Performance Analysis of the AdaGrad Family of Algorithms

Bagepalli Abhishek and Sanjay Singh

Abstract Optimisation algorithms in Machine Learning play a vital role in proper convergence during function evaluation to get better results, thereby forming one of the most important parts of Machine Learning. The AdaGrad family of algorithms, particularly, is noteworthy for several reasons. Their performance on certain types of data distributions is yet to be studied for robust adaptation. In this paper, we looked into the performance of these algorithms when pitted against different types of data distributions, the effect of cosine annealing on different hyper-parameters, and how one can improve upon them. We found that cosine annealing, improves the overall error rate average of the AdaGrad family by a factor of 2.5. These results would be vastly helpful in noise modeling and removal techniques, primarily in image and signal processing.

1 Introduction

Modern architectures in deep learning have revolutionized several application areas, from computer vision to speech recognition. Optimization methods have played an enormous role in the success of such models and have taken a significant place in artificial intelligence.

The gradient descent method(and its variants), which is by far the most popular of them, has several problems in the form of severe dependence on learning rates, causing issues in some instances, hence calling for better methods. It lays the foundation of an algorithm that adaptively scales the gradient value following the data

Bagepalli Abhishek
Department of Information and Communication Technology, Manipal Institute of Technology, MAHE, Manipal-576104 INDIA, e-mail: `bagepalliabhishek@gmail.com`

Sanjay Singh
Department of Information and Communication Technology, Manipal Institute of Technology, MAHE, Manipal-576104 INDIA e-mail: `sanjay.singh@manipal.edu`

or in correspondence with time. It begs the question, 'why not do both?'. In this paper, we ponder this question and perform experiments to determine if the simultaneous adaption of the learning rate, along with the annealing of hyper-parameters, improves the performance of such algorithms. Existing algorithms that adaptively scale learning rates include the AdaGrad family of algorithms, which are noteworthy for their performance [**?**]. Coming to the chronological update of hyperparameters, Cosine Annealing[7] exists, reducing the learning rate's value over time.

The data distributions we considered are by the Central Limit Theorem [6], which focuses on commonly occurring data distributions in nature. The algorithms we considered are AdaGrad [8], RMSProp(root mean square propagation), AdaDelta, and AMSGrad [10].

Our approach included the creation of specific data distributions, monitoring the performance of said algorithms, and critical comparison among them. We also looked into the workings of each algorithm in order to better their performance and came up with interesting and helpful conclusions. This paper's main contributions are:

- This paper proves that simultaneous updation of hyperparameters in different optimization algorithms for both time and the data itself results in improving the performance of all considered algorithms, namely AdaGrad, RMSProp, AdaDelta, and AMSGrad.

This paper is organized as follows. The section 2 describes the related work. We outline the dataset used and the details of the experiment under section 3. Section 4. The results are presented in section 7 and are discussed in section 7. Section 8 is where we wrap up our paper and discuss future work.

2 Related Work

Issues fueled the need for better optimization algorithms in the performance of the gradient descent algorithm (and other related algorithms). The unstable convergence of gradient descent is excellently summarized in [1]. Several solutions were suggested in the paper, but no concrete details were provided. A much more comprehensive study on the derivation and performance of better optimization algorithms was done by [3], where the AdaGrad family of algorithms is introduced. The features of the AdaGrad algorithms are further summarized in [2], where the central focus is on AdaGrad's performance in the presence of saddle points. Finally, suggestions were made by [9], where the emphasis was placed on endowing algorithms with long-term memory of gradients. To test the previously stated hypothesis, our paper uses the contributions made in optimization methods, data distributions, and learning rate schedules to effectively determine the simultaneous usage of hyperparameter updates in coherence with both time and data distribution.

