Building Neural Network Model in BASE SAS® (From Scratch)

Kennesaw State University
Center for Statistics and Analytical Research

Soujanya Mandalapu
Department of Statistics & Analytical Sciences

Advisor: Dr. Sherry Ni

Kennesaw State University
College of Science and Mathematics
Department of Statistics and Analytical Sciences

INTRODUCTION

Artificial neural networks are statistical learning models, inspired by biological neural networks (central nervous systems, such as the brain), that are used in machine learning. These networks are represented as systems of interconnected "neurons", which send messages to each other. The connections within the network can be systematically adjusted based on inputs and outputs, making them ideal for supervised learning. ANNs are heavily used in image and speech recognition and their applications are reflected in other fields such as weather prediction, financial and transportation.

In this project, we used real dataset provided by a financial company. The company requested us to build an algorithm for Neural Network in Base SAS. This data contains 5000 observations and 500+ attributes.

METHODS

Training a neural network basically means calibrating all of the "weights" by repeating two key steps, forward propagation and back propagation. In forward propagation, we apply a set of weights to the input data and calculate an output. For the first forward propagation, the set of weights is selected randomly. In back propagation, we measure the margin of error of the output and adjust the weights accordingly to decrease the error. Neural networks repeat both forward and back propagation until the weights are calibrated to accurately predict an output. Each repetition of forward and back propagation to adjust weights is called an iteration or an epoch.

In this project we developed code in SAS (Macros) to implement standard neural network with one hidden layer (Fig 1). Three separate macros were created for forward propagation, backward propagation and to control the iterations (epochs). In this model, we used sigmoid activation function to transform the input signal into an output signal and weights are adjusted with learning rate 0.1. With the random weights, first observation was forward propagated, error was calculated and weights were adjusted through backward propagation. The adjusted weights of the first observation were transferred to second observation and the process is repeated until the last observation (Fig 2 & Fig 3). Once the weights are updated for last observation, those weights are used to forward propagate on train and test data to generate ASE and misclassification rate. This whole process is termed as one iteration. The weights of first iteration are fed into the second iteration and the process is repeated until the desired number of iterations. Running iterations in batches is another unique feature in this algorithm. This feature enables us to stop and start the iterations as needed without sacrificing the time.

Standardizing the data and shuffling the input data are two steps that are critical for this algorithm to perform as expected. This model is scalable to increase the input variables and any number of hidden nodes. Hundred iterations were run on train and test data and the accuracy, misclassification rate and ASE were compared with the same model ran in Python (Fig 4).

