Comparison Between Summation Units and Product Units Autoencoders Using the Backpropagation Algorithm

[1]Dan'abba Abdullahi Umar, [2]Salihu Aish Abdulkarim

[1,2]Department of Computer Science,
Federal University Dutse, Nigeria.
Dutse, Nigeria.

Email address: danabba1@gmail.com

Abstract
Artificial neural networks (ANN) or neural networks (NN) is an information processing system that mimics the structure and operating principles found in the information processing systems of human beings. The ANN is developed and derived to have a function similar to the human brain by memorizing and learning various tasks and behaving accordingly. Multi-layer perceptrons are the widely used class of NN in which neurons are arranged in layers, with the outputs of one layer being used as inputs to the next layer. In this research work, we trained summation units and product units autoencoders with three different image datasets using the backpropagation (BP) algorithm. Afterward, the results of the two autoencoders were then compared (accuracy performances) after training both with the datasets for a hundred epochs each one at a time. The overall training results show that the summation units autoencoder outperforms the product units autoencoder. Thus, the summation units autoencoder should be considered when training autoencoders with backpropagation algorithms with image datasets like MNIST handwritten dataset, Fashion-MNIST dataset, and Cifar-10 small images dataset.

Keywords: Artificial Neural Networks, Autoencoders, backpropagation, Summation Units, Product Units.

INTRODUCTION

ANN or NN is an information processing system that mimics the structure and operating principles found in the information processing systems of human beings. The ANN architecture is developed to have a function similar to the human brain and to behave accordingly as far as functionality and the inter-neuron connection are concerned [1]. Similarly, the NN is made up of neurons (nodes) and synapses (connections) that link neurons together. The neuron's output is computed by aggregating its inputs (i.e., as a weighted sum) and activating the result by applying an activation function such as **tanh**. Multi-layer perceptrons are the widely used class of NNs in which neurons are arranged in layers, with the outputs of one layer being used as inputs to the next layer. The main reasons for the NN investigations are: to try to get an

understanding of how the human brain functions, and the desire to build machines that are capable of solving complex problems that sequentially operating computers were unable to solve [2].

BACKGROUND OF THE RESEARCH

The study of NN is one of the most rapidly expanding fields attracting researchers from a wide variety of disciplines [3]. The NN has been applied successfully in many applications areas like speech recognition, handwritten character recognition, steering of an autonomous vehicle, radar target detection, classification, and medical diagnosis of heart attacks [4-7]. Many ANN types had advantages in improving system performance for broad applications, including some for dimension reduction, and classification. For example, [8] trained a product network to determine the optimum width of transistors in a CMOS switch. The authors show that the product networks can be successfully trained with genetic algorithms (GA), and how penalty functions can assist in avoiding local minima. [9], proposed a P-S model which is a BP net that contains only product units in the hidden layer and all summing units in other layers. [10], trained product units neural networks (PUNNs) using a cooperative particle swarm optimizer (CPSO) which is a variant of the particle swarm optimizer (PSO) that splits the problem vector. The CPSO outperforms the standard PSO consistently and the performance of the CPSO improves as the split factor is increased until a critical ratio (i.e. number of weights divides by the number of swarms) is reached. [11], presented a new method for regression based on the evolution of a type of FFNNs whose basis function units are products of the inputs raised to real number power. The performance of the model was evaluated in five widely used benchmark functions and a hard real-world problem of microbial growth modeling. [12], developed an efficient hybrid algorithm with both local and global search capabilities based on the Levenberg–Marquardt (L–M) algorithm and the particle swarm optimizer. In the proposed method, learning process takes place without the risk of being trapped at local minima or jumping over the best solution. [13], discussed the practical details of the creation of a deep convolutional auto-encoder in the very popular Caffe deep learning framework. The developed models provide very good results in dimensionality reduction and unsupervised clustering tasks. [14], employed a deep autoencoder network to analyze numerous accuracy measures, including data completeness, positional accuracy, shape accuracy, orientation consistency, and semantic accuracy, and to obtain a more comprehensive and objective quality assessment result that outperformed other methods. [15], introduced an adversarial variational Bayes (AVB), which was a technique for training variational autoencoders (VAEs) with arbitrarily expressive inference models. The studies show a clear theoretical justification and retain most of the advantages of standard variational autoencoders, and are easy to implement. [16], applied l_2

normalization constraint to deep autoencoder representation. The proposed model cluster well in the Euclidean space and applying a simple k-means clustering on the representation produces high clustering accuracies and works well than another method defining additional clustering losses. [17], introduced a new autoencoder-based algorithm for non-negative matrix factorization. The author explored the potential of the proposed model for clustering applications and showed that it can learn hierarchical factorizations, each of which corresponds to a different and meaningful clustering. [18], proposed a novel manifold Galerkin and manifold LSPG projection techniques that project dynamical-system models onto arbitrary continuously-differentiable nonlinear manifolds. In this current research work, we trained summation units and product units autoencoders with three different image datasets with the BP training algorithm. Then we compared the results after training the two autoencoders with each of the datasets for a hundred epochs. This paper is organized as follows: the methodology section that discusses the datasets, the BP algorithm, the summation units and the product units NN, and the autoencoders. Next is the results and conclusion section, and the references section.

