**Problem**

Vanilla RNNs tend to face the problem of vanishing or exploding gradients: iterative applications of the weight matrix and the activation function cause the gradient to expand or shrink exponentially in the number of time steps:

\[
\frac{\partial h_t}{\partial h_k} = \prod_{\tau=t}^{T} W^T \text{diag}(\sigma'(h_{\tau-1}))
\]

where 
\(I_k\) is the identity matrix.

**Approach**

The idea of constraining eigenvalues suggests thinking about \(W\) as a rotation in space. \([2]\) terms this a unitary RNN (uRNN) as the weight \(W\) is unitary (or in the case with real entries, orthogonal).

\(W\) is kept orthogonal via a modified update rule (on the right) based on \([3]\) and \([4]\).

Overall architecture:

**Toy Experiments**

**Task 1 – First term recall**

5, 7, 9, 53, 99, ...

\[\text{(Ans: 5)}\]

**Task 2 – k-th highest term**

5, 7, 9, 53, 99, 2, 100, ...

\[\text{(Ans: 18)}\]

The uRNN outperforms the vanilla RNN on both toy experiments, especially on longer sequence lengths. With more timesteps, the gradients on the RNN should start to vanish, making it harder to train.

**Language Modelling**

The task of language modelling involves predicting the next word given a sequence of previous words.

- Conducted on Penn Treebank Dataset.
- Successfully bounded gradient norms.
- Example sentences suggest "long-term" memory.
- 2ms / 1 – 10% slower per batch despite "costly" matrix exponential operation.

**Discussion**

**Advantages**

- Great at contextual tasks which require "long-term" memory
- Concrete bound on gradients
- Potentially better at training networks before the recurrent layer

**Disadvantages**

- Negligibly slower (especially for larger batch sizes)

**References**


**Acknowledgments**

We would like to thank Michael Hahn for his guidance, advice and various suggestions throughout the project.