Hidden Markov model

Jianxin Wu

LAMDA Group
National Key Lab for Novel Software Technology
Nanjing University, China
wujx2001@gmail.com

February 11, 2020

Contents

In this chapter, we will introduce the basic concepts in the hidden Markov model (HMM) and a few important HMM learning algorithms.

1 Sequential data and the Markov property

The hidden Markov model deals with sequential data, too. However, unlike in the dynamic time warping, we do not assume the sequential data can be aligned. In the HMM, the data are supposed to possess the Markov property. We will first have a closer look at various types of sequential data, then introduce the Markov property and HMM.

1.1 Various sequential data and models

Depending on their properties and our objectives, sequential data have been modeled in different ways. As has been introduced, if the sequential data can be aligned, dynamic time warping and other related algorithms (e.g., string matching methods) can be applied to handle them.

However, there are many types of sequential data that cannot be aligned—e.g., stock price data. A great many methods have been proposed to handle sequential data. We introduce a small subset of these methods very briefly in this section, but will avoid going into details.

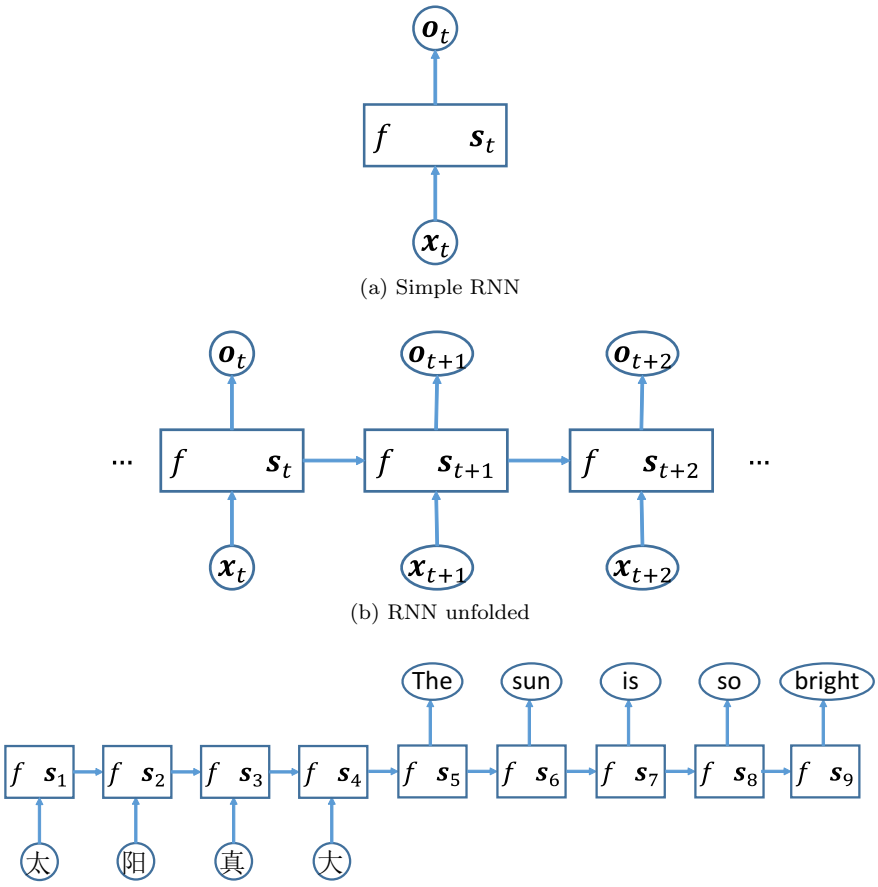
If dependencies in the sequence are not long-term, it is possible to use short-term history data to predict the next element in a sequence. For example, let $x_1, x_2, \ldots, x_t$ be an input sequence, and the objective is to predict x_{t+1} based on the existing sequence. We can extract the most recent history data in a short time period or time window, e.g.,

$$(x_{t-k+1}, x_{t-k+2}, \ldots, x_t)$$

is a time window with k readings in k time steps. Since we assume the dependencies among sequential data have short range, a reasonably large k should provide enough information to predict x_{t+1}. In addition, since the size of the time window is fixed (e.g., k), we can use all sorts of machine learning methods to predict x_{t+1} using the fixed length vector $(x_{t-k+1}, x_{t-k+2}, \ldots, x_t)$. If a linear relationship is considered, the moving average (MA) and autoregressive (AR) models are two example models popularly used in statistical analysis of time series (aka, sequential data), but different assumptions on noise and linear relationships are utilized in these two models. AR and MA can be combined to form the autoregressive-moving-average (ARMA) model.

Statistical models (e.g., ARMA) have been thoroughly analyzed, and many theoretical results are available. However, linear relationships are usually insufficient to model sequential data, and the assumptions of these statistical methods are often broken in the real world. We also want to deal with variable length input in many situations, rather than a fixed length time window. The recurrent neural network (RNN) can handle these complex situations.

Figure 1a shows a simple RNN architecture. At time step t, the input is a vector $\boldsymbol{x}_t$, and the simple RNN also maintains a *state* vector $\boldsymbol{s}_t$ (which is updated at every time step). The state vector is to be learned using the training data, and is taught by the training data to encode useful knowledge for fulfilling the learning objective included in the inputs till now: $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{t-1})$.



(a) Simple RNN

(b) RNN unfolded

(c) RNN translation

Figure 1: A simple RNN hidden unit, its unfolded version, and its application in language translation.

After observing $\boldsymbol{x}_t$, the hidden state vector should be updated at time $t+1$, given the current state vector $\boldsymbol{s}_t$ and input $\boldsymbol{x}_t$. A simple RNN updates the state using an affine transform and a nonlinear activation function, e.g.,

$$\boldsymbol{s}_{t+1} = f(W_x \boldsymbol{x}_t + W_s \boldsymbol{s}_t + \boldsymbol{b}_s)\,, \tag{1}$$

in which W_x, W_s, and $\boldsymbol{b}_s$ are parameters of the affine transform, and f is the nonlinear activation function (e.g., the logistic sigmoid function).

A simple RNN hidden unit may have an optional output vector $\boldsymbol{o}_t$. For example, if we want to choose a word from a dictionary, one possible way is to calculate $W_o \boldsymbol{s}_t + \boldsymbol{b}_o$ to obtain a score for each word, in which W_o is a parameter matrix, whose number of rows equals the number of words in the dictionary, and $\boldsymbol{b}_o$ is the bias parameter. Then, a softmax transformation will turn these scores into a probability distribution, and we can sample from this distribution to determine which word will be the output at time t.

The same hidden unit architecture (Figure 1a) is used for $t = 1, 2, \ldots$, in which the parameters remain unchanged but with different values of the input, state, and output vectors. We can treat an RNN with T time steps as a network with T layers: the hidden unit with the input, state, and output vectors for the t-th time step is the t-th layer in the network. One way to understand its data flow is to *unfold* these units along the time axis, as shown in Figure 1b. After an RNN is unfolded, it can be viewed as a *deep* network with many layers. Although these layers have different input, state, and output vectors, they share the *same* set of parameter values (e.g., W_x, W_s, $\boldsymbol{b}_s$, W_o, and $\boldsymbol{b}_o$). Methods for training deep neural networks (e.g., stochastic gradient descent) can be adopted to learn the parameters of an RNN in the unfolded network.

One application of RNN is machine translation, which is illustrated in Figure 1c. Given one sentence in the source language (e.g., Chinese), an RNN (with parameters already learned from training data) reads it in as a sequence of words. No output is given before the sentence is completely read and processed. After that, no input is needed for subsequent layers (time steps), and the translation to the target language (e.g., English) is given by the output nodes as another sequence of words.

Note that this figure is only for illustrative purposes. An RNN for real world translation (or other tasks) will be much more complex. For example, when long sequences are presented, a simple RNN will have difficulty in learning the parameters due to a problem called the *vanishing gradient* or *exploding gradient*. Also, a simple RNN is incapable of learning dependencies between inputs that are far apart. More complex hidden recurrent units have been proposed to deal with these difficulties—e.g., the Long-Short Term Memory (LSTM) hidden unit, the Gated Recurrent Unit (GRU), and the Minimal Gated Unit (MGU).

