EXPONENTIALLY DECAYING FLOWS FOR OPTIMIZATION IN DEEP LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

The field of deep learning has been craving for an optimization method that shows outstanding property for both optimization and generalization. We propose a method for mathematical optimization based on flows along geodesics, that is, the shortest paths between two points, with respect to the Riemannian metric induced by a non-linear function. In our method, the flows refer to Exponentially Decaying Flows (EDF), as they can be designed to converge on the local solutions exponentially. In this paper, we conduct experiments to show its high performance on optimization benchmarks (i.e., convergence properties), as well as its potential for producing good machine learning benchmarks (i.e., generalization properties).

1 INTRODUCTION

Due to recent progress in the field of machine learning, it becomes more and more important to develop and sophisticate methods of solving hard optimization problems. At the same time, in this field, such methods are additionally required to elicit decent generalization performance from statistical models. An efficient method of mathematical optimization, however, does not always produce sufficient generalization properties, since these are involved with two distinct mathematical problems; The former is to find one of the solutions which minimize a given (possibly non-convex) objective function, and the latter is to adjust parameters so that a statistical estimator achieves its best result. To address such a hard issue, we introduce a new mathematical perspective on optimization, and develop a method for machine learning based on this perspective. We then empirically show its rapid convergence rate and high compatibility with deep learning techniques, as well as good statistical properties.

In this field, many optimization methods have been proposed and modified so that they fit specific problems or models. One of the current standard methods is the gradient descent method. The method tends to converge slowly in general optimization problems. However, with various specific techniques, such as mini-batch training and batch normalization (Ioffe & Szegedy (2015)), it has been found to be efficient for state-of-the-art purposes in the field of deep learning. Another class of methods that are now becoming popular and standard is adaptive methods, such as AdaGrad (Duchi et al. (2011)) and Adam (Kingma & Ba (2015)). Compared to the gradient descent method, these methods have been shown to improve convergence rates with almost the same computational cost as the gradient descent method, but are reported to result in poor statistical outcomes in some cases of machine learning (Wilson et al. (2017)).

Other class of methods that have been thoroughly studied in the theory of mathematical optimization is second-order methods, such as the Newton method and the Gauss-Newton method. These methods possess great convergence properties, and in particular, have a potential to overcome plateau's problems (Dauphin et al. (2014)). Furthermore, when it comes to applications in stochastic settings, the method based on the Gauss-Newton Matrix (or Fisher information Matrix) is shown to asymptotically attain the best statistical result, which is called Fisher efficiency (see Amari (1998)). Despite these attractive characteristics, the methods have not yet been spotlighted in the field of machine learning due to several severe drawbacks; They suffer from high computational cost in general and their useful properties are no longer guaranteed in practical settings (see Section 12 in Martens (2014)). One of the continuously developing second-order methods in this field, K-FAC (Ba et al. (2017), Grosse & Martens (2016)), successfully produced high convergence rate empirically with relatively low computational cost. However, it still requires much effort to become compatible with

some deep learning techniques. In addition, it is unclear whether the method has advantages in generalization performance.

In our approach, by introducing a Riemannian metric induced by non-linear functions, we constitute dynamical systems which describe motions along the shortest route from arbitrary initial points to the zeros of non-linear functions on the corresponding Riemannian manifold, that is, geodesic with respect to the Riemannian metric. One of the remarkable characteristics of our approach is that it enables us to flexibly design flows of such dynamical systems to control convergence rates. The results for the flows are then applicable to mathematical optimization problems, in particular, with deep neural network (DNN) models. In this paper, after providing mathematical ground of our methods, we experimentally demonstrate their performance in various aspects, from convergence rates to statistical properties.

2 GEODESIC FLOWS FOR NON-LINEAR EQUATIONS

We start by establishing some essential properties of dynamics which are effective for the analysis of non-linear equations. Let $F\colon \mathbb{R}^N \to \mathbb{R}^N$ be a smooth function and J be the Jacobian with variable $w \in \mathbb{R}^N$, that is, $J = \partial F/\partial w$. In this section, we deal with the well-posed case that there exists a connected closed subset $\Omega \subset \mathbb{R}^N$, where J is regular and the equation has a unique solution ξ. Therefore, the positive matrix $G = J^T J$ induces a Riemannian metric g on Ω and (Ω, g) becomes a Riemannian manifold under some appropriate conditions. Let us then consider time evolution of variable w on this manifold. Our main purpose in this section is to study the characteristics of dynamical systems in which $w(t)$ moves on geodesics between any point in Ω and ξ with respect to the metric g.

Let L be a Lagrangian given by

$$L\left(w, v\right) = \frac{1}{2}\, v^T G(w) v \tag{1}$$

with $v = dw/dt$ (also written as $\dot{w}$). The Euler-Lagrange equation for L is then expressed as

$$\frac{dp}{dt} - \nabla_w L = 0 \tag{2}$$

with momentum vector $p = Gv$. If the boundary condition at two points in Ω, $w(t_0) = w_0, w(t_1) = w_1$, is imposed on (2), a geodesic between w_0 and w_1 is obtained as the solution. In contrast, if we give an appropriate initial condition, w describes the motion along the geodesic from w_0 to w_1. In fact, the following statement holds;

Theorem 2.1. *Let w_0 be an arbitrary point in Ω and $(w(t), p(t))$ be a solution of equation (2) with the following initial condition;*

$$w(0) = w_0, \quad p(0) = -J(w_0)^T F(w_0). \tag{3}$$

Then w satisfies

$$F\big(w(t)\big) = (1-t)F\big(w_0\big), \tag{4}$$

for $t \in [0, 1]$. In particular, $w(t)$ passes through the point which is a solution of non-linear equation $F(w) = 0$ at $t = 1$, that is, $\xi = w(1)$.

We briefly describe the outline of the proof for the statement above. Throughout this paper, we regard functions of w as those of t in the trivial way; for instance, $F(t) = F(w(t))$. Note that p can be expressed as $p = J^T dF/dt$. Then using the Beltrami identity for (2) with the initial condition above leads to the equation

$$\frac{d}{dt}F = -F_0, \tag{5}$$

where $F_0 = F(0)$. Thus, a closed form expression is obtained as

$$F(t) = (1-t)F_0, \tag{6}$$

which gives $F(1) = 0$ as asserted in Theorem 2.1.

