Assignment 1 ZHOU DUO

August 30, 2021

Regression through Neural Network by neuralnet & keras

ZHOU DUO, G2103160J

SPMS, NTU

1 Introduction of Neural Network

1.1 What is Neural Network

- Neural networks are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.
- Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.Based on this characteristic, choose to use neural network regression analysis.

1.2 How do neural networks work

- Think of each individual node as its own linear regression model, composed of input data, weights, a bias (or threshold), and an output. The formula would look something like this:

$$activation\ function : \sum_{i=1}^{m} w_i \cdot x_i + bias$$

- Once an input layer is determined, weights are assigned. These weights help determine the importance of any given variable, with larger ones contributing more significantly to the output compared to other inputs. All inputs are then multiplied by their respective weights and then summed. Afterward, the output is passed through an activation function(mostly ReLU this assignment)which determines the output.

$$Rectified\ Linear\ Unit\ = max(0, x)$$

1.3 How do neural networks work

1.3.1 Activation function

- This assignment used perceptrons to illustrate some mathematics at play here, neural networks leverage ReLU neurons, which are distinguished by having values between 0 and 1. Since neural networks behave similarly to decision trees, cascading data from one node to another, having x values between 0 and 1 will reduce the impact of any given change of a single variable on the output of any given node, and subsequently, the output of the neural network.
- If that output exceeds a given threshold, it activates the node, passing data to the next layer in the network. This results in the output of one node becoming in the input of the next node.

1.3.2 Cost function

- We adjust the weights or the threshold and achieve different outcomes from the model. When we observe one decision, we can see how a neural network could make increasingly complex decisions depending on the output of previous decisions or layers.
- As we start to think about more practical use cases for classification, we'll leverage supervised learning, or labeled datasets, to train the algorithm. As we train the model, we'll want to evaluate its accuracy using a cost (or loss) function. This is also commonly referred to as the mean squared error (MSE). Sometimes, we use SSE,too. In the equation below:

$$SSE = \sum_{i=1}^{n} w_i (y_i - \hat{y}_i)^2$$

$$MSE = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

- Ultimately, the goal is to minimize our cost function to ensure correctness of fit for any given observation. As the model adjusts its weights and bias, it uses the cost function and reinforcement learning to reach the point of convergence, or the local minimum. The process in which the algorithm adjusts its weights is through gradient descent, allowing the model to determine the direction to take to reduce errors (or minimize the cost function). With each training example, the parameters of the model adjust to gradually converge at the minimum.
- This assignment use MLP to do some regression analysis. The assginment 2 will introduce keras to do regression prodiction.

2 DNN in concrete dataset

- chapter7. neural network
- refer to p231 in ML with R

2.1 Step 1 – collecting data

- For this analysis, we will utilize data on the compressive strength of concrete donated to the UCI Machine Learning Data Repository (https://archive.ics.uci.edu/ml/datasets/concrete+compressive+strength) by I-Cheng Yeh. The concrete dataset contains 1,030 examples of concrete with eight features describing the components used in the mixture.
- These features are thought to be related to the final compressive strength and they include the amount (in kilograms per cubic meter) of cement, slag, ash, water, superplasticizer, coarse aggregate, and fine aggregate used in the product in addition to the aging time (measured in days).

[1]:
```
getwd()
```

'/Users/clause/Documents/NTUPresentationBeamer-master'

2.2 Step 2 – exploring and preparing the data

- As usual, we'll begin our analysis by loading the data into an R object using the read.csv() function, and confirming that it matches the expected structure:

[2]:
```
concrete <- read.csv("Concrete.csv")
str(concrete)
summary(concrete[9])
```

```
'data.frame':   1030 obs. of  9 variables:
 $ cement      : num  540 540 332 332 199 ...
 $ slag        : num  0 0 142 142 132 ...
 $ ash         : num  0 0 0 0 0 0 0 0 0 0 ...
 $ water       : num  162 162 228 228 192 228 228 228 228 228 ...
 $ superplastic: num  2.5 2.5 0 0 0 0 0 0 0 0 ...
 $ coarseagg   : num  1040 1055 932 932 978 ...
 $ fineagg     : num  676 676 594 594 826 ...
 $ age         : int  28 28 270 365 360 90 365 28 28 28 ...
 $ strength    : num  80 61.9 40.3 41 44.3 ...
```

```
   strength
 Min.   : 2.33
 1st Qu.:23.71
 Median :34.45
 Mean   :35.82
 3rd Qu.:46.13
 Max.   :82.60
```

- The nine variables in the data frame correspond to the eight features and one outcome we expected, although a problem has become apparent. Neural networks work best when the input data are scaled to a narrow range around zero, and here, we see values ranging anywhere from zero up to over a thousand.

- Typically, the solution to this problem is to rescale the data with a normalizing or standardization function. If the data follow a bell-shaped curve , then it may make sense to use standardization via R's built-in scale() function.

$$nomalized = \frac{x - min(x)}{max(x) - min(x)}$$

- On the other hand, if the data follow a uniform distribution or are severely nonnormal, then normalization to a 0-1 range may be more appropriate. In this case, we'll use the latter to normalize our dataset.
- To confirm that the normalization worked, we can see that the minimum and maximum strength are now 0 and 1, respectively:

[3]:
```
# normalizing
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}
concrete_norm <- as.data.frame(lapply(concrete, normalize))
summary(concrete_norm[[9]])
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.2664  0.4001  0.4172  0.5457  1.0000
```

- Following Yeh's precedent in the original publication, we will partition the data into a training set with 75 percent of the examples and a testing set with 25 percent. The CSV file we used was already sorted in random order, so we simply need to divide it into two portions:
- We'll use the training dataset to build the neural network and the testing dataset to evaluate how well the model generalizes to future results. As it is easy to overfit a neural network, this step is very important.

[4]:
```
# dividing into train & test
concrete_train <- concrete_norm[1:773, ]
concrete_test <- concrete_norm[774:1030, ]
str(concrete_train)
str(concrete_test)
```

```
'data.frame':   773 obs. of  9 variables:
 $ cement      : num  1 1 0.526 0.526 0.221 ...
 $ slag        : num  0 0 0.396 0.396 0.368 ...
 $ ash         : num  0 0 0 0 0 0 0 0 0 0 ...
 $ water       : num  0.321 0.321 0.848 0.848 0.561 ...
 $ superplastic: num  0.0776 0.0776 0 0 0 ...
 $ coarseagg   : num  0.695 0.738 0.381 0.381 0.516 ...
 $ fineagg     : num  0.206 0.206 0 0 0.581 ...
 $ age         : num  0.0742 0.0742 0.739 1 0.9863 ...
 $ strength    : num  0.967 0.742 0.473 0.482 0.523 ...
'data.frame':   257 obs. of  9 variables:
 $ cement      : num  0.639 0.639 0.409 0.541 0.541 ...
```

```
$ slag        : num  0 0 0 0 0 0 0 0 0 0 ...
$ ash         : num  0 0 0 0 0 0 0 0 0 0 ...
$ water       : num  0.513 0.513 0.513 0.505 0.505 ...
$ superplastic: num  0 0 0 0 0 0 0 0 0 0 ...
$ coarseagg   : num  0.715 0.901 0.881 0.779 0.779 ...
$ fineagg     : num  0.364 0.477 0.452 0.401 0.401 ...
$ age         : num  0.0742 0.0165 0.0742 0.0165 0.0742 ...
$ strength    : num  0.437 0.114 0.251 0.235 0.368 ...
```

