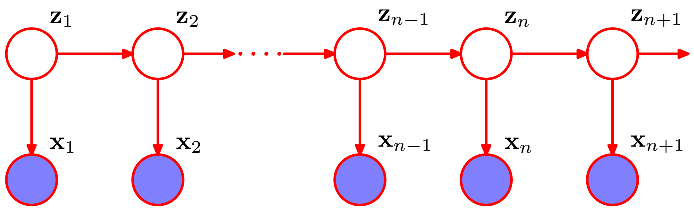
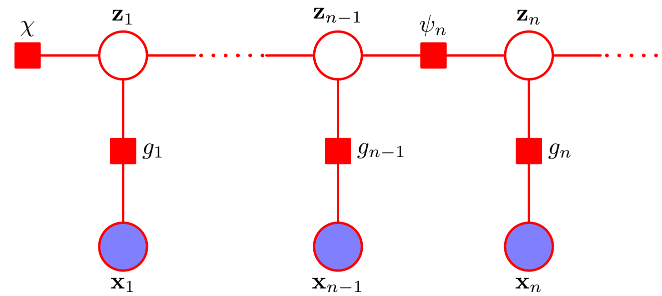
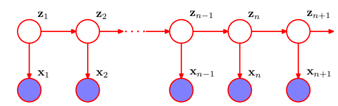
Computing marginals, version two

We can apply sum-product to our E step graph.



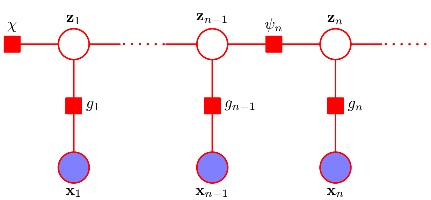
Factor graph





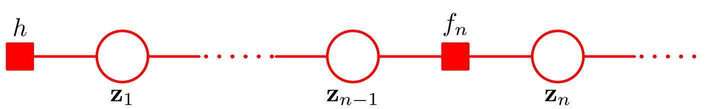
(Directed graph for reference)

Simplified factor graph

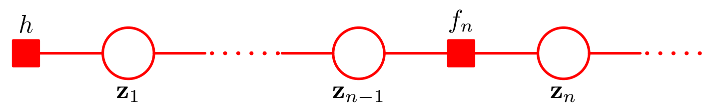


(Canonical factor graph from previous slide)

Since we condition on all the x's, we can simplify the graph by treating the emissions as constants, and putting them into the factors for the z's to get a simple chain.



Review of sum-product concepts



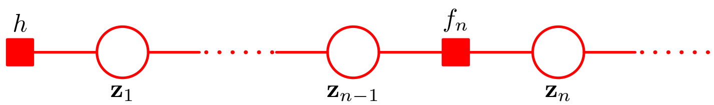
The marginal for each node is a product of the incoming messages.

This is analogous to setting up the marginal as a product of alpha and beta factors in the previous treatment.

Since we have a chain, this is just two messages, one coming from the left, the other from the right.

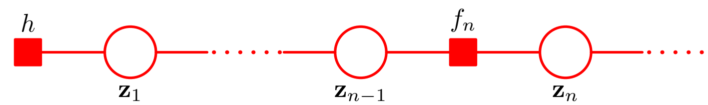
To compute all marginals, we pass the left and right messages from one end to the other.

Sum-product for HMM



$$h = p(z_1)\underbrace{p(x_1|z_1)}_{\text{extra for the nodes we pruned}}$$

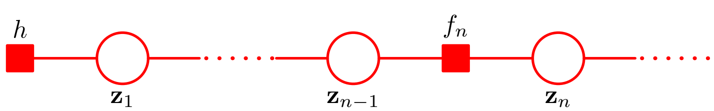
Sum-product for HMM



$$h = p(z_1)\,p(x_1|z_1)$$

$$f_n = p(z_n|z_{n-1})\underbrace{p(x_n|z_n)}_{\text{extra for pruned nodes}}$$

Sum-product for HMM

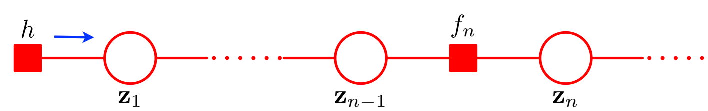


The nodes all have at most two links (it is a chain) so they just pass the incoming message to the outgoing link.

i.e., $\mu_{f_n \to z_n}(z_n) = \mu_{f_n \to f_{n+1}}(z_n)$

The nodes also (metaphorically) is where we think of the messages being stored if we are computing multiple marginals (which we are in this case).

