Deep Learning using Rectified Linear Units (ReLU)

Abien Fred M. Agarap
Department of Computer Science
Adamson University
Manila, Philippines
abien.fred.agarap@adamson.edu.ph

ABSTRACT

We introduce the use of rectified linear units (ReLU) as the classification function in a deep neural network (DNN). Conventionally, ReLU is used as an activation function in DNNs, with Softmax function as their classification function. However, there have been several studies on using a classification function other than Softmax, and this study is an addition to those. We accomplish this by taking the activation of the penultimate layer h_{n-1} in a neural network, then multiply it by weight parameters θ to get the raw scores o_i. Afterwards, we threshold the raw scores o_i by 0, i.e. $f(o) = \max(0, o_i)$, where $f(o)$ is the ReLU function. We provide class predictions $\hat{y}$ through arg max function, i.e. arg max $f(x)$.

CCS CONCEPTS

• **Computing methodologies → Supervised learning by classification**; **Neural networks**;

KEYWORDS

artificial intelligence; artificial neural networks; classification; convolutional neural network; deep learning; deep neural networks; feed-forward neural network; machine learning; rectified linear units; softmax; supervised learning

1 INTRODUCTION

A number of studies that use deep learning approaches have claimed state-of-the-art performances in a considerable number of tasks such as image classification[9], natural language processing[15], speech recognition[5], and text classification[18]. These deep learning models employ the conventional softmax function as the classification layer.

However, there have been several studies[2, 3, 12] on using a classification function other than Softmax, and this study is yet another addition to those.

In this paper, we introduce the use of rectified linear units (ReLU) at the classification layer of a deep learning model. This approach is the novelty presented in this study, i.e. ReLU is conventionally used as an activation function for the hidden layers in a deep neural network. We accomplish this by taking the activation of the penultimate layer in a neural network, then use it to learn the weight parameters of the ReLU classification layer through backpropagation.

We demonstrate and compare the predictive performance of DL-ReLU models with DL-Softmax models on MNIST[10], Fashion-MNIST[17], and Wisconsin Diagnostic Breast Cancer (WDBC)[16] classification. We use the Adam[8] optimization algorithm for learning the network weight parameters.

2 METHODOLOGY

2.1 Machine Intelligence Library

Keras[4] with Google TensorFlow[1] backend was used to implement the deep learning algorithms in this study, with the aid of other scientific computing libraries: matplotlib[7], numpy[14], and scikit-learn[11].

2.2 The Datasets

In this section, we describe the datasets used for the deep learning models used in the experiments.

2.2.1 MNIST. MNIST[10] is one of the established *standard* datasets for benchmarking deep learning models. It is a 10-class classification problem having 60,000 training examples, and 10,000 test cases – all in grayscale, with each image having a resolution of 28×28.

2.2.2 Fashion-MNIST. Xiao et al. (2017)[17] presented the new Fashion-MNIST dataset as an alternative to the conventional MNIST. The new dataset consists of 28×28 grayscale images of $70,000$ fashion products from 10 classes, with $7,000$ images per class.

2.2.3 Wisconsin Diagnostic Breast Cancer (WDBC). The WDBC dataset[16] consists of features which were computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. There are 569 data points in this dataset: (1) 212 – Malignant, and (2) 357 – Benign.

2.3 Data Preprocessing

We normalized the dataset features using Eq. 1,

$$z = \frac{X - \mu}{\sigma} \tag{1}$$

where X represents the dataset features, μ represents the mean value for each dataset feature $x^{(i)}$, and σ represents the corresponding standard deviation. This normalization technique was implemented using the `StandardScaler`[11] of scikit-learn.

For the case of MNIST and Fashion-MNIST, we employed Principal Component Analysis (PCA) for dimensionality reduction. That is, to select the representative features of image data. We accomplished this by using the `PCA`[11] of `scikit-learn`.

2.4 The Model

We implemented a feed-forward neural network (FFNN) and a convolutional neural network (CNN), both of which had two different classification functions, i.e. (1) softmax, and (2) ReLU.

