The University of Akron

Management Department

Advanced Data Analytics 6500-663

Business Report

Modeling the strength of concrete with artificial neural networks (ANN)

By Kirill Samaray

Introduction

Concrete is one of the most widely used building materials in the world because of its durability, dependability, and affordability. However, because a number of different factors, including mix design, environmental considerations, and others, are involved, predicting the strength of concrete structures is a difficult process. Artificial neural networks (ANNs) have become a promising method for simulating and forecasting concrete strength in recent years. The backpropagation technique is used by ANNs to learn from past data and provide precise predictions. The purpose of this business report is to investigate how artificial neural networks (ANNs) can be used to simulate concrete strength and to assess how well the backpropagation algorithm performs when it comes to making precise predictions.

Problem Statement

To predict the strength of the concrete is not an easy task although remains a vital one for safety and quality reasons. Conventional mathematical models are helpful but consume an excessive amount of time and may be resource-consuming. Besides, the accuracy of mathematical models sometimes doesn't deliver a desired accuracy. One of the alternative approaches utilized nowadays is the usage of artificial neural networks (ANNs) which have shown promise in modeling the strength using the backpropagation algorithm. The problem statement of this report is to explore the usage of ANNs in modeling the strength of concrete to identify its effectiveness.

Objectives

The objectives of this business report which will use R programming language, are as follows:

- To provide an overview of ANNs in predicting concrete strength and to identify the most effective approaches and parameters.
- To evaluate the performance of ANNs in predicting the strength of concrete by comparing the predicted values with the actual experimental results.
- To assess the accuracy of the backpropagation algorithm in improving the prediction accuracy of ANNs.
- To provide recommendations for the use of ANNs with backpropagation algorithm in predicting concrete strength and to identify areas for future research.

Methodology

The methodology used in this report is based on the analysis of the publicly available data set "concrete.csv" and a review of the existing literature: Brett Lantz, Machine Learning with R, 2nd

Ed., Packet Publishing, 2015 (ISBN: 978-1-78439-390-8), Navdeep Singh Gill, Artificial Neural Networks Applications and Algorithms, 15 January 2023

The following report is split into the sections:

Step 1. Overview of the model: The overview of the ANN model specifying its advantages and disadvantages will be discussed here.

Step 2. Collecting data: The first step is to collect a large dataset of concrete strength values, along with the associated features. This dataset will be used to train and test the ANN model.

Step 3. Exploring and preparing data: The collected data will be explored and preprocessed if necessary. The data will also be normalized to ensure that all variables are on the same scale. The ANN architecture will also be created. This will involve selecting the appropriate number of layers and neurons, as well as the activation function and learning rate.

Step 4. Model training: The ANN model will be trained using the backpropagation algorithm. The dataset will be divided into training and testing sets. The training set will be used to train the model while the testing set will be used to evaluate the final performance of the model.

Step 5. Model evaluation: The performance of the ANN model will be evaluated by comparing the predicted values with the actual experimental results. The correlation will be calculated to assess the performance of the model.

Step 6. Model optimization: The number of hidden layers will be adjusted to optimize the model.

Step 7. Recommendations: Finally, recommendations will be provided for the use of ANNs with backpropagation algorithm in predicting concrete strength, along with suggestions for future research.

