AdaLA: Adapting Gradient Estimation by Looking Ahead

Sharan Ramjee
sramjee@stanford.edu
Department of Computer Science
Stanford University

Abstract

First-order gradient methods used for the optimization of neural networks broadly fall into two categories: stochastic gradient descent (SGD) methods and adaptive learning rate methods. While SGD methods lead to better generalization performance on large datasets, adaptive learning rate methods lead to faster convergence in early training phases. In order to address these issues, Zhuang *et al.*(1) proposed the AdaBelief optimizer, which adapts the step sizes through the use of an Exponential Moving Average (EMA) of the gradients to quantify the "belief" in the step size to be taken. AdaBelief, however, assumes a smooth parameter space that does not change much in terms of the gradients, which, in turn, justifies the use of the EMA of the gradients to "predict" the next gradients. However, this assumption does not always hold in complex parameter spaces as the gradients could drastically change at every iteration of the optimizer, in which case, the EMA of the gradients is not a good approximation of the next gradients. In order to address this issue, this paper proposes AdaLA: Adaptive Gradient Estimation by Looking Ahead, which is simple modification to AdaBelief that looks ahead to better adapt step sizes. Through extensive experiments, this paper demonstrates that AdaLA leads to better convergence across a variety of Deep Learning tasks, datasets, and models. The implementation of AdaLA can be found on GitHub: `https://github.com/sharanramjee/adala-optimizer`.

1 Introduction

Given the enormous dimensionality of the parameter space, state-of-the-art neural networks are typically trained with first-order gradient methods (2), which can be categorized into two types of methods: stochastic gradient descent (SGD) methods (3; 4; 5; 6) and adaptive learning rate methods (7; 8; 9; 10). While SGD methods lead to better generalization performance on large datasets, adaptive learning rate methods lead to faster convergence in early training phases (11; 12). Furthermore, adaptive learning rate methods have been empirically found to be unstable when training Generative Adversarial Networks (GANs) (13).

In order to address the above issues, Zhuang *et al.*(1) proposed the AdaBelief optimizer, which is a modification to Adam. In contrast to Adam, AdaBelief views m_t, which is the exponential moving average (EMA) of the gradients g_t as the prediction of the next gradients. Intuitively, if g_t deviates significantly from m_t, then a small step is taken as we have a weak belief in g_t. However, if g_t is close to the prediction m_t, then a large step is taken as we have a strong belief in g_t. Here, $v_t = g_t^2$ and $s_t = (g_t - m_t)^2$ are the EMAs of g_t^2 and $(g_t - m_t)^2$, respectively. As observed in the next section, while m_t is divided by $\sqrt{v_t}$ in Adam, it is divided by $\sqrt{s_t}$ in AdaBelief, where $\frac{1}{\sqrt{s_t}}$ is viewed as the "belief" in the observation. AdaBelief is a variant of Adam that modifies it without extra parameters and had three advantages as examined by Zhuang *et al.*(1): (1) Fast convergence as in other adaptive learning rate methods, (2) Good generalization as in SGD methods, and (3) Training stability in complex settings such as GANs.

AdaBelief assumes a smooth parameter space that does not change much in terms of the gradients. As such, the EMA of the gradient m_t can be used to "predict" the next gradient g_t. However, this assumption does not always hold in complex parameter spaces as the gradients could drastically change at every iteration of the optimizer, in which case, m_t is not a good approximation of the gradient. In order to address this issue, this paper proposes AdaLA: Adaptive Gradient Estimation by Looking Ahead.

2 Method

Similar to AdaBelief, AdaLA makes a simple modification to Adam that does not make use of any extra parameters. AdaLA takes a step in the direction of steepest descent (similar to Adam), computes the gradient at the next step, and uses it to adapt the learning rate, similar to the way AdaBelief uses the EMA of the gradient instead of the actual gradient at the next step. The pseudo-code of Adam, AdaBelief, and AdaLA are given below (modifications to Adam highlighted in *blue*). By the convention used in (10), the notations are as follows:

