Spectral Learning of Weighted Automata

A Forward-Backward Perspective

Borja Balle[1], Xavier Carreras[1], Franco M. Luque[2], and Ariadna Quattoni[1]

[1]Universitat Politècnica de Catalunya, Barcelona, Spain
[2]Universidad Nacional de Córdoba and CONICET, Córdoba, Argentina
{bballe,carreras,aquattoni}@lsi.upc.edu
francolq@famaf.unc.edu.ar

Submitted: December 5th, 2012
Revised: June 30th, 2013

Abstract

In recent years we have seen the development of efficient provably correct algorithms for learning Weighted Finite Automata (WFA). Most of these algorithms avoid the known hardness results by defining parameters beyond the number of states that can be used to quantify the complexity of learning automata under a particular distribution. One such class of methods are the so-called spectral algorithms that measure learning complexity in terms of the smallest singular value of some Hankel matrix. However, despite their simplicity and wide applicability to real problems, their impact in application domains remains marginal to this date. One of the goals of this paper is to remedy this situation by presenting a derivation of the spectral method for learning WFA that – without sacrificing rigor and mathematical elegance – puts emphasis on providing intuitions on the inner workings of the method and does not assume a strong background in formal algebraic methods. In addition, our algorithm overcomes some of the shortcomings of previous work and is able to learn from statistics of substrings. To illustrate the approach we present experiments on a real application of the method to natural language parsing.

1 Introduction

Learning finite automata is a fundamental task in Grammatical Inference. Over the years, a multitude of variations on this problem have been studied. For example, several learning models with different degrees of realism have been considered, ranging from query models and the learning in the limit paradigm, to the more challenging PAC learning framework. The main differences between these models are the ways in which learning algorithms can interact with the target machine. But not only the choice of learning model makes a difference in the study of this task, but also the particular kind of target automata that must be learned. These can range from the classical acceptors for regular languages like Deterministic Finite Automata (DFA) and Non-deterministic Finite Automata (NFA), to the more general Weighted Finite Automata

(WFA) and Multiplicity Automata (MA), while also considering intermediate case like several classes of Probabilistic Finite Automata (PFA).

Efficient algorithms for learning all these classes of machines have been proposed in query models where algorithms have access to a minimal adequate teacher. Furthermore, most of these learning problems are also known to have polynomial information-theoretic complexity in the PAC learning model. But despite these encouraging results, it has been known for decades that the most basic problems regarding learnability of automata in the PAC model are computationally untractable under both complexity-theoretic and cryptographic assumptions. Since these general worst-case results preclude the existence of efficient learning algorithms for all machines under all possible probability distributions, lots of efforts have been done in identifying problems involving special cases for which provably efficient learning algorithms can be given. An alternative approach has been to identify additional parameters beyond the number of states that can be used to quantify the complexity of learning a particular automaton under a particular distribution. A paradigmatic example of this line of work are the PAC learning algorithms for PDFA given in (Ron et al, 1998; Clark and Thollard, 2004; Palmer and Goldberg, 2007; Castro and Gavaldà, 2008; Balle et al, 2012a) whose running time depend on a distinguishability parameter quantifying the minimal distance between distributions generated by different states in the target machine.

Spectral learning methods are a family of algorithms that also fall into this particular line of work. In particular, starting with the seminal works of Hsu et al (2009) and Bailly et al (2009), efficient provably correct algorithms for learning non-deterministic machines that define probability distributions over sets of strings have been recently developed. A work-around to the aforementioned hardness results is obtained in this case by including the smallest singular value of some Hankel matrix in the bounds on the running time of spectral algorithms. The initial enthusiasm generated by such algorithms has been corroborated by the appearance of numerous follow-ups devoted to extending the method to more complex probabilistic models. However, despite the fact that these type of algorithms can be used to learn classes of machines widely used in applications like Hidden Markov Models (HMM) and PNFA, the impact of these methods in application domains remains marginal to this date. This remains so even when implementing such methods involves just a few linear algebra operations available in most general mathematical computing software packages. One of the main purposes of this paper is to try to remedy this situation by providing practical intuitions around the foundations of these algorithms and clear guidelines on how to use them in practice.

In our opinion, a major cause for the gap between the theoretical and practical development of spectral methods is the overwhelmingly theoretical nature of most papers in this area. The state of the art seems to suggest that there is no known workaround to these long mathematical proofs when seeking PAC learning results. However, it is also the case that most of the times the derivations given for these learning algorithms provide no intuitions on why or how one should expect them to work. Thus, obliterating the matter of PAC bounds, our first contribution is to provide a new derivation of the spectral learning algorithm for WFA that stresses the main intuitions behind the method. This yields an efficient algorithm for learning stochastic WFA defining probability distributions over strings. Our second contribution is showing how a simple transformation of this algorithm yields a more sample-efficient learning method that can work with substring statistics in contrast to the usual prefix statistics used in other methods.

Finite automata can also be used as building blocks for constructing more general context-free grammatical formalisms. In this paper we consider the case of non-deterministic Split Head-Automata Grammars (SHAG). These are a family of hidden-state parsing models that have been successfully used to model the significant amount of non-local phenoma exhibited by dependency structures in natural language. A SHAG is composed by a collection of stochastic automata and can be used to define a probability distribution over dependency structures for a given sentence. Each automaton in a SHAG describes the generation of particular head-modifier sequences. Our third contribution is to apply the spectral method to the problem of learning the constituent automata of a target SHAG. Contrary to previous works where PDFA were used as basic constituent automata for SHAG, using the spectral method allows us to learn SHAG built out of non-deterministic automata.

1.1 Related Work

In the last years multiple spectral learning algorithms have been proposed for a wide range of models. Many of these models deal with data whose nature is emminently sequential, like the work of (Bailly et al, 2009) on WFA, or other works on particular subclasses of WFA like HMM (Hsu et al, 2009) and related extensions (Siddiqi et al, 2010; Song et al, 2010), Predictive State Representations (PSR) (Boots et al, 2011), Finite State Transducers (FST) (Balle et al, 2011), and Quadratic Weighted Automata (QWA) (Bailly, 2011). Besides direct applications of the spectral algorithm to different classes of sequential models, the method has also been combined with convex optimization algorithms in (Balle et al, 2012b; Balle and Mohri, 2012).

Despite this overwhelming diversity, to our knowledge the only previous work that has considered spectral learning for the general class of probabilistic weighted automata is due to Bailly et al (2009). In spirit, their technique for deriving the spectral method is similar to ours. However, their elegant mathematical derivations are presented assuming a target audience with a strong background on formal algebraic methods. As such their presentation lacks the intuitions necessary to make the work accessible to a more general audience of machine learning practitioners. In contrast – without sacrificing rigor and mathematical elegance – our derivations put emphasis on providing intuitions on the inner working of the spectral method.

Besides sequential models, spectral learning algorithms for tree-like structures appearing in context-free grammatical models and probabilistic graphical models have also been considered (Bailly et al, 2010; Parikh et al, 2011; Luque et al, 2012; Cohen et al, 2012; Dhillon et al, 2012). In section 6.4 we give a more detailed comparision between our work on SHAG and related methods that learn tree-shaped models. The spectral method has been applied as well to other classes of probabilistic mixture models (Anandkumar et al, 2012c,a).

2 Weighted Automata and Hankel Matrices

In this section we present Weighted Finite Automata (WFA), the finite state machine formulations that will be used throughout the paper. We begin by introducing some notation for dealing with functions from strings to real numbers and then proceed to define Hankel matrices. These matrices will play a very important role in the derivation of the spectral learning algorithm given in Section 4. Then we proceed to describe the algebraic formulation of WFA and its relation to Hankel matrices. Finally, we discuss some special properties of stochastic WFA realizing probability distributions over strings. These properties will allow us to use the spectral method to learn from substring statistics, thus yielding more sample-efficient methods than other approaches based on string or prefix statitics.

2.1 Functions on Strings and their Hankel Matrices

Let Σ be a finite alphabet. We use σ to denote an arbitrary symbol in Σ. The set of all finite strings over Σ is denoted by $\Sigma^\star$, where we write λ for the empty string. We use bold letters to represent vectors $\mathbf{v}$ and matrices $\mathbf{M}$. We use $\mathbf{M}^+$ to denote the *Moore–Penrose pseudoinverse* of some matrix $\mathbf{M}$.

Let $f : \Sigma^\star \to \mathbb{R}$ be a function over strings. The *Hankel matrix* of f is a bi-infinite matrix $\mathbf{H}_f \in \mathbb{R}^{\Sigma^\star \times \Sigma^\star}$ whose entries are defined as $\mathbf{H}_f(u, v) = f(uv)$ for any $u, v \in \Sigma^\star$. That is, rows are indexed by prefixes and columns by suffixes. Note that the Hankel matrix of a function f is a very redundant way to represent f. In particular, the value $f(x)$ appears $|x| + 1$ times in $\mathbf{H}_f$, and we have $f(x) = \mathbf{H}_f(x, \lambda) = \mathbf{H}_f(\lambda, x)$. An obvious observation is that a matrix $\mathbf{M} \in \mathbb{R}^{\Sigma^\star \times \Sigma^\star}$ satisfying $\mathbf{M}(u_1, v_1) = \mathbf{M}(u_2, v_2)$ for any $u_1v_1 = u_2v_2$ is the Hankel matrix of some function $f : \Sigma^\star \to \mathbb{R}$.

We will be considering (finite) sub-blocks of a bi-infinite Hankel matrix $\mathbf{H}_f$. An easy way to define such sub-blocks is using a *basis* $\mathcal{B} = (\mathcal{P}, \mathcal{S})$, where $\mathcal{P} \subseteq \Sigma^\star$ is a set of prefixes and $\mathcal{S} \subseteq \Sigma^\star$ a set of suffixes. We write $p = |\mathcal{P}|$ and $s = |\mathcal{S}|$. The sub-block of $\mathbf{H}_f$ defined by $\mathcal{B}$ is the $p \times s$ matrix $\mathbf{H}_\mathcal{B} \in \mathbb{R}^{\mathcal{P} \times \mathcal{S}}$ with $\mathbf{H}_\mathcal{B}(u, v) = \mathbf{H}_f(u, v) = f(uv)$ for any $u \in \mathcal{P}$ and $v \in \mathcal{S}$. We may just write $\mathbf{H}$ if the basis $\mathcal{B}$ is arbitrary or obvious from the context.

Not all bases will be equally useful for our purposes. In particular, we will be interested in so-called closed basis. Let $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ be a basis and write $\Sigma' = \Sigma \cup \{\lambda\}$. The *p-closure*[1] of $\mathcal{B}$ is the basis $\mathcal{B}' = (\mathcal{P}', \mathcal{S})$, where $\mathcal{P}' = \mathcal{P}\Sigma'$. Equivalently, a basis $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ is said to be *p-closed* if $\mathcal{P} = \mathcal{P}'\Sigma'$ for some $\mathcal{P}'$ called the *root* of $\mathcal{P}$. It turns out that a Hankel matrix over a p-closed basis can be partitioned into $|\Sigma| + 1$ blocks of the same size. This partition will be central to our results. Let $\mathbf{H}_f$ be a Hankel matrix and $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ a basis. For any $\sigma \in \Sigma'$ we write $\mathbf{H}_\sigma$ to denote the sub-block of $\mathbf{H}_f$ over the basis $(\mathcal{P}\sigma, \mathcal{S})$. That is, the sub-block $\mathbf{H}_\sigma \in \mathbb{R}^{\mathcal{P}\sigma \times \mathcal{S}}$ of $\mathbf{H}_f$ is the $p \times s$ matrix defined by $\mathbf{H}_\sigma(u, v) = \mathbf{H}_f(u\sigma, v)$. Thus, if $\mathcal{B}'$ is the p-closure of $\mathcal{B}$, then for a particular ordering of the strings in $\mathcal{P}'$, we have

$$\mathbf{H}_{\mathcal{B}'}^\top = \left[\, \mathbf{H}_\lambda^\top \, \middle| \, \mathbf{H}_{\sigma_1}^\top \, \middle| \, \cdots \, \middle| \, \mathbf{H}_{\sigma_{|\Sigma|}}^\top \, \right] \; .$$

The *rank* of a function $f : \Sigma^\star \to \mathbb{R}$ is defined as the rank of its Hankel matrix: $\operatorname{rank}(f) = \operatorname{rank}(\mathbf{H}_f)$. The rank of a sub-block of $\mathbf{H}_f$ cannot exceed $\operatorname{rank}(f)$, and we will be specially interested on sub-blocks with full rank. We say that a basis $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ is *complete* for f if the sub-block $\mathbf{H}_\mathcal{B}$ has full rank: $\operatorname{rank}(\mathbf{H}_\mathcal{B}) = \operatorname{rank}(\mathbf{H}_f)$. In this case we say that $\mathbf{H}_\mathcal{B}$ is a *complete sub-block* of $\mathbf{H}_f$. It turns out that the rank of f is related to the number of states needed to compute f with a weighted automaton, and that the p-closure of a complete sub-block of $\mathbf{H}_f$ contains enough information to compute this automaton. These two results will provide the basis for the learning algorithm presented in Section 4.

2.2 Weighted Finite Automata

A widely used class of functions mapping strings to real numbers is that of functions defined by *weighted finite automata* (WFA) or in short *weighted automata* (Mohri,

[1]The notation *p*-closure stands for prefix-closure. A similar notion can be defined for suffixes as well.

