Liquid Time-Constant Networks

Paul Jason Mello
Department of Computer Science and Engineering
University of Nevada, Reno
pmello@unr.edu

Abstract

Liquid Time-Constant Networks (LTCNs) are an extended class of Continuous-Time Recurrent Neural Networks (CTRNNs) with adaptive temporal dynamics. In LTCNs, the liquid mechanism is implemented through a "leaky" gate which helps modulate the response to the input. LTCNs improve over their CTRNN counterparts by allowing neurons to operate across multiple timescales simultaneously within each layer and control forgetfulness of the models gates. This allows for stable, bounded behavior, with increased expressivity resulting in complex modeling capabilities. In this work, we will explore the core dynamical system which make LTCNs work and elucidate their inner mechanisms. I provide a simple comparative evaluation against Kolmogorov-Arnold Networks (KANs) and the Kalman Filter (KF) to give a general understanding of their effectiveness. The code for this work can be found here: `https://github.com/pauljmello/Liquid-Time-Constant-Networks`.

1 Summary

This work explores the implementation details of LTCNs with the goal of understanding the fundamental leakiness mechanisms. While these networks have shown impressive capabilities which exceed their parental relatives, namely RNNs and LSTMs, LTCNs also have their own set of tradeoffs. Despite this, LTCNs exhibit adaptive and expressive modeling abilities making them particularly useful for time-series forecasting, especially in regimes where robust temporal modeling is necessary. The liquid nature of these networks positions them as a powerful tool for real-time applications in a diverse array of domains.

2 Introduction

Many natural processes can be described by differential equations including robotic control, weather forecasting, and fluid dynamics. CTRNNs have modeled these processes accurately by interpreting each hidden state as a first order ordinary differential equation. However, due to their underlying mathematics, they suffer from forced compromises between capturing fast shifts in data and long time-horizon memory. Previously, other works have been proposed to handle these tradeoffs including Time-Aware Multi-Scale Recurrent Neural Network [1] and more prominently, Long Short-Term Memory (LSTMs) [3]. While these networks have been shown to minimize these tradeoffs, they still require significant parameter costs with no explicit guarantees on the bounded behavior of the model.

LTCNs, as proposed by Hasani et al. [2], handle the previously described tradeoffs by *learning the leak itself*. Each neuron effectively has its own decay rate which allows for complex temporal dynamics to emerge within each layer and neuron. Similar to how weight decay helps improve generalization through inducing a form of regularization, the leakiness induces forgetting over time. Through this learned leakiness and non-linear gating, the LTC cells become a bilinear ODE in the following form: $dx(t)/dt = -x(t)/\tau + S(t)$ where, $S(t) \in \mathbb{R}^M$ and non-linearity is introduced

through $S(t) = f(x(t), I(t), t, \theta)(A - x(t))$. The parameters are determined by θ and A resulting in the hidden state ODE in equation 1:

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

Here, the forgetting gate function is explicitly defined as:

$$f(\mathbf{h}(t), \mathbf{I}(t), t, \theta) = \tanh(\mathbf{h}(t)\boldsymbol{\gamma}_r^\top + \mathbf{I}(t)\boldsymbol{\gamma}^\top + \boldsymbol{\mu}) \tag{2}$$

with learnable parameters $\theta = \{\boldsymbol{\tau}, \boldsymbol{\gamma}_r, \boldsymbol{\gamma}, \boldsymbol{\mu}, \mathbf{A}\}$.

In Algorithm 1, I illustrate the training dynamics of my LTCN implementation which slightly varies from the traditional model with a fused ODE solver, but the fundamentals the same. The learned leakiness mechanism operates through two components that form the adaptive leakiness: $\boldsymbol{\alpha}(t) = \boldsymbol{\tau}^{-1} + \mathbf{g}(t)$ where $\boldsymbol{\tau}^{-1}$ represents learned base time constants and $\mathbf{g}(t) = f(x(t), I(t), t, \theta)$ is the contextual gating function described previously in Equation 2.

This allows learned leakiness and contextual adaptation, where a larger τ enables longer memory and a lower τ corresponds to quicker information leakage. Additionally, for directness,

Algorithm 1 Training a Liquid Time-Constant Network (LTCN)

Require: Dataset $\mathcal{D} = \{(\mathbf{I}_{1:T}, \mathbf{y}_{1:T})\}$, time step Δt, inner steps L, learning rate η, epochs E
Ensure: Optimized parameters $\boldsymbol{\theta} = \{\boldsymbol{\tau}, \boldsymbol{\gamma}, \boldsymbol{\gamma}_r, \boldsymbol{\mu}, \mathbf{A}, \mathbf{W}, \mathbf{b}\}$

