When Recurrent Models Don't Need to be Recurrent

by John Miller and Moritz Hardt

Oleg Voinov
September 7

Intro

What Is Recurrent Neural Network

General form:
$$h_t = \phi_w(h_{t-1}, x_t)$$

Classical RNN:
$$h_t = \rho\,(Wh_{t-1} + Ux_t)$$

Linear:
$$h_t = Wh_{t-1} + Ux_t$$

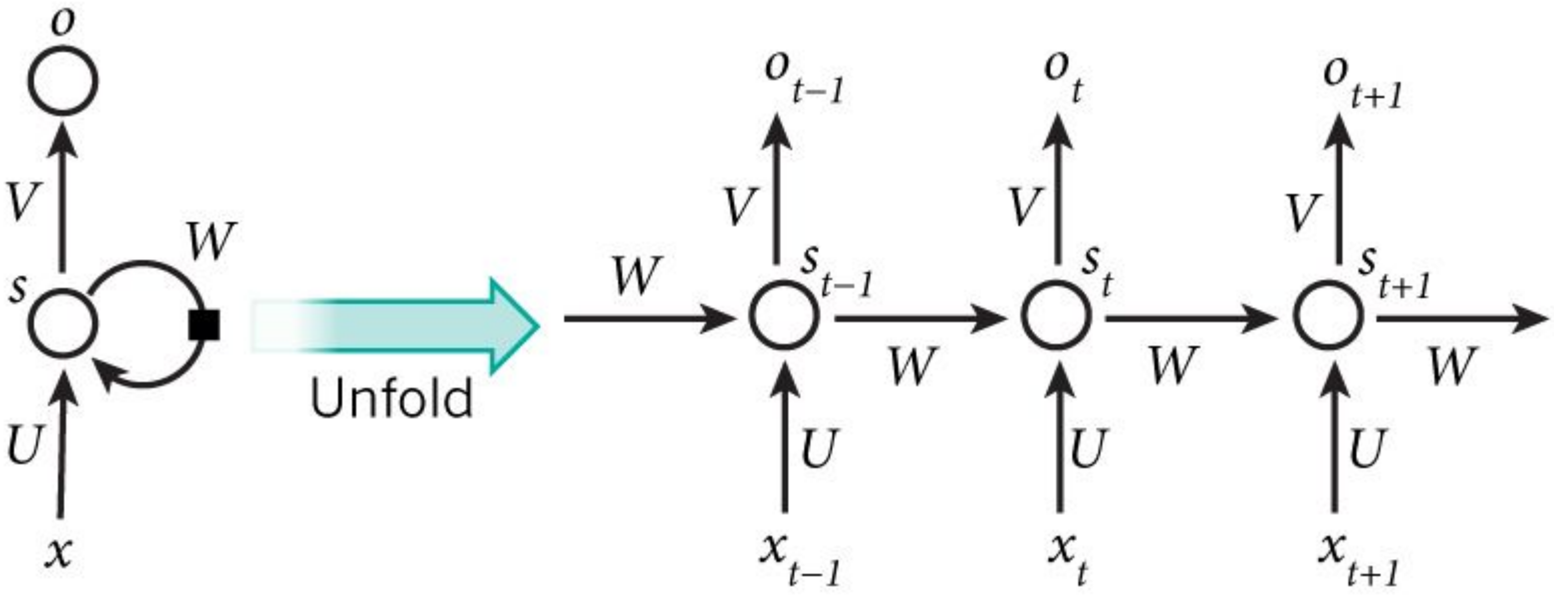
What Is Recurrent Neural Network

LSTM:

$$f_t = \sigma(W_f h_{t-1} + U_f x_t)$$
$$i_t = \sigma(W_i h_{t-1} + U_i x_t)$$
$$o_t = \sigma(W_o h_{t-1} + U_o x_t)$$
$$z_t = \tanh(W_z h_{t-1} + U_z x_t)$$
$$c_t = i_t \circ z_t + f_t \circ c_{t-1}$$
$$h_t = o_t \cdot \tanh(c_t),$$

