IFT 6760A - Lecture 7
Spectral Learning of Weighted Automata / HMMs

Scribe(s): Mahmoud Nassif, Bhairav Mehta, Mostafa Abdelnaim **Instructor:** Guillaume Rabusseau

1 Summary

In the previous lecture we went over PCA, CCA and wrapped up with a discussion about Locally Linear Embedding (LLE).

In this lecture we will introduce HMMs and Weighted Automata (WA). This will lead us to a deep dive into how they work and how they relate to one another. We will then conclude with the spectral learning algorithm.

2 Intuition

Let's start off with a general form of the function that we are trying to approximate with a WA:

$$\overbrace{x_1, x_2, ..., x_k}^{\in \mathcal{X}:sequences} \mapsto f(x_1, ..., x_k)$$

f is a function that maps sequences of elements from an input space $\mathcal{X}$ to some output space $\mathcal{Y}$. We will only focus on the case where $\mathcal{X}$ is a discrete alphabet and $\mathcal{Y} = \mathbb{R}$. This has various practical usages. (i.e. it can represent the probability that a sequence of words might form a meaningful sentence, it can represent the confidence of a robot about its location given a sequence of probing data, etc.).

To state the problem formally, we would also need to define a few other elements. Let Σ be a finite alphabet ($\Sigma = \{a, b\}$) and we let Σ^* be the set of all sequences built on Σ (e.g. $\Sigma^* = \{\lambda, a, ab, aa, baab, ...\}$) where λ represents the empty word.

Ultimately, we want to learn a function $f : \Sigma^* \to \mathbb{R}$.

3 [Quick Review] HMMs

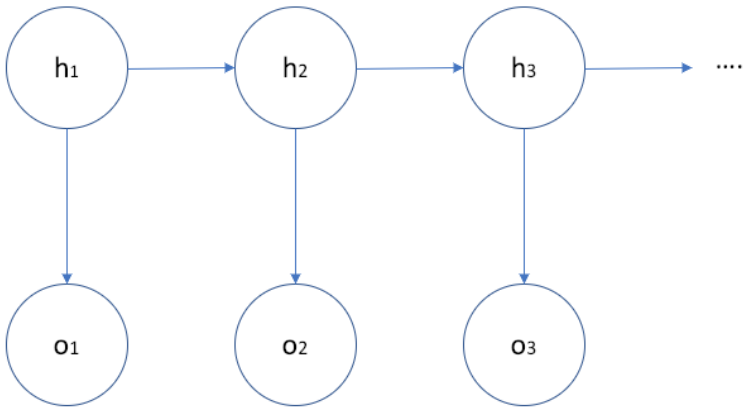

Figure 1: graphical model representing a HMM

Hidden Markov model or HMM is a statistical Markov model with hidden states. In other words, it is a system that has a stochastic internal state that randomly changes. State transitions in an HMM are *Markovian*, which means that their state transitions only depend on the current state they are in, rather than depending on the current *and* past states.

For example, imagine a robot in a maze with a set of definite locations represented as squares on the floor. Let's assume that the robot has an internal state $h(t)$, which denotes its true position in the world. These internal hidden states are analogous to the ones described above, but the robot gets only observations $o(t)$, from which it has to deduce what the underlying $h(t)$ is. Assume that its current belief is that it is in an alley, which happens to be in accordance with the scanning data of its laser sensor $o(t)$ that senses open space in opposing direction but walls on the orthogonal axis. At $t+1$, the robot moves to another square, the observation changes $o(t+1)$ and now the laser sensor identifies open space in 2 orthogonal directions and walls in the other directions. With only the current observation and the previous state, if well calibrated, the robot is more confidence that its current state $h(t+1) \rightarrow \phi(h(t), o(t+1))$ is that it has reached the curb at the end of the alley. Depending on how many of these curbs exist in the maze, the robot will set a higher probability of being in those locations rather than anywhere else in the maze. The more observations it has, the higher the confidence about its location will be as it tries to retrofit the sequence of observations to the sequence of possible hidden states (locations) that can generate those observations.

Since the observations o_t can be noisy, we want to estimate the true hidden states h_t from the observations, which in the robot's case, can help us localize the robot.

4 Automata

Before we describe how weighted automata and HMMs are related, we will introduce some notation that will formally define the weighted automata model.