for $e \leftarrow 1$ **to** E **do**
  for all minibatch $(\mathbf{I}_{1:T}, \mathbf{y}_{1:T}) \in \mathcal{D}$ **do**
    $\mathbf{h}_0 \leftarrow \mathbf{0}$
    for $t \leftarrow 1$ **to** T **do**
      $\mathbf{h} \leftarrow \mathbf{h}_{t-1}$
      for $s = 1$ **to** L **do**
        $\mathbf{g} \leftarrow \tanh(\mathbf{h}\boldsymbol{\gamma}_r^\top + \mathbf{I}_t\boldsymbol{\gamma}^\top + \boldsymbol{\mu})$
        $\mathbf{h} \leftarrow (\mathbf{h} + \frac{\Delta t}{L}(\mathbf{g} \odot \mathbf{A})) \odot (\mathbf{1} + \frac{\Delta t}{L}(\boldsymbol{\tau}^{-1} + \mathbf{g}))^{-1}$
      end for
      $\hat{\mathbf{y}}_t \leftarrow \mathbf{W}\mathbf{h} + \mathbf{b}$
    end for
    $\mathcal{L} \leftarrow \frac{1}{T}\sum_{t=1}^{T} \|\hat{\mathbf{y}}_t - \mathbf{y}_t\|^2$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}$
  end for
end for

This processes modulates any given neurons ability to preserve global boundedness. This property has been particularly useful in reducing the number of parameters that are necessary to fit complex time-dependent trajectories when compared to other networks like spiking neural networks. This has lead to liquid net processes such as LTCNs being introduced to spiking neural networks [5].

3 Methodology

To benchmark these methods, I constructed a simplistic dataset of dampened sine waves under the equation $s(t) = sin(t)e^{-0.1t}$. I generated 1000 evenly spaced samples between $[0, 10]$ and utilized a sliding window with a length of 100. Two network types were trained, namely LTCNs and KANs, along with the error correcting KF. KF was added to help justify my desire to update my own KF code repository found here: `https://github.com/pauljmello/Kalman-Filter`. I train these for 250 epochs at a batch size of 256, with the AdamW optimizer [4] with a learning rate of 0.01. Training is done using a $70 - 30$ training-validation split respectively. Below I illustrate the brief differences between each models architecture:

Table 1: Model Configurations

Model	Core size	Key hyper-params	Regularisation	#Params
LTCN	32 liquid neurons	τ=5, Δt=0.01, 10 sub-steps	–	1,185
KAN	layers $[1, 8, 8, 1]$	grid = 5, cubic splines	ℓ_1=10^{-4}	1,280
Kalman	3-state oscillator	identity H matrix	fixed after analytic init.	22

4 Results

The results of this work demonstrate one of the key benefits of LTCNs, the reduction of overfitting. This is evident from the LTCN having the same loss trajectory for both training and validation as shown in Figure 1.

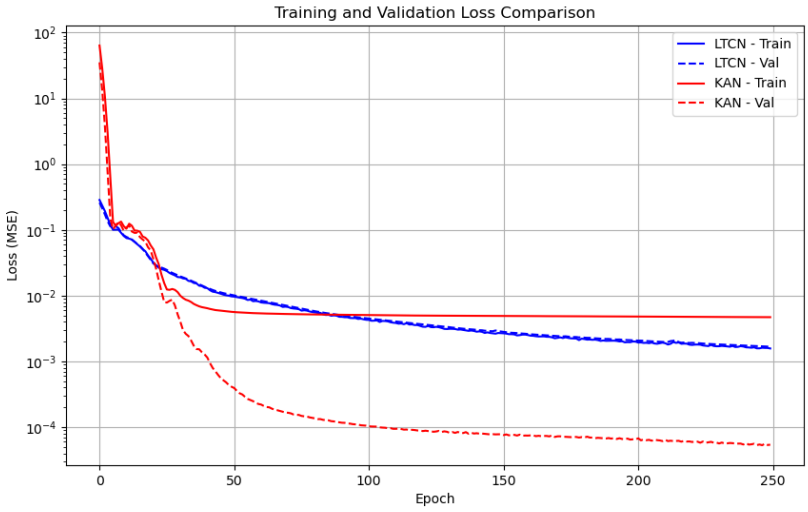

Figure 1: Loss comparison between LTCNs and KANs on fitting a dampened sine wave after 250 epochs of training 1185 and 1280 parameters respectively.

This is unsurprising given the dataset, as KANs are better equipped to model fixed problem sets by attenuating its B-splines. LTCN's however, excel in a slightly different data space because of their internal "leakiness". This liquid property allows LTCNs to better model long time-horizons because they have explicit control over their forgetfulness. While this dataset demonstrates a small fixed problem, LTCNs come close to the effectiveness of KANs, while being designed for significantly longer data streams.

As shown in Figure 2, KAN is able to trace the dampened sine wave near perfectly, while LTCNs quickly learn to model the ground truth function precisely. Simultaneously, KF tends to underestimate the signal in the beginning stages before converging towards near optimal predictions. This is likely due to the linear dynamics of the model struggling to model the non-linear dynamics of the dataset.

