Liquid Neural Networks: an introduction

S.Viret
IP2I Lyon

→ **Disclaimer:**

→ Based on my personal knowledge of LNN, which is basic for the moment

→ Liquid networks are a relatively young field, things are moving fast

→ In most of the slides I tried to put some extra links providing deeper insight on the topic, which I found useful to prepare this talk. **I strongly advise to look at them if you're interested.**

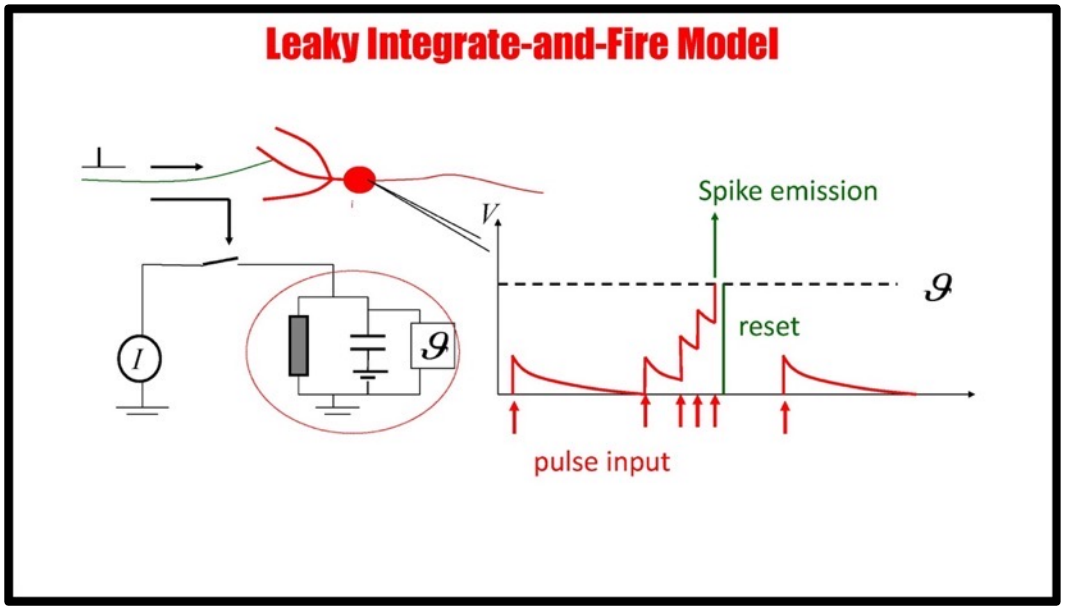
→ **Outline:**

1. Neural networks: artificial vs brain-inspired
2. Building more brain-friendly ANNs
3. Liquid neural networks
4. Next steps

1. Neural networks: artificial vs brain-inspired

→ How is a biological neuron working ?

→ Neuron acts as an RC-circuit [1]: Leaky Integrate and Fire model (*Lapicque, 1907*):



$$C\frac{dV}{dt} = -\frac{V}{R} + I(t)$$

→ ***V*** is the neuron output level, ***I*** the input activity. ***C*** and ***R*** are neuron membrane capacitance and resistance respectively.

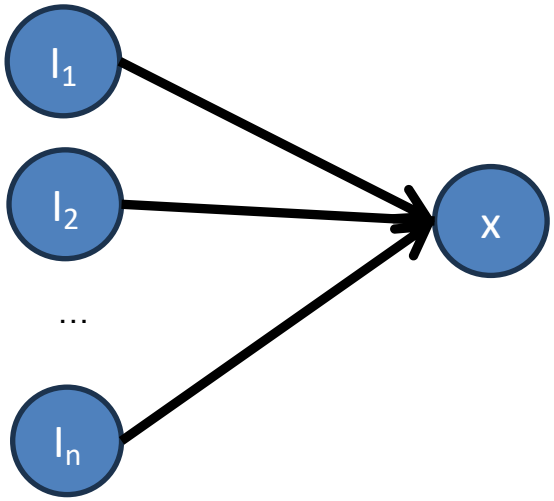
→ The neuron emit a pulse whenever ***V*** gets larger than a certain threshold. ***V*** then gets reset.

→ This is the most standard version of **Spiking Neural Networks [2],** which are particularly attractive for hardware-based approaches (*eg. using neuromorphic chips*)

[1] https://en.wikipedia.org/wiki/Biological_neuron_model
[2] https://en.wikipedia.org/wiki/Spiking_neural_network

→ How is an artificial neuron working ?

