Sum-of-Squares Optimization

Akilesh Tangella

1 Introduction

Polynomial optimization is a fundamental task in mathematics and computer science. Such tasks rose to popularity with the advent of linear and semidefinite programming. Research on these topics has led to beautiful theoretical results, such as the simplex algorithm, duality, the ellipsoid algorithm, etc. A number of applications also exist in a variety of fields, such as operations research, combinatorial optimization, etc [JM07].

Though linear and semidefinite programming are powerful, they are limited. In both cases, the objective function being optimized is linear, and thus, polynomials with higher degree cannot be directly dealt with. Thus, other tools and ideas have been developed to deal with such cases. At a high level, two paradigms exist for polynomial optimization. The first is local search, where one starts with some solution and then iteratively improves it until a local minimum or maximum is reached. Gradient descent, a ubiquitous tool in machine learning, is an example of such a process. Although local search is useful in many settings, problems emerge with it. Particularly, it can get stuck at local minima or maxima, and never obtain the globally optimal solution [BS16].

The other paradigm, which Sum-of-Squares (SOS) optimization follows, takes a global approach, exploiting the structure of the polynomial being optimized. At the heart of SOS optimization lies a connection between three seemingly disparate tasks/objects: 1) checking whether a polynomial is nonnegative over a given domain (this leads to what is known as the SOS proof system), 2) semidefinite programming, and 3) probability distribution-like entities known as pseudodistributions. In this exposition, we make the details of this connection clear and show how it is useful in designing algorithms [BS16].

From a broader perspective, SOS optimization inspires new ideas for the algorithm design process. Particularly, in the current state of affairs, each algorithmic problem usually requires a creative, tailor-made solution. However, since polynomial optimization encapsulates many problems in theoretical computer science, SOS optimization can be viewed as a step toward a general framework for algorithm design. Lastly, SOS introduces the novel idea of turning proofs into algorithms. We see what is meant by this in section 4.

2 Convex Programming and Relaxations

In this section we discuss the concept of a relaxation in linear and semidefinite programming. The general ideas presented here are important. Before beginning we note one convention we will use throughout the rest of the paper: approximation factors for minimization problems will be greater than or equal to 1 and for maximization problems will be less than or equal to 1.

2.1 Linear Programming

Definition 1 (Linear Program) *A linear program (LP) is an optimization problem of the following form:*

Input: *A matrix $A \in \mathbb{R}^{m \times n}$, an vector $b \in \mathbb{R}^m$, and a vector $c \in \mathbb{R}^n$.*

Goal: *Find a vector $x \in \mathbb{R}^n$ to minimize (maximize) $c^T x$ subject to $Ax \geq (\leq) b$.*

The inequality between vectors above is termwise in its components. Notice that the objective function and the constraints are all linear, hence the name linear programming. If the entries of A, b, and c are rational and can be expressed with l bits of space, then LP can be solved in poly(n, m, l) time via the ellipsoid algorithm. When the entries of x are restricted to be integers, and in particular, 0 or 1, LP encapsulates many combinatorial optimization problems. This version of LP is called integer linear programming (ILP). ILP is NP-complete, however, as we will see in the next section, this is no reason to completely lose hope [Kha16].

2.2 LP Relaxations

A paradigm for designing approximation algorithms for problems that can be expressed as ILPs is as follows:

- Cast the problem as an ILP.
- Drop the constraint that the solutions should only be integer-valued vectors to obtain an LP (this process is known as the LP relaxation of the problem).
- Solve the LP, which gives a fractional-valued vector as its solution. Note that since the feasible region for the LP is a superset of the feasible region for the ILP, the solution that the LP obtains is at least as good as the optimal ILP solution.
- Round the LP solution to an integer solution and show that the objective function does not change by more than a factor of α. This gives an α-approximation algorithm.

We now design approximation algorithms for two famous combinatorial optimization problems: vertex cover and set cover. In the former, a deterministic rounding procedure allows for a 2-approximation, while in the latter, a randomized rounding procedure is used to obtain a useful approximation [Kha16].

2.2.1 Vertex Cover

Recall that a vertex cover of a graph $G = (V, E)$ is a subset of vertices such that for any edge in the graph, at least one of its incident vertices is in the subset. The vertex cover problem is to output a vertex cover of minimum cardinality. Letting x representing a vector where each variable corresponds to a vertex in the graph, we can express vertex cover by the following ILP:

Minimize: $\sum_{i \in V} x_i$.

Subject to:

- $\forall (i, j) \in E$, $x_i + x_j \geq 1$
- $\forall i \in V, x_i \in \{0, 1\}$

Let's briefly discuss what this ILP "means." For each vertex i, x_i is assigned 1 if it is part of the vertex cover, thus, the objective function is the size of the vertex cover. The first constraint indicates that for each edge, at least one of the vertices should be labelled 1 (be part of the vertex cover), and the second constraint enforces that the solutions should be 0/1 vectors. So indeed, a solution to this ILP gives the minimum cardinality vertex cover.

We relax the ILP to an LP by removing the integrality constraints. Now, we must convert the fractional solution of this LP to a feasible 0/1 vector. Let the fractional solution vector be x^*. The rounding procedure is: for each vertex $i \in V$ if $x_i^* \geq 1/2$ in the LP solution set x_i to 1, otherwise set it to 0.

Claim 2 *The above rounding procedure gives a 2-approximation for vertex cover.*

Proof We show two things. First, that the rounding procedure results in a valid vertex cover, and second that it is a 2-approximation. To see that the rounding procedure results in a valid vertex cover, consider any edge $(i, j) \in E$. In the fractional solution, $x_i^* + x_j^* \geq 1$, thus we must have either x_i^* or x_j^* is greater than or equal to 1/2. This means at least one of x_i or x_j is 1, so $x_i + x_j \geq 1$ in the rounded vector as well. To see that the rounding procedure results in a 2-approximation, note that the cost only blows up when we round up to 1, and we only round values greater than or equal to 1/2 up to 1, so no term in the solution vector gets increased by over a factor of 2 [Kha16]. ■

Before moving on, we discuss the important idea of an integrality gap. The integrality gap is the worst case gap between the cost of an optimal LP solution and an optimal ILP solution. One can intuitively think of the integrality gap as a lower bound on the approximation factor that can be obtained via LP relaxation for a minimization problem (consequently, one sanity check is to make sure the integrality gap is greater than or equal to 1).

Claim 3 *The integrality gap for vertex cover is 2.*

Proof Consider the complete graph on n vertices. The minimum vertex cover is of size $n - 1$. This is because if we had a vertex cover of cardinality less than $n - 1$, then we have at least two vertices not in the vertex cover so the edge between them is uncovered. On the

other hand, an optimal LP solution is $x_i = 1/2$ for all $i \in V$, which gives an objective function value of $n/2$. Thus, the blowup factor of an LP relaxation on this instance must be at least 2 [Kha16]. ■

2.2.2 Set Cover

In the previous section, the rounding procedure is deterministic. That is each fractional solution maps to exactly one rounded solution. In this section, we find the need for randomized rounding procedures. Recall the set cover problem. We are given a universe of n elements, $U = \{1, 2, ..., n\}$ and a collection of m subsets of U, S_1, S_2,...,S_m. The goal is to find the smallest collection of these subsets whose union is U. We can cast set cover as an ILP as follows, where each component in the vector x corresponds to a subset in the collection:

Minimize: $\sum_{j=1}^{m} x_{S_j}$

Subject to:

- $\forall i \in U, \sum_{S_j : i \in S_j} x_{S_j} \geq 1$
- $\forall j \in \{1, 2, ..., m\}, x_{S_j} \in \{0, 1\}$

Again, let's briefly discuss what this ILP "means." Setting a component in x to 1 means we include the corresponding subset in our set cover. The integrality of x is enforced in the second constraint. The first constraint enforces the fact that for each element in the universe, at least one subset in the set cover contains it [Moi16].

