Liquid Time-Constant Networks

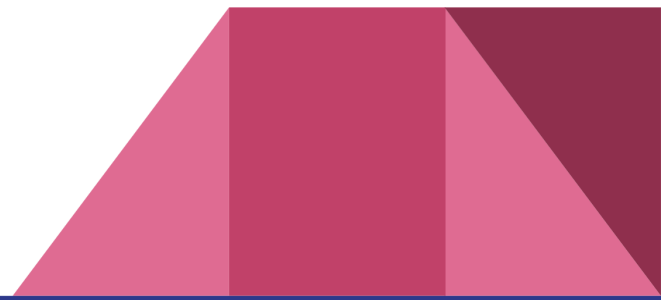
Andrew R. Darnall

April 15th 2024

Classical Artificial Neural Networks

As you all know modern Deep Learning architectures are usually **deep** (i.e. more than 3 hidden layers) and are **computationally expensive** to train

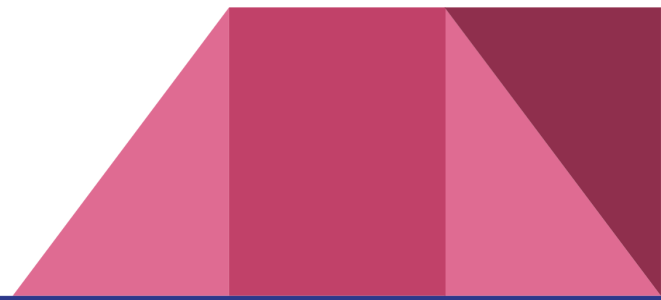
We know from a proven theorem that they are **universal function approximators**, and thus since many physical phenomena can be modeled by mathematical functions, they can be trained to learn (empirically) said function that binds an input to an output



Classical Artificial Neural Networks

There are currently many **discrete** Deep Neural Network architectures, such as **LSTMs**, **CNNs**, and so on

But despite all being made for different tasks, they all can evaluated based on their **expressivity** by measuring how the output of a network changes as the input 'moves' along a one dimensional path (i.e. the trajectory)



Limits of Classical Deep Learning Architectures

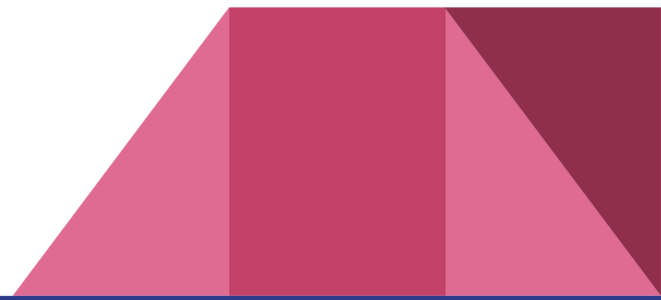
However despite some architectures being remarkably promising with the results that they have achieved, such as **AlexNet**, **ChatGPT** and **BERT**

There are always some issues that plague these architectures:

- Explainability
- Interpretability
- **Robustness**
- **Expressivity**

The Continuous Approach

Several studies have explored different approaches to solve said issues, and the approach that I present to you today is that of moving from the **discrete** to the **continuous**, via **Continuous Time** and **Continuous Depth** Artificial Nerual Networks

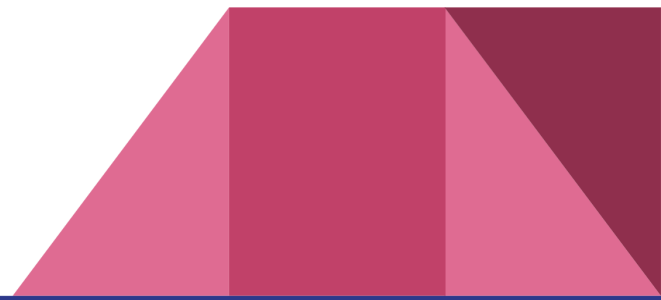


Discrete Formulation

As you all know, a modern (discrete) Deep Learning architecture is nothing more than a very elaborate and parameterized **composition of functions**

A four layer Artificial Neural Network has analytical formulation:

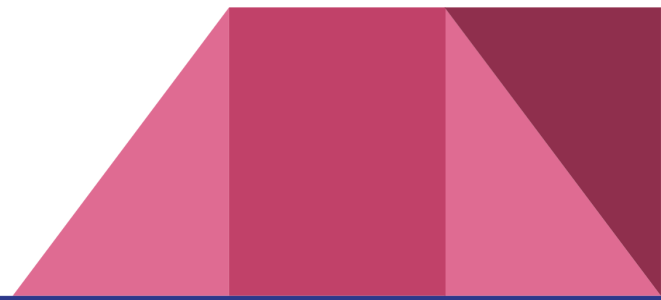
$f \circ f \circ f \circ f(x)$, with $f(x) = W_0 \alpha(W_h x + b_h) + b_0$



Continuous formulation

For the **continuous** approach, however, the evolution of the hidden state of the network is modeled by a **differential equation**

If the ANN uses **Ordinary Differential Equations**, then they belong to a family of Continuous ANNs called **Neural ODEs**



Why Differential Equations

Differential Equations have been broadly used in various fields of science and engineering to aid scholars with modeling a given phenomena of interest

In the realm of **physics**, more specifically in classical mechanics (ordinary) differential equations are amply used to model various types of physical laws such as **simple harmonic motion**, as described by:

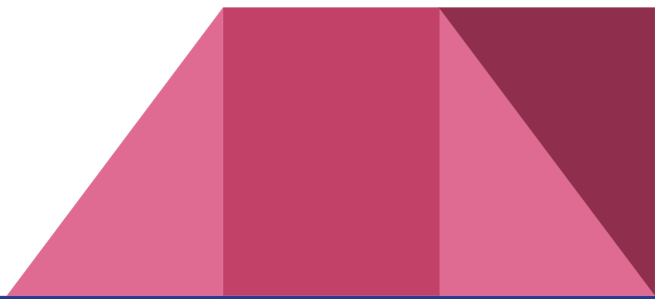
- $\frac{d^2x(t)}{dt^2} = -kx$

In general, diff. eq. are good at modeling phenomena that varies through **time**

Dynamical Systems

dynamical systems can be defined as the time-evolution of a particular phenomenon, given the environment and the circumstance under which it operates. The evolution of a dynamical system is typically formulated by differential equations. In our context, for instance, a **time-continuous neural network's state** is the **phenomenon** with its time-evolution expressed by:

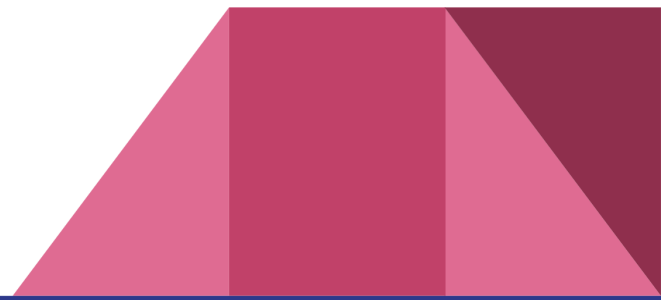
- $\dot{x}(t) = f(I(t), x(t), \theta)$



Neural ODEs

Neural ODEs are a Continuouos Time Network which is determined by Ordinary Differential Equations, with the analytical form:

- $\frac{dx(t)}{dt} = f(x(t), I(t), t, \theta)$
- $x(t) \in \mathbb{R}^D$, the Hidden State of the Network
- $I(t)$, the Input of the Network
- t, the time variable
- θ, the parameters of the Network

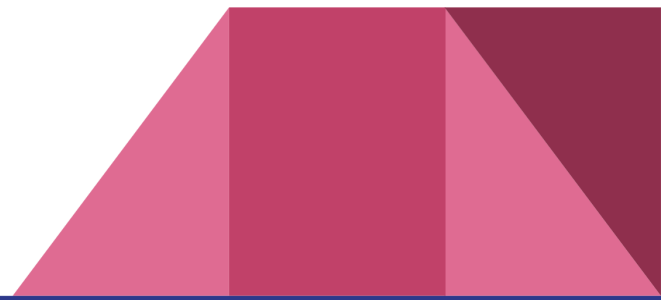


Neural ODEs

To compute the state of a Neural ODE, a Numerical ODE solver is used, among the many Numerical ODE solvers, some of the most known are:

- (Forward) Euler's Method
- Runge-Kutta Methods

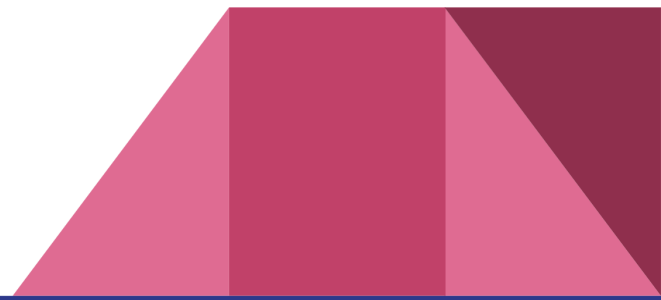
A Numerical Solver is nothing more than an algorithm that specifies the approximation of a solution to a mathematical problem for which the solution either doesn't exist or is far too complex to compute



