TRANSITUS

Received
January 04, 2026

Revised
January 23, 2026

Accepted
February 03 2026

Published
February 07, 2026

Keywords
BNNM, PDE, Nonlinear Equation, Solution Models, Neural Networks.

Multilayer Neural Network Models and their Application to Solving Nonlinear Partial Differential Equations

Sakshi Katiyar[†]

[†]Department of Mathematics, Harcourt Butler Technical University, Kanpur, India.

Corresponding Author: imsakshikatiyar2002@gmail.com

DOI/url: `https://journalmanager.transitus.in/index.php/jamss`

Abstract

One of the most powerful techniques for modelling and solving complex problems in different fields that are used in day-to-day life, is artificial neural networks (ANNs). Among them, the Multilayer Perceptron (MLP), i.e., feedforward neural network, has been widely used and found to be very successful in learning non-linear relationships in data. Mimicking neurons in the brain of a human, the MLPs process the data input by the user in a series of multiple layers, including weighted connections, bias terms, and an activation function, bit by bit, which then remaps the input to produce the output. The simplest MLP is made up of an input layer, one or more than one(i.e., multiple) hidden layers, and an output layer. This neural network model learns by modifying its internal parameters with the help of the backpropagation technique, which forces the model to iteratively reduce the error that arises between the output obtained and the output that was predicted. This learning involves optimization techniques such as gradient descent.
In my work, we consider the multilayer perceptron models applied to the context of nonlinear PDEs. In particular, we investigate the construction of the Bilinear Neural Network Models (BNNMs) and show how they can be utilized to obtain exact analytical solutions for strong nonlinear systems, e.g., the p-gBKP equation. With a tensorized neural network architecture embedded in the Hirota bilinear framework, the work demonstrates the capability of neural architectures for symbolic computing and mathematical modeling.

1 Introduction

Imagine if we go to another state and the boards are marked with a different language. We don't know how to read that language, but we can use our phone to translate it into the language that can be understood by us. This is one of the uses of the neural network [1]. This model is a fundamental component of machine learning or deep learning, in which the algorithm simulates the functioning of the human brain. To put it another way, neural networks analyze and identify patterns in data, just like the human brain does, and then forecast the results for a fresh set of data [1]. Such a network consists of multiple(or one) layers(or layer) of artificial neurons. The data enters through the first layer(commonly called the input layer), and the last layer(i.e., the output layer) predicts the output; between the two layers, there are one or many layers(also called hidden layers). Each input neuron in the first layer is transferred to the next layer with a channel, and each channel consists of a numerical value known as a weight. Now, we get a product of the input with its corresponding weight, and then each hidden layer receives the sum obtained. Every single neuron has a numeric value called the bias, that is added to the value now and then passed through the activation function, which determines if the values in the neurons in the last layer will be activated or not based on the activation function, and the activated neuron is now passed to the other layer. In this manner, the data is propagated to the function. This method is known as the forward propagation [1].
In the last layer, the neuron with the highest value determines the output, which is based on probability. Now, the predicted output is compared to the actual output of the system to predict the error. Now, the information travels backward through our network, and this method is known as back propagation [1]. The input values are now adjusted, and this process is repeated iteratively until the model is trained to predict the correct result. In recent years, neural networks have been useful in optimization and pattern recognition using techniques like gradient descent or back propagation to predict the output based on the analyzed data. Training of neural networks is like adjusting the weights by iteratively performing forward and back propagation to obtain the desired output based on the given input, which can also be described as the minimization of the error in predicting the output every iteration, and training the model based on the dataset [2].

1.1 Multilayer Perceptron

The most useful technique of neural networks, by the back propagation algorithm, is the multilayer perceptron, which is the extension of the original perceptron with one layer. It was proposed by Rosenblatt [3], in the year 1950 [2], being inspired by the biological neuron. As per him, perceptron models can only solve linearly separable problems [4]. In 1969, Minsky and Papert proved that single-layer perceptrons cannot solve even simple non-linear problems like XOR functions [5]. This led to the development of better models and architectures in perceptrons. In the 1980s, various researchers, as discussed by Rumelhart and D.E. in their paper, proposed that perceptrons can be exposed to multiple layers for better results using the back-propagation algorithm, which led to the development of MLPs, which used various optimization techniques like gradient descent [6]. The multilayer perceptron consists of one layer or more than one layers after the input layer, and the weight is now processed with the value input, and it may pass through the first hidden layer, which, on adding the bias, passes the numerical value to the second layer, and the process continues the same as a neural network.
The key components of a multilayer perceptron are discussed below:
Input Layer: This layer consists of several input nodes that represent the features of the dataset. Each input is typically a numeric value corresponding to a feature or attribute in a dataset.
Hidden Layer: Layers that are not directly connected to the environment are called the hidden layer. The neural network may contain one or more hidden layers. These types of networks are very powerful and extremely complicated. These hidden layers may change the condition all the time until they reach the

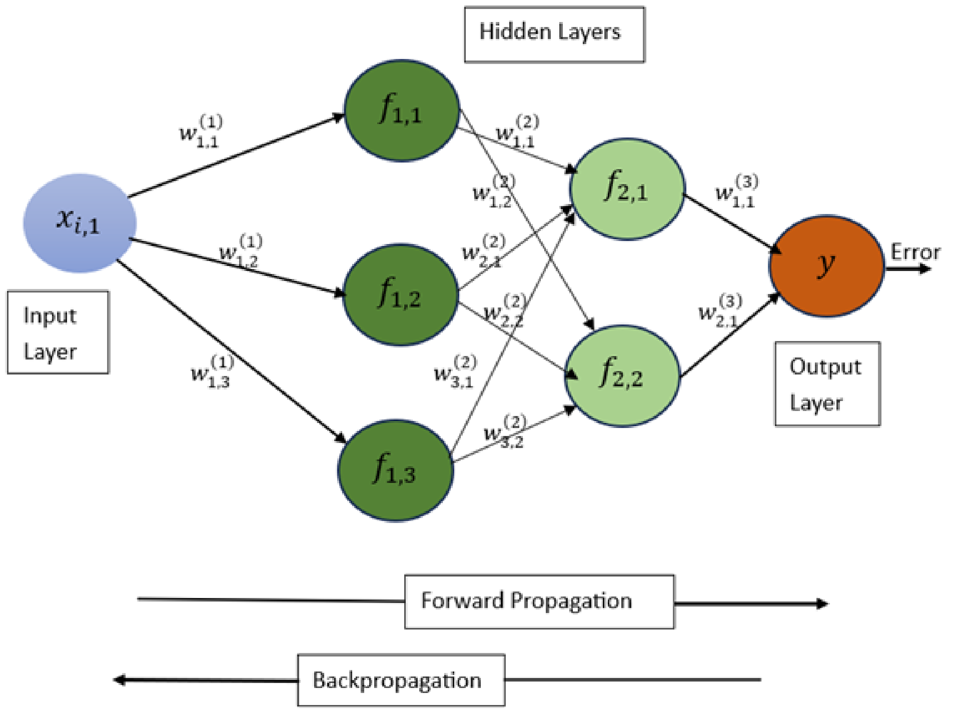


Figure 1: In the figure above, we have an input layer ($x_{i,1}$ corresponding to the weight $w_{1,1}^{(1)}$) ($w_{j,k}^{(l)}$ = weight of the neuron j in the layer l connecting to neuron k in the layer $l+1$), $f_{j,k}$ is the neuron k in the layer j, and y is the neuron in the output layer.

equilibrium state.
Activation Function: The Activation function is solely responsible for deciding if the value in the neurons in the last hidden layer will go to the output layer or not. This function, most commonly used today, is the logistic function, also known as the sigmoid function.
Threshold Function: We can say that the Threshold function is the activation function that is used to analyze the output, which depends on the final value of the sum of inputs. It is also called a step function or the Heaviside function.

