A Full Numerical Rank Basis Selection Algorithm for Spectral Learning

Yibin Zhao
University of Toronto
ybzhao@cs.toronto.edu

Gerald Penn
University of Toronto
gpenn@cs.toronto.edu

Abstract

This paper re-visits the basis selection methods for Hankel matrices as in spectral learning methods. The known algorithms have many limitations, one being that no such algorithm could find a full numerical rank sub-block. In this paper, we present an efficient, deterministic divide-and-conquer algorithm that finds a full numerical rank basis. Our solution is especially appealing due to its stable behaviour and its versatility in training Hidden Markov Models. We also provide two parameters for controlling the trade-off between computational complexity and model performance.

1 Introduction

Spectral learning methods have attracted the attention of many researchers in computational linguistics (CL) and the natural language processing (NLP) community due to the strong theoretical properties of the algorithms and the advantages of Hidden Markov Models in processing sequences (Hsu et al., 2012; Balle et al., 2011; Cohen et al., 2012).

It has been challenging to obtain language-modelling results with spectral methods that are competitive with neural methods mainly due to two issues, as pointed out by Quattoni and Carreras (2019): the scalability of handling long-range dependencies, and the loss function for the spectral learning scheme does not align with that of the language modelling task. For the first issue, a natural solution is to increase the length of sequences for learning the distribution. As a consequence, the size of the Hankel matrix increases so drastically that it is not practical to perform factorizations (i.e. singular value decomposition) on it. Hence, people seek to find informative yet much smaller sub-blocks of the Hankel matrices, induced by a set of prefixes and a set of suffixes, to perform the matrix factorizations. We call this *basis selection* for Hankel matrices.

Over the past two decades, many basis selection methods have been proposed. One basic approach is to choose all the prefixes and suffixes observed in the sample (Bailly et al., 2009). Although this method has many good theoretical properties (Denis et al., 2016) for modelling a probability and is always optimal in terms of loss, it is practically infeasible for large corpora. Previously, the most popular approach was to choose prefixes and suffixes of length $\leq T$ (Hsu et al., 2012; Siddiqi et al., 2010). Bases chosen by this method tend to lose too much information, making it hard to model long-term dependencies. Balle et al. (2012) provided a randomized algorithm by cutting a string at each time step over n iterations, resulting in an $n \times n$ sub-block. When n is sufficiently large, there is a high probability the algorithm returns a full-rank sub-block of the empirical Hankel matrix. But n must be $\mathrm{polylog}\, n$ times the rank of a complete Hankel matrix in order to obtain a sub-block with nearly full rank.

The work of Quattoni et al. (2017) proposed an algorithm that performs maximum matching on the underlying bipartite graphs realized by Hankel matrices to get a full structural rank sub-block. The bases generated by this method are very compact. This method was the first to exploit the structural properties of Hankel matrices in this task. The authors claimed that bases generated by bipartite matching have approximately the full numerical rank of a complete empirical Hankel matrix. But there is no theoretical analysis, and the matching properties that they used in claiming the rank observation do not hold in most cases.

In this paper, we present an efficient algorithm for finding a compact, full-rank basis of an empirical Hankel matrix in a divide-and-conquer approach. Our method recursively computes the basis for sets of sub-strings with a fixed first symbol. The algorithm then combines such computed bases for each first symbol in the alphabet. We also present a

trade-off between the run-time of the algorithm and the size of the resulting basis through the choice of parameters called α and γ. Due to the nice theoretical properties of a full numerical rank sub-block (Denis et al., 2016) in learning a weighted automaton and the limitation of each established method mentioned above, we believe that our basis selection method is very competitive and can be an excellent complement to the method of Quattoni et al. (2017) for improving the accuracy of language models without overly increasing the run time for both training and testing.

2 Preliminaries

2.1 Notation

We start by introducing the notation we use throughout this paper. Let Σ be a finite alphabet with $|\Sigma|$ symbols and let ϵ denote the null sequence. We use Σ^* to denote the set of all strings over Σ. The concatenation of two strings $x, x' \in \Sigma^*$ is denoted by $x \cdot x'$. $\mathcal{P}$ is a set of unique prefixes and $\mathcal{S}$ is a set of unique suffixes in sample W. We always assume $\epsilon \in \mathcal{P}$ and $\epsilon \in \mathcal{S}$ unless stated otherwise. For some strings q and t, we define $\mathcal{P}(q) = \{p \in \mathcal{P} | \exists v \in \Sigma,^* p = q \cdot v\}$ and $\mathcal{S}(t) = \{s \in \mathcal{S} | \exists u \in \Sigma, *s = u \cdot a\}$. For a suffix t, the set of all prefixes of strings with suffix t is denoted by $\mathcal{P}^*(t) = \{p \in \mathcal{P} | \exists s \in \mathcal{S}(t), p \cdot s \in W\}$. We similarly define $\mathcal{S}^*(q) = \{s \in \mathcal{S} | \exists p \in \mathcal{P}(q), p \cdot s \in W\}$. Let a be some symbol in Σ, we define $\mathcal{P}_a = \{v \in \Sigma^* | a \cdot v \in \mathcal{P}\}$ and $\mathcal{S}_a = \{u \in \Sigma^* | u \cdot a \in \mathcal{S}\}$. Note that both $\mathcal{P}_a$ and $\mathcal{S}_a$ contain ϵ. Also, $\mathcal{S}_a^* = \{s \in \mathcal{S} | \exists p \in \mathcal{P}_a, a \cdot p \cdot s \in W\}$ and $\mathcal{P}_a^* = \{p \in \mathcal{P} | \exists s \in \mathcal{S}_a, p \cdot s \cdot a \in W\}$.

For a sequence $x = x_1 \cdots x_n \in \Sigma,^*$ we let $x^\top = x_n \cdots x_1$ denote the reversed sequence of x. If we define $W^\top = \{w^\top | w \in W\}$ as the multi-set of reverse sequences, then $\mathbf{H}(W^\top) = \mathbf{H}^\top(W)$. In addition, we let $\mathcal{P}^\top = \{p^\top | p \in \mathcal{P}\}$ and $\mathcal{S}^\top = \{s^\top | s \in \mathcal{S}\}$. $\mathcal{S}^\top$ and $\mathcal{P}^\top$ are the respective prefix and suffix set of sample $W^\top$.

For a matrix $\mathbf{M}$, we denote the ith row of $\mathbf{M}$ by $\mathbf{M}(i,:)$, the jth column by $\mathbf{M}(:,j)$. For $i \geq 1$, we use $\boldsymbol{e}_i$ to denote the standard basis vector with the i-th entry as 1 and 0 otherwise.

2.2 Non Deterministic Weighted Finite State Automata

Here, we define a class of Non-Deterministic Weighted Automaton (WA) over strings. Let $x = x_1 \cdots x_n$ be a sequence of length n over finite alphabet Σ. A WA with k states is defined as a tuple: $A = (\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\}_{\sigma \in \Sigma})$ where $\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_\infty \in \mathbb{R}^k$ are the initial and final weight vectors and $\mathbf{A}_\sigma \in \mathbb{R}^{k \times k}$ are the transition matrices associated with each letter $\sigma \in \Sigma$. The function $f_A : \Sigma^* \to [0, 1]$ realized by a WA A is defined as:

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

The equation above is an algebraic representation of the computation performed by a WA on a sequence x. For an explanation of this observation, we refer readers to (Hsu et al., 2012; Balle et al., 2012; Quattoni et al., 2017).

