Appendices

A   The Training Protocol
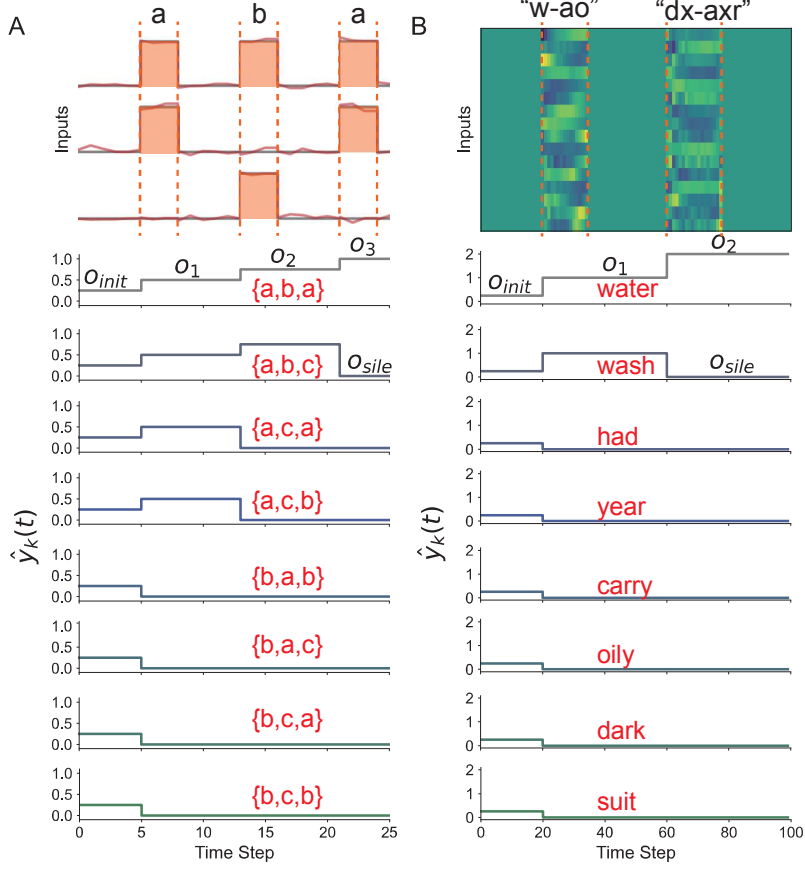
Figure S5: Illustration of the augmentation of temporal sequences and the setting of ramping target functions. (A) An example temporal sequence, $a - b - a$, in the synthetic task with $K = 8$, which is used to train the model shown in Fig.3C of maintext. The upper panel is augmented input stimuli by inserting duration and adding Gaussian noise, the lower panel is ramping target functions of readout neurons. (B) An example sequence, "water", in the key-word spotting task with $K = 8$ (used in Tab.1 of maintext), which includes the follwing words: "water, wash, had, year, carry, oily, dark, suit". The upper panel is augmented MFCC feature vector of "water", the lower is the corresponding target functions of read-out neurons.

To train the model, we first augment training data and construct their target functions (see illustration in Fig.2A of maintext), after that we apply backpropagation-through-time (BPTT) algorithm to optimize the model parameters $\{\mathbf{W}^{in}, \mathbf{W}^{rec}, \mathbf{W}^{out}, \mathbf{I}^b\}$. Denote $\mathbf{y}(t)$ the actual model output sequence and $\hat{\mathbf{y}}(t)$ the corresponding target function. The loss function is given by

$$L = \frac{1}{T_b B} \sum_{b=1}^{B} \sum_{k=1}^{K} \sum_{t=1}^{T_b} \left[y_k^b(t) - \hat{y}_k^b(t) \right]^2, \tag{3}$$

where $b = 1, \ldots, B$ denote the index of temporal sequences in the training data, T_b the length of the bth temporal sequence, and $k = 1, \ldots, K$ the index of the output neuron. We adopt the ADAM optimizer, with the learning rate varying in the range of $lr = [1e^{-4}, 1e^{-2}]$, and the decay rates of the first and second moments are 0.9 and 0.999.

Inspired by the ramping activity of decision making neurons observed in the experiment [1–3], we set the target function of a read-out neuron increases with the unfold of the items in the temporal

13

sequence this neuron encodes. Consider the temporal sequence of the first class, denoted as $\mathbf{S}^1$, is presented to the network. The target functions of read-out neurons are constructed as below.

(1) The task of discriminating four synthetic temporal sequences ($K = 4$), Fig.2 in the main text.

In this task, $\mathbf{S}^1$ consists of two consecutive items $\{S_1^1, S_2^1\}$ with the duration ΔT, as shown in Fig.2A-B in the main text. Before the temporal sequence is presented, all read-out neurons are at the spontaneous state, whose target functions are set to be $\hat{y}_k = O_{init}$, for $k = 1, ..., K$. When the first item S_1^1 appears, both the target functions of neuron 1 and 2 increase to $\hat{y}_{1,2} = O_1$, and the target functions of other neurons decay to the silent state, $\hat{y}_{3,4} = O_{sile}$. This lasts for the duration ΔT, until the second item appears. When S_2^1 appears, the target function of neuron 1 further increases to $\hat{y}_1 = O_2$, while the target function of neuron 2 decreases to O_{sile}, and the target functions of other neurons keep the same, i.e., $\hat{y}_{2,3,4} = O_{sile}$. In this task, the parameters used are $O_{init} = 0.25$, $O_1 = 0.5$, $O_2 = 1$, $O_{sile} = 0$.

