Parallelizing Linear Recurrent Neural Nets Over Sequence Length

Eric Martin and Chris Cundy

Fill in institutes

Abstract

RNN training and inference generally takes time linear in the sequence length because of non-linear sequential dependencies. We show the training and inference of RNNs with only linear sequential dependencies can be parallelized over the sequence length using the parallel scan algorithm, leading to rapid training on long sequences even with small minibatch size. We use this insight and a parallel linear recurrence CUDA kernel to accelerate several state of the art RNN architectures by up to 9x and to solve a synthetic sequence classification task with a one million time step dependency.

Background

Large minibatches are necessary for computational performance but create large memory requirements and damage model generalization ability.

Linear RNNs and convolutional models such as strongly typed RNNs, Wavenet, Bytenet, Quasi-RNNs, and simple recurrent units have achieved state of the art results on many sequential tasks with rapid training times.

Given x_t, λ_t can compute $h_t = \lambda_t h_{t-1} + x_t$ for $t = 1 \dots T$ on p processors in $O(T/p + \log(p))$ with the classic parallel scan algorithm. Backpropagation of gradient can also be parallelized with the same algorithm.

3	1	5	0	2	4	2	6	1
3	4	**9**	0	2	**6**	2	8	9
			+9			**+9+6**		
3	4	9	9	11	15	17	23	24

Figure 1: Cumulative sum parallelized over 3 processors

Gated Impulse Linear Recurrence

Given a fast algorithm for evaluating linear recurrences, we introduce a new linear recurrent layer called **gated impulse linear recurrence (GILR)**

$$
\begin{aligned}
g_t &= \sigma(U x_t + b_g) \\
i_t &= \tau(V x_t + b_z) \\
h_t &= g_t \odot h_{t-1} + (1 - g_t) \odot i_t
\end{aligned}
$$

Linear Surrogate RNNs

RNNs have a transition function $s_t = f(s_{t-1}, x_t)$. s_t serves dual roles as a summary of the past as well as the output of the unit. Non-linear f in units such as vanilla RNN and LSTM prevents parallelization over sequence length.

Replacing the summary of the past s_{t-1} with a linear surrogate $\tilde{s}_{t-1}$ allows the easy adaption of any existing RNN architecture for parallel computation. Several existing RNNs can be viewed as linear surrogate RNNs.

The state of an LSTM consists of (c_t, h_t). c_t is already computed by linear recurrence, so a linear surrogate LSTM must only compute a linear $\tilde{h}_t$. A **GILR-LSTM** uses $\tilde{h} = \text{GILR}(x)$

Accelerating Existing Linear Surrogate RNNs

We compare the performance of several recent linear RNNs implemented with the parallel linear recurrence kernel against a baseline serial linear recurrence kernel.

Seq. Len.	SRU	QRNN(2)	QRNN(10)	GILR-LSTM
16	0.28	0.38	0.78	0.61
256	0.84	0.86	0.99	0.91
4,096	1.38	1.18	1.05	0.98
65,536	9.21	6.68	2.05	1.41

Table 1: Parallel kernel speedup for a variety of LS-RNNs. All models use two stacked RNN layers with 256 hidden units, keeping the GPU memory usage constant by fixing $bT = 65,536$ for minibatch size b and sequence length T. QRNN(k) refers to a QRNN with filter size k.

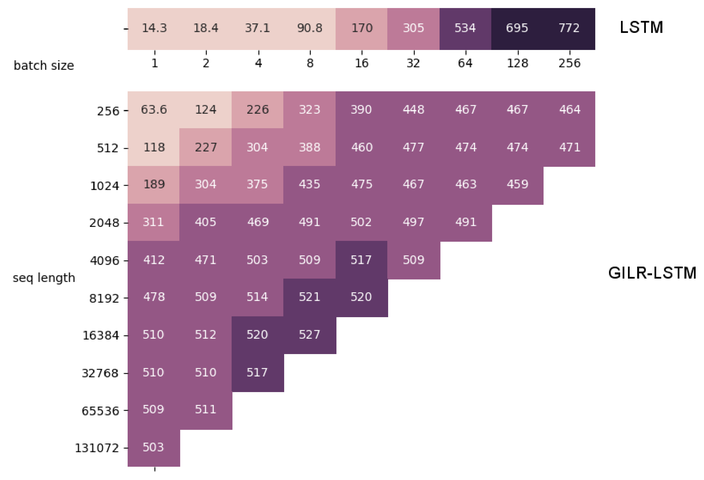

Figure 2: Throughput (thousand steps/s) for same sized LSTM and GILR-LSTM

Synthetic Long-Term Dependency

We tested the GILR-LSTM by comparing to the CuDNN LSTM implementation on a synthetic memorisation task (problem 2b from We compared the performance on variants with different values of n. We obtained speedups of over 6x measured in wall clock time to convergence. Further, the GILR-LSTM attained convergence when the time dependence of the problem had a length of **one million time steps**.

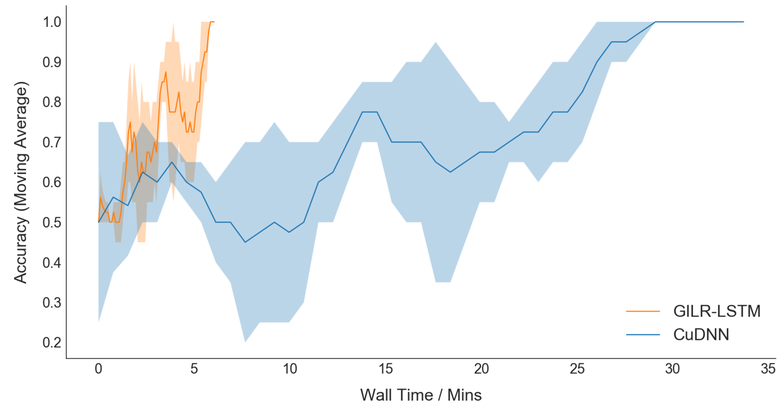

Figure 3: Accuracy on the memorisation task with 8,192 sequence length

Conclusion

Nunc tempus venenatis facilisis. **Curabitur suscipit** consequat eros non porttitor. Sed a massa dolor, id ornare enim. Fusce quis massa dictum tortor **tincidunt mattis**. Donec quam est, lobortis quis pretium at, laoreet scelerisque lacus. Nam quis odio enim, in molestie libero. Vivamus cursus mi at *nulla elementum sollicitudin*.

Additional Information

Maecenas ultricies feugiat velit non mattis. Fusce tempus arcu id ligula varius dictum.

- Curabitur pellentesque dignissim
- Eu facilisis est tempus quis
- Duis porta consequat lorem

References

Acknowledgements