Implementing Neural Network Algorithm

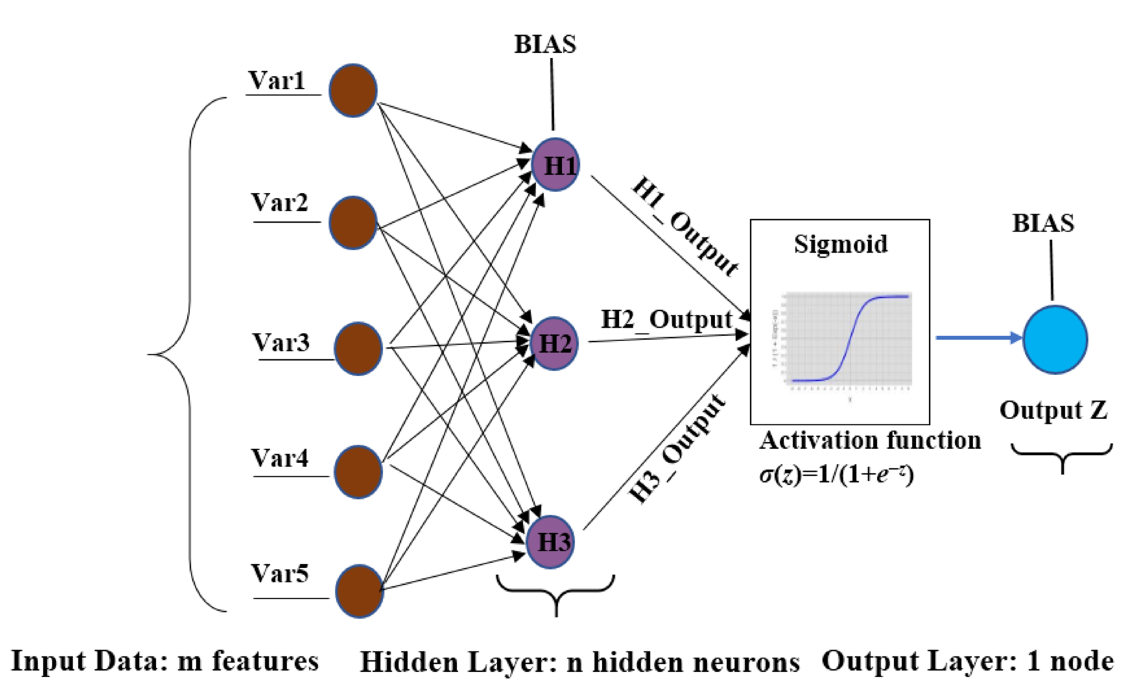

Fig 1. Neural Network Architecture

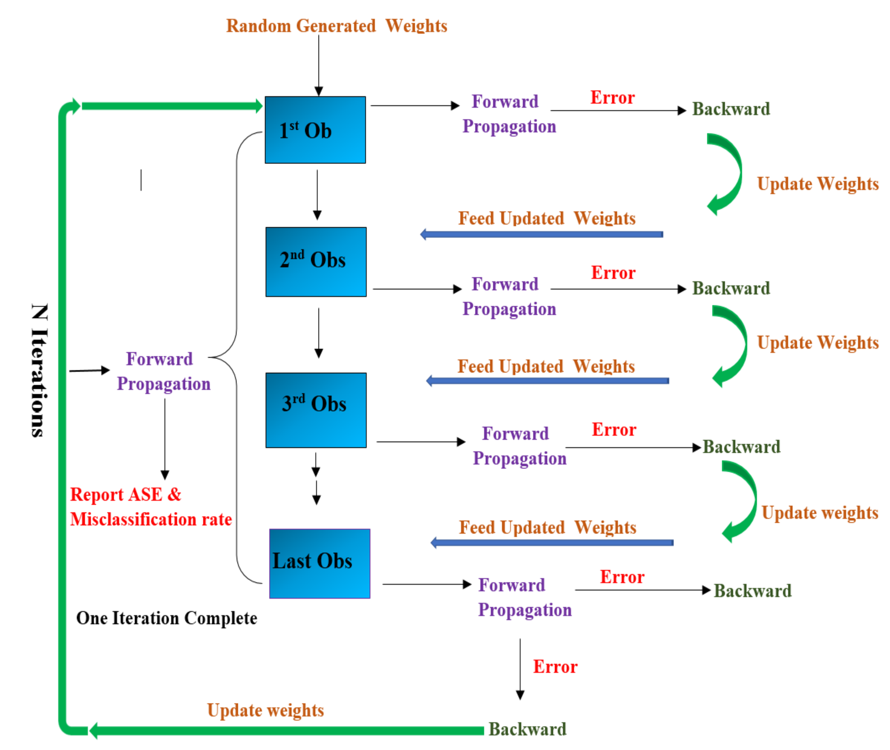

Fig 2. Neural Network Algorithm

Step 1: Standardized Data before being fed into model

Var1	Var2	Var3	Var4	Var5	Var1_H1	Var2_H1	Var3_H1	Var4_H1	Var5_H1	Bias_H1
1	0.62249	1.0143	0.7190	1.1830	0	0	0	0	0	0

Var1_H2	Var2_H2	Var3_H2	Var4_H2	Var5_H2	Bias _H2	Var1_H3	Var2_H3	Var3_H3
0	0	0	0	0	0	0	0	0

Var4_H3	Var5_H3	Bias _H3	H1_Output	H2_Output	H3_Output	Bias _Output
0	0	0	0	0	0	0

H1	H2	H3	Actual	OutputZ	NNetError	Predicted
0	0	0	0	0	0	0

Step 2: Initial Weights are randomly generated

Var1	Var2	Var3	Var4	Var5	Var1_H1	Var2_H1	Var3_H1	Var4_H1	Var5_H1	Bias_H1
1	0.62249	1.0143	0.7190	1.1830	0.92087	0.70797	0.62601	0.90365	0.11236	0.64727

Var1_H2	Var2_H2	Var3_H2	Var4_H2	Var5_H2	Bias _H2	Var1_H3	Var2_H3	Var3_H3
0.17533	0.58943	0.13338	0.52636	0.24904	0.18099	0.91941	0.31108	0.36089

