Chapter 4

Hidden Markov Models (HMMs)

4.1 Definition of a Hidden Markov Model (HMM)

There is a variant of the notion of DFA with output, for example a transducer such as a gsm (generalized sequential machine), which is widely used in machine learning.

This machine model is known as *hidden Markov model*, for short *HMM*.

There are three new twists compared to traditional gsm models:

(1) There is a finite set of states Q with n elements, a bijection $\sigma\colon Q \to \{1, \ldots, n\}$, and *the transitions between states are labeled with probabilities* rather that symbols from an alphabet. For any two states p and q in Q, the edge from p to q is labeled with a probability $A(i, j)$, with $i = \sigma(p)$ and $j = \sigma(q)$.

The probabilities $A(i,j)$ form an $n \times n$ matrix $A = (A(i,j))$.

(2) There is a finite set $\mathbb{O}$ of size m (called the *observation space*) of possible outputs that can be emitted, a bijection $\omega\colon \mathbb{O} \to \{1,\dots,m\}$, and for every state $q \in Q$, *there is a probability $B(i,j)$ that output $O \in \mathbb{O}$ is emitted (produced)*, with $i = \sigma(q)$ and $j = \omega(O)$.

The probabilities $B(i,j)$ form an $n \times m$ matrix $B = (B(i,j))$.

(3) *Sequences of outputs* $\mathcal{O} = (O_1,\dots,O_T)$ (with $O_t \in \mathbb{O}$ for $t = 1,\dots,T$) emitted by the model are *directly observable*, but the sequences of states $\mathcal{S} = (q_1,\dots,q_T)$ (with $q_t \in Q$ for $t = 1,\dots,T$) that caused some sequence of output to be emitted are *not observable*.

In this sense the states are hidden, and this is the reason for calling this model a *hidden Markov model*.

Example 4.1. Say we consider the following behavior of some professor at some university.

On a *hot day* (denoted by Hot), the professor comes to class *with a drink* (denoted D) with probability 0.7, and *with no drink* (denoted N) with probability 0.3.

On the other hand, on a *cold day* (denoted Cold), the professor comes to class *with a drink* with probability 0.2, and *with no drink* with probability 0.8.

Suppose a student intrigued by this behavior recorded a sequence showing whether the professor came to class with a drink or not, say NNND.

Several months later, *the student would like to know whether the weather was hot or cold the days he recorded the drinking behavior of the professor*.

Now the student heard about machine learning, so he constructs a probabilistic (hidden Markov) model of the weather.

Based on some experiments, he determines the probability of a going from a hot day to another hot day to be 0.75, the probability of a going from a hot day to a cold day to be 0.25, the probability of going from a cold day to another cold day to be 0.7, and the probability of going from a cold day to a hot day to be 0.3.

He also knows that when he started his observations, it was a *cold day* with probability 0.45, and a *hot day* with probability 0.55.

The above data determine an HMM depicted in Figure 4.1.

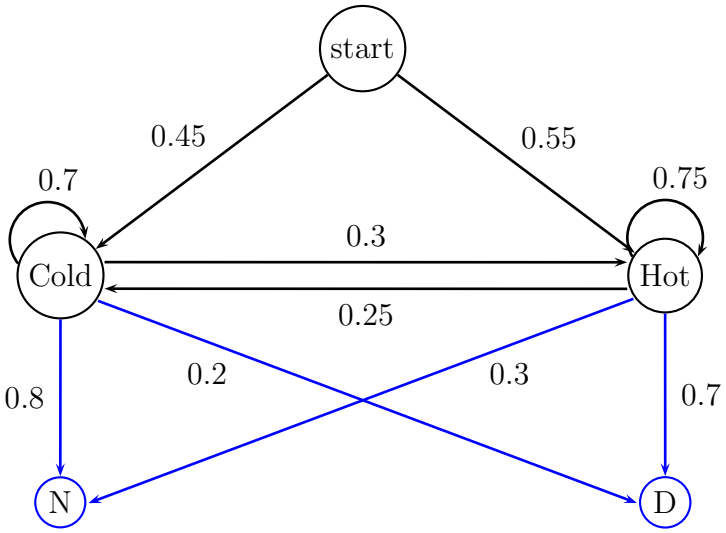


Figure 4.1: Example of an HMM modeling the "drinking behavior" of a professor at the University of Pennsylvania.