- $f(\theta) \in \mathbb{R}, \theta \in \mathbb{R}^d$: f is the loss function to minimize, θ is the parameter in $\mathbb{R}^d$
- $\prod_{\mathcal{F},M}(y) = \text{argmin}_{x\in\mathcal{F}} ||M^{1/2}(x-y)||$: projection of y onto a convex feasible set $\mathcal{F}$
- g_t: the gradient at step t
- m_t: exponential moving average (EMA) of g_t
- v_t, s_t: v_t is the EMA of g_t^2, s_t is the EMA of error in the predicted gradient
- α, ϵ: α is the learning rate, default is 10^{-3}; ϵ is a small number, typically set as 10^{-8}
- β_1, β_2: smoothing parameters, typical values are $\beta_1 = 0.9, \beta_2 = 0.999$
- β_{1t}, β_{2t} are the momentum for m_t and v_t respectively at step t, and typically set as constant (e.g. $\beta_{1t} = \beta_1, \beta_{2t} = \beta_2, \forall t \in \{1, 2, ...T\}$

Algorithm 1: Adam Optimizer

```
Initialize $\theta_0$, $m_0 \leftarrow 0$ , $v_0 \leftarrow 0$, $t \leftarrow 0$
While $\theta_t$ not converged
    $t \leftarrow t+1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
    $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
    $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
    Bias Correction
        $\widehat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$, $\widehat{v_t} \leftarrow \frac{v_t}{1-\beta_2^t}$
    Update
        $\theta_t \leftarrow \prod_{\mathcal{F},\sqrt{\widehat{v_t}}} \left(\theta_{t-1} - \frac{\alpha \widehat{m_t}}{\sqrt{\widehat{v_t}}+\epsilon}\right)$
```

Algorithm 2: AdaBelief Optimizer

```
Initialize $\theta_0$, $m_0 \leftarrow 0$ , $s_0 \leftarrow 0$, $t \leftarrow 0$
While $\theta_t$ not converged
    $t \leftarrow t+1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
    $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
    $s_t \leftarrow \beta_2 s_{t-1} + (1-\beta_2)(g_t - m_t)^2 + \epsilon$
    Bias Correction
        $\widehat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$, $\widehat{s_t} \leftarrow \frac{s_t}{1-\beta_2^t}$
    Update
        $\theta_t \leftarrow \prod_{\mathcal{F},\sqrt{\widehat{s_t}}} \left(\theta_{t-1} - \frac{\alpha \widehat{m_t}}{\sqrt{\widehat{s_t}}+\epsilon}\right)$
```

Algorithm 3: AdaLA Optimizer

```
Initialize $\theta_0$, $m_0 \leftarrow 0$ , $s_0 \leftarrow 0$, $t \leftarrow 0$
While $\theta_t$ not converged
    $t \leftarrow t+1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
    $g_{t+1} \leftarrow \nabla_\theta f_{t+1}(\theta_t)$
    $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
    $s_t \leftarrow \beta_2 s_{t-1} + (1-\beta_2)(g_t - g_{t+1})^2 + \epsilon$
    Bias Correction
        $\widehat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$, $\widehat{s_t} \leftarrow \frac{s_t}{1-\beta_2^t}$
    Update
        $\theta_t \leftarrow \prod_{\mathcal{F},\sqrt{\widehat{s_t}}} \left(\theta_{t-1} - \frac{\alpha \widehat{m_t}}{\sqrt{\widehat{s_t}}+\epsilon}\right)$
```

3 Experimental Setup

In accordance to the experiments conducted by Zhuang *et al.*(1), the performance of AdaLA in comparison to the other optimizers when it comes to optimizing neural networks will be performed on two datasets corresponding to three sets of tasks, each with their own evaluation models and metric for quantitative evaluation as given in table 1. For qualitative evaluation, the test accuracy curves, test perplexity curves, and sample image generations will be reported for the image classification, language modeling, and image generation tasks, respectively. For quantitative evaluation, the best test accuracy, best test perplexity, and best Frechet Inception Distance (FID) (14) will be reported for the image classification, language modeling tasks, and image generation tasks, respectively.

