

Hidden Markov models

Herman Kamper

2024-02, CC BY-SA 4.0

Part of speech tagging

HMM definition and notation

The three HMM problems

2. Most likely state sequence: Viterbi

1. Marginal probability of sequence

3. Learning HMM parameters

Use logs: The log-sum-exp trick

HMM topologies

Other aspects of HMMs

Part of speech tagging

Given input:

this is a simple sentence

the goal is to identify the part of speech (syntactic category) for each word:

this/DET is/VERB a/DET simple/ADJ sentence/NOUN

I use this task as a running example to introduce HMMs.

The set of part of speech (POS) categories can differ based on the application, corpus annotators and language.

One universal tag set used by Google (Petrov et al. 2011):

Tag	Description	Example
VERB	Verbs (all tenses and modes)	eat, ate, eats
NOUN	Nouns (common and proper)	home, Micah
PRON	Pronouns	I, you, your, he
ADJ	Adjectives	yellow, bigger, wildest
ADV	Adverbs	quickly, faster, fastest
ADP	Adpositions (prepositions and postpositions)	of, in, by, under
CONJ	Conjunctions	and, or, but
DET	Determiners	a, an, the, this
NUM	Cardinal numbers	one, two, first, second
PRT	Particles, other function words	up, down, on, off
X	Other: foreign words, typos, abbreviations	brasserie, abcense, HMM
.	Punctuation	?, !, .

POS tagging is hard

Ambiguity:

glass of water/NOUN	vs	water/VERB the plants
lie/VERB down	vs	tell a lie/NOUN
wind/VERB down	vs	a mighty wind/NOUN

Sparsity:

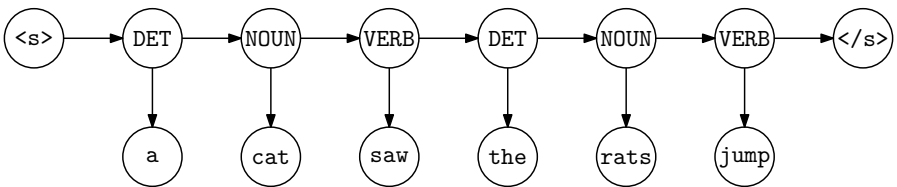
- Words we never see
- Word-tag pairs we never see

A probabilistic model for tagging

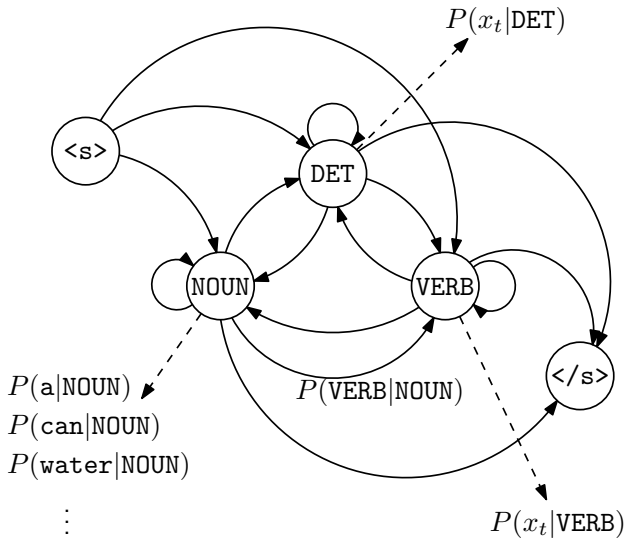
Let x_t denote the word and z_t denote the tag at time step t .

- Initialisation: $z_0 = \langle s \rangle$
- Repeat:
 - Choose a tag based on the previous tag: $P(z_t|z_{t-1})$
 - If $z_t = \langle /s \rangle$: Break
 - Choose a word conditioned on its tag: $P(x_t|z_t)$

If we knew all the transition probabilities $P(z_t|z_{t-1})$ and all the emission probabilities $P(x_t|z_t)$, we could generate a sentence:



Could also represent this with a state diagram:

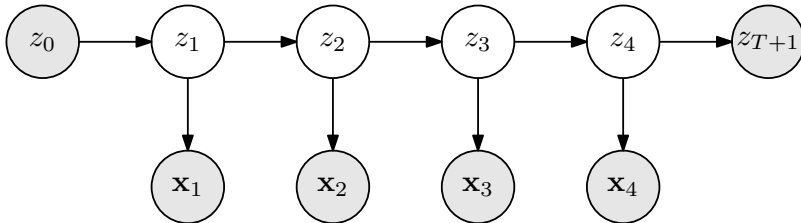


HMM definition and notation

A hidden Markov model (HMM) defines a probability distribution over a sequence of states and output observations:

- Output sequence: $\mathbf{x}_{1:T} = \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T$. We denote these as vectors, but they can also be scalars or discrete observations.
- State sequence: $z_{0:T+1} = z_0, z_1, \dots, z_T, z_{T+1}$. The integer variable $z_t \in \{0, \dots, K + 1\}$ represents the state at step t .

Often $\mathbf{x}_{1:T}$ is observed and $z_{0:T+1}$ is hidden:



This is a directed graphical model representation. Contrast this with the state diagram drawn earlier.