2.4.1 Softmax. Deep learning solutions to classification problems usually employ the softmax function as their classification function (*last layer*). The softmax function specifies a discrete probability distribution for K classes, denoted by $\sum_{k=1}^{K} p_k$.

If we take **x** as the activation at the penultimate layer of a neural network, and θ as its weight parameters at the softmax layer, we have **o** as the input to the softmax layer,

$$o = \sum_{i}^{n-1} \theta_i x_i \qquad (2)$$

Consequently, we have

$$p_k = \frac{exp(o_k)}{\sum_{k=0}^{n-1} exp(o_k)} \qquad (3)$$

Hence, the predicted class would be $\hat{y}$

$$\hat{y} = \underset{i \in 1, \ldots, N}{\arg\max} \; p_i \qquad (4)$$

2.4.2 Rectified Linear Units (ReLU). ReLU is an activation function introduced by [6], which has strong biological and mathematical underpinning. In 2011, it was demonstrated to further improve training of deep neural networks. It works by thresholding values at 0, i.e. $f(x) = max(0, x)$. Simply put, it outputs 0 when $x < 0$, and conversely, it outputs a linear function when $x \geq 0$ (refer to Figure 1 for visual representation).
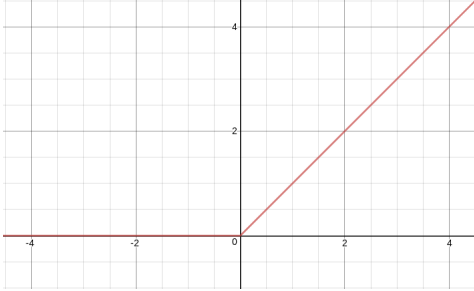

Figure 1: The Rectified Linear Unit (ReLU) activation function produces 0 **as an output when** $x < 0$**, and then produces a linear with slope of** 1 **when** $x > 0$**.**

We propose to use ReLU not only as an activation function in each hidden layer of a neural network, but also as the classification function at the last layer of a network.

Hence, the predicted class for ReLU classifier would be $\hat{y}$,

$$\hat{y} = \underset{i \in 1, \ldots, N}{\arg\max} \; max(0, o) \qquad (5)$$

2.4.3 Deep Learning using ReLU. ReLU is conventionally used as an activation function for neural networks, with softmax being their classification function. Then, such networks use the softmax cross-entropy function to learn the weight parameters θ of the neural network. In this paper, we still implemented the mentioned loss function, but with the distinction of using the ReLU for the

prediction units (see Eq. 6). The θ parameters are then learned by backpropagating the gradients from the ReLU classifier. To accomplish this, we differentiate the ReLU-based cross-entropy function (see Eq. 7) w.r.t. the activation of the penultimate layer,

$$\ell(\theta) = -\sum y \cdot log\big(max(0, \theta x + b)\big) \qquad (6)$$

Let the input **x** be replaced the penultimate activation output **h**,

$$\frac{\partial \ell(\theta)}{\partial \mathbf{h}} = -\frac{\theta \cdot y}{max\big(0, \theta h + b\big) \cdot \ln 10} \qquad (7)$$

The backpropagation algorithm (see Eq. 8) is the same as the conventional softmax-based deep neural network.

$$\frac{\partial \ell(\theta)}{\partial \theta} = \sum_{i} \left[\frac{\partial \ell(\theta)}{\partial p_i} \left(\sum_{k} \frac{\partial p_i}{\partial o_k} \frac{\partial o_k}{\partial \theta} \right) \right] \qquad (8)$$

Algorithm 1 shows the rudimentary gradient-descent algorithm for a DL-ReLU model.

Algorithm 1: Mini-batch stochastic gradient descent training of neural network with the rectified linear unit (ReLU) as its classification function.

Input: $\{x^{(i)} \in \mathbb{R}^m\}_{i=1}^{n}, \theta$
Output: W
for *number of training iterations* **do**
  for *i = 1, 2, \ldots n* **do**
$$\nabla_\theta = \nabla_\theta - \frac{\theta \cdot y}{max\big(0, \theta h + b\big) \cdot \ln 10}$$
  $\theta = \theta - \alpha \cdot \nabla_\theta \ell(\theta; x^{(i)})$
Any standard gradient-based learning algorithm may be used. We used adaptive momentum estimation (Adam) in our experiments.

