Reminder: class web links and infrastructure

- Check in with our web page (http://j.ee.washington.edu/~bilmes/classes/ee596a_winter_2013/) for up to date announcements, homeworks, etc.
- All homeworks will be due via our dropbox (https://catalyst.uw.edu/collectit/dropbox/bilmes/25379)
- All questions should be posted to our discussion board (https://catalyst.uw.edu/gopost/board/bilmes/31332)
- You can contact me anonymously if you wish via anonymous email (https://catalyst.uw.edu/umail/form/bilmes/4144)
Announcements, Assignments, and Reminders

- Recall, HW was due last Monday, 1/21. Last problem required ideas of elimination, OK to turn in/update this problem this evening (but no later).
- Extra office hour today, 3:45-4:45pm.
- Will still either have video lecture makeup class, or actual lecture makeup. Stay tuned to web page/email for announcements relevant to this.

Relevant Readings: Readings/handouts

- Readings are in a sub-directory “reading_drafts” directly below our web page (http://j.ee.washington.edu/~bilmes/classes/ee596a_winter_2013/).
- uid is this class name (lower case) and pwd are the quarter/year of the class.
- Note, the PDF file is password protected. Send me email if you have trouble (adobe reader should have no problems reading it).
Cumulative Outstanding Reading

- Read 8.1 - 8.3 in “doc.pdf”
- Read HMM section in readings.

What HMMs can do: review so far

- Observations are not i.i.d..
- Observations are not Viterbi i.i.d..
- For homogeneous case, HMM stationarity condition is dependent on underlying Markov chain stationarity condition.
- Graphical model for GM HMM.
Thus, \( \text{cov}(X_t, X_{t+h}) \) is in general not equal to zero.

But recall, \( A^h \xrightarrow{h} 1\pi^\top \) from lecture 3, and this is a matrix with all rows equal to the stationary distribution.

Therefore, we have

\[
\text{cov}(X_t, X_{t+h}) = \sum_{ij} \mu_i \mu_j (A^h)_{ij} \pi_i - \left( \sum_i \mu_i \pi_i \right) \left( \sum_i \mu_i \pi_i \right)' \tag{7.13}
\]

\[
\xrightarrow{h} \sum_{ij} \mu_i \mu_j' \pi_j \pi_i - \left( \sum_i \mu_i \pi_i \right) \left( \sum_i \mu_i \pi_i \right)' = 0 \tag{7.14}
\]

Thus, while the covariance between two observations is not necessarily zero in an HMM, once we are at a stationary distribution, this covariance goes to zero exponentially fast.

**Exercise:** For this property to hold, must we assume a unique stationary distribution here?

Example of the decay in the mutual-information correlation from a real-world HMM. I.e., we see \( f(\tau) = I(X_t; X_{t+\tau}) \), where \( I() \) is the mutual information function.

Mutual information is stronger than correlation.

This is compared against i.i.d. samples (high peak at \( \tau = 0 \) is expected).
State Duration Modeling

- If we have multiple parallel states in series, all of which share the same observation distribution, we can construct much more interesting (multimodal) distributions.
- For example, the following left chain will have distribution as shown on the right (a mixture of negative binomial distributions).

Other examples: very long chains, ladders, fixed-length distributions (histograms), and so on.

HMMs can have flexible distributions, cost of extra states.

HMMs Generative Accuracy

Theorem 7.2.1 (Necessary conditions for generative HMM accuracy.)

An HMM with joint observation distribution \( p_h(X_{1:T}) \) will accurately model the true distribution \( p(X_{1:T}) \) only if the following three conditions hold for all \( t \):

- \( I_h(X_{\neg t}; Q_t) \geq I(X_t; X_{\neg t}) \),
- \( I_h(Q_t; X_t) \geq I(X_t; X_{\neg t}) \), and
- \( |D_Q| \geq 2I(X_t; X_{\neg t}) \)

where \( I_h(X_{\neg t}; Q_t) \) (resp. \( I_h(Q_t; X_t) \)) is the information transmission rate between \( X_{\neg t} \) and \( Q_t \) (resp. \( Q_t \) and \( X_t \)) under an HMM, and \( I(X_t; X_{\neg t}) \) is the true information transmission rate between \( I(X_t; X_{\neg t}) \).
Suff conditions for HMMs Generative Accuracy

Theorem 7.3.1

Sufficient conditions for HMM accuracy. An HMM $p_h(X_{1:T})$ will accurately represent a true discrete distribution $p(X_{1:T})$ if the following conditions hold for all $t$:

- $H(Q_t|X_{<t}) = 0$
- $p_h(X_t = x_t|q_{x_{<t}}) = p(X_t = x_t|X_{<t} = x_{<t})$.

where $q_{x_{<t}} = f(x_{<t})$ is the unique state sub-sequence associated with $x_{<t}$.

- Quite strong and unrealistic requirements, but they guarantee accuracy nonetheless.
- Note $\{< t\} \Delta = \{1, 2, \ldots, t - 1\}$

Proof: Suff conds for HMMs Generative Accuracy

Proof.

