

Hasani et al. (2021) — Liquid Time-Constant Networks

Full Citation: Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, Radu Grosu. “Liquid Time-Constant Networks.” The Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI-21), pp. 7657–7666, 2021.

Paper Focus: This paper introduces an abstract, implementation-friendly formulation of LTC networks. It provides: (1) the core LTC ODE in a clean form, (2) a custom fused ODE solver, (3) a BPTT training algorithm, (4) bounds on time-constant and hidden state, (5) a trajectory-length-based expressivity analysis, and (6) extensive experimental results. This is the key paper for **implementing** an LTC network.

1. Context: From Neural ODEs to LTCs

The paper begins by noting that the state of a neural ODE, $\mathbf{x}(t) \in \mathbb{R}^D$, is defined by the solution of:

$$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), \mathbf{I}(t), t, \theta) \quad (\text{Neural ODE})$$

with a neural network f parametrized by θ . A more stable variant — the continuous-time RNN (CT-RNN) — adds a damping term:

$$\frac{d\mathbf{x}(t)}{dt} = -\frac{\mathbf{x}(t)}{\tau} + f(\mathbf{x}(t), \mathbf{I}(t), t, \theta) \quad (\text{CT-RNN})$$

The term $-\mathbf{x}(t)/\tau$ assists the autonomous system to reach an equilibrium state with time-constant τ . Here $\mathbf{x}(t)$ is the hidden state, $\mathbf{I}(t)$ is the input, t is time, and f is parametrized by θ .

The paper proposes an **alternative formulation**: let the hidden state flow be declared by a system of linear ODEs of the form $d\mathbf{x}(t)/dt = -\mathbf{x}(t)/\tau + \mathbf{S}(t)$, and let $\mathbf{S}(t) \in \mathbb{R}^M$ represent the following nonlinearity:

$$\mathbf{S}(t) = f(\mathbf{x}(t), \mathbf{I}(t), t, \theta)(A - \mathbf{x}(t)) \quad (\text{S definition})$$

with parameters θ and A . By plugging \mathbf{S} into the hidden state equation, we get the core LTC equation.

2. Core LTC Hidden State ODE

Equation 1 — The LTC ODE (key)

$$\frac{d\mathbf{x}(t)}{dt} = -\left[\frac{1}{\tau} + f(\mathbf{x}(t), \mathbf{I}(t), t, \theta)\right] \mathbf{x}(t) + f(\mathbf{x}(t), \mathbf{I}(t), t, \theta)A \quad (1)$$

This is the central equation of the paper and the primary equation you will implement. It says the rate of change of the hidden state has two components:

1. **State-dependent decay:** $-[1/\tau + f(\cdot)]\mathbf{x}(t)$ — This pulls the state toward zero. The decay rate is not just $1/\tau$ (as in a CT-RNN) but is augmented by $f(\cdot)$, which depends on the current state and input. When f is large, the decay is faster; when f is small, the decay is slower.
2. **Nonlinear drive:** $f(\cdot)A$ — This pushes the state toward the bias vector A , scaled by the nonlinear function f . The vector A acts as a set of target values (analogous to synaptic reversal potentials in the biophysical model).

Parameters and variables:

Symbol	Dimensions	Description
$\mathbf{x}(t)$	$N \times 1$	Hidden state vector of N neurons at time t . Main state variable.
$\mathbf{I}(t)$	$M \times 1$	External input vector of dimension M at time t .
τ	$N \times 1$	Vector of intrinsic time-constants, one per neuron. All entries positive.

Symbol	Dimensions	Description
$f(\cdot)$	Outputs $N \times 1$	A neural network parametrized by θ . Maps state and input to activation values.
θ	varies	All learnable parameters of f . Includes weights, recurrent weights, biases.
A	$N \times 1$	Bias vector (reversal potential analog). Learnable. Determines target state.

Liquid Time-Constant property: A neural network f not only determines the derivative of the hidden state $\mathbf{x}(t)$, but also serves as an input-dependent varying time-constant:

$$\tau_{sys} = \frac{\tau}{1 + \tau \cdot f(\mathbf{x}(t), \mathbf{I}(t), t, \theta)} \quad (1a)$$

This property enables single elements of the hidden state to identify specialized dynamical systems for input features arriving at each time-point.