Analytical vs Numerical Solutions

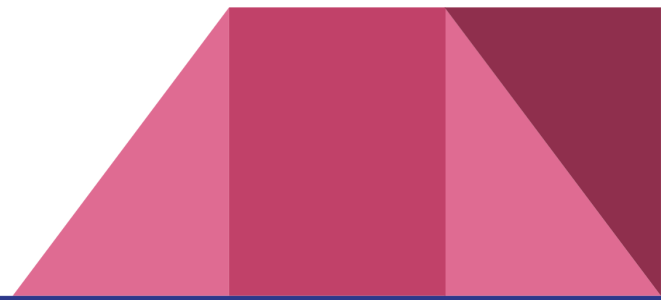
In Mathematics, certain formulations do not always have an analytical solution, such as a closed form to an optimization problem

In these cases we can approximate a solution with a certain degree of error (or precision), such methods are called Numerical Methods



Order of Numerical Solvers

The order of a numerical ordinary differential equation (ODE) solver refers to the accuracy of the method used to approximate the solution. It reflects how rapidly the error decreases with decreasing step size



Order of Numerical Solvers

- Red graph: Analytical solution
- Green graph: Second order Runge-Kutta Numerical Method (Midpoint method)
- Blue graph: First order Numerical Method (Forward Euler)

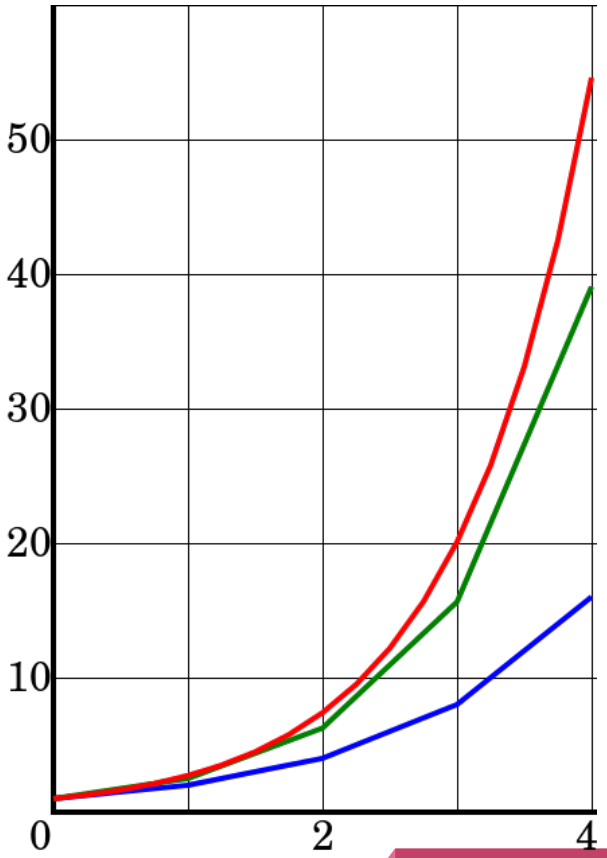


Figure 1: Numerical ODE solvers

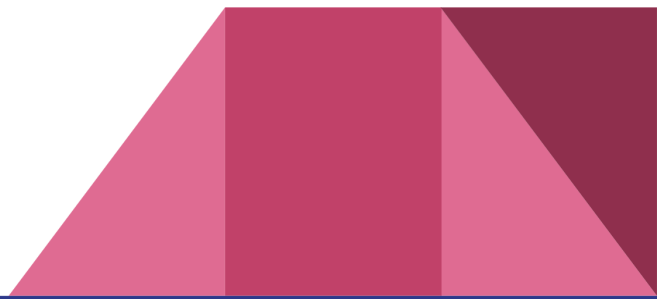
Forward Euler

The Forward Euler method approximates the solution at each time step by using the derivative at the current time to predict the value at the next time step.

Given an initial value problem:

$\frac{dy}{dt} = f(t, y)$ with $y(t_0) = y_0$

where $f(t, y)$ represents the derivative function and y_0 is the initial value of the solution at t_0.

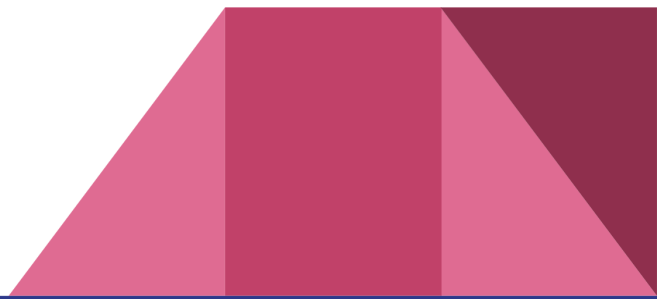


Forward Euler

The Forward Euler method updates the solution from time t_n to time t_{n+1} (where h is the step size) using the following formula:

$$y_{n+1} = y_n + h \cdot f(t_n, y_n)$$

The Computational Complexity of a single Forward Euler step is of $O(n)$, for N steps the total computational complexity of the method is of $O(N \cdot n)$



Neural ODEs

The main advantages of Neural ODEs are:

- Memory Efficiency: because backpropagation is applied to the Numerical ODE solver without backpropagating through the steps of the solver itself, and thus does not require the saving of intermediate values of the forward pass, which is a major bottleneck in the training of Deep Networks. The training thus has a **constant memory** cost.
- Adaptive Computations: Since modern ODEs provide a guarantee on the growth of the approximation error and adapt their evalutation strategy on the fly (i.e. by adjusting the step size based on the collected error). After training, accuracy can be reduced based in favor of real time low energy applications

Neural ODEs

- Continuous Time Series Models: Unlike recurrent neural networks, which require discretizing observation and emission intervals, continuously-defined dynamics can naturally incorporate data which arrives at arbitrary times. (i.e. These networks perform well on irregularly sampled data and can capture more subtle yet important changes in the data which might have long-term effects)

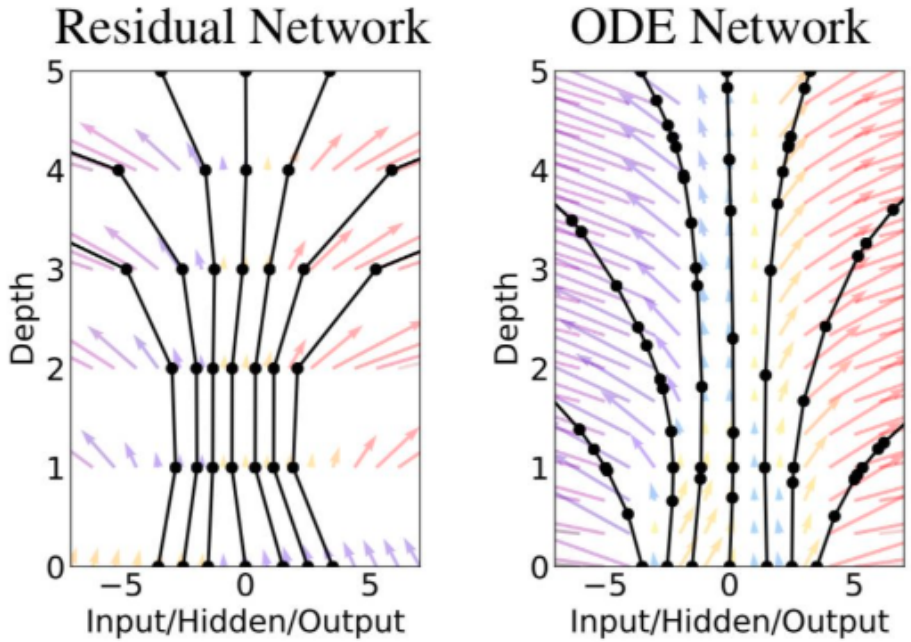
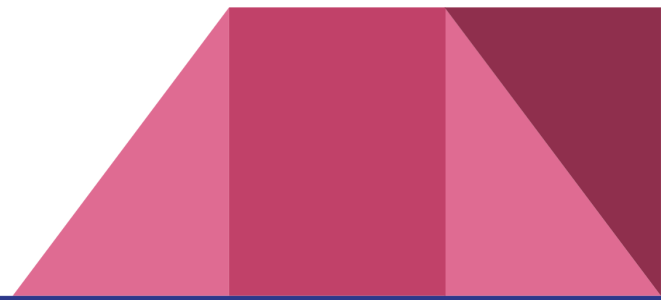


Figure 2: (Discrete) ResNet vs (Continuous) ODE Net

Backpropagation for Neural ODEs

Applying the classical Reverse-mode automatic differentiation is a challenging task in the context of Neural ODEs because of the application of the method through the ODE sovler's steps.

