SUPPLEMENTARY INFORMATION FOR:

Universal Forward Training and Structure-free Learning

This PDF file includes:

Supplementary Texts

This paper carries out a theoretical analysis and builds a theoretical system based on Hypothesis 1-3. The Hypotheses used as the basis only need to be reasonable and applicable and do not necessarily need to be proved. However, the whole mathematical calculus process can deepen the understanding of the relationship and meaning of each hypothesis.

Hypothesis 1

Hypothesis 1: For a multi-input-single-output (MISO) function with continuous derivatives (smooth)

$$f: x \mapsto y = f(x), x = [x^1, x^2, \dots, x^P],$$

(a) (Existentiality): For any certain data point (x_0, y_0), the function has the local linearization:

$$y_0 = f(x_0) = C_0 + Cx_0^T, \text{ where } \mathrm{C} = [\mathrm{C}_1, \dots, \mathrm{C}_\mathrm{P}].$$

(b) (Uniqueness): If C_p is the corresponding partial derivative of the output y_0 to the input x_0^p, then C is unique.

(c) (Approximation): In the sufficiently small neighbor of any certain data point (x_0, y_0), any point (x, y) can be approximated as:

$$y = f(x) \approx C_0 + Cx^T,$$

where C and C_0 is determined by (x_0, y_0) (that is $y_0 = f(x_0) = C_0 + Cx_0^T$ in Hypothesis 1(a)(b)).

(d) (Continuity): C and C_0 are also the continuous function of input x.

Proof:

(a)-(b) (Existentiality and Uniqueness):

For a multi-input-single-output (MISO) function

$$f: x \mapsto y = f(x), x = [x^1, x^2, \dots, x^P],$$

with continuous derivatives is a continuous and derivable function.

If the mapping function is continuous and derivable, then $\frac{\partial y}{\partial x^p}$ is existent and unique.

For any certain data point (x_0, y_0), define

$$C_p \triangleq \frac{\partial y}{\partial x^p}|_{x=x_0}$$

and

$$C_0 = y_0 - \sum_{p=1}^{\mathrm{P}} C_p {x_0}^p$$

So

$$y_0 = f(x_0) = C_0 + \sum_{p=1}^{\mathrm{P}} C_p {x_0}^p = C_0 + C x_0^T$$

where, $C = [C_1, \dots, C_{\mathrm{P}}]$.

According to the existence and uniqueness of C_p, **Hypothesis 1**(a) and (b) can be proved.

(c) (Approximation):

The proof of **Hypothesis 1**(c) is obviously easy to be accepted because of continuity and "substituting curve with straight".

According to the "substituting curve with straight", for a smooth "curve", we can approximate it with "straight". For any data point (x, y) in the sufficiently small neighbor of a certain data point (x_0, y_0), that is

$$y = f(x) \approx C_0' + C' x^T.$$

In the neighbor area of (x_0, y_0), all the data satisfy it, including(x_0, y_0). And

$$y_0 = f(x_0) = C_0 + C x_0^T,$$

by Hypothesis 1(a).

So $C_0' = C_0$, $C' = C$ is a reasonable solution (The derivation cannot guarantee the uniqueness of the solution).

According to the continuity of $f(x)$ and $g(x) = C_0 + C x^T$,

$\lim_{x \to x_0} f(x) - g(x) = f(x_0) - g(x_0) = 0$.

Then, for a sufficiently small $\varepsilon > 0$ and $\varepsilon \to 0$,

$\exists \delta > 0$, when $|x - x_0| < \delta$ then $|f(x) - g(x)| < \varepsilon$,

So

$$y = f(x) \approx g(x) = C_0 + Cx^T.$$

(d) (Continuity):

(1) C and C_0, as observable properties variables, are the function of input x, according to Causality and Determinism of **Hypothesis 2**.

(2) According to the continuity of derivatives (The premise of **Hypothesis 1**), C (**Hypothesis 1(b)**, Uniqueness, C_p is the corresponding partial derivative) is the continuous function of input x.

Because

$$C_0 = y - \sum_{p=1}^{P} C_p x^p$$

and the continuity of the function and C, C_0 is also the continuous function of input x.

Proven.

Extension of Hypothesis 1(d):

First, try to explain that C and C_0, as observable properties variables, are the function of input x. It is obvious because only the input can change other variables when the function is fixed, according to Causality and Determinism of **Hypothesis 2**. If not, in this situation, then C and C_0 are constant, and $y = C_0 + Cx^T$ just a linear function, which does not conform to the premise of any function. Thus, C and C_0 are the function of input x. The case of constant (linear function) can be regarded as a special case.

Secondly, for any data point (x', y') in the sufficiently small neighbor of a certain data point (x, y), and

$$\begin{cases} y = f(x) = C_0 + Cx^T = C_s[1; x^T] = C_s X \\ y' = f(x') = C_0' + C'x'^T = C_s'\left[1; x'^T\right] = C_s' X' \end{cases}$$

where $\mathrm{X} = [1; x]$, $\mathrm{X}' = \left[1; x'^T\right]$, $C_s = [C_0, C]$, $C_s' = [C_0', C']$

$$\begin{cases} C_s = X^{\dagger} y \\ C_s' = X'^{\dagger} y' \end{cases}$$

then according to the continuity of function f, it can be seen that

$$\lim_{x' \to x} y' = y, \ \lim_{x' \to x} X' = X$$

According to the continuity of the generalized inverse matrix (Ref. 29, 30), it can be obtained that:

$$\mathrm{rank}(X') = \mathrm{rank}(X) = 1,$$

then

$$\lim_{x' \to x} X'^{\dagger} = X^{\dagger}.$$

Thus

$$\lim_{x' \to x} C_s' = C_s$$

It indicates continuity.

At the same time, for sufficiently small $\Delta \mathrm{x} = x' - x$, $C_s' \approx C_s$, same as **Hypothesis 3(b).**

Hypothesis 2

Hypothesis 2 as follows is based on "causality and determinism" in the classical scientific view. Stable means that the output is fixed in the same case. Independence is to simplify the follow-up analysis.

Hypothesis 2: For a stable parametric system,

(a) (Causality and Determinism): Any output and observable properties (or characters) of any system are determined stably by the (i) input, (ii) system parameters (weights), and (iii) system structure (the sum of covert parameters, hidden variables, topology, and other influencing factors).

(b) (Independence and Causal Decoupling): The effect of (i) input, (ii) parameters, and (iii) structure could be considered independent of each other.

"Causality and Determinism" in Hypothesis 2 is a classical scientific view, not suitable for all systems like chaotic systems in an engineering sense. Similarly, independence is a simplified and idealized treatment.

By Hypothesis 2, the matrix $\mathbb{C}_{\boldsymbol{X}} = \aleph(\boldsymbol{X}, \boldsymbol{W}, \boldsymbol{SS}) = \aleph_{\boldsymbol{X}}(\boldsymbol{SS}, \boldsymbol{W}) = \aleph_{\boldsymbol{X},\boldsymbol{W}}(\boldsymbol{SS}) = \aleph_{\boldsymbol{X},\boldsymbol{SS}}(\boldsymbol{W})$. Thus $\mathbb{C}_{\boldsymbol{X}}$ could be an observable window to analyze the system structure and parameters, and a representative of the system. This is the ideological basis on $\mathbb{C}_{\boldsymbol{X}}$ of system design in the next section.

Hypothesis 3

Hypothesis 3: For two stable parametric systems processed by LL

$$\begin{cases} \boldsymbol{Y} = \mathbb{H}(\boldsymbol{W}) = \mathbb{C}_{\boldsymbol{X}}\boldsymbol{W} \\ \boldsymbol{Y}' = \mathbb{H}'(\boldsymbol{W}') = \mathbb{C}'_{\boldsymbol{X}}\boldsymbol{W}' \end{cases}$$

(a) (Invariability): The systems with the same structure and parameters ($\boldsymbol{W} = \boldsymbol{W}'$), for the same input vector $\boldsymbol{X}$ of the dataset, have the same output vector ($\boldsymbol{Y} = \boldsymbol{Y}'$) and the same constant matrix ($\mathbb{C}_{\boldsymbol{X}} = \mathbb{C}'_{\boldsymbol{X}}$).

(b) (Comparability): The systems with the same structure and **comparable** parameter ($\boldsymbol{W} \approx \boldsymbol{W}'$), for the same input vector $\boldsymbol{X}$ of the dataset, have the **comparable** constant matrix ($\mathbb{C}_{\boldsymbol{X}} \approx \mathbb{C}'_{\boldsymbol{X}}$), and $\lim_{\boldsymbol{W}' \to \boldsymbol{W}} \mathbb{C}'_{\boldsymbol{X}} = \mathbb{C}_{\boldsymbol{X}}$.

Proof:

(a) (Invariability):

Suppose a dataset contains N_d data $\{(\mathrm{x}_i, \mathrm{y}_i)|\ i = 1, \dots, N_d\}$, where output $y = [\ y^1, y^2, \dots, y^Q]$, input $x = [x^1, x^2, \dots, x^P\ ]$, and let $\boldsymbol{Y} = [y_1, y_2, \dots, y_{N_d}]^T{}_{(N_d \times Q)\times 1}$, $\mathbf{X} = [\mathrm{x}_1, \mathrm{x}_2, \dots, \mathrm{x}_{N_d}]^T{}_{N_d \times 1}$,

$\mathbb{C}_X = [C_{x_1}^{(1)}; C_{x_1}^{(2)}; \dots; C_{x_1}^{(Q)}; C_{x_2}^{(1)}; \dots; C_{x_{N_d}}^{(1)}; C_{x_{N_d}}^{(2)}; \dots; C_{x_{N_d}}^{(Q)}]_{(N_d \times Q)\times(S+1)}$, $\mathbf{W} = [1, W]^T{}_{(S+1)\times 1}$.

Because the systems with the same structure and parameters (weights) W, for the same input x of the dataset, have the same mapping function and output (by **Hypothesis 2**), that is,

$$y = F(x, W) = H(W)$$

So, the systems have the same output vector $\boldsymbol{Y} = [y_1, y_2, \dots, y_{N_d}]^T{}_{(N_d \times Q)\times 1}$.

For any data (x, y) in $\{(\mathrm{x}_i, \mathrm{y}_i)\}$, according to the existence and uniqueness of C (by **Hypothesis 1**), the existence and uniqueness of

$$[C_x^{(1)}; C_x^{(2)}; \dots; C_x^{(Q)}]_{Q\times(S+1)}$$

is correct, and

$$y = H(W) = \mathrm{C}_x \boldsymbol{W}$$

Make further efforts, for all the data in the dataset $\{(\mathrm{x}_i, \mathrm{y}_i)\}$, the existence and uniqueness of

$$\mathbb{C}_{\boldsymbol{X}} = [C_{x_1}^{(1)}; C_{x_1}^{(2)}; \dots; C_{x_1}^{(Q)}; C_{x_2}^{(1)}; \dots; C_{x_{N_d}}^{(1)}; C_{x_{N_d}}^{(2)}; \dots; C_{x_{N_d}}^{(Q)}]_{(N_d \times Q)\times(S+1)}$$

is correct. Thus the systems with the same structure and parameter (weights) $\boldsymbol{W}$, for the same input vector $\boldsymbol{X}$ of the dataset, have the same constant matrix $\mathbb{C}_{\boldsymbol{X}}$.

(b) (Comparability): The systems with the same structure and **comparable** parameter ($\boldsymbol{W} \approx \boldsymbol{W}'$), for the same input vector $\boldsymbol{X}$ of the dataset, have the **comparable** constant matrix ($\mathbb{C}_{\boldsymbol{X}} \approx \mathbb{C}'_{\boldsymbol{X}}$), and $\lim_{\boldsymbol{W}' \to \boldsymbol{W}} \mathbb{C}'_{\boldsymbol{X}} = \mathbb{C}_{\boldsymbol{X}}$.

Define: $|\boldsymbol{v}| \triangleq |v_1| + |v_2| + \cdots + |v_N|$, $\boldsymbol{v} = [v_1, v_2, \dots, v_N]$.

The weights can be considered to be the input of the current system, for the same $\boldsymbol{X}$ of the dataset. Continuity ("continuous derivable" of **Hypothesis 1**) is the important prerequisite for all discussions in this paper. According to the **Hypothesis 3**(a) and the **Hypothesis 1(d)**,

$$\lim_{\boldsymbol{W}' \to \boldsymbol{W}} \boldsymbol{Y}' = \boldsymbol{Y}, \ \lim_{\boldsymbol{W}' \to \boldsymbol{W}} \mathbb{C}'_X = \mathbb{C}_{\boldsymbol{X}}.$$

Thus, for sufficiently small $|\Delta \boldsymbol{W}| = |\boldsymbol{W} - \boldsymbol{W}'|$, $\mathbb{C}'_X \approx \mathbb{C}_X$. **Proven.**

Extension of Hypothesis 3(b)

Hypothesis 3(b) is obviously easy to accept because of continuity.

