Shallow RNNs: A Method for Accurate Time-series Classification on Tiny Devices

Don Kurian Dennis*
Carnegie Mellon University

Durmus Alp Emre Acar
Boston University

Vikram Mandikal†
University of Texas at Austin

Vinu Sankar Sadasivan†
IIT Gandhinagar

Harsha Vardhan Simhadri
Microsoft Research India

Venkatesh Saligrama
Boston University

Prateek Jain
Microsoft Research India

Abstract

Recurrent Neural Networks (RNNs) capture long dependencies and context, and hence are the key component of typical sequential data based tasks. However, the sequential nature of RNNs dictates a large inference cost for long sequences even if the hardware supports parallelization. To induce long-term dependencies, and yet admit parallelization, we introduce novel shallow RNNs. In this architecture, the first layer splits the input sequence and runs several independent RNNs. The second layer consumes the output of the first layer using a second RNN thus capturing long dependencies. We provide theoretical justification for our architecture under weak assumptions that we verify on real-world benchmarks. Furthermore, we show that for time-series classification, our technique leads to substantially improved inference time over standard RNNs without compromising accuracy. For example, we can deploy audio-keyword classification on tiny Cortex M4 devices (100MHz processor, 256KB RAM, no DSP available) which was not possible using standard RNN models. Similarly, using ShaRNN in the popular Listen-Attend-Spell (LAS) architecture for phoneme classification [4], we can reduce the lag in phoneme classification by 10-12x while maintaining state-of-the-art accuracy.

1 Introduction

We focus on the challenging task of time-series classification on tiny devices, a problem arising in several industrial and consumer applications [25, 22, 31], where tiny edge-devices perform sensing, monitoring and prediction in a limited time and resource budget. A prototypical example is an interactive cane for people with visual impairment, capable of recognizing gestures that are observed as time-traces on a sensor embedded onto the cane [24].

Time series or sequential data naturally exhibit temporal dependencies. Sequential models such as RNNs are particularly well-suited in this context because they can account for temporal dependencies by attempting to derive relations from the previous inputs. Nevertheless, directly leveraging RNNs for prediction in constrained scenarios mentioned above is challenging. As observed by several authors [29, 14, 30, 9], the sequential nature by which RNNs process data fundamentally limits parallelization leading to large training and inference costs. In particular, in time-series classification, at inference time, the processing time scales with the size, T, of the receptive window, which is unacceptable in resource constrained settings.

*Work done as a Research Fellow at Microsoft Research India.
†Work done during internships at Microsoft Research India.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

A solution proposed in literature [29, 14, 30, 9] is to replace sequential processing with parallelizable feed-forward and convolutional networks. A key insight exploited here is that most applications require relatively small receptive window, and that this size can be increased with tree-structured networks and dilated convolutions. Nevertheless, feedforward/convolutional networks utilize substantial working memory, which makes them difficult to deploy on tiny devices. For this reason, other methods such as [28, 2] also are not applicable for our setting. For example, a standard audio keyword detection task with a relatively modest setup of 32 conv filters would itself need a working memory of 500KB and about 32X more computation than a baseline RNN model (see Section 5).

Shallow RNNs. To address these challenges, we design a novel layered RNN architecture that is parallelizable/limited-recurrence while still maintaining the receptive field length (T) and the size of the baseline RNN. Concretely, we propose a simple 2-layer architecture that we refer to as ShaRNN. Both the layers of ShaRNN are composed of a collection of shallow recurrent neural networks that operate independently. More precisely, each sequential data point (receptive window) is divided into independent parts called *bricks* of size k, and a *shared RNN* operates on each brick independently, thus ensuring a small model size and short recurrence. That is, ShaRNN's bottom layer restarts from an initial state after every $k << T$ steps, and hence only has a short recurrence. The outputs of T/k parallel RNNs are input as a sequence into a second layer RNN, which then outputs a prediction after T/k time. In this way, for $k \approx O(\sqrt{T})$ we obtain a speedup of $O(\sqrt{T})$ in inference time in the following two settings:

(a) **Parallelization:** here we parallelize inference over T/k independent RNNs thus admitting speed-ups on multi-threaded architectures,

(b) **Streaming:** here we utilize receptive (sliding) windows and reuse computation from older sliding window/receptive fields.

We also note that, in contrast to the proposed feed-forward methods or truncated RNN methods [23], our proposal admits fully receptive fields and thus does not result in loss of information. We further enhance ShaRNN by combining it with the recent MI-RNN method [10] to reduce the receptive window sizes; we call the resulting method *MI-ShaRNN*.

While a feedforward layer could be used in lieu of our RNN in the next layer, such layers lead to significant increase in model size and working RAM to be admissible in tiny devices.

Performance and Deployability. We compare the two-layer MI-ShaRNN approach against other state-of-art methods, on a variety of benchmark datasets, tabulating both accuracy and budgets. We show that the proposed 2-layer MI-ShaRNN exhibits significant improvement in inference time while also improving accuracy. For example, on Google-13 dataset, MI-ShaRNN achieves 1% higher accuracy than baseline methods while providing 5-10x improvement in inference cost. A compelling aspect of the architecture is that it allows for reuse of most of the computation, which leads to its deployability on the tiniest of devices. In particular, we show empirically that the method can be deployed for real-time time-series classification on devices as those based on the tiny ARM Cortex M4 microprocessor[3] with just 256KB RAM, 100MHz clock-speed and no dedicated Digital Signal Processing (DSP) hardware. Finally, we demonstrate that we can replace bi-LSTM based encoder-decoder of the LAS architecture [4] by ShaRNN while maintaining close to best accuracy on publicly-available TIMIT dataset [13]. This enables us to deploy LAS architecture in streaming fashion with a lag of 1 second in phoneme prediction and $O(1)$ amortized cost per time-step; standard LAS model would incur lag of about 8 seconds as it processes the entire 8 seconds of audio before producing predictions.

Theory. We provide theoretical justification for the ShaRNN architecture and show that significant parallelization can be achieved if the network satisfies some relatively weak assumptions. We also point out that additional layers can be introduced in the architecture leading to hierarchical processing. While we do not experiment with this concept here, we note that, it offers potential for exponential improvement in inference time.

[3] `https://en.wikipedia.org/wiki/ARM_Cortex-M#Cortex-M4`

In summary, the following are our main contributions:

- We show that under relatively weak assumptions, recurrence in RNNs and consequently, the inference cost can be reduced significantly.
- We demonstrate this inference efficiency via a two-layer ShaRNN (and MI-ShaRNN) architecture that uses only shallow RNNs with a small amount of recurrence.
- We benchmark MI-ShaRNN (enhancement of ShaRNN with MI-RNN) on several datasets and observe that it learns nearly as accurate models as standard RNNs and MI-RNN. Due to limited recurrence, ShaRNN saves 5-10x computation cost over baseline methods. We deploy MI-ShaRNN model on a tiny *microcontroller* for real-time audio keyword detection, which, prior to this work, was not possible with standard RNNs due to large inference cost with receptive (sliding) windows. We also deploy ShaRNN in LAS architecture to enable streaming phoneme classification with less than 1 second of lag in prediction.

