Convergence Analysis of Optimizers

Anthony Garcia (Kim Sewon)
Department of Mathematics
University of Seoul
prepared7913@gmail.com

***Abstract*—Using the right optimizer in a machine learning scenario is one of the most critical factors determining the performance of a model. SGDM, Adagrad, RMSProp, and Adam, which incorporate momentum terms into SGD or utilize dynamic learning rates, have been extensively utilized in various machine learning tasks. Despite the ability to empirically choose the most suitable optimizer for a given situation, previous research has not rigorously elucidated the role of momentum, and discussions regarding dynamic learning rates have been notably absent. In this study, I will present a novel unified perspective for understanding SGD, SGDM, Adagrad, RMSProp, and Adam, providing much deeper interpretations of these optimizers. I also will compare the convergence properties of the five optimizers on a well-structured mathematical example, using a polynomial function. Finally, In controlled experiments, I will empirically evaluate the formulas and roles of each optimizer, considering their performance in a given scenario.**

I. Introduction

The optimization Problem in deep learning arises from the non-convex nature of neural network structures. Machine learning and deep learning architectures may consist of tens of millions to hundreds of millions of parameters (weights), depending on the model. The objective function can be viewed as a function with a number of variables equal to the count of weights. For instance, if a deep learning model has 100 million weights, the corresponding objective function becomes a function with 100 million parameters. Consequently, comprehending the objective function of a complex deep learning model becomes practically unfeasible.

Commonly, we employ gradient descent-based methods for optimizing deep learning models, including SGDM, Adagrad, RMSProp, Adam, and others. The effectiveness of these optimizers in converging deep learning models has been empirically demonstrated. Nevertheless, the convergence of gradient descent is guaranteed only in convex scenarios, and a well-established understanding of its convergence in non-convex situations is still lacking. Other optimizers based on gradient descent face similar challenges, and some have been observed to diverge even in convex situations.

Although finding the global minimum in non-convex functions can be challenging, leveraging gradient descent-based optimizers appropriately enables optimization without becoming trapped in local minima. Therefore, when tackling optimization problems in deep learning, it is crucial to approach them as non-convex optimization problems. This entails considering local minima, plateaus, and global minima, with effective solutions involving various strategies such as diverse initialization methods, learning rate adjustments, and regularization.

Identify applicable funding agency here. If none, delete this.

II. Unified Perspective for Optimizers

In this section, I present some key consistent insights for optimizer based on gradient descent. We will first delve into the attempt to dynamically adjust the step size and direction in gradient descent, understanding the principles behind each optimizer that emerged from these endeavors. Ultimately, we will explore the notion that all the optimizers examined are special cases of Adam.

A. Gradient Descent

The gradient descent method follows the following formula. α represents the step size, and ∇f denotes the direction of the step.

$$\mathbf{x_{t+1}} = \mathbf{x_t} - \alpha \nabla \mathbf{f}(\mathbf{x}) \tag{1}$$

B. Stochastic Gradient Descent (Mini-Batch)

In mini-batch stochastic gradient descent, gradients are computed using a randomly chosen point from a small subset. However, to mathematically handle stochastic gradient descent, the random selection in the mini-batch is sometimes replaced with the arithmetic mean over the mini-batch. You can observe this concept in the following formula.

$$\mathbf{g_t} = \nabla \tilde{\mathbf{f}}_\mathbf{t}(\mathbf{x}) = \frac{\mathbf{1}}{\mathbf{t}} \sum_{\mathbf{i=1}}^{\mathbf{t}} \nabla \mathbf{f_i}(\mathbf{x_t}) \tag{2}$$

$$\mathbf{x_{t+1}} = \mathbf{x_t} - \alpha \mathbf{g_t} \tag{3}$$

A mini-batch involves dividing the entire dataset into small subsets. Mini-batch gradient descent offers various advantages such as computational efficiency, memory efficiency, improved generalization performance, and enhanced convergence speed. However, most importantly, the mini-batch stochastic gradient descent method introduces randomness to navigate through non-convex situations, providing the potential to escape local minima and discover the global minimum. By infusing probabilistic elements into gradient descent, which guarantees convergence only in convex situations, mini-batch stochastic gradient descent encourages effective exploration of optimal values even in non-convex scenarios.