METHODOLOGY

This section will discuss the datasets used in training the NN models (autoencoders), the training algorithm, and the mode of conducting the research work. First of all, the datasets used in conducting this research work were the MNIST handwritten dataset, the Fashion-MNIST dataset, and the CIFAR-10 small images classification dataset which are all available and downloadable online.

MNIST Handwritten Dataset

The word MNIST is an acronym that stands for Mixed/Modified National Institute of Standards and Technology (MNIST). The MNIST dataset originated from the National Institute of Standards and Technology (NIST) [19]. The MNIST database is a collection of images of handwritten digits zero to nine inclusive. The training set consists of handwritten numbers from 250 different people, of which 50% were high school students and 50% were from the Census Bureau. The test set is also the same proportion of handwritten digital data. The MNIST dataset totally contains 60,000 images in the training set or samples and 10,000 patterns in the testing set or samples.

Fashion-MNIST Dataset

The Fashion-MNIST dataset consists of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28 x 28 grayscale image, associated with a label from 10 classes. The Fashion-MNIST dataset was intended to serve as a direct drop-in replacement for the original MNIST handwritten dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits. Members of the AI, ML, and Data Science community love this dataset and use it as a benchmark to validate their algorithms.

CIFAR-10 Dataset

The word CIFAR stands for Canadian Institute for Advanced Research, 10 classes **(CIFAR-10).** The CIFAR-10 dataset consists of 60,000 with 32 x 32 coloured images in 10 different classes of objects like cats, planes, and cars, with 6,000 images per class. There are 50,000 training images and 10,000 test images. The dataset is divided into five training batches and one test batch, each with 10,000 images. The test batch contains exactly 1,000 randomly-selected images from each class. The training batches contain the remaining images in random order. However, some training batches may contain more images from one class than another. The training batches contain exactly 5,000 images from each class.

Backpropagation Algorithm

The term backpropagation is a short form for "backward propagation of errors". The BP algorithm employs gradient descent (GD) to attempt to minimize the square error between the network output values and the target values for these outputs. The method involves propagating the error signal backwards through the network after a forward pass, by computing the gradient for each synaptic link and nodal bias using the chain rule [12]. According to [20], the BP algorithm contains two main phases, referred to as the forward and backward phases respectively.The forward phase is required to compute the output values and the local derivatives at various nodes, while the backward phase is required to accumulate the products of these local values over all paths from the node to the output. The BP algorithm works in the following manners as illustrated in figure 1. Firstly, inputs X arrive through the pre-connected path which is then modeled using real weights W which is usually randomly selected. Then calculate the output for every neuron from the input layer to the hidden layers, and to the output layer. Also, calculate the error in the outputs i.e. error = actual output – desired output. Lastly, travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased. This process is repeated until the desired output is achieved.

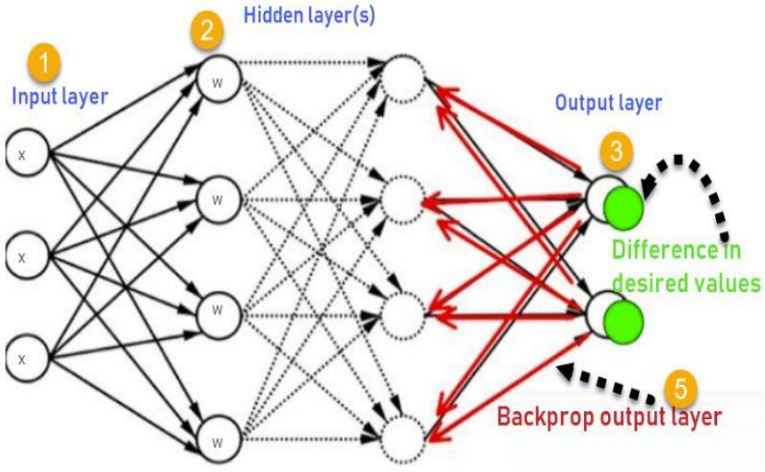

Figure 1: How Backpropagation Algorithm works [21]

