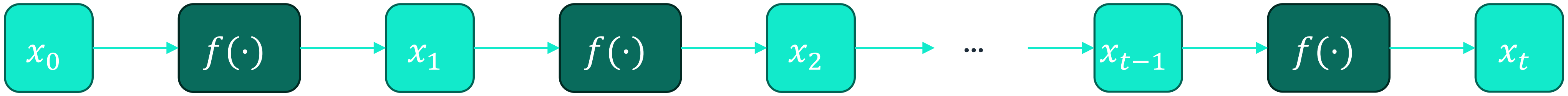
Machine Discovery

RNNs are parallelizable!

Y. H. Lim, Q. Zhu, J. Selfridge, M. F. Kasim

ICLR 2024

Sequential nature of RNN



- The next state depends on the previous states: can't be parallelized

$$x_{t+1} = f(x_t)$$

- Sequential models are slow in GPU/TPU

Many says RNN is not parallelizable

Recurrent models typically factor computation along the symbol positions of the input and output sequences. Aligning the positions to steps in computation time, they generate a sequence of hidden states h_t, as a function of the previous hidden state h_{t-1} and the input for position t. This inherently sequential nature precludes parallelization within training examples, which becomes critical at longer sequence lengths, as memory constraints limit batching across examples. Recent work has achieved significant improvements in computational efficiency through factorization tricks [21] and conditional computation [32], while also improving model performance in case of the latter. The fundamental constraint of sequential computation, however, remains.

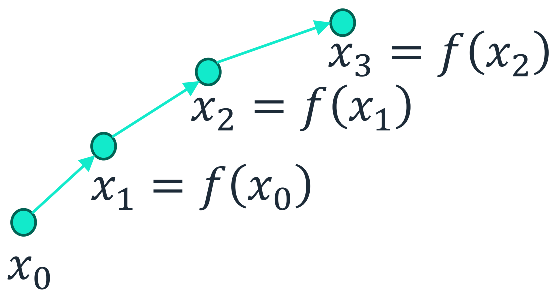
Vaswani *et al.*, Attention is all you need, 2017

Many says RNN is not parallelizable

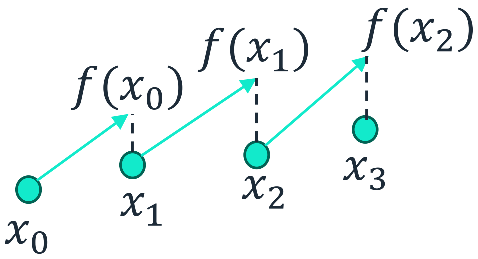
Recurrent neural networks (RNNs) have played a central role since the early days of deep learning, and are a natural choice when modelling sequential data (Elman, 1990; Hopfield, 1982; McCulloch and Pitts, 1943; Rumelhart et al., 1985). However, while these networks have strong theoretical properties, such as Turing completeness (Chung and Siegelmann, 2021; Kilian and Siegelmann, 1996), it is well-known that they can be hard to train in practice. In particular, RNNs suffer from the vanishing and exploding gradient problem (Bengio et al., 1994; Hochreiter, 1991; Pascanu et al., 2013), which makes it difficult for these models to learn about the long-range dependencies in the data. Several techniques were developed that attempt to mitigate this issue, including orthogonal/unitary RNNs (Arjovsky et al., 2016; Helfrich et al., 2018), and gating mechanisms such as long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) and gated recurrent units (GRUs) (Cho et al., 2014a). Nonetheless, these models are still slow to optimize due to the inherently sequential nature of their computation (Kalchbrenner et al., 2016), and are therefore hard to scale.

