SPAN: A Stochastic Projected Approximate Newton Method

Xunpeng Huang,[1] **Xianfeng Liang,**[2] **Zhengyang Liu,**[1,*] **Lei Li,**[1] **Yue Yu,**[3,†] **Yintan Li**[1]
[1]Bytedance AI Lab, [2]University of Science and Technology China, [3]Tsinghua University
{huangxunpeng, liuzhengyang.lozycs, lileilab, liyitan}@bytedance.com,
zeroxf@mail.ustc.edu.cn, yu-y14@mails.tsinghua.edu.cn

Abstract

Second-order optimization methods have desirable convergence properties. However, the exact Newton method requires expensive computation for the Hessian and its inverse. In this paper, we propose SPAN, a novel approximate and fast Newton method. SPAN computes the inverse of the Hessian matrix via low-rank approximation and stochastic Hessian-vector products. Our experiments on multiple benchmark datasets demonstrate that SPAN outperforms existing first-order and second-order optimization methods in terms of the convergence wall-clock time. Furthermore, we provide a theoretical analysis of the per-iteration complexity, the approximation error, and the convergence rate. Both the theoretical analysis and experimental results show that our proposed method achieves a better trade-off between the convergence rate and the per-iteration efficiency.

1 Introduction

Mathematical optimization plays an important role in machine learning. Many learning tasks can be formulated as a problem of minimizing a finite sum objective:

$$\min_{x\in\mathbb{R}^d} F(x) \triangleq \frac{1}{N}\sum_{i=1}^{N} f_i(x), \tag{1}$$

where N is the number of samples, d is the dimension of parameters and $f_i(x)$ denotes the loss function for sample i. In order to solve Eq.(1), many first- and second-order methods have been proposed and has the following update paradigm:

$$x_{t+1} = x_t - \eta_t H^{-1}(x_t)g(x_t), t = 0, 1, 2, \ldots, \tag{2}$$

where $g(x_t)$ is the gradient and η_t is the step size at t-th iteration. The term $H^{-1}(x_t)$ can be set differently in different methods. First-order methods set $H^{-1}(x_t)$ as an identity matrix. The resulting updating procedure becomes gradient descent (GD) or stochastic gradient descent (SGD) depending on whether the gradient $g(x_t)$ is calculated over the whole sample set or one random sample (Robbins and Monro 1985; Li et al. 2014; Cotter et al. 2011). A series of first-order linearly convergent methods and their variance reduction variants were proposed to accelerate the above iteration updates, including SVRG (Johnson and Zhang 2013), SAGA (Defazio, Bach, and Lacoste-Julien 2014), SDCA (Shalev-Shwartz and Zhang 2014), etc. Their main intuition is to balance the computational cost of $g(x_t)$ and the approximation gap between the stochastic gradient $g(x_t)$ and the global gradient $\nabla F(x_t)$.

Compared with first-order optimizers, second-order optimization methods regard $H^{-1}(x_t)$ in Eq. (2) as the Hessian inverse (the standard Newton Method) or certain carefully constructed Hessian inverse (quasi-Newton methods). The matrix $H^{-1}(x_t)$ can be thought to adjust the step size and gradient coordinates through the high order information or the quasi-Newton condition. Second-order methods usually achieve a better convergence rate than first-order ones.

However, second-order algorithms are not widely used in large-scale optimization problems due to expensive computation cost. Computing the Hessian matrix and its inverse requires high computation complexity and consumes large memory. Standard Newton method takes $\mathcal{O}(Nd^2 + d^3)$ per iteration, where N and d are the number of samples and the number of unknown parameters, respectively. Quasi-Newton methods like BFGS and L-BFGS (Liu and Nocedal 1989) are faster, but still requires $\mathcal{O}(Nd + d^2)$. However, they do not maintain the local quadratic convergence rate. For problems with large parameters, it is not affordable even if the inverse calculation could fit in memory.

Our goal is to develop an approximate yet efficient Newton method with provable convergence guarantee. We aim to reduce per-iteration computation cost for the Hessian inverse calculation while maintaining the approximation accuracy. To this end, we propose *Stochastic Projected Approximate Newton* method (SPAN), a novel and generic approach to speed up second-order optimization calculation. The main contributions of the paper are as follows:

- We propose a novel second-order optimization method, SPAN, to achieve a better trade-off between Hessian approximation error and per-iteration efficiency. Inside the method, we propose a stochastic sampling technique to construct the Hessian approximately. Since it only requires first-order oracles and Hessian-vector products, the Hes-

*Corresponding author
†Her work was done as a research intern in Bytedance AI Lab.

sian and its inverse can be computed very efficiently.

- We present a theoretical analysis about the Hessian approximation error and the convergence rate. SPAN achieves linear-quadratic convergence with a provable Hessian approximation error bound.
- We conduct experiments on multiple real datasets against existing state-of-the-art methods. The results validate that our proposed method achieves state-of-the-art performance in terms of the wall-clock time, with almost no sacrificing on the construction accuracy of the Hessian.

2 Related Work

In order to improve the per-iteration efficiency, several stochastic second-order methods have tried to seek the trade-off between per-iteration computational complexity and convergence rate, which can be divided into two main categories.

Stochastic Newton Methods. A series of sub-sampled Newton methods are proposed to solve the problems where the sample size N is far larger than the feature size d. In these methods, the Hessian matrix is approximated with a small subset of training samples. NewSamp (Erdogdu and Montanari 2015) and similar methods by (Roosta-Khorasani and Mahoney 2016; Byrd et al. 2011) sample from function set $\{f_i\}$ randomly and construct a regularized m-rank approximate Hessian with truncated singular value decomposition to improve the per-iteration efficiency. However, they need to compute the sub-sampled Hessian, which has an $\mathcal{O}(d^2)$ computational cost even if there is only one single instance. Removing the requirements of second-order oracle in NewSamp-type methods, our SPAN further accelerates the iteration with stochastic low-rank approximation techniques, which improves the complexity by a factor nearly $\mathcal{O}(d/m)$.

