Unraveling the Black Box of Neural Networks: A Dynamic Extremum Mapper

Authors: Shengjian Chen, Shengjian Chen

Date: 2025-10-13T00:00:00+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 extrema of the model function. We further prove that the number of extrema in a neural network is positively correlated with the number of its parameters. We then propose a new algorithm that is significantly different from back-propagation algorithm, which mainly obtains the values of parameters by solving a system of linear equations. Some difficult situations, such as gradient vanishing and overfitting, can be simply explained and dealt with in this framework.

Full Text

Preamble

Unraveling the Black Box of Neural Networks:

A Dynamic Extremum Mapper

Shengjian Chen
Intelligent Robotics Center, Jihua Laboratory
Foshan, 528200, China
chshengj@alumni.sysu.edu.cn, chensj@jihualab.ac.cn

Abstract

We argue that neural networks are not black boxes, and that their generalization capability stems from the ability to dynamically map a dataset to the extrema of the model function. We further prove that the number of extrema in a neural network is positively correlated with the number of its parameters. We then propose a novel algorithm that differs significantly from back-propagation,

which primarily obtains parameter values by solving a system of linear equations. Within this framework, challenging issues such as gradient vanishing and overfitting can be explained and addressed in a straightforward manner.

Keywords: neural network, generalization, black box, extreme increment, homogeneous system of linear equations, large language model

1 Introduction

Although artificial intelligence models based on neural networks have been extensively studied and widely applied—achieving prediction accuracy in image recognition, natural language processing, text processing, and question answering that far surpasses traditional machine learning algorithms—research on their underlying principles remains limited, and they are still generally regarded as black boxes. As model parameters have grown rapidly, from ANN to CNN, RNN, and now to GPT and LLM [**?**, **?**], system complexity has increased sharply while stability has become more fragile. Without understanding the internal logic, we cannot quickly identify root causes and solutions when models malfunction. While neural network algorithms can be confidently deployed in domains with low real-time requirements, such as image classification and AI-generated artwork, fields demanding high real-time performance and safety —particularly autonomous driving [**?**, **?**]—require deeper attention to the fundamental principles of neural networks and clear elucidation of the conditions under which they succeed or fail, so that artificial intelligence can better serve human society.

Despite the prohibitive complexity of modern neural network architectures, scholars continue striving to explore their working principles. Buhrmester et al. [**?**, **?**] investigated recently popular explanation methods that attempt to interpret neural networks by analyzing input-output connections. Black-box explainers, characterized by their ability to reveal model interaction details without accessing internal structure, are divided into ante-hoc systems with global, model-agnostic features [**?**, **?**] and post-hoc systems with local, model-specific features [**?**, **?**]. Oh et al. [**?**, **?**] analyzed neural networks through reverse engineering, finding them extremely vulnerable to various attacks and noting that the boundary between white-box and black-box models is not clearly defined. Tishby and Zaslavsky [**?**, **?**] proposed the Information Plane, arguing that neural networks primarily optimize the Information Bottleneck between compression and prediction at each layer, with Shwartz-Ziv and Tishby [**?**, **?**] later proving its effectiveness. While these works provide valuable references for foundational neural network research, significant gaps remain.

Current researchers appear overly focused on engineering approaches to explain neural network behavior, neglecting theoretical or mathematical perspectives. Neural networks may seem complex, but their clear structure—composed essentially of identical neurons—makes them particularly amenable to mathematical analysis. It is necessary to revisit the pioneering work of Cybenko [**?**, **?**] and

Hornik et al. [**?**, **?**], who proved that feedforward neural networks can approximate any continuous function on a compact set. Specifically, a feedforward neural network with a single hidden layer containing sufficient neurons and using the sigmoid activation function can approximate any complex function with arbitrary accuracy, providing a fundamental mathematical principle for neural networks. The limitation of this work is its lack of a method for finding the specific function for a given dataset or determining whether that function is optimal. Our work addresses these deficiencies.