→ Well, it's slightly different **[3]**:



$$x = f(I, \theta) = \sigma\left(\sum_{1}^{n} w_i I_i + b_i\right)$$

$\theta_i = (w_i, b_i)$: the weights to be trained

σ : non-linear activation function

→ Synaptic and neuronal activity of a perceptron are far from the LIF model. They are lots of good historical/technical reasons for that (*universal approximation, gradient calculation, synchronous I/O streams,…*).

→ To put in a nutshell: artificial neural nets (**ANN**) are simpler, easier, and more adapted to computers.

[3] https://en.wikipedia.org/wiki/Perceptron

→ Why brain-inspired neural networks are important?

→ ANNs represent the vast majority of today's ML landscape. Artificial neurons [4] are getting more and more complex over the years, but the idea is roughly always the same than for the perceptron node.

→ However we know that biological neural structures are much more efficient than ANNs. Our brain uses ~20W to work, so slightly less than any GPU trying to deal with apparently simpler tasks [5].

→ The field of brain-inspired NN is currently dominated by SNNs, but SNNs are difficult to adapt for classic ML applications (*synaptic activity is the main problem*).

→ A low power brain-inspired ANN architecture based on SNN dynamics could be a serious game changer.

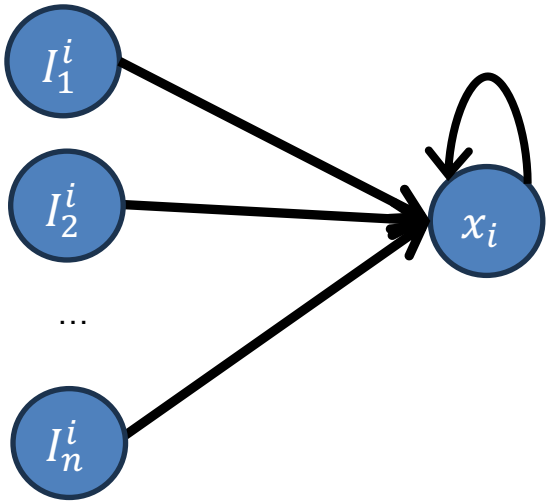
[4] https://en.wikipedia.org/wiki/Artificial_neuron
[5] https://www.humanbrainproject.eu/en/follow-hbp/news/2023/09/04/learning-brain-make-ai-more-energy-efficient/

2. Building more brain-friendly ANNs

→ Step 1: giving ANN some memory

→ Perceptron is a static network. The neural state of an hidden node depends only on learned weights and input, and is insensitive to its previous value.

→ This is not a problem for many use cases (*eg single image processing*), but clearly not optimal in some situations *(eg speech analysis, anomaly detection in sequential data,…)*



→ **Recurrent neural networks** [6] introduce some time dependencies. Hidden state at step t depends on the previous steps.

$$x_t = f(I, \theta, x) = \sigma\left(\sum_1^n w_i I_i + b_i\right) + g(x_{t-1})$$

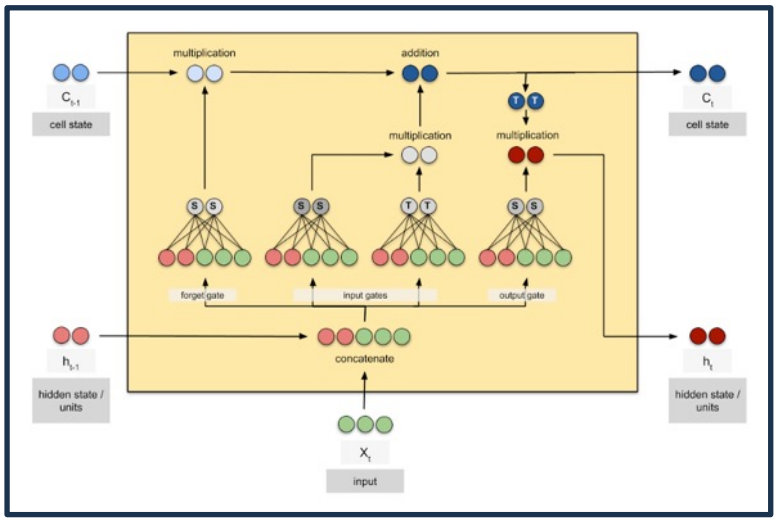
[6] https://en.wikipedia.org/wiki/Recurrent_neural_network