Summation Units Neural Networks (SUNNs)

The SUNNs are traditionally composed of multiple layers of summation units and these simple units sum their inputs, where each input is multiplied by a variable weight. The summation is usually then squashed by a non-linear equation such as the logistic function [8]. A standard multilayer NN uses summation units in the hidden and output layers to compute the net input signal to units. The net input signal is a weighted sum of the inputs connected to that unit [22]. Networks composed of these units can approximate any continuous function to an arbitrary degree of accuracy; provided that the hidden layers contain a sufficient number of hidden units. However, there are some functions that are complicated enough that the number of summation units it takes to duplicate them is prohibitive [8].

Product Units Neural Networks (PUNNs)

The PUNNs is a special type of ANN that provide a method of automatically learning the higher-order input combinations required for efficient learning in NN and are an alternative to classical NN [10, 23]. In the PUNNs, the hidden layer summation units are replaced by product units (PUs) to compute the weight product of inputs as thus;

$$net_{yj} = \prod_{i=1}^{\mathrm{I}} z_i{}^{v_{ji}} \dots\dots\dots\dots\dots\dots\dots (1)$$

Instead of

$$net_{yj} = \sum_{i=1}^{\mathrm{I}} z_i v_{ji} \dots\dots\dots\dots\dots\dots\dots\dots . (2)$$

Where net_{yj} is the net input to the hidden unit y_j, z_i is an input unit, v_{ji} is the weight between hidden unit y_j and input z_i, and I is the total number of input units (including a bias unit to the hidden layer). The main advantages of product-unit-based NN are increased information capacity and the ability to form higher-order combinations of the inputs [24]. The increased information capacity of the PUNNs is measured by their capacity for learning random Boolean patterns compared to standard summation networks. When Boolean inputs are processed in product units, the best performance is obtained by using inputs of {+1, -I} [25]. If the imaginary component is ignored, with these inputs, the activation function is equivalent to a cosine summation function with {-1,+1} inputs mapped {I,D}. The PUNNs is approximately 3N, compared to 2N for a single threshold logic function, where N is the number of inputs to the unit. However, networks based on product units have more local minima and more probability of becoming trapped in the local minima [11].

AUTOENCODERS

Autoencoders are ANN models that learned efficient data coding in an unsupervised manner. The

autoencoders consist of three components i.e. the encoder that takes the input and converts it into something readable to the model, the code or latent representation of the encoded data or inputs, and the decoder that reads the magic combination of number and transform it into something similar to the input. Thus, the autoencoders learn how to efficiently compress and encode input data and then learn how to reconstruct the data back from the reduced encoded representation to a representation that is as close to the original inputs as possible. The goal of an autoencoder is to learn a representation for a set of data, usually for dimensionality reduction by training the network to ignore signal noise. Along with the reduction side, a reconstructing side is also learned, where the autoencoder tries to generate from the reduced encoding a representation as close as possible to its original input [26]. The following are some of the common basic types of autoencoders: denoising autoencoder, sparse autoencoder, deep autoencoder, contractive autoencoder, undercomplete autoencoder, convolutional autoencoder, and variational autoencoder [27,28]. Basically, the autoencoders are constructed using summation units in their hidden layers. This means that the summation units is by default invaded the autoencoders structure. Therefore, in this context, we will be referring to the autoencoders that have the summation units in their hidden layers as the summation units autoencoder and those have the product units in their hidden layers as the product units autoencoders. Figure 2 presented a simple structure of an autoencoder.

