Training Simulation of Learning Algorithm for Deep Learning Neural Network

Nurul Afiqah Mohd Hasbullah

***Abstract*— Neural network is one of a significant in deep learning. The most applied algorithm to train deep learning neural network is Backpropagation. Thus, how to train and simulate learning algorithm for deep learning network? What is the best condition of building and training neural networks? Over the years, there has been many available resources for building and replicating neural network especially in Phyton, MATLAB and C++ language. Thus, Java language is chosen to build a neural network simulator which combines Java application and Simbrain application. 1000 MNIST and 500 MNIST dataset will be implemented to train and test the neural network. The Java application will allow user to create a collection of 28x28 pixel data to validate neural network that is created using BeanShell script in Simbrain application. This paper will compare on two types of neural network, Backpropagation Neural Network (BNN) and Backpropagation with Noise Out Neural Network (BNONN). Noise Out is one of a method to prune neurons in hidden layer of neural network. By applying Noise Out, the performance is improving on Final Training MSE by 77%, Testing MSE by 49% and Elapsed time by 60%.Another factor that affect the accuracy in neural network is by applying different number of neurons in a hidden layer. As the neurons increase, the Mean Square Error (MSE) will be large. This can be improved by adding a second hidden layer with 28 neurons.**

***Index Terms*— Backpropagation, Deep Learning, Java, Neural Network, Noise Out, Simbrain, Simulation.**

I. Introduction

MACHINE learning has been a norm for many industrial purposes especially in high tech company such as Google and Apple. There are many applications of machine learning such as face recognition for lock security purpose in iPhone and language recognition for Bixby application in Samsung product. Machine learning is a part of Artificial Intelligence (AI). There are two types of learning: Deep Learning and Reinforcement Learning. Deep Learning is introduced by [1], where it introduced the essential old architecture of a neural network, which is Deep Feedforward or Multilayer Perceptron (MLP). The backpropagation in deep learning neural network is additionally clarified with the assistance of hands-on programming in a book by [2]. Deep learning is later on discussed in [3], [4] and [5]. On the other hand, Reinforcement Learning is implemented in a game of Go [6] and traffic light [7]. The different between Deep Learning and Reinforcement Learning is Deep Learning implement an algorithm and is trained by training set while Reinforcement Learning is trained by preliminaries and mistake. Deep Learning has many applications including image recognition [8], synthetization of speech [9], forecasting [10] [11] [12] and medical issues detection [13] [14] [15].

Nurul Afiqah Mohd Hasbullah is with Engineering and Technology Department, Massey University East Precinct, Dairy Flat Highway (SH17), 0632, Albany, Auckland, New Zealand (e-mail: fiqahas25@gmail.com).

Deep Learning implements Artificial Neural Network (ANN) where it builds a neural network based on biological neurons in the human brain, which is called Deep Neural Network (DNN). There are various types of architecture of neural network other than MLP in Deep Learning. There are five models that have the most accessible reference. The one that has the most research papers on is Convolutional Neural Network (CNN). In [16], it is expressed that weight sharing can decrease the multifaceted nature of structure in CNN contrast with other neural systems. In any case, in [17], it expresses that CNN requires more opportunity to prepare contrast with Capsule Network (Caps Net) and Residual Network (Res Net). [18] analyze the consequence of picture acknowledgment on MNIST dataset with MLP and CNN. It says that CNN does well than 2-layer MLP. Other than CNN, we likewise have a Recurrent Neural Network (RNN). In [19], it is discussed that RNN is hard to train over the past few years but has interesting features which ignore the limitation of other available architecture. [20] propose that utilizing programming languages Java and C# on RNN is superior to Python. Besides, we have Deep Believe Network (DBN). [21] infer that it requires some investment to prepare dataset with DBN. [22] express that unsupervised DBN accomplished more exactness contrasted with supervised DBN. Next, we have a Spiking Neural Network (SNN). [23] introduce Poisson distribution to improve the irregularity in spikes. By presenting novel-spiked backpropagation system in [24], it improves the precision, inertness and vitality effectiveness in SNN. In addition, we have Binarized Neural Network (BNN). [25] introduced a strategy to prepare BNN which prompted lessen memory size and power utilization by utilizing bit-wise activity.

