ISSN: 2831 - 6789

Research Article

Journal of Robotics and Automation Research

A Study of Artificial Neural Network and Its Implementation from Scratch

***Nilay Kushawaha, Ankhi Roy**

[1]*Department of Physics, Indian Institute of Technology Indore, Khandwa, 453552 Indore, India.*

[2]*Department of Physics, Indian Institute of Technology Indore, Khandwa, 453552 Indore, India.*

***Corresponding author**
Nilay Kushawaha. Department of Physics, Indian Institute of Technology Indore, Khandwa, 453552 Indore, India.

Submitted: 23 May 2022; **Accepted:** 31 May 2022; **Published:** 09 Jun 2022

Citation: *Nilay Kushawaha, Ankhi Roy.(2022). A study of Artificial Neural Network and its implementation from scratch. J Robot Auto Res 3(2), 158-164.*

Abstract

One of the major problems in the field of artificial intelligence (AI) is the use of machine learning model as a black box, even though it might be helpful in a few cases but understanding the internal structure and the operating mechanism will assist the user to tweak the variables in a more efficient and productive manner. In this paper we have introduced the working of an artificial neural network (ANN) by taking the example of a three layered neural network. The entire mathematics behind the working of neural network along with the different evaluation metrics required to assess the performance of the model are discussed in this paper. We have also created a custom neural network from scratch and compared it with the keras based model on three different datasets - susy dataset [8], cardiovascular dataset [6], churn dataset [7]. The results obtained demonstrates that the overall performance of both the models are almost identical which gives an idea that it is possible to train a neural network from scratch without the use of any framework.

Keywords: Artificial neural network (ANN); Activation function; Optimizer; Efficiency; Object oriented programming (OOP)

Introduction

ARTIFICIAL Neural Networks (ANN) have been around for quite a long time, they have been studied for many years in the hope of achieving human-like performance. ANN can be thought of as analogous to a very small network in the cerebral cortex. It is possible to mimic certain parts of the neurons such as dendrites, cell bodies, and axons using a simplified mathematical model. The biological neuron has many similarities with the artificial neuron, the input nodes in an artificial neuron are same as the dendrites in a biological neuron. The axon and the axon terminal of the biological neuron transfer the synapses from the dendrites to the synaptic cleft only when the strength of signal is greater than a certain threshold value just like a step function [1].

The working of an ANN can simply be understood as the mapping of function from one space to some other space either linearly or using some higher-order relation. It can be used for solving both classification as well as regression problems. In principle an ANN mainly consists of three layers: an input layer, one or more hidden layers and an output layer. The Perceptron is one of the simplest ANN architectures, with only an input layer and an output layer, or it is simply a one-to-one mapping of the input and output layer. The shortcomings of the perceptron were soon realized since a single layer of perceptron was unable to solve the non-linear classification problem. It was also unable to approximate any continuous functional mapping.

However this problem can be solved by stacking multiple layers of perceptron or using a fully connected perceptron layer [2].

The only restriction is that the network must be feed-forward, so that it contains no feedback loops. This ensures that the network outputs can be calculated as explicit functions of the input weights. The objective of this paper is to present the working of an ANN in the simplest form possible both for classification as well as regression. It is seen that people can train a neural network using some framework but they struggle to understand the mathematics associated with it. In this paper, we will first understand the mathematics behind the working of a neural network and then train a neural network from scratch using the bare minimum mathematics required. The programming language we are going to use is python, however the same can be done in any object-oriented programming (OOP) language. In order to compare the performance of the custom model and the keras based model, we will be using three different datasets coming from different fields (biomedical, high energy physics, and industrial sector), more details about the dataset will be discussed in the later section.

