Recurrent neural network dynamical systems for biological vision

Supplementary material

A Recurrent neural network dynamical systems in neuroscience

Single neuron biophysical models. The dynamics of single neurons must be captured to some reasonable fidelity in order to develop reliable models of networks. Two of the most widely studied models for single neurons are the Hodgkin-Huxley model [1] and the leaky integrate-and-fire model [2]. The Hodgkin-Huxley model was developed to explain the ionic mechanisms underlying action potential generation and propagation in neurons, using the squid giant axon as the experimental model. However, the complexity of this model limits its use in large-scale simulations involving thousands of neurons. Simplified models like the leaky integrate-and-fire model have been developed to address this challenge. This model abstracts away the ion channel dynamics and instead focuses on capturing the core behavior of a spiking neuron.

Handcrafted networks. Before the accessibility of deep learning tools, neuroscientists have proposed models of RNNs by manually constructing their recurrent weights. One such example is the Hopfield network [3], which stores patterns and recalls them even from incomplete or corrupted inputs. This is achieved by constructing its weights based on a Hebbian learning rule that helps form attractors corresponding to the stored patterns [4]. Handcrafted continuous-time RNN dynamical systems have also been suggested to explain various observations in neuroscience literature, such as the functioning of head-direction cells [5] and how the brain maintains stable eye positions [6].

Another important subfield in neuroscience studies the dynamical properties of RNN dynamical systems where the recurrent weights are randomly generated from some (controlled) distribution [7, 8]. This was motivated by the need to study large-scale network behavior before the advent of deep learning. Various dynamical motifs, such as oscillatory and chaotic regimes, have been identified in RNNs from such models through autocorrelations and Lyapunov exponents analyses.

Trained networks. Once artificial RNNs are adapted for continuous-time dynamics and applied to biologically relevant tasks, they can be used to study various aspects of brain function. One approach is to compare the emergent properties of RNN activity with the patterns observed in real neurons. For instance, RNNs can generate low-dimensional dynamical structures, such as line attractors, which correspond to stable states or trajectories in neural state space [9]. This approach was first applied to investigate how the prefrontal cortex integrates sensory inputs to guide context-dependent decisions, focusing on how neurons in this brain region handle the selection of relevant information while ignoring irrelevant inputs [10].

Deep equilibrium models are a class of artificial neural networks that are remarkably similar to RNN dynamical systems [11]. At its core, deep equilibrium models rely on fixed points of single layers (equivalent to infinitely stacking the same layer) instead of relying on multiple layers. However, the similarities start to break down when considering the order of computations in the presence of multiple layers. In RNNs with multiple layers, the activations of every layer are updated sequentially at each time step. In the case of deep equilibrium models, a single layer is first simulated to its fixed point before any computation in the next layer is performed.

In a different approach, continuous-time RNN dynamical systems were trained on a wide variety of cognitive tasks simultaneously [12]. This work investigates how such RNNs organize their internal architecture and how their recurrent units become functionally specialized for different aspects of cognition. The analysis of these functional clusters revealed that tasks that required similar cognitive processes, such as decision-making across sensory modalities, recruited overlapping clusters of neurons. This indicates that the network reused certain clusters across tasks that shared underlying computational principles.

1

A.
Kernel size = 1
Example kernels Weight matrix Singular values

Kernel size = 7
Example kernels Weight matrix Singular values

B.
Oscillatory regime

Chaotic regime
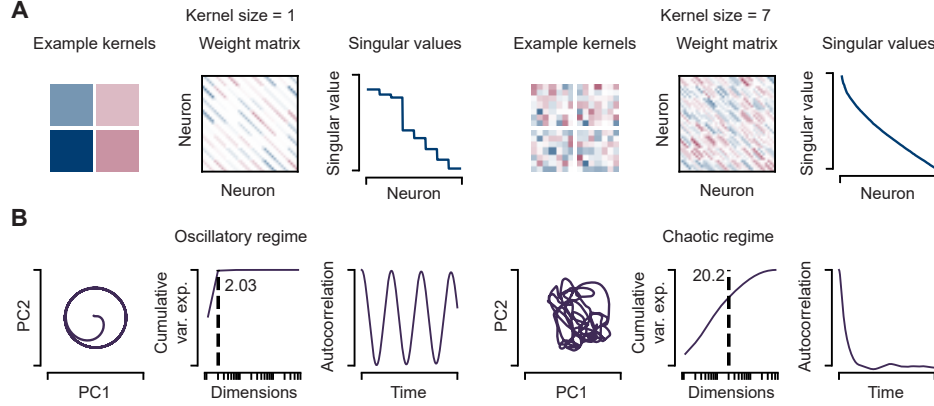
Figure S1: **A.** Examples of convolutional kernels, their resultant flattened weight matrices as described in equation (2) and their distribution of singular values. **B.** Gaussian-initialized CordsNets exhibiting an oscillatory regime (left) and a chaotic regime (right). In each regime, we plot the population trajectory of the space of the first two principal components (left), the cumulative variance explained with participation ratio indicated (middle), and the activity autocorrelation (right).