Biophysical justification (inline): The dynamics of non-spiking neurons' potential $v(t)$ can be written as a system of linear ODEs of the form (Lapicque, 1907; Koch and Segev, 1998):

$$dv/dt = -g_l v(t) + S(t) \quad (\text{bio-1})$$

where S is the sum of all synaptic inputs and g_l is a leakage conductance. All synaptic currents to the cell can be approximated in steady-state by the nonlinearity (Koch and Segev, 1998; Wicks et al., 1996):

$$S(t) = f(v(t), I(t))(A - v(t)) \quad (\text{bio-2})$$

where $f(\cdot)$ is a sigmoidal nonlinearity depending on the state of all presynaptic neurons $v(t)$ and external inputs $I(t)$. Plugging these two equations together yields an equation equivalent to Eq. 1.

Connection to Dynamic Causal Models (DCMs): Eq. 1 also resembles DCMs (Friston, Harrison, and Penny, 2003) with a bilinear dynamical system approximation (Penny, Ghahramani, and Friston, 2005). DCMs are formulated by a second-order approximation of $dx/dt = F(\mathbf{x}(t), \mathbf{I}(t), \theta)$ resulting in:

$$dx/dt = (A + \mathbf{I}(t)B)\mathbf{x}(t) + C\mathbf{I}(t) \quad (\text{DCM})$$

with $A = dF/dx$, $B = dF^2/(dx \cdot dI)$, $C = dF/dI$.

3. LTC Forward Pass by a Fused ODE Solver

The Problem of Stiffness

Solving Eq. 1 analytically is non-trivial due to the nonlinearity. The state at any time T can be computed by a numerical ODE solver that simulates from $x(0)$ to $x(T)$. However, LTCs' ODE realizes a system of **stiff equations** (Press et al., 2007). This requires an exponential number of discretization steps when simulated with a Runge-Kutta (RK) based integrator. Consequently, ODE solvers based on RK (such as Dormand-Prince, the default in `torchdiffeq`) are **not suitable** for LTCs.

Equation 2 — Fused Euler Discretization

The paper designs a new ODE solver that fuses the explicit and implicit Euler methods. The discretization numerically unrolls a given dynamical system of the form $dx/dt = f(x)$ by:

$$x(t_{i+1}) = x(t_i) + \Delta t \cdot f(x(t_i), x(t_{i+1})) \quad (2)$$

The key insight: replace only the $x(t_i)$ that occurs **linearly** in f by $x(t_{i+1})$. As a result, Eq. 2 can be solved for $x(t_{i+1})$ **symbolically** — no iterative Newton-Raphson steps are needed. This gives the stability of implicit Euler with the efficiency of explicit Euler.

Equation 3 — Fused Solver Closed-Form Update (key equation)

Applying the fused solver to the LTC representation and solving for $\mathbf{x}(t + \Delta t)$:

$$\mathbf{x}(t + \Delta t) = \frac{\mathbf{x}(t) + \Delta t \cdot f(\mathbf{x}(t), \mathbf{I}(t), t, \theta) \odot A}{1 + \Delta t \left(\frac{1}{\tau} + f(\mathbf{x}(t), \mathbf{I}(t), t, \theta) \right)} \quad (3)$$

This is the equation you implement in your forward pass. It computes one update state for an LTC network.

Critical implementation details:

- $f(\cdot)$ and **all divisions** are applied **element-wise** across the N neurons.
- \odot is the Hadamard (element-wise) product.
- This is applied L times (unfolding steps) per input time step for accuracy.

Typical choice for f : For a tanh nonlinearity:

$$f = \tanh(\gamma_r \mathbf{x} + \gamma \mathbf{I} + \mu) \quad (3a)$$

where $\gamma_r \in \mathbb{R}^{N \times N}$ are recurrent weights, $\gamma \in \mathbb{R}^{M \times N}$ are input weights, and $\mu \in \mathbb{R}^{N \times 1}$ are biases. The activation function can be arbitrary (tanh, sigmoid, ReLU, Hard-tanh).