2009). These functions are also known as *rational power series* (Salomaa and Soittola, 1978; Berstel and Reutenauer, 1988). A WFA over Σ with n states can be defined as a tuple $A = \langle \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\} \rangle$, where $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_\infty \in \mathbb{R}^n$ are the *initial* and *final weight* vectors, and $\mathbf{A}_\sigma \in \mathbb{R}^{n \times n}$ the transition matrix associated to each alphabet symbol $\sigma \in \Sigma$. The function f_A realized by a WFA A is defined by

$$f_A(x) = \boldsymbol{\alpha}_1^\top \mathbf{A}_{x_1} \cdots \mathbf{A}_{x_t} \boldsymbol{\alpha}_\infty = \boldsymbol{\alpha}_1^\top \mathbf{A}_x \boldsymbol{\alpha}_\infty \ ,$$

for any string $x = x_1 \cdots x_t \in \Sigma^\star$ with $t = |x|$ and $x_i \in \Sigma$ for all $1 \leq i \leq t$. We will write $|A|$ to denote the number of states of a WFA. The following characterization of the set of functions $f : \Sigma^\star \to \mathbb{R}$ realizable by WFA in terms of the rank of their Hankel matrix $\operatorname{rank}(\mathbf{H}_f)$ was given in (Carlyle and Paz, 1971; Fliess, 1974). We also note that the construction of an equivalent WFA with the minimal number of states from a given WFA was first given in (Schützenberger, 1961).

Theorem 2.1 (Carlyle and Paz (1971); Fliess (1974)). *A function $f : \Sigma^\star \to \mathbb{R}$ can be defined by a WFA iff $\operatorname{rank}(\mathbf{H}_f)$ is finite, and in that case $\operatorname{rank}(\mathbf{H}_f)$ is the minimal number of states of any WFA A such that $f = f_A$.*

In view of this result, we will say that A is *minimal* for f if $f_A = f$ and $|A| = \operatorname{rank}(f)$.

Another useful fact about WFA is their invariance under change of basis. It follows from the definition of f_A that if $\mathbf{M} \in \mathbb{R}^{n \times n}$ is an invertible matrix, then the WFA $B = \left\langle \mathbf{M}^\top \boldsymbol{\alpha}_1, \mathbf{M}^{-1} \boldsymbol{\alpha}_\infty, \{\mathbf{M}^{-1} \mathbf{A}_\sigma \mathbf{M}\} \right\rangle$ satisfies $f_B = f_A$. Sometimes B will be denoted by $\mathbf{M}^{-1} A \mathbf{M}$. This fact will prove very useful when we consider the problem of learning a WFA realizing a certain function.

Weighted automata are related to other finite state computational models. In particular, WFA can also be defined more generally over an arbitrary semi-ring instead of the field of real numbers, in which case there are sometimes called *multiplicity automata* (MA) (e.g. Beimel et al (2000)). It is well known that using weights over an arbitrary semi-ring more computational power is obtained. However, in this paper we will only consider WFA with real weights. It is easy to see that several other models of automata (DFA, PDFA, PNFA) can be cast as special cases of WFA.

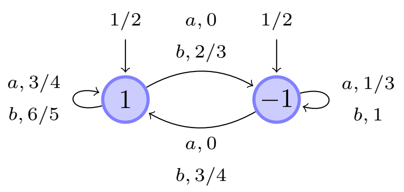
2.2.1 Example

Figure 1 shows an example of a weighted automaton $A = \langle \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\} \rangle$ with two states defined over the alphabet $\Sigma = \{a, b\}$, with both its algebraic representation (Figure 1(b)) in terms of vectors and matrices and the equivalent graph representation (Figure 1(a)) useful for a variety of WFA algorithms (Mohri, 2009). Letting $\mathcal{W} = \{\epsilon, a, b\}$, then $\mathcal{B} = (\mathcal{W}\Sigma', \mathcal{W})$ is a p-closed basis. The following is the Hankel matrix of A on this basis shown with two-digit precision entries:

$$\mathbf{H}_{\mathcal{B}}^\top = \begin{array}{c} \\ \epsilon \\ a \\ b \end{array} \begin{array}{c} \begin{array}{ccccccc} \epsilon & a & b & aa & ab & ba & bb \end{array} \\ \left[\begin{array}{ccccccc} 0.00 & 0.20 & 0.14 & 0.22 & 0.15 & 0.45 & 0.31 \\ 0.20 & 0.22 & 0.45 & 0.19 & 0.29 & 0.45 & 0.85 \\ 0.14 & 0.15 & 0.31 & 0.13 & 0.20 & 0.32 & 0.58 \end{array}\right] \end{array}$$

3 Observables in Stochastic Weighted Automata

Previous section introduces the class of WFA in a general setting. As we will see in next section, in order to learn an automata realizing (an approximation of) a function



(a)

$$\boldsymbol{\alpha}_1^\top = \begin{bmatrix} 1/2 & 1/2 \end{bmatrix} \quad \boldsymbol{\alpha}_\infty^\top = \begin{bmatrix} 1 & -1 \end{bmatrix}$$
$$\mathbf{A}_a = \begin{bmatrix} 3/4 & 0 \\ 0 & 1/3 \end{bmatrix} \quad \mathbf{A}_b = \begin{bmatrix} 6/5 & 2/3 \\ 3/4 & 1 \end{bmatrix}$$

(b)

Figure 1: Example of a weighted automaton over $\Sigma = \{a, b\}$ with 2 states: (a) graph representation; (b) algebraic representation.

$f : \Sigma^\star \to \mathbb{R}$ using a spectral algorithm, we will need to compute (an estimate) of a sub-block of the Hankel matrix $\mathbf{H}_f$. In general such sub-blocks may be hard to obtain. However, in the case when f computes a probability distribution over $\Sigma^\star$ and we have access to a sample of i.i.d. examples from this distribution, estimates of sub-blocks of $\mathbf{H}_f$ can be obtained efficiently. In this section we discuss some properties of WFA which realize probability distributions. In particular, we are interested in showing how different kinds of statistics that can be computed from a sample of strings induce functions on $\Sigma^\star$ realized by similar WFA.

We say that a WFA A is *stochastic* if the function $f = f_A$ is a probability distribution over $\Sigma^\star$. That is, if $f(x) \geq 0$ for all $x \in \Sigma^\star$ and $\sum_{x \in \Sigma^\star} f(x) = 1$. To make it clear that f represents a probability distribution we may sometimes write it as $f(x) = \mathbb{P}[x]$.

An interesting fact about distributions over $\Sigma^\star$ is that given an i.i.d. sample generated from that distribution one can compute an estimation $\hat{\mathbf{H}}_f$ of its Hankel matrix, or of any finite sub-block $\hat{\mathbf{H}}_\mathcal{B}$. When the sample is large enough, these estimates will converge to the true Hankel matrices. In particular, suppose $S = (x^1, \ldots, x^m)$ is a sample containing m i.i.d. strings from some distribution $\mathbb{P}$ over $\Sigma^\star$ and let us write $\hat{\mathbb{P}}_S(x)$ for the empirical frequency of x in S. Then, for a fixed basis $\mathcal{B}$, if we compute the *empirical Hankel matrix* given by $\hat{\mathbf{H}}_\mathcal{B}(u, v) = \hat{\mathbb{P}}_S[uv]$, one can show using McDiarmid's inequality that with high probability the following holds (Hsu et al, 2009):

$$\|\mathbf{H}_\mathcal{B} - \hat{\mathbf{H}}_\mathcal{B}\|_F \leq O\left(\frac{1}{\sqrt{m}}\right) .$$

This is one of the pillars on which the finite sample analysis of the spectral method lies. We will discuss this further in Section 4.2.1.

Note that when f realizes a distribution over $\Sigma^\star$, one can think of computing other probabilistic quantities besides probabilities of strings $\mathbb{P}[x]$. For example, one can define the function f_p that computes probabilities of prefixes; that is, $f_\mathrm{p}(x) = \mathbb{P}[x\Sigma^\star]$. Another probabilistic function that can be computed from a distribution over $\Sigma^\star$ is the expected number of times a particular string appears as a substring of random strings; we use f_s to denote this function. More formally, given two strings $w, x \in \Sigma^\star$ let $|w|_x$ denote the number of times that x appears in w as a substring. Then we can write $f_\mathrm{s}(x) = \mathbb{E}[|w|_x]$, where the expectation is with respect to w sampled from f: $\mathbb{E}[|w|_x] = \sum_{w \in \Sigma^\star} |w|_x \mathbb{P}[w]$.

In general the class of stochastic WFA may include some pathological examples with states that are not connected to any terminating state. In order to avoid such cases we introduce the following technical condition. Given a stochastic WFA $A = \langle \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\} \rangle$ let $\mathbf{A} = \sum_{\sigma \in \Sigma} \mathbf{A}_\sigma$. We say that A is *irredundant* if $\|\mathbf{A}\| < 1$ for some submultiplicative matrix norm $\|\cdot\|$. Note that a necessary condition for this to happen is that the spectral radius of $\mathbf{A}$ is less than one: $\rho(\mathbf{A}) < 1$. In particular, irredundancy implies that the sum $\sum_{k\geq 0}\mathbf{A}^k$ converges to $(\mathbf{I}-\mathbf{A})^{-1}$. An interesting property of irredundant stochastic WFA is that both f_{p} and f_{s} can also be computed by WFA as shown by the following result.

Lemma 3.1. *Let $\langle\boldsymbol{\alpha}_1,\boldsymbol{\alpha}_\infty,\{\mathbf{A}_\sigma\}\rangle$ be an irredundant stochastic WFA and write: $\mathbf{A} = \sum_{\sigma\in\Sigma}\mathbf{A}_\sigma$, $\tilde{\boldsymbol{\alpha}}_1^\top = \boldsymbol{\alpha}_1^\top(\mathbf{I}-\mathbf{A})^{-1}$, and $\tilde{\boldsymbol{\alpha}}_\infty = (\mathbf{I}-\mathbf{A})^{-1}\boldsymbol{\alpha}_\infty$. Suppose $f:\Sigma^\star\to\mathbb{R}$ is a probability distribution such that $f(x) = \mathbb{P}[x]$ and define functions $f_{\mathrm{p}}(x) = \mathbb{P}[x\Sigma^\star]$ and $f_{\mathrm{s}}(x) = \mathbb{E}[|w|_x]$. Then, the following are equivalent:*

1. *$A = \langle\boldsymbol{\alpha}_1,\boldsymbol{\alpha}_\infty,\{\mathbf{A}_\sigma\}\rangle$ realizes f,*
2. *$A_{\mathrm{p}} = \langle\boldsymbol{\alpha}_1,\tilde{\boldsymbol{\alpha}}_\infty,\{\mathbf{A}_\sigma\}\rangle$ realizes f_{p},*
3. *$A_{\mathrm{s}} = \langle\tilde{\boldsymbol{\alpha}}_1,\tilde{\boldsymbol{\alpha}}_\infty,\{\mathbf{A}_\sigma\}\rangle$ realizes f_{s}.*

Proof. In the first place we note that because A is irredundant we have

$$\tilde{\boldsymbol{\alpha}}_1^\top = \boldsymbol{\alpha}_1^\top\sum_{k\geq 0}\mathbf{A}^k = \sum_{x\in\Sigma^\star}\boldsymbol{\alpha}_1^\top\mathbf{A}_x \enspace ,$$

where the second equality follows from a term reordering. Similarly, we have $\tilde{\boldsymbol{\alpha}}_\infty = \sum_{x\in\Sigma^\star}\mathbf{A}_x\boldsymbol{\alpha}_\infty$. The rest of the proof follows from checking several implications.

($1\Rightarrow 2$) Using $f(x) = \boldsymbol{\alpha}_1^\top\mathbf{A}_x\boldsymbol{\alpha}_\infty$ and the definition of $\tilde{\boldsymbol{\alpha}}_\infty$ we have:

$$\mathbb{P}[x\Sigma^\star] = \sum_{y\in\Sigma^\star}\mathbb{P}[xy] = \sum_{y\in\Sigma^\star}\boldsymbol{\alpha}_1^\top\mathbf{A}_x\mathbf{A}_y\boldsymbol{\alpha}_\infty = \boldsymbol{\alpha}_1^\top\mathbf{A}_x\tilde{\boldsymbol{\alpha}}_\infty \enspace .$$

($2\Rightarrow 1$) It follows from $\mathbb{P}[x\Sigma^+] = \sum_{\sigma\in\Sigma}\mathbb{P}[x\sigma\Sigma^\star]$ that

$$\mathbb{P}[x] = \mathbb{P}[x\Sigma^\star] - \mathbb{P}[x\Sigma^+] = \boldsymbol{\alpha}_1^\top\mathbf{A}_x\tilde{\boldsymbol{\alpha}}_\infty - \boldsymbol{\alpha}_1^\top\mathbf{A}_x\mathbf{A}\tilde{\boldsymbol{\alpha}}_\infty = \boldsymbol{\alpha}_1^\top\mathbf{A}_x(\mathbf{I}-\mathbf{A})\tilde{\boldsymbol{\alpha}}_\infty \enspace .$$

($1\Rightarrow 3$) Since we can write $\sum_{w\in\Sigma^\star}\mathbb{P}[w]|w|_x = \mathbb{P}[\Sigma^\star x\Sigma^\star]$, it follows that

$$\begin{aligned}\mathbb{E}[|w|_x] &= \sum_{w\in\Sigma^\star}\mathbb{P}[w]|w|_x \\ &= \sum_{u,v\in\Sigma^\star}\mathbb{P}[uxv] = \sum_{u,v\in\Sigma^\star}\boldsymbol{\alpha}_1^\top\mathbf{A}_u\mathbf{A}_x\mathbf{A}_v\boldsymbol{\alpha}_\infty = \tilde{\boldsymbol{\alpha}}_1^\top\mathbf{A}_x\tilde{\boldsymbol{\alpha}}_\infty \enspace .\end{aligned}$$

($3\Rightarrow 1$) Using similar arguments as before we observe that

$$\begin{aligned}\mathbb{P}[x] &= \mathbb{P}[\Sigma^\star x\Sigma^\star] + \mathbb{P}[\Sigma^+ x\Sigma^+] - \mathbb{P}[\Sigma^+ x\Sigma^\star] - \mathbb{P}[\Sigma^\star x\Sigma^+] \\ &= \tilde{\boldsymbol{\alpha}}_1^\top\mathbf{A}_x\tilde{\boldsymbol{\alpha}}_\infty + \tilde{\boldsymbol{\alpha}}_1^\top\mathbf{A}\mathbf{A}_x\mathbf{A}\tilde{\boldsymbol{\alpha}}_\infty - \tilde{\boldsymbol{\alpha}}_1^\top\mathbf{A}\mathbf{A}_x\tilde{\boldsymbol{\alpha}}_\infty - \tilde{\boldsymbol{\alpha}}_1^\top\mathbf{A}_x\mathbf{A}\tilde{\boldsymbol{\alpha}}_\infty \\ &= \tilde{\boldsymbol{\alpha}}_1^\top(\mathbf{I}-\mathbf{A})\mathbf{A}_x(\mathbf{I}-\mathbf{A})\tilde{\boldsymbol{\alpha}}_\infty \enspace .\end{aligned}$$

A direct consequence of this constructive result is that given a WFA realizing a probability distribution $\mathbb{P}[x]$ we can easily compute WFA realizing the functions f_{p} and f_{s}; and the converse holds as well. Lemma 3.1 also implies the following result, which characterizes the rank of f_{p} and f_{s}.