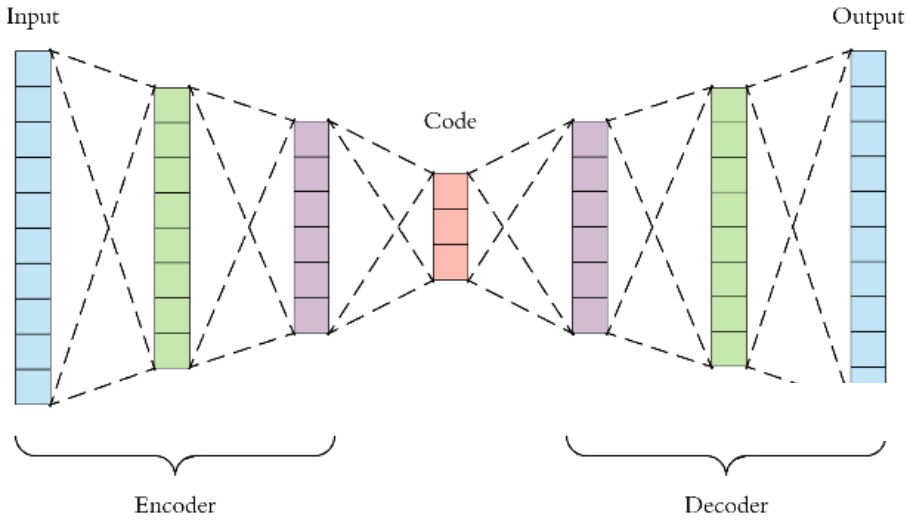

Figure 2: Structure of Autoencoder [29]

This research work was conducted in the Google Colaboratory or simply Google colab. The Google colab is a free notebook environment that runs on

cloud-provided by Google which includes GPU support as well. It is an executable document that let you write, run, and share code within the Google Drive. We can write or execute codes in Python, create, upload, share, import, publish as well as export external datasets from other sources. Colab connects your notebook (a notebook is composed of cells, each of which can contain code, text, images, and more) to a cloud-based runtime, meaning you can execute Python code without required any setup on your own machine. Similarly, the BP gradient-based algorithm written in the Python programming language was the training algorithm used in training our autoencoders NN models with the three datasets.

Thus, we begin by defining a multilayer perceptron class (MLP) constructor that takes the number of inputs, a variable number of the hidden layer(s), and a number of output(s). Then created a generic representation of the layers and a random connection of weights for the layers and then save the derivatives and activation per layer. Defined the forward propagations, and compute the forward propagation of the network based on input signals, and save the activation for backpropagation. Iterate through the network layers, and calculate matrix multiplication between the previous activation and weight matrix. Next we applied the activation function (sigmoid), and return the output layers. Back propagate the error signal i.e. iterate backward through the network layers. Then get activation for the previous layer and apply the sigmoid activation function, and then get activation for the current layer. Reshape activations to have them as a 2d column matrix and save derivative after applying matrix multiplication. Back propagate the next error and then defined and train the NN models (i.e. the autoencoders). We trained the autoencoders with the datasets for one hundred complete cycles (epochs) each with a learning rate (i.e. step to apply to gradient descent) of 0.001 running forward and backward propagation. We've one epoch when the entire dataset is passed forward and backward through the NN only once. Firstly, we trained the summation units autoencoder with the MNIST handwritten dataset. We specified the size of the encoded representation i.e. encoding dimension, compression factor and the input place holder (784 = 28 x 28), and then the decoded representation which is just the reconstruction of the input. We normalize all values between 0 and 1 and flatten the 28 x 28 images into vectors of size 784 by dividing the training and testing dataset by 255. Enter the training loop and iterate through all the training datasets. Perform gradient descent on the derivatives and keep track of mean square error (MSE) for reporting later (i.e. epoch complete), report the training error. Then update the weights by stepping down the gradient. Lastly, we compiled the program, viewed and tabulated the results.

The same process was repeated with the MNIST-Fashion dataset and then with the CIFAR-10 small image datasets. Consequently, the same procedure was carried out with the product units autoencoder with the

three datasets. Figure 3 shows the schematic diagram of the training process, while figures 4, 5, and 6 shows the graphical comparisons representations between the two autoencoders with respect to the three datasets. The following are the full meaning of the terms as they appeared in figure 3:

- BP: Backpropagation
- AU: Autoencoder
- SUNNS: summation units neural networks
- PUNNS: product units neural networks
- MNIST: MNIST handwritten dataset
- F-MNIST: Fashion-MNIST dataset.
- CIFAR-10: CIFAR-10 small images dataset.
- SUNNS AU + MNIST: training of the summation units autoencoder with the MNIST handwritten dataset.
- SUNNS AU + F-MNIST: training of the summation units autoencoder with the fashion-MNIST dataset.
- SUNNS AU + CIFAR-10: training of the summation units autoencoder with CIFAR-10 small images dataset.
- PUNNS + MNIST: training of the product units autoencoder with the MNIST handwritten dataset.
- PUNNS + F-MNIST: training of the product units autoencoder with the fashion- MNIST dataset.
- PUNNS + CIFAR-10: training of the product units autoencoder with CIFAR-10 small images dataset.
- SUNNS AU + MNIST VS PUNNS + MNIST: comparison between the training results of the summation units autoencoder with the MNIST handwritten dataset and the product units autoencoder with the MNIST handwritten dataset.
- SUNNS AU +F-MNIST VS PUNNS + F-MNIST: comparison between the training results of the summation units autoencoder with the Fashion-MNIST dataset and the product units autoencoder with the Fashion-MNIST dataset.

