Unraveling the Black-box Magic: An Analysis of Neural Networks' Dynamic Local Extrema

Authors: Shengjian Chen, Shengjian Chen

Date: 2025-07-08T11:01:44+00:00

Abstract

We point out that neural networks are not black boxes, and their generalization stems from the ability to dynamically map a dataset to the local extrema of the model function. We further prove that the number of local extrema in a neural network is positively correlated with the number of its parameters, and on this basis, we give a new algorithm that is different from the back-propagation algorithm, which we call the extremum-increment algorithm. Some difficult situations, such as gradient vanishing and overfitting, can be reasonably explained and dealt with in this framework.

Full Text

Preamble

Unraveling the Black-box Magic: An Analysis of Neural Networks' Dynamic Local Extrema
Shengjian Chen
Intelligent Robotics Center, Jihua Laboratory
Foshan, 528200, China
chensj@jihualab.ac.cn, chshengj@mail2.sysu.edu.cn

Abstract

We point out that neural networks are not black boxes, and their generalization stems from the ability to dynamically map a dataset to the local extrema of the model function. We further prove that the number of local extrema in a neural network is positively correlated with the number of its parameters, and on this basis, we give a new algorithm that is different from the back-propagation algorithm, which we call the extremum-increment algorithm. Some difficult situations, such as gradient vanishing and overfitting, can be reasonably explained and dealt with in this framework.

Keywords: Neural network, generalization, black box, extreme increment, linear equation system

1 Introduction

Although artificial intelligence models based on neural networks have been extensively studied and widely applied, and their prediction accuracy in fields such as image recognition, natural language processing, text processing and question answering far exceeds that of traditional machine learning algorithms, there is a lack of relevant research on their underlying principles, and they are still generally regarded as black boxes. With the rapid increase of model parameters, from ANN to CNN, RNN, and then to GPT and LLM [**?**, **?**], its complexity also increases sharply, while the stability of the system becomes more vulnerable accordingly. If the model malfunctions, it is impossible for us to quickly identify the root cause of the problem and solve it immediately without understanding its logic. For some fields with low requirements for real-time performance, such as image classification and AI-generated artwork, the application of neural network algorithms can be confidently promoted. However, for some fields that require high real-time performance, especially safety, such as autonomous driving [**?**, **?**], it is necessary to pay more attention to the underlying principles of neural networks and clarify the conditions under which they take effect and fail, so that artificial intelligence can better serve human society.

Although the model structure of neural networks has become prohibitively complex, some scholars still strive to explore their working principles. Buhrmester et al. (2021) investigated the explainers that have been popular in recent years. This method attempts to explain neural networks by analyzing the connections between inputs and outputs. The characteristic of black-box explainers is that it does not need to access the internal structure of the model to reveal all the interaction details of the model. They are mainly divided into ante-hoc systems with a global, model-agnostic feature [**?**, **?**] and post-hoc ones with a local, model-specific feature [**?**, **?**]. Oh et al. (2019) analyzed neural networks from the perspective of reverse engineering, and found that they are extremely vulnerable to different types of attacks, and pointed out that the boundary between a white box and a black box is not obvious. Tishby and Zaslavsky (2015) took a different approach and proposed the Information Plane. Furthermore, they believed that the main goal of neural networks was to optimize the Information Bottleneck between the compression and prediction of each layer. Shwartz-Ziv and Tishby (2017) further proved the effectiveness of this method. These works provide useful references for the underlying research of neural networks, but there is still a long way to go.

Current researchers seem to be overly obsessed with using engineering methods to explain how neural networks work, while neglecting to explain from a theoretical or mathematical perspective. Neural networks may seem complex, but their structure is clear and they are basically composed of neurons with the same construction, making them particularly suitable for mathematical analy-

sis. It is necessary to revisit the pioneering work of Cybenko (1989) and Hornik et al. (1989) in this area, which proved that feedforward neural networks can approximate any continuous function on a compact set. That is, a feedforward neural network that adopts a single hidden layer with a sufficient number of neurons, and uses the sigmoid function as the activation function can approximate any complex function with arbitrary accuracy, providing a basic mathematical principle for neural networks. The only drawback of this work is that it does not provide a method on how to find the specific function on a given dataset and whether this function is the optimal one. Our work makes up for its deficiencies.

Specifically, our contributions mainly include the following aspects: 1) We present the main characteristics of an ideal machine learning model, and based on which we provide the general model training steps. This work is mainly discussed in Section 2. 2) We discuss whether neural networks satisfy the ideal model characteristics and point out from a mathematical perspective that neural networks mainly achieve generalization by mapping a dataset to the local extrema of the function. We further present a model training algorithm different from the back-propagation (BP) algorithm, namely the extremum-increment (EI) algorithm. This work is mainly discussed in Sections 3 and 4. 3) Based on EI algorithm, we can relatively easily point out the causes of some common problems, such as vanishing/exploding gradients, overfitting, etc., and provide corresponding solutions. This work is mainly discussed in Section 5.

2 General Characteristics of an Ideal Model

Let' s temporarily put aside the concept of neural networks and imagine the basic characteristics of a model that satisfies a dataset and the target task. The training goal of machine learning is to obtain a function curve that can precisely fit the inputs of all samples with their corresponding outputs. That is to say, this model can clearly tell us what the exact value of each input sample is after processing. For example, for classification problems, this model can give an output of "This is a cat." instead of a vague answer of "This is very likely to be a cat."

