STOCHASTIC GEODESIC OPTIMIZATION FOR NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

We develop a novel and efficient algorithm for optimizing neural networks inspired by a recently proposed geodesic optimization algorithm. Our algorithm, which we call Stochastic Geodesic Optimization (SGeO), utilizes an adaptive coefficient on top of Polyak's Heavy Ball method effectively controlling the amount of weight put on the previous update to the parameters based on the change of direction in the optimization path. Experimental results on strongly convex functions with Lipschitz gradients and deep Autoencoder benchmarks show that SGeO reaches lower errors than established first-order methods and competes well with lower or similar errors to a recent second-order method called K-FAC (Kronecker-Factored Approximate Curvature). We also incorporate Nesterov style lookahead gradient into our algorithm (SGeO-N) and observe notable improvements.

1 INTRODUCTION

First order methods such as Stochastic Gradient Descent (SGD) with Momentum (Sutskever et al., 2013) and their variants are the methods of choice for optimizing neural networks. While there has been extensive work on developing second-order methods such as Hessian-Free optimization (Martens, 2010) and Natural Gradients (Amari, 1998; Martens & Grosse, 2015), they have not been successful in replacing them due to their large per-iteration costs, in particular, time and memory.

Although Nesterov's accelerated gradient and its modifications have been very effective in deep neural network optimization (Sutskever et al., 2013), some research have shown that Nesterov's method might perform suboptimal for strongly convex functions (Aujol et al., 2018) without looking at local geometry of the objective function. Further, in order to get the best of both worlds, search for optimization methods which combine the the efficiency of first-order methods and the effectiveness of second-order updates is still underway.

In this work, we introduce an adaptive coefficient for the momentum term in the Heavy Ball method as an effort to combine first-order and second-order methods. We call our algorithm Geodesic Optimization (GeO) and Stochastic Geodesic Optimization (SGeO) (for the stochastic case) since it is inspired by a geodesic optimization algorithm proposed recently (Fok et al., 2017). The adaptive coefficient effectively weights the momentum term based on the change in direction on the loss surface in the optimization process. The change in direction can contribute as implicit local curvature information without resorting to the expensive second-order information such as the Hessian or the Fisher Information Matrix.

Our experiments show the effectiveness of the adaptive coefficient on both strongly-convex functions with Lipschitz gradients and general non-convex problems, in our case, deep Autoencoders. GeO can speed up the convergence process significantly in convex problems and SGeO can deal with ill-conditioned curvature such as local minima effectively as shown in our deep autoencoder benchmark experiments. SGeO has similar time-efficiency as first-order methods (e.g. Heavy Ball, Nesterov) while reaching lower reconstruction error. Compared to second-order methods (e.g., K-FAC), SGeO has better or similar reconstruction errors while consuming less memory.

The structure of the paper is as follows: In section 2, we give a brief background on the original geodesic and contour optimization introduced in Fok et al. (2017), neural network optimization methods and the conjugate gradient method. In section 3, we introduce our adaptive coefficient specifically designed for strongly-convex problems and then modify it for general non-convex cases.

In section 4, we discuss some of the related work in the literature. Section 5 illustrates the algorithm's performance on convex and non-convex benchmarks. More details and insights regarding the algorithm and the experiments can be found in the Appendix.

2 BACKGROUND

2.1 GEODESIC AND CONTOUR OPTIMIZATION

The goal is to solve the optimization problem $\min_{\theta \in \mathbb{R}} f(\theta)$ where $f : \mathbb{R}^D \to \mathbb{R}$ is a differentiable function. Fok et al. (2017) approach the problem by following the geodesics on the loss surface guided by the gradient. In order to solve the geodesic equation iteratively, the authors approximate it using a quadratic. In the neighbourhood of θ_t, the solution of the geodesic equation can be approximated as:

$$\theta_{t+1} = \theta_t + v_t \delta_t - c_t \delta_t^2 \tag{1}$$

where θ_t is the current point, v_t is the unit tangent vector of the geodesic at θ_t, δ_t is the step size and:

$$c_t = \frac{1}{2}\big[\nabla f(\theta_t) - 2(v_t \cdot \nabla f(\theta_t))v_t\big] \tag{2}$$

and θ_{t+1} is the next point. The tangent vector is estimated by the normalized difference vector between the current point and the previous point $v_t = \theta_t - \theta_{t-1}$ and the first tangent vector is set to the gradient $v_1 = \nabla f(\theta_1)$. For a detailed explanation we refer the reader to Fok et al. (2017).

2.2 NEURAL NETWORKS

We consider a neural network with a differentiable loss function $f : \mathbb{R}^D \to \mathbb{R}$ with the set of parameters θ. The objective is to minimize the loss function f with a set of iterative updates to the parameters θ. Gradient descent methods propose the following update:

$$\theta_{t+1} = \theta_t - \epsilon \nabla f(\theta_t) \tag{3}$$

where ϵ is the learning rate. However, this update can be very slow and determining the learning rate ϵ can be hard. A large learning rate can cause oscillations and overshooting. A small learning rate can slow down the convergence drastically.

Heavy Ball method In order to speed up the convergence of gradient descent, one can add a momentum term (Polyak, 1964).

$$d_{t+1} = \mu d_t - \epsilon \nabla f(\theta_t); \quad \theta_{t+1} = \theta_t + d_{t+1} \tag{4}$$

where d is the velocity and μ is the momentum parameter.

Nestrov's method Nesterov's accelerated gradient (Nesterov, 1983) according to Sutskever et al. (2013) can be rewritten as a momentum method:

$$d_{t+1} = \mu d_t - \epsilon \nabla f(\theta_t + \mu d_t); \quad \theta_{t+1} = \theta_t + d_{t+1} \tag{5}$$

Nesterov's momentum is different from the heavy ball method only in where we take the gradient. For the class of convex functions with Lipschitz gradients, Nesterov's momentum is shown to be optimal with a convergence rate of $O(1/t^2)$ (Nesterov, 1983). Note that in both these methods d_t is the previous update $\theta_t - \theta_{t-1}$.