B Analysis of dynamical characteristics

B.1 Converting a convolution into a 2-D weight matrix

CordsNets replace the recurrent and input weight matrices of dynamical RNNs with convolutional operations. This is possible due to the fact that convolutions are linear operations, and can therefore be rewritten as a 2-D matrix operating on a flattened 1-D vector. Here, we review the steps required to do this on a single channel. Consider an input matrix $\mathbf{X}$ of size $H_X \times W_X$ and a convolutional kernel $\mathbf{K}$ of size $H_K \times W_K$. Applying the convolutional kernel $\mathbf{K}$ onto the input $\mathbf{X}$ would result in an output $\mathbf{Y}$ with size $(H_X - H_K + 1) \times (W_X - W_K + 1)$, assuming no padding and a stride of 1. The (i, j)-th element of $\mathbf{Y}$ can be computed as:

$$Y_{i,j} = \sum_{m=0}^{H_K-1} \sum_{n=0}^{W_K-1} X_{i+m,j+n} K_{m,n} \tag{1}$$

This can be flattened into a linear operation:

$$\mathbf{Y}^{\text{flat}} = \mathbf{K}^{\text{flat}} \mathbf{X}^{\text{flat}} \tag{2}$$

where $\mathbf{X}^{\text{flat}}$ is a $H_X W_X \times 1$ vector representing the flattened input. $\mathbf{K}^{\text{flat}}$ would therefore have size $(H_X - H_K + 1)(W_X - W_K + 1) \times H_X W_X$. The elements of $\mathbf{K}^{\text{flat}}$ can be determined by gathering the terms of $\mathbf{K}$ that are multiplied with a specific $X_{i+m,j+n}$ in equation (1). This is visualized in Figure S1A, along with the singular values of $\mathbf{K}^{\text{flat}}$. We see that the singular values decay smoothly for sufficiently large kernel sizes, just like in full-rank matrices.

B.2 Randomly-initialized CordsNets

Gaussian-initialized dynamical RNNs with tanh activation functions have been extensively studied in neuroscience literature [7, 13]. Depending on the variance of the Gaussian initialization, fully-connected RNNs can exhibit three distinct dynamical regimes: stable, oscillatory and chaotic. In the stable regime, the network converges to a stable fixed point and remains stationary. In the oscillatory regime, the network evolves over time in a periodic manner in low-dimensional activity space (Figure S1B, left). We estimate activity dimensionality from its participation ratio (PR):

$$\text{PR} = \frac{\left(\sum_i \lambda_i\right)^2}{\sum_i \lambda_i^2} \tag{3}$$

where λ_i represents the i-th eigenvalue of the activity covariance matrix [14, 15]. The activity autocorrelation also reflects this periodicity (Figure S1B, left). In contrast, the chaotic regime

2

is characterized by aperiodic and unpredictable behavior. This regime is identified by a positive Lyapunov exponent, indicating exponential divergence of nearby trajectories. Network activity has a much higher dimensionality compared to the oscillatory regime, and the autocorrelation decays quickly to zero (Figure S1B, right).

B.3 Comparison with other architectures

We compare the solutions found by training CordsNets, fully-connected RNNs, low-rank RNNs [16] and sparsely-connected RNNs [17] on five cognitive tasks across different activation functions, learning rates, network sizes and initializations. In addition to CordsNets, the other architectures in our comparison are:

- Fully-connected RNNs have full-rank weight matrices of size $N \times N$, where N is the number of neurons.

- Low-rank RNNs have low-rank weight matrices with rank $R \ll N$. This is implemented by decomposing the matrix into two smaller matrices $\mathbf{P}$ and $\mathbf{Q}$ with sizes $N \times R$ and $R \times N$ so that $\mathbf{W}_{\text{low-rank}} = \mathbf{PQ}$.

- Sparsely-connected RNNs have sparse weight matrices, implemented by an element-wise mask $\mathbf{W}_{\text{sparse}} = \mathbf{W} \odot \mathbf{M}$, where every element in the mask $\mathbf{M}$ is 1 with some probability p and 0 otherwise.

The cognitive tasks are the same tasks previously adopted in the analysis of low-rank RNNs [16]:

- The **perceptual decision-making** (PDM) task consists of a fixation epoch of 100 ms, followed by a stimulus epoch of 800 ms, a delay epoch of 100 ms and a decision epoch of 20 ms. During the stimulus epoch, a noisy signal (Gaussian with standard deviation 0.1) drawn uniformly from $\{\pm 0.4, \pm 0.2, \pm 0.1\}$ is presented as input to the networks. The output of the networks during the decision epoch should indicate the sign of the signal.