Define: $|\boldsymbol{v}| \triangleq |v_1| + |v_2| + \cdots + |v_N|,\ \boldsymbol{v} = [v_1, v_2, \ldots, v_N]$**.**

If $|\boldsymbol{W} - \boldsymbol{W}'| = |\Delta \boldsymbol{W}| = \varepsilon > 0,\ \varepsilon$ is sufficiently small.

then $|\boldsymbol{Y} - \boldsymbol{Y}'| = |\Delta \boldsymbol{Y}| = \delta,\ |\mathbb{C}_{\boldsymbol{X}} - \mathbb{C}'_X| = \theta$ and $\lim\limits_{\varepsilon \to 0} \delta = 0,\ \lim\limits_{\varepsilon \to 0} \theta = 0.$

So,

$$|\Delta \boldsymbol{Y}| = |\boldsymbol{Y} - \boldsymbol{Y}'| = |\mathbb{C}_{\boldsymbol{X}} \boldsymbol{W} - \mathbb{C}'_X \boldsymbol{W}'| = |\mathbb{C}_{\boldsymbol{X}} \boldsymbol{W} - \mathbb{C}'_X \boldsymbol{W}'| = |\mathbb{C}_{\boldsymbol{X}} \boldsymbol{W} - \mathbb{C}_{\boldsymbol{X}} \boldsymbol{W}' + \mathbb{C}_{\boldsymbol{X}} \boldsymbol{W}' - \mathbb{C}'_X \boldsymbol{W}'|$$
$$\leq |\mathbb{C}_{\boldsymbol{X}}||\Delta \boldsymbol{W}| + |\mathbb{C}_{\boldsymbol{X}} - \mathbb{C}'_X||\boldsymbol{W}'|$$

$$\Longrightarrow |\mathbb{C}_{\boldsymbol{X}} - \mathbb{C}'_X| \geq \frac{|\Delta \boldsymbol{Y}| - |\mathbb{C}_{\boldsymbol{X}}||\Delta \boldsymbol{W}|}{|\boldsymbol{W}'|}$$

The above inequality gives the lower limit of error $|\mathbb{C}_{\boldsymbol{X}} - \mathbb{C}'_X|$. Two phenomena were observed at the same time.

1) because $\lim\limits_{\boldsymbol{W}' \to \boldsymbol{W}} \boldsymbol{Y}' = \boldsymbol{Y}$, and $\lim\limits_{|\Delta \boldsymbol{W}| \to \boldsymbol{0}} |\Delta \boldsymbol{Y}| = \boldsymbol{0}$ **(Hypothesis 3**(a)**)**

$$\lim_{\substack{\boldsymbol{W}' \to \boldsymbol{W} \\ |\Delta \boldsymbol{W}| \to \boldsymbol{0}}} \frac{|\Delta \boldsymbol{Y}| - |\mathbb{C}_{\boldsymbol{X}}||\Delta \boldsymbol{W}|}{|\boldsymbol{W}'|} = \boldsymbol{0}$$

Therefore, $\Delta \boldsymbol{W}$ needs to be small enough in engineering implementation.

2) The smaller $|\boldsymbol{W}'|$ is, the larger the lower limit of error $|\mathbb{C}_{\boldsymbol{X}} - \mathbb{C}'_X|$ is.

So the system weight should not be too small.

End

System Matrixization

For a high-dimensional parameterized system with multi-input-multi-output (MIMO) mapping function, any output y or observable properties (P or characters C) is the function of the input x, the system parameter (weights) W and system structure (SS) in **Hypothesis 2,** labeled as $y = \aleph(x, W, SS)$ or $P = \aleph(x, W, SS)$. $\aleph$ means a high-dimensional complex mapping space.

For a high-dimensional parameterized system with multi-input-multi-output (MIMO) mapping function, any output y is the function of the input x and the system parameter (weights) W, in **Hypothesis 2,** that is

$$y = F(x, W) = [f^1(x, W), f^2(x, W), \dots, f^Q(x, W)]$$

where

output $y = [\ y^1, y^2, \dots, y^Q]$,

input $x = [x^1, x^2, \dots, x^P]$,

weights $\mathrm{W} = [w_1, w_2, \dots, w_S]$,

P, Q, and S are the dimensions of input, output, and weights, respectively,

and $f^\alpha = \pi^\alpha \circ F$ is the αth component function of F.

According to the function linearization LL discussed in **Hypothesis 1**,

$$f^\alpha(x, W) = C_0^\alpha + C^\alpha[x, W]^T = C^{(\alpha)}[1, x, W]^T,$$

then

$$y^T = C_S[1, x, W]^T,$$

where $C_S = [C^{(1)}; C^{(2)}; \dots; C^{(Q)}]$.

Obviously, in the case of x and W fixed, C_S is only related to the structure of the system. This processing also means a new approximate description method of complex mapping space's local characteristics by a hyperplane (determined by C_S). On the other hand, C_S varies with x and W. Therefore, C_S can only describe the local system characteristics of current x and W. In order to describe the global characteristics of the system more comprehensively, more different x and W need to be added. In engineering applications, x and W usually do not seek to change at the same time, so the changes of x and W are discussed respectively.

If the input of the system remains fixed, the output can be changed into a function of the system parameters:

$$y = H(W) = [h^1(W), h^2(W), \dots, h^Q(W)] = C_x[1, \mathrm{W}]^T$$

where $C_x = [C_x^{(1)}, C_x^{(2)}, \dots, C_x^{(Q)}]$. The linearization of the function h^{α} is realized in the neighborhood of W for the data point (x, y^{α}), $y^{\alpha} = C_x^{(\alpha)}[1, W]^T$. If C_x can be obtained, the local spatial distribution characteristics for the current W in (x, y) can be obtained, as shown in **Hypothesis 1(c)**. Of course, only one data point is considered, which obviously cannot describe the whole plant space globally, so more data points need to be added.

Suppose a dataset $(X, Y) = \{(x_i, y_i) |\ i = 1, \dots, N_d\}$ contains N_d data, and let

$$\boldsymbol{Y} = [y_1, y_2, \dots, y_{N_d}]^T{}_{(N_d \times Q) \times 1},$$

$$\mathbf{X} = [\mathrm{x}_1, \mathrm{x}_2, \dots, \mathrm{x}_{N_d}]^T{}_{N_d \times 1},$$

$$\mathbb{C}_X = [C_{x_1}^{(1)}; C_{x_1}^{(2)}; \dots; C_{x_1}^{(Q)}; C_{x_2}^{(1)}; \dots; C_{x_{N_d}}^{(1)}; C_{x_{N_d}}^{(2)}; \dots; C_{x_{N_d}}^{(Q)}]_{(N_d \times Q) \times (S+1)},$$

$$\boldsymbol{W} = [1, W]^T,$$

then

$$\boldsymbol{Y} = \mathbb{H}(\boldsymbol{W}) = \mathbb{C}_X \boldsymbol{W}.$$

$\mathbb{C}_X$ (Constant matrix or Character matrix) is a character variables matrix of function, remains constant for the fixed $\boldsymbol{X}$. On the contrary, it will change with the change of $\boldsymbol{X}$. $\mathbb{C}$ has a clear mathematical meaning, and each element is a corresponding partial derivative of the output to the input, as shown in **Hypothesis 1(b)**. The global characteristics of the function are obtained by all local characteristics of the fitted hyperplanes for all the data of the dataset, when the dataset is large enough, due to **Hypothesis 1(c)**. Thus, the effect, accuracy, or granularity described by the constant matrix is related to the size and distribution of the dataset. Constant matrix, as a novel type of function description method, is different from the traditional mathematical analytic formula, drawing, modeling, or black box method.

Similarly obtains

$$y = \aleph(x, W, SS) = \aleph_{SS}(x, W) = \aleph_{W,SS}(x),$$

$$P = \aleph(x, W, SS) = \aleph_x(SS, W) = \aleph_{x,W}(SS),$$

or other functions and their corresponding matrix form, respectively, like

$$\boldsymbol{Y} = \mathbb{G}(\boldsymbol{X}) = \mathbb{C}_W \boldsymbol{X},$$

$$\text{or } \mathbf{Y} = \mathbb{F}(\boldsymbol{W}, \boldsymbol{X}) \approx \mathbb{C}_S[\boldsymbol{W}; \boldsymbol{X}],$$

which have obviously different physical meanings and application prospects. In spite of these, they have the same mathematical form. Hence $\mathbb{C}_X$, $\mathbb{C}_S$ or $\mathbb{C}_W$ are collectively labeled by $\mathbb{C}$ (constant matrix), and the corresponding functional relationship are collectively labeled by

$$\boldsymbol{Y} = \mathbb{F}(\boldsymbol{X}) = \mathbb{C}\boldsymbol{X}.$$

In the paper, $\boldsymbol{Y} = \mathbb{H}(\boldsymbol{W}) = \mathbb{C}_X\boldsymbol{W}$ is taken into consideration to illustrate the solution and application method of the constant matrix.

As a result of local linearization processing, for a function $y = f(x)$, $\boldsymbol{Y} = \mathbb{F}(\boldsymbol{X}) = \mathbb{C}\boldsymbol{X}$ in the case of a given sampling dataset $(\boldsymbol{X}, \boldsymbol{Y})$. When input matrix $\boldsymbol{X}$ is a serial fixed sampling data, the constant matrix $\mathbb{C}$ only related to the function f, realizes (i) numerization of the function and (ii) local decoupling of multivariate, by linearization processing, as shown in Hypothesis 1(a). Therefore, the processing method has four characteristics: (i) linearization, (ii) matrixization, (iii) numerization, and (iv) decoupling.

Geometric Understanding

Using the classical concepts of linearization and isomorphism, and three proposed hypotheses, LL-IC system theory is constructed, which is further explained as follows.

System matrixization and system feature matrix $\mathbb{C}_X$ is obtained by the LL process. Local linearity is used to describe the global nonlinearity, which reflects the dialectical unity relations between the whole and the part, linearity and nonlinearity. Visualized understanding of LL is that at each sampling data point, the local linear hyperplane is used to expand its surrounding areas, and the approximate description of the global space is completed like a tent or membrane structure building (e.g. Olympiastadion München and The Shed in NewYork), as shown in Fig. S10b-d. System matrixization has characteristics of linearization, matrixization, quantification, and decoupling.

IC stemming from causal determinism solves the solution of $\mathbb{C}_X$ and W. The core of IC is to ensure that all influences are kept fixed except for the changes(like weights) we care about. The isomorphic system spaces constructed by different parameters W can be transformed by the continuous change of W, which could be imagined as the continuous stretching and torsion of the system space, see Fig. S10e,f. In the process of stretching and torsion, the change of gradient is often limited. This is also the reason of reuse and why $\mathbb{C}_X$ is effective.–

S1-S6 are algorithms constructed directly from the concepts of LL and IC, and the geometric understanding can be found in Fig. S10. In contrast, S7 no longer requires complex matrices, is simpler, less computation and resourse requirement. Convergence is achieved by repeated iterations according to $\Delta \boldsymbol{W} = k\boldsymbol{W_r}$.

The geometric understanding of LIFT S7 is presented in Fig.1 e-f. $\boldsymbol{W}$ is the current parameters; $\boldsymbol{W}^*$ is the expected parameters; the black dashed lines $\boldsymbol{WW}^*$represent the ideal tuning direction; the red solid lines represent the weight-adjusted trajectory $\Delta \boldsymbol{W}$**.** $\Delta \boldsymbol{W} = k\boldsymbol{W_r}$, the direction of adjustment is determined by randomly generated $\boldsymbol{W_r}$, which has an obvious error with the ideal gradient direction $\boldsymbol{WW}^*$. While the systematic error will decrease, and the active ingredient is extracted by scalar product calculation.

Convergence analysis

Obviously, in Fig.1 e-f, the LIFT S7 convergence condition can be obtained as:

$$l_r k\|\boldsymbol{W_r}\| < \|\boldsymbol{WW'}\| \text{ and } <\boldsymbol{E'},\boldsymbol{E}>\neq \boldsymbol{0}.$$

where

$$k = (\boldsymbol{Y''}-\boldsymbol{Y'})^{\dagger}(\boldsymbol{Y}-\boldsymbol{Y'}) = \frac{<\boldsymbol{E'},\boldsymbol{E}>}{<\boldsymbol{E},\boldsymbol{E}>}$$

l_r is the learning rate, $\boldsymbol{WW^*}$ is the ideal gradient direction, $\boldsymbol{WW'}$ has the same direction with $\Delta\boldsymbol{W}$, and $\Delta\boldsymbol{W^*WW'}$ is a isosceles triangle (Idealized abstraction, due to the complex spatial distribution of the system, this abstract result does not necessarily hold.).

According to the geometry understanding, the weight adjustment $l_r\Delta\boldsymbol{W}$ should be in the range of $\boldsymbol{WW'}$, that is $l_r k\|\boldsymbol{W_r}\| < \|\boldsymbol{WW'}\|$.

Additional Notes:

(1) If $<\boldsymbol{E'},\boldsymbol{E}>=\boldsymbol{0}$, $\Delta\boldsymbol{W}=\boldsymbol{0}$, no adjustment for current $\boldsymbol{W}$ and training ineffective. The probability of orthogonality of two vectors in a multidimensional space is very low, thus ensuring the success of the algorithm, which converges in the overall process.

(2) Due to $l_r k\|\boldsymbol{W_r}\| < \|\boldsymbol{WW'}\|$, Although $\|\boldsymbol{WW'}\|$ is difficult to obtain in practical engineering, it does give us some hints, such as: $\|\boldsymbol{W_r}\|$ and the learning rate l_r should be appropriately small.

System analysis

System matrixization is a geometric algebraization method, has characteristics of linearization, matrixization, quantification, and decoupling, and enables two-dimensionalized representation of high-dimensional systems. To some extent, $\mathbb{C}_{\boldsymbol{X}}$ can be used as a tool for systems analysis, like system function.

Here's a list of some pre-thinking.

(1) Similarity

According to the Hypothesis 3: For two stable parametric systems processed by LL. (a) (Invariability): The systems with the same structure and parameters ($\boldsymbol{W}=\boldsymbol{W'}$), for the same input vector $\boldsymbol{X}$ of the dataset, have the same output vector ($\boldsymbol{Y}=\boldsymbol{Y'}$) and the same constant matrix ($\mathbb{C}_{\boldsymbol{X}}=\mathbb{C}'_{\boldsymbol{X}}$).(b) (Comparability): The systems with the same structure and **comparable** parameter ($\boldsymbol{W}\approx\boldsymbol{W'}$), for the same input vector $\boldsymbol{X}$ of the dataset, have the **comparable** constant matrix ($\mathbb{C}_{\boldsymbol{X}}\approx\mathbb{C}'_{\boldsymbol{X}}$), and $\lim_{\boldsymbol{W'}\to\boldsymbol{W}}\mathbb{C}'_{\boldsymbol{X}}=\mathbb{C}_{\boldsymbol{X}}$.

$$\mathrm{Si} = \|\mathbb{C}_{\boldsymbol{X}} - \mathbb{C}'_{\boldsymbol{X}}\|$$

Si measures the direct similarity between the two systems. The smaller the Si, the higher the similarity.

