International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys.13, No.1 (May-23)

http://dx.doi.org/10.12785/ijcds/130194

Improving Performance of Deep Learning Models Using Input Propagation

Mahalingam P. R.[1] **and Dheeba J.**[1]

[1]*School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, India*

Received 31 Mar. 2022, Revised 2 May. 2023, Accepted 14 May. 2023, Published 30 May. 2023

Abstract: Artificial Neural Networks (ANNs) act by fitting decision boundaries between different classes of data, thereby performing classification. They approximate the decision hyperplane into a weighted sum of inputs, thereby making the decision simpler. This paper explores the working of ANNs in a simple scenario where the decision boundary is quadratic, finds that weighted sum won't be enough to perform non-linear classifications since decision boundaries remain linear in nature. To overcome the limitations of the existing model, an Input-Propagated Artificial Neural Network (IP-ANN) is proposed, which learns nonlinear functions directly by incrementally applying input values on the network directly. The performance of the proposed model is compared with the regular ANNs, and experimental analysis shows that the regular ANN has obtained an accuracy of 72.36%, while IP-ANN achieved an accuracy of 98.5%.

Keywords: Machine learning, Artificial Neural Networks, Deep Learning, Polynomial Neural Networks, Input propagation.

1. INTRODUCTION

Artificial Neural Networks (ANNs) has been a steady choice for researchers around the world for applications using machine learning. ANNs has performed well in cloud based classification[1] , clinical predictions[2] , and disease predictions[3], [4]. One of the attractive features of ANNs is the relatively small prediction time (even though training can be slow, testing at real time is fast), which enables them to be integrated into a variety of software and hardware applications. Modelled after biological neural networks, an ANN contains a series of "artificial neurons" which are arranged into layers, with each layer fully collected to the nearby layer like a bipartite graph. The connections are appropriately weighted, and each neuron performs some kind of aggregation and normalization to give the output. When a network is trained, it adjusts the weights between layers so that the output error is minimized as much as possible.

ANNs are lucrative because of the huge variety of mathematical functions they can learn from the input data. But conventional ANNs suffer from some limitations of linearity, which can severely hamper their utility in certain scenarios. In this paper, a new model is proposed that can rectify the problem of ANNs where non-linear decision boundaries are not fit properly. A special dataset is synthesized for this purpose.

Ample work has been found relating to performance improvement of ANNs in nonlinear datasets. Most of them have concentrated on the mathematical model, which has given rise to polynomial neural networks. Some other methods have been found to use nature-inspired algorithms to modify the training regime. In this paper, the approach is focused on modifying the architecture of regular ANN, and incorporating an additional mathematical step into the neuron.

The main objective of the present work is to create a model that can overcome the limitations presented by deep networks constructed out of conventional artificial neurons. To this end, our contributions in IP-ANN are as follows.

- *Creation of a new model* - After understanding the pitfalls of the existing deep network architecture, a new neural model is proposed that performs an additional forward propagation of input to all layers in the network, enabling it to learn higher order polynomial functions with minimal rework.
- *Analysis on hypothetical dataset* - Once the model is explored, a hypothetical dataset is introduced, which requires a quadratic decision boundary for proper classification. The same is used for training a regular deep network and an IP-ANN, and it is observed that IP-ANN performs much better.
- *Specific use cases* - Once the working is verified with the hypothetical dataset, the same is studied by

E-mail address: prmahalingam@gmail.com, dheeba.j@vit.ac.in

giving specific use cases as input. The domain under consideration is healthcare data, with focus placed on cardiovascular diseases.

- *Performance* - The performance of IP-ANN is further analyzed with a wider range of benchmarked datasets, and it is observed that it is consistently performing at par, or better than, deep networks. The caveat of this model is also analysed, which is the increase training time with number of features.

- *Recommendations* - The study ends with a set of recommendations on when the model can be used, and how further studies may be conducted to improve the training speed of the model.

The rest of the paper is organized as follows. Section 2 gives a background study of the concept. Section 3 deals with an exploration of how conventional ANNs model nonlinear functions using approximation, and Section 4 introduces a novel concept called "Input-propagated ANNs" (IP-ANNs) which is aimed at enhancing the performance of conventional ANNs by taking the assistance of inputs at each layer. Section 5 compares the performance of IP-ANN against conventional ANNs in a variety of situations, and Section 6 discusses a specific use case in the form of medical data. Closing remarks and future scope is discussed in Section 7.

2. BACKGROUND

Artificial Neural Networks are good at approximating nonlinear functions[5] when large datasets are available. This is because decision boundaries are learnt and evolved, and unless there is enough data it may not be optimal. The use of large datasets are necessary because ANNs work with largely linear combinations, followed by nonlinear transfer functions (like sigmoid). A number of attempts have been made in improving the performance of ANNs in the case of nonlinear decision boundaries. This is since majority of real world applications cannot be modelled as linear functions. Functions like sigmoid, tanh, ReLU, etc. incorporate a good level of nonlinearity by making the output a function of the input. But that may not be enough if the boundary is extremely nonlinear.

Wu et al[6] proposed an ANN weight adjustment algorithm based on Beetle Antennae Search. They work on the hidden-output matrix and adjust them in such a way that there is rapid convergence towards the global optimum. It used a state space search with step size based on antenna length of the beetle. This algorithm was observed to give an acceptable level of accuracy without overfitting. Meng et al[7] proposed a self-adaptive RBF classifier based on fuzzy c-means and quantum-inspired PSO for transformer fault analysis. It extends the regular ANN to accommodate a set of neurons in the hidden layer to exclusively transfer the input data to the output layer. Hence linearity is ensured by the input, while RBF adds an element of nonlinearity.