Table 1: Experimental Setup

Task	Dataset	Evaluation Models	Evaluation Metric
Image Classification	CIFAR-10	VGG11, ResNet34, DenseNet121	Test set accuracy
Language Modeling	Penn TreeBank	1-layer, 2-layer, 3-layer LSTMs	Test set perplexity
Image Generation	CIFAR-10	WGAN, WGAN-GP	Frechet Inception Distance

4 State-of-the-Art Optimizers

The performance of AdaLA, in accordance to the experimental setup outlined in the next section, will be compared to the following state-of-the-art optimizers: AdaBelief (1), SGD (5), AdaBound (15), Yogi (16), Adam (10), MSVAG (17), RAdam (18), AdamW (19), and Fromage (20).

Extensive hyperparameter tuning was performed for each of the state-of-the-art optimizers across the experiments. The hyperparameters for AdaLA were chosen to be the same as those of Adam as they led to better convergence empirically. Furthermore, in order to improve convergence near the minima, a weight decay of 5×10^{-4} was used across all the optimizers. In accordance to the tuning performed by Zhuang *et al.*(1), the following hyperparameters were chosen for the image classification and language modeling tasks:

AdaLA, AdaBelief, Yogi, Adam, RAdam, AdamW The default parameters of Adam were used: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $\alpha = 10^{-3}$.

SGD and Fromage Learning rates of 0.1 and 0.01 were used for SGD and Fromage, respectively. Additionally, a momentum of 0.9, which is the default for networks such as ResNet (21) and DenseNet (22) was used. The rest of the hyperparameters were set to the Adam defaults: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 0.001$.

AdaBound Learning rates of 0.1 and 0.001 were used for the initial and final learning rates, respectively. The rest of the hyperparameters were set to the Adam defaults: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\gamma = 0.001$.

MSVAG A learning rate of 0.1 was used. The rest of the hyperparameters were set to the Adam defaults: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 0.001$.

For the image generation task, a learning rate of 2×10^{-4} was chosen for all the optimizers. Furthermore, in accordance to the improved empirical results obtained by Zhuang *et al.*(1) using the following hyperparameters for AdaBelief, the same hyperparameters were set for AdaLA during the WGAN and WGAN-GP training: $\beta_1 = 0.5$ and $\epsilon = 10^{-12}$.

5 Results

In accordance to the analysis done by Zhuang *et al.*(1) across the three tasks, for each optimizer, a search is performed for the optimal hyperparameters. The mean and standard deviation of the test set accuracy (under optimal hyperparameters) for three runs with random initialization are then reported.

5.1 Image Classification

The image generation performance of AdaLA in comparison to the other state-of-the-art optimizers is compared on the CIFAR-10 dataset (23) using the VGG11 (24), ResNet34 (21), and DenseNet121 (22) models. The qualitative results are given in Fig. 1 and the quantitative results are given in Table. 2.

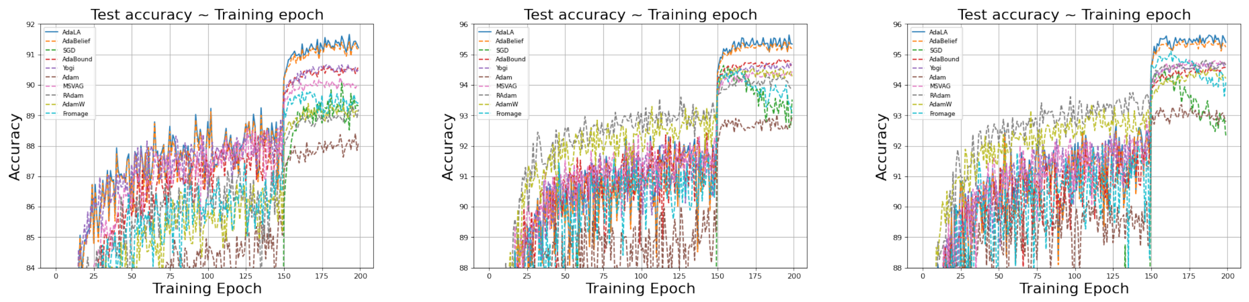


Figure 1: Test accuracy ($[\mu \pm \sigma]$) on the CIFAR-10 dataset. Visualization code modified from AdaBound (15).

