Note on HMM inference

Brendan O'Connor

November 27, 2012

HMM notation: $P(z_1..z_T|x_1..x_T)$ is the posterior probability of path $\vec{z}$ given observations $\vec{x}$.

The Viterbi, Forward, and Backward algorithms each make a series of tables: one per timestep t, of size K. Consider just the Viterbi and Forward algorithms; they have related declarative semantics, shown in the left column:

what it means		to calculate: for $t = 1..T$,	
$V_t[k] \equiv$	$\max_{z_1..z_{t-1}} P(z_t = k,\ x_1..x_t,\ z_1..z_{t-1})$	$V_t[k] :=$	$\max_j V_{t-1}[j] P_{trans}(j \to k) P_{emit}(k \to x_t)$
$\alpha_t[k] \equiv$	$P(z_t = k,\ x_1..x_t)$		
$=$	$\sum_{z_1..z_{t-1}} P(z_t = k,\ x_1..x_t,\ z_1..z_{t-1})$	$\alpha_t[k] :=$	$\sum_j \alpha_{t-1}[j] P_{trans}(j \to k) P_{emit}(k \to x_t)$

You can use the Forward algorithm to solve what's known as the "filtering" problem, $p(z_t = k|x_1..x_t)$: when you're at timestep t, given all your current information, you'd like to know where you currently are, or what the current latent state is. (Think of an airplane guidance system trying to know where it is, or a speech recognizer wanting to know the speaker's current word as they're talking.) Since $p(z_t|x_1..x_t) \propto p(z_t, x_1..x_t)$, you just renormalize the Forward α_t table to get this. An entry in the Viterbi table $V_t[k]$, by contrast, tells you the probability of the best path that ends with k. And of course you don't explicitly sum or maximize over all possible predecessor paths for these; due to the Markovian structure of the Bayes net, you can proceeed left-to-right and only look at the immediate previous position when calculating the new α_t or V_t table. The Forward and Viterbi algorithms, when calculating each new table, have identical form except for the sum vs max: thus they are also called the *sum-product* and *max-product* algorithms. The algorithm to apply at a step t is shown in the right column above.

OK, but what about when we have seen all the evidence and want to make historical inferences about what happened? For example, the max-path problem $\arg\max_{z_1..z_T} p(z_1..z_T|x_1..x_T)$ (i.e. the speech recognizer has received the entire utterance and wants to work out the most likely sequence of words that generated it), which we solve exactly using Viterbi. At the end of the algorithm, the best score in the final table V_T tells you the best $p(\vec{z}, \vec{x})$. But actually getting the path is tricky. If you look at a $\hat{z}_t = \arg\max_k V_t[k]$, that actually says that if you only look at evidence up to t, then for all paths from 1 to t, the best path ends with state $\hat{z}_t$. That's not necessarily the state on the best path for the entire sequence: you might see new evidence that changes your beliefs about what was going on at t. To correctly use the Viterbi algorithm you have to track *backpointers* (more details in the assigned readings); it's just another set of tables, $B_t[k]$, where you store the argmaxes instead of the maxes: track which previous state j would be best to use to get to k at t. Then once you're done, you start off by using the best-scoring state in V_T, look at which state it was most likeliy to come from, and keep reading it off backwards.

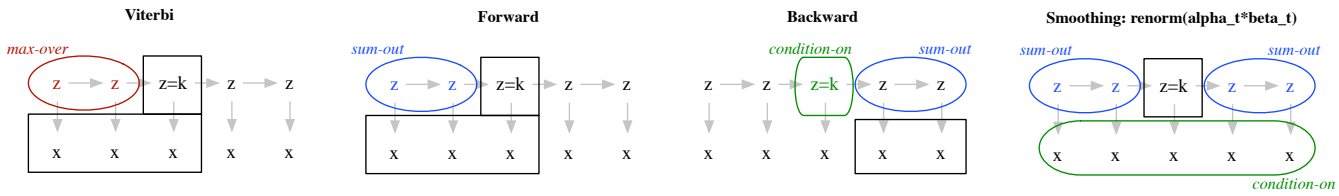

Figure 1: Picture of the probabilities in the V_t, α_t, and β_t tables.

To run EM, we don't want either the filtering quantity or the max-path probability; instead, we actually want to solve $p(z_t = k|x_1..x_T)$ for every t: the probability of a state given all evidence, including evidence in the future, because we recorded the sequence of observations for a whole bunc of data and now want to train the parameters. (This is called the "smoothing" problem.) As we've seen, if you had the following β_t table, the desired posterior will be proportional to $\alpha_t[k]\beta_t[k]$, where

$$\begin{aligned}\beta_t[k] &\equiv P(x_{t+1}..x_T \mid z_t = k) \\ &= \sum_{z_{t+1}..z_T} P(x_{t+1}..x_T,\ z_{t+1}..z_T \mid z_t = k)\end{aligned}$$

and it turns out you can compute this with right-to-left variable elimination, similar to how left-to-right variable elimination was used to create the forward tables.

Sidenote: HMM algorithms are actually used to make airplanes fly and computers understand language. The EM algorithm was originally discovered to train HMM's by Baum and Welch (1970!) for speech recognition. Every time you talk to Siri or whatever, a computer somewhere is running (a variant of) the Viterbi algorithm on your acoustic data. The airplane stuff ("state-space tracking", also for robots, rockets, etc.) isn't actually discrete HMM's as we've learned them, but rather a continuous Gaussian version called Kalman filters (which is where the smoothing/filtering terminology comes from; if you have a copy, see Kevin Murphy's machine learning textbook, chapter 17. http://www.cs.ubc.ca/~murphyk/MLbook/index.html).