Artificial Neural Network Performance Boost using Probabilistic Recovery with Fast Cascade Training

Andreas Maniatopoulos, Alexandros Gazis, Venitis P. Palikaras, Nikolaos Mitianoudis
Democritus University of Thrace
Xanthi, 67100
Greece

Received: January 1, 2020. Revised: July 24, 2020. 2nd Revised: November 16, 2020. Accepted: November 16, 2020. Accepted: November 23, 2020. Published: November 24, 2020.

Abstract— Pattern Recognition and Classification is considered one of the most promising applications in the scientific field of Artificial Neural Networks (ANN). However, regardless of the vast scientific advances in almost every aspect of the technology and mathematics, neural networks still need to be fairly large and complex (i.e., deep), in order to provide robust results. In this article, we propose a novel ANN architecture approach that aims to combine two fairly small Neural Networks based on an introduced probability term of correct classification. Additionally, we present a second ANN, used to reclassify the potentially incorrect results by using the most probable error-free results as additional training data with the predicted labels. The proposed method achieves a rapid decrease in the mean square error compared to other large and complex ANN architectures with a similar execution time. Our approach demonstrates increased effectiveness when applied to various databases, related to wine, iris, the Modified National Institute of Standards and Technology (MNIST) database, the Canadian Institute for Advanced Research (Cifar32), and Fashion MNIST classification problems.

Keywords— pattern recognition, artificial intelligence, classification techniques, Machine learning complexity optimization, time reduction.

I. Introduction

DATA classification is frequently performed by Artificial Neural Networks (ANN) like [1], [2], [3] consisting of a number of nodes, termed neurons. These neurons are organized in layers. This first layer is the input layer, the last layer is the output layer, while the middle layers are termed hidden layers. In a feed-forward ANN, the output of each neuron is calculated by a weighted combination of the outputs of the neurons of the previous layer. This output is also passed through a non-linear function, called the activation function. The weights of these neurons are estimated by minimizing the mean square error between the ANN's outputs to set input-output pairs (training data). This procedure is also known as back propagation and is usually performed by iterative batch optimization, by iterating over the training data multiple times, called epochs, until the error is minimized. This is the training phase of the network. By quantizing the ANN's output to highlight the output neuron with greatest value as ones and the rest as zeros, we can use ANNs to perform classification [4]. In the testing phase, the trained ANN is used to classify new, never seen, samples [5]. Novel cost functions (i.e. cross-entropy) and activation functions like Rectified Linear Unit (ReLU) have been recently proposed to facilitate the training of deeper networks [6], [7], and [8].

Historically, classification accuracy mainly depends on the size of the Neural Network, and the number of epochs of network training. More specifically, choosing a small number of layers/neurons may not provide the network with the required degrees of freedom to discriminate between the various data classes in complex classification problems. In contrast, a larger number of layers/neurons may lead the network to overfit. This article examines the classification accuracy of an Artificial Neural Network (ANN) architecture

that uses two smaller ANN for probabilistic fast training to benchmark the properties of an ANN [9]. Recent advances in the field shift their efforts into applying probabilistic neural networks with deep learning techniques to education [10], medicine [11,12], image recognition [13], power cost [14], and real life applications like autonomous driving [15], memristors [16], electrical machines [17], and crowd monitoring [18].

Our ANN's method takes inspiration from these types of ANNs but, the technical novelty of our architecture is that we use the probability to evaluate the outcome of the system. This means that to achieve a high accuracy rate, during each batch, an algorithm must evaluate an ANN's output, and instantly recognize potential misclassifications. Many researchers have focused their efforts on result evaluation in correlation with ANNs' performance such as hyperparameter tuning [19] and changing statistics [20]. Furthermore, as most ANNs' input datasets are usually continuous data streams many scientists have researched extensible multiple parallel predictions ANNs for raw data streams [21]. Analytically, for an ANN to make multiple predictions on the fly, it must be capable of evaluating the likelihood of each prediction through a distribution of possible future outcomes [22]. The first ANNs to use this approach were hidden Markov models [23], deep learning generative models [24], and Autoregressive Integrated Moving Average forecasting methods [25]. In more recent studies like in [26], researchers study short term forecasting optimization of weight parameters by using complex ANN architectures such as multi-layer feed-forward network and recurrent networks.

Similarly, our study focuses on reducing the computational and memory cost of neural network topologies, like in [27]. Specifically, this is achieved by using traditional ANN models and activation functions but by introducing a novel ANN architecture that evaluates the ANN output using a probabilistic algorithmic approach. Our architecture, except for minimizing the ANN's memory requirements [28], it sets a trade-off between preserving the ANN's accuracy ratio and optimizing the system's performance. Lastly, in the next sections, we present the aims of this study, the test/training phases of our experiments, and the proposed algorithm for ANN optimization.

