CS369N: Beyond Worst-Case Analysis
Lecture #7: Smoothed Analysis*

Tim Roughgarden†

November 30, 2009

1 Context

This lecture is last on flexible and robust models of "non-worst-case data". The idea is again to assume that there is some "random aspect" to the data, while stopping well short of average-case analysis. Recall our critique of the latter: it encourages overfitting a brittle algorithmic solution to an overly specific data model.

Thus far, we've seen two data models that assume only that there is "sufficient randomness" in the data and make no other commitments.

1. In Lecture #4 we studied semirandom graph models, where nature goes first and randomly plants a solution (like a bisection or a clique), while an adversary goes second and is allowed to perturb nature's choice subject to preserving the planted solution. Recall that in the motivating problems, completely random instances were easy or intuitively meaningless.

2. In Lecture #6 we studied pseudorandom data. Here, an adversary could choose a distribution on the data subject to a lower bound on the "minimum randomness" of the data. In the motivating applications (hashing and related problems), random data was unrealistic and overly easy.

Smoothed analysis has the same flavor, with the nature and an adversary reversing roles relative to a semirandom model: an adversary first picks an input, which is subsequently perturbed randomly by nature. This model inherits the attractive robustness of the above two data models, and has the additional advantage that it has a natural interpretation as a model of "problem formation". To explain, consider a truism that seems downbeat at first: *inaccuracies are inevitable in a problem formulation.* For example, when solving a linear

†Department of Computer Science, Stanford University, 462 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: tim@cs.stanford.edu.

programming problem, there is arguably the "ground truth" version, which is what you would solve if you could perfectly estimate all of the relevant quantities (costs, inventories, future demands, etc.) and then the (probably inaccurate) version that you wind up solving. The latter linear program can be viewed as a perturbed version of the former one. It might be hard to justify any assumptions about the real linear program, but whatever it is, it might be uncontroversial to assume that you instead solve a perturbed version of it. Smoothed analysis is a direct translation of this idea.

2 Basic Definitions

Consider a cost measure $\text{cost}(A, z)$ for algorithms A and inputs z. In most of the smoothed analysis applications so far, cost is the running time of A on z. For a fixed input size n and measure σ of perturbation magnitude, the standard definition of the *smoothed complexity of* A is

$$\sup_{\text{inputs } z \text{ of size } n} \left\{ \mathbf{E}_{r(\sigma)}[\text{cost}(A, z + \sigma r(\sigma))] \right\}, \tag{1}$$

where "$z + r(\sigma)$" denotes the input obtained from z under the perturbation $r(\sigma)$. For example, z could be an element of $[-1, 1]^n$, and $r(\sigma)$ a spherical Gaussian with directional standard deviation σ. Observe that as perturbation size σ goes to 0, this measure tends to worst-case analysis. As the perturbation size goes to infinity, the perturbation dwarfs the original input and the measure tends to average-case analysis (under the perturbation distribution). Thus smoothed complexity is an interpolation between the two extremes of worst- and average-case analysis.

We say that an algorithm has *polynomial smoothed complexity* if its smoothed complexity (1) is a polynomial function of both n and $1/\sigma$. In this case, the algorithm runs in expected polynomial running time as long as the perturbations have magnitude at least inverse polynomial in the input size.

3 Smoothed Analysis of Local Search for the TSP

3.1 The 2-OPT Heuristic

Recall that a local search algorithm for an optimization problem maintains a feasible solution, and iteratively improves that solution via "local moves" as long as is possible, terminating with a locally optimal solution. Local search heuristics are common in practice, in many different application domains. For many such heuristics, the worst-case running time is exponential while the empirical performance on "real-world data" is quite fast. Discrepancies such as this cry out for a theoretical explanation, and provide an opportunity for smoothed analysis.[1]

[1] Another issue with a local search heuristic, that we ignore in this lecture, is that it terminates at a locally optimal solution that could be much worse than a globally optimal one. Here, the gap between theory and

We illustrate the smoothed analysis of local search heuristics with a relatively simple example: the 2-OPT heuristic for the Traveling Salesman Problem (TSP) in the plane with the ℓ_1 metric. The input is n points in the square $[0,1] \times [0,1]$. The goal is to define a tour (i.e., an n-cycle) $v_1, \ldots, v_n$ to minimize

$$\sum_{i=1}^{n} \|v_{i+1} - v_i\|_1, \tag{2}$$

where v_{n+1} is understood to be v_1 and $\|x\|_1$ denotes the ℓ_1 norm $|x_1| + |x_2|$. This problem is NP-hard. It does admit a polynomial-time approximation scheme [**?**], but it is slow and local search is a common way to attack the problem in practice.