1.2 The Markov property

The HMM, unlike RNNs, only handles data with the Markov property, which is a property for some stochastic processes. A stochastic process (or random

process) is a sequence of ordered random variables, which can be treated as the evolution of states of a random system over time. Hence, unlike the RNN, which does not explicitly impose assumptions on the sequential data, the HMM interprets the sequential data from a probabilistic point of view. If $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_t$ is a sequence, then each $\boldsymbol{x}_i$ is considered as a random vector (or its instantiation). We use $\boldsymbol{x}_{1:t}$ as an abbreviation of the entire sequence.

Let $\boldsymbol{x}_t$ be the coordinates of an autonomous driving car at time t, the precise values of which are essential in autonomous driving. However, we do not have a precise way to directly observe this variable. Some measurements that can be directly observed at time t—e.g., GPS readings and the video frame taken by a video camera—are collectively denoted by $\boldsymbol{o}_t$, which are useful for us to estimate $\boldsymbol{x}_t$.

Uncertainty exists in the observations. The GPS readings are not robust, and its coordinates may jump frequently even in a short time period. Cameras can help reduce uncertainty in GPS readings, but scenes at different locations may look alike (e.g., imagine driving in the Great Sandy desert in Australia or the Taklamakan desert in Xinjiang, China). Hence, we need to use probabilistic methods to handle such uncertainties—i.e., our estimate of $\boldsymbol{x}_t$ is a distribution (instead of a single value).

When we move on from time $t-1$ to t, we can have an initial estimate of $\boldsymbol{x}_t$ based on 1) the distribution of $\boldsymbol{x}_{t-1}$; and, 2) the driving speed & direction of the car (i.e., based on the dynamic model). After taking into account the new observation $\boldsymbol{o}_t$, we can update our belief in $\boldsymbol{x}_t$ based on these evidences. The Kalman filter is one of the popular tools to estimate $\boldsymbol{x}_t$ in this setup.[1]

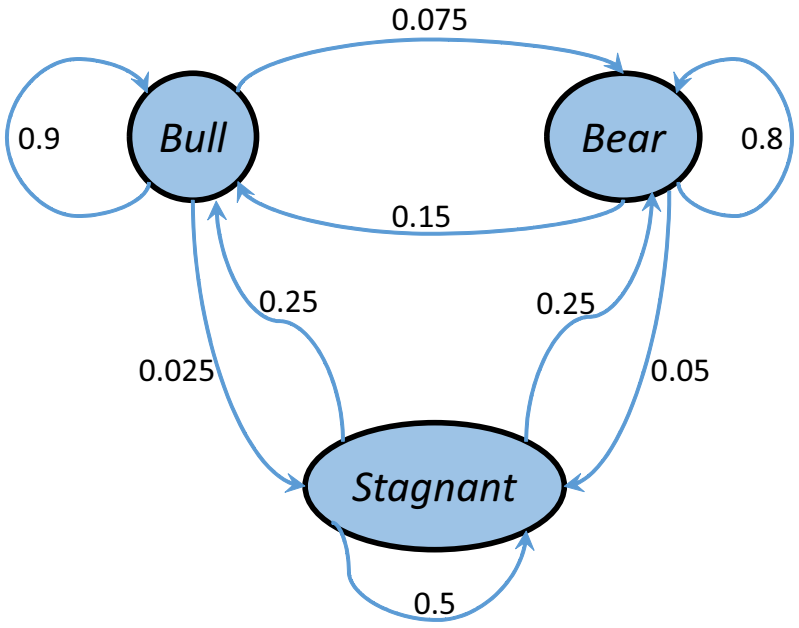
More technical details of the Kalman filter can be found in Chapter 13. In this chapter, we want to emphasize that in this stochastic process we have just described, the estimation of $\boldsymbol{x}_t$ only requires two things: $\boldsymbol{x}_{t-1}$ and $\boldsymbol{o}_t$—any previous hidden state ($\boldsymbol{x}_{1:t-2}$) or previous observation ($\boldsymbol{o}_{1:t-1}$) is not needed at all!

This is not surprising in our setup. Since $\boldsymbol{x}_{t-1}$ is a random vector, its estimation is an entire distribution. If the probability estimate for $\boldsymbol{x}_{t-1}$ has included all information in $\boldsymbol{x}_{1:t-2}$ and $\boldsymbol{o}_{1:t-1}$, we no longer need this so long as we already estimated the distribution of $\boldsymbol{x}_{t-1}$. This kind of "memoryless" property is called the *Markov property*, which states that the future evolution of a stochastic process only depends on the current state, but not on any preceding ones. It is obvious the Kalman filter makes the Markov assumption—i.e., the Markov property is assumed in Kalman filtering.

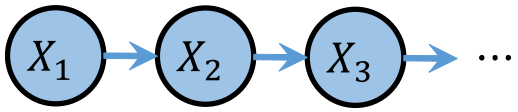
We have used the natural numbers $1, 2, \ldots$ to denote discrete time steps. A discrete-time stochastic process satisfying the Markov property is known as a Markov chain (also called the DTMC, discrete-time Markov chain).[2]

[1] Rudolf Emil Kalman, the primary co-inventor of this method, is an American electrical engineer and mathematician.

[2] Both the Markov property and Markov chain are named after Andrey Andreyevich Markov, a Russian mathematician. Markov's inequality, which is reviewed in Chapter 2, is also named after him.



(a) A DTMC example



(b) DTMC graphical model

Figure 2: A discrete-time Markov chain example and its graphical model illustration.

1.3 Discrete time Markov chain

Figure 2a specifies the evolution of an example DTMC. It models a hypothetical and overly simplified stock market. The random variable in consideration (X) has three possible states: bull market, bear market, or stagnant market, which are the three filled nodes. Arcs denote transitions between different states, and numbers around arcs are the transition probabilities. For example, the arc from the "Bull" node to itself means that a bull market has a 90% chance to remain in the next time step (but also has a 7.5% probability to transit to a bear market and 2.5% to a stagnant one).

If we ignore the transition probabilities and specific state symbols, the evolution process can be succinctly depicted as the graphical model in Figure 2b.[3]

[3] A graphical model (or probabilistic graphical model) uses nodes to represent random variables and arcs to denote probabilistic dependence among nodes. Arcs are replaced by lines in

Note that random variables are considered observable in a DTMC, which are drawn as filled circle nodes in graphical models.

Suppose X is discrete and has N possible values, denoted by symbols $S_1, S_2, \ldots, S_N$. To study how X evolves, we first need to observe X_1 at the first time step. Before we observe its value, there is a prior distribution to specify X. We denote the prior distribution by $p(X_1)$, and its p.m.f. is specified by a vector

$$\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_N)^T ,$$

with $\pi_i = \Pr(X_1 = S_i)$, $\pi_i \geq 0$ $(1 \leq i \leq N)$, and $\sum_{i=1}^{N} \pi_i = 1$.

Because the evolution of X follows a DTMC, X_t only depends on X_{t-1} but not on $X_{1:t-2}$. The transition from X_{t-1} to X_t is stochastic (i.e., probabilistic), fully specified by a *state transition probability matrix* A. A is an $N \times N$ matrix, with

$$A_{ij} = \Pr(X_t = S_j | X_{t-1} = S_i) . \tag{2}$$

Because X_t must be one of the N states, we know

$$\begin{array}{ll} \sum_{j=1}^{N} A_{ij} = 1 & \text{for any } 1 \leq i \leq N \\ A_{ij} \geq 0 & \text{for any } 1 \leq i, j \leq N . \end{array} \tag{3}$$

That is, every row of a transition matrix sums to 1. Hence, the numbers around all arcs emitting from any node in Figure 2a should sum to 1 (e.g., $0.5 + 0.25 + 0.25 = 1$ for the stagnant market node). We call a matrix satisfying these constraints a stochastic matrix. More precisely, it is a right stochastic matrix. A real matrix whose entries are non-negative and every column sums to one is called a left stochastic matrix.

One key benefit of the Markov property is that it greatly reduces the model complexity. If the Markov assumption is invalid, we need to estimate X_t using all previous states $X_{1:t-1}$. Hence, the parameters of $p(X_t | X_{1:t-1})$ require N^t numbers to completely specify it. This exponential increase of model complexity (aka, the curse of dimensionality) makes such a model intractable. Assuming A remains constant in different time steps, the two parameters $(\boldsymbol{\pi}, A)$ fully specify a DTMC because of the Markov assumption. Hence, we only need $N^2 - 1$ numbers to specify the parameters of a DTMC (cf. problem 1.) For notational simplicity, we use λ to denote the set of all parameters—i.e., $\lambda = (\boldsymbol{\pi}, A)$.

