Towards glimpsing optimizer in deep learning: from SGD to Lookahead to the future

Wenhao Tang

School of Big Data and Software Engineering, Chongqing University, Chonqqing, 400044 P.R.China

Email: whtang@cqu.edu.cn

Student Number: 202124021006

Abstract

Optimizers are critical to the performances of neural networks but are often used as black-box optimizers because it is difficult to get a practical explanation of their advantages and disadvantages. In this article, we first introduce the vanilla gradient descent in deep learning, and the most common optimization algorithms with their motivation and category, such as: accelerated schemes and adaptive learning rate schemes. Then we introduce some latest works in this field, Lookahead and multi-optimizer combination. Finally, we summarize that theoretical analysis and practicality are the most important indicators for choosing an optimizer.

Index Terms

Deep learning, Overview, Optimizer, Subgradient optimization.

I. Introduction

Optimizers are critical to the performances of neural networks, once a task has been selected, an architecture designed and a dataset assembled, one needs to train a neural network to make it performant as most neural networks return random outputs in the absence of training. Thus, it has long been understood that optimizers are a primordial component of deep learning and that a good optimizer can drastically improve the performance of a given architecture [1].

This article aims at providing the reader with intuitions about the behavior of different algorithms for optimizing gradient descent that will help her put them to use. In Section II, we define the models, problems, and optimizers in deep learning and their optimization process. Subsequently, in Section III, we are first going to look at how to introduce vanilla gradient descent into deep learning and summarize the challenges of its variants. Then we introduce the most common optimization algorithms and their extensions by showing their motivation to resolve these challenges and how this leads to the derivation of their update rule. Moreover, in Section IV, we present the new work Lookahead [2] proposed by Hinton et al. in 2019 and one of the latest directions, the multi-optimizer combination. Afterward, we briefly analyze the possible shortcomings of the current state-of-the-art optimization methods, and summarize that theoretical analysis and practicality are the most important indicators for choosing an optimizer.

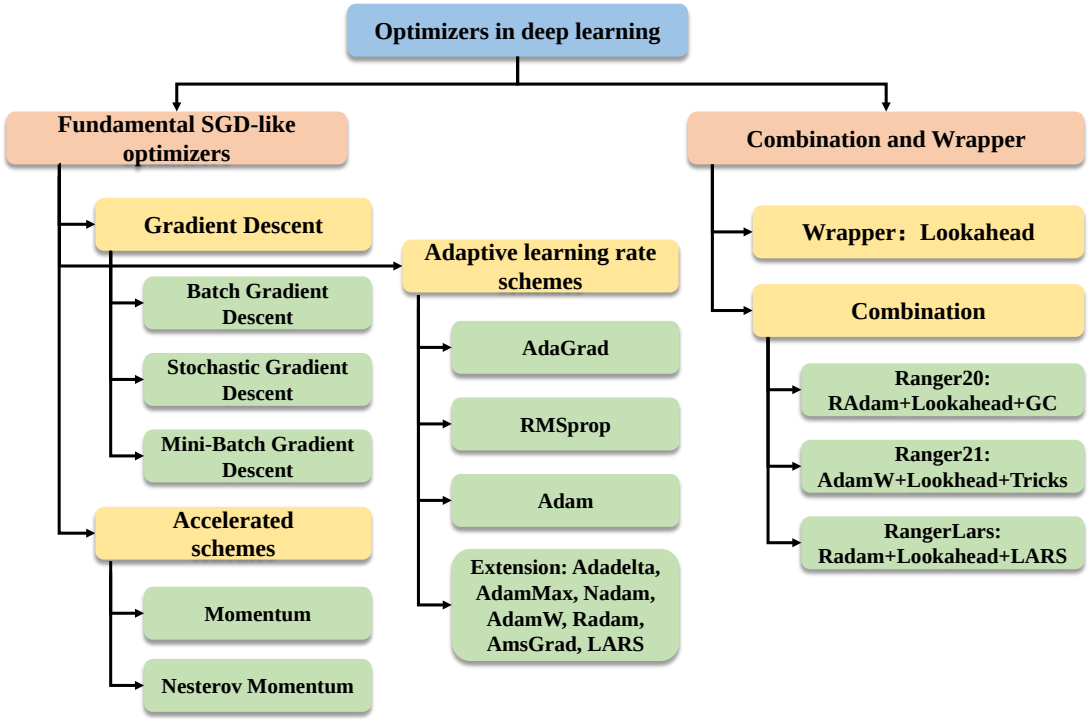


Fig. 1: Overview of optimizers of deep learning introduced in this article. Best viewed in color.

II. Problem Formulation

A. Deep Neural Networks

Deep learning models represented by Deep Neural Networks (DNNs) [3], [4] have been successfully applied to various applications, such as Computer Vision [5]–[7], Natural Language

Processing [8]–[10], and Reinforcement Learning [11]–[13]. Typically, their models $f_\theta(\cdot)$ with parameters θ can be formally formulated as:

$$\begin{aligned} \mathcal{Z}^0 &= \mathcal{X};\ h^l\left(\mathcal{Z}^{l-1}\right) = 1 \\ \widetilde{\mathcal{Y}} &= f_\theta(\mathcal{X}) = \sigma\left(h^l\left(\sigma\left(h^{l-1}\left(\cdots\sigma\left(h^1\left(\mathcal{Z}^0\right)\right)\right)\right)\right)\right), \end{aligned} \tag{1}$$

where $\mathcal{X}$ and $\widetilde{\mathcal{Y}}$ is the input and output, respectively. And $\sigma(\cdot)$ is the activation function which provide the necessary nonlinearity of the model to be able to learn complex representation [14]. Sigmod and Rectified Linear Unit (ReLU) [15], [16] are two of the most commonly used. There can be formulated as:

$$\begin{aligned} \text{Sigmod}(x) &= \frac{1}{1+e^{-x}} \\ \text{ReLU}(x) &= \max(0, x) \end{aligned} \tag{2}$$