C. Stochastic Gradient Descent with Momentum

In a physical context, momentum is the product of velocity and mass. In the optimization context, assuming v = 1, momentum represents mass. In other words, it signifies inertia. In the context of optimizers, the concept of momentum involves considering not only the current gradient value but also the previous gradient values. Using exponential moving averages, it is designed to assign smaller weights to values that are further away from the present moment, diminishing the influence of past values as they become more distant.

$$\mathbf{g_t} = \frac{\mathbf{1}}{\mathbf{t}}\sum_{\mathbf{i=1}}^{\mathbf{t}} \nabla \mathbf{f_i}(\mathbf{x_t}) \tag{4}$$

$$\mathbf{m_t} = \beta \mathbf{m_{t-1}} + (\mathbf{1}-\beta)\mathbf{g_t} \tag{5}$$

$$\mathbf{x_{t+1}} = \mathbf{x_t} - \alpha \mathbf{m_t} \tag{6}$$

Let's expand our definition of exponentially weighed averages for three consecutive elements of the new sequence m_t. Combining all of them together we get and then simplifying it a bit. It provides a little bit more intuition of how momentum works. Moreover, it can be noted that Stochastic Gradient Descent (SGD) is a special case of SGDM when $\beta = 1$.

$$\mathbf{m_t} = (\mathbf{0.081g_{t-2}} + \mathbf{0.09g_{t-1}}) + \mathbf{0.1g_t}(\beta = 0.9) \tag{7}$$

D. Adagrad

So far, we have examined cases where the learning rate is fixed. However, when the learning rate is fixed, a critical issue arises near the optimum known as oscillation. Adagrad was the first optimizer to address this problem by dynamically adjusting the learning rate as iterations progress. It starts with a large step size initially, and as iterations continue, the step size decreases favorably in a convex manner. The significance lies in the attempt to have a large step size initially that progressively decreases as it approaches the optimum. However, Adagrad has a fatal drawback where the learning rate decreases excessively as iterations progress. This issue will be confirmed through experiments presented later, demonstrating that this becomes a significant problem.

$$\mathbf{g_i^{(k)}} = \frac{\mathbf{1}}{\mathbf{t}}\sum_{\mathbf{j=1}}^{\mathbf{k}} \nabla \mathbf{f_i}(\mathbf{x_j}) \tag{8}$$

$$\mathbf{v_i^{(k)}} = \sum_{\mathbf{j=1}}^{\mathbf{k}} (\mathbf{g_i^{(j)}})^{\mathbf{2}} \tag{9}$$

$$\mathbf{x_i^{(k+1)}} = \mathbf{x_i^{(k)}} - \frac{\alpha}{\sigma + \sqrt{\mathbf{v_i^{(k)}}}}\mathbf{g_i^{(k)}} \tag{10}$$

Here, sigma is a very small value introduced to prevent the denominator from becoming zero. Adagrad is SGD with dynamic learning rate. We can see that SGD is special case of Adagrad in $v_i^{(k)} = 0$.

E. RMSProp

Adagrad drastically reduces the learning rate based on the cumulative squared sum of gradients. RMSProp was introduced to address the monotonically decreasing issue of Adagrad. To understand RMSProp, one must grasp the concept of Exponential Moving Average (EMA). While arithmetic mean considers the weights of all terms equally, EMA assigns exponentially decreasing weights to averages as they become more distant from the present moment. While Adagrad simply adds the squared sum of all weights up to the previous step, RMSProp uses EMA to reduce the impact. This helps decrease the rate at which weights diminish, allowing for more stable convergence.

$$\mathbf{g_i^{(k)}} = \frac{\mathbf{1}}{\mathbf{t}}\sum_{\mathbf{j=1}}^{\mathbf{k}} \nabla \mathbf{f_i}(\mathbf{x_j}) \tag{11}$$

$$\mathbf{v_i^{(k+1)}} = \beta_\mathbf{2}\mathbf{v_i^{(k)}} + (\mathbf{1}-\beta_\mathbf{2})(\mathbf{g_i^{(k)}})^\mathbf{2} \tag{12}$$

$$\hat{\mathbf{v}}_\mathbf{i}{}^{\mathbf{(k+1)}} = \frac{\mathbf{v_i^{(k+1)}}}{\mathbf{1}-\beta_\mathbf{2}^{\mathbf{k+1}}} \tag{13}$$

$$\mathbf{x_i^{(k+1)}} = \mathbf{x_i^{(k)}} - \frac{\alpha}{\sigma + \sqrt{\hat{\mathbf{v}}_\mathbf{i}^{\mathbf{(k+1)}}}}\mathbf{g_i^{(k)}} \tag{14}$$

F. Adam

Adam is a combination of RMSProp and Momentum, representing the most general form among all optimizers. It can be quickly recognized that by appropriately selecting constants in Adam's formula, one can transform it into the equations of other optimizers.