Specifically, our contributions include: (1) We present the main characteristics of an ideal machine learning model and derive general model training steps, discussed primarily in Section 2. (2) We examine whether neural networks satisfy these ideal characteristics, demonstrating mathematically that neural networks achieve generalization primarily by mapping a dataset to local extrema of the function. We introduce a novel training algorithm distinct from back-propagation—the extremum-increment (EI) algorithm—discussed in Sections 3 and 4. (3) Based on the EI algorithm, we can readily explain the causes of common problems such as vanishing/exploding gradients and overfitting, and provide corresponding solutions, discussed in Section 5.

2 General Characteristics of an Ideal Model

Let us temporarily set aside the concept of neural networks and consider the fundamental characteristics a model should possess to satisfy a dataset and target task. The goal of machine learning training is to obtain a function curve that precisely fits all sample inputs to their corresponding outputs. That is, the model should clearly state the exact value for each input sample rather than providing vague probabilistic answers. For classification problems, the model should output "This is a cat" rather than "This is very likely a cat."

2.1 Precise Mapping

Situations without identical-type samples. For visualization purposes, we limit the sample size to three in this discussion. As shown in Figure 1 [Figure 1: see original paper], let the dataset be $D = \{(x^{(i)}, y^{(i)}) | i \in [1, 3]\}$, where $(x^{(i)}, y^{(i)})$ represents the i-th sample, $x^{(i)}$ is the surface representation of the sample, and $y^{(i)}$ is the essence (label) 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 conventional concepts of feature and label, instead using "surface" and "essence" to refer to $x^{(i)}$ and $y^{(i)}$, respectively. To grasp the core problem and simplify it, both surface and essence in Section 2 are represented as scalars. The function F shown in Figure 1 is the ideal model we seek, as it can precisely map any surface $x^{(i)}$ to its corresponding essence $y^{(i)}$.

Situations with identical-type samples. As shown in Figure 2 [Figure 2: see original paper], if a new sample essentially identical to one in dataset D is added

—for instance, sample $(x^{(3,1)}, y^{(3)})$—the function curve F must change shape so the new sample falls exactly on the curve. At this point, a local maximum appears on the function curve between samples $(x^{(3,1)}, y^{(3)})$ and $(x^{(3)}, y^{(3)})$.

Similarly, as shown in Figure 3 [Figure 3: see original paper], if additional samples with essence $y^{(3)}$—such as $(x^{(3,2)}, y^{(3)})$, $(x^{(3,3)}, y^{(3)})$, and $(x^{(3,4)}, y^{(3)})$—are continuously added, the function curve must further deform to accommodate these samples, forming multiple local minima and 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

Obtaining the aforementioned ideal function typically requires enormous computation. For a function with limited parameters, the degree of curve shape change is constrained, and the number of extreme values cannot increase arbitrarily. How then should we handle situations where a sample' s surface changes slightly while its essence remains unchanged? A natural approach is to expand the essence from a single point to an interval, allowing samples with slightly different surfaces but identical essence to fall within 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 sufficiently small. We adjust the precise mapping function F to the approximate fitting function F^*, making the difference between $F^*(x^{(3,5)})$ and $F^*(x^{(3)})$ as small as possible. When $|F^*(x^{(3,5)}) - F^*(x^{(3)})|$ is sufficiently small, we can approximately consider that surfaces falling within the interval $[F^*(x^{(3,5)}), F^*(x^{(3)})]$ all have essence $y^{(3)}$. 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 typically a one-dimensional vector or 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 element ranges from 0 to 255—the surface becomes fixed and cannot be further modified. However, the essence differs; it is usually an abstract concept that can be represented by any scalar or vector. As shown in Figure 4, if the curve shape of function F is restricted, each essence requires a tolerance interval. How is this interval selected? One method divides the range of function F into N equal-length intervals, where N is the total number of essence types. Each interval is assigned to an essence, and surfaces falling within the same interval share the same essence. When dividing essences using this method, the objective function' s value range should be finite.