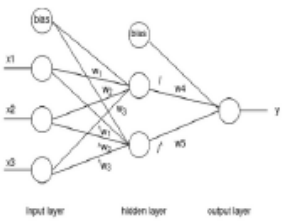

Figure 1 *Artificial Neural Network with 3 input neurons, 2 hidden neurons and one output neuron*

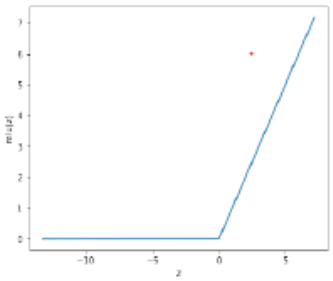

Figure 2 *Relu activation function*

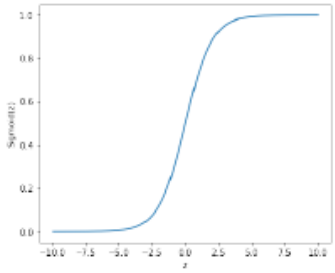

Figure 3 *Sigmoid activation function*

Activation function and Optimizer

Activation functions are used to bring some sort of non-linearity in the network such that it can learn more complex patterns from the data. Without activation function, it simply is a one-to-one mapping of the function. There are various activation functions out there, some of them are sigmoid, softmax, tanh, relu, leaky Kushawaha and Roy relu, elu, etc. Sigmoid, softmax, tanh are mostly used in the output layer of the neural network whereas relu and its variants are used in the input and hidden layer.

We will be mainly focusing on Rectified Linear Unit (ReLU) activation function as shown in figure 2 and one of its variants, leaky relu [3]. The output of relu keeps on increasing linearly for a positive input value, whereas when the input gets negative the output of relu becomes zero which leads to the dead neuron issue. In order to solve this we multiply a small smoothing factor to the input of the neuron such that the output becomes a small negative number close to zero but not completely zero, thus avoiding the dead neuron problem. The reason for choosing leaky relu instead of relu in the hidden layer is that it does not completely eliminate the negative output values from the neurons. The activation function in the output layer is sigmoid as shown in figure 3, the motivation for choosing it is due to its ability to convert the positive numbers to a value close to 1 and the negative numbers to a value close to 0, thus allowing us to perform binary classification.

The optimizer we will be using is stochastic gradient descent (SGD) algorithm. Even though we have more advanced optimizers, still we are focusing on the old gradient descent method due to its simplicity and physical significance. The working of gradient descent can be understood by taking the example of a ball rolling down a hill, after traveling a certain amount of distance it reaches the point of lowest energy or the minima of the slope. There is a slight difference between the working of gradient descent and SGD algorithm [3].

In gradient descent, we need to load the complete training data at once in the memory (RAM), which might not be possible when we have a very large dataset consisting of million records. However, in the SGD algorithm instead of loading the complete data at once, we pass one instance at a time, calculate its gradient and then update the weights accordingly. Another advantage of SGD is that instead of reaching the global minima gently, the loss function will first bounce up and down in the starting phase which might help the algorithm to jump out of local minima.

The steps associated with gradient descent include :

1. Initializing the weights with some random value. However, the weight initialization is not that random there are various weight initialization techniques where the weights are drawn either from a random or a normal distribution [4].
2. Calculating the derivative of loss function so as to know the direction (sign) of the slope.
3. Updating the weights by using the slope and a small learning rate (η) using the formula.

$$\omega(t) = \omega(t-1) - \eta * \delta L/\delta\omega \quad (1)$$

The loss function used for classification and regression are binary-cross entropy and mean squared error. The expression for bina-

ry-cross entropy loss function is $L = -y_i\, l_n\,(y_i)+(1-y_i) l_n\,(1-y_i) / n$ and the expression for mean squared error (MSE) is $L=-(\sum(y_i-y_i)^2 / n$ We do not use the same loss function for both classification as well as regression because we want to penalize our model more in case of classification than in regression. The presence of "log" will yield a very high loss when the model's prediction is incorrect and vice-versa.

Working of Neural Network

To understand the working of a simple neural network, we need to have a basic understanding of matrix algebra [12]. Matrices are nothing but an array of numbers that could be of any dimension. The dot product of two matrices is an element-wise multiplication which results in a scalar value. The use of matrices in neural network makes the computation more faster and compresses all the calculations into simple notation.

