A Survey on Proposed Methods to Address Adam Optimizer Deficiencies

Sajad Norouzi, MohammadReza Ebrahimi
Department of Electrical and Computer Engineering, University of Toronto
Emails: s.norouzi@mail.utoronto.ca, mr.ebrahimi@mail.utoronto.ca

***Abstract*—Training neural networks, especially deep neural networks, has been a long-lasting challenge in the machine learning community. Fundamentally, training a neural network is a non-convex optimization problem and therefore, leveraging Stochastic Gradient Descent (SGD) is a natural choice. However, training deep architectures using SGD is extremely time-consuming. Currently, the most commonly used method for training deep networks is Adam [1]. However, recent theoretical works suggest that despite its attractive convergence rate, Adam could generalize worse than SGD, especially for very deep networks [2]. As a result, we are witnessing a considerable interest in improving established optimization methods (like Adam and momentum assisted SGD) both in terms of convergence speed and generalization.**

In this project, we want to shed some light on the following question: Is this the time to replace conventional Adam with one of its proposed variants?

***Index Terms*—deep neural network, deep learning, non-convex optimization, Stochastic Gradient Descent, Adam, AdamW, cosine annealing**

I. Introduction

There is a subtle correspondence between each machine learning method and an optimization technique. Among all of the many optimization problems imaginable in machine learning, perhaps the most difficult one is training (deep) neural networks [3]. Therefore, one of the main barriers in the progress of deep neural architectures is overcoming the many challenges involved in the optimization aspect of the problem. These days, it is not a surprise to spend many days on numerous machines to find the best set of practical strategies for training a single neural network model. As a result, due to the expensive and time-consuming nature of training, developing a specialized set of optimization techniques have become a dominant research stream.

A. Challenges in Training Neural Networks

The main challenges in training neural networks stem from the non-convexity of the problem. The very first implication of this property is that we have to deal with local minima. Indeed, nearly any deep model is essentially guaranteed to have an extremely large number of local minima [3], however, the overall belief is that for sufficiently large networks, most local minima have a small cost function value [4]. However, having the same loss function value doesn't mean that the peroformance of the networks on test data is similar. Sharp local minimas are points that the network overfitted trainning data and it's obvisouly undesirable. In fact, by adopting careful momentum methods, the optimization algorithm can avoid falling into sharp minima and prevent overfitting. On the other hand, plateaus and saddle points are more common and problematic [4], especially in high dimensional settings. Vanishing or exploding gradient problem arises when the neural network architecture becomes very deep, which can make learning impossible or unstable respectively. The reason for this problem lies in consecutive multiplication of weights of different layers in the gradient toward the backward path in the backpropagation algorithm.

It is worth mentioning that, noisy or inexact gradient estimations, sensitivity to parameter initialization, and loss function cliffs (extremely steep regions in the loss function) are other challenges in solving the neural network training problem. Various optimization algorithms are designed to overcome imperfections in the gradient estimate [3], like choosing a surrogate loss function. Batch normalization [5] is a technique to dramatically reduce sensitivity to initialization as well as speeding up the training process. Furthermore, loss function cliffs can be avoided using gradient clipping [6].

B. Adaptive Gradient Methods

Stochastic gradient descent (SGD) is the most basic yet one of the most dominant approaches for training deep neural networks [3], and has been empirically shown to be efficient on large datasets [7]. However, SGD in its vanilla form suffers from slow convergence rate due to its static learning rate across dimensions and time. Therefore, adaptive variants of SGD have been proposed in recent years to cope with this issue.

Adagrad [8] is among the first generation of such adaptive algorithms to surpass vanilla SGD in terms of optimization time. It started a long chain of adaptive gradient methods. Despite the theoretical convergence guarantee for convex objectives, it does not perform well in the non-convex settings due to the rapid decaying of learning rate. A variant to Adagrad, RMS-prop [9], tackles this issue by using an exponential moving average on second-order momentums. In addition to this, Adam [1] combines momentum with RMS-prop to create one of the most popular optimization algorithms in training deep neural networks. A detailed discussion of these methods will be presented in section II.

However, research like [2] suggest that for highly over-parameterized neural networks (e.g. convolutional neural networks like ResNet [10]), training with Adam or its variants could generalize worse than SGD. In fact, carefully-tuned

SGD, with proper momentum, weight decay, and learning rate decay strategies, can outperform adaptive gradient algorithms eventually ([2], [7]). This has risen the interest in developing methods to address Adam's deficiencies.

