BENCHMARKING ADAPTIVE METHODS FOR STOCHASTIC GRADIENT DESCENT

Xinhao Liu
NYU Shanghai

ABSTRACT

This report presents a comprehensive survey of adaptive stochastic gradient descent algorithms and benchmarks them on different datasets and model architectures. The use of gradient-based optimization methods to solve optimization problems is explained. The experiments conducted in this study compare the performance of different adaptive methods in stochastic gradient descent, including a special case where Adam fails to converge. However, we found that these methods work very well for generic cases, which are more common in reality. Our findings provide insights into the behavior of different optimization algorithms under different conditions, identify their limitations, and suggest best practices.

1 INTRODUCTION

Convex optimization is a significant problem in the field of statistics and numerical machine learning. In the context of parametric machine learning, optimization methods are often used to find the optimal parameters that best fit the data within a certain model set . Due to the complexity of modern machine learning models and data, gradient-based optimization methods are the prominent approach to solving optimization problems (Bottou et al., 2018). It is regarded as one of the fundamental problems in machine learning and optimization. It aims to find the optimal solution for a given optimization problem by minimizing a loss or objective function by iteratively updating the current parameters $\boldsymbol{x}_t$ according to the gradient $\nabla f(\boldsymbol{x})$ produced by the objective function f at $\boldsymbol{x}$, scaled by a step size α.

Stochastic gradient descent (SGD), as a special category of gradient-based optimization algorithm, is the most popular and widely-used genre of convex optimization methods to better fit modern computation devices and accommodate big data (Ruder, 2016). As shown in equation 1 (I^k), SGD selects a subset of the given data according to a uniform distribution so it can balance the trade-off between parallel computing and data efficiency.

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \nabla f_{I^k}\left(\boldsymbol{x}_k\right) \tag{1}$$

One of the largest problem in SGD is the choice of the hyperparameter step size α. It is shown that a large step size can induce the non-convergence of the optimization, while a small step size might cause the parameters to stuck in local optima (Nar & Sastry, 2018). Hence, the global convergence of SGD algorithms is largely dependent on the choice of α. There is no one-size-fits-all guideline for the choice a good step size. The choice is usually empirical and data-dependent, which is troublesome if we want to generalize an algorithm to solve different categories of problems.

In recent years, a series of adaptive methods (Duchi et al., 2011; Hinton et al., 2012; Kingma & Ba, 2014) are proposed aiming to adaptively find a good step size so that the parameters are updated with an appropriate learning rate and accelerate the convergence of the algorithm. There is also much debate on the analytical and empirical analysis of the convergence of these algorithms (Défossez et al., 2022; Reddi et al., 2019; De et al., 2019; Traoré & Pauwels, 2021; Shi et al., 2021). However, it is still unclear and debatable about the performance, convergence, and efficiency of these algorithms when they are applied to real problems.

This project conducts a comprehensive survey of the adaptive SGD algorithm and benchmark them on different datasets and model architectures. The report will provide insights into the behavior of

Algorithm 1 Generic Adaptive Method Setup (Reddi et al., 2019)

Input: $\boldsymbol{x}_1 \in \mathcal{F}$, step size $\{\alpha_t > 0\}_{t=1}^T$, sequence of functions $\{\phi_t, \psi_t\}_{t=1}^T$
for $t = 1$ **to** T **do**
  $\boldsymbol{g}_t = \nabla f_t(\boldsymbol{x}_t)$
  $\boldsymbol{m}_t = \phi_t(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t)$ and $\boldsymbol{V}_t = \psi_t(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t)$
  $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \alpha_t \boldsymbol{m}_t / \sqrt{\boldsymbol{V}_t}$
end for

different optimization algorithms under different conditions, identify their limitations, and suggest best practices for their use.

The importance of this benchmark lies in its potential to guide researchers and practitioners in the selection of the most suitable optimization algorithm for their specific problem. It can also help researchers in developing new and better optimization algorithms by identifying the limitations of existing ones.