(2) Sensitivity

With the change of parameters($\boldsymbol{W} \to \boldsymbol{W}'$), the system will change($\mathbb{C}_{\boldsymbol{X}} \to \mathbb{C}'_{\boldsymbol{X}}$). Se is calculated by the following formula to show the sensitivity of the system to parameter changes.

$$\mathrm{Se} = \|\mathbb{C}_{\boldsymbol{X}} - \mathbb{C}'_{\boldsymbol{X}}\|$$

$\mathbf{W} = [w_1, w_2, \dots, w_S]$, for different parameter components w_s, the sensitivity of the system to different parameters can be measured by Se_s.

$$\mathrm{Se}_s = \|\mathbb{C}_{\boldsymbol{X}} - \mathbb{C}'_{\boldsymbol{X}}\|_{\Delta w_s}$$

Solving $\max\{\mathrm{Se}_s\}$, the maximum corresponds to the most sensitive parameter w_s of the current system.

(3) smoothability

$\mathbb{C}_{\boldsymbol{X}}$ has a clear mathematical meaning, and each element is a corresponding partial derivative of the output to the input, as shown in **Hypothesis 1(b)**.

$$\mathrm{Sm} = \|\mathbb{C}_{\boldsymbol{X}}\|$$

The smaller the Sm, the smoother the system space.

Solving $\max\{\mathbb{C}_{\boldsymbol{X}}\}$, the maximum maybe correspond to the most steepest place of the current system.

Notes:

(1) The distribution of $(\mathbf{X}, \mathbf{Y})$ is known.

(2) Only considering the local space, we can get local similarity, sensitivity and smoothability.

Supplementary Methods

Method S1

Direct Method:

Two completely isomorphic systems with different parameters, which are stable and smooth parametric systems, satisfy

$$\begin{cases} \boldsymbol{Y} = \mathbb{H}(\boldsymbol{W}) = \mathbb{C}_{\boldsymbol{X}}\boldsymbol{W} \\ \boldsymbol{Y}' = \mathbb{H}'(\boldsymbol{W}') = \mathbb{C}'_{X}\boldsymbol{W}' \end{cases}$$

respectively according to **Hypothesis 1,** and

$$\mathbb{C}'_X \approx \mathbb{C}_X$$

by **Hypothesis 3.** Then,

$$\mathbb{C}'_X \approx \mathbb{C}_X = \boldsymbol{Y}\boldsymbol{W}^{\dagger},$$

where $\boldsymbol{W}^{\dagger}$ is the Moore-Penrose generalized inverse of $\boldsymbol{W}$.

The performance of the known system is used to directly solve the feature matrix of the unknown system.

Method S2

Difference Method:

Two completely isomorphic systems with different parameters, which are stable and smooth parametric systems, satisfy

$$\begin{cases} \boldsymbol{Y} = \mathbb{H}(\boldsymbol{W}) = \mathbb{C}_{\boldsymbol{X}}\boldsymbol{W} \\ \boldsymbol{Y}' = \mathbb{H}'(\boldsymbol{W}') = \mathbb{C}'_{X}\boldsymbol{W}' \end{cases}$$

respectively according to **Hypothesis 1,** and

$$\mathbb{C}'_X \approx \mathbb{C}_X$$

by **Hypothesis 3.** Then,

$$\mathbb{C}'_X \approx \mathbb{C}_X \approx (\boldsymbol{Y} - \boldsymbol{Y}')(\boldsymbol{W} - \boldsymbol{W}')^{\dagger} = \Delta\boldsymbol{Y}\Delta\boldsymbol{W}^{\dagger},$$

where

$$\begin{cases} \Delta\boldsymbol{Y} = \boldsymbol{Y} - \boldsymbol{Y}' \\ \Delta\boldsymbol{W} = \boldsymbol{W} - \boldsymbol{W}' \end{cases},$$

and $\Delta \boldsymbol{W}^{\dagger}$ is the Moore-Penrose generalized inverse of $\Delta \boldsymbol{W}$.

Solve for the feature matrix of the unknown system using the difference in performance between the known and unknown systems.

Method S3

Direct Method:

For a stable and smooth parametric system with training data $(\boldsymbol{X}, \boldsymbol{Y})$ and current weights $\boldsymbol{W}'$, its current output vector

$$\boldsymbol{Y}' = \mathbb{H}(\boldsymbol{W}') = \mathbb{C}_X \boldsymbol{W}'.$$

So the system feature matrix

$$\mathbb{C}_X = \boldsymbol{Y}' \boldsymbol{W}'^{\dagger}$$

And the object output and weights are

$$\boldsymbol{Y} = \mathbb{H}(\boldsymbol{W}) = \mathbb{C}_X \boldsymbol{W}.$$

Then update the weights by

$$\boldsymbol{W} = \mathbb{C}_{\boldsymbol{X}}^{\dagger} \boldsymbol{Y} = (\boldsymbol{Y}' \boldsymbol{W}'^{\dagger})^{\dagger} \boldsymbol{Y} = \boldsymbol{W}' \boldsymbol{Y}'^{\dagger} \boldsymbol{Y}$$

Let

$$k = \boldsymbol{Y}'^{\dagger} \boldsymbol{Y}$$

Then

$$\boldsymbol{W} = \boldsymbol{W}' k$$

Thus it can be seen that k is the scale coefficient of $\boldsymbol{W}'$.

Method S4

For a stable and smooth parametric system with training data $(\boldsymbol{X}, \boldsymbol{Y})$ and current weights $\boldsymbol{W}'$, its current output vector

$$\boldsymbol{Y}' = \mathbb{H}(\boldsymbol{W}') = \mathbb{C}_X \boldsymbol{W}'.$$

And the object output and weights are

$$\boldsymbol{Y} = \mathbb{H}(\boldsymbol{W}) = \mathbb{C}_X \boldsymbol{W}.$$

Then

$$\Delta \boldsymbol{W} = \boldsymbol{W} - \boldsymbol{W}' = \mathbb{C}_{\boldsymbol{X}}^{\dagger}(\boldsymbol{Y} - \boldsymbol{Y}') = \mathbb{C}_{\boldsymbol{X}}^{\dagger}\boldsymbol{E},$$

where $\boldsymbol{E} = \boldsymbol{Y} - \boldsymbol{Y}'.$

Because the gap between $\boldsymbol{W}$ and $\boldsymbol{W}'$ is always too large to obtain a sufficiently accurate $\mathbb{C}_{\boldsymbol{X}}^{\dagger}$. So we need to reduce this gap to a sufficiently small one. Given a little disturbance parameter $\boldsymbol{W_r}$, and the output changes to

$$\boldsymbol{Y}'' = \mathbb{H}(\boldsymbol{W}' + \boldsymbol{W_r}) = \mathbb{C}_X(\boldsymbol{W}' + \boldsymbol{W_r}).$$

So the system feature matrix

$$\mathbb{C}_X = (\boldsymbol{Y}'' - \boldsymbol{Y}')(\boldsymbol{W}' + \boldsymbol{W_r} - \boldsymbol{W}')^{\dagger} = (\boldsymbol{Y}'' - \boldsymbol{Y}')\boldsymbol{W_r}^{\dagger}$$

Then update the weights by

$$\begin{cases} \boldsymbol{W} = \boldsymbol{W}' + l_r \Delta \boldsymbol{W} \\ \Delta \boldsymbol{W} = \mathbb{C}_{\boldsymbol{X}}^{\dagger}\boldsymbol{E} = ((\boldsymbol{Y}'' - \boldsymbol{Y}')\boldsymbol{W_r}^{\dagger})^{\dagger}(\boldsymbol{Y} - \boldsymbol{Y}') = \boldsymbol{W_r}(\boldsymbol{Y}'' - \boldsymbol{Y}')^{\dagger}(\boldsymbol{Y} - \boldsymbol{Y}') \end{cases}.$$

where, l_r is the learning rate, $\boldsymbol{E}' = \boldsymbol{Y}'' - \boldsymbol{Y}'.$

Let

$$k = (\boldsymbol{Y}'' - \boldsymbol{Y}')^{\dagger}(\boldsymbol{Y} - \boldsymbol{Y}') = \frac{< \boldsymbol{E}', \boldsymbol{E} >}{< \boldsymbol{E}, \boldsymbol{E} >}$$

Then

$$\begin{cases} \boldsymbol{W} = \boldsymbol{W}' + l_r \Delta \boldsymbol{W} \\ \Delta \boldsymbol{W} = k\boldsymbol{W_r} \end{cases}$$

It is an iterative training method and trains all training data synchronously in an iteration.

Algorithm S1

Algorithm S1 Direct method

Input: Training data $\{(x_i, y_i)\}$, $i = 1, \dots, N_d$;
System structure and forward mapping $y = \mathrm{F}(x, W)$.

1 W' with random initialization;
2 $y_i' = \mathrm{F}(x_i, W')$;
3 $\mathrm{Y}' = [y'_1, y'_2, \dots, y'_{N_d}]^T$;
4 $C_X \approx C_X' = Y'W'^{\dagger}$;
5 $Y = [y_1, y_2, \dots, y_{N_d}]^T$;
6 $W = C_x^{\dagger} Y$.

Output: W.

Algorithm S2

Algorithm S2 Difference method

Input: Training data $\{(x_i, y_i)\}$, $i = 1, \dots, N_d$; r and l_r;
System structure and forward mapping $y = \mathrm{F}(x, W)$.

W with random initialization in [-1, 1];
$Y = [y_1, y_2, \dots, y_{N_d}]^T$;
for $i = 1$ **to** N_d do
  $y_i^{(1)} = \mathrm{F}(x_i, W)$;
end for
$Y^{(1)} = [y_1^{(1)}, y_2^{(1)}, \dots, y_{N_d}^{(1)}]^T$;
$\mathrm{E} = Y - Y^{(1)}$;
while ($epoch < Max$ and $MSE(E) > \varepsilon$)
  W_r with random initialization in [-r, r];
  for $i = 1$ **to** N_d **do**
    $y_i^{(2)} = \mathrm{F}(x_i, W + W_r)$;
  end for
  $Y^{(2)} = [y_1^{(2)}, y_2^{(2)}, \dots, y_{N_d}^{(2)}]^T$;
  $C_X \approx C_X' = (Y^{(2)} - Y^{(1)}){W_r}^{\dagger}$;
  $\Delta W = C_x^{\dagger} E$;
  $W = W + l_r \Delta W$;
  for $i = 1$ **to** N_d **do**
    $y_i^{(1)} = \mathrm{F}(x_i, W)$;
  end for
  $Y^{(1)} = [y_1^{(1)}, y_2^{(1)}, \dots, y_{N_d}^{(1)}]^T$;
  $\mathrm{E} = Y - Y^{(1)}$;
  $epoch++$;
end while

Output: W.

Algorithm S3

Algorithm S2 Difference method with reuse of $C_x^{\dagger}$.

Input: Training data $\{(x_i, y_i)\}$, $i = 1, \dots, N_d$; r and l_r;
System structure and forward mapping $y = \mathrm{F}(x, W)$.

W with random initialization in [-1, 1];
$Y = [y_1, y_2, \dots, y_{N_d}]^T$;
for $i = 1$ **to** N_d **do**
  $y_i^{(1)} = \mathrm{F}(x_i, W)$;
end for
$Y^{(1)} = [y_1^{(1)}, y_2^{(1)}, \dots, y_{N_d}^{(1)}]^T$;
$\mathrm{E} = Y - Y^{(1)}$;
while ($epoch < Max$ and $MSE(E) > \varepsilon$)
  W_r with random initialization in [-r, r];
  for $i = 1$ **to** N_d **do**
    $y_i^{(2)} = \mathrm{F}(x_i, W + W_r)$;
  end for
  $Y^{(2)} = [y_1^{(2)}, y_2^{(2)}, \dots, y_{N_d}^{(2)}]^T$;
  $C_X \approx C_X' = (Y^{(2)} - Y^{(1)}){W_r}^{\dagger}$;
  while $MSE(E)$ decreasing
    $\Delta W = C_x^{\dagger} E$;
    $W = W + l_r \Delta W$;
    for $i = 1$ **to** N_d **do**
      $y_i^{(1)} = \mathrm{F}(x_i, W)$;
    end for
    $Y^{(1)} = [y_1^{(1)}, y_2^{(1)}, \dots, y_{N_d}^{(1)}]^T$;
    $\mathrm{E} = Y - Y^{(1)}$;
  end while
  $epoch++$;
end while

Output: W.

Algorithm S4

Algorithm S4 Difference method with batch process of training data

Input: Training data $\{(x_i, y_i)\}$, $i = 1, \dots, N_d$; r, l_r, and M_b;
System structure and forward mapping $y = \mathrm{F}(x, W)$.