Definition 1 (Weighted Automata)**.** *A weighted automata (WA) with m states over Σ (finite alphabet), is a tuple*

$$\mathcal{A} = (\boldsymbol{\alpha}_0, \{\mathbf{A}^\sigma\}_{\sigma\in\Sigma}, \boldsymbol{\alpha}_\infty)$$

where:

- $\boldsymbol{\alpha}_0 \in \mathbb{R}^m$ *is the initial weight vector*
- $\boldsymbol{\alpha}_\infty \in \mathbb{R}^m$ *is the final weight vector*
- $\mathbf{A}^\sigma \in \mathbb{R}^{m\times m}$ *is the transition matrix for each $\sigma \in \Sigma$*

The weighted automata (WA) $\mathcal{A}$ computes a function

$$f_\mathcal{A} : \Sigma^* \to \mathbb{R}$$
$$\underbrace{x_1x_2...x_k}_{x\in\Sigma^*} \to \boldsymbol{\alpha}_0^\top \underbrace{\mathbf{A}^{x_1}\mathbf{A}^{x_2}...\mathbf{A}^{x_k}}_{\mathbf{A}^x}\boldsymbol{\alpha}_\infty$$

It is important to note that the number of states m does not necessarily have to equal the cardinality of the alphabet Σ. For those familiar with recurrent networks in deep learning, $\boldsymbol{\alpha}_0$ can be though of as the *initial* hidden state, $\boldsymbol{\alpha}_\infty$ as the output weights, and the $\mathbf{A}^\sigma$ as the recurrent weights.

Example

In this example we will implement a WA that is capable of counting the number of letters 'a' in a sentence fed as an input. Our WA requirements will have two states: a state that keeps track of the increment and a state that keeps track of the number of times the letter 'a' was encountered. Let's call them $h_1^{(t)}$ and $h_2^{(t)}$ respectively. In the event that 'a' occurred, the new states $h_1^{(t+1)}$ and $h_2^{(t+1)}$ can be determined by the previous states $h_1^{(t)}$ and $h_2^{(t)}$.

Let's build a matrix that can generate $\mathbf{h}^{(t+1)}$ from $\mathbf{h}^{(t)}$:

$$(\mathbf{h}^{(t+1)})^\top = (\mathbf{h}^{(t)})^\top\mathbf{A}^a \quad \text{where} \quad \mathbf{A}^a = \begin{pmatrix}1 & 1\\ 0 & 1\end{pmatrix} \quad \text{is the transition matrix of } a$$

With our transition matrix, we can define all the components of the WA of interest:

$$\Sigma = \{a, b\} \quad \boldsymbol{\alpha}_0 = \begin{pmatrix}1\\0\end{pmatrix} \quad \boldsymbol{\alpha}_\infty = \begin{pmatrix}0\\1\end{pmatrix} \quad \mathbf{A}^a = \begin{pmatrix}1 & 1\\ 0 & 1\end{pmatrix} \quad \mathbf{A}^b = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$$
$$\begin{aligned} f(aaba) =& \boldsymbol{\alpha}_0^\top\mathbf{A}^a\mathbf{A}^a\mathbf{A}^b\mathbf{A}^a\boldsymbol{\alpha}_\infty \\ =& \boldsymbol{\alpha}_0^\top(\mathbf{A}^a)^3\boldsymbol{\alpha}_\infty = 3 \end{aligned}$$

Notice how the power of the transition matrix of a can be simplified:

$$(\mathbf{A}^a)^m = \begin{pmatrix}1 & m\\ 0 & 1\end{pmatrix}$$

We will now show the benefits of rewriting an HMM as a Weighted Automata. Recall, the formal definition of an HMM:

