# COMS 4705.H: Hidden Markov Models

Karl Stratos

February 10, 2017

# Motivation: Part-of-Speech (POS) Tagging

**Task.** Given a sentence, output a sequence of POS tags.

**Ambiguity.** A word can have many possible POS tags.

```
the/DT man/NN saw/VBD the/DT cut/NN
the/DT saw/NN cut/VBD the/DT man/NN
```

**Solution.** Use a statistical approach to disambiguate.

# Overview

Derivation of an HMM

Parameter Estimation from Labeled Data

Computation with an HMM

- Marginalization and Inference
- Forward Algorithm
- Viterbi Algorithm
- Practical Issues

Beam Search

# Sequence Labeling with a Probabilistic Model

Vocabulary $V$, set of POS tags $L$

$$V = \{\texttt{prim}, \texttt{that}, \texttt{Arya}, \texttt{fastidiously}, \texttt{1988}, \ldots\}$$
$$L = \{\texttt{DT}, \texttt{NN}, \texttt{VBD}, \texttt{JJ}, \ldots\}$$

# Sequence Labeling with a Probabilistic Model

Vocabulary $V$, set of POS tags $L$

$$V = \{\texttt{prim}, \texttt{that}, \texttt{Arya}, \texttt{fastidiously}, \texttt{1988}, \ldots\}$$
$$L = \{\texttt{DT}, \texttt{NN}, \texttt{VBD}, \texttt{JJ}, \ldots\}$$

Want to define a **joint** distribution $p(x_1 \ldots x_m, \ y_1 \ldots y_m)$ over

1. Any sentence $x_1 \ldots x_m \in V^m$
2. A corresponding sequence of POS tags $y_1 \ldots y_m \in L^m$

# Sequence Labeling with a Probabilistic Model

Vocabulary $V$, set of POS tags $L$

$$V = \{\texttt{prim},\ \texttt{that},\ \texttt{Arya},\ \texttt{fastidiously},\ \texttt{1988},\ \ldots\}$$
$$L = \{\texttt{DT},\ \texttt{NN},\ \texttt{VBD},\ \texttt{JJ},\ \ldots\}$$

Want to define a **joint** distribution $p(x_1 \ldots x_m,\ y_1 \ldots y_m)$ over

1. Any sentence $x_1 \ldots x_m \in V^m$
2. A corresponding sequence of POS tags $y_1 \ldots y_m \in L^m$

Why? Then we can infer for any given $x_1 \ldots x_m$

$$\begin{aligned} y_1^* \ldots y_m^* &= \operatorname*{arg\,max}_{y_1 \ldots y_m \in L^m} p(y_1 \ldots y_m | x_1 \ldots x_m) \\ &= \operatorname*{arg\,max}_{y_1 \ldots y_m \in L^m} p(x_1 \ldots x_m,\ y_1 \ldots y_m) \end{aligned}$$

# A Left-to-Right Generative Process

By the chain rule, we may assume that

$$\begin{aligned} & p(x_1 \dots x_m,\ y_1 \dots y_m) \\ & \quad = p(y_1) \times p(x_1|y_1) \times p(y_2|x_1, y_1) \times p(x_2|x_1, y_1, y_2) \cdots \\ & \qquad \times p(y_m|\{x_i, y_i\}_{i=1}^{m-1}) \times p(x_m|\{x_i, y_i\}_{i=1}^{m-1}, y_m) \\ & \qquad \times p(\texttt{STOP}|\{x_i, y_i\}_{i=1}^{m}) \end{aligned}$$

# A Left-to-Right Generative Process

By the chain rule, we may assume that

$$
\begin{aligned}
& p(x_1 \dots x_m,\ y_1 \dots y_m) \\
& \qquad = p(y_1) \times p(x_1|y_1) \times p(y_2|x_1, y_1) \times p(x_2|x_1, y_1, y_2) \cdots \\
& \qquad\quad \times p(y_m| \left\{x_i, y_i\right\}_{i=1}^{m-1}) \times p(x_m| \left\{x_i, y_i\right\}_{i=1}^{m-1}, y_m) \\
& \qquad\quad \times p(\texttt{STOP}| \left\{x_i, y_i\right\}_{i=1}^{m})
\end{aligned}
$$

Design a tractable model by making **independence assumptions**.

- What kind of assumption is reasonable for POS tagging?