We have for all $t$:

$$D(p(X_t|X_{<t})||p_h(X_t|X_{<t}))$$

$$= \sum_{x_{1:t}} p(x_{1:t}) \log \frac{p(x_t|x_{<t})}{p_h(x_t|x_{<t})}$$

$$= \sum_{x_{1:t}} p(x_{1:t}) \log \frac{p(x_t|x_{<t})}{\sum_{q_t} p_h(x_t|q_t)p_h(q_t|x_{<t})}$$

$$= \sum_{x_{1:t}} p(x_{1:t}) \log \frac{p(x_t|x_{<t})}{p_h(x_t|q_{x_{<t}})}$$

$$= \sum_{x_{1:t}} p(x_{1:t}) \log \frac{p(x_t|x_{<t})}{p(x_t|x_{<t})}$$

$$= 0$$
Cont. Proof: Suff conds for HMMs Generative Accuracy

It then follows, using the above equation, that:

\[
0 = \sum_t D(p(X_t|X_{<t}) || p_h(X_t|X_{<t}))
\]  
(7.7)

\[
= \sum_t \sum_{x_{1:t}} p(x_{1:t}) \log \frac{p(x_t|x_{<t})}{p_h(x_t|x_{<t})} = \sum_t \sum_{x_{1:T}} p(x_{1:T}) \log \frac{p(x_t|x_{<t})}{p_h(x_t|x_{<t})}
\]  
(7.8)

\[
= \sum_{x_{1:T}} p(x_{1:T}) \log \frac{\prod_t p(x_t|x_{<t})}{\prod_t p_h(x_t|x_{<t})} = \sum_{x_{1:T}} p(x_{1:T}) \log \frac{p(x_{1:T})}{p_h(x_{1:T})}
\]  
(7.9)

\[
= D(p(X_{1:T}) || p_h(X_{1:T}))
\]  
(7.10)

- Strong conditions \( H(Q_t|X_{<t}) = 0 \), not likely to happen in practice.
- Is this really what we need for an HMM, generative accuracy? We’ll address this again soon.

Mealy vs. Moore Machines

- Mealy vs. Moore finite state automata:
  - Moore machine has only one possible output for each state — the output is a function only of the current state.
  - Mealy machine has only one possible output for each state-input pair — given current state, the input determines both the next state and the current output.
  - Mealy machine: given a current state, and a next state (determined by input), there is an output.
  - Hence, we can think of a Mealy machine as one where the output is a function of the transition between two states.
Mealy vs. Moore HMMs

- Mealy (or Jelinek) HMMs are ones where the observation distribution is a function of the transition between states.
- Markov transition diagram often shown with symbols on the edges between state nodes (note, this is not a graphical model, rather a labeled state-transition diagram).

HMMs can do

- Moore (or Rabiner) HMMs are ones where the observation is a function of the state itself.
- Markov transition diagram often shown with symbols on the nodes of the diagram.

Two ways to show the pronunciation model for the word “and”

Left: Moore machine, Right: Mealy machine.

The output label on each Mealy arcs is shown as input:output and ε means “null”
Mealy vs. Moore HMMs

- Graphical model for Moore-HMM: as we have seen it, for each , is a parent of for each .

- Graphical model for Mealy-HMM: has two parents, either or .

Mealy and Moore HMMs are equivalent in that they correspond to the same class of probability distributions.

- Mealy → Moore conversion: cluster adjacent variables in the graphical model.
- More → Mealy conversion: Use a separate arc for each possible output with observation randomness applied to the arc.
- It is sometimes more efficient, in terms of number of states, to specify a language in terms of a Mealy HMM (see parenthesis matching example in the readings).
Distributive Law and Other Objects

- Only one property needed for this algorithm to work, namely distributive law $ab + ac = a(b + c)$ along with factorization.
- Distributive law allows sending sums inside of factors.
- Nothing unique about the algebraic operations of “sum” and “product” for this to work.
- Other objects and their algebraic operations have a distribute law, and in general any set of objects that is a commutative semiring will work as well.

Commutative Semirings

Definition 7.4.1

A commutative semiring is a set $K$ with two binary operators $\oplus$ and $\otimes$ having three axioms, for all $a, b, c \in K$.

$S1$: $\oplus$ is commutative $(a \oplus b) = (b \oplus a)$ and associative $(a \oplus b) \oplus c = a \oplus (b \oplus c)$, and $\exists$ a unique (“additive”) identity called $\tilde{0} \in K$ such that $k \oplus \tilde{0} = k$ for all $k \in K$. I.e., $(K, \oplus)$ is a commutative monoid.

$S2$: $\otimes$ is also associative, commutative, and $\exists$ a unique (“multiplicative”) identity called $\tilde{1} \in K$ s.t. $k \otimes \tilde{1} = k$ for all $k \in K$ ($(K, \otimes)$ is also a commutative monoid).

$S3$: the distributive law holds: $(a \otimes b) \oplus (a \otimes c) = a \otimes (b \oplus c)$ for all $a, b, c \in K$.

This, and factorization w.r.t. a graph $G$ is all that is necessary for the message passing algorithms to work. There are many commutative semirings.
Commutative Semirings

- Additive (i.e., $\oplus$-based) inverse need not exist. If additive inverse exists, then we get a commutative ring (“semi-ring” since we need not have additive inverse). Note, in algebra texts, a ring often doesn’t require multiplicative identity, but we assume it exists here.
- Above definition does not mention $0 \otimes k = 0$, but this follows from above properties since $k \otimes k = k \otimes (k \oplus 0) = (k \otimes k) \oplus (k \otimes 0)$ so that $k \otimes 0$ must also be an additive identity, meaning that $k \otimes 0 = 0$.
- This is useful with evidence with delta functions, where the delta functions multiplies by zero anything that does not abide by the evidence value.
- Same message passing protocol and message passing scheme on a junction tree will work to ensure that all clusters reach a state where they are the appropriate “marginals”
- Marginals in this case dependent on ring.

