SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

01.112 Machine Learning, Spring 2018
Lecture Notes for Week 10

17. Hidden Markov Models (IV)

Last update: Thursday 29th March, 2018 12:32

1 Inference (continued)

Today, we continue our discussions on the inference algorithm. We first formally introduce the dynamic programming algorithm used for inference in HMM. The algorithm is called the *forward-backward algorithm.*

The Forward-backward Algorithm

For every state sequence $y_1, y_2, \ldots, y_n$ there is

- A path through the graph (shown in class on the whiteboard) that has the sequence of states $\texttt{START}, \langle 1, y_1 \rangle, \ldots, \langle n, y_n \rangle, \texttt{STOP}$
- The path associated with state sequence $\mathbf{y} = y_1, \ldots, y_n$ has score equal to $p(\mathbf{x}, \mathbf{y}; \theta)$.
- $\alpha_u(j)$ is the sum of scores of all paths from $\texttt{START}$ to the state (j, u).
- $\beta_u(j)$ is the sum of scores of all paths from state $\langle j, u \rangle$ to the final state $\texttt{STOP}$.

Given an input sequence $x_1, \ldots, x_n$ for any $u \in 1, \ldots, N, j \in 1, \ldots, n$ the forward and backward probability can be calculated recursively.

1. Forward probability:

The forward probabilities are defined as:

$$\alpha_u(j) = p(x_1, \ldots, x_{j-1}, y_j = u; \theta) \tag{1}$$

- Base case:

$$\alpha_u(1) = a_{\texttt{START},u} \quad \forall u \in 1, \ldots, N-1 \tag{2}$$

- Recursive case:

$$\alpha_u(j+1) = \sum_v \alpha_v(j) a_{v,u} b_v(x_j) \quad \forall u \in 1, \ldots, N-1, j = 1, \ldots, n-1 \tag{3}$$

2. Backward probability:

The backward probabilities are defined as:

$$\beta_u(j) = p(x_j, \ldots, x_n | y_j = u; \theta) \tag{4}$$

- Base case:

$$\beta_u(n) = a_{u,\texttt{STOP}} b_u(x_n) \quad \forall u \in 1, \ldots, N-1 \tag{5}$$

- Recursive case:

$$\beta_u(j) = \sum_v a_{u,v} b_u(x_j) \beta_v(j+1) \quad \forall u \in 1, \ldots, N-1, j = n-1, \ldots, 1 \tag{6}$$

Discussions What is the time and space complexity of the algorithm?

2 Unsupervised Learning (continued)

Now let us return to the unsupervised learning problem. We are now ready to see how the expected counts can be computed efficiently using the forward and backward scores.

Transition Parameters

Note that Count$(u; v)$ is defined as the expected number of times that we see the (u, v) pair in the *entire* dataset. Assume we have m instances in our dataset, it is easy to see that we can decompose the Count$(u; v)$ as follows:

$$\text{Count}(u; v) = \sum_{i=1}^{m} \text{Count}^{(i)}(u; v)$$

where Count$^{(i)}(u; v)$ is the expected number of times we see the pair (u, v) from the i-th instance.

Assume instance $\mathbf{x}^{(i)}$ has length n, and $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)})$. Assume we have just finished $t-1$ iterations of EM and we have the current estimated model parameter set θ^{t-1} (which consists of emission and transition parameters). We have:

$$\text{Count}^{(i)}(u; v) = \sum_{j=0}^{n} p(y_j = u, y_{j+1} = v | \mathbf{x}^{(i)}; \theta^{t-1}) \tag{7}$$

$$= \sum_{j=0}^{n} \frac{1}{Z} \alpha_u(j) b_u(x_j^{(i)}) a_{u,v} \beta_v(j+1) \tag{8}$$

where

$$Z = \sum_{u'} \alpha_{u'}(j)\beta_{u'}(j)$$

for any choice of $j \in \{1, \ldots, n\}$.

