Wikipedia
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Español
Français
Italiano
Русский
Edit links

Forward–backward algorithm

From Wikipedia, the free encyclopedia
(Redirected from Forward-backward algorithm)

The **forward–backward algorithm** is an inference algorithm for hidden Markov models which computes the posterior marginals of all hidden state variables given a sequence of observations/emissions $o_{1:t} := o_1, \dots, o_t$, i.e. it computes, for all hidden state variables $X_k \in \{X_1, \dots, X_t\}$, the distribution $P(X_k \mid o_{1:t})$. This inference task is usually called *smoothing*. The algorithm makes use of the principle of dynamic programming to compute efficiently the values that are required to obtain the posterior marginal distributions in two passes. The first pass goes forward in time while the second goes backward in time; hence the name *forward–backward algorithm*.

The term *forward–backward algorithm* is also used to refer to any algorithm belonging to the general class of algorithms that operate on sequence models in a forward–backward manner. In this sense, the descriptions in the remainder of this article refer but to one specific instance of this class.

Contents [hide]

Overview [edit]

In the first pass, the forward–backward algorithm computes a set of forward probabilities which provide, for all $k \in \{1, \dots, t\}$, the probability of ending up in any particular state given the first k observations in the sequence, i.e. $P(X_k \mid o_{1:k})$. In the second pass, the algorithm computes a set of backward probabilities which provide the probability of observing the remaining observations given any starting point k, i.e. $P(o_{k+1:t} \mid X_k)$. These two sets of probability distributions can then be combined to obtain the distribution over states at any specific point in time given the entire observation sequence:

$$P(X_k \mid o_{1:t}) = P(X_k \mid o_{1:k}, o_{k+1:t}) \propto P(o_{k+1:t} \mid X_k) P(X_k \mid o_{1:k})$$

The last step follows from an application of the Bayes' rule and the conditional independence of $o_{k+1:t}$ and $o_{1:k}$ given X_k.

As outlined above, the algorithm involves three steps:

1. computing forward probabilities
2. computing backward probabilities
3. computing smoothed values.

The forward and backward steps may also be called "forward message pass" and "backward message pass" - these terms are due to the *message-passing* used in general belief propagation approaches. At each single observation in the sequence, probabilities to be used for calculations at the next observation are computed. The smoothing step can be calculated simultaneously during the backward pass. This step allows the algorithm to take into account any past observations of output for computing more accurate results.

The forward–backward algorithm can be used to find the most likely state for any point in time. It cannot, however, be used to find the most likely sequence of states (see Viterbi algorithm).

Forward probabilities [edit]

The following description will use matrices of probability values rather than probability distributions, although in general the forward-backward algorithm can be applied to continuous as well as discrete probability models.

We transform the probability distributions related to a given hidden Markov model into matrix notation as follows. The transition probabilities $\mathbf{P}(X_t \mid X_{t-1})$ of a given random variable X_t representing all possible states in the hidden Markov model will be represented by the matrix $\mathbf{T}$ where the row index, i, will represent the start state and the column index, j, represents the target state. The example below represents a system where the probability of staying in the same state after each step is 70% and the probability of transitioning to the other state is 30%. The transition matrix is then:

$$\mathbf{T} = \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$$

In a typical Markov model we would multiply a state vector by this matrix to obtain the probabilities for the subsequent state. In a hidden Markov model the state is unknown, and we instead observe events associated with the possible states. An event matrix of the form:

$$\mathbf{B} = \begin{pmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{pmatrix}$$

provides the probabilities for observing events given a particular state. In the above example, event 1 will be observed 90% of the time if we are in state 1 while event 2 has a 10% probability of occurring in this state. In contrast, event 1 will only be observed 20% of the time if we are in state 2 and event 2 has an 80% chance of occurring. Given a state vector (π), the probability of observing event j is then:

$$\mathbf{P}(O = j) = \sum_i \pi_i b_{i,j}$$

This can be represented in matrix form by multiplying the state vector (π) by an observation matrix ($\mathbf{O_j} = \mathrm{diag}(b_{*,o_j})$) containing only diagonal entries. Each entry is the probability of the observed event given each state. Continuing the above example, an observation of event 1 would be:

$$\mathbf{O_1} = \begin{pmatrix} 0.9 & 0.0 \\ 0.0 & 0.2 \end{pmatrix}$$

This allows us to calculate the probabilities associated with transitioning to a new state and observing the given event as:

$$\mathbf{f_{0:1}} = \pi \mathbf{O_1}$$

The probability vector that results contains entries indicating the probability of transitioning to each state and observing the given event. This process can be carried forward with additional observations using:

$$\mathbf{f_{0:t}} = \mathbf{f_{0:t-1}} \mathbf{T} \mathbf{O_t}$$

This value is the forward probability vector. The i'th entry of this vector provides:

$$\mathbf{f_{0:t}}(i) = \mathbf{P}(o_1, o_2, \dots, o_t, X_t = x_i | \pi)$$

Typically, we will normalize the probability vector at each step so that its entries sum to 1. A scaling factor is thus introduced at each step such that:

$$\mathbf{\hat{f}_{0:t}} = c_t^{-1}\ \mathbf{\hat{f}_{0:t-1}} \mathbf{T} \mathbf{O_t}$$

where $\mathbf{\hat{f}_{0:t-1}}$ represents the scaled vector from the previous step and c_t represents the scaling factor that causes the resulting vector's entries to sum to 1. The product of the scaling factors is the total probability for observing the given events irrespective of the final states:

$$\mathbf{P}(o_1, o_2, \dots, o_t | \pi) = \prod_{s=1}^{t} c_s$$

This allows us to interpret the scaled probability vector as:

$$\mathbf{\hat{f}_{0:t}}(i) = \frac{\mathbf{f_{0:t}}(i)}{\prod_{s=1}^{t} c_s} = \frac{\mathbf{P}(o_1, o_2, \dots, o_t, X_t = x_i | \pi)}{\mathbf{P}(o_1, o_2, \dots, o_t | \pi)} = \mathbf{P}(X_t = x_i | o_1, o_2, \dots, o_t, \pi)$$

We thus find that the product of the scaling factors provides us with the total probability for observing the given sequence up to time t and that the scaled probability vector provides us with the probability of being in each state at this time.

Backward probabilities [edit]

A similar procedure can be constructed to find backward probabilities. These intend to provide the probabilities:

$$\mathbf{b_{t:T}}(i) = \mathbf{P}(o_{t+1}, o_{t+2}, \dots, o_T | X_t = x_i)$$

That is, we now want to assume that we start in a particular state ($X_t = x_i$), and we are now interested in the probability of observing all future events from this state. Since the initial state is assumed as given (i.e. the prior probability of this state = 100%), we begin with:

$$\mathbf{b_{T:T}} = [1\ 1\ 1\ \dots]^T$$

Notice that we are now using a column vector while the forward probabilities used row vectors. We can then work backwards using:

$$\mathbf{b_{t-1:T}} = \mathbf{T}\mathbf{O_t}\mathbf{b_{t:T}}$$

While we could normalize this vector as well so that its entries sum to one, this is not usually done. Noting that each entry contains the probability of the future event sequence given a particular initial state, normalizing this vector would be equivalent to applying Bayes' theorem to find the likelihood of each initial state given the future events (assuming uniform priors for the final state vector). However, it is more common to scale this vector using the same c_t constants used in the forward probability calculations. $\mathbf{b_{T:T}}$ is not scaled, but subsequent operations use:

$$\mathbf{\hat{b}_{t-1:T}} = c_t^{-1}\mathbf{T}\mathbf{O_t}\mathbf{\hat{b}_{t:T}}$$

where $\mathbf{\hat{b}_{t:T}}$ represents the previous, scaled vector. This result is that the scaled probability vector is related to the backward probabilities by:

$$\mathbf{\hat{b}_{t:T}}(i) = \frac{\mathbf{b_{t:T}}(i)}{\prod_{s=t+1}^{T} c_s}$$