The programming languages that are broadly used for training DNN are Python, Java, MATLAB and C++. The

programming language that has the greatest number of articles is MATLAB [26] [27] , followed by C++ [28] [29] [30] , Python [31] [32] [33] , and Java [34] [35] .There are several available deep learning frameworks to ease the user in building a neural network from scratch. In the research paper by [36], it is expressed that Caffe and DeepLearning4J(DL4J) has its precision diminished while the quantity of neurons bigger. This is the other way around to TensorFlow and Torch where its exactness created as the quantity of neurons expanded. Theano has the best precision with minimal number of neurons while DL4J requires the longest time in preparing. The ranking list from the best to worst is Theano (with Keras), TensorFlow, Caffe, Torch, Deeplearning4J. As for [37], H2O is chosen as the best frameworks compare to TensorFlow and DL4J. According to [38], between CNTK, TensorFlow and Theano, CNTK has outstood the other two in term of recognizing image accurately but TensorFlow has the lowest time consuming followed by CNTK and Theano.

There are few available simulators for the neural network. Firstly, [39] Introduced LENS, a free-of-charge neural network simulator that is written in C and Tel language. Besides that, [40] generates a code in C++ to build neural network simulator in Brian Simulator. In [41], it is concluded that NEST simulator is the most well-known compare to BRIAN, GENESIS, NEST, and NEURON while NEURON is capable of detailed architecture. Later on, [42] introduce Dyna Sim which is a toolbox for neural network simulation in MATLAB. Dyna Sim shows better visualization compare to existing simulator such as NEURON, NEST and XPP since deep learning neural network requires large storage and CPU memory. Thus, there are several research papers that proposed the technique to make the neural network run efficiently. Firstly, [43] proposed a Dropout technique which is basically a technique to reduce overfitting by removing the unnecessary neuron that has zero weight during the process of training the neural network. Later in [44], a technique to reduce memory storage is introduced with named Noise Out. This method recognizes over the top neuron and parameter while keeping up the precision of the neural system by removing a neuron with the highest activation value. In ongoing year, there has been a neural system test system and structure with Java stage named Simbrain [45] [46]. This bundle structures permit UI and scripting neural system in BeanShell content to make neural system test system. It offers assortment of structures and focuses towards understudy and expert. In addition, it has the bit of leeway to hold enormous number of neurons and dependability to run either in low CPU or supercomputer.

This research paper is written to study on how to create the best neural network simulator to learn 10-digit numbers from 0 to 9. The language that is chosen to build the simulator is Java language. This is due to the fact that there are many available open sources for other programming language such as Python, MATLAB and C++. The dataset to train and tests the neural network which has numerous samples of digits is taken from Modified National Institute of Standards and Technology database (MNIST). The MNIST dataset from http://yann.lecun.com/exdb/mnist/ is originally available in image file, yet for this research paper, the dataset must be converted into a Comma Separated Value (CSV) file to be compared with another CSV file which has the value inserted from a user for validation of network. The MNIST CSV file is thoroughly created from the MNIST excel file available in https://www.kaggle.com/oddrationale/mnist-in-csv. The plan to build the neural network simulator is by merging a Java application (for translating digit on drawing pane into CSV file) with an open source Java Archive (for neural network visualization). Based on reading numerous research paper, it is finalized that Simbrain Java Archive will be used as a platform to simulate neural network. Both of Java application and Simbrain Java Archive is a Graphic User Interface (GUI). All of the neural networks presented in this paper will be implementing backpropagation algorithm plus an additional Noise Out pruning method while training neural networks. Therefore, investigating the best condition of building and training neural networks will be the outcome of this paper. The use of the latest GUI and pruning method is what set this paper from other numerous papers of the same field.

II. METHODOLOGY

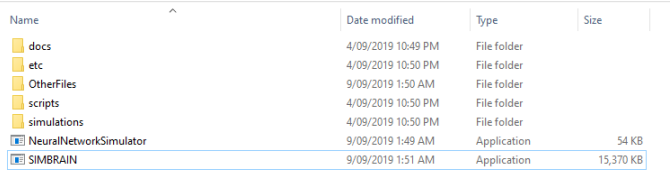

Figure 1 shows the current directory with Neural Network Simulator Application and SIMBRAIN application

The idea of creating neural network simulator is to combine Java application and SIMBRAIN application. Java application will receive digit numbers that is drawn by user and then translate them into rows of data in CSV file. To create the Java application, Java Archive (JAR) file is build and compiled by NetBeans software from numerous written Java script. The jar file is then converted into Java application by using Launch4j software available in the http://launch4j.sourceforge.net/. This Java application is named as Neural Network Simulator. The same method is also applied to SIMBRAIN application. SIMBRAIN application is actually a wrapped form from an original Simbrain Jar file into an application with a console header, hence the name of the application is uppercased. The reason to convert the Simbrain Jar file into console header application is to allow user read the necessary information while training and testing neural network. Simbrain Jar package is an open source available in the https://www.simbrain.net/Downloads/downloads_main.html. As shown in Fig.1, It comes with another necessary folder such as “docs”, “etc.”,” scripts” and “simulations”. The Jar package has its own library in the repository available in https://github.com/simbrain/simbrain. With these library, neural network simulator can be created by writing a BeanShell script with the help of Simbrain library. To run a neural network, user just need to press the name of the

BeanShell script available under a tab named Simulations in SIMBRAIN application. After the neural network is build, train, tested and validated, user can save the current neural network in a form of ZIP file. The manipulated variables of neural networks in this paper are number of neurons, number of hidden layer and addition of Noise Out pruning while backpropagation algorithm and network learning rate are the constant variables.

