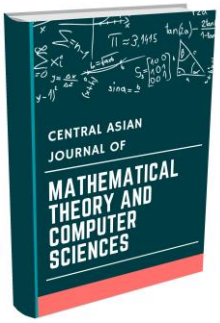


CENTRAL ASIAN JOURNAL OF MATHEMATICAL THEORY AND COMPUTER SCIENCES

http://cajmtcs.centralasianstudies.org/index.php

Volume: 01 Issue: 01 | Sep 2020

Multi-layer neural network perception for large data prediction using R Programming

Yagyanath Rimal

[1]Faculty of Science and Technology, Pokhara University, Nepal
Corresponding E-mail: rimal.yagya@pu.edu.np

ABSTRACT

This review article primarily focus on neural network perceptionitechniques for large data analysis using r programming on secondary data set. Although there is a large confusion while selctioning an appropriate analysis tools for the newer researchersr when there was a large set of associative variables like CTG data sets having 2126 observations in 22 attributing variables of numeric types. The last variable NSP with categorical type 1,2 and 3 stores best, mediam and least properties. This article attempts to explore the detail analysis using the concept of multi-ilayered neural networks procedures for sample CTG data sets where each step to reach conclusion were sufficiently explained. Its purpose is to explain the simplest form of analysis when researcher meets large number of dependent and independent varialbes research data structure using software R whose results are summarized and inferenced with sufficiently with interemediate results and conclusions were drawn using appropriate graphical interpretation were analyze using 8 hidden layers with activation of relu of 21 independent variables and loss is calculated with different epoch of categorical crossentropy. The each model value is evaluated with confusion matrixed prediction. The model is considered as best fit when the two-line graphs were in constant in more iterations. The tenser flow package is widely used in image recognition, computer vision, speech / sound recognition and time series analysis and many more.

ARTICLE INFO

Article history:
Received 10 July 2020
Received in revised form 24 July 2020
Accepted 31 August 2020

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

1.INTRODUCTION

Big data and neural networks are becoming unique notion of electronic media innovation in modern development (Zhang, 2017). On the one hand, neural networks can extract abstract features from large raw data sets (Stephen Notley, 2018). They can combine multiple sources of information, process heterogeneous data and acquire dynamic changes in data (Haluk Demirkan, 2015) which transform the value of big data. Machine learning needs a greater combination with cognitive science and neuroscience for solving the fundamental scientific problems in order to improve the analysis of big data using neural networks (Bruke, 2017). The neural network is a hardware / 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). The rule is types of backward propagation, learning mechanism of supervised process that occurs with each cycle called as "epoch" through which data flow to activation forward outputs and the propagation of errors towards behind the adjustments of its weight inputs. The number of neurons in the output layer must be directly related to the type of work performed by the neural network. That determine the number of neurons in the output layer (Saxena, 2018). In a feedforward network, information moves in one direction forward from input nodes to hidden nodes and to output nodes. Which are empirically determined through a cross-validation methodology (Bouziane, 2018). Neural networks process of estimating the results is technically known as direct propagation. Then we compare the result with the actual output. The task is to make the output of the neural network approach for actual output. In sometimes each of these neurons is contributing an error to the final output. The error terms are calculated by minimizing the value / weight of the neurons that contribute most to the error and this occurs when returning to neurons in the neural network and where the error is found, this process is known as backward

propagation. To reduce this number of iterations to minimize the error, neural networks use a common algorithm known as gradient descent, which allows you to optimize your activity quickly and efficiently. The multi-layer perceptron is the basic training unit of a neural network, it is a perceptron that requires more input and produces output. Which were calculated, input was directly combined and output was calculated based on a threshold value that adds weights to inputs and adds a bias in each perceptron also has a bias that can be considered flexible. The perceptron is somewhat similar to the constant b of a linear function y = ax + b. It allows us to move the line up and down to better adjustment of forecast with the data. A neuron applies non-linear transformations to inputs and biases. The activation function takes the sum of the weighted input (w1 * x1 + w2 * x2 + w3 * x3 + 1 * b) as an argument and returns the output of the neuron.

$$a = f(\sum_{i=0}^{N} w_i x_i)$$

There are sigmoid, tan, relu activation functions use while forward propagation process when output is actual output. In neural network we update the biases and the weights depending on the error adjustment. This process of weight and bias updating is known as backward propagation. The backward propagation algorithms work by determining the loss of output and propagating it back to the network. The weights are updated to reduce the resulting error of each neuron. The first step to reduce the error is to determine the gradient of each node then final result again. This iteration of forward and backward propagation is known as an iteration of training also known as Epoch.