A stochastic language $\mathcal{L} : \Sigma^* \to \mathbb{R}$ is a probability distribution over $\Sigma.^*$ If for every $x \in \Sigma,^* \mathcal{L}(x) > 0$, we say that $\mathcal{L}$ has full-support. We say $\mathcal{L}$ is rational when there exists a WA A such that $f_A(x) = \mathcal{L}(x)$ for any sequence x. For our purpose, we assume the stochastic language that we sample from is rational and has full-support. We refer readers to (Denis et al., 2016; Balle et al., 2012) for a rigorous discussion of stochastic languages in relation to WA and spectral learning.

2.3 Hankel Matrices and Underlying Graphs

We now introduce the concept of a Hankel matrix in WA. This paper focuses on Hankel matrices in the context of learning a stochastic language $\mathcal{L}$ from sample W.

The Hankel matrix of $\mathcal{L}$ is a bi-infinite matrix $\mathbf{H}_\mathcal{L} : \Sigma^* \times \Sigma^* \to \mathbb{R}$ with entries indexed by prefixes and suffixes. The rank of $\mathcal{L}$ is defined as $\text{rank}(\mathcal{L}) = \text{rank}(\mathbf{H}_\mathcal{L})$. Given $\mathcal{P}, \mathcal{S} \in \Sigma,^*$ a Hankel sub-block is denoted by $\mathbf{H}_\mathcal{L}^{(\mathcal{P},\mathcal{S})} \in \mathbb{R}^{|\mathcal{P}| \times |\mathcal{S}|}$. The rows and columns of $\mathbf{H}_\mathcal{L}^{(\mathcal{P},\mathcal{S})}$ are indexed by $\mathcal{P}$ and $\mathcal{S}$ with $\mathbf{H}_\mathcal{L}^{(\mathcal{P},\mathcal{S})}(p, s) = \mathcal{L}(p \cdot s)$. We say that $(\mathcal{P}, \mathcal{S})$ is a basis for $\mathcal{L}$ if $\text{rank}(\mathcal{L}) = \text{rank}(\mathbf{H}_\mathcal{L})$. For any symbol $\sigma \in \Sigma$, we define $\mathbf{H}_\sigma \in \mathbb{R}^{|\mathcal{P}| \times |\mathcal{S}|}$ as $\mathbf{H}_\sigma(p, s) = \mathcal{L}(p \cdot \sigma \cdot s)$.

In the context of learning, the *empirical* Hankel matrix of a finite multi-set W sampled $i.i.d.$ from $\mathcal{L}$ is denoted by $\mathbf{H}(W)$, or simply, $\mathbf{H}$. For any string $x \in \Sigma,^*$ we define $\mathbf{H}_x$ by $\mathbf{H}_x(p, s) = \mathbf{1}_{p \cdot s = x}$ and $\mathbf{H} = \frac{1}{|W|} \sum_{w \in W} \mathbf{H}_w$. Since W is finite, we can always find a minimal $\mathcal{P}, \mathcal{S} \in \Sigma^*$ such that $\mathbf{H}^{(\mathcal{P},\mathcal{S})}$ contains all the non-trivial entries of $\mathbf{H}(W)$. We say this matrix is the complete (empirical) Hankel matrix. In addition, we define $\boldsymbol{h}_x(x') = \mathbf{1}_{x'=x}$, and $\boldsymbol{h}_\mathcal{P} \in \mathbb{R}^{|\mathcal{P}|}, \boldsymbol{h}_\mathcal{S} \in \mathbb{R}^{|\mathcal{S}|}$ by $\boldsymbol{h}_\mathcal{P}(p) = \frac{1}{|W|} \sum_{w \in W} \boldsymbol{h}_w(p)$ and $\boldsymbol{h}_\mathcal{S}(s) = \frac{1}{|W|} \sum_{w \in W} \boldsymbol{h}_w(s)$. For any $\sigma \in \Sigma$, we define

$\mathbf{H}_\sigma \in \mathbb{R}^{|\mathcal{P}|\times|\mathcal{S}|}$ by $\mathbf{H}_\sigma(p,s) = \mathbf{H}(p, \sigma \cdot s)$. Notice that $\mathbf{H}(p, \sigma \cdot s) = \mathbf{H}(p \cdot \sigma, s)$ as $p \cdot (\sigma \cdot s) = (p \cdot \sigma) \cdot s$. If the sub-block Hankel matrix $\mathbf{H}^{(\mathcal{P}^*,\mathcal{S}^*)}$ induced by $\mathcal{P}^* \subseteq \mathcal{P}$ and $\mathcal{S}^* \subseteq \mathcal{S}$ satisfies $\text{rank}(\mathbf{H}^{(\mathcal{P}^*,\mathcal{S}^*)}) = \text{rank}(\mathbf{H}^{(\mathcal{P},\mathcal{S})})$, we say that $(\mathcal{P}^*, \mathcal{S}^*)$ is a *basis* for $\mathbf{H}(W)$. We denote the row of $\mathbf{H}$ induced by a prefix p by $\mathbf{H}(p,:)$, and the column induced by a suffix s by $\mathbf{H}(:,s)$.

Let $G = (\mathcal{P}, \mathcal{S}, E, \mu)$ denote the underlying bipartite graph represented by $\mathbf{H}^{(\mathcal{P},\mathcal{S})}$, where $p \cdot s \in E$ if and only if $\mathbf{H}^{(\mathcal{P},\mathcal{S})}(p,s) \neq 0$ and $\mu : E \to \mathbb{N}$ is the weight function such that $\mu(ps) = \mathbf{H}^{(\mathcal{P},\mathcal{S})}(p,s)$. Since all information in the 4-tuple can be directly derived from W, we also use $G(W)$ to denote such a graph. $d_G(v)$ is the degree of a vertex $v \in V(G)$. Let $N_G(v)$ denote the neighbours of v in G, $|N_G(v)| = d_G(v)$. We also use $N_G(V)$ to denote the union of neighbours of vertices in V. Without specifying, we assume G is the graph by default and use $d(v), N(v), N(V)$ as short forms instead. Supposing V is a set with $v \in V$, we use $V - v$ as a short form to denote $V \setminus \{v\}$.

In the context of an underlying graph, $\mathcal{S}^*(q)$ and $\mathcal{P}^*(t)$ correspond to $N(\mathcal{P}(q))$ and $N(\mathcal{S}(t))$, respectively. Note that, while $\mathcal{S}^*(q) = N(\mathcal{P}(q))$ in graph G, $\mathcal{P}(q)$ is likely not to be equal to $N(\mathcal{S}^*(q))$. Alternatively, if we let $W_p^a = \{w_p^a \in \Sigma^* | a \cdot w_p^a \in W\}$ and $W_s^a = \{w_s^a \in \Sigma^* | w_s^a \cdot a \in W\}$ be two multi-sets of samples induced from the original sample W and let $H = G(W_p^a)$ and $I = G(W_s^a)$ be the bipartite graphs built from these samples, we then have $\mathcal{S}_a^* = N_H(\mathcal{P}_a)$ and $\mathcal{P}_a^* = N_I(\mathcal{S}_a)$. Moreover, $\mathcal{S}_a^*$ is essentially the same as $\mathcal{S}^*(a)$.