We focus on the *2-OPT* local search heuristic. Here, the allowable local moves are swaps: given a tour, one removes two edges, leaving two connected components, and then reconnects the components in the unique way that yields a new tour (Figure **??**). Such a swap is *improving* if it yields a tour with strictly smaller objective function value (2). Since there are only $O(n^2)$ possible swaps from a given tour, each iteration of the algorithm can be implemented in polynomial time. In the worst case, however, this local search algorithm might require an exponential number of iterations before finding a locally optimal solution [**?**]. In contrast, the algorithm has polynomial smoothed complexity.

Theorem 3.1 ([?]) *The smoothed complexity of the 2-OPT heuristic for the TSP in the plane with the ℓ_1 metric is polynomial. In particular, the expected number of iterations is $O(\sigma^{-1} n^6 \log n)$.*[2]

Theorem 3.1 is underspecified until we describe the perturbation model, which happily accommodates a range of distributions. As usual, the adversary goes first and chooses an arbitrary set of n points $p_1, \ldots, p_n$ in $[0,1] \times [0,1]$, which are then randomly perturbed by nature. We assume only that the noise δ_i added to p_i has a distribution given by a density function f_i satisfying $f_i(x) \leq 1/\sigma$ everywhere. In particular, this forces the support of f_i to have area at least σ.[3] We also assume that $p_i + \delta_i$ lies in the square $[0,1] \times [0,1]$ with probability 1.

3.2 The High-Level Plan

Most smoothed analyses have two conceptual parts. First, one identifies a sufficient condition on the data under which the given algorithm has good performance. Second, one shows that the sufficient condition is likely to hold for perturbed data. Generally the sufficient condition and the consequent algorithm performance are parameterized, and this parameter can be

practice is not as embarrassing — on real data, local search algorithms can produce pretty lousy solutions. Generally one invokes a local search algorithm many times (either with random starting points or a more clever preprocessing step) and returns the best of all locally optimal solutions found.

[2] Better upper bounds are possible via a more complicated analysis [**?**].

[3] Cf., the diffuse adversaries of Lecture #2 and the block sources of Lecture #6.

interpreted as a "condition number" of the input.[4] Thus the essence of a smoothed analysis is showing that a perturbed instance is likely to have a "good condition number".

For Theorem 3.1, our sufficient condition is that every swap that ever gets used by the local search algorithm results in a significantly improved tour. Precisely, consider a local move that begins with some tour, removes the edges $(u, v), (x, y)$, and replaces them with the edge $(u, x), (v, y)$ (as in Figure **??**). Note that the decrease in the TSP objective under this swap is

$$\|u - v\|_1 + \|x - y\|_1 - \|u - x\|_1 - \|v - y\|_1, \tag{3}$$

and is independent of what the rest of the tour is. We call the swap ϵ-*bad* if (3) is strictly between 0 and ϵ. We prove Theorem 3.1 by lower bounding the probability that there are no bad ϵ-swaps (as a function of ϵ) and upper bounding the number of iterations when this condition is satisfied. In terms of the previous paragraph, the parameter ϵ is playing the role of a condition number of the input.

3.3 Proof of Theorem 3.1

The key lemma is an upper bound on the probability that there is an ϵ-bad swap.

Lemma 3.2 *For every perturbed instance and $\epsilon > 0$, the probability (over the perturbation) that there is ϵ-bad swap is $O(\epsilon\sigma^{-1}n^4)$.*

Proof: Since there are $O(n^4)$ potential swaps, we can prove a bound of $O(\epsilon/\sigma)$ for a fixed swap and then apply the Union Bound. Fix a swap, say of $(u, v), (x, y)$ with $(u, x), (v, y)$. Each of u, v, x, y is of the form $p_i + \delta_i$ for an adversarially fixed point p_i and a random perturbation δ_i. We prove the probability bound using only the random perturbation of the point y, conditioned on an arbitrary fixed value of u, v, x.

The expression in (3) splits into two separate sums S_1 and S_2, one for each of the coordinates. Call (S_1, S_2) a *bad pair* if $S_1 + S_2 \in (0, \epsilon)$. Since $|S_1|, |S_2| \leq 4$ — each of u, v, x, y lies in the square — the Lebesgue measure (i.e., area) of the set of bad pairs is $O(\epsilon)$.