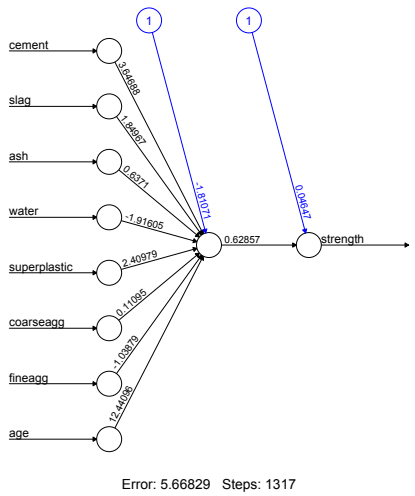
2.3 Step 3 – training a model on the data

- To model the relationship between the ingredients used in concrete and the strength of the finished product, we will use a multilayer feedforward neural network. The neuralnet package by Stefan Fritsch and Frauke Guenther provides a standard and easy-to-use implementation of such networks. It also offers a function to plot the network topology.
- We'll begin by training the simplest multilayer feedforward network with only a single hidden node to explore the relationship between strenth and others:

[5]:
```
# check & install packages
requiredPackages <- 'neuralnet'
if (length(setdiff(requiredPackages, rownames(installed.packages()))) > 0) {
  install.packages(setdiff(requiredPackages, rownames(installed.packages())))
}
library('neuralnet')
concrete_model <- neuralnet(strength ~ .,
                            data = concrete_train)
```

- We can then visualize the network topology using the plot() function on the resulting model object:

[6]:
```
plot(concrete_model) # here is some thing wrong, jupyter cannot shou the plot,␣
↪so I use the plot showed by Rstudio
```



- In this simple model, there is one input node for each of the eight features, followed by a single hidden node and a single output node that predicts the concrete strength. The weights for each of the connections are also depicted, as are the bias terms (indicated by the nodes labeled with the number 1).

$$activation\ function : \sum_{i=1}^{m} w_i \cdot x_i + bias$$

- The bias terms are numeric constants that allow the value at the indicated nodes to be shifted upward or downward, much like the intercept in a linear equation.
- At the bottom of the figure, R reports the number of training steps and an error measure called the Sum of Squared Errors (SSE), is the sum of the squared predicted minus actual values.

$$SSE = \sum_{i=1}^{n} w_i (y_i - \hat{y}_i)^2$$

- A lower SSE implies better predictive performance. This is helpful for estimating the model's performance on the training data, but tells us little about how it will perform on unseen data.

2.4 Step 4 – evaluating model performance

- The network topology diagram gives us a peek into the black box of the ANN, but it doesn't provide much information about how well the model fits future data. To generate predictions on the test dataset, we can use the compute() as follows:

[7]:
```
model_results <- compute(concrete_model, concrete_test[1:8])
str(model_results)
```

```
List of 2
 $ neurons   :List of 2
  ..$ : num [1:257, 1:9] 1 1 1 1 1 1 1 1 1 1 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:257] "774" "775" "776" "777" ...
  .. .. ..$ : chr [1:9] "" "cement" "slag" "ash" ...
  ..$ : num [1:257, 1:2] 1 1 1 1 1 1 1 1 1 1 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:257] "774" "775" "776" "777" ...
  .. .. ..$ : NULL
 $ net.result: num [1:257, 1] 0.39 0.244 0.25 0.227 0.332 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:257] "774" "775" "776" "777" ...
  .. ..$ : NULL
```

- The compute() function works a bit differently from the predict() functions we've used so far. It returns a list with two components: *neurons*, which stores the neurons for each layer in the network, and *net.result*, which stores the predicted values. We'll want the latter:
- Because this is a numeric prediction problem rather than a classification problem, we cannot use a confusion matrix to examine model accuracy. Instead, we must measure the correlation

between our predicted concrete strength and the true value. This provides insight into the strength of the linear association between the two variables.

[8]:
```
predicted_strength <- model_results$net.result
cor(predicted_strength, concrete_test$strength)
```

A matrix: 1 × 1 of type dbl 0.7203357

- Correlations close to 1 indicate strong linear relationships between two variables. Therefore, the correlation here of about 0.723 indicates a fairly strong relationship. This implies that our model is doing a fairly good job, even with only a single hidden node.
- Given that we only used one hidden node, it is likely that we can improve the performance of our model. Let's try to do a bit better.