2 Applications and Uses of Neural Networks

Neural Network, as of the time, has turned out to be one of the most powerful tools that is widely used by scientists and researchers to find the exact solutions for different nonlinear partial differential equations [7].

Neural Networks are not just used in a theoretical or experimental way to find the exact solution, but also in real-life day-to-day applications such as image classification, medical diagnosis, stock market prediction, and natural language processing. Libraries like PyTorch or TensorFlow in Python are widely used to build a feedforward neural network and prepare a model for industry applications [7]. Let us see one real-life application of a neural network model using Python libraries to build and train a neural network to recognize handwritten digits (0–9) from 8×8 grayscale images using a machine learning (Logistic) classifier. The Python code for the implementation of this application and the result are as follows:

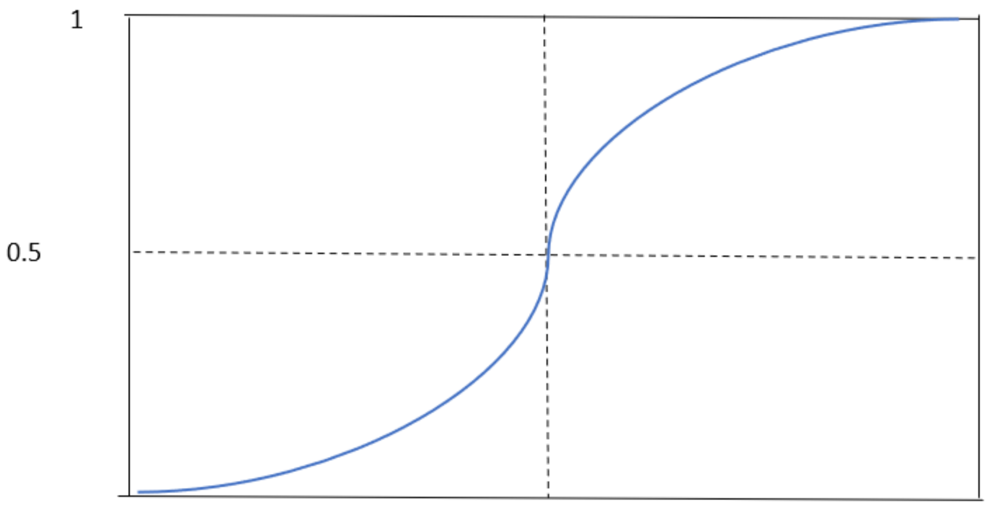
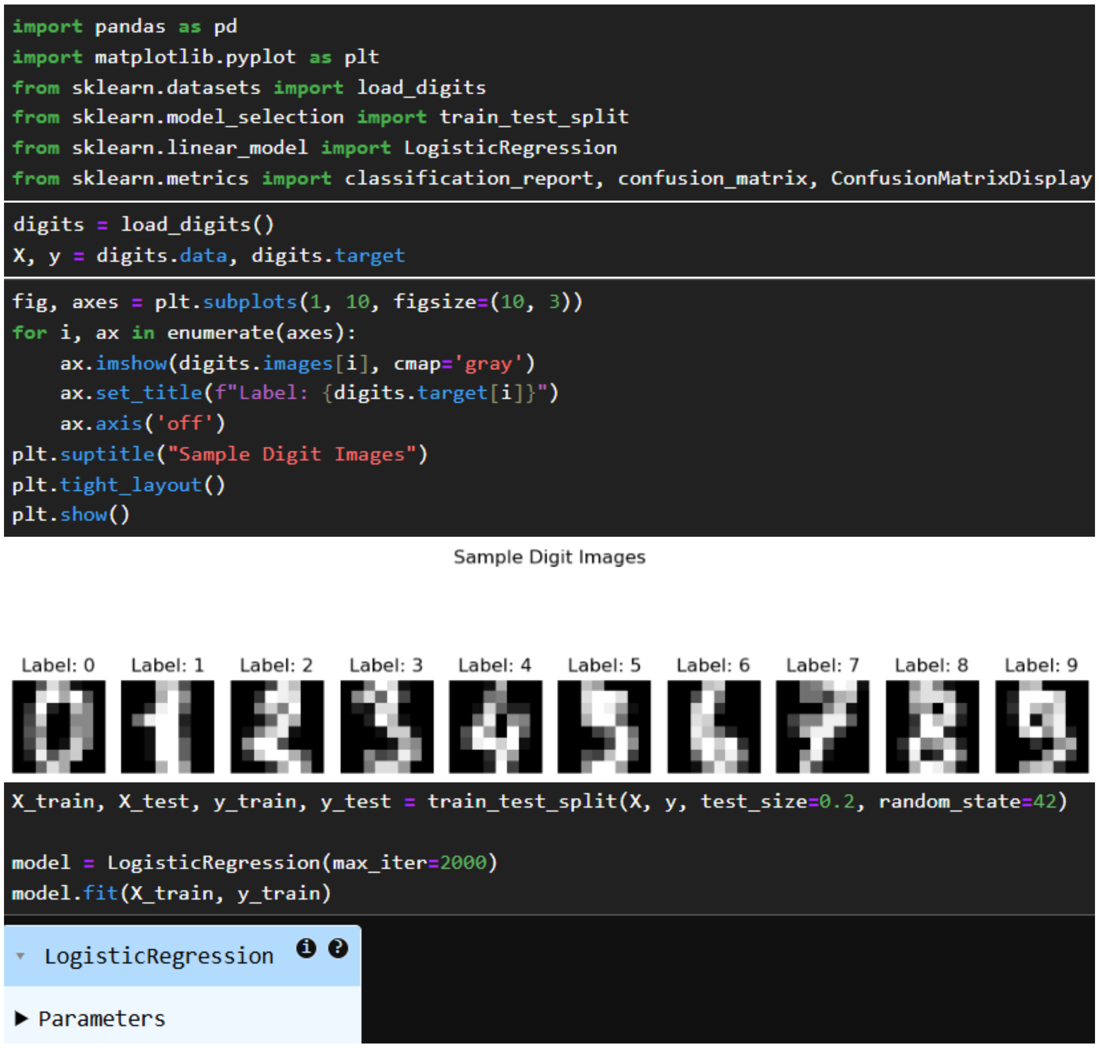