# First-Order HMM Assumptions

1. At any position $i$, the word depends on the current tag only.

$$p(x_i | \{x_j, y_j\}_{j=1}^{i-1}, y_i) = p(x_i | y_i)$$

2. At any position $i$, the tag depends on the previous tag only.

$$p(y_i | \{x_j, y_j\}_{j=1}^{i-1}) = p(y_i | y_{i-1})$$

# Model Parameters

- $|V| \times |L|$ "emission" probabilities

$$o(x|y) = \text{probability of emitting word } x \text{ given tag } y$$

- $|L|^2 + 2\,|L|$ "transition" probabilities

$$\begin{aligned} t(y'|y) &= \text{probability of transitioning from tag } y \text{ to } y' \\ t(y|*) &= \text{probability of starting with tag } y \\ t(\texttt{STOP}|y) &= \text{probability of ending with tag } y \end{aligned}$$

Used to calculate

$$p(x_1 \ldots x_m,\ y_1 \ldots y_m) = \prod_{i=1}^{m+1} t(y_i|y_{i-1}) \times \prod_{i=1}^{m} o(x_i|y_i)$$

where $y_0 = *$ and $y_{m+1} = \texttt{STOP}$ are special symbols.

# Overview

# Labeled Data

- Consists of $N$ annotated sentences $(x^{(1)}, y^{(1)}) \dots (x^{(N)}, y^{(N)})$ where $l_i = \left|x^{(i)}\right| = \left|y^{(i)}\right|$ and $y_0^{(i)} = *$, $y_{l_i+1}^{(i)} = \texttt{STOP}$.

- Define **count**$(y, y')$ for $y, y' \in L \cup \{*, \texttt{STOP}\}$:

$$\textbf{count}(y, y') = \sum_{i=1}^{N} \sum_{\substack{j=1: \\ y_{j-1}^{(i)}=y \\ y_j^{(i)}=y'}}^{l_i+1} 1$$

- Define **count**$(x, y)$ for $x \in V,\ y \in L$:

$$\textbf{count}(x, y) = \sum_{i=1}^{N} \sum_{\substack{j=1: \\ x_j^{(i)}=x \\ y_j^{(i)}=y}}^{l_i} 1$$

# Parameter Estimation

- For all $y, y'$ with **count**$(y, y') > 0$, set

$$t(y'|y) = \frac{\textbf{count}(y, y')}{\sum_{y' \in L} \textbf{count}(y, y')}$$

Otherwise $t(y'|y) = 0$.

# Parameter Estimation

- For all $y, y'$ with **count**$(y, y') > 0$, set

$$t(y'|y) = \frac{\textbf{count}(y, y')}{\sum_{y' \in L} \textbf{count}(y, y')}$$

  Otherwise $t(y'|y) = 0$.

- For all $x, y$ with **count**$(x, y) > 0$, set

$$o(x|y) = \frac{\textbf{count}(x, y)}{\sum_{x \in V} \textbf{count}(x, y)}$$

  Otherwise $o(x|y) = 0$.

# Justification

**Claim.** The solution of

$$o^*, t^* = \underset{\substack{o,\, t:\;\; o(x|y),\, t(y'|y) \geq 0 \\ \sum_{y'} t(y'|y) = \sum_x o(x|y) = 1}}{\arg\max} \sum_{i=1}^{N} \log p(x^{(i)}, y^{(i)})$$

where $p(x, y)$ is the distribution of an HMM is given by

$$o^*(x|y) = \frac{\textbf{count}(x, y)}{\sum_x \textbf{count}(x, y)} \qquad t^*(y'|y) = \frac{\textbf{count}(y, y')}{\sum_{y'} \textbf{count}(y, y')}$$

# Overview

# Setting

- We now assume that we have parameters $o(x|y)$ and $t(y'|y)$.

# Setting

- We now assume that we have parameters $o(x|y)$ and $t(y'|y)$.

- They define the joint probability distribution

$$p(x_1 \dots x_m,\ y_1 \dots y_m) = \prod_{i=1}^{m+1} t(y_i|y_{i-1}) \times \prod_{i=1}^{m} o(x_i|y_i)$$

  over any words $x_1 \dots x_m \in V^m$ and POS tags $y_1 \dots y_m \in L^m$.

# Setting

- We now assume that we have parameters $o(x|y)$ and $t(y'|y)$.
- They define the joint probability distribution

$$p(x_1 \ldots x_m,\ y_1 \ldots y_m) = \prod_{i=1}^{m+1} t(y_i|y_{i-1}) \times \prod_{i=1}^{m} o(x_i|y_i)$$

  over any words $x_1 \ldots x_m \in V^m$ and POS tags $y_1 \ldots y_m \in L^m$.
- **Given a fixed sentence** $x_1 \ldots x_m \in V^m$, we often wish to perform two critical calculations (next slide).