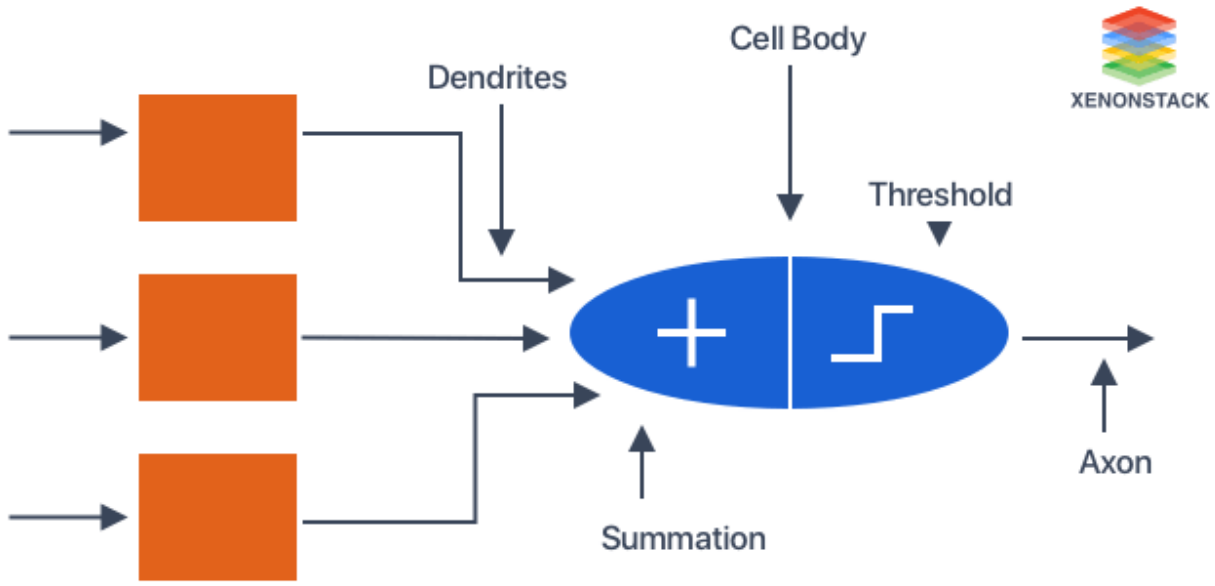
Step 1. Overview of the model

Artificial Neural Networks (ANNs) are types of machine learning algorithms that are having similar working principles as a human brain. It has a massive number of connected nodes that are structured in layers. There is an input which is sent to the nodes either from the external sources or other connected nodes. This is transformed into the output which is subsequently sent to other nodes in the system. To measure the strength of each connection there is a weight attached to each one of them.

The inner working principle of the algorithm can be only partially understood thus it is called “black box”. The way the input is transformed into the output may not be clear although the output is accurate. It may cause some complications especially in the fields which require a high level of precision such as healthcare. In simple terms, a user may not always understand why the algorithm makes a certain decision or how it reached a certain outcome.

This algorithm may have lots of applications in real life. Apart from the healthcare that has been mentioned, the ANNs can be utilized in finance, marketing, and others. Large chunks of data can be processed subsequently identifying trends and market predictions. One of the common applications of the algorithm nowadays can be found in the form of the chatbots and virtual assistants that can understand a simple human language and generate recommendations based on that.

Just like a human used neurons to create some connections, memorize things and learn, the process of ANNs involves adjusting the weights of the connections between the nodes based on the input data and desired output. ANNs can learn from labeled or unlabeled data and can be trained using supervised or unsupervised learning algorithms. During training, the network adjusts the weights of the connections to minimize the difference between the output produced by the network and the desired output.



(Singh, 2023)

The inputs received from the external sources are accumulated by the cell body and then transferred further as per activation function.

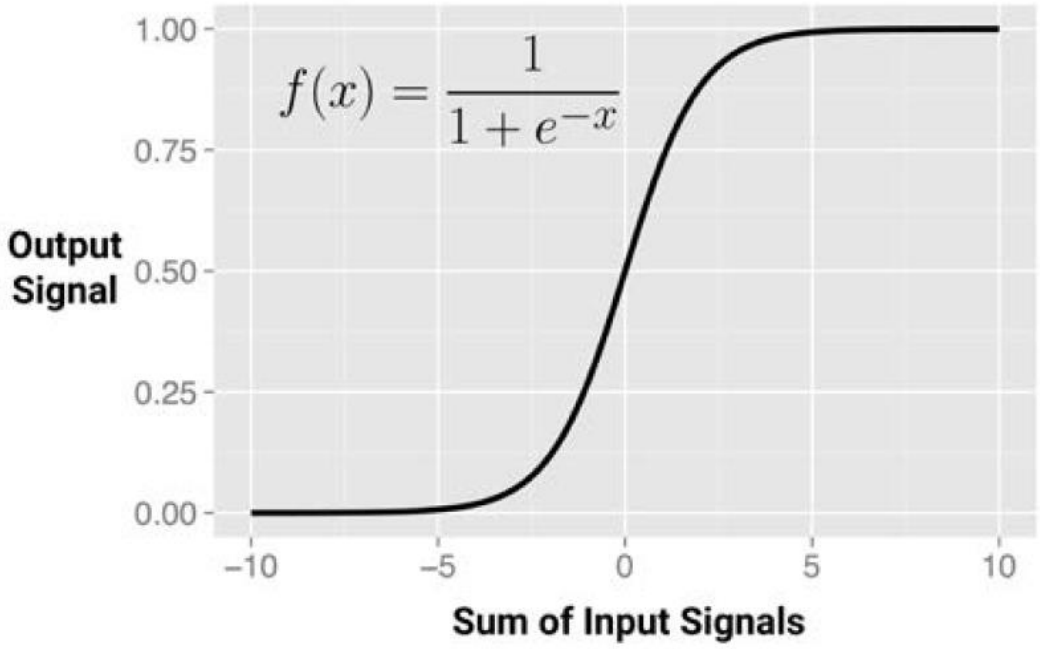
$$y(x) = f\left(\sum_{i=1}^{n} w_i x_i\right)$$