2 Related Work

Stacked Architecture. Our multi-layered RNN resembles stacked RNNs studied in the literature [15, 16, 27] but they are unrelated. The goal of Stacked RNNs is to produce complex models and subsume conventional RNNs. Each layer is fully recurrent, and feeds output of the first layer to the next level. The next level is another fully recurrent RNN. As such, stacked RNN architectures lead to increased model size and recurrence, which results in worse inference time than standard RNNs.

Recurrent Nets (Training). Conventional works on RNNs primarily address challenges arising during training. In particular for large receptive window T, RNNs suffer from vanishing and exploding gradient issues. A number of works propose to circumvent this issue in a number of ways such as Gated architectures [7, 17] or adding residual connections in RNNs [18, 1, 21] or through constraining the learnt parameters [32]. Several recent works attempt to reduce the number of gates and parameters [8, 6, 21] to reduce model size but as such suffer from poor inference time, since they are still fully recurrent. Different from these works, our focus is on reducing model size as well as inference time and view these works as complementary to our paper.

Recurrent Nets (Inference Time). Recent works have begun to focus on RNN inference cost. [3] proposes to learn skip connections that can avoid evaluating all the hidden states. [10] exploits domain knowledge that true signature is significantly shorter than the time-trace to trim down length of the sliding windows. Both of these approaches are complementary and we indeed leverage the second in our approach. A recent work on dilated RNNs [5] is interesting. While it could serve as a potential solution, we note that, in its original form, dilated RNN also has a fully recurrent first layer, which is therefore infeasible. One remedy is to introduce dilation in the first layer to improve inference time. But, dilation skips steps and hence can miss out on critical local context.

Finally, CNN based methods [29, 14, 30, 9, 2] allow higher parallelization in the sequential tasks but as discussed in Section 1, also lead to significantly larger working RAM requirement when compared to RNNs, thus cannot be considered for deployment on tiny devices (see Section 5).

3 Problem Formulation and Proposed ShaRNN Method

In this paper, we primarily focus on the time-series classification problem, although the techniques apply to more general sequence-to-sequence problems like phoneme classification problem discussed in Section 5. Let $\mathcal{Z} = \{(X_1, y_1), \ldots, (X_n, y_n)\}$ where X_i is the i-th sequential data point with $X_i = \big[x_{i,1},\ x_{i,2}, \ldots, x_{i,T}\big] \in \mathbb{R}^{d \times T}$ and $x_{i,t} \in \mathbb{R}^d$ is the t-th time-step data point. $y_i \in \big[C\big]$ is the label of X_i where C is the number of class labels. $x_{i,t:t+k}$ is the shorthand for $x_{i,t:t+k} = \big[x_{i,t}, \ldots, x_{i,t+k}\big]$.

Given training data $\mathcal{Z}$, the goal is to learn a classifier $f : \mathbb{R}^{d \times T} \rightarrow \big[C\big]$ that can be used for efficient inference, especially on tiny devices. Recurrent Neural Networks (RNN) are popularly used for modeling such sequential problems and maintain a hidden state $h_{t-1} \in \mathbb{R}^{\hat{d}}$ at the t-th step that is updated using:

$$h_t = R(h_{t-1}, x_t), t \in \big[T\big], \quad \hat{y} = f(h_T),$$

where $\hat{y}$ is the prediction by applying a classifier f on h_T and $\hat{d}$ is the dimensionality of the hidden state. Due to the sequential nature of RNN, inference cost of RNN is $\Omega(T)$ even if the hardware

supports large amount of parallelization. Furthermore, practical applications require handling a continuous stream of data, e.g., smart-speaker listening for certain audio keywords.

A standard approach is to use sliding windows (receptive field) to form a stream of test points on which inference can be applied. That is, given a stream $\mathcal{X} = [x_1, x_2, \ldots,]$, we form sliding windows $X^s = x_{(s-1)\cdot\omega+1:(s-1)\cdot\omega+T} \in \mathbb{R}^{d\times T}$ which stride by $\omega > 0$ time-steps after each inference. RNN is then applied to each sliding window X^s which implies amortized cost for processing each time-step data point (x_t) is $\Theta(\frac{T}{\omega})$. To ensure high-resolution in prediction, ω is required to be a fairly small constant independent of T. Thus, amortized inference cost for each time-step point is $O(T)$ which is prohibitively large for tiny devices. So, we study the following key question: "*Can we process each time-step point in a data stream in $o(T)$ computational steps?*"

3.1 ShaRNN

Shallow RNNs (ShaRNN) are a hierarchical collection of RNNs organized at two levels. $\frac{T}{k}$ RNNs at ground-layer operate completely in parallel with fully shared parameters and activation functions, thus ensuring small model size and parallel execution. An RNN at the next level take inputs from the ground-layer and subsequently outputs a prediction.

Formally, given a sequential point $X = [x_1, \ldots, x_T]$ (e.g. sliding window in streaming data), we split it into *bricks* of size k, where k is a parameter of the algorithm. That is, we form T/k bricks: $\mathcal{B} = [B_1, \ldots, B_{T/k}]$ where $B_j = x_{((j-1)\cdot k+1):(j\cdot k)}$. Now, ShaRNN applies a standard recurrent model $\mathcal{R}^{(1)} : \mathbb{R}^{d\times k} \to \mathbb{R}^{\hat{d}_1}$ on each brick, where $\hat{d}_1$ is the dimensionality of hidden states of $\mathcal{R}^{(1)}$. That is,

$$\nu_j^{(1)} = \mathcal{R}^{(1)}(B_j), \;\; j \in [T/k].$$

Note that $\mathcal{R}^{(1)}$ can be any standard RNN model like GRU, LSTM etc. We now feed output of each layer into another RNN to produce the final state/feature vector that is then fed into a feed forward layer. That is,

$$\nu_{T/k}^{(2)} = \mathcal{R}^{(2)}([\nu_1^{(1)}, \ldots, \nu_{T/k}^{(1)}]), \quad \hat{y} = f(\nu_{T/k}^{(2)}),$$

where $\mathcal{R}^{(2)}$ is the second layer RNN and can also be any standard RNN model. $\nu_{T/k}^{(2)} \in \mathbb{R}^{\hat{d}_2}$ is the hidden-state obtained by applying $\mathcal{R}^{(2)}$ to $\nu_{1:T/k}^{(1)}$. f applies the standard feed-forward network to $\nu_{T/k}^{(2)}$. See Figure 1 for a block-diagram of the architecture. That is, ShaRNN is defined by parameters Λ composed of shared RNN parameters at the ground-level, RNN parameters at the next level, and classifier weights for making a prediction. We train the ShaRNN based on minimizing an empirical loss function over training set $\mathcal{Z}$.

Naturally, ShaRNN is an approximation of a true RNN and in principle has less modeling power (and recurrence). But as discussed in Section 4 and shown by our empirical results in Section 5, ShaRNN can still capture enough context from the entire sequence to effectively model a variety of time-series classification problems with large T (typically $T \geqslant 100$). Due to parallel k RNNs in the bottom layer that are processed by $\mathcal{R}^2$ in the second layer, ShaRNN inference cost can be reduced to $O(T/k + k)$ for multi-threaded architectures with k-wise parallelization; $k = \sqrt{T}$ leads to smallest inference cost.

