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

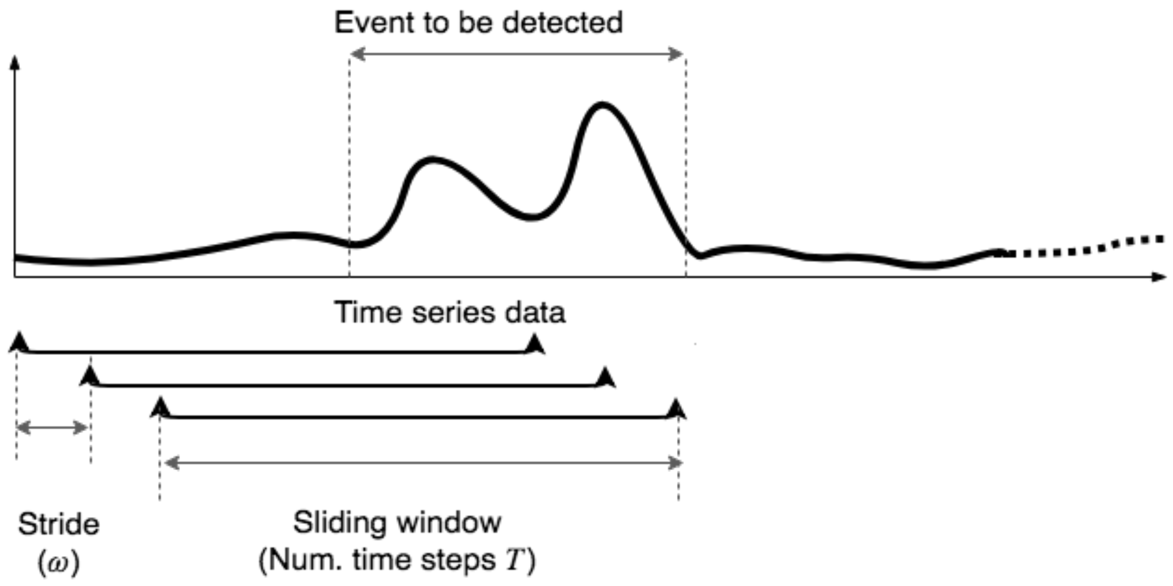
**Slides to be updated.*

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 scarscity (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

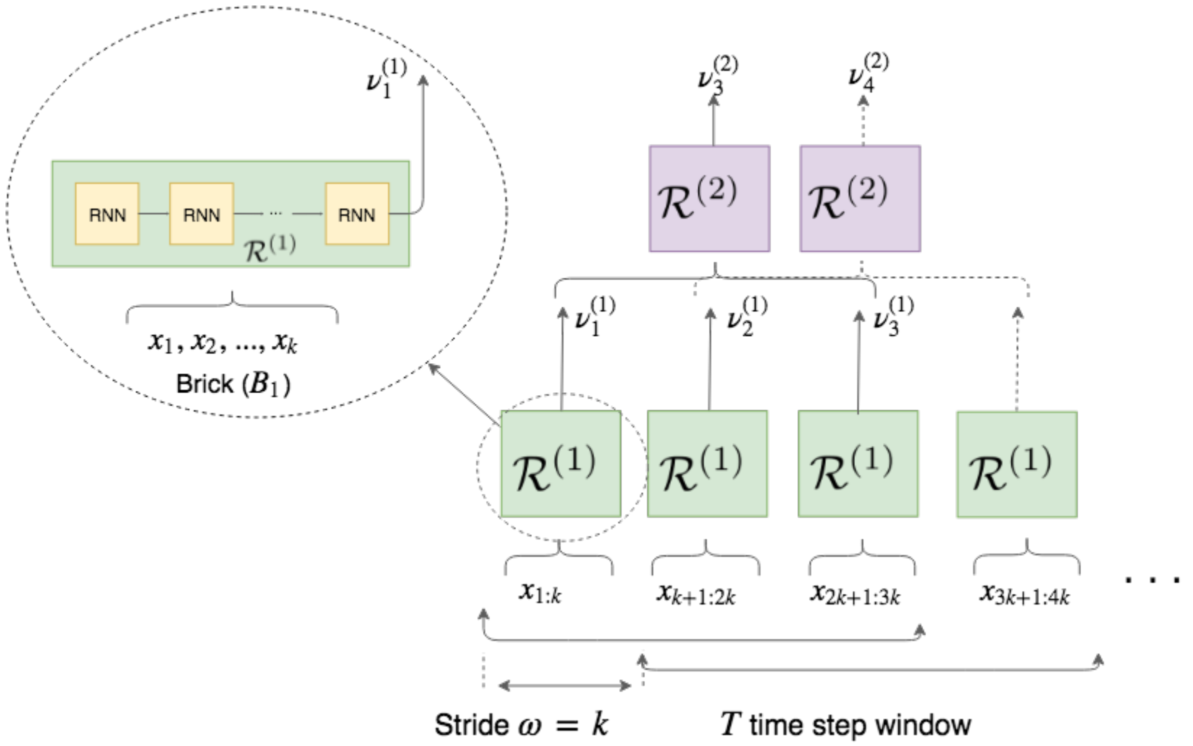
- How to solve time series problem on tiny devices
  - RNNs:
    - Good fit for time series problems with long dependencies,
    - Smaller models, but no parallelization [28, 14], requires *O(T)* time. Small but too Slow!
  - CNNs:
    - Can be adapted to time series problems.
    - Higher parallelization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