With u, v, x fixed, consider varying the first coordinate y_1 of y between 0 and 1. The sum S_1 takes on a given value for at most three distinct values of y_1 (depending on whether y_1 is less than both v_1 and x_1, greater than both of them, or strictly in between them), and similarly for S_2 and y_2. Thus any given pair (S_1, S_2) arises from at most 9 different choices of y. Thus the Lebesgue measure of the set of values of y that result in a bad pair is $O(\epsilon)$. Since in our perturbation model the density of y at any given point is at most $1/\sigma$, the probability that y takes on a value resulting in a bad pair is $O(\epsilon/\sigma)$. ■

We now show that Lemma 3.2 implies Theorem 3.1. First, note that since all points lie in the square $[0, 1] \times [0, 1]$, every tour has length (2) at most $2n$ (and at least 0). Thus, if there are no ϵ-bad swaps, then the local search algorithm must terminate within $\frac{2n}{\epsilon}$ iterations.

[4] In numerical analysis, the condition number of a problem has a very precise meaning, which is roughly the worst-case fluctuation in the output as a function of perturbations to the input. Here we use the term much more loosely.

Also, the worst-case number of iterations of the algorithm is at most the number of different tours, which is upper bounded by $n!$. Using these observations and Lemma 3.2, we have

$$\begin{aligned}
\mathbf{E}[\#\text{ of iterations}] &= \sum_{M=1}^{N!} \mathbf{Pr}[\#\text{ of iterations} \geq M] \\
&\leq \sum_{M=1}^{N!} \mathbf{Pr}\left[\text{there is a } \frac{2n}{M}\text{-bad swap}\right] \\
&\leq \sum_{M=1}^{N!} O\left(\frac{n^5}{M\sigma}\right) \\
&= O\left(\sigma^{-1} n^6 \log n\right),
\end{aligned}$$

where the final step uses the estimate $\ln(n!) = O(n \log n)$ for the harmonic series $1 + \frac{1}{2} + \cdots + \frac{1}{n!}$.

4 Smoothed Analysis of the Perceptron Algorithm

4.1 Smoothed Analysis of the Simplex Method

Recall the *simplex algorithm* for linear programming, which has exponential worst-case running time but is almost always very fast in practice. The "killer application" of smoothed analysis is the result that the simplex algorithm has polynomial smoothed complexity [**?**]. Recall that an implementation of the simplex method requires a choice of a pivot rule — which dictates which polytope vertex to go to next, when there are multiple adjacent vertices with better objective function values — and the result of Spielman and Teng [**?**] concerns the "shadow pivot rule" [**?**]. This is the same pivot rule that was analyzed in the average-case analyses of the simplex method in the 1980s (e.g. [**?**]). The idea is to project the high-dimensional feasible region onto a plane (the "shadow"), where running the simplex algorithm is easy. Every vertex in the projection is a vertex of the original polytope, though some of the former's vertices will be sucked into the interior of the shadow. It is non-trivial but feasible to project in a way that ensures that the optimal solution appears as a vertex in the shadow. In terms of the high-level plan in Section 3.2, a sufficient condition for a polynomial running time is the that the number of vertices of the shadow is only polynomial. The hard work in [**?**] is to prove that this sufficient condition holds for perturbed inputs.

4.2 From Linear Programming to Containing Halfspaces

The original analysis of Spielman and Teng [**?**] has been improved and simplified [**?**, **?**] but remains too technical to cover here. We instead discuss a nice but weaker guarantee of Blum and Dunagan [**?**] for a different algorithm that can be used to solve linear programs, called the *perceptron algorithm*. To motivate the algorithm, we discuss how linear programming reduces to finding a single halfspace that contains a given set of points on the sphere.

Consider a linear program of the form

$$\max c^T y$$

subject to

$$Ay \leq b.$$

We consider only the feasibility version of this linear program, meaning for a target objective function value c^* we search for a feasible solution to the linear system

$$\begin{aligned} c^T y &\geq c^* \\ Ay &\leq b. \end{aligned}$$

These two problems are polynomial-time equivalent in the standard sense (just binary search over all the choices for c^*), but this reduction isn't really kosher in the context of smoothed complexity — since the constraint $c^T y \geq c^*$ is being imposed by an outer algorithm loop, there's no justification for perturbing it.

Given a linear system as above, it can be reduced to solving a linear system of the form $Xw < 0$, where X is a constraint matrix and w is the vector of decision variables, with the additional constraint that each row x_i lies on the sphere ($\|x\|_2 = 1$). The idea of the reduction is to introduce a dummy variable set to 1 in order to "homogenize" the right-hand side (c^*, b) and then rescale all the constraints; we leave the details as an exercise.[5] In other words, we seek a vector w such that $w \cdot x_i < 0$ for every i, which is the same as seeking a halfspace (for which w is the normal vector) that contains all of the x_i's. For the rest of this lecture, we focus on this version of the problem.