Figure 2: Sigmoid Function

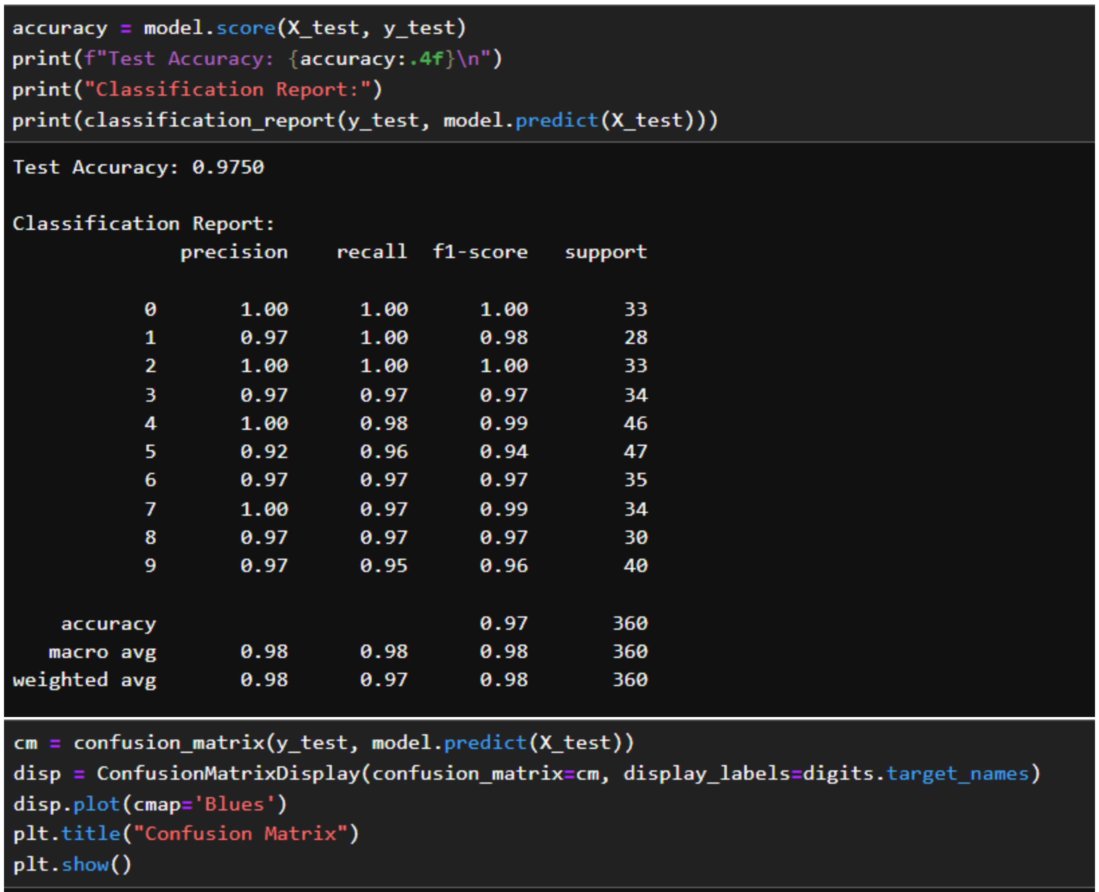


```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
```

```
digits = load_digits()
X, y = digits.data, digits.target
```

```
fig, axes = plt.subplots(1, 10, figsize=(10, 3))
for i, ax in enumerate(axes):
    ax.imshow(digits.images[i], cmap='gray')
    ax.set_title(f"Label: {digits.target[i]}")
    ax.axis('off')
plt.suptitle("Sample Digit Images")
plt.tight_layout()
plt.show()
```

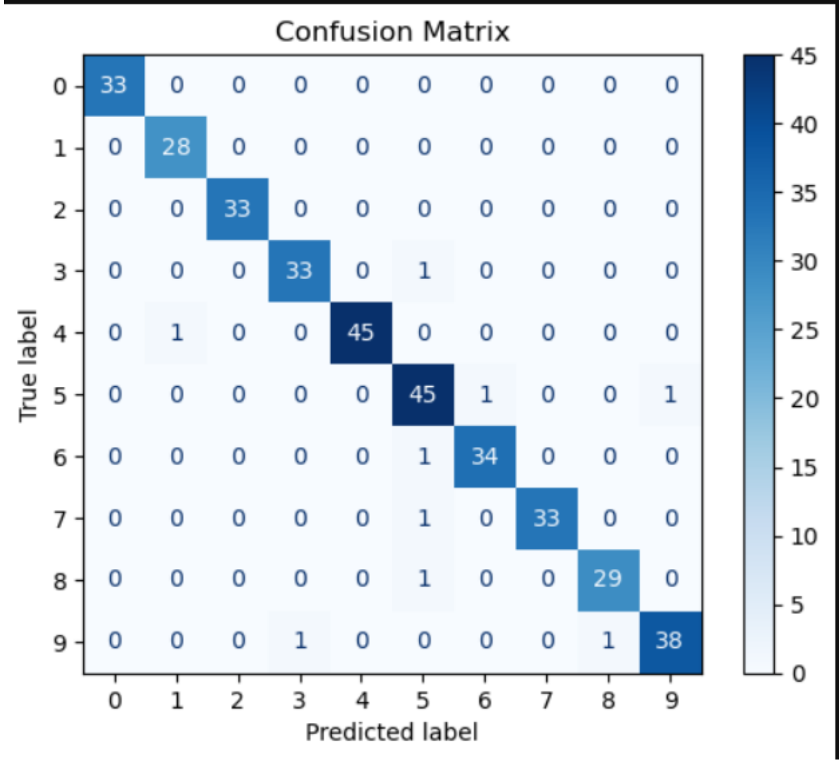
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LogisticRegression(max_iter=2000)
model.fit(X_train, y_train)
```

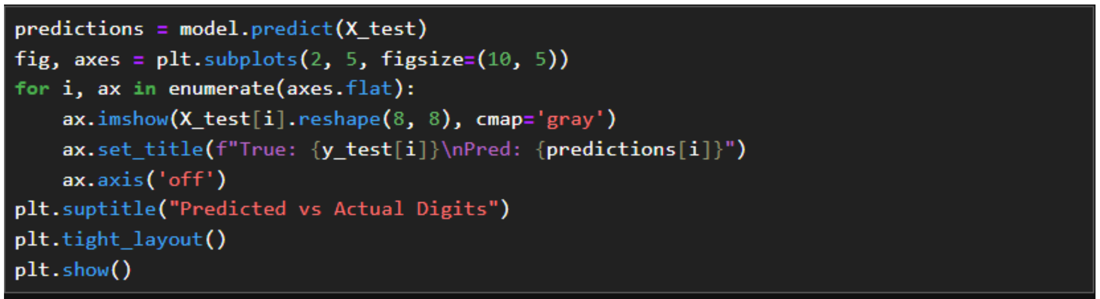


```
accuracy = model.score(X_test, y_test)
print(f"Test Accuracy: {accuracy:.4f}\n")
print("Classification Report:")
print(classification_report(y_test, model.predict(X_test)))
```

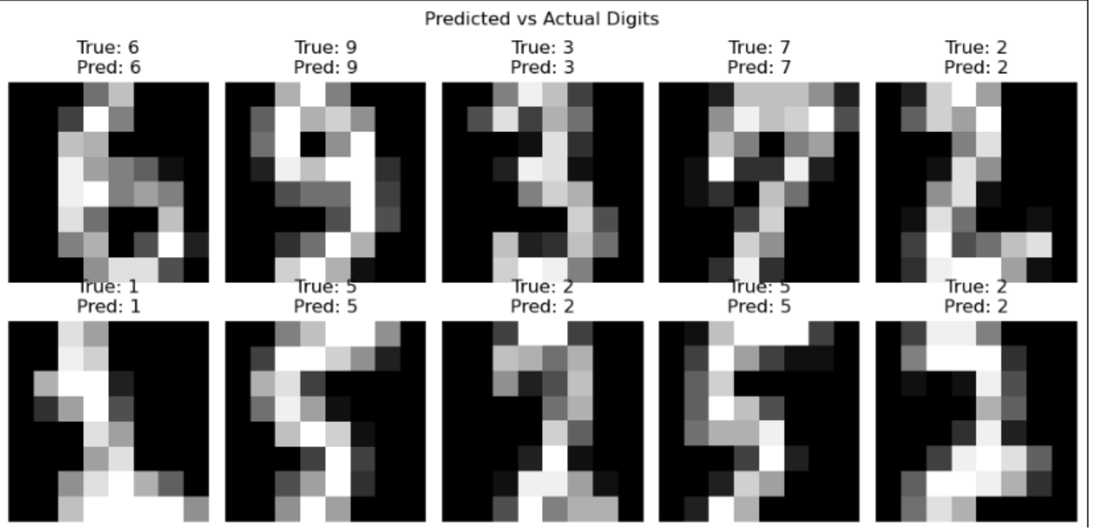
```
Test Accuracy: 0.9750

