Efficient Implementation of Stochastic Proximal Point Algorithm for Matrix and Tensor Completion

Aysegul Bumin and Kejun Huang
Department of Computer and Information Science and Engineering, University of Florida
Email: (aysegul.bumin,kejun.huang)@ufl.edu

***Abstract*—We propose an efficient implementation of the stochastic proximal point algorithm (SPPA) for large-scale nonlinear least squares problems. SPPA has been shown to converge faster and more stable than the celebrated stochastic gradient descent (SGD) algorithm, and its many variations. However, the per-iteration update of SPPA itself is defined to be an optimization problem and has long been considered expensive. In this paper, we show that for nonlinear least squares problems, each iteration of SPPA can be carried out efficiently. Using Gauss-Newton along with the help of the kernel trick, we get an efficient implementation of the SPPA updates with the same order of complexity as SGD. The result is encouraging that it admits more flexible choices of the step sizes under similar assumptions. The proposed algorithm is elaborated for the problem of matrix and tensor completion. Real data experiments showcase its effectiveness in terms of convergence compared to SGD and its variants.**

I. Introduction

Stochastic (sub)gradient descent (SGD) and its variants have been widely used in the algorithm design for large-scale machine learning problems [1]. There are two main reasons for preferring the stochastic methods. On the one hand, obtaining the full gradient information may be too costly due to large size of the data set; on the other hand, in machine learning it is typically not necessary to solve the formulated problem to very high accuracy, since the ultimate goal of most tasks is to generalize well on unseen test data rather than fitting the training data. As a result, stochastic algorithms such as SGD have gained tremendous popularity.

Many methods have been developed as extensions of the plain vanilla SGD algorithm in order to accelerate its convergence rate. There is a particular line of research which focuses on reducing the variance of the stochastic gradient, resulting in the famous algorithms such as SVRG [2] and SAGA [3], however they suffer from significant time/memory complexities. Looking at a more recent past, there is another line of research based on adaptive learning schemes such as AdaGrad [4] and Adam [5], which are shown to be more effective in keeping the algorithm fully stochastic and light-weight.

A. Stochastic proximal point algorithm (SPPA)

In this work, we consider a different type of stochastic algorithm called the stochastic proximal point algorithm (SPPA), also known as incremental proximal point method [6], [7] or stochastic proximal iterations [8]. Consider the following optimization problem with the objective function in the form of a finite sum of component functions

$$\underset{\boldsymbol{\theta}\in\mathbb{R}^d}{\text{minimize}} \quad \frac{1}{n}\sum_{i=1}^{n} \ell_i(\boldsymbol{\theta}) = L(\boldsymbol{\theta}). \tag{1}$$

SPPA takes the following simple form:

Algorithm 1 Stochastic proximal point algorithm (SPPA)

1: initialize $\boldsymbol{\theta}_0$, $t \leftarrow 0$
2: **repeat**
3: randomly draw i from $\{1, \dots, n\}$
4: $\boldsymbol{\theta}_{t+1} \leftarrow \arg\min_{\boldsymbol{\theta}} \lambda_t \ell_i(\boldsymbol{\theta}) + (1/2)\|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|^2$
5: $t \leftarrow t + 1$
6: **until** convergence

The update rule in line 3 is called the proximal operator of the function $\lambda_t \ell_i$ evaluated at $\boldsymbol{\theta}_t$, and is sometimes denoted as $\text{Prox}_{\lambda_t \ell_i}(\boldsymbol{\theta}_t)$. This is the stochastic version of the proximal point algorithm, which dates back to Rockafellar [9]. Due to the abstraction of the per-iteration update rule, SPPA is not as extensively applicable as SGD. It is also asking for more information from the problem than merely the first-order derivatives. However, with the help of more information inquired, there is also hope that it provides faster and more robust convergence guarantees.

Convergence analyses of SPPA have been conducted mostly on convex problems [6], [8], [10]. The studies show that SPPA is much more robust to instabilities, and converges similar to SGD for convex problems. Most authors also accept the premise that the proximal operator is sometimes difficult to evaluate, and thus proposed variations to the plain vanilla version to handle more complicated problem structures [11], [12], [13], [14].