As an example, with intricate modification, Amsgrad [11] tries to solves the short memory problem of exponential moving average in Adam. In addition, the following variations of Adam algorithm has been proposed over the last year: Quasi-hyperbolic Adam (QHAdam) [12], Partial Adam (Padam) [7], Normalized Direction Adam (ND-Adam) [13], Accelerated Adam (AAdam) [14], Switches from Adam to SGD (SWAS) [15], and Adam with weight decay (AdamW) [16].

The number of recent publications in this area offers an incentive for this project to compare some of the most interesting variants of Adam and bring some insight to help decide if we should surrogate vanilla Adam with its fancier versions.

In section II, we will present a review of related adaptive gradient methods. Section III quickly reviews some of the most important ideas for learning rate adjustments. Section IV provides results of numerical experiments on selected methods along with detailed experiment settings. Finally, section V summarizes the main findings of this project.

II. Review of Related Gradient Methods

Stochastic gradient descent (SGD) and its numerous variants are the most commonly used optimization methods in machine learning and more specifically, in deep learning [3]. SGD obtains an unbiased estimate of the gradient using a minibatch of training samples. By moving in the direction of an exponentially decaying moving average of past gradients, momentum method [17] tries to speed up the convergence, especially in cases where the Hessian matrix is poorly conditioned (i.e. the objective function is much more sensitive in some dimensions, e.g. has a canyon shape with steep sides [3]). Momentum tries to cancel out the zig-zag-shaped traverse path in the landscape of loss function, at a cost of bearing degrees of overshoot. On the other hand, momentum method helps to avoid sharp local minima and saddle points in the objective function.

$$\begin{aligned} \boldsymbol{v}_t &= \alpha \boldsymbol{v}_{t-1} + \nabla f(x_t) \\ \boldsymbol{x}_{t+1} &= \boldsymbol{x}_t - \epsilon \boldsymbol{v}_t \end{aligned} \tag{1}$$

where ϵ is the learning rate or step size, and α is the moving average exponent factor. Note that using $\alpha = 0$, (3) downgrades to the vanilla SGD.

One of the popular choices for improving the convergence speed of momentum assited SGD is Nesterov accelerated gradient, where the gradient is evaluated after the momentum direction is added to the current point:

$$\begin{aligned} \boldsymbol{v}_t &= \alpha \boldsymbol{v}_{t-1} + \nabla f(x_t + v_t) \\ \boldsymbol{x}_{t+1} &= \boldsymbol{x}_t - \epsilon \boldsymbol{v}_t \end{aligned} \tag{2}$$

this algorithm has a intuition that with looking one step ahead, we can fix the current momentum direction.

With simple changing the variables ($x_t = x_t + v_t$) Equation (2) can also be written as followig:

$$\begin{aligned} \boldsymbol{v}_t &= \alpha \boldsymbol{v}_{t-1} + \nabla f(x_t) \\ \hat{\boldsymbol{v}}_t &= \alpha \boldsymbol{v}_t + \nabla f(x_t) \\ \boldsymbol{x}_{t+1} &= \boldsymbol{x}_t - \epsilon \hat{\boldsymbol{v}}_t \end{aligned} \tag{3}$$

Forcing the learning rate to be the same for each dimension is restrictive. Adagrad [8] is one of the first methods to propose adaptive learning rate across dimensions. The update rule for Adagrad is as follows:

$$\boldsymbol{c}_t = \boldsymbol{c}_{t-1} + \nabla f(x_t) \otimes \nabla f(x_t) \tag{4}$$

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \epsilon \nabla f(x_t) \otimes \frac{1}{\sqrt{\boldsymbol{c}_t}} \tag{5}$$

where we show element-wise multiplication by $\otimes$ symbol. Note that, while in (3) $\boldsymbol{v}_t$ represents the first order momentum, in (5) $\boldsymbol{c}_t$ is called the second-order momentum. Despite the theoretical convergence guarantee for convex objectives, from (4), accumulation of second order momentums in each iteration, makes the update term very small. RMS-prop solves this issue by exploiting an exponential decaying moving average on second-order momentum:

$$\begin{aligned} \boldsymbol{c}_t &= \alpha \boldsymbol{c}_{t-1} + (1-\alpha) \nabla f(x_t) \otimes \nabla f(x_t) \\ \boldsymbol{x}_{t+1} &= \boldsymbol{x}_t - \epsilon \nabla f(x_t) \otimes \frac{1}{\sqrt{\boldsymbol{c}_t}} \end{aligned} \tag{6}$$

