Smoothed Analysis in Learning: Tensors and k-Means

Hunter Lang
MIT
hjl@mit.edu

Carlos Cortez
MIT
cortezc@mit.edu

Abstract

Most learning problems are hard in the worst-case, so much of the current research focuses on finding good heuristics, polynomial-time approximation algorithms, or special cases with provable accuracy and runtime guarantees. But many algorithms that are inefficient in the worst case consistently seem to run fast in practice (Simplex being the classical example). Smoothed analysis gives a theoretical framework for understanding performance on real-world data, specifically for problems where some component is not adversarial. This is a natural assumption in learning, since data in the learning setting are usually prone to measurement or modeling noise. We survey the (very much ongoing) application of smoothed analysis to learning problems by way of two examples: k-means and tensor decomposition.

I. Introduction

Spielman and Teng [1], [2] introduced smoothed analysis to give a more suitable framework for predicting real-life algorithm performance. Not every algorithm that runs fast in practice is polynomial-time; worst-case analysis falls short of explaining why some "slow" algorithms are empirically quite efficient. As a first application of their techniques, the original paper [1] gave a proof that the Simplex algorithm has polynomial "smoothed complexity": if you assume the input data are subject to random noise, Simplex runs in expected polynomial time. This sparked a host of papers applying smoothed analysis to classic combinatorial optimization problems [2]. The key assumption of the smoothed setting is that some component of the problem is not adversarial. The hope is that worst-case instances are somehow isolated in the input space (indeed, the worst-case inputs to many algorithms are intricate and fragile), so that any real data is unlikely to be a worst-case instance. In recent years, there has been an increasing trend of applying smoothed analysis to learning problems [2], [4], [5], [6]. The noisy assumptions required in the smoothed model are natural in the learning setting, where data are gathered by surveys, measurements, samples, and other noise-prone or imperfect methods. In the next section, we give a brief formal definition of smoothed analysis, following [2], to set the environment for the examples that follow. In Section III we discuss at a high level the result of [4], showing that the k-means clustering method has polynomial smoothed runtime. In Section IV we

examine a tensor decomposition algorithm in a smoothed setting, showing not only that it runs in polynomial time on smoothed instances, but also that it works provably well, i.e. that it recovers good factors. In Section V we conclude and offer some speculation about directions for further research.

II. Preliminaries

When A is an algorithm for solving problem P, we denote by $T_A[x]$ the runtime of A on an input instance x. We consider the input domain Ω to problem P as the union of disjoint subdomains $\{\Omega_1, \ldots, \Omega_n, \ldots\}$, where Ω_n is the family of all inputs of size n. Given this setting we can define worst-case and smoothed runtimes. The *worst-case measure* is defined as

$$\mathrm{WC}_A(n) = \max_{x \in \Omega_n} T_A[x].$$

The *smoothed complexity* of A with σ-Gaussian perturbations is given by

$$\mathrm{Smoothed}_A^{\sigma}(n) = \max_{x \in [-1,1]^n} \mathbb{E}_g[T_A(x+g)],$$

where g is a Gaussian random vector of variance σ^2. The original input x is perturbed by g to obtain a new input $x+g$, which is given to the algorithm.

We can view σ as a parameter that interpolates between worst-case and average-case analysis. When $\sigma = 0$ the smoothed complexity is the same as the worst-case measure. When σ is very large, the smoothed complexity approaches the average-case complexity, since g dominates the original x. Here we are interested in the case where σ is small compared to $||x||$, so that $x+g$ is only a slight perturbation. We now give an example of using smoothed analysis to analyze the runtime of a learning algorithm, k-means.

III. k-means

The k-means algorithm solves the *clustering problem*, set as follows. Given a set S of points in a (generally high-dimensional) space $\mathbb{R}^d$, output a k-partition of S (a collection of k "clusters") such that the elements in each cluster are similar under a chosen metric. k-means begins by arbitrarily choosing the k cluster centers (how to choose these centers is the subject of much research and debate). The main algorithm consists of two iterative steps: first, it assigns each point in S to the closest center. Next, it computes the center of mass of each existing cluster (the cluster mean). These means become the new cluster centers and the assignment process repeats. The algorithm terminates when the local improvements to cluster centers no longer affect the assignment of points.