3 Data Creation and Collection

This section describes the dataset creation process and the rationale for choosing those data distributions. The Central Limit Theorem [6] states that when a large sample size has a finite variance, the samples will be normally distributed. Hence, it seemed logical to consider datasets involving the Gaussian distribution, which is the most commonly occurring one. We considered data in two dimensions, which called for multivariate gaussian distributed data. The distribution of Gaussian Data in two dimensions involves information about the mean values (μ) of the data in each dimension (which, in our case, is a vector of dimension 2×1) and a variance-covariance matrix (Σ) (dimension 2×2) which has variances (σ^2) along its main-diagonal and covariances ($\sigma(x,y)$) along its counter-diagonal. It gives rise to three cases of data distributions, each with varying values in the variance-covariance matrix as follows:

i) Σ is arbitrary
ii) $\sigma^2 = 2\sigma(x,y)$
iii) $\Sigma = \sigma^2 I(\sigma(x,y) = 0)$ (where I is an identity matrix of dimension 2x2)

Out of these three cases, we have considered the first two cases. In addition to the above data, we considered two cases of mixed distributions [4]. The reason for such a consideration is that most real-life random variables are not derived from a single distribution but a mixture of distributions. True to its name, a mixed distribution contains a mixture of two or more distributions. Before delving into the cases considered, let us look into another type of data distribution-the Exponential Distribution [5]. We have a mixed distribution as, "Exponential + Gaussian (μ_1, Σ_1) + Gaussian (μ_2, Σ_2)," where $\mu_1 = [0,\ 0]^T, \mu_2 = [3,\ 3]^T,\ \Sigma_1 = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}$), and $\Sigma_2 = \begin{bmatrix} 5 & 5 \\ 5 & 5 \end{bmatrix}$.

The creation of each dataset considered in our study was meticulously designed, keeping in mind the challenges posed by each distribution. All datasets were created using the '`numpy.random`' library in Python. To ensure uniformity across cases involving different parameter updates, we ensured that all distributions shared a common seed node. A total of five hundred samples were considered for each case of the data distribution, and each of those samples was considered a starting point for each algorithm.

For each of the five hundred data points in each data distribution, four algorithms(as described below) were run for five hundred iterations each, providing ample metrics to measure the error rates of each algorithm.

Hence, for each point in data distribution, we collected the error data at each of the five hundred iterations for all four algorithms. A circular loss function was considered for our study to ensure a definite minimum.

4 Methodology

This section describes the methodology used in this paper.

4.1 AdaGrad Family of Algorithms

The AdaGrad family has several members, of which we decided to monitor the performance of four prominent members:

i) **AdaGrad** or the adaptive gradients algorithm is a wonderful solution to the rigidity of learning rates in Gradient Descent algorithms. It adaptively scales the learning rate for each dimension. Let us look into the working of AdaGrad. We consider the sum of squared partial derivatives and use it as a scaling factor for the learning rate. The scaling is done by dividing the current learning rate by the square root of the sum of squared partial derivatives until that iteration. A small value is added to the denominator to avoid division by zero. With the new step size in hand, the parameter can be updated, and the next iteration can be performed, and thus, the algorithm continues. AdaGrad has one hyperparameter, α, which denotes the initial step size or learning rate. The set of equations for AdaGrad is as follows:

$$\alpha_{t+1} = \frac{\alpha_t}{\sqrt{\varepsilon + S_{t+1}}} \tag{1}$$

where S_t denotes the sum of squared partial derivatives at time t. The parameter of AdaGrad is updated as per the following equation

$$\theta_{t+1} = \theta_t - \alpha_{t+1} \nabla f(\theta_t) \tag{2}$$

ii) **RMSProp**: A problem with AdaGrad is that the search can become too slow if the denominator becomes too large, resulting in minimal learning rates for each dimension or parameter. It could cause the algorithm to stop before the intended end is reached. RMSProp or Root Mean Square Propagation follows a central idea of storing the changing average of the squared gradients for each parameter and then dividing the gradient by the square root of the mean square. To prevent issues like the ones posed by AdaGrad, RMSProp used an exponentially decaying average to not give as much importance to gradient information from the extreme past, enabling it to converge rapidly the moment it located a convex bowl.
The process of updating, while similar to that of AdaGrad, replaces the square root of the sum of squared gradients with a root mean square(RMS) measure of the sum of squared gradients up to that iteration. Once the new step size is calculated, the parameter is updated similarly as followed in AdaGrad. We note that RMSProp has two hyperparameters, η denoting initial step size or learning rate, and ρ denoting a measure of momentum. The required relations