2.3 CONJUGATE GRADIENTS

The conjugate gradient algorithm (Hestenes & Stiefel, 1952) attempts to solve the quadratic problem $\min_{\theta \in \mathbb{R}^n} \frac{1}{2}\theta^T Q \theta - b^T \theta$ where Q is a $n \times n$ positive definite matrix and the update is:

$$\theta_{t+1} = \theta_t + \epsilon_t d'_t; \quad d'_{t+1} = -g_{t+1} + \gamma_t d'_t \tag{6}$$

where ϵ and γ are step sizes and d' is the search direction. In the non-quadratic case, $\min_{\theta \in \mathbb{R}^n} f(\theta)$, the function is locally approximated using a quadratic where $g_t = \nabla f(\theta_t)$ and $Q = \nabla^2 f(\theta_t)$ and ϵ and γ are:

$$\epsilon_t = -\frac{g_t^T d'_t}{d_t'^T [\nabla^2 f(\theta_t)] d'_t}; \quad \gamma_t = \frac{g_{t+1}^T [\nabla^2 f(\theta_t)] d'_t}{d_t'^T [\nabla^2 f(\theta_t)] d'_t} \tag{7}$$

Clearly, one can see the conjugate gradient method as a momentum method where ϵ_t is the learning rate and $\epsilon_t \gamma_{t-1}$ is the momentum parameter:

$$\theta_{t+1} = \theta_t - \epsilon_t g_t + \epsilon_t \gamma_{t-1} d'_{t-1} \tag{8}$$

Note that $d'_t = (\theta_{t+1} - \theta_t)/\epsilon_t$ (We added the prime notation to avoid confusion with $d_t = \theta_{t+1} - \theta_t$ throughout the paper). To avoid calculating the Hessian $\nabla^2 f$ which can be very expensive in terms of computation and memory, ϵ is usually determined using a line search, i.e. by approximately calculating $\epsilon_t = \arg\min_\epsilon f(\theta_t + \epsilon d_t)$ and several approximations to γ have been proposed. For example, Fletcher & Reeves (1964) have proposed the following:

$$\gamma_t^{FR} = \frac{\|g_t\|^2}{\|g_{t-1}\|^2} \tag{9}$$

Note that γ^{FR} (with an exact line search) is equivalent to the original conjugate gradient algorithm in the quadratic case.

3 Stochastic Geodesic Optimization

The adaptive coefficient that appears before the unit tangent vector in equation 2 has an intuitive geometric interpretation:

$$\bar{g}_t \cdot \bar{d}_t = \cos(\pi - \phi_t) \quad \text{where} \quad \bar{g}_t = \frac{g_t}{\|g_t\|}; \quad \bar{d}_t = \frac{d_t}{\|d_t\|} \tag{10}$$

where ϕ_t is the angle between the previous update $d_t = \theta_t - \theta_{t-1}$ and the negative of the current gradient $-g_t$. Since $0 \leq \phi \leq \pi$, thus $-1 \leq \cos(\pi - \phi_t) \leq 1$. The adaptive coefficient embeds a notion of direction change on the path of the algorithm which can be interpreted as implicit second-order information. The change in direction at the current point tells us how much the current gradient's direction is different from the previous gradients which is similar to what second-order information (e.g. the Hessian) provide. For more details on the adaptive coefficient we refer the reader to Appendix C.

3.1 Strongly Convex Functions with Lipschitz gradients

We propose to apply this implicit second-order information to the Heavy Ball method of Polyak (1964) as an adaptive coefficient for the momentum term such that, in strongly-convex functions with Lipschitz gradients, we reinforce the effect of the previous update when the directions align, i.e. in the extreme case: $\phi = 0$ and decrease when they don't, i.e. the other extreme case: $\phi = \pi$. Thus, we write the coefficient as

$$\gamma_t^C = 1 - \bar{g}_t \cdot \bar{d}_t \tag{11}$$

with C indicating "convex". It's obvious that $0 \leq \gamma_t^C \leq 2$. Note that we will use the bar notation (e.g. $\bar{d}$) throughout the paper indicating normalization by magnitude. A $\ddot{\mu}$-strongly convex function f with L-Lipschitz gradients has the following properties:

$$f(\theta') \geq f(\theta) + f'(\theta) \cdot (\theta' - \theta) + \frac{\ddot{\mu}}{2}\|\theta' - \theta\|_2^2; \quad \|f'(\theta) - f'(\theta')\|_2 \leq L(\theta - \theta')^2 \tag{12}$$

Applying the proposed coefficient to the Heavy Ball method, we have the following algorithm which we call GeO (Geodesic Optimization):

Algorithm 1: GeO (strongly convex and Lipschitz)

Initialize θ_1
Set $d_1 = \nabla f(\theta_1)$
for $t = 1$ ***to*** T **do**
  Calculate the gradient $g_t = \nabla f(\theta_t)$
  Calculate adaptive coefficient $\gamma_t^C = 1 - \bar{g}_t \cdot \bar{d}_t$
  Calculate update $d_{t+1} = \alpha \gamma_t^C d_t - \epsilon g_t$
  Update parameters $\theta_{t+1} = \theta_t + d_{t+1}$

where T is total number of iterations and α is a tunable parameter set based on the function being optimized and ϵ is the learning rate.

Incorporating Nesterov's momentum We can easily incorporate Nesterov's lookahead gradient into GeO by modifying line 4 to $g_t = \nabla f(\theta_t + \mu d_t)$ which we call GeO-N. In GeO-N the gradient is taken at a further point $\theta_t + \mu d_t$ where μ is a tunable parameter usually set to a value close to 1.

3.2 NON-CONVEX FUNCTIONS

However, the algorithm proposed in the previous section would be problematic for non-convex functions such as the loss function when optimizing neural networks. Even if the gradient information was not partial (due to minibatching), the current direction of the gradient cannot be trusted because of non-convexity and poor curvature (such as local minima, saddle points, etc). To overcome this issue, we propose to alter the adaptive coefficient to

$$\gamma_t^{NC} = 1 + \bar{g}_t \cdot \bar{d}_t \tag{13}$$

with NC indicating "non-convex". By applying this small change we are reinforcing the previous direction when the directions do not agree thus avoiding sudden and unexpected changes of direction (i.e. gradient). In other words, we choose to trust the previous history of the path already taken more, thus acting more conservatively. To increase efficiency, we use minibatches, calling the following algorithm SGeO (Stochastic Geodesic Optimization):

Algorithm 2: SGEO (NON-CONVEX)

1 Initialize θ_1
2 Set $d_1 = \nabla f(\theta_1)$
3 **for** $t = 1$ ***to*** T **do**
4 Draw minibatch from training set
5 Calculate the gradient $g_t = \nabla f(\theta_t)$
6 Calculate adaptive coefficient $\gamma_t^{NC} = 1 + \bar{g}_t \cdot \bar{d}_t$
7 Calculate update $d_{t+1} = \epsilon_t \gamma_t^{NC} \bar{d}_t - \epsilon_t \bar{g}_t$
8 Update parameters $\theta_{t+1} = \theta_t + d_{t+1}$

Further we found that using the unit vectors for the gradient $\bar{g}$ and the previous update $\bar{d}$, when calculating the next update in the non-convex case makes the algorithm more stable. In other words, the algorithm behaves more robustly when we ignore the magnitude of the gradient and the momentum term and only pay attention to their directions. Thus, the magnitudes of the updates are solely determined by the corresponding step sizes, which are in our case, the learning rate and the adaptive geodesic coefficient. Same as the strongly convex case, we can integrate Nesterov's lookahead gradient into SGeO by replacing line 5 with $g_t = \nabla f(\theta_t + \mu d_t)$ which we call SGeO-N.

