WNGrad: Learn the Learning Rate in Gradient Descent

Xiaoxia Wu [* 1 2] Rachel Ward [* 1 2] Léon Bottou [2]

Abstract

Adjusting the learning rate schedule in stochastic gradient methods is an important unresolved problem which requires tuning in practice. If certain parameters of the loss function such as smoothness or strong convexity constants are known, theoretical learning rate schedules can be applied. However, in practice, such parameters are not known, and the loss function of interest is not convex in any case. The recently proposed *batch normalization* reparametrization is widely adopted in most neural network architectures today because, among other advantages, it is robust to the choice of Lipschitz constant of the gradient in loss function, allowing one to set a large learning rate without worry. Inspired by batch normalization, we propose a general nonlinear update rule for the learning rate in batch and stochastic gradient descent so that the learning rate can be initialized at a high value, and is subsequently decreased according to gradient observations along the way. The proposed method is shown to achieve robustness to the relationship between the learning rate and the Lipschitz constant, and near-optimal convergence rates in both the batch and stochastic settings ($O(1/T)$ for smooth loss in the batch setting, and $O(1/\sqrt{T})$ for convex loss in the stochastic setting). We also show through numerical evidence that such robustness of the proposed method extends to highly nonconvex and possibly non-smooth loss function in deep learning problems. Our analysis establishes some first theoretical understanding into the observed robustness for batch normalization and weight normalization.

1. Introduction

Recall the standard set-up for gradient descent: we consider the general problem of minimizing a "loss function" $f : \mathbb{R}^d \to \mathbb{R}$,

$$\min_x \quad f(x) \tag{1}$$

and given access only to first-order/gradient evaluations of f, we iteratively move in the direction of the negative gradient until convergence: $x_{j+1} \leftarrow x_j - \eta_j \nabla f(x_j)$. Gradient descent enjoys nice convergence guarantees if the learning rate $\eta_j = \eta$ is tuned just right according to the scale of the smoothness of the gradient function $\nabla f(x)$; on the other hand, if the learning rate is chosen slightly larger than the optimal value, gradient descent with constant learning rate can oscillate or even diverge. Thus, in practice, one instead uses iteration-dependent learning rate η_j, chosen via line search methods (Dimitri, 1999; Wright and Nocedal, 2006). Line search methods work well in the "batch" set-up where the gradients $\nabla f(x_j)$ are observed *exactly*, but notoriously become less effective in the *stochastic* setting, where only noisy gradient evaluations are given. Recall the standard setting for stochastic gradient descent: Instead of observing a full gradient $\nabla f(x_k)$ at iteration k, we observe a stochastic gradient g_k, or a random vector satisfying $\mathbb{E}(g_k) = \nabla f(x_k)$ and having bounded variance $\mathbb{E}\|g_k\|^2 \le G^2$. Stochastic gradient descent is the optimization algorithm of choice in deep learning problems, and, more generally, in many large-scale optimization problems where the objective function f can be expressed as a sum of a number component functions f_i, of which only have access to a subset (the so-called "training data").

In the stochastic setting, the issue of how to choose the learning rate is less resolved. There are different guidelines for setting the learning "schedule" $\eta_1, \eta_2, \ldots$, each guideline having its own justification in the form of a convergence result given a set of structural assumptions on the loss function f. The classical Robbins/Monro theory (Robbins and Monro,

---

[*]Equal contribution [1]Department of Mathematics, University of Texas at Austin, Austin, TX [2]Facebook AI Research, New York, New York, USA. Correspondence to: Xiaoxia Wu <xwu@math.utexas.edu>.

1951) says that if the learning rate is chosen such that

$$\sum_{k=1}^{\infty} \eta_k = \infty \quad \text{and} \quad \sum_{k=1}^{\infty} \eta_k^2 < \infty, \tag{2}$$

and if the loss function is sufficiently smooth, then $\lim_{k\to\infty} \mathbb{E}[\|\nabla f(x_k)\|^2] = 0$ ((Bottou et al., 2016), Corollary 4.12). If the loss function is moreover strongly convex, the stochastic gradient update $x_{k+1} \leftarrow x_k - \eta_k g_k$ will converge in expectation to the minimizer .

If the loss function is convex but not necessarily smooth, then setting $\eta_k = c/\sqrt{k}$ results in a convergence guarantee of the form $\mathbb{E}[f(x_k) - f^*] \leq O(\log(k)/\sqrt{k})$, with an optimal constant if c is chosen properly to depend on the stochastic variance G^2 ((Shamir and Zhang, 2013), Theorem 2). If the loss function is μ-*strongly* convex and has L-Lipschitz smooth gradient, then setting $\eta_k = c/k$ where c is sufficiently small compared to $\frac{\mu}{L}$ gives $\mathbb{E}[f(x_k) - f^*] \leq O(1/k)$ ((Bottou et al., 2016), Theorem 4.7).

If, moreover, the loss function can be expressed as the average of a number of component functions f_i, each of which is itself convex, and if the noisy gradient direction at each iteration is actually the direction of the exact gradient of one of the component functions chosen i.i.d. uniformly from the universe of component functions, and if a bound on the "consistency" parameter $\sigma^2 = \mathbb{E}_i[\|f_i(x^*)\|^2]$ is known, then one may take a constant learning rate $\eta_k = \eta$ chosen in just the right way with respect to μ, L, and σ^2 to achieve *exponential* convergence, in expectation, up to a radius around the optimal solution (Bach and Moulines, 2011; Needell et al., 2014).

Thus, in the stochastic setting, there is no clear "best choice" for the learning rate. In many deep learning problems, where the underlying loss function is highly non-convex, one often tests several different learning rate schedules of the form

$$\eta_k = \frac{\eta_0}{1 + k/\tau}, \text{ or } \eta_k = \frac{\eta_0}{1 + \sqrt{k}/\tau}, \text{ or } \eta_k = \frac{D}{kG};$$

where D is the maximal diameter of the feasible set, and G is the norm of the current gradient or an average of recent gradients; the schedule which works best on the problem at hand is then chosen. Another popular and effective choice is to start with a constant learning rate η_0 which gives good empirical convergence results or start with a small one followed by a warmup scheme (Goyal et al., 2017), maintain this constant learning rate for a fixed number of epochs over the training data, then decrease the learning rate $\eta_1 \leftarrow 0.1\eta_0$, and repeat this process until convergence .