Thiria et al[8] discussed about a method where nonlinear transfer functions are learnt using ANNs after appropriate flattening of multidimensional input at the first hidden layer. This was demonstrated in the case of transfer functions related to scatterometers. Hoekstra and Duin[9] mapped the shape of a classification function to the generalization capability of the corresponding classifier. The paper introduced a nonlinearity measure which is then used to perform interpolation and scaling of the dataset. It can be thought of as embedding the nonlinearity within the output classes themselves to avoid locating the same during training. Huang and Ma[10] gave a comprehensive mathematical analysis of neural networks, and it shows how any why ANN propagations are based on matrix operations. Ali et al[11] proposed an implementation that takes the assistance of ontology for prediction of heart diseases. Makantasis et al[12] also explore the matrix algebra implementation of neural networks and weights, with focus on nonlinearity using activation functions. Martin et al[13] discussed about nonlinear function approximation using kernel functions.

There has been a good amount of work related to polynomial neural networks. Ivakhenko[14] did a detailed study on complex decision surfaces and identified that they can be approximated by a set of polynomials. The work highlighted the requirement of multilayer networks to implement approximations of polynomial decision functions. It works by taking nonlinear functions and working pairwise when it comes to inputs at each layer, thereby generating approximations from partial descriptions. The coefficients are identified by solving smaller equations with only those pair of unknowns, and minimizing mean squared error of the system. It eventually generates a set of optimization equations, which can approximate the nonlinear system to a good extent. Following up on this concept, Oh et al[15], [16], [17], [18], [19], [20] and Maric[21] have built a number of models that use Group Method of Data Handling (GMDH) to generate approximations. They have proposed a model called the Polynomial Neural Network (PNN) which entails the process followed in GMDH and generates partial descriptions, but gives the additional flexibility of allowing linear, quadratic, or cubic transformations at each level of the partial description. Hence in a layer the data may get raised to an appropriate power, giving nonlinearity. They later expand the concept by allowing Genetic Algorithms to decide the polynomial power at a node, and add fuzziness to the model. They also perform yet another expansion where wavelets are used for faster convergence. They also propose models where PNN generates virtual inputs, that are later processed by generating multiple local models, and using Fuzzy C-means clustering for final output generation.

Some other applications proposed for PNN are as follows. Ghazali et al[22] used a modified PNN, which has an additional internal layer to find the product of sum terms. This enables multiple linear terms to be combined to have higher order polynomials. Al-Tahrawi and Al-Khatib[23] noted that PNNs are able to preserve context because of

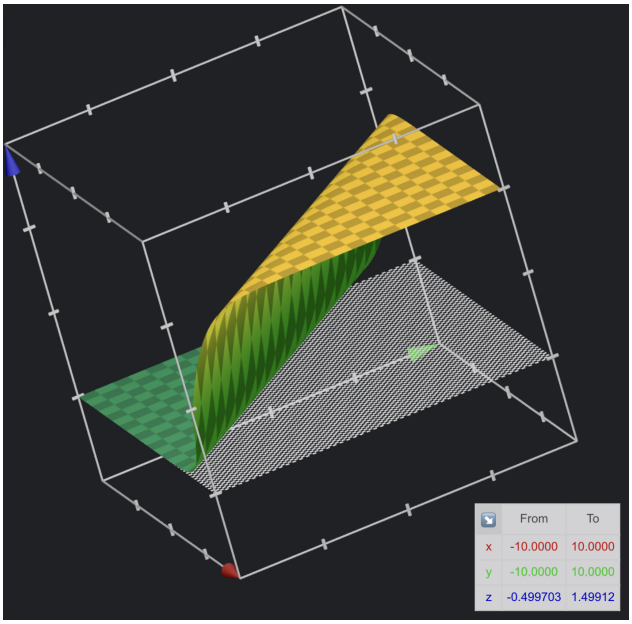

Figure 1. Plot of sigmoid output for a linear input expression.

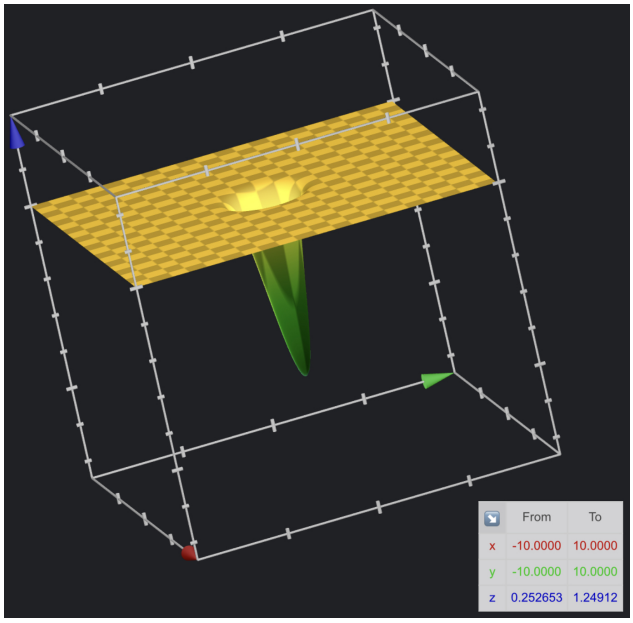

Figure 2. Plot of sigmoid output for a quadratic input expression.

partial descriptions, and used them to classify Arabic text. Kileel et al[24] explored PNNs in the perspective of deep polynomial neural networks by using polynomial activation functions. A different treatment is given by Piazza et al[25]. They propose a regular neural network, but have a complex activation function which considers a number of possible activations internally, and gives an additional coefficient vector to add all activations together to generate the output. Hence a number of possibilities are taken simultaneously, and by the time training is complete, there will be a proper weight to each level of nonlinearity as propagation happens through the layers.

All the above neural models result in generating a linear combination of inputs that can be sent to a sigmoid or tanh activation function for producing the output. If the expression

$$sig(x + y)$$

is plotted, the graph looks like the one given in Figure 1. While the plot looks non-linear, the predictive power is demarcated by a line that forms the surface of the sigmoid function. The performance greatly changes if the input expression for sigmoid changes from linear to quadratic. The plot of

$$sig(x^2 + y^2)$$

looks like the surface in Figure 2. The classification surface resembles a hat, which is a nonlinear surface in itself. This difference is to be leveraged in order to improve deep networks to IP-ANN.

