Adaptively Truncating Backpropagation Through Time to Control Gradient Bias

Christopher Aicher[1] **Nicholas J. Foti**[2] **Emily B. Fox**[1,2]

[1] Department of Statistics, University of Washington
[2] Paul G. Allen School of Computer Science and Engineering, University of Washington
{aicherc, nfoti, ebfox}@uw.edu

Abstract

Truncated backpropagation through time (TBPTT) is a popular method for learning in recurrent neural networks (RNNs) that saves computation and memory at the cost of bias by truncating backpropagation after a fixed number of lags. In practice, choosing the optimal truncation length is difficult: TBPTT will not converge if the truncation length is too small, or will converge slowly if it is too large. We propose an adaptive TBPTT scheme that converts the problem from choosing a temporal lag to one of choosing a tolerable amount of gradient bias. For many realistic RNNs, the TBPTT gradients decay geometrically *in expectation* for large lags; under this condition, we can control the bias by varying the truncation length adaptively. For RNNs with smooth activation functions, we prove that this bias controls the convergence rate of SGD with biased gradients for our non-convex loss. Using this theory, we develop a practical method for adaptively estimating the truncation length during training. We evaluate our adaptive TBPTT method on synthetic data and language modeling tasks and find that our adaptive TBPTT ameliorates the computational pitfalls of fixed TBPTT.

1 INTRODUCTION

Recurrent neural networks (RNNs) are a popular method of processing sequential data for wide range of tasks such as language modeling, machine translation and reinforcement learning.

As with most deep neural networks, RNNs are typically trained with gradient descent. These gradients can be calculated efficiently using *backpropagation through time* (BPTT) which applies backpropagation to the unrolled network (Werbos et al., 1990). For long sequential data, BPTT is both computationally and memory intensive, hence approximations based on *truncating* BPTT (TBPTT) have been proposed (Williams and Zipser, 1995; Sutskever, 2013). However, this truncation causes the gradients to be biased. When the truncation level is not sufficiently large, the bias introduced can cause SGD to not converge. In practice, a large truncation size is chosen heuristically (e.g. larger than the expected 'memory' of the system) or via cross-validation.

Quantifying the bias due to truncation is difficult. Depending on the parameters of the RNN, the gradient bounds for backpropagation either *explode* or *vanish* (Bengio et al., 1994; Pascanu et al., 2013). When the gradients vanish, the bias in TBPTT can be bounded. Recent work has analyzed conditions for the parameters of the RNN to enforce this vanishing gradient condition (Miller and Hardt, 2019). However, these approaches are very restrictive and prevent the RNN from learning long-term dependencies.

To bound the bias in TBPTT, instead of restricting the parameters, we formalize the heuristic assumption that the gradients in backpropagation should rapidly decay for steps beyond the 'memory' of the RNN. Specifically, we assume gradient bounds that decay exponentially *in expectation* rather than uniformly. Under this assumption, we show that the bias in TBPTT decays geometrically and also how to estimate an upper bound for this bias given a minibatch of backpropagated gradients. Using this estimated upper bound, we propose an adaptive truncation scheme to control the bias. In addition, we prove non-asymptotic convergence rates for SGD when the *relative* bias of our gradients is bounded. In particular, we show that when the relative bias, $\delta < 1$, SGD with biased gradients converges at the rate $(1-\delta)^{-1}$ compared to SGD with exact (unbiased) gradients. In our experiments on synthetic and text data we see that

(i) our heuristic assumption holds empirically for these tasks, (ii) our adaptive TBPTT method controls the bias, while fixed TBPTT does not, and (iii) that our adaptive TBPTT method is competitive with or outperforms the optimal fixed TBPTT.

The paper is organized as follows. First, we review generic RNNs and BPTT in Section 2. Then, we develop our theoretical results in Section 3. Using this theory, we develop estimators for the bias and propose an adaptive TBPTT SGD scheme in Section 4. Finally, we test our proposed adaptive TBPTT training scheme in Section 5 on both synthetic and language modeling data.

2 BACKGROUND

A generic RNN with inputs $x_t \in \mathbb{R}^{d_x}$ and hidden states $h_t \in \mathbb{R}^{d_h}$ at time step t evolves as

$$h_t = H(h_{t-1}, x_t; \theta) \tag{1}$$

for some function $H : \mathbb{R}^{d_h \times d_x} \to \mathbb{R}^{d_h}$ with parameters $\theta \in \Theta$ (e.g, weights and biases) of the model. This framework encompasses most popular RNNs. We now present some examples that we use later.

Simple RNN: A simple RNN is a linear map composed with a non-linear activation function ρ

$$h_t = H(h_{t-1}, x_t, \theta) = \rho(W h_{t-1} + U x_t) \ ,$$

where the weight matrices W, U are the parameters.

Long Short-Term Memory (LSTM): A popular class of sequence models are LSTMs (Hochreiter and Schmidhuber, 1997). The hidden state consists of a pair of vectors $h_t = (c_t, \tilde{h}_t)$

$$\begin{aligned}
f_t &= \sigma(W_f \tilde{h}_{t-1} + U_f x_t) \\
i_t &= \sigma(W_i \tilde{h}_{t-1} + U_i x_t) \\
o_t &= \sigma(W_o \tilde{h}_{t-1} + U_o x_t) \\
z_t &= \tanh(W_z \tilde{h}_{t-1} + U_z x_t) \\
c_t &= i_t \circ z_t + f_t \circ c_{t-1} \\
\tilde{h}_t &= o_t \cdot \tanh(c_t) \ ,
\end{aligned}$$

where the eight matrices W_*, U_*, for $* \in \{f, i, o, z\}$ are the parameters, σ is the logistic function, and $\circ$ denotes elementwise multiplication. LSTMs are examples of *gated-RNNs* which capture more complex time dependence through the use of gate variables f_t, i_t, o_t.

Stacked RNNs: RNNs can be composed by *stacking* the hidden layers. This allows different layers to learn structure at varying resolutions. Each RNN-layer treats the output of the previous layer as its input. Specifically, each layer $l = 1, \ldots, N_l$ is described as

$$h_t^{(l)} = H_{(l)}(h_{t-1}^{(l)}, h_t^{(l-1)}, \theta_{(l)}) \ ,$$

where $h_t^{(0)} = x_t$.

2.1 TRAINING RNNs

To train an RNN, we minimize a loss that can be decomposed into individual time steps $\mathcal{L}(\theta) = \frac{1}{T}\sum_{t=1}^{T} \mathcal{L}_t(\theta)$. For example, the individual loss at step t may measure the accuracy of h_t at predicting target outputs $y_t \in \mathbb{R}^{d_y}$.

Given $X = x_{1:T}$ and $\mathcal{L}_{1:T}$, gradient descent methods are typically used to train the RNN to minimize the loss. To scale gradient descent for large T, we use stochastic gradient descent (SGD), which uses a random estimator $\hat{g}$ for the full gradient $g = \nabla_\theta \mathcal{L}$.