2 UPDATE RULES

Generic update rule. (Reddi et al., 2019) proposes a generic framework that accounts for the update rules in different adaptive methods, as shown in algorithm 1. It provides a high-level abstraction of the idea of adaptive methods and is flexible to encapsulate different many popular adaptive methods. It is worth nothing that the function ϕ_t and ψ_t in the algorithm is an abstraction of the specific update rule used in different methods. In general, we observe that all adaptive methods first compute the gradient g_t of f_t at the current parameter $\boldsymbol{x}_t$. The next step features the main design of adaptive methods. They use two functions $\phi_t : \mathbb{R}^{t \times n} \mapsto \mathbb{R}^n$ and $\psi_t : \mathbb{R}^{t \times n} \mapsto \mathbb{R}^{n \times n}$ to compute $\boldsymbol{m}_t$ and $\boldsymbol{V}_t$, which controls the direction and norm of the update respectively. Finally, the current parameter $\boldsymbol{x}_t$ is updated by the learning rate $\alpha_t \boldsymbol{m}_t / \sqrt{\boldsymbol{V}_t}$. We want to make a remark that we will refer to α_t by "step size" and effective update $\alpha_t / \sqrt{\boldsymbol{V}_t}$ by "learning rate" hereafter. In the rest of this section, we will introduce and explain the specific update rules of different adaptive methods and their corresponding formulas.

Momentum SGD. The most trivial adaptive methods based on SGD is the so-called momentum method (Qian, 1999). The name "momentum" comes from the fact that the method physically resembles behavior of a falling object with a momentum. Momentum SGD computes a moving average between the previous updates and the current gradient. In the context of algorithm 1, for a hyperparameter $\beta \in [0, 1]$, we have ϕ_t designed as:

$$\phi_t = \beta \boldsymbol{m}_{t-1} + (1 - \beta) \boldsymbol{g}_t \quad \text{and} \quad \boldsymbol{V}_t = \boldsymbol{I}^{n \times n}, \tag{Momentum}$$

and $\alpha_t = \alpha / \sqrt{t}$ (required for convergence). In practice, we tend to set a very large value for β (0.9 or 0.99). By updating the current parameter by the momentum, momentum method is believed to have a better ability to escape from local optima and dampen the oscillation around the global optima, compared to SGD. This is because momentum method employs the previous update to redirect and rescale the current update. At a local optima, previous updates tend to be large so the parameter will be updated by a large norm and escape the region. On the other hand, at a global optimal point, the oscillation leads to opposite gradients and they cancel out due to the moving average.

AdaGrad. (Duchi et al., 2011) proposes AdagGrad based on SGD to addresses the challenges of choosing an appropriate learning rate for different parameters. It adapts the learning rate individually for each parameter by scaling it based on the historical gradient information. The intuition behind AdaGrad is to give smaller updates to frequently occurring parameters and larger updates to infrequently occurring parameters. In the context of algorithm 1, AdaGrad modifies the update rule by introducing a diagonal matrix which accumulates the sum of squares of the gradients:

$$\phi_t \left(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t \right) = \boldsymbol{g}_t \quad \text{and} \quad \psi_t \left(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t \right) = \frac{\operatorname{diag} \left(\sum_{i=1}^t \boldsymbol{g}_i^2 \right)}{t}, \tag{2}$$

and $\alpha_t = \alpha / \sqrt{t}$. Note that in contrast to the same $\alpha_t = \alpha / \sqrt{t}$ in momentum SGD, this update rule of α_t aims to imply a modest learning rate decay of $\alpha / \sqrt{\sum_{i=1}^t \boldsymbol{g}_{ij}^2})$ for each dimension j. In other

words, the effective learning rate $\alpha_t/\sqrt{\boldsymbol{V}_t} = \alpha/\sqrt{(\sum_{i=1}^t \boldsymbol{g}_i^2)}$. This design helps AdaGrad perform very well especially in the case of sparse gradient, i.e., where only a small subset of its elements are non-zero, while the rest are zero or close to zero. However, one limitation of AdaGrad is that $\boldsymbol{V}_t$ keeps increasing during training, which can eventually lead to very small learning rates and slow down the learning process.