In this example, the set of states is $Q = \{\text{Cold}, \text{Hot}\}$, and the set of outputs is $\mathbb{O} = \{\text{N}, \text{D}\}$.

We have the bijection $\sigma\colon \{\text{Cold}, \text{Hot}\} \to \{1, 2\}$ given by $\sigma(\text{Cold}) = 1$ and $\sigma(\text{Hot}) = 2$, and the bijection $\omega\colon \{\text{N}, \text{D}\} \to \{1, 2\}$ given by $\omega(\text{N}) = 1$ and $\omega(\text{D}) = 2$.

The portion of the state diagram involving the states Cold, Hot, is analogous to an NFA in which the transition labels are probabilities; it is the underlying Markov model of the HMM.

For any given state, the probabilities of the outgoing edges sum to 1.

The start state is a convenient way to express the probabilities of starting either in state Cold or in state Hot.

Also, from each of the states Cold and Hot, we have emission probabilities of producing the ouput N or D, and these probabilities also sum to 1.

We can also express these data using *matrices*.

The matrix

$$A = \begin{pmatrix} 0.7 & 0.3 \\ 0.25 & 0.75 \end{pmatrix}$$

describes the transitions of the Markov model,

the vector

$$\pi = \begin{pmatrix} 0.45 \\ 0.55 \end{pmatrix}$$

describes the probabilities of starting either in state Cold or in state Hot,

and the matrix

$$B = \begin{pmatrix} 0.8 & 0.2 \\ 0.3 & 0.7 \end{pmatrix}$$

describes the emission probabilities.

The student would like to solve what is known as the decoding problem.

Namely, *given the output sequence NNND, find the most likely state sequence of the Markov model that produces the output sequence NNND.*

Is it (Cold, Cold, Cold, Cold), or (Hot, Hot, Hot, Hot), or (Hot, Cold, Cold, Hot), or (Cold, Cold, Cold, Hot)?

Given the probabilities of the HMM, it seems unlikely that it is (Hot, Hot, Hot, Hot), but how can we find the most likely one?

Before going any further, we wish to address a notational issue.

The issue is how to denote the states, the ouputs, as well as (ordered) sequences of states and sequences of output.

In most problems, states and outputs have "meaningful" names.

For example, if we wish to describe the evolution of the temperature from day to day, it makes sense to use two states "Cold" and "Hot," and to describe whether a given individual has a drink by "D," and no drink by "N."

Thus our set of states is $Q = \{\text{Cold}, \text{Hot}\}$, and our set of outputs is $\mathbb{O} = \{\text{N}, \text{D}\}$.

However, when computing probabilities, we need to use *matrices* whose rows and columns are indexed by positive integers, so we need a mechanism to associate a *numerical index* to every state and to every output, and this is the purpose of the bijections $\sigma\colon Q \to \{1, \ldots, n\}$ and $\omega\colon \mathbb{O} \to \{1, \ldots, m\}$.

In our example, we define σ by $\sigma(\text{Cold}) = 1$ and $\sigma(\text{Hot}) = 2$, and ω by $\omega(\text{N}) = 1$ and $\omega(\text{D}) = 2$.

Some author circumvent (or do they?) this notational issue by assuming that the set of outputs is $\mathbb{O} = \{1, 2, \ldots, m\}$, and that the set of states is $Q = \{1, 2, \ldots, n\}$.

The disadvantage of doing this is that in "real" situations, it is often more convenient to name the outputs and the states with more meaningful names than $1, 2, 3$ *etc.*

Warning: The task of naming the elements of the output alphabet can be challenging, for example in speech recognition.

Let us now turn to *sequences*.

For example, consider the sequence of six states (from the set $Q = \{\text{Cold}, \text{Hot}\}$),

$$\mathcal{S} = (\text{Cold}, \text{Cold}, \text{Hot}, \text{Cold}, \text{Hot}, \text{Hot}).$$

Using the bijection $\sigma\colon \{\text{Cold}, \text{Hot}\} \to \{1, 2\}$ defined above, the sequence $\mathcal{S}$ is completely determined by the sequence of indices

$$\begin{aligned}\sigma(\mathcal{S}) = (&\sigma(\text{Cold}), \sigma(\text{Cold}), \sigma(\text{Hot}), \sigma(\text{Cold}),\\ &\sigma(\text{Hot}), \sigma(\text{Hot})) = (1, 1, 2, 1, 2, 2).\end{aligned}$$

More generally, we will denote a sequence of length T of states from a set Q of size n by

$$\mathcal{S} = (q_1, q_2, \ldots, q_T),$$

with $q_t \in Q$ for $t = 1, \ldots, T$.