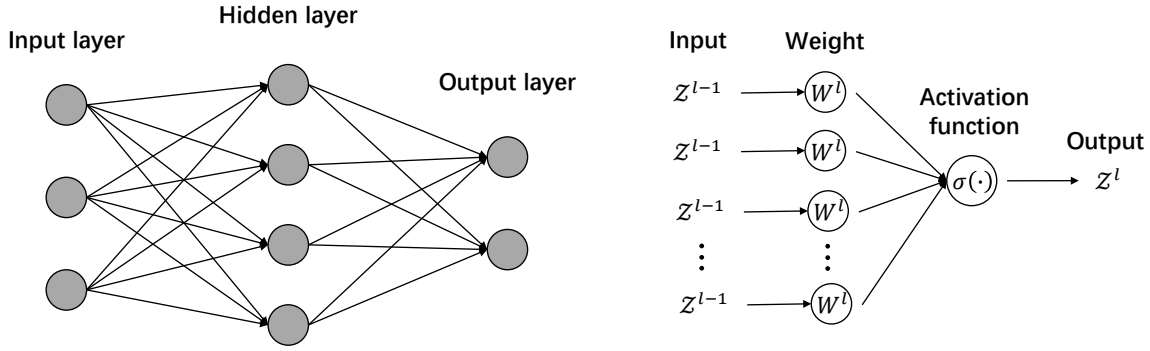


Fig. 2: The visualization examples of the features and the patch scores produced by our method. We employ CAM for feature visualization and the baseline is EfficientNet-B3 [7] here. The red box indicates the distressed area.

B. Model optimization problem

With the model $f_\theta(\cdot)$, we can formulate the optimization problem of it. The goal of this optimization problem is that minimize the distance $\ell\left(\widetilde{\mathcal{Y}}, \mathcal{Y}\right)$ between output of model $\widetilde{\mathcal{Y}}$ and targets $\mathcal{Y}$, where $\ell(\cdot)$ is the distance metric function, which can also be called cost function. Specifically, for regression and classification problems, $\ell(\cdot)$ is often chosen to be the Mean Squared Deviation (MSE) and Cross-entropy (CE), respectively. There can be formulated as:

$$\begin{aligned} \text{MSE}\left(\widetilde{\mathcal{Y}}, \mathcal{Y}\right) &= \sum\left(\widetilde{\mathcal{Y}} - \mathcal{Y}\right)^2 \\ \text{CE}\left(\widetilde{\mathcal{Y}}, \mathcal{Y}\right) &= -\sum \mathcal{Y} \log \widetilde{\mathcal{Y}} \end{aligned} \tag{3}$$

Finally, this optimization problem can be formulated as:

$$\hat{\theta} \leftarrow \arg\min_{\theta} \ell\left(f_{\theta}\left(\mathcal{X}\right), \mathcal{Y}\right). \tag{4}$$

C. Optimizer definition

In general, we consider model optimization problems in the deep learning domain as complex non-convex optimization problems. The problem does not have an analytical solution and needs to be solved using heuristic, iterative methods which referred to as optimizer. Moreover, we will not discuss algorithms that are infeasible to compute in practice for high-dimensional data sets, e.g. second-order methods such as Newton [17]. Therefore, we can denote the optimizer as $\mathcal{O}\left(\theta, \ell, \mathcal{X}, \mathcal{Y}\right)$. The process of parameter update can be formulated as:

$$\mathcal{O}\left(\theta, \ell, \mathcal{X}, \mathcal{Y}\right) := \theta_{t+1} \leftarrow \theta_t + \Delta\theta \tag{5}$$

III. Stochastic Gradient Descent variants

Gradient Descent (GD) is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks [17]. The algorithm can be formulated as:

$$\mathcal{O}_{GD} := \theta_{t+1} \leftarrow \theta_t - \eta \nabla_{\theta} \ell\left(f_{\theta}\left(\mathcal{X}\right), \mathcal{Y}\right) \tag{6}$$

where η is the learning rate, and it controls the step size of the parameter update in the direction of the gradient.

In this section, we will first introduce Stochastic Gradient Descent (SGD), which introduces GD into the optimization process of deep learning, and some challenges of it. Then we will outline some algorithms that are widely used by the Deep Learning community to deal with the aforementioned challenges. Finally, we will summarize these classical optimizers based on SGD.

A. Stochastic Gradient Descent

Firstly, we will take a fresh look at this problem from a statistical perspective. The dataset $D \in \{\mathcal{X}, \mathcal{Y}\}$ are sampled from the true unknown distribution $p\left(x\right)$. Therefore, we can get the formulation of ideal cost function and gradient:

$$\begin{aligned} \ell\left(f_{\theta}\left(x\right)\right) &= E_x\left[\ell\left(f_{\theta}\left(x\right)\right)\right] = \int p\left(x\right) \ell\left(f_{\theta}\left(x\right)\right) dx \\ \nabla_{\theta} \ell\left(f_{\theta}\left(x\right)\right) &= \nabla_{\theta} E_x\left[\ell\left(f_{\theta}\left(x\right)\right)\right] = E_x\left[\nabla_{\theta} \ell\left(f_{\theta}\left(x\right)\right)\right] \end{aligned} \tag{7}$$

Because the true distribution $p(x)$ is not available, pioneers [18], [19] in the field introduce Monte Carlo estimation and thus obtain the commonly used gradient expression:

$$\begin{gathered}\mathcal{X}=\{x_1,x_2,...,x_N\},\mathcal{Y}=\{y_1,y_2,...,y_N\}\\ E_x\left[\nabla_\theta\ell\left(f_\theta\left(x\right)\right)\right]\approx\frac{1}{M}\sum_{i=1}^{M}\nabla_\theta\ell\left(f_\theta\left(x_i\right),y_i\right)\end{gathered}\tag{8}$$

- When $M=N$, we term it Batch Gradient Descent.
- When $M=1$, we term it Stochastic Gradient Descent.
- When $M<N$, we term it Mini-Batch Gradient Descent.

1) Batch Gradient Descent: Gradient Descent, namely Batch Gradient Descent (BGD), calculates the gradient of the cost function (with respect to the parameter θ) for the entire training dataset $D\in\{\mathcal{X},\mathcal{Y}\}$:

$$\mathcal{O}_{BGD}:=\theta_{t+1}\leftarrow\theta_t-\eta\frac{1}{N}\sum\nabla_\theta\ell\left(f_\theta\left(\mathcal{X}\right),\mathcal{Y}\right)\tag{9}$$

Since we need to compute the gradient of the entire dataset to perform a *single* update, Batch Gradient Descent can be very slow and intractable for datasets that do not fit in memory. The batch gradient descent method also does not allow us to update our model online, i.e., with new examples online Of particular importance, Batch Gradient Descent is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces [17].