Now we take a different expression that the coefficient $(1 - t)$ in (6) is replaced by a different monotonically decreasing function, that is,

$$F(t) = \rho(t)F_0, \tag{7}$$

where ρ denotes a monotonically decreasing smooth function from (t_0, t_1) onto $(0, 1)$. Then, we give the following differential equation whose solution is of the closed form (7);

$$\frac{d}{dt}F = \chi \cdot F, \quad \chi(t) = \frac{d}{dt}\ln(\rho(t)). \tag{8}$$

A motion described by this differential equation differs from the one that is described by (2), but these two motions are along the same geodesic.

Theorem 2.2. *Let $w_0 \in \Omega$ be an arbitrary point. The differential equation*

$$J\frac{dw}{dt} = \chi \cdot F, \quad t \in (t_0, t_1) \tag{9}$$

with an initial condition $w_0 = w(t_0)$ has a unique solution that satisfies $F(w(t_1)) = 0$. Furthermore, the orbit under flow f defined by $f(w_0, t) = w(t)$ coincides with that of the geodesic equation (2).

Note that since equation (9) is equivalent to equation (8), the orbit is invariant under coordinate transformations.

With respect to the choice of ρ, the end point t_1 can be set as ∞ under some appropriate conditions and Theorem 2.2 still holds in the sense that

$$\lim_{t\to\infty} F(w(t)) = 0. \tag{10}$$

In particular, if we set $\rho(t) = e^{-t}$, then $\chi(t) = -1$ and F can be represented as $F(t) = e^{-t}F_0$, so that the convergence rate of F is exponential.

Definition 2.3. Let w be a solution of the differential equation

$$J\frac{dw}{dt} = -F, \quad t \in (t_0, t_1) \tag{11}$$

with an initial condition $w_0 = w(t_0)$. The flow $f(w_0, t) = w(t)$ is called an *exponentially decaying flow* (EDF) of non-linear equation $F(w) = 0$.

For the end of this section, we present a connection between EDF and the classical Newton method. If we apply the Euler method to the differential equation (9) with step size $\triangle t$, the corresponding iteration step can be written as

$$w_{i+1} = w_i + \triangle t \cdot \chi(\tau_i) \cdot J(w_i)^{-1}F(w_i), \quad \tau_{i+1} = \tau_i + \triangle t, \tag{12}$$

which recovers the Newton method with step size $\eta_i = \triangle t \cdot \chi(\tau_i)$.

3 Application to mathematical optimizations

Consider smooth functions $\varphi\colon \mathbb{R}^N \to \mathbb{R}^M$ and $L\colon \mathbb{R}^M \to \mathbb{R}$. In this section, we develop a method based on EDF for a mathematical optimization problem of the form

$$\min_{w\in\Omega} L(\varphi(w)). \tag{13}$$

In this section, the area $\Omega \subset \mathbb{R}^N$ is supposed to be a compact subset where there exists no stationary point except for a minima.

Let $F\colon \mathbb{R}^N \to \mathbb{R}^M$ denote derivative $\partial L/\partial\varphi$. An example of such problems is least squares one in which L is given by

$$L(\varphi) = \frac{1}{2}\|\varphi\|^2. \tag{14}$$

In this case, $F = \varphi$. In particular, if $M = N$ and a minimal value of loss function $\widehat{L} = L \circ \varphi$ is zero, the optimization problem is equivalent to solving non-linear equation $F(w) = 0$.

For optimization problems, we set up a standard equation that the gradient of loss function is zero, that is, $\nabla\widehat{L}(w) = 0$. Note that the gradient can be expressed as $\nabla\widehat{L}(w) = J_\varphi^T F(w)$ with Jacobian $J_\varphi = \partial\varphi/\partial w$. Applying Theorem 2.2, we obtain the differential equation

$$H\frac{dw}{dt} = \chi \cdot J_\varphi^T F, \tag{15}$$

where H is the Hessian of loss function $\widehat{L}$ with respect to w. Since second order differentiation is typically computationally heavy, we seek other equations which possess almost the same property as (15) especially in terms of asymptotic behavior.

Let us decompose momentum $H\dot{w}$ as

$$H\frac{dw}{dt} = \frac{d}{dt}\left(J_\varphi^T F\right) = \left(\frac{d}{dt}J_\varphi\right)^T F + G\frac{dw}{dt}, \tag{16}$$

where J_F denotes the Jacobian of F, and G is a symmetric matrix defined by

$$G = J_\varphi^T J_F = J_\varphi^T H_L J_\varphi \tag{17}$$

with Hessian matrix H_L of L with respect to φ. We then consider the following equation instead of (15);

$$G\frac{dw}{dt} = \chi \cdot J_\varphi^T F. \tag{18}$$

This equation no longer describes the motion along the geodesic related to (15) in general. However, if $M = N$ and J_φ is invertible, then w moves on another geodesic with respect to a different metric $G_F = J_F^T J_F$. In addition, if $\rho(t) = e^{-t}$, F converges exponentially, which implies that $\nabla\widehat{L} = J_\varphi F$ also converges exponentially in (18) as well as in (15). In general cases, if a condition that

$$\frac{d}{dt}\left(\frac{1}{2}\|F\|^2\right) = \chi\langle J_F G^{-1} J_\varphi^T F, F\rangle \leq \chi\|F\|^2 \tag{19}$$

is satisfied, then F converges to 0. This shows that in the neighborhood of solution ξ of equation $\nabla\widehat{L} = 0$, the momentum $G\dot{w}$ sufficiently approximates $H\dot{w}$ by (16).

Definition 3.1. The flow given by

$$H\frac{dw}{dt} = -J_\varphi^T F \tag{20}$$

is referred to *EDF of type H* and written as EDF-H. Similarly, the flow given by

$$G\frac{dw}{dt} = -J_\varphi^T F \tag{21}$$

is referred to *EDF of type G* and written as EDF-G.