→ Step 2: Making RNN memory more realistic

→ The training of such networks is based on the same technique than standard ANNs: backpropagation. Just have to unfold the network in time dimension**: backpropagation trough time** (**BPTT**) [7]

→ BPTT involves a lot of gradient products, thus leading divergence problems *(Vanishing gradient* [8]***).*** ***S***tandard RNN cannot properly learn long term dependencies (*static ANNs with large number of hidden layers have the same problem BTW*).

→ Our brain is obviously solving this problem by sorting the info between what deserves to be kept (***long term***), and what is useful only for a short time scale (***short term***).



→ This is the main idea behind the **Long Short Term Memory** (LSTM) cell [9]. The hidden state **x** depends now also on memory cells **c**, which have the ability to forget only useless info along time.

$$x_t = f(I, \theta, x_{t-1}, c_{t-1})$$

[7] https://en.wikipedia.org/wiki/Backpropagation_through_time
[8] https://arxiv.org/pdf/1211.5063
[9] https://colah.github.io/posts/2015-08-Understanding-LSTMs/

→ Vanishing gradient in static ANNs:

→ Gradient problem also occurs for very deep static network. Skipping nodes was a solution proposed since quite a while, but was first used after the AI boom with **residual neural network** [10]

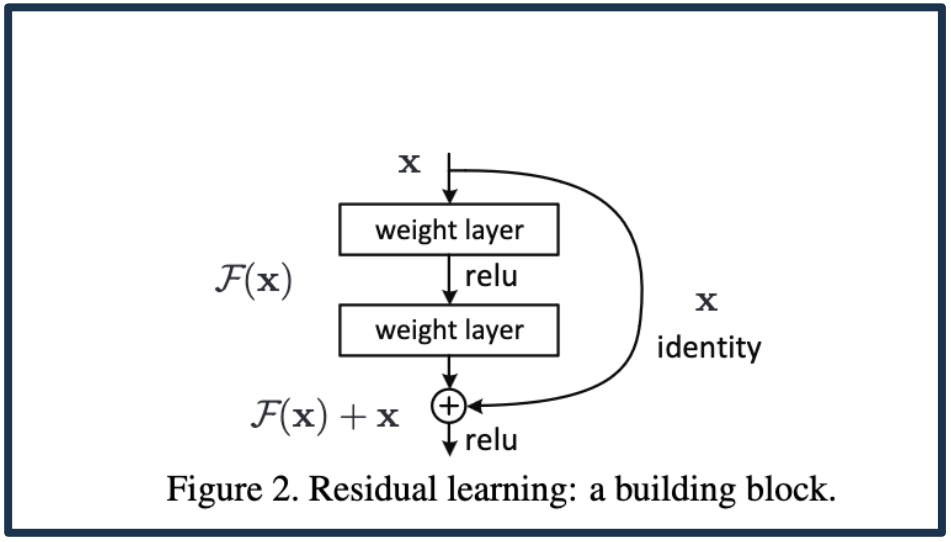


Figure 2. Residual learning: a building block.

$$x_{i+1} = x_i + \sigma\left(\sum_1^n w_i x_i + b_i\right)$$

$$x_{i+1} = x_i + f(x_i, \theta_i)$$

→ Recurrent version of the residual NN also exists [11], tough much less used than LSTM/GRU units:

$$\mathrm{x}(t+1) = f(x(t), \theta, I(t)) + x(t)$$

[10] https://arxiv.org/abs/1512.03385
[11] https://www.mdpi.com/2078-2489/9/3/56

→ From dicrete to continuous state:

→ The variation of the hidden state in a residual network share some similarities with the Euler method, which is the oldest ordinary diff. equation (**ODE**) solving technique **[12]:**

ResNet: $x(t+1) = f(I(t), \theta, x(t)) + x(t)$

Euler: $x(t+h) = h \cdot f(t, x(t)) + x(t)$

→ Led to the development of **Neural ODE network [13]:** hidden state is defined continuously as the solution of an ODE.