2) Stochastic Gradient Descent: Stochastic Gradient Descent (SGD) in contrast performs a parameter update for *each* training example x_i and target y_i:

$$\mathcal{O}_{SGD}:=\theta_{t+1}\leftarrow\theta_t-\eta\nabla_\theta\ell\left(f_\theta\left(x_i\right),y_i\right)\tag{10}$$

Batch Gradient Descent performs redundant calculations on large datasets because it recalculates the gradient for similar examples before each parameter update. SGD eliminates this redundancy by performing one update at a time. Therefore, it is usually much faster and can also be used for online learning. SGD performs frequent updates with high variance, resulting in severe fluctuations in the cost function, as shown in Figure 3.

Although the Batch Gradient Descent method converges to the minimum of the basin in which the parameters are located, the fluctuation of SGD allows it to jump to new, possibly better, local minima on the one hand. On the other hand, this eventually makes convergence to the minimum difficult, since SGD jumps constantly. However, it has been shown that when we slowly decrease

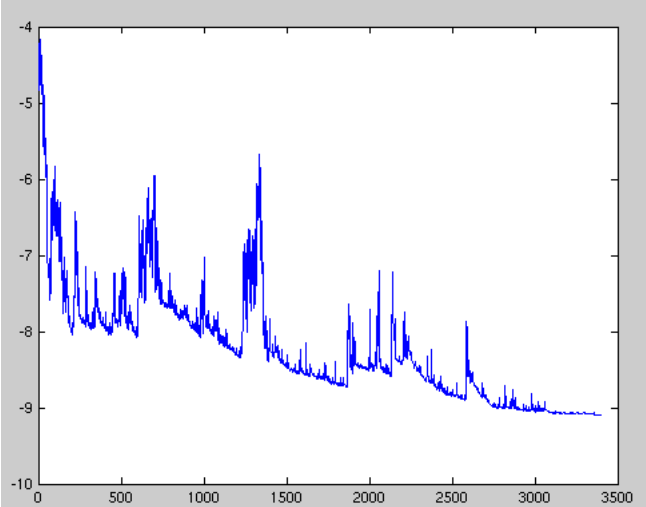


Fig. 3: Fluctuations in the total objective function as gradient steps. Source: [20].

the learning rate, SGD shows the same convergence behavior as Batch Gradient Descent, almost certainly converging to a local or the global minimum for non-convex and convex optimization, respectively [17].

3) Mini-Batch Gradient Descent: Mini-batch Gradient Descent (MBGD) eventually takes the best of both and updates for each mini-batch of M training examples:

$$\mathcal{O}_{MBGD} := \theta_{t+1} \leftarrow \theta_t - \eta \frac{1}{M} \sum \nabla_\theta \ell \left(f_\theta \left(x_i \right), y_i \right) \tag{11}$$

This way, it *a) reduces the variance of the parameter updates, which can lead to more stable convergence*; and *b) can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient w.r.t. a mini-batch very efficient* [17]. Common mini-batch sizes range from 32 to 512, but may vary for different applications. Mini-batch Gradient Descent is usually the preferred algorithm when training deep neural networks. ***Unless mentioned, we regard Mini-Batch Gradient Descent as SGD in the following sections.***

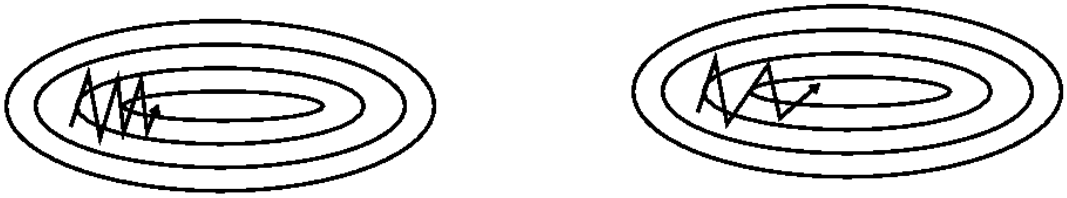
4) Challenges: However, SGD does not guarantee good convergence, but provides a number of challenges that need to be addressed:

- Choosing an appropriate learning rate is difficult. A learning rate that is too small can lead to too slow convergence, while a learning rate that is too large can hinder convergence and cause the loss function to fluctuate or even diverge around the minimum.

- In addition, the same learning rate applies to all parameter updates. If our data is sparse and our features vary widely in different dimensions, we may not want to update all features to the same extent, but rather make larger updates to features that rarely occur.
- Another key challenge in minimizing the complex non-convex cost functions common to neural networks is to avoid getting trapped in their numerous suboptimal local minima [17]. Dauphin et al. [21] argue that the difficulty actually comes not from local minima, but from saddle points, i.e., points that slope upward in one dimension and downward in the other. These saddle points are usually surrounded by a plateau of the same error, which makes it difficult for SGD to escape, even if the gradients in all dimensions are close to zero.

B. Momentum and Nesterov Momentum

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another [22], which are common around local optima. In these cases, the SGD wavers over the vicinity of the minimum rather than moving hesitantly along the trough toward the local optimum, as shown in Figure 4.



(a) SGD without Momentum

(b) SGD with Momentum

Fig. 4: The trajectory of parameter updates using SGD and SGDM. Source: [23]

1) Momentum: Momentum [24] is a method that helps SGD to accelerate and suppress oscillations in the relevant direction, as seen in Figure 4b. It does this by adding a fraction γ of the update vector from that past step v_{t-1} to the current update vector v_t. The update rule of SGD with Momentum (SGDM) can be formulated as:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \frac{1}{M} \sum \nabla_\theta \ell \left(f_{\theta_{t-1}} \left(x_i^{t-1} \right), y_i^{t-1} \right) \\ \mathcal{O}_{SGDM} &:= \theta_{t+1} \leftarrow \theta_t - v_t \end{aligned} \tag{12}$$

where t is the current update step, x_i^{t-1} is the sample x_i at update step $t-1$, and the momentum term γ is usually set to 0.9 or a similar value [17].