Why “hidden”? Why “Markov”?

An HMM is specified by:

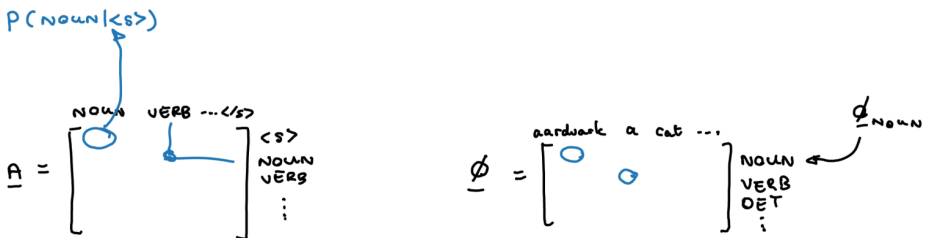
- A set of states: $\{0, 1, \dots, K + 1\}$
- Transition probabilities: \mathbf{A} with $A_{i,j} = P_{\mathbf{A}}(z_t = j | z_{t-1} = i)$
- Emission distribution for each state: $p_{\phi}(\mathbf{x}_t | z_t)$
- Group the parameters together: $\theta = \{\mathbf{A}, \phi\}$

I denote the emission distributions as continuous, but these can also be discrete.

The start and end states are special:¹

- Always start in $z_0 = 0$.
Transitioning out of this start state is captured by $A_{0,j}$.
- Always end in $z_{T+1} = K + 1$.
Transitioning into this final state is captured by $A_{j,K+1}$.
- States 0 and $K + 1$ are non-emitting.
They don't produce an \mathbf{x} when we move in or out of them.

POS example:



¹The classic description of HMMs explicitly defines separate start and end probabilities, but we don't need to do this if we use these non-emitting states.

The three HMM problems

Problem 1: The marginal probability

Given an observed sequence $\mathbf{x}_{1:T}$ and a trained HMM with parameters θ , what is the probability of the observed sequence $p_{\theta}(\mathbf{x}_{1:T})$?

Problem 2: The most likely state sequence

Given an observed sequence $\mathbf{x}_{1:T}$ and a trained HMM with parameters θ , what is the most likely state sequence through the HMM?

$$\arg \max_{z_{0:T+1}} P_{\theta}(z_{0:T+1} | \mathbf{x}_{1:T})$$

Problem 3: Learning

Given training data $\mathbf{x}_{1:T}$, how do we choose the HMM parameters θ to maximize $p_{\theta}(\mathbf{x}_{1:T})$?²

This is the conventional ordering of the problems (Rabiner and Juang 1993), but we will do it slightly differently by starting with problem 2.

²In this note I will sometimes be a bit sloppy and drop the dependence of the distribution on θ (or \mathbf{A} or ϕ), but just remember that they are there.

2. Most likely state sequence: Viterbi

We want to find the most likely state sequence given the observed sequence:

$$\begin{aligned} \arg \max_{z_{0:T+1}} P(z_{0:T+1} | \mathbf{x}_{1:T}) &= \arg \max_{z_{0:T+1}} \frac{p(\mathbf{x}_{1:T}, z_{0:T+1})}{p(\mathbf{x}_{1:T})} \\ &= \arg \max_{z_{0:T+1}} p(\mathbf{x}_{1:T}, z_{0:T+1}) \end{aligned}$$

We could just calculate $p(\mathbf{x}_{1:T}, z_{0:T+1})$ for every possible state sequence $z_{0:T+1}$. But there are K^T possible sequences. Instead we use dynamic programming. (Of course.)

Define

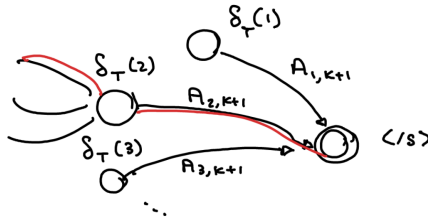
$$\delta_t(j) \triangleq \max_{z_{0:t-1}} p(\mathbf{x}_{1:t}, z_{0:t-1}, z_t = j)$$

These variables can be calculated recursively:

$$\begin{aligned} \delta_t(j) &= \max_{z_{0:t-1}} p(\mathbf{x}_{1:t}, z_{0:t-1}, z_t = j) \\ &= \max_{z_{0:t-1}} p(\mathbf{x}_t | \mathbf{x}_{1:t-1}, z_{0:t-1}, z_t = j) p(\mathbf{x}_{1:t-1}, z_{0:t-1}, z_t = j) \\ &= \max_{z_{0:t-1}} p(\mathbf{x}_t | z_t = j) P(z_t = j | \mathbf{x}_{1:t-1}, z_{0:t-1}) p(\mathbf{x}_{1:t-1}, z_{0:t-1}) \\ &= \max_{z_{0:t-1}} p(\mathbf{x}_t | z_t = j) P(z_t = j | z_{t-1}) p(\mathbf{x}_{1:t-1}, z_{0:t-1}) \\ &= \max_{i=1}^K \max_{z_{0:t-2}} p(\mathbf{x}_t | z_t = j) P(z_t = j | z_{t-1} = i) p(\mathbf{x}_{1:t-1}, z_{0:t-2}, z_{t-1} = i) \\ &= \max_{i=1}^K p(\mathbf{x}_t | z_t = j) P(z_t = j | z_{t-1} = i) \max_{z_{0:t-2}} p(\mathbf{x}_{1:t-1}, z_{0:t-2}, z_{t-1} = i) \\ &= \max_{i=1}^K p(\mathbf{x}_t | z_t = j) A_{i,j} \delta_{t-1}(i) \end{aligned}$$

