An overview of Optimization Algorithms for Deep Learning and Neural Networks

Mert Canatan
Department of Electrical and Computer Engineering
University of Florida
Gainesville, FL 32611
mert.canatan@ufl.edu

1 Introduction

Optimization is a critical component in deep learning. I think optimization for neural networks is an interesting topic for theoretical research due to various reasons. First, its tractability despite non-convexity is an intriguing question and may greatly expand our understanding of tractable problems. Second, classical optimization theory is far from enough to explain many phenomena. Therefore, I would like to understand the challenges and opportunities from a theoretical and practical perspective by reviewing the existing research in this field. [9]

1.1 Related Work

Most popular optimization algorithms for training deep learning and neural networks use first-order gradient methods and these algorithms can be generally categorized as accelerated schemes such as heavy-ball method [2], Nesterov accelerated gradient (NAG) [3], and stochastic gradient descent (SGD) with momentum [4]; and adaptive methods such as Adagrad [5], AdaDelta [6], RMSProp [7], and Adam [8]. For many models such as convolutional neural networks (CNNs), adaptive methods typically converge faster but generalize worse compared to accelerated schemes (e.g., SGD).

1.2 This Paper

First-order gradient method optimization algorithms for training neural networks are reviewed and some of the algorithms: SGD with momentum and Adam are implemented on CIFAR 10 dataset [15]. Both the advantages and the disadvantages of these algorithms (accelerated scheme and adaptive methods) are discussed. Also, AdaBelief optimizer [1] is reviewed and implemented because it has a good generalization as in accelerated schemes, and fast convergence as in adaptive methods. The intuition behind the AdaBelief is to adapt the stepsizes according to the "belief" in the current gradient direction. Viewing the exponential moving average (EMA) of the noisy gradient as the prediction of the gradient at the next time step, if the observed gradient greatly deviates from the prediction, current observation is distrusted and and a small step is taken; if the observed gradient is close to the prediction, it is trusted and large step is taken.

The organization of this paper is as follows: Formulation of neural network as an optimization problem and its relation to classical machine learning and optimization problems. Algorithms to solve the formulated optimization problem. Experiments to show the performance of different algorithms to solve the optimization problem. Concluding remarks and future work for the neural network optimization.

Preprint. Under review.

2 Problem Statement

In this paper, optimization for a supervised learning problem with feedforward neural network is considered to keep the problem formulation simple. Suppose we are given data points $x_i \in \mathbb{R}^{d_x}$, $y_i \in \mathbb{R}^{d_y}$, $i = 1, \ldots, n$, where n is the number of samples. The input instance x_i can represent a feature vector of an object, an image, a vector that represents a word, etc. The output instance y_i can represent a real-valued vector or scalar such as in a regression problem, or an integer-valued vector or scalar such as in a classification problem.

The aim is to predict y_i based on the information of x_i, therefore we want to learn the underlying mapping that maps each x_i to y_i. To approximate the mapping $f_\theta : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$ neural network is used, which maps input x to a predicted output $\hat{y}$. A standard fully connected neural network is given by

$$f_\theta(x) = W^L \phi(W^{L-1} \ldots \phi(W^2 \phi(W^1 x))), \tag{1}$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is the neuron activation function, W^j is a matrix of dimension $d_j \times d_{j-1}$, $j = 1, \ldots, L$ and $\theta = (W^1, \ldots, W^L)$ represents the collection of all parameters. In here, $d_0 = d_x$ and $d_L = d_y$. When applying the scalar function ϕ to a matrix Z, θ is applied to each entry of Z. Another way to write down the neural network is to use a recursion formula:

$$z^0 = x; \quad z^l = \phi(W^l z^{l-1}),\ l = 1, \ldots, L. \tag{2}$$

In practice, the recursive expression should be $z^l = \phi(W^l z^{l-1} + b^l)$, where b^l is the "bias" term. However, in this paper the bias term is omitted for the simplicity.