Using the bijection $\sigma\colon Q \to \{1, \ldots, n\}$, the sequence $\mathcal{S}$ is completely determined by the sequence of indices

$$\sigma(\mathcal{S}) = (\sigma(q_1), \sigma(q_2), \ldots, \sigma(q_T)),$$

where $\sigma(q_t)$ is some index from the set $\{1, \ldots, n\}$, for $t = 1, \ldots, T$.

The problem now is, *what is a better notation for the index denoted by $\sigma(q_t)$?*

Of course, we could use $\sigma(q_t)$, but this is a heavy notation, so *we adopt the notational convention to denote the index $\sigma(q_t)$ by i_t.*

Remark: We contemplated using the notation σ_t for $\sigma(q_t)$ instead of i_t. However, we feel that this would deviate too much from the common practice found in the literature, which uses the notation i_t.

Going back to our example

$$\mathcal{S} = (q_1, q_2, q_3, q_4, q_4, q_6) = (\text{Cold}, \text{Cold}, \text{Hot}, \text{Cold}, \text{Hot}, \text{Hot}),$$

we have

$$\sigma(\mathcal{S}) = (\sigma(q_1), \sigma(q_2), \sigma(q_3), \sigma(q_4), \sigma(q_5), \sigma(q_6)) = (1, 1, 2, 1, 2, 2),$$

so the sequence of indices
$(i_1, i_2, i_3, i_4, i_5, i_6) = (\sigma(q_1), \sigma(q_2), \sigma(q_3), \sigma(q_4), \sigma(q_5), \sigma(q_6))$ is given by

$$\sigma(\mathcal{S}) = (i_1, i_2, i_3, i_4, i_5, i_6) = (1, 1, 2, 1, 2, 2).$$

So, the fourth index i_4 is has the value 1.

We apply a similar convention to sequences of outputs.

For example, consider the sequence of six outputs (from the set $\mathbb{O} = \{\mathrm{N}, \mathrm{D}\}$),

$$\mathcal{O} = (\mathrm{N}, \mathrm{D}, \mathrm{N}, \mathrm{N}, \mathrm{N}, \mathrm{D}).$$

Using the bijection $\omega\colon \{\mathrm{N}, \mathrm{D}\} \to \{1, 2\}$ defined above, the sequence $\mathcal{O}$ is completely determined by the sequence of indices

$$\begin{aligned}\omega(\mathcal{O}) &= (\omega(\mathrm{N}), \omega(\mathrm{D}), \omega(\mathrm{N}), \omega(\mathrm{N}), \omega(\mathrm{N}), \omega(\mathrm{D})) \\ &= (1, 2, 1, 1, 1, 2).\end{aligned}$$

More generally, we will denote a sequence of length T of outputs from a set $\mathbb{O}$ of size m by

$$\mathcal{O} = (O_1, O_2, \ldots, O_T),$$

with $O_t \in \mathbb{O}$ for $t = 1, \ldots, T$.

Using the bijection $\omega \colon \mathbb{O} \to \{1, \ldots, m\}$, the sequence $\mathcal{O}$ is completely determined by the sequence of indices

$$\omega(\mathcal{O}) = (\omega(O_1), \omega(O_2), \ldots, \omega(O_T)),$$

where $\omega(O_t)$ is some index from the set $\{1, \ldots, m\}$, for $t = 1, \ldots, T$.

This time, *we adopt the notational convention to denote the index $\omega(O_t)$ by ω_t*.

Going back to our example

$$\mathcal{O} = (O_1, O_2, O_3, O_4, O_5, O_6) = (\mathrm{N}, \mathrm{D}, \mathrm{N}, \mathrm{N}, \mathrm{N}, \mathrm{D}),$$

we have

$$\begin{aligned} \omega(\mathcal{O}) &= (\omega(O_1), \omega(O_2), \omega(O_3), \omega(O_4), \omega(O_5), \omega(O_6)) \\ &= (1, 2, 1, 1, 1, 2), \end{aligned}$$

so the sequence of indices
$(\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6) = (\omega(O_1), \omega(O_2), \omega(O_3), \omega(O_4),$
$\omega(O_5), \omega(O_6))$ is given by

$$\omega(\mathcal{O}) = (\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6) = (1, 2, 1, 1, 1, 2).$$

HMM's are among the most effective tools to solve the following types of problems:

(1) **DNA and protein sequence alignment** in the face of mutations and other kinds of evolutionary change.