W with random initialization in [-1, 1];
$Y = [y_1, y_2, \dots, y_{N_d}]^T$;
while (iteration $< Max$ and $MSE(E) > \varepsilon$)
  for $k = 0$ **to** $(N_d/M_b - 1)$ **do**
    $Y = [y_{k+1}, y_{k+2}, \dots, y_{k+M_b}]^T$;
    for $i = (k + 1)$ **to** $(k + M_b)$ **do**
      $y_i^{(1)} = \mathrm{F}(x_i, W)$;
    end for
    $Y^{(1)} = [y_{k+1}^{(1)}, y_{k+2}^{(1)}, \dots, y_{k+M_b}^{(1)}]^T$;
    $\mathrm{E} = Y - Y^{(1)}$;
    W_r with random initialization in [-r, r];
    for $i = (k + 1)$ **to** $(k + M_b)$ **do**
      $y_i^{(2)} = \mathrm{F}(x_i, W + W_r)$;
    end for
    $Y^{(2)} = [y_{k+1}^{(2)}, y_{k+2}^{(2)}, \dots, y_{k+M_b}^{(2)}]^T$;
    $C_X \approx C_X' = (Y^{(2)} - Y^{(1)}){W_r}^{\dagger}$;
    $\Delta W = C_x^{\dagger} E$;
    $W = W + l_r \Delta W$;
    for $i = (k + 1)$ **to** $(k + M_b)$ **do**
      $y_i^{(1)} = \mathrm{F}(x_i, W)$;
    end for
    $Y^{(1)} = [y_{k+1}^{(1)}, y_{k+2}^{(1)}, \dots, y_{k+M_b}^{(1)}]^T$;
    $\mathrm{E} = Y - Y^{(1)}$;
  end for
  iteration $++$;
end while

Output: W.

Algorithm S5

Algorithm S5 Difference method with batch process of training data and weights.

Input: Training data $\{(x_i, y_i)\}$, $i = 1, \ldots, N_d$; r, l_r, M_b and N_b; System structure and forward mapping $y = \mathrm{F}(x, W)$.

W with random initialization in [-1, 1];
$Y = [y_1, y_2, \ldots, y_{N_d}]^T$;
while (iteration $< Max$ and $MSE(E) > \varepsilon$)
  for $k = 0$ **to** $(N_d/M_b - 1)$ **do**
    $Y = [y_{k+1}, y_{k+2}, \ldots, y_{k+M_b}]^T$;
    for $i = (k + 1)$ **to** $(k + M_b)$ **do**
      $y_i^{(1)} = \mathrm{F}(x_i, W)$;
    end for
    $Y^{(1)} = [y_{k+1}^{(1)}, y_{k+2}^{(1)}, \ldots, y_{k+M_b}^{(1)}]^T$;
    $\mathrm{E} = Y - Y^{(1)}$;
    for $p = 0$ **to** $(S/N_b - 1)$ **do**
      $W_b = [W_{p+1}, W_{p+2}, \ldots, W_{p+N_b}]$;
      W_r with random initialization in [-r, r];
      for $i = (k + 1)$ **to** $(k + M_b)$ **do**
        $y_i^{(2)} = \mathrm{F}(x_i, W_b + W_r)$;
      end for
      $Y^{(2)} = [y_{k+1}^{(2)}, y_{k+2}^{(2)}, \ldots, y_{k+M_b}^{(2)}]^T$;
      $C_p \approx C'_{X,W_t} = (Y^{(2)} - Y^{(1)}){W_r}^{\dagger}$;
    end for
    $C_x^{\dagger} = [C_1^{\dagger}; C_2^{\dagger}; \ldots; C_{S/N_b-1}^{\dagger}]$;
    $\Delta W = C_x^{\dagger} E$;
    $W = W + l_r \Delta W$;
    for $i = (k + 1)$ **to** $(k + M_b)$ **do**
      $y_i^{(1)} = \mathrm{F}(x_i, W)$;
    end for

$Y^{(1)} = [y_{k+1}^{(1)}, y_{k+2}^{(1)}, \cdots, y_{k+M_b}^{(1)}]^T$;

E $= Y - Y^{(1)}$;

end for

iteration $++$;

end while

Output: W.

Algorithm S6

Algorithm S6 Difference method with batch process of training data and weights and reuse of $C_x^{\dagger}$.

Input: Training data $\{(x_i, y_i)\}$, $i = 1, \ldots, N_d$; r, l_r, M_b and N_b; System structure and forward mapping $y = \mathrm{F}(x, W)$.

W with random initialization in [-1, 1];
$Y = [y_1, y_2, \ldots, y_{N_d}]^T$;
for $k = 0$ **to** $(N_d/M_b - 1)$ **do**
  $Y = [y_{k+1}, y_{k+2}, \ldots, y_{k+M_b}]^T$;
  for $i = (k+1)$ **to** $(k + M_b)$ **do**
    $y_i^{(1)} = \mathrm{F}(x_i, W)$;
  end for
  $Y^{(1)} = [y_{k+1}^{(1)}, y_{k+2}^{(1)}, \ldots, y_{k+M_b}^{(1)}]^T$;
  $\mathrm{E} = Y - Y^{(1)}$;
  while (iteration $< Max$ and $MSE(E) > \varepsilon$)
  W random reordering
    for $p = 0$ **to** $(S/N_b - 1)$ **do**
      $W_b = [W_{p+1}, W_{p+2}, \ldots, W_{p+N_b}]$;
      W_r with random initialization in [-r, r];
      for $i = (k+1)$ **to** $(k + M_b)$ **do**
        $y_i^{(2)} = \mathrm{F}(x_i, W_b + W_r)$;
      end for
      $Y^{(2)} = [y_{k+1}^{(2)}, y_{k+2}^{(2)}, \ldots, y_{k+M_b}^{(2)}]^T$;
      $C_p \approx C'_{X,W_t} = (Y^{(2)} - Y^{(1)}){W_r}^{\dagger}$;
    end for
    $C_x^{\dagger} = [C_1^{\dagger}; C_2^{\dagger}; \ldots; C_{S/N_b-1}^{\dagger}]$;
    while $MSE(E)$ decreasing
      $\Delta W = C_x^{\dagger} E$;
      $W = W + l_r \Delta W$;
      for $i = (k+1)$ **to** $(k + M_b)$ **do**

$y_i^{(1)} = \mathrm{F}(x_i, W)$;

end for

$Y^{(1)} = [y_{k+1}^{(1)}, y_{k+2}^{(1)}, \cdots, y_{k+M_b}^{(1)}]^T$;

$\mathrm{E} = Y - Y^{(1)}$;

end while

end for

iteration $++$;

end while

Output: W.

Algorithm S7

Algorithm S7 Improved LIFT without matrix.

Input: Training data $\{(x_i, y_i)\}$, $i = 1, \dots, N_d$; r and l_r;
System structure and forward mapping $y = \mathrm{F}(x, W)$.

W with random initialization in [-1, 1];
$Y = [y_1, y_2, \dots, y_{N_d}]^T$;
for $i = 1$ **to** N_d **do**
  $y_i^{(1)} = \mathrm{F}(x_i, W)$;
end for
$Y^{(1)} = [y_1^{(1)}, y_2^{(1)}, \dots, y_{N_d}^{(1)}]^T$;
$\mathrm{E} = Y - Y^{(1)}$;
while ($epoch < Max$ and $MSE(E) > \varepsilon$)
  W_r with random initialization in [-r, r];
  for $i = 1$ **to** N_d **do**
    $y_i^{(2)} = \mathrm{F}(x_i, W + W_r)$;
  end for
  $Y^{(2)} = [y_1^{(2)}, y_2^{(2)}, \dots, y_{N_d}^{(2)}]^T$;
  $\mathrm{E}' = Y^{(2)} - Y^{(1)}$;
  $k = \frac{<E', E>}{<E, E>}$
  $\Delta W = kW_r$;
  $W = W + l_r \Delta W$;
end while

Output: W.

Supplementary Experiments

Experiment S1: IIR

The *N*th-order IIR(Infinite Impulse Response) Filter has a system function:

$$H(z)=\frac{Y(z)}{X(z)}=\frac{\sum_{k=0}^{N} b_k z^{-k}}{1-\sum_{k=1}^{N} a_k z^{-k}},$$

and

$$H(e^{j\omega})=H(z)|_{z=e^{j\omega}}=|H(e^{j\omega})|\,e^{j\beta(i\omega)},$$

where $|H(e^{j\omega})|$ is amplitude-frequency characteristic, and $\beta(j\omega)$ is phase-frequency characteristic.

For this IIR filter system, input is angular frequency $\omega = k2\pi/1000,\ k = 0,1,\ldots,500,$ corresponding output is the amplitude-frequency characteristic $|H(e^{j\omega})|$, that is

$$|H(e^{j\omega})|=f(\omega)=\left|\sum_{k=0}^{N} b_k z^{-k}/(1-\sum_{k=1}^{N} a_n z^{-k})\right|_{z=e^{j\omega}}=\left|\sum_{k=0}^{N} b_k e^{-jk\omega}/(1-\sum_{k=1}^{N} a_k e^{-jk\omega})\right|$$

And the system parameter $W=[a_1,a_2,\ldots,a_N,b_0,b_1,\ldots,b_N]$. It is a single-input-single-output (SISO) system. The IIR filter system is designed by the solution of $\{a_k\}$ and $\{b_k\}$.

The difference equation corresponding to the IIR system is

$$y(n)=\sum_{k=0}^{N} b_k x(n-k)+\sum_{k=1}^{N} a_k y(n-k)$$

It means that there is feedback between the input and output of the IIR system.

Experiment 1: Design a *100*-order IIR filter so that its amplitude-frequency characteristics meet specific requirements (WL-type filter).

Specific model parameters are given in Table S1.

Experiment S2: DNN

Neural networks can be used for classification and regression prediction and are a fundamental element of modern AI solutions. Classical Multi-Layer Feedforward Networks (MLFNs) consist of an input layer, multiple hidden layers, and an output layer. Suppose the classical structure is P-

L-Q, and the specific parameters are shown in Table S2. The neural network adopts a fully connected structure. The feedforward neural networks perform forward computation by iterating layer by layer:

$$\begin{cases} \mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \\ \mathbf{a}^{(l)} = \phi(\mathbf{z}^{(l)}) \end{cases}$$

Experiment 2: Design a fully-connected DNN with **40 hidden layers**, to fit a two-input-two-output nonlinear system (two mixed benchmark functions: Ackley and Adjiman).

Specific model parameters are given in Table S2.

Experiment S3: SaNN

To verify the adaptability of nonclassical nonlinear components in ANN, this section selects the MIMO **Sampling Neural Network** (SaNN). MIMO SaNN uses single-input-single-output (SISO) Sampling neural network (SaNN) as the neuron activation function and accomplishes neural network learning by neuron activation functions training. The feedforward neural networks perform forward computation by iterating layer by layer:

$$\begin{cases} \mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \\ \mathbf{a}^{(l)} = \phi(\mathbf{z}^{(l)}) \end{cases}$$

where the neurons' input and output satisfy the following formula:

$$\begin{cases} t = \dfrac{\mathrm{Ns}}{2}(\mathrm{Softsign}(\mathbf{z}^{(l)}) + 1) = \dfrac{\mathrm{Ns}}{2}\left(\dfrac{\mathbf{z}^{(l)}}{1 + |\mathbf{z}^{(l)}|} + 1\right) \\ \mathbf{a}^{(l)} = \displaystyle\sum_{n=1}^{\mathrm{Ns}} W_n Sa\left[\frac{\pi}{T}(t - nT)\right] \end{cases}$$

The main differences between SaNN and traditional multi-layer neural networks are i)The connection weights between hidden layers do not need to be trained. The inter-layer link weights were randomly initialized with 0 or ± 1, and then remained fixed, in Experiment 3. ii) The number of training weights for each layer in SaNN is $M_l \times N_s$, rather than $M_l \times M_l$ in MLFNs. Since usually $N_s < M_l$, it is beneficial to reduce the network size. iii) The activation function of each neuron is completely independent and trainable, while the traditional network uses a pre-set and fixed activation function, like sigmoid, tanh, ReLU, ELU, and so on.

Experiment 3: Design a SaNN with **10** hidden layers, to fit a two-input-two-output nonlinear system (two mixed benchmark functions: Ackley and Adjiman).

Specific model parameters are given in Table S3.

Experiment S4: SaKAN

KAN(**Kolmogorov-Arnold Network**) is a novel and hot network architecture based on Kolmogorov-Arnold Representation Theorem, that has emerged in nearly a month. It improves the network performance and interpretability by replacing the weight parameters with learnable univariate functions, and has the potential to become an important direction for driving the development of deep learning models. Its main features are that the learnable activation and B-spline function replacing the traditional linear weights.

As a new type of network architecture, there are still many areas to be improved, such as the difficulty of network training. To further verify the universality of the LIFT algorithm, we constructed a SaKAN (KAN with SaNN) network that replaced the B-spline function with the SaNN. The basic structure of SaKAN is as follows:

$$\begin{cases} W_{i,j}^{(l)} = \varphi_{SaNN}\left(x_i^{(l)}\right) \\ y_j^{(l)} = \sum_i W_{i,j}^{(l)} \\ x_j^{(l)} = Softsign\left(y_j^{(l+1)}\right) = \dfrac{y_j^{(l+1)}}{\left|y_j^{(l+1)}\right| + 1} \end{cases}$$

where the function φ_{SaNN} is the learnable input-output relationship of SaNN, which weights are the system parameters to be sought.

Experiment 4: Design a SaKAN with **1** hidden layers, to fit a two-input-two-output nonlinear system (two mixed benchmark functions: Ackley and Adjiman).

Specific model parameters are given in Table S4.

Experiment S5: RAN

RAN: Neural Network with Randomly selected Activation functions.

Here employed is still the MLP(Multilayer Perceptron) or MLFNs (Multi-Layer Feedforward Networks), classical neural network structure. The neural networks perform forward computation by iterating layer by layer:

$$\begin{cases} \mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \\ \mathbf{a}^{(l)} = \phi(\mathbf{z}^{(l)}) \end{cases}$$

Where the activation functions $\phi(.)$ are no longer a fixed function, but the specific functions randomly selected at initialization from 10 functions. Details in Table S6. These functions include traditional activation functions, non-continuous segmented functions, and non-derivative functions.

Experiment 5: Design a RAN with **3** hidden layers, to fit a two-input-two-output nonlinear system (two mixed benchmark functions: Ackley and Adjiman).

Specific model parameters are given in Table S5.

