Liquid Neural Networks (LNNs): Mathematical Foundations

Comprehensive Equation Reference for Implementation

Purpose: This document serves as the foundational mathematical reference for implementing Liquid Neural Network classes in MATLAB, Python, and Julia. It consolidates all equations, parameters, variables, and implementation-relevant details from three key papers:

1. **Hasani et al. (2018)** — *Liquid Time-constant Recurrent Neural Networks as Universal Approximators*
2. **Hasani et al. (2021)** — *Liquid Time-Constant Networks* (AAAI-21)
3. **Zhu et al. (2025)** — *Liquid Neural Networks: Next-Generation AI for Telecom from First Principles*

---

1. Biological Motivation and Core Concept

Liquid Time-Constant (LTC) networks are inspired by the nervous system of the nematode *Caenorhabditis elegans*. Unlike standard continuous-time RNNs (CT-RNNs) where the time-constant is a fixed scalar, LTC neurons have a **nonlinearly varying time-constant** that depends on the incoming presynaptic signals. This means each neuron's temporal dynamics adapt to its inputs, giving individual neurons much richer expressivity.

The key innovation is that the synaptic transmission between neurons is modeled by **sigmoidal nonlinearities** (modeling biophysical synaptic interactions) rather than simple constant weights. The postsynaptic neuron's state is therefore governed by an ODE whose effective time-constant varies as a function of the presynaptic states.

---

2. Single-Neuron Membrane Dynamics (2018 Paper)

Equation 1 — Membrane Integrator ODE

The dynamics of a hidden or output neuron i, with membrane potential $V_i(t)$, are modeled as a membrane integrator:

$$C_{m_i} \frac{dV_i}{dt} = G_{Leak_i}\Big(V_{Leak_i} - V_i(t)\Big) + \sum_{j=1}^{n} I_{in}^{(ij)} \tag{1}$$

What it does: This is the fundamental single-neuron ODE. The left side is the capacitive current (charge accumulation on the membrane). The right side has two terms: (a) a passive leak current that pulls the membrane potential toward its resting value, and (b) the sum of all incoming synaptic currents from other neurons.

Parameter/Variable definitions:

- $V_i(t)$ — Membrane potential (hidden state) of neuron i at time t. This is the primary state variable you are solving for.
- C_{m_i} — Membrane capacitance of neuron i. Controls how quickly the neuron's voltage changes in response to current. Larger values = slower dynamics.
- G_{Leak_i} — Leak conductance of neuron i. Determines the rate at which the neuron returns to rest when no input is present.
- V_{Leak_i} — Leak (resting) potential of neuron i. The equilibrium voltage the neuron decays toward in the absence of input.
- $I_{in}^{(ij)}$ — Total incoming current to neuron i from neuron j. This includes chemical synaptic currents and possibly gap junctions.
- n — Number of presynaptic neurons connected to neuron i.

Interpretation: This is a leaky integrator model from computational neuroscience. The neuron is an RC circuit: C_{m_i} is the capacitor and G_{Leak_i} is the resistor. Without input, $V_i(t) \to V_{Leak_i}$ exponentially with time-constant $\tau_i = C_{m_i}/G_{Leak_i}$.

Equation 2 — Chemical Synaptic Transmission

Chemical synaptic transmission from presynaptic neuron j to postsynaptic neuron i is modeled as:

$$I_{s_{ij}} = \frac{w_{ij}}{1 + e^{-\gamma_{ij}(V_j + \mu_{ij})}}\Big(E_{ij} - V_i(t)\Big) \tag{2}$$

What it does: This is the core nonlinear synapse model that distinguishes LTCs from standard RNNs. The synaptic current depends on: (a) a sigmoidal gating function of the presynaptic voltage V_j, and (b) the driving force $(E_{ij} - V_i)$, which is the difference between the synaptic reversal potential and the postsynaptic voltage. The current is zero when $V_i = E_{ij}$.

Parameter/Variable definitions:

- w_{ij} — Maximum synaptic conductance (weight) from neuron j to neuron i. Always positive. Controls the maximum magnitude of the synaptic current.
- γ_{ij} — Slope (gain) of the sigmoidal activation for the synapse from j to i. Controls how sharply the synapse turns on/off.
- μ_{ij} — Horizontal shift (bias/threshold) of the sigmoidal activation. Determines at what presynaptic voltage V_j the synapse is half-activated.
- E_{ij} — Synaptic reversal potential. Determines whether the synapse is **excitatory** ($E_{ij} > V_i$ typically) or **inhibitory** ($E_{ij} < V_i$ typically). This is a critical parameter for network behavior.
- $V_j(t)$ — Presynaptic membrane potential of neuron j.
- $V_i(t)$ — Postsynaptic membrane potential of neuron i.

The sigmoid function is defined as:

$$\sigma_i(V_j(t)) = \frac{1}{1 + e^{-\gamma_{ij}(V_j + \mu_{ij})}} \tag{2a}$$

This is a standard logistic sigmoid bounded in $(0, 1)$.