4 RELATED WORK

There has been extensive work on large-scale optimization techniques for neural networks in recent years. A good overview can be found in Bottou et al. (2018). Here, we discuss some of the work more related to ours in three parts.

4.1 GRADIENT DESCENT VARIANTS

Adagrad (Duchi et al., 2011) is an optimization technique that extends gradient descent and adapts the learning rate according to the parameters. Adadelta (Zeiler, 2012) and RMSprop (Tieleman & Hinton, 2012) improve upon Adagrad by reducing its aggressive deduction of the learning rate. Adam (Kingma & Ba, 2014) improves upon the previous methods by keeping an additional average of the past gradients which is similar to what momentum does. Adaptive Restart (Odonoghue & Candes, 2015) proposes to reset the momentum whenever rippling behaviour is observed in accelerated gradient schemes. AggMo (Lucas et al., 2018) keeps several velocity vectors with distinct parameters in order to damp oscillations. AMSGrad (Reddi et al., 2018) on the other hand, keeps a longer memory of the past gradients to overcome the suboptimality of the previous algorithms on simple convex problems. We note that these techniques are orthogonal to our approach and can be adapted to our geodesic update to further improve performance.

4.2 ACCELERATED METHODS

Several recent works have been focusing on acceleration for gradient descent methods. Meng & Chen (2011) propose an adaptive method to accelerate Nesterov's algorithm in order to close a small gap in its convergence rate for strongly convex functions with Lipschitz gradients adding a possibility of more than one gradient call per iteration. In Su et al. (2014), the authors propose a differential equation for modeling Nesterov inspired by the continuous version of gradient descent, a.k.a. gradient flow. Wibisono et al. (2016) take this further and suggest that all accelerated methods have a continuous time equivalent defined by a Lagrangian functional, which they call the Bregman Lagrangian. They also show that acceleration in continuous time corresponds to traveling on the same curve in spacetime at different speeds. It would be of great interest to study the differential equation of geodesics in the same way. In a recent work, Defazio (2018) proposes a differential geometric interpretation of Nesterov's method for strongly-convex functions with links to continuous time differential equations mentioned earlier and their Euler discretization.

4.3 SECOND-ORDER METHODS

Second-order methods are desirable because of their fine convergence properties due to dealing with bad-conditioned curvature by using local second-order information. Hessian-Free optimization (Martens, 2010) is based on the truncated-Newton approach where the conjugate gradient algorithm is used to optimize the quadratic approximation of the objective function. The natural gradient method (Amari, 1998) reformulates the gradient descent in the space of the prediction functions instead of the parameters. This space is then studied using concepts in differential geometry. K-FAC (Martens & Grosse, 2015) approximates the Fisher information matrix which is based on the natural gradient method. Our method is different since we are not using explicit second-order information but rather implicitly deriving curvature information using the change in direction.

5 EXPERIMENTS

We evaluated SGeO on strongly convex functions with Lipschitz gradients and benchmark deep autoencoder problems and compared with the Heavy-Ball and Nesterov's algorithms and K-FAC.

5.1 STRONGLY CONVEX AND LIPSCHITZ FUNCTIONS

We borrow these three minimization problems from Meng & Chen (2011) where they try to accelerate Nesterov's method by using adaptive step sizes. The problems are Anisotropic Bowl, Ridge Regression and Smooth-BPDN. The learning rate ϵ for all methods is set to $\frac{1}{L}$ except for Nesterov which is set to $\frac{4}{3L+\ddot{\mu}}$ and the momentum parameter μ for Heavy Ball, Nesterov and GeO-N is set to the following:

$$\mu = \frac{1 - \sqrt{\ddot{\mu}\epsilon}}{1 + \sqrt{\ddot{\mu}\epsilon}}$$

where L is the Lipschitz parameter and $\ddot{\mu}$ is the strong-convexity parameter. The adaptive parameter γ_t for Fletcher-Reeves is set to γ_t^{FR} and for GeO and GeO-N is $\gamma_t^C = 1 - \bar{g}_t \cdot \bar{d}_t$. The function-specific parameter α is set to 1, 0.5 and 0.9 in that order for the following problems. It's important to note that the approximate conjugate gradient method is only exact when an exact line search is used, which is not the case in our experiments with a quadratic function (Ridge Regression).

Anisotropic Bowl The Anisotropic Bowl is a bowl-shaped function with a constraint to get Lipschitz continuous gradients:

$$\begin{aligned} f(\theta) &= \sum_{i=1}^{n} i \cdot \theta_{(i)}^4 + \frac{1}{2}\|\theta\|_2^2 \\ \text{subject to } &\|\theta\|_2 \leq \tau \end{aligned} \tag{14}$$

As in Meng & Chen (2011), we set $n = 500$, $\tau = 4$ and $\theta_0 = \frac{\tau}{\sqrt{n}}\mathbf{1}$. Thus $L = 12n\tau^2 + 1 = 96001$ and $\mu = 1$. Figure 1 shows the convergence results for our algorithms and the baselines. The algorithms terminate when $f(\theta) - f^* < 10^{-12}$. GeO-N and GeO take only 82 and 205 iterations