Clearly, Momentum accumulates energy in the small ball go downhill, and releases energy in the small ball go uphill to help it leave the canyon. Similarly, SGDM can relieve the model from being trapped by local optima and can find the global optimum faster and better compared to the traditional SGD.

2) Nesterov Momentum: However, because the update magnitude is not significantly improved, the optimization speed of SGDM is still not satisfactory in the context of deep learning.

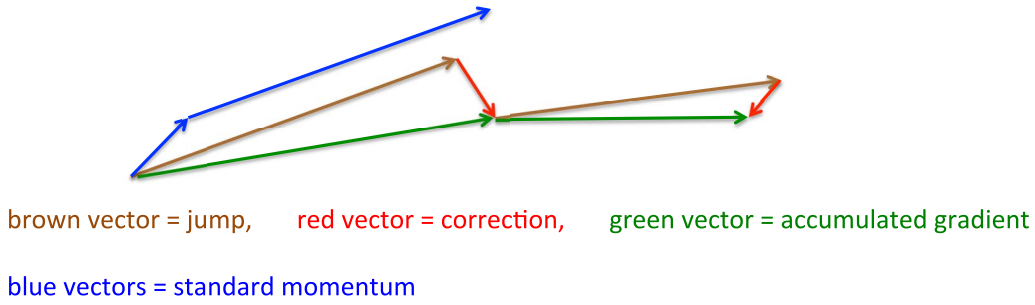


Fig. 5: The visualization of Nesterov Momentum's update rule. Best viewed in color. Source: [25].

Nesterov accelerated gradient (NAG) [26] is a better type of momentum to accelerate the optimization process of DNNs. As shown in the Figure 5, SGD with Nesterov Momentum (SGDNM) can be simply divided into two steps: First make a *big jump* in the direction of the previous accumulated gradient. Then measure the gradient where you end up and make a *correction*. The update rule of SGDNM can be formulated as:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \frac{1}{M} \sum \nabla_\theta \ell \left(f_{\theta_{t-1} - \gamma v_{t-1}} \left(x_i^{t-1} \right), y_i^{t-1} \right) \\ \mathcal{O}_{SGDNM} &:= \theta_{t+1} \leftarrow \theta_t - v_t \end{aligned} \tag{13}$$

where $f_{\theta_{t-1} - \gamma v_{t-1}}$ can be viewed as the *big jump*, and v_t is the *correction*. SGDNM relative to the traditional SGD update magnitude increase, each step can be analogous to two steps (one big step and one small step). Therefore, the optimization process of SGDNM is faster and better than previous optimizers like SGD.

C. AdaGrad and RMSprop

With Momentum and Nesterov Momentum, we are able to address the challenge of SGD in optimizing deep learning models where DNNs are trapped in local optimal points. On the other hand, its excellent design also substantially improves the optimization speed. However,

the challenges about how to adjust learning rate are not addressed, see Section III-A4. In this subsection, I will introduce two main solutions, namely AdaGrad [27] and RMSprop [25], with excellent concepts that are consistently used to this day [2], [28].

1) AdaGrad: AdaGrad [27], which stand by **Ada**ptive **Gra**dient, is a subgradient method[1] that dynamically incorporate knowledge of the geometry of the data observed in earlier iterations to perform more informative gradient-based learning. Duchi et al. [27] state that *the adaptation allows us to find needles in haystacks in the form of very predictive but rarely seen features*. Intuitively, as shown in Figure 6, AdaGrad can make the model parameters θ update as soon as possible along the direction of more stable gradient. Due to simplifying the complex tuning process of learning rate, many outstanding models in deep learning are trained based on this optimizer. For example, Dean et al. [30] have found that AdaGrad greatly improved the robustness of SGD and used it for training large-scale neural nets at Google, which among other things learned to recognize cats in YouTube videos. Moreover, Pennington et al. [31] used AdaGrad to train GloVe word embeddings, as infrequent words require much larger updates than frequent ones.

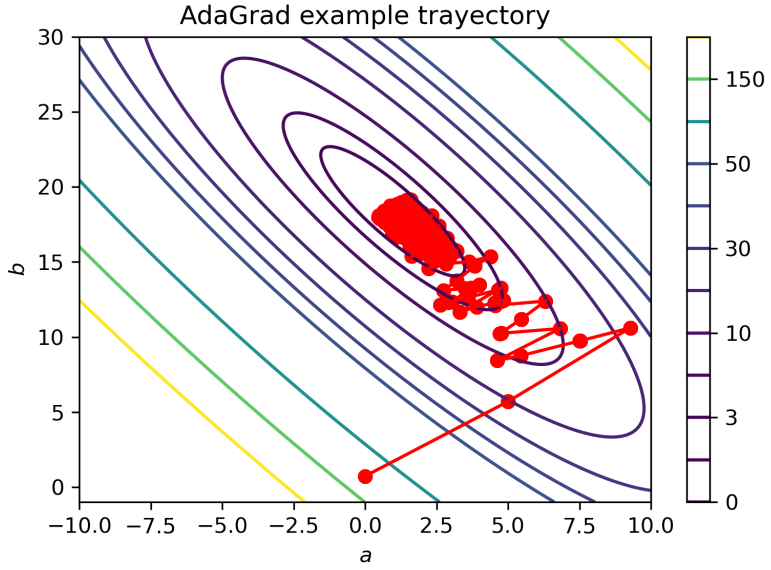


Fig. 6: The trajectory of parameter updates using AdaGrad. Best viewed in color. Source: [32].

[1]Note: The subgradient method mentioned here is same with the previous SGD-like methods, and the main reason for terming the optimizer as a subgradient optimization algorithm in the field of deep learning is to be able to apply to the non-differentiable functions included in the model. To see more details about subgradient, please refer to [29].