Streaming. Recall that in the streaming setting, we form sliding windows $X^s = x_{s\cdot\omega+1:s\cdot\omega+T} \in \mathbb{R}^{d\times T}$ by striding each window by $\omega > 0$ time-steps. Hence, if $\omega = k \cdot q$ for $q \in \mathbb{N}$ then the inference cost of X^{s+1} can be reduced by reusing previously computed $\nu_j^{(1)}$ vectors $\forall j \in [q+1, T/k]$ for X^s.

Below claim provides a formal result for the same.

Claim 1. *Let both layers RNNs $\mathcal{R}^{(1)}$ and $\mathcal{R}^{(2)}$ of ShaRNN have same hidden-size and per-time step computation complexity C_1. Then, given T and ω, the additional cost of applying ShaRNN to X^{s+1} given X^s is $O(T/k + q \cdot k) \cdot C_1$, where $X^s = x_{(s-1)\cdot\omega+1:(s-1)\cdot\omega+T}$, ω is the stride-length of sliding window, and the brick-size $\omega = q \cdot k$ for some integer $q \geqslant 1$. Consequently, the total amortized cost can be bounded by $O(\sqrt{q \cdot T}C_1)$ if $k = \sqrt{T/q}$.*

See Appendix A for a proof of the claim.

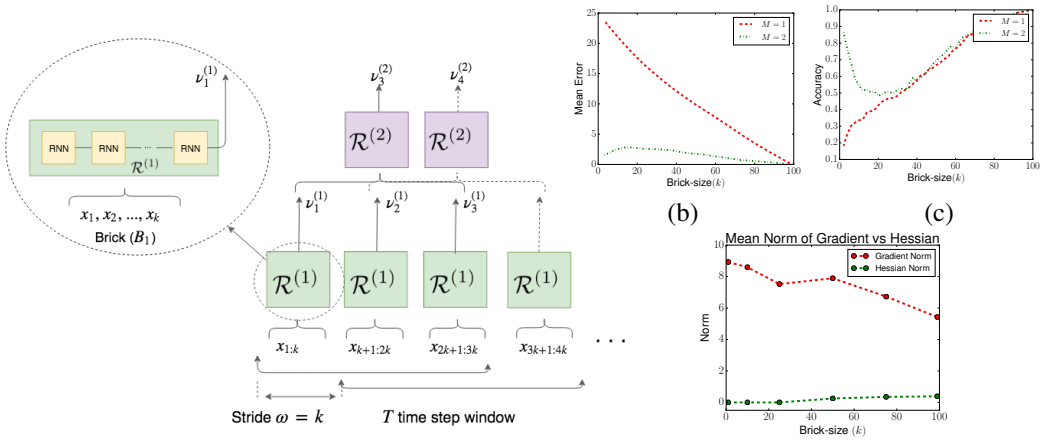


Figure 1: **(a)** ShaRNN applies RNN $\mathcal{R}^{(1)}$ independently to bricks $x_{1:k}, x_{k+1:2k}, \ldots$ to compute $\nu_k^{(1)}$ for all k. The second layer RNN $\mathcal{R}^{(2)}$ produces class labels or in multi-layer case, inputs for the next layer. Note that $\nu_2^{(1)}, \nu_3^{(1)}$ can be reused for evaluating the next window. **(b), (c)**: Mean squared approximation error and the prediction accuracy of ShaRNN with zeroth and first order approximation ($M = 1, 2$ respectively in Claim 3) with different brick-sizes k (for Google-13). Note the large error with $M = 1$ (same as truncation method in [23]). $M = 2$ introduces significant improvement, especially for small k, but clearly needs larger M to achieve better accuracy. **(d)**: Comparison of norm of gradient vs Hessian of $\mathcal{R}(h_t, x_{t+1:t+k})$ with varying k. $\mathcal{R}$ is FastRNN [21] with swish activation. Smaller Hessian norm indicates that the first-order approximation of $\mathcal{R}$ (Claim 3) by ShaRNN is more accurate than the 0-th order one (ShaRNN with $M = 1$) suggested by [23].

3.2 Multi-layer ShaRNN

Above claim shows that selecting small k leads to a large number of bricks and hence, a large number of points to be processed by second layer RNN $\mathcal{R}^{(2)}$ which will be the bottleneck in inference. However, using the same approach, we can replace the second layer with another layer of ShaRNN to bring down the cost. By repeating the same process, we can design a general L layer architecture where each layer is equipped with a RNN model $\mathcal{R}^{(l)}$ and the output of a l-th layer brick is given by:

$$\nu_j^{(l)} = \mathcal{R}^{(l)}([\nu_{(j-1)\cdot k+1}^{(l-1)}, \ldots, \nu_{(j-1)k+k}^{(l-1)}]),$$

for all $1 \leqslant j \leqslant T/k^l$, where $\nu_j^{(0)} = x_j$. The predicted label is given by $\hat{y} = f(\nu_{T/k^{L-1}}^{(L)})$.

Using argument similar to the claim in the previous section, we can reduce the total inference cost to $O(\log T)$ by using $k = O(1)$ and $L = \log T$.

Claim 2. *Let all layers of multi-layer ShaRNN have same hidden-size and per-time step complexity C_1 and let $k = \omega$. Then, the additional cost of applying ShaRNN to X^{s+1} is $O(T/k^L + L \cdot k) \cdot C_1$, where $X^s = x_{(s-1)\cdot\omega+1:(s-1)\cdot\omega+T}$. Consequently, selecting $L = \log(T)$, $k = O(1)$, and assuming $\omega = O(1)$, the total amortized cost is $O(C_1 \cdot \log(T))$.*

That is, we can achieve exponential speed-up over $O(T)$ cost for standard RNN. However, such a model can lead to a large loss in accuracy. Moreover, constants in the cost for large L are so large that a network with smaller L might be more efficient for typical values of T.

3.3 MI-ShaRNN

Recently, [10] showed that several time-series training datasets are coarse and the sliding window size T can be decreased significantly by using their multi-instance based algorithm (MI-RNN). MI-RNN finds tight windows around the actual signature of the class, which leads to significantly smaller models and reduces inference cost. Our ShaRNN architecture is orthogonal to MI-RNN and can be combined to obtain even higher amount of inference saving. That is, MI-RNN takes the dataset

$\mathcal{Z} = \{(X_1, y_1), \ldots, (X_n, y_n)\}$ with X_i being a sequential data point over T steps and produces a new set of points X'_j with labels y'_j, where each X'_j is sequential data point over T' and $T' \leqslant T$. MI-ShaRNN applies ShaRNN to the output of MI-RNN so that the inference cost is dependent only on $T' \leqslant T$, and captures the key signal in each data point.

4 Analysis

In this section, we provide theoretical underpinnings of ShaRNN approach and we also put it in context of work by [23] that discusses RNN models for which we can get rid of almost all of the recurrence.