RMSProp. Root Mean Square Propagation (RMSProp) is another adaptive optimization algorithm proposed by (Hinton et al., 2012) that addresses some of the limitations of AdaGrad. To prevent $\boldsymbol{V}_t$ from growing too fast and vanishing the learning rate, RMSProp updates it with the moving average of the previous $\boldsymbol{V}_t$. With a hyperparameter similar to momentum SGD, the update rule of RMSProp is the following:

$$\phi_t\left(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t\right) = \boldsymbol{g}_t \quad \text{and} \quad \psi_t\left(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t\right) = \beta \boldsymbol{V}_{t-1} + (1-\beta)\operatorname{diag}\left(\sum_{i=1}^t \boldsymbol{g}_i^2\right). \tag{3}$$

By this design of ψ_t, RMSProp addresses the issue of continually increasing sum of squares in AdaGrad by using a decaying average of the squared gradients. This allows the algorithm to give more importance to recent gradients and effectively adapt the learning rate. By considering only a window of past gradients, RMSProp can prevent the learning rate from becoming too small and facilitate faster convergence.

Adam. Adaptive Moment Estimation (Adam) is an adaptive optimization algorithm that combines the concepts of momentum and RMSProp. It is introduced in (Kingma & Ba, 2014) and has gained popularity for its robust performance in a wide range of deep learning tasks. Adam maintains two moving average estimations: the first moment estimate (mean) and the second moment estimate (variance) of the gradients. With two hyperparameters β_1, β_2, the update rule of Adam is as follows:

$$\begin{aligned}
&\phi_t\left(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t\right) = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t, \\
&\psi_t\left(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t\right) = \beta_2 \boldsymbol{V}_{t-1} + (1-\beta_2)\operatorname{diag}\left(\sum_{i=1}^t \boldsymbol{g}_i^2\right).
\end{aligned} \tag{4}$$

We can also expand out the recursion to get a compact form that is better for later convergence analysis:

$$\begin{aligned}
&\phi_t\left(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t\right) = (1-\beta_1)\sum_{i=1}^t \beta_1^{t-i}\boldsymbol{g}_i, \\
&\psi_t\left(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t\right) = (1-\beta_2)\operatorname{diag}\left(\sum_{i=1}^t \beta_2^{t-i}\boldsymbol{g}_i^2\right).
\end{aligned} \tag{5}$$

(Kingma & Ba, 2014) argues that Adam combines the benefits of momentum SGD and RMSProp. The momentum term $\boldsymbol{m}_t$ helps Adam accelerate convergence and navigate flatter regions of the loss landscape, similar to momentum SGD. The second moment estimate $\boldsymbol{V}_t$ acts as an adaptive learning rate, scaling the updates based on the historical gradient variances. The work also mentions a correction term to $\boldsymbol{m}$ and $\boldsymbol{V}$ that corrects the bias introduced when $\boldsymbol{m}$ and $\boldsymbol{V}$ are initialized as zero, though this correction term is out of the scope of this generic framework. In general, Adam shows excellent performance in various deep learning applications, providing fast convergence and robustness to different types of data and network architectures.

However, it is pointed out in (Reddi et al., 2019) that the convergence proof in (Kingma & Ba, 2014) is problematic, and counterexample can be provided to provably show the non-convergence of Adam. We will first introduce the algorithm proposed by (Reddi et al., 2019) and discuss about convergence in section 3.