Moreover, we can formulate and understand AdaGrad again from the point of view of the parameters rather than the direction of the gradient. We define G_t is the outer product of all previous subgradients g_t,

$$\begin{aligned} g_t &= \frac{1}{M} \sum \nabla_\theta \ell \left(f_{\theta_t} \left(x_i^t \right), y_i^t \right) \\ G_t &= \sum_{j=1}^{t} g_t g_t^\top. \end{aligned} \tag{14}$$

Then we can get the standard update rule of AdaGrad,

$$\mathcal{O}_{AdaGrad} := \theta_{t+1} \leftarrow \theta_t - \frac{\eta g_t}{\sqrt{G_t} + \epsilon} \tag{15}$$

where ϵ is a smoothing term that avoids division by zero (usually on the order of 10^{-8}). And we expand Equation 15 according to the parameter dimension,

$$\mathcal{O}_{AdaGrad} := \begin{bmatrix} \theta_{t+1}^1 \\ \theta_{t+1}^2 \\ \vdots \\ \theta_{t+1}^m \end{bmatrix} \leftarrow \begin{bmatrix} \theta_t^1 \\ \theta_t^2 \\ \vdots \\ \theta_t^m \end{bmatrix} - \begin{bmatrix} \frac{\eta}{\sqrt{G_t^{(1,1)}} + \epsilon} \\ \frac{\eta}{\sqrt{G_t^{(2,2)}} + \epsilon} \\ \vdots \\ \frac{\eta}{\sqrt{G_t^{(m,m)}} + \epsilon} \end{bmatrix} \odot \begin{bmatrix} g_t^1 \\ g_t^2 \\ \vdots \\ g_t^m \end{bmatrix}, \tag{16}$$

where $\odot$ presents matrix multiplication, and $G_t^{(j,j)}$ is the j element in the G_t diagonal. From the Equation 16, it is clear that the update rule for AdaGrad adapts the step size for each parameter j according to $\eta \left(\epsilon + G_t^{(j,j)} \right)^{-1/2}$, while standard subgradient methods have fixed step size η for every parameter.

2) RMSprop: Although AdaGrad adaptively adjusts the learning rate of different parameters using cumulative squared gradients G_t, Hinton et al. still found its shortcomings. First, the cumulative squared gradient is *monotonically increasing*, which can lead to smaller and smaller learning rates later in the optimization. Second, the update method of the cumulative squared gradient $G_t = \sum_{j=1}^{t} g_t g_t^\top$ makes it inevitably prone to drastic changes in the early stages, which can have irreversible effects on the parameter optimization.

To address aforementioned issues, RMSprop, which stand for **R**oot **M**ean **S**quare **Prop**agation, is proposed by Hinton et al. in Lecture 6e in his Coursera Class [25]. Instead of directly using cumulative squared gradients G_t to adjust the learning rate, RMSprop divides the learning rate by an exponentially meaning average (EMA) of squared gradients to achieve a smoother learning rate adjustment. On the other hand, the cumulative gradient updated by EMA R_t is no longer

monotonic, solving the problem of decreasing learning rate in the later stages of training. The update rule of RMSprop can be formulated as:

$$\begin{aligned} R_t &= \mu R_{t-1} + (1-\mu)\, g_t g_t^\top \\ \mathcal{O}_{RMSprop} := \theta_{t+1} &\leftarrow \theta_t - \frac{\eta g_t}{\sqrt{R_t + \epsilon}}, \end{aligned} \tag{17}$$

where the μ is the update term of EMA, and Hinton suggests it to be set 0.9. While a good default value for the learning rate η is 10^{-3}.

D. Adam

At the end of Lecture 6e in Hinton's Coursera Class [25], he suggests that the improvement idea of combining RMSprop and Momentum is possible. Thus, Adam [33] is introduced, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. Similarly to Momentum, Adam also has a momentum term m_t and normalizes it using EMA based on the previous Momentum:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\, g_t, \tag{18}$$

where the β_1 is the update term of EMA. In addition to that, Adam has a cumulative squared gradient term v_t, which is almost exactly similar to cumulative squared gradients of RMSprop R_t. And it also is updated by EMA with hyperparameter β_2,

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\, g_t g_t^\top. \tag{19}$$

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method [17]. As m_t and v_t are initialized as vectors of 0, the authors of Adam observe that they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small (i.e. β_1 and β_2 are close to 1). Thus, they counteract these biases by computing bias-corrected first and second moment estimates:

$$\begin{aligned} \widehat{m}_t &= \frac{m_t}{1-\beta_1^t}, \\ \widehat{v}_t &= \frac{v_t}{1-\beta_2^t}. \end{aligned} \tag{20}$$

Finally, we can get the standard update rule of Adam:

$$\mathcal{O}_{Adam} := \theta_{t+1} \leftarrow \theta_t - \frac{\eta \widehat{m}_t}{\sqrt{\widehat{v}_t + \epsilon}}, \tag{21}$$

where the default values of 0.9 for β_1, 0.999 for β_2, and 10^{-8} for ϵ are proposed by the authors.

E. Extension and Conclusion

1) Extension: In addition to the classical SGD-like optimizers mentioned above, more than hundreds of SGD-like optimization methods have been proposed in the field of deep learning in the past two decades [34]. Some of these methods are also widely known, such as Adadelta [35], AdamMax [33], Nadam [36], AdamW [28] etc. Of these, all the rest are improved versions of Adam, except for Adadelta which is about the improvement of AdaGrad. We will briefly introduce three improvements to Adam, and we will explain the update rule of Adadelta in more detail due to its specificity.

AdamMax: The authors of Adam regard the v_t term in Adam as the ℓ_2 norm of the past gradients and current gradient. Then they find generalize this update to the ℓ_∞, models converge to the more stable. This method is termed as AdamMax [33].

Nadam: We have also seen that Nesterov accelerated gradient (NAG) is superior to vanilla momentum in Section III-B2. Inspired by this, Nadam (Nesterov-accelerated Adaptive Moment Estimation) [36] thus combines Adam and NAG, and modify the original momentum term m_t.

AdamW: Loshchilov et al. [28] demonstrate L_2 regularization and weight decay regularization are equivalent for standard SGD, but this is not the case for adaptive gradient algorithms, such as Adam. Thus, AdamW, which decouple the weight decay from the optimization steps, is proposed to tackle this issue.