Let $\mathcal{R} : \mathbb{R}^{d+\hat{d}} \rightarrow \mathbb{R}^{\hat{d}}$ be a standard RNN model that maps the given hidden state $h_{t-1} \in \mathbb{R}^{\hat{d}}$ and data point $x_t \in \mathbb{R}^d$ into the next hidden state $h_t = \mathcal{R}(h_{t-1}, x_t)$. Overloading notation, $\mathcal{R}(h_0, x_1, \ldots, x_t) = \mathcal{R}(h_{t-1}, x_t)$. We define a function to be recurrent if the following holds:

$$\mathcal{R}(h_0, x_1, \ldots, x_t) = \mathcal{R}(\mathcal{R}(h_0, x_1, \ldots, x_{t-1}), x_t).$$

The final class prediction using feed-forward layer is given by: $\hat{y} = f(h_T) = f(\mathcal{R}(h_0, x_{1:T}))$. Now, ShaRNN attempts to untangle and approximate the dependency of $f(h_T)$ and $\mathcal{R}(h_0, x_{1:T})$ on h_0, by using Taylor's theorem. Below claim shows the condition under which the approximation error introduced by ShaRNN is small.

Claim 3. *Let $\mathcal{R}(h_0, x_1, \ldots, x_t)$ be an RNN and let $\|\nabla_h^M \mathcal{R}(h, x_{t:t+k})\| \leqslant O(\epsilon \cdot M!)$ for some $\epsilon \geqslant 0$ where ∇_h^M is Mth order derivative with respect to h. Also let $\|\mathcal{R}(h_0, x_{1:t}) - h_0\| = O(1)$, $\|\nabla_h^m \mathcal{R}(h_0, x_{t+1:t+k})\| = O(m!)$ for all $t \in [T]$. Then, there exists an ShaRNN defined by functions $\mathcal{R}^{(1)}$, $\mathcal{R}^{(2)}$ and brick-size k, s.t.:*

$$\|\mathcal{R}^{(2)}(\nu_1^{(1)}, \ldots, \nu_{T/k}^{(1)}) - \mathcal{R}(h_0, x_{1:T})\| \leqslant \epsilon \cdot M \cdot T, \text{ where } \nu_j^{(1)} = \mathcal{R}^{(1)}(h_0, x_{(j-1)\cdot k+1:j\cdot k}).$$

See Appendix A for a detailed proof of the claim.

The above claim shows that the hidden state computed by ShaRNN is close to the state computed by a fully recursive RNN, hence the final output $\hat{y}$ would also be close. We now compare this result to the result of [23], which showed that $\|\mathcal{R}(h_0, x_{1:T}) - \mathcal{R}(h_0, x_{T-k+1:T})\| \leqslant \epsilon$ for large enough k if $\mathcal{R}$ satisfies a contraction property. That is, if $\|\mathcal{R}(h_{t-1}, x_t) - \mathcal{R}(h'_{t-1}, x_t)\| \leqslant \lambda \|h_{t-1} - h'_{t-1}\|$ where $\lambda < 1$. However, $\lambda < 1$ is a strict requirement and do not hold in practice. Due to this, if we only compute $\mathcal{R}(h_0, x_{T-k+1:T})$ as suggested by the above result (for some reasonable values of k), then resulting accuracy on several datasets drops significantly (see Figure 1(b),(c)).

In the context of Claim 3, result of [23] is a special case with $M = 1$, i.e., the result only applies a $0-$th order Taylor series expansion. Figure 1 (d) shows how norm of the gradient that bounds error due to the 0-th order expansion is significantly larger than the norm of the Hessian which bounds error due to the 1-st order expansion.

Case study with FastRNN: We now instantiate Claim 3 for a simple FastRNN model [21] with a first-order approximation i.e., with $M = 2$ in Claim 3.

Claim 4. *Let $\mathcal{R}(h_0, x_1, \ldots, x_t)$ be a FastRNN model with parameters U, W. Let $\|U\| \leqslant O(1)$, $\|\nabla_h^2 \mathcal{R}(h_0, x_{t:t+k})\| \leqslant O(\epsilon)$ for any k-length sequence. Then, there exists an ShaRNN defined by functions $\mathcal{R}^{(1)}$, $\mathcal{R}^{(2)}$ and brick-size k s.t.: $\|\mathcal{R}^{(2)}(\nu_1^{(1)}, \ldots, \nu_{T/k}^{(1)}) - \mathcal{R}(h_0, x_{1:T})\| \leqslant \epsilon$, where $\nu_j^{(1)} = \mathcal{R}^{(1)}(h_0, x_{(j-1)\cdot k+1:j\cdot k})$.*

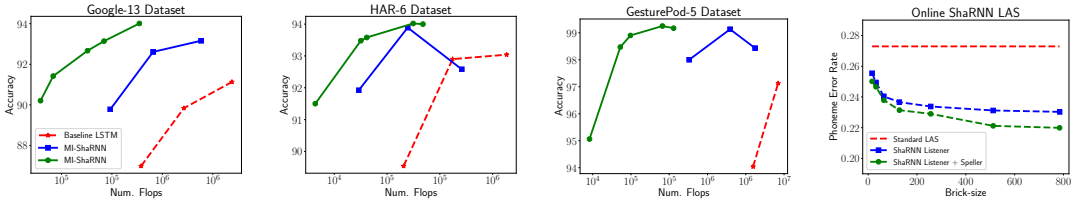
Note that $\|U\| = O(1)$ holds for all the benchmarks that were tried in [21]. Moreover, this assumption is significantly weaker than the typical $\|U\| < 1$ assumption required by [23]. Finally, the Hessian term is significantly smaller than the derivative term (Figure 1 (d)), hence the approximation error and prediction error should be significantly smaller than the one we would get by 0-th order approximation (see Figure 1 (b), (c)).

5 Empirical Results

We conduct experiments to study: a) performance of MI-ShaRNN with varying hidden state dimensions at both the layers $\mathcal{R}^{(1)}$ and $\mathcal{R}^{(2)}$ to understand how its accuracy stacks up against baseline

Table 1: Table compares maximum accuracy achieved by each of the method for different model sizes, i.e., different hidden-state sizes indicated by numbers in bracket; MI-ShaRNN reports two numbers for the first and the second layer, respectively. Table also reports the corresponding computational cost (amortized number of flops required per data point inference) for each method. T denotes the no. of time-steps for the dataset, T' denotes the trimmed number of time-steps obtained by MI-RNN, k is the selected brick-length for MI-ShaRNN. Note that for all but one datasets MI-ShaRNN is able to achieve similar or better accuracy compared to baseline LSTMs.

