

Article

Orthogonal Neural Network: An Analytical Model for Deep Learning

Yonghao Pan, Hongtao Yu *, Shaomei Li and Ruiyang Huang

Information Technology Institute, Information Engineering University, Zhengzhou 450001, China; panyonghao2016@163.com (Y.P.)

* Correspondence: 15937101921@139.com

Abstract: In the current deep learning model, the computation between each feature and parameter is defined in the real number field. This, together with the nonlinearity of the deep learning model, makes it difficult to analyze the relationship between the values of the computational process and the original features from computation in the real number field. We extend the operational rules of the deep learning model in space and propose the orthogonal neural network (ONN) model, in which the features are set orthogonally to each other in space by “modulating” each input feature of the deep learning model to different orthogonal bases. Because the modulated numerical features are orthogonal to each other, they can be separated from the computations of the ONN model. By “demodulating” the model during and after the calculation, we can obtain a numerical relationship between the results and the original features, which can further provide theoretical and computational support for our analysis of the model. Finally, we compute the weights for each input feature as an interpretable deep learning approach, and describe how the model focuses attention on each feature based on the application of the orthogonal neural network model on two typical models: convolutional neural networks and graph neural networks.

Keywords: deep learning; convolutional neural network; graph neural network



Citation: Pan, Y.; Yu, H.; Li, S.; Huang, R. Orthogonal Neural Network: An Analytical Model for Deep Learning. *Appl. Sci.* **2024**, *14*, 1532. <https://doi.org/10.3390/app14041532>

Academic Editor: Luis Javier Garcia Villalba

Received: 4 December 2023

Revised: 2 February 2024

Accepted: 14 February 2024

Published: 14 February 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Artificial intelligence technology has been used to achieve unquestionable success in various fields, bringing a new way of thinking to human science, technology, and productive life. As one of the most crucial technical methods of artificial intelligence, deep learning is the direction of research experiencing the most attention in academia and industry today. Deep learning, with its end-to-end learning approach, is capable of efficient automatic feature extraction and has experienced substantial improvements in accuracy, triggering yet another peak in AI development.

However, deep learning models are characterized by complex models, a vast number of parameters, and nonlinearities, somewhat resembling a black box, and meaningful insight into their inner working mechanisms is lacking. Deep neural networks have been shown to have low robustness, e.g., in studies of adversarial attacks on convolutional neural networks [1–6] and graph neural networks [7–10]. As a result, deep neural networks are limited in their use in some areas where security is a concern. In certain research advances in explainable AI [11], some extremely clever research perspectives on deep learning models have been proposed for analysis. For example, the CAM [12] algorithm can be used to locate the main regions of interest in image classification models. The LIME [13] algorithm uses the idea of local linearization to analyze linearization within the neighborhood of results.

In this paper, we propose an extended model of the deep neural networks that can analyze the results from each step of computation in the deep neural networks and decompose them into the parts corresponding to each input feature.

Analysis is a primary method for studying a problem. The analysis of the object as a whole is formed by decomposing the object of study into its parts and synthesizing them

after studying each part separately. Therefore, how to break down the research object from the whole into its parts is an important starting point. It is envisaged that a microscopic perspective can be chosen as a starting point, where a single input numerical feature of a deep learning model—for example, a pixel in image data or a single feature on a node in graph data—serves as the unit of analysis. Related studies in the literature have not usually been conducted from the perspective of individual input features. There are two main reasons for this. First, the number of features and model parameters is often huge, and it is difficult to intuitively focus the research perspective on a single input feature. Furthermore, the basic idea of deep learning is to train the model in a large amount of feature data to obtain the model parameters with the minimum loss, and in this perspective specifically focusing on individual features is of little significance.

In human exploration of the laws of nature, many superposition phenomena have been discovered: for example, waves in the water and the superposition of different light colors. By studying the decomposition of waves based on wavelength, it is possible to determine the pattern of wave motion. Research on the decomposition of light based on wavelength is precisely spectral theory. In Fourier's classic work on heat conduction, *The Analytical Theory of Heat* [14], the problem of triangular series expansion is studied and discussed. After continuous refinement by mathematicians, the Fourier series, as it has come to be known, was further developed and refined in the study of signal analysis and processing as the Fourier transform. Since trigonometric functions have a clear physical meaning in the study of signals, expanding the signal into a Fourier series representation is equivalent to decomposing the signal into different frequencies for study. It is also inferred that the signal is decomposed into frequency space. The information loaded on each frequency is irrelevant, because the trigonometric function representing frequency is orthogonal. Likewise, the good properties of this orthogonality of frequencies have inspired the progress in communication technology and the development of frequency division multiplexing. The information can be modulated to different frequencies in the same transmission channel to achieve multiplexed information transmission. Inspired by this theory, we consider whether we can study the deep learning model as a communication system, treating each input feature as a signal, and investigate the deep learning model by analyzing the relationship between the signal and the communication system.

Based on the above reflection, we propose an analytical model for deep learning named orthogonal neural network model in this paper. The contributions are listed as follows.

- The orthogonal neural network model we propose is an interpretable deep learning approach.
- Modulation and demodulation are introduced in communication theory to extend the definition of the deep learning model into frequency space. The property of orthogonality between frequencies is used to ensure that the individual initial features of the input orthogonal neural network model are both independent of each other and correspond to meaning in the deep learning model.
- The model proposed in this paper can be used to analyze a type of relationship between the computed results of a deep learning model and the initial features in the form of weights based on an interpretable deep learning approach. From the size of the weights, it is possible to determine the primary and secondary features on which the deep learning model will focus.

An additional note is that the most used activation function in current deep learning models is the ReLU [15] function, which is in Equation (1). In this paper, discussion of the activation functions of deep learning models is restricted to the ReLU function.

$$ReLU(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \quad (1)$$

2. Orthogonal Neural Network Model

From the perspective of each numerical feature of the sample input into the deep learning model, the operations of the numerical features are defined in the real number field.

This makes it impossible to separate them from the numbers in subsequent calculations of the deep learning model. We propose the orthogonal neural network (ONN) model as an extension of the deep learning model. This ONN model modulates numerical features of the sample onto orthogonal bases and extends the calculation rules corresponding to the deep learning model in the space constructed by the orthogonal bases. Then, the ONN model analyzes the weights corresponding to the input numerical features by demodulating values on the orthogonal bases. The model establishes a relationship between input features and results, quantifying this relationship in the form of weights.