The multilayer perceptron consists of several layers called the hidden layers stacked between the input layer and the output layer. In the output layer we prediction yes or no of binary classification problem target. We could also have two neurons to predict each of the two classes. When we calculate the multiple layers, we initialize weights and biases with random values, as a weight matrix in the hidden layer, wout as a weighting matrix in the output layer and bout as a deviation matrix in the output layer were calculated in the same way, the product of the input matrix and the weights assigned to the edges of the hidden layer. For activation function will return the output as 1 / (1 + exp (-x)) in each node then apply an activation function (sigmoid is again used) to predict the output process propagation forward. After comparing the prediction with the actual output and calculating the error (Real - Prediction). The error is the mean square loss = ((Y-t) ^ 2) / 2 the value is calculated. Similarly, calculate the slope / gradient of the exit layer neurons. The sigmoid gradient can be returned as x * (1 - x). Then, we calculate the change factor (delta) in the output layer, depending on the error gradient multiplied by the activation slope of the output layer. In this step, the error will propagate back to the network, which means an error in the hidden layer. Calculate the change factor (delta) in the hidden layer, multiply the error in the hidden layer. Update weights in the output and the hidden layer: the weights in the network can be updated based on the errors calculated for the training examples.Th elearning rate is the amount of weights that are updated with biases in the output and the hidden layer.The neural network is the science of neurons similar to the neuronal system of the human body, in which each neuron is connected to the dendrites that transmit signals to another neuron in the form of electrical impulses, the body refers to signals and decides what special actions should be taken for such cases. The artificial neuron is the heart of the neural network (Noon, 2013). The activation node always receives information from the dendrites that perform a competitive probability action (Johnson, 2004). Where the input signals are analyzed progressively until the appropriate decision is made (Gurney, 2004). Therefore, a neural network is a software system and hardware training modeled after the operation (Lim, 2018).

Tensor flow is an open source numerical calculation library provided by google for the intelligence of the machines. It hides all the programming necessary to build deep learning models and offers developers interface to program. The Keras API for tensor flow provides a high-level interface for neural networks (Shwe, 2018). Deep learning libraries are now available with R and a developer can easily download tensor flow which is similar way to other R libraries. In tensor flow, the graph nodes represent mathematical operations, while the edges of the graph represent the multidimensional data arrays communicated to each other is now available in the public domain. Tensor flow uses GPU processing if properly configured. The flexible architecture allows you to implement processing on one or more CPUs or GPUs on a desktop, server or mobile device with a single API. (McAteer, 2018)Tensor flow was originally developed by researchers organization with the purpose of performing machine learning and searching for deep neural networks, but the system is fairly general to be applicable in a wide variety of other good domains.

2. NEURAL NETWORK USING R PROGRAMMING

```
> library(keras)
> install_karas()
> library(keras)
> library(tensorflow)
> data=read.csv(file.choose(),header =TRUE)
```

Here we are using secondary data sets CTG data base of 2126 observations in 22 variables attributes where all data row are in numeric types. The last variable NSP with categorical type 1,2 and 3. The NSP is dependent variable of other 21variables is independent variable use for prediction to its multi linear multi class data base as sample data to explore multi linear perception of deep neural network data analysis purpose.

```
> str(data)
'data. frame':      2126 obs. of 22 variables:
 $ LB    : int  120 132 133 134 132 134 134 122 122 122 ...
$ NSP    : int  2 1 1 1 1 3 3 3 3 3 ...
> data=as.matrix(data)# change data to matirx
> dimnames(data)=NULL # remove default names of caribles
> data[,1:21]=normalize(data[,1:21])# normalized data using keras
> data[,22]=as.numeric(data[,22])-1# Making NSP new category of 012
```

```
> summary(data)# description of all variables with new
names
  V1        V2
Min.:0.2894   Min. :0.000e+00       1st Qu.:0.3747 1st
Qu.:0.000e+00
Median :0.3862   Median :4.572e-06   Mean   :0.3833
Mean   :8.915e-06
3rd Qu.:0.3949   3rd Qu.:1.596e-05   Max.   :0.4537   Max.
:5.983e-05
     V21               V22
 Min.   :-0.0035625   Min.   :0.0000              1st Qu.:
0.0000000   1st Qu.:0.0000
 Median : 0.0000000   Median :0.0000 Mean   : 0.0009039
Mean   :0.3043
 3rd Qu.: 0.0027577   3rd Qu.:0.0000          Max.   :
0.0035719   Max.   :2.0000
> set.seed(1234) # data partition with 70/30
> ind=sample(2,nrow(data),replace=T, prob=c(0.7,0.3))
> training=data[ind==1,1:21]# first 21 columns
> test=data[ind==2,1:21]
```