Shallow RNN - ShaRNN

- ✓ Parallelization
- ✓ Small Size
- ✓ Compute Reuse

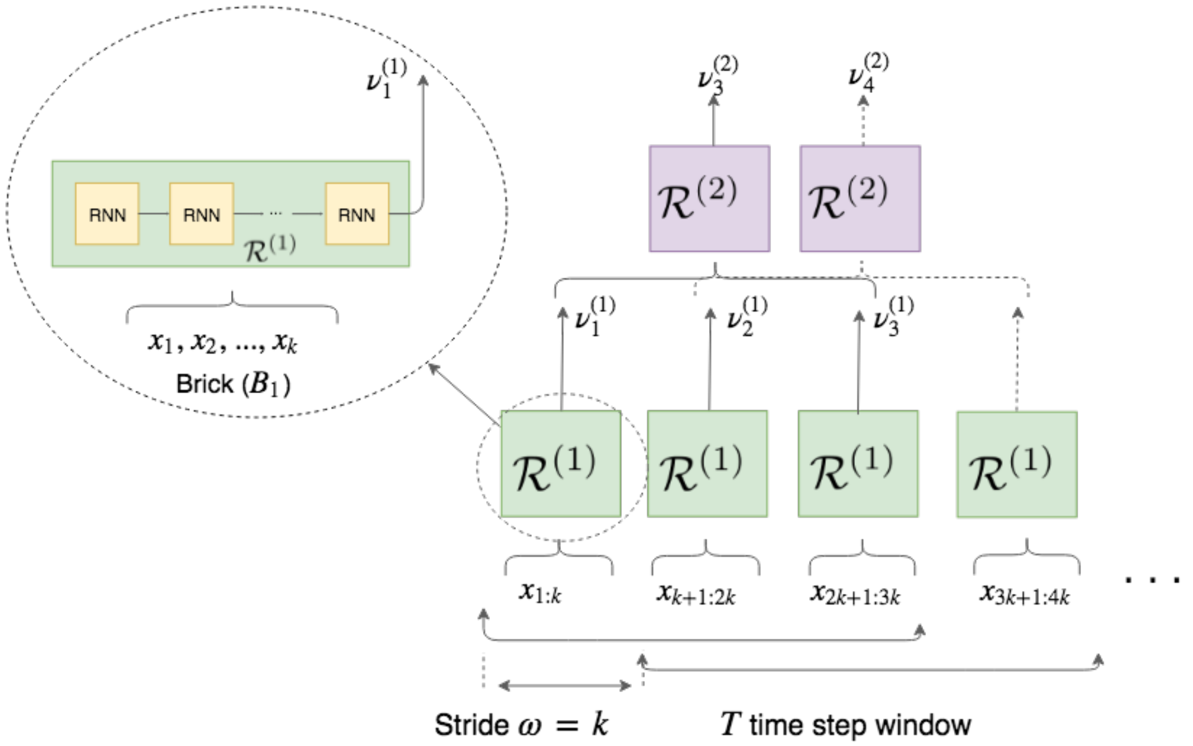
Shallow RNN - ShaRNN

- 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.



