Basics of Markov and hidden Markov models and their application in information processing

According to an excerpt from the book "Speech and Language Processing", 3rd edition,

by Dan Jurafsky and James H. Martin, Stanford University, USA

Prepared : Prof. Davor Petrinović

University of Zagreb, Faculty of Electrical Engineering and Computing

Zagreb, November 2022.

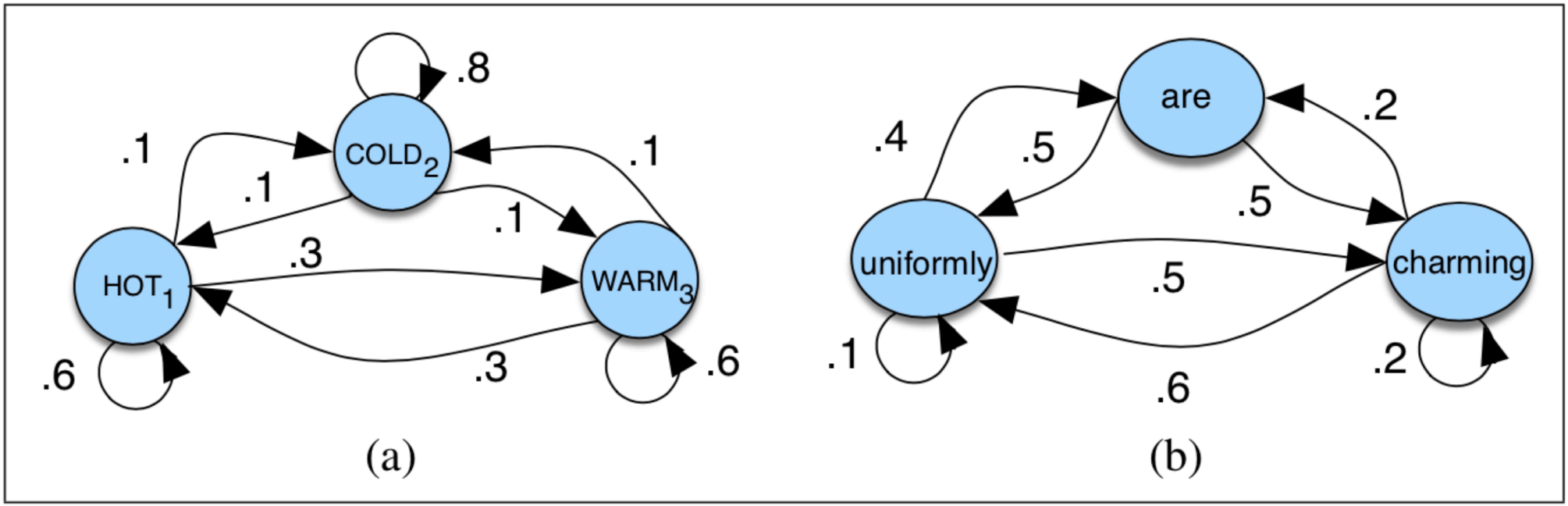
The source used in the preparation of the lecture material for the IP course

- Hundreds of books and review papers are devoted to the topic of Markov and hidden Markov models, so the available literature is truly extremely extensive and rich.
- The electronic book entitled "*Speech and Language Processing*", 3rd ed., was chosen as the most suitable source for this subject by Dan Jurafsky and James H. Martin of Stanford University. The working version of the third edition of this book is available at the link: https://web.stanford.edu/~jurafsky/slp3/
- Moreover, the particular source related to Markov models, which was used for the preparation of these materials is Appendix A of this book. https://web.stanford.edu/~jurafsky/slp3/A.pdf

Markov chain

Markov chain - illustration

- Markov chain for weather (a) and for words (b), showing states and transitions.
- The initial distribution π is also required;
  - π = [0.1, 0.7, 0.2] for model (a) would mean a probability of 0.7 of starting in state 2 (cold), a probability of 0.1 of starting in state 1 (hot), etc.



Markov chain

- Andrey Andreyevich Markov (14. 6. 1856 – 20. 7. 1922.)
- Russian mathematician
- He is best known for his work on stochastic processes, and especially for his research on the model that will later be known precisely by his surname as the Markov chain.

Markov Assumption

- Let's imagine that this chain goes through a series of states $q_1\ q_2 \ldots\ q_i$
- What determines the probability that in the i^{th} step we are in some selected state a, e.g., $q_i=a$ (e.g., a=HOT for the case of the example of a weather chain)?

$$\textbf{Markov Assumption:} \quad P(q_i = a|q_1 \ldots q_{i-1}) = P(q_i = a|q_{i-1}) \qquad \text{(A.1)}$$

- Markov assumes that the probability is determined exclusively by the first previous state q_{i-1} (from which it came), and not by the earlier states of the model: : $q_{i-2,}\ q_{i-3,\ ..}\ q_1$

Formal description of the Markov chain

- $Q = q_1\ q_2\ ...\ q_N$ is a set of N possible states of the model;
- $A = a_{11}\ a_{12}\ ...\ a_{N1},\ ...\ a_{NN}$ is the transition probability matrix A, where each member a_{ij} represents the probability of transition from state i to state j, where the sum of elements a_{ij} in each row (i.e., for columns j=1 to N) is equal to unity, because the system must either remain in the same state (i -> i), or must move to any new state (i -> j);
- $\pi = \pi_1, \pi_2, ..., \pi_N$ is the vector of the initial probability distribution by states. The value of π_i is the probability that the Markov chain will start in state i.

An example of calculating the probability of a series of observations for a given model

- For the earlier example of the weather change model which is shown by a state diagram with N=3 states: **Hot** (1), **Cold** (2) and **Warm** (3) and with the transition probabilities A and the initial distribution π shown in the same figure, it is necessary to determine the probabilities of observing these two sequences :

- Hot, Hot, Hot, Hot
- Cold, Hot, Cold, Hot
  - What does the difference in these probabilities tell you about the weather pattern encoded by the state diagram in the image?

What if model states are not directly observable?

- The Markov chain is useful when we need to calculate the probability of a sequence of observable events.
- However, in many cases the events we are interested in are hidden, … we do not observe them directly, but only indirectly.
- For example, word labels in the sentence (written text) are not visible to us. There are a total of 9 **parts of speech** in English: *nouns, pronouns, verbs, adverbs, adjectives, articles, prepositions, conjunctions,* and *interjections*.
- Words can also be labeled **by their function as a part of sentence** as: *subject, predicate, object, attribute, apposition, adverbial modifier*, and independent elements of the sentence: *direct address,* and *parenthesis.*

An example of hidden word tags in the text

- Analyze these sentences and add tags for each word according to the word type and service in the sentence:
  - *Our new school has a huge sports hall.*
  - *I'm going to the sea by bus tomorrow morning.*
  - *They had an argument about the match.*
  - *Old professor Matić always forgets his class register.*
- For help, … search “9 parts of speech in English” & ”parts of a sentence in English”

What if model states are not directly observable?

- When reading, word labels (tags) are not listed in the text, yet we do understand it successfully. Instead, we only see words and **have to infer the labels ourselves** from a string of words. Therefore, **we call labels hidden states** because they are not directly observable.

- A ***H**idden **M**arkov **M**odel* (**HMM**) allows us to simultaneously talk about observed events (like the sequence of words in the text we see) as well as hidden events (like part-of-speech tags) that we think are causal factors of the observable words in our probabilistic model.

Hidden Markov Model

Hidden Markov Model, HMM

- It is formally described by these elements:

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} \ldots a_{ij} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, \ldots, v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state i
$\pi = \pi_1, \pi_2, \ldots, \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

Hidden Markov Model

- The set of states of the model Q, the transition matrix A and the vector of the initial probability distribution of states π are the same as for the earlier description of the Markov chain.
- However, for the hidden Markov model we also have a **sequence of observed output symbols** $\boldsymbol{O} = o_1\, o_2 \ldots o_T$, which we can also call a sequence of observations of length T, where each observation is drawn from the **finite dictionary of output symbols**, $\boldsymbol{V} = v_1, v_2, \ldots, v_V$.
- An additional parameter of the HMM model is the **probability matrix of an individual observation** $\boldsymbol{B}=b_i(o_t)$ which describes the sequence of observation probabilities, which are also called **emission probabilities**. Each element $b_i(o_t)$ of this matrix expresses the probability of generating the output symbol o_t in the i^{th} state of the model.
- The number of model states N, the length of the observation sequence T, and the size of the output symbol dictionary V are generally distinct (integer) numbers .

Hidden Markov Model- Assumptions

- Similar to the Markov chain, the first-order hidden Markov model is based on these two assumptions:
  - 1) the probability of a certain state depends exclusively on the previous state:

$$\textbf{Markov Assumption:} \quad P(q_i|q_1 \ldots q_{i-1}) = P(q_i|q_{i-1}) \qquad \text{(A.4)}$$

  - 2) the probability of the output observation o_i depends only on the state q_i in which that observation was produced, and not on any other state or any other observation:

$$\textbf{Output Independence:} \quad P(o_i|q_1 \ldots q_i, \ldots, q_T, o_1, \ldots, o_i, \ldots, o_T) = P(o_i|q_i) \qquad \text{(A.5)}$$

Hidden Markov Model - Example - by Jason Eisner, 2002.

- Imagine you are a climatologist in the year 2799 studying the history of global warming. You can't find weather records for Baltimore, Maryland for the summer of 2020, but you do find Jason Eisner's personal diary listing how much ice cream Jason ate each day of that summer.
- We aim to use these recorded observations **to estimate** the temperature each day.
- We will simplify this task as much as possible by assuming that there are only two types of days: **C**(old) and **H**(ot).

Hidden Markov Model- Example- by Jason Eisner, 2002.

- **Task**: given a sequence of recorded observations *O* (an integer in the sequence represents the number of ice creams eaten on a particular day), find a "hidden" sequence *Q* of weather states (**H** or **C**) that caused Jason to eat one or more ice creams. He never ate more than three, but he always ate at least one ice cream.
- Figure A.2 shows an example of an HMM model for the described "ice cream" task. The two hidden states (**H** and **C**) correspond to warm and cold weather, and the observations (drawn from the alphabet $O = \{1, 2, 3\}$) correspond to the number of ice creams Jason ate on a given day.

Hidden Markov Model- Example- by Jason Eisner, 2002.