3. ARTIFICIAL NEURAL NETWORKS AND LINEARITY

Conventional ANNs[26] do a good job in learning fairly complex patterns, but the behavior is quite linear in nature. Appropriate activation functions are required to expand the decision boundary beyond that. The next section deals with evaluating the performance of a standard ANN using a synthetic dataset.

A. *Dataset synthesis and preprocessing*

Consider a dataset that contains two attributes ‘x’ and ‘y’, and a function ‘f’ that simulates a circle which is centered at (50,50), and has a radius of 30. Any point that is inside the circle is marked as f=1, while those outside are marked as 0. The dataset is built using points from (0,0) to (100,100), which makes a total of 10201 points. There will be 7380 points marked as 0 (since outside the circle), and 2821 points marked as 1 (on the circle, or inside it). The curve given by Eqn. 1 will act as the separator line. A classifier should be able to learn this curve and perform classification properly.

$$(x - 50)^2 + (y - 50)^2 = 30^2 \quad (1)$$

The input is compressed into the range [0,1] by dividing all numbers of X and Y by 100. This revises Eqn. 1 as follows.

$$100(x^2 + y^2 - x - y + 0.5) = 9 \quad (2)$$

Eqn. 2 can be visualized as in Figure 3.

B. *Standard ANN architecture*

A regular ANN works with a set of artificial neurons, which work with a set of “weights” and “biases” to generate a biased weighted sum of inputs. The basic architecture

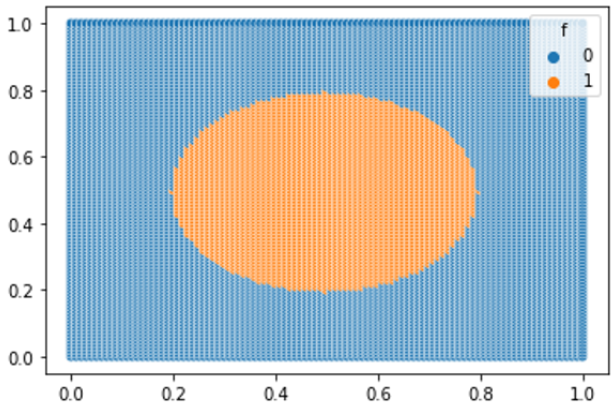

Figure 3. Visualization of function in Eqn 2 generated using Python.

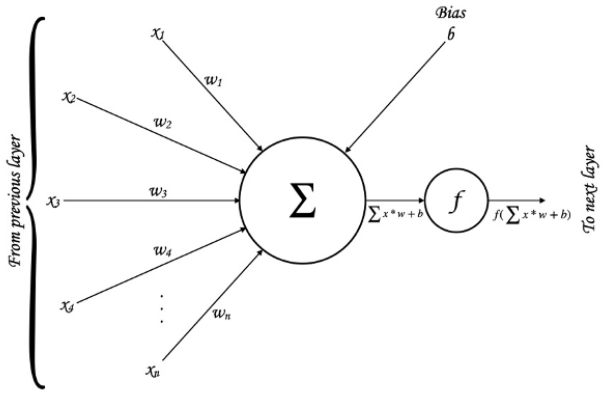

Figure 4. Architecture of an artificial neuron.

followed from an implementation perspective is given in Figure 4, taking the number of inputs as 'n' for the neuron.

From Figure 2, the equation for a neuron can be written as:

$$y = f(\sum(x * w) + b) \tag{3}$$

where x is the input vector, w is the weight vector, b is the bias vector, f is the activation function, and y is the output. The operation * denotes pairwise multiplication.

The main aspect to be considered is that the equation to be learned is a proper polynomial. An ANN is set up with the following configuration:

- Input layer : 2 neurons (for x and y)
- Hidden layer 1 : 2 neurons (first level preprocessing for x and y) with ReLU activation
- Hidden layer : 4 neurons (to approximate a bounding box) with ReLU activation

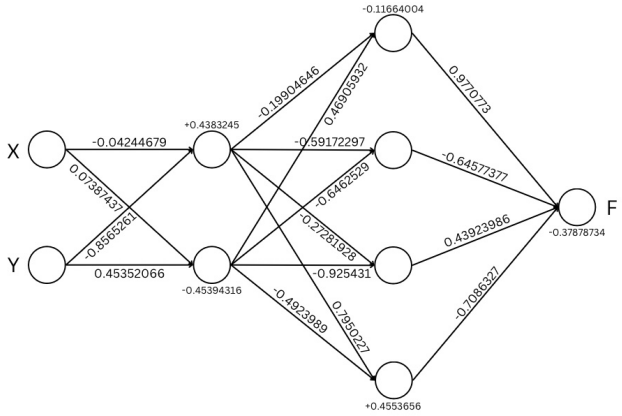

Figure 5. Standard ANN designed for the given dataset.

- Output layer : 1 neuron (binary classification) with Sigmoid activation

After training for 500 epochs on the dataset with a stratified train-test split of 75-25, the network took values as in Figure 5.

On deriving the values, it can be seen that they can be stretched into a long polynomial, which represents the function F. The derivation is done using node-level outputs, assuming that Nij represents node i at layer j. Layers are numbered from 0 so that Input layer is marked as 0. Similarly, nodes are marked from 0.