Other Semi-Rings

Here, $A$ denotes arbitrary commutative semiring, $S$ is arbitrary finite set, $\Lambda$ is arbitrary distributed lattice.

<table>
<thead>
<tr>
<th></th>
<th>$K$</th>
<th>&quot;($\oplus$, 0)&quot;</th>
<th>&quot;($\otimes$, 1)&quot;</th>
<th>short name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$A$</td>
<td>(+, 0)</td>
<td>(\cdot, 1)</td>
<td>semiring</td>
</tr>
<tr>
<td>2</td>
<td>$A[x]$</td>
<td>(+, 0)</td>
<td>(\cdot, 1)</td>
<td>polynomial</td>
</tr>
<tr>
<td>3</td>
<td>$A[x, y, \ldots]$</td>
<td>(+, 0)</td>
<td>(\cdot, 1)</td>
<td>polynomial</td>
</tr>
<tr>
<td>4</td>
<td>(0, $\infty$)</td>
<td>(+, 0)</td>
<td>(\cdot, 1)</td>
<td>sum-product</td>
</tr>
<tr>
<td>5</td>
<td>(0, $\infty$)</td>
<td>(min, $\infty$)</td>
<td>(\cdot, 1)</td>
<td>min-product</td>
</tr>
<tr>
<td>6</td>
<td>(0, $\infty$)</td>
<td>(max, 0)</td>
<td>(\cdot, 1)</td>
<td>max-product</td>
</tr>
<tr>
<td>7</td>
<td>(0, $\infty$)</td>
<td>(kmax, 0)</td>
<td>(\cdot, 1)</td>
<td>k-max-product</td>
</tr>
<tr>
<td>8</td>
<td>(−$\infty$, $\infty$)</td>
<td>(min, $\infty$)</td>
<td>(+, 0)</td>
<td>min-sum/tropical</td>
</tr>
<tr>
<td>9</td>
<td>(−$\infty$, $\infty$)</td>
<td>(max, −$\infty$)</td>
<td>(+, 0)</td>
<td>max-sum</td>
</tr>
<tr>
<td>10</td>
<td>(−$\infty$, $\infty$)</td>
<td>(LOG-ADD, $\infty$)</td>
<td>(\cdot, 0)</td>
<td>log</td>
</tr>
<tr>
<td>11</td>
<td>{0, 1}</td>
<td>(OR, 0)</td>
<td>(AND, 1)</td>
<td>Boolean</td>
</tr>
<tr>
<td>12</td>
<td>$2^S$</td>
<td>(\cup, 0)</td>
<td>(\cap, S)</td>
<td>Set</td>
</tr>
<tr>
<td>13</td>
<td>$\Lambda$</td>
<td>(\vee, 0)</td>
<td>(\wedge, 1)</td>
<td>Lattice</td>
</tr>
<tr>
<td>14</td>
<td>$\Lambda$</td>
<td>(\wedge, 1)</td>
<td>(\vee, 0)</td>
<td>Lattice</td>
</tr>
</tbody>
</table>
Example: Viterbi/MPE

- Most-probable explanation (e.g., Viterbi assignment) is just the max-product ring.
- Here, we wish to compute

\[
\arg\max_{x_{V\setminus E}} p(x_{V\setminus E}, \bar{x}_E)
\]  

(7.11)

- After message passing with the max-product ring on a junction tree, cluster functions will reach the "max-marginal" state, where we have:

\[
\psi_C(x_C) = \max_{x_{V\setminus C}} p(x_C, x_{V\setminus C})
\]

(7.12)

- What about a "k-max" operation (i.e., finding the k highest scoring assignments to the variables?) How would we define the operators "⊕" and "⊗"?

\[k\]-best

- We know Viterbi-decoding/MPE for HMMs.

\[
q^{*}_{1:T}^{(1)} \in \arg\max_{q_{1:T} \in D_{Q_{1:T}}} p(\bar{x}_{1:T}, q_{1:T})
\]

(7.13)

- What if we want the second best. I.e.,

\[
q^{*}_{1:T}^{(2)} \in \arg\max_{q_{1:T} \in D_{Q_{1:T}}, q_{1:T} \neq q^{*}_{1:T}^{(1)}} p(\bar{x}_{1:T}, q_{1:T})
\]

(7.14)

- and the third best

\[
q^{*}_{1:T}^{(3)} \in \arg\max_{q_{1:T} \in D_{Q_{1:T}}, q_{1:T} \notin \{q^{*}_{1:T}^{(1)}, q^{*}_{1:T}^{(2)}\}} p(\bar{x}_{1:T}, q_{1:T})
\]

(7.15)

- And the \(k\)th best, \(q^{*}_{1:T}^{(k)}\).
Let $\mathbb{R}_+^k$ be the set of (descending-order) sorted $k$-tuples of non-negative real numbers.

If $s \in \mathbb{R}_+^k$, then

$$s = (s^1, s^2, \ldots, s^k)$$

(7.16)

with $s^1 \geq s^2 \geq \cdots \geq s^k$. Let's call it an “sk-tuple”.

