Methods of Moments for Learning Stochastic Languages: Unified Presentation and Empirical Comparison

Borja Balle[1] BBALLE@CS.MCGILL.CA
William L Hamilton[1] WHAMIL3@CS.MCGILL.CA
Joelle Pineau JPINEAU@CS.MCGILL.CA
Reasoning and Learning Laboratory, School of Computer Science, McGill University, Montreal, QC, Canada

Abstract

Probabilistic latent-variable models are a powerful tool for modelling structured data. However, traditional expectation-maximization methods of learning such models are both computationally expensive and prone to local-minima. In contrast to these traditional methods, recently developed learning algorithms based upon the method of moments are both computationally efficient and provide strong statistical guarantees. In this work we provide a unified presentation and empirical comparison of three general moment-based methods in the context of modelling stochastic languages. By rephrasing these methods upon a common theoretical ground, introducing novel theoretical results where necessary, we provide a clear comparison, making explicit the statistical assumptions upon which each method relies. With this theoretical grounding, we then provide an in-depth empirical analysis of the methods on both real and synthetic data with the goal of elucidating performance trends and highlighting important implementation details.

1. Introduction

Probabilistic models with latent variables are a powerful and flexible tool for modelling complex probability distributions over structured data. Unfortunately, this power comes at a price: learning and predicting with latent variable models is typically a computationally expensive task, to the point where obtaining exact solutions can become intractable. Designing efficient, scalable, and accurate approximation algorithms for these tasks is an important challenge that needs to be solved if we want to use these models for solving large-scale real world problems.

A classic solution to the problem of learning latent variable models under the maximum likelihood criterion is the expectation–maximization (EM) algorithm (Dempster et al., 1977). However, this algorithm suffers from two fundamental limitations: there is a high computational cost on large state spaces, and no statistical guarantees about the accuracy of the solutions obtained are available. A way to overcome the limitations of EM is by considering relaxed versions of the maximum likelihood principle (Varin, 2008).

A recent alternative line of work consists of designing learning algorithms for latent variable models exploiting an entirely different statistical principle: the so-called method of moments (Pearson, 1894). The key idea underlying this principle is that, since the low order moments of a distribution are typically easy to estimate, by writing a set of equations that relate the moments with the parameters of the distribution and solving these equations using estimated moments, one can obtain approximations to the parameters of the target distribution. In some cases solving these equations only involves spectral decompositions of matrices or tensors and basic linear algebra operations (Anandkumar et al., 2012d;b). Since moment estimation can be performed in time linear in the number of examples, and the required algebraic operations involve dimensions independent of the number of training examples, this approach can provide extremely fast learning algorithms. In addition, statistical analyses show that these algorithms are robust to noise and can learn models satisfying some basic assumptions from samples of size polynomial in the relevant parameters (see Hsu et al., 2009; Anandkumar et al., 2012d; Balle & Mohri, 2012; Hsu & Kakade, 2013, and references therein).

A witness of the generality of the method of moments is the wide and ever-growing class of models that can be learned

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

[1]These authors contributed equally to this work.

with this approach. These include multiple probabilistic models over sequences, such as HMM (Hsu et al., 2009; Anandkumar et al., 2012d;b), weighted automata (Bailly et al., 2009; Balle et al., 2012), predictive states representations (Boots et al., 2011; Hamilton et al., 2013), and variants thereof (Siddiqi et al., 2010; Balle et al., 2011; Stratos et al., 2013; Bailly et al., 2013b). Building on top of these sequential models, algorithms for learning context-free formalisms used in natural language processing have been proposed (Bailly et al., 2010; 2013a; Cohen et al., 2012; 2013; Luque et al., 2012; Dhillon et al., 2012). Other classes of latent variable models known to be learnable using this approach include: multiple classes of tree-structured graphical models (Parikh et al., 2011; Song et al., 2013); mixture, admixture, and topic models (Anandkumar et al., 2012c;a;b;d; Chaganty & Liang, 2013); and community detection models (Anandkumar et al., 2013). Furthermore, the method can be extended using features functions (Boots et al., 2011; Recasens & Quattoni, 2013) and kernelized to deal with continuous observation spaces (Song et al., 2010; Boots et al., 2013).

Despite the expectations and promises raised by this new class of methods, their use is far from widespread in applied domains. In particular, few works in this area report empirical results, and in most cases these address only fairly restricted tasks. Therefore, it is unclear at this point whether methods of moments can provide truly competitive learning algorithms for wide ranges of problems and match the expectations set forward by their theoretical appeal. One explanation for this is that most of the papers describing the fundamentals of these methods provide very little or no implementation details – a notable exception being (Cohen et al., 2013). In addition, although competitive results have been obtained using methods of moments in NLP (Cohen et al., 2013; Balle et al., 2013) and robotics (Boots et al., 2011), naive implementations tend to perform much worse than expected, and significant amounts of tuning and optimization are needed to obtain fast and accurate algorithms.

The goal of this paper is to present a unified review of three different methods of moments in the context of learning hidden Markov models (HMM) and other probabilistic models over sequences. We consider the SVD-based algorithm from (Hsu et al., 2009), the convex optimization approach described in (Balle et al., 2012), and the symmetric tensor decomposition framework of (Anandkumar et al., 2012b). We choose these three methods because they are the easiest to present in our context, and in addition, almost every other algorithm based on the method of moments uses (variations of) these as a subroutine. Our comparison touches upon several aspects of these algorithms that we believe have been neglected in previous works. First, by giving a unified presentation we are able to stress the sometimes subtle differences between the three methods and discuss in which settings is each of them statistically consistent. And using this unified theoretical framework we provide a novel algebraic proof of consistency for one of the three methods (symmetric tensor decomposition). Then we present empirical experiments with synthetic and real data comparing between these algorithms and EM. By presenting results using two different accuracy metrics (word error rate and perplexity), our experiments unveil some interesting subtleties about the different methods. Finally, we provide a thorough description of implementation details required to make each method scalable and accurate.[2]

2. Stochastic Languages and Automata

2.1. Notation

We use bold letters to denote vectors $\mathbf{v} \in \mathbb{R}^d$, matrices $\mathbf{M} \in \mathbb{R}^{d_1 \times d_2}$, and third order tensors $\mathbf{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$. Given a matrix $\mathbf{M}$ we write $\|\mathbf{M}\|_F$ for its Frobenius norm and $\|\mathbf{M}\|_*$ for its trace/nuclear norm. We use $\mathbf{M}^+$ to denote the Moore–Penrose pseudo-inverse of $\mathbf{M}$. Sometimes we shall give names to the columns and rows of a matrix using ordered index sets $\mathcal{I}$ and $\mathcal{J}$. In this case we will write $\mathbf{M} \in \mathbb{R}^{\mathcal{I} \times \mathcal{J}}$ to denote a matrix of size $|\mathcal{I}| \times |\mathcal{J}|$ with rows indexed by $\mathcal{I}$ and columns indexed by $\mathcal{J}$.

A matrix $\mathbf{M} \in \mathbb{R}^{d \times d}$ is symmetric if $\mathbf{M} = \mathbf{M}^\top$. Similarly, a tensor $\mathbf{T} \in \mathbb{R}^{d \times d \times d}$ is symmetric if for any permutation ρ of the set $\{1, 2, 3\}$ we have $\mathbf{T} = \mathbf{T}^\rho$, where $\mathbf{T}^\rho(i_1, i_2, i_3) = \mathbf{T}(i_{\rho(1)}, i_{\rho(2)}, i_{\rho(3)})$ for every $i_1, i_2, i_3 \in [d]$. Given vectors $\mathbf{v}_i \in \mathbb{R}^{d_i}$ for $1 \leq i \leq 3$, we can take tensor products to obtain matrices $\mathbf{v}_1 \otimes \mathbf{v}_2 = \mathbf{v}_1 \mathbf{v}_2^\top \in \mathbb{R}^{d_1 \times d_2}$ and tensors $\mathbf{v}_1 \otimes \mathbf{v}_2 \otimes \mathbf{v}_3 \in \mathbb{R}^{d_1 \times d_2 \times d_3}$. For convenience we also write $\mathbf{v} \otimes \mathbf{v} \otimes \mathbf{v} = \mathbf{v}^{\otimes 3}$, which is a third order symmetric tensor.