2.4 The spectral learning method

We now give a description of the spectral learning algorithm. Given a training set of samples W and a parameter $k \in \mathbb{N}^+$, the spectral algorithm computes a WA A with k states. We refer readers to (Hsu et al., 2012) for the theory behind this algorithm:

1. Select a sub-block $(\mathcal{P}, \mathcal{S})$ of the complete Hankel matrix, denoted by $\mathbf{H}^{(\mathcal{P},\mathcal{S})}$ or $\mathbf{H}$.

2. Compute Hankel matrices for the sub-block selection $(\mathcal{P}, \mathcal{S})$, including $\mathbf{H}$, $\boldsymbol{h}_\mathcal{P}$, $\boldsymbol{h}_\mathcal{S}$ and $\mathbf{H}_\sigma$ for all $\sigma \in \Sigma$.

3. Compute the k-rank factorization of $\mathbf{H}$ through SVD (i.e. $\mathbf{H} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$). Define forward matrix $\mathbf{F} = \mathbf{U}\mathbf{\Sigma} \in \mathbb{R}^{|\mathcal{P}|\times k}$ and backward matrix $\mathbf{B} = \mathbf{V} \in \mathbb{R},^{|\mathcal{S}|\times k}$ then $\mathbf{H} \approx \mathbf{F}\mathbf{B}^\top$ is a k-rank factorization.

4. Build the WA A of k states. The model parameters are computed by:

   (a) Initial vector $\boldsymbol{\alpha}_0^\top = \boldsymbol{h}_\mathcal{S}^\top \mathbf{B}$,

   (b) Final vector $\boldsymbol{\alpha}_\infty^\top = \mathbf{F}^+ \boldsymbol{h}_\mathcal{P}$,

   (c) Transition matrices $\mathbf{A}_\sigma = \mathbf{F}^+ \mathbf{H}_\sigma \mathbf{B}$.

Note that we use $\mathbf{M}^+$ to denote the Moore-Penrose pseudo-inverse of a matrix $\mathbf{M}$. The computation is dominated by SVD, which has a run-time complexity of $O(\min(|\mathcal{P}|, |\mathcal{S}|)^3)$. In Section 4.4.2, we will present a trade off between run-time and the resulting sub-block size in our basis selection algorithm. This, in turn, affects the run-time of the SVD step.

3 Properties of Hankel matrices

We will show some general properties regarding a Hankel matrix with arbitrarily large but finite sets of alphabet and samples. For the following propositions, we denote the prefix and suffix of a complete Hankel matrix over samples W, $\mathbf{H}$, by $\mathcal{P}$ and $\mathcal{S}$ respectively.

Proposition 1. [1] *Let $P^* \subseteq \mathcal{P}$ and $S^* \subseteq \mathcal{S}$ be spanning sets of row and column spaces, respectively. The rank of the sub-block Hankel matrix $\mathbf{H}^{(P^*,S^*)}$ is equal to the rank of $\mathbf{H}^{(\mathcal{P},\mathcal{S})}$.*

As we present in later sections, our algorithm finds a row spanning set for the complete Hankel matrix. To get a column spanning set, simply perform the same algorithm on its transposition. Combining these two sets thus results in a basis of the complete Hankel matrix.

Proof. Suppose $\text{rank}(\mathbf{H}^{(P,*S^*)}) < \text{rank}(\mathbf{H}^{(\mathcal{P},\mathcal{S})})$. Since P^* spans the row space, $\text{rank}(\mathbf{H}^{(P,*\mathcal{S})}) = \text{rank}(\mathbf{H}^{(\mathcal{P},\mathcal{S})})$. There must exist a column induced by a suffix $s \notin S,^*$ $\mathbf{H}^{(P,*\mathcal{S})}(:,s)$, that is not a linear combination of columns of $\mathbf{H}^{(P,*S^*)}$. Since S^* spans the column space, $\mathbf{H}^{(\mathcal{P},\mathcal{S})}(:s)$ is a linear combination of columns of $\mathbf{H}.^{(\mathcal{P},S^*)}$ This means that $\mathbf{H}^{(P,*\mathcal{S})}(:,s)$ must also be a linear combination of columns of $\mathbf{H},^{(P,*S^*)}$ which creates a contradiction. □

The following two propositions point out a crucial insight into reducing the size of the sub-block basis without losing rank.

[1] A stronger statement of this proposition is true for any matrix.

Proposition 2. *Let $U \subseteq \mathcal{P}$ and $V \subseteq \mathcal{S}$ be the sets of vertices in G such that $\forall u \in U, d(u) = 1$ and $(u, \epsilon) \in E$ and $\forall v \in V, d(v) = 1$ and $(\epsilon, v) \in E$. Let u, v be any vertex in U, V, respectively. Then,*

$$\text{rank}(\mathbf{H}^{(\mathcal{P}\setminus(U-u),\mathcal{S}\setminus(V-v)}) = \text{rank}(\mathbf{H}).$$

Proof. We prove this by showing that $\mathcal{P} \setminus (U - u)$ is a row spanning set and $\mathcal{S} \setminus (V - v)$ is a column spanning set. When $|U| = 1$, $U - u = \emptyset$ and $\mathcal{P} \setminus (U - u) = \mathcal{P}$. Consider $|U| > 1$, and let $u' \in U$ be any vertex other than u. By the assumption in the proposition, $\mathbf{H}(u, :) = \mathbf{H}(u', :)$ as $\mathbf{H}(u, \epsilon) = \mathbf{H}(u', \epsilon) = 1$ and $\mathbf{H}(u, w) = \mathbf{H}(u', w) = 0$ for any $w \in \mathcal{S}, w \neq \epsilon$. Thus $\mathcal{P} \setminus (U - u)$ is a row spanning set. A similar argument can be used to prove $\mathcal{S} \setminus (V - v)$ is a column spanning set, which we omit.

By Proposition 1, we can infer that $\text{rank}(\mathbf{H}^{(\mathcal{P}\setminus(U-u),\mathcal{S}\setminus(V-v)}) = \text{rank}(\mathbf{H})$. □

We denote matrix $\mathbf{H}^{(\mathcal{P}\setminus(U-u),\mathcal{S}\setminus(V-v))}$ for any u and v by $\mathbf{H}'$ and let $\mathcal{P}'$ and $\mathcal{S}'$ denote the prefix and suffix set for $\mathbf{H}'$. We do not distinguish such matrices by the choice of u, v since deductions in later sections do not depend on their specific properties.