(Lantz, 2015)

The formula above describes an artificial neuron with n input dendrites. Different w weights represent how much n inputs (xi) will contribute to the sum of input signals. The activation function f(x) uses the net total and the resulting signal y(x) which is also an output.

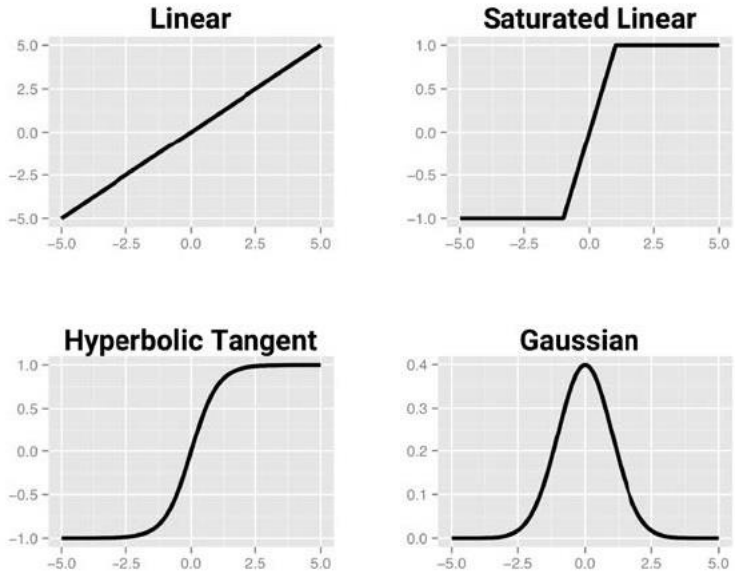
There are various neural networks used although the major characteristics are the following:

The activation function. It is one of the major components of ANNs. This function works with the output of each of the neurons and decides whether the neuron should be activated. It considers the all the input weights from the layers and creates a non-linear transformation to generate the output. This information then goes on to the next layer. If there was no activation function, we would have only linear combinations of inputs and wouldn't have a chance to work with complex data. Here are the following variations of the activation function: sigmoid, tanh, ReLU, threshold and others. Each one of them is suitable for different applications.



(Lantz, 2015)

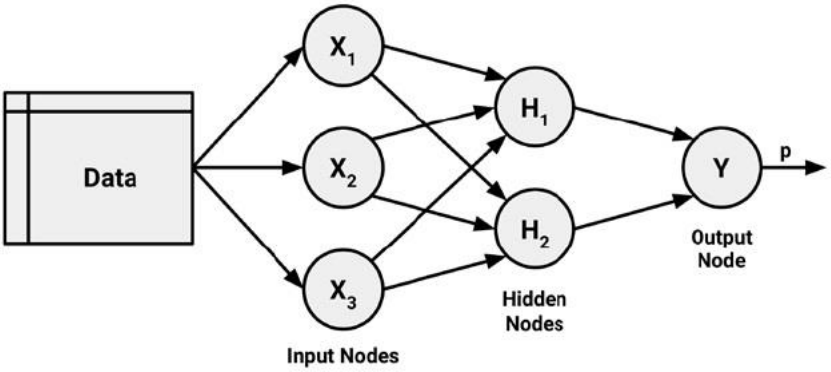
On the picture above there is the most popular activation function – sigmoid. The main advantage of it over the threshold one is that the output is not binary with output ranging anywhere from 0 to 1. Besides, it is possible to calculate derivative for the all the inputs which is important for optimization algorithms.



