HMM algorithm

1 Notation

Hidden state $X_1, X_2, ..., X_M$.

Observed state $Y_1, Y_2, ..., Y_N$.

Observations $Y_1, Y_2, ..., Y_T$.

Transition probability matrix with $T_{ij} = P(X_{t+1} = i | X_t = j) \in \mathbb{R}^{M*M}$

Observation probability matrix with $O_{ij} = P(Y_t = i | X_t = j) \in \mathbb{R}^{N*M}$

Initial state distribution $\pi_i = P(X_1 = i) \in \mathbb{R}^M$

For simplicity, I would use ω to represent $Y_1, ..., Y_T$, and l to represent $X_1, ..., X_T$, and λ to represent the model parameter (T, O, π).

2 Likelihood Computation: The Forward Algorithm

Likelihood of the input: Compute $P(\omega|\lambda)$ for the input ω and HMM λ.

Ex: How likely the sentence 'I went to the mall' occurs?

The likelihood of the input is just the marginal probability which can be computed by summing over all possible tag sequences:

$P(Y_1, ..., Y_T) = \sum_l P(Y_1, ..., Y_T, X_1, ..., X_T) = \sum_l \prod_{i=1}^{T} P(Y_i|X_i)P(X_i|X_{i-1})$

For an HMM with M hidden states and an observation sequence of T observations, there are T^M possible hidden sequences. For real tasks, where M and T are both large, T^M is a very large number, so we cannot compute the total observation likelihood by computing a separate observation likelihood for each hidden state sequence and then summing them.

Instead of using such an extremely exponential algorithm, we use an efficient $O(N^2T)$ algorithm called the forward algorithm. The forward algorithm is a dynamic programming algorithm, that is, an algorithm that uses a table to store intermediate values as it builds up the probability of the observation sequence. The forward algorithm computes the observation probability by summing over the probabilities of all possible hidden state paths that could generate the observation sequence, but it does efficiently by implicitly folding each of these paths into a single forward trellis.

Dynamic programming: Recursively decompose a problem into smaller sub-problems, similar to mathematical induction.

Base step: initial values for $i = 1$.

Inductive step: assume we know the values for $i = k$, lets compute $i = k + 1$.

For the forward algorithm, each cell of the forward algorithm trellis at $\alpha_k(q)$ represents the probability of being in hidden state q after seeing the first k observations:

$\alpha_k(q) = P(Y_1, ..., Y_k, X_k = q)$

And we have the inductive step:

$$\begin{aligned}
&\alpha_k(q) \\
&= P(Y_1, ..., Y_k, X_k = q) \\
&= \sum_{q_0} P(Y_1, ..., Y_k, X_{k-1} = q_0, X_k = q) \\
&= \sum_{q_0} P(Y_1, ..., Y_{k-1}, X_{k-1} = q_0) P(X_k = q | X_{k-1} = q_0) P(Y_k | X_k = q) \\
&= \sum_{q1} \alpha_{k-1}(q_0) P(X_k = q | X_{k-1} = q_0) P(Y_k | X_k = q)
\end{aligned}$$

Finally, we can give a formal definition of the forward recursion as follows:

1. Initialization:

$\alpha_1(q) = p(Y_1, X_1 = q) = P(X_1 = q | X_0) p(Y_1 | X_1 = q)$

2. Recursion:

$\alpha_k(q) = \sum_{q_0} \alpha_{k-1}(q_0) P(X_k = q | X_{k-1} = q_0) P(Y_k | X_k = q)$

3. Termination(calculate the likelihood of the input):

$P(\omega | \lambda) = \sum_{i=1}^{M} \alpha_T(i)$

3 Decoding: The Viterbi Algorithm

Decoding: Given as input an HMM $\lambda = (T, O, \pi)$ and a sequence of observations $\omega = Y_1, ..., Y_T$, find the most probable sequence of states $l = X_1, ..., X_T$?

Inference: What is the most likely sequence of tags for the given sequence of words ω?/ What are the latent states that most likely generate the sequence of word ω?

Ex: What's the POS tags of 'I went to the mall' occurs?

We might propose to find the best sequence as follows: For each possible hidden state sequence, we could run the forward algorithm and compute the likelihood of the observation sequence given that hidden state sequence. Then we could choose the hidden state sequence with the maximum observation likelihood. It should be clear from the previous section that we cannot do this because there are an exponentially large number of state sequences.

