Dynamics-inspired Neuromorphic Visual Representation Learning

Zhengqi Pei [1 2] Shuhui Wang [1 3]

Abstract

This paper investigates the dynamics-inspired neuromorphic architecture for visual representation learning following Hamilton's principle. Our method converts weight-based neural structure to its dynamics-based form that consists of finite sub-models, whose mutual relations measured by computing path integrals amongst their dynamical states are equivalent to the typical neural weights. Based on the entropy reduction process derived from the Euler-Lagrange equations, the feedback signals interpreted as stress forces amongst sub-models push them to move. We first train a dynamics-based neural model from scratch and observe that this model outperforms traditional neural models on MNIST. We then convert several pre-trained neural structures into dynamics-based forms, followed by fine-tuning via entropy reduction to obtain the stabilized dynamical states. We observe consistent improvements in these transformed models over their weight-based counterparts on ImageNet and WebVision in terms of computational complexity, parameter size, testing accuracy, and robustness. Besides, we show the correlation between model performance and structural entropy, providing deeper insight into weight-free neuromorphic learning.

1. Introduction

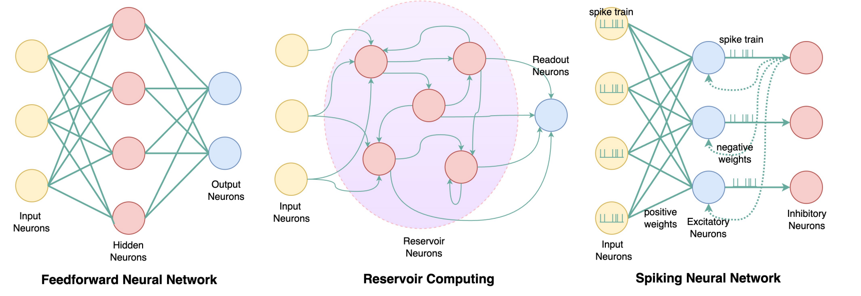
A biological brain learns by both the structural evolution via rewiring neural pathways (Chklovskii et al., 2004) and the numerical evolution via strengthening/weakening neural connections (Cho et al., 2015). Following the rule of biological neurons, the artificial neural networks (ANNs) mimic the biological brain with neurons organized in a fixed layered structure as the fully connected neural network (Hinton et al., 2006), CNN (Krizhevsky et al., 2017) and Transformers (Liu et al., 2021). In real applications such as image categorization, different neural structures lead to varying performance on widely-used datasets (Golubeva et al., 2020) such as ImageNet (Deng et al., 2009). Despite the great success, ANNs have several intrinsic drawbacks, *e.g.*, the requirement of a massive number of parameters, gradient vanishing or explosion (Pascanu et al., 2013), and redundant computations (RoyChowdhury et al., 2018). The fixed structure of ANNs is considered suboptimal to approximate the brain precisely and efficiently (Han et al., 2021).

It has been proved that evolving neural mechanisms, such as NeuroEvolution (Stanley & Miikkulainen, 2002) and neural architecture search (NAS) (Elsken et al., 2019) that alter the layers and parameters, can significantly outperform their counterparts with fixed structures (Assunção et al., 2018). Nonetheless, these mechanisms established as dynamic neural networks (Han et al., 2021) require a large amount of expensive fitness evaluations that are inefficient for real-time learning (Stork et al., 2019) and appear unstable. As another direction of research endeavors, some attempts, *e.g.*, weight agnostic neural network (Gaier & Ha, 2019), Spiking neural networks (Basegmez, 2014) and weight mirror (Akrout et al., 2019), reconsider the importance of neural weight and structure. They either rely on explicit neural weight computation or have yet to develop an efficient learning mechanism to, for example, overcome the difficulties of training a spiking neural network via supervised learning (Huynh et al., 2022), which are still inefficient due to the enormous search space of model structure (Ren et al., 2021). In general, the weight-based ANNs can hardly achieve unified structural and numerical evolutions.

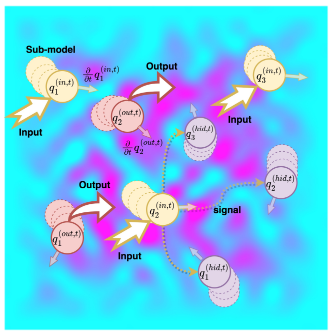
In this study, we consider the structure and numerical learning from the neuromorphic dynamics aspect, inspired by Hebb's learning rule (Cooper, 2005), which states that neural connections between neurons with similar dynamic behaviors tend to be stronger. Rather than explicitly implementing Hebb's rule of the neural connections as weights, we represent the dynamic behaviors of neurons (neuronal dynamics) as their spatial coordinates. Neurons with similar dynamic behaviors have closer spatial coordinates, leading to naturally stronger connections. To formalize the whole mechanism, we first reinterpret the universal approximation

[1]Institute of Computing Technology, Chinese Academy of Sciences [2]School of Artificial Intelligence, University of Chinese Academy of Sciences [3]Peng Cheng Laboratory. Zhengqi Pei <peizhengqi22@mails.ucas.ac.cn>. Correspondence to: Shuhui Wang <wangshuhui@ict.ac.cn>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).



(a) **Weights-based neural networks**: weights are trainable variants that are explicitly isolated from each other; they are treated as the essential parameters directly affected by the feedback signals, such as predictive error during the back-propagation process.



(b) **Dynamics-inspired neuromorphic system**: neural weights are path integrals between neuronal dynamics.

Figure 1: **Comparison between neural networks and dynamics-inspired neuromorphic system.** We interpret the neurons as sub-models $q_i^{(l,t)}$ embedded in the high-dimensional neuronal state space. The indices l and t refer to a subsystem's index and time step. In Fig. 1b, sub-models of different subsystems, corresponding to input, hidden, and output layers, are mixed in the neuronal state space. The feedforward and feedback signals push the neurons to continue moving until equilibrium. The neuronal dynamics determine the nonlinear spatial "density" nearby, affecting the signals traveling among neurons in a curved/nonlinear manner. The area in pink indicates higher density, while the area in cyan indicates lower density.

theorem (UAT) (Scarselli & Tsoi, 1998) into a dynamical alternative, which claims that neural weights can be calculated as the covariant of neuronal states while retaining the approximation capacity of the whole neural system. In the dynamical UAT (Eq. 2), neurons receive the input signals that affect neuronal dynamics and emit the processed signals to other neurons. The signals between neurons amplify or decay during transmission via path integral. During training, the neurons move until their neuronal states reach equilibrium. In this way, the trainable units are the neuronal states. The neural weights between neurons are not necessarily maintained as concrete trainable units since they can be expressed as path integrals between neuronal states.

Accordingly, we propose a Dynamics-inspired Neuromorphic (DyN) learning framework where the trainable parameters are the neuronal dynamics (Figure 1b). DyN applies dynamics-based updating rules on finite sub-models, *i.e.*, the functional neurons receiving and emitting signals with neuronal states changing dynamically. The model learning and inference are undertaken via computing the path integral between the neuronal dynamics. Computationally, it allows one to change coordinates of distinct neurons efficiently with functional integrals (Weinberg, 1995), facilitating a more global interaction among neurons, compared to the layer-by-layer update rules used in deep learning architectures (Pascanu et al., 2013).

Experimentally, we first validate DyN on MNIST (Deng, 2012) to verify its capacity as a universal approximator. As in the case of LeNet-5 (LeCun et al., 2015), we observe that when we transform both the convolutional and fully-connected layers to their DyN form, the parameters have been reduced by $10 \sim 30$, yet the transformed DyN models outperform the original LeNet-5 by 0.2%. Then, we transform the weight-based layers of many pre-trained deep neural models, *e.g.*, DenseNet (Huang et al., 2017) and SwinT (Liu et al., 2021), to the DyN alternatives, followed by fine-tuning on ImageNet (Deng et al., 2009). As a result, the parameters have been reduced by $5 \sim 10$. Still, the transformed models outperform the original ones on both ImageNet and WebVision (Li et al., 2017). These observations reveal the potential of building a more efficient neural computing architecture focusing on neuronal dynamics rather than weight transport.

We also consider practical implementation issues assuming that existing computing devices naturally contain noises (Braverman et al., 2015), *e.g.*, the quantization error of digitization, affecting the parameter precision during storage and calculation. Specifically, we assume a model's parameter space is reconstructed by noise uniformly on an ϵ-ball. We round all parameters to a certain precision, *e.g.*, round a 5-digit value to a 2-digit value, to see how the performance of the "quantized" model is affected. The results indicate that our model is robust to different noise levels, suggesting that we can accelerate the model via hashing without a significant loss of accuracy. Codes are available[1].

2. Preliminaries

Interpreting ANN as a dynamical system. A typical ANN comprises neurons organized in a specific structure and trainable neural weights connecting them. The weights

[1]https://github.com/pzqpzq/flat-learning

among different neurons keep updating during training with layer-by-layer updating rules. According to the dynamics theory, an ANN is a dynamic system where the neurons dynamically interact towards a minimal objective function, *e.g.*, the cross-entropy loss. However, ANN's fixed structure seems suboptimal because it constrains the learning toward a universally optimal network configuration and imposes an unnecessary computational burden on the training and inference. In comparison, our method is straightforward. By treating each neuron state as a basic trainable unit, we *replace the neural weights between neurons with the dynamic interaction between neuronal dynamics*. Neural weight values are measurements of the transient interaction between neurons, which are directly accessible from the dynamical states of neurons. Neurons are allowed to interact fully with each other during training, facilitating a comprehensive release of model capacity.

Sub-models and subsystems. A neuron's dynamical state, *e.g.*, spatial location, velocity, acceleration, activation/inhibition, *etc.*, determines its spatial coordinates in a d-dimensional phase space. Neurons with similar behaviors are located closely in the phase space. We call these dynamical neurons as *sub-models* to distinguish them from node-like neurons in the computational sense. A sub-model is a functional neuron receiving and/or emitting signals and changing its dynamical states. A group of sub-models sharing identical global settings, *e.g.*, a hidden layer in an MLP or a convolutional layer in a CNN, refers to a *subsystem*.

We interpret the dynamical states of a sub-model with index i as a time-variant embedding: $q_i^{(l,t)} \in \mathbb{R}^d$, where t refers to the time-step, and l refers to the index of *subsystem* that contains the sub-model. In Figure 2, we define two vector fields $E_i^{(l,t)}, R_i^{(l,t)} : \mathbb{R}^d \mapsto \mathbb{R}^s$. For instance, $E_i^{(l,t)}$ converts a direction $v \in \mathbb{R}^d$ into a signal $E_i^{(l,t)}(v) \in \mathbb{R}^s$. Intuitively, we have $R_j^{(t^*)}(\mathbf{0}) = \sum_i S_{ij}^{(t,t^*)}(q_j^{(t^*)})$ and $E_i^{(t)}(\mathbf{0}) = \sum_j S_{ij}^{(t,t^*)}(q_i^{(t)})$. We define $R_j^{(t^*)}(\mathbf{0}) \equiv R_j^{(t^*)}$ and $E_i^{(t)}(\mathbf{0}) \equiv E_i^{(t)}$ for simplicity, and we set $E_i^{(t)}(u) = E_i^{(t)}(v), u \neq v$, assuming that a sub-model emits signal isotropically in any direction. We also define a mapping $S_{ij}^{(t,t^*)} : \mathbb{R}^d \mapsto \mathbb{R}^s$ that describes how the signal emitted from $q_i^{(t)}$ is varying along the path towards $q_j^{(t^*)}$, *e.g.*, $S_{ij}^{(t,t^*)}(v) = E_i^{(t)} - \|v - q_i^{(t)}\|_p / \|q_j^{(t^*)} - q_i^{(t)}\|_p$.

Accordingly, we can transform any tensor-formed neural layer into subsystems with a specified topology. As presented in Table 1, a sub-model refers to a neuron for an MLP's fully-connected layer M_{FC}. A convolution layer M_C (kernel's window $k \times k$, with N_{in} and N_{out} channels) refers to $2k$ subsystems, each containing $N_{in} + N_{out}$ sub-models. An attention layer with $M_Q, M_K \in \mathbb{R}^{T \times d_k}$ and $M_V \in \mathbb{R}^{T \times d_v}$, where d_k and d_v are the hidden dimensions

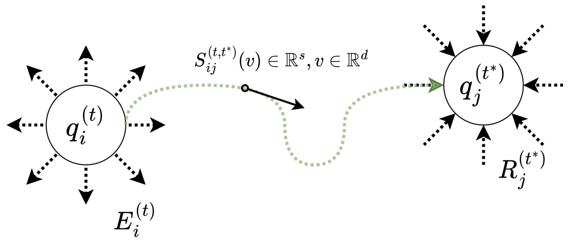


Figure 2: **Signal is varying along with the nonlinear path between sub-models.**

and T is the sequence length, refers to two subsystems containing $2d_k + d_v$ and T sub-models, respectively. We list the terminologies of a typical *DNN* to the related concepts of *DyN* in Appendix B.