Proposition 3. *The matrix obtained by removing the row and column induced by ϵ satisfies the following equations:*

$$\text{rank}(\mathbf{H}^{(\mathcal{P},\mathcal{S}-\epsilon)}) + 1 = \text{rank}(\mathbf{H})$$
$$\text{rank}(\mathbf{H}^{(\mathcal{P}-\epsilon,\mathcal{S})}) + 1 = \text{rank}(\mathbf{H}).$$

Also, the following equalities hold:

$$\text{rank}(\mathbf{H}^{(\mathcal{P}'-\epsilon,\mathcal{S}')}) + 1 = \text{rank}(\mathbf{H})$$
$$\text{rank}(\mathbf{H}^{(\mathcal{P}',\mathcal{S}'-\epsilon)}) + 1 = \text{rank}(\mathbf{H}).$$

Proof. By symmetry, we can safely omit the proof for $\mathbf{H}^{(\mathcal{P},\mathcal{S}-\epsilon)}$ and $\mathbf{H}.^{(\mathcal{P}',\mathcal{S}'-\epsilon)}$

We can always find a sequence $w \in W$ that is neither a prefix nor a suffix of any other sequence. Otherwise, W is non-finite, which contradicts our assumption. By Hankel's construction, $w \in \mathcal{P}$ and $w \in \mathcal{S}$ as $w \cdot \epsilon = \epsilon \cdot w = w \in W$. We claim that such a sequence w satisfies the condition for both U and V in Proposition 2. Thus, there is such a non-empty V.

Let $v \in V$ be arbitrary and let $\mathcal{S}' = \mathcal{S} \setminus (V - v)$. By the proof of Proposition 2, we know that $\mathbf{H}(:, v) = \boldsymbol{e}_1$ and $\mathbf{H}'(:, v) = \boldsymbol{e}_1$ (but these two $\boldsymbol{e}_1$'s are distinct). This directly implies that $\mathbf{H}(\epsilon, :)$ is not a linear combination of rows in $\mathbf{H}.^{(\mathcal{P}-\epsilon,\mathcal{S})}$ Hence, removing the row induced by ϵ reduces the rank of H by 1.

Similarly, $\mathbf{H}'(\epsilon, :)$ is not a linear combination of rows in $\mathbf{H}.^{(\mathcal{P}'-\epsilon,\mathcal{S}')}$ We then have $\text{rank}(\mathbf{H}^{(\mathcal{P}'-\epsilon,\mathcal{S}')}) + 1 = \text{rank}(\mathbf{H}') = \text{rank}(\mathbf{H})$ by Proposition 2. □

4 A Divide-And-Conquer Algorithm For Basis Selection

4.1 A High Level Description of the Algorithm

We present a high-level description of the algorithm as well as its intuitions. In the next two subsections, we discuss correctness and limitations in the binary-alphabet case and then extend the results to any finite alphabet.

The algorithm returns a row spanning set for the complete Hankel matrix $\mathbf{H}$ in the form of $P^* \subseteq \mathcal{P}$. By applying the same algorithm to $\mathbf{H},^{\top}$ one can derive a column spanning set, $S^* \subseteq \mathcal{S}$. Recall Proposition 1: such $\mathcal{P}^*$ and $\mathcal{S}^*$ form a basis for the Hankel matrix.

Let us start by defining a subroutine ROW-REDUCE that takes an input matrix and two additional parameters α and γ, returning a full-rank sub-block of the matrix.

Algorithm 1 ROW-REDUCE$(W, P, S, \alpha, \gamma)$

```
if |P| ∈ [α, γ) then
    P' ← ROW-BASIS(H^(P,S)(W))
    S' ← N_G(W)(P')
else
    P' ← P, S' ← S
return (P', S')
```

ROW-BASIS computes the input matrix's row basis, such as by Gaussian Elimination (GE) or QR decomposition on $M,^{\top}$ which has a run-time complexity of $O(n^2 m)$ or $O(nm \cdot \text{rank}(M))$ for M of size $n \times m$. Faster randomized algorithms (Cheung et al., 2013) are known to run in[2] $\tilde{O}(\text{nnz}(M) + k^{\omega})$ to compute a linearly independent row set of size $\min\{\text{rank}(M), k\}$, where $\omega < 2.38$ is the matrix multiplication exponent. One can compute a row set of size exactly $\text{rank}(M)$ by iteratively doubling the size k from $k = 1$. Since this only requires at most $O(\log \text{rank}(M))$ iterations, our ROW-REDUCE algorithm runs in at most

[2] $\tilde{O}(\cdot)$ hides polylogarithmic factors.

$\tilde{O}(\text{nnz}(M) + \text{rank}(M)^\omega)$. Practically speaking, when γ is sufficiently small, there is no real advantage to using a randomized algorithm for ROW-BASIS.

Our parameter α serves as a lower threshold for ROW-REDUCE. When a set P is small, performing ROW-BASIS cannot reduce the overall size by much while still costing run-time. We can choose to bypass such "redundant" computations using α.

We now present the algorithm, ROW-SPAN , for computing a row spanning set for the complete Hankel matrix. The algorithm first divides the current set of samples W by the first symbol of each string and recursively computes spanning prefix sets for each symbol that appears. Since the spanning prefix sets, when combined, might be much larger than the rank of the Hankel matrix of the current sample, our algorithm attempts to reduce the size of the combined set by performing ROW-REDUCE on it.

When the combined set is large, it becomes computationally expensive to perform Row-Reduce. Our parameter γ then allows us to choose to bypass such computations at the expense of having larger sub-blocks. Moreover, if each recursively computed prefix subset is large, then this is intuitively a good indication that there is limited promise for the algorithm to further reduce the size of the combined set. As such, we additionally have the lower-bound condition in line 7. A formal analysis of the runtime is presented in subsection 4.4.1.

As mentioned above, we apply this algorithm to both H and $H^\top$ to get the row and column spanning sets, $P^* \subseteq \mathcal{P}$ and $S^* \subseteq \mathcal{S}$, which then form a basis for Hankel matrix $\mathbf{H}(W)$. To compute the column spanning set, one can simply call ROW-SPAN$(\Sigma, W^\top, \mathcal{S}^\top, \mathcal{P}^\top, \alpha, \gamma)$. Reversing the sequences in the first set of the returned tuple, one can obtain the column spanning set $S.^*$ ROW-SPAN is recursive. We choose to present it in a top-down, divide-and-conquer fashion to facilitate its inductive analysis and our later discussion. In fact, a slightly faster, recursion-free, bottom-up approach exists.

The parameters α and γ are related to the estimation of the rank of the complete Hankel matrix $\mathbf{H}(W)$. When γ is sufficiently large, say $\gamma > |\Sigma| \cdot \text{rank}(W)$ and $\alpha \leq \text{rank}(W)$, the resulting basis of ROW-SPAN is a minimal basis where both the row spanning set and column spanning set are linearly independent.