In general, a neural network contains three layers :

1. Input layer: It takes the input vectors from the user and multiplies the respective branch weights to it.
2. Hidden layer: Hidden layer is the set of neurons where all the computations are performed on the input data. It is responsible for learning more complex patterns from the data.
3. Output layer: It gives the final predicted output value based on the input features. Figure 1 shows a simple artificial neural network with three input neurons, two hidden neurons and one output neuron. The presence of three input neurons or three independent features will allow the model to draw a decision boundary in the 3-dimensional space to separate the two classes. A single feature in the input layer is equivalent to fixing a threshold value, where the particle is categorized as a signal if the value of the feature is greater than the threshold.

The activation function used in the input layer, hidden layer is leaky relu and in the output layer is sigmoid. The inputs vectors x1, x2 and x3 are first normalized to the interval [0,1]. They are then multiplied with some random branch weights and a bias term is added to it along with an activation function on top of it. Weights and biases (commonly referred to as w and b) are the learnable parameters of a neural network. We are writing the algebraic form of the equation however, it can also be expressed in matrix form as product of a row and a column vector.

$$\kappa = (\omega_1.x_1 + \omega_2.x_2 + \omega_3.x_3 + b_1) \quad (2)$$
$$l = \text{leaky relu } (\kappa) \quad (3)$$
$$l = \max(\alpha * \kappa, \kappa) \quad (4)$$

"l" is the output of the first node of the hidden layer. α is the leakage that we introduce in the relu function, it is usually set to a small number (≈ 0.00001)

$$l = \max(\alpha * \omega_1.x_1 + \omega_2.x_2 + \omega_3.x_3 + b, \omega_1.x_1 + \omega_2.x_2 + \omega_3.x_3 + b_1) \quad (5)$$

Similarly, the output of the second node of hidden layer will be

$$\hat{l} = \max(\alpha * \omega^1.x_1 + \omega^2.x_2 + \omega^3.x_3 + b, \omega^1.x1 + \omega^2.x_2 + \omega^3.x_3 + b_1) \quad (6)$$
$$z = (\omega_4.l + \omega 5.\hat{l} + b^2) \quad (7)$$
$$\hat{y} = 1 / 1+e^{-z} \quad (8)$$

The predicted output ˆy is compared with the actual value and loss function is calculated, the loss function is then minimized using an optimizer (SGD in our case) by adjusting the weights. Let's try to understand the complete mathematics of weight updation by taking the case of classification. The loss function for classification is

$$L = -y_i\, l_n\,(y_i)+(1-y_i)\, l_n\,(1-y_i) / n \quad (9)$$

We assume that the output of the leaky relu activation function is always a positive number however, we will consider all these aspects when we code it in python. Also, we drop the constant "n" (total number of records) for simplicity. Now, that we have the loss function so we will calculate the required slope for all the weights, biases and then update each of them one by one in the upcoming iteration.

$$\frac{\delta L}{\delta w_1} = -\frac{\delta}{\delta w_1}\left(y_i ln(\hat{y_i}) + (1-y_i) ln(1-\hat{y_i})\right) \quad (10)$$

$$\frac{\delta L}{\delta w_1} = -\frac{\delta}{\delta w_1}\left(y_i ln(\frac{1}{1+e^{-z}}) + (1-y_i) ln(1-\frac{1}{1+e^{-z}})\right) \quad (11)$$

$$\frac{\delta L}{\delta w_1} = -\frac{y_i}{\hat{y_i}}\frac{\delta}{\delta w_1}\left((\frac{1}{1+e^{-z}})\right) - \frac{1-y_i}{1-\hat{y_i}}\frac{\delta}{\delta w_1}\left((1-y_i)(1-\frac{1}{1+e^{-z}})\right) \quad (12)$$

