AdaLoss: A computationally-efficient and provably convergent adaptive gradient method

Xiaoxia Wu[1*], **Yuege Xie**[2], **Simon Du**[3], **and Rachel Ward**[2]

[1] Microsoft [2] University of Texas at Austin [3] University of Washington

Abstract

We propose a computationally-friendly adaptive learning rate schedule, "AdaLoss", which directly uses the information of the loss function to adjust the stepsize in gradient descent methods. We prove that this schedule enjoys linear convergence in linear regression. Moreover, we provide a linear convergence guarantee over the non-convex regime, in the context of two-layer over-parameterized neural networks. If the width of the first-hidden layer in the two-layer networks is sufficiently large (polynomially), then AdaLoss converges robustly *to the global minimum* in polynomial time. We numerically verify the theoretical results and extend the scope of the numerical experiments by considering applications in LSTM models for text clarification and policy gradients for control problems.

1 Introduction

Gradient-based methods are widely used in optimizing neural networks. One crucial component in gradient methods is the learning rate (a.k.a. step size) hyper-parameter, which determines the convergence speed of the optimization procedure. An optimal learning rate can speed up the convergence but only up to a certain threshold value; once it exceeds this threshold value, the optimization algorithm may no longer converge. This is by now well-understood for convex problems; excellent works on this topic include [26], [4], [27], [14], [6], and the recent review for large-scale stochastic optimization [5].

While determining the optimal step size is theoretically important for identifying the optimal convergence rate, the optimal learning rate often depends on certain unknown parameters of the problem. For example, for a convex and L-smooth objective function, the optimal learning rate is $O(1/L)$ where L is often unknown to practitioners. To solve this problem, adaptive methods [12, 24] are proposed since they can change the learning rate on-the-fly according to gradient information received along the way. Though these methods often introduce additional hyper-parameters compared to gradient descent (GD) methods with well-tuned stepsizes, the adaptive methods are provably robust to their hyper-parameters in the sense that they still converge at suboptimal parameter specifications, but modulo (slightly) slower convergence rate [19, 31]. Hence, adaptive gradient methods are widely used by practitioners to save a large amount of human effort and computer power in manually tuning the hyper-parameters.

[*] Work done at UT Austin. See full version (including appendix) https://arxiv.org/abs/2109.08282

Among many variants of adaptive gradient methods, one that requires a minimal amount of hyper-parameter tuning is *AdaGrad-Norm* [31], which has the following update

$$\mathbf{w}_{j+1} = \mathbf{w}_j - \eta \nabla f_\xi(\mathbf{w}_j)/b_{j+1} \tag{1}$$

$$\text{with} \quad b_{j+1}^2 = b_j^2 + \|\nabla f_\xi(\mathbf{w}_j)\|^2; \tag{2}$$

above, $\mathbf{w}_j$ is the target solution to the problem of minimizing the finite-sum objective function $F(\mathbf{w}) := \sum_{i=1}^n f_i(\mathbf{w})$, and $\nabla f_\xi(\mathbf{w}_j)$ is the stochastic (sub)-gradient that depends on the random index $\xi \sim \text{Unif}\{1, 2, \dots\}$ satisfying the conditional equality $\mathbb{E}_\xi[\nabla f_\xi(\mathbf{w}_j)|\mathbf{w}_j] = \nabla F(\mathbf{w}_j)$. However, computing the norm of the (sub)-gradient $\nabla f_\xi(\mathbf{w}_j) \in \mathbb{R}^d$ in high dimensional space, particularly in settings which arise in training deep neural networks, is not at all practical. Inspired by a Lipschitz relationship between the objective function F and its gradient for a certain class of objective functions (see details in Section 2), we propose the following scheme for b_k which is significantly more computationally tractable compared to computing the norm of the gradient[1]:

$$\textbf{AdaLoss} \quad b_{j+1}^2 = b_j^2 + \alpha|f_\xi(\mathbf{w}_j) - c|$$

where $\alpha > 0$ and c are the tuning parameters. With this update, we theoretically show that AdaLoss converges with an upper bound that is tighter than AdaGrad-Norm under certain conditions.

Theoretical investigations into adaptive gradient methods for optimizing neural networks are scarce. Existing analyses only deal with general (non)-convex and smooth functions, and thus, only concern convergence to first-order stationary points [21, 8]. However, it is sensible to instead target global convergence guarantees for adaptive gradient methods in this setting in light of a series of recent breakthrough papers showing that (stochastic) GD can converge to the global minima of over-parameterized neural networks [11, 10, 22, 1, 40]. By adapting their analysis, we are able to answer the following open question:

[1] Code is available at https://github.com/willway1023yx/adaloss.

What is the iteration complexity of adaptive gradient methods in over-parameterized networks?

In addition, we note that these papers require the step size to be sufficiently small to guarantee global convergence. In practice, these optimization algorithms can use a much larger learning rate while still converging to the global minimum. Thus, we make an effort to answer the question:

What is the optimal stepsize in optimizing neural networks?

Contributions. First, we study AdaGrad-Norm [31] in the linear regression setting and significantly improve the constants in the convergence bounds—$\mathcal{O}\left(L^2/\epsilon\right)$ [31, 34][2] in the deterministic gradient descent setting—to a near-constant dependence $\mathcal{O}\left(\log(L/\epsilon)\right)$ (Theorem 1).

Second, we develop an adaptive gradient method called *AdaLoss* that can be viewed as a variant of the "norm" version of AdaGrad but with better computational efficiency and easier implementation. We provide theoretical evidence that AdaLoss converges at the same rate as AdaGrad-Norm *but with a better convergence constant* in the setting of linear regression (Corollary 1).

Third, for an overparameterized two-layer neural network, we show the learning rate of GD can be improved to the rate of $\mathcal{O}(1/\|\mathbf{H}^{\infty}\|)$ (Theorem 2) where $\mathbf{H}^{\infty}$ is a Gram matrix which only depends on the data.[3] We further prove AdaLoss converges to the global minimum in polynomial time and does so robustly, in the sense that *for any choice of hyper-parameters* used, our method is guaranteed to converge to the global minimum in polynomial time (Theorem 3). The choice of hyper-parameters only affects the rate but not the convergence. In particular, we provide explicit expressions for the polynomial dependencies in the parameters required to achieve global convergence.[4]

We numerically verify our theorems in linear regression and a two-layer neural network (Figure 1, 2 and 3). To demonstrate the easy implementation and extension of our algorithm for practical purposes, we perform experiments in a text classification example using LSTM models, as well as for a control problem using policy gradient methods (Section 5).

Related Work. Closely related work to ours can be divided into two categories as follows.

Adaptive Gradient Methods. Adaptive Gradient (AdaGrad) Methods, introduced independently by Duchi, Hazan, and Singer [12] and McMahan and Streeter [24], are now widely used in practice for online learning due in part to their robustness to the choice of stepsize. The first convergence guarantees proved in [12] were for the setting of online convex optimization, where the loss function may change from iteration to iteration. Later convergence results for variants of AdaGrad were proved in [19] and [25] for offline convex and strongly convex settings. In the non-convex and smooth setting, Ward, Wu, and Bottou [31] and Li and Orabona [21] prove that the "norm" version of AdaGrad converges to a stationary point at rate $O\left(1/\varepsilon^2\right)$ for stochastic GD and at rate $O\left(1/\varepsilon\right)$ for batch GD. Many modifications to AdaGrad have been proposed, namely, RMSprop [15], AdaDelta [37], Adam [16], AdaFTRL[28], SGD-BB[30], AcceleGrad [20], Yogi [35], Padam [7], to name a few. More recently, accelerated adaptive gradient methods have also been proven to converge to stationary points [3, 8, 36, 38, 41].