Training includes 1523 records and column includes 603 records

```
> trainingtarget=data[ind==1,22] #NSP
> testtarget=data[ind==2,22]
```

One hot encoding method is the process of converting single column into many columns of categorical variables with its null values in rest dummy variables is known as one hot encoding.

```
> trainlebels=to_categorical(trainingtarget)
> testlabels=to_categorical(testtarget)
> print(testlabels)
      [,1] [,2] [,3]
 [1,]    1    0    0              [2,]    1    0    0              [3,]    1
0    0
 [4,]    0    0    1              [5,]    0    0    1              [6,]    0
1    0
> model=keras_model_sequential()
> model %>%
+   layer_dense(units = 8,activation = 'relu',input_shape =
c(21))%>%
+   layer_dense(unit=3,activation = 'softmax')
```

Here we are using keras packages to design model each layer was fully connected therefore layer_dense is use to design model where first case we were using 8 hidden layers with activation of relu of 21 independent variables. Similarly, in the second layer we are using 3 hidden layers where activation is softmax which keep ranges between 0 and 1 probabilities.

```
> summary(model)
Layer (type)   Output Shape   Param #
dense_1 (Dense) (None, 8)    176             dense_2 (Dense)
(None, 3)    27
Total params: 203                    Trainable params: 203
            Non-trainable params: 0
```

Here in first hidden layer there were 21*8+8=176 neurons and similarly another output layer there were 3*8+3=27 neuron. All together there were 203 neurons while designing model compilation process the loss is categorical cross entropy function is use because there were three categorical type in response variable in case binary response variable, we use binary_crossentropy model. Adam is optimizer and matrix is accuracy.

```
> model%>%
+ compile(loss='categorical_crossentropy',
optimizer='adam',      metrics='accuracy')
```

Then fitting the model with target variable and trainlebels having 200 times inebriation with 32 batch size and validation split of 0.2 (20 percent data).

```
> history= model %>%
+   fit(training,   trainlebels,   epoch=200,
batcch_size=32,     validation_split=0.2)
Train on 1218 samples, validate on 305 samples
Epoch 1/200
1218/1218 [==] - 1s 958us/step - loss: 1.1572 - acc:
0.1626 - val_loss: 1.1464 - val_acc: 0.0098
Epoch 2/200
1218/1218 [===] - 0s 62us/step - loss: 1.0528 - acc:
0.2537 - val_loss: 1.0288 - val_acc: 0.8164
……..
Epoch 198/200
1218/1218 [===] - 0s 68us/step - loss: 0.3638 - acc:
0.8588 - val_loss: 0.7235 - val_acc: 0.7639
Epoch 199/200
1218/1218 [===] - 0s 71us/step - loss: 0.3638 - acc:
0.8612 - val_loss: 0.7180 - val_acc: 0.7639
Epoch 200/200
1218/1218 [===] - 0s 66us/step - loss: 0.3632 - acc:
0.8629 - val_loss: 0.7228 - val_acc: 0.7639
```