AMSGrad. AMSGrad is a modification of Adam that addresses a limitation of Adam related to the biased estimation of the second moment. Proposed by (Reddi et al., 2019), AMSGrad aims to provide a more stable and reliable estimation of the second moment to prevent the issue of increasing variance estimates. The update rule of AMSGrad is similar to Adam, with a modification in the

calculation of the second moment:

$$\begin{aligned}\phi_t\left(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t\right) &= (1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}\boldsymbol{g}_i, \\ \psi_t\left(\boldsymbol{g}_1, \ldots, \boldsymbol{g}_t\right) &= \max\left[\boldsymbol{V}_{t-1}, (1-\beta_2)\operatorname{diag}\left(\sum_{i=1}^{t}\beta_2^{t-i}\boldsymbol{g}_i^2\right)\right].\end{aligned} \tag{6}$$

In AMSGrad, instead of using a simple moving average of the squared gradients, the maximum of the historical second moments and the current second moment estimate is taken. This modification prevents the unbounded growth of the variance estimate, which can occur in Adam and potentially lead to slower convergence or performance degradation. The remaining steps of bias correction and parameter update in AMSGrad follow the same procedure as in Adam.

By incorporating the max operation, AMSGrad ensures that the second moment estimate is always non-decreasing, effectively bounding the learning rate and providing more stability during optimization. This modification has shown improved convergence properties and better generalization performance in some cases where Adam might suffer from poor convergence due to biased estimates. However, it's worth noting that AMSGrad may be slightly more computationally expensive than Adam due to the additional max operation. Overall, AMSGrad is a useful variant of Adam that addresses the issue of increasing variance estimates, making it a valuable optimization algorithm for deep learning tasks.

3 CONVERGENCE ANALYSIS

Now that we have introduced several popularly-used optimization methods, it is vital to verify the convergence of these algorithms analytically. Unfortunately, the nature of the problem is highly dependent on the data distribution and model architecture, and it is impossible to find a specific task that fairly evaluates different methods. We will mainly rely on the conventional online learning framework (Zinkevich, 2003) for convergence analysis. The framework largely resembles the setting of stochastic gradient descent. At each time $t = 0, 1, \cdots, T$, the optimizer is given an arbitrary convex function $f_t(\boldsymbol{x})$ (possibly not drawn from a uniform distribution) and the parameter at the previous time step $\boldsymbol{x}_{t-1}$ as input. The output is an updated parameter $\boldsymbol{x}_t$. The goal is to find a global optimal point $\boldsymbol{x}^* := \arg\min \sum_{i=1}^{T} f_t(\boldsymbol{x})$ that minimizes the sum of the cost at each time step. We evaluate the performance of the optimizer by the regret defined by:

$$R(T) = \sum_{i=1}^{T}[f_t(\boldsymbol{x}_t) - f_t(\boldsymbol{x}^*)]. \tag{7}$$

3.1 REGRET BOUND

In (Kingma & Ba, 2014), the authors argue that Adam has a regret bound of $\mathcal{O}(\sqrt{T})$. In other words, we have

$$\lim_{T\to\infty}\frac{R(T)}{T} = \lim_{T\to\infty}\mathcal{O}\left(\frac{1}{\sqrt{T}}\right) = 0. \tag{8}$$

The authors give proof of this argument in the appendix of the paper. However, it is recently pointed out by several other papers(Défossez et al., 2022; Reddi et al., 2019) that this argument is wrong because counterexamples can be given to disprove the argument.

(Reddi et al., 2019) gives a very trivial example where Adam fails to converge to the optimal point. Actually, Adam converges to the worst point that contributes to the largest loss. Suppose the feasible domain for $\boldsymbol{x}$ is $\mathcal{F} = [-1, 1]$ and constant $C > 2$. Let the functions be defined as:

$$f_t(x) = \begin{cases} Cx, & \text{for } t \bmod 3 = 1 \\ -x, & \text{otherwise} \end{cases}. \tag{9}$$

We can observe that $x^* = -1$ because it contributes to the lowest loss in one period (three steps) and hence the long run. Suppose we choose the hyperparameters in Adam as $\beta_1 = 0$ and $\beta_2 = \frac{1}{1+C^2}$, it is shown that Adam (and RMSProp) converges to the worst suboptimal point $x = 1$ under this

setting. We omit the proof here and refer readers to the appendix in (Reddi et al., 2019) for more details. The authors explain that the intuition is that the large gradient C is unable to counteract the -1 gradients since it is scaled down by a factor of almost C for the given value of β_2.