Global Convergence for Neural Networks. A series of papers showed that gradient-based methods provably reduce to zero training error for over-parameterized neural networks [11, 10, 22, 1, 40]. In this paper, we study the setting considered in [11], which showed that for learning rate $\eta = O(\lambda_{\min}(\mathbf{H}^{\infty})/n^2)$, GD finds an ε-suboptimal global minimum in $O\left(\log(1/\epsilon)\right)/(\eta\lambda_{\min}(\mathbf{H}^{\infty}))$ iterations for the two-layer over-parameterized ReLU-activated neural network. As a by-product of the analysis in this paper, we show that the learning rate can be improved to $\eta = O(1/\|\mathbf{H}^{\infty}\|)$ which results in faster convergence. We believe the proof techniques developed in this paper can be extended to deep neural networks, following recent works [10, 1, 40].

Notation Throughout, $\|\cdot\|$ denotes the Euclidean norm if it applies to a vector and the maximum eigenvalue if it applies to a matrix. We use $N(\mathbf{0}, \mathbf{I})$ to denote a standard Gaussian distribution, where $\mathbf{I}$ denotes the identity matrix and $U(S)$ denotes the uniform distribution over a set S. We use $[n] := \{0, 1, \ldots, n\}$, and we write $[\mathbf{x}]_i$ to denote the entry of the i-th dimension of the vector $\mathbf{x}$.

2 AdaLoss Stepsize

Let $\{Z_1, \ldots, Z_n\}$ be empirical samples drawn uniformly from an unknown underlying distribution $\mathcal{S}$. Define $f_i(\mathbf{w}) = f(\mathbf{w}, Z_i) : \mathbb{R}^d \to \mathbb{R}, i = 1, 2, \ldots, n$. Consider minimizing the empirical risk defined as finite sum of $f_i(\mathbf{w})$ over $i \in [n]$. The standard algorithm is stochastic gradient descent (SGD) with an appropriate step-size [5]. Stepsize tuning for optimization problems, including training neural networks, is generally challenging because the convergence of the algorithm is very sensitive to the stepsize: too small values of the stepsize mean slow progress while too large values lead to the divergence of the algorithm.

To find a suitable learning rate schedule, one could use the information on past and present gradient norms as described in equation (2), and the convergence rate for SGD is $\mathcal{O}\left(1/\varepsilon^2\right)$, the same order as for well-tuned stepsize [19, 21, 31]. However, in high dimensional statistics, particularly in the widespread application of deep neural networks, computing the norm of the (sub)-gradient $\nabla f_i(\mathbf{w}_j) \in \mathbb{R}^d$ for $i \in [n]$ at every iteration j is impractical. To tackle the problem, we recall the popular setting of linear regression and two-layer network regression [10], which assumes at optimal $\mathbf{w}^*$, $\nabla f_i^* = 0$, then

$$\|\nabla f_i(\mathbf{w})\|^2 = \|\nabla f_i(\mathbf{w}) - \nabla f_i^*\|^2 \leq C|f_i(\mathbf{w}) - f_i^*|.$$

The norm of the gradient is bounded by the difference between $f_i(\mathbf{w}_j)$ and f_i^*. The optimal value f_i^* is a fixed number,

[2]The rate is for Case (2) of Theorem 3 in [34] and of Theorem 2.2 in [31]. L is $\bar{\lambda}_1$ in Theorem 1.

[3]Note that this upper bound is independent of the number of parameters. As a result, using this stepsize, we show GD enjoys a faster convergence rate. This choice of stepsize directly leads to an improved convergence rate compared to [11].

[4]Note that this section has greatly subsumes [32]. However, Theorem 3 is a much improved version compared to Theorem 4.1 in [32]. This is due to our new inspiration from Theorem 1.

Algorithm 1: AdaLoss Algorithm

```
1: Input: Initialize $\mathbf{w}_0 \in \mathbb{R}^d, b_0 > 0, c > 0, j \leftarrow 0$, and the total iterations $T$.
2: for $j = 1, 2, 3, \dots T$ do
3:     Generate a random index $\xi_j$
4:     $b_{j+1}^2 \leftarrow b_j^2 + \alpha|f_{\xi_j}(\mathbf{w}_j) - c|$
5:     $\mathbf{w}_{j+1} \leftarrow \mathbf{w}_j - \frac{\eta}{b_{j+1}}\nabla f_{\xi_j}(\mathbf{w}_j)$
6: end for
```

which could possibly be known as prior or estimated under some conditions. For instance, for an over-determined linear regression problem or over-parameterized neural networks, we know that $f_i^* = 0$. For the sake of the generality of our proposed algorithm, we replace f^* with a constant c. Based on the observation, we propose the update in Algorithm 1.

Our focus is b_{k+1}, a parameter that is changing at every iteration according to the loss value of previous computational outputs. There are four positive hyper-parameters, b_0, η, α, c, in the algorithm. η is for ensuring homogeneity and that the units match. b_0 is the initialization of a monotonically increasing sequence $\{b_k\}_{k=1}^\infty$. The parameter α is to control the rate of updating $\{b_k\}_{k=1}^\infty$ and the constant c is a surrogate for the ground truth value f^* ($c = 0$ if $f^* = 0$).

The algorithm makes a significant improvement in *computational efficiency* by using the direct feedback of the value of (stochastic) loss. For the above algorithm, $\xi_j \sim \text{Unif}\{1, 2, \dots, n\}$ satisfies the conditional equality $\mathbb{E}_{\xi_j}[\nabla f_{\xi_j}(\mathbf{w}_j)|\mathbf{w}_j] = \nabla F(\mathbf{w}_j)$. As a nod to the use of the information of the stochastic loss for the stepsize schedule, we call this method **adaptive loss** (AdaLoss); we focus on analysis of this algorithm on linear regression and two-layer over-parameterized neural networks.

3 AdaLoss in Linear Regression

Consider the linear regression:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2, \quad \mathbf{y} \in \mathbb{R}^n \text{ and } \mathbf{X} \in \mathbb{R}^{n\times d} \tag{3}$$

Suppose the Gram matrix $\mathbf{X}^\top\mathbf{X}$ is a positive definite matrix with the smallest singular value $\bar{\lambda}_0 > 0$, and the largest singular value $\bar{\lambda}_1 > 0$. Denote $\mathbf{V}$ the unitary matrix from the singular value decomposition of $\mathbf{X}^\top\mathbf{X} = \mathbf{V}\Sigma\mathbf{V}^T$. Suppose we have the optimal solution $\mathbf{X}\mathbf{w}^* = \mathbf{y}$. The recent work [34] implies that the convergence rate using the adaptive stepsize update in (2) enjoys linear convergence. However, the linear convergence is under the condition that the effective learning rate $2\eta/b_0$ is less than the critical threshold $1/\bar{\lambda}_1$ (i.e.,$b_0 \geq \eta\bar{\lambda}_1/2$). If we initialize the effective learning rate larger than the threshold, the algorithm falls back to a sub-linear convergence rate with an order $\mathcal{O}\left(\bar{\lambda}_1/\varepsilon\right)$. Suspecting that this might be due to an artifact of the proof, we here tighten the bound that admits the linear convergence $\mathcal{O}\left(\log\left(1/\varepsilon\right)\right)$ for any b_0 (Theorem 1).