Table 2: Best test accuracy (higher is better) on the CIFAR-10 dataset

Optimizer	VGG11	ResNet34	DenseNet121
Fromage	89.77%	94.51%	95.10%
AdamW	89.39%	94.59%	94.55%
RAdam	89.30%	94.33%	94.81%
MSVAG	90.24%	94.44%	94.81%
Adam	88.40%	93.02%	93.35%
Yogi	90.67%	94.71%	94.76%
AdaBound	90.62%	94.85%	94.58%
SGD	90.11%	94.64%	94.50%
AdaBelief	91.41%	95.44%	95.51%
AdaLA	**91.66%**	**95.64%**	**95.63%**

5.2 Language Modeling

The language modeling performance of AdaLA in comparison to the other state-of-the-art optimizers is compared on the Penn TreeBank dataset (25) using the 1-layer, 2-layer, and 3-layer LSTM models (26). The qualitative results are given in Fig. 2 and the quantitative results are given in Table. 3.

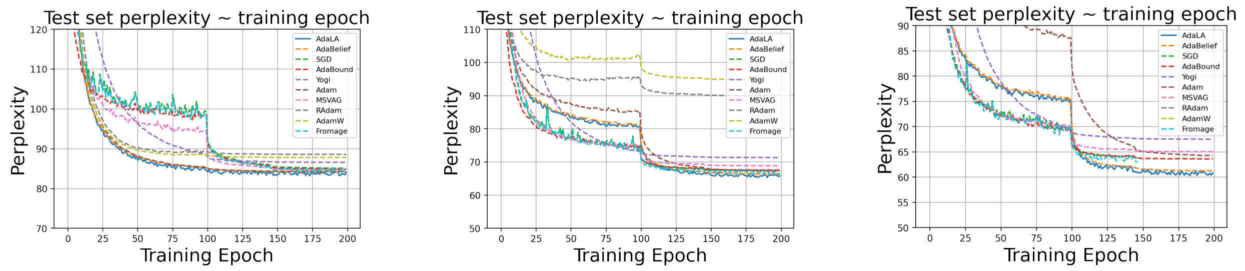


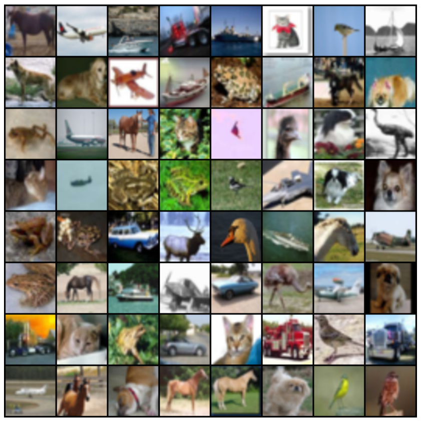
Figure 2: Test perplexity ($[\mu \pm \sigma]$) on the Penn TreeBank dataset.

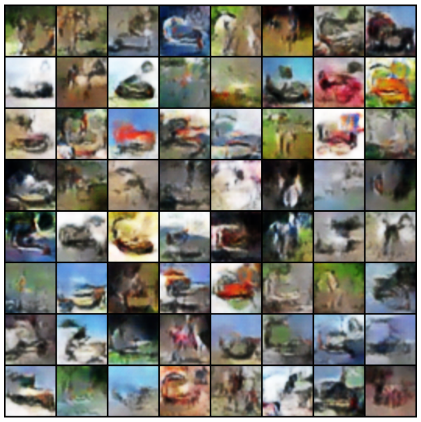
Table 3: Best test perplexity (lower is better) on the Penn TreeBank dataset

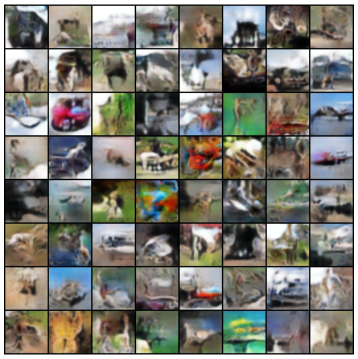
Optimizer	1-layer LSTM	2-layer LSTM	3-layer LSTM
Fromage	84.07	66.43	63.04
AdamW	87.80	94.86	104.49
RAdam	88.57	90.00	93.12
MSVAG	84.68	68.83	65.03
Adam	84.28	67.27	64.28
Yogi	86.59	71.33	67.51
AdaBound	84.78	67.53	63.58
SGD	85.07	67.42	63.77
AdaBelief	84.21	66.29	61.23
AdaLA	**83.24**	**65.36**	**60.33**