For other tasks of discriminating four temporal sequences, the target functions are designed similarly.

(2) The task of discriminating eight synthetic temporal sequences ($K = 8$), Fig.3C in the main text

In this task, $\mathbf{S}^1$ consists of three consecutive items $\{S_1^1, S_2^1, S_3^1\}$ with the duration ΔT, as shown in Fig.3C in the main text. The design of target functions is illustrated in Fig.S1A, similar to the task with $K = 4$. Before the sequence is presented, all target functions of all read-out neurons are set to be $\hat{y}_k = O_{init}$ for $k = 1, ..., 8$. When the first item S_1^1 appears, the target functions of neurons $1 - 4$ increase to be $\hat{y}_k = O_1$, for $k = 1, 2, 3, 4$, since the four neurons share the same first item; while the target functions of other neurons decay to the silent state, i.e., $\hat{y}_k = O_{sile}$ for $k = 5, 6, 7, 8$. When S_2^1 appears, the target functions of neurons $1 - 2$ keep increasing to $\hat{y}_k = O_2$ for $k = 1, 2$; while the target functions of neurons $3 - 4$ decay to silent, and the target functions of other neurons keep silent, i.e., $\hat{y}_k = O_{sile}$, for $k = 3, ..., 8$. When S_3^1 appears, only the target function of neuron 1 keep increasing to $\hat{y}_1 = O_3$, while all other neurons are in the silent state, i.e., $\hat{y}_k = O_{sile}$, for $k = 2, ..., 8$. The parameters used are $O_{init} = 0.25$, $O_1 = 0.5$, $O_2 = 0.75$, $O_3 = 1$, and $O_{sile} = 0$.

For other tasks of discriminating eight temporal sequences, the target functions are designed similarly.

(3) The key-word spotting task of discriminating eight or twelve temporal sequences ($K = 8, 12$), Table 1 in the main text.

In this task, $\mathbf{S}^1$ denotes the temporal sequence of the key word 'water', which is chunked into two different items by syllables, i.e., 'w-ao' and 'dx-axr'. The training data is augmented according to the learning protocol described in the main text, as illustrated in Fig.S1B (upper panel). Before the sequence is presented, all read-out neurons are at the spontaneous state, i.e., $\hat{y}_k = O_{init}$, for $k = 1, ..., K$, with $K = 8, 12$. When the first item ('w-ao') appears, the target functions of neurons $1 - 2$ increase to O_1, while the target functions of other neurons decay to O_{sile}. When the second item appears, only the target function of neuron 1 keep increasing to O_2, while the target functions of other neurons are, $\hat{y}_k = O_{sile}$ for $k = 2, ..., K$. The design of target functions for $\mathbf{S}^1$ in 8-classes task is illustrated in Fig.S1B (lower panel). The parameters used are $O_{init} = 0.25$, $O_1 = 1$, $O_2 = 2$, $O_{sile} = 0$.

B The Visualization Method

B.1 Principle Component Analysis (PCA) of neural activities in the RNN (Sec.2.4.1 in the Main text)

We apply PCA to find out the low dimensional state space which captures the major variance of neural activities in the RNN. Denote $\mathbf{r}^k(t)$ the activity vector of the RNN in response to the kth temporal sequence at moment t, for $k = 1, ..., K$, whose element is given by $r_i^k(t) = r_i(t)$, for $i = 1, ..., N$, with N the number of neurons in the RNN. Combining the activity of the RNN at all moments, we obtain a martix $\mathbf{R}^k = \{\mathbf{r}^k(t)\}$, whose dimension is $N \times T$, with T the length of kth temporal sequence. Define the covariance matrix $\mathbf{X} = \sum_{i=1}^{N_k} \sum_{k=1}^{K} (\mathbf{R}^k - <\mathbf{R}>)(\mathbf{R}^k - <\mathbf{R}>)^T$, where $<\mathbf{R}>$ is obtained by averaging activity vectors over time and the total testing trials, $K \times N_k$, with N_k the number of trails in each class. We apply PCA to the matrix $\mathbf{X}$ and obtain a set of eigenvectors with decreasing eigenvalues. The eigenvalue of each PC reflects the variability of RNN response over time and trials along the corresponding eigenvector. As shown in Fig.2D in the main text, the first

14

3 PCs account for nearly 90% of the variance of RNN activities. On the basis of this analysis, we project the RNN activities onto the first 3 PCs to visualize the evolution of the RNN state over time for different temporal sequences, as shown in Fig.2E in the main text.

