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.

1. 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. [EXAMPLES]. 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]. (FIX): There are two active fronts: some researchers have followed the standard line of smoothed analysis, focusing on algorithm *runtimes*. Others have aimed to show that algorithms still *work* on perturbed instances of problems; this has also been called smoothed analysis, as in [6]. We give examples of both. [explain what 4 and 6 are] 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 3 we discuss at a high level the result of [4], showing that the k-means method has polynomial smoothed runtime. In section 4 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 5 we conclude and offer some speculation about directions for further research.

2. 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.

3. 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 e.g. [?]). 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 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 $1/\sigma$.

Again, note that this theorem gives no guarantees on the optimality of the output. The argument essentially

4. Tensor Decomposition

(FIX): Among other problems, tensor decomposition methods can be used to learn topic models, multi-view mixture models, phylogenetic trees, and detect communities. Tensors are especially useful in learning because under mild conditions, the factors of a rank-k 3-tensor decomposition $T = \sum_{i=1}^{k} a_i \otimes b_i \otimes c_i$ are unique [7]. These conditions extend naturally to higher-order tensors. Assume we are given a l-order tensor T of dimension n (so $T \in \mathbb{R}^{n^{\times l}}$) and of rank R and we want to decompose it into a sum of rank-one tensors, i.e.:

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

We first describe conditions under which such decomposition is unique and computable in polynomial time.

The algorithm above 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.

5. Conclusion

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 324332, 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*, 18:95138, 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*, 45:325342, 1980.