4 Modification schemes for EDF-based methods

Like second order methods, in EDF-based methods, matrix inversion has to be carried out, which requires expensive computational cost, particularly in large systems. Moreover, we often encounter rank deficient matrices in optimization problems. To deal with rank deficiency, in general, we need pseudo-inverse matrices which are more computationally heavy. Therefore, instead of the inverse of matrix $A = G, H$, for fixed $v \in \mathbb{R}^M$, we consider a projection which maps $r = \arg\min_{x\in\mathbb{R}^M} \|Ax - v\|$ to a vector $P_k(A, v)$ in the k-th order Krylov subspace $\mathcal{K}_k(A, v) = \mathrm{span}\{v, Av, A^2v, \ldots, A^{k-1}v\}$ such that $r = P_\infty(A, v)$. One of the basic methods to construct such a projection for indefinite symmetric systems is the minimum residual method (Paige & Saunders (1975)), which requires only the matrix multiplication. Therefore, for numerical computations, we use the following differential equation approximated by the k-th order Krylov subspace;

$$\frac{dw}{dt} = \chi \cdot P_k(A, J_\varphi F), \quad A = G, H. \tag{22}$$

k is a hyperparameter that interpolates between the gradient method and the method based on (15) or (18). In fact, in the case that $k = 1$, equation (22) has the form

$$\frac{dw}{dt} = c \cdot \nabla\widehat{L}, \quad c = \chi\frac{\langle u, \nabla\widehat{L}\rangle}{\langle u, u\rangle}, \quad u = A\,\nabla\widehat{L}, \tag{23}$$

which reduces to a kind of gradient methods, but the coefficient c conveys information about G or H unlike the standard gradient methods.

Next, similar to the Levenberg-Marquardt algorithm, we modify equation (18) by adding a damping factor to G in order that the method becomes more stable. So, we set

$$(G + \lambda I)\,\frac{dw}{dt} = \chi \cdot J_\varphi^T F, \tag{24}$$

where λ is a continuous positive function of t. Then, we take the same approximation as (22) with $A = G+\lambda I$. The damping factor λ plays several important roles in solving practical problems. First, it has solutions well-behaved near singularities. Even in the case that G is not invertible, equation (24) can be defined and solved unlike (18). If we choose λ such that it approaches to 0 as rapidly as the gradient in (18), the asymptotic behavior of (24) is almost the same as that of (18). Particularly, in the case that $\chi = -1$, we set $\lambda = a\|J_\varphi^T F\|^b$ with $a, b > 0$, so that the convergence rate of (24) stays exponential. Second, the damping factor makes the method compatible with stochastic approaches such as mini-batch training, in deep learning models. Since the orbit of (24) is affected by the gradient due to the damping factor λ, the method based on this equation could take advantage of stochastic approaches as other gradient methods do. (For implementation of the algorithm, see Appendix A.)

Finally, to accelerate the EDF-based methods, it is sometimes effective to change equation (18) into a second-order differential equation, particularly in the case in which the approximation method with k is applied. Specifically, we take the equation

$$G\frac{d^2w}{dt^2} + \kappa \cdot G\frac{dw}{dt} + J_\varphi^T F = 0, \tag{25}$$

where κ is a real-valued function of t. The convergence properties of the methods are controlled by the following differential equation;

$$\frac{d^2F}{dt^2} + \kappa \cdot \frac{dF}{dt} + F = 0. \tag{26}$$

There are two ways of determining κ. The first one is to set $\kappa = \alpha$ with constant α (W1), which leads to a similar scheme to the momentum gradient decent method. The other one is to set $\kappa(t) = \alpha t^{-1}$ (W2). In this setting, the equation can be discretized in a similar way as described in Su et al. (2014), which is analogous to the Nesterov's acceleration scheme.

5 Optimization problems in the field of deep learning

In this section, we present how optimization problems are set up in the field of deep learning. Let $x = \{x_j\}_{j=0}^{n-1}$ and $y = \{y_j\}_{j=0}^{n-1}$ denote training datasets of input and output, respectively, where n is the size of dataset. Let d_x and d_y be dimensions of input data and output data, respectively. We write φ_{nn} for neural networks with parameters $w \in \mathbb{R}^N$, and define φ by the direct sum of vectors $\{\varphi_j\}$ given by $\varphi_j(w) = \varphi_{\mathrm{nn}}(x_j, w)$, that is, $\varphi = \oplus\varphi_j$. Note that $M = n \times d_y$ in this case. Then finding a minima of a given loss function is proposed as a standard optimization problem to train networks.

For the image classification tasks, there are two typical cases of setting loss functions. In the first case, the loss is set as (14). As already mentioned, in this case, $F = \varphi$ and $H_L = I$. In the second case, the loss is given by cross entropy with softmax function, that is,

$$L(\varphi) = \frac{1}{n}\sum_{j=0}^{n-1} y_j \cdot \ln\left(\theta_j\right), \tag{27}$$

where θ denotes the softmax function and $\theta_j = \theta(\varphi_j)$. In this case, F is expressed by the direct sum such that $F = \oplus F_j$ with

$$F_j = \frac{1}{n}\left(s_j\theta_j - y_j\right), \quad j = 0, \ldots, n-1, \tag{28}$$

where s_j denotes a sum of all elements in vector y_j for each j. Note that if each y_j is given as a probability vector, then $s_j = 1$. Moreover, H_L is expressed as $H_L = \oplus H_j$ with

$$H_j = \frac{s_j}{n}\left(\mathrm{diag}(\theta_j) - \theta_j \otimes \theta_j\right), \tag{29}$$

where $\otimes$ denotes the outer product. In both cases, the loss functions take the minimum value 0 if and only if $F = 0$.