The objective of picking a parameter for neural network is making the predicted output $\hat{y_i} = f_\theta(x_i)$ close to the true output y_i, therefore we want to minimize the distance between y_i and $\hat{y_i}$. For a certain distance metric $\ell(\cdot, \cdot)$, the problem of finding the optimal parameters can be written as

$$\underset{\theta}{\text{minimize}} \quad F(\theta) \triangleq \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f_\theta(x_i)). \tag{3}$$

For regression problems, $\ell(y, z)$ is often chosen to be quadratic loss function $\ell(y, z) = \|y - z\|^2$. For binary classification problem, a popular choice of ℓ is $\ell(y, z) = \log(1 + \exp(-yz))$.

Technically, the neural network given by (2) should be called fully connected feed-forward networks (FCN). However neural networks used in practice often have more complicated structure. For computer vision tasks, Convolutional Neural Networks (CNNs) are standard. In natural language processing, extra layers such as "attention" are commonly added. Nonetheless, for our purpose of understanding the optimization problem, FCN model will be discussed and few cases will be mentioned for CNN for results.

For better understanding of the neural network optimization problem 7, several classical optimization problems should be considered.

2.1 Relation of Neural Networks with Least Squares

One special form of neural network formulation 7 is the linear regression problem (least squares):

$$\underset{w \in \mathbb{R}^{d \times 1}}{\text{minimize}} \quad \|y - w^\top X\|^2, \tag{4}$$

where $X = (x_1, \ldots, x_n) \in \mathbb{R}^{d \times n}$, $y \in \mathbb{R}^{1 \times n}$. If there is only one linear neuron that maps the input x to $w^\top x$ and the loss function is quadratic, then the general neural network problem 7 reduces to the least squares problem (4). The least squares problem is mentioned here for two reasons:

- It is one of the simplest forms of a neural network problem.
- When analyzing neural network optimization performance, researchers use least squares as a benchmark for comparison purposes.

2.2 Relation of Neural Networks with Matrix Factorization

Neural network optimization 7 is closely related to a fundamental problem in numerical computation: matrix factorization. If there is only one hidden layer of linear neurons and the loss function is quadratic, and the input data matrix X is the identity matrix, the neural network problem 7 reduces to

$$\underset{W_1,W_2}{\text{minimize}} \quad \|Y - W_2W_1\|_F^2, \tag{5}$$

where $W_2 \in \mathbb{R}^{d_y \times d_1}$, $W_1 \in \mathbb{R}^{d_1 \times n}$, $Y \in \mathbb{R}^{d_y \times n}$ and $\|\cdot\|_F$ indicates the Frobenious norm of a matrix. If $d_1 < \min\{n, d_y\}$, then the above problem gives the best rank-d_1 approximation of the matrix Y. Matrix factorization is widely used in engineering, and it has many popular extensions such as non-negative matrix factorization [10] and low-rank matrix completion. Neural network can be viewed as an extension of two-factor matrix factorization to multi-factor nonlinear matrix factorization. [13]

3 Algorithms

The goals of algorithm design for neural network optimization are at least two-fold: first, converge faster; second, improve certain metric of interest. The metrics of interest can be very different from the optimization loss, and is often measured on unseen data. a faster method does not necessarily generalize better, and not necessarily improves the metric of interest. Due to this gap, a common algorithm design strategy is: try an optimization idea to improve the convergence speed, but only accept the idea if it passes a certain "perfomance check". [13] In this section, first-order gradient based optimization algorithms that are commonly used in deep learning will be discussed. We can categorize these algorithms in two ways: One is the "Accelerated Schemes", and the other is "Adaptive Methods". "Accelerated Schemes" known for their generalization, and the "Adaptive Methods" known for their convergence speed. One specific algorithm for each method will be provided and newly introduced AdaBelief optimizer [1] will be mentioned since they propose to take advantage of fast convergence (as in Adaptive Methods) while having good generalization (as in Accelerated Schemes).

3.1 Stochastic Gradient Descent (SGD) with Momentum

A large class of methods for neural network optimization are based on gradient descent (GD). The basic form of GD is

$$\theta_{t+1} = \theta_t - \alpha_t \nabla F(\theta_t), \tag{6}$$

where α_t is the step-size or learning rate and $\nabla F(\theta_t)$ is the gradient of the loss function for the t-th iterate.