Table 1: **From neural layer to DyN.** We denote $P(x)$ as a subsystem containing x sub-models.

Models	Layer Types	DyN Types
MLP	$M_{FC} \in \mathbb{R}^{m \times n}$	$P(m)$+$P(n)$
CNN	$M_C \in \mathbb{R}^{k \times k \times N_{in} \times N_{out}}$	$2k \cdot P(N_{in} + N_{out})$
Transformer	$M_Q \in \mathbb{R}^{T \times d_k}$ $M_K \in \mathbb{R}^{T \times d_k}$ $M_V \in \mathbb{R}^{T \times d_v}$	$P(2d_k + d_v)$+$P(T)$

Universal Approximation Theorem (UAT). This part focuses on Cybenko's arbitrary-width UAT (Cybenko, 1989), which demonstrates the approximation capabilities of a feed-forward neural network in the space of continuous functions between two Euclidean spaces. It states that, $\sigma : \mathbb{R} \mapsto \mathbb{R}$ *is not polynomial if and only if for every* $n \in \mathbb{N}$, $m \in \mathbb{N}$, *compact* $K \subseteq \mathbb{R}^n$, $f : K \mapsto \mathbb{R}^m$, $\varepsilon > 0$, *there exists* $k \in \mathbb{N}$, $A \in \mathbb{R}^{k \times n}$, $b \in \mathbb{R}^k$ *and* $C \in \mathbb{R}^{m \times k}$ *such that*

$$\sup_{x \in K} \|f(x) - g(x)\| < \varepsilon \tag{1}$$

where $g(x) = C \cdot (\sigma(A \cdot x + b))$. There are various UAT for the arbitrary-depth case and some widely used architectures like convolutional neural networks. However, they all rely on a setting in that the trainable units are neural weights between neurons. Next, we will present an alternative to this typical UAT by considering the neuronal states rather than their weights as the trainable units.

Principle of dynamic subsystems. Based on Cybenko's UAT, we propose a *dynamical UAT* with trainable neuronal states. The *dynamical UAT* can approximate a time-variant sequential function. It states that, *for* $d, s, M, N \in \mathbb{N}$, *given a system of sub-models with a set of time-variant coordinates* $\{q_i^{(t)} \in \mathbb{R}^d, i \in [1, N]\}$ *that receive and emit time-variant signals* $R_i^{(t)} \in \mathbb{R}^s$ *and* $E_i^{(t)} \in \mathbb{R}^s$, *then for* *arbitrary nonlinear sequential mapping $\mathbb{R}^{s\times M} \mapsto \mathbb{R}^{s\times M}$ between $R_i^{(t)}$ and $E_i^{(t)}$ for any $i \in [1, N]$, there exists a set of matrices $\mathcal{A} \in \mathbb{R}^{s\times s}$, $\mathcal{B} \in \mathbb{R}^{s\times d}$, $\mathcal{C} \in \mathbb{R}^{s\times d}$, $\mathcal{D} \in \mathbb{R}^{d\times s}$, $\mathcal{E} \in \mathbb{R}^{d\times s}$ and $\mathcal{F} \in \mathbb{R}^{d\times d}$, such that for $t \in [1, M]$:*

$$\begin{aligned} R_i^{(t)} &= \sum_{j\neq i}^{N} E_j^{(t-\epsilon)} \cdot \varphi(q_j^{(t-\epsilon)}, q_i^{(t)}) \\ E_i^{(t)} &= \mathcal{A}R_i^{(t)} + \mathcal{B}q_i^{(t)} + \mathcal{C}\frac{d}{dt}q_i^{(t)} \\ \frac{d}{dt}q_i^{(t)} &= \mathcal{D}R_i^{(t)} + \mathcal{E}E_i^{(t)} + \mathcal{F}q_i^{(t)} \end{aligned} \tag{2}$$

where a non-polynomial bi-linear mapping $\varphi : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ is used to compute the path integral between sub-models.

The proof is presented in Appendix C. The *dynamical UAT* considers neural weights as the covariants of the trainable neuronal states, implying the equivalence between a weight-based neural structure and a neuron-state-based one. The time-invariant parameter matrices $\{\mathcal{A}, ..., \mathcal{F}\}$ describe a sub-model's emitting and receiving mechanism. There is no activation function in Eq. 2 because φ already introduces nonlinearity. As a result, the dynamics amongst trainable neurons are sufficient to approximate arbitrary time-variant sequential functions on a specific metric function φ. Despite that, one can still use the nonlinear activation functions on the summation of received signals in practical implementation to introduce more nonlinearity and further enhance the stability of the whole learning system.

Formulation of a *DyN* system. Based on Eq. 2, we formalize the *DyN* system that concentrates on learning neuronal dynamics rather than neural weights. The proposed DyN system contains subsystems $\{P^{(l)}, l \in [1, L]\}$ interpreted as nodes in a directed graph $\mathscr{G}$, and the directed edge from $P^{(l^*)}$ to $P^{(l)}$ indicates that the emitted signals of $P^{(l^*)}$ are received by $P^{(l)}$. Each subsystem contains finite time-variant sub-models with d-dimensional embedding-like dynamical states $\{q_i^{(l,t)} \in \mathbb{R}^d, i \in [1, N_l], N_l \in \mathbb{N}\}$. Each sub-model can emit signals $E_i^{(l,t)} \in \mathbb{R}^s$ and receive signals $R_i^{(l,t)} \in \mathbb{R}^s$. We assume $t^* = t$ and $S_{ij}^{(t,t^*)} = S_{ij}^{(t)}$ for simplicity (see Appendix D). Then we generalize the dynamics among subsystems as follows:

$$\begin{aligned} R_i^{(l,t)} &= \sum_{j=1}^{N_{l*}} \Psi_R^{(l)}(E_j^{(l^*,t)}, q_j^{(l^*,t)}, q_i^{(l,t)}) \\ E_i^{(l,t)} &= \Psi_E^{(l)}(R_i^{(l,t)}, q_i^{(l,t)}, \frac{\partial}{\partial t}q_i^{(l,t)}) \\ \frac{\partial}{\partial t}q_i^{(l,t)} &= \Psi_Q^{(l)}(R_i^{(l,t)}, E_i^{(l,t)}, q_i^{(l,t)}) \end{aligned} \tag{3}$$

where the nonlinear $\Psi_R^{(l)}, \Psi_E^{(l)} : \mathbb{R}^s \times \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}^s$ and $\Psi_Q^{(l)} : \mathbb{R}^s \times \mathbb{R}^s \times \mathbb{R}^d \mapsto \mathbb{R}^d$ describe the I/O properties of a subsystem l, and l^* indicates the index of a subsystem that links to $P^{(l)}$ in graph $\mathscr{G}$. Intuitively, Eq. 3 describes a dynamic system where a sub-model interacts with its adjacent sub-models by transmitting signals.

3. The DyN mechanism

We first show how to convert a fully-connected layer of $\mathbb{R}^{a\times b}$ into $a + b$ sub-models. Then we generalize this mechanism for arbitrary neural layers and show the equivalence between *DyN* mechanism and entropy reduction.

Interpreting an FC layer as neuronal path integrals. For a feed-forward neural network, the l-th fully-connected layer with M input neurons and N output neurons is a pre-trained weight matrix $\mathbf{T} \in \mathbb{R}^{M\times N}$. Here, we present a method to represent $\mathbf{T}$ as two subsystems of d-dimensional sub-models: $P^{(l)} = \{q_i^{(l)}, i \in [1, M]\}$ and $P^{(l+1)} = \{q_j^{(l+1)}, j \in [1, N]\}$. The received signals of $P^{(l)}$ are $[R_1^{(l,t)}, ..., R_M^{(l,t)}] = R^{(l,t)} \in \mathbb{R}^{1\times M}$, and the emitted signals of $P^{(l+1)}$ are $[E_1^{(l+1,t)}, ..., E_N^{(l+1,t)}] = E^{(l+1,t)} \in \mathbb{R}^{1\times N}$. Our goal is to establish the subsystem-related mapping Ψs (see Eq. 3) such that $R^{(l,t)}\mathbf{T} \rightarrow E^{(l+1,t)}$ with $P^{(l)}$ and $P^{(l+1)}$. We set $\Psi_Q = 0$ for the inference stage with fixed sub-models. Generally, we interpret Ψs as follows (we abbreviate the input arguments as $\circ$ for convenience):

$$\begin{gathered} \Psi_R^{(l+1)}(\circ) = E_j^{(l,t)} \cdot \varphi(q_i^{(l,t)}, q_j^{(l+1,t)}) \\ \Psi_E^{(l)}(\circ) = R_i^{(l,t)};\ \ \Psi_E^{(l+1)}(\circ) = R_j^{(l+1,t)} \\ \Psi_Q^{(l)}(\circ) = 0;\ \ \Psi_Q^{(l+1)}(\circ) = 0 \end{gathered} \tag{4}$$

where the metric function φ refers to a nonlinear path integral, which is not a computation-friendly formulation. Instead, we can convert the nonlinear φ into the weighted sum of multiple linear relations and efficiently deal with the non-linearity by splitting each sub-model into H copies, *i.e.*, $q_i^{(l,t)} \rightarrow \{q_{ih}^{(l,t)}, h \in [1, H]\}$. Unless specified, we initialize H as $(d)^{-1} \cdot MN \cdot (M + N)^{-1}$. Next, we define the nonlinear relations between sub-models via the L_p-norm:

$$\varphi(q_i^{(l)}, q_j^{(l+1)}) = \sum_{h=1}^{H} \mu_h^{(l;l+1)} \left\| q_{ih}^{(l)} - q_{jh}^{(l+1)} \right\|_p \tag{5}$$

where $p \in \mathbb{N}^+$, and $\mu_h^{(l;l+1)} \in \mathbb{R}$ is a trainable shared coefficient related to $q_{*h}^{(l)}$ and $q_{*h}^{(l+1)}$. The total amount of sub-models is flexible during training, as we can merge a set of sub-models with similar dynamical behaviors. For example, $q_{ih_x}^{(l_x,t)}$ and $q_{jh_y}^{(l_y,t)}$ can be merged at time-step T if $\sum_{t=T-5}^{T} \|q_{ik_x}^{(l_x,t)} - q_{jk_y}^{(l_y,t)}\| \leq 0.1$. Once the adjacent sub-models are merged, their subsequent dynamical behaviors will be synchronous and stored as a single sub-model. In the next section, we will show how to find proper $P^{(l)}$ and $P^{(l+1)}$ with Eq. 4 and Eq. 5.

Training a fixed FC layer with *DyN* mechanism. We describe the DyN mechanism to learn $P^{(l)}$ and $P^{(l+1)}$ such that $R^{(l,t)}\mathbf{T} \rightarrow E^{(l+1,t)}$ with Ψs defined in Eq. 4 and Eq. 5. We denote $v_{ij;h} = q_{ih}^{(l)} - q_{jh}^{(l+1)}$, and the stress force $\mathscr{F}_{ij} = \mathbf{T}_{ij} - \varphi(q_i^{(l)}, q_j^{(l+1)})$ between sub-models under the target $\mathbf{T}$. A sub-model moves in the direction of its stress force:

$$\begin{aligned} \frac{\partial q_{ih}^{(l)}}{\partial t} &= -\mu_h^{(l;l+1)} \sum_{i=1}^{M} \frac{v_{ij;h}}{\|v_{ij;h}\|_p} \cdot \mathscr{F}_{ij} \\ \frac{\partial q_{jh}^{(l+1)}}{\partial t} &= \mu_h^{(l;l+1)} \sum_{j=1}^{N} \frac{v_{ij;h}}{\|v_{ij;h}\|_p} \cdot \mathscr{F}_{ij} \\ \frac{\partial \mu_h^{(l;l+1)}}{\partial t} &+ \sum_{i=1}^{M}\sum_{j=1}^{N} \|v_{ij;h}\|_p \cdot \mathscr{F}_{ij} = 0 \end{aligned} \tag{6}$$

The last line of Eq. 6 follows the energy and momentum conservation laws, the mechanism in Eq. 6 can approximate arbitrary linear transformation and is consistent with back-propagation (Appendix G and F).