A. Java Application

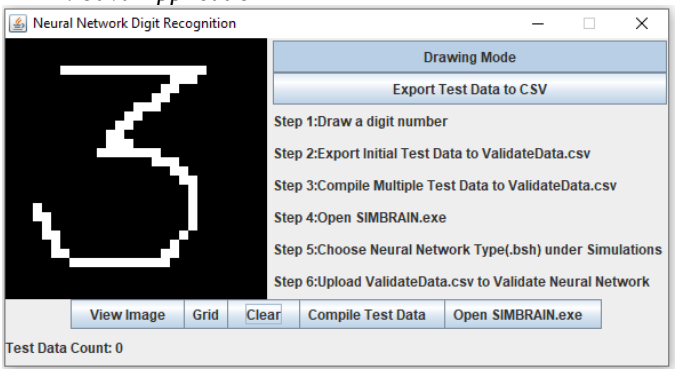

Figure 2 shows Java Application named Neural Network Simulator

In Java application shown in Fig. 2, there are three panels, which are east panel, west panel and south panel. West panel has drawing pane with black background and white input color when mouse button is pressed. The reason for choosing black background is to make the value of the pixel becomes 0 which indicating no data. Originally, one pixel has three values represents Red, Green and Blue. In order to convert a pixel into a neuron, the value of the pixel has to be divided by 3. Toggle and instructions label are situated in east panel while button and test data count label is placed in south panel. Starting from left, View Image button allow user to see the image drawn in 28x28 pixel size. Grid button functions as a guide to draw a digit number on the drawing panel and Clear button simply reset the drawing pane to black in color. In the top east corner, Drawing Mode toggle allow user to change the cursor shape into a plus symbol indicating that drawing is allowed. Export Test Data to CSV toggle functions to change the image drawn on drawing panel into 28x28 pixel by converting a buffered image of the digit drawn into a CSV file. The buffered image is converted into 784 values to match the original MNIST dataset which has a value of 28 width multiply by 28 heights. Once the Export Test Data to CSV toggle is pressed, test data count is changed to 1, indicating a value of a drawn digit number has been written on the first line in the CSV file. If the user wants to write multiple digit number into the CSV file, Compile Test Data button performs as an appended for each value of digit drawn into a newline of CSV file. As the Compile Test Data button clicked increase, the value of test data count will increment by one. After the user finish compiling the test data to the CSV file named ValidateData.csv, Open SIMBRAIN.exe button utilized to run SIMBRAIN application instantaneously. With the CSV file, user can validate the test data in the CSV file with the neural network available in SIMBRAIN application. The steps taken to run a specific neural network and validate the neural network is explained in the east panel.