Experiment S6: MLP

MLP(Multilayer Perceptron) or MLFNs (Multi-Layer Feedforward Networks) is a classical neural network structure. DNN in Experiment S2 is also a MLP with more hidden layers.

In order to compare LIFT with BP, a three-layer network structure is selected. Because basic BP is not suitable for deep networks.

The neural networks perform forward computation by iterating layer by layer:

$$\begin{cases} \mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \\ \mathbf{a}^{(l)} = \phi(\mathbf{z}^{(l)}) \end{cases}$$

Experiment 6: Design a MLP with **3** hidden layers, to fit a two-input-two-output nonlinear system (two mixed benchmark functions: Ackley and Adjiman).

Specific model parameters are given in Table S6.

Supplementary Tables

Table S1.

Some parameters of **Experiment 1** for IIR design.

Items	**Descriptions or Parameters**
Input P	1
Output Q	1
Weights S	201 where 100 for $\{a_n\}$, n=1,…, 100, and 101 for $\{b_n\}$, n=0,1,…, 100.
Object	WL-type filter, the object as shown in Fig. 2, S1 and Data S1.
Input	$\omega \in (0,2\pi)$
Output	$\left\|H(e^{i\omega})\right\|$
*N*th-order	100
N_d	500
W initialization	random in [-1, 1]
Pre-training	Yes, **Algorithm S1**
Iterative training algorithm	**Algorithm S3**
Learn rate l	$\begin{cases} 0.1, \text{epoch} < 500000 \\ 0.01, \text{others} \end{cases}$
W_r	Random in [-0.01, 0.01]
Epoch Max	100 Million
Training upper limit per Epoch	100

Table S2.

Some parameters of **Experiment 2** of DNN:

Items	**Descriptions or Parameters**
Input	$\mathbf{X} = [x_1, x_2, \dots, x_P]_{1\times P}$
Output	$\mathbf{Y} = [y_1, y_2, \dots, y_Q]_{1\times Q}$
The weights matrix between layer *l*-1 and layer *l*	$\mathbf{W}^{(l)}$
The bias of neurons in layer l	$\mathbf{b}^{(l)}$
Input dimension, P	2
Output dimension, Q	2
The number of hidden layers, L	40
Weights dimension, S	15680
The number of neurons in layer l, *Ml*	*Ml*=20 for all layers
Activation function	'tanh' for all neurons: $a_i^{(l)} = \emptyset_i^{(l)}(z_i^{(l)}) = \frac{e^{z_i^{(l)}} - e^{-z_i^{(l)}}}{e^{z_i^{(l)}} + e^{-z_i^{(l)}}}$
Object function	Benchmark functions: Normalized Ackley and Adjiman $\begin{cases} y_1 = f(x_1, x_2) = e^{-0.2\sqrt{x_1^2 + x_2^2}} + 3\cos 2x_1 + \sin 2x_2 \\ y_2 = f(x_1, x_2) = \cos(x_1)\sin(x_2) - \frac{x_1}{x_2^2 + 1} \end{cases}$ $x_1 \in [0,10], x_2 \in [0,10]$ See **Fig. S3**.
Training data set N_d	10000
Test data set N_t	10000
Batch process of weights M_b	98
Batch process of training data N_b	98
W initialization	random in [-1, 1]
Pre-training	No
Iterative training algorithm	**Algorithm S6**
Learning rate lr	0.005
W_r	Random in [-0.01, 0.01]
Update batch of training data	Randomly selected (shuffling) every 25 iterations (Batch)
Iteration maximum	10000
Training upper limit per iteration (**reuse**)	10000

Table S3.

Some parameters of **Experiment 3** of SaNN.

Items	**Parameters**
Input dimension, P	2
Output dimension, Q	2
The number of hidden layers, L	10
Weights dimension, S	2000
The number of neurons in layer l, M_l	M_l=20 for all layers
Activation function	SISO SaNN neurons (details in **Experiment S3**).
The number of weights of a single SISO SaNN neuron, N_s	10
Object function	Benchmark functions: Normalized Ackley and Adjiman $\begin{cases} y_1 = f(x_1, x_2) = e^{-0.2}\sqrt{x_1^2 + x_2^2} + 3\cos 2x_1 + \sin 2x_2 \\ y_2 = f(x_1, x_2) = \cos(x_1)\sin(x_2) - \frac{x_1}{x_2^2 + 1} \end{cases}$ $x_1 \in [0,10], x_2 \in [0,10]$ See **Fig. S5**.
Training data set N_d	10000
Test data set N_t	A total of 10000. In **Fig. S4**, test 100 data after each iteration. In **Fig. S5**, test the whole 10000 data.
Batch process of weights M_b	100
Batch process of training data N_b	100
W initialization of SISO SaNN for all neurons	Random in [-1, 1]
Inter-layer link weights W initialization	Randomly initialized with 0 or ± 1. As shown in **Fig. S4h**, a total of 3680, initialized as "0" has 1802(48.97%), "1" has 905(24.59%), and "-1" has 973 (26.44%).
Pre-training	Yes, **Algorithm S1** by randomly selected 100 train data.
Iterated operation	**Algorithm S6**
Learn rate lr	0.01
W_r	Random in [-0.01, 0.01]
Update batch of training data	Randomly selected (shuffling) every 25 iterations (Batch)
Iteration maximum	50000
Training upper limit per iteration (**reuse**)	10000

Table S4.

Some parameters of **Experiment 4** of SaKAN.

Items	**Parameters**
Input dimension, P	2
Output dimension, Q	2
The number of hidden layers, L	1
Weights dimension, S	400
The number of neurons in layer l, M_l	M_l=10 for all layers
Activation function	SISO SaNN neurons (details in **Experiment S4**).
The number of weights of a single SISO SaNN neuron, N_s	10
Object function	Benchmark functions: Normalized Ackley and Adjiman $\begin{cases} y_1 = f(x_1, x_2) = e^{-0.2}\sqrt{x_1^2 + x_2^2} + 3\cos 2x_1 + \sin 2x_2 \\ y_2 = f(x_1, x_2) = \cos(x_1)\sin(x_2) - \frac{x_1}{x_2^2 + 1} \end{cases}$ $x_1 \in [0,10], x_2 \in [0,10]$
Training data set N_d	10000
Test data set N_t	A total of 10000.
Batch process of weights M_b	400 (No batch process)
Batch process of training data N_b	1000
W initialization of SISO SaNN for all neurons	Zeros
Pre-training	None
Iterated operation	**Algorithm S7**
Learn rate lr	1
W_r	Random in [-0.01, 0.01]
Update batch of training data	Randomly selected (shuffling) every 25 iterations (Batch)
Iteration maximum	50000
Training upper limit per iteration (**reuse**)	1

Table S5.

Some parameters of **Experiment 5** of RAN.

Items	**Parameters**		
Input dimension, P	2		
Output dimension, Q	2		
The number of hidden layers, L	3		
Weights dimension, S	5200		
The number of neurons in layer l, M_l	M_l=50 for all layers		
Activation functions	**Randomly selected in 10 functions.**		
	No.	Functions	Description
	1	$\emptyset(\mathrm{x}) = \frac{1}{1+e^{-x}}$	sigmoid
	2	$\emptyset(\mathrm{x}) = \tanh(\mathrm{x}) = \frac{e^{2x}-1}{e^{2x}+1}$	tanh
	3	$\emptyset(\mathrm{x}) = \tanh(\mathrm{x}/2) = \frac{e^{x}-1}{e^{x}+1}$	$\tanh(\mathrm{x}/2)$
	4	$\emptyset(\mathrm{x}) = \begin{cases} x^2, -1 < x < 1 \\ x, \text{others} \end{cases}$	x^2 (**Discontinuity**)
	5	$\emptyset(\mathrm{x}) = \ln(1+\lvert\mathrm{x}\rvert)$	ln **(Non derivable)**
	6	$\emptyset(\mathrm{x}) = \max(0,\mathrm{x}) = \begin{cases} 0, x < 0 \\ x, x \geq 0 \end{cases}$	ReLU (**Non derivable**)
	7	$\emptyset(\mathrm{x}) = \sin(0.4x)$	sin
	8	$\emptyset(\mathrm{x}) = \mathrm{atan}(x)$	arctan
	9	$\emptyset(\mathrm{x}) = \frac{x}{1+\lvert x\rvert}$	softsign
	10	$\emptyset(\mathrm{x}) = \begin{cases} \frac{x}{1+x}, x > 4 \\ 0.2x, -4 \leq x \leq 4 \\ \frac{x}{1-x}, x < -4 \end{cases}$	Piecewise function (**Non derivable**)
Object function	Benchmark functions: Normalized Ackley and Adjiman $\begin{cases} y_1 = f(x_1,x_2) = e^{-0.2}\sqrt{x_1^2+x_2^2} + 3\cos 2x_1 + \sin 2x_2 \\ y_2 = f(x_1,x_2) = \cos(x_1)\sin(x_2) - \frac{x_1}{x_2^2+1} \end{cases}$ $x_1 \in [0,10], x_2 \in [0,10]$		
Training data set N_d	10000		
Test data set N_t	A total of 10000.		

Batch process of weights M_b	5200 (No batch process)
Batch process of training data N_b	100
W initialization	Random
Pre-training	None
Iterated operation	**Algorithm S7**
Learn rate lr	1
W_r	Random in [-0.001, 0.001]
Update batch of training data	Randomly selected (shuffling) every 25 iterations (Batch)
Iteration maximum	2,000,000
Training upper limit per iteration (**reuse**)	1

Table S6.

Some parameters of **Experiment 6** of MLP.

Items	**Parameters**
Input dimension, P	2
Output dimension, Q	2
The number of hidden layers, L	3
Weights dimension, S	5200
The number of neurons in layer l, M_l	M_l=50 for all layers
Activation functions	**tanh functions:** $\emptyset(\mathrm{x}) = \tanh(\mathrm{x}) = \frac{e^{2x}-1}{e^{2x}+1}$
Object function	Benchmark functions: Normalized Ackley and Adjiman $\begin{cases} y_1 = f(x_1, x_2) = e^{-0.2\sqrt{x_1^2 + x_2^2}} + 3\cos 2x_1 + \sin 2x_2 \\ y_2 = f(x_1, x_2) = \cos(x_1)\sin(x_2) - \frac{x_1}{x_2^2 + 1} \end{cases}$ $x_1 \in [0,10], x_2 \in [0,10]$
Training data set N_d	10000
Test data set N_t	A total of 10000.
Batch process of weights M_b	5200 (No batch process)
Batch process of training data N_b	100
W initialization	Random
Pre-training	None
Iterated operation	**BP** **Algorithm S7**
Learn rate lr	1 for S7 0.001 for BP(The learning rate is too large, the network is not easy to converge.)
W_r	Random in [-0.001, 0.001]
Update batch of training data	Randomly selected (shuffling) every 25 iterations (Batch)
Iteration maximum	2,000,000
Training upper limit per iteration (**reuse**)	1