for RMSProp are as follows:

$$S_{t+1} = \rho S_t + (1-\rho)(\nabla f(\theta_t))^2 \tag{3}$$

and the updated step size is given by

$$\eta_{t+1} = \frac{\eta_t}{\varepsilon + RMS(S_{t+1})}. \tag{4}$$

The final parameter is updated as per the following equation

$$\theta_{t+1} = \theta_t - \eta_{t+1}\nabla f(\theta_t) \tag{5}$$

iii) **AdaDelta** is a more robust Adagrad extension that adapts learning rates based on a moving window of gradient updates rather than accumulating all previous gradients. AdaDelta continues to learn in this manner even after numerous updates have been completed. It is designed to boost the optimization process by typing to reduce the number of iterations. It involves the calculation of a step size for each parameter of the objective function for every iteration. The calculation of the learning rate is similar to that of AdaGrad, and the sum of squared partial derivatives is similar to that of RMSProp. However, there is a minor change in the form of a decaying moving average of the squared change to the parameter, δ.
The parameter δ is further updated in successive iterations using the hyperparameter, ρ, like in RMSProp. The updated delta value calculates the learning rate in the next iteration. Following this, the change to the parameter is calculated, and finally, the parameter is updated using the latest value of the change. AdaDelta has one hyperparameter, ρ-a measure of momentum.
AdaDelta does not require manual setting and tuning of an initial learning rate like the other algorithms. The updation in the learning rate for AdaDelta is given by

$$\gamma_{t+1} = \frac{\varepsilon + \sqrt{\delta_t}}{\varepsilon + \sqrt{S_t}} \tag{6}$$

where S_t denotes the decaying moving average of the squared partial derivative. The intermediate change to the parameter in the current iteration is given by

$$\Delta_{t+1} = \gamma_{t+1}\nabla f(\theta_t). \tag{7}$$

The updated decaying moving average is given by

$$\delta_{t+1} = \delta_t + (1-\rho)\Delta_{t+1}^2. \tag{8}$$

Finally, the relation for parameter updation is given by

$$\theta_{t+1} = \theta_t - \Delta_{t+1}. \tag{9}$$

iv) **AMSGrad**: It is a stochastic optimization method designed to address a convergence problem with Gradient Descent and Adam-based optimizers [9]. AMSGrad updates the parameters using the maximum of previously squared gradients rather than the exponential average. It maintains a first and second-moment vector and a maximum value of the second-moment vector for each parameter undergoing updation as part of the search process. Initially, it initialized to zero, and each value undergoes updation as they progress through the respective iteration. The first-moment vector employs a hyperparameter, β_1, and is updated using the objective function gradient and β_1.
The second-moment vector undergoes a similar updation but uses the square of the gradient and a new hyperparameter, β_2. The maximum value of the second-moment vector is computed, and the parameter is updated using the first-moment vector, the maximum of the second-moment vector, the initial step size, or the learning rate. AMSGrad has three hyperparameters, ζ-initial step size or learning rate, β_1-decay factor for the first-moment vector, and β_2-decay factor for the second moment vector. The first-moment vector is given by