Similarly we can show that

$$\hat{P} = \max_{z_{0:T+1}} p(\mathbf{x}_{1:T}, z_{0:T+1}) = \max_{j=1}^K A_{j,K+1} \delta_T(j)$$



To find the best state sequence, we start with this last equation and backtrack our steps:

$$\hat{z}_T = \arg \max_{j=1}^K A_{j,K+1} \delta_T(j)$$

The next state is determined by looking at the $\arg \max$ that gave us $\delta_T(\hat{z}_T)$, i.e.

$$\hat{z}_{T-1} = \arg \max_{j=1}^K p(\mathbf{x}_T | z_T = \hat{z}_T) A_{j,\hat{z}_T} \delta_{T-1}(j)$$

and so on.

When calculating $\delta_t(j)$, we normally just store the backtrace (the $\arg \max$), denoted as $\psi_t(j)$.

Viterbi algorithm

- Initialisation:

$$\delta_1(j) = p(\mathbf{x}_1 | z_1 = j) A_{0,j}$$

for $j = 1, 2, \dots, K$

- Recursion:

$$\delta_t(j) = \max_{i=1}^K p(\mathbf{x}_t | z_t = j) A_{i,j} \delta_{t-1}(i)$$
$$\psi_t(j) = \arg \max_{i=1}^K p(\mathbf{x}_t | z_t = j) A_{i,j} \delta_{t-1}(i)$$

for $t = 2, 3, \dots, T$ and $j = 1, 2, \dots, K$

- Termination:

$$\hat{P} = \max_{j=1}^K A_{j,K+1} \delta_T(j)$$
$$\hat{z}_T = \arg \max_{j=1}^K A_{j,K+1} \delta_T(j)$$

- Backtracking:

$$\hat{z}_t = \arg \max_{j=1}^K p(\mathbf{x}_{t+1} | z_{t+1} = \hat{z}_{t+1}) A_{j,\hat{z}_{t+1}} \delta_t(j)$$
$$= \psi_{t+1}(\hat{z}_{t+1})$$

for $t = T - 1, T - 2, \dots, 1$

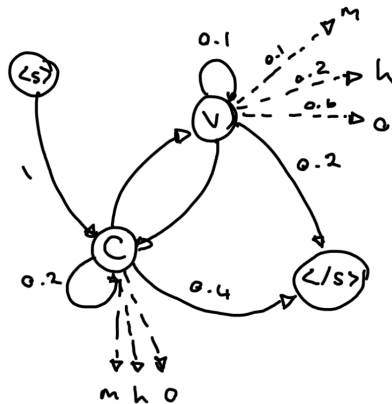
Example: Viterbi

Transition and emission probabilities:

$$\mathbf{A} = \begin{matrix} & \begin{matrix} c & v & \langle /s \rangle \end{matrix} \\ \begin{matrix} c \\ v \end{matrix} & \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.2 & 0.4 & 0.4 \\ 0.7 & 0.1 & 0.2 \end{bmatrix} \end{matrix} \begin{matrix} \langle s \rangle \\ c \\ v \end{matrix}$$

$$\phi = \begin{matrix} & \begin{matrix} m & h & o \end{matrix} \\ \begin{matrix} c \\ v \end{matrix} & \begin{bmatrix} 0.6 & 0.2 & 0.2 \\ 0.1 & 0.3 & 0.6 \end{bmatrix} \end{matrix} \begin{matrix} c \\ v \end{matrix}$$

Observed sequence: $x_{1:3} = m \ o \ h$



Calculate the Viterbi variables $\delta_t(j)$:

		$x_1 = m$	$x_2 = o$	$x_3 = h$	
0 : <s>					
1 : C					
2 : V					
3 : </s>					

$$\underline{t=1}: \delta_1(j) = P(x_1 | z_1=j) \cdot A_{o,j}$$

$$\begin{aligned} \delta_1(c) &= P(x_1 | z_1=c) \cdot A_{<s>,c} \\ &= P(m|C) \cdot A_{<s>,c} \end{aligned}$$

$$= 0.6 \times 1 = \underline{0.6} \rightarrow$$

$$\delta_1(v) = P(m|V) \cdot A_{<s>,v}$$

$$= 0.1 \times 0 = \underline{0} \rightarrow$$

$$\underline{t=2}: \delta_2(j) = \max_{i=1}^2 P(x_2 | z_2=j) \cdot A_{i,j} \cdot \delta_1(i)$$

$$\delta_2(c) = \max \begin{cases} i=1: P(o|C) \cdot A_{c,c} \delta_1(c) \\ i=2: P(o|c) \cdot A_{v,c} \delta_1(v) \end{cases}$$

$$= \max \begin{cases} 0.2 \times 0.2 \times 0.6 \\ 0.2 \times 0.7 \times 0 \end{cases}$$