(Lantz, 2015)

There are also other options of activation functions shown above. The main difference between the functions is the output they generate. It could be either (0,1), (-1,+1), or (-inf,+inf). Certain functions fit different types of data differently so a user can construct specialized ANN. Linear activation functions generate network like a linear regression, a Gaussian function creates a Radial Basis Function (RBF) network.

Network Topology. This one constitutes the structure of the ANNs. It denotes the arrangement of nodes, layers, and connections between them that determines how the network processes information. Input nodes are the series of neurons which receive unprocessed signals. Layers are the groups of inputs and outputs. The network can be created of a single layer or multiple ones depending on the complexity of data. The latter ones use a series of hidden layers which process the signals from the input nodes before transferring it to the output node. Such networks are also called Deep Neural Networks (DNN) and the training process is called deep learning. Weights are normally labeled as w1, w2, w3, etc. Here are the following variations of network topologies: feedforward, recurrent, and convolutional. Feedforward networks are the simplest type of topology which is made of series of layers with information flowing in one direction, from input to output. Recurrent networks have feedback connections, which allow information to flow back into the network, making them well suited to modeling sequences and time-series data. Convolutional networks are designed to process data that has a grid-like structure, such as images or audio. Each one of the topologies has its own advantages and the selection is normally based on the problem and type of data used.



(Lantz, 2015)

Apart from the layer's diversity, there could also be a different number of nodes. It usually depends on the number of features in the inputs, the same as the number of outputs are dictated by the number of outcomes required or the number of class levels. As for the hidden nodes, it is left at the discretion of the user before the training phase.

Throughout the data processing stage, connections between neurons become stronger or weaker depending on the outcome. Consequently, the connection weights are adjusted to illustrate various patterns if any. The algorithm called backpropagation was created to increase the efficiency of adjusting the weights. Even though it helped to increase the popularity of the ANN in general, there are still certain disadvantages illustrated below:

Strengths	Weaknesses
• Can be adapted to classification or numeric prediction problems • Capable of modeling more complex patterns than nearly any algorithm • Makes few assumptions about the data's underlying relationships	• Extremely computationally intensive and slow to train, particularly if the network topology is complex • Very prone to overfitting training data • Results in a complex black box model that is difficult, if not impossible, to interpret

(Lantz, 2015)

Backpropagation algorithm consists of two processes and multiple iterations of cycles of those processes. The starting weights are set at random after which the iterations continue until a stopping criterion is met. There is a forward phase when neurons are activated one by one from the input layers to the output ones, passing it through the weights and activation functions until the output is achieved. And there is also a backward phase when the final output signal is compared to the true value produced by the training data. The error is then propagated backwards to adjust the weights and minimize any errors in the future.

$$E(X,\theta) = \frac{1}{2N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2,$$

(McGonagle, 2023)

In the formula above, we have yi which is a target value for input-output pair (xi, yi) and yi is the computed output of the network on input xi.

A gradient descent is used for determining how much should the weights be changed. A derivative of each activation function is used to obtain the gradient in the direction of each of the weights. The gradient then tells how much the error should be adjusted and it will try to adjust it so the error will be reduced as much as possible. This reduction amount is also known as a learning rate. The greater the learning rate, the faster the algorithm will perform.

Training Algorithm. The training algorithm is needed when we work with weights of the ANN during the training process. The main function of it is to make sure the difference between the predicted output of the network and the actual output is minimal. There are several types of training algorithms, including stochastic gradient descent (SGD), Adam, and RMSprop. SGD is the most used training algorithm and works by updating the weights based on the error between the predicted output and the actual output.