In some experiments, we found the DL-ReLU models perform on par with the softmax-based models.

2.5 Data Analysis

To evaluate the performance of the DL-ReLU models, we employ the following metrics:

(1) Cross Validation Accuracy & Standard Deviation. The result of 10-fold CV experiments.
(2) Test Accuracy. The trained model performance on unseen data.
(3) Recall, Precision, and F1-score. The classification statistics on class predictions.
(4) Confusion Matrix. The table for describing classification performance.

3 EXPERIMENTS

All experiments in this study were conducted on a laptop computer with Intel Core(TM) i5-6300HQ CPU @ 2.30GHz x 4, 16GB of DDR3 RAM, and NVIDIA GeForce GTX 960M 4GB DDR5 GPU.

Table 1 shows the architecture of the VGG-like CNN (from

Keras[4]) used in the experiments. The last layer, dense_2, used the softmax classifier and ReLU classifier in the experiments.

The Softmax- and ReLU-based models had the same hyper-parameters, and it may be seen on the Jupyter Notebook found in the project repository: https://github.com/AFAgarap/relu-classifier.

Table 1: Architecture of VGG-like CNN from Keras[4].

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 14, 14, 32)	320
conv2d_2 (Conv2D)	(None, 12, 12, 32)	9248
max_pooling2d_1 (MaxPooling2)	(None, 6, 6, 32)	0
dropout_1 (Dropout)	(None, 6, 6, 32)	0
conv2d_3 (Conv2D)	(None, 4, 4, 64)	18496
conv2d_4 (Conv2D)	(None, 2, 2, 64)	36928
max_pooling2d_2 (MaxPooling2)	(None, 1, 1, 64)	0
dropout_2 (Dropout)	(None, 1, 1, 64)	0
flatten_1 (Flatten)	(None, 64)	0
dense_1 (Dense)	(None, 256)	16640
dropout_3 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 10)	2570

Table 2 shows the architecture of the feed-forward neural network used in the experiments. The last layer, dense_6, used the softmax classifier and ReLU classifier in the experiments.

Table 2: Architecture of FFNN.

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 512)	131584
dropout_4 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 512)	262656
dropout_5 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 512)	262656
dropout_6 (Dropout)	(None, 512)	0
dense_6 (Dense)	(None, 10)	5130

All models used Adam[8] optimization algorithm for training, with the default learning rate $\alpha = 1 \times 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-8}$, and no decay.

3.1 MNIST

We implemented both CNN and FFNN defined in Tables 1 and 2 on a normalized, and PCA-reduced features, i.e. from 28×28 (784) dimensions down to 16×16 (256) dimensions.

In training a FFNN with two hidden layers for MNIST classification, we found the results described in Table 3.

Despite the fact that the Softmax-based FFNN had a slightly higher test accuracy than the ReLU-based FFNN, both models had 0.98 for their F1-score. These results imply that the FFNN-ReLU is on par with the conventional FFNN-Softmax.

Figures 2 and 3 show the predictive performance of both models for MNIST classification on its 10 classes. Values of correct prediction in the matrices seem to be balanced, as in some classes,

Table 3: MNIST Classification. Comparison of FFNN-Softmax and FFNN-ReLU models in terms of % accuracy. The training cross validation is the average cross validation accuracy over 10 splits. Test accuracy is on unseen data. Precision, recall, and F1-score are on unseen data.