Theorem 1. ***(Improved AdaGrad-Norm Convergence)*** *Consider the problem* (3) *and*

$$\mathbf{w}_{t+1} = \mathbf{w}_t - (\eta/b_{t+1})\,\mathbf{X}^T\left(\mathbf{X}\mathbf{w}_t - \mathbf{y}\right) \tag{4}$$

$$\textit{with} \quad b_{t+1}^2 = b_t^2 + \|\mathbf{X}^T\left(\mathbf{X}\mathbf{w}_t - \mathbf{y}\right)\|^2 \tag{5}$$

We have $\|\mathbf{w}_T - \mathbf{w}^*\|^2 \leq \epsilon$ *for* [5]

$$T = \widetilde{\mathcal{O}}\left(\left(\max\{\frac{b_0}{\bar{\lambda}_1}, \eta\} + \frac{\Delta_X}{\eta\bar{\lambda}_1} + \frac{\eta}{s_0^2}T_1\right)\frac{\bar{\lambda}_1\log\left(1/\epsilon\right)}{\eta\bar{\lambda}_0}\right). \tag{6}$$

where $\Delta_X = \|\mathbf{X}(\mathbf{w}_0 - \mathbf{w}^*)\|^2$ $s_0 := [\mathbf{V}^\top\mathbf{w}_0 - \mathbf{V}^\top\mathbf{w}^*]_1$*;* $T_1 = \log\left(\frac{\left(\eta\bar{\lambda}_1\right)^2 - 4(b_0)^2}{\eta^2\bar{\lambda}_1^2}\right)\mathbb{1}_{\{2b_0\leq\eta\bar{\lambda}_1\}}$ *and . Here* $[\cdot]_1$ *corresponds to the dimension scaled by the largest singular value of* $\mathbf{X}^\top\mathbf{X}$ *(see (23) in appendix), i.e.* $\bar{\lambda}_1$*.*

We state the explicit complexity T in Theorem 4 and the proof is in Section A. Our theorem significantly improves the sub-linear convergence rate when $b_0 \leq \eta\bar{\lambda}_1/2$ compared to [34] and [31]. The bottleneck in their theorems for small b_0 is that they assume the dynamics b_t updated by the gradient $\|\mathbf{X}^T\left(\mathbf{X}\mathbf{w}_t - y\right)\|^2 \approx \varepsilon$ for all $j = 0, 1, \dots,$ which results in taking as many iterations as $N \approx ((\eta\bar{\lambda}_1)^2 - 4b_0^2)/\epsilon$ in order to get $b_N \geq \eta\bar{\lambda}_1/2$. Instead, we explicitly characterize $\{b_t\}_{t\geq 0}$. That is, for each dimension $i \in [d]$,

$$\begin{aligned} s_{t+1}^{(i)} &:= \left([\mathbf{V}^\top\mathbf{w}_{t+1}]_i - [\mathbf{V}^\top\mathbf{w}^*]_i\right)^2 \\ &= \left(1 - \eta\bar{\lambda}_i/b_{t+1}\right)^2\left([\mathbf{V}^\top\mathbf{w}_t]_i - [\mathbf{V}^\top\mathbf{w}^*]_i\right)^2. \end{aligned} \tag{7}$$

If $b_0 \leq \eta\bar{\lambda}_i/2$, each i-th sequence $\{s_t^{(i)}\}_{t=0}^k$ is monotone increasing up to $b_k \leq \frac{\eta\bar{\lambda}_i}{2}$, thereby taking significantly fewer iterations, *independent of the prescribed accuracy* ε, for b_t to reach the critical value $\eta\bar{\lambda}_1/2$ shown in the following lemma.

Lemma 1. ***(Exponential Increase for*** $2b_0 < \eta\bar{\lambda}_1$***)*** *Consider the same setting as Theorem 1. Suppose we start with small initialization:* $0 < b_0 < \bar{\lambda}_1/2$*. For the update of AdaGrad-Norm in* (4)*, there exists the first index* N *such that* $b_{N+1} \geq \bar{\lambda}_1/2$ *and* $b_N < \bar{\lambda}_1/2$*, and* N *satisfies*

$$\textbf{\textit{(AdaGrad-Norm)}}\ N \leq \frac{\log\left(1 + \frac{\left(\eta\bar{\lambda}_1\right)^2 - 4(b_0)^2}{\eta^2\bar{\lambda}_1^2}\right)}{\log\left(1 + \frac{4}{\eta^2}\left([\mathbf{V}^T\mathbf{w}_0]_1 - [\mathbf{V}^T\mathbf{w}^*]_1\right)^2\right)} + 1$$

For AdaLoss with $b_{t+1}^2 = b_t^2 + \|\mathbf{X}\mathbf{w}_t - \mathbf{y}\|^2$*,* N *satisfies*

$$\textbf{\textit{(AdaLoss)}}\ N \leq \frac{\log\left(1 + \frac{\left(\eta\bar{\lambda}_1\right)^2 - 4b_0^2}{\eta^2\bar{\lambda}_1}\right)}{\log\left(1 + \frac{4}{\eta^2\bar{\lambda}_1}\left([\mathbf{V}^T\mathbf{w}_0]_1 - [\mathbf{V}^T\mathbf{w}^*]_1\right)^2\right)} + 1$$

Suppose $\bar{\lambda}_1 > 1$. For AdaLoss, we see that when the initialization $2b_0 \leq \eta\bar{\lambda}_1$, b_t updated by AdaLoss is more likely to take more iterations than AdaGrad-Norm to reach a value greater than $\eta\bar{\lambda}_1$ (see the red part in Lemma 1). Furthermore, a more interesting finding is that AdaLoss's upper bound is smaller than AdaGrad-Norm's if $\|\mathbf{X}\left(\mathbf{w}_t - \mathbf{w}^*\right)\|^2 \geq \|\mathbf{w}_t - \mathbf{w}^*\|^2$ (see Lemma 2). Thus, the upper bound of AdaLoss could be potentially tighter than AdaGrad-Norm when $2b_0 \geq \eta\bar{\lambda}_1$, but possibly looser than AdaGrad-Norm

[5] $\widetilde{\mathcal{O}}$ hide logarithmic terms.

when $2b_0 \leq \eta\bar{\lambda}_1$. To see this, we follow the same process and have the convergence of AdaLoss stated in Corollary 1.

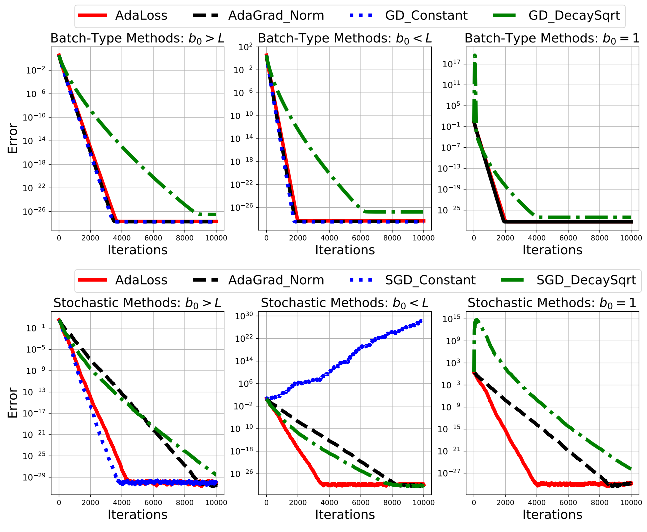


Figure 1: The left three plots are linear regression in the deterministic setting, and the right three plots in the stochastic setting. The x-axis is the number of iterations, and y-axis is the error $\|\mathbf{w}_t - \mathbf{w}^*\|^2$ in log scale. Each curve is an independent experiment for algorithms: AdaLoss (red), AdaGrad-Norm (black), (stochastic) GD with constant stepsize (blue) and SGD with square-root decaying stepsize (green).