2.1 Precise Mapping

Situations where there are no same-type samples. For the sake of visualization, in the discussion of this paragraph, we limit the sample size to 3. As shown in Figure 1 [Figure 1: see original paper], let the dataset be $D = \{(x^{(i)}, y^{(i)}) | i \in [1, 3]\}$, where $(x^{(i)}, y^{(i)})$ is the i-th sample, $x^{(i)}$ is the original representation of the sample, and $y^{(i)}$ is the category to which $x^{(i)}$ belongs. Our goal is to find a function F for each $x^{(i)}$ such that $y^{(i)} = F(x^{(i)})$.

To reveal the true working principle of neural networks, we abandon the concepts of feature and label, and instead use surface and essence to refer to $x^{(i)}$ and $y^{(i)}$. Meanwhile, in order to grasp the key of the problem and simplify it, both the surface and the essence in Section 2 are represented by scalars. The function F

shown in Figure 1 is the ideal model we hope to obtain, because for any surface $x^{(i)}$, it can precisely give the corresponding essence $y^{(i)}$.

Situations where there are same-type samples. As shown in Figure 2 [Figure 2: see original paper], if a new sample that is essentially the same as one in the dataset D is added, for instance, adding sample $(x^{(3,1)}, y^{(3)})$, then the function curve F needs to change its shape so that the new sample can just fall onto the function curve. At this time, there is a local maximum between the samples $(x^{(3,1)}, y^{(3)})$ and $(x^{(3)}, y^{(3)})$ on the function curve.

Similarly, as shown in Figure 3 [Figure 3: see original paper], if new samples with the essence of $y^{(3)}$, such as $(x^{(3,2)}, y^{(3)})$, $(x^{(3,3)}, y^{(3)})$, and $(x^{(3,4)}, y^{(3)})$, are continuously added, the function curve needs to further change its shape to accommodate these new samples, thereby forming multiple local minima/maxima. If function F can achieve such shape alteration, it possesses the ability to precisely map any surface to its essence, meaning it has true generalization capability.

2.2 Weakened Mapping

To obtain the aforementioned ideal function, the computation is usually enormous. For a function with a limited number of parameters, the degree of change in its curve shape is limited, and the number of extreme values cannot be increased at will. Then, how should we handle the situation when the surface of a sample has only changed slightly while its essence remains unchanged? A natural idea is to expand the essence from a single point to an interval, so that samples with slightly different surfaces but the same essence can be concentrated in this interval. As shown in Figure 4 [Figure 4: see original paper], we add sample $(x^{(3,5)}, y^{(3)})$ where the distance between $x^{(3,5)}$ and $x^{(3)}$ is small enough. We adjust the precise mapping function F to the approximately fitting function F^*, making the difference between $F^*(x^{(3,5)})$ and $F^*(x^{(3)})$ as close as possible. When $|F^*(x^{(3,5)}) - F^*(x^{(3)})|$ is small enough, we can approximately consider that the surfaces falling within the interval $[F^*(x^{(3,5)}), F^*(x^{(3)})]$ all have the essence of $y^{(3)}$. Then we call function F^* a weakened model of function F.

Interval partition. Each sample consists of both a surface and an essence. A surface is usually a one-dimensional vector or a multi-dimensional matrix. Once the algorithm for generating the surface is determined, for example, using a two-dimensional matrix to represent a grayscale image, where each matrix element' s value ranges from 0 to 255, then the surface is definite and we cannot make further changes to it. However, the essence is different. It is usually just an abstract concept and can be represented by any scalar or vector. As shown in Figure 4, if the shape of the curve of the function F is restricted, then each essence requires a tolerance interval. How is this interval selected? One method is to divide the range of the function F into N intervals with the same length, where N is the total number of essence types. Then each interval is assigned to each essence, and the essences of surfaces that fall within the same interval are the same. When dividing essences using this method, the range of values of the

objective function should be finite.

2.3 N Classification to Binary Classification

There is a problem in the interval partition method. When there are many types of essences and the value range of the function F is limited within a small interval, for example, the value of each element in the output layer of a neural network is limited within a small interval $(0, 1)$, then overlap is prone to occur. One solution is to reduce the types of substance and thereby expand each partition. But this introduces two new problems. One is to what extent the essence types should be reduced? Second, how should the essence of being excluded be handled?

For the above problems, we can reduce the number of essences to only one type. That is to say, the target model changes from an N-classification function F to N binary classification functions $\{F_j | j \in [1, N]\}$, and the j-th binary classification function F_j only determines whether the input sample belongs to the j-th type of essences. That is, for any given sample $(x^{(i)}, y^{(i)})$, where $i > 0$, the ideal objective function F_j satisfies:

$$F_j(x^{(i)}) = \begin{cases} 1 & y^{(i)} = j \\ 0 & y^{(i)} \neq j \end{cases}$$

The weakened objective function F_j^* satisfies:

$$F_j^*(x^{(i)}) \in \begin{cases} [LB^*, UB^*], & y^{(i)} = j \\ [LB^*, UB^*], & y^{(i)} \neq j \end{cases}$$

where LB and UB are the lower and upper limits of the function F_j respectively, and correspondingly, LB^* and UB^* are the lower and upper limits of the function F_j^*. For the ideal function F_j, each given sample is adjusted to be its extremum point. Figure 5 [Figure 5: see original paper] presents an instance of a binary classification function F_3. We adjust the parameters of F_3 so that all samples of the third-essence are adjusted to the upper limit of the function' s value range, all other-essence samples are adjusted to the lower limit. Correspondingly, the weakened function F_3^* uses the midpoint of the value range as the dividing line. The same parameter adjustment is made for other binary classification functions (such as F_1, F_2).