(2) **Speech understanding systems**, also called **Automatic speech recognition**. When we talk, our mouths produce sequences of sounds from the sentences that we want to say. This process is complex.

Multiple words may map to the same sound, words are pronounced differently as a function of the word before and after them, we all form sounds slightly differently, and so on.

All a listener can hear (perhaps a computer system) is the sequence of sounds, and the listener would like to reconstruct the mapping (backward) in order to determine what words we were attempting to say.

For example, when you "talk to your TV" to pick a program, say *game of thrones*, you don't want to get *Jessica Jones*.

(3) **Optical character recognition (OCR)**. When we write, our hands map from an idealized symbol to some set of marks on a page (or screen).

The marks are observable, but the process that generates them isn't.

A system performing OCR, such as a system used by the post office to read addresses, must discover which word is most likely to correspond to the mark it reads.

The reader should review Example 4.1 illustrating the notion of HMM.

Let us consider another example taken from Stamp [**?**].

Example 4.2. Suppose we want to determine the average annual temperature at a particular location over a series of years in a distant past where thermometers did not exist.

Since we can't go back in time, we look for indirect evidence of the temperature, say in terms of the *size of tree growth rings*.

For simplicity, assume that we consider the two temperatures Cold and Hot, and three different sizes of tree rings: small, medium and large, which we denote by S, M, L.

In this example, the set of states is $Q = \{\text{Cold}, \text{Hot}\}$, and the set of outputs is $\mathbb{O} = \{\text{S}, \text{M}, \text{L}\}$.

We have the bijection $\sigma\colon \{\text{Cold}, \text{Hot}\} \to \{1, 2\}$ given by $\sigma(\text{Cold}) = 1$ and $\sigma(\text{Hot}) = 2$, and the bijection $\omega\colon \{\text{S}, \text{M}, \text{L}\} \to \{1, 2, 3\}$ given by $\omega(\text{S}) = 1$, $\omega(\text{M}) = 2$, and $\omega(\text{L}) = 3$.

The HMM shown in Figure 4.2 is a model of the situation.

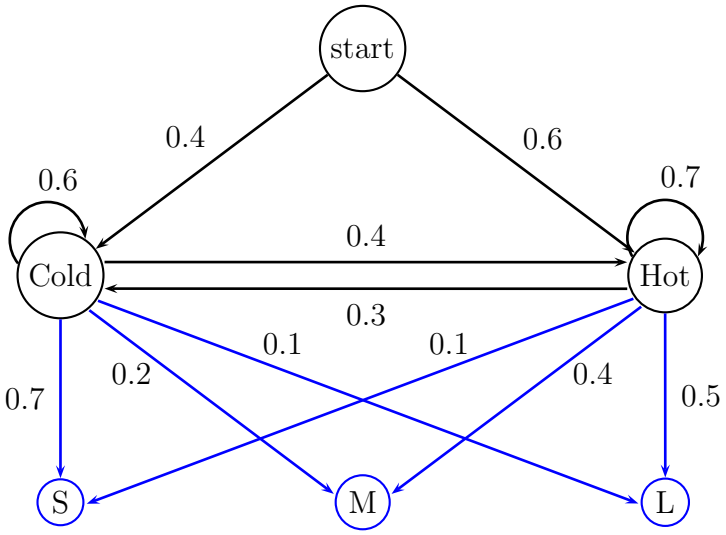


Figure 4.2: Example of an HMM modeling the temperature in terms of tree growth rings.

Suppose we observe the sequence of tree growth rings (S, M, S, L).

What is *the most likely sequence* of temperatures over a four-year period which yields the observations (S, M, S, L)?

Going back to Example 4.1, which corresponds to the HMM graph shown in Figure 4.3, *we need to figure out the probability that a sequence of states* $\mathcal{S} = (q_1, q_2, \ldots, q_T)$ *produces the output sequence* $\mathcal{O} = (O_1, O_2, \ldots, O_T)$.