2.1. Modulation and Demodulation on Orthogonal Bases

To simplify the description of the problem, we introduce here for discussion a function set composed of sine functions on $t \in [0, 2\pi]$ in Equation (2). The “ n ” in Equation (2) is the number of input features.

$$\{\sin(t), \sin(2t), \dots, \sin(nt)\} \quad (2)$$

The sine functions in this function set are orthogonally, as in Equation (3).

$$\frac{1}{\pi} \int_0^{2\pi} \sin(nt) \sin(mt) dt = \begin{cases} 1, & n = m \geq 1 \\ 0, & \text{other} \end{cases} \quad (3)$$

For a “modulation” process, modulating x_1 to a sine wave of frequency n can be expressed as Equation (4).

$$x_1 \rightarrow x_1 \sin(nt) \quad (4)$$

The amplitude modulation of the sine wave with a value x_1 causes the value x_1 to be loaded onto the sine wave, which is the amplitude information after modulation. “Demodulation” is the inverse process of “modulation”. For a “demodulation” process, demodulating x_1 from a sine wave of frequency n can be expressed as (5).

$$x_1 \sin(nt) \xrightarrow{\frac{1}{\pi} \int_0^{2\pi} x_1 \sin(nt) \sin(nt) dt} x_1 \quad (5)$$

Here, modulation and demodulation are entirely defined by computation to separately modulate all values at the input of a deep learning model to different frequencies so that these values can exist independently during the computation. Such a process is implemented in a communication system using a circuit whose corresponding mathematical description differs from this one. However, the computational result is the same; here, it is fully implemented by computation.

For a value $x_i \in \mathbb{R}$, given as input to the deep learning model, the operations involved are mainly multiplication and addition, i.e., linear operations. In standard models, the computational units that generate nonlinearity are mainly the activation function and maximum pooling, which are characterized by selection of the linear computation results in other links. Although understood as nonlinear in terms of computational properties, the output of their operations is still the result of linear computation of the values of model input. The values input to the deep learning model are denoted as $\{x_i \mid i \in \mathbb{N}, 1 \leq i \leq N\}$, for each computational session, the values in the result can be expressed as $\sum_{i=1}^N w_i x_i$, w_i is the weight corresponding to x_i , and $w_i = 0$ if the computational result does not contain the input x_i .

Therefore, by assigning a frequency to each value input in the deep learning model and modulating it according to specific rules, the values generated by each computational link in the deep learning model become a linear superposition of a set of sine functions, which can be expressed as $\sum_{i=1}^N w_i x_i \sin(it)$. When it is necessary to extract an x_j of interest, “demodulation” can be computationally implemented, such as in Equation (6).

$$\frac{1}{\pi} \int_0^{2\pi} \sum_{i=1, i \leq j \leq N}^N w_i x_i \sin(it) \sin(jt) dt = w_j x_j \quad (6)$$

The conclusion from calculating the orthogonality of sinusoidal functions is that the summation is nonzero only when $i = j$, which means that x_j can be demodulated only by using $\sin(jt)$. Through the above modulation and demodulation computation process, each input value to the deep learning model is actually assigned a “frequency” label, which can be used to extract the value corresponding to the initial input value at any time during model operation, and this label has a clear physical meaning and is easy to compute.

2.2. Operational Rules of ONN Model

Application of frequency modulation to the values input into the deep learning model is equivalent to converting the input values into a function, so it is necessary to extend the computational rules of the deep learning model such that it can satisfy both the operation of the function and the computational logic in being consistent with the original numerical computation. In the following discussion, extension of the operation rules is defined for each calculation unit.

First, we discuss the case of a single neuron, which consists of an affine function and an activation function. For n numeric features, $\mathbf{x} = \{x_1, x_2, \dots, x_n\}$, the weights corresponding to the features are $\mathbf{w} = \{w_1, w_2, \dots, w_n\}$, and b is the bias and can be defaulted. Thus, the affine function is $a = \sum_{i=1}^n w_i x_i + b = \mathbf{w}\mathbf{x}^T + b$. Then, over the activation function, it constitutes a single neuron, which can be expressed as (7). As explained at the beginning of the article, the activation function is limited to the ReLU function.

$$\text{ReLU}(a) = \begin{cases} a & \text{if } a \geq 0 \\ 0 & \text{if } a < 0 \end{cases} \quad (7)$$

We perform frequency modulation for the numerical elements in $\mathbf{x}$. For the sake of discussion, the frequency corresponding to each element is the subscript of that element. $\mathbf{x}_{mod} = \{x_i \sin(it) \mid x_i \in \mathbf{x}\}$, and the frequency assigned for the bias b assignment is $n + 1$ such that the affine function becomes $a_{mod} = \sum_{i=1}^n w_i x_i \sin(it) + b \sin((n + 1)t) = \mathbf{w}\mathbf{x}_{mod}^T + b \sin((n + 1)t)$. It can be seen that after modulation of the numerical elements, the calculation of the affine function changes from a numerical value to a function containing a variable t , which is only useful in the demodulation session and, therefore, is not additionally written. Since the calculation result of the affine function changes from a numerical value to a function, it is necessary to extend the definition of the operation rules for the activation function, which is in Equation (8).

$$\text{ReLU}_{mod}(a_{mod}) = \begin{cases} a_{mod} & \text{if } a \geq 0 \\ 0 & \text{if } a < 0 \end{cases} \quad (8)$$

The judgment condition is the same as the original activation function. Due to the premise of frequency modulation, we can obtain a by demodulating the values in a_{mod} by Equation (9).

$$a = \sum_{i=1}^{n+1} \int_0^{2\pi} a_{mod} \sin(it) dt \quad (9)$$

Here, we can figuratively understand the extended definition of the activation function as a switch function, and the affine function calculates a_{mod} as a signal. When $a > 0$, the switch opens the signal through; when $a \leq 0$, the switch closes the signal blocking. There are differences in the definition of the activation function for the judgment condition when $a = 0$. In the original activation function, the $a = 0$ condition can be included in either the greater- or less-than conditions. In the extended definition of the activation function, the $a = 0$ condition can only be included in the less-than condition because consistency with the original definition must be guaranteed in the numerical computation logic of

the extended definition. In the original definition of the condition $a = 0$, the activation function outputs $a = 0$. In the extended definition, if the equal condition is included in the greater-than condition, the activation function outputs a_{mod} , which is inconsistent with the computational logic of the original activation function.

When the neuron is not in the input layer, each output of the previous layer is a linear combination of sine functions, which can be expressed as $\sum_i c_i \sin(it)$, and the modulated $\mathbf{x}_{mod} = \{x_i \sin(it) \mid x_i \in \mathbf{x}\}$ is not essentially different, but is a linear combination of sine functions, so it still satisfies the above discussion of affine functions and activation functions. The output of each computational unit is a linear combination of sine functions.