Given a tensor $\mathbf{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ and matrices $\mathbf{M}_i \in \mathbb{R}^{d_i \times d_i'}$ we consider the contraction operation that produces a tensor $\mathbf{T}' = \mathbf{T}(\mathbf{M}_1, \mathbf{M}_2, \mathbf{M}_3) \in \mathbb{R}^{d_1' \times d_2' \times d_3'}$ with entries given by $\mathbf{T}'_{j_1 j_2 j_3} = \mathbf{M}_{1 i_1 j_1} \mathbf{M}_{2 i_2 j_2} \mathbf{M}_{3 i_3 j_3} \mathbf{T}^{i_1 i_2 i_3}$, where we used Einstein's summation convention.

Let Σ be a finite alphabet. We use $\Sigma^\star$ to denote the set of all finite strings over Σ, and we write λ for the empty string. Given two strings $u, v \in \Sigma^\star$ we write $w = uv$ for their concatenation, in which case we say that u is a prefix of w, and v is a suffix of w. Given two sets of strings $\mathcal{P}, \mathcal{S} \subseteq \Sigma^\star$ we write $\mathcal{P}\mathcal{S}$ for the set obtained by taking every string of the form uv with $u \in \mathcal{P}$ and $v \in \mathcal{S}$. When singletons are involved, we write $u\mathcal{S}$ instead of $\{u\}\mathcal{S}$. If $f : \Sigma^\star \to \mathbb{R}$ is a function, we use $f(\mathcal{P})$ to denote $\sum_{u \in \mathcal{P}} f(u)$. Given strings $u, v \in \Sigma^\star$, we denote by $|v|_u$ the number of occurrences of u as a substring of v.

[2] Code available here: https://github.com/ICML14MoMCompare/

2.2. Latent Variable Models

A *stochastic language* is a probability distribution over $\Sigma^\star$. More formally, it is a function $f : \Sigma^\star \to \mathbb{R}$ such that $f(x) \geq 0$ for every $x \in \Sigma^\star$ and $\sum_{x \in \Sigma^\star} f(x) = 1$. The main learning problem we consider in this paper is to infer a stochastic language $\hat{f}$ from a sample $S = (x^1, \ldots, x^m)$ of i.i.d. strings generated from some f. In order to give a succinct representation for $\hat{f}$ we use hypothesis classes based on finite automata. In the following we present several types of automata that are used throughout the paper.

A *weighted automaton (WA)* over Σ is a tuple $A = \langle \boldsymbol{\alpha}_0, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\}_{\sigma \in \Sigma} \rangle$, with $\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_\infty \in \mathbb{R}^n$ and $\mathbf{A}_\sigma \in \mathbb{R}^{n \times n}$. The vectors $\boldsymbol{\alpha}_0$ and $\boldsymbol{\alpha}_\infty$ are called the initial and final weights, respectively. Matrices $\mathbf{A}_\sigma$ are transition operators. The size n of these objects is the number of states of A. A weighted automaton A computes a function $f_A : \Sigma^\star \to \mathbb{R}$ as follows:

$$f_A(x_1 \cdots x_t) = \boldsymbol{\alpha}_0^\top \mathbf{A}_{x_1} \cdots \mathbf{A}_{x_t} \boldsymbol{\alpha}_\infty = \boldsymbol{\alpha}_0^\top \mathbf{A}_x \boldsymbol{\alpha}_\infty \quad . \quad (1)$$

A function $f : \Sigma^\star \to \mathbb{R}$ is *realized* by A if $f_A = f$. If f_A is a stochastic language, then A is a *stochastic automaton*.

A learning task one might consider in this setting is the following: assuming the target stochastic language can be realized by some WA, try to find a stochastic WA realizing approximately the same distribution (w.r.t. some metric). The methods given in (Hsu et al., 2009; Bailly et al., 2009) – which we review in Section 3 – can be used to solve this problem, provided one is content with a WA A such that $\hat{f} = f_A$ approximates f but is not necessarily stochastic. It turns out that this is an essential limitation of using WA as a hypothesis class: in general, checking whether a WA A is stochastic is an undecidable problem (Denis & Esposito, 2008). Thus, if one imperatively needs the hypothesis to be a probability distribution, it is necessary to consider methods that produce a WA which is stochastic *by construction* (e.g., a probabilistic automaton). Alternatively one can employ heuristics to approximate a probability distribution with a WA which is not stochastic (see Section 4.2).

A *probabilistic automaton (PA)* is a WA $A = \langle \boldsymbol{\alpha}_0, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\} \rangle$ where the weights satisfy (1) $\boldsymbol{\alpha}_0 \geq 0$, with $\boldsymbol{\alpha}_0^\top \mathbf{1} = 1$ and (2) $\boldsymbol{\alpha}_\infty \geq 0$, $\mathbf{A}_\sigma \geq 0$, with $\sum_\sigma \mathbf{A}_\sigma \mathbf{1} + \boldsymbol{\alpha}_\infty = \mathbf{1}$. These conditions say that $\boldsymbol{\alpha}_0$ can be interpreted as probabilities of starting in each state and that $\mathbf{A}_\sigma$ and $\boldsymbol{\alpha}_\infty$ define a collection of emission/transition and stopping probabilities. It is easy to check that PA are stochastic by construction; that is, when A is a PA the function f_A is a stochastic language. A *deterministic PA (DPA)* is a PA where $\boldsymbol{\alpha}_0(i) = 1$ for some $i \in [n]$, and for every $\sigma \in \Sigma$ and every $i \in [n]$ there exists a unique $j \in [n]$ such that $\mathbf{A}_\sigma(i, j) > 0$.

A *factorized weighted automaton (FWA)* is a tuple $A = \langle \boldsymbol{\alpha}_0, \boldsymbol{\alpha}_\infty, \mathbf{T}, \{\mathbf{O}_\sigma\}_{\sigma \in \Sigma} \rangle$ with initial and final weights $\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_\infty \in \mathbb{R}^n$, transition weights $\mathbf{T} \in \mathbb{R}^{n \times n}$, and emission weights $\mathbf{O}_\sigma \in \mathbb{R}^{n \times n}$, where the matrices $\mathbf{O}_\sigma$ are diagonal. One can readily transform a FWA into a WA by taking $B = \langle \boldsymbol{\beta}_0, \boldsymbol{\beta}_\infty, \{\mathbf{B}_\sigma\} \rangle$ with $\boldsymbol{\beta}_0 = \boldsymbol{\alpha}_0$, $\boldsymbol{\beta}_\infty = \boldsymbol{\alpha}_\infty$, and $\mathbf{B}_\sigma = \mathbf{O}_\sigma \mathbf{T}$. A *hidden Markov model (HMM)* is a FWA where the weights satisfy (1) $\boldsymbol{\alpha}_0 \geq 0$ with $\boldsymbol{\alpha}_0^\top \mathbf{1} = 1$, (2) $\mathbf{T} \geq 0$ with $\mathbf{T}\mathbf{1} = \mathbf{1}$, and (3) $\boldsymbol{\alpha}_\infty \geq 0$, $\mathbf{O}_\sigma \geq 0$, with $\sum_\sigma \mathbf{O}_\sigma \mathbf{1} + \boldsymbol{\alpha}_\infty = \mathbf{1}$. It can be easily checked that these conditions imply that the WA obtained from a HMM is a PA. For convenience, given a HMM we also define the observation matrix $\mathbf{O} \in \mathbb{R}^{\Sigma \times n}$ with entries $\mathbf{O}(\sigma, i) = \mathbf{O}_\sigma(i, i)$.