- The **parametric working memory** task (PWM) consists of a fixation epoch of 100 ms, a stimulus epoch of 100 ms, a delay epoch randomly drawn from 500 ms to 2000 ms, a second stimulus epoch of 100 ms and a decision epoch of 100 ms. In each stimulus epoch, a signal drawn uniformly from $\{10, 11, ..., 34\}$ is presented to the network. The output of the networks during the decision epoch should compute the normalized differences in values of the two signals.

- The **contextual decision-making** task (CDM) consists of a fixation epoch of 100 ms, a context epoch of 350 ms, a stimulus epoch of 800 ms, a second context epoch of 500 ms and a decision epoch of 20 ms. The networks have four inputs: 2 cues and 2 noisy signals drawn uniformly from $\{\pm 0.4, \pm 0.2, \pm 0.1\}$. One of the cues is set at 0.1 for a given trial, while the other is set to 0. The networks are expected to output the sign of one of the signals depending on which cue is non-zero.

- The **multi-sensory decision-making** task (MDM) has the same task structure and network inputs as CDM, except the 2 noisy signals have the same sign. Similarly, the networks are expected to output either signal during the decision epoch.

- The **delayed match-to-sample** task (DMS) consists of a fixation epoch of 100 ms, a stimulus epoch of 500 ms, a delay epoch randomly drawn from 500 ms to 3000 ms, a second stimulus epoch of 500 ms and a decision epoch of 1000 ms. In both stimulus epochs, one of two possible signals is presented, and the output of the networks during the decision epoch should indicate whether the signals presented in both stimulus epochs are the same.

Table S1: Number of trainable parameters in the recurrent weights of different network architectures across three different sizes. To match parameter counts, we vary the kernel size of CordsNets, weight matrix rank of low-rank RNNs and weight matrix sparsity of sparsely-connected RNNs.

Neurons	CordsNet	Low-Rank RNN	Sparse RNN
125	400 (kernel size 4)	500 (rank 2)	500 (sparsity 0.032)
216	900 (kernel size 5)	864 (rank 2)	864 (sparsity 0.0185)
512	2304 (kernel size 6)	2048 (rank 2)	2048 (sparsity 0.008)

3
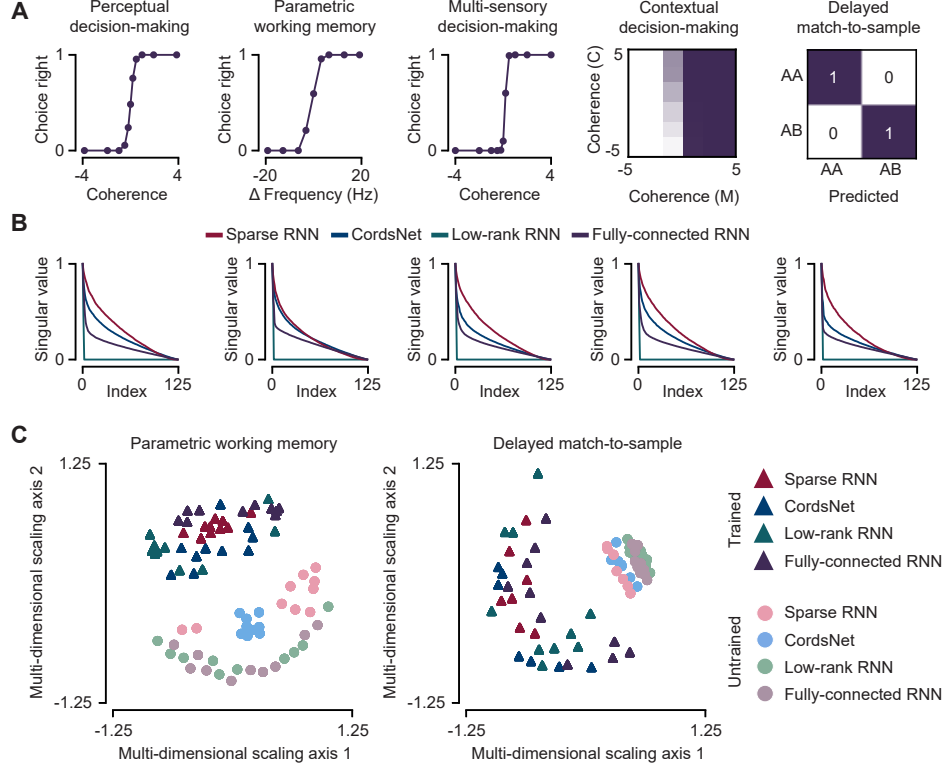
Figure S2: **A.** Examples of psychometric curves of CordsNets trained to solve five cognitive tasks. **B.** Normalized singular value distribution for different architectures on each respective task. **C.** Multi-dimensional scaling plots of distance matrices obtained after aligning trajectories using Procrustes analysis for the parametric working memory task (left) and the delayed match-to-sample task (right).