II. Aims and Objectives

In [29], Bishop mentions that classification errors occur in the regions of parameter x space, where the largest of all posterior probabilities of sample x belonging to the k-th class C_k is relatively low. The main reason for this phenomenon is that there is a strong overlap between different classes. In several applications, it might be better not to make a classification decision in such cases. This technique is called reject option, where the rejected samples are classified by a human expert instead. The reject-option technique can be described by the following algorithm:

$$if\ max_k P(C_k \mid x)) \geq \Theta,\ then\ classify\ x,\ else\ reject\ x \quad (1)$$

where Θ defines a rejection threshold.

The larger the value of Θ, the fewer points will be classified by the ANN and will need to be labelled by a human expert. However, the current explosion of available information and unlabeled data renders human intervention an unviable option. In addition, an automated definition of Θ, based on the given dataset, will also be required.

In this article, we propose a direct extension of the reject option-idea, where no human intervention will be needed to classify the rejected data. Instead, another ANN, trained using the successfully trained samples of the first ANN, will take on the role of the human expert and will reclassify the rejected data. More specifically, we propose to improve the performance of traditional ANN architectures, by adding a probability term that evaluates the classification results, dividing the dataset into possibly correct and incorrect data.

The "correctly" classified data are determined by the probability function, and by using the corresponding predicted labels, are selected to exist alongside the original dataset to perform a second round of training of another ANN. The outcome is that if "incorrect" results occur, these are reclassified by the second ANN thus providing dynamic feedback to the proposed system.

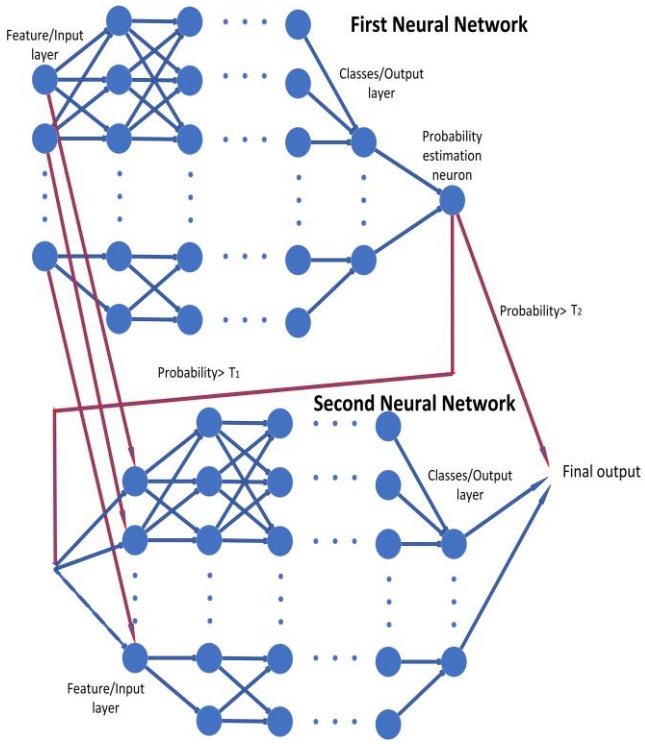

Fig. 1 The proposed ANN architecture, consisting of two ANNs connected back to back

The proposed scheme features improved performance compared to the original architecture, without the extra computational cost of larger architectures. In the following section, we will analyze the proposed approach. Objective

evaluation for each of the tested datasets indicated an improvement offered to the original architecture with minimal computational cost.

III. The Proposed Algorithm

The concept presented aims at increasing the classification performance of common ANN architectures (deep or shallow) using a simple strategy. The proposed probabilistic fast cascade training consists of two ANN connected back to back, sharing input data. The second ANN has extra training inputs those testing samples that achieved high probability of correct classification. Thus, the second ANN aims at reclassifying only the most-likely wrong results of the first ANN, while leaving the correctly classified data unchanged.

All experiments conducted were developed using MATLAB's Neural Network toolbox and specifically Deep Learning toolbox that provides a framework for designing, analyzing, and implementing deep neural networks with algorithms, pre-trained models, and applications [30]. Additionally, the production code of the ANN proposed was also written in Python mainly consuming Keras, a deep learning application protocol interface (API) used for machine learning and deep learning experimentation [31].

Lastly, the architecture of our ANN is shown in Figure 1 where the red lines indicate results or data points that are transferred from the first network to the second. Let us examine this architecture with more detail during the training and testing phase of the two networks.