We want to make a few remarks to this example provided in Reddi et al. (2019). First, the example includes a periodic sequence of functions to optimize. This is very practical especially considering the case when we want to train a model iteratively using the data for several epochs. Although in reality, we usually shuffle the data or draw the data from a uniform random distribution, we will show later in section 4 that it also contributes to the non-convergence of Adam.

Second, we want to point out that equation 4 and equation 5 shows that Adam is the same as RMSProp when $\beta_1 = 0$. Although we understand that (Reddi et al., 2019) aims to use this example to show the case when the scaling term absorbs the effect of a large but significant gradient, it is somewhat unfair because the example essentially shows the pitfall of RMSProp, but not Adam.

Third, in correspondence to the second remark, (Reddi et al., 2019) actually also argues that there always exists a (stochastic) online optimization problem where Adam fails to converge to the optimal point, as long as we have $\beta_1 < \sqrt{\beta_2}$, which is a more reasonable assumption to make because (Kingma & Ba, 2014) also uses a similar assumption in their proof. Moreover, the default setting of Adam (for example, $\beta_1 = 0.9, \beta_2 = 0.99$ in PyTorch) usually falls into this assumption.

Furthermore, (Reddi et al., 2019) shows that AMSGrad provably has a regret bound of $\mathcal{O}(\sqrt{T})$ and hence also have the same convergence rate shown in equation 8. We again omit the proof here and have to admit that we are unsure about the correctness of the proof.

3.2 Change in learning rate

(Reddi et al., 2019) also provides another tool to evaluate different adaptive methods qualitatively and intuitively. The authors defines the change in the inverse of learning rate as

$$\Gamma_{t+1} = \left(\frac{\sqrt{V_{t+1}}}{\alpha_{t+1}} - \frac{\sqrt{V_t}}{\alpha_t} \right). \tag{10}$$

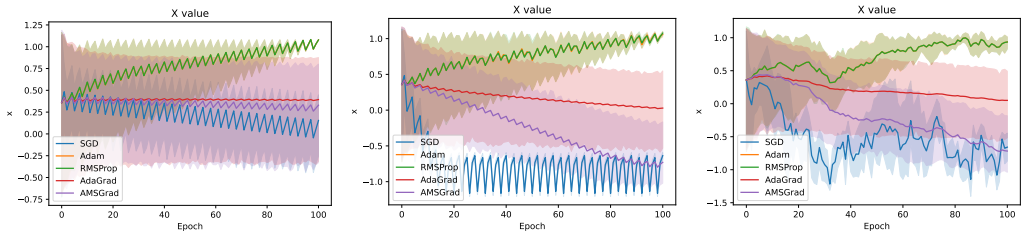
Intuitively, we want to keep Γ_{t+1} positive, i.e., keep the effective learning rate decreasing along the training. It is observed that this property hold for momentum SGD, AdaGrad, and AMSGrad, but is potentially indefinite for RMSProp and Adam. This can also give an intuitive explanation why RMSProp and Adam may fail under certain circumstances.

4 Experiments

In this section, we will mainly show three categories of experiments. One of them follows the counterexample provided in section 3.1 by (Reddi et al., 2019). The other two will cover some commonly used datasets and machine learning models. We will run the adaptive methods motioned above in section 2 and compare how the loss decreases with respect to training iterations. We do not fine-tune the hyperparameters for these methods because we do not care what is the minimum loss achieved by these methods. Instead, we are interested in how these methods behave under the same set of hyperparameters. This also aligns with the ultimate purpose of the design of adaptive methods.