B.2 Stability analysis of tree-structured attractors (Sec.2.4.2 in the main text)

To uncover the structure of the state space of the learned RNN, we firstly find fixed points of the RNN dynamics, and then analyze their stability.

(1) Finding fixed points in the RNN dynamics

Identifying fixed points or slow points in the dynamics of the RNN helps us to understand the dynamical properties of our model. Specifically, a fixed point is a state that remains unchanged over time to a constant input, while a slow point is a state that changes very slow over time to a constant input. The dynamics of the RNN is given by $\tau d\mathbf{x}(t)/dt = -\mathbf{x}(t) + \mathbf{W}^{rec}\mathbf{r}(t) + \mathbf{W}^{in}\mathbf{I}(t) + \mathbf{I}^b$, which can be approximately solved using the first order Euler method with the simulation time step $\Delta t = 1$, and the neural activity $\mathbf{r}(t)$ at time $t+1$ can be approximately written as $\mathbf{r}(t+1) = F(\mathbf{r}(t), \mathbf{I}(t+1)) = \tanh\left((1 - \Delta t/\tau) * \arctan\left(\mathbf{r}(t)\right) + \Delta t/\tau(\mathbf{W}^{rec}\mathbf{r}(t) + \mathbf{W}^{in}\mathbf{I}(t) + \mathbf{I}^b)\right)$, where $F(\cdot)$ is the state update function. Denote $\mathbf{r}^*$ a fixed point in the state space of the RNN, which satisfies the equation, $\mathbf{r}^* = F(\mathbf{r}^*, \mathbf{I})$ for a particular input $\mathbf{I}$. To get a better understanding of the network dynamics, we are also interested in slow points, which approximately satisfy $\mathbf{r}^* \approx F(\mathbf{r}^*, \mathbf{I})$.

To identify these fixed or slow points, we follow the approach developed by Sussillo and Barak [4] by solving an optimization problem. We focus only on the fixed and slow points when the network receives no external input (i.e., $\mathbf{I} = \mathbf{0}$). We define q as an variable describing the speed at which the network state evolves over time, which is given by,

$$q = \frac{1}{2}||F(\mathbf{r}, \mathbf{I} = \mathbf{0}) - \mathbf{r}||^2. \tag{4}$$

The smaller q is, the slower the network dynamics evolves. To optimize q over the RNN state $\mathbf{r}$, we use BPTT and initialize from many conditions sampled from the distribution of network states during training. The neural states satisfying $q < 10^{-7}$ are considered to be fixed points. For the synthetic task as shown in Fig. 2 in the main text, we identify 7 fixed points in the RNN dynamics.

(2) Stability analysis of fixed points

For each fixed point $\mathbf{r}^*$, we analyze its stability using the perturbation method [5]. Specifically, we apply perturbations to a fixed point $\mathbf{r}^*$, denoted as $\Delta\mathbf{r}(t)$ and $\Delta\mathbf{I}(t)$, respectively. The network state at $t+1$ can be updated with the first-order Taylor expansion, which is given by,

$$\mathbf{r}(t+1) = F(\mathbf{r}^* + \Delta\mathbf{r}(t), \mathbf{I}^* + \Delta\mathbf{I}(t)) \approx F(\mathbf{r}^*, \mathbf{I}^*) + \mathbf{J}^{rec}\Delta\mathbf{r}(t) + \mathbf{J}^{inp}\Delta\mathbf{I}(t), \tag{5}$$

where $\{\mathbf{J}^{rec}, \mathbf{J}^{inp}\}$ are Jacobian matrices given by $J^{rec}_{ij}(\mathbf{r}^*, \mathbf{I}^*) = \frac{\partial F(\mathbf{r}, \mathbf{I})_i}{\partial r_j}|_{(\mathbf{r}^*, \mathbf{I}^*)}$ and $J^{inp}_{ij}(\mathbf{r}^*, \mathbf{I}^*) = \frac{\partial F(\mathbf{r}, \mathbf{I})_i}{\partial I_j}|_{(\mathbf{r}^*, \mathbf{I}^*)}$.

The eigendecomposition of the recurrent Jocabian matrix $\mathbf{J}^{rec}$ can be expressed as $\mathbf{J}^{rec} = \mathbf{R}\mathbf{\Lambda}\mathbf{L}$, where $\mathbf{R}$ and $\mathbf{L}$ are matrices that contain the right and left eigenvectors of $\mathbf{J}^{rec}$, respectively, and $\mathbf{\Lambda}$ is a diagonal matrix containing complex-valued eigenvalues. The value of the real part of an eigenvalue determines the stability of the corresponding eigenmode, with those smaller than 1 representing stable modes and those larger than 1 representing unstable modes. As shown in Fig.S6, the maximums of real parts of all eigenvalues of each fixed point are all smaller than 1, indicating that all these fixed points are stable attractors of the RNN.