Training an MLP with *DyN* from scratch. Now we proceed to train an MLP from scratch, *i.e.*, we only know the input $R^{(in)} \in \mathbb{R}^{N_{in}}$ and the target output $T^{(out)} \in \mathbb{R}^{N_{out}}$ of each training sample. Given an MLP that contains two randomly initiated weights $W^{(in;hid)} \in \mathbb{R}^{N_{in} \times N_{hid}}$ and $W^{(hid;out)} \in \mathbb{R}^{N_{hid} \times N_{out}}$, the transmitting signals along with each layer are defined as $E^{(hid)} = \sigma(R^{(in)} W^{(in;hid)})$ and $E^{(out)} = \sigma(E^{(hid)} W^{(hid;out)})$. Following Eq. 6, we can represent this MLP as three subsystems: $P^{(in)} = \{q_i^{(in)}, i \in [1, N_{in}]\}$, $P^{(hid)} = \{q_k^{(hid)}, k \in [1, N_{hid}]\}$, and $P^{(out)} = \{q_j^{(out)}, j \in [1, N_{out}]\}$. The path integral between sub-models of $P^{(x)}$ and $P^{(y)}$ respectively is denoted by $\varphi^{(xy)} \in \mathbb{R}^{N_x \times N_y}$. We denote $\Phi_j^{(xy)} = \sum_i \sigma(R_i^{(x)}) \cdot v_{ij}^{(xy)}$ as the signals received by $q_j^{(y)}$ from all the sub-models in $P^{(y)}$, where $v_{ij}^{(xy)} = q_i^{(x)} - q_j^{(y)}$. Our goal is to update the sub-models such that $T^{(out)} = \sigma(\sigma(R^{(in)}\varphi^{(in;hid)})\varphi^{(hid;out)})$. Using back-propagation to replace $\mathscr{F}_{ij}$ in Eq. 6 with the gradients regarding neural weights, we can update the sub-models as follows:

$$\begin{aligned} \frac{\partial q_{i;h}^{(in,t)}}{\partial t} &\approx \mu_h^{(in;hid)} \Phi_i^{(in)} \Phi_i^{(hid;in)} \\ \frac{\partial q_{j;h}^{(out,t)}}{\partial t} &\approx \mu_h^{(hid;out)} \Phi_j^{(out)} \Phi_j^{(h;out)} \\ \frac{\partial q_{k;h}^{(hid)}}{\partial t} &\approx \mu_h^{(in;hid)} \Phi_j^{(in;hid)} + \mu_h^{(hid;out)} \Phi_j^{(out;hid)} \end{aligned} \tag{7}$$

where $\Phi_i^{(in)} = \sigma(R_i^{(in)})$ and $\Phi_j^{(out)} = T_j^{(out)} - E_j^{(out)}$. The coefficients μ_h are updated recursively using Eq. 6, *i.e.*, we first update the sub-models, then reconstruct the nonlinear φ and obtain the gradients of $\|\Phi^{(out)}\|_2$ to φ. The resulting gradients are the stress $\mathscr{F}$ updating μ_h as in Eq. 6.

Converting general tensor data flow into signals. The previously introduced cases focus on constructing *DyN* mechanism based on a fixed neural structure, *i.e.*, the prior knowledge that tells a sub-model exactly it should process which signals have been provided. However, to build a *DyN* system from scratch without any reference neural structure, a sub-model should be able to distinguish the signals to be processed from all the received signals. Therefore, we need more signal components to store the positional features. For example, as presented in Fig. 3, a sub-model q_{11} in 2-dimensional neural state space emits signal E_{11} that contains the unique ID of q_{11} by appending a positional encoding $[0, 0]^\top$ on its original signal $[\mathbf{X}_{11}]$, then the other sub-models can know that E_{11} is emitted by q_{11} based on the positional features.

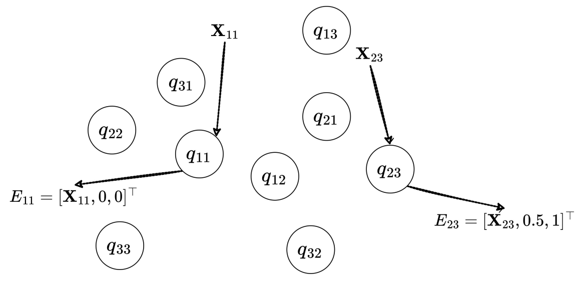


Figure 3: **Converting a tensor into signals emitted by sub-models.** Suppose we have a tensor input $[\mathbf{X}_{ij}] \in \mathbb{R}^{3\times3}$, then there could be 9 sub-models receiving the tensor input. Each sub-model q_{ij} receives signals $\mathbf{X}_{ij}$ and emits signals E_{ij} that contain the positional features.

Generally, for a tensor of rank r, *i.e.*, $\mathbf{X} \in \mathbb{R}^{d_1 \times \ldots \times d_r}$, we use at most $\prod_{k=1}^{r} d_k$ sub-models to receive the signals referring to $\mathbf{X}$. A sub-model receives a signal that refers to an entry of $\mathbf{X}$ and emits a signal of $\mathbb{R}^{1+r}$ that contains the r-dimensional positional feature. There are many techniques to reduce the number of sub-models necessary to represent the tensor input. Let's denote V_x as a set of reduced $\mathbf{X}$ by specifying the index of its x-th dimension, *e.g.*, for $\mathbf{X} \in \mathbb{R}^{3\times3\times2}$, we have $V_2[y] = \mathbf{X}[:, y, :]$. If we observe that the components of V_x are similar (using a method like KL-divergence), then we can reduce the sub-models along with the x-th dimension, *i.e.*, we need only $\prod_{k\neq x}^{r} d_k$ sub-models.

For a neural structure with hidden layers, we might need a larger shape of signals, *e.g.*, $\mathbb{R}^{r+2}$, to store the layer-wise feature that shows the signals come from which hidden layers. The formulation of positional features is not unique, and they are generally concatenated with the original signal vectors. Based on the trained positional features, the subsystem-related function Ψs can tell the sub-model to receive signals from which sub-models. The target label $T^{(out)} \in \mathbb{R}^{N_{out}}$ corresponds to N_{out} sub-models, each emitting signal $E_i^{(out)} \in \mathbb{R}^s$. Then we obtain the normalized feedback signal $\Phi_i^{(out)} = T_i^{(out)} - \|E_i^{(out)}\|_p$, where s and p are hyper-parameters introduced in previous sections.

Training arbitrary layer with *DyN*. The Eq. 7 can be seen as dynamical back-propagation, which implicitly updates neural connections while minimizing the stress force between neuronal states, see Appendix F. To train arbitrary layer, *e.g.*, convolution or self-attention, with *DyN* mechanism, we first obtain the dummy states by computing the path integrals amongst neurons, recovering the weight-based layers according to Table 1. Next, we compute the dummy stress force (gradient descent) and apply Eq. 6 to reduce the stress force. See Alg. 1 for the whole procedure.

Algorithm 1 *DyN* learning for general neural structure

```
Input: Neuronal dynamics $Q$, Desired Output $T$
repeat
    Dummy States $A = Rel(Q, Q)$ via Eq. 5
    Stress force $F = Grad(A, T)$ via Eq. 7
    Update $Q = Reduce(F, Q)$ via Eq. 6
until $Q$ reaches equilibrium
```

There are some practical tips to reduce computational complexity and memory. For example, one can cluster dynamically similar sub-models. The dynamical states of sub-models can be encoded as a sparse matrix via methods like vector quantization (see Eq. 12). The lossy dynamical states with an exact resolution $1/\delta$ can be implemented via a Cantor expansion $\mathbb{R}^d \mapsto \mathbb{N} \in [1, \delta^{-d}]$, where δ^{-d} should not exceed the maximal allowable integer allowed in the current computing system, *e.g.*, 1.8×10^{108} for any floating-point number represented as a 64-bit double-precision value.

Inference stage for a DyN system. The inference stage involves the specific matrix-vector query designed for distance matrices via faster linear algebra (Indyk & Silwal, 2022). Now we present how to feedforward an MLP on a DyN system with L_1-norm. Given an MLP that contains weights $W^{(in;hid)} \in \mathbb{R}^{N_{in} \times N_{hid}}$ and $W^{(hid;out)} \in \mathbb{R}^{N_{hid} \times N_{out}}$. Its DyN alternative has three subsystems: $P^{(in)} = \{q_i^{(in)}, i \in [1, N_{in}]\}$, $P^{(hid)} = \{q_k^{(hid)}, k \in [1, N_{hid}]\}$ and $P^{(out)} = \{q_j^{(out)}, j \in [1, N_{out}]\}$. The input signals $R^{(in)} \in \mathbb{R}^{N_{in}}$ are received by $P^{(in)}$, which emits $E^{(in)} = R^{(in)}$ to $P^{(hid)}$. Then $P^{(hid)}$ receives $R^{(hid)} \in \mathbb{R}^{N_{hid}}$ from $P^{(in)}$ and emits $E^{(hid)} = \sigma(R^{(hid)})$ to $P^{(out)}$. Finally, $P^{(out)}$ receives $R^{(out)} \in \mathbb{R}^{N_{out}}$ from $P^{(hid)}$ and emits $E^{(out)} = R^{(out)}$. Specifically, the inference stage that converts $E^{(in)}$ to $R^{(hid)}$ is as follows:

$$\begin{aligned} R^{(hid)}[x] &= \sum_{i=1}^{N_{in}} E_i^{(in)} \cdot \varphi(q_i^{(in)}, q_x^{(hid)}) \\ &= \sum_{h=1}^{H} \mu_h \cdot \sum_{y=1}^{d} \sum_{i=1}^{N_{in}} E_i^{(in)} \cdot |q_{ih}^{(in)}[y] - q_{xh}^{(hid)}[y]| \end{aligned} \tag{8}$$

Let's denote π_y^+ as a set of i such that $q_{ih}^{(in)}[y] \geq q_{xh}^{(hid)}[y]$, and likewise, denote π_y^-. We now focus on the inner loop and rearrange it as follows:

$$\begin{aligned} & E_i^{(in)} \Big(\sum_{i \in \pi_y^+} (q_{ih}^{(in)}[y] - q_{xh}^{(hid)}[y]) + \\ & \sum_{i \in \pi_y^-} (q_{xh}^{(hid)}[y] - q_{ih}^{(in)}[y]) \Big) \\ &= q_x[y] \cdot (Z_{neg} - Z_{pos}) + \Delta_{pos} - \Delta_{neg} \end{aligned} \tag{9}$$

where $Z_{neg} = \sum_{\pi_y^-} E_i^{(in)}$, $Z_{pos} = \sum_{\pi_y^+} E_i^{(in)}$, $\Delta_{pos} = \sum_{\pi_y^+} E_i^{(in)} q_{ih}^{(in)}[y]$, and $\Delta_{neg} = \sum_{\pi_y^-} E_i^{(in)} q_{ih}^{(in)}[y]$ are preprocessed values. This method reduces the computational complexity of matrix-vector query from $O(N_{in}N_{hid})$ to $O(Hd \cdot max(N_{in}, N_{hid}))$. Similarly, we can compute the inference stage of any tensor-based layer containing n neurons on a DyN system with L_p-norm using Eq. 8 and Eq. 9. This mechanism roughly reduces these tensor-based inference stages from $O(n^2)$ to $O(Hndp)$, where H is the number of copies as presented in Eq. 5.

Understanding *DyN* mechanism as entropy-reduction. Updating the dynamical states of the sub-models to approximate arbitrary mapping in Eq. 7 is equivalent to reducing the structural entropy among the sub-models. Based on the Euler-Lagrange equation, we can conclude that (see Appendix G for details):

$$\frac{\partial q_i^{(t)}}{\partial t} = \eta_i \cdot \exp\left(- \frac{\sum_{k \neq i} \frac{\partial}{\partial t} \mathcal{L}_{ik}^{(t)}}{\sum_{k \neq i} \mathcal{L}_{ik}^{(t)}} \cdot t \right) \tag{10}$$

where $\eta_i \in \mathbb{R}$ is a trainable parameter related to $q_i^{(t)}$, and $\mathcal{L}_{ik}^{(t)}$ is a Lagrangian that measures the energy flow for $q_k^{(t)}$. The form of $\mathcal{L}_{ik}^{(t)}$ is not unique. For example, it can be:

$$\mathcal{L}_{ik}^{(t)} = \frac{1}{2} m_i \cdot \frac{\partial^2 q_i^{(t)}}{\partial t^2} - U_i \cdot S_{ki}^{(t)}(q_i^{(t)}) \tag{11}$$

where $m_i \in \mathbb{R}$ and $U_i \in \mathbb{R}^{d \times s}$ are trainable parameters, referring to the mass and potential energy of a sub-model i. Intuitively, a sub-model tends to move toward the region with a lower structural entropy of the energy distribution, and the dynamical signals can be regarded as packets of energy (visualized in Figure 6 of Appendix H). This result verifies the potential prospect that a well-formed *DyN* system can be updated via global dynamics.

Issues with Activation functions. The input-to-output function of a sub-model is non-linear even without an activation function, because the path integral of DyN has already induced non-linearity. Recall that when a signal is transmitted from a sub-model to another, it will be multiplied by the path integral between these two sub-models, and the path integral

Table 2: **Evaluating weight-based neural models and their DyN forms on MNIST.** We evaluate each model's original form on MNIST and convert its layers into *DyN* forms trained using *DyN* mechanisms. We also test the case of training *LeNet-5* with Eq. 6 only, *i.e.*, we simply convert the layers of a pre-trained *LeNet-5* into DyN forms without further training.