Var4_H3	Var5_H3	Bias _H3	H1_Output	H2_Output	H3_Output	Bias _Output
0.98716	0.44222	0.98709	0.13747	0.33574	0.89631	0.44536

H1	H2	H3	Actual	OutputZ	NNetError	Predicted
0	0	0	0	0	0	0

Step 3: NODE values (H1,H2,H3, OutputZ) and NNetError are updated after Front Propagation

Var1	Var2	Var3	Var4	Var5	Var1_H1	Var2_H1	Var3_H1	Var4_H1	Var5_H1	Bias_H1
1	0.62249	1.0143	0.7190	1.1830	0.92087	0.70797	0.62601	0.90365	0.11236	0.64727

Var1_H2	Var2_H2	Var3_H2	Var4_H2	Var5_H2	Bias _H2	Var1_H3	Var2_H3	Var3_H3
0.17533	0.58943	0.13338	0.52636	0.24904	0.18099	0.91941	0.31108	0.36089

Var4_H3	Var5_H3	Bias _H3	H1_Output	H2_Output	H3_Output	Bias _Output
0.98716	0.44222	0.98709	0.13747	0.33574	0.89631	0.44536

H1	H2	H3	Actual	OutputZ	NNetError	Predicted
0.96852	0.82225	0.97586	0	0.84931	0.84931	1

Step 4: Updating weights using Back Propagation Algorithm

Var1	Var2	Var3	Var4	Var5	Var1_H1	Var2_H1	Var3_H1	Var4_H1	Var5_H1	Bias_H1
1	0.62249	1.0143	0.7190	1.1830	0.92082	0.70794	0.62596	0.90361	0.11230	0.64723

Var1_H2	Var2_H2	Var3_H2	Var4_H2	Var5_H2	Bias _H2	Var1_H3	Var2_H3	Var3_H3
0.17480	0.58910	0.13284	0.52597	0.24841	0.18045	0.91919	0.31093	0.36066

Var4_H3	Var5_H3	Bias _H3	H1_Output	H2_Output	H3_Output	Bias _Output
0.98700	0.44195	0.98686	0.12694	0.32680	0.88571	0.43449

H1	H2	H3	Actual	OutputZ	NNetError	Predicted
0.96852	0.82225	0.97586	0	0.84931	0.84931	1

Step 5: Front Propagation was run using weights updated from back propagation. This completes one iteration.

Var1	Var2	Var3	Var4	Var5	Var1_H1	Var2_H1	Var3_H1	Var4_H1	Var5_H1	Bias_H1
1	0.62249	1.0143	0.7190	1.1830	0.92087	0.70797	0.62601	0.90365	0.11236	0.64727

Var1_H2	Var2_H2	Var3_H2	Var4_H2	Var5_H2	Bias _H2	Var1_H3	Var2_H3	Var3_H3
0.17533	0.58943	0.13338	0.52636	0.24904	0.18099	0.91941	0.31108	0.36089

Var4_H3	Var5_H3	Bias _H3	H1_Output	H2_Output	H3_Output	Bias _Output
0.98716	0.44222	0.98709	0.13747	0.33574	0.89631	0.44536

H1	H2	H3	Actual	OutputZ	NNetError	Predicted
0.98781	0.91302	0.99075	0	0.85015	0.85015	1

Step 6: After 100 iterations error is reduced substantially

Var1	Var2	Var3	Var4	Var5	Var1_H1	Var2_H1	Var3_H1	Var4_H1	Var5_H1	Bias_H1
1	0.62249	1.0143	0.7190	1.1830	0.92369	0.70972	0.62887	0.90567	0.11569	0.65009

Var1_H2	Var2_H2	Var3_H2	Var4_H2	Var5_H2	Bias _H2	Var1_H3	Var2_H3	Var3_H3
0.17843	0.59136	0.13653	0.52859	0.25270	0.18409	0.91554	0.30867	0.35696

Var4_H3	Var5_H3	Bias _H3	H1_Output	H2_Output	H3_Output	Bias _Output
0.98438	0.43764	0.98322	-0.71269	-0.45459	0.043792	-0.41526

H1	H2	H3	Actual	OutputZ	NNetError	Predicted
0.98822	0.92202	0.99071	0	0.18312	0.18312	0