Corollary 3.2. *Suppose $f:\Sigma^\star\to\mathbb{R}$ is stochastic and admits a minimal irredundant WFA. Then $\operatorname{rank}(f) = \operatorname{rank}(f_{\mathrm{p}}) = \operatorname{rank}(f_{\mathrm{s}})$.*

Proof. Since all the constructions of Lemma 3.1 preserve the number of states, the result follows from considering minimal WFA for f, f_{p}, and f_{s}.

From the point of view of learning, Lemma 3.1 provides us with tools for proving two-sided reductions between the problems of learning f, f_{p}, and f_{s}. Since for all these problems the corresponding empirical Hankel matrices can be easily computed, this implies that for each particular task we can use the statistics which better suit its needs. For example, if we are interested in learning a model that predicts the next symbol in a string we might learn the function f_{p}. On the other hand, if we want to predict missing symbols in the middle of string we might learn the distribution f itself. Using Lemma 3.1 we see that both could be learned from substring statistics.

4 Duality, Spectral Learning, and Forward-Backward Decompositions

In this section we give a derivation of the spectral learning algorithm. Our approach follows from a duality result between minimal WFA and factorizations of Hankel matrices. We begin by presenting this duality result and some of its consequences. Afterwards we proceed to describe the spectral method, which is just an efficient implementation of the arguments used in the proof of the duality result. Finally we give an interpretation of this method from the point of view of forward and backward recursions in finite automata. This provides extra intuitions about the method and stresses the role played by factorizations in its derivation.

4.1 Duality and Minimal Weighted Automata

Let f be a real function on strings and $\mathbf{H}_f$ its Hankel matrix. In this section we consider factorizations of $\mathbf{H}_f$ and minimal WFA for f. We will show that there exists an interesting relation between these two concepts. This relation will motivate the algorithm presented on next section that factorizes a (sub-block of a) Hankel matrix in order to learn a WFA for some unknown function.

Our initial observation is that a WFA $A = \langle \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\}\rangle$ for f with n states induces a factorization of $\mathbf{H}_f$. Let $\mathbf{P} \in \mathbb{R}^{\Sigma^\star \times n}$ be a matrix whose uth row equals $\boldsymbol{\alpha}_1^\top \mathbf{A}_u$ for any $u \in \Sigma^\star$. Furthermore, let $\mathbf{S} \in \mathbb{R}^{n \times \Sigma^\star}$ be a matrix whose columns are of the form $\mathbf{A}_v \boldsymbol{\alpha}_\infty$ for all $v \in \Sigma^\star$. It is trivial to check that one has $\mathbf{H}_f = \mathbf{P}\mathbf{S}$. The same happens for sub-blocks: if $\mathbf{H}_\mathcal{B}$ is a sub-block of $\mathbf{H}_f$ defined over an arbitrary basis $\mathcal{B} = (\mathcal{P}, \mathcal{S})$, then the corresponding restrictions $\mathbf{P}_\mathcal{B} \in \mathbb{R}^{\mathcal{P} \times n}$ and $\mathbf{S}_\mathcal{B} \in \mathbb{R}^{n \times \mathcal{S}}$ of $\mathbf{P}$ and $\mathbf{S}$ induce the factorization $\mathbf{H}_\mathcal{B} = \mathbf{P}_\mathcal{B}\mathbf{S}_\mathcal{B}$. Furthermore, if $\mathbf{H}_\sigma$ is a sub-block of the matrix $\mathbf{H}_{\mathcal{B}'}$ corresponding to the p-closure of $\mathbf{H}_\mathcal{B}$, then we also have the factorization $\mathbf{H}_\sigma = \mathbf{P}_\mathcal{B}\mathbf{A}_\sigma\mathbf{S}_\mathcal{B}$.

An interesting consequence of this construction is that if A is minimal for f – i.e. $n = \operatorname{rank}(f)$ – then the factorization $\mathbf{H}_f = \mathbf{P}\mathbf{S}$ is in fact a *rank factorization*. Since in general $\operatorname{rank}(\mathbf{H}_\mathcal{B}) \leq n$, in this case the factorization $\mathbf{H}_\mathcal{B} = \mathbf{P}_\mathcal{B}\mathbf{S}_\mathcal{B}$ is a rank factorization if and only if $\mathbf{H}_\mathcal{B}$ is a complete sub-block. Thus, we see that a minimal WFA that realizes a function f induces a rank factorization on any complete sub-block of $\mathbf{H}_f$. The converse is even more interesting: give a rank factorization of a complete sub-block of $\mathbf{H}_f$, one can compute a minimal WFA for f.

Let $\mathbf{H}$ be a complete sub-block of $\mathbf{H}_f$ defined by the basis $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ and let $\mathbf{H}_\sigma$ denote the sub-block of the p-closure of $\mathbf{H}$ corresponding to the basis $(\mathcal{P}\sigma, \mathcal{S})$. Let

$\mathbf{h}_{\mathcal{P},\lambda} \in \mathbb{R}^{\mathcal{P}}$ denote the p-dimensional vector with coordinates $\mathbf{h}_{\mathcal{P},\lambda}(u) = f(u)$, and $\mathbf{h}_{\lambda,\mathcal{S}} \in \mathbb{R}^{\mathcal{S}}$ the s-dimensional vector with coordinates $\mathbf{h}_{\lambda,\mathcal{S}}(v) = f(v)$. Now we can state our result.

Lemma 4.1. *If $\mathbf{H} = \mathbf{PS}$ is a rank factorization, then the WFA $A = \langle \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\}\rangle$ with $\boldsymbol{\alpha}_1^\top = \mathbf{h}_{\lambda,\mathcal{S}}^\top \mathbf{S}^+$, $\boldsymbol{\alpha}_\infty = \mathbf{P}^+\mathbf{h}_{\mathcal{P},\lambda}$, and $\mathbf{A}_\sigma = \mathbf{P}^+\mathbf{H}_\sigma\mathbf{S}^+$, is minimal for f.*

Proof. Let $A' = \langle \boldsymbol{\alpha}'_1, \boldsymbol{\alpha}'_\infty, \{\mathbf{A}'_\sigma\}\rangle$ be a minimal WFA for f that induces a rank factorization $\mathbf{H} = \mathbf{P}'\mathbf{S}'$. It suffices to show that there exists an invertible $\mathbf{M}$ such that $\mathbf{M}^{-1}A'\mathbf{M} = A$. Define $\mathbf{M} = \mathbf{S}'\mathbf{S}^+$ and note that $\mathbf{P}^+\mathbf{P}'\mathbf{S}'\mathbf{S}^+ = \mathbf{P}^+\mathbf{H}\mathbf{S}^+ = \mathbf{I}$ implies that $\mathbf{M}$ is invertible with $\mathbf{M}^{-1} = \mathbf{P}^+\mathbf{P}'$. Now we check that the operators of A correspond to the operators of A' under this change of basis. First we see that $\mathbf{A}_\sigma = \mathbf{P}^+\mathbf{H}_\sigma\mathbf{S}^+ = \mathbf{P}^+\mathbf{P}'\mathbf{A}'_\sigma\mathbf{S}'\mathbf{S}^+ = \mathbf{M}^{-1}\mathbf{A}'_\sigma\mathbf{M}$. Now observe that by the construction of $\mathbf{S}'$ and $\mathbf{P}'$ we have ${\boldsymbol{\alpha}'_1}^\top\mathbf{S}' = \mathbf{h}_{\lambda,\mathcal{S}}$, and $\mathbf{P}'\boldsymbol{\alpha}'_\infty = \mathbf{h}_{\mathcal{P},\lambda}$. Thus, it follows that $\boldsymbol{\alpha}_1^\top = {\boldsymbol{\alpha}'_1}^\top\mathbf{M}$ and $\boldsymbol{\alpha}_\infty = \mathbf{M}^{-1}\boldsymbol{\alpha}'_\infty$.

This result shows that there exists a duality between rank factorizations of complete sub-blocks of $\mathbf{H}_f$ and minimal WFA for f. A consequence of this duality is that all minimal WFA for a function f are related via some change of basis. In other words, modulo change of basis, there exists a unique minimal WFA for any function f of finite rank.

Corollary 4.2. *Let $A = \langle \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\}\rangle$ and $A' = \langle \boldsymbol{\alpha}'_1, \boldsymbol{\alpha}'_\infty, \{\mathbf{A}'_\sigma\}\rangle$ be minimal WFA for some f of rank n. Then there exists an invertible matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$ such that $A = \mathbf{M}^{-1}A'\mathbf{M}$.*

Proof. Suppose that $\mathbf{H}_f = \mathbf{PS} = \mathbf{P}'\mathbf{S}'$ are the rank factorizations induced by A and A' respectively. Then, by the same arguments used in Lemma 4.1, the matrix $\mathbf{M} = \mathbf{S}'\mathbf{S}^+$ is invertible and satisfies the equation $A = \mathbf{M}^{-1}A'\mathbf{M}$.

4.2 A Spectral Learning Algorithm

The spectral method is basically an efficient algorithm that implements the ideas in the proof of Lemma 4.1 to find a rank factorization of a complete sub-block $\mathbf{H}$ of $\mathbf{H}_f$ and obtain from it a minimal WFA for f. The term *spectral* comes from the fact that it uses SVD, a type of spectral decomposition. We describe the algorithm in detail in this section and give a complete set of experiments that explores the practical behavior of this method in Section 5.

Suppose $f : \Sigma^\star \to \mathbb{R}$ is an unknown function of finite rank n and we want to compute a minimal WFA for it. Let us assume that we know that $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ is a complete basis for f. Our algorithm receives as input: the basis $\mathcal{B}$ and the values of f on a set of strings $\mathcal{W}$. In particular, we assume that $\mathcal{P}\Sigma'\mathcal{S} \cup \mathcal{P} \cup \mathcal{S} \subseteq \mathcal{W}$. It is clear that using these values of f the algorithm can compute sub-blocks $\mathbf{H}_\sigma$ for $\sigma \in \Sigma'$ of $\mathbf{H}_f$. Furthermore, it can compute the vectors $\mathbf{h}_{\lambda,\mathcal{S}}$ and $\mathbf{h}_{\mathcal{P},\lambda}$. Thus, the algorithm only needs a rank factorization of $\mathbf{H}_\lambda$ to be able to apply the formulas given in Lemma 4.1.

Recall that the *compact SVD* of a $p \times s$ matrix $\mathbf{H}_\lambda$ of rank n is given by the expression $\mathbf{H}_\lambda = \mathbf{U}\boldsymbol{\Lambda}\mathbf{V}^\top$, where $\mathbf{U} \in \mathbb{R}^{p\times n}$ and $\mathbf{V} \in \mathbb{R}^{s\times n}$ are orthogonal matrices, and $\boldsymbol{\Lambda} \in \mathbb{R}^{n\times n}$ is a diagonal matrix containing the singular values of $\mathbf{H}_\lambda$. The most interesting property of compact SVD for our purposes is that $\mathbf{H}_\lambda = (\mathbf{U}\boldsymbol{\Lambda})\mathbf{V}^\top$ is a rank factorization. We will use this factorization in the algorithm, but write it in a different way. Note that since $\mathbf{V}$ is orthogonal we have $\mathbf{V}^\top\mathbf{V} = \mathbf{I}$, and in particular $\mathbf{V}^+ = \mathbf{V}^\top$. Thus, the factorization above is equivalent to $\mathbf{H}_\lambda = (\mathbf{H}_\lambda\mathbf{V})\mathbf{V}^\top$.

With this factorization, equations from Lemma 4.1 are written as follows:

$$\begin{aligned}\boldsymbol{\alpha}_1^\top &= \mathbf{h}_{\lambda,\mathcal{S}}^\top \mathbf{V} \ , \\ \boldsymbol{\alpha}_\infty &= (\mathbf{H}_\lambda \mathbf{V})^+ \mathbf{h}_{\mathcal{P},\lambda} \ , \\ \mathbf{A}_\sigma &= (\mathbf{H}_\lambda \mathbf{V})^+ \mathbf{H}_\sigma \mathbf{V} \ .\end{aligned}$$

These equations define what we call from now on the *spectral learning* algorithm. The running time of the algorithm can be bound as follows. Note that the cost of computing a compact SVD and the pseudo-inverse is $O(|\mathcal{P}||\mathcal{S}|n)$, and the cost of computing the operators is $O(|\Sigma||\mathcal{P}|n^2)$. To this we need to add the time required in order to compute the Hankel matrices given to the algorithm. In the particular case of stochastic WFA described in Section 3, approximate Hankel matrices can be computed from a sample S containing m examples in time $O(m)$ – note that the running time of all linear algebra operations is independent of the sample size. Thus, we get a total running time of $O(m + n|\mathcal{P}||\mathcal{S}| + n^2|\mathcal{P}||\Sigma|)$ for the spectral algorithm applied to learn any stochastic function of the type described in Section 3.