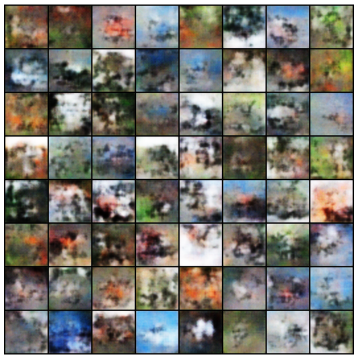
5.3 Image Generation

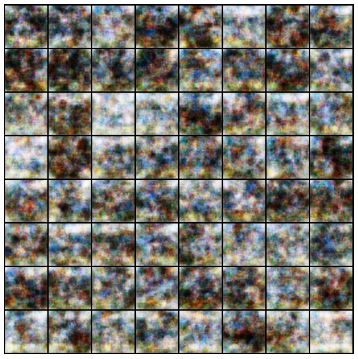
The image classification performance of AdaLA in comparison to the other state-of-the-art optimizers is compared on the CIFAR-10 dataset (23) using the Wasserstein-Generative Adversarial Network (WGAN) (27) and the improved version with gradient penalty (WGAN-GP) (28) with a vanilla CNN generator. Each optimizer is used to train the model for 100 epochs, generate 64,000 fake images (60,000 real images in CIFAR-10) from noise. The Frechet Inception Distance (FID) (14) is then computed between the fake and real images in order to quantitatively assess (lower FID is better) the quality and diversity of the generated images. The qualitative results are given in Figs. 3 and 4 while the quantitative results are given in Table. 4.