Sketch Newton method in (Pilanci and Wainwright 2017) adopts sketching techniques to approximate Hessian. A non-uniform probability distribution was introduced to sample rows of $\sqrt{\nabla^2 F(x)}$ in (Xu et al. 2016). Both of them use new approximate Hessian construction and proper sampling methods to improve Hessian estimation efficiency. Differ from such sketch Newton methods, our SPAN does not need the *Hessian Decomposition* assumption for sketched updates in (Xu et al. 2016; Pilanci and Wainwright 2017).

Rather than estimating the Hessian, LiSSA by (Agarwal, Bullins, and Hazan 2017) approximates the Hessian inverse with matrix Taylor expansion. The most elegant part of LiSSA is that the Hessian inverse estimation is unbiased, and the approximation error only depends on the matrix concentration inequalities. Compared with LiSSA, SPAN guarantees more robust per-iteration complexity when the objective function is not *Generalized Linear Model* (GLM). Moreover, we do not require the initial solution is close to the optimum. Additionally, the procedure of calculating the approximate Hessian for each sample in SPAN is independent, which means our method has better parallelism compared with LiSSA.

Stochastic Quasi-Newton Methods. The quasi-Newton methods can also be improved with stochastic approximation techniques. S-LBFGS by (Moritz, Nishihara, and Jordan 2016) adopts the randomization to the classical L-BFGS and integrates the widely used gradient variance reduction techniques. Subsequently, SB-BFGS in (Gower, Goldfarb, and Richtárik 2016) extends BFGS with matrix sketching to approximate Hessian inverse. Stochastic quasi-Newton methods have a better per-iteration complexity compared with stochastic Newton methods. However, most of them cannot explicitly demonstrate the benefit of introducing the curvature information in the convergence analysis. Such problem is solved in our SPAN by establishing the Hessian approximation error bound which can hardly be analyzed in stochastic quasi-Newton methods.

Symbols	Description
N	The number of samples in Eq.(1)
d	The dimension of decision variables in Eq.(1)
B	A subset with $B \subseteq [N] = \{1, 2, \dots, N\}$
b	The size of B with $b = \lvert B \rvert$
F	The objective function in Eq.(1)
$f_i(x)$	The loss function for sample i in Eq.(1)
$f_B(x)$	The batch loss with $f_B(x) = \frac{1}{b}\sum_{i\in B} f_i(x)$
$g_B(x)$	The gradient of batch loss with $g_B(x) = \nabla f_B(x)$
$H_B(x)$	The batch Hessian with $H_B(x) = \nabla^2 f_B(x_t)$
$\sigma_i(A)$	The i-th top non-zero singular vector of matrix A
$p_i(A)$	The singular vector of matrix A corresponding to $\sigma_i(A)$
$\mathrm{rank}(A)$	The rank number of matrix A
$\lVert\cdot\rVert$	The Euclidean norm of a vector or L_2 norm of a matrix

Table 1: Important mathematical notations in this paper.

3 The Proposed SPAN Method

In this section, we will first present the overall idea and intuition of our proposed SPAN. Then, we will describe three technical components and the full algorithm. For better illustration, we list some important notations and their descriptions in Table 1.

The goal of SPAN method is to optimize Eq. (1) with respect to the decision variable x (i.e., model parameters) using the mini-batched iterative update:

$$x_{t+1} = x_t - \eta_t H_B^{-1}(x_t) g_B(x_t), \tag{3}$$

where t is the iteration index. Note that the straightforward calculation of Eq. (3) requires $\mathcal{O}(Nd + bd^2 + d^3)$ which is time-consuming for models with a large d.

To make the computation faster, instead of calculating the batch Hessian H_B (the abbreviation of $H_B(x_t)$) and then taking the inverse explicitly, our idea is to replace H_B in Eq. (3) with an approximate Hessian $\hat{H}_B$. Ideally, $\hat{H}_B$ should be bounded within a small region around the true Hessian H_B. We propose to use the projected approximation for the Hessian $\hat{H}_B = PH_BP^T$ where P is a carefully constructed orthogonal projector (Horn and Johnson 2012). To further ensure $\hat{H}_B$ invertible, we add an additional perturbation term ΔH, $\hat{H}_B = PH_BP^T + \Delta H$. Note that such formulation is sufficient to approximate H_B, since we can consider $P^* = \Sigma_{i=1}^k p_i(H_B)p_i(H_B)^T$ to obtain the optimal k-rank approximation of H_B. However, in practice, we can hardly find $p_i(H_B)$ exactly without knowing the batch Hessian. Thus, it is challenging to construct P with desired efficiency and approximation accuracy. At the first glance, this seems infeasible, while our main intuition is based on the observation that affine transformation A over random vectors tends to have larger components on the top singular vectors of A. Such intuition is helpful to find an accurate approximation of orthonormal projector P^*. More details follow later.

In the remaining of this section, we present the construction of an approximate Hessian and the resulting optimization algorithm. In particular, we design the algorithm components guided by the following questions.

1. How to design a structure for P so that $\hat{H}_B = PH_BP^T$ can be computed efficiently without knowing H_B?
2. How to make the inverse robust — permitting $\hat{H}_B$ to be invertible in any circumstance?
3. How to balance the Hessian approximation error $\|\hat{H}_B - H_B\|$ and iteration complexity flexibly?

3.1 Stochastic Projected Approximation

We construct the approximate Hessian as $\hat{H}_B = PH_BP^T$ which can be calculated without knowing H_B. A proper P essentially decides the direction where the Hessian H_B would project onto. We decompose P into the product of the form $P = UU^T$, where $U \in \mathbb{R}^{d\times l}$ is an orthonormal matrix. With that, we construct an approximate Hessian $\hat{H}_B$ by

$$\hat{H}_B(x) = UU^T H_B(x) UU^T. \tag{4}$$

One can easily verify that the Moore-Penrose inverse of $\hat{H}_B$ can be computed efficiently via $\hat{H}_B^\dagger = U\left(U^T H_B U\right)^{-1} U^T$ since the size of $U^T H_B U \in \mathbb{R}^{l\times l}$ is much smaller than H_B and can be calculated by the product of U^T and $H_B U$.