This is useful because it allows us to find the total probability of being in each state at a given time, t, by multiplying these values:

$$\gamma_{\mathbf{t}}(i) = \mathbf{P}(X_t = x_i | o_1, o_2, \dots, o_T, \pi) = \frac{\mathbf{P}(o_1, o_2, \dots, o_T, X_t = x_i | \pi)}{\mathbf{P}(o_1, o_2, \dots, o_T | \pi)} = \frac{\mathbf{f_{0:t}}(i) \cdot \mathbf{b_{t:T}}(i)}{\prod_{s=1}^{T} c_s} = \hat{\mathbf{f}}_{\mathbf{0:t}}(i) \cdot \hat{\mathbf{b}}_{\mathbf{t:T}}(i)$$

To understand this, we note that $\mathbf{f_{0:t}}(i) \cdot \mathbf{b_{t:T}}(i)$ provides the probability for observing the given events in a way that passes through state x_i at time t. This probability includes the forward probabilities covering all events up to time t as well as the backward probabilities which include all future events. This is the numerator we are looking for in our equation, and we divide by the total probability of the observation sequence to normalize this value and extract only the probability that $X_t = x_i$. These values are sometimes called the "smoothed values" as they combine the forward and backward probabilities to compute a final probability.

The values $\gamma_{\mathbf{t}}(i)$ thus provide the probability of being in each state at time t. As such, they are useful for determining the most probable state at any time. It should be noted, however, that the term "most probable state" is somewhat ambiguous. While the most probable state is the most likely to be correct at a given point, the sequence of individually probable states is not likely to be the most probable sequence. This is because the probabilities for each point are calculated independently of each other. They do not take into account the transition probabilities between states, and it is thus possible to get states at two moments (t and t+1) that are both most probable at those time points but which have very little probability of occurring together, i.e. $\mathbf{P}(X_t = x_i, X_{t+1} = x_j) \neq \mathbf{P}(X_t = x_i)\mathbf{P}(X_{t+1} = x_j)$. The most probable sequence of states that produced an observation sequence can be found using the Viterbi algorithm.

Example [edit]

This example takes as its basis the umbrella world in Russell & Norvig 2010 Chapter 15 pp. 566 in which we would like to infer the weather given observation of a man either carrying or not carrying an umbrella. We assume two possible states for the weather: state 1 = rain, state 2 = no rain. We assume that the weather has a 70% chance of staying the same each day and a 30% chance of changing. The transition probabilities are then:

$$\mathbf{T} = \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$$

We also assume each state generates 2 events: event 1 = umbrella, event 2 = no umbrella. The conditional probabilities for these occurring in each state are given by the probability matrix:

$$\mathbf{B} = \begin{pmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{pmatrix}$$

We then observe the following sequence of events: {umbrella, umbrella, no umbrella, umbrella, umbrella} which we will represent in our calculations as:

$$\mathbf{O_1} = \begin{pmatrix} 0.9 & 0.0 \\ 0.0 & 0.2 \end{pmatrix} \; \mathbf{O_2} = \begin{pmatrix} 0.9 & 0.0 \\ 0.0 & 0.2 \end{pmatrix} \; \mathbf{O_3} = \begin{pmatrix} 0.1 & 0.0 \\ 0.0 & 0.8 \end{pmatrix} \; \mathbf{O_4} = \begin{pmatrix} 0.9 & 0.0 \\ 0.0 & 0.2 \end{pmatrix} \; \mathbf{O_5} = \begin{pmatrix} 0.9 & 0.0 \\ 0.0 & 0.2 \end{pmatrix}$$

Note that $\mathbf{O_3}$ differs from the others because of the "no umbrella" observation.