2.4 General Training Process of an Ideal Model

In summary, the ideal training process for all machine learning models that are essentially classification problems can be summarized as the following steps: 1) Transform the N-class objective function F into a family of binary classification functions $\{F_j | j \in [1, N]\}$ and initialize all parameters. 2) For each F_j, adjust the

parameters so that each training surface $x^{(i)}$ is exactly one of the extrema of the function. 3) For each F_j, adjust the parameters so that the training samples of the j-th essence become a local maximum, and those of non-j-th essence become a local minimum. 4) Adjust the parameters to make the local maximum the global maximum and the local minimum the global minimum.

The model trained through the above steps can accurately map the input to the output, thereby enabling the machine to precisely answer "yes" or "no" . If in actual debugging, we cannot find or it is very difficult to find an ideal function like that in Figure 5, then a weakened function is trained that the surfaces or points within the surfaces' neighbors is adjusted to be its local extreme points first, and then the parameters of the function are adjusted to move these local extreme points as close as possible to the upper or lower limit of the function' s range, thereby achieving a function that is close to that in Figure 5.

3.1 Model Decomposition

Any type of neural network, whether it is a traditional artificial neural network, an improved convolutional neural network, or a recurrent neural network, is composed of three parts: an input vector with a fixed number of elements, an intermediate processing layer with undetermined parameters, and an output vector with the number of elements equal to the number of essence types. To reduce computational complexity and focus on the main working process of neural networks, we only conduct derivative analysis on the fully connected neural networks. Additionally, the output layer of mainstream neural networks often uses the softmax function, which is just a normalization operation added to the sigmoid function. To simplify the operation steps, we directly use the sigmoid function as the output layer, so that both the hidden layers and the output layer use the sigmoid function.

Moreover, we remove the biases because they are irrelevant to the essential attributes of the model but make the calculations lengthy and reduce readability. Then on this basis, we analyze the structure of a fully connected neural network based on an N-classification problem.

Figure 6 [Figure 6: see original paper] is a schematic diagram of a fully connected neural network structure expressed directly through numerical relations and without graphical representation. Each sample is denoted as (x, y), where the surface x is an m-dimensional column vector, $x = (x_1, x_2, \ldots, x_m)^T$, and $y \in [1, l_n]$ is the essence corresponding to x, where l_n is the number of elements in the neural network' s output vector, representing the number of essence types.

The neural network has a total of n layers with the same processing method, with the first $n-1$ layers being hidden layers and the n-th layer being the output layer. The total number of elements in the u-th layer (with the input vector being the 0-th layer) is denoted as l_u, and the v-th element in the u-th layer is denoted as $h_v^{[u]}(x)$, where $v \in [1, l_u]$. As can be seen from Figure 6, disregarding the dazzling connection of the neurons, a neural network is actually a set composed

of l_n composite functions $\{h_v^{[n]}(x)|v \in [1, l_n]\}$, with each composite function sharing the same hidden layers.

For samples belonging to the v-th essence where $v \in [1, l_n]$, the target output vector of the neural network is $(0, \ldots, h_v^{[n]}(x) = 1, \ldots 0)^T$. For all other essence types, the target output vector of the neural network is $(\omega, \ldots, h_v^{[n]}(x) = 0, \ldots \omega)^T$ where one of the ω is 1 and the others are all 0. In this simplified model, it is worth mentioning that when the sigmoid function is used as the output, the upper and lower limits of $h_v^{[n]}(x)$ can only approach 1 and 0 infinitely. When we transform the output layer that seems to be composed of an l_n-dimensional vector into l_n scalars, the entire model becomes very clear. That is, each composite function $h_v^{[n]}(x)$ is actually a binary classification problem of the v-th essence.

Therefore, a neural network with a multi-dimensional vector output layer can actually be regarded as a collection of multiple binary classification functions as shown in Figure 7 [Figure 7: see original paper]. We can analyze each function $h_v^{[n]}(x)$ separately and then integrate them to obtain the characteristics of the neural network.