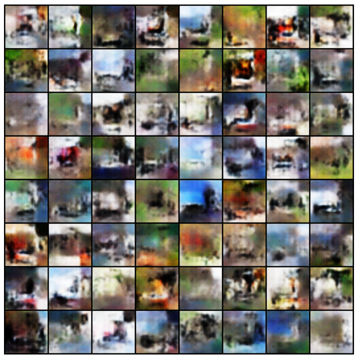

(a) Real samples

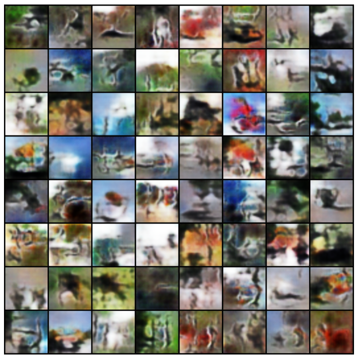

(b) AdaLA

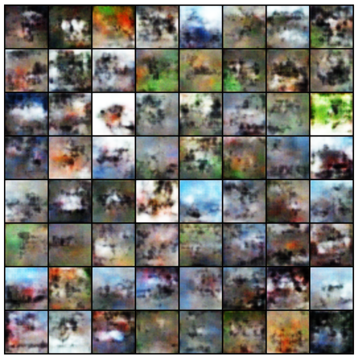

(c) AdaBelief

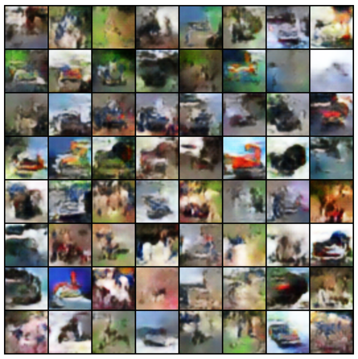

(d) SGD

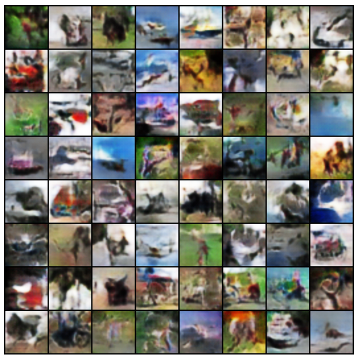

(e) AdaBound


(f) Yogi


(g) Adam


(h) MSVAG


(i) RAdam


(j) AdamW

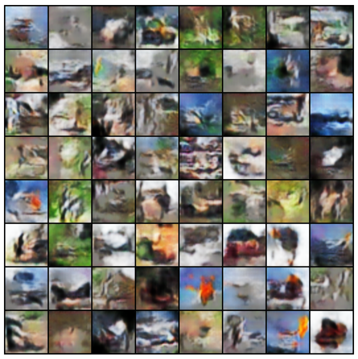
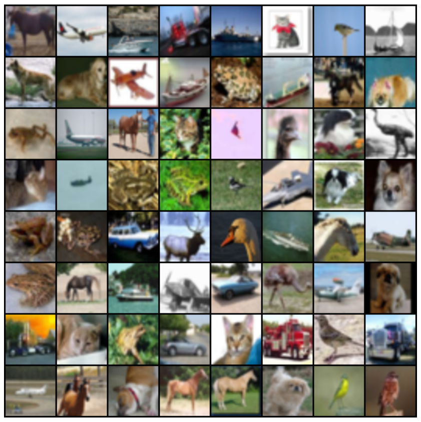
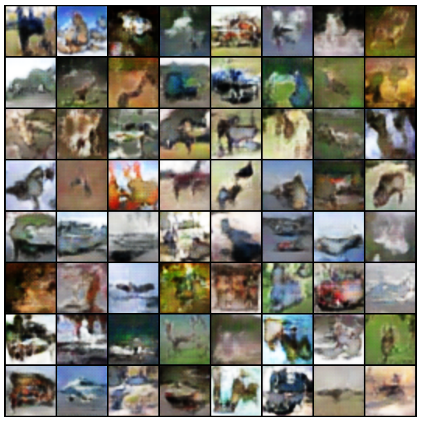
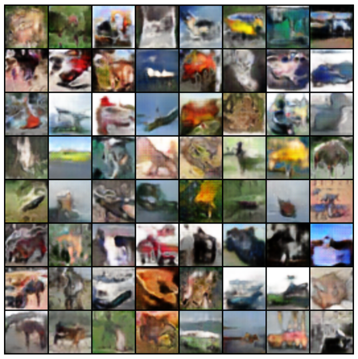
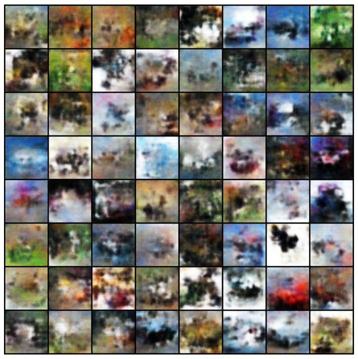

(k) Fromage

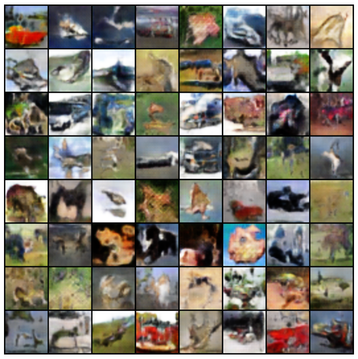
Figure 3: Fake samples from WGAN trained on the CIFAR-10 dataset using different optimizers.