4.2.1 Sample Complexity of Spectral Learning

The spectral algorithm we just described can be used even when $\mathbf{H}$ and $\mathbf{H}_\sigma$ are not known exactly, but approximations $\hat{\mathbf{H}}$ and $\hat{\mathbf{H}}_\sigma$ are available. In this context, an approximation means that we have an estimate for each entry in these matrices; that is, we know an estimate of f for every string in $\mathcal{W}$. A different concept of approximation could be that one knows f exactly in some, but not all strings in $\mathcal{W}$. In this context, one can still apply the spectral method after a preliminary matrix completion step; see (Balle and Mohri, 2012) for details. When the goal is to learn a probability distribution over strings – or prefixes, or substrings – we are always in the first of these two settings. In these cases we can apply the spectral algorithm directly using empirical estimations $\hat{\mathbf{H}}$ and $\hat{\mathbf{H}}_\sigma$. A natural question is then how close to f is the approximate function $\hat{f}$ computed by the learned automaton $\hat{A}$. Experiments described in the following sections explore this question from an empirical perspective and compare the performance of spectral learning with other approaches. Here we give a very brief outline of what is known about the *sample complexity* of spectral learning. Since an in-depth discussion of these results and the techniques used in their proofs is outside the scope of this paper, for further details we refer the reader to papers where these bounds were originally presented (Hsu et al, 2009; Bailly et al, 2009; Siddiqi et al, 2010; Bailly, 2011; Balle, 2013).

All known results about learning stochastic WFA with spectral methods fall into the well-known *PAC-learning* framework (Valiant, 1984; Kearns et al, 1994). In particular, assuming that a large enough sample of i.i.d. strings drawn from some distribution f over $\Sigma^\star$ realized by a WFA is given to the spectral learning algorithm, we know that with high probability the output WFA computes a function $\hat{f}$ that is close to f. Sample bounds in this type of results usually depend polynomially on the usual PAC parameters – accuracy ε and confidence δ – as well as other parameters depending on the target f: the size of the alphabet Σ, the number of states n of a minimal WFA realizing f, the size of the basis $\mathcal{B}$, and the smallest singular values of $\mathbf{H}$ and other related matrices.

These results come in different flavors, depending on what assumptions are made on the automaton computing f and what criteria is used to measure how close $\hat{f}$ is to f. When f can be realized by a Hidden Markov Model (HMM), Hsu et al (2009) proved a PAC-learning result under the L_1 distance restricted to strings in Σ^t for some $t \geq 0$

– their bound depends polynomially in t. A similar result was obtained in (Siddiqi et al, 2010) for Reduced Rank HMM. For targets f computed by a general stochastic WFA, Bailly et al (2009) gave a similar results under the milder L_∞ distance. When f can be computed by a Quadratic WFA one can obtain L_1 bounds over all $\Sigma^\star$; see (Bailly, 2011). The case where the function can be computed by a Probabilistic WFA was analyzed in (Balle, 2013), where L_1 bounds over strings in $\Sigma^{\leq t}$ are given. It is important to note that, with the exception of (Bailly, 2011), none of these methods is guaranteed to return a stochastic WFA. That is, though the hypothesis $\hat{f}$ is close to a probability distribution in L_1 distance, it does not necessarily assign a non-negative number to each strings, much less adds up to one when summed over all strings – though both properties are satisfied *in the limit*. In practice this is a problem when trying to evaluate these methods using perplexity-like accuracy measures. We do not face this difficulty in our experiments because we use WER-like accuracy measures. See the discussion in Section 8 for pointers to some attempts to solve this problem.

Despite their formal differences, all these PAC-learning results rely on similar proof techniques. Roughly speaking, the following three principles lay at the bottom of these results:

1. Convergence of empirical estimates $\hat{\mathbf{H}}$ and $\hat{\mathbf{H}}_\sigma$ to their true values at a rate of $O(m^{-1/2})$ in terms of Frobenius norms; here m is the sample size.
2. Stability of linear algebra operations – SVD, pseudoinverse and matrix multiplication – under small perturbations. This implies that when the errors in empirical Hankel matrices are small, we get operators $\hat{\boldsymbol{\alpha}}_1$, $\hat{\boldsymbol{\alpha}}_\infty$, and $\hat{\mathbf{A}}_\sigma$ which are close to their true values, modulo a change of basis.
3. Mild aggregation of errors when computing $\sum |f(x) - \hat{f}(x)|$ over large sets of strings.

We note here that the first of these points, which we already mentioned in Section 3, is enough to show the statistical consistency of spectral learning. The other two points are rather technical and lie at the core of finite-sample analyses of spectral learning of stochastic WFA.

4.2.2 Choosing the parameters

When run with approximate data $\hat{\mathbf{H}}_\lambda$, $\hat{\mathbf{H}}_\sigma$ for $\sigma \in \Sigma$, $\hat{\mathbf{h}}_{\lambda,\mathcal{S}}$, and $\hat{\mathbf{h}}_{\mathcal{P},\lambda}$, the algorithm also receives as input the number of states n of the target WFA. That is because the rank of $\hat{\mathbf{H}}_\lambda$ may be different from the rank of $\mathbf{H}_\lambda$ due to the noise, and in this case the algorithm may need to ignore some of the smallest singular values of $\hat{\mathbf{H}}_\lambda$, which just correspond to zeros in the original matrix that have been corrupted by noise. This is done by just computing a *truncated SVD* of $\hat{\mathbf{H}}_\lambda$ up to dimension n – we note that the cost of this computation is the same as the computation of a compact SVD on a matrix of rank n. It was shown in (Bailly, 2011) that when empirical Hankel matrices are sufficiently accurate, inspection of the singular values of $\hat{\mathbf{H}}$ can yield accurate estimates of the number of states n in the target. In practice one usually chooses the number of states via some sort of cross-validation procedure. We will get back to this issue in Section 5.

The other important parameter to choose when using the spectral algorithm is the basis. It is easy to show that for functions of rank n there always exist complete basis with $|\mathcal{P}| = |\mathcal{S}| = n$. In general there exist infinitely many complete basis and it is safe to assume in theoretical results that at least one is given to the algorithm. However, choosing a basis in practice turns out to be a complex task. A common choice are basis of the form $\mathcal{P} = \mathcal{S} = \Sigma^{\leq k}$ for some $k > 0$ (Hsu et al, 2009; Siddiqi et al, 2010). Another approach, is to choose a basis that contains the most frequent elements observed in the sample, which depending on the particular target model can be either strings, prefixes, suffixes, or substrings. This approach is motivated by the theoretical results from (Balle et al, 2012b). It is shown there that a random sampling strategy will succeed with high probability in finding a complete basis when given a large enough sample. This suggests that including frequent prefixes and suffixes might be a good heuristic. This approach is much faster than the greedy heuristic presented in (Wiewiora, 2005), which for each prefix added to the basis makes a computation taking exponential time in the number of states n. Other authors suggest using the largest Hankel matrix that can be estimated using the given sample; that is, build a basis that includes every prefix and suffix seen in the sample (Bailly et al, 2009). While the statistical properties of such estimation remain unclear, this approach becomes computationally unfeasible for large samples because in this case the size of the basis *does* grow with the number of examples m. All in all, designing an efficient algorithm for obtaining an optimal sample-dependent basis is an open problem. In our experiments we decided to adopt the simplest sample-dependent strategy: choosing the most frequent prefixes and suffixes in the sample. See Sections 5 and 7 for details.

4.3 The Forward-Backward Interpretation

We say that a WFA $A = \langle \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\} \rangle$ with n states is *probabilistic* if the following are satisfied:

1. All parameters are non-negative. That is, for all $\sigma \in \Sigma$ and all $i, j \in [n]$: $\mathbf{A}_\sigma(i,j) \geq 0$, $\boldsymbol{\alpha}_1(i) \geq 0$, and $\boldsymbol{\alpha}_\infty(i) \geq 0$.

2. Initial weights add up to one: $\sum_{i \in [n]} \boldsymbol{\alpha}_1(i) = 1$.

3. Transition and final weights from each state add up to one. That is, for all $i \in [n]$: $\boldsymbol{\alpha}_\infty(i) + \sum_{\sigma \in \Sigma} \sum_{j \in [n]} \mathbf{A}_\sigma(i,j) = 1$.

This model is also called in the literature *probabilistic finite automata* (PFA) or *probabilistic non-deterministic finite automata* (PNFA). It is obvious that probabilistic WFA are also stochastic, since $f_A(x)$ is the probability of generating x using the given automaton.

It turns out that when a probabilistic WFA $A = \langle \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\} \rangle$ is considered, the factorization induced on $\mathbf{H}$ has a nice probabilistic interpretation. Analyzing the spectral algorithm from this perspective yields additional insights which are useful to keep in mind.

Let $\mathbf{H}_f = \mathbf{PS}$ be the factorization induced by a probabilistic WFA with n states on the Hankel matrix of $f_A(x) = f(x) = \mathbb{P}[x]$. Then, for any prefix $u \in \Sigma^\star$, the uth row of $\mathbf{P}$ is given by the following n-dimensional vector:

$$\mathbf{P}_u(i) = \mathbb{P}[u\ ,\ s_{|u|+1} = i] \qquad i \in [n]\ .$$

That is, the probability that the probabilistic transition system given by A generates the prefix u and ends up in state i. The coordinates of these vectors are usually called

NNP	,	VBN	IN	NNP	NNP	,	VBZ	CC	VBZ	JJ	,	NN	CC	NN	NNS	.
Noun	.	Verb	Adp	Noun	Noun	.	Verb	Conj	Verb	Adj	.	Noun	Conj	Noun	Noun	.
Bell	,	based	in	Los	Angeles	,	makes	and	distributes	electronic	,	computer	and	building	products	.

Figure 2: An example sentence from the training set. The bottom row is are the words, which we do not model. The top row are the part-of-speech tags using the original tagset of 45 tags. The middle row are the simplified part-of-speech tags, using a tagset of 12 symbols.

forward probabilities. Similarly, the column of $\mathbf{S}$ given by suffix $v \in \Sigma^\star$ is the n-dimensional vector given by:

$$\mathbf{S}_v(i) = \mathbb{P}[v \mid s = i] \qquad i \in [n] \ .$$

This is the probability of generating a suffix s when A is started from state i. These are usually called *backward probabilities*.

The same interpretation applies to the factorization induced on a sub-block $\mathbf{H}_\mathcal{B} = \mathbf{P}_\mathcal{B}\mathbf{S}_\mathcal{B}$. Therefore, assuming there exists a minimal WFA for $f(x) = \mathbb{P}[x]$ which is probabilistic[2], Lemma 4.1 says that a WFA for f can be learned from information about the forward and backward probabilities over a small set of prefixes and suffixes. Teaming this basic observation with the spectral method and invariance under change of basis one can show an interesting fact: forward and backward (empirical) probabilities for a probabilistic WFA can be recovered (modulo a change of basis) by computing an SVD on (empirical) string probabilities. In other words, though state probabilities are *non-observable*, they can be recovered (modulo a linear transformation) from *observable* quantities.

5 Experiments on Learning PNFA

In this section we present some experiments that illustrate the behavior of the spectral learning algorithm at learning weighted automata under different configurations. We also present a comparison to alternative methods for learning WFA, namely to baseline unigram and bigram methods, and to an Expectation Maximization algorithm for learning PNFA (Dempster et al, 1977).

The data we use are sequences of part-of-speech tags of English sentences, hence the weighted automata we learn will model this type of sequential data. In Natural Language Processing, such sequential models are a central building block in methods for part-of-speech tagging. The data we used is from the Penn Treebank (Marcus et al, 1994), where the part-of-speech tagset consists of 45 symbols. To test the learning algorithms under different conditions, we also did experiments with a simplified tagset of 12 tags, using the mapping by Petrov et al (2012). We used the standard partitions for training (sections 2 to 21, with 39,832 sequences with an average length of 23.9) and validation (section 24, with 1,700 sequences with an average length of 23.6); we did not use the standard test set. Figure 2 shows an example sequence from the training set.

As a measure of error, we compute the *word error rate (WER)* on the validation set. WER computes the error at predicting the symbol that most likely follows a given

[2]This is not always the case, see (Denis and Esposito, 2008) for details.

prefix sequence, or predicting a special STOP symbol if the given prefix is most likely to be a complete sequence. If w is a validation sequence of length t, we evaluate $t+1$ events, one per each symbol w_i given the prefix $w_{1:i-1}$ and one for the stopping event; note that each event is independent of the others, and that we always use the correct prefix to condition on. WER is the percentage of errors averaged over all events in the validation set.

We would like to remind the reader that a WFA learned by the spectral method is only guaranteed to realize a probabilistic distribution on Σ^* when we use an *exact* complete sub-block of the Hankel of a stochastic function. In experiments, we only have access to a finite sample, and even though the SVD is robust to noise, we in fact observe that the WFA we obtain do not define distributions. Hence, standard evaluation metrics for probabilistic language models such as perplexity are not well defined here, and we prefer to use an error metric such as WER that does not require normalized predictions. We also avoid saying that these WFA compute probabilities over strings, and we will just say they compute scores.

5.1 Methods Compared

We now describe the weighted automata we compare, and give some details about how they were estimated and used to make predictions.

Unigram Model A WFA with a single state, that emits symbols according to their frequency in training data. When evaluating WER, this method will always predict the most likely symbol (in our data NN, which stands for singular noun).

Bigram Model A deterministic WFA with $|\Sigma|+1$ states, namely one special start state λ and one state per symbol σ, and the following operators:

- $\boldsymbol{\alpha}_1(\lambda) = 1$ and $\boldsymbol{\alpha}_1(\sigma) = 0$ for $\sigma \in \Sigma$
- $\mathbf{A}_\sigma(i,j) = 0$ if $\sigma \neq j$
- For each state i, $\mathbf{A}_\sigma(i,\sigma)$ for all σ and $\boldsymbol{\alpha}_\infty(i)$ is a distribution estimated from training counts, without smoothing.