With the above discussion on the extension of the activation function operation rules, we initially show the extension method. In order to further provide a normatively rigorous mathematical representation of the arithmetic process, we define a function set F .

$$F = \left\{ \sum_i c_i \sin(it), c_i \in \mathbb{R}, i \in \mathbb{N}, i \in [1, n], t \in [0, 2\pi] \right\} \quad (10)$$

The function set F consists of linear combinations of sine functions with integer frequencies, where F satisfies $\forall a, b \in \mathbb{R}, f_1, f_2 \in F, af_1 + bf_2 \in F$. Thus, the function set F is complete. For the convenience of discussion, we define a size measure for $f \in F$. Here, $f \in F$ represents the linear superposition of a set of sine functions, corresponding to the deep learning computation, where the information modulated on the corresponding sine functions needs to be demodulated and summed. Combining the definition of the function set F , we define the sum of the coefficients of the sine functions $S(f)$ for $f \in F$ in Equation (11).

$$S(f) = \sum_{i=1}^n c_i = \frac{1}{\pi} \sum_{i=1}^n \int_0^{2\pi} f \cdot \sin(it) dt \quad (11)$$

Here, we further provide a more normative definition of the ReLU activation function in Equation (12). The $x_i \sin(it)$ and $\sum_i c_i \sin(it)$ in the previous discussion belong to the function set F .

$$ReLU(f) = \begin{cases} f & \text{if } S(f) > 0 \\ 0 & \text{if } S(f) \leq 0 \end{cases} \quad (12)$$

For a single-layer neural network containing n neurons, the input layer contains m features. The calculation rule is output $Y = ReLU(X \cdot W)$, where $X \in F^{1 \times m}$ is a function matrix denoting the input layer of the neural network, $W \in \mathbb{R}^{m \times n}$ is a numerical matrix denoting the parameters of that layer of the neural network, $Y = F^{1 \times n}$ is a function matrix denoting the output layer of the neural network, which is in Equation (13).

$$Y_{1,j} = \sum_{i=1}^m X_{1,i} \cdot W_{i,j} \quad (13)$$

Due to $X_{1,i} \in F$, $W_{i,j} \in \mathbb{R}$, obviously $X_{1,i} \cdot W_{i,j} \in F$. For the calculation rules of multilayer neural networks, one layer contains n neurons. The output of the previous layer has m neurons, so the input of the layer is the output of the previous layer as $X \in F^{1 \times m}$ and $W \in \mathbb{R}$ is the parameter matrix of the neural network of the current layer. The output is $Y \in F^{1 \times n}$, and $Y = ReLU(X \cdot W)$ is consistent with the above. Multilayer neural networks [16] can be constructed according to such rules.

Multilayer neurons are logically composed as a linear computational combination of multiple neurons. It should be emphasized here that their composition is characterized as a linear combination but, as a whole, does not satisfy the property of linearity. Since the activation function in a single neuron is nonlinear, the condition of linearity is not satisfied. Therefore, the combination of linear superposition of neurons also does not satisfy the conditions for linearity, but its calculation results in a linear combination of the input features. This phenomenon occurs because the ReLU function is a piecewise linear

function and can therefore guarantee the properties of partial linear calculation under piecewise conditions.

In typical deep learning models, multiple layers of neurons are generally used as fully connected layers in combination with other calculation units to build models capable of achieving complex learning functions. In convolutional neural networks [17], the basic calculation units are convolution, activation function, pooling, and multilayer neurons. Next, we discuss the convolutional neural network used in computer vision as an example.

Convolution is one of the operations for extracting features in a convolutional neural network. For a two-dimensional numerical matrix $X \in \mathbb{R}^{U \times W}$, the convolution kernel is $G \in \mathbb{R}^{k \times k}$, with k being an odd number, and the convolution is computed as $H = X * G$, which is in Equation (14), where $H \in \mathbb{R}^{M \times N}$.

$$H_{m,n} = \sum_{i=1}^k \sum_{j=1}^k X_{m-\frac{k+1}{2}+i, n-\frac{k+1}{2}+j} \cdot G_{i,j} \quad (14)$$

Expansion of the convolution operation involves transforming the input numerical matrix $X \in \mathbb{R}^{U \times W}$, replaced by the function matrix $X \in F^{U \times W}$, then the output function matrix $H \in F^{M \times N}$. The calculation formula is still written as $H = X * G$. From the previous discussion, the formula is obviously compatible with two operations.

The graph neural network [18–20] is a model recently proposed in deep learning that is mainly composed of graph convolutions. For a graph with M nodes, each containing N features, the structural information of the graph is the adjacency matrix $A \in \mathbb{R}^{M \times M}$, and the feature information is the matrix $X \in \mathbb{R}^{M \times N}$. Each graph convolution layer comprises a fully connected layer of K neurons with a parameter matrix of $W \in \mathbb{R}^{N \times K}$. The graph convolution layer is defined as $Y = \text{ReLU}(LXW)$, where the Laplace matrix $L \in \mathbb{R}^{M \times M}$ is calculated from the adjacency matrix A and the output $Y \in \mathbb{R}^{M \times K}$. When multiple graph convolution layers are combined, the matrix Y output from the previous layer is used as the feature matrix X as input for this layer.

The extension of the operational rules mainly involves converting the numerical matrix $X \in \mathbb{R}^{M \times N}$, which represents the characteristic information, into a function matrix $X \in F^{M \times N}$. Since the activation functions have been discussed, the main discussion here is on the function matrix $X \in F^{M \times N}$ left multiplication numerical matrix $L \in \mathbb{R}^{M \times M}$, which is in Equation (15), and the function matrix $X \in F^{M \times N}$ right multiplication parameter matrix $W \in \mathbb{R}^{N \times K}$, which is in Equation (16). Here, the matrix multiplication rules correspond to the original model for the inner product multiplication operation of the matrix.

$$Y_{i,j}^1 = L_{i,:} \cdot X_{:,j} = \sum_{l=1}^M L_{i,l} \cdot X_{l,j} \quad (15)$$

$$Y_{i,j}^2 = X_{i,:} \cdot W_{:,j} = \sum_{l=1}^N X_{i,l} \cdot W_{l,j} \quad (16)$$

Obviously, $Y_{i,j}^1, Y_{i,j}^2 \in F$, and the extended graph convolution layer is $Y = \text{ReLU}(LXW)$, where $Y \in F^{M \times K}$ is a function matrix. The operation here also satisfies the associative law, which is $(LX)W = L(XW)$.

2.3. Further Optimization for Orthogonal Bases

In the previous discussion, inspired by the frequency division multiplexing technology in communication technology, we introduced the concepts of “frequency”, “modulation”, and “demodulation” and extended the operation rules of the deep learning model. This discussion revolves around the physical meaning of sine function and frequency. From a mathematical point of view, the sine function is introduced to form a function set and the elements in this function set are orthogonal to each other on the interval $[0, 2\pi]$. The sine function on the interval $[0, 2\pi]$ is taken as a set of orthogonal bases to form a space

and further extend the corresponding definition of the operation rules of the deep learning model on this space.