Algorithm 2 ROW-SPAN$(\Sigma, W, \mathcal{P}, \mathcal{S}, \alpha, \gamma)$

1: $P \leftarrow \emptyset, S \leftarrow \emptyset, U \leftarrow \emptyset\ L \leftarrow 0$
2: **for** $\sigma \in \Sigma$ with $\mathcal{P}_\sigma \neq \emptyset$ **do**
3: $\quad (P^*_\sigma, S^*_\sigma, u, L_\sigma) \leftarrow$ ROW-SPAN$(\Sigma,\ W^\sigma_p, \mathcal{P}_\sigma, \mathcal{S}^*_\sigma, \alpha, \gamma)$
4: $\quad P \leftarrow P \cup \{p | \exists p_\sigma \in P^*_\sigma, p = \sigma \cdot p_\sigma\} - (\sigma \cdot u)$
5: $\quad S \leftarrow S \cup S^*_\sigma, U \leftarrow U \cup \{\sigma \cdot u\}$
6: $\quad L \leftarrow L_\sigma$ if $L < L_\sigma$
7: **if** $\gamma > |P| \geq 2L$ **then**
8: $\quad (P, S) \leftarrow$ ROW-REDUCE$(W, P, S, \alpha, \gamma)$
9: $\quad L \leftarrow \max(|P|, 2L)$
10: **if** $|U| = 0$ **then**
11: $\quad u \leftarrow \epsilon$
12: **else**
13: $\quad$ Pick an arbitrary $u \in U$
14: $\quad P \leftarrow P \cup \{u\}, S \leftarrow S \cup \{\epsilon\}$
15: $P^* \leftarrow P \cup \{\epsilon\}, S^* \leftarrow S \cup \mathcal{S}^*(\epsilon)$
16: **return** (P^*, S^*, u, L)

4.2 Simple Case with an Alphabet of 2 symbols

To motivate the analysis of the algorithm, it will be illustrative to consider a simple alphabet with 2 symbols, $\Sigma = \{a, b\}$, before generalizing the results to arbitrary finite alphabets. We hold off the proof of correctness and the generalized results until subsection 4.3.

We start with the division scheme. Recall from the definition of $\mathcal{P}(q)$ in subsection 2.1, the intersection $\mathcal{P}(a) \cap \mathcal{P}(b) = \emptyset$. Notice that $\mathcal{P}_\sigma$ is simply $\mathcal{P}(\sigma)$ with the first symbol trimmed off of all sequences in the set. Recall also that $\epsilon \in \mathcal{P}_a$ and $\epsilon \in \mathcal{P}_b$ by definition. We then have $\mathcal{P}_a \cap \mathcal{P}_b = \epsilon$. However, the ϵ's in $\mathcal{P}_a$ and $\mathcal{P}_b$ are distinct in that the ϵ in $\mathcal{P}_a$ corresponds to the prefix a in $\mathcal{P}$ while the ϵ in $\mathcal{P}_b$ instead corresponds to b. Hence, the algorithm does not repeatedly consider the same prefix.

Before we move on to the combining steps of the algorithm, there are a couple of conditions we need to establish to outline our inductive proof of correctness.

1. After line 6, $P \cap U = \emptyset$. This is straightforward to see, as we intentionally remove the elements in U from P at line 4.

2. At line 10, $|U| = 0$ iff all $\mathcal{P}_\sigma$ are empty. Within each iteration of the for-loop, we add a unique element to U. As such, we can only

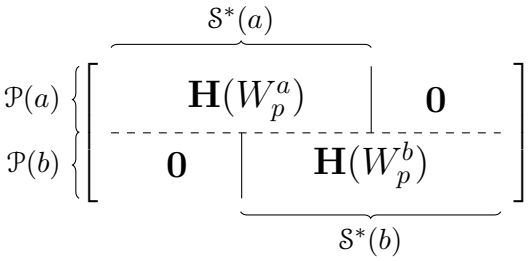


Figure 1: A visual representation of $\mathbf{H}^{(\mathcal{P}-\epsilon,\mathcal{S}-\epsilon)}$ with alphabet of 2 symbols.

end up with $|U| = 0$ when the if-condition is not satisfied for all iterations of the loop.

3. After line 9, the pair (P, S) satisfies $\text{rank}(\mathbf{H}^{(P,S)}) + 1 \leq \text{rank}(\mathbf{H}^{(P^*,S^*)})$, where P^* and S^* are the returned prefix and suffix set. Given fixed u, the matrix $\mathbf{H}^{(P-\epsilon,S)}$ is a sub-block of the matrix $\mathbf{H},^{(P^*,S^*)}$ which means that the rank of the latter must be at least as large as the former. We can then obtain the desired inequality by Proposition 3.

We now consider the combining steps (the algorithm except line 3). In the case of a 2-symbol alphabet, consider first the set $\mathcal{P}(a)$. Recall that $\mathcal{S}^*(a) = N_G(\mathcal{P}(a))$. Then each row and each column in the sub-block matrix $\mathbf{H}^{(\mathcal{P}(a),\mathcal{S}^*(a))}$ is non-empty (i.e., there is at least one non-zero entry). In contrast, the sub-block matrix $\mathbf{H}^{(\mathcal{P}(a),\mathcal{S}\setminus\mathcal{S}^*(a))} = \mathbf{0}$. To see this, consider the underlying bipartite graph G. Any $v \in \mathcal{S}$ that is not in $N(\mathcal{P}(a))$ does not have an edge connecting to the vertex set $\mathcal{P}(a)$. Similarly, $\mathbf{H}^{(\mathcal{P}(b),\mathcal{S}\setminus\mathcal{S}^*(b))} = \mathbf{0}$. When $\mathcal{S}^*(a)$ and $\mathcal{S}^*(b)$ have no intersection, then simply combining a basis for $\mathbf{H}(W_p^a)$ and one for $\mathbf{H}(W_p^b)$ with the ϵ row and column of $\mathbf{H}(W)$ yields a full numerical rank sub-block. In fact, the resulting basis is an optimal solution if the condition holds for all recursive calls. However, this is not always the case. When $\mathcal{S}^*(a) \cap \mathcal{S}^*(b) \neq \emptyset$, there is no guarantee that combining ϵ and the two column spanning sets of $\mathbf{H}(W_p^a)$ and $\mathbf{H}(W_p^b)$ would yield a column spanning set for $\mathbf{H}(W)$. This is the reason for keeping at least all the non-trivial columns. In contrast, combining the rows still results in a row spanning set, since $\mathcal{P}(a) \cap \mathcal{P}(b)$ is always an empty set. We give a visual representation of the matrix in Figure 1.

4.3 General Case

In the general case, we consider a finite alphabet with $|\Sigma| = n \in \mathbb{N}^+$ symbols. Many results in subsection 4.2 can be generalized to an arbitrarily larger finite alphabet. In this subsection, we present an inductive proof of the correctness of this algorithm.

Our goal is to prove that the algorithm returns a row spanning set $P.^*$ We shall consider a stronger statement that implies this:

Claim 1. ROW-SPAN returns a tuple (P^*, S^*, u) s.t. P^* is a row spanning set of the input Hankel matrix $\mathbf{H}(W)$, for any $s \in N_G(P^*)$, $s \in S^*$ and where $u \in P^*$ is the only prefix that satisfies the condition in Proposition 2.

