Third-order Smoothness Helps: Faster Stochastic Optimization Algorithms for Finding Local Minima

Yaodong Yu[*‡], **Pan Xu**[*†] and **Quanquan Gu**[†]

[‡]University of Virginia [†]University of California, Los Angeles

Stochastic Nonconvex Optimization

- **Optimization problem:**

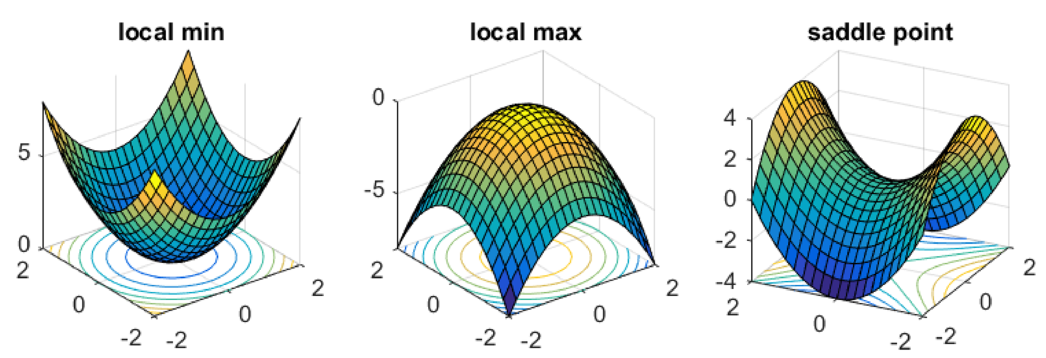
$$\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}) = \mathbb{E}_{\xi\sim\mathcal{D}}[F(\mathbf{x};\xi)]$$

  - $F(\mathbf{x};\xi):\mathbb{R}^d\rightarrow\mathbb{R}$ is a stochastic function
  - ξ is a random variable sampled from a fixed distribution $\mathcal{D}$
  - $f(\mathbf{x})$ is nonconvex

The (ϵ,ϵ_H)-second-order stationary point $\mathbf{x}$, i.e., approximate local minimum, is defined as

$$\|\nabla f(\mathbf{x})\|_2\le\epsilon,\ \text{and}\ \lambda_{\min}(\nabla^2 f(\mathbf{x}))\ge -\epsilon_H$$

- **Why approximate local minimum?**
  - A local minimum is adequate and can be as good as a global minimum in terms of generalization performance.
  - Many machine learning problems such as matrix completion, matrix sensing and phase retrieval, there is no spurious local minimum, i.e., all local minima are global minima.



Preliminaries

- **Geometric Distribution**
A random integer X follows a geometric distribution with parameter p, denoted as $\mathrm{Geom}(p)$, if it satisfies that

$$\mathbb{P}(X=k)=p^k(1-p),\quad \forall k=0,1,\ldots.$$

- **Smoothness**
(First-order smoothness) A differentiable function f is L_1-smooth, if

$$\|\nabla f(\mathbf{x})-\nabla f(\mathbf{y})\|_2\le L_1\|\mathbf{x}-\mathbf{y}\|_2,\quad \text{for all } \mathbf{x},\mathbf{y}\in\mathbb{R}^d.$$

- **Hessian Lipschitz**
(Second-order Smoothness) A twice-differentiable function f is L_2-Hessian Lipschitz, if

$$\|\nabla^2 f(\mathbf{x})-\nabla^2 f(\mathbf{y})\|_2\le L_2\|\mathbf{x}-\mathbf{y}\|_2,\quad \text{for all } \mathbf{x},\mathbf{y}\in\mathbb{R}^d.$$

- **Third-order Derivative**
The third-order derivative of function f: $\mathbb{R}^d\rightarrow\mathbb{R}$ is a three-way tensor $\nabla^3 f(\mathbf{x})\in\mathbb{R}^{d\times d\times d}$ which is defined as

$$[\nabla^3 f(\mathbf{x})]_{ijk}=\frac{\partial}{\partial x_i\partial x_j\partial x_k}f(\mathbf{x}),$$

for $i,j,k=1,\ldots,d$ and $\mathbf{x}\in\mathbb{R}^d$.