- SUNNS AU + CIFAR-10 VS PUNNS + CIFAR-10: comparison between the training results of the summation units autoencoder with the CIFAR-10 small images dataset and the product units autoencoder with the CIFAR-10 small images dataset.

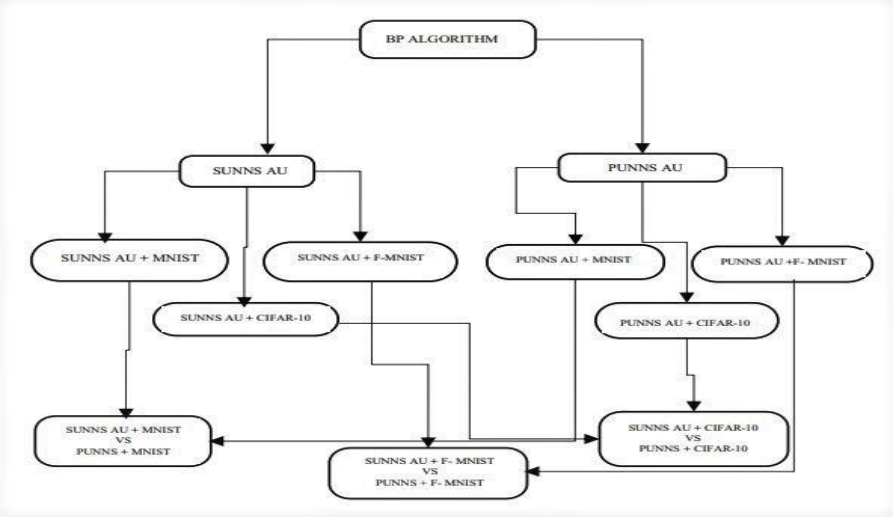

Figure 3: shows the schematic diagram of the training process

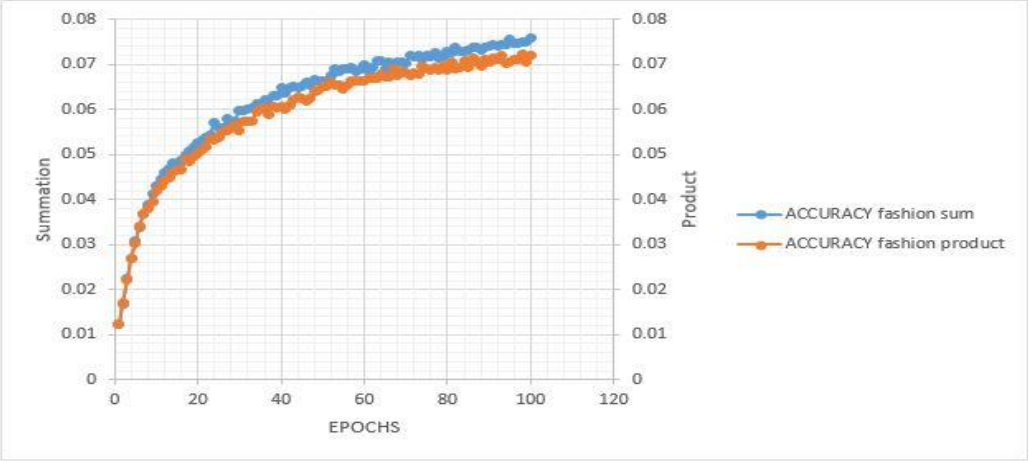

Figure 4: SUNNS AU + MNIST VS PUNNS AU + MNIST ACCURACIES

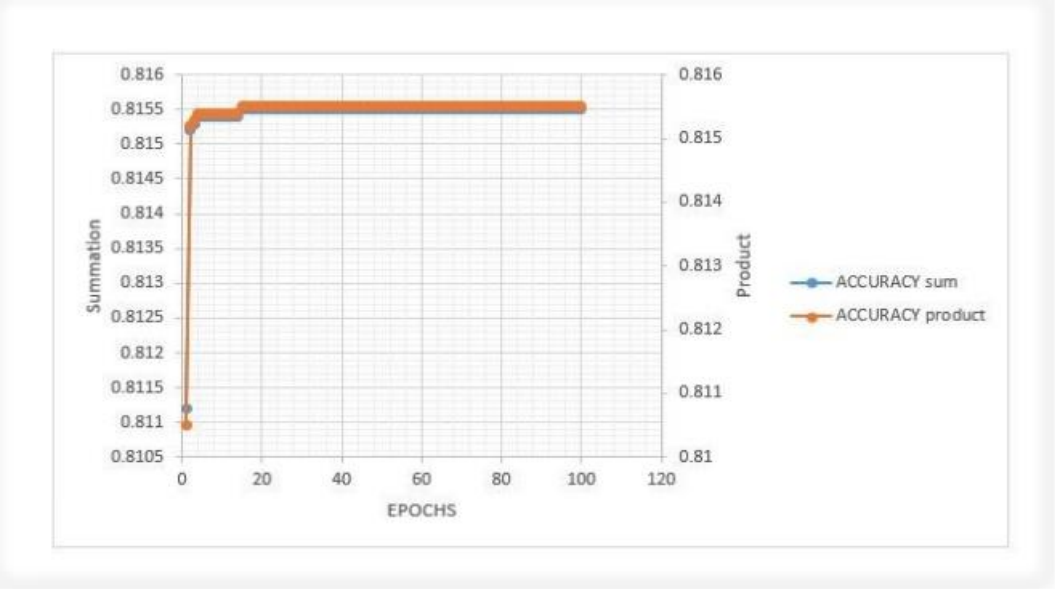

Figure 5: SUNNS AU + F-MNIST VS PUNNS AU + F-MNIST ACCURACIES

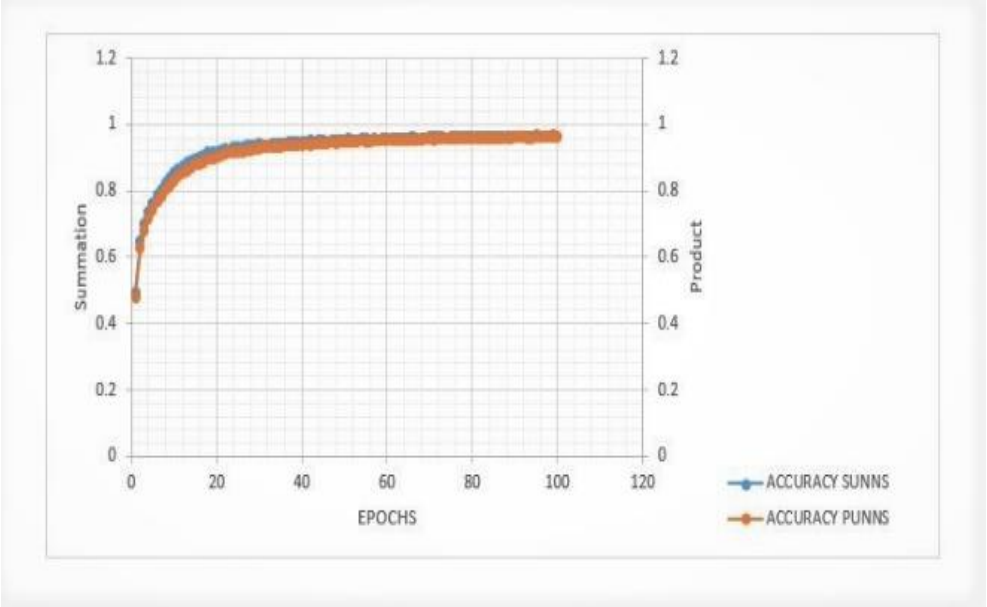

Figure 6: SUNNS AU + CIFAR-10 VS PUNNS AU + CIFAR-10 ACCURACIES

RESULTS AND DISCUSSION

The following tables show the summaries of the results of training our autoencoders NN models.

Table 1: Summary for summation units autoencoder training with three datasets

ACCURACIES FOR SUMMATION UNITS AUTOENCODER			
EPOCH	MNIST HANDWRITTEN	FASHION-MNIST	CIFAR-10
1	0.8112	0.0122	0.4952
2	0.8152	0.0167	0.6486
3	0.8153	0.0224	0.7007
4	0.8154	0.0270	0.7375
5	0.8154	0.0307	0.7638
--	-------	-------	------
10	0.8154	0.0430	0.8533
---	--------	--------	-------
---	-----	-------	-----
---	--------	--------	--------
97	0.8155	0.0746	0.9654
98	0.8155	0.0749	0.9659
99	0.8155	0.0750	0.9665
100	0.8155	0.0760	0.9660