Model	Layer Type	No.Copies		No.Params		Test Acc. (%)	
		FC	Conv	Memory	Disk	Fixed (Eq. 6)	Unfixed (Alg. 1)
3-Layered NN	FC	-		2,290k		97.89±0.10	
	DyN	50	-	1360k	160k	-	98.32±0.03
	DyN	75	-	2170k	250k	-	**98.36±0.02**
LeNet-5	FC, Conv	-		61.8k		99.06±0.10	
	DyN	2	3	14.50k	2.03k	81.44	99.13±0.10
	DyN	2	5	16.48k	2.25k	84.95	99.15±0.07
	DyN	3	6	23.01k	2.98k	96.28	99.21±0.05
	DyN	5	8	36.04k	4.44k	98.10	99.21±0.09
	DyN	7	7	46.11k	5.56k	98.83	**99.23±0.06**

Table 3: **Evaluating weight-based neural models and their DyN forms on ImageNet and WebVision.** We convert the layers of the pretrained neural models into *DyN* forms and finetune them using Alg. 1 on the training sets. All the pre-trained weight-based neural models come from *torch.hub*.

Model Configs		No.Params (Millions)	MACs (GFLOPs)	ImageNet (%)		WebVision (%)	
Structure	Layer Type			Ideal	δ=1e^{-3}	Ideal	δ=1e^{-3}
DenseNet-161	FC, Conv	28.68	7.82	75.254	71.336	68.973	61.429
	DyN	**6.05**	3.28 (0.089)	**75.314**	**75.246**	**69.033**	**68.984**
ResNet-152	FC, Conv	60.40	11.58	77.014	75.776	69.879	59.435
	DyN	**6.51**	5.25 (3.5e-3)	**77.203**	**76.604**	**70.005**	**69.998**
ViT-S-224	FC, Conv, Attn	36.38	1.11	80.108	80.038	72.665	72.509
	DyN	**3.71**	0.45 (0.75e-3)	**80.150**	**80.122**	**72.728**	**72.716**
SwinT-S-224	FC, Conv, Attn	49.94	8.52	82.634	82.070	72.755	72.604
	DyN	10.38	3.35 (0.024)	82.646	82.604	72.802	72.740
	DyN	**6.65**	2.37 (0.018)	**82.688**	**82.660**	**72.934**	**72.842**

is computed using the weighted sum of several linearities as in Eq. 5. However, a DyN model without an explicit activation function risks non-convergence and depends heavily on a good initialization of the learnable parameters. Thus, we implement the activation function like the corresponding ANN to stabilize the implementation. When a sub-model receives signals from the other sub-models, it simply takes an activation function (Sigmoid, ReLU, or GELU) similar to its ANN counterpart on the summation of the currently received signals. Then it emits the activated summed signal to the others. The activation endows DyN with more non-linearity and larger model capacity.

4. Experiments

Datasets and compared approaches. We evaluate *DyN* on three visual classification datasets, MNIST (Deng, 2012), ImageNet (Deng et al., 2009)) and WebVision (Li et al., 2017). We first conduct experiments on MNIST to compare a 3-layered feedforward neural network trained via back-propagation with its DyN-formed alternative trained from scratch via Eq. 7. Likewise, we compare the LeNet-5 (LeCun et al., 2015) with its DyN alternative trained from scratch. Each convolutional layer is converted into *DyN* forms based on the policy presented in Table 1. We further validate our approaches on ImageNet and WebVision, by converting mainstream pre-trained neural models built on ImageNet training split from torch.hub, including DenseNet-161 (Huang et al., 2017), ResNet-152 (He et al., 2016), ViTs (Dosovitskiy et al., 2020) and Swin-Transformer (Liu et al., 2021), to their DyN forms. On all datasets, for a fair comparison, we set the model configuration of the original ANNs and their DyN alternatives (*e.g.*, the number of hidden units, validation criterion, SEED, *etc.*) to be the same. The training/testing splits of all the datasets follow the official settings. For Webvision results, we only use the testing split to report accuracy, while the finetuning is conducted on the training split of ImageNet instead of Webvision.

Evaluation metrics and implementation details. We report the top-1 accuracy, the number of parameters, and the computational complexity that measures how many operations are needed for each model during the inference phase. We repeat the training procedure for each configuration 20 times to calculate its mean and standard deviation. In addition to the typical evaluation (*e.g.*, $ideal$ columns in Table 3), we also conduct experiments under varying parameter resolution $1/\delta$, where $\delta > 0$ to measure a model's robustness during inference. We truncate a trainable parameter matrix X into its quantized form $p_\delta(X)$ as:

$$p_\delta(X_{ij}) = \min(X) + \lfloor \frac{X_{ij} - \min(X)}{\delta \cdot J} \rfloor \delta \cdot J \qquad (12)$$

where $J = \max(X) - \min(X)$. For an idealized weight-based or *DyN* model, we set $\delta = 0$ or $\delta = 1e^{-6}$. When $\delta = 0$, $p_\delta(X_{ij}) = X_{ij}$. The trainable units in a weight-based ANN are the weights $\{W_{ij}, i \in [1, a], j \in [1, b]\}$, while the ones in a DyN model are the input neuronal states $Q_i \in \mathbb{R}^{d\times a}$ and output neuronal states $Q_j \in \mathbb{R}^{d\times b}$. The computational complexities of a weight-based model and its DyN alternative are measured using Multiply-Accumulate units (MACs), *i.e.*, the number of FLOPs (Patil & Kulkarni, 2018). For a DyN model, we also compute its computational complexity in a physically meaningful way (denoted in brackets next to the MACs), which assumes that the computation of path integral amongst sub-models is executed instantaneously (a phenomenon that exists in the spatiotemporal liquid crystal structures (Zhang et al., 2021)). The dimension d defaults to 9 unless otherwise noted. Note that d can still be tuned for numerical model optimization with higher/lower values. We fine-tune the models with one NVIDIA RTX3090 24GB GPU on a cloud server. The inference stage is implemented on a laptop with 32GB memory.

4.1. Visual classification

The main results on three datasets are presented in Table 2 and Table 3. Compared against feedforward neural networks and LeNet-5, our randomly initialized DyN models trained from scratch via Alg. 1 demonstrate higher accuracy, lower computational complexity, and reduced parameter size. Then we use several pre-trained models as backbone networks and convert their FC, convolution, and attention layers into DyN forms. The final dynamical states of the sub-models are determined by fine-tuning the transformed neural models on ImageNet's training set. This process continues until the stress force amongst sub-models is lower than a certain threshold, *e.g.*, 10^{-3} of the normalized distances between sub-models. We observe that the DyN alternative of each neural model achieves significant improvement in accuracy, especially under lower parameter resolution, *i.e.*, a higher δ value, see Figure 5 and Appendix I. Besides, a neural model with more neural blocks transformed via DyN mechanism performs better than the one with less DyN blocks, *e.g.*, swinDyN in Table 3 and Figure 8a. These results show that DyN mechanism preserves more information efficiently by encouraging all-dynamic neuron interaction.

4.2. Relation to the asymmetric convolutions

Given a convolutional layer $M_C \in \mathbb{R}^{k\times k\times N_{in}\times N_{out}}$, we have two policies to convert M_C into its DyN form. The first is the k^2 policy: converting M_C into k^2 subsystems $\{P^{(11)}(n), P^{(12)}..., P^{(kk)}(n)\}$, where $n = N_{in} + N_{out}$, such that $\varphi(q_k^{(ij)}, q_l^{(ij)}) = M_C[i, j, k, l]$. The second is the $2k$ policy (Table 1): converting M_C into $2k$ subsystems $\{P^{(in,1)}(n), ..., P^{(in,k)}(n), P^{(out,1)}(n), ..., P^{(out,k)}(n)\}$, making each subsystem correspond to an input or output channel, such that $\varphi(q_k^{(in,i)}, q_l^{(out,j)}) = M_C[i, j, k, l]$. We notice that the $2k$ policy significantly reduces the parameters while preserving the accuracy in Table 4 compared with the k^2 policy. The experiments use the $2k$ policy to convert a convolutional layer.

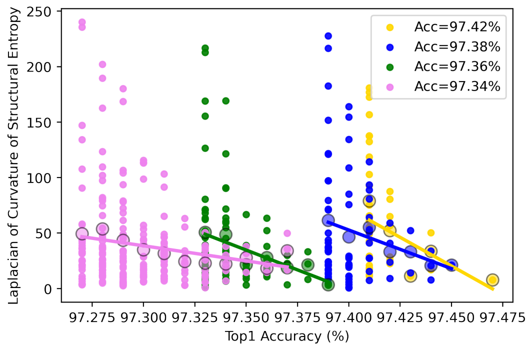
Table 4: **Comparison of k^2 and $2k$ policies in convolution.**

KERNEL	NO.COPIES		NO.PARAMS	TEST ACC (%)
	FC	CONV		
k^2	5	5	6.27 K	99.15±0.07
$2k$	5	5	3.97 K	99.16±0.06
k^2	5	8	7.99 K	99.19±0.06
$2k$	5	8	4.32 K	99.20±0.08
k^2	7	7	8.65 K	99.24±0.04
$2k$	7	7	5.43 K	99.24±0.03
k^2	8	5	8.11 K	99.22±0.05
$2k$	8	5	5.81 K	99.23±0.04

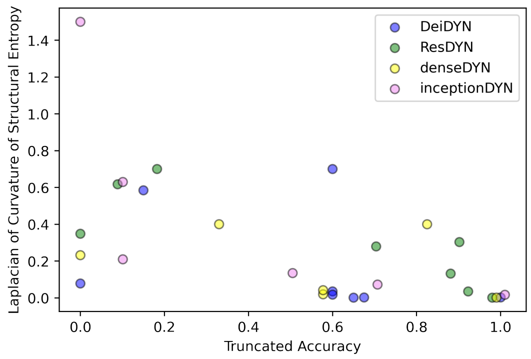
This idea is similar to the asymmetric convolution, which converts $k \times k$ convolution into stacked $1 \times k$ and $k \times 1$ ones (Szegedy et al., 2016). It implies that the parameter-sharing mechanism also works in DyN and explores the neuronal covariants that induce a way to boost performance.

4.3. Correlation with structural entropy

We examine DyN under different δ via Eq. 12. Though larger δ leads to lower parameter precision, we observe an existence of peak that corresponds to the optimal setting of δ such that a model achieves its best testing accuracy (Figure 7b in Appendix H). We postulate that the peak corresponds to some regularization effect that prevents overfitting, which is also related to the cross-entropy and the system's structural configuration. To validate our postulate, we first evaluate the new coordinates q_i of sub-models with a varying δ by $q_i(\delta) = \delta \times \lfloor q_i/\delta \rfloor$, then we count the spatial distribution of each newly resulted sub-model in terms of



(a) 2-layered DyN models on MNIST



(b) Deep DyN models on ImageNet

Figure 4: Scattered points and their expectations that represent model performances for simple 2-layer DyN model with distinct resolution on MNIST (a), and for several mainstream neural models transformed via DyN approach on ImageNet (b).

coordinates $Pr(v_x, \delta) = |\{q_i|q_i(\delta) = v_x\}|/|\{q_i\}|$, where $v_x \in \mathbb{R}^d$. Moreover, we calculate the structural entropy

$$\psi(\delta) = -\sum_{v_x} Pr(v_x, \delta) \cdot \log Pr(v_x, \delta) \tag{13}$$

to measure the structural disorder in terms of the system's energy distribution. To connect the structural entropy with its physical meaning, we evaluate the Laplacian of curvature κ of $\psi(\delta)$ (denoted by $LapCurSE$), which accounts for the energy of surface diffusion flow (Sethian & Chopp, 1999)

$$LapCurSE(\delta) = \left\|\frac{\partial^2}{\partial\delta^2}\kappa(\psi(\delta))\right\| \tag{14}$$

and we observe that an optimal structural setting always refers to a lower $LapCurSE$, whose expected value is negatively correlated with model performance (Figure 4a). This observation implies that optimal performance requires a stable structure instantiated as a minimal surface of energy distribution: $\delta_{optimal} = \arg\min_\delta LapCurSE(\delta)$, which ensures that all sub-models find the dynamical states that make them the most stable with the lowest energy. We evaluate the $LapCurSE$s of the *DyN* models of several mainstream models on ImageNet and observe that an optimal performance always refers to a lower $LapCurSE$ (Figure 4b). We also observe that the total computational cost of neurons that follow the dynamical UAT is theoretically and experimentally conservative (Appendix H).