Orvieto *et al.*, Resurrecting Recurrent Neural Networks for Long Sequences, 2023

Simplest parallel algorithm for sequence: Multi-shoot algorithm with Picard iteration



Sequential
(non-parallelizable)



Multi-shoot
(parallelizable)

Multi-shoot algorithm:

1. Get some initial guess of $x_i^{(0)}$
2. Evaluate the function $f(x_i)$ (**parallelizable**)
3. Update the value of x_i
$$x_i^{(k+1)} \leftarrow f\left(x_{i-1}^{(k)}\right)$$
4. repeat step #2 until $x_{i+1} = f(x_i)$

Simplest update algorithm (Picard iteration):
$$x_i^{(k+1)} \leftarrow f\left(x_{i-1}^{(k)}\right)$$

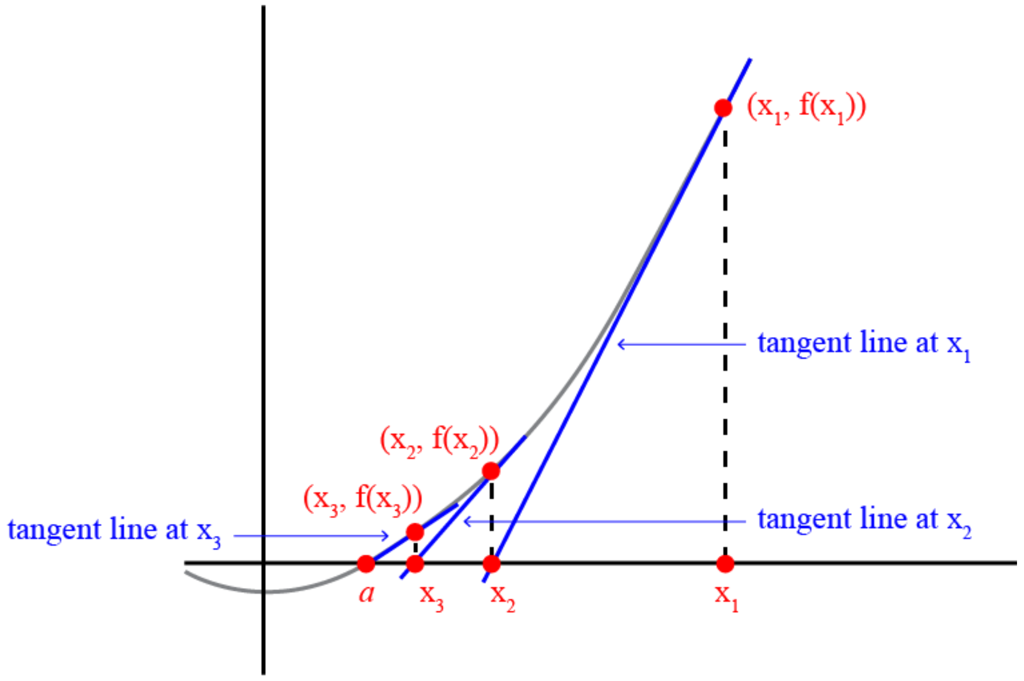
Picard iteration is really bad: rarely converge
Alternative: **Newton's method**