A. Training Phase

At first, a feed-forward ANN is used to classify coarsely the test samples. This ANN is trained and evaluated as an independent neural network, using the training dataset that is available. This part is the common training any ANN architecture (deep or shallow) would require. The output of this ANN is V_{est}. This concludes the first part of the training.

The second phase of the training would entail the training of a second neural network. A typical method would train the network using the training dataset. Using a combination of the two networks, we can employ more data for training the second ANN. We can use the first, coarse ANN, which has already been trained, to roughly classify the data from the testing dataset. Thus, we can use the data from the training dataset plus data from the testing dataset that have been correctly classified.

To achieve this, we need a formula to compute the probability of correct classification. This is used to determine, whether the aforementioned classification is acceptable or not. In the positive case, they will be used as future training data, other- wise they will be reclassified again by the second ANN, once it is trained. The proposed correct classification probability formula is the following:

$$p(i) = e^{-(1-softmax_{\max}(i))2log(2)} \quad (2)$$

where i represents the index of each sample and $softmax_{max}$ represents the maximum activation of the last softmax neuron layer of the ANN.

Since the final classification is usually based on softmax, the probability function must be equal to 0.5 in the middle of the two nearest classes. The aim of the function is to provide logarithmic 1-1 mapping in the interval [0.5, 1], whereas it is less strict in the interval [0, 0.5], where the samples are considered improperly classified anyway.

The training of the first ANN is performed for a fixed number of epochs. After training of the first ANN, each testing sample is passed through the first ANN and is assigned a correct-classification probability, using (2). We use a threshold T_1, as a probability defining similarity of classified samples to training data. The samples that possess a probability above this rather strict measure are considered almost equivalent to samples belonging to the training dataset. These are then used to train the neural network more efficiently. Afterwards, through a detailed grid search for all datasets in questions, we defined the threshold to be set at:

$$T_1 = 1 - \left(N_{classes} / \left(k * ScalingFuntcion\right)\right) \quad (3)$$

where $N_{classes}$ is the number of classes in the classification task and kE[50,120] is a value determined by experimentation. The scaling function introduced derived from examination of various datasets of different size and shapes. The function is calculated as follows:

$$ScalingFunction = 1 + \left(\left(N_{samples} - 200\right) / 4000\right) \quad (4)$$

Where $N_{samples}$ is the number of samples in the Dataset. This threshold value is adaptive to each individual classification problem and has shown to work well in our experiments.

B. Testing Phase

The training set is augmented with the potentially correctly classified samples from the testing dataset and is ready to train the second ANN, which is randomly initialized. The second ANN is trained for a smaller number of epochs, compared to the first ANN, since the training dataset is augmented in this case.

The samples that score a correct probability of over the upper limit, are used as a way to skew the training of the second neural network slightly. As the second neural network only activates when the test data are known, but certainly still unlabeled, we are allowed to use the information as training data, using the predicted labels of the first neural network. That way, only when the complete system is evaluated and the training phase is completed, the ground truth of the test data is presented.

This method of enlarging the training dataset of our system is completely optional, as it produces a small but still noticeable increase in overall accuracy. The majority of the improvement however is from reevaluating the potentially wrong results of the first coarse neural network by the second stage.

Once the second ANN is trained, it is ready to reclassify the testing dataset or rather a subset of it. To reduce the computational cost, the second network is forced to re-classify only those testing samples that have scored a probability below a second threshold T_2 by the first coarse ANN during the previous phase. The second threshold T_2 is set at:

$$T_2 = 1 - q\left(N_{classes}/10 * ScalingFunction\right) \quad (5)$$

where qE [1, 2] is also determined by extensive testing. The remaining samples retain their original classification from the first coarse network. This second threshold is less strict and enables several samples to be reclassified, especially those who are not confidently attributed to any category or class. As a result, this has a positive effect on the proposed system's performance and accuracy.

Additionally, we have decided to focus their experiments on a broad array of different datasets to provide more accurate results. Using exactly the same architecture for various datasets is not proposed as every problem has an appropriate ANN solution. Specifically, it is noted that the tests performed do not use exactly the same architecture per-se, as in each dataset different data cleansing and preparation techniques were used. The functions presented were thoroughly tested to perform optimally for all of the tested scenarios regardless of the small changes to the ANN.

Lastly, we clarify that regarding all the above-mentioned equations, the activation functions used to develop our ANN were the following:

- $f\left(x\right) = \dfrac{1}{1+e^{-x}}$ for softmax activation function (6)
- $f\left(x\right) = \max\left\{0, x\right\} = \begin{cases} 0 & \text{if } x \le 0 \\ x & \text{if } x > 0 \end{cases}$ for ReLU function (7)

IV. Benefits of the Proposed Scheme

The proposed scheme features several benefits. Using the above method, the ANN reduces the chance to fall into local minima, due to the random initialization of the second ANN.