4. Algorithm 1 — LTC Update by Fused ODE Solver

Parameters: $\theta = \{\tau^{(N \times 1)}, \gamma^{(M \times N)}, \gamma_r^{(N \times N)}, \mu^{(N \times 1)}, A^{(N \times 1)}\}$, plus L = number of unfolding steps, Δt = step size, N = number of neurons.

Inputs: M -dimensional input $\mathbf{I}(t)$ of length T , initial state $\mathbf{x}(0)$.

Output: Next LTC neural state $\mathbf{x}_{t+\Delta t}$.

Function FusedStep($\mathbf{x}(t)$, $\mathbf{I}(t)$, Delta_t, theta):

```

     $\mathbf{x}(t + \text{Delta\_t}) = [\mathbf{x}(t) + \text{Delta\_t} * f(\mathbf{x}(t), \mathbf{I}(t), t, \text{theta}) .* A] / [1 + \text{Delta\_t} * (1/\tau + f(\mathbf{x}(t), \mathbf{I}(t), t, \text{theta}))]$ 
    // f(*), and all divisions are applied element-wise
    // .* is the Hadamard product
    return  $\mathbf{x}(t + \text{Delta\_t})$ 

```

```

 $\mathbf{x}_{\{t + \text{Delta\_t}\}} = \mathbf{x}(t)$ 
for i = 1 ... L:
     $\mathbf{x}_{\{t + \text{Delta\_t}\}} = \text{FusedStep}(\mathbf{x}(t), \mathbf{I}(t), \text{Delta\_t}, \text{theta})$ 
return  $\mathbf{x}_{\{t + \text{Delta\_t}\}}$ 

```

Computational complexity: For an input sequence of length T : $O(L \times T)$, where L is the number of discretization steps. A dense LTC with N neurons and a dense LSTM with N cells have the same complexity.

5. Training LTC Networks by BPTT

Why BPTT over the Adjoint Method

Neural ODEs were originally trained using the adjoint sensitivity method for constant memory cost. However, the adjoint method **forgets the forward-time computational trajectories**, leading to numerical errors in the backward pass. This has been repeatedly noted by the community (Gholami et al., 2019; Zhuang et al., 2020).

Direct backpropagation through time (BPTT) trades memory for **accurate recovery** of the forward pass during reverse mode integration.

Algorithm 2 — Training LTC by BPTT

Inputs: Dataset of traces $[I(t), y(t)]$ of length T , RNN-cell = $f(I, x)$.

Parameters: Loss function $L(\theta)$, initial param θ_0 , learning rate α , output $w = W_{out}$, bias = b_{out} .

```

for i = 1 ... number of training steps:
  (I_b, y_b) = Sample training batch
  x := x_{t_0} ~ p(x_{t_0})
  for j = 1 ... T:
    x = f(I(t), x)
    y_hat(t) = W_out * x + b_out
    L_total = Sum_{j=1}^{T} L(y_j(t), y_hat_j(t))
    grad_L(theta) = dL_total / dtheta
  theta = theta - alpha*grad_L(theta)
return theta

```

One can substitute vanilla SGD with Adam (Kingma and Ba, 2014).

Complexity Comparison (Table 1)

Property	Vanilla BPTT	Adjoint Method
Time	$O(L \times T \times 2)$	$O((L_f + L_b) \times T)$
Memory	$O(L \times T)$	$O(1)$
Depth	$O(L)$	$O(L_b)$
FWD accuracy	High	High
BWD accuracy	High	Low

Where: L = number of discretization steps, $L_f = L$ during forward pass, $L_b = L$ during backward pass, T = length of sequence, Depth = computational graph depth.

Key takeaway: BPTT achieves a high degree of accuracy on **both** forward and backward integration trajectories, with similar computational complexity, at larger memory costs.

6. Bounds on τ and Neural State of LTCs

Theorem 1 — Time-Constant Bounds

Let x_i denote the state of a neuron i within an LTC, identified by Eq. 1, and let neuron i receive M incoming connections. Then, the time-constant of the neuron, τ_{sys_i} , is bounded to the following range:

$$\frac{\tau_i}{1 + \tau_i W_i} \leq \tau_{sys_i} \leq \tau_i \quad (4)$$

where $W_i = \sum_j |w_{ij}|$ is the sum of absolute incoming weights to neuron i .