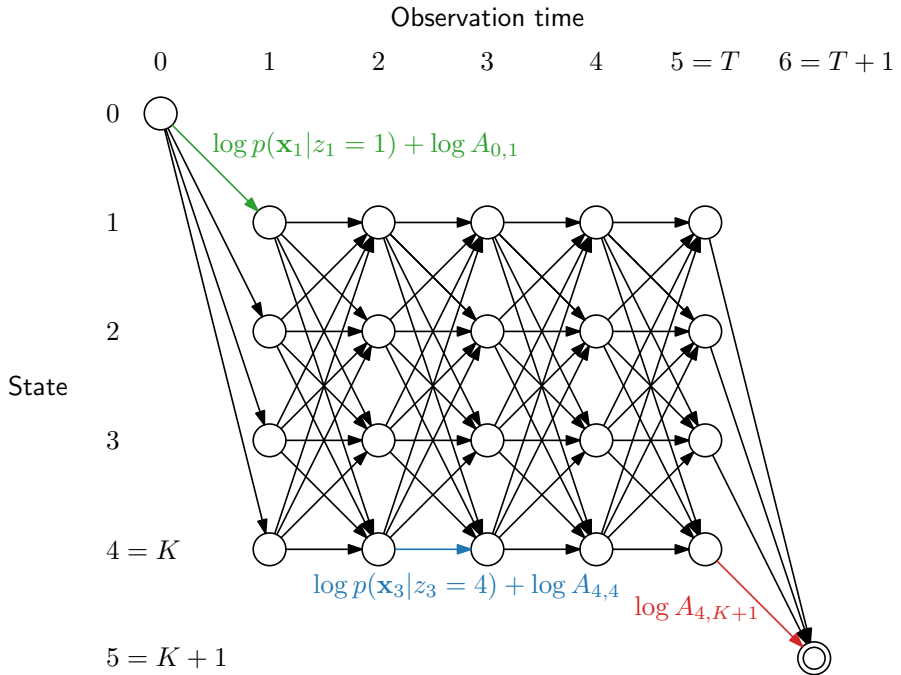
$$= 0.024$$

Calculated Viterbi variables $\delta_t(j)$ with backtracking:

		$x_1 = \mathbf{m}$	$x_2 = \mathbf{o}$	$x_3 = \mathbf{h}$	
0 : <s>	1				
1 : C		$\nearrow 0.6$	$\leftarrow 0.024$	$\swarrow 0.02016$	
2 : V		$\nearrow 0$	$\nearrow 0.144$	$\leftarrow 0.00432$	
3 : </s>					0.008064

Does this example seem familiar?

Just finding the shortest path through a DAG



See my separate note on [dynamic programming](#) for more details.

Time complexity: In the Viterbi recursion, for each node in the graph, we need to consider the max over K terms. There are roughly $K \cdot T$ nodes, which gives $\mathcal{O}(K^2T)$ terms in the max operations. This is much better than the $\mathcal{O}(K^T)$ possible paths.

1. Marginal probability of sequence

We want the probability of a sequence $p(\mathbf{x}_{1:T})$. We will also need this for solving problem 3.

Marginalising we have:

$$p(\mathbf{x}_{1:T}) = \sum_{z_{0:T+1}} p(\mathbf{x}_{1:T}, z_{0:T+1})$$

We are summing over all possible state sequences. One of these corresponds exactly to the Viterbi path above. In the summation, this path will be the one with the highest value \hat{P} .

Again we could in principle enumerate over all possible sequences, but in practice there are too many: K^T . Again we turn to dynamic programming. (Of course.)

Define forward variables:

$$\alpha_t(j) \triangleq p(\mathbf{x}_{1:t}, z_t = j)$$

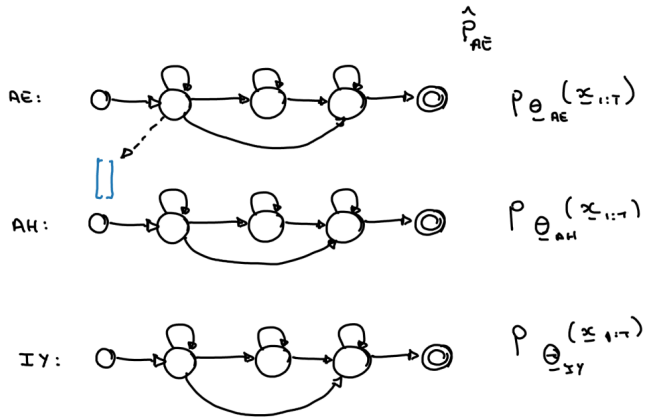
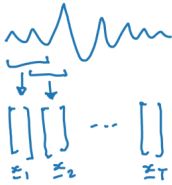
These variables can be calculated recursively:

$$\begin{aligned} \alpha_t(j) &= p(\mathbf{x}_{1:t}, z_t = j) \\ &= p(\mathbf{x}_t | \mathbf{x}_{1:t-1}, z_t = j) p(\mathbf{x}_{1:t-1}, z_t = j) \\ &= p(\mathbf{x}_t | z_t = j) \sum_{i=1}^K p(\mathbf{x}_{1:t-1}, z_t = j, z_{t-1} = i) \\ &= p(\mathbf{x}_t | z_t = j) \sum_{i=1}^K P(z_t = j | z_{t-1} = i) p(\mathbf{x}_{1:t-1}, z_{t-1} = i) \\ &= p(\mathbf{x}_t | z_t = j) \sum_{i=1}^K A_{i,j} \alpha_{t-1}(i) \end{aligned}$$

The marginal probability can then be calculated as:

$$p(\mathbf{x}_{1:T}) = \sum_{j=1}^K A_{j,K+1} \alpha_T(j)$$

Example: Why would we need the marginal?