We first consider estimating g using the gradient of the loss at a random individual time step, $\mathcal{L}_s$, where s is a *random* index drawn uniformly from $\{1, \ldots, T\}$. In Section 4.1, we discuss efficient ways of computing Eq. (3) for multiple losses $\mathcal{L}_{s:s+t}$ simultaneously.

Unrolling the RNN, the gradient of $\mathcal{L}_s$ is

$$\nabla_\theta \mathcal{L}_s = \frac{\partial \mathcal{L}_s}{\partial \theta} + \sum_{k=0}^{s} \frac{\partial \mathcal{L}_s}{\partial h_{s-k}} \cdot \frac{\partial h_{s-k}}{\partial \theta}, \tag{2}$$

which can be efficiently computed using *backpropagation through time* (BPTT) (Werbos et al., 1990; Williams and Zipser, 1995).

When s is large, unrolling the RNN is both computationally and memory prohibitive; therefore in practice, the backpropagated gradients in Eq. (2) are truncated after K steps (Williams and Zipser, 1995; Sutskever, 2013)

$$\widehat{\nabla}_\theta \mathcal{L}_s^K = \frac{\partial \mathcal{L}_s}{\partial \theta} + \sum_{k=0}^{K} \frac{\partial \mathcal{L}_s}{\partial h_{s-k}} \frac{\partial h_{s-k}}{\partial \theta}, \tag{3}$$

where $K \ll T$.

Let $\hat{g}_K = \widehat{\nabla}_\theta \mathcal{L}_s^K$ be our stochastic gradient estimator for g truncated at K steps. When $K = T$, $\hat{g}_T$ is an unbiased estimate for g; however in general for $K < T$, the gradient estimator $\hat{g}_K$ is *biased*, $\mathbb{E}[\hat{g}_K(\theta)] \neq g(\theta)$. In practice, the truncation length K is chosen heuristically to be "large enough" to capture the memory in the underlying process, in hope that the bias of $\hat{g}_K(\theta)$ does not affect the convergence of SGD. In general, there are no guarantees on the size of this bias or the convergence of the overall optimization for fixed K.

2.2 VANISHING AND EXPLODING GRADIENT

Let $\| \cdot \|$ denote the spectral norm for matrices and Euclidean norm for vectors.

To analyze the bias of $\hat{g}_K$, we are interested in the behavior of $\frac{\partial \mathcal{L}_t}{\partial h_{t-k}}$ for large k. Pascanu et al. (2013) observed

$$\frac{\partial \mathcal{L}_t}{\partial h_{t-k}} = \frac{\partial \mathcal{L}_t}{\partial h_t} \prod_{r=1}^{k} \frac{\partial h_{t-r+1}}{\partial h_{t-r}} \quad . \tag{4}$$

In particular, the repeated product of Jacobian matrices $\frac{\partial h_t}{\partial h_{t-1}}$ cause Eq. (4) to tend to *explode* to infinity or *vanish* to zero. When θ has an exploding gradient, then the bias of $\hat{g}_K$ is unbounded. When θ has a vanishing gradient, then the bias of $\hat{g}_K$ is small; however if the gradient decays too rapidly, the RNN cannot learn long-term dependences (Bengio et al., 1994; Pascanu et al., 2013; Miller and Hardt, 2019). In practice, LSTMs and other gated-RNNs have been seen to work in a middle ground where (for appropriate θ and inputs $x_{1:T}$) the gate variables prevent the gradient from exploding or vanishing (Hochreiter and Schmidhuber, 1997; Belletti et al., 2018). However, *gradient bounds*, based on the Jacobian $\|\frac{\partial h_t}{\partial h_{t-1}}\| \leq \lambda$, either explode or vanish

$$\left\| \frac{\partial \mathcal{L}_t}{\partial h_{t-k}} \right\| \leq \left\| \frac{\partial \mathcal{L}_t}{\partial h_t} \right\| \cdot \lambda^k \quad . \tag{5}$$

In light of Eq. (5), several approaches have been proposed in the literature to restrict θ to control λ.

Unitary training methods have been proposed to restrict θ such that $\lambda \approx 1$ for all θ, but do not bound the bias of the resulting gradient (Arjovsky et al., 2016; Jing et al., 2017; Vorontsov et al., 2017).

Stable or *Chaos-Free* training methods have been proposed to restrict θ such that $\lambda < 1$ (Laurent and von Brecht, 2017; Miller and Hardt, 2019). In particular, Miller and Hardt (2019) call an RNN H *stable* for parameters θ if it is a contraction in h, that is

$$\sup_{\substack{h,h' \in \mathbb{R}^{d_h} \\ x \in \mathbb{R}^{d_x}}} \frac{\|H(h,x,\theta) - H(h',x,\theta)\|}{\|h-h'\|} \leq \lambda < 1 \tag{6}$$

and call an RNN H *data-dependent stable* if the supremum over Eq. (6) is restricted to *observed* inputs $x \in X$. Let $\Theta_{\lambda\text{-Stable}}$ be the set of parameters θ satisfying Eq. (6) and $\Theta^X_{\lambda\text{-Stable}}$ be the set of parameters θ satisfying the data-dependent version.

Miller and Hardt (2019) show that if $\theta \in \Theta_{\lambda\text{-Stable}}$ the RNN gradients have an exponential forgetting property (as $\|\frac{\partial H}{\partial h}\| < \lambda$), which prevents the RNN from learning long-term dependences. We desire conditions on θ where we can bound the bias, but are less restrictive than Eq. (6).

3 THEORY

In this section, we consider bounding the bias in TBPTT when θ satisfies a *relaxation* of the contraction restriction Eq. (6). Under this condition and a bound on $\|\partial h_t / \partial \theta\|$, we show that both the *absolute bias* and *relative bias* are bounded and decay geometrically for large K. Finally, we prove the convergence rate of SGD for gradients with bounded relative bias. Full proofs of theorems can be found in the Supplement.

3.1 GEOMETRIC DECAY FOR LARGE LAGS

To reduce notation, we define $\phi_k = \|\frac{\partial L_s}{\partial h_{s-k}}\|$ to be the gradient norm of loss $\mathcal{L}_s$ at time s with respect to the hidden state k lags in the past. Note that ϕ_k is a random variable as s is a random index.

Our relaxation of Eq. (6) is to assume the norm of the backpropagated gradient ϕ_k decays geometrically, *on average* for large enough lags k. More formally,

Assumption (A-1). *For θ fixed, there exists $\beta \in (0,1)$ and $\tau \geq 0$ such that*

$$\mathbb{E}\left[\phi_{k+1}\right] \leq \beta \cdot \mathbb{E}\left[\phi_k\right] \ \text{, for all } k \geq \tau \tag{7}$$

This generalizes the vanishing gradient condition to hold *in expectation*.