Supplementary Figures

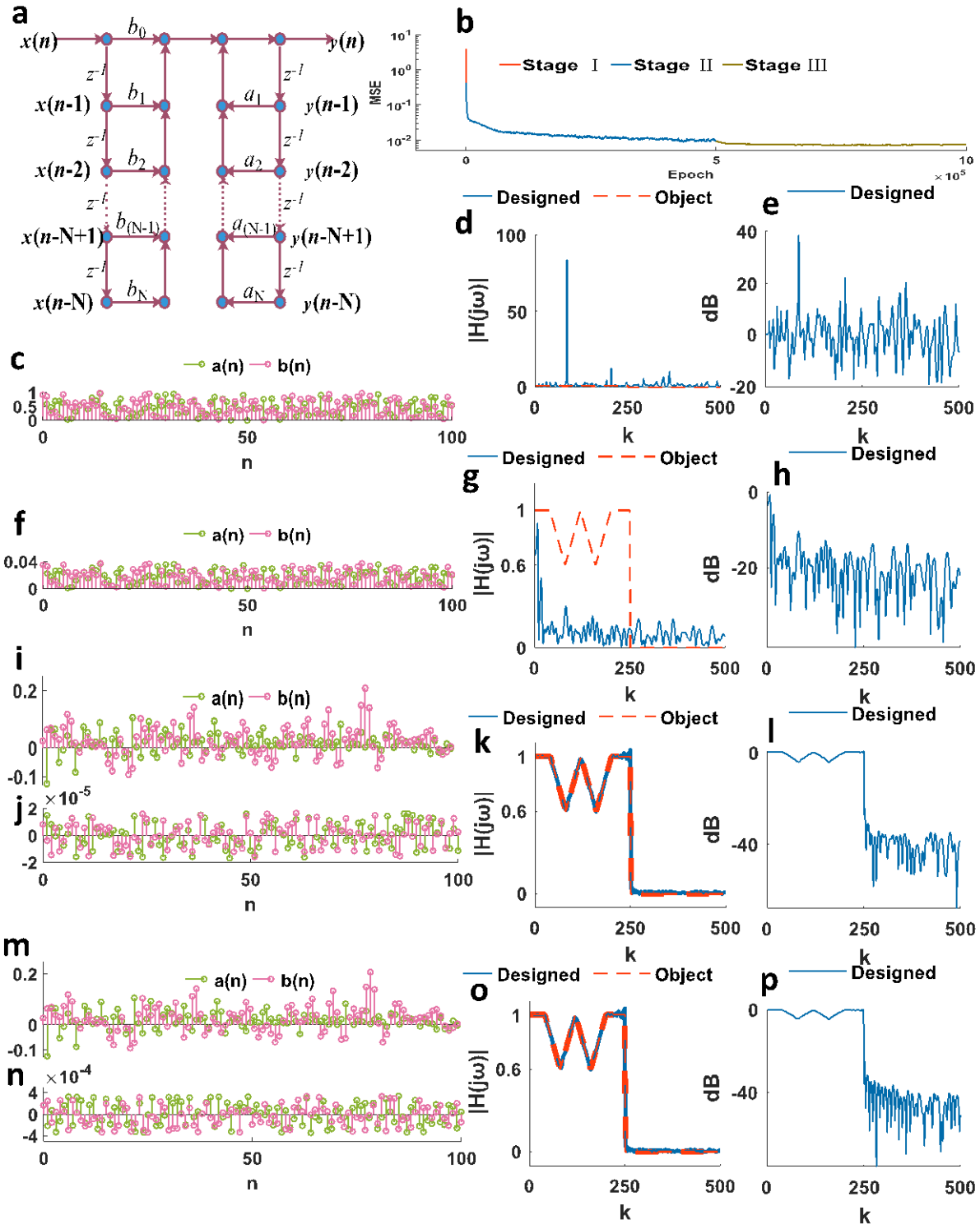

Fig. S1. The results of Experiment 1 on IIR design. Stage I is pre-training in Algorithm S1; Stage II is the iterative training in epoch 500,000, learn rate 0.1, in Algorithm S3; Stage III is the iterative training in epoch 1000,000, learn rate 0.01, in Algorithm S3. **a,** The system structure of classical direct-type *N*th-order IIR Filter, input $x(n)$ and output $y(n)$. **b,** The convergence results of Experiment 1 on IIR design. **c,** Randomly initialized system parameters $\{a_k\}$ and $\{b_k\}$. **d,** Amplitude-frequency characteristic curve $\left|H(e^{j\omega})\right|$ of randomly initialized system. **e,** Bode plot for the magnitude of the frequency response ($20\lg\left|H(e^{j\omega})\right|$) of randomly initialized system. **f,** System parameters after pre-training (Stage I). **g,** $\left|H(e^{j\omega})\right|$ after Stage I. **h,** $20\lg\left|H(e^{j\omega})\right|$ after Stage I. **i,** System parameters after Stage II. **j,** Adjusted value of system parameters in epoch 500,000. **k,** $\left|H(e^{j\omega})\right|$ after Stage II. **l,** $20\lg\left|H(e^{j\omega})\right|$ after Stage II. **m,** System parameters after Stage III. **n,** the Adjusted value of system parameters in epoch 1000,000. **o,** $\left|H(e^{j\omega})\right|$ after Stage III. **p,** $20\lg\left|H(e^{j\omega})\right|$ after Stage III.

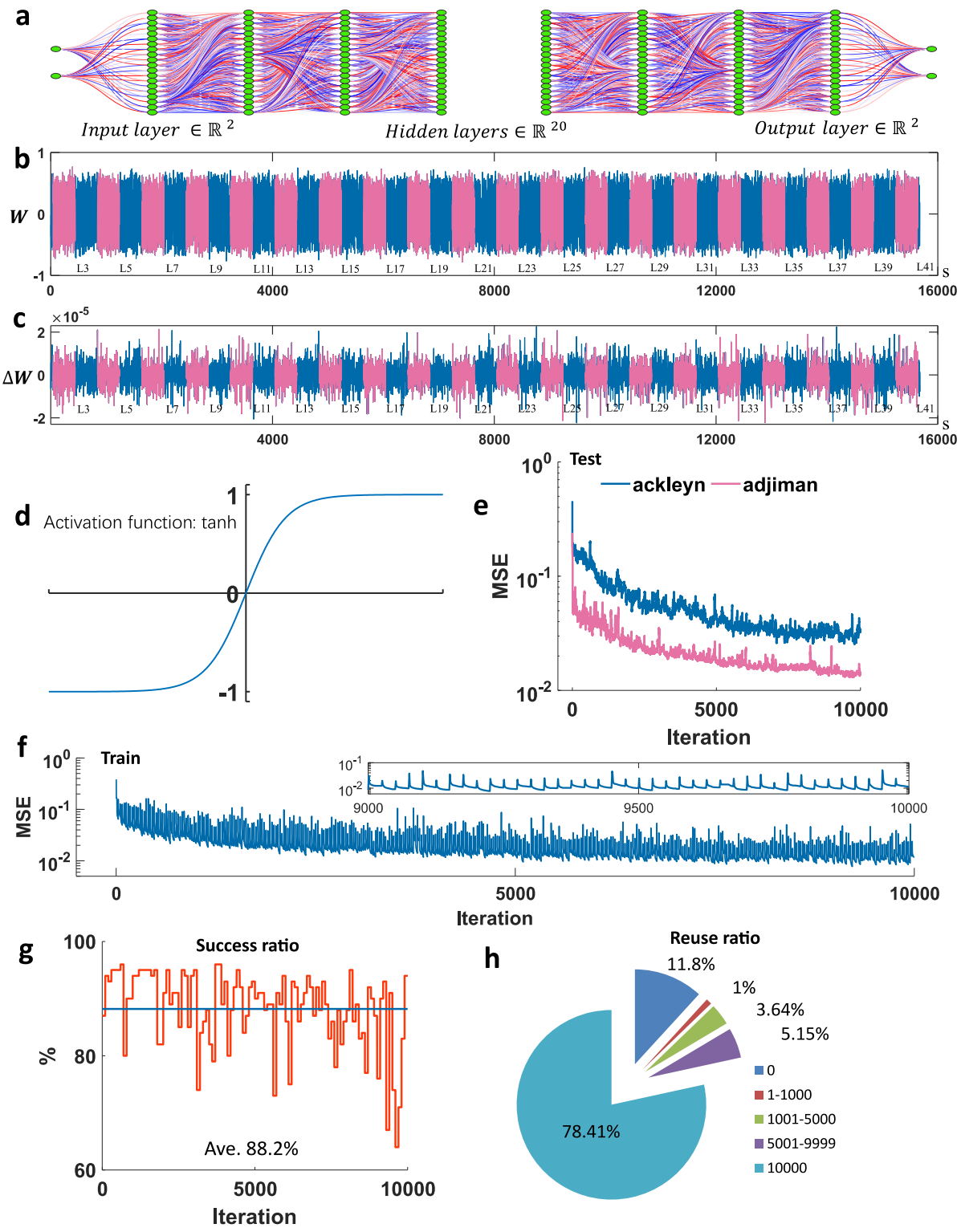
a
Input layer ∈ ℝ 2
Hidden layers ∈ ℝ 20
Output layer ∈ ℝ 2
b
W
c
ΔW
×10-5
d
Activation function: tanh
e
Test
ackleyn
adjiman
MSE
Iteration
f
Train
MSE
Iteration
g
Success ratio
Ave. 88.2%
%
Iteration
h
Reuse ratio
11.8%
1%
3.64%
5.15%
78.41%
0
1-1000
1001-5000
5001-9999
10000

Fig. S2. The results of Experiment 2 on DNN design, in 10,000 iterations, learning rate 0.005, and Algorithm S6. a, The structure of DNN. **b,** The network weighs in 10,000 iterations. **c,** The adjusted value of network weights in 10,000 iterations. **d,** The activation function of neurons is tanh. **e,** Two benchmark function output convergence in MSE of the test. **f,** Training MSE. MSE continuously reduced for each update training data batch per 25 iterations. **g,** The success ratio with iteration. The ratio is calculated once per 100 iterations. The whole average success ratio is 88.2% as shown in the blue line. **h,** The reuse situation of $\mathbb{C}_X$. "0" means the current $\mathbb{C}_X$ is fail to train. "10000 times" is the preset up-limit of reuse.

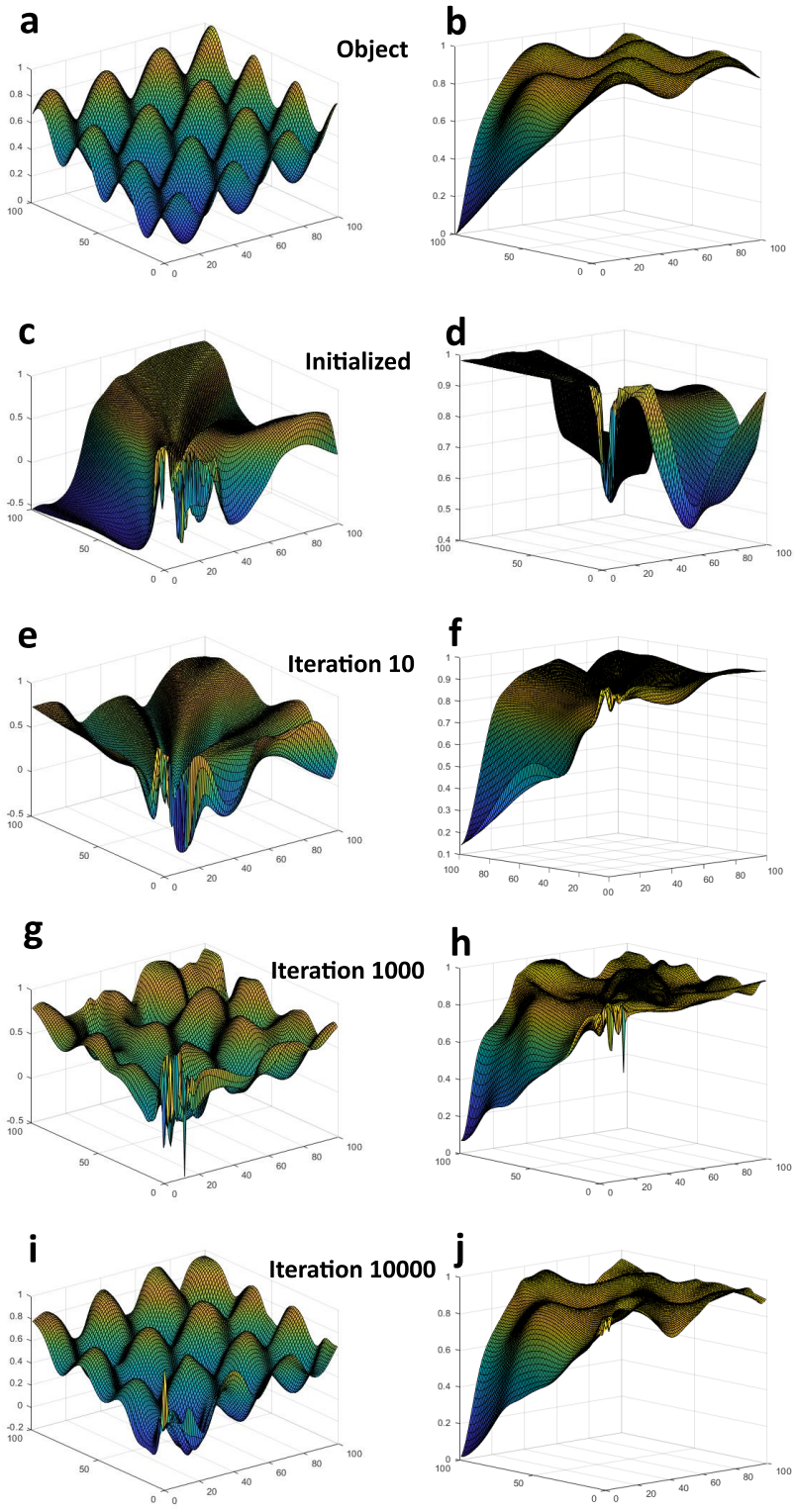

Fig. S3. The results of Experiment 2 on DNN design (The supplement of Fig.S2.). a, The benchmark function: Ackley. **b,** The benchmark function: Adjiman. **c,** Initialized network output

of Ackley with the MSE 0.4536. **d,** Initialized network output of Adjiman with the MSE 0.2399. **e,** Network output of Ackley with the MSE 0.1906 in iteration 10. **f,** Network output of Adjiman with the MSE 0.0503 in iteration 10. **g,** Network output of Ackley with the MSE 0.0922 in 1000 iterations. **h,** Network output of Adjiman with the MSE 0.0305 in 1000 iterations. **i,** Network output of Ackley with the MSE 0.0323 in 10000 iterations. **j,** Network output of Adjiman with the MSE 0.0130 in 10000 iterations.

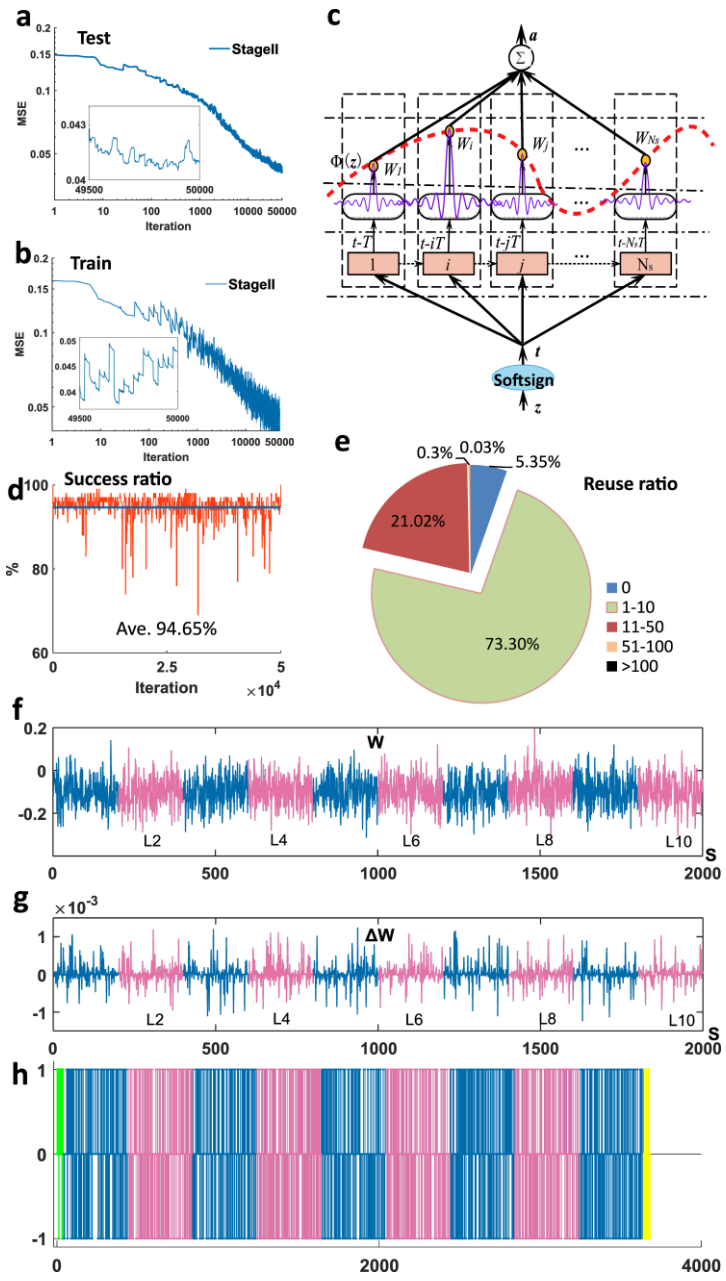

Fig. S4. The results of Experiment 3 on SaNN design, in iteration 50,000, learning rate 0.01, Algorithm S1, and Algorithm S6. a, benchmark function output convergence in MSE of the test. **b,** MSE of training. Update training data batch per 25 iterations. **c,** The structure of SISO SaNN. **d,** The success ratio with iteration. The ratio is calculated once per 100 iterations. The whole average success ratio is 88.2% as shown in the blue line. **e,** The reuse situation of $\mathbb{C}_X$. "0" means the failure calculation of the current $\mathbb{C}_X$. "10000 times" is the preset up limit of reuse. **f,** The network weights in iteration 10,000. **g,** The adjusted value of network weights in 10,000 iterations. **h,** Inter-layer link weights between different layers. A total of 3680, initialized as "0" has 1802(48.97%), "1" has 905(24.59%), and "-1" has 973 (26.44%).