Forward algorithm

- Initialisation:

$$\alpha_0(j) = \begin{cases} 1 & \text{if } j = 0 \\ 0 & \text{for } j = 1, 2, \dots, K \end{cases}$$
$$\alpha_1(j) = p(\mathbf{x}_1 | z_1 = j) A_{0,j}$$

for $j = 1, 2, \dots, K$

- Recursion:

$$\alpha_t(j) = p(\mathbf{x}_t | z_t = j) \sum_{i=1}^K A_{i,j} \alpha_{t-1}(i)$$

for $t = 2, 3, \dots, T$ and $j = 1, 2, \dots, K$

- Termination:

$$p(\mathbf{x}_{1:T}) = \sum_{j=1}^K A_{j,K+1} \alpha_T(j)$$

Compare this algorithm to the Viterbi algorithm: The max operations have just been changed to sums.

The $\alpha_0(j)$ initialisation is only used later for solving problem 3 and not necessary for calculating $p(\mathbf{x}_{1:T})$.

Example: Forward algorithm

Calculate the forward variables $\alpha_t(j)$:

		$x_1 = m$	$x_2 = o$	$x_3 = h$	
0 : <s>					
1 : C					
2 : V					
3 : </s>					

$$\alpha_3(j) = p(x_3 = h | z_3 = j) \sum_{i=1}^K A_{i,j} \alpha_2(i)$$

$$j=1: \alpha_3(c) = p(h|c) \cdot [A_{c,c} \cdot \alpha_2(c) + A_{v,c} \cdot \alpha_2(v)]$$

$$= 0.2 \times [0.2 \times 0.024 + 0.1 \times 0.144]$$

$$= 0.02112 \rightarrow$$

Calculated forward variables $\alpha_t(j)$ and marginal probability $P(x_{1:3})$:

		$x_1 = m$	$x_2 = o$	$x_3 = h$	
0 : <s>	1				
1 : C		0.6	0.024	0.02112	
2 : V		0	0.144	0.0072	
3 : </s>					0.009888

Backward algorithm

We could have alternatively gone in the reverse direction by defining backwards variables:

$$\beta_t(j) \triangleq p(\mathbf{x}_{t+1:T} | z_t = j)$$

which can also be calculated recursively.

- Initialisation:

$$\beta_{T+1}(j) = \begin{cases} 1 & \text{for } j = K + 1 \\ 0 & \text{for } j = 1, 2, \dots, K \end{cases}$$

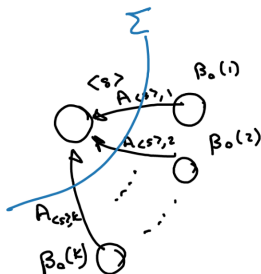
$$\beta_T(j) = A_{j,K+1} \\ \text{for } j = 1, 2, \dots, K$$

- Recursion:

$$\beta_t(j) = \sum_{i=1}^K p(\mathbf{x}_{t+1} | z_{t+1} = i) A_{j,i} \beta_{t+1}(i) \\ \text{for } t = T - 1, T - 2, \dots, 0 \text{ and } j = 1, 2, \dots, K$$

- Termination:

$$p(\mathbf{x}_{1:T}) = \sum_{j=1}^K A_{0,j} \beta_0(j)$$



Again the $\beta_{T+1}(j)$ initialisation isn't required for $p(\mathbf{x}_{1:T})$ and only becomes relevant for solving problem 3.

3. Learning HMM parameters

Goal: We want to learn the HMM parameters $\theta = \{\mathbf{A}, \phi\}$ from training sequences $\mathbf{x}_{1:T}$.

Chicken and egg problem:

- If we were in a supervised setting with a known label sequence $z_{0:T+1}$ that goes with training sequence $\mathbf{x}_{1:T}$, then we could just use maximum likelihood estimation (MLE), e.g.

$$\hat{A}_{i,j} = \frac{C(z_t = i, z_{t+1} = j)}{C(z_t = i)}$$

For POS tagging, this would be like knowing the tags for the words in your training data.

- If we weren't given the labelled state sequences but we knew the HMM parameters θ , then we could find the best state sequence using Viterbi.

But we have neither labels nor the parameters!

A sketch of expectation-maximisation (EM)

- Initialisation: Choose initial $\theta^{(0)}$.
- Recursion: For each iteration m :
 - **E-step:** Compute expected counts using $\theta^{(m-1)}$.
 - **M-step:** Set $\theta^{(m)}$ by using MLE with the expected counts from the E-step.
- Termination: Repeat until θ converges (or some other criterion).