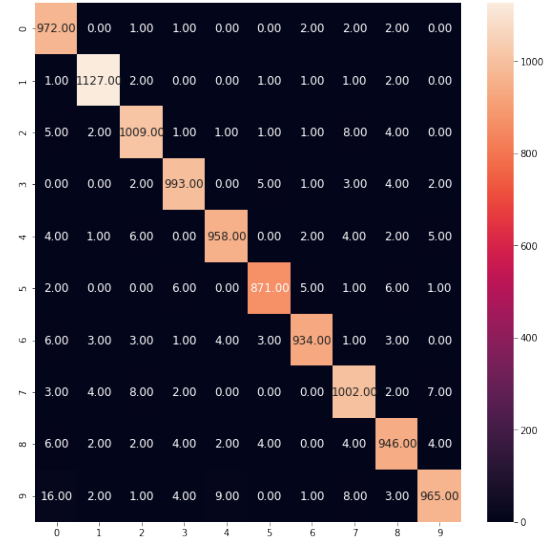| Metrics / Models	FFNN-Softmax	FFNN-ReLU
Training cross validation	$\approx 99.29\%$	$\approx 98.22\%$
Test accuracy	97.98%	97.77%
Precision	0.98	0.98
Recall	0.98	0.98
F1-score	0.98	0.98

Figure 2: Confusion matrix of FFNN-ReLU on MNIST classification.

the ReLU-based FFNN outperformed the Softmax-based FFNN, and vice-versa.

In training a VGG-like CNN[4] for MNIST classification, we found the results described in Table 4.

Table 4: MNIST Classification. Comparison of CNN-Softmax and CNN-ReLU models in terms of % accuracy. The training cross validation is the average cross validation accuracy over 10 splits. Test accuracy is on unseen data. Precision, recall, and F1-score are on unseen data.

Metrics / Models	CNN-Softmax	CNN-ReLU
Training cross validation	$\approx 97.23\%$	$\approx 73.53\%$
Test accuracy	95.36%	91.74%
Precision	0.95	0.92
Recall	0.95	0.92
F1-score	0.95	0.92

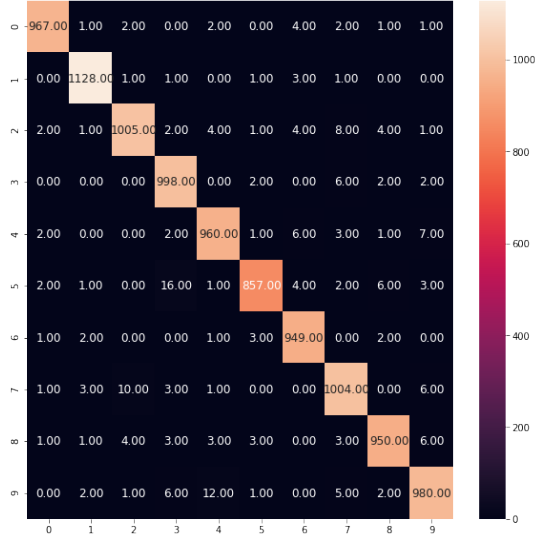**Figure 3: Confusion matrix of FFNN-Softmax on MNIST classification.**

Figure 4: Confusion matrix of CNN-ReLU on MNIST classification.

The CNN-ReLU was outperformed by the CNN-Softmax since it converged slower, as the training accuracies in cross validation were inspected (see Table 5). However, despite its slower convergence, it was able to achieve a test accuracy higher than 90%. Granted, it is lower than the test accuracy of CNN-Softmax by $\approx 4\%$, but further optimization may be done on the CNN-ReLU to achieve an on-par performance with the CNN-Softmax.

Table 5: Training accuracies and losses per fold in the 10-fold training cross validation for CNN-ReLU on MNIST Classification.

Fold #	Loss	Accuracy (×100%)
1	1.9060128301398311	0.32963837901722315
2	1.4318902588488513	0.5091768125718277
3	1.362783239967884	0.5942213337366827
4	0.8257899198037331	0.7495911319797827
5	1.222473526516734	0.7038720233118376
6	0.4512576775334098	0.8729090907790444
7	0.49083630082824015	0.8601818182685158
8	0.34528968995411613	0.9032199380288064
9	0.30161443973038743	0.912663755545276
10	0.279967466075669	0.9171823807790317

Figures 4 and 5 show the predictive performance of both models for MNIST classification on its 10 classes. Since the CNN-Softmax converged faster than CNN-ReLU, it has the most number of correct predictions per class.