With a set of known parameters λ, we can *generate* a sequence for this DTMC (i.e., *simulating* how X evolves or to *sample* from it). To generate a sequence, we first sample from $\boldsymbol{\pi}$ and get an instantiation of X_1, say $X_1 = S_{q_1}$. Then, we sample from the row corresponding to X_1 (i.e., the q_1-th row) in A and get an instantiation of X_2. The sampling process can continue for any number of time steps, and can generate an arbitrary length sequence whose distribution follows the DTMC.

undirected graphical models. The Markov chain and HMM are both conveniently represented as graphical models. More complex graphical models and their inference algorithms are not discussed in this introductory book because they are advanced topics. Graphical models are important tools for learning and recognition.

1.4 Hidden Markov models

A sequence generated (sampled) from a DTMC only depends on probabilistic transitions, which may generate strange sequences. For example, a series of transitions "bull–bear–bull–bear" is possible to be sampled from the DTMC in Figure 2a, but this sequence is in general unlikely if the unit of a time step is a day instead of a year. In the autonomous driving car example, if observations $\boldsymbol{o}_t$ are not considered, we may find our car in Nanjing, China at time t (seconds), but in New York city, USA at time $t+1$ (seconds), which is impossible because teleportation has yet to be invented.

Observations are useful in dramatically reducing this kind of impossible or unreasonable transition between states. The GPS readings may be off by 10 meters, but will not locate a car at a location in New York if the real location is in Nanjing. A hidden Markov Model uses observations and states simultaneously: the state is what we want to estimate (e.g., the car's precise location) but is not directly observable (i.e., is hidden); the observations can be observed, and we can estimate the hidden states using these observations.

The hidden Markov model is illustrated in Figure 3 as a graphical model. We use Q_t to denote the state random variable at time t, and O_t the observation random variable. To fully specify an HMM, we need the following five items:

- N: the number of possible states. We use N symbols $S_1, S_2, \ldots, S_N$ to denote them.
- M: the number of possible observations if we assume the observation is discrete too. M is often determined by or derived from domain knowledge. We use M symbols $V_1, V_2, \ldots, V_M$ to denote possible observations. The observation O_t only depends on Q_t, but Q_t does not depend on O_t. Hence, HMM is a directed graphical model.

  The observation in an HMM can be, e.g., a normal distribution or even a GMM. In this book, we only focus on the simple scenario where the observation is a discrete random variable.
- $\boldsymbol{\pi}$: the prior (initial) state distribution. $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_N)$ and $\pi_i = \Pr(Q_1 = S_i)$.
- A: the state transition matrix. $A_{ij} = \Pr(Q_t = S_j | Q_{t-1} = S_i)$ is the probability of the next state being S_j if the current one is S_i, $1 \le i, j \le N$. Note that we assume A does not change when t changes.
- B: the observation probability matrix. Instead of denoting one probability as B_{jk}, we use $b_j(k) = \Pr(O_t = V_k | Q_t = S_j)$ to denote the probability of the observation being V_k when the state is S_j, $1 \le j \le N$, $1 \le k \le M$. And, we assume B does not change when t changes.

N and M determine the architecture or structure of a hidden Markov model. $\boldsymbol{\pi}$, A, and B are parameters of the HMM. For notational simplicity, we use

$$\lambda = (\boldsymbol{\pi}, A, B)$$

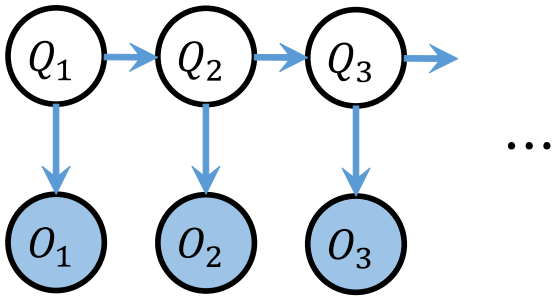


Figure 3: The hidden Markov model.

to denote all the parameters of an HMM too. Although we can manually design the state space for an HMM (or even manually set the values in A and B), it is convenient to specify the structure (N and M) and let an HMM learn its set of states and parameters from the training data.

The Markov property is translated in the HMM notation as

$$\Pr(Q_t|Q_{1:t-1}, O_{1:t-1}) = \Pr(Q_t|Q_{t-1})\,. \tag{4}$$

for any t. HMM is a generative model. After the structure and parameters are fixed, we can easily compute the joint probability of T hidden states $q_{1:T}$ (where q_t is the index of the state at time t, $1 \le q_t \le N$ for $1 \le t \le T$) and T observations $o_{1:T}$ (where o_t is index of the observation at time t, $1 \le o_t \le M$):

$$\Pr(Q_1 = S_{q_1}, O_1 = V_{o_1}, \cdots, Q_T = S_{q_T}, O_T = V_{o_T}) \tag{5}$$

$$= \pi_{q_1} b_{q_1}(o_1) A_{q_1 q_2} b_{q_2}(o_2) A_{q_2 q_3} b_{q_3}(o_3) \cdots A_{q_{T-1} q_T} b_{q_T}(o_T) \tag{6}$$

$$= \pi_{q_1} b_{q_1}(o_1) \prod_{t=2}^{T} A_{q_{t-1} q_t} b_{q_t}(o_t)\,. \tag{7}$$

In other words, because of the conditional independence (Q_t is conditionally independent of $Q_{1:t-2}$ and $O_{1:t-1}$ if Q_{t-1} is known), the joint probability is the product of a series of probabilities, whose computations never involve the joint of more than two random variables. For notational simplicity, we abbreviate the joint probability as $\Pr(q_{1:T}, o_{1:T})$.

Based on this joint probability mass function, we can sample from the HMM to generate an arbitrary length sequence of hidden states and observations. To generate a length T sequence, we use the following procedure:

1. $t \leftarrow 1$
2. Sample $Q_1 = S_{q_1}$ from $\boldsymbol{\pi}$
3. Sample the observation $O_1 = V_{o_1}$ using the p.m.f. which is in the q_1-th row of B

4. If the current time $t = T$, terminate; otherwise, sample the next state $Q_{t+1} = S_{q_{t+1}}$ using the p.m.f. that is in the q_t-th row of A

5. Sample the observation $O_{t+1} = o_{t+1}$ using the p.m.f. that is in the q_{t+1}-th row of B

6. $t \leftarrow t + 1$, and go to line 4.

2 Three basic problems in HMM learning

There are three basic learning problems in HMMs, whose definitions follow that in Lawrence R. Rabiner, *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, Proceedings of the IEEE, Vol 77, No. 2, 1989, pp. 257–286. In all these three problems, we assume the HMM structure (i.e., N and M) has been fixed.

The first basic problem is to evaluate the probability of an observation sequence $o_{1:T}$ when the model parameters are fixed—i.e., to calculate $\Pr(o_{1:T}|\lambda)$. This *evaluation* problem, as will be shown soon, is relatively simple to solve among the three basic problems. However, the evaluation problem has many important applications in HMMs.

For example, given a fixed parameter set λ and two sequences $o_{1:T}$ and $o'_{1:T}$, the evaluated sequence probability will tell us which sequence has a higher chance of being observed. Probably a more important usage of the evaluation problem is model selection. If we have an observation sequence $o_{1:T}$ and two sets of parameters λ_1 and λ_2 (which are, e.g., learned using different methods), it is reasonable to conjecture the model with a larger probability of observing this sequence is a better fit to the observations. Note that when we allow the parameters λ to change, the probability of observing $o_{1:T}$ given parameters λ is the *likelihood* of the parameters. Given a training sequence $o_{1:T}$, maximum likelihood estimation will find a parameter set that maximizes this likelihood (which is the third basic problem).

The second basic problem is the *decoding* problem. When we are given a fixed parameter set λ and an observation sequence $o_{1:T}$, what is the best hidden state sequence corresponding to these observations? As we have illustrated using the autonomous driving car example, it is usually the hidden states (rather than the observations) that are useful for an application. Hence, decoding an optimal hidden state sequence from the observation sequence is a very important problem in HMM learning.