Although supported by no theoretical guarantee, RMS-prop's superior empirical performance raised attention to exponential moving average versions of Adagrad [7]. Among the proposed variants, Adam is the most popular one. Fusing the exponential moving average on second-order momentum with the first-order momentum, Adam implements both adaptive learning rate and momentum acceleration:

$$\begin{aligned} \boldsymbol{v}_t &= \beta_1 \boldsymbol{v}_{t-1} + (1-\beta_1) \nabla f(x_t) \\ \boldsymbol{c}_t &= \beta_2 \boldsymbol{c}_{t-1} + (1-\beta_2) \nabla f(x_t) \otimes \nabla f(x_t) \\ \boldsymbol{x}_{t+1} &= \boldsymbol{x}_t - \epsilon_t \boldsymbol{v}_t \otimes \frac{1}{\sqrt{\boldsymbol{c}_t}} \end{aligned} \tag{7}$$

where learning rate is updated using $\epsilon_t = \epsilon/\sqrt{t}$ in each iteration. Note that by choosing $\beta_1 = 0$, Adam reduces to RMSprop. Basically Adam is a combination of RMSProp and Momentum. One of the first variants of Adam is adding Nesterov to Adam (Nadam). It is shown that Nadam can help the optimization in some applications.

$$\begin{aligned} \boldsymbol{v}_t &= \beta_1 \boldsymbol{v}_{t-1} + (1-\beta_1) \nabla f(x_t) \\ \boldsymbol{c}_t &= \beta_2 \boldsymbol{c}_{t-1} + (1-\beta_2) \nabla f(x_t) \otimes \nabla f(x_t) \\ \hat{\boldsymbol{v}}_t &= \beta_1 \boldsymbol{v}_t + (1-\beta_1) \nabla f(x_t) \\ \boldsymbol{x}_{t+1} &= \boldsymbol{x}_t - \epsilon_t \hat{\boldsymbol{v}}_t \otimes \frac{1}{\sqrt{\boldsymbol{c}_t}} \end{aligned} \tag{8}$$

In [11], some problems in the convergence of Adam are mentioned. More specifically, because of the short memory imposed by the exponentially decaying moving average, it is shown that in some cases gradient information vanishes. The

update rule for Amsgrad, shows an intricate modification to Adam in order to introduce a long-term memory:

$$\begin{aligned}
&\boldsymbol{v}_t = \beta_1 \boldsymbol{v}_{t-1} + (1-\beta_1)\nabla f(x_t) \\
&\boldsymbol{c}_t = \beta_2 \boldsymbol{c}_{t-1} + (1-\beta_2)\nabla f(x_t) \otimes \nabla f(x_t) \\
&\hat{\boldsymbol{c}}_t = \max(\hat{\boldsymbol{c}}_{t-1}, \boldsymbol{c}_t) \qquad (9) \\
&\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \epsilon_t \boldsymbol{v}_t \otimes \frac{1}{\sqrt{\hat{\boldsymbol{c}}_t}} \qquad (10)
\end{aligned}$$

By introducing step (9), an extra long-term memory is preserved and ensures the decay of the effective learning rate. This implies an $O(1/\sqrt{T})$ convergence rate in the convex settings.

Generally, recent advances in deep learning show that making models more adaptive in a meaningful way can help the models to achieve better results while it might become harder to explain their behavior. Following two algorithms propose some hyper-parameters to make the update rule more adaptive in order to achieve better results.

It is shown [2] that Adam suffers from generalization issue, specifically over highly over-parameterized architectures (like very deep convolutional neural networks) in comparison to momentum assisted SGD. One explanation for this is that the trade-off between momentum acceleration and learning rate adaptation needs to be tuned more precisely.