2.3 From N-Classification to Binary Classification

The interval partition method faces a problem: when many essence types exist and the function F' s value range is limited to a small interval—for example, when each element in a neural network' s output layer is constrained to $(0, 1)$

–overlap becomes likely. One solution reduces essence types to expand each partition, but this introduces two new questions: to what extent should essence types be reduced, and how should excluded essences be handled?

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

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

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 function F_j, and LB^* and UB^* are the lower and upper limits of 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 binary classification function F_3. We adjust F_3' s parameters so all third-essence samples reach the function' s upper limit, while all other-essence samples reach the lower limit. Correspondingly, the weakened function F_3^* uses the midpoint of the value range as the dividing line. The same parameter adjustment applies to 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 follows: (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 parameters so each training surface $x^{(i)}$ becomes exactly one of the function' s extrema. (3) For each F_j, adjust parameters so training samples of the j-th essence become local maxima, and non-j-th essence samples become local minima. (4) Adjust parameters to make local maxima become global maxima and local minima become global minima.

A model trained through these steps can accurately map inputs to outputs, enabling the machine to precisely answer “yes”or “no.”In Section 3, we demonstrate that neural networks can be decomposed into binary classification functions, with corresponding surfaces mapped to local extrema by finding the general

solution of a homogeneous system of linear equations, thereby establishing the validity of Steps 1 and 2. In Section 4, we present a method for mapping surfaces to global extrema by enumerating particular solutions of the homogeneous system, thus confirming Steps 3 and 4.

3 Model Decomposition and Extreme Points

3.1 Model Decomposition

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

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

Figure 6 [Figure 6: see original paper] schematically illustrates a fully connected neural network structure expressed through numerical relationships rather than graphical representation. Each sample is denoted as (x, y), where the surface x is an m-dimensional column vector $x = (x_1, x_2, \dots, x_m)^T$, and $y \in [1, l_n]$ is the essence corresponding to x, with l_n representing the number of elements in the neural network' s output vector (the number of essence types).

The neural network has n layers with identical processing methods, where the first $n-1$ layers are hidden layers and the n-th layer is the output layer. The total number of elements in the u-th layer (with the input vector as the 0-th layer) is denoted l_u, and the v-th element in the u-th layer is denoted $h_v^{[u]}(x)$, where $v \in [1, l_u]$. As Figure 6 shows, disregarding the complex neuron connections, a neural network is actually a set 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 neural network' s target output vector is $(0, \dots, h_v^{[n]}(x) = 1, \dots, 0)^T$. For all other essence types, the target output vector is $(\omega, \dots, h_v^{[n]}(x) = 0, \dots, \omega)^T$, where one ω is 1 and the others are 0. In this simplified model, when the sigmoid function serves as the output, the upper and lower limits of $h_v^{[n]}(x)$ can only approach 1 and 0 asymptotically. When we transform the output layer from an l_n-dimensional

vector into l_n scalars, the entire model becomes clear: each composite function $h_v^{[n]}(x)$ is actually a binary classification problem for the v-th essence.

Therefore, a neural network with a multi-dimensional output vector can 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 neural network' s characteristics.

3.2 Extreme Points of the Model

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

$$h_v^{[u]}(x) = S\left(\sum_{k=1}^{l_{u-1}} w_{v,k}^{[u]} \cdot h_k^{[u-1]}(x)\right), \quad u > 1$$

$$h_v^{[1]}(x) = S\left(\sum_{k=1}^{m} w_{v,k}^{[1]} \cdot x_k\right), \quad u = 1$$

where $S(\theta) = \frac{1}{1+e^{-\theta}}$ is the sigmoid function, and $w_{v,k}^{[u]}$ represents the parameters between the $(u-1)$-th and u-th layers, identical to traditional neural network parameters. For enhanced readability, let $z_v^{[u]}(x) = \sum_{k=1}^{l_{u-1}} w_{v,k}^{[u]} \cdot h_k^{[u-1]}(x)$. Taking the partial derivative of $h_v^{[u]}(x)$:

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

where $t \in [1, m]$, using the derivative property $S'(\theta) = S(\theta) \cdot (1 - S(\theta))$. Let $c_v^{[u]}(x) = S(z_v^{[u]}(x)) \cdot (1 - S(z_v^{[u]}(x)))$, then:

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

For extreme points, $\frac{\partial h_v^{[u]}(x)}{\partial x_t} = 0$. Since $c_v^{[u]}(x) > 0$, we have $\sum_{k=1}^{l_{u-1}} w_{v,k}^{[u]} \cdot \frac{\partial h_k^{[u-1]}(x)}{\partial x_t} = 0$. Starting from the output layer $(u = n)$ and taking partial derivatives of all components of x, we obtain the system:

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

When a surface x is given, this system constitutes a homogeneous linear system of m equations with 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 coefficient matrix rank be $r(n,v)$. When $r(n,v)<l_{n-1}$, the system has infinitely many solutions. Since $r(n,v)\leq m$, as long as the last hidden layer' s neuron count l_{n-1} exceeds m during network design, we can always find infinitely many parameter combinations making surface x an extremum point of binary classification function $h_v^{[n]}(x)$ for any $v\in[1,l_n]$. This is the primary reason for neural networks' strong generalization ability, and where the black box begins to be unveiled.

The curves of 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, $L(n,-)$ forms a homogeneous linear system of $m\cdot l_n$ equations with $l_n\cdot l_{n-1}$ 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 surface x an extremum point of each binary classification function. We can then select a particular solution where, when surface x belongs to the v-th essence, the corresponding extremum is a maximum, and when x belongs to other essences, the extremum is a minimum. That is, $h_v^{[n]}(x)$ satisfies the ideal termination condition:

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

If finding this particular solution proves difficult, constraints can be relaxed by adopting a weakened termination condition:

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

3.3 Continuous Optimization of Parameter Combinations

The above discussion covers only the case with a single training sample. When sample count increases, let:

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

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 training the neural network with dataset $\Phi = \{(x^{(i)}, y^{(i)}) | i \in [1,\phi]\}$, we solve the homogeneous linear 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, a particular solution meeting the conditions can be found from the general solution. Otherwise, parameters between the $(n-2)$-th and $(n-1)$-th layers must be introduced, requiring expansion of partial derivatives in system $L(n,v)$. Substituting $\frac{\partial h_k^{[n-1]}(x)}{\partial x_t} = c_k^{[n-1]}(x) \cdot \sum_{p=1}^{l_{n-2}} w_{k,p}^{[n-1]} \cdot \frac{\partial h_p^{[n-2]}(x)}{\partial x_t}$ into $L(n,v)$ and simplifying yields:

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

Similarly, we obtain system $L(n-1,-)$, which can be viewed as a homogeneous nonlinear system 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 $L(n-1,-)$ appears nonlinear, its regular structure allows treating $w_{v,k}^{[n]} \cdot w_{k,p}^{[n-1]}$ as a whole, enabling solution methods for homogeneous linear equations. We then find the particular solution of system $L(n-1,-,\Phi)$ that meets requirements. By solving homogeneous equations layer by layer, the dataset can be mapped to the neural network.

4 General Training Method

4.1 The EI Algorithm Framework

From the above discussion, we obtain a preliminary model training framework called the EI algorithm, whose main steps differ significantly from current methods like the BP algorithm. First, the BP algorithm uses gradient updates to approximate ideal parameter values, while the EI algorithm attempts to directly obtain parameter values by solving equation systems. Second, the BP algorithm updates all parameters each iteration, whereas the EI algorithm updates only some parameters. Debugging all training samples to the model' s extremum points is the framework' s key. This subsection discusses algorithm details more deeply.