$$\frac{\delta L}{\delta w_1} = \frac{y_i}{\hat{y_i}}\frac{e^{-z}}{(1+e^{-z})^2}\frac{\delta}{\delta w_1}(-z) + \frac{1-y_i}{1-\hat{y_i}}\frac{e^{z}}{(1+e^{z})^2}\frac{\delta}{\delta w_1}(z) \quad (13)$$

$$\frac{\delta L}{\delta w_1} = -\frac{y_i}{\hat{y_i}}\frac{e^{-z}}{(1+e^{-z})^2} w_4 x_1 + \frac{1-y_i}{1-\hat{y_i}}\frac{e^{z}}{(1+e^{z})^2} w_4 x_1 \quad (14)$$

Similarly for bias,

$$\frac{\delta L}{\delta b_1} = -\frac{y_i}{\hat{y_i}}\frac{e^{-z}}{(1+e^{-z})^2}(w_4 + w_5) + \frac{1-y_i}{1-\hat{y_i}}\frac{e^{z}}{(1+e^{z})^2}(w_4 l + w_5 \hat{l}) \quad (15)$$

Table 1: Value of different metrics for custom and keras based model

Model	Dataset	Accuracy	Recall (0)	Recall (1)	Precision (0)	Precision (1)	F1 score (0)	F1 score (1)	ROC-AUC
Custom	Susy	0.77	0.87	0.65	0.75	0.81	0.80	0.72	0.758
	Cardiovascular	0.64	0.66	0.61	0.63	0.64	0.65	0.63	0.637
	Telecom churn	0.73	0.74	0.66	0.93	0.30	0.82	0.41	0.702
Keras	Susy	0.77	0.86	0.67	0.75	0.80	0.80	0.73	0.761
	Cardiovascular	0.64	0.65	0.63	0.64	0.64	0.64	0.64	0.639
	Telecom churn	0.71	0.70	0.78	0.95	0.30	0.81	0.43	0.739

The formula for weight, bias updation is :

$$w_1(t) = w_1(t-1) - \eta * \frac{\delta L}{\delta w_1} \tag{16}$$

$$b_1(t) = b_1(t-1) - \eta * \frac{\delta L}{\delta b_1} \tag{17}$$

$$w_1(t) = w_1(t-1) - \eta * \left(-\frac{y_i}{\hat{y_i}} \frac{e^{-z}}{(1+e^{-z})^2} w_4 x_1 + \frac{1-y_i}{1-\hat{y_i}} \frac{e^{z}}{(1+e^{z})^2} w_4 x_1 \right) \tag{18}$$

$$b_1(t) = b_1(t-1) - \eta * \left(-\frac{y_i}{\hat{y_i}} \frac{e^{-z}}{(1+e^{-z})^2} (w_4 + w_5) + \frac{1-y_i}{1-\hat{y_i}} \frac{e^{z}}{(1+e^{z})^2} (w_4 l + w_5 \hat{l}) \right) \tag{19}$$

In similar manner we can derive the weight updation formula for the remaining weights. The weight updation continues till we achieve the global minima or a considerable loss value. The same concept applies for regression as well, the only difference lies in the choice of loss function and the expression of slope that is calculated using the loss function [5].

Once we have the respective weights and biases, we just need to transverse through the network to get the final prediction. The predicted output will be a continuous value in the range 0 to 1, in order to convert it into binary class, we pass it through some threshold. The value of this threshold can be computed using the Receiver's Operating Characteristic (ROC) curve [13].

Data collection and metrics

About dataset

We are using the cardiovascular disease dataset from kaggle, susy dataset from UCI repository and telecom churn dataset from kaggle too. The shape of cardiovascular disease dataset and the susy dataset is approximately same with 15,000 records for training the model and 15,000 records for testing [6-8]. However, the size of telecom churn dataset is very small with only 776 records for training and 667 records for testing. The reason for selecting this dataset is to check the performance of the model on small dataset. We are selecting only the best three possible features keeping in mind that none of them are internally correlated. The target features of all the three datasets contains only two classes 0 and 1, where 0 denotes the negative class and 1 denotes the positive class.
[]η is the learning rate ≈ 0.0001