Padam introduces a variant to Adam which enables interpolating between momentum assisted SGD and Amsgrad, using a single hyperparameter p:

$$\begin{aligned}
&\boldsymbol{v}_t = \beta_1 \boldsymbol{v}_{t-1} + (1-\beta_1)\nabla f(x_t) \\
&\boldsymbol{c}_t = \beta_2 \boldsymbol{c}_{t-1} + (1-\beta_2)\nabla f(x_t) \otimes \nabla f(x_t) \\
&\hat{\boldsymbol{c}}_t = \max(\hat{\boldsymbol{c}}_{t-1}, \boldsymbol{c}_t) \\
&\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \epsilon_t \boldsymbol{v}_t \otimes \frac{1}{\hat{\boldsymbol{c}}_t^{p}} \qquad (11)
\end{aligned}$$

where $p \in (0, 0.5]$ is called the *partially adaptive parameter*. In one end, by letting $p \to 0$, Padam reduces to momentum assisted SGD, and at the other end, $p = 0.5$ resembles Amsgrad. Therefore, by tuning parameter p, Padam seeks the best trade-off between the effect of momentum and learning rate adaptation. Results in [16] support that by carefully tuning p, Padam achieves a convergence speed as fast as that of Adam, while generalizing as well as SGD.

Quasi Hyperbolic Adam (QHAdam) [12] is another variant which tries to tune the contribution of the current state gradient to the update rule more adaptively. The Update rule is:

$$\begin{aligned}
&\boldsymbol{v}_t = \beta_1 \boldsymbol{v}_{t-1} + (1-\beta_1)\nabla f(x_t) \\
&\boldsymbol{c}_t = \beta_2 \boldsymbol{c}_{t-1} + (1-\beta_2)\nabla f(x_t) \otimes \nabla f(x_t) \\
&\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \epsilon_t \frac{\nabla f(x_t)(1-\nu_1) + \boldsymbol{v}_t \nu_1}{\sqrt{(1-\nu_2)(\nabla f(x_t))^2 + \nu_2 \boldsymbol{c}_t}} \qquad (12)
\end{aligned}$$

While the formula may seem a bit complex at the first glance, it is only decoupling the contribution of the gradient at the current point to the final update rule from the first and the second momentum. setting $\nu_1 = 1$ and $\nu_2 = 1$ recovers adam, $\nu_1 = 0$ and $\nu_2 = 1$ recovers RMSprop, and $\nu_1 = \beta_1$ and $\nu_2 = 1$ recovers Nadam. So as we discussed, it's a step toward making update rule more flexible to obtain hybrid update rules.

AdamW [16] is another algorithm which discusses that complex update rule of the Adam defeat the purpose of the regularization. They change the update rule to decouple the weight decay update from the original error signal of the loss function. So the algorithm is:

$$\begin{aligned}
&\boldsymbol{v}_t = \beta_1 \boldsymbol{v}_{t-1} + (1-\beta_1)\nabla f(x_t) \\
&\boldsymbol{c}_t = \beta_2 \boldsymbol{c}_{t-1} + (1-\beta_2)\nabla f(x_t) \otimes \nabla f(x_t) \\
&\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \epsilon_t \boldsymbol{v}_t \otimes \frac{1}{\sqrt{\boldsymbol{c}_t}} - \lambda \boldsymbol{x}_t \qquad (13)
\end{aligned}$$

λ is L2 regularization parameter. Using this update rule we should omit the L2 regularization part from the loss function. We should also mention that because the λ is not multiplied by the learning rate anymore, we need to decay λ. The suggestion of the original paper is to decay the λ similar to learning rate decay schedule. Without this decoupling, due to the second momentum division, original Adam decay the weights which are larger much less, that is totally in contrast with the original purpose of L2 regularization. So, the idea totally makes sense and the results are promising. Moreover learning rate and regularization hyper-parameters are not correlated anymore so we can use grid search instead of random search on learning rate and regularization to find the best combination of hyper-parameters.

III. Learning Rate Annealing

One of the most challenging aspects of optimizing high-dimensional and non-convex objectives is determining the best learning rate. Among many proposed methods for systematically scheduling the learning rate, we focus on learning rate annealing. In this technique, optimization starts with a relatively high learning rate but it is gradually decreased during the training. The simple intuition behind this approach is that by starting with a relatively large learning rate we can quickly traverse from the initial parameter values to a region with a lower loss function, and by gradually lowering the learning rate we can explore the deeper, but narrower parts of the loss function [18]. Perhaps, the most commonly used form of learning rate annealing is a step learning rate decay. In this approach, the learning rate is reduced by a constant factor after a certain number of epochs.