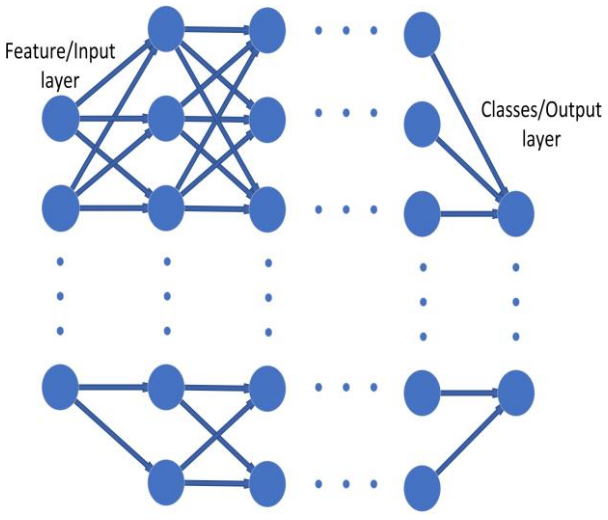

Fig. 2 The architecture of a typical single ANN topology

At the same time, there exists minimal danger of discarding very good classification results, since these samples retain their classification in the second ANN. The second ANN with a random initialization offers a "fresh" look at the incorrectly classified testing data, with an augmented training dataset. Some testing samples that lie below threshold T_1 and might be incorrectly classified are reclassified by the second network, using a less strict threshold T_2.

Equations (3) and (5), regarding thresholds T_1 and T_2 act effectively to encapsulate the notion of larger datasets containing a priori more information. As a result, the possibility of integrating potentially correctly classified samples as a way of enriching the training dataset is increased. Accordingly, the possibility of potential misclassifications is reduced proportionally to the total number of samples contained in the given dataset.

Another advantage of the proposed scheme is that this concept can be used and can improve almost any ANN architecture, including deep architectures, such as convolutional neural networks and recurrent neural networks. The only extra computational cost entails the calculation probability of correct classification for each sample. Another advantage is that this approach can be applied only when needed to increase the classification performance of a given ANN. One can run the simple ANN, get coarse results very quickly, and improve these results only when needed or extra time is available. Probabilistic Cascade ANN can thus improve the performance on any problem while having the flexibility of not being computationally expensive compared to longer and deeper ANNs.

The most important advantage of the proposed twin ANN topology is the fact that similarly performing typical single-ANN as presented in Figure 2 tend to be much more complex, requiring up to 4 times more trainable parameters. In fact, the twin ANN structure is more symbolic than real. That is to say, the first ANN can be discarded after performing the coarse classification. The second ANN will occupy the same memory allocation as the first one. The only extra computational cost are the first classification results and their evaluation as "correct" or "incorrect" classifications. These claims will be verified in the experimental section. Consequently, the proposed ANN structure generally leads to lower training time, smaller models, which can "fit" in smaller GPUs, thus being less computationally expensive.

Furthermore, this combination of two coarse - finer ANN may be regarded either as adding a post-processing step to a coarse ANN. The fine ANN can be considered as a post-processing refinement to a simple-coarse ANN that is used to reclassify data that have been possibly incorrectly classified by the first network. Moreover, it is noted that the ANN proposed can be regarded as an expansion of the typical single ANN used providing similar and in many cases better output results. Finally, the suggested approach may also fall into the category of data distillation and augmentation

approaches [31]. The difference is that here we use the testing dataset to augment the training dataset and thus improve the classification performance of any given ANN architecture.

V. Experimental results

In this section, we benchmark the proposed probabilistic two-stage ANN and compare its performance with a single-stage ANN. The datasets that will be used in our experiments are five popular datasets: the IRIS, MNIST, Wine, Fashion MNIST and Cifar32 datasets. These datasets are available on-line from the UC Irvine Machine Learning Repository [32].We create the training and testing datasets using the complete datasets in the following fashion. We randomly choose 70% of the total samples for training, while the remaining 30% are used as testing data. This procedure was conducted 20 times in random, thus creating 20 different data sets. These 20 different data sets were used by all competing ANN architectures.