We consider two flavours of EM for HMMs.

Hard EM: Viterbi re-estimation

If we had the hard assignments of observations \mathbf{x}_t to states z_t , then the MLE estimates for the HMM parameters θ would be as follows.

- Transition probabilities:

$$\hat{A}_{i,j} = \frac{C(z_t = i, z_{t+1} = j)}{C(z_t = i)}$$

- Emission distributions:

- We collect all the the \mathbf{x} 's assigned to state k and then set ϕ_k to the MLE.
- If $p_{\phi_k}(\mathbf{x}_t | z_t = k)$ is a continuous density, this will be the MLE for that density.
- If $P_{\phi_k}(x_t | z_t = k)$ is a discrete probability mass function for a categorical distribution, then the MLE is

$$\begin{aligned}\hat{\phi}_{k,c} &= P(x_t = c | z_t = k) \\ &= \frac{C(z_t = k, x_t = c)}{C(z_t = k)}\end{aligned}$$

The hard EM algorithm

- Initialisation:
 - For each training sequence $\mathbf{x}_{1:T}$: Assign a sensible $z_{0:T+1}$ (what is sensible?)
 - Estimate $\theta^{(0)}$ using the MLE equations (previous page).
- Recursion: For each iteration m :
 - **E-step**: For each training sequence $\mathbf{x}_{1:T}$:
 - * Apply Viterbi with $\theta^{(m-1)}$ to get the optimal state sequence $\hat{z}_{0:T+1}$.
 - * Calculate: $\hat{P} = p(\mathbf{x}_{1:T}, \hat{z}_{0:T+1})$
 - Accumulate the scores: $J^{(m)} = \sum_{n=1}^N \log \hat{P}^{(n)}$
(where n is the index of the training sequence)
 - **M-step**: Use all the optimal state sequences $\hat{z}_{0:T+1}$ for all the training sequences and estimate $\theta^{(m)}$ using the MLE equations (previous page).
- Termination: Compare score $J^{(m)}$ to the previous one $J^{(m-1)}$ and decide whether to stop.

Although this algorithm uses hard assignments, it is still guaranteed to converge to a local optimum. This is because every E- and M-step guarantees that $J^{(m)} \geq J^{(m-1)}$.

Soft EM: Baum-Welch re-estimation

An instance of the forward-backward algorithm. Here we use soft counts instead of hard assignments.

Define the probability of being in state j at time t :

$$\gamma_t(j) \triangleq P(z_t = j | \mathbf{x}_{1:T})$$

which can be calculated as

$$\begin{aligned}\gamma_t(j) &= \frac{p(z_t = j, \mathbf{x}_{1:T})}{p(\mathbf{x}_{1:T})} \\ &= \frac{p(\mathbf{x}_{1:t}, \mathbf{x}_{t+1:T}, z_t = j)}{p(\mathbf{x}_{1:T})} \\ &= \frac{p(\mathbf{x}_{t+1:T} | z_t = j, \mathbf{x}_{1:t}) p(\mathbf{x}_{1:t}, z_t = j)}{p(\mathbf{x}_{1:T})} \\ &= \frac{\alpha_t(j) \beta_t(j)}{p(\mathbf{x}_{1:T})}\end{aligned}$$

Define the probability of transitioning from state i at time t to state j at $t + 1$:

$$\xi_t(i, j) \triangleq P(z_t = i, z_{t+1} = j | \mathbf{x}_{1:T})$$

which can be calculated as

$$\begin{aligned}\xi_t(i, j) &= \frac{p(z_t = i, z_{t+1} = j, \mathbf{x}_{1:T})}{p(\mathbf{x}_{1:T})} \\ &= \frac{p(\mathbf{x}_{1:t}, \mathbf{x}_{t+1}, \mathbf{x}_{t+2:T}, z_t = i, z_{t+1} = j)}{p(\mathbf{x}_{1:T})} \\ &= \frac{\alpha_t(i) A_{i,j} p(\mathbf{x}_{t+1} | z_{t+1} = j) \beta_{t+1}(j)}{p(\mathbf{x}_{1:T})}\end{aligned}$$

The E- and M-steps at iteration m now change to:

- **E-step:**

- Forward-backward: Calculate all the forward variables $\alpha_t(j)$ and backward variables $\beta_t(j)$ using the current $\theta^{(m-1)} = \{\mathbf{A}^{(m-1)}, \phi^{(m-1)}\}$.
- Calculate the expected number of transitions out of state j :

$$\sum_{t=0}^T \gamma_t(j)$$

- Calculate the expected number of transitions from state i to j :

$$\sum_{t=0}^T \xi_t(i, j)$$

- **M-step:**

- Transition probabilities:

$$\hat{A}_{i,j} = \frac{\sum_{t=0}^T \xi_t(i, j)}{\sum_{t=0}^T \gamma_t(i)}$$

- Emission probability mass function for categorical distribution:

$$\begin{aligned} \hat{\phi}_{k,c} &= P(x_t = c | z_t = k) \\ &= \frac{\sum_{t=1}^T \mathbb{I}\{x_t = c\} \gamma_t(k)}{\sum_{t=1}^T \gamma_t(k)} \end{aligned}$$

Side note: Start and end states

We assume we start in state 0 and end in state $K + 1$. We therefore need to be a bit careful with $\gamma_t(j)$ when $t = 0$ and with $\xi_t(i, j)$ when $t = 0$ or $t = T$.