Equation 3 — Electrical Synapse (Gap Junction)

An electrical synapse (gap junction) between nodes j and i is modeled as a bidirectional junction based on Ohm's law:

$$\hat{I}_{ij} = \hat{\omega}_{ij}\Big(v_j(t) - v_i(t)\Big) \tag{3}$$

What it does: Gap junctions are direct electrical connections between neurons. Current flows proportionally to the voltage difference between connected neurons, like a resistor. Unlike chemical synapses, they are bidirectional and linear.

Parameter/Variable definitions:

- $\hat{\omega}_{ij}$ — Gap junction conductance between neurons j and i. Always non-negative.
- $v_j(t), v_i(t)$ — Membrane potentials of neurons j and i.

Interpretation: Gap junctions contribute a linear coupling term to the dynamics. They tend to synchronize connected neurons. In the universal approximation proof, gap junctions add a linear term to both the system time-constant τ_{sys} and the equilibrium state vector A.

Equation 4 — Full Single-Neuron LTC Dynamics (One Chemical Synapse)

Combining the membrane equation with one chemical synapse from neuron j:

$$\frac{dV_i}{dt} = \frac{G_{Leak_i}}{C_{m_i}}\Big(V_{Leak_i} - V_i(t)\Big) + \frac{w_{ij}}{C_{m_i}}\sigma_i(V_j(t))\Big(E_{ij} - V_i\Big) \tag{4}$$

What it does: This is Eq. 1 expanded with one chemical synapse substituted in. The first term is the leak, and the second term is the sigmoidal synaptic current divided by capacitance.

Equation 5 — Reformulated LTC Dynamics with Varying Time-Constant

By defining $\tau_i = C_{m_i}/G_{Leak_i}$ (the intrinsic neuronal time-constant), Eq. 4 can be rewritten as:

$$\frac{dV_i}{dt} = -\Big(\frac{1}{\tau_i} + \frac{w_{ij}}{C_{m_i}}\sigma_i(V_j)\Big)V_i + \Big(\frac{V_{Leak_i}}{\tau_i} + \frac{w_{ij}}{C_{m_i}}\sigma_i(V_j)E_{ij}\Big) \tag{5}$$

What it does: This reformulation reveals the key LTC property: the **effective time-constant of the system** is no longer τ_i but varies nonlinearly:

$$\tau_{system} = \frac{1}{1/\tau_i + (w_{ij}/C_{m_i})\,\sigma_i(V_j)} \tag{5a}$$

Interpretation: When presynaptic activity $\sigma_i(V_j)$ is large (strong input), the effective time-constant decreases (faster dynamics). When input is weak, the neuron reverts to its intrinsic time-constant τ_i. This is the "liquid" property — the temporal dynamics are input-dependent.

---

3. Network-Level LTC Dynamics (2018 Paper)

Equation 6 — Matrix-Form Network ODE

The overall network dynamics of an LTC RNN with n output neurons and N hidden (inter)neurons, representing all internal states as $u(t) = [u_1(t), \ldots, u_{n+N}(t)]^T$:

$$\dot{u}(t) = -\Big(1/\tau + W\sigma(u(t))\Big)u(t) + A + W\sigma(u(t))B \tag{6}$$

What it does: This is the full network ODE in matrix form. Every neuron's state is updated simultaneously. The nonlinear term $W\sigma(u(t))$ modulates both the decay rate (through the time-constant) and the input drive (through the reversal potentials B).

Parameter/Variable definitions:

- $u(t) \in \mathbb{R}^{n+N}$ — Full state vector: n output neurons + N hidden neurons.
- $\tau^{n+N} > 0$ — Vector of intrinsic neuronal time-constants. All entries are positive.
- $\sigma(\cdot)$ — Element-wise C^1-sigmoid function applied to the state vector.
- $W \in \mathbb{R}^{(n+N)\times(n+N)}$ — Effective weight matrix. This is the product of the raw weight matrix (shape $(n+N)\times(n+N)$) and an $(n+N)$ vector containing $1/C_{m_i}$ values.
- $A \in \mathbb{R}^{n+N}$ — Vector of resting state contributions. Contains all V_{Leak_i}/C_{m_i} values.
- $B \in \mathbb{R}^{n+N}$ — Vector of synaptic reversal potentials. Contains all E_{ij} values.

Constraints on A and B: Both A and B have entries bounded to a range $[-\alpha, \beta]$ for $0 < \alpha < +\infty$ and $0 \leq \beta < +\infty$.

Architecture note: Hidden neurons have recurrent connections among themselves, but connect to output (motor) neurons in a **feed-forward** manner only.

---

4. Abstract LTC Formulation (2021 Paper)

The 2021 paper provides a cleaner, more implementation-friendly formulation.