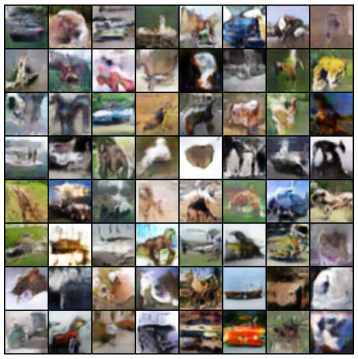

(a) Real samples

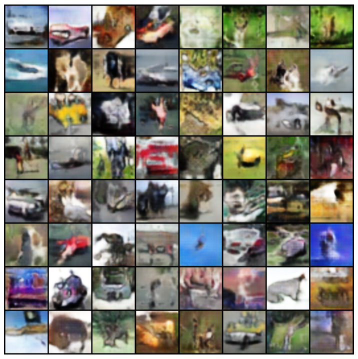

(b) AdaLA

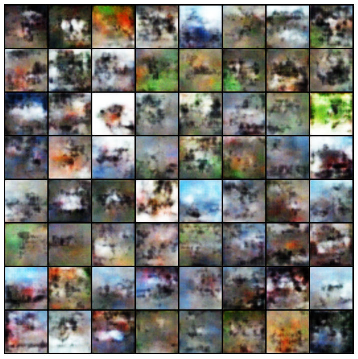

(c) AdaBelief

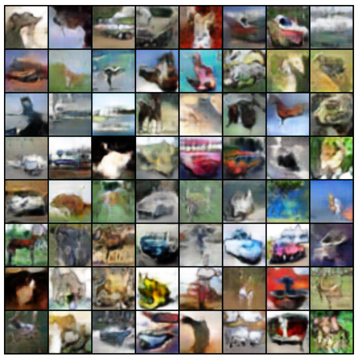

(d) SGD

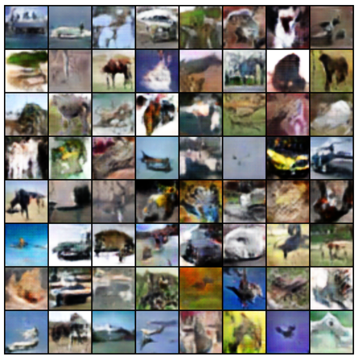

(e) AdaBound


(f) Yogi


(g) Adam


(h) MSVAG


(i) RAdam


(j) AdamW

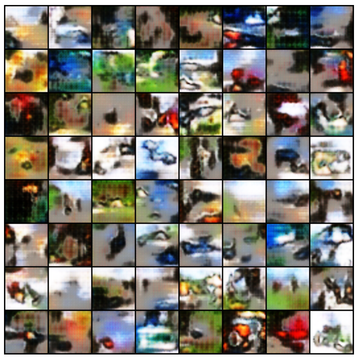

(k) Fromage

Figure 4: Fake samples from WGAN trained on the CIFAR-10 dataset using different optimizers.

Table 4: Best FID score (lower is better) on the CIFAR-10 dataset

Optimizer	WGAN	WGAN-GP
Fromage	110.12	98.53
AdamW	94.96	104.26
RAdam	101.94	84.94
MSVAG	448.43	254.21
Adam	97.49	73.55
Yogi	103.76	88.32
AdaBound	428.57	97.98
SGD	458.67	244.10
AdaBelief	83.13	61.81
AdaLA	**81.47**	**59.84**

6 Discussion

For the image classification task, the AdaLA optimizer outperforms the other state-of-the-art optimizers considered across all the models (VGG11, ResNet34, and DenseNet121) both in terms of the CIFAR-10 test accuracy curves (qualitative) and best test accuracy (quantitative). For the language modeling task, the AdaLA optimizer outperforms the other state-of-the-art optimizers considered across all the models (1-layer, 2-layer, and 3-layer LSTMs) both in terms of the Penn TreeBank test perplexity curves (qualitative) and best test perplexity (quantitative). For the image generation task, the AdaLA optimizer outperforms the other state-of-the-art optimizers considered across all the models (WGAN and WGAN-GP) both in terms of the quality of the images generated (qualitative) and best FID score of the images generated (quantitative). Through these extensive experiments, AdaLA is demonstrated to be task, dataset, and model-agnostic and can thus be used for any Deep Learning scenario involving gradient descent-based learning.

While the performance gain achieved over AdaBelief is minimal, AdaLA outperforms the other optimizers by a large margin. The performance gain achieved over AdaBelief can be attributed to the ability of AdaLA to "look ahead" and adapt the next step size to take during gradient descent. As mentioned earlier, AdaBelief uses the EMA of gradients to adapt step sizes. While this works well in practice, given the high dimensionality of the parameter space of neural networks, assuming a smooth parameter space often leads to inaccurate "beliefs" when adapting the step sizes using the EMA of gradients. As such, "looking ahead" to adapt step sizes is a better strategy that leads to improved performance, as observed in the previous section.

An important takeaway from the CIFAR-10 test accuracy curves (Fig. 1) and Penn TreeBank test perplexity curves (Fig. 2) is the instability of AdaLA during gradient descent. In particular, the adaptive learning rate methods are known to be unstable when nearing convergence. While AdaBelief addresses this issue using an EMA of the gradients, this issue is exacerbated in the case of AdaLA as the step sizes can drastically change at every step of gradient descent based on the curvature of the parameter space. However, "looking ahead" prevents AdaLA from taking step sizes that could cause issues such as overshooting or oscillation around the minima and as such, AdaLA leads to improved convergence over the other optimizers, despite the instability caused by a lack of using an EMA for the gradients.