This forward pass computation despite being straightforward is **computationally expensive** and introduces **additional numerical error** (because it is a Numerical Method)

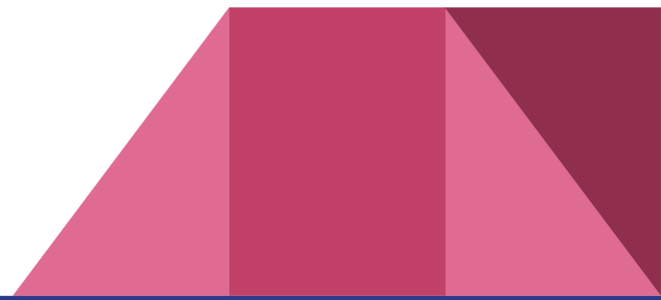


The Adjoint Sensitivity Method

An alternative formulation has been found in order to apply a Numerical Method without incurring in the same issues as the trivial version of the Backpropagation algorithm **and** is applicable to any Numerical ODE solver

The **Adjoint Sensitivity Method** has the advantage of:

- scaling linearly with respect to the problem's size
- low memory cost
- explicitly controlling the numerical error

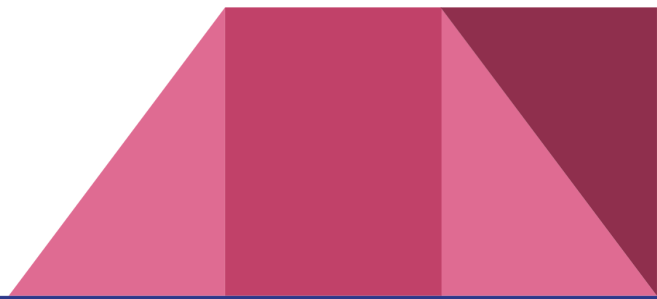


Liquid Time Constant Networks: CT-RNNs Formulation

An improvement was made on the analytical formulation of Continuous-Time RNNs:

$$\frac{dx(t)}{dt} = -\frac{x(t)}{\tau} + f(x(t), I(t), t, \theta)$$

where the addition of the 'stabilizing' term: $-\frac{x(t)}{\tau}$ assists the model in reaching an equilibrium state, where τ : **time-constant**



Liquid Time Constant Networks: Analytical Formulation

MIT's CSAIL proposed an alternative formulation to the CT-RNN, pivoting on its time constant term in order to achieve stability over the dynamics of the system:

$$\frac{dx(t)}{dt} = -\frac{x(t)}{\tau} + S(t)$$

Where $S(t) \in \mathbb{R}^M$ is a non-linear function of the form:

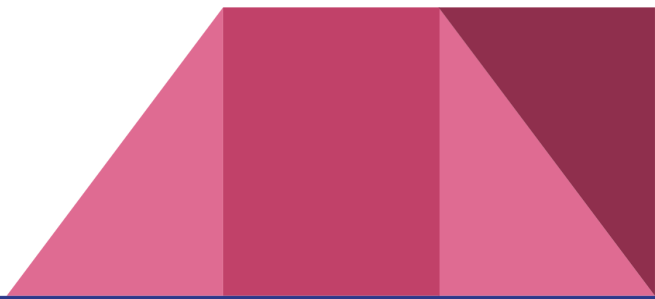
$$S(t) = f(x(t), I(t), t, \theta)(A - x(t))$$

With A and θ its parameters

Liquid Time Constant Networks: Analytical Formulation

By plugging the non-linearity which enriches the expressiveness of the overall model, we obtain:

$$\frac{dx(t)}{dt} = -[\frac{1}{\tau} + f(x(t), I(t), t, \theta)]x(t) + f(x(t), I(t), t, \theta)A$$



Liquid Time Constant Networks: The New CT-RNN formulation

Notice how the combined non-linearity $S(t)$ has made it so that the **time-constant** of the previous formulation:

$$\frac{dx(t)}{dt} = -\frac{x(t)}{\tau} + f(x(t), I(t), t, \theta)$$

Has now become a modulated dynamical system itself:

$$\frac{dx(t)}{dt} = -[\frac{1}{\tau} + f(x(t), I(t), t, \theta)]x(t) + f(x(t), I(t), t, \theta)A$$

Liquid Time Constant Networks

The **time-constant** allows for single elements of the hidden state to identify a **specialized dynamical system** for each input feature arriving at each time-point.

Therefore the network not only determines the derivative of the hidden state $x(t)$ but also serves as an input-dependent varying time constant

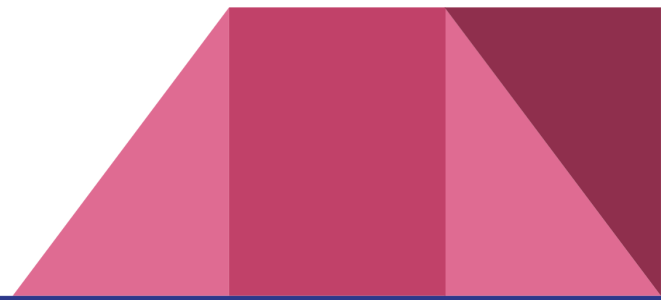
The final form of the **Liquid Time-Constant** is:

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

Not the first Liquid Computational Model

Historically speaking, this is not the first 'Liquid' Computational Model, because in the early 2000s a book called **Liquid State Machines: Theory and Applications** explored an alternative formulation to the **Turing Machine Computational Model** which was more biologically plausible and adaptable (computationally wise)

Thus grating computationally a property that our brains have which is that of **neuro plasticity**

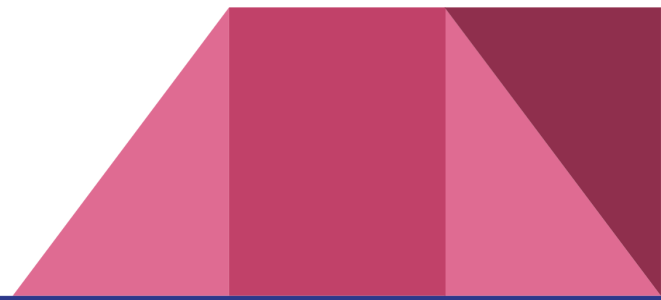


Liquid Time Constant Networks

So essentially MIT found a more 'flexible' analytical formulation for a Neural ODE

The **time-constant** is learned through the training process (τ)

And since LTCs are part of the family of Neural ODEs they too require the assistance of Numerical ODE solvers



Expressivity of LTCs

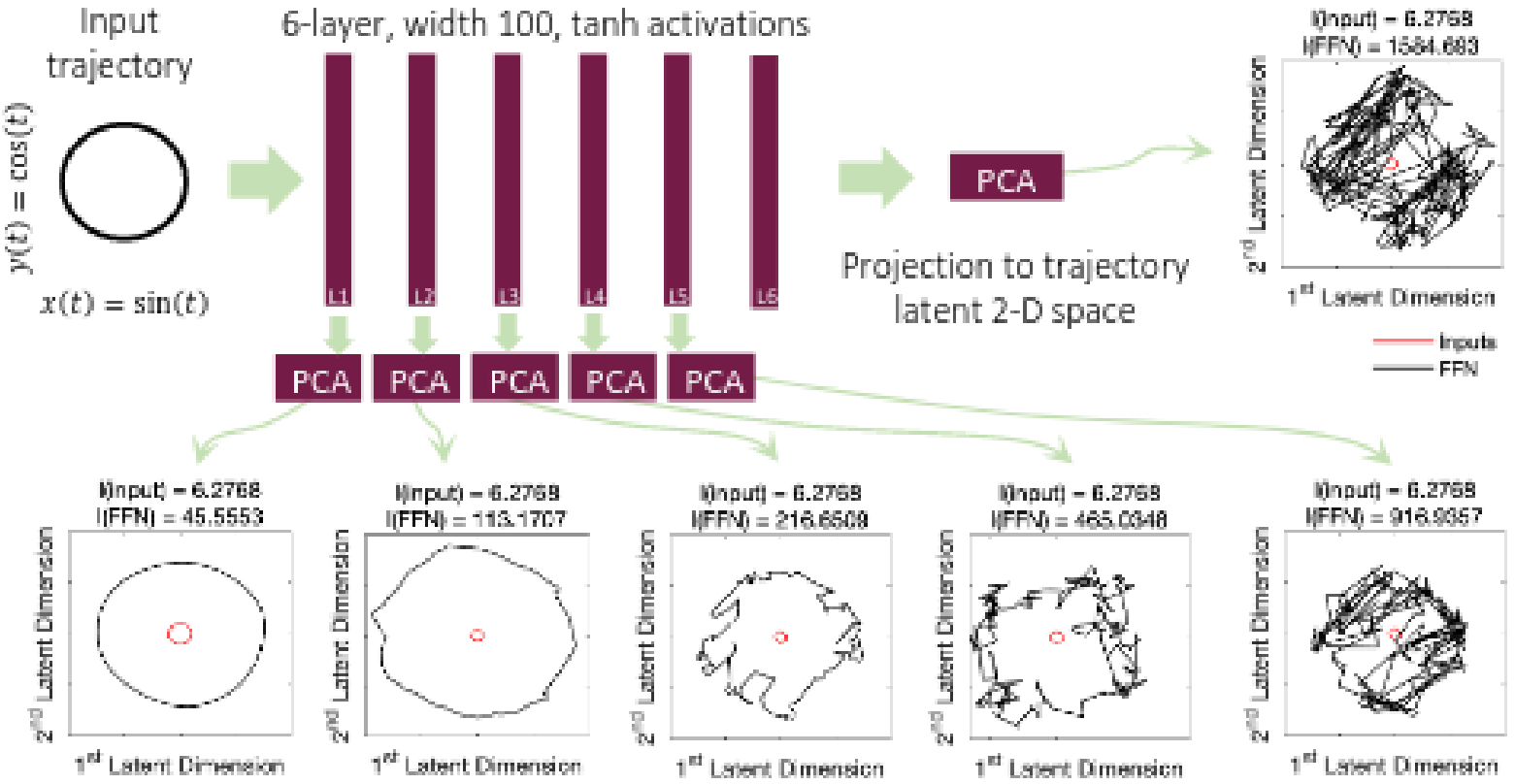
The paper references a **standard** methodology introduced in the field of Deep Learning for measuring the **expressivity** of an ANN