However, it is not always easy to define what *best* or *optimal* means in the decoding problem. As will be shown, different optimality criteria have been proposed and different answers to the decoding problem will be given accordingly. In addition to finding the best sequence of hidden states, the decoding problem has other applications in HMM. For example, suppose we have learned an HMM model with a large N, but the decoding procedure has found that many symbols (in $S_1, S_2, \ldots, S_N$) have never appeared in the decoded optimal sequences. This fact suggests that N is too big for our problem at hand (i.e.,

many state symbols are wasted), and we may need to learn a new model with a smaller N.

The third and final basic problem in HMM is to *learn* the optimal parameters for an HMM. We have assumed the HMM model—i.e., the parameters $\lambda = (\boldsymbol{\pi}, A, B)$—have been given or assigned optimal values in the first two basic problems. In real world applications, however, we have to learn these parameters for ourselves.

As aforementioned, the basic idea is to find the parameters that have the largest likelihood. That is, given N, M, and the training sequence $o_{1:T}$, we want to find the parameter set λ that maximize the likelihood function $\Pr(o_{1:T}|\lambda)$. Note that in this maximum likelihood estimation, λ is not a random vector. Its appearance after the conditional probability symbol ("|") is only to indicate that the involved probability is computed using λ as the parameter values, and to be consistent with notations in the literature.

The training data for HMM learning, however, is different from those in methods we have seen (e.g., SVM). If T is large enough, one training sequence $o_{1:T}$ might be sufficient to learn a good HMM model. Of course, we can also use multiple training sequences to learn HMM parameters.

The key strategy for solving the first two basic problems is dynamic programming, which we have experienced in the dynamic time warping method in the previous chapter. The third basic HMM problem can be solved by using the Expectation-Maximization (EM) algorithm. We will introduce the details of the solutions to these problems in the rest of this chapter.

3 α, β, and the evaluation problem

Given N, M, λ, and $o_{1:T}$, the evaluation problem tries to compute $\Pr(o_{1:T}|\lambda)$. The law of total probability gives a way to compute it, as

$$\Pr(o_{1:T}|\lambda) = \sum_{q_{1:T}\in\Omega} \Pr(o_{1:T}, q_{1:T}|\lambda) \tag{8}$$

$$= \sum_{q_{1:T}\in\Omega} \Pr(o_{1:T}|q_{1:T}, \lambda) \Pr(q_{1:T}|\lambda)\,, \tag{9}$$

in which Ω is the space of all possible sequences of hidden states. Note that $\Pr(o_{1:T}|q_{1:T}, \lambda)$ means $\Pr(O_{1:T} = V_{o_{1:T}}|Q_{1:T} = S_{q_{1:T}}, \lambda)$. When the meaning is obvious from the symbols and contexts, we will omit the random variable names in equations.

It is obvious that

$$\Pr(o_{1:T}|q_{1:T}, \lambda) = \prod_{t=1}^{T} \Pr(o_t|q_t, \lambda) = \prod_{t=1}^{T} b_{q_t}(o_t)$$

and

$$\Pr(q_{1:T}|\lambda) = \pi_{q_1} \prod_{t=2}^{T} A_{q_{t-1}q_t}\,.$$

In other words, in order to use the law of total probability to compute $\Pr(o_{1:T}|\lambda)$, we need to generate $|\Omega|$ sequences of hidden states and observations. Because each state can take the values of N possible symbols and there are T time steps, $|\Omega| = N^T$, which means Equation 9 is not tractable.

The complexity of Equation 9 comes from variations in the states $Q_{1:T}$. Because the states are hidden, we have to enumerate all possibilities and compute the expectation, which leads to the exponentially increasing complexity. However, the Markov assumption says that given $Q_t = S_{q_t}$, Q_{t+1} is independent of $Q_{1:t-1}$ and $O_{1:t-1}$. In addition, O_t only depends on Q_t. In other words, we can divide the calculation $\Pr(o_{1:T}|\lambda)$ into two smaller problems: $\Pr(o_{1:T-1}|\lambda)$ and $\Pr(o_T|\lambda)$, and combine them by enumerating all possible states of Q_{T-1} and Q_T (whose complexity is $N \times N = N^2$). Similarly, the calculation of $\Pr(o_{1:T-1}|\lambda)$ can be further divided into $\Pr(o_{1:T-2}|\lambda)$ and $\Pr(o_{T-1}|\lambda)$—a typical dynamic programming formulation!

3.1 The forward variable and algorithm

The law of total probability tells us

$$\Pr(o_{1:T}|\lambda) = \sum_{i=1}^{N} \Pr(o_{1:T}, Q_T = S_i|\lambda) \tag{10}$$

$$= \sum_{i=1}^{N} \Pr(o_{1:T-1}, Q_T = S_i|\lambda) b_i(o_T)\,, \tag{11}$$

and then we need to compute $\Pr(o_{1:T-1}, Q_T = S_i|\lambda)$. Using the law of total probability again, we have

$$\Pr(o_{1:T-1}, Q_T = S_i|\lambda) \tag{12}$$

$$= \sum_{j=1}^{N} \Pr(o_{1:T-1}, Q_T = S_i, Q_{T-1} = S_j|\lambda) \tag{13}$$

$$= \sum_{j=1}^{N} \Pr(o_{1:T-1}, Q_{T-1} = S_j|\lambda) \Pr(Q_T = S_i|o_{1:T-1}, Q_{T-1} = S_j, \lambda) \tag{14}$$

$$= \sum_{j=1}^{N} \Pr(o_{1:T-1}, Q_{T-1} = S_j|\lambda) \Pr(Q_T = S_i|Q_{T-1} = S_j, \lambda) \tag{15}$$

$$= \sum_{j=1}^{N} \Pr(o_{1:T-1}, Q_{T-1} = S_j|\lambda) A_{ji}\,. \tag{16}$$

Note that in the above derivation, we have implicitly used *conditional independence* among variables without a proof. In the exercise problems for this chapter, we will describe the d-separation method that can precisely reveal such conditional independence.

The recursion from $T-1$ to T is not obvious yet. However, because

$$\Pr(o_{1:T-1}|\lambda) = \sum_{j=1}^{N} \Pr(o_{1:T-1}, Q_{T-1} = S_j|\lambda)\,, \tag{17}$$

if we can compute $\Pr(o_{1:T-1}, Q_{T-1} = S_j|\lambda)$ for *all* $1 \leq j \leq N$, we can evaluate $\Pr(o_{1:T-1}|\lambda)$, and similarly $\Pr(o_{1:T}|\lambda)$ for time T.

Hence, in a dynamic programming solution for the HMM evaluation problem, the quantity to be evaluated is $\Pr(o_{1:t}, Q_t = S_i|\lambda)$ for all $1 \leq t \leq T$ and $1 \leq i \leq N$. The *forward algorithm* (or forward procedure) defines this quantity as the forward variable $\alpha_t(i)$:

$$\alpha_t(i) = \Pr(o_{1:t}, Q_t = S_i|\lambda)\,, \tag{18}$$

which is the probability that at time t, the hidden state is S_i and the observation history till time t is $o_{1:t}$. The recursion between forward variables in two consecutive time steps is:

$$\alpha_{t+1}(i) = \left(\sum_{j=1}^{N} \alpha_t(j)A_{ji}\right) b_i(o_{t+1})\,, \tag{19}$$

whose proof is easy by using Equations 12 and 16.

It is obvious that when $t = 1$, we have

$$\alpha_1(i) = \pi_i b_i(o_1)\,.$$

Hence, we can start the recursion from $t = 1$, and move from left to right (i.e., t increases) until $t = T$ (hence this method is called the forward algorithm). The forward algorithm is described in Algorithm 1.

Algorithm 1 The forward algorithm

1: **Initialization**: $\alpha_1(i) = \pi_i b_i(o_1)$, for all $1 \leq i \leq N$

2: **Forward recursion**: For $t = 1, 2, \ldots, T-2, T-1$ and all $1 \leq i \leq N$,

$$\alpha_{t+1}(i) = \left(\sum_{j=1}^{N} \alpha_t(j)A_{ji}\right) b_i(o_{t+1})$$

3: **Output**:

$$P(o_{1:T}|\lambda) = \sum_{i=1}^{N} \alpha_T(i)\,. \tag{20}$$

It is obvious that the complexity of the forward algorithm is $\mathcal{O}(TN^2)$, which is efficient and much faster than N^T. Dynamic programming has once again proven itself to be an effective strategy in removing redundant computations.