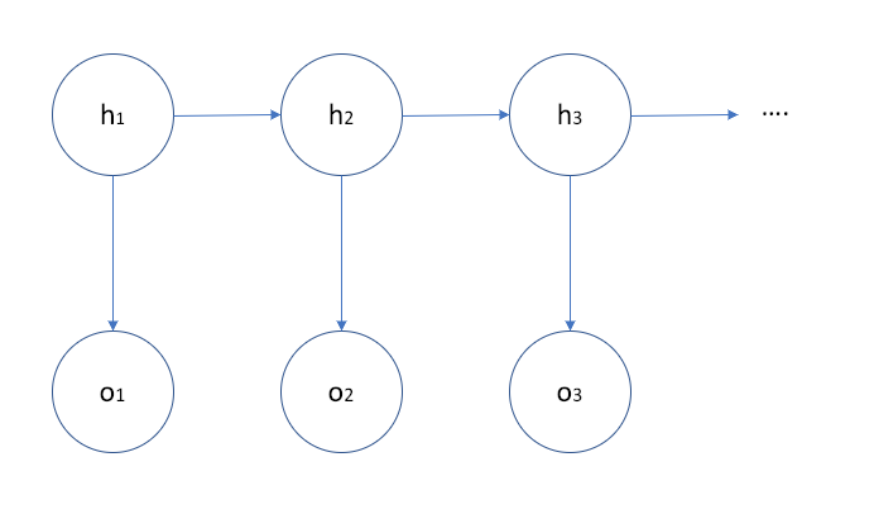

Figure 2: graphical model representing a HMM

Definition 2 (HMM). *Given a set of states $S = \{1, ..., n\}$ and a set of observations $\Sigma = \{1, ..., p\}$, a HMM is given by*

- *transition probabilities $\mathbf{T} \in \mathbb{R}^{n \times n}$ where $\mathbf{T}_{ij} = \mathbb{P}(h_{t+1} = j | h_t = i)$*
- *observation probabilities: $\mathbf{O} \in \mathbb{R}^{p \times n}$ where $\mathbf{O}_{ij} = \mathbb{P}(o_t = i | h_t = j)$*
- *initial distribution: $\boldsymbol{\pi} \in \mathbb{R}^n$ where $\boldsymbol{\pi}_i = \mathbb{P}(h_1 = i)$*

An HMM defines a probability distribution over all possible sequences of observations, the probability of a given sequence $x_1 x_2 x_3 \dots x_k$:

$$\mathbb{P}(x_1 x_2 x_3 \dots x_k) = \sum_{i_1=1}^{n} \mathbb{P}(h_1 = i_1)\mathbb{P}(o_1 = x_1 | h_1 = i_1) \sum_{i_2=1}^{n} \mathbb{P}(h_2 = i_2 | h_1 = i_1)\mathbb{P}(o_2 = x_2 | h_2 = i_2)$$
$$\sum_{i_3=1}^{n} \mathbb{P}(h_3 = i_3 | h_2 = i_2)\mathbb{P}(o_3 = x_3 | h_3 = i_3) \cdots \sum_{i_k=1}^{n} \mathbb{P}(h_k = i_k | h_{k-1} = i_{k-1})\mathbb{P}(o_k = x_k | h_k = i_k)$$

We can rewrite this expression in matrix form using the initial state distribution vector $\boldsymbol{\pi}$, as well as the transition and observation probabilities $\mathbf{T}$ and $\mathbf{O}$.

$$\mathbb{P}(x_1 x_2 x_3 ...) = \sum_{i_1} \boldsymbol{\pi}_{i_1} \mathbf{O}_{x_1, i_1} \sum_{i_2} \mathbf{T}_{i_1, i_2} \mathbf{O}_{x_2, i_2} \sum_{i_3} \mathbf{T}_{i_2, i_3} \mathbf{O}_{x_3, i_3} \dots$$

For the last step, we notice similarities between this computation and the one of a WA. In particular, we note that the following terms of the HMM product can be rewritten in terms of the following parameters of a WA:

$$\begin{aligned} \boldsymbol{\alpha}_0 &= \boldsymbol{\pi} \\ (\mathbf{A}^x)_{i,j} &= \mathbf{O}_{x,i} \mathbf{T}_{i,j} \\ \mathbf{A}^x &= diag\left(\mathbf{O}_{x,1}, \mathbf{O}_{x,2} \dots \mathbf{O}_{x,n}\right) \mathbf{T} \end{aligned}$$

As a result, we transform the HMM's product of sums above into an equivalent WA computation:

$$\mathbb{P}(\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3...) = \boldsymbol{\alpha}_0^\top \mathbf{A}^{x_1}\mathbf{A}^{x_2}\dots\mathbf{A}^{x_k}\mathbf{1} \quad \text{where } \boldsymbol{\alpha}_0 = \boldsymbol{\Pi},\ \boldsymbol{\alpha}_\infty = \mathbf{1}$$

This transformation from an arbitrary HMM to a corresponding WA leads us to the following property:

Proposition 3. *Any probability distribution computed by a HMM can be computed by a WA.*

5 Learning WAs and HMMs

In the previous section, we showed that HMMs can be transformed into weighted automata, but how can we use this to learn HMMs?

