slag)
> hist(ash)
> hist(water)
```

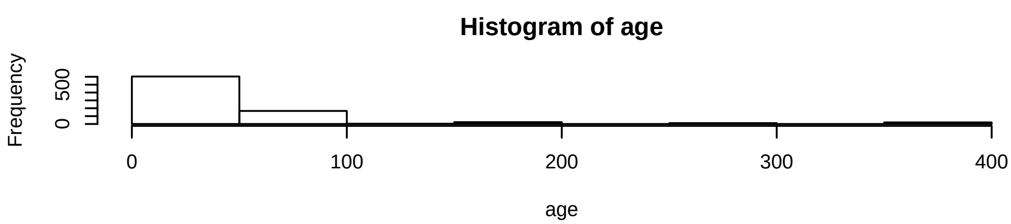



```
> par(mfrow=c(4,1))
> hist(superplastic)
> hist(coarseagg)
> hist(fineagg)
> hist(age)
```







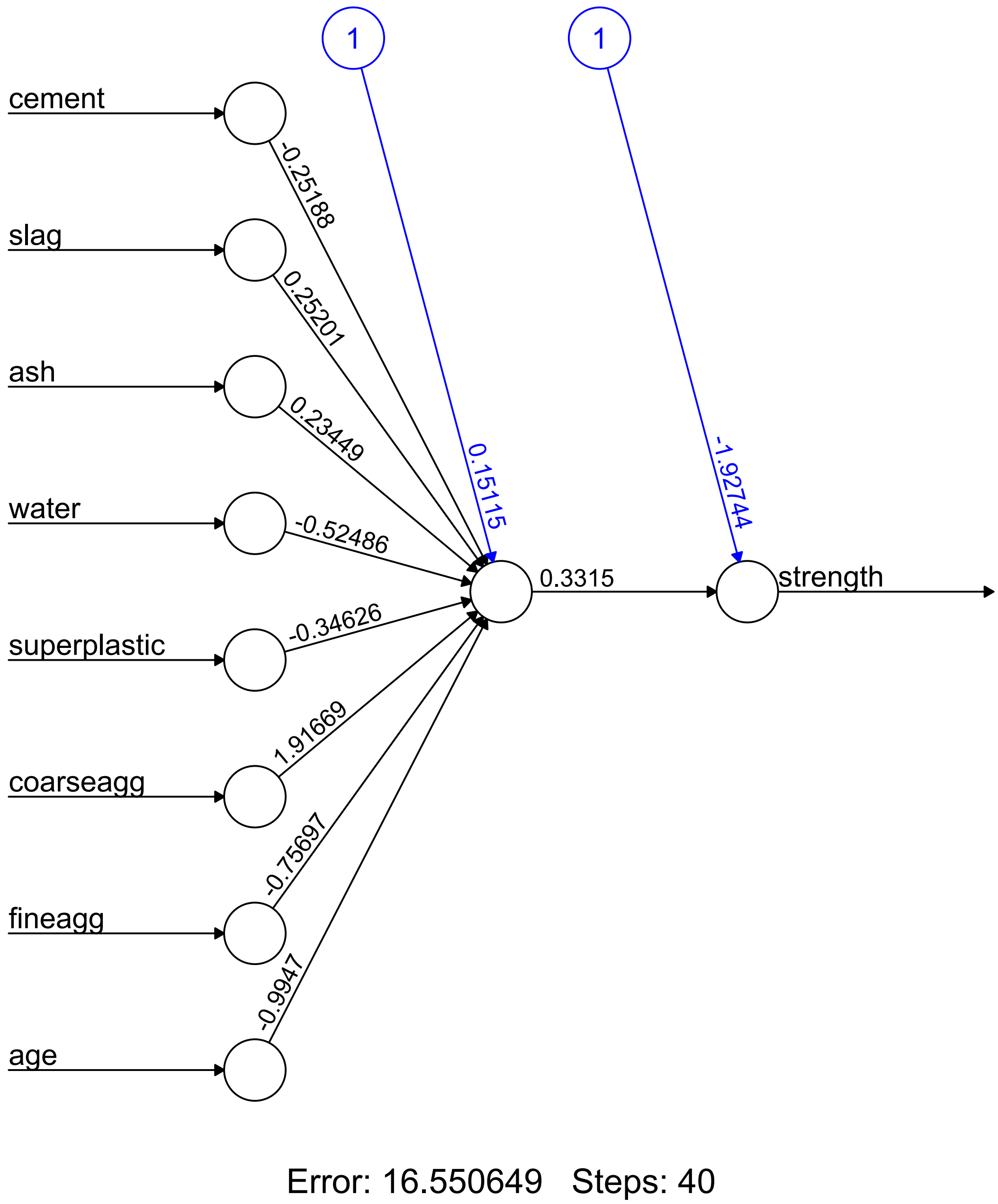


While many of the variables have bell-shaped distributions, others, like age, do not. It likely won't affect the results too much whether we choose range standardization or statistical standardization in this case, so you might try both. For the purposes of our tutorial, we'll go with range standardization. This will allow us to practice writing a function as well! This normalize function applies to variable vectors, and we'll need to apply it to every column of our data frame - including the target!

