SVGD: a Virtual Gradients Descent Method for Stochastic Optimization

Zheng Li and Shi Shu

September 12, 2019

SVGD: A VIRTUAL GRADIENTS DESCENT METHOD FOR STOCHASTIC OPTIMIZATION

A PREPRINT

Zheng Li[a] and Shi Shu*[a,b]

[a]*School of Mathematics and Computational Science, Xiangtan University, Xiangtan, Hunan, 411105, China*
[b]*Hunan Key Laboratory for Computation and Simulation in Science and Engineering, Xiangtan University, Xiangtan 411105, China*

September 3, 2019

ABSTRACT

Inspired by dynamic programming, we propose Stochastic Virtual Gradient Descent (SVGD) algorithm where the Virtual Gradient is defined by computational graph and automatic differentiation. The method is computationally efficient and has little memory requirements. We also analyze the theoretical convergence properties and implementation of the algorithm. Experimental results on multiple datasets and network models show that SVGD has advantages over other stochastic optimization methods.

Keywords computational graph · automatic differentiation · stochastic optimization · machine learning

1 Introduction

Stochastic gradient-based optimization is most widely used in many fields of science and engineering. In recent years, many scholars have compared SGD[1] with some adaptive learning rate optimization methods[2, 3]. [4] shows that adaptive methods often display faster initial progress on the training set, but their performance quickly plateaus on the development/test set. Therefore, many excellent models [5, 6, 7] still use SGD for training. However, SGD is greedy for the objective function with many multi-scale local convex regions (cf. Figure 1 of [8] or Fig. 1, left) because the negative of the gradient may not point to the minimum point on coarse-scale. Thus, the learning rate of SGD is difficult to set and significantly affects model performance[9].

Unlike greedy methods, dynamic programming (DP) [10] converges faster by solving simple sub-problems that decomposed from the original problem. Inspired by this, we propose the **virtual gradient** to construct a stochastic optimization method that combines the advantages of SGD and adaptive learning rate methods.

Consider a general objective function with the following composite form:

$$J = F(\boldsymbol{\sigma}), \quad \boldsymbol{\sigma} = \boldsymbol{f}(\boldsymbol{\theta}) \in \Omega_{\boldsymbol{\sigma}}, \tag{1}$$

where $\boldsymbol{\theta} \in \Omega_{\boldsymbol{\theta}} = \mathbb{R}^n, \Omega_{\boldsymbol{\sigma}} = \boldsymbol{f}(\Omega_{\boldsymbol{\theta}}) \subseteq \mathbb{R}^m$, functions F and each component function of $\boldsymbol{f}$ is first-order differentiable.

We note that:

$$F(\boldsymbol{\sigma}^*) = F(\boldsymbol{f}(\boldsymbol{\theta}^*)), \quad \boldsymbol{\sigma}^* = \underset{\boldsymbol{\sigma}\in\Omega_{\boldsymbol{\sigma}}}{\arg\min}\ F(\boldsymbol{\sigma}), \quad \boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}\in\Omega_{\boldsymbol{\theta}}}{\arg\min}\ F(\boldsymbol{f}(\boldsymbol{\theta})). \tag{2}$$

In addition, when we minimize $F(\boldsymbol{\sigma})$ and $F(\boldsymbol{f}(\boldsymbol{\theta}))$ with the same iterative method, the former should converge faster because the structure of F is simpler than $F \circ \boldsymbol{f}$. Based on these facts, we construct sequences $\{\boldsymbol{\sigma}^{(t)}\}$ and $\{\boldsymbol{\theta}^{(t)}\}$ that

*Corresponding author

This work is supported by the National Natural Science Foundation of China (Grant No. 11971414) and Key Research and Development Program of Hunan Province, P. R. China (Grant No. 2017SK2014).

Email addresses: lizheng.math.ai@gmail.com (Zheng Li), shushi@xtu.edu.cn (Shi Shu)

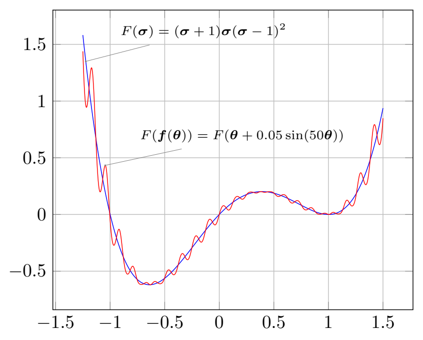
converge to $\boldsymbol{\sigma}^*$ and $\boldsymbol{\theta}^*$, respectively, with equations:

$$\boldsymbol{\sigma}^{(t)} = \boldsymbol{f}(\boldsymbol{\theta}^{(t)}), t = 0, 1, \cdots . \tag{3}$$

Fig. 1 (right) shows the relationship between $\{\boldsymbol{\sigma}^{(t)}\}$ and $\{\boldsymbol{\theta}^{(t)}\}$. The sequence $\{\boldsymbol{\sigma}^{(t)}\}$ can be obtained by using first-order iterative methods (see Sec.5 for details):

$$\boldsymbol{\sigma}^{(t+1)} = \boldsymbol{\sigma}^{(t)} - \alpha \mathscr{T}^* \nabla_{\boldsymbol{\sigma}} J\big|_{\boldsymbol{\sigma}=\boldsymbol{\sigma}^{(t)}}, \tag{4}$$

where α is the learning rate, $\mathscr{T}^*$ is an operator of mappping $\mathbb{R}^m \to \mathbb{R}^m$.



$$\begin{tikzcd}
\cdots \arrow[r] & \boldsymbol{\sigma}^{(t)} \arrow[r] \arrow[d] & \boldsymbol{\sigma}^{(t+1)} \arrow[r] \arrow[d] & \cdots \arrow[r] & \boldsymbol{\sigma}^* \\
\cdots \arrow[r] & \boldsymbol{\theta}^{(t)} \arrow[r] & \boldsymbol{\theta}^{(t+1)} \arrow[r] & \cdots \arrow[r] & \boldsymbol{\theta}^*
\end{tikzcd}$$

Figure 1

The difficulty in constructing operator $\mathscr{T}^*$ is how to make the condition (3) holds true. Let $\boldsymbol{M} = \left(\frac{\partial \boldsymbol{f}_i(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_j}\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}} \right)_{m\times n}$, $\mathscr{T}$ is an operator of mapping $\mathbb{R}^m \to \mathbb{R}^m$, we give the following iterations:

$$\boldsymbol{\sigma}^{(t+1)} = \boldsymbol{\sigma}^{(t)} - \alpha \boldsymbol{M}\boldsymbol{M}^T \mathscr{T} \nabla_{\boldsymbol{\sigma}} J\big|_{\boldsymbol{\sigma}=\boldsymbol{\sigma}^{(t)}}, \tag{5}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \boldsymbol{M}^T \mathscr{T} \nabla_{\boldsymbol{\sigma}} J\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}. \tag{6}$$

Since $\boldsymbol{M}^T \mathscr{T} \nabla_{\boldsymbol{\sigma}} J$ in Eqn.(6) is equivalent to the position of $\nabla_{\boldsymbol{\theta}} J$ in gradient descent method, we define $\boldsymbol{M}^T \mathscr{T} \nabla_{\boldsymbol{\sigma}} J$ as the **virtual gradient** of function J for variable $\boldsymbol{\theta}$.

For Eqn.(6), it is easy to prove that the condition (3) holds when $\boldsymbol{f}$ is a linear mapping. If $\boldsymbol{f}$ is a nonlinear mapping, let the second-derivatives of $\boldsymbol{f}$ be bounded and $\alpha = o(1), \boldsymbol{\sigma}^{(t)} = \boldsymbol{f}(\boldsymbol{\theta}^{(t)})$, owing to (5) and (6) and Taylor formula, the following holds true:

$$\begin{aligned}
||\boldsymbol{\sigma}^{(t+1)} - \boldsymbol{f}(\boldsymbol{\theta}^{(t+1)})|| &= ||(\boldsymbol{\sigma}^{(t)} - \alpha \boldsymbol{M}\boldsymbol{M}^T \mathscr{T} \nabla_{\boldsymbol{\sigma}} J\big|_{\boldsymbol{\sigma}=\boldsymbol{\sigma}^{(t)}}) - \boldsymbol{f}(\boldsymbol{\theta}^{(t)} - \alpha \boldsymbol{M}^T \mathscr{T} \nabla_{\boldsymbol{\sigma}} J\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}})|| \\
&= O(||\alpha \boldsymbol{M}^T \mathscr{T} \nabla_{\boldsymbol{\sigma}} J\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}||_2^2) = O(\alpha^2),
\end{aligned} \tag{7}$$

In this case, the condition (3) holds, approximately.