However, what needs to be further discussed is that from the perspective of concrete calculation in the computer, the continuous function in an interval is a sampled discrete sequence on the computer. When the sampling interval is T in the interval $[0, 2\pi]$, there are $K = \frac{2\pi}{T}$ sampling points, so the sequence sampled by $\sin(nkT)$ is $\{\sin(nkT), 0 \leq k \leq K\}$, and the sampled sequence has the same orthogonal property, which is in Equation (17).

$$\frac{1}{\pi} \sum_{k=0}^K \sin(nkT) \sin(mkT) = \begin{cases} 1, & \text{if } n = m \geq 0 \\ 0, & \text{if } \text{other} \end{cases} \quad (17)$$

Therefore, when performing operations on the computer, we actually use a set of orthogonal sequences after sampling the sine function, which is in Equation (18).

$$\left\{ \{\sin(kT)\}_{k=0}^K, \{\sin(2kT)\}_{k=0}^K, \dots, \{\sin(nkT)\}_{k=0}^K \right\} \quad (18)$$

When n features need to be modulated, n frequencies need to be allocated and generated. In order to ensure that the sine functions of these n frequencies do not produce frequency aliasing, when the maximum frequency is n , it is necessary to satisfy the Nyquist sampling theorem when discretizing, which in this case means that a sequence of at least $2n$ points is needed to represent the sine functions on the interval. In the scenario of deep learning, this n is always enormous. When allocating $2n$ values of storage space for each frequency, $2n^2$ floating point values need to be allocated to memory, which will cause considerable resource consumption. At the same time, there will also be quantization errors in the process of discrete sampling of continuous functions. After entering the deep learning algorithm, the cumulative errors resulting from these quantization errors will affect its accuracy.

Therefore, the space constructed by the sine function on the interval $[0, 2\pi]$ as the orthogonal basis has clear physical significance, but it is challenging to realize in practical applications. Further, consider constructing a set of orthogonal bases from a mathematical perspective and the equivalent, realizing the theoretical process discussed in the previous article. The most common orthogonal basis in mathematics is the standard orthogonal basis of Euclidean space, which can be expressed as an orthogonal vector set $\mathcal{A} = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$ and (19). The “ n ” is the number of input features which is consistent with Equation (2).

$$\begin{aligned} \alpha_1 &: [1, 0, 0, \dots, 0] \\ \alpha_2 &: [0, 1, 0, \dots, 0] \\ &\vdots \\ \alpha_n &: \underbrace{[0, 0, 0, 1, \dots]}_{n-1} \end{aligned} \quad (19)$$

The use of the orthogonal basis is not only simple and elegant in form, but can also allow directly avoiding the quantization error generated by digital sampling of the sine function when the sine function is used as the orthogonal basis. The position index of the nonzero item in the base vector also completely corresponds to the base number. Using sparse data structures for storage can significantly save the storage space and improve the operation efficiency. Another point is that as a standard orthogonal basis, its modulus is 1, and the nonzero term value is also 1, which is convenient for the calculation.

Corresponding to function set F , define vector set E .

$$E = \left\{ \sum_i c_i \alpha_i, c_i \in \mathbb{R}, i \in \mathbb{N}, i \in [1, n], \alpha_i \in \mathcal{A} \right\} \quad (20)$$

Evidently, E satisfies $\forall a, b \in \mathbb{R}, \mathbf{e}_1, \mathbf{e}_2 \in E, a\mathbf{e}_1 + b\mathbf{e}_2 \in E$, with completeness. When discussing function set F , the sum function $S(f) = \sum_i c_i$ is defined for $f \in F$. For vector set E , the corresponding operation $S(\mathbf{e}) = \sum_i c_i$ for $\mathbf{e} \in E$. And, $S(\mathbf{e})$ is exactly the sum of

elements of vector $\mathbf{e} \in E$. This is also the advantage of using the standard orthogonal basis of Euclidean space.

Therefore, the extended operation rules defined on the function set F discussed in the previous section can be directly replaced by the vector set E , which is equivalent. The corresponding calculation logic has been fully discussed for (10), (12)–(16), and (20), and there will be no further detailed discussion here.

2.4. Summary of the Model

In the previous discussion, the features input into the depth learning model are loaded onto different bases by introducing a set of orthogonal bases, and the orthogonal relationship between the bases is used to ensure the input features remain independent of each other. Further, according to the numerical computation relationship of each feature in the original model, the corresponding operation rules are defined in the space composed of this set of orthogonal bases, which is the extension discussed for the deep learning model. From the perspective of numerical calculation, the two are entirely consistent, but the latter can separate the features on each based on the orthogonal relationship between the bases.

Most of the computational units of the deep learning model are linear, so the computation between features and model parameters is actually a linear operation on the real number field. In the expanded deep learning model, the features exist as vectors in space, and their operation with the model parameters becomes a linear operation between the vectors in space.

Regarding the two typical nonlinear operation units, activation function and maximum pooling, the operation rules in vector set E can still be expanded and defined through the analysis of their operation logic and the use of their “partially linear” nature; that is, the vectors involved in the operation can be linearly expressed with an orthogonal basis. Therefore, we can extend the deep learning model to operate in the vector set E .

It should be emphasized again that the operation here maintains the property that the output results of the computational units can all be expressed as linear combinations of orthogonal bases, but is not a linear system and does not satisfy the definition of a linear system. On the other hand, if the system is linear, the deep learning model does not have the problem of insufficient interpretability, and the linear system can directly solve the system function using impulse response methods.

The most important feature of the discussion in this paper is the introduction of the orthogonal function set and orthogonal vector set to extend the deep learning model with operational rules and introduce the concept of modulation and demodulation, so the extended model is called **orthogonal neural network (ONN)** in this paper. The process of loading features onto an orthogonal basis is expressed as orthogonal modulation of features. The definition of the model is shown in Figure 1, which includes the modulated data, the parameters of the deep learning model, and the algorithmic rules defined in the extension in this section. Then, we can compute the analysis data of the classification results based on the orthogonal neural network model by demodulating values on the orthogonal bases. The ONN model can be applied to almost any deep learning model that uses the ReLU activation function. In the following, we detail the applications of the ONN model on two typical deep learning models: convolutional neural network and graph neural network.

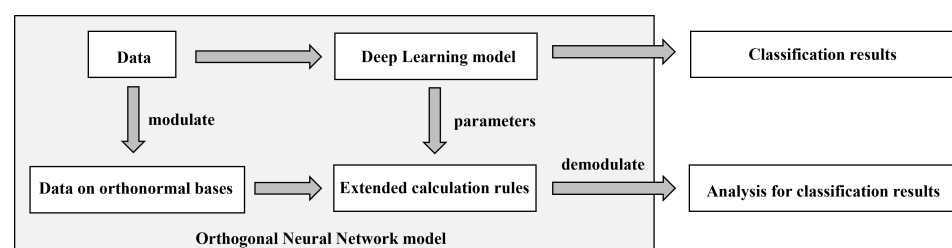


Figure 1. Orthogonal neural network model.