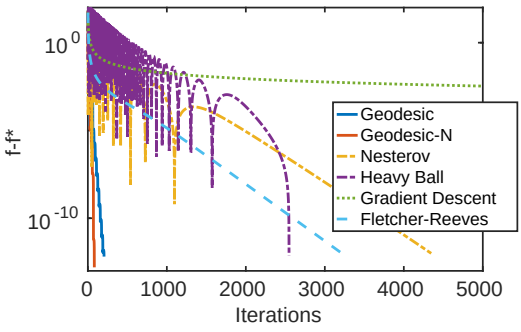


Figure 1: Anisotropic Bowl

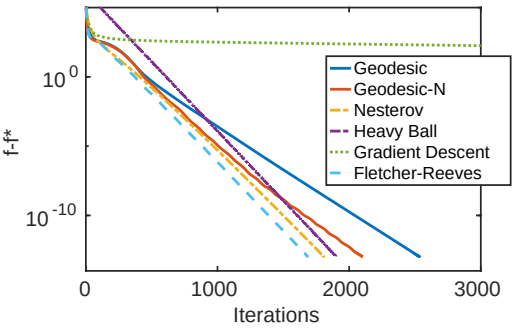


Figure 2: Ridge Regression

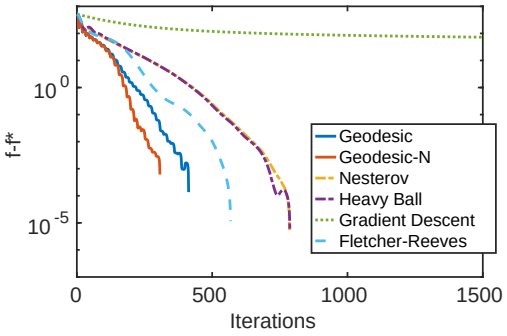


Figure 3: Smooth-BPDN

Figures 1 to 3: Results from experiments on strongly convex functions with Lipschitz gradients. Geodesic and Geodesic-N are our methods and the baselines are Gradient Descent, Heavy Ball, Nesterov and Fletcher-Reeves. All methods start from the same point and are terminated within a tolerance of the global optimum. The horizontal axes show the iterations and the vertical axes show the distance to the optimal value. Figures are best viewed in color.

to converge, while the closest result is that of Heavy-Ball and Fletcher-Reeves which take approximately 2500 and 3000 iterations respectively.

Ridge Regression The Ridge Regression problem is a linear least squares function with Tikhonov regularization:

$$f(\theta) = \frac{1}{2}\|A\theta - b\|_2^2 + \frac{\lambda}{2}\|\theta\|_2^2 \tag{15}$$

where $A \in \mathbb{R}^{m\times n}$ is a measurement matrix, $b \in \mathbb{R}^m$ is the response vector and $\gamma > 0$ is the ridge parameter. The function $f(\theta)$ is a positive definite quadratic function with the unique solution of $\theta^* = (A^TA + \lambda I)^{-1}A^Tb$ along with Lipschitz parameter $L = \|A\|_2^2 + \lambda$ and strong convexity parameter $\ddot{\mu} = \lambda$.

Following Meng & Chen (2011), $m = 1200$, $n = 2000$ and $\lambda = 1$. A is generated from $U\Sigma V^T$ where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times m}$ are random orthonormal matrices and $\Sigma \in \mathbb{R}^{m\times m}$ is diagonal with entries linearly distanced in $[100, 1]$ while $b = \text{randn}(m, 1)$ is drawn (i.i.d) from the standard normal distribution. Thus $\ddot{\mu} = 1$ and $L \approx 1001$. Figure 2 shows the results where Fletcher-Reeves, which is a conjugate gradient algorithm, performs better than other methods but we observe similar performances overall except for gradient descent. The tolerance is set to $f(\theta) - f^* < 10^{-13}$.

Smooth-BPDN Smooth-BPDN is a smooth and strongly convex version of the BPDN (basis pursuit denoising) problem:

$$\begin{aligned} f(\theta) &= \frac{1}{2}\|A\theta - b\|_2^2 + \lambda\|\theta\|_{\ell_1,\tau} + \frac{\rho}{2}\|\theta\|_2^2 \\ \text{where } \|\theta\|_{\ell_1,\tau} &= \begin{cases} |\theta| - \frac{\tau}{2} & \text{if } |\theta| \geq \tau \\ \frac{1}{2\tau}\theta^2 & \text{if } |\theta| < \tau \end{cases} \end{aligned} \tag{16}$$

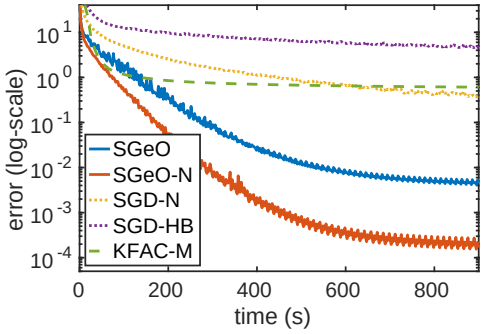


Figure 4: MNIST dataset.

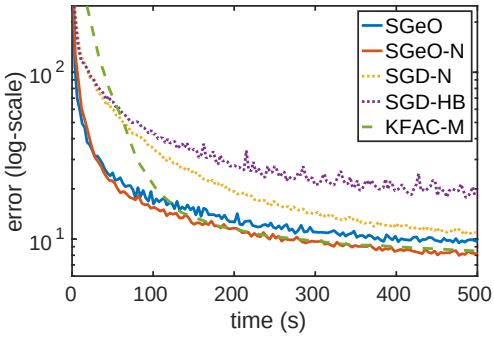


Figure 5: FACES dataset.

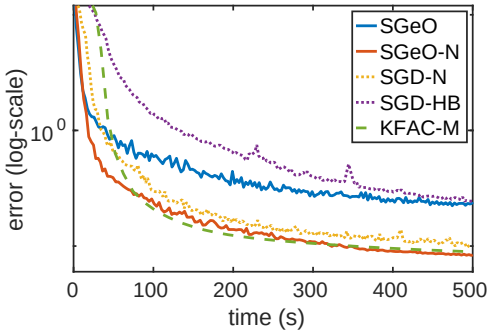


Figure 6: CURVES dataset

Figures 4 to 6: Results from autoencoder experiments on three datasets. The horizontal axis shows computation time and the vertical axis shows log-scale training error. Our methods are SGeO and SGeO-N and the baselines are SGD-HB and SGD-N, variants of SGD that use the Heavy Ball momentum and Nesterov's momentum respectively, along with K-FAC. All methods use the same initialization. SGeO-N is able to outperform other methods on the MNIST dataset and performs similarly to K-FAC on the other two while outperforming other baselines. Figures are best viewed in color.

and $\|\cdot\|_{\ell_1,\tau}$ is a smoothed version of the ℓ_1 norm also known as Huber penalty function with half-width of τ. The function is strongly convex $\ddot{\mu} = \rho$ because of the quadratic term $\frac{\rho}{2}\|\theta\|_2^2$ with Lipschitz constant $L = \|A\|_2^2 + \frac{\lambda}{\tau} + \rho$.