- HMM model for the relationship between temperature and number of ice creams eaten

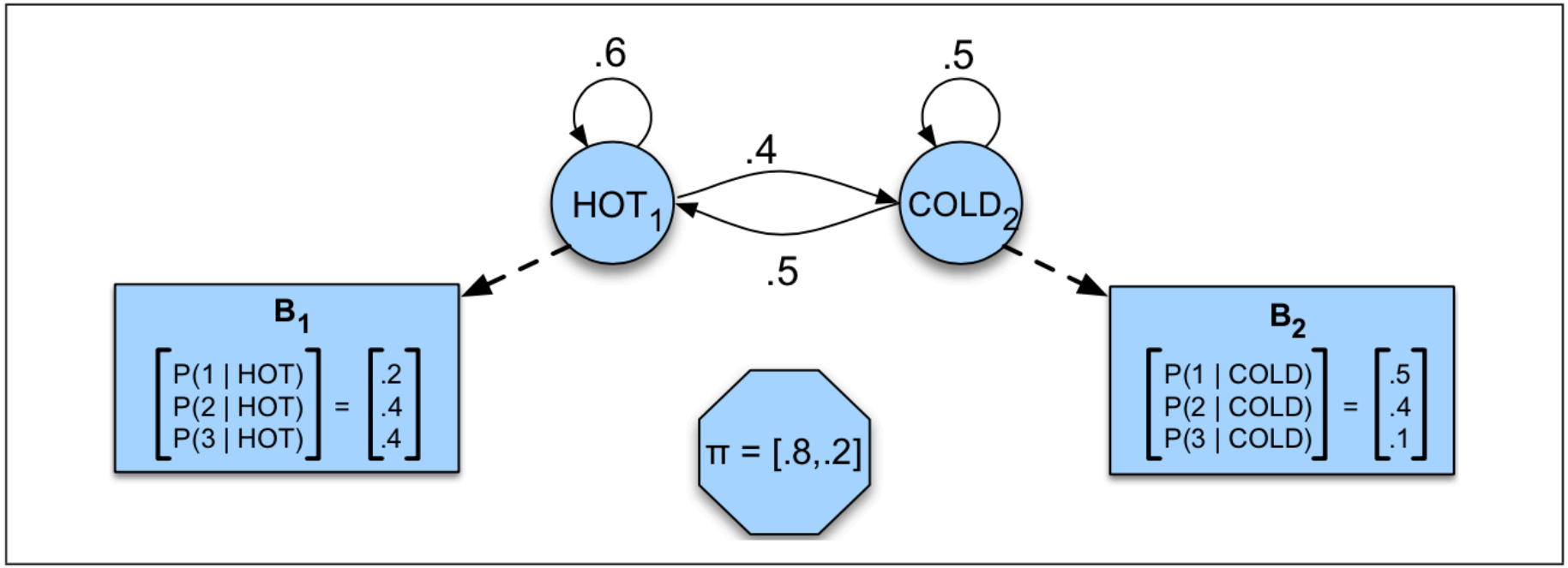


Figure A.2 A hidden Markov model for relating numbers of ice creams eaten by Jason (the observations) to the weather (H or C, the hidden variables).

What problems do we even want/need to solve for HMM models?

- Lawrence R. Rabiner, in a highly influential 1989 review article entitled "*A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*", (which was based on Jack Ferguson's tutorials from the 1960s), introduced the idea that hidden Markov models should be characterized by three fundamental problems:
- **Determination of likelihood**,
- **Decoding the hidden state**, and
- **Model training**.

- https://web.ece.ucsb.edu/Faculty/Rabiner/ece259/Reprints/tutorial%20on%20hmm%20and%20applications.pdf

What problems do we even want/need to solve for HMM models?

- Problem 1 (**Likelihood**): Given an HMM model $\lambda = (A, B)$ and an observation sequence O, determine the likelihood $P(O|\lambda)$, i.e., how likely is it that the observed sequence O was generated exactly as the output of the given model λ.
- Problem 2 (**Decoding**): Given an observation sequence O and an HMM model $\lambda = (A, B)$, discover the best hidden state sequence Q that generated that observation.
- Problem 3 (**Training**): Given an observation sequence O and the set possible states of an HMM, learn the HMM parameters A and B so that the model best describes the relationship between the sequence of observations and the set of hidden states.

How will we solve these problems?

- In order to solve problems 1 and 3 (likelihood of observation for the given model and training the model), we will use two algorithms that can significantly reduce the complexity of calculating the likelihood of observation, namely:
- "**Forward**" algorithm,
- "**Backward**" algorithm, and
- their combination, "**Forward-Backward**" algorithm, while
- to determine the best sequence of hidden states, we will use: **"Viterbi"** algorithm.

Likelihood calculation using the ”Forward" algorithm

Algorithm ”Forward" for calculation of the observation likelihood

- Our first problem is to calculate the likelihood of a particular sequence of observations for a given model.
- For example, for the HMM model in Figure A.2 that statistically describes the consumption of ice cream, let's determine what is the probability of observing the sequence "3 1 3"?
- Mathematically written...
- **Likelihood calculation**: Given an HMM model, $\lambda = (A, B)$, and a sequence of observations O, determine the likelihood $P(O|\lambda)$.

Algorithm ”Forward" for calculation of the observation likelihood

- For a Markov chain, where the external observations are identical to the hidden events, we could calculate the probability of "3 1 3" simply by following the states labeled 3 1 3 and multiplying the probabilities along the arcs.
- For a hidden Markov model, things are not so simple. We want to determine the probability of a sequence of observations of eaten ice creams like "3 1 3", but we don't know what the hidden sequence of states was!
- How to solve this problem...?
- We have to assume that we "nevertheless somehow know" the sequence of states, because then it is possible to find this probability ... and then repeat it for all possible sequences of states and add them up.

Algorithm ”Forward"- with an assumed and fixed sequence of hidden states

- Given such a one-to-one mapping and the Markov assumption expressed by equation A.4, for a certain sequence of hidden states $Q = q_0, q_1, q_2, \ldots, q_T$ and the corresponding sequence of output observations $O = o_1, o_2, \ldots, o_T$, the likelihood of the observed sequence is found as :

$$P(O|Q) = \prod_{i=1}^{T} P(o_i|q_i) \qquad \text{(A.6)}$$

- The calculation of the *forward probability* for our observation of ice creams "3 1 3" for one possible sequence of hidden states "hot hot cold" is shown in equation A.7. and Figure A.3, which graphically shows this calculation.

Algorithm ”Forward"- with an assumed and fixed sequence of hidden states

- For one assumed sequence of hidden states (Hot, Hot, Cold):

$$P(3\ 1\ 3|\text{hot hot cold}) = P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold}) \qquad \text{(A.7)}$$

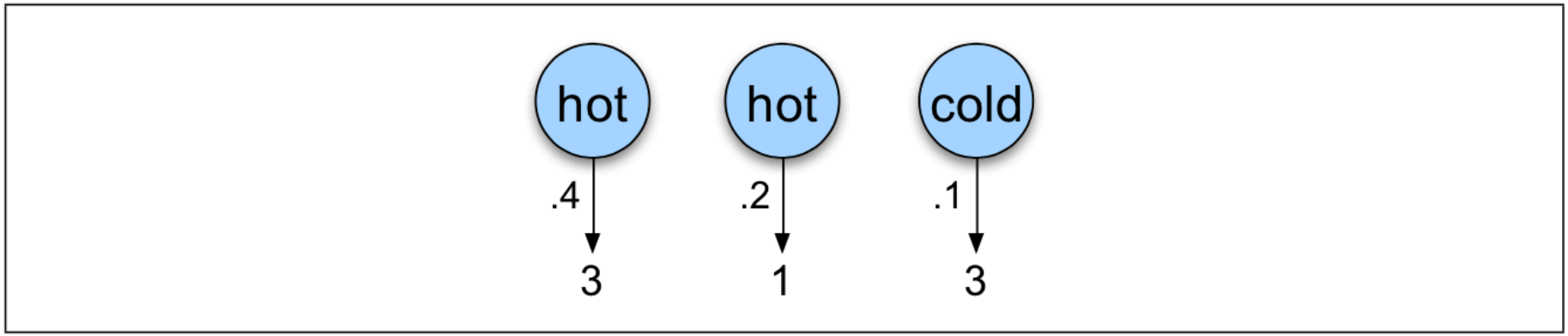


Figure A.3 The computation of the observation likelihood for the ice-cream events *3 1 3* given the hidden state sequence *hot hot cold*.

Algorithm "Forward"- combining all possible sequences of hidden states

- We don't actually know what the hidden sequence of states (weather changes) was, so we have to calculate the probability of ice cream events "3 1 3" by summing all possible time sequences of states, weighted by their probability.
- Let's first calculate the **joint probability** that we are in a certain time sequence of states *Q* and that we generate a certain output sequence *O* of ice cream events. In general, we can write this with this expression as:

$$P(O,Q) = P(O|Q) \times P(Q) = \prod_{i=1}^{T} P(o_i|q_i) \times \prod_{i=1}^{T} P(q_i|q_{i-1}) \qquad \text{(A.8)}$$

Algorithm ”Forward" - joint probability of a sequence of states *Q* and observations *O*

- For one assumed sequence of hidden states:

$$P(3\ 1\ 3, \text{hot hot cold}) = P(\text{hot}|\text{start}) \times P(\text{hot}|\text{hot}) \times P(\text{cold}|\text{hot}) \times P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold}) \quad \text{(A.9)}$$

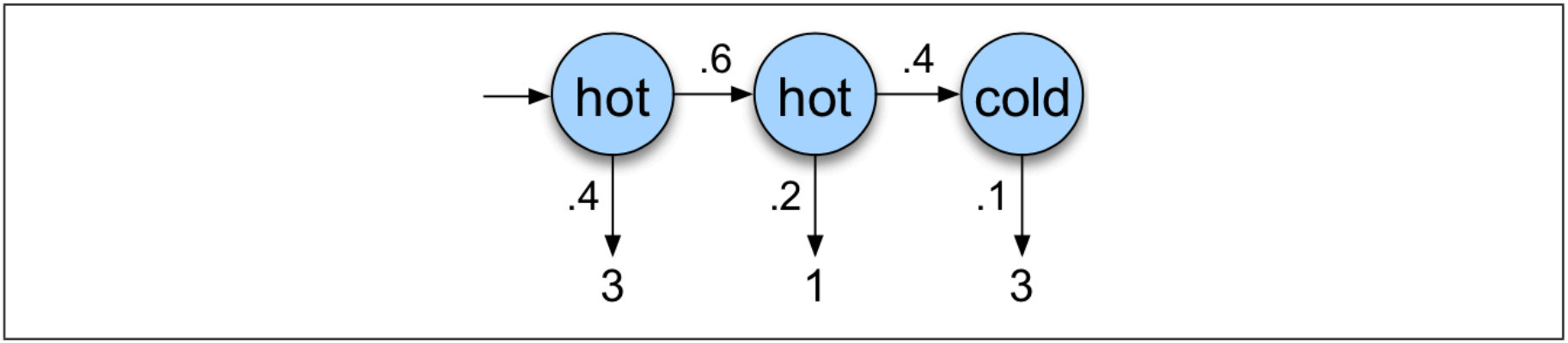


Figure A.4 The computation of the joint probability of the ice-cream events *3 1 3* and the hidden state sequence *hot hot cold*.

Algorithm "Forward"- combining all possible sequences of hidden states

- Now that we know how to calculate the joint probability of observation for a certain sequence of hidden states, we can calculate the total probability of observation by simply adding up all possible sequences of hidden states:

$$P(O) = \sum_{Q} P(O,Q) = \sum_{Q} P(O|Q)P(Q) \qquad \text{(A.10)}$$

- For this particular example, the number of possible combinations for the sequence of weather conditions is 8, i.e., it is necessary to add up the cases: "cold cold cold", "cold cold hot", "hot hot cold", and so on up to the eighth possible sequence of states "hot hot hot".