Traditionally, when *learning* an HMM, we are interested in learning the observation and transition probabilities, $\mathbf{O}$ and $\mathbf{T}$ respectively. Since, as described, we only see observations o, we resort to attempting to learn these quantities from data. Naively, one can use maximum likelihood estimation (MLE), but will see quickly that the MLE maximization becomes intractable. Instead, traditionally, we use an algorithm called expectation-maximization (EM). While the details of EM will not be covered here, the basic idea is that we estimate both quantities of interest simultaneously, holding one constant while maximizing the other, and then switching the roles. Although EM makes the problem tractable, it is not guaranteed to converge to the global optimum.

In the following paragraphs, we define important properties of WAs and their mappings f.

Definition 4. *(Minimality of Weighted Automata)*
A weighted automata is minimal if its number of states is minimal. That is, $\mathcal{A}$ is minimal if and only if any WA $\mathcal{B}$ such that $f_\mathcal{A} = f_\mathcal{B}$ has at least as many states as $\mathcal{A}$ Equivalently, a minimal WA for a function $f : \Sigma^* \text{ will be the smallest model that can compute } f$ (assuming that it exists).

Definition 5. *(Rank of f)*
Given a function $f : \Sigma^* \to \mathbb{R}$, the rank of f is the number of states of a minimal WA computing f (if f cannot be computed by a WA, we let $rank(f) = \infty$).

We now define the crucial notion of *Hankel* matrix. Given $f : \Sigma^* \to \mathbb{R}$, a Hankel matrix is a *bi-infinite* matrix whose rows and columns are indexed by words on Σ, $\mathbf{H} \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$, defined by $(\mathbf{H}_f)_{uv} = f(uv)$ for any $u, v \in \Sigma^*$. Here uv represents the concatenation of u and v; we will often call u a prefix and v a suffix. In other words, the rows can be thought of all prefixes that can be made from the alphabet, columns all the suffixes.
The following fundamental and surprising results relates the rank of the Hankel matrix with the size of a WA computing the corresponding function.

Theorem 6. *Let $f : \Sigma^* \to \mathbb{R}$. Its Hankel matrix $\mathbf{H}_f \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$ is a bi-infinite matrix defined by $(\mathbf{H}_f)_{u,v} = f(uv)$. We have $rank(\mathbf{H}_f) = rank(f)$.*

From a machine learning perspective, this theorem is very relevant because its proof is constructive: given a low rank factorization of the Hankel matrix, it shows how one can recover a WA computing the corresponding function. We will see that this can be leveraged to recover (or approximate) an underlying WA from an estimation of the Hankel matrix $\mathbf{H}_f$. In practice, we construct a subblock of $\mathbf{H}_f$, take the SVD of that subblock, and use the resulting low-rank approximation to recover the WA that best models the training data.

Proposition 7. *WAs are invariant under change of basis.*

Proof. Let $\mathcal{B} = (\boldsymbol{\beta}_0, \{\mathbf{B}^\sigma\}_{\sigma\in\Sigma}, \boldsymbol{\beta}_\infty)$ be the WA defined by

$$\boldsymbol{\beta}_0^\top = \boldsymbol{\alpha}_0^\top \mathbf{M}$$

$$\mathbf{B}^\sigma = \mathbf{M}^{-1}\mathbf{A}^\sigma\mathbf{M}$$

$$\boldsymbol{\beta}_\infty = \mathbf{M}^{-1}\boldsymbol{\alpha}_\infty$$

where $\mathbf{M}$ is an invertible matrix. Now, we want to show that for any word x, the two functions are equivalent. For any x,

$$f_\mathcal{B} = \boldsymbol{\alpha}_0^\top \mathbf{M}\mathbf{M}^{-1}\mathbf{A}^x\mathbf{M}\mathbf{M}^{-1}\boldsymbol{\alpha}_\infty$$

which, after the full multiplication, reduces to:

$$f_\mathcal{B} = \boldsymbol{\alpha}_0^\top \mathbf{A}^x\boldsymbol{\alpha}_\infty = f_\mathcal{A}$$

So we can conclude that both WA $\mathcal{A}$ and $\mathcal{B}$ compute the same function, if $\mathcal{B}$ is just $\mathcal{A}$ under a change of basis:

$$f_\mathcal{B} = f_\mathcal{A}$$

□

We will now prove Theorem 6. We want to show that the rank of the Hankel matrix $(\mathbf{H}_f)$ is exactly equal to the rank of the function f. We will construct this proof in two parts: since we'd like to show that $x = y$, we will first show that $x \leq y$, and then $y \leq x$.

Proof. Here we show that $rank(\mathbf{H}) \leq rank(f)$, which we do by showing that for any WA with m states that computes f, the $rank(\mathbf{H})$ is smaller than m.