In the following, we develop a method to obtain a U while keeping $\|\hat{H}_B - H_B\|$ small. We derive a method to calculate $H_B U$ without requiring Hessian H_B.

Our method is to randomly choose a set of column vectors Ω and project them to the space expanded by the singular vectors of H_B. From the projection, one can extract an orthonormal basis U to expand a low-dimensional space for H_B to project to. The procedure is inspired by the *Proto-Algorithm* (Halko, Martinsson, and Tropp 2011), combining with the secant equation for Hessian-vector products calculation. We utilize the standard Gaussian distribution to generate such $\Omega \in \mathbb{R}^{d\times l}$ and calculate the projection $Y = H_B\Omega$ directly. To make the computation more efficient, the set of random vectors Ω only contains l elements where $l \ll d$.

Notice that for any matrix V whose i-th column vector is presented as v_i, the extended Hessian-vector product on a sample batch B, denoted as $\Psi_B(x, V)$, is

$$\Psi_B(x,V) \triangleq H_B(x)V = [H_B(x)v_1 \quad H_B(x)v_2 \quad \ldots],$$
$$H_B(x)v_i = \frac{d}{d\alpha} g_B(x+\alpha v_i)\Big|_{\alpha=0}.$$

In practice, the Hessian-vector product $H_B(x)v_i$ can be calculated by the finite difference of gradients like Algorithm 1.

U is then constructed as follows:

Algorithm 1 The Hessian-Vector Product

```
**Input:** $F$, $B$, $\hat{x}$, $v$
Choose a large constant $C \to \infty$
Calculate residual perturbation $\hat{v} = v/C$
Form the gradient difference $\delta = g_B(\hat{x}+\hat{v}) - g_B(\hat{x})$
**Return:** $C\delta$
```

1. calculate the extended Hessian-vector product of Ω to get $Y = H_B(x)\Omega = \Psi_B(x, \Omega)$;
2. calculate the basis U via QR decomposition $Y = UR$.

With the above-constructed U, $\hat{H}_B(x)$ can be further constructed again using the extended Hessian-vector product, $\hat{H}_B(x) = UU^T\Psi_B(x,U)U^T$. But there is no need to calculate $\hat{H}_B$ explicitly, it suffices to calculate its Moore-Penrose inverse. One can verify that with such constructed $\hat{H}_B$, the error in the term of $\|\hat{H}_B(x) - H_B(x)\|$ is well-bounded (Halko, Martinsson, and Tropp 2011).

3.2 Robust Hessian Inversion

The above constructed approximate Hessian is simplified with one flaw ($\hat{H}_B$ is actually not invertible) since the orthonormal matrix U is low-rank. This can be alleviated with a perturbed version of Eq.(4).

$$\hat{H}_B(x) = UU^T H_B(x)UU^T + \lambda\left(I - UU^T\right) \tag{5}$$

where λ is a carefully chosen constant with $\lambda > 0$.

The purpose of the perturbation term $\lambda\left(I - UU^T\right)$ is to introduce the invertibility of the approximate matrix $\hat{H}_B(x)$ presented in Eq. (4). We will give an informal analysis here to show the importance of choosing a proper λ. On one hand, a large λ will impair captures of the main actions of H_B, since it increases the lower bound of the Hessian approximation error $\|\hat{H}_B - H_B\|$ as

$$\begin{aligned}&\left\|UU^T H_B(x)UU^T + \lambda\left(I-UU^T\right) - H_B(x)\right\|\\ \geq &\left\|\lambda\left(I-UU^T\right)\right\| - \left\|UU^T H_B(x)UU^T - H_B(x)\right\|.\end{aligned}$$

On the other hand, from an iteration perspective, we can neither choose a tiny λ since it will lose the benefit of the curvature information induced by approximate Newton. In particular, if we regard the SVDs of $I - UU^T$ and $U^T H_B(x) U$ as follows

$$I - UU^T = U_\perp U_\perp^T, \quad U^T H_B(x)U = \hat{U}\Lambda\hat{U}^T.$$

The SVD of constructed Hessian $\hat{H}_B(x_t)$ in Eq. (5) and its inverse can be formulated as

$$\hat{H}_B(x) = \begin{bmatrix} U\hat{U} & U_\perp\end{bmatrix}\begin{bmatrix}\Lambda & 0\\ 0 & \lambda I\end{bmatrix}\begin{bmatrix}\hat{U}^T U^T\\ U_\perp^T\end{bmatrix}$$
$$\hat{H}_B^{-1}(x) = \begin{bmatrix} U\hat{U} & U_\perp\end{bmatrix}\begin{bmatrix}\Lambda^{-1} & 0\\ 0 & \lambda^{-1} I\end{bmatrix}\begin{bmatrix}\hat{U}^T U^T\\ U_\perp^T\end{bmatrix}.$$

It can be observed that a tiny λ will make the singular vectors associated with λ^{-1} dominate the action of $\hat{H}_B^{-1}(x)$, which

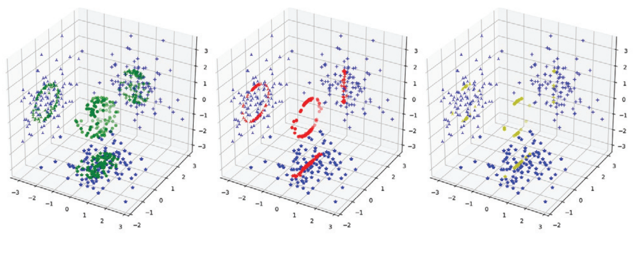

Figure 1: The normalized projection, i.e., $H_B\Omega$, $H_B^4\Omega$ and $H_B^{10}\Omega$ (LTR) of standard gaussian vectors Ω (blue points) where $H_B = \mathrm{diag}\{1,2,3\}$.

impairs the introduction of curvature information taken by $U\hat{U}$ and Σ.