Proof basis: Constructed based on the bounded, monotonically increasing sigmoidal nonlinearity for neural network f and its replacement in the LTC network dynamics. Since $0 \leq f \leq W_i$ (for sigmoid-like activations):

- When $f = 0$ (no input activation): $\tau_{sys} = \tau / (1 + \tau \cdot 0) = \tau$ (maximum, slowest).
- When $f = W_i$ (maximum activation): $\tau_{sys} = \tau / (1 + \tau W_i)$ (minimum, fastest).

Interpretation: A stable varying time-constant significantly enhances the expressivity of this form of time-continuous RNNs. The time-constant is always positive and finite.

Theorem 2 — Hidden State Bounds

Let x_i denote the state of a neuron i within an LTC, identified by Eq. 1, and let neuron i receive M incoming connections. Then, the hidden state of any neuron i , on a finite interval $Int \in [0, T]$, is bounded as follows:

$$\min(0, A_i^{min}) \leq x_i(t) \leq \max(0, A_i^{max}) \quad (5)$$

where A_i^{min} and A_i^{max} are the minimum and maximum values of the bias vector A for neuron i .

Proof basis: Constructed based on the sign of the LTC equation’s compartments and an approximation of the ODE model by an explicit Euler discretization.

Interpretation — State Stability: This guarantees that the outputs of LTCs **never explode** even if their inputs grow to infinity. This is a highly desirable property for robust deployment that standard neural ODEs do not possess.

7. Universal Approximation of LTCs

Theorem 3 — LTCs are Universal Approximators

Let $\mathbf{x} \in \mathbb{R}^n$, $S \subset \mathbb{R}^n$ and $\dot{\mathbf{x}} = F(\mathbf{x})$ be an autonomous ODE with $F : S \rightarrow \mathbb{R}^n$ a C^1 -mapping on S . Let D denote a compact subset of S and assume that the simulation of the system is bounded in the interval $I = [0, T]$. Then, for a positive ϵ , there exist an LTC network with N hidden units, n output units, and an output internal state $\mathbf{u}(t)$, described by Eq. 1, such that for any rollout $\{\mathbf{x}(t)|t \in I\}$ of the system with initial value $\mathbf{x}(0) \in D$, and a proper network initialization:

$$\max_{t \in I} |\mathbf{x}(t) - \mathbf{u}(t)| < \epsilon \quad (6)$$

Proof note: The proof defines an n -dimensional dynamical system and places it into a higher dimensional system. The second system is an LTC. The fundamental difference of the proof of LTC’s universality to that of CT-RNNs (Funahashi and Nakamura, 1993) lies in the distinction of the semantics of both systems where the LTC network contains a **nonlinear input-dependent term in its time-constant module** which makes parts of the proof non-trivial.

8. Measuring Expressivity by Trajectory Length

The paper extends the trajectory length framework (Raghu et al., 2017) — originally developed for static deep networks — to continuous-time models.

Trajectory length definition: The arc length of a given trajectory $I(t)$ (e.g., a circle in 2D space):

$$l(I(t)) = \int_t \left\| \frac{dI(t)}{dt} \right\| dt \quad (\text{traj-len})$$

By establishing a lower bound for the growth of the trajectory length, one can set a barrier between networks of shallow and deep architectures, regardless of weight configuration.

Experimental setup: Instances of Neural ODEs, CT-RNNs, and LTCs with shared f are initialized with weights $\sim \mathcal{N}(0, \sigma_w^2/k)$ and biases $\sim \mathcal{N}(0, \sigma_b^2)$. Networks are exposed to a circular input trajectory $I(t) = \{I_1(t) = \sin(t), I_2(t) = \cos(t)\}$ for $t \in [0, 2\pi]$. The first two principal components of hidden layer activations form the 2D latent trajectory, and its length is measured.

Definition — Computational Depth (L): For one hidden layer of f in a CT network, L is the average number of integration steps by the solver for each incoming input sample. For f with n layers, the total depth is $n \times L$.

