Neural Processing Letters (2023) 55:6311–6338
https://doi.org/10.1007/s11063-022-11140-w

AdaLip: An Adaptive Learning Rate Method per Layer for Stochastic Optimization

George Ioannou[1] · **Thanos Tagaris**[1] · **Andreas Stafylopatis**[1]

Accepted: 22 December 2022 / Published online: 4 January 2023

Abstract
Various works have been published around the optimization of Neural Networks that emphasize the significance of the learning rate. In this study we analyze the need for a different treatment for each layer and how this affects training. We propose a novel optimization technique, called AdaLip, that utilizes an estimation of the Lipschitz constant of the gradients in order to construct an adaptive learning rate per layer that can work on top of already existing optimizers, like SGD or Adam. A detailed experimental framework was used to prove the usefulness of the optimizer on three benchmark datasets. It showed that AdaLip improves the training performance and the convergence speed, but also made the training process more robust to the selection of the initial global learning rate.

Keywords Neural networks · Online learning · Stochastic optimization · Adaptive learning rate · Lipschitz constant.

1 Introduction

Neural Networks produce state-of-the-art results for various research fields, such as image recognition [1, 2], speech recognition [3, 4], machine translation [5], autonomous driving [6], text generation [7] and many others. As more and more data become available, the need for deep neural networks becomes more evident. Deep networks can be trained through many learning algorithms, most of them being variations of Stochastic Gradient Descent (SGD). The training task of a neural network can be represented as an optimization problem of finding the best parameters w^* that minimize the loss function, $f : \Re^d \to \Re$, given a set of training

✉ George Ioannou
geoioannou@islab.ntua.gr

Thanos Tagaris
thanos@islab.ntua.gr

Andreas Stafylopatis
andreas@cs.ntua.gr

[1] Artificial Intelligence and Learning Systems Laboratory, National Technical University of Athens, 15780 Zografou, Greece

samples x:

$$w^* = \underset{w}{\operatorname{argmin}} f(w \mid x) \tag{1}$$

The update rule of SGD can be summarized as an iterative movement in the opposite direction of the gradient. That can be seen in the following equation:

$$w_{t+1} = w_t \overbrace{-\alpha_t \underbrace{\nabla f(w_t)}_{g_t}}^{u_t} \tag{2}$$

where α_t is the learning rate. For simplicity, we'll often refer to the gradient as g_t and the update as u_t. The convergence and performance of SGD is greatly influenced by the learning rate, which is why it is one of the most important hyperparameters to fine-tune in a neural network. Selecting a value of α_t larger than the optimal can lead to unpredictable oscillations of the learning curve and even to divergence. On the other hand, small values reduce the convergence speed and make the loss more prone to be trapped in a local minima [8, 9].

The most common practice is to decrease the learning rate during training [10, 11]. However, there are many indications that this is not the best scheduling strategy [12]. Numerous works have been published about the optimal learning rate. The best ones revolve around techniques that change it adaptively depending on various conditions and metrics. In this study a new method will be presented changing the learning rate of each layer based on the Lipschitz constant.

2 Related Work

As mentioned previously, the learning rate is one of the most important hyperparameters in Gradient Descent. Consequently there have been numerous studies aiming at identifying the optimal learning rate. The most common approaches focus on defining a strategy to change the learning rate during training. These are usually referred to as *schedules*. The earliest instance of such a schedule, is the Robbins/Monro theory [10], which states that the learning rate should be chosen to satisfy the following equations:

$$\sum_{t=1}^{\infty} \alpha_t = \infty \text{ and } \sum_{t=1}^{\infty} {\alpha_t}^2 < \infty$$

Another scheduling scheme calls for starting at a relatively high learning rate to achieve fast convergence and then half that every few iterations to ensure small proximity to w^* [13]. Due to its simplicity, variants of this scheme have been employed to train some of the most popular architectures [14]. Another strategy that has proven to be effective in practice is to start training with a constant learning rate and to decrease it by a factor of 2-10 once the loss stops decreasing [15].

A strong assumption usually made by optimization algorithms is that the loss function f is convex. In this setting, a good selection of learning rate is shown to be:

$$\alpha_t = \frac{\alpha_0}{\sqrt{t}} \tag{3}$$

which guarantees a convergence of $\mathbb{E}[f(w_t) - f^*] \leq \mathcal{O}(\log(t)/\sqrt{t})$ without any smooth assumptions, if the base learning rate α_0 is chosen properly [16]. [17] proves with a slightly different framework that the learning rate of Eq. 3 achieves a convergence of $\mathcal{O}(1/\sqrt{t})$.

A strategy gaining significant popularity lately is scheduling the learning rate in a periodical fashion. More specifically, cyclical learning rate schedules (i.e. both increasing and decreasing the learning rate during training) have proven to be very effective in practice [8, 9, 12]. This helps the network to escape bad local minima that the training process could be stuck.

One of the main disadvantages with using SGD as an optimizer is that it scales the update with the magnitude of the gradient in all directions. Sometimes this may lead to slower convergence rates and poor performance. Ideally, the optimization procedure would benefit from being able to choose different learning rates for different weights or set of weights of the network. For example, it could be useful to set higher learning rates for small gradients (or the opposite), when it is needed to reached a better point in the loss space [18].

To overcome this issue, a lot of methods have been proposed, that offer “adaptive" learning rates. The first such algorithm was AdaGrad [19], which adapts the learning rate with the accumulated squared gradient for each parameter individually. This has been shown to improve the convergence rate of SGD in non-convex settings, especially in highly sparse data, as it decreases the learning rate faster for frequent parameters, and slower for infrequent parameters. One major drawback, however, is that the adaptive learning rate tends to get small over time, due to the accumulation of gradients from the beginning of the training.

RMSProp [20] is another optimizer that aims to fix the aforementioned shortcomings of AdaGrad. Instead of accumulating all the squared gradients, it uses an exponential moving average. This is helpful because the moving average of the gradients does not get extremely large forcing the overall learning rate closer to zero. Another major improvement in optimization techniques came in the form of Adam [21], which in addition to scaling the learning rate with the moving average of squared gradients, also averages the gradients themselves. This optimizer has grown substantially in popularity and currently is the “default choice" in most deep learning frameworks [15, 22].

This family of adaptive optimizers, however, are far from perfect and have received a lot of criticism as they lead to biased gradient updates which change the underlying optimization problem [18]. In fact, it has been shown that in some cases, adaptive methods often find drastically different solutions compared to SGD [22]. Other techniques have been proposed to alleviate the aforementioned issues of adaptive learning rates, namely AMSGrad [23] and AdaBound [24], which provide strong theoretical proofs of convergence. [25] proposes another variation of adaptive learning rates, where they theoretically derive the Lipschitz constant for neural networks with different types of loss functions. Another attempt to compute the Lipschitz constant was made by [26]. However, none of these approaches, has yet to surpass the popularity of Adam.

Another approach for adapting the learning rate during training is to update it at each iteration (using backpropagation) in order to maximize some criterion. This is equivalent to “learning” the learning rate for some external goal, e.g. the minimization of the cost function [27], or the squared norm of its derivative [18]. The first technique is interesting, however it falls under the same pitfall mentioned previously, of using a scalar learning rate.

In this paper a new adaptive method is proposed that computes the learning rate based on the Lipschitz constant. The main contributions of the present paper are the following:

- A novel adaptive optimization technique that approximates the Lipschitz constant of the gradient of the loss function to estimate the optimal learning rate.
- An empirical analysis indicating an heterogeneity of the magnitude of the gradients of different layers. This led us to the use of a different learning rate per layer.

Table 1 Differences in popular adaptive optimization methods and how the are structured based on the optimization framework of Algorithm 1

	SGD	AdaGrad	RMSProp	Adam
ϕ_t	g_t	g_t	g_t	$(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}g_i$
ψ_t	$\mathbb{1}$	$\text{diag}(\sum_{i=1}^{t} g_i^2)$	$(1-\beta_2)\,\text{diag}(\sum_{i=1}^{t} \text{fi}_2^{t-i} g_i^2))$	$(1-\beta_2)\,\text{diag}(\sum_{i=1}^{t} \text{fi}_2^{t-i} g_i^2)$

- Through an experimental study we provide insights on neural network training as well as some recommended practices.

3 Theoretical Analysis

3.1 Preliminaries

Before describing our contributions, we will present a generic framework, which can represent all adaptive optimization methods. In the sequel, we will adopt standard notation and relevant mathematical techniques from the literature [21, 23]. We will use $\mathcal{F} \in \Re^d$ to denote the set of feasible points for the weights w_t. We assume that the set $\mathcal{F}$ has bounded diameter $D_\infty \in \Re$, if $\|x - y\| \leq D_\infty$, for all $x, y \in \mathcal{F}$. In the feasible set $\mathcal{F}$, there is another assumption that $\|\nabla f_t(x)\|_\infty$ is bounded $\forall\, t \in [0, 1, ..., T]$. This generic framework is portrayed in Algorithm 1.

Algorithm 1 Generic framework of adaptive optimization methods

Input: $w_t \in \mathcal{F}$, initial step α_0, functions $\{\phi_t, \psi_t\}_{t=1}^{T}$
1: **for** t=1 **to** T **do**
2: $\quad g_t = \nabla f_t(w_t)$
3: $\quad m_t = \phi_t(g_1, ..., g_t)$ and $V_t = \psi_t(g_1, ..., g_t)$
4: $\quad \alpha_t = \alpha_0/\sqrt{t}$
5: $\quad \hat{w}_{t+1} = w_t - \alpha_t m_t/\sqrt{V_t}$
6: $\quad w_{t+1} = \Pi_{\mathcal{F},\sqrt{V_t}}(\hat{w}_{t+1})$
7: **end for**

Table 1 summarizes how various algorithms fit with the generic framework. The main differences in these algorithms lie in the selection of the "averaging" functions ϕ and ψ, through which the parameters m_t and V_t are updated. Through this abstraction, the differences among the various optimizers become more apparent.