Equation 7 (2021 Eq. 1) — Core LTC Hidden State ODE

$$\frac{d\mathbf{x}(t)}{dt} = -\left[\frac{1}{\tau} + f\big(\mathbf{x}(t), \mathbf{I}(t), t, \theta\big)\right]\mathbf{x}(t) + f\big(\mathbf{x}(t), \mathbf{I}(t), t, \theta\big)A \tag{7}$$

What it does: This is the central equation you will implement. It says the rate of change of the hidden state has two parts: (a) a state-dependent decay term $-[1/\tau + f(\cdot)]\mathbf{x}(t)$ that pulls the state toward zero with an input-dependent rate, and (b) a drive term $f(\cdot)A$ that pushes the state toward the bias vector A scaled by the nonlinear function f.

Parameter/Variable definitions:

- $\mathbf{x}(t) \in \mathbb{R}^N$ — Hidden state vector of N neurons at time t. This is the main state you integrate.
- $\mathbf{I}(t) \in \mathbb{R}^M$ — External input vector of dimension M at time t.
- $\tau \in \mathbb{R}^{N\times 1}$ — Vector of intrinsic time-constants, one per neuron. All entries positive.
- $f(\cdot)$ — A neural network (arbitrary architecture) parametrized by θ. Outputs a value in $\mathbb{R}^N$ for each neuron. Typically: $f = \tanh(\gamma_r\mathbf{x} + \gamma\mathbf{I} + \mu)$ or $f = \sigma(\gamma_r\mathbf{x} + \gamma\mathbf{I} + \mu)$.
- θ — All learnable parameters of f. Includes weights, recurrent weights, and biases.
- $A \in \mathbb{R}^{N\times 1}$ — Bias vector (analogous to synaptic reversal potentials E_{ij}). Learnable.

Derivation context: This comes from writing the hidden state flow as $d\mathbf{x}/dt = -\mathbf{x}/\tau + \mathbf{S}(t)$ where $\mathbf{S}(t) = f(\mathbf{x}(t), \mathbf{I}(t), t, \theta)(A - \mathbf{x}(t))$. Plugging $\mathbf{S}$ into the linear ODE yields Eq. 7.

Effective system time-constant:

$$\tau_{sys} = \frac{\tau}{1 + \tau \cdot f\big(\mathbf{x}(t), \mathbf{I}(t), t, \theta\big)} \tag{7a}$$

This is the liquid time-constant: it varies with the input and hidden state through f.

Equation 8 (2021 Eq. 2) — Fused Euler Discretization

The fused ODE solver discretizes $dx/dt = f(x)$ by mixing explicit and implicit Euler:

$$x(t_{i+1}) = x(t_i) + \Delta t \cdot f\big(x(t_i),\ x(t_{i+1})\big) \tag{8}$$

What it does: This replaces only the $x(t_i)$ terms that appear **linearly** in f with $x(t_{i+1})$, making the equation solvable symbolically for $x(t_{i+1})$. This gives better stability than explicit Euler (important because LTC ODEs are stiff) without the full cost of implicit Euler.

Equation 9 (2021 Eq. 3) — Fused Solver Closed-Form Update (key)

This is the primary equation you will implement in your forward pass:

$$\mathbf{x}(t + \Delta t) = \frac{\mathbf{x}(t) + \Delta t \cdot f\big(\mathbf{x}(t), \mathbf{I}(t), t, \theta\big) \odot A}{1 + \Delta t\Big(\frac{1}{\tau} + f\big(\mathbf{x}(t), \mathbf{I}(t), t, \theta\big)\Big)} \tag{9}$$

What it does: This computes one forward step of the LTC network. Given the current state $\mathbf{x}(t)$ and input $\mathbf{I}(t)$, it produces the next state $\mathbf{x}(t + \Delta t)$. All operations (division, multiplication) are **element-wise**. The $\odot$ symbol denotes the Hadamard (element-wise) product.

Implementation notes:

- $f(\cdot)$ and all divisions are applied **element-wise** across the N neurons.
- This is applied L times (unfolding steps) per input time step for accuracy.
- Computational complexity for input sequence of length T: $O(L \times T)$.
- A dense LTC with N neurons has the same complexity as a dense LSTM with N cells.

Typical choice for f: For a tanh nonlinearity:

$$f = \tanh\big(\gamma_r \mathbf{x} + \gamma \mathbf{I} + \mu\big) \tag{9a}$$

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.

---

5. Training: Backpropagation Through Time (2021 Paper)

Algorithm — BPTT for LTCs (2021 Algorithm 2)

The LTC is trained by vanilla BPTT (not the adjoint method), which trades memory for accurate gradient computation:

1. **Forward pass:** For each time step $j = 1, \ldots, T$:
   - Compute $\mathbf{x} = f(\mathbf{I}(t), \mathbf{x})$ using Eq. 9 (applied L times)
   - Compute output: $\hat{y}(t) = W_{out} \cdot \mathbf{x} + b_{out}$
   - Accumulate loss: $L_{total} = \sum_{j=1}^{T} \mathcal{L}(y_j(t), \hat{y}_j(t))$
2. **Backward pass:** Compute $\nabla L(\theta) = \partial L_{total}/\partial\theta$ via standard backpropagation through the unrolled computation graph.
3. **Update:** $\theta \leftarrow \theta - \alpha \nabla L(\theta)$ (or use Adam optimizer).