# Marginalization and Inference

1. What is the probability of $x_1 \dots x_m$ under the HMM?

$$\sum_{y_1 \dots y_m \in L^m} p(x_1 \dots x_m,\ y_1 \dots y_m)$$

# Marginalization and Inference

1. What is the probability of $x_1 \dots x_m$ under the HMM?

$$\sum_{y_1 \dots y_m \in L^m} p(x_1 \dots x_m,\ y_1 \dots y_m)$$

2. What is the most probable $y_1 \dots y_m \in L^m$ under the HMM?

$$\arg\max_{y_1 \dots y_m \in L^m} p(x_1 \dots x_m,\ y_1 \dots y_m)$$

# Number of Possible Tag Sequences

- **Exponential in the length of the sentence**
- Enumerating all $m^{|L|}$ candidates is clearly not practical.
- We will exploit the HMM assumptions to perform marginalization/inference **exactly** and with **polynomial complexity**.

# Overview

# Left-to-Right Incremental Marginalization

- **Idea.** No need to consider all $m^{|L|}$ candidates because of the left-to-right generative process and independence assumptions under the HMM

# Left-to-Right Incremental Marginalization

- **Idea.** No need to consider all $m^{|L|}$ candidates because of the left-to-right generative process and independence assumptions under the HMM

- **Forward algorithm.** For $i = 1 \dots m$, for all $y \in L$,

$$\pi(i, y) := \sum_{y_1 \dots y_i \in L^i :\ y_i = y} p(x_1 \dots x_i, y_1 \dots y_i)$$

  We will see that computing each $\pi(i, y)$ takes $O(|L|)$ time using **dynamic programming**.

# Left-to-Right Incremental Marginalization

- **Idea.** No need to consider all $m^{|L|}$ candidates because of the left-to-right generative process and independence assumptions under the HMM

- **Forward algorithm.** For $i = 1 \dots m$, for all $y \in L$,

$$\pi(i, y) := \sum_{y_1 \dots y_i \in L^i :\ y_i = y} p(x_1 \dots x_i, y_1 \dots y_i)$$

  We will see that computing each $\pi(i, y)$ takes $O(|L|)$ time using **dynamic programming**.

- Total runtime?

# Base Case ($i = 1$)

$$\pi(1, y) := \sum_{y_1 \in L:\, y_1 = y} p(x_1, y_1)$$
$$= t(y|*) \times o(x_1|y)$$

# Main Body ($i > 1$)

$$\pi(i, y') := \sum_{y_1 \ldots y_i :\, y_i = y'} p(x_1 \ldots x_i, y_1 \ldots y_i)$$

# Main Body ($i > 1$)

$$\pi(i, y') := \sum_{y_1 \ldots y_i :\, y_i = y'} p(x_1 \ldots x_i, y_1 \ldots y_i)$$

$$= \sum_{y_1} \cdots \sum_{y_{i-1}} p(x_1 \ldots x_i, y_1 \ldots y_{i-1}\ y')$$

# Main Body ($i > 1$)

$$\pi(i, y') := \sum_{y_1 \ldots y_i:\ y_i = y'} p(x_1 \ldots x_i, y_1 \ldots y_i)$$

$$= \sum_{y_1} \cdots \sum_{y_{i-1}} p(x_1 \ldots x_i, y_1 \ldots y_{i-1}\ y')$$

$$= \sum_{y_1} \cdots \sum_{y_{i-1}} p(x_1 \ldots x_{i-1}, y_1 \ldots y_{i-1}) \times t(y'|y_{i-1}) \times o(x_i|y')$$

# Main Body ($i > 1$)

$$\begin{aligned}
\pi(i, y') &:= \sum_{y_1 \ldots y_i :\, y_i = y'} p(x_1 \ldots x_i, y_1 \ldots y_i) \\
&= \sum_{y_1} \cdots \sum_{y_{i-1}} p(x_1 \ldots x_i, y_1 \ldots y_{i-1}\ y') \\
&= \sum_{y_1} \cdots \sum_{y_{i-1}} p(x_1 \ldots x_{i-1}, y_1 \ldots y_{i-1}) \times t(y' | y_{i-1}) \times o(x_i | y') \\
&= \sum_{y} \sum_{y_1 \ldots y_{i-2}} p(x_1 \ldots x_{i-1}, y_1 \ldots y_{i-2}\ y) \times t(y' | y) \times o(x_i | y')
\end{aligned}$$