Layer 0

```
N00 = x
N10 = y
```

Layer 1

```
N01 = -0.04244679x - 0.8565261y + 0.4383245
N11 = 0.07387437x + 0.45352066y - 0.45394316
```

Layer 2

```
N02 = 0.04310034503x + 0.38321658048y -
    0.41681324999
N12 = -0.0226247852x + 0.21373712604y +
    0.03399540862
N22 = -0.0567853294x - 0.186025244y +
    0.3005096980056
N32 = -0.0701218201x - 0.904270767y +
    1.02736464011267
```

Layer 3

```
N03 = sig(0.0814710961556924x + 0.79549252511807y -
     1.40402778921863)
```

The input to sigmoid remains a linear function of X and Y, and the sole nonlinearity in the model is introduced by

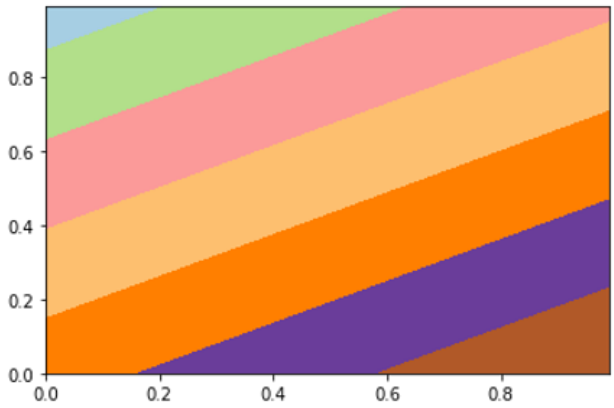

Figure 6. Raw decision boundaries for Standard ANN.

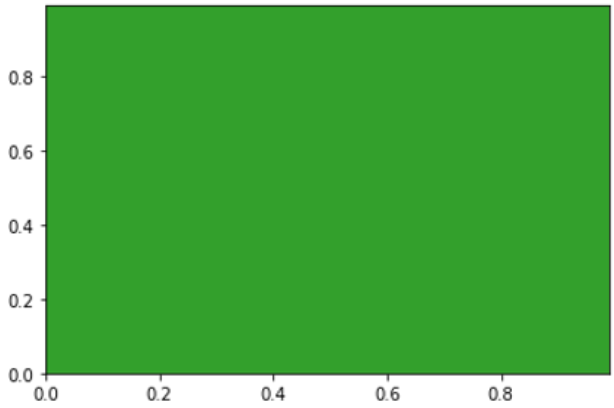

Figure 7. Binarized decision boundaries for Standard ANN.

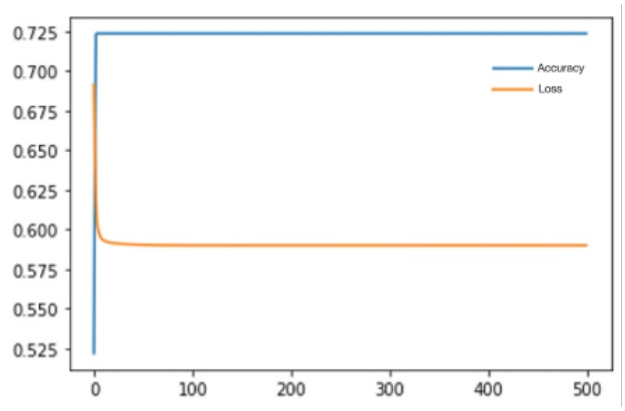

Figure 8. Training accuracy and loss for standard ANN.

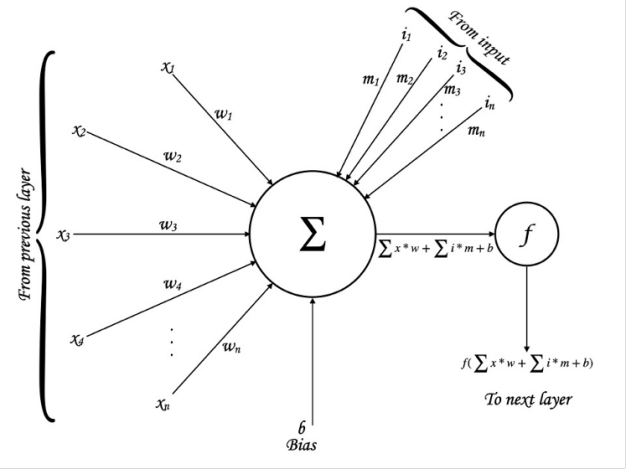

Figure 9. Neuron in IP-ANN.

the final activation function. Here, effect of ReLU has been ignored (since the value is rarely going to be negative in the given example). The function's decision boundary doesn't resemble the function given in Eqn. 1. Figure 6 shows the decision boundary for the system in raw sigmoid output, and Figure 7 shows the same boundary after binarization with values greater than 0.5 denoting 1, and 0 otherwise.

The plot in Figure 6 shows a number of bars, which indicate a strong linear relation expressed in terms of inputs. After binarization, there is only one color in the contour plot, which means that the entire model got overfit to a single output. The accuracy and training loss have been plotted in Figure 8.

The training accuracy nearly levelled off at 72%, and the testing accuracy was found to be 72.36%. Even though 72% seems to be a good number, the dataset is highly imbalanced, and out of 10201 entries, 72.34% are represented by the value 0. Hence it simply denotes an overfit to class 0, which started after the first few epochs itself.

4. INPUT PROPAGATED NETWORKS

The basic problem with the regular ANN architecture is evident from the previous section. There is no proper method by which nonlinearity can be introduced into the system. Considering suggestions from refs. [14] and [25], a new artificial neuron is proposed, which enables inputs to be propagated along the network to all layers. The neuron will indirectly generate a partial description at each layer, by iteratively incorporating inputs to the decision boundary. The inputs will play a part in the output generated by each neuron, and consequently cause nonlinear functions to be learnt by the network. The ANN which uses the new neuron is being named IP-ANN (Input-Propagated Artificial Neural Network).