There exists some more recent work focusing on convergence analyses of SPPA for nonconvex optimization problems [14], [15], [16]. The analyses performed by [14] and [15] are based on an imaginary sequence (that is not computed in practice) $\{\widetilde{\boldsymbol{\theta}}_t\}$ as $\widetilde{\boldsymbol{\theta}}_t = \arg\min_{\boldsymbol{\theta}} \lambda_t L(\boldsymbol{\theta}) + (1/2)\|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|^2$, i.e., the proximal operator *of the full loss function* from the algorithm sequence $\{\boldsymbol{\theta}_t\}$. The results show that this imaginary sequence $\{\widetilde{\boldsymbol{\theta}}_t\}$ converges to a stationary point in expectation. More recently, [16] as well showed stationary convergence of SPPA for a specific problem with convex objective and nonconvex constraints.

II. SPPA-GN for Nonlinear Least Squares

The main contribution of this paper is to apply SPPA to a large family of nonconvex optimization problems and show that the seemingly complicated proximal operator update can still be efficiently obtained, namely the nonlinear least squares (NLS) problem.

A NLS problem takes the following form

$$\underset{\boldsymbol{\theta}\in\mathbb{R}^d}{\text{minimize}} \quad \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}(\varphi_i(\boldsymbol{\theta}))^2, \tag{2}$$

where each φ_i is a general nonlinear function with respect to $\boldsymbol{\theta}$. It is a classical nonlinear programming problem [17], [18] with many useful applications as of today, including least squares neural networks [19], where each φ_i corresponds to the residual of fitting for the ith data sample for regression. We will show that problems of this form can be efficiently executed by SPPA, despite its seemingly complication.

In this section we describe how to efficiently evaluate the proximal operator, which is the core computation of SPPA, when it is applied to NLS (2). As we will see, the update rules are surprisingly simple but very powerful, leading to a per-iteration complexity that is almost as efficient as one SGD step. However, since it is in the framework of SPPA, the resulting convergence will be a lot more robust. We also describe in detail how it can be applied to matrix completion and tensor completion.

A. General description

To apply SPPA to a general NLS problem, the main challenge is to efficiently evaluate the proximal operator

$$\boldsymbol{\theta}_{t+1} \leftarrow \arg\min_{\boldsymbol{\theta}} \frac{\lambda_t}{2}(\varphi_i(\boldsymbol{\theta}))^2 + \frac{1}{2}\|\boldsymbol{\theta}-\boldsymbol{\theta}_t\|^2. \tag{3}$$

Notice that this function itself is a nonlinear least squares objective, although with only one component function together with the proximal term.

The traditional wisdom to solve a NLS problem is to apply the Gauss-Newton (GN) algorithm: at each iteration, we first take a first-order approximation of the vector-valued function inside the Euclidean norm, and set the update as the solution of the approximated linear least squares problem. It is a well-known algorithm that can be found in many standard textbooks, e.g., [17], [20], [18].

To apply GN to (3), we first take linear approximation of φ_i at the current update $\overline{\boldsymbol{\theta}}$ as

$$\varphi_i(\boldsymbol{\theta}) \approx \varphi_i(\overline{\boldsymbol{\theta}}) + \nabla\varphi_i(\overline{\boldsymbol{\theta}})^\top(\boldsymbol{\theta}-\overline{\boldsymbol{\theta}}),$$

and set the solution of the following problem $\boldsymbol{\theta}^+$ as the next update

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \frac{\lambda_t}{2}(\varphi_i(\overline{\boldsymbol{\theta}}) + \nabla\varphi_i(\overline{\boldsymbol{\theta}})^\top(\boldsymbol{\theta}-\overline{\boldsymbol{\theta}}))^2 + \frac{1}{2}\|\boldsymbol{\theta}-\boldsymbol{\theta}_t\|^2. \tag{4}$$

Obviously, (4) has a closed form solution

$$\boldsymbol{\theta}^+ = \boldsymbol{\theta}_t - \left(\frac{1}{\lambda_t}\boldsymbol{I} + \boldsymbol{g}_i\boldsymbol{g}_i^\top\right)^{-1}\boldsymbol{g}_i(\varphi_i(\overline{\boldsymbol{\theta}}) - \boldsymbol{g}_i^\top(\overline{\boldsymbol{\theta}}-\boldsymbol{\theta}_t)), \tag{5}$$