Note that unlike with WA, both PA and HMM readily define probability distributions. But this comes at a price: there are stochastic WA realizing probability distributions that cannot be realized by any PA or HMM with a finite number of states (Denis & Esposito, 2008). In terms of representational power, both PA and HMM are equivalent when the number of states is unrestricted. However, in general PA provide more compact representations: given a PA with n states one can always obtain an HMM with $\min\{n^2, n|\Sigma|\}$ states realizing the same distribution. Moreover, there are PA with n states such that every HMM realizing the same distribution needs more than n states (Dupont et al., 2005). These facts imply that different hypothesis classes for learning stochastic languages impose different limitations upon the learned representation.

Given a stochastic language f that assigns probabilities to strings, there are two functions computing aggregate statistics that one can consider: f^{p} for probabilities of prefixes, and f^{s} for expected number of occurrences of substrings. In particular, we have $f^{\mathrm{p}}(x) = f(x\Sigma^\star) = \sum_{y \in \Sigma^\star} f(xy)$, and $f^{\mathrm{s}}(x) = \mathbb{E}_{y \sim f}[|y|_x] = \sum_{y,z \in \Sigma^\star} f(yxz)$. Note that given a sample S of size m generated from f, it is equally easy to estimate the empirical probabilities $\hat{f}_S(x) = (1/m) \sum_{i=1}^m \mathbb{I}[x^i = x]$, as well as empirical prefix probabilities $\hat{f}^{\mathrm{p}}_S(x) = (1/m) \sum_{i=1}^m \mathbb{I}[x^i \in x\Sigma^\star]$ and empirical substring occurrence expectations $\hat{f}^{\mathrm{s}}_S(x) = (1/m) \sum_{i=1}^m |x^i|_x$. It is shown in (Balle et al., 2013) that when f is realized by a WA, PA, or HMM, then so are f^{p} and f^{s}. This result provides an explicit conversion that preserves the number of states and that can be easily reversed. Therefore, in terms of learning algorithms, one can work with any of these three representations indistinctively.

2.3. Learning Latent Variable Models with EM

EM is an iterative algorithm for locally maximizing the non-convex log-likelihood function (Dempster et al., 1977). It alternates between two types of steps: an *expectation* (E) step, where the expected distribution of the hidden variables are computed, and a *maximization* (M) step,

where the parameters of the model are updated by maximizing the joint likelihood of the observed data and the expected hidden variables' distributions. To avoid getting stuck in local maxima, restarts and other heuristics are usually necessary (Hulden, 2012). In practice, given enough time to explore the space of parameters these heuristics yield very competitive models (e.g., Verwer et al., 2012).

3. Learning Algorithms Based on the Method of Moments

The key idea behind method of moments algorithms is to derive equations relating the parameters of some stochastic automaton realizing the target distribution to statistics of the distribution. Then, using estimations of these statistics computed from observed data, one can solve these equations for the parameters of a hypothesis model. In the case of rational stochastic languages, these equations involve mostly linear algebra operations that can be solved using several methods. In this section we describe three algorithms for solving these equations. Our selection is representative of the possible approaches to the method of moments, all of which involve either singular value decompositions, convex optimization, or symmetric tensor decompositions. For ease of presentation, we assume that we have access to the target stochastic language f, which can be used to compute the probability $f(x)$ of any string $x \in \Sigma^\star$. Very few modifications are needed when the algorithms are applied to empirical estimates $\hat{f}_S$ computed from a sample S. We give detailed descriptions of these modifications wherever they are needed.

3.1. Singular Value Decomposition Method

A key step underlying the method of moments algorithms for learning stochastic languages is the arrangement of a finite set of values of f into a matrices or tensors in a way such that spectral factorizations of these linear objects reveal information about the operators of a WA, PA, or HMM realizing f. As a simple example, consider $f = f_A$ for some WA $A = \langle \boldsymbol{\alpha}_0, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\}\rangle$ with n states. Given two sets of strings $\mathcal{P}, \mathcal{S} \subset \Sigma^\star$ which we call prefixes and suffixes, consider the matrix $\mathbf{H} \in \mathbb{R}^{\mathcal{P}\times\mathcal{S}}$ with entries given by $\mathbf{H}(u,v) = f(uv)$. This is the Hankel matrix[3] of f on prefixes $\mathcal{P}$ and suffixes $\mathcal{S}$. Writing $f(u,v) = (\boldsymbol{\alpha}_0^\top \mathbf{A}_u)(\mathbf{A}_v \boldsymbol{\alpha}_\infty)$ we see that this Hankel matrix can be written as $\mathbf{H} = \mathbf{PS}$, where $\mathbf{P} \in \mathbb{R}^{\mathcal{P}\times n}$ with the uth row equal to $\boldsymbol{\alpha}_0^\top \mathbf{A}_u$, and $\mathbf{S} \in \mathbb{R}^{n\times\mathcal{S}}$ with vth column equal to $\mathbf{A}_v \boldsymbol{\alpha}_\infty$. Then it is easy to see that the Hankel matrix $\mathbf{H}_\sigma \in \mathbb{R}^{\mathcal{P}\times\mathcal{S}}$ with entries $\mathbf{H}_\sigma(u,v) = f(u\sigma v)$ for some $\sigma \in \Sigma$ can be written as $\mathbf{H}_\sigma = \mathbf{PA}_\sigma\mathbf{S}$. Thus, a way to recover the operators $\mathbf{A}_\sigma$ of A is to obtain a factorization $\mathbf{H} = \mathbf{PS}$ and use it to solve for $\mathbf{A}_\sigma$ in the expression of $\mathbf{H}_\sigma$. In practice $\mathbf{H}$ is factorized via a singular value decomposition, hence the name spectral method.

We now proceed to give the details of the algorithm, which is based on (Hsu et al., 2009; Bailly et al., 2009). The algorithm computes a minimal WA that approximates f but which, in general, is not stochastic. As input, the method requires sets of prefixes and suffixes $\mathcal{P}, \mathcal{S} \subset \Sigma^\star$, and the number of states n of the target automaton.

The algorithm starts by computing the Hankel matrices $\mathbf{H}, \mathbf{H}_\sigma \in \mathbb{R}^{\mathcal{P}\times\mathcal{S}}$ for each $\sigma \in \Sigma$. It also computes vectors $\mathbf{h}_{\lambda,\mathcal{S}} \in \mathbb{R}^{\mathcal{S}}$ with $\mathbf{h}_{\lambda,\mathcal{S}}(v) = f(v)$ and $\mathbf{h}_{\mathcal{P},\lambda} \in \mathbb{R}^{\mathcal{P}}$ with $\mathbf{h}_{\mathcal{P},\lambda}(u) = f(u)$. Next, it computes the reduced SVD[4] decomposition $\mathbf{H} = \mathbf{UDV}^\top$ with $\mathbf{U} \in \mathbb{R}^{\mathcal{P}\times n}$, $\mathbf{V} \in \mathbb{R}^{\mathcal{S}\times n}$ and diagonal $\mathbf{D} \in \mathbb{R}^{n\times n}$. The algorithm then returns a WA $A = \langle \boldsymbol{\alpha}_0, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\}\rangle$ given by $\boldsymbol{\alpha}_0^\top = \mathbf{h}_{\lambda,\mathcal{S}}^\top \mathbf{V}$, $\boldsymbol{\alpha}_\infty = \mathbf{D}^{-1}\mathbf{U}^\top \mathbf{h}_{\mathcal{P},\lambda}$, and $\mathbf{A}_\sigma = \mathbf{D}^{-1}\mathbf{U}^\top \mathbf{H}_\sigma \mathbf{V}$.

