Diabetes Detection Using Artificial Neural Networks & Back-Propagation Algorithm

Ms. Divya(Assistant Professor(MSIT)), Raman Chhabra, Sumit Kaur, Swagata Ghosh

Abstract:- This is a project aimed at predicting the chances of diabetes in a person, that whether or not is he/she prone to it. We have used certain parameters namely: number of pregnancies, glucose, BP, skin fold, insulin, body mass index, pedigree and age. The database of 768 patients with these parameters each was taken from National Institute of Diabetes and Digestive and Kidney Diseases. Using neural network feed forward prediction model in conjunction with back propagation algorithm, and given training data set, we predicted whether a subject was likely to have diabetes.

Index Terms:- Artifical Neural Network,Transfer Function,Body Mass Index

1. Introduction

The aim of our project was to use certain key parameters to predict whether a person was suffering or likely to suffer from diabetes. These parameters are:

- Number of pregnancies
- Blood Glucose Level
- BP (Systolic)
- Skin Fold
- Insulin
- Body Mass Index
- Pedigree
- Age

We used a 3 layer neural network with one hidden layer and customizable number of hidden layer neurons with a customization option for the prediction function used. We tried out approximating using 3 functions:

- Binary Sigmoidal
  $f(x)=1/(1+e^{-x})$
- Bipolar Sigmoidal
  $f(x)=2/(1+e^{-x}) - 1$
- Hyperbolic Tangential
  $f(x)=(e^{x} - e^{-x})/ (e^{x} + e^{-x})$

The more the number of hidden layer neurons the more accurate was the result, but it took more computation time. So, a decent trade off value of 20 was chosen. Experimental results showed that binary sigmoidal function was the most accurate of all.

2. Previous Works

Neural network simulations appear to be a recent development. However, this field was established before the advent of computers, and has survived at least one major setback and several eras. Many important advances have been boosted by the use of inexpensive computer emulations. Following an initial period of enthusiasm, the field survived a period of frustration and disrepute. During this period when funding and professional support was minimal, important advances were made by relatively few researchers. These pioneers were able to develop convincing technology which surpassed the limitations identified by Minsky and Papert. Minsky and Papert, published a book (in 1969) in which they summed up a general feeling of frustration (against neural networks) among researchers, and was thus accepted by most without further analysis [7]. Currently, the neural network field enjoys a resurgence of interest and a corresponding increase in funding. The first artificial neuron was produced in 1943 by the neurophysiologist Warren McCulloch and the logician Walter Pits. But the technology available at that time did not allow them to do too much [2].

3.1 ANN (Artificial Neural Network) using back–propagation algorithm

In order to train a neural network to perform some task, we must adjust the weights of each unit in such a way that the error between the desired output and the actual output is reduced. This process requires that the neural network compute the error derivative of the weights (EW). In other words, it must calculate how the error changes as each weight is increased or decreased slightly. The back propagation algorithm is the most widely used method for determining the EW. The back-propagation algorithm is easiest to understand if all the units in the network are linear. The algorithm computes each EW by first computing the EA, the rate at which the error changes as the activity level of a unit is changed. For output units, the EA is simply the difference between the actual and the desired output. To compute the EA for a hidden unit in the layer just before the output layer, we first identify all the weights between that hidden unit and the output units to which it is connected. We then multiply those weights by the EAs of those output units and add the products. This sum equals the EA for the chosen hidden unit. After calculating all the EAs in the hidden layer just before the output layer, we can compute in like fashion the EAs for other layers, moving from layer to layer in a direction opposite to the way activities propagate through the network. This is what gives back propagation its name. Once the EA has been computed for a unit, it is straight forward to compute the EW for each incoming connection of the unit. The EW is the product of the EA and the activity through the incoming connection. Note that for non-linear units, the back-propagation algorithm includes an extra step. Before back-propagating, the EA must be converted into the EI, the rate at which the error changes as the total input received by a unit is changed [8].

3.2 Pseudo Code

```
1.  {

2.  Initialize nodes and layers to form the neural network.

3.  Initialize biases, weights and learning rate.
```

4. Obtain training samples and their expected output values.

5. While(terminating condition is not true)
   //it could be a specified no: of iterations or error rate to prevent infinite loop.

6. {

7. Initialize each input layer 'i' node with normalized inp_par[i]

8. For each hidden layer node 'j'

9. {

10. Calculate input to node as value of "sum" given by:
    Sum=∑wt[i][j]*inp_val[i], where wt[i][j] is the weight associated with the link (i,j). Here, 'i' is input layer node and 'j' is the current hidden layer node.

11. Calculate output from node 'j' according to the prediction function used + the 'bias' value for the layer.

12. }

13. For each output layer node 'j'

14. {

15. Calculate input to node as value of "sum" given by:
    Sum=∑wt[i][j]*val[i], where wt[i][j] is the weight associated with the link (i,j). Here, 'i' is hidden layer node and 'j' is the current output layer node.

16. Calculate output from node 'j' according to the prediction function used + the 'bias' value for the layer.

17. }

18. Calculate error for each node 'j' of output layer as:
    Err_out[j]=(Expected output – Obtained output)*Obtained output*(1 – Obtained output).

19. Calculate error for each node 'j' of hidden layer as:
    a. Calculate "sum" as:
       Sum=∑Err_out[i]*wt[i][j], where wt[i][j] is the weight associated with the link (i,j) between hidden layer node 'j' and output layer node 'i'.
    b. Err_hid[j]=(Obtained output)*(1 – Obtained output)*sum

20. // Adjust weights and biases according to the errors calculated by back propagation.