3.2 Extreme Points of the Model

Specifically, the expression of the function $h_v^{[u]}(x)$ satisfies:

$$h_v^{[u]}(x) = \begin{cases} S\left(\sum_{k=1}^{l_{u-1}} w_{v,k}^{[u]} \cdot h_k^{[u-1]}(x)\right), & u > 1 \\ S\left(\sum_{k=1}^{m} w_{v,k}^{[u]} \cdot x_k\right), & u = 1 \end{cases}$$

The function $S(\theta) = \frac{1}{1+e^{-\theta}}$ is the sigmoid function, and $w_{v,k}^{[u]}$ represents the parameters between the $(u-1)$-th layer and the u-th layer of the neural network, which is the same as the parameters in traditional neural networks. To further enhance readability, let $z_v^{[u]}(x) = \sum_{k=1}^{l_{u-1}} w_{v,k}^{[u]} \cdot h_k^{[u-1]}(x)$. Then we take the partial derivative of the function $h_v^{[u]}(x)$:

$$\frac{\partial h_v^{[u]}(x)}{\partial x_t} = S'(z_v^{[u]}(x)) \cdot \frac{\partial z_v^{[u]}(x)}{\partial x_t} = S(z_v^{[u]}(x)) \cdot (1 - S(z_v^{[u]}(x))) \cdot \frac{\partial z_v^{[u]}(x)}{\partial x_t}$$

where $t \in [1, m]$, and the derivative result of the function $S(\theta)$ is adopted, i.e., $\frac{\partial}{\partial \theta} S(\theta) = S(\theta) \cdot (1 - S(\theta))$. Let $c_v^{[u]}(x) = S(z_v^{[u]}(x)) \cdot (1 - S(z_v^{[u]}(x)))$, then:

$$\frac{\partial h_v^{[u]}(x)}{\partial x_t} = c_v^{[u]}(x) \cdot \frac{\partial z_v^{[u]}(x)}{\partial x_t} = c_v^{[u]}(x) \cdot \sum_{k=1}^{l_{u-1}} w_{v,k}^{[u]} \cdot \frac{\partial h_k^{[u-1]}(x)}{\partial x_t}$$

When $\frac{\partial h_v^{[u]}(x)}{\partial x_t} = 0$, since $c_v^{[u]}(x) > 0$, then $\sum_{k=1}^{l_{u-1}} w_{v,k}^{[u]} \cdot \frac{\partial h_k^{[u-1]}(x)}{\partial x_t} = 0$. Then starting from the output layer, i.e., $u = n$, we take the partial derivatives of all components of x respectively, and obtain the following system of equations:

$$L(n,v) = \begin{cases} \sum_{k=1}^{l_{n-1}} w_{v,k}^{[n]} \cdot \frac{\partial h_k^{[n-1]}(x)}{\partial x_1} = 0 \\ \sum_{k=1}^{l_{n-1}} w_{v,k}^{[n]} \cdot \frac{\partial h_k^{[n-1]}(x)}{\partial x_2} = 0 \\ \vdots \\ \sum_{k=1}^{l_{n-1}} w_{v,k}^{[n]} \cdot \frac{\partial h_k^{[n-1]}(x)}{\partial x_m} = 0 \end{cases}$$

When a surface x is given, the above system of equations is a homogeneous linear equation system consisting of m equations, l_{n-1} independent variables $\{w_{v,k}^{[n]} | k \in [1, l_{n-1}]\}$ and $m \cdot l_{n-1}$ coefficients $\{\frac{\partial h_k^{[n-1]}(x)}{\partial x_t} | k \in [1, l_{n-1}], t \in [1, m]\}$. Let the rank of the coefficient matrix of $L(n, v)$ be $r(n, v)$, then when $r(n, v)$ is less than the number of unknowns l_{n-1}, the linear equation system has infinitely many solutions. Since $r(n, v) \leq m$, as long as the number of neurons in the last hidden layer l_{n-1} is greater than m when designing a neural network, we can always find infinitely many parameter combinations that make the surface x be an extremum point of the binary classification function $h_v^{[n]}(x)$ for $v \in [1, l_n]$. This is the main reason why neural networks have strong generalization ability, and the black box begins to be unveiled.

The shapes of the curves for the other binary classification functions can be adjusted simultaneously. Let $L(n, -) = L(n, 1) \oplus L(n, 2) \oplus \cdots \oplus L(n, l_n)$. When given a surface x, $L(n, -)$ is a homogeneous linear system of equations consisting of $m \cdot l_n$ equations, $l_n \cdot l_{n-1}$ variables $\{w_{v,k}^{[n]} | v \in [1, l_n], k \in [1, l_{n-1}]\}$, and $m \cdot l_{n-1}$ coefficients $\{\frac{\partial h_k^{[n-1]}(x)}{\partial x_t} | k \in [1, l_{n-1}], t \in [1, m]\}$. Any solution of $L(n, -)$ makes the surface x be the extremum point of each binary classification function. Then we can select a particular solution such that when the surface x belongs to the v-th essence, the corresponding extremum is the maximum value, and when x belongs to other essences, the extremum is the minimum value. That is, $h_v^{[n]}(x)$ satisfies the ideal termination condition of model training:

$$h_v^{[n]}(x) = \begin{cases} 1 & y = v \\ 0 & y \neq v \end{cases}, \quad y \in [1, l_n]$$

If it is difficult to find the above particular solution, then the constraints can be relaxed, that is, a weakened termination condition can be adopted:

$$h_v^{[n]}(x) \in \begin{cases} (0.5, 1], & y = v \\ [0, 0.5), & y \neq v \end{cases}, \quad y \in [1, l_n]$$