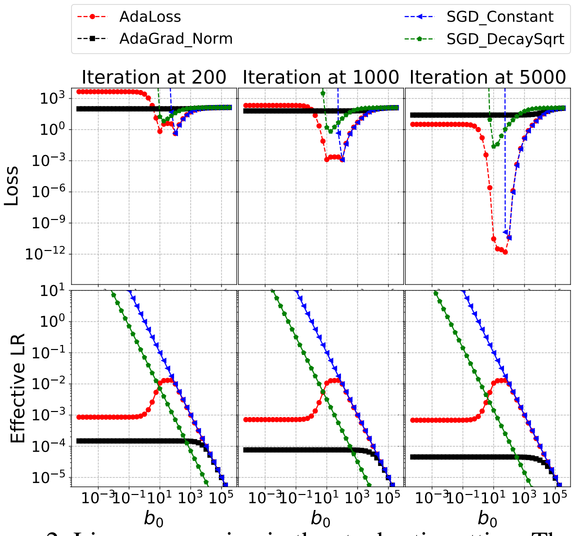


Figure 2: Linear regression in the stochastic setting. The top (bottom) 3 figures plot the average of loss (effective stepsize $1/b_t$) w.r.t. b_0, for iterations t in $[101, 200]$, $[991, 1000]$ and $[4901, 5000]$ respectively.

Corollary 1. ***(AdaLoss Convergence)*** *Consider the same setting as Theorem 1 but with the b_t updated by: $b_{t+1}^2 = b_t^2 + \|\mathbf{X}\mathbf{w}_t - \mathbf{y}\|^2$. We have $\|\mathbf{w}_T - \mathbf{w}^*\|^2 \leq \epsilon$ for*

$$T = \widetilde{\mathcal{O}}\left(\left(\max\{\frac{b_0}{\bar{\lambda}_1}, \eta\} + \frac{\Delta}{\eta\bar{\lambda}_1} + \frac{\eta}{s_0^2}\widetilde{T}_1\right)\frac{\bar{\lambda}_1 \log(1/\epsilon)}{\eta\bar{\lambda}_0}\right), \quad (8)$$

where $\widetilde{T}_1 = \log\left(\frac{(\eta\bar{\lambda}_1)^2 - 4(b_0)^2}{\eta^2\bar{\lambda}_1}\right)\mathbb{1}_{\{2b_0 \leq \eta\bar{\lambda}_1\}}$ and $\Delta = \|\mathbf{w}_0 - \mathbf{w}^*\|^2$.

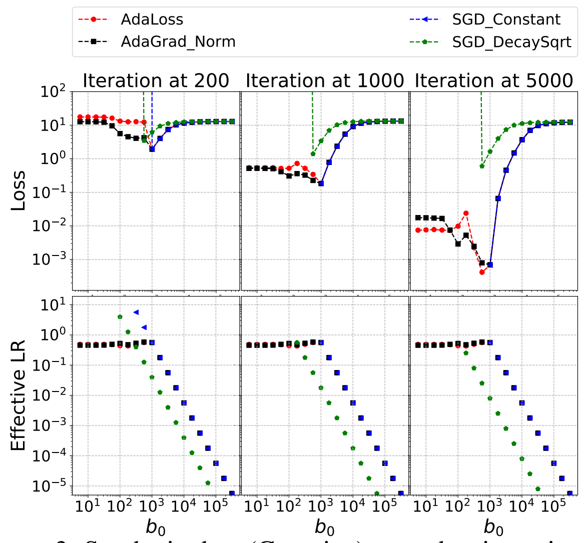


Figure 3: Synthetic data (Gaussian) – stochastic setting for two-layer neural network. The top (bottom) 3 figures plot the average of loss (effective stepsize $1/b_t$) w.r.t. b_0 for iterations t in $[101, 200]$, $[991, 1000]$ and $[4901, 5000]$. Top row: all four curves overlap after $b_0 \geq 10^4$, while $b_0 \in [10^3, 10^4]$, red,blue and black curves. Bottom row: all colors overlap after $b_0 \geq 10^3$; red and black curves overlap for all b_0.

For the explicit form of T, see Corollary 2 in the appendix. Suppose the first term in the bounds of (6) and (8) takes the lead and $\lambda_1 > 1$, AdaLoss has a tighter upper bound than AdaGrad-Norm, unless $\bar{\lambda}_1 \leq 1$. Hence, our proposed computationally-efficient method AdaLoss is a preferable choice in practice since $\bar{\lambda}_1$ is usually not available.[6]

For general functions, stochastic GD (SGD) is often limited to a sub-linear convergence rate [19, 40, 31]. However, when there is no noise at the solution ($\nabla f_i(x^*) = 0$ for all i), we prove that the limit of b_t is bounded, $\lim_{t\to\infty} b_t < \infty$, which ensures the linear convergence. Observe the update:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta}{b_{t+1}}\left(\mathbf{x}_{\xi_t}^\top \mathbf{w}_t - y_{\xi_t}\right)\mathbf{x}_{\xi_t} \text{ with } b_{t+1}^2 = b_t^2 + R_t$$

Here, R_t is defined for AdaGrad-Norm and AdaLoss as

$$\textbf{(AdaGrad-Norm)}\ R_t = \|\mathbf{x}_{\xi_t}\left(\mathbf{x}_{\xi_t}^\top \mathbf{w}_t - y_{\xi_t}\right)\|^2 \quad (9)$$

$$\textbf{(AdaLoss)}\ R_t = \left(\mathbf{x}_{\xi_t}^\top \mathbf{w}_t - y_{\xi_t}\right)^2. \quad (10)$$

We show the linear convergence for AdaGrad-Norm by replacing general strongly convex functions [34] with linear regression (Theorem 7). For AdaLoss, we follow the same process of their proof and derive the convergence in Theorem 8. Due to page limit, we put them in the appendix. The main discovery in this process is the crucial step—inequality (63)—that improves the bound using AdaLoss. The intuition is that

[6] Although one cannot argue that an algorithm is better than another by comparing their worse-case upper bounds, it might give some implication of the overall performance of the two algorithms considering the derivation of their upper bounds is the same.

the add-on value of AdaLoss, $(\mathbf{x}_{\xi_t}^\top \mathbf{w}_t - y_{\xi_t})^2$, is smaller than that of AdaGrad-Norm ($\|\mathbf{x}_{\xi_t}(\mathbf{x}_{\xi_t}^\top \mathbf{w}_t - y_{\xi_t})\|^2$).

In Proposition 1, we compare the upper bounds of Theorem 7 and Theorem 8. The proposition shows that using AdaLoss in the stochastic setting achieves a tighter convergence bound than AdaGrad-Norm when $2b_0 \geq \eta\bar{\lambda}_1$.

Proposition 1. ***(Stochastic AdaLoss v.s. Stochastic AdaGrad-Norm)*** *Consider the problem* (3) *where* $\bar{\lambda}_1 > 1$ *and the stochastic gradient method in* (10) *with* $2b_0 \geq \eta \sup_i \|\mathbf{x}_i\|$. *AdaLoss improves the constant in the convergence rate of AdaGrad-Norm up to an additive factor:* $(\bar{\lambda}_1 - 1)\|\mathbf{w}_0 - \mathbf{w}^*\|^2$.