Complexity comparison (single layer):

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**

The adjoint method has lower memory but introduces numerical errors in the backward pass. BPTT is preferred for LTCs because accurate gradients are critical.

---

6. Bounds on Time-Constant and Neural State

Equation 10 (2018 Eq. 40) — Time-Constant Bounds (with gap junctions)

For neuron i receiving N chemical synapses and P gap junctions:

$$\frac{C_i}{g_i + \sum_{j=1}^{N} w_{ij} + \sum_{j=1}^{P} \hat{w}_{ij}} \;\leq\; \tau_i \;\leq\; \frac{C_i}{g_i + \sum_{j=1}^{P} \hat{w}_{ij}} \tag{10}$$

What it does: Proves the neuronal time-constant is always bounded between a minimum (when all chemical synapses are fully active) and a maximum (when all chemical synapses are silent). Gap junctions always contribute to the lower bound of τ.

Derivation: The sigmoid in Eq. 2 is bounded: $0 < \sigma < 1$. Setting $\sigma = 1$ gives τ_{min}; setting $\sigma = 0$ gives τ_{max}.

The **minimum** (fastest dynamics):

$$\tau_i^{min} = \frac{C_i}{g_i + \sum_{j=1}^{N} w_{ij} + \sum_{j=1}^{P} \hat{w}_{ij}} \tag{10a}$$

The **maximum** (slowest dynamics):

$$\tau_i^{max} = \frac{C_i}{g_i + \sum_{j=1}^{P} \hat{w}_{ij}} \tag{10b}$$

Equation 11 (2021 Eq. 4) — Simplified Time-Constant Bounds (no gap junctions)

For the abstract formulation (2021 paper), where neuron i receives M incoming connections and $W_i = \sum_j |w_{ij}|$:

$$\frac{\tau_i}{1 + \tau_i W_i} \;\leq\; \tau_{sys_i} \;\leq\; \tau_i \tag{11}$$

What it does: Shows the system time-constant is bounded between $\tau_i/(1 + \tau_i W_i)$ (maximum input activation) and τ_i (zero input activation). The time-constant can never exceed the intrinsic τ_i and can never drop below $\tau_i/(1 + \tau_i W_i)$.

Equation 12 (2021 Eq. 5) — Hidden State Bounds

For neuron i within an LTC, on a finite interval $[0, T]$:

$$\min(0, A_i^{min}) \;\leq\; x_i(t) \;\leq\; \max(0, A_i^{max}) \tag{12}$$

What it does: The hidden state of every LTC neuron is always bounded. This guarantees **state stability** — outputs never explode even if inputs grow to infinity. A_i^{min} and A_i^{max} are the minimum and maximum values of the bias vector A for neuron i.

Interpretation: This is a critical property for robust deployment. Standard neural ODEs and RNNs can have unbounded hidden states. LTCs cannot, by construction.

Equation 13 (2018 Eq. 50) — State Bounds (biophysical form)

In the biophysical formulation, for neuron i receiving N chemical synapses:

$$\min_{t\in I}\left(V_{Leak_i},\ E_{ij}^{min}\right)\ \le\ v_i(t)\ \le\ \max_{t\in I}\left(V_{Leak_i},\ E_{ij}^{max}\right) \tag{13}$$

What it does: The membrane potential is always bounded between the minimum of (leak potential, smallest reversal potential) and the maximum of (leak potential, largest reversal potential). The neuron can never go above the most excitatory reversal potential or below the most inhibitory one.

---

7. Universal Approximation (2018 Paper)

Theorem (2018 Theorem 1)

Let S be an open subset of $\mathbb{R}^n$, $F : S \to \mathbb{R}^n$ a C^1-mapping, and $\dot{x} = F(x)$ determine a dynamical system on S. Let D be a compact subset of S and consider a finite trajectory $I = [0, T]$. Then, for a positive ϵ, there exist an integer N and an LTC network with N hidden units, n output units, such that for any trajectory $\{x(t); t \in I\}$ with initial value $x(0) \in D$ and proper initialization:

$$\max_{t\in I} |x(t) - u(t)| < \epsilon \tag{14}$$

Interpretation: LTCs are universal approximators of continuous dynamical systems. Any finite trajectory of any n-dimensional ODE system can be approximated arbitrarily closely by an LTC with enough hidden units.

Key equations from the proof:

Lemma 1 — Existence and uniqueness (2018 Eq. 7):

$$\dot{x} = -(1/\tau + F(x))x + A + BF(x) \tag{15}$$

For bounded C^1 function F, this has a unique solution on $[0, \infty)$.