Instead, the most common decoding algorithms for HMMs is the Viterbi algorithm. Like the forward algorithm, Viterbi is a kind of dynamic programming that makes uses of a dynamic programming trellis.

The idea is to process the observation sequence left to right, filling out the trellis. Each cell of the trellis, $\theta_k(q)$, represents the probability that the HMM is in state q after seeing the first k observations and passing through the most probable state sequence $X_1, ..., X_{k-1}$. The value of each cell $\theta_k(q)$ is computed by recursively taking the most probable path that could lead us to this cell. Formally, each cell expresses the probability:

$\theta_k(q) = max_q max_{X_1, ...X_{k-1}} P(X_1, ...X_{k-1}, X_k = q, Y_1, ..., Y_k)$

And we have the inductive step:

$$\begin{aligned}
&\theta_k(q) \\
&= max_q max_{X_1, ...X_{k-1}} P(X_1, ...X_{k-1}, X_k = q, Y_1, ..., Y_k) \\
&= max_{q_0} max_{X_1, ...X_{k-2}} P(X_1, ...X_{k-2}, X_{k-1} = q_0, Y_1, ..., Y_{k-1}) P(X_k = q, X_{k-1} = q_0) P(Y_k | X_k = q) \\
&= max_{q_0} \theta_{k-1}(q_0) P(X_k = q | X_{k-1} = q_0) P(Y_k | X_k = q)
\end{aligned}$$

Finally, we can give a formal definition of the Viterbi recursion as follows:

1. Initialization:

$\theta_k(q) = P(Y_1 | X_1 = q) P(X_1 = q | X_0)$

2. Recursion:

$\theta_k(q) = max_{q_0} \theta_{k-1}(q_0) P(X_k = q | X_{k-1} = q_0) P(Y_k | X_k = q)$

$path_k(q) = argmax_{q_0}\theta_{k-1}(q_0)P(X_k = q|X_{k-1} = q_0)P(Y_k|X_k = q)$

3. Termination:

The best score: $P^* = max_{q=1}^{M}\theta_T(q)$

The start of backtrace: $path_T^* = argmax_{q=1}^{M}\theta_T(q)$

4 Backward

The backward probability $\beta_k(C)$ is the probability of seeing the observations from time $k+1$ to the end, given that we are in state C at time k:

$\beta_k(C) = P(Y_{k+1}, ..., Y_n|X_k = C)$

And we have the inductive step:

$$\begin{aligned}&\beta_k(C)\\ &= P(Y_{k+1}, ..., Y_n|X_k = C)\\ &= \sum_q P(P(Y_{k+2}, ..., Y_n|t_{k+1} = q)P(q|C)P(x_{k+1}|q)\\ &= \sum_q \beta_{k+1}(q)P(q|C)P(Y_{k+1}|q)\end{aligned}$$

Finally, we can give a formal definition of the Backward recursion as follows:

1. Initialization:

$\beta_T(C) = 1$

2. Inductive step:

$\beta_k(C) = \sum_q \beta_{k+1}(q)P(q|C)P(Y_{k+1}|q)$

3. Termination:

$P(\omega|\lambda) = \sum_{i=1}^{M} \beta_1(i)\pi(i)P(Y_1|X_1 = i)$

5 Baum Welch

Learning: Given an observation sequence ω and the set of possible states l in the HMM, learn the HMM parameters λ.

The standard algorithm for HMM training is the Baum-Welch algorithm, a special case of the Expectation-Maximization algorithm. Baum-Welch algorithm iteratively estimates the transition counts and emission counts.

The forward and backward probabilities can help us compute the transition probability T_{ij} and observation probability O_{ij} from an observation sequence, even though the actual path taken through the machine is hidden.