Said method simply relies on the monitoring of the **arc-length** of an 2D input trajectory

And as the trajectory moves across the network it becomes a **more complex** pattern and thus increases its trajectory length

A **PCA** is applied to each layer's activations and then only the **first 2 latent dimensions** are used to graph the evolution of the pattern through the network

Trajectory Length



Fused ODE Solver

Since a Numerical ODE solver has to apply **discretization steps** in order to compute the output of the hidden layer, using certain numerical methods can cause an **exponential** increase in the time-discretization steps, therefore the researchers proposed an alternative formulation which is that of **fusing** both implicit and explicit **euler methods** (hence the name 'Fused ODE Solver')

The Fused Numerical ODE solver **unrolls** a dynamical system of the form $\frac{dx}{dt} = f(x)$ into:

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

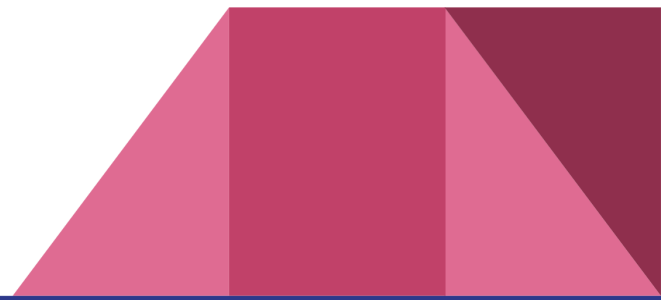
Algebraic Formulation of the Fused ODE Solver

Applying the fused solver to the previously seen LTC formulation, solving it for $x(t + \Delta t)$, we get:

- $x(t + \Delta t) = \frac{x(t)+\Delta t f(x(t),I(t),t,\theta)A}{1+\Delta t(\frac{1}{\tau}+f(x(t),I(t),t,\theta))}$

Which is an **algebraic formulation** of:

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



The LTCs' Mathematical Formulation: Justification 1

LTCs are loosely related to the computational models of neural dynamics in small species (such as the Nematode C.Elegans), put together with synaptic transmission mechanisms. The dynamics of non-spiking neurons' potential $v(t)$ an be written as a system of linear ODEs of the form:

- $\frac{dv(t)}{dt} = -g_l v(t) + S(t)$

Where:

- S is the sum of all pre-synaptic sources
- g_l is a leakage conductance

The LTCs' Mathematical Formulation: Justification 1

All synaptic currents to the cell can be approximated in steady-state by the following nonlinearity:

- $S(t) = f(v(t), I(t))(A - v(t))$

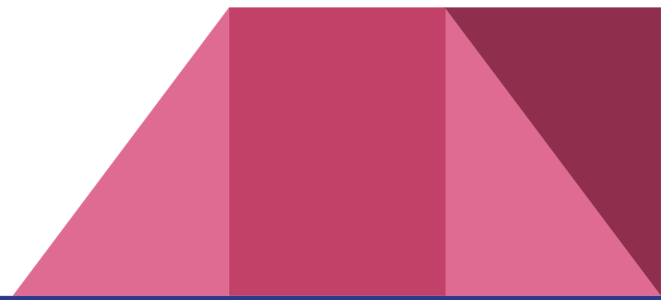
where $f(.)$ is a **sigmoidal nonlinearity** depending on the state of all neurons, $v(t)$ which are presynaptic to the current cell, and external inputs to the cell

The LTCs' Mathematical Formulation: Justification 1

By plugging in these two equations, we obtain an equation similar to equation:

$$\frac{dx(t)}{dt} = -[\frac{1}{\tau} + f(x(t), I(t), t, \theta)]x(t) + f(x(t), I(t), t, \theta)A$$

LTCs are inspired by this foundation



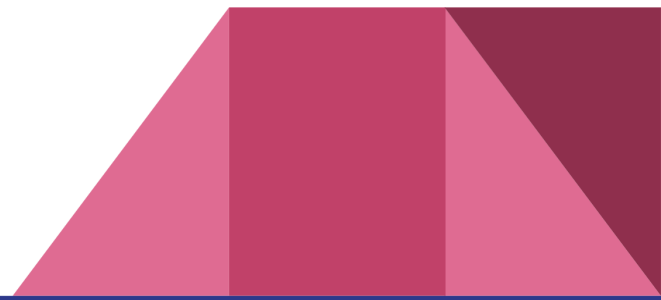
The LTCs' Mathematical Formulation: Justification 2

The previous equation might resemble that of the famous **Dynamic Causal Models** (DCMs) with a Bilinear dynamical system approximation. DCMs are formulated by taking a second-order approximation (Bilinear) of the dynamical system

DCM and bilinear dynamical systems have shown promise in learning to capture complex fMRI time-series signals. LTCs are **introduced as variants** of continuous-time (CT) models that are **loosely inspired by biology**, show great **expressivity**, **stability**, and performance in modeling time series

Historical Importance of Brain Inspired Models

This highly promising approach is not only scientifically interesting due to its inter-dsiciplinary connections but it follows the same principle as other existing architectures that came before, which is being a computational model which derives from an accurate scientific study of biological cognitive functions, (recall the experiements with the visual system of a cat that validated many hypotheses on Convolutional Nueral Networks



Contribution map of Dr. Hasani

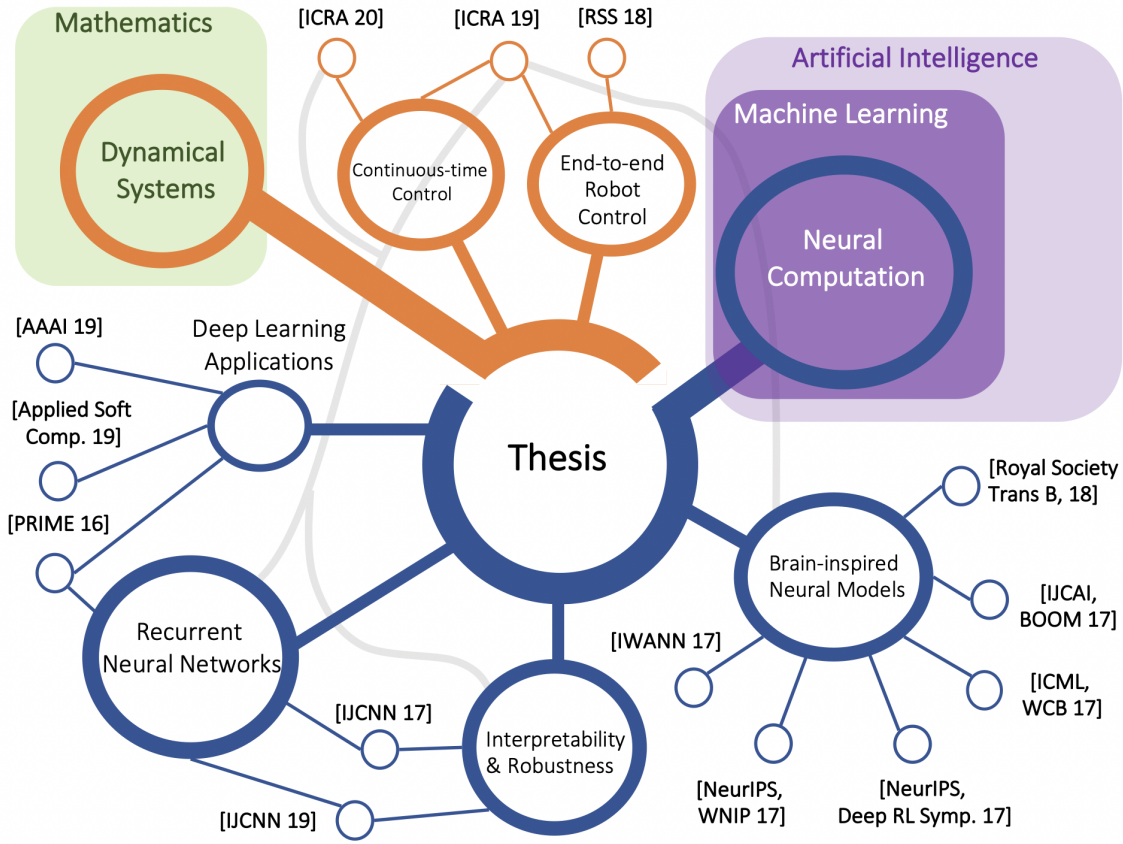


Figure 3: Dr. Hasani's PhD dissertation's contribution map

LTC Benchmarks: Time Series Predictions

LTCs have been compared to current (as of 2020) state-of-the-art models against **time-series predictions** over several different datasets