$$m(t) = \beta_1(t)m(t-1) + (1-\beta_1(t))\nabla f(\theta(t-1)). \tag{10}$$

The second-moment vector is given by

$$v(t) = \beta_2(t)v(t-1) + (1-\beta_2(t))(\nabla f(\theta(t-1)))^2. \tag{11}$$

The maximum of the second-moment parameter vector is given by

$$\hat{v}(t) = max\{\hat{v}(t-1), v(t)\} \tag{12}$$

and the final updated parameter is given by

$$\theta_{t+1} = \theta_t - \frac{\zeta_t m(t)}{\sqrt{\hat{v}(t)}}. \tag{13}$$

5 Learning Schedule

We have used cosine annealing [7] to study the behavior of various optimization algorithms discussed in subsection 4. Cosine Annealing is a type of learning rate schedule that has the effect of starting with a high learning rate and rapidly decreasing to a low value before rapidly increasing again. It is done in coherence with the fact that as an algorithm approaches a particular minimum, it needs to take smaller steps as the scope narrows rapidly. The cosine function, a decreasing function, offers a unique opportunity to tune a parameter accordingly. The resetting of the learning rate acts as a simulated restart of the learning process, and the use of good weights as the restart's starting point is referred to as a "warm restart," as opposed to a "cold restart," which may use a new set of small random numbers as a starting point.

Cosine Annealing requires specific values as input for the update process. Such an update enabled a highly successful implementation of previously mentioned algorithms, which needed to perform satisfactorily on datasets created with certain distributions in mind. The learning schedule as per the cosine annealing [7] is given by

$$\eta = \eta_{\min} + \frac{1}{2}\left(\eta_{\max} - \eta_{\min}\right)\left(1 + \cos\left(\frac{T_{\text{cur}}}{T}\pi\right)\right) \tag{14}$$

where $\eta_{\min}$ and $\eta_{\max}$ are ranges for learning rate, and T_{cur} accounts for how many iterations have been performed since the last restart, and T is the total number of iterations.

6 Evaluation Metrics

As mentioned in the data section, we collected error data in Euclidean distance from the origin, which is the minimum of our circular loss function. For each of the five hundred points in every distribution, Euclidean error data were collected for five hundred iterations for each of AdaGrad, RMSProp, AdaDelta, and AMSGrad. The error data for each algorithm was stored in a NumPy array considering ease of access and management. Equipped with the above data, we computed the average error of each algorithm for every single one of the five hundred data points.

Next, we equipped each of the four algorithms with cosine annealing for each hyperparameter (as stated above) and again computed the error norms for all data points. We derived meaningful conclusions from the data and plots based on the observed results and careful comparative analysis. All plots were created using 'matplotlib' in Python.

7 Results and Discussion

This section presents the findings from the experiment. We evaluate the performances of AdaGrad, RMSProp, AdaDelta, and AMSGrad with and without the incorporation of cosine annealing for all involved hyperparameters. The main idea behind introducing cosine annealing is to improve algorithm performance by reducing the number of iterations before convergence to a minimum.

Now we look at the respective performances of the four algorithms on the following data distributions

1. Multivariate Gaussian Case - 1 is included in the figures 1-2
2. Multivariate Gaussian Case - 2 is included in the figures 3-4
3. Mixed Case - 2 is included in the figures 5-6

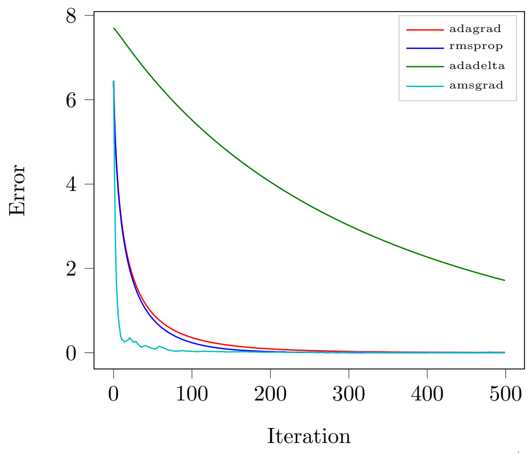

Fig. 1 Plot of Average Sum of Squared Error vs. Iteration for Case-1 of Multivariate Gaussian Data