According to the analysis above, the sequence $\{F(\boldsymbol{f}(\boldsymbol{\theta}^{(t)}))\}$ yields similar convergence as $\{F(\boldsymbol{\sigma}^{(t)})\}$ in Eqn.(6) and Eqn.(5), but faster than minimizing the function $F(\boldsymbol{f}(\boldsymbol{\theta}))$ with the same first-order method, directly.

Note that the iterative method (6) is derived based on the composite form (1) and this form is generally not unique, it is inconvenient for our algorithm design. We begin by introducing the computational graph. It is a directed graph, where each node indicates a variable that may be a scalar, vector, matrix, tensor, or even a variable of another type, and each edge unique corresponds to an operation which maps a node to another. We sometimes annotate the output node with the name of the operation applied. In particular, the computational graph corresponding to the objective function is a DAG(directed acyclic graphs) [11]. For example, the computational graph of the objective function J shown in Fig. 2 (a), the corresponding composite form (1) is:

$$\begin{cases}
J = F(\boldsymbol{\sigma}), \boldsymbol{\sigma} = \begin{bmatrix} \tilde{\boldsymbol{\sigma}}_1 \\ \tilde{\boldsymbol{\sigma}}_2 \end{bmatrix} = \boldsymbol{f}(\boldsymbol{\theta}) = \begin{bmatrix} \tilde{\boldsymbol{f}}_1(\tilde{\boldsymbol{\theta}}_1, \tilde{\boldsymbol{\theta}}_2) \\ \tilde{\boldsymbol{f}}_2(\tilde{\boldsymbol{\theta}}_2, \tilde{\boldsymbol{\theta}}_3; \tilde{\boldsymbol{\sigma}}_1) \end{bmatrix}, \quad \boldsymbol{\theta} = [\tilde{\boldsymbol{\theta}}_1^T, \tilde{\boldsymbol{\theta}}_2^T, \tilde{\boldsymbol{\theta}}_3^T]^T \\
\tilde{\boldsymbol{f}}_1 : \mathbb{R}^{n_1^{\boldsymbol{\theta}}} \times \mathbb{R}^{n_2^{\boldsymbol{\theta}}} \to \mathbb{R}^{n_1^{\boldsymbol{\sigma}}}, \quad \tilde{\boldsymbol{f}}_2 : \mathbb{R}^{n_2^{\boldsymbol{\theta}}} \times \mathbb{R}^{n_3^{\boldsymbol{\theta}}} \times \mathbb{R}^{n_1^{\boldsymbol{\sigma}}} \to \mathbb{R}^{n_2^{\boldsymbol{\sigma}}}
\end{cases}. \tag{8}$$

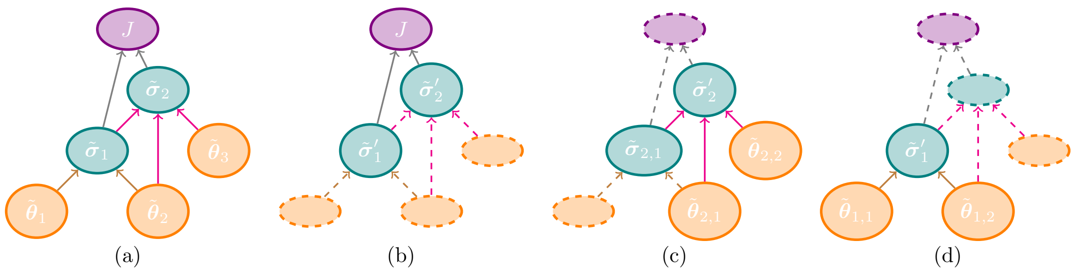


Figure 2: 🟠, 🟢, 🟣 are nodes associated with leaf values, hidden values and output value, and $\longrightarrow$, $\longrightarrow$, $\longrightarrow$ are edges associated with operations $\tilde{\boldsymbol{f}}_1, \tilde{\boldsymbol{f}}_2, F$.

For a given general objective function, let $\boldsymbol{G}$ correspond to a computational graph that maps the set of leaf values $\boldsymbol{V}_{\boldsymbol{\theta}}^{\boldsymbol{G}} = \{\tilde{\boldsymbol{\theta}}_j | j = 1, \cdots, N\}$ to the output value J, where the set of hidden values is $\boldsymbol{V}_{\boldsymbol{\sigma}}^{\boldsymbol{G}} = \{\tilde{\boldsymbol{\sigma}}_i | i = 1, \cdots, M\}$. Let $\boldsymbol{V}_{\boldsymbol{\theta}\to\boldsymbol{\sigma}}^{\boldsymbol{G}} := \{\tilde{\boldsymbol{\sigma}}'_i | i = 1, \cdots, M'\} = \{\tilde{\boldsymbol{\sigma}}_i \in \boldsymbol{V}_{\boldsymbol{\sigma}}^{\boldsymbol{G}} | \mathrm{dist}(\tilde{\boldsymbol{\sigma}}_i, \boldsymbol{V}_{\boldsymbol{\theta}}^{\boldsymbol{G}}) = 1\}$. In this paper, the objective function J in Eqn.(1) will be expressed as the following composite form:

$$J = F(\boldsymbol{\sigma}), \boldsymbol{\sigma} = \begin{bmatrix} \tilde{\boldsymbol{\sigma}}'_1 \\ \vdots \\ \tilde{\boldsymbol{\sigma}}'_{M'} \end{bmatrix} = \tilde{\boldsymbol{f}}(\boldsymbol{\theta}) = \begin{bmatrix} \tilde{\boldsymbol{f}}_1(\tilde{\boldsymbol{\theta}}_{1,1}, \cdots, \tilde{\boldsymbol{\theta}}_{1,N_1}; \tilde{\boldsymbol{\sigma}}'_{1,1}, \cdots, \tilde{\boldsymbol{\sigma}}'_{1,N'_1}) \\ \vdots \\ \tilde{\boldsymbol{f}}_{M'}(\tilde{\boldsymbol{\theta}}_{M',1}, \cdots, \tilde{\boldsymbol{\theta}}_{M',N_{M'}}; \tilde{\boldsymbol{\sigma}}'_{M',1}, \cdots, \tilde{\boldsymbol{\sigma}}'_{M',N'_{M'}}) \end{bmatrix} \tag{9}$$

where $\tilde{\boldsymbol{\sigma}}'_i = \tilde{\boldsymbol{f}}_i(\tilde{\boldsymbol{\theta}}_{i,1}, \cdots, \tilde{\boldsymbol{\theta}}_{i,N_i}; \tilde{\boldsymbol{\sigma}}'_{i,1}, \cdots, \tilde{\boldsymbol{\sigma}}'_{i,N'_i}) \in \boldsymbol{V}_{\boldsymbol{\theta}\to\boldsymbol{\sigma}}^{\boldsymbol{G}}, i = 1, \cdots, M'$, and

$$\begin{cases} \{\tilde{\boldsymbol{\theta}}_{i,1}, \cdots, \tilde{\boldsymbol{\theta}}_{i,N_i}\} = \{\tilde{\boldsymbol{\theta}}_k \in \boldsymbol{V}_{\boldsymbol{\theta}}^{\boldsymbol{G}} |\ \mathrm{dist}(\tilde{\boldsymbol{\theta}}_k, \tilde{\boldsymbol{\sigma}}'_i) = 1\} \\ \{\tilde{\boldsymbol{\sigma}}'_{i,1}, \cdots, \tilde{\boldsymbol{\sigma}}'_{i,N'_i}\} = \{\tilde{\boldsymbol{\sigma}}_k \in \boldsymbol{V}_{\boldsymbol{\sigma}}^{\boldsymbol{G}} |\ \mathrm{dist}(\tilde{\boldsymbol{\sigma}}_k, \tilde{\boldsymbol{\sigma}}'_i) = 1, \tilde{\boldsymbol{\sigma}}_k \preceq \tilde{\boldsymbol{\sigma}}'_i\} \end{cases}.$$