For each architecture-task pair, we trained 10 networks across three learning rates $(10^{-2}, 10^{-3}, 10^{-4})$, three activation functions (ReLU, tanh and softplus) and three network sizes (Table S1). We successfully trained all networks on all tasks. Figure S2A shows examples of psychometric curves for CordsNet on all five tasks. We also computed the singular values of the recurrent weights for all architectures (Figure S2B) and consistently found that CordsNets have singular value distributions that are the most similar to fully-connected RNNs across all hyperparameters.

More importantly, we want to compare the neural trajectories of all architectures and quantify how close they are to the trajectories of fully-connected RNNs. For each task and across every hyperparameter setting, there are:

$$(2 \text{ trained/untrained}) \times (4 \text{ architectures}) \times (10 \text{ random initializations}) = 80 \text{ networks} \quad (4)$$

to be considered. Between each pair of networks, we align their neural trajectories using Procrustes alignment and compute the resultant Procrustes distance, which ultimately gives us a 80×80 distance matrix for one particular task and hyperparameter setting. This matrix can be visualized using multi-dimensional scaling [18, 19] (Figure S2C). We observe that networks of all architectures have similar aligned trajectories when they are either all trained or all untrained. This suggests

Table S2: Mean distance compared to fully-connected RNN of each architecture-task pair.

Task	Sparse RNN	CordsNet	Low-Rank RNN
PDM	**0.62** ($\pm$0.02)	0.64 ($\pm$0.03)	0.77 ($\pm$0.02)
PWM	0.84 ($\pm$0.02)	**0.75** ($\pm$0.02)	0.85 ($\pm$0.01)
MDM	0.75 ($\pm$0.02)	**0.73** ($\pm$0.03)	0.76 ($\pm$0.02)
CDM	**0.69** ($\pm$0.01)	0.74 ($\pm$0.03)	0.66 ($\pm$0.01)
DMS	0.94 ($\pm$0.04)	**0.86** ($\pm$0.05)	0.92 ($\pm$0.07)

4
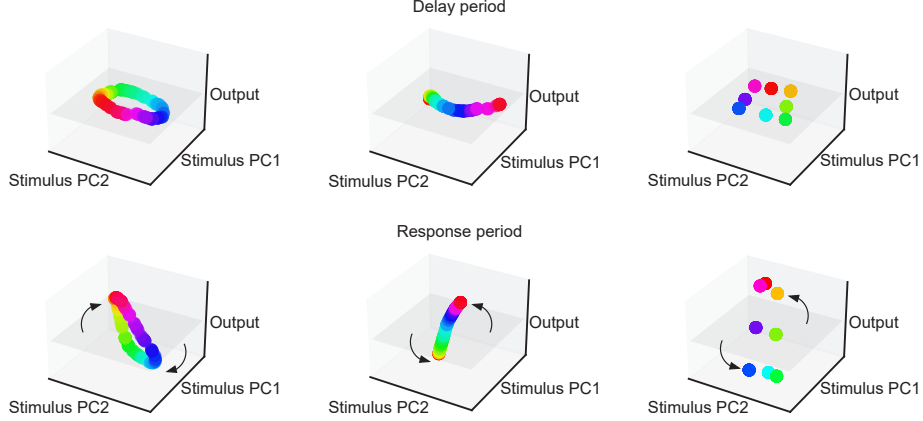
Figure S3: Evolution of neural activity across time in CordsNets trained to perform a memory-pro delayed-response task [20]. During the delay epoch, neural activity converges to different fixed/slow points depending on the stimulus that was presented in the previous epoch. Depending on the type of stimulus presented, the trained networks may exhibit ring (left), line (middle) or point attractors (right) during the delay epoch. In the response epoch, neural activity rotates to the output axis and approximately preserves the same geometry [20].

that all network architectures are solving the tasks in similar ways. Finally, we compute the mean distances of each network architecture to fully-connected RNNs (Table S2) averaged across all tasks and hyperparameters, and find that on average, CordsNets have the shortest mean distance to fully-connected RNNs. We conclude that the convolutional recurrent structure of CordsNets does not significantly restrict dynamical expressivity.