We can write 3 as a finite-sum optimization problem:

$$\underset{\theta}{\text{minimize}} \quad F(\theta) \triangleq \frac{1}{B}\sum_{i=1}^{B} F_i(\theta). \tag{7}$$

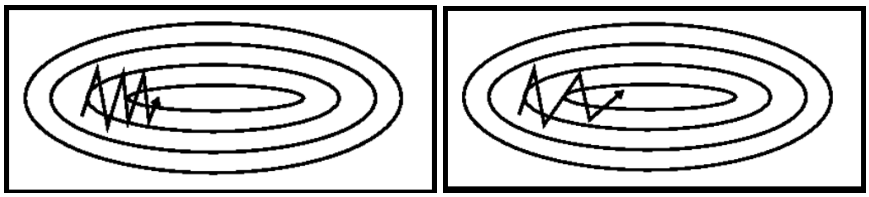
Each $F_i(\theta)$ represents the sum of training loss for a mini-batch of training sample (e.g., 32, 64, or 512 samples), and B is the total number of mini-batches (smaller than the total number of training samples n). The exact expression of F_i does not matter in this section, as we only need to know how to compute the gradient $\nabla F_i(\theta)$. [13]

A more practical variant of GD is the Stochastic Gradient Descent (SGD). In SGD, at the t-th iteration, randomly pick i and update the parameter by

$$\theta_{t+1} = \theta_t - \alpha_t \nabla F_i(\theta_t), \tag{8}$$

where $F_i(\theta) \triangleq \ell(y_i, f_\theta(x_i))$. SGD has trouble navigating areas where the surface curves much more steeply in one dimension than in another, which are common around local optima. In these scenarios, SGD cannot make a hesitant progress as can be seen in Fig. 1a.

Momentum is a method that helps accelerate SGD in the relevant direction and dampens oscillations as can be seen in the Fig. 1b.



(a) SGD without momentum **(b)** SGD with momentum

Figure 1: SGD oscillations without momentum and with momentum

Essentially, when using momentum, we push a ball down a hill. The ball accumulates momentum as it rolls downhill, becoming faster and faster on the way. The same thing happens to parameter updates: The momentum term increases for dimensions whose gradient points in the same directions and reduce updates for dimensions whose gradients change directions. As a result, faster convergence and reduced oscillations are gained in this method. [12]

Algorithm 1 Stochastic Gradient Descent (SGD) with momentum

Input: Training Set $\mathcal{T}$; Learning Rate α; Normal Distribution Std: σ; Momentum Term Weight: ρ
Output: Model Parameters $\boldsymbol{\Theta}$
Initialize parameters with Normal distribution $\boldsymbol{\Theta} \sim \mathcal{N}(0, \sigma^2)$, convergence $tag = False$, Initialize Momentum term $\Delta \boldsymbol{v} = 0$
while *tag* == *False* **do**
  Shuffle the training set $\mathcal{T}$
  for *each data instance* $(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{T}$ **do**
    Compute $\hat{\boldsymbol{\Theta}} = \boldsymbol{\Theta} - \gamma \cdot \rho \cdot \Delta \boldsymbol{v}$
    Compute gradient $\nabla_{\boldsymbol{\Theta}} \mathcal{L}(\hat{\boldsymbol{\Theta}}; (\boldsymbol{x}_i, \boldsymbol{y}_i))$ on training instance $(\boldsymbol{x}_i, \boldsymbol{y}_i)$
    Update term $\Delta \boldsymbol{v} = \rho \cdot \Delta \boldsymbol{v} = (1 - \rho) \cdot \nabla_{\boldsymbol{\Theta}} \mathcal{L}(\hat{\boldsymbol{\Theta}}; (\boldsymbol{x}_i, \boldsymbol{y}_i))$
    Update variable $\boldsymbol{\Theta} = \boldsymbol{\Theta} - \gamma \cdot \Delta \boldsymbol{v}$
    if *convergence condition holds* **then**
      $tag = True$
    end
  end
end
return model variable $\boldsymbol{\Theta}$

3.2 Adam

Adaptive Moment Estimation (Adam) is a method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t, it also keeps an exponential decaying average of past gradients m_t [12], similar to momentum:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \end{aligned} \tag{9}$$

where m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively. As m_t and v_t are initialized as vectors of 0's, 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).