Numerical Experiments. To verify the convergence results in linear regression, we compare four algorithms: (a) AdaLoss with $1/b_t$, (b) AdaGrad-Norm with $1/b_t$ (c) SGD-Constant with $1/b_0$, (d) SGD-DecaySqrt with $1/(b_0 + c_s\sqrt{t})$ (c_s is a constant). See Appendix D for experimental details. Figure 1 implies that AdaGrad-Norm and AdaLoss behave similarly in the deterministic setting, while AdaLoss performs much better in the stochastic setting, particularly when $b_0 \leq L =: \sup_i \|\mathbf{x}_i\|$. Figure 2 implies that stochastic AdaLoss and AdaGrad-Norm are robust to a wide range of initialization of b_0. Comparing AdaLoss with AdaGrad-Norm, we find that when $b_0 \leq 1$, AdaLoss is not better than AdaGrad-Norm at the beginning (at least before 1000 iterations, see the first two figures at the top row), albeit the effective learning rate is much larger than AdaGrad-Norm. However, after 5000 iterations (3rd figure, 1st row), AdaLoss outperforms AdaGrad-Norm in general.

4 AdaLoss in Two-Layer Networks

We consider the same setup as in [11] where they assume that the data points, $\{\mathbf{x}_i, y_i\}_{i=1}^n$, satisfy

Assumption 1. *For* $i \in [n]$, $\|\mathbf{x}_i\| = 1$ *and* $|y_i| = O(1)$.

The assumption on the input is only for the ease of presentation and analysis. The second assumption on labels is satisfied in most real-world datasets. We predict labels using a two-layer neural network

$$f(\mathbf{W}, \mathbf{a}, \mathbf{x}) = \frac{1}{\sqrt{m}} \sum_{r=1}^{m} a_r \sigma(\langle \mathbf{w}_r, \mathbf{x} \rangle) \qquad (11)$$

where $\mathbf{x} \in \mathbb{R}^d$ is the input, for any $r \in [m]$, $\mathbf{w}_r \in \mathbb{R}^d$ is the weight vector of the first layer, $a_r \in \mathbb{R}$ is the output weight, and $\sigma(\cdot)$ is ReLU activation function. For $r \in [m]$, we initialize the first layer vector with $\mathbf{w}_r(0) \sim N(\mathbf{0}, \mathbf{I})$ and output weight with $a_r \sim U(\{-1, +1\})$. We fix the second layer and train the first layer with the quadratic loss. Define $u_i = f(\mathbf{W}, \mathbf{a}, \mathbf{x}_i)$ as the prediction of the i-th example and $\mathbf{u} = [u_1, \ldots, u_n]^\top \in \mathbb{R}^n$. Let $\mathbf{y} = [y_1, \ldots, y_n]^\top \in \mathbb{R}^n$ and

$$L(\mathbf{W}) = \frac{1}{2}\|\mathbf{u} - \mathbf{y}\|^2 \text{ or } L(\mathbf{W}) = \sum_{i=1}^{n} \frac{1}{2}(f(\mathbf{W}, \mathbf{a}, \mathbf{x}_i) - y_i)^2$$

We use k for indexing since $\mathbf{u}(k)$ is induced by $\mathbf{W}(k)$. According to [11], the matrix below determines the convergence rate of GD.

Definition 1. *The matrix* $\mathbf{H}^\infty \in \mathbb{R}^{n\times n}$ *is defined as follows. For* $(i,j) \in [n] \times [n]$.

$$\begin{aligned}\mathbf{H}_{ij}^\infty &= \mathbb{E}_{\mathbf{w}\sim N(\mathbf{0},\mathbf{I})}\left[\mathbf{x}_i^\top \mathbf{x}_j \mathbb{I}\left\{\mathbf{w}^\top \mathbf{x}_i \geq 0, \mathbf{w}^\top \mathbf{x}_j \geq 0\right\}\right] \\ &= \mathbf{x}_i^\top \mathbf{x}_j(\pi - \arccos\left(\mathbf{x}_i^\top \mathbf{x}_j\right))/(2\pi)\end{aligned}$$

This matrix represents the kernel matrix induced by Gaussian initialization and ReLU activation function. We make the following assumption on $\mathbf{H}^\infty$.

Assumption 2. *The matrix* $\mathbf{H}^\infty \in \mathbb{R}^{n\times n}$ *in Definition 1 satisfies* $\lambda_{\min}(\mathbf{H}^\infty) \triangleq \lambda_0 > 0$.

[11] showed that this condition holds as long as the training data is not degenerate. We also define the following empirical version of this Gram matrix, which is used in our analysis. For $(i,j) \in [n] \times [n]$: $\mathbf{H}_{ij} = \frac{1}{m}\sum_{r=1}^m \mathbf{x}_i^\top \mathbf{x}_j \mathbb{I}\{\mathbf{w}_r^\top \mathbf{x}_i \geq 0, \mathbf{w}_r^\top \mathbf{x}_j \geq 0\}$.

We first consider GD with a constant learning rate (η) $\mathbf{W}(k+1) = \mathbf{W}(k) - \eta\frac{\partial L(\mathbf{W}(k))}{\partial \mathbf{W}}$. [11] showed gradient descent achieves zero training loss with learning rate $\eta = O(\lambda_0/n^2)$. Based on the approach of eigenvalue decomposition in [2] (c.f. Lemma 10), we show that the maximum allowable learning rate can be improved from $O(\lambda_0/n^2)$ to $O(1/\|\mathbf{H}^\infty\|)$.

Theorem 2. ***(Gradient Descent with Improved Learning Rate)*** *Under Assumptions 1 and 2, if the number of hidden nodes* $m = \Omega\left(\frac{n^8}{\lambda_0^4 \delta^3}\right)$ *and we set the stepsize* $\eta = \Theta\left(\frac{1}{\|\mathbf{H}^\infty\|}\right)$, *then with probability at least* $1 - \delta$ *over the random initialization, we have* $L(\mathbf{W}(T)) \leq \varepsilon$ *for* [7]

$$T = \widetilde{O}\left((\|\mathbf{H}^\infty\|/\lambda_0)\log(1/\varepsilon)\right)$$

Note that since $\|\mathbf{H}^\infty\| \leq n$, Theorem 2 gives an $O(\lambda_0/n)$ improvement. The improved learning rate also gives a tighter iteration complexity bound $O\left((\|\mathbf{H}^\infty\|/\lambda_0)\log(n/\varepsilon)\right)$, compared to the $O\left((n^2/\lambda_0^2)\log(n/\varepsilon)\right)$ bound in [11]. Empirically, we find that if the data matrix is approximately orthogonal, then $\|\mathbf{H}^\infty\| = O(1)$ (see Figure 6 in Appendix C). Therefore, we show that the iteration complexity of gradient descent is nearly independent of n.

We surprisingly found that there is a strong connection between over-parameterized neutral networks and linear regression. We observe that $\mathbf{H}^\infty$ in the over-parameterized setup and $\mathbf{X}^\top\mathbf{X}$ in linear regression share a strikingly similar role in the convergence. Based on this observation, we combine the induction proof of Theorem 2 with the convergence analysis of Theorem 1. An important observation is that one needs the overparameterization level m to be sufficiently large so that the adaptive learning rate can still have enough "burn in" time to reach the critical value for small initialization b_0, while ensuring that the iterates $\|\mathbf{W}(t) - \mathbf{W}(0)\|_F$ remain sufficiently small and the positiveness of the Gram matrix. Theorem 3 characterizes the convergence rate of AdaLoss.