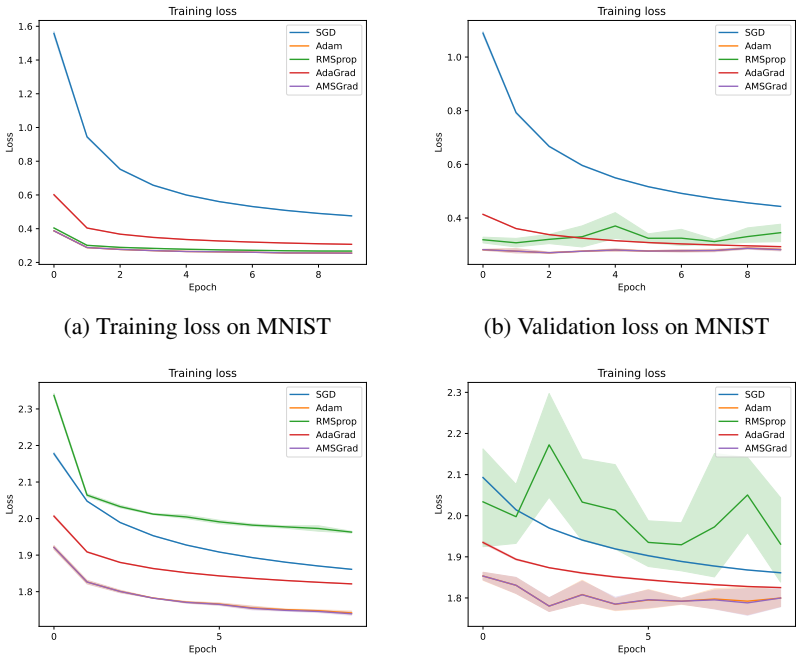
4.1 Special case

In this experiment, we implement the function as described by equation 9. We set $\beta_1 = 0, \beta_2 = \frac{1}{1+C^2}$ for Adam and similarly $\beta = \frac{1}{1+C^2}$ for RMSProp. The learning constant step size α is set to 0.1. All other hyperparameters are the default in PyTorch. Figure 1 shows the experiment result with different settings. We can observe that RMSProp and Adam converges to suboptimal point $x = 1$, while other adaptive methods are able to move the parameter towards $x = -1$. This confirms the claim made in (Reddi et al., 2019) and clearly shows the simple yet effective improvement in AMSGrad compared to Adam. Comparing figures 1a and 1b, we show that the non-convergence of RMSProp and Adam does not depend on the choice of C as long as we have $C > 2$. Figure 1c shows that this example also applies to the stochastic case where the sequence of functions does not



(a) $C = 2.1$ (b) $C = 5$ (c) Stochastic ($C = 5$)

Figure 1: **Experiment on special case**. We did experiment on the special case with two different C values. We also extend the example into a stochastic version where the two cases are chosen based on a random sampling. All experiments are done with 5 random seeds (5 different random initialization of x. Note that RMSProp and Adam overlap for all three subfigures due to the settings.



(a) Training loss on MNIST (b) Validation loss on MNIST

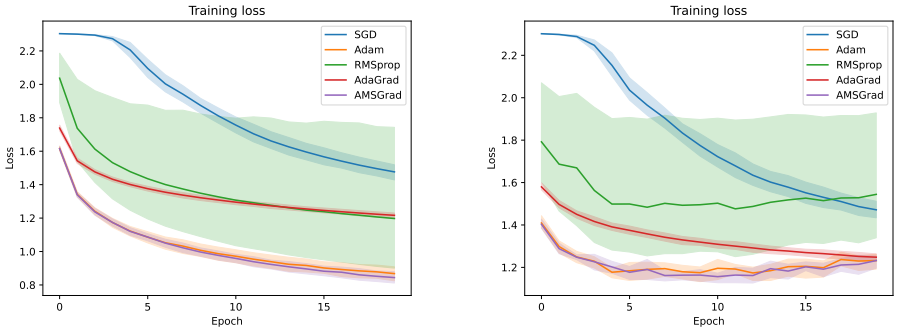
(c) Training loss on CIFAR-10 (d) Validation loss on CIFAR-10

Figure 2: **Experiment for linear regression**. We show the training and validation loss of linear regression on both the MNIST and CIFAR-10 dataset. All experiments are done with 5 random seeds (5 different random initialization of parameters).

behave necessarily periodically but asymptotically, which is closer to the reality in the practice of SGD.