3.2 Background

Definition 1 A function $f : \Re^d \to \Re$ is L-Lipschitz continuous if for all $x, y \in \Re^d$ there is a $L > 0$ where[1],

$$\|f(x) - f(y)\| \leq L\|x - y\|$$

and the smallest L that satisfies the above equation is called Lipschitz constant.

[1] All the norms discussed in this study are Euclidean, $\|\cdot\| = \|\cdot\|_2$.

Let $f : \Re^d \to \Re$ be a function with a smooth gradient:

$$\|\nabla f(w_1) - \nabla f(w_2)\| \leq L\|w_1 - w_2\| \ \ \forall w_1, w_2 \in \Re^d \tag{4}$$

Consider an optimization problem, where the goal is to find the set of parameters w that minimize the loss function f (i.e. Eq. 1), through Gradient Descent (Eq. 2).

Lemma 1 *Given a convex loss function f, with a L-Lipschitz continuous gradient (Def. 1) with L being the Lipschitz constant, the optimal learning rate for Gradient Descent is:*

$$\alpha^* = \frac{1}{L} \tag{5}$$

The proof for that can be found in the Appendix A. Computing the optimal learning rate, however, requires prior knowledge of the Lipschitz constant of the loss function's gradient, which is generally not the case.

In practise, it is difficult to find an overall good learning rate because that can change from dataset to dataset and from model to model. Also, because the majority of deep learning models are trained with variants of SGD, meaning noisy gradients, calculating the exact value of L is not easily feasible. If, however, the Lipschitz constant could be *learned* during training and its approximate value is accurate, then one could adapt the learning rate towards its optimal value as training progresses. Up till now this has remained an open research question [18].

3.3 Learning the Lipschitz Constant

Considering that the update steps are quite small for each iteration, we can calculate the constant L of a small subspace, the one that the optimizer explores, assuming that it is L-smooth. This means that the information of the subspace of the loss function needed was computed during a single forward pass. A multiple forward pass scheme could be approached by computing the gradients of different nearby directions, but it costs extra training time making it extremely infeasible for large scale datasets and deep models. Below, we present the approximation of L in a stochastic environment. It can be derived from Eq. 4, if (w_1, w_2) is substituted by (w_t, w_{t-1}):

$$\|\nabla f(w_t) - \nabla f(w_{t-1})\| \leq L\|w_t - w_{t-1}\|$$

From Eq. 2:

$$\begin{aligned}
\|\nabla f(w_t) - \nabla f(w_{t-1})\| &\leq L\|w_{t-1} - \alpha_t \nabla f(w_{t-1}) - w_{t-1}\| \\
\|\nabla f(w_t) - \nabla f(w_{t-1})\| &\leq L\alpha_t \|\nabla f(w_{t-1})\| \\
L &\geq \frac{\|\nabla f(w_t) - \nabla f(w_{t-1})\|}{\alpha_t \|\nabla f(w_{t-1})\|}
\end{aligned}$$

The analysis so far has been based around Gradient Descent. In a realistic scenario the stochastic version will take its place. This translates to the gradients being calculated from a subset of the dataset (i.e. a batch) and, as a result, from a subspace of the loss space. These gradients, g_t, are noisy versions of the total gradient but their expected value is equal to the total:

$$\mathbb{E}_x[g_t] = \mathbb{E}_x[\nabla f(w_t \| x_t)] = \nabla f(w_t)$$

By renaming $\nabla f(w_t)$ as g_t and using Eq. 5, the above inequality transforms to this:

$$\alpha^* \leq \alpha_t \frac{\|g_{t-1}\|}{\|g_t - g_{t-1}\|} \tag{6}$$

where α^* is the optimal learning rate and α_t is the initial learning rate.

Equation 6 offers a way of approximating the optimal value of the learning rate α^*. Ideally, we would want to set the learning rate at each to the value indicated. This practice, however, isn't recommended, as this equation exhibits a high degree of variance from iteration to iteration, which would result in very unstable training.

One issue that arises is due to the denominator, which is the norm of the difference of the gradients of two iterations. This can be seen as the magnitude of change of the direction the optimizer chooses. The problem happens especially near local minima, where the updates are smaller and close to each other. In this situation the denominator might reach values close to zero, causing the learning rate to explode. A workaround is to add a small positive term, c_t, to the denominator:

$$\alpha^* \leq \alpha_t \frac{\|g_{t-1}\|}{\|g_t - g_{t-1}\| + c_t} \tag{7}$$

This creates an artificial low bound to the denominator, which can change over the course of training. In the following sections, the impact of the hyperparameter c_t to the training process, will be explored in more detail.

The instability of the algorithm, however, isn't solely attributed to the denominator; the stochastic nature of the algorithm also plays its part. Equation 7 helps with approximating the optimal learning rate of the loss' subspace, visible through each batch x_t. Our goal is to approximate the optimal learning rate of the underlying loss function, though. To achieve this goal, the moving average of α^* for each batch is computed:

$$S_t = \gamma \cdot S_{t-1} + (1 - \gamma) \cdot \frac{\|g_{t-1}\|}{\|g_t - g_{t-1}\| + c_t}$$

with $\gamma \in (0, 1)$ being the coefficient of the moving average. In closed form this can be written as:

$$S_t = \gamma^t \cdot S_0 + (1 - \gamma) \sum_{i=1}^{t} \gamma^{t-i} \frac{\|g_{i-1}\|}{\|g_i - g_{i-1}\| + c_i} \tag{8}$$

with $S_0 = 1$. Thus, the approximation A_t of the optimal learning rate α^* is calculated as a product of α_t and the above exponential moving average:

$$\begin{aligned} A_t &= \alpha_t \cdot S_t \\ &= \alpha_t \cdot \left[\gamma^t \cdot S_0 + (1 - \gamma) \sum_{i=1}^{t} \gamma^{t-i} \frac{\|g_{i-1}\|}{\|g_i - g_{i-1}\| + c_i} \right] \end{aligned} \tag{9}$$

Lemma 2 *Consider a loss function f that satisfies Definition 1 with bounded gradients $||\nabla f(w_t)|| \leq G$, $\forall\, w_t \in \Re^d$. Let M be the minimum norm of the gradients that is non-zero, then the moving average S_t of Eq. 8 is bounded.*

$$\frac{M(1 - \gamma)}{2G + \max_t c_t} \leq S_t \leq 1 + \frac{G}{\min_t c_t}$$

The above lemma proves that the moving average of Eq. 8 is bounded and, as a result, the learning rate A_t is also bounded. This will be helpful for the next theorem to prove the convergence of a SGD optimizer with such learning rate. For the proof of convergence, the term of regret will be used. Regret (R) is the sum of all previous differences between the current network's prediction $f_t(w_t)$ and the best possible prediction $f_t(w^*)$, obtained from the optimal set of weights w^*. The goal of the proof is to show that the regret averaged over time reaches zero. The regret can be written in the following form:

$$R(T) = \sum_{t=1}^{T} \left[f_t(w_t) - f_t(w^*) \right] \tag{10}$$

Theorem 1 *Assume that the loss function has bounded gradients, $\|g_t\| \leq G$, minimum non-zero norm of gradients M and the weights are in the sphere $\|w_t\| \leq r$. Let $\alpha_t = \alpha_0/\sqrt{t}$ and $\gamma \in (0, 1)$, then an SGD optimizer with Eq. 9 as its learning rate achieves the following guarantees for all $T > 1$ and $c_t \geq \frac{(1-\gamma)\|g_{t-1}\|}{\left(\sqrt{\frac{t}{t-1}}-\gamma\right)S_{t-1}} - \|g_t - g_{t-1}\|$.*

$$\frac{R(T)}{T} \leq \frac{2r^2(2G + c_T)}{\alpha_0 M(1-\gamma)\sqrt{T}} + \frac{G^2\alpha_0}{\sqrt{T}}\left(1 + \frac{G}{c_1}\right)$$

which leads to

$$\frac{R(T)}{T} = \mathcal{O}\left(\frac{1}{\sqrt{T}}\right) \quad \textit{and} \quad \lim_{T\to\infty} \frac{R(T)}{T} = 0$$

The proofs for both Lemma 2 and Theorem 1 can be found in the Appendix A.

3.4 Motivation

Usually in neural networks, parameters within the same layer share similar properties. From layer to layer, though, they might differ. An occasion where this is important is the case of vanishing gradients [28], where the early layers of the network won't be trained adequately. While this problem has been addressed successfully with non-saturating activation functions [29], better weight initialization strategies [30] and transformation layers [28], we can see, in practice, none of the above solve the issue directly.

In order to have a better insight of the training process and how layers differ from each other, an experimental scheme was performed in order to observe the magnitude of the gradients and the magnitude of the updates in 3 different datasets comparing two optimizers, SGD and Adam with steady learning rates. To measure that we computed the mean of the absolute values of the gradients and the updates of each layer of the network for every iteration. The results are shown in Fig. 1. The networks that were used are shown in Appendix B. The blue lines correspond to the earlier iterations of the training and as the colormap reaches the red color, it is the indication for the last iterations. The layers displayed are only Convolutional and Dense layers. We can see that the left two columns display the networks that were trained with SGD. As expected, the magnitude of the updates is proportional to the magnitude of the gradients and their shapes are quite similar. On the other hand, the networks trained by the Adam optimizer showed faster convergence and achieved higher accuracy. Analysing the two right columns of the Fig. 1, we can assess that the magnitude of the updates is not proportional to the magnitude of the gradients and does not mimic the shape of the gradients. This is an insight on what makes Adam better than SGD in this specific scenario. Adam