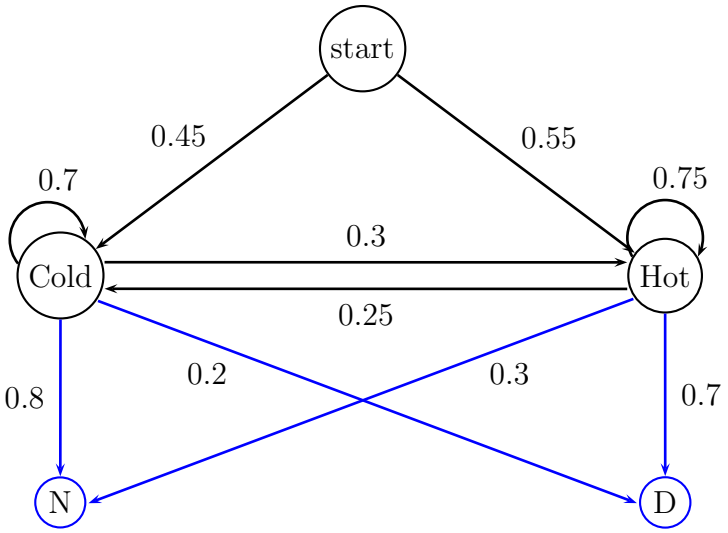


Figure 4.3: Example of an HMM modeling the "drinking behavior" of a professor at the University of Pennsylvania.

Then the probability that we want is just the product of the probability that we begin with state q_1, times the product of the probabilities of each of the transitions, times the product of the emission probabilities.

With our notational conventions, $\sigma(q_t) = i_t$ and $\omega(O_t) = \omega_t$, so we have

$$\mathsf{Pr}(\mathcal{S}, \mathcal{O}) = \pi(i_1)B(i_1, \omega_1) \prod_{t=2}^{T} A(i_{t-1}, i_t)B(i_t, \omega_t).$$

In our example, $\omega(\mathcal{O}) = (\omega_1, \omega_2, \omega_3, \omega_4) = (1, 1, 1, 2)$, which corresponds to NNND.

The brute-force method is to compute these probabilities for all $2^4 = 16$ sequences of states of length 4 (in general, there are n^T sequences of length T).

For example, for the sequence $\mathcal{S} = (\text{Cold}, \text{Cold}, \text{Cold}, \text{Hot})$, associated with the sequence of indices $\sigma(\mathcal{S}) = (i_1, i_2, i_3, i_4) = (1, 1, 1, 2)$, we find that

$$\begin{aligned}\mathsf{Pr}(\mathcal{S}, \text{NNND}) &= \pi(1)B(1,1)A(1,1)B(1,1)A(1,1)B(1,1) \\ &\quad A(1,2)B(2,2) \\ &= 0.45 \times 0.8 \times 0.7 \times 0.8 \times 0.7 \times 0.8 \\ &\quad \times 0.3 \times 0.7 = 0.0237.\end{aligned}$$

A much more efficient way to proceed is to use a method based on *dynamic programming*.

Recall the bijection $\sigma : \{\text{Cold}, \text{Hot}\} \to \{1, 2\}$, so that we will refer to the state Cold as 1, and to the state Hot as 2.

For $t = 1, 2, 3, 4$, for every state $i = 1, 2$, *we compute* $score(i, t)$ *to be the highest probability that a sequence of length* t *ending in state* i *produces the output sequence* $(O_1, \ldots, O_t)$, and for $t \geq 2$, we let $pred(i, t)$ be *the state that precedes* i *in a best sequence of length* t *ending in* i.

Initially, we set

$$score(j, 1) = \pi(j)B(j, \omega_1), \quad j = 1, 2,$$

and since $\omega_1 = 1$ we get $score(1, 1) = 0.45 \times 0.8 = 0.36$ and $score(2, 1) = 0.55 \times 0.3 = 0.165$.

Next we compute $score(1, 2)$ and $score(2, 2)$ as follows.

For $j = 1, 2$, for $i = 1, 2$, compute temporary scores

$$tscore(i, j) = score(i, 1)A(i, j)B(j, \omega_2);$$

then pick the *best* of the temporary scores,

$$score(j, 2) = \max_i tscore(i, j).$$

Since $\omega_2 = 1$, we get $tscore(1,1) = 0.36 \times 0.7 \times 0.8 = 0.2016$, $tscore(2,1) = 0.165 \times 0.25 \times 0.8 = 0.0330$, and $tscore(1,2) = 0.36 \times 0.3 \times 0.3 = 0.0324$, $tscore(2,2) = 0.165 \times 0.75 \times 0.3 = 0.0371$.

