HIGH-PERFORMANCE RNNS WITH SPIKING NEURONS

Anonymous authors
Paper under double-blind review

ABSTRACT

The increasing need for compact and low-power computing solutions for machine learning applications has triggered a renaissance in the study of energy-efficient neural network accelerators. In particular, in-memory computing neuromorphic architectures have started to receive substantial attention from both academia and industry. However, most of these architectures rely on spiking neural networks, which typically perform poorly compared to their non-spiking counterparts in terms of accuracy. In this paper, we propose a new adaptive spiking neuron model that can also be abstracted as a low-pass filter. This abstraction enables faster and better training of spiking networks using back-propagation, without simulating spikes. We show that this model dramatically improves the inference performance of a recurrent neural network and validate it with three complex spatio-temporal learning tasks: the temporal addition task, the temporal copying task, and a spoken-phrase recognition task. Application of these results will lead to the development of highly-accurate spiking models for ultra-low-power neuromorphic hardware used in edge-computing applications.

1 INTRODUCTION

Exponential growth in computational power and efficiency have played a vital role in the development of neural networks and their training algorithms. However, it has also led to higher design complexity and increasing difficulty to keep up with Moore's law (Schaller, 1997; Waldrop, 2016). Recent years have also seen the movement of computation from data-centres to compact, distributed, and portable embedded systems. These factors have created a demand for energy-efficient AI-capable devices, leading to the development of dedicated and optimized von Neumann-style Artificial Neural Network (ANN) accelerators (Aimar et al., 2018; Cavigelli and Benini, 2016; Chen et al., 2016) and a renewed interest in alternative brain-inspired, event-based neuromorphic systems (Chicca et al., 2014; Frenkel et al., 2019; Davies et al., 2018; Moradi et al., 2018; Akopyan et al., 2015; Qiao et al., 2015; Neckar et al., 2019). In this paper, we focus on designing neural network models for mixed-signal neuromorphic chips that on one hand can achieve high energy-efficiency and on the other maintain the high accuracy performance of von Neumann-style ANN accelerators (see supplementary section D).

A key difference between neural networks deployed on Graphical Processing Units (GPUs) and on most neuromorphic platforms is the use of spikes or train of pulses to represent signals. Such networks are called Spiking Neural Networks (SNNs). Neuromorphic systems use SNNs to implement in-memory computing systems that lower energy consumption (Payvand et al., 2019; Chi et al., 2016; Ahn et al., 2015). However, such systems have a number of drawbacks that inhibit their growth and applicability to real-world problems: (1) mixed-signal neuromorphic systems comprise analogue circuits that suffer from Complementary Metal-Oxide-Semiconductor (CMOS) mismatch effects (Pelgrom et al., 1989) which can substantially degrade performance; (2) rate-based SNN models need to generate a large number of spikes to represent signals, therefore reducing their low-power benefits; (3) the spiking non-linearity SNNs makes it very difficult to train them using gradient-descent methods like back-propagation (or Backprop).

In this paper, we describe a new neuron model that addresses the problems of coding efficiency and noise tolerance in mixed-signal spiking neuromorphic chips and discuss how its Low-Pass Filter (LPF) abstraction enables training spiking Recurrent Neural Networks (RNNs) using Backprop. This is a significant breakthrough as it enables training and deployment of energy-efficient spiking neural network devices (Davies et al., 2018; Moradi et al., 2018; Furber and Furber, 1996)).

2 THE SPIKING NEURON MODEL

Several spiking neuron models have been proposed for efficiently encoding signals, that typically use rate- or time-coded spike-generation schemes (Diehl et al., 2015; Rueckauer et al., 2017; Bohte, 2012; Mostafa, 2017). In rate-coding, the firing rate of the neuron is proportional to the input signal. Therefore, achieving high data-resolution with rate-coding requires a large number of spikes, which is not energy-efficient (Nair and Indiveri, 2019). On the other hand, time-coding codecs require complex circuit designs, especially on hardware that is subject to mismatch and noise. The solution proposed in this work is the use of an Adaptive Integrate and Fire (aI&F) neuron model, which can also be interpreted as an asynchronous $\Sigma\Delta$ circuit (Nair and Indiveri, 2019; Bohte, 2012; Yoon, 2016). This mechanism reduces the spike count by encoding the error between an internal state and the input. The aI&F neuron model implemented with current-mode neuromorphic circuits (Nair and Indiveri, 2019) can be described by the following equations:

$$\tau_{mem}\frac{dI_{mem}}{dt} = \alpha_L(I_L - I_{mem}) - s + i \tag{1}$$

$$\tau_w\frac{ds}{dt} = \alpha_s(I_{mem} - I_L) - s \tag{2}$$

$$I_{mem} = 0, \quad \text{when } I_{mem} > \Delta \tag{3}$$

where the currents I_{mem} and I_L represent the "membrane potential" and "leak reversal potential" variables. The term s represents the neuron adaptation current, i the input current, τ_{mem} the membrane time constant, α_L a gain factor, Δ the threshold, α_s the adaptation coupling parameter and τ_w is the adaptation time constant. The aI&F model is a feedback loop that tries to decrease the difference between the $i(t)$ and $s(t)$. The difference, $i(t) - s(t)$, is filtered with gain, α_L, and time constant, τ_{mem}. When the output of this filter, I_{mem}, exceeds the spiking threshold, Δ, I_{mem} is reset and a spike is generated. The $\Sigma\Delta$ circuit model used in this work is different from the aI&F model in the computation of the feedback term. Instead of filtering I_{mem}, we operate on the spike train generated by the spiking neuron. This ensures that the noise inserted by the spike-generation mechanism is also suppressed by the $\Sigma\Delta$ feedback. The modified feedback equation is as follows:

$$\tau_k\frac{ds}{dt} = \alpha_s(\delta_i I_{in} - I_L) - s \tag{4}$$

Where, δ_i indicates the spike train and I_{in} is a programmable maximum current value that the analogue filter implementation can generate. The product $\delta_i I_{in}$ models the feedback filter that integrates a current I_{in} for the duration of the spikes.

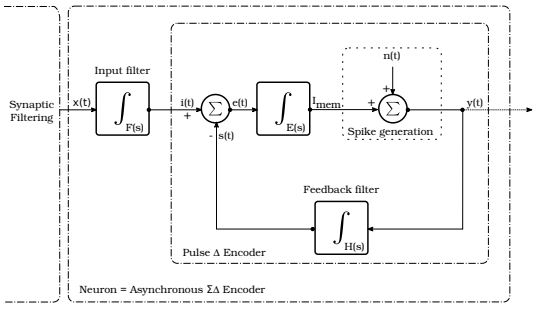


Figure 1: Block diagram of a $\Sigma\Delta$ neuron circuit. The block marked pulse Δ encoder is similar to the aI&F neuron model. The LPF stage at the input makes it an asynchronous $\Sigma\Delta$ loop.

Figure 1 shows a block diagram of the circuit implementation of the Equation 3 with the modification described by Equation 4. In this diagram, $F(s)$ is a first order LPF that receives inputs to the neuron. $H(s)$ is a first-order low-pass filter that produces $s(t)$ in response to spikes generated by the neuron. $E(s)$ is also a first-order low-pass filter on the difference between the input current $i(t)$ and the feedback signal $s(t)$. When the output of $E(s)$, I_{mem}, exceeds the spiking threshold (Δ) of the neuron, a spike-event is produced. With each spike-event, $s(t)$ increases and $i(t) - s(t)$ decreases. It can be shown that the Laplace domain representation of the output spike train can be expressed by

the equation:

$$Y(s) = \frac{X(s)F(s)E(s)}{1 + H(s)E(s)} + \frac{N(s)}{1 + H(s)E(s)} \tag{5}$$

where, $N(s)$ indicates the Laplace-domain representation of the noise introduced into the loop, for example by the spike generation mechanism. The $\Sigma\Delta$ loop described by Equation 5 is similar to a continuous-time Δ modulation loop (Pavan et al., 2017) with the key difference being that the output spikes are unipolar.

The input signals to the neuron may be encoded as spikes trains or as continuous analogue values from a sensor. The transmitted analogue signal is reconstructed from a spike train by simply low-pass filtering it. An advantage of the $\Sigma\Delta$ neuron model is that it comes with a low-pass filter in its input stage. Therefore, a $\Sigma\Delta$ neuron is a "codec" - It can both encode an analogue signal into a spike train and decode an incoming spike train back to the transmitted analogue signal. As the low-pass filter at the input stage is agnostic of the type of input to it, a $\Sigma\Delta$ can encode and decode both types of signals - spike trains or continuous analogue ones. A description of the biological motivation and noise-filtering properties of the model is provided in supplementary section A. The circuit implementing this model has been fully characterized in Nair and Indiveri (2019).

3 Training a spiking RNN without simulating spikes

The neuron model introduced in the previous section allows us to train a recurrent SNN by treating the spiking neurons as LPFs and modifying the recurrent ANN equations suitably. We will show that the trained weight parameters of the recurrent ANN model can be mapped to a recurrent SNN, without additional training. We measure the effectiveness of this mapping procedure by comparing the temporal dynamics of the neurons in the recurrent ANN to the low-pass filtered spike trains generated by the spiking neurons in the corresponding recurrent SNN. In this demonstration, we use high precision synaptic weights. This is typically not available in most spiking neuromorphic platforms. However, the same mapping procedure can be used for mapping ANNs trained with binary or noisy weights. Before introducing the mapping procedure, we describe three operations that are needed for it.