For example, Eqn.(8) can be expressed as:

$$J = F(\boldsymbol{\sigma}), \boldsymbol{\sigma} = \begin{bmatrix} \tilde{\boldsymbol{\sigma}}'_1 \\ \tilde{\boldsymbol{\sigma}}'_2 \end{bmatrix} = \tilde{\boldsymbol{f}}(\boldsymbol{\theta}) = \begin{bmatrix} \tilde{\boldsymbol{f}}_1(\tilde{\boldsymbol{\theta}}_{1,1}, \tilde{\boldsymbol{\theta}}_{1,2}) \\ \tilde{\boldsymbol{f}}_2(\tilde{\boldsymbol{\theta}}_{2,1}, \tilde{\boldsymbol{\theta}}_{2,2}; \tilde{\boldsymbol{\sigma}}_{2,1}) \end{bmatrix},$$

where

$$\begin{cases} \tilde{\boldsymbol{\sigma}}'_1 = \tilde{\boldsymbol{\sigma}}_1, \tilde{\boldsymbol{\theta}}_{1,1} = \tilde{\boldsymbol{\theta}}_1, \tilde{\boldsymbol{\theta}}_{1,2} = \tilde{\boldsymbol{\theta}}_2 \\ \tilde{\boldsymbol{\sigma}}'_2 = \tilde{\boldsymbol{\sigma}}_2, \tilde{\boldsymbol{\sigma}}_{2,1} = \tilde{\boldsymbol{\sigma}}_1, \tilde{\boldsymbol{\theta}}_{2,1} = \tilde{\boldsymbol{\theta}}_2, \tilde{\boldsymbol{\theta}}_{2,2} = \tilde{\boldsymbol{\theta}}_3 \end{cases}.$$

In deeping learning, the gradient of the objective function is usually calculated by the Automatic Differentiation (AD) technique[12, 9]. Our following example introduces how to calculate the gradient of J in Eqn.(8) using AD technique.

1. Find the Operation F associated with output value J and its input node $\{\tilde{\boldsymbol{\sigma}}'_1, \tilde{\boldsymbol{\sigma}}'_2\}$, cf. Fig. 2 (b). Then, calculate the following gradients:

$$\boldsymbol{g}_{\boldsymbol{w}}^J := \boldsymbol{g}_{\boldsymbol{w}\to J}(\tilde{\boldsymbol{\sigma}}'_1, \tilde{\boldsymbol{\sigma}}'_2) = \nabla_{\boldsymbol{w}} F, \quad \boldsymbol{w} = \tilde{\boldsymbol{\sigma}}'_1, \tilde{\boldsymbol{\sigma}}'_2.$$

2. Perform the following steps by the partial order $\succeq$ of $\{\tilde{\boldsymbol{\sigma}}'_1, \tilde{\boldsymbol{\sigma}}'_2\}$:

   (a) Find Operation $\tilde{\boldsymbol{f}}_2$ and it's input nodes $\{\tilde{\boldsymbol{\theta}}_{2,1}, \tilde{\boldsymbol{\theta}}_{2,2}, \tilde{\boldsymbol{\sigma}}'_{2,1}\}$ which associated with hidden value $\tilde{\boldsymbol{\sigma}}'_2$, cf. Fig. 2 (c). Let:

$$F_2(\tilde{\boldsymbol{\theta}}_{2,1}, \tilde{\boldsymbol{\theta}}_{2,2}; \tilde{\boldsymbol{\sigma}}'_{2,1}) := F(\cdot, \tilde{\boldsymbol{f}}_2(\tilde{\boldsymbol{\theta}}_{2,1}, \tilde{\boldsymbol{\theta}}_{2,2}; \tilde{\boldsymbol{\sigma}}'_{2,1})),$$

   where '$\cdot$' denotes that it is treated as a constant during the calculation of gradients and will not be declared later. Calculate the following gradients:

$$\boldsymbol{g}_{\boldsymbol{w}}^{\tilde{\boldsymbol{\sigma}}'_2} := \boldsymbol{g}_{\boldsymbol{w}\to\tilde{\boldsymbol{\sigma}}'_2}(\nabla_{\tilde{\boldsymbol{\sigma}}'_2} J; \tilde{\boldsymbol{\theta}}_{2,1}, \tilde{\boldsymbol{\theta}}_{2,2}; \tilde{\boldsymbol{\sigma}}'_{2,1}) = \nabla_{\boldsymbol{w}} F_2 = \left(\frac{\partial \tilde{\boldsymbol{\sigma}}'_2}{\partial \boldsymbol{w}}\right)^T \nabla_{\tilde{\boldsymbol{\sigma}}'_2} J, \quad \boldsymbol{w} = \tilde{\boldsymbol{\theta}}_{2,1}, \tilde{\boldsymbol{\theta}}_{2,2}, \tilde{\boldsymbol{\sigma}}'_{2,1},$$

   where $\nabla_{\tilde{\boldsymbol{\sigma}}'_2} J = \nabla_{\tilde{\boldsymbol{\sigma}}'_2} F$.

(b) Find Operation $\tilde{\boldsymbol{f}}_1$ and it's input nodes $\{\tilde{\boldsymbol{\theta}}_{1,1}, \tilde{\boldsymbol{\theta}}_{1,2}\}$ which associated with hidden value $\tilde{\boldsymbol{\sigma}}'_1$, cf. Fig. 2 (d). Let:

$$F_1(\tilde{\boldsymbol{\theta}}_{1,1}, \tilde{\boldsymbol{\theta}}_{1,2}) := F(\tilde{\boldsymbol{f}}_1(\tilde{\boldsymbol{\theta}}_{1,1}, \tilde{\boldsymbol{\theta}}_{1,2}), \tilde{\boldsymbol{f}}_2(\cdot, \cdot, \tilde{\boldsymbol{f}}_1(\tilde{\boldsymbol{\theta}}_{1,1}, \tilde{\boldsymbol{\theta}}_{1,2}))).$$

Calculate following gradients:

$$\boldsymbol{g}_{\boldsymbol{w}}^{\tilde{\boldsymbol{\sigma}}'_1} := \boldsymbol{g}_{\boldsymbol{w}\to\tilde{\boldsymbol{\sigma}}'_1}(\nabla_{\tilde{\boldsymbol{\sigma}}'_1} J; \tilde{\boldsymbol{\theta}}_{1,1}, \tilde{\boldsymbol{\theta}}_{1,2}) = \nabla_{\boldsymbol{w}} F_1 = \left(\frac{\partial \tilde{\boldsymbol{\sigma}}'_1}{\partial \boldsymbol{w}}\right)^T \nabla_{\tilde{\boldsymbol{\sigma}}'_1} J, \quad \boldsymbol{w} = \tilde{\boldsymbol{\theta}}_{1,1}, \tilde{\boldsymbol{\theta}}_{1,2},$$

where $\nabla_{\tilde{\boldsymbol{\sigma}}'_1} J = \nabla_{\tilde{\boldsymbol{\sigma}}'_1} F + \nabla_{\tilde{\boldsymbol{\sigma}}'_1} F_2$.