Theorem 3. ***(Two-layer networks)*** *Consider Assumptions 1 and 2, and suppose the width satisfies* $m =$

[7] $\widetilde{O}$ and $\widetilde{\Omega}$ hide $\log(n), \log(1/\lambda_0), \log(1/\delta)$ terms.

$\Omega\left(\frac{n^8}{\lambda_0^4\delta^3}+\frac{\eta^2 n^6}{\lambda_0^2\delta^2\alpha^2}\left(\frac{\lambda_0}{\alpha^2\sqrt{n}\varepsilon}+\frac{1}{\alpha^2\varepsilon}\right)\mathbb{1}_{b_0<\eta(C(\lambda_0+\|\mathbf{H}^\infty\|))/2}\right)$. *Then, the AdaLoss update using* $b_{k+1}^2=b_k^2+\alpha^2\sqrt{n}\|\mathbf{y}-\mathbf{u}(k)\|^2$ *admits the following convergence results.*
(a) *If* $b_0 \geq \eta C(\lambda_0+\|\mathbf{H}^\infty\|)/2$*, then with probability* $1-\delta$ *with respect to the random initialization, we have* $\min_{t\in[T]}\|\mathbf{y}-\mathbf{u}(t)\|^2\leq\varepsilon$ *after*

$$T=\widetilde{O}\left(\left(\frac{b_0}{\eta\lambda_0}+\frac{\alpha^2 n^{3/2}}{\eta^2\lambda_0^2\delta}\right)\log\left(\frac{1}{\varepsilon}\right)\right).$$

(b) *If* $0<b_0\leq\eta C(\lambda_0+\|\mathbf{H}^\infty\|)/2$*, then with probability* $1-\delta$ *with respect to the random initialization, we have* $\min_{t\in[T]}\|\mathbf{y}-\mathbf{u}(t)\|^2\leq\varepsilon$ *after*

$$T=\widetilde{O}\left(\frac{\lambda_0+\sqrt{n}}{\alpha^2\sqrt{n}\varepsilon}+\left(\frac{\alpha^2 n^{3/2}}{\eta^2\lambda_0^2\delta}+\frac{\|\mathbf{H}^\infty\|^2}{\lambda_0^2}\right)\log\left(\frac{1}{\varepsilon}\right)\right).$$

To our knowledge, this is the first global convergence guarantee of any adaptive gradient method for neural networks robust to initialization of b_0. It improves the results in [31], where AdaGrad-Norm is shown only to converge to a stationary point. Besides the robustness to the hyper-parameter, two key implications in Thm 4.2: (1) Adaptive gradient methods can converge *linearly* in certain two-layer networks using our new technique developed for linear regression (Theorem 1); (2) But that linear convergence and robustness comes with *a cost*: the width of the hidden layer has to be much wider than n^8. That is, when the initialization b_0 satisfying Case (b), the leading rate for m is its second term, i.e., $(\eta^2 n^6)/(\alpha^4\epsilon)$, which is larger than n^8 if ϵ is sufficiently small.

We remark that Theorem 3 is different from Theorem 3 in [34] which achieves convergence by assuming a PL inequality for the loss function. The condition—PL inequality—is not guaranteed in general. The PL inequality is satisfied in our two-layer network problem when the Gram matrix $\mathbf{H}^\infty$ is strictly positive (see Proposition 2). That is, in order to satisfy PL-inequality, we use induction to show that the model has to be sufficiently overparameterized, i.e., $m=O\left((poly(n^8,\alpha,\eta,\lambda_0,\delta,\varepsilon)\right)$.

Theorem 3 applies to two cases. In the first case, the effective learning rate at the beginning (η/b_0) is smaller than the threshold $2/(C(\lambda_0+\|\mathbf{H}^\infty\|))$ that guarantees the global convergence of gradient descent (c.f. Theorem 2). In this case, the convergence has two terms, and the first term $\frac{b_0}{\eta\lambda_0}\log\left(\frac{1}{\epsilon}\right)$ is the standard gradient descent rate if we use η/b_0 as the learning rate. Note this term is the same as Theorem 2 if $\eta/b_0=\Theta(1/\|\mathbf{H}^\infty\|)$. The second term comes from the upper bound of b_T in the effective learning rate η/b_T (c.f. Lemma 11). This case shows if α is sufficiently small that the second term is smaller than the first term, then we have the same rate as gradient descent.

In the second case, the initial effective learning rate, η/b_0, is greater than the threshold that guarantees the convergence of gradient descent. Our algorithm guarantees either of the followings happens after T iterations: (1) The loss is already small, so we can stop training. This corresponds to the first term $(\lambda_0+\sqrt{n})/(\alpha^2\sqrt{n\varepsilon})$. (2) The loss is still large, which makes the effective stepsize, η/b_k, decrease with a good rate, i.e., if (2) keeps happening, the stepsize will decrease till $\eta/b_k\leq 2/(C(\lambda_0+\|\mathbf{H}^\infty\|))$, and then it comes to the first case. Note that the second term here is the same as the second term of the first case, but the third term, $(\|\mathbf{H}^\infty\|/\lambda_0)^2\log(1/\epsilon)$ is slightly worse than the rate of the gradient descent. The reason is that the loss may increase due to the large learning rate at the beginning (c.f. Lemma 12).

When comparing AdaGrad-Norm, one could get the same convergence rate as AdaLoss. The comparison between AdaGrad-Norm and AdaLoss are almost the same as in linear regression. The bounds of AdaGrad-Norm and AdaLoss are similar, since our analysis for both algorithms is the worst-case analysis. However, numerically, AdaLoss can behave better than AdaGrad-Norm: Figure 3 shows that AdaLoss performs almost the same as or even better than AdaGrad-Norm with SGD. As for extending Theorem 3 to the stochastic setting, we leave this for future work. We devote the rest of the space to real data experiments.

Fine-tuning in deeper networks. To see how AdaLoss compared with AdaGrad-Norm and SGD with constant/decay stepsize, we performed fine-tuning experiments on pretrained DNNs, vision transformer [29] ViT-S/16 to compare (a) AdaLoss with η/b_t, (b) AdaGrad-Norm with η/b_t[8] (c) SGD-Constant with η/b_0, where $\eta=0.1$, (d) SGD-DecaySqrt $\eta/\sqrt{b_0^2+t}$. We fine-tune the pretrained model (ViT-S/16) on CIFAR100 (with 45k training and 5k validation images) over 10 epochs, and show test accuracy (with mean and std over three independent runs) of the best model chosen by validation data, on 10k test images. Table 1 shows that AdaLoss is more robust to varying b_0s. The results can be reproducible through the code given in Footnote 1.