Finally, it should be noted that AdaLA, while a simple modification over Adam or AdaBelief, leads to longer training times as a result of "looking ahead". Looking ahead requires taking a step, computing the gradients, reverting the model parameters to the initial state before taking the step, and then taking an adapted step based on the "looked ahead" gradients. This essentially means that every step of AdaLA requires an additional step to "look ahead", which leads to longer training times, especially in the case of large models, where updating the model parameters is expensive.

7 Conclusion and Future Work

This paper presents AdaLA, a first-order gradient algorithm for optimization that adaptively scales the step sizes by "looking ahead". AdaLA is shown to lead to improved convergence over other state-of-the-art optimizers across a variety of Deep Learning tasks, datasets, and models while using the same hyperparameters as Adam.

Future work on improving AdaLA can focus on addressing the two drawbacks of AdaLA: instability and increased training times. The instability of AdaLA could potentially be addressed by incorporating an EMA of gradients (similar to AdaBelief) when adapting the step sizes. Perhaps a weighted linear or non-linear combination of the "looked ahead" gradients and EMA gradient estimates will lead to improved stability while retaining AdaLA's improved convergence. Finally, the increased training time issue of AdaLA could potentially be addressed by taking k steps forward before taking a step back as performed by the Lookahead Optimizer (29).

References

[1] J. Zhuang, T. Tang, S. Tatikonda, N. Dvornek, Y. Ding, X. Papademetris, and J. S. Duncan, "Adabelief optimizer: Adapting stepsizes by the belief in observed gradients," *arXiv preprint arXiv:2010.07468*, 2020.

[2] Y. Nesterov, "Gradient methods for minimizing composite functions," *Mathematical Programming*, vol. 140, no. 1, pp. 125–161, 2013.

[3] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[4] Y. E. Nesterov, "A method for solving the convex programming problem with convergence rate o (1/kˆ 2)," in *Dokl. akad. nauk Sssr*, vol. 269, pp. 543–547, 1983.

[5] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, pp. 1139–1147, PMLR, 2013.

[6] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *Ussr computational mathematics and mathematical physics*, vol. 4, no. 5, pp. 1–17, 1964.

[7] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.

[8] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[9] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[11] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," *arXiv preprint arXiv:1705.08292*, 2017.

[12] K. Lyu and J. Li, "Gradient descent maximizes the margin of homogeneous neural networks," *arXiv preprint arXiv:1906.05890*, 2019.

[13] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *arXiv preprint arXiv:1406.2661*, 2014.

[14] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," *arXiv preprint arXiv:1706.08500*, 2017.

[15] L. Luo, Y. Xiong, Y. Liu, and X. Sun, "Adaptive gradient methods with dynamic bound of learning rate," *arXiv preprint arXiv:1902.09843*, 2019.

[16] S. Reddi, M. Zaheer, D. Sachan, S. Kale, and S. Kumar, "Adaptive methods for nonconvex optimization," in *Proceeding of 32nd Conference on Neural Information Processing Systems (NIPS 2018)*, 2018.

[17] L. Balles and P. Hennig, "Dissecting adam: The sign, magnitude and variance of stochastic gradients," in *International Conference on Machine Learning*, pp. 404–413, PMLR, 2018.

[18] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," *arXiv preprint arXiv:1908.03265*, 2019.

[19] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[20] J. Bernstein, A. Vahdat, Y. Yue, and M.-Y. Liu, "On the distance between two neural networks and the stability of learning," *arXiv preprint arXiv:2002.03432*, 2020.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[22] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[23] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[24] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[25] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, “Building a large annotated corpus of english: The penn treebank,” 1993.

[26] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[27] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, “Improved training of wasserstein gans,” *arXiv preprint arXiv:1704.00028*, 2017.

[28] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans,” *arXiv preprint arXiv:1606.03498*, 2016.

[29] M. R. Zhang, J. Lucas, G. Hinton, and J. Ba, “Lookahead optimizer: k steps forward, 1 step back,” *arXiv preprint arXiv:1907.08610*, 2019.