B.3 Analyzing the transition dynamics of attractor states (Sec.2.4.3 in the main text)

We take the transitions from stable points $S_1^{1,2}$ to S_1^2 and S_2^2 as examples to illustrate the transition dynamics of the RNN. We combine the network activity trajectories in response to the kth class temporal sequence, for $k = 1, ..., K$, denoted as $\hat{\mathbf{R}}^k = \{\mathbf{r}^k(t_1 : t_2)\}$, to construct the matrix $\hat{\mathbf{X}}$ as in Sec.B.1 of SI, where t_1 and t_2 denote the moments of the appearance of item $S_1^{1,2}$ and the ending of the sequence, respectively. Define the covariance matrix $\hat{\mathbf{X}} = \sum_{i=1}^{N_k} \sum_{k=1}^{K} (\hat{\mathbf{R}}^k - <\hat{\mathbf{R}}>)(\hat{\mathbf{R}}^k - <\hat{\mathbf{R}}>)^T$, where $<\hat{\mathbf{R}}>$ is the average of network activities over time from t_1 to t_2 and over $K \times N_k$

15
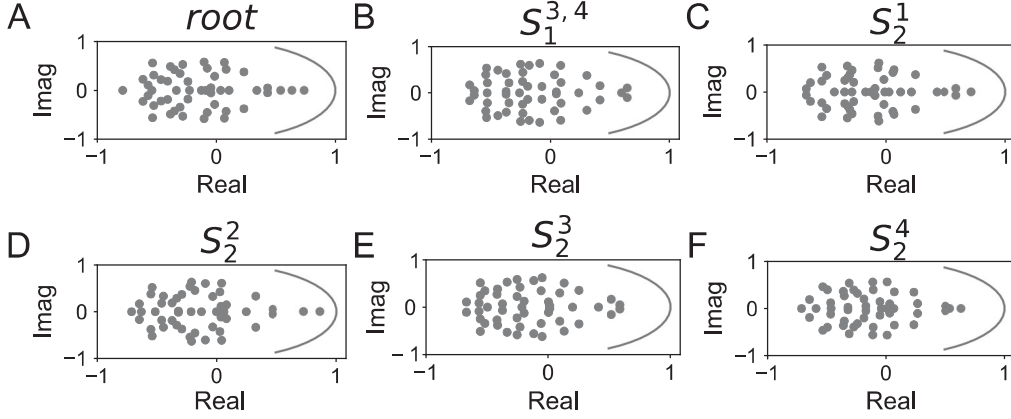
Figure S6: The distributions of complex eigenvalues for six attractor states of the RNN used in the synthetic task. The result for the other fixed point is shown in Fig.2D.

trails, with N_k the number of trail in each class. We perform PCA on the matrix $\hat{\mathbf{X}}$ and obtain a set of eigenvectors with decreasing eigenvalues. The first two eigenvectors (denoted as PC_1 and PC_2) account for the major variance of neural responses.

We further analyze the right eigenvectors of the Jocabian matrix $\mathbf{J}^{rec}$ around the stable fixed point $S_1^{1,2}$. The larger the eigenvalue is, the more unstable the network activity state is along the eigenvector under noise perturbation. We project the right eigenvectors onto the hyperplane spanned by PC_1 and PC_2, with their lengths determined by the corresponding eigenvalues. We observe that the two most unstable directions among all eigenvectors point to nodes S_2^1 and S_2^2, as shown in Fig.2E-F. This indicates that it is much easier for the network state transferring from node $S_1^{1,2}$ to nodes S_2^1 and S_2^2 than to other states.

C   Transfer Learning Tasks

C.1   The construction of phoneme Sequences (Sec.3.1 in the main text)

The three primitive items, 'pcl', 'tcl' and 'pau' used in Fig.3 in the main text, are randomly selected from the TIMIT dataset [6]. We transform the raw waves of three phoneme items into Mel-frequency cepstral coefficients (MFCC) with a window size of 25ms and stride of 10ms, roughly mimicking the information processing in the cochlea in the brain. The obtained MFCC feature vectors have 16 channels, and the MFCC vectors are further normalized by subtracting their mean and dividing by their standard deviation, along the time dimension, which are used as inputs to the RNN. Using the four-class classification task as an example, as shown in Fig.2B in the main text, we construct four temporal sequences, which are "pcl-tcl", "pcl-pau", "tcl-pcl", and "tcl-pau". We augment these sequence by inserting varied intervals between primitive items and adding Gaussian noises. We then train the model using ramping target functions. In the task, the total number of augmented data is 10000, which are randomly divided into 8000 training and 2000 testing examples.