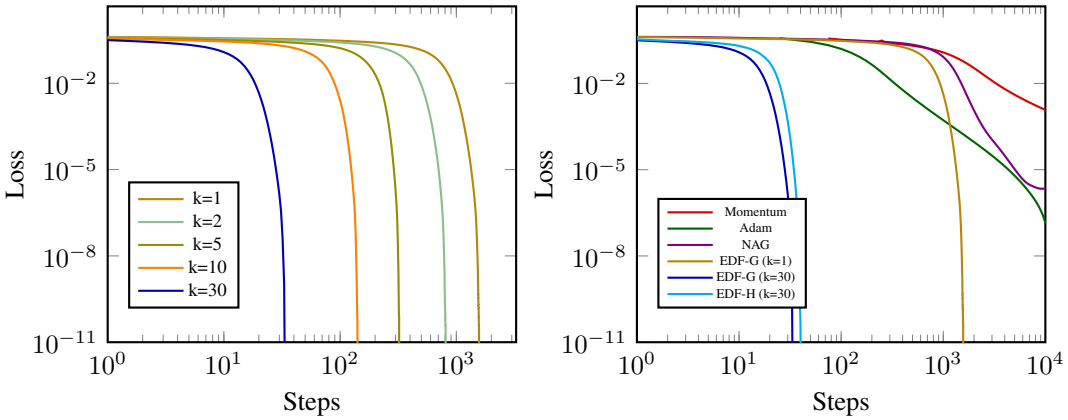

Figure 1: Convergence curves for various optimizers

6 EXPERIMENTS

In this study, we conducted following three groups of experiments; First, we examined optimization performance of the EDF-based methods on a data-fitting problem with a simple convolutional neural network model. Second, we tested both optimization and generalization performance in standard settings of classification tasks in which we employed residual network models (Resnet He et al. (2016)) and CIFAR-10/100 datasets. Third, we incorporated some techniques of data augmentation and regularization in the training into our experiment as we would like to measure effectiveness of our methods in more practical settings.

The primary purpose of the experiments in this paper is not to pursue the state-of-the-art performance, but to explore how the statistical results pertain to those of optimization when the loss functions are minimized. Therefore, we tuned hyperparameters of each method in accordance with the optimization benchmark.

It should be noted that the conjecture concerning non-convex optimization problems in the field of deep learning (Baldi & Hornik (1989), Choromanska et al. (2015)) is still an open problem (studied for linear cases in Baldi & Lu (2012), Kawaguchi (2016)). Hence, for experiments in this paper, we do not discuss whether each optimization method actually reaches to a global solution.

6.1 OPTIMIZATION PERFORMANCE ON A STANDARD DATA-FITTING PROBLEM

We evaluated convergence performance of EDF-based methods (type G and type H) on a data-fitting problem of CIFAR-10, that is, full-batch training in the context of deep learning. The model we employed in these experiments was the convolutional neural network that consisted of two convolution filters with rectified linear units and max pooling layers, followed by two fully connected layers. For EDF-based methods, the step size was fixed to 1.0, and no damping factors were used. In addition, the acceleration technique derived from W2 of (25) was adapted, since W2 achieved better performance than W1. A similar experiment with a different type of second-order methods was conducted in Sohl-Dickstein et al. (2014).

First, we examined change in convergence performance of EDF-G depending on hyperparameter k, the order of Krylov subspace. The results are illustrated in the left-hand side of Figure 1. The presented trend is consistent with the theoretical fact that k interpolates between the gradient method (for small k) and the method based on dynamics (18) (for large k). In other words, when k is small, the method is similar to a gradient method, which converges slow, but as k becomes larger, the method leads to better approximation of the inverse matrix, which gives a rapidly decaying flow.

Next, we compared EDF-G ($k = 1, 30$) with EDF-H and other standard optimizers in deep learning: gradient descent methods with Polyaks momentum scheme (Momentum) and Nesterov's acceleration scheme (NAG), and Adam. The step sizes for Momentum, NAG, and Adam were fixed to 0.01, 0.001, and 0.001, respectively.

