Neural Network Machine Learning Techniques for Large Noisy Data Prediction using R Programming

Yagyanath Rimal
Faculty of Science and Technology
Pokhara University ,Nepal

Abstract: This review paper primarily discusses on neural network machine learning techniques for large data analysis using r programming. Although there is large gap between analysis of research data when there were large set of in associative variables. This paper tries to explore the analysis of predicative modeling procedures steps of large data sets were explained sufficiently to reach conclusion. Its purpose is to explain the simplest way of neural network analysis whose data structure were in scattered using R software whose outputs were sufficiently summarized with various intermediate output and graphical interpretation to get conclusion with example bostonhousing datasets. Therefore, this paper presents easiest way of neural network analysis when data sets with large dimensions and its strengths for data analysis using R programming.

Keywords: Hidden Neuron, Neural Network, over fitted Data, Rectified Linear Unit, Multi-Layer Perceptron

Introduction

The neural network is the science of neurons similar to the neuronal system of the human body, in which each neuron is connected to dendrites that transmit signals to another neuron in the form of electrical impulses; the body refers to those signals and decides which special actions should be taken for such cases. The artificial neuron is the heart of the neural network (Noon, 2013). The activation node always takes input from the dendrites which performs a competitive probability action (Johnson, 2004). Where the multiple input process produces and produces a single output in which all input signals are analyzed progressively until the appropriate decision is taken (Gurney, 2004). The various terms of input as variables whose magnitude is considered as a certain weight denoted as (w) are passed to the summary function which already has a certain fixed distortion, the bias always evaluates and produces both true and false judgment. After obtaining zero or one of the activation functions, the limit is calculated, which is further tested with the real prediction with the value with y. This process is continuously tested with binary, categorical or numerical values of the search variables; This process is known as single-level perception using a single node. Basically, there are three parts of the neural network: input levels, hidden levels and output levels. Input levels always take the input of variables and output levels always output. The middle layer has all the powers to analyze data that takes all the input and produces output for output levels. The terms of the input level as a subscript x (1,2,3,4 ... m) and hidden layer h (1,2,3 ... n) could easily be determined during the design of the model formulation. If a network has multiple hidden levels, it is known as deep learning. Deep learning is a machine learning technique that teaches the computer to do what is natural for human learning through examples taken from previous recordings. Therefore, a neural network is a hardware training and software system modeled after the operation (Lim, 2018).

There are several activation functions, the most popular is the threshold or pass function that passes 1 if the activation with bios is greater than zero, otherwise it goes to zero. The sigmoid or logistic (1/1 + ex) function is a widely used activation function that is best for predicting probability. Another is a hyperbolic tangent sigma based on but over -1 to 1 is similar to the regression values where the gradient is deeper y = (ex-ex) / (ex-ex).). The rectified linear unit (RELU) is the best version of the neural network model; it also produces 1 if x is + ve; otherwise, it produces 0 (Sycorax, 2017). It is popular because of the less expensive architecture and the quicker approach, but requires more iteration of experiments. The desired task output is only obtained when their errors and previous hat values go back to neuronal function, so that errors are minimal in each pass. The next extension is the error feedback process to enter the weight is greatly reduced to the spread of the next time, so the network could easily be adjusted their weight at any time iteration to have minimal errors (errors = errors-1). J (Ø) = 1 / mΣmi-m (ZM-yi) 2, this requires more

computing power, using the descent of the systematic gradient finally finds the minimum rate of learning.