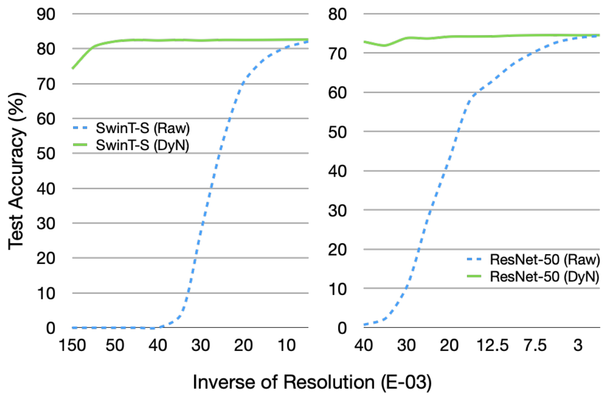


Figure 5: On ImageNet, as the noise δ^{-1} increases, the accuracy of a neural model decreases, while that of the *DyN* alternative almost retains.

5. Limitations

See Appendix J.

6. Conclusion

We propose a dynamics-inspired neuromorphic architecture that interprets neural representation and learning from dynamics theory. It emphasizes the state representation of the neurons rather than the neural weights. In visual classification, our architecture fully exploits each neuronal parameter, demonstrating superiority in accuracy, parameters, and computational complexity. More investigation on the correlation between model performance and structural entropy reveals that learning via structural mechanism is better than numerical mechanism in efficiency and explainability. Future work includes applying DyN on multimodal data, new tasks (*e.g.*, retrieval, and QA), and providing an in-depth physical interpretation of the neuronal state space.

Acknowledgements

This work was supported in part by the National Key R&D Program of China under Grant 2018AAA0102000, in part by National Natural Science Foundation of China: 62022083 and 62236008. We thank anonymous reviewers for helpful suggestions that improved this work.

References

Abarbanel, H. D. and Rouhi, A. Phase space density representation of inviscid fluid dynamics. *The Physics of fluids*, 30(10):2952–2964, 1987.

Akrout, M., Wilson, C., Humphreys, P., Lillicrap, T., and Tweed, D. B. Deep learning without weight transport. *Advances in neural information processing systems*, 32, 2019.

Assunção, F., Lourenço, N., Machado, P., and Ribeiro, B. Evolving the topology of large scale deep neural networks. In *European Conference on Genetic Programming*, pp. 19–34. Springer, 2018.

Basegmez, E. The next generation neural networks: Deep learning and spiking neural networks. In *Advanced Seminar in Technical University of Munich*, pp. 1–40. Citeseer, 2014.

Braverman, M., Schneider, J., and Rojas, C. Space-bounded church-turing thesis and computational tractability of closed systems. *Physical review letters*, 115(9):098701, 2015.

Chklovskii, D. B., Mel, B., and Svoboda, K. Cortical rewiring and information storage. *Nature*, 431(7010): 782–788, 2004.

Cho, R. W., Buhl, L. K., Volfson, D., Tran, A., Li, F., Akbergenova, Y., and Littleton, J. T. Phosphorylation of complexin by pka regulates activity-dependent spontaneous neurotransmitter release and structural synaptic plasticity. *Neuron*, 88(4):749–761, 2015.

Cooper, S. J. Donald o. hebb's synapse and learning rule: a history and commentary. *Neuroscience & Biobehavioral Reviews*, 28(8):851–874, 2005.

Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Deng, L. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

Gaier, A. and Ha, D. Weight agnostic neural networks. *Advances in neural information processing systems*, 32, 2019.

Golubeva, A., Neyshabur, B., and Gur-Ari, G. Are wider nets better given the same number of parameters? *arXiv preprint arXiv:2010.14495*, 2020.

Han, Y., Huang, G., Song, S., Yang, L., Wang, H., and Wang, Y. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hinton, G. E., Osindero, S., and Teh, Y. W. A fast learning algorithm for deep belief nets. *Neural Computation*, 18: 1527–1554, 2006.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Huynh, P. K., Varshika, M. L., Paul, A., Isik, M., Balaji, A., and Das, A. Implementing spiking neural networks on neuromorphic architectures: A review. *arXiv preprint arXiv:2202.08897*, 2022.

Indyk, P. and Silwal, S. Faster linear algebra for distance matrices. *arXiv preprint arXiv:2210.15114*, 2022.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, may 2017. ISSN 0001-0782.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

LeCun, Y. et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20(5):14, 2015.

Li, W., Wang, L., Li, W., Agustsson, E., and Van Gool, L. Webvision database: Visual learning and understanding from web data. *arXiv preprint arXiv:1708.02862*, 2017.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10012–10022, 2021.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pp. 1310–1318. PMLR, 2013.

Patil, P. A. and Kulkarni, C. A survey on multiply accumulate unit. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pp. 1–5. IEEE, 2018.

Ren, P., Xiao, Y., Chang, X., Huang, P.-Y., Li, Z., Chen, X., and Wang, X. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)*, 54(4):1–34, 2021.

RoyChowdhury, A., Sharma, P., and Learned-Miller, E. G. Reducing duplicate filters in deep neural networks. 2018.

Scarselli, F. and Tsoi, A. C. Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural networks*, 11(1): 15–37, 1998.

Sethian, J. A. and Chopp, D. Motion by intrinsic laplacian of curvature. *Interfaces and Free boundaries*, 1(1):107–123, 1999.

Stanley, K. O. and Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

Stork, J., Zaefferer, M., and Bartz-Beielstein, T. Improving neuroevolution efficiency by surrogate model-based optimization with phenotypic distance kernels. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pp. 504–519. Springer, 2019.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Weinberg, S. *The quantum theory of fields*, volume 2. Cambridge university press, 1995.

Zhang, R., Redford, S. A., Ruijgrok, P. V., Kumar, N., Mozaffari, A., Zemsky, S., Dinner, A. R., Vitelli, V., Bryant, Z., Gardel, M. L., et al. Spatiotemporal control of liquid crystal structure and dynamics through activity patterning. *Nature materials*, 20(6):875–882, 2021.

A. Symbol Glossary

The label on the right indicates the first time the symbol or notation is defined or used. As usual, $\mathbb{R}$, $\mathbb{Z}$, $\mathbb{C}$, and $\mathbb{N}$ denotes the reals, the integers, the complex numbers and the natural numbers, respectively.

$q_i^{(l,t)} \in \mathbb{R}^d$	The dynamical states of a sub-model i of subsystem l at time-step t	Sec 2
$E_i^{(l,t)} : \mathbb{R}^d \mapsto \mathbb{R}^s$	Vector field for the signals emitted by $q_i^{(l,t)}$	Figure 2
$R_i^{(l,t)} : \mathbb{R}^d \mapsto \mathbb{R}^s$	Vector field for the signals received by $q_i^{(l,t)}$	Figure 2
$R_i^{(l,t)}$, $E_i^{(l,t)}$	The simplified notations for $R_i^{(l,t)}(\mathbf{0})$ and $E_i^{(l,t)}(\mathbf{0})$	Figure 2
N_l	The number of sub-models in the subsystem $P^{(l)}$	Sec 3
$R^{(l,t)}$, $E^{(l,t)}$	The simplified notations for $[R_1^{(l,t)}, ..., R_{N_l}^{(l,t)}]$ and $[E_1^{(l,t)}, ..., E_{N_l}^{(l,t)}]$	Sec 3
$S_{ij}^{(l,t,t^*)}(v) \in \mathbb{R}^s$	The temporal signal at $v \in \mathbb{R}^d$ from $q_i^{(l,t)}$ to $q_j^{(l,t^*)}$	Figure 2
$P(x)$	Subsystem containing x sub-models	Table 1
$\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{F}$	The matrices used to present the principle of dynamic subsystems	Eq. 2
$\varphi : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$	A metric function that helps to compute the path integral between sub-models	Eq. 2
$P^{(l)}$	Subsystem with an index of l	Sec 2
$\mathscr{G}$	A directed graph that describes the topological relations amongst the subsystems	Sec 2
Ψ_R, Ψ_E, Ψ_Q	The subsystem-related mappings that describe the dynamic properties of a subsystem l	Eq. 3
l^*	The index of a subsystem such that there is a directed edge from $P^{(l^*)}$ to $P^{(l)}$ in graph $\mathscr{G}$	Eq. 3
$\mathscr{F}_{ij}$	The stress force between sub-models q_i and q_j	Eq. 6
v_{ij}	Relative position between the current $q_i^{(l,t)}$ and its adjacent $q_j^{(l+1,t)}$	Eq. 6
$v_{ij}^{(xy)}$	Relative position between arbitrary $q_i^{(x)}$ and another arbitrary $q_j^{(y)}$	Eq. 7
$\varphi^{(xy)}$	Path integrals between each sub-model in $P^{(x)}$ and $P^{(y)}$	Eq. 7
$\Phi_j^{(xy)}$	The signals received by $q_j^{(y)}$ from all the sub-models in $P^{(y)}$	Eq. 7
$\mu_h^{(x;y)}$	The shared coefficient related to the sub-models $q_{*h}^{(x)}$ and $q_{*h}^{(y)}$	Eq. 5
σ	An activation function, *e.g.*, Sigmoid function	Eq. 1
δ	The inverse resolution; a lower δ means a better parameter precision	Eq. 12
p_δ	A function that truncates the model's parameters in terms of δ	Eq. 12
ε	An error threshold	Eq. 1
ϵ	A variational unit that approaches to zero	Eq. 2
ψ	The structural entropy	Eq. 4
H	The number of copies	Eq. 5
$\mathcal{L}$	The Lagrangian function	Eq. 56
$\mathscr{L}$	The loss function	Eq. 34
$T^{(out)}$	The target output for a neural model	Eq. 7
$\mathbf{T}$	A target matrix that a DyN system approximates	Eq. 4

B. Terminologies Comparison between *DNN* and *DyN*

Table 5: **Intuitional Comparisons of *DNN* and *DyN*.**

Framework	DNN	DyN
Basic components	Artificial neurons	Sub-models
Structured components	Neural layer	Subsystem
Component interaction	Connection between neurons	Path integral between neuronal dynamics
Model update manner	Adjust neural weights	Adjust neuronal dynamics
Objective function	Classification loss function based on gradient descent	Entropy reduction based on stress force
Data flow	Layered representation	Signals
System propagation manner	Layer-by-layer	Time-by-time
End of training criteria	Convergence	Neuronal states reach equilibrium

C. Some Principles and Proofs

Theorem C.1. Principle of dynamic subsystems *(the existence of global neuronal rules for the dynamic system as a universal approximator): For every $d, s, M, N \in \mathbb{N}$, given a system of sub-models with a set of time-variant coordinates $\{q_i^{(t)} \in \mathbb{R}^d, i \in [1, N]\}$ that receive and emit time-variant signals $R_i^{(t)} \in \mathbb{R}^s$ and $E_i^{(t)} \in \mathbb{R}^s$, then for arbitrary sequential mapping $\mathbb{R}^{s\times M} \mapsto \mathbb{R}^{s\times M}$ that defines the nonlinear relations between $R_i^{(t)}$ and $E_i^{(t)}$ for any $i \in [1, N]$, there exists a set of matrices $\mathcal{A} \in \mathbb{R}^{s\times s}$, $\mathcal{B} \in \mathbb{R}^{s\times d}$, $\mathcal{C} \in \mathbb{R}^{s\times d}$, $\mathcal{D} \in \mathbb{R}^{d\times s}$, $\mathcal{E} \in \mathbb{R}^{d\times s}$, and $\mathcal{F} \in \mathbb{R}^{d\times d}$, such that $t \in [1, M]$:*

$$\begin{aligned} R_i^{(t)} &= \sum_{j\neq i}^{N} E_j^{(t-\epsilon)} \cdot \varphi(q_j^{(t-\epsilon)}, q_i^{(t)}) \\ E_i^{(t)} &= \mathcal{A}R_i^{(t)} + \mathcal{B}q_i^{(t)} + \mathcal{C}\frac{d}{dt}q_i^{(t)} \\ \frac{d}{dt}q_i^{(t)} &= \mathcal{D}R_i^{(t)} + \mathcal{E}E_i^{(t)} + \mathcal{F}q_i^{(t)} \end{aligned} \tag{15}$$

where the non-polynomial 2-form $\varphi : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ is used to compute the path integral between sub-models.