4.3 The Perceptron Algorithm

The perceptron algorithm is traditionally used for classification. The input is n points in $\mathcal{R}^d$, rescaled to lie on the d-sphere, and a *label* $b_i \in \{-1, +1\}$ for each. The goal is to find a halfspace that puts all of the "+1 points" on one side and the "-1 points" on the other side (Figure **??**). This "halfspace separation" problem is equivalent to the "halfspace containment" problem derived in the previous section (just replace the "+1 points" with the corresponding antipodal points). The algorithm itself is extremely simple. The motivation for the mail step is that it makes the candidate solution w "more correct" on x_i, by increasing $w_t \cdot x_i$ by $b_i x_i \cdot x_i = \|x_i\|_2^2 = 1$. Of course, this update could screw up the classification of other x_i's, and we need to prove that the procedure eventually terminates.

Theorem 4.1 ([?]) *If there is a unit vector w^* such that $sgn(w^* \cdot x_i) = b_i$ for every i, then the Perceptron algorithm terminates with a feasible solution w_t in at most $1/gamma^2$ iterations, where*

$$\gamma = \min_i |w^* \cdot x_i|. \tag{4}$$

[5] In fact, this introduces an extra technical constraint that the dummy variable has to be non-zero. We ignore this detail throughout the lecture but it is not hard to accommodate, see Blum and Dunagan [**?**].

Input: n points on the sphere $x_1, \ldots, x_n$ with labels $b_1, \ldots, b_n \in \{-1, +1\}$.

1. Initialize t to 1 and w_1 to the all-zero vector.
2. While there is a point x_i such that $\text{sgn}(w \cdot x_i) \neq b_i$, increment t and set $w_t = w_{t-1} + b_i x_i$.

Figure 1: The Perceptron Algorithm.

Geometrically, the parameter γ is the cosine of the smallest angle that a point x_i with the separating halfspace defined by w^* (Figure **??**).[6] In classification problems, this is often called the *margin*. It can be viewed as a condition number in the sense discussed in Section 3.2. Since Theorem 4.1 applies to every feasible unit vector w^*, one typically thinks of w^* as the solution that maximizes the margin γ and gives the tightest upper bound.

Obviously, the bound in Theorem 4.1 is not so good when the margin γ is tiny, and it is known that the perceptron algorithm can require an exponential number of iterations to converge. The hope is that perturbed instances have reasonably large margins, in which case Theorem 4.1 will imply a good upper bound on the smoothed complexity of the perceptron algorithm.

Proof of Theorem 4.1: We first claim an upper bound on the rate of growth of the norm of the vector w: for every iteration t,

$$\|w_{t+1}\|^2 \leq \|w_t\|^2 + 1. \tag{5}$$

For the proof, let x_i be the point chosen in iteration t and write

$$\|w_{t+1}\|^2 = \|w_t + b_i x_i\|^2 = \|w_t\|^2 + \|x_1\|^2 + 2b_i(w_t \cdot x_i) \leq \|w_t\|^2 + 1,$$

where in the inequality we use the facts that x_i lies on the sphere and that $b_i(w_t \cdot x_i) \leq 0$ (since $\text{sgn}(w_t \cdot x_i) \neq b_i$).

Second, we claim a lower bound on the rate of growth of the projection of the vector w onto the assumed feasible solution w^*: for every t,

$$w_{t+1} \cdot w^* \geq w_t \cdot w^* + \gamma, \tag{6}$$

where γ is defined as in (4). For the proof, let x_i be the point chosen in iteration t and write

$$w_{t+1} \cdot w^* = (w_t + b_i x_i) \cdot w^* = w_t \cdot w^* + b_i(x_i \cdot w^*) \geq w_t + \gamma,$$

where in the inequality we use the definition of γ and the fact that $\text{sgn}(x_i \cdot w^*) = b_i$.

Inequalities (5) and (6) imply that, after t iterations,

$$\sqrt{t} \geq \|w_{t+1}\| = \|w_{t+1}\|\|w^*\| \geq w_{t+1} \cdot w^* \geq t\gamma;$$

recall that w^* is a unit vector. This implies that the iteration count t never exceeds $1/\gamma^2$. ■

[6] This is also the formal definition of γ when the x_i's are not unit vectors, as will be the case in Section 4.4. This corresponds to normalizing (4) by $\|x_i\|$.