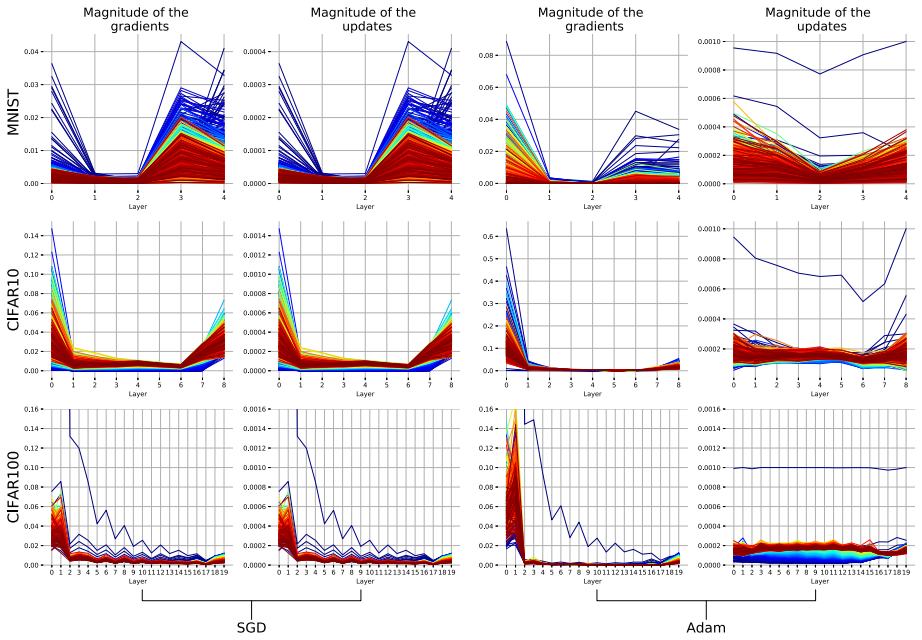

Fig. 1 The magnitude of the gradients and the updates per layer for all datasets (MNIST, CIFAR10, CIFAR100) for two different optizizers (SGD and Adam). The blue lines depict the first iterations. As the color changes to red, it shows iterations closer to the end

is able to construct updates closer to the optimal ones independently of the magnitude of the gradient. SGD usually suffers from that, meaning that some layers in the middle might display low gradients, hindering the path to the optimal updates. This shows the usefulness of an adaptive learning rate per layer that will help any optimizer, like SGD or Adam, scale the gradients and, in consequence, the updates in a way that it will make it easier to reach the optimal weights.

All the weights and gradients that were mentioned in the previous equations of Sect. 3 were the full vectors containing every parameter of the network. However, in a realistic setting it is difficult to construct one large vector of this size (usually would have millions of dimensions) and perform calculations, such as multiplications, computing norms, etc. This is the second reason why it is preferable to compute the updates of the network for every layer separately. This led to the motivation of constructing an optimizer that employs a different adaptive learning rate per layer. In the next section, the full algorithm will be displayed and how it fits with other optimizers.

4 Proposed Optimization Framework

This paper proposes an optimization method, called AdaLip, which adapts the learning rate per layer. This is helpful because it can alleviate issues that occur from underfitted layers, while being able to work with any scheduler that monitors the global learning rate α_t. Additionally, it can efficiently work on top of existing optimizers that use an adaptive learning rate per parameter, e.g. Adam. This study focuses in the conjunction between AdaLip and

three of the most popular optimizers in deep learning: SGD, Adam and RMSProp. A second variation of AdaLip, with a slight modification in its update rule, will also be introduced in Appendix C.

4.1 AdaLip

Algorithms 2, 3 and 4 show the three new optimizers that are created. The first is the vanilla version of AdaLip, where the algorithm adapts the learning rates of a SGD optimizer. As discussed previously, the algorithm estimates the Lipschitz constant for each batch and, through a moving average, approximates the optimal learning rate A_t (Eq. 9). This is done for each layer independently for the reasons mentioned in Sect. 3.4. It is important to note that the norms displayed in the Algorithms 2, 3 and 4 are computed using only the gradients of the weights that are in the same layer with the weight that is being updated in that iteration.

Algorithm 2 AdaLip

Input: $w_t \in \mathcal{F}$, initial step α_0, $\gamma \in (0, 1)$, c_t
Initialize: $S_0 = 1$, $g_0 = 0$
for t=1 **to** T **do**
  $g_t = \nabla f_t(w_t)$
  $\alpha_t = \alpha_0/\sqrt{t}$
  $S_t = \gamma \cdot S_{t-1} + (1-\gamma)\frac{\|g_{t-1}\|}{\|g_t - g_{t-1}\| + c_t}$
  $A_t = \alpha_t \cdot S_t$
  $\hat{w}_{t+1} = w_t - A_t \cdot g_t$
end for

The AdaLip methodology can be applied on top of existing optimizers. One such instance is AdaLip + RMSProp, which will be referred to as RMSLip and is described in Algorithm 3. Again, the optimal learning rate is estimated and is applied for each layer separately. The difference, here, is that each parameter is scaled by the moving average of the squares of its past gradients, as dictated by the RMSProp update rule.

Algorithm 3 RMSLip

Input: $w_t \in \mathcal{F}$, initial step α_0, $\gamma \in (0, 1)$, c_t
Initialize: $S_0 = 1$, $g_0 = 0$, $v_0 = 0$
for t=1 **to** T **do**
  $g_t = \nabla f_t(w_t)$
  $\alpha_t = \alpha_0/\sqrt{t}$
  $S_t = \gamma \cdot S_{t-1} + (1-\gamma)\frac{\|g_{t-1}\|}{\|g_t - g_{t-1}\| + c_t}$
  $A_t = \alpha_t \cdot S_t$
  $v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) g_t^2$
  $\hat{w}_{t+1} = w_t - A_t \cdot g_t / (\sqrt{v_t} + \epsilon)$
end for

Finally, we'll examine the combination of AdaLip and Adam, which will be referred to as AdamLip. The update rule of Adam is left unchanged (i.e. the moving averages of the gradients m_t and their squares v_t and their bias-correction), however in this version, the constant learning rate is substituted with A_t (Eq. 9).

Algorithm 4 AdamLip

Input: $x_t \in \mathcal{F}$, initial step α_0, $\gamma \in (0.1)$, c_t
Initialize: $S_0 = 1$, $g_0 = 0$, $m_0 = 0$, $v_0 = 0$
for t=1 **to** T **do**
  $g_t = \nabla f_t(w_t)$
  $\alpha_t = \alpha_0/\sqrt{t}$
  $S_t = \gamma \cdot S_{t-1} + (1-\gamma)\frac{\|g_{t-1}\|}{\|g_t - g_{t-1}\| + c_t}$
  $A_t = \alpha_t \cdot S_t$
  $m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) g_t$
  $v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) g_t^2$
  $\hat{m}_t = m_t/(1-\beta_1^t)$
  $\hat{v}_t = v_t/(1-\beta_2^t)$
  $\hat{w}_{t+1} = w_t - A_t \cdot \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$
end for

4.2 Implementation Details

This Section will explain how the hyperparameter values for AdaLip were chosen and initialized.

From Theorem 1 the optimal c_t has been calculated in order to have a guarantee of convergence. In practice, the experiments were run with a constant $c_t = c = 10^{-8}$, which is close to the theoretical c_t. This will be discussed in more detail in Sect. 6.2.

For the initial learning rate α_0 we selected many values in order to test its robustness and it will be further analyzed in Sect. 5. The initial value of the moving average S_0 is set to one, because then the overall learning rate will start at the initial learning rate α_0.

Another important hyperparameter to tune is the coefficient γ of the moving average. Its range is (0, 1) and the proposed value is 0.8, which was found empirically. Lower values tend to affect the learning rate to change drastically from iteration to iteration. This could be beneficial to overcome bad local minima that the the network can get stuck, but it also makes the algorithm unstable. On the other hand, higher values will smooth out the overall learning rate containing any possible spikes.

5 Experimental Framework

In order to accurately capture the performance of this novel optimization technique, three benchmark datasets were used, MNIST [31], CIFAR10 [32] and CIFAR100 [32]. AdaLip will be compared with its counterparts (i.e. AdaLip versus SGD, RMSLip vs RMSProp and AdamLip vs Adam) in terms of convergence speed and robustness to the selection of the initial learning rate a_0. The goal is to determine whether or not this addition improves the training procedure of a Neural Network.

Because of the varying levels of difficulty in each task, a different architecture was used for each. These can be seen in Tables 6, 7 and 8, in Appendix B of the Appendix. The first two are small versions of the VGG [33] architecture, while the CIFAR100 one has a lot more depth and utilizes BatchNormalization [28] layers as well as Dropout [34].

For result stability, every setup for every optimizer was run multiple times (25 runs for MNIST and CIFAR10, 10 runs for CIFAR100). 25 runs are enough to capture the overall variance of the different initializations of the networks for MNIST and CIFAR10. Regarding CIFAR100, we selected less runs because, due to the larger number of training epochs

compared to the other two datasets, it was observed that the variance between runs was quite small.

In every run, the optimizers were initialized with the same weights and hyperparameters. Because different optimizers perform best with different values for the base learning rate, various values were tested for each optimizer and dataset.

To evaluate the results, we first found the peak training accuracy for each run and then took the median of those values over all 25 runs, for each learning rate independently. This was done because we consider the most successful optimizer to be the one that outperforms the others consistently, independent of random initialization. The next step of evaluation is to assess how sensitive is the optimizer to the selection of the base learning rate. For this reason we took the max, mean, median and std out of all the different learning rates experiments. This is displayed in Tables 2, 3 and 4 for each dataset, which will be discussed below. Another important feature of an optimizer is the speed of convergence. That can be measured with a lot of ways. In this study the metric was the number of epochs needed to reach 95% of the maximum training peak. The maximum training is computed out of all runs and different learning rate experiments. Similar with the accuracy measurement, the mean, median and best (lowest) Epoch of convergence (98% of maximum peak for MNIST and 95% for both CIFAR) were computed to capture a more complete image of the training process.