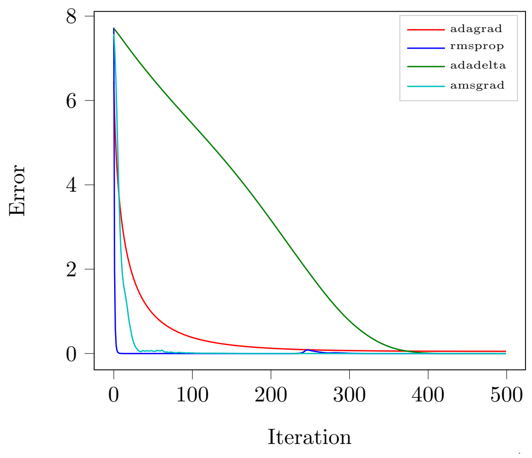

Fig. 2 Plot of Average Sum of Squared Error vs Iteration for Case-1 of Multivariate Gaussian Data with Cosine Annealing

Owing to the limited availability of GPU resources, we limited the data size to five hundred samples per data distribution and five hundred iterations per data point. Observing the algorithm's performance over much larger datasets that would strain the GPU would give a better idea of their feasibility.

The limited number of mixed distributions considered is also a potential improvement where multiple other distributions can also be factored in to observe and decide on the usage of the algorithms. Domain-specific knowledge would aid in creating mixed distribution data, which would be immensely helpful in areas involving noise modeling for image and signal processing tasks.

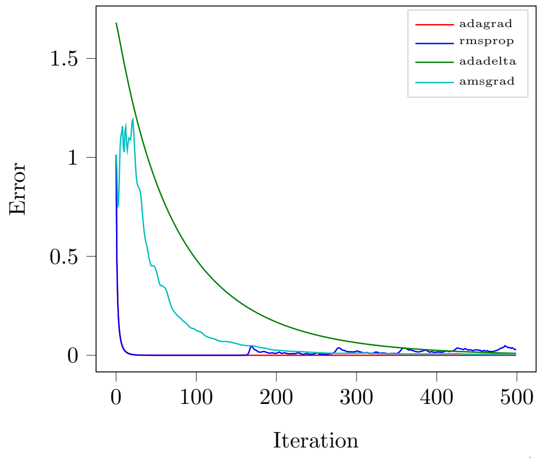

Fig. 3 Plot of Average Sum of Squared Error vs. Iteration for Case-2 of Multivariate Gaussian Data

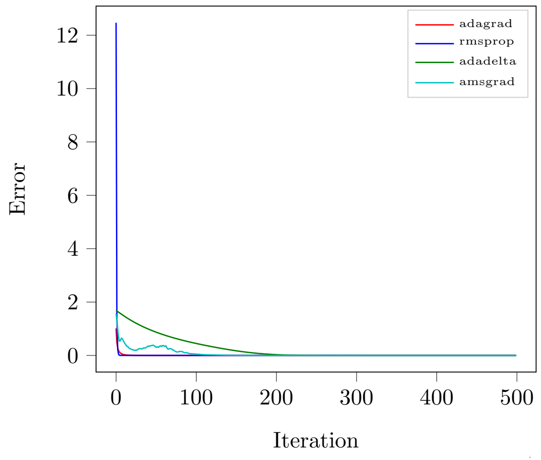

Fig. 4 Plot of Average Sum of Squared Error vs Iteration for Case-2 of Multivariate Gaussian Data with Cosine Annealing

The study of recurrent data distribution types in domain-specific areas is another area with scope for improvement. It would make conclusions derived from those results tailormade for the respective domain.

As seen in case-1 of the multivariate gaussian data distribution, before cosine annealing is performed, AMSGrad outperforms its counterparts, rapidly converging to a shallow error rate. While the subsequent best algorithms, i.e., AdaGrad and RMSProp, take about a hundred iterations to converge, AMSGrad converges almost instantly. The performance of AdaDelta, however, is inferior.