3. Calculate the gradients of J:

$$\nabla_{\tilde{\boldsymbol{\theta}}_1} J = \boldsymbol{g}_{\tilde{\boldsymbol{\theta}}_{1,1}}^{\tilde{\boldsymbol{\sigma}}'_1}, \nabla_{\tilde{\boldsymbol{\theta}}_2} J = \boldsymbol{g}_{\tilde{\boldsymbol{\theta}}_{1,2}}^{\tilde{\boldsymbol{\sigma}}'_1} + \boldsymbol{g}_{\tilde{\boldsymbol{\theta}}_{2,1}}^{\tilde{\boldsymbol{\sigma}}'_2}, \nabla_{\tilde{\boldsymbol{\theta}}_3} J = \boldsymbol{g}_{\tilde{\boldsymbol{\theta}}_{2,2}}^{\tilde{\boldsymbol{\sigma}}'_2}.$$

According to the analysis above, the computational graph of $\{\nabla_{\tilde{\boldsymbol{\theta}}_k} J | \ k = 1, 2, 3\}$ can be shown as Fig. 3 (a). If $\mathscr{T}$ is a broadcast-like operator, the computational graph of **vitrual gradients** can be shown as Fig. 3 (b), where $z_1 = \boldsymbol{g}_{\tilde{\boldsymbol{\theta}}_2\to\tilde{\boldsymbol{\sigma}}_1}(\mathscr{T}\nabla_{\tilde{\boldsymbol{\sigma}}_1} J; \tilde{\boldsymbol{\theta}}_1, \tilde{\boldsymbol{\theta}}_2)$, $z_2 = \boldsymbol{g}_{\tilde{\boldsymbol{\theta}}_2\to\tilde{\boldsymbol{\sigma}}_2}(\mathscr{T}\nabla_{\tilde{\boldsymbol{\sigma}}_2} J; \tilde{\boldsymbol{\theta}}_2, \tilde{\boldsymbol{\theta}}_3; \tilde{\boldsymbol{\sigma}}_1)$ and $\{\nabla_{\tilde{\boldsymbol{\theta}}_k}^{(\boldsymbol{G},\mathscr{T})} J | \ k = 1, 2, 3\}$ is defined by the Eqn.(10).

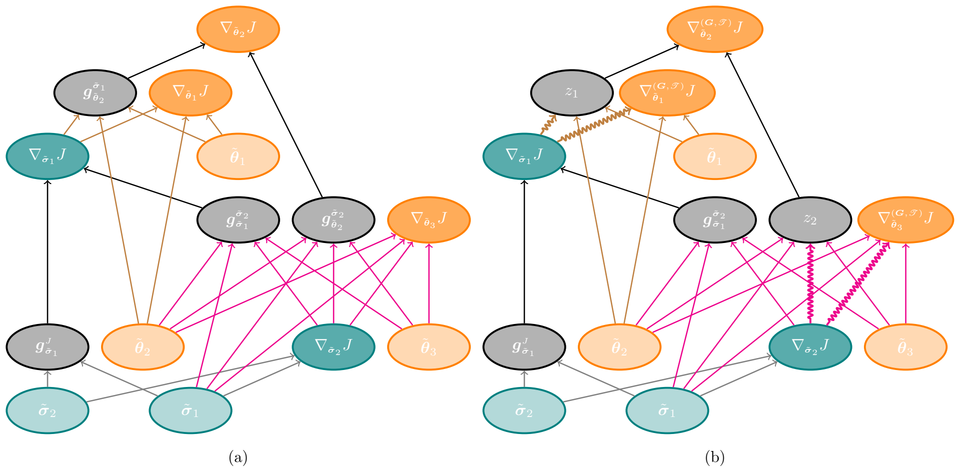

$\nabla_{\tilde{\theta}_2} J$
$g_{\tilde{\theta}_2}^{\tilde{\sigma}_1}$
$\nabla_{\tilde{\theta}_1} J$
$\nabla_{\tilde{\sigma}_1} J$
$\tilde{\theta}_1$
$g_{\tilde{\sigma}_1}^{\tilde{\sigma}_2}$
$g_{\tilde{\theta}_2}^{\tilde{\sigma}_2}$
$\nabla_{\tilde{\theta}_3} J$
$g_{\tilde{\sigma}_1}^{J}$
$\tilde{\theta}_2$
$\nabla_{\tilde{\sigma}_2} J$
$\tilde{\theta}_3$
$\tilde{\sigma}_2$
$\tilde{\sigma}_1$
(a)
$\nabla_{\tilde{\theta}_2}^{(G,\mathscr{T})} J$
z_1
$\nabla_{\tilde{\theta}_1}^{(G,\mathscr{T})} J$
$\nabla_{\tilde{\sigma}_1} J$
$\tilde{\theta}_1$
$g_{\tilde{\sigma}_1}^{\tilde{\sigma}_2}$
z_2
$\nabla_{\tilde{\theta}_3}^{(G,\mathscr{T})} J$
$g_{\tilde{\sigma}_1}^{J}$
$\tilde{\theta}_2$
$\nabla_{\tilde{\sigma}_2} J$
$\tilde{\theta}_3$
$\tilde{\sigma}_2$
$\tilde{\sigma}_1$
(b)

Figure 3: $\longrightarrow$ are edges associated with operation $+$ and $x \rightsquigarrow y$ denotes $\mathscr{T}x \longrightarrow y$.

According to the definition of **virtual gradient**, for any $\tilde{\boldsymbol{\theta}}_k \in \boldsymbol{V}_{\boldsymbol{\theta}}^{\boldsymbol{G}}$:

$$\begin{aligned}\nabla_{\tilde{\boldsymbol{\theta}}_k}^{(\boldsymbol{G},\mathscr{T})} J &= \sum_{\tilde{\boldsymbol{\sigma}}'_i \in \boldsymbol{V}_{\boldsymbol{\theta}\to\boldsymbol{\sigma}}} \left(\frac{\partial \tilde{\boldsymbol{\sigma}}'_i}{\partial \tilde{\boldsymbol{\theta}}_k}\right)^T \mathscr{T}_i \nabla_{\tilde{\boldsymbol{\sigma}}'_i} J \\ &= \sum_{\tilde{\boldsymbol{\sigma}}'_i \in \boldsymbol{V}_{\boldsymbol{\theta}\to\boldsymbol{\sigma}}} \sum_j \delta(\tilde{\boldsymbol{\theta}}_{i,j}, \tilde{\boldsymbol{\theta}}_k) \boldsymbol{g}_{\tilde{\boldsymbol{\theta}}_{i,j}\to\tilde{\boldsymbol{\sigma}}'_i}(\mathscr{T}_{i,j}\nabla_{\tilde{\boldsymbol{\sigma}}'_i} J; \tilde{\boldsymbol{\theta}}_{i,1}, \cdots, \tilde{\boldsymbol{\theta}}_{i,N_1}; \tilde{\boldsymbol{\sigma}}'_{i,1}, \cdots, \tilde{\boldsymbol{\sigma}}'_{i,N'_i}). \end{aligned} \tag{10}$$

Obviously, $\nabla_{\tilde{\boldsymbol{\theta}}_k} J = \nabla_{\tilde{\boldsymbol{\theta}}_k}^{(\boldsymbol{G},I)} J$ where I is an identity operator. The **bprop** operation $\boldsymbol{g}_{\boldsymbol{w}\to\tilde{\boldsymbol{\sigma}}'_i}$ is uniquely determined by $\tilde{\boldsymbol{f}}_i$.

Then, the Eqn.(6) can be written as the following virtual gradient descent iteration:

$$\tilde{\boldsymbol{\theta}}_k^{(t+1)} = \tilde{\boldsymbol{\theta}}_k^{(t)} - \alpha \nabla_{\tilde{\boldsymbol{\theta}}_k}^{(\boldsymbol{G},\mathscr{T})} J\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}, \quad \forall \tilde{\boldsymbol{\theta}}_k \in \boldsymbol{V}_{\boldsymbol{\theta}}^{\boldsymbol{G}} \tag{11}$$

We prove that the SVGD (Alg. 1) has advantages over SGD, RMSProp and Adam in training speed and test accuracy by experiments on multiple network models and datasets.

In Sec.2 we describe the operator $\mathscr{T}$ and the SVGD algorithm of stochastic optimization. Sec.3 introduce two methods to encapsulate SVGD, and Sec.4 provides a theoretical analysis of convergence. Sec.6 compares our method with other methods by experiments.