C.2   The performance measurement (Fig.3A in the main text)

In transfer learning, the model learns to allocate phoneme contents to the pre-trained tree-like attractor template. To evaluate the effect of reusing the ordinal structure, we calculate the discrepancy between the learned attractor structure and the template, measuring by the Euclidean distance between them. Each item and its followed duration can be seen as a temporal chunk. If the model learns to reuse the pre-trained attractor dynamics, the last state of each temporal chunk should exactly stay at the corresponding attractor state. To test this, as an example, for the sequence $\mathbf{S}^1 = \{S_1^1, S_2^1\}$ in the four-class discrimination task, the learned state vector of S_1^1 item is obtained by averaging the state vectors over the last five time steps before the item S_2^1 is presented. Similarly, the learned state vector of S_2^1 item is obtained by averaging the state vectors over the last five time steps before the end of the sequence. The learned state vectors of other sequence items are computed similarly. After obtaining

16
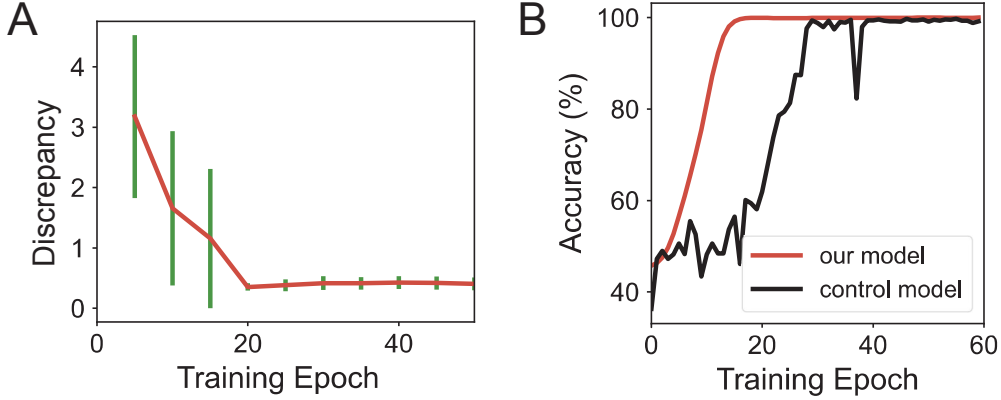
Figure S7: (A) The discrepancy between the learned dynamics and the template dynamics in the task of partially reusing the ordinal template. (B) In the transfer learning scenario, our model exhibits significantly accelerated learning compared to the control model using the cross-entropy loss.

these learned state vectors, we calculate the Euclidean distance between them and the corresponding tree-structured attractor state vectors in the template. The final discrepancy is computed by averaging the results of seven attractor states. As shown in Fig.3A and Fig.SS7, as the training progresses, the discrepancy between the learned attractor states and the template attractor states decreases gradually to zero, indicating that the model learns to allocate the input contents to the ordinal template stored in the RNN.

In transfer learning setting, We also compare our model with tree attractor structure to a pretrained recurrent neural network without using tree structrue. As shown in Fig. S7B, we train a control model having the same network structure as our model, except that the cross-entropy loss, rather than the ramping target function, is used. By this, the control model learns the sequence discrimination task but no longer acquires the tree-like attractors. We then apply the control model to the same transfer learning task (following the same protocol of freezing the connections in the recurrent and read-out layers). The results demonstrate that our model exhibits significantly accelerated learning process compared to the control model. This experiment demonstrates that the tree-structured attractor dynamics (the schema) is indeed indispensable for transfer learning. Together with our other analyses in Fig.3A and Fig. S7A), which verify that the attractor template is indeed reused after transfer learning, we come to the conclusion that the tree-structured attractor dynamics plays a critical role in transfer learning.

As shown in Fig. S8, we show that our model can learn to process sequences of depth 3 by combining two primitive templates of depth 2. In this experiment, for simplicity, we consider two independent recurrenct networks and there are no recurrent connections between them. Firstly, two RNNs learn and form same tree attractor structures of depth 2 through training on a four-class synthetical task. Then, following the transfer learning procedure as described in Sec.3.1, we freeze the recurrent connections in two RNNs and only optimize the feedforwad and readout connections, as shown in Fig. S8A. New eight-class phoneme task are constructed, written as "pcl-tcl-pcl", "pcl-tcl-pau", "pcl-pau-pcl", "pcl-pau-tcl", "tcl-pcl-tcl", "tcl-pcl-pau", "tcl-pau-tcl" and "tcl-pau-pcl". Our findings are as follows: 1) the pretrained attractor structures significantly accellerate the learning speed in a new sequence task, as depcited in Fig. S8C; 2) the network indeed combines and reuses two primitive tree attractor dynamics to facilitate the learning speed, as shown in Fig. S8D. These results demonstrate that our model has the capability to adapt to processing temporal sequences of varying lengths.