Algorithm "Forward"- combining all possible sequences of hidden states

- At first glance it sounds simple, … but is it even feasible?
- For an HMM model with *N* hidden states and an observation sequence of length *T*, there are N^T possible hidden state sequences.
- For real applications, where the values of *N* and *T* are large, $\boldsymbol{N^T}$ **is an exponentially large number**, so we cannot simply calculate the total observation probability by computing a separate observation likelihood of each hypothesized hidden sequence of states and then sum them.
- Is there a solution...?
- The answer lies precisely in the **efficient recursive implementation** the ”**Forward**” algorithm.

Algorithm "Forward"- efficient recursive implementation

- Instead of directly combining all sequences, the complexity of which is extremely exponential, we use an efficient O(N^2T) algorithm called the **Forward algorithm**.
- It belongs to the group of dynamic programming algorithms, i.e. it is an algorithm that uses a table to store intermediate values while building the total probability of a sequence of observations.
- The ”**Forward**" algorithm calculates the likelihood of an observation by summing the probabilities of all possible hidden state paths that could generate the observed sequence, but it does so effectively by implicitly folding each of the possible state paths into a single "**forward trellis**".

Algorithm "Forward" - using state trellis

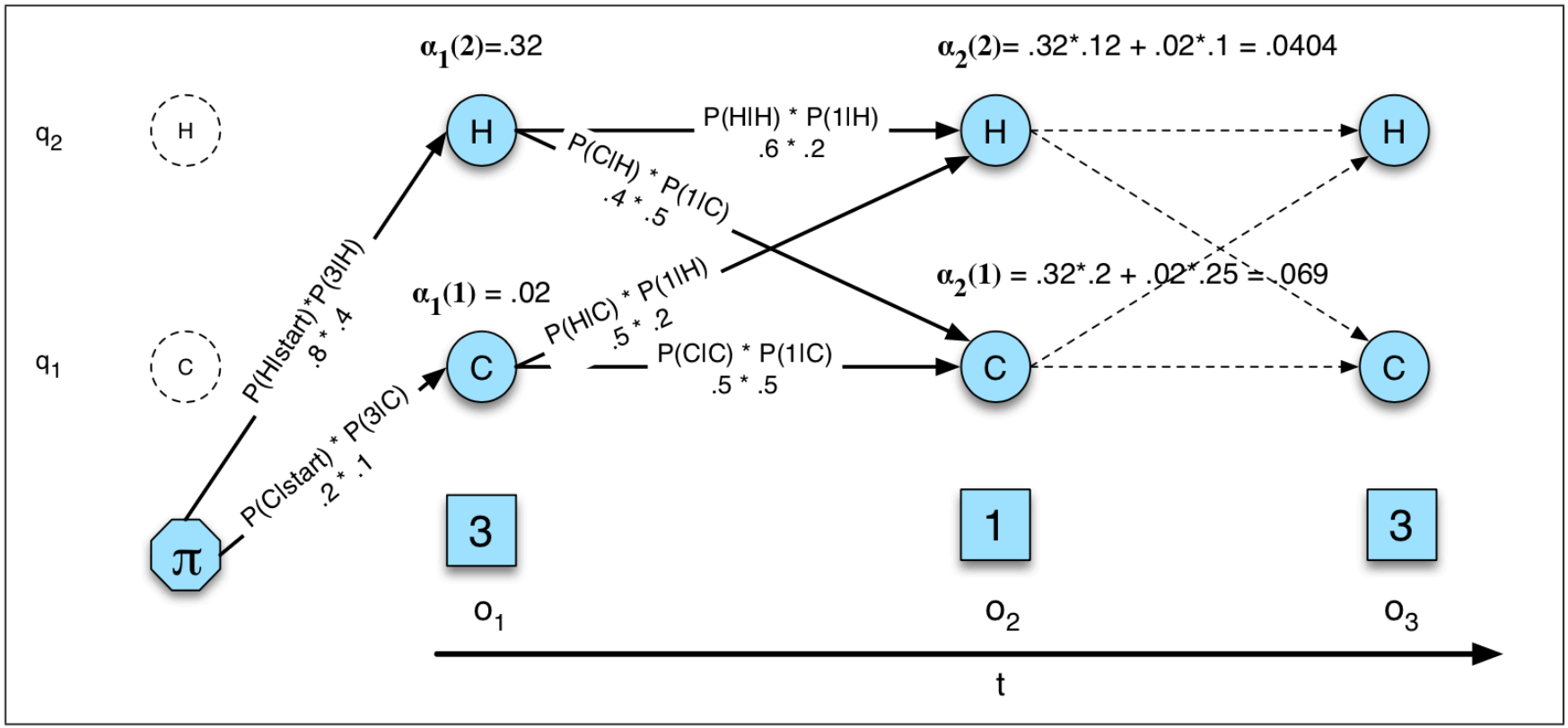


Figure A.5 The forward trellis for computing the total observation likelihood for the ice-cream events *3 1 3*. Hidden states are in circles, observations in squares. The figure shows the computation of $\alpha_t(j)$ for two states at two time steps. The computation in each cell follows Eq. A.12: $\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t)$. The resulting probability expressed in each cell is Eq. A.11: $\alpha_t(j) = P(o_1, o_2 \ldots o_t, q_t = j|\lambda)$.

Algorithm "Forward"- using state trellis

- Figure A.5 shows an example of such a ”**forward-trellis**" for computing the likelihood of the observed sequence "3 1 3" given all possible hidden sequences of states.
- For simplicity of presentation, the calculation is illustrated only for the first two observed output symbols o_1=3 and o_2=1, and it is extended analogously for the last observed symbol o_3=3.
- Each column in this grid corresponds to an individual observed symbol, while each row indicates one of the possible states in which the HMM model was at that moment (**H**, q_t=2 or **C**, q_t=1).
- Arrows point to all possible transitions from any given state in the current observation step *t* to any new (or the same) state in the next time step *t*+1. Horizontal arrows indicate retaining the same state, and diagonal arrows indicate its change.

Algorithm "Forward"- using state trellis

- How do we interpret the displayed values in the forward trellis?
- The HMM network starts from the initial state π (start) in which the system is before the appearance of the first output symbol and moves to state **H** (q_1=2) with probability 0.8 or to state **C** (q_1=1) with probability 0.2.
- At the first time moment t=1, the symbol o_1=3 is observed, so the probability that the model is in state **H** will be π_2* $b_2(o_1=3)$ = 0.8*0.4=0.32, and the probability that the state **C** is a new model state in t=1 should be equal to π_1* $b_1(o_1=3)$ = 0.2*0.1=0.02.
- These probabilities are marked in the figure as $\alpha_1(2)$=0.32 and $\alpha_1(1)$=0.02, where the subscript 1 denotes the first time moment t=1.
- In this example, state **H** is indexed as the second state and state **C** as the first state, so that numerical state index is indicated as an argument inside the parentheses of forward probability α.

Algorithm "Forward" - using state trellis

- Now we use these two values to calculate $\alpha_2(2)$ and $\alpha_2(1)$ based on the transition matrix *A* (which describes the probabilities of all possible forward paths in the trellis), and by knowing the probability of observing the second output symbol o_2=1 in each of the two possible states (**H** and **C**) which are defined in the matrix of output probabilities *B*.
- Specifically, into state **H** at *t*=2 we can end up via two possible forward paths:
  - straight path with probability $\alpha_1(2) * a_{22} * b_2(o_2=1) = 0.32 * 0.6 * 0.2$, or over
  - diagonal path from state **C** with probability $\alpha_1(1) * a_{12} * b_2(o_2=1) = 0.02 * 0.5 * 0.2$.
- By adding these two probabilities, we get $\alpha_2(2)=0.0404$.
- Analogously, by adding the probabilities for two possible paths that end in state **C** in step *t*=2, we get the forward probability $\alpha_2(1)=0.069$.
- It means that for model λ, the likelihood of observing the first two symbols "3 1" is equal to $\alpha_2(2) + \alpha_2(1)=0.1094$. We now repeat the same procedure for o_3=3, in order to obtain the likelihood of observing the entire sequence for the given model.

Algorithm ”Forward" - forward probability

- In general, each node $\alpha_t(j)$ in the trellis of this "Forward" algorithm represents the probability that the model will find itself in state *j* after examining the first *t* observations, and this with regard to the given parameters of the model λ, so we call it "**forward probability**".
- The value of each node $\alpha_t(j)$ is calculated by summing the probabilities of each path that could lead us to that node. Formally, each node expresses the following probability:

$$\alpha_t(j) = P(o_1, o_2 \ldots o_t, q_t = j|\lambda) \qquad \text{(A.11)}$$

- .. here $q_t=j$ means "that the t^{th} state in the series of model states is precisely the state *j*".

Algorithm ”Forward"- forward probability

- So, we calculate this probability $\alpha_t(j)$ by summing over the extensions of all the paths that lead to the current node. For a given state q_j at time t, we calculate the value $\alpha_t(j)$ as:

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t) \qquad \text{(A.12)}$$

- The three multiplication factors in each term of this sum that we use to calculate the forward probability at time t based on the extension of all previous paths are:
- $\alpha_{t-1}(i)$ … the **previous forward path probability** from the previous time step,
- a_{ij} ... the **transition probability** from previous state q_i to the current state q_j,
- $b_j(o_t)$ ... **likelihood of observing** the output symbol o_t in the current state j.

Algorithm ”Forward"- forward probability

- Consider the computation of $\alpha_2(2)$ in Figure A.5, i.e., the probability of being in state 2 in step *t*=2 and generating the partial observation “3 1”, by applying this **induction step**.
- We calculate this value by expanding the α-probabilities from time step 1, through two possible paths, where each expansion consists of the three mentioned factors:

$$\alpha_1(1) \times P(\mathbf{H}|\mathbf{C}) \times P(1|\mathbf{H}) + \alpha_1(2) \times P(\mathbf{H}|\mathbf{H}) \times P(1|\mathbf{H})$$

- Figure A.6 shows a general visualization of this induction step to calculate the value at one new trellis node, based on the forward probabilities from the previous step for an *N*-state HMM.