Evaluation metrics

The performance of any machine learning model is evaluated on the basis of some tandard metrics. In this section we will discuss some of them and their advantages over one another. A confusion matrix is a table often used to describe the performance of a classification model based on the actual and predicted outputs. There are four possible outcomes for a binary classifier : it can correctly classify a signal event (true positive, TP), classify a background event as signal (false positive, FP), correctly classify a background event (true negative, TN) or classify a signal event as background noise (false negative, FN). We can define various metrics from these values :

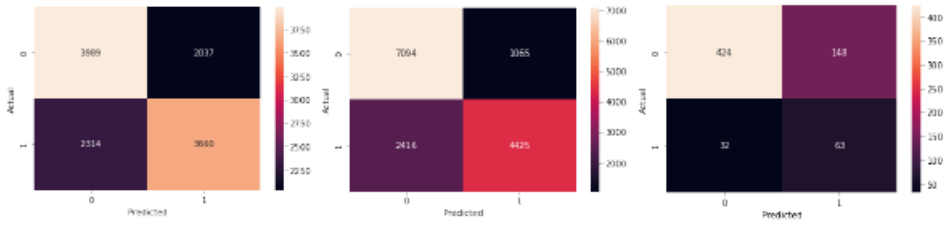

Figure 7 *Confusion matrix for custom model trained on cardiovascular dataset*

Figure 8 *Confusion matrix for custom model trained on susi dataset*

Figure 9 *Confusion matrix for custom model trained on telecom churn dataset*

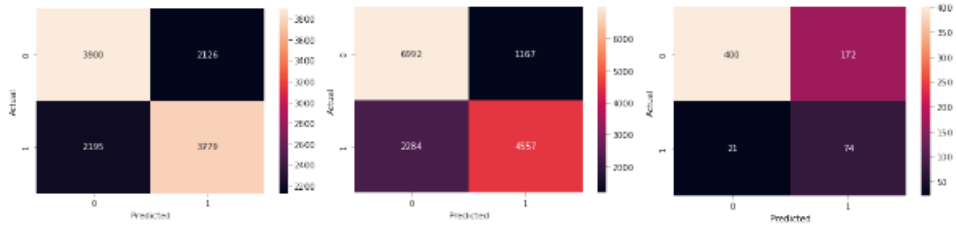

Figure 10 *Confusion matrix for keras model trained on cardiovascular dataset*

Figure 11 *Confusion matrix for keras model trained on susi dataset*

Figure 12 *Confusion matrix for keras model trained on telecom churn dataset*

Signal efficiency or recall is the ratio of the correctly identified signal to the total signal present in the dataset. The formula for recall is : Recall = TP / TP+FN

- Background rejection or specificity is the ability of the model to identify and discard the background events. The formula for specificity is : Specificity = TN / TN+FP
- Precision refers to the ratio of true positive to the total positive predictions made by the model. The formula for precision is : Precision = TP / TP+FP
- Accuracy refers to the number of correctly predicted data points out of all the data points present in the dataset. Accuracy could be a misleading metric if the dataset is highly imbalanced, in that case we can go with precision, recall or f1 score. The formula for accuracy is :
  Accuracy = TP+TN/TP+FP+TN+FN
- F1 score is a special case of F-beta score when beta equals to 1. It is defined as the harmonic mean of precision and recall. F1 score allows a model to be evaluated taking both precision and recall into account using a single score, which is helpful in comparing models. The formula for f1 score is :
  F1score = 2Precision.Recall / Precision+Recall
- ROC or receiver's operating characteristic is a trade off between the true positive rate (recall) and the false positive rate also called as false alarm rate. The area under the roc curve is used as summary of the model's skill. A model having an AUC equal to or less than 0.5 is considered as a non-skill or dumb model. We will be using all these metrics to compare the efficiency of our custom model with the keras based model.