3.3 Continuous Optimization of Parameter Combinations

The above discussion only covers the situation where there is only one training sample. What should be done when the number of samples increases? Let:

$$L(n,v,x^{(i)}) = \begin{cases} \sum_{k=1}^{l_{n-1}} w_{v,k}^{[n]} \cdot \frac{\partial h_k^{[n-1]}(x)}{\partial x_1}|_{x=x^{(i)}} = 0 \\ \sum_{k=1}^{l_{n-1}} w_{v,k}^{[n]} \cdot \frac{\partial h_k^{[n-1]}(x)}{\partial x_2}|_{x=x^{(i)}} = 0 \\ \vdots \\ \sum_{k=1}^{l_{n-1}} w_{v,k}^{[n]} \cdot \frac{\partial h_k^{[n-1]}(x)}{\partial x_m}|_{x=x^{(i)}} = 0 \end{cases}$$

Then, $L(n,-,x^{(i)}) = L(n,1,x^{(i)}) \oplus L(n,2,x^{(i)}) \oplus \cdots \oplus L(n,l_n,x^{(i)})$. When we train the neural network with a dataset $\Phi = \{(x^{(i)},y^{(i)})|i \in [1,\phi]\}$, we are actually solving the following homogeneous linear equation system:

$$L(n,-,\Phi) = L(n,-,x^{(1)}) \oplus L(n,-,x^{(2)}) \oplus \cdots \oplus L(n,-,x^{(\phi)})$$

If $L(n,-,\Phi)$ has infinitely many solutions, then a particular solution that meets the conditions can be found from the general solution of the system of equations. Otherwise, parameters between the $(n-2)$-th layer and the $(n-1)$-th layer need to be introduced. That is, we need to expand the partial derivatives in the system of equations $L(n,v)$ again. We substitute $\frac{\partial h_k^{[n-1]}(x)}{\partial x_t} = c_k^{[n-1]}(x) \cdot \sum_{p=1}^{l_{n-2}} w_{k,p}^{[n-1]} \cdot \frac{\partial h_p^{[n-2]}(x)}{\partial x_t}$ into $L(n,v)$, and simplify the system:

$$L(n-1,v) = \begin{cases} \sum_{k=1}^{l_{n-1}} w_{v,k}^{[n]} \cdot \sum_{p=1}^{l_{n-2}} w_{k,p}^{[n-1]} \cdot \frac{\partial h_p^{[n-2]}(x)}{\partial x_1} = 0 \\ \sum_{k=1}^{l_{n-1}} w_{v,k}^{[n]} \cdot \sum_{p=1}^{l_{n-2}} w_{k,p}^{[n-1]} \cdot \frac{\partial h_p^{[n-2]}(x)}{\partial x_2} = 0 \\ \vdots \\ \sum_{k=1}^{l_{n-1}} w_{v,k}^{[n]} \cdot \sum_{p=1}^{l_{n-2}} w_{k,p}^{[n-1]} \cdot \frac{\partial h_p^{[n-2]}(x)}{\partial x_m} = 0 \end{cases}$$

Similarly, we can obtain the system of equations $L(n-1,-)$, which can be regarded as a homogeneous nonlinear system of equations consisting of $m \cdot l_n$ equations with $l_{n-1} \cdot l_n + l_{n-2} \cdot l_{n-1}$ independent variables $\{w_{v,k}^{[n]}|v \in [1,l_n], k \in [1,l_{n-1}]\}$ and $\{w_{k,p}^{[n-1]}|k \in [1,l_{n-1}], p \in [1,l_{n-2}]\}$. Although $L(n-1,-)$ seems to be a nonlinear system of equations, due to its very regular structure, for instance, $w_{v,k}^{[n]} \cdot w_{k,p}^{[n-1]}$ can be regarded as a whole, then the solution method for homogeneous linear equations can still be adopted. Then we just need to find the particular solution of the equation system $L(n-1,-,\Phi)$ that meets the requirements.

By solving the homogeneous equations layer by layer, the dataset can be mapped to the neural network.

4.1 General Training Method

From the above discussion, we have obtained a preliminary model training framework, which we call the EI algorithm. Its main steps have significant differences from the current commonly used neural network training methods, such as the BP algorithm. Firstly, the BP algorithm uses gradient updates to approximate the ideal values of parameters, while the EI algorithm attempts to directly obtain the values of parameters by solving systems of equations. Secondly, the BP algorithm needs to update all parameters each time, while the EI algorithm only needs to update some parameters. Debugging all training samples to the extremum points of the model is the key to the entire framework. In this subsection, we will have a more in-depth discussion on the details of the algorithm.

Table 1 shows the state of neural network parameters based on EI algorithm at each round where $W^{[u]} = \{w_{v,k}^{[u]} | v \in [1, l_u], k \in [1, l_{u-1}]\}$, "init" indicates that the parameters remain at their initial values, and "update" indicates that the parameters are updated in the current round. In the first round, we first solve the equation system $L(n, -, \Phi)$. If there is a solution, only the parameters $W^{[n]}$ need to be updated. Otherwise, in the second round, we solve the equation system $L(n-1, -, \Phi)$. If there is a solution, the parameters $W^{[n]}$ and $W^{[n-1]}$ need to be updated. Otherwise, we solve the equation system $L(n-2, -, \Phi)$, and so on.

Algorithm 4.1 presents the main steps for precisely mapping a dataset to the neural network model. The symbols used, unless otherwise specified, have the same meanings as those in the previous text. In the initial stage of the algorithm, we manually label the sample set Φ. If a sample $(x^{(i)}, y^{(i)})$ is classified as the j-th essence where $j \in [1, l_n]$, then $(x^{(i)}, y^{(i)}) = (x^{(i)}, j)$. After that, we initialize the parameter set W to non-zero real numbers.