3.2 The backward variable and algorithm

We can interpret the $\alpha_t(i)$ variable as the following: one person stands at time t and looks back; $\alpha_t(i)$ is the probability of this person observing state symbol S_i and the history observation sequence $o_{1:t}$. After obtaining $\alpha_t(i)$, if this person turns around and looks into the future, which information is still missing? We *have observed the state* $Q_t = S_i$ and observations $o_{1:t}$, and still have to observe $o_{t+1:T}$! Hence, the backward variable $\beta_t(i)$ is defined as

$$\beta_t(i) = \Pr(o_{t+1:T}|Q_t = S_i, \lambda)\,, \tag{21}$$

i.e., $\beta_t(i)$ is the probability of observing future output sequence $o_{t+1:T}$ if the hidden state is S_i at time t.

It is easy to derive the recursive relationship for $\beta_t(i)$, which is

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

It is obvious that the recursive updates move backward (i.e., use time $t+1$ probabilities to calculate those at time t). Hence, the backward algorithm (the backward procedure) must be initialized with $t = T$, and $\beta_T(i)$ is the probability of observing nothing after time T given the current state $Q_T = S_i$, which is 1. Finally, we have

$$\Pr(o_{1:T}|\lambda) = \sum_{i=1}^{N} \pi_i b_i(o_1) \beta_1(i)\,. \tag{23}$$

The proofs of the above two equations are left as exercises. Putting these facts together, we have come up with the backward algorithm in Algorithm 2.

Algorithm 2 The backward algorithm

1: **Initialization**: $\beta_T(i) = 1$, for all $1 \le i \le N$

2: **Backward recursion**: For $t = T-1, T-2, \ldots, 2, 1$ and all $1 \le i \le N$,

$$\beta_t(i) = \sum_{j=1}^{N} A_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$$

3: **Output**:

$$\Pr(o_{1:T}|\lambda) = \sum_{i=1}^{N} \pi_i b_i(o_1) \beta_1(i)\,. \tag{24}$$

The forward and backward procedures must give the same answer because they are computing the same probability. Figure 4 includes Matlab/Octave code to evaluate $\Pr(o_{1:T}|\lambda)$, whose results show that the forward and backward algorithms indeed return the same answer to the evaluation problem.

```
iter = 1;
for iter = 1:1000
    N = 3;  % number of states
    Pi = rand(1,N); Pi = Pi / sum(Pi); % prior distribution
    A = rand(N,N); % state transition matrix
    A(1,3) = 0; % cannot have a transition from state 1 to 3
    for i=1:N     A(i,:) = A(i,:) / sum(A(i,:));   end
    M = 3; % number of outputs
    B = rand(N,M); % output probability matrix
    for i=1:N     B(i,:) = B(i,:) / sum(B(i,:));   end
    T = 5; % number of time steps
    O = randi(M, 1, T); % outputs

    Alpha = zeros (T, N); % alpha
    Beta = ones (T, N);  % beta
    % Compute Alpha
    Alpha(1,:) = Pi .* B(:, O(1))';
    for t = 2:T
        Alpha(t,:) = (Alpha(t-1,:) * A) .* B(:,O(t))';
    end
    % Compute Beta
    for t = (T-1):-1:1
        Beta(t,:) = A * (B(:,O(t+1)) .* Beta(t+1,:)');
    end
    Gamma = Alpha.*Beta; % (unnormalized) gamma
    % two ways to compute the sequence probablity
    p1 = sum(Alpha(end,:));
    p2 = sum(Gamma(1,:));
    assert(abs(p1-p2)<1e-12);
    % can we find an invalid transition from state 1 to 3?
    [~,I]=max(Gamma');
    for i=1:T-1
        if I(i)==1 && I(i+1)==3
            disp(['1-->3 at iteration ' num2str(iter) '!'])
            return
        end
    end
end
```

Figure 4: Sample code to compute α, β, and (unnormalized) γ variables. Note that this code is only for illustrative purposes—it is not practical for solving real world HMM problems.

However, the code in Figure 4 is listed only for illustrative purposes. When T is a large number, the computed probability will be a very small number (e.g., $< 10^{-400}$). $\Pr(o_{1:T}|\lambda)$ may be smaller than the minimum floating point number that can be represented by a `float` (single precision) or `double` (double precision) type in a computer system. Numerical errors (such as rounding errors) will also quickly accumulate. Hence, implementing algorithms in hidden Markov models is complex and tricky. We will not dive into HMM implementation details, but interested readers can refer to publicly available HMM implementation source code such as the HTK package.[4]

4 γ, δ, ψ, and the decoding problem

Now we move on to the second basic problem: decoding. As we have discussed, one major question is: given an observation sequence $o_{1:T}$, what is the criterion to determine the best (or optimal) hidden state sequence?

4.1 γ and the independently decoded optimal states

One straightforward idea is to use a simple criterion: find the maximum likelihood state for each time step independently. That is, for any $1 \le t \le T$, we can compute $\Pr(Q_t = S_i|o_{1:T}, \lambda)$ for all $1 \le i \le N$, and set Q_t to the one leading to the maximum probability.

To solve this problem, we can define the γ variables as

$$\gamma_t(i) = \Pr(Q_t = S_i|o_{1:T}, \lambda)\,, \tag{25}$$

and set

$$q_t = \arg\max_{1\le i\le N} \gamma_t(i) \tag{26}$$

for all $1 \le t \le T$. We can then decode the hidden state Q_t as S_{q_t}.

$\gamma_t(i)$ is the probability of Q_t being S_i when we have observed the complete observation sequence $o_{1:T}$. In fact, we do not need to design a new algorithm to compute this variable: it can be easily computed from $\alpha_t(i)$ and $\beta_t(i)$.

Because $O_{1:t}$ and $O_{t+1:T}$ are independent of each other if Q_t is known, we have

$$\begin{aligned}
\Pr(Q_t = S_i, o_{1:T}|\lambda) &= \Pr(o_{1:t}, o_{t+1:T}|Q_t = S_i, \lambda)\Pr(Q_t = S_i|\lambda) && (27)\\
&= \Pr(o_{1:t}|Q_t = S_i, \lambda)\Pr(o_{t+1:T}|Q_t = S_i, \lambda)\Pr(Q_t = S_i|\lambda) && (28)\\
&= \Pr(o_{t+1:T}|Q_t = S_i, \lambda)\Pr(Q_t = S_i, o_{1:t}|\lambda) && (29)\\
&= \alpha_t(i)\beta_t(i)\,. && (30)
\end{aligned}$$

Then, we can compute $\gamma_t(i)$ as

$$\gamma_t(i) = \frac{\Pr(Q_t = S_i, o_{1:T}|\lambda)}{\Pr(o_{1:T}|\lambda)} \tag{31}$$

[4] http://htk.eng.cam.ac.uk/

$$= \frac{\Pr(Q_t = S_i, o_{1:T}|\lambda)}{\sum_{j=1}^{N} \Pr(Q_t = S_j, o_{1:T}|\lambda)} \tag{32}$$

$$= \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \,. \tag{33}$$

To compute the γ variables, we can first set $\gamma_t(i) = \alpha_t(i)\beta_t(i)$, and then ℓ_1 normalize the γ values for every time step t. Note that we can find q_t by finding the maximum element in the unnormalized γ values (i.e., $\alpha_t(i)\beta_t(i)$) without the normalization, because the normalization constant $\Pr(o_{1:T}|\lambda)$ will not change the index of the largest γ value at time t.

One byproduct of this derivation is: now we have three equivalent ways to solve the evaluation problem: using α, β, and $\alpha\beta$. Based on the above derivation, we have

$$\Pr(o_{1:T}|\lambda) = \sum_{i=1}^{N} \alpha_t(i)\beta_t(i) \tag{34}$$

for *any* time t!

This criterion finds the best state for each time t independently, which may lead to state sequences that should not appear. For example, in Figure 4, we have set the transition probability from S_1 to S_3 as zero—i.e., this transition must never happen. However, if we use γ to decode hidden states independently, running that piece of code shows that this impossible transition indeed happens in the decoded state sequences. Hence, this criterion (Equation 26) has serious problems in some applications.