Dataset	**Baseline LSTM**			**MI-RNN**			**MI-ShaRNN**		
	Acc(%)	Flops	T	Acc(%)	Flops	T'	Acc(%)	Flops	k
Google-13	91.13 (64)	4.89M	99	93.16 (64)	2.42M	49	**94.01** (64, 32)	0.59M	8
HAR-6	93.04 (32)	1.36M	128	91.78 (32)	0.51M	48	**94.02** (32, 8)	0.17M	16
GesturePod-5	97.13 (48)	8.37M	400	98.43 (48)	4.19M	200	**99.21** (48, 32)	0.83M	20
STCI-2	99.01 (32)	2.67M	162	98.43 (32)	1.33M	81	**99.23** (32, 32)	0.30M	8
DSA-19	85.17 (64)	7.23M	129	**88.11** (64)	5.05M	90	87.36 (64, 48)	1.10M	15



(a) (b) (c) (d)

Figure 2: (a),(b),(c): Accuracy vs inference cost: we vary model size (hidden dimensions) to obtain accuracy vs inference cost curve for different methods. All the three plots show that MI-ShaRNN produces more accurate models with as much as $8-10$x reduction in the inference cost. (d): Error-rate of standard LAS method [12] and of ShaRNN based streaming LAS with varying brick-sizes k on the TIMIT [13] dataset. We report results when both Listener+Speller use ShaRNN vs when only Listener uses it. ShaRNN Listener+Speller with $k = 64$ incurs 12x smaller lag in phoneme prediction vs baseline LAS ($k = 784$).

models across different model sizes, b) inference cost improvement that MI-ShaRNN produces for standard time-series classification problems over baseline models and MI-RNN models, c) if MI-ShaRNN can enable certain time-series classification tasks on devices based on the tiny Cortex M4 with only 100MHz processor and 256KB RAM. Recall that MI-ShaRNN uses ShaRNN on top of trimmed data points given by MI-RNN. MI-RNN is known to have better performance than baseline LSTMs, so naturally MI-ShaRNN has better performance than ShaRNN. Hence, we present results for MI-ShaRNN and compare them to MI-RNN to demonstrate advantage of ShaRNN technique.

Datasets: We benchmark our method on standard datasets from different domains like audio keyword detection (Google-13), wake word detection (STCI-2), activity recognition (HAR-6), sports activity recognition (DSA-19), gesture recognition (GesturePod-5). The number after hyphen in dataset name indicates the number of classes in the dataset. See Table 3 in appendix for more details about the datasets. All the datasets are available online (see Table 3) except STCI-2 which is a proprietary wake word detection dataset.

Baselines: We compare our algorithm MI-ShaRNN (LSTM) against the baseline LSTM method as well as MI-RNN (LSTM) method. Note that MI-RNN as well as MI-ShaRNN build upon an RNN cell. For simplicity and consistency, we have selected LSTM as the base cell for all the methods, but we can train each of them with other RNN cells like GRU [7] or FastRNN [21]. We implemented all the algorithms on TensorFlow and used Adam for training the models [19]. The inference code for Cortex M4 device was written in C and compiled onto the device. All the presented numbers are averaged over 5 independent runs. The implementation of our algorithm is released as part of the EdgeML [11] library.

Hyperparameter selection: The main hyperparameters are: a) hidden state sizes for both the layers of MI-ShaRNN. b) brick-size k for MI-ShaRNN. In addition, the number of time-steps T is associated with each dataset. MI-RNN prunes down T and works with $T' \leqslant T$ time-steps. We provide results

Table 2: Deployment on Cortex M4: accuracy of different methods vs inference time cost (ms) on M4 device with 256KB RAM and 100MHz processor. For low-latency keyword spotting (Google-13), the total inference time budget is 120 ms.

	Baseline		**MI-RNN**		**MI-ShaRNN**	
	16	32	16	32	(16, 16)	(32,16)
Acc.	86.99	89.84	89.78	92.61	**91.42**	**92.67**
Cost	456	999	226	494	**70.5**	**117**

with varying hidden state sizes to illustrate trade-offs involved with selecting this hyperparameter (Figure 2). We select $k \approx \sqrt{T}$ with some variation to optimize w.r.t the stride length ω for each dataset; we also provide an ablation study to illustrate impact of different choices of k on accuracy and the inference cost (Figure 3, Appendix).

Comparison of accuracies: Table 1 compares accuracy of MI-ShaRNN against baselines and MI-RNN for different hidden dimensions at $\mathcal{R}^1$ and $\mathcal{R}^2$. In terms of prediction accuracies, MI-ShaRNN performs much better than baselines and is competitive to MI-RNN on all the datasets. For example, with only $k = 8$, MI-ShaRNN is able to achieve 94% accuracy on the Google-13 dataset while MI-RNN model is applied for $T = 49$ steps and baseline LSTM for $T = 99$ steps. That is, with only 8-deep recurrence, MI-ShaRNN is able to compete with accuracies of 49 and 99 deep LSTMs.

For inference cost, we study the amortized cost per data point in the sliding window setting (See Section 3). That is, baseline and MI-RNN for each sliding window recomputes the entire prediction from scratch. But, MI-ShaRNN can re-use computation in the first layer (see Section 3) leading to significant saving in inference cost. We report inference cost as the additional floating point operations (flops) each model would need to execute for every new inference. For simplicity, we treat both addition and multiplication to be of same cost. The number of non-linearity computations are small and are nearly same for all the methods so we ignore them.

Table 1 clearly shows that to achieve best accuracy, MI-ShaRNN is up to 10x faster than baselines and up to 5x faster than MI-RNN, even on a single threaded hardware architecture. Figure 2 shows computation vs accuracy trade-off for three datasets. We observe that for a range of desired accuracy values, MI-ShaRNN is 5-10x faster than the baselines.

Next, we compute accuracy and flops for MI-ShaRNN with different brick sizes k (see Figure 3 of Appendix). As expected, $k \sim \sqrt{T}$ setting requires fewest flops for inference, but the story for accuracy is more complicated. For this dataset, we do not observe any particular trend for accuracy; all the accuracy values are similar, irrespective of k.

Deployment of Google-13 on Cortex M4: we use ShaRNN to deploy a *real-time* keyword spotting model (Google-13) on a Cortex M4 device. For time series classification (Section 3), we will need to slide windows and infer classes on each window. Due to small working RAM of M4 devices ($256KB$), for real-time recognition, the method needs to finish the following tasks within a budget of 120ms: collect data from the microphone buffer, process them, produce ML based inference and smoothened out predictions for one final output.

Standard LSTM models for this task work on 1s windows, whose featurization generates a 32×99 feature vector; here $T = 99$. So, even a relatively small LSTM (hidden size 16), takes on 456ms to process one window, exceeding the time budget (Table 2). MI-RNN is faster but still requires 225ms. Recently, a few CNN based methods have also been designed for low-resource keyword spotting [26, 20]. However, with just 40 filters applied to the standard 32×99 filter-bank features, the working memory requirement balloons up to $\approx$ 500KB which is beyond typical M4 devices' memory budget. Similarly, compute requirement of such architectures also easily exceed the latency budget of 120ms. See Figure 4, in the Appendix for a comparison between CNN models and ShaRNN.

In contrast, our method is able to produce inference in only 70ms, thus is well-within latency budget of M4. Also, MI-ShaRNN holds two arrays in the working RAM: a) input features for 1 brick and b) buffered final states from previous bricks. For the deployed MI-ShaRNN model, with timesteps $T = 49$, brick-size $k = 8$ working RAM requirement is just 1.5 KB.

ShaRNN for Streaming Listen Attend Spell (LAS): LAS is a popular architecture for phoneme classification in given audio stream. It forms non-overlapping time-windows of length 784 ($\approx$ 8 seconds) and apply an encoder-decoder architecture to predict a sequence of phonemes. We study

LAS applied to TIMIT dataset [13]. We enhance the standard LAS architecture to exploit time-annotated ground truth available in TIMIT dataset, which improved baseline phoneme error rate from publicly reported 0.271 to 0.22. Both Encoder and Decoder layer in standard and enhanced LAS consists of fully recurrent bi-LSTMs. So for each time window (of length 784) we would need to apply entire encoder-decoder architecture to predict the phoneme sequence, implying a potential lag of ≈ 8 seconds (784 steps) in prediction.