Algorithm 2 SPPA-GN

initialize $\boldsymbol{\theta}_0$, $t \leftarrow 0$
repeat
  randomly draw i from $\{1,\dots,n\}$
  $\boldsymbol{\theta}^+ \leftarrow \boldsymbol{\theta}_t$
  repeat
    $\overline{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}^+$
    $\boldsymbol{\theta}^+ \leftarrow \boldsymbol{\theta}_t - \frac{\varphi_i(\overline{\boldsymbol{\theta}}) - \nabla\varphi_i(\overline{\boldsymbol{\theta}})^\top(\overline{\boldsymbol{\theta}}-\boldsymbol{\theta}_t)}{\lambda_t^{-1}+\|\nabla\varphi_i(\overline{\boldsymbol{\theta}})\|^2}\nabla\varphi_i(\overline{\boldsymbol{\theta}})$
  until convergence
  $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}^+$, $t \leftarrow t+1$
until convergence

where we denote $\boldsymbol{g}_i = \nabla\varphi_i(\overline{\boldsymbol{\theta}})$ to simplify notation. Notice that the matrix to be inverted in (5) has a simple "identity plus rank-one" structure, implying that it can be efficiently computed in linear time. Using the "kernel trick" [18, pp.332]

$$(\boldsymbol{A}^\top\boldsymbol{A} + \alpha\boldsymbol{I})^{-1}\boldsymbol{A}^\top = \boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{A}^\top + \alpha\boldsymbol{I})^{-1}, \tag{6}$$

update (5) simplifies to

$$\boldsymbol{\theta}^+ = \boldsymbol{\theta}_t - \frac{\varphi_i(\overline{\boldsymbol{\theta}}) - \nabla\varphi_i(\overline{\boldsymbol{\theta}})^\top(\overline{\boldsymbol{\theta}}-\boldsymbol{\theta}_t)}{\lambda_t^{-1} + \|\nabla\varphi_i(\overline{\boldsymbol{\theta}})\|^2}\nabla\varphi_i(\overline{\boldsymbol{\theta}}). \tag{7}$$

As we can see, each GN update only takes $\mathcal{O}(d)$ flops, which is as cheap as that of a SGD step. To fully obtain the proximal operator (3), one has to run GN for several iterations. However, thanks to the superlinear convergence rate of GN near its optimal [20], which is indeed the case if we initiate at $\boldsymbol{\theta}_t$ because of the proximal term, it typically takes no more than 5–10 GN updates. The detailed description of the proposed algorithm, which we term SPPA-GN, for solving general NLS problems is shown in Algorithm 2.

In the rest of this section, we describe how SPPA-GN can be applied to two widely used unsupervised learning problems, matrix completion and tensor completion.

B. Application to matrix completion

In matrix completion, one is given a subset of entries from a big matrix $\boldsymbol{X} \in \mathbb{R}^{m\times n}$, and the goal is to infer the unseen entries based on the assumption that the rank of $\boldsymbol{X}$, k, is much smaller than its ambient dimension. To do so, a standard problem formulation is to find factor matrices $\boldsymbol{A} \in \mathbb{R}^{m\times k}$ and $\boldsymbol{B} \in \mathbb{R}^{n\times k}$ such that $\boldsymbol{X} \approx \boldsymbol{A}\boldsymbol{B}^\top$ for the given set of data entries [21], [22], [23]. Specifically, the problem is formulated as

$$\underset{\boldsymbol{A},\boldsymbol{B}}{\text{minimize}} \quad \frac{1}{|\mathcal{S}|}\sum_{(i,j)\in\mathcal{S}}\frac{1}{2}(\boldsymbol{a}_i^\top\boldsymbol{b}_j - X_{ij})^2, \tag{8}$$

where $\boldsymbol{a}_i$ and $\boldsymbol{b}_j$ are the ith row and jth row of $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively, and $\mathcal{S}$ denotes the index set of the available entries for training. After solving (8), we use the product $\boldsymbol{A}\boldsymbol{B}^\top$ to predict the unseen entries from $\boldsymbol{X}$. Problem (8) is an instance of NLS.