# Main Body ($i > 1$)

$$
\begin{aligned}
\pi(i, y') &:= \sum_{y_1 \ldots y_i :\ y_i = y'} p(x_1 \ldots x_i, y_1 \ldots y_i) \\
&= \sum_{y_1} \cdots \sum_{y_{i-1}} p(x_1 \ldots x_i, y_1 \ldots y_{i-1}\ y') \\
&= \sum_{y_1} \cdots \sum_{y_{i-1}} p(x_1 \ldots x_{i-1}, y_1 \ldots y_{i-1}) \times t(y' | y_{i-1}) \times o(x_i | y') \\
&= \sum_{y} \sum_{y_1 \ldots y_{i-2}} p(x_1 \ldots x_{i-1}, y_1 \ldots y_{i-2}\ y) \times t(y' | y) \times o(x_i | y') \\
&= \sum_{y} \pi(i-1, y) \times t(y' | y) \times o(x_i | y')
\end{aligned}
$$

# Final Marginalization

Obtain the probability of $x_1 \ldots x_m$ under the HMM by

$$\sum_{y_1 \ldots y_m} p(x_1 \ldots x_m,\ y_1 \ldots y_m) = \sum_{y \in L} \pi(m, y)$$

# Overview

# Left-to-Right Incremental Maximization

- **Same Idea.** Use the properties of the HMM.

# Left-to-Right Incremental Maximization

- **Same Idea.** Use the properties of the HMM.

- **Viterbi algorithm.** For $i = 1 \dots m$, for all $y \in L$,

$$\pi(i, y) := \max_{y_1 \dots y_i \in L^i :\, y_i = y} \; p(x_1 \dots x_i, y_1 \dots y_i)$$

# Left-to-Right Incremental Maximization

- **Same Idea.** Use the properties of the HMM.

- **Viterbi algorithm.** For $i = 1 \dots m$, for all $y \in L$,

$$\pi(i, y) := \max_{y_1 \dots y_i \in L^i :\, y_i = y} \; p(x_1 \dots x_i, y_1 \dots y_i)$$

- The *only* difference from the forward alg: "$\sum$" $\mapsto$ "$\max$"

$$\pi(1, y) = t(y|*) \times o(x_1|y)$$
$$\pi(i, y') = \max_{y \in L} \; \pi(i-1, y) \times t(y'|y) \times o(x_i|y')$$

# Left-to-Right Incremental Maximization

- **Same Idea.** Use the properties of the HMM.

- **Viterbi algorithm.** For $i = 1 \dots m$, for all $y \in L$,

$$\pi(i, y) := \max_{y_1 \dots y_i \in L^i :\, y_i = y} \; p(x_1 \dots x_i, y_1 \dots y_i)$$

- The *only* difference from the forward alg: "$\sum$" $\mapsto$ "$\max$"

$$\pi(1, y) = t(y|*) \times o(x_1|y)$$
$$\pi(i, y') = \max_{y \in L} \; \pi(i-1, y) \times t(y'|y) \times o(x_i|y')$$

- But how do we extract the actual **tag sequence**?

$$y_1^* \dots y_m^* = \underset{y_1 \dots y_m \in L^m}{\arg\max} \; p(x_1 \dots x_m,\; y_1 \dots y_m)$$

# Backtracking

- Keep an *additional* chart to record the **path**:

$$\beta(i, y') = \arg\max_{y \in L} \; \pi(i-1, y) \times t(y'|y) \times o(x_i|y')$$

for $i = 2 \ldots m$.

# Backtracking

- Keep an *additional* chart to record the **path**:

$$\beta(i, y') = \arg\max_{y \in L} \ \pi(i-1, y) \times t(y'|y) \times o(x_i|y')$$

for $i = 2 \ldots m$.

- After running Viterbi, we can "backtrack"

$$\begin{aligned} y_m^* &= \arg\max_{y \in L} \ \pi(m, y) \\ y_{m-1}^* &= \beta(m, y_m^*) \\ &\vdots \\ y_1^* &= \beta(2, y_2^*) \end{aligned}$$

and return $y_1^* \ldots y_m^*$.

# Overview

# Log Space

- For numerical stability, always operate in **log space**.