1.1. Adaptive Learning Rate Rules

In the stochastic setting, it can be advantageous to set different learning rates for different component functions f_i (or for different coordinates), with larger learning rates for components with smaller gradients, and smaller learning rates for components with larger gradients, to balance their respective influences. This heuristic is theoretically justified in some cases. The family of adaptive gradient (AdaGrad) algorithms (Duchi et al., 2011) dynamically update each coordinate learning rate by the reciprocal of the root-mean-square of the elements of the gradients for that coordinate which have been observed so far. AdaGrad has rigorous theoretical backing: it provably achieves the optimal $\mathbb{E}[f(x_k) - f^*] \leq O(1/\sqrt{k})$ regret guarantee in the convex setting, with a better constant compared to plain stochastic gradient descent depending on the geometry of the problem. Despite originally being designed in the convex setting, AdaGrad has proven to be very useful beyond the convex set-up – in particular, it improves convergence performance over standard stochastic gradient descent in settings where data is sparse and sparse parameters are more informative; such examples abound in natural language processing. Several subsequent modifications to AdaGrad have been proposed to combat this accumulation including Adadelta (Zeiler, 2012), RMSprop (Srivastava and Swersky), Adam (Kingma and Ba, 2014) and AdaBatch (Alexandre and Francis, 2017); however, these algorithms (except AdaBatch) come with no guarantees of convergence. These adaptive subgradient methods cannot be applied as a general panacea for the learning rate problem, however as they result in biased gradient updates which change the underlying optimization problem. The recent paper (Wilson et al., 2017) provides evidence that while these methods do speed up training time in neural network applications, they nevertheless result in worse generalization error compared to simple methods such as plain stochastic gradient descent with a single learning rate.

Another line of work on adaptive learning rates (Needell et al., 2014; Zhao and Zhang, 2015) consider importance sampling in stochastic gradient descent in the setting where the loss function can be expressed as a sum of component functions, and provide precise ways for setting different constant learning rates for different component functions based on their

Lipschitz constants; if the sampling distribution over the parameters is weighted so that parameters with smaller Lipschitz constants are sampled less frequently, then this reparametrization affords a faster convergence rate, depending on the *average* Lipschitz constant between all parameters, rather than the *largest* Lipschitz constant between them. Of course, in practice, the Lipschitz constants are not known in advance, and must be learned along the way.

This begs the question: if we take a step back to the batch/non-stochastic gradient descent setting, is it possible to learn even a *single* Lipschitz constant, corresponding to the gradient function ∇f, so that we can match the convergence rate of gradient descent with optimized constant learning rate which requires knowledge of the Lipschitz constant beforehand? To our knowledge, this question has not been addressed until now.

1.2. Weight Normalization

To answer this question, we turn to simple reparametrizations of weight vectors in neural networks which have been proposed in recent years and have already gained widespread adaptations in practice due to their effectiveness in accelerating training times without compromising generalization performance, while simultaneously being robust to the tuning of learning rates. The celebrated batch normalization (Ioffe and Szegedy, 2015) accomplishes these objectives by normalizing the means and variances of minibatches in a particular way which reduces the dependence of gradients on the scale of the parameters or their initial values, allowing the use of much higher learning rates without the risk of divergence. Inspired by batch normalization, the *weight normalization* algorithm (Salimans and Kingma, 2016) was introduced as an even simpler reparametrization, also effective in making the resulting stochastic gradient descent more robust to specified learning rates and initialization (Tygert et al., 2015). The weight normalization algorithm, roughly speaking, reparametrizes the loss function in polar coordinates, and runs (stochastic) gradient descent with respect to polar coordinates: If the loss function is $f(x)$ where x is a d-dimensional vector, then weight normalization considers instead $x = \frac{r}{\|v\|}v$, where v is a d-dimensional vector, r is a scalar, and $\|v\|$ is the Euclidean norm of v. The analog of the weight normalization algorithm in the batch gradient setting would simply be gradient descent in polar coordinates as follows:

$$
\begin{aligned}
v_{k+1} &= v_k - \eta \nabla_v f\left(\frac{r_k}{\|v_k\|}v_k\right) \\
&= v_k - \eta \frac{r_k}{\|v_k\|} P_{v_k^\perp}\left(\nabla f\left(\frac{r_k}{\|v_k\|}v_k\right)\right); \\
r_{k+1} &= r_k - \eta \nabla_r f\left(\frac{r_k}{\|v_k\|}v_k\right) \\
&= r_k - \eta \left\langle \nabla f\left(\frac{r_k}{\|v_k\|}v_k\right), \frac{v_k}{\|v_k\|}\right\rangle
\end{aligned}
\tag{3}
$$

where $P_{v^\perp}(u)$ denotes the orthogonal projection of u onto the subspace of co-dimension orthogonal to v. One important feature of note is that, since the gradient of f with respect to v is orthogonal to the current direction v, the norm $\|v_k\|$ grows monotonically with the update, thus effectively producing a dynamically-updated decay in the *effective* learning rate $\frac{r\eta}{\|v\|^2}$. More precisely, considering weight normalization in the batch setting, restricted to the unit sphere (fixing $r_k = 1$), the gradient update reduces to

$$
\begin{aligned}
\frac{v_{k+1}}{\|v_k\|} &= \frac{v_k}{\|v_k\|} - \frac{\eta}{\|v_k\|^2} P_{v_k^\perp}\left(\nabla f\left(\frac{v_k}{\|v_k\|}\right)\right); \\
\|v_{k+1}\|^2 &= \|v_k\|^2 + \frac{\eta^2}{\|v_k\|^2}\left\|P_{v_k^\perp}\left(\nabla f\left(\frac{v_k}{\|v_k\|}\right)\right)\right\|^2.
\end{aligned}
\tag{4}
$$

1.3. Our contributions

Weight normalization (and, to an even larger extent, batch normalization) has proven in practice to be very robust to the choice of the scale of Lipschitz constant η. Inspired by this, and in a first attempt at theoretical understanding of such normalization, we are inspired to consider the following method for updating the learning rate in batch and stochastic gradient descent more generally: starting from $x_1 \in \mathbb{R}^d$ and $b_1 > 0$, repeat until convergence

$$
\begin{aligned}
x_{k+1} &= x_k - \frac{1}{b_k}\nabla f(x_k); \\
b_{k+1} &= b_k + \frac{1}{b_k}\|\nabla f(x_k)\|^2.
\end{aligned}
\tag{5}
$$

As a nod to its inspiration, weight normalization, we call this algorithm WNGrad, but note that the update can also be interpreted as a close variant of AdaGrad with the dynamic update applied to a single learning rate; indeed, WNGrad b-update satisfies

$$b_{k+1}^2 = b_k^2 + 2\|\nabla f(x_k)\|^2 + \frac{1}{b_k^2}\|\nabla f(x_k)\|^4$$
$$= b_k^2 + 2\|\nabla f(x_k)\|^2 + O(\|\nabla f(x_k)\|^4)$$

which matches the coordinate-wise update rule in AdaGrad if ∇f is one dimension. Nevertheless, WNGrad update (5) offers some insight and advantages over the family or modifications/improvements of AdaGrad update – first, it gives a precise correspondence between the accumulated gradient and current gradient in the update of the b_k. Additionally, it does not require any square root computations, thus making the update more efficient.