Instead, using ShaRNN we can divide both the encoder and decoder layer in bricks of size k. This makes it possible to give phoneme classification for every k steps of points thereby bringing down lag from 784 steps to k steps. However, due to small brick size k, in principle we might lose significant amount of context information. But due to the corrective second layer in ShaRNN (Figure 1) we observe little loss in accuracy. Figure 2 shows performance of two variants of ShaRNN + LAS: a) ShaRNN Listener that uses ShaRNN only in encoding layer, b) ShaRNN Listener + Speller that uses ShaRNN in both the encoding and decoding layer. Figure 2 (d) shows that using ShaRNN in both the encoder and decoder is more beneficial than using it only in encoder layer. Furthermore, decreasing k from 784 to 64 leads to marginal increase in error from 0.22 to 0.238 while reducing the lag significantly; from 8 seconds to 0.6 seconds. In fact, even at $k = 64$ this model's performance is significantly better than the reported error of standard LAS (0.27) [12]. See Appendix C for details.

References

[1] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013.

[2] James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. Quasi-recurrent neural networks. *arXiv preprint arXiv:1611.01576*, 2016.

[3] Víctor Campos, Brendan Jou, Xavier Giró i Nieto, Jordi Torres, and Shih-Fu Chang. Skip RNN: Learning to skip state updates in recurrent neural networks. In *International Conference on Learning Representations*, 2018.

[4] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.

[5] Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael Witbrock, Mark A Hasegawa-Johnson, and Thomas S Huang. Dilated recurrent neural networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, 2017.

[6] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[7] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[8] Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. Capacity and trainability in recurrent neural networks. *arXiv preprint arXiv:1611.09913*, 2016.

[9] Yann Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *ICML*, 2017.

[10] Don Kurian Dennis, Chirag Pabbaraju, Harsha Vardhan Simhadri, and Prateek Jain. Multiple instance learning for efficient sequential data classification on resource-constrained devices. In *NeurIPS*, 2018.

[11] Don Kurian Dennis, Sridhar Gopinath, Chirag Gupta, Ashish Kumar, Aditya Kusupati, Shishir G Patil, Harsha Vardhan Simhadri. EdgeML: Machine Learning for resource-constrained edge devices.

[12] Janna Escur. Exploring automatic speech recognition with tensorflow. Master's thesis, 2018.

[13] John S Garofolo, Lori F Lamel, William M Fisher, Jonathan G Fiscus, David S Pallett, Nancy L Dahlgren, and Victor Zue. Darpa timit acoustic phonetic continuous speech corpus, 1993.

[14] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann Dauphin. Convolutional sequence to sequence learning. In *ICML*, 2017.

[15] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013.

[16] Michiel Hermans and Benjamin Schrauwen. Training and analysing deep recurrent neural networks. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, 2013.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8), 1997.

[18] Herbert Jaeger, Mantas Lukosevicius, Dan Popovici, and Udo Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural networks : the official journal of the International Neural Network Society*, 20, 2007.

[19] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[20] Rajath Kumar, Vaishnavi Yeruva, and Sriram Ganapathy. On convolutional lstm modeling for joint wake-word detection and text dependent speaker verification. *Proc. Interspeech 2018*, 2018.

[21] Aditya Kusupati, Manish Singh, Kush Bhatia, Ashish Kumar, Prateek Jain, and Manik Varma. Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In *NeurIPS*, 2018.

[22] Benoît Latré, Bart Braem, Ingrid Moerman, Chris Blondia, and Piet Demeester. A survey on wireless body area networks. *Wireless Networks.*, 2011.

[23] John Miller and Moritz Hardt. When recurrent models don't need to be recurrent. *arXiv preprint arXiv:1805.10369*, 4, 2018.

[24] Shishir G. Patil, Don Kurian Dennis, Chirag Pabbaraju, Nadeem Shaheer, Harsha Vardhan Simhadri, Vivek Seshadri, Manik Varma, and Prateek Jain. Gesturepod: Enabling on-device gesture-based interaction for white cane users. Technical report, New York, NY, USA, 2019.

[25] C. Perera, C. H. Liu, and S. Jayawardena. The emerging internet of things marketplace from an industrial perspective: A survey. *IEEE Transactions on Emerging Topics in Computing*, Dec 2015.

[26] Tara N Sainath and Carolina Parada. Convolutional neural networks for small-footprint keyword spotting. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[27] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013.

[28] Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms, 2015.

[29] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *SSW*, 2016.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

[31] A. Y. Yang, S. Iyengar, P. Kuryloski, and R. Jafari. Distributed segmentation and classification of human actions using a wearable motion sensor network. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2008.

[32] Jiong Zhang, Qi Lei, and Inderjit S Dhillon. Stabilizing gradients for deep neural networks via efficient svd parameterization. *arXiv preprint arXiv:1803.09327*, 2018.

A Proofs

Proof of Claim 1. Recall that for streaming setting, sliding windows X^s can then be broken intro bricks $B_j = x_{((j-1)\cdot k+1):(j\cdot k)}$ where $(s-1)\cdot q+1 \leqslant j \leqslant (s-1)\cdot q+T/k$. Now first layer of ShaRNN compute $\nu_j^{(1)}$ for all j. Hence, for the next sliding window $X^{s+1} = x_{s\cdot\omega+1:s\cdot\omega+T}$, we can reuse $\nu_j^{(1)}$ from the previous window where $s\cdot q+1 \leqslant j \leqslant (s-1)\cdot q+T/k$. Note that the second layer would still need to be computed from scratch. Hence, for new window X^{s+1}, we need to compute $\mathcal{R}^{(1)}$ over $\omega = q\cdot k$ new steps. Furthermore, $\mathcal{R}^{(2)}$ needs to be computed over T/k steps. So the total compute requirement is: $\left(\frac{T}{k}+q\cdot k\right)\cdot C_1$. Second part of Claim follows by setting $k=\sqrt{T/q}$. □

We note that Claim 2 is for multi layer ShaRNN case and it agrees with Claim 1 if $L=1$.

Proof of Claim 3. Define,

$$\nu_j^{(1)} = vec([\mathcal{R}(h_0, x_{t:t+k-1}); \nabla_h^1\mathcal{R}(h_0, x_{t:t+k-1}); \ldots; \frac{1}{M!}\nabla_h^{M-1}\mathcal{R}(h_0, x_{t:t+k-1})]), \quad (1)$$

where $t=(j-1)\cdot k+1$ and $j\in[T/k]$. Using Claim 5, $\nu_j^{(1)}$ is a recurrent function of h_0, x_i's, and can be computed by an RNN $\mathcal{R}^{(1)}$ applied to $x_{t:t+k-1}$ and h_0.