# Log Space

- For numerical stability, always operate in **log space**.
- For Viterbi, it's a simple change:

$$\pi(1, y) = \log t(y|*) + \log o(x_1|y)$$
$$\pi(i, y') = \max_{y \in L} \; \pi(i-1, y) + \log t(y'|y) + \log o(x_i|y')$$

# Log Space

- For numerical stability, always operate in **log space**.

- For Viterbi, it's a simple change:

$$\pi(1, y) = \log t(y|*) + \log o(x_1|y)$$
$$\pi(i, y') = \max_{y \in L} \ \pi(i-1, y) + \log t(y'|y) + \log o(x_i|y')$$

- For the forward algorithm, we need a helper function:

$$\text{logsum}(\log(c_1) \ldots \log(c_n))$$

returns $\log(c_1 + \cdots + c_n)$ **without exponentiating** $\log(c_i)$!

# Log Space: Forward Algorithm

- Original:

$$\pi(1, y) = t(y|*) \times o(x_1|y)$$
$$\pi(i, y') = \sum_{y \in L} \pi(i-1, y) \times t(y'|y) \times o(x_i|y')$$

- Log space:

$$\pi(1, y) = \log t(y|*) + \log o(x_1|y)$$
$$\pi(i, y') = \operatorname*{logsum}_{y \in L} \pi(i-1, y) + \log t(y'|y) + \log o(x_i|y')$$

# Trick to Sum Logs

**Input**: $\log a \geq \log b$
**Output**: $\log(a + b)$

- If $\log a < -\infty$: return $-\infty$.
- If $\log b - \log a < -20$: return $\log a$.
- If $\log b - \log a \geq -20$: return

$$\log a + \log(1 + \exp(\log b - \log a))$$

# Justification of the Trick

$$\log(a+b) = \log\left(a\left(1+\frac{b}{a}\right)\right)$$
$$= \log(a) + \log(1 + \exp(\log b - \log a))$$

- Even if $\exp(\log a)$ and $\exp(\log b)$ underflow to zero, $\exp(\log b - \log a)$ does not.

$$\log a = -99999$$
$$\log b = -100000$$
$$\log b - \log a = -1$$

# Debugging

- How do you debug the forward/Viterbi algorithm?
- The (only) surest check:
  1. Generate a small synthetic HMM, say with $|V| = 10, |L| = 5$.
  2. Generate a short random sentence, say length 7.
  3. **Brute-force**: enumerate all $5^7$ possible sequences for exact marginalization and inference.
  4. Run your forward/Viterbi.
  5. Make sure 4 is precisely the same as 3.
  6. Repeat 2–5 many times.

# Overview

# Heads-Up

- We will now talk about an extremely general technique called **beam search**.
  - Applicable to many models other than HMMs

- Possibly the most practical trick in NLP you'll learn in this course

# Score Function Under an HMM

- Given a fixed input sequence $x = (x_1 \dots x_m)$, an HMM defines the "score" of a candidate sequence $y = (y_1 \dots y_m)$ as

$$\text{score}_x(y) = \prod_{i=1}^{m} \text{score}_x(y_i | y_1 \dots y_{i-1})$$

where each local score is **restricted** to only depend on the previous label $y_{i-1}$ and current input $x_i$.

$$\text{score}_x(y_i | y_1 \dots y_{i-1}) := t(y_i | y_{i-1}) \times o(x_i | y_i)$$

# Score Function Under an HMM

- Given a fixed input sequence $x = (x_1 \ldots x_m)$, an HMM defines the "score" of a candidate sequence $y = (y_1 \ldots y_m)$ as

$$\text{score}_x(y) = \prod_{i=1}^{m} \text{score}_x(y_i | y_1 \ldots y_{i-1})$$

where each local score is **restricted** to only depend on the previous label $y_{i-1}$ and current input $x_i$.

$$\text{score}_x(y_i | y_1 \ldots y_{i-1}) := t(y_i | y_{i-1}) \times o(x_i | y_i)$$

- **With this restriction**, we can efficiently and exactly compute

$$\arg\max_{y_1 \ldots y_m} \text{score}(y_1 \ldots y_m) \qquad \text{(Viterbi)}$$

$$\sum_{y_1 \ldots y_m} \text{score}(y_1 \ldots y_m) \qquad \text{(forward)}$$

# General Score Function

- Now suppose we have a local score that can depend arbitrarily on **all previous labels** $y_1 \dots y_{i-1}$:

$$\text{score}_x(y_i | y_1 \dots y_{i-1}) = f(x_1 \dots x_m, y_1 \dots y_{i-1})$$

# General Score Function

- Now suppose we have a local score that can depend arbitrarily on **all previous labels** $y_1 \ldots y_{i-1}$:

$$\text{score}_x(y_i|y_1 \ldots y_{i-1}) = f(x_1 \ldots x_m, y_1 \ldots y_{i-1})$$

- Without any Markov assumption, we can't hope to do inference/marginalization efficiently and exactly.