4.4 The Condition Number of Perturbed Instances

We now outline the argument why perturbed instances have large margins, and hence the perceptron algorithm has good smoothed complexity. The proof essentially boils down to some basic properties of high-dimensional Gaussians, as previously seen in Lecture #4 when we discussed learning mixtures of Gaussians. We will be similarly hand-wavey in this lecture, though all of the intuitions are accurate and can be made precise, with some work.

An annoying technical issue that pervades smoothed and average-case analyses of linear programming is dealing with infeasible inputs. The polynomial smoothed complexity of the simplex method [**?**] is valid regardless of whether or not a perturbation results in a feasible instance (with infeasible instances detected as such). As stated, the perceptron algorithm will not converge on an infeasible instance and so we cannot hope for an analogously strong result for it. We avoid this issue by defining a perturbation model that only yields feasible instances.

Precisely, we first allow an adversary to choose points $x_1, \ldots, x_n$ on the sphere in $\mathcal{R}^d$ and a unit vector w^*. Then, nature adds a random perturbation to each x_i. We assume that each perturbation is a spherical Gaussian with directional standard deviation σ, although other distributions will also work fine.[7] Finally, we *define* the label b_i of the perturbed version $\hat{x}_i$ of x_i as $\text{sgn}(w^* \cdot \hat{x}_i)$. That is, we label the $\hat{x}_i$'s so that w^* is a feasible solution.

What is the margin (4) of such a perturbed instance? For convenience, we assume that the dimension d is large (polynomial in n) so that we can rely on concentration bounds for high-dimensional Gaussians (although the results hold in any dimension). We also assume that σ is small (at most $1/d$, say). Now consider a single perturbed point $\hat{x}_i = x_i + \delta_i$. We want to lower bound the magnitude of the cosine of the angle between $\hat{x}_i$ and w^*, which is

$$\frac{|w^* \cdot (x_i + \delta_i)|}{\|x_i + \delta_i\|}. \tag{7}$$

The denominator is close to 1 as long as the directional standard deviation σ is small. In more detail, we have

$$\|x_i + \delta_i\|^2 = \underbrace{\|x_i\|^2}_{=1} + \|\delta_i\|^2 + 2(x_i \cdot \delta_i). \tag{8}$$

Recall from Lecture #4 (Counterintuitive Fact #1) that high-dimensional spherical Gaussians are well approximated by the uniform distribution on a suitable sphere; here, this translates to $\|\delta_i\|^2$ being sharply concentrated around $d\sigma^2$. Also, $x_i \cdot \delta_i$ — the projection length of the spherical Gaussian δ_i along the fixed unit vector (direction) x_i — is distributed like a one-dimensional Gaussian with zero mean and standard deviation σ. The magnitude of this projection length is thus concentrated around σ. Thus, the expression (8) is very close to 1 with high probability. By the same reasoning, in the numerator of (7), $|w^* \cdot \delta_i|$ is concentrated around σ. Finally, the perturbed margin (7) is minimized when $w^* \cdot x_i = 0$ (this is intuitive but a little technical to make rigorous).

[7] This perturbation knocks the points off of the sphere, but not by much if σ is small. We will be thinking of σ as much less than $1/\sqrt{d}$, where d is number of dimensions.

Summarizing, the denominator in (7) lies in $[\frac{1}{2}, 2]$ (say) with very high probability, and in the worst case the numerator in (7) is distributed like the absolute value of a Gaussian with standard deviation σ. It follows that, for every $\epsilon > 0$, the probability that (7) is more than ϵ is $O(\epsilon/\sigma)$. By a Union Bound, the probability that (4) is more than ϵ for any point x_i — i.e., the probability that the margin (4) is more than ϵ — is $O(n\epsilon/\sigma)$. Applying Theorem 4.1 shows that the perceptron algorithm has good smoothed complexity in the following sense.

Theorem 4.2 ([?]) *For every $\epsilon > 0$, the perceptron algorithm halts on a perturbed instance within*

$$O\left(\frac{n^2}{\sigma^2\epsilon^2}\right)$$

iterations with probability at least $1 - \epsilon$.

Unlike Theorem 3.1 (and also unlike the analysis of the simplex method in [**?**]), the quadratic dependence on $\frac{1}{\epsilon}$ in the guarantee of Theorem 4.2 precludes concluding that the expected running time of the perceptron algorithm is polynomial on a perturbed instance. (Mimicking the proof of Theorem 3.1 yields a bound proportional to the square root of the worst-case number of iterations.) Thus Theorem 4.2 shows only the weaker but still interesting fact that the perceptron algorithm has polynomial smoothed complexity with high probability.