EM Model A non-deterministic WFA with n states trained with Expectation Maximization (EM), where n is a parameter of the method. The learning algorithm initializes the WFA randomly, and then it proceeds iteratively by computing expected counts of state transitions on training sequences, and re-setting the parameters of the WFA by maximum likelihood given the expected counts. On validation data, we use a special operator $\tilde{\boldsymbol{\alpha}}_\infty = \vec{1}$ to compute prefix probabilities, and we use the $\boldsymbol{\alpha}_\infty$ resulting from EM to compute probabilities of complete sequences.

Spectral Model A non-deterministic WFA with n states trained with the spectral method, where the parameters of the method are a basis $(\mathcal{P},\mathcal{S})$ of prefixes and suffixes, and the number of states n. We experiment with two ways of setting the basis:

Σ Basis: We consider one prefix/suffix for each symbol in the alphabet, that is $\mathcal{P} = \mathcal{S} = \Sigma$. This is the setting analyzed by Hsu et al (2009) in their theoretical work. In this case, the statistics gathered at training to estimate the automaton will correspond to unigram, bigram and trigram statistics.

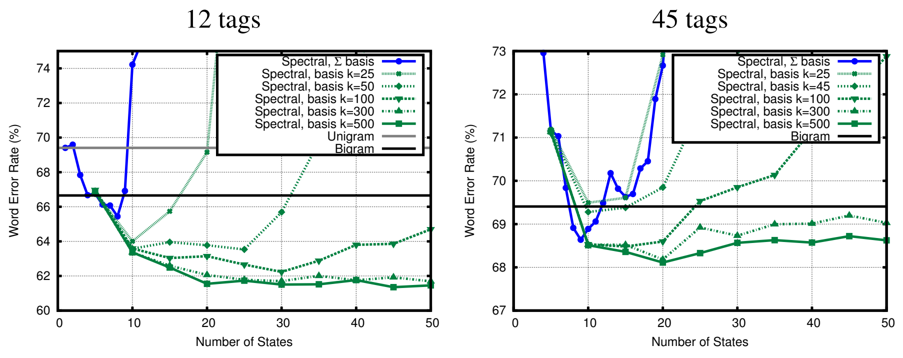


Figure 3: Performance of the spectral method in terms of WER relative to the number of states, compared to the baseline performance of an unigram and a bigram model. The left plot corresponds to the simplified tagset of 12 symbols, while the right plot corresponds to the tagset of 45 symbols. For the spectral method, we show a curve corresponding to the Σ basis, and curves for the extended that use the k most frequent training subsequences.

Top-k Basis: In this setting we set the prefixes and suffixes to be frequent subsequences of the training set. In particular, we consider all subsequences of symbols up to length 4, and sort them by frequency in the training set. We then set $\mathcal{P}$ and $\mathcal{S}$ to be the most frequent k subsequences, where k is a parameter of the model.

Since the training sequences are quite long, relative to the size of the sequences in the basis, we choose to estimate from the training sample a Hankel sub-block for the function $f_{\mathrm{s}}(x) = \mathbb{E}[|w|_x]$. Hence, the spectral method will return $A_{\mathrm{s}} = \langle \tilde{\boldsymbol{\alpha}}_1, \tilde{\boldsymbol{\alpha}}_\infty, \{\mathbf{A}_\sigma\}\rangle$ as defined in Lemma 3.1. We use Lemma 3.1 to transform A_{s} into A and then into A_{s}. To calcule WER on validation data, we use A_{s} to compute scores of prefix sequences, and A to compute scores of complete sequences.

As a final detail, when computing next-symbol predictions with WFA we kept normalizing the state vector. That is, if we are given a prefix sequence $w_{1,i}$ we compute $\boldsymbol{\alpha}^{i\,\top} A_\sigma \tilde{\boldsymbol{\alpha}}_\infty$ as the score for symbol σ and $\boldsymbol{\alpha}^{i\,\top} \boldsymbol{\alpha}_\infty$ as the score for stopping, where $\boldsymbol{\alpha}^i$ is a normalized state vector at position i. It is recursively computed as $\boldsymbol{\alpha}^1 = \boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}^{i+1} = \frac{\boldsymbol{\alpha}^{i\,\top} A_{w_i}}{\boldsymbol{\alpha}^{i\,\top} A_{w_i} \tilde{\boldsymbol{\alpha}}_\infty}$. This normalization should not change the predictions, but it helps avoiding numerical precision problems when validation sequences are relatively long.

5.2 Results

We trained all types of models for the two sets of tags, namely the simplified set of 12 tags and the original tagset of 45 tags. For the simplified set, the unigram model obtained a WER of 69.4% on validation data and the bigram improved to 66.6%. For the original tagset, the unigram and bigram WER were of 87.2% and 69.4%.

We then evaluated spectral models trained with the Σ basis. Figure 3 plots the WER of this method as a function of the number of states, for the simplified tagset

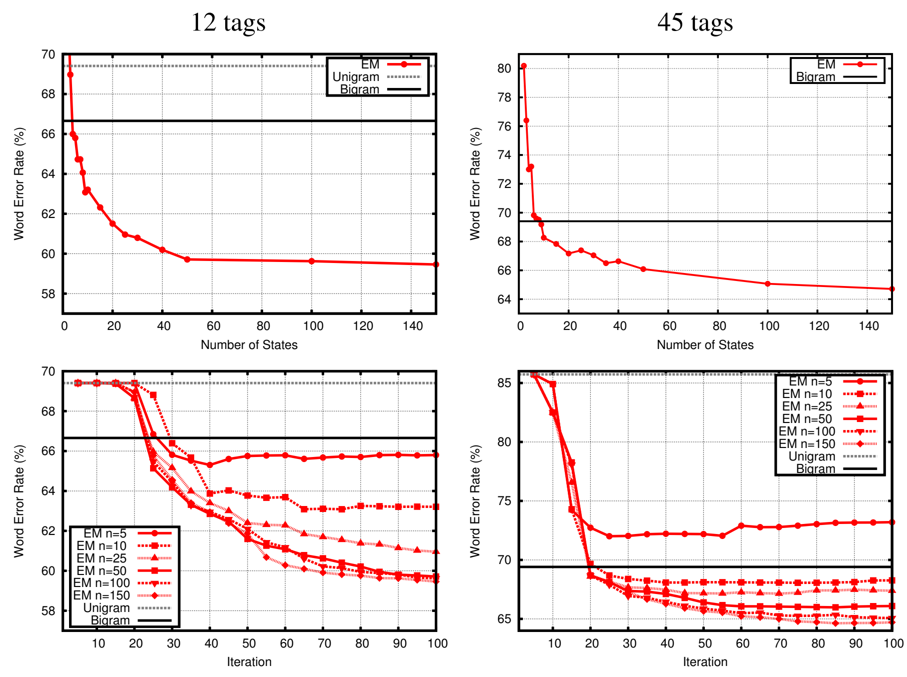


Figure 4: Top plots: performance of EM with respect to the number of states, where each model was run for 100 iterations. Bottom: convergence of EM in terms of WER at validation. Left plots correspond to the simplified tagset of 12 tags, while right plots correspond to the original tagset of 45 symbols.

(left) and the original one (right). We can see that the spectral method improves the bigram baseline when the number of states is 6–8 for the simplified tagset and 8–11 for the original tagset. While the improvements are not huge, one interpretation of this result is that the spectral method is able compress a bigram-based deterministic WFA with $|\Sigma| + 1$ states into a non-deterministic WFA with less states. The same plot also shows curves of performance for the spectral method, where the basis corresponds to the most frequent k subsequences in training, for several k. We clearly can see that as k grows the performance improves significantly. We also can see that the choice of the number of states is less critical than with the Σ basis.

We now comment on the performance of EM, which is presented in Figure 4. The top plots present the WER as a function of the number of states, for both tagsets. Clearly, the performance significantly improves with the number of states, even for large number of states up to 150. The bottom plots show convergence curves of WER in terms of the number of EM iterations, for some selected number of states. The performance of EM improves the bigram baseline after 20 iterations, and gets somewhat stable (in terms of WER) at about 60 iterations. Note that the cost of one EM iteration requires to compute expectations on all training sequences, a computation that takes quadratic time with the number of states.

Figure 5 summarizes the best curves of all methods, for the two tagsets. For machines up to 20 states in the simplified tagset, and 10 states in the original tagset, the performance of the spectral method with extended basis is comparable of that of EM.

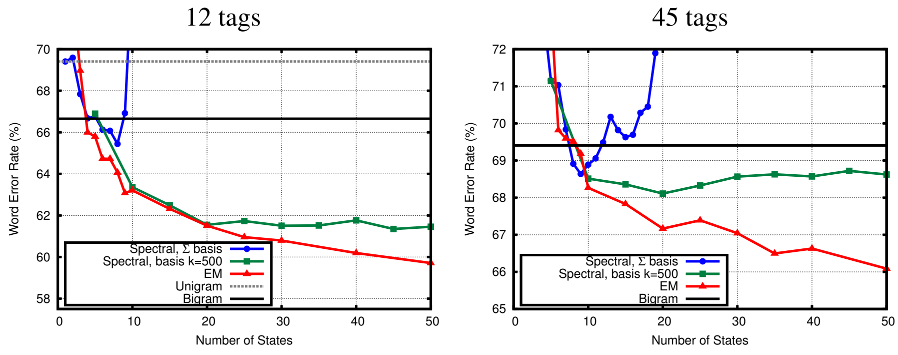


Figure 5: Comparison of different methods in terms of WER on validation data with respect to number of states. The left plot corresponds to the simplified tagset of 12 symbols, while the right plot corresponds to the tagset of 45 symbols.

Yet, the EM algorithm is able to improve the results when increasing the number of states. We should note that in our implementation, the runtime of a single EM iteration is at least twice of the total runtime of learning a model with spectral method.

6 Non-Deterministic Split Head-Automata Grammars

In this section we develop an application of the spectral method for WFA to the problem of learning split head-automata grammars (SHAG) (Eisner and Satta, 1999; Eisner, 2000), a context-free grammatical formalism whose derivations are dependency trees. A dependency tree is a type of syntactic structure where the basic element is a dependency, a syntactic relation between two words of a sentence represented as a directed arc in the tree. Figure 6 shows a dependency tree for an English sentence. In our application, we will assume that the training set will be in the form of sentences (i.e. input sequences) paired with their dependency tree. From this type of data, we will learn probabilistic SHAG models using the spectral method that will be then used to predict the most likely dependency tree for test sentences. In the rest of this section we first define SHAG formally. We then describe how the spectral method can be used to learn a SHAG, and finally we describe how we parse sentences with our SHAG models. Then, in section 7 of this article we present experiments.

6.1 SHAG

We will use $x_{i:j} = x_i x_{i+1} \cdots x_j$ to denote a sequence of symbols x_t with $i \leq t \leq j$. A SHAG generates sentences $s_{0:N}$, where symbols $s_t \in \Sigma$ with $1 \leq t \leq N$ are regular words and $s_0 = \star \notin \Sigma$ is a special root symbol. Let $\bar{\Sigma} = \Sigma \cup \{\star\}$. A derivation y, i.e. a dependency tree, is a collection of *head-modifier* sequences $\langle h, d, x_{1:T} \rangle$, where $h \in \bar{\Sigma}$ is a word, $d \in \{\text{LEFT}, \text{RIGHT}\}$ is a direction, and $x_{1:T}$ is a sequence of T words, where each $x_t \in \Sigma$ is a *modifier* of h in direction d. We say that h is the *head* of each x_t. Modifier sequences $x_{1:T}$ are ordered head-outwards, i.e. among $x_{1:T}$, x_1 is the word closest to h in the derived sentence, and x_T is the furthest. A derivation y of a sentence $s_{0:N}$ consists of a LEFT and a RIGHT head-modifier sequence for each s_t, i.e. there are always two sequences per symbol in the sentence. As special cases, the

LEFT sequence of the root symbol is always empty, while the RIGHT one consists of a single word corresponding to the head of the sentence. We denote by $\mathcal{Y}$ the set of all valid derivations. See Figure 6 to see the head-modifier sequences associated with an example dependency tree.

Assume a derivation y contains $\langle h, \text{LEFT}, x_{1:T}\rangle$ and $\langle h, \text{RIGHT}, x'_{1:T'}\rangle$. Let $\mathcal{L}(y, h)$ be the derived sentence *headed* by h, which can be expressed as

$$\mathcal{L}(y, x_T) \cdots \mathcal{L}(y, x_1)\ h\ \mathcal{L}(y, x'_1) \cdots \mathcal{L}(y, x'_{T'}).$$

The language generated by a SHAG are the strings $\mathcal{L}(y, \star)$ for any $y \in \mathcal{Y}$. [3]

6.1.1 Probabilistic SHAG

In this article we use probabilistic versions of SHAG where probabilities of head-modifier sequences in a derivation are independent of each other:

$$\mathbb{P}(y) = \prod_{\langle h,d,x_{1:T}\rangle \in y} \mathbb{P}(x_{1:T}|h, d) \quad . \tag{1}$$

In the literature, standard *arc-factored* models further assume that

$$\mathbb{P}(x_{1:T}|h, d) = \prod_{t=1}^{T+1} \mathbb{P}(x_t|h, d, \sigma_t) \quad ,$$

where x_{T+1} is always a special STOP word, and σ_t is the state of a deterministic automaton generating $x_{1:T+1}$. For example, setting $\sigma_1 = \text{FIRST}$ and $\sigma_{t>1} = \text{REST}$ corresponds to first-order models, while setting $\sigma_1 = \text{NULL}$ and $\sigma_{t>1} = x_{t-1}$ corresponds to sibling models (Eisner, 2000; McDonald et al, 2005; McDonald and Pereira, 2006).