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

→ Proposal came with new and efficient training techniques which revived the continuous time networks field

→ Neural ODE opened the door to the modelisation of continuously varying neural activity in ANNs. Important point for the liquid networks

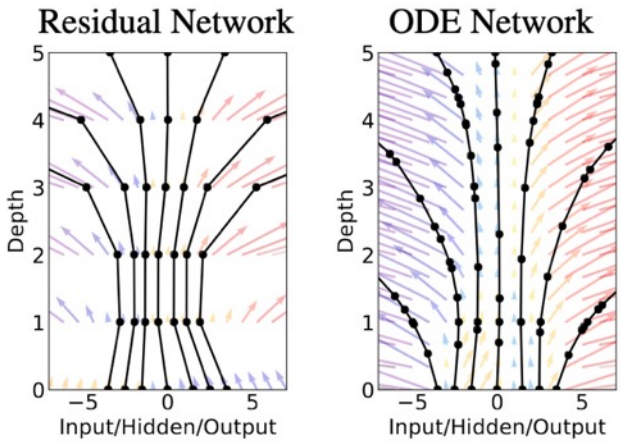


Figure 1: *Left:* A Residual network defines a discrete sequence of finite transformations. *Right:* A ODE network defines a vector field, which continuously transforms the state. *Both:* Circles represent evaluation locations.

[12] https://en.wikipedia.org/wiki/Euler_method
[13] https://arxiv.org/pdf/1806.07366

→ Step 3: continuous time and synaptic activity with ANNs

→ Neural activity is provided by an ODE:

$$C\frac{dV}{dt} = -\frac{V}{R} + S(t)$$

→ But synaptic activity (***S(t)***) is rather complex to modelize, it depends on many parameters. Current consensus in the Hodgkin-Huxley model **[14],** where we can write S as:

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

→ Where f is a gating function describing the synaptic transmission, depending on input signal and neuron potential, usually modeled by a sigmoid **[15]**.

[14] https://en.wikipedia.org/wiki/Hodgkin%E2%80%93Huxley_model
[15] https://www.tqmp.org/RegularArticles/vol13-2/p105/p105.pdf

→ Step 3: continuous time and synaptic activity with ANNs

→ Continuous time RNN were proposed initially in 1993 **[16],** with a simplified synaptic activity:

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

→ The time constant τ helps to regulate state level, preventing it to diverge. It acts as a **pseudo-forget gate**

→ Neural ODE development, in particular their RNN extension **[17],** put this model back into light. ODE RNN indeed show great expressivity, but are still prone to gradient issue because of the lack of forgeting mechanism.

→ ODE-LSTM and ODE-GRU mechanisms were proposed to sort that **[18],** but those leads to rather complex architectures and learning processes.

[16] https://doi.org/10.1016/S0893-6080(05)80125-X
[17] https://arxiv.org/pdf/1907.03907
[18] https://arxiv.org/pdf/2307.05126

3. Liquid neural networks

→ C.Elegans, an interesting case study:

→ **C.Elegans** is a small nematode, which is able to do smart things with a pretty scarce neural architecture (*only 302 neurons*) **[19]**

→ It's behaviour was extensively studied and modelized, in particular it's synaptic activity S(t)

$$S(t) = S(V,t) = f(t, I(t))(A - V(t)) \Rightarrow f(t,I) = \frac{w}{1 + e^{-\gamma(I(t)+\mu)}} = w\sigma(I)$$

→ C.Elegans neural activity can therefore be modelled as:

$$C\frac{dV}{dt} = -\frac{V(t)}{R} + w\sigma(I(t))(A - V(t))$$

→ Or:

$$\frac{dV}{dt} = -\left(\frac{1}{RC} + \frac{w}{C}\sigma(I)\right)V(t) + \frac{w}{C}A\sigma(I(t))$$

→ CT RNN with input dependent time constant…

[19] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6578605/

→ **Liquid time constant:**

→ The C.Elegans neural activity can be summarized by the following ODE:

$$\dot{x} = -\frac{x}{\tau_L(t, I, x, \theta)} + A \cdot f(x(t), I(t), \theta, t)$$

