1 Introduction

Let $\mathbf{x} = (x^1, \ldots, x^M)$ denote a sequence of inputs (e.g., a sequence fo words), and let $\mathbf{y} = (y^1, \ldots, y^M)$ denote the corresponding sequence of outputs (e.g., a sequence of part of speech tags. We use x and y denote a single input and output token, respectively.

In general, we use M to denote the length of a sequence. For simplicity, we assume that $\mathbf{x}$ is encoded using D words, so there are D^M possible length-M sequences $\mathbf{x}$. We also assume that there are are L possible labels for each y^j, so there are L^M possible length-M sequences of $\mathbf{y}$.

Our goal is to use a first-order hidden Markov model (HMM) to model the joint distribution $P(\mathbf{x}, \mathbf{y})$. An HMM is defined as follows:

$$P(\mathbf{x}, \mathbf{y}) = P(\texttt{End}|y^M) \prod_{j=1}^{M} P(x^j|y^j) P(y^j|y^{j-1}), \tag{1}$$

where `End` is a special token denoting the end of a sentence and y^0 is always the special token denoting the start of a sentence. (1) defines a set of conditional independence assumptions. For instance:

$$P(\mathbf{x}|\mathbf{y}) = \prod_{j=1}^{M} P(x^j|y^j),$$

which essentially states that the probability of the individual words are all conditionally independent of each other given the output sequence $\mathbf{y}$.

Note that an HMM model uses the same probability distributions $P(x^j|y^j)$ and $P(y^j|y^{j-1})$ through-out the entire sequence. The total size of an HMM model is:

Component	No. Parameters
$P(x^j\|y^j)$	$D \times L$
$P(y^j\|y^{j-1)}$	$L \times L$
$P(y^1\|y^0)$	L
$P(\texttt{End}\|y^M)$	L

Note also that the probabilities of entire sequences $P(\mathbf{x}, \mathbf{y})$ can often be very small (exponentially small in M), and so it can often be more convenient to work on log-probability space.

$$\log P(\mathbf{x}, \mathbf{y}) = \left(\log P(\texttt{End}|y^M) + \sum_{j=1}^{M} \left(\log P(x^j|y^j) + \log P(y^j|y^{j-1}) \right) \right), \tag{2}$$

Finally, note that given a fixed $\mathbf{x}$, $\forall \mathbf{y} : P(\mathbf{y}|\mathbf{x}) \propto P(\mathbf{x}, \mathbf{y})$. This can be derived via: $P(\mathbf{y}|\mathbf{x}) = P(\mathbf{x}, \mathbf{y})/P(\mathbf{x})$, which has a fixed $P(\mathbf{x})$ in the denominator for all $\mathbf{y}$.

2 Viterbi Algorithm for Making Predictions

Given a test instance $\mathbf{x}$ and a trained HMM model, we make predictions by selecting the $\mathbf{y}$ that maximizes $P(\mathbf{y}|\mathbf{x})$, i.e.,:

$$\begin{aligned} \text{argmax}_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}) &= \text{argmax}_{\mathbf{y}} \log P(\mathbf{y}|\mathbf{x}) = \text{argmax}_{\mathbf{y}} \log P(\mathbf{x}, \mathbf{y}) \\ &= \text{argmax}_{\mathbf{y}} \log P(\texttt{End}|y^M) + \sum_{j=1}^{M} \left(\log P(x^j|y^j) + \log P(y^j|y^{j-1}) \right). \end{aligned} \tag{3}$$

Naively iterating over all possible $\mathbf{y}$ takes exponential time (w.r.t. the length $|\mathbf{x}| = M$). We show here how to use a dynamic programming approach known as the Viterbi algorithm to efficiently compute (3).

Let $\tilde{P}$ denote the log probability, e.g., $\tilde{P}(\mathbf{x}, \mathbf{y}) = \log P(\mathbf{x}, \mathbf{y})$. Define: $\tilde{A}_{ab} = \tilde{P}(y^j = a | y^{j-1} = b)$ and $\tilde{O}_{wz} = \tilde{P}(x^j = w | y^j = z)$. Let $\hat{\mathbf{y}}_a^j$ denote the best length-j prefix solution that ends in $y^j = a$:

$$\hat{\mathbf{y}}_a^j = \left(\text{argmax}_{\mathbf{y}^{1:j-1}} \tilde{P}(\mathbf{y}^{1:j-1} \oplus a, \mathbf{x}^{1:j}) \right) \oplus a, \tag{4}$$

where $\oplus$ denotes sequence concatenation or appending (should be clear from context which one it is).