Then

$$\begin{aligned} score(1,2) &= \max\{tscore(1,1), tscore(2,1)\} \\ &= \max\{0.2016, 0.0330\} = 0.2016, \end{aligned}$$

and

$$\begin{aligned} score(2,2) &= \max\{tscore(1,2), tscore(2,2)\} \\ &= \max\{0.0324, 0.0371\} = 0.0371. \end{aligned}$$

Since the state that leads to the optimal score score$(1,2)$ is 1, we let $pred(1,2) = 1$, and since the state that leads to the optimal score score$(2,2)$ is 2, we let $pred(2,2) = 2$.

We compute $score(1,3)$ and $score(2,3)$ in a similar way.

For $j = 1, 2$, for $i = 1, 2$, compute

$$tscore(i,j) = score(i,2)A(i,j)B(j,\omega_3);$$

then pick the *best* of the temporary scores,

$$score(j,3) = \max_i tscore(i,j).$$

Since $\omega_3 = 1$, we get

$$score(1,3) = \max\{0.1129, 0.0074\} = 0.1129,$$

and

$$score(2,3) = \max\{0.0181, 0.0083\} = 0.0181.$$

We also get $pred(1,3) = 1$ and $pred(2,3) = 1$.

Finally, we compute $score(1,4)$ and $score(2,4)$ in a similar way.

For $j = 1, 2$, for $i = 1, 2$, compute

$$tscore(i, j) = score(i, 3)A(i, j)B(j, \omega_4);$$

then pick the *best* of the temporary scores,

$$score(j, 4) = \max_i tscore(i, j).$$

Since $\omega_4 = 2$, we get

$$score(1, 4) = \max\{0.0158, 0.0009\} = 0.0158,$$

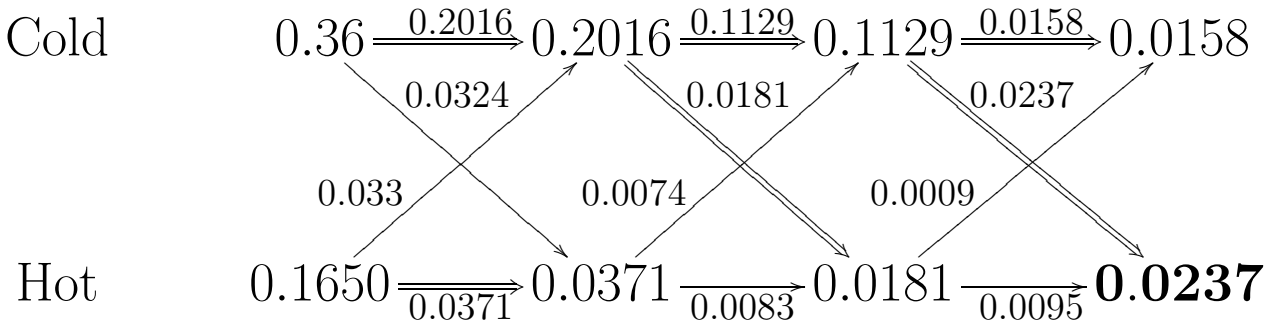
and

$$score(2, 4) = \max\{0.0237, 0.0095\} = 0.0237,$$

and $pred(1, 4) = 1$ and $pred(2, 4) = 1$.

Since $\max\{score(1, 4), score(2, 4)\} = 0.0237$, *the state with the maximum score is* Hot, and by following the predecessor list (also called backpointer list), we find the most likely state sequence to produce the sequence NNND to be (Cold, Cold, Cold, Hot).

The stages of the computations of $score(j, t)$ for $i = 1, 2$ and $t = 1, 2, 3, 4$ can be recorded in the following diagram called a *lattice* or a *trellis* (which means lattice in French!):



Double arrows represent the predecessor edges.

For example, the predecessor $pred(2, 3)$ of the third node on the bottom row labeled with the score 0.0181 (which corresponds to Hot), is the second node on the first row labeled with the score 0.2016 (which corresponds to Cold).

The two incoming arrows to the third node on the bottom row are labeled with the temporary scores 0.0181 and 0.0083.

The node with the highest score at time $t = 4$ is Hot, with score 0.0237 (showed in bold), and by following the double arrows backward from this node, we obtain the most likely state sequence (Cold, Cold, Cold, Hot).

The method we just described is known as the *Viterbi algorithm*.