As in Meng & Chen (2011), we set $A = \frac{1}{\sqrt{n}} \cdot \text{randn}(m, 1)$ where $m = 800$, $n = 2000$, $\lambda = 0.05$, $\tau = 0.0001$ and $\ddot{\mu} = 0.05$. The real signal is a random vector with 40 nonzeros and $b = A\theta^* + e$ where $e = 0.01\frac{\|b\|_2}{\sqrt{m}} \cdot \text{randn}(m, 1)$ is Gaussian noise. Also $L = \|A\|_2^2 + \frac{\lambda}{\tau} + \rho \approx 502.7$. Since we cannot find the solution analytically, Nesterov's method is used as an approximation to the solution (f_N^*) and the tolerance is set to $f(\theta) - f_N^* < 10^{-12}$. Figure 3 shows the results for the algorithms. GeO-N and GeO converge in 308 and 414 iterations respectively, outperforming all other methods. Closest to these two is Fletcher-Reeves with 569 iterations and Nesterov and Heavy Ball converge similarly in 788 iterations.

5.2 Deep Autoencoders

To evaluate the performance of SGeO, we apply it to 3 benchmark deep autoencoder problems first introduced in Hinton & Salakhutdinov (2006) which use three datasets, MNIST, FACES and CURVES. Due to the difficulty of training these networks, they have become standard benchmarks for neural network optimization. To be consistent with previous literature (Martens, 2010; Sutskever et al., 2013; Martens & Grosse, 2015), we use the same network architectures as in Hinton & Salakhutdinov (2006) and also report the reconstruction error instead of the log-likelihood objective. The layer structure for MNIST, FACES and CURVES are [1000 500 250 30 250 500 1000], [2000 1000 500 30 500 1000 2000] and [400 200 100 50 25 6 25 50 100 200 400] respectively. All layers use sigmoid activations except the middle layer for MNIST, the middle and the last layer for FACES and the middle layer for CURVES which use linear activation.

Our baselines are the Heavy Ball algorithm (SGD-HB) (Polyak, 1964), SGD with Nesterov's Momentum (SGD-N) (Sutskever et al., 2013) and K-FAC (Martens & Grosse, 2015), a second-order method utilizing natural gradients using an approximation of the Fisher information matrix. Both the baselines and SGeO were implemented using MATLAB on GPU with single precision on a single machine with a 3.6 GHz Intel CPU and an NVIDIA GeForce GTX 1080 Ti GPU with 11 GBs of memory.

The results are shown in Figures 4 to 6. Since we are mainly interested in optimization and not in generalization, we only report the training error, although we have included the test set performances in the Appendix B. We report the reconstruction relative to the computation time to be able to compare with K-FAC, since each iteration of K-FAC takes orders of magnitude longer than SGD and SGeO. The per-iteration graphs can be found in the Appendix A.

All methods use the same parameter initialization scheme known as "sparse initialization" introduced in Martens (2010). The experiments for the Heavy Ball algorithm and SGD with Nesterov's momentum follow Sutskever et al. (2013) which were tuned to maximize performance for these problems. For SGeO, we chose a fixed momentum parameter and used a simple multiplicative schedule for the learning rate:

$$\epsilon_t = \epsilon_1 \times \beta_\epsilon^{\lfloor \frac{t}{K} \rfloor}$$

where the initial value (ϵ_1) was chosen from $\{0.1,0.15,0.2,0.3,0.4,0.5\}$ and is decayed (K) every 2000 iterations (parameter updates). The decay parameter (β_ϵ) was set to 0.95. For the momentum parameter μ, we did a search in $\{0.999,0.995,0.99\}$. The minibatch size was set to 500 for all methods except K-FAC which uses an exponentially increasing schedule for the minibatch size. For K-FAC we used the official code provided [1] by the authors with default parameters to reproduce the results. The version of K-FAC we ran was the Blk-Tri-Diag approach which achieves the best results in all three cases. To do a fair comparison with other methods, we disabled iterate averaging for K-FAC. It is also worth noting that K-FAC uses a form of momentum (Martens & Grosse, 2015).

In all three experiments, SGeO-N is able to outperform the baselines (in terms of reconstruction error) and performs similarly as (if not better than) K-FAC. We can see the effect of the adaptive coefficient on the Heavy Ball method, i.e. SGeO, which also outperforms SGD with Nesterov's momentum in two of the experiments, MNIST and FACES, and also outperforms K-FAC in the MNIST experiment. Use of Nesterov style lookahead gradient significantly accelerates training for the MNIST and CURVES dataset, while we see this to a lesser extent in the FACES dataset. This is also the case for the other baselines (Sutskever et al., 2013; Martens & Grosse, 2015). Further, we notice an interesting phenomena for the MNIST dataset (Figure 4). Both SGeO and SGeO-N reach very low error rates, after only 900 seconds of training, SGeO and SGeO-N arrive at an error of 0.004 and 0.0002 respectively.

6 Conclusion and future work

We proposed a novel and efficient algorithm based on adaptive coefficients for the Heavy Ball method inspired by a geodesic optimization algorithm. We compared SGeO against SGD with Nesterov's Momentum and regular momentum (Heavy Ball) and a recently proposed second-order method, K-FAC, on three deep autoencoder optimization benchmarks and three strongly convex functions with Lipschitz gradients. We saw that SGeO is able to outperform all first-order methods that we compared to, by a notable margin. SGeO is easy to implement and the computational overhead it has over the first-order methods, which is calculating the dot product, is marginal. It can also perform as effectively as or better than second-order methods (here, K-FAC) without the need for expensive higher-order operations in terms of time and memory. We believe that SGeO opens new and promising directions in high dimensional optimization research and in particular, neural network optimization. We are working on applying SGeO to other machine learning paradigms such as CNNs, RNNs and Reinforcement Learning. It remains to analyse the theoretical properties of SGeO such as its convergence rate in convex and non-convex cases which we leave for future work.

[1] http://www.cs.toronto.edu/ jmartens/docs/KFAC3-MATLAB.zip

REFERENCES

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

Jean François Aujol, Aude Rondepierre, JF Aujol, Charles Dossal, et al. Optimal convergence rates for nesterov acceleration. *arXiv preprint arXiv:1805.05719*, 2018.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

Aaron Defazio. On the curved geometry of accelerated optimization. *arXiv preprint arXiv:1812.04634*, 2018.

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.

Reeves Fletcher and Colin M Reeves. Function minimization by conjugate gradients. *The computer journal*, 7(2):149–154, 1964.