State bounds from Lemma 1 (2018 Eq. 10-11):

$$\min\left\{|x_i(0)|, \frac{\tau(A + BM)}{1 + \tau M}\right\} \le x_i(t) \le \max\left\{|x_i(0)|, \frac{\tau(A + BM)}{1 + \tau M}\right\} \tag{16}$$

$$\sqrt{n}\, C_1 \le x(t) \le \sqrt{n}\, C_2 \tag{17}$$

System time-constant for the proof (2018 Eq. 16):

$$\tau_{sys} = \frac{1}{1/\tau + W_l \sigma(Cx + \mu)} \tag{18}$$

---

8. Expressivity — Trajectory Length Bounds (2021 Paper)

Equation 19 (2021 Eq. 7) — Neural ODE / CT-RNN Trajectory Length Lower Bound

For Neural ODEs with $d \times L$ computational depth, width k, weight variance σ_w^2, bias variance σ_b^2:

$$\mathbb{E}\left[l\left(z^{(d)}(t)\right)\right] \ge O\left(\left(\frac{\sigma_w\sqrt{k}}{\sqrt{\sigma_w^2 + \sigma_b^2 + k\sqrt{\sigma_w^2 + \sigma_b^2}}}\right)^{d\times L} l(I(t))\right) \tag{19}$$

Equation 20 (2021 Eq. 8) — CT-RNN Trajectory Length Lower Bound

$$\mathbb{E}\left[l\big(z^{(d)}(t)\big)\right] \geq O\left(\left(\frac{(\sigma_w - \sigma_b)\sqrt{k}}{\sqrt{\sigma_w^2 + \sigma_b^2 + k\sqrt{\sigma_w^2 + \sigma_b^2}}}\right)^{d\times L} l(I(t))\right) \tag{20}$$

Equation 21 (2021 Eq. 9) — LTC Trajectory Length Lower Bound (key)

$$\mathbb{E}\left[l\big(z^{(d)}(t)\big)\right] \geq O\left(\left(\frac{\sigma_w\sqrt{k}}{\sqrt{\sigma_w^2 + \sigma_b^2 + k\sqrt{\sigma_w^2 + \sigma_b^2}}}\right)^{d\times L} \left(\frac{\|z^{(d)}\|}{\sigma_w} + \frac{1}{\min(\delta t, L)}\right) l(I(t))\right) \tag{21}$$

What these bounds mean: LTCs have a **larger** trajectory length lower bound than both Neural ODEs and CT-RNNs. The additional factor $(\|z^{(d)}\|/\sigma_w + 1/\min(\delta t, L))$ in Eq. 21 explains LTCs' consistently longer and more complex latent trajectories observed experimentally. Longer trajectory length = greater expressivity = ability to represent more complex temporal patterns.

Key empirical finding: With Hard-tanh activations, 1 layer, width 100, $\sigma_w^2 = 2$, $\sigma_b^2 = 1$, the computational depth (average solver steps) was: Neural ODE = 0.61, CT-RNN = 4.05, **LTC = 81.01**.

---

9. Closed-Form Continuous-Time (CfC) Networks (2025 Paper Context)

The 2025 paper (Zhu et al.) discusses that while LTCs require iterative ODE solvers, a **closed-form solution** (CfC) can approximate the LTC ODE solution directly with a few parameters. CfCs are represented by a specially designed deep neural network structure that eliminates the need for numerical ODE solvers entirely.

CfC Key Idea: Instead of numerically solving Eq. 7, CfC networks learn a direct mapping that approximates the solution of the ODE. This dramatically reduces computational cost while maintaining the adaptability of LTCs.

Neural Circuit Policies (NCPs): NCPs combine multiple CfC or LTC neurons into a structured 4-layer architecture:

1. **Sensory neuron layer** — receives raw input
2. **Inter neurons layer** — processes and integrates information
3. **Command neurons layer** — forms higher-level representations
4. **Motor neurons layer** — produces outputs

These layers have **sparse** connections both within and between them, mimicking biological neural circuits. This sparsity reduces computational complexity and is a key design principle for implementation.

---

10. Implementation Architecture Summary

Complete Forward Pass (per time step)

Given input $\mathbf{I}(t)$ and current state $\mathbf{x}(t)$:

1. **Compute nonlinearity** f:
$$f_{val} = \text{activation}\big(\gamma_r \cdot \mathbf{x}(t) + \gamma \cdot \mathbf{I}(t) + \mu\big) \tag{22}$$

2. **Apply fused solver** L times:
$$\mathbf{x}(t + \Delta t) = \frac{\mathbf{x}(t) + \Delta t \cdot f_{val} \odot A}{1 + \Delta t \cdot (1/\tau + f_{val})} \tag{23}$$

3. **Compute output**:
$$\hat{y}(t) = W_{out} \cdot \mathbf{x}(t + \Delta t) + b_{out} \tag{24}$$

Output layer

The readout from the LTC hidden state to the output is a simple linear layer. The output weights W_{out} and bias b_{out} are separate from the LTC cell parameters.