- **Third-order Derivative Lipschitz**
(Third-order Smoothness) A thrice-differentiable function f has L_3-Lipschitz third-order derivative, if

$$\|\nabla^3 f(\mathbf{x})-\nabla^3 f(\mathbf{y})\|_F\le L_3\|\mathbf{x}-\mathbf{y}\|_2,\quad \text{for all } \mathbf{x},\mathbf{y}\in\mathbb{R}^d.$$

Exploiting Third-order Smoothness

Suppose $\|\nabla f(\mathbf{x})\|_2\le$ and $\mathbf{x}$ is not an (ϵ,ϵ_H)-second-order stationary point, then there must exist a unit vector $\mathbf{v}$ such that

$$\mathbf{v}^\top\nabla^2 f(\mathbf{x})\,\mathbf{v}\le-\frac{\epsilon_H}{2}.$$

- **Without Third-order Smoothness**

$$\widetilde{\mathbf{y}}=\mathop{\mathrm{argmin}}_{\mathbf{y}\in\{\mathbf{u},\mathbf{w}\}} f(\mathbf{y}),\ \ \mathbf{u}=\mathbf{x}-\widetilde{\alpha}\,\mathbf{v},\ \mathbf{w}=\mathbf{x}+\widetilde{\alpha}\,\mathbf{v},$$

the step size $\widetilde{\alpha}$ can be set as $\widetilde{\alpha}=O(\epsilon_H/L_2)$, the negative curvature descent step is guaranteed to attain the following function value decrease

$$f(\widetilde{\mathbf{y}})-f(\mathbf{x})=-O(\epsilon_H^3/L_2^2).$$

- **With Third-order Smoothness**

$$\widehat{\mathbf{y}}=\mathop{\mathrm{argmin}}_{\mathbf{y}\in\{\mathbf{u},\mathbf{w}\}} f(\mathbf{y}),\ \ \mathbf{u}=\mathbf{x}-\widehat{\alpha}\,\mathbf{v},\ \mathbf{w}=\mathbf{x}+\widehat{\alpha}\,\mathbf{v},$$

the step size $\widehat{\alpha}$ can be set as $\widehat{\alpha}=O(\sqrt{\epsilon_H/L_3})$ which is larger than previous step size $\widetilde{\alpha}$, the negative curvature descent step is guaranteed to attain the following function value decrease

$$f(\widehat{\mathbf{y}})-f(\mathbf{x})=-O(\epsilon_H^2/L_3).$$

Theoretical Analysis

- **Negative Curvature Descent Step**

If the input $\mathbf{x}$ of the negative curvature algorithm (with larger step size) satisfies $\lambda_{\min}(\nabla^2 f(\mathbf{x}))<-\epsilon_H$, then with probability $1-\delta$, the algorithm will return $\widehat{\mathbf{y}}$ such that $\mathbb{E}_\zeta[f(\mathbf{x})-f(\widehat{\mathbf{y}})]\ge 3\epsilon_H^2/8L_3$, where $\mathbb{E}_\zeta$ denotes the expectation over the Rademacher random variable ζ. Furthermore, if we choose $\delta\le\epsilon_H/(3\epsilon_H+8L_2)$, it holds that

$$\mathbb{E}[f(\widehat{\mathbf{y}})-f(\mathbf{x})]\le-\frac{\epsilon_H^2}{8L_3},$$

where $\mathbb{E}$ is over all randomness of the algorithm, and the total runtime complexity is $\widetilde{O}\big((L_1^2/\epsilon_H^2)\big)$.

- **Total Runtime Complexity Analysis**

Let $f(\mathbf{x})=\mathbb{E}_{\xi\sim\mathcal{D}}[F(\mathbf{x};\xi)]$, suppose the third derivative of $f(\mathbf{x})$ is L_3-Lipschitz, and each stochastic function $F(\mathbf{x};\xi)$ is L_1-smooth and L_2-Hessian Lipschitz continuous. Suppose that the stochastic gradient $\nabla F(\mathbf{x};\xi)$ satisfies the gradient sub-Gaussian condition with parameter σ. Set batch size $B=\widetilde{O}(\sigma^2/\epsilon^2)$ and $\epsilon_H\gtrsim\epsilon^{2/3}$. If our algorithm **FLASH** adopts online algorithms, such as Oja's algorithm, to compute the negative curvature, then **FLASH** finds an (ϵ,ϵ_H)-second-order stationary point with probability at least $1/3$ in runtime

$$\widetilde{O}\bigg(\frac{L_1\sigma^{4/3}\Delta_f}{\epsilon^{10/3}}+\frac{L_3\sigma^2\Delta_f}{\epsilon^2\epsilon_H^2}+\frac{L_1^2L_3\Delta_f}{\epsilon_H^4}\bigg).$$