- Notation: denote $s^i$ to indicate an element of a sorted $k$-tuple $s$, use subscripts, such as $s_1, s_2$ to indicate particular $k$-tuples.
- All $k$-tuples will be assumed to be sorted in descending order.
Operations on Sorted $k$-tuples

- Given $s_1, s_2 \in \mathbb{R}^k_+$, define an operation to merging, re-sorting, and truncate so that result is in $\mathbb{R}^k_+$.
- That is we first merge:

$$\text{merge}(s_1, s_2) = (s_1^1, s_1^2, \ldots, s_1^k, s_2^1, s_2^2, \ldots, s_2^k) \quad (7.17)$$

- Then sort:

$$\text{sort}(\text{merge}(s_1, s_2)) = (s^1, s^2, \ldots, s^{2k}) \quad (7.18)$$

where $s^1 \geq s^2 \geq \ldots s^{2k}$.
- Then truncate, such that

$$k\text{-truncate}(\text{sort}(\text{merge}(s_1, s_2))) = (s^1, s^2, \ldots, s^k) = s \quad (7.19)$$

which just takes the first $k$ items in the length $2k$ tuple.

- Hence, if $k = 1$, then $s_1 \oplus s_2 = s$ with $s^1 = \max(s_1^1, s_2^1)$. So this is just the max operator.
Examples: $k = 1$

For example, when $k = 1$, we have that:

\[(1) \oplus (3) = (3)\]  \hspace{1cm} (7.22)
\[(2) \oplus (3) = (3)\]  \hspace{1cm} (7.23)
\[(0) \oplus (1) = (1)\]  \hspace{1cm} (7.24)
\[(0) \oplus (0) = (0)\]  \hspace{1cm} (7.25)

Examples: $k = 3$

As another example, when $k = 3$, we have

\[(3, 2, 1) \oplus (4, 2, 1) = (4, 3, 2)\]  \hspace{1cm} (7.26)
\[(3, 2, 1) \oplus (3, 2, 1) = (3, 3, 2)\]  \hspace{1cm} (7.27)
\[(0, 0, 0) \oplus (3, 2, 1) = (3, 2, 1)\]  \hspace{1cm} (7.28)
\[(1, 0, 0) \oplus (3, 2, 1) = (3, 2, 1)\]  \hspace{1cm} (7.29)
\[(10, 9, 0) \oplus (3, 2, 1) = (10, 9, 3)\]  \hspace{1cm} (7.30)
\[(10, 0, 0) \oplus (3, 0, 0) = (10, 3, 0)\]  \hspace{1cm} (7.31)
\[(0, 0, 0) \oplus (1, 0, 0) = (1, 0, 0)\]  \hspace{1cm} (7.32)
Scalar conversion

- operation of converting a non-negative scalar to a sorted $k$-tuple
- For $r \in R_+$, which is the set of non-negative reals, we have
  \[
  \overrightarrow{R}_+^k(r) = (r, 0, 0, \ldots, 0) \tag{7.33}
  \]
  where there are $k - 1$ zeros following the $r$.
- Thus, any scalar value is considered a $k$-tuple with potentially only one non-zero entry.
- E.g., the value 3 is represented as $(3, 0, 0)$.
- Thus, we can repeatedly “insert” elements $r_1, r_2, \ldots, r_\ell \in R_+$ (not necessarily sorted) into a $k$-tuple by forming:
  \[
  \overrightarrow{R}_+^k(r_1) \oplus \overrightarrow{R}_+^k(r_2) \oplus \cdots \oplus \overrightarrow{R}_+^k(r_\ell) = (r_{\sigma_1}, r_{\sigma_2}, \ldots, r_{\sigma_k}) \tag{7.34}
  \]
  where $r_{\sigma_1} \geq r_{\sigma_2} \geq \cdots \geq r_{\sigma_k}$ and $r_{\sigma_k}$ is not strictly less than any of the elements that may have been truncated from $r_1, r_2, \ldots, r_\ell$.

Properties

- Note that this operation $\oplus$ has a number of properties.
- For any $s_1, s_2, s_3 \in \overrightarrow{R}_+^k$, we have that
  \[
  s_1 \oplus s_2 \in \overrightarrow{R}_+^k \quad \text{(closure)} \tag{7.35}
  \]
  \[
  s_1 \oplus s_2 = s_2 \oplus s_1 \quad \text{(commutativity)} \tag{7.36}
  \]
  \[
  (s_1 \oplus s_2) \oplus s_3 = s_1 \oplus (s_2 \oplus s_3) \quad \text{(associativity)} \tag{7.37}
  \]
- Also, there exists an additive identity, call it $\emptyset$, such that
  \[
  s_1 \oplus \emptyset = s_1 \tag{7.38}
  \]
  with $\emptyset = (0, 0, \ldots, 0)$.
- Therefore, $(\overrightarrow{R}_+^k, \oplus)$ itself forms a **commutative semi-group**. It is a semi-group rather than a group since we are not requiring an “additive” inverse.
Repeated operations

- Given a set of elements $s_1, s_2, \ldots, s_\ell \in \mathbb{R}_+^k$, we can repeatedly compute $\oplus$ as
  \[
  \bigoplus_{i=1}^{\ell} s_i = s = (s^1, s^2, \ldots, s^k)
  \]  
  (7.39)
- Then, $(s^1, s^2, \ldots, s^k)$ is the $k$-tuple containing the $k$ largest elements in the union of all of the elements in $s_1, s_2, \ldots, s_\ell$.
- This operation is well defined due to the above closure and associativity properties.
- Operation remains in $\mathbb{R}_+^k$, and the order doesn’t matter.