3.2. Convex Optimization Method

This method recovers the operators of a WA by solving an optimization problem involving the sum of a Frobenius norm loss and a trace norm regularizer. As with the SVD method, the learned WA is not stochastic by construction. To motivate the algorithm, recall from the previous section that if f is computed by a WA with n states, then a Hankel matrix of f admits a factorization of the form $\mathbf{H} = \mathbf{PS}$, $\mathbf{P} \in \mathbb{R}^{\mathcal{P}\times n}$, $\mathbf{S} \in \mathbb{R}^{n\times\mathcal{S}}$. Now suppose that $\mathbf{P}$ has rank n. Then, taking $\mathbf{B}_\sigma = \mathbf{PA}_\sigma\mathbf{P}^+ \in \mathbb{R}^{\mathcal{P}\times\mathcal{P}}$ we have $\mathbf{B}_\sigma\mathbf{H} = \mathbf{H}_\sigma$ and $\mathrm{rank}(\mathbf{B}_\sigma) \leq n$. Since the WA given by $B = \langle \boldsymbol{\beta}_0, \boldsymbol{\beta}_\infty, \{\mathbf{B}_\sigma\}\rangle$ with $\boldsymbol{\beta}_0^\top = \boldsymbol{\alpha}_0^\top \mathbf{P}^+$ and $\boldsymbol{\beta}_\infty = \mathbf{P}\boldsymbol{\alpha}_\infty$ satisfies $f_A = f_B$, this motivates an algorithm that looks for a low-rank solution of $\mathbf{MH} = \mathbf{H}_\sigma$.

An algorithm based on this principle is described in (Balle et al., 2012). As input, the method requires sets of prefixes $\mathcal{P}$ and suffixes $\mathcal{S}$ with $\lambda \in \mathcal{P} \cap \mathcal{S}$, and a regularization parameter $\tau > 0$. The number of states of the WA produced by this algorithm is equal to the number of prefixes $|\mathcal{P}|$.

The algorithm starts by computing two Hankel matrices $\mathbf{H} \in \mathbb{R}^{\mathcal{P}\times\mathcal{S}}$ and $\mathbf{H}_\Sigma \in \mathbb{R}^{\mathcal{P}\Sigma\times\mathcal{S}}$, where $\mathbf{H}$ is defined like before, and $\mathbf{H}_\Sigma(u\sigma, v) = f(u\sigma v)$. Note that because we have $\lambda \in \mathcal{P} \cap \mathcal{S}$, now the vectors $\mathbf{h}_{\mathcal{P},\lambda}$ and $\mathbf{h}_{\lambda,\mathcal{S}}$ are contained inside of $\mathbf{H}$. The operators of the hypothesis are obtained by solving the optimization problem

$$\mathbf{A}_\Sigma \in \operatorname*{argmin}_{\mathbf{M}\in\mathbb{R}^{\mathcal{P}\Sigma\times\mathcal{P}}} \|\mathbf{MH} - \mathbf{H}_\Sigma\|_F^2 + \tau\|\mathbf{M}\|_* \ , \qquad (2)$$

and then taking the submatrices $\mathbf{A}_\sigma \in \mathbb{R}^{\mathcal{P}\times\mathcal{P}}$ given by

[3] In real analysis a matrix $\mathbf{M}$ is Hankel if $\mathbf{M}(i,j) = \mathbf{M}(k,l)$ whenever $i+j = k+l$, which implies that $\mathbf{M}$ is symmetric. In our case we have $\mathbf{H}(u,v) = \mathbf{H}(w,z)$ whenever $uv = wz$, but $\mathbf{H}$ is not symmetric because string concatenation is not commutative.

[4] When using an approximation $\hat{\mathbf{H}}$, the algorithm computes the n-truncated SVD instead.

$\mathbf{A}_\sigma(u,u') = \mathbf{A}_\Sigma(u\sigma,u')$. The output automaton is obtained by taking $A = \langle \boldsymbol{\alpha}_0, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\}\rangle$, with the operators recovered from $\mathbf{A}_\Sigma$, $\boldsymbol{\alpha}_0 = \mathbf{e}_\lambda$ the indicator vector corresponding to the empty prefix, and $\boldsymbol{\alpha}_\infty = \mathbf{h}_{\mathcal{P},\lambda}$.

3.3. Symmetric Tensor Decomposition Method

The tensor decomposition method can be applied when the target distribution is generated by a HMM. The idea behind this approach is to observe that when f can be realized by a FWA, then the factorization of the Hankel matrix associated with a symbol $\sigma \in \Sigma$ becomes $\mathbf{H}_\sigma = \mathbf{P}\mathbf{O}_\sigma\mathbf{T}\mathbf{S}$. Since $\mathbf{P}$, $\mathbf{S}$, and $\mathbf{T}$ appear in the decomposition for all σ, and $\mathbf{O}_\sigma$ is diagonal, this implies that under some assumptions on the ranks of these matrices, all the $\mathbf{H}_\sigma$ admit a joint diagonalization. Stacking these matrices together yields a Hankel tensor $\mathbf{H}_{\mathcal{P},\Sigma,\mathcal{S}} \in \mathbb{R}^{\mathcal{P}\times\Sigma\times\mathcal{S}}$ with a particular structure that can be exploited to recover first $\mathbf{O}$, and then the transition matrix $\mathbf{T}$ and the weight vectors $\boldsymbol{\alpha}_0$ and $\boldsymbol{\alpha}_\infty$. The algorithm we describe in this section implements this idea by following the symmetrization and whitening approach of (Anandkumar et al., 2012b). Our presentation is a variant of their method, which extends the method to work with arbitrary sets of prefixes $\mathcal{P}$ and suffixes $\mathcal{S}$, and also is able to recover the set of stopping probabilities. We present a consistency analysis of this variant in the Supplementary Material.

Again, the method needs as input sets of prefixes $\mathcal{P}$ and suffixes $\mathcal{S}$ with $\lambda \in \mathcal{P}\cap\mathcal{S}$, and the number of states n of the target HMM, which must satisfy $n \leq |\Sigma|$. The algorithm proceeds in four stages. In its first stage, the algorithm computes a set of Hankel matrices and tensors. In particular, a third order tensor $\mathbf{H}_{\mathcal{P},\Sigma,\mathcal{S}} \in \mathbb{R}^{\mathcal{P}\times\Sigma\times\mathcal{S}}$ with entries $\mathbf{H}_{\mathcal{P},\Sigma,\mathcal{S}}(u,\sigma,v) = f(u\sigma v)$, a Hankel matrix $\mathbf{H}_{\mathcal{P},\mathcal{S}} \in \mathbb{R}^{\mathcal{P}\times\mathcal{S}}$ with entries $\mathbf{H}_{\mathcal{P},\mathcal{S}}(u,v) = f(uv)$, and a Hankel matrix $\mathbf{H}^{\mathrm{p}}_{\mathcal{P},\Sigma} \in \mathbb{R}^{\mathcal{P}\times\Sigma}$ with entries $\mathbf{H}^{\mathrm{p}}_{\mathcal{P},\Sigma}(u,\sigma) = f(u\sigma\Sigma^\star)$. Integrating over the different dimensions of the tensor $\mathbf{H}_{\mathcal{P},\Sigma,\mathcal{S}}$, the algorithm obtains three more matrices: $\bar{\mathbf{H}}_{\Sigma,\mathcal{S}} \in \mathbb{R}^{\Sigma\times\mathcal{S}}$ with entries $\bar{\mathbf{H}}_{\Sigma,\mathcal{S}}(\sigma,v) = \sum_u f(u\sigma v)$, $\bar{\mathbf{H}}_{\mathcal{P},\mathcal{S}} \in \mathbb{R}^{\mathcal{P}\times\mathcal{S}}$ with entries $\bar{\mathbf{H}}_{\mathcal{P},\mathcal{S}}(u,v) = \sum_\sigma f(u\sigma v)$, and $\bar{\mathbf{H}}_{\mathcal{P},\Sigma} \in \mathbb{R}^{\mathcal{P}\times\Sigma}$ with entries $\bar{\mathbf{H}}_{\mathcal{P},\Sigma}(u,\sigma) = \sum_v f(u\sigma v)$.