Feed-Forward vs RNN

RNN is not feed-forward



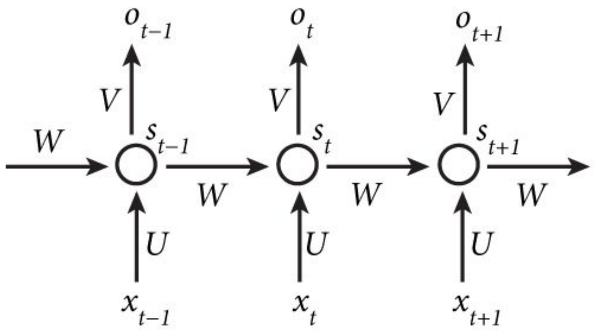
Feed-Forward vs RNN

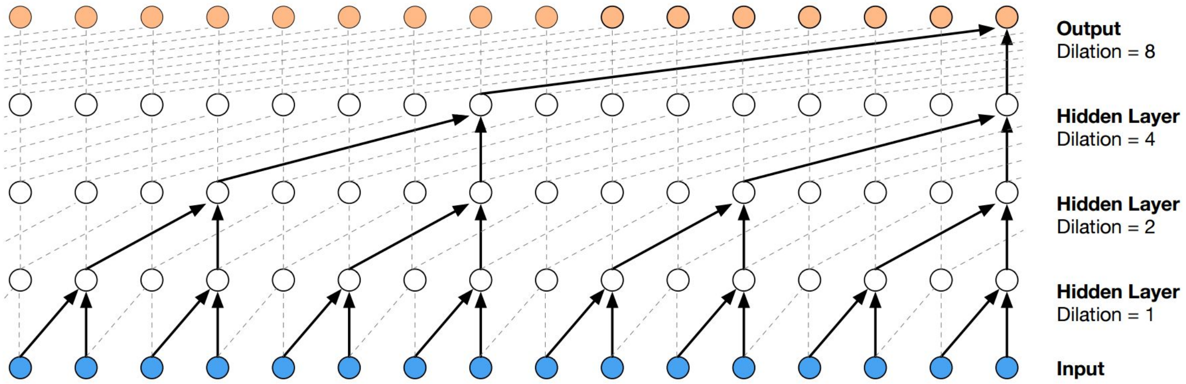
RNN: $$h_t = \phi_w(h_{t-1}, x_t)$$

Truncated RNN: $$h_t^k = \phi_w(h_{t-1}^k, x_t), \quad h_{t-k}^k = 0$$

Why Feed-Forward Instead of RNN

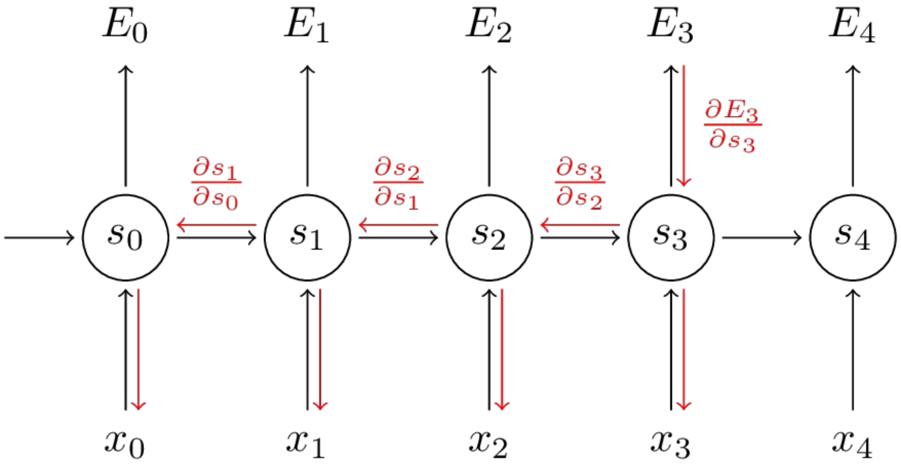
- Parallelization





Why Feed-Forward Instead of RNN

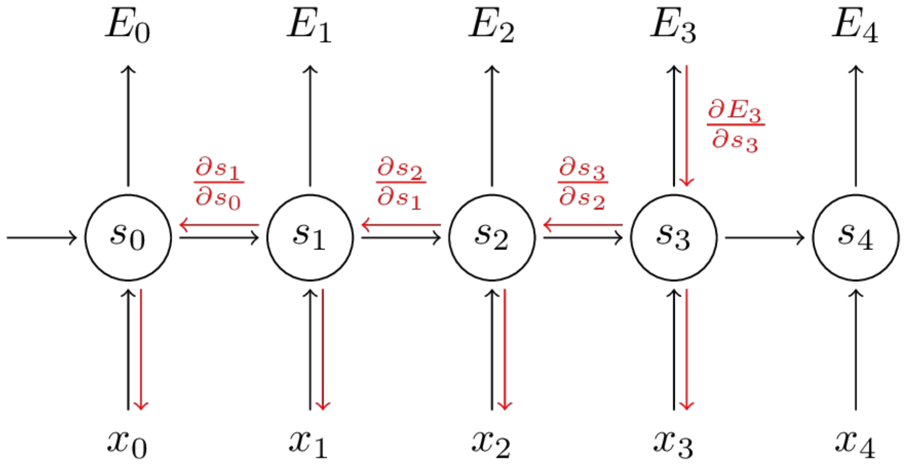
- Parallelization
- Trainability



$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{\infty} \frac{\partial E_3}{\partial s_3} \frac{\partial s_3}{\partial s_{3-k}} \frac{\partial s_{3-k}}{\partial W}$$

Why Feed-Forward Instead of RNN

- Parallelization
- Trainability

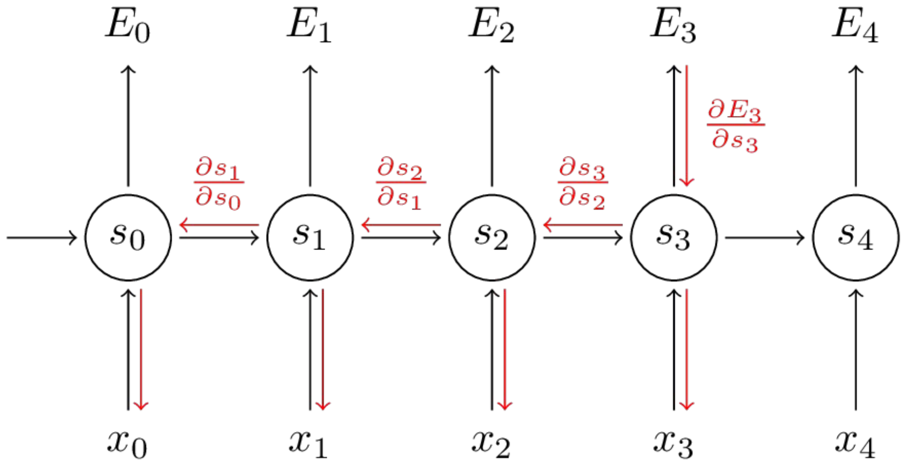


truncated backpropagation to the rescue

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{K} \frac{\partial E_3}{\partial s_3} \frac{\partial s_3}{\partial s_{3-k}} \frac{\partial s_{3-k}}{\partial W}$$