Computational Depth (Table 2)

Activations	Neural ODE	CT-RNN	LTC
tanh	0.56 ± 0.016	4.13 ± 2.19	9.19 ± 2.92
sigmoid	0.56 ± 0.00	5.33 ± 3.76	7.00 ± 5.36
ReLU	1.29 ± 0.10	4.31 ± 2.05	56.9 ± 9.03
Hard-tanh	0.61 ± 0.02	4.05 ± 2.17	81.01 ± 10.05

(100 tries, $\Delta t = 0.01$, $T = 100$, 1 layer, width 100, $\sigma_w^2 = 2$, $\sigma_b^2 = 1$)

Key empirical observations:

1. Exponential growth of trajectory length of Neural ODEs and CT-RNNs with Hard-tanh/ReLU activations, with unchanged latent space shape regardless of weight profile.
2. LTCs show slower growth rate but achieve **much greater complexity** in absolute terms.
3. Consistently more complex trajectories for LTCs across all settings.

4. Trajectory length does **not** grow by depth in multi-layer tanh/sigmoid continuous-time networks (unlike static deep networks).
 5. Trajectory length varies by activation, weight/bias variance, width, and depth.
 6. Linear growth with network width.
 7. Faster-than-linear growth with weight variance.
 8. Reluctant to the choice of ODE solver.
-

Theorem 4 — Trajectory Length Bounds for Neural ODEs and CT-RNNs

Let $dx/dt = f_{n,k}(\mathbf{x}(t), \mathbf{I}(t), \theta)$ with $\theta = \{W, b\}$ represent a Neural ODE, and $dx(t)/dt = -\mathbf{x}(t)/\tau + f_{n,k}(\mathbf{x}(t), \mathbf{I}(t), \theta)$ with $\theta = \{W, b, \tau\}$ a CT-RNN. f is randomly weighted with Hard-tanh activations. Let $\mathbf{I}(t)$ be a 2D input trajectory with progressive points having a perpendicular component to $\mathbf{I}(t)$ for all δt , with L = number of solver steps. Then, defining the projection of the first two principal components' scores of the hidden states as the 2D latent trajectory space of a layer d , $z^{(d)}(I(t)) = z^{(d)}(t)$:

For Neural ODEs:

$$\mathbb{E} [l(z^{(d)}(t))] \geq O \left(\left(\frac{\sigma_w \sqrt{k}}{\sqrt{\sigma_w^2 + \sigma_b^2 + k\sqrt{\sigma_w^2 + \sigma_b^2}}} \right)^{d \times L} l(I(t)) \right) \quad (7)$$

For CT-RNNs:

$$\mathbb{E} [l(z^{(d)}(t))] \geq O \left(\left(\frac{(\sigma_w - \sigma_b) \sqrt{k}}{\sqrt{\sigma_w^2 + \sigma_b^2 + k\sqrt{\sigma_w^2 + \sigma_b^2}}} \right)^{d \times L} l(I(t)) \right) \quad (8)$$

Proof basis: Follows similar steps as Raghu et al. (2017) on trajectory length bounds for deep networks with piecewise linear activations, with careful considerations due to the continuous-time setup. A recurrence is formulated between the norm of the hidden state gradient in layer $d + 1$, $\|dz/dt^{(d+1)}\|$, and the expectation of the norm of the right-hand side of the ODEs. The recurrence is then rolled back to reach the inputs.

Key difference between Eqs. 7 and 8: The CT-RNN bound (Eq. 8) has $(\sigma_w - \sigma_b)$ instead of σ_w in the numerator, explaining why CT-RNNs produce shorter trajectories than Neural ODEs.

Theorem 5 — Growth Rate of LTC's Trajectory Length (key)

Let Eq. 1 determine an LTC with $\theta = \{W, b, \tau, A\}$. With the same conditions on f and $\mathbf{I}(t)$ as in Theorem 4:

$$\mathbb{E} [l(z^{(d)}(t))] \geq O \left(\left(\frac{\sigma_w \sqrt{k}}{\sqrt{\sigma_w^2 + \sigma_b^2 + k\sqrt{\sigma_w^2 + \sigma_b^2}}} \right)^{d \times L} \left(\sigma_w + \frac{\|z^{(d)}\|}{\min(\delta t, L)} \right) l(I(t)) \right) \quad (9)$$

Proof basis: Constructs the recurrence between the norm of the hidden state gradients and the components of the right-hand side of LTCs separately, which progressively build up the bound.