Step 2. Collecting data

We used the real data available from the open sources called concrete.csv which contains 1030 examples of observations of concrete with 8 different features of components used in the mixture. The features are the following: the amount (kg/m^3) of cement, slag, ash, water, superplasticizer, coarse aggregate, fine aggregate, aging time (days). All of these features are supposed to be related to the final compressive strength of the concrete.

Step 3. Exploring and preparing data

First, we loaded the data into an R object with the help of read.csv() and inspected the structure of the data:

```
> concrete = read.csv("concrete.csv")
> 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 ...
```

As is shown on the picture above, the structure matches with the one described earlier: 8 different features of the concrete with one outcome. Since neural networks work best when the input values are built around zero with a narrow range, we had to rescale the data by either normalizing or standardization function. We used normalization function in our case:

```
> normalize = function(x) {
+ return((x - min(x)) / (max(x) - min(x)))
+ }
```

With the help of lapply() function we can utilize normalize() function to every column of the concrete data frame and confirm whether our approach worked out:

```
> concrete_norm <- as.data.frame(lapply(concrete, normalize))
> summary(concrete_norm$strength)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.2664  0.4001  0.4172  0.5457  1.0000
> summary(concrete$strength)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2.33   23.71   34.45   35.82   46.13   82.60
```

Here we called summary() function and compared our normalized min and max values with the previous ones. Since with the normalized option it shows min as 0 and max as 1, it means that the normalization worked well.

Then we split the data into 75% of data into training set and 25% into testing data set by dividing it into 2 parts considering the randomness of the original file:

```
> concrete_train = concrete_norm[1:773, ]
> concrete_test = concrete_norm[774:1030, ]
```

Step 4. Model training

In order to create the model indicating the relationship between its features and final outcome (strength) we used a multilayer feedforward neural network. A package called neuralnet was used for such purpose which also provides a functionality to plot a network topology. Here is a syntax of the above-mentioned package:

Neural network syntax

using the `neuralnet()` function in the `neuralnet` package

Building the model:

```
m <- neuralnet(target ~ predictors, data = mydata,
               hidden = 1)
```

- `target` is the outcome in the `mydata` data frame to be modeled
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` specifies the data frame in which the `target` and `predictors` variables can be found
- `hidden` specifies the number of neurons in the hidden layer (by default, 1)

The function will return a neural network object that can be used to make predictions.

Making predictions:

```
p <- compute(m, test)
```

- `m` is a model trained by the `neuralnet()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier

The function will return a list with two components: `$neurons`, which stores the neurons for each layer in the network, and `$net.result`, which stores the model's predicted values.

Example:

```
concrete_model <- neuralnet(strength ~ cement + slag
  + ash, data = concrete)
model_results <- compute(concrete_model,
  concrete_data)
strength_predictions <- model_results$net.result
```