4.2 δ, ψ, and the jointly decoded optimal states

To eliminate impossible state transitions, one possible solution is to jointly decode the entire optimal state sequence:

$$q_{1:T} = \arg\max_{Q_{1:T}} \Pr(Q_{1:T}|o_{1:T}, \lambda) \tag{35}$$

$$= \arg\max_{Q_{1:T}} \Pr(Q_{1:T}, o_{1:T}|\lambda) \,. \tag{36}$$

The key to its solution is once again dynamic programming. The recursive relationship is as follows. Given $o_{1:t}$, it is very useful if we know the optimal paths for N subproblems:

$$\max_{Q_{1:t-1}} \Pr(Q_{1:t-1}, o_{1:t}, Q_t = S_i|\lambda), \quad 1 \le i \le N \,.$$

These subproblems are similar to the objective in Equation 36, but with an additional constraint $Q_t = S_i$ for the i-th subproblem. We can solve at least the following two problems using the answers to these N subproblems:

- The optimal state sequence till time t. If a subproblem $i^\star$ has the largest probability among all N subproblems, then the optimal parameter of this

subproblem ($q_{1:t-1}$) plus $q_t = i^\star$ is the optimal hidden state sequence for observations $o_{1:t}$.

- The optimal $q_{1:t+1}$ for $o_{1:t+1}$ can be divided into three parts: $q_{1:t-1}$, q_t, and q_{t+1}. If an oracle tells us $q_t = i^\star$, the $q_{1:t-1}$ part can be found by the $i^\star$-th subproblem. Because of the Markov assumption, we only need to consider N possible transitions $S_{q_t} \to S_{q_{t+1}}$ to decide which state is optimal for Q_{t+1}. Although we do not have access to an oracle, we can try all N subproblems (i.e., $i^\star = 1, 2, \ldots, N$), which is still tractable.

The objectives of these subproblems are denoted by a new variable

$$\delta_t(i) = \max_{Q_{1:t-1}} \Pr(Q_{1:t-1}, o_{1:t}, Q_t = S_i|\lambda)\,. \tag{37}$$

The recursive relationship is also obvious (by translating the above description into mathematics):

$$\delta_{t+1}(i) = \max_{1\le j\le N} \left(\delta_t(j)A_{ji}b_i(o_{t+1})\right)\,, \tag{38}$$

in which $\delta_t(j)$ is the probability of the j-th subproblem, A_{ji} transits from S_j (at time t) to S_i (at time $t+1$), and $b_i(o_{t+1})$ is the probability of observing $V_{o_{t+1}}$ when the state is S_i; i.e., $\delta_t(j)A_{ji}b_i(o_{t+1})$ is the probability of the optimal state sequence when an oracle tells us $Q_t = S_j$ for an observation sequence $o_{1:t+1}$. This recursive relationship is a forward one. Hence, we should start the recursion from $t = 1$.

After the δ variables are computed, it is easy to find q_T, by

$$q_T = \arg\max_{1\le i\le N} \delta_T(i)\,. \tag{39}$$

According to Equation 38, if we know the optimal state at time $t+1$ is q_{t+1}, we just need to find which j leads to the largest $\delta_t(j)A_{ji}b_i(o_{t+1})$; then S_j is the optimal state for Q_t. That is, we need to record the optimal transitions from time t to $t+1$. Using $\psi_{t+1}(i)$ to denote the optimal state at time t if the optimal state is i at time $t+1$, we have

$$\psi_{t+1}(i) = \arg\max_{1\le j\le N} \left(\delta_t(j)A_{ji}b_i(o_{t+1})\right) \tag{40}$$

$$= \arg\max_{1\le j\le N} \left(\delta_t(j)A_{ji}\right)\,. \tag{41}$$

The initialization should start at $t = 1$. According to the definition of the δ variables, we have

$$\delta_1(i) = \pi_i b_i(o_1)$$

for $1 \le i \le N$. Putting the initialization, recursion, and state tracking equations together, we get the Viterbi algorithm for decoding the optimal hidden

Algorithm 3 Viterbi decoding

1: **Initialization**: $\delta_1(i) = \pi_i b_i(o_1)$, $\psi_1(i) = 0$ for all $1 \le i \le N$

2: **Forward recursion**: For $t = 2, 3, \ldots, T-2, T-1$ and all $1 \le i \le N$,

$$\delta_{t+1}(i) = \max_{1 \le j \le N} \left(\delta_t(j) A_{ji} b_i(o_{t+1}) \right) , \tag{42}$$

$$\psi_{t+1}(i) = \arg\max_{1 \le j \le N} \left(\delta_t(j) A_{ji} \right) . \tag{43}$$

3: **Output**: The optimal state q_T is determined by

$$q_T = \arg\max_{1 \le i \le N} \delta_T(i) , \tag{44}$$

and the rest of the optimal path is determined by: for $t = T-1, T-2, \ldots, 2, 1$

$$q_t = \psi_{t+1}(q_{t+1}) . \tag{45}$$

states that is the solution of Equation 36. The Viterbi algorithm is shown in Algorithm 3.[5]

Note that the initialization $\psi_1(i) = 0$ is in fact not used at all in the Viterbi algorithm. The complexity of Viterbi decoding is $\mathcal{O}(TN^2)$.

Equations 42 and 43 involve the sum or maximization of several probabilities, respectively. As $\delta_t(j)$ and A_{ji} form two discrete distributions when j varies from 1 to N, we can treat them as *messages* passing between Q_t and Q_{t+1}. Message passing algorithms (such as sum-product and max-product) can solve many inference problems in graphical models, with the forward, backward, and Viterbi algorithms all as special cases in the message passing family.

5 ξ and learning HMM parameters

Given N, M, and a training sequence $o_{1:T}$, to learn the optimal parameters $\lambda = (\boldsymbol{\pi}, A, B)$ is the most difficult among the three basic problems. The classic algorithm is called the Baum–Welch algorithm.[6] Baum–Welch is a maximum likelihood (ML) estimation algorithm, which maximizes the likelihood of λ for the observation sequence $o_{1:T}$:

$$\lambda^\star = \arg\max_{\lambda} \Pr(o_{1:T}|\lambda) . \tag{46}$$

Note that we assume only one training sequence is used, but the generalization of Baum–Welch to multiple training sequences is easy to do.

[5] Andrew James Viterbi is an American electrical engineer and businessman. He is a co-founder of Qualcomm Inc. He proposed the Viterbi decoding algorithm in 1967, but did not patent it.

[6] This algorithm is named after Leonard Esau Baum and Lloyd Richard Welch, two American mathematicians.

Baum–Welch is an iterative algorithm. With initial (e.g., randomly initialized or by clustering the training sequence) parameters $\lambda^{(1)}$, we can compute its likelihood $\ell^{(1)} = \Pr(o_{1:T}|\lambda^{(1)})$. Then, we can find a new set of parameters $\lambda^{(2)}$ such that its likelihood $\ell^{(2)} = \Pr(o_{1:T}|\lambda^{(2)})$ is higher than $\ell^{(1)}$ (or at least the same). We can then move on to find the next set of better parameters $\lambda^{(3)}$, $\lambda^{(4)}$ and so on, until the likelihood converges.

Baum–Welch is in fact a special case of the more general Expectation-Maximization (EM) algorithm for maximum likelihood estimation. Hence, it is guaranteed to converge to a local maximum of the likelihood function. For more details of the EM algorithm, please refer to Chapter 14. In that chapter, there is also an exercise problem that derives Baum–Welch updating equations from the EM perspective.

5.1 Baum–Welch: Updating λ as expected proportions

The Baum–Welch algorithm uses a new variable ξ and a few simple equations to update $\lambda^{(r+1)}$ based on $\lambda^{(r)}$, in which r is the iteration number. In this chapter, we will ignore the proof that Baum–Welch always increases or keeps the same likelihood. Instead, we will focus on the intuitions behind the ξ variables and the updating equations.