Scalar multiplication

- Given $r \in \mathbb{R}_+$, and $s \in \mathbb{R}_+^k$, define the scalar left-multiplication as an operation $\mathbb{R}_+ \times \mathbb{R}_+^k \rightarrow \mathbb{R}_+^k$ called as follows:
  \[
  r \cdot s = (rs^1, rs^2, \ldots, rs^k)
  \]  
  (7.40)
- I.e., we scalar multiply each element in $s$ by $r$, and since $r \geq 0$, the order does not change.
- When clear from the context, we drop the $\cdot$ notation as is done in standard scalar-vector multiplication, so that $r \cdot s = rs$. 
Vector (tuple) addition

- For $s_1, s_2 \in \mathbb{R}_+^k$, we also define standard vector addition as $s = (s^1, s^2, \ldots, s^k) = s_1 + s_2$ where $s^i = s^i_1 + s^i_2$.
- Since input operands are both sorted, so is result. I.e., since $s^1_1 \geq s^1_2$ and $s^2_1 \geq s^2_2$ we have $s^1 = s^1_1 + s^1_2 \geq s^1_2 + s^2_2 = s^2$.
- Note that $s_1 + s_2$ using the “+” operator is standard vector addition, while $s_1 \oplus s_2$ is the merge, sort, and truncate operation we mentioned above.

Properties of scalar mult

- Note that for all $r_1, r_2 \in \mathbb{R}_+$, and $s_1, s_2 \in \mathbb{R}_+^k$, we have that
  \[
  r_1 (s_1 \oplus s_2) = r_1 s_1 \oplus r_1 s_2 \quad \text{distributive property \ (7.41)}
  \]
  \[
  (r_1 + r_2) s_1 = (r_1 s_1) + (r_2 s_1) \quad \text{distributive property \ (7.42)}
  \]
  \[
  (r_1 r_2) s_1 = r_1 (r_2 s_1) \quad \text{associativity \ (7.43)}
  \]
Reminder - Viterbi Path

The following slide is from lecture 5.

MPE/Viterbi path - summary

Forward Equations

\[ \alpha_q^m(1) = p(\bar{x}_1|Q_1 = q) \] (7.21)

\[ \alpha_q^m(t) = p(\bar{x}_t|Q_t = q) \max_r p(Q_t = q|Q_{t-1} = r) \alpha_r^m(t-1) \] (7.22)

And the forward equation for storing the back indices:

\[ \hat{\alpha}_q^m(t) \in \arg\max_r p(Q_t = q|Q_{t-1} = r) \alpha_r^m(t-1) \] (7.23)

Backward algorithm, to compute the Viterbi path

1. Compute \( q_T^* \in \arg\max_q \alpha_q^m(T) \)
2. for \( t = T \ldots 2 \) do
3. \[ \text{Set } q_{t-1}^* \leftarrow \hat{\alpha}_{q_t}^m(t) \]
**k-best paths**

- Viterbi path is:
  \[
  q_{1:T}^* \in \operatorname{argmax}_{q_{1:T}} p(\bar{x}_{1:T}, q_{1:T})
  \]  
  \(7.44\)

- Our goal is to find a set of distinct assignments
  \[
  q_{1:T}^*(1), q_{1:T}^*(2), \ldots, q_{1:T}^*(k)
  \]  
  \(7.45\)

with

\[
p(\bar{x}_{1:T}, q_{1:T}^*(1)) \geq p(\bar{x}_{1:T}, q_{1:T}^*(2)) \geq \cdots \geq p(\bar{x}_{1:T}, q_{1:T}^*(k))
\]  
\(7.46\)

and any other assignments not mentioned have score no higher than 
\(p(\bar{x}_{1:T}, q_{1:T}^*(k))\).

- Each of \(p(\bar{x}_{1:T}, q_{1:T}^*(i))\) is scalar value in \(\mathbb{R}_+\), so therefore, we have that

\[
\bigoplus_{q_{1:T} \in D_{Q_{1:T}}} \mathbb{R}_+^k (p(\bar{x}_{1:T}, q_{1:T})) = (p(\bar{x}_{1:T}, q_{1:T}^*(1)), p(\bar{x}_{1:T}, q_{1:T}^*(2)), \ldots, p(\bar{x}_{1:T}, q_{1:T}^*(k)))
\]  
\(7.47\)

For simplicity, we will assume that a scalar \(r \in \mathbb{R}_+\) will automatically graduate from a scalar into a \(k\)-tuple via the operator \(\oplus\), so that \(r_1 \oplus r_2 \equiv \mathbb{R}_+^k (r_1) \oplus \mathbb{R}_+^k (r_2)\).

- If \(k = 1\), then we have Viterbi (MPE) path value. That is, we have given an exponential calculation that expresses the Viterbi path value.

- If \(k > 1\) we have an exponential computation that expresses a vector consisting of the \(k\)-best path values.