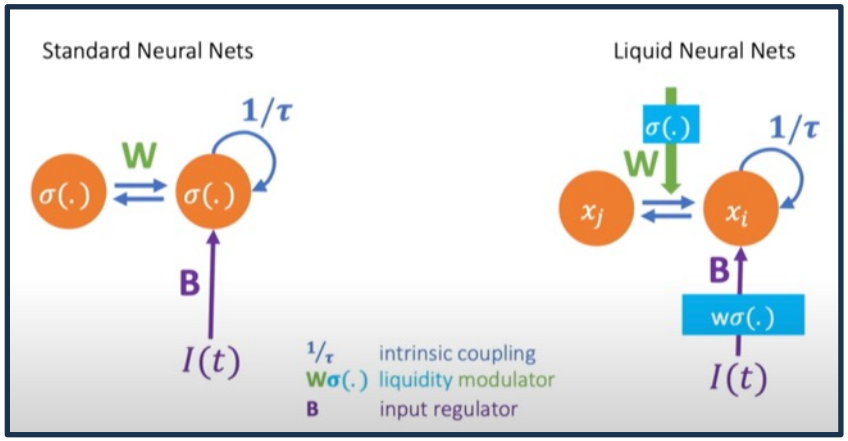
→ The time constant of the equation depends on the initial state, providing much more flexibility to the neuron than a standard CT-RNN.

→ Moreover, a network defined from this new **liquid time constant (LTC)** cell fulfills the universal approximation theorem and has its states bound. **LTC is a brain-inspired NN with ANN properties [20],[21].**

[20] https://arxiv.org/pdf/1811.00321
[21] https://repositum.tuwien.at/handle/20.500.12708/1068

→ Why are liquid neural networks attractive?

→ Many articles, videos, explains in details the pro and cons of LNN (*the best I found so far are* **[22]** ***(talk from the main LNN model author)*** *and* **[23])**



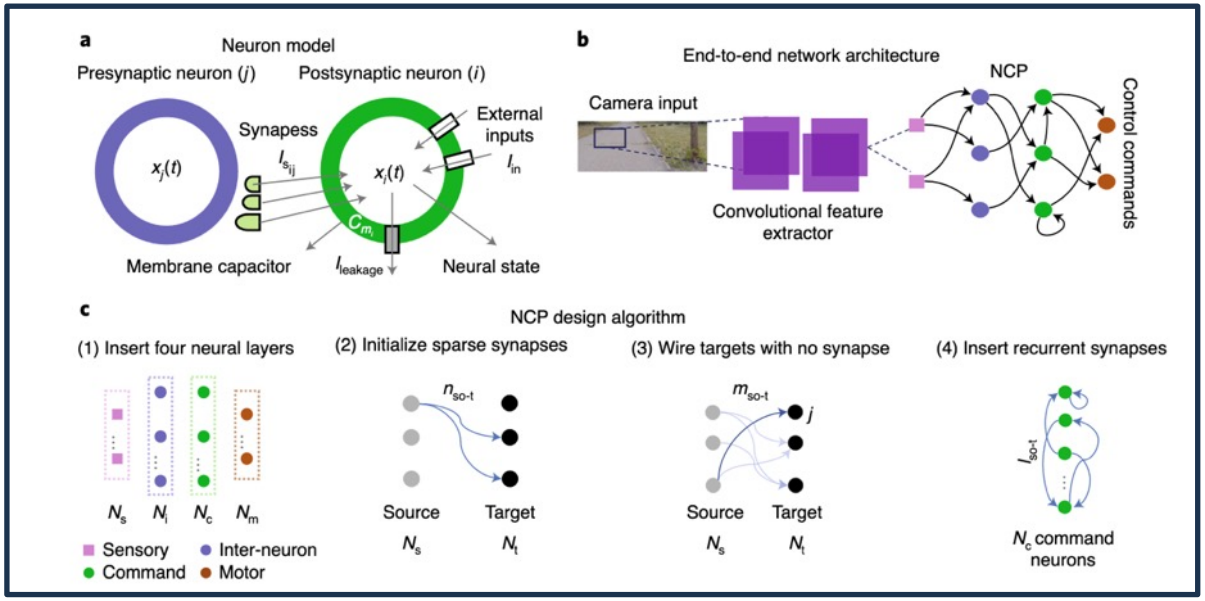
→ The take home message is that LNN, thanks to non linear synaptic activity **are much more expressive than other type of recurrent networks.**

→ As in the brain, you use the neurons more efficiently and therefore need rather more compact architectures

[22] https://www.youtube.com/watch?v=IlliqYiRhMU
[23] https://deepgram.com/learn/liquid-neural-networks

→ Why are liquid neural networks attractive?