Input re-scaling: When implementing an SNN in mixed-signal neuromorphic systems, the state variables of the neuron are represented by voltages or currents that are of the order of mV or nA. Using such small values when training an ANN in software may lead to computational instability. To avoid this we train the network with normalized input signals and re-scale the parameters and activation functions after training. For example, if a single layer calculation is represented as $y = \sigma_{nl}(W \cdot x)$, where σ_{nl} is a non-linear activation function, x, W and y are the inputs, weights, and outputs from the layer, respectively, then, to re-scale the inputs by a factor γ, the activation function used in the ANN will be modified, during inference, as $y = \overline{\sigma_{nl}}(W \cdot \gamma \cdot x)$, where, $\overline{\sigma_{nl}}(.) = \sigma_{nl}\left(\frac{.}{\gamma}\right)$

Low-pass filtering: The $\Sigma\Delta$ neuron tracks the input if its time constants, τ_{mem} and τ_w, are much smaller than $\frac{1}{\text{Input Bandwidth}}$. When the input signal is being tracked accurately, it implies that it can be reconstructed from the spikes generated by the neuron. The feedback filter with the time-constant, τ_w, reconstructs the signal from the spike train inside the $\Sigma\Delta$ loop. Therefore, if we assume that the neuron is tracking its input, we can model the neuron as an LPF. This is an approximation that ignores the high-frequency components injected by the spiking mechanism, as they are suppressed by feedback loop (as evident from the $N(s)$ term in the transfer function of $\Sigma\Delta$ neuron, Equation 5). Note that Equation 3 filters the difference between I_{mem} and i. Therefore, if the bandwidth of $E(s)$ is much larger than the other filters in the loop, then sufficiently band-limited input signals will pass through a first-order LPF. We model this by using a discrete-time Euler approximation to incorporate a LPF-term at the output of the RNN stage. This results in the Low-Pass Recurrent Neural Network (lpRNN) cell by a simple tweak to the classical equation:

$$y_t = \alpha \odot y_{t-1} + (1 - \alpha) \odot \sigma(W_{rec} \cdot y_{t-1} + W_{in} \cdot x_t + b) \tag{6}$$

where, σ, $\odot$ and $\cdot$ denote non-linearity, element-wise Hadamard product and matrix multiplication functions, respectively. The variables α, y_n, x_n, W_{rec}, W_{in}, and b represent the retention ratio vector, input vector, output vector, recurrent connectivity weight matrix, input connectivity weight matrix,

and biases, respectively. The subscripts on variable y and x indicate the time step. α models the time constant of the recurrent ANN, and it is matched to the SNN time constant by setting it to $\alpha = e^{\frac{-Ts}{\tau_s}}$ where, Ts is the time-step of the input data-stream fed to the recurrent ANN, and τ_s is the feedback time constant of the $\Sigma\Delta$ neurons used in the desiredSNN.

For example, if the recurrent ANN is being trained to detect speech from an audio-signal, then Ts should be set equal to the time difference between the consecutive samples. The value of τ_s, for the ANN set to $min(\tau_w, \tau_{mem})$ in the $\Sigma\Delta$ equations. τ_s must therefore be chosen such that the signals being transmitted lie well within the pass-band of the feedback filter. This ensures that all the in-band components are transmitted well, even when different neurons in the systems have different values of τ_s, for example, due to mismatch. This is an important observation for mixed-signal systems, where mismatch effects may result in different neurons to have differing time-constants. We will demonstrate that the effect of device mismatch is well-tolerated for most practical cases and leads to gradual degradation in performance as it increases. It must be noted that while computing Ts or the bandwidth of an audio or sensor measurement is easy, it is not trivial for other types of data-sets not directly related to sensory signals, such as text.

Saturating non-linearity: It has already been shown in the literature that the Rectified Linear Unit (ReLU) non-linearity is a good non-linear model of the aI&F neuron (Yoon, 2016; Bohte, 2012). However, the $\Sigma\Delta$ neuron also filters incoming spike trains using a low-pass filter which limits its maximum output current to I_{in} (see Equation 4). To model this effect we can either set I_{in} in the SNN to the largest activation output found in the ANN simulation or clamp the maximum output of the activation function in the ANN simulation to I_{in}. In our experiments, we do the latter.

3.1 THE MAPPING PROCEDURE

First, the recurrent ANN cell is modified by replacing the RNN units with the lpRNN. The modified network is then trained using Backprop with conventional Autograd tools provided by libraries such as PyTorch or Tensorflow. This gives us the synaptic weights for the recurrent SNN. Then, the largest value attained by the state variables in the trained network is mapped to I_{in}. This ensures that the spiking neurons do not saturate. Finally, the inputs to the SNN are re-scaled to suitable currents or voltage values as described earlier.

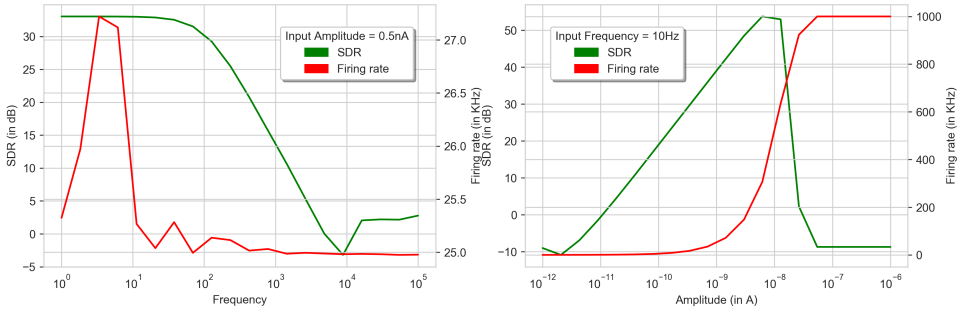
Limitation: A recurrent SNN is a continuous-time system that spikes hundreds to thousands of times per second to achieve the necessary transmission accuracy. Therefore, the time step of the transient simulation needs to be made very fine. The mapping algorithm assumes that the mapped SNN operates on the same sequence that the original ANN is trained on. This is a problem. Training a recurrent ANN on a long sequence using back-propagation is computationally expensive and often intractable because of vanishing and exploding gradients. For example, with speech signals sampled at the standard rate of 44.1 kHz, even a short utterance is thousands of samples long. If we are unable to train an ANN for the desired task, the mapping mechanism is useless. Our approach to addressing this issue is to train the ANN with sub-sampled signals. After we compute the desired weights, we rescale the time-constants of the network before mapping it to the SNN. If the simulation time-step for the SNN is $T_{s_{SNN}}$ and that of ANN is $T_{s_{ANN}}$, then the time constants of the two simulations are given by $\alpha_{ANN} = e^{-\frac{T_{s_{ANN}}}{\tau}}$ and $\alpha_{SNN} = e^{-\frac{T_{s_{SNN}}}{\tau}}$. We only rescale the time constants without changing the weights of the mapped network, introducing inaccuracies in the mapped network. The mismatch arises because a single time-step of the ANN corresponds to several simulation time-steps in the mapped SNN ($= \frac{T_{s_{ANN}}}{T_{s_{SNN}}}$). The mapping is exact for a first-order LPF because of the Linear Time-Invariant (LTI) property. However, even though an RNN is non-linear, by making the low-pass filtering effect more dominant (for example, with $\alpha = 0.99$), we observe that the mapped dynamics match well.

4 EXPERIMENTAL RESULTS

All the spiking simulations described in the following sections are run using a custom spiking simulator for systems of 1st-order LPFs (Spiker). The motivation for design and operation of Spiker is described in supplementary section E.

4.1 ENCODING PERFORMANCE OF THE SIGMA–DELTA NEURON MODEL

The $\Sigma\Delta$ neuron model used in the mapped SNN is not an ideal transmitter of information as the spiking mechanism introduces error akin to quantization noise in Analog to Digital Converters (ADCs). We measure this using a Signal-to-Distortion ratio (SDR) metric when encoding a sinusoidal input and reconstructing it with an LPF. The SDR is the ratio between the energy contained in the transmitted signal and total energy in all other frequency components generated by the distortions introduced in the signal chain. The results of these experiments are shown in Figure 2. We note in Figure 2a that highest SDR of the $\Sigma\Delta$ neuron is 55dB, with a 20 dB/decade roll-off as a function of frequency with a pole corresponding to τ_{mem}. Figure 2b highlights the input amplitude-dependence of the SDR. We note in Figure 2b that the SDR improves as a logarithmic function of the input amplitude and then drops suddenly. The logarithmic improvement in SDR is because the error component corresponding to the spiking threshold, Δ, becomes a smaller fraction of the input amplitude. The sudden drop occurs when the input amplitude approaches and exceeds I_{in} in Equation 4. This is because the maximum attainable value of the feedback term, $s(t)$ in Equation 4 is I_{in}. Under these conditions, I_{mem} is always greater than $s(t)$, causing the neuron to fire at a very high rate. For the rest of the SNN simulations in this paper, the $\Sigma\Delta$ neuron settings listed in Figure 2 caption are used.



(a) SDR as a function of the frequency of the input sinusoid.

(b) SDR as a function of the amplitude of the input sinusoid.

Figure 2: Signal to distortion ratio of the $\Sigma\Delta$ neuron model used in the SNN as a function of a sinusoidal input (a) Frequency and (b) Amplitude. The data is collected by feeding the $\Sigma\Delta$ model with a single-tone sinusoidal input riding on a DC bias that ensures that the input signal is always non-negative. The transient simulations are run with a simulation time step of 1μs. This is the reason for the saturation in the firing rate in (b). The SDR ratio is reported after subtracting the DC component. The neuron parameters for the simulation are $\tau_{mem} = 0.007s$, $\tau_w = 0.0014s$, $\alpha_L = 5000$, $\alpha_s = 1$, $\Delta = 0.1nA$, $I_{in} = 40nA$, $I_L = 0nA$.