2 Stochastic Virtual Gradients Descent Method

In this section, we will use the **accumulate squared gradient** in the RMSProp to construct the operator $\mathscr{T}$. According to Eqn.(7), Eqn.(3) holds when the mapping $\boldsymbol{f}$ is linear. Based on this fact, we designed the following SVGD algorithm. The functions and variables in the algorithm are given by Eqn.(9) and Eqn.(10).

Algorithm 1: *SVGD*, our proposed algorithm for stochastic optimization. $(\nabla J)^2$ indicates the elementwise square $\nabla J \odot \nabla J$. Good default settings for the tested machine learning problems are $\epsilon = 10^{-6}$ and $\rho = 0.9$. All operations are element-wise.

```
Require: $\boldsymbol{G}$: computational graph associated with function $J^{(\tau)} = L(\hat{y}(\boldsymbol{x}^{(\tau)}, \boldsymbol{\theta}), y^{(\tau)})$
Require: $\alpha$: Learning rate
Require: $m$: Minibatch size
Require: $s \in [0, +\infty)$: Scaling coefficient
Require: $\boldsymbol{\theta}$: Initial parameter
/* define operator $\nabla^{(\boldsymbol{G},\mathscr{T})}$ before training */
for $\tilde{\boldsymbol{\sigma}}'_i \in \boldsymbol{V}^{\boldsymbol{G}}_{\boldsymbol{\theta}\to\boldsymbol{\sigma}}$ do
    $\boldsymbol{r}_i = \boldsymbol{0}$ // Initialize gradient accumulation variable
    for $j \in \{1, \cdots, N_i\}$ do
        if $\tilde{\boldsymbol{f}}_i$ about $\tilde{\boldsymbol{\theta}}'_{i,j}$ is linear then
            $\boldsymbol{g}_{\tilde{\boldsymbol{\theta}}'_{i,j}\to\tilde{\boldsymbol{\sigma}}'_i}(\mathscr{T}_{i,j}\nabla_{\tilde{\boldsymbol{\sigma}}'_i} J^{(\tau)}; \cdots; \cdots) = \boldsymbol{g}_{\tilde{\boldsymbol{\theta}}'_{i,j}\to\tilde{\boldsymbol{\sigma}}'_i}(s\frac{\nabla_{\tilde{\boldsymbol{\sigma}}'_i} J^{(\tau)}}{\sqrt{\boldsymbol{r}_i + \epsilon}}; \cdots; \cdots)$ // define $\mathscr{T}_{i,j}$
        else
            $\boldsymbol{g}_{\tilde{\boldsymbol{\theta}}'_{i,j}\to\tilde{\boldsymbol{\sigma}}'_i}(\mathscr{T}_{i,j}\nabla_{\tilde{\boldsymbol{\sigma}}'_i} J^{(\tau)}; \cdots; \cdots) = \boldsymbol{g}_{\tilde{\boldsymbol{\theta}}'_{i,j}\to\tilde{\boldsymbol{\sigma}}'_i}(\nabla_{\tilde{\boldsymbol{\sigma}}'_i} J^{(\tau)}; \cdots; \cdots)$ // define $\mathscr{T}_{i,j}$
        end
    end
end
/* update $\boldsymbol{\theta}$ */
while $\boldsymbol{V}^{\boldsymbol{G}}_{\boldsymbol{\theta}^{(t)}}$ not converged do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $y^{(i)}$.
    for $\tilde{\boldsymbol{\sigma}}'_i \in \boldsymbol{V}^{\boldsymbol{G}}_{\boldsymbol{\theta}\to\boldsymbol{\sigma}}$ parallel do
        $\boldsymbol{r}_i \leftarrow \rho\boldsymbol{r}_i + (1-\rho)\frac{1}{m}\sum_{i=\tau}^{m}\left(\nabla_{\tilde{\boldsymbol{\sigma}}'_i} J^{(\tau)}\right)^2$ // Accumulate squared gradient
    end
    for $\tilde{\boldsymbol{\theta}}_j \in \boldsymbol{V}^{\boldsymbol{G}}_{\boldsymbol{\theta}}$ parallel do
        $\tilde{\boldsymbol{\theta}}_j \leftarrow \tilde{\boldsymbol{\theta}}_j - \alpha\frac{1}{m}\sum_{i=\tau}^{m}\nabla^{(\boldsymbol{G},\mathscr{T})}_{\tilde{\boldsymbol{\theta}}_j} J^{(\tau)}$ // apply update
    end
end
```

SVGD works well in neural network training tasks (Fig.9, 11, 12), it has a relatively faster convergence rate and better test accuracy than SGD, RMSProp, and Adam.

For the linear operation Conv2D [13] and matrix multiplication MatMul as follows:

$$\begin{cases} \text{Conv2D} : (\mathbb{R}^N \times \mathbb{R}^{H_{in}} \times \mathbb{R}^{W_{in}} \times \mathbb{R}^{C_{in}}, \mathbb{R}^{H_k} \times \mathbb{R}^{W_k} \times \mathbb{R}^{C_{in}} \times \mathbb{R}^{C_{out}}) \to \mathbb{R}^N \times \mathbb{R}^{H_{out}} \times \mathbb{R}^{W_{out}} \times \mathbb{R}^{C_{out}} \\ \text{MatMul} : (\mathbb{R}^N \times \mathbb{R}^{C_{in}}, \mathbb{R}^{C_{in}} \times \mathbb{R}^{C_{out}}) \to \mathbb{R}^N \times \mathbb{R}^{C_{out}} \end{cases},$$

there are $\text{Dim}(\boldsymbol{r}_{\text{Conv2D}}) = H_{out}W_{out}C_{out} < H_kW_kC_{in}C_{out}$ and $\text{Dim}(\boldsymbol{r}_{\text{MatMul}}) = C_{out} < C_{in}C_{out}$. Thus, SVGD also has less memory requirements than RMSProp and Adam for deep neural networks.

For the same stochastic objective function, the learning rate at timestep t in SVGD has the following relationship with the stepsize in the SGD and RMSProp:

$$\alpha_{SVGD}(t) \approx \alpha_{SGD}(t), \quad s * \alpha_{SVGD}(t) \approx \alpha_{RMSProp}(t).$$

3 Encapsulation

In this section, we introduce two methods to generate the computational graph of virtual gradient. We begin by assumming that the objective function is J (cf. Fig. 4 (b)), the set $\boldsymbol{V}^{\boldsymbol{G}}_{\boldsymbol{\theta}\to\boldsymbol{\sigma}} = \{\tilde{\boldsymbol{\sigma}}'_1, \tilde{\boldsymbol{\sigma}}'_2\}$ (cf. Fig. 4 (a)), and the function used to construct the computational graph of gradients is "gradients", cf. Fig. 4 (c).

$$\begin{array}{ccc|c|c}
\tilde{\boldsymbol{f}}_i & \sim & \boldsymbol{g}_{\boldsymbol{w}\to\tilde{\boldsymbol{\sigma}}'_i}(\nabla_{\tilde{\boldsymbol{\sigma}}'_i}J;\cdots;\cdots) & J = F(\tilde{\boldsymbol{f}}_2 \circ \tilde{\boldsymbol{f}}_1, \tilde{\boldsymbol{f}}_1) & \nabla_{\boldsymbol{w}}J = \text{gradients}(J, \boldsymbol{w}) \\
\| & & \Downarrow & \Downarrow & \Downarrow \\
\hat{\boldsymbol{f}}_i & \sim & \boldsymbol{g}_{\boldsymbol{w}\to\tilde{\boldsymbol{\sigma}}'_i}(\mathscr{T}\nabla_{\tilde{\boldsymbol{\sigma}}'_i}J;\cdots;\cdots) & \hat{J} = F(\hat{\boldsymbol{f}}_2 \circ \hat{\boldsymbol{f}}_1, \hat{\boldsymbol{f}}_1) & \nabla^{(\boldsymbol{G},\mathscr{T})}_{\boldsymbol{w}}J = \text{gradients}(\hat{J}, \boldsymbol{w}) \\
 & & \text{(a)} & \text{(b)} & \text{(c)}
\end{array}$$

Figure 4

We hope to generate the computational graph of virtual gradients by using the function "gradients", Fig. 4 (c).