Okay. But what about Count(u)? One idea is: $\sum_v$ Count$(u; v)$, but is there a simpler way to compute such a count? Actually, it is not difficult to see that we have the following:

$$\begin{aligned} \text{Count}^{(i)}(u) &= \sum_{j=1}^{n} p(y_j = u | x^{(i)}; \theta^{t-1}) && (9) \\ &= \sum_{j=1}^{n} \frac{1}{Z} \alpha_u(j)\beta_u(j) && (10) \end{aligned}$$

where

$$Z = \sum_{u'} \alpha_{u'}(j)\beta_{u'}(j)$$

for any choice of $j \in \{1, \ldots, n\}$.

Great. This seems to be much easier to compute given α and β scores.

Emission Parameters

Now let's move to the computation of the expected counts required for re-estimating the emission parameters. Similarly, we have:

$$\text{Count}(u \to o) = \sum_{i=1}^{m} \text{Count}^{(i)}(u \to o)$$

$$\text{Count}^{(i)}(u \to o) = \sum_{j=1}^{n} p(y_j = u, x_j^{(i)} = o | \mathbf{x}^{(i)}; \theta^{t-1}) \quad (11)$$

Hmm, it seems that the p term above would only possibly return you a non-zero value if the j-th observation in $\mathbf{x}^i$ is exactly o. What does this mean? How do we simplify this above expression? What we can do is to actively look for such observation o in the observation sequences instead, and aggregate the counts accordingly. This leads to the following:

$$\begin{aligned} \text{Count}^{(i)}(u \to o) &= \sum_{j=1}^{n} p(y_j = u, x_j^{(i)} = o | \mathbf{x}^{(i)}; \theta^{t-1}) && (12) \\ &= \sum_{j \text{ such that } x_j^{(i)} = o} p(y_j = u, x_j^{(i)} = o | \mathbf{x}^{(i)}; \theta^{t-1}) && (13) \\ &= \sum_{j \text{ such that } x_j^{(i)} = o} \frac{1}{Z} \alpha_u(j)\beta_u(j) && (14) \end{aligned}$$

where

$$Z = \sum_{u'} \alpha_{u'}(j)\beta_{u'}(j)$$

for any choice of $j \in \{1, \ldots, n\}$.

Now we can see that, with the help of the forward and backward probabilities, we are able to compute the expected counts efficiently. This section completes the required operations for the E-step of the soft-EM algorithm for unsupervised learning of HMMs. As we have mentioned earlier, the M-step of the soft-EM algorithm is the same as that of the hard-EM.

(Optional) Connections between Soft and Hard EM

The two types of EM algorithm both come with a theoretical guarantee: they converge to a local maximum of the objective function, which is the joint log-likelihood of the observed data:

$$\sum_i \log p(\mathbf{x}^{(i)}) \tag{15}$$

However, what is the connection between them? We have already said that the M-step for both algorithms are identical, and the only difference is their E-steps. Now let us have a closer look at the E-steps.

In hard EM, during the E-step, we use the Viterbi algorithm to find the most probable $\mathbf{y}$ sequences for each input sequence $\mathbf{x}$, while in soft EM, we use the forward-backward algorithm to find the expected counts. In any case, we are doing the following:

Hard EM find $\arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) = \arg\max_{\mathbf{y}} p(\mathbf{x}, \mathbf{y})$ and use this single $\mathbf{y}$ when computing the counts.

Soft EM compute $p(\mathbf{y}|\mathbf{x})$ and consider each $\mathbf{y}$'s contribution towards the counts.

Now let us look at the following term:

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{x})} = \frac{p(\mathbf{x}, \mathbf{y})}{\sum_{\mathbf{y}'} p(\mathbf{x}, \mathbf{y}')}$$

Now, when we compute the above term, we replace the original (transition and emission) parameters a and b by a^t and b^t respectively (where $t >> 1$, which is sometimes called a *temperature*), we arrive at:

$$p'(\mathbf{y}|\mathbf{x}) = \frac{(p(\mathbf{x}, \mathbf{y}))^t}{\sum_{\mathbf{y}'} (p(\mathbf{x}, \mathbf{y}'))^t}$$

This is because each term (which is either a transition or an emission probability term) inside $p(\mathbf{x}, \mathbf{y})$ or $p(\mathbf{x}, \mathbf{y}')$ will be raised to the power of t.

