I have run experiment for all the given libraries (Neuralnet, Deepnet, H2O, MXNET, Tensorflow and KerasR, and TensorFlow and Keras). However, I only success in running the experiment in Neuralnet and TensorFlow and Keras. Below is the report.

For all the experiment, I load the dataset by using the following code.

```
concrete <- read.csv(file = "/cloud/project/No 7/Concrete_Data.csv")
```

This experiment was conducted by using R Studio Cloud.

After loading the data, we can preview it by using the following command.

```
knitr::kable(head(concrete), caption = "Partial Table Preview")
```

The figure bellow shows the result.

cement	slag	ash	water	superplastic	coarseagg	fineagg	age	strength
540.0	0.0	0	162	2.5	1040.0	676.0	28	79.99
540.0	0.0	0	162	2.5	1055.0	676.0	28	61.89
332.5	142.5	0	228	0.0	932.0	594.0	270	40.27
332.5	142.5	0	228	0.0	932.0	594.0	365	41.05
198.6	132.4	0	192	0.0	978.4	825.5	360	44.30
266.0	114.0	0	228	0.0	932.0	670.0	90	47.03

Only six data can be seen because we use head function which help us to show only the first few data.

The next step is to normalize our data by defining new function called ‘normalize’.

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

Then, we can now normalise our data by using the following command.

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

After normalising it, we can once again preview our data.

cement	slag	ash	water	superplastic	coarseagg	fineagg	age	strength
1.000	0.000	0	0.321	0.078	0.695	0.206	0.074	0.967
1.000	0.000	0	0.321	0.078	0.738	0.206	0.074	0.742
0.526	0.396	0	0.848	0.000	0.381	0.000	0.739	0.473
0.526	0.396	0	0.848	0.000	0.381	0.000	1.000	0.482
0.221	0.368	0	0.561	0.000	0.516	0.581	0.986	0.523
0.374	0.317	0	0.848	0.000	0.381	0.191	0.245	0.557

The next step is to split our data into train and test data by using the following command.

```
#training set
concrete_train <- concrete_norm[1:773, ]

#test set
concrete_test <- concrete_norm[774:1030, ]
```

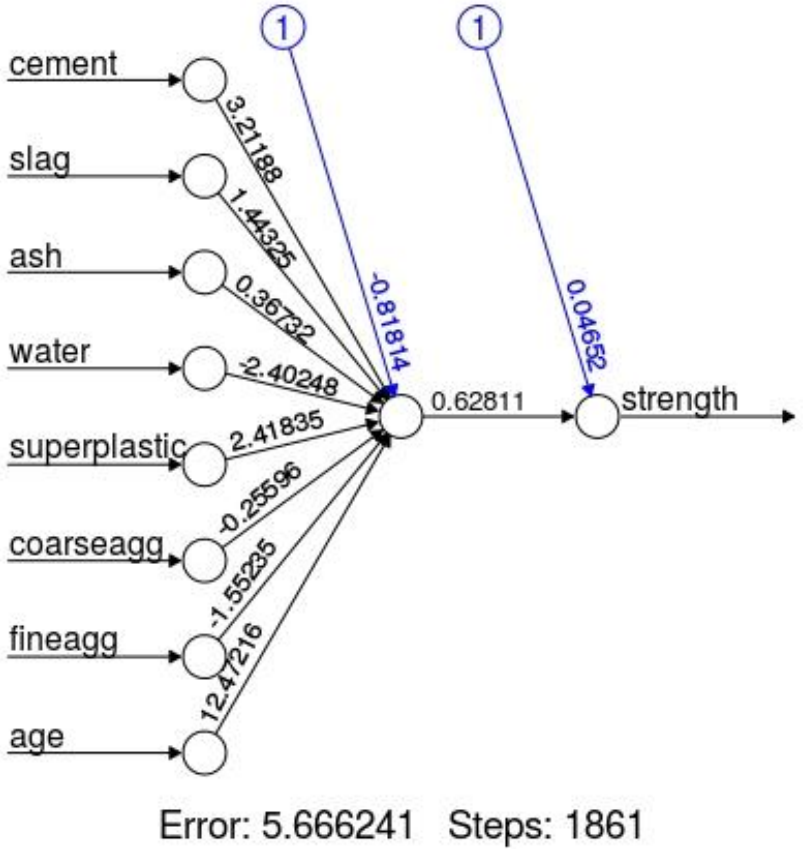
After following all of the above step, for the neuralnet library, we can just start to build our neural network model by using the following command.

```
concrete_model <- neuralnet(strength ~ cement + slag + ash + water + superplastic +
coarseagg + fineagg + age , data = concrete_train, hidden = 1)
```

We can also plot our model by using the following command.

```
plot(concrete_model)
```

The result is as shown bellows.