We relax the ILP to an LP by removing the integrality constraints. Let x^* be the fractional solution. Let's informally see what goes wrong with a similar deterministic rounding procedure to the previous section: suppose every component of x^* that is less than 1/2 is set to 0 and otherwise set to 1. Consider the following instance: $U = \{1, 2, 3, 4\}$, $S_1 = \{2, 3, 4\}$, $S_2 = \{1, 3, 4\}$, $S_3 = \{1, 2, 4\}$, $S_4 = \{1, 2, 3\}$. The optimal fractional solution sets $x^*_{S_1} = x^*_{S_2} = x^*_{S_3} = x^*_{S_4} = 1/3$, which means x is the 0 vector. Thus, no guarantees can be given on the approximation factor.

Instead, consider the following randomized rounding procedure. For each $x^*_{S_j}$, $j \in \{1, 2, ..., m\}$, in the fractional solution set x_{S_j} to 1 with probability $\alpha x^*_{S_j}$ and to 0 with probability $1 - \alpha x^*_{S_j}$. We will specify α later. What is the probability that the rounded solution forms a set cover? For each $i \in U$, let $Y_i = \sum_{S_j : i \in S_j} x_{S_j}$. Note that $\mathbb{E}[Y_i] \geq \alpha$. We have for each element $i \in U$:

$\Pr[i \text{ is uncovered}] =$
$\Pr[\text{None of the subsets containing } i \text{ has its corresponding solution vector term rounded to 1}] =$

$$\Pr[Y_i < 1] = \Pr\left[Y_i < \mathbb{E}[Y_i] \cdot \frac{1}{\mathbb{E}[Y_i]}\right] \leq \exp\left(-\frac{\mathbb{E}[Y_i]}{2}(1 - 1/\mathbb{E}[Y_i])^2\right) =$$

$$\exp\left(-\frac{(\mathbb{E}[Y_i] - 1)^2)}{2\,\mathbb{E}[Y_i]}\right) \leq \exp\left(\frac{-\mathbb{E}[Y_i]}{8}\right) \leq e^{-\alpha/8}$$

The first inequality is due to the multiplicative Chernoff bound and the last inequality is true whenever $\alpha \geq 2$, which we will set it to be. Note that when $\alpha = 16 \ln n$, we have that the probability i is left uncovered is upper bounded by $1/n^2$, so by the union bound, the probability that all the elements get covered is lower bounded by $1 - 1/n$. Now, what is the approximation factor achieved by this algorithm? We can analyze this in expectation. We have: $\mathbb{E}[\sum_{j=1}^{m} x_{S_j}] = \alpha \sum_{j=1}^{m} x^*_{S_j}$, so the expected value of the rounded solution is at most α times greater than the optimal ILP solution. Finally, we also have the following fact, which we state without proof [Moi16].

Fact 4 *The integrality gap for the set cover problem is* $\Omega(\log n)$ *[Kha16].*

2.3 Semidefinite Programming

Definition 5 (Positive Semidefinite (PSD) Matrix) *A matrix* $X \in \mathbb{R}^{n \times n}$ *is positive semidefinite (PSD) (denoted* $X \succeq 0$*) if it satisfies any of the following (equivalent) properties:*

- $\forall a \in \mathbb{R}^n$, $a^T X a \geq 0$.
- $X = B^T B$ *for some matrix* $B \in \mathbb{R}^{n \times n}$*. This is known as the Cholesky decomposition.*
- *All of* X*'s eigenvalues are nonnegative.*

Definition 6 (Semidefinite Program) *A semidefinite program (SDP) is an optimization problem of the following form:*

Input: *Matrices* $C, A_i \in \mathbb{R}^{n \times n}$ *and vector* $b \in \mathbb{R}^m$ *where* $i \in \{1, 2, .., m\}$.

Goal: *Find an* $X \in \mathbb{R}^{n \times n}$ *to minimize* $\langle C, X \rangle = \sum_{1 \leq i,j \leq n} c_{ij} x_{ij}$ *such that* $\langle A_i, X \rangle = b_i$ *for each* $i \in \{1, 2, .., m\}$ *and* $X \succeq 0$.

Note that SDPs encapsulate LPs, since LPs consist of the case when the matrix X is diagonal. Thus, SDPs can solve a broader class of problems than LPs. Via the ellipsoid method SDPs can be solved to arbitrarily (but not 0) accuracy in time polynomial in the dimension of the matrix and polylogarithmic in the inverse of the accuracy desired [Moi16].

3 SDP Relaxation for MAXCUT

Recall the MAXCUT problem. Given a graph $G = (V, E)$, we can bipartition the vertices by labelling some vertices as 1 and others as -1. The resulting number of edges which have exactly one incident vertex labelled 1 and another labelled -1 is called the cut size induced by the bipartition. The goal of the MAXCUT problem is to find the bipartition which induces a cut of maximum size.

There is a simple 1/2-approximation algorithm for MAXCUT. For each vertex, label it 1 with probability 1/2 and -1 with probability 1/2. Output the resulting bipartition. Let's calculate the expected value of the cut size of the resulting algorithm. An edge $(i, j) \in E$ in the graph is part of the cut either if i is labelled 1 and j is labelled -1 (happens with probability 1/4) or if i

is labelled -1 and j is labelled 1 (happens with probability 1/4). Thus, the expected cut size is $|E|/2$. Via the method of conditional expectations, this algorithm can be derandomized and a cut size that is within 1/2 of the maximum cut size can be obtained with probability 1. Can we increase the approximation factor beyond 1/2? Inspired by the previous section on LPs, we can try to cast MAXCUT as an ILP. Letting the components of z correspond to edges and the components of x correspond to vertices, the ILP for MAXCUT is:

Maximize: $\sum_{(i,j)\in E} z_{(i,j)}$.

Subject to:

- $\forall (u, v) \in E, z_{(u,v)} \leq \frac{1+x_u}{2} + \frac{1+x_v}{2}$
- $\forall (u, v) \in E, z_{(u,v)} \leq \frac{1-x_u}{2} + \frac{1-x_v}{2}$
- $\forall (u, v) \in E, z_{(u,v)} \in \{0, 1\}$
- $\forall v \in V, x_v \in \{-1, 1\}$

We leave it to the reader to confirm that this ILP corresponds to the MAXCUT problem. If we relax the integrality constraints on z and x, an optimal fractional solution to the LP is $\forall (u, v) \in E, z_{(u,v)} = 1$ and $\forall v \in V, x_v = 0$. This gives the largest possible MAXCUT of $|E|$. However, in a complete graph the MAXCUT is about size $|E|/2$ (put half the edges in one partition and the other half in another partition), which means LP relaxation cannot guarantee an approximation factor better than 1/2. So, what to do? Consider the following SDP, where $X \in \mathbb{R}^{|V|\times|V|}$ (note that this program does indeed align with definition 6 of an SDP):

Maximize: $\sum_{(u,v)\in E} \frac{1}{2} - \frac{1}{2} X_{uv}$.

Subject to:

- $\forall v \in V, X_{vv} = 1$
- $X \succeq 0$.