In this experiment, we tested the performance of a single ANN against the proposed probabilistic cascade architecture, where the second ANN features the same architecture as the first one. For the Wine and IRIS dataset, we used a Fully-Connected Network (FCN) consisting of 2 hidden layers with 20 neurons each. For the Fashion MNIST and MNIST datasets, we used a deep ANN, consisting of two convolutional layers and a FCN. The first convolutional layer consisted of 32 3 X 3 filters and a max- pooling layer. The second convolutional layer consisted of 64 3 X 3 layers and a max-pooling layer, while the FCN layer consisted of 128 neurons. For the Cifar32 dataset, we used a deep ANN consisting of three convolutional and a fully-connected layer. The first convolutional layer consisted of 32 3 X 3 filters and a max-pooling layer. The second convolutional layer consisted of 64 3 X 3 layers and a max-pooling layer. The third convolutional layer consisted of 128 3 X 3 layers and a max-pooling layer, while the FCN layer consisted of 1024 neurons.

All ANNs used the same training and optimization methods. The activation function for all hidden layer nodes was the ReLU and for the output layers was the softmax function. The network's cost function was categorical cross-entropy [33], which was then optimized via the Adam optimizer [34]. The training of each network was only terminated after a fixed number of epochs, in order to compare their time complexity. At the end of each ANN, an accuracy metric is calculated, based on the percentage of correctly classified samples, to the sum of the test data. The ANNs were implemented in Python using the Keras library.

Firstly, we conducted some experiments to identify optimal values for the parameters K and q in thresholds T_1 and T_2. For this purpose, we used the Wine dataset, but the conclusions hold for the other datasets as well. We set K=20 and we estimated the average accuracy for 20 versions of the Wine dataset for various values of q. Some indicative results are shown in Table 1, where it is clear that a value of q=1.2 yields the best results. Consequently, we set q=1.2 and we explore the average ac- curacy for various values of K. The indicative results of Table 2 recommend an optimal value of K=20. These values for q and K has shown to perform best at the other two datasets.

The next step was to evaluate the performance gain acquired by the proposed scheme, compared to the original ANN (Single ANN) that was designed to perform classification for each of the three datasets. Table 3 contains the average accuracy for the 20 versions for each of the three datasets. In the case of the IRIS dataset, the proposed approach yields an improvement of 4.66% in classification accuracy. In the case of the MNIST dataset, the proposed approach yields a slight improvement 0.57%. The small improvement is due to the fact that the single ANN for the MNIST dataset already performs very well, thus there is very little room for improvement. In the case of the Wine dataset, we get an improvement of 7.77%. In the case of the Cifar32 and Fashion MNIST datasets, we get an improvement of 3.82% and 6.08% respectively.

Table 1. Average Classification Accuracy for various values of q and K=100 for the proposed ANN and the Wine Dataset. NA denotes that the network didn't train.

q	1	1.2	1.5	1.7	2
Accuracy	94.4%	98.76%	95.67%	96.29%	NA

Table 2. Average Classification Accuracy for various values of K and q = 1.2 of the proposed ANN for the Wine Dataset.

K	50	70	100	120
Accuracy	96.29%	93.21%	98.76%	95.67%

Table 3. Average Classification Accuracy for 20 random segmentations of the IRIS, MNIST, Wine, Cifar32 and Fashion MNIST Datasets.

Datasets	***IRIS***	***MNIST***	***Wine***	***Cifar32***	***Fashion MNIST***
Single ANN	92.33%	96.34%	90.98%	71.1%	89.46%
Proposed Topology	96.99%	96.91%	98.76%	74.92%	95.54%

The Cifar32 Dataset's complexity almost makes the use of a pre-trained model a necessity for good accuracy, which is out of scope of this research. In total, we get an average classification accuracy improvement of 4.582%. In essence, the proposed approach can boost the performance of ANN architectures and it seems to perform better when the original

ANN leaves enough room for performance improvement.

Another strong point of the proposed method emerges, when we compare the proposed network architecture with an equally performing larger ANN. One could easily register a strong difference in computational complexity between a single ANN and the proposed method for similar performance levels. To verify this, we used the Fashion MNIST dataset and a fully-connected network as a workbench. The proposed algorithm used two networks, featuring two hidden layers with 10 neurons each, both trained for 20 epochs. The proposed network featured the same, if not slightly better, accuracy compared to a two-layer network, with 35 neurons at each layer that was trained for 50 epochs. In summary, training twice 8,070 variables (i.e. 16,140 variables) for 20 epochs with the use of the proposed topology, the performance is similar to training 27,475 variables for 50 epochs, a rather much more computationally demanding task. Similar conclusions can be gathered from the other datasets, such as the Cifar dataset.

On the one hand, it is important to note that the simple ANN topology tries to model the given problem all in one go, no matter how difficult some training examples are to simulate/understand. On the other hand, the suggested topology models the majority of the training examples more coarsely, while the rest more "difficult" examples are modeled in a second neural network. This discrimination leads to overall better modeling of the problem as shown in the results of Table 4.