4.2 DEMONSTRATION OF THE MAPPING MECHANISM

We demonstrate the mapping mechanism using multi-layer RNNs. The ANN dynamics are compared against a signal obtained by low-pass filtering the spike trains generated by the $\Sigma\Delta$ neuron. Our assertion is that the mapping mechanism will map any recurrent ANN to an equivalent recurrent SNN. Therefore, instead of demonstrating the mapping for a particular task, we set synaptic weights to random samples from a Gaussian distribution. We then compare the dynamics of all the neuron units in the mapped and original networks. To ensure that our nodes do not saturate, we constrain the largest eigenvalue of the recurrent weight matrices to 1.4. The motivation for this trick was from obtained from Echo-State Networks (ESNs)(Jaeger, 2002; Jaeger et al., 2007). The quality or goodness of fit is measured using an Normalized mean square error (NMSE) metric. It is used to compare two time series signals x_{ref} and x using the following formulation:

$$NMSE = 1 - \frac{||x_{ref} - x||^2}{||x_{ref} - mean(x_{ref})||^2} \tag{7}$$

where, $||.||$ indicates the L2 norm. The $NMSE$ metric lies between 1 and $-\infty$, with 1 indicating a perfect match and $-\infty$ indicating a very bad fit. In our results, we report the mean and standard deviation in the NMSE scores for all the units in a layer.

The effectiveness of the mapping mechanism is demonstrated using a four-layer RNN. The input and output stages are implemented as fully-connected feed-forward layers, and the recurrent layers are also interleaved with fully-connected layers. The input feature dimension was set to two and the output to three. The input data was a weighted sum of sinusoidal signals that were band-limited to 50 Hz and sampled at 1 MHz for 0.2 seconds. The high sampling rate is necessary to accurately capture the dynamics of the SNN, whose neuron models are highly non-linear, in a transient simulation. The length of the simulation is a key consideration as we want our mapped SNN implementation to remain matched for arbitrarily long sequences. Computational considerations limited the duration of our simulations, but this should be tested before large scale deployment in real-world use.

Fully-digital neuromorphic platforms, such as Intel Loihi, IBM TrueNorth, or SpinNaker, do not suffer for mismatch issues. However, mixed-signal neuromorphic chips, such as Qiao et al. (2015); Neckar et al. (2019); Schemmel et al. (2012), are potentially more energy-efficient than their digital counterparts but suffer from device mismatch. A $\Sigma\Delta$ feedback loop naturally compensates for such effects but there are many components in the model that lies outside the feedback loop. To study this, we add the effect of mismatch in our simulations by sampling the parameters of the mapped SNN as per Equation 8.

$$p = p \cdot (1 + c_{v_p} \cdot \mathcal{N}(0, 1)) \tag{8}$$

where, c_{v_p} is the coefficient of variation ($= \frac{standard\ deviation}{mean}$) in the parameter, p.

To understand the statistics in the quality of the mapping mechanism, we generate multiple samples of network parameters and measure the quality of fit. These results are tabulated in Tables 1 and 2, where we list the measured mean of and standard deviation in $NMSE$ values for a 4-layer RNN with 51 and 500 units per layer, respectively. With no mismatch effects, the reconstruction is very good to all layers, in both cases. Furthermore, we observe nearly perfect reproduction of the network dynamics for up to 2 layers, and a gradual degradation as the size, depth and mismatch of the network increases. This is excellent news for practical neuromorphic applications as they tend to be shallower and smaller. Visualization of the transient dynamics of all the nodes in the recurrent ANN and the mapped SNN is provided in supplementary section F.

We also show statistics for $c_{v_p} = 0.2$, for realistic values of CMOS mismatch, to $c_{v_p} = 2$, for systems with highly-mismatched devices such as memristors or plastic electronics. We note that the mapping mechanism is robust even for such large values of c_{v_p}, highlighting the importance of the feedback compensation effect of the $\Sigma\Delta$ loop. This is useful for designers of mixed-signal neuromorphic chips because the robustness of the mapping mechanism simplifies the design requirements that, in turn, reduces the energy and area consumed by the chips.

Layer	μ_{NMSE}				σ_{NMSE}			
	$c_{v_p} = 0$	$c_{v_p} = 0.2$	$c_{v_p} = 1$	$c_{v_p} = 2$	$c_{v_p} = 0$	$c_{v_p} = 0.2$	$c_{v_p} = 1$	$c_{v_p} = 2$
Rec. layer 1	1.0	0.9	-5.1	-4.0	0.1	0.3	55.1	15.5
Rec. layer 2	1.0	0.5	-8.8	-17.5	0.1	1.1	54.4	41.4
Rec. layer 3	0.9	-1.5	-36.8	-100.2	0.1	6.4	117.1	300.0
Rec. layer 4	0.9	-3.7	-107.5	-278.9	0.2	8.2	202.4	535.9

Table 1: Goodness of fit when mapping a 4 layer network with 51 units per hidden layer for different values of mismatch c_{v_p}. Each data-point computed from 100 samples.

Layer	μ_{NMSE}				σ_{NMSE}			
	$c_{v_p} = 0$	$c_{v_p} = 0.2$	$c_{v_p} = 1$	$c_{v_p} = 2$	$c_{v_p} = 0$	$c_{v_p} = 0.2$	$c_{v_p} = 1$	$c_{v_p} = 2$
Rec. layer 1	1.0	1.0	0.4	0.6	0.1	0.1	1.8	1.3
Rec. layer 2	0.9	-1.0	-48.9	-132.0	0.1	1.6	52.1	143.7
Rec. layer 3	0.8	-6.8	-185.1	-683.6	0.2	2.9	73.2	295.9
Rec. layer 4	0.2	-6.3	-133.7	-416.3	0.9	3.5	64.0	203.6

Table 2: Goodness of fit when mapping a 4 layer network with 500 units per hidden layer for different values of mismatch c_{v_p}. Each data-point computed from 100 samples.

4.3 LEARNING PERFORMANCE OF THE LPRNN CELL

The key idea behind enabling the mapping between an recurrent ANN to its spiking equivalent was the addition of a low pass filter to the the state variables. It must be highlighted that the idea of using a low-pass filter model for neurons is fairly old and has been studied in various contexts (Beer, 1995; Mozer, 1992; Jaeger et al., 2007). The novelty in this work is in identifying its use in the mapping mechanism and in the study of its learning properties. The mapping procedure would be of no use if the resulting ANN was unable to perform as well as their unfiltered counterparts. The experiments in this section focuses on the performance of the lpRNN cell.

In our experiments, we prefer to set α in Equation 6 by sampling from a distribution with a common mean value shared by all the neuron units in the network. This simplifies the design of the neuromorphic system by eliminating the need to create precise tunable time constants in the neuron implementations. Interestingly, we note that the lpRNN cell performs very well even when they are randomly sampled from a uniform distribution. It is also more biologically plausible, as it is unlikely that neurons or dendrites can accurately set their time-constants themselves with arbitrary precision. If α is not learnt, the number of trainable parameters in the lpRNN cell is the same as a SimpleRNN.

First, we study the short-term memory capabilities of the low pass RNN cells using the synthetic addition and copying tasks that are commonly used for this purpose (Hochreiter and Schmidhuber, 1997; Arjovsky et al., 2016; Le et al., 2015). Next, we compare the performance of the lpRNN cell vs a SimpleRNN cell in a speech recognition task. In our experiments, we observe a dramatic improvement in the performance of the RNN cell with the addition of the low-pass filtering term. We think that this improvement is because the LPF acts as a temporal regularizer and a theoretical argument for it is provided in supplementary section C. We also study the effect of filtering on Long Short-Term Memory (LSTM) cells in supplementary section H.

4.4 SYNTHETIC BENCHMARKS

Temporal addition task: The addition task (Le et al., 2015) involves processing two parallel input data streams of equal length. The first stream comprises random numbers $\in (0, 1)$ and the second is full of zeros except at two time steps. At the end of the sequence, the network should output the sum of the two numbers in the first data stream corresponding to the time-steps when the second stream had non-zero entries. The baseline to beat is a mean square error (mse) of 0.1767, corresponding to a network that always generates 1. We adopt a curriculum learning (Bengio et al., 2009) procedure for this task, by first training the RNN cell on a short sequence and progressively increasing the number of time steps. Each curriculum used 10,000 training and 1000 test samples. The length of the task was incremented when the mse went below 0.001. With random initialization, the SimpleRNN cell failed to converge beyond sequence length 40. On the other hand, the lpRNN cell was able to transfer learning for sequences shorter than 150 steps. While, the benefits of curriculum learning appears to have reduced beyond that, the lpRNN cells were able to achieve better than 0.001 mse for sequences up to 642. The performance of the lpRNN cell is at par or slightly inferior to other works in literature, and we achieved this result purely by random initialization (Hochreiter and Schmidhuber, 1997; Arjovsky et al., 2016; Le et al., 2015; Hu et al., 2018). More details of the experiment are provided in the supplementary material.

Temporal copying task: We train the RNN cells on a varying length copying task as defined in (Graves et al., 2014) instead of the original definition (Hochreiter and Schmidhuber, 1997; Arjovsky et al., 2016). This problem is harder to solve than the temporal addition task. The network receives a sequence of up to S symbols (in the original definition, S is fixed) drawn from an alphabet of size K. At the end of S symbols, a sequence of T blank symbols ending with a trigger symbol is passed. The trigger symbol indicates that the network should reproduce the first S symbols in the same order. We adopt a curriculum learning procedure here too and first train the network on a short sequence (T=3) and gradually increase it (T$\leq$200). The sequence length is incremented when the categorical accuracy is better than 99%. The SimpleRNN cell failed at this task even for T=30 with categorical accuracy dropping to 84% when it predicted only S + T blank symbols. The lpRNN cell was able to achieve 99% accuracy for up to 120 time steps. After that, it generates T blank entries accurately but the accuracy of the last S symbols drops (For T=200, it was 96%). More details of the experiment are provided in the supplementary material.