If one had already computed the length-M prefix solutions $\hat{\mathbf{y}}_a^M$, then one can predict by simply selecting the $\hat{\mathbf{y}}_a^M$ with highest score $\tilde{P}(\hat{\mathbf{y}}_a^M, \mathbf{x})$. For $j = 1$, computing each $\hat{\mathbf{y}}_a^1$ is trivial since by definition $\hat{\mathbf{y}}_a^1 = a$. Via the Viterbi algorithm, one can compute $\hat{\mathbf{y}}_a^j$ for $j > 1$ by just looping through all the $\hat{\mathbf{y}}_b^{j-1}$ and choosing the best one:

$$\begin{aligned} \hat{\mathbf{y}}_a^j &= \left(\text{argmax}_{\mathbf{y}^{1:j-1} \in \{\hat{\mathbf{y}}_1^{j-1}, \ldots, \hat{\mathbf{y}}_L^{j-1}\}} \tilde{P}(\mathbf{y}^{1:j-1} \oplus a, \mathbf{x}^{1:j}) \right) \oplus a \\ &= \left(\text{argmax}_{\mathbf{y}^{1:j-1} \in \{\hat{\mathbf{y}}_1^{j-1}, \ldots, \hat{\mathbf{y}}_L^{j-1}\}} \tilde{P}(\mathbf{y}^{1:j-1}, \mathbf{x}^{1:j-1}) + \tilde{P}(y^j = a | y^{j-1}) + \tilde{P}(x^j | y^j = a) \right) \oplus a. \end{aligned} \tag{5}$$

Computing each $\hat{\mathbf{y}}_a^j$ takes $\mathcal{O}(L)$ running time (enumerating over all $\hat{\mathbf{y}}_a^{j-1}$), and so the total running time of this approach is $\mathcal{O}(L^2 M)$. Note that you need to save the $\tilde{P}(\hat{\mathbf{y}}_a^j, \mathbf{x}^{1:j})$ that you compute along the way while computing each (5), since they are used when computing each $\hat{\mathbf{y}}_a^{j+1}$. Finally, note that you have to operate using log probabilities, because the actual probabilities will underflow for any reasonably sized M.

3 Forward-Backward Algorithm for Computing Marginals

Define a sequence of vectors α^j which will correspond to the unnormalized probability of:

$$\alpha^j(a) \propto \sum_{\mathbf{y}^{1:j-1}} P(\mathbf{y}^{1:j-1} \oplus a, \mathbf{x}^{1:j}) \quad \textit{//ignoring the transition to the End state.}$$

We similarly define a sequence of vectors β^j which will correspond to the unnormalized probability of:

$$\beta^j(b) \propto \sum_{\mathbf{y}^{j+1:M}} P(y^{j+1} | b) P(\mathbf{y}^{j+1:M}, \mathbf{x}^{j+1:M}).$$

We first observe some marginal probabilities we can compute using $\alpha^j(a)$ and $\beta^j(b)$:

$$P(y^j = a, \mathbf{x}) = \frac{\alpha^j(a)\beta^j(a)}{\sum_{a'} \alpha^j(a')\beta^j(a')}, \tag{6}$$

$$P(y^j = a, y^{j+1} = b, \mathbf{x}) = \frac{\alpha^j(a) P(x^{j+1} | y^{j+1} = b) P(y^{j+1} = b | y^j = a) \beta^{j+1}(b)}{\sum_{a',b'} \alpha^j(a') P(x^{j+1} | y^{j+1} = b') P(y^{j+1} = b' | y^j = a') \beta^{j+1}(b')}. \tag{7}$$

Note that these quantities are used in the EM-algorithm for unsupervised training of HMMs.

We will show how to compute each $\alpha^j(a)$ and $\beta^j(b)$ recursively similar to the Viterbi algorithm. We initalize α^0 and β^M as:

$$\alpha^0(a) = \begin{cases} 1 & \text{if } a = \texttt{Start} \\ 0 & \text{otherwise} \end{cases}, \qquad \beta^M(b) = P(\texttt{End} | y^M = b).$$

If you are not using the `End` state, then you can set each $P(\texttt{End} | y^M = b) = 1$ in the above formula.

Define $P_y(b|a) = P(y^{j+1} = b|y^j = a)$, and $P_x(w|z) = P(x^j = w|y^j = z)$. Similar to the Viterbi algorithm, we recursively define each α^j as:

$$\alpha^j(a) = P(x^j|a) \sum_{a'} \alpha^{j-1}(a') P_y(a|a'). \tag{8}$$

We can also recursively define each β^j as:

$$\beta^j(b) = \sum_{b'} \beta^{j+1}(b') P_y(b'|b) P(x^{j+1}|b'). \tag{9}$$

(8) is known as the Forward Algorithm, and (9) is known as the Backward Algorithm.