Why Feed-Forward Instead of RNN

- Parallelization
- Trainability
- Memory footprint

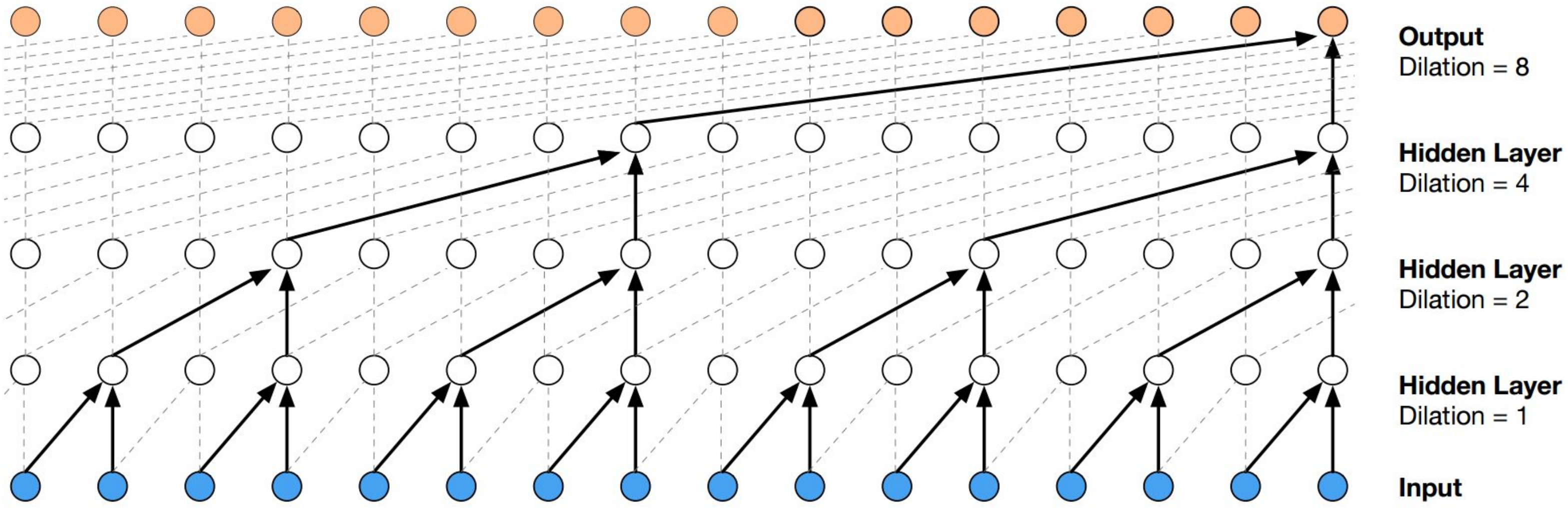


truncated backpropagation to the rescue

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{K} \frac{\partial E_3}{\partial s_3} \frac{\partial s_3}{\partial s_{3-k}} \frac{\partial s_{3-k}}{\partial W}$$

