Fault tolerance in neural networks

Dipti Chaudhari (015005131)
Masters of Science in Computer Science.
Department of Computer Science and Engineering,
California State University,
Long Beach, California.
dipti.chaudhari.24@gmail.com

Tanmay Kadam (014731533)
Masters of Science in Computer Science.
Department of Computer Science and Engineering,
California State University,
Long Beach, California.
tkadam7171@gmail.com

Abstract — ***This paper explains the overall concept behind the working of neural networks. It mainly focusses on the performance and efficiency of a neural network in different cases and who the network responds or reacts to that particular situation. It implements the nature of distributed feedforward neural network with decentralized event-driven time management, by injecting faults (intermittent faults caused by unreliable communication or faulty hardware components).***

Keywords – neural networks, fault tolerance, matrix, redundancy.

I.INTRODUCTION

Neural network is a computer system modeled on the human brain and the nervous system. Some of the most popular systems based on neural networks are game playing and decision making (example: Chess and poker), pattern recognition (example: Face Identification) and sequence recognition (example: Gesture and speech recognition). We can say that neural networks is used everywhere. The main reason for using neural network is its self organization property and its distributed nature of system for fault tolerance.

This paper is structured as follows. It explains how simulation of a neural network is done and then how a particular network is trained with the user knowledge from the database.

After the training is carried out how do we test a particular network and find the prediction accuracy. In order to find the desired results how and why some changes are made in the network and what changes are observed in the accuracy. We then represent this result in a graphical representation.

The motivation for this paper is that the brain is capable of handling faults and continue functioning without remarkably affecting its performance. This feature is also exhibited by neural networks. Studying this behavior will help in implementing fault tolerance in systems which are installed on unreliable hardware.

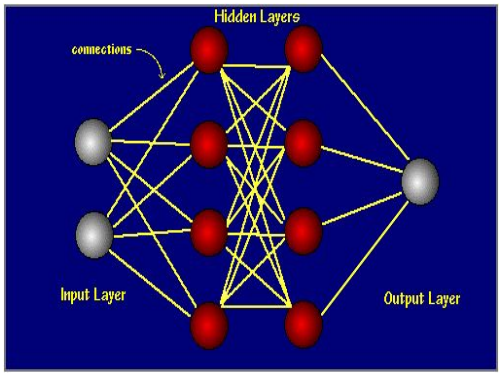

Fig 1: Basic neural network

The above figure clearly explains the basic representation of a neural network. It contains a set of input layer, output layer and

hidden layer. The number of input layer and output layer can be changed , the complexity of the hidden layer changes with the complexity of the problem we are working on. We calculate the performance of this neural network and then we make some changes in the hidden layer by removing a particular node and see what is the performance efficiency of that neural network now.

II. PAST RESEARCH

In the previous study the fault tolerance on neural networks was conducted using c++. The MPI library was used to carry out all the operation that were needed to find out the performance and efficiency of a neural network.

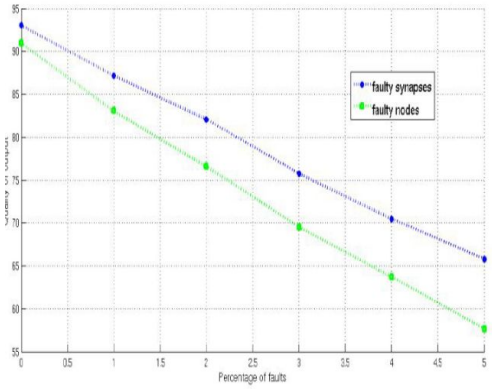

Fig 2: performance vs fault

Here the communication links were treated as different processes. The functioning and failure of communication was controlled by MPI libraries. From the graphical representation it is observed that as the number of faults increases the performance of the problem or the network decreases.

The more number of nodes in the network the more complicated it gets. Using this study we can conclude that which node does not affect the overall performance and efficiency of the neural network.

III. IMPLEMENTATION

What we basically do in the implementation process is that we pass a set of data to the network and the network is trained to accept this data in particular intervals.

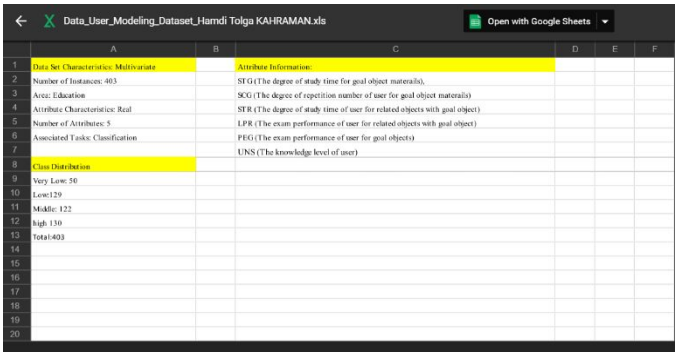

Fig 3: Data Set used For the implementation

As the data is passed the neural network changes its default values and accepts the trained values from the data set. 90% of the data is used for the training purpose of the network. As the network is trained it further needs to be tested to find the prediction accuracy of the neural network. 10% of the remaining dataset is used to test the data and find the prediction accuracy.