→ Based on C.Elegans again, LNN proponents also proposed a new procedure to build small networks based on LTC cells: **Neural Circuit Policies** [23]



→ Using this algorithm and LTC-based networks they were able to get similar performance wrt state of the art RNN networks, with much less free parameters.

Table 2 | Network size comparison

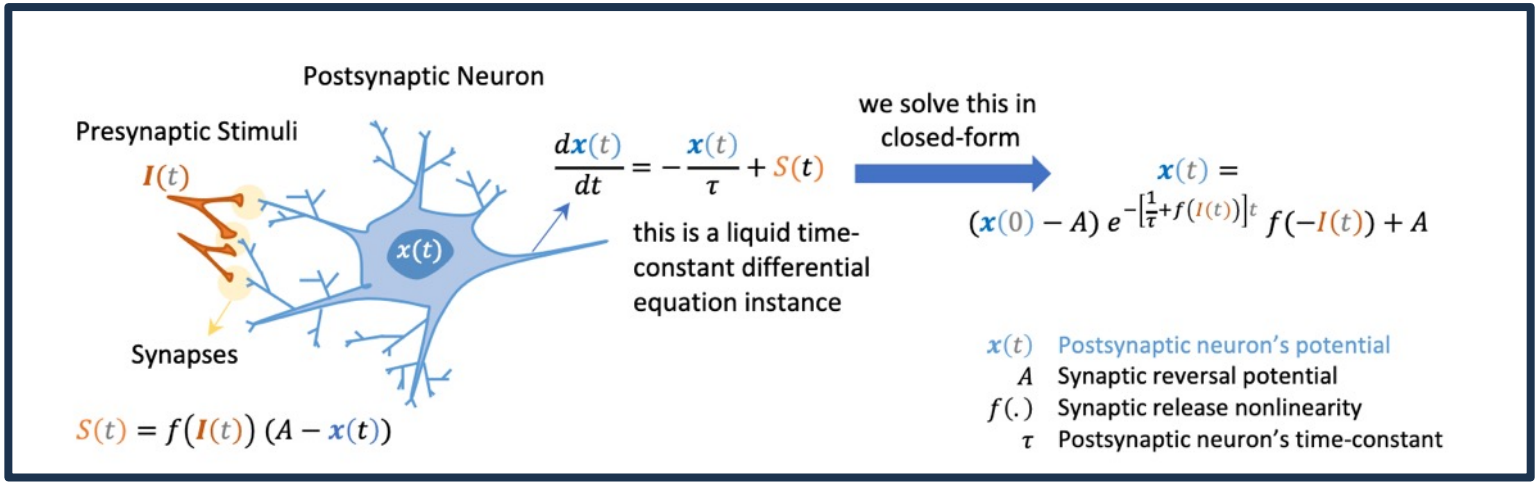
Model	Convolutional layers parameters	RNN neurons	RNN synapses	RNN trainable parameters
CNN	5,068,900	-	-	-
CT-RNN	79,420	64	6,112	6,273
LSTM	79,420	64	24,640	24,897
NCP	79,420	**19**	**253**	**1,065**

→ Code available in PyTorch and Keras, along with examples [24][25]

[23] https://publik.tuwien.ac.at/files/publik_292280.pdf
[24] https://ncps.readthedocs.io/en/latest/quickstart.html
[25] https://github.com/mlech26l/ncps/

→ Closed form solution for easier training

→ In 2021, LNN developers released a simplified version of their cell, using a closed-form approximation of the neuron potential **x(t)** **[26].**



→ Keep the continuous time behavior and non-linear synaptic stimuli, but get rid of the ODE solving part. CF cells are also available in the NCP code **[24] .** They also added the possibility to get long-term memory (***CfCmm model***).

[26] https://arxiv.org/pdf/2106.13898

→ LNN limitations:

→ LNN are can have vanishing gradient problem (*long-term dependencies*). Hybrids architectures have been proposed (*eg LTC based-state fed to an LSTM cell* **[27]).** Need further studies (*LTCmm*, *CfCmm model is a first potential solution tough*).