Table 1 shows neural network parameter states under the 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 parameters remain at initial values, and "update" indicates parameters are updated in the current round. In the first round, we solve system $L(n, -, \Phi)$. If a solution exists, only parameters $W^{[n]}$ need updating. Otherwise, in the second round, we solve $L(n-1, -, \Phi)$. If a solution exists, parameters $W^{[n]}$ and $W^{[n-1]}$ require updating. Otherwise, we solve $L(n-2, -, \Phi)$, and so on.

Algorithm 4.1 presents the main steps for precisely mapping a dataset to the neural network model. Symbols retain their previous meanings unless otherwise specified. Initially, we manually label sample set Φ. If sample $(x^{(i)}, y^{(i)})$ belongs to the j-th essence where $j \in [1, l_n]$, then $(x^{(i)}, y^{(i)}) = (x^{(i)}, j)$. We then initialize parameter set W to non-zero real numbers.

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

```
function FittingCurve()
    Init(W)
    for u  [1, n-1], v  [1, l_u], t  [1, m], i  [1, $\phi$] do
        Calculate( h^{[u]}_v(x)/ x_t|_{x=x^{(i)}})
    end for
    for u  [1, n-1], v  [1, l_u], i  [1, $\phi$] do
        Calculate(h^{[u]}_v(x^{(i)}))
    end for
    u $\leftarrow$ n
    while u $\geq$ 1 do
        W^{[u:n]} $\leftarrow$ L(u, -, Φ)  // the general solution, equals {W^{[j]}|j  [u,
        W_{[u:n]} $\leftarrow$ Polarize({h^{[n]}_v(x)|v  [1, l_n]}, W^{[u:n]}, Φ)  // the p
        if W^{[u:n]} $\neq$ {}  W^{[u:n]} $\neq$ {0} then
            W $\leftarrow$ Update(W_{[u:n]})
            break
        end if
```

```
        u $\leftarrow$ u - 1
    end while
    if u $\geq$ 1 then
        return W
    else
        return Error()
    end if
end function
```

Like the BP algorithm, parameter updates execute layer by layer from the last hidden layer to the first. We first calculate values and partial derivatives of all hidden layer neurons, then solve for the general solution $W^{[u:n]} = \{W^{[j]} | j \in [u, n]\}$ of system $L(u, -, \Phi)$. Next, we select a particular solution $W_{[u:n]}$ satisfying the termination condition from $W^{[u:n]}$—an operation we call "polarize." If a particular solution $W_{[u:n]}$ is found, parameters W are updated. If no particular solution is found, it indicates that parameter set $W^{[u:n]}$ cannot precisely map sample set Φ to the neural network, requiring introduction of parameters $W^{[u-1]}$ to find particular solution $W_{[u-1:n]}$ of system $L(u-1, -, \Phi)$. If no particular solution meeting requirements is found after traversing all parameters, we must consider adjusting the network structure (e.g., increasing hidden layers or nodes per layer).

4.2 Reducing Computational Complexity

In Algorithm 4.1, the polarization time for selecting particular solution $W_{[u:n]}$ from general solution $W^{[u:n]}$ is uncertain because we do not know the particular solution's characteristics and must verify each general solution instance through enumeration. To reduce training time, we can relax the model training termination condition. That is, when sample $(x^{(i)}, y^{(i)})$ belongs to the v-th essence, the v-th binary classification function value $h_v^{[n]}(x^{(i)})$ need only be much larger than any other binary classification function value $h_q^{[n]}(x^{(i)})$, without requiring these values to be maxima or minima. This is equivalent to the weakened condition:

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

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

where α and β are sufficiently small positive real numbers. This corresponds to using the softmax function as the output layer. Thus, a neural network with softmax output can be viewed as a weakened version of an ideal model.