To apply SPPA-GN for solving (8), we only need to specify the corresponding $\nabla\varphi_{ij}(\boldsymbol{A},\boldsymbol{B})$. Notice that the (i,j)th

Algorithm 3 SPPA-GN for matrix completion

1: initialize $\boldsymbol{A}$ and $\boldsymbol{B}$, $t \leftarrow 0$
2: **repeat**
3: randomly draw (i, j) from $\mathcal{S}$
4: $\boldsymbol{a} \leftarrow \boldsymbol{a}_i$, $\boldsymbol{b} \leftarrow \boldsymbol{b}_j$
5: **repeat**
6: $\gamma \leftarrow \dfrac{\boldsymbol{a}^\top \boldsymbol{b}_j + \boldsymbol{a}_i^\top \boldsymbol{b} - \boldsymbol{a}^\top \boldsymbol{b} - X_{ij}}{\lambda_t^{-1} + \|\boldsymbol{a}\|^2 + \|\boldsymbol{b}\|^2}$
7: $\begin{cases} \boldsymbol{a} \leftarrow \boldsymbol{a}_i - \gamma \boldsymbol{b} \\ \boldsymbol{b} \leftarrow \boldsymbol{b}_j - \gamma \boldsymbol{a} \end{cases}$
8: **until** convergence
9: $\boldsymbol{a}_i \leftarrow \boldsymbol{a}$, $\boldsymbol{b}_j \leftarrow \boldsymbol{b}$, $t \leftarrow t + 1$
10: **until** convergence

component only involve the ith row of $\boldsymbol{A}$ and the jth row of $\boldsymbol{B}$. Let $\boldsymbol{\theta} = (\boldsymbol{a}, \boldsymbol{b})$ and

$$\varphi_{ij}(\boldsymbol{\theta}) = \boldsymbol{a}^\top \boldsymbol{b} - X_{ij},$$

it is easy to see that

$$\nabla_{\boldsymbol{a}} \varphi_{ij} = \boldsymbol{b}, \quad \nabla_{\boldsymbol{b}} \varphi_{ij} = \boldsymbol{a}.$$

We let $\boldsymbol{a}_i$ and $\boldsymbol{b}_j$ denote the $\boldsymbol{\theta}_t$ that is obtained outside of the GN iterates, and $\boldsymbol{a}$ and $\boldsymbol{b}$ denote the current GN update. The scaling factor in front of $\nabla\varphi$ in line 7 of Algorithm 1 equals to

$$\frac{\boldsymbol{a}^\top \boldsymbol{b}_j + \boldsymbol{a}_i^\top \boldsymbol{b} - \boldsymbol{a}^\top \boldsymbol{b} - X_{ij}}{\lambda_t^{-1} + \|\boldsymbol{a}\|^2 + \|\boldsymbol{b}\|^2}.$$

These suffice to fully characterize SPPA-GN for matrix completion, as shown in Algorithm 3.

C. Application to tensor completion

A tensor is a data array with more than two indexes, which has been proven extremely useful for data with multiple modalities [24], [25], [26]. Simply put, it can be considered higher-order extension of a matrix.

Similar to matrix completion, the task of tensor completion is to fit a low-rank tensor to an incomplete data tensor, and use the low-rank model to predict the values of the unobserved entries. There are multiple definitions of a tensor rank, such as the CP-rank and the multi-linear (Tucker) rank, leading to different tensor completion techniques [27], [28], [29]. In this paper we focus on the CP-rank derived from the canonical polyadic decomposition of a tensor [30].