To contrast (A-1) with $\theta \in \Theta_{\lambda\text{-Stable}}$, we observe that if $\theta \in \Theta_{\lambda\text{-Stable}}$ then the gradient norms ϕ_k must *uniformly* decay exponentially

$$\phi_{k+1} \leq \lambda \cdot \phi_k \text{ for all } k \quad . \tag{8}$$

Eq. (7) is less restrictive than Eq. (8) as $\phi_{k+1} \leq \beta \cdot \phi_k$ only occurs for $k > \tau$ and in expectation rather than uniformly. Denote the set of θ that satisfy (A-1) with β, τ for inputs $X = x_{1:T}$ as $\Theta^X_{\beta,\tau}$. Then, we have $\Theta_{\lambda\text{-Stable}} \subseteq \Theta^X_{\lambda\text{-Stable}} \subset \Theta^X_{\beta,\tau}$ for $(\beta,\tau) = (\lambda, 0)$. Therefore (A-1) is a more general condition. For illustration, we present two examples where $\theta \in \Theta^X_{\beta,\tau}$ but $\theta \notin \Theta^X_{\lambda\text{-Stable}}$.

Nilpotent Linear RNN: Consider a simple RNN with linear activation $h_t = W h_{t-1} + U x_t$ where W is a *nilpotent* matrix with index k, that is $W^k = 0$. Then $\partial h_t / \partial h_{t-k} = W^k = 0$ hence $(W, U) \in \Theta^X_{0,k}$; however the norm $\|\partial h_t / \partial h_{t-k}\| = \|W\|$ can be arbitrarily large, thus $(W, U) \notin \Theta_{\lambda\text{-Stable}}$. For this example, although H is not stable, k-repeated composition $h_t = (H \circ \cdots \circ H)(h_{t-k}, x_{t-k+1:t})$ is stable.

Unstable RNN with Resetting: Consider a generic RNN with θ chosen such that H is unstable, Lipschitz with constant $\lambda > 1$, but with a *resetting* property $H(h, x) = 0$, whenever $x \in \mathcal{X}_0$. Then,

$$\mathbb{E}_S[\phi_{k+1}] \leq \Pr(x_s, \ldots, x_{s-k+1} \notin \mathcal{X}_0) \cdot \lambda \cdot \mathbb{E}_S[\phi_k]$$

Although the RNN is unstable, if the probability $\{x_s, \ldots, x_{s-k+1}\}$ *hits* the resetting set $\mathcal{X}_0$ is greater than $1 - \beta\lambda^{-1}$ for sufficiently large k and for some $\beta < 1$, then $\Pr(x_s, \ldots, x_{s-k+1} \notin \mathcal{X}_0) \cdot \lambda \leq \beta$ and therefore $\theta \in \Theta_{\beta,k}$ with $\theta \notin \Theta^X_{\lambda\text{-Stable}}$. For this example, although H is not stable, properties of input distribution x_t can lead to H have vanishing gradients in *expectation*.

3.2 TBPTT BIAS BOUNDS

We now show the usefulness of assumption (A-1), that is if $\theta \in \Theta^X_{\beta,\tau}$, then the bias of TBPTT is bounded and decays geometrically in K. To do so, we additionally assume the partial derivatives of the hidden state with respect to the parameters is bounded.

Assumption (A-2). *For θ fixed, there exists $M < \infty$ such that $\|\frac{\partial H(x_t,h_t,\theta)}{\partial \theta}\| \leq M$ for all t.*

For most typical RNNs, where θ are weights and biases, if the inputs x_t and h_t are bounded then M can be bounded and assumption (A-2) holds.

We now show both the bias of TBPTT is guaranteed to decay geometrically for large K.

Theorem 1 (Bias Bound). *If (A-1) and (A-2) hold for θ, then the absolute bias is upper bounded as*

$$\|\mathbb{E}\left[\hat{g}_K(\theta)\right] - g(\theta)\| \leq \mathcal{E}(K, \theta) \ ,$$

where

$$\mathcal{E}(K,\theta) = \begin{cases} M \cdot \mathbb{E}\left[\sum_{k=K+1}^{\tau-1} \phi_k + \frac{\phi_\tau}{1-\beta}\right], & K < \tau \\ M \cdot \mathbb{E}\left[\phi_\tau\right] \cdot \frac{\beta^{K-\tau}}{1-\beta}, & K \geq \tau \end{cases} .$$

And the relative bias is upper bounded by

$$\frac{\|\mathbb{E}\left[\hat{g}_K(\theta)\right] - g(\theta)\|}{\|g(\theta)\|} \leq \Delta(K,\theta) \ ,$$

where

$$\Delta(K,\theta) = \frac{\mathcal{E}(K,\theta)}{\max_{k \leq K} \|\mathbb{E}\, \hat{g}_k(\theta)\| - \mathcal{E}(k,\theta)} \ .$$

when the denominator is positive.

Note that $\mathcal{E}(K,\theta)$ decays geometrically for $K \geq \tau$ and therefore $\Delta(K,\theta)$ decays geometrically for large enough K (as the denominator is monotone increasing).

Using this upper bound, we define $\kappa(\delta, \theta)$ to be the *smallest truncation length* for the parameters θ with guaranteed relative bias less than δ. That is

$$\kappa(\delta,\theta) = \min_K\{\Delta(K,\theta) < \delta\} \ . \qquad (9)$$

The geometric decay in $\Delta(K,\theta)$ ensures $\kappa(\delta,\theta)$ is small.

Finally, we define the *adaptive* TBPTT gradient estimator to be $\hat{g}(\theta) = \hat{g}_{\kappa(\delta,\theta)}(\theta)$, which truncates BPTT after $\kappa(\delta,\theta)$ steps and therefore has relative bias less than δ.

3.3 SGD WITH BIASED GRADIENTS

We use stochastic gradient descent (SGD) to learn θ

$$\theta_{n+1} = \theta_n - \gamma_n \cdot \hat{g}(\theta_n) \ , \qquad (10)$$

where $\{\gamma_n\}_{n=1}^N$ are stepsizes. When using SGD for non-convex optimization, such as in training RNNs, we are interested in convergence to stationary points of $\mathcal{L}$, where θ is called a ϵ-stationary point of $\mathcal{L}(\theta)$ if $\|g(\theta)\|^2 \leq \epsilon$ (Nesterov, 2013).

Usually the stochastic gradients $\hat{g}$ are assumed to be unbiased; however during training RNNs with TBPTT, the truncated gradients are biased. Based on Section 3.2, we consider the case when $\hat{g}(\theta)$ has a bounded relative bias,

$$\|\mathbb{E}\left[\hat{g}(\theta)\right] - g(\theta)\| \leq \delta\|g(\theta)\|, \ \forall\theta \ . \qquad (11)$$

such as for our adaptive estimator $\hat{g}(\theta) = \hat{g}_{\kappa(\delta,\theta)}(\theta)$.