Algorithm ”Forward" – induction step *t*-1 -> *t*

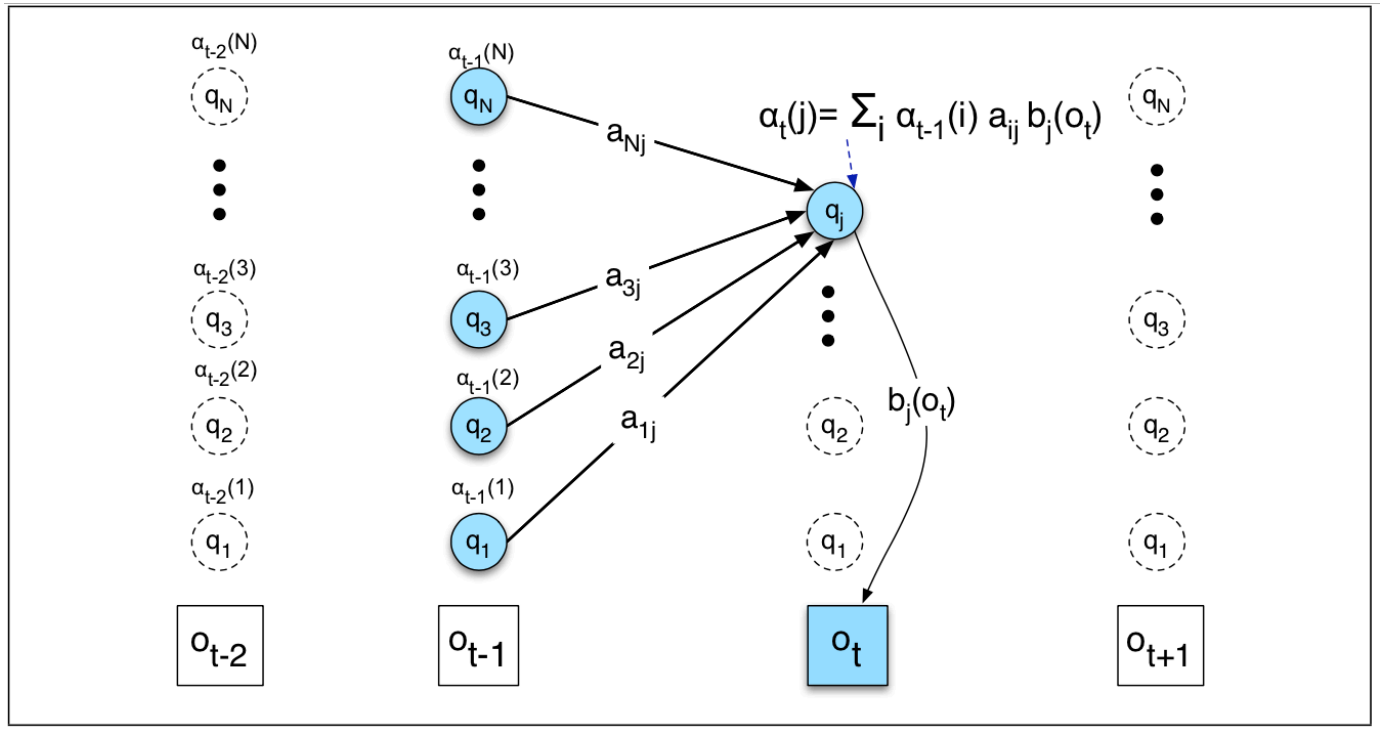


Figure A.6 Visualizing the computation of a single element $\alpha_t(i)$ in the trellis by summing all the previous values α_{t-1}, weighted by their transition probabilities a, and multiplying by the observation probability $b_i(o_t)$. For many applications of HMMs, many of the transition probabilities are 0, so not all previous states will contribute to the forward probability of the current state. Hidden states are in circles, observations in squares. Shaded nodes are included in the probability computation for $\alpha_t(i)$.

Algorithm ”Forward"– formal description

function FORWARD(*observations* of len T, *state-graph* of len N) **returns** *forward-prob*

create a probability matrix *forward[N,T]*
for each state s **from** 1 **to** N **do** ; initialization step
$forward[s,1] \leftarrow \pi_s * b_s(o_1)$
for each time step t **from** 2 **to** T **do** ; recursion step
for each state s **from** 1 **to** N **do**

$$forward[s,t] \leftarrow \sum_{s'=1}^{N} forward[s',t-1] * a_{s',s} * b_s(o_t)$$

$$forwardprob \leftarrow \sum_{s=1}^{N} forward[s,T]$$; termination step

return *forwardprob*

Figure A.7 The forward algorithm, where $forward[s,t]$ represents $\alpha_t(s)$.

Algorithm ”Forward" – main expressions

1. Initialization:

$$\alpha_1(j) \;=\; \pi_j b_j(o_1) \;\; 1 \le j \le N$$

2. Recursion:

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \le j \le N, 1 < t \le T$$

3. Termination:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

Algorithm ”Forward"– Termination

- The final step of this induction procedure is the summation of all forward probabilities in the last step $t=T$, over all hidden states of the model $i=1,2,...N$, which gives the required observation likelihood $P(O|\lambda)$ of the entire sequence $O = o_1, o_2, ..., o_T$, for the given HMM model $\lambda = (A, B, \pi)$ over all possible sequences of hidden states.

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

- A higher value of this likelihood indicates a good match between the model and the dynamic statistical properties of the observed sequence O, which are described by the parameters of this HMM model $\lambda = (A, B, \pi)$.

Decoding hidden states using Viterbi algorithm

The task of decoding the states of the model

- For any model, such as an HMM, that contains hidden variables, the task of determining the sequence of those variables that are the underlying source of some sequence of observations is called the **decoding task**.
- In the "ice cream" example, given the observation sequence of the number of eaten ice creams "3 1 3" and with the known HMM model λ, the task of the decoder is to find the **best hidden time sequence of states**.
- ... or more formally written:
- **Decoding**: Based on the given HMM model $\lambda = (A, B, \pi)$ for the given sequence of observed symbols $O = o_1, o_2, ..., o_T$, find the most probable sequence of hidden states $Q = q_1, q_2, q_3, ... q_T$.

The task of decoding the states of the model

- How to find that best sequence of hidden states?
- We could execute the Forward algorithm and calculate the likelihood of the observation sequence with respect to each possible assumed sequence of states, and in the end, we would choose exactly the sequence with the highest observation probability.
- Such an approach is only possible for models with a small number of states with short observation sequences, because the total number of possible state sequences is N^T, and for each we must determine the observation likelihood.
- Instead, the most commonly used HMM state decoding algorithm is the **Viterbi algorithm**, which we will call **Viterbi** for short, which does not have the exponential complexity of the direct approach.

The task of decoding– Viterbi algorithm

- Viterbi is also a type of dynamic programming that uses lattices for dynamic programming, and
- it strongly resembles another variant of dynamic programming, the shortest path algorithm (*minimum edit distance*).
- It has extremely widespread application in numerous engineering fields.
- Viterbi AJ (April **1967**). "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm". *IEEE Transactions on Information Theory*. **13** (2): 260–269. doi:10.1109/TIT.1967.1054010

- Andrea Giacomo Viterbi, born 9.3.1935, Bergamo, Italy

Viterbi algorithm – using state trellis

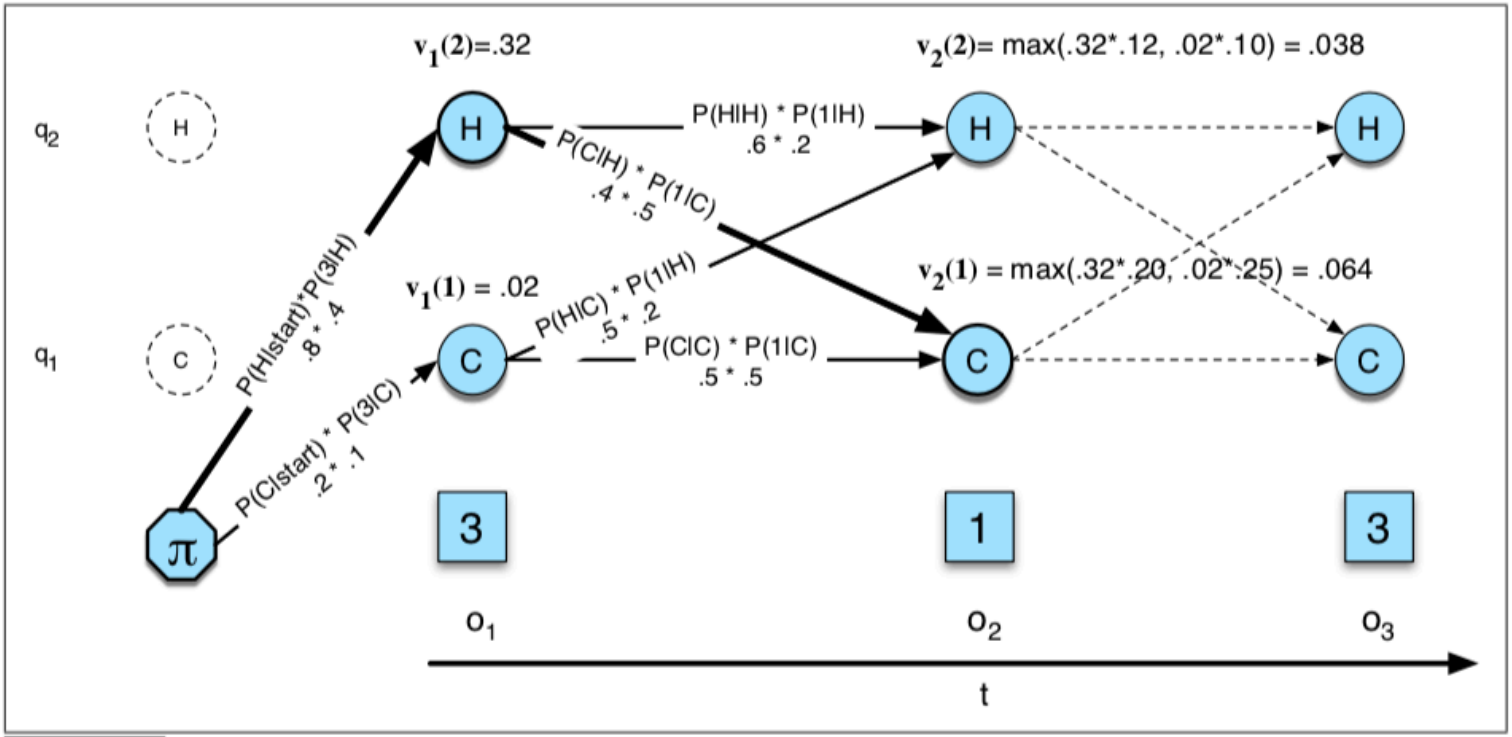


Figure A.8 The Viterbi trellis for computing the best path through the hidden state space for the ice-cream eating events *3 1 3*. Hidden states are in circles, observations in squares. White (unfilled) circles indicate illegal transitions. The figure shows the computation of $v_t(j)$ for two states at two time steps. The computation in each cell follows Eq. A.14: $v_t(j) = \max_{1 \le i \le N-1} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$. The resulting probability expressed in each cell is Eq. A.13: $v_t(j) = P(q_0, q_1, \ldots, q_{t-1}, o_1, o_2, \ldots, o_t, q_t = j|\lambda)$.

Viterbi algorithm – using state trellis

- The idea is to process the sequence of observations from left to right, filling the trellis, in which instead of forward probabilities we now write the **Viterbi probabilities** $v_t(j)$.
- Each node of this trellis, $v_t(j)$, represents the probability that the HMM ended up in state j after having previously observed the first t output symbols and having passed **through the most probable sequence of states** $q_1, \ldots, q_{t-1}$, given the model λ.
- The value of each node $v_t(j)$ is calculated by a recursive procedure, keeping only the most probable path that could lead us through the trellis to this new node.