1. Let weighted automata $\mathcal{A} = (\boldsymbol{\alpha}_0, \mathbf{A}^\sigma\}_{\sigma\in\Sigma}, \boldsymbol{\alpha}_\infty)$ with m states

2. Let $\mathbf{H}_{u,v} = f(u,v) = (\boldsymbol{\alpha}_0^T\mathbf{A}^u)(\mathbf{A}^v\boldsymbol{\alpha}_\infty)$, which is true for all prefixes, suffixes u, v.

3. We can rewrite the Hankel matrix as $\mathbf{H} = \mathbf{PS}$, where $\mathbf{P} \in \mathbb{R}^{\Sigma^*\times m} = \begin{pmatrix} — \boldsymbol{\alpha}_0^T\mathbf{A}^u — \end{pmatrix}_{u\in\Sigma^*}$ and $\mathbf{S} \in \mathbb{R}^{m\times\Sigma^*} = \begin{pmatrix} — \mathbf{A}^v\boldsymbol{\alpha}_\infty — \end{pmatrix}^T_{v\in\Sigma^*}$, which is the rank-m factorization of the Hankel matrix.

4. We know that the rank of a matrix product $\mathbf{AB} \leq min(rank(\mathbf{A}), rank(\mathbf{B}))$, and since we know that the rank of a matrix is given by the $min(rows, cols)$, we know that the $rank(\mathbf{H}) \leq m = rank(f)$, if the WA associated with f is *minimal*.

From the proof above, we see that the minimal WA associated with f *induces* the rank of the Hankel matrix. We know show that $rank(f) \leq rank(\mathbf{H})$.
We do this by showing that if the $rank(\mathbf{H}_f) = m$, then we can find a WA with m states computer f, which will allow us to conclude that $rank(f) \leq m$ (since $rank(f)$ is the size of the smallest WA computing f).

1. For each $x \in \Sigma^*$, we *define* $\mathbf{H}^x \in \mathbb{R}^{\Sigma^*\times\Sigma^*}$ by $(\mathbf{H}^x)_{u,v} = f(uxv)$

2. We can now rewrite $(\mathbf{H}^x)_{uv} = \boldsymbol{\alpha}_0^T\mathbf{A}^u\mathbf{A}^x\mathbf{A}^v\boldsymbol{\alpha}_\infty$, which gives us the factorization $\mathbf{H}^x = \mathbf{P}\mathbf{A}^x\mathbf{S}$.

3. If we assume that $\mathbf{P}, \mathbf{S}$ are full rank and *are given*, and that $\mathbf{P}$ is left-invertible, and $\mathbf{S}$ is right invertible, we can recover $\mathbf{A}^x = \mathbf{P}^+\mathbf{H}^x\mathbf{S}^+$, where the $+$ superscript denotes the pseudo-inverse operation.

4. We note that since $\mathbf{H} = \mathbf{H}^\lambda$, where λ is the empty word, $(\mathbf{H}^x)_{uv} = \mathbf{H}_{u,xv} = \mathbf{H}_{ux,v}$, which implies that any row of $\mathbf{H}^x$ is a row of $\mathbf{H}$. Similarly, any columns of $\mathbf{H}^x$ is a row of H. Consequently the row space (respectively column space) of $\mathbf{H}^x$ is a subspace of the row space (respectively column space) of H.

5. To prove our property of interest: that $rank(f) \leq rank(\mathbf{H})$, we let $m = rank(\mathbf{H})$ and let $\mathbf{H} = \mathbf{PS}$ be any rank-m factorization of $\mathbf{H}$ (note that this implies that $\mathbf{P} \in \mathbb{R}^{\Sigma^*\times m}$ and $\mathbf{S} \in \mathbb{R}^{m\times\Sigma^*}$ are both full rank).

6. We construct a weighted automata $\mathcal{A} = (\boldsymbol{\alpha}_0, \{\mathbf{A}^\sigma\}_{\sigma\in\Sigma}, \boldsymbol{\alpha}_\infty)$ with m states that computes f.

7. For each $\sigma \in \Sigma$, let $\mathbf{A}^\sigma \in \mathbb{R}^{m\times m} = \mathbf{P}^+\mathbf{H}^\sigma\mathbf{S}^+$, $\boldsymbol{\alpha}_0^T = \mathbf{P}_{\lambda,:}$ (first row), and $\boldsymbol{\alpha}_\infty = \mathbf{S}_{:,\lambda}$ (first column).

