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

Don Kurian Dennis, Durmus Alp Emre Acar, Vikram Mandikal, Vinu Sankar Sadasivan, Harsha Vardhan Simhadri, Venkatesh Saligrama, Prateek Jain

*Slides to be updated.*
Outline

• Introduction
• Background
• Shallow RNNs
• Results
Introduction

- Time series classification:
  - Detecting events in a continuous **stream** of data.
  - Data partitioned into overlapping windows (sliding windows).
  - Detection/Classification performed on each window.
Introduction

• Time Series on Tiny Devices:
  • Resource scarcity (few KBs of RAM, tiny processors)
  • Cannot run standard DNN techniques.

• Examples:
  • Interactive cane for people with visual impairment [24]:
    • Recognizes gestures coming as time-traces on a sensor. 32kB RAM, 40MHz Processor.
  • Audio-keyword classification on MXChip:
    • Detect speech commands and keywords. 100MHz processor, 256KB RAM.
Background

• How to solve time series problem on tiny devices
  • RNNs:
    • Good fit for time series problems with long dependencies,
    • Smaller models, but no parallelization [28, 14], requires $O(T)$ time. Small but too Slow!
  • CNNs:
    • Can be adapted to time series problems.
    • Higher parallelization [28, 14] but much larger working RAM. Fast but too big!
Shallow RNN - ShaRNN

- ✔ Parallelization
- ✔ Small Size
- ✔ Compute Reuse
Shallow RNN - ShaRNN

- Hierarchical collection of RNNs organized at two levels.
- Output of first layer is the input of second layer.
- $x_{1:T}$ data is split into bricks of size $k$. 

![Diagram of ShaRNN structure](image)
Shallow RNN - ShaRNN

- $\mathcal{R}^{(1)}$ RNN is applied to each brick:
  - $v_i^{(1)}: \mathcal{R}^{(1)}$ outputs.

- $\mathcal{R}^{(1)}$ bricks:
  - Operate completely in parallel,
  - Fully shared parameters.
Shallow RNN - ShaRNN

- $k$ is hyperparameter:
  - Controls inference time.
- $\mathcal{R}^{(1)}$ bricks on $k$ length series
- $\mathcal{R}^{(2)}$ bricks on $\frac{T}{k}$ length series
- Overall $O\left(\frac{T}{k} + k\right)$ inference time.
- If $k = O\left(\sqrt{T}\right)$:
  - Overall time is $O\left(\sqrt{T}\right)$ instead of $O(T)$
Our method is able to achieve similar or better accuracy compared to baselines in all but one datasets.

Different model sizes (different hidden-state sizes) $\rightarrow$ numbers in bracket,
- MI-ShaRNN reports two numbers for the first and the second layer.

Computational cost (amortized number of flops required per data point inference) for each method.
- MI refers to method of [10] which leads to smaller models and it is orthogonal to ShaRNN.

### Results - Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Baseline LSTM</th>
<th>MI-RNN</th>
<th>MI-ShaRNN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc(%)</td>
<td>Flops</td>
<td>$T$</td>
</tr>
<tr>
<td>Google-13</td>
<td>91.13 (64)</td>
<td>4.89M</td>
<td>99</td>
</tr>
<tr>
<td>HAR-6</td>
<td>93.04 (32)</td>
<td>1.36M</td>
<td>128</td>
</tr>
<tr>
<td>GesturePod-5</td>
<td>97.13 (48)</td>
<td>8.37M</td>
<td>400</td>
</tr>
<tr>
<td>STCI-2</td>
<td>99.01 (32)</td>
<td>2.67M</td>
<td>162</td>
</tr>
<tr>
<td>DSA-19</td>
<td>85.17 (64)</td>
<td>7.23M</td>
<td>129</td>
</tr>
</tbody>
</table>
Results - Deployment

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th></th>
<th>MI-RNN</th>
<th></th>
<th>MI-ShaRNN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16</td>
<td>32</td>
<td>16</td>
<td>32</td>
<td>(16, 16)</td>
</tr>
<tr>
<td>Acc.</td>
<td>86.99</td>
<td>89.84</td>
<td>89.78</td>
<td>92.61</td>
<td><strong>91.42</strong></td>
</tr>
<tr>
<td>Cost</td>
<td>456</td>
<td>999</td>
<td>226</td>
<td>494</td>
<td><strong>70.5</strong></td>
</tr>
</tbody>
</table>

- Accuracy of different methods vs inference time cost (ms).
- Deployment on Cortex M4:
  - 256KB RAM and 100MHz processor,
  - The total inference time budget is 120 ms.
- Low-latency keyword spotting (Google-13).
Demo Video Here: dkdennis.xyz/static/sharnn-neurips19-demo.mp4
Thank you!