3.2 Fashion-MNIST

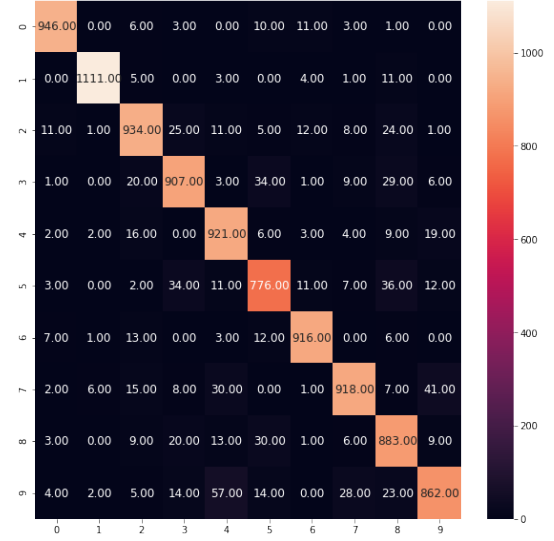We implemented both CNN and FFNN defined in Tables 1 and 2 on a normalized, and PCA-reduced features, i.e. from 28×28 (784)
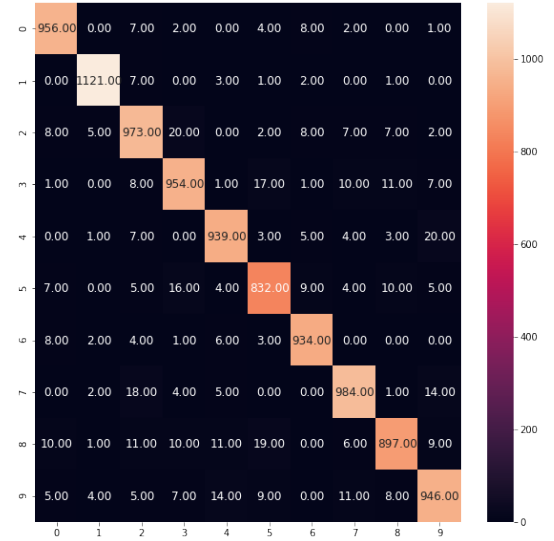

Figure 5: Confusion matrix of CNN-Softmax on MNIST classification.

dimensions down to 16×16 (256) dimensions. The dimensionality reduction for MNIST was the same for Fashion-MNIST for fair comparison. Though this discretion may be challenged for further investigation.

In training a FFNN with two hidden layers for Fashion-MNIST classification, we found the results described in Table 6.

Despite the fact that the Softmax-based FFNN had a slightly higher test accuracy than the ReLU-based FFNN, both models had 0.89 for their F1-score. These results imply that the FFNN-ReLU is on par with the conventional FFNN-Softmax.

Table 6: Fashion-MNIST Classification. Comparison of FFNN-Softmax and FFNN-ReLU models in terms of % accuracy. The training cross validation is the average cross validation accuracy over 10 splits. Test accuracy is on unseen data. Precision, recall, and F1-score are on unseen data.

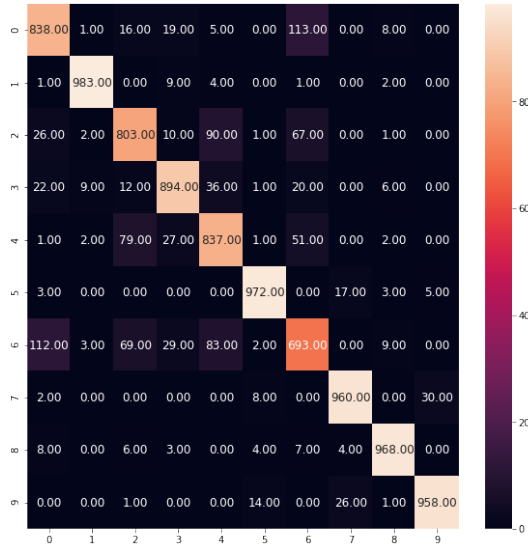| Metrics / Models	FFNN-Softmax	FFNN-ReLU
Training cross validation	≈ 98.87%	≈ 92.23%
Test accuracy	89.35%	89.06%
Precision	0.89	0.89
Recall	0.89	0.89
F1-score	0.89	0.89

Figure 6: Confusion matrix of FFNN-ReLU on Fashion-MNIST classification.