4.3 Reducing Computational Scale

If each training sample corresponds to an extreme point on the model curve –by adding equation set $L(n, -, x^{(i)})$–the required network parameter scale becomes extremely large and training time increases significantly. Can we 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 samples need to be extreme points; others need only satisfy the weakened termination condition. Which samples can have relaxed restrictions? An intuitive idea is that only representatives of adjacent samples need to satisfy strict conditions.

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

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

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

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

where $\dim(x)$ represents sample surface dimension and γ is a sufficiently small positive real number, then samples A and B of the same essence are located within each other' s neighborhood. Due to function continuity, sample A and B values are close on each binary classification function. Thus, one sample need not be sent to the training algorithm but only requires function value verification. A further weakened training algorithm adjusts as follows: (1) Manually classify the training sample set and designate it as the major category. (2) Use a numerical algorithm (e.g., clustering) to divide samples of each major category into minor categories, each with a central sample. (3) Train the model on all central samples. (4) After training, verify whether predicted values of all non-central samples on the neural network meet specified accuracy requirements. If requirements are satisfied, the algorithm ends; otherwise, mark non-central samples that fail verification as central samples and repeat steps 3 and 4.

5 Analysis of Common Problems

5.1 Gradient Vanishing/Explosion

Gradient vanishing/explosion is a common and challenging issue in neural network training, particularly in deep networks. Scholars have proposed various mitigation methods, such as batch normalization [**?**, **?**] and LSTM architecture [**?**, **?**]. In the BP algorithm, gradient vanishing/explosion is typically regarded as an abnormal issue to avoid.

Regarding gradient vanishing, as discussed in Sections 3 and 4, after network parameter initialization, the number of parameter updates required varies with sample size. If particular solution $W_{[u:n]}$ can be found from general solution $W^{[u:n]}$, parameters $W^{[1:u-1]}$ of earlier hidden layers can remain at initial values. According to the neural network characteristics revealed by the EI algorithm, gradient vanishing is an inevitable result. Gradient explosion is similar. In the EI algorithm, when solving system $L(1,-,\Phi)$, cases may exist where no solution exists—that is, the solution value is infinite, corresponding to gradient explosion in the BP algorithm. If the EI algorithm is adopted, simply increasing hidden layers or parameters per layer suffices.

5.2 Overfitting

The overfitting problem [**?**, **?**] appears different from gradient vanishing/explosion but is essentially caused by the same operational process. In the EI algorithm, if system $L(1,-,\Phi)$ has solutions, but when increasing samples, system $L(1,-,\Phi_\Delta)$ may have no solution where $\Phi \subset \Phi_\Delta$. That is, the neural network with current parameter scale can only accommodate a limited number of samples Φ, manifesting as the overfitting phenomenon in the BP algorithm. This is an inherent characteristic: neural networks have only a limited number of extreme values under parameter constraints. Rather than overfitting, we might say it fits just right.

The BP algorithm reduces model dependence on training samples by adding noise to samples and network parameters [**?**, **?**]. This method resembles the clustering operation in Section 4.3, enabling a fixed-structure neural network to accommodate more samples, but often at the cost of model accuracy. Another approach increases hidden layers or parameters per layer—that is, increases independent variables in system $L(1,-,\Phi_\Delta)$—thereby accommodating more samples without sacrificing accuracy, at the cost of increased training time.

5.3 Adding Noise

During neural network training, we often enhance robustness by adding noise to existing samples and retraining [**?**, **?**]. We observe that after adding noise, even when humans perceive minimal difference between original and modified samples, machine prediction accuracy drops sharply. This phenomenon can be explained by the neighborhood concept. Let initial sample be $A = (x, y)$ where $x = (x_1, x_2, \dots, x_m)^T$, and 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:

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

The noisy sample may significantly deviate from the original sample' s neighborhood. If it also falls outside neighborhoods of other same-essence samples,

the neural network cannot correctly process it ($y_{\Delta} \neq y$). With too many noisy samples, model convergence becomes difficult because we can add random noise arbitrarily. This is why we call the neural network input vector a "surface" — there is a significant difference between what a neural network perceives and what humans see.