Table 1: Weights' states in different stages.

stage	weight	$W^{[1]}$	$W^{[2]}$	...	$W^{[n-2]}$	$W^{[n-1]}$	$W^{[n]}$
$L(n, -, \Phi)$	init	init	...	init	init	update	
$L(n-1, -, \Phi)$	init	init	...	init	update	update	
$L(n-2, -, \Phi)$	init	init	...	update	update	update	

Algorithm 1 Precise mapping from input to output

Input: $\phi = \{(x^{(i)}, y^{(i)}) | i \in [1, \phi]\}$
Output: $W = \{w_{v,k}^{[u]} | u \in [1, n], v \in [1, l_u], k \in [1, l_{u-1}]\}$

```
1: function FittingCurve()
2:   Init(W)
3:   for u  [1, n-1], v  [1, l_u], t  [1, m], i  [1, ] do
```

```
    Calculate( h^{[u]}_v(x)/ x_t|_{x=x^{(i)}})
  end for
  for u   [1, n-1], v   [1, l_u], i   [1,  ] do
    Calculate(h^{[u]}_v(x^{(i)}))
  end for
  u ← n
  while u   1 do
    W^{[u:n]} ← L(u, -,  )  // for calculating W[u:n]
    W^{[u:n]} ← Polarize({h^{[n]}_v(x)|v   {1, l_n}}, W^{[u:n]},  )
    if W^{[u:n]}   {}   W^{[u:n]}   {0} then
      W ← Update(W^{[u:n]});
      break  // the general solution, equals to {W^{[j]}|j   [u, n]}
    end if
    u ← u - 1
  end while
  if u   1 then
    return W
  else
    return Error()
  end if
end function
```

Just like the BP algorithm, the parameter update is executed layer by layer from the last hidden layer to the first hidden layer. We first calculate the values and partial derivatives of all the neurons in the hidden layers, and then solve the general solution $W^{[u:n]} = \{W^{[j]}|j \in [u,n]\}$ of the equation group $L(u,-,\Phi)$. Then, we select a particular solution $W^{[u:n]}$ that satisfies the termination condition from $W^{[u:n]}$. We call this operation "polarize" . If a particular solution $W^{[u:n]}$ is found, the parameters W are then updated. If no particular solution is found, it indicates that the parameter set $W^{[u:n]}$ cannot precisely map the sample set Φ to the neural network. Then, we need to introduce the parameter $W^{[u-1]}$ to find the particular solution $W^{[u-1:n]}$ of the equation group $L(u-1,-,\Phi)$. If no particular solution that meets the requirements is found after traversing all the parameters of the neural network, we need to consider adjusting the structure of the neural network (such as increasing the number of hidden layers or the number of nodes in each hidden layer).

4.2 Reduce the Computational Complexity

In Algorithm 4.1, the polarization time for selecting a specific solution $W^{[u:n]}$ from the general solution $W^{[u:n]}$ is uncertain. This is because we do not know the characteristics of the specific solution and can only verify each instance of the general solution through enumeration. To reduce the training time, we can relax the termination condition of the model training. That is, when a sample $(x^{(i)}, y^{(i)})$ is a sample of the v-th essence, the value of the v-th binary classification function $h_v^{[n]}(x^{(i)})$ just needs to be much greater than the values of any

other binary classification function $h_q^{[n]}(x^{(i)})$, without considering whether these values are maximum or minimum values. This is equivalent to the following weakened condition:

$$\frac{h_v^{[n]}(x^{(i)})}{\sum_{j=1}^{l_n} h_j^{[n]}(x^{(i)})} > 1-\alpha, \quad v \in [1, l_n], y^{(i)} = v$$

$$\frac{h_q^{[n]}(x^{(i)})}{\sum_{j=1}^{l_n} h_j^{[n]}(x^{(i)})} < \beta, \quad \text{any } q \in [1, l_n], q \neq v$$

where α and β are two sufficiently small positive real numbers. This is the situation when the softmax function is used as the output layer of the neural network. That is to say, a neural network using the softmax function can be regarded as a weakened version of an ideal model.

4.3 Reduce the Computational Scale

If each training sample corresponds to an extreme point on the model curve, that is, by adding an equation set $L(n, -, x^{(i)})$, the required scale of network parameters will be extremely large and the training time will also increase significantly. Is there a way to reduce the number of equation sets? To this end, we propose the concept of surface neighborhood. In a further weakened neural network, only a portion of the samples need to be the extreme points, and the other samples can only satisfy the weakened termination condition 2. Then which samples can have their restrictions relaxed? An intuitive idea is that only the representative of all adjacent samples needs to satisfy the strict condition.