We consider a general order-D tensor $\boldsymbol{X}$ of size $I_1 \times \cdots \times I_D$. A tensor has rank K if there are D matrices $\boldsymbol{A}^{(1)} \in \mathbb{R}^{I_1 \times K}, \ldots, \boldsymbol{A}^{(D)} \in \mathbb{R}^{I_D \times K}$ such that the $(i_1, \ldots, i_D)$th entry equals to

$$\left\langle \boldsymbol{a}_{i_1}^{(1)}, \ldots, \boldsymbol{a}_{i_D}^{(D)} \right\rangle = \sum_{k=1}^{k} \prod_{d=1}^{D} \boldsymbol{A}^{(d)}(i_d, k),$$

where we use $\langle \cdot \rangle$ to denote the "inner-product" between D vectors, and $\boldsymbol{a}_{i_d}^{(d)}$ denotes the i_dth row of the factor matrix $\boldsymbol{A}^{(d)}$. We use a boldface letter $\boldsymbol{i}$ to denote the index of a particular entry of $\boldsymbol{X}$, i.e., $\boldsymbol{i} = (i_1, \ldots, i_D)$. With these notations, we can formulate the tensor completion problem as

$$\underset{\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(D)}}{\text{minimize}} \quad \sum_{\boldsymbol{i} \in \mathcal{S}} \left(\left\langle \boldsymbol{a}_{i_1}^{(1)}, \ldots, \boldsymbol{a}_{i_D}^{(D)} \right\rangle - X(\boldsymbol{i}) \right)^2, \tag{9}$$

where again $\mathcal{S}$ denotes the index set of available entries of the data tensor $\boldsymbol{X}$. This is another instance of NLS.

Similar to the matrix case, we only have to specify the corresponding $\nabla\varphi_{\boldsymbol{i}}$ to be able to fully execute SPPA-GN. We use $*$ to denote element-wise multiplication of two vectors, and define

$$\boldsymbol{a}^{(-d)} = \boldsymbol{a}^{(1)} * \cdots * \boldsymbol{a}^{(d-1)} * \boldsymbol{a}^{(d+1)} * \cdots * \boldsymbol{a}^{(D)}.$$

It is easy to see that

$$\nabla_{\boldsymbol{a}^{(d)}} \varphi_{\boldsymbol{i}} = \boldsymbol{a}^{(-d)}.$$

Furthermore, the scaling factor in front of the gradient in line 7 of Algorithm 1 equals to

$$\frac{\sum_d \langle \boldsymbol{a}^{(d)}, \boldsymbol{a}^{(-d)} \rangle - (D-1) \langle \boldsymbol{a}_{i_1}^{(1)}, \ldots, \boldsymbol{a}_{i_D}^{(D)} \rangle - X(\boldsymbol{i})}{\lambda_t^{-1} + \sum_d \|\boldsymbol{a}^{(-d)}\|^2}$$

The resulting SPPA-GN for tensor completion is fleshed out in Algorithm 4. As a sanity check, when $D = 2$, this algorithm recovers the aforementioned SPPA-GN for matrix completion.

Algorithm 4 SPPA-GN for tensor completion

1: initialize $\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(D)}$, $t \leftarrow 0$
2: **repeat**
3: randomly draw $(i_1, \ldots, i_D)$ from $\mathcal{S}$
4: **for** $d = 1, \ldots, D$ **do**
5: $\boldsymbol{a}^{(d)} \leftarrow \boldsymbol{a}_{i_d}^{(d)}$
6: **end for**
7: **repeat**
8: **for** $d = 1, \ldots, D$ **do**
9: $\boldsymbol{a}^{(-d)} \leftarrow \boldsymbol{a}^{(1)} * \cdots * \boldsymbol{a}^{(d-1)} * \boldsymbol{a}^{(d+1)} * \cdots * \boldsymbol{a}^{(D)}$
10: **end for**
11: $\gamma \leftarrow \dfrac{\sum_d \langle \boldsymbol{a}^{(d)}, \boldsymbol{a}^{(-d)} \rangle - (D-1) \langle \boldsymbol{a}^{(1)}, \ldots, \boldsymbol{a}^{(D)} \rangle - X(\boldsymbol{i})}{\lambda_t^{-1} + \sum_d \|\boldsymbol{a}^{(-d)}\|^2}$
12: **for** $d = 1, \ldots, D$ **do**
13: $\boldsymbol{a}^{(d)} \leftarrow \boldsymbol{a}_{i_d}^{(d)} - \gamma \boldsymbol{a}^{(-d)}$
14: **end for**
15: **until** convergence
16: **for** $d = 1, \ldots, D$ **do**
17: $\boldsymbol{a}_{i_d}^{(d)} \leftarrow \boldsymbol{a}^{(d)}$
18: **end for**
19: $t \leftarrow t + 1$
20: **until** convergence