In [3], Vattani shows that an adversary can choose initial centers that will force k-means to iterate $2^{\Omega(n)}$ times for $d \geq 2$. But k-means is usually very fast in practice (see, for instance, [10]). So the setting seems ripe for smoothed analysis: there is a large gap between theoretical and observed runtimes, and the assumption that data points are subject

to noise is reasonable. Before we proceed it is important to point out that aside from a slow worst-case runtime, there is another problem with k-means: there are no guarantees on the quality of the solution it produces; the algorithm is only guaranteed to converge to a local optimum and is sensitive to the initial choice of cluster centers [9]. There are mild conditions under which a variant of the above algorithm outputs a provably good clustering [8], but these techniques do not use what we consider smoothed analysis, instead defining "separability conditions" on the input instance. We focus on the first problem, a problem of runtime, and summarize the argument of Arthur et al. [4], the main result of which is the following theorem.

Theorem 1. Fix a set $S' \subset [0,1]^d$ of n data points, and independently perturb each point in S' by a Gaussian distribution with mean 0 and variance σ^2. Label the new set S. Then the expected running time of k-means on S is bounded by a polynomial in n and $1/\sigma$.

The proof of Theorem 1 makes heavy use of the potential function

$$\Psi = \sum_{x \in S} ||x - c(x)||^2,$$

where $c(x)$ is the center of the cluster to which the point x is assigned, and consists of two steps. It first shows that the potential is bounded by a polynomial after the first iteration of the algorithm. The second step shows that the potential decreases by a polynomial in each iteration (or rather, in a sequence of iterations). The key insight of the paper is its introduction of *transition blueprints*, directed multigraphs that model the flow of points between clusters in an iteration of the algorithm. Vertices are clusters and edges are drawn when points in S go from one cluster to another in an iteration. The potential can decrease in two ways: cluster centers can change, or points can be reassigned from one cluster to another. No iteration can decrease the potential. For the potential difference to be small in a given iteration, it must be the case that no cluster center moved very much, and that no points were reassigned to a cluster very far from their current cluster. The paper examines the probability that the potential decrease is small for a given transition blueprint; this leads naturally to an upper-bound on the probability that the minimum potential decrease across all iterations is small. Because of the small perturbations of the input set, transition blueprints are general enough so that each one represents multiple possible iterations of the algorithm, but specific enough that they *almost* uniquely determine what happens during an iteration. The paper does a case-by-case analysis of different types of blueprints to show that with high probability, there are no iterations where cluster centers don't move far and points don't get swapped between far-away clusters. This implies the result.

k-means has expected polynomial smoothed complexity, matching observations in practice. The full analysis is quite intricate, so the resulting polynomial has degree much higher than empirical results would suggest, but this is in line with other results in smoothed analysis of runtimes [2]. But we still have no provable guarantees for optimality. In the next section, we examine in detail the more interesting example of analyzing an algorithm *with provable guarantees* in a smoothed setting.

IV. Tensor Decomposition

Tensor decomposition methods can be used for community detection and for learning topic models (including LDA), multi-view mixture models, and phylogenetic trees. Tensors are especially useful because under mild conditions, the factors of a rank-R 3-tensor decomposition $T = \sum_{i=1}^{R} a_i \otimes b_i \otimes c_i$ are unique [7], and there exists a polynomial-time algorithm for recovering them, initially due to Jennrich. Denote by A the $n \times R$ matrix whose columns are the a_i, and likewise for B. Then the conditions for Jennrich's algorithm are that A and B have rank R, and that no two vectors c_i, c_j are linearly dependent. Jennrich's algorithm extends naturally to higher-order tensors. Assume we are given a order-l tensor T of dimension n (so $T \in \mathbb{R}^{n^{\times l}}$) and rank R. We want to decompose it into a sum of rank-one tensors, i.e.:

$$T = \sum_{i=1}^{R} u_i^{(1)} \otimes u_i^{(2)} \otimes \cdots \otimes u_i^{(l)}.$$

Jennrich's algorithm works for order-3 tensors, so to work with the general case a good approach is to flatten a tensor T into an order 3 tensor so as to obtain a decomposition, through Jennrich's, of the form

$$T = \sum_{i=1}^{R} U_i \otimes V_i \otimes W_i$$

where

$$U_i = u_i^{(1)} \odot \cdots \odot u_i^{(\lfloor \frac{l-1}{2} \rfloor)},\ V_i = v_i^{(\lfloor \frac{l-1}{2} \rfloor + 1)} \odot \cdots \odot v_i^{(2\lfloor \frac{l-1}{2} \rfloor)}, \text{ and } W_i = u_i^{(l)} \text{ or } u_i^{(l-1)} \odot u_i^{(l)}.$$