Let $A = (x^{(a)}, y^{(a)})$ and $B = (x^{(b)}, y^{(b)})$ be two samples in the dataset $\Phi = \{(x^{(i)}, y^{(i)}) | i \in [1, \phi]\}$ where $a, b \in [1, \phi]$. Then the distance between these two samples is defined as:

$$D_s(A, B) = \sqrt{\frac{1}{2 \cdot \dim(x)} \sum_{j=1}^{\dim(x)} (x_j^{(a)} - x_j^{(b)})^2}$$

If A and B are samples of the same essence, that is, $y^{(a)} = y^{(b)}$, and the proximity criterion is satisfied:

$$D_s(A, B) < \gamma$$

where $\dim(x)$ represents the dimension of a sample surface, and γ is a sufficiently small positive real number. Then we say that samples A and B of the same essence are located within each other' s neighborhood. Due to the continuity of the function, it can be known that the function values of samples A and B

are close on each binary classification function. Thus, one of the samples does not need to be sent to the algorithm for training but only needs to verify its function value. Then a further weakened training algorithm can be adjusted as follows: 1) Manually classify the training sample set and designate it as the major category. 2) Utilize a certain numerical algorithm, such as clustering algorithm, to further divide the samples of each major category into several minor categories. Each minor category has a central sample. 3) Train the model for all the central samples. 4) After the training is completed, verify whether the predicted values of all non-central samples on the neural network are within the specified accuracy range. If the accuracy requirements are met, the algorithm ends. Otherwise, mark the non-central samples that do not meet the requirements as central samples and repeat steps 3) and 4).

5.1 Gradient Vanishing/Explosion

The problem of gradient vanishing/exploding is a common and very difficult issue encountered during the training of neural networks, and it is particularly prone to occur in deep networks. To address this problem, some scholars have proposed various methods to alleviate the adverse effects of gradient vanishing/exploding on parameter updates, such as using batch normalization [**?**, **?**] and LSTM architecture [**?**, **?**]. In the BP algorithm, the problem of gradient vanishing/exploding is often regarded as an abnormal issue that should be avoided.

Regarding the problem of gradient vanishing, as discussed in Sections 3 and 4, after the initialization of network parameters, the number of parameter updates required by the neural network varies depending on the sample size. If the particular solution $W^{[u:n]}$ can be found from the general solution $W^{[u:n]}$, then the parameters $W^{[1:u-1]}$ of the earlier hidden layers can remain at their initial values. That is to say, according to the neural network characteristics revealed by the EI algorithm, gradient vanishing is an inevitable result. Gradient explosion is also a similar problem. In the EI algorithm, when we calculate the equation set $L(u, -, \Phi)$, there may be cases where no solution exists. That is to say, the value of the solution is infinity, which corresponds to the gradient explosion in the BP algorithm. If the EI algorithm is adopted, we only need to continue to solve the equation set $L(u-1, -, \Phi)$.

5.2 Overfitting

The overfitting problem [**?**, **?**] seems different from the gradient vanishing/explosion problem, but in essence, both are caused by the same operational process. In the EI algorithm, if the equation set $L(1, -, \Phi)$ has solutions, but when we increase the number of the samples, the equation set $L(1, -, \Phi_{\Delta})$ maybe has no solution where $\Phi \subset \Phi_{\Delta}$. That is, the neural network with the current parameter scale can only accommodate a limited number of samples Φ, manifested as the overfitting phenomenon of the BP algorithm. This is an inherent characteristic of neural networks that there are only a limited number

of extreme values under the condition of limited parameters. Rather than saying it' s overfitting, we would rather say it fits just right.

The BP algorithm reduces the model' s dependence on the trained samples by adding noise to the samples and network parameters [**?**, **?**]. This method is similar to the clustering operation described in Section 4.3, which enables a fixed-structure neural network to accommodate more samples, but this often comes at the cost of model accuracy. Another approach is to increase the number of hidden layers or the number of parameters in each layer, that is, to increase the number of independent variables in the equation set $L(1, -, \Phi_{\Delta})$, thereby accommodating more samples without sacrificing accuracy, at the cost of an increased training time.

5.3 Adding noise

During the training process of neural networks, we often enhance their robustness by adding noise to the existing samples and re-feeding them into the neural network for training [**?**, **?**]. This is because we have observed that after adding noise, even when humans do not perceive much difference between the previous and subsequent samples, the prediction accuracy of the machine drops sharply. This phenomenon can be explained by the concept of neighborhood. Let the initial sample be $A = (x, y)$ where $x = (x_1, x_2, \dots, x_m)^T$, and the noisy sample be $A_{\Delta} = (x_{\Delta}, y_{\Delta})$ where $x_{\Delta} = (x_1 + \Delta x_1, x_2 + \Delta x_2, \dots, x_m + \Delta x_m)^T$, then:

$$D_s(A, A_{\Delta}) = \sqrt{\frac{1}{2 \cdot \dim(x)} \sum_{j=1}^{\dim(x)} (\Delta x_j)^2}$$

The noisy sample may significantly deviate from the neighborhood of the original sample. If it is not within the neighborhood of other same-essence samples either, then the neural network will be unable to correctly process this sample, that is, $y_{\Delta} \neq y$. If there are too many noisy samples, it is difficult for the model to converge because we can add random noise. This is why we call the input vector of a neural network a "surface" , as there is a significant difference between what a neural network perceives and what humans see.

5.4 Shallow/Deep Networks

From the discussions in Sections 3 and 4, it can be concluded that the number of samples that a neural network can precisely fit is mainly positively correlated with the total number of network parameters, and has no necessary relationship with the depth of the network structure. If the number of samples is limited, we can directly adopt a network structure with only one hidden layer. According to the condition that homogeneous linear equations have a general solution, the number of parameters of a single-hidden-layer network should be greater than the product of the sample number, the surface dimension, and the essence

types. If the number of samples is large and can be dynamically increased, we can adopt a "tilted trapezoidal" network structure, in which the parameters of the last hidden layer are the most, and then the number of parameters decreases successively towards the first hidden layer. That is, in the EI algorithm, the calculation of invalid equation sets is minimized as much as possible.