For gradients with bounded relative bias $\delta < 1$ (and the additional assumptions below), Poljak and Tsypkin (1973) and Bertsekas and Tsitsiklis (1989) prove that the averaged SGD sequence θ_n *asymptotically* converges to a stationary point when the stepsizes are $\gamma_n \propto n^{-1}$. However, *non-asymptotic* convergence rates are also of interest, as they are useful in practice to understand the non-asymptotic performance of the algorithm. Ghadimi and Lan (2013) prove non-asymptotic convergence rates for SGD with unbiased gradients. We extend these results to the case of SGD with biased gradients. Similar results were previously investigated by Chen and Luss (2018), but with weaker bounds[1].For our SGD convergence bound we need two additional assumptions.

Assumption (A-3). *The gradients are L-Lipschitz*

$$\|g(\theta) - g(\theta')\| \leq L\|\theta - \theta'\|, \quad \forall\theta,\theta' \ .$$

This assumption holds for generic RNNs as long as the activation functions are smooth (e.g. σ or $\tanh$, but not RELU). Second, we assume that the variance of our stochastic gradient estimator is uniformly bounded.

Assumption (A-4). $\mathbb{E}\,\|\hat{g}(\theta) - \mathbb{E}\,\hat{g}(\theta)\|^2 \leq \sigma^2$ *for all θ.*

We can now present our main theorem regarding convergence rates of SGD with biased gradients.

[1]They consider the case of consistent but biased estimators, where the gradients are uniformly bounded and the relative error is controlled with high probability (rather than in expectation). See the Supplement for additional discussion.

Theorem 2 (SGD with Biased Gradients)**.** *If the relative bias of $\hat{g}(\theta)$ is bounded by $\delta < 1$ for all θ_n and (A-3) and (A-4) both hold, then SGD, Eq.* (10)*, with stepsizes $\gamma_n \leq \frac{1-\delta}{L(1+\delta)^2}$ satisfies*

$$\min_{n=1,\ldots,N+1} \|g(\theta_n)\|^2 \leq \frac{2D_{\mathcal{L}} + L\sigma^2 \sum_{n=1}^N \gamma_n^2}{(1-\delta)\sum_{n=1}^N \gamma_n} \ , \quad (12)$$

where $D_{\mathcal{L}} = (\mathcal{L}(\theta_1) - \min_{\theta^*} \mathcal{L}(\theta^*))$.

In particular, the optimal fixed stepsize for N fixed is $\gamma_n = \sqrt{2D_{\mathcal{L}}/(NL\sigma^2)}$, for which the bound is

$$\min_{n=1,\ldots,N+1} \|g(\theta_n)\|^2 \leq \frac{1}{1-\delta} \cdot \sqrt{\frac{8D_{\mathcal{L}}L\sigma^2}{N}} \ . \quad (13)$$

When $\delta = 0$, Thm. 2 reduces to the smooth non-convex convergence rate bound (Ghadimi and Lan, 2013). The price of biased gradients in SGD is the factor $(1-\delta)^{-1}$.

In practice, when the constants in Thm. 2 are unknown (e.g. $D_{\mathcal{L}}$), we can use a decaying stepsize $\gamma_n = \gamma \cdot n^{-1/2}$. Once $\gamma_n \leq \frac{1-\delta}{L(1+\delta)^2}$, Thm. 2 implies

$$\min_{n=1,\ldots,N+1} \|g(\theta_n)\|^2 = \mathcal{O}\left(\frac{1}{1-\delta} \cdot \frac{\log n}{\sqrt{n}}\right) \ . \quad (14)$$

Thm. 2 provides bounds of the form $\min_n \|g(\theta_n)\|^2 < \epsilon$, but does not say which iterate is best. This can be accounted for by using a random-stopping time (Ghadimi and Lan, 2013) or using variance reduction methods such as SVRG (Reddi et al., 2016). We leave these extensions as future work. In our experiments, we select θ_n by evaluating performance on a validation set.

What happens when (A-1) is violated? We do not restrict θ to $\Theta_{\beta,\tau}^X$ during training, therefore whenever $\theta_n \notin \Theta_{\beta,\tau}$ (or in practice when $\kappa(\delta, \theta_n)$ is larger than our computational budget allows), we are not able to construct $\hat{g}(\theta_n)$ such that the relative bias is bounded. In these cases, we use $\hat{g} = \hat{g}_{K_{\max}}$ for some large truncation $K_{\max}$. Although $\hat{g}_{K_{\max}}$ does not satisfy (A-1), we assume the stationary points of interest θ^* are in $\Theta_{\beta,\tau}^X$ and that eventually θ_n ends in a neighborhood of a θ^* that is a subset of $\Theta_{\beta,\tau}$ where our theory holds.

The advantage of an adaptive $\kappa(\delta, \theta)$ over a fixed K is that it ensures convergence when possible (relative bias $\delta < 1$), while being able to get away with using a smaller K during optimization for computational speed-ups.

4 ADAPTIVE TBPTT

The theory in Sec. 3 naturally suggests an adaptive TBPTT algorithm that we summarize in Alg. 1. Our method selects a truncation level by periodically estimating $\kappa(\delta, \theta)$ over the course of SGD. In this section, we describe our implementation and how to estimate the quantities necessary to choose the truncation level[2].

Algorithm 1 Adaptive TBPTT

```
1: Input: initial parameters $\theta_0$, initial truncation $K_0$,
   stepsizes $\gamma_{1:N}$, relative bias tolerance $\delta$, batch size
   $S$, window size $R$
2: for $n = 0, \ldots, N-1$ do
     ▷ Compute adaptive truncation
3:   Sample random minibatch $\mathcal{S}$ of size $S$
4:   Calculate $\phi_k$ using $\mathrm{BPTT}(R, 1)$ for Eq. (18)
5:   Calculate $\hat{P}_{\mathcal{S}}[\phi_k]$ for $k \in [0, R]$ using Eq. (20)
6:   Estimate $\hat{\beta}$ using Eq. (21) or (22)
7:   Set $K_n = \hat{\kappa}(\delta, \theta)$ using Eq. (25)
     ▷ Update $\theta$ with streaming gradients
8:   for $m = 1, \ldots, T/K_n$ do
9:     Get minibatch $\mathcal{S}_m$ defined in Eq. (17)
10:    Calculate $\hat{g}(\theta_n) = \mathrm{BPTT}(2K_n, K_n)$ on $\mathcal{S}_m$
11:    Set $\theta_n = \theta_n - \gamma_n \cdot \sqrt{K_n} \cdot \hat{g}(\theta_n)$
12:  end for
13:  Set $\theta_{n+1} = \theta_n$
14: end for
15: Return $\theta_{1:N}$.
```