Fig 3. Implementation of Neural Network

Performance of Our Algorithm vs Python Algorithm

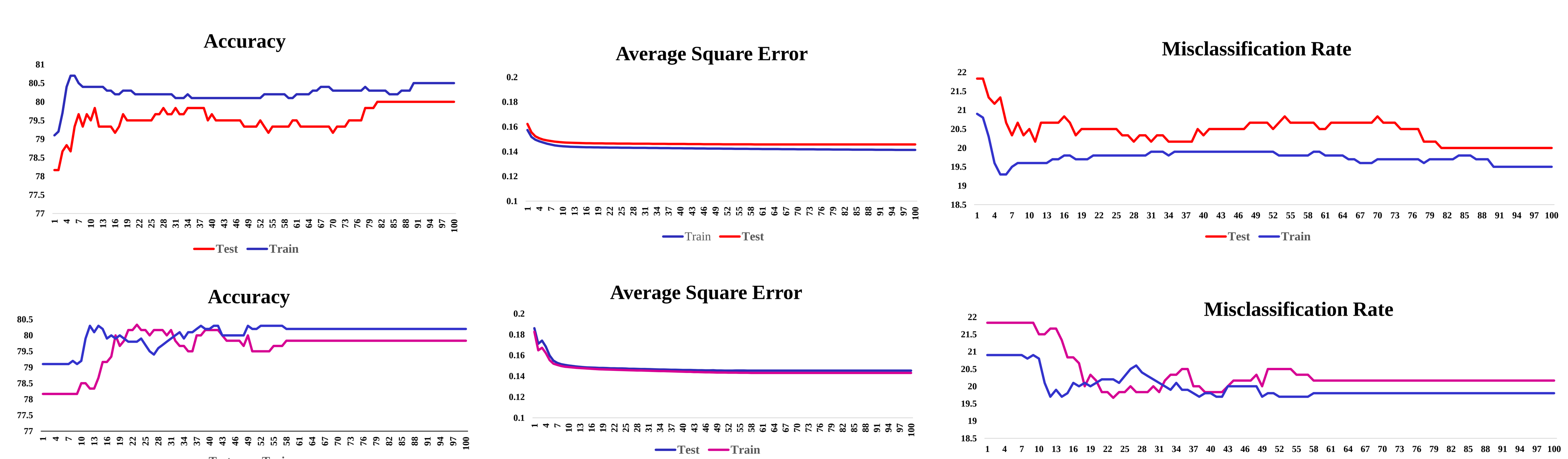

Fig 4. Comparison of Neural Network in SAS and Python

SUMMARY

- Best Accuracy (80.66%) was found at 80th iteration and the performance is similar to Python Neural Network
- The nuts and bolts of Neural Networks has been shown
- No such algorithm exist in Base SAS using Macros
- This algorithm is beneficial for companies using Base SAS

Future Work

- Model can be trained with more number of variables and observations to achieve better accuracy.