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        33
           1       0.97      1.00      0.98        28
           2       1.00      1.00      1.00        33
           3       0.97      0.97      0.97        34
           4       1.00      0.98      0.99        46
           5       0.92      0.96      0.94        47
           6       0.97      0.97      0.97        35
           7       1.00      0.97      0.99        34
           8       0.97      0.97      0.97        30
           9       0.97      0.95      0.96        40

    accuracy                           0.97       360
   macro avg       0.98      0.98      0.98       360
weighted avg       0.98      0.97      0.98       360
```

```
cm = confusion_matrix(y_test, model.predict(X_test))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=digits.target_names)
disp.plot(cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```





```python
predictions = model.predict(X_test)
fig, axes = plt.subplots(2, 5, figsize=(10, 5))
for i, ax in enumerate(axes.flat):
    ax.imshow(X_test[i].reshape(8, 8), cmap='gray')
    ax.set_title(f"True: {y_test[i]}\nPred: {predictions[i]}")
    ax.axis('off')
plt.suptitle("Predicted vs Actual Digits")
plt.tight_layout()
plt.show()
```



In this application, We have used the BSD open-licensed dataset from scikit-learn (named as Digits Dataset and can be obtained through the link: `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html`). We have used Logistic regression, which is a simple yet powerful classifier for multiclass image recognition. The test accuracy after training the model is approximately 93 percent to 97 percent.

3 Types of Neural Network Models

Neural Networks have become pivotal in artificial intelligence research. Over the years, several architectures have emerged, each tailored to specific data types, tasks, and learning constraints. Let's discuss a few of them for a better understanding of Neural Network models.

3.1 Feedforward Neural Networks (FNNs)

Feedforward is the earliest type of artificial neural network. In this type, the connection between the nodes does not form a cycle. This type of neural network may have one or more sets of hidden layers, which are connected to the input layer on one end and to the output layer on the other, just like the normal structure of every neural network. The activation functions majorly used in this type of neural network are Sigmoid, Tanh, or ReLU [8]. This type of neural network can be used in tasks such as Pattern Recognition, Function Approximation, or Structured Data Classification [8].

3.2 Convolutional Neural Networks (CNNs)

CNNs are a type of neural network that is designed to automatically and adaptively learn spatial hierarchies of features from visual data. Some of the notable layers of CNNs are Conv2D, MaxPooling2D, Flatten, and

Dense [9]. Such neural networks focus mainly on tasks like Image Classification (ImageNet, MNIST), Object Detection (YOLO, R-CNN), or Medical Imaging Diagnostics [9].

3.3 Recurrent Neural Networks (RNNs)

RNN models are best suited for sequential data due to their internal memory. It allows memory/data to influence current outputs. Its architecture consists of feedback loops in hidden layers, shared parameters across time steps, and input that can vary in sequence length in such types of neural networks. Such models are mainly used in tasks like Time-Series forecasting, Language Modeling, and Audio Synthesis [10].

3.4 Long Short-term Memory Networks (LSTMs)

LSTMs are a very special kind of RNNs, which avoid vanishing gradient problems. The core units of the architecture of such neural networks are the cell state, input gate, forget gate, and output gates. LSTMs use Sigmoid + Tanh activation functions and can remember the information even longer than LSTMs. Such neural networks are mainly used in tasks like Machine translation, Speech-to-text systems, or Stock prediction [11].

3.5 Gated Recurrent Units (GRUs)

GRUs combine the forget and input gates into a single update gate, which simplifies LSTMs and leads to faster training. The architecture of GRU consists of the update gate and reset gate. GRUs consist of fewer parameters than LSTMs, but they give comparable performance. The main application fields of LSTMs are text generation, chatbots, and anomaly detection in sequential data [12].

3.6 Autoencoders (AEs)

Autoencoders are unsupervised neural networks that learn to encode input into a compressed representation and then decode it back. Its architecture consists of an encoder, a decoder, and a loss function. The fields that majorly use Autoencoders are anomaly detection, image denoising, and feature extraction [13].

3.7 Generative Adversarial Networks (GANs)

GANs consist of two networks, a generator that creates data and a discriminator that evaluates data as real or fake. The application fields of GAN are deepfake generation, data augmentation, and art and media synthesis [14].

4 Bilinear Neural Networ Method (BNNM)

Neural networks had their origin at the introduction of the perceptron by Rosenblatt (1958) [4], which was capable of solving linear separable problems. The restrictions of single-layer networks became apparent with the publication of Minsky and Papert (1969) [5], who showed that they could not represent functions such as XOR. This resulted in multilayer architectures, for which backpropagation had been introduced by Rumelhart and Hinton [6]. The study on the neural networks for solving nonlinear PDEs witnessed a major change by the introduction of BNNM, or the Bilinear Neural Network Method, in [8] by Zhang and Bilige in 2019. They incorporated neural networks within the framework of the Hirota bilinear formalism (HBF), where exact explicit solutions for some complex nonlinear equations, including the p-gBKP equation, can be generated. Subsequent work by Liu et al. (2023–2024) [9] obtained a general class of solutions comprising lump, breather, and interaction waves for a variety of (3+1)-dimensional nonlinear PDEs. Later developments also led to enhancements in the neural architecture, single and multi-layered models, and combination with symbolic computation tools such as Maple, Mathematica, or MATLAB. These models exhibit both theoretical stability and practical flexibility in deriving exact solutions with different initial and function parameters.