III. Experiments

We now show some real data experiments to demonstrate the effectiveness of the proposed SPPA-GN algorithm applied to matrix and tensor completions. We compare our proposed algorithm with three variants of SGD: the original version with various step strategies, AdaGrad [4], and Adam [5], with the

default settings suggested in their original paper. We carefully coded all the algorithms, unless otherwise mentioned, in Python. All the experiments are performed through Jupyter Notebook on a Linux Desktop with Intel i7 cores vPro 8th Gen and 16 GB of RAM.

Remark. Since the main theme of this paper is to show that SPPA, a generic algorithmic framework with abstract definition of its updates, can be efficiently implemented for nonlinear least squares, we only compared our method to other stochastic algorithms *for the same formulations* (8) *and* (9). We are aware that there are other formulations for matrix/tensor completion, such as those based on convex relaxations [31], [27]. The purpose of our experiments is not to achieve more accurate predictions in completion. We try to demonstrate that for the same basic formulations, SPPA achieves a much faster convergence rate than other stochastic algorithms.

A. Matrix completion on MovieLens

We test the matrix completion performance on the MovieLens data set [32]. Specifically, all algorithms are test on the `ml-latest-small` file, which consists of approximately 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users. The data set is split into training/testing by the creator. We ran all algorithms by drawing samples from the training set, and we evaluate the mean squared error on the test set.

As mentioned before, all algorithms are implemented by ourselves except for SGD for matrix completion, which is adopted from Albert Au Yeung's blog[1]. In recommendation systems and collaborative filtering people usually add bias terms to improve prediction accuracy, which is included in this implementation. However, for fair comparison, these bias terms are set to be zeros in our experiment.

The convergence plots on the test set are shown in Figures 1 and 2 for two different ranks. In all Figures, the label "Best y" shows the constant value that gives the best performance is y, and the labels 1/t and 1/sqrt(t) show that the step size is diminishing with rates 1/t and 1/sqrt(t) respectively. As we can see, SPPA indeed converges the fastest for various step size choices. Interestingly, the performance of SGD comes second best, outperforming AdaGrad and Adam. Our experience is that AdaGrad and Adam did work well on synthetic data set when the data matrices are indeed approximately low rank. However, on real data they do not perform as well as one would expect.

B. Tensor completion on Facebook wall posts

We test the tensor completion performance on the Facebook wall posts data set[2], which consists of wall post records of approximately 47,000 Facebook users over the span of more than years. We form a $46952 \times 46951 \times 1592$, with the (i, j, k) entry indicating the number of wall posts user i wrote to user j on day k. This can be treated as a tensor completion problem because not all user interactions are in the form of public wall posts,

[1]http://www.albertauyeung.com/post/python-matrix-factorization/

[2]http://konect.uni-koblenz.de/networks/facebook-wosn-wall

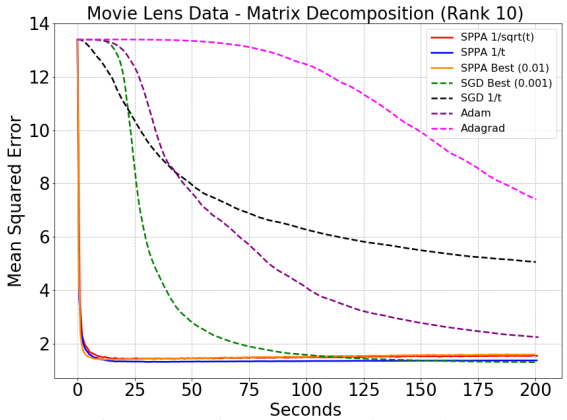

Fig. 1: MovieLens data with rank 10.