Feed-Forward Overperforming RNNs

- WaveNet on speech synthesis

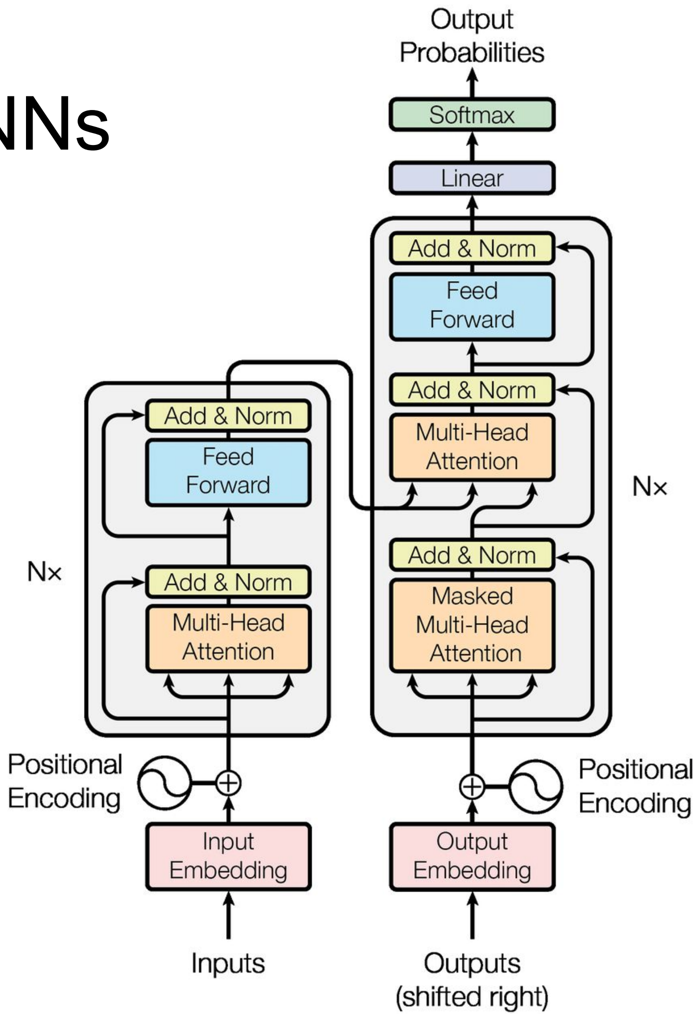


it is autoregressive

Feed-Forward Overperforming RNNs

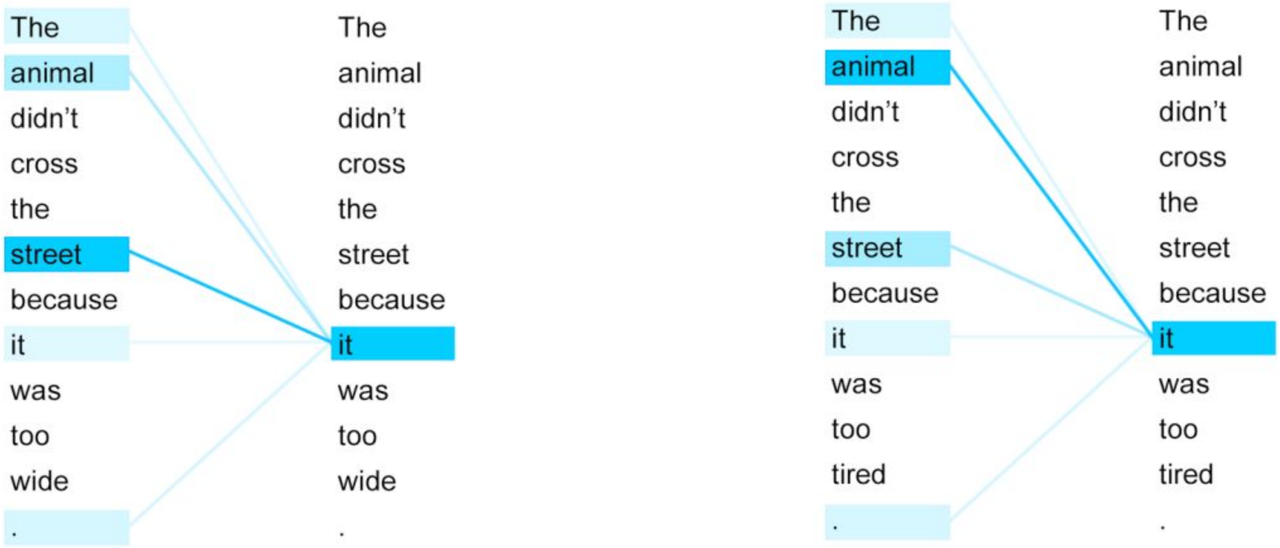
- WaveNet on speech synthesis
- Transformer on machine translation

it is not autoregressive



Feed-Forward Overperforming RNNs

- WaveNet on speech synthesis
- Transformer on machine translation



Feed-Forward Overperforming RNNs

- WaveNet on speech synthesis
- Transformer on machine translation
- Temporal convolutional network
  by Bai et al. on multiple tasks

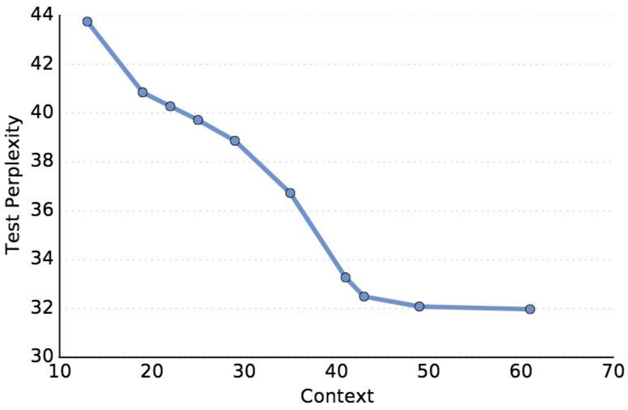
Bai et al. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling // arXiv, 2018

Why Feed-Forward Outperform RNN

i.e. why full history doesn't help

- Full history is unnecessary, [Dauphin et al]

perplexity $P = 2^H = 2^{-\sum_s p(s) \log_2 p(s)}$



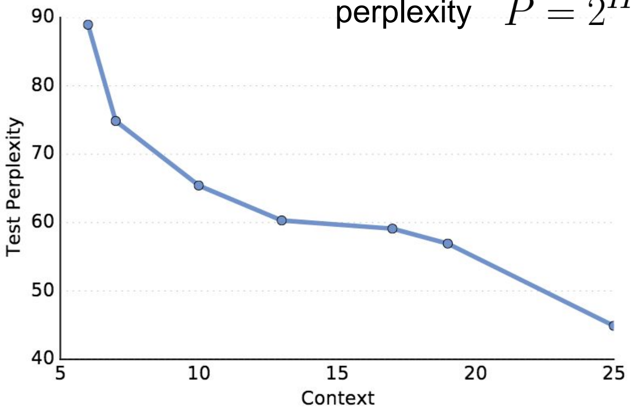


Figure 4. Test perplexity as a function of context for Google Billion Word (left) and Wiki-103 (right). We observe that models with bigger context achieve better results but the results start diminishing quickly after a context of 20.