In computing the forward probabilities we begin with:

$$\mathbf{f_{0:0}} = \begin{pmatrix} 0.5 & 0.5 \end{pmatrix}$$

which is our prior state vector indicating that we don't know which state the weather is in before our observations. While a state vector should be given as a row vector, we will use the transpose of the matrix so that the calculations below are easier to read. Our calculations are then written in the form:

$$(\hat{\mathbf{f}}_{\mathbf{0:t}})^T = c^{-1}\mathbf{O_t}(\mathbf{T})^T(\hat{\mathbf{f}}_{\mathbf{0:t-1}})^T$$

instead of:

$$\hat{\mathbf{f}}_{\mathbf{0:t}} = c^{-1}\hat{\mathbf{f}}_{\mathbf{0:t-1}}\mathbf{T}\mathbf{O_t}$$

Notice that the transformation matrix is also transposed, but in our example the transpose is equal to the original matrix. Performing these calculations and normalizing the results provides:

$$(\hat{\mathbf{f}}_{\mathbf{0:1}})^T = c_1^{-1}\begin{pmatrix} 0.9 & 0.0 \\ 0.0 & 0.2 \end{pmatrix}\begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}\begin{pmatrix} 0.5000 \\ 0.5000 \end{pmatrix} = c_1^{-1}\begin{pmatrix} 0.4500 \\ 0.1000 \end{pmatrix} = \begin{pmatrix} 0.8182 \\ 0.1818 \end{pmatrix}$$

$$(\hat{\mathbf{f}}_{\mathbf{0:2}})^T = c_2^{-1}\begin{pmatrix} 0.9 & 0.0 \\ 0.0 & 0.2 \end{pmatrix}\begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}\begin{pmatrix} 0.8182 \\ 0.1818 \end{pmatrix} = c_2^{-1}\begin{pmatrix} 0.5645 \\ 0.0745 \end{pmatrix} = \begin{pmatrix} 0.8834 \\ 0.1166 \end{pmatrix}$$

$$(\hat{\mathbf{f}}_{\mathbf{0:3}})^T = c_3^{-1}\begin{pmatrix} 0.1 & 0.0 \\ 0.0 & 0.8 \end{pmatrix}\begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}\begin{pmatrix} 0.8834 \\ 0.1166 \end{pmatrix} = c_3^{-1}\begin{pmatrix} 0.0653 \\ 0.2772 \end{pmatrix} = \begin{pmatrix} 0.1907 \\ 0.8093 \end{pmatrix}$$

$$(\hat{\mathbf{f}}_{\mathbf{0:4}})^T = c_4^{-1}\begin{pmatrix} 0.9 & 0.0 \\ 0.0 & 0.2 \end{pmatrix}\begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}\begin{pmatrix} 0.1907 \\ 0.8093 \end{pmatrix} = c_4^{-1}\begin{pmatrix} 0.3386 \\ 0.1247 \end{pmatrix} = \begin{pmatrix} 0.7308 \\ 0.2692 \end{pmatrix}$$

$$(\hat{\mathbf{f}}_{\mathbf{0:5}})^T = c_5^{-1}\begin{pmatrix} 0.9 & 0.0 \\ 0.0 & 0.2 \end{pmatrix}\begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}\begin{pmatrix} 0.7308 \\ 0.2692 \end{pmatrix} = c_5^{-1}\begin{pmatrix} 0.5331 \\ 0.0815 \end{pmatrix} = \begin{pmatrix} 0.8673 \\ 0.1327 \end{pmatrix}$$

For the backward probabilities we start with:

$$\mathbf{b_{5:5}} = \begin{pmatrix} 1.0 \\ 1.0 \end{pmatrix}$$

We are then able to compute (using the observations in reverse order and normalizing with different constants):

$$\hat{\mathbf{b}}_{4:5} = \alpha\begin{pmatrix}0.7 & 0.3\\0.3 & 0.7\end{pmatrix}\begin{pmatrix}0.9 & 0.0\\0.0 & 0.2\end{pmatrix}\begin{pmatrix}1.0000\\1.0000\end{pmatrix} = \alpha\begin{pmatrix}0.6900\\0.4100\end{pmatrix} = \begin{pmatrix}0.6273\\0.3727\end{pmatrix}$$