Ricky Fok, Aijun An, and Xiaogong Wang. Geodesic and contour optimization using conformal mapping. *Journal of Global Optimization*, 69(1):23–44, 2017.

Magnus Rudolph Hestenes and Eduard Stiefel. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS Washington, DC, 1952.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Manuel Laguna and Rafael Martí. Experimental testing of advanced scatter search designs for global optimization of multimodal functions. *Journal of Global Optimization*, 33(2):235–255, 2005.

James Lucas, Shengyang Sun, Richard Zemel, and Roger Grosse. Aggregated momentum: Stability through passive damping. *arXiv preprint arXiv:1804.00325*, 2018.

James Martens. Deep learning via hessian-free optimization. In *ICML*, volume 27, pp. 735–742, 2010.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417, 2015.

Xiangrui Meng and Hao Chen. Accelerating nesterov's method for strongly convex functions with lipschitz gradient. *arXiv preprint arXiv:1109.6058*, 2011.

Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pp. 543–547, 1983.

Brendan Odonoghue and Emmanuel Candes. Adaptive restart for accelerated gradient schemes. *Foundations of computational mathematics*, 15(3):715–732, 2015.

Victor Picheny, Tobias Wagner, and David Ginsbourger. A benchmark of kriging-based infill criteria for noisy optimization. *Structural and Multidisciplinary Optimization*, 48(3):607–626, 2013.

Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. 2018.

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.

S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved April 14, 2019, from `http://www.sfu.ca/~ssurjano`.

Ilya Sutskever, James Martens, George E Dahl, and Geoffrey E Hinton. On the importance of initialization and momentum in deep learning. *ICML (3)*, 28(1139-1147):5, 2013.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Andre Wibisono, Ashia C Wilson, and Michael I Jordan. A variational perspective on accelerated methods in optimization. *proceedings of the National Academy of Sciences*, 113(47):E7351–E7358, 2016.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

A AUTOENCODER PER-ITERATION RESULTS

Here we include the per-iteration results for the autoencoder experiments in Figures 7 to 9. We reported the reconstruction error vs. running time in the main text to make it easier to compare to K-FAC. K-FAC, which is a second-order algorithm, converges in fewer iterations but has a high per-iteration cost. All other methods are first-order and have similar per-iteration costs.

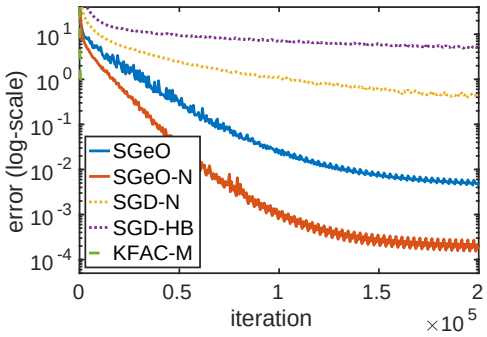


Figure 7: MNIST dataset.

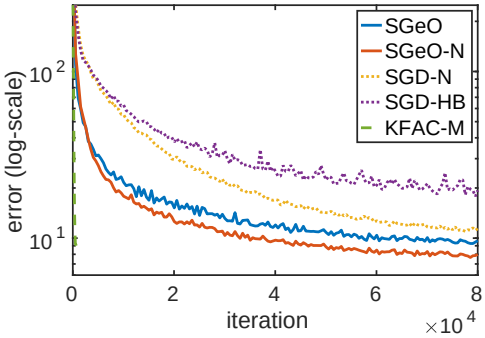


Figure 8: FACES dataset.

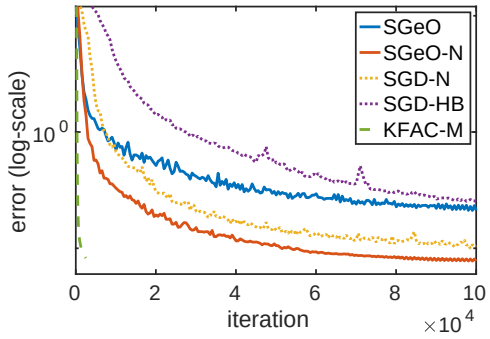


Figure 9: CURVES dataset

Figures 7 to 9: Results from autoencoder experiments on three datasets. The horizontal axis shows iterations and the vertical axis shows log-scale training error. K-FAC is a second-order method and takes much fewer iterations to converge. However, the per-iteration cost is much higher than the other methods. SGeO and SGeO-N are our methods and the baselines are Nesterov (SGD-N), Heavy Ball (SGD-HB) and KFAC-M (M indicating a form of momentum). Figures are best viewed in color.

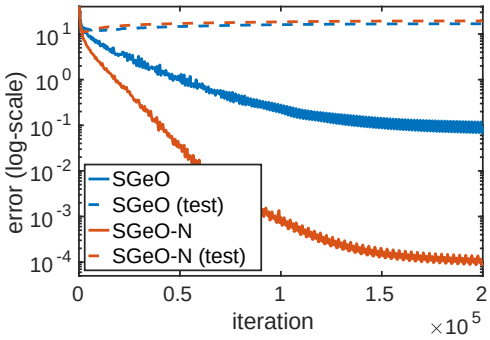


Figure 10: MNIST dataset.

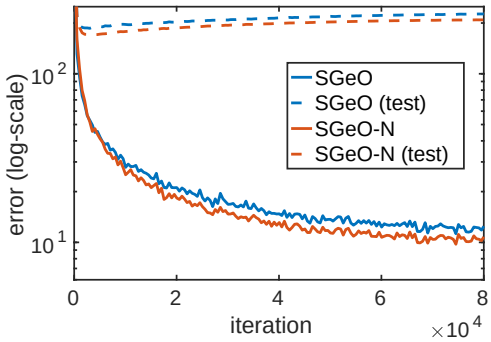


Figure 11: FACES dataset.

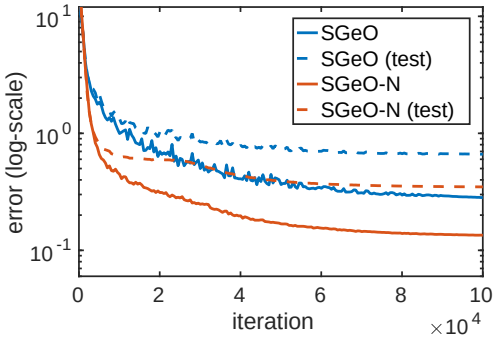


Figure 12: CURVES dataset

Figures 10 to 12 show the performance of the autoencoders evaluated on the test set while training. Note that in all cases the algorithms are tuned to maximize performance on the training set. The dashed lines show the test error while the continuous lines show the training error. The letter "N" in SGeO-N indicates the use of Nesterov style lookahead gradient. Figures are best viewed in color.