Hence, a proper λ is needed to balance the Hessian approximation error and the l_2 norm of constructed Hessian inverse. Specifically, we will further discuss λ(see Theorem 4.2) and the Hessian approximation error $\|\hat{H}_B(x) - H_B(x)\|$(see Lemma 4.1) in Section 4. Rigorous proof will be deferred to our full version.

3.3 Power Iteration

In this part, we introduce a power iteration technique to balance $\|\hat{H}_B - H_B\|$ and iteration complexity more flexibly. That is to say, with limit iteration complexity sacrificing, the construction of $\hat{H}_B$ in Eq. (5) can be improved through some auxiliary steps in the main algorithm. Generally speaking, any matrix U obtained as in Section 3.1 is valid for $\hat{H}_B$'s construction. However, in practice, a better orthogonal projector UU^T maintains that top singular vectors of $H_B(x_t)$ are rotated less after performing the projection on $H_B(x_t)$, e.g., $UU^T H_B(x_t)UU^T$. The matrix U is usually obtained from the projection $H_B(x_t)\Omega$ of random vectors. Thus, we hope basis vectors of the projection have larger components on top singular vectors e.g., $p_1(H_B), p_2(H_B)$, compared with those on $p_{d-1}(H_B)$ and $p_d(H_B)$. Such a requirement can be satisfied by taking the power of the Hessian.

Specifically, as shown in Figure 1 showed, normalized $H_B\Omega$ (green dots) seems to like a unit ball, while they degenerate to a unit circle (red dots) when random vectors Ω are projected through H_B^4 (red dots) because the component of projection on $[1,0,0]$ almost have become 0. A similar phenomenon also happens for the component of projection on $[0,1,0]$ when taking a larger power for Hessian, i.e., $H_B^{10}\Omega$ (yellow dots). As a result, the normalized projections nearly collapse to two points (yellow dots), i.e., $[0,0,1]$ and $[0,0,-1]$. That is to say, the component of projection on bottom singular vectors tends to vanish. Besides, such a phenomenon becomes more significant as the power of Hessian increases. With such an observation, we can calculate a better U with the following steps:

1. generate a *standard Gaussian matrix* Ω and set $Y_0 = \Omega$;
2. iteratively use the extended Hessian-vector product to get $Y_j = [H_B(x)]^j\Omega = \Psi_B(x, Y_{j-1})$;
3. calculate the basis U via QR decomposition $Y_q = UR$.

Algorithm 2 Stochastic Projected Approximate Newton

```
1:  Input: $F$, $x_0$, $T$, $m$, $l$, $q$, $\eta_t$
2:  for $t = 0$ to $T$ do
3:      Select a uniformed sample batch $B \subseteq [N]$
4:      Generate a standard Gaussian matrix $\Omega \in \mathbb{R}^{d\times l}$, and
        set $Y_0 = \Omega$
5:      for $j = 1$ to $2q+1$ do
6:          $Y_j = \Psi_B(x_t, Y_{j-1})$
7:      end for
8:      Compute the QR decomposition $Y_{2q+1} = UR$
9:      Set $Z = \Psi_B(x_t, U)$ , $\lambda_{\min,t} = \frac{1}{2}\sigma_{\min}(Z^TU)$
        and $\lambda \leq \min\{\sigma_{m+1,t}, \lambda_{\min,t}\}$
10:     Recover the Hessian approximation inverse
        $\hat{H}_B^{-1}(x_t) = U\left(Z^TU\right)^{-1}U^T + \lambda^{-1}\left(I - UU^T\right)$
11:     Calculate $x_{t+1} = x_t - \eta_t \hat{H}_B^{-1}(x_t)\nabla F(x_t)$
12: end for
13: Return: $x_{T+1}$
```

3.4 Details of SPAN

In this section, we show the complete algorithm about SPAN in Algorithm 2. Also, we explain the iteration complexity of SPAN, and compare it with the state-of-the-art optimizers.

In Algorithm 2, we use $\sigma_{i,t}$ as the abbreviation of $\sigma_i(H_B(x_t))$. At each iteration, we first select a sample batch (Step 3) and generate some random matrix (Step 4). With the random matrix Ω, a proper U can be found through the process we introduced in Section 3.3 (Step 5 to Step 8). After that, we compute $H_B(x_t)U$ (Step 9) as an intermediate variable and select the perturbation constant λ. Finally, we calculate the constructed Hessian inverse $\hat{H}_B^{-1}(x_t)$ (Step 10) like Eq. (5) and update the decision variables (Step 11).

Iteration Efficiency. In order to illustrate the computational complexity of each iteration, we first introduce some condition numbers, which are designed with respect to the component functions, i.e., $f_i(\cdot)$ and $f_B(\cdot)$. In such case, one typically assumes that each component is bounded by $\beta_{b,k}(x) \triangleq \max_B \sigma_k(H_B)$ and $\alpha_{b,k}(x) \triangleq \min_B \sigma_k(H_B)$ like LiSSA (Agarwal, Bullins, and Hazan 2017), we define

$$\overline{\kappa}_{b,k} \triangleq \max_x \frac{\beta_{b,k}(x)}{\alpha_{N,d}(x)}, \quad \hat{\kappa}_{b,k} \triangleq \frac{\max_x \beta_{b,k}(x)}{\min_x \alpha_{N,d}(x)},$$
$$\tilde{\kappa}_{b,k} \triangleq \max_x \frac{\beta_{b,k}(x)}{\alpha_{b,d}(x)} \quad \text{and} \quad \dot{\kappa}_{b,k} \triangleq \max_{x,B} \frac{\sigma_k(H_B(x))}{\sigma_d(H_B(x))}.$$