This SDP is a relaxation for MAXCUT. In what way? Well suppose we restrict X to be of the form xx^T, where $x \in \mathbb{R}^{|V|}$. Suppose even further that we restrict the entries of x are $+1/-1$. Then, the SDP would exactly output the MAXCUT, with the entries in x specifying whether to label a vertex 1 or -1. Restricting the SDP in this way is analogous to putting integrality constraints on an LP. Solving this restricted SDP efficiently is unlikely, because MAXCUT is NP-hard. So, we need to use the solution to the relaxed SDP and round it to a solution for MAXCUT.

Let X be the matrix output by the relaxed SDP. Since X is PSD we can write it as $Y^T Y$ for some matrix $Y \in \mathbb{R}^{|V|\times|V|}$. Similar to X, thinking of the columns of Y as corresponding to vertices and letting y_v be the column corresponding to vertex v, we have that $X_{uv} = \langle y_u, y_v \rangle$. Note that the columns of Y are unit vectors since all the diagonal entries of X are 1. From this, how do we get a rounded vector x from the last paragraph. Choose a random unit vector $a \in \mathbb{R}^{|V|}$ and let $x_v = \text{sign}(\langle a, y_v \rangle)$ for each $v \in V$.

Theorem 7 (SDP Aprroximation for MAXCUT) *The above rounding procedure gives a*

$$\min_{0\le\theta\le\pi} \frac{2\theta}{\pi(1-\cos\theta)} \approx 0.879$$

-approximation to MAXCUT in expectation.

Proof We must compare the following two quantities: 1) the contribution of each edge in the relaxed SDP to the objective function and 2) the expected contribution of each edge in the rounded solution to the objective function. In the relaxed SDP, the contribution of edge (u, v) is:

$$\frac{1}{2} - \frac{1}{2}X_{uv} = \frac{1}{2} - \frac{1}{2}\langle y_u, y_v\rangle = \frac{1}{2} - \frac{\cos\theta}{2}$$

where θ is the angle between vectors y_u and y_v. Without loss of generality we take θ to be $0 \le \theta \le \pi$. The last inequality uses the fact that the inner product of unit vectors is the cosine of the angle between them. To obtain the expected contribution of edge (u, v) in the rounded solution, we calculate the probability that edge (u, v) is part of the cut, which is equivalent to calculating the probability sign(a, y_u) and sign(a, y_v) differ. This happens if and only if the orthogonal hyperplane to a lies between y_u and y_v. The probability of this occurring is $\frac{\theta}{\pi}$ since a is chosen uniformly. So, the expected contribution of an edge is $\frac{\theta}{\pi}$. Thus, the worst case approximation ratio is:

$$\min_{0\le\theta\le\pi} \frac{\frac{\theta}{\pi}}{\frac{1}{2} - \frac{\cos\theta}{2}} = \min_{0\le\theta\le\pi} \frac{2\theta}{\pi(1-\cos\theta)} \approx 0.879$$

[GW94] ■

4 Sum-of-Squares (SOS) Fundamentals

4.1 Generalizing SDP Relaxations

We begin right where we left off: the MAXCUT problem. Interestingly, MAXCUT can be phrased as a polynomial optimization problem over the hypercube:

$\max_{\{-1,1\}^{|V|}} \frac{1-x_i x_j}{2}$

Subject to: $g_v(x) = x_v^2 - 1 = 0$ for all $v \in V$.

Clearly, since MAXCUT is NP-Hard, this polynomial optimization is difficult. Let's think for a moment what relaxations truly do: they take a problem over a restricted domain which is difficult to solve and enlarge the search space. They then solve the problem in the enlarged search space and map it usefully to a solution in the restricted domain. We can think of the SDP relaxation for MAXCUT as a relaxation for the above polynomial optimization problem in the following way. First, we replace the monomial $x_i x_j$ with the variable X_{ij} and we remove the integrality constraints on the X_{ij}'s. The result is the relaxed SDP for MAXCUT in the previous section [Sch16].

Definition 8 (Polynomial Optimization Problem) *A polynomial optimization problem Q over a domain D (commonly the boolean hypercube or $\mathbb{R}^n$ for some positive integer n) is an optimization problem of the following form:*

Input: *A polynomial $p : D \longrightarrow \mathbb{R}$ and polynomials $g_i : D \longrightarrow \mathbb{R}$, $\forall i \in \{1, 2, ..., m\}$.*

Goal: *Maximize $p(x)$ over $x \in D$ such that $g_i(x) = 0$, $\forall i \in \{1, 2, ..., m\}$.*

Can the above technique for relaxing polynomial optimization problems to SDPs be extended? This question lies at the heart of SOS optimization and as we will see, the answer is indeed yes. Assume the maximum degree of any polynomial $p, g_1, g_2, ..., g_m$ is at most $2d$. Similar to the MAXCUT relaxation, we replace each monomial $\prod_{i \in S} x_i$ with an SDP variable X_S for each ordered multiset S such that $|S| \leq 2d$. We let $X_\emptyset = 1$. These variables are then arranged into a matrix X with each row and column indexed by SDP variables corresponding to monomials of degree at most d. We then let each matrix entry correspond to an SDP variable: if R is the multiset indexing the row and C is the multiset indexing the column then the corresponding entry is $X_{R \cup C}$. Consequently, every SDP variable for multisets with cardinality at most $2d$ will be included in X. Note by basic combinatorics that $X's$ dimension is $(n+1)^d \times (n+1)^d$, where n is the length of the vectors in the domain we are optimizing over. The matrix X for $d = 2$ is shown below with the corresponding monomials in parentheses (assume there are two variables x_1 and x_2, in other words $n = 2$):

$$\begin{array}{c} \\ \emptyset \\ \{1\}(x_1) \\ \{2\}(x_2) \end{array} \begin{array}{c} \begin{array}{ccc} \emptyset & \{1\}(x_1) & \{2\}(x_2) \end{array} \\ \begin{pmatrix} X_\emptyset(1) & X_{\{1\}}(x_1) & X_{\{2\}} \\ X_{\{1\}}(x_1) & X_{\{1,1\}}(x_1^2) & X_{\{1,2\}}(x_1 x_2) \\ X_{\{2\}}(x_2) & X_{\{2,1\}}(x_1 x_2) & X_{\{2,2\}}(x_2^2) \end{pmatrix} \end{array}$$

Before solving for X (and taking the hint that we are doing an SDP relaxation) we set the following constraints:

- $X_\emptyset = 1$ (Normalization Property)
- For multisets S, T, U, V, $X_{S \cup T} = X_{U \cup V}$. This is a natural constraint because $X_{S \cup T}$ corresponds to the monomial that results when the monomials associated with X_S and X_T are multiplied (similarly true for $X_{U \cup V}$). (Commutativity or Symmetry Property).
- $X \succeq 0$. This is a natural constraint due to claim 10 below. (PSD Property)

Definition 9 (Kronecker Product) *The Kronecker product of an $m \times n$ matrix A and a $j \times k$ matrix B is the $mj \times nk$ matrix denoted $A \otimes B$. Its entries can be indexed by pairs such that the $(a, b), (c, d)$-th entry is $A_{ac} B_{bd}$ where $1 \leq a \leq m$, $1 \leq b \leq n$, $1 \leq c \leq j$, and $1 \leq d \leq k$. The Kronecker product is the generalization of the vector outer product. The d-th Kronecker power of a matrix A results from taking the Kronecker product of A with itself d times and is denoted $A^{\otimes d}$.*

Example 1 $\left([1, x]^T\right)^{\otimes 2} = [1, x, x, x^2]^T$. *Note the dimensions match up, the 2^{nd} Kronecker product of a 1×2 vector should be a 1×4 vector.*

Claim 10 *For any $y \in \mathbb{R}^n$ the matrix X_y that results when we set $X_S = \prod_{i \in S} y_i$ for X's entries is PSD.*

Proof Let $\hat{y}^T = [1 y^T]$. Then note that $X_y = \hat{y}^{\otimes d}(\hat{y}^{\otimes d})^T$. Noting that the inner product of two vectors a and b is $\langle a, b \rangle = a^T b$, we have for any vector $v \in \mathbb{R}^n$ that $v^T X_y v = \left\langle v, \hat{y}^{\otimes d} \right\rangle^2 \geq 0$. Therefore, by definition 5, X_y is PSD. ■

Any feasible solution $y \in \mathbb{R}^n$ to Q yields a feasible matrix X, by simply setting $X_S = \prod_{i \in S} y_i$ for X's entries. Our description of the SDP relaxation still requires some formalization provided in the definition below.