$$\hat{\mathbf{b}}_{3:5} = \alpha\begin{pmatrix}0.7 & 0.3\\0.3 & 0.7\end{pmatrix}\begin{pmatrix}0.9 & 0.0\\0.0 & 0.2\end{pmatrix}\begin{pmatrix}0.6273\\0.3727\end{pmatrix} = \alpha\begin{pmatrix}0.4175\\0.2215\end{pmatrix} = \begin{pmatrix}0.6533\\0.3467\end{pmatrix}$$

$$\hat{\mathbf{b}}_{2:5} = \alpha\begin{pmatrix}0.7 & 0.3\\0.3 & 0.7\end{pmatrix}\begin{pmatrix}0.1 & 0.0\\0.0 & 0.8\end{pmatrix}\begin{pmatrix}0.6533\\0.3467\end{pmatrix} = \alpha\begin{pmatrix}0.1289\\0.2138\end{pmatrix} = \begin{pmatrix}0.3763\\0.6237\end{pmatrix}$$

$$\hat{\mathbf{b}}_{1:5} = \alpha\begin{pmatrix}0.7 & 0.3\\0.3 & 0.7\end{pmatrix}\begin{pmatrix}0.9 & 0.0\\0.0 & 0.2\end{pmatrix}\begin{pmatrix}0.3763\\0.6237\end{pmatrix} = \alpha\begin{pmatrix}0.2745\\0.1889\end{pmatrix} = \begin{pmatrix}0.5923\\0.4077\end{pmatrix}$$

$$\hat{\mathbf{b}}_{0:5} = \alpha\begin{pmatrix}0.7 & 0.3\\0.3 & 0.7\end{pmatrix}\begin{pmatrix}0.9 & 0.0\\0.0 & 0.2\end{pmatrix}\begin{pmatrix}0.5923\\0.4077\end{pmatrix} = \alpha\begin{pmatrix}0.3976\\0.2170\end{pmatrix} = \begin{pmatrix}0.6469\\0.3531\end{pmatrix}$$

Finally, we will compute the smoothed probability values. These result also must be scaled so that its entries sum to 1 because we did not scale the backward probabilities with the c_t's found earlier. The backward probability vectors above thus actually represent the likelihood of each state at time t given the future observations. Because these vectors are proportional to the actual backward probabilities, the result has to be scaled an additional time.

$$(\gamma_0)^T = \alpha\begin{pmatrix}0.5000\\0.5000\end{pmatrix} \circ \begin{pmatrix}0.6469\\0.3531\end{pmatrix} = \alpha\begin{pmatrix}0.3235\\0.1765\end{pmatrix} = \begin{pmatrix}0.6469\\0.3531\end{pmatrix}$$

$$(\gamma_1)^T = \alpha\begin{pmatrix}0.8182\\0.1818\end{pmatrix} \circ \begin{pmatrix}0.5923\\0.4077\end{pmatrix} = \alpha\begin{pmatrix}0.4846\\0.0741\end{pmatrix} = \begin{pmatrix}0.8673\\0.1327\end{pmatrix}$$

$$(\gamma_2)^T = \alpha\begin{pmatrix}0.8834\\0.1166\end{pmatrix} \circ \begin{pmatrix}0.3763\\0.6237\end{pmatrix} = \alpha\begin{pmatrix}0.3324\\0.0728\end{pmatrix} = \begin{pmatrix}0.8204\\0.1796\end{pmatrix}$$

$$(\gamma_3)^T = \alpha\begin{pmatrix}0.1907\\0.8093\end{pmatrix} \circ \begin{pmatrix}0.6533\\0.3467\end{pmatrix} = \alpha\begin{pmatrix}0.1246\\0.2806\end{pmatrix} = \begin{pmatrix}0.3075\\0.6925\end{pmatrix}$$