3. ONN Model Applied on Convolutional Neural Networks

In this section, we provide an example of the ONN model applied on convolutional neural network to analyze the weights corresponding to the input numerical features.

3.1. A Simple Convolutional Neural Network as An Example

We construct a simple convolutional neural network for the application of image classification in computer vision. We choose the Fashion-MNIST [21] dataset and construct a convolutional neural network consisting of two convolutional layers, two pooling layers, and three fully connected layers.

Input layer: $1 \times 28 \times 28$, stored as a matrix of $\mathbb{R}^{1 \times 28 \times 28}$. Because it is a grayscale image, the first dimension is 1.

Layer 1: convolution layer, convolution kernel size is 5×5 , the output is six feature maps, sliding step $stride = 1$, so the convolution kernel is an $\mathbb{R}^{6 \times 1 \times 5 \times 5}$ matrix and the output is an $\mathbb{R}^{6 \times 24 \times 24}$ matrix, and activation is by the ReLU activation function.

Layer 2: pooling layer, uses maximum pooling, the pooling window size is 2×2 , sliding step $stride = 2$, and the output is an $\mathbb{R}^{6 \times 12 \times 12}$ matrix.

Layer 3: convolution layer, convolution kernel size is 5×5 , the output is 12 feature maps, sliding step $stride = 1$, so the convolution kernel is an $\mathbb{R}^{12 \times 6 \times 5 \times 5}$ matrix and the output is an $\mathbb{R}^{12 \times 8 \times 8}$ matrix, and activation is by the ReLU activation function.

Layer 4: pooling layer, uses maximum pooling, the pooling window size is 2×2 , sliding step $stride = 2$, and output is an $\mathbb{R}^{12 \times 4 \times 4}$ matrix.

The last three layers are fully connected layers, spreading the $\mathbb{R}^{12 \times 4 \times 4}$ output of the fourth layer into a vector of $\mathbb{R}^{192}$. Each layer consists of 120, 60, and 10 neurons. The ReLU activation function is applied to the output of the first two layers. The first layer neuron output is a vector of $\mathbb{R}^{120}$, the second layer neuron output is a vector of $\mathbb{R}^{60}$, and the third layer output is a vector of $\mathbb{R}^{10}$, which is the basis for the final classification discrimination, and the index of the largest value in the vector corresponds to the classification, i.e., the classification of the image with the input layer of $1 \times 28 \times 28$.

3.2. Extended CNN in ONN Model

In orthogonal modulation of the input layer, a vector $\alpha \in \mathcal{A}$ is assigned to each pixel according to a certain rule. The grayscale image in this example is actually a two-dimensional matrix of $\mathbb{R}^{28 \times 28}$, so a total of $28 \times 28 = 784$ vectors $\alpha \in \mathcal{A}$ need to be assigned one by one for correspondence to the image pixels, and for the pixel point $x_{i,j}$, the vector assigned is $\alpha_{i \times 28 + j} \in \mathcal{A}$. Thus, the input layer after modulation is an $E^{1 \times 28 \times 28}$ matrix. The details of the extended CNN in the ONN model are as follows.

Input layer: $1 \times 28 \times 28$, stored as a matrix of $E^{1 \times 28 \times 28}$.

Layer 1: convolution layer, convolution kernel size is 5×5 , the output is six feature maps, sliding step $stride = 1$, so the convolution kernel is an $E^{6 \times 1 \times 5 \times 5}$ matrix and the output is an $E^{6 \times 24 \times 24}$ matrix, and activation is by the ReLU activation function.

Layer 2: pooling layer, using maximum pooling, the pooling window size is 2×2 , sliding step $stride = 2$, and the output is an $E^{6 \times 12 \times 12}$ matrix.

Layer 3: convolution layer, convolution kernel size is 5×5 , the output is 12 feature maps, sliding step $stride = 1$, so the convolution kernel is an $E^{12 \times 6 \times 5 \times 5}$ matrix and the output is an $E^{12 \times 8 \times 8}$ matrix, and activation is by the ReLU activation function.

Layer 4: pooling layer, using maximum pooling, the pooling window size is 2×2 , sliding step $stride = 2$, and output is an $E^{12 \times 4 \times 4}$ matrix.

The last three layers are fully connected layers, spreading the $E^{12 \times 4 \times 4}$ output of the fourth layer into a vector of E^{192} . Each layer consists of 120, 60, and 10 neurons. The ReLU activation function is applied to the output of the first two layers. The first layer neuron output is a vector of E^{120} , the second layer neuron output is a vector of E^{60} , and the third layer output is a vector of E^{10} , which is the basis for the final classification discriminations and the index of the largest value in the vector corresponds to the classification, i.e., the classification of the image with the input layer of $1 \times 28 \times 28$.

3.3. Discussion

The features enter the first four layers from the input layer and are computed according to the orthogonal neural network operation rules based on the extended operation rules. The output of the fourth layer is an $E^{12 \times 4 \times 4}$ matrix, which is spread to an E^{192} matrix and then enters the fully connected network, and the final output is an E^{10} matrix. This is further expressed as $E^{10} = \{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_{10}\}$, so $\{S(\mathbf{e}_1), S(\mathbf{e}_2), \dots, S(\mathbf{e}_{10})\}$, which corresponds to the output of the original network model. However, each $\mathbf{e} \in E$ in the computation process contains a lot of useful information, based on which a wealth of analysis can be performed on the convolutional neural network model. Since $\mathbf{e} \in \{\sum_i c_i \alpha_i, c_i \in \mathbb{R}, i \in \mathbb{N}, \alpha_i \in \mathcal{A}\}$, in this case, $i \in \{i \in \mathbb{N}, 1 \leq i \leq 784\}$, a corresponding α_i is assigned to each pixel point in the modulation stage of the input layer, and the value c_i on α_i in the computation process is the result from computation of the pixel point corresponding to the input layer in the convolutional neural network. All matrices with elements of $\mathbf{e} \in E$ generated in the computation process of the convolutional neural network contain a linear combination of its relationship with the value of each pixel in the input layer.

Figure 2a is an original image in the dataset for an Ankle Boot, corresponding to an Ankle Boot label of 10. This is equivalent to the result of calculating $\{S(\mathbf{e}_1), S(\mathbf{e}_2), \dots, S(\mathbf{e}_{10})\}$ in the vector matrix whose output is $E^{10} = \{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_{10}\}$, so $S(\mathbf{e}_{10})$ is the expected result from calculation of the maximum. The image is input into the orthogonal neural network, where the network parameters are the same as for the original convolution neural network, and the correct classification expectation is obtained, that is, maximum of $S(\mathbf{e}_{10})$. Further, we put $\mathbf{e}_{10}$ through the correspondence between the orthogonal vectors and the initial features to recover the position of each component of the result in the original image, as shown in Figure 2b, where blue is the component with negative values and red is the component with positive values, and the darker the color, the larger the value. It can be observed from the figure that the initial pixel points have positive and negative effects on the calculation. For convenient observation, the figure uses a gray background to describe the outline of the original image content.