SAS CODE

```
/******************** Created Main Macro with the below paramters********************/
%macro neuraltest(dsn=, hiddenextension=, hiddennodes=, outextension=,
outputnodes=, LR=, StartIteration=, EndIteration=);

  %let dsid=%sysfunc(open(&dsn));
  %let nvar=%sysfunc(attrn(&dsid, nvars));
%GENERATEVARLIST(DSN=SOU_FOR.curnnetvar_stdtrain, exclude=RESP_DV MK);
%put &varname;
%let inputvarname = &varname;
/***************** Preparing the data for Front Propagation*************/
data hidden_weights (drop=&inputvarname);
set &dsn;
   %do p=1 %to &nvar;
   %let var=%sysfunc(varname(&dsid, &p));
       %do q=1 %to &hiddennodes;
       &var=&var&hiddenextension&q;
       &var&hiddenextension&q=rand('uniform');
       %end;
    %end;
/* Close the data set */
%let rc=%sysfunc(close(&dsid));
run;

/***************************** Macro for FRONT PROPAGATION ***********************/
%MACRO FRONTPROPAGATION (fpdsn=, fpdsnout=);

  data &fpdsnout(drop=k l a b);
  set &fpdsn;
  array inputdata(*) &inputvarname;
  array hiddenweights(&nvar, &hiddennodes) &hidden_weights_name;
  array hiddenintercepts(*) &hidden_interceptname;
  array hiddennodes(*) &hidden_nodename;
  array outweights (&hiddennodes, &outputnodes) &output_weights_name;
  array outintercepts(*) &out_interceptname;
  array outputnodes(*) &output_nodename;
    do k=1 to &hiddennodes;
         hiddennodes(k) =  hiddennodes(k)+ hiddenintercepts(k);
       do l=1 to &nvar;
          put k=l=hiddennodes(k)=;
         hiddennodes(k)=hiddennodes(k)+(inputdata(l) * hiddenweights(l, k));
      end;
        hiddennodes(k)= 1/(1+exp(-hiddennodes(k)));
        put k=l=hiddennodes(k)=;
    end;
  NNetError=RESP_DV-OutputZ; NNetErrorSQ = NNetError*NNetError;
  NNetPred1 = OutputZ;
  NNetPred0=1-NNetPred1;
  if NNetPred1 > NNetPred0 then NNPrediction= 1;
  else NNPrediction= 0;
  NNCorrect = (NNPrediction=RESP_DV);
  run;
  %MEND ;
/* **************BACKPROPAGATION Macro ******************** */
%MACRO BACKPROPAGATION (bpdsn=, bpdsnout= );

 data &bpdsnout(drop=k l);
set &bpdsn;
array inputdata(*) &inputvarname;
array hiddennodes(*) &hidden_nodename;
Array deltahiddennodes(*) &deltahidden_nodename;
Array deltahiddenoutweight(*) &deltaoutput_weightname;
Array deltahiddenintercept(*) &deltahidden_interceptname;
Array hiddenintercepts(*) &hidden_interceptname;
Array outweightsbp (*) &output_weights_name;
Array hiddenweights(&nvar, &hiddennodes) &hidden_weights_name;
Array deltahiddenweights(&nvar, &hiddennodes) &deltahidden_weightname;

DELTAZ= OutputZ*(1-OutputZ)*(Resp_DV-OutputZ);
LR=&LR;
deltaBIAS_RESP_DV = LR*DELTAZ*1;
BIAS_RESP_DV=(BIAS_RESP_DV) + (deltaBIAS_RESP_DV);
do k=1 to &hiddennodes;
deltahiddennodes(k)= hiddennodes(k)*(1-hiddennodes(k))*outweightsbp (k)*DELTAZ;
deltahiddenoutweight(k)=LR*DELTAZ*hiddennodes(k);
 do k=1 to &hiddennodes;
     do l=1 to &nvar;
     deltahiddenweights(l,k)= LR* deltahiddennodes(k)*inputdata(l);
     end;
 end;
 do k=1 to &hiddennodes;
     do l=1 to &nvar;
     hiddenweights(l,k)= hiddenweights(l,k)+deltahiddenweights(l,k);
     end;
 end;
 run;
 %MEND;
 %do ITER= &StartIteration %to &EndIteration;
   %do loop=1 %to &TRAINNUMOBS;

 data curnnetvar_s;
 set SOU_FOR.curnnetvar_stdtrain (firstobs=&loop obs=&loop);run;
 data neuraltestdataFP;
 merge curnnetvar_s neuraltestdataFP;
 run;

 %FRONTPROPAGATION(fpdsn=neuraltestdataFP, fpdsnout=neuraltestdataFP);
 %BACKPROPAGATION(bpdsn=neuraltestdataFP, bpdsnout=neuraltestdataBP);
  /* Front propogating the last observation weight on whole training dataset*/
  data neuraltestdataFPTRAIN_ITR1(drop=i);
  set neuraltestdataFPTRAIN_ITR1;
   do i = 1 to &TRAINNUMOBS;
   output;
   end;
   run;

 data neuraltestdataFPTRAIN_ITR1;
 /* set neuraltestdataFPTRAIN_ITR1;*/
 merge SOU_FOR.curnnetvar_stdtrain  neuraltestdataFPTRAIN_ITR1;
 run;

 %FRONTPROPAGATION(fpdsn=neuraltestdataFPTRAIN_ITR1, fpdsnout=neuraltestdataFPTRAIN_Fi

 /* calculating the Fit statisics on training dataset******/
  proc sql;
  create table SOU_FOR.NEURALTRAINITER&ITER as
  select mean(nneterrorsq) as NNASE, sum(NNCorrect )/count(*) as NNAccuracy,
  (1-(sum(NNCorrect )/count(*))) as NNMCR from neuraltestdataFPTRAIN_FinaLITR&ITER;
  quit;

  /*Running the main macro by passing all parameters*/
 %neuraltest(dsn=curnnetvar1, hiddenextension=_H, hiddennodes=3,
 outextension=_RESPONSE, outputnodes=1, LR=0.1, StartIteration=4, EndIteration=6);
```