5.1 MNIST

First, AdaLip was tested on MNIST, a dataset of small grayscale images of handwritten digits, with a size of 28×28. It consists of 60000 training images and 10000 for testing. The

Table 2 Training set accuracy (top 4 rows) and Epochs of convergence (bottom 3 rows) for the MNIST dataset

	SGD	AdaLip	Adam	AdamLip	RMSProp	RMSLip
Max	**1.0**	**1.0**	0.999	**1.0**	**1.0**	**1.0**
Median	0.976	**0.996**	**0.999**	**0.999**	**1.0**	**1.0**
Mean	0.663	**0.883**	**0.999**	**0.999**	**1.0**	**1.0**
Std	0.422	**0.288**	0.001	**0.000**	**0.000**	**0.000**
Mean	4.5	**3.1**	1.75	**1.5**	1.5	**1**
Median	2.5	**1.5**	2	**1.5**	1.5	**1**
Best	2	**1**	**1**	**1**	**1**	**1**

The bold refers to which is the best

Table 3 Training set accuracy (top 4 rows) and Epochs of convergence (bottom 3 rows) for the CIFAR10 dataset

	SGD	AdaLip	Adam	AdamLip	RMSProp	RMSLip
Max	**0.945**	0.940	0.978	**0.984**	0.975	**0.987**
Median	0.756	**0.838**	0.971	**0.973**	0.968	**0.983**
Mean	0.614	**0.675**	0.816	**0.826**	0.910	**0.937**
Std	0.346	**0.310**	0.299	**0.298**	0.120	**0.084**
Mean	**27.4**	27.6	**22**	22.571	21.2	**20.6**
Median	30	**29**	**21**	22	19	**17**
Best	**21**	22	**14**	**14**	16	**15**

The bold refers to which is the best

Table 4 Training set accuracy (top 4 rows) and Epochs of convergence (bottom 3 rows) for the CIFAR100 dataset

	SGD	AdaLip	Adam	AdamLip	RMSProp	RMSLip
Max	0.965	**0.978**	0.966	**0.967**	0.956	**0.966**
Median	0.935	**0.965**	0.956	**0.961**	0.953	**0.962**
Mean	0.928	**0.939**	0.952	**0.958**	0.945	**0.957**
Std	**0.034**	0.054	0.015	**0.010**	0.016	**0.012**
Mean	60	**53.66**	44.75	**42.5**	45.4	**41.6**
Median	62.5	**49**	42.5	**40.5**	40	**38**
Best	44	**41**	**34**	**34**	37	**34**

The bold refers to which is the best

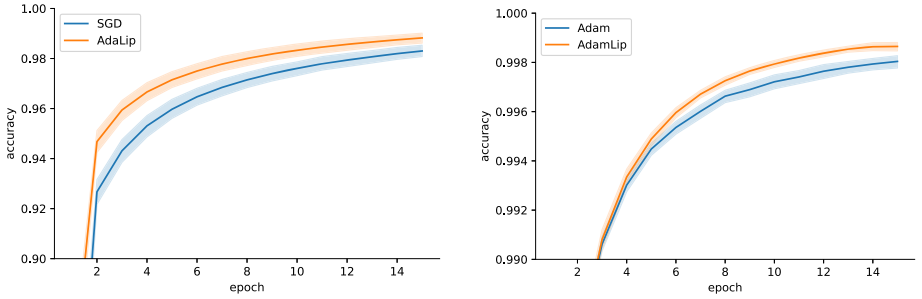

Fig. 2 Mean Curves out of all learning rates on MNIST for Adam and SGD based optimizers

images were normalized to the range of [0, 1] and no further preprocessing or augmentation was added.

The SGD-based optimizers were examined for 8 different learning rates for this task (i.e. 0.005, 0.01, 0.05, 0.1, 0.5, 0.7, 1.0, 1.3), Adam-based ones were examined for 4 (i.e. 0.0005, 0.001, 0.005, 0.01) and RMSProp optimizers for 6 (i.e. 0.0003, 0.0005, 0.0007, 0.001, 0.005, 0.01).

AdaLip in conjunction with every optimizer seems to perform better in both max and mean performance. Also, it has lower std meaning that the results are more stable and fluctuate less with the change of learning rate. Regarding the Epochs of convergence we see a slight improvement in mean and median of the Epochs of conversion. Considering that MNIST is an easy to train dataset and it converges extremely fast, there is not any room for big improvements to achieve. For a closer look, the mean curve of all learning rates for SGD and Adam based optimizers is displayed in Fig. 2. The shading represents the variance from different learning rate experiments. The AdaLip versions seem to converge faster and reach higher accuracies.

5.2 CIFAR10

CIFAR10 was the second dataset to test our methodology. CIFAR10 consists of colored images with 32 × 32 size. The training set has 50000 images and the test set 10000, while they are divided in 10 classes. The images were normalized as in MNIST between [0, 1] and no further preprocessing was added.

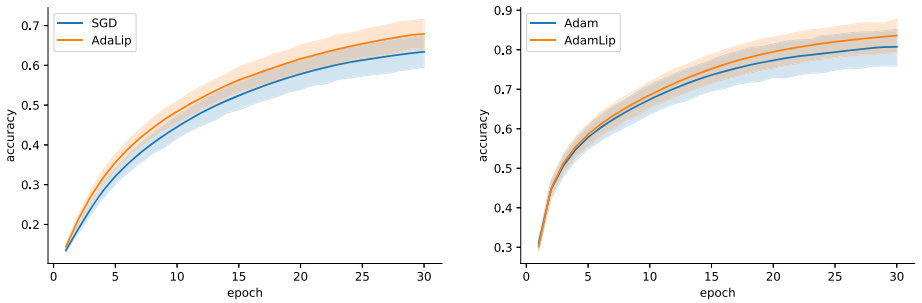

Fig. 3 Mean Curves out of all learning rates on CIFAR10 for Adam and SGD based optimizers

In this task, the SGD-based optimizers were examined for 10 learning rates (i.e. 0.005, 0.01, 0.05, 0.07, 0.1, 0.2, 0.3, 0.5, 0.7, 1.0), Adam-based ones for 7 learning rates (i.e. 0.0002, 0.0003, 0.0005, 0.0007, 0.001, 0.005, 0.01) and RMSProp-based ones for 5 (i.e. 0.0003, 0.0005, 0.0007, 0.001, 0.005). The same procedure was followed here, which yielded similar results with MNIST. AdamLip and RMSLip outperformed their counterparts on mean and max scores. AdaLip scored 0.5% lower in max performance but the mean and median results were quite an improvement compared to SGD. Regarding the std scores RMSLip shows great stability while the rest display similar behavior. As far as the Epochs of convergence analysis, Adam and SGD are slight ahead (by 1 epoch or less on average) but RMSLip is faster than its counterpart. The mean curve of all learning rates for SGD and Adam based optimizers is plotted in Fig. 3. It is clear that the AdaLip versions converge faster and to higher accuracy.

5.3 CIFAR100

The last dataset used for testing the proposed framework was CIFAR100. It has the same images as CIFAR10, but, instead of 10 classes, the dataset is divided in 100. This makes it a lot more difficult to train a neural network and to achieve a good generalization score. To help with the generalization, image augmentation was performed after the normalization. The augmentation techniques were random rotations, flips and width/height shifts that help with the better training of the network.

Here we examined 6 learning rates (i.e. 0.005, 0.01, 0.05, 0.1, 0.5, 0.7) for SGD-based optimizers, 4 learning rates (i.e. 0.0005, 0.001, 0.005, 0.01) for Adam-based ones and 6 (i.e. 0.0003, 0.0005, 0.0007, 0.001, 0.005, 0.01) for RMSProp-based ones. Again, in all max and mean scores the AdaLip-based algorithms show an improvement in the overall training procedure. AdaLip has while having a lower std. Regarding the Epochs of convergence AdaLip shows great improvement over SGD. AdamLip converges by 2 epochs faster on average than Adam, while RMSLip displays a faster overall training progress. Figure 3 shows the mean curves, like in the previous datesets.

5.4 Generalization Performance

The focus of this paper is mainly revolving around the training performance of the optimizers, but, the generalization performance is extremely important as well. In order to measure the testing scores of each optimizer, we computed the mean of the maximum test accuracy of each run of the aforementioned experimental process. Meaning that for each optimizer and

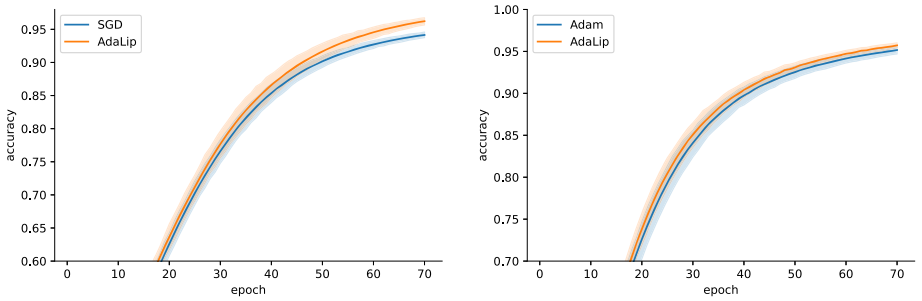

Fig. 4 Mean Curves out of all learning rates on CIFAR100 for Adam and SGD based optimizers

Table 5 Test accuracies for the MNIST, CIFAR10 and CIFAR100 datasets

	MNIST	CIFAR10	CIFAR100
SGD	0.9896	0.6632	0.5706
AdaLip	**0.9909**	**0.6670**	**0.5802**
Adam	0.9910	**0.7545**	0.5937
AdamLip	**0.9911**	**0.7545**	**0.5981**
RMSProp	0.9902	0.7528	0.5868
RMSLip	**0.9909**	**0.7622**	**0.5884**

The bold refers to which is the best

for each learning rate a mean testing accuracy is extracted for the three benchmark datasets. Table 5 displays the best test accuracies of the optimizers among all the starting learning rates for each case. We can see that in most cases the scores are close, but AdaLip's version display an improvement. Specifically, AdamLip's performance is slightly better than Adam in all 3 datasets. RMSProp seems to achieve better generalization in CIFAR10 than RMSLip, but on the the other datasets the improvement is smaller. Lastly, regarding SGD and AdaLip, AdaLip seems to perform better in all 3 datasets. Overall, the test scores show a slight advantage for the AdaLip-based optimizers.

6 Discussion and Future Work

In this section a discussion will be held about various aspects of the proposed algorithm, the theoretical analysis as well as some insights on the whole training process of DNNs.

6.1 Learning Rates Per Layer

As previously mentioned in Sect. 3.4, the norms of the weights of the layers follow a U-like shape. Specifically, the first and last layers tend to exhibit larger norms. In Figs. 5, 6 and 7 we can see that in most situations the algorithm chose a different learning rate for different layers, which seems to validate our initial intuition. It is important to note that the AdaLip and AdamLip optimizers show a wider variance over time compared to RMSLip. Another observation is that despite the norms of the weights have a fixed tendency the learning rates per layer do not. This means that the learning rate successfully adapts to the needs of each architecture and dataset uniquely.