4.1 General Description of BNNM

To solve the nonlinear partial differential equations (PDEs), a new method was proposed by Zhang et al. [15]. In the nineteenth century, the study of nonlinear PDEs began in various fields that used the bilinear transformation for approximation. For example, Ma et al. proposed the procedure for obtaining the arbitrary function interaction solution and lump solution, The technique to obtain breather-type kink soliton solutions was proposed by Sun et al. [17]. Let's examine the billinear form and then tensor formula for the bilinear neural network for p-gBKP equation [15]

$$\sigma_3^1 = -1, \quad \sigma_3^2 = 1, \quad \sigma_3^3 = 1, \quad \sigma_3^4 = -1, \quad \sigma_3^5 = 1, \quad \sigma_3^6 = 1, \quad \sigma_3^7 = -1,$$
$$\sigma_3^8 = 1, \quad \sigma_3^9 = 1$$

Hence,

$$D_{(3,r)}D_{(3,t)}f\cdot f = 2f_{r,t}f - 2f_rf_t,$$
$$D^2_{(3,s)}f\cdot f = 2f_{s,s}f - 2f_s^2,$$
$$D^4_{(3,r)}f\cdot f = 6f_{r,r}^2,$$
$$D^3_{(p,r)}D_{(p,s)}f\cdot f = 6f_{r,r}f_{r,s}.$$

At $p = 2$:

$$D_{(2,t)}D_{(2,r)}f\cdot f = 2f_{rt}f - 2f_rf_t,$$
$$D^2_{(2,s)}f\cdot f = 2f_{ss}f - 2f_s^2,$$
$$D^4_{(2,r)}f\cdot f = 2f_{rrrr}f - 8f_{rrr}f_r + 6f_{rr}^2,$$
$$D^3_{(p,r)}D_{(p,s)}f\cdot f = 2f_{rrrs}f - 6f_{rrs}f_r + 6f_{rs}f_{rr} - 2f_sf_{rrr}.$$

i.e., we get the Hirota bilinear operator when $p = 2$ [15]. Now,

$$B_{(p\text{-gBKP}_s)}(u) = \frac{B_{(p\text{-gBKP}_s)}(f)}{f^2}.$$

Therefore, the relationship between the reduced p-gBKP equation [15] and the generalized p-gBKP equation is valid. Thus, if f solves the generalized p-gBKP equation, then the reduced p-gBKP equation will also be solved [15]. In Figure 3, we can see the tensor model of the neural network, where the neurons of the input layer are denoted by x_i, $f_{n,k}$ is the neuron k in the hidden layer n, and y is the neuron of the output layer. Let us say that we have k neurons in each hidden layer and a total of n hidden layers. Then, the tensor formula of the nonlinear neural network is provided by in order to determine the precise analytical solutions for the bilinear p-gBKP equation [15]:

$$y = w_{k,y}^{(n)} \cdot F_k^{(n)}\left(\gamma_k^{(n)}\right),$$

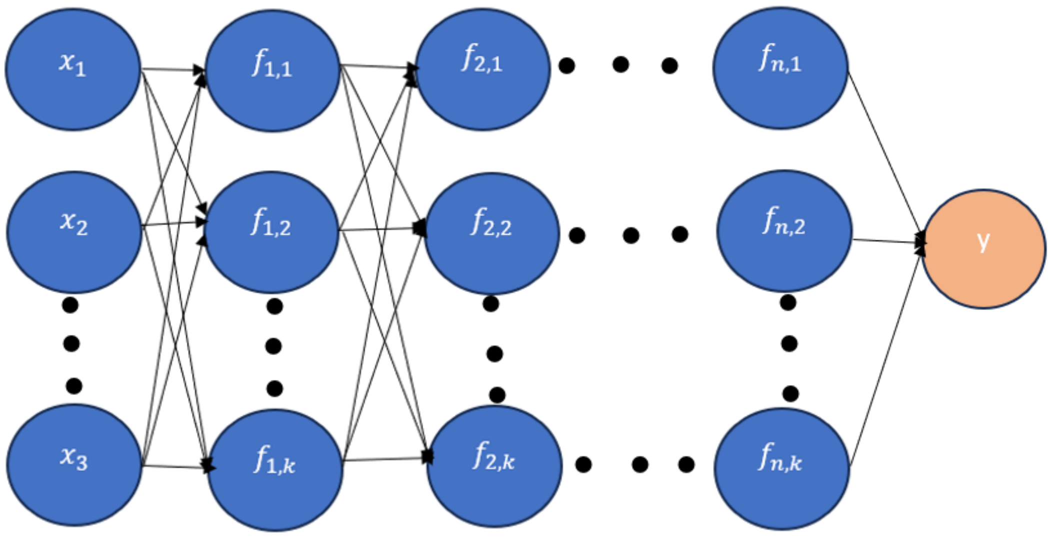


Figure 3: Structure of the Neural Network Model for the tensor formula.

where $w_{j,k}^{(l)}$ can be defined as the weight of the neuron k to $f_{n,k}$, $F_k^{(n)}$ is the generalized activation function applied to a neuron $f_{n,k}$ such that, in the last layer $F_k^{(n)}\left(\gamma_k^{(n)}\right) \geq 0$, and $\gamma_k^{(n)}$ is its pre-activation input, i.e., (weighted sum + bias) [15]:

$$\gamma_k^{(l)} = w_{j,k}^{(l-1)} \cdot F_j^{(l-1)}\left(\gamma_j^{(l-1)}\right) + b_k^{(l)}, \quad l = 1, 2, \ldots, n$$

where $b_k^{(l)}$ is the threshold, which acts as a constant here [15].
A complicated equation is produced by replacing the neural network-based expression with the bilinear version of the nonlinear partial differential equations. This is made simpler by setting the coefficients of each term in the resulting expression to zero, which results in an algebraic system of equations. The unknown coefficients are then found by solving these equations with symbolic computation tools like Maple, MATLAB, or Mathematica. To obtain precise analytical solutions for the system, the coefficients are found and then reinserted into the neural network structure and the bilinear transformation framework. [15].

4.2 Applications of BNNM via Classical Test Functions

Bilinear Neural Network Model also covers many classical methods of the exact solution of partial differential equations when specific functions are given to the single-layered network model [16].
(i) If the activation function of the first neuron $F_1(\gamma_1)$ is chosen to be (γ_1^2), $F_2(\gamma_2)$ is given an input (γ_2^2), and $F_3(\gamma_3)$ is given a constant value C, then this single-layered neural network is reduced to a lump solution's conventional test function, as shown in the figure below [16]:

$$y = 1 \cdot \gamma_1^2 + 1 \cdot \gamma_2^2 + 1 \cdot C,$$

$$y = \gamma_1^2 + \gamma_2^2 + C.$$

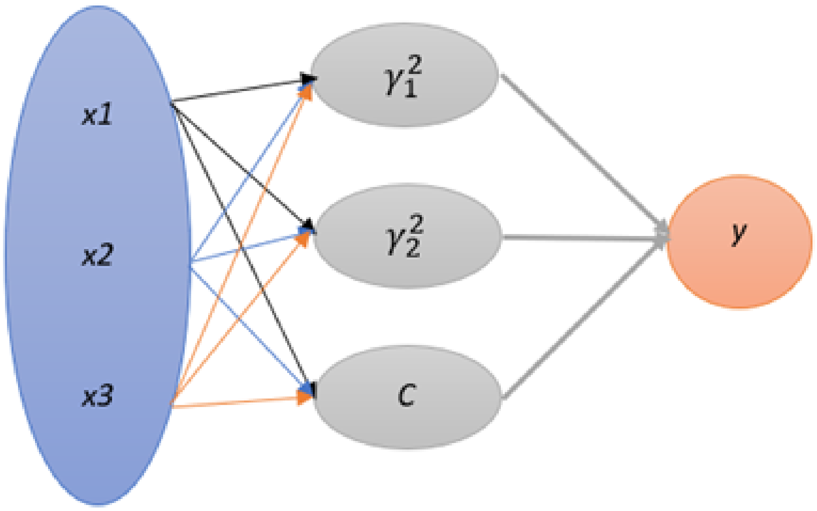


Figure 4

(ii) If $F_1(\gamma_1)$ is chosen to be $e^{-p_1\gamma_1}$, $F_2(\gamma_2)$ is $\cos(p\gamma_2)$, and $F_3(\gamma_3)$ is $e^{p_1\gamma_1}$ in this model, then the equation obtained is The breather-type kink soliton's classical test function, as shown below [16].