The authors counteract these biases by computing bias-corrected first and second moment estimates:

$$\begin{aligned} \widehat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \widehat{v}_t &= \frac{v_t}{1-\beta_2^t} \end{aligned} \tag{10}$$

They then use these to update the parameters, which yields the Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\widehat{v}_t} + \epsilon} \widehat{m}_t \tag{11}$$

The authors propose default values of 0.9 for β_1, 0.999 for β_2, and 10^{-8} for ϵ. They show empirically that Adam works well in practice and compares favorably to other adaptive learning-method algorithms.

Notations By the convention in [8], the following notations are used for this section and the next section:

- $f(\theta) \in \mathbb{R}$, $\theta \in \mathbb{R}^d$: 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 EMA of g_t^2, s_t is the EMA of $(g_t - m_t)^2$
- 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., $\beta_{1t} = \beta_1$, $\beta_{2t} = \beta_2$, $\forall t \in \{1, 2, \ldots, T\}$)s

Algorithm 2 Adam Optimizer

Initialize $\theta_0, m_0 \leftarrow 0, v_0 \leftarrow 0, t \leftarrow 0$
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
  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
  $\theta_t \leftarrow \prod_{\mathcal{F}, \sqrt{\widehat{v}_t}} (\theta_{t-1} - \frac{\alpha \widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon})$
end

3.3 AdaBelief Optimizer

AdaBelief optimizer propose a good generalization as in accelerated schemes, and fast convergence as in adaptive methods. The intuition behind the AdaBelief is to adapt the stepsizes according to the "belief" in the current gradient direction. Viewing the exponential moving average (EMA) of the noisy gradient as the prediction of the gradient at the next time step, if the observed gradient greatly deviates from the prediction, current observation is distrusted and and a small step is taken; if the observed gradient is close to the prediction, it is trusted and large step is taken. AdaBelief can be easily modified from Adam. Denote the observed gradient at step t as g_t and its exponential moving average (EMA) as m_t. Denote the EMA of g_t^2 and $(g_t - m_t)^2$ as v_t and s_t, respectively. m_t is divided by $\sqrt{v_t}$ in Adam, while it is divided by $\sqrt{s_t}$ in AdaBelief. Intuitively, $\frac{1}{\sqrt{s_t}}$ is the "belief" in the observation: viewing m_t as the prediction of the gradient, if g_t deviates much form m_t, there

is a weak belief in g_t, and take a small step; if g_t is close to the prediction m_t, there is a strong belief in g_t, and take a large step.

Algorithm 3 AdaBelief Optimizer

Initialize $\theta_0, m_0 \leftarrow 0, s_0 \leftarrow 0, t \leftarrow 0$
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$s_t \leftarrow \beta_2 s_{t-1} + (1 - \beta_2)(g_t - m_t)^2$
Bias Correction
$\widehat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$, $\widehat{s}_t \leftarrow \frac{s_t+\epsilon}{1-\beta_2^t}$
Update
$\theta_t \leftarrow \prod_{\mathcal{F}, \sqrt{\widehat{s}_t}} (\theta_{t-1} - \frac{\alpha \widehat{m}_t}{\sqrt{\widehat{s}_t}+\epsilon})$
end

Adam and AdaBelief are summarized in 2 and 3, where all operations are element-wise, with difference marked in red. Note that no extra parameters are introduced in AdaBelief. Specifically, in Adam, the update direction is $m_t/\sqrt{v_t}$, where v_t is the EMA of g_t^2; in AdaBelief, the update direction is $m_t/\sqrt{s_t}$, where s_t is the EMA of $(g_t - m_t)^2$. Intuitively, viewing m_t as the prediction of g_t, AdaBelief takes a large step when observation g_t is close to prediction m_t, and a small step when the observation greatly deviates from the prediction. $\widehat{\cdot}$ represents bias-corrected value. Note that an extra ϵ is added to s_t during bias-correction, in order to better match the assumption that s_t is bounded below (the lower bound is at least ϵ).