Dauphin et al. Language Modeling with Gated Convolutional Networks // ICML'17

Why Feed-Forward Outperform RNN

i.e. why full history doesn’t help

- Full history is unnecessary, [Dauphin et al]
- Full history is not used, [Miller and Hardt] (the paper)

Miller and Hardt. When Recurrent Models Don't Need To Be Recurrent // arXiv, 2018

The Actual Paper

RNN and Feed-Forward Truncated RNN

RNN $$h_t = \phi_w(h_{t-1}, x_t)$$

Truncated RNN $$h_t^k = \phi_w(h_{t-1}^k, x_t), \quad h_{t-k}^k = 0$$

Stability

$$h_t = \phi_w(h_{t-1}, x_t)$$

State-transition map is stable = it is contractive:

$$\left\|\phi_w(h, x) - \phi_w(h', x)\right\| \leq \lambda \left\|h - h'\right\|$$

$$\lambda < 1$$

Stability

General form: $$h_t = \phi_w(h_{t-1}, x_t)$$

$$\left\|\phi_w(h, x) - \phi_w(h', x)\right\| \leq \lambda \left\|h - h'\right\|, \quad \lambda < 1$$

Classical RNN: $$h_t = \rho\left(W h_{t-1} + U x_t\right)$$

$\|W\| < 1/L_\rho$, L_ρ is Lipschitz constant of ρ

Linear: $$h_t = W h_{t-1} + U x_t$$

$$\|W\| \leq \lambda < 1$$

* LSTM

Claims

- Stable RNNs are well approximated by truncated RNNs
  - outputs are close
  - parameters are close

- Real-world RNNs are effectively stable

Theory

Outputs Are Close

$$h_t = \phi_w(h_{t-1}, x_t)$$
$$h_t^k = \phi_w(h_{t-1}^k, x_t), \quad h_{t-k}^k = 0$$

Lemma 1. *Assume ϕ_w is λ-contractive and L_x-Lipschitz in x. Assume the input sequence $\|x_t\| \leq B_x$ for all t. If $k \geq O\left(\log\left(\frac{L_x B_x}{(1-\lambda)\varepsilon}\right)\right)$, then the difference in hidden states $\left\|h_t - h_t^k\right\| \leq \varepsilon$.*

i.e. stable RNNs don't have long-term memory:

- stable = vanishing gradients
- long-term memory requires exploding gradients, [Pascanu et al]

Pascanu, Mikolov, Bengio. On the difficulty of training Recurrent Neural Networks // ICML'13

Gradients Are Close

$$h_t = \phi_w(h_{t-1}, x_t)$$
$$h_t^k = \phi_w(h_{t-1}^k, x_t), \quad h_{t-k}^k = 0$$

Lemma 2. *Assume p (and therefore p^k) is Lipschitz and smooth. Assume ϕ_w is smooth, λ-contractive, and Lipschitz in x and w. Assume the inputs satisfy $\|x_t\| \leq B_x$, then*

$$\left\|\nabla_w p_T - \nabla_w p_T^k\right\| = \gamma k \lambda^k,$$

where $\gamma = O\left(B_x(1-\lambda)^{-2}\right)$, suppressing dependence on the Lipschitz and smoothness parameters.

i.e. gradients of RNN and truncated RNN **with the same parameters** are close

Gradients Are Close

$$h_t = \phi_w(h_{t-1}, x_t)$$
$$h_t^k = \phi_w(h_{t-1}^k, x_t), \quad h_{t-k}^k = 0$$

Lemma 2. *Assume p (and therefore p^k) is Lipschitz and smooth. Assume ϕ_w is smooth, λ-contractive, and Lipschitz in x and w. Assume the inputs satisfy $\|x_t\| \leq B_x$, then*

$$\left\| \nabla_w p_T - \nabla_w p_T^k \right\| = \gamma k \lambda^k,$$

where $\gamma = O\left(B_x (1-\lambda)^{-2}\right)$, suppressing dependence on the Lipschitz and smoothness parameters.

Lemma 3. *For any $w, w' \in \Omega$, suppose ϕ_w is smooth, λ-contractive, and Lipschitz in w. If p is Lipschitz and smooth, then*

$$\left\| \nabla_w p_T(w) - \nabla_w p_T(w') \right\| \leq \beta \left\| w - w' \right\|,$$

where $\beta = O\left((1-\lambda)^{-3}\right)$, suppressing dependence on the Lipschitz and smoothness parameters.

i.e. gradients of RNNs **with slightly different parameters** are close

Weights Are Close

$$h_t = \phi_w(h_{t-1}, x_t)$$
$$h_t^k = \phi_w(h_{t-1}^k, x_t), \quad h_{t-k}^k = 0$$

Proposition 2. *Under the assumptions of Lemmas 2 and 3, for compact, convex Ω, after N steps of projected gradient descent with step size $\alpha_t = \alpha/t$, $\left\|w_{\text{recurr}}^N - w_{\text{trunc}}^N\right\| \leq \alpha\gamma k \lambda^k N^{\alpha\beta+1}$.*

Weights Are Close

$$h_t = \phi_w(h_{t-1}, x_t)$$
$$h_t^k = \phi_w(h_{t-1}^k, x_t), \quad h_{t-k}^k = 0$$

Proposition 2. *Under the assumptions of Lemmas 2 and 3, for compact, convex Ω, after N steps of projected gradient descent with step size $\alpha_t = \alpha/t$,* $\|w^N_{\text{recurr}} - w^N_{\text{trunc}}\| \le \alpha\gamma k \lambda^k N^{\alpha\beta+1}$.

a must

too fast lr-decay, but theory suggests that OK [Bertsekas]

Bertsekas. Nonlinear Programming. Athena Scientific, 1999

Main Result

$$h_t = \phi_w(h_{t-1}, x_t)$$
$$h_t^k = \phi_w(h_{t-1}^k, x_t), \quad h_{t-k}^k = 0$$