B. BeanShell Script

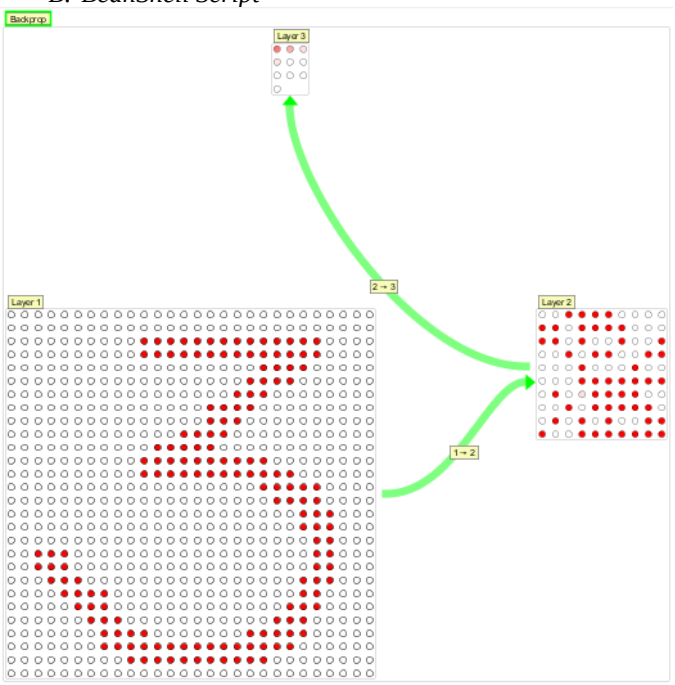

Figure 3 shows neural network framework in SIMBRAIN Application

To create a neural network, the name of the BeanShell script must be the same as a folder that contains the MNIST dataset for training and testing neural network. The folder must contain 4 CSV files which hold the MNIST dataset for training, expected training result, MNIST dataset for testing and expected testing result for neural network. The MNIST dataset for training has 1000 rows (in which 100 datasets targeted for each one of a digit number) while the MNIST dataset for testing has 500 rows (in which 50 datasets targeted for each one of a digit number). MNIST dataset is supposedly to have 60 000 rows for training and 10 000 rows for testing neural network, yet if these huge datasets are applied to the neural network, the computer will be jammed even with the maximum of heap size in Java Runtime Environment (JRE). Therefore, the original MNIST dataset has been reduced to 1000 rows for training and 500 rows for testing. As explained before, one-digit picture in the original MNIST dataset has 28x28 pixels, therefore each row in the MNIST dataset CSV file has 784 values, indicating that one picture is 28 width multiply by 28 height. As for the expected training and testing result data, each row has 10 values indicating that there are 10 values for digit number 0 until 9. In the BeanShell script, network component, network, Backpropagation network, input data, target data, console and date format are initialized in the global variable. The main void function of the BeanShell script includes five other void functions named buildTrainNetwork(), testNetwork, simulate() and finish().

The first function, buildTrainNetwork(), runs the mechanism of building and training neural network. Firstly, the function will print on a console the name of the current BeanShell

script. Next, backpropagation network object is created with the desired number of neurons and hidden layer. The location of input layer, hidden layer and output layer is set specifically to be viewed in SIMBRAIN application. The location of CSV files for MNIST dataset for training and expected training result is included as a set for the neural network to be trained. All neural network in this research applied Backpropagation algorithm, therefore the training set for neural network is randomized and the learning rate is set to 0.01 as a constant variable. The user has the option to change the number of iterations by following the directory "scripts>scriptmenu" and edit the BeanShell script. The stopwatch is started right before the loop for training neural network which incremented until the maximum number of iterations. In the loop, the BeanShell script will print the time, iteration and Mean Square Error (MSE) into a console and a new CSV file named with a concatenation of the BeanShell script name and "ChartData" string. As an example, the BeanShell script of a neural network with one hidden layer of 400 neurons and applied backpropagation only algorithm has the name of "BNN4001L". Therefore, the new CSV file created will be named "BNN4001LChartData". This new CSV file will be utilized as inputs for creating image of three charts which are Iteration vs MSE, Time vs MSE and Time vs Iteration. Additionally, if Noise Out pruning is required, the pruning will occur in this loop. This pruning method is carried out by finding a neuron with the highest activation value and remove the neuron from the neural network. After the loop has finished, the stopwatch will be finished and the elapsed time is calculated by subtracting the start time from the finish time. Then, the elapsed time will be printed on console. The activation value of neurons is cleared in this void function to allow the neural network to be tested afterwards.

The second function named testNetwork() execute the mechanism of testing neural network. The location of CSV files for MNIST dataset for testing and expected testing result is included as a set for the neural network to be tested. Testing neural network occurs in a loop until the maximum length of rows in expected testing result CSV files achieved. In the loop, the activation value is forced in the input layer and after the network is updated, the activation value in output layer is stored. The activation value is compared to expected testing result to get MSE value. The value of MSE in the current row length is then added to the value of MSE from previous row length. Afterwards, the BeanShell script will print the total value of MSE for testing neural network.

The third function in the BeanShell script prints the command to a batch file named with the same name as the new CSV file (example: BNN4001LchartData.bat) to run a Visual Basic (VB) script in "OtherFiles" folder. This VB script is available from open source Github https://github.com/end2endzone/csvplot to converts the new CSV file (example: BNN4001LchartData.csv) into three images of charts: Iteration vs MSE, Time vs MSE and Time vs Iteration. After the VB script finishes the chart creation, the final function named finish is called. This function is simply printing "Finish" to the console indicating that the process of building, training and testing neural network has finish.

Neural network graphic as shown in Fig.3 will be popped in SIMBRAIN window after the whole processes finished. To validate the neural network with ValidateData.csv, user must double click the "Backprop" label on the left corner of the neural network and find "Validate Input Data" tab. After uploading the CSV file, user need to press the play button to go through the test data one by one. While validation of neural network occurs, the graphic in the SIMBRAIN application shows which neurons in hidden layers that affect result in output layer. The input layer has 28 neurons in width and 28 neurons in height, indicate the correct 28x28 pixels size of MNIST. As for the output layer, it has 10 neurons with 3 neurons in each row. The saturation of red color in neurons indicates the concentration of activation value in the neurons. As an example, if the user is testing a digit 3, the first neuron in the second row of output layer should be highly concentrated in red color to show a correct indication that the current tested digit number is 3. There is also a possibility that other neurons in the output layer concentration in red color due to the fact that the neural network might learn the current digit as another digit number.