Figures 6 and 7 show the predictive performance of both models for Fashion-MNIST classification on its 10 classes. Values of correct prediction in the matrices seem to be balanced, as in some classes, the ReLU-based FFNN outperformed the Softmax-based FFNN, and vice-versa.

In training a VGG-like CNN[4] for Fashion-MNIST classification, we found the results described in Table 7.

Similar to the findings in MNIST classification, the CNN-ReLU was outperformed by the CNN-Softmax since it converged slower, as the training accuracies in cross validation were inspected (see Table 8). Despite its slightly lower test accuracy, the CNN-ReLU had the same F1-score of 0.86 with CNN-Softmax – also similar to the findings in MNIST classification.

Figures 8 and 9 show the predictive performance of both models for Fashion-MNIST classification on its 10 classes. Contrary to the findings of MNIST classification, CNN-ReLU had the most number of correct predictions per class. Conversely, with its faster convergence, CNN-Softmax had the higher cumulative correct predictions per class.
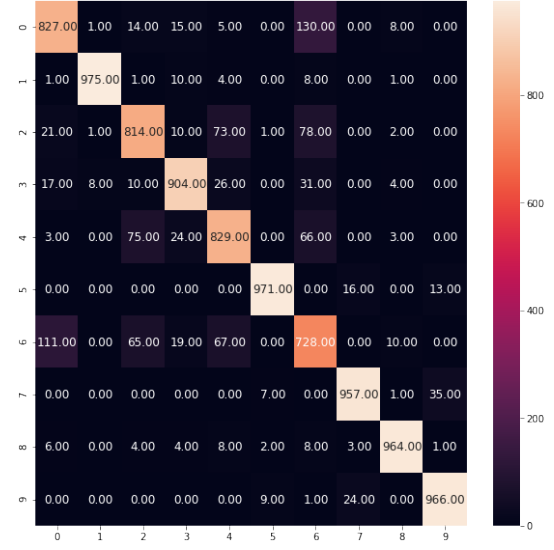

Figure 7: Confusion matrix of FFNN-Softmax on Fashion-MNIST classification.

Table 7: Fashion-MNIST Classification. Comparison of CNN-Softmax and CNN-ReLU models in terms of % accuracy. The training cross validation is the average cross validation accuracy over 10 splits. Test accuracy is on unseen data. Precision, recall, and F1-score are on unseen data.

Metrics / Models	CNN-Softmax	CNN-ReLU
Training cross validation	≈ 91.96%	≈ 83.24%
Test accuracy	86.08%	85.84%
Precision	0.86	0.86
Recall	0.86	0.86
F1-score	0.86	0.86

Table 8: Training accuracies and losses per fold in the 10-fold training cross validation for CNN-ReLU for Fashion-MNIST classification.