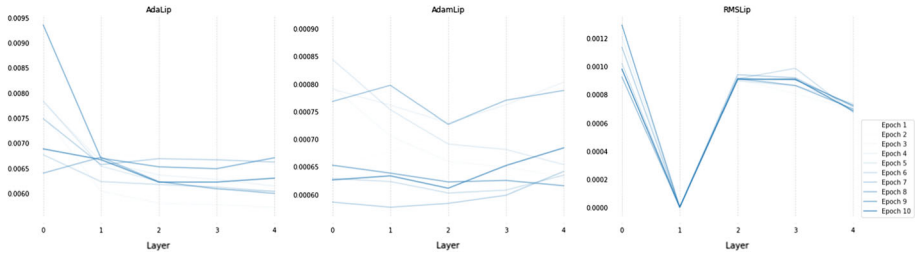

Fig. 5 MNIST Learning rates per layer

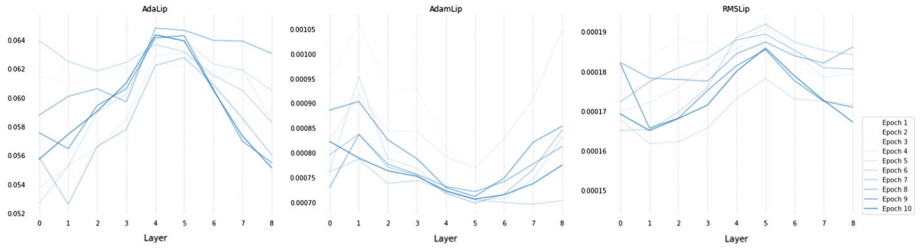

Fig. 6 CIFAR10 Learning rates per layer

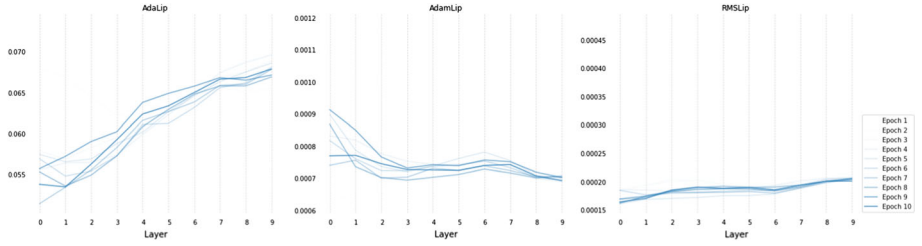

Fig. 7 CIFAR100 Learning rates per layer

6.2 Theoretical versus Practical c_t

One of the main discussion points is the selection of c_t and how it stands between the two most important features of an optimizer, the theoretical guarantee of convergence and the overall performance. As mentioned in the Related Work section one of the most used schedules for a guarantee in convergence is dividing the initial learning rate by the square root of the number of the iterations (see Eq. 3). However, in practice this is not preferred. The decay applied by the square root is too great for any real application that needs thousands of iterations to reach a good performance. With that many iterations the learning rate will become extremely small at an early stage of the training resulting in converging to a bad local minimum. On the other hand, a more steady learning rate seems to provide better results. This can be seen in Fig. 8 where constant learning rate achieves better convergence than the theoretical one.

Recent studies [8, 9, 12] show that an oscillating learning rate performs in some cases better than a strictly decreasing one, because it helps the optimizer overcome local minima.

Similarly to the case of c_t, the theoretical value has the same performance as a constant value, except for some cases where constant c_t has a slight edge over the theoretical one. This can be seen in Fig. 9, where all runs with the same learning rate are really close, but constant

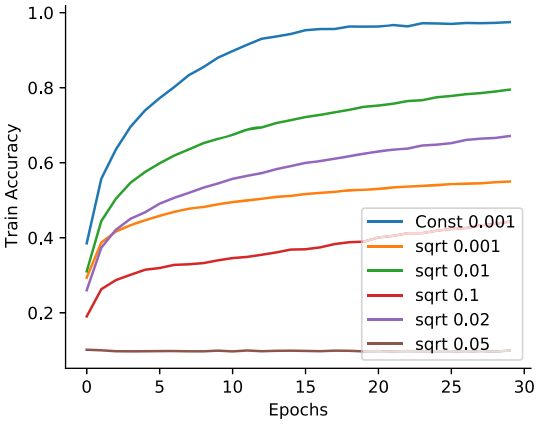

Fig. 8 Constant Learning rate vs Square root decay (with various learning rates) with Adam on CIFAR10

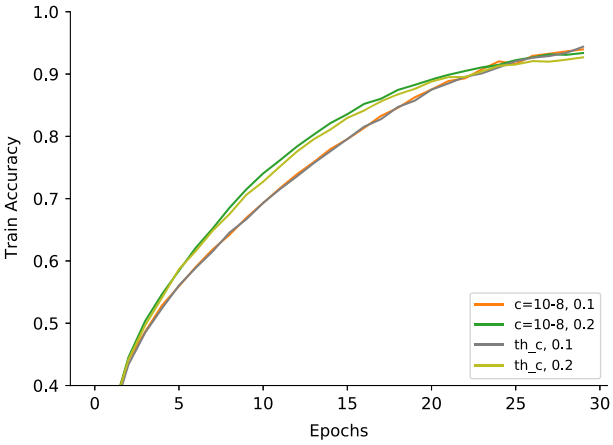

Fig. 9 Different runs with learning rates (0.1, 0.2) and different values of c_t (theoretical c_t, 10^{-8}) with AdaLip on CIFAR10

c_t shows a small improvement. This trade-off between the guarantee of convergence and better results is quite important and can be rephrased as a trade-off of stability versus peak performance. One of the reasons that this behavior exists lies in the nature of SGD. In order to prove that SGD's (or its variants) Regret converges to 0 over time, various assumptions are being made about the loss landscape. In practice, though, the landscape may not be ideal and converging to the first local minimum might not be satisfactory [35, 36]. Naturally, escaping saddle points and local minima has been a focal point in optimization research [11, 37, 38].

6.3 SGD Based Equation for Optimal LR

In order to find the optimal learning rate we used Lemma 1, which is based on Gradient Descent. However, we have to mention the reason that the optimal learning rate based on SGD was not used. Various experiments showed that the equation derived from SGD leads to a quite unstable learning progress. The times when the network was trained normally, the convergence speed was notably slower to the point that no amount of fine-tuning could have

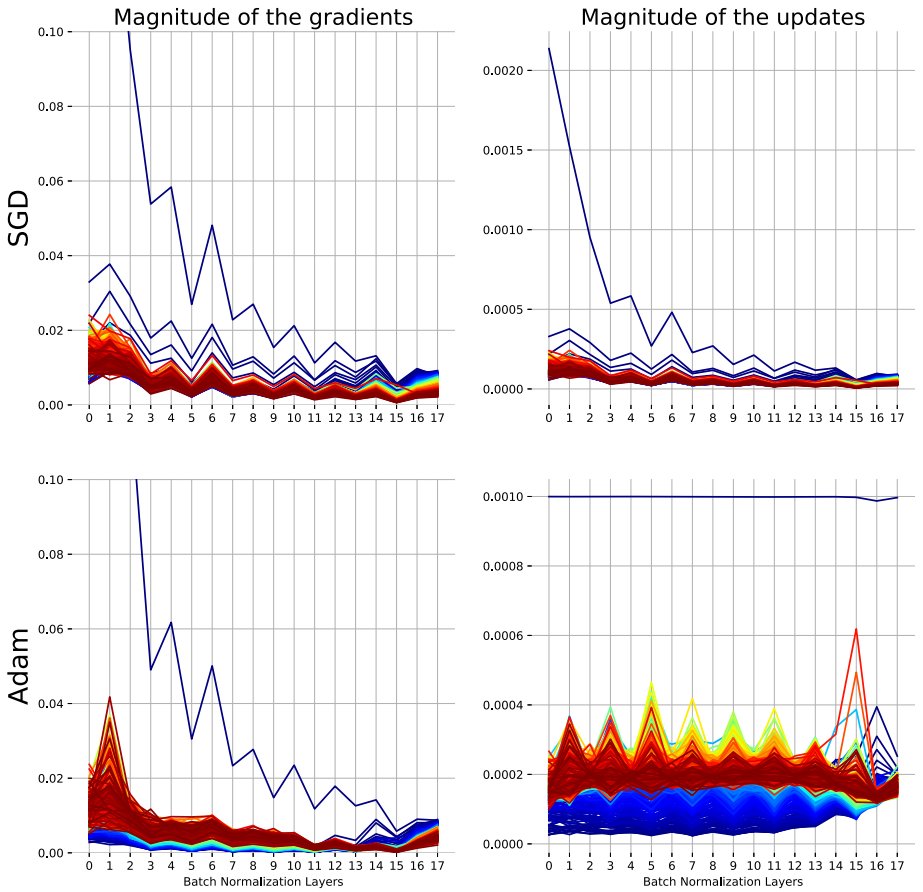

Fig. 10 Magnitude of gradients and updates on CIFAR100 of the BatchNormalization layers (regarding gamma and beta weights). The blue lines depict the first iterations. As the color changes to red, it shows iterations closer to the end

made it competitive. In the Appendix in Lemma 6 there is the full formula for the optimal learning rate based on SGD. It would be very interesting for future work to see if there is a way to apply this efficiently.

6.4 Impact of Batch Normalization

Another important observation about the norms of the layers arises with the use of the Batch Normalization. As can be seen from Fig. 10, the magnitude of the BatchNormalization weights does not always follow the pattern shown earlier in Fig. 1. This means that the *gamma* and *beta* weights of the BatchNormalization layers display a different behaviour compared with the rest of the weights. Although Batchnormalization helps the training to smooth the gradients of the layers [28], it creates weights that suffer from the same problem. This is the reason that AdaLip is able to construct a suitable learning rate for all these parameters. This is enforced by the fact that in practice, with networks that apply Batch Normalization (Sec. 5.3), AdaLip seems to perform better than other optimizers.