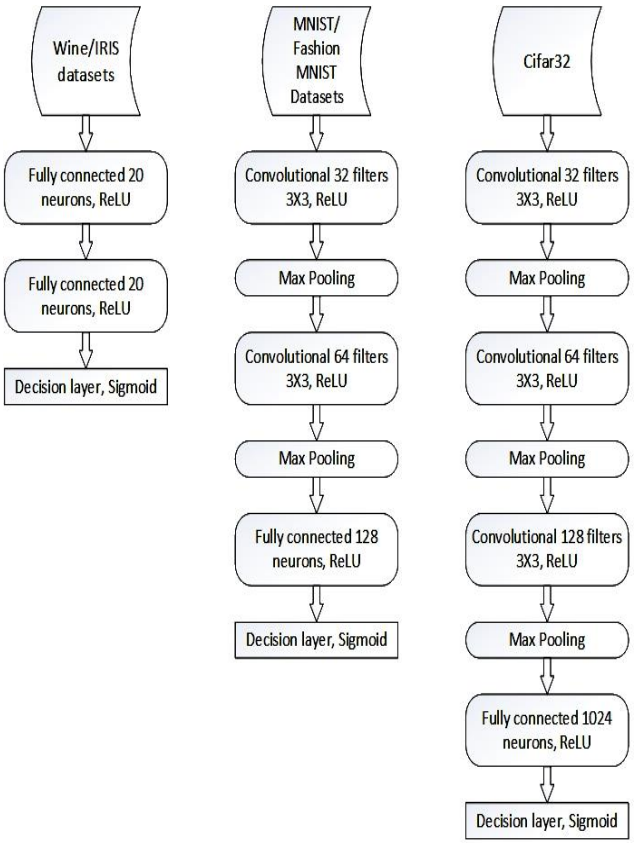

Fig. 3 The topology of the networks used in our test cases

Additionally, it must be underlined that the single ANN topology aims to match the accuracy levels presented in Tables 1, 2, 3 and 4. Lastly, the architectural topologies of our test cases are presented in Figure 3 while the results of our experiments for each dataset are presented in Table 5.

Table 4. Computational Requirements for equal performance to proposed topology results.

Fashion MNIST Dataset	
Single ANN	27,475 variables 50 epochs
Proposed	8,070 x2 variables 20 epochs
Cifar Dataset	
Single ANN	19,016,778 variables 50 epochs
Proposed	628,810 x2 variables 50 epochs

Table 5. Statistical results of the tested topologies.

Dataset	*ANN Comparison*	*Mean Square Error*	*Categorical Cross Entropy*
Cifar	Typical	0,1550	1,3600
	Proposed	0,1280	1,2376
	Improvement	17,4193548 %	9,0000 %
MNIST	Typical	0,0482	0,3160
	Proposed	0,0329	0,1434
	Improvement	31,7427386 %	54,6202532 %
Fashion MNIST	Typical	0,0691	0,4080
	Proposed	0,0547	0,1799
	Improvement	20,8393632 %	55,9068627 %
Wine	Typical	0,0458	0,1640
	Proposed	0,0331	0,1170
	Improvement	27,7292576 %	28,6585366 %
Iris	Typical	0,1470	0,1543
	Proposed	0,1150	0,1299
	Improvement	21,7687075 %	15,8133506 %
Overall Improvement		23 %	25,98 %

VI. Conclusions

In this paper, we propose a strategy to improve the performance of any ANN architecture. The original ANN architecture is used to classify the original testing data. A probabilistic evaluation of the accuracy of each classification is then performed. The most trustworthy classifications of the testing samples are augmenting the training set and are used to train a second version of the original ANN architecture with a random initialization. The second network then reclassifies the poorly classified samples, and the combination of the correctly classified samples from the first neural

network are combined with the newly classified samples from the second network. The result features improved classification accuracy compared to the original trained network as a single unit.

This article shows merit as it proposes a novel ANN architecture approach that aims to combine two smaller classifications ANNs and create on the fly a new ANN that monitors and classifies incorrect results. The suggested approach uses several innovative ideas not particularly studied in the current bibliography like the use of connected back-to-back ANNs combined with other ANNs using cascade training. Furthermore, we argue that the equalization factor in our experiments is accuracy's level as it is a crucial factor to develop a fast but also reliable ANN model.

The proposed strategy has no limitation and thus can be used with any deep or shallow ANN architecture. Finally, it can boost performance without increasing the memory requirements of the network, which is extremely important for modern deep machine learning architectures.