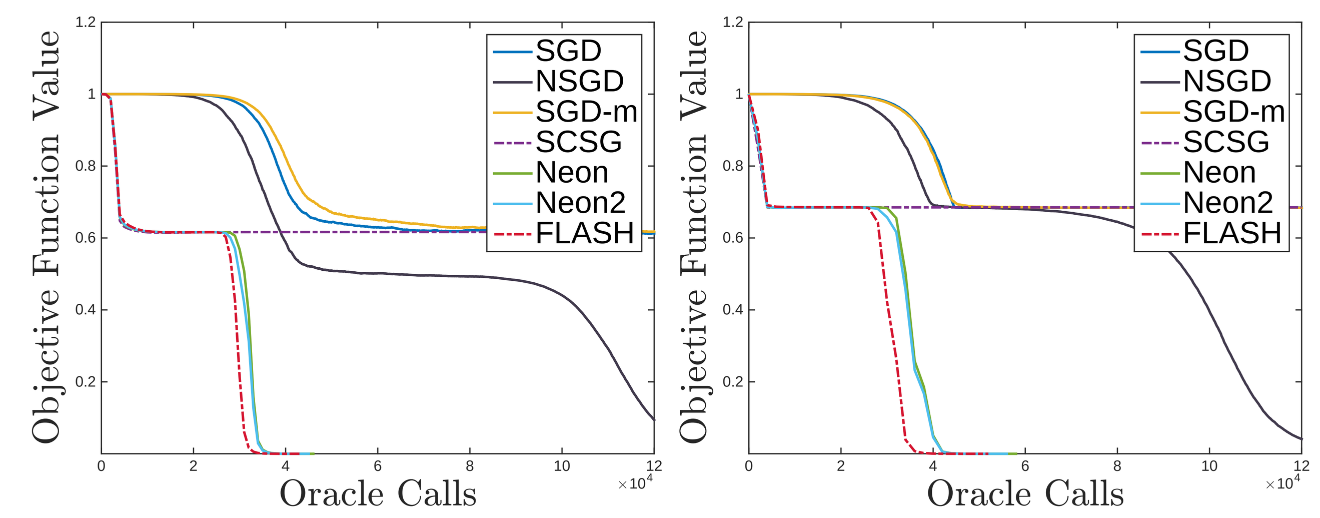
Numerical Experiments

- **Baseline Algorithms**
(1) stochastic gradient descent (**SGD**); (2) SGD with momentum (**SGD-m**); (3) noisy stochastic gradient descent (**NSGD**); (4) Stochastically Controlled Stochastic Gradient (**SCSG**); (5) NEgative-curvature-Originated-from-Noise (**Neon**); (6) NEgative-curvature-Originated-from-Noise 2 (**Neon2**).
- **Optimization Problems**
  - **Matrix Sensing**

$$\min_{\mathbf{U}\in\mathbb{R}^{d\times r}}\mathcal{L}(\mathbf{U})=\frac{1}{2m}\sum_{i=1}^m\big(\langle\mathbf{A}_i,\mathbf{U}\mathbf{U}^\top\rangle-b_i\big)^2$$