Table 2: Summary for product units autoencoder training with three datasets

ACCURACIES FOR PRODUCT UNITS AUTOENCODERS			
EPOCH	MNIST HANDWRITTEN	FASHION-MNIST	CIFAR-10
1	0.8105	0.0124	0.4769
2	0.8152	0.0172	0.6264
3	0.8153	0.0223	0.6813
4	0.8154	0.0269	0.7157
5	0.8154	0.0303	0.7431
--	---------	-------	--------
10	0.8154	0.0420	0.8304
---	-------	-------	-------
---	------	-----	------
---	--------	-------	------
97	0.8155	0.0711	0.9631
98	0.8155	0.0723	0.9617
99	0.8155	0.0704	0.9616
100	0.8155	0.0720	0.9634

Looking at tables 1 and 2, it turned out that the summation units autoencoder has higher accuracies results than the product units autoencoder. This implies that the summation units autoencoder is more robust than the product units autoencoder when trained with the image datasets like the MNIST handwritten dataset, the Fashion-MINST dataset, and the CIFAR-10 small images dataset. Also at this point, we have achieved the aim of carrying out this research work; which was comparison between the summation units and the product units autoencoders. Moreover, according to the works' [30] and [31] (as cited in [32]), and [33] the reasons for the poor

performance of the product units autoencoder was due to the weight initialization and secondly, the search space for the product units can be extremely convoluted with numerous local minima that trap the gradient descent algorithm.

CONCLUSION

In this paper, we trained the summation units and the product units autoencoders for the purpose of comparison (accuracy performances) between the two autoencoders. It turns out that the summation units autoencoder outperforms the product units autoencoder with the BP training algorithm. Therefore, in summary, the summation units autoencoder should be considered when training image datasets like the MNIST handwritten dataset, the Fashion-MNIST dataset, and the CIFAR-10 small image dataset with backpropagation algorithm as the training algorithm. Lastly, the particle swarm optimization (PSO) algorithm can be used as the training algorithm for further research. This is due to the PSO features i.e. the inertia weight, the maximum velocity, the acceleration constants, and the size of the population.

REFERENCES

[1] Bataineh, Mohammad Hindi (2012). "Artificial neural network for studying human performance." MS (Master of Science) thesis, University of Iowa, 2012. https://doi.org/10.17077/etd.r6q1np0x

[2]Muller, J. Reinhardt, and M.T. Strickland (1995). Neural networks: an introduction. Number v. 1 in Physics of neural networks. Springer, 1995. ISBN 9783540602071. http://books.google.ie/books?id=EFUzMYjOXk8C

[3]A Ismail and A P Engelbrecht (1999). Training Product units in Feedforward Neural Networks using Particle Swarm Optimization. In Proceedings of the International Conference on Artificial Intelligence, pages 36–40, 1999.

[4] Guyon, I. (1991). Neural networks and applications tutorial. *Physics Reports*, *207*(3-5), 215-259.

[5] Harrison, R. F., Marshall, S. J., & Kennedy, R. L. (1991, July). The early diagnosis of heart attacks: a neuro computational approach. In *IJCNN-91-Seattle International Joint Conference on Neural Networks* (Vol. 1, pp. 1-5). IEEE.

[6] Cohen, M., Franco, H., Morgan, N., Rumelhart, D., & Abrash, V. (1992). (Modified: 16 Jul 2019), Context-dependent multiple distribution phonetic modeling with MLPs. *Advances in Neural Information Processing Systems*, *5*.

[7] Haykin, S., & Network, N. (2004). A comprehensive foundation. *Neural networks*, *2*(2004), 41.

[8] Janson, D. J., & Frenzel, J. F. (1993). Training product unit neural networks with genetic

algorithms. *IEEE Expert*, *8*(5), 26-33.

[9] Wang, J. H., & Lin, J. H. (1995, October). Qualitative analysis of the BP composed of product units and summing units. In *1995 IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century* (Vol. 1, pp. 35-39). IEEE.

[10] Bergh, F. V. D., & Engelbrecht, A. P. (2001, July). Effects of swarm size on cooperative particle swarm Optimisers. In *Proceedings of the 3rd annual conference on genetic and evolutionary computation* (pp. 892-899).

[11] Martínez-Estudillo, A., Martínez-Estudillo, F., Hervás-Martínez, C., & García-Pedrajas, N. (2006). Evolutionary product unit based neural networks for regression. *Neural Networks*, *19*(4), 477-486.

[12] Ismail, A., Jeng, D. S., & Zhang, L. L. (2013). An optimised product-unit neural network with a novel PSO–BP hybrid training algorithm: Applications to load–deformation analysis of axially loaded piles. *Engineering applications of artificial intelligence*, *26*(10), 2305-2314.

[13] Turchenko, V., Chalmers, E., & Luczak, A. (2017). A deep convolutional auto-encoder with pooling-unpooling layers in Caffe. *arXiv preprint arXiv:1701.04949*.