```
> scaleRange = function(x) {
+   return((x-min(x))/(max(x)-min(x)))
+ }
> concrete_norm = as.data.frame(lapply(concrete, scaleRange))
> train_norm=concrete_norm[TrainInd, ]
> valid_norm=concrete_norm[!TrainInd, ]
```

Let's get started with something simple, a neural network with only one hidden unit, and see how it performs compared to the linear regression. You can then plot the resulting network structure with the associated weights using the plot() command.

```
> library(neuralnet)
> nnet1 = neuralnet(strength~cement+slag+ash+water+superplastic+coarseagg+fineagg+age, data=train_norm, hidden=1)
> #plot(nnet1)
```



To predict the strength variable on the validation data, we'll need to use the compute() function of the neuralnet package. This function computes not only the final model output (in the $net.result component), but also the computed value at each neuron (in the $neurons component). We'll only want to retrieve the former. We'll use the predicted values to calculate a validation R^2 as well as a MAPE.

```
> results1 = compute(nnet1, valid_norm[,1:8])
> nnet1Pred=results1$net.result
> nnet1Rsq = cor(nnet1Pred,valid_norm$strength)
> nnet1MAPE = mean((nnet1Pred-valid_norm$strength)/valid_norm$strength)
> cat("1 Hidden Unit R-squared:", nnet1Rsq)
```

```
1 Hidden Unit R-squared: 0.840282435
```

```
> cat("1 Hidden Unit MAPE:", nnet1MAPE)
```

```
1 Hidden Unit MAPE: 0.08960350585
```

The validation R^2 from our model increases dramatically to 0.83, and the MAPE calculation has a problem. It is important to rescale the data to accurately compare the predicted values with the actual values of concrete strength. This is the problem with the MAPE calculation, as we've created 0 variables for strength in the observations that had the minimum strength. If those minimum strength observations are in our validation data, the MAPE will be undefined. To rescale our predictions to the original measure of strength, we need to multiply them by the range of the original variables and add in the minimum value. We will then recompute both the R^2 (which shouldn't change since we're only taking a linear transformation of our predictions) and the MAPE for the neural network model.

```
> nnet1PredRescale = nnet1Pred*(max(concrete$strength)-min(concrete$strength))+min(concrete$strength)
> nnet1Rsq = cor(nnet1PredRescale,valid$strength)
> nnet1MAPE = mean((nnet1PredRescale-valid$strength)/valid$strength)
> cat("1 Hidden Unit R-squared after Rescaling:", nnet1Rsq)
```

```
1 Hidden Unit R-squared after Rescaling: 0.840282435
```

```
> cat("1 Hidden Unit MAPE after Rescaling:", nnet1MAPE)
```

```
1 Hidden Unit MAPE after Rescaling: 0.06577119777
```

Our MAPE is starting to look like something we might have more confidence in! This was an very simple neural network model (a good place to start) so let's see how we do if we increase the number of hidden units first to 3 and then to 5:

```
> nnet3 = neuralnet(strength~cement+slag+ash+water+superplastic+coarseagg+fineagg+age, data=train_norm, hidden=3)
> nnet5 = neuralnet(strength~cement+slag+ash+water+superplastic+coarseagg+fineagg+age, data=train_norm, hidden=5)
> results3 = compute(nnet3, valid_norm[,1:8])
> nnet3PredRescale = results3$net.result*(max(concrete$strength)-min(concrete$strength))+min(concrete$strength)
> nnet3Rsq = cor(nnet3PredRescale,valid$strength)
> nnet3MAPE = mean((nnet3PredRescale-valid$strength)/valid$strength)
> cat("3 Hidden Unit R-squared after Rescaling:", nnet3Rsq)
```

```
3 Hidden Unit R-squared after Rescaling: 0.9196445059
```

```
> cat("3 Hidden Unit MAPE after Rescaling:", nnet3MAPE)
```

```
3 Hidden Unit MAPE after Rescaling: 0.0395292331
```

```
> results5 = compute(nnet5, valid_norm[,1:8])
> nnet5PredRescale = results5$net.result*(max(concrete$strength)-min(concrete$strength))+min(concrete$strength)
> nnet5Rsq = cor(nnet5PredRescale,valid$strength)
> nnet5MAPE = mean((nnet5PredRescale-valid$strength)/valid$strength)
> cat("5 Hidden Unit R-squared after Rescaling:", nnet5Rsq)
```

```
5 Hidden Unit R-squared after Rescaling: 0.9353053032
```

```
> cat("5 Hidden Unit MAPE after Rescaling:", nnet5MAPE)
```

```
5 Hidden Unit MAPE after Rescaling: 0.01566389657
```