Definition 4.1. A *hidden Markov model*, for short *HMM*, is a quintuple $M = (Q, \mathbb{O}, \pi, A, B)$ where

- Q is a finite set of *states* with n elements, and there is a bijection $\sigma\colon Q \to \{1, \ldots, n\}$.
- $\mathbb{O}$ is a finite *output alphabet* (also called *set of possible observations*) with m observations, and there is a bijection $\omega\colon \mathbb{O} \to \{1, \ldots, m\}$.
- $A = (A(i,j))$ is an $n \times n$ matrix called the *state transition probability matrix*, with

$$A(i,j) \geq 0, \quad 1 \leq i, j \leq n, \quad \text{and} \quad \sum_{j=1}^{n} A(i,j) = 1,$$

  $i = 1, \ldots, n$.
- $B = (B(i,j))$ is an $n \times m$ matrix called the *state observation probability matrix* (also called *confusion matrix*), with

$$B(i,j) \geq 0, \quad 1 \leq i, j \leq n, \quad \text{and} \quad \sum_{j=1}^{m} B(i,j) = 1,$$

  $i = 1, \ldots, n$.

A matrix satisfying the above conditions is said to be *row stochastic*. Both A and B are row-stochastic.

We also need to state the conditions that make M a Markov model. To do this rigorously requires the notion of random variable and is a bit tricky (see the remark in the notes), so we will cheat as follows:

(a) Given any sequence of states $(q_1, \ldots, q_{t-2}, p, q)$, the conditional probability that q is the tth state given that the previous states were $q_1, \ldots, q_{t-2}, p$ is equal to the conditional probability that q is the tth state given that the previous state at time $t-1$ is p:

$$\mathsf{Pr}(q \mid q_1, \ldots, q_{t-2}, p) = \mathsf{Pr}(q \mid p).$$

This is the *Markov property*.

(b) Given any sequence of states $(q_1, \ldots, q_i, \ldots, q_t)$, and given any sequence of outputs $(O_1, \ldots, O_i, \ldots, O_t)$, the conditional probability that the output O_i is emitted depends only on the state q_i, and not any other states or any other observations:

$$\begin{aligned}\mathsf{Pr}(O_i \mid q_1, \ldots, q_i, \ldots, q_t, O_1, \ldots, O_i, \ldots, O_t) \\ = \mathsf{Pr}(O_i \mid q_i).\end{aligned}$$

This is the *output independence* condition.

Examples of HMMs are shown in Figure 4.1, Figure 4.2, and Figure 4.4 shown below.

Note that an ouput is emitted when visiting a state, not when making a transition, as in the case of a gsm.

So the analogy with the gsm model is only partial; it is meant as a motivation for HMMs.

If we ignore the output components $\mathbb{O}$ and B, then we have what is called a *Markov chain*.

There are three types of problems that can be solved using HMMs:

(1) **The decoding problem**: Given an HMM $M = (Q, \mathbb{O}, \pi, A, B)$, for any observed output sequence $\mathcal{O} = (O_1, O_2, \ldots, O_T)$ of length T, *find a most likely sequence of states* $\mathcal{S} = (q_1, q_2, \ldots, q_T)$ *that produces the output sequence* $\mathcal{O}$.

More precisely, with our notational convention that $\sigma(q_t) = i_t$ and $\omega(O_t) = \omega_t$, this means finding a sequence $\mathcal{S}$ such that the probability

$$\mathsf{Pr}(\mathcal{S}, \mathcal{O}) = \pi(i_1)B(i_1, \omega_1) \prod_{t=2}^{T} A(i_{t-1}, i_t)B(i_t, \omega_t)$$

is *maximal*.

This problem is solved effectively by the *Viterbi algorithm*.

(2) **The evaluation problem**, also called **the likelyhood problem**:
Given a finite collection $\{M_1, \ldots, M_L\}$ of HMM's with the same output alphabet $\mathbb{O}$, for any output sequence $\mathcal{O} = (O_1, O_2, \ldots, O_T)$ of length T, *find which model M_ℓ is most likely to have generated $\mathcal{O}$*.

More precisely, given any model M_k, we compute the probability $tprob_k$ that M_k could have produced $\mathcal{O}$ along any path.

Then we pick an HMM M_ℓ for which $tprob_\ell$ is maximal. We will return to this point after having described the Viterbi algoritm.

A variation of the Viterbi algorithm called the *forward algorithm* effectively solves the evaluation problem.