In this paper, we provide some basic theoretical guarantees about WNGrad update. Surprisingly, we are able to provide guarantees *for the same learning rate update rule* in both the batch and stochastic settings.

In the batch gradient descent setting, we show that WNGrad will converge to a weight vector x_T satisfying $\|\nabla f(x_T)\|^2 \le \epsilon$ in at most $T = O(\frac{(f(x_1)-f^*+L)^2}{\epsilon})$ iterations, if f has L-Lipschitz smooth gradient. The proof involves showing that if b_k grows up to the critical level $b_k \ge L$, it *automatically stabilizes*, satisfying $b_k \le CL$ for all time[1]. This should be compared to the standard gradient descent convergence rate using constant learning rate η, which in the ideal case $\eta = 1/L$ achieves $O(\frac{L}{\epsilon})$ convergence rate, but which *is not guaranteed to converge at all* if the learning rate is even slightly too big, $\eta \ge 2/L$. Thus, WNGrad is a provably robust variant to gradient descent which is provably robust to the scale of Lipschitz constant, when parameters like the Lipschitz smoothness are not known in advance.

On the other hand, in the stochastic setting, the b_k update in WNGrad has dramatically different behavior, growing like $O(\frac{\sqrt{k}}{G})$, where G is a bound on the variance of the stochastic gradients. As a result, in the stochastic setting, we also show that WNGrad, achieves the optimal $O(1/\sqrt{T})$ rate of convergence for convex loss functions, and moreover settles in expectation on the "correct" constant, $b_k \sim \frac{\sqrt{k}}{G}$. Thus, WNGrad also works robustly in the stochastic setting, and finds a good learning rate.

We supplement all of our theorems with numerical experiments, which show that WNGrad competes favorably to plain stochastic gradient descent in terms of robustness to the Lipschitz constant of the loss function, speed of convergence, and generalization error, in training neural networks on two standard data sets.

2. WNGrad for Batch Gradient Descent

Consider a smooth function $f : \mathbb{R}^d \to \mathbb{R}$ with L-Lipschitz continuous gradient (denoted $f \in C_L^1$): for any $x, y \in \mathbb{R}^d$,

$$\|\nabla f(x) - \nabla f(y)\| \le L\|x - y\|$$

and the optimization problem

$$\min_x \quad f(x).$$

With knowledge of the Lipschitz constant L, the standard gradient descent update with constant learning rate iterates, starting at $x_1 \in \mathbb{R}^d$,

$$x_j \leftarrow x_{j-1} - \eta \nabla f(x_{j-1}). \tag{6}$$

The following convergence result is classical ((Nesterov, 1998), (1.2.13)).

Lemma 2.1 *Suppose that $f \in C_L^1$ and that $f^* > -\infty$. Consider gradient descent with constant learning rate $\eta > 0$.*
If $\eta = \frac{\delta}{L}$ and $\delta \le 1$, then

$$\min_{j=1:T} \|\nabla f(x_j)\|^2 \le \epsilon$$

after at most a number of steps

$$T = \frac{2L(f(x_1) - f^*)}{\delta \epsilon};$$

---

[1]C is a constant factor

On the other hand, gradient descent can oscillate or diverge once $\eta \geq \frac{2}{L}$.

Note that this result requires the knowledge of Lipschitz constant L or an upper bound estimate. Even if such a bound is known, the algorithm is quite conservative; the Lipschitz constant represents the *worst case* oscillation of the function ∇f over all points x, y in the domain; the *local* behavior of gradient might be much more regular, indicating that a larger learning rate (and hence, faster convergence rate) might be permissible. In any case, it is beneficial to consider a modified gradient descent algorithm which, starting from a large initial learning rate, decreases the learning rate according to gradient information received so far, and stabilizes at at a rate depends on the local smoothness behavior and so no smaller than $1/L$.

We consider the following modified gradient descent scheme:

Algorithm 1 WNGrad – Batch Setting

Input: Tolerance $\epsilon > 0$
Initialize $x_1 \in \mathbb{R}^d, b_1 > 0, j \leftarrow 1$
repeat
    $j \leftarrow j + 1$
    $x_j \leftarrow x_{j-1} - \frac{1}{b_{j-1}} \nabla f(x_{j-1})$
    $b_j \leftarrow b_{j-1} + \frac{\|\nabla f(x_j)\|^2}{b_{j-1}}$
until $\|\nabla f(x_j)\|^2 \leq \epsilon$

Remark 2.2 *Initializing b_1 and scale invariance. Ideally, one could initialize b_1 in WNGrad by sampling $R \geq 2$ points $u_1, u_2, \ldots, u_R$ close to the initialization x_1, and $\nabla f(u_1), \ldots, \nabla f(u_R)$, and take*

$$b_1 = \max_{j \neq k} \frac{\|\nabla f(u_j) - \nabla f(u_k)\|}{\|u_j - u_k\|} \leq L.$$

If this is not possible, it is also reasonable to consider an initialization $b_1 = C\|\nabla f(x_1)\|$ with a constant $C \geq 0$. With either choice, one observes that the resulting WNGrad algorithm is invariant to the scale of f: if f is replaced by λf, then the sequence of iterates $x_1, x_2, \ldots$ remains unchanged.

We show that the WNGrad algorithm has the following properties:

- After a reasonable number of initial iterations, either $\|\nabla f(x_k)\|^2 \leq \epsilon$ or $b_k \geq L$

- If at some point $b_k \geq L$, then the learning rate stabilizes: $b_j \leq CL$ for all $j \geq k$.

As a consequence, we have the following convergence result.