Key insight: Compared to Eqs. 7 and 8, the LTC bound (Eq. 9) has an additional multiplicative factor: $(\sigma_w + \|z^{(d)}\|/\min(\delta t, L))$. This factor grows with:

- The weight variance σ_w (explaining the faster-than-linear growth observed in experiments).
 - The norm of the latent trajectory $\|z^{(d)}\|$ (the bound gets larger as the trajectory becomes more complex).
 - The inverse of the minimum of step size and solver steps (finer discretization amplifies the bound).
-

9. Discussion of Theoretical Bounds (from paper)

1. **Neural ODEs:** The bound is very similar to that of a static n -layer deep network, except for exponential dependence on solver steps L instead of layers.
2. **CT-RNNs:** The bound suggests shorter trajectory length compared to Neural ODEs, due to the $(\sigma_w - \sigma_b)$ base of the exponent. This matches experiments consistently.
3. **LTCs:** Fig. 2B and Fig. 3C show faster-than-linear growth for LTC’s trajectory length as a function of weight distribution variance. This is confirmed by the extra multiplicative factor in Eq. 9.
4. **Width dependence:** LTC’s lower bound depicts linear growth of trajectory length with width k , validated by experiments in Fig. 3B.
5. **Computed depths:** Given computational depth L from Table 2 for Hard-tanh, the lower bounds for Neural ODEs, CT-RNNs, and LTCs justify the longer trajectory length of LTCs observed experimentally.

10. Experimental Results Summary

Time Series Predictions (Table 3)

Dataset	Metric	LSTM	CT-RNN	Neural ODE	CT-GRU	LTC (ours)
Gesture	accuracy	64.57%	59.01%	46.97%	68.31%	69.55%
Occupancy	accuracy	93.18%	94.54%	90.15%	91.44%	94.63%
Activity recognition	accuracy	95.85%	95.73%	97.26%	96.16%	95.67%
Sequential MNIST	accuracy	98.41%	96.73%	97.61%	98.27%	97.57%
Traffic	squared error	0.169	0.224	1.512	0.389	0.099
Power	squared error	0.628	0.742	1.254	0.586	0.642
Ozone	F1-score	0.284	0.236	0.168	0.260	0.302

LTCs achieved 5%–70% performance improvement over other RNN models in 4/7 experiments, with comparable performance in the remaining 3.

Person Activity (Table 4 — 1st Setting)

Algorithm	Accuracy
LSTM	83.59%
CT-RNN	81.54%
Latent ODE	76.48%
CT-GRU	85.27%
LTC (ours)	85.48%

Person Activity (Table 5 — 2nd Setting)

Algorithm	Accuracy
RNN Δ_t	0.797
RNN-Decay	0.800
RNN GRU-D	0.806
RNN-VAE	0.343
Latent ODE (D enc.)	0.835
ODE-RNN	0.829
Latent ODE (C enc.)	0.846
LTC (ours)	0.882

Half-Cheetah Kinematic Modeling (Table 6)

Algorithm	MSE
LSTM	2.500
CT-RNN	2.838
Neural ODE	3.805
CT-GRU	3.014
LTC (ours)	2.308

11. Scope and Limitations (from paper)

Long-term Dependencies: Similar to many variants of time-continuous models, LTCs express the **vanishing gradient phenomenon** when trained by gradient descent. Although the model shows promise on a variety of time-series prediction tasks, they would not be the obvious choice for learning long-term dependencies in their current format.

Choice of ODE Solver: Performance of time-continuous models is heavily tied to their numerical implementation. While LTCs perform well with advanced variable-step solvers and the Fused fixed-step solver, their performance is **majorly influenced** when off-the-shelf explicit Euler methods are used.

Time and Memory: Neural ODEs are remarkably fast compared to more sophisticated models such as LTCs. LTCs significantly enhance expressive power at the expense of elevated time and memory complexity.

Causality: Models described by ODE semantics inherently possess causal structures. LTCs’ semantics resemble DCMs (Friston et al., 2003) with a bilinear dynamical system approximation. A natural application domain would be the control of robots in continuous-time observation and action spaces.