Dataset	Metric	LSTM	CT-RNN	Neural ODE	CT-GRU	LTC (ours)
Gesture	(accuracy)	64.57% ± 0.59	59.01% ± 1.22	46.97% ± 3.03	68.31% ± 1.78	**69.55%** ± **1.13**
Occupancy	(accuracy)	93.18% ± 1.66	94.54% ± 0.54	90.15% ± 1.71	91.44% ± 1.67	**94.63%** ± **0.17**
Activity recognition	(accuracy)	95.85% ± 0.29	95.73% ± 0.47	**97.26%** ± 0.10	96.16% ± 0.39	95.67% ± 0.575
Sequential MNIST	(accuracy)	**98.41%** ± 0.12	96.73% ± 0.19	97.61% ± 0.14	98.27% ± 0.14	97.57% ± 0.18
Traffic	(squared error)	0.169 ± 0.004	0.224 ± 0.008	1.512 ± 0.179	0.389 ± 0.076	**0.099** ± 0.0095
Power	(squared-error)	0.628 ± 0.003	0.742 ± 0.005	1.254 ± 0.149	**0.586** ± 0.003	0.642 ± 0.021
Ozone	(F1-score)	0.284 ± 0.025	0.236 ± 0.011	0.168 ± 0.006	0.260 ± 0.024	**0.302** ± 0.0155

LTC Benchmark: Human Activity Dataset

The Human Activity dataset contains a series of **human actions** with a period of 211 ms.

The experiments with LTCs where conducted in 2 different settings:

- First Setting: where the inputs are unchanged
- Second Setting: where the inputs where modified to match the modifications made by the authors of the dataset, in order to give a more 'fair' comparrison between the models

LTC Benchmarks: Person Activity Dataset - 1st Setting

Unchanged inputs:

Algorithm	Accuracy
LSTM	83.59%± 0.40
CT-RNN	81.54%± 0.33
Latent ODE	76.48%± 0.56
CT-GRU	85.27%± 0.39
LTC (ours)	**85.48%± 0.40**

LTC Benchmarks: Person Activity Dataset - 1nd Setting

Modified inputs:

Algorithm	**Accuracy**
RNN Δ_t *	0.797± 0.003
RNN-Decay*	0.800± 0.010
RNN GRU-D*	0.806± 0.007
RNN-VAE*	0.343± 0.040
Latent ODE (D enc.)*	0.835± 0.010
ODE-RNN *	0.829 ± 0.016
Latent ODE(C enc.)*	0.846 ± 0.013
LTC (ours)	**0.882 ± 0.005**

LTC Benchmark: Half-Cheetah Dynamics

LTCs have also been used to model the dynamics of the Half-Cheetah

17 input observations | 6 control outputs | ϕ = joint angle

Algorithm	MSE
LSTM	2.500± 0.140
CT-RNN	2.838± 0.112
Neural ODE	3.805 ± 0.313
CT-GRU	3.014± 0.134
LTC (ours)	**2.308± 0.015**

Neural Circuit Policies

It is a 'reconfiguration' of the **tap-withdrawal** (neural circuit) mechanism of the Nematode C. Elegans used for autonomous driving tasks

The Nematode's neural circuit has showed promising results in the context of **end-to-end** autonomous driving

Thus this particular architecture and task shows that by using the LTC or CfC models for the underlying **artificial neurons**, then a 'biologically accurate' **neural circuit** can be implemented and deployed

The NCP architecture is quite complex and tied to both **computational neuroscience** and **reinforcement learning**; it does however show remarkable promise for biologically plausible AI