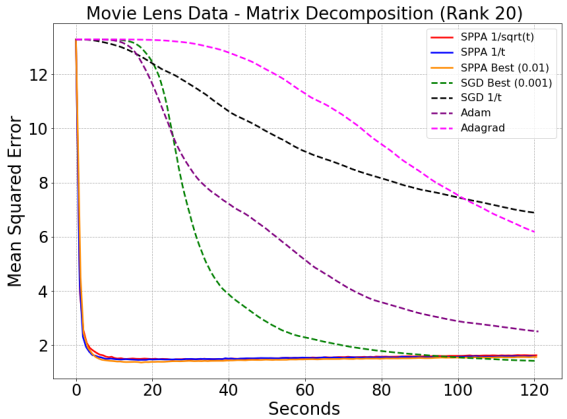

Fig. 2: MovieLens data with rank 20.

but we might discover unrevealed relationships between people by fitting a low-rank tensor to the observed wall post counts.

In this case we randomly sampled 10% of the tensor entries and use them as the test set to evaluate the prediction performances of various stochastic algorithms. The samples that train the model are drawn from the rest of the available entries.

The convergence plots for this data set is shown in Figures 3 and 4 for two different ranks. As we can see, the performance of these algorithms are consistent with how they did in the matrix case, although the data set has been completely changed. SPPA with constant step size converges the fastest at the beginning, although the other algorithms tend to swamp at a bigger objective value. SPPA performed similarly with different constant step sizes, the constant step size in Figures 3 and 4 is 0.1. The performance of SPPA with diminishing step size is closer to the SPPA with constant step size in lower rank setting. Being the second best, SGD performed comparably better than other algorithms with constant step size 0.1, converging not as fast at the beginning, but eventually converging to a more accurate solution.

IV. Conclusion

In this paper we presented an efficient implementation of the stochastic proximal point algorithm (SPPA) for nonlinear least squares problems. The specific approach to efficiently

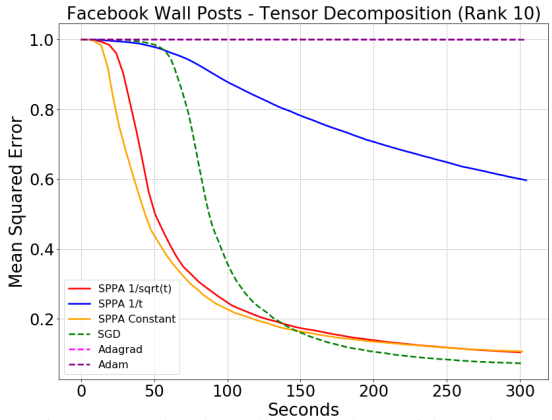

Fig. 3: Facebook wall posts data with rank 10.

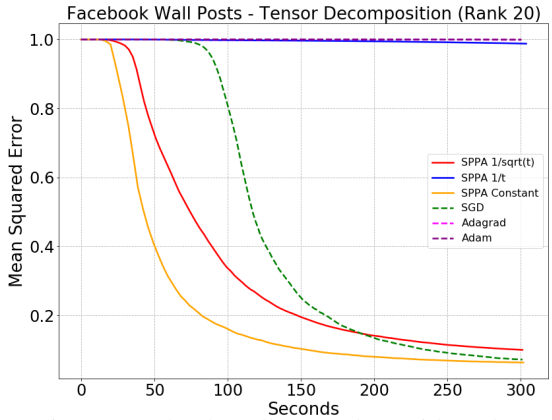

Fig. 4: Facebook wall posts data with rank 20.

evaluate the proximal operators is based on the Gauss-Newton (GN) algorithm with a twist of the kernel trick, hence the name SPPA-GN. The resulting algorithm has similar per-iteration complexity to that of SGD, while enjoying faster and more robust convergence under milder conditions. The algorithm was elaborated on two important problems, matrix and tensor completion, with implementation details fully driven in the paper. Real data experiments on the MovieLens data and Facebook wall posts data showed the effectiveness of SPPA-GN.

References

[1] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018.

[2] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in Neural Information Processing Systems*, 2013, pp. 315–323.

[3] A. Defazio, F. Bach, and S. Lacoste-Julien, "Saga: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Advances in Neural Information Processing Systems*, 2014, pp. 1646–1654.

[4] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[5] J. B. Diederik P. Kingma, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations*, 2014.

[6] D. P. Bertsekas, "Incremental proximal methods for large scale convex optimization," *Mathematical programming*, vol. 129, no. 2, p. 163, 2011.