(3) **The training problem**, also called **the learning problem**: Given a set $\{\mathcal{O}_1, \ldots, \mathcal{O}_r\}$ of output sequences on the same output alpabet $\mathbb{O}$, usually called a set of *training data*, given Q, *find the "best" π, A, and B for an HMM M that produces all the sequences in the training set*, in the sense that the HMM $M = (Q, \mathbb{O}, \pi, A, B)$ is the most likely to have produced the sequences in the training set.

The technique used here is called *expectation maximization*, or *EM*. It is an iterative method that starts with an initial triple π, A, B, and tries to impove it.

There is such an algorithm known as the *Baum-Welch* or *forward-backward algorithm*, but it is beyond the scope of this introduction.

Let us now describe the Viterbi algorithm in more details.

4.2 The Viterbi Algorithm and the Forward Algorithm

Given an HMM $M = (Q, \mathbb{O}, \pi, A, B)$, for any observed output sequence $\mathcal{O} = (O_1, O_2, \ldots, O_T)$ of length T, *we want to find a most likely sequence of states* $\mathcal{S} = (q_1, q_2, \ldots, q_T)$ *that produces the output sequence* $\mathcal{O}$.

Using the bijections $\sigma\colon Q \to \{1, \ldots, n\}$ and $\omega\colon \mathbb{O} \to \{1, \ldots, m\}$, we can work with sequences of indices, and recall that we denote the index $\sigma(q_t)$ associated with the tth state q_t in the sequence $\mathcal{S}$ by i_t, and the index $\omega(O_t)$ associated with the tth output O_t in the sequence $\mathcal{O}$ by ω_t.

Then we need to find a sequence $\mathcal{S}$ such that the probability

$$\mathsf{Pr}(\mathcal{S}, \mathcal{O}) = \pi(i_1)B(i_1, \omega_1) \prod_{t=2}^{T} A(i_{t-1}, i_t)B(i_t, \omega_t)$$

is maximal.

In general, there are n^T sequences of length T.

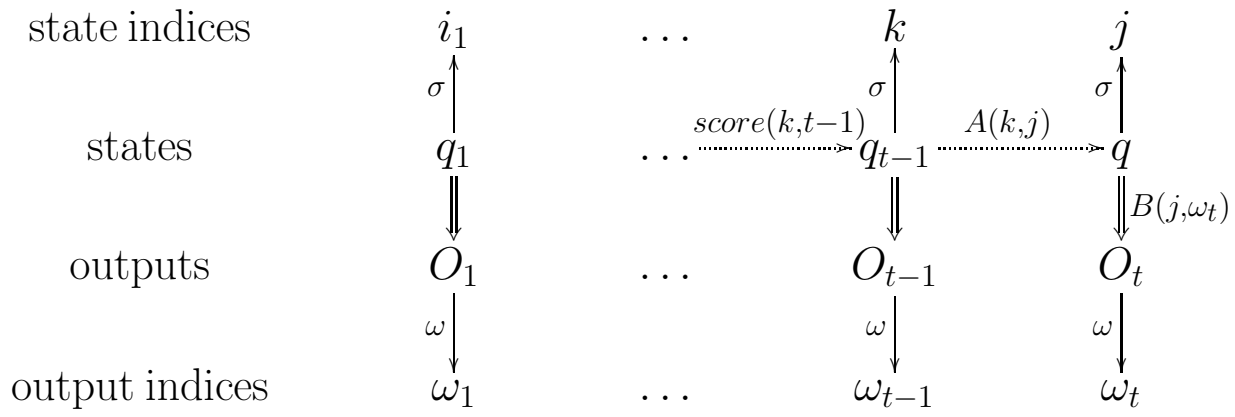
This problem can be solved efficiently by a method based on *dynamic programming*.

For any t, $1 \leq t \leq T$, for any state $q \in Q$, if $\sigma(q) = j$, then *we compute score(j, t), which is the largest probability that a sequence $(q_1, \ldots, q_{t-1}, q)$ of length t ending with q has produced the output sequence $(O_1, \ldots, O_{t-1}, O_t)$*.

The point is that if we know $score(k, t-1)$ for $k = 1, \ldots, n$ (with $t \geq 2$), then we can find $score(j, t)$ for $j = 1, \ldots, n$, because if we write $k = \sigma(q_{t-1})$ and $j = \sigma(q)$ (recall that $\omega_t = \omega(O_t)$), then the probability associated with the path $(q_1, \ldots, q_{t-1}, q)$ is

$$tscore(