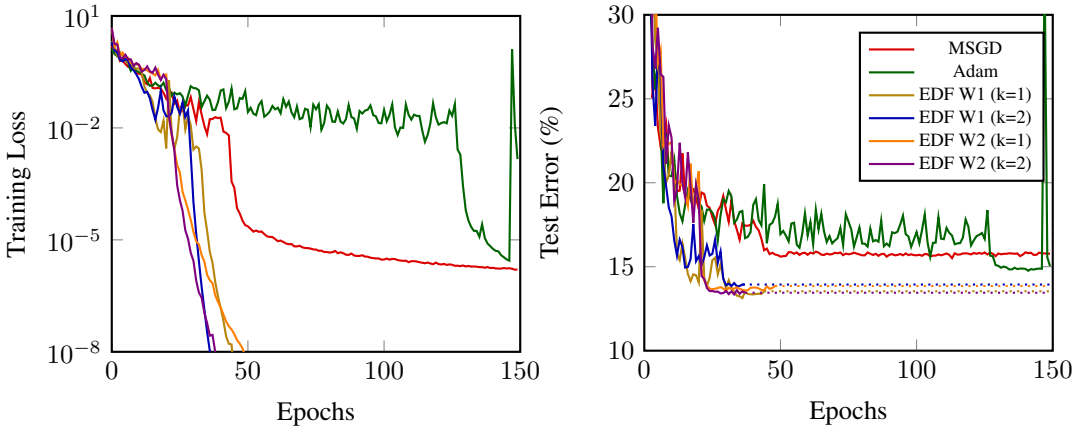

Figure 2: Training loss and test error for Resnet-56 on CIFAR-10

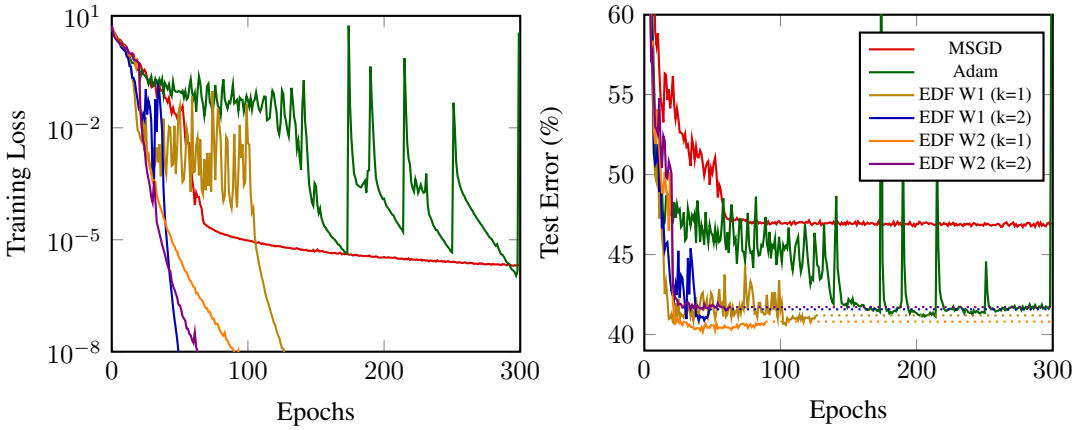

Figure 3: Training loss and test error for Resnet-110 on CIFAR-100

The right-hand side of Figure 1 shows that both EDF-based methods made the loss decrease more rapidly than other standard optimizers. Even when EDF-based methods were reduced to the gradient methods at $k = 1$, EDF-G outperformed standard optimizers.

The figure also presents the difference between EDF-G and EDF-H. While their convergence rates around extremes were almost the same, the overall convergence rates of EDF-G were better than that of EDF-H.

As has been found, second-order-methods on full-batch training converge to the solution within a few iterations. However, with such a training, the quality of generalization becomes worse compared to the time when stochastic approach with a mini-batch of small size is taken. Like other second-order-methods, EDF also suffers from a negative impact of full-batch training on generalization performance, though setting the hyperparameter k small makes EDF be compatible with stochastic approaches and take their advantages (see Appendix B).

6.2 Performances on classification tasks in deep learning

In the following experiments, we compared both optimization and generalization performance of EDF-G with other methods, momentum stochastic gradient decent method (MSGD) and Adam on classification tasks for CIFAR-10/100. The experiments were conducted, employing residual network models with batch normalization (Resnet-56 for CIFAR-10 and Resnet-110 for CIFAR-100), and working with a mini-batch of size 250.

For CIFAR-100, during pre-investigation of the dataset, we found that 9 pairs of data, each of which has the same image but different labels, contaminated the neural network and might have had a non-negligible influence on both optimization and generalization performance (See Appendix C). Therefore, in our experiments on CIFAR-100, we excluded them from the training dataset and used the rest of the data (size = 49982).

At the end of each epoch, we computed training loss in the following manner. First, the loss was defined as (27) where n is the total number of data. Second, to calculate the statistics for batch normalization in the loss, as described in Ioffe & Szegedy (2015), we adopted so-called "inference mode" in which moving averages of mean and variance over mini-batches are used for batch-normalization.

For EDF, we tested the case $k = 1$ and $k = 2$, with the damping factor set as $a = b = 1$. The momentum coefficient was fixed to $\alpha = 0.9$ for the W1 acceleration (see 25).

For each of the tasks, we ran tests on 10 different learning rates; for EDF between 0.1 and 2.0, for Momentum SGD between 0.1 and 10.0, for Adam between 0.0001 and 0.1. We then chose the one with the best convergence performance for each optimizer. For EDF with W2, to obtain stability in convergence, the learning rate was set to 0.2 times that of the initial values at the end of the 20-th epoch. Such a change in learning rate in the middle of optimization did not bring advantages to either optimization or generalization performances for other methods including EDF with W1 (see Appendix D).

The results are presented in Figures 2 and 3. As shown in the figures, with respect to the optimization performance, EDF reached an optimal solution with smaller error at higher convergence rate than Adam and Momentum SGD, even when $k = 1, 2$. Moreover, EDF overall attained better generalization performance than other methods.

6.3 Performances with techniques in a more practical setting

For classification tasks, generalization performance often improves by adopting several techniques, such as data augmentation and regularization. In this group of experiments, employing the data augmentation based on random horizontal flip and shift and the L^2 regularization, we conducted comparisons between EDF and other methods that are similar to those in the previous sections.

When adopting these techniques, rate decay scheme has as large impact on optimization performance as learning rate itself. Because effectiveness of each rate decay scheme much depends on optimization methods, it is almost impossible to find the best scheme that is applicable to all the methods. For this reason, for MSGD and Adam, as initial learning rates, we chose one of the standard rates for MSGD and Adam, 0.1 and 0.001, respectively, and at the ends of the 100-th, 140-th, 180-th epochs, reset the rates to 0.1 times the rates at the moment. For EDF, we ran tests on two different rates 0.5 and 0.75, the optimal rates found in the experiments of Section 6.2, and reset to 0.2 times those of the initial values, only at the end of the 100-th epoch, in order to demonstrate performance of the EDF more clearly. Among the results obtained with EDF, we chose the ones with the best optimization performance.

The result of the comparison is presented in Figures 4 and 5. As can be seen, we found a condition in which EDF achieved better performance than other methods on optimization while achieving sufficient levels of generalization.

7 Conclusion

Obtaining good statistical results from limited available data is a critical goal in machine learning. To reach this goal, while developing an effective model is an essential approach, eliciting the best performance from the fixed model through optimization is important as well. In our study, to examine the performance of our optimization methods, Exponentially Decaying Flows (EDF) based methods, we explored their generalization properties pertaining to results of optimization. Our experiments showed that EDF-based methods are more likely to achieve optimal solutions which generalize the test data well than other standard optimizers are. Therefore, EDF-based methods are considered to be optimization methods that have a high potential in their application to various tasks, and thus, are worthwhile to be sophisticated through future studies.