8. We claim that for any word $x = x_1x_2...x_k \in \Sigma^*$, we have $\mathbf{A}^x = \mathbf{A}^{x_1}\mathbf{A}^{x_2}..\mathbf{A}^{x_k} = \mathbf{P}^+\mathbf{H}^x\mathbf{S}^+$. First, for any $x \in \Sigma^*$ let $\mathbf{\Pi}^x \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$ be defined by $\mathbf{\Pi}_{u,v} = 1$ if $u = xv$ and 0 otherwise, for all $u, v \in \Sigma^*$. One can easily check that $\mathbf{H}^x = \mathbf{H}\mathbf{\Pi}^x$ and $\mathbf{\Pi}^{xy} = \mathbf{\Pi}^x\mathbf{\Pi}^y$ for all $x, y, \in \Sigma^*$. Now, suppose the hypothesis true for given words x and y (the base case for the induction is immediate), we have

$$\mathbf{A}^{xy} = \mathbf{A}^x\mathbf{A}^y = \mathbf{P}^+\mathbf{H}^x\mathbf{S}^+\mathbf{P}^+\mathbf{H}^y\mathbf{S}^+.$$

Using the fact that $\mathbf{H}^y = \mathbf{H}\mathbf{\Pi}^y = \mathbf{P}\mathbf{S}\mathbf{\Pi}^y$ it follows that

$$\mathbf{A}^{xy} = \mathbf{P}^+\mathbf{H}^x\mathbf{S}^+\mathbf{S}\mathbf{\Pi}^y\mathbf{S}^+$$

but since $\mathbf{S}^+\mathbf{S}$ is the projection onto the row space of $\mathbf{H}$, which contains the row space of $\mathbf{H}^x$, it follows that $\mathbf{H}^x\mathbf{S}^+\mathbf{S} = \mathbf{H}^x$ and

$$\mathbf{A}^{xy} = \mathbf{P}^+\mathbf{H}^x\mathbf{\Pi}^y\mathbf{S}^+ = \mathbf{P}^+\mathbf{H}\mathbf{\Pi}^x\mathbf{\Pi}^y\mathbf{S}^+ = \mathbf{P}^+\mathbf{H}\mathbf{\Pi}^{xy}\mathbf{S}^+ = \mathbf{P}^+\mathbf{H}^{xy}\mathbf{S}^+.$$

9. It follows from the previous step that

$$f_A(x) = \boldsymbol{\alpha}_0^T\mathbf{P}^+\mathbf{H}^x\mathbf{S}^+\boldsymbol{\alpha}_\infty = (\mathbf{P}\mathbf{P}^+\mathbf{H}^x\mathbf{S}^+\mathbf{S})_{\lambda,\lambda} = (\mathbf{H}^x)_{\lambda,\lambda} = f(\lambda x\lambda) = f(x)$$

for all $x \in \Sigma^*$.

The last step of the proof deserves a closer look. Why within $(\mathbf{P}\mathbf{P}^+\mathbf{H}^x\mathbf{S}^+\mathbf{S})_{\lambda,\lambda}$, do $\mathbf{P}\mathbf{P}^+$ and $\mathbf{S}^+\mathbf{S}$ act like the identity, when we know they are not the identity?

From previous lectures, we know that $\mathbf{P}\mathbf{P}^+$ and $\mathbf{S}^+\mathbf{S}$ are projection matrices, but since $\mathbf{P}\mathbf{P}^+$ is a projection onto the range of $\mathbf{H}$, and since the range of $\mathbf{H}^x$ is a subspace of the range of $\mathbf{H}$ (same with $\mathbf{S}^+\mathbf{S}$ with the rowspace), we know that they operate like the identity on these subspaces. □

This shows us that if we had the Hankel matrix, we could recover the associated weighted automata. But, since we don't have the Hankel matrix, what can we do to approximate it? In particular, we can never hope to have access to the *infinite* Hankel matrix. The following corollary shows that having access to only a sub-block of the Hankel matrix is sufficient.