Viterbi algorithm – Viterbi probability

- Formally, each node expresses this **Viterbi probability**:

$$v_t(j) = \max_{q_1,\cdots,q_{t-1}} P(q_1 \ldots q_{t-1}, o_1, o_2 \ldots o_t, q_t = j|\lambda) \qquad \text{(A.13)}$$

- … recognize from the above expression that we determine the most probable path by finding the maximum likelihood over all possible previous sequences of states up to time instance *t*-1.
- Given that in the last step we have already calculated the probabilities that we are in any possible model state at time step *t*-1, we calculate the Viterbi probability in step *t* taking only the **most probable extension of the paths** leading to the current node.
- This is the basis of the high efficiency of this recursive procedure!

Viterbi algorithm – Viterbi probability

- For the assumed state q_j at time t, the value of the Viterbi probability $v_t(j)$ at step t is calculated as:

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t) \qquad \text{(A.14)}$$

- The three factors in each product that are **maximized** in equation A.14 and used to extend the previous paths for the purpose of computing the Viterbi probability at time t are:
- $v_{t-1}(i)$ … **prior Viterbi probability** from the previous time step t-1 in all states of the model,
- a_{ij} …… the **probability of transition** from the previous state q_i into the current state q_j,
- $b_j(o_t)$ …. the **probability of observing the output symbol** o_t in the current state j.

Viterbi algorithm– formal description

function VITERBI(*observations* of len T,*state-graph* of len N) **returns** *best-path*, *path-prob*

create a path probability matrix *viterbi[N,T]*
for each state s **from** 1 **to** N **do** ; initialization step
    $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
    $backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$; termination step
$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T]$; termination step
bestpath $\leftarrow$ the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*

Viterbi algorithm– formal description

- The previous image shows the pseudocode for the Viterbi algorithm.
- Note that the Viterbi algorithm is identical to the Forward algorithm, except that it only takes **the maximum likelihood over the previous paths** in the trellis, while the Forward algorithm takes the sum of the probabilities over all paths.
- The Viterbi algorithm has one additional component that the ”Forward" algorithm does not have: it has the so-called **backpointers**.
- While the "Forward" algorithm only needs to calculate the overall probability of observing the given model over all paths, the Viterbi algorithm must calculate the probability along the best path, but at the same time the **most probable sequence of states** (the best path through the lattice).

Viterbi algorithm– backpointers

- The best sequence of states is found by tracing the path of hidden states that led to each state, as illustrated in Figure A.10, and then at the end when we have reached the last observed symbol o_T, hence ending up in the most probable final state q_T, we go back to the first state, backtracking the best path to the beginning, obtaining the so-called **Viterbi backtrace**.
- Therefore, for each node of the trellis, in addition to the Viterbi probability $v_t(j)$, we must also record the **backpointer** that connects that node with the corresponding state in the previous time step **from which we came with the highest probability**.
- Such a best identified return path is shown by bold arrows in the trellis in Figure A.10. by assuming T=2 (i.e., for abbreviated sequence).

Viterbi algorithm– backpointers & backtrace

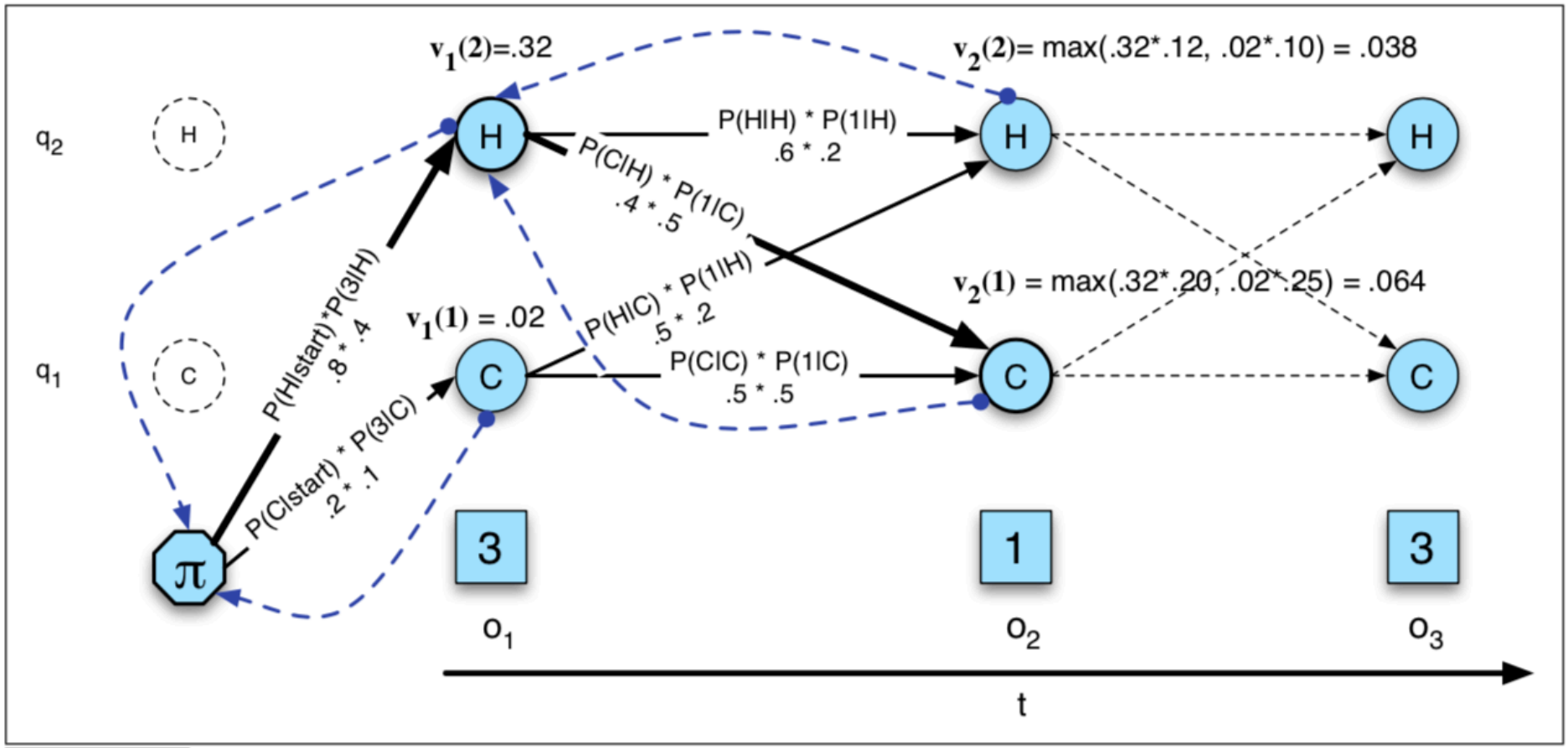


Figure A.10 The Viterbi backtrace. As we extend each path to a new state account for the next observation, we keep a backpointer (shown with broken lines) to the best path that led us to this state.

Viterbi algorithm – using state trellis

- How do we interpret the displayed values in the state trellis in A10?
- The HMM network starts from the initial state π (**start**) in which the system is before the appearance of the first output symbol, and moves to state **H** (q_1=2) with probability 0.8 or to state **C** (q_1=1) with probability 0.2.
- At the first time moment t=1, the symbol o_1=3 is observed, so the probability that the model was in state **H** will be π_2* $b_2(o_1=3)$ = 0.8*0.4=0.32, and the probability that it ended in state **C** should be equal to π_1* $b_1(o_1=3)$ = 0.2*0.1=0.02.
- These Viterbi probabilities are marked in the figure as $v_1(2)$=0.32 and $v_1(1)$=0.02, where the subscript 1 denotes the first time instant t=1.
- This part of the description is identical to the ”Forward" algorithm.

Viterbi algorithm – using state trellis

- Now we use these two values to calculate $v_2(2)$ and $v_2(1)$ in the second time step based on the transition matrix *A*, and by knowing the probability of observing the second output symbol o_2=1 in each of the two possible states (**H** and **C**) that are defined in matrix of output probabilities *B*.
- Specifically, we can end up in state **H** in step *t*=2 via two possible paths: a straight path with probability $v_1(2) * a_{22} * b_2(o_2=1)$ = 0.32 * 0.6 * 0.2, or via a diagonal path from state **C** with probability, $v_1(1) * a_{12} * b_2(o2=1)$ = 0.02 * 0.5 * 0.2.
- Instead of adding these two values, as we did with the ”Forward" algorithm, **we only take the larger** of these two probabilities and get $v_2(2)$=0.038, and thus choose only the horizontal branch.

Viterbi algorithm – using state trellis

- Analogously, by maximizing the probability for two possible paths that end in state **C** in step t=2, we get the probability $v_2(1)$=0.064 by using the diagonal path.
- It means that given the model λ, the probability of observing the first two symbols "3 1" is equal to either $v_2(2)$=0.038 or $v_2(1)$=0.064, depending on the state in which we ended up at the moment t=2.
- If this was at the same time the last symbol of the observed sequence O, we would choose only the path of the highest probability that ends in state **C**, because $v_2(1)>v_2(2)$, and which according to the backpointer was preceded by state **H** in step t=1, which gives the best path for the first two observed symbols (q_1=2 & q_2=1), that is marked by thick arrows on A10.
- We repeat the same procedure for the third observed symbol o_3=3.

Viterbi algorithm – partial state trace?

- The partial solution found for the best sequence of hidden states up to step *t*=2 based on the first two observed symbols **does not necessarily have to be the most probable sequence for the entire observed sequence!**
- It may happen that the node with the highest Viterbi probability in step *t*=3 recalls the state **H** as a previous state, so then the first two states according to the backpointers would be **H H**, and not **H L**, as invoked by only the first two observed symbols.
- Therefore, it is necessary to execute this recursive algorithm **until the last observed symbol** and only then return from the final state of the highest Viterbi probability to the first observation using the backpointers that we built in the recursive procedure.
- Using a partial optimal state path can be "**short-sighted**"!

Viterbi algorithm– main expressions

1. **Initialization:**

$$\begin{aligned} v_1(j) &= \pi_j b_j(o_1) \qquad 1 \le j \le N \\ bt_1(j) &= 0 \qquad\qquad\quad 1 \le j \le N \end{aligned}$$

2. **Recursion**

$$\begin{aligned} v_t(j) &= \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \\ bt_t(j) &= \operatorname*{argmax}_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \end{aligned}$$

3. **Termination:**

$$\begin{aligned} \text{The best score:} \quad P* &= \max_{i=1}^{N} v_T(i) \\ \text{The start of backtrace:} \quad q_T* &= \operatorname*{argmax}_{i=1}^{N} v_T(i) \end{aligned}$$

Model training using “Forward-Backward” algorithm

HMM model training

- Of all tasks presented so far related to Markov models, this one is the most complex.
- **Training**: Given a sequence of observations *O* and a set of possible HMM states, train the HMM parameters *A* and *B* using the chosen criterion.
- Input to the algorithm:
  - unlabeled sequence of observations *O*, and
  - the dictionary of potential hidden states *Q*, but not the sequence of states itself.
- ... for the example of the "ice cream" task, the input would be a sequence of observations, i.e., the number of eaten ice creams O = {1, 3, 2, 3, ...,} with a set of hidden states: **H** and **C**.