Table 1: Accuracy of CIFAR100 for Vision Transformer (ViT-S/16)

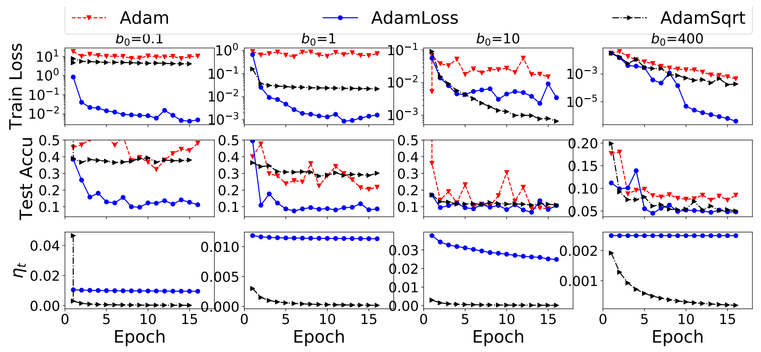
b_0	AdaLoss	AdaGrad-Norm	SGD_Constant	SGD_DecaySqrt
0.01	90.65 ± 0.09	68.34 ± 2.87	N/A	90.76 ± 0.04
0.1	90.64 ± 0.15	86.27 ± 0.43	N/A	90.50 ± 0.16
1	90.58 ± 0.12	89.33 ± 0.02	83.21 ± 0.81	90.52 ± 0.07
10	90.50 ± 0.17	90.83 ± 0.17	90.36 ± 0.11	90.44 ± 0.19
100	89.62 ± 0.12	89.99 ± 0.11	89.85 ± 0.14	89.58 ± 0.06

5 Apply AdaLoss to Adam

In this section, we consider the application of AdaLoss in the practical domain. Adam [16] has been successfully applied to many machine learning problems. However, it still requires fine-tuning the stepsize η in Algorithm 2. Although the default value is $\eta=0.001$, one might wonder if this is the optimal value. Therefore, we apply AdaLoss to make the value η robust to any initialization (see the blue part in Algorithm 2) and name it **AdamLoss**. We take two tasks to test the robustness of AdamLoss and compare it with the default Adam as well as AdamSqrt, where we literally let $\eta=1/\sqrt{b_0+t}$. Note that for simplicity, we set $\alpha=1$. More experiments are provided in the appendix for different α.

The first task is two-class (Fake/True News) text classification using one-layer LSTM (see Section D for details). The left plot in Figure 4 implies that the training loss is very robust to any initialization of the AdamLoss algorithm and subsequently achieves relatively better test accuracy. The

[8]Note that the real implementation of AdaGrad-Norm is not taking the norm of whole gradient of the network, but adapts a stepsize for each neuron or each convolution channel (see [31]).



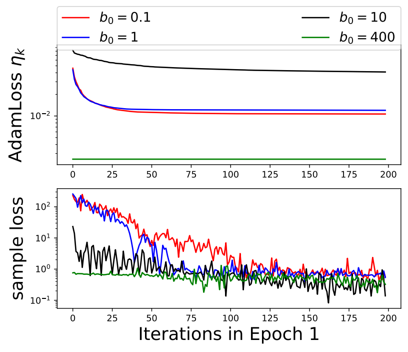
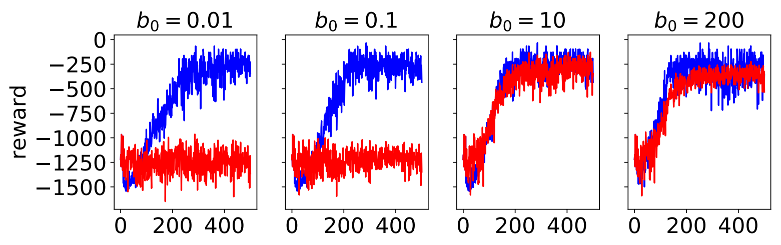


Figure 4: Text Classification—LSTM model. On the left, the top four plots are the training loss w.r.t. epoch; the middle ones are test accuracy w.r.t. epoch; the bottom ones stepsize η w.r.t. epoch. On the right, the top (bottom) plot is the stepsize η_t (the stochastic loss) w.r.t. to iterations in the 1st epoch.



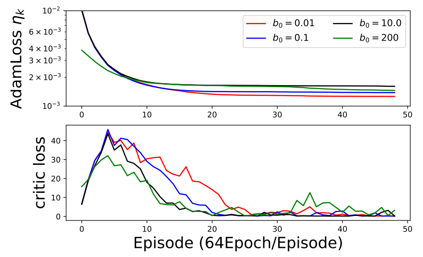


Figure 5: Inverted Pendulum Swingup with Actor-critic Algorithm. On the left, the 4 plots are the rewards (scores) w.r.t. number of frames 10000 with roll-out length 2048. The red curve is for Adam and blue is for AdamLoss. On the right, the top (bottom) figure is the stepsize η_t (the stochastic loss) w.r.t. to total episode.

right plot in Figure 4 captures the dynamics of $1/b_t$ for the first 200 iterations at the beginning of the training. We see that when $b_0 = 0.1$ (red) or $b_0 = 1$ (blue), the stochastic loss (bottom right) is very high such that after 25 iterations, it reaches $1/b_t \approx 0.01$ and then stabilizes. When $b_0 = 400$, the stochastic loss shows a decreasing trend at the beginning, which means it is around the critical threshold.

Algorithm 2: AdamLoss

```
1:  Input: $x_1$, $\beta_1 = 0.9$, $\beta_2 = 0.99$, and positive value $\eta$ $\alpha_0$ and $b_0$. Set $m_0 = v_0 = \hat{v}_0 = 0$
2:  for $t = 1, 2, 3, \dots T$ do
3:      $b_t = b_{t-1} + \alpha|f_t(x_t)|$
4:      $\eta_t = 1/\sqrt{b_t}$
5:      $g_t = \nabla f_t(x_t)$ (Get the gradient)
6:      $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$
7:      $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
8:      $\hat{m}_t = m_t/(1-\beta_1^t)$
9:      $\hat{v}_t = v_t/(1-\beta_2^t)$
10:     (Adam) $x_{t+1} = x_t - \eta \hat{m}_t/\sqrt{\hat{v}_t + \epsilon}$
11:     (AdamLoss) $x_{t+1} = x_t - \eta_t\, \hat{m}_t/\sqrt{\hat{v}_t + \epsilon}$
12: end for
```

The second task is to solve the classical control problem: inverted pendulum swing-up. One popular algorithm is the actor-critic algorithm [17], where the actor algorithm is optimized by proximal policy gradient methods [39], and the critic algorithm is optimized by function approximation methods [13]. The actor-network and critic-network are fully connected layers with different depths. We use Adam and AdamLoss to optimize the actor-critic algorithm independently for four times and average the rewards. The code source is provided in the supplementary material. The left plot of Figure 5 implies that AdaLoss is very robust to different initialization, while the standard Adam is extremely sensitive to $\eta = \frac{1}{b_0}$. Interestingly, AdamLoss does better when starting with $\eta_0 = \frac{1}{200}$. We plot the corresponding $1/b_t$ on the right-hand side in Figure 5. We see that regardless of the initialization of b_0, the final value $1/b_t \approx 0.01$ reaches a value between 0.002 and 0.001.

Overall, AdamLoss is shown numerically robust to any initialization b_0 for the two-class text classification and the inverted pendulum swing-up problems. See appendix for more experiments.

Broader Impact

Our theoretical results make a step forward to explain the linear convergence rate and zero training error using adaptive gradient methods in neural networks. Our new technique for "linear" convergence proof (Theorem 1 and Theorem 3) could be used to improve the recent sublinear convergence results of Adam-type methods [16, 8, 36, 38, 41, 9]. Based on a theoretical understanding of the complexity bound of adaptive gradient methods and the relationship between loss and gradient, we proposed a provably convergent adaptive gradient method (AdaLoss). It is computationally-efficient and could potentially be a useful optimization method for those large-scale data training. Particularly, it can be applied to the domains where tuning hyper-parameters are very expensive, thus making a potentially positive impact on society.