Corollary 8. *Let $f : \Sigma^* \to \mathbb{R}$ be a function of rank m. Let $\mathcal{P}, \mathcal{S} \subseteq \Sigma^*$ be such that $\lambda \in \mathcal{P} \cap \mathcal{S}$ and such that the subblock $\tilde{\mathbf{H}}_f \in \mathbb{R}^{\mathcal{P}\times\mathcal{S}}$ of the full Hankel matrix satisfies $rank(\tilde{\mathbf{H}}_f) = rank(\mathbf{H}_f) = m$*

Then, for any rank-m factorization $\tilde{\mathbf{H}}_f = \mathbf{P}\mathbf{S}$, the WA $\mathcal{A} = (\boldsymbol{\alpha}_0, \{\mathbf{A}^\sigma\}_{\sigma\in\Sigma}, \boldsymbol{\alpha}_\infty)$, where $\boldsymbol{\alpha}_0^T = \mathbf{P}_{\lambda,:}$ (first row), and $\boldsymbol{\alpha}_\infty = \mathbf{S}_{:,\lambda}$ (first column), $\mathbf{A}^\sigma = \mathbf{P}^+\tilde{\mathbf{H}}^\sigma\mathbf{S}^+$, where $\tilde{\mathbf{H}}^\sigma \in \mathbb{R}^{\mathcal{P}\times\mathcal{S}}$ are the matrices defined by $(\tilde{\mathbf{H}}^\sigma)_{u,v} = f(u\sigma v)$ for all $u \in \mathcal{P}, v \in \mathcal{S}$, is such that $f_A = f$.

6 Practical Considerations

In the basic setup where the data are strings sampled from probability distribution on Σ^*, Hankel matrix is estimated by empirical probabilities, and factorization and low-rank approximation is computed using SVD, the overall learning process can be summarized as following:

(i) Choose $\mathcal{P}, \mathcal{S} \subseteq \Sigma^*$

(ii) Estimate $\mathbf{H}, \mathbf{H}^\sigma \in \mathbb{R}^{\mathcal{P}\times\mathcal{S}}$ from the training data.

(iii) Perform rank m factorization (m can be considered as the hyper-parameter) $\mathbf{H} \approx \mathbf{P}\mathbf{S}$ (using SVD)

(iv) $\boldsymbol{\alpha}_0^\top = \mathbf{P}_{\lambda,:}$, and $\boldsymbol{\alpha}_\infty = \mathbf{S}_{:,\lambda}$ and $\mathbf{A}^\sigma = \mathbf{P}^+\tilde{\mathbf{H}}^\sigma\mathbf{S}^+$

We give some important notes about the previous steps (i) and (ii) above. First, the pair $\mathcal{P}, \mathcal{S}$ defining the sub-block and are called basis, $\mathcal{P}$ must contain strings reaching each possible state of the WA and $\mathcal{S}$ must contain string producing different outcomes for each pair of states in the WA; which are prefixes and suffixes respectively. There are some approaches to choose them, the following are different approaches for doing that

(i) (a) Choose all words of length $\leq K$ for $\mathcal{P}$ and $\mathcal{S}$ [3]

(b) Identify all prefixes and suffixes in the training data. [1]

(c) Take K most frequent prefixes and suffixes [2]

(ii) In this note will discuss how can we estimate the Hankel matrix: if f is a probability distribution, this would be an easy task as we can just use empirical probabilities in the training data to fill the entries of $\mathbf{H}$. For example, suppose that the training data is of the form $\mathcal{D} = (aa, aa, ab, abb, abb, a, ab, \lambda, \lambda, ...)$, we can use the empirical probability distribution of the training set to build the Hankel matrix with rows represent $\mathcal{P}$ and columns represent $\mathcal{S}$ which we have identified from the training data, and the matrix values will be the empirical frequencies:

$$\tilde{H} = \begin{pmatrix} 0.02 & 0.15 & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

Let us now consider the case where f is not a probability distribution. In this case, we can assume that our training data consists of pairs of input/output examples, but in this case there are entries in the Hankel matrix corresponding to words that we never saw in the training data. What should we do here? (note that when f was a probability it made sense to put 0 for words that we did not see in the training data). Let's look at the following example: $\mathcal{D} = \{(a, 0.5), (b, 1), (aa, 2), ...\} \subseteq \Sigma^* \mathbb{R}$. The the Hankel matrix would look like

$$\tilde{H} = \begin{pmatrix} ? & 0.5 & -1 \\ 0.5 & 2 & ? \\ -1 & ? & ? \end{pmatrix}$$

Here, we need to guess the blanks in the data matrix, denoted by (?), i.e. we need to perform **matrix completion** on the Hankel matrix $\mathbf{H}$ before performing the SVD step of the spectral method.