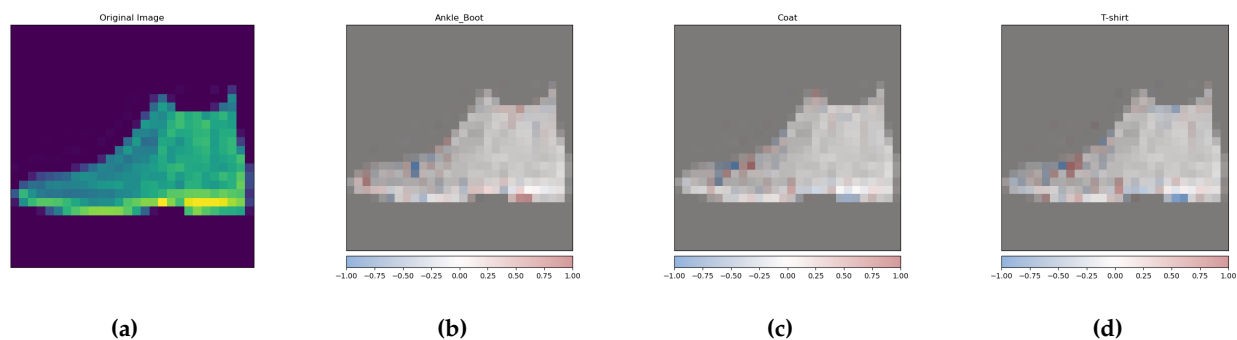


Figure 2. Original and analysis samples: (a) original image; (b) analysis of Ankle Boot classification; (c) analysis of Coat classification; (d) analysis of T-shirt classification.

The feature analysis plots corresponding to Coat and T-shirt for the categories in the calculation are shown in Figure 2c,d. The weights in the corresponding classification calculation results of the pixel points drawn in the figure show no obvious distribution pattern in the overall view. However, from the numerical point of view, the relationship between input pixels and output calculation results is established, and the calculation basis for image classification by the convolutional neural network can be obtained clearly and unambiguously, breaking the black box of the deep learning model and establishing the model with pixel points as the analysis angle.

Moreover, it can be observed from the feature analysis graph that both positive and negative numbers are included in the classification calculation result, which is the result obtained by combining multiple features, reflecting the complexity of the deep learning model. Both positive and negative numbers play an essential role in these results, with

positive numbers playing a positive role in correct classification and negative numbers playing a positive role in incorrect classification, which can be analyzed in detail in the ONN model.

The analysis is performed from the intermediate process of the corresponding convolutional neural network model, such as extracting a vector $E_{12,12}$ from the middle of the first $E^{24 \times 24}$ feature map in the data output of the first convolutional layer, with the result shown in Figure 3a. It is possible to analyze which input pixel points are involved in the computation of a point in the first feature map output from this convolutional layer, and what the linear relationship on an orthogonal basis is. The red box in the figure marks the range of the convolutional computation. The analysis results for a point $E_{4,4}$ at an intermediate position in the first feature map $E^{8 \times 8}$, which is the output of the second convolutional layer computation, are given in Figure 3b. The red box marks the features of the input layer that are included in the computation of this point. The analysis of the two convolutional layers shows that the image convolution can contain part of the position information in the calculation, and a clear correspondence can be specifically derived in the first convolutional layer. Since the activation function and maximum pooling are bypassed when entering the second convolutional layer, part of the position information has been lost, and the position information can still be determined based on the distribution of pixel points when analyzing the input layer pixels corresponding to the calculation result of that point; however, this position information does not contain the part that is not activated at the output of the first convolutional layer, nor the part that is filtered out by maximum pooling, which is more accurate in terms of scope.

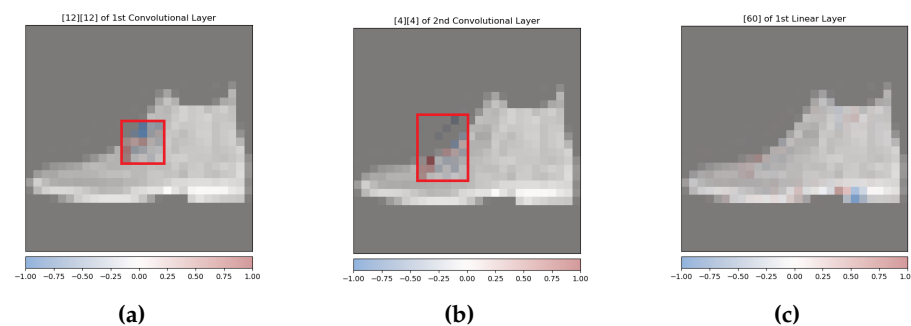


Figure 3. Analysis of the intermediate process of CNN: (a) first convolutional layer; (b) second convolutional layer; (c) first fully connected layer.

In Figure 3c are the results from the analysis of $e \in E$ output by the 60th neuron in the first fully connected layer, from which it is possible to analyze the input layer pixel point corresponding to the computational result output by this neuron. It can be seen that the layer has completely lost the convolutional position information preserved in the convolutional layer. Thus, it is possible to analyze the relationship between the input layer pixel point and the computational result of this point. It is also possible to further understand the role of the fully connected layer in the convolutional neural network, for which all spatial information is reconstructed for further progress in thoroughly learning the inherent features.

The complexity in deep learning is reflected in cases when removing a maximum pixel does not necessarily affect the classification result, such as convolution, pooling, and the computation of fully connected layers, which can prevent a single pixel from having too much impact on the classification result. In essence, this is due to the overall nonlinearity of the model, which can even be described as “complex” nonlinearity.

4. ONN Model Applied on Graph Neural Networks

The graph neural network is a relatively recent model in deep learning that has attracted wide attention due to its excellent performance with graph data. We choose the Cora [22] dataset and construct a graph neural network consisting of two graph convolu-

tional layers. The Cora dataset includes academic paper citation relationships with 2708 papers and 1433 keywords as features. The citation relations between papers are represented by the adjacency matrix $A \in \mathbb{R}^{2708 \times 2708}$, and all features are represented by the matrix $X \in \mathbb{R}^{2708 \times 1433}$. The node data of the graph mainly represents the features, and the graph's structure data mainly represents the paper's citation relationship; that is, the connected edge relationship between the nodes. The row and column indexes of matrix A correspond to the node indexes. The row indexes of matrix X correspond to the node indexes, and the column indexes correspond to the feature indexes. The Laplace matrix L is calculated from the adjacency matrix A in Equation (21). D is diagonal matrix.

$$L = D - A \quad (21)$$

4.1. A Graph Neural Network as An Example

The graph neural network consists of two graph convolutional layers with the following structure and parameters.