We perform our induction on the maximum length of a sequence T in the sample W. When $T = 0$, Condition 2 always holds.[3]Hence, by line 15, we explicitly added the ϵ row and column back in and the resulting P^* and S^* are exactly $\{\epsilon\}$ and $u = \epsilon$. As for the inductive case, assume Claim 1 holds for $T \leq t$. We want to prove it for $T = t + 1$. For any $\sigma \in \Sigma$, the longest sequence in W_p^σ must be at most t. Hence, in the recursive step, the returned tuple $(P_\sigma^*, S_\sigma^*, u_\sigma)$ satisfies the correctness condition by the inductive hypothesis. As a result, after line 6, the set P is a row spanning set for $\mathbf{H},^{(\mathcal{P}-\epsilon,\mathcal{S}-\epsilon)}$ U satisfies the condition in Proposition 2 and no prefix P has the same property, by Condition 1. Moreover, S has all neighbours of both P and U as in graph $G(W)$. By Condition 2, we know that U must be non-empty and that the else-block in lines 12–14 must be executed. This then leads us to Condition 3, which is $\text{rank}(\mathbf{H}^{(P,S)}) + 1 \leq \text{rank}(\mathbf{H}^{(P^*,S^*)})$. By adding the ϵ row back in, as in line 15, we have $\text{rank}(\mathbf{H}^{(\mathcal{P},\mathcal{S})}) = \text{rank}(\mathbf{H}^{(P^*,S^*)})$, which also implies that P^* is a spanning set. The condition $N_G(P^*) \subseteq S^*$ naturally holds, since each assignment to S' and S^* guarantees it. Notice also we added u to P' at line 14, which consequently proves the condition between u and P^* in the claim. This concludes the proof.

We remark that, unlike the random-cut algorithm (Balle et al., 2012) or the bipartite matching algorithm (Quattoni et al., 2017), our proof suggests that this algorithm does not require the length of sequences (length of support) in the sample W to be uniform, which provides more flexibility in choosing training data.

[3]We assume this exists and $\mathbf{H}(\epsilon, \epsilon) \neq 0$ in this case.

4.4 Run-time Complexity

In this subsection, we consider the run-time complexity of the algorithm with a fixed moment size of T. That is, the length of each sequence is at most T symbols long. We set $n = |\mathcal{P}|$ and $m = |\mathcal{S}|$. Then, the complete Hankel matrix $\mathbf{H}$ is an $n \times m$ matrix. We also let $w = |W|$ denote the total number of words in the multi-set of samples W.

Our theoretical runtime complexity is presented in Theorem 1. We remark that our algorithm is more efficient when the stochastic language follows Heaps' Law. By Heaps' Law, the size of the vocabulary (i.e., the number of unique words) $V = Kw^{\beta} = \Theta(w^{\beta})$, where K is typically between 10 and 100 and β, between 0.4 and 0.6 for English corpora. It is easy to see that both n and m are bounded above by $(T+1)V$ since each unique word can spawn at most T unique prefixes and T unique suffixes, with one addition of the null prefix and suffix ϵ. Moreover, by the same argument, the number of non-zero entries $\text{nnz}(\mathbf{H}) \leq (T+1)V$.

4.4.1 Run-time Complexity of Basis Selection algorithm

Given a fixed γ parameter, the total running time of the algorithm is maximal when $\alpha = 1$. For the rest of this subsection, we only evaluate the running time of the algorithm with $\alpha = 1$ and we represent the complexity in terms of T, w, γ.

Theorem 1. *The run-time complexity of the algorithm* ROW-SPAN *is* $\tilde{O}(TV + n\gamma^{(\omega-1)}) = \tilde{O}(Tw^{\beta} + n\gamma^{(\omega-1)})$*, or* $O(nm\gamma \log \gamma)$

We separate the running time of the algorithm into two parts. The first part is the total cost of all recursive calls other than ROW-REDUCE. The second part is the total cost of the ROW-REDUCE algorithm.

The running time of all operations other than ROW-REDUCE is dominated by the set-union operations. As the complete prefix and suffix sets $\mathcal{P}$ and $\mathcal{S}$ are known and fixed, we can safely assume a table representation. In that case, the set union operation is linear in the size of the smaller set to be unioned. That is, in each recursion the additional cost to calling ROW-REDUCE is $O(|\mathcal{S}^*(\epsilon)|)$. Consider the corresponding cells of $\mathcal{S}^*(\epsilon)$ in the complete Hankel matrix $\mathbf{H}$. By the recursion, there is no overlap between the corresponding cells of two different calls. Then, $\sum |\mathcal{S}^*(\epsilon)| \leq nm$ or $\sum |\mathcal{S}^*(\epsilon)| \leq TV$. Therefore, the running time of the first part is $O(nm)$ or $O(TV)$.

Now, consider the total cost of ROW-REDUCE.

Lemma 1. *Suppose given valid input,* ROW-SPAN *returns a prefix subset* P^* *and a value* L*. Then* $L < \gamma$ *and the total cost of* ROW-REDUCE *throughout its entire operation is at most* $\tilde{O}(TV + nL^{\omega-1}) \leq \tilde{O}(TV + n\gamma^{\omega-1})$ *or* $O(nmL \log L) \leq O(nm\gamma \log \gamma)$.

Proof. By our recursive algorithm, we can define a unique tree, where each node, except the root, can be denoted by a unique prefix $p \in \mathcal{P}$. For a node with prefix p, there is a call of ROW-SPAN with input $W_p = \{w \in \Sigma^* | p \cdot w \in W\}$ where the condition in line 7 is satisfied and $|P| \leq \gamma$, i.e., it performs a non-trivial case of ROW-REDUCE. For two nodes with prefixes p and $p,'$ p' is a child of p, p is a prefix of p' and no other node p'' is both a prefix of p' and suffix of p. We assign three attributes to each node p: the total number of non-zero entries $U_p = \text{nnz}(H^{(P,S)}(W_p))$ and $|P_p|$, the size $|P|$ after line 6, and the final value L_p. The one-time cost of ROW-REDUCE for node p is then at most $\tilde{O}(U_p + L_p^{\omega})$, or $O(|P|mL_p)$ if using GE or QR. For the root, we denote it by the trivial string ϵ and let size $U_p = \text{nnz}(\mathbf{H}) = O(TV)$, $|P_\epsilon| = |\mathcal{P}| = n$ and $L_\epsilon = L$ be the final returned L.

Let N_h be all nodes of height h. The root has height 0. For any distinct pair of nodes $x, y \in P_h$, neither can be a prefix of the other. Then, we observe:

$$\sum_{p \in N_h} U_p \leq \text{nnz}(\mathbf{H}) \leq TV, \tag{2}$$

$$\sum_{p \in N_h} |P_p| \leq n, \tag{3}$$

for all valid heights h. Notice that $L = \max_{p \in N_1} L_p$. By line 9, we obtain a third observation:

$$L_p \leq \frac{1}{2^{h-1}} L \quad \forall p \in N_h. \tag{4}$$

$L_p < \gamma$, and the total height $H \leq \log_2 L + 2$. By the convexity of the function x^{ω} on $[0, \infty)$ for any $\omega \geq 2$, the condition in line 7 and (3) further yield:

$$\sum_{p \in N_h} L_p^{\omega} \leq \frac{2^{h-1} n}{L} \cdot \frac{1}{2^{\omega(h-1)}} L^{\omega}$$

$$\leq \frac{n}{2^{(\omega-1)(h-1)}} L.^{\omega-1}$$

For a randomized algorithm, by summing over the entire tree using (2), (4) and above, we reckon a

total cost of $\tilde{O}(TV + nL^{\omega-1})$ where we recall that $\tilde{O}(\cdot)$ hides polylogarithmic factors. If we use GE or QR instead, by (3) and (4), we get a total cost of $O(nmL \log L) \leq O(nm\gamma \log \gamma)$. □

Theorem 1 follows by combining our run-time analysis for both parts. We remark that the run-time guarantees using randomized algorithms from (Cheung et al., 2013) are significantly better asymptotically than the ones using Gaussian Elimination. Note that $TV \leq O(nm)$ and we should always choose $\gamma \leq \min(n, m)$. This run-time difference is most apparent when the complete Hankel matrix $\mathbf{H}(W)$ is sparse, i.e., $TV = \tilde{O}(n + m)$.