Newton's method for root finder

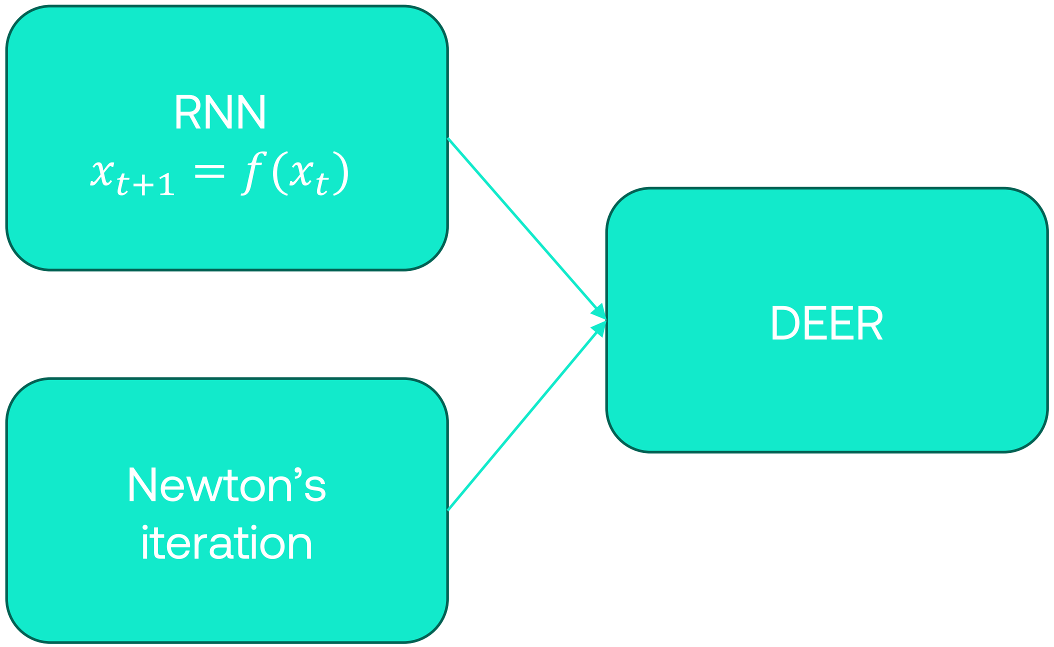
- Finding x so that $f(x) = 0$
- For 1-dimension, iterate:

$$x^{(k+1)} \leftarrow x^{(k)} - \frac{f\left(x^{(k)}\right)}{f'\left(x^{(k)}\right)}$$