4.1 COMPUTING TBPTT GRADIENTS

Following Williams and Zipser (1995), we denote $\mathrm{BPTT}(K_1, K_2)$ to be truncated backpropagation for K_2 losses $\mathcal{L}_{s-K_2+1:s}$ backpropagated over K_1 steps, that is

$$\mathrm{BPTT}(K_1, K_2) = \frac{1}{K_2} \sum_{k'=0}^{K_2-1} \sum_{k=s-t}^{K_1} \frac{\partial \mathcal{L}_{s-k'}}{\partial h_{s-k}} \cdot \frac{\partial h_{s-k}}{\partial \theta} \quad (15)$$

which can be computed efficiently using the recursion

$$b_k = \begin{cases} b_{k-1} \cdot \dfrac{\partial h_{s-k+1}}{\partial h_{s-k}} + \dfrac{\partial \mathcal{L}_{s-k}}{\partial h_{s-k}} & \text{if } k < K_2 \\ b_{k-1} \cdot \dfrac{\partial h_{s-k+1}}{\partial h_{s-k}} & \text{if } k \geq K_2 \end{cases} \quad (16)$$

with $\mathrm{BPTT}(K_1, K_2) = \frac{1}{K_2} \sum_{k=0}^{K_1} b_k \cdot \frac{\partial h_{s-k}}{\partial \theta}$. It is important to include the normalization factor $\frac{1}{K_2}$, to ensure regularizations (such as dropout or weight decay) do not change for different values of K_2.

When $K_1 = K$ and $K_2 = 1$, then we obtain $\mathrm{BPTT}(K, 1) = \hat{g}_K$. However, individually calculating S samples of $\hat{g}_K$ using $\mathrm{BPTT}(K, 1)$ takes $\mathcal{O}(JK)$ computation; whereas the same gradients (plus extra lags)

[2]Code for our implementation and experiments is at `https://github.com/aicherc/adaptive_tbptt`.

can be computed using $\mathrm{BPTT}(J+K, K)$ in $\mathcal{O}(J+K)$ time. In practice a popular default setting is to set $K_1 = K_2 = K$ (as done in TensorFlow (Abadi et al., 2016)); however this overweights small lags, as the k-th loss is only backpropagated only $K-k$ steps. To ensure all K losses are backpropagated at least K steps, in our experiments we use $\mathrm{BPTT}(2K, K)$.

We also scale the gradient updates $\gamma_n \hat{g}$ by $\sqrt{K_n}$ to account for the decreasing variance in $\mathrm{BPTT}(2K, K)$ as K increases. If we did not scale the gradient updates, then as K increases, the resulting increase in the computational cost per step is not offset.

To handle the initialization of h_{s-k} in Eq. (15), we partition $\{1, \ldots, T\}$ into T/K contiguous subsequences

$$\mathcal{S}_m = [(m-1)*K+1, \ldots, m*K] \ . \tag{17}$$

By sequentially processing $\mathcal{S}_m$ in order, the final hidden state of the RNN on $\mathcal{S}_m$ can be used as the input for the RNN on $\mathcal{S}_{m+1}$.

4.2 ESTIMATING GEOMETRIC DECAY RATE

A prerequisite for determining our adaptive truncation level is estimating the geometric rate of decay in Eq. (7), β, and the lag at which it is valid, τ. We consider the case where we are given a batch of gradient norms ϕ_k for s in a random minibatch $\mathcal{S}$ of size $|\mathcal{S}| = S$ that are backpropagated over a window $k \in [0, R]$

$$\left\{ \phi_k = \left\| \frac{\partial \mathcal{L}_s}{\partial h_{s-k}} \right\| \ : \ s \in \mathcal{S}, k \in [0, R] \right\} \ . \tag{18}$$

The gradient norms ϕ_k can be computed iteratively in parallel using the same architecture as truncated backpropagation $\mathrm{BPTT}(R, 1)$. The window size R should be set to some large value. This should be larger than the τ of the optimal θ^*. The window size R can be large, since we only estimate β periodically.

We first focus on estimating β given an estimate $\hat{\tau} > \tau$. We observe that if $\hat{\tau} \geq \tau$ and (A-1) holds, then

$$\log \beta \geq \max_{k' > k \geq \hat{\tau}} \frac{\log \mathbb{E}[\phi_k] - \log \mathbb{E}[\phi_{k'}]}{k - k'} \ . \tag{19}$$

Eq. (19) states that $\log \beta$ bounds the slope of $\log \mathbb{E}[\phi_t]$ between any pair of points larger than τ.

Using Eq. (19), we propose two methods for estimating β. We replace the expectation $\mathbb{E}$ with the empirical approximation based on the minibatch $\mathcal{S}$

$$\mathbb{E}[f(s)] \approx \hat{P}_{\mathcal{S}}[f(s)] := \frac{1}{\|\mathcal{S}\|} \sum_{s \in \mathcal{S}} f(s). \tag{20}$$

Substituting the empirical approximation into Eq. (19) and restricting the points to $[\hat{\tau}, R]$, we obtain

$$\hat{\beta} = \log \left[\max_{\hat{\tau} \leq k < k' \leq R} \frac{\log \hat{P}_{\mathcal{S}}[\phi_k] - \log \hat{P}_{\mathcal{S}}[\phi_{k'}]}{k - k'} \right] \ . \tag{21}$$

Because this estimate of β is based on the maximum it is sensitive to noise: a single noisy pair of $\hat{P}_{\mathcal{S}}[\phi_k]$ completely determines $\hat{\beta}$ when using Eq. (21). To reduce this sensitivity, we could use a $(1-\alpha)$ quantile instead of strict max; however, to account for the noise in $\hat{P}_{\mathcal{S}}[\phi_k]$, we use linear regression, which is a weighted-average of the pairs of slopes

$$\tilde{\beta} = \log \left[\frac{\sum_{k,k'} (\log \hat{P}_{\mathcal{S}}[\phi_k] - \log \hat{P}_{\mathcal{S}}[\phi_{k'}](k - k'))}{\sum_{k,k'} (k-k')^2} \right] \ . \tag{22}$$

This estimator is not guaranteed to be consistent for an upper bound on β (i.e. as $|\mathcal{S}| \to T$, $\tilde{\beta} \not\geq \beta$); however, we found Eq. (22) performed better in practice.

The correctness and efficiency of both methods depends on the size of both the minibatch $\mathcal{S}$ and the window $[\hat{\tau}, R]$. Larger minibatches improve the approximation accuracy of $\hat{P}_{\mathcal{S}}$. Large windows $[\hat{\tau}, R]$ are necessary to check (A-1), but also lead to additional noise.

In practice, we set $\hat{\tau}$ to be a fraction of R; in our experiments we did not see much variability in $\hat{\beta}$ once $\hat{\tau}$ was sufficiently large, therefore we use $\hat{\tau} = \frac{9}{10} R$.

4.3 ESTIMATING THE TRUNCATION LEVEL

To estimate $\kappa(\delta, \theta)$ in Eq. (9), we obtain empirical estimates for the absolute and relative biases of the gradient.