Adadelta: First, we define the $\sqrt{R_t} + \epsilon$ of Equation 17 is the root mean squared (RMS) error criterion of the gradient,$RMS\left[g\right]_t$:

$$RMS\left[g\right]_t = \sqrt{R_t} + \epsilon. \tag{22}$$

Similarly, we can get the root mean squared error of parameter updates $\Delta\theta_t$:

$$\begin{aligned} E\left[\Delta\theta^2\right]_t &= \mu E\left[\Delta\theta^2\right]_{t-1} + (1-\mu)\,\Delta\theta_t^2 \\ RMS\left[\Delta\theta\right]_t &= \sqrt{E\left[\Delta\theta^2\right]_t} + \epsilon. \end{aligned} \tag{23}$$

With them, we can get the standard update rule of Adadelta:

$$\begin{aligned} \Delta\theta_t &= -\frac{RMS\left[\Delta\right]_{t-1}}{RMS\left[g\right]_t} g_t \\ \theta_{t+1} &= \theta_t + \Delta\theta_t \end{aligned} \tag{24}$$

Unlike most other adaptive learning rate methods, ***Adadelta does not require setting any learning rate, including the initial learning rate***, as it has been eliminated from the update rule. For more details about its theoretical analysis, see Paper [35].

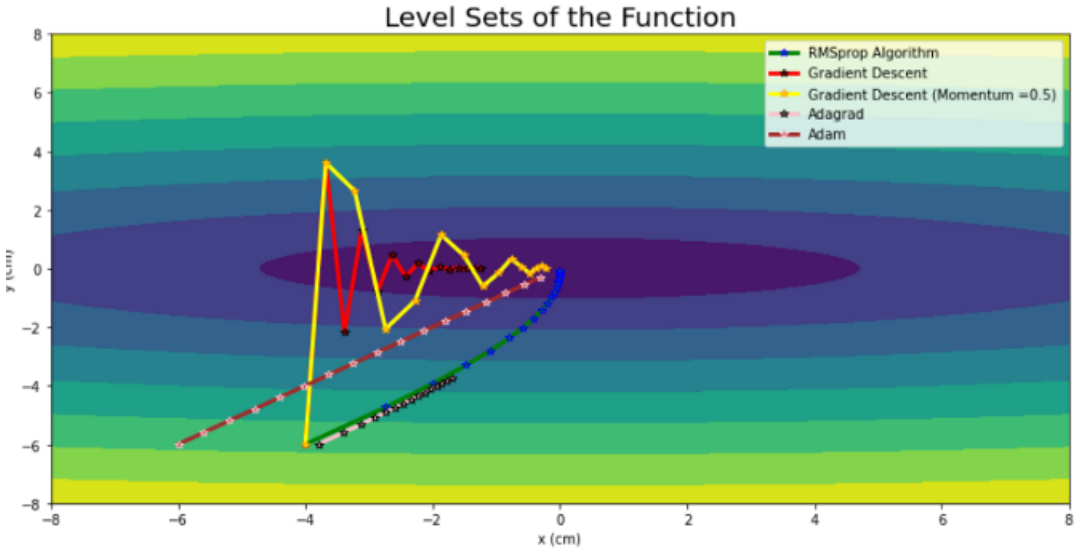
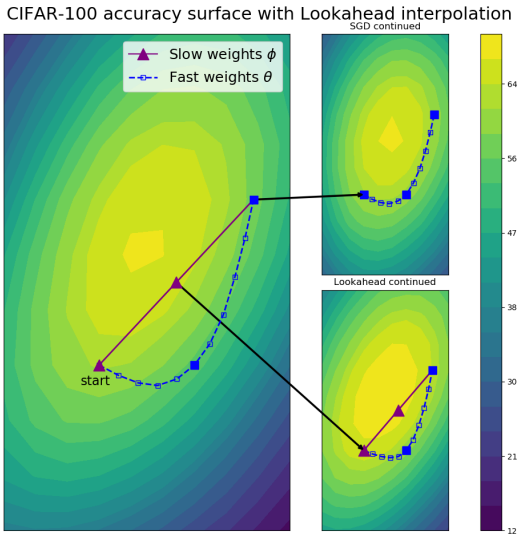


Fig. 7: Trajectory during Convergence for different SGD-like methods. Best viewed in color. Source: [37].

2) Conclusion: In summary, RMSprop is an extension of Adagrad that deals with its radically diminishing learning rates. It is identical to Adadelta, except that Adadelta uses the RMS of parameter updates in the numerator update rule. Adam, finally, adds bias-correction and momentum to RMSprop. Insofar, RMSprop, Adadelta, and Adam are very similar algorithms that do well in similar circumstances. Kingma et al. [33] show that its bias-correction helps Adam slightly outperform RMSprop towards the end of optimization as gradients become sparser. Insofar, Adam might be the best overall choice [17].

IV. Lookahead and look ahead

Although a large number of SGD-like methods address the shortcomings of vanilla SGD in two ways: a) adaptive learning rate schemes, such as AdaGrad and Adam, and (2) accelerated schemes, such as Momentum and Nesterov Momentum. However, the scholars concerned did not stop there and continued their pursuit of a modern optimizer with faster convergence, greater robustness, and fewer hyperparameters. In this section, we present the new work Lookahead [2] proposed by Hinton et al. in 2019 and one of the latest directions, the multi-optimizer combination.



Algorithm 1 Lookahead Optimizer:

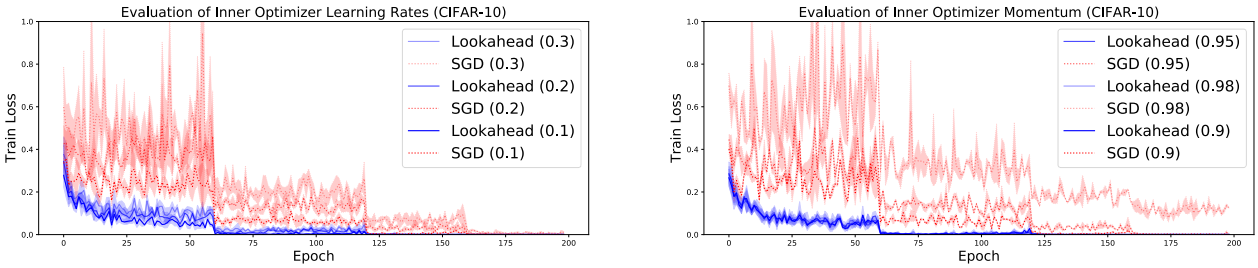
Require: Initial parameters θ_0, cost functionℓ
Require: Synchronization period k, slow weights step size η, optimizer $\mathcal{O}$
for $t = 1, 2, \cdots$ **do**
  Synchronize parameters $\phi_{t,0} \leftarrow \theta_{t-1}$
  for $i = 1, 2, \cdots, k$ **do**
    Sample minibatch of data $d \sim \mathcal{D}$
    $\phi_{t_i} \leftarrow \phi_{t,i-1} + \mathcal{O}\left(\ell, \phi_{t,i-1}, d\right)$
  end for
  Perform outer update $\theta_t \leftarrow \theta t - 1 + \eta\left(\phi_{t,k} - \theta_{t-1}\right)$
end for
return parameters θ