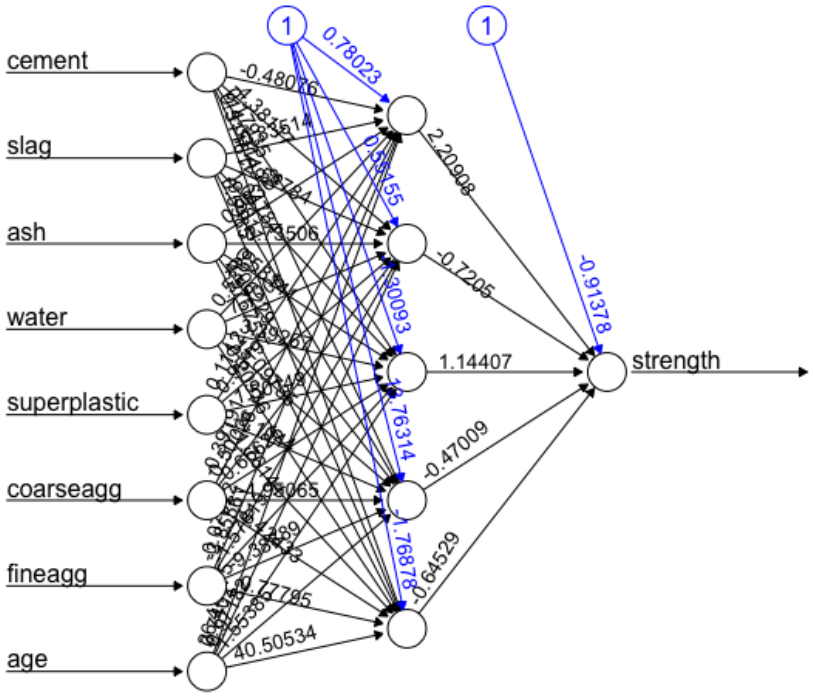
2.5 Step 5 – improving model performance

- As networks with more complex topologies are capable of learning more difficult concepts, let's see what happens when we increase the number of hidden nodes to five. We use the neuralnet() function as before, but add the loop to set the hidden nodes =1,2,3,4,5 respectively :

[9]:
```
for (i in c(1,2,3,4,5)) {
  hidden <- i
  concrete_model <- neuralnet(strength ~ .,
                              data = concrete_train, hidden = hidden)
  model_results <- compute(concrete_model, concrete_test[1:8])
  predicted_strength <- model_results$net.result
  output <- cor(predicted_strength, concrete_test$strength)
  print(output)
}
```

```
          [,1]
[1,] 0.7201878
          [,1]
[1,] 0.767318
          [,1]
[1,] 0.797926
          [,1]
[1,] 0.7976516
          [,1]
[1,] 0.7998474
```

- When hidden nodes = 5, correlation get the highest level. Plotting the network again, we see a drastic increase in the number of connections. We can see how this impacted the performance as follows:
- Notice that the reported error (measured again by SSE) has been reduced from 5.67 in the previous model to 1.67 here. Additionally, the number of training steps rose from 1317 to 27791, which should come as no surprise given how much more complex the model has become. More complex networks take many more iterations to find the optimal weights.



- Applying the same steps to compare the predicted values to the true values, we now obtain a correlation around 0.80, which is a considerable improvement over the previous result of 0.72 with a single hidden node:

3 DNN in qsar_fish_toxicity dataset

- refer to p231 in ML with R

3.1 Step 1 – collecting data

- For this analysis, we will utilize data on the compressive strength of concrete donated to the UCI Machine Learning Data Repository (https://archive.ics.uci.edu/ml/datasets/QSAR+fish+toxicity).
- This dataset was used to develop quantitative regression QSAR models to predict acute aquatic toxicity towards the fish Pimephales promelas (fathead minnow) on a set of 908 chemicals. LC50 data, which is the concentration that causes death in 50% of test fish over a test duration of 96 hours, was used as model response.
- The model comprised 6 molecular descriptors: MLOGP (molecular properties), CIC0 (information indices), GATS1i (2D autocorrelations), NdssC (atom-type counts), NdsCH ((atom-type counts), SM1_Dz(Z) (2D matrix-based descriptors).