- In either case there is much optimization we can do thanks to the aforementioned properties.
 Optimization via dist. property

Because of the distributed property above, we have that

\[
\bigoplus_{q_1:T \in D_{Q_1:T}} p(\bar{x}_{1:T}, q_{1:T})
\]

(7.48)

\[
= \bigoplus_{q_1:T \in D_{Q_1:T}} \prod_{t} p(x_t|q_t)p(q_t|q_{t-1})
\]

(7.49)

\[
= \bigoplus_{q_T} p(x_T|q_T) p(q_T|q_{T-1}) \ldots \left( \bigoplus_{q_2} p(x_2|q_2)p(q_3|q_2) \left( \bigoplus_{q_1} p(x_1|q_1)p(q_2|q_1) \right) \right)
\]

(7.50)

\[
= \bigoplus_{q_T} p(q_T|q_{T-1}) \ldots \left( p(x_3|q_3) \bigoplus_{q_2} p(q_3|q_2) \left( p(x_2|q_2) \bigoplus_{q_1} p(x_1|q_1)p(q_2|q_1) \right) \right)
\]

(7.51)

Note:
\[
\alpha_t^{(k)}(q) \in \mathbb{R}^k_+ \text{ is a } k\text{-tuple for all } q, t.
\]

Hence, for each \((t, q)\), need a tuple of size \(k\) to store the \(k\) values, unlike standard \((k = 1)\) Viterbi (a 3D tensor).
Max marginal interpretation

- In Viterbi case, we have max marginal interpretation:
  \[
  \alpha_t^m(q_t) = \max_{q_1:t-1} p(q_1:t-1, q_t, \bar{x}_{1:t})
  \]  
  \[ (7.54) \]

- Similar thing can be done here.

- Suppose:
  \[
  \alpha_t^{(k)}(q_t) = (p(\bar{x}_{1:t}, q_{1:t-1}(1), q_t), p(\bar{x}_{1:t}, q_{1:t-1}(2), q_t), \ldots, p(\bar{x}_{1:t}, q_{1:t-1}(k), q_t))
  \]

- then
  \[
  p(\bar{x}_{1:t}, q_{1:t-1}(1), q_t) = \max_{q_1:t-1} p(q_1:t-1, q_t, \bar{x}_{1:t})
  \]  
  \[ (7.55) \]

We also have that

\[
 p(\bar{x}_{1:t}, q_{1:t-1}(1), q_t) \geq p(\bar{x}_{1:t}, q_{1:t-1}(2), q_t) \geq \cdots \geq p(\bar{x}_{1:t}, q_{1:t-1}(k), q_t)
\]  
\[ (7.56) \]

Also, \( p(\bar{x}_{1:t}, q_{1:t-1}(k), q_t) \geq p(\bar{x}_{1:t}, q'_{1:t-1}, q_t) \) where \( q'_{1:t-1} \) is any other sequence of states up to time \( t - 1 \) that have not been included in the above.

\( \alpha_t^{(k)}(q) \) gives us the probabilities of the top \( k \) sequence of states leading up to and ending at state \( q \) at time \( t \).

This generalizes the (max margin) fact that \( \alpha_t^m(q) \) gives us the top scoring sequence of states leading up to and ending at state \( q \) at time \( t \).
**$k$-best scores**

- if we then compute
  \[
  \bigoplus_{q_T} \alpha^{(k)}_T(q_T) = (p(\bar{x}_{1:T}, q^*_1(1)), p(\bar{x}_{1:T}, q^*_1(2)), \ldots, p(\bar{x}_{1:T}, q^*_1(k)))
  \]
  (7.57)
  we'll get the desired values, i.e., the scores of the $k$-best paths through the HMM.

- But how do we get the $k$-best paths?

**arg $k$ max**

- max returns the max and $\text{argmax}$ returns the argument that achieves the max.
- $\bigoplus$ returns the max $k$ elements.
- we can generalize $\bigoplus$ as well in the same way. For
- We call this $\text{arg}\bigoplus$, pronounced “arg-k-max”.
- Note that $\text{arg}\bigoplus$ has an implicit (but not expressed) dependence on $k$.
- If we have $s_1, s_2 \in \mathbb{R}^k_+$, and $k = 1$, then
  \[
  \text{arg}\bigoplus(s_1, s_2)
  \]
  (7.58)
  should produce the index, 1 or 2, depending on which of $s_1$ and $s_2$ contain the max.
arg $k$: max

- With $k > 1$, arg$\oplus$ should produce a set of $k$ indices indicating where the $k$-largest values are.
- With $k > 1$ the values could come from either $s_1$ or $s_2$, so the indices themselves have to be a 2-tuple, stating
  1. which $k$-tuple (either $s_1$ or $s_2$) the entry lies in and
  2. where in that selected $k$-tuple the entry is.
- In other words, arg$\oplus$ returns a list of pairs of integers, where each pair is a tuple-identifier $i$ and a position $1 \leq j \leq k$ within that tuple.
- List of pairs ordered: the first pair indexes the tuple and tuple entry of the largest element in the ordered lists of numbers; the second pair indexes the tuple and tuple entry of the second largest element in the ordered lists of numbers; and so on.

For example, suppose $k = 3$, then

$$\text{arg}\oplus((3.0, 2.0, 1.0), (6.0, 5.0, 2.0)) = \{((2, 1), (2, 2), (1, 1))\} \quad (7.59)$$

Another example with $k = 4$ would be:

$$\text{arg}\oplus((4.0, 3.0, 2.1, 0.5), (7.0, 6.0, 2.0, 1.0), (5.0, 2.0, 1.0, 1.0))$$

$$= \{((2, 1), (2, 2), (3, 1), (1, 1))\} \quad (7.60)$$

$$= \{((2, 1), (2, 2), (3, 1), (1, 1))\} \quad (7.61)$$
Another example, suppose $k = 3$, then

$$\arg\oplus((3.0, 2.0, 1.0), (6.0, 5.0, 3.0))$$

$$= \{((2, 1), (2, 2), (1, 1)), ((2, 1), (2, 2), (2, 3))\}$$

Hence, we can have more than list of pairs that achieves the same max $k$ values.

thus, $\arg\oplus$ should be interpreted as a set of lists, each list in the set is a list of pairs.