VII. Future Work

The future work of this publication lies in the optimization of the ANN introduced which is a viable, fast, and computationally efficient method to boost classification performance in several neural network architectures. Firstly, a network topology optimization is currently under research, mainly focusing on machine learning as a mean to mine the input data. Secondly, we argue that after several experiments conducted, the aforementioned ANN topology could be expanded to more complex detests. For example, an idea for future research would be to use an ANN of similar size to study the optimization of algorithmic information theory axioms, such as the Solomonoff–Kolmogorov–Chaitin complexity. These types of algorithms simulate the Entropia (i.e. "randomness") of a real-world dataset, i.e., short datasets characterized by random data samples and an unknown amount of noise [35]. We could apply these algorithms as a means to optimize the computational resources of the proposed ANN by examining different datasets. Moreover, with trial and error unsupervised learning, we could further tweak the probability prediction of our ANN to include more possible random states and avoid overfitting the tested dataset.

Thirdly, we could create a new module to be added to MATLAB's Deep Learning toolbox documentation. Specifically, we may write simple instructions on the variables and libraries used in our ANN code, create a new package in MATLAB, and generate a single installation file to be shared in the toolbox. Furthermore, given that MATLAB is not an open-source software, we could use its integration tool to convert our code to a "lower level", that is much closer to the hardware programming language. We propose the use of C programming as it is robust, available for existing processor architecture, and flexible for future developers and researchers, as the default language for UNIX, and UNIX-flavored systems. Lastly, we could focus our efforts on developing an API that can be used by other researchers, who are unfamiliar with complex programming languages. A useful solution would be to create a library for Python, R, or Java, which are the de-facto programming languages used in machine learning and statistical analysis.

Acknowledgment

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research.

References

[1] B. S. Aloysius N., Geetha M., "A review on deep convolutional neural networks", International Conference on Communication and Signal Processing, 2017, pp. 588-592, doi:10.1109/ICCSP.2017.8286426.

[2] O. I. Abiodun, A. Jantan, A. E. Omolara, K.V. Dada, A. M. Umar, O.U. Linus, et al., "Comprehensive Review of Artificial Neural Network Applications to Pattern Recognition", IEEE Access, vol. 7, 2019, pp. 158820-158846, doi:10.1109/ACCESS.2019.2945545.

[3] S. Theodoridis, K. Koutroumbas, "Pattern Recognition", Academic Press, 2019, doi:10.1016/B978-1-59749-272-0.X0001-2.

[4] R. O. Duda, P. E. Hart, D.G. Stork, "Pattern classification", John Wiley and Sons, 2012, isbn:978-0-471-05669-0.

[5] T. Nagpal, Y.S. Brar, "Artificial neural network approaches for fault classification: comparison and performance", Springer Neural Computing and Applications, vol. 25, 2014, pp. 1863–1870, doi:0.1007/s00521-014-1677-y.

[6] S. Theodoridis, "Machine learning: a Bayesian and optimization perspective", Academic press, 2015, isbn:978-0-12-801522-3.

[7] Z. Qin, D. Kim, T.Gedeon, "Rethinking Softmax with Cross-Entropy: Neural Network Classifier as Mutual Information Estimator", arXiv, 2020, preprint: arXiv:1911.10688.

[8] T. Pang, K. Xu, Y. Dong, C. Du, N. Chen, J. Zhu, "Rethinking Softmax Cross-Entropy Loss for Adversarial Robustness", arXiv, 2020, preprint: arXiv:1905.10626.

[9] Y. Zeinali, B. Story, "Competitive probabilistic neural network", Ios Press Integrated Computer Aided Engineering, vol. 24, 2017, pp. 105-118, doi:10.3233/ICA-170540.

[10] C. Wu, H. Jiang, P. Wang, "Education quality detection method based on the probabilistic neural network algorithm" Diagnostyka, vol.21, 2020, pp. 79-86, doi:10.29354/diag/127194.

[11] N. Feng, S. Xu, Y. Liang, K. Liu, "A Probabilistic Process Neural Network and Its Application in ECG