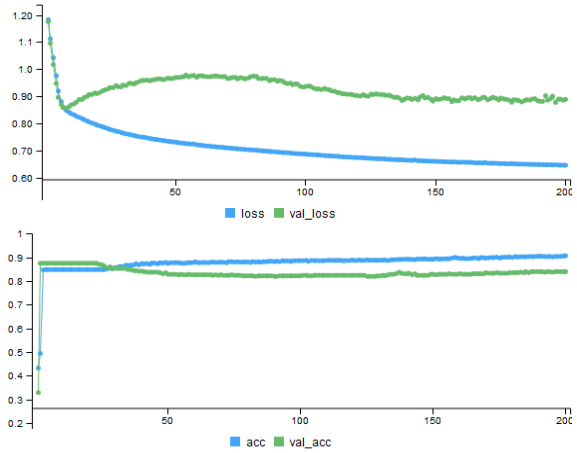


Fig: Loss Line Graph

From the above figure first graph demonstrate loss based on training data and which demonstrate similarity if loss is upwarded it produces overfitting. Similarly, the second graph demonstrates the accuracy which is linear initially when 30 to 40 it becomes constant. Which concludes that after100 iteration it become stabilized gradually.

```
> plot(history)
```

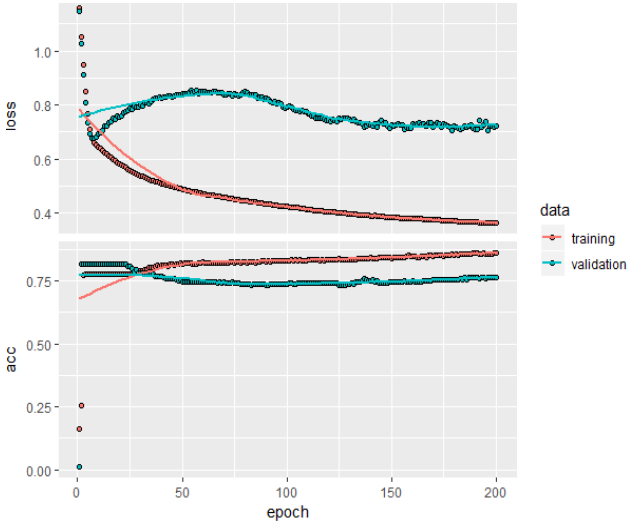


Fig: Static plot

This line graph demonstrates static plot of loss and accuracy. While model is running there is output 1218 indicates that 1523*.8 (80 percent of total data in training).

```
> model%>%
+   evaluate(test, testlabels)
603/603 [==] - 0s 3us/step
$`loss`   [1] 0.4454212
$acc      [1] 0.8441128
```

This model evaluates test data and test labels the loss is lower and accuracy is higher is better in the test data sets.

```
> prob=model%>%
+   predict_proba(test)
> pred=model%>%
+   predict_classes(test)
> table1=table(Predicted=pred,Actual=testtarget)
> table1
      Actual
Predicted   0   1   2
      0 437  39  21
      1  22  53   9
      2   1   2    19
```

The confusion matrix between prediction on test and prediction on test classes demonstrate misclassification is quite high in second category where there were 39 miss classification.

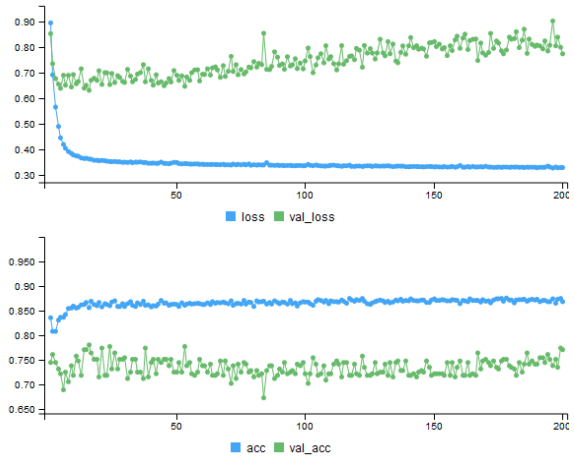
```
> cbind(prob,pred,testtarget)
                          pred testtarget
 [1,] 0.9822145104 0.01717662 0.0006088500    0         0
          [2,] 0.9770679474 0.02199711 0.0009349044
0         0
 [3,] 0.9627258778 0.03376918 0.0035049950    0         0
          [4,] 0.0009644828 0.32539266 0.6736428738
2         2
 [5,] 0.0008492609 0.31680214 0.6823486090    2         2
          [6,] 0.8990210295 0.07724921 0.0237296913
0         1
[21,] 0.0481427945 0.64524496 0.3066122532    1         1
          [22,] 0.9821509123 0.01708566 0.0007634280
0         0
[87,] 0.0110401763 0.53466856 0.4542911947    1         2
          [200,] 0.0152891139 0.56129420 0.4234166443
1         2
```