$$\mathbf{g_i^{(k)}} = \frac{\mathbf{1}}{\mathbf{t}}\sum_{\mathbf{j=1}}^{\mathbf{k}} \nabla \mathbf{f_i}(\mathbf{x_j}) \tag{15}$$

$$\mathbf{m^{(k+1)}} = \beta_\mathbf{1}\mathbf{m^{(k)}} + (\mathbf{1}-\beta_\mathbf{1})\mathbf{g_i^{(k)}} \tag{16}$$

$$\mathbf{v_i^{(k+1)}} = \beta_\mathbf{2}\mathbf{v_i^{(k)}} + (\mathbf{1}-\beta_\mathbf{2})(\mathbf{g_i^{(k)}})^\mathbf{2} \tag{17}$$

$$\hat{\mathbf{m}}^{\mathbf{(k+1)}} = \frac{\mathbf{m^{(k+1)}}}{\mathbf{1}-\beta_\mathbf{1}^{\mathbf{k+1}}} \tag{18}$$

$$\hat{\mathbf{v}}_\mathbf{i}{}^{\mathbf{(k+1)}} = \frac{\mathbf{v_i^{(k+1)}}}{\mathbf{1}-\beta_\mathbf{2}^{\mathbf{k+1}}} \tag{19}$$

$$\mathbf{x_i^{(k+1)}} = \mathbf{x_i^{(k)}} - \frac{\alpha}{\sigma + \mathrm{RMS}(\mathbf{g_i})}\mathbf{m^{(k+1)}} \tag{20}$$

III. Virtual Experiments

As mentioned earlier, it is impossible to have a perfect understanding of the objective function of a deep learning model. Consequently, comprehending the behavior of optimizers through the deep learning model itself can be challenging. In this context, we will set a perfectly known polynomial function as the objective function to compare and interpret the behavior of optimizers. To generalize these results to deep learning, additional considerations are necessary. However, this experiment provides insights into the operating principles of optimizers. Through an improved understanding of optimizers, one can gain the insight needed to select the appropriate optimizer in various deep learning scenarios.

The virtual experiment is conducted under the following assumptions and settings:

1) The objective function is approximated by a polynomial function.
2) The entire dataset is considered without taking into account mini-batches.
3) No consideration for stochastic elements.
4) The learning rates for SGD and SGDM are set to 0.001, while the initial learning rates for Adagrad, RMSProp, and Adam are set to 0.01.
5) The number of iterations is set to 512.
6) The starting point is uniformly set to $x = 10$.

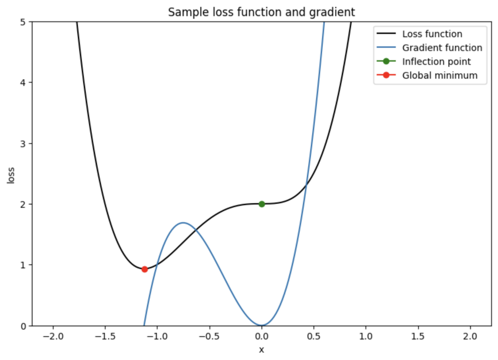

Fig. 1. $2x^4 - 3x^3 + 2$

The first experiment is designed to test how well each optimizer overcomes saddle points and converges to the minimum in the case of non-convex functions.

To better understand the following experimental results, it is helpful to compare them in pairs. Let's start by examining Adagrad and RMSProp. Adagrad struggles to converge to the minimum even after many iterations, while RMSProp demonstrates rapid convergence. One mathematical difference between Adagrad and RMSProp lies in the rate at which they decrease the learning rate. Adagrad sharply reduces the learning rate based on the cumulative squared sum of gradients, whereas RMSProp smoothly decreases it based on the exponentially weighted moving average. Consequently, Adagrad exhibits poor convergence due to rapid learning rate reduction, while RMSProp rapidly converges to the minimum, leveraging a relatively large initial learning rate.

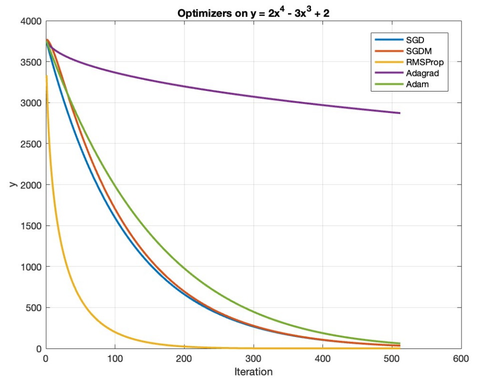

Fig. 2. $2x^4 - 3x^3 + 2$

Adam combines the adaptive learning rate of RMSProp with the influence of the momentum term, resulting in a more stable convergence pattern than RMSProp alone. However, it's crucial to note that rapid convergence in the case of polynomial functions doesn't necessarily imply superiority, as it may indicate vulnerability to local minima. On the other hand, SGD and SGDM also show stable convergence patterns, but this assessment should consider the assumption of a polynomial function with a single minimum.