Classification", IEEE Access, vol. 7, 2019, pp. 50431-50439, doi:10.1109/ACCESS.2019.2910880.
[12] C. Yang, J. Yang, Y. Liu, X. Geng, "Cancer Risk Analysis Based on Improved Probabilistic Neural Network", Frontiers in Computational Neuroscience, vol.14, 2020, doi:10.3389/fncom.2020.00058.
[13] A.V. Savchenko, "Probabilistic neural network with complex exponential activation functions in image recognition", IEEE Transactions on Neural Networks and Learning Systems, vol.31, num.2, 2019, pp. 651-60, doi:10.1109/TNNLS.2019.2908973.
[14] M. Xiang, J. Yu, Z. Yang, Y. Yang, H. Yu, H. He, "Probabilistic power flow with topology changes based on deep neural network", International Journal of Electrical Power and Energy Systems, vol.117, 2020, pp. 105650, doi:10.1016/j.ijepes.2019.105650.
[15] N. Aljeri, A. Boukerche, "A Probabilistic Neural Network-Based Road Side Unit Prediction Scheme for Autonomous Driving", IEEE International Conference on Communications, 2019, pp. 1-6, doi:10.1109/ICC.2019.8761749.
[16] Y. Akhmetov, A.P. James, "Probabilistic neural network with memristive crossbar circuits", IEEE International Symposium on Circuits and Systems, 2019, pp. 1-5, doi:10.1109/ISCAS.2019.8702153.
[17] F. Min, J. Xue, F. Ma, "Probabilistic Neural Network Motor Bearing Fault Diagnosis Based on Improved Feature Extraction", IOP Journal of Physics, vol. 1684, num. 1, 2020, pp. 012158, doi:10.1088/1742-6596/1684/1/012158.
[18] B.H. Lohithashva, Manjunath Aradhya V.N., Basavaraju H.T., Harish B.S., "Unusual Crowd Event Detection: An Approach Using Probabilistic Neural Network", Springer Information Systems Design and Intelligent Applications: Advances in Intelligent Systems and Computing", vol.862, 2020, pp. 012158, doi:10.1007/978-981-13-3329-3_50.
[19] A.K. Sahoo, C. Pradhan, H. Das, "Performance evaluation of different machine learning methods and deep-learning based convolutional neural network for health decision making", Springer Nature Inspired Computing for Data Science, vol.871, 2020, pp. 201-212, doi:10.1007/978-3-030-33820-6_8.
[20] M. Sayed-Mouchaweh, E. Lughofer, "Learning in non-stationary environments: methods and applications", Springer Science and Business Media, 2012, doi:10.1007/978-1-4419-8020-5.
[21] M. Mohammadi, A. Al-Fuqaha, S. Sorour, M. Guizani, "Deep learning for IoT big data and streaming analytics: A survey", IEEE Communications Surveys and Tutorials, vol.20, num.4, 2018, pp. 2923-2960, doi:10.1109/COMST.2018.2844341.
[22] Y. Cui, S. Ahmad, J. Hawkins, "Continuous online sequence learning with an unsupervised neural network model", Neural computation, vol.28, num.11, 2016, pp. 2474-2504, doi:10.1162/NECO_a_00893.
[23] L.Rabiner, B. Juang, "An introduction to hidden Markov models", IEEE ASSP Magazine, vol.3, num.1, 1986 pp. 4-16, doi:10.1109/MASSP.1986.1165342.
[24] A. Oussidi, A. Elhassouny, "Deep generative models: Survey", IEEE International Conference on Intelligent Systems and Computer Vision, 2018, pp. 1-8, doi:10.1109/ISACV.2018.8354080.
[25] G.P. Zhang, "Time series forecasting using a hybrid ARIMA and neural network model", Neurocomputing, vol.50, 2003 pp. 159-175, doi:10.1016/S0925-2312(01)00702-0.
[26] A. Olawoyin, Y. Chen, "Predicting the future with artificial neural network", Elsevier Procedia Computer Science, vol.140, 2018, doi: 10.1016/j.procs.2018.10.300.
[27] F.P. Casale, J. Gordon, N. Fusi, "Probabilistic neural architecture search", arXiv, 2020, preprint arXiv:1902.05116.
[28] N.S. Sohoni, C.R. Aberger, M. Leszczynski, J. Zhang, C. Ré, "Low-memory neural network training: A technical report", 2019, preprint arXiv:1904.10631.
[29] C. M. Bishop, "Neural Networks for Pattern Recognition", Oxford University Press, 1995, isbn:978-0-19-853864-6.
[30] MATLAB, "Deep Learning Toolbox Documentation" (2020), Available: https://mathworks.com/help/deeplearning/.
[31] F. Chollet et al. (2015), "Python Keras Api", Available: https://keras.io.
[32] D. Dua, C. Graff, "UCI Machine Learning Repository", Irvine, CA: University of California, School of Information and Computer Science, 2017, Available:https://archive.ics.uci.edu/ml.
[33] S. Zhang, L. Yao, A. Sun, Y. Tay, "Deep Learning Based Recommender System: A Survey and New Perspectives", ACM Computer Surveys, vol.52, no.1, 2019, doi:10.1145/3285029.
[34] A. Shrestha, A. Mahmood, "Review of Deep Learning Algorithms and Architectures", IEEE Access, vol.7, pp. 53040-53065, 2019, doi:10.1109/ACCESS.2019.
[35] G. Ruffini, "Models, Networks and Algorithmic Complexity", Starlab technical note, TN00339, 2016, pp. 12-15, doi:10.13140/RG.2.2.19510.50249.