Dealing With Numerical Instability. In practice directly implementing (8) and (9) leads to numerical instability. For instance, α and β can all overflow or underflow if we compute them as is. The first observation we do not actually need to compute the denominators in (6) and (7) in order to compute those marginal probabilities. All we need to do is to renormalize each α^j and β^j after each step in (8) and (9), i.e.:

$$\tilde{\alpha}^j(a) = \frac{1}{C_\alpha^j} \left(P(x^j|a) \sum_{a'} \alpha^{j-1}(b) P_y(a|a') \right). \tag{10}$$

and

$$\tilde{\beta}^j(b) = \frac{1}{C_\beta^j} \left(\sum_{b'} \beta^{j+1}(b') P_y(b'|b) P(x^{j+1}|b') \right), \tag{11}$$

for some choice of C_α^j and C_β^j such that overflow and underflow do not happen. Afterwards, we can compute (6) and (7) as:

$$P(y^j = a, \mathbf{x}) = \frac{\tilde{\alpha}^j(a)\tilde{\beta}^j(a)}{\sum_{a'} \tilde{\alpha}^j(a')\tilde{\beta}^j(a')}, \tag{12}$$

$$P(y^j = a, y^{j+1} = b, \mathbf{x}) = \frac{\tilde{\alpha}^j(a) P(x^{j+1}|y^{j+1}=b) P(y^{j+1}=b|y^j=a) \tilde{\beta}^{j+1}(b)}{\sum_{a',b'} \tilde{\alpha}^j(a') P(x^{j+1}|y^{j+1}=b') P(y^{j+1}=b'|y^j=a') \tilde{\beta}^{j+1}(b')}. \tag{13}$$

Relationship to Viterbi. Viterbi keeps track of the best sequence of length-j that ends in some $y^j = a$, and also keeps track of the probability of that sequence. The Forward algorithm keeps track of the marginal probability of all sequences of length-j that ends in some $y^j = a$. Thus, the Viterbi takes the max whereas the Forward algorithm takes the sum. Because Viterbi doesn't sum, the probabilities of the single best sequence will shrink exponentially as the sequence grows, hence necessitating taking the log to ensure numerical stability. With the Forward-Backward algorithm, we keep track of the unnormalized probabilities, which can both overflow or underflow, but because it's a product of a bunch of sums, you can normalize at each iteration and still maintain correctness.

4 Training

Supervised Training. In the supervised setting, we are given a training set of N training examples:

$$S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N,$$

and our goal is to learn the parameters of the HMM to maximize the likelihood on S:

$$\text{argmax} \prod_{i=1}^N P(\mathbf{x}_i, \mathbf{y}_i).$$

In this case, training is very straightforward counting:

$$P(y^j = b|y^{j-1} = a) = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M_i}\mathbf{1}_{[y_i^j = b \wedge y_i^{j-1} = a]}}{\sum_{i=1}^{N}\sum_{j=1}^{M_i}\mathbf{1}_{[y_i^{j-1} = a]}}.$$

$$P(x^j = w|y^j = a) = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M_i}\mathbf{1}_{[x_i^j = w \wedge y_i^j = a]}}{\sum_{i=1}^{N}\sum_{j=1}^{M_i}\mathbf{1}_{[y_i^j = a]}}.$$

To see

Unsupervised Training. In the unsupervised setting, we are given a training set of N training examples containing only the $\mathbf{x}$'s:

$$S = \{\mathbf{x}_i\}_{i=1}^{N},$$

and the maximum likelihood problem is thus:

$$\text{argmax}\prod_{i=1}^{N} P(\mathbf{x}_i) = \text{argmax}\prod_{i=1}^{N}\sum_{\mathbf{y}} P(\mathbf{x}_i, \mathbf{y}).$$

We solve this learning problem using an alternating procedure, of first inferring the marginal distribution of the $\mathbf{y}$'s, and then using that to estimate the model parameters. If we new the $\mathbf{y}$'s, then we can use the same approach as the supervised setting:

$$P(y^j = b|y^{j-1} = a) = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M_i} P(y_i^j = b, y_i^{j-1} = a, \mathbf{x}_i)}{\sum_{i=1}^{N}\sum_{j=1}^{M_i} P(y_i^{j-1} = a, \mathbf{x}_i)}. \tag{14}$$

$$P(x^j = w|y^j = a) = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M_i}\mathbf{1}_{[x_i^j = w]} P(y_i^j = a, \mathbf{x}_i)}{\sum_{i=1}^{N}\sum_{j=1}^{M_i} P(y_i^j = a, \mathbf{x}_i)}. \tag{15}$$

Recall that we can estimate these marginal probabilities using the Forward-Backward algorithm. Thus the EM procedure for unsupervised training of HMMs is:

1. INIT: randomly initialize HMM model
2. E-STEP: run Forward-Backward to compute marginal probabilities (6), (7)
3. M-STEP: maximium likelihood estimate of new HMM model parameters (14), (15)
4. If not converged, repeat from Step 2