Definition 11 (Sum-of-Squares Relaxation at Degree $2d$) *Given a polynomial optimization problem Q with $\deg(p) \leq 2d$ and $\deg(g_i) \leq 2d$ for all $i \in \{1, 2, .., m\}$, the degree-$2d$ sum-of-squares relaxation for Q is denoted as $sos_d(Q)$. A variable X_S is defined for each unordered multiset S with elements in $\{1, 2, .., n\}$ and size less than or equal to $2d$. These variables are arranged in a $(n+1)^d \times (n+1)^d$ matrix X. X's rows and columns are indexed by the multisets and the entry corresponding to the row indexed by multiset U and column indexed by multiset V contains variable $X_{U \cup V}$. Let poly$(\leq 2d)$ be the set of polynomials of degree at most $2d$ and define the linear operator $\tilde{\mathbb{E}} : poly(\leq 2d) \longrightarrow \mathbb{R}$ such that $\tilde{\mathbb{E}}\left[\prod_{i \in S} x_i\right] = X_S$ when $|S| \leq 2d$. Then $sos_d(Q)$ is the following optimization problem:*

Maximize: $\tilde{\mathbb{E}}[p(x)]$

Subject to:

- $X \succeq 0$
- $X_\emptyset = 1$
- $\tilde{\mathbb{E}}\left[g_i(x) \prod_{i \in U} x_i\right] = 0, \forall i \in \{1, 2, ...m\}, U \subseteq \{1, 2, .., n\}, \deg(g_i) + |U| \leq 2d.$

Since SDPs can be solved in polynomial time, $\text{sos}_d(Q)$ can be solved in time polynomial in $(n+1)^d$. In the case of the MAXCUT relaxation, the resulting SDP has a beautiful geometric meaning, as seen in the previous section. However, for general polynomial optimization problems how do we make sense out of the definition for $\text{sos}_d(Q)$. In this case, although there is no elegant geometric picture, the notions of pseudodistributions help provide an explanation [Sch16].

4.2 Pseudomoments, Pseudoexpectations, and Pseudodistributions

Suppose we could solve Q directly. The result would either be an optimal solution y^* or a distribution Y over a set of solutions in $\mathbb{R}^n$ which maximize the objective polynomial p so that $\text{OPT}(Q) = \mathbb{E}_{y \in Y}[p(y)]$. However, we cannot expect that $\text{sos}_d(Q)$ outputs an actual distribution over the solutions to Q, however, $\tilde{\mathbb{E}}$ satisfies many of the properties of the expectation operator and other useful properties:

- *Linearity of Expectation:* $\tilde{\mathbb{E}}[p(x) + q(x)] = \tilde{\mathbb{E}}[p(x)] + \tilde{\mathbb{E}}[q(x)]$ when $\deg(p), \deg(q) \leq 2d$ since $\tilde{\mathbb{E}}$ is a linear operator.

- *Non-Negativity of Low-Degree Squares:* $\tilde{\mathbb{E}}[p(x)^2] \geq 0$ if $\deg(q) \leq d$ because X is PSD and thus we can write $\tilde{\mathbb{E}}[p(x)^2]$ as $v^T X v \geq 0$ where v is the coefficient of vector for p.

For this reason we call $\tilde{\mathbb{E}}$ the pseudoexpectation. The solution to $\text{sos}_d(Q)$ specifies the pseudoexpectations of the monomials with degree at most $2d$, which we call pseudomoments:

$$\tilde{\mathbb{E}}\left[\prod_{i \in S} x_i\right] = X_S.$$

Thus, we can think of X as a distribution-like object over the solutions to Q. The solution to $\text{sos}_d(Q)$ is known as a pseudodistribution [Sch16].

4.3 Hierarchies, The Big Picture

Note that for an SOS relaxation at degree $2d$ we require that $\deg(p) \leq 2d$ and $\deg(g_i) \leq 2d$, $\forall i \in \{1, 2, ..., m\}$. By increasing d we can obtain SOS relaxations with more constraints and variables. These relaxations turn out to be more powerful, however they come at a cost since solving an SDP with more constraints and variables requires more time. The family of SOS relaxations that results from increasing the degree is known as the sum-of-squares hierarchy [Sch16].

4.4 Sum-of-Squares Proofs, The Dual View

Why is the SOS relaxation so powerful? To see, we must consider the dual version of the optimization problem $\text{sos}_d(Q)$.

As a warmup, we begin with a discussion of duality in the theory of linear programming. Consider the two linear programs:

Maximize $c^T x$

Subject to:

- $Ax \leq b$
- $x \geq 0$.

and:

Minimize $y^T b$

Subject to:

- $y^T A \geq c^T$
- $y \geq 0$

Though they may seem unrelated, they are in fact intimately related, and are known as duals of each other.

Theorem 12 (Weak Duality of LPs) *If x is a a feasible solution (satisfies the constraints) to the first linear program above and y is a feasible solution to the second linear program above then:*

$$c^T x \leq y^T b$$

Proof Since $x \geq 0$ multiplying both sides of $y^T A \geq c^T$ (which is also true) by x gives:

$$y^T A x \geq c^T x.$$

Similarly, since $Ax \leq b$ and $y^T \geq 0$ we have:

$$y^T A x \leq y^T b$$

The result follows immediately [Moi16]. ■

In fact the two LPs are even more closely tied, under certain mild conditions, both of them have the same optimal value! This result is known as strong duality, however, we will not give a proof for it here. Broadly, duality relates the solutions to two seemingly unrelated problems. SDPs also possess a theory of duality and consequently, so do SOS relaxations. To begin our venture into duality for SOS relaxations we introduce some convenient notational changes. Recall the inner product between two matrices used in the definition of SDPs in section 6: $\langle A, B \rangle = \sum_{(i,j)} A_{ij} B_{ij}$. We then let $P, G_1, G_2, ..., G_m$ be $(n+1)^d \times (n+1)^d$ matrices such that $\langle P, X \rangle = \tilde{\mathbb{E}}[(p(x)]$ and $\langle G_i, X \rangle = \tilde{\mathbb{E}}[g_i(x)]$ (note that the polynomial constraints can be redefined to capture the SDP constraints). We can then rephrase $\text{sos}_d(Q)$ as a minimization SDP program, $\text{sos}_d^-(Q)$:

Minimize $-\langle P, X \rangle$

Subject to:

- $\langle G_i, X \rangle, \forall i \in \{1, 2, .., m\}$
- $X \geq 0$
- $\langle M_\emptyset, X \rangle = 1$

$M_\emptyset$ is the matrix with 1 in the entry with both row index and column index corresponding to $\emptyset$ and it encapsulates the condition that $X_\emptyset = 1$ from above. It is quite clear that the optimal value for $\text{sos}_d^-(Q)$ is the negative of the optimal value for $\text{sos}_d(Q)$. The dual SDP program for $\text{sos}_d^-(Q)$ is $\text{sos}_d^+(Q)$:

- $\max_{y \in \mathbb{R}^{m+1}} y_\emptyset$
- Subject to:
  - $-P - y_\emptyset \cdot M_\emptyset - \sum_{j \in \{1,2,..,m\}} y_j \cdot G_j = S \geq 0$

We can think of y as a vector in $\mathbb{R}^{m+1}$ whose first index is $\emptyset$ and whose last m indices are $\{1, 2, ..., m\}$. SDP duality says that the optimal value for $\text{sos}_d^+(Q)$ is an upper bound for the optimal value of $\text{sos}_d^-(Q)$, thus letting $y_\emptyset^*$ be the optimal value of $\text{sos}_d^+(Q)$, we have:

$$y_\emptyset^* = c + \text{OPT}(\text{sos}_d^-(Q)) = c - \text{OPT}(\text{sos}_d(Q))$$

for some nonnegative number c. From the definition of $\text{sos}_d^+(Q)$ we have:

$$P = -y_\emptyset^* \cdot M_\emptyset - S + \sum_{j \in \{1,2,..,m\}} y_j^* \cdot G_j$$

Take:

$$S' = S + c \cdot M_\emptyset$$

Note that S' is PSD since S is PSD, $c \cdot M_\emptyset$ is PSD (note that multiplying $\sqrt{c} \cdot M_\emptyset$ by itself results in $c \cdot M_\emptyset$), and because the sum of two PSD matrices is PSD. Rewriting P by using the definition of $y_\emptyset^*$ (that results from duality) and the definition of S', we get (below $\text{OPT} = \text{OPT}(\text{sos}_d(Q))$):

$$\begin{aligned} P &= -(c - \text{OPT}) \cdot M_\emptyset - S + \sum_{j \in \{1,2,..,m\}} y_j^* \cdot G_j = \\ & M_\emptyset \cdot \text{OPT} - c \cdot M_\emptyset - S + \sum_{j \in \{1,2,..,m\}} y_j^* \cdot G_j = \\ & M_\emptyset \cdot \text{OPT} - S' + \sum_{j \in \{1,2,..,m\}} y_j^* \cdot G_j \end{aligned}$$

Let $x \in \mathbb{R}^n$. Let $\hat{x} = [1, x^T]^T$. Note that S' is positive semidefinite so we can write it as $S' = AA^T$ for some matrix A. But note that AA^T can also be written as the sum of the outer products of the i^{th} column of A with the i^{th} row of A^T (which are both the same vectors) where i ranges over the number of rows/columns in A. Thus, S' is of the form $\sum aa^T$. Taking the quadratic form of P with $\hat{x}^{\otimes d}$ we get:

$$(\hat{x}^{\otimes d})^T P(\hat{x}^{\otimes d}) = \text{OPT}(\hat{x}^{\otimes d})^T M_\emptyset(\hat{x}^{\otimes d}) - (\hat{x}^{\otimes d})^T \sum \left(aa^T\right)(\hat{x}^{\otimes d}) + \sum_{j \in \{1,2,..,m\}} y_j^* (\hat{x}^{\otimes d})^T G_j(\hat{x}^{\otimes d})$$

Note that the left hand side is just $p(x)$ and that $(\hat{x}^{\otimes d})^T G_j(\hat{x}^{\otimes d}) = g_j(x)$. Since $(\hat{x}^{\otimes d})^T M_\emptyset(\hat{x}^{\otimes d}) = 1$ and since for any two vectors we have $\langle a, b \rangle = a^T b$, overall, we get:

$$p(x) = OPT - \sum \left\langle a, \hat{x}^{\otimes d} \right\rangle^2 + \sum_{j \in \{1,2,..,m\}} y_j^* g_j(x)$$

Let $q_a(x)$ be the polynomial $\left\langle a, \hat{x}^{\otimes d} \right\rangle$ with coefficients specified by a, then we get:

$$p(x) = OPT - \sum q_a(x)^2 + \sum_{j \in \{1,2,..,m\}} y_j^* g_j(x)$$

Though this may not seem like much at first, it is actually quite astounding! In fact, the right hand side of the equation above has a special name: it is known as a sum-of-squares proof. For any vector x in the feasible region for Q, the $g_j(x)$'s is 0 and since $q_a(x)^2$ is always non-negative, we get that for any feasible point OPT is an upper bound for $p(x)$. Overall, we have the following theorem.

Theorem 13 (Sum-of-Squares (SOS) Proof) *The dual of $sos_d(Q)$ provides a degree-d sum-of-squares proof for the inequality $p(x) \leq OPT(sos_d(Q))$ for all x in the feasible region for Q.*

The above theorem also suggests we can go the other way. Providing a SOS proof that $p(x)$ has some upper bound c implies that the objective function of the SDP (the pseudoexpectation of $p(x)$) is upper bounded by this value, as well. Thus, SOS proofs are a method for upper bounding pseudoexpectations. This is the basic theory of duality for SOS [Sch16]. This is the reason SOS-based techniques are said to take proofs to algorithms!

5 Toolkit for SOS Proofs and Pseudoexpectations

As seen in the last section, coming up with SOS proofs for various polynomial inequalities is an important task. As a result, a standard toolkit for this endeavors would be invaluable. Furthermore, tools for manipulating pseudoexpectations are also useful. In this section, we explore these goals.

5.1 SOS Proofs

We begin with slightly changing the setting and definition of an SOS-proof and by introducing some notation. This will be especially useful for notational convenience. First, we change our constraints from equalities to inequalities. We let our set of polynomial constraints be $G = \{g_1(x) \geq 0, g_2(x) \geq 0, ..., g_m(x) \geq 0\}$. We say G implies $p(x) \geq 0$ with an SOS-proof, if we can write:

$$p(x) = \sum_{S \subset \{1,2,..,m\}} b_S(x) \prod_{i \in S} g_i(x)$$

where $b_S(x)$ are sums of squares of polynomials. For each $S \subset \{1, 2, ..., m\}$ if $\deg\left(b_S(x) \prod_{i \in S} g_i(x)\right) \leq d$ then we say the proof is of degree d and write:

$$G \vdash_d p(x) \geq 0.$$

A few remarks are in hand:

- $G \vdash r(x) \geq s(x)$ is the same as $G \vdash r(x) - s(x) \geq 0$.
- Our constraint set G can include equalities of the form $r(x) = s(x)$, since this can just be encapsulated by two inequalities: $r(x) - s(x) \leq 0$ and $r(x) - s(x) \geq 0$.
- If there are no constraints, we simply write: $\vdash_d p(x) \geq 0$

We now start the main task: expanding our SOS-proof toolkit:

Claim 14 (Squaring SOS-Proofs) *If $\vdash_d p(x) \leq q(x)$, and p and q are sums of squares then $\vdash_{2d} p(x)^2 \leq q(x)^2$.*

Proof $q(x)^2 - p(x)^2 = (q(x) + p(x))(q(x) - p(x))$ Since p, q are sums of squares, their sum and difference are too. Since the product of two sum of squares is also a sum of squares, the result follows [Hop18]. ■

Theorem 15 (SOS Triangle Inequality) *Let t be a power of 2. Then* $\vdash_t (a+b)^t \leq 2^{t-1}(a^t+b^t)$.