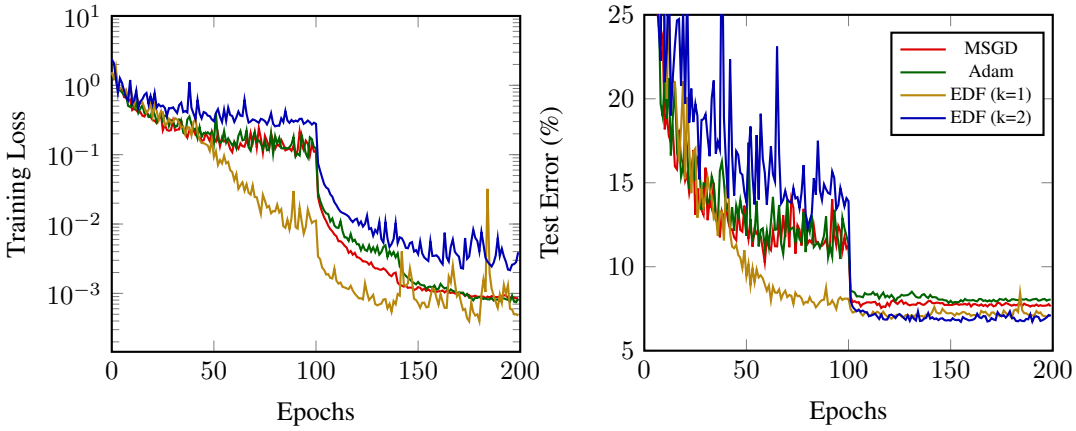

Figure 4: Training loss and test error for Resnet-56 on CIFAR-10 with data augmentation

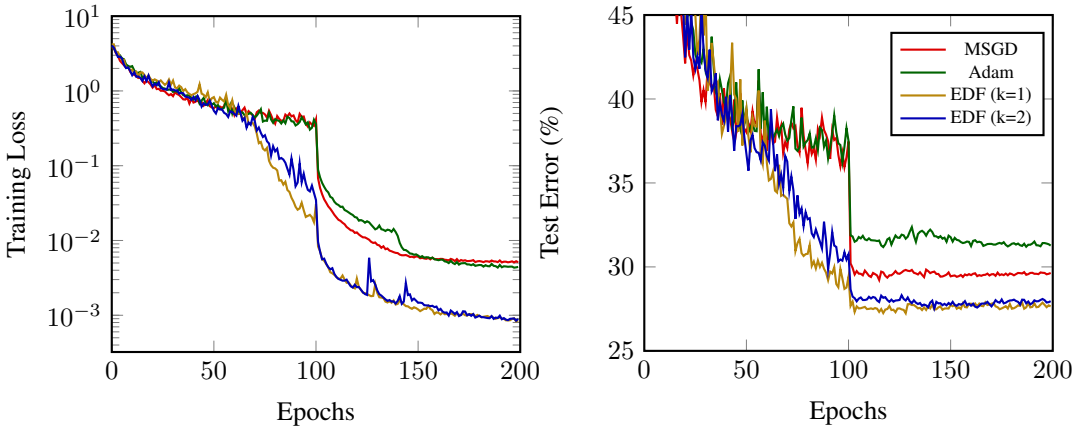

Figure 5: Training loss and test error for Resnet-110 on CIFAR-100 with data augmentation

REFERENCES

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

Jimmy Ba, Roger Grosse, and James Martens. Distributed second-order optimization using kronecker-factored approximations. In *International Conference on Learning Representations*, 2017.

Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989.

Pierre Baldi and Zhiqin Lu. Complex-valued autoencoders. *Neural Networks*, 33:136–147, 2012.

Anna Choromanska, Yann LeCun, and Gérard Ben Arous. Open problem: The landscape of the loss surfaces of multilayer networks. In *Conference on Learning Theory*, pp. 1756–1760, 2015.

Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pp. 2933–2941, 2014.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Roger Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pp. 573–582, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Kenji Kawaguchi. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems*, pp. 586–594, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

James Martens. New insights and perspectives on the natural gradient method. *arXiv:1412.1193*, 2014.

Christopher C Paige and Michael A Saunders. Solution of sparse indefinite systems of linear equations. *SIAM journal on numerical analysis*, 12(4):617–629, 1975.

Jascha Sohl-Dickstein, Ben Poole, and Surya Ganguli. Fast large-scale optimization by unifying stochastic gradient and quasi-newton methods. In *International Conference on Machine Learning*, pp. 604–612, 2014.

Weijie Su, Stephen Boyd, and Emmanuel Candes. A differential equation for modeling nesterovs accelerated gradient method: Theory and insights. In *Advances in Neural Information Processing Systems*, pp. 2510–2518, 2014.

Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in neural information processing systems*, pp. 4148–4158, 2017.

A ALGORITHMS OF EDF-BASED METHODS

In terms of computation of the EDF-based methods with GPU, the Jacobian-vector-product can be carried out at almost the same cost as the gradient of loss function. In fact, multiplying a vector by Jacobian and its transposed matrix (written as R-op and L-op, respectively) are implemented in combination with gradients of scholar functions. For the psuedo-code for update scheme of the EDF-G with L/R-op, refer to Algorithm 1, and Algorithm 2 in particular case that $k = 1$.

Algorithm 1: Update scheme for EDF-G with non-preconditioned MINRES

Input : w_i, k, a, b, η
Output: w_{i+1}
Compute $g = \nabla L(\varphi)$
Set $s = \|g\|,\ \lambda = as^b$
Initialize parameters for MINRES as
$\beta = s,\ \xi = \beta,\ \gamma_0 = \gamma_1 = 1,\ \sigma_0 = \sigma_1 = 0,\ v = v_0 = 0,\ v_1 = g,\ q_0 = q_1 = 0$
for $j = 1$ to k **do**
$v_1 = v_1/\beta$
Compute $u = J_F v_1$ using R-op
Compute $u = J_\varphi^T u$ using L-op
$u = u + \lambda g$
$\alpha = \langle v_1, u\rangle$
$u = u - \alpha v_1 - \beta v_0$
$v_0 = v_1,\ v_1 = u$
$\delta = \gamma_1\alpha - \gamma_0\sigma_1\beta,\ \rho_2 = \sigma_1\alpha + \gamma_0\gamma_1\beta,\ \rho_3 = \sigma_0\beta$
$\beta = \|v_1\|$
$\rho_1 = \sqrt{\delta^2 + \beta^2}$
$\gamma_0 = \gamma_1,\ \gamma_1 = \delta/\rho_1,\ \sigma_0 = \sigma_1,\ \sigma_1 = \beta/\rho_1$
$u = q_0,\ q_0 = q_1,\ q_1 = (1/\rho_1)(v_0 - \rho_2 q_0 - \rho_3 u)$
$v = v + \gamma_1\xi q_1$
$\xi = -\sigma_1\xi$
end for
$w_{i+1} = w_i - \eta v$