12. Parameter Summary Table

Parameter	Symbol	Dimensions	Learnable	Typical Range	Class Visibility	Interpretation
Time-constant	τ	$N \times 1$	Yes	[0.001, 10.0]	Public	Intrinsic decay speed per neuron. Larger = slower. Init: log-uniform [0.01, 1.0] or constant 1.0.
Input weights	γ	$M \times N$	Yes	Init: $\mathcal{N}(0, \sigma_w^2/k)$	Private	Maps input features to neuronal activation.
Recurrent weights	γ_r	$N \times N$	Yes	Init: $\mathcal{N}(0, \sigma_w^2/k)$	Private	Recurrent connections among hidden neurons.
Biases	μ	$N \times 1$	Yes	Init: $\mathcal{N}(0, \sigma_b^2)$	Private	Bias/threshold for activation function.
Bias vector	A	$N \times 1$	Yes	Init: $\mathcal{N}(0, 1)$ or zeros	Private	Target state / reversal potential analog.
Output weights	W_{out}	$N \times N_{out}$	Yes	Xavier init	Private	Linear readout from hidden state.
Output bias	b_{out}	$N_{out} \times 1$	Yes	0 init	Private	Output layer bias.
Step size	Δt	scalar	No	[0.001, 1.0]	Public	ODE solver step size. Smaller = more accurate.
Unfolding steps	L	scalar	No	[1, 10]	Public	Fused solver substeps per input time step.

Parameter	Symbol	Dimensions	Learnable	Typical Range	Class Visibility	Interpretation
Number of neurons	N	scalar	No	[4, 256]	Public	Network width.
Input dimension	M	scalar	No	Task-dependent	Public	Set at construction.
Activation	—	—	No	tanh, sigmoid, ReLU, Hard-tanh	Public	Choice of nonlinearity for f .

Initialization conventions from experiments: $\sigma_w^2 = 2$, $\sigma_b^2 = 1$.

13. Variable Summary Table

Variable	Symbol	Dimensions	Type	Description
Hidden state	$\mathbf{x}(t)$	$N \times 1$	Dependent (ODE solution)	Primary state vector evolved by the solver.
Input	$\mathbf{I}(t)$	$M \times 1$	Independent (given)	External input at time t .
Time	t	scalar	Independent	Current simulation time.
Output	$\hat{y}(t)$	$N_{out} \times 1$	Dependent	Network output: $W_{out} \cdot \mathbf{x} + b_{out}$.
Nonlinearity value	f_{val}	$N \times 1$	Dependent	Output of the neural network f .
System time-constant	τ_{sys}	$N \times 1$	Dependent	$\tau/(1 + \tau f)$. Changes every step.
Loss	L_{total}	scalar	Dependent	Accumulated training loss.
Gradients	$\nabla L(\theta)$	same as θ	Dependent	Parameter gradients from BPTT.

14. Equation Index (Order of Appearance in Paper)

Paper Eq. #	Description
CT-RNN (inline)	Standard CT-RNN: $dx/dt = -x/\tau + f(x, I, t, \theta)$
S (inline)	Nonlinear drive: $S(t) = f(x, I, t, \theta)(A - x(t))$
Eq. 1	Core LTC ODE
Eq. 1a (inline)	System time-constant: $\tau_{sys} = \tau/(1 + \tau f)$
bio-1 (inline)	Biophysical neuron: $dv/dt = -g_l v + S(t)$
bio-2 (inline)	Synaptic current: $S(t) = f(v, I)(A - v)$
DCM (inline)	Dynamic Causal Model connection
Eq. 2	Fused Euler discretization
Eq. 3	Fused solver closed-form update
Eq. 3a (inline)	Typical f : $\tanh(\gamma_r x + \gamma I + \mu)$
Algorithm 1	LTC update by fused ODE solver
Algorithm 2	Training LTC by BPTT
Table 1	BPTT vs adjoint complexity
Eq. 4	Time-constant bounds (Theorem 1)
Eq. 5	Hidden state bounds (Theorem 2)
Eq. 6	Universal approximation (Theorem 3)
Table 2	Computational depth of models
traj-len (inline)	Trajectory length definition
Eq. 7	Neural ODE trajectory length bound (Theorem 4)
Eq. 8	CT-RNN trajectory length bound (Theorem 4)

Paper Eq. #	Description
Eq. 9	LTC trajectory length bound (Theorem 5)
Tables 3–6	Experimental results