3.2 Step 2 – exploring and preparing the data

- As usual, we'll begin our analysis by loading the data into an R object using the read.csv() function, and confirming that it matches the expected structure:

[10]:
```
QSAR <- read.csv("qsar_fish_toxicity.csv", header = F, sep = ';')
str(QSAR)
summary(QSAR[7])
```

```
'data.frame':   908 obs. of  7 variables:
 $ V1: num  3.26 2.19 2.12 3.03 2.09 ...
 $ V2: num  0.829 0.58 0.638 0.331 0.827 0.331 0 0 0.499 0.134 ...
 $ V3: num  1.676 0.863 0.831 1.472 0.86 ...
 $ V4: int  0 0 0 1 0 0 0 1 0 0 ...
 $ V5: int  1 0 0 0 0 0 0 0 0 0 ...
 $ V6: num  1.45 1.35 1.35 1.81 1.89 ...
 $ V7: num  3.77 3.12 3.53 3.51 5.39 ...
```

```
       V7
 Min.   :0.053
 1st Qu.:3.152
 Median :3.987
 Mean   :4.064
 3rd Qu.:4.907
 Max.   :9.612
```

- The 7 variables in the data frame correspond to the 6 features and one outcome we expected, although a problem has become apparent. Neural networks work best when the input data are scaled to a narrow range around zero.
- We stilly use the solution to this problem is to rescale the data with a normalizing or standardization function. Normalization to a 0-1 range may be more appropriate.

$$nomalized = \frac{x - min(x)}{max(x) - min(x)}$$

- To confirm that the normalization worked, we can see that the minimum and maximum strength are now 0 and 1, respectively:

[11]:
```
# normalizing
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}
QSAR_norm <- as.data.frame(lapply(QSAR, normalize))
summary(QSAR_norm[[7]])
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.3242  0.4116  0.4196  0.5078  1.0000
```

- Because the dataset is relatively small, we will partition the data into a training set with 67 percent of the examples and a testing set with 33 percent. The CSV file we used was already

sorted in random order, so we simply need to divide it into two portions:

- We'll use the training dataset to build the neural network and the testing dataset to evaluate how well the model generalizes to future results. As it is easy to overfit a neural network, this step is very important.

[12]:
```
# dividing into train & test
QSAR_train <- QSAR_norm[1:606, ]
QSAR_test <- QSAR_norm[607:908, ]
```