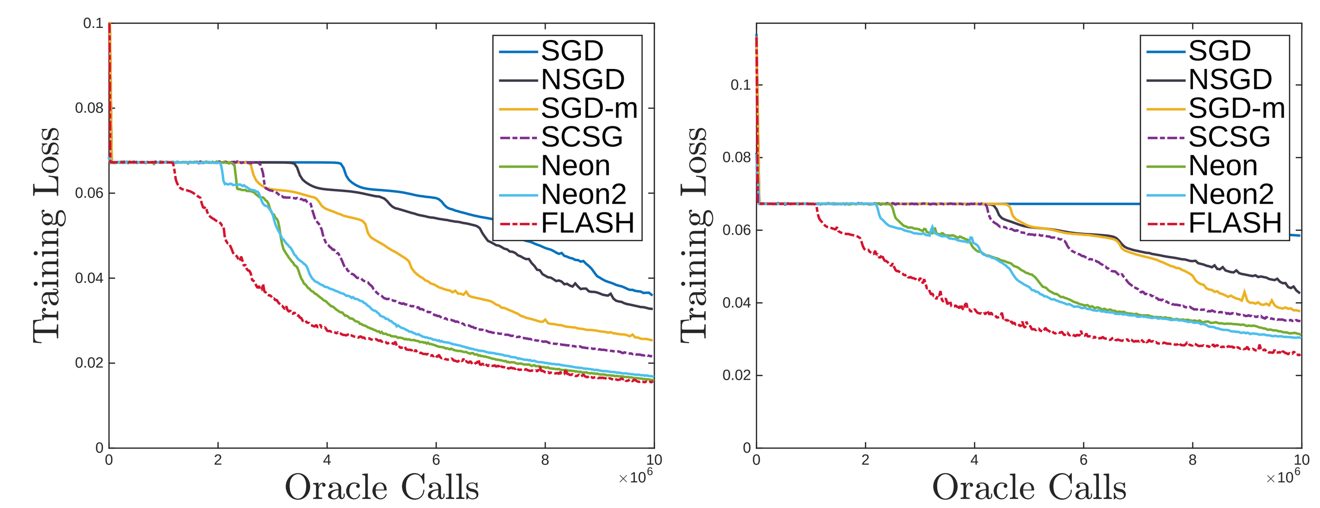
  - **Deep Autoencoder**

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^d}\mathcal{L}(\boldsymbol{\theta})=\frac{1}{2n}\sum_{i=1}^n\|\mathbf{x}_i-f(\mathbf{x}_i;\boldsymbol{\theta})\|_F^2$$



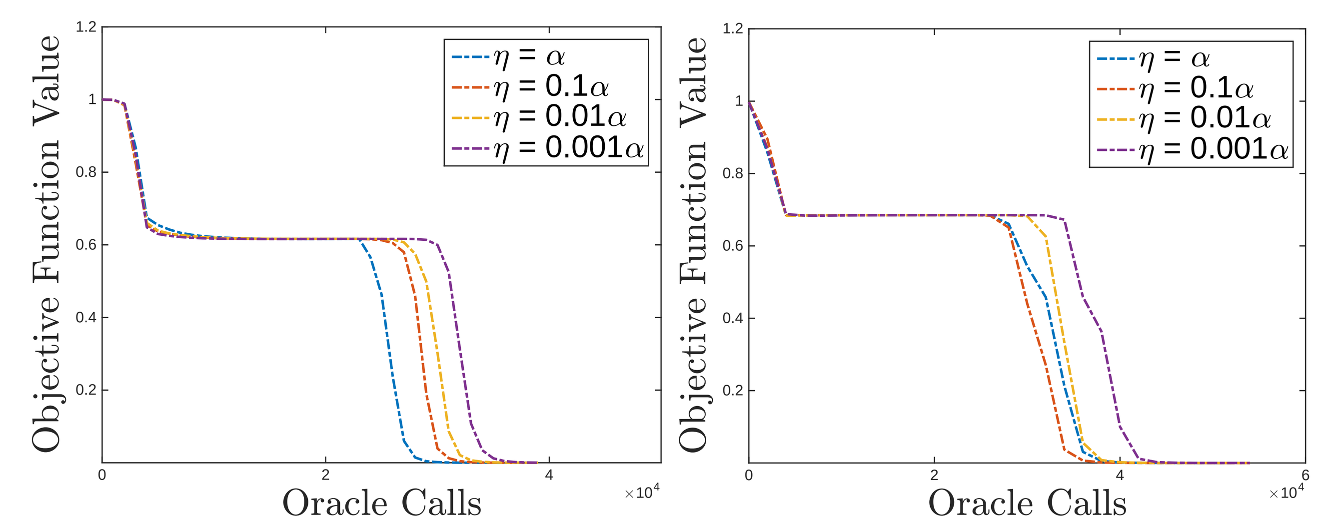
(a) Matrix Sensing ($d=50$) (b) Matrix Sensing ($d=100$)

Figure: Convergence of different algorithms for matrix sensing: objective function value versus the number of oracle calls.



(a) AE-1, Training (b) AE-2, Training

Figure: Convergence of different algorithms for two deep autoencoders: Training loss versus the number of oracle calls.



(a) Varying NC Step Size ($d=50$) (b) Varying NC Step Size ($d=100$)

Figure: Different negative curvature step size comparison of **FLASH** for matrix sensing.