B.4 Attractor formation

A key dynamical feature of RNNs is their ability to retain information about past stimuli over time through attractor states, enabled by carefully tuned recurrent weights. We want to check if the convolutional structure of CordsNets would prevent these attractors from forming. To do this, we train CordsNets on a memory-pro delayed-response task [20]. The task consists of epochs of random duration, starting with a fixation epoch lasting between 300 ms to 700 ms, a stimulus epoch lasting between 200 ms to 1600 ms, a delay epoch lasting between 200 ms to 1600 ms, and a response epoch lasting between 300 ms to 700 ms. The random epoch durations have been known to promote attractor formation. During the stimulus epoch, the network input can represent a circular variable, a continuous linear variable, or a variable with 8 discrete states. The network output in the response epoch should match the initial input. For each type of input, we successfully trained CordsNets to generate ring, line, and point attractors, respectively, during the delay epoch (Figure S3, top). During the response epoch, neural activity shifts out of the output null space, creating a rotational effect that approximately preserves the structure of the attractors. These results confirm that CordsNets are able to manifest attractors despite their restricted weight structures.

C Training for image classification

C.1 Model architecture

The architectures of the CordsNets trained to perform image classification are described in Table S3. We generally follow the layer structure of ResNet-18 [21], but with two key modifications: we omit normalization and replace max pooling with average pooling, as averaging is a linear operation and thus more compatible with the dynamics of RNNs. Another important design choice that we made is to keep residual connections in our models. The brain performs cognition in a highly distributed manner, leveraging a multitude of interconnected regions that work in concert to process information, solve problems, and generate behaviors. Residual connections avoid the simplicity of a single-path structure and allow for more sophisticated pathways for information to travel through the network.

5

Table S3: CordsNet architectures for image classification. In each block, the top convolution represents a feedforward transformation, while the bottom convolution represents the recurrent weights in the recurrent dynamical system.

Output size	CordsNet-R2	CordsNet-R4	CordsNet-R6	CordsNet-R8
112×112	\multicolumn{4}{c}{3x3 average pool, stride 2}			
56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
28×28		$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
14×14			$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
7×7				$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
1×1	\multicolumn{4}{c}{average pool, linear, softmax}			

C.2 Training details

For all datasets used in our study, we selected image augmentation techniques widely recognized as universally beneficial [22]. Additional details can be found in Table S4

- **MNIST.** `RandomAffine` (5 degrees random rotation, 5% translation, 0.05 scaling factor and 5 degrees shear angle), `ColorJitter` (10% brightness, 10% contrast), `ElasticTransform` (scaling factor 20, smoothness 5)

- **Fashion-MNIST.** `RandomHorizontalFlip`, `RandomAffine` (5 degrees random rotation, 5% translation, 0.05 scaling factor and 5 degrees shear angle), `ColorJitter` (10% brightness, 10% contrast), `CutMix`, `MixUp`

- **CIFAR-10/CIFAR-100.** `RandomHorizontalFlip`, `RandomAugment`, `CutMix`, `MixUp`

- **ImageNet.** `RandomHorizontalFlip`, `RandomAugment`

Table S4: Training specifics for each step of our proposed method to efficiently initialize CordsNet. In step 3, the PReLU parameter a is varied from 0.95 to 0 in 20 steps of 0.05. For ImageNet, 0.1 epochs corresponds to 16000 iterations.

Dataset	Epochs	Optimizer	Learning rate	Batch size
Step 1 – Feedforward CNN				
ImageNet	3×30	SGD	`1e-1, 1e-2, 1e-3`	256
Others	3×100	SGD	`1e-1, 1e-2, 1e-3`	256
Step 2 – Linear RNN dynamical system				
ImageNet	30	AdamW	`1e-5`	32
Others	100	AdamW	`1e-5`	32
Step 3 – Parametric annealing				
ImageNet	20×0.1	AdamW	`1e-5`	8
Others	20×1	AdamW	`1e-5`	8
Step 4 – Fine tune				
ImageNet	20	AdamW	`1e-5`	8
Others	20	AdamW	`1e-5`	8

6

Table S5: Time required (in hours) to complete each step of our proposed initialization method, as benchmarked on a server with 2x RTX 4090 GPUs. The time required to train CordsNet for one full epoch is shown as a control, and the equivalent number of epochs (by time taken) using our method is computed in the final column.

Model	Dataset	Step 1	Step 2	Control	Epochs needed
R2	MNIST	5.99	22.82	1.46	39.70
	F-MNIST	4.05	20.65	1.47	36.77
	CIFAR-10	4.20	19.03	1.25	38.53
	CIFAR-100	4.06	19.86	1.24	39.25
	ImageNet	26.14	145.9	31.87	8.40
R4	MNIST	6.06	38.05	2.86	35.40
	F-MNIST	4.29	36.75	2.88	34.21
	CIFAR-10	4.22	31.85	2.40	35.02
	CIFAR-100	4.26	32.34	2.34	35.67
	ImageNet	27.44	245.2	62.27	7.38
R6	MNIST	6.06	50.99	4.31	33.24
	F-MNIST	4.50	49.95	4.26	32.77
	CIFAR-10	4.39	43.22	3.60	33.23
	CIFAR-100	4.44	43.45	3.55	33.49
	ImageNet	28.79	334.2	90.85	7.00
R8	MNIST	6.10	67.44	5.65	33.00
	F-MNIST	4.84	65.76	5.63	32.85
	CIFAR-10	4.82	56.53	4.73	32.97
	CIFAR-100	4.70	57.50	4.71	33.19
	ImageNet	31.20	439.2	121.04	6.89

C.3 Time benchmark

We train every model-dataset combination with a variable number of GPUs depending on memory requirements, as shown.