---

11. Summary of All Numbered Equations

#	Equation	Source	Role
1	$C_{m_i}\frac{dV_i}{dt} = G_{Leak_i}(V_{Leak_i} - V_i) + \sum I_{in}^{(ij)}$	2018 Eq.1	Membrane integrator ODE
2	$I_{s_{ij}} = \frac{w_{ij}}{1+e^{-\gamma_{ij}(V_j+\mu_{ij})}}(E_{ij} - V_i)$	2018 Eq.2	Chemical synapse model
2a	$\sigma_i(V_j) = \frac{1}{1+e^{-\gamma_{ij}(V_j+\mu_{ij})}}$	2018 (inline)	Sigmoidal activation
3	$\hat{I}_{ij} = \hat{\omega}_{ij}(v_j - v_i)$	2018 Eq.3	Gap junction model
4	$\frac{dV_i}{dt} = \frac{G_{Leak_i}}{C_{m_i}}(V_{Leak_i} - V_i) + \frac{w_{ij}}{C_{m_i}}\sigma_i(V_j)(E_{ij} - V_i)$	2018 Eq.4	Single synapse LTC neuron
5	$\frac{dV_i}{dt} = -(1/\tau_i + \frac{w_{ij}}{C_{m_i}}\sigma_i(V_j))V_i + (\frac{V_{Leak_i}}{\tau_i} + \frac{w_{ij}}{C_{m_i}}\sigma_i(V_j)E_{ij})$	2018 Eq.5	Varying time-constant form
5a	$\tau_{system} = \frac{1}{1/\tau_i + (w_{ij}/C_{m_i})\sigma_i(V_j)}$	2018 (inline)	Nonlinear time-constant
6	$\dot{u} = -(1/\tau + W\sigma(u))u + A + W\sigma(u)B$	2018 Eq.6	Full network matrix ODE
7	$\frac{d\mathbf{x}}{dt} = -[1/\tau + f(\mathbf{x}, \mathbf{I}, t, \theta)]\mathbf{x} + f(\mathbf{x}, \mathbf{I}, t, \theta)A$	2021 Eq.1	Core LTC ODE (abstract)
7a	$\tau_{sys} = \frac{\tau}{1+\tau f(\mathbf{x},\mathbf{I},t,\theta)}$	2021 (inline)	System time-constant
8	$x(t_{i+1}) = x(t_i) + \Delta t \cdot f(x(t_i), x(t_{i+1}))$	2021 Eq.2	Fused Euler discretization
9	$\mathbf{x}(t + \Delta t) = \frac{\mathbf{x}(t)+\Delta t \cdot f(\cdot)\odot A}{1+\Delta t(1/\tau+f(\cdot))}$	2021 Eq.3	**Fused solver update (implement this)**
9a	$f = \tanh(\gamma_r\mathbf{x} + \gamma\mathbf{I} + \mu)$	2021 (inline)	Typical nonlinearity
10	$C_i/(g_i + \sum w_{ij} + \sum \hat{w}_{ij}) \le \tau_i \le C_i/(g_i + \sum \hat{w}_{ij})$	2018 Eq.40	Time-constant bounds (bio)
11	$\tau_i/(1 + \tau_i W_i) \le \tau_{sys_i} \le \tau_i$	2021 Eq.4	Time-constant bounds (abstract)
12	$\min(0, A_i^{min}) \le x_i(t) \le \max(0, A_i^{max})$	2021 Eq.5	Hidden state bounds
13	$\min(V_{Leak_i}, E_{ij}^{min}) \le v_i(t) \le \max(V_{Leak_i}, E_{ij}^{max})$	2018 Eq.50	State bounds (biophysical)
14	$\max_{t\in I} \lvert x(t) - u(t)\rvert < \epsilon$	2018 Thm.1	Universal approximation
15	$\dot{x} = -(1/\tau + F(x))x + A + BF(x)$	2018 Eq.7	Lemma 1 (existence/uniqueness)
16	State bound via $\tau(A + BM)/(1 + \tau M)$	2018 Eq.10	Proof state bounds
17	$\sqrt{n}C_1 \le x(t) \le \sqrt{n}C_2$	2018 Eq.11	Norm bound on state
18	$\tau_{sys} = 1/(1/\tau + W_l\sigma(Cx + \mu))$	2018 Eq.16	System τ (proof)
19	Neural ODE trajectory length lower bound	2021 Eq.7	Expressivity bound
20	CT-RNN trajectory length lower bound	2021 Eq.8	Expressivity bound
21	LTC trajectory length lower bound	2021 Eq.9	Expressivity bound
22	$f_{val} = \text{activation}(\gamma_r\mathbf{x} + \gamma\mathbf{I} + \mu)$	Implementation	Nonlinearity computation
23	Fused step (= Eq. 9 restated)	Implementation	Core update step
24	$\hat{y} = W_{out}\mathbf{x} + b_{out}$	Implementation	Output computation