3.1 Extend the API libraries

As shown in Fig. 4, We begin by replacing $\tilde{\boldsymbol{f}}_i$ with $\hat{\boldsymbol{f}}_i$, where $\hat{\boldsymbol{f}}_i$ is a copy of $\tilde{\boldsymbol{f}}_i$ but corresponds to a new **bprob** operation. Then, call the function "gradients" to generate the computational graph of virtual gradients.

In order to achieve the idea above, in programming, we need to extending core libraries to customize new operations of $\hat{\boldsymbol{f}}_i$ and its **bprop** operation. Fig. 5 shows that we need to extend 3 libraries in the layered architecture of TensorFlow [14].

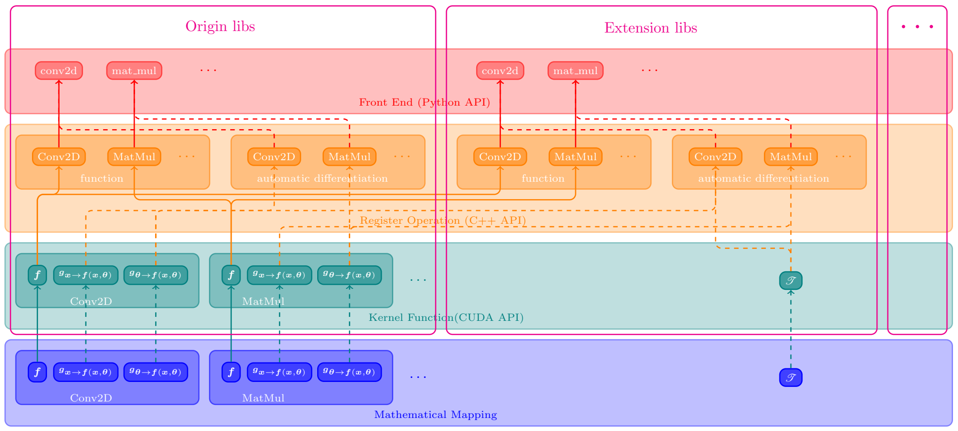


Figure 5: Layered architecture of Tensorflow

3.2 Modify the topology of the calculation graph

According to Eqn.(10) and Fig. 3, the computational graph of the virtual gradients can be obtained by adding new nodes on the computational graph of the gradients and reroute the inputs and outputs of new nodes. cf. Fig. 6.

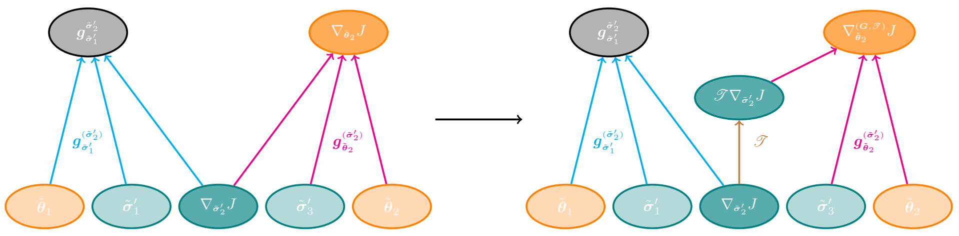


Figure 6: Subgraph views of gradients and virtual gradients. **Left:** the part of the computational graph of the gradients. **Right:** the part of the computational graph of the virtual gradients.

4 Convergence Analysis

In this section, we will analyze the theoretical convergence of Eqn.(6) under some assumptions.

Lemma 4.1. *Let $\boldsymbol{M}$ be a random $(m \times n)$-matrix, $\boldsymbol{M}_{11}, \cdots, \boldsymbol{M}_{ij}, \cdots, \boldsymbol{M}_{mn}$ be an i.i.d. variable from $U(-\infty, +\infty)$. Then*

$$f_\lambda(\boldsymbol{v}, \boldsymbol{u}) := E\left[\boldsymbol{v}^T \boldsymbol{M}^T \boldsymbol{M} \boldsymbol{u} \middle| \, ||\boldsymbol{M}||_2 = \lambda\right] \propto \boldsymbol{v}^T \boldsymbol{u}, \quad \forall \lambda > 0, \forall \boldsymbol{v}, \boldsymbol{u} \in \mathbb{R}^n. \tag{12}$$

Proof. Let $\boldsymbol{e}_i$ be the unit vector whose i-th component is 1, f_λ is bilinear, Then

$$f_\lambda(\boldsymbol{v}, \boldsymbol{u}) = \sum_{i=1}^{n} \sum_{j=1}^{n} \boldsymbol{v}_i \boldsymbol{u}_j f_\lambda(\boldsymbol{e}_i, \boldsymbol{e}_j) = \boldsymbol{v}^T \boldsymbol{C} \boldsymbol{u}, \quad \boldsymbol{C}_{ij} := E\left[\boldsymbol{e}_i^T \boldsymbol{M}^T \boldsymbol{M} \boldsymbol{e}_j \middle| \, ||\boldsymbol{M}||_2 = \lambda\right]. \tag{13}$$

Since $\boldsymbol{M}_{11}, \cdots, \boldsymbol{M}_{ij}, \cdots, \boldsymbol{M}_{mn}$ be an i.i.d. variable from $U(-\infty, +\infty)$, the following holds true:

$$\begin{cases} E[\boldsymbol{e}_1^T \boldsymbol{M}^T \boldsymbol{M} \boldsymbol{e}_1 | \, ||\boldsymbol{M}||_2 = \lambda] = \cdots = E[\boldsymbol{e}_n^T \boldsymbol{M}^T \boldsymbol{M} \boldsymbol{e}_n | \, ||\boldsymbol{M}||_2 = \lambda] := c_0 > 0 \\ E[\boldsymbol{e}_i^T \boldsymbol{M}^T \boldsymbol{M} \boldsymbol{e}_j | \, ||\boldsymbol{M}||_2 = \lambda] = 0, \quad i, j \in \{1, \cdots, n\} \end{cases}.$$

Thus:

$$f_\lambda(\boldsymbol{v}, \boldsymbol{u}) = c_0 \boldsymbol{v}^T \boldsymbol{u}. \tag{14}$$

□

Fig. 7 proof our lemma.

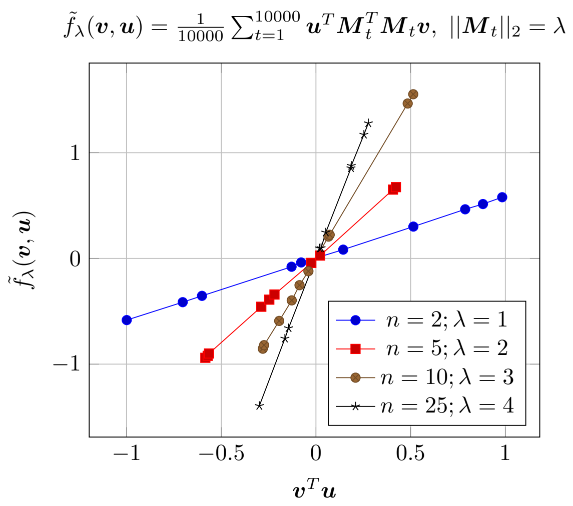


Figure 7: The relationship between $\boldsymbol{v}^T \boldsymbol{u}$ and the estimate of $f_\lambda(\boldsymbol{v}, \boldsymbol{u})$. Each point corresponds to a pair of random vector $(\boldsymbol{v}, \boldsymbol{u})$ and a random matrix set $\{\boldsymbol{M}_t | t = 1, \cdots, 10000\}$.