We will define a SHAG using a collection of weighted automata to compute probabilities. Assume that for each possible head h in the vocabulary $\bar{\Sigma}$ and each direction $d \in \{\text{LEFT}, \text{RIGHT}\}$ we have a weighted automaton that computes probabilities of modifier sequences as follows:

$$\mathbb{P}(x_{1:T}|h, d) = (\boldsymbol{\alpha}_1^{h,d})^\top \mathbf{A}_{x_1}^{h,d} \cdots \mathbf{A}_{x_t}^{h,d} \boldsymbol{\alpha}_\infty^{h,d} \quad .$$

Then, this collection of weighted automata defines an *non-deterministic SHAG* that assigns a probability to each $y \in \mathcal{Y}$ according to (1).

6.2 Learning SHAG

A property of our non-deterministic SHAG models is that the probability of a derivation factors into the probability of each head-modifier sequence. In other words, the state processes only model horizontal structure of the tree, and different WFA do not interact in a derivation. In addition, in this article we make the assumption that training sequences come paired with dependency trees, i.e. we assume a supervised setting. Hence, we do not deal with the hard problem of inducing grammars from sequences.

These two facts make the application of the spectral method for WFA almost trivial. From the training set, we can decompose each dependency tree into sequences of modifiers, and create a training set for each head of direction containing the corresponding sequences of modifiers. Then, for each head and direction, we can learn WFA by direct application of the spectral method.

[3]Throughout the paper we assume we can distinguish the words in a derivation, irrespective of whether two words at different positions correspond to the same symbol.

6.3 Parsing with Non-Deterministic SHAG

Given a sentence $s_{0:N}$ we would like to find its most likely derivation,

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}(s_{0:N})} \mathbb{P}(y).$$

This problem, known as MAP inference, is known to be intractable for hidden-state structure prediction models, as it involves finding the most likely tree structure while summing out over hidden states. We use a common approximation to MAP based on first computing posterior marginals of tree edges (i.e. dependencies) and then maximizing over the tree structure (see Park and Darwiche (2004) for complexity of general MAP inference and approximations). For parsing, this strategy is sometimes known as MBR decoding; previous work has shown that empirically it gives good performance (Goodman, 1996; Clark and Curran, 2004; Titov and Henderson, 2006; Petrov and Klein, 2007). In our case, we use the non-deterministic SHAG to compute posterior marginals of dependencies. We first explain the general strategy of MBR decoding, and then present an algorithm to compute marginals.

Let (s_i, s_j) denote a dependency between head word i and modifier word j. The posterior or *marginal probability* of a dependency (s_i, s_j) given a sentence $s_{0:N}$ is defined as

$$\mu_{i,j} = \mathbb{P}((s_i, s_j) \mid s_{0:N}) = \sum_{y \in \mathcal{Y}(s_{0:N}) \,:\, (s_i,s_j) \in y} \mathbb{P}(y) \ .$$

To compute marginals, the sum over derivations can be decomposed into a product of inside and outside quantities (Baker, 1979). In Appendix A we describe an inside-outside algorithm for non-deterministic SHAG. Given a sentence $s_{0:N}$ and marginal scores $\mu_{i,j}$, we compute the parse tree for $s_{0:N}$ as

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}(s_{0:N})} \sum_{(s_i,s_j) \in y} \log \mu_{i,j} \qquad (2)$$

using the standard projective parsing algorithm for arc-factored models (Eisner, 2000). Overall we use a two-pass parsing process, first to compute marginals and then to compute the best tree.

6.4 Related Work

There have been a number of works that apply spectral learning methods to tree structures. Dhillon et al (2012) present a latent-variable model for dependency parsing, where the state process models vertical interactions between heads and modifiers, such that hidden states pass information from the root of the tree to each leaf. In their model, given the state of a head, the modifiers are independent of each other. In contrast, in our case the hidden states model interactions between the children of a head, but hidden states do not pass information vertically. In our case the application of the spectral method is straightforward, while the vertical case requires taking into account that at each node the sequence from the root to the node branches out into multiple children.

There have been extensions by Bailly et al (2010) and Cohen et al (2012) of the spectral method for probabilistic context-free grammars (PCFG), a formalism that includes SHAG. In this case the state process can model horizontal and vertical interactions simultaneously, by making use of tensor operators associated to the rules of the grammar. Recently, Cohen et al (2013a) have presented experiments to learn phrase-structure models using the a spectral method.

The works mentioned so far model a joint distribution over trees of different sizes, which is the suitable setting for models like natural language parsing. In contrast, Parikh et al (2011) presented a spectral method to learn distributions over labelings of a fixed (though arbitrary) tree topology.

In all these cases, the learning setting is supervised, in the sense that training sequences are paired with their tree structure, and the spectral algorithm is used to induce the hidden state process. A more ambitious problem is that of grammatical inference, where the goal is to induce the model only from sequences. Regarding spectral methods, Mossel and Roch (2005) study the induction of the topology of a phylogenetic tree-shaped model, and Hsu et al (2012) discuss spectral techniques to induce PCFG, with dependency grammars as a special case.

7 Experiments on Learning SHAG

In this section we present experiments with SHAG. We learn non-deterministic SHAG using different versions of the spectral algorithm, and compare them to non-deterministic SHAG learned with EM and to some baseline deterministic SHAG.

Our experiments involve fully unlexicalized models, i.e. parsing part-of-speech tag sequences. While this setting falls behind the state-of-the-art, it is nonetheless valid to analyze empirically the effect of incorporating hidden states via weighted automata, which results in large improvements. At the end, we present some analysis of the automaton learned by the spectral algorithm to see the information that is captured in the hidden state space.

All the experiments were done with the dependency version of the English WSJ Penn Treebank, using the standard partitions for training and validation (see section 5). The models were trained using the modifier sequences extracted from the training dependency trees, and they were evaluated parsing the validation set and computing the Unlabeled Attachment Score (UAS). UAS is an accuracy measure that accounts for the percentage of tokens that were assigned the correct head word (note that in a dependency tree, each word modifies exactly one head).

7.1 Methods Compared

As a SHAG is a collection of automata, each one has its own alphabet $\Sigma^{h,d}$, defined as the set of symbols ocurring in the training modifier sequences for that head h and direction d. We compare the following models:

Baseline Models Deterministic SHAG with a fixed global DFA structure. The PDFA transition probabilities for each head and direction are estimated using the training modifier sequences. We define two concrete baselines depending on the DFA structure:

DET: A single state DFA as in Figure 7(a).

DET+F: Two states, one emitting the first modifier of a sequence, and another emitting the rest, as shown in Figure 7(b) (see Eisner and Smith (2010) for a similar deterministic baseline).

EM Model A non-deterministic SHAG with n states trained with Expectation Maximization (EM) as in section 5.

Spectral Models Non-deterministic SHAG where the WFA are trained with the spectral algorithm. As in section 5, we use substring expectation estimations and then we use Lemma 3.1 to obtain WFA that approximate full sequence distributions. The number of states for each WFA is $\min(|\Sigma^{h,d}|, n)$, where n is a parameter of the model. We do experiments with two variants of the spectral method:

Σ' Basis: The basis for each WFA is $\mathcal{P}^{h,d} = \mathcal{S}^{h,d} = (\Sigma^{h,d})' = \Sigma^{h,d} \cup \{\lambda\}$. For this model, we use an additional parameter m, a minimal mass used to discard states. In each WFA, we discard the states with proportional singular value $< m$.

Extended Basis: f is a parameter of the model, namely a *cut factor* that defines the size of the basis as follows. For each WFA, we use as basis $\mathcal{P}^{h,d}$ and $\mathcal{S}^{h,d}$ the set of $|\Sigma^{h,d}|f$ most frequent training subsequences of symbols (up to length 4). Hence, f is a relative size of the basis for a WFA, proportional to the size of its alphabet. We always include the empty sequence λ in the basis.

7.2 Results

The results for the deterministic baselines were a UAS of 68.52% for DET and a UAS of 74.80% for DET+F.

In the first set of experiments with the spectral method, we evaluated the models trained with the Σ' basis. Figure 8(a) shows the resulting UAS scores in terms of the parameter n (the number of states). We plot curves for the basic spectral model with no state discarding ($m = 0$) and with state discarding for different values of minimal mass ($m > 0$). The basic model improves over the baselines, reaching a peak UAS of 79.75% with 9 states, but then the accuracy starts to drop and with 20 states it performs worse than DET+F. The curves for the models with the singular-value based state discarding strategy also have a peak at 9 states, but then they converge to a stable performance, always above the baselines. The best result is a UAS of 79.81% for $m = 0.0001$, but the best overall curve is for $m = 0.0005$, with a peak of 79.79% and converging then to 79.64%. Although very simple, our state discarding strategy seems to be effective to obtain models with stable performance.

In the second set of experiments with the spectral method, we evaluated models estimated with extended basis. Figure 8(b) shows curves for different cut factors f, plotting UAS scores in terms of the number of states.[4] Here, we clearly see that the performance largely improves and is more stable with bigger values for f.[5] The best results are clearly better than the ones of the basic model (UAS 80.90% vs. 79.81%) and, more interestingly, the curves reach stability without the need of a state discarding strategy.

The results for the experiments with EM are shown in Figure 9. The left figure plots accuracy with respect to the number of states, where we see that EM obtains improvements as the number of states increases (though for $n > 100$ the improvements are small). The right plot shows the convergence of EM in terms of accuracy relative to

[4]It must be clear that $f = 1$ is not equivalent to a Σ' basis. While both have the same basis size, the Σ' basis only has sequences of length ≤ 1, while the extended model may include longer sequences and discard unfrequent symbols.

[5]For $f > 10$ we did not see significant improvements in the performance.

the number of iterations. As in the experiments with WA, EM needs about 50 iterations to obtain a stable performance.

To summarize, in Figure 10(a) we compare the best runs of each method. In terms of accuracy, the spectral method with extended basis obtains accuracies comparable to EM. We would like to note that, as in the experiments with WFA, in terms of training time the spectral algorithm is much faster than EM (each EM iteration takes at least twice the time of running the spectral method).

7.3 Result Analysis

Our purpose in this section is to see what information is encoded in the models learned by the spectral algorithm. However, hidden state spaces are hard to interpret, and this is even harder if they are projected into a non-probabilistic space through a basis change, as in our case. To do the analysis, we build DFA that approximate the behaviour of the non-deterministic models when they generate highly probable sequences. The DFA approximations allows us to observe in a simple way some linguistically relevant phenomena encoded in the states, and to compare them with manually encoded features of well-known models. In this section we describe the DFA approximation construction method, and then we use it to analyze the most relevant unlexicalized automaton in terms of number of dependencies, namely, the automaton for h =NN and $d = \text{LEFT}$.

7.3.1 DFA Approximation for Stochastic WFA

When generating a sequence, a DFA is in a single state at each step of the generation. However, in a PNFA, what we have at each step is a vector with the probabilities of being at each state. More generally, in a WFA, at each step we have an arbitrary vector in $\mathbb{R}^n$, called the forward-state vector. For a WFA and a given sequence $x = x_1 \dots x_t$, the forward-state vector after generating x is defined as

$$\boldsymbol{\alpha}(x) = \left(\boldsymbol{\alpha}_1^\top \mathbf{A}_{x_1} \cdots \mathbf{A}_{x_t}\right)^\top .$$

While generating a sequence x, a WFA traverses the $\mathbb{R}^n$ space with a path $\boldsymbol{\alpha}(x_1)$, $\boldsymbol{\alpha}(x_1 x_2), \dots, \boldsymbol{\alpha}(x_1 \dots x_t)$, resembling a deterministic process in an infinite-state space.

To build a DFA approximation, we first compute a set of forward vectors corresponding to the most frequent prefixes of training sequences. Then, we cluster these vectors using a Group Average Agglomerative algorithm using the cosine similarity measure (Manning et al, 2008). Each cluster i defines a state in the DFA, and we say that a sequence $m_{1:t}$ is in state i if its corresponding forward vector at time t is in cluster i. The transitions in the DFA are defined using a procedure that looks at how sequences traverse the states. If a sequence $m_{1:t}$ is at state i at time $t-1$, and goes to state j at time t, then we define a transition from state i to state j with label m_t. This procedure may require merging states to give a consistent DFA, because different sequences may define different transitions for the same states and modifiers. After doing a merge, new merges may be required, so the procedure must be repeated until a DFA is obtained.

Figure 11 illustrates the DFA construction process showing fictitious forward vectors in a 3 dimensional space. The forward vectors correspond to the prefixes of the sequence “JJ JJ DT STOP”, a frequent left-modifier sequence for nouns. In this example, we construct a 3 state automaton by clustering the vectors into three different sets and then defining the transitions as described in the previous paragraphs.

7.3.2 Experiments on Unlexicalized WFA

A DFA approximation for the automaton (NN,LEFT) is shown in Figure 12. The vectors were originally divided into ten clusters, but the DFA construction required two state mergings, leading to a eight state automaton. The state named I is the initial state. Clearly, we can see that there are special states for punctuation (state 9) and coordination (states 1 and 5). States 0 and 2 are harder to interpret. To understand them better, we computed an estimation of the probabilities of the transitions, by counting the number of times each of them is used. We found that our estimation of generating STOP from state 0 is 0.67, and from state 2 it is 0.15. Interestingly, state 2 can transition to state 0 generating PRP$, POS or DT, that are usual endings of modifier sequences for nouns (recall that modifiers are generated head-outwards, so for a left automaton the final modifier is the left-most modifier in the sentence).

8 Conclusion

The central objective of this paper was to offer a broad view of the main results in spectral learning in the context of grammatical inference, and more precisely in the context of learning weighted automata. With this goal in mind, we presented the recent advances in the field in a way that makes the main underlying principles of spectral learning accessible to a wide audience.

We believe this to be useful since spectral methods are becoming an interesting alternative to the classical EM algorithms widely used for grammatical inference. One of the attractiveness of the spectral approach resides in its computational efficiency (at least in the context of automata learning). This efficiency might open the door to large-scale applications of automata learning, where models can be inferred from big data collections.