[7] ——, "Incremental gradient, subgradient, and proximal methods for convex optimization: A survey," *Optimization for Machine Learning*, vol. 2010, no. 1-38, p. 3, 2011.

[8] E. K. Ryu and S. Boyd, "Stochastic proximal iteration: a non-asymptotic improvement upon stochastic gradient descent," *Preprint*, 2014.

[9] R. T. Rockafellar, "Monotone operators and the proximal point algorithm," *SIAM journal on control and optimization*, vol. 14, no. 5, pp. 877–898, 1976.

[10] P. Bianchi, "Ergodic convergence of a stochastic proximal point algorithm," *SIAM Journal on Optimization*, vol. 26, no. 4, pp. 2235–2260, 2016.

[11] M. Wang and D. P. Bertsekas, "Incremental constraint projection-proximal methods for nonsmooth convex optimization," *SIAM J. Optim.(to appear)*, 2013.

[12] J. C. Duchi and F. Ruan, "Stochastic methods for composite and weakly convex optimization problems," *SIAM Journal on Optimization*, vol. 28, no. 4, pp. 3229–3259, 2018.

[13] H. Asi and J. C. Duchi, "Stochastic (approximate) proximal point methods: Convergence, optimality, and adaptivity," *SIAM Journal on Optimization*, vol. 29, no. 3, pp. 2257–2290, 2019.

[14] D. Davis and D. Drusvyatskiy, "Stochastic model-based minimization of weakly convex functions," *SIAM Journal on Optimization*, vol. 29, no. 1, pp. 207–239, 2019.

[15] H. Asi and J. C. Duchi, "The importance of better models in stochastic optimization," *Proceedings of the National Academy of Sciences*, vol. 116, no. 46, pp. 22 924–22 930, 2019.

[16] S. Chen, Z. Deng, S. Ma, and A. M.-C. So, "Manifold proximal point algorithms for dual principal component pursuit and orthogonal dictionary learning," *arXiv preprint arXiv:2005.02356*, 2020.

[17] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, 1999.

[18] S. Boyd and L. Vandenberghe, *Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares*. Cambridge University Press, 2018.

[19] P. P. Van Der Smagt, "Minimisation methods for training feedforward neural networks," *Neural networks*, vol. 7, no. 1, pp. 1–11, 1994.

[20] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[21] R. Sun and Z.-Q. Luo, "Guaranteed matrix completion via non-convex factorization," *IEEE Transactions on Information Theory*, vol. 62, no. 11, pp. 6535–6579, 2016.

[22] R. Ge, J. D. Lee, and T. Ma, "Matrix completion has no spurious local minimum," in *Advances in Neural Information Processing Systems*, 2016, pp. 2973–2981.

[23] Y. Chi, Y. M. Lu, and Y. Chen, "Nonconvex optimization meets low-rank matrix factorization: An overview," *IEEE Transactions on Signal Processing*, vol. 67, no. 20, pp. 5239–5269, 2019.

[24] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[25] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky, "Tensor decompositions for learning latent variable models," *Journal of Machine Learning Research*, vol. 15, pp. 2773–2832, 2014.

[26] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551–3582, 2017.

[27] S. Gandy, B. Recht, and I. Yamada, "Tensor completion and low-n-rank tensor recovery via convex optimization," *Inverse Problems*, vol. 27, no. 2, p. 025010, 2011.

[28] J. Liu, P. Musialski, P. Wonka, and J. Ye, "Tensor completion for estimating missing values in visual data," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 208–220, 2012.

[29] A. Krishnamurthy and A. Singh, "Low-rank matrix and tensor completion via adaptive sampling," in *Advances in Neural Information Processing Systems*, 2013, pp. 836–844.

[30] F. L. Hitchcock, "The expression of a tensor or a polyadic as a sum of products," *Journal of Mathematics and Physics*, vol. 6, no. 1-4, pp. 164–189, 1927.

[31] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," *Foundations of Computational mathematics*, vol. 9, no. 6, p. 717, 2009.

[32] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *AMC Transactions on Interactive Intelligent Systems (TIIS)*, vol. 5, no. 4, pp. 1–19, 2015.