The goal of the second stage is to obtain an orthogonal decomposition of a tensor derived from $\mathbf{H}_{\mathcal{P},\Sigma,\mathcal{S}}$ as follows. Assuming $\bar{\mathbf{H}}_{\mathcal{P},\mathcal{S}}$ has rank at least n, the algorithm first finds matrices $\mathbf{Q}_\mathcal{P} \in \mathbb{R}^{n\times\mathcal{P}}$ and $\mathbf{Q}_\mathcal{S} \in \mathbb{R}^{n\times\mathcal{S}}$ such that $\tilde{\mathbf{H}}_{\mathcal{P},\mathcal{S}} = \mathbf{Q}_\mathcal{P}\bar{\mathbf{H}}_{\mathcal{P},\mathcal{S}}\mathbf{Q}_\mathcal{S}^\top$ is invertible, and then computes $\mathbf{N} = \mathbf{Q}_\mathcal{S}^\top\tilde{\mathbf{H}}^{-1}_{\mathcal{P},\mathcal{S}}\mathbf{Q}_\mathcal{P} \in \mathbb{R}^{\mathcal{S}\times\mathcal{P}}$. Combining these, a matrix $\mathbf{X}_\Sigma \in \mathbb{R}^{\Sigma\times\Sigma}$ and a tensor $\mathbf{Y}_\Sigma \in \mathbb{R}^{\Sigma\times\Sigma\times\Sigma}$ are obtained as follows: $\mathbf{X}_\Sigma = \bar{\mathbf{H}}_{\Sigma,\mathcal{S}}\mathbf{N}\bar{\mathbf{H}}_{\mathcal{P},\Sigma}$, and $\mathbf{Y}_\Sigma = \mathbf{H}_{\mathcal{P},\Sigma,\mathcal{S}}(\mathbf{N}^\top\tilde{\mathbf{H}}^\top_{\Sigma,\mathcal{S}},\mathbf{I},\mathbf{N}\bar{\mathbf{H}}_{\mathcal{P},\Sigma})$. One can show that both $\mathbf{X}_\Sigma$ and $\mathbf{Y}_\Sigma$ are symmetric.[5] Then, assuming $\mathbf{X}_\Sigma$ is positive definite of rank at least n, we can find $\mathbf{W} \in \mathbb{R}^{\Sigma\times n}$ such that $\mathbf{W}^\top\mathbf{X}_\Sigma\mathbf{W} = \mathbf{I}$. This is used to whiten the tensor $\mathbf{Y}_\Sigma$ by taking $\mathbf{Z}_\Sigma = \mathbf{Y}_\Sigma(\mathbf{W},\mathbf{W},\mathbf{W}) \in \mathbb{R}^{n\times n\times n}$. Next we compute the robust orthogonal eigendecomposition $\mathbf{Z}_\Sigma = \sum_{i\in[n]}\gamma_i\mathbf{z}_i^{\otimes 3}$ using a power method for tensors similar to that used to compute eigendecompositions of matrices (Anandkumar et al., 2012b). Using these robust eigenpairs $(\gamma_i,\mathbf{z}_i)$ we build a matrix $\tilde{\mathbf{O}} \in \mathbb{R}^{\Sigma\times n}$ whose ith column is $\gamma_i(\mathbf{W}^\top)^+\mathbf{z}_i$. After a normalization operation, this will be the observation matrix of the output model.

The third stage recovers the rest of parameters (up to normalization) via a series of matrix manipulations. Let $\tilde{\mathbf{O}}_\mathcal{P} = \bar{\mathbf{H}}_{\mathcal{P},\Sigma}(\tilde{\mathbf{O}}^\top)^+ \in \mathbb{R}^{\mathcal{P}\times n}$ and $\tilde{\mathbf{O}}_\mathcal{S}^\top = \tilde{\mathbf{O}}^+\bar{\mathbf{H}}_{\Sigma,\mathcal{S}} \in \mathbb{R}^{n\times\mathcal{S}}$. We start by taking $\tilde{\boldsymbol{\alpha}}_0^\top = \mathbf{e}_\lambda^\top\tilde{\mathbf{O}}_\mathcal{P}$ and $\tilde{\boldsymbol{\alpha}}_\infty = \tilde{\mathbf{O}}_\mathcal{S}^\top\mathbf{e}_\lambda$: respectively, the rows of $\tilde{\mathbf{O}}_\mathcal{P}$ and $\tilde{\mathbf{O}}_\mathcal{S}$ corresponding to λ. Similarly, the algorithm computes $\tilde{\mathbf{T}} = \tilde{\mathbf{O}}_\mathcal{P}^+\bar{\mathbf{H}}_{\mathcal{P},\mathcal{S}}\mathbf{H}^+_{\mathcal{P},\mathcal{S}}\tilde{\mathbf{O}}_\mathcal{P} \in \mathbb{R}^{n\times n}$.

In the last stage the model parameters are normalized as follows. Let $\tilde{\mathbf{D}}_\gamma = \mathrm{diag}(\gamma_1^2,\ldots,\gamma_n^2) \in \mathbb{R}^{n\times n}$ and $\tilde{\mathbf{D}}_\mathcal{S} = \tilde{\mathbf{O}}^\top{\mathbf{H}^{\mathrm{p}}_{\mathcal{P},\Sigma}}^+\tilde{\mathbf{O}}_\mathcal{P} \in \mathbb{R}^{n\times n}$. To obtain $\boldsymbol{\alpha}_\infty$ we first compute $\boldsymbol{\beta} = \tilde{\mathbf{D}}_\mathcal{S}\tilde{\mathbf{T}}^+\tilde{\mathbf{D}}_\gamma\tilde{\boldsymbol{\alpha}}_\infty \in \mathbb{R}^n$ and then let $\boldsymbol{\alpha}_\infty(i) = \boldsymbol{\beta}(i)/(1+\boldsymbol{\beta}(i))$. The initial weights are obtained as $\boldsymbol{\alpha}_0^\top = \tilde{\boldsymbol{\alpha}}_0^\top\tilde{\mathbf{D}}_\Sigma^+\tilde{\mathbf{D}}_\mathcal{S}^+$, where $\tilde{\mathbf{D}}_\Sigma = \mathbf{I} - \mathrm{diag}(\boldsymbol{\alpha}_\infty)$. Finally, we let $\mathbf{O} = \tilde{\mathbf{O}}\tilde{\mathbf{D}}_\Sigma$ and $\mathbf{T} = \tilde{\mathbf{D}}_\mathcal{S}\tilde{\mathbf{T}}\tilde{\mathbf{D}}_\mathcal{S}^+\tilde{\mathbf{D}}_\Sigma^+$. When working with empirical approximations these matrices are not guaranteed to satisfy the requirements in the definition of a HMM. In this case, a last step is necessary to enforce the constraints by projecting the parameters into the simplex (Duchi et al., 2008).

One important benefit of the symmetric tensor decomposition approach is that, since it returns proper HMM parameters, it can be used to initialize other optimization algorithms (e.g., maximum likelihood methods such as EM).