A. Architecture of the IP-ANN neuron

The structure of a neuron in IP-ANN is given in Figure 9. To propagate the input along the network, the entire input is transferred to each node, along another set of coefficients, called "multipliers". The network will also be designed such that all layers will contain exactly N nodes (where N is the input dimension), and only the output layer is allowed to

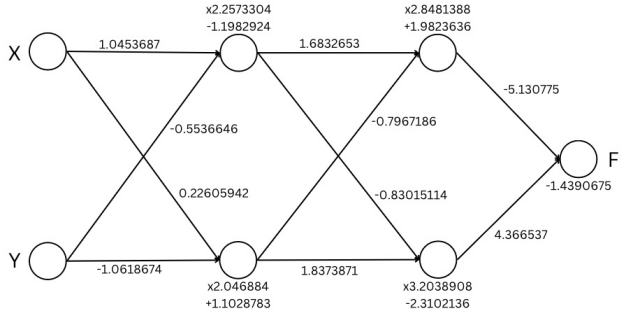

Figure 10. IP-ANN for the dataset in Section 3.

have a different specification, based on the type of output expected (one node + sigmoid, if it is binary, k nodes + softmax if multiclass). The multipliers are tuned in such a way that only mj is non-zero for the jth node in a layer. Others will be set to 0. Hence it becomes just like the bias vector, with exactly N multipliers playing a part in each layer. The output of each neuron now changes as in Equation 4.

$$y = f(\sum(x * w) + \sum(i * m) + b) \tag{4}$$

Here, there is an additional term consisting of the input vector i, and multiplier vector m. Since the multiplier vector is non-zero for exactly one value only, the sum will be simply ij*mj for the jth node in the layer. This helps incorporate the effect of input X in all layers, progressively raising them to higher powers. The network is now able to learn the polynomial terms by itself. The partial description is no longer in terms of the input variables, but in terms of order of the polynomial.

B. Testing the IP-ANN

An IP-ANN model to train on the dataset synthesized in Section 3 requires two nodes in all hidden layers to accommodate x and y. The output layer needs to have one node with sigmoid activation to accommodate binary classification. In the first hidden layer, there will be 4 weights connecting to the layer, just like in normal ANNs. Similarly, the bias vector contains two values, one per node. When it comes to the multiplier, only m1 of node 1 is non-zero, while only m2 of node 2 is non-zero. Hence it effectively reduces to just 2 multiplier values, with node 1 affected by X, and node 2 affected by Y. The architecture is given in Figure 10. Multipliers will be trained in the same way as done for weights and biases.

Proceeding as with conventional ANN, the following values are obtained.

Layer 0

```
N00 = x
N10 = y
```

Layer 1

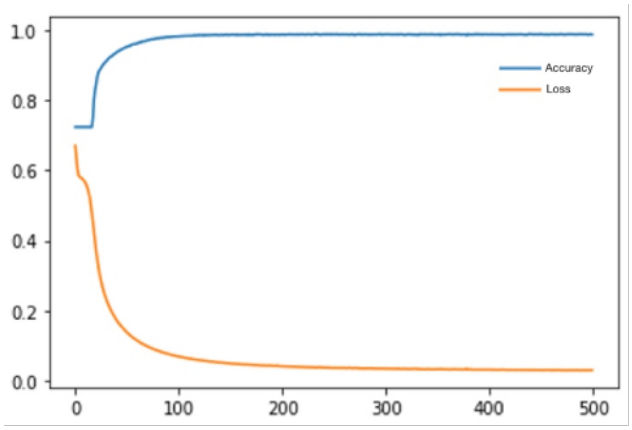

Figure 11. Training accuracy and loss for IP-ANN on the dataset given in Section 3.

```
N01 =  2.359742545718 48$x^2
       - 1.249803932983 84$xy
       - 1.1982924
N11 =  0.462717409847 28$xy
       -2.173519391181 6$y^2
       + 1.1028783
```

Layer 2

```
N02 =  11.313014499$x^3
       - 7.0417587594$x^2y
       + 4.932074471 29$xy^2
       - 8.247434334 89$x
       + 1.9823636
N12 =  -6.276239341 46$x^2y
       + 6.048039640 42$xy^2
       - 12.795047056 3$y^3
       + 9.679524922 24$y
       - 2.3102136
```

Layer 3

```
N03 =  sig(-58.044531966 1$x^3
           -55.870046388 1$y^3
           +8.724248493 5$x^2y
           +1.103624472$xy^2
           +42.315729899 6$x
           +42.266003715 4$y
           -21.697762262 1)
```

Contrary to the regular ANN, this expression is nonlinear in nature, and it is given as the input to sigmoid. This can generate a better result, and the accuracy-loss plot is shown in Figure 11.

The accuracy is found to reach values close to 100%. Model inspection shows an accuracy of 98.6% on training, and 98.5% on validation. So the model is able to give a good classification, as well as avoid overfitting to a single class. The decision boundaries of the classifier are plotted in Figure 12 and Figure 13.

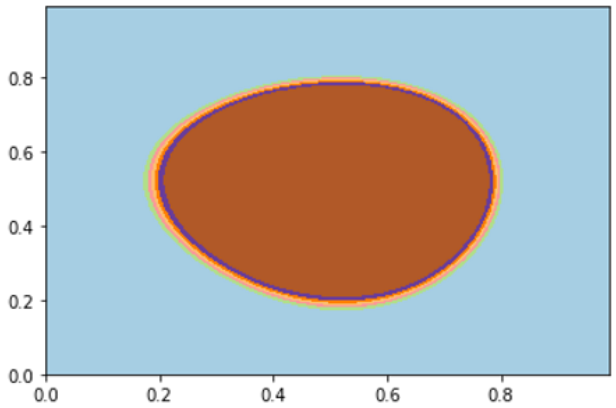

Figure 12. Raw decision boundaries for IP-ANN.