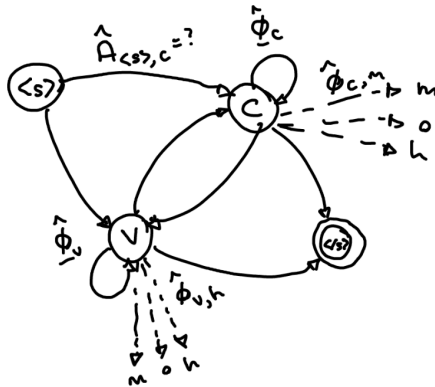
As one example of these edge cases, we need

$$\gamma_0(j) = \begin{cases} 1 & \text{if } j = 0 \\ 0 & \text{otherwise} \end{cases}$$

Fortunately the initialisations for $\alpha_0(j)$ and $\beta_{T+1}(j)$ given earlier ensure that $\gamma_t(j)$ and $\xi_t(i, j)$ are correct at the start and end of a sequence. The one additional requirement is that we don't have a $p(\mathbf{x}_{T+1} | z_{t+1} = j)$ term in the numerator when calculating $\xi_T(i, j)$. (Why not?)

Example: Soft and hard EM

- States: $\{0, 1, 2, 3\}$
- Output vocabulary: $\{m, o, h\}$
- Training sequence: $x_{1:T} = m \ o \ o \ m \ o \ h \ o \ h \ o$



Hard EM

m o o m o h o h o

E-step: $\langle s \rangle$ C C \rightarrow V c \rightarrow V v V C \rightarrow V $\langle /s \rangle$ Viterbi
 m o o m o h o h o

M-step:

$$\hat{\theta}_{c,v} = \frac{\text{Count}(c,v)}{\text{Count}(c)} = \frac{3}{4} = 0.75$$

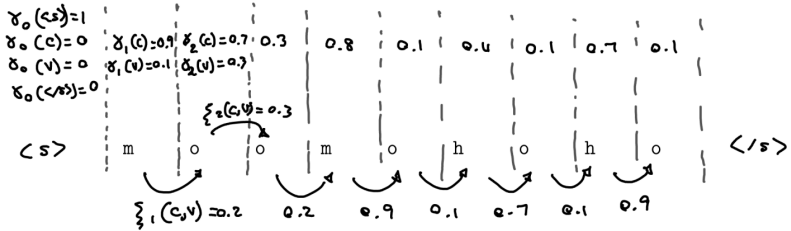
$$\hat{\phi}_{v,o} = P(x_t=o | z_t=v) = \frac{\text{Count}(v \rightarrow o)}{\text{Count}(v)} = \frac{4}{5} = 0.8$$

Soft EM

m o o m o h o h o

E-step:

Forward -
backward



M-step:

$$\hat{A}_{c,v} = \frac{\text{Soft Count}(c,v)}{\text{Soft Count}(c)} = \frac{0.2 + 0.3 + 0.2 + 0.9 + \dots}{0 + 0.9 + 0.7 + 0.3 + 0.8 + \dots} = \frac{\sum_{t=0}^T \sum_j \delta_t(i,j)}{\sum_{t=0}^T \delta_t(j)}$$

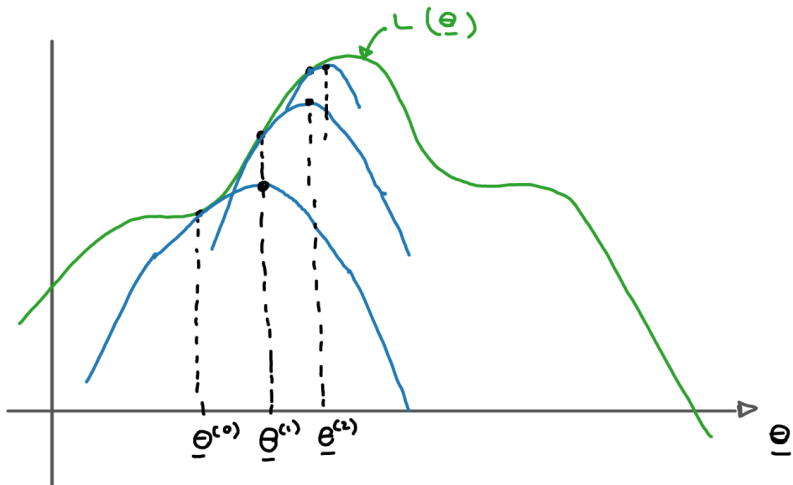
δ
 \sum

Why does EM improve the likelihood?

Log likelihood:

$$\begin{aligned} L(\theta) &= \sum_{n=1}^N \log p_{\theta}(\mathbf{x}_{1:T_n}^{(n)}) \\ &= \sum_{n=1}^N \log \left[\sum_{z_{1:T_n}} p_{\theta}(\mathbf{x}_{1:T_n}^{(n)}, z_{1:T_n}^{(n)}) \right] \geq \text{ELBO}(\theta) \end{aligned}$$

Intuitively:



Use logs: The log-sum-exp trick

We have seen before that when multiplying many probabilities, we can get underflow. So let's work in the log domain! But just taking the log isn't enough in this case.