Sum-product for HMM



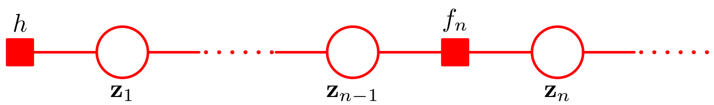
Factor node actions on left to right messages

$$\mu_{f_n \to f_{n+1}}(z_n) = \sum_{z_{n-1}} f_n(z_{n-1}, z_n)\mu_{f_{n-1} \to f_n}(z_{n-1})$$

The first message is

$$h = p(z_1)\,p(x_1|z_1) = p(x_1, z_1) = \alpha(z_1)$$

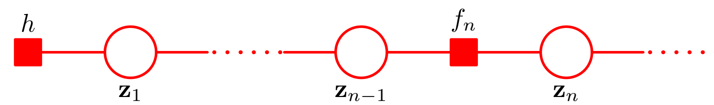
Sum-product for HMM



If we identify $\mu_{f_n \to f_{n+1}}(z_n) = \alpha(z_n)$

$$\begin{aligned}\alpha(z_n) &= \sum_{z_{n-1}} f_n(z_{n-1}, z_n)\alpha(z_{n-1}) \\ &= \sum_{z_{n-1}} p(z_n|z_{n-1})\, p(x_n|z_n)\alpha(z_{n-1}) \\ &= p(x_n|z_n)\sum_{z_{n-1}} p(z_n|z_{n-1})\alpha(z_{n-1})\end{aligned}$$

Sum-product for HMM



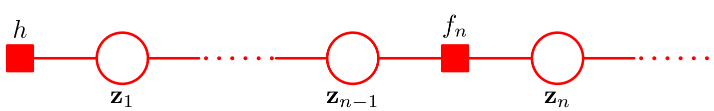
Factor node actions on right to left messages

$$\mu_{f_{n+1}\to f_n}(z_n) = \sum_{z_{n+1}} f_{n+1}(z_n, z_{n+1})\mu_{f_{n+2}\to f_{n+1}}(z_{n+1})$$

Identify $\beta(z_n) \equiv \mu_{f_{n+1}\to f_n}(z_n)$ to get

$$\beta(z_n) = \sum_{z_{n+1}} f_{n+1}(z_n, z_{n+1})\beta(z_{n+1})$$

Sum-product for HMM



Identify $\beta(z_n) \equiv \mu_{f_{n+1}\to f_n}(z_n)$ to get

$$\beta(z_n) = \sum_{z_{n+1}} f_{n+1}(z_n, z_{n+1})\beta(z_{n+1})$$

Recalling that $f_{n+1} = p(z_{n+1}|z_n)\, p(x_{n+1}|z_{n+1})$

$$\beta(z_n) = \sum_{z_{n+1}} p(z_{n+1}|z_n)p(x_{n+1}|z_{n+1})\beta(z_{n+1})$$

Sum-product for HMM

We have re-derived the alpha-beta version of forward-backward

Forward

$$\alpha(z_1) = p(z_1)\, p(x_1|z_1)$$
$$\alpha(z_n) = p(x_n|z_n)\sum_{z_{n-1}} p(z_n|z_{n-1})\alpha(z_{n-1})$$

Backward

$$\beta(z_N) = 1$$
$$\beta(z_n) = \sum_{z_{n+1}} p(z_{n+1}|z_n)p(x_{n+1}|z_{n+1})\beta(z_{n+1})$$

Sum-product for E step in the HMM learning problem (review)

Given all $\alpha(z_n)$ and $\beta(z_n)$

$$\gamma(z_n) = \frac{\alpha(z_n)\beta(z_n)}{p(X)}$$

$$p(X) = \sum_{z_n} \alpha(z_n)\beta(z_n)$$

$$\xi(z_{n-1}, z_n) = \frac{\alpha(z_{n-1})\, p(x_n | z_n)\, p(z_n | z_{n-1})\, \beta(z_n)}{p(X)}$$

Rescaled alpha beta (Bishop, 13.2.4)

The alpha-beta algorithm has similar precision problems to the ones for EM where we discussed the fix of scaling log quantities by the max, before exponentiation for normalizing.