III. Result and Discussion

This paper will be discussing on the manipulated variables of the neural networks, which are number of neurons, number of hidden layer and addition of Noise Out pruning. It is mentioned before that backpropagation algorithm, momentum, and learning rate of 0.01 are kept as constant variable. The results of final training MSE, total testing MSE and elapsed time in neural networks are printed on console and CSV files. These results are then collected and converted into line charts to analyze the trend of MSE. This section is divided into two parts, Part 1: Effect of number of neurons and number of hidden layer and Part 2: Effect of Addition of Noise Out Pruning.

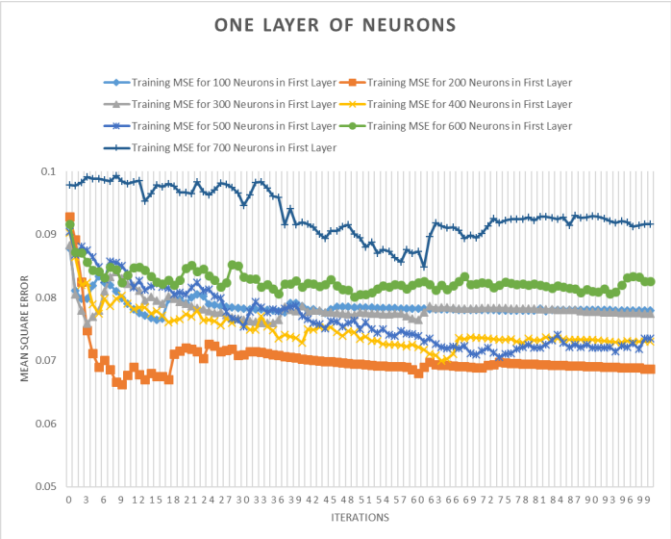

Figure 4 shows result of MSE while training neural network with one hidden layer

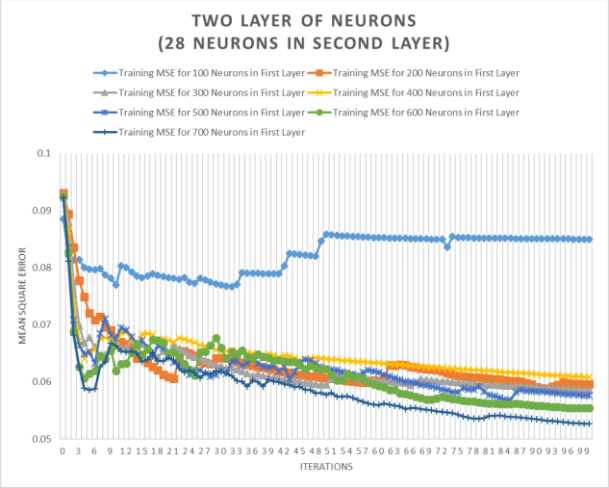

Figure 5 shows result of MSE while training neural network with two hidden layers

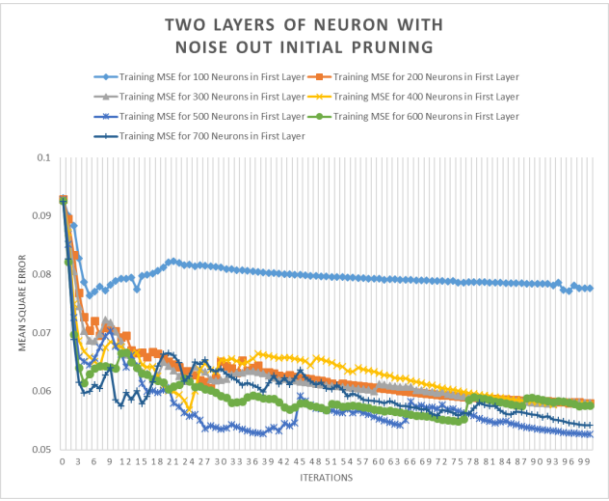

Figure 6 shows result of MSE while training neural network with two hidden layers and Noise Out initial pruning