| Fold #	Loss	Accuracy (×100%)
1	0.7505188028133193	0.7309229651162791
2	0.6294445606858231	0.7821584302325582
3	0.5530192871624917	0.8128293656488342
4	0.468552251288519	0.8391494002614356
5	0.4499297190579501	0.8409090909090909
6	0.45004472223195163	0.8499999999566512
7	0.4096944159454683	0.855610110994295
8	0.39893951664539995	0.8681098779960613
9	0.37760543597664203	0.8637190683266308
10	0.34610279169377683	0.8804367606156083
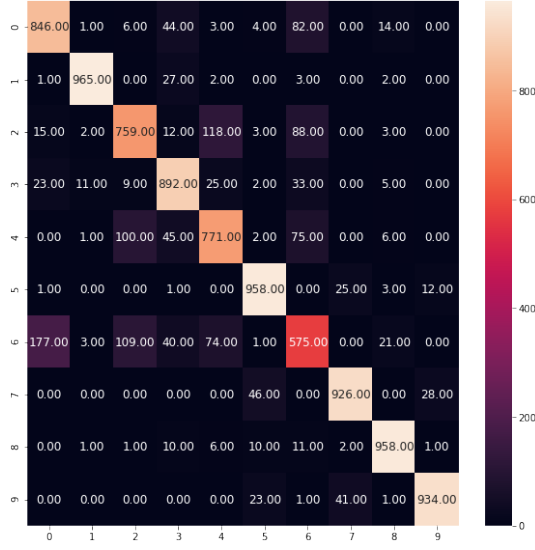
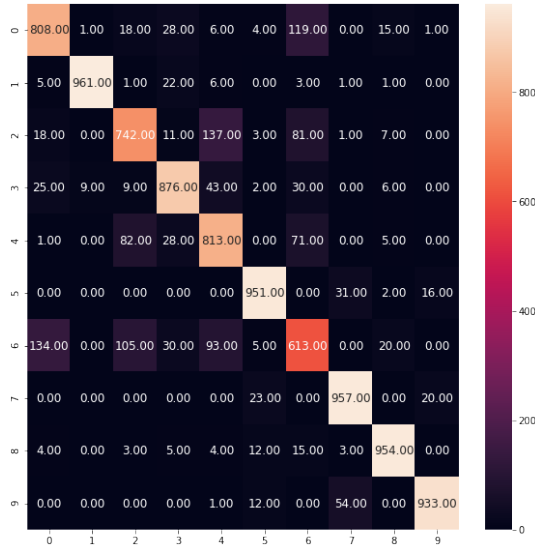**Figure 8: Confusion matrix of CNN-ReLU on Fashion-MNIST classification.**

Figure 9: Confusion matrix of CNN-Softmax on Fashion-MNIST classification.

3.3 WDBC

We implemented FFNN defined in Table 2, but with hidden layers having 64 neurons followed by 32 neurons instead of two hidden layers both having 512 neurons. For the WDBC classfication, we only normalized the dataset features. PCA dimensionality reduction might not prove to be prolific since WDBC has only 30 features.

In training the FFNN with two hidden layers of [64, 32] neurons, we found the results described in Table 9.

Table 9: WDBC Classification. Comparison of CNN-Softmax and CNN-ReLU models in terms of % accuracy. The training cross validation is the average cross validation accuracy over 10 splits. Test accuracy is on unseen data. Precision, recall, and F1-score are on unseen data.

Metrics / Models	FFNN-Softmax	FFNN-ReLU
Training cross validation	$\approx 91.21\%$	$\approx 87.96\%$
Test accuracy	$\approx 92.40\%$	$\approx 90.64\%$
Precision	0.92	0.91
Recall	0.92	0.91
F1-score	0.92	0.90

Similar to the findings in classification using CNN-based models, the FFNN-ReLU was outperformed by the FFNN-Softmax in WDBC classification. Consistent with the CNN-based models, the FFNN-ReLU suffered from slower convergence than the FFNN-Softmax. However, there was only 0.2 F1-score difference between them. It stands to reason that the FFNN-ReLU is still comparable with FFNN-Softmax.
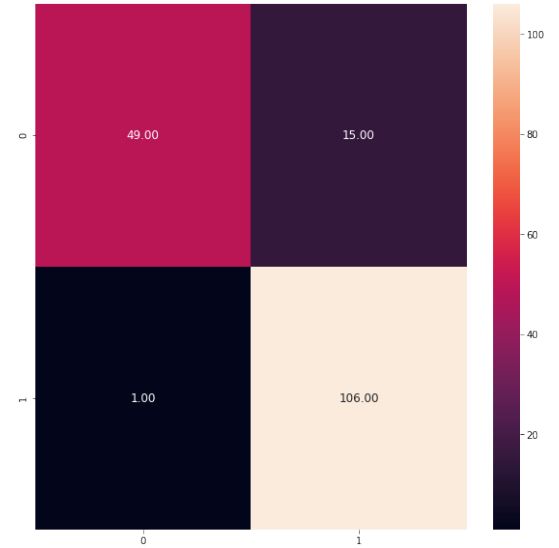

Figure 10: Confusion matrix of FFNN-ReLU on WDBC classification.