Given $\hat{\beta}, \hat{\tau}$, our estimated bound for the absolute bias is

$$\hat{\mathcal{E}}(K, \theta) = \begin{cases} \hat{M} \cdot \hat{P}_{\mathcal{S}} \left[\sum_{k=K+1}^{\hat{\tau}-1} \phi_k + \frac{\phi_{\mathcal{S},\hat{\tau}}}{1-\hat{\beta}} \right], & K < \hat{\tau} \\ \hat{M} \cdot \hat{P}_{\mathcal{S}}[\phi_{\hat{\tau}}] \cdot \frac{\hat{\beta}^{K-\hat{\tau}}}{1-\hat{\beta}}, & K \geq \hat{\tau} \end{cases} \tag{23}$$

where we estimate an upper-bound $\hat{M}$ for M by keeping track of an upper-bound for $h_{s,t}$ and $x_{s,t}$ during training.

Similarly, our estimated bound for the relative bias is

$$\hat{\Delta}(K, \theta) = \frac{\hat{\mathcal{E}}(K, \theta)}{\| \max_{k \leq K} \hat{P}_{\mathcal{S}}[\hat{g}_k(\theta)] \| - \hat{\mathcal{E}}(k, \theta)} \tag{24}$$

In our implementation, we make the simplifying assumption that $\|\hat{P}_{\mathcal{S}}[\hat{g}_K / \hat{M}]\| \approx \hat{P}_{\mathcal{S}} \| \sum_{k=0}^{K} \phi_k \|$, which allows us to avoid calculating $\hat{M}$.

Our estimate for K with relative error δ is

$$\hat{\kappa}(\delta, \theta) = \min_{K \in [K_{\min}, K_{\max}]} \{ \hat{\Delta}(K, \theta) < \delta \} \ , \tag{25}$$

where $K_{\min}$ and $K_{\max}$ are user-specified bounds.

4.4 RUNTIME ANALYSIS OF ALGORITHM 1

Our adaptive TBPTT scheme, Algorithm 1, consists of estimating the truncation length (lines 3-7) and updating the parameters with SGD using TBPTT (lines 8-12); whereas a fixed TBPTT scheme skips lines 3-7.

Updating the parameters with $\mathrm{BPTT}(2K, K)$ streaming over $\{1, \ldots, T\}$ (lines 8-12), takes $\mathcal{O}(T/K \cdot K) = \mathcal{O}(T)$ time and memory[3]. The additional computational cost of adaptive TBPTT (lines 3-7) is dominated by the calculation of gradient norms ϕ_k, Eq. (18), using $\mathrm{BPTT}(R, 1)$, which takes $\mathcal{O}(R)$ time and memory. If we update the truncation length α times each epoch, then the total cost for adaptive TBPTT is $\mathcal{O}(T + \alpha R)$. Therefore, the additional computation cost is negligible when $\alpha R << T$.

In Algorithm 1, we only update K once per epoch ($\alpha = 1$); however more frequent updates ($\alpha < 1$) allow for less stale estimates at additional computational cost.

5 EXPERIMENTS

In this section we demonstrate the advantages of our adaptive TBPTT scheme (Algorithm 1) in both synthetic copy and language modeling tasks. We compare using fixed TBPTT and our adaptive TBPTT to train RNNs with SGD. We evaluate performance using perplexity (PPL) on the test set, for the θ_n that achieve the best PPL on the validation set. To make a fair comparison, we measure PPL against the number of data passes (epochs) used in training (counting the data used to estimate $\hat{\kappa}(\delta, \theta_n)$). In Section 5.3, we demonstrate that the best θ_n appear to satisfy (A-1) (e.g., $\theta \in \Theta_{\beta,\lambda}$) by presenting the gradient norms $\mathbb{E}\left[\phi_k\right]$ against lag k. In the Supplement we present additional experiments, applying our adaptive TBPTT scheme to temporal point process modeling (Du et al., 2016).

5.1 SYNTHETIC COPY EXPERIMENT

The 'copy' synthetic task is used to test the RNN's ability to remember information seen earlier (Hochreiter and Schmidhuber, 1997; Arjovsky et al., 2016; Jing et al., 2017; Vorontsov et al., 2017). We consider a special variant of this task from (Mujika et al., 2018), which we review now. Let $A = \{a_i\}$ be a set of symbols, where the first I represent data and remaining two represent "blank" and "start recall". Each input consists of sequence of m random data symbols followed by the "start recall" symbol and $m-1$ more blanks. The desired output consists of m blanks followed by the original sequence of m data symbols. For example when $m = 6$ and $A = \{A, B, C, -, \#\}$

```
Input:   ACBBAB#-----
Output:  ------ACBBAB
```

We concatenate multiple of such inputs to construct $x_{1:T}$ and multiple outputs to construct $y_{1:T}$. We expect that TBPTT with $K > m$ will perform well, while $K < m$ will perform poorly.

In our experiments, we consider both a *fixed* $m = 10$ and a *variable* m drawn uniformly over $[5, 10]$. For the variable copy length experiment, we expect TBPTT to degrade more gradually as K decreases. We set $I = 6$ and use training data of length $T = 256,000$ and validation and test data of length $T = 64,000$.

Model and Training Setup We train separate 2-layer LSTMs with a embedding input layer and a linear output-layer to both the fixed- and variable- copy tasks by minimizing the cross-entropy loss. The embedding dimension is set to 6 and hidden and cell dimensions of the LSTM layers are set to 50. We train θ using SGD using a batchsize of $S = 64$ and a fixed learning rate of $\gamma = 1.0$ with fixed TBPTT $K \in [5, 10, 15, 20, 30]$ and our adaptive TBPTT method $\delta \in [0.9, 0.5, 0.1]$, $W = 100$, $K_0 = 15$ and $[K_{\min}, K_{\max}] = [2, 100]$.

Results Figure 1 shows the results for the synthetic copy task. The left figures present the test set PPL against the number of data epochs used in training. We see that adaptive methods (black solid lines) perform as well as or better than the best fixed methods (colored dashed). In particular, TBPTT with $K = 5$ (blue) does not learn how to accurately predict the outputs as K is too small $5 = K \leq m = 10$. On the other hand, $K = 30$ (purple) takes much longer to converge. The center figures show how $\hat{\kappa}(\delta, \theta_n)$ evolves for the adaptive TBPTT methods over training. The adaptive methods initially use small K as the backpropagated gradient vanish rapidly in the early epochs; however as the adaptive TBPTT methods learn θ the necessary K for a relative error of δ increases until they eventually level off at $\kappa(\delta, \theta_N)$. The right figures show the estimated relative bias δ of the gradient estimates during training. We see that the adaptive methods are able to roughly control δ to be less than their target values, while the fixed methods initially start with low δ and before increasing and leveling off. Additional figures for the validation PPL and tables of numerical values can be found in the Supplement.