Consider now the following recursion, where at each step we are computing a set of $k$ pairs of integers.

$$\tilde{\alpha}^{(k)}(q) \in \arg\oplus_{r} p(Q_t = q | Q_{t-1} = r) \alpha^{(k)}_{t-1}(r)$$

Where is the use of the observation scores $p(\bar{x}_t|q_t)$? Discuss.

Also, note the use of “$\in$”.

Collectively, $\left\{\tilde{\alpha}^{(k)}(q)\right\}_{t\in\{1,2,...,T\},q\in\{1,2,...,N\}}$ is a $|D_Q| \times T$ matrix of $k$-tuples of pairs (so $2kTN$ entries in total).

The set of pairs at $\tilde{\alpha}^{(k)}(q)$ identify the location of the $k$ largest items in the collective set of $k$-tuples in the previous time step that: 1) lead to entry $(q,t)$ in the matrix, and 2) have accounted for the transition matrix $p(q|r)$ for $r$ ranging over the first element in every pair $\tilde{\alpha}^{(k)}(q)$.
Backtracking recursion

- We can denote this list of pairs via:

\[
\tilde{\alpha}_t^{(k)}(q) = \left( (\tilde{q}^1, \tilde{k}^1), (\tilde{q}^2, \tilde{k}^2), \ldots, (\tilde{q}^k, \tilde{k}^k) \right) \tag{7.65}
\]

where \( 1 \leq \tilde{q}^i \leq |D_Q|, \ 1 \leq \tilde{k}^i \leq k \)

- denote quantities like the \( \ell \)th entry in this tuple of pairs using double arguments \( "(q)(\ell)" \) as in:

\[
(\tilde{q}, \tilde{k}) = \tilde{\alpha}_t^{(k)}(q)(\ell) \tag{7.66}
\]

where \( 1 \leq \ell \leq k, \ 1 \leq \tilde{q} \leq |D_Q|, \) and \( 1 \leq \tilde{k} \leq k \).

At the very last time step \( T \), we can then compute

\[
\tilde{\alpha}_{T}^{(k)} \in \arg\oplus_{r} \alpha_{T}^{(k)}(r) \tag{7.67}
\]

which then gives a length-\( k \) list of pairs of states and position in the \( k \)-tuple at that state.

- The states in this list correspond to the states of the \( k \) maximum paths through the HMM, and the index into the \( k \)-tuple gives the position within the \( k \)-tuple from which we can obtain the value.

- Note also that the observation scores \( (p(x_T|q) \) over \( q \) at time \( T \) are finally (implicitly) used here.
Recall again standard Viterbi backtracking from lecture 5, on next slide:

Forward Equations

\[
\alpha^m_q(1) = p(\bar{x}_1|Q_1 = q) \tag{7.21}
\]
\[
\alpha^m_q(t) = p(\bar{x}_t|Q_t = q) \max_r p(Q_t = q|Q_{t-1} = r)\alpha^m_r(t-1) \tag{7.22}
\]

And the forward equation for storing the back indices:

\[
\tilde{\alpha}^m_q(t) \in \arg\max_r p(Q_t = q|Q_{t-1} = r)(t - 1) \tag{7.23}
\]

Backward algorithm, to compute the Viterbi path

1. Compute \( q^*_T \in \arg\max_q \alpha^m_q(T) \)
2. for \( t = T \ldots 2 \) do
3. \( q^*_{t-1} \leftarrow \tilde{\alpha}^m_{q^*_t}(t) \)
$k$-best Backtracking

- At the final $T$ step, we need to find the $k$ top scoring positions, like above.

$$
\left( (q_{T-1}^*, k_{T-1}^*), (q_T^*, k_T^*)^2, \ldots, (q_T^*, k_T^*)^k \right) = \tilde{\alpha}_T^{(k)}(r) = \text{arg} \oplus \alpha_T^{(k)}(r)
$$

(7.68)

- Once we have these $k$ state-element pairs, we look up the index at each of those $k$ state-element pairs, to get a new set of $k$ state-element pairs. That is, we construct a new $k$-tuple of pairs as follows:

$$
\left( \tilde{\alpha}_T^{(k)}(q_{T-1}^*)(k_{T-1}^*), \tilde{\alpha}_T^{(k)}(q_T^*)(k_T^*)^2, \ldots, \tilde{\alpha}_T^{(k)}(q_T^*)(k_T^*)^k \right)
$$

(7.69)

Note that each entry in this $k$-tuple is still a pair, but it a pair that points to an entry in the collection of $k$-tuples at time $T - 1$.