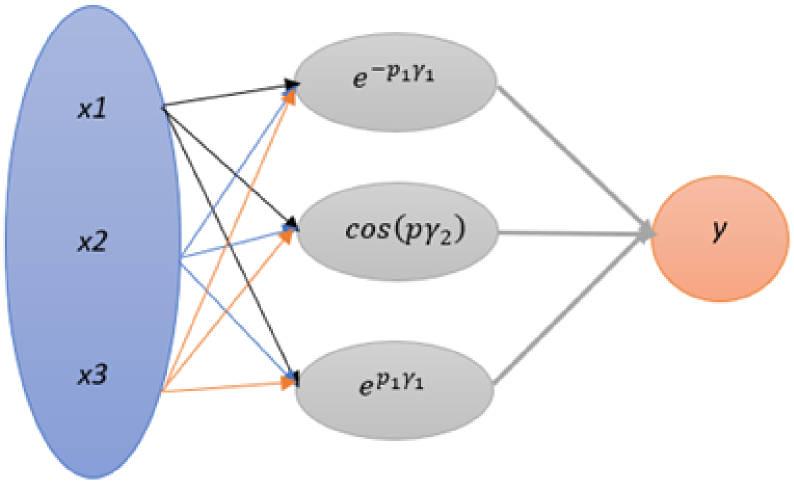


Figure 5

$$y = 1 \cdot e^{-p_1\gamma_1} + 1 \cdot \cos(p\gamma_2) + 1 \cdot e^{p_1\gamma_1}$$
$$y = e^{-p_1\gamma_1} + \cos(p\gamma_2) + e^{p_1\gamma_1}$$

Where both p and p_1 are parameters.

(iii) If $F_1(\gamma_1)$ is chosen to be 1, $F_2(\gamma_2)$ is γ_1^2, and $F_3(\gamma_3)$ is γ_3^2, in this model, then the equation obtained is The traditional rational solution test function, as shown in the figure below [16].

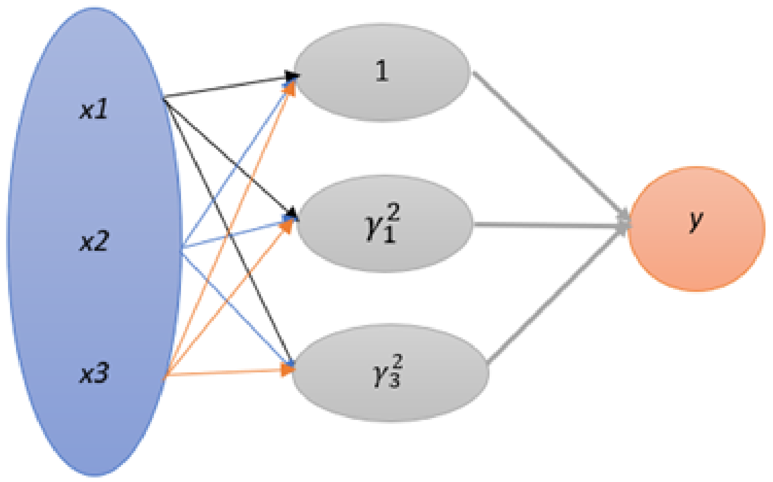


Figure 6

$$y = 1 \cdot 1 + 1 \cdot \gamma_2^2 + 1 \cdot \gamma_3^2$$
$$y = 1 + \gamma_2^2 + \gamma_3^2$$

(iv) If $F_1(\gamma_1)$ is chosen to be $e^{-\gamma_1}$, $F_2(\gamma_2)$ is $\tan(\gamma_2)$, $F_3(\gamma_3)$ is $\tan(\gamma_3)$, and $F_4(\gamma_4)$ is e^{γ_1}, in this model, then The resulting equation is the periodic wave equation's classical test function, as shown in the figure below [16].

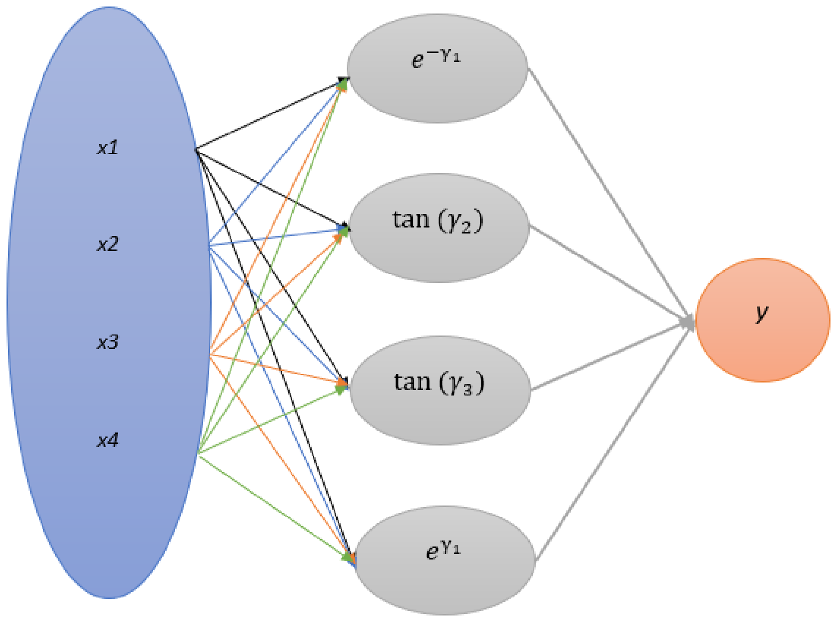


Figure 7

$$y = 1 \cdot e^{-\gamma_1} + 1 \cdot \tan(\gamma_2) + 1 \cdot \tan(\gamma_3) + 1 \cdot e^{\gamma_1}$$
$$y = e^{-\gamma_1} + \tan(\gamma_2) + \tan(\gamma_3) + e^{\gamma_1}$$

(v) If $F_1(\gamma_1)$ is chosen to be $e^{-\gamma_1}$, $F_2(\gamma_2)$ is $\cos(\gamma_2)$, $F_3(\gamma_3)$ is $\sin(\gamma_3)$, and $F_4(\gamma_4)$ is e^{γ_1}, in this model, then the equation obtained is the wave equation's classical test function, as shown in the figure below [16].