Figures 10 and 11 show the predictive performance of both models for WDBC classification on binary classification. The confusion matrices show that the FFNN-Softmax had more false negatives than FFNN-ReLU. Conversely, FFNN-ReLU had more false positives than FFNN-Softmax.

4 CONCLUSION AND RECOMMENDATION

The relatively unfavorable findings on DL-ReLU models is most probably due to the *dying neurons* problem in ReLU. That is, no gradients flow backward through the neurons, and so, the neurons become stuck, then eventually "die". In effect, this impedes the
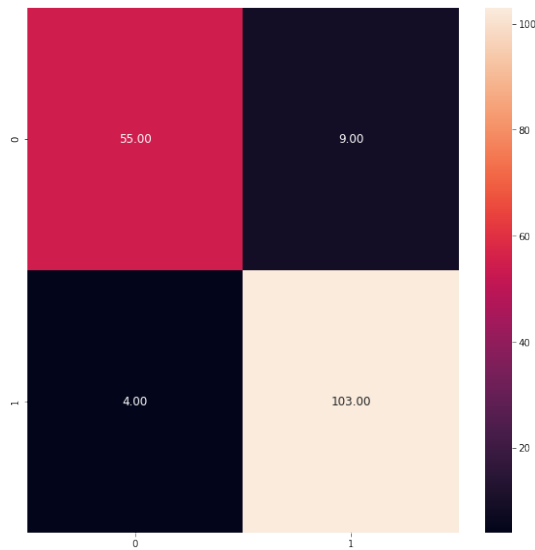
Figure 11: Confusion matrix of FFNN-Softmax on WDBC classification.

learning progress of a neural network. This problem is addressed in subsequent improvements on ReLU (e.g. [13]). Aside from such drawback, it may be stated that DL-ReLU models are still comparable to, if not better than, the conventional Softmax-based DL models. This is supported by the findings in DNN-ReLU for image classification using MNIST and Fashion-MNIST.

Future work may be done on thorough investigation of DL-ReLU models through numerical inspection of gradients during backpropagation, i.e. compare the gradients in DL-ReLU models with the gradients in DL-Softmax models. Furthermore, ReLU variants may be brought into the table for additional comparison.

5 ACKNOWLEDGMENT

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). http://tensorflow.org/ Software available from tensorflow.org.
[2] Abien Fred Agarap. 2017. A Neural Network Architecture Combining Gated Recurrent Unit (GRU) and Support Vector Machine (SVM) for Intrusion Detection in Network Traffic Data. *arXiv preprint arXiv:1709.03082* (2017).
[3] Abdulrahman Alalshekmubarak and Leslie S Smith. 2013. A novel approach combining recurrent neural network and support vector machines for time series classification. In *Innovations in Information Technology (IIT), 2013 9th International Conference on*. IEEE, 42–47.
[4] François Chollet et al. 2015. Keras. https://github.com/keras-team/keras. (2015).
[5] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. 2015. Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems*. 577–585.
[6] Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. 2000. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature* 405, 6789 (2000), 947.
[7] J. D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering* 9, 3 (2007), 90–95. https://doi.org/10.1109/MCSE.2007.55
[8] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.
[10] Yann LeCun, Corinna Cortes, and Christopher JC Burges. 2010. MNIST handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist* 2 (2010).
[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[12] Yichuan Tang. 2013. Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239* (2013).
[13] Ludovic Trottier, Philippe Gigu, Brahim Chaib-draa, et al. 2017. Parametric exponential linear unit for deep convolutional neural networks. In *Machine Learning and Applications (ICMLA), 2017 16th IEEE International Conference on*. IEEE, 207–214.
[14] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. 2011. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering* 13, 2 (2011), 22–30.
[15] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745* (2015).
[16] William H Wolberg, W Nick Street, and Olvi L Mangasarian. 1992. Breast cancer Wisconsin (diagnostic) data set. *UCI Machine Learning Repository [http://archive. ics. uci. edu/ml/]* (1992).
[17] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. (2017). arXiv:cs.LG/1708.07747
[18] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alexander J Smola, and Eduard H Hovy. 2016. Hierarchical Attention Networks for Document Classification.. In *HLT-NAACL*. 1480–1489.