$$(\gamma_4)^T = \alpha\begin{pmatrix}0.7308\\0.2692\end{pmatrix} \circ \begin{pmatrix}0.6273\\0.3727\end{pmatrix} = \alpha\begin{pmatrix}0.4584\\0.1003\end{pmatrix} = \begin{pmatrix}0.8204\\0.1796\end{pmatrix}$$

$$(\gamma_5)^T = \alpha\begin{pmatrix}0.8673\\0.1327\end{pmatrix} \circ \begin{pmatrix}1.0000\\1.0000\end{pmatrix} = \alpha\begin{pmatrix}0.8673\\0.1327\end{pmatrix} = \begin{pmatrix}0.8673\\0.1327\end{pmatrix}$$

Notice that the value of γ_0 is equal to $\hat{\mathbf{b}}_{0:5}$ and that γ_5 is equal to $\hat{\mathbf{f}}_{0:5}$. This follows naturally because both $\hat{\mathbf{f}}_{0:5}$ and $\hat{\mathbf{b}}_{0:5}$ begin with uniform priors over the initial and final state vectors (respectively) and take into account all of the observations. However, γ_0 will only be equal to $\hat{\mathbf{b}}_{0:5}$ when our initial state vector represents a uniform prior (i.e. all entries are equal). When this is not the case $\hat{\mathbf{b}}_{0:5}$ needs to be combined with the initial state vector to find the most likely initial state. We thus find that the forward probabilities by themselves are sufficient to calculate the most likely final state. Similarly, the backward probabilities can be combined with the initial state vector to provide the most probable initial state given the observations. The forward and backward probabilities need only be combined to infer the most probable states between the initial and final points.

The calculations above reveal that the most probable weather state on every day except for the third one was "rain." They tell us more than this, however, as they now provide a way to quantify the probabilities of each state at different times. Perhaps most importantly, our value at γ_5 quantifies our knowledge of the state vector at the end of the observation sequence. We can then use this to predict the probability of the various weather states tomorrow as well as the probability of observing an umbrella.

Performance [edit]

The brute-force procedure for the solution of this problem is the generation of all possible N^T state sequences and calculating the joint probability of each state sequence with the observed series of events. This approach has time complexity $O(T \cdot N^T)$, where T is the length of sequences and N is the number of symbols in the state alphabet. This is intractable for realistic problems, as the number of possible hidden node sequences typically is extremely high. However, the forward–backward algorithm has time complexity $O(N^2 T)$.

An enhancement to the general forward-backward algorithm, called the Island algorithm, trades smaller memory usage for longer running time, taking $O(N^2 T \log T)$ time and $O(N^2 \log T)$ memory. On a computer with an unlimited number of processors, this can be reduced to $O(N^2 T)$ total time, while still taking only $O(N^2 \log T)$ memory.[1]

In addition, algorithms have been developed to compute $\mathbf{f}_{0:t+1}$ efficiently through online smoothing such as the fixed-lag smoothing (FLS) algorithm Russell & Norvig 2010 Figure 15.6 pp. 580.

Pseudocode [edit]

```
ForwardBackward(guessState, sequenceIndex):
    if sequenceIndex is past the end of the sequence, return 1
    if (guessState, sequenceIndex) has been seen before, return saved result
    result = 0
    for each neighboring state n:
        result = result + (transition probability from guessState to
                           n given observation element at sequenceIndex)
                        * ForwardBackward(n, sequenceIndex+1)
    save result for (guessState, sequenceIndex)
    return result
```

Python example [edit]

Given HMM (just like in Viterbi algorithm) represented in the Python programming language:

```
states = ('Healthy', 'Fever')
end_state = 'E'

observations = ('normal', 'cold', 'dizzy')

start_probability = {'Healthy': 0.6, 'Fever': 0.4}

transition_probability = {
   'Healthy' : {'Healthy': 0.69, 'Fever': 0.3, 'E': 0.01},
   'Fever' : {'Healthy': 0.4, 'Fever': 0.59, 'E': 0.01},
   }

emission_probability = {
   'Healthy' : {'normal': 0.5, 'cold': 0.4, 'dizzy': 0.1},
   'Fever' : {'normal': 0.1, 'cold': 0.3, 'dizzy': 0.6},
   }
```