- Faster and more robust convergence than Picard iteration

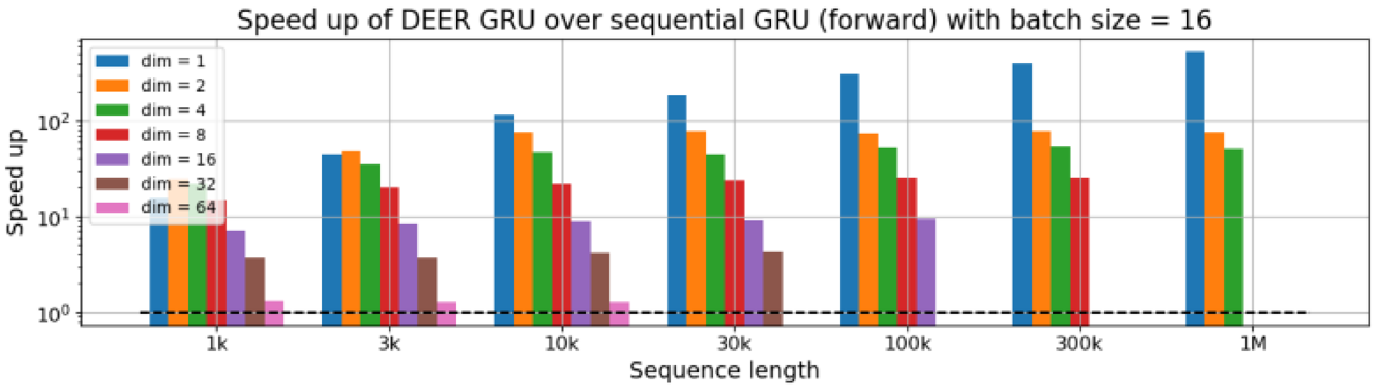


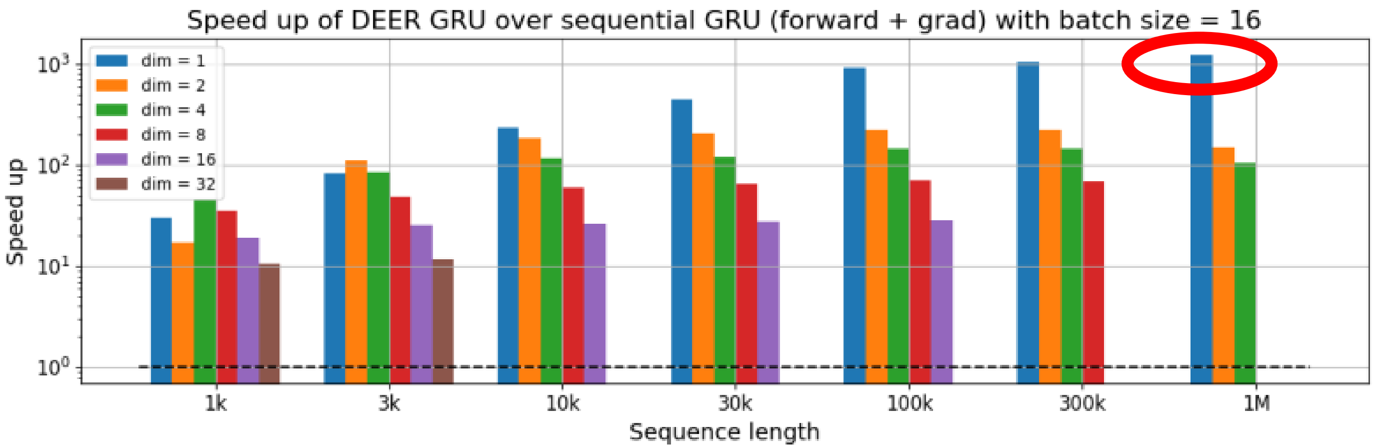
RNN + Newton's iteration: DEER



- Completely parallelizable in GPU
- Details in our paper:
  - https://arxiv.org/abs/2309.12252