4.5 REAL-WORLD BENCHMARK - SPEECH RECOGNITION

The target for the lpRNN cell are neuromorphic platforms that are typically resource-constrained, due to power, memory, and area restrictions, and can only implement small networks. Therefore, we present the performance of the lpRNN cell on a problem that is compatible with such systems (see supplementary section D). We chose a limited vocabulary spoken commands detection task using the Google commands dataset (Warden, 2018), which comprises 36 classes of short-length spoken commands such as "left", "up", or "go". The dataset was created for hardware and algorithm developers to evaluate their low footprint neural network models in limited dictionary speech recognition tasks. The neural network receives a Mel-spectrogram as inputs, and comprises, from input to output, two

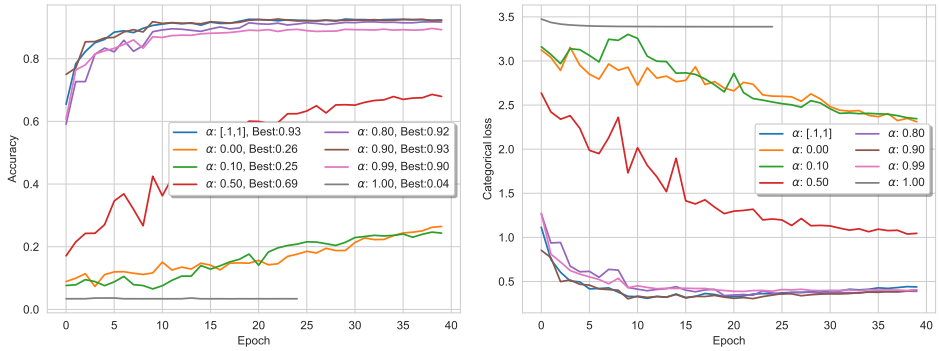


Figure 3: Performance of the lpRNN cell on the Google commands detection task: Categorical accuracy (Left) and Cross entropy loss (Right). We note that as higher values of α and random sampling results in faster convergence and better accuracy.

fully-connected dense layers with 128 and 32 units with batch normalization, two RNN layers with 128 units, two fully-connected dense layers, topped by a softmax readout. In total, the network has about 80K trainable parameters. We use Adam optimizer (Kingma and Ba, 2014), with a learning rate of 0.001 for training. We compare the performance of the low pass filtered RNN cell with the unfiltered one in Figure 3. The figure shows the convergence properties of the low pass filtered cell for different values of α. We note that the low-pass filtered cells massively outperform their unfiltered counterparts achieving better than 92% accuracy on the validation set. We further note that the performance of the network improves as α increases. Interestingly, randomly sampling α also appears to perform almost as well as the best-tested case for $\alpha = 0.95$. The unfiltered SimpleRNN cell completely fails to converge above the chance level. The high performance of the lpRNN cell is a crucial result because not only did the addition of the low-pass filter enable mapping of the recurrent ANN model to neuromorphic platforms, it also resulted in a much-improved performance. Unfortunately, it is computationally prohibitive to compare the accuracy results of the recurrent SNN to the reference model on the full dataset. We demonstrate the mapping accuracy using the NMSE metric for a single command in supplementary section G. A complete analysis of the accuracy performance will require testing on a neuromorphic system.

5 CONCLUSION AND OUTLOOK

We presented a novel aI&F spiking neuron model and discussed its spike-coding and noise-tolerance benefits. Then, we discussed how the LPF abstraction of the aI&F neuron model enables training of recurrent SNNs without having to simulate the complex dynamics of a large network of spiking neurons. This strategy enables training a recurrent SNN using standard optimization algorithms such as backpropagation. The mapping technique that allowed us to achieve this result is studied in detail, and its performance is validated with example spatio-temporal processing tasks. We also observe that the LPF dramatically improves the inference performance of the RNN cell. The proposed model and its use in networks will be instrumental in designing a new generation of ultra-low-power neuromorphic chips able to solve complex real-world problems.

REFERENCES

Ahn, J., Yoo, S., Mutlu, O., and Choi, K. (2015). Pim-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 336–348. IEEE.

Aimar, A., Mostafa, H., Calabrese, E., Rios-Navarro, A., Tapiador-Morales, R., Lungu, I.-A., Milde, M. B., Corradi, F., Linares-Barranco, A., Liu, S.-C., et al. (2018). Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps. *IEEE transactions on neural networks and learning systems*.

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., Imam, N., Nakamura, Y., Datta, P., Nam, G.-J., et al. (2015). Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1537–1557.

Arjovsky, M., Shah, A., and Bengio, Y. (2016). Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128.

Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

Beer, R. D. (1995). On the dynamics of small continuous-time recurrent neural networks. *Adaptive Behavior*, 3(4):469–509.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM.

Bohte, S. M. (2012). Efficient spike-coding with multiplicative adaptation in a spike response model. In *Advances in Neural Information Processing Systems*, pages 1835–1843.

Brette, R. and Gerstner, W. (2005). Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *Journal of neurophysiology*, 94(5):3637–3642.

Cavigelli, L. and Benini, L. (2016). Origami: A 803-gop/s/w convolutional network accelerator. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(11):2461–2475.

Chan, V., Liu, S.-C., and van Schaik, A. (2007). Aer ear: A matched silicon cochlea pair with address event representation interface. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 54(1):48–59.

Chen, Y.-H., Krishna, T., Emer, J. S., and Sze, V. (2016). Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138.

Chi, P., Li, S., Xu, C., Zhang, T., Zhao, J., Liu, Y., Wang, Y., and Xie, Y. (2016). Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 27–39. IEEE Press.

Chicca, E., Stefanini, F., Bartolozzi, C., and Indiveri, G. (2014). Neuromorphic electronic circuits for building autonomous cognitive systems. *Proceedings of the IEEE*, 102(9):1367–1388.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., et al. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99.

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., and Pfeiffer, M. (2015). Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Frenkel, C., Lefebvre, M., Legat, J.-D., and Bol, D. (2019). A 0.086-mm2 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm cmos. *IEEE transactions on biomedical circuits and systems*, 13(1):145–158.

Furber, S. B. and Furber, S. B. (1996). *ARM system Architecture*. Addison-Wesley Menlo Park.

Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural Computation*, 12(10):2451–2471.

Gerstner, W. and Kistler, W. M. (2002). *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press.

Gewaltig, M.-O. and Diesmann, M. (2007). Nest (neural simulation tool). *Scholarpedia*, 2(4):1430.

Ghose, S., Hsieh, K., Boroumand, A., Ausavarungnirun, R., and Mutlu, O. (2018). Enabling the adoption of processing-in-memory: Challenges, mechanisms, future research directions. *arXiv preprint arXiv:1802.00320*.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *CoRR*, abs/1410.5401.

Hamilton, T. J., Jin, C., Van Schaik, A., and Tapson, J. (2008). An active 2-d silicon cochlea. *IEEE Transactions on biomedical circuits and systems*, 2(1):30–43.

Henaff, M., Szlam, A., and LeCun, Y. (2016). Recurrent orthogonal networks and long-memory tasks. *arXiv preprint arXiv:1602.06662*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hu, Y., Huber, A., Anumula, J., and Liu, S.-C. (2018). Overcoming the vanishing gradient problem in plain recurrent networks. *arXiv preprint arXiv:1801.06105*.

Jaeger, H. (2002). *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*. GMD-Forschungszentrum Informationstechnik.

Jaeger, H., Lukoševičius, M., Popovici, D., and Siewert, U. (2007). Optimization and applications of echo state networks with leaky-integrator neurons. *Neural networks*, 20(3):335–352.

Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2016). Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Le, Q. V., Jaitly, N., and Hinton, G. E. (2015). A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*.

Li, S., Li, W., Cook, C., Zhu, C., and Gao, Y. (2018). Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5457–5466.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. (1994). The penn treebank: annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology*, pages 114–119. Association for Computational Linguistics.

Moradi, S., Qiao, N., Stefanini, F., and Indiveri, G. (2018). A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs). *Biomedical Circuits and Systems, IEEE Transactions on*, 12(1):106–122.

Mostafa, H. (2017). Supervised learning based on temporal coding in spiking neural networks. *IEEE transactions on neural networks and learning systems*, 29(7):3227–3235.

Mozer, M. C. (1992). Induction of multiscale temporal structure. In *Advances in neural information processing systems*, pages 275–282.

Nair, M. V. and Indiveri, G. (2019). An ultra-low power sigma-delta neuron circuit. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5.

Neckar, A., Fok, S., Benjamin, B. V., Stewart, T. C., Oza, N. N., Voelker, A. R., Eliasmith, C., Manohar, R., and Boahen, K. (2019). Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model. *Proceedings of the IEEE*, 107(1):144–164.

Neil, D., Pfeiffer, M., and Liu, S.-C. (2016). Phased lstm: Accelerating recurrent network training for long or event-based sequences. In *Advances in neural information processing systems*, pages 3882–3890.

Oliphant, T. (2006). *Guide to NumPy*.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*.

Pavan, S., Schreier, R., and Temes, G. C. (2017). *Understanding delta-sigma data converters*. John Wiley & Sons.

Payvand, M., Nair, M. V., Müller, L. K., and Indiveri, G. (2019). A neuromorphic systems approach to in-memory computing with non-ideal memristive devices: From mitigation to exploitation. *Faraday Discussions*, 213:487–510.

Pelgrom, M., Duinmaijer, A., and Welbers, A. (1989). Matching properties of MOS transistors. *IEEE Journal of Solid-State Circuits*, 24(5):1433–1440.