4.4.2 Run-time Complexity in respect of SVD

There is a trade off between run time and the resulting sub-block size in our basis selection algorithm, controlled by the parameters α and γ. As mentioned before, the computational complexity is dominated by SVD, which has a run-time complexity of $O(\min(|P|, |S|)^3)$.

Since this is a divide-and-conquer algorithm, there are many recursive calls where the subsample is small, leading to small row spanning sets. Even if we do not perform ROW-BASIS on rows, the resulting row spanning set size is still reasonably small. This is the fundamental reason for introducing α. Generally speaking, α should be around $\gamma/|\Sigma|$ to trigger the ROW-BASIS subroutine. Consider when α is larger than $\gamma/|\Sigma| + 1$, and suppose the returned sub-block matrices in the recursive call all have a prefix set of size $\alpha - 1$. In the recursive call, ROW-BASIS is not performed, as $\alpha - 1 < \alpha$. Moreover, in the current call, ROW-BASIS is not performed either since the total size of prefixes is greater than γ.

According to our need, we can choose a larger γ, which leads to a more compact basis but with longer run-time, and *vice versa*. This trade-off is not linear, as we demonstrate in our experimental evaluations in section 5.

5 Experimental Evaluation

In this section, we evaluate our basis selection algorithm ROW-SPAN , refereed to as RS. The inner algorithm ROW-REDUCE that combines rows and performs matrix rank algorithm ROW-BASIS is referred to as RR.

Implementation Details Our algorithm is implemented in Python.We consider two choices of matrix rank algorithm, GE and QR, for the ROW-BASIS algorithm in RR.

For GE, we implemented an algorithm that returns a reduced row echelon form (RREF) of a matrix with partial pivoting. To reduce redundancy in linear algebraic operations, our implementation additionally returns the RREF of the Hankel sub-block $\mathbf{H}^{(P^*,S^*)}$. Note that the RREF of a matrix $\mathbf{M}$ has the same row span as $\mathbf{M}$.

Our QR-decomposition-based RR algorithm uses a sparse QR implementation from SuiteSparse.[4] Since a matrix $\mathbf{M}$ passed to RR typically has a much larger number of columns than rows, our algorithm performs an additional QR decomposition first on $\mathbf{M}$ to obtain a column spanning set C. Note that the worst-case runtime of QR on $\mathbf{M}$ depends quadratically on the number of rows, but only linearly on the columns. Our algorithm then proceeds to perform a QR on $(\mathbf{M}^{(:,C)})^\top$ to retrieve a row spanning set.

Note that we discarded the lowerbound check of $|P| \geq 2L$ in line 7 of RS from our implementation. This lowerbound check is only included for proof-theoretic reasons and has little practical benefit.

Setup and Datasets All experiments are performed on a Linux machine using an Intel® Core™ i7-9700KF CPU @ 3.60GHz and 12G RAM and 32G SWAP. All computations are performed on a CPU. We measured the processing time using the Python built-in function `time.process_time()`.

For our evaluations, we considered two datasets: the English Penn Treebank dataset (Marcus et al., 1993) with universal part-of-speech tags (12 symbols), and the War and Peace dataset (Karpathy et al., 2016) for a character-based model (104 symbols and a total of ~ 2.58m non-space characters). Datasets were cleaned and prepared using tools from SPiCe.[5] Hankel matrices are computed using the `scikit-splearn` package.[6]

Evaluations For our evaluations, we draw comparisons between our proposed algorithm RS using QR or GE and a maximal-matching-based algorithm, referred to as MM, from (Quattoni et al., 2017). Since no implementation of the MM algorithm was available, all experiments are performed using our own re-implementation of the algorithm

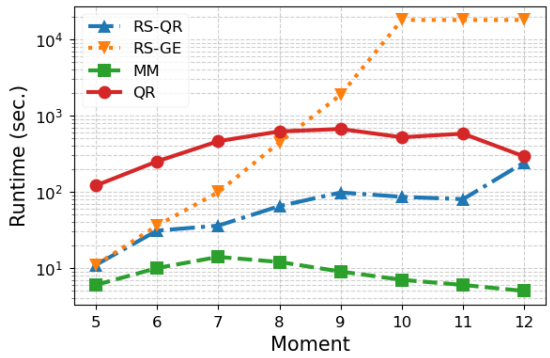
[4] SuiteSparse https://people.engr.tamu.edu/davis/suitesparse.html. Python wrapper for sparse QR from https://github.com/yig/PySPQR.

[5] https://spice.lis-lab.fr/index.php

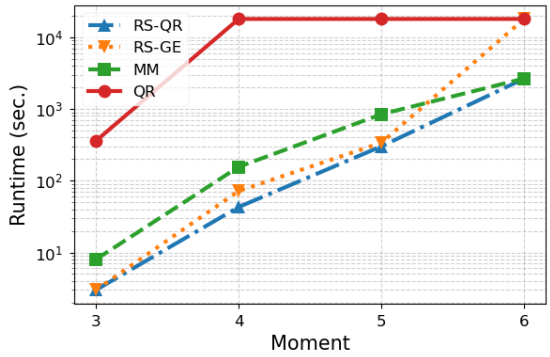
[6] https://pypi.org/project/scikit-splearn/

in Python. A plain sparse QR decomposition algorithm, referred to simply as QR, is used as a baseline. Note that QR always returns optimal-sized bases (prefix and suffix sets of size $\mathrm{rank}(\mathbf{H})$). A timeout limit of 5 hours (180k seconds) is set for all experiments.

Runtime comparison to MM. We compared the runtime of our basis-selection algorithm RS against MM. We used the Penn Treebank and War and Peace datasets with various choices of moments. For these experiments, the parameters are fixed[7] to ensure that RR always returns a full numerical rank basis of optimal size.



(a)



(b)

Figure 2: Runtime comparison for varying moments. (a) Penn Treebank, (b) War and Peace.

On the Penn Treebank corpus, RS is comparable to MM for smaller moments, but significantly less efficient for larger moments ($T \geq 8$), see Figure 2 (a). The runtime of both the QR- and GE-based RS algorithms increases as the moment grows larger. This is as expected, since the dimension of the complete empirical Hankel matrix increases ($\sim 13k \times 13k$ for $T = 5$, $\sim 55k \times 55k$ for $T = 10$). We observe that the increase in runtime scales sub-linearly to the size of $\mathbf{H}$. From $T = 5$ to 10, the runtime increases by a factor of less than 7, from 11s to 73s, while the size of $\mathbf{H}$ increases by a factor of over 17. This suggests that RS is able to exploit the structure and sparsity of Hankel matrices.