17
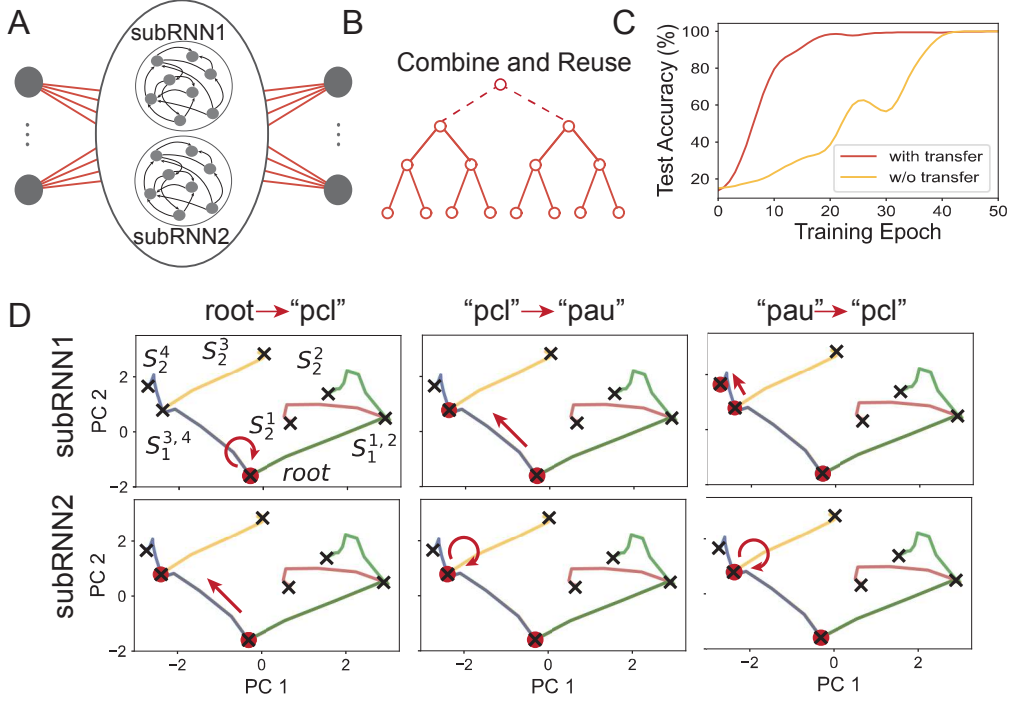
Figure S8: An example demonstrating primitive attractor template can be combined and reused to represent longer sequence. A. Network structure: two parts of a RNN (subRNN1 and subRNN2) encoding two depth- 2 tree-structured attractors. B. Schematic diagram: illustrating the combination and reuse of two tree- structured primitive templates. C. Significant acceleration in learning compared to the case without transfer learning. D. Attractor transition dynamics in the two subRNNs when a phoneme sequence "pcl"-"pau"-"pcl" is presented. The transition dynamics of subRNN1(the upper pannel) and subRNN2(the lower panel) are shown. In root-"pcl", when the network receives the 1st sequence item "pcl", subRNN1's state remain at the root node (red circles), while that of subRNN2 transits from the root node to $S_1^{3,4}$(shown in red arrow). In 1 "pcl"-"pau", when given the 2nd item "pau," subRNN1's state moves from the root node to $S_1^{3,4}$ while that 1 of subRNN2 stay at state $S_1^{3,4}$. In "pau"-"pcl", when given the 3rd item "pcl," subRNN1's state further 1 transits from $S_1^{3,4}$ to S_2^4. Thus, in transfer learning, the network combines two tree-structured attractor 12 templates to represent a longer sequence.

D Key-word spotting tasks

D.1 Dataset Construction for Key-Word Spotting Task (Sec.3.2 in the maintext)

To construct the training and testing datasets for the key-word spotting task in Sec.3.2 in the main text, we randomly select twelve key words from the TIMIT dataset based on the word labeling. The TIMIT dataset comprises 6300 English sentences read by 630 speakers. The twelve key words are illustrated in Table.S1, which can be chunked into two different primitive items according to syllables. For example, 'wash' is chunked into 'w-ao' and 'sh'.

In the four-class key-word spotting task, we use the four words, "water, wash, had, year". In the eight-class task, we add another four words, "carry, oily, dark, suit". In the twelve-class task, we use all the selected words. In each task, the total number of temporal sequences for each word is nearly 310, which are randomly divided into 10 training and 300 testing examples. The raw wave of each temporal sequence is transformed into MFCC features using a time window size of 25ms and stride of 10ms. The obtrained MFCC vectors of each word is further normalized by subtracting their means and divided by their standard deviations along the time dimension.

18

In the training data, following our training protocol described in the main text, we chunk the MFCC vectors of a sequence into two primitive items according to the syllables of the key word, and then augment the data by inserting varied intervals between primitive items and adding Gaussian white noises. We then train the augmented temporal sequences of key words using ramping target functions, more details see Sec. A in SI. In the testing data, we present the MFCC vectors of each sequence without augmenting, rather we stretch or compress the MFCC sequences. The warping effect is specified by a variable F, with $F = 1$ denoting no warping, $F > 1$ stretching and $F < 1$ compressing.

Table S2: 12 words and their chunks, each chunk consists of phonemes or syllables.

Words	Chunks
water	['w-ao','dx-axr']
wash	['w-ao','sh']
had	['hv','eh-dcl']
year	['y-ih','axr']
carry	['kcl-k', 'eh-r-iy']
oily	['oy','l-iy']
dark	['dcl-d','aa-r-kcl-k']
suit	['s','ux-tcl']
greasy	['gcl','g-r-iy-s-iy']
rag	['r','ae-gcl-g']
ask	['ae','s-kcl']
like	['l','ay-kcl']

During training, both our model and the control model are trained on data with $F = 1$. During testing, we apply a warping factor F to compress or stretch the original MFCC feature vectors using linear interpolation while retaining their spatial structure (particularly, we use $numpy.interp$ function to realize temporal warping, which works by finding the two nearest data points surrounding each value in the target data, constructing a straight line between these two points, and calculating the stretched or compressed target data along this line). After temporal warping, we scale the total strength of MFCC feature vectors by dividing a scaling factor $\sqrt{F}$, mimicking the divisive normalization process in the neural system [7]. Finally, the scaled versions of testing words are used to evaluate the model's classification robustness. Notably, scaling the strength of a MFCC feature vector slightly improves models' performance during testing, but in any case, our model outperforms other models significantly.