Graph convolution layer 1 consists of the Laplacian matrix $L \in \mathbb{R}^{2708 \times 2708}$ multiplied by the feature matrix $X \in \mathbb{R}^{2708 \times 1433}$, which is realized by aggregating the features of neighboring nodes through the graph structure to obtain a matrix of $\mathbb{R}^{2708 \times 1433}$ and then multiplied by a fully connected network containing 16 neurons, i.e., with the parameter matrix $W_1 \in \mathbb{R}^{1433 \times 16}$, and then over the ReLU activation function, and the whole graph convolution layer is computed as $X' = \text{ReLU}(LXW_1)$.

Graph convolution layer 2: a second feature aggregation is performed by multiplying the Laplacian matrix $L \in \mathbb{R}^{2708 \times 2708}$ with the output $X' \in \mathbb{R}^{2708 \times 16}$ of the previous graph convolution layer to obtain a matrix of $\mathbb{R}^{2708 \times 16}$, which is then multiplied by a fully connected network containing 7 neurons, i.e., with the parameter matrix $W_2 \in \mathbb{R}^{16 \times 7}$, which is the classification result of the graph neural network calculated as $Y = LX'W_2$, where $Y \in \mathbb{R}^{2708 \times 7}$ denotes the classification result of all 2708 nodes, with each calculated node corresponding to the value of seven classifications, where the largest value is the classification result given by the graph neural network.

4.2. Extended GNN in ONN Model

The feature matrix $X \in \mathbb{R}^{2708 \times 1433}$ is modulated by assigning a vector $\alpha \in \mathcal{A}$ to each feature according to a certain law, and for feature $x_{i,j} \in X$; the assigned vector is $\alpha_{i \times 28 + j} \in \mathcal{A}$. Thus, the modulated feature matrix $X_{mod} \in E^{2708 \times 1433}$ enters the graph convolution layer to perform matrix multiplication with the activation function according to the extended operation rules.

Graph convolution layer 1 consists of the Laplacian matrix $L \in \mathbb{R}^{2708 \times 2708}$ multiplied by the feature matrix $X \in E^{2708 \times 1433}$, which is realized by aggregating the features of neighboring nodes through the graph structure to obtain a matrix of $E^{2708 \times 1433}$, and then multiplied by a fully connected network containing 16 neurons, i.e., with the parameter matrix $W_1 \in \mathbb{R}^{1433 \times 16}$, and then over the ReLU activation function, and the whole graph convolution layer is computed as $X' = \text{ReLU}(LXW_1)$.

Graph convolution layer 2: a second feature aggregation is performed by multiplying the Laplacian matrix $L \in \mathbb{R}^{2708 \times 2708}$ with the output $X' \in E^{2708 \times 16}$ of the previous graph convolution layer to obtain a matrix of $E^{2708 \times 16}$, which is then multiplied by a fully connected network containing seven neurons, i.e., with the parameter matrix $W_2 \in \mathbb{R}^{16 \times 7}$ which is the classification result of the graph neural network calculated as $Y = LX'W_2$, where $Y \in E^{2708 \times 7}$ denotes the classification result of all 2708 nodes, and each calculated node corresponds to the value of seven classifications, where the largest value is the classification result given by the graph neural network.

The result output of the model is a matrix $Y_{mod} \in E^{2708 \times 7}$. The result contains the calculation results of all 2708 nodes. For each node, the result is a matrix of E^7 , then $\{S(e_1), S(e_2), \dots, S(e_7)\}$ that corresponds to the original network model calculation results for node classification.

4.3. Discussion

The analysis of the orthogonal neural network model has been described in more detail in the section on convolutional neural networks, and the analysis of orthogonal neural network model in graph neural networks is explained in this section. The expansion method of the graph neural network is given above for the Cora dataset. Since the graph data have the characteristics of being large scale and sparse, it is not convenient to directly use the calculation results for the dataset in discussion; instead, here, a set of simple calculation data are constructed for discussion.

The y-axis in Figure 4 represents the six nodes, and the coordinate values correspond to the respective node numbers, and the x-axis represents the six features. The coordinate values correspond to the feature numbers. The color of the blocks in the figure corresponds to the values within the blocks, which are extracted from the model calculation results and are used only to illustrate the analysis method, and they do not correspond to the calculations in the actual dataset. Here, it is assumed that the analysis is performed by drawing from the classification calculation result $\mathbf{e}_1$ of node 1. Nodes 21 and 35 are the neighbor nodes of node 1, and nodes 1400, 1512, and 2700 are the second-order neighbor nodes; that is, the neighbor nodes of the neighbor nodes. From the data, it can be seen that the computational results of the graph neural network include not only the features of its own nodes but also the features of neighboring nodes, which is the characteristic that enables the graph neural network to learn and aggregate the features of neighboring nodes through the structural information of the graph data and significantly improves the performance of deep learning in processing graph data.

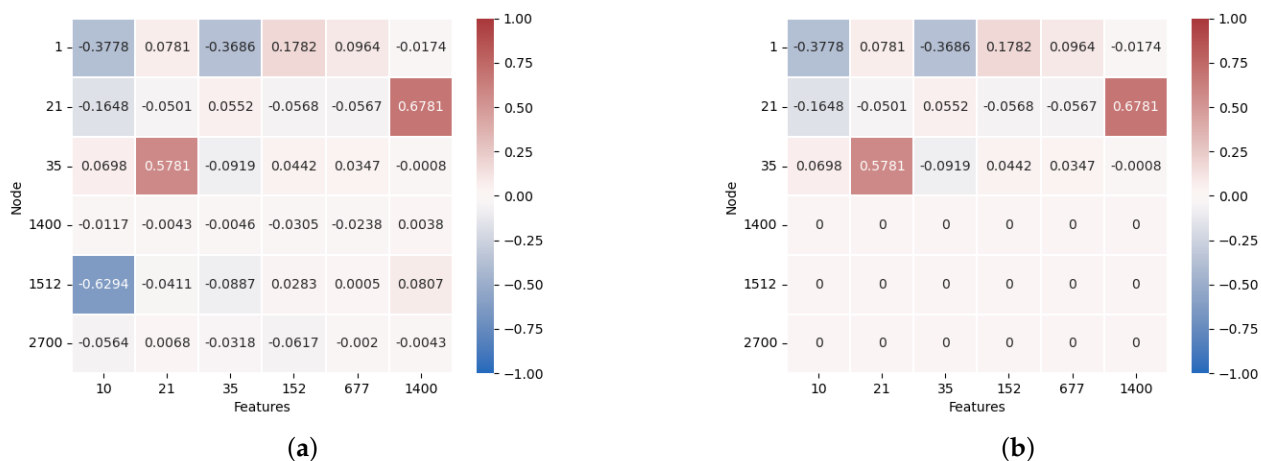


Figure 4. Analysis of (a) GNN and (b) first graph convolutional layer.