Here $\odot$ is the Khatri-Rao product: given $M \in \mathbb{R}^{m \times r},\ N \in \mathbb{R}^{n \times r}$, $M \odot N$ is the matrix of dimension $mn \times r$ whose i-th column is $M_i \otimes N_i$. [MORE–it also only works for 3-tensors] Jennrich's algorithm works only if $R \leq n$, that is, only if the rank of the tensor is at most the dimension of its factors. In the overcomplete case where $R > n$, we can "flatten" the factors using the Khatrio-Rao product; if the flattened factors still satisfy the conditions of Jennrich's algorithm, then we can solve this higher-order decomposition problem. Also note that overcomplete decomposition is NP-hard in general [11] [12]. [MENTION REM 1.8 before next sentence / paragraph] We show that in a smoothed setting where the tensor T is perturbed, the flattened factors will indeed satisfy the conditions for Jennrich's algorithm, so we can get around the worst-case inefficiency.

TODO: put remark 1.8, condition 2.2 and a noiseless version of corollary 2.4 here. they explain very clearly why we want to show that the kruskal rank multiplies.

The overcomplete algorithm above [THE ONE MENTIONED IN REM 1.8] works well in the *exact* case, when we know T. But in the learning setting, we don't know T exactly–we have an approximation $\tilde{T}$ of T, since we can only take polynomially many samples [VAGUE?]. Hence, we need to make sure the algorithm is robust in the presence of noise. In particular, we prove that if each entry of $E = T - \tilde{T}$ is bounded by $\epsilon \cdot \text{poly}_l(1/k, 1/n, 1/\delta)$, then we can compute a decomposition in which each rank-1 tensor in the decomposition has an additive error of at most ϵ.

Lemma 1. *Let $\mathcal{V} \subset \mathbb{R}^{nm}$ be a subspace with $\dim(\mathcal{V}) = \delta nm$. Furthermore, suppose r, θ, δ' satisfy $\delta' \leq \delta/2$, $r\delta' m < \delta n/2$ and $\theta = \frac{1}{nm^{3/2}}$. Then there exist r matrices $M_1, M_2, \ldots, M_r$ of dimension $n \times m$ such that*

1. *$vec(M_i) \in \mathcal{V}$ $\forall i \in [r]$*
2. *$M_1, \ldots, M_r$ form an ordered (θ, δ') orthogonal system*

where $\vec{M_i}$ is the vector resulting from flattening M_i. In particular, if $m \leq \sqrt{n}$, the matrices form an ordered $(\theta, \delta/2)$ orthogonal system.

Proof. We treat vectors $M \in \mathcal{V}$ as $n \times m$ matrices for simplicity. The proof of Lemma 1 involves various intermediate results.

Definition 1. (Robust Dimension of Projections) Let

$$\dim_i^\tau(\mathcal{V}) = \max_d \text{s.t. } \exists \text{ orthonormal } v_1, \ldots, v_d \in \mathbb{R}^n \text{ and } M_1, \ldots, M_d \in \mathcal{V}$$

$$\text{with } ||M_t|| \leq \tau \text{ and } v_t = M_t(i) \ \forall t \in [d]$$

We try to find many column projections having robust dimension of about δn. We have the

Lemma 2. *For a subspace $\mathcal{V} \subset \mathbb{R}^{p_1 \cdot p_2}$ and for any $\tau \geq \sqrt{p_2}$ we have:*

$$\sum_{i \in [p_2]} \dim_i^\tau(\mathcal{V}) \geq \dim(\mathcal{V})$$

Proof. Writing $d = \dim(\mathcal{V})$ and taking B to be a $(p_1 p_2) \times d$ matrix whose columns are an orthonormal basis of $\mathcal{V}$ we have $\sigma_d(B) = 1$. We form p_2 matrices of dimension p_1 by indexing the rows of B by $[p_1] \times [p_2]$ and taking B_i to be the rows corresponding to indices of the form (j, i) with $j \in [p_1]$. We define $d_i = \max_t \text{s.t. } \sigma_t(B_i) \geq \frac{1}{\sqrt{p_2}}$ We show

$$\sum_{i \in [p_2]} \dim_i^\tau(\mathcal{V}) \geq \sum_{i \in [p_2]} d_i \geq \dim(\mathcal{V})$$

.