Qiao, N., Mostafa, H., Corradi, F., Osswald, M., Stefanini, F., Sumislawska, D., and Indiveri, G. (2015). A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Frontiers in neuroscience*, 9:141.

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in neuroscience*, 11:682.

Sarpeshkar, R. (1998). Analog versus digital: Extrapolating from electronics to neurobiology. *Neural Computation*, 10(7):1601–1638.

Schaller, R. R. (1997). Moore's law: past, present and future. *IEEE spectrum*, 34(6):52–59.

Schemmel, J., Grübl, A., Hartmann, S., Kononov, A., Mayr, C., Meier, K., Millner, S., Partzsch, J., Schiefer, S., Scholze, S., Schüffny, R., and Schwartz, M. (2012). Live demonstration: A scaled-down version of the BrainScaleS wafer-scale neuromorphic system. In *IEEE International Symposium on Circuits and Systems ISCAS 2012*, page 702.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Stimberg, M., Brette, R., and Goodman, D. (2019). Brian 2: an intuitive and efficient neural simulator. *BioRxiv*, page 595710.

Waldrop, M. M. (2016). The chips are down for moore's law. *Nature News*, 530(7589):144.

Wang, Y. and Tian, F. (2016). Recurrent residual learning for sequence classification. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 938–943.

Warden, P. (2018). Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*.

Xingjian, S., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., and Woo, W.-c. (2015). Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810.

Yoon, Y. C. (2016). Lif and simplified srm neurons encode signals into spikes via a form of asynchronous pulse sigma–delta modulation. *IEEE transactions on neural networks and learning systems*, 28(5):1192–1205.

A MOTIVATION FOR THE USE OF THE $\Sigma\Delta$ NEURON MODEL

A.1 CODING MECHANISM

A neuron in deep learning has one primary function - the non-linear transformation of its input. However, a biological neuron and its neuromorphic counterpart have the added job of encoding information in spike trains. The most popular model for this encoding mechanism is rate coding, where the neuron fires at a rate proportional to the incoming signal. This mechanism is not efficient for transmitting high-resolution data. For example, to transmit a signal at 8-bit resolution, it will require O(256) spikes for each sample. An improvement to this coding mechanism is theaI&F model (Brette and Gerstner, 2005), which can be interpreted as an asynchronous delta-sigma ($\Delta\Sigma$) loop (Yoon, 2016; Bohte, 2012; Nair and Indiveri, 2019). The $\Delta\Sigma$ mechanism is a time-coding model that is more efficient in its use of spike trains (Nair and Indiveri, 2019) than rate-coding models. Other time-coding schemes have also been proposed in literature (Rueckauer et al., 2017; Gerstner and Kistler, 2002). However, the advantage of the $\Delta\Sigma$ interpretation is that it leads to highly power-efficient circuit implementations (Nair and Indiveri, 2019) that is tolerant to mismatch and noise effects. Furthermore, the model allows us to treat the neuron state as an analogue variable and ignoring the specific timing details of the encoding spike trains (Nair and Indiveri, 2019). The independence from the monitoring precise spike-times is beneficial because state-dependency, noise and device mismatch cause different neurons to generate spikes at different times for the same input. Modelling it is computationally expensive. The $\Delta\Sigma$ feedback loop ensures that they all represent the same signal with the same accuracy in spite of their differences (Nair and Indiveri, 2019). This abstraction enables the network designer to only look at the internal state of the neuron when optimizing the network weights for an SNN. It is a crucial enabler for this paper as simulating and training mismatch-prone SNNs is computationally much more expensive than ANNs.

A.2 BIOLOGICAL NEURAL NETWORKS ARE LOW PASS FILTERING

Activation functions used in neural networks and apply a non-linearity to a weighted sum of input signals. However, ANNs assume that when the input changes, the internal state of the neuron or dendrites can also change immediately to reflect the new input. This behaviour ignores the fact that biological neuronal channels are LPFs (Gerstner and Kistler, 2002). Modelling the inertial or low-pass filtering property is essential to implement and study recurrent neural networks in any neuromorphic system as the transitional dynamics deviate completely in its absence. The $\Delta\Sigma$ neuron models the filtering behaviour with a first-order low pass filter. We argue that not only is this modelling essential, it is also a useful constraint to impose on RNNs.

A.3 CHANNEL NOISE AND RECONSTRUCTION ACCURACY IN A SPIKING NEURON

The $\Sigma\Delta$ scheme encodes the information in the relative timing of the spikes. This implies that any noise in the spike timing, i.e. jitter, introduced during transmission of the spikes will result in distortion of the reconstructed signal, for example, in situations when the transmitting and receiving neurons are on separate chips. The distortions are modelled by a random variable Δ_t, which is sampled from a normal random distribution, $\Delta_t \sim \mathcal{N}(0, j_\sigma^2)$ with probability distribution function $Pr(\Delta_t)$. Note that a fixed delay does not contribute to distortion and is ignored. To compute the effect of spike timing noise, we calculate the Laplace domain representation of the transmitted signal as follows. If $I_D(t)$ is the Dirac delta function, the transmitted spike train $y(t)$, jitter-affected spike train, $y'(t)$, the desired filtered spike train, $s(t)$, and the filtered version of the jitter-affected spike-train,

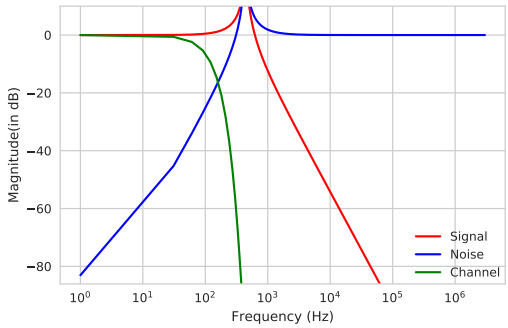


Figure A.1: The different transfer function of the $\Sigma\Delta$ neuron model. Red: Signal transfer function, Blue: noise transfer function, Green: Signal transfer function with spike timing noise.

$r(t)$ can be expressed as,

$$y(t) = \sum_{k=0}^{N} I_D(t - t_k) \tag{9}$$

$$y'(t) = \sum_{k=0}^{N} I_D(t - t_k - \Delta_t) \tag{10}$$

$$s(t) = y(t) * h(t) = \sum_{k=0}^{N} h(t - t_k) \tag{11}$$

$$\implies S(s) = \sum_{k=0}^{N} H(s) e^{-st_k} \tag{12}$$

$$r(t) = y'(t) * h(t) = \sum_{k=0}^{N} h(t - t_k - \Delta_t) \tag{13}$$

The mean value of $r(t)$ can then be computed as

$$\bar{r}(t) = E[r(t)] = \sum_{k=0}^{N} \int_{\Delta_t=-\infty}^{\infty} Pr(\Delta_t) \cdot h(t - t_k - \Delta_t)$$

$$= \sum_{k=0}^{N} Pr \star h(t - t_k) \tag{14}$$

$$\implies \overline{R}(s) = \mathscr{L}\{\bar{r}(t)\} = \sum_{k=0}^{N} P(s)H(s)e^{-st_k} = P(s)S(s) \tag{15}$$

where, $P(s) = \mathscr{L}\{\Pr(t)\} = e^{\frac{s^2 j_\sigma^2}{2}}$. $\overline{R}(s)$ is plotted in Figure A.1 as the recovery transfer function (RTF) with $j_\sigma = 1ms$. We see that for sufficiently band-limited signal, the channel noise does not affect the signal transmission accuracy.

B Comparison to other RNN models

The low pass filtering behaviour of neurons is well-known in neuro-scientific literature (and in other fields) and has been studied in the past with RNNs as well(Beer, 1995; Mozer, 1992; Jaeger et al., 2007). For example, ESNs as proposed by Herbert Jaeger (Jaeger et al., 2007) has an identical formulation to the lpRNN where the recurrent layer uses leaky integration units. In ESNs, the spectral radius of the initialization values of the recurrent kernel is constrained to confers an "echo-state

property" to the network. The recurrent or input connectivity weights are not trained during the learning process. Instead, only the read-out linear classifier is trained. In the lpRNN model, the spectral radius of the recurrent kernel is not constrained and all the weight matrices, including the retention ratios if required, are trained. lpRNN also shares similarities with recurrent residual networks proposed by Yiren Wang (Wang and Tian, 2016), which are described by the following equations

$$y_t = f(g(y_{t-1})) + \sigma(y_{t-1}, x_t, W) \tag{16}$$

where W denotes input and recurrent kernels, and other symbols have the same meaning as equation 6. In equation 16, g and f are identity and a hyperbolic tangent functions, respectively. A comparison can also be made with the LT-RNN model proposed by Mikael Henaff (Henaff et al., 2016), whose update equations are:

$$h_t = \sigma(W_{in} \cdot x + b) + V \cdot h_{t-1} \tag{17}$$

$$y_t = W \cdot h_t \tag{18}$$

where W and V are 2-D transition matrices that are learned during the training process. In this case, it is possible that the LT-RNN cell reduces to an lpRNN, but is unlikely to occur in practice. Similar analogies can also be made to the IndRNN model (Li et al., 2018) and recurrent identity networks (Hu et al., 2018). Generally speaking, the main difference between the lpRNN cells and popular RNN models in use today is the(re)indroduction of the filtering term into the RNN model with impositions on boundary and train-ability conditions. We see that in addition to enabling their use in neuromorphic platforms, this results in more stable convergence properties due to a temporal regularization effect, as described in the Section C.