Similarly, define:

$$\nu_j^{(2)} = \mathcal{R}(h_0, x_{t:t+k-1}) + \sum_{m=1}^{M-1}\frac{1}{m!}\nabla_h^m\mathcal{R}(h_0, x_{t:t+k})\cdot(\nu_{j-1}^{(2)}-h_0)^{\otimes m} + \frac{1}{M!}\nabla_h^M\mathcal{R}(\zeta, x_{t:t+k})\cdot(\nu_{j-1}^{(2)}-h_0)^{\otimes M}, \quad (2)$$

where $\nu_0^{(2)} = h_0$. Note that there exists a simple bi-linear function $\mathcal{R}^{(2)}$ s.t. $\nu_j^{(2)} = \mathcal{R}^{(2)}(\nu_{j-1}^{(2)}, \nu_j^{(1)})$.

Using the assumptions mentioned in the Claim, we will now show that $\nu_j^{(2)} \approx h_t$ for ShaRNN with $\mathcal{R}^{(1)}, \mathcal{R}^{(2)}$ defined above and where $t=j\cdot k$.

Using Taylor's theorem:

$$\mathcal{R}(h_0, x_{1:t+k-1}) = \mathcal{R}(h_0, x_{t:t+k-1}) + \sum_{m=1}^{M-1}\frac{1}{m!}\nabla_h^m\mathcal{R}(h_0, x_{t:t+k})\cdot(h_{t-1}-h_0)^{\otimes m} + \frac{1}{M!}\nabla_h^M\mathcal{R}(\zeta, x_{t:t+k})\cdot(h_{t-1}-h_0)^{\otimes M}, \quad (3)$$

where $\zeta = \lambda h_0 + (1-\lambda)h_{t-1}$ for some $\lambda > 0$.

Using triangular inequality:

$$\|\mathcal{R}(h_0, x_{1:t+k-1}) - \nu_j^{(2)}\| \leqslant \|\frac{1}{M!}\nabla_h^M\mathcal{R}(\zeta, x_{t:t+k-1})\| \times \|(h_{t-1}-h_0)^{\otimes M}\| + \sum_{m=1}^{M-1}\frac{1}{m!}\|\nabla_h^m\mathcal{R}(h_0, x_{t:t+k-1})\| \times \|(h_{t-1}-h_0)^{\otimes M} - (\nu_{j-1}^{(2)}-h_0)^{\otimes M}\|,$$

where $t=(j-1)\cdot k+1$. Using the assumptions of claims along with standard algebraic manipulations, we get:

$$\|\mathcal{R}(h_0, x_{1:t+k-1}) - \nu_j^{(2)}\| \leqslant \epsilon + O(M\epsilon)\|\nu_{j-1}^{(2)} - h_{t-1}\|.$$

The claim now follows by applying the above result recursively for all $j\in T/k$.

□

Claim 5. *If f is a recurrent function, i.e., $f(h_0, x_{t:t+k}) = f(f(x_{t:t+k-1}, h_0), x_{t+k})$. Then, it's higher-order derivatives are also recurrent.*

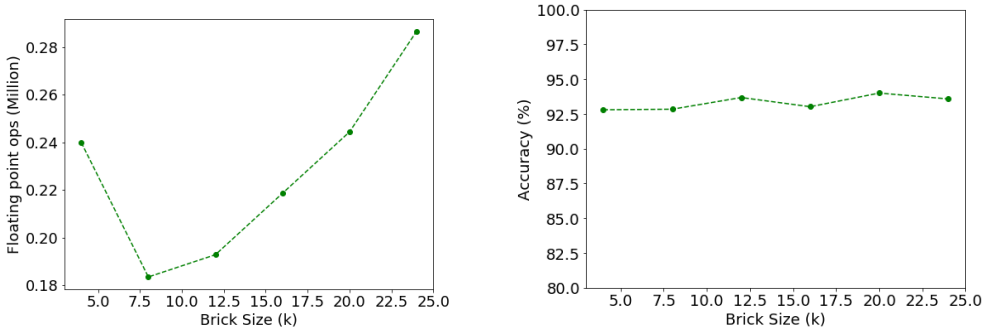
Proof of Claim 4. FastRNN updates hidden state as: $h_t = \alpha \cdot \sigma(Uh_{t-1} + Wx_t + b) + \beta h_{t-1}$ where $\beta \approx 1 - \alpha$, $\alpha = O(1/T)$ and the activation function σ is ReLU. Using the updates, we have: $\|h_t - h_{t-1}\| \leqslant \frac{\|U\|+1}{T}\|h_{t-1}\|$, i.e., $\|h_t\| \leqslant \exp(\|U\| + 1)$ for all t. Now by assumption $\|U\| = O(1)$, we have: $\|h_t\| = O(1)$ for all t. Similarly, $\|\nabla_h \mathcal{R}(h_{t-1}, x_t)\| \leqslant (1 + \frac{\|U\|+1}{T})\|\nabla_h \mathcal{R}(h_{t-2}, x_{t-1})\|$. Using similar arguments as above, we have $\|\nabla_h \mathcal{R}(h_{t-1}, x_t)\| \leqslant O(1)$ for all t. Claim now follows by combining Claim 3 with the bounds on $\|h_t\|$, $\|\nabla_h \mathcal{R}(h_{t-1}, x_t)\|$ and $\|\nabla_h^2 \mathcal{R}(h_{t-1}, x_t)\|$. □

B Additional Empirical Results

Dataset	#Steps (Baseline)	Feat. Dim.	#Train	#Val	#Test	Source
Google-13	99	32	51088	6798	6835	URL1
HAR-6	128	9	6220	1132	2947	URL2
STCI-2	162	32	42788	5223	5224	Proprietary
DSA-19	129	45	4560	2280	2280	URL3
GesturePod-5	400	6	13432	2684	2552	URL4
TIMIT	784	39	4389	231	1680	URL5

Table 3: Dataset details: Source of dataset, the number of timesteps, feature dimension and the number of data points in train, test and validation tests.

URL1 `http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz`
URL2 `https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones`
URL3 `https://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities`
URL4 `https://www.microsoft.com/en-us/research/publication/gesturepod-programmable-gesture-recognition-augmenting-assistive-devices/`
URL5 `https://catalog.ldc.upenn.edu/LDC93S1`



(a) (b)

Figure 3: Accuracy and inference cost vs brick size (k) on HAR-6 dataset for a model with hidden dimension 32 at both the layers. Inference cost in terms of number of floating point operations (flops) behaves as expected as show in in (a). The accuracy trend, shown in (b), is tricky at extreme values of k. When k is very small, the lower layer is very shallow, while for high values of k, the higher layer becomes shallow.

A comparison of CNN based models and ShaRNN is given in Figure 4 for Google-13. Note that *none* of the CNN models satisfy compute requirement of $\leqslant 0.15M$ flops on M4 device. The best CNN model that at least satisfies the RAM requirement ($< 256KB$) is 3% less accurate than ShaRNN. Interestingly, an RNN even with a small k is more powerful than CNN as it applies non-linearity k times while a CNN layer applies non-linearity only once per k-sized filter.