The ξ variable involves three other values: t (the time) and (i, j), which are state indexes:

$$\xi_t(i, j) = \Pr(Q_t = S_i, Q_{t+1} = S_j|o_{1:T}, \lambda)\,. \tag{47}$$

$\xi_t(i, j)$ is the conditional probability of the states for t and $t+1$ being S_i and S_j, respectively, when the observation sequence $o_{1:T}$ is presented. In other words, $\xi_t(i, j)$ is the *expected proportion* of transition from S_i (at time t) to S_j (at time $t+1$). Based on this interpretation, it is natural to use $\xi_t(i, j)$ (computed based on the parameters $\lambda^{(r)}$) to update the value for A_{ij} in $\lambda^{(r+1)}$!

For example, if there are 3 states $\{S_1, S_2, S_3\}$ and $\sum_{t=1}^{T-1} \xi_t(2, 1) = 100$, this means there are 100 (expected) transitions from S_2 to S_1 in the entire training sequence. Suppose $\sum_{t=1}^{T-1} \xi_t(2, 2) = 150$ and $\sum_{t=1}^{T-1} \xi_t(2, 3) = 250$. Then, it is natural to update A_{21} as

$$\frac{100}{100 + 150 + 250} = 0.2\,,$$

because this is the expected proportion of transitions from S_2 to S_1. Similarly, A_{22} and A_{23} can be updated by their estimated proportion 0.3 and 0.5, respectively. The same idea can be used to update $\boldsymbol{\pi}$ and B.

5.2 How to compute ξ

Using the definition of conditional probabilities, we have

$$\xi_t(i, j) \Pr(o_{1:T}|\lambda) = \Pr(Q_t = S_i, Q_{t+1} = S_j, o_{1:T}|\lambda)\,. \tag{48}$$

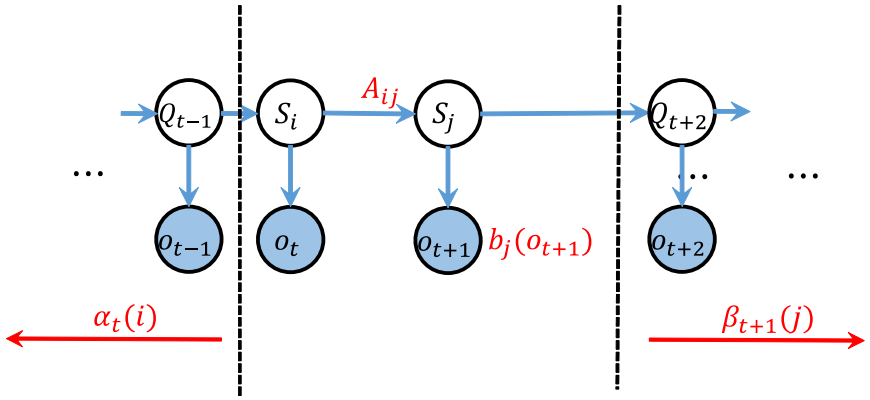


Figure 5: Illustration of how to compute $\xi_t(i,j)$. ()

Table 1: Summary of the variables in HMM learning.

	Definition	Recursion/Calculation
α	$\alpha_t(i) = \Pr(o_{1:t}, Q_t = S_i \lvert \lambda)$	$\alpha_{t+1}(i) = \left(\sum_{j=1}^{N} \alpha_t(j) A_{ji}\right) b_i(o_{t+1})$
β	$\beta_t(i) = \Pr(o_{t+1:T} \lvert Q_t = S_i, \lambda)$	$\beta_t(i) = \sum_{j=1}^{N} A_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$
γ	$\gamma_t(i) = \Pr(Q_t = S_i \lvert o_{1:T}, \lambda)$	$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}$
δ	$\delta_t(i) = \max_{Q_{1:t-1}} \Pr(Q_{1:t-1}, o_{1:t}, Q_t = S_i \lvert \lambda)$	$\delta_{t+1}(i) = \max_{1 \le j \le N} (\delta_t(j) A_{ji} b_i(o_{t+1}))$
ξ	$\xi_t(i,j) = \Pr(Q_t = S_i, Q_{t+1} = S_j \lvert o_{1:T}, \lambda)$	$\xi_t(i,j) = \frac{\alpha_t(i) A_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) A_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}$

Hence, we can find the probability $\Pr(Q_t = S_i, Q_{t+1} = S_j, o_{1:T}|\lambda)$ and use it to compute $\xi_t(i,j)$. This probability can be factored into the product of four probabilities: $\alpha_t(i)$, A_{ij}, $b_j(o_{t+1})$, and $\beta_{t+1}(j)$, as shown in Figure 5. For convenience of reading, we listed the HMM variables in Table 1. Now we have

$$\xi_t(i,j) = \frac{\alpha_t(i)A_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\Pr(o_{1:T}|\lambda)}\,. \tag{49}$$

Because $\xi_t(i,j)$ is a probability, we have $\sum_{i=1}^{N}\sum_{j=1}^{N}\xi_t(i,j) = 1$; hence, we know

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{\alpha_t(i)A_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\Pr(o_{1:T}|\lambda)} = 1\,, \tag{50}$$

or in an equivalent form

$$\Pr(o_{1:T}|\lambda) = \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)A_{ij}b_j(o_{t+1})\beta_{t+1}(j)\,, \tag{51}$$

for any $1 \le t \le T-1$. This equation provides yet another way to solve the evaluation problem.

And, comparing the definition of γ and ξ, we immediately get (by the law of total probability)

$$\gamma_t(i) = \sum_{j=1}^{N}\xi_t(i,j)\,. \tag{52}$$

The parameters $\lambda = (\boldsymbol{\pi}, A, B)$ can be updated using γ and ξ.

- Since $\gamma_1(i)$ is the expected proportion of $Q_1 = S_i$, we can update π_i using $\gamma_1(i)$.
- The expected probability of transition from S_i to S_j is $\xi_t(i,j)$ at time t. Hence, the expected number of transitions from S_i to S_j in the training sequence is $\sum_{t=1}^{T-1}\xi_t(i,j)$, while $\sum_{t=1}^{T-1}\gamma_t(i)$ is the expected number of times any state is S_i. Then, A_{ij}, the probability of transiting from S_i to S_j, is the proportion of transitions $S_i \to S_j$ in all transitions starting from S_i, i.e.,
$$\frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\gamma_t(i)}\,.$$
- To update B, we need to estimate two terms: the expected number of times in hidden state S_j ($\sum_{t=1}^{T}\gamma_t(j)$); and the number of times the hidden state is S_j *and* the observation is V_k at the same time ($\sum_{t=1}^{T}[\![o_t = k]\!]\gamma_t(j)$), in which $[\![\cdot]\!]$ is the indicator function. Then, we can update $b_j(k)$ as the ratio between these two terms.

Summarizing the results till now, we arrive at the Baum–Welch algorithm, which is described in Algorithm 4.

Algorithm 4 The Baum–Welch algorithm

Initialize the parameters $\lambda^{(1)}$ (e.g., randomly)
$r \leftarrow 1$
while the likelihood has not converged **do**
    Use the forward procedure to compute $\alpha_t(i)$ for all t $(1 \le t \le T)$ and all i $(1 \le i \le N)$ based on $\lambda^{(r)}$
    Use the backward procedure to compute $\beta_t(i)$ for all t $(1 \le t \le T)$ and all i $(1 \le i \le N)$ based on $\lambda^{(r)}$
    Compute $\gamma_t(i)$ for all t $(1 \le t \le T)$ and all i $(1 \le i \le N)$ according to the equation in Table 1
    Compute $\xi_t(i,j)$ for all t $(1 \le t \le T-1)$ and all i,j $(1 \le i,j \le N)$ according to the equation in Table 1
    Update the parameters to $\lambda^{(r+1)}$

$$\pi_i^{(r+1)} = \gamma_1(i) \qquad 1 \le i \le N \tag{53}$$

$$A_{ij}^{(r+1)} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \qquad 1 \le i,j \le N \tag{54}$$

$$b_j^{(r+1)}(k) = \frac{\sum_{t=1}^{T} [\![o_t = k]\!] \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \qquad 1 \le j \le N, 1 \le k \le M \tag{55}$$

    $r \leftarrow r + 1$
end while

Exercises

1. Suppose a DTMC models the evolution of a random variable X, which is discrete and has N possible values (states). Show that we need $N^2 - 1$ numbers to fully specify this DTMC.