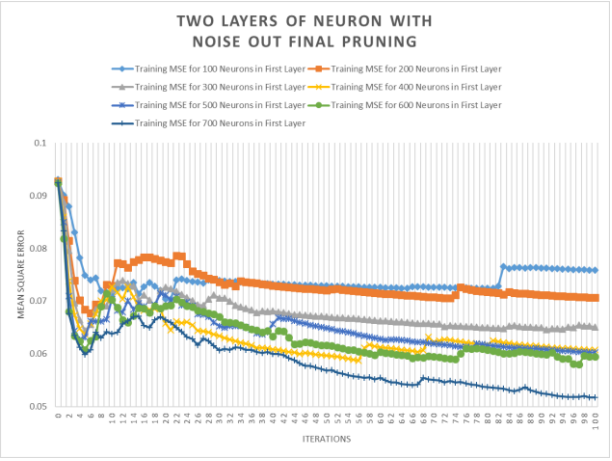

Figure 7 shows result of MSE while training neural network with two hidden layers and Noise Out final pruning

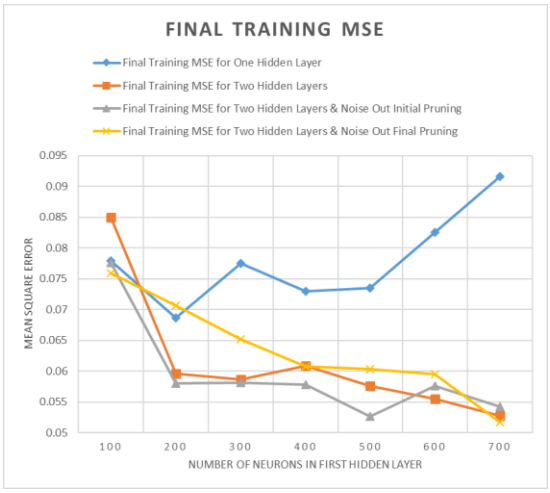

Figure 8 shows the result of final training MSE for all different neural network architecture

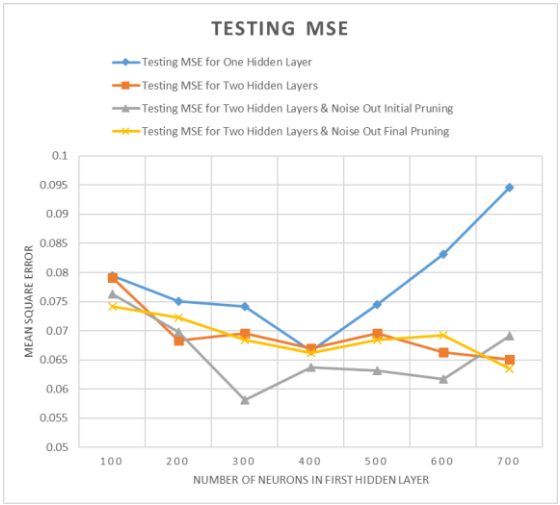

Figure 9 shows the result of testing MSE for all different neural network architecture

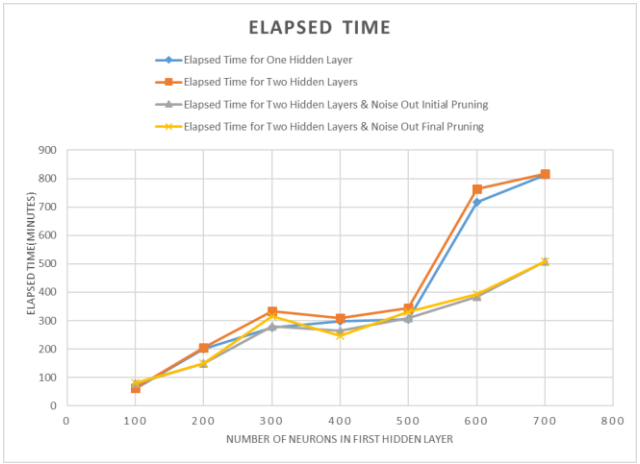

Figure 10 shows the result of elapsed time for all different neural network architecture

Table 1 shows the performance of neural network by calculating the difference between neural network with only one hidden layer and neural network with two hidden layers plus Noise Out

	One Hidden Layer	Two Hidden Layers and Noise Out Final Pruning	Performance
Final Training MSE	0.091648206	0.051732505	77%
Testing MSE	0.09457	0.063448041	49%
Elapsed Time	812.4977	508.5942	60%

Part 1: Effect of number of neurons and number of hidden layers

Figure 4 shows result of MSE while training neural network with one hidden layer. From this figure, it can be seen that the training MSE has no real pattern and it is unstable. Contrary to the result of training MSE in Figure 5, additional hidden layer reduces the training MSE with the lowest being at the architecture with 700 neurons in first hidden layer. It is observed when the size of neurons in increasing, the trend of the MSE can be seen to decline sharply indicating good accuracy.

The final training MSE and testing MSE for neural network with two hidden layers are smaller than the result from neural network with one hidden layer, yet, it requires more time to complete the training.