We can write implementation like this:

```
def fwd_bkw(x, states, a_0, a, e, end_st):
    L = len(x)

    fwd = []
    f_prev = {}
    # forward part of the algorithm
    for i, x_i in enumerate(x):
        f_curr = {}
        for st in states:
            if i == 0:
                # base case for the forward part
                prev_f_sum = a_0[st]
            else:
                prev_f_sum = sum(f_prev[k]*a[k][st] for k in states)

            f_curr[st] = e[st][x_i] * prev_f_sum

        sum_prob = sum(f_curr.values())
        for st in states:
            f_curr[st] /= sum_prob # normalising to make sum == 1

        fwd.append(f_curr)
        f_prev = f_curr

    p_fwd = sum(f_curr[k]*a[k][end_st] for k in states)

    bkw = []
    b_prev = {}
    # backward part of the algorithm
    for i, x_i_plus in enumerate(reversed(x[1:]+(None,))):
        b_curr = {}
        for st in states:
            if i == 0:
                # base case for backward part
                b_curr[st] = a[st][end_st]
            else:
                b_curr[st] = sum(a[st][l]*e[l][x_i_plus]*b_prev[l] for l in states)


        sum_prob = sum(b_curr.values())
        for st in states:
            b_curr[st] /= sum_prob # normalising to make sum == 1

        bkw.insert(0,b_curr)
        b_prev = b_curr

    p_bkw = sum(a_0[l] * e[l][x[0]] * b_curr[l] for l in states)

    # merging the two parts
    posterior = []
    for i in range(L):
        posterior.append({st: fwd[i][st]*bkw[i][st]/p_fwd for st in states})

    assert p_fwd == p_bkw
    return fwd, bkw, posterior
```

The function `fwd_bkw` takes the following arguments: `x` is the sequence of observations, e.g. `['normal', 'cold', 'dizzy']`; `states` is the set of hidden states; `a_0` is the start probability; `a` are the transition probabilities; and `e` are the emission probabilities.

For simplicity of code, we assume that the observation sequence `x` is non-empty and that `a[i][j]` and `e[i][j]` is defined for all states i,j.

In the running example, the forward-backward algorithm is used as follows:

```
def example():
    return fwd_bkw(observations,
                   states,
                   start_probability,
                   transition_probability,
                   emission_probability,
                   end_state)

for line in example():
    print(' '.join(map(str, line)))
```

See also [edit]

- Baum-Welch algorithm
- Viterbi algorithm
- BCJR algorithm

References [edit]

1. ^ J. Binder, K. Murphy and S. Russell. Space-Efficient Inference in Dynamic Probabilistic Networks. Int'l, Joint Conf. on Artificial Intelligence, 1997.

- Lawrence R. Rabiner, A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77 (2), p. 257–286, February 1989. 10.1109/5.18626
- Lawrence R. Rabiner, B. H. Juang (January 1986). "An introduction to hidden Markov models". *IEEE ASSP Magazine*: 4–15.
- Eugene Charniak (1993). *Statistical Language Learning*. Cambridge, Massachusetts: MIT Press. ISBN 978-0-262-53141-2.
- *Stuart Russell and Peter Norvig (2010). Artificial Intelligence A Modern Approach 3rd Edition. Upper Saddle River, New Jersey: Pearson Education/Prentice-Hall. ISBN 978-0-13-604259-4.*

External links [edit]

- An interactive spreadsheet for teaching the forward–backward algorithm (spreadsheet and article with step-by-step walk-through)
- Tutorial of hidden Markov models including the forward–backward algorithm
- Collection of AI algorithms implemented in Java (including HMM and the forward–backward algorithm)

Categories: Dynamic programming | Error detection and correction | Machine learning algorithms | Markov models