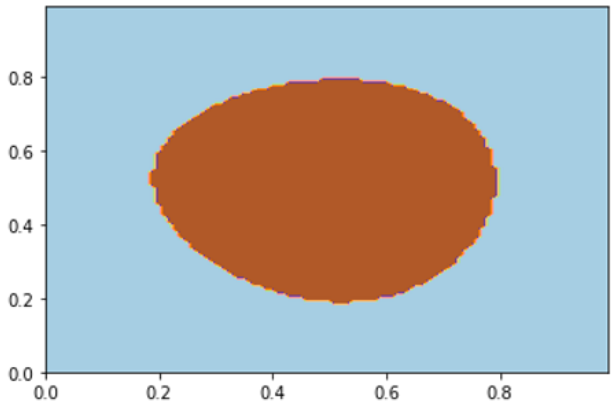

Figure 13. Binarized decision boundaries for IP-ANN.

As observed in the regular ANN, there are a number of bands, each showing a small range of values. But the noticeable difference is that all of them are centred around the original region of interest, and are highly compressed in nature. So the classifier is able to give a good performance. The binarized output from sigmoid makes the decision boundary even more precise.

The decision boundary closely follows the original one, and except for a small region amounting to nearly 2%, classified the inputs correctly. Figure 14 shows the misclassification region, which is a plot of the model's error. The misclassification evidently occurred at the boundary, and other regions are quite accurate.

5. PERFORMANCE COMPARISON

The IP-ANN model was tested for stability and performance using a variety of datasets. The datasets covered the case of binary classifiers as well as multiclass classifiers. No activation function was used in any of the hidden layers, and the output layer contained a sigmoid activation function for binary classification problems, and softmax for multiclass classification problems. The performance was measured against a standard ANN with the same number of layers.

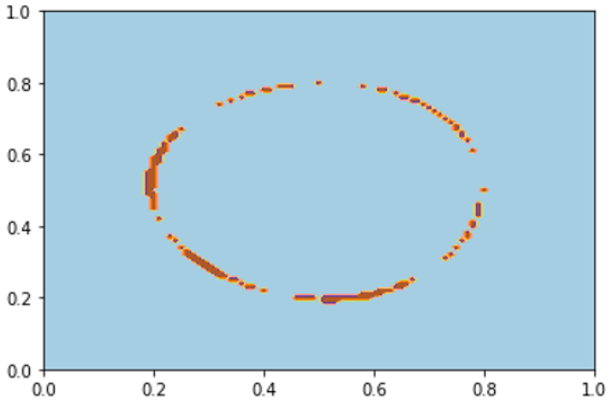

Figure 14. Error region for IP-ANN on data given in Section 3.

In the standard ANN, ReLU was used in all hidden layers as activation function, and output layer was same as that in the IP-ANN model. The models were compared for four datasets. The architecture is described in Table 1.

Performance was measured with 10-fold cross validation. The results are summarized in Tables 2 to 5.

IP-ANN has performed at par, or better than standard ANN in all cases. The difference is evident in binary datasets, while it is difficult to find in MNIST (considering the dataset has 784 training attributes after flattening, both models could have reached the upper limit of accuracy for that dataset, without overfitting).

One aspect that should be kept in mind is that IP-ANN needs the same number of nodes in all layers, and the number is equal to the number of input features. Hence the training will be slower than normal for datasets having a large number of features, and large batch size. The training time per epoch is listed in Table 6.

For heart disease datasets which contain less than 20 features (no feature selection was done prior to training), the training time is nearly the same. The additional effort to modify the multipliers had negligible effect. For MNIST and Fashion MNIST, the training time is more than 4 times that of a standard ANN, since the hidden layer contains 784 nodes in IP-ANN whereas it is only 128 for standard ANN. The "tapering" effect cannot be incorporated into IP-ANN in the current design.

6. EXAMPLE APPLICATION ON MEDICAL DATA

Neural networks has been used in a number of domains like structural analysis[29], [30], mechanical predictions, healthcare[31], etc. because of its versatility. In order to further analyse the impact of input propagation, a comparison is being made between input-propagated networks and conventional deep networks with respect to three heart disease datasets:

1) Heart disease clinical dataset[32]

TABLE I. Testing configuration on different datasets.

Dataset	Description	Standard ANN	IP-ANN
Cleveland Heart Disease dataset[27]	14 attributes, binary classification	2 hidden layers with configuration (13, 13). ReLU used in hidden layers, sigmoid at output. 100 epochs for training.	2 hidden layers with configuration (13, 13). No activation used in hidden layers, sigmoid at output. 100 epochs for training.
Heart failure clinical data [28]	14 attributes, binary classification	2 hidden layers with configuration (13, 13). ReLU used in hidden layers, sigmoid at output. 100 epochs for training.	2 hidden layers with configuration (13, 13). No activation used in hidden layers, sigmoid at output. 100 epochs for training.
Fashion MNIST	28x28 matrix of images, flattened to 784 attributes. 10 output classes	1 hidden layer with 128 nodes, ReLU activation at hidden layer, softmax at output. 5 epochs for training.	1 hidden layer with 784 nodes, no activation at hidden layer, softmax at output. 5 epochs for training.
MNIST	28x28 matrix of images, flattened to 784 attributes. 10 output classes	1 hidden layer with 128 nodes, ReLU activation at hidden layer, softmax at output. 5 epochs for training.	1 hidden layer with 784 nodes, no activation at hidden layer, softmax at output. 5 epochs for training.

TABLE II. Training accuracy

Dataset	Standard ANN	IP-ANN
Cleveland heart dataset	88.19%	95.78%
Heart failure clinical data	86.36%	93.39%
Fashion MNIST	89.07%	89.86%
MNIST	98.67%	99.10%

TABLE III. Training loss

Dataset	Standard ANN	IP-ANN
Cleveland heart dataset	0.283	0.134
Heart failure clinical data	0.308	0.169
Fashion MNIST	0.299	0.273
MNIST	0.044	0.034

TABLE IV. Testing accuracy

Dataset	Standard ANN	IP-ANN
Cleveland heart dataset	80%	83.33%
Heart failure clinical data	79.31%	86.21%
Fashion MNIST	86.53%	87.11%
MNIST	97.6%	97.6%

TABLE V. Testing loss

Dataset	Standard ANN	IP-ANN
Cleveland heart dataset	0.548	0.498
Heart failure clinical data	0.529	0.329
Fashion MNIST	0.422	0.360
MNIST	0.087	0.080

TABLE VI. Training Time per epoch (Batch size = 128).