Such condition numbers have the following relations

$$\overline{\kappa}_{b,k} \leq \hat{\kappa}_{b,k} \quad \text{and} \quad \dot{\kappa}_{b,k} \leq \tilde{\kappa}_{b,k}. \tag{6}$$

We compare per-iteration complexity among state-of-the-art optimizers, including SPAN, and list the results in Table 2 where we ignore log terms of d and different κs for brevity. The iteration complexity of SPAN consists of three terms. The first term $\mathcal{O}(Nd)$ represents the time complexity of the full gradient computation (Step 11). The second term indicates the complexity of power iteration (Step 5 to Step 7). The calculation of $\Psi_B(\cdot)$ in Step 6 requires $\mathcal{O}(bld)$

where the fact $q \propto \log d$ and $b = \Theta(\dot{\kappa}_{b,1}^2 \dot{\kappa}_{b,m}^2 \log d)$ will be detailedly demonstrated in our full version. The last term $\mathcal{O}(l^2 d)$ shows the complexity of QR decomposition (Step 8). In particular, the most time-consuming step in constructing $\hat{H}_B^{-1}(x_t)$ is to calculate $(Z^T U)^{-1}$, which leads to an $\mathcal{O}(l^3)$ time complexity. For a given $g(x_t)$, the computational cost of $\hat{H}_B^{-1}(x_t) g(x_t)$ is only $\mathcal{O}(ld)$ when the order of matrix multiplication is appropriately arranged. Owing to the fact that $l \ll d$, such computational cost is much smaller than $\mathcal{O}(l^2 d)$ (the third term).

Comparing with the NewSamp (Erdogdu and Montanari 2015) which updates decision variables with a sub-sampled constructed Hessian inverse, SPAN takes a significant acceleration at each iteration through the only first-order oracle and the Hessian-vector product requirements when the m-th singular value is close to the minimum singular value with $\dot{\kappa}_{b,m}^2 \leq \sqrt{d/m}$. In addition, per-iteration complexity in LiSSA (Agarwal, Bullins, and Hazan 2017) is even worse than NewSamp in general cases. However, in GLM models, LiSSA has better performance because of the fast calculation of Hessian-vector products. Even in GLM models, our SLAN can still be faster than LiSSA when the approximate Hessian is good enough, or the condition number $\overline{\kappa}_{b,1}$ is large with $\dot{\kappa}_{b,m}^2 l \leq \overline{\kappa}_{b,1}$. Notice that, the condition numbers of batch loss chosen, i.e., $\dot{\kappa}_{b,1}^2 \geq d$, will not be too large. Otherwise, per-iteration efficiency is governed by batch Hessian or Hessian-vector products calculation, which impairs the acceleration achieved by matrix approximation techniques.

4 Theoretical Results

We will give a theoretical analysis of our results in this section. We will first introduce some standard assumptions for the optimization problems, and then bound the error between our approximate Hessian and the sub-sampled one. Finally, we will show the local linear-quadratic convergence of our main algorithm (Algorithm 2), with the detailed coefficients of the convergence rate. We further compare the convergence rate of our method with NewSamp (Erdogdu and Montanari 2015) and LiSSA (Agarwal, Bullins, and Hazan 2017). Due to space limitations, the details of proof arguments are provided in the full version.

Assumption 1. *(Hessian Lipschitz Continuity) For any subset $B \subset [N]$ and a second-order differentiable objective function F like the Eq.*(1)*, there exists a constant M_b depending on b, such that $\forall x, \hat{x} \in \mathcal{D}$*

$$\|H_B(x) - H_B(\hat{x})\| \leq M_b \|x - \hat{x}\|.$$

Assumption 2. *(Gradient Lipschitz Continuity and Strong Convexity) For any subset $B \subset [N]$ and a second-order differentiable function F like the Eq.*(1)*, there exists constants μ_b and L_b depending on the b, such that for any $x, \hat{x} \in \mathcal{D}$*

$$\mu_b \|x - \hat{x}\| \leq \|\nabla f_B(x) - \nabla f_B(\hat{x})\| \leq L_b \|x - \hat{x}\|.$$

Assumption 3. *(Bounded Hessian) For any $i \in \{1, 2, ..., N\}$, and the Hessian of the function $f_i(x)$ in Eq.* (1)*, $\nabla^2 f_i(x)$ is upper bounded by an absolute constant K, i.e.,*

$$\max_{i \leq N} \left\|\nabla^2 f_i(x)\right\| \leq K.$$

Lemma 4.1. *Suppose the Assumption 1, 2 and 3 hold. For every iteration t in Algorithm 2, if the parameters satisfy: $m \leq l - 4$, $\lambda \leq \min\left\{\sigma_{m+1,t}, \frac{1}{2}\sigma_{\min}(Z^T U)\right\}$ and*

$$q \geq \left\lceil \frac{1}{2} \log_{\frac{3}{2}} \left(34\sqrt{\frac{l}{l-m}} + \frac{16\sqrt{l(d-m)}}{l-m+1} \right) \right\rceil \propto \log d,$$

with probability at least $1 - 6e^{m-l}$, we have

$$\left\|\hat{H}_B(x_t) - H_B(x_t)\right\| \leq 3\sigma_{m+1,t} \triangleq \epsilon_H.$$