Fig. 8: (Left) Visualizing Lookahead (k = 10) through a ResNet-32 test accuracy surface at epoch 100 on CIFAR-100 [38] (an image classification dataset). The authors project the weights onto a plane defined by the first, middle, and last fast (inner-loop) weights. The fast weights are along the blue dashed path. All points that lie on the plane are represented as solid, including the entire Lookahead slow weights path (in purple). Lookahead (middle, bottom right) quickly progresses closer to the minima than SGD (middle, top right) is able to. (Right) Pseudocode for Lookahead. Best viewed in color. Source: [2].

A. Lookahead Optimizer: k steps forward, 1 step back

As shown in Figure 8, Hinton et al. revisited the SGD optimization process and found that its greatest blocking was still hovering around the optimal point and that the optimizer was more blind without a clearer global direction. Naturally, the authors came up with a Nesterov-like approach that allows the optimizer to take a *big jump* first and then *correct* the direction. However, unlike Nesterov Momentum, the Lookahead just takes *one big jump* but generalizes to k-steps. In addition, the accumulated k-steps are updated as corrected using EMA, from *"big jump + correction"* of Nesterov to "EMA $\{\text{big jump}, \text{big jump}, ...\}_k$". Algorithm 1 shows the specific algorithm procedure of Lookahead. Therefore, as shown in Figure 8, Lookahead can see farther and select the best direction from it, alleviating the blindness of vanilla SGD. Moreover, Lookahead can easily replace the internal optimizer, unlike Adam adapting Nesterov, which requires modifying the update steps of Adam [36]. With these features, Lookahead can converge faster and better than the internal optimizer and is more robust to parameter perturbations.

Specific experiments can be seen in Figure 9.



(a) CIFAR-10 Train Loss: Different LR

(b) CIFAR-10 Train Loss: Different momentum

Fig. 9: The authors fix Lookahead parameters and evaluate on different inner optimizers.. Best viewed in color. Source: [2].

B. Multi-optimizer combination

As optimizers are critical to the performances of neural networks, every year a large number of papers innovating on the subject are published. However, while most of these publications provide incremental improvements to existing algorithms, they tend to be presented as new optimizers rather than composable algorithms. Thus, many worthwhile improvements are rarely seen outside of their initial publication.

Ranger20: Based on this, Wright et al. built Ranger20 [39] in 2019[2]. Ranger20 is a synergistic optimizer combining RAdam (Rectified Adam) [42] and Lookahead [2], and GC (gradient centralization) [44] in one optimizer. Their team used the Ranger optimizer in *capturing 12 leaderboard records on the FastAI global leaderboards* [45].

Ranger21: Moreover, in [1], Wright et al. proposed Ranger21, which combines adamW [28], Lookahead and seven tricks. The experiments demonstrate that Ranger21 provides significantly improved validation accuracy and training speed, smoother training curves, and is even able to train a ResNet50 on ImageNet2012 without Batch Normalization layers.

RangerLars: Inspired by Ranger, Grankin et al. propose RangerLars [46], which combines RAdam, Lookahead, and LARS [41], to better adapt to the trend of deep learning with big data, big models, and large-scale training.

[2]Note: Multi-optimizer combination is one of the state-of-the-art directions in the field of deep learning optimization methods, and not the only one. There is still a lot of outstanding work exploring how to improve the fundamental SGD-like optimizer, such as: AmsGrad [40], LARS [41], RAdam [42]. More latest optimizers are available in [43].

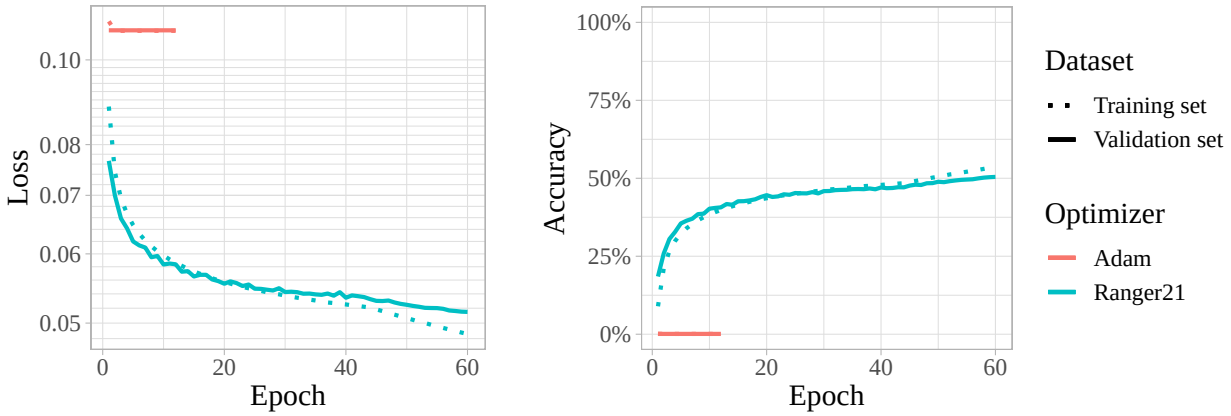


Fig. 10: Evolution of the loss and accuracy, using either Adam or Ranger21, when training a Normalizer-Free ResNet50 convolutional neural network on the ImageNet dataset. The y axis of the loss is displayed in logarithmic scale while the y axis of the accuracy is in percents. Adam was stopped prematurely at 10 epochs as it was not converging. Best viewed in color. Source: [1].