→ LNN are difficult to tune (*there are a lot of hyperparameters*): here also some interesting studies were conducted *(see eg **[28])***

→ The main code development is now done within a private company **[29].** I suspect that the public code available is not maintained anymore. It makes it not always easy to understand,… Not sure there will be a lot of effort in that sector…

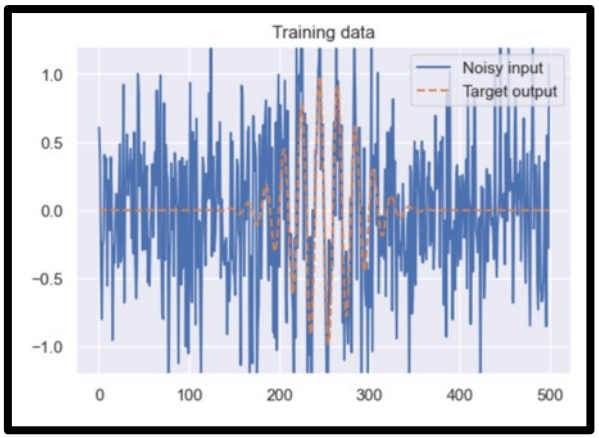
[27] *https://pub.aimind.so/combining-the-power-of-liquid-neural-networks-a-hybrid-approach-with-lstm-c360b560361a*
[28] *https://arxiv.org/abs/2304.08691*
[29] *https://www.liquid.ai/*

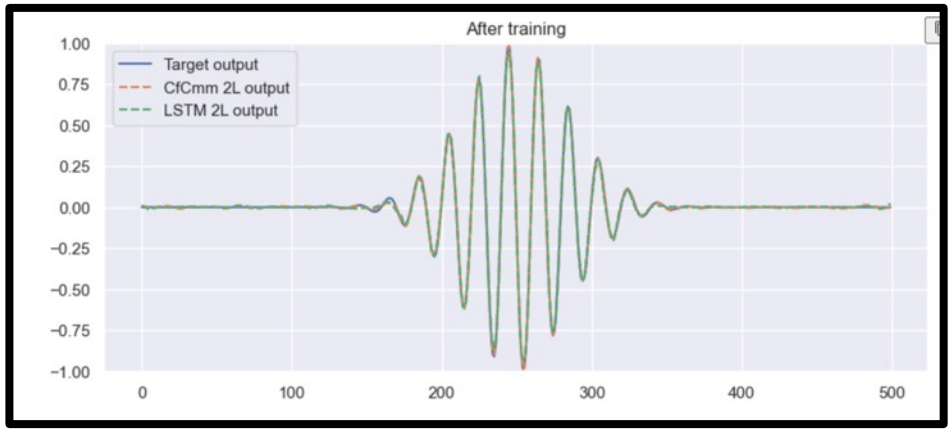
→ Playing with LNN:

→ On the **THINKII github page** you will find some 'working' examples based on the **ncps** package. Those examples are in Keras:

- *Simple example with LTC*
- *Simple example with CfC*
- *Denoising example*

→ Good starting point to see how to implement those cells into your prefered network, but keep in mind there are many hyperparameters requiring tuning here, this is the really complicated part, as usual





https://gitlab.in2p3.fr/think2/models/-/blob/main/LNNs/README.md

4. Next steps

→ **Liquid neural net on hardware platforms:**

→ A simplified version of the liquid network named **LTC-SE** was recently presented **[28],** claimed to have the same level of perf.

→ The code **[31]** looks indeed much simpler, looks more like an LIF neural model, but didn't had too much time to look into it.

→ The idea behind this simplification is to facilitate usage of LTC networks on hardware platform. Found at least one paper where this approach has been implemented on a neuromorphic chip (**LoiHi2**) **[32] .** Fast training technique have also been proposed (see eg **[30])**

→ If confirmed this could open a large domain of possibilities. To be followed up…

[30] *https://arxiv.org/pdf/2112.11231*
[31] *https://github.com/michaelkhany/liquid_time_constant_networks/tree/main*
[32] *https://arxiv.org/pdf/2407.20590*

Conclusion

→ **Brain inspired neural networks and artificial neural networks are conceptually and technically very different**

→ **Bringing together both fields has been a long standing quest over the past decades**

→ **Liquid neural networks could mark the beginning of the end for this quest. They are the first 'brain-inspired RNNs'.**

→ **Basic LNN are complex objects with a lot of hyperparams, but new simplers models, more adapted to an hardware implementation, start to appear.**

→ **Coming years will be important: as the gradient, this field will either explode or vanish**