Suppose the RHS of the inequality was false. Then define $S_i \subset \mathbb{R}^d$ to be the $d - d_i$-dimensional subspace spanned by the $d - d_i$ rightmost singular vectors of B_i. Since $d - \sum_i d_i > 0$, $\cap_i S_i^\perp \neq \emptyset$, so we can take $\alpha \in \cap_i S_i^\perp$ to be a unit vector. Then for every i we have $||B_i \alpha|| \leq \frac{1}{\sqrt{p_2}}$, but this implies $\sigma_d(B) \leq \sum_i ||B_i \alpha||^2 \leq p_2 \cdot \frac{1}{p_2} = 1$, a contradiction.

For the LHS, we consider the d_i top left-singular vectors of B_i. From a general result for matrices (see lemma B.1 in [6]), they can be represented as $\beta_j B_i$ where $1 \leq j \leq d_i$ and $\beta_j \in \mathbb{R}^d$ satisfies $||\beta_j||_2 \leq \sqrt{p_2}$. Then $\beta_1 B, \ldots, \beta_{d_i} B$ are vectors in $\mathbb{R}^{p_1 p_2}$ that show $\dim_i^\tau(\mathcal{V}) \geq d_i$.

□

We can now construct the matrices M_i

□

V. Conclusion

Smoothed analysis is well-suited to learning problems because learning data are inherently noisy. What's not so clear is the best way to apply these techniques: what exactly should be "smoothed" with perturbations? What should we look at once we've perturbed the inputs? In this survey we've given two examples of smoothed analysis in learning: one in a setting with provable guarantees and the other without. These two examples represent, to us, two different vectors for further research. Many learning heuristics have no good runtime bounds but run efficiently in practice. In these cases, smoothed analysis may be able to bridge the gap, as it has for k-means. Separately, there are several machine learning algorithms that work provably well under certain mild input conditions–finding such algorithms has been a major subject of machine learning research in recent years. Less well-known is whether these input conditions are preserved by the application of noise, and if they are, whether the output of the algorithm is stable. In the case of tensor decomposition, [6] indeed shows that the conditions for uniqueness are preserved by Gaussian perturbation, and that the factors of the perturbed tensor are close to the factors of the original tensor, so the output is consistent. Whether these results translate to other algorithms seems to be a wide-open area for further investigation.

References

[1] Daniel A. Spielman and Shang-Hua Teng. "Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually takes Polynomial Time." *Journal of the ACM, Vol 51 (3)*, pp. 385 - 463, 2004.

[2] Daniel A. Spielman and Shang-Hua Teng. "Smoothed Analysis: An Attempt to Explain the Behavior of Algorithms in Practice" *Communications of the ACM, Vol. 52 No. 10*, pp. 76 - 84, 2009.

[3] Andrea Vattani. "k-means Requires Exponentially Many Iterations Even in the Plane." *Proc. of the 25th ACM Symp. on Computational Geometry (SoCG)*, pp 324-332, 2009.

[4] David Arthur, Bodo Manthey, and Heiko Roeglin. "Smoothed Analysis of the k-means Method." 2010.

[5] Adam Tauman Kalai, Alex Samorodnitsky, and Shang-Hua Teng. "Learning and Smoothed Analysis", *IEEE 54th Annual Symposium on Foundations of Computer Science*, pp. 395 - 404, 2009.

[6] Aditya Bhaskara, Moses Charikar, Ankur Moitra, and Aravindan Vijayaraghavan. "Smoothed Analysis of Tensor Decomposition", *CoRR*, 2013.

[7] J. Kruskal. "Three-way Arrays: Rank and Uniqueness of Trilinear Decompositions", *Linear Algebra and Applications*, 18:95 - 138, 1977.

[8] Rafail Ostrovsky and Yuval Rabani and et al. "The Effectiveness of Lloyd-Type Methods for the k-Means Problem",

[9] G. Milligan. "An examination of the effect of six types of error perturbation on fifteen clustering algorithms," *Psychometrika*, 45:325 - 342, 1980.

[10] Sariel Har-Peled and Bardia Sadri. "How fast is the k-means method?", *Algorithmica*, 41(3):185 - 202, 2005.

[11] Johan Håstad. "Tensor Rank is NP-complete", *Journal of Algorithms* pp. 644 - 654, 1990.

[12] Christopher J. Hillar and Lek-Heng Lim. "Most tensor problems are NP hard" *CoRR*, 2009.