Theorem 1. *Let p be Lipschitz and smooth. Assume ϕ_w is smooth, λ-contractive, Lipschitz in x and w. Assume the inputs are bounded, and the prediction function f is L_f-Lipschitz. If $k = O(\log(\gamma N^\beta/\varepsilon))$, then after N steps of projected gradient descent with step size $\alpha_t = 1/t$, $\|y_T - y_T^k\| \le \varepsilon$.*

i.e. stable RNN is well approximated by feed-forward truncated RNN

Experiments

Gradients and Weights Are Close

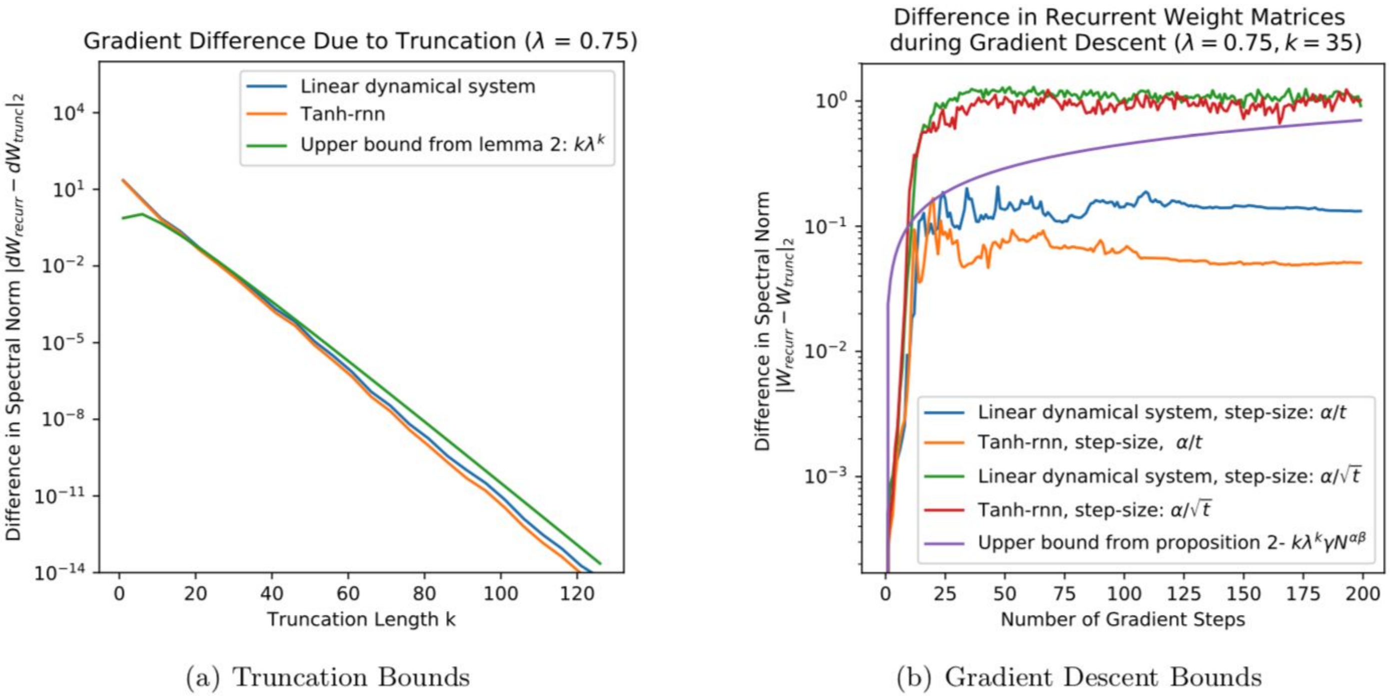


Figure 1: Empirical validation of Lemma 2 and Proposition 2 on random Gaussian instances. Without the $1/t$ rate, the gradient descent bound no longer appears qualitatively correct, suggesting the $O(1/t)$ rate is necessary.

Stability Is OK

Stable RNN vs arbitrary RNN:

- same performance on Wikitext-2 benchmark
- arbitrary RNN are effectively stable

Stability Is OK

Stable RNN vs arbitrary RNN:

- same performance on Wikitext-2 benchmark
- arbitrary RNN are effectively stable

sketchy

Stability is OK

Arbitrary RNN (LSTM) vs truncated arbitrary RNN (LSTM)