4 Experiments

Performance comparison between SGD with momentum, Adam, and AdaBelief in terms of convergence and accuracy is performed. CIFAR 10 dataset [15] and VGG 11 neural network arhitecture [16] is selected. CIFAR 10 is a dataset that contains $60,000$ 32×32 color images in 10 different classes and it is widely used in machine learning and computer vision. These 10 different classes represent airlines, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are $6,000$ images for each class. VGG 11 is a sophisticated Convolutional Neural Network (CNN) architecture and can be seen in the Fig. 2. [1]

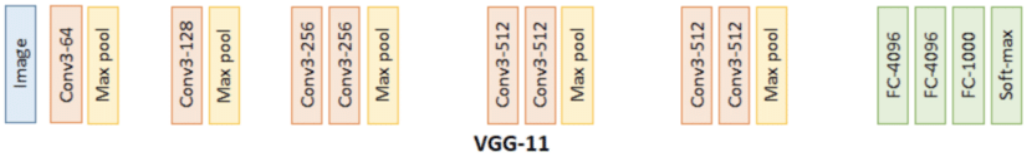


Figure 2: VGG 11 Architecture

4.1 Hyperparameter Tuning

The following hyperparameters are used [1]:

- *SGD with momentum*: Set the momentum as 0.9, which is default for many networks. Search learning rate among $\{10, 1, 0.1, 0.01, 0.001\}$,
- *Adam*: Search for optimal β_1 among $\{0.5, 0.6, 0.7, 0.8, 0.9\}$, searched for α as in SGD, and set other parameters as their own default values in the literature.
- *AdaBelief*: Used default parameters of Adam: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $\alpha = 10^{-3}$.

[1] `https://www.researchgate.net/publication/336550999_Model_Fusion_via_Optimal_Transport/figures?lo=1`

4.2 CNN on Image Classification

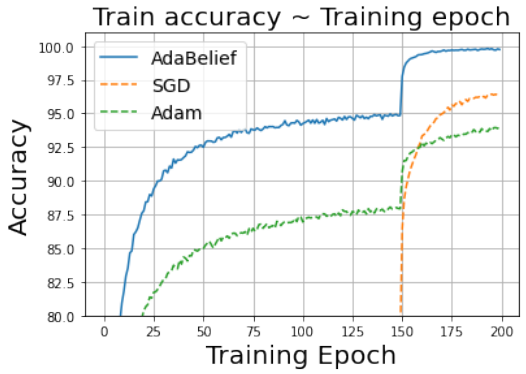
Experiments were performed using VGG 11 CNN architecture on CIFAR 10 dataset. *Official implementation* of AdaBound [11] is implemented, hence achieved *exact replication* of [11] in Adam and SGD with momentum case. For each optimizer, optimal hyperparameters are searched, and reported the mean and the standard deviation of test-set accuracy (under optimal hyperparameters) for 3 runs with random initialization.

In Fig. 3, AdaBelief achieves fast convergence as in adaptive methods such as Adam while achieving better accuracy than SGD with momentum. For all experiments, the model is trained for 200 epoch with a batch size of 128, and the learning rate is multiplied by 0.1 at epoch 150.

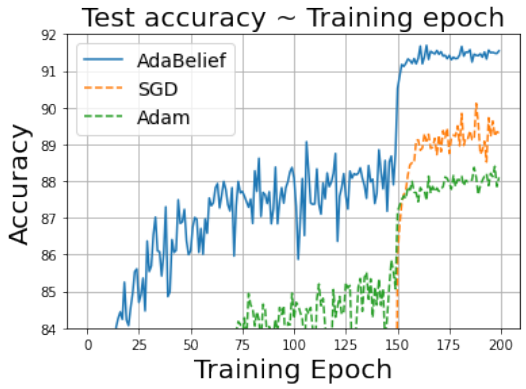
By using classification _ cifar10 under PyTorch Experiments on [2] the model is trained in 200 epoch with the parameters under the Hyperparameter Tuning section. Curves of the trained models are saved in a file called curves and the following Figure 3b are plotted using those curves for different optimizers. Table 1 shows the best test accuracy for three optimizers.