B Generalization Experiments

We include generalization experiments on the test set here. However, as mentioned before, our focus is optimization and not generalization, we are aware that the choice of optimizer can have a significant effect on the performance of a trained model in practise. Results are shown in Figures 10 to 12. SGeO-N shows a significant better predictive performance than SGeO on the CURVES data set and both perform similarly on the two other datasets.. Note that the algorithms are tuned for best performance on the training set. Overfitting can be dealt with in various ways such as using appropriate regularization during training and using a small validation set to tune the parameters.

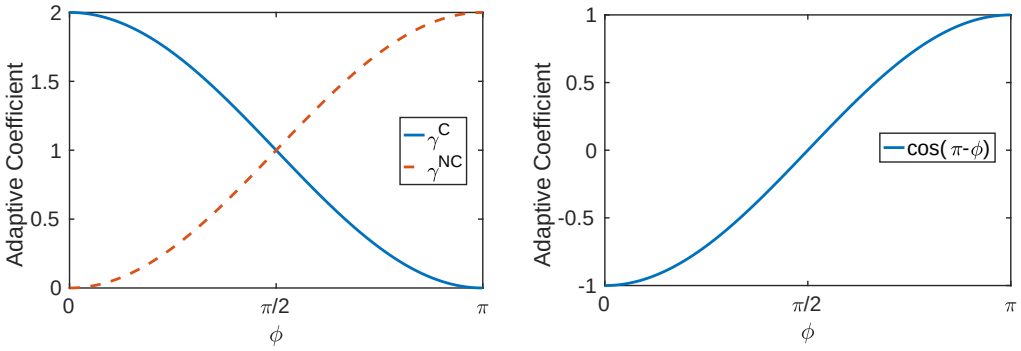
C Adaptive Coefficient behaviour

C.1 Geometric interpretation

Figure 13 (b) shows the dot product value $\bar{g} \cdot \bar{d}$ which is equivalent to $\cos(\pi - \phi)$ (where ϕ is the angle between the previous update and the negative of the current gradient) for different values of ϕ. Figure 13 (a) shows the adaptive coefficient (γ) behaviour for different values of ϕ for both convex and non-convex cases. Recall that the adaptive coefficient is used on top the Heavy Ball method. For strongly convex function with Lipschitz gradients we set $\gamma^C = 1 - \bar{g} \cdot \bar{d}$ and for non-convex cases $\gamma^{NC} = 1 + \bar{g} \cdot \bar{d}$.

C.2 Strongly convex functions with Lipschitz gradients

Here we include the values of the adaptive coefficient during optimization from our experiments. The plots in Figures 14 to 16 show γ_t at each iteration for GeO and GeO-N. For the Anisotropic



(a) Adaptive coefficient

(b) Cosine value

Figure 13: Adaptive Coefficient and cosine (dot product) values for the strongly convex $\gamma^C = 1 - \bar{g} \cdot \bar{d}$ and non-convex $\gamma^{NC} = 1 + \bar{g} \cdot \bar{d}$ cases where $\bar{g} \cdot \bar{d}$ is equal to $\cos(\pi - \phi)$.

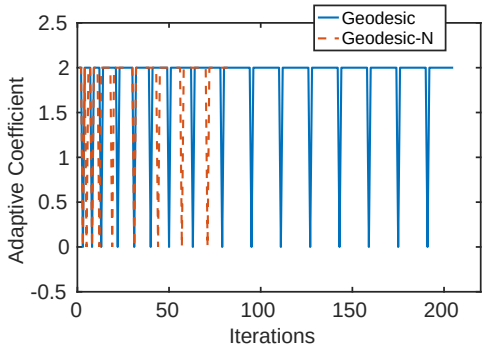


Figure 14: Anisotropic Bowl

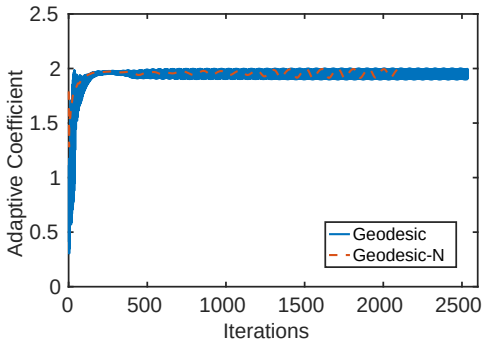


Figure 15: Ridge Regression

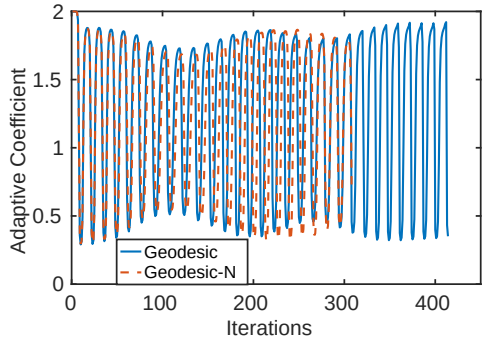


Figure 16: Smooth-BPDN

Figures 14 to 16 show the value of the adaptive coefficient per iteration during training for the strongly convex functions with Lipschitz gradients from our experiments. Note that here $\gamma_t^C = 1 - \bar{g}_t \cdot \bar{d}_t$. Figures are best viewed in color.

Bowl and the Smooth BPDN problem, the values fluctuates between 0 and 2 periodically, with the Smooth-BPDN behaving more sinusoidal and the Anisotropic Bowl more pulse-like. However, for the ridge regression problem, the values gradually increase from 0 in the beginning and stay close to 2 thereafter, indicating $\phi \approx 0$ (convergence) since $\gamma^C = 1 - \cos(\pi - \phi)$

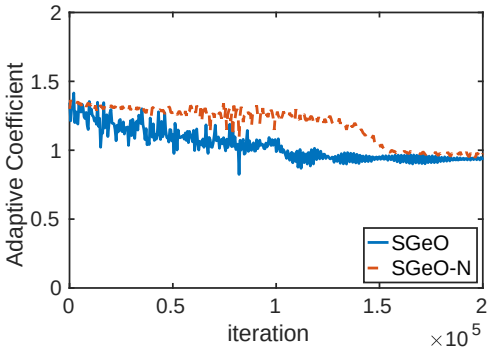


Figure 17: MNIST dataset.