Baum – Welch algorithm („Forward-Backward”)

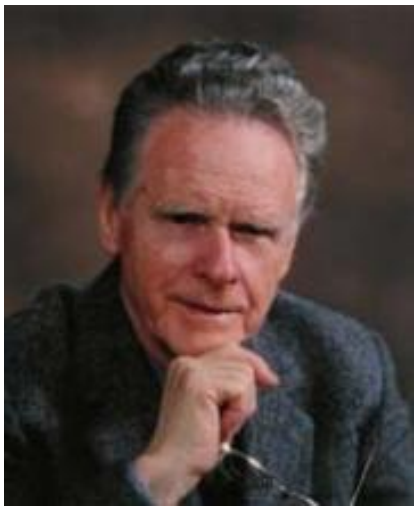
- The standard algorithm for training HMM models is the "Forward-Backward" algorithm, which is known as the Baum-Welch algorithm (Baum, 1972) after the surnames of the authors.
- Baum, L. E. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. In Shisha, O. (Ed.), *Inequalities III: Proceedings of the 3rd Symposium on Inequalities*, 1–8. Academic Press.

- Leonard Esau **Baum** (23.8.1931 –14.8.2017.) American mathematician

- Lloyd Richard **Welch** (28.9.1927. -) American information theorist

Baum – Welch algorithm („Forward-Backward")

- It represents a special case of *Expectation Maximization* (*EM*) algorithm (Dempster et al., 1977).
- It will enable the training of the transition probability matrix *A* and the observation probability matrix of the output symbols *B* of the HMM.
- EM is an iterative algorithm, which first calculates an initial probability estimate, then uses those estimates to calculate better estimates, and so on, iteratively improving the probabilities it learns.



- Arthur Pentland Dempster (born 1929), Professor Emeritus in the Harvard University Department of Statistics

HMM model training

- To explain how this algorithm works, let's start from a much simpler case of training a fully observable Markov model, i.e.,
  - suppose that we somehow know the temperature (state) and the number of ice creams eaten (observation) for each day for three series of observations.

```
3    3   2        1    1    2        1    2  3
hot  hot cold     cold cold cold    cold hot hot
```

- By knowing this "alignment" of states and observations, we can easily calculate the parameters of the HMM by maximum likelihood estimation from the set of input data for training, by simple counting.

HMM model training– vector π

- Let's first determine the vector π of initial state probabilities

3	3	2	1	1	2	1	2	3
hot	hot	cold	cold	cold	cold	cold	hot	hot

- We see that in the first step t=1 the **hot** state appears in one of the three cases, and **cold** in the remaining two of the three cases, so we can adjust the probability distribution of the initial state to the available input samples that we use for training as:

$$\pi_h = 1/3 \text{ i } \pi_c = 2/3$$

HMM model training– transition matrix *A*

- Let's calculate the matrix *A* from all possible state transitions for these three observation sequences by simply counting all the state changes from the first to the second step and from the second to the third step...

```
3    3   2        1    1    2        1    2  3
hot  hot cold     cold cold cold    cold hot hot
```

- The model went from the **hot** state to the **hot** state twice and from the **hot** state to the **cold** state only once, so we form the first two transition probabilities as:

$$p(\text{hot}|\text{hot}) = 2/3 \quad p(\text{cold}|\text{hot}) = 1/3$$

- Analogously, the model went from **cold** to **cold** twice and once from **cold** to **hot**, so we also find two other transition probabilities as:

$$p(\text{cold}|\text{cold}) = 2/3 \quad p(\text{hot}|\text{cold}) = 1/3$$

HMM model training– observation matrix *B*

- In a similar way, by counting, we also find the emission probabilities in the observation matrix *B* of the HMM model:

```
3    3   2          1    1    2          1    2  3
hot  hot cold       cold cold cold       cold hot hot
```

$p(1|\text{hot}) = 0/4 = 0$ $\quad$ $p(1|\text{cold}) = 3/5 = .6$

$p(2|\text{hot}) = 1/4 = .25$ $\quad$ $p(2|\text{cold} = 2/5 = .4$

$p(3|\text{hot}) = 3/4 = .75$ $\quad$ $p(3|\text{cold}) = 0$

HMM model training

- For a real HMM, we cannot calculate these probabilities by counting directly from the input observation sequences because unfortunately we do not know which state sequence the model took for a given input.

- For example, suppose we didn't know the temperature on day 2 only and you had to guess it somehow, but you (magically) still had the above probabilities and temperatures for all the other days.
- You could apply Bayesian arithmetic based on the known probabilities to get estimates of the likely temperature on that missing day, and then use those to get the expected numbers of high and low temperature occurrences for that 2nd day.

HMM model training– initialization of BW alg.

- However, the real problem is even more difficult because we do not know the statistics for any of the hidden states!!
- The Baum-Welch algorithm solves this by *iteratively estimating statistics*.
- We will start with **an arbitrary (random) estimate of the probabilities of transitions and observations**, and then use these estimated probabilities to derive progressively better probabilities.
- We will do this by calculating the likelihood of observing *O* with the "Forward" algorithm and then dividing that total probability mass between all possible state paths that contributed to that total forward probability of the model with the current parameters *A* and *B*.
- In order to understand the algorithm, we need to introduce another useful probability that is related to the forward probability, which is the **backward probability**.

Backward probability of HMM model –„Backward” algorithm

Backward probability of HMM

- Completely analogous to the definition of the forward probability, the **backward probability β** will be equal to the probability of observing the given sequence from the time instant *t*+1 to the end of the sequence (*t*=*T*), but with the assumption that we were in a state *i* at the time *t* (and with respect to the given model λ):

$$\beta_t(i) = P(o_{t+1}, o_{t+2} \ldots o_T | q_t = i, \lambda) \qquad \text{(A.15)}$$

- Similar to the forward probability, this backward probability can be calculated by an iterative process that starts from the last observed symbol at the time *t*=*T*, and then goes back step by step until the desired time *t* (which can also be *t*=1, if we want determine the overall likelihood of the entire observation *O* for the given model).

“Backward” algorithm – main expressions

1. **Initialization:**

$$\beta_T(i) \;=\; 1, \quad 1 \le i \le N$$

2. **Recursion**

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(o_{t+1})\, \beta_{t+1}(j), \quad 1 \le i \le N, 1 \le t < T$$

3. **Termination:**

$$P(O|\lambda) = \sum_{j=1}^{N} \pi_j\, b_j(o_1)\, \beta_1(j)$$

“Backward” algorithm– induction *t*+1-> *t*

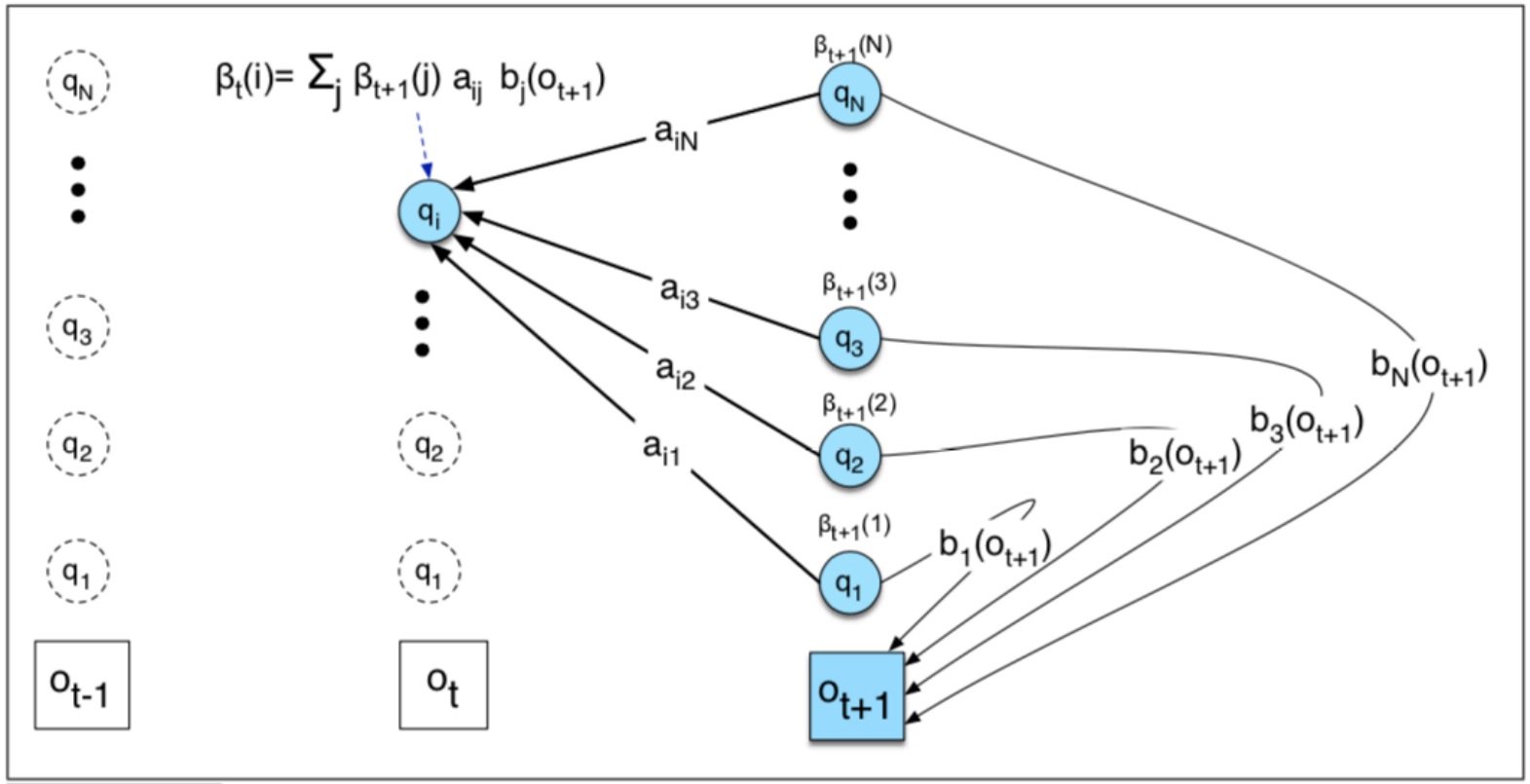


Figure A.11 The computation of $\beta_t(i)$ by summing all the successive values $\beta_{t+1}(j)$ weighted by their transition probabilities a_{ij} and their observation probabilities $b_j(o_{t+1})$. Start and end states not shown.

„Forward-Backward” algorithm for HMM model training

HMM model training

- We are now ready to explore how the forward and backward probabilities can help us to calculate the transition probability a_{ij} and the probability of observing the output symbol o_t in state i, $b_i(o_t)$ from a sequence of observations, even though the actual state path is unknown (hidden).
- Let's start by showing how to estimate the transition probability a_{ij} with a variant of simple maximum likelihood estimation:

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i} \qquad \text{(A.16)}$$

- ... thus as the quotient of the expected number of transitions from state *i* to state *j* and the total expected number of transitions from state *i* to any state.