[14] Yongyang Xu, Zhanlong Chen, Zhong Xie & Liang Wu (2017). Quality assessment of building footprint data using a deep autoencoder network. *International Journal of Geographical Information Science*. http://dx.doi.org/10.1080/13658816.2017.1341632.

[15] Lars Meschedar, Sebastian Nowozin & Andreas Geiger (2018). Adversarial VariationalBayes: Unifying Variational Autoencoders and Generative AdversarialNetworks. arXiv:1701.04722v4 [cs.LG] 11 Jun 2018.

[16] Caglar Aytekin, Xingyang Ni, Francesco Cricri & Emre Aksu (2018). Clustering and UnsupervisedAnomaly Detection with l_2 Normalized Deep Auto-Encoder Representations.*Cambridge, Massachusetts London, England.*

[17] El Khatib, A., Huang, S., Ghodsi, A., & Karray, F. (2018, July). Nonnegative matrix factorization using autoencoders and exponentiated gradient descent. In *2018 International Joint Conference on Neural Networks (IJCNN)* (pp. 1-8). IEEE.

[18] Kookjin Lee, Kevin T. Carlberg, (2020). Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders, Journal of Computational Physics, Volume 404, 2020, 108973, ISSN 0021-9991, https://doi.org/10.1016/j.jcp.2019.108973.(https://www.sciencedirect.com/science/article/pii/S0021999119306783)

[19] Molnar, Christoph (2019). "Interpretable machine learning. A Guide for Making Black Box Models Explainable", 2019. https://christophm.github.io/interpretable-ml-book/.

[20] Khan, M., Jan, B., & Farman, H. (2019). *Deep learning: convergence to big data analytics* (p. 93). Singapore: Springer. https://doi.org/10.1007/978-981-13-3459-7_1

[21] Daniel Johnson: Back Propagation in Neural Network: Machine Learning Algorithm (2022). https://www.guru99.com/backpropogation-neural-network.html

[22] Ismail, A., & Engelbrecht, A. P. (2000, July). Global optimization algorithms for training product unit neural networks. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium* (Vol. 1, pp. 132-137). IEEE.

[23] David Gries Fred B. Schneider (2008).Fundamentals of the New Artificial Intelligence (2nd Ed.) *Springer London Ltd.*

[24] Guerrero-Enamorado, A., & Ceballos-Gastell, D. (2016). An experimental study of evolutionary product-unit neural network algorithm. *Computación y Sistemas*, *20*(2), 205-218.

[25]Arisara Pornwattanavichai, Prawpan Brahmasakha na sakolnagara, Pongsakorn Jirachanchaisiri, Janekhwan Kitsupapaisan&Saranya Maneero (2019). Enhanced Tweet Hybrid Recommender System Using Unsupervised Topic Modeling and Matrix Factorization-Based Neural Network. Michael W. Berry, Azlinah Mohamed & Bee Wah Yap (Eds.), Supervised and Unsupervised Learning for Data Science. *Springer Nature Switzerland AG 2020*.

[26] Charu C. Aggarwal (2018). Neural Networks and Deep Learning. (first ed.) https://link.springer.com/chapter/10.1007/978-3-319-94463-0_2

[27] Hubens Nathan (2020). Deep inside Autoencoders. https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f

[28] Bandyopadhyay Hmrishav (2022). Autoencoders in Deep Learning: Tutorial & Use Cases [2022] https://www.v7labs.com/blog/autoencoders-guide

[29] Dertat Arden (2017). Applied Deep Learning – Part 3: Autoencoders. https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

[30] Lapedes, A. & Farber, R. (1987). Nonlinear signal processing using neural networks: Prediction and system modelling. Technical Report LA – UR- 87- 2662, Los Alamos National Laboratory, Los Alams, NM.

[31] Durbin R., Rumelhart D.E. (1990). Product Units with Trainable Exponents and Multi-Layer

Networks. In Neurocomputing. NATO ASI Series (Series F: Computer and Systems Sciences), vol 68. *Springer, Berlin, Heidelberg*. https://doi.org/10.1007/978-3642-76153-9_2

[32] Leerink, Laurens & Giles, C. & Horne, William & Jabri, Marwan. (1996). Learning with Product Units. 7.

[33] Martínez, Cesar & Martínez-Estudillo, Francisco & Gutiérrez, Pedro Antonio. (2006). Classification by means of Evolutionary Product-Unit Neural Networks. 1525 - 1532. 10.1109/IJCNN.2006.246614.