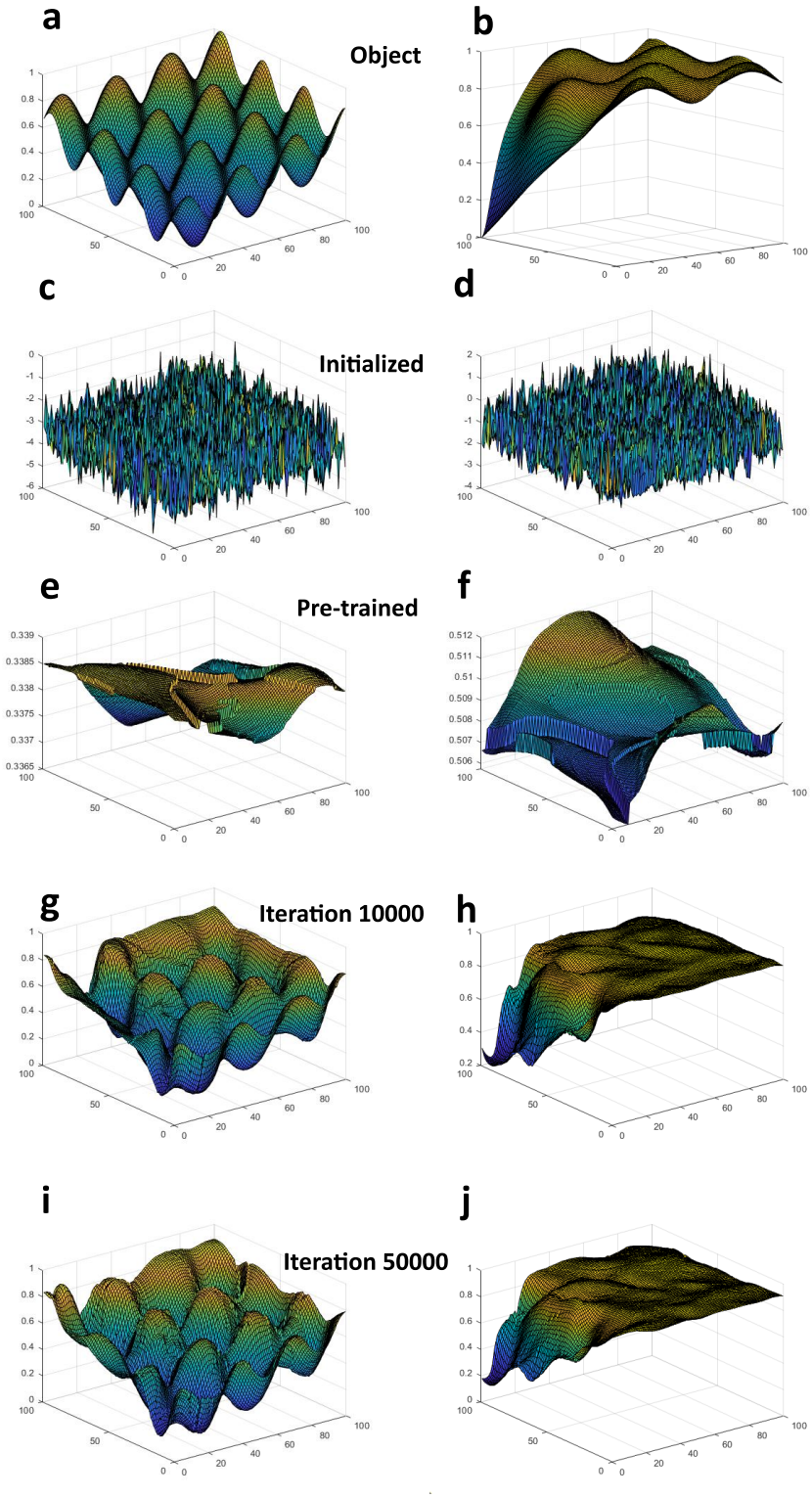

Fig. S5. The results of Experiment 3 on SaNN design (The supplement of Fig.S4.). Test the whole 10000 test dataset, different from the test in Fig.3 just 100 randomly selected test data. **a,**

The benchmark function: Ackley. **b,** The benchmark function: Adjiman. **c,** Initialized network output of Ackley with the MSE 0.7417. **d,** Initialized network output of Adjiman with the MSE 0.7154. **e,** Network output of Ackley with the MSE 0.1844 by pre-train in Algorithm S1. **f,** Network output of Adjiman with the MSE 0.1380 by pre-train in Algorithm S1. **f,** Network output of Ackley with the MSE 0.0696 in 10000 iterations in Algorithm S6. **h,** Network output of Adjiman with the MSE 0.0467 in 10000 iterations in Algorithm S6. **i,** Network output of Ackley with the MSE 0.0499 in 50000 iterations in Algorithm S6. **j,** Network output of Adjiman with the MSE 0.0410 in 50000 iterations in Algorithm S6.

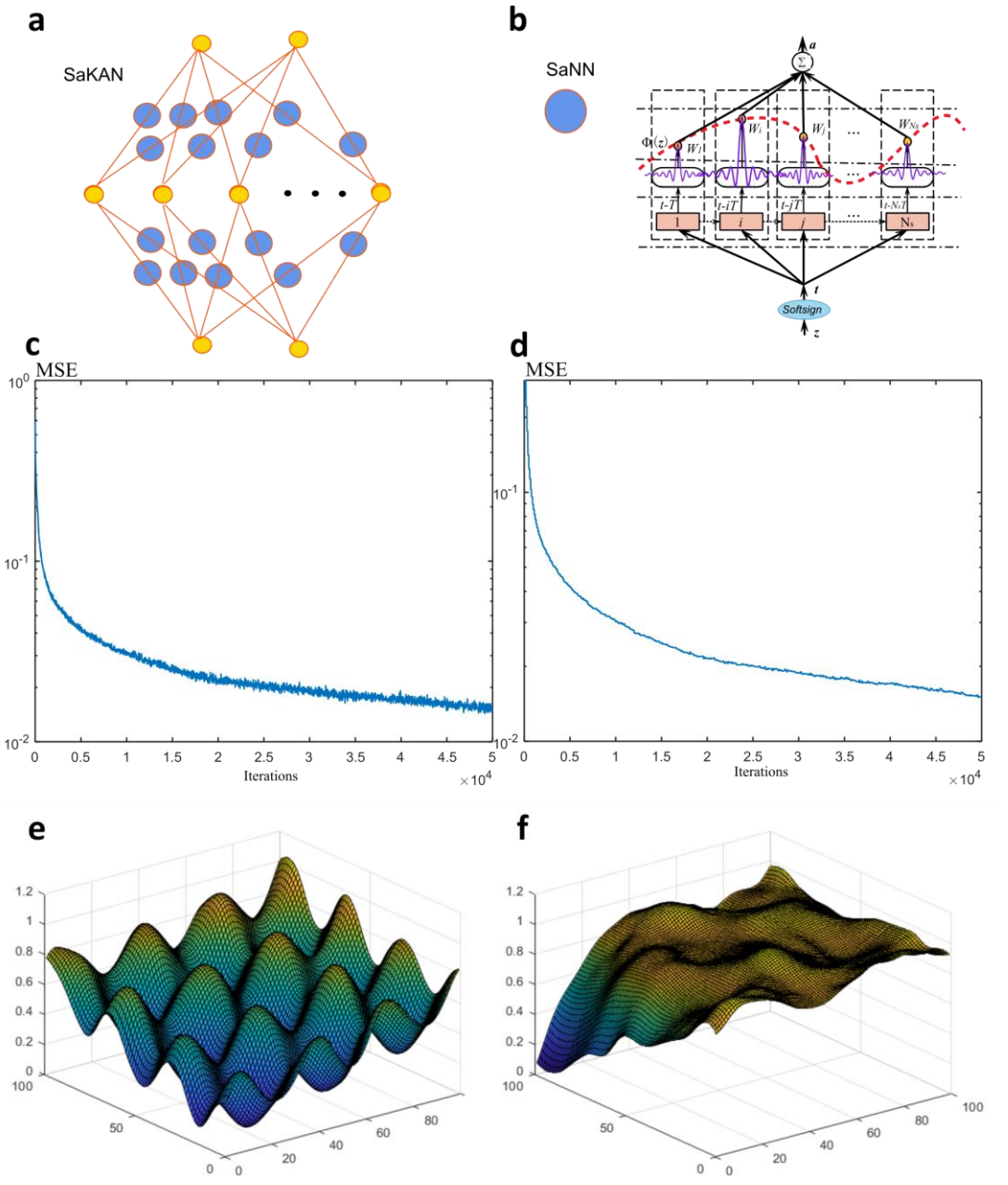

Fig. S6. The results of Experiment 4 on SaKAN design. a, The structure of SaKAN. **b,** The structure of SISO SaNN. **c,** MSE of training. **d,** MSE of testing. **e,** Network output of Ackley with the MSE 0.0155 in 50000 iterations in Algorithm S7. **f,** Network output of Adjiman with the MSE 0.0142 in 50000 iterations in Algorithm S7.

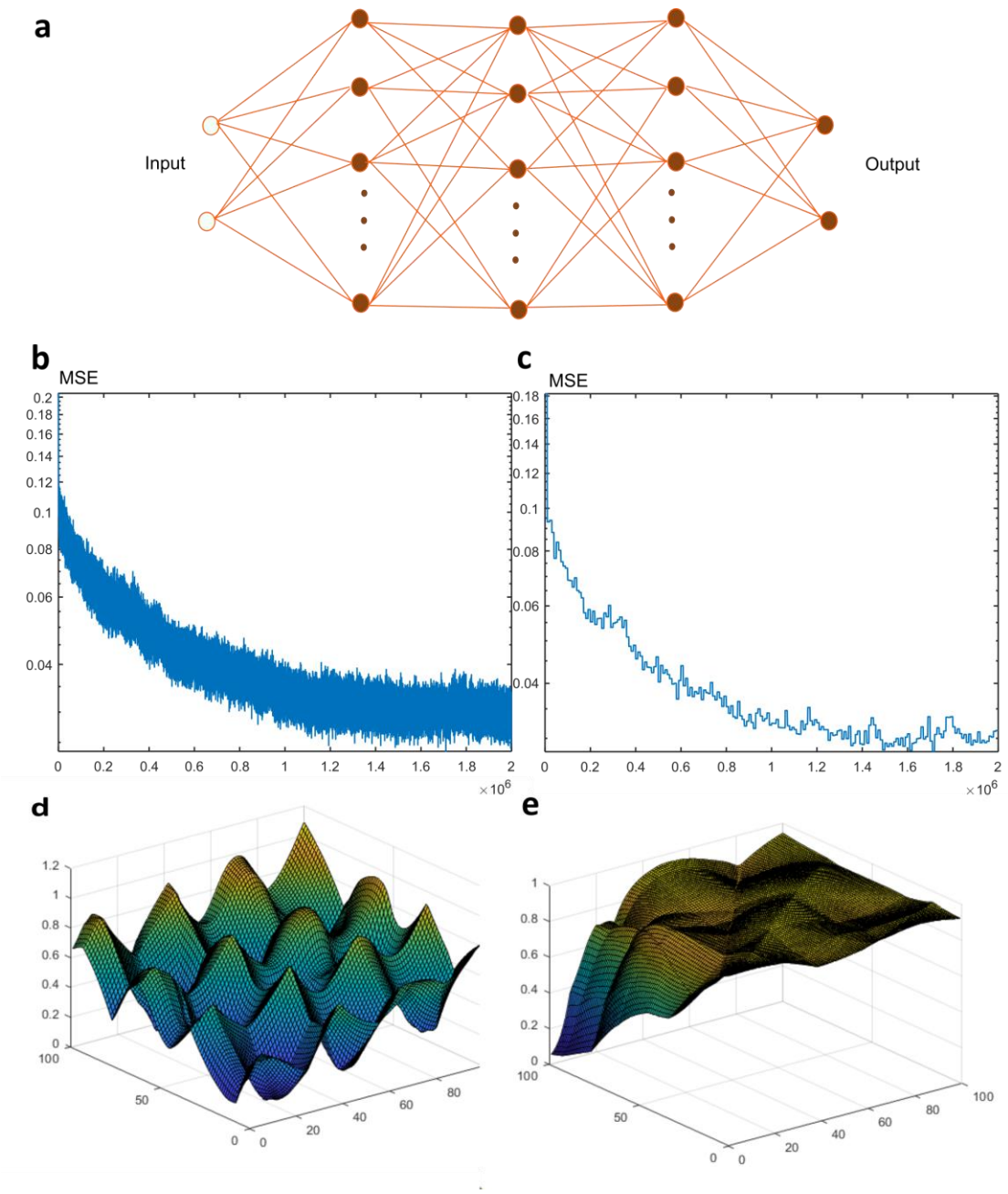

Fig. S7. The results of Experiment 5 on RAN design. a, The structure of RAN. The solid circle represents neurons with randomly selected activation functions. **b,** MSE of training. **c,** MSE of testing. **d,** Network output of Ackley with the MSE 0.0324 in 2,000,000 iterations in Algorithm S7. e, Network output of Adjiman with the MSE 0.0265 in 2,000,000 iterations in Algorithm S7.