HMM model training – transition matrix *A*

- How to calculate the numerator of this expression?
- Suppose we had some estimate of the probability that a given transition *i* → *j* occurred at a certain point in time *t* in the sequence of observations.
- If we knew this probability for each instant *t*, we could simply sum the transitions over all time instants to estimate the total number of *i* → *j* transitions.
- More formally, ... let's define the probability ξ_t /ksaɪ/ as the probability of being in state *i* at time *t*, and state *j* at time *t*+1, given the sequence of observed symbols *O* and given the current model λ:

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j | O, \lambda) \qquad \text{(A.17)}$$

HMM model training – transition matrix *A*

- To determine ξ_t we first compute an auxiliary probability that is similar to ξ_t but differs in that it includes the probability of observing *O*.

$$\text{not-quite-}\xi_t(i,j) = P(q_t = i, q_{t+1} = j, O|\lambda) \qquad \text{(A.18)}$$

- Notice the differently expressed conditional probability in expressions A.17 and A.18!
- Figure A.12 shows the various probabilities involved in the calculation of this auxiliary quantity, which we will call "not-quite-ξ_t".

HMM model training – determining „not-quite-ξ_t"

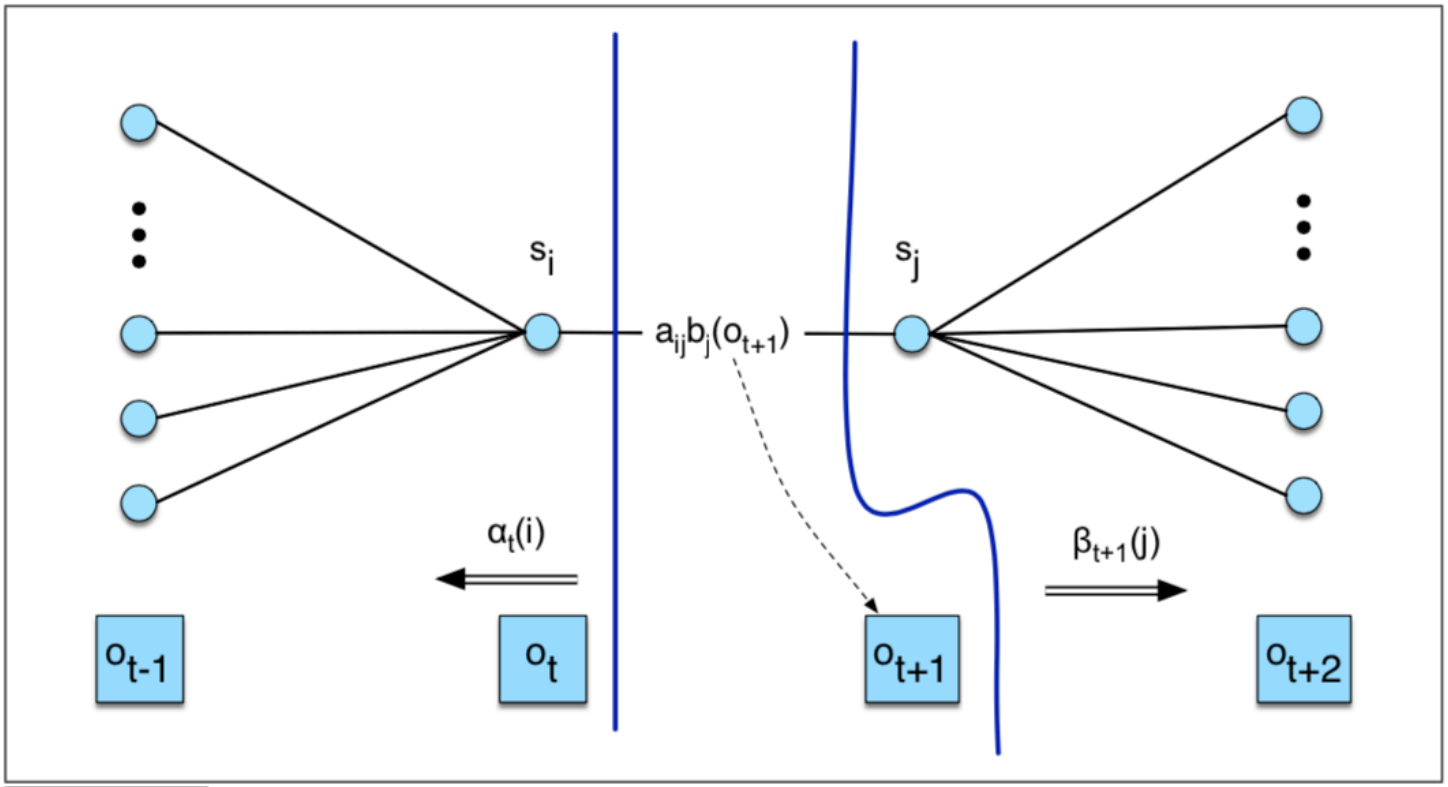


Figure A.12 Computation of the joint probability of being in state i at time t and state j at time $t+1$. The figure shows the various probabilities that need to be combined to produce $P(q_t = i, q_{t+1} = j, O|\lambda)$: the α and β probabilities, the transition probability a_{ij} and the observation probability $b_j(o_{t+1})$. After Rabiner (1989) which is ©1989 IEEE.

HMM model training – determining „not-quite-ξ_t”

- These are:
  - transition probability for the arc in question, a_{ij},
  - forward probability α before the arc (calculated with the ”Forward" algorithm),
  - backward probability β after the arc (calculated by the "Backward" algorithm),
  - the probability of the observed symbol immediately after the arc (at time t+1).
- These four factors are multiplied to produce the value of this auxiliary quantity, as follows:

$$\text{not-quite-}\xi_t(i,j) = \alpha_t(i)\, a_{ij} b_j(o_{t+1})\beta_{t+1}(j) \qquad \text{(A.19)}$$

HMM model training – determining ξ_t

- In order to ultimately determine ξ_t from this auxiliary quantity, we use a well-known rule from probability theory which reads:

$$P(X|Y,Z) = \frac{P(X,Y|Z)}{P(Y|Z)} \tag{A.20}$$

- So, in fact, you just need to divide this quantity by the total likelihood of the observed sequence *O* for the given model λ, P($O|\lambda$), which can be found based on the forward and backward probabilities as:

$$P(O|\lambda) = \sum_{j=1}^{N} \alpha_t(j)\beta_t(j) \tag{A.21}$$

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij} b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \tag{A.22}$$

HMM model training – finding new a_{ij}

- The expected number of transitions from state *i* to state *j* is then simply the sum of ξ_t for each *t*.
- For our estimate of the transition probability a_{ij} in Equation A.16, we only need one additional thing: that is the total expected number of transitions from state *i* in the denominator of that expression.
- It can be obtained by summing all transitions from state *i* to any state of the model, which gives the final formula for estimating new a_{ij}:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i,k)} \qquad \text{(A.23)}$$

- Recognize that to calculate the new values of $\hat{a}_{ij}$ we used the current values of the same parameters a_{ij} (we're pulling ourselves up by the bootstraps)!

HMM model training – finding new $b_j(v_k)$

- Let's remember that in addition to the elements of the transition matrix *A*, we also need the emission probabilities of the model in *B*.
- It is the probability of observing a given symbol v_k from the dictionary of output symbols *V*, given state *j*: $b_j(v_k)$. We will find it by trying to calculate this quotient:

$$\hat{b}_j(v_k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\text{expected number of times in state } j} \qquad \text{(A.24)}$$

- … therefore as a ratio of the expected number of observations of the symbol v_k in state *j* in relation to the total expected number of moments in which the HMM model was in state *j*. For this we will need to know the probability of being in state *j* at time *t*, and we will denote this auxiliary probability with $\gamma_t(j)$.

HMM model training – finding $\gamma_t(j)$

- So, we define the **auxiliary probability** $\gamma_t(j)$ as:

$$\gamma_t(j) = P(q_t = j|O, \lambda) \tag{A.25}$$

- Analogous to the calculation of ξ_t, here we will also define the conditional probability differently, which we will compensate for by dividing this modified conditional probability by the likelihood of the entire sequence of observed symbols for the given model, according to the expression:

$$\gamma_t(j) = \frac{P(q_t = j, O|\lambda)}{P(O|\lambda)} \tag{A.26}$$

HMM model training – finding $\gamma_t(j)$

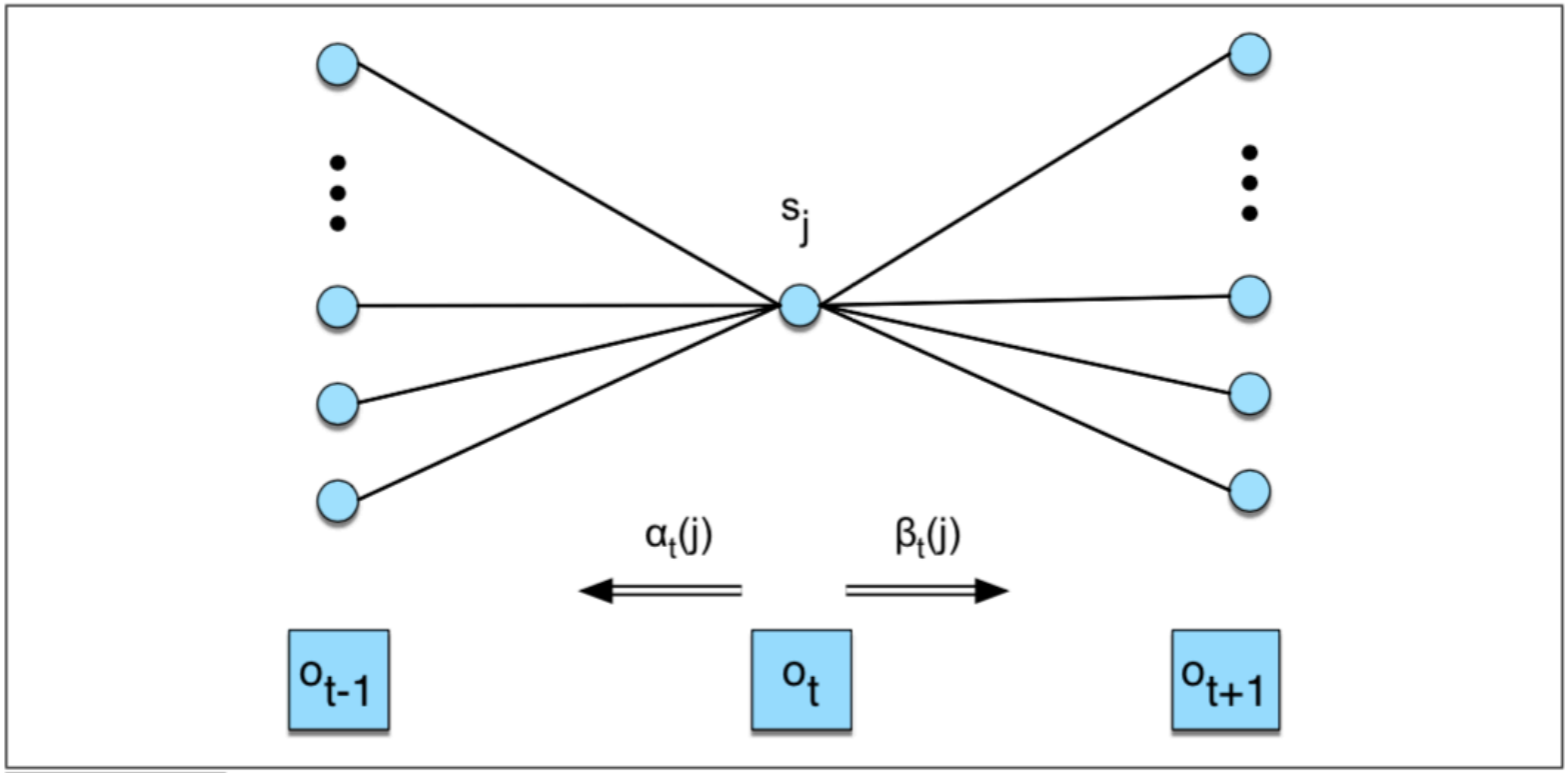


Figure A.13 The computation of $\gamma_t(j)$, the probability of being in state j at time t. Note that γ is really a degenerate case of ξ and hence this figure is like a version of Fig. A.12 with state i collapsed with state j. After Rabiner (1989) which is ©1989 IEEE.