In order to test the changed data for fault tolerance, what we do in this neural network is that we remove one of the node from the hidden layer and now again find the prediction accuracy. As the number of nodes are removed there is a difference found in the prediction accuracy of the given neural network.

IV. SIMULATION TOOLS

The following simulation tools are used during the implementation on the neural networks:

- ★ Python
  - Numpy
  - Pandas

- Scikitlearn
- Google Tensorflow
- pyPlots

★ IDE
- Pycharm Community

★ Environment
- Linux

Numpy, pandas and scikitlearn tools are used for all the matrix calculation. A neural network is basically represenated as a matrix table in a program.

V.CASE STUDY

Case study suggests that we show the result of our program. How the test accuracy is affected when certain number or fraction of links selected randomly are turned off.

Here in Fig 4 and 5 is depicting the actual performance of the system.

The algorithm we used is as follow:

```
# function to get one-zero random numpy array
def get_one_zero_random_numpy_matrix(num_of_rows, num_of_cols, num_of_zeros):
        total_elements = num_of_rows * num_of_cols
        mask = np.concatenate((np.zeros(num_of_zeros), np.ones(total_elements - num_of_zeros)))
    np.random.shuffle(mask)
        mask = np.reshape(mask, newshape=[num_of_rows, num_of_cols])
    return (mask)

# function to reset random links (set to zero)
def reset_links(weights, what_fraction_to_reset):
    num_of_rows = weights.shape[0]
    num_of_cols = weights.shape[1]
        total_links = int(num_of_rows * num_of_cols)
    num_of_links_to_reset = int(total_links * what_fraction_to_reset)
    mask = get_one_zero_random_numpy_matrix(num_of_rows, num_of_cols, num_of_links_to_reset)
    return (weights * mask)
```

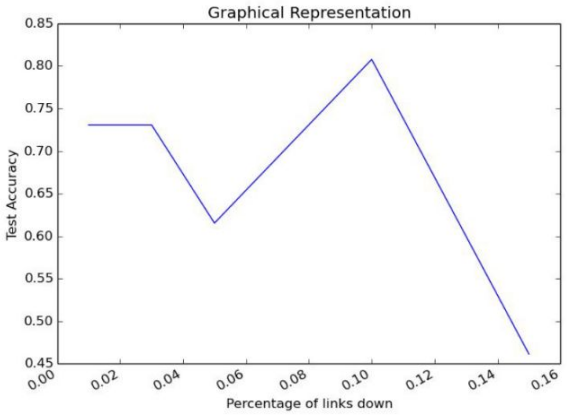

Fig 5: Reset fraction of links in hidden layer
Training accuracy = 0.711206896552
Testing accuracy = 0.730769230769

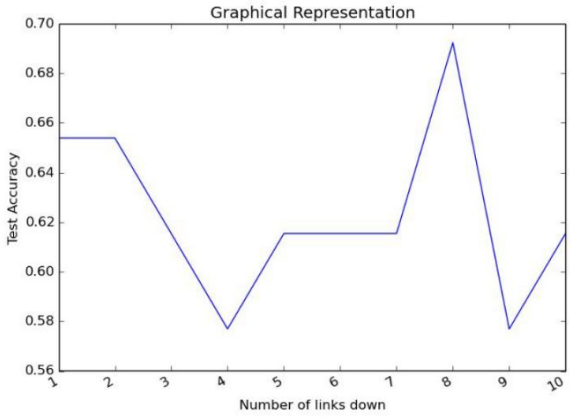

Fig 4: Reset # of links in hidden layer
Down_link = [1,2,3,4,5,6,7,8,9,10]
Training accuracy = 0.61745362522
Testing accuracy = 0.653846153846

The experiment shows that how neural network systems are fault tolerant. We saw even if the functional units are selected

VI.CONCLUSION

randomly or in some particular fashion the performance of the system is not hampered to big extend. Also as observed the accuracy also shown a significant increase but that is because of the training of the network sometimes shows the random behavior. Another important observation made is that fault tolerance also depends on the quality of data. Our data set was quiet reliable hence the performance is showing significant sustainability.

VII.FUTURE WORK

The accuracy is sometimes degraded gracefully.It means the system is not turned down but the performance is affected to some extend. The future work in this project would be to actually come up with the number of iterations needed over the original value to gain the actual test accuracy.
This will require

VIII.REFERENCES

[1] Conventional fault-tolerance and neural computers, Will R. Moore

[2] Skeletonization: A technique for trimming the fat from a network via relevance assessment,M. Mozer and P. Smolensky

[3] J. Dongarra, H. Meuer, E. Strohmaier, and H. Simon, "Top 500 supercomputer sites." May 2012, [Online]. Available: http://www.top500.org/.

[4] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in Proc. MICRO 36, 2003, pp. 29–40.

[5] S. Furber and S. Temple, "Spinnaker a universal spiking neural network architecture," University of Manchester, Manchester, UK, Tech. Rep., Oct. 2010.

[6] S. Furber and D. Lester, "SpiNNaker project." May 2012, [Online]. Available: http://apt.cs.man.ac.uk/projects/SpiNNaker/.