Theorem 2.3 (Global convergence for smooth loss function) *Consider the WNGrad algorithm. Set $b_1 \geq \|\nabla f(x_1)\|$. Suppose that $f \in C_L^1$, x^* is the point satisfying $\nabla f(x^*) = 0$. and that $f^* > -\infty$.*

Then we have the guarantee

$$\min_{k=1:T} \|\nabla f(x_k)\|^2 \leq \epsilon$$

after   ***Case 1*** $T = \frac{2(f(x_1) - f^*)(b_1 + 8(f(x_1) - f^*))}{\epsilon}$ *steps if $b_1 \geq L$, and*

                     Case 2 $T = 1 + \frac{L^2(1-\delta)}{\epsilon} + \frac{16\left((f(x_1) - f^*) + (\frac{3}{16} + \frac{5}{8\delta})L\right)^2}{\epsilon}$ *steps if $b_1 = \delta L < L$, $\quad \delta \in (0, 1]$.*

Comparing the convergence rate of batch gradient descent in Theorem 2.3 and the classical convergence result in Lemma 2.1, we see that WNGrad adjusts the learning rate automatically with decreasing learning rate $1/b_j$ based on the gradient information received so far, and without knowledge of the constant L, and still achieves linear convergence at nearly the same rate as gradient descent in Lemma 2.1 with constant learning rate $\eta \leq \frac{1}{L}$.

We will use the following lemmas to prove Theorem 2.3. For more details, see Appendix A.1.

Lemma 2.4 *Fix $\epsilon \in (0,1]$ and $L > 0$. Consider the sequence*

$$b_1 > 0; \qquad b_{j+1} = b_j + \frac{\|\nabla f(x_{j+1})\|^2}{b_j}$$

after $N = \max\left\{1, \lceil \frac{L(L-b_1)}{\eta\epsilon} \rceil + 1\right\}$ iterations, either $\min_{k=1:N} \|\nabla f(x_k)\|^2 \leq \epsilon$, or $b_N > L$.

Lemma 2.5 *Suppose that $f \in C_L^1$, $f^* > -\infty$ and $b_1 \geq \|\nabla f(x_1)\|$. Denote by k_0 the first index such that $b_{k_0} > L$. Then for all $k \geq k_0$,*

$$b_k \leq b_{k_0} + 8(f(x_{k_0}) - f^*)$$

and moreover, if $k_0 > 1$,

$$f(x_{k_0}) \leq f(x_1) + \frac{L^2}{2b_1}.$$

Lemma 2.5 guarantees that the learning rate stabilizes once it reaches the (unknown) Lipschitz constant, up to an additive term. To be complete, we can also bound b_{k_0} as a function of L, then arrive at the main result of this section.

Lemma 2.6 *Suppose that $f \in C_L^1$ and that $f^* > -\infty$. Denote by k_0 the first index such that $b_{k_0} > L$. Then*

$$b_{k_0} \leq 3L + \frac{2L^2}{b_1}.$$

3. WNGrad for Stochastic Gradient Descent

We now shift from the setting of batch gradient descent to stochastic gradient descent. The update rule to the learning rate in WNGrad extends without modification to this setting, but now that the gradient norms do not converge to zero but rather remain noisy, the WNGrad learning rate $\frac{1}{b_k}$ does not converge to a fixed size, but rather settles eventually on the rate of $\frac{G}{\sqrt{k}}$, where G is a bound on the variance of the stochastic gradients. In order to tackle this issue and derive a convergence rate, we assume for the analysis that the loss function is convex but not necessarily smooth.

Algorithm 2 WNGrad – Stochastic Setting

> **Input:** Tolerance $\epsilon > 0$
> Initialize $x_1 \in \mathbb{R}^d$, $b_1 > 0$, $j \leftarrow 1$
> **repeat**
> $j \leftarrow j + 1$
> $x_j \leftarrow x_{j-1} - \frac{1}{b_{j-1}} g_{j-1}$
> $b_j \leftarrow b_{j-1} + \frac{\|g_j\|^2}{b_{j-1}}$
> **until** $f(x_j) \leq \epsilon$

Consider the general optimization problem

$$\min_x \quad f(x)$$

from stochastic gradient information. Instead of observing full gradients $\nabla f(x_k)$, we observe stochastic gradients $g_k \in \mathbb{R}^d$ satisfying $\mathbb{E}(g_k) = \nabla f(x_k)$. Let $\overline{x}_k = \frac{1}{k} \sum_{i=1}^k x_i$.

Theorem 3.1 *Consider WNGrad algorithm. Suppose $f(x)$ is convex. Suppose, that, independent of x_k,*

$$\mathbb{E}\|g_k\|^2 \leq G^2$$

and that for all k,

$$\|g_k\| \geq \gamma$$

and $\mathbb{E}\|x_k - x^*\|^2 \leq D^2$. Then, with initialization $b_1 \geq \|g_1\|$,

$$f(\overline{x}_k) - f^* \leq \frac{G^2(D^2 + 2)}{\gamma\sqrt{k}} + \frac{b_1\|x_1 - x^*\|^2}{2k}.$$

Remark 3.2 *Under the same assumptions, excluding the assumption that $\gamma^2 \leq \|g_k\|^2$, one obtains the same convergence rate using decreasing learning rate $\eta_k = \frac{c}{\sqrt{k}}$ for some constant c.*

We will use the following lemma, which is easily proved by induction.

Lemma 3.3 *Consider a positive constant $a > 0$ and a sequence of positive numbers $t_1, t_2, \ldots$ and for each k,*

$$t_k + \frac{a}{t_k} \leq t_{k+1}.$$

Then,

$$\sqrt{2ak} \leq t_k.$$

Proof of Theorem 3.1: First, note that under the stated assumptions, b_k satisfies Lemma 3.3 for $a = \gamma^2$. Thus, with probability 1,

$$b_k \geq \gamma\sqrt{k}.$$

Now,

$$\|x_{k+1} - x^*\|^2 = \|x_k - x^*\|^2 + \frac{1}{b_k^2}\|g_k\|^2 - 2\frac{1}{b_k}\langle x_k - x^*, g_k\rangle,$$

so

$$2\langle x_k - x^*, g_k\rangle = b_k\|x_k - x^*\|^2 - b_k\|x_{k+1} - x^*\|^2 + \frac{1}{b_k}\|g_k\|^2.$$

Thus,

$$2\sum_{\ell=1}^{k}\langle x_\ell - x^*, g_\ell\rangle \leq 2\sum_{\ell=1}^{k}\langle x_\ell - x^*, g_\ell\rangle + b_k\|x_{k+1} - x^*\|^2$$

$$= \sum_{\ell=2}^{k}(b_\ell - b_{\ell-1})\|x_\ell - x^*\|^2 + b_1\|x_1 - x^*\|^2 + \sum_{\ell=1}^{k}\frac{1}{b_\ell}\|g_\ell\|^2$$

$$= \sum_{\ell=2}^{k}\frac{\|g_\ell\|^2}{b_{\ell-1}}\|x_\ell - x^*\|^2 + b_1\|x_1 - x^*\|^2 + \sum_{\ell=1}^{k}\frac{1}{b_\ell}\|g_\ell\|^2$$

$$\leq \sum_{\ell=1}^{k-1}\frac{\|g_{\ell+1}\|^2}{\sqrt{\ell}\gamma}\|x_{\ell+1} - x^*\|^2 + b_1\|x_1 - x^*\|^2 + \sum_{\ell=1}^{k}\frac{1}{\sqrt{\ell}\gamma}\|g_\ell\|^2.$$