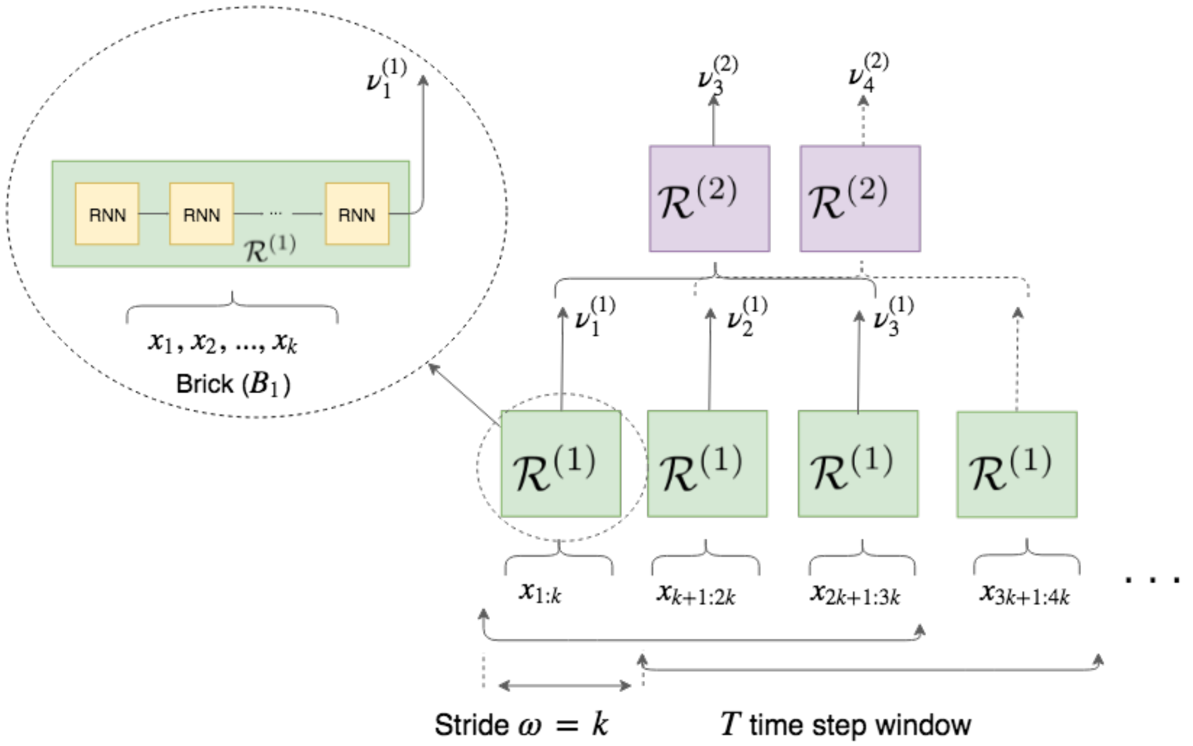
Shallow RNN - ShaRNN

- $\mathcal{R}^{(1)}$ RNN is applied to each brick:
  - $\upsilon_i^{(1)}$: $\mathcal{R}^{(1)}$ 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 $\frac{\mathrm{T}}{\mathrm{k}}$ length series
- Overall $O(\frac{\mathrm{T}}{\mathrm{k}} + \mathrm{k})$ inference time.
- If $k = O(\sqrt{T})$:
  - Overall time is $O(\sqrt{T})$ instead of O(T)



Results - Datasets

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

- 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) -> 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 - Deployment

	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**

- 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!