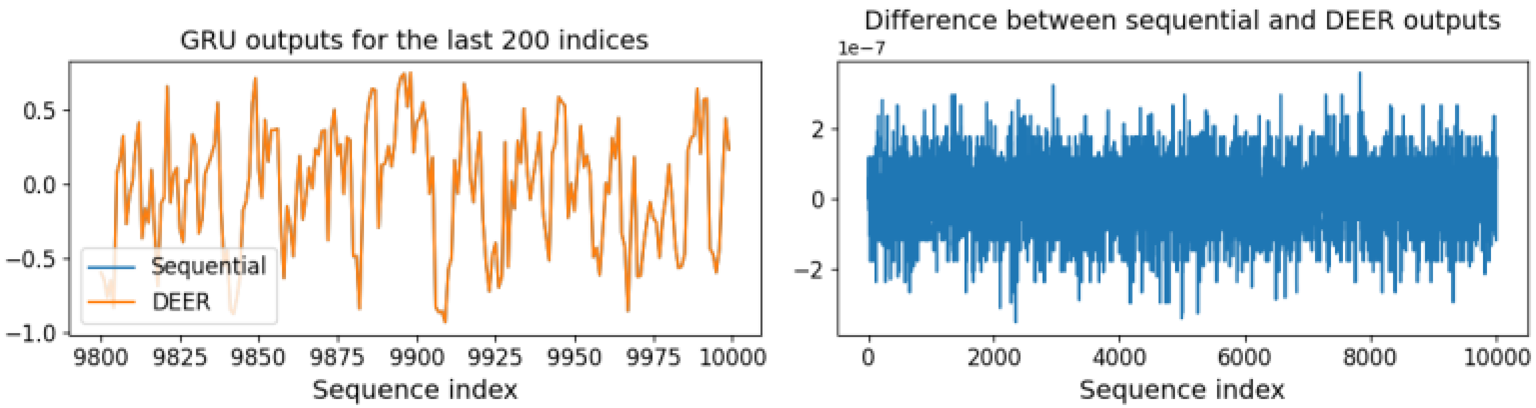
Results: speed up





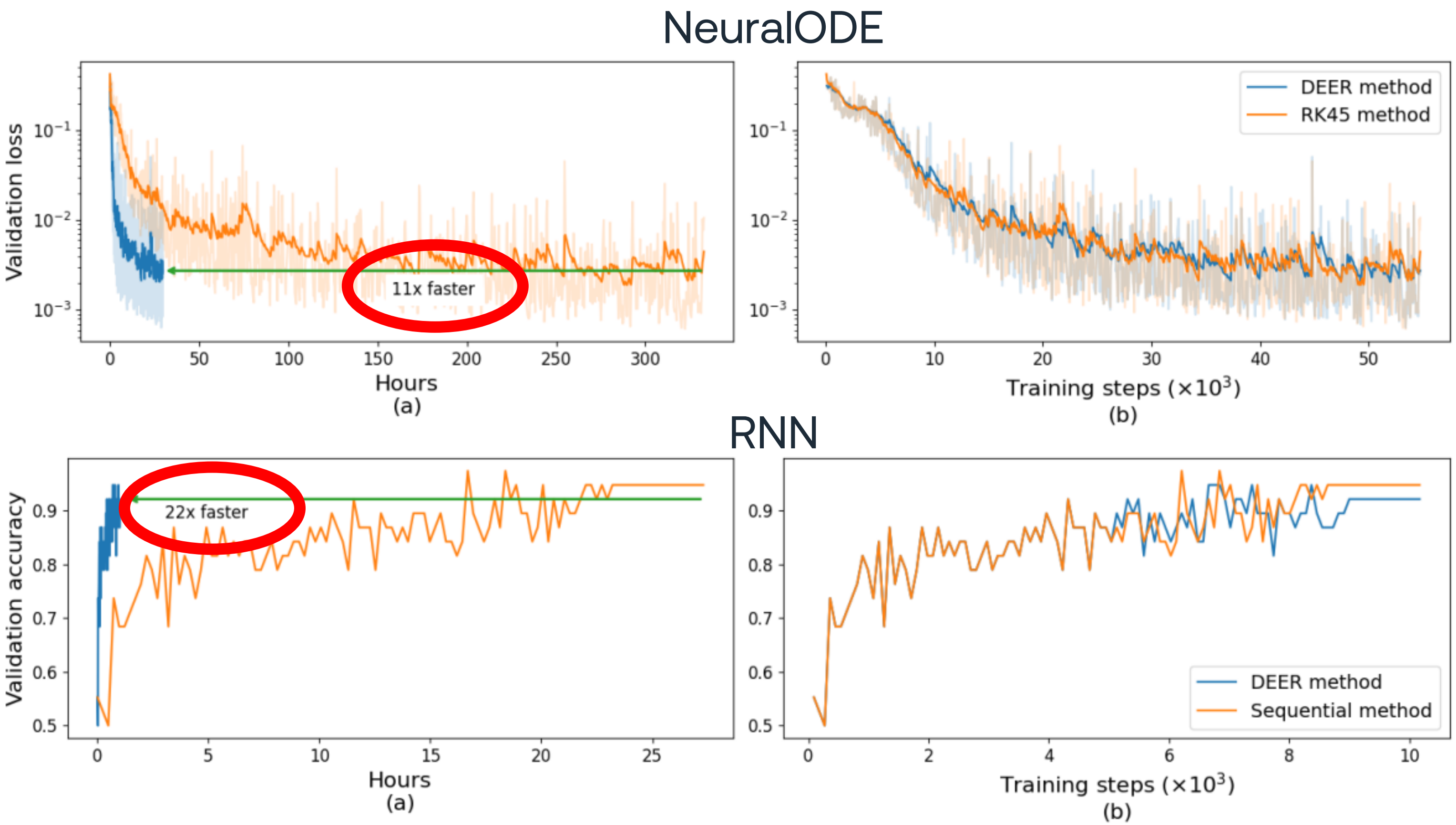
Up to 1000x faster

Results: output comparison



(a) (b)

Results: usage in NeuralODE & RNN training



Training up to 11x and 22x faster with DEER

Conclusion:
RNN is parallelizable!

Curious? See https://arxiv.org/abs/2309.12252