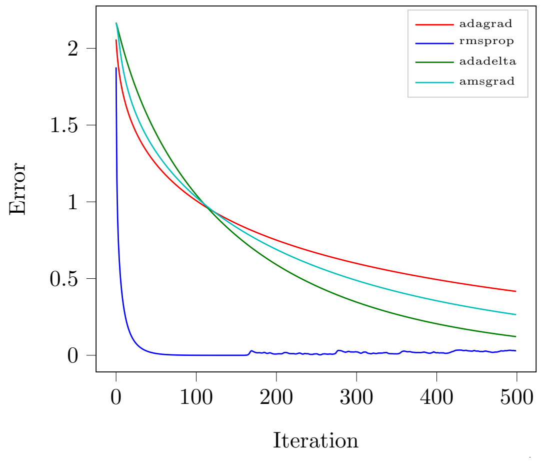

Fig. 5 Plot of Average Sum of Squared Error vs. Iteration for Case-2 of Mixed Data

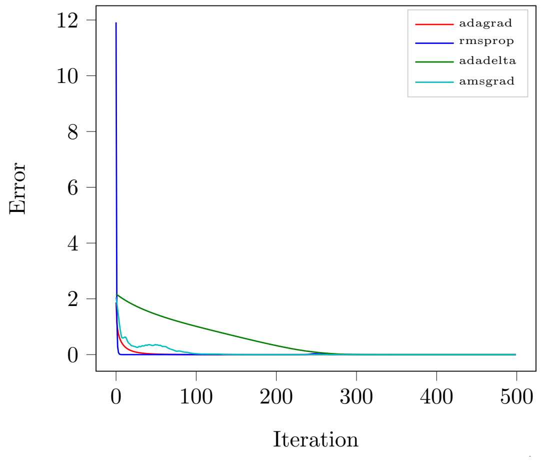

Fig. 6 Plot of Average Sum of Squared Error vs Iteration for Case-1 of Mixed Data with Cosine Annealing

The same case is rerun, but now with cosine annealing for all hyperparameters, giving rise to some incredible observations. RMSProp outperforms all other algorithms, converging almost instantly. AMSGrad holds its ground, with a slightly better performance than before. AdaGrad's performance is almost unchanged, while a good improvement is seen in the case of AdaDelta, which now converges accurately, albeit taking about 400 iterations to do so. In case 2 of the multivariate gaussian data distribution, RMSProp and AdaGrad outperform their counterparts before cosine annealing. However, a slight issue is observed in the case of RMSProp, as there is a very slight increase in its error rate, even after initial convergence. AMSGrad takes a wayward path before finally converging, with the error rate initially increasing before falling towards the end towards a decent convergence value. AdaDelta, while monotonically decreasing, takes about 400 iterations to converge properly to the minimum.

The same case is rerun but with cosine annealing for all hyperparameters; the observations are noteworthy for the following reasons. There is a marked improvement in the performance of all algorithms, with RMSProp and AdaGrad still converging faster than the rest. The initial error rate for AdaGrad, however, is much smaller when compared to RMSProp. AMSGrad, while having a near-perfect convergence, initially dabbles uncertainly, while AdaDelta converges monotonically before reaching the minimum around iteration 200.

The inferences that can be drawn from the above results are that RMSProp and AdaGrad respond excellently to cosine annealing. However, RMSProp needs to be adjusted to stop further movement after the minimum is found. Due to its not settling in local minima, however, RMSProp's biggest strength becomes its biggest weakness, causing the error rate to fluctuate perennially. Therefore, AdaGrad is the best choice for all three cases of multivariate gaussian data when cosine annealing is involved. When cosine annealing is not involved, RMSProp is still the best bet for case 2, while AMSGrad is a better choice for case 1.