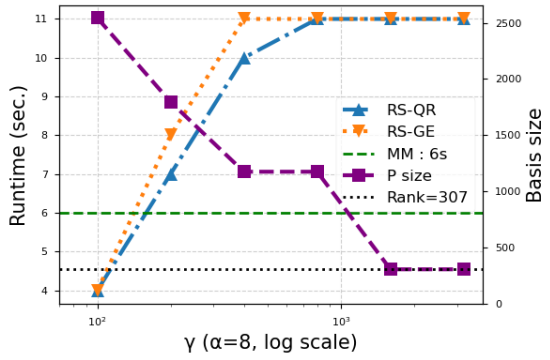
The runtime of MM is less than 16s for all tested moments. We remark that the number of nonzero entries (nnz) of the Hankel matrix remains roughly the same from $T = 8$ to 10, at between $99k$ and $110k$. The observed runtime of MM is also as expected, since the theoretical runtime of the MM algorithm scales linearly to the nnz of $\mathbf{H}$.

Note that the timeouts that accrued to RS-GE for moments $T = 10$ to 12 are due to memory limitations. The runtime of RS-GE is also consistently worse than RS-QR. We believe such inefficiency is due to our sub-optimal implementation using dense matrix operations for GE. While it is possible to utilize sparse matrices in Python, no sparse matrix implementations are particularly suitable to GE due to excessive row and column slicing and modifications to the sparsity structure. We are optimistic that a tailor-made sparse matrix implementation can significantly improve our runtime: the returned RREF of sub-block matrices is always sparse, with less than 4 nonzero entries per column on average when computing row-spanning sets. This implementation is beyond the scope of this paper.

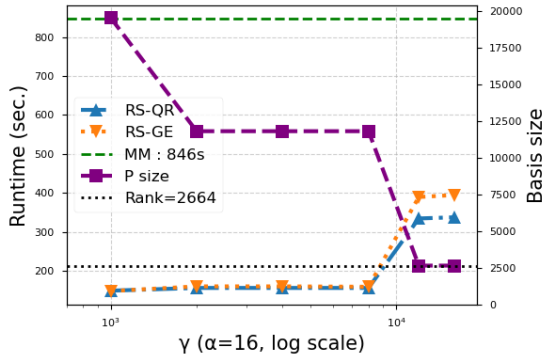
Contrary to the above, for the War and Peace corpus, both RS-QR and RS-GE consistently outperformed MM, see Figure 2 (b). We stopped at $T = 6$ since both RS-QR and RS-GE timeout at $T = 7$ due to memory limitations. Hankel matrices for this set of experiments are considerably larger: $\mathbf{H}$ is $\sim 58k \times 58k$ in size with $\sim 225k$ nnz even at $T = 4$, and the largest matrix is $\sim 304k \times 304k$ with over $1.1m$ nnz. These matrices have ranks that are considerably smaller relative to their size, with a rank-to-rows ratio below 0.035, compared to 0.09–0.37 for the Penn Treebank dataset when moment $T \geq 8$. A relatively large rank-to-rows ratio suggests that the number of samples in the Penn Treebank dataset may be insufficient. We expect that, for sufficiently large datasets, our method would achieve better runtime performance, albeit at the cost of increased memory usage.

Finally, we remark that our algorithm RS is readily parallelizable. Our divide-and-conquer scheme is well-suited to parallel computing. The linear-algebraic operations are efficient to perform in parallel as well. The sparse QR implementation from SuiteSparse supports multi-thread computing. The elapsed times of RS-QR are typically less than 40%

[7] It suffices to choose $\gamma = |\mathcal{P}|$ and $\alpha = 1$.



(a)



(b)

Figure 3: Runtime and Basis size trade-off in γ (a) Penn Treebank ($T = 5$), (b) War and Peace ($T = 5$).

of the reported processing time.

Impact of α and γ. We study the impact of the parameters α and γ to the runtime of RS and the sizes of the bases produced. For fixed γ, we experimented with varying the value of α ($\alpha < 0.01\gamma$ for both corpora) in a range that does not affect the resulting basis size. Under such conditions, α only has a marginal effect on the runtime — less than 10% of the optimal runtime.

The parameter γ, on the other hand, has a significant impact and controls the trade-off between runtime and basis size, demonstrated by Figure 3. Increasing γ generally increases the runtime of RS while reducing the size of the basis. When γ is close to but smaller than the threshold that produces an optimal-sized basis, the runtime of RS is significantly shorter (50% to 0.1%) compared to the case where γ is above the threshold. The size of the spanning prefix/suffix set produced in this case is often within a factor of 3 times the optimal size (i.e., $\mathrm{rank}(\mathbf{H})$). For a large Hankel matrix with relatively small rank, one could prefer a smaller γ to leverage runtime reduction.

We note that the impact of γ to both runtime and basis size is not gradual, especially for larger alphabets. This is expected from our algorithmic structure: the combined prefix set passed to RR can be as large as $|\Sigma|$ times the largest prefix set returned from RS recursion. One can smoothe the effect of γ by partitioning the alphabet into a few groups and performing multiple RR steps for each group before executing RR on the entire alphabet.

6 Conclusions

We presented a simple divide-and-conquer algorithm for choosing a full numerical rank basis for Hankel matrices with good theoretical running time guarantees and a practical trade-off between running time and the resulting sub-block size.

References

Raphaël Bailly, François Denis, and Liva Ralaivola. 2009. Grammatical inference as a principal component analysis problem. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 33–40, New York, NY, USA. ACM.

Borja Balle, Ariadna Quattoni, and Xavier Carreras. 2011. A spectral learning algorithm for finite state transducers. In *Machine Learning and Knowledge Discovery in Databases*, pages 156–171, Berlin, Heidelberg. Springer Berlin Heidelberg.

Borja Balle, Ariadna Quattoni, and Xavier Carreras. 2012. Local loss optimization in operator models: A new insight into spectral learning. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pages 1819–1826, USA. Omnipress.

Ho Yee Cheung, Tsz Chiu Kwok, and Lap Chi Lau. 2013. Fast matrix rank algorithms and applications. *J. ACM*, 60(5):31:1–31:25.

Shay B. Cohen, Karl Stratos, Michael Collins, Dean P. Foster, and Lyle Ungar. 2012. Spectral learning of latent-variable PCFGs. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 223–231, Jeju Island, Korea. Association for Computational Linguistics.

François Denis, Mattias Gybels, and Amaury Habrard. 2016. Dimension-free concentration bounds on hankel matrices for spectral learning. *J. Mach. Learn. Res.*, 17(1):1071–1102.

Daniel Hsu, Sham M. Kakade, and Tong Zhang. 2012. A spectral algorithm for learning hidden Markov models. *Journal of Computer and System Sciences*, 78(5):1460 – 1480. JCSS Special Issue: Cloud Computing 2011.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. 2016. Visualizing and understanding recurrent networks. *In ICLR Workshop Track*.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: the Penn treebank. *Computational Linguistics*, 19(2):313–330.

Ariadna Quattoni and Xavier Carreras. 2019. Interpolated spectral NGram language models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5926–5930, Florence, Italy. Association for Computational Linguistics.

Ariadna Quattoni, Xavier Carreras, and Matthias Gallé. 2017. A maximum matching algorithm for basis selection in spectral learning. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1477–1485. PMLR.

Sajid Siddiqi, Byron Boots, and Geoffrey Gordon. 2010. Reduced-rank hidden Markov models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 741–748, Chia Laguna Resort, Sardinia, Italy. PMLR.