- **CordsNet-R2.** 8x RTX 4090 (ImageNet), 2x RTX 4090 (others)

- **CordsNet-R4.** 4x H100 80GB (ImageNet), 4x RTX 4090 (others)

- **CordsNet-R6.** 4x H100 80GB (ImageNet), 4x RTX 4090 (others)

- **CordsNet-R8.** 8x H100 80GB (ImageNet), 8x RTX 4090 (others)

One of the control experiments we have tested is to train a separate model in the same amount of time our proposed method took. In order to obtain an accurate estimate, we run every step of every model on a single server with 2x RTX 4090 GPUs. We simulated the training for approximately 5 minutes for each step and extrapolated the time required for the entire step to be complete. We then computed the number of epochs needed to be simulated by our control experiment by:

$$\text{Control epochs} = \frac{\text{Time taken for step 1} + \text{Time taken for step 2}}{\text{Time taken for 1 control epoch}} + \text{Epochs in step 3} \quad (5)$$

This is because the parametric annealing step in step 3 involves training with the full loss function, which is essentially equivalent to 1 control epoch. The benchmark times are reported in Table S5. The duration of each epoch is influenced by the specific image augmentation techniques we have chosen to employ. The time taken for the MNIST dataset is particularly high due to the `ElasticTransform` augmentation, which is notably time-consuming. Training on the MNIST datasets takes longer than on the CIFAR datasets because of the larger MNIST training sets. For simplicity and to account for variability, we keep the number of control epochs at 10 for ImageNet and 40 for the other datasets.

7
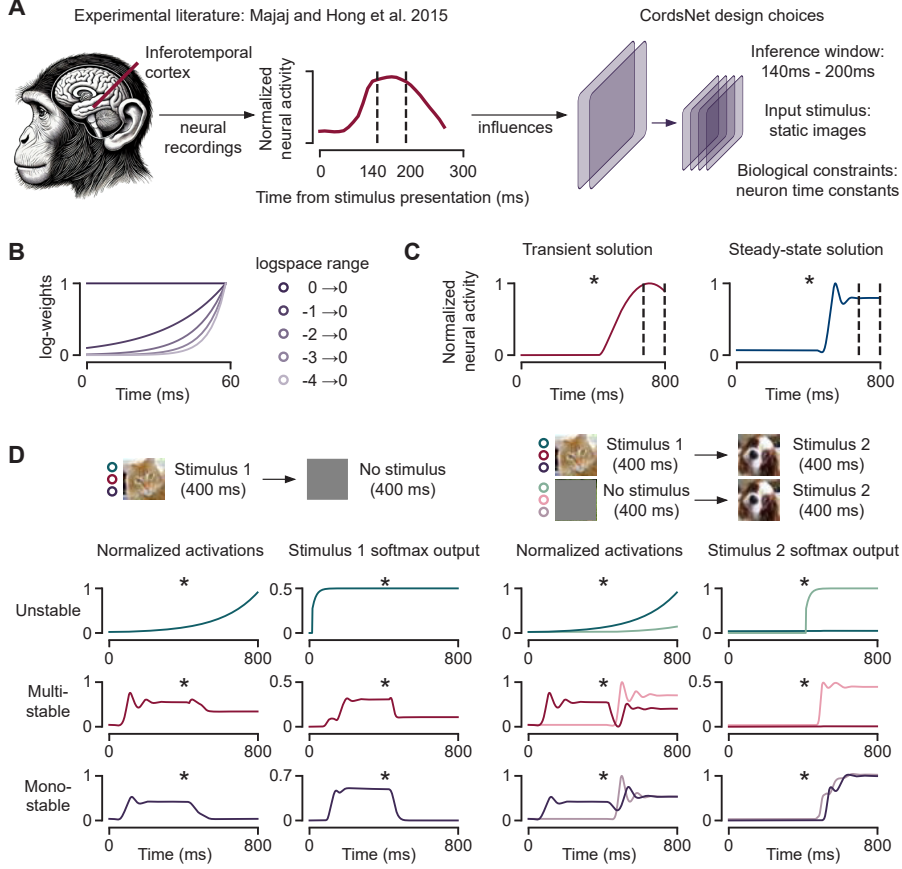
Figure S4: **A.** Building artificial neural network models of the biological visual system based on experimental literature [23]. The inference window of our loss function is derived from neural recordings. **B.** Logarithmic scale applied to the cross-entropy loss terms at different time steps. **C.** Trained networks can either classify images in a transient state where network activity is changing within the time window (left) or a stationary state where activity is unchanged (right). **D.** Without the spontaneous penalty term in the loss function, networks can exhibit three classes of solutions: an unstable solution where network activity blows up after the inference window (top, green), a multi-stable solution where the network remains stable after the first inference window but fails to classify subsequent images (middle, red), and a mono-stable solution that correctly classifies any number of inputs presented sequentially across time (bottom, purple).