Results

The efficiency of our custom model is compared with another model having same specifications but trained using the keras framework [9]. In case of susy and cardiovascular dataset, the models are trained for 100 epochs each whereas in case of telecom churn dataset the models are trained for 300 epochs. The reason for extra epochs in case of churn dataset is due to the low number of training records. Figure 4, 5, 6 shows the decrement in the value of loss function with respect to epochs for the three datasets. It can be seen that for the custom model the value of loss is initially high, one of the reasons for this could be the use of old gradient descent optimizer whereas in the keras based model we are using the state of art "adam" optimizer.

The confusion matrix of susy dataset for both custom and keras based model are nearly identical as shown in figure 8, 11. The roc curve of the custom model represented in green shade and the keras based model represented in black almost coincide with each other, which implies that the performance of both the models on the test dataset is almost same. The precision recall curve for both the models shows a comparable trend as shown in figure 14. Coming to the cardiovascular disease dataset, we can once more see that the confusion matrix for both custom and keras based model are approximately same as shown in figure 7, 10 and as a consequence the roc curve and the precision recall curve for both the models are also identical as shown in figure 13.

The final dataset is the telecom churn prediction. As mentioned earlier the number of training records in it is very low which makes the result suspicious and unreliable with respect to the other two datasets. However as shown in figure 18, the roc curve for the keras based model is slightly better than the custom model, the reason for the same could be the presence of low training records,

similar is the case with the precision recall curve. Table 1 displays the value of accuracy, precision, recall, f1 score and roc auc score for all the three datasets trained on the custom and keras based models. []False positive rate (FPR) is calculated as the number of false positive divided by the sum of the number of false positives and true negatives