4.2 Linear regression

Linear regression is a simple yet widely used algorithm. We implement linear regression by the linear layer in PyTorch. We fit the model on the MNIST and CIFAR-10 (Krizhevsky et al., 2009) dataset. We simply flatten the images and treat them as n-dimensional vectors and do linear regression. Figure 2 shows the result of the experiments. We can observe that Adam and AMSGrad generally has the best performance compared to other adaptive methods and there is no observable difference between them. One interesting phenomenon to notice is that all methods has small variance in training loss, but RMSProp has very large variance in validation loss. Adam and AMSGrad also has a considerable validation loss variance on CIFAR-10. One speculation of the reason is that



(a) Training loss (b) Validation loss

Figure 3: **Experiment for CNN**. We show the training and validation loss of convolutional network on the CIFAR-10 dataset. All experiments are done with 5 random seeds (5 different random initialization of parameters).

different optimizers have different "perfered" convergence region. It is possible that some region has low loss for the training set but have large loss for the validation set.

4.3 Convolutional networks

The last experiment is to optimize parameters in a convolutional neural network (CNN) on the CIFAR-10 dataset (Krizhevsky et al., 2009). We can observe from figure 3 that the behavior of these adaptive methods almost resembles that in figure 2. It is noticeable that RMSProp has even larger variance on both the training and validation dataset. This might be caused by the complexity of the model and the distribution of dataset. Another interesting observation from Figure 3b is that all methods suffer from overfitting more or less. It would be another interesting topic to investigate how these methods behaves around the optimal point. This is very important because it can verify the effective ness of the decreasing learning rate.

4.4 Summary

To summarize this section. We want to make a very import remark. In this section, we conduct three categories of experiments to compare the performance of different adaptive methods in stochastic gradient descent. One of these categories followed a counterexample provided by (Reddi et al., 2019), which showed that Adam fails to converge in a special case where the loss function has a certain structure. However, we found that Adam and other adaptive methods work very well for generic cases, which are more common in reality. This may be related to the statistical distribution of data, as the distribution in the special case is not very common. Our experiments demonstrate the importance of testing optimization algorithms on a variety of datasets and model architectures to gain insights into their behavior under different conditions.

5 Conclusion

In this report, we conduct a comprehensive survey of adaptive stochastic gradient descent algorithms and benchmark them on different datasets and model architectures. Our findings provide insights into the behavior of different optimization algorithms under different conditions, identify their limitations, and suggest best practices for their use. We believe the report has a potential to guide researchers and practitioners in the selection of the most suitable optimization algorithm for their specific problem. It can also help researchers in developing new and better optimization algorithms by identifying the limitations of existing ones. We hope that our findings will contribute to the development of more efficient and effective optimization algorithms for machine learning applications.

REFERENCES

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM review*, 60(2):223–311, 2018.

Soham De, Anirbit Mukherjee, and Enayat Ullah. Convergence guarantees for rmsprop and adam in non-convex optimization and an empirical comparison to nesterov acceleration. In *ICLR*, 2019.

Alexandre Défossez, Leon Bottou, Francis Bach, and Nicolas Usunier. A simple convergence proof of adam and adagrad. *TMLR*, 2022.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12(7), 2011.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Kamil Nar and Shankar Sastry. Step size matters in deep learning. *Advances in Neural Information Processing Systems*, 31, 2018.

Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12 (1):145–151, 1999.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Naichen Shi, Dawei Li, Mingyi Hong, and Ruoyu Sun. Rmsprop converges with proper hyper-parameter. In *ICLR*, 2021.

Cheik Traoré and Edouard Pauwels. Sequential convergence of adagrad algorithm for smooth convex optimization. *Operations Research Letters*, 49(4):452–458, 2021.

Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th international conference on machine learning (icml-03)*, pp. 928–936, 2003.