Now let us see what happens if we increase the temperature t.

$$p'(\mathbf{y}|\mathbf{x}) = \lim_{t \to +\infty} \frac{(p(\mathbf{x}, \mathbf{y}))^t}{\sum_{\mathbf{y}'} (p(\mathbf{x}, \mathbf{y}'))^t} = \begin{cases} 1 & \text{if } \mathbf{y} \equiv \arg\max_{\mathbf{y}} p(\mathbf{x}, \mathbf{y}) \\ 0 & \text{o.w.} \end{cases}$$

What does this mean? In this case, we can see that for each input sequence $\mathbf{x}$ we only consider a single optimal $\mathbf{y}$ sequence, which is the output of the Viterbi decoding algorithm. In other words, we are essentially making an hard assignment to each input sequence $\mathbf{x}$ at the E-step of the EM algorithm. This shows that theoretically the hard EM can be viewed as a special case of the extended version of the soft EM algorithm that is augmented with a temperature t, and when $t \to +\infty$.

3 Max-Marginal Decoding with Forward-backward

We can also do decoding with forward-backward algorithm now. Recall we could use Viterbi algorithm to perform decoding, where we could find the optimal $\mathbf{y}$ sequence for a given $\mathbf{x}$ sequence as follows:

$$y_1^*, y_2^*, \ldots, y_n^* = \arg\max_{y_1, \ldots, y_n} p(y_0, y_1, \ldots, y_{n+1} | x_1, \ldots, x_n; \theta) \tag{16}$$

$$= \arg\max_{y_1, \ldots, y_n} p(x_1, \ldots, x_n, y_0, y_1, \ldots, y_{n+1}; \theta) \tag{17}$$

An alternative idea is to find the optimal sequence using the following approach:

$$y_i^* = \arg\max_{y_i} p(y_i | x_1, \ldots, x_n; \theta) \quad \forall i \in 1, 2, \ldots, n \tag{18}$$

In other words, we are interested in finding the optimal y tag at the position i, given the entire observation sequence $\mathbf{x}$. Let us see how to compute such a marginal probability term:

$$\begin{aligned} p(y_i = u | x_1, \ldots, x_n; \theta) &= \frac{p(x_1, x_2, \ldots, x_{i-1}, y_i = u, x_i, \ldots, x_n; \theta)}{p(x_1, \ldots, x_n; \theta)} \\ &= \frac{p(x_1, x_2, \ldots, x_{i-1}, y_i = u; \theta) p(x_i, \ldots, x_n | y_i = u; \theta)}{p(x_1, \ldots, x_n; \theta)} \\ &= \frac{\alpha_u(i) \beta_u(i)}{\sum_v \alpha_v(j) \beta_v(j)} \end{aligned} \tag{19}$$

with any choice of $j \in \{1, 2, \ldots, n\}$.

Therefore, we have:

$$y_i^* = \arg\max_u p(y_i = u | x_1, \ldots, x_n; \theta) = \arg\max_u \frac{\alpha_u(i) \beta_u(i)}{\sum_v \alpha_v(j) \beta_v(j)} = \arg\max_u \alpha_u(i) \beta_u(i) \tag{20}$$

This shows that we can in fact use the forward-backward algorithm to perform decoding as well, instead of using the Viterbi algorithm. Note that the objective or the goal of the two decoding algorithms are very different. This decoding algorithm is sometimes called *max-marginal decoding*. In practice, however, the Viterbi algorithm is usually preferred over such a decoding strategy, and often Viterbi algorithm yields better solutions.

Discussions What is the time complexity of such a decoding algorithm?

Advanced Topic: Structured Prediction with Hypergraphs

The forward-backward algorithm is a dynamic programming algorithm that works on linear chain structures. The algorithm is in fact a special case of a more general algorithm called the inside-outside algorithm that works on tree structures. The inside-outside algorithm is again a special case of a more general class of algorithm that works on more general structured representations (directed acyclic graphs, or hypergraphs). The algorithm is also used when learning the discriminative counterpart of the models such as Markov random fields, conditional random fields, parsing conditional random fields and hypergraph/forest based models. The algorithms also have intrinsic relations with the *back-propagation* algorithms used in neural networks.

We are looking for researchers to work together on advanced topics related to the above, which can potentially lead to groundbreaking results. If you are interested in learning more, drop me an email to set up a time for a discussion on this research.