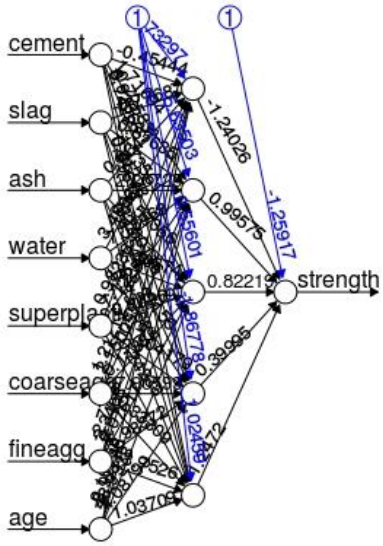
We can now build a predictor and evaluate the result.

```
#building the predictor, exclude the target variable column
model_results <- compute(concrete_model, concrete_test[1:8])

#store the net.results column
predicted_strength <- model_results$net.result

cor(predicted_strength, concrete_test$strength)
```

When we use only 1 neuron in our hidden layer, we will get around 72.7% accuracy; however, if we tried to use 5 neurons as depicted by the following figure,



We will get approximately 75.3% accuracy.

Next, I tried to use deepnet package. Here, we need to convert our data into matrix by using the following command.

```
X <- as.numeric(as.matrix(concrete_train[,1:8]))
X = matrix(as.numeric(X),ncol=8)
head(X)

Y <- as.matrix(concrete_train["strength"])
head(Y)
```

When I printed the result, we can see that there is no problem with this step.

```
> X <- as.numeric(as.matrix(concrete_train[,1:8]))
> X = matrix(as.numeric(X),ncol=8)
> head(X)
          [,1]      [,2] [,3]      [,4]       [,5]      [,6]
[1,] 1.0000000 0.0000000    0 0.3210863 0.07763975 0.6947674
[2,] 1.0000000 0.0000000    0 0.3210863 0.07763975 0.7383721
[3,] 0.5262557 0.3964942    0 0.8482428 0.00000000 0.3808140
[4,] 0.5262557 0.3964942    0 0.8482428 0.00000000 0.3808140
[5,] 0.2205479 0.3683918    0 0.5607029 0.00000000 0.5156977
[6,] 0.3744292 0.3171953    0 0.8482428 0.00000000 0.3808140
          [,7]       [,8]
[1,] 0.2057200 0.07417582
[2,] 0.2057200 0.07417582
[3,] 0.0000000 0.73901099
[4,] 0.0000000 1.00000000
[5,] 0.5807827 0.98626374
[6,] 0.1906673 0.24450549
> Y <- as.matrix(concrete_train["strength"])
> head(Y)
   strength
1 0.9674847
2 0.7419958
3 0.4726548
4 0.4823720
5 0.5228603
6 0.5568706
>
```

This is exactly the same with the original data. However, when I build, my NN model, train it and evaluate it by using the following command,

```
nn <- nn.train(X, Y, activationfun = "sigm", output="linear", hidden = c(5))

X_test <- as.numeric(as.matrix(concrete_test[,1:8]))
X_test = matrix(as.numeric(X_test),ncol=8)

Y_test <- as.matrix(concrete_test["strength"])

y_pred = nn.predict(nn, X_test)
head(y_pred)

nn.test(nn, X_test, Y_test)
```

I got only around 18.9% accuracy on the test data. This terrible result is, in my opinion, because of the activation function. For this regression problem, I tried to assign linear as its activation function (According to the documentation,

activationfun	activation function of hidden unit.Can be "sigm","linear" or "tanh".Default is "sigm" for logistic function

)

However, when I tried to assign linear as the activation function of hidden unit, this is the result :

```
> nn <- nn.train(X, Y, activationfun = "linear", output="linear", hidden = c(5))
Error in nn.ff(nn, batch_x, batch_y, s) : unsupport activation function!
```

As can be seen, it forced me to use only either sigmoid or tanh. Therefore, my conclusion is that this unwanted result is due to the activation function problem.