One way to handle this is to reformulate the alpha-beta algorithm in terms of:

$$\hat{\alpha}(z_n) = p(z_n | x_1, \ldots, x_n) = \frac{\alpha(z_n)}{p(x_1, \ldots, x_n)}$$

$$\hat{\beta}(z_n) = \frac{p(x_{n+1}, \ldots, x_N | z_n)}{p(x_{n+1}, \ldots, x_N | x_1, \ldots, x_n)}$$

Classic HMM computational problems

Given data, what is the HMM (**learning**).

Given an HMM, what is the **distribution over the state** variables. Also, **how likely** are the observations, given the model.

Given an HMM, what is the most likely **state sequence** for some data?

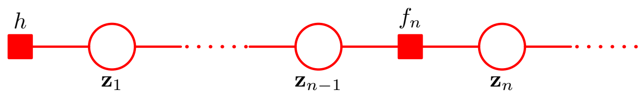
Viterbi algorithm (special case of max-sum)

Recall max-sum

Forward direction is like sum-product, except
- We take the max instead of sum
- We use sum of logs instead of product
- We remember incoming variable values that give max (*)

Backwards direction is simply backtracking on (*).

Recall simplified factor graph



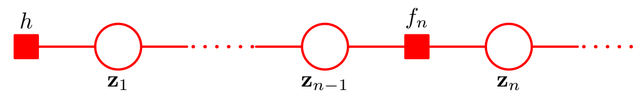
$h = p(z_1)\,p(x_1|z_1)$ $\qquad$ $f_n = p(z_n|z_{n-1})\,p(x_n|z_n)$

Left to right messages

$$\omega(z_n) = \log(x_n|z_n) + \max_{z_{n-1}}\left\{\log\left(p(z_n|z_{n-1}) + \omega(z_{n-1})\right)\right\}$$

$$\omega(z_1) = \log(p(z_1)) + \log(p(x_1|z_1))$$

Intuitive understanding



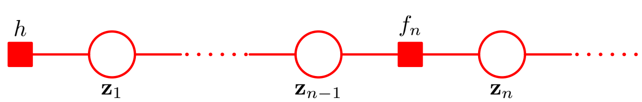
$$\omega(z_n) = \log(x_n|z_n) + \max_{z_{n-1}}\left\{\log\left(p(z_n|z_{n-1}) + \omega(z_{n-1})\right)\right\}$$

Consider all possible paths to each of the k states for time n.

The message encodes the probabilities for the maximum probability path for each of the K states.

EG, if you are in state k, this records is the probability of being there by via the maximal probably sequence.

Intuitive understanding



The message is the vector of probabilities for the maximum probability path for each of the K states.

$$\omega(z_n) = \log(x_n|z_n) + \max_{z_{n-1}}\left\{\underbrace{\log\left(p(z_n|z_{n-1}) + \omega(z_{n-1})\right)}_{\text{Consider getting there from each previous state k'}}\right\}$$

For each state k

The message is the vector of probabilities for the maximum probability path for each of the K states.

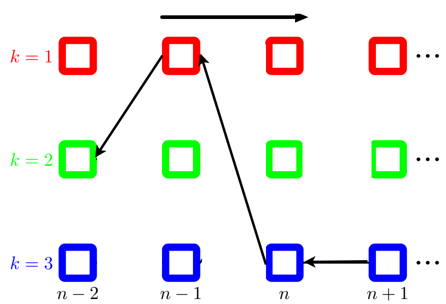
$$\omega(z_n) = \log(x_n|z_n) + \max_{z_{n-1}}\left\{\underbrace{\log\left(p(z_n|z_{n-1}) + \omega(z_{n-1})\right)}_{\text{Consider getting there from each previous state k'}}\right\}$$

For each state k

We can see that this is the new maximum

For Viterbi, we need to remember the previous state, k', for each k.

Intuitive understanding



The max path is shown (but we only know it when we get to the end).

To find the path, we need to chase the back pointers.

Final comments on learning

In many applications, the states have specified meaning, and are available in training data, so EM is not needed.

(Most authors still call this an HMM because states are hidden when the model is used).

We described training the HMM based on a single data sequence, but often multiple sequences that come from the same HMM are used (modifying the algorithm is very straightforward).