Algorithm 2: Update scheme for $k = 1$

Input : w_i, a, b, η
Output: w_{i+1}
Compute $g = \nabla L(\varphi)$
Set $s = \|g\|,\ \lambda = as^b$
Compute $u = J_F g$ using R-op
Compute $u = J_\varphi^T u$ using L-op
$u = u + \lambda g$
$c = \langle g, u\rangle / \langle u, u\rangle,\ v = cg$
$w_{i+1} = w_i - \eta v$

B COMPARISON BETWEEN FULL-BATCH AND STOCHASTIC TRAININGS

Figure 6 shows the results of experiments using EDF for simple examples that compare a full-batch training with stochastic trainings. In this example, the convolutional network similar to that used in Section 6.1 was employed on MNIST. The curve labeled as "EDF F" depicts the result of Full-batch training per step, and those labeled as "EDF S" illustrate the results of stochastic trainings per epoch with a mini-batch of size 500.

Let us divide the dataset $\{x, y\} = \{x_j, y_j\}_{j=0}^{n-1}$ into distinct subsets of size p such that

$$\{x, y\} = \bigcup_{i=1}^{l} \{x^{(i)}, y^{(i)}\}, \quad \{x^{(i)}, y^{(i)}\} = \{x_j, y_j\}_{j=p(i-1)}^{pi-1}, \quad pl = n. \tag{30}$$

Then, gradients and Jacobian-vector-multiplications are calculated as

$$\nabla L(\varphi) = \sum_i \nabla L\big(\varphi^{(i)}\big), \quad Gv = \sum_i J_{\varphi^{(i)}}^T J_{F^{(i)}} v, \tag{31}$$

where $\varphi^{(i)}$ and $F^{(i)}$ are the subcomponents of φ and F, respectively, corresponding to the decomposition above. Thus, for a fixed k, if we set p as the same size as a mini-batch, then the computational cost per step of a full-batch training becomes almost the same as that per epoch of a mini-batch training.

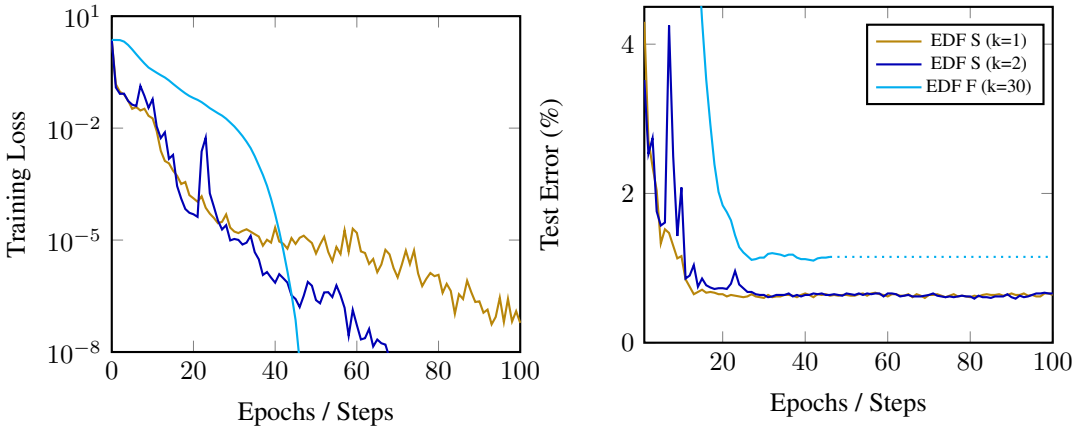

Figure 6: Comparison between full-batch and stochastic trainings on MNIST

C IRREGULAR DATA IN CIFAR-100

The dataset of CIFAR-100 includes 9 pairs of irregular data, each of which has the same image but different labels. These data are enumerated in Figure 7. For instance, the 8393-rd and the 36874-th images are the same, but they have different labels, "girl (class 35)" and "baby (class 2)."

Such pairs contaminate the training process. In fact, in our experiment, when the network model was optimized with the full dataset, one of the images in each 9 pairs above could not be classified correctly, which resulted in stagnated training accuracy at 99.982 %. Moreover, generalization also deteriorated when irregular data were contained. For the details of these results, see Figure 8.