3.4. Statistical Guarantees

Under some natural assumptions, these three methods are known to be consistent. Assuming the Hankel matrices and tensors are computed from a WA with n states, the SVD and convex optimization methods are consistent whenever $\mathbf{H}$ has rank n (Hsu et al., 2009; Balle et al., 2012). To guarantee the consistency of the tensor decomposition method we require that f can be realized by some HMM with $n \leq |\Sigma|$ states and $\bar{\mathbf{H}}_{\mathcal{P},\mathcal{S}}$ has rank n (see (Anandkumar et al., 2012b) and the proof in the Supplementary Material). We note that in most cases, whether these conditions are satisfied or not may depend on our choice of $\mathcal{P}$ and $\mathcal{S}$.

[5] When working with approximate Hankel matrices and tensors this is not necessarily true. Thus one needs to consider the symmetrized versions $(\mathbf{X}_\Sigma + \mathbf{X}_\Sigma^\top)/2$ and $\sum_\rho \mathbf{Y}_\Sigma^\rho/6$, where the sum is taken over all the permutations ρ of $\{1,2,3\}$.

Finite sample bounds can also be obtained for some of these methods, which show that the error of method of moments typically decreases at a parametric rate $O_p(m^{-1/2})$. Moreover, explicit dependencies on several task-related parameters can be obtained with these types of analyses. In the case of stochastic languages, one obtains bounds that depend polynomially on the number of states n, the alphabet size $|\Sigma|$, the dimensions of the Hankel matrices $\sqrt{|\mathcal{P}||\mathcal{S}|}$, and the nth singular value of $\mathbf{H}$ (see Hsu et al., 2009; Anandkumar et al., 2012b; Hsu & Kakade, 2013; Balle, 2013, for explicit bounds and further pointers) .

4. Empirical Comparison

We compare the empirical performance of the methods described above using two contrasting performance metrics on synthetic data and a real-world natural language processing (NLP) task. The goal of this analysis is to elucidate the performance and implementation tradeoffs between the different moment-based methods, while comparing them to a state-of-the-art EM baseline.

4.1. Methods Compared

The following methods are compared: *Spec-Str*, the spectral method (3.1) applied to empirical estimates $\hat{f}_S$; *Spec-Sub*, the spectral method (3.1) applied to empirical estimates $\hat{f}^{\mathrm{s}}_S$; *CO*, the convex optimization method (3.2) applied to empirical estimates $\hat{f}^{\mathrm{s}}_S$ and using the alternating direction method of multipliers (ADMM) (Boyd et al., 2011) optimization algorithm; *Tensor*, the symmetric tensor decomposition method using the tensor-power method (Anandkumar et al., 2012b) (3.3) applied to empirical estimates $\hat{f}_S$; *EM*, an optimized implementation[6] of the Baum-Welch algorithm (Dempster et al., 1977); *EM-Tensor*, the EM method initialized with the solution from the tensor method.

Versions of the spectral method using both string ($\hat{f}_S$) and substring ($\hat{f}^{\mathrm{s}}_S$) estimates are included in order to demonstrate a basic tradeoff: using $\hat{f}^{\mathrm{s}}_S$ maximizes the amount of information extracted from training data but leads to dense Hankel matrices while using $\hat{f}_S$ leads to sparse Hankel matrix estimates. The CO method uses $\hat{f}^{\mathrm{s}}_S$ as its complexity scales with the size of the Hankel matrix and the optimization routines do not preserve sparsity. The tensor method uses $\hat{f}_S$, as our implementation relies on exploiting the sparsity of the Hankel estimates.

Table 1 summarizes important characteristic and implementation details for the different moment-based methods.[7]

[6]The Treba EM library (Hulden, 2012) was used.

[7]The (subjective) implementation difficulty simply highlights significant disparities. Code lengths do not include libraries.

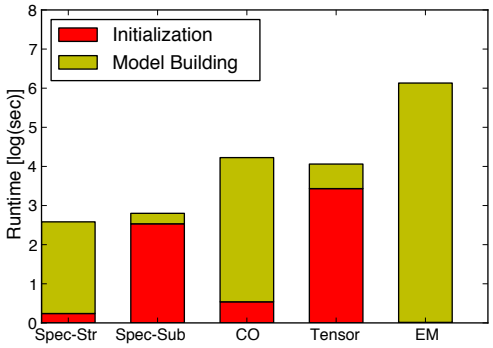

Figure 1. Runtimes (log scale) for initializing and building models of size 10, 20, and 30 on a synthetic 12-symbol domain.

4.2. Performance Metrics

We examine two contrasting performance metrics: one based upon perplexity and the other on word-error-rate (WER). We define the perplexity of a model $\mathcal{M}$ on a test set $\mathcal{T}$ as: $\mathrm{Per}(\mathcal{M}) = 2^{-\sum_{x\in\mathcal{T}} p_*(x)\log(p_{\mathcal{M}}(x))}$, where $p_*(x)$ is the true probability of the string (estimated as its empirical frequency in the test set if necessary) and $p_{\mathcal{M}}(x)$ is the probability assigned to the string by the model. For the spectral methods and the CO method, the models are not guaranteed to return true probabilities, so thresholding to (0,1] is employed (see Cohen et al., 2013, for a discussion of other possible heuristics). In contrast, the WER metric measures the fraction of incorrectly predicted symbols when, for each prefix of strings in the test set, the most likely next symbol is predicted.

4.3. Hyper-parameter optimization

To ensure a fair comparison, the only hyper-optimization performed for each domain was a search for the best model-size, or in the case of CO, a search for τ. The search-schedule was fixed across all domains but was not uniform across all methods. In particular, (1) EM is orders-of-magnitude slower than the other methods (see Figure 1), necessitating a more coarse-grained search, and (2) for tensor decomposition the space of model-sizes is upper-bounded by $|\Sigma|$, restricting the search. Further details can be found in the Supplementary Material.

4.4. Synthetic Experiments

We tested the algorithms on 12 synthetic problems taken from the set of problems used in the PAutomaC competition (Verwer et al., 2012). These domains were selected to represent a diverse sampling of possible target stochastic languages. The different classes of models used to generate the datasets are HMM, PA, and DPA. We selected four models from each of these classes, such that within each class exactly half of the problems have the property that $n < |\Sigma|$. Table A.1 in (Verwer et al., 2012) provides de-

Table 1. Characterizing the different moment-based methods according to their time complexities (for n-state models) and implementation difficulty. Advanced techniques used to make the algorithms scalable and improve performance are highlighted.

Method	Implementation Difficulty	Advanced Techniques	Time Complexity	Returns PA
Spectral	EASY ($\approx$ 100-200 Python lines)	• Feature-variance normalizing (Cohen et al., 2013). • Randomized SVD (Halko et al., 2011).	$O(n\|\Sigma\|\|\mathcal{P}\|\|\mathcal{S}\|)$	No
CO	HARD ($\approx$ 1000-1250 Python + C++ lines)	• ADMM (with over-relaxation and inexact proximal operators) (Boyd et al., 2011). • Randomized SVD.	$O(\|\Sigma\|\|\mathcal{P}\|^3)$ per iteration	No
Tensor	MEDIUM ($\approx$ 750-1000 Python + C++ lines)	• Sparse LSQR (Paige & Saunders, 1982). • Simplex projection (Duchi et al., 2008).	$O((\|\Sigma\|+n)\|\mathcal{P}\|\|\mathcal{S}\|+Rn^2)$, where R is # of tensor-power iterations	Yes

Table 2. Performance of difference methods on synthetic data in terms of WER. Model sizes (or the τ order) are listed in parentheses. The WER of the true model use to generate the data is included for comparison. Note that the WER of a learned model can be lower than that of the true model on the test data, as WER relies on scores not probability assignments.