5 Discussion

LTCNs, KANs, and KF all have different tradeoffs, applications, and benefits. In general, the choice of model is critical when attempting to model any given data. In this case, I oversimplify the modeling task, simply to illustrate comparisons between each model type. KANs excel here due to their static mapping that enables smooth target approximation through B-splines, while LTCNs utilize a unique decay rate per-neuron to enable a multi-scale approximation. KF on the other hand uses a very lightweight system to approximate the underlying system dynamics through corrective be-

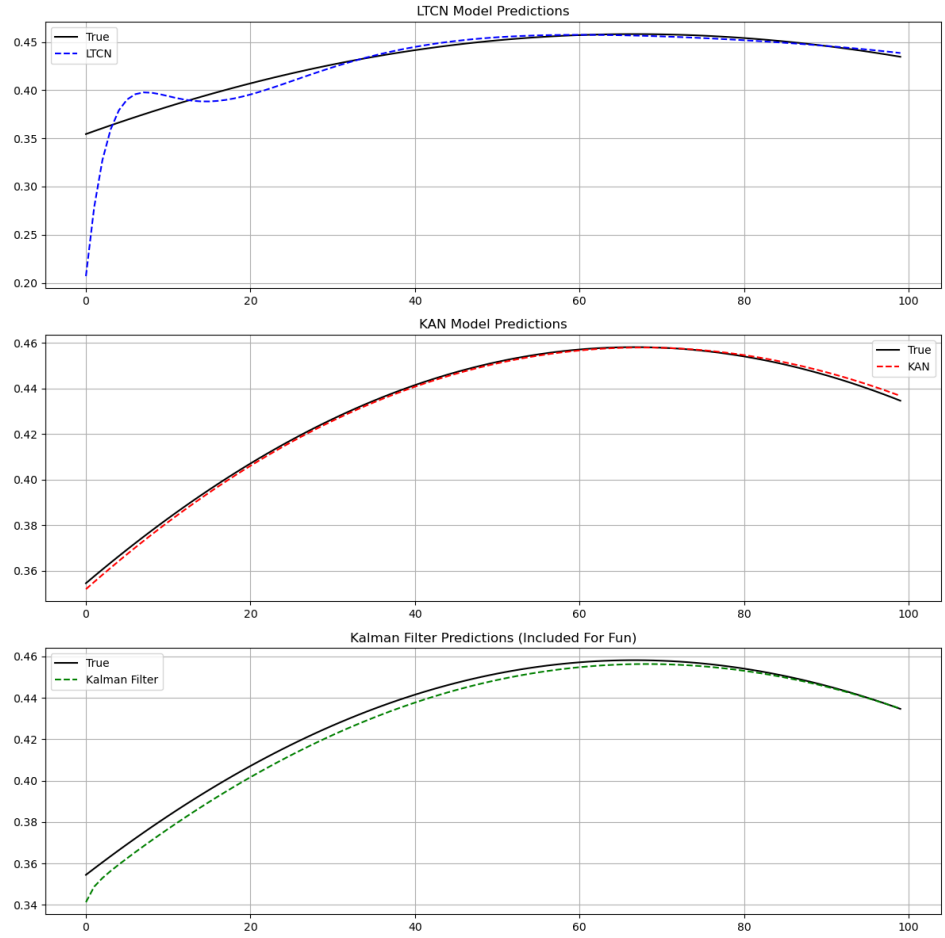

Figure 2: Final model prediction comparison on dampened sine wave data using a 100 step time-horizon. Notice how LTCNs quickly adapt to the function over a small period of time, while the other models are immediately precise. The first few epochs represents a warmup phase.

havior and is fundamentally not a probabilistic network method like LTCNs or KANs. Ultimately, LTCNs provide a strong and competitive alternative to other models and are best used when the time-horizon is long, data may be noisy or shift phases, and reducing overfitting is critical.

6 Conclusion

LTCNs form an incredibly rich and expressive networking architecture designed for long time-horizon modeling due to their fundamental architecture being derived from RNNs. LTCNs fundamentally control their own information leakage, providing a more flexible structure to the fixed gating structures of RNNs. Our experiments demonstrate the effectiveness of LTCNs in their modeling capabilities as they approach the efficiency of KANs in short function approximation windows. While KANs offer better performance when comparing accuracy-per-flop, the neuron specific time-constants encapsulate the stationary dynamics effectively with very few parameters. In real world scenarios, where data is not so smooth and may be noisy or shifting phases, the adaptability of LTCNs will prove significantly more useful. As I continue to develop various model types and expand my understanding of differing network architectures, it is always pleasant to see that the gating mechanisms of RNNs are so fundamental to networks and so effective for modeling.

References

[1] Zipeng Chen, Qianli Ma, and Zhenxi Lin. Time-aware multi-scale rnns for time series modeling. 08 2021.

[2] Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. Liquid time-constant networks, 2020.

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[5] Marzieh Hassanshahi Varposhti, Mahyar Shahsavari, and Marcel van Gerven. Energy-efficient spiking recurrent neural network for gesture recognition on embedded gpus, 2024.