Consider the forward variables:

$$\alpha_t(j) = p(\mathbf{x}_t | z_t = j) \sum_{i=1}^K A_{i,j} \alpha_{t-1}(i)$$

We now switch to working with $\log \alpha_t(j)$ throughout (and similar for the other variables):

$$\begin{aligned} \log \alpha_t(j) &= \log p(\mathbf{x}_t | z_t = j) + \log \sum_{i=1}^K A_{i,j} \alpha_{t-1}(i) \\ &= \log p(\mathbf{x}_t | z_t = j) + \log \sum_{i=1}^K e^{\log A_{i,j} + \log \alpha_{t-1}(i)} \end{aligned}$$

But, sadly, the summation for calculating $\log \alpha_t(j)$ can't happen in the log domain. Even with all the required probabilities in the log domain, we would first have to bring the probabilities back to the linear domain before summing.

The log-sum-exp trick

We often end up having to calculate terms like

$$\log \sum_{k=1}^K e^{b_k}$$

where we could potentially get underflow/overflow when calculating e^{b_k} .

Luckily we have a trick that allows us to calculate the sum! The log-sum-exp trick works as follows:

$$\begin{aligned} \log \sum_{k=1}^K e^{b_k} &= \log \left[\sum_{k=1}^K e^{b_k - B} e^B \right] \\ &= \log e^B + \log \sum_{k=1}^K e^{b_k - B} \\ &= B + \log \sum_{k=1}^K e^{b_k - B} \end{aligned}$$

with $B = \max_{k=1}^K b_k$.

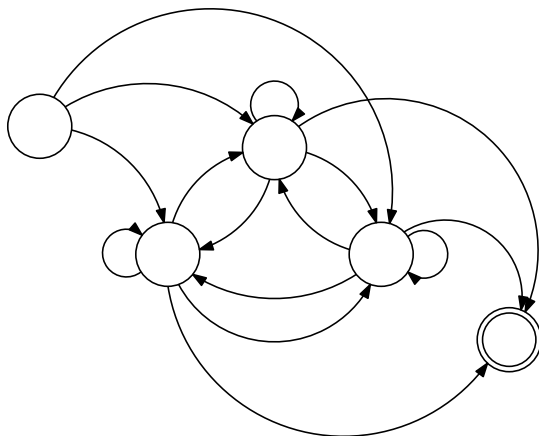
Now at least we don't get underflow/overflow from the biggest term in the summation and hopefully the other terms are also slightly better.

HMM topologies

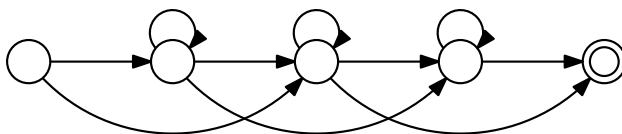
Say $\hat{A}_{i,j} = 0$, i.e. $P_{\hat{A}}(z_t = j | z_{t-1} = i) = 0$. How will the estimate $\hat{A}_{i,j}$ change during EM training?

Turning off some transitions at initialisation allows us to specify different HMM topologies.

Fully connected topology:



Left-to-right topology:



Other aspects of HMMs

Smoothing

Some state transitions or some state-output combinations might never occur in the training data. Similar to what we do with N -gram language models, we could use smoothing to assign some mass in $P(x_t|z_t)$ and $P(z_t|z_{t-1})$ to unseen events.

Extensions of HMMs

- Higher-order HMMs

HMMs today

- Although we looked at POS tagging as a running example in this note, HMMs are rarely used for supervised or unsupervised POS tagging today.
- But HMMs are still used in speech recognition:
 - HMM-based speech recognition was the standard up to the early 2010s
 - HMMs still used today in hybrid HMM-DNN speech recognition systems

Videos covered in this note

- [A first hidden Markov model example](#) (14 min)
- [Hidden Markov model definition](#) (9 min)
- [The three HMM problems](#) (3 min)
- [The Viterbi algorithm for HMMs](#) (24 min)
- [Viterbi HMM example](#) (19 min)
- [Why do we want the marginal probability in an HMM?](#) (7 min)
- [The forward algorithm for HMMs](#) (19 min)
- [Learning in HMMs](#) (8 min)
- [Hard expectation maximisation for HMMs](#) (12 min)
- [Soft expectation maximisation for HMMs](#) (20 min)
- [Why expectation maximisation works](#) (12 min)
- [The log-sum-exp trick](#) (9 min)
- [Hidden Markov models in practice](#) (4 min)

Acknowledgements

This note uses content from:

- Johan du Preez's HMM lectures at Stellenbosch University
- Sharon Goldwater's NLP course at the University of Edinburgh

References

- S. Petrov, D. Das, and R. McDonald, "[A universal part-of-speech tagset](#)," *arXiv*, 2011.
- L. Rabiner and B.-H. Juang, *Fundamentals of Speech Recognition*, 1993.
- H. Tang, "[Hidden Markov models \(part 1\)](#)," *University of Edinburgh*, 2021.