7 Conclusion

To summarize, this paper proposes a novel optimization method, called AdaLip, which uses the theoretical optimal learning rate based on the Lipschitz constant to produce an adaptive learning rate per layer. The motivation behind the idea was an analysis on the different behaviors of different layers of a network. We show that this method helps with the overall training process of neural networks in convergence speed. This was supported by a number of experiments on three benchmark datasets. An advantage to the proposed technique is that it can work together with various existing optimizers, such as Adam or RMSProp, while being more robust in selecting the starting learning rate.

Funding Open access funding provided by HEAL-Link Greece. The authors did not receive support from any organization for the submitted work.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose. The authors give consent for publication.

Code availability The code can be found at https://github.com/geoioannou/AdaLip The authors have equal contributed to the work.

Appendix A Proofs

Proof of **Lemma 1**

Let $w_{t+1} = w_t - \alpha \nabla f(w_t)$ be the update rule. From the L-Lipschitz continuity we have the following:

$$\nabla^2 f(w) \preceq LI$$

meaning that the eigenvalues of the Hessian are bounded by L. This can be written as:

$$u^T \nabla^2 f(v) u \leq u^T L I u \tag{A.1}$$

From the Taylor series of f, it can be derived that:

$$\begin{aligned} f(w_{t+1}) = f(w_t) + \nabla f(w_t)^T (w_{t+1} - w_t) + \\ + \frac{1}{2}(w_{t+1} - w_t)^T \nabla^2 f(w_t)(w_{t+1} - w_t) \end{aligned} \tag{A.2}$$

Using Eq. (A.1) and the update rule on the second part of Eq. (A.2):

$$\begin{aligned} f(w_{t+1}) &\leq f(w_t) + \nabla f(w_t)^T (w_{t+1} - w_t) + \frac{1}{2} L \| w_{t+1} - w_t \|^2 \\ &= f(w_t) + \nabla f(w_t)^T (-\alpha \nabla f(w_t)) + \frac{1}{2} L \| - \alpha \nabla f(w_t) \|^2 \end{aligned}$$

$$= f(w_t) - \alpha \nabla f(w_t)^T \nabla f(w_t) + \frac{1}{2} L \alpha^2 \|\nabla f(w_t)\|^2$$
$$= f(w_t) - (\alpha - \frac{\alpha^2 L}{2}) \|\nabla f(w_t)\|^2$$

In order to guarantee the decrease of the function f during the iterations, then the following must be true:

$$(\alpha - \frac{\alpha^2 L}{2}) > 0 \Rightarrow \alpha < \frac{2}{L}$$

To maximize the decrease of function f per iteration, the derivative of f in regard to α must be zero:

$$\frac{\partial(\alpha - \frac{\alpha^2 L}{2})}{\partial \alpha} = 0 \Rightarrow \alpha = \frac{1}{L}$$

□

Proof of **Lemma 2**

From Eq. 8, S_t can be written as:

$$S_t = \gamma^t S_0 + (1-\gamma) \sum_{i=1}^{t} \gamma^{t-i} \frac{||g_{i-1}||}{||g_i - g_{i-1}|| + c_i}$$

For the upper bound we take:

$$S_t \leq S_0 + (1-\gamma) \sum_{i=1}^{t} \gamma^{t-i} \frac{G}{\min_t(c_t)}$$
$$\leq 1 + \frac{(1-\gamma)G}{\min_t(c_t)} \sum_{i=1}^{t} \gamma^{t-i}$$

The sum on the right is the sum of a geometric series and because $\gamma \neq 1$ then it can be transformed to:

$$\leq 1 + \frac{(1-\gamma)G}{\min_t(c_t)} \cdot \frac{1-\gamma^t}{1-\gamma}$$
$$\leq 1 + \frac{(1-\gamma^t)G}{\min_t(c_t)} \leq 1 + \frac{G}{\min_t(c_t)}$$

For the lower bound we use :

$$S_t \geq (1-\gamma) \sum_{i=1}^{t} \gamma^{t-i} \frac{\|g_{i-1}\|}{\|g_i - g_{i-1}\| + c_i}$$
$$\geq (1-\gamma) \sum_{i=1}^{t} \gamma^{t-i} \frac{M}{2G + \max_t(c_t)}$$
$$\geq (1-\gamma) \frac{M}{2G + \max_t(c_t)} \sum_{i=1}^{t} \gamma^{t-i}$$

Computing the same geometric series we get:

$$S_t \geq \frac{M}{2G + \max_t(c_t)} (1-\gamma^t) \geq \frac{M(1-\gamma)}{2G + \max_t(c_t)}$$

which concludes the proof of the upper and lower bounds of S_t.

□

Proof of **Theorem 1**

From the projected stochastic subgradient method $w_{t+1} = \Pi_D(w_t - A_t g_t)$, we have:

$$\begin{aligned} ||w_{t+1} - w^*||^2 &\leq ||w_t - w^* - A_t g_t||^2 \\ &= ||w_t - w^*||^2 - 2A_t\langle g_t, w_t - w^*\rangle + A_t^2||g_t||^2 \end{aligned}$$

From Lemma 3 we have:

$$\leq ||w_t - w^*||^2 - 2A_t(f_t(w_t) - f_t(w^*)) + A_t^2||g_t||^2$$

Rearranging this we get:

$$\begin{aligned} 2A_t(f_t(w_t) - f_t(w^*)) &\leq ||w_t - w^*||^2 - ||w_{t+1} - w^*|| + A_t^2||g_t||^2 \\ f_t(w_t) - f_t(w^*) &\leq \frac{1}{2A_t}||w_t - w^*||^2 - \\ -\frac{1}{2A_t}||w_{t+1} - w^*|| &+ \frac{A_t}{2}||g_t||^2 \end{aligned}$$

Summing from t=1,2,..,T to form the Regret, we have:

$$\begin{aligned} R(T) &= \sum_{t=1}^{T}\left(f_t(w_t) - f_t(w^*)\right) \leq \\ &\leq \sum_{t=1}^{T}\frac{1}{2A_t}||w_t - w^*||^2 - \sum_{t=1}^{T}\frac{1}{2A_t}||w_{t+1} - w^*|| + \\ &+ \sum_{t=1}^{T}\frac{A_t}{2}||g_t||^2 \end{aligned}$$

Unfolding the sums we get:

$$\begin{aligned} &\leq \left(\frac{1}{2A_1}||w_1 - w^*||^2 + \frac{1}{2A_2}||w_2 - w^*||^2 + ...\right. \\ &\left. + \frac{1}{2A_T}||w_T - w^*||^2\right) - \\ &- \left(\frac{1}{2A_1}||w_2 - w^*||^2 + \frac{1}{2A_2}||w_3 - w^*||^2 + ... + \frac{1}{2A_T}||w_{T+1} - w^*||^2\right) \\ &+ \sum_{t=1}^{T}\frac{A_t}{2}||g_t||^2 \\ &= \frac{1}{2A_1}||w_1 - w^*||^2 + \sum_{t=1}^{T-1}\left(\frac{1}{2A_{t+1}} - \frac{1}{2A_t}\right)||w_{t+1} - w^*||^2 - \\ &- \frac{1}{2A_T}||w_{T+1} - w^*||^2 + \frac{1}{2}\sum_{t=1}^{T}A_t||g_t||^2 \\ &\leq \frac{1}{2A_1}||w_1 - w^*||^2 + \sum_{t=1}^{T-1}\left(\frac{1}{2A_{t+1}} - \frac{1}{2A_t}\right)||w_{t+1} - w^*||^2 + \end{aligned}$$

$$+\frac{1}{2}\sum_{t=1}^{T} A_t ||g_t||^2$$

Because the gradients are bounded $\|g_t\| \leq G$ and using lemma 4 we have:

$$\leq \frac{4r^2}{2A_1} + \frac{1}{2}\sum_{t=1}^{T} A_t G^2 + \sum_{t=1}^{T-1}\left(\frac{1}{2A_{t+1}} - \frac{1}{2A_t}\right)||w_{t+1} - w^*||^2$$

$$= \frac{2r^2}{A_1} + \frac{G^2}{2}\sum_{t=1}^{T} A_t + \frac{1}{2}\sum_{t=1}^{T-1}\left(\frac{1}{A_{t+1}} - \frac{1}{A_t}\right)||w_{t+1} - w^*||^2$$

From lemma 5 we prove that with certain c_t the difference is positive so we can use lemma 4 to get:

$$\leq \frac{2r^2}{A_1} + \frac{G^2}{2}\sum_{t=1}^{T} A_t + \frac{4r^2}{2}\sum_{t=1}^{T-1}\left(\frac{1}{A_{t+1}} - \frac{1}{A_t}\right)$$

Now the sum becomes telescopic:

$$= \frac{2r^2}{A_1} + \frac{G^2}{2}\sum_{t=1}^{T} A_t + 2r^2\left(\frac{1}{A_T} - \frac{1}{A_1}\right)$$

$$= \frac{2r^2}{A_T} + \frac{G^2}{2}\sum_{t=1}^{T} A_t$$

$$= \frac{2r^2\sqrt{T}}{\alpha_0 S_T} + \frac{G^2\alpha_0}{2}\sum_{t=1}^{T}\frac{S_t}{\sqrt{t}}$$

Clearly the behavior of A_t affects the convergence of the whole algorithm. The left term shows that if A_t decreases really fast (i.e. $\mathcal{O}(1/t^2)$) the algorithm diverges. On the other hand, if A_t is constant or increases then the right term (the sum) will diverge. Lemma 5 shows that c_t controls how fast A_t decreases. There can be many c_t that lead to convergence with various speeds. Here one of them is presented that satisfies the following equation:

$$c_t = \max\left\{\frac{(1-\gamma)\|g_{t-1}\|}{\left(\sqrt{\frac{t}{t-1}}-\gamma\right)S_{t-1}} - \|g_t - g_{t-1}\|,\ \ c_{t-1}\right\} \tag{A.3}$$