Neural Circuit Policies

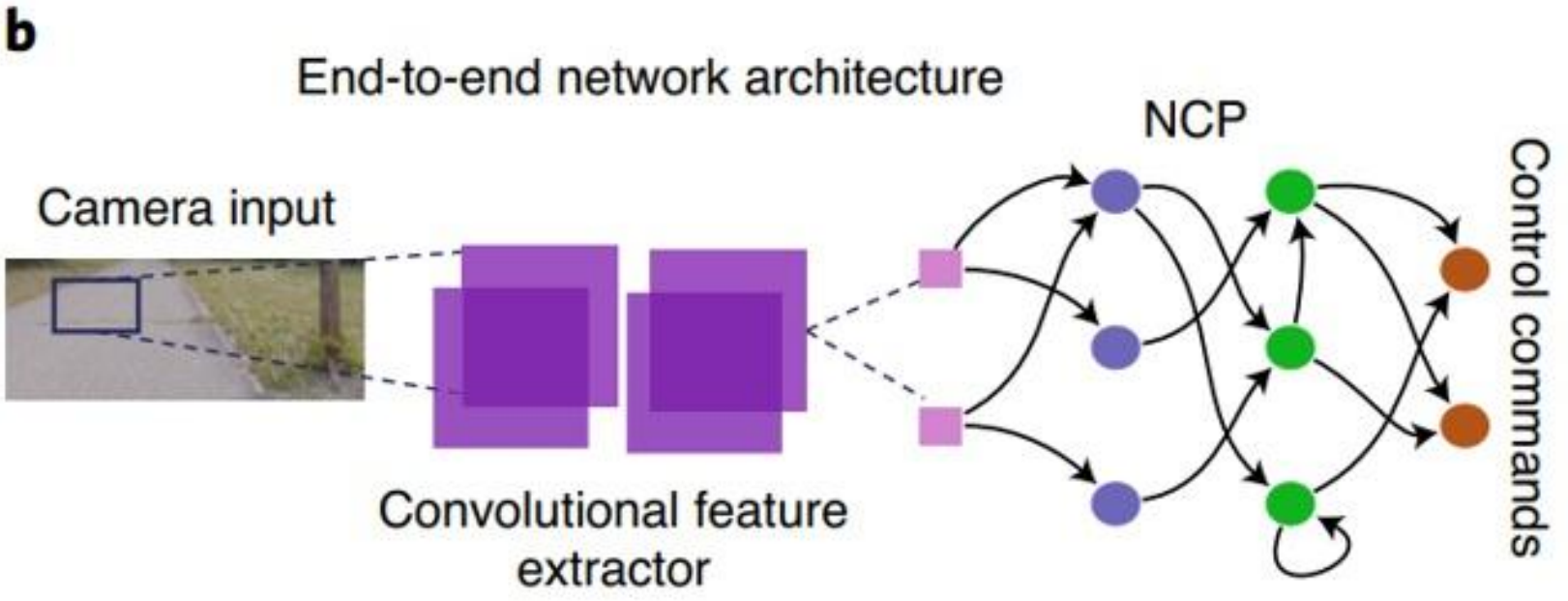


Figure 4: NCP General Architecture

Neural Circuit Policies

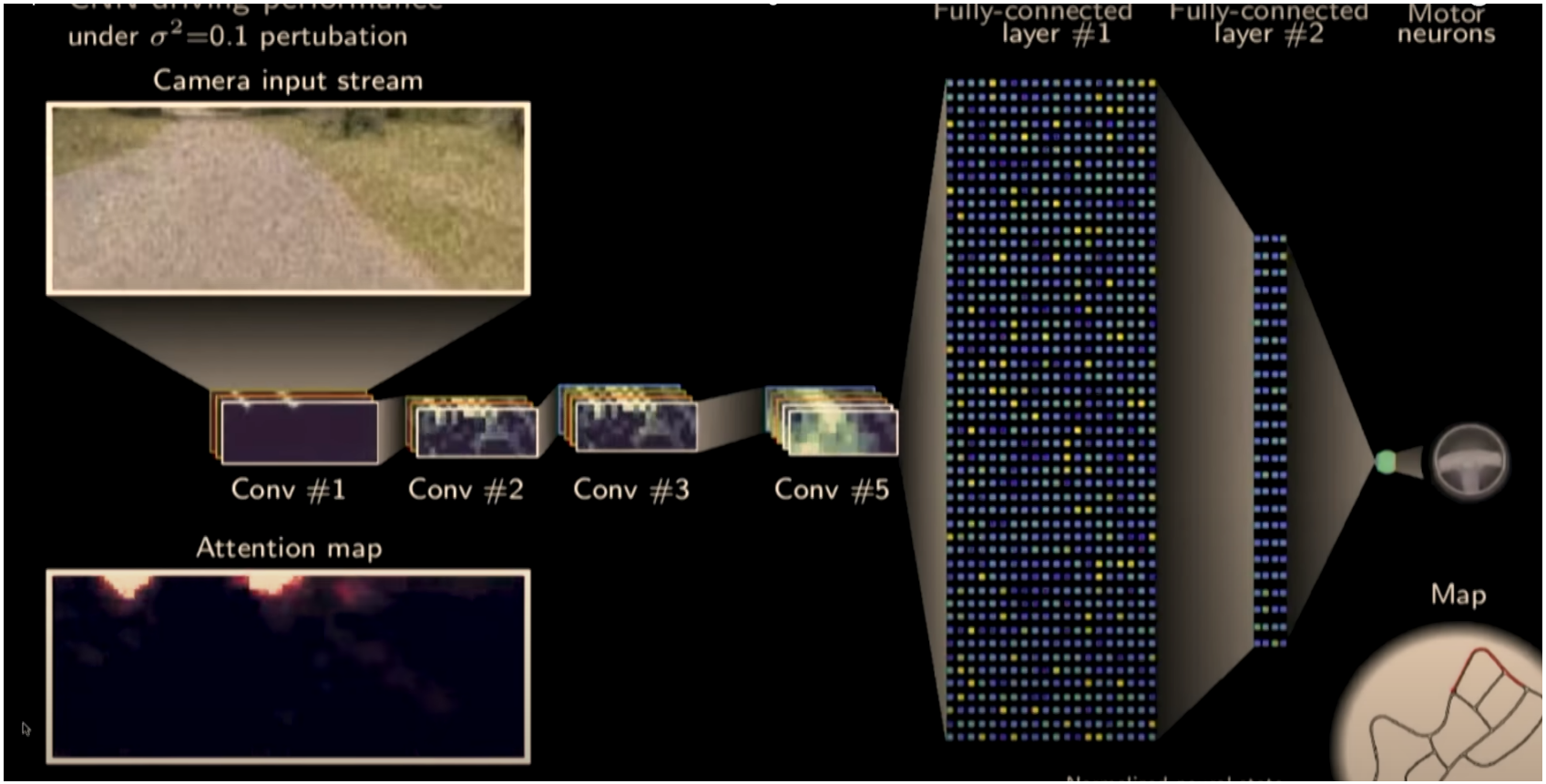


Figure 5: TEDx Talk CNN Architecture

Neural Circuit Policies

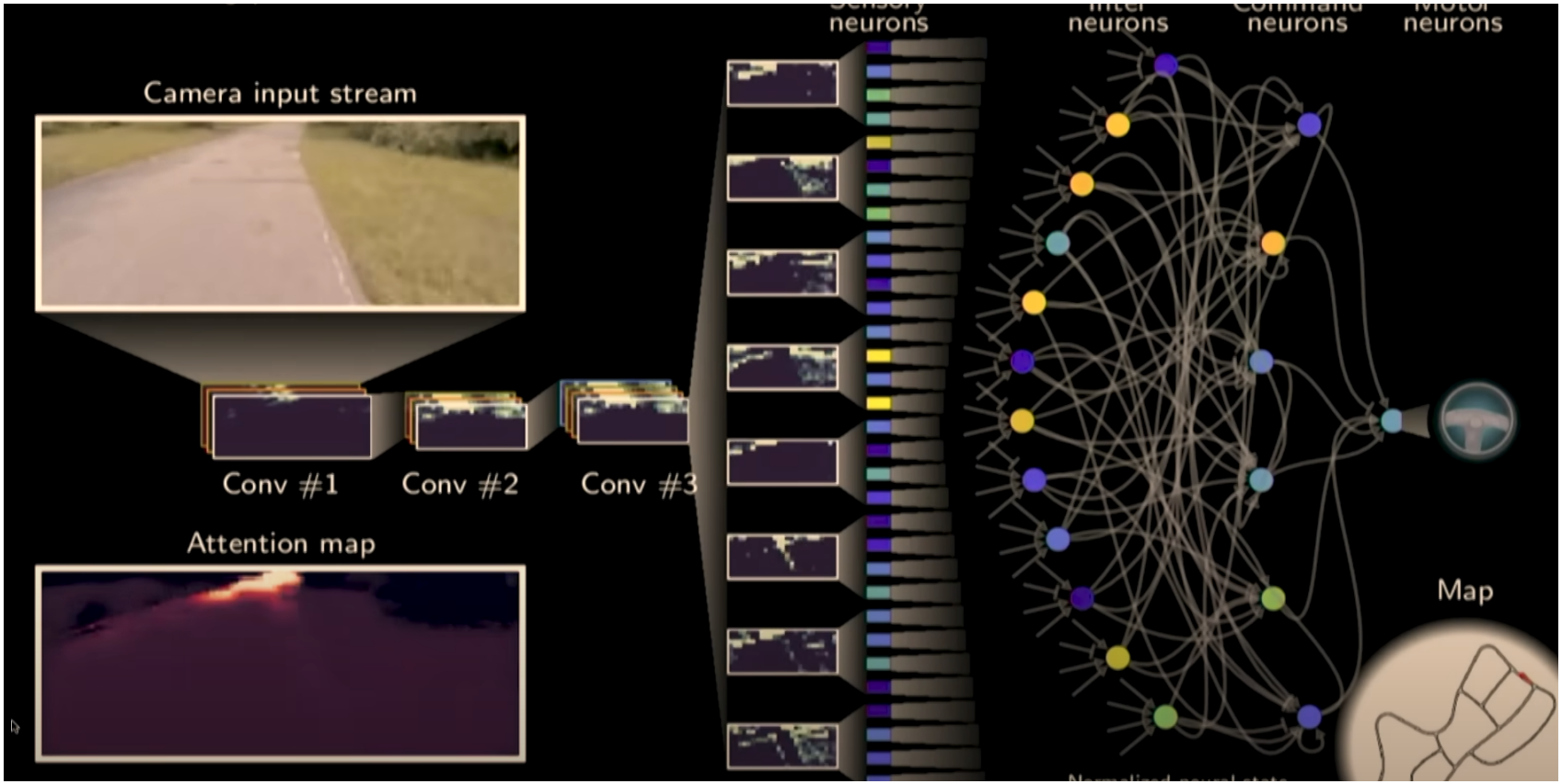
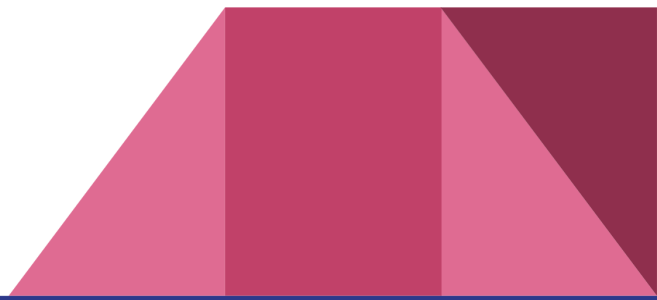


Figure 6: TEDx Talk NCP Architecture

Neural Circuit Policies

Checkout this TEDx talk on YouTube:

- Liquid Net TEDx Talk - Dr. Daniela Rus



Closed Form Continuous Time Networks

LTCs have showed very promising results with modeling complex time-series data and with handling irregularly sampled data however they have slow **inference** and **learning** speeds due to the computational overhead introduced by the ODE solver

As previously mentioned, ideally we'd solve such complex equations in **closed form**, but as we know such a solution doesn't always exist, but a **new formulation** has shown to yield **faster** inference performances whilst maintaining the **same expressivity** as its ODE-based counterpart

Said formulation is that of the **Closed Form Continuous Time Networks** (CfCs)

Closed Form Continuous Time Networks

The researchers have proven (mathematically) that given the analytical formulation of an **LTC**, an **approximate** closed-form solution is:

- $x(t) = (x_0 - A)e^{[w_\tau + f(I(t),\theta)]t} f(-I(t), \theta) + A$

It is in general possible to **compile a trained LTC model** into its closed form version, thus having high expressivity and fast inference performances

From NCPs to CfCs

The researchers have shown that their theoretical claim holds in practice by compiling a NCP model trained for a **lane keeping** task into a CfC

LTC (ODE) models can still be significantly beneficial in solving continuously defined physics problems and control tasks. Moreover, for generative modeling, continuous normalizing flows built by ODEs are the suitable choice of model as they ensure invertibility unlike CfCs. This is because differential equations guarantee invertibility

The researchers audited the performance of the 'compilation' between models by using records of actual test-runs of the deployed NCP on the **autonomous vehicle** and plugged the wieghts and parameters of the audited model (the NCP) into the CfC model obtaining a **MSE** of **0.006** by the CfC model