Proof. According to Lemma C.2 and Lemma C.3, an arbitrary linear transformation requires finite distinct subsystems, and an arbitrary nonlinear transformation $\mathcal{T}_S(R_i^{(t)}) = E_i^{(t+\epsilon)}$ can be achieved by a universal rule of dynamics $\hat{\Psi}^{(S)}$,

$$\mathcal{T}_S(R_i^{(t)}), q_i^{(t+\epsilon)} = \hat{\Psi}^{(S)}(R_i^{(t)}, q_i^{(t)}) \tag{16}$$

where $E_i^{(t)}$ is the signal emitted from the sub-model $q_i^{(t)}$ itself, and $R_i^{(t)}$ is the resultant signals received by $q_i^{(t)}$ from all the other sub-models. Eq. 16 can be regarded as the static form of Theorem C.1. Likewise, Lemma C.3 reveals that we can also approximate an arbitrary nonlinear transformation $\mathcal{T}_Q(q_i^{(t)}) = q_i^{(t+\epsilon)}$ by a universal rule of dynamics $\hat{\Psi}^{(Q)}$,

$$E_i^{(t+\epsilon)}, \mathcal{T}_Q(q_i^{(t)}) = \hat{\Psi}^{(Q)}(R_i^{(t)}, \mathcal{T}_{LQ}(q_i^{(t)}, \frac{\partial}{\partial t}q_i^{(t)})) \tag{17}$$

where $\mathcal{T}_{LQ}$ is a linear transformation to interpret the continuous transformation of the dynamical states. Thus, Eq. 17 can be reinterpreted via a rule of dynamics in the linear form $\hat{\Psi}^{(LQ)}$,

$$E_i^{(t+\epsilon)}, \mathcal{T}_Q(q_i^{(t)}) = \hat{\Psi}^{(LQ)}(R_i^{(t)}, q_i^{(t)}, \frac{\partial}{\partial t}q_i^{(t)}) \tag{18}$$

Then the dynamical form Eq. 15 can be proved by induction. Specifically, given a set of signals $E^{(t)} = \{E_i^{(t)}\}$ and $R^{(t)} = \{R_i^{(t)}\}$ accompanied by $Q^{(t)} = \{q_i^{(t)}\}$, we first reach arbitrary expected states of signals $\{R^{(t)}, E^{(t+\epsilon)}\}$ via Eq. 16, then we reach arbitrary expected states of sub-models $Q^{(t+\epsilon)}$ via Eq. 18, followed by a successive reaching for an arbitrary targeted signals $\{R^{(t+\epsilon)}, E^{(t+2\epsilon)}\}$. This recursive step finally constructs a complete dynamic form of Lemma C.3. This fact implies that the sub-models with specific dynamical states can approximate arbitrary instant nonlinear transformation. A specific predecessor state can obtain such specified dynamical states under the constraint that the received signals are provided.

□

Lemma C.2. *The weighted sum of H distinct distance matrices is sufficient to approximate any matrix $\mathbf{T} \in \mathbb{R}^{m\times n}$ in any degree of precision. Specifically, the upper bound of an optimized H is given by*

$$H^{optimized} \leq \lceil \frac{mn}{d\cdot(m+n)} \rceil \tag{19}$$

where d refers to the dimension of units whose L_2-norms are computed to generate those distance matrices.

Proof. First, we count the number of distinguished values an arbitrary quantized matrix can have. The matrix elements $\mathbf{T}_{ij}$ range from 0 to 1 with a resolution $1/\delta$ that divides the domain into $1/\delta$ partitions. Then the total number of distinguished values is $\Phi(T) = \delta^{-mn}$. Likewise, the permutations of m units $Q^{row} \in \mathbb{R}^{m\times d}$ and n units $Q^{col} \in \mathbb{R}^{n\times d}$ are $\Omega(Q^{row}) = \delta^{-md}$ and $\Omega(Q^{col}) = \delta^{-nd}$, implying that the number of combinations of Q^{row} and Q^{col} is $\Omega([Q^{row}; Q^{col}]) = \Omega(Q) = \delta^{-d(m+n)}$. Then we need to eliminate the duplicate states of Q, which can be categorized into three cases, *i.e.*, self-permutation (SP), transitional invariance (TI), and rotational invariance (RI). Specifically, SP refers to the case that when the identities of sub-models are exchanged, the resulting distance matrix remains unchanged. TI and RI mean that when the sub-models of Q^{row} and Q^{col} move together in a translational or rotational way, the resulting distance remains unchanged. The numbers of the three cases are approximated as follows:

$$\begin{aligned} \Omega_{SP}(Q) &= (m!\cdot n!)^d \\ \Omega_{TI}(Q) &= \delta^{-1}\cdot\prod_{k=1}^{d}(1-L_k^{row}L_k^{col}) \\ \Omega_{RI} &= Sf^{(d)}(\max(L_k^{row}L_k^{col})\cdot\delta^{-1}) \end{aligned} \tag{20}$$

where $L_k^{row} = \max(Q^{row}[:,k]) - \min(Q^{row}[:,k])$ measures the maximum range of Q^{row} in the k-th dimension, and $Sf^{(d)}$ is the spherical area of d-sphere. Therefore, each pair of Q^{row} and Q^{col} covers $\hat{\Omega}(Q)$ non-duplicate states:

$$\hat{\Omega}(Q) = \frac{\Omega(Q)}{\Omega_{SP}(Q)\cdot\Omega_{TI}(Q)\cdot\Omega_{RI}(Q)} \tag{21}$$

Then the total number of non-duplicate states achieved by H subsystems is

$$\Omega([Q^{(1)}; Q^{(2)}...Q^{(H)}]) = \Omega_H(Q) = \frac{(\hat{\Omega}(Q))^H}{(H!)^d} \tag{22}$$

Our goal is to find a proper H such that $\Omega_H(Q) = \Omega(T)$, yielding

$$\lim_{\delta\to 0} H = \lceil \frac{mn}{d\cdot(m+n)} \rceil \tag{23}$$

which is consistent with Eq. 19. □

Lemma C.3. Existence of universal rule for static subsystems: *given a DyN system composed of sub-models whose rules of dynamics determine their successive states interacted with emitted or received signals, for any continuous function that maps the receival signals to the emitted signals of all the sub-models, there exists a universal rule of dynamics followed by every sub-model, ensuring that the relation between the receival signals and the emitted signals of all the sub-models approximates the provided continuous function. The universal dynamics rule is mathematically equivalent to a set of linear transformations.*

Proof. Suppose there exists an expected universal rule Ψ, and a fake sub-model $q_*^{(t)}$ that complements the signals received by each real sub-model. Each real sub-model $q_i^{(t)}$ is equipped with an exclusive rule of dynamics Ψ_i that might be different from Ψ, we have

$$E_i^{(t+\epsilon)}, q_i^{(t+\epsilon)} = \Psi_i(R_i^{(t)}, q_i^{(t)}) = \Psi(R_i^{(t)} + S_{*i}^{(t)}, q_i^{(t)}) \tag{24}$$

Taking the total derivatives over time-step on the middle and right sides of Eq. 24

$$\frac{\partial \Psi_i}{\partial q_i^{(t)}} \cdot \frac{\partial q_i^{(t)}}{\partial t} + \frac{\partial \Psi_i}{\partial R_i^{(t)}} \cdot \frac{\partial R_i^{(t)}}{\partial t} = \frac{\partial \Psi}{\partial q_i^{(t)}} \cdot \frac{\partial q_i^{(t)}}{\partial t} + \frac{\partial \Psi}{\partial R_i^{(t)}} \cdot \frac{\partial R_i^{(t)}}{\partial t} + \frac{\partial \Psi}{\partial S_{*i}^{(t)}} \cdot \frac{\partial S_{*i}^{(t)}}{\partial t} \tag{25}$$

The signal emitted from the fake sub-model is expected to be a zero constant, therefore,

$$\left(\frac{\partial \Psi_i}{\partial q_i^{(t)}} - \frac{\partial \Psi}{\partial q_i^{(t)}}\right) \cdot \frac{\partial q_i^{(t)}}{\partial t} = \left(\frac{\partial \Psi}{\partial R_i^{(t)}} - \frac{\partial R_i^{(t)}}{\partial t}\right) \cdot \frac{\partial R_i^{(t)}}{\partial t} \tag{26}$$

since $q_i^{(t)}$ and $R_i^{(t)}$ cannot be constant variables, the Eq. 26 is satisfied for all cases if and only if

$$\frac{\partial \Psi_i}{\partial q_i^{(t)}} - \frac{\partial \Psi}{\partial q_i^{(t)}} = \frac{\partial \Psi}{\partial R_i^{(t)}} - \frac{\partial R_i^{(t)}}{\partial t} = 0 \tag{27}$$

which implies that

$$\Psi(R_i^{(t)}, q_i^{(t)}) = W_q \cdot q_i^{(t)} + W_R \cdot R_i^{(t)} + b = \Psi_i(R_i^{(t)}, q_i^{(t)}) + b^{'} \tag{28}$$

where W_q, W_R, b and $b^{'}$ are proper matrices. Hence, if each sub-model's rule of dynamics refers to its complete linear transformation ψ_i, then there exists a universal linear transformation, substituting for each specified ψ_i such that the expected nonlinear transformation of the DyN system remains unchanged based on Lemma C.4. □

Lemma C.4. *If each sub-model's dynamics rule is a linear transformation, the configured system can approximate any continuous function to the expected precision.*

Proof. According to Lemma C.2, the configured system can approximate any matrix corresponding to the tensor-based neural weights by manipulating the sub-models. Therefore, the combination of each sub-model's linear transformation and the overall weights between each pair of sub-models is equivalent to a feedforward neural network with arbitrary width as illustrated in the universal approximation theorem (Scarselli & Tsoi, 1998). □

D. Path integral formulation for neuromorphic system

Recall that the emitting signals are denoted by $E_i^{(t)} \in \mathbb{R}^d$, and the receiving signals are denoted by $R_i^{(t)}$. We also introduce the dynamical signal $S_{ij}^{(t^*,t)}(v) \in \mathbb{R}^d$ to analyze the temporal state of the signal from $q_i^{(t^*)}$ to $q_j^{(t)}$. As presented in Eq. 3, we can represent the relation between $S_{ij}^{(t^*,t)}(q_i^{(t^*)})$ and $S_{ij}^{(t^*,t)}(q_j^{(t)})$ via a nonlinear subsystem-related function $\Psi : \mathbb{R}^s \times \mathbb{R}^d \mapsto \mathbb{R}^s$ as follows:

$$S_{ij}^{(t^*,t)}(q_j^{(t)}) = \Psi\Big(S_{ij}^{(t^*,t)}(q_i^{(t^*)}), q_j^{(t)} - q_i^{(t^*)}\Big) \tag{29}$$

This equation is also mathematically equivalent to the path integral between $q_i^{(t^*)}$ and $q_j^{(t)}$ as follows:

$$S_{ij}^{(t^*,t)}(q_j^{(t)}) = \int_{q_i^{(t^*)}}^{q_j^{(t)}} \hat{\Psi}\Big(S_{ij}^{(t^*,t)}(v), v\Big) dv \tag{30}$$

where $\hat{\Psi} : \mathbb{R}^s \times \mathbb{R}^d \mapsto \mathbb{R}^s$ refers to the global field consisting of several $\Psi^{(l)}$s corresponding to distinct subsystems $P^{(l)}$s. In fact, we can conclude that the relation between $\hat{\Psi}$ and $\Psi^{(l)}$s holds for:

$$\hat{\Psi}(S, u) = \sum_l \mathcal{W}^{(l)} \Psi^{(l)}(S, u) \tag{31}$$

where $\mathcal{W}^{(l)}$s are trainable linear transformations. Under this setting, we can assume that the emitting time t^* is irrelevant because the biases induced by the time delay can be learned by the $\mathcal{W}^{(l)}$s. Specifically, the bias induced by each subsystem is polynomial to the resultant bias induced by all the subsystems. Thus, the relation between the biases can be approximated via a weighted sum of linear transformations.

E. Principle of hierarchical structures

Let's define $S_{ij}^{(k,t)}$ as the signals emitted from $q_i^{(l,t^*)}$ at an unspecified time-step t^* and received by $q_j^{(k,t)}$ at a specified time-step t. Then the directed edge from $P^{(l)}$ to $P^{(k)}$ means that there exists a linear mapping $\mathcal{T}^{(lk)} : E_i^{(l,t)} \times q_i^{(l,t)} \times q_j^{(k,t+1)} \mapsto S_{ij}^{(k,t+1)}$ and a linear mapping $\mathcal{T}^{(k)} : q_j^{(k,t)} \times S_{ij}^{(k,t)} \mapsto q_j^{(k,t+1)}$ such that

$$E_j^{(k,t+1)} = \sum_{i \neq j} \mathcal{T}^{(lk)}(E_i^{(l,t)}, q_i^{(l,t)}, \mathcal{T}^{(k)}(q_j^{(k,t)}, \mathcal{T}^{(lk)}(E_i^{(l,t-1)}, q_i^{(l,t-1)}, q_j^{(k,t)}))) \tag{32}$$

According to Lie algebra homomorphism, there exists a nonlinear mapping $\mathcal{T}^{(lk)}$ such that

$$\frac{\partial}{\partial t} E_j^{(k,t+1)}, \frac{\partial}{\partial t} q_j^{(k,t)} = \sum_{i \neq j} \mathcal{T}^{(lk)}(\frac{\partial}{\partial t} E_i^{(l,t-1)}, \frac{\partial}{\partial t} q_i^{(l,t)}) \tag{33}$$

The Eq. 33 is the principle of hierarchical structures (PoHS). Suppose a tree-based structure describes the recursive relations between the root system and its subsystems, sub-subsystems, *etc.*. Then PoHS states that this tree-based structure is equivalent to a linearly hierarchical structure containing a set of subsystems. Furthermore, Eq. 33 reveals that a well-formed neuromorphic system does not require a specified set of discrete trainable units isolated from each other.