# General Score Function

- Now suppose we have a local score that can depend arbitrarily on **all previous labels** $y_1 \dots y_{i-1}$:

$$\text{score}_x(y_i|y_1 \dots y_{i-1}) = f(x_1 \dots x_m, y_1 \dots y_{i-1})$$

- Without any Markov assumption, we can't hope to do inference/marginalization efficiently and exactly.
- But we can **approximate** it.

# Beam Search

- A hack to approximate a **set** of top-$K$ candidate sequences

$$\mathcal{B} \approx \underset{y_1 \ldots y_m}{\text{K-argmax}} \ \mathsf{score}_x(y_1 \ldots y_m)$$

for **any** score function of the form

$$\mathsf{score}_x(y) = \prod_{i=1}^{m} \mathsf{score}_x(y_i | y_1 \ldots y_{i-1})$$

# Uses of the Beam Search

- The best sequence can be approximated as

$$\arg\max_{(y_1 \ldots y_m) \in \mathcal{B}} \ \mathsf{score}(y_1 \ldots y_m)$$

- The total score of all sequences can be approximated as

$$\sum_{(y_1 \ldots y_m) \in \mathcal{B}} \mathsf{score}(y_1 \ldots y_m)$$

# Idea

- Maintain a "beam" $\mathcal{B}_i$ at each time step $i = 1 \ldots m$ where

$$\mathcal{B}_i \approx \underset{y_1 \ldots y_i}{\text{K-argmax}} \; \textsf{score}_x(y_1 \ldots y_i)$$

# Beam Search Algorithm

- Base case ($i = 1$):

$$\mathcal{B}_1 = \underset{y \in L}{\text{K-argmax}} \; \textsf{score}_x(y)$$

- Main body ($i > 1$):

$$\mathcal{B}_i = \underset{\substack{(y_1 \ldots y_{i-1}) \in \mathcal{B}_{i-1} \\ y_i \in L}}{\text{K-argmax}} \; \textsf{score}_x(y_1 \ldots y_{i-1}) \times \textsf{score}_x(y_i | y_1 \ldots y_{i-1})$$

# Leaky Priority Queue

- A "leaky" priority queue $q$ with capacity $K$
- Accepts a stream of elements $[\texttt{thing}, \texttt{score}]$ but maintains only $K$ elements with the highest scores seen so far.
- Both push and pop: $O(\log K)$ worst-case time complexity
- Assume a $O(K \log K)$ operation `dump`:

$$q.\texttt{dump}() = [q.\texttt{pop}() \text{ for } K \text{ times}]$$

- Exercise: try implementing it with a standard priority queue.

## Implementation

- $q \leftarrow \texttt{leaky\_priority\_queue}(K)$
- $q.\texttt{push}([y_1, \text{score}_x(y_1)]) \qquad \forall y_1 \in L$
- For $i = 2 \ldots m$:
  - $\mathcal{B}_{i-1} \leftarrow q.\texttt{dump}()$
  - For $(y, s) \in \mathcal{B}_{i-1}$:

$$q.\texttt{push}([y.\text{append}(y_i), s \times \text{score}_x(y_i|y)]) \qquad \forall y_i \in L$$

- Return $q.\texttt{dump}()$.

# Implementation

- $q \leftarrow \texttt{leaky\_priority\_queue}(K)$
- $q.\texttt{push}([y_1, \mathsf{score}_x(y_1)]) \qquad \forall y_1 \in L$
- For $i = 2 \ldots m$:
  - $\mathcal{B}_{i-1} \leftarrow q.\texttt{dump}()$
  - For $(y, s) \in \mathcal{B}_{i-1}$:

$$q.\texttt{push}([y.\mathsf{append}(y_i), s \times \mathsf{score}_x(y_i|y)]) \qquad \forall y_i \in L$$

- Return $q.\texttt{dump}()$.

Runtime complexity: $O(|L|\, K \log K m)$

Compare with first-order HMM's forward/Viterbi: $O(|L|^2\, m)$

# Parting Remarks

- HMMs are important: master these concepts.
- Computation over **structured objects** (sequences)
  - Arguably the most distinguishing aspect of NLP as a field
- We will revisit many of the same ideas in parsing (trees).