From NCPs to CfCs

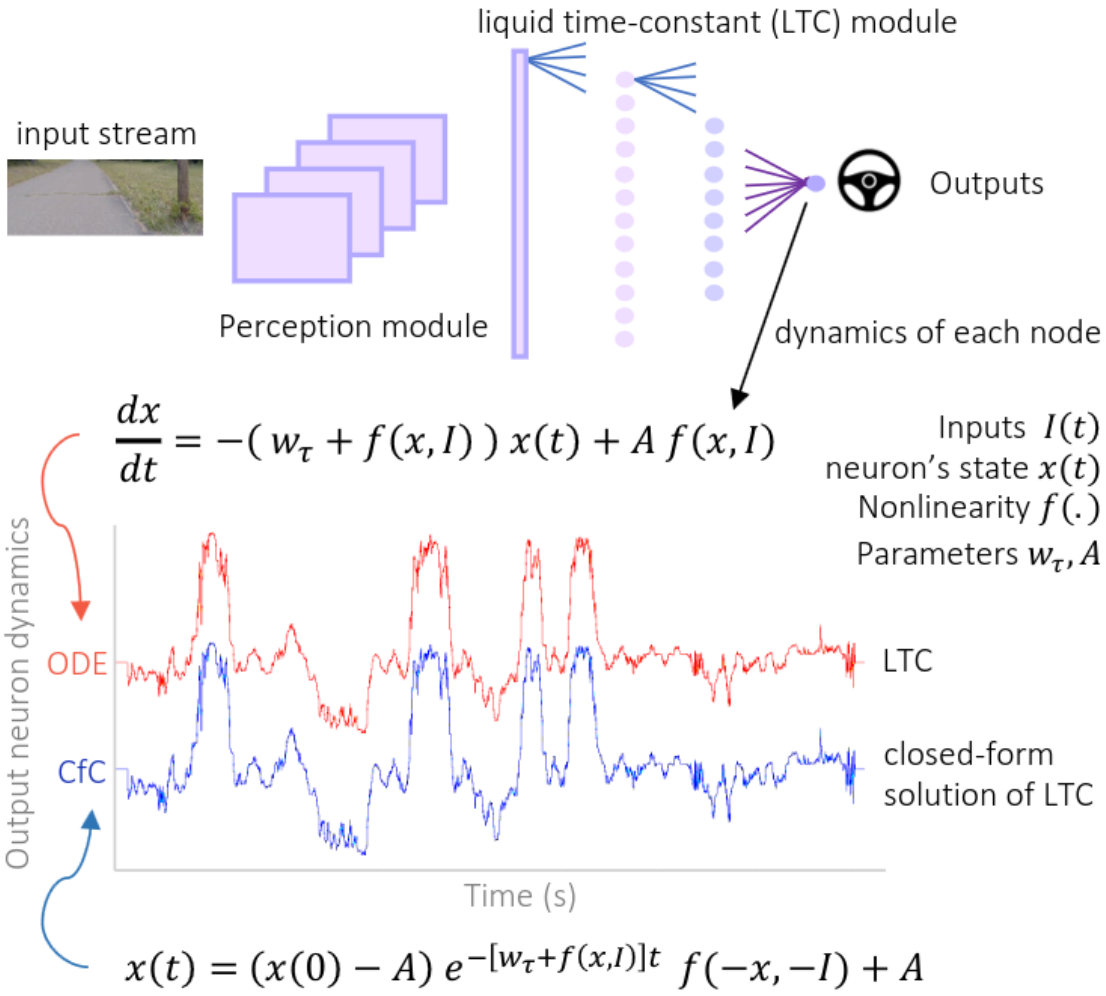
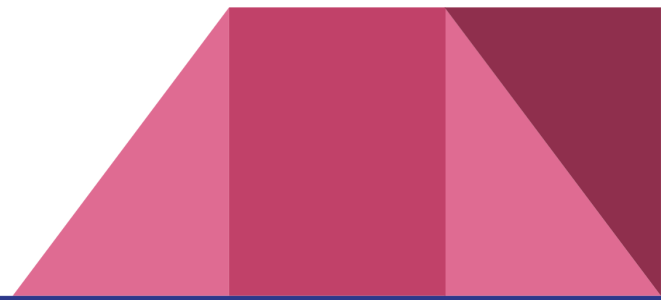


Figure 7: NCP to CfC performances

A CfC-based Neural Network Architecture

The following CfC-based architecture not only produces a **scalable** ANN but also solves the **gradient-instability** issues that LTCs suffered from thus improving the modeling capabilities for long-term temporal dependencies

Also, the CfC-based model has the **same computational complexity** as a discrete 'classical' RNN but with the provably **higher expressivity**



A CfC-based Neural Network Architecture

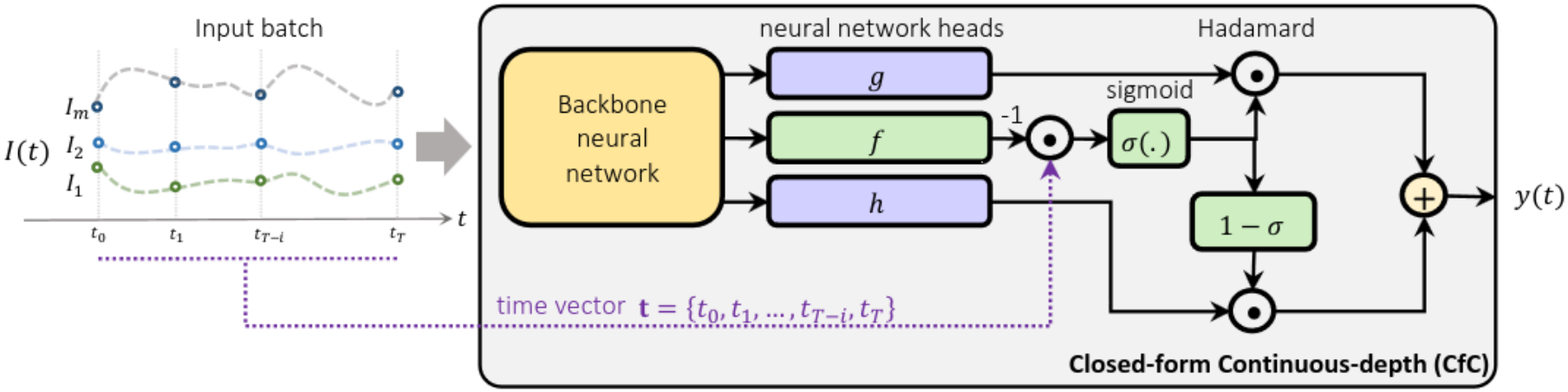


Figure 8: CfC-based Neural Architecture

CfC Benchmark: PhysioNet Challenge

The PhysioNet Challenge 2012 dataset considers the prediction of the mortality of 8000 patients admitted to the intensive care unit (ICU). The features represent time series of medical measurements of the first 48 hours after admission. The data is irregularly sampled in time, and over features, i.e., only a subset of the 37 possible features is given at each time point

CfCs perform competitively to other baselines while performing **160 times faster** training time compared to **ODE-RNNs** and **220 times** compared to **continuous latent models**. CfCs are also, on average, three times faster than advanced discretized gated recurrent models

PhysioNet Results

Model	AUC Score (%)	time per epoch (min)
†RNN-Impute (*7*)	0.764 ± 0.016	0.5
†RNN-delta-t (*7*)	0.787 ± 0.014	0.5
†RNN-Decay (*7*)	0.807 ± 0.003	0.7
†GRU-D (*36*)	0.818 ± 0.008	0.7
†Phased-LSTM (*37*)	**0.836** ± 0.003	0.3
∗IP-Nets (*31*)	0.819 ± 0.006	1.3
∗SeFT (*32*)	0.795 ± 0.015	0.5
†RNN-VAE (*7*)	0.515 ± 0.040	2.0
†ODE-RNN (*7*)	**0.833** ± 0.009	16.5
†Latent-ODE-RNN (*7*)	0.781 ± 0.018	6.7
†Latent-ODE-ODE (*7*)	0.829 ± 0.004	22.0
LTC (*1*)	0.6477 ± 0.010	0.5
Cf-S (ours)	0.643 ± 0.018	**0.1**
CfC-noGate (ours)	**0.840** ± 0.003	**0.1**
CfC (ours)	**0.839** ± 0.002	**0.1**
CfC-mmRNN (ours)	0.834 +- 0.006	**0.2**

Figure 9: PhysioNet Challenge Data

CfC Benchmark: PhysioNet Challenge

The **IMDB sentiment analysis dataset** consists of 25,000 training and 25,000 test sentences. Each sentence corresponds to either positive or negative sentiment. The sentences are tok-enized in a word-by-word fashion with a vocabulary consisting of 20,000 most frequently occurring words in the dataset. Each token is mapped to a vector using a trainable word embedding. The word embedding is initialized randomly. No pretrain- ing of the network or the word embedding is performed.