SGD with momentum (%)	Adam (%)	AdaBelief (%)
90.08	88.36	**91.63**

Table 1: Best test set accuracy comparison between SGD with momentum, Adam, and AdaBelief. In Image Classification task AdaBelief outperforms both SGD with momentum and Adam in terms of both convergence speed and the highest accuracy.



(a) Training Accuracy vs. Epoch



(b) Test Accuracy vs. Epoch

Figure 3: Comparison of SGD with momentum, Adam and AdaBelief on Training and Test Accuracy using VGG 11 as a CNN architecture. Blue solid line is AdaBelief, orange dashed line is SGD with momentum and green dashed line is Adam. AdaBelief receives the highest accuracy on both training and test sets.

[2] `https://github.com/juntang-zhuang/Adabelief-Optimizer`

5 Conclusion

In this project, first-order gradient based optimization for deep learning is reviewed. Started with a problem formulation in optimization perspective, and then talked about the algorithms to solve this problem. There are two different approaches in these algorithms: First is the "Accelerated Schemes" such as Stochastic Gradient Descent (SGD) with momentum, and the second is "Adaptive Methods" such as Adam. A novel algorithm called "AdaBelief" is mentioned which intuitively proposes an algorithm for taking the best of two methods (fast convergence as adaptive methods and good generalization as accelerated schemes). I implemented SGD with momentum, Adam, and AdaBelief on CIFAR 10 dataset using VGG 11 CNN architecture with the default parameters for all methods.

In future, three different direction can be considered developing an optimization algorithms for deep learning and neural networks. First is the training stability of these first order gradient based optimization algorithms. Especially for the complex neural network structures such as Generative Adversarial Networks (GANs) [14] training stability can be analyzed and more stable optimization methods can be developed. Second is the second-order gradient based optimization algorithms for training neural network can be utilized for practical purposes to achieve better optimal points with faster convergence like in first-order gradient methods. Third is the global optimization of neural networks (GONs) [13]. Finding a global optimal solution for neural networks would give an answer to most of the problems in a lot of applications. For developing global optimization for neural networks, other non-convex optimization methods such as Matrix/Tensor Factorization can be pioneer since they are mature.

References

[1] Juntang Zhuang, Tommy Tang, Sekhar Tatikonda, Nicha Dvornek, Yifan Ding, Xenophon Papademetris, James S. Duncan "Adabelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients," arXiv preprint arXiv:2010.07468, 2020.

[2] Boris 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.

[3] Yu Nesterov, "A method of solving a convex programming problem with convergence rate $o(1/\hat{k2})$," in Sov. Math. Dokl, 1983, vol. 27

[4] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton, "On the importance of initialization and momentum in deep learning," in International conference on machine learning, 2013, pp. 1139–1147.

[5] John Duchi, Elad Hazan, and Yoram Singer, "Adaptive subgradient methods for online learning and stochastic optimization," Journal of machine learning research, vol. 12, no. Jul, pp. 2121–2159, 2011.

[6] Matthew D Zeiler, "Adadelta: an adaptive learning rate method," arXiv preprint arXiv:1212.5701, 2012.

[7] Alex Graves, "Generating sequences with recurrent neural networks," arXiv preprint arXiv:1308.0850, 2013.

[8] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[9] Sun, R.-Y. (2020) Optimization for Deep Learning: An Overview. Journal of the Operations Research Society of China, 8, 249-294. https://doi.org/10.1007/s40305-020-00309-6

[10] Fu, X., Huang, K., Sidiropoulos, N. D., and Ma, W.-K. Nonnegative matrix factorization for signal and data analytics: Identifiability, algorithms, and applications. IEEE Signal Processing Magazine, 36:59–80, 2019.

[11] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun, "Adaptive gradient methods with dynamic bound of learning rate," arXiv preprint arXiv:1902.09843, 2019.

[12] Ruder, Sebastian. "An overview of gradient descent optimization algorithms." arXiv preprint arXiv:1609.04747 (2016).

[13] R. Sun, "Optimization for deep learning: theory and algorithms," arXiv preprint arXiv:1912.08957, 2019.

[14] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In Proceedings of NIPS, pages 2672– 2680, 2014. papers.nips.cc/paper/5423-generativeadversarial-nets.pdf.

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

[16] Karen Simonyan and Andrew Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014.