Part 2: Effect on Addition of Noise Out Pruning

The Noise Out pruning method is divided into two parts, Noise Out initial pruning and Noise Out final pruning. The initial pruning only applies the basic of Noise Out algorithm, which is to remove a neuron in hidden layer that contains the highest activation value while iterating the training process. The final pruning completes the Noise Out algorithm by inserting additional noise neuron in the output with a sigmoidal function. Figure 7 shows the result of applying initial Noise Out pruning on neural network with two hidden layers. From this graph, it visualizes that the training MSE becomes smoother compare to the one in Figure 6. However, this phenomena changes in Figure 7 which hold the results of Noise Out final pruning. In Figure 7, it is clearly seen that there is a bounce of data in the early iterations. The training MSE with Noise Out final pruning is larger than initial pruning except for the neural network with 700 neurons in its first layer. This indicates that since the number of iterations is 100, the Noise Out final pruning will be effective on neural network with large number of neurons in first hidden layer. This pattern is also seen at the testing MSE as shown in Figure 9. The main purpose of Noise Out algorithm is proven with Figure 10 where it has reduced the elapsed time for neural network to train. Based on the Table 1, the performance of applying two hidden layers and Noise Out pruning affects the performance on Final Training MSE by 77%, Testing MSE by 49% and Elapsed time by 60%.

IV. Conclusion

It is concluded that the MSE decreased when the number of neurons and number of hidden layers increased. The Noise Out pruning method decreased the elapsed time, yet the training MSE is still quite large except for the neural network with large number of neurons in its hidden layer. A suggestion in the future is that to let neural network be observed by changing the learning rate, learning momentum, and Dataset. Therefore, the aim to create neural network simulator with current available algorithm is finally achieved.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," nature, vol. 521, no. 7553, p. 436, 2015.

[2] M. A. Nielsen, Neural networks and deep learning, vol. 25. Determination press San Francisco, CA, USA:, 2015.

[3] J. Schmidhuber, "Deep learning in neural networks: An overview," Neural Networks, vol. 61, pp. 85–117, 2015.

[4] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," Proceedings of the IEEE, vol. 105, no. 12, pp. 2295–2329, 2017.

[5] W. G. Hatcher and W. Yu, "A survey of deep learning: platforms, applications and emerging research trends," IEEE Access, vol. 6, pp. 24411–24432, 2018.

[6] D. Silver, 2016.

[7] L. Li, Y. Lv, and F.-Y. Wang, "Traffic signal timing via deep reinforcement learning," IEEE/CAA Journal of Automatica Sinica, vol. 3, no. 3, pp. 247–254, 2016.

[8] A. Ashiquzzaman and A. K. Tushar, "Handwritten Arabic numeral recognition using deep learning neural networks," in 2017 IEEE International Conference on Imaging, Vision \& Pattern Recognition (icIVPR), 2017, pp. 1–4.

[9] F. Raffaeli and S. Awad, "Adaptive Speech Synthesis and training using low-cost AI Computing," in 2018 14th International Computer Engineering Conference (ICENCO), 2018, pp. 32–35.

[10] T. Hill, L. Marquez, M. O'Connor, and W. Remus, "Artificial neural network models for forecasting and decision making," International journal of forecasting, vol. 10, no. 1, pp. 5–15, 1994.

[11] A. Moncada, W. Richardson, and R. Vega-Avila, "Deep Learning to Forecast Solar Irradiance Using a Six-Month UTSA SkyImager Dataset," Energies, vol. 11, no. 8, p. 1988, 2018.

[12] R. Carbonneau, K. Laframboise, and R. Vahidov, "Application of machine learning techniques for supply chain demand forecasting," European Journal of Operational Research, vol. 184, no. 3, pp. 1140–1154, 2008.

[13] S. Hell and V. Argyriou, "Machine learning architectures to predict motion sickness using a Virtual Reality rollercoaster simulation tool," in 2018 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR), 2018, pp. 153–156.

[14] G. Chartrand, P. M. Cheng, E. Vorontsov, M. Drozdzal, S. Turcotte, C. J. Pal, S. Kadoury, and A. Tang, "Deep learning: a primer for radiologists," Radiographics, vol. 37, no. 7, pp. 2113–2131, 2017.

[15] M. Adams, W. Chen, D. Holcdorf, M. W. McCusker, P. D. Howe, and F. Gaillard, "Computer vs human: Deep learning versus perceptual training for the detection of neck of femur fractures," Journal of medical imaging and radiation oncology, vol. 63, no. 1, pp. 27–32, 2019.

[16] M. Xin, "Research on image classification model based on deep convolution neural network," 2019.

[17] A. Palvanov and Y. Im Cho, "Comparisons of deep learning algorithms for MNIST in real-time environment," International Journal of Fuzzy Logic and Intelligent Systems, vol. 18, no. 2, pp. 126–134, 2018.