C. Future: more optimizer is better?

Lack of theoretical proofs: Despite the excellent experimental results obtained by Lookahead and some combinatorial-based optimizers, there is a great lack of detailed theoretical proofs. Zhou et al. prove why the Lookahead works well in [47], but most of the latest optimizers prefer to use experimental data to prove their effectiveness rather than traditional mathematical proofs.

How to choose components? The selection of components for the combination seems to be more of empirical analysis and lacks a criterion. This may lead to subsequent additions of components that may require extensive experimentation and, similarly, we can not ensure their general availability.

Better is really better? Sivaprasad et al. propose that the performance of optimizers, particularly in deep learning, depends considerably on their chosen hyperparameter configuration in [48]. And they argue that a fair assessment of optimizers' performance must take the computational cost of hyperparameter tuning into account, i.e., how easy it is to find good hyperparameter configurations using an automatic hyperparameter search. Thus, the practicality of the optimizer may be more reliable than its apparent performance. Moreover, this may also explain why in some of the latest deep learning downstream tasks, there are still many SOTA approaches that

use SGDs with Momentum to train their models.

V. Conclusion

This article aims to help readers briefly understand the development of optimizers in deep learning, and the design motivation and specific update rules of commonly used optimizers. It also briefly introduces some recent work in the field. The following conclusions are derived:

- Most of the latest optimizers prefer to use experimental data to prove their effectiveness rather than traditional mathematical proofs.
- In the case of a small number of tuning parameters, the method with an adaptive learning rate represented by Adam tends to perform better.
- Tuning parameters are very important for the optimizer, and a well-tuned SGD still performs well in many downstream tasks.
- An optimizer with detailed theoretical analysis and practicality is a good choice. Apparent superior performance can be misleading.

References

[1] L. Wright and N. Demeure, "Ranger21: a synergistic deep learning optimizer," *arXiv preprint arXiv:2106.13731*, 2021.

[2] M. Zhang, J. Lucas, J. Ba, and G. E. Hinton, "Lookahead optimizer: k steps forward, 1 step back," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[3] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[4] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[6] 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.

[7] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[10] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *arXiv preprint arXiv:2107.13586*, 2021.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[13] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[14] A. D. Rasamoelina, F. Adjailia, and P. Sinčák, "A review of activation function for artificial neural network," in *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*. IEEE, 2020, pp. 281–286.

[15] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 315–323.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[17] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

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

[19] D. Saad, "Online algorithms and stochastic approximations," *Online Learning*, vol. 5, pp. 6–3, 1998.

[20] Wikipedia contributors, "Stochastic gradient descent — Wikipedia, the free encyclopedia," 2022, [Online; accessed 15-May-2022]. [Online]. Available: https://en.wikipedia.org/wiki/Stochastic_gradient_descent

[21] 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," *Advances in neural information processing systems*, vol. 27, 2014.

[22] R. Sutton, "Two problems with back propagation and other steepest descent learning procedures for networks," in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society, 1986*, 1986, pp. 823–832.

[23] "Momentum and learning rate adaptation," 2022, [Online; accessed 15-May-2022]. [Online]. Available: https://www.willamette.edu/~gorr/classes/cs449/figs/valley2.gif

[24] Y. Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2)," in *Doklady an ussr*, vol. 269, 1983, pp. 543–547.

[25] Hinton, Geoffrey E, "Lecture 6 of neural networks for machine learning," 2022, [Online; accessed 15-May-2022]. [Online]. Available: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[26] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.

[27] 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.

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

[29] Wikipedia contributors, "Subderivative — Wikipedia, the free encyclopedia," 2022, [Online; accessed 25-May-2022]. [Online]. Available: https://en.wikipedia.org/wiki/Subderivative#The_subgradient

[30] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, "Large scale distributed deep networks," *Advances in neural information processing systems*, vol. 25, 2012.

[31] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[32] Daniel Villarraga (SYSEN 6800 Fall 2021), "Adagrad," 2022, [Online; accessed 25-May-2022]. [Online]. Available: https://optimization.cbe.cornell.edu/index.php?title=AdaGrad

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

[34] R. M. Schmidt, F. Schneider, and P. Hennig, "Descending through a crowded valley-benchmarking deep learning optimizers," in *International Conference on Machine Learning*. PMLR, 2021, pp. 9367–9376.

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

[36] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

[37] Saket Thavanani, "Deep learning optimization algorithms using numpy," 2022, [Online; accessed 25-May-2022]. [Online]. Available: https://ai.plainenglish.io/comparative-performance-of-deep-learning-optimization-algorithms-using-numpy-e9f63b908c7f

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

[39] L. Wright, "Ranger - a synergistic optimizer." https://github.com/lessw2020/Ranger-Deep-Learning-Optimizer, 2019.

[40] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," *arXiv preprint arXiv:1904.09237*, 2019.

[41] Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh, "Large batch optimization for deep learning: Training bert in 76 minutes," *arXiv preprint arXiv:1904.00962*, 2019.

[42] 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.

[43] SirRob1997, "Crowded-valley—results," 2022, [Online; accessed 31-May-2022]. [Online]. Available: https://github.com/SirRob1997/Crowded-Valley---Results#list-of-optimizers

[44] H. Yong, J. Huang, X. Hua, and L. Zhang, "Gradient centralization: A new optimization technique for deep neural networks," in *European Conference on Computer Vision*. Springer, 2020, pp. 635–652.

[45] Wright, Less, "How we beat the fastai leaderboard score by +19.77synergy of new deep learning techniques for your consideration." 2022, [Online; accessed 31-May-2022]. [Online]. Available: https://lessw.medium.com/how-we-beat-the-fastai-leaderboard-score-by-19-77-a-cbb2338fab5c

[46] mgrankin, "over9000," 2022, [Online; accessed 31-May-2022]. [Online]. Available: https://github.com/mgrankin/over9000

[47] P. Zhou, H. Yan, X. Yuan, J. Feng, and S. Yan, "Towards understanding why lookahead generalizes better than sgd and beyond," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[48] P. T. Sivaprasad, F. Mai, T. Vogels, M. Jaggi, and F. Fleuret, "Optimizer benchmarking needs to account for hyperparameter tuning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 9036–9045.