Using the bounds of S_t from Lemma 2 we have:

$$R(T) \leq \frac{2r^2\sqrt{T}}{\alpha_0\frac{M(1-\gamma)}{2G+\max_T(c_T)}} + \frac{G^2\alpha_0}{2}\sum_{t=1}^{T}\frac{1}{\sqrt{t}}\left(1 + \frac{G}{\min_T(c_T)}\right)$$

From the Eq. A.3 of c_t it is clear that c_t is non-decreasing, thus, the above inequality transforms into:

$$R(T) \leq \frac{2r^2(2G + c_T)\sqrt{T}}{\alpha_0 M(1-\gamma)} + \frac{G^2\alpha_0}{2}\left(1 + \frac{G}{c_1}\right)\sum_{t=1}^{T}\frac{1}{\sqrt{t}}$$

$$R(T) \leq \frac{2r^2(2G + c_T)\sqrt{T}}{\alpha_0 M(1-\gamma)} + \frac{G^2\alpha_0}{2}\left(1 + \frac{G}{c_1}\right)\int_0^T \frac{1}{\sqrt{t}}dt$$

$$R(T) \leq \frac{2r^2(2G + c_T)\sqrt{T}}{\alpha_0 M(1-\gamma)} + G^2\alpha_0\left(1 + \frac{G}{c_1}\right)\sqrt{T}$$

Dividing by T:

$$\frac{R(T)}{T} \leq \frac{2r^2(2G + c_T)}{\alpha_0 M(1-\gamma)\sqrt{T}} + \frac{G^2\alpha_0}{\sqrt{T}}\left(1 + \frac{G}{c_1}\right)$$

which shows that:

$$\frac{R(T)}{T} = \mathcal{O}\left(\frac{1}{\sqrt{T}}\right) \text{ and } \lim_{T\to\infty}\frac{R(T)}{T} = 0$$

□

Lemma 3 *A function $f : \Re^d \longrightarrow \Re$ is convex, then for all $x, y \in \Re^d$,*

$$f(y) \geq f(x) + \nabla f(x)^T (y - x)$$

Lemma 4 *Given a weight $w_i \in \Re_d$ and $\|w_i\| \leq r$, then $\|w_n - w_m\|^2 \leq 4r^2$, $\forall n, m$.*

Proof From the triangular inequality we have:

$$\begin{aligned}\|w_n - w_m\|^2 &= \|w_n^2 - 2w_n w_m + w_m^2\| \\ &\leq \|w_n\|^2 + 2\|w_n\|\|w_m\| + \|w_m\|^2 \\ &\leq r^2 + 2rr + r^2 = 4r^2\end{aligned}$$

□

Lemma 5 *Let A_t be the learning rate from Eq. 9, $\alpha_t = \alpha_0/\sqrt{t}$. Then $A_t \leq A_{t-1}$ is true, if c_t is a positive function of t and satisfies the following inequality:*

$$c_t \geq \frac{(1-\gamma)\|g_{t-1}\|}{\left(\sqrt{\frac{t}{t-1}} - \gamma\right) S_{t-1}} - \|g_t - g_{t-1}\|$$

Proof Starting from the inequality we have:

$$\begin{aligned}&A_t \leq A_{t-1} \\ &\alpha_t \cdot S_t \leq \alpha_{t-1} \cdot S_{t-1} \\ &\frac{\alpha_0}{\sqrt{t}}\left(\gamma S_{t-1} + (1-\gamma)\frac{\|g_{t-1}\|}{\|g_t - g_{t-1}\| + c_t}\right) \leq \frac{\alpha_0}{\sqrt{t-1}} S_{t-1} \\ &\gamma S_{t-1} + (1-\gamma)\frac{\|g_{t-1}\|}{\|g_t - g_{t-1}\| + c_t} \leq \sqrt{\frac{t}{t-1}} S_{t-1} \\ &(1-\gamma)\frac{\|g_{t-1}\|}{\|g_t - g_{t-1}\| + c_t} \leq \left(\sqrt{\frac{t}{t-1}} - \gamma\right) S_{t-1} \\ &\|g_t - g_{t-1}\| + c_t \geq \frac{(1-\gamma)\|g_{t-1}\|}{\left(\sqrt{\frac{t}{t-1}} - \gamma\right) S_{t-1}} \\ &c_t \geq \frac{(1-\gamma)\|g_{t-1}\|}{\left(\sqrt{\frac{t}{t-1}} - \gamma\right) S_{t-1}} - \|g_t - g_{t-1}\|\end{aligned}$$

□

Lemma 6 *Given a convex loss function f, with an L-Lipschitz continuous gradient (Def. 1), the optimal learning rate for Stochastic Gradient Descent is:*

$$\alpha^* = \frac{\|\nabla f(w_t)\|^2}{L \cdot E\left[\|\nabla f(w_t)\|^2\right]}$$

Proof The update rule for SGD is the following:

$$w_{t+1} = w_t - \alpha \nabla f_t(w_t)$$

where f_t is the partial f computed by a mini-batch at iteration t. Starting with same equations as Lemma 1 we have:

$$f(w_{t+1}) \leq f(w_t) + \nabla f(w_t)^T (w_{t+1} - w_t) + \frac{1}{2} L \|w_{t+1} - w_t\|^2$$

$$f(w_{t+1}) \leq f(w_t) + \nabla f(w_t)^T (-\alpha \nabla f_t(w_t)) + \frac{1}{2} L \| - \alpha \nabla f_t(w_t)\|^2$$

Taking the expected value of both sides:

$$E[f(w_{t+1})] \leq E\left[f(w_t)\right] - \alpha E\left[\nabla f(w_t)^T \nabla f_t(w_t)\right] + \\ + E\left[\frac{\alpha^2 L}{2} \|\nabla f_t(w_t)\|^2\right]$$

$$E[f(w_{t+1})] \leq f(w_t) - \alpha \nabla f(w_t)^T E\left[\nabla f_t(w_t)\right] + \frac{\alpha^2 L}{2} E\left[\|\nabla f_t(w_t)\|^2\right]$$

Using the fact that $E[\nabla f_t(w_t)] = \nabla f(w_t)$:

$$E[f(w_{t+1})] \leq f(w_t) - \alpha \|\nabla f(w_t)\|^2 + \frac{\alpha^2 L}{2} E\left[\|\nabla f_t(w_t)\|^2\right]$$

Taking the derivative equal to zero similar with Lemma 1:

$$\frac{\partial \left(-\alpha \|\nabla f(w_t)\|^2 + \frac{\alpha^2 L}{2} E\left[\|\nabla f_t(w_t)\|^2\right]\right)}{\partial \alpha} = 0$$

$$-\|\nabla f(w_t)\|^2 + \alpha L E\left[\|\nabla f_t(w_t)\|^2\right] = 0$$

$$\alpha = \frac{\|\nabla f(w_t)\|^2}{L \cdot E\left[\|\nabla f_t(w_t)\|^2\right]}$$

□

Appendix B Model Architectures

The detailed architectures for the networks used in the experimental study are presented here.

Appendix C Extension: AdaLip-U

Here we'll introduce a variation of AdaLip, called AdaLip-U. This variation uses the fact that the magnitude of the update has similar behaviour with the magnitude of the gradient in

Table 6 Architecture for MNIST Model

Layer	Units	Padding	Activation
3 × 3 Conv	32	Valid	Relu
3 × 3 Conv	64	Valid	Relu
2 × 2 MaxPool	–	–	–
Flatten	–	–	–
Dense	128	–	Relu
Dense	10	–	Softmax

Table 7 Architecture for CIFAR10 Model

Layer	Units	Padding	Activation
3 × 3 Conv	32	Valid	Relu
3 × 3 Conv	32	Valid	Relu
2 × 2 MaxPool	–	–	–
3 × 3 Conv	64	Valid	Relu
3 × 3 Conv	64	Valid	Relu
2 × 2 MaxPool	–	–	–
3 × 3 Conv	128	Same	Relu
3 × 3 Conv	128	Same	Relu
2 × 2 MaxPool	–	–	–
Flatten	–	–	–
Dense	200	–	Relu
Dense	10	–	Softmax

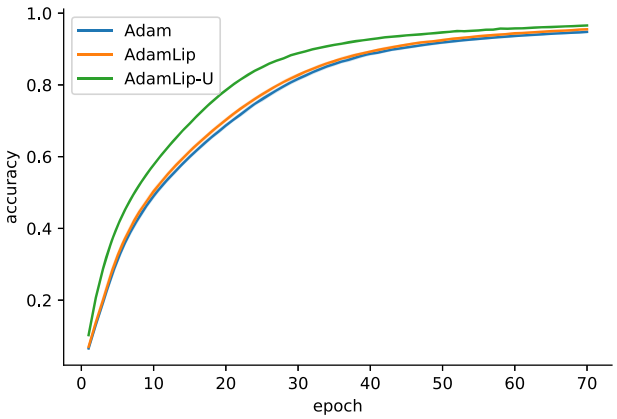

Fig. 11 AdamLip-U vs best Adam and AdamLip in Cifar100

GD. Continuing from Eq. 6 and assuming that $\alpha_{t-1} \approx \alpha_t$, we have:

$$\alpha^* \leq \alpha \frac{\|g_{t-1}\|}{\|g_t - g_{t-1}\|} = \alpha \frac{\|g_{t-1}\|}{\|g_t - g_{t-1}\|} \cdot \frac{\alpha_{t-1}}{\alpha_{t-1}}$$
$$= \alpha \frac{\|\alpha_{t-1} \cdot g_{t-1}\|}{\|\alpha_{t-1} \cdot g_t - \alpha_{t-1} \cdot g_{t-1}\|}$$