Transition counts:

$T_{ij} = \frac{\text{expected number of transitions from state i to state j}}{\text{expected number of transitions from state i}}$

Emission counts:

$O_j(v_k) = \frac{\text{expected number of times in state j and observing symbol}v_k}{\text{expected number of times in state j}}$

If we define

$\xi_t(i, j) = P(X_t = i, X_{t+1} = j|\omega, \lambda)$

$\gamma_t(j) = P(q_t = j|\omega, \lambda)$

Then we can get:

$\xi_t(i, j) = P(X_t = i, X_{t+1} = j, \omega|\lambda)/P(\omega|\lambda)$

$\gamma_t(j) = P(q_t = j, \omega|\lambda)/P(\omega|\lambda)$

In general,

$$\begin{aligned} & P(Y_1, ..., Y_T, X_k = C) \\ = & P(Y_1, ..., Y_k, X_k = C) P(Y_{k+1}, ..., Y_T | X_k = C) \\ = & \alpha_k(C) \beta_k(C) \end{aligned}$$

The first part can be computed by forward algorithm, and the latter part can be computed by backward algorithm.

We also have:

$$\begin{aligned} & P((Y_1, ..., Y_T, X_k = C, X_{k+1} = H) \\ = & P(Y_1, ..., Y_k, X_k = C) P(Y_{k+1}, ..., Y_T | X_{k+1} = H) P(H|C) P(Y_{k+1}|H) \\ = & \alpha_k(C) \beta_{k+1}(H) P(H|C) P(Y_{k+1}|H) \end{aligned}$$

E step:

$\gamma_t(j) = \alpha_t(j)\beta_t(j)/\alpha_T(q_F)$

$\xi_t(i,j) = \alpha_t(i) T_{ij} O_{j,O_{t+1}} \beta_{t+1}(j)/\alpha_T(q_F)$

M step:

$\hat{T_{ij}} = \sum_{t=1}^{T-1} \xi_t(i,j) / \sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i,k)$

$\hat{O_j(v_k)} = \sum_{t=1 s.t. O_t = v_k}^{T} \gamma_t(j) / \sum_{t=1}^{T} \gamma_t(j)$

6 Loss function for Baum Welch

EM optimize the log Likelihood of the input:

$$\begin{aligned} & log P(\omega|\lambda) \\ = & log \sum_l P(\omega, l|\lambda) \\ = & log \sum_l \Pi_{i=1}^{M} P(X_i|X_{i-1}) P(Y_i|X_i) \end{aligned}$$

Then according to Jensen's inequality,

$$\begin{aligned} & log \sum_l (\omega, l|\lambda) \\ = & log \sum_l \frac{(P(\omega,l|\lambda)}{P(l|\omega,\lambda^{(s)})} P(l|\omega,\lambda^{(s)}) \\ \geq & \sum_l P(l|\omega,\lambda^{(s)}) log \frac{(P(\omega,l|\lambda)}{P(l|\omega,\lambda^{(s)})} \end{aligned}$$

Define

$f(\lambda) = log \sum_l P(\omega, l|\lambda)$

$g_s(\lambda) = \sum_l P(l|\omega,\lambda^{(s)}) log \frac{P(\omega,l|\lambda)}{P(l|\omega,\lambda^{(s)})}$

So we have $f(\lambda) \geq g_s(\lambda)$

Optimizing $g_s(\lambda)$

$$\begin{aligned} & g_s(\lambda) \\ = & \sum_l P(l|\omega,\lambda^{(s)}) log \frac{P(\omega,l|\lambda)}{P(l|\omega,\lambda^{(s)})} \\ = & \sum_l P(l|\omega,\lambda^{(s)}) (log P(\omega,l|\lambda) - P(l|\omega,\lambda^{(s)})) \end{aligned}$$

Since the term $P(l|\omega,\lambda^{(s)}))$ doesn't have λ, so we can just ignore this part and find the λ which can maximize the other part:

$$\begin{aligned} & max_\lambda g_s(\lambda) \\ = & max_\lambda \sum_l P(l|\omega,\lambda^{(s)}) log P(\omega,l|\lambda) \\ = & max_\lambda \sum_l P(l|\omega,\lambda^{(s)}) \sum_i (log P(X_i|X_{i-1}) + log P(Y_i|X_i)) \end{aligned}$$

In a word, we can minimize the negative log loss (or maximize the log likelihood $max_\lambda P(\omega|\lambda)$) by maximizing $g_s(\lambda) = \sum_t P(l|\omega, \lambda^s) log P(\omega, l|\lambda)$ at iteration s. $max_\lambda g_s(\lambda)$ has a closed-form solution such that the parameters can be directly calculated by the expected counts.