Type	ID	$\|\Sigma\| \geq n$	Spec-Str	Spec-Sub	CO	Tensor	EM	EM-Tensor	True
HMM	1	✗	80.3 (72)	**71.3** (31)	89.8 (10^{-5})	86.4 (3)	75.7 (10)	78.2 (3)	68.8 (63)
	14	✗	70.0 (6)	70.2 (10)	85.2 (10^{-5})	89.4 (3)	**68.6** (20)	75.5 (3)	68.4 (15)
	33	✓	78.7 (3)	76.7 (3)	95.3 (10^{-3})	83.6 (3)	**74.3** (20)	76.7 (3)	74.1 (13)
	45	✓	80.2 (2)	80.1 (10)	90.1 (10^{-5})	87.9 (3)	78.1 (10)	**70.1** (3)	78.1 (14)
PA	29	✗	65.7 (70)	**47.3** (41)	67.1 (10^{-3})	88.7 (3)	49.2 (40)	74.1 (3)	47.2 (36)
	39	✓	66.2 (32)	**62.0** (30)	83.4 (10^{-5})	93.2 (4)	63.3 (30)	71.3 (4)	59.3 (6)
	43	✗	83.4 (10)	78.0 (12)	89.2 (10^{-5})	89.3 (3)	**77.4** (40)	78.1 (3)	77.1 (67)
	46	✓	90.7 (20)	79.3 (20)	95.7 (10^{-5})	95.5 (4)	**77.5** (40)	80.4 (4)	77.3 (19)
DPA	6	✗	62.0 (20)	50.2 (68)	78.1 (10^{-5})	59.7 (5)	**47.4** (40)	59.7 (5)	46.9 (19)
	7	✓	95.7 (10)	50.6 (20)	66.0 (10^{-3})	81.4 (4)	**48.1** (40)	87.2 (4)	48.3 (12)
	27	✗	83.1 (50)	**75.5** (22)	91.3 (10^{-4})	92.1 (7)	83.0 (10)	83.8 (7)	73.0 (19)
	42	✓	67.8 (40)	61.4 (16)	73.9 (10^{-4})	92.5 (5)	**58.1** (40)	75.7 (5)	56.6 (6)

tailed descriptions of these different problems.

Table 2 summarizes the performance of the different methods in terms of WER and Table 3 the performance in terms of perplexity.

Several conclusions can be drawn from these results. First, we note that Spec-Sub and EM are the top performers in terms of WER. And though EM outperforms Spec-Sub on a majority of domains, their performance is quite close (compared to the gap between those two and the other algorithms), while Spec-Sub is 40x faster. In terms of perplexity EM and CO are the top-performers; EM performs best on a majority of domains but with a large penalty in terms of runtime. We also observe that the tensor decomposition initialized EM method usually outperformed the tensor method alone in terms of WER; in terms of perplexity its behavior was more erratic.

Figure 1 summarizes a representative example of the runtime costs of the algorithms, distinguishing between initialization and model-building phases. For the moment-methods, the Hankel matrices (and their spectral decompositions) only need to be computed once prior to hyper-parameter optimization. This represents a major advantage of the moment-based methods compared to EM, where no computation results are reused.

4.5. Natural Language Processing Problem

In addition to synthetic experiments, we compared the methods performance on a NLP task. In this task the methods were used to learn a model of the parts-of-speech (POS) tags from the Penn-Treebank Corpus (Marcus et al., 1993). There are 11 distinct POS tags in the task, so $|\Sigma| = 11$.

Table 4 summarizes the performance of the algorithms on the NLP data. Here, we again see that EM is the top performer in terms of WER with the Spec-Sub method performing only slightly worse. In terms of perplexity the CO method outperforms all the other methods by a large margin. Overall, the results on the NLP data reinforce the conclusions reached via the synthetic experiments.

5. Discussion

In this work we provided a unified presentation of three methods of moments for learning probabilistic models over stochastic languages. Beyond providing a solid and unified foundation for cross-method comparisons, this presentation also extended the symmetric tensor decomposition method to work with arbitrary prefix and suffix bases.

With this foundation in place we discussed several concrete instantiations of these approaches, highlighting implementation techniques that are necessary for scalability and performance, and we empirically compared these methods, along with an EM baseline, on both synthetic and real-world data. The synthetic experiments elucidated several important performance trends, which we hope will serve as aids for future research in this area and as impetus for the adoption of moment-based methods in applied settings.

Table 3. Performance of difference methods on synthetic data in terms of perplexity. Model sizes (or the τ order) are listed in parentheses. The perplexity of the true model use to generate the data is included for comparison.

Type	ID	$\lvert\Sigma\rvert \geq n$	Spec-Str	Spec-Sub	CO	Tensor	EM	EM-Tensor	True
HMM	1	✗	$\approx 10^9$ (40)	441.07 (8)	**44.77** (10^{-5})	66.8 (3)	500.1 (30)	126.14 (3)	29.90 (63)
	14	✗	306.30 (1)	133.71 (30)	128.53 (10^{-3})	253.44 (3)	**116.84** (20)	133.09 (3)	116.80 (15)
	33	✓	51.14 (3)	49.22 (4)	57.05 (10^{-5})	60.04 (3)	**32.14** (10)	32.21 (3)	31.87 (13)
	45	✓	$\approx 10^6$ (70)	**31.87** (19)	36.6 (10^{-5})	40.47 (7)	107.75 (40)	62.24 (7)	24.04 (14)
PA	29	✗	$\approx 10^5$ (41)	39.32 (49)	34.57 (10^{-4})	$\approx 10^{14}$ (3)	**25.09** (40)	80.55 (3)	24.03 (36)
	39	✓	8170.69 (8)	42.8 (30)	11.24 (10^{-2})	22.42 (5)	**10.43** (5)	11.34 (5)	10.00 (6)
	43	✗	$\approx 10^6$ (2)	115.62 (4)	**36.61** (10^{-2})	36.90 (3)	461.23 (40)	56.94 (3)	32.64 (67)
	46	✓	44.12 (2)	34.51 (26)	25.28 (10^{-5})	32.1 (4)	**12.02** (40)	14.26 (4)	11.98 (19)
DPA	6	✗	$\approx 10^9$ (56)	95.12 (40)	104.68 (10^{-5})	691.28 (3)	**67.32** (40)	247.07 (3)	66.96 (19)
	7	✓	999.83 (4)	70.24 (48)	62.74 (10^{-5})	664.53 (5)	**51.27** (40)	$\approx 10^{16}$ (5)	51.22 (12)
	27	✗	$\approx 10^6$ (21)	238.48 (19)	102.85 (10^{-5})	212.22 (4)	94.90 (40)	**71.63** (4)	42.43 (19)
	42	✓	$\approx 10^5$ (12)	59.43 (31)	**23.91** (10^{-2})	39.00 (5)	168.52 (40)	292.90 (5)	16.00 (6)

Table 4. Performance of difference methods on 11-symbol NLP dataset in terms of both WER and perplexity. Model sizes (or the τ order) are listed in parentheses.

Metric	Spec-Str	Spec-Sub	CO	Tensor	EM	EM-Tensor
WER	66.6 (38)	60.43 (46)	84.7 (10^{-5})	85.3 (3)	**59.8** (20)	62.6 (3)
Perplexity	$4.03 \cdot 10^{15}$ (70)	$1.00 \cdot 10^{14}$ (9)	$\mathbf{4.15 \cdot 10^9}$ (10^{-3})	$4.99 \cdot 10^{14}$ (7)	$1.33 \cdot 10^{13}$ (2)	$2.43 \cdot 10^{13}$ (7)

The NLP experiment demonstrates that these trends appear to carry over to noisy real-world settings.

With respect to the WER metric, the experiments demonstrated that the Spec-Sub method produces WER results competitive with EM with a speed-up factor of 40x. This makes Spec-Sub an attractive candidate for applications requiring low WERs, given its combination of speed, simplicity, and accuracy.

With respect to model perplexity, the experiments demonstrated that the convex relaxation of the spectral method can produce highly accurate models. The good performance of CO in this setting is intriguing given its relatively poor WER performance and the fact that the algorithm is not guaranteed to return a PA (in contrast to the tensor and EM methods).