C MEMORY IN AN LPRNN

We can analyze the evolution of an lpRNN cell by using an approach similar to the power iteration method, described by Razvan Pascanu (Pascanu et al., 2013). To do this analysis, we approximate the lpRNN update equation as:

$$y_t = \alpha \odot y_{t-1} + (1-\alpha) \odot (W_{rec} \cdot y_{t-1} + W_{in} \cdot x_t + b) \tag{19}$$

where, for simplicity, we also make the added assumption that all units of the lpRNN layer have the same retention factor, α. The gradient terms during back-propagation through time can now be expressed as a product of several terms that have the form:

$$\frac{\delta y_t}{\delta y_k} = [(1-\alpha)W_{rec}^T + \alpha]^l \tag{20}$$

where, t and k, are time step indices with $t > k$ and $l = t - k$. If an eigenvalue of the W_{rec}^T matrix is λ, then the corresponding eigenvalue of the matrix $[(1-\alpha)W_{rec}^T + \alpha]$ can be written as

$$(1-\alpha)\lambda + \alpha \tag{21}$$

Looking at the eigenvalue of the gradient terms as computed in equation 21, we note that α acts like a **temporal regularizer** on the eigenvalues of the recurrent network. It can also be seen that by scaling α to lie between 0 and 1, the operation of the network shifts between that of purely non-inertial recurrent to a completely inertial network stuck in its initial state, respectively. This insight helps us understand why lpRNNs perform well in long memory tasks.

Hochreiter (Hochreiter and Schmidhuber, 1997) defined the constant error carousel (CEC) as a central feature of the LSTM networks that allowed it to remember past events. In a crude sense, this corresponds to setting the retention ratio, $\alpha = 1$. Forget gates were subsequently added by Felix A. Gers (Gers et al., 2000) to the original LSTM structure, that allowed the network to also erase unnecessary events that were potentially trapped in the CEC. This means that the average effective weight of the self-connection in the CEC was made < 1. A randomly initialized set of α values with a reasonably large number of cells appears to have similar functionality. By setting $\alpha < 1$, the network is guaranteed to lose memory over time, but if some of the αs are close to 1, it may retain the information for a longer time frame. Moreover, the regularization effect of αs also prevents the eigenvalues of the recurrent network from becoming too small, ensuring that memory is never lost immediately. We expect that the lpRNN model has a reduced representational power than gated RNN cells, not simply because it has 4x fewer parameters, but because the lpRNN state is guaranteed to fade with time whereas a gated cell can potentially store a state indefinitely.

D THE NEUROMORPHIC SIGNAL CHAIN

The $\Delta\Sigma$ mapping mechanism requires defining suitable time constants for the lpRNN cell being trained by backprop. This can be derived for continuous-time signals from sensors or real-world signals such as an audio input by taking into account the bandwidth of the incoming signals as described earlier. We illustrate a reference neuromorphic signal chain for processing audio input in Figure D.2. The data received from the audio sensor is first filtered by an audio filtering stage such as the cochlea chips (Chan et al., 2007; Sarpeshkar, 1998; Hamilton et al., 2008). These systems typically implement mel-spaced filter banks. A neural network processes the filter outputs and drives an actuator system. A minimal configuration of weights and connectivity required to implement the lpRNN cell in a neuromorphic platform is illustrated in Figure D.2. It is a memory array with spiking neurons attached to the periphery of the system. Each memory cell acts as a transconductance stage - it receives voltage spikes and generates a scaled output current. These currents are summed by the Kirchoff's current law and integrated by the neurons. Readers familiar with memory design and computer architecture may identify this as an in-memory computational unit. An in-memory neural network accelerator is energy-efficient, primarily because it eliminates movement of synaptic weights (Ghose et al., 2018; Qiao et al., 2015) from the memory to a far-away processing module. Instead, the activations of the neurons are transmitted to the other nodes in the network. The computation is no longer memory-bound unlike RNN computation on von Neumann style architecture. Figure D.2 implements a single spiking RNN stage (equivalent to an lpRNN), with green and red boxes highlighting the input and recurrent kernels, respectively. The architecture can be modified to implement a fully connected layer by eliminating recurrent connections. Note that an equivalent configuration can be also set up in fully-digital neuromorphic systems such as (Frenkel et al., 2019; Davies et al., 2018; Akopyan et al., 2015) that do not suffer from noise and mismatch issues, but may consume more area and power.

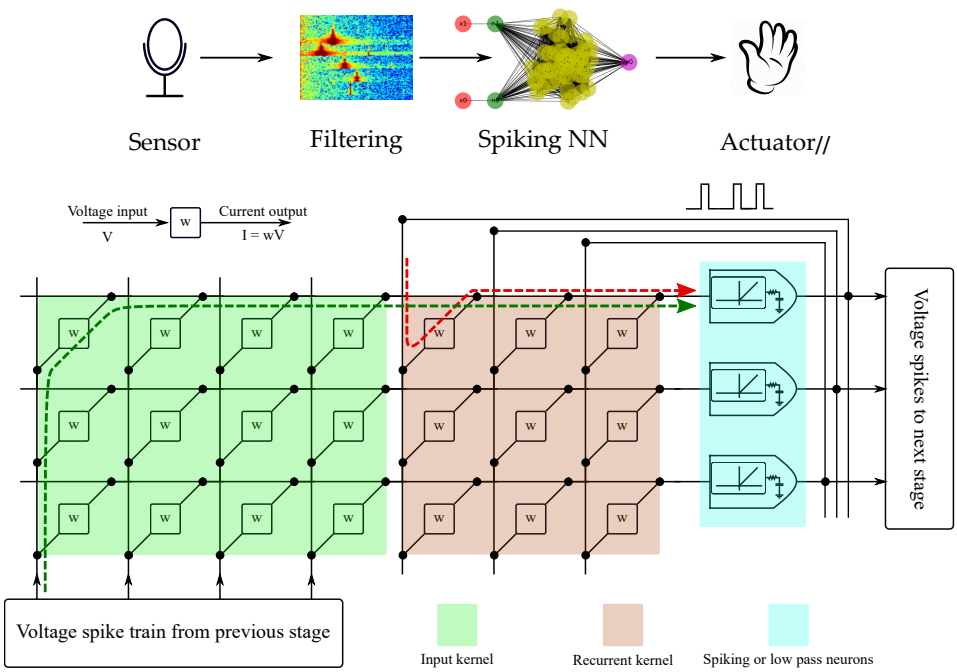


Figure D.2: Top: A neuromorphic signal chain. Bottom: Architecture of an SNN accelerator implementing an lpRNN.

E SPIKER : A SYSTEM-LEVEL SPIKING SIMULATOR

Spiker is a transient-simulator for simulating large networks of spiking neurons and synapses using the Basic Linear Algebra Subprogams (BLAS) libraries. It is written in Python and uses the

Numpy (Oliphant, 2006) library. There is also a PyTorch (Paszke et al., 2017) version that supports GPU-acceleration. Unlike spiking simulation tools like Brian2 (Stimberg et al., 2019) or NEST (Gewaltig and Diesmann, 2007), which are general-purpose solvers of Ordnary Differential Equations (ODEs), Spiker is a highly-specific simulator for systems where the only differential equation implemented is that of a first-order LPFs. The simulator is not designed to solve any differential equation. Instead, it allows us to run massive simulations of large networks comprising neuron and synapse models whose building blocks include first-order LPFs.

E.1 How does Spiker work?

Key idea #1 Constraining the support to first-order LPFs has the advantage that it allows us to create closed-form solutions to the differential equations at all time-step resolutions without loss of accuracy. The differential equation corresponding to a first-order LPF is the following:

$$\tau \frac{dx}{dt} = -x \tag{22}$$

The closed-form solution to this, ignoring the initial conditions takes the form

$$x = x_0 \cdot e^{\frac{-t}{\tau}} \tag{23}$$

A useful property of the exponential function is that $e^{-t1+t2} = e^{-t1} \cdot e^{-t2}$. Therefore, we have a simple closed-form method to compute the state of a LPF at any arbitrary time, given an initial condition. This implies that if all the building blocks of a system are made of modules which have an exponential solution, then, given their initial state, it is possible to compute the state of the entire system at some arbitrary time in the future, precisely. This is a well-known property of all LTI systems.

Key idea #2 However, a spiking system is not LTI. Therefore, it is not possible to predict the state of a spiking neural network at an arbitrary time in the future. Instead, the Spiker simulator takes tiny temporal steps and computes the state of the network variables at each step using the closed-form solution. At the end of each time-step, the neuron models check if it should spike and then resets the corresponding variable to zero. The spiking input to the synapses is a train of 1-bit values corresponding to the presence or absence of an incident spike from an upstream neuron. This implies that the precision of the spike-event is limited to the resolution of the simulation time-step. However, the key insight here is that, if the signals being transmitted have a bandwidth much smaller than the simulation time-step, the higher-order effects can be safely assumed to be gone.

Moreover, in spiking neurons, small amounts of jitter in the spike-timing is well-tolerated. This is because of the noise-cancelling property of the feedback loop. Therefore, the Spiker simulator allows the user to set the simulation time-step to a large value that offers fast transient simulations — a fast-simulation trades-off against the precision of the simulated spike-timing.

Finally, the simulator also allows us to program and simulate noise and mismatch effects in the neuron parameters, such as mismatch in the spiking threshold and time-constants. The Spiker simulator was used to run all the SNN simulations in this paper.

F Visualizing mapping of a recurrent SNN

The top row of Figure F.3 shows the mapping mechanism in action for a single test case. We note that as the depth of the network increases, the quality of fit degrades, but is still close to perfect as measured by the NMSE metric. The bottom row of Figure F.3 shows the mapping mechanism in action for devices with a very large $c_{v_p} = 1$. We note that even in such cases, the performance in lower layers remains fairly stable and only gradually degrades as the depth increases.