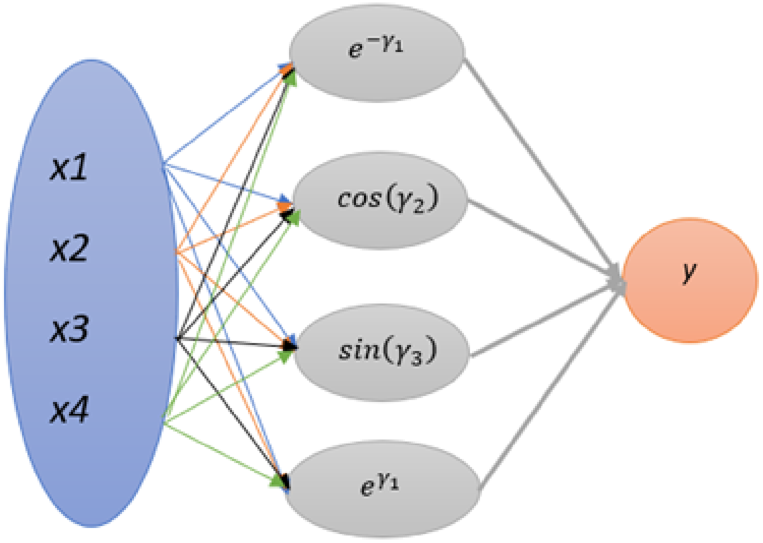


Figure 8

$$y = 1 \cdot e^{-\gamma_1} + 1 \cdot \cos(\gamma_2) + 1 \cdot \sin(\gamma_3) + 1 \cdot e^{\gamma_1}$$
$$y = e^{-\gamma_1} + \cos(\gamma_2) + \sin(\gamma_3) + e^{\gamma_1}$$

(vi) If $F_1(\gamma_1)$ is chosen to be γ_1^2, $F_2(\gamma_2)$ is γ_2^2, and $F_3(\gamma_3)$ is e^{γ_3} (or $\cosh(\gamma_3)$), in this single-layered model, then the equation obtained is The trigonometric hyperbolic cosine function, often known as the traditional test function of the interaction solution between the lump and exponential function, as shown in the figure below [16].

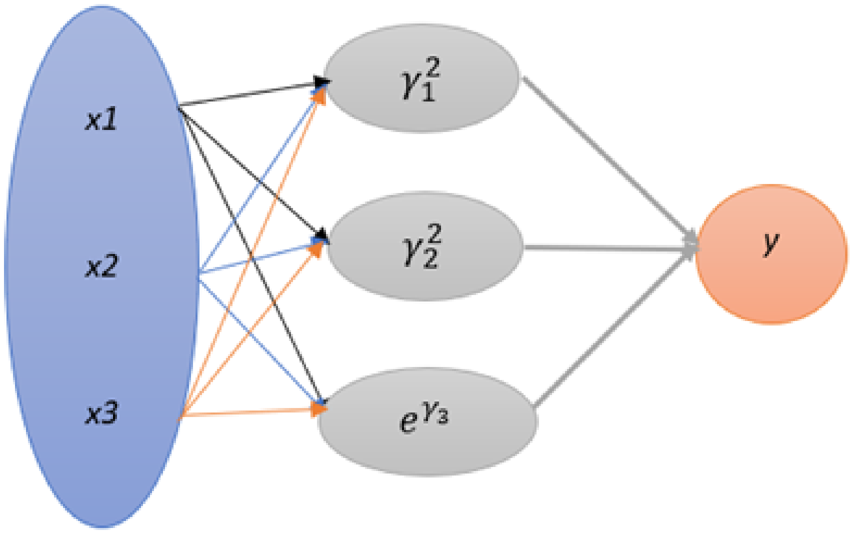


Figure 9

$$y = \gamma_1^2 + \gamma_2^2 + e^{\gamma_3} \quad \text{or} \quad y = \gamma_1^2 + \gamma_2^2 + \cosh(\gamma_3)$$

(vii) If $F_1(\gamma_1)$ is chosen to be γ_1^2, $F_2(\gamma_2)$ is γ_2^2, and $F_3(\gamma_3)$ is any arbitrary function in this single-layered model, then The resulting equation represents the traditional test function of the lump-arbitrary function interaction solution, as shown in the figure below [16].

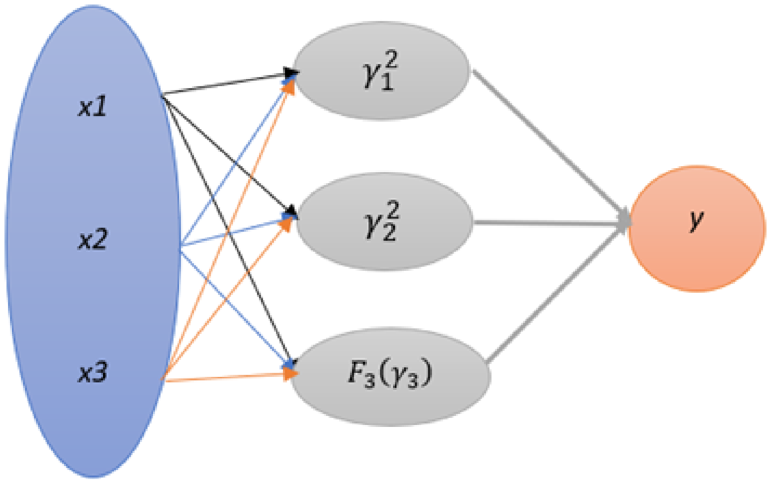


Figure 10

$$y = \gamma_1^2 + \gamma_2^2 + F_3(\gamma_3)$$

(viii) If $F_1(\gamma_1)$ is chosen to be γ_1^2, $F_2(\gamma_2)$ is γ_2^2, and $F_3(\gamma_3)$ is $\text{sech}(p_2, \gamma_3)$ in this single-layered model, then the equation obtained is The standard test function for the lump-hyperbolic secant function interaction solution, as shown in the figure below [16].

$$y = \gamma_1^2 + \gamma_2^2 + \operatorname{sech}(p_2, \gamma_3)$$

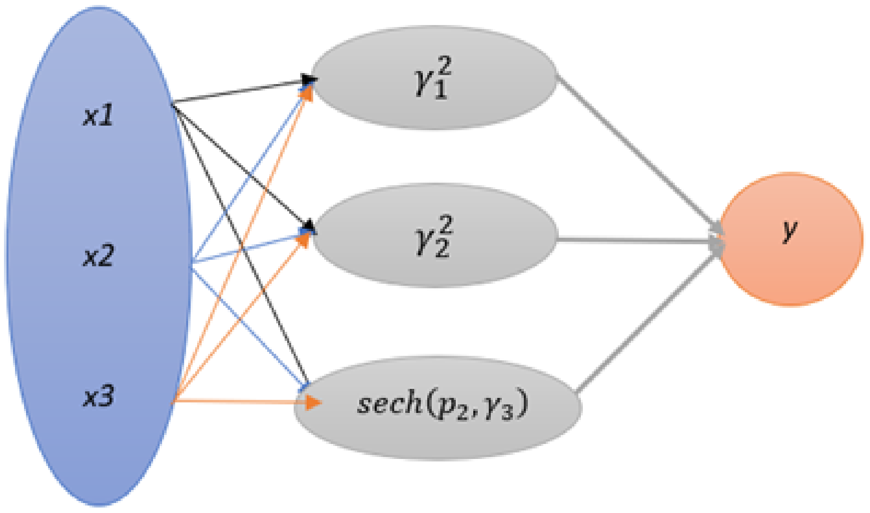


Figure 11

(ix) If $F_1(\gamma_1)$ is chosen to be $\cosh(-p_1\gamma_1)$, $F_2(\gamma_2)$ is $\cos(p\gamma_2)$, and $F_3(\gamma_3)$ is $\cosh(p_1\gamma_1)$ in this single-layered model, then the equation obtained is The Periodic lump-type solution's classical test function, as shown in the figure below [16].