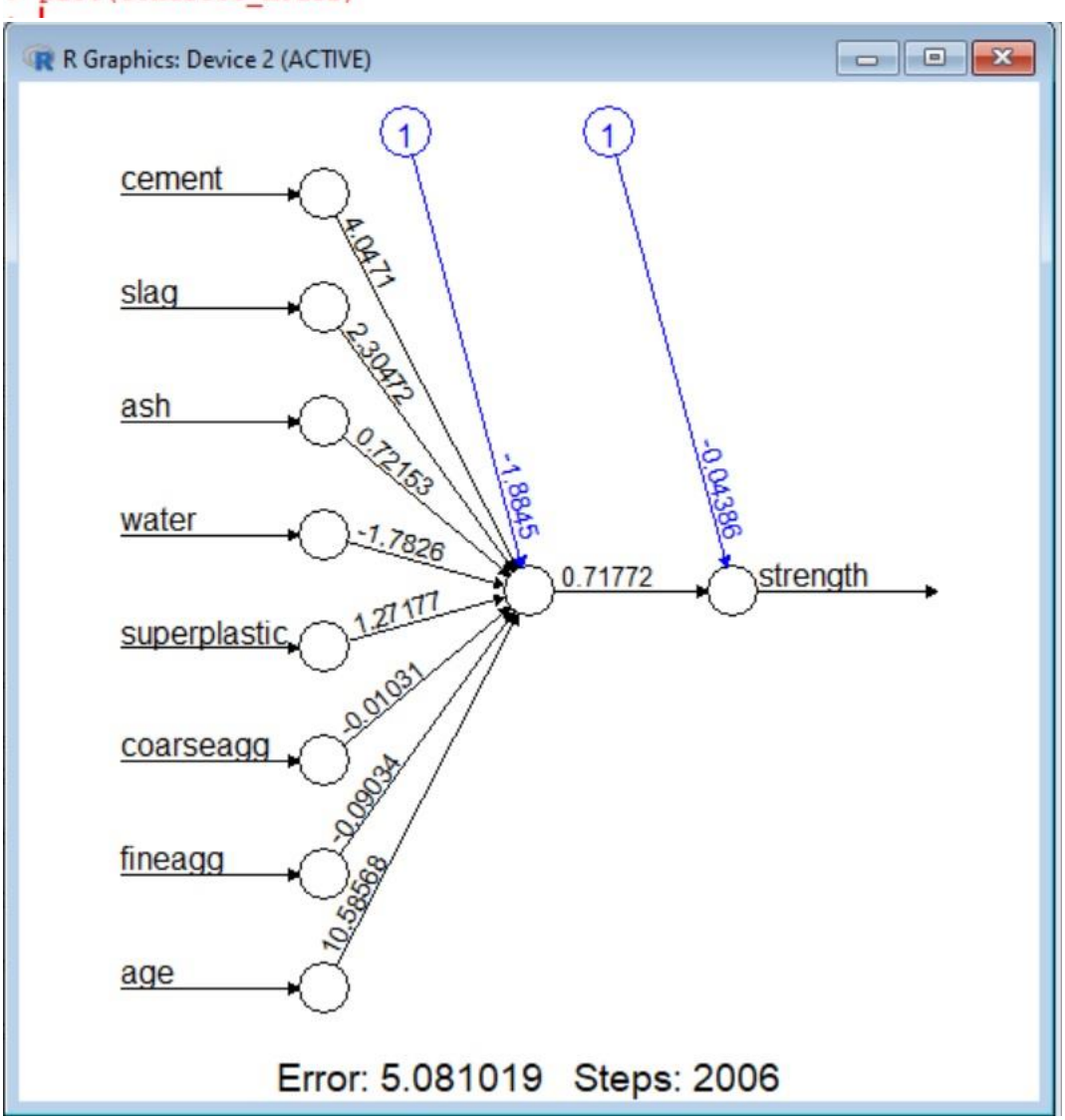
(Lantz, 2015)

As we mentioned earlier, there could be multiple hidden nodes, so we started the training with only one:

```
> concrete_model = neuralnet(strength ~ cement + slag
+ + ash + water + superplastic + coarseagg + fineagg + age,
+ data = concrete_train)
```

In order to visualize our model we used a plot() function:

```
> plot(concrete_model)
```



In the picture above we can see one input for each of the features and one hidden node plus one output node indicating the strength of the concrete. All the weights are shown on the picture as well for all the eight features with bias terms illustrated by the nodes with the number 1. These bias terms are the constants which enable the nodes values to be adjusted upward or downward. The values mentioned at the bottom of the diagram are the number of the training steps and the Sum of Squared Errors which constitutes the error measure. The lowe this number, the better predictive performance of the model. It helps with estimating the performance on the training data, although doesn't help with the same for the testing data.

Step 5. Model evaluation

We have already mentioned earlier that network topology diagram helps us to understand the black box of the Artificial Neural Network. Nevertheless, to see how the model fits future data we used compute() function so that to generate predictions on the test dataset:

```
> model_results = compute(concrete_model, concrete_test[1:8])
```

With this function we were able to create a list with 2 parts: $neurons with neurons for every layer in the network and $net.result with predicted values.

```
> predicted_strength = model_results$net.result
```

Since we are dealing with numeric values and not the classification ones we could not use a confusion matrix but substitute it with the correlation between predicted values and true ones. We used cor() function to get the information on the strength of the linear association between two values:

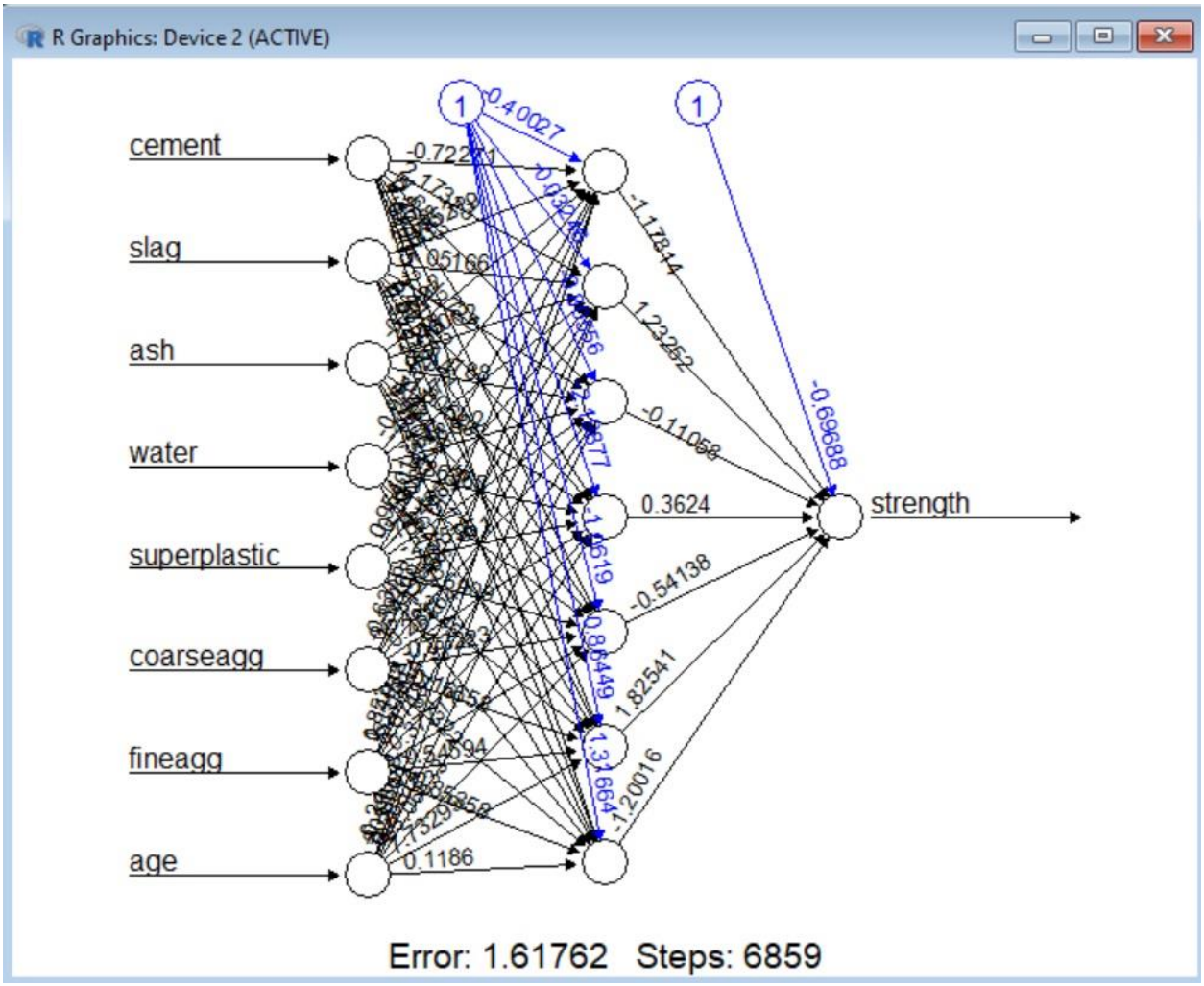
```
> cor(predicted_strength, concrete_test$strength)
          [,1]
[1,] 0.8060488
```

We can see a correlation of 0.806 which is quite close to 1 indicating a strong relationship between predicted strength and true strength of the concrete. It also means that the model we created shows a decent performance even though only 1 hidden layer was used.