It proves our hypothesis of simultaneous updation of hyperparameters according to data and time. We look at two cases with a mixture of data distributions to further verify the hypothesis.

In case of the mixed distributions, we consider a mixture of exponential and two gaussian data mixtures, as mentioned in the data section. We now look at the performance of different algorithms before cosine annealing is incorporated. Only RMSProp reaches the minimum, while all other algorithms fail to do so. However, the convergence rate is prolonged for RMSProp, with the best minimum reached around iteration 23.

On incorporating cosine annealing, however, A marked improvement is observed in all algorithms, with RMSProp being the fastest. However, there is a deviation from the best minimum to a slightly increased error rate. AdaGrad, AMSGrad, and AdaDelta converge to the same minimum, although AdaDelta is considerably slower, reaching the minimum around iteration 250.

Once again, based on the marked improvement of the algorithms' performance, the hypothesis that simultaneous updation of hyperparameters concerning data and time stands correct.

8 Conclusion

The AdaGrad family of algorithms is an integral part of the optimization process, thus forming a vital part in Machine Learning and Deep Learning. However, the performance of each algorithm varies from one data distribution to the other. In this paper, we have investigated the performance of the AdaGrad family of algorithms on different data distributions, which are representative of data distributions most commonly seen in real life. We have observed that with conventional learning schedules, AMSGrad converges almost instantly, while others do not give a satisfactory performance in the first case of multivariate Gaussian data. The second case saw RMSProp and AdaGrad outperforming their counterparts. On incorporating cosine annealing, a marked improvement was observed in the performance of all algorithms in all cases of data distribution that were considered, thereby confirming our initial hypothesis that the simultaneous usage of hyperparameter updates in coherence with both time and the data distribution would lead to an improvement in algorithm performance.

References

1. Ahn, K., Zhang, J., Sra, S.: Understanding the unstable convergence of gradient descent. In: K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, S. Sabato (eds.) International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA, *Proceedings of Machine Learning Research*, vol. 162, pp. 247–257. PMLR (2022). URL https://proceedings.mlr.press/v162/ahn22a.html
2. Antonakopoulos, K., Mertikopoulos, P., Piliouras, G., Wang, X.: AdaGrad avoids saddle points. In: K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, S. Sabato (eds.) Proceedings of the 39th International Conference on Machine Learning, *Proceedings of Machine Learning Research*, vol. 162, pp. 731–771. PMLR (2022). URL https://proceedings.mlr.press/v162/antonakopoulos22a.html
3. Duchi, J.C., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**, 2121–2159 (2011). DOI 10.5555/1953048.2021068. URL https://dl.acm.org/doi/10.5555/1953048.2021068
4. Ghojogh, B., Ghojogh, A., Crowley, M., Karray, F.: Fitting a mixture distribution to data: Tutorial (2019). DOI 10.48550/ARXIV.1901.06708. URL https://arxiv.org/abs/1901.06708
5. Gupta, A., Zeng, W.B., Wu, Y.: Exponential Distribution, pp. 23–43 (2010). DOI 10.1007/978-0-8176-4987-6_2
6. Kwak, S., Kim, J.: Central limit theorem: The cornerstone of modern statistics. Korean Journal of Anesthesiology **70**, 144 (2017). DOI 10.4097/kjae.2017.70.2.144
7. Loshchilov, I., Hutter, F.: Sgdr: Stochastic gradient descent with warm restarts (2016). DOI 10.48550/ARXIV.1608.03983. URL https://arxiv.org/abs/1608.03983
8. Lydia, A., Francis, S.: Adagrad - an optimizer for stochastic gradient descent **Volume 6**, 566–568 (2019)
9. Reddi, S.J., Kale, S., Kumar, S.: On the convergence of adam and beyond. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net (2018). URL https://openreview.net/forum?id=ryQu7f-RZ
10. Shaziya, H.: A study of the optimization algorithms in deep learning (2020). DOI 10.1109/ICISC44355.2019.9036442