D   Loss function ablation study

The loss function we ultimately selected for training our networks on image classification is closely related to the standard template of a cross-entropy loss term with some optional regularization terms. We have a log-weighted sum of cross-entropy losses over 30 time steps and a spontaneous penalty term (which is a form of regularization). We first reintroduce the loss function here:

$$
\begin{aligned}
\text{loss} = &\overbrace{\texttt{logspace(-3,0,steps=30)}}^{\text{log-weighting (Figure S4B)}} * \texttt{CEloss(output[}\overbrace{\texttt{170:200}}^{\text{Inference window (Figure S4A)}}\texttt{],labels)} \\
&+ \underbrace{\texttt{1e-3 * MSEloss(activity[290:300],spontaneous)}}_{\text{Spontaneous penalty (Figure S4C)}}
\end{aligned}
\tag{6}
$$

As stated in the main text, the network is simulated for 100 time steps (interpreted as 2 ms per time step) without any input to allow the network to arrive at a steady state spontaneous activity level (`spontaneous`). The inference time window of `[170:200]` corresponds to 140 ms to 200 ms after stimulus presentation. This time window was selected based on experimental recordings [23] showing

8

Table S6: Ablation studies for the `logscale` range used to weigh the cross-entropy loss terms across time (top) and the coefficient of the spontaneous penalty term (bottom).

	`logscale` range	
Range	Steady-state solution	Transient solution
$0 \to 0$	8	2
$-1 \to 0$	10	0
$-2 \to 0$	10	0
$-3 \to 0$	10	0
$-4 \to 0$	10	0
	Spontaneous penalty coefficient	
Coefficient	Mono-stable solution	Other solutions
0	7	13
10^{-5}	16	4
10^{-4}	20	0
10^{-3}	20	0
10^{-2}	20	0
10^{-1}	20	0

heightened neural activity in the inferotemporal cortex of macaque monkeys following stimulus presentation (Figure S4A). Just like in conventional supervised learning settings, we calculate the cross-entropy loss across all 30 time steps within the selected window and sum them after weighing them with a logarithmic scale across time (Figure S4B). The purpose of this term is to ensure that we do not get solutions where the accuracy peaks in the middle of the inference window and drops off towards the end of the window, which we refer to as a transient solution. Instead, we want a network where neural activity is stable throughout, which we refer to as a steady-state solution (Figure S4C). We train 10 CordsNet-R4s on CIFAR-10 across various `logscale` ranges to verify this. We find that varying the range of the `logscale` does not significantly impact the trained networks, as long as it is used (Table S6).

A fundamental characteristic of the brain is that it runs continuously, unlike artificial networks that reset (or shut down) after each inference. Therefore, after our network correctly classifies an image, we want it to accurately classify subsequent images, starting from the steady-state activity produced by the previous image. Intuitively, we reason that this property can only be attained by a mono-stable network, where there are no other fixed points in the vicinity of the activity space around its spontaneous activity level. The main property of a mono-stable network is that it returns to the same spontaneous activity level (before stimulus onset) after the presented image has been removed. This motivates the spontaneous penalty term in our loss function. Without this term, we find that trained networks can either be unstable (Figure S4D, green) where network activity blows up after the first inference window, multi-stable (Figure S4D, red) where the network remains stable after first inference but does not return to the original spontaneous activity level, or mono-stable ((Figure S4D, purple) which is the desired property. We train 20 CordsNet-R4s on CIFAR-10 across 6 different spontaneous penalty coefficient values. We find that we need a sufficiently large coefficient (Table S6) to prevent unwanted solutions (unstable and multi-stable) from emerging in our trained networks.