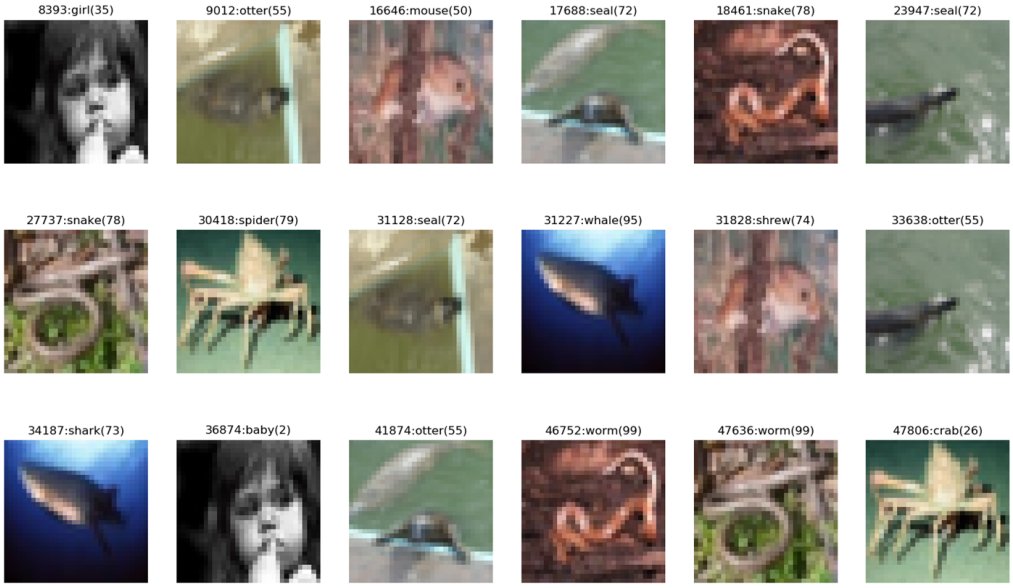

Figure 7: Duplicated images with different labels in CIFAR-100

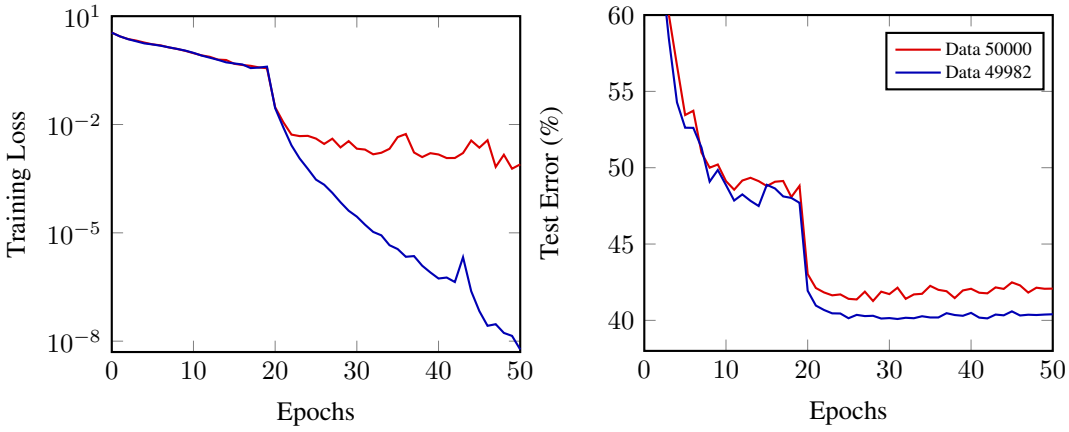

Figure 8: Comparison between EDF performance on the full dataset and that without irregular data

D PERFORMANCES WITH THE RATE DECAY SCHEME

Figure 9 and Figure 10 present the optimization and generalization performance of each optimizer when the rate decay scheme was adopted to the experiment with the same setting as in Figure 2 and Figure 3. The rate decay scheme was that the learning rate was set to 0.2 times that of the initial values at the end of the 20-th epoch.

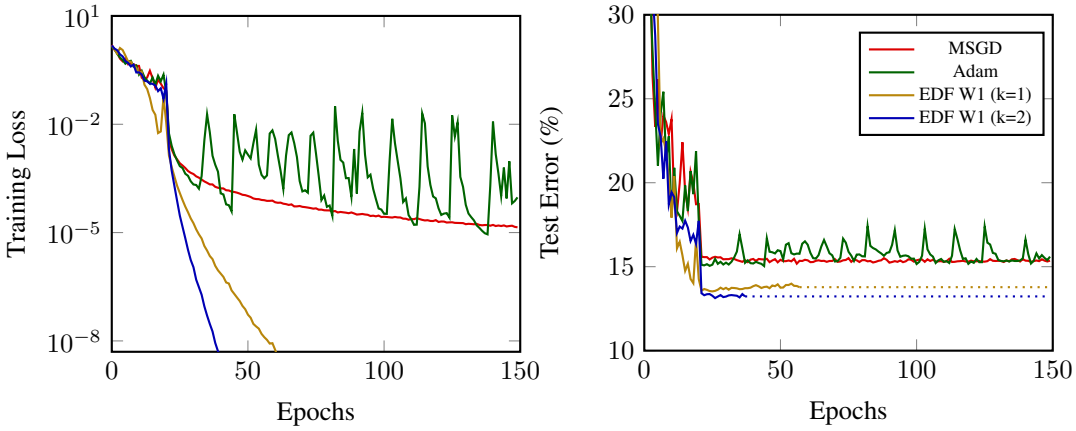

Figure 9: Training loss and test error for Resnet-56 on CIFAR-10 with the rate decay scheme

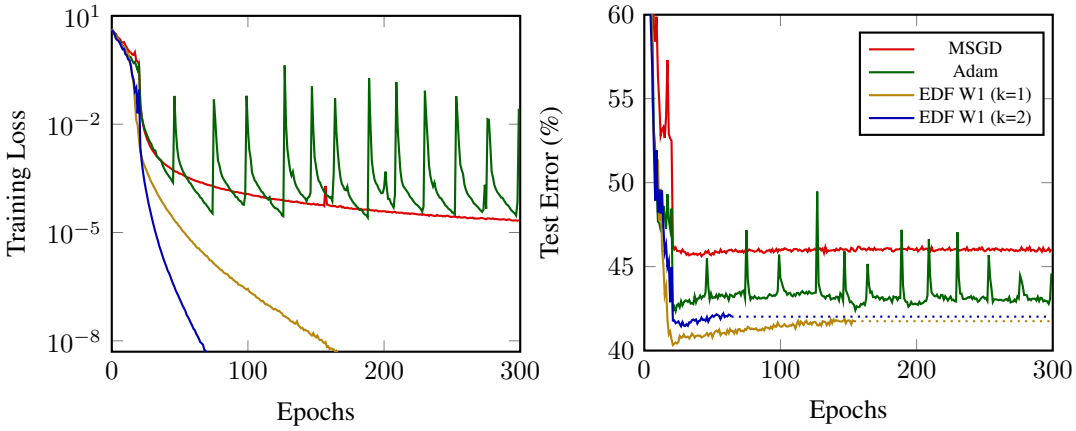

Figure 10: Training loss and test error for Resnet-110 on CIFAR-100 with the rate decay scheme