---

12. Complete Parameter Reference Table

Parameter	Symbol	Dimensions	Type	Learnable	Typical Range	Class Visibility	Interpretation
Time-constant	τ	$N \times 1$	Independent	Yes	$[0.001, 10.0]$	**Public** (user-tunable hyperparameter/init)	Intrinsic decay speed per neuron. Larger = slower dynamics. Initialize positive, often log-uniform in $[0.01, 1.0]$.
Input weights	γ	$M \times N$	Independent	Yes	$[-2, 2]$ (init)	**Private** (internal learned)	Maps input features to neuronal activation. Init: $\mathcal{N}(0, \sigma_w^2/k)$.
Recurrent weights	γ_r	$N \times N$	Independent	Yes	$[-2, 2]$ (init)	**Private** (internal learned)	Recurrent connections among hidden neurons. Init: $\mathcal{N}(0, \sigma_w^2/k)$.
Biases	μ	$N \times 1$	Independent	Yes	$[-1, 1]$ (init)	**Private** (internal learned)	Bias/threshold for activation. Init: $\mathcal{N}(0, \sigma_b^2)$.
Bias vector (A)	A	$N \times 1$	Independent	Yes	$[-1, 1]$ (init)	**Private** (internal learned)	Target state / reversal potential analog. Determines where state is driven toward.
Output weights	W_{out}	$N \times N_{out}$	Independent	Yes	Xavier init	**Private** (internal learned)	Linear readout from hidden state.
Output bias	b_{out}	$N_{out} \times 1$	Independent	Yes	0 init	**Private** (internal learned)	Output layer bias.
Step size	Δt	scalar	Independent	No (hyperparameter)	$[0.001, 1.0]$	**Public** (user-configurable)	ODE solver step size. Smaller = more accurate but slower.
Unfolding steps	L	scalar	Independent	No (hyperparameter)	$[1, 10]$	**Public** (user-configurable)	Number of fused solver substeps per input. More = more accurate.
Number of neurons	N	scalar	Independent	No (hyperparameter)	$[4, 256]$	**Public** (user-configurable)	Network width. LTCs work well with small N (19 neurons achieved autonomous driving).
Input dimension	M	scalar	Independent	No (architecture)	Task-dependent	**Public** (set at construction)	Determined by input data.
Activation function	—	—	Independent	No (hyperparameter)	tanh, sigmoid, ReLU, Hard-tanh	**Public** (user-configurable)	Choice of nonlinearity for f. tanh is most common.

Parameter	Symbol	Dimensions	Type	Learnable	Typical Range	Class Visibility	Interpretation
Membrane capacitance	C_{m_i}	$N \times 1$	Independent	Optional	$[0.1, 10.0]$	**Public** (biophysical mode)	Only needed if implementing biophysical variant (2018 form).
Leak conductance	G_{Leak_i}	$N \times 1$	Dependent (= C_{m_i}/τ_i)	Derived	Derived from τ, C_m	**Private** (derived)	Computed from C_m and τ.
Leak potential	V_{Leak_i}	$N \times 1$	Independent	Optional	$[-1, 0]$	**Public** (biophysical mode)	Resting potential in biophysical variant.
Synaptic weight	w_{ij}	$N \times N$	Independent	Yes	$[0, 2]$	**Private** (biophysical mode)	Maximum synaptic conductance. Always ≥ 0 in biophysical variant.
Sigmoid slope	γ_{ij}	$N \times N$	Independent	Yes	$[0.5, 5.0]$	**Private** (biophysical mode)	Steepness of synaptic sigmoid.
Sigmoid shift	μ_{ij}	$N \times N$	Independent	Yes	$[-2, 2]$	**Private** (biophysical mode)	Threshold for synaptic activation.
Reversal potential	E_{ij}	$N \times N$	Independent	Yes	$[-1, 1]$	**Private** (biophysical mode)	Excitatory if $> V_i$, inhibitory if $< V_i$.
Gap junction weight	$\hat{\omega}_{ij}$	P connections	Independent	Yes	$[0, 1]$	**Private** (biophysical mode, optional)	Electrical coupling. Often omitted in modern implementations.

---

13. Complete Variable Reference Table

Variable	Symbol	Dimensions	Dependent/Independent	Description
Hidden state	$\mathbf{x}(t)$ or $V_i(t)$	$N \times 1$	**Dependent** (solution of ODE)	Primary state vector. Evolved by the ODE 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** (computed from $\mathbf{x}$)	Network output, linear readout of hidden state.
Nonlinearity value	f_{val}	$N \times 1$	**Dependent** (computed)	Output of the neural network f at current state/input.
Effective time-constant	τ_{sys}	$N \times 1$	**Dependent** (varies with state)	$\tau/(1 + \tau f)$. Changes every time step.
Synaptic current	$I_{s_{ij}}$	scalar per pair	**Dependent**	Current from synapse $j \to i$.
Gap junction current	$\hat{I}_{ij}$	scalar per pair	**Dependent**	Current through gap junction $j \leftrightarrow i$.

Variable	Symbol	Dimensions	Dependent/Independent	Description
Loss	L_{total}	scalar	**Dependent**	Training loss accumulated over time.
Gradients	$\nabla L(\theta)$	same as θ	**Dependent**	Parameter gradients from BPTT.