Learning Objectives

You need to know:

1. How to learn HMMs using the standard (soft-)EM
2. What are the connections between the hard and soft EM
3. How to do max-marginal decoding with the forward-backward algorithm

Recap W10D2

① Supervised Learning — $a_{uv} = \frac{\text{count}(u,v)}{\text{count}(u)}$ (Find parameters given X, Y.), $b_u(o) = \frac{\text{count}(u \to o)}{\text{count}(u)}$

Supervised Learning — $y^* = \underset{y}{\text{argmax}}\ P(y|x) = \underset{y}{\text{argmax}}\ P(x,y)$ (Find y)

↓

Enumerate all the paths in ē picture.

↓

But more efficient to use Dynamic Programming

↓

Recover path via back-calculating (VITERBI ALGO)

② Unsupervised Learning

W10D4

START → y_1 → y_2 → ... → y_{j-1} → y_j → y_{j+1} → ... → y_n → STOP

y_1 → x_1, y_2 → x_2, y_{j-1} → x_{j-1}, y_j → x_j, y_{j+1} → x_{j+1}, y_n → x_n

$\alpha_u(j)$: $B_u(j)$

To apply count(u, v)

$$= \sum_{j=0}^{n} P(y_j = u,\ y_{j+1} = v \mid x_1 \dots x_n)$$

$$= \sum_{j=0}^{n} \frac{P(x_1, \dots, x_{j-1}, y_j = u, x_j, y_{j+1} = v, x_{j+1}, \dots x_n}{P(x_1 \dots x_n)}$$

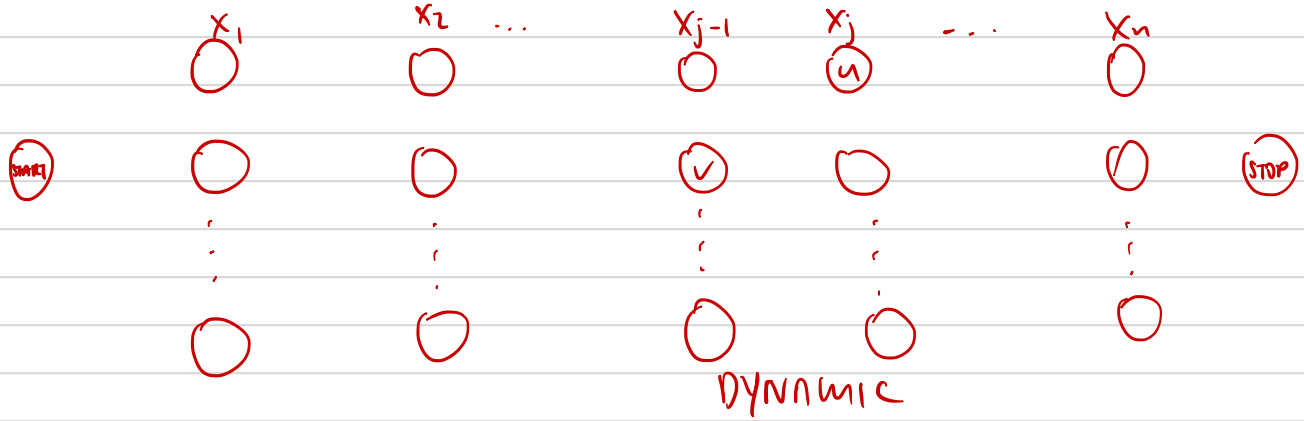
$$= \sum_{j=0}^{n} \frac{\alpha_u(j) \cdot b_u(x_j) \cdot a_{u,v} \cdot B_v(j+1)}{\sum_{u'} \alpha_{u'}(k) \cdot B_{u'}(k)}$$