Corollary 4.1.1. *For $\boldsymbol{M}$ defined in Lemma 4.1, if $\boldsymbol{v}^T\boldsymbol{u} > 0$, then:*

$$E\left[\boldsymbol{v}^T\boldsymbol{M}^T\boldsymbol{M}\boldsymbol{u}\right] > 0. \tag{15}$$

Theorem 4.2. *Let F and $\boldsymbol{f}$ be second-order differentiable functions with random variables in their expression, we set:*

$$\boldsymbol{v}^T\mathscr{T}\boldsymbol{v} > 0, \quad i \in 1, \cdots, m, \ \forall \boldsymbol{v} \in \mathbb{R}^m \backslash \{\boldsymbol{0}\}.$$

If each component of Jacobian matrix $\boldsymbol{M} = \left(\frac{\partial \boldsymbol{f}_i(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_j}\right)_{m\times n}$ is an i.i.d. variable from $U(-\infty, +\infty)$, then, for $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha * \boldsymbol{M}\mathscr{T}\nabla_{\boldsymbol{f}(\boldsymbol{\theta})}F(\boldsymbol{f}(\boldsymbol{\theta}))|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}$ and $\nabla_{\boldsymbol{\theta}}F(\boldsymbol{f}(\boldsymbol{\theta}))|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}} \neq \boldsymbol{0}$ there exists a $\alpha > 0$ such that

$$E\left[F(\boldsymbol{f}(\boldsymbol{\theta}^{(t+1)})) - F(\boldsymbol{f}(\boldsymbol{\theta}^{(t)}))\right] < 0,$$

Proof. Without loss of generality, we can assume $\boldsymbol{\theta} \in \mathbb{R}^n, \boldsymbol{\sigma} = \boldsymbol{f}(\boldsymbol{\theta}) \in \mathbb{R}^m, m > n$. Then, the Maclaurin series for $F(\boldsymbol{f}(\boldsymbol{\theta})))$ around the point $\boldsymbol{\theta}^{(t)}$ is:

$$\begin{aligned}F(\boldsymbol{f}(\boldsymbol{\theta}^{(t+1)})) - F(\boldsymbol{f}(\boldsymbol{\theta}^{(t)})) &= -\alpha(\nabla_{\boldsymbol{\theta}}J|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}})^T\boldsymbol{M}\mathscr{T}\nabla_{\boldsymbol{\sigma}}J|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}} + o(\alpha)\\ &= -\alpha(\nabla_{\boldsymbol{\sigma}}J|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}})^T\boldsymbol{M}^T\boldsymbol{M}\mathscr{T}\nabla_{\boldsymbol{\sigma}}J|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}} + o(\alpha).\end{aligned}$$

Let $\boldsymbol{v} = \nabla_{\boldsymbol{\sigma}}J|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}$. According to corollary 4.1.1:

$$E\left[F(\boldsymbol{f}(\boldsymbol{\theta}^{(t+1)})) - F(\boldsymbol{f}(\boldsymbol{\theta}^{(t)}))\right] = -\alpha E\left[\boldsymbol{v}^T\boldsymbol{M}^T\boldsymbol{M}\mathscr{T}\boldsymbol{v}\right] + o(\alpha) < 0.$$

□

Although our convergence analysis in Thm.4.2 only applies to the assumption of uniform distribution, we empirically found that SVGD often outperforms other methods in general cases.

5 Related Work

First-order methods. For general first-order methods, The moving direction $\boldsymbol{p}^{(t)}$ of the variables can be regarded as the function of the stochastic gradient $\boldsymbol{g}^{(t)}$:

- **SGD:** $\boldsymbol{p}^{(t)} = \mathscr{T}\boldsymbol{g}^{(t)} := -\boldsymbol{g}^{(t)}$.
- **Momentum:**[15] Let $\boldsymbol{m}^{(0)} = \boldsymbol{0}, \boldsymbol{m}^{(t+1)} = c\,\boldsymbol{m}^{(t)} + \boldsymbol{g}^{(t)}$. Then:
$$\boldsymbol{p}^{(t)} = \mathscr{T}\boldsymbol{g}^{(t)} := -\boldsymbol{m}^{(t+1)}.$$
- **RMSProp:** Let $\boldsymbol{r}^{(0)} = \boldsymbol{0}, \boldsymbol{r}^{(t+1)} = \rho\,\boldsymbol{r}^{(t)} + (1-\rho)\,\boldsymbol{g}^{(t)} \odot \boldsymbol{g}^{(t)}$. Then:
$$\boldsymbol{p}^{(t)} = \mathscr{T}\boldsymbol{g}^{(t)} := -\frac{\boldsymbol{g}^{(t)}}{\sqrt{\delta + \boldsymbol{r}^{(t+1)}}}.$$
- **Adam:** Let $\boldsymbol{s}^{(0)} = \boldsymbol{r}^{(0)} = \boldsymbol{0}, \boldsymbol{s}^{(t+1)} = \rho_1\,\boldsymbol{s}^{(t)} + (1-\rho_1)\,\boldsymbol{g}^{(t)}, \boldsymbol{r}^{(t+1)} = \rho_2\,\boldsymbol{r}^{(t)} + (1-\rho_2)\,\boldsymbol{g}^{(t)} \odot \boldsymbol{g}^{(t)}$. Then:
$$\boldsymbol{p}^{(t)} = \mathscr{T}\boldsymbol{g}^{(t)} := -\frac{\boldsymbol{s}^{(t+1)}/(1-\rho_1^t)}{\sqrt{\delta + \boldsymbol{r}^{(t+1)}/(1-\rho_2^t)}}.$$

However, in SVGD method, $\boldsymbol{p}^{(t)} = -\nabla_{\boldsymbol{\theta}}^{(\boldsymbol{G},\mathscr{T})}J$ cannot be written as a function of $\boldsymbol{g}^{(t)}$. Thus, SVGD is not essentially a first-order method.

Global minimum. A central challenge of non-convex optimization is avoiding sub-optimal local minima. Although it has been shown that the variable can sometimes converges to a neighborhood of the global minimum by adding noise[16, 17, 18, 19, 20], the convergence rate is still a problem. Note that the DP method has some probability to escape "appropriately shallow" local minima because the moving direction of the variable is generated by solving several sub-problems instead of the original problem. We use computational graph and automatic differentiation to generate the sub-problems in DP, such as what we did in the SVGD method.

6 Experiments

In this section, we evaluated our method on two benchmark datasets using several different neural network architectures. We train the neural networks using RMSProp, Adam, SGD, and SVGD to minimize the cross-entropy objective function with L_1 weight decay on the parameters to prevent over-fitting. To be fair, for different methods, a given objective function will be minimized with different learning rates. All extension libs, algorithm, and experimental logs in this paper can be found at the URL: `https://github.com/LizhengMathAi/svgd`.

The following experiments show that SVGD has a relatively faster convergence rate and better test accuracy than SGD, RMSProp, and Adam.

6.1 Multi-layer neural network

In our first set of experiments, we train a 5-layer neural network (Fig. 8) on the MNIST [21] handwritten digit classification task.

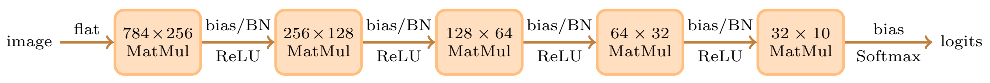


Figure 8: MLP architecture for MNIST with 5 parameter layers (245482 params).

The model is trained with a mini-batch size of 32 and weight decay of 1.0×10^{-4}. In Table 1, we decay α at 1.6k and 3.6k iterations and summarize the optimal learning rates for RMSProp, Adam, SGD, and SVGD by hundreds of experiments.

		RMSProp	**Adam**	**SGD**	**SVGD**(s=0.1)
α	**iter:** $[0 \sim 1599]$	0.001	0.001	0.1	0.01
	iter: $[1600 \sim 3599]$	0.0005	0.00005	0.05	0.005
	iter: $[1600 \sim 5999]$	0.00005	0.00005	0.01	0.001
test top-1 error		1.80%	1.94%	1.76%	1.60%

Table 1: The test error and learning rates in **MLP** experiments.

In Table 1 and Fig. 9 we compare the error rates and their descent process process on the CIFAR-10 test set, respectively.