---

14. Class Design Recommendations

Public Properties (user-facing)

- `N` — Number of hidden neurons (set at construction)
- `M` — Input dimension (set at construction)
- `N_out` — Output dimension (set at construction)
- `tau` — Time-constants (initializable, learnable)
- `dt` — Step size (hyperparameter)
- `L` — Number of unfolding steps (hyperparameter)
- `activation` — Choice of nonlinearity (`'tanh'`, `'sigmoid'`, `'relu'`, `'hard_tanh'`)
- `solver` — Choice of solver (`'fused'`, `'rk45'`, `'euler'`)

Private Properties (internal)

- `W_in` (γ) — Input weight matrix
- `W_rec` (γ_r) — Recurrent weight matrix
- `bias` (μ) — Bias vector
- `A` — Bias/reversal vector
- `W_out` — Output weight matrix
- `b_out` — Output bias vector
- `state` ($\mathbf{x}$) — Current hidden state

Key Methods

- `forward(I, x)` or `step(I, x)` — One time step using Eq. 9
- `reset_state()` — Initialize hidden state to zeros
- `get_tau_sys()` — Compute current effective time-constant (Eq. 7a)
- `get_state_bounds()` — Return theoretical bounds (Eq. 12)
- `get_tau_bounds()` — Return time-constant bounds (Eq. 11)

Initialization Guidelines

- τ: Initialize from log-uniform distribution in $[0.01, 1.0]$, or constant $\tau = 1.0$.
- γ, γ_r: Initialize from $\mathcal{N}(0, \sigma_w^2/k)$ where k is the fan-in. Common: $\sigma_w^2 = 2$.
- μ: Initialize from $\mathcal{N}(0, \sigma_b^2)$. Common: $\sigma_b^2 = 1$.
- A: Initialize from $\mathcal{N}(0, 1)$ or zeros.
- W_{out}: Xavier/Glorot initialization.
- Hidden state $\mathbf{x}(0)$: Initialize to zeros or sample from $p(x_{t_0})$.

---

15. Practical Notes for Implementation

Stiffness and Solver Choice

LTC ODEs are **stiff** (Press et al. 2007), meaning standard Runge-Kutta solvers (like RK45/Dormand-Prince used in `torchdiffeq`) require an exponential number of steps. The **fused solver** (Eq. 9) is specifically designed for LTCs and should be the default. It combines the stability of implicit Euler with the efficiency of explicit Euler.

Numerical Stability

- All divisions in Eq. 9 are element-wise. Add a small epsilon ($\sim 10^{-8}$) to the denominator to avoid division by zero.

- The sigmoid/tanh bounds naturally prevent state explosion (Eq. 12), but clipping may still be prudent during early training.
- When using `tanh` activation, $f \in [-1, 1]$, so $\tau_{sys} \in [\tau/(1+\tau), \tau/(1-\tau)]$. With `sigmoid`, $f \in [0, 1]$, so $\tau_{sys} \in [\tau/(1+\tau), \tau]$.

Architecture: NCP Sparse Wiring

For Neural Circuit Policies (NCPs), implement sparse connectivity:

- Sensory -> Inter: sparse connections
- Inter -> Inter: sparse recurrent connections
- Inter -> Command: sparse connections
- Command -> Command: sparse recurrent connections
- Command -> Motor: sparse connections

This can be implemented via binary masks on the weight matrices.

Language-Specific Notes

- **MATLAB:** Use a class with `properties` (Public/Private) and `methods`. The fused step (Eq. 9) maps directly to element-wise `./` and `.*` operations.
- **Python:** Use a `torch.nn.Module` subclass (for PyTorch) or `tf.keras.layers.Layer` (TensorFlow). Parameters registered via `nn.Parameter`.
- **Julia:** Use a `struct` with `Flux.@functor` annotation (for Flux.jl) or define as `AbstractLuxLayer` (for Lux.jl). Julia's multiple dispatch makes the ODE formulation natural with `DifferentialEquations.jl`.

---

16. Key Theoretical Results Summary

1. **LTCs are universal approximators** (2018 Theorem 1): Any finite trajectory of any continuous dynamical system can be approximated to arbitrary precision by an LTC with enough hidden units.
2. **Bounded dynamics** (2018 Lemmas 2-3, 2021 Theorems 1-2): Both the time-constant and hidden state are provably bounded, guaranteeing stability.
3. **Superior expressivity** (2021 Theorems 4-5): LTCs produce longer latent trajectories than Neural ODEs and CT-RNNs, indicating higher representational capacity. The trajectory length grows linearly with width and has a faster-than-linear dependence on weight variance.
4. **Efficient computation** (2021): The fused solver enables $O(L \times T)$ complexity matching LSTM, while achieving both high forward and backward accuracy (unlike the adjoint method).
5. **Sparse + small networks suffice** (2025): As few as 19 liquid neurons (in NCP configuration) have achieved autonomous driving, highlighting the extreme parameter efficiency of LTCs.