[7064] Shallow RNN: Accurate Time-series Classification on Resource Constrained Devices

We thank the reviewers for their comments; we will make the suggested improvements and fix the minor typos.

Reviewer 1: *Model size compared to standard RNN (L41)*: We meant that SRNN is able to maintain the same model size as an RNN while increasing parallelizability.

GesturePod-5 vs GesturePod-6: Both refer to the same dataset and this is a typo.

S-RNN and MI-RNN (L199): Without MI-RNN also, SRNN performs well compared to baseline (e.g. on Google-13, it achieves 1% higher accuracy despite 9x reduction in flops). We will add these results and more details in supplementary.

Latency budget: This is mostly dependent on the stride length that in turn is task specific. For keyword spotting, 100-150ms strides are known to be sufficient for achieving reasonable accuracy.

Variance across random seeds: We observed little variance across seeds. We will add results with confidence interval bounds in supplementary material.

T, w and k in Claim 1: We are indeed given T and ω, and Claim 1 sets k as: $k = T/\omega$ (up to a few minor corner cases).

Reviewer 2: *Re-running RNN on each window:* This is standard practise in time-series classification as in this domain the RNNs usually are trained on fixed length windows and do not handle varying window lengths accurately. For example, in the keyword spotting problem with the Google-13 dataset, if we run RNN for 1.5 secs instead of 1 sec windows (length of training windows), the accuracy drops by >5%.

Streaming SRNN: For incoming streams, partitions are distributed. We provided intuitive explanation in L159-168. At a high level, we point out that in the streaming setting SRNN is able to reuse computation (by reusing $\nu_j^{(1)}$'s) causing the amortized cost to comes down.

Claim 1: It claims that for a given $\omega = q \cdot k$, $k = \sqrt{T/q}$ is optimal. That is, set $k = T/\omega$. Hence, if $\omega = \sqrt{T}$ then $k = \sqrt{T}$ which is the setting when SRNN achieves best speed-up over vanilla sliding window RNNs.

Claim 2: Claim 2 is for multi-layer SRNN while Claim 1 is for 2-layer SRNN. k values match for $L = 1$.

Claim 3: ∇_h^M is the M-th order derivative wrt h; we will define it in the next draft.

Claim 3, 4: These claims provide an indication as to why SRNN is able to achieve comparable performance to standard RNNs in practice despite smaller recurrence. That is, we show that if the M-th order derivative of the RNN is small then a specific version of SRNN can reasonably approximate the fully recurrent RNN. These claims are in contrast with claims of [22], that require the 1-st order derivative itself to be small. In Figure 1(b), (c), (d) we provide limited empirical evidence to support our assumption, i.e., we show that 1-st order derivative can be much larger than the 2-nd order and hence approximation error by non-retrained version of SRNN is also relatively small.

S-RNN vs CNN: As mentioned on L296-302, working RAM and computation requirement of CNN based solutions, designed specifically for low-powered devices [25,19], is still too large to fit on devices like the Cortex M4. Here we present a table with more explicit numbers. Note that *none* of the CNN models satisfy compute requirement of $\leq 0.15M$ flops on M4 device. The best CNN model that at least satisfies the RAM requirement ($< 256KB$) is 3% less accurate than SRNN.

No. Filters	No. Pools	Acc.	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
SRNN	-	**0.91**	**26.5**	**0.09M**

SRNN with small k, ω = CNN? Intuitively, 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. Furthermore, in practice we observe that $\omega \approx \sqrt{T}$ and $k \approx \sqrt{T}$, which is larger than typical CNN filters of size $3 - 5$.

Reviewer 3: *Bound in Claim 1:* Yes, from the Claim point of view L137 is problematic as in practice generally $\omega \approx \sqrt{T}$. For $\omega = O(1)$ also, we can get $\sqrt{T}$ amortized cost but that requires a slightly more complicated version of SRNN which we didn't discuss in this paper for ease of exposition. We can add details of the same in the supplementary.

$O(T/k)$ extra memory: Yes extra memory is needed, but with RNNs in the streaming setting, we are latency bound and memory is a smaller issue. For instance, the MXChip used in the included video has a 256KB RAM where as the SRNN model's memory requirement is only 26.5KB and the T/k extra memory turns out to be about $6KB$. We can easily fit the model computation in RAM.

Multilayer S-RNN: It is more beneficial than 2-layer SRNN only for very large value of T; in the type of problems we studied T was large but not large enough to require multi-layer SRNN (L188-190).