Acknowledgments

This work was partially done while XW, SD and RW were participating the program on "Foundation of Data Science" (Fall 2018) in the Simons Institute for the Theory of Computing at UC Berkeley. YX was partially funded by AFOSR MURI FA9550-19-1-0005 and NSF HDR-1934932. We would also thank Facebook AI Research for partial support of RW's Research. RW was supported in part by AFOSR MURI FA9550-19-1-0005, NSF DMS 1952735, and NSF 2019844.

References

[1] Allen-Zhu, Z.; Li, Y.; and Song, Z. 2018. A convergence theory for deep learning via over-parameterization. *arXiv preprint arXiv:1811.03962*.

[2] Arora, S.; Du, S. S.; Hu, W.; Li, Z.; and Wang, R. 2019. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. *arXiv preprint arXiv:1901.08584*.

[3] Barakat, A.; and Bianchi, P. 2018. Convergence of the ADAM algorithm from a Dynamical System Viewpoint. *arXiv preprint arXiv:1810.02263*.

[4] Bertsekas, D. P. 1999. *Nonlinear programming*.

[5] Bottou, L.; Curtis, F. E.; and Nocedal, J. 2018. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2): 223–311.

[6] Bubeck, S.; et al. 2015. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4): 231–357.

[7] Chen, J.; and Gu, Q. 2018. Closing the Generalization Gap of Adaptive Gradient Methods in Training Deep Neural Networks. *arXiv preprint arXiv:1806.06763*.

[8] Chen, X.; Liu, S.; Sun, R.; and Hong, M. 2019. On the Convergence of A Class of Adam-Type Algorithms for Non-Convex Optimization. In *International Conference on Learning Representations*.

[9] Défossez, A.; Bottou, L.; Bach, F.; and Usunier, N. 2020. On the Convergence of Adam and Adagrad. *arXiv preprint arXiv:2003.02395*.

[10] Du, S. S.; Lee, J. D.; Li, H.; Wang, L.; and Zhai, X. 2018. Gradient descent finds global minima of deep neural networks. *arXiv preprint arXiv:1811.03804*.

[11] Du, S. S.; Zhai, X.; Poczos, B.; and Singh, A. 2019. Gradient Descent Provably Optimizes Over-parameterized Neural Networks. In *International Conference on Learning Representations*.

[12] Duchi, J.; Hazan, E.; and Singer, Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul): 2121–2159.

[13] Fujimoto, S.; Hoof, H.; and Meger, D. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *International Conference on Machine Learning*, 1587–1596.

[14] Haykin, S.; et al. 2005. Cognitive radio: brain-empowered wireless communications. *IEEE journal on selected areas in communications*, 23(2): 201–220.

[15] Hinton, G.; Srivastava, N.; and Swersky, K. 2012. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.

[16] Kingma, D. P.; and Ba, J. L. 2014. Adam: Amethod for stochastic optimization. In *ICLR*, volume abs/1212.5701.

[17] Konda, V. R.; and Tsitsiklis, J. N. 2000. Actor-critic algorithms. In *Advances in neural information processing systems*, 1008–1014.

[18] Lei, Q.; Wu, L.; Chen, P.-Y.; Dimakis, A.; Dhillon, I.; and Witbrock, M. 2019. Discrete Adversarial Attacks and Submodular Optimization with Applications to Text Classification. *Systems and Machine Learning (SysML)*.

[19] Levy, K. 2017. Online to Offline Conversions, Universality and Adaptive Minibatch Sizes. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 30*, 1613–1622. Curran Associates, Inc.

[20] Levy, Y. K.; Yurtsever, A.; and Cevher, V. 2018. Online Adaptive Methods, Universality and Acceleration. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 31*, 6501–6510.

[21] Li, X.; and Orabona, F. 2018. On the Convergence of Stochastic Gradient Descent with Adaptive Stepsizes. *arXiv preprint arXiv:1805.08114*.

[22] Li, Y.; and Liang, Y. 2018. Learning Overparameterized Neural Networks via Stochastic Gradient Descent on Structured Data. *arXiv preprint arXiv:1808.01204*.

[23] McIntire, G. 2017. Fake news dataset.

[24] McMahan, H. B.; and Streeter, M. J. 2010. Adaptive Bound Optimization for Online Convex Optimization. In *COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010*, 244–256.

[25] Mukkamala, M. C.; and Hein, M. 2017. Variants of RMSProp and Adagrad with Logarithmic Regret Bounds. In *International Conference on Machine Learning*, 2545–2553.

[26] Nash, S. G.; and Nocedal, J. 1991. A numerical study of the limited memory BFGS method and the truncated-Newton method for large scale optimization. *SIAM Journal on Optimization*, 1(3): 358–372.

[27] Nesterov, Y. 2005. Smooth minimization of non-smooth functions. *Mathematical programming*, 103(1): 127–152.

[28] Orabona, F.; and Pál, D. 2015. Scale-free algorithms for online linear optimization. In *International Conference on Algorithmic Learning Theory*, 287–301. Springer.

[29] Steiner, A.; Kolesnikov, A.; Zhai, X.; Wightman, R.; Uszkoreit, J.; and Beyer, L. 2021. How to train your

ViT? Data, Augmentation, and Regularization in Vision Transformers. *arXiv:2106.10270*.

[30] Tan, C.; Ma, S.; Dai, Y.-H.; and Qian, Y. 2016. Barzilai-Borwein step size for stochastic gradient descent. In *Advances in Neural Information Processing Systems*, 685–693.

[31] Ward, R.; Wu, X.; and Bottou, L. 2020. AdaGrad stepsizes: Sharp convergence over nonconvex landscapes. *Journal of Machine Learning Research*, 21: 1–30.

[32] Wu, X.; Du, S. S.; and Ward, R. 2019. Global convergence of adaptive gradient methods for an over-parameterized neural network. *arXiv preprint arXiv:1902.07111*.

[33] Wu, X.; Ward, R.; and Bottou, L. 2018. WNGrad: Learn the Learning Rate in Gradient Descent. *arXiv preprint arXiv:1803.02865*.

[34] Xie, Y.; Wu, X.; and Ward, R. 2020. Linear convergence of adaptive stochastic gradient descent. In *International Conference on Artificial Intelligence and Statistics*, 1475–1485. PMLR.

[35] Zaheer, M.; Reddi, S.; Sachan, D.; Kale, S.; and Kumar, S. 2018. Adaptive Methods for Nonconvex Optimization. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 31*, 9815–9825. Curran Associates, Inc.

[36] Zaheer, M.; Reddi, S.; Sachan, D.; Kale, S.; and Kumar, S. 2018. Adaptive Methods for Nonconvex Optimization. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 31*, 9815–9825. Curran Associates, Inc.

[37] Zeiler, M. D. 2012. ADADELTA: An Adaptive Learning Rate Method. *CoRR*, abs/1212.5701.

[38] Zhou, D.; Tang, Y.; Yang, Z.; Cao, Y.; and Gu, Q. 2018. On the convergence of adaptive gradient methods for nonconvex optimization. *arXiv preprint arXiv:1808.05671*.

[39] Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 8697–8710.

[40] Zou, D.; Cao, Y.; Zhou, D.; and Gu, Q. 2018. Stochastic gradient descent optimizes over-parameterized deep ReLU networks. *arXiv preprint arXiv:1811.08888*.

[41] Zou, F.; Shen, L.; Jie, Z.; Zhang, W.; and Liu, W. 2018. A Sufficient Condition for Convergences of Adam and RMSProp. *arXiv preprint arXiv:1811.09358*.