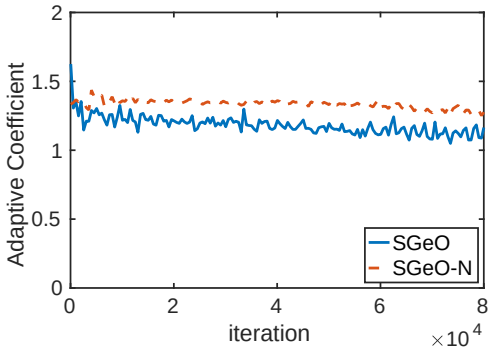


Figure 18: FACES dataset.

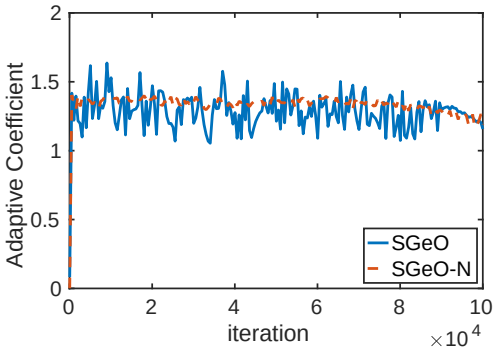


Figure 19: CURVES dataset

Figures 17 to 19 show the behaviour of the adaptive coefficient per iteration for the autoencoder experiments. Note that here $\gamma_t^{NC} = 1 + \bar{g}_t \cdot \bar{d}_t$ and the experiments are taken from optimization on the training set. We can see the effect of Nesterov's lookahead gradient on the autoencoder benchmarks. Figures are best viewed in color.

C.3 AUTOENCODERS

The adaptive coefficient values during training for the autoencoder experiments is shown in Figures 17 to 19. The values for all three problem stay close to values between 1 and 1.5 indicating $\frac{\pi}{2} < \phi < \frac{2\pi}{3}$. However, interpreting these values would not be as exact as our convex experiments due to partial gradient information, stochasticity and the non-convex nature of the problems. Further, adding lookahead gradient clearly decreases fluctuations in the adaptive coefficient value, thus a more stable training process. [t]

D GLOBAL OPTIMIZATION BENCHMARKS

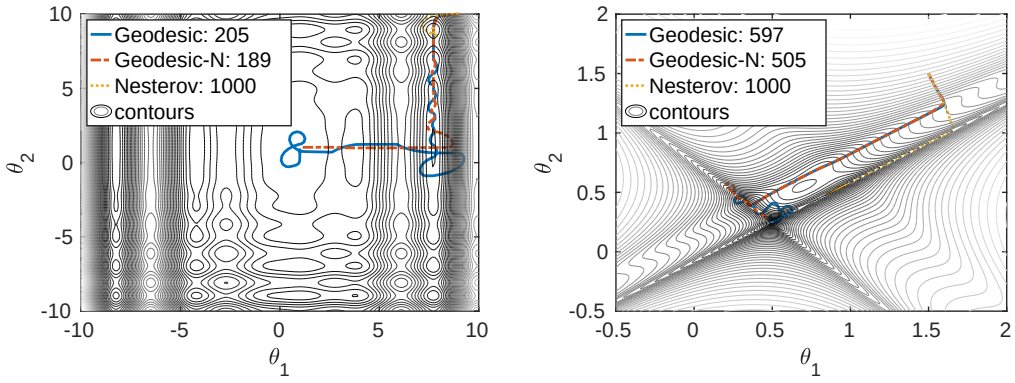
We compared our method (non-stochastic version using γ^{NC}) with Nesterov's method on global optimization benchmarks with two parameters to facilitate visualization. The results showing the contour plots and the paths taken by the algorithms can be found in Figure 20.

Levy Function The Levy function (Surjanovic & Bingham; Laguna & Martí, 2005) features a wavy surface in both directions, multiple ravines and local minima. The global minimum is at $(0, 0)$. The function is:

$$f(\theta) = \sin^2(\pi w_1) + (w_1 - 1)^2[1 + 10\sin^2(\pi w_1 + 1)] + (w_2 - 1)^2[1 + \sin^2(2\pi w_2)] \quad (17)$$

where $w_i = 1 + \frac{\theta_i - 1}{4}$ for $i = 1, 2$. We initialize all three methods at $(9, 10)$.

Scaled Goldstein-Price Function The scaled Godstein-Price function (Surjanovic & Bingham; Picheny et al., 2013) features several local minima, ravines and plateaus which can be representative



(a) Levy function.

(b) Goldstein-Price function.

Figure 20: Results from the scaled Goldstein-Price and the Levy function experiments. The figures show the contour plots of the functions and the paths taken by the methods in different colors. The numbers in front of each method's name in the legend show the total iterations it took for the algorithm to get to a tolerance of 0.001 from the global minimum or reached maximum iterations (1000). Geodesic is our method and N indicates the use of Nesterov's lookahead gradient. The global minimum for (a) is at (0,0) and for (b) is at (0.5,0.25). All methods start from the same point and the hyper-parameters are tuned for best performance. Although the algorithms used are not stochastic, the adaptive coefficient used for these problems is γ^{NC} since the functions are non-convex. Figures are best viewed in color.

of a deep neural network's loss function. The global minimum is at $(0.5, 0.25)$. The function is:

$$f(\theta) = \frac{1}{2.427}\Big[\log\Big([1+(\bar{\theta}_1+\bar{\theta}_2+1)^2(19-14\bar{\theta}_1+3\bar{\theta}_1^2-14\bar{\theta}_2+6\bar{\theta}_1\bar{\theta}_2+3\bar{\theta}_2^2)] \\ [30+(2\bar{\theta}_1-3\bar{\theta}_2)^2(18-32\bar{\theta}_1+12\bar{\theta}_1^2+48\bar{\theta}_2-36\bar{\theta}_1\bar{\theta}_2+27\bar{\theta}_2^2)]\Big) - 0.8693\Big] \tag{18}$$

where $\bar{\theta}_i = 4\bar{\theta}_i - 2$ for $i = 1, 2$. We initialize all methods at $(1.5, 1.5)$.

Details The momentum parameter μ for both Nesterov and Geodesic-N was set to 0.9. The learning rate for all methods is fixed and is tuned for best performance. The results from both experiments indicate that Geodesic is able to effectively escape local minima and recover from basins of attraction, while Nesterov's method gets stuck at local minima in both cases. We can also observe the effect of lookahead gradient on our method where the path taken by Geodesic-N is much smoother than Geodesic.