E Fitting to neural data

The neural similarity metric computed in the main text is simply the original Brain-Score [24] extended to fit across all time steps rather than time-averaged quantities. Let N_t be the number of time steps, N_i be the number of images, N_1 be the number of neurons in CordsNet and N_2 be the number of neurons in the experimental recording. In each trial, we use a 90/10 train-test split (for a total of 10 splits), such that for each training split, we fit a $[N_t N_i, N_1]$ matrix of CordsNet neural activity with a $[N_t N_i, N_2]$ matrix of experimental data [23] using 25-component partial least squares regression. We compute the Pearson correlation coefficients of all N_2 neurons for each test split and select the median value. We then computed the mean coefficient across all 10 splits. We do not apply

9

any noise ceiling correction. We repeat the fitting process after shuffling the neural data in the time axis. The entire process of fitting shuffled and unshuffled data is repeated over 20 trials. Finally, we perform a paired t-test across the 20 data points to look for any statistically significant differences between the scores when fitting on shuffled and unshuffled data.

F   Code availability

Code for training and analyzing CordsNets, along with selected trained checkpoints, can be found at:

`https://github.com/wmws2/cordsnet`

References

[1] Hodgkin, A. L. & Huxley, A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology* **117**, 500–544 (1952).

[2] Knight, B. W. Dynamics of encoding in a population of neurons. *The Journal of General Physiology* **59**, 734–766 (1972).

[3] Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences* **79**, 2554–2558 (1982).

[4] Battista, A. Low-dimensional continuous attractors in recurrent neural networks: from statistical physics to computational neuroscience. *Université Paris sciences et lettres* PhD thesis (2020).

[5] Zhang, K. Representation of spatial orientation by the intrinsic dynamics of the head-direction cell ensemble: a theory. *Journal of Neuroscience* **16**, 2112–2126 (1996).

[6] Seung, H. S. How the brain keeps the eyes still. *Proceedings of the National Academy of Sciences* **93**, 13339–13344 (1996).

[7] Sompolinsky, H., Crisanti, A. & Sommers, H. J. Chaos in random neural networks. *Physical Review Letters* **61**, 259–262 (1988).

[8] van Vreeswijk, C. & Sompolinsky, H. Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science* **274**, 1724–1726 (1996).

[9] Battista, A. & Monasson, R. Capacity-resolution trade-off in the optimal learning of multiple low-dimensional manifolds by attractor neural networks. *Physical Review Letters* **124**, 048302 (2020).

[10] Mante, V., Sussillo, D., Shenoy, K. V. & Newsome, W. T. Context-dependent computation by recurrent dynamics in prefrontal cortex. *Nature* **503**, 78–84 (2013).

[11] Bai, S., Kolter, J. Z. & Koltun, V. Deep equilibrium models. *Advances in neural information processing systems* **32** (2019).

[12] Yang, G. R., Joglekar, M. R., Song, H. F., Newsome, W. T. & Wang, X.-J. Task representations in neural networks trained to perform many cognitive tasks. *Nature Neuroscience* **22**, 297–306 (2019).

[13] Mastrogiuseppe, F. & Ostojic, S. Linking connectivity, dynamics, and computations in low-rank recurrent neural networks. *Neuron* **99**, 609–623 (2018).

[14] Gao, P. *et al.* A theory of multineuronal dimensionality, dynamics and measurement. *BioRxiv* (2017).

[15] Clark, D. G., Abbott, L. & Litwin-Kumar, A. Dimension of activity in random neural networks. *Physical Review Letters* **131**, 118401 (2023).

[16] Dubreuil, A., Valente, A., Beiran, M., Mastrogiuseppe, F. & Ostojic, S. The role of population structure in computations through neural dynamics. *Nature neuroscience* **25**, 783–794 (2022).

10

[17] Song, H. F., Yang, G. R. & Wang, X.-J. Training excitatory-inhibitory recurrent neural networks for cognitive tasks: a simple and flexible framework. *PLoS computational biology* **12**, e1004792 (2016).

[18] Williams, A. H., Kunz, E., Kornblith, S. & Linderman, S. Generalized shape metrics on neural representations. *Advances in Neural Information Processing Systems* **34**, 4738–4750 (2021).

[19] Ostrow, M., Eisen, A., Kozachkov, L. & Fiete, I. Beyond geometry: Comparing the temporal structure of computation in neural circuits with dynamical similarity analysis. *Advances in Neural Information Processing Systems* **36** (2024).

[20] Driscoll, L. N., Shenoy, K. & Sussillo, D. Flexible multitask computation in recurrent networks utilizes shared dynamical motifs. *Nature Neuroscience* 1–15 (2024).

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

[22] Cubuk, E. D., Zoph, B., Shlens, J. & Le, Q. V. Randaugment: Practical data augmentation with no separate search. *arXiv preprint arXiv:1909.13719* **2**, 7 (2019).

[23] Majaj, N. J., Hong, H., Solomon, E. A. & DiCarlo, J. J. Simple learned weighted sums of inferior temporal neuronal firing rates accurately predict human core object recognition performance. *Journal of Neuroscience* **35**, 13402–13418 (2015).

[24] Schrimpf, M. *et al.* Brain-score: Which artificial neural network for object recognition is most brain-like? *BioRxiv* (2018).

11