[3]As K increases, $\mathrm{BPTT}(2K, K)$ takes more time per step $\mathcal{O}(K)$, but there are less steps per epoch $\mathcal{O}(T/K)$; hence the overall computation time is $\mathcal{O}(T)$

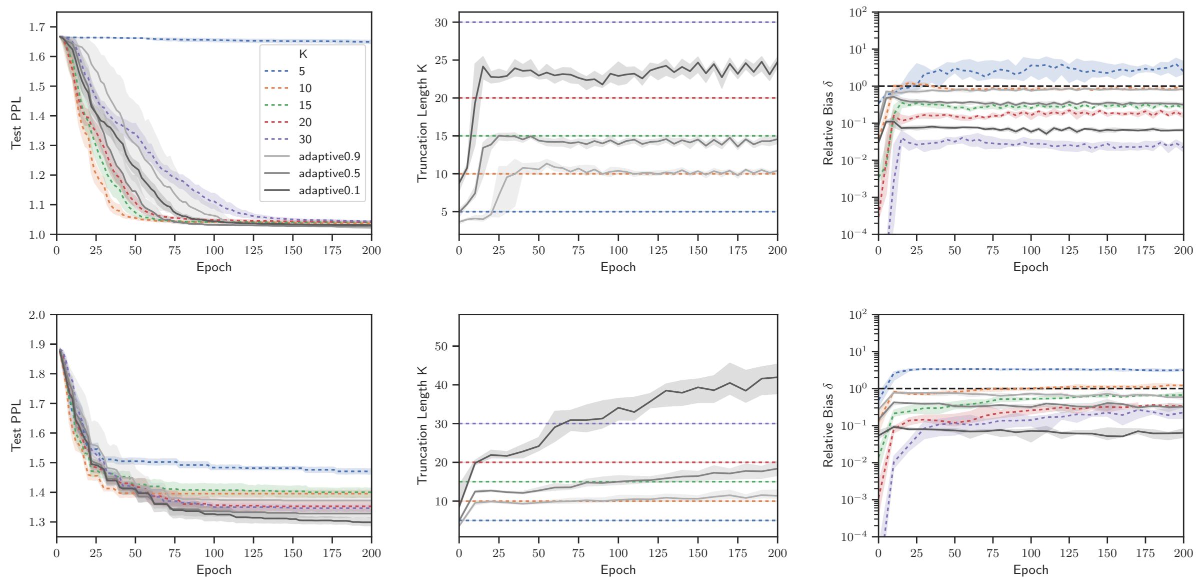

Figure 1: Synthetic Copy Results: (left) Test PPL vs epoch, (center) $\hat{\kappa}(\delta, \theta_n)$ vs epoch (right) $\hat{\delta}(K)$ vs epoch. (Top) fixed $m = 10$, (bottom) variable $m \in [5, 10]$. Error bars are 95 percentiles over 50 minibatch estimates. Solid dark lines are our adaptive TBPTT methods, dashed colored lines are fixed TBPTT baselines. We see that the adaptive method converges in fewer epochs (left), while maintaining a controlled relative bias $\delta(K) \leq \delta$ (right).

5.2 LANGUAGE MODELING EXPERIMENTS

We also evaluate performance on language modeling tasks, where the goal is to predict the next word. We train and evaluate models on both the Penn Treebank (PTB) corpus (Marcus et al., 1993; Mikolov et al., 2010) and the Wikitext-2 (Wiki2) corpus (Merity et al., 2016). The PTB corpus contains about 1 millon tokens with a truncated vocabulary of 10k. The Wikitext-2 is twice the size of PTB and with a vocabulary of 30k.

Model and Training Setup For both the PTB and Wiki2 corpus, we train 1-layer LSTMs with a word embedding layer input and a linear output layer. The embedding dimension, hidden state, and cell state dimensions are all 900 for the PTB following (Lei et al., 2017) and 512 for the Wiki2 corpus following (Miller and Hardt, 2019). We use a batchsize of $S = 32$ and a fixed learning rate of $\gamma = 10$ for fixed TBPTT $K \in [10, 50, 100, 200, 300]$ and our adaptive TBPTT method $\delta \in [0.9, 0.5, 0.1]$. We set $W = 400$, $K_0 = 100$ and $[K_{\min}, K_{\max}] = [10, 400]$ for Algorithm 1.

Results Figure 2 (left) presents the test PPL and K_n against the training epoch for both language modeling tasks. We again see that our adaptive methods are competitive with the best fixed K methods, while controlling the relative bias. From the $K(\delta, \theta_n)$ vs epoch figures, our adaptive method seems to quickly converge to a constant. Therefore, on the real language data task, we have transformed the problem of selecting a fixed-K to choosing a continuous parameter $\delta \in (0, 1)$. Additional figures for the validation PPL and tables of numerical values can be found in the Supplement.

5.3 EMPIRICALLY CHECKING (A-1)

Figure 3 plots the gradient norm $\phi_k = \partial \mathcal{L}_s / \partial h_{s-k}$ vs k evaluated at the best θ_n (as measured on the validation set). Note that the y-axis is on a log-scale We see that the expected norm $\hat{P}_S[\phi_k]$ (blue-line) of the gradients decay geometrically for large k; however any individual ϕ_k (gray lines) are quite noisy and do not strictly decay. Therefore it appears that our RNNs satisfy (A-1), even though they are unstable, and thus the relative bias can be bounded.

5.4 CHALLENGES IN HIGHER DIMENSIONS

During our experiments, we found that when training RNNs with high-dimensional h, but without introducing regularization on θ (in the form of dropout or weight decay), our estimates $\hat{\beta}$ were often close to or greater than 1; therefore our conservative relative error bound lead to extremely large (impractical) truncation estimates K. During inspection, we found that although most dimensions of $\frac{\partial \mathcal{L}_s}{\partial h_{s-k}}$ decay rapidly with k, a few dimensions did not and these dimensions cause the overall norm