5.5 Probability

The traditional view holds that the output layer of a neural network provides the probability that a surface of the input layer belongs to different essences. We believe this view is not entirely accurate, at least not in the strictly statistical sense of probability. In statistics, the probability of a random event is defined as the ratio of the certain output to the total output. No matter how large our training sample set is, we cannot exhaust or nearly exhaust the entire sample space, and there is no clear specific relationship between the finite sample set and the infinite sample space. For instance, we can add various noises to the existing samples, and the sample set can be easily expanded several times or even infinitely. Additionally, as shown in Figure 8 [Figure 8: see original paper], the training sample set does not necessarily occupy all the extreme points of the trained binary classification function $h_v^{[n]}(x)$. Those unoccupied maximum points are not necessarily occupied by the v-th essence samples, and the minimum points are not necessarily occupied by non-v-th essence samples. In extreme cases, even if there is a sample that makes $h_v^{[n]}(x) = 1$ hold true, it may still be a non-v-th essence sample, although this situation is rare.

6.1 Polarization

Apart from enumeration, we have not yet proposed an efficient algorithm to find the particular solutions that meet the requirements from the general solutions. This is the key to whether the EI algorithm can be practically applied. Can we draw on existing mature machine learning algorithms, such as clustering and gradient descent, to solve the efficiency problem of the algorithm?

6.2 The Output Layer

For the convenience of calculation and demonstration, we adopted the sigmoid function as the processing unit of the output layer. Although we believe that the selection of the function does not affect the overall characteristics of neural networks, what would be the difference in the final result if other functions were used, such as the commonly used softmax function?

6.3 Activation Functions

Our analysis is based on the case where the neural network is a continuous function, that is, the hidden layer neurons use a continuous sigmoid function. If

other functions are adopted, especially non-differentiable functions such as the ReLu function, how should the analysis be conducted?

6.4 Saddle Points

Our overall discussion is conducted under the assumption that a sample satisfying the system of equations $\{\frac{\partial h_v^{[n]}(x)}{\partial x_t} = 0 | t \in [1, m]\}$ are the extreme points of the binary classification function. For multivariate functions, the fact that all first-order partial derivatives are zero does not necessarily imply that this is an extreme point of the function. It could be a saddle point. Although we believe that neither the extreme points nor the saddle points significantly affect our conclusion, this remains a topic worthy of discussion.

6.5 Alternative Functions

From our analysis, it can be seen that the strong generalization ability of neural networks depends on the dynamic variability of their function curves, especially the dynamic adjustment of extreme points. Then, can other functions with similar properties provide equally strong generalization ability? For example, the sine function has infinitely many extreme points, and its range is limited to a finite interval. The number of extreme points of a polynomial is positively correlated with its degree. These two seemingly simple functions may have unexpected generalization ability.

7 Summary

We used the EI algorithm from a mathematical perspective to explain the specific reasons why neural networks have strong generalization ability, supplementing the shortcomings in the works of Cybenko (1989) and Hornik et al. (1989). We also present an algorithmic framework that is different from the BP algorithm, although there is currently no efficient computational method for the polarization. Compared with the mature BP algorithm, there are still many follow-up tasks to be improved in the EI algorithm. Taking advantage of each other' s strengths is expected to bring a new research idea to the field of artificial intelligence.

References

V. Buhrmester, D. Münch, and M. Arens. Analysis of explainers of black box deep neural networks for computer vision: A survey. *Machine Learning and Knowledge Extraction*, 3(4):966–989, 2021.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2021.

Z. C. Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

S. J. Oh, B. Schiele, and M. Fritz. Towards reverse-engineering black-box neural networks. *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 121–144, 2019.

M. T. Ribeiro, S. Singh, and C. Guestrin. "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*, pages 1135–1144, 2016.

C. F. G. D. Santos and J. P. Papa. Avoiding overfitting: A survey on regularization methods for convolutional neural networks. *ACM Computing Surveys (Csur)*, 54(10s):1–25, 2022.

S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How does batch normalization help optimization? *Advances in Neural Information Processing Systems*, 31, 2018.

R. Shwartz-Ziv and N. Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

N. Tishby and N. Zaslavsky. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2015.

J. Wu, S. Yang, R. Zhan, Y. Yuan, L. S. Chao, and D. F. Wong. A survey on LLM-generated text detection: Necessity, methods, and future directions. *Computational Linguistics*, pages 1–66, 2025.

W. Xia, Y. Zhang, Y. Yang, J.-H. Xue, B. Zhou, and M.-H. Yang. GAN inversion: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3121–3138, 2022.

X. Ying. An overview of overfitting and its solutions. In *Journal of Physics: Conference Series*, volume 1168, page 022022. IOP Publishing, 2019.

Y. Yu, X. Si, C. Hu, and J. Zhang. A review of recurrent neural networks: LSTM cells and network architectures. *Neural Computation*, 31(7):1235–1270, 2019.

Note: Figure translations are in progress. See original paper for figures.