# Filters	# Pools	Accuracy	Size(KB)	Flops
10	2	0.81	375.1	1.1M
10	4	0.85	90.4	1.5M
20	2	0.83	753.3	3.9M
20	4	0.88	190.1	5.6M
30	4	0.90	299.1	12.1M
ShaRNN	-	**0.91**	**26.5**	**0.09M**

Figure 4: Accuracy, size and number of flops comparison of ShaRNN and CNN models.

C Online LAS with ShaRNN

Listen-Attend-Spell is a popular end-to-end architecture for transcribing speech with phonemes. The architecture consists of two parts— the listener and the speller. The listener is a pyramidal recurrent network which encodes the filter bank spectra input. The speller uses an attention-based recurrent network to decode the listener output and produces phonemes. Standard LAS transcribes audio input with 784 steps which corresponds to about 8secs worth of audio clip. While standard LAS architecture's phoneme error-rate on the TIMIT dataset is 0.27 [4], after a few enhancements like dropping a layer and thresholding out predictions with low confidence, we can achieve the baseline error rate of 0.251.

Now, the LAS architecture is designed to transcribe static input, i.e., where a fixed-length audio clip (of $\leqslant$ 8sec or 784 steps) and does not readily generalize to the streaming setting where the audio data is flowing in continuously. One approach is to form non-overlapping windows of fixed size and apply LAS on each of them independently. Naturally such a technique would incur a large lag in phoneme predictions. Another approach is to form sliding windows, but in that case it is not clear how to reconcile predictions from the overlapping sliding windows.

We focus on the streaming setting and propose an ShaRNN based approach for making the LAS architecture streaming, i.e., with predictions with small lag of say $\leqslant$ 1sec — this has been illustrated in Figure 5. Intuitively, as new batch of audio data arrives, the goal is to process the new batch of data and predict phonemes contained in the batch; note that batch-size should ideally be small so that there is a small lag in prediction. However, as phoneme prediction can be highly contextual, we cannot process every batch independently and would require context from past few batches of audio as well. But, standard LAS architecture is ill-suited for such task, furthermore, naively processing the past few batches would lead to significant computational overhead.

Below we describe our ShaRNN-based architecture that can appropriately re-use computation to ensure accurate phoneme prediction with a small batch of audio thus ensuring prediction with small lag and low computational cost.

C.1 Encoder

We replace the bottom two layers of pyramidal encoder by a 3-layer ShaRNN where the first two layers partitions the input into "bricks" of size l_f while the third layer recaptures the receptive field by processing bottom layer's output via a bi-LSTM. The output of the third layer is the final code/embedding of the input-sequence. Similar to sliding-window streaming setting (Section 3), we can re-use computation from the bricks to process the new l_f-sized brick of audio data efficiently.

C.2 Decoder

We replace the attention-based architecture in LAS with an inverted pyramidal decoder — the number of output states for each of the layer in the inverted pyramidal decoder is twice the number of input states. Thus, after two layers of the inverted pyramidal decode we obtain the same number of output states as the input to the encoder. Each of these output states are then processed by an Multilayer Perceptron (MLP) layer to compute the probability distribution over the space of all phonemes.

[4] [4] did not report results on any publicly available dataset, but this error-rate matches the publicly reported numbers [12]

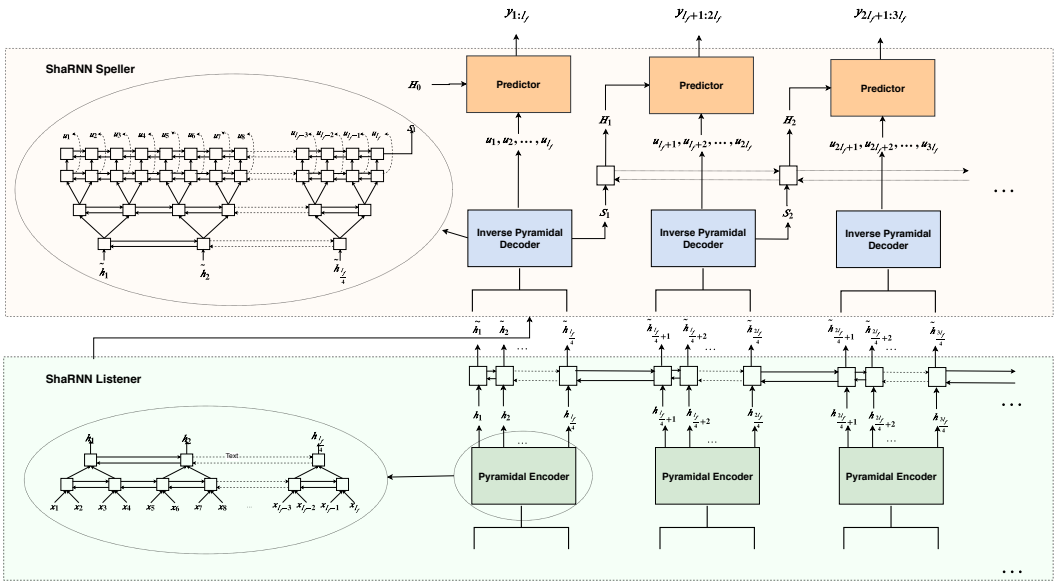


Figure 5: ShaRNN based online LAS.

Applying the above Decoder in the streaming setting will incur significant computation overhead. To alleviate this concern, we again use ShaRNN to enable re-use of the computation across the decoder layer as was done for the encoder layer. In particular, we use one recurrent network to compute the 'summary' of all the output states (denoted as S_i in Figure 5) of a given fragment. Another recurrent network processes the past "summaries" S_{i-1}, S_{i-2} and S_{i-3} to produce H_i which is the 'correction' factor for each of the output states (u_j in Figure 5) of the ith fragment. This correction term concatenated with each u_j is the input to a two layer MLP with softmax output over the phonemes.

Hence, the phoneme distribution is obtained for each new input frame/batch and the network is trained using the time aligned phoneme transcription available in the TIMIT dataset. The final prediction by the model is obtained by removing labels predicted with a low confidence (less than a threshold) and collapsing the repeating phonemes.

C.3 Argument

We first replace the encoder in LAS while retaining the decoder, we see an improvement in the phoneme error rate from 0.251 to 0.240 ($l_f = 64$) by doing this. Using the ShaRNN encoder, the streaming input can be transcribed every l_f input frames, thus there is no need to wait for the entire speech input. Even though the lag for prediction is reduced, this still involves the attention computation across all the encoder states which is expensive especially when the input speech is long and runs into hours. To avoid this, we replace the decoder with an ShaRNN decoder where the need for attention is eliminated by predicting a phoneme for each input frame and not just the unique phonemes. With this substitution, we observe a further improvement in the phoneme error rate to 0.238 ($l_f = 64$).

Surprisingly, it turns out that our new architecture is able to better model the phoneme prediction problem. The error rate for the "offline" version of our model, i.e., where $l_f = 784$ is 0.220. This error-rate is significantly better than the rate of 0.251 that we could obtain using enhancements of the standard LAS model.

As noted above, using our ShaRNN based architecture with $l_f = 64$, we could still achieve error rate of 0.238 which is marginally larger than the best error rate achieved by $l_f = 784$. However, lag in phoneme predictions in $l_f = 64$ case is 12x smaller than the lag incurred by our architecture with $l_f = 784$, i.e., in the offline case.