The cbind function demonstrate probability between predict and test categorical of all iteration model. The first prediction is accurately. Similarly, the 2 and 2 demonstrates both prediction and target category has classified accurately with third category. The sixth model is wrong because test target is second category whereas prediction as first category and so on.

```
> model1 %>%
+   layer_dense(units = 50,activation = 'relu',input_shape =
c(21))%>%
+   layer_dense(unit=3,activation = 'softmax')
> model1%>%
+   compile(loss='categorical_crossentropy',
optimizer='adam',      metrics='accuracy')
> summary(model1)
Layer (type)  Output Shape     Param #
dense_3 (Dense) (None, 50)   200                          dense_4
(Dense) (None, 3)     153
Total params: 353                    Trainable params: 556
            Non-trainable params: 0
> history= model1%>%
+   fit(training,      trainlebels,      epoch=200,
batcch_size=32,      validation_split=0.2)
Train on 1218 samples, validate on 353 samples
Epoch 1/200
1218/1218 [===] - 0s 191us/step - loss: 0.8951 - acc:
0.8358 - val_loss: 0.8530 - val_acc: 0.7443
Epoch 2/200

Epoch 199/200
1218/1218 [===] - 0s 79us/step - loss: 0.3279 - acc:
0.8752 - val_loss: 0.7996 - val_acc: 0.7738
Epoch 200/200
1218/1218 [===] - 0s 81us/step - loss: 0.3278 - acc:
0.8686 - val_loss: 0.7739 - val_acc: 0.7705
> model3 %>%
+   evaluate(test,testlabels)
> prob=model%>%
+   predict_proba(test)
> pred=model%>%
+   predict_classes(test)
> table3=table(Predicted=pred,Actual= testtarget)
> table1
            Actual
Predicted       0   1   2
                      0 437  39  21
                      1  22  53   9
                       2   1   2   19
> model1 <pointer: 0x0>
> table2
      Actual
Predicted       0   1   2
                      0 454  35  7
                     1  32  58   16
                     2   3   8    30
Model3 %>%
```

```
+ layer_dense(units = 50,activation = 'relu',input_shape = c(21))%>%
  Layer_dense(units=8, activation='relu')%>%
+ layer_dense(unit=3,activation = 'softmax')
> model3%>%
+ compile(loss='categorical_crossentropy', optimizer='adam',     metrics='accuracy')
> summary(model3)
```



Model3

	Actual		
Predicted	0	1	2
0	433	45	11
1	22	45	5
2	5	4	33

The model two has better result than other model of confusion matrix.

Therefore large data sets of many variables were reduced largely using machine learnnig adding hidden layers using artifical neural network to get accurate fine tune in large data sets.The keras use tensorflow as backend analyzed large deep learning data sets are the modern demanding data analysis package are being useing by large company linke google, microsoft etc.

CONCLUSIONS

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 of 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 methods in the field of neural network research. Although the neuronal network has less interpretability of the decision tree. But it is better suited to 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 force and over tighten. However, its application to algorithmic negotiation has not been studied, partly because of its computational complexity. This article describes the vision of the multilayer neural network prediction using hidden multi-layer data sets to predict the dependent variable. Neural networks have been very successful in several model recognition applications in modern society.

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] Lim, S. (2018). Adaptive Learning Rule for Hardware based Deep Neural Networks.

[7] McAteer, M. (2018). An introduction to probabilistic programming, now available in TensorFlow Probability.

[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] Shwe, M. (2018). An introduction to probabilistic programming, now available in TensorFlow Probability. Retrieved from https://medium.com/tensorflow/an-introduction-to-probabilistic-programming-now-available-in-tensorflow-probability-6dcc003ca29e?fbclid=IwAR2tP68gZEXfGBs_Q5vYVZUZxICcv8nRkvpkn8n3AsHN2ZlRglij55ngSzw

[12] 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.

[13] Zhang, Y. (2017). Big data analysis using neural networks.