D.2 Ablation study (Sec.4 in the main text)

In our training protocol, the introduction of varied temporal intervals between phonemes or syllables has the potential to generate spectral structures that closely resemble those present in the test examples. Consequently, the apparent efficacy of our model may be attributed to the exposure to training data that closely mirrors the test data, rather than leveraging tree attractor structures. To alleviate this concern and demonstrate the importance of the training protocol, we compare our model with several control models in the key-word spotting task. It demonstrates that the tree-structured attractor dynamics (the schema) contributes to the robustness of our model to warping sequences.

In our model, each training sequence is augmented to be 1.5 times of its original length by inserting varied intervals between phoneme or syllable chunks, and the model is trained using the ramping target function. In control model 2, training examples adopt the same augmentation protocol as in our model, but the model is trained using cross-entropy loss. In the control model 1, training examples are just warped sequences with the warping value chosen randomly in the range of $(1, 1.5)$. This makes the training and test data even more similar in the control model 1 than that in our model by data augmentation (in both models, the lengths of sequences are kept to be in the same range). We used the cross-entropy loss to train the control model 2. Thus, both control model 1 and 2 would not acquire proper tree-structured attractors.

We find that when the temporal warping value falls in the range of [1, 1.5], i.e., the range of training sequences, both models exhibit similar good test accuracies. However, when the warping value is outside of the training range, our model outperforms the control model significantly, as shown in Fig. S9.

19
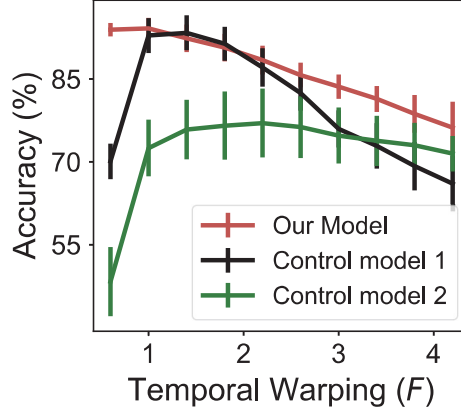
Figure S9: Performance comparison of our model with two control models. In our model, the training data is augmented by introducing varied intervals between "chunks" of a given word. Each sequence is augmented to be 1.5 times of its original length. Following the same augmentation protocol as in our model, Control model 2 (green line) is trained using the cross-entropy loss. In Control model 1(black line), the cross-entropy loss is used, and the training data are warping sequences with the warping value chosen randomly in the range of (1, 1.5). The test accuracy of each model is calculated by averaging over 8 trials at each warping value F.

This supports our statement that the tree-structured attractor dynamics (the schema) contributes to the robustness of our model to warping sequences. The underlying mechanism can be attributed to two factors: 1) attractors ensure the stable responses of our model to stretching/compressing inputs; 2) the tree-structured attractor dynamics further enhances averaging out noises in inputs over time, behaving like an evidence-accumulation process (see Fig.4C)

We also apply our training protocol to other recurrent networks used in machine learning, including Long Short-Term Memory (LSTM) [9] and Gated Recurrent Unit (GRU)[10]. Using the datasets in the key-words spotting task, both GRU and LSTM are trained by using either our learning protocol or the conventional learning method (which involves data augmentation and the cross-entropy loss function). The results are compared in Table.1 in the main text.

E   Experiments with different training Settings and model architectures

We find that the tree-structured attractors also emerge when training using fixed interval values for the clean synthetical data, as shown in Fig. S10A. However, for the noisy spoken words, we do need an amount of variations to achieve good performances.We also examine our method on GRU network to test its university across model architectures. As shown in Fig. S10B, GRU network can also learn a tree-structured attractors successfully.

F   Model Parameters used in this study

All models are trained using PyTorch 2.0 [8].

(1) Parameters in Fig.2 in the main text:

The parameters of the RNN are: $M = 3$, $N = 50$, $K = 4$, $\tau = 2$. The parameters for data augmentation are: $\Delta t = 3$, $\Delta T_{max} = 30$, the noise strength inserted between neighboring items is $\sigma = 0.01$. The recurrent connection weights $\mathbf{W}^{rec}$ are initialized by sampling from an independent Gaussian distribution with zero mean and SD equal to $g/\sqrt{N}$, with g representing the 'gain' of the network, $g = 0.2$. For the design of target functions, see Sec. A. The batch size is $B = 16$, and the total number of training epochs is 100.