Now, since $\langle x_k - x^*, \nabla f(x_k)\rangle = \mathbb{E}\langle x_k - x^*, g_k\rangle$ and since $\mathbb{E}\|g_\ell\|^2 \leq G^2$, conditioned on $g_{\ell-1}, \ldots, g_1$, we apply the law of iterated expectation to obtain

$$2\sum_{\ell=1}^{k}\langle x_\ell - x^*, g_\ell\rangle \leq \frac{1}{\gamma}\left(\sum_{\ell=1}^{k-1}\frac{1}{\sqrt{\ell}}G^2\mathbb{E}(\|x_{\ell+1} - x^*\|^2)\right) + b_1\|x_1 - x^*\|^2 + \frac{1}{\gamma}\sum_{\ell=1}^{k}\frac{1}{\sqrt{\ell}}G^2$$

$$\leq \frac{2G^2(D^2 + 1)}{\gamma}\sqrt{k-1} + b_1\|x_1 - x^*\|^2 + \frac{2}{\gamma}G^2\sqrt{k}$$

where in the final inequality, we use that $\sum_{\ell=1}^{k}\frac{1}{\sqrt{\ell}} \leq 2\sqrt{k}$.

From Jensens inequality, and recalling that by convexity $f(x_k) - f^* \leq \langle x_k - x^*, \nabla f(x_k) \rangle = \mathbb{E}\langle x_k - x^*, g_k \rangle$, we conclude

$$f(\overline{x}_k) - f^* \leq \frac{1}{k} \sum_{\ell=1}^{k} (f(x_\ell) - f^*)$$

$$\leq \frac{G^2(D^2 + 1)\sqrt{k-1} + \gamma(b_1/2)\|x_1 - x^*\|^2 + G^2\sqrt{k}}{\gamma k}$$

$$\leq \frac{G^2(D^2 + 2)\sqrt{k} + \gamma(b_1/2)\|x_1 - x^*\|^2}{\gamma k}.$$

4. Numerical Experiments

With guaranteed convergence of WNGrad in both batch and stochastic settings under appropriate conditions[2], we perform experiments in this section to show that WNGrad exhibits the same robustness for highly non-convex loss functions associated to deep learning problems.

Consider a loss function f whose gradient has Lipschitz constant L. Then, the gradient of the rescaled loss function λf has Lipschitz constant λL. If we were to also rescale b_1 to λb_1, then the dynamics $x_j \leftarrow x_{j-1}$ would remain unchanged due to scale invariance. If instead we fix $b_1 = 1$ while letting λ vary, we can test the robustness of WNGrad to different Lipschitz constants, and compare its robustness to stochastic gradient descent (SGD, Algorithm 5 in Appendix). To be precise, we consider the following variant of WNGrad, Algorithm 3, and explore its performance as we vary λ. Note that λ in this algorithm is analogous to the constant learning rate η in weight normalization and batch normalization as discussed in (4).

Algorithm 3 WNGrad - Scaled Loss Function

Input: Tolerance $\epsilon > 0$
Initialize $x_1 \in \mathbb{R}^d, b_1 \leftarrow 1, j \leftarrow 1$
repeat
    $j \leftarrow j + 1$
    $b_j \leftarrow b_{j-1} + \frac{\lambda^2 \|g_{j-1}\|^2}{b_{j-1}}$
    $x_j \leftarrow x_{j-1} - \frac{\lambda g_{j-1}}{b_j}$
until $\|\nabla f(x_j)\|^2 \leq \epsilon$

WNGrad is mainly tested on two data sets: MNIST (LeCun et al., 1998a) and CIFAR-10 (Krizhevsky, 2009). Table 1 is the summary. We use batch size 100 for both MNIST and CIFAR-10. The experiments are done in PyTorch and parameters are by default if no specification is provided. The data sets are preprocessed with normalization using mean and standard deviation of the entire train samples. Details in implementing WNGrad in a neural network are explained in Appendix A.3.

We first test a wide range of the scale of the loss function[3] with two fully connected layers (without bias term) on MNIST (input dimension is 784) in a very simple setting excluding other factors that come into effect, such as regularization (weight decay), dropout, momentum, batch normalization, etc. In addition, we repeat 5 times for each experiment in order to avoid the initialization effect since random initialization of weight vectors is used in our experiments.

The outcome of the experiments shown in Figure 1 verifies that WNGrad is very robust to the Lipschitz constant, while SGD is much more sensitive. *This shows that the learning rate can be initialized at a high value if we consider λ to be the learning rate.* When picking $\lambda = 0.562$ and $\lambda = 0.056$, we have the train/test loss with respect to epoch shown in blue and dark-red curves respectively. With larger scale of Lipschitz constant ($\lambda = 0.562$), WNGrad does much better than SGD in both training and test loss. It is interesting to note that even with smaller scale of the Lipschitz constant $\lambda = 0.056$, even thought SGD obtains the smaller training loss but does worse in generalization. On the contrary, WNGrad gives better generalization (smaller test loss) despite of the larger train loss. Thus, WNGrad to some extend is not only robust to the scale of Lipschitz constant but also generalizes well – we aim to study this property of WNGrad in future work.

---

[2]We assume non-convex smooth loss function in batch setting and convex not necessarily smooth in stochastic setting
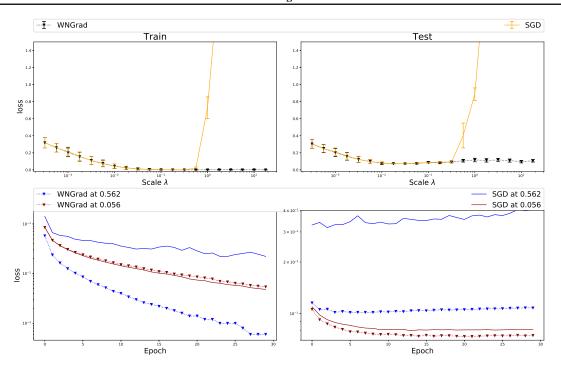[3]$\lambda \in \{10^{-0.25j+1.25}, j \in \{0, 1, 2, \cdots, 19\}\}$