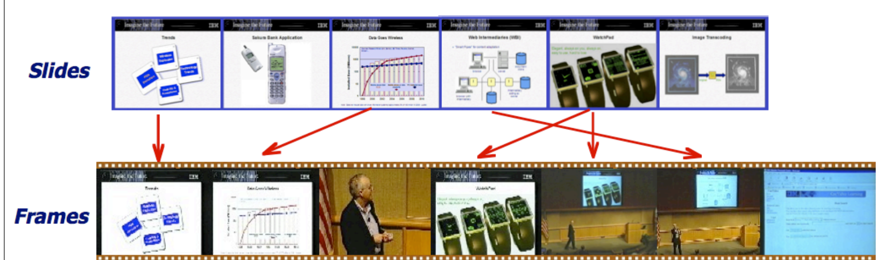
Two HMM examples (specified states)

Domain is SLIC (Semantically Linked Instructional Content).

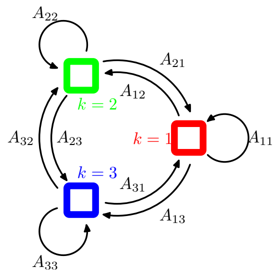
1) Temporal information for matching video frames to slides.

2) Aligning noisy speech transcripts with slides.

Matching slides to video frames



Matching slides to video frames



Our state sequence corresponds to what slide is being shown.

$$p(X,Z|\theta) = p(z_1|\pi)\left[\prod_{n=2}^{N} p(z_n|z_{n-1}, A)\right]\prod_{m=1}^{N} p(x_m|z_m, \phi)$$

Matching slides to video frames

$$p(X,Z|\theta) = p(z_1|\pi)\left[\prod_{n=2}^{N} p(z_n|z_{n-1}, A)\right]\boxed{\prod_{m=1}^{N} p(x_m|z_m, \phi)}$$

From image matching

Matching slides to video frames

$$p(X,Z|\theta) = p(z_1|\pi)\left[\prod_{n=2}^{N} p(z_n|z_{n-1}, A)\right]\prod_{m=1}^{N} p(x_m|z_m, \phi)$$

$p(z_n|z_{n-1}, A) = f(z_n - z_{n-1})$ encodes slide jump statistics.

We assume that only the jump matters. IE, going from slide 6 to 8 has the same chance of going from 10 to 12.

$p(z_1|\pi)\left[\prod_{n=2}^{N} p(z_n|z_{n-1}, A)\right]$ says how likely a sequence is, without looking at the images.

Aligning speech to slides

Why bother?

Mistakes in speech transcripts can be corrected.
Speech transcripts are noisy and to poorly on jargon
But jargon words often appear on slides.

We can highlight or auto-laser-point what the speaker is pointing to

We can improve close-captioning.

Aligning speech to slides

A reasonable model for some speakers is that they say some approximation of their bullet points, with some extra stuff before and after.

Automated speech recognizers try to produce results that are plausible on a phoneme level.

If a slide word is used, its phoneme sequence will likely be approximated in the phonemes in the speech transcript.

We can calibrate the phoneme "confusion matrix."

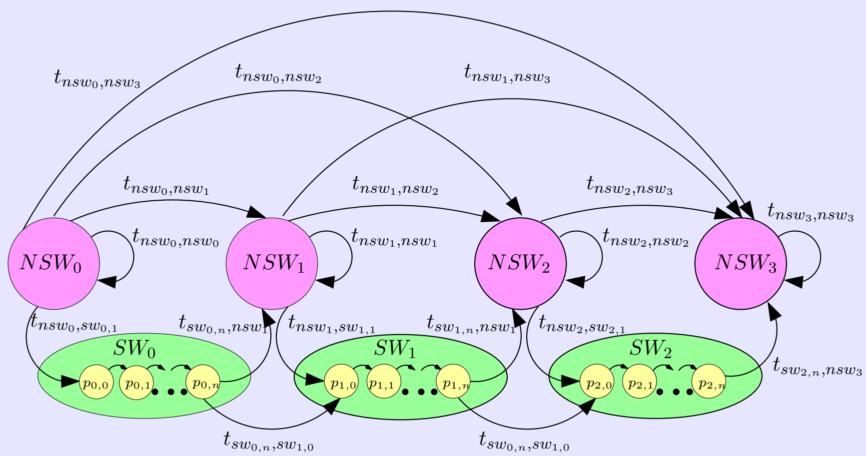
Aligning speech to slides

We assume that going backwards does not happen.

We have an HMM state for each slide word

We also have an HMM state for emitting phonemes between slide words.

Sequential Alignment Model



Aligned speech for correction

Speaker says : maliciousness

ASR produces: my dishes nests ***m ay d ih sh ah z n eh s t***

with slide word phoneme sequence

Slide word : maliciousness ***m ah l ih sh ah s n ah s***

If the same mistake is made later, where the word is not on the slide, we can propagate the correction.