(2) Parameters in Fig.3 in the main text:

20
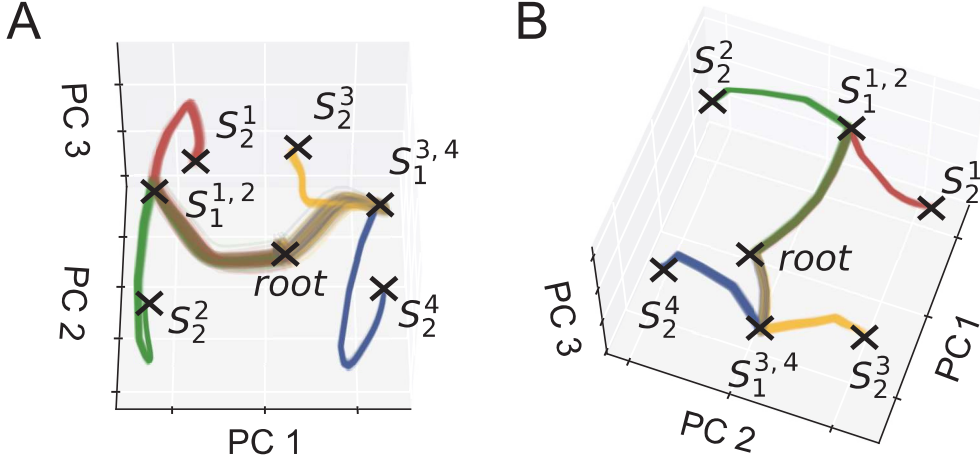
Figure S10: (A) In a 4-class synthetic sequence task, our model successfully learns a tree-like attractor structure from sequence inputs with a fixed interval of 10 time steps. (B) A GRU model trained using our method also learns a tree-like attractor structure on the 4-class synthetic task.

The parameters of the RNN are: $M = 16$, $N = 50$, $K = 4$ in Fig.4A-B and $K = 8$ in Fig.4C-D, $\tau = 2$. The parameters for data augmentation are: $\Delta T_{max} = 30$, the noise strength inserted between neighboring items is $\sigma = 0.01$. For details of the construction of phoneme temporal sequences, see Sec. C in SI. $\mathbf{W}^{rec}$ is initialized by sampling from an independent Gaussian distribution with zero mean and SD equal to $g/\sqrt{N}$, $g = 0.2$. For the design of target functions, see Sec. A in SI. The batch size is $B = 16$, and the total number of training epochs is 100.

(3) Parameters in Fig.4 in the main text:

The parameters of the RNN are: $M = 16$, $N = 100$, $K = 4$, $\tau = 2$. The parameters for data augmentation are: $\Delta T_{max} = 60$, the noise strength inserted between neighboring items is $\sigma = 0.01$. The recurrent connection weights $\mathbf{W}^{rec}$ are initialized by sampling from an independent Gaussian distribution with zero mean and SD equal to $g/\sqrt{N}$, with g representing the 'gain' of the network, $g = 0.2$. For the design of target functions, see Sec. A. The batch size is $B = 16$, and the total number of training epochs is 200.

(4) Parameters in Table 1 in the main text:

The parameters of LSTM, GRU and RNN are: $M = 16$, $N = 100$, $K = 4$. The time constant of neurons in the RNN is $\tau = 2$. The parameters for data augmentation are: $\Delta T_{max} = 60$, the noise strength inserted between neighboring items is $\sigma = 0.01$. The recurrent connection weights $\mathbf{W}^{rec}$ are initialized by sampling from an independent Gaussian distribution with zero mean and SD equal to $g/\sqrt{N}$, with g representing the 'gain' of the network, $g = 0.2$. The connections of LSTM and GRU adopt Glorot are initialized by an uniform distribution. For the design of target functions, see Sec. A. The batch size is $B = 16$, and the total number of training epochs is 200.

References

[1] Wang, X. J. (2008). Decision making in recurrent neuronal circuits. Neuron, 60(2), 215-234.

[2] Okazawa, G., & Kiani, R. (2023). Neural Mechanisms that Make Perceptual Decisions Flexible. Annual Review of Physiology, 85, 191-215.

[3] Shadlen, M. N., & Newsome, W. T. (1996). Motion perception: seeing and deciding. Proceedings of the national academy of sciences, 93(2), 628-633.

[4] Sussillo, D., & Barak, O. (2013). Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. Neural computation, 25(3), 626-649.

21

[5] Maheswaranathan, N., Williams, A., Golub, M., Ganguli, S., & Sussillo, D. (2019). Reverse engineering recurrent networks for sentiment classification reveals line attractor dynamics. Advances in neural information processing systems, 32.

[6] Garofolo, J. S., Lamel, L. F., Fisher, W. M., Fiscus, J. G., Pallett, D. S. & Dahlgren, N. L. (1993). DARPA TIMIT Acoustic Phonetic Continuous Speech Corpus CDROM NIST

[7] Carandini, M., & Heeger, D. J. (2012). Normalization as a canonical neural computation. Nature Reviews Neuroscience, 13(1), 51-62.

[8] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 32.

[9] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[10] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

22