Figure 1. Two fully connected layers on MNIST. The first row are plots of mean Train/Test loss of 5 repeated experiments with respect to the scale of Lipschitz constant at epoch 30 and the second row are plots of mean loss with respect to epoch at $\lambda = 0.562$ (blue curves) and $\lambda = 0.056$ (dark-red curves). The error bar of the plots in the first row means one standard deviation of five repeated experiments and no error bars shows in the second row for neatness. Better read on screen.
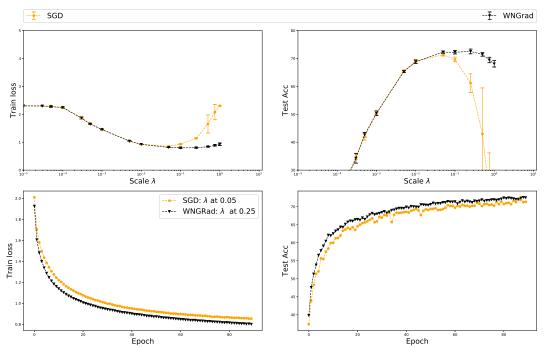

Figure 2. The simple convolutional network on CIFAR10. Top left (right) is the average training loss (test accuracy) of 5 repeated experiments with respect to the scale of Lipschitz constant at epoch 90. Bottom left (right) is the plot of mean train loss (test accuracy) with respect to epoch at the best λ found at this epoch. The error bar of the plots in the first row means one standard deviation of five repeated experiments and no error bars shown in the second row for neatness. Better read on screen.

Now we continue to compare the methods on a larger dataset, CIFAR10, with a wide range of scale λ [4] from 0.0001 to 1. We apply a simple convolution neural network (see Table 2 for details) with weight decay 10^{-4}, of which the result shown in Figure 2. In comparison with SGD, WNGrad is very robust to the scale λ – it performs better at $\lambda \in [0.01, 1]$ and does as well as SGD when λ smaller than 0.01. When at the best λ for each algorithm ($\lambda = 0.25$ WNGrad and $\lambda = 0.05$ SGD), WNGrad outperforms in training and testing along the way.

A common practice to train deep and recurrent neural neural networks is to add momentum to stochastic gradient descent (Sutskever et al., 2013). Recent adaptive moment estimation (Adam) (Kingma and Ba, 2014) seems to improve performance of models on a number of datasets. However, these methods are considerably sensitive to the scale of Lipschitz constant and require careful tuning in order to obtain the best result. Here we incorporate our algorithms with momentum (WNGrad-Momentum) and adapt "Adam" way (WN-Adam, Algorithm 4) in the hope to improve the robustness to the relationship between the learning rate and the Lipschitz constant. We use ResNet-18 training on CIFAR10 in Figure 3. Because of the batch normalization designed in ResNet-18, we widen the range of λ up to 10. As we can see, WN-Adam (green curve) and WNGrad-Momentum (black curve) do seem to be more robust compared to Adam (red) and SGD-Momentum (orange). Particularly, WN-Adam is very robust even at $\lambda = 10$ and still does fairly well in generalization.

Algorithm 4 WN-Adam

Input: Tolerance $\epsilon > 0$
Initialize $x_1 \in \mathbb{R}^d, \hat{g}_1 \leftarrow 0, b_1 \leftarrow 1, j \leftarrow 1$
repeat
$\quad j \leftarrow j + 1$
$\quad \hat{g}_j \leftarrow \beta_1 \hat{g}_{j-1} + (1 - \beta_1) g_{j-1}, \beta_1 = 0.9$
$\quad b_j \leftarrow b_{j-1} + \lambda^2 \frac{\|g_{j-1}\|^2}{b_{j-1}}$
$\quad x_j \leftarrow x_{j-1} - \frac{\lambda}{b_j} \frac{\hat{g}_j}{1 - \beta_1^{j-1}}$
until $f(x_j) \leq \epsilon$

Figure 3. ResNet-18 on CIFAR10. Top plots are the snapshots of training at the 60th epoch. Reading instruction, see Figure 2.

---

[4] $\lambda \in \{1, \frac{3}{4}, \frac{1}{2}, \frac{1}{4}, \frac{1}{10}, \frac{1}{20}, \frac{1}{100}, \frac{1}{200}, \frac{1}{1000}, \frac{1}{2000}, \frac{1}{10000}\}$.

5. Conclusion

We propose WNGrad, an method for dynamically updating the learning rate $1/b_k$ according to gradients received so far, which works in both batch and stochastic gradient methods and converges.

In the batch gradient descent setting, we show that WNGrad converges to a weight vector w_T satisfying $\|\nabla f(w_T)\|^2 \leq \epsilon$ in at most $T = O(\frac{L+L^2}{\epsilon})$ iterations, if f has L-Lipschitz smooth gradient. This nearly matches the convergence rate for standard gradient descent with fixed learning rate $1/L$, but WNGrad does not need to know L in advance.

In the stochastic setting, the b_k update in WNGrad has different behavior, growing like $O(\frac{\sqrt{k}}{G})$, where G is a bound on the variance of the stochastic gradients. As a result, in the stochastic setting, we also show that WNGrad achieves the optimal $O(1/\sqrt{T})$ rate of convergence for convex loss functions, and moreover settles in expectation on the "correct" rate, $b_k \sim \frac{\sqrt{k}}{G}$. Thus, WNGrad works robustly in the stochastic setting, and finds a good learning rate.

In numerical experiments, WNGrad competes favorably to plain stochastic gradient descent in terms of robustness to the relationship between the learning rate and the Lipschitz constant and generalization error in training neural networks on two standard data sets. And such robustness extends further to the algorithm that incorporates momentum (WN-Adam and WNGrad-Momentum).

Acknowledgments

References

D. Alexandre and B. Francis. Adabatch: Efficient gradient aggregation rules for sequential and parallel stochastic gradient methods. *arXiv preprint arXiv:1711.01761*, 2017.

F. Bach and E. Moulines. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems*, 2011.

L. Bottou, F. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. In *arXiv preprint arXiv:1606.04838*, 2016.

B. Dimitri. *Nonlinear programming*. Athena scientific Belmont, 1999.

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

P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. URL http://arxiv.org/abs/1706.02677.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456, 2015.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://dblp.uni-trier.de/db/journals/corr/corr1412.html#KingmaB14.

A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998a.

Y. LeCun, L. Bottou, G. Orr, and K. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 1998b.

D. Needell, R. Ward, and N. Srebro. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. In *Advances in Neural Information Processing Systems*, 2014.

Y. Nesterov. Introductory lectures on convex programming volume i: Basic course. 1998.

H. Robbins and S. Monro. A stochastic approximation method. In *The Annals of Mathematical Statistics*, volume 22, pages 400–407, 1951.