Apart from scalability, some important questions about the different properties of EM versus spectral learning remain unanswered. That been said, in broad terms we can make two main distinctions between spectral learning and EM:

- EM attempts to minimize the KL divergence between the model distribution and the observed distribution. In contrast, the spectral method attempts to minimize an ℓ_p distance between model and observed distribution.
- EM searches for stable points of the likelihood function. Instead, the spectral method finds an approximate minimizer of a global loss function.

Most empirical studies, including ours, suggest that the statistical performance of spectral methods is similar to that of EM (e.g. see Cohen et al (2013b) for experiments learning latent-variable probabilistic context free grammars). However, our empirical understanding is still quite limited and more research needs to be done to understand the relative performance of each algorithm with respect to the complexity of the target model (i.e., size of the alphabet and number of states). Nonetheless, spectral methods offer a very competitive computational performance when compared to iterative methods like EM.

A key difference between the spectral method and other approaches to induce weighted automata is at the conceptual level, particularly in the way in which the learning problem is framed. This conceptual difference is precisely what we tried to emphasize in our presentation of the subject. In a snapshot, the central idea of the spectral approach to learning functions over $\Sigma^\star$ is to directly exploit recurrence relations satisfied by families of functions. This is done by providing algebraic formulations of these recurrence relations.

Because spectral learning for grammatical inference is still a young field, many problems remain open. At a technical level, we have already mentioned the two most important: how to choose a sample-dependent basis for the Hankel matrices fed to the method, and how to guarantee that the output WFA is stochastic or probabilistic. The former problem has been discussed at large in Section 4.2.2, where we gave heuristics for choosing the input parameters given to the algorithm. The latter problem has received less attention in the present paper, mainly because our experimental framework is not affected by it. However, ensuring the output of the spectral method is a proper probability distribution is important in many applications. Different solutions have been proposed to address this issue: Bailly (2011) gave a spectral method for Quadratic WFA which by definition always define a non-negative function; heuristics to modify the output of a spectral algorithm in order to enforce non-negativity were discussed in (Cohen et al, 2013b) in the context of PCFG, though they also apply to WFA; for HMM one can use methods based on spectral decompositions of tensors to overcome this problem (Anandkumar et al, 2012b); one can obtain probabilistic WFA by imposing some convex constraints on the search space of the optimization-based spectral method presented in (Balle et al, 2012b). All these methods rely on variations of the SVD-based method presented in this paper. An interesting exercise would be to compare their behavior in practical applications.

Besides these technical questions, several conceptual questions regarding spectral learning and its relations to EM remain open. In particular, we would like to have a deeper understanding of the relations between EM, spectral learning and split-merge algorithms, both from a theoretical perspective and from a practical point of view. On the other hand, the principles that underlie spectral learning can be applied to any computational or probabilistic model with some notion of locality, in the sense that the model admits some strong Markov-like conditional independece assumptions. Several extensions along these lines can already be found in the literature, but the limits of these techniques remain largely unknown. From the perspective of grammatical inference, learning beyond stochastic rational languages is the most promising line of work.

References

Anandkumar A, Foster DP, Hsu D, Kakade SM, Liu YK (2012a) Two SVDs suffice: Spectral decompositions for probabilistic topic modeling and latent dirichlet allocation. CoRR abs/1204.6703

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

Anandkumar A, Hsu D, Kakade SM (2012c) A method of moments for mixture models and hidden Markov models. COLT

Bailly R (2011) Quadratic weighted automata: Spectral algorithm and likelihood maximization. Journal of Machine Learning Research

Bailly R, Denis F, Ralaivola L (2009) Grammatical inference as a principal component analysis problem. In: ICML, p 5

Bailly R, Habrard A, Denis F (2010) A spectral approach for probabilistic grammatical inference on trees. In: Hutter M, Stephan F, Vovk V, Zeugmann T (eds) ALT, Springer, Lecture Notes in Computer Science, vol 6331, pp 74–88

Baker JK (1979) Trainable grammars for speech recognition. In: Klatt DH, Wolf JJ (eds) Speech Communication Papers for the 97th Meeting of the Acoustical Society of America, pp 547–550

Balle B (2013) Learning finite-state macines: Algorithmic and statistical aspects. PhD thesis, Universitat Politcnica de Catalunya

Balle B, Mohri M (2012) Spectral learning of general weighted automata via constrained matrix completion. In: Advances in Neural Information Processing Systems 25, pp 2168–2176

Balle B, Quattoni A, Carreras X (2011) A spectral learning algorithm for finite state transducers. ECML–PKDD

Balle B, Castro J, Gavaldà R (2012a) Learning probabilistic automata: A study in state distinguishability. Theoretical Computer Science

Balle B, Quattoni A, Carreras X (2012b) Local loss optimization in operator models: A new insight into spectral learning. In: Langford J, Pineau J (eds) Proceedings of the 29th International Conference on Machine Learning (ICML-12), Omnipress, New York, NY, USA, ICML '12, pp 1879–1886

Beimel A, Bergadano F, Bshouty N, Kushilevitz E, Varricchio S (2000) Learning functions represented as multiplicity automata. JACM

Berstel J, Reutenauer C (1988) Rational Series and Their Languages. Springer

Boots B, Siddiqi S, Gordon G (2011) Closing the learning planning loop with predictive state representations. I J Robotic Research

Carlyle JW, Paz A (1971) Realizations by stochastic finite automata. J Comput Syst Sci 5(1):26–40

Castro J, Gavaldà R (2008) Towards feasible PAC-learning of probabilistic deterministic finite automata. In: International colloquium on Grammatical Inference (ICGI)

Clark A, Thollard F (2004) PAC-learnability of probabilistic deterministic finite state automata. Journal of Machine Learning Research

Clark S, Curran JR (2004) Parsing the wsj using ccg and log-linear models. In: Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume, Barcelona, Spain, pp 103–110, DOI 10.3115/1218955.1218969, URL http://www.aclweb.org/anthology/P04-1014

Cohen SB, Stratos K, Collins M, Foster DP, Ungar L (2012) Spectral learning of latent-variable pcfgs. In: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Jeju Island, Korea, pp 223–231, URL http://www.aclweb.org/anthology/P12-1024

Cohen SB, Stratos K, Collins M, Foster DP, Ungar L (2013a) Experiments with spectral learning of latent-variable pcfgs. In: Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Atlanta, Georgia, pp 148–157, URL http://www.aclweb.org/anthology/N13-1015

Cohen SB, Stratos K, Collins M, Foster DP, Ungar L (2013b) Experiments with spectral learning of latent-variable pcfgs. Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (HTL-NAACL)

Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society

Denis F, Esposito Y (2008) On rational stochastic languages. Fundam Inform

Dhillon P, Rodu J, Collins M, Foster D, Ungar L (2012) Spectral dependency parsing with latent variables. In: Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, Association for Computational Linguistics, Jeju Island, Korea, pp 205–213, URL http://www.aclweb.org/anthology/D12-1019

Eisner J (2000) Bilexical grammars and their cubic-time parsing algorithms. In: Bunt H, Nijholt A (eds) Advances in Probabilistic and Other Parsing Technologies, Kluwer Academic Publishers, pp 29–62

Eisner J, Satta G (1999) Efficient parsing for bilexical context-free grammars and head-automaton grammars. In: Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL), University of Maryland, pp 457–464

Eisner J, Smith NA (2010) Favor short dependencies: Parsing with soft and hard constraints on dependency length. In: Bunt H, Merlo P, Nivre J (eds) Trends in Parsing Technology: Dependency Parsing, Domain Adaptation, and Deep Parsing, Springer, chap 8, pp 121–150

Fliess M (1974) Matrices de Hankel. Journal de Mathématiques Pures et Appliquées 53:197–222

Goodman J (1996) Parsing algorithms and metrics. In: Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Santa Cruz, California, USA, pp 177–183, DOI 10.3115/981863.981887, URL http://www.aclweb.org/anthology/P96-1024

Hsu D, Kakade SM, Zhang T (2009) A spectral algorithm for learning hidden markov models. In: Proc. of COLT

Hsu D, Kakade SM, Liang P (2012) Identifiability and unmixing of latent parse trees. In: Advances in Neural Information Processing Systems (NIPS)

Kearns MJ, Mansour Y, Ron D, Rubinfeld R, Schapire RE, Sellie L (1994) On the learnability of discrete distributions. Symposium on Theory of Computation (STOC)

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

Manning CD, Raghavan P, Schütze H (2008) Introduction to Information Retrieval, 1st edn. Cambridge University Press, Cambridge

Marcus MP, Santorini B, Marcinkiewicz MA (1994) Building a large annotated corpus of english: The penn treebank. Computational Linguistics 19

McDonald R, Pereira F (2006) Online learning of approximate dependency parsing algorithms. In: Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics, pp 81–88

McDonald R, Pereira F, Ribarov K, Hajic J (2005) Non-projective dependency parsing using spanning tree algorithms. In: Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Vancouver, British Columbia, Canada, pp 523–530, URL http://www.aclweb.org/anthology/H/H05/H05-1066

Mohri M (2009) Weighted automata algorithms. In: Handbook of Weighted Automata, Springer

Mossel E, Roch S (2005) Learning nonsingular phylogenies and hidden markov models. In: Proc. of STOC

Palmer N, Goldberg PW (2007) PAC-learnability of probabilistic deterministic finite state automata in terms of variation distance. Theor Comput Sci

Parikh A, Song L, Xing E (2011) A spectral algorithm for latent tree graphical models. ICML

Park JD, Darwiche A (2004) Complexity results and approximation strategies for map explanations. Journal of Artificial Intelligence Research 21:101–133

Paskin M (2001) Cubic-time parsing and learning algorithms for grammatical bigram models. Tech. Rep. UCB/CSD-01-1148, University of California, Berkeley

Petrov S, Klein D (2007) Improved inference for unlexicalized parsing. In: Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference, Association for Computational Linguistics, Rochester, New York, pp 404–411, URL http://www.aclweb.org/anthology/N/N07/N07-1051

Petrov S, Das D, McDonald R (2012) A universal part-of-speech tagset. In: Proceedings of LREC

Ron D, Singer Y, Tishby N (1998) On the learnability and usage of acyclic probabilistic finite automata. Journal of Computing Systems Science

Salomaa A, Soittola M (1978) Automata-Theoretic Aspects of Formal Power Series. Springer-Verlag: New York

Schützenberger M (1961) On the definition of a family of automata. Information and Control 4:245–270

Siddiqi S, Boots B, Gordon G (2010) Reduced-rank hidden markov models. AISTATS

Song L, Boots B, Siddiqi S, Gordon G, Smola A (2010) Hilbert space embeddings of hidden markov models. ICML

Titov I, Henderson J (2006) Loss minimization in parse reranking. In: Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Sydney, Australia, pp 560–567, URL http://www.aclweb.org/anthology/W/W06/W06-1666

Valiant LG (1984) A theory of the learnable. Communications of the ACM

Wiewiora E (2005) Learning predictive representations from a history. In: Proceedings of the 22nd international conference on Machine learning, ACM, pp 964–971

A An Inside-Outside Algorithm for Non-Deterministic SHAG

In this section we sketch an algorithm to compute marginal probabilities of dependencies. Our algorithm is an adaptation of the parsing algorithm for SHAG by Eisner and Satta (1999) to the case of non-deterministic head-automata, and has a runtime cost of $O(n^2N^3)$, where n is the number of states of the model, and N is the length of the input sentence. Hence the algorithm maintains the standard cubic cost on the sentence length, while the quadratic cost on n is inherent to the computations defined by our model in Eq. (2). The main insight behind our extension is that, because the computations of our model involve state-distribution vectors, we need to extend the standard inside/outside quantities to be in the form of such state-distribution quantities.[6]

Throughout this section we assume a fixed sentence $s_{0:N}$. Let $\mathcal{Y}(x_{i:j})$ be the set of derivations that yield a subsequence $x_{i:j}$. For a derivation y, we use $\mathrm{root}(y)$ to indicate the root word of it, and use $(x_i, x_j) \in y$ to refer a dependency in y from head x_i to modifier x_j. Following Eisner and Satta (1999), we use decoding structures related to complete half-constituents (or "triangles", denoted C) and incomplete half-constituents (or "trapezoids", denoted I), each decorated with a direction (denoted L and R). We assume familiarity with their algorithm.

We define $\boldsymbol{\theta}^{\mathrm{I,R}}_{i,j} \in \mathbb{R}^n$ as the inside score-vector of a right trapezoid dominated by dependency (s_i, s_j),

$$\boldsymbol{\theta}^{\mathrm{I,R}}_{i,j} = \sum_{\substack{y \in \mathcal{Y}(s_{i:j}) \,:\, (s_i,s_j) \in y \;, \\ y = \{\langle s_i, \mathrm{R}, x_{1:t} \rangle\} \,\cup\, y' \;,\; x_t = s_j}} \mathbb{P}(y') \boldsymbol{\alpha}^{s_i,\mathrm{R}}(x_{1:t}) \;.$$

The term $\mathbb{P}(y')$ is the probability of head-modifier sequences in the range $s_{i:j}$ that do not involve s_i. The term $\boldsymbol{\alpha}^{s_i,\mathrm{R}}(x_{1:t})$ is a *forward* state-distribution vector —the qth coordinate of the vector is the probability that s_i generates right modifiers $x_{1:t}$ and remains at state q. Similarly, we define $\boldsymbol{\phi}^{\mathrm{I,R}}_{i,j} \in \mathbb{R}^n$ as the outside score-vector of a right trapezoid, as

$$\boldsymbol{\phi}^{\mathrm{I,R}}_{i,j} = \sum_{\substack{y \in \mathcal{Y}(s_{0:i} s_{j:n}) \,:\, \mathrm{root}(y) = s_0, \\ y = \{\langle s_i, \mathrm{R}, x_{t:T} \rangle\} \,\cup\, y' \;,\; x_t = s_j}} \mathbb{P}(y') \boldsymbol{\beta}^{s_i,\mathrm{R}}(x_{t+1:T}) \;,$$

[6]Technically, when working with the projected operators the state-distribution vectors will not be distributions in the formal sense. However, they correspond to a projection of a state distribution, for some projection that we do not recover from data (namely a change of basis as discussed in section 2.2). This projection has no effect on the computations because it cancels out.

where $\boldsymbol{\beta}^{s_i,\mathrm{R}}(x_{t+1:T}) \in \mathbb{R}^n$ is a *backward* state-distribution vector —the qth coordinate is the probability of being at state q of the right automaton of s_i and generating $x_{t+1:T}$. Analogous inside-outside expressions can be defined for the rest of structures (left/right triangles and trapezoids). With these quantities, we can compute marginals as

$$\mu_{i,j} = \begin{cases} (\boldsymbol{\theta}^{\mathrm{I,R}}_{i,j})^\top \, \boldsymbol{\phi}^{\mathrm{I,R}}_{i,j} \, Z^{-1} & \text{if } i < j \;, \\ (\boldsymbol{\theta}^{\mathrm{I,L}}_{i,j})^\top \, \boldsymbol{\phi}^{\mathrm{I,L}}_{i,j} \, Z^{-1} & \text{if } j < i \;, \end{cases}$$

where $Z = \sum_{y \in \mathcal{Y}(s_{0:N})} \mathbb{P}(y) = (\boldsymbol{\theta}^{\mathrm{C,R}}_{0,N})^\top \, \boldsymbol{\alpha}^{\star,\mathrm{R}}_\infty$.