$$y = e^{-p_1\gamma_1} + \cos(p\gamma_2) + e^{p_1\gamma_1}$$

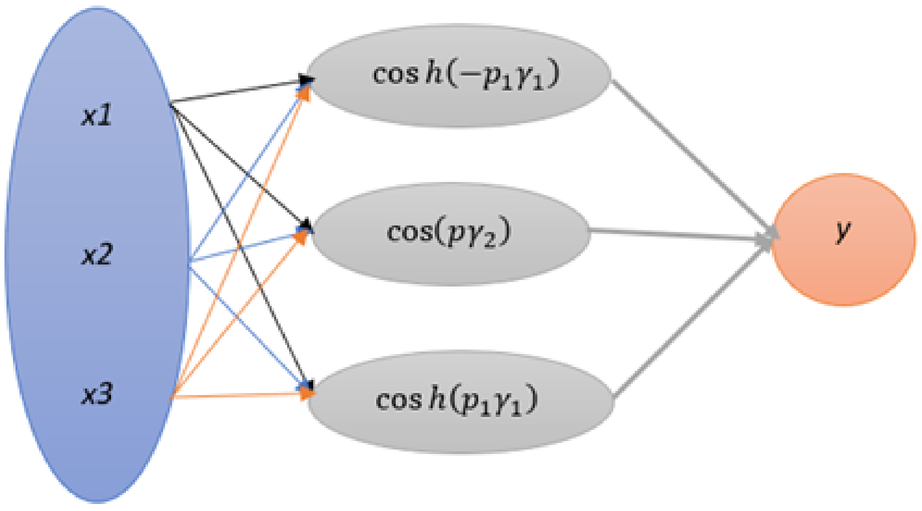


Figure 12

In all the above classical test functions, $w_{j,k} = 1$ and $l = 1$ as it is a one-layer model. Now, test operations can be created in bilinear methods too by giving specific weights, thresholds, and functions [16].

Zhang and Bilige initially presented BNNM for nonlinear partial differential equations [15] in 2019. Then, in the year 2020, Zhang, Bilige, and Chaolu [17] applied the BNNM to the p-gBKP equation, where they constructed 19 exact function solutions. In 2023–2024, Yuanlin Liu et al. [16] used the BNNM with a single layer to derive lump, breather, and interaction solutions for (3+1)-dimensional equations, emphasizing flexibility via threshold choices. This was followed by Runfa Zhang et al., who proposed an improved BNNM to showcase M-lump and lump–breather wave solutions in 2024. After this, Xia, Zhang, Luo, and many other researchers worked on finding exact solutions by using specific single- or multi-layered BNNMs.

5 Conclusions

The present work is concerned with the design and use of multilayer perceptron-based neural networks specifically for solving nonlinear partial differential equations, in the Bilinear Neural Network Model (BNNM) framework. It began with understanding concepts in neural network design, forward and backward propagation, and perceptron structure. We further explored their applicability to solve mathematical models such as the p-gBKP equation in terms of tensor representation and bilinear Hirota operators. Encoding various well-known exact solutions (such as lump, breather, rational, periodic, and interaction waveforms) inside neural networks and adjusting values of activation and threshold functions, weights, and biases, it is possible to obtain many known exact solutions. This indicates that neural networks are not just computational machines but symbolic function approximators for mathematical systems. The results support the conclusion that BNNMs are a powerful and flexible tool for solving nonlinear PDEs and have great prospects in symbolic computation and scientific model research.

Declarations

Ethics approval and consent to participate

Not applicable.

Conflict of interest

The author claims that there are no conflicts of interest.

Data availability statement

No datasets have been generated or analyzed during the current investigation.

References

[1] Simplilearn, *Director, Simplilearn. [Film]. Online: Neural Network In 5 Minutes — What Is A Neural Network? — How Neural Networks Work*, Aug. 21, 2019.

[2] M.E.a.Y.Ghanou, *"Neural Architectures Optimization and Genetic Algorithms," WSEAS Transactions On Computer*, vol. 8, no. 3, 2009.

[3] Rosenblatt, *"The Perceptron: A Theory of Statistical Separability in," Cornell Aeronautical Laboratory*, Jan. 1958.

[4] F. Rosenblatt, *"The Perceptron: A probabilistic model for information storage and organization in the brain," Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[5] M. Minsky, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, 1969.

[6] D. H. G. & W. R. Rumelhart, *Learning representations by back-propagating errors*, Nature, vol. 323, no. 6088, pp. 533–536, 1968.

[7] L. & L. S. Zhang, *Exact solutions of the generalized bilinear KP equation via neural network models*, Chaos, Solitons & Fractals, p. 134, 2020.

[8] D. E. H. G. E. & W. R. J. Rumelhart, *Learning representations by back-propagating errors*, Nature, vol. 323, no. 6088, pp. 533–536, 1986.

[9] A. S. I. & H. G. E. Krizhevsky, *ImageNet classification with deep convolutional neural networks*, Advances in Neural Information Processing Systems, 2012.

[10] J. L. Elman, *Finding structure in time*, Cognitive Science, vol. 14(2), pp. 179–211, 1990.

[11] S. & S. J. Hochreiter, *Long short-term memory*, Neural Computation, vol. 9(8), pp. 1735–1780, 1997.

[12] K. et al., *Learning phrase representations using RNN encoder–decoder for statistical machine translation*, EMNLP, 2014.

[13] G. E. & S. R. R. Hinton, *Reducing the dimensionality of data with neural networks*, Science, vol. 313, no. 5786, pp. 504–507, 2006.

[14] I. et al., *Generative adversarial nets*, NeurIPS, 2014.

[15] S. B. R. F. Zhang, *Bilinear neural network method to obtain the exact analytical solutions of nonlinear partial differential equations and its application to pgBKP equation*, 2019.

[16] M.-C. L., M. A., F.-C. Z., Run-Fa Zhang, *Generalized lump solutions, classical lump solutions and rogue waves of the (2+1)-dimensional Caudrey–Dodd–Gibbon–Kotera–Sawada-like equation*, Applied Mathematics and Computation, vol. 403, no. 126201, 2021.

[17] S. B. & C. T. Run Fa Zhang, *Fractal Solitons, Arbitrary Function Solutions, Exact Periodic Wave and Breathers for a Nonlinear Partial Differential Equation by Using Bilinear Neural Network Method*, Journal of Systems Science and Complexity, vol. 34, pp. 122–139, 2021.

[18] M. E. & Y. Ghanou, *Neural architectures optimization and Genetic algorithms*, WSEAS Transactions On Computer, Issue 3, Volume 8, 2009.

[19] R.-F. Z. & S. Bilige, *Bilinear neural network method to obtain the exact analytical solutions of nonlinear partial differential equations and its application to p-gBKP equation*, Nonlinear Dynamics, vol. 95, no. 1, pp. 3041–3048, Jan. 2019.

[20] W. Q. Z. & L. X. Ma, *Lump solutions to dimensionally reduced p-gBKP equations*, Nonlinear Dynamics, vol. 4, pp. 923–931, 2016.

[21] R. B., S. B., Y. L., J. G., X. Zhang, *Interaction phenomenon to dimensionally reduced p-gBKP equation*, Mod. Phys. Lett. B, vol. 32(6), p. 1850074, 2018.

[22] Z. Y. Ma, W. X., *Exact solutions via bilinear neural network models for nonlinear evolution equations*, Applied Mathematics and Computation, 2021.

[23] Z. Y. Ma, W. X., *Bilinear neural network method for a generalized (3+1)-dimensional p-gBKP equation*, Communications in Theoretical Physics, 2018.