Proof We use induction. We take $t = 2$ as the base case. We need to show $\vdash_2 (a+b)^2 \leq 2(a^2+b^2)$. This follows because $2(a^2+b^2) - (a+b)^2 = a^2+b^2-2ab = (a-b)^2$. Now suppose:

$$\vdash_{t/2} (a+b)^{t/2} \leq 2^{t/2-1}(a^{t/2}+b^{t/2})$$

Using the claim above for squaring SOS-proofs, we get:

$$\vdash_t (a+b)^t \leq 2^{t-2}(a^{t/2}+b^{t/2})^2$$

Then, we can just apply the proof for the base case taking $a' = a^{t/2}$ and $b' = b^{t/2}$, and the result follows immediately [Hop18]. ■

Many times we want to take integer constraints, for instance $x \in \{0,1\}$ and relax them to $0 \leq x \leq 1$. As such the following SOS-proof is useful.

Claim 16 (SOS Boolean Inequality) $x^2 = x \vdash 0 \leq x \leq 1$

Proof The constraint is really two inequalities $x \leq x^2$ and $x \geq x^2$. The latter implies $x \geq 0$. We have:

$$1 - x = (1-x)^2 + (x - x^2) \geq 0$$

The last inequality follows again because $x \geq x^2$. Overall this implies $x \leq 1$, as desired [Hop18]. ■

We end with an SOS version of one of our most beloved inequalities, Cauchy-Schwartz.

Theorem 17 (SOS Cauchy-Schwartz Inequality) *Let $x_1, x_2, .., x_n, y_1, y_2, ..., y_n$ be a set of variables. Then:*

$$\vdash_2 \left(\sum_{i\in\{1,2,..,n\}} x_i y_i\right)^2 \leq \left(\sum_{i\in\{1,2,...,n\}} x_i^2\right)\left(\sum_{i\in\{1,2,...,n\}} y_i^2\right)$$

Proof The equality below can be checked via expanding both sides carefully.

$$\left(\sum_{i\in\{1,2,...,n\}} x_i^2\right)\left(\sum_{i\in\{1,2,...,n\}} y_i^2\right) - \left(\sum_{i\in\{1,2,..,n\}} x_i y_i\right)^2 =$$

$$\sum_{1\leq i,j\leq n} (x_i y_j - x_j y_i)^2$$

Since the right hand side is a sum of squares, we are done [Hop18]. ■

5.2 Pseudoexpectations

As in the case of SOS proofs, standardizing tools for manipulating pseudoexpectations is useful. As such, we end this section with such a tool: the pseudoexpectation version of the Cauchy-Schwartz inequality.

Theorem 18 (Pseduoexpectation Version of Cauchy-Schwartz Inequality) *Let μ be a degree d pseudodistribution and let p and q be polynomials of degree at most $d/2$, then:*

$$\left(\tilde{\mathbb{E}}_\mu pq\right)^2 \leq \left(\tilde{\mathbb{E}}_\mu p^2\right)\left(\tilde{\mathbb{E}}_\mu q^2\right)$$

Proof We can assume both $\tilde{\mathbb{E}}_\mu p^2$ and $\tilde{\mathbb{E}}_\mu q^2$ are positive since if either is 0, the proof becomes trivial. We can also scale p and q by positive scalars such that:

$$\tilde{\mathbb{E}}_\mu p^2 = \tilde{\mathbb{E}}_\mu q^2 = 1$$

Thus, we are left to prove:

$$\tilde{\mathbb{E}}_\mu pq \leq 1$$

We have:

$$\tilde{\mathbb{E}}_\mu (p-q)^2 \geq 0 \implies \tilde{\mathbb{E}}_\mu p^2 + \tilde{\mathbb{E}}_\mu q^2 - 2\tilde{\mathbb{E}}_\mu pq \geq 0 \implies 2 \geq 2\tilde{\mathbb{E}}_\mu pq$$

as desired. Note that throughout this proof we have used the fact that $\tilde{\mathbb{E}}_\mu h^2 \geq 0$ for any polynomial h of degree at most $d/2$, this property is also stated in the last section where pseudoexpectations are introduced [Hop18]. ■

6 Tensor Decomposition and SOS

In this section, we explore an important problem in unsupervised learning known as tensor decomposition. We give a classical algorithm for a certain version of this problem. We then outline (but do not rigorously prove) how SOS-based techniques can be used for it.

6.1 Tensor Basics

6.1.1 What is a Tensor?

In our case, a tensor will be a grid of numbers arranged in a dimension ≥ 3. An $n_1 \times n_2 \times ... \times n_k$ k-tensor contains $n_1 n_2 n_3 \cdot ... \cdot n_k$ numbers. An $n_1 \times n_2 \times n_3$ tensor is a set of n_3 $n_1 \times n_2$ matrices stacked on top of each other. So the (i, j, k)-th entry of this matrix (where $1 \leq i \leq n_1$, $1 \leq j \leq n_2$, $1 \leq k \leq n_3$) is the (i, j)-th entry of the k-th matrix in the stack. For most of these notes, we will we will focus on 3-tensors [Rou17].

6.1.2 Tensor Products

Tensor products are the higher dimensional analogs of outer products for matrices, and are denoted by the symbol $\otimes$. Specifically, given vectors $v_1, v_2, ..., v_s$ with lengths $n_1, n_2, ..., n_s$, respectively, their tensor product $v_1 \otimes v_2 \otimes ... \otimes v_s$ is the $n_1 \times n_2 \times ... \times n_s$ s-tensor whose $(i_1, i_2, ..., i_s)$ entry is $v_1(i_1)v_2(i_2) \cdot ... \cdot v_s(i_s)$, where $v_s(i_s)$ denotes the i_s-th entry in vector v_s [Rou17]. Note another name for the tensor product is the Kronecker product, which we have already seen above.

6.1.3 Tensor Rank

Similar to the matrix case, a rank one tensor is a tensor that can be written as a tensor product of vectors. The rank of a tensor T is the minimum number r such that T can be written as the sum of r rank one tensors [Moi14].

6.1.4 Matrix Decomposition and the Rotation Problem

We turn to a famous example called Spearman's Hypothesis. Charles Spearman was a famous psychologist who believed human intelligence consisted of two parts, math and verbal, and that a person's performance on a test should depend on how good one is at both these types of intelligence. Spearman then took the test scores of m individuals on n tests and constructed a $m \times n$ matrix M, where the (i, j)-th entry is the score of individual i on test j. To test if M agreed with his ideas on intelligence, Spearman took a rank 2 approximation to M of the form $u_1 v_1^T + u_2 v_2^T$ for vectors u_1, v_1, u_2, v_2 (perhaps by doing a singular value decomposition), and confirmed his hypothesis to a reasonable extent by viewing u_1 and u_2 as vectors representing each students math and verbal abilities, respectively, and v_1 and v_2 as vectors representing how much each exam tests math and verbal abilities, respectively.

The problem with this approach is that even if we had some algorithm which spits factor analyses of matrices (approximating them in terms of outer products of vectors) and even if every matrix M had some factoring which can be nicely interpreted, such as in Spearman's case above, the algorithm may not spit out the factoring that is easily interpretable since matrix factorizations are not unique. To see why this is true first note that $u_1 v_1^T + u_2 v_2^T = UV^T$, where U is the matrix with columns u_1 and u_2 and V is the matrix with columns v_1 and v_2. Also, note for any orthogonal matrix O, $OO^T = I$. Take some orthogonal matrix O which is not the identity. We have:

$$UV^T = U(I)V^T = U(OO^T)V^T = (UO)(O^T V^T)$$

Clearly, since $O \neq I$, $UO \neq U$ and $O^T V^T \neq V^T$, taking the columns of UO and rows of $O^T V^T$ leads to a new factorization of the matrix. This is known as the rotation problem.

What is nice about tensors is that they do not suffer from this problem, in other words, under very general conditions, the factors when we express them as sums of outer products are unique [Moi14].