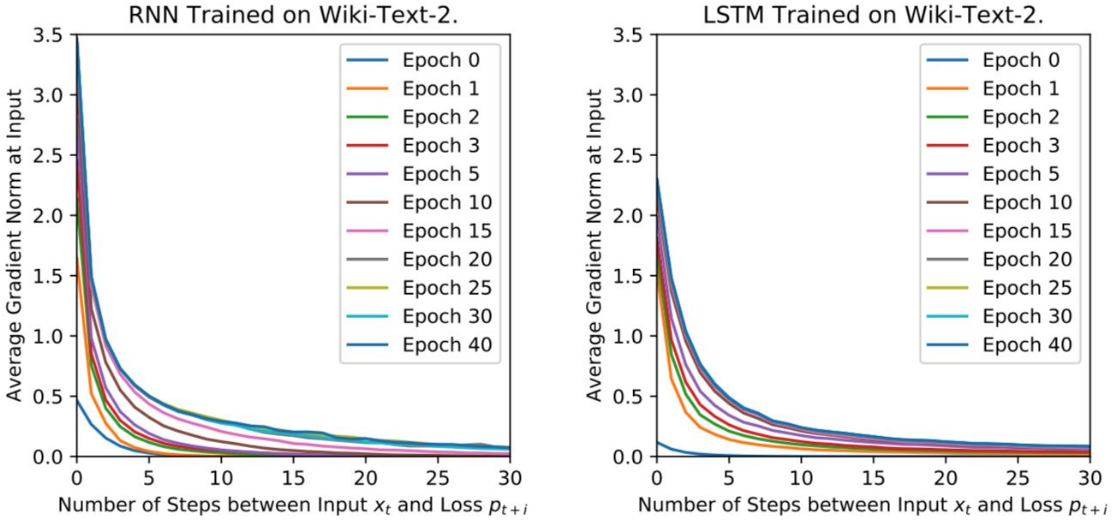
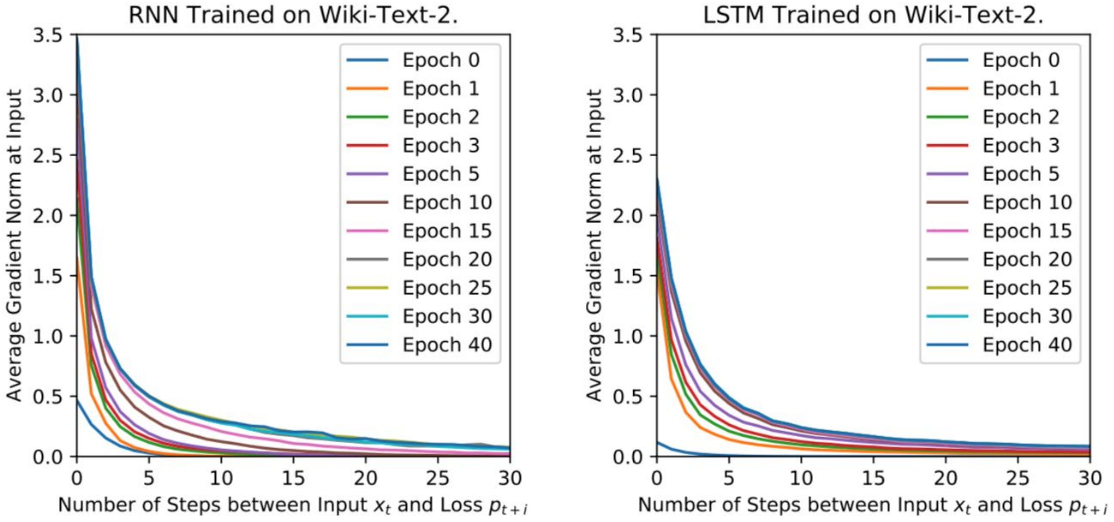


Figure 2: Norm of the gradient with respect to inputs, $\|\nabla_{x_t} p_{t+i}\|$, as the distance between the input and the loss grows, averaged over the entire held-out set. The gradient vanishes for moderate values of i in both cases. The RNN has test perplexity 146.7 and the LSTM has test perplexity of 92.3.

Stability is OK

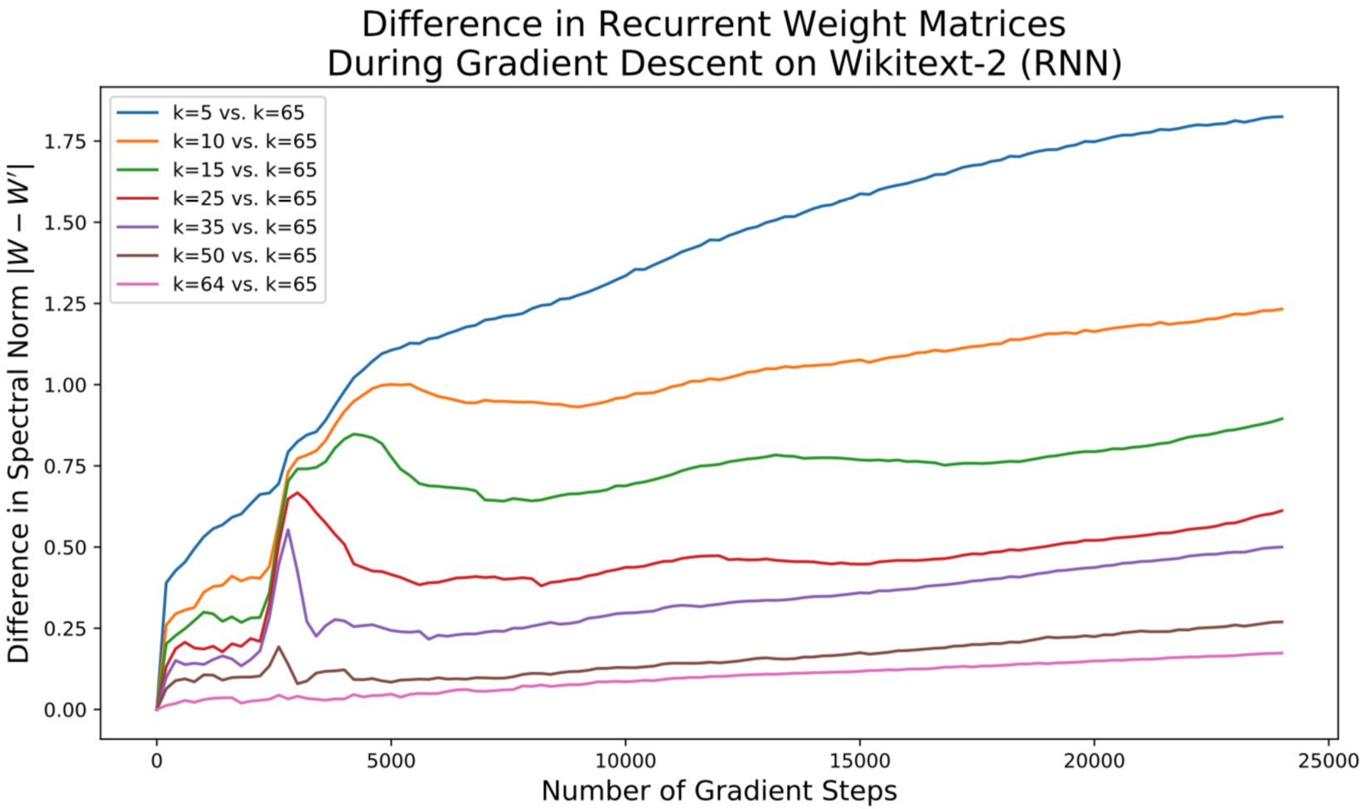
Arbitrary RNN (LSTM) vs truncated arbitrary RNN (LSTM)