Step 6. Model optimization

We mentioned earlier that increasing the number of hidden layers usually helps with managing more complex tasks. We decided to try improving the performance of our model by increasing the number of hidden layers to 7:

```
> concrete_model2 = neuralnet(strength ~ cement + slag +
+ ash + water + superplastic +
+ coarseagg + fineagg + age,
+ data = concrete_train, hidden = 7)
> plot(concrete_model2)
```



After training the model and plotting the diagram, we could notice an increased number of connections between the nodes. It also resulted in a reduced number of errors from 5.08 down to 1.62. The number of training steps increased from 2006 to 6859 which also proves that the complexity of the model has been increased.

Then we used the same approach as before and calculated a new correlation:

```
> model_results2 = compute(concrete_model2, concrete_test[1:8])
> predicted_strength2 = model_results2$net.result
> cor(predicted_strength2, concrete_test$strength)
             [,1]
[1,] 0.9347889
```

As shown above, by increasing the number of hidden layers we could reach a new correlation of 0.935 which is a 16% improvement.

Step 7. Recommendations

Based on the analysis the following recommendations are suggested:

- Promote future development of artificial neural network (ANN) technology for modeling the strength of concrete. It has shown high accuracy and precision in predicting concrete strength and can help companies with optimizing its industrial performance.
- Invest in educating the team of professionals with the necessary skills and expertise to develop and deploy ANN models effectively. This team should have experience in data analysis, programming, and statistical modeling to ensure the success of ANN technology adoption.
- Invest in the necessary infrastructure, such as computing resources and software tools, to support the development and deployment of ANN models.
- Conduct further research to improve the accuracy and reliability of the ANN models. This can involve exploring different ANN architectures, experimenting with various input and output parameters, and testing the models under different conditions to validate their performance.

Conclusion

The usage of artificial neural network (ANN) technology for modeling the strength of concrete can provide large benefits to the company when it comes to the quality, efficiency, and cost savings. It was shown that the ANN technology can demonstrate a high accuracy and precision in predicting concrete strength, making it a valuable tool for optimizing concrete mix designs and improving the quality and reliability of concrete products.

We presented a detailed analysis of the model training in this report and demonstrated the potential of ANN technology in concrete strength modeling as well as provided recommendations for the company to successfully adopt and implement this technology. As the demand for high-quality and durable concrete structures increases, it is vital for companies in the concrete industry to stay ahead of their competitors by adopting the latest technologies. The adoption of ANN technology for concrete strength modeling can help the company to achieve this goal and position itself as a leader in the industry.

References

1. Brett Lantz, Machine Learning with R, 2nd Ed., Packet Publishing, 2015 (ISBN: 978-1-78439-390-8)
2. Navdeep Singh Gill, Artificial Neural Networks Applications and Algorithms, 15 January 2023
3. John McGonagle, George Shaikouski, Christopher Williams, Backpropagation,

Coding part

```
concrete = read.csv("concrete.csv")
str(concrete)

normalize = function(x) {
return((x - min(x)) / (max(x) - min(x)))
}

concrete_norm <- as.data.frame(lapply(concrete, normalize))

summary(concrete_norm$strength)
summary(concrete$strength)

concrete_train = concrete_norm[1:773, ]
concrete_test = concrete_norm[774:1030, ]

install.packages("neuralnet")
library(neuralnet)

concrete_model = neuralnet(strength ~ cement + slag
+ ash + water + superplastic + coarseagg + fineagg + age,
data = concrete_train)

plot(concrete_model)

model_results = compute(concrete_model, concrete_test[1:8])

predicted_strength = model_results$net.result
```

```
cor(predicted_strength, concrete_test$strength)

concrete_model2 = neuralnet(strength ~ cement + slag +
ash + water + superplastic +
coarseagg + fineagg + age,
data = concrete_train, hidden = 7)

plot(concrete_model2)

model_results2 = compute(concrete_model2, concrete_test[1:8])
predicted_strength2 = model_results2$net.result
cor(predicted_strength2, concrete_test$strength)
```