2. Let A be the transition matrix of an HMM model. Prove $A^k = \underbrace{A \dots A}_{k\ A\text{'s}}$ is a right stochastic matrix for any positive integer k.

3. (Conditional independence) We say A and B are conditionally independent given C, denoted as

$$A \perp B \mid C\,,$$

if $p(A, B|C) = p(A|C)p(B|C)$ always holds. A, B, and C can be discrete or continuous and can be either single-variate or multivariate random variables. In this problem, we use various simple probabilistic graphical models in Figure 6 to illustrate the conditional dependence among sets of variables.

In a directed graphical model, the arrows indicate direct dependencies—one node depends on its parents (those who have edges pointing to it). For example, Figure 6a says C depends on A, B depends on C, but A depends on nothing—that is, the joint density can be factored as

$$p(A, B, C) = p(A)p(C|A)p(B|C)\,.$$

(a) For the simple case 1.1 in Figure 6a, prove that $A \perp B \mid C$.

(b) For the simple case 1.2 in Figure 6b, prove that $A \perp B \mid C$.

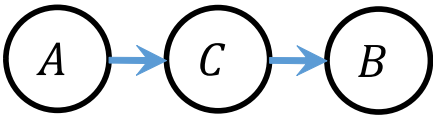
(c) For the simple case 2 in Figure 6c, prove that $A \perp B \mid C$.

(d) Case 3.1 in Figure 6d is a bit more delicate. Show that when C is *not* observed, we have $p(A, B) = p(A)p(B)$—that is, A and B are independent. However, when C is observed, A and B are *not* conditionally independent. Try to find an intuitive example to explain this phenomenon.

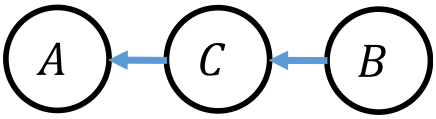
This phenomenon is called *explaining away*. When two (or more) causes can both cause the same effect, these causes become dependent on each other after we observe that effect.

(e) Case 3.2, which is a variant of case 3.1, is shown in Figure 6e. Explain intuitively the following fact: even if C is not observed, A and B become dependent when any of C's descendants is observed.

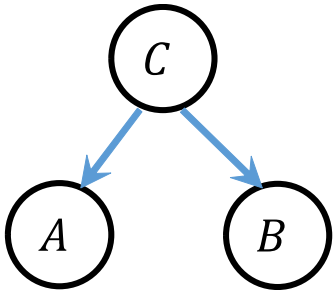
4. (d-separation) We have been quite sloppy in this chapter when dealing with dependence or independence among variables in the hidden Markov model. In this problem, we will introduce d-separation, an algorithm that can precisely determine any conditional dependence or independence in HMM. In fact, d-separation works well in Bayesian networks, a more



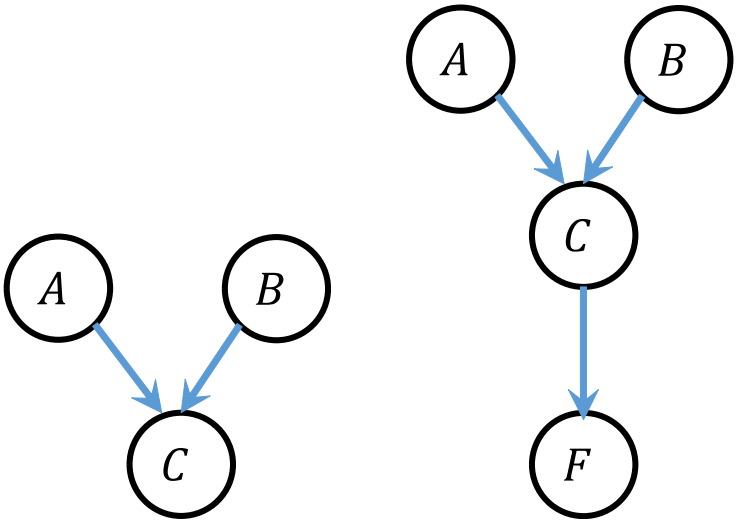
(a) Case 1.1



(b) Case 1.2



(c) Case 2



(d) Case 3.1 (e) Case 3.2

Figure 6: Various graphical model structures.

general class of probabilistic graphical models. HMM is an example of a Bayesian network.

Let A, B, and C be three sets of random variables, which are denoted as nodes in a directed probabilistic graphical model. A *trail* is an undirected path (i.e., ignoring the arrow directions and without any loop) in the graph. We say a trail is *d-separated* by Z if one or more of the following three situations happen:

i. There is a directed chain and one of its middle nodes (i.e., excluding the starting and the ending nodes) is in Z. Case 1.1 and 1.2 in Figure 6 are examples of such cases, but a directed chain can have more than three nodes.

ii. There are nodes in the path that form a "common cause" (i.e., case 2 in Figure 6c) and the middle node is in Z.

iii. There are nodes in the path that form a "common effect" (i.e., case 3.1 or case 3.2 in Figure 6d and Figure 6e, respectively), and the middle node is *not* in Z. Furthermore, *none of the descendants* of the middle node is in Z. Note that the descendant of the middle node may *not* be in the path.

Let u be a node in A and v be a node in B; let P be a trail that starts with u and ends at v. The d-separation rule states that $A \perp B \mid C$ if and only if all P is d-separated by Z for an arbitrary such trail P.

Use the d-separation rule to decide whether the following statements concerning Figure 7 are correct or not. Justify your answers.

(a) $B \perp C \mid A$

(b) $C \perp D \mid F$

5. Prove the joint distribution of an HMM model is correctly calculated by Equation 7. (Hint: Mathematical induction is useful.)

6. Prove the following equations. (Hint: use d-separation to determine conditional independence.)

   (a) $\alpha_{t+1}(i) = \left(\sum_{j=1}^{N} \alpha_t(j) A_{ji}\right) b_i(o_{t+1})$;

   (b) $\beta_t(i) = \sum_{j=1}^{N} A_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$;

   (c) $\Pr(o_{1:T}|\lambda) = \sum_{i=1}^{N} \pi_i b_i(o_1) \beta_1(i)$.

7. (n-step transition matrix) The transition matrix A can also be called a one-step transition matrix, because A_{ij} is the probability of transferring from one state S_i to another state S_j in one time step (though it is possible that $i = j$). The n-step transition matrix $A(n)$ is defined as

$$A_{ij}(n) \triangleq \Pr(X_{t+n} = S_j | X_t = S_i)\,, \tag{56}$$

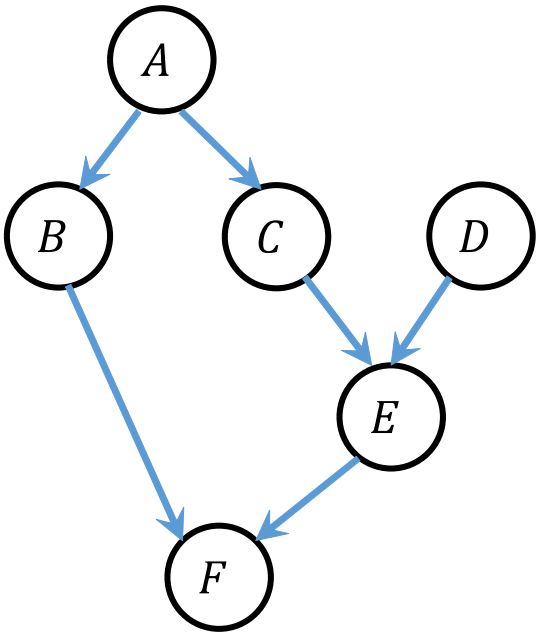


Figure 7: Example of d-separation.

i.e., the probability of transferring from one state S_i to another state S_j in *exactly* n one-step transitions.

The *Chapman–Kolmogorov equations* state that

$$A_{ij}(m+n) = \sum_{k=1}^{N} A_{ik}(m) A_{kj}(n)\,, \tag{57}$$

in which m and n are positive integers.

(a) Explain the meaning of the Chapman–Kolmogorov equations.

(b) Use the Chapman–Kolmogorov equations to find $A(n)$, whose (i,j)-th entry is $A_{ij}(n)$.

(c) Show that $\mathbf{1} \in \mathbb{R}^N$ (a vector of all '1's) is an eigenvector of $A(n)$ for any positive integer n.