Next, I run my experiment by using H2O library. For H2O, the following code was used.

```
library(h2o)
localH2O = h2o.init(ip="localhost", port = 54321,
                    startH2O = TRUE, nthreads=-1)

concrete_train <- h2o.importFile("/cloud/project/No 7/Concrete_Data.csv")
concrete_test <- h2o.importFile("/cloud/project/No 7/Concrete_Data.csv")

y = names(concrete_train)[9]
x = names(concrete_train)[1:8]

concrete_train[,y] = as.factor(concrete_train[,y])
concrete_test[,y] = as.factor(concrete_test[,y])

model = h2o.deeplearning(x=x,
                         y=y,
                         training_frame=concrete_train,
                         activation="Rectifier",
                         hidden = c(5,5),
                         l1 = 1e-5,
                         epochs = 50)

print(model)
```

Everything works fine; however, the result is also terrible.

```
Model Details:
==============

H2OMultinomialModel: deeplearning
Model ID:  DeepLearning_model_R_1586679654636_1
Status of Neuron Layers: predicting strength, 845-class classification, multinomial distributi
tropy loss, 5,145 weights/biases, 74.3 KB, 51,500 training samples, mini-batch size 1
  layer units      type dropout       l1       l2 mean_rate rate_rms
1     1     8     Input  0.00 %       NA       NA        NA       NA
2     2     5 Rectifier  0.00 % 0.000010 0.000000  0.000838 0.000326
3     3     5 Rectifier  0.00 % 0.000010 0.000000  0.001597 0.001934
4     4   845   Softmax      NA 0.000010 0.000000  0.403238 0.387930
  momentum mean_weight weight_rms mean_bias bias_rms
1       NA          NA         NA        NA       NA
2 0.000000    0.207664   0.392314  0.744299 0.358363
3 0.000000   -0.282799   0.494429  1.024674 0.347343
4 0.000000   -2.882476   1.540614 -7.379771 1.573250


H2OMultinomialMetrics: deeplearning
** Reported on training data. **
** Metrics reported on full training frame **

Training Set Metrics:
=====================

Extract training frame with `h2o.getFrame("RTMP_sid_bf6c_5")`
MSE: (Extract with `h2o.mse`) 0.9907903
RMSE: (Extract with `h2o.rmse`) 0.9953845
Logloss: (Extract with `h2o.logloss`) 6.409642
Mean Per-Class Error: 0.9962525
```

The MSE loss is very high.

Next, I tried to use MXNET. The problem with this one is that I cannot install the package on this version of R. it said that R no longer support this package. My code can be found on my GitHub.

For the next experiment, I used tensorflow and kerasR. The problem that I faced with this experiment is that I cannot build the model.

```
> mod <- Sequential()
Error in modules$keras.models$Sequential(layers = layers) :
  attempt to apply non-function
```

My complete code can also be found on my GitHub.

For the last experiment, I used tensorflow and keras library. There is no difference in the data preparation process. In order to build my model, I used the following code.

```
model <- keras_model_sequential()

n_units = 5
model %>%
  layer_dense(units = n_units,
              activation = 'relu',
              input_shape = dim(X)[2]) %>%
  layer_dense(units = n_units, activation = 'relu') %>%
  layer_dense(units = n_units, activation = 'relu') %>%
  layer_dense(units = 1)

summary(model)
```

The result of the model summary is as follows.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 5)                 45
_________________________________________________________________
dense_5 (Dense)              (None, 5)                 30
_________________________________________________________________
dense_6 (Dense)              (None, 5)                 30
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 6
=================================================================
Total params: 111
Trainable params: 111
Non-trainable params: 0
```

After building the model, I train it by using the following code.

```
model %>% compile(
  loss = 'mean_squared_error',
  optimizer = optimizer_adam()
)

model %>% fit(
  X, Y,
  epochs = 50, batch_size = 32, verbose = 1,
  validation_split = 0.1
)
```

The final loss after 50 epochs was 0.0141.