G Mapping the Google spoken commands network

Each 1-second recording from the dataset is transformed using the Mel-spectrogram into 25 frequency and 128 temporal bins. This sequence represents a single speech command for the ANN. Mapping

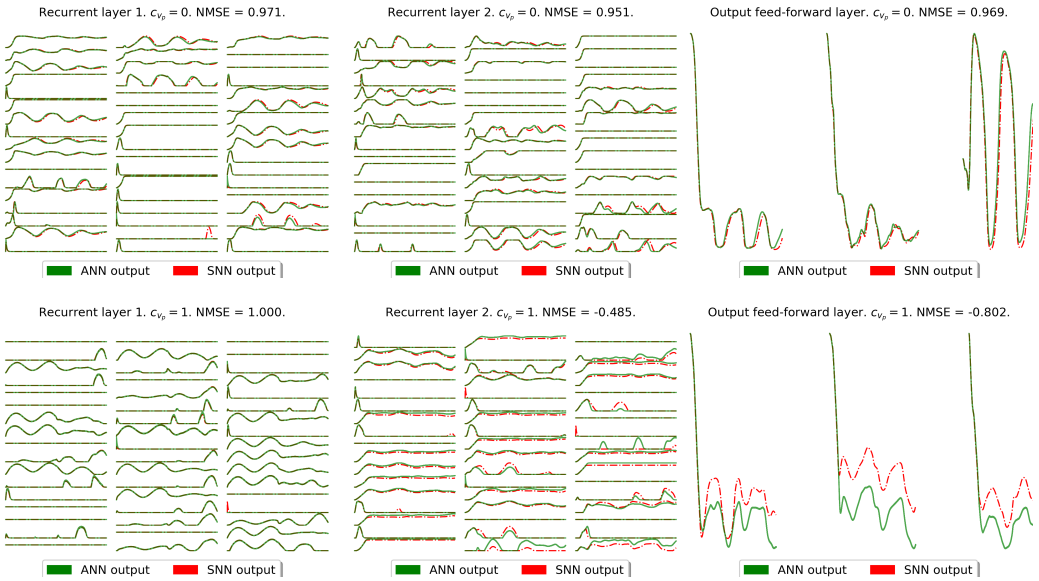


Figure F.3: Dynamics in a two-layer RNN with 51 units per layer. The SNN output shown in the figures is the trace obtain by filtering the spike trains. The NMSE measures the quality of fit with 1 indicating a perfect match and $-\infty$ a very bad fit as described in Equation 7.

the trained ANN to an SNN is challenging because of the limitation described earlier; A short length sequence does not give the SNN enough time to generate the spikes required to transmit information. On the other hand, it is too difficult to train a longer length sequence that is several thousand samples long. To address this issue, we train the ANN using the short sequence with 128 bins and demonstrate the mapped SNN by feeding it the same spectrogram data that is up-sampled to 1 MHz. The quality of the mapped recurrent SNN is demonstrated using the weights from the trained recurrent ANN for a single case in Figure G.4. The bottom row of Figure G.4 also shows the mapping mechanism in action for RNN layers that are affected by mismatch with $c_{v_p} = 0.2$. The results indicate an excellent fit between the mapped and the original networks, even with mismatch. The prediction made by the SNN also matched that of the ANN. Unfortunately, it is computationally prohibitive to compare the accuracy results of the SNN to the reference model on the full dataset. We only demonstrate the mapping accuracy for the two recurrent layers of the network in Figure G.4 for a single command in this paper. A complete analysis of the accuracy performance will require testing on a neuromorphic system and will be the focus of a follow-up work.

H EXTENDING THE LOW-PASS FILTERING IDEA TO OTHER RNN MODELS

Our goal with introducing lpRNN cell was to enable faster and better training of SNNs. However, the analysis performed here indicates that low pass filtering also provides temporal regularization features which can benefit ANN-RNNs such as LSTMs. Therefore, we propose to extend the LSTM formulation by applying a low pass filter at the output (h), and call it an **lpLSTM** cell:

$$\begin{aligned}
\text{Forget gate: } f_t &= Sigmoid(W_f x_t + W_{rec_f} h_{t-1} + b_f) \\
\text{Input gate: } i_t &= Sigmoid(W_i x_t + W_{rec_i} h_{t-1} + b_i) \\
\text{Output gate: } o_t &= Sigmoid(W_o x_t + W_{rec_o} h_{t-1} + b_o) \\
\text{State : } c_t &= f_t \odot c_{t-1} + i_t \odot Relu(W_c x_t + W_{rec_c} h_{t-1} + b_c) \\
\text{Output : } \bar{h}_t &= o_t \odot Relu(c_t) \\
\text{Filtered Output : } h_t &= \alpha \odot h_{t-1} + (1 - \alpha) \odot \bar{h}_t
\end{aligned} \tag{24}$$

where, W_{rec_x}, W_x, b_x indicate the recurrent kernel, input kernel, and bias for the corresponding gate or state. Similar formulations for other RNN cells such as GRU (Chung et al., 2014), IndRNN (Li

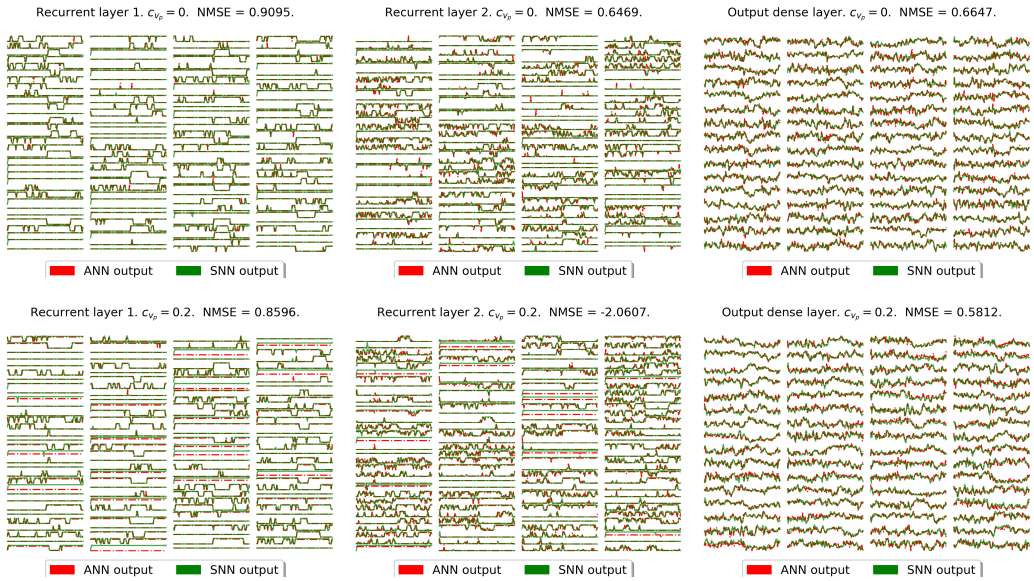


Figure G.4: Mapping a 2-layer 128 unit recurrent ANN trained to discriminate commands from the Google speech dataset to an equivalent recurrent SNN. The top row shows the mapping in action for neuron models unaffected by mismatch and the bottom row demonstrates it for mismatched units. The lpRNN cells have $\alpha = 0.99$.

et al., 2018), Phased-LSTMs (Neil et al., 2016), Convolutional LSTMs (Xingjian et al., 2015), etc can be easily made.

H.1 Experimental results for the lpLSTM cell

In this section, we benchmark the Low-Pass Long Short-Term Memory (lpLSTM) cells to their unfiltered variants. In our experiments, the learning rate was set to 0.005 and normalized gradient was clipped to 1. Current works describe use of various task-specific initialization constraints to solve the addition and copying tasks better (Henaff et al., 2016; Le et al., 2015). Instead of that, we use a data-driven **curriculum learning protocol** in our experiments and are able to obtain dramatically improved performance on these tasks. The networks used for the copying and addition tasks are fairly small. We also test it on a character-level language modelling task with the Penn Treebank dataset (Marcus et al., 1994) using a large network with roughly 19M parameters(Kim et al., 2016). We swap the lpLSTM cells with an LSTM cells in our experiments and the network architecture and hyper-parameter settings were left unchanged from the values reported by the original authors. A summary of our observations are as follows: The lpRNN cell exhibits a dramatically improved performance over the SimpleRNN cell. The low pass filter appears to have a temporal stabilization effect even for LSTM cells. However, when other regularization and stabilization techniques such as Dropout(Srivastava et al., 2014) are introduced, the benefit appears muted. It is possible that the large networks with low-pass RNN layers require network architecture tweaks to benefit from the filtering property, but it was not investigated in this work.

H.1.1 Temporal addition task

The addition task (Le et al., 2015) involves processing two parallel input data streams of equal length. The first stream comprises random numbers $\in (0, 1)$ and the second is full of zeros except at 2 time steps. At the end of the sequence, the network should output the sum of the two numbers in the first data stream corresponding to the time-steps when the second stream had non-zero entries. The baseline to beat is a mean square error (mse) of 0.1767, corresponding to a network that always generates 1.

We first train the RNN cell being tested on a short sequence and progressively increase the length. Each curriculum used 10,000 training and 1000 test samples. The results are shown in Figures H.5, where each stage of the curriculum learning process is marked with bands of different colours. The width of the band indicates the number of epochs taken for convergence. The length of the task was incremented when the mse went below 0.001. With random initialization, the SimpleRNN cell failed to converge beyond sequence length 40, even with curriculum learning. On the other hand, both the lpRNN and LSTM cells benefit from the curriculum learning protocol. The lpRNN cell was able to transfer learning for sequences shorter than 150 steps. While, the benefits of curriculum learning appears to have reduced beyond that, the lpRNN cells were able to achieve better than 0.001 mse for sequences up to 642. The performance of the lpRNN cell is at par or slightly inferior to other works in literature (Hochreiter and Schmidhuber, 1997; Arjovsky et al., 2016; Le et al., 2015; Hu et al., 2018), we achieved this result purely by random initialization. Another interesting outcome of this experiment was the effectiveness of a 2-unit LSTM cell in solving this task. It was was able to add sequences much longer(we tested up to 100K) than any reported work (where the networks are only able to solve the task for about 1/100th of the sequence length).