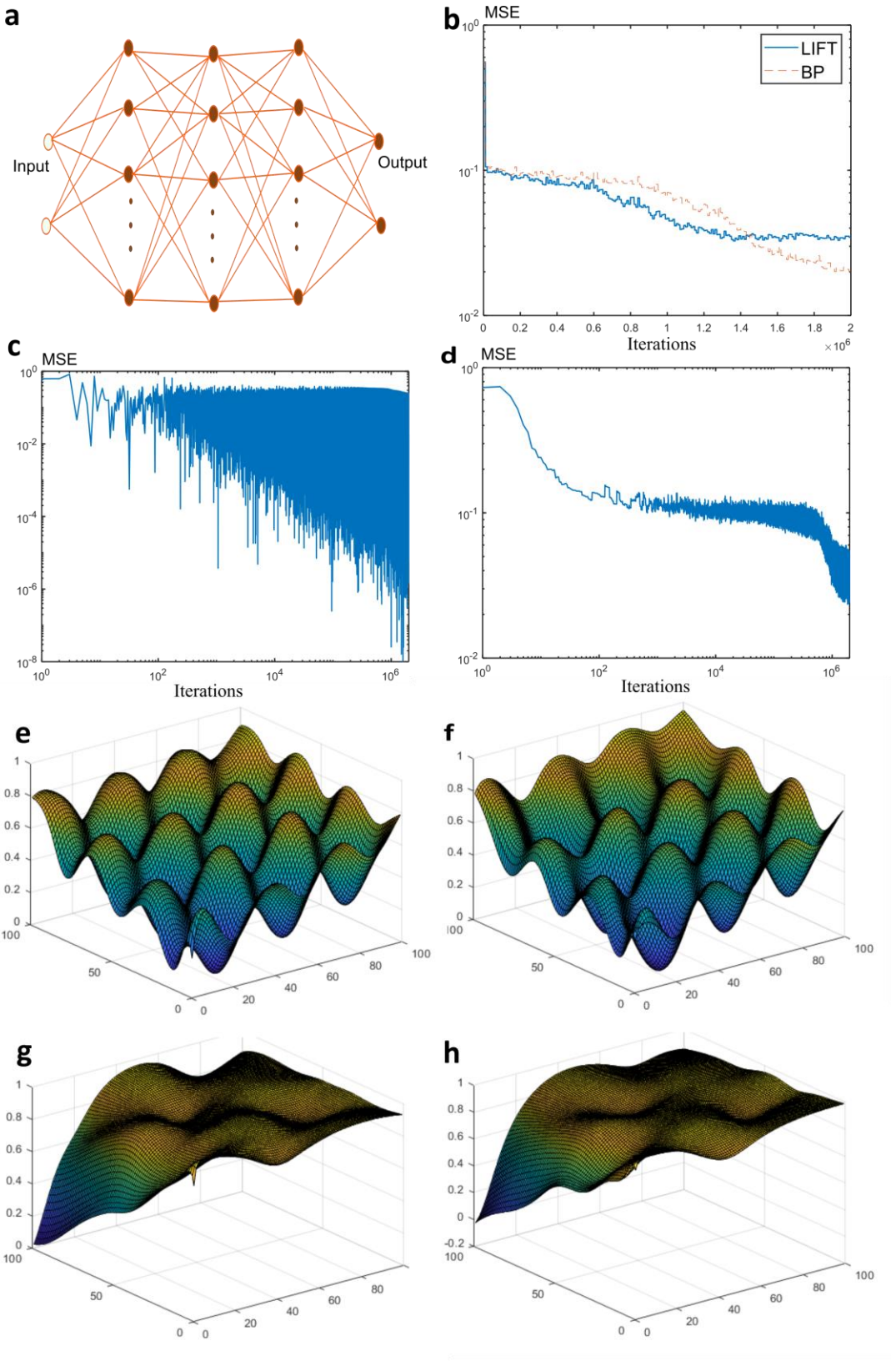
a
Input
Output
b
MSE
LIFT
BP
Iterations
c
MSE
Iterations
d
MSE
Iterations
e
f
g
h

Fig. S8. The results of Experiment 6 on MLP design. a, The structure of MLP with activation functions tanh . **b,** MSE of training. **c,** MSE of testing in BP. **d,** MSE of testing in LIFT. **e,** Network output of Ackley with the MSE 0.0239 in 2,000,000 iterations in BP. **f,** Network output of Ackley with the MSE 0.0391 in 2,000,000 iterations in LIFT S7. **g,** Network output of Adjiman with the MSE 0.0181 in 2,000,000 iterations in BP.**h,** Network output of Adjiman with the MSE 0.0256 in 2,000,000 iterations in LIFT S7.

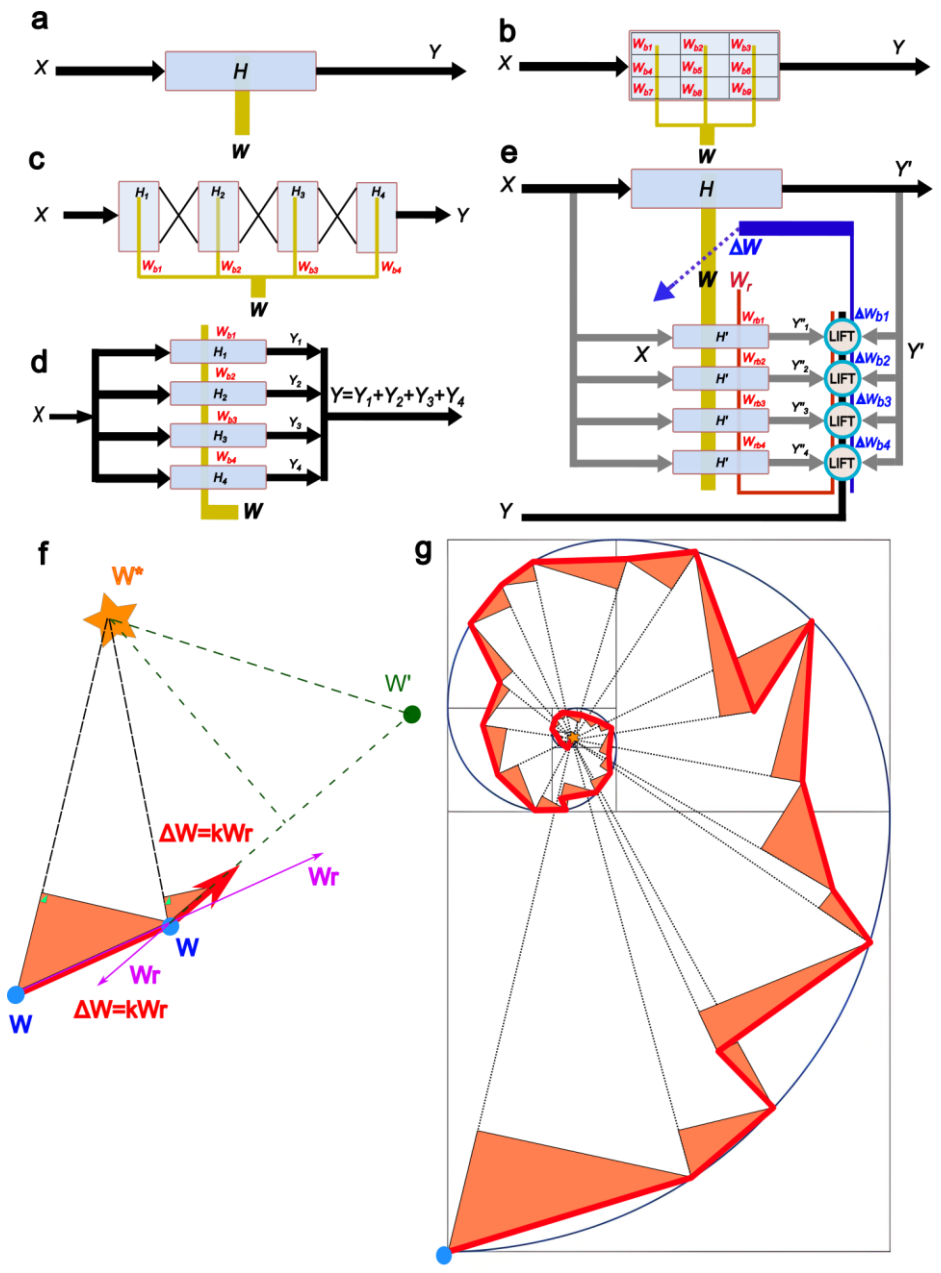

Fig. S9. Diagrammatic sketch. a, The structure diagram of parametric system indicated by $\boldsymbol{H}$. **b,** The sub-region structure with batch processes($\boldsymbol{W_{b1}}$ to $\boldsymbol{W_{b9}}$). **c,** The hierarchical structure with batch processes($\boldsymbol{W_{b1}}$ to $\boldsymbol{W_{b4}}$). **d,** The parallel structure with batch processes($\boldsymbol{W_{b1}}$ to $\boldsymbol{W_{b4}}$). **e,** The parallel training diagram with batch processes($\boldsymbol{W_{b1}}$ to $\boldsymbol{W_{b4}}$). $\boldsymbol{H'}$ is the copy(reuse module) of $\boldsymbol{H}$. See Algorithm S5, S6. **f,** Sketch of LIFT method. $\boldsymbol{W}$ is the current parameters; $\boldsymbol{W^*}$ is the expected parameters; the red solid lines represent the weight-adjusted trajectory $\boldsymbol{\Delta W}$; the black dashed lines represent the ideal tuning direction.

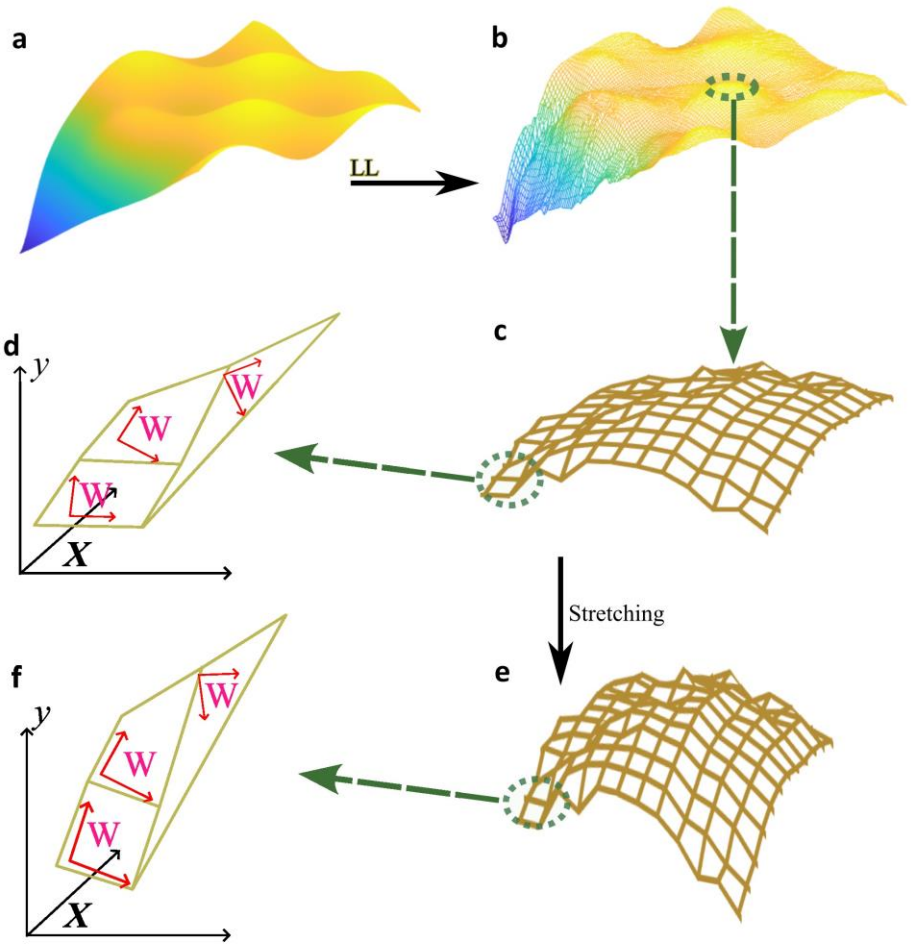

Fig. S10. The diagrammatic sketch of visualized understanding of LL-IC. a, System space. **b-d,** System approximation by local linearization. **e-f,** The diagrammatic sketch of isomorphism comparability. System approximation after stretching caused by weight adjustment. In the process of stretching and torsion, the change of gradient is often limited. This is also the reason why $\mathbb{C}_{\boldsymbol{X}}$ is effective. It's also the reason of its reuse.