F. Consistency with back-propagation

First, let's deduce Eq. 6 using approaches applied in back-propagation. This equation is initially derived from a dynamics-inspired view in the main paper. Specifically, we will deduce Eq. 6 by computing the gradient of the loss function for each trainable parameter by the chain rule. The loss function between two arbitrary sub-models $q_i^{(l)}$ and $q_j^{(l+1)}$ of distinct subsystems is defined as

$$\begin{aligned} \mathscr{L}_{ij} &= (\mathbf{A}_{ij} - \varphi(q_i^{(l)}, q_j^{(l+1)}))^2 \\ \mathscr{L}_i &= \sum_{j=1}^{M} \mathscr{L}_{ij} \end{aligned} \tag{34}$$

where $\mathbf{A} \in \mathbb{R}^{M \times N}$ and $[\varphi(q_i^{(l)}, q_j^{(l+1)})]_{M,N} \in \mathbb{R}^{M \times N}$ are, respectively, the target matrix and the weighted distance between sub-models (defined in Eq. 5). Then we calculate the partial derivative of the loss function $\mathscr{L}_{ij}$ for a sub-model $q_{i;h}^{(l)}$ as

$$\frac{\partial \mathscr{L}_{ij}}{\partial q_{i;h}^{(l)}} = 2\mathscr{F}_{ij} \cdot \mu_h^{(l;l+1)} \cdot \frac{v_{ij;h}}{\|v_{ij;h}\|_p} \tag{35}$$

where $v_{ij;h} = q_{i;h}^{(l)} - q_{j;h}^{(l+1)}$, and the stress force between sub-models under a target $\mathbf{A}$ is denoted by $\mathscr{F}_{ij} = \mathbf{A}_{ij} - \varphi(q_i^{(l)}, q_j^{(l+1)})$. Since the collective loss function for a sub-model $q_{i;h}^{(l)}$ is $\mathscr{L}_i$, thus,

$$\frac{\partial q_{i;h}^{(l)}}{\partial t} = -\sum_{j=1}^{M} \frac{\partial \mathscr{L}_{ij}}{\partial q_{i;h}^{(l)}} = -\mu_h^{(l;l+1)} \cdot \sum_{j=1}^{M} \frac{v_{ij;h}}{\|v_{ij;h}\|_p} \cdot \mathscr{F}_{ij} \tag{36}$$

which is consistent with Eq. 6. Similarly, the update rules for $q_{j;h}^{(l+1)}$ and $\mu_h^{(l;l+1)}$ are accessible via computing the gradient of the relevant loss function. These facts guarantee that these dynamics-inspired update rules are consistent with the rules derived via computing gradient descent for a specified loss function like back-propagation does. Therefore, we can extend Eq. 6 to a detailed formulation by applying back-propagation on the stress force $\mathscr{F}_{ij}$, which is replaced with the gradient of loss function for the sub-models rather than the neural weights.

Given a multilayer perceptron that contains two tensor-formed weights $W^{(ih)} \in \mathbb{R}^{N_{in} \times N_h}$ and $W^{(ho)} \in \mathbb{R}^{N_h \times N_{out}}$. The transmitting signals along with each layer are defined using a sequence: $[R^{(in)}, E^{(in)} = R^{(in)}, R^{(h)}, E^{(h)} = \sigma(R^{(h)}), R^{(out)}, E^{(out)} = R^{(out)}]$. Our goal is to make $E^{(out)} \to T^{(out)}$. First, we define a computation-friendly formulation of the path integral between two arbitrary sub-models $q_i^{(x)}$ and $q_j^{(y)}$ as follows.

$$I_{ij}^{(xy)} = \frac{1}{2}(q_i^{(x)} - q_j^{(y)})^2 = \frac{1}{2} v_{ij}^{(xy)2} = \frac{1}{2} \sum_{h=1}^{H} \mu_h \cdot v_{ij;h}^{(xy)2} \tag{37}$$

where H is the number of shared coefficients required to convert non-linearity into a good linearity set. The number of H is discussed in Lemma C.2. Recall that the signals along with each layer are computed as follows

$$\begin{aligned} R_j^{(h)} &= \sum_i \sigma(R_i^{(in)}) \cdot I_{ij}^{(in;h)} \\ R_k^{(out)} &= \sum_j \sigma(R_j^{(h)}) \cdot I_{jk}^{(h;out)} \end{aligned} \tag{38}$$

The loss function is defined by

$$\mathscr{L} = \sum_k \frac{1}{2}(R_k^{(out)} - T_k^{(out)})^2 = \sum_k \frac{1}{2}\varepsilon_k^2 \tag{39}$$

The resultant signals received by different sub-models are defined by

$$\begin{aligned} \Phi_j^{(xy)} = \sum_i E_{ij}^{(xy)} &= \sum_i \sigma(S_i^{(x)}) \cdot v_{ij}^{(xy)} \\ \Phi_i^{(in)} &= \sigma(S_i^{(in)}) \\ \Phi_k^{(out)} &= T_k^{(out)} - S_k^{(out)} \end{aligned} \tag{40}$$

Instead of computing the gradient of the loss function for the weights (path integral I_{ij}), we compute the gradients for the dynamical states of a sub-model as $q_k^{(out)}$

$$\begin{aligned} \frac{\partial q_k^{(out)}}{\partial t} \propto \frac{\partial \mathscr{L}}{\partial q_k^{(out)}} &= \sum_j \frac{\partial \mathscr{L}}{\partial I_{jk}^{(h;out)}} \frac{\partial I_{jk}^{(h;out)}}{\partial q_k^{(out)}} \\ &= \varepsilon_k \cdot \sum_j \sigma(S_j^{(h)}) \cdot v_{jk}^{(h;out)} \\ &= \Phi_k^{(out)} \cdot \Phi_k^{(h;out)} \end{aligned} \tag{41}$$

A more detailed update rule regarding μ_h is as follows:

$$\frac{\partial q_{k;h}^{(out)}}{\partial t} \propto \frac{\partial \mathscr{L}}{\partial q_{k;h}^{(out)}} = \Phi_k^{(out)} \cdot \Phi_k^{(h;out)} \cdot \mu_h \tag{42}$$

Similarly, we compute the gradient of the loss function for $q_i^{(in)}$,

$$\begin{aligned} \frac{\partial q_i^{(in)}}{\partial t} = \frac{\partial \mathscr{L}}{\partial q_i^{(in)}} &= \sum_j \frac{\partial \mathscr{L}}{\partial I_{ij}^{(in;h)}} \frac{\partial I_{ij}^{(in;h)}}{\partial q_i^{(in)}} \\ &= \sum_j \frac{\partial \mathscr{L}}{\partial S_k^{(out)}} \frac{\partial S_k^{(out)}}{\partial S_j^{(h)}} \frac{\partial S_j^{(h)}}{\partial I_{ij}^{(in;h)}} \frac{\partial I_{ij}^{(in;h)}}{\partial q_i^{(in)}} \\ &= \sum_j \sum_k \varepsilon_k \cdot I_{jk}^{(h;out)} \cdot \frac{\partial \sigma(S_j^{(h)})}{\partial S_j^{(h)}} \cdot \sigma(S_i^{(in)}) \cdot v_{ij}^{(in;h)} \\ &= \sigma(S_i^{(in)}) \cdot \sum_j v_{ij}^{(in;h)} \cdot \frac{\partial \sigma(S_j^{(h)})}{\partial S_j^{(h)}} \cdot \sum_k \varepsilon_k \cdot I_{jk}^{(h;out)} \\ &= \Phi_i^{(in)} \cdot \sum_j v_{ji}^{(h;in)} \cdot \sigma(S_j^{(h)}) \cdot (1 - \sigma(S_j^{(h)})) \cdot \Phi_j^{(out;h)} \end{aligned} \tag{43}$$

In the equilibrium state (meaning that the feedback signal $\Phi_j^{(ou;h)}$ is extremely weak and stable), term $(1-\sigma(S_j^{(h)}))\cdot\Phi_j^{(out;h)}$ is degenerated to a specific constant independent of index j, so that Eq. 43 can be approximated as

$$\begin{aligned} \frac{\partial q_i^{(in)}}{\partial t} &\propto \Phi_i^{(in)} \cdot \sum_j v_{ji}^{(h;in)} \cdot \sigma(S_j^{(h)}) \\ &= \Phi_i^{(in)} \cdot \Phi_i^{(h;in)} \end{aligned} \tag{44}$$

Eq. 44 is obviously consistent with Eq. 7. Now we have the dynamical forms of update rules for sub-models toward a specific loss function. In other words, we can approximate arbitrary nonlinear functions via training the sub-models rather than the neural weights connecting them.

G. Generalized rules of dynamics in DyN systems

The neuromorphic dynamics are derived from Hamilton's principle and the Euler-Lagrange equation:

$$\frac{d}{dt}\frac{\partial \mathcal{L}_i^{(l,t)}}{\partial \dot{q}_i^{(l,t)}} - \frac{\partial \mathcal{L}_i^{(l,t)}}{\partial q_i^{(l,t)}} = 0 \tag{45}$$

where the Lagrangian $\mathcal{L}_i^{(l,t)} = S_i^{(l,t)} \cdot \psi_i^{(l,t)}$ measures the energy distribution of signals $S_i^{(l,t)}$ and structural entropy $\psi_i^{(l,t)}$. According to Lagrangian mechanics described in Eq. 45, where the non-relativistic Lagrangian $\mathcal{L}$ for sub-models in a specific subsystem is defined by

$$\mathcal{L} = \mathcal{T} - \mathcal{V} = \mathcal{T} = \frac{1}{2}m_0\dot{r}^2 \tag{46}$$

where r represents the dynamical state of a sub-model. Thus

$$\frac{\partial \mathcal{L}}{\partial \dot{q}} = \frac{\partial \mathcal{L}}{\partial \dot{r}} = m_0\frac{\partial r}{\partial t} \tag{47}$$

Then substitute Eq. 47 into Eq. 45, obtaining

$$m_0\frac{\partial r}{\partial t}\frac{\partial^2 r}{\partial t^2} = \frac{\partial \mathcal{L}}{\partial t} \tag{48}$$

Summing both sides of Eq. 49

$$m_0 \cdot \sum_{k\neq x}\frac{\partial r_{xk}^{(t)}}{\partial t}\cdot\frac{\partial^2 r_{xk}^{(t)}}{\partial t^2} = \frac{m_0}{2}\cdot\sum_{k\neq x}\frac{\partial}{\partial t}\left(\frac{\partial r_{xk}^{(t)}}{\partial t}\right)^2 = \sum_{k\neq x}\frac{\partial \mathcal{L}_{xk}^{(t)}}{\partial t} \tag{49}$$

To satisfy the conservation of momentum and Newton's third law, we have

$$\begin{gathered}\sum_{k\neq x}\left(\frac{\partial r_{xk}^{(t)}}{\partial t}\right)^2 = \left(\frac{\partial r_{xx}^{(t)}}{\partial t}\right)^2 \\ \sum_{k\neq x}\frac{\partial}{\partial t}\left(\frac{\partial r_{xk}^{(t)}}{\partial t}\right)^2 = -\frac{\partial}{\partial t}\left(\frac{\partial r_{xx}^{(t)}}{\partial t}\right)^2\end{gathered} \tag{50}$$

Then the middle term of Eq. 49 can be simplified to

$$\frac{m_0}{2}\cdot\sum_{k\neq x}\frac{\partial}{\partial t}\left(\frac{\partial r_{xk}^{(t)}}{\partial t}\right)^2 = -\frac{m_0}{2}\cdot\frac{\partial}{\partial t}\left(\frac{\partial r_{xx}^{(t)}}{\partial t}\right)^2 = -m_0\cdot\frac{\partial r_{xx}^{(t)}}{\partial t}\cdot\frac{\partial^2 r_{xx}^{(t)}}{\partial t^2} \tag{51}$$

Summing both sides of Eq. 46

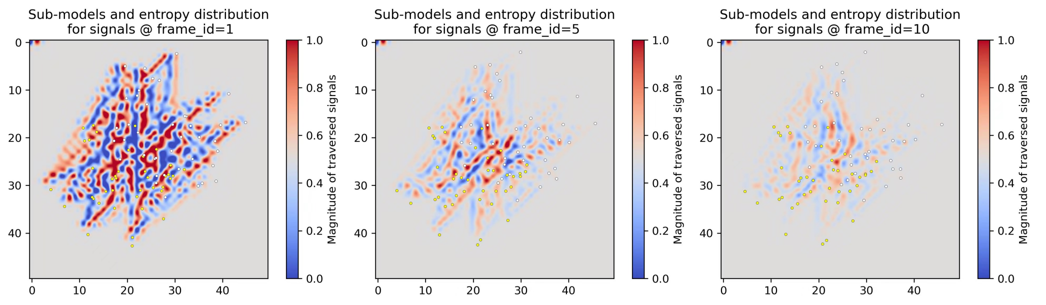
$$\sum_{k\neq x}\mathcal{L}_{xk}^{(t)} = \frac{m_0}{2}\cdot\sum_{k\neq x}\left(\frac{\partial r_{xk}^{(t)}}{\partial t}\right)^2 \tag{52}$$

Then according to Eq. 50, Eq. 52 can be simplified to

$$\sum_{k\neq x}\mathcal{L}_{xk}^{(t)} = \frac{m_0}{2}\cdot\left(\frac{\partial r_{xx}^{(t)}}{\partial t}\right)^2 \tag{53}$$

Then we substitute Eq. 53 into Eq. 51 to eliminate m_0, obtaining

$$\frac{\partial^2 r_{xx}^{(t)}}{\partial t^2} = -\frac{1}{2}\cdot\frac{\sum_{k\neq x}\frac{\partial}{\partial t}\mathcal{L}_{xk}^{(t)}}{\sum_{k\neq x}\mathcal{L}_{xk}^{(t)}}\cdot\frac{\partial r_{xx}^{(t)}}{\partial t} = -\frac{\Lambda_x^{(t)}}{2}\cdot\frac{\partial r_{xx}^{(t)}}{\partial t} \tag{54}$$



(a) Frame-id=1 (b) Frame-id=5 (c) Frame-id=10

Figure 6: **Equivalence between neuromorphic learning and entropy reduction.** As presented by Eq. 56, the sub-models tend to move toward the region with lower structural entropy, which is visualized by colored spatial distribution.