In Figure 4a, the larger positive values of feature 1400 of node 21 and feature 21 of node 35 indicate that these two features play a major role in this classification, and the larger and negative values of feature 10 of the second node 1512 indicate that this feature will also have a considerable influence on the classification. If $\mathbf{e}_1$ corresponds to a correct classification, the two positive features play a positive role in the classification, while the negative features play a negative role. If $\mathbf{e}_1$ corresponds to a wrong classification, the two types of features play the opposite role, with the negative features playing a positive role in control by ensuring that the classification calculation value cannot be greater than the value resulting from the correct classification. In contrast, the positive features play the exact opposite role.

In Figure 4b, the data represents a value in the matrix $\mathbb{R}^{2708 \times 16}$ of the output of the first graph convolutional layer, corresponding to a vector denoted as $\mathbf{e} \in E$ after orthogonal modulation. The reduction in the orthogonal vector information contained therein enables the analysis of another property of the graph neural network, which is that the model contains several graph convolutional layers and is able to aggregate the features of several

orders of neighboring nodes. The figure shows the output of the first graph convolutional layer, thus aggregating the features of the first-order neighbor nodes. Since $S(\mathbf{e}) > 0$, it enters the second graph convolutional layer through the activation function.

The analysis here also shows again that positive and negative values in the results are important. The reason for the good performance of the deep learning model lies in the complex computing process involved, which ensures diversity in feature learning. The idea of graph purification has been proposed in the study of graph neural networks for improving the model robustness by eliminating those nodes or connected edges that look significantly different from their neighbors according to the statistical laws of the graph structure. The idea of improving robustness, while also reducing the predictive performance of the model, is evident in the analytical approach in this paper, where the features that vary widely in classification can be learned in the process of parameter training with negative parameters, which can also play an important role. From a statistical point of view, small probability samples also have a meaningful existence and cannot be ignored because of their small probability; small probability samples must exist, and it is precisely because of their existence that the dataset has diversity.

5. Conclusions

In this paper, we proposed an analytical model for deep learning models, the orthogonal neural network model, which can analyze the computational process and results of deep learning models under limited conditions and analyze the quantitative relationship between the computational values of each part of the model and the initial features. The model establishes the relationship between the input features and results, quantifying this relationship in the form of weights. We applied the ONN model using two typical deep learning models, convolutional neural network and graph neural network, and demonstrated that the model, which uses an interpretable deep learning approach, focuses on each feature based on these weights. In subsequent studies, we will further analyze and investigate each potential research direction in deep learning based on the analytical findings of the proposed orthogonal neural network model.

Author Contributions: Methodology, Y.P.; Software, Y.P., R.H. and S.L.; Validation, H.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Program of Song Shan Laboratory (included in the management of Major Science and Technology Program of Henan Province) (221100210700-03).

Data Availability Statement: Data can be obtained by contacting the author (panyonghao2016@163.com).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Carlini, N.; Wagner, D. Towards Evaluating the Robustness of Neural Networks. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2017.
2. Goodfellow, I.; Shlens, J.; Szegedy, C. Explaining and Harnessing Adversarial Examples. In Proceedings of the International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015.
3. Kurakin, A.; Goodfellow, I.; Bengio, S. Adversarial examples in the physical world. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
4. Moosavi-Dezfooli, S.M.; Fawzi, A.; Frossard, P. DeepFool: A simple and accurate method to fool deep neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016.
5. Xie, C.; Wang, J.; Zhang, Z.; Zhou, Y.; Xie, L.; Yuille, A. Adversarial Examples for Semantic Segmentation and Object Detection. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017.
6. Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; Vladu, A. Towards deep learning models resistant to adversarial attacks. In Proceedings of the 6th International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
7. Xu, K.; Chen, H.; Liu, S.; Chen, P.Y.; Weng, T.W.; Hong, M.; Lin, X. Topology attack and defense for graph neural networks: An optimization perspective. In Proceedings of the 28th International Joint Conference on Artificial Intelligence, Macao, China, 13–16 August 2019; pp. 3961–3967.

8. Wu, H.; Wang, C.; Tyshetskiy, Y.; Docherty, A.; Lu, K.; Zhu, L. Adversarial Examples for Graph Data: Deep Insights into Attack and Defense. In Proceedings of the 28th International Joint Conference on Artificial Intelligence, Macao, China, 13–16 August 2019; pp. 4816–4823.
9. Zügner, D.; Akbarnejad, A.; Günnemann, S. Adversarial Attacks on Neural Networks for Graph Data. In Proceedings of the 28th International Joint Conference on Artificial Intelligence, Macao, China, 13–16 August 2019.
10. Zügner, D.; Günnemann, S. Adversarial Attacks on Graph Neural Networks via Meta Learning. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
11. Linardatos, P.; Papastefanopoulos, V.; Kotsiantis, S. Explainable ai: A review of machine learning interpretability methods. *Entropy* **2021**, *23*, 18. [[CrossRef](#)] [[PubMed](#)]
12. Zhou, B.; Khosla, A.; Lapedriza, A.; Oliva, A.; Torralba, A. Learning Deep Features for Discriminative Localization. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016.
13. Ribeiro, M.T.; Singh, S.; Guestrin, C. Why Should I Trust You?: Explaining the Predictions of Any Classifier. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016.
14. Fourier, J. *The Analytical Theory of Heat*; Dover Publishers: Mineola, NY, USA, 1906.
15. Nair, V.; Hinton, G.E. Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair. In Proceedings of the International Conference on Machine Learning, Haifa, Israel, 21–24 June 2010.
16. McCulloch, W.S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *J. Symb. Log.* **1943**, *9*, 49–50. [[CrossRef](#)]
17. Zhang, Z.; Cui, P.; Zhu, W. Imagenet classification with deep convolutional neural networks. *Commun. ACM* **2017**, *60*, 84–90.
18. Zhou, J.; Cui, G.; Hu, S.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; Sun, M. Graph neural networks: A review of methods and applications. *AI Open* **2021**, *1*, 57–81. [[CrossRef](#)]
19. Zhang, Z.; Cui, P.; Zhu, W. Deep Learning on Graphs: A Survey. *IEEE Trans. Knowl. Data Eng.* **2022**, *34*, 249–270. [[CrossRef](#)]
20. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Philip, S.Y. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *32*, 4–24. [[CrossRef](#)] [[PubMed](#)]
21. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv* **2017**, arXiv:1708.07747..
22. Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; Eliassi-Rad, T. Collective Classification in Network Data Articles. *Ai Mag.* **2008**, *29*, 93–106. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.