6.1.5 Discrepancies between Linear Algebra for Matrices and Tensors

Although above we see tensors do have advantages over matrices, in many other aspects they are far worse and awkward. Here is a list:

- In matrices, the column rank equals the row rank, in tensors there is no similar notion.
- The best rank k approximation of a matrix is the best rank k approximation of the best rank $k + 1$ approximation of the matrix (for instance, when using the SVD to get the best rank t approximation one truncates and leaves the first t terms of summation).

- For a matrix, decomposing it using real or complex numbers yields the same rank, but this is not the case with tensors.

In general, given standard complexity assumptions, there is no nice way to compute the minimum rank of a tensor by decomposing it (this task is NP-Hard) (however, we will see later, that under *very* but not *fully* general conditions there is an algorithm for decomposing 3-tensors) [Rou17].

6.2 Classical Algorithm for Tensor Decomposition

6.2.1 The Tensor Decomposition Problem and Upper Bounds on Rank

The tensor decomposition problem is given a rank r tensor T, to output a decomposition of it. The noisy tensor decomposition problem is to recover an approximate rank r decomposition of T. An ϵ-approximate decomposition of T with respect to some norm $\|\cdot\|$ is a decomposition D, such that $\|T - D\| \leq \epsilon\|T\|$.

We end with a discussion on upper bounds for tensor rank. Let us consider 3-tensors that are $n \times n \times n$ (most results with a little extra work are generalizable when the the dimensions are not all equal). An upper bound for such a tensor, T, is n^3 (by literally adding the tensor with T_{ijk} in the (i, j, k) position and 0's else where for each triple of i, j, and k, for $1 \leq i, j, k \leq n$). More formally, let e_s denote the n-dimensional vector with a 1 in the s-th position and 0's elsewhere. Then:

$$T = \sum_{1 \leq i,j,k \leq n} T_{ijk} e_i e_j e_k$$

This upper bound can be improved to n^2. To see why, view T as n slices of $n \times n$ matrices. Let these matrices be known as $M_1, M_2, ..., M_n$. Then, we have:

$$T = \sum_{i=1}^{n} M_i \otimes e_i$$

We are kind of abusing notation here by referring to the tensor product between a matrix and vector. Each matrix M_i has rank at most n, since it is an $n \times n$ matrix, and thus, can be decomposed into the sum of at most n vector outer products. Thus, in the decomposition of T above, we can split each of the n terms into a sum of at most n vector outer products, thus giving a rank at most n^2.

As we will see in the next section, under very general assumptions, there is an algorithm for decomposing 3-tensors for rank at most n. However, the upper bound we have given for 3-tensor rank is n^2, and in fact, with very high probability a random tensor (with entries chosen independently at random from the interval $[0, 1]$) has rank on the order of n^2. This task of decomposing 3-tensors with rank much greater than n is known as the overcomplete case, and is one of the main focuses of recent research.

6.2.2 Jennrich's Algorithm

Theorem 19 *Given an $n \times n \times n$ rank r 3-tensor $T = \sum_{i=1}^{r} u_i \otimes v_i \otimes w_i$, there is an algorithm to recover this decomposition (up to constant scaling) given the following conditions: 1) the vectors*

in the set $\{u_1, ..., u_r\}$ *are linearly independent, 2) the vectors in the set* $\{v_1, v_2, ..., v_r\}$ *are linearly independent, 3) the vectors in the set* $\{w_1, ..., w_r\}$ *are pairwise linearly independent.*

We note two things. Firstly, condition 3 is weaker than requiring all the vectors in the set $\{w_1, ..., w_r\}$ to be linearly independent. This is because if all the vectors are linearly independent then any two of them must be linearly independent. However, if each pair of them are linearly independent, then the entire set might not be linearly independent, here is an example: $\{(1, 0, 0), (0, 1, 1), (1, 1, 1)\}$. Secondly, the conditions limit the rank r to be $\leq n$ because if a set of n-dimensional vectors is linearly independent then it has at most n vectors in it. Here is the algorithm:

- Choose uniformly at random unit vectors $x, y \in \mathbb{R}^n$.
- Define the matrix A_x as follows. Think of T as n slices of $n \times n$ matrices, M_1, M_2,...,M_n. Then let $A_x = \sum_{i=1}^{n} x_i M_i$ (the sum of the n matrices weighted by the coordinates of x). Define A_y analogously.
- Compute the eigendecompositions of the matrices $A_x A_y^{-1}$ and $A_x^{-1} A_y$. Then, the u_i's and v_i's will appear as the columns/rows of the matrices in the eigendecomposition (more clarity is given below). u_i and v_i can be paired off since the eigenvalues corresponding to them will be reciprocals. From this, we have a linear system of equations and can solve for the w_i's.

We now prove the correctness of this algorithm. We have the following lemma:

Lemma 20 $A_x = \sum_{j=1}^{r} \langle w_j, x \rangle u_j v_j^T$ *and* $A_y = \sum_{j=1}^{r} \langle w_j, y \rangle u_j v_j^T$

Proof Let $w_i = (w_{i1}, ..., w_{in})$ and $x = (x_1, .., x_n)$. We think of T in its decomposed form $\sum_{i=1}^{r} u_i \otimes v_i \otimes w_i$. Then if we think of each of the r tensors in this summation as being sliced into n $n \times n$ matrices, A_x results when we multiply x_i times the sum of the i-th slice from each of the tensors for $1 \leq i \leq n$ and sum the resulting matrices up. x_i times the sum of the i-th slice from each of the tensors is $\sum_{j=1}^{r} x_i w_{ji} u_j v_j^T$. Now, we sum over all the slices to get $A_x = \sum_{i=1}^{n} \sum_{j=1}^{r} x_i w_{ji} u_j v_j^T = \sum_{j=1}^{r} \sum_{i=1}^{n} x_i w_{ji} u_j v_j^T = \sum_{j=1}^{r} \langle w_j, x \rangle u_j v_j^T$, as desired. The proof for A_y is analogous. ■

Now, let U and V be the matrices with columns $\{u_1, ..., u_r\}$ and $\{v_1, ..., v_r\}$, respectively. Then, we have $A_x = UDV^T$ and $A_y = UEV^T$, where D and E are the diagonal matrices with entries $\{\langle w_1, x \rangle, ..., \langle w_r, x \rangle\}$ and $\{\langle w_1, y \rangle, ..., \langle w_r, y \rangle\}$, respectively. We now have:

$$A_x A_y^{-1} = (UDV^T)(UEV^T)^{-1} = UDV^T V^{-T} E^{-1} U^{-1} = U(DE^{-1})U^{-1}$$

and:

$$A_x^{-1} A_y = (UDV^T)^{-1}(UEV^T) = V^{-T} D^{-1} U^{-1} UEV^T = V^{-T}(D^{-1}E)V^T$$

Now, assume the eigendecompositions for these two matrices are unique up to constant scaling (this is true if the eigenvalues are unique and we will soon explain the intuition for why this is the case). Then, the columns of first matrix in the first eigendecomposition must be the u_i's and the rows of the second matrix in the second eigendecomposition must be the

v_i's. But how can we pair off a u_i with a v_i? Well, when we invert a diagonal matrix, we get a diagonal matrix with entries as multiplicative inverses of the original matrix, and when we multiply two diagonal matrices we get a diagonal matrix with the products of the original entries as entries. Thus, DE^{-1} and $D^{-1}E$ have entries of the form $\frac{\langle w_i,x\rangle}{\langle w_i,y\rangle}$ and $\frac{\langle w_i,y\rangle}{\langle w_i,x\rangle}$, respectively for $1 \leq i \leq r$, so indeed the eigenvalues will be reciprocals of each other so we can pair the eigenvectors off accordingly.