Dataset	Standard ANN	IP-ANN
Cleveland heart dataset	1ms	1ms
Heart failure clinical data	1ms	1ms
Fashion MNIST	5s	22s
MNIST	5s	23s

2) Framingham heart disease dataset[33]
3) Cleveland heart disease dataset[34]

All analysis is done considering 7 layers, with all layers containing the same number of neurons, decided by the number of features in the input dataset. Comparison is done based on F-score as well as Matthew correlation coefficient[35], with 10-fold cross validation. The results are given in tables 7 and 8.

The analysis shows that Framingham dataset doesn't seem to be a good choice for classification using neural networks. But in all cases, input propagation was able to give a better performance compared to conventional deep networks.

7. CONCLUSION AND FUTURE SCOPE

A novel model has been introduced that can approximate non-linear functions with good degree of accuracy. While regular ANNs rely on the activation function to generate a complex hyperplane, input propagation enables the same to be done on a much larger scale, with much better accuracy.

In general, the model has performed at par, or better than, existing ANN architectures. With an improvement in accuracy by up to 10%, the model can be used for a number of practical problems. The model has been built to ensure minimal modifications in the current training process. Accordingly, only one set of multipliers have been added. But the effect produced by this minor change far outweighs the additional effort. In future efforts have to be taken to taper the layers towards the output so that training time is improved.

Despite the higher training time as given in Table 2, training is a one-time process, and IP-ANN should perform better compared to a standard ANN architecture. This can be extended in future by giving additional nonlinear functions of the input (log, exponential, etc.) to the neurons. Similarly, the possibility of using GPUs[36] may be explored to improve training time on large datasets. Fused activation functions[37] may also improve the performance, but to a smaller extent, since the presence of activations is less in this model. Other possibilities would be making the model flexible by modifying the input vector given to all layers, which requires further analysis in terms of balancing propagation and manipulation of the input. In practice, the model can be used for mission critical applications, including healthcare. The same has been demonstrated using two heart disease datasets, where the accuracy improved by more than 7%.

8. ACKNOWLEDGMENTS

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

REFERENCES

[1] J. Lee, R. C. Weger, S. K. Sengupta, and R. M. Welch, "A neural network approach to cloud classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 28, no. 5, pp. 846–855, 1990.

[2] W. Penny and D. Frost, "Neural networks in clinical medicine," *Medical Decision Making*, vol. 16, no. 4, pp. 386–398, 1996.

[3] N. Rajesh, T. Maneesha, S. Hafeez, and H. Krishna, "Prediction of heart disease using machine learning algorithms," *International Journal of Engineering and Technology (UAE)*, vol. 7, no. 2.32, pp. 363–366, 2018.

[4] S. Ambekar and R. Phalnikar, "Disease risk prediction by using convolutional neural network," in *2018 Fourth international conference on computing communication control and automation (ICCUBEA)*. IEEE, 2018, pp. 1–5.

[5] L. Hadjiiski, P. Geladi, and P. Hopke, "A comparison of modeling nonlinear systems with artificial neural networks and partial least squares," *Chemometrics and intelligent laboratory systems*, vol. 49, no. 1, pp. 91–103, 1999.

TABLE VII. F-score

Dataset	DNN	IP-ANN
Cleveland heart dataset	0.76	0.81
Heart failure clinical data	0.61	0.67
Framingham dataset	0.13	0.16

TABLE VIII. Mathew Correlation Coefficient

Dataset	DNN	IP-ANN
Cleveland heart dataset	0.57	0.67
Heart failure clinical data	0.44	0.48
Framingham dataset	0.13	0.19

[6] Q. Wu, Z. Ma, G. Xu, S. Li, and D. Chen, "A novel neural network classifier using beetle antennae search algorithm for pattern classification," *IEEE access*, vol. 7, pp. 64 686–64 696, 2019.

[7] K. Meng, Z. Y. Dong, D. H. Wang, and K. P. Wong, "A self-adaptive rbf neural network classifier for transformer fault analysis," *IEEE Transactions on Power Systems*, vol. 25, no. 3, pp. 1350–1360, 2010.

[8] S. Thiria, C. Mejia, F. Badran, and M. Crepon, "A neural network approach for modeling nonlinear transfer functions: Application for wind retrieval from spaceborne scatterometer data," *Journal of Geophysical Research: Oceans*, vol. 98, no. C12, pp. 22 827–22 841, 1993.

[9] A. Hoekstra and R. P. Duin, "On the nonlinearity of pattern classifiers," in *Proceedings of 13th international conference on pattern recognition*, vol. 4. IEEE, 1996, pp. 271–275.

[10] D.-S. Huang and S.-D. Ma, "Linear and nonlinear feedforward neural network classifiers: a comprehensive understanding," *Journal of Intelligent Systems*, vol. 9, no. 1, pp. 1–38, 1999.

[11] F. Ali, S. El-Sappagh, S. R. Islam, D. Kwak, A. Ali, M. Imran, and K.-S. Kwak, "A smart healthcare monitoring system for heart disease prediction based on ensemble deep learning and feature fusion," *Information Fusion*, vol. 63, pp. 208–222, 2020.

[12] K. Makantasis, A. Doulamis, N. Doulamis, A. Nikitakis, and A. Voulodimos, "Tensor-based nonlinear classifier for high-order data analysis," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 2221–2225.

[13] C. A. Martín, R. I. Pinillo, A. Barasinski, and F. Chinesta, "Code2vect: An efficient heterogenous data classifier and nonlinear regression technique," *Comptes Rendus Mécanique*, vol. 347, no. 11, pp. 754–761, 2019.