5.4 Shallow vs. Deep Networks

From Sections 3 and 4, we conclude that the number of samples a neural network can precisely fit is primarily positively correlated with total network parameters, not necessarily with network depth. If sample count is limited, we can directly adopt a single-hidden-layer network structure. According to homogeneous linear equation solution conditions, a single-hidden-layer network' s parameters should exceed the product of sample number, surface dimension, and essence types. If sample count is large and can dynamically increase, we can adopt a "tilted trapezoidal" network structure where the last hidden layer has the most parameters, decreasing successively toward the first hidden layer. This minimizes calculation of invalid equation sets in the EI algorithm.

5.5 Probability

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

6 Discussion

6.1 Polarization

Beyond enumeration, we have not yet proposed an efficient algorithm to find a particular solution from a general solution. This is key to practical EI algorithm application. Polarization can be summarized as follows:

Problem 1: Given a set of binary classification functions $\{h_v^{[n]}(x)|v \in [1, l_n]\}$ with known local extreme points, how can we flip their values arbitrarily? For example, for parabolic function $y = a(x-b)^2$, changing parameter a' s sign flips

its extreme point, and increasing $|a|$ increases the extreme value. Furthermore, apart from $L(n, -, \Phi)$, systems $L(u, -, \Phi)$ with $u \in [1, n-1]$ are atypical homogeneous linear equations. Whether their unique structure facilitates obtaining particular solutions warrants further discussion.

6.2 The Output Layer

For calculation and demonstration convenience, we adopted the sigmoid function as the output layer processing unit. Although Section 4.2 demonstrated that using softmax as the output layer corresponds to a weakened model, complete partial differential analysis of the softmax function remains worth discussing.

6.3 Activation Functions

Our analysis assumes neural networks are continuous functions—that is, hidden layer neurons use continuous sigmoid functions. How should analysis proceed if other functions are adopted, particularly non-differentiable functions like ReLU?

6.4 Saddle Points

Our discussion assumes samples satisfying the system $\{\frac{\partial h_v^{[n]}(x)}{\partial x_t} = 0 | t \in [1, m]\}$ are extreme points of binary classification functions. For multivariate functions, zero first-order partial derivatives do not necessarily indicate extreme points; they could be saddle points. This is not problematic if polarization algorithms can find global maxima or minima, but otherwise remains a topic worthy of discussion.

6.5 Alternative Functions

Our analysis shows that neural networks' strong generalization ability depends on dynamic function curve variability, especially dynamic extreme point adjustment. Can other functions with similar properties provide equally strong generalization? For example, the sine function has infinitely many extreme points with range limited to a finite interval, and a polynomial' s extreme point count is positively correlated with its degree. These seemingly simple functions may possess unexpected generalization ability.

7 Summary

From a mathematical perspective, we explain why neural networks have strong generalization capabilities, supplementing the limitations in Cybenko [**?**, **?**] and Hornik et al. [**?**, **?**]. We also present the corresponding EI algorithm, which differs from the BP algorithm. Without an effective polarization method, the EI algorithm can at least serve as an important submodule of the BP algorithm—that is, we can first initialize parameters using an EI algorithm general

solution instance, then train using the BP algorithm. If an efficient polarization algorithm is found, the EI algorithm could become a strong BP algorithm competitor, particularly when the most ideal model is required.

Acknowledgments and Disclosure of Funding: I am extremely grateful to my supervisor, Professor Yang Lihua, who provided initial guidance and assistance on the basic research direction when I began my research journey. Without his guidance, I might have chosen a different path, missing this study. This research was funded by the Science and Technology Research Project of Key Areas in Nanhai District, Foshan City (Grant No. 2230032004637).

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. In *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.