As far as we know, if we approximate the Hessian with an m-rank+sparse construction, the optimal Hessian approximation error will be $\sigma_{m+1,t} - \sigma_{d,t}$ which comes from NewSamp (Erdogdu and Montanari 2015). Corresponding to such an approximation error, the construction complexity of NewSamp requires $\mathcal{O}(\dot{\kappa}_{b,1} d^2 + md^2)$ which can hardly be endured with a slightly larger instance dimension. While, from Lemma 4.1, SPAN improves the complexity by a factor at least $\mathcal{O}(d/(l\dot{\kappa}_{b,m}^2))$ $(l \ll d)$ when the stochastic Hessian can be well approximated and keeps the approximation error nearly three times the optimal ($\sigma_{n,t}$ is a tiny constant). Such a guarantee cannot be promised for any quasi-Newton method. Although LiSSA (Agarwal, Bullins, and Hazan 2017) has a similar error bound for the approximate Hessian inverse, that approximation error is not comparable because it depends almost entirely on the concentration inequalities of the sub-sample processing but not the matrix approximation techniques. Additionally, we only bound the Hessian approximation error when $q = \mathcal{O}(\log d)$ and $\lambda \leq \min\left\{\sigma_{m+1,t}, \frac{1}{2}\sigma_{\min}(Z^T U)\right\}$. In fact, such upper bounds on q and λ are possibly pessimistic and can likely be improved to a more average quantity. However, since the parameters $q = \mathcal{O}(1)$ and $\lambda = \frac{1}{2}\sigma_{m+1}(Z^T U)$ suffice for convergence in our experimental settings, we have not tried to optimize it further.

Theorem 4.2. *Suppose $\lambda_{\min} \leq 2\sigma_{m+1,t}$. Frame the hypotheses of Lemma. 4.1, if the parameters satisfy:*

$$\eta_t \leq \frac{\sigma_{d,t}}{96\lambda_{\min,t} - 16\sigma_{d,t}}$$
$$\text{and } b = \Theta\left(K^2 \sigma_{m+1,t}^2 \sigma_{d,t}^{-4} \log(d)\right),$$

with probability at least $1 - 6e^{m-l}$, we have

$$\|x_{t+1} - x^*\| \leq c_{1,t} \|x_t - x^*\| + c_{2,t} \|x_t - x^*\|^2.$$

The coefficients $c_{1,t}$ and $c_{2,t}$ are

$$c_{1,t} = 1 - \frac{\sigma_{d,t}^2}{36\lambda_{\min,t}^2}\eta_t \quad \text{and} \quad c_{2,t} = \frac{M_b \eta_t}{\lambda_{\min,t}}.$$

Remark. *Notice that, we require $\lambda_{\min,t} \leq 2\sigma_{m+1,t}$ in Theorem 4.2, which is only introduced to simplify the formulation of coefficients, i.e., $c_{1,t}$ and $c_{2,t}$. For any $\lambda_{\min,t} \leq \gamma\sigma_{m+1,t}$ $(\gamma \geq 2)$, we can obtain a similar convergence rate by setting $\lambda = \gamma^{-1}\lambda_{\min,t}$. Thus, the theorem is without loss of generality.*

Algorithm	Per-iteration Complexity
SVRG, SAGA, SDCA	$\mathcal{O}\left(Nd + (\hat{\kappa}_{b,1} d)\right)$
NewSamp(Erdogdu and Montanari 2015)	$\mathcal{O}\left(Nd + \dot{\kappa}_{b,1}^2 d^2 + md^2\right)$
LiSSA(Agarwal, Bullins, and Hazan 2017)	$\mathcal{O}\left(Nd + \tilde{\kappa}_{b,1}^2 \overline{\kappa}_{b,1} d^2\right)$
SPAN (ours)	$\mathcal{O}\left(Nd + \dot{\kappa}_{b,1}^2 \dot{\kappa}_{b,m}^2 dl + l^2 d\right)$

Table 2: Per-iteration complexity comparisons among stochastic first-order optimization methods, sub-sampled Newton Methods, LiSSA and ours. Notice that $\overline{\kappa}_{b,k} \leq \hat{\kappa}$ and $\dot{\kappa}_{b,k} \leq \tilde{\kappa}_{b,k}$.

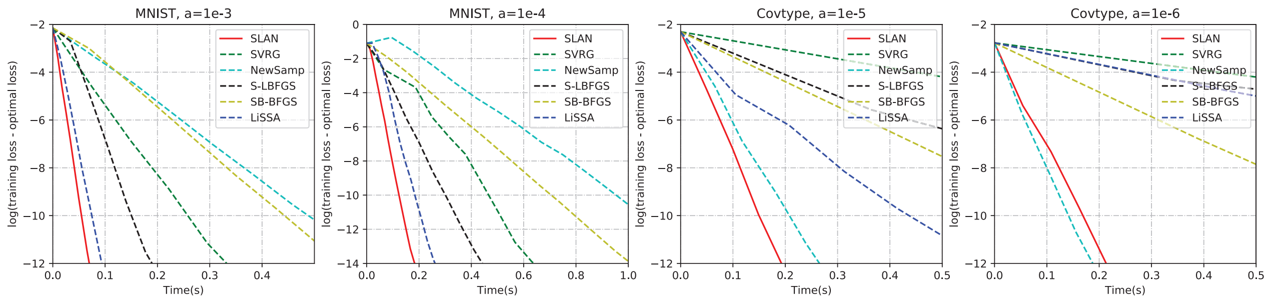

Figure 2: Training loss versus running time. The first two columns are for MNIST4-9 dataset. The last two are for CovType dataset. Note SPAN achieves the best or near the best performance with respect to wall-clock time.