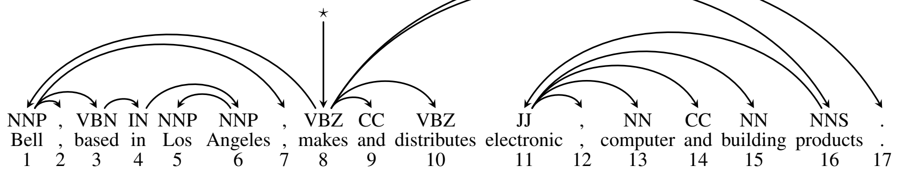
Finally, we sketch the equations for computing inside scores in $O(N^3)$ time. The outside equations can be derived analogously (see Paskin (2001)). For $0 \leq i < j \leq N$:

$$\boldsymbol{\theta}^{\mathrm{C,R}}_{i,i} = \boldsymbol{\alpha}^{s_i,\mathrm{R}}_1 \tag{3}$$

$$\boldsymbol{\theta}^{\mathrm{C,R}}_{i,j} = \textstyle\sum_{k=i+1}^{j} \left((\boldsymbol{\theta}^{\mathrm{C,R}}_{k,j})^\top \boldsymbol{\alpha}^{s_k,\mathrm{R}}_\infty \right) \boldsymbol{\theta}^{\mathrm{I,R}}_{i,k} \tag{4}$$

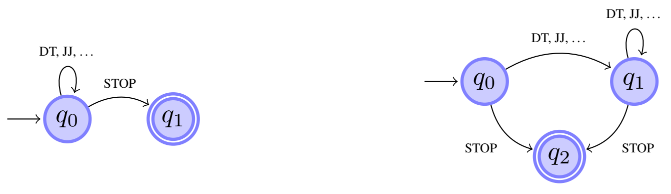
$$\boldsymbol{\theta}^{\mathrm{I,R}}_{i,j} = \textstyle\sum_{k=i}^{j} \left((\boldsymbol{\theta}^{\mathrm{C,L}}_{k+1,j})^\top \boldsymbol{\alpha}^{s_j,\mathrm{L}}_\infty \right) (\mathbf{A}^{s_i,\mathrm{R}}_{s_j})^\top \boldsymbol{\theta}^{\mathrm{C,R}}_{i,k} \tag{5}$$

Fig. 13 illustrates these equations. Fig. 13(a) corresponds to the basic case of Eq. 3, and Figs. 13(b) and 13(c) correspond respectively to Eqs. 4 and 5 for a fixed k.



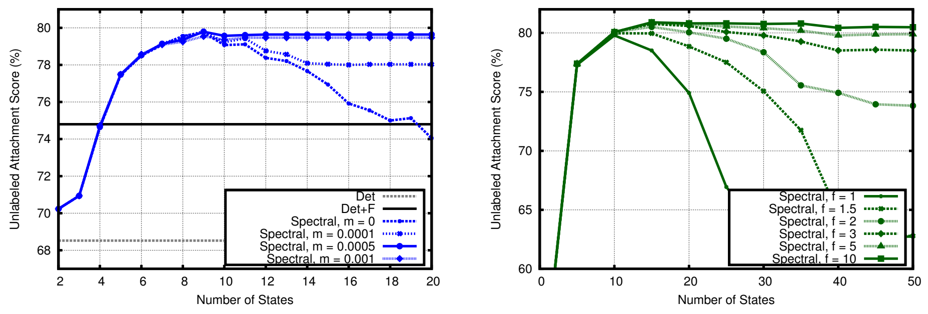
head	*dir.*	*modifiers*	*head*	*dir.*	*modifiers*
$\star$	LEFT	λ	CC_9	LEFT	λ
$\star$	RIGHT	VBZ_8	CC_9	RIGHT	λ
NNP_1	LEFT	λ	VBZ_{10}	LEFT	λ
NNP_1	RIGHT	$,_2$ VBN_3 $,_7$	VBZ_{10}	RIGHT	λ
$,_2$	LEFT	λ	JJ_{11}	LEFT	λ
$,_2$	RIGHT	λ	JJ_{11}	RIGHT	$,_{12}$ NN_{13} CC_{14} NN_{15}
VBN_3	LEFT	λ	$,_{12}$	LEFT	λ
VBN_3	RIGHT	IN_4	$,_{12}$	RIGHT	λ
IN_4	LEFT	λ	NN_{13}	LEFT	λ
IN_4	RIGHT	NNP_6	NN_{13}	RIGHT	λ
NNP_5	LEFT	λ	CC_{14}	LEFT	λ
NNP_5	RIGHT	λ	CC_{14}	RIGHT	λ
NNP_6	LEFT	NNP_5	NN_{15}	LEFT	λ
NNP_6	RIGHT	λ	NN_{15}	RIGHT	λ
$,_7$	LEFT	λ	NNS_{16}	LEFT	JJ_{11}
$,_7$	RIGHT	λ	NNS_{16}	RIGHT	λ
VBZ_8	LEFT	NNP_1	$._{17}$	LEFT	λ
VBZ_8	RIGHT	CC_9 VBZ_{10} NNS_{16} $._{17}$	$._{17}$	RIGHT	λ

Figure 6: An example of a dependency tree. Each arc in the dependency tree represents a syntactic relation between a head token (the origin of each arc) and one of its modifier tokens (the arc destination). The special root token is represented by $\star$. For each token, we print the part-of-speech, the word itself and its position, though our head-automata grammars only model sequences of part of speech tags. The table below the tree prints all head-modifier sequences of the tree. The subscript number next to each tag is the position of the corresponding token. Note that for a sentence of n tokens there are always $2(n+1)$ sequences, even though most of them are empty.



(a) (b)

Figure 7: Unlexicalized DFAs illustrating the features encoded in the deterministic baselines. For clarity, on each automata we added a separate final state, and a special ending symbol STOP. (a) DET. (b) DET+F.



(a) (b)

Figure 8: Accuracy of the different spectral methods (UAS in function of the number of states). (a) Curves for the Σ' basis: basic spectral method ($m = 0$) and state discarding with minimum mass $m > 0$. (b) Curves for the extended basis with different cut factors f.

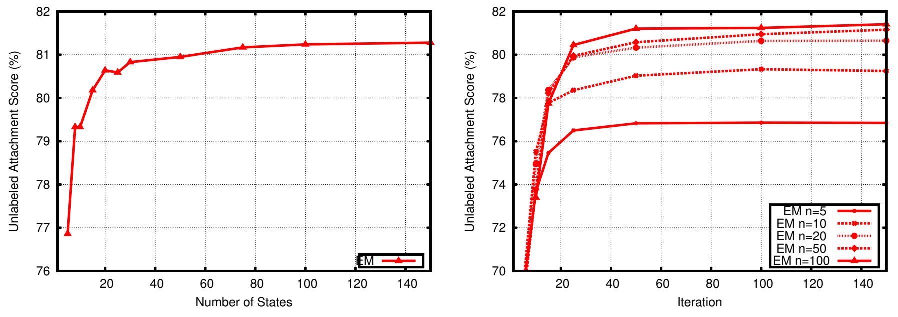


Figure 9: Accuracy for EM. (a) UAS with respect to number of states. (b) UAS with respect to number of training iterations. Curves for different numbers of states n.

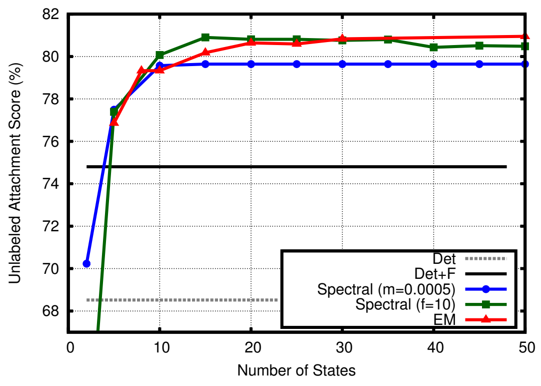
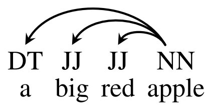


Figure 10: Comparison of different methods in terms of UAS with respect to the number of states.

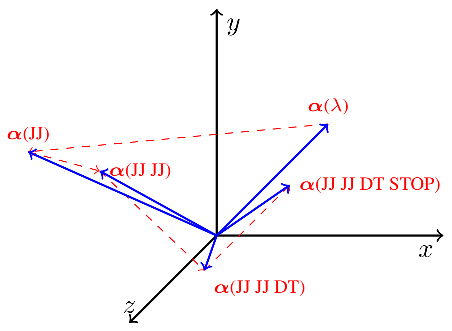
(a)



(b)

$$c_0 = \{\boldsymbol{\alpha}(\lambda), \boldsymbol{\alpha}(\text{JJ JJ DT STOP})\}$$
$$c_1 = \{\boldsymbol{\alpha}(\text{JJ}), \boldsymbol{\alpha}(\text{JJ JJ})\}$$
$$c_2 = \{\boldsymbol{\alpha}(\text{JJ JJ DT})\}$$

(c)



(d)

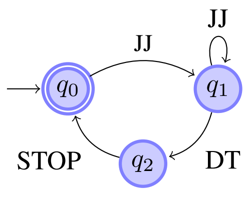


Figure 11: Example of construction of a 3 state DFA approximation. (a) Concrete example for the modifier sequence “JJ JJ DT STOP”. (b) Forward vectors $\boldsymbol{\alpha}$ for the prefixes of the sequence. (c) Cosine similarity clustering. (d) Resulting DFA after adding the transitions.

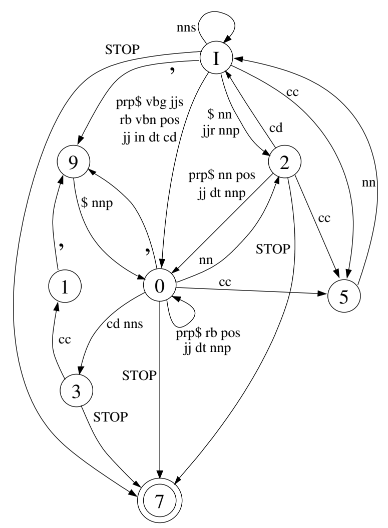


Figure 12: DFA approximation for the generation of NN left modifier sequences.

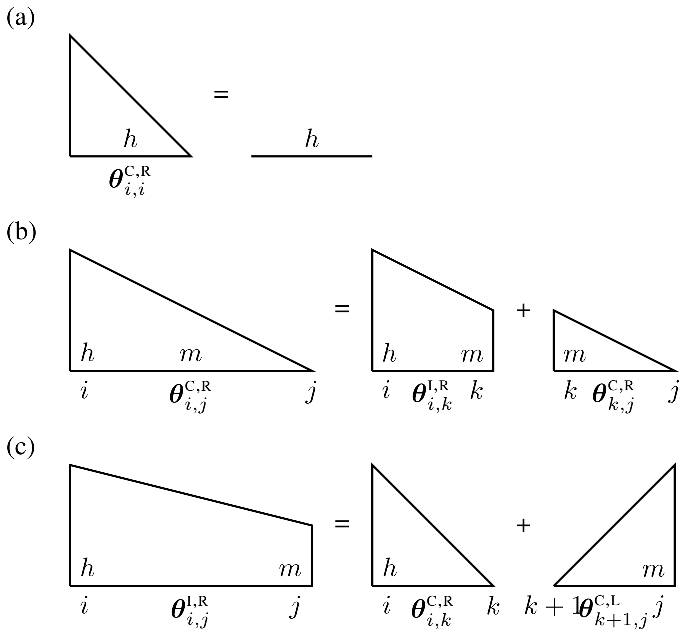

(a)
h
$\boldsymbol{\theta}^{\mathrm{C,R}}_{i,i}$
=
h
(b)
h
m
i
$\boldsymbol{\theta}^{\mathrm{C,R}}_{i,j}$
j
=
h
m
i
$\boldsymbol{\theta}^{\mathrm{I,R}}_{i,k}$
k
+
m
k
$\boldsymbol{\theta}^{\mathrm{C,R}}_{k,j}$
j
(c)
h
m
i
$\boldsymbol{\theta}^{\mathrm{I,R}}_{i,j}$
j
=
h
i
$\boldsymbol{\theta}^{\mathrm{C,R}}_{i,k}$
k
+
m
$k+1$
$\boldsymbol{\theta}^{\mathrm{C,L}}_{k+1,j}$
j

Figure 13: Graphical depiction of the inside scores computations for the right half-constituents. Computations for left half-constituents are symmetrical. (a) Empty right half-constituent (“triangle”). (b) Non-empty complete right half-constituent (“triangle”). (c) Incomplete right half-constituent (“trapezoid”).