$\tilde{\alpha}_T^{(k)}(q_{T-1}^*) = \text{arg} \oplus \alpha_{T-1}^{(k)}(r)$

(7.70)

Each entry in a $k$-tuple at time $T - 1$ also contains a pair, so the above $k$-tuple of pairs can be converted into a new tuple of pairs.

$$
\left( (q_{T-1}^*, k_{T-1}^*), (q_{T-1}^*, k_{T-1}^*), \ldots, (q_T^*, k_T^*)^k \right)
\quad = \left( \tilde{\alpha}_T^{(k)}(q_{T-1}^*)(k_{T-1}^*), \tilde{\alpha}_T^{(k)}(q_T^*)(k_T^*)^2, \ldots, \tilde{\alpha}_T^{(k)}(q_T^*)(k_T^*)^k \right)
$$

(7.70)

This can then be used to construct the $k$-tuple of pairs at time step $T - 2$ as follows:

$$
\left( (q_{T-2}^*, k_{T-2}^*), (q_{T-2}^*, k_{T-2}^*), \ldots, (q_T^*, k_T^*)^k \right)
\quad = \left( \tilde{\alpha}_T^{(k)}(q_{T-1}^*)(k_{T-1}^*), \tilde{\alpha}_T^{(k)}(q_T^*)(k_T^*)^2, \ldots, \tilde{\alpha}_T^{(k)}(q_T^*)(k_T^*)^k \right)
$$

(7.71)
What HMMs can do

Semiring

A-best

Summary

Scratch

**k-best Backtracking algorithm**

The above backtracking algorithm can be written as follows:

1. Compute \( (q_{1}^{*}, k_{T}^{*}), (q_{2}^{*}, k_{T}^{*}), \ldots, (q_{k}^{*}, k_{T}^{*}) ) = \hat{\alpha}_{T}^{(k)} \in \arg\oplus_{r} \alpha_{T}^{(k)}(r) \)
2. for \( t = T \ldots 2 \) do
3. \[
\left( \hat{\alpha}_{t}^{(k)}(q_{1}^{*}) (k_{1}^{*}), \hat{\alpha}_{t}^{(k)}(q_{2}^{*}) (k_{2}^{*}), \ldots, \hat{\alpha}_{t}^{(k)}(q_{k}^{*}) (k_{k}^{*}) \right)
\]

**Complete k-best Forward/Backward algorithm**

1. *forwards pass:*
2. for \( q = 1 \ldots |D_Q| \) do
3. \[
\alpha_{1}^{(k)}(q) \leftarrow \Pi_{t}^{k} p(\bar{x}_{1}|Q_{t} = q) ;
\]
4. for \( t = 2 \ldots T \) do
5. for \( q = 1 \ldots |D_Q| \) do
6. \[
\alpha_{t}^{(k)}(q) \leftarrow p(\bar{x}_{t}|Q_{t} = q) \oplus_{r} p(Q_{t} = q|Q_{t-1} = r) \alpha_{t-1}^{(k)}(r) ;
\]
7. \[
\hat{\alpha}_{t}^{(k)}(q) \in \arg\oplus_{r} p(Q_{t} = q|Q_{t-1} = r) \alpha_{t-1}^{(k)}(r) ;
\]
8. \[
\hat{\alpha}_{T}^{(k)}(q) \in \arg\oplus_{r} \alpha_{T}^{(k)}(r) ;
\]
9. *backwards pass:*
10. Identify \( (q_{1}^{*}, k_{T}^{*}), (q_{2}^{*}, k_{T}^{*}), \ldots, (q_{k}^{*}, k_{T}^{*}) ) = \hat{\alpha}_{T}^{(k)} ;
\]
11. for \( t = T \ldots 2 \) do
12. \[
\left( \hat{\alpha}_{t}^{(k)}(q_{1}^{*}) (k_{1}^{*}), \hat{\alpha}_{t}^{(k)}(q_{2}^{*}) (k_{2}^{*}), \ldots, \hat{\alpha}_{t}^{(k)}(q_{k}^{*}) (k_{k}^{*}) \right) \leftarrow
\]

---

Prof. Jeff Bilmes EE596A/Winter 2013/DGMs – Lecture 7 - Jan 30th, 2013 page 7-59 (of 66)
Analysis

- one forward pass of $T$ steps and one backwards pass of $T$ steps, so the compute is still linear in $T$.
- Also, at each time step $t$ we need storage for $k|D_Q|$ cells, so the memory is also still linear in $T$.
- Finding the single max entry at each step for a given state costs $O(|D_Q|)$ (leading to $O(|D_Q|^2)$ overall).
- Finding the $k$ max entries for a state can still be done in $O(|D_Q|)$ time (yes, independent of $k \leq |D_Q|$) time using a variant of the quick-sort algorithm
  - such an algorithm finds the $k$-top entries in an arbitrary order (i.e., it doesn’t find the $k$ top in sorted order).
  - Thus, to sort the top $k$ items, we pay an additional $O(k \log k)$ (but typically $k \ll |D_Q|$).
  - Therefore, the overall time-complexity is now $O(T(|D_Q|^2k \log k))$.

Analysis - memory

- at each $(t,q)$ position, we need now to store a $k$-tuple and in each $k$-tuple entry we need three values (the value and the two integers).
- Therefore, the memory has gone from $O(T|D_Q|)$ to $O(kT|D_Q|)$, which is $k$-times worse.
- there are other algorithms for computing $k$-best that do not require $k$ times the memory, but they do require $k$ times the time-cost – we cover this next time.
- Also, there are other interesting time-space tradeoffs as well with dynamic models, such as the Island algorithm as we will soon see.
Sources for Today’s Lecture

- “doc.pdf” sections 8.1 - 8.3
What HMMs can do

Semiring

k-best

Summary

Scratch

Scratch Paper