Now, the last issue to consider is the uniqueness of the eigenvalues. Suppose, two of the eigenvalues in DE^{-1} were equal. Then, we would have: $\frac{\langle w_i,x\rangle}{\langle w_i,y\rangle} = \frac{\langle w_j,x\rangle}{\langle w_j,y\rangle}$. However, such an event will not take place (a more rigorous argument is needed to truly prove this) because w_i and w_j can never be multiples of each other due to our condition that the vectors in the set $\{w_1, ..., w_r\}$ are pairwise linearly independent [Har70].

6.3 Incorporating Sum-of-Squares

SOS-based techniques are utilized to deal with various issues in tensor decomposition. We give a very high level outline how this can be done. First, we rephrase the tensor decomposition problem, as follows:

Definition 21 (Redefining Tensor Decomposition Problem) *Given the first three moments of a uniform distribution over the unit vectors a_1, a_2,...,a_n, recover the vectors. Recall that the k-th moment of such a distribution is:*

$$M_k = \frac{1}{n}\sum_{i=1}^{n} a_i^{\otimes k}$$

We note that $a_1, a_2, .., a_n$ can be taken to be orthogonal due to something known as the whitening transformation [BS16]. We consider the case when $a_1, a_2, .., a_n$ are themselves randomly chosen unit vectors and deal with an overcomplete case. In particular, we consider when $n << d^{1.5}$. In that case, the following general plan of attack works:

- Lift the third moment M_3 into sixth moment M_6.
- Apply Jennrich's algorithm on sixth moments. This should give: $a_1^{\otimes 2}, ..., a_n^{\otimes 2}$ (up to some error). From here $a_1, a_2, ...a_n$ can be recovered. Intuitively, Jennrich's algorithm should work reasonably well since the $a_i^{\otimes 2}$ has length d^2 and $n << d^{1.5}$ so all the vectors in $\{a_1^{\otimes 2}, ..., a_n^{\otimes 2}\}$ should look linearly independent.
- The main question at hand is how to lift the third moments to sixth moments. One approach would be to find some distribution over unit vectors which has third moment close to M_3 and output the sixth moment of that distribution. This approach works in the sense that the outputted sixth moment will actually be close to the sixth moment of $a_1, a_2, ..., a_n$. However, the issue remains how to find such a distribution. Since pseudodistributions can be optimized over, via SDPs as shown above, reducing the search space to such objects, we get a pseudodistribution with desirable properties. Using tools like the quadratic sampling lemma (see [BS16]) which lets one sample from a distribution with similar properties to a given pseudodistribution, we can usefully achieve various needed tasks.

Of course there are many details to be filled in, but hopefully, this provides a glimpse into how SOS-based techniques are used in real algorithmic problems [SHM+16].

7 Historical Notes and Conclusion

The origin of SOS optimization is algebraic geometry. In the late 19th centuries mathematicians David Hilbert and Hermann Minkowski asked whether any non-negative polynomial could be written as the sum of squares. Hilbert showed the answer is no with a non-constructive argument and in the 1960s Motzkin gave an example of such a polynomial, now known as the Motzkin polynomial:

$$1 + x^4y^2 + x^2y^4 - 3x^2y^2$$

The nonnegativity of this polynomial can be easily verified via the AM-GM inequality:

$$\frac{1 + x^4y^2 + x^2y^4}{3} \geq (x^4y^2x^2y^4)^{1/3} = x^2y^2$$

As his 17th problem, Hilbert famously asked whether any nonnegative polynomial can be expressed as the sum of squares of rational functions (which are quotients of polynomials). In this case, the answer turned out to be affirmative due to the work of Emil Artin. Then, in a result now known as Positivstellensatz, Krivine and Stengle showed that any unsatisfiable set of polynomial constraints can be proven so via an SOS proof. Basically, the proof is of the form: if the polynomial constraints are satisfiable, then there exists polynomials p_i, such that $\sum_i p_i^2 = -1$, however, this is not possible so the polynomial constraints are unsatisfiable [BS16] [Kri64] [Ste74].

In the early 2000s, Pablo Parrilo (2000) and Jean Laserre (2001) independently discovered the relationship between SOS proofs, semidefinite programming, and polynomial optimization, which is elaborated upon in this exposition. This is the reason that the SOS hierarchy described in this paper is alternatively known as the Parrilo-Laserre hierarchy. Naum Shor had similar but not as general results, as well [Par00] [Las01].

SOS-based techniques have had a significant impact in theoretical computer science. They have been used to prove lower bounds in complexity theory and have an intimate connection to the unique games conjecture. They have also been used for algorithmic applications as seen in this paper. Problems SOS-based techniques have been used for are: planted clique, MAXCUT, planted sparse vector, dictionary learning, tensor decomposition, etc [BS16].

Though SOS techniques are many times viewed as highly theoretical, efforts have been made to make them practical. For instance, Parrilo's research group has a MATLAB tool known as SOSTOOLS for formulating and solving SOS optimization problems. The details can be found on the tool's website [PPA+18]. In terms of its applications, SOS optimization has the potential to impact fields, such as machine learning, control and dynamics of robots, statistics, quantum computation, software verification, and more [BS16].

8 Acknowledgements

I want to thank Professor Aaron Roth for his valuable mentorship. I also thank him for providing the LaTeX template used in this paper. I am also grateful to Professor Sanjeev Khanna for introducing him to theoretical computer science. I would also like to thank Sidhanth Mohanty for a helpful discussion.

References

[BS16] Boaz Barak and David Steurer. Proofs, beliefs, and algorithms through the lens of sum-of-squares. 2016.

[GW94] Michel Goemans and David Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *26th Annual ACM Symposiumn on the Theory of Computing*, 1994.

[Har70] R. Harshman. Foundations of the parfac procedure: model and conditions for an 'explanatory' multi-mode factor analysis. *UCLA Working Papers in Phonetics*, pages 1–84, 1970.

[Hop18] Sam Hopkins. Clustering and sum of squares proofs: Six blog posts on unsupervised learning. 2018.

[JM07] Bernd Gartner Jiri Matousek. *Understanding and Using Linear Programming*. Springer, 2007.

[Kha16] Sanjeev Khanna. Lecture notes in randomized algorithms 3-4: Approximation via randomized rounding. 2016.

[Kri64] Jean-Louis Krivine. Anneaux préordonnés. *Journal d'analyse mathématique*, 1964.

[Las01] Jean Laserre. Global optimization with polynomials and the problem of moments. *SIAM Journal of Optimization*, 11:796–917, 2001.

[Moi14] Ankur Moitra. Algorithmic aspects of machine learningy. 2014.

[Moi16] Ankur Moitra. Lecture notes in advanced algorithms 11 (introduction to linear programming) and 19 (maxcut and semidefinite programming). 2016.

[Par00] Pablo Parrilo. Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization. phd thesis. 2000.

[PPA+18] Pablo Parrilo, Antonis Papachristodoulou, James Anderson, Giorgio Valmorbida, Stephen Prajna, and Peter Seiler. Sostools - a sum of squares optimization toolbox for matlab. 2018.

[Rou17] Tim Roughgarden. Modern algorithmic toolbox lecture 10 (tensors and low-rank discovery). 2017.

[Sch16] Tselil Schramm. Intro to the sum-of-squares hierarchy. 2016.

[SHM+16] David Steurer, Sam Hopkins, Tengyu Ma, Tselil Schramm, and Jonathan Shi. Tensor decompositions, sum-of-squares proofs, and spectral algorithms. 2016.

[Ste74] Gilbert Stengle. A nullstellensatz and a positivstellensatz in semialgebraic geometry. *Mathematische Annalen*, 1974.