T. Salimans and D. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016.

O. Shamir and T. Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In *International Conference on Machine Learning*, pages 71–79, 2013.

G. Hinton N. Srivastava and K. Swersky. Neural networks for machine learning-lecture 6a-overview of mini-batch gradient descent.

I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

M. Tygert, A. Szlam, S. Chintala, M. Ranzato, Y. Tian, and W. Zaremba. Convolutional networks and learning invariant to homogeneous multiplicative scalings. *arXiv preprint arXiv:1506.08230*, 2015.

A. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pages 4151–4161, 2017.

S. Wright and J. Nocedal. *Numerical Optimization*. Springer New York, New York, NY, 2006. ISBN 978-0-387-40065-5. doi: 10.1007/978-0-387-40065-5_3. URL https://doi.org/10.1007/978-0-387-40065-5_3.

M. Zeiler. Adadelta: an adaptive learning rate method. In *arXiv preprint arXiv:1212.5701*, 2012.

P. Zhao and T. Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *International Conference on Machine Learning (ICML)*, pages 1–9, 2015.

A. Appendix

A.1. Proof ingredients

Lemma A.1 (Descent Lemma) *Let $f \in C_L^1$, i.e., $\forall x, y \in \mathbb{R}^d$, $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$. Then,*

$$f(x) \leq f(y) + \langle \nabla f(y), x - y \rangle + \frac{L}{2}\|x - y\|^2.$$

A.1.1. PROOF OF LEMMA 2.4

Proof: If $b_1 \geq L$, we are done. So suppose $b_1 \leq b_N < L$. Thus,

$$L > b_N = b_1 + \sum_{k=1}^{N-1} \frac{\|\nabla f(x_{k+1})\|^2}{b_k}$$
$$> b_1 + \sum_{k=1}^{N-1} \frac{\|\nabla f(x_{k+1})\|^2}{L}.$$

So $\sum_{k=2}^{N} \|\nabla f(x_k)\|^2 \leq L(L - b_1)$, and hence

$$\min_{k=1:N} \|\nabla f(x_k)\|^2 \leq \frac{1}{N-1} \sum_{k=2}^{N} \|\nabla f(x_k)\|^2$$
$$\leq \frac{L(L - b_1)}{(N-1)} \leq \epsilon.$$

A.1.2. Proof of Lemma 2.5

Suppose k_0 is the first index such that $b_{k_0} > L$. Then $b_j > L$ for all $j \geq k_0$, and by Lemma A.1, for $j \geq k_0$,

$$
\begin{aligned}
f(x_{j+1}) &\leq f(x_j) - \frac{1}{b_j}\left(1 - \frac{L}{2b_j}\right)\|\nabla f(x_j)\|^2 \\
&\leq f(x_j) - \frac{1}{2b_j}\|\nabla f(x_j)\|^2 \\
&\leq f(x_j) - \frac{1}{2b_{j+1}}\|\nabla f(x_j)\|^2 \\
&\leq f(x_{k_0}) - \sum_{\ell=1}^{j} \frac{1}{2b_{k_0+\ell-1}}\|\nabla f(x_{k_0+\ell-1})\|^2.
\end{aligned}
$$

Taking $j \to \infty$,

$$
\sum_{\ell=1}^{\infty} \frac{\|\nabla f(x_{k_0+\ell-1})\|^2}{b_{k_0+\ell-1}} \leq 2(f(x_{k_0}) - f^*).
$$

Now, if $k_0 > 1$, then

$$
\begin{aligned}
b_{k_0+j} - b_{k_0} &= \sum_{\ell=1}^{j} \frac{\|\nabla f(x_{k_0+\ell})\|^2}{b_{k_0+\ell-1}} \\
&\leq 2\sum_{\ell=1}^{j} \frac{\|\nabla f(x_{k_0+\ell-1}) - \nabla f(x_{k_0+\ell})\|^2 + \|\nabla f(x_{k_0+\ell-1})\|^2}{b_{k_0+\ell-1}} \\
&\leq 2\sum_{\ell=1}^{j} \frac{L^2\|x_{k_0+\ell-1} - x_{k_0+\ell}\|^2 + \|\nabla f(x_{k_0+\ell-1})\|^2}{b_{k_0+\ell-1}} \\
&= 2\sum_{\ell=1}^{j} \frac{L^2\|\nabla f(x_{k_0+\ell-1})\|^2}{b_{k_0+\ell-1}^3} + 2\sum_{\ell=1}^{j} \frac{\|\nabla f(x_{k_0+\ell-1})\|^2}{b_{k_0+\ell-1}} \\
&\leq 4\sum_{\ell=1}^{j} \frac{\|\nabla f(x_{k_0+\ell-1})\|^2}{b_{k_0+\ell-1}} \\
&\leq 8(f(x_{k_0}) - f^*) \quad (7)
\end{aligned}
$$

since $b_{k_0} \geq L$. Finally, since $b_j \leq L$ for $j = 1 = 1, 2, \ldots, k_0 - 1$, we can bound $f(x_{k_0}) - f^*$. By Lemma A.1,

$$
\begin{aligned}
f(x_{k_0}) &\leq f(x_1) + \frac{L}{2}\sum_{j=1}^{k_0-1} \frac{\|\nabla f(x_j)\|^2}{b_j^2} \\
&\leq f(x_1) + \frac{L\|\nabla f(x_1)\|^2}{2b_1^2} + \frac{L}{2}\sum_{j=1}^{k_0-2} \frac{\|\nabla f(x_{j+1})\|^2}{b_j^2} \\
&\leq f(x_1) + \frac{L\|\nabla f(x_1)\|^2}{2b_1^2} + \frac{L}{2b_1}(b_{k_0-1} - b_1) \\
&\leq f(x_1) + \frac{Lb_{k_0-1}}{2b_1} \quad (8)
\end{aligned}
$$

since $b_1 \geq \|\nabla f(x_1)\|$ and $b_{k_0-1} \leq L$.