3.3 Step 3 – training a model on the data & evaluating model performance

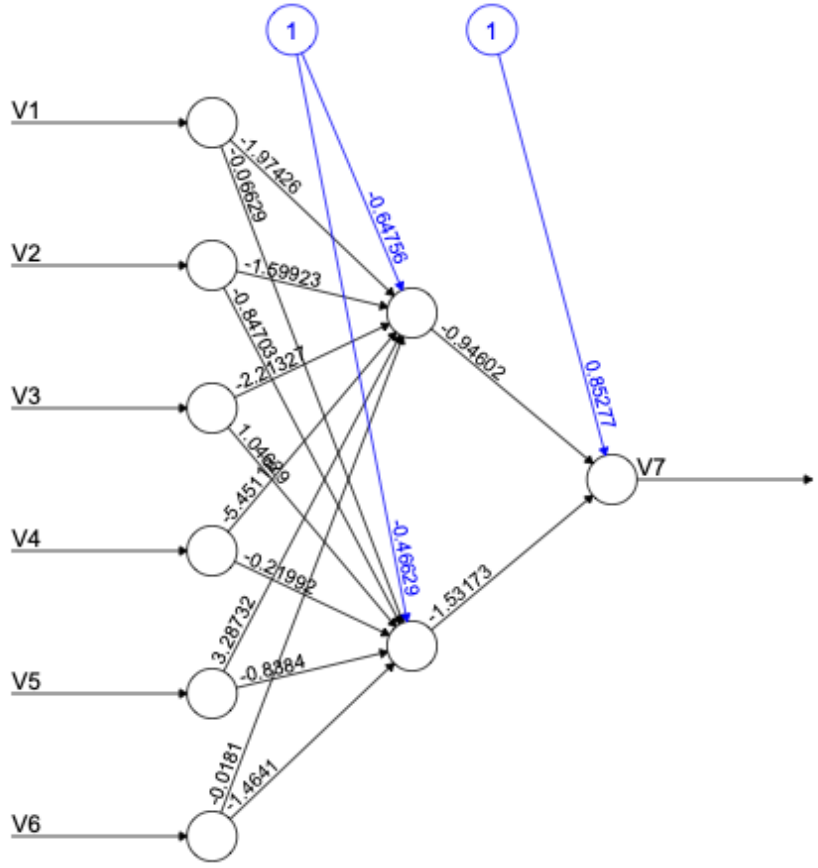
- To model the relationship between the ingredients of chemicals acute aquatic toxicity towards the fish Pimephales promelas, we will use a multilayer feedforward neural network. The neuralnet package by Stefan Fritsch and Frauke Guenther provides a standard and easy-to-use implementation of such networks. It also offers a function to plot the network topology.
- As networks with more complex topologies are capable of learning more difficult concepts, we increase the number of hidden nodes to five. We use the neuralnet() function as before, but add the loop to set the hidden layer =1,2,3,4,5 respectively :

[15]:
```
# neuralnet modeling, try hidden layers = 1-5
library('neuralnet')
for (i in c(1,2,3,4,5)) {
  QSAR_model <- neuralnet(V7 ~ V1 + V2 + V3 + V4 +V5 + V6,
                          data = QSAR_train, hidden = hidden)
  # visualization
  plot(QSAR_model)
  model_results <- neuralnet::compute(QSAR_model, QSAR_test[1:6])
  predicted_strength <- model_results$net.result
  output <- cor(predicted_strength, QSAR_test$V7)
  print(output)
}
```

```
         [,1]
[1,] 0.745825
          [,1]
[1,] 0.8044063
          [,1]
[1,] 0.7771851
         [,1]
[1,] 0.763678
         [,1]
[1,] 0.759999
```

- It can be seen that the performance of the single-layer perceptron is hidden node = 2, with corelation =0.80
- We can then visualize the network topology using the plot() function on the resulting model

object:



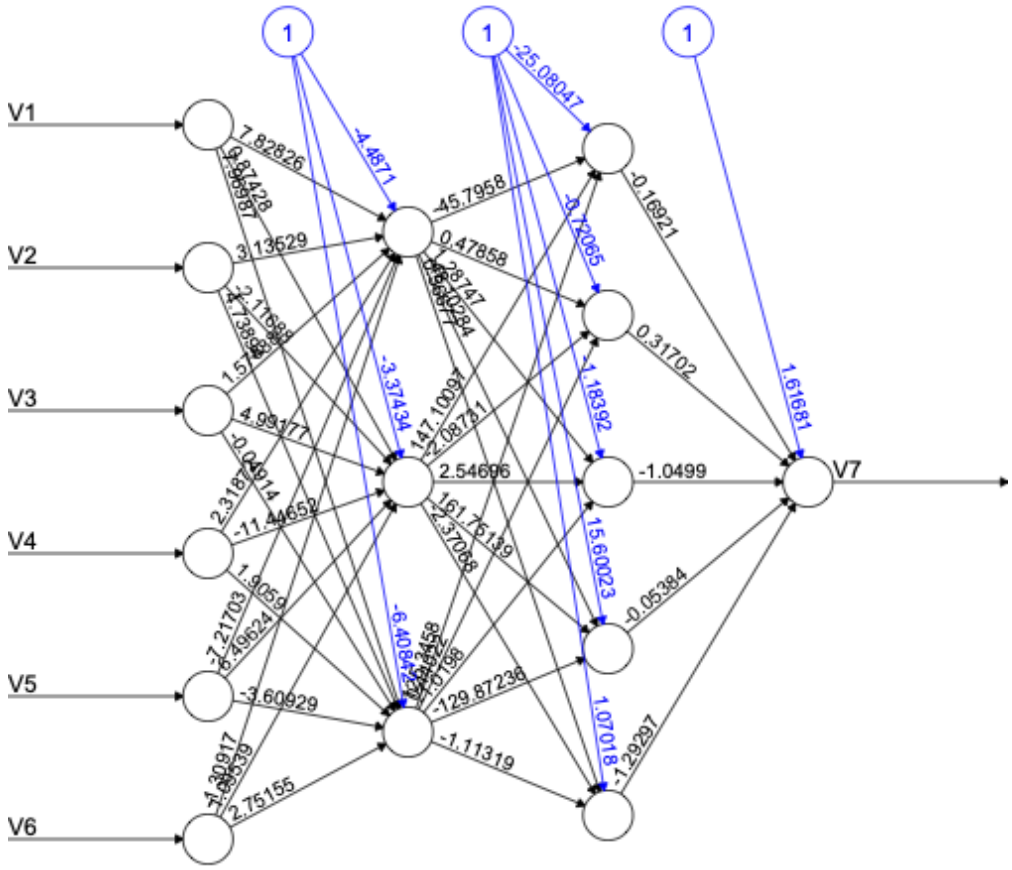
3.4 Step 4 – increase layers to improve model performance

- Then, we try to increase the hidden layers to 2, nodes from (2,1) to (4,5).

[18]:
```
# neuralnet modeling, try hidden layers = 1-5
library('neuralnet')
for (i in c(2,3,4)) {
  for (j in c(1,2,3,4,5)){
      hidden <- c(i,j)
      QSAR_model <- neuralnet(V7 ~ V1 + V2 + V3 + V4 +V5 + V6,
                          data = QSAR_train, hidden = c(2,hidden))
      model_results <- neuralnet::compute(QSAR_model, QSAR_test[1:6])
      predicted_strength <- model_results$net.result
      output <- cor(predicted_strength, QSAR_test$V7)
      cat(i,j)
      print(output)
  }  }
```

```
2 1          [,1]
[1,] 0.7877105
2 2          [,1]
[1,] 0.8005624
2 3          [,1]
[1,] 0.7998047
2 4          [,1]
[1,] 0.7979071
2 5          [,1]
[1,] 0.7854103
3 1          [,1]
[1,] 0.7973076
3 2          [,1]
[1,] 0.7912572
3 3          [,1]
[1,] 0.8108005
3 4          [,1]
[1,] 0.7966013
3 5          [,1]
[1,] 0.7903481
4 1          [,1]
[1,] 0.7909494
4 2          [,1]
[1,] 0.7939436
4 3          [,1]
[1,] 0.7824846
4 4          [,1]
[1,] 0.7923388
4 5          [,1]
[1,] 0.7918751
```

- When hidden layers=2, hidden nodes = (3,5), correlation get the highest level. Plotting the network again, we see a drastic increase in the number of connections. We can see how this impacted the performance as follows:
- Notice that the reported error (measured again by SSE) has been reduced from 3.03 in the previous model to 2.31 here. Additionally, the number of training steps rose from 949 to 7743, which should come as no surprise given how much more complex the model has become. More complex networks take many more iterations to find the optimal weights.
- Applying the same steps to compare the predicted values to the true values, we now obtain a correlation around 0.81, which is a considerable improvement over the previous result of 0.80 with a single hidden layer and node.



Error: 2.313045 Steps: 7743