While step learning decay is still the most common decay approach, some other learning rate schedulers have been proposed recently. Cosine annealing [19] is periodically increasing and decreasing the learning rate following a cosine function. The search algorithm tries to explore different local minima and expand the search space of the objective landscape.

the formal definition of cosine annealing scheduler is:

$$\eta_t = \eta^i_{\text{min}} + \frac{1}{2}(\eta^i_{\text{max}} - \eta^i_{\text{min}})\left(1 + \cos\left(\frac{T_{\text{cur}}}{T_i}\pi\right)\right) \qquad (14)$$

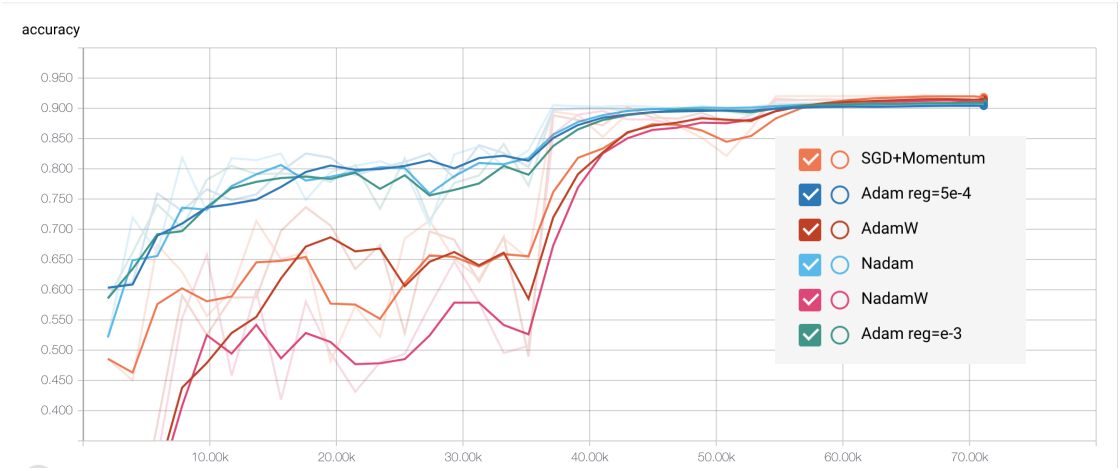

Fig. 1. Test accuracy as a function of iteratiions

Fig. 2. Test accuracy as a function of iteratiions

where η^i_{min} and η^i_{max} are ranges for the multiplier, and T_{cur} accounts for how many epochs have been performed since the last period restart.

In practice, a more simple version of the scheduler is used:

$$\eta_t = 0.5 + 0.5\left(1 + \cos\left(\frac{T_{\text{cur}}}{T_i}\pi\right)\right) \tag{15}$$

One attractive advantage of such method is that by saving the weights at the end of each cycle, one can build an "ensemble of models" at the cost of training a single model [20]. Also, [16] reports a 15% relative improvement in test errors compared to Adam both on CIFAR-10 and ImageNet32x32 by utilizing cosine annealing.

Cyclical Learning [21] rate is another variant of periodically increasing and decreasing the learning rate which instead of using Cosine function uses a linear function. [21] justifies increasing the learning rate by arguing that, although the increase might have a short-term negative effect on the loss, it can achieve a longer-term beneficial effect by converging to the best local optima in sense of generalization. [21] also showed that decaying the range of the learning rate scheduler can be helpful. It makes sense because at the beginning of the learning you might want to escape from local minima and explore a wide range of loss landscape, but after some cycles, it's clear that we want to reduce the amount of exploration.