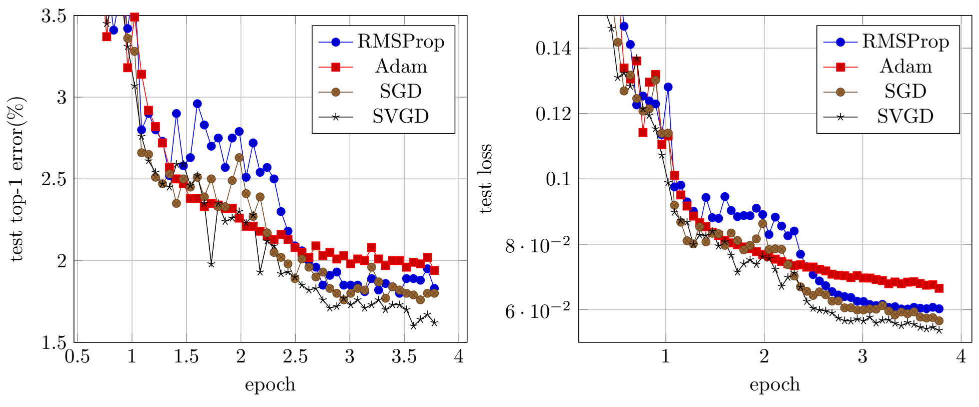


Figure 9: Comparison of first-order methods on **MNIST** digit classification for 3.75 epochs.

6.2 Convolutional neural network

We train a VGG model (Fig. 10) on the CIFAR-10 [22] classification task and follow the simple data augmentation in [23, 24] for training and evaluate the original image for testing.

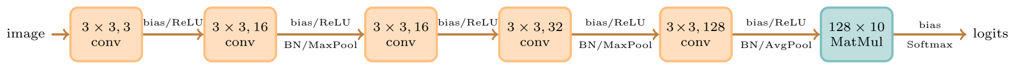


Figure 10: VGG model architecture for **CIFAR-10** with 6 parameter layers (46126 params).

The model is trained with a mini-batch size of 128 and weight decay of 1.0×10^{-5}. In Table 2, we decay α at 12k and 24k iterations and summarize the optimal learning rates for RMSProp, Adam, SGD, and SVGD by hundreds of experiments.

		RMSProp	**Adam**	**SGD**	**SVGD**(s=0.001)
α	**iter:** $[0 \sim 11999]$	0.02	0.02	2.0	2.0
	iter: $[12000 \sim 23999]$	0.01	0.01	0.5	0.5
	iter: $[24000 \sim 34999]$	0.002	0.005	0.005	0.005
test top-1 error		17.78%	18.02%	17.32%	17.07%

Table 2: The test error and learning rates in **VGG** experiment.

In Table 2 and Fig. 11 we compare the error rates and their descent process on the CIFAR-10 test set, respectively.

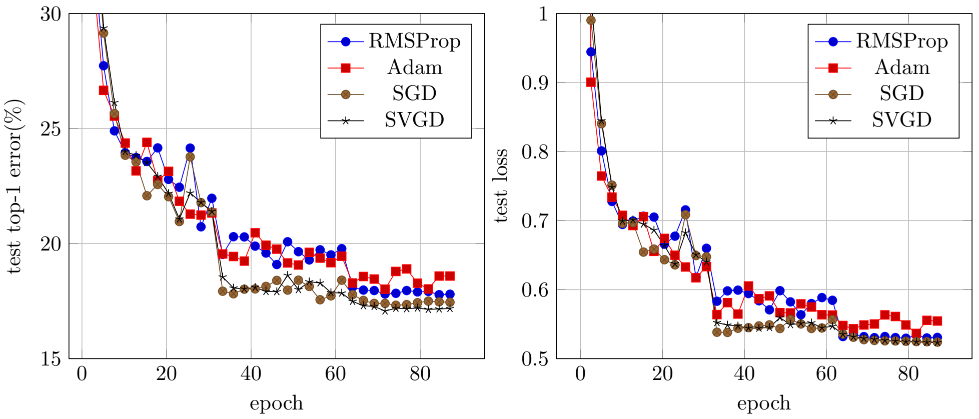


Figure 11: Comparison of first-order methods on **CIFAR-10** dataset for 90 epochs.

6.3 Deep neural network

We use the same hyperparameters with [24] to train ResNet-20 model(0.27M params) on the CIFAR-10 classification task. In Table 3, we decay α at 12k and 24k iterations and summarize the optimal learning rates for RMSProp, Adam, SGD, and SVGD by hundreds of experiments.

		RMSProp	**Adam**	**SGD**	**SVGD**(s=0.01)
α	**iter:** $[0 \sim 31999]$	0.001	0.001	0.1	0.5
	iter: $[32000 \sim 41999]$	0.0001	0.0001	0.01	0.02
	iter: $[42000 \sim 49999]$	0.0001	0.00005	0.001	0.01
test top-1 error		11.18%	11.12%	10.69%	8.62%

Table 3: The test error and learning rates in **ResNet** experiments.

In Table 3 and Fig. 12 we compare the error rates and their descent process on the CIFAR-10 test set, respectively. The top-1 error fluctuations in experiments do not exceed 1%. See [25] for more information on the CIFAR-10 experimental record.

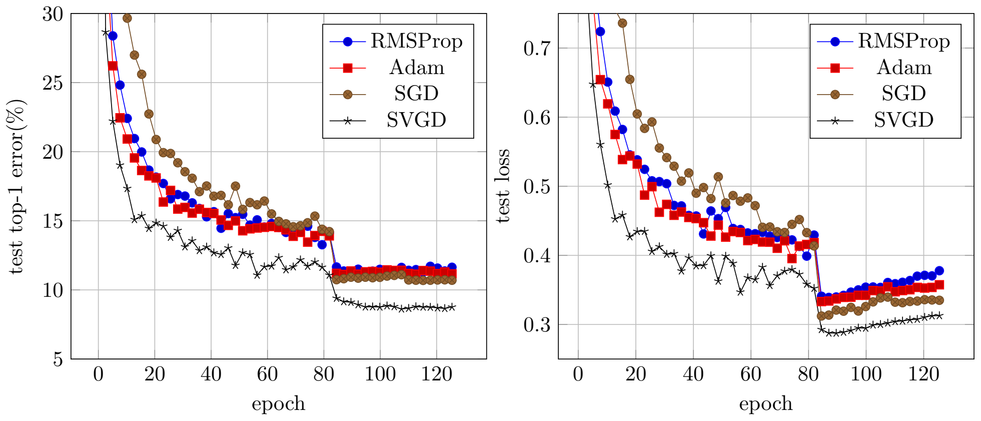


Figure 12: Comparison of first-order methods on **CIFAR-10** dataset for 125 epochs.

References

[1] David Saad. Online algorithms and stochastic approximations. *Online Learning*, 5, 1998.

[2] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14, 2012.

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

[4] 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*, pages 4148–4158, 2017.

[5] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[7] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[8] Yuchen Zhang, Percy Liang, and Moses Charikar. A hitting time analysis of stochastic gradient langevin dynamics. *arXiv preprint arXiv:1702.05575*, 2017.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[10] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[11] Krishnaiyan Thulasiraman and Madisetti NS Swamy. *Graphs: theory and algorithms*. Wiley Online Library, 1992.

[12] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, 18:1–43, 2018.

[13] Yann LeCun et al. Generalization and network design strategies. In *Connectionism in perspective*, volume 19. Citeseer, 1989.

[14] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

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

[16] Arvind Neelakantan, Quoc V Le, and Ilya Sutskever. Neural programmer: Inducing latent programs with gradient descent. *arXiv preprint arXiv:1511.04834*, 2015.

[17] Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

[18] Karol Kurach, Marcin Andrychowicz, and Ilya Sutskever. Neural random-access machines. *arXiv preprint arXiv:1511.06392*, 2015.

[19] Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

[20] Albert Zeyer, Patrick Doetsch, Paul Voigtlaender, Ralf Schlüter, and Hermann Ney. A comprehensive study of deep bidirectional lstm rnns for acoustic modeling in speech recognition. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2462–2466. IEEE, 2017.

[21] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[22] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[23] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Artificial intelligence and statistics*, pages 562–570, 2015.

[24] 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*, pages 770–778, 2016.

[25] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.