Theorem 4.2 shows that SPAN has a similar composite convergence rate like NewSamp (Erdogdu and Montanari 2015) and LiSSA (Agarwal, Bullins, and Hazan 2017) where c_1^t and c_2^t are coefficients of the linear and quadratic terms, respectively. Comparing the convergence with NewSamp whose first-order coefficient $c_{1,t}$ is abbreviated as $1-\alpha$, ours can be similar to $1-\alpha/6$. In particular, truncated SVD is introduced in NewSamp to find an optimal m-rank approximate Hessian so that c_1^t in NewSamp can be regarded as the optimal coefficient in most stochastic Newton method settings. For LiSSA, its convergence rate is constrained by the initial decision variable x_0 and the sample size for calculating Hessian-vector product strictly, which causes that we cannot compare its convergence rate with ours directly. However, SPAN has a better convergence wall-clock time which is validated in our experimental results (see Section 5). In addition, during the analysis of the convergence rate, we notice that promoting the Hessian approximation error ϵ_H not only reduces the first-order coefficient in the linear-quadratic convergence rate but also expands the range of step size. This coincides with our intuition that a faster convergence usually requires a smaller Hessian approximation error. In summary, SPAN is designed to achieve a better trade-off between the iteration complexity and the convergence rate. With randomizing the process of constructing the special approximate Hessian, SPAN both accelerates the iteration and limits Hessian approximation error.

5 Experiments

To evaluate the performances and fully demonstrate the advantages of our proposed method, we conduct our experiments on several machine learning tasks with different objective functions, including binary image classification and text classification. As we get similar conclusions on these two tasks, we only present the experimental result on the binary image classification in this section. We will further show all experimental details, including the parameters of all optimizers and the results on text classification tasks in the full version due to space limitations.

For the image classification tasks, we refer to the experimental settings in (Agarwal, Bullins, and Hazan 2017) and (Erdogdu and Montanari 2015). We utilize the following objective function:

$$\min_x -\frac{1}{N}\sum_{i=1}^{N} \log \frac{1}{1+\exp\left(-y_i \theta_i^T x\right)} + \frac{1}{2} a \left\| x \right\|^2,$$

where $\theta_i \in R^d$ and $y_i \in \{-1, 1\}$ are the instances and labels, respectively, and a is the L_2 regularization parameter which influences the condition number of the objective.

We choose state-of-the-art first- and second-order optimization methods as baselines, including SVRG (Johnson and Zhang 2013), NewSamp (Erdogdu and Montanari 2015), S-LBFGS (Moritz, Nishihara, and Jordan 2016), SB-BFGS (Gower, Goldfarb, and Richtárik 2016), and LiSSA (Agarwal, Bullins, and Hazan 2017). We implement all the methods in C++ and Intel Math Kernel Library(MKL). All the code for our experiments can be found in our full version.

We adopt two datasets in our experiment, including the MNIST4-9 dataset (Agarwal, Bullins, and Hazan 2017) consisting of approximate 1.2×10^4 instances, and CovType dataset (Erdogdu and Montanari 2015) consisting of approximate 5.0×10^5 instances. For each dataset, we evaluate all the methods on two different condition numbers, respectively.

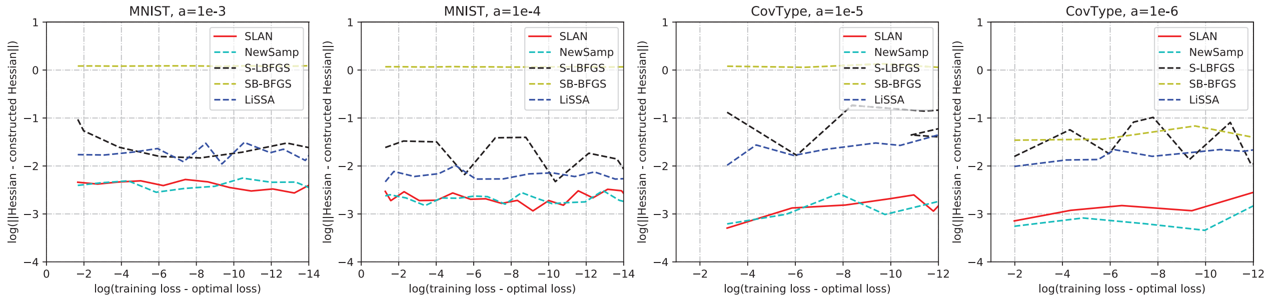

Figure 3: Hessian approximation error. Notice that SPAN achieves near lowest error.

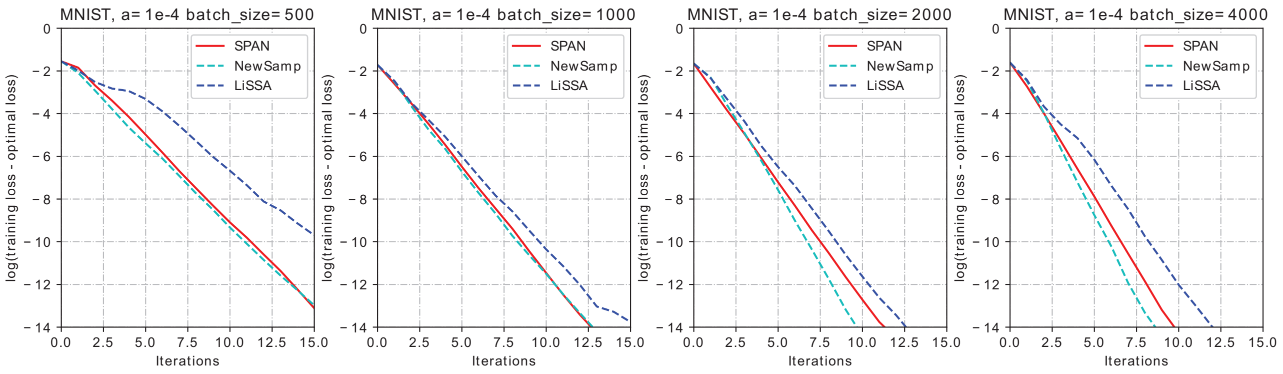

Figure 4: Empirical convergence rate comparison on MNIST4-9 dataset. Four columns represent different batch-size for sub-sampled Hessian.

Besides, for the fairness of comparison, in each set of experiments, we pick the optimal hyperparameters for every method and present the wall-clock time of all the methods in Figure 2.