Note that $\mathcal{L}$ can be approximated as time-invariant when $\partial t \mapsto 0$ since $\mathcal{L}$ varies with the combination of all sub-models and signals, whose overall dynamics are relatively static for a particular sub-model. Then we solve the differential Eq. 54, which yields

$$\frac{\partial r_{xx}^{(t)}}{\partial t} = \eta_x \cdot \exp\left(- \frac{\Lambda_x^{(t)}}{2} \cdot t \right) \tag{55}$$

where the entropy indicator $\Lambda_x^{(t)}$ measures the structural entropy (can be evaluated via methods similar to Eq. 13) of $\mathcal{L}$ over the system of sub-models, and η_x is a constant value related to its corresponding sub-model $q_x^{(t)}$.

The comprehensive form of Eq. 55 is as follows:

$$\frac{\partial r_i^{(l,t)}}{\partial t} = \eta_i \cdot \exp\left(- \frac{\sum_{k \neq i} \frac{\partial}{\partial t} \mathcal{L}_{ik}^{(l,t)}}{\sum_{k \neq i} \mathcal{L}_{ik}^{(l,t)}} \cdot t \right) = \eta_i \cdot \exp\left(- \Lambda_i^{(l,t)} \cdot t \right) \tag{56}$$

where $r_i^{(l,t)}$ is the positional vector of a sub-model, and η_i is a constant related to $q_i^{(l,t)}$. This equation is equipped with an unspecific Lagrangian $\mathcal{L}$, for instance, $\mathcal{L}_{ik}^{(l,t)} = S_{ik}^{(l,t)} \Phi_i^{(l,t)}$, where $S_{ik}^{(l,t)}$ is emitted from $q_i^{(L,t^*)}$ and received by $q_k^{(l,t)}$, being influenced by the resultant potential field $\Phi_i^{(L,t)}$ around $q_i^{(L,t)}$. The signals $S_{ik}^{(l,t)}$ refers to the feedback control correlated with the loss function for the current task, and the potential field $\Phi_i^{(l,t)}$ is a trainable parameter related to distinct sub-models. Note that we can simplify $\Phi_i^{(l,t)}$ as a constant field by adding shared coefficients applied in Eq. 5.

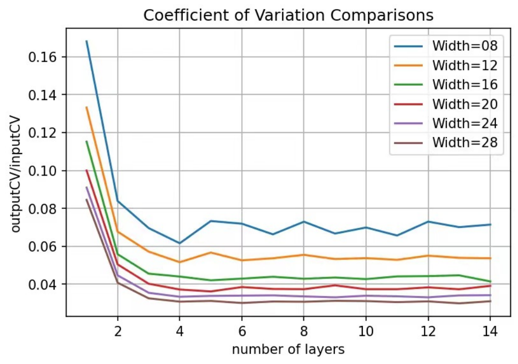
H. Conservation of workload and computational complexity with increasing number of sub-models

By Hamilton's principle, the variables in Eq. 54 and Eq. 55 should have several restrictions, including

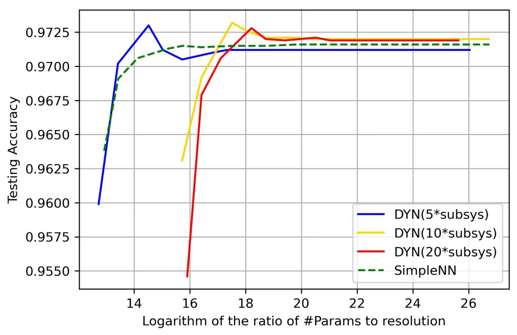
$$\begin{gathered} \sum_x \eta_x = C_\eta \\ \sum_x \sum_{k \neq x} \mathcal{L}_{xk}^{(t)} = C_{\mathcal{L}} \\ \sum_x \sum_{k \neq x} \frac{\partial}{\partial t} \mathcal{L}_{xk}^{(t)} = C_{\partial \mathcal{L}} \end{gathered} \tag{57}$$

where C_η, $C_{\mathcal{L}}$ and $C_{\partial\mathcal{L}}$ are time-invariant constants. Therefore, the summation of the entropy indicator $\Lambda_x^{(t)}$ can also be approximated as a time-invariant constant

$$\lim_{N \mapsto \infty} \sum_{x=1}^{N} \Lambda_x^{(t)} \approx \frac{C_{\partial \mathcal{L}}}{C_{\mathcal{L}}} = N \cdot \bar{\Lambda} \tag{58}$$



(a)



(b)

Figure 7: (a) The ratio of output C_v to input C_v as the number of layers increases with different numbers of sub-models in the hidden layer (width). (b) The horizontal axis measures the logarithm of the ratio of parameters' size and resolution (LRPR); the vertical axis measures the testing accuracy of the truncated models corresponding to each specific LRPR.

where N is defined as the total number of sub-models, and $\bar{\Lambda}$ is defined as a constant referring to the averaged entropy indicator. Therefore, the total path length of all sub-models can be approximated in terms of Eq. 54 as a time-variant function $I(t)$

$$I(t) = \sum_{x=1}^{N} \frac{\partial r_x^{(t)}}{\partial t} = \sum_{x=1}^{N} \eta_x \cdot \exp\left(-\Lambda_i^{(t)} \cdot t\right) \approx C_\eta \cdot \exp\left(-\bar{\Lambda} \cdot t\right) \tag{59}$$

The time-step t is a small value since the sub-models of a DyN system generate signals almost instantaneously. The total workload $W(T)$ that is linearly correlated with the computational complexity is evaluated

$$W(T) = \int_0^T I(t)dt = \frac{NC_\eta C_{\mathcal{L}}}{C_{\partial\mathcal{L}}} \cdot (1 - e^{-\bar{\Lambda}T}) \tag{60}$$

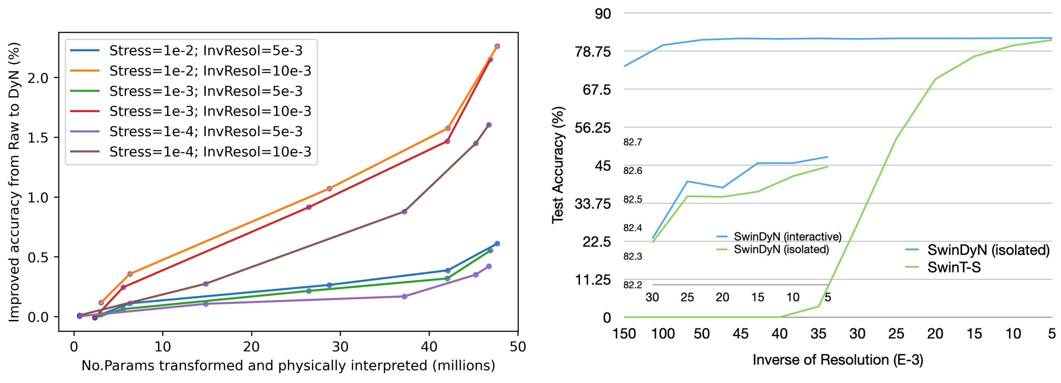
where T is the total time required to reach an equilibrium state. Based on Eq. 60, as the required number of sub-models increases largely to deal with an increasingly complicated computational task, the total workload and computational complexity do not increase accordingly but gradually approach a specific value. This fact also implies a neuro-biological correlation that when the brain arises a concept, the power of the cortical regions related to the concept remains unchanged after several learning events that supply more specified knowledge.

I. Learning with more subsystems and more sub-models

We apply an interactive mechanism to the current architecture to boost the computational ability of a DyN system without explosive growth of computational complexity. The principle of hierarchical structures (Eq. 33) implies that a well-formed neuromorphic system does not require a specified set of discrete trainable units isolated from each other. Besides, inspired by the phase space density representation (Abarbanel & Rouhi, 1987), a dynamic system represented by infinite interactive particles can be treated as a linear combination of many shallow layers, each of which is interpreted as an isolated dynamic system of different density. Specifically, each layer of density ρ_i refers to a subsystem with a specific shared coefficient h_i, which disassembles the overall neuromorphic system into several partially independent subsystems. Therefore, a discrete sub-model $q_{i;k_i}^{(l;t)}$ is equivalent to an interactive region with a density of h_{k_i}. The variational density $g_{ij} = \rho_{k_i} - \rho_{k_j} = h_{k_i} - h_{k_j}$ measures the potential energy generated by the interaction (receiving or emitting signals) between sub-models

$$m_i \frac{\partial}{\partial t} q_{i;k_i}^{(l,t)} = \int_{q_{j;k_j}^{(l,t)} \in P^{(l)}} \psi(v_{ij}^{(l,t)}, f_{ij}^{(l,t)}, g_{ij}^{(l,t)}) dP^{(l)} \tag{61}$$

where m_i is a constant related to the density h_{k_i}, and ψ is a linear transformation concatenates sub-models' variational dynamics. The notations here are consistent with Eq. 6. Therefore, a DyN system with infinite sub-models can be approximated as the one with finite subsystems, in which the dynamical states of sub-models are interactively correlated with other sub-models from all subsystems. The increased number of computing units has led to some burden in the software implementation but no increase in the overall computational complexity. This fact is validated experimentally and mathematically (Appendix H).



(a) (b)

Figure 8: (a) As the number of parameters transformed and instantiated via physical view increases, the DyN models outperform their original forms more significantly as the improved accuracy increases. The presented models are distinguished in terms of the inverse of resolution δ and the stress-threshold that determines when DyN process ends. (b) The $SwinDyN$ refers to the pre-trained SwinT-S-224 model whose neural layers are converted into DyN forms. As δ increases, though the original model performance greatly degrades, the DyN alternatives remain almost unchanged. The interactive one (Eq. 61) outperforms the isolated one (Eq. 6). This phenomenon is attributed to the global dynamics of sub-models that are moving and interacting with others in a fully stabilized manner.

J. Limitations

Initialization and convergence. In most cases, with a better initialization and a proper setting of hyper-parameters, there could be a phenomenon of few-step convergence. However, this phenomenon is not stable if we initialize DyN with some extreme settings. Even though the overall convergence rate of DyN is better than ANN, we still need to dig inside before reaching a validated conclusion.

Simultaneity and time delay. In our simulation and the experiments, we update the sub-models similarly to an ANN (epochs-by-epochs). An epoch is undertaken as follows, *i.e.*, the feedback signals result in the stress forces amongst each sub-model, and the stress forces cause the sub-models to change their dynamical states. This setting works in the software implementation, and we do not need to guarantee that the neurons are strictly updated simultaneously. However, in the idealized setting where the physical properties matter, we need to consider the issues with simultaneity.

Memory usage and parallelism. In the common cases of DyN, the memory used in the inference stage is much lower than that of ANN. For model training, the memory consumed by the trainable parameters (neural states in sub-models) is far less than that of an ANN. Nevertheless, we need extra memory to expand the dynamical states to compute the path integrals. The upper bound of the total memory, including the trainable parameters and the calculation of path integrals, would not exceed the original ANN (Lemma C.2). However, there is still room for further reduction of the memory consumption for model training.