IV. Numerical Experiments

For numerical experiment part, we focused on the image classification task which has been one of the main tasks in deep learning. we selected ResNet [10], and adapted the TensorFlow implementation of the model from [1]. We trained all the networks on cifar10 dataset using Nvidia-Titan X. Because of the number of proposed methods and long time of training, we couldn't explore all the models in this time, so we limited our project to more promising modifications of the Adam. We selected AdamW and compared the results with Adam, Nadam, momentum assisted SGD. The total number of

[1]https://github.com/tensorflow/models/tree/master/official/resnet

training epochs is 182 and the learning rate scheduler is based on the original ResNet paper that is multiplying by 0.1 at the epochs 91, 136, and 182. The original paper used momentum assisted SGD with base learning rate equals to 0.1 and L2 regularization hyper-parameter equals to 5e-4. We replaced the optimizer with Adam and because of this change, we had to optimize the learning rate again. Based on our experiments base learning rate equals to 0.001 is the best and selecting different epochs to decay learning rate does not affect the final accuracy. We also tuned the L2 regularization parameter. With base learning rate equals 0.001, L2 regularization equals, 5e-4 we trained Adam, Nadam, AdamW, NadamW. The results proved that AdamW can achieve better results than Adam in image classification. One might valid argument is that we need to tune regularization parameters per optimization algorithms to make sure which one better, but we can say that AdamW can get so close to the accuracy of SGD momentum without any extra regularization re-tuning which is definitely a benefit. We have explored NadamW that we are not aware of any previous exploration on that. The final results for this task is shown in the Table I.

TABLE I
TEST ACCURACY ON CIFAR10

Optimizer	Accuracy
SGD+Momentum	**0.9168**
Adam	0.9035
AdamW	**0.9155**
Nadam	0.9068
NadamW	0.9126

In Fig. 1 you can see the curve of test accuracy improvement. We also replaced the original learning rate decay scheduler with cosine annealing scheduler explained in section III to first check if this scheduler improves the final accuracy and next to compare the compatibility of Adam, AdamW, and NadamW with cosine annealing. Fig. 2 shows that this annealing improves the speed of convergence to a good point. Table II summarizes the final results.

TABLE II
TEST ACCURACY ON CIFAR10 WITH COSINE ANNEALING

Optimizer	Accuracy
Adam	0.9052
AdamW	**0.9187**
NadamW	0.9160

V. CONCLUSION

In this project, we gathered various modifications of Adam optimization and explained their difference in variables update rules. Then as the first step of verifying and comparing the proposed methods, we selected AdamW which seems one of the most promising ones with a clear theoretical explanation. We applied this algorithm on image classification task over Cifar10 using ResNet. As our result shows, the final accuracy is better than vanilla Adam without retuning the regularization parameter (which is required for Adam). We then utilized cosine annealing for Adam, AdamW, and NadamW. The results show that the cosine annealing improves the accuracy of all the algorithms. Moreover, there is no difference between their compatibility with cosine annealing. After applying cosine annealing, AdamW surpasses SGD momentum with step learning rate decay which shows the power of AdamW plus cosine annealing.

To sum up, we verified that AdamW helps the accuracy of image classification task and the reason behind this is that the second momentum used in Adam is in contradiction with the purpose of weight decay.

REFERENCES

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

[2] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4148–4158.

[3] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[4] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Advances in neural information processing systems*, 2014, pp. 2933–2941.

[5] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[6] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning*, 2013, pp. 1310–1318.

[7] J. Chen and Q. Gu, "Closing the generalization gap of adaptive gradient methods in training deep neural networks," *arXiv preprint arXiv:1806.06763*, 2018.

[8] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[9] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, p. 14, 2012.

[10] 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*, 2016, pp. 770–778.

[11] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," 2018.

[12] J. Ma and D. Yarats, "Quasi-hyperbolic momentum and adam for deep learning," *arXiv preprint arXiv:1810.06801*, 2018.

[13] Z. Zhang, L. Ma, Z. Li, and C. Wu, "Normalized direction-preserving adam," *arXiv preprint arXiv:1709.04546*, 2017.

[14] A. Tato and R. Nkambou, "Improving adam optimizer," 2018.

[15] N. S. Keskar and R. Socher, "Improving generalization performance by switching from adam to sgd," *arXiv preprint arXiv:1712.07628*, 2017.

[16] I. Loshchilov and F. Hutter, "Fixing weight decay regularization in adam," *arXiv preprint arXiv:1711.05101*, 2017.

[17] 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.

[18] "cs231 convolutional neural networks for visual recognition," http://cs231n.github.io/neural-networks-3/annealing-the-learning-rate, accessed: 2018-12-06.

[19] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[20] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, "Snapshot ensembles: Train 1, get m for free," 2017.

[21] L. N. Smith, "Cyclical learning rates for training neural networks," *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Mar 2017. [Online]. Available: http://dx.doi.org/10.1109/WACV.2017.58