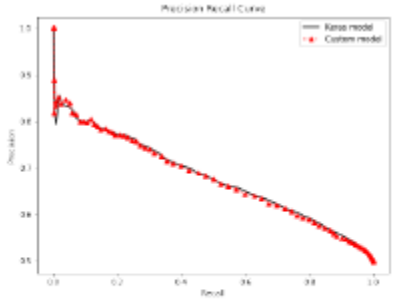

Figure 13 *Precision recall curve for cardiovascular dataset*

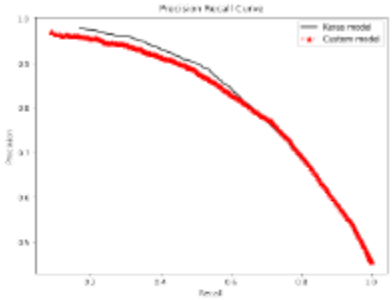

Figure 14 *Precision recall curve for susi dataset*

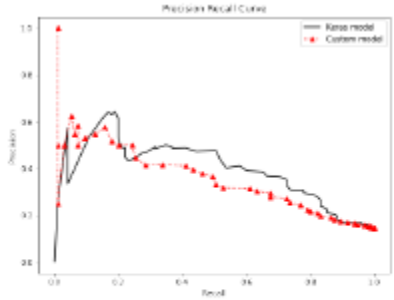

Figure 15 *Precision recall curve for telecom churn dataset*

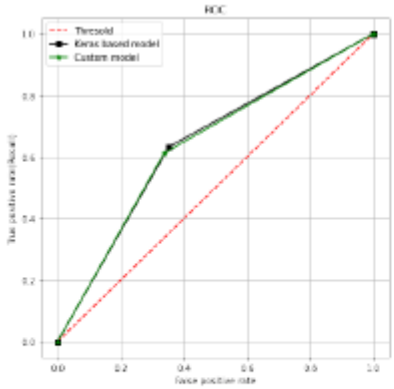

Figure 16 *ROC curve for cardiovascular dataset*

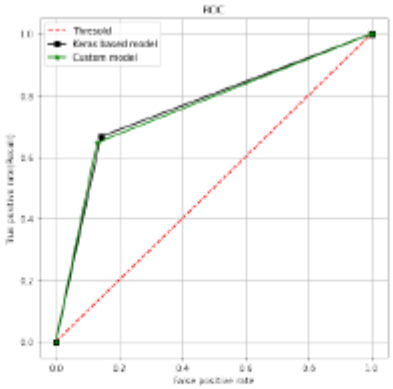

Figure 17 *ROC curve for susi dataset*

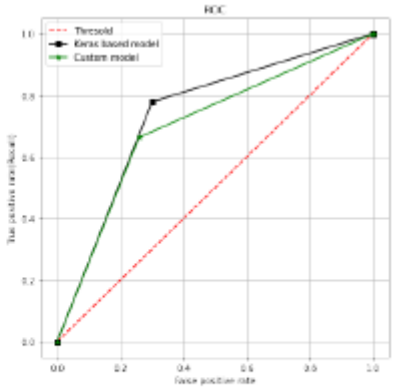

Figure 18 *ROC curve for telecom churn dataset*

Conclusion and discussion

The first objective of this paper is to present the working of artificial neural network for calculating the optimal set of weights and biases in the most simplest way possible and to develop a better understanding of the relationship between the structure of a neural network, its ability to perform input-output mapping. The mathematics introduced in this paper is sufficient enough for a beginner to get familiar about the working of artificial neural network and the different metrics used to evaluate the performance of model.

The second objective is to create a neural network from scratch without using any framework. The results presented in section 5 demonstrates that the performance of the custom neural network is comparable to the well known keras based model, which gives us an idea that we can create a custom neural network from scratch according to our use case. This could necessarily be helpful for the beginners to get a hands-on experience of how the weight updation takes place and a neural architecture ets trained. Though the code is not completely ready for use in production still it can be used for learning purpose, experimentation, etc.

Availability of data and materials

In this paper we have used the open sourced data from kaggle and UCI repository. The respective links for all the datasets along with the author's name are mentioned in the reference section.

Competing interests

The authors declare that they have no competing interests. [] Github repository link for custom neural network : https://github.com/nilay121/ Simple-ANN-from-Scratch

Funding

Not applicable

Author's contributions

Nk wrote the custom code and analyzed the datasets for keras and custom based model. AR helped in interpreting the results and writing the paper. All authors read and approved the final manuscript

Acknowledgements

Not applicable

References

1. Alzubi, J., Nayyar, A., & Kumar, A. (2018, November). Machine learning from theory to algorithms: an overview. In Journal of physics: conference series (Vol. 1142, No. 1, p. 012012). IOP Publishing.
2. Schuman, C. D., & Birdwell, J. D. (2013). Dynamic artificial neural networks with affective systems. PloS one, 8(11), e80455.
3. Géron, A. (2017). Hands-on machine learning with scikit-learn and tensorflow: Concepts. Tools, and Techniques to build intelligent systems.
4. Datta, L. (2020). A survey on activation functions and their relation with xavier and he normal initialization. arXiv preprint arXiv:2004.06632.
5. Bishop, C. M. (1995). Neural networks for pattern recognition. Oxford university press.
6. Cardiovascular disease dataset, Svetlana Ulianova (kaggle).
7. Telecom Churn Dataset, Baligh Mnassri (kaggle).
8. SUSY Data Set UCI, Daniel Whiteson.
9. Introduction to keras, Ketkar, Nikhil, Springer.
10. JETNET 3.0—A versatile artificial neural network package, Carsten Peterson and Thorsteinn R¨ognvaldsson and Leif L¨onnblad.
11. Van Der Smagt, P. P., Krˆse, B. J., Krose, B. J., & Smagt, P. P. (1993). An introduction to neural networks.
12. Arfken, G. B., & Weber, H. J. (1999). Mathematical methods for physicists.
13. Body, R., Carlton, E., Sperrin, M., Lewis, P. S., Burrows, G., Carley, S., ... & Mackway-Jones, K. (2017). Troponin-only Manchester Acute Coronary Syndromes (T-MACS) decision aid: single biomarker re-derivation and external validation in three cohorts. Emergency Medicine Journal, 34(6), 349-356.