A.1.3. PROOF OF LEMMA 2.6

We use shorthand $\nabla f_k = \nabla f(x_k)$. Let $k_0 \geq 1$ be the first index such that $b_{k_0} \geq L$. Then,

$$
\begin{aligned}
b_{k_0} = b_{k_0-1} + \frac{\|\nabla f_{k_0}\|^2}{b_{k_0-1}} &\leq b_{k_0-1} + 2\frac{\|\nabla f_{k_0} - \nabla f_{k_0-1}\|^2 + \|\nabla f_{k_0-1}\|^2}{b_{k_0-1}} \\
&\leq b_{k_0-1} + 2\left(\frac{L^2\|x_{k_0} - x_{k_0-1}\|^2}{b_{k_0-1}} + \frac{\|\nabla f_{k_0-1}\|^2}{b_{k_0-1}}\right) \\
&= b_{k_0-1} + 2\left(\frac{L^2\|\nabla f_{k_0-1}\|^2}{b_{k_0-1}^3} + \frac{\|\nabla f_{k_0-1}\|^2}{b_{k_0-1}}\right) \\
&\leq b_{k_0-1} + \frac{2L^2\|\nabla f_{k_0-1}\|^2}{b_{k_0-1}^2 b_{k_0-2}} + \frac{2\|\nabla f_{k_0-1}\|^2}{b_{k_0-1}} \\
&= b_{k_0-1} + \frac{2L^2(b_{k_0-1} - b_{k_0-2})}{b_{k_0-1}^2} + \frac{2\|\nabla f_{k_0-1}\|^2}{b_{k_0-1}} \\
&\leq b_{k_0-1} + \frac{2L^2}{b_{k_0-1}} + \frac{2\|\nabla f_{k_0-1}\|^2}{b_{k_0-2}} \\
&\leq 3L + \frac{2L^2}{b_1}.
\end{aligned}
\tag{9}
$$

A.2. Proof of Theorem 2.3

By Lemma 2.4, if $\min_{k=1:N} \|\nabla f(x_k)\|^2 \leq \epsilon$ is not satisfied after $N = \lceil \frac{L(L-b_1)}{\epsilon} \rceil \leq \frac{(1-\delta)L^2}{\epsilon}$ steps, then there is a first index $k_0 \leq N$ such that $b_{k_0} > L$. By Lemma 2.5, for all $k \geq k_0$,

$$
b_k \leq b_{k_0} + 8(f(x_{k_0}) - f^*),
$$

so set

$$
P = b_{k_0} + 8(f(x_{k_0}) - f^*).
$$

If $k_0 = 1$, then it follows that

$$
f(x_M) \leq f(x_1) - \frac{\sum_{k=1}^{M} \|\nabla f(x_k)\|^2}{2(b_1 + 8(f(x_1) - f^*))}
\tag{10}
$$

and thus the stated result holds straightforwardly.

Otherwise, if $k_0 > 1$, then, by Lemma A.1, for any $M \geq 1$,

$$
\begin{aligned}
f(x_{k_0+M}) &\leq f(x_{k_0+M-1}) - \frac{1}{2b_{k_0+M-1}}\|\nabla f(x_{k_0+M-1})\|^2 \\
&\leq f(x_{k_0+M-1}) - \frac{1}{2P}\|\nabla f(x_{k_0+M-1})\|^2 \\
&\leq f(x_{k_0}) - \frac{1}{2P}\sum_{k=1}^{M} \|\nabla f(x_{k_0+k-1})\|^2.
\end{aligned}
$$

By Lemma 2.6, since $b_1 \geq \delta L$, we have

$$
b_{k_0} \leq \left(3 + \frac{2}{\delta}\right)L.
$$

By Lemma 2.4,

$$
f(x_{k_0}) - f^* \leq f(x_1) - f^* + \frac{L}{2\delta}.
$$

Thus,

$$\min_{k=1:M} \|\nabla f(x_{k_0+k-1})\|^2 \leq \frac{1}{M} \sum_{k=1}^{M} \|\nabla f(x_{k_0+k-1})\|^2$$

$$\leq \frac{2P(f(x_{k_0}) - f^*)}{M}$$

$$= \frac{2(b_{k_0} + 8(f(x_{k_0}) - f^*))(f(x_{k_0}) - f^*)}{M}$$

$$\leq \frac{2b_{k_0}(f(x_{k_0}) - f^*)}{M} + \frac{16(f(x_{k_0}) - f^*)^2}{M}$$

$$\leq \frac{2(3 + \frac{2}{\delta})L(f(x_{k_0}) - f^*)}{M} + \frac{16(f(x_{k_0}) - f^*)^2}{M}.$$

Thus, once

$$M \geq \frac{16(f(x_1) - f^* + (\frac{3}{16} + \frac{5}{8\delta})L)^2}{\epsilon},$$

we are assured that

$$\min_{k=1:N+M} \|\nabla f(x_k)\|^2 \leq \epsilon$$

where $N \leq \frac{L^2(1-\delta)}{\epsilon}$.

A.3. Implementing the Algorithm in A Neural Network

In this section, we give the details for implementing our algorithm in a neural network. In the standard neural network architecture, the computation of each neuron consists of an elementwise nonlinearity of a linear transform of input features or output of previous layer:

$$y = \phi(\langle w, x \rangle + b), \tag{11}$$

where w is the d-dimensional weight vector, b is a scalar bias term, x,y are respectively a d-dimensional vector of input features (or output of previous layer) and the output of current neuron, $\phi(\cdot)$ denotes an elementwise nonlinearity. When using backpropogation (LeCun et al., 1998b) the stochastic gradient of g in Algorithms 2, 3 and 4 represent the gradient of the current neuron (see Figure 4). Thus, when implementing our algorithm in PyTorch, WNGrad is one learning rate associated to one neuron, while SGD has one learning rate for all neurons.
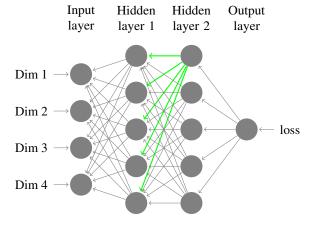

Figure 4. An example of backproporgation of two hidden layers. Green edges represent the stochastic gradient g in Algorithm 3 and 4.

A.4. Tables and Algorithms

Algorithm 5 SGD

Input: Tolerance $\epsilon > 0$
Initialize $x_1 \in \mathbb{R}^d, b_1 \leftarrow 1, j \leftarrow 1$
repeat
    $j \leftarrow j + 1$
    $x_j \leftarrow x_{j-1} - \lambda g_{j-1}$
until $\|\nabla f(x_j)\|^2 \leq \epsilon$

Table 1. Statistics of data sets. DIM is the dimension of a sample

DATASET	TRAIN	TEST	CLASSES	DIM
MNIST	60,000	10,000	10	28×28
CIFAR-10	50,000	10,000	10	32×32

Table 2. architecture for five-layer convolution neural network

LAYER TYPE	CHANNELS	OUT DIMENSION
5×5 CONV RELU	6	28
2×2 MAX POOL, STR.2	6	14
5×5 CONV RELU	16	10
2×2 MAX POOL, STR.2	6	5
FC RELU	N/A	120
FC RELU	N/A	84
FC RELU	N/A	10