[18] P. Y. Simard, D. Steinkraus, J. C. Platt, and others, "Best practices for convolutional neural networks applied to visual document analysis.," in Icdar, 2003, vol. 3, no. 2003.

[19] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," arXiv preprint arXiv:1506.00019, 2015.

[20] R. Priya, X. Wang, Y. Hu, and Y. Sun, "A Deep Dive into Automatic Code Generation Using Character Based Recurrent Neural Networks," in 2017 International Conference on Computational Science and Computational Intelligence (CSCI), 2017, pp. 369–374.

[21] D. Wang and Y. Shang, "Modeling physiological data with deep belief networks," International journal of information and education technology (IJIET), vol. 3, no. 5, p. 505, 2013.

[22] A. M. Abdel-Zaher, "Breast cancer classification using deep belief networks," 2015.

[23] M. Fatahi, M. Ahmadi, M. Shahsavari, A. Ahmadi, and P. Devienne, "evt_MNIST: A spike based version of traditional MNIST," arXiv preprint arXiv:1604.06751, 2016.

[24] J. H. Lee, "Training Deep Spiking Neural Networks Using Backpropagation."

[25] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," arXiv preprint arXiv:1602.02830, 2016.

[26] B. Kim and Y. H. Park, "Beginner's guide to neural networks for the MNIST dataset using MATLAB," The Korean Journal of Mathematics, vol. 26, no. 2, pp. 337–348, 2018.

[27] M. H. Beale, M. T. Hagan, and H. B. Demuth, "Neural network toolbox," User's Guide, MathWorks, vol. 2, pp. 77–81, 2010.

[28] I. Corona, B. Biggio, and D. Maiorca, "AdversariaLib: An open-source library for the security evaluation of machine learning algorithms under attack," arXiv preprint arXiv:1611.04786, 2016.

[29] B. Gong, B. Jou, X. Y. Felix, and S.-F. Chang, "Tamp: A Library for Compact Deep Neural Networks with Structured Matrices.," in ACM Multimedia, 2016, vol. 4322.

[30] J. Jin, "A c++ library for multimodal deep learning," arXiv preprint arXiv:1512.06927, 2015.

[31] A. Tahmassebi, "ideeple: Deep learning in a flash," in Disruptive Technologies in Information Sciences, 2018, vol. 10652, p. 106520S.

[32] C. J. Beckham, "Classification and regression algorithms for WEKA implemented in Python," 2015.

[33] S. Raschka, Python machine learning. Packt Publishing Ltd, 2015.

[34] J. Heaton, "Programming Neural Networks with Encog3 in Java 1st Edition, Heaton Research," 2011.

[35] J. Heaton, "Encog: library of interchangeable machine learning models for Java and C\#.," Journal of Machine Learning Research, vol. 16, pp. 1243–1247, 2015.

[36] V. Kovalev, A. Kalinovsky, and S. Kovalev, "Deep learning with theano, torch, caffe, tensorflow, and deeplearning4j: Which one is the best in speed and accuracy?," 2016.

[37] Y. Kochura, S. Stirenko, O. Alienin, M. Novotarskiy, and Y. Gordienko, "Comparative analysis of open source frameworks for machine learning with use case in single-threaded and multi-threaded modes," in 2017 12th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT), 2017, vol. 1, pp. 373–376.

[38] A. Shatnawi, G. Al-Bdour, R. Al-Qurran, and M. Al-Ayyoub, "A comparative study of open source deep learning frameworks," in 2018 9th International Conference on Information and Communication Systems (ICICS), 2018, pp. 72–77.

[39] D. L. Rohde, LENS: The light, efficient network simulator. School of Computer Science, Carnegie Mellon University, 1999.

[40] D. F. Goodman, "Code generation: a strategy for neural network simulators," Neuroinformatics, vol. 8, no. 3, pp. 183–196, 2010.

[41] R. A. Tikidji-Hamburyan, "Software for Brain Network Simulations: A Comparative Study," 2017.

[42] J. S. Sherfey, "DynaSim: A MATLAB Toolbox for Neural Modeling and Simulation," 2018.

[43] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," The Journal of Machine Learning Research, vol. 15, no. 1, pp. 1929–1958, 2014.

[44] M. Babaeizadeh, "NoiseOut: A Simple Way to Prune Neural Networks," 2016.

[45] J. Yoshimi, "Simbrain: A visual framework for neural network analysis and education," Brains, Minds, and Media, vol. 3, p. 27, 2008.

[46] Z. Tosi and J. Yoshimi, "Simbrain 3.0: a flexible, visually-oriented neural network simulator," Neural Networks, vol. 83, pp. 1–10, 2016.