[14] A. G. Ivakhnenko, "Polynomial theory of complex systems," *IEEE transactions on Systems, Man, and Cybernetics*, no. 4, pp. 364–378, 1971.

[15] S.-K. Oh and W. Pedrycz, "The design of self-organizing polynomial neural networks," *Information Sciences*, vol. 141, no. 3-4, pp. 237–258, 2002.

[16] S.-K. Oh, W. Pedrycz, and H.-S. Park, "Genetically optimized fuzzy polynomial neural networks," *IEEE Transactions on Fuzzy Systems*, vol. 14, no. 1, pp. 125–144, 2006.

[17] W. Huang, S.-K. Oh, and W. Pedrycz, "Fuzzy wavelet polynomial neural networks: analysis and design," *IEEE Transactions on Fuzzy Systems*, vol. 25, no. 5, pp. 1329–1341, 2016.

[18] S.-K. Oh, W. Pedrycz, and B.-J. Park, "Polynomial neural networks architecture: analysis and design," *Computers & Electrical Engineering*, vol. 29, no. 6, pp. 703–725, 2003.

[19] S.-K. Oh, W.-D. Kim, B.-J. Park, and W. Pedrycz, "A design of granular-oriented self-organizing hybrid fuzzy polynomial neural networks," *Neurocomputing*, vol. 119, pp. 292–307, 2013.

[20] S.-B. Roh, S.-K. Oh, and W. Pedrycz, "Design of fuzzy radial basis function-based polynomial neural networks," *Fuzzy sets and systems*, vol. 185, no. 1, pp. 15–37, 2011.

[21] I. Maric, "Optimization of self-organizing polynomial neural networks," *Expert systems with applications*, vol. 40, no. 11, pp. 4528–4538, 2013.

[22] R. Ghazali and D. Al-Jumeily, "Application of pi-sigma neural networks and ridge polynomial neural networks to financial time series prediction," in *Artificial Higher Order Neural Networks for Economics and Business*. IGI Global, 2009, pp. 271–293.

[23] M. M. Al-Tahrawi and S. N. Al-Khatib, "Arabic text classification using polynomial networks," *Journal of King Saud University-Computer and Information Sciences*, vol. 27, no. 4, pp. 437–449, 2015.

[24] J. Kileel, M. Trager, and J. Bruna, "On the expressive power of deep polynomial neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[25] F. Piazza, A. Uncini, and M. Zenobi, "Artificial neural networks with adaptive polynomial activation function," in *Proc. of the IJCNN*, vol. 2, 1992, pp. 343–349.

[26] M. E. Ulug, "Artificial neural network method and architecture," Apr. 18 1995, uS Patent 5,408,588.

[27] R. Detrano, A. Janosi, W. Steinbrunn, M. Pfisterer, J.-J. Schmid,

S. Sandhu, K. H. Guppy, S. Lee, and V. Froelicher, "International application of a new probability algorithm for the diagnosis of coronary artery disease," *The American journal of cardiology*, vol. 64, no. 5, pp. 304–310, 1989.

[28] D. Chicco and G. Jurman, "Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone," *BMC medical informatics and decision making*, vol. 20, no. 1, pp. 1–16, 2020.

[29] M. H. Rafiei, W. H. Khushefati, R. Demirboga, and H. Adeli, "Supervised deep restricted boltzmann machine for estimation of concrete," *ACI Materials Journal*, vol. 114, no. 2, p. 237, 2017.

[30] M. H. Rafiei and H. Adeli, "A novel machine learning-based algorithm to detect damage in high-rise building structures," *The Structural Design of Tall and Special Buildings*, vol. 26, no. 18, p. e1400, 2017.

[31] U. R. Acharya, S. L. Oh, Y. Hagiwara, J. H. Tan, and H. Adeli, "Deep convolutional neural network for the automated detection and diagnosis of seizure using eeg signals," *Computers in biology and medicine*, vol. 100, pp. 270–278, 2018.

[32] D. Chicco and G. Jurman, "Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone," *BMC medical informatics and decision making*, vol. 20, no. 1, pp. 1–16, 2020.

[33] A. B. Fernandez, M. J. Keyes, M. Pencina, R. D'Agostino, C. J. O'Donnell, and P. D. Thompson, "Relation of corneal arcus to cardiovascular disease (from the framingham heart study data set)," *The American journal of cardiology*, vol. 103, no. 1, pp. 64–66, 2009.

[34] A. K. Gárate-Escamila, A. H. El Hassani, and E. Andrès, "Classification models for heart disease prediction using feature selection and pca," *Informatics in Medicine Unlocked*, vol. 19, p. 100330, 2020.

[35] D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation," *BMC genomics*, vol. 21, pp. 1–13, 2020.

[36] N. Lopes and B. Ribeiro, "An evaluation of multiple feed-forward networks on gpus," *International journal of neural systems*, vol. 21, no. 01, pp. 31–47, 2011.

[37] M. Asaduzzaman, M. Shahjahan, and K. Murase, "Faster training using fusion of activation functions for feed forward neural networks," *International journal of neural systems*, vol. 19, no. 06, pp. 437–448, 2009.

Mahalingam P. R. received his B.Tech (2010) in Computer Science and Engineering, and M.Tech (2013) in Computer science and Engineering (Information Systems) from Mahatma Gandhi University, Kerala. He is presently pursuing PhD in Healthcare Data Analytics at School of Computer Science and Engineering at Vellore Institute of Technology, Vellore. His research interests include machine learning, data analytics, blockchain, cloud computing, and evolutionary algorithms.

Dheeba J. received her B.E (2005), M.E (2008) and Ph.D (2013) in Computer science and Engineering from Anna University, Chennai. She is presently working as Associate Professor in School of Computer Science and Engineering at Vellore Institute of Technology, Vellore. Her research interests include medical image analysis, Machine learning approaches and evolutionary algorithms.