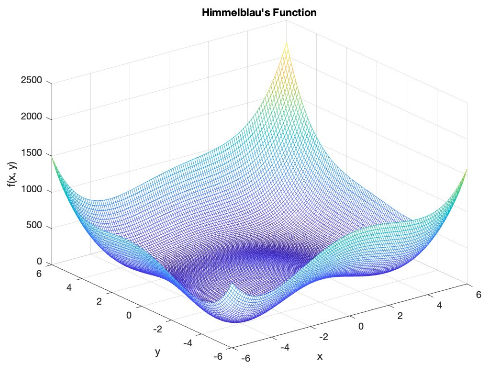

Fig. 3. Himmelblau's Function

The second experiment is designed to compare the convergence patterns of optimizers in a three-dimensional space using the non-convex objective function Himmelblau, which has four minima.

In the second experiment, Adagrad, RMSProp, and Adam exhibit similar patterns to the first one. On the other hand, SGD and SGDM show a rapid convergence towards the closest local minimum, indicating vulnerability to getting trapped in local minima when both global and local minima are present.

IV. Empirical Evaluations

In this section, we experimented with various optimizers on well-known CNN models (LeNet-5, AlexNet, VGGNet) using the most commonly used benchmark datasets (MNIST, Fashion MNIST, CIFAR-10).

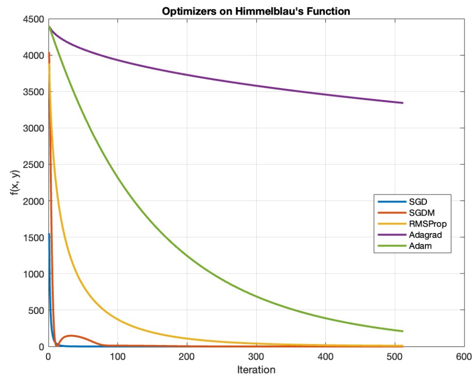

Fig. 4. Himmelblau's Function

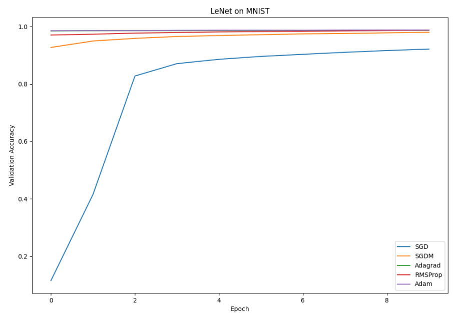

Fig. 5. LeNet5 on MNIST

In this experiment, it can be observed that SGD converges to different points even after multiple iterations. This suggests that SGD might be getting trapped in local minima and failing to converge to the global minimum.

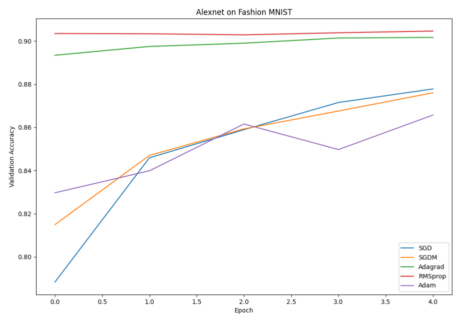

Fig. 6. AlexNet on Fashion MNIST

In the second experiment, only Adagrad and RMSProp exhibit favorable convergence patterns, while other optimizers seem constrained to local minima. This highlights the challenge of selecting the most suitable optimizer for real, complex deep learning models, as it is difficult to predict which optimizer will deliver optimal performance. Therefore, choosing the right optimizer based on a deep understanding of their behavior remains a crucial task.

V. Summary and Future Direction

References

[1] Oh Ilseok, Machine Learning.
[2] Mykel J. Kochenderfer, Algorithms for Optimization.
[3] Yanli Liu, An Improved Analysis of Stochastic Gradient Descent with Momentum(2020).
[4] John Duchi, Adaptive Subgradient Methods for Online Learning and Stochastic Optimization(2011).
[5] Alexandre Defossez, A Simple Convergence Proof of Adam and Adagrad(2022).
[6] Rachel Ward, AdaGrad stepsizes: Sharp convergence over nonconvex landscapes(2020)
[7] Fangyu Zou, A Sufficient Condition for Convergences of Adam and RMSProp.
[8] Diederik P. Kingma, Adam: A Method for Stochastic Optimization(2015).
[9] Pierre Foret, Sharpness-Aware Minimization for Efficiently Improving Generalization(2021).