From Figure 2, we can find that our method SPAN achieves the best results in almost all experimental settings. NewSamp gets a very close performance on the CovType dataset but consumes a lot of time on the MNIST dataset. LiSSA performs very close on the MNIST dataset but does not perform well on the CovType dataset. Meanwhile, there are significant gaps between SPAN and the other methods in all the experimental settings.

To further explain the gaps between the second-order baselines and SPAN, we evaluate the empirical approximation error between the real full Hessian and the constructed approximate Hessian in Figure 3. We can see that the approximation error of our method is very close to that of the theoretical optimal solution (NewSamp) but much more accurate than the other methods. The results of the experiment demonstrates that our constructed approximate Hessian in SPAN makes better use of second-order information than other stochastic second-order optimizers (S-LBFGS, SB-BFGS, and LiSSA). Moreover, Figure 4 illustrates the empirical convergence rate of these methods. As we can see, the empirical convergence rate of our proposed method is much better than LiSSA but slightly worse than NewSamp, which confirms our theoretical results and insights again.

Even though NewSamp guarantees an excellent convergence rate, the efficiency of the algorithm mainly depends on the number of parameters. NewSamp needs a long time to perform an iteration, even when there are only 784 parameters in MNIST dataset. As a comparison, our proposed SPAN is much more competitive in terms of the robustness of the feature dimension and the per-iteration efficiency in various scenarios.

Considering the experiments mentioned above, SPAN has a tolerable Hessian approximation error which results in the runner up with respect to the empirical convergence rate. Besides, it enjoys the benefit of per-iteration efficiency, which makes it outperform others in practice. In summary, we conclude that SPAN achieves a better trade-off between per-iteration efficiency and convergence rate. Furthermore, it is robust for the sub-sampled batch size.

6 Conclusion and future work

Newton and quasi-Newton methods converge at faster rates than gradient descent methods. However, they are often expensive, computationally. In this paper, we propose SPAN, a novel method to optimize a smooth-strongly convex objective function. SPAN utilizes the first-order oracle for Hessian approximation. Therefore, it is much faster than Newton method and its alike. We give a theoretical analysis of its approximation accuracy and convergence rate. Experiments on several real datasets demonstrate that our proposed method outperforms previous state-of-the-art methods.

For the future work, the constructed approximate Hessian of SPAN can be combined with advanced first-order methods, i.e., SVRG (Johnson and Zhang 2013) and SAGA (Defazio, Bach, and Lacoste-Julien 2014), or help to introduce second-order information to complex objective functions, e.g., cubic regularization (Nesterov and Polyak 2006; Kohler and Lucchi 2017; Tripuraneni et al. 2018), neural networks with low complexity and competitive convergence rate.

Acknowledge

The authors thank Zhe Wang and Linyun Yu for their valuable discussions.

References

Agarwal, N.; Bullins, B.; and Hazan, E. 2017. Second-order stochastic optimization for machine learning in linear time. *The Journal of Machine Learning Research* 18(1):4148–4187.

Byrd, R. H.; Chin, G. M.; Neveitt, W.; and Nocedal, J. 2011. On the use of stochastic hessian information in optimization methods for machine learning. *SIAM Journal on Optimization* 21(3):977–995.

Cotter, A.; Shamir, O.; Srebro, N.; and Sridharan, K. 2011. Better mini-batch algorithms via accelerated gradient methods. In *Advances in neural information processing systems*, 1647–1655.

Defazio, A.; Bach, F.; and Lacoste-Julien, S. 2014. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, 1646–1654.

Erdogdu, M. A., and Montanari, A. 2015. Convergence rates of sub-sampled newton methods. *arXiv preprint arXiv:1508.02810*.

Gower, R.; Goldfarb, D.; and Richtárik, P. 2016. Stochastic block bfgs: Squeezing more curvature out of data. In *International Conference on Machine Learning*, 1869–1878.

Halko, N.; Martinsson, P.-G.; and Tropp, J. A. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review* 53(2):217–288.

Horn, R. A., and Johnson, C. R. 2012. *Matrix analysis*. Cambridge university press.

Johnson, R., and Zhang, T. 2013. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, 315–323.

Kohler, J. M., and Lucchi, A. 2017. Sub-sampled cubic regularization for non-convex optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 1895–1904. JMLR. org.

Li, M.; Zhang, T.; Chen, Y.; and Smola, A. J. 2014. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 661–670. ACM.

Liu, D. C., and Nocedal, J. 1989. On the limited memory bfgs method for large scale optimization. *Mathematical programming* 45(1-3):503–528.

Moritz, P.; Nishihara, R.; and Jordan, M. 2016. A linearly-convergent stochastic l-bfgs algorithm. In *Artificial Intelligence and Statistics*, 249–258.

Nesterov, Y., and Polyak, B. T. 2006. Cubic regularization of newton method and its global performance. *Mathematical Programming* 108(1):177–205.

Pilanci, M., and Wainwright, M. J. 2017. Newton sketch: A near linear-time optimization algorithm with linear-quadratic convergence. *SIAM Journal on Optimization* 27(1):205–245.

Robbins, H., and Monro, S. 1985. A stochastic approximation method. In *Herbert Robbins Selected Papers*. Springer. 102–109.

Roosta-Khorasani, F., and Mahoney, M. W. 2016. Sub-sampled newton methods ii: local convergence rates. *arXiv preprint arXiv:1601.04738*.

Shalev-Shwartz, S., and Zhang, T. 2014. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. In *International Conference on Machine Learning*, 64–72.

Tripuraneni, N.; Stern, M.; Jin, C.; Regier, J.; and Jordan, M. I. 2018. Stochastic cubic regularization for fast nonconvex optimization. In *Advances in Neural Information Processing Systems*, 2904–2913.

Xu, P.; Yang, J.; Roosta-Khorasani, F.; Ré, C.; and Mahoney, M. W. 2016. Sub-sampled newton methods with non-uniform sampling. In *Advances in Neural Information Processing Systems*, 3000–3008.