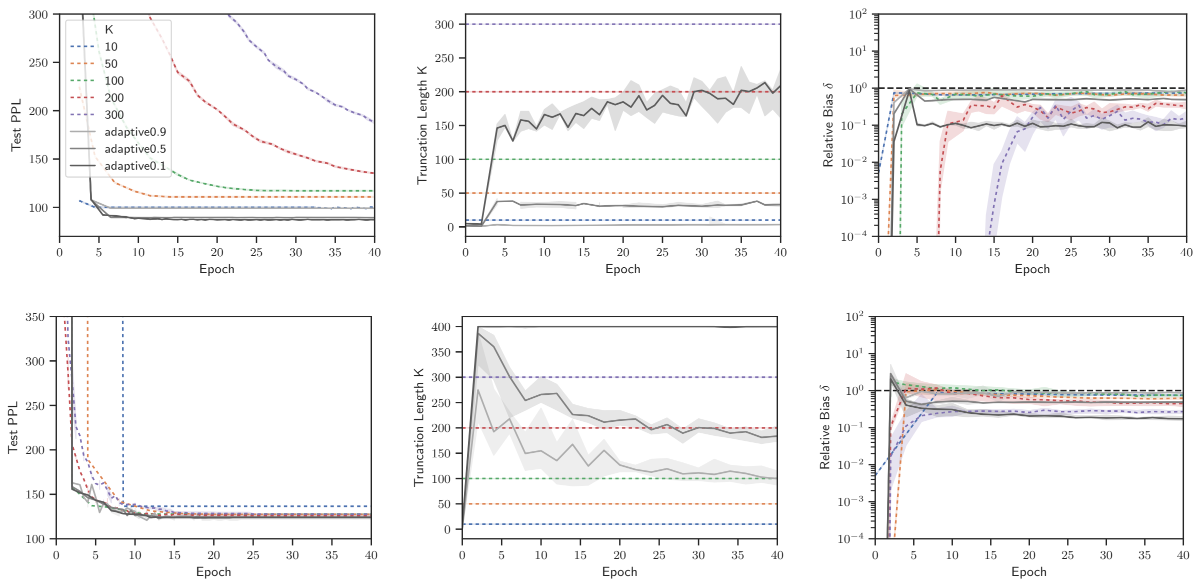

Figure 2: Language Modeling Results. (left) Test PPL vs epoch, (center) $\hat{\kappa}(\delta, \theta_n)$ vs Epoch, (right) $\hat{\delta}(K)$ vs epoch. (Top) PTB (bottom) Wiki2. Error bars are 95 percentiles over 50 minibatch estimates. Solid dark lines are our adaptive TBPTT methods, dashed colored lines are fixed TBPTT baselines. Our adaptive methods are competitive with the best fixed K methods, while controlling the relative bias.

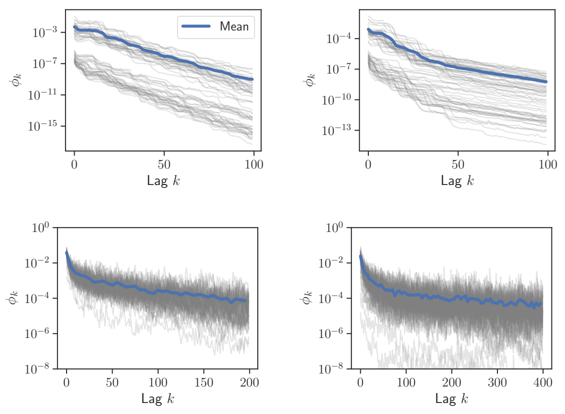

Figure 3: Gradient Norms ϕ_k vs k at the best θ showing geometric decay *on average* (blue line) for large k. The gray lines are separate draws of ϕ_k. (Top-left) fixed-length 'copy' task, (top-right) variable-length 'copy' task, (bottom-left) PTB, (bottom-right) Wiki2.

$\|\frac{\partial \mathcal{L}_s}{\partial h_{s-k}}\|$ to decay slowly, thus $\hat{\beta} \approx 1$. However if these dimensions do not influence $\partial \mathcal{L}/\partial \theta$ (i.e. if $\frac{\partial h_t}{\partial \theta}$ is close to zero), then these dimensions should be ignored. Therefore, to better apply our results to higher-dimensional h, we suspect one should replace the Euclidean norm with a norm that weights dimensions of $\frac{\partial \mathcal{L}_s}{\partial h_{s-t}}$ by $\frac{\partial h_t}{\partial \theta}$ (such as the Mahalanobis norm $\|x\|_\Sigma = x^T \Sigma^{-1} x$ for some positive definite matrix Σ), but we leave this for future work.

6 DISCUSSION

In this work, we developed an adaptively truncating BPTT scheme for RNNs that satistify a generalized vanishing gradient property. We show that if the gradient decays geometrically *in expectation* (A-1), then we can control the relative bias of TBPTT (Theorem 1) and guarantee non-asymptotic convergence bounds for SGD (Theorem 2). We additionally show how to take advantage of these ideas in practice in Algorithm 1, by developing estimators for the relative bias based on backpropagated gradients. We evaluate our proposed method on synthetic copy tasks and language modeling and find it performs similarly to the best fixed-K TBPTT schemes, while still controlling the relative bias of the gradient estimates. In future work, we are interested in methods that restrict the parameters to $\Theta_{\beta,\tau}$ and alternatives to the Euclidean norm for our error bounds in Section 3.2.

Acknowledgements

We thank the reviewers for their feedback. We also thank members of the Dynamode lab at UW for their helpful discussions. This work was supported in part by ONR Grant N00014-18-1-2862 and NSF CAREER Award IIS-1350133.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *Symposium on Operating Systems Design and Implementation*, pages 265–283, 2016.

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128, 2016.

Francois Belletti, Alex Beutel, Sagar Jain, and Ed Chi. Factorized recurrent neural architectures for longer range dependence. In *International Conference on Artificial Intelligence and Statistics*, pages 1522–1530, 2018.

Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Dimitri P Bertsekas and John N Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*, volume 23. Prentice hall Englewood Cliffs, NJ, 1989.

Jie Chen and Ronny Luss. Stochastic gradient descent with biased but consistent gradient estimators. *arXiv preprint arXiv:1807.11880*, 2018.

Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. Recurrent marked temporal point processes: Embedding event history to vector. In *SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1555–1564. ACM, 2016.

Saeed Ghadimi and Guanghui Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4): 2341–2368, 2013.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Li Jing, Yichen Shen, Tena Dubcek, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. Tunable efficient unitary neural networks (EUNN) and their application to RNNs. In *International Conference on Machine Learning*, pages 1733–1741, 2017.

Thomas Laurent and James von Brecht. A recurrent neural network without chaos. In *International Conference on Learning Representations*, 2017.

Tao Lei, Yu Zhang, and Yoav Artzi. Training RNNs as fast as CNNs. *arXiv preprint arXiv:1709.02755*, 2017.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The Penn treebank. 1993.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *International Speech Communication Association*, 2010.

John Miller and Moritz Hardt. Stable recurrent models. In *International Conference on Learning Representations*, 2019.

Asier Mujika, Florian Meier, and Angelika Steger. Approximating real-time recurrent learning with random Kronecker factors. In *Advances in Neural Information Processing Systems*, pages 6594–6603, 2018.

Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*, volume 87. Springer Science & Business Media, 2013.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

BT Poljak and Ya Z Tsypkin. Pseudogradient adaptation and training algorithms. *Automation and Remote Control*, 34:45–67, 1973.

Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabas Poczos, and Alex Smola. Stochastic variance reduction for nonconvex optimization. In *International Conference on Machine Learning*, pages 314–323, 2016.

Ilya Sutskever. *Training Recurrent Neural Networks*. University of Toronto Toronto, Ontario, Canada, 2013.

Eugene Vorontsov, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal. On orthogonality and learning recurrent networks with long term dependencies. In *International Conference on Machine Learning*, pages 3570–3578, 2017.

Paul J Werbos et al. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, Architectures, and Applications*, 433, 1995.