Given the effectiveness of the training protocol, we made the task more complex by allowing the second stream to have up to 10 unmasked entries during training. The trained cell was tested with a data stream having more than 10 masked entries. The LSTM cell was successful in solving this problem a mse less than 1e-3, indicating that it learnt a general add and accumulate operation. Figure H.5c shows the evolution of the gating functions, internal state, and the state variables of an LSTM cell that was trained only on a fixed length sequence of 100 with mse less than 0.001. Contrast this against the stable dynamics of the network trained by curriculum learning in Figure H.5d in a 100K sequence with mse less than 1e-3 (Figure H.5) indicating almost perfect long-term memory and addition. To our knowledge, this kind of generalized learning by an LSTM cell has not been shown before.

H.1.2 TEMPORAL COPYING TASK

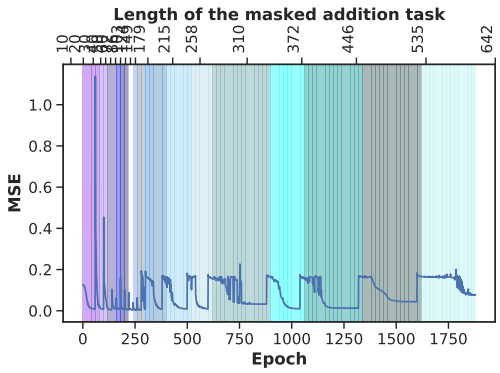
We train the RNN cells on a varying length copying task as defined in (Graves et al., 2014) instead of the original definition (Hochreiter and Schmidhuber, 1997; Arjovsky et al., 2016). This problem is harder to solve than the temporal addition task. The network receives a sequence of up to S symbols (in the original definition, S is fixed) drawn from an alphabet of size K. At the end of S symbols, a sequence of T blank symbols ending with a trigger symbol is passed. The trigger symbol indicates that the network should reproduce the first S symbols in the same order. We first train the network on a short sequence (T=3) and gradually increase it (T$\leq$200). The sequence length is incremented when the categorical accuracy is better than 99%.

The SimpleRNN cell failed at this task even for T=30 with categorical accuracy dropping to 84% when it predicted only S + T blank symbols. The lpRNN cell was able to achieve 99% accuracy for up to 120 time steps. After that, it generates T blank entries accurately but the accuracy of the last S symbols drops (For T=200, it was 96%). However, the LSTM cell achieves more than 99.5+% accuracy for all tested sequence lengths, highlighting the advantage of curriculum learning. This is a big improvement over reported results (Arjovsky et al., 2016; Graves et al., 2014; Bai et al., 2018) where LSTM cells solved the task for much smaller values of S and T.

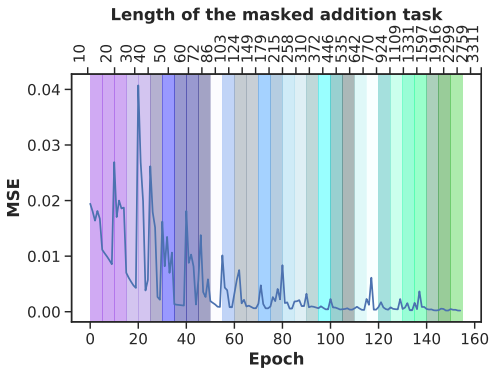
We observed stability issues when training an LSTM cell for sequences longer than 30, even if it eventually converged by using smaller learning rates and gradient norm scaling. This makes a good test case to validate the temporal regularization property of the lpLSTM cell. In our tests, the lpLSTM cell converged without instability with categorical accuracy higher than 99.5% for all tested values of T($\in [3, 500]$). It also to generalized larger values of S than the other cells (≤ 25). The lpLSTM cell exhibited a gradual degradation in performance for larger values of S. We stop our simulations when the categorical accuracy fell below 96%. These results are summarized in Figure H.6.

H.1.3 PENN TREEBANK (PTB) CHARACTER MODEL

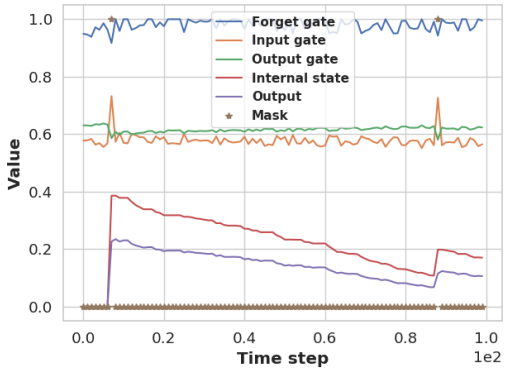
We studied temporal regularization in a network trained on the PTB dataset (Marcus et al., 1994) by replacing the LSTM cells by its low pass variants. We choose a model with 19M parameters (Kim et al., 2016) and trained all variants using the same settings as described in (Kim et al., 2016) for 25 epochs. We note that both lpLSTM cells converge to a better score on the training set and a marginally



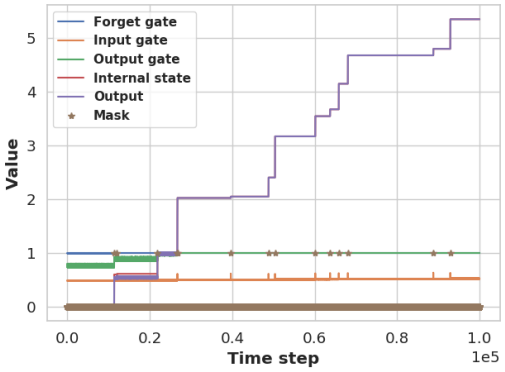
(a) Curriculum learning with a 128 unit lpRNN



(b) Curriculum learning in a 2 hidden unit LSTM



(c) Plain LSTM without curriculum learning



(d) Plain LSTM with curriculum learning

Figure H.5: Curriculum learning on the masked addition task. LSTM cell trained without curriculum learning results in unstable state variables (c). When trained with curriculum learning it looks much more stable (d). Stars in (c) and (d) indicate value of the add mask.

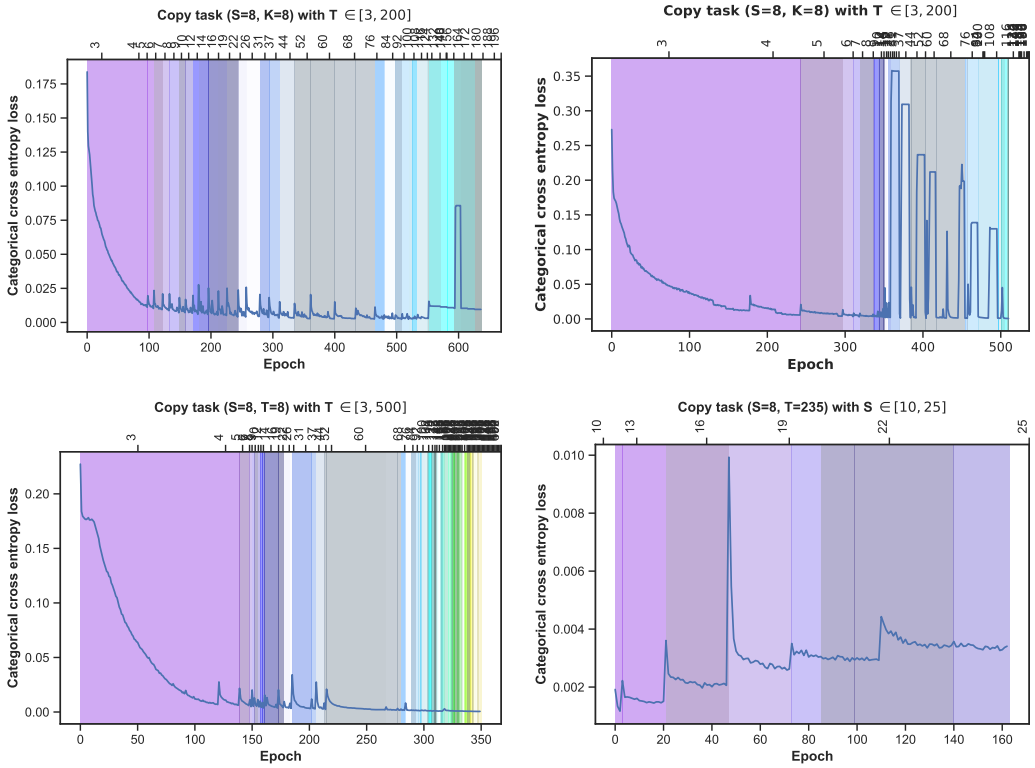


Figure H.6: Curriculum learning on the variable length copying task for a 256 unit lpRNN (top left) and a 128 unit LSTM cell (top right) and a 128 unit lpLSTM cell(bottom left and right).

poorer score on the train/validation set (refer Table 3). The lpLSTM cell with relu activation also converges unlike the plain relu LSTM cell validating our claim on temporal regularization.

Table 3: Impact of temporal regularization on the Penn Treebank model.

	Activation	**Train perplexity**	**Validation Perplexity**	**Test Perplexity**
LSTM	relu	approx. 641	approx. 641	**Fails to converge**
	tanh	46.0948	83.9807	80.0873
lpLSTM	tanh	41.0545	84.6127	81.7519
	relu	43.0602	84.1484	80.6946