The task of matrix completion is often encountered in recommender systems, where the entries of a matrix are, for instance, categorical values of ratings. Suppose that we have the following matrix with missing entries:

$$\mathbf{M} = \begin{pmatrix} 1 & 2 & 4 \\ ? & 4 & ? \\ -1 & ? & ? \end{pmatrix}$$

The task of matrix completion asks to fill the missing entries in this matrix. Without further assumptions, this is an ill-posed problem, we could put whatever values we want to fill the missing entries! However, if you know that the rank of $\mathbf{M}$ is 1, then there is now a unique choice of values to use for filling the blanks... This is the main idea between matrix completion: one can leverage the fact the a matrix is of low rank to fill its missing entries. Ideally, if a matrix is of rank R, we would like to be able to recover all of its entries after observing only $\mathcal{O}(nR)$ entries (which is the order of the number of parameters needed to specify a rank R matrix, using e.g. the SVD parametrization).

7 WAs as recurrent models

In this last section, we briefly show a high level view of WA showing how they relate to other models such as RNNs. We also present how the WA model can naturally be extended to the setting of input sequences of continuous vectors instead of discrete symbols.
Let us give a generic definition of a recurrent model subsuming both WAs, RNNs and HMMs:

Definition 9. *Recurrent model*
A recurrent model maps a sequence of inputs $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k \in \mathcal{X}$ *to:*

(i) A sequence of latent states $\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_k \in \mathbb{R}^n$ *where*
$\mathbf{h}_t = \phi(\mathbf{h}_{t-1}, \mathbf{x}_t)$ *for some recurrent function* $\phi : \mathbb{R}^m \times \mathcal{X} \to \mathbb{R}^m$

(ii) Sequence of outputs $\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_k \in \mathbb{R}^p$ *where*
$\mathbf{y}_t = \psi(\mathbf{h}_t)$ *for some output function* $\psi : \mathbb{R}^m \to \mathbb{R}^p$

Example: for a WA $\mathcal{A} = (\boldsymbol{\alpha}_0, \mathbf{A}^\sigma, \boldsymbol{\alpha}_\infty)$

$$\begin{aligned} \mathbf{h}_0 &= \boldsymbol{\alpha}_0 \\ \mathbf{h}_t &= \phi(\mathbf{h}_{t-1}, \sigma_t) = (\mathbf{A}^\sigma)^\top \mathbf{h}_{t-1} \\ \mathbf{y}_t &= \psi(\mathbf{h}_t) = \boldsymbol{\alpha}_\infty^\top \boldsymbol{h}_t \end{aligned} \tag{*}$$

Now, let $\mathcal{T} \in \mathbb{R}^{\Sigma \times m \times m}$ be defined by $\mathcal{T}_{\sigma,:,:} = \mathbf{A}^\sigma$ for each $\sigma \in \Sigma$. We can rewrite equation (*) as:

$$\mathbf{h}_t = \mathcal{T} \bullet_1 \mathbf{e}_\sigma \bullet_2 \mathbf{h}_{t-1}$$

where $\mathbf{e}_\sigma \in \mathbb{R}^\Sigma$ is the one hot encoding of the symbol σ.This suggests a natural generalization of WA where instead of restricting inputs to be one-hot encodings of symbols from a discrete alphabet, we allow continuous vectors as inputs [4]:
Continuous WA: A continuous WA is a tuple $\mathcal{A} = (\boldsymbol{\alpha}_0, \mathcal{T}, \boldsymbol{\alpha}_\infty)$ where $\mathcal{T} \in \mathbb{R}^{d \times m \times m}$ computing the function

$$f_\mathcal{A}(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k) = \mathbf{h}_k^\top \boldsymbol{\alpha}_\infty$$

where each $\mathbf{x}_i \in \mathbb{R}^d$ is a vector and teh sequence of latent states $\mathbf{H}_i$ is recursaively defined by

$$\mathbf{h}_0 = \boldsymbol{\alpha}_0$$

$$\mathbf{h}_t = \mathcal{T} \bullet_1 \boldsymbol{x}_t \bullet_2 \mathbf{h}_{t-1}$$

References

[1] R. Bailly, F. Denis, and L. Ralaivola. Grammatical inference as a principal component analysis problem. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 33–40, 2009. ISBN 978-1-60558-516-1.

[2] B. Balle, X. Carreras, F. M. Luque, and A. Quattoni. Spectral learning of weighted automata. *Machine Learning*, 96:33–63, 2013.

[3] D. Hsu, S. M. Kakade, and T. Zhang. A spectral algorithm for learning hidden markov models, 2008.

[4] G. Rabusseau, T. Li, and D. Precup. Connecting weighted automata and recurrent neural networks through spectral learning. *arXiv preprint arXiv:1807.01406*, 2018.