It is should be noted, however, that EM was the top-performer on a majority of domains; though it is considerably more expensive in terms of runtime. This is an important finding as it demonstrates that an optimized implementation of EM with random restarts is not significantly disadvantaged by the issue of local-minima and that the primary drawback of EM compared to the moment-methods is its computational inefficiency, especially on large state spaces.

Our experiments also elucidate other interesting properties of the methods. First, the results demonstrate that the advantage of using $\hat{f}^{\mathrm{s}}_S$ estimates, which lead to dense Hankel estimates and extract more information from the training sample (compared to $\hat{f}$ estimates), is quite pronounced in the case of the spectral method. In addition, these experiments highlight the unpredictable nature of initializing EM with a moment-based solution.

Of course, for the sake of clarity of presentation and analysis, this empirical comparison did exclude certain settings and methods. We did not examine the effect of large or continuous alphabets, as in those cases the performance of moment-methods are contingent upon the feature representation or kernel embedding employed (Song et al., 2010; Boots et al., 2013). And we did not examine different methods of choosing prefix and suffix bases. However, both these issues are largely orthogonal to the core learning problem, as we expect all three moment-methods to benefit equally from feature-representations and basis selections.

That said, our analysis offers a clear picture of the current empirical state-of-the-art in moment-based methods for modelling stochastic languages. This work also raises a number of important directions and open questions for future work. Of particular interest are problems such as: (1) establishing theoretical justification for CO's strong performance on the perplexity metric; (2) relaxing the theoretical constraints of tensor-based methods; and (3) developing algorithms in which moment-initialized maximum likelihood optimization is guaranteed to improve solutions.

Acknowledgments

The authors are grateful to Animashree Anandkumar, Furong Huang, Percy Liang, and Andreu Mayo for sharing their implementations of some of the algorithms described in this paper. Financial support for this research was provided by the NSERC Discovery and CGS-M programs, and the James McGill Research Fund.

References

Anandkumar, A., Foster, D. P., Hsu, D., Kakade, S. M., and Liu, Y. A spectral algorithm for latent Dirichlet allocation. In *NIPS*, 2012a.

Anandkumar, A., Ge, R., Hsu, D., Kakade, S. M., and Telgarsky, M. Tensor decompositions for learning latent variable models. *CoRR*, abs/1210.7559, 2012b.

Anandkumar, A., Hsu, D., Huang, F., and Kakade, S. M. Learning mixtures of tree graphical models. In *NIPS*, 2012c.

Anandkumar, A., Hsu, D., and Kakade, S. M. A method of moments for mixture models and hidden Markov models. In *COLT*, 2012d.

Anandkumar, A., Ge, R., Hsu, D., and Kakade, S. A tensor spectral approach to learning mixed membership community models. In *COLT*, 2013.

Bailly, R., Denis, F., and Ralaivola, L. Grammatical inference as a principal component analysis problem. In *ICML*, 2009.

Bailly, R., Habrard, A., and Denis, F. A spectral approach for probabilistic grammatical inference on trees. In *ALT*, 2010.

Bailly, R., Carreras, X., Luque, F., and Quattoni, A. Unsupervised spectral learning of WCFG as low-rank matrix completion. In *EMNLP*, 2013a.

Bailly, R., Carreras, X., and Quattoni, A. Unsupervised spectral learning of finite state transducers. In *NIPS*, 2013b.

Balle, B. *Learning Finite-State Machines: Algorithmic and Statistical Aspects*. PhD thesis, Universitat Politècnica de Catalunya, 2013.

Balle, B. and Mohri, M. Spectral learning of general weighted automata via constrained matrix completion. In *NIPS*, 2012.

Balle, B., Quattoni, A., and Carreras, X. A spectral learning algorithm for finite state transducers. In *ECML-PKDD*, 2011.

Balle, B., Quattoni, A., and Carreras, X. Local loss optimization in operator models: A new insight into spectral learning. In *ICML*, 2012.

Balle, B., Carreras, X., Luque, F.M., and Quattoni, A. Spectral learning of weighted automata: A forward-backward perspective. *Machine Learning*, 2013.

Boots, B., Siddiqi, S., and Gordon, G. Closing the learning planning loop with predictive state representations. *International Journal of Robotics Research*, 2011.

Boots, B., Gretton, A., and Gordon, G. Hilbert space embeddings of predictive state representations. In *UAI*, 2013.

Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 2011.

Chaganty, A. T. and Liang, P. Spectral experts for estimating mixtures of linear regressions. In *ICML*, 2013.

Cohen, S. B., Stratos, K., Collins, M., Foster, D. P., and Ungar, L. Spectral learning of latent-variable PCFGs. *ACL*, 2012.

Cohen, S. B., Stratos, K., Collins, M., Foster, D. P., and Ungar, L. Experiments with spectral learning of latent-variable PCFGs. In *NAACL-HLT*, 2013.

Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 1977.

Denis, F. and Esposito, Y. On rational stochastic languages. *Fundamenta Informaticae*, 2008.

Dhillon, P. S., Rodu, J., Collins, M., Foster, D. P., and Ungar, L. H. Spectral dependency parsing with latent variables. In *EMNLP-CoNLL*, 2012.

Duchi, John, Shalev-Shwartz, Shai, Singer, Yoram, and Chandra, Tushar. Efficient projections onto the L_1-ball for learning in high dimensions. In *ICML*, 2008.

Dupont, P., Denis, F., and Esposito, Y. Links between probabilistic automata and hidden Markov models: probability distributions, learning models and induction algorithms. *Pattern Recognition*, 2005.

Halko, N., Martinsson, P., and Tropp, J. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

Hamilton, W.L., Fard, M.M., and Pineau, J. Modelling sparse dynamical systems with compressed predictive state representations. In *ICML*, 2013.

Hsu, D., Kakade, S. M., and Zhang, T. A spectral algorithm for learning hidden Markov models. In *COLT*, 2009.

Hsu, Daniel and Kakade, Sham M. Learning mixtures of spherical Gaussians: moment methods and spectral decompositions. In *ITCS*, 2013.

Hulden, M. Treba: Efficient numerically stable EM for PFA. In *ICGI*, 2012.

Luque, F.M., Quattoni, A., Balle, B., and Carreras, X. Spectral learning in non-deterministic dependency parsing. *EACL*, 2012.

Marcus, M., Marcinkiewicz, M., and Santorini, B. Building a large annotated corpus of english: The Penn Treebank. *Computational linguistics*, 19(2):313–330, 1993.

Paige, C. and Saunders, M. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, 8(1):43–71, 1982.

Parikh, A. P., Song, L., and Xing, E.P. A spectral algorithm for latent tree graphical models. In *ICML*, 2011.

Pearson, K. Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London*, 1894.

Recasens, A. and Quattoni, A. Spectral learning of sequence taggers over continuous sequences. In *ECML-PKDD*, 2013.

Siddiqi, S. M., Boots, B., and Gordon, G. Reduced-rank hidden Markov models. In *AISTATS*, 2010.

Song, L., Boots, B., Siddiqi, S., Gordon, G., and Smola, A. Hilbert space embeddings of hidden Markov models. In *ICML*, 2010.

Song, L., Ishteva, M., Parikh, A., Xing, E., and Park, H. Hierarchical tensor decomposition of latent tree graphical models. In *ICML*, 2013.

Stratos, K., Rush, A. M., Cohen, S., and Collins, M. Spectral learning of refinement hmms. In *CoNLL*, 2013.

Varin, C. On composite marginal likelihoods. *Advances in Statistical Analysis*, 2008.

Verwer, S., Eyraud, R., and Higuera, C. Results of the PAutomaC probabilistic automaton learning competition. In *ICGI*, 2012.