START → y_1 → y_2 → ... → y_{j-1} → y_j → y_{j+1} → ... → y_n → STOP

$\alpha_u(j)$ ← u, v → $B_v(j+1)$

y_1 → x_1, y_2 → x_2, y_{j-1} → x_{j-1}, y_j → x_j, y_{j+1} → x_{j+1}, y_n → x_n

EG



DYNAMIC

Steps for Dynamic Programming

FORWARD ALGO

↓ going backward

① $\alpha_{u(1)} = a_{START,u}$

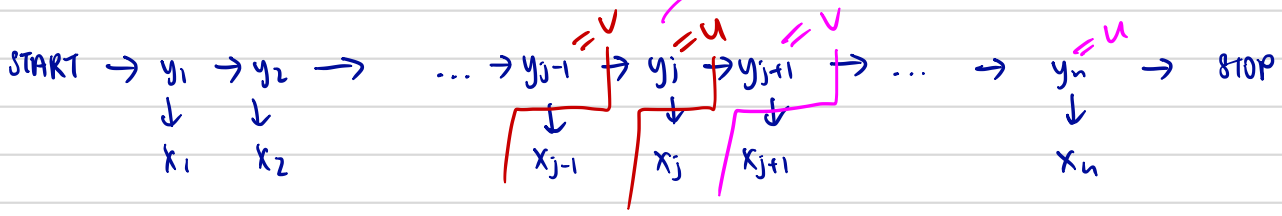
② $\alpha_{u(j)} = P(X_1 \dots, X_{j-1}, y_j = u)$

$= \sum_v P(x_1 \dots x_{j-2}, y_{j-1} = v, x_{j-1}, y_j = u)$

$\alpha_v(j-1)$ $b_v(x_{j-1})$

$= \sum_v \alpha_v(j-1) \cdot b_v(x_{j-1}) \cdot a_{vu}$

$\alpha_u(j) = \sum_v \alpha_v(j-1) \cdot b_v(x_{j-1}) \cdot a_{vu}$



① $\beta_u(n) = a_{u,STOP} \cdot b_u(x_n) = a_{u,STOP} \cdot 1$

② $\beta_u(j) = P(X_j \dots X_n \mid y_j = u)$

$= \sum_v P(x_j \dots x_n, y_{j+1} = v \mid y_j = u)$

$= \sum_v p(x_j \mid y_j = u) \cdot p(y_{j+1} = v \mid y_j = u) \cdot p(x_{j+1} \dots x_n \mid y_{j+1} = v)$

$= b_u(x_j) \cdot a_{uv} \cdot \sum \beta_v(j+1)$ for $j = n-1, \dots, 1$, for all u

$= \sum_v \beta_v(j+1) \cdot b_u(x_j) \cdot a_{uv}$

Space Complexity : $O(n) = O(nT)$ ← no of nodes involved

Time Complexity : $O(n) = O(nT^2)$

Summary

$$a_{u,v} = \frac{\text{count}(u,v)}{\text{count}(u)} = \frac{\sum_{i=1}^{m} \sum_{j=0}^{n} \frac{\alpha_u(j) \cdot a_{uv} \cdot b_u(x_j) \cdot \beta_v(j+1)}{\sum_{u'} \alpha_{u'}(k) \beta_{u'}(k)}}{\sum_{i=1}^{m} \sum_{j=1}^{n} \frac{\alpha_u(j) \cdot \beta_u(j)}{\sum_{u'} \alpha_{u'}(k) \cdot \beta_{u'}(k)}}$$

↗ forward + backward + soft EM. ↗ observation

$$b_u(0) = \frac{\text{count}(u \to 0)}{\text{count}(u)} = \frac{\sum_{i=1}^{n} P(y_j = u, x_j = 0 \mid x_1 \dots x_n)}{\text{count}(u)}$$

$$= \frac{\sum_{j \text{ st } x_j = 0} \frac{P(x_1, \dots x_{j-1}, y_j = u, x_j, \dots, x_n)}{\sum_{u'} \alpha_{u'}(u) \beta_{u'}(u)}}{\text{count}(u)} = \frac{\sum_{j \text{ st } x_j = 0} \frac{P(x_1 \dots x_{j-1}, y_j = u) P(x_j \dots x_n \mid y_j = u)}{\sum_{u'} \alpha_{u'}(u) \beta_{u'}(u)}}{\text{count}(u)}$$

$$= \frac{\sum_{j \text{ st } x_j = 0} \frac{\alpha_u(j) \cdot \beta_u(j)}{\sum_{u'} \alpha_{u'}(u) \beta_{u'}(u)}}{\text{count}(u)}$$