21. For each weight wt[i][j] between input and hidden layer
    a. Wt[i][j]=wt[i][j]+lr*Err_hid[j]*(Obtained output at j), where 'lr' is the learning rate

22. For each weight wt[i][j] between hidden and output layer
    a. Wt[i][j]=wt[i][j]+lr*Err_out[j]*(Obtained output at j), where 'lr' is the learning rate

23. Bias_hid=bias_hid+lr*RMS(Err_hid[j]), where Err_hid[j] is the error of a hidden layer node j.
24. Bias_out=bias_out+lr*RMS(Err_out[j]), where Err_out[j] is the error of a output layer node j.

25. }

26. //Training is complete, display results in user console. Start predicting new samples.

27. Accept inp_par[] from the user and perform steps 7 – 16.

28. The resulting pattern obtained from the output of all the output layer nodes is compared with a pattern sequence to detect a match and the respective class of final output is displayed to the user. The accuracy of prediction will largely depend on the data set given and the parameters entered. It will largely depend on the statistical correlation of the values of the parameters and the respective final output class.

29. }

4. Result Obtained

We trained our neural network with 200 samples from the database and tested it with 50 samples. We obtained specificity of 82.14% & sensitivity of 88.8%. We obtained RMS error rate of 0.019% for the 50 samples tested. (Here we calculated error in terms of percentage error of output from output expected i.e. 0 or 1.) On an average it took about 1275 iterations to converge to the result with learning rate of 0.1 and momentum of 0.9, which was found out to be an optimal value for binary sigmoidal function.

5. Conclusion

This project was aimed at modeling neural network for a prediction scenario – diabetes and its type prediction here and to analyse the prediction results and compare the network efficiency upon changing the network parameters like the number of hidden layer nodes, type of normalization function used, prediction function used, learning rate and the initial biases for each layer. The correlation between training samples, their expected outputs and the number of iterations required to predict with RMS error below a threshold of 0.02% was also analysed. It took an average 1250 iterations to achieve the output. The maximum number of iterations was kept to 10000 so that the algorithm doesn't get stuck into an infinite loop if the prediction is not very accurate and quick. In general, we took 8 basic parameters to predict diabetes and its type in a person. The number of samples taken was not too many and so the algorithm took more than expected iterations to achieve optimum result. Initially we had assumed that 500 iterations would be more than enough for the task. Also on comparing the different prediction functions, it was found that sigmoidal function was the best and quickest to predict the result. Sigmoidal functions of two types namely: Binary and Bipolar were compared. Binary sigmoidal function gave results faster in less number of iterations. Bipolar sigmoidal gave results which were more accurate, but, in general it took about 250-300 iterations more than its counterpart. So, we have a trade-off between accuracy and the number of iterations, and in fact if we multiply both the functions and an optimization constant together and perform an optimization based on some criteria function we can improve the overall time-efficiency & accuracy of the algorithm by atleast a factor of 2. The implementation of this optimization was also left as an extension to the basic version of the project. Hyperbolic tangential function was also accurate but it took too many iterations to converge to the final result. In fact, it more often than not took more than 7000 iterations to obtain the final result, and so we had to set the threshold number of iterations to 10000. Otherwise, a threshold of about 3000 was sufficient. This is mainly due to the geometrical nature of the hyperbolic curve in consideration. The rate of change of the slope of tangent to the curve was slow, and so it took long to converge to a constant stable value.

6. Refrences

[1] An introduction to neural computing. Aleksander, I. and Morton, H. 2nd edition

[2] Neural Networks at Pacific Northwest National Laboratory http://www.emsl.pnl.gov:2080/docs/cie/neural/neural.homepage.html

[3] Artificial Neural Networks in Medicine http://www.emsl.pnl.gov:2080/docs/cie/techbrief/NN.techbrief.ht

[4] Industrial Applications of Neural Networks (research reports Esprit, I.F.Croall, J.P.Mason)

[5] A Novel Approach to Modelling and Diagnosing the Cardiovascular System http://www.emsl.pnl.gov:2080/docs/cie/neural/papers2/keller.wcnn95.abs.html

[6] Electronic Noses for Telemedicine http://www.emsl.pnl.gov:2080/docs/cie/neural/papers2/keller.ccc95.abs.html
An Introduction to Computing with Neural Nets (Richard P. Lipmann, IEEE ASSP Magazine, April 1987)

[7] Pattern Recognition of Pathology Images http://kopernik-eth.npac.syr.edu:1200/Task4/pattern.html

[8] Developments in autonomous vehicle navigation. Stefan Neuber, Jos Nijhuis, Lambert Spaanenburg. Institut fur Mikroelektronik Stuttgart, Allmandring 30A, 7000 Stuttgart-80

[9] Klimasauskas, CC. (1989). The 1989 Neuro Computing Bibliography. Hammerstrom, D. (1986). A Connectionist/Neural Network Bibliography.

[10] DARPA Neural Network Study (October, 1987-February, 1989). MIT Lincoln Lab.

[11] Neural Networks, Eric Davalo and Patrick Naim.

[12] Assimov, I (1984, 1950), Robot, Ballatine, New York.

[13] Learning internal representations by error propagation by Rumelhart, Hinton and Williams (1986).

[14] Alkon, D.L 1989, Memory Storage and Neural Systems, Scientific American, July, 42-50

[15] Minsky and Papert (1969) Perceptrons, An introduction to computational geometry, MIT press, expanded edition.

[16] Neural computers, NATO ASI series, Editors: Rolf Eckmiller Christoph v. d. Malsburg