IMDB Sentiment Analysis Results

Model	Test accuracy (%)
†HiPPO-LagT (*40*)	**88.0** ± 0.2
†HiPPO-LegS (*40*)	88.0 ± 0.2
†LMU (*39*)	87.7 ± 0.1
†LSTM (*20*)	87.3 ± 0.4
†GRU (*30*)	86.2 ± n/a
∗ReLU GRU (*48*)	84.8 ± n/a
∗Skip LSTM (*49*)	86.6 ± n/a
†expRNN (*41*)	84.3 ± 0.3
†Vanilla RNN (*49*)	67.4 ± 7.7
∗coRNN (*42*)	86.7 ± 0.3
LTC (*1*)	61.8 ± 6.1
Cf-S (ours)	81.7 ± 0.5
CfC-noGate (ours)	87.5 ± 0.1
CfC (ours)	85.9 ± 0.9
CfC-mmRNN (ours)	**88.3** ± 0.1

Figure 10: IMDB Sentiment Analysis Data

CfC Benchmark: Physical Dynamics Modeling

The **Walker2D dataset** consists of kinematic simulations of the **MuJoCo** physics engine on the Walker2d-v2 OpenAI gym environment using four different stochastic policies. The objective is to predict the physics state of the next time step. The training and testing sequences are provided at irregularly-sampled intervals. The square error on the test set is the reported metric.

As shown in the following table, CfCs outperform the other baselines by a large margin rooting for their **strong capability to model irregularly sampled** physical dynamics with missing phases. It is worth mentioning that on this task, **CfCs even outperform Transformers by a considerable 18%**.

Physical Dynamics Modeling Results

Model	Square-error	Time per epoch (min)
†ODE-RNN (*7*)	1.904 ± 0.061	0.79
†CT-RNN (*33*)	1.198 ± 0.004	0.91
†Augmented LSTM (*20*)	1.065 ± 0.006	0.10
†CT-GRU (*34*)	1.172 ± 0.011	0.18
†RNN-Decay (*7*)	1.406 ± 0.005	0.16
†Bi-directional RNN (*38*)	1.071 ± 0.009	0.39
†GRU-D (*36*)	1.090 ± 0.034	0.11
†PhasedLSTM (*37*)	1.063 ± 0.010	0.25
†GRU-ODE (*7*)	1.051 ± 0.018	0.56
†CT-LSTM (*35*)	1.014 ± 0.014	0.31
†ODE-LSTM (*9*)	0.883 ± 0.014	0.29
coRNN (*42*)	3.241 ± 0.215	0.18
Lipschitz RNN (*43*)	1.781 ± 0.013	0.17
LTC (*1*)	**0.662** ± 0.013	0.78
Transformer (*46*)	0.761 ± 0.032	0.8
Cf-S (ours)	0.948 ± 0.009	0.12
CfC-noGate (ours)	**0.650** ± 0.008	0.21
CfC (ours)	**0.643** ± 0.006	0.08
CfC-mmRNN (ours)	**0.617** ± 0.006	0.34

Figure 11: Physical Dynamics Data

What about LTCs?

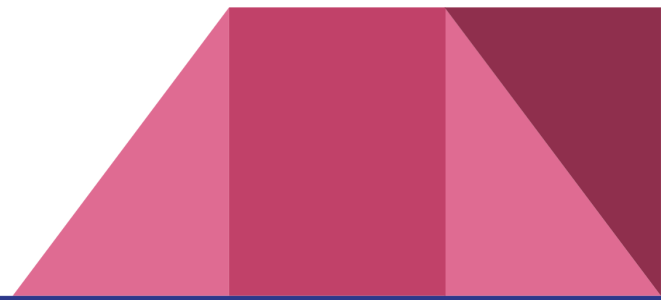
So after seeing how CfCs outperform the ODE-based models, does that mean that LTCs are already obsolete?

The answer is **no**, because implicit ODE-based models can still be significantly beneficial in solving continuously defined physics problems and control tasks. Moreover, for generative modeling, continuous normalizing flows built by ODEs are the suitable choice of model as they ensure invertibility unlike CfCs. This is because differential equations guarantee invertibility (under certain conditions) and with CfCs being an approximation of a closed form, they are **not** invertible.

Applications of CfCs

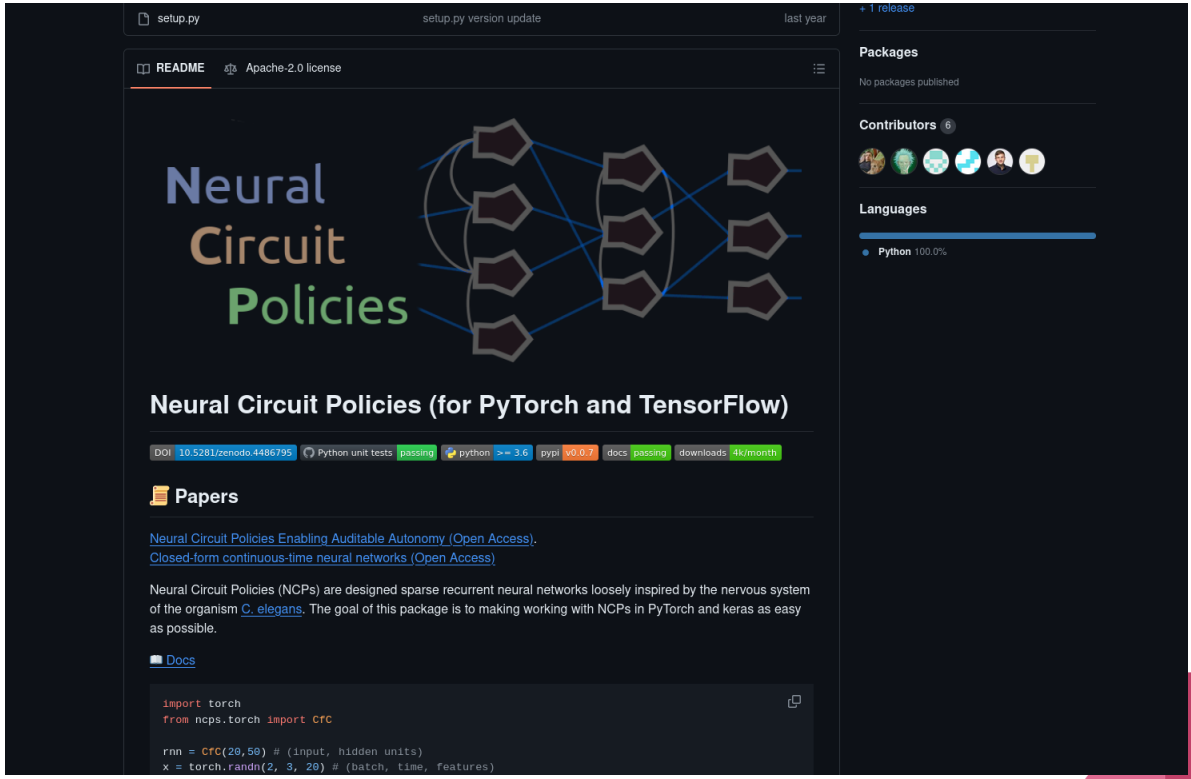
We use CfCs when:

- data has limitations and irregularities (e.g., medical data, financial time series, robotics, closed loop control and robotics and multi-agent autonomous systems in supervised and reinforcement learning schemes)
- training and inference efficiency of a model is important ((i.e. **embedded applications**)
- when interpretability matters



Python implementations

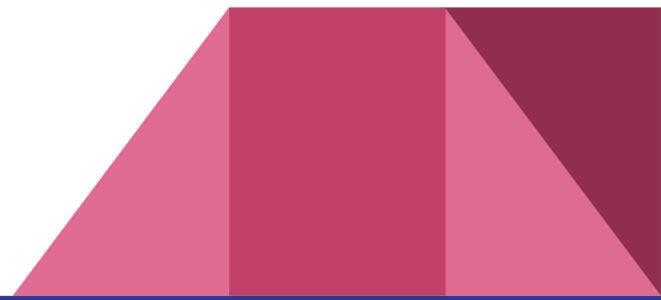
Luckily our fellow scholars have kindly implemented in the Python programming language, the studied architectures



Closing Remarks

In summary, the LTC model is a new type of technology which strongly relies on 'intricate' mathematics, from which it gains its higher expressivity

But the underlying mathematics is biologically inspired, as the original studies on artificial neurons were, however with a higher correlation to real life biological cognitive processes, bringing the field of Deep Learning and Artificial Intelligence as a whole closer to that of **computational neuroscience**



Closing Remarks

Since it has been proven that LTCs can approximate any dynamical system, thus can learn any underlying phenomena and is not only better suited than current discrete and static neural network architectures for time-series related tasks, but may replace many state of the art architectures in various fields, from that of **Computer Vision** to that of **Robotics** and so on

Sources

- Liquid Time Constant Networks
- Biological Insipration of LTCs
- Liquid State Machines
- Neural ODEs
- Dr. Hasani's PhD dissertation on LTCs
- On the Expressive Power of Deep Neural Networks
- Closed form Continuous Time Networks
- Neural Circuit Policies
- Approximation of dynamical systems by continuous time recurrent neural networks
- GitHub Page for Python implementations of the presented architectures