HMM model training – finding $\gamma_t(j)$

- The calculation of this auxiliary probability $\gamma_t(j)$ is illustrated in Figure A.13, where the numerator of expression A.26 is equal to the product of the forward probability up to time *t* and the backward probability from the end of the sequence to the same time instance *t*.

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)} \qquad \text{(A.27)}$$

- With this, we have prepared all the necessary quantities for calculating emission probabilities in matrix *B*.
- For the numerator of expression A.24, we accumulate $\gamma_t(j)$ for all time steps *t* in which the observed symbol o_t is equal to the symbol v_k whose emission probability we are looking for. For the denominator, we sum $\gamma_t(j)$ in all time steps *t*, regardless of the observed symbol.

HMM model training – finding new $b_j(v_k)$

- The final result is the proportion: ... how many times when we were in state *j* we observed exactly the v_k symbol (the notation s.t. after the sum in the numerator reads "Sum $\gamma_t(j)$ over all *t* for which the observed symbol at the moment *t* was the v_k symbol"):

$$\hat{b}_j(v_k) = \frac{\sum_{t=1\ s.t.\ O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \qquad \text{(A.28)}$$

HMM model training– iterative re-estimation

- Equation A.23 (for new transition probabilities) and equation A.28 (for new emission probabilities) allow re-estimation of transition probabilities *A* and emission probabilities *B* from a given sequence (one or more) of observed symbols O, ...
- .... but assuming that we already have a previous estimate of *A* and *B*, because for the calculation of these new estimates we had to use auxiliary probabilities ξ_t(i,j) and $\gamma_t(j)$ that were determined based on the existing model $\lambda = (A, B)$.
- Such re-estimations form *the very core* of this iterative algorithm "Forward-Backward", which is why we say that we ”pulling ourselves by the bootstraps".

"Forward-Backward" algorithm – formal description

function FORWARD-BACKWARD(*observations* of len T, *output vocabulary* V, *hidden state set* Q) **returns** *HMM=(A,B)*

initialize A and B

iterate until convergence

E-step

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)} \quad \forall\, t \text{ and } j$$

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij} b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(q_F)} \quad \forall\, t,\ i, \text{ and } j$$

M-step

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

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t.\ O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

return A, B

Figure A.14 The forward-backward algorithm.

"Forward-Backward" algorithm

- The Forward-Backward algorithm (shown in Figure A.14) starts with some initial estimation of the HMM parameters $\lambda = (A, B)$.
- Then, we iteratively repeat the two steps. As with other applications of the expectation maximization algorithm (EM), the described Forward-Backward algorithm, which also belongs to the same family of algorithms, has two key steps:
- **the expectation step** or the so-called *E-step* and **maximization step** or so-called *M-step*.
- In the *E-step*, we calculate the expected number of state occupations $\gamma_t(j)$ and the expected number of state transitions ξ_t from the "fixed" probabilities *A* and *B* determined in the previous step, while in the *M-step* we use $\gamma_t(j)$ and ξ_t to recalculate the new probabilities *A* and *B*, which we then "fix" and use in the next *E-step*.

"Forward-Backward" algorithm

- Although in principle the "Forward-Backward" algorithm can be used to perform completely unsupervised training of parameters *A* and *B*, in practice the initial conditions are very important.
- For this reason, additional information is often passed to the algorithm.
- For example, for HMM-based automatic speech recognition, the structure of the HMM is often set manually, and only the emission probabilities *B* and transition probabilities *A* (which must not be equal to zero) are trained from a set of observation sequences *O*.

Presentation abstract

Hidden Markov Models (HMM)

- In this presentation, we introduced **hidden Markov models** for the **probabilistic classification of observations** of arbitrary lengths, but exclusively of a discrete dictionary of output symbols, the so-called discrete HMMs.
- Hidden Markov Models (HMMs) are a way of formally associating a series of **observed symbols** with a series of **hidden classes** or **hidden states** that explain these observations.
- The process of discovering a sequence of hidden states, given a sequence of observations, is known as **decoding or inference**. The **Viterbi** algorithm, which is commonly used for this purpose, is presented.
- The parameters of the HMM are the transition probability matrix *A* and the emission probability matrix *B*. Both can be trained with the **Baum-Welch** or the **Forward-Backward** algorithm.

Bibliographical and Historical Notes

Hidden Markov Models – Historical notes

- Markov chains were first used by Andrey Andreyevich Markov (1913), *Андре́й Андре́евич Ма́рков*, (translation Markov 2006), to predict whether an upcoming letter in Pushkin's *Eugene Onegin* would be a vowel or a consonant.
- Hidden Markov models were developed by Baum and colleagues at the Princeton Institute for Defense Analyzes (Baum and Petrie 1966, Baum and Eagon 1967).

- The Viterbi algorithm was first applied to speech and language processing in the context of speech recognition by Vintsyuk (1968), but the algorithm has what Kruskal (1983) calls "*a remarkable history of multiple independent discovery and publication*".

Hidden Markov Models – Historical notes

- Kruskal et al provide at least the following independently discovered versions of the same Viterbi algorithm published in four separate fields:

Citation	Field
Viterbi (1967)	information theory
Vintsyuk (1968)	speech processing
Needleman and Wunsch (1970)	molecular biology
Sakoe and Chiba (1971)	speech processing
Sankoff (1972)	molecular biology
Reichert et al. (1973)	molecular biology
Wagner and Fischer (1974)	computer science

- … which shows that they all came up with the same idea at almost the same moment in history.

Hidden Markov Models – Historical notes

- The use of the term *Viterbi* is now standard for the application of *dynamic programming* to any kind of probabilistic maximization problem in speech and language processing.
- For non-probabilistic problems (such as for *minimum edit distance*), the plain term *dynamic programming* is often used.
- Forney, Jr. (1973) wrote an early survey paper that explores the origin of the Viterbi algorithm in the context of information and communications theory.

Hidden Markov Models – Historical notes

- Presentation of the idea that hidden Markov models should be characterized by three fundamental problems was modeled after an influential tutorial by Rabiner (1989), which was itself based on tutorials by Jack Ferguson of IDA in the 1960s.
- Jelinek (1997) and Rabiner and Juang (1993) give very complete descriptions of the “Forward-Backward” algorithm as applied to the speech recognition problem.
- Jelinek (1997) also shows the relationship between “Forward-Backward” and general EM algorithms.

Literature

- **Baum, L. E.** (1972). An inequality and associated maximization technique in statistical estimation for probabilis- tic functions of Markov processes. In Shisha, O. (Ed.), *Inequalities III: Proceedings of the 3rd Symposium on In- equalities*, 1–8. Academic Press.
- **Baum, L. E. and Eagon, J. A.** (1967). An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, *73*(3), 360– 363.
- **Baum, L. E. and Petrie, T.** (1966). Statistical inference for probabilistic functions of finite-state Markov chains. *Annals of Mathematical Statistics*, *37*(6), 1554–1563.
- **Dempster, A. P., Laird, N. M., and Rubin, D. B.** (1977). Maximum likelihood from incomplete data via the *EM* algorithm. *Journal of the Royal Statistical Society*, *39*(1), 1–21.
- **Eisner, J.** (2002). An interactive spreadsheet for teaching the forward-backward algorithm. In *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teach- ing NLP and CL*, 10–18.
- **Forney, Jr., G. D.** (1973). The Viterbi algorithm. *Proceedings of the IEEE*, *61*(3), 268–278.

- **Jelinek, F.** (1997). *Statistical Methods for Speech Recognition*. MIT Press.
- **Kruskal, J. B.** (1983). An overview of sequence comparison. In Sankoff, D. and Kruskal, J. B. (Eds.), *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, 1–44. Addison-Wesley.
- **Markov, A. A.** (1913). Essai d'une recherche statistique sur le texte du roman "Eugene Onegin" illustrant la liaison des epreuve en chain. *Izvistia Imperatorskoi Akademii Nauk (Bulletin de l'Acade´mie Impe´riale des Sciences de St.-Pe´tersbourg)*, *7*, 153–162.
- **Markov, A. A.** (2006). Classical text in translation: A. A. Markov, an example of statistical investigation of the text Eugene Onegin concerning the connection of samples in chains. *Science in Context*, *19*(4), 591–600. Translated by David Link.
- **Needleman, S. B. and Wunsch, C. D.** (1970). A general method applicable to the search for similarities in the amino-acid sequence of two proteins. *Journal of Molecular Biology*, *48*, 443–453.
- **Rabiner, L. R.** (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257–286

- **Rabiner, L. R. and Juang, B. H.** (1993). *Fundamentals of Speech Recognition*. Prentice Hall.
- **Reichert, T. A., Cohen, D. N., and Wong, A. K. C.** (1973). An application of information theory to genetic mutations and the matching of polypeptide sequences. *Journal of Theoretical Biology*, *42*, 245–261.
- **Sakoe, H. and Chiba, S.** (1971). A dynamic programming approach to continuous speech recognition. In *Proceedings of the Seventh International Congress on Acoustics*, Vol. 3, 65–69. Akadeḿ miai Kiadoó .
- **Sankoff, D.** (1972). Matching sequences under deletion- insertion constraints. *Proceedings of the National Academy of Sciences*, *69*, 4–6.
- **Vintsyuk, T. K.** (1968). Speech discrimination by dynamic programming. *Cybernetics*, *4*(1), 52–57. Russian Kibernetika 4(1):81-88. 1968.
- **Viterbi, A. J.** (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, *IT-13*(2), 260–269.
- **Wagner, R. A. and Fischer, M. J.** (1974). The string-to-string correction problem. *Journal of the Association for Computing Machinery*, *21*, 168–173.