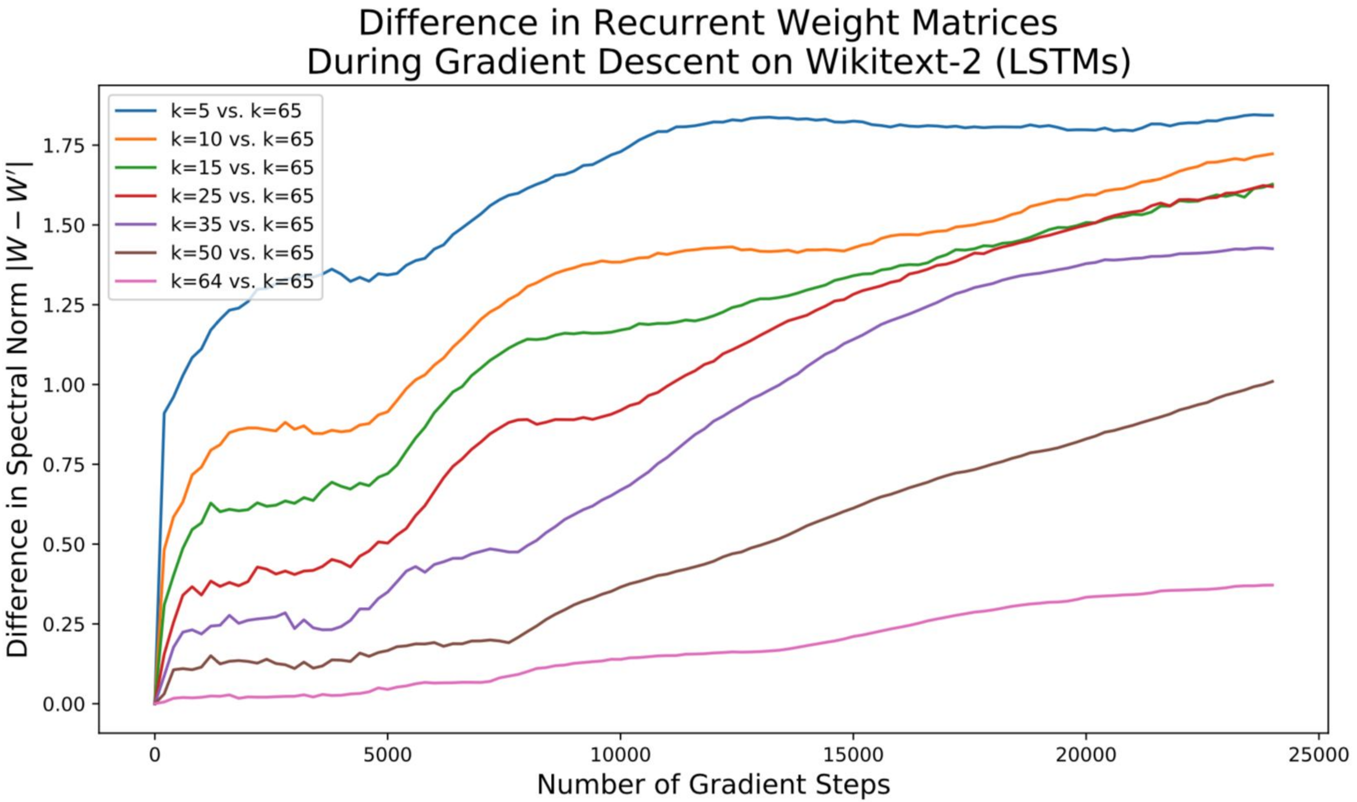
SOTA is 40-100

Figure 2: Norm of the gradient with respect to inputs, $\|\nabla_{x_t} p_{t+i}\|$, as the distance between the input and the loss grows, averaged over the entire held-out set. The gradient vanishes for moderate values of i in both cases. The RNN has test perplexity 146.7 and the LSTM has test perplexity of 92.3.

Weights Are Close for Arbitrary



Weights Are Close for Arbitrary



Summary

- Stable RNNs are well approximated by truncated RNNs
  - outputs are close
  - parameters are close

- Real-world RNNs are effectively stable

Summary

- Stable RNNs are well approximated by truncated RNNs
    - outputs are close
    - parameters are close

strange learning-rate scheduling $\alpha_t = \alpha/t$

- Real-world RNNs are effectively stable

needs more backup