Table 8 Architecture for CIFAR100 Model

Layer	Units	Padding	Activation
3×3 Conv	64	Same	Relu
BatchNorm	–	–	–
2×2 MaxPool	–	–	–
3×3 Conv	128	Same	Relu
BatchNorm	–	–	–
2×2 MaxPool	–	–	–
3×3 Conv	256	Same	Relu
BatchNorm	–	–	–
3×3 Conv	256	Same	Relu
BatchNorm	–	–	–
2×2 MaxPool	–	–	–
3×3 Conv	512	Same	Relu
BatchNorm	–	–	–
3×3 Conv	512	Same	Relu
BatchNorm	–	–	–
2×2 MaxPool	–	–	–
3×3 Conv	512	Same	Relu
BatchNorm	–	–	–
3×3 Conv	512	Same	Relu
BatchNorm	–	–	–
2×2 MaxPool	–	–	–
Flatten	–	–	–
Dense	512	–	Relu
BatchNorm	–	–	–
Dropout(0.5)	–	–	–
Dense	100	–	Softmax

$$\approx \frac{\|\alpha_{t-1} \cdot g_{t-1}\|}{\|\alpha_t \cdot g_t - \alpha_{t-1} \cdot g_{t-1}\|}$$

This variation is based on the previous and current update of the optimizer, rather than its gradients. By denoting $u_t = -\alpha_t \cdot g_t$ the update of the optimizer, we have:

$$\alpha^* \leq \alpha \frac{\|u_{t-1}\|}{\|u_t - u_{t-1}\|} \tag{C4}$$

By using this as the batch's optimal learning rate, instead of Eq. 6, the learning rate of AdaLip-U can be derived:

$$A_t = \alpha_t \cdot S_t = \alpha_t \cdot \left[\gamma^t \cdot S_0 + (1-\gamma) \sum_{i=1}^{t} \gamma^{t-i} \frac{\|u_{i-1}\|}{\|u_i - u_{i-1}\| + c_i} \right] \tag{C5}$$

The previous update is stored from the previous iteration. The current update is the update the optimizer would compute without using AdaLip-U (i.e the update of SGD with the initial learning rate). Like previously with AdaLip, this new version generates three new

optimizers based on SGD, RMSProp and Adam, called AdaLip-U, RMSLip-U and AdamLip-U respectively. As shown in Fig. 11, AdamLip-U shows much potential in some cases, however it was much more unstable than its counterpart and more sensitive to the selection of the learning rate. In Algorithm 5, AdaLip-U is presented in detail.

Algorithm 5 AdaLip-U

Input: $w_t \in \mathcal{F}$, initial step α_0, $\gamma \in (0, 1)$, c_t
Initialize: $S_0 = 1$, $g_0 = 0$, $u_0 = 0$
for t=1 **to** T **do**
  $g_t = \nabla f_t(w_t)$
  $\alpha_t = \alpha_0/\sqrt{t}$
  $\hat{u}_t = -\alpha_t \cdot g_t$
  $S_t = \gamma \cdot S_{t-1} + (1-\gamma)\frac{\|u_{t-1}\|}{\|\hat{u}_t - u_{t-1}\| + c_t}$
  $A_t = \alpha_t \cdot S_t$
  $u_t = -A_t \cdot g_t$
  $\hat{w}_{t+1} = w_t + u_t$
end for

References

1. Litjens G, Kooi T, Bejnordi BE, Setio AAA, Ciompi F, Ghafoorian M, Van Der Laak JA, Van Ginneken B, Sánchez CI (2017) A survey on deep learning in medical image analysis. Med Image Anal 42:60–88
2. Minaee S, Boykov YY, Porikli F, Plaza AJ, Kehtarnavaz N, Terzopoulos D (2021) Image segmentation using deep learning: a survey. IEEE Trans Pattern Anal Mach Intell 2:668
3. Padmanabhan J, Johnson Premkumar MJ (2015) Machine learning in automatic speech recognition: a survey. IETE Tech Rev 32(4):240–251
4. Kumar A, Verma S, Mangla H(2018) A survey of deep learning techniques in speech recognition. In: 2018 international conference on advances in computing, communication control and networking (ICACCCN), pp 179– 185. IEEE
5. Yang S, Wang Y, Chu X (2020) A survey of deep learning techniques for neural machine translation. arXiv preprint arXiv:2002.07526
6. Grigorescu S, Trasnea B, Cocias T, Macesanu G (2020) A survey of deep learning techniques for autonomous driving. J Field Robot 37(3):362–386
7. Iqbal T, Qureshi S (2020) The survey: text generation models in deep learning. J King Saud Univ Comput Inf Sci 6:998
8. Loshchilov I, Hutter F (2016) Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983
9. Huang G, Li Y, Pleiss G, Liu Z, Hopcroft JE, Weinberger KQ (2017) Snapshot ensembles: Train 1, get m for free. arXiv preprint arXiv:1704.00109
10. Robbins H, Monro S (1951) A stochastic approximation method. Ann Math Stat 22(3):400–407. https://doi.org/10.1214/aoms/1177729586
11. Kleinberg R, Li Y, Yuan Y (2018) An alternative view: when does SGD escape local minima? In: Dy JG, Krause A (eds.) Proceedings of the 35th international conference on machine learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10–15. Proceedings of Machine Learning Research, vol 80, pp 2703–2712. PMLR. http://proceedings.mlr.press/v80/kleinberg18a.html
12. Smith LN (2017) Cyclical learning rates for training neural networks. In: 2017 IEEE winter conference on applications of computer vision, WACV 2017, Santa Rosa, CA, USA, March 24–31, pp 464–472. IEEE Computer Society. https://doi.org/10.1109/WACV.2017.58
13. Bottou L, Curtis FE, Nocedal J (2018) Optimization methods for large-scale machine learning. SIAM Rev 60(2):223–311
14. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770– 778
15. Chollet F (2017) Deep learning with python, 1st edn. Manning Publications Co., New York

16. Shamir O, Zhang T (2013) Stochastic gradient descent for non-smooth optimization: convergence results and optimal averaging schemes. In: International conference on machine learning, pp 71– 79
17. Zinkevich M (xxxx) Online convex programming and generalized infinitesimal gradient ascent. In: Proceedings of the twentieth international conference on international conference on machine learning. ICML'03, pp 928–935. AAAI Press
18. Wu X, Ward R, Bottou L(2018) Wngrad: learn the learning rate in gradient descent. CoRR abs/1803.02865. arXiv:1803.02865
19. Duchi JC, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res 12:2121–2159
20. Tieleman T, Hinton G (2012) Lecture 6.5–RmsProp: divide the gradient by a running average of its recent magnitude. COURSERA Neural Netw Mach Learn 2:58
21. Kingma DP, Ba J (2015) Adam: a method for stochastic optimization. In: Bengio Y, LeCun Y (eds.) 3rd International conference on learning representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings. arXiv:1412.6980
22. Wilson AC, Roelofs R, Stern M, Srebro N, Recht B (2017) The marginal value of adaptive gradient methods in machine learning. In: Advances in neural information processing systems, pp 4148– 4158
23. Reddi SJ, Kale S, Kumar S (2018) On the convergence of adam and beyond. In: 6th international conference on learning representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, Conference Track Proceedings. OpenReview.net. https://openreview.net/forum?id=ryQu7f-RZ
24. Luo L, Xiong Y, Liu Y, Sun X (2019) Adaptive gradient methods with dynamic bound of learning rate. In: 7th international conference on learning representations, ICLR 2019, New Orleans, LA, USA, May 6–9, 2019. OpenReview.net. https://openreview.net/forum?id=Bkg3g2R9FX
25. Yedida R, Saha S (2019) LipschitzLR: using theoretically computed adaptive learning rates for fast convergence
26. Fazlyab M, Robey A, Hassani H, Morari M, Pappas GJ (2019) Efficient and accurate estimation of lipschitz constants for deep neural networks. In: Wallach HM, Larochelle H, Beygelzimer A, d'Alché-Buc F, Fox EB, Garnett R (eds.) Advances in neural information processing systems 32: annual conference on neural information processing systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pp 11423– 11434. https://proceedings.neurips.cc/paper/2019/hash/95e1533eb1b20a97777749fb94fdb944-Abstract.html
27. Baydin AG, Cornish R, Martínez-Rubio D, Schmidt M, Wood F (2018) Online learning rate adaptation with hypergradient descent. In: 6th international conference on learning representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, Conference Track Proceedings. OpenReview.net. https://openreview.net/forum?id=BkrsAzWAb
28. Ioffe S, Szegedy C (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Bach FR, Blei DM (eds.) Proceedings of the 32nd international conference on machine learning, ICML 2015, Lille, France, 6–11 July. JMLR Workshop and Conference Proceedings, vol 37, pp 448–456. JMLR.org. http://proceedings.mlr.press/v37/ioffe15.html
29. Nair V, Hinton GE (2010) Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th international conference on machine learning (ICML-10), pp 807– 814
30. Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp 249–256
31. LeCun Y, Cortes C (2010) MNIST handwritten digit database
32. Krizhevsky A (2009) Learning multiple layers of features from tiny images
33. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition
34. Srivastava N, Hinton GE, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958
35. Choromanska A, LeCun Y, Arous GB (2015) Open problem: the landscape of the loss surfaces of multilayer networks. In: Grünwald P, Hazan E, Kale S (eds.) Proceedings of The 28th conference on learning theory, COLT 2015, Paris, France, July 3–6. JMLR Workshop and Conference Proceedings, vol 40, pp 1756–1760. JMLR.org. http://proceedings.mlr.press/v40/Choromanska15.html
36. Li H, Xu Z, Taylor G, Studer C, Goldstein T (2018) Visualizing the loss landscape of neural nets. In: Advances in neural information processing systems, pp 6389– 6399
37. Ge R, Huang F, Jin C, Yuan Y (2015) Escaping from saddle points-online stochastic gradient for tensor decomposition. In: Grünwald P, Hazan E, Kale S (eds.) Proceedings of The 28th conference on learning theory, COLT 2015, Paris, France, July 3–6, . JMLR workshop and conference proceedings, vol 40, pp 797–842. JMLR.org. http://proceedings.mlr.press/v40/Ge15.html
38. Jin C, Ge R, Netrapalli,P, Kakade SM, Jordan MI (2017) How to escape saddle points efficiently. In: Precup D, Teh YW (eds.) Proceedings of the 34th international conference on machine learning, ICML 2017,

Sydney, NSW, Australia, 6-11 August. Proceedings of machine learning research, vol 70, pp 1724–1732. PMLR. http://proceedings.mlr.press/v70/jin17a.html