Big Data and neural networks are becoming one of the driving forces of innovation, social promotion and the development of life (Zhang, 2017). It can be clearly demonstrated how neural and large data networks are perfectly combined and reinforced by reviewing the basic concepts and key technology in large quantities, as well as the induction of the neural network research framework. On the one hand, neural networks are able to extract abstract features from raw data (Stephen Notley, 2018). They can combine multiple sources of information, process heterogeneous data and acquire dynamic changes in data (Haluk Demirkan, 2015). They are the bridge for the implementation of the transformation of the value of big data. On the other hand, a large volume of big data provides huge examples of training that allow the training of neural networks with a large number of parameters. However, there are still some problems in the neural network model. In the aspect of neural network research, the structure needs more research and development; The network scale lacks theoretical guidelines; and the learning algorithm is having some inherent problem. In the big data aspect there are also three fundamental scientific problems to guarantee coherence in the high-dimensional dispersed space; implement storage space; and to represent the temporal correlation and implement the big data forecast. More research is urgently needed in this area, including theoretical and practical aspects. In particular, inter domain surveys are important. Research in the area can be carried out in coordination with the understanding of processing large amounts of data in the human brain. It needs a greater combination with cognitive science and neuroscience to solve the fundamental scientific problems in the search for neural networks and Big Data, in order to improve the big data analysis using neural networks (Bruke, 2017). The neural network is a hardware and / or software system modeled on the functioning of neurons in the human brain. Neural networks, also called artificial neural networks, are a variety of deep learning technology, which also falls within artificial intelligence (Rouse, 2018). Although ANN researchers are generally not worried that their networks are similar to biological systems, some have done so. Neural social networks are generally organized in layers. The layers are composed of a series of interconnected "nodes" that contain an "activation function". Patterns are presented to the network through the "input level", which communicates with one or more "hidden levels" in which the actual processing is performed through a system of weighted "connections". The hidden layers are then linked to an "output level" where the response is sent as shown in the graph below. Most RNAs contain some form of "learning rule" that modifies link weights based on the input models presented. In a sense, ANN learns from the example how their biological counterparts do; A child learns to recognize dogs of other pets. Although there are many different types of learning rules used by neural networks, this demonstration only affects one; The delta rule The delta rule is often used by the most common class of RNA called inverse neural networks (BPNN). Backpropagation is an abbreviation for backtrack error propagation.

With the delta rule, as with other types of backward propagation, "learning" is a supervised process that occurs with each cycle or "epoch" through a flow of activation of forward outputs and the propagation of errors towards behind the adjustments. of weight. The number of neurons in the output layer must be directly related to the type of work performed by the neural network. To determine the number of neurons in the output layer, first consider the intended use of the neural network. (Saxena, 2018). In a feedforward network, information moves in one direction - forward - from input nodes, through hidden nodes (if any), and to output nodes. There are no cycles or cycles, there is no specific rule for this, in general, they are empirically determined through a cross-validation methodology (Bouziane, 2018).

Neural Network Using R Programming

Here I m using housing data for 506 data sets of census data of Boston from the 1970 census. The data frame BostonHousing contains the original data of 14 variables as independent additional spatial information (Leisch, 2018).

```
> #bostonhousing
> library(keras)
> library(mlbench)
> library(dplyr)
> library(magrittr)
> library(neuralnet)
> data("BostonHousing")
> data=BostonHousing
> ?BostonHousing # description of data sets
> str(data)
'data. frame':          506 obs. of 14 variables:
 $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.0690
 $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
 $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7
 $ chas   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524
0.524 0.524 0.524
 $ rm     : num  6.58 6.42 7.18 7 7.15 ...
 $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 1
 $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
 $ rad    : num  1 2 2 3 3 3 5 5 5 5 ...
```

$ tax : num 296 242 242 222 222 222 311 311 311 3
$ ptratio: num 15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
$ b : num 397 397 393 395 397 ...
$ lstat : num 4.98 9.14 4.03 2.94 5.33 ...
$ medv : num 24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16
> data%<>% mutate_if(is.factor,as.numeric)

This line convert factor variable into numeric variable neural network works on only numeric variables.

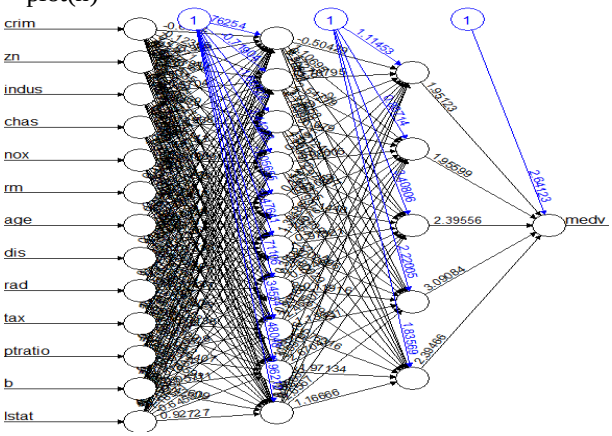
> str(data)
'data.frame': 506 obs. of 14 variables:
$ crim : num 0.00632 0.02731 0.02729 0.03237 0.0690
$ zn : num 18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
$ indus : num 2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7
$ chas : num 1 1 1 1 1 1 1 1 1 1 ...
$ nox : num 0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524
$ rm : num 6.58 6.42 7.18 7 7.15 ...
$ age : num 65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 1
$ dis : num 4.09 4.97 4.97 6.06 6.06 ...
$ rad : num 1 2 2 3 3 3 5 5 5 5 ...
$ tax : num 296 242 242 222 222 222 311 311 311 31
$ ptratio: num 15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2
$ b : num 397 397 393 395 397 ...
$ lstat : num 4.98 9.14 4.03 2.94 5.33 ...
$ medv : num 24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16
>

```
n=neuralnet(medv~crim+zn+indus+chas+nox+rm+age+dis+
rad+tax+ptratio+b+lstat,
data=data,hidden=c(10,5),linear.output=F,lifesign='full',rep=
1)
```

Here the medv is dependent variables of all other 13 variables are independent. The hidden layer are two 10 is first and 5 is in second hidden layer.

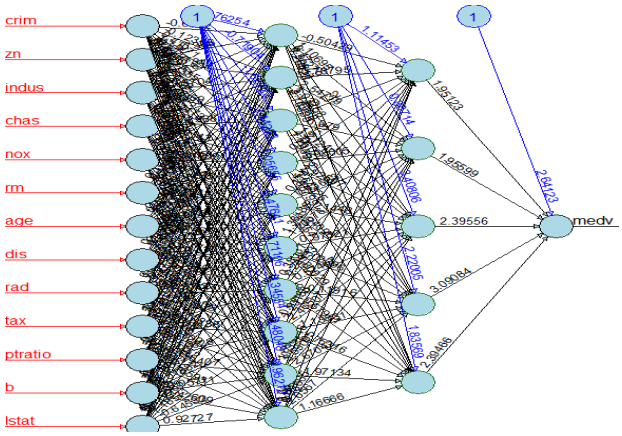
hidden: 10, 5 thresh: 0.01 rep: 1/1 steps: 16
error: 138664.5781 time: 0.02 secs

> plot(n)



Neural variable plot there were 10 and 5 two hidden layer.

```
> plot(n,col.hidden='darkgreen',
+ col.entry.synapse ='red',show.weights = T,
+ information =F,fill='lightblue' )
```



This color plot describes the 13 input variables with numeric values and two hidden layers with 10 and 5 neurons in each layer completely connected network with single bias in each layer is connected with some weight and finally medv is single neuron. There were four layers there were 13 input and single output.

```
> data=as.matrix(data)# converting into matrix
> dimnames(data)=NULL
> set.seed(1234)
> ind=sample(2,n row(data),replace=T,prob=c(.7,.3))
# splitting data
> training=data[ind==1,1:13]
# keeping independent variables
> test=data[ind==2,1:13]
> trainingtarget=data[ind==1,14]
# keeping dependent variable
> testtarget=data[ind==2,14]
```

Normalizing independent variable is calculated using formula (value-mean/standard deviation. Out of 506 data sets test will divided into two independent variables be 151 and training is 355 records parted for model design.

```
> m=colMeans(training)# calculate mean
> m
 [1]   3.6488963099  10.0521126761  10.9879718310
1.0591549296   0.5540053521   6.2853436620
69.0267605634   3.7887416901   9.4140845070
405.3267605634  18.4290140845 358.5856619718 [13]
12.7420563380
> s=apply(training,2,sd)#calculates standard deviation
> s
 [1]   9.0825860903  21.3949137202   6.6116126523
0.2362474194   0.1147606135   0.6790055916
27.9627261879   2.0379791678   8.5579001030
164.2190910798   2.1507528277  89.9415900215  [13]
7.1117072650
> training=scale(training,center=m,scale=s)
> test=scale(test,center=m,scale=s)
```

This process calculates normalization in both training and test samples. Here the point is to consideration is that we use same mean and standard deviation for both samples.

```
install.packages("devtools")
require(devtools)
library(devtools)
devtools::install_github("rstudio/reticulate", force=TRUE)
library(reticulate)
devtools::install_github("r-lib/processx",force=TRUE)
library(processx)
devtools::install_github("rstudio/tensorflow",force=TRUE)
library(tensorflow)
devtools::install_github("rstudio/keras",force=TRUE)
```

```
library(keras)
> model <- keras_model_sequential()
> model %>%
+   layer_dense(units = 5, activation='relu',input_shape =
c(13)) %>%
+   layer_dense(units = 1)# for output
> model %>% compile(loss='mse',optimizer='rmsprop',
+                   metrics='mae')
```

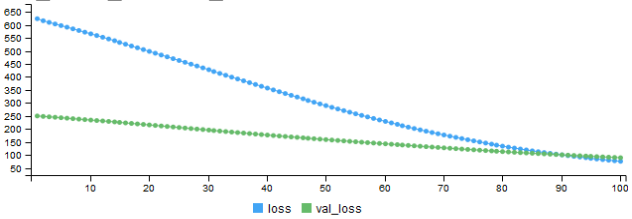
Here this neural network is densely connected to each nodes from one level to another level. Here units 5 represents the number of hidden layer and outputs single neuron with activation function is Rectified Linear Unit where there were 13 variables as input variables. After design model the model is compiled with optimizer is rmsprop and metrics is mean absolute errors.

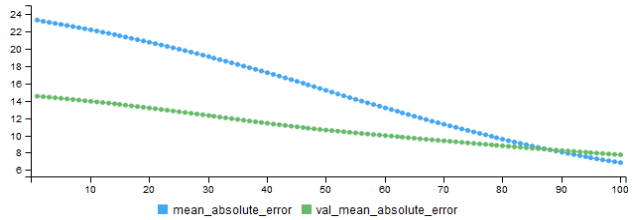
```
> mymodel<-model %>%
+   fit(training,
+       trainingtarget,
+       epochs = 100 ,
+       batch_size = 32 ,
+       validation_split = 0.2)
```

Here the above model is fitting with target and source with epochs 100 with validation split 0.2 which use 20 percent data from training data sets for out of sample errors, which produces following output

```
Train on 284 samples, validate on 71 samples
Epoch 1/100
284/284 [====] - 0s 1ms/step - loss: 656.6251 –
mean_absolute_error: 24.1395 –
val_loss: 309.5449 – val_mean_absolute_error: 16.8285
Epoch 2/100
284/284 [====] - 0s 54us/step - loss: 650.8414 -
mean_absolute_error: 24.0361 - val_loss: 308.5933 -
val_mean_absolute_error: 16.8073

Epoch 99/100
284/284 [====] - 0s 67us/step - loss: 137.8865 -
mean_absolute_error: 9.6698 - val_loss: 244.4070 -
val_mean_absolute_error: 14.6707
Epoch 100/100
284/284 [====] - 0s 85us/step - loss: 135.5420 -
mean_absolute_error: 9.5453 - val_loss: 243.7032 -
val_mean_absolute_error: 14.6438
```





The above figure represents two lines, the lower lines describes the validation loss and above lines describes the mean square errors. Initially there were large gap gradually reduced narrower and converse at 90.

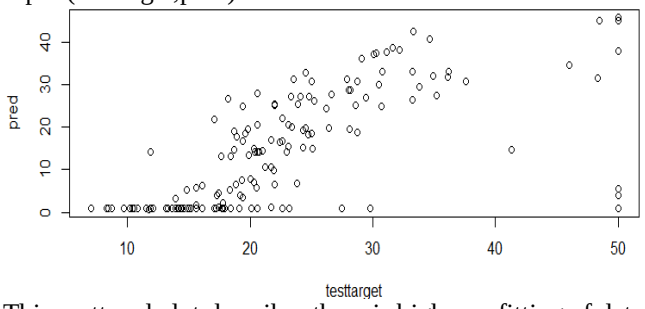
```
> summary(mymodel)
 Length Class  Mode
params  8      -none- list
metrics 4      -none- list
> model %>% evaluate(test,testtarget)
151/151 [=====] - 0s 13us/step
$loss    [1] 202.038932
$mean_absolute_error   [1] 11.52268751
```

The evaluate function express that there is 202 percentage loss and mean absolute errors is 11 percent

```
> pred=model%>% predict(test)
> mean((testtarget-pred)^2)
[1] 202.0389313
```

Which is similar to loss value

```
> plot(testtarget,pred)
```



This scattered plot describes there is high overfitting of data therefore for good prediction the scatter data is in diagonal straight line. Therefore, there is lots of improvement in this model.

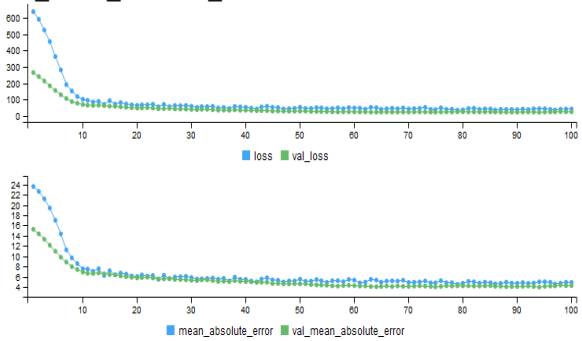
```
>  model <- keras_model_sequential()
> model %>%
+   layer_dense(units = 100, activation='relu',input_shape =
c(13)) %>%
+   layer_dropout(rate=0.4)%>%
+   layer_dense(units = 50, activation='relu') %>%
+   layer_dropout(rate=0.3)%>%
+   layer_dense(units = 20, activation='relu') %>%
+   layer_dropout(rate=0.2)%>%
+   layer_dense(units = 1)
> #layer_activation('softmax')
> summary(model)
```

Here we are adding 100 neurons in first hidden layers 13*100+100 become 1400 neuron similarly in second hidden layer there were 50*13+50 become with dropout of 40 percent and 30 percent and 20 percent in each subsequent which means that 40 percent of components were dropped out to zero.

```
_________________________________________________
Layer (type)    Output Shape    Param #
=================================================
dense_22 (Dense)(None, 100)      1400
_________________________________________________
dropout_7 (Dropout) (None, 100) 0
_________________________________________________
dense_23 (Dense)    (None, 50)       5050
_________________________________________________
dropout_8 (Dropout) (None, 50)       0
_________________________________________________
dense_24 (Dense)    (None, 20)                1020
_____________________________________________
dropout_9 (Dropout) (None, 20)                0
_________________________________________________
dense_25 (Dense)    (None, 1)        21
=================================================
Total params: 7,491
Trainable params: 7,491
Non-trainable params: 0
_________________________________________________
```

```
> model %>% compile(loss='mse',
+           optimizer=optimizer_rmsprop(lr=0.001),
+               metrics='mae')
> #fit model
> mymodel<-model %>%
+   fit(training,
+       trainingtarget,
+       epochs = 100 ,
+       batch_size = 32 ,
+       validation_split = 0.2)
Train on 284 samples, validate on 71 samples
Epoch 1/100
284/284 [====] - 0s 2ms/step - loss: 636.9227 -
mean_absolute_error: 23.5547 - val_loss: 262.7128 -
val_mean_absolute_error: 15.0683
Epoch 2/100
284/284 [====] - 0s 109us/step - loss: 579.2154 -
mean_absolute_error: 22.3369 - val_loss: 232.0326 -
val_mean_absolute_error: 13.8860
Epoch 3/100
284/284 [====] - 0s 55us/step - loss: 509.4533 -
mean_absolute_error: 20.6511 - val_loss: 200.0774 -
val_mean_absolute_error: 12.6300

Epoch 98/100
284/284 [====] - 0s 88us/step - loss: 43.6551 -
mean_absolute_error: 4.8959 - val_loss: 19.7974 -
val_mean_absolute_error: 3.6739
Epoch 99/100
284/284 [====] - 0s 78us/step - loss: 35.2704 -
mean_absolute_error: 4.3979 - val_loss: 19.1389 -
val_mean_absolute_error: 3.5984
Epoch 100/100
284/284 [=====] - 0s 85us/step - loss: 40.1729 -
mean_absolute_error: 4.7151 - val_loss: 19.2505 -
val_mean_absolute_error: 3.6201
```

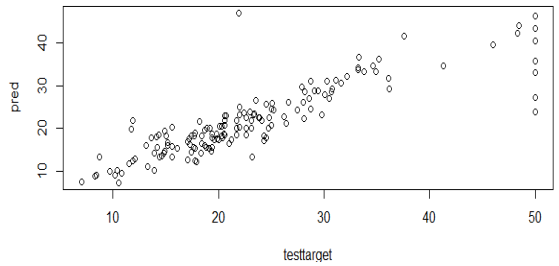


This picture describes there is large drop in overfitting of data when increasing 10 to 100 neuron in first hidden layers

```
> summary(mymodel)
        Length Class  Mode
params  8      -none- list
metrics 4      -none- list
> model %>% evaluate(test,testtarget)
151/151 [=====] - 0s 0us/step
$loss [1] 24.59003024
$mean_absolute_error [1] 2.937888456
```

The loss and mean square errors were significantly reduced from 202 to 24 percentage and mean absolute errors is nearly to 3 percent.

```
> pred=model%>% predict(test)
> mean((testtarget-pred)^2)
[1] 24.59003043
> plot(testtarget,pred)
```



This scattered plot also has largely improvement in the model whose values are more or less in diagonal implies improvement which could further improvement with using learning rate of 0.001.

Conclusion

A neural network is a powerful computational data model capable of capturing and representing complex input / output relationships. Neural networks are the most successful methods for analyzing large amounts of data. The simulation of the neural structure in the brain to construct neural network structure models and the simulation of a memory mechanism in the brain to develop learning algorithms are two basic methodologies in the investigation of neural networks. Although the neuronal network has less interpretability than the decision tree. But it is more suitable for noisy datasets for unrecognized patterns. Deep neural networks are powerful types of artificial neural networks that use different hidden levels; It has a versatile application in modern society for its superior predictive properties, including robustness and over-tightening. However, its application to algorithmic commerce has not been previously studied, partly due to its computational complexity. This paper describes the general description of the neural network prediction using boston multilayer data sets concealed to predict the dependent variable. Neural networks have been very successful in several model recognition applications in modern society. Therefore, the loss and mean square errors were significantly reduced from the 506 data sets of census data of Boston from the 1970 census with from 202 to 24 percentage and mean absolute errors is nearly to 3 percent and Therefore, over fitted data in any data base can be easily summarized using neural network model with adding hidden layers in-between input to output variables.

References

[1] Bouziane, A. (2018). *Neural network analysis* .
[2] Bruke, j. (2017). *Big data analysis using neural networks.*
[3] Gurney, K. (2004). An introduction to Neural Network. *UCL Press Limited is an imprint of the Taylor & Francis Grou*.
[4] Haluk Demirkan, C. B. (2015). Innovations with Smart Service Systems: Analytics, Big Data, Cognitive Assistance, and the Internet of Everything.
[5] Johnson, S. a. (2004). Neural Coding Strategies and Mechanisms of Competition. *Cognitive Syatems Research*.
[6] Leisch, F. (2018). *BostonHousing R Data sets of ML Bench data sets.*
[7] Lim, S. (2018). Adaptive Learning Rule for Hardware based Deep Neural Networks .
[8] Noon, H. (2013). Artificial Neural Network : Beginning of the AI revolution.
[9] Rouse, M. (2018). *Big_data_analysis_using_neural_networks.*
[10]Saxena, S. (2018). *Becoming Human: Artificial Intelligence Magazine.*
[11] Stephen Notley, M. M.-I. (2018). Examining the Use of Neural Networks for Feature Extraction: A Comparative Analysis using Deep Learning, Support Vector Machines, and K-Nearest Neighbor Classifiers.
[12]Sycorax, P. F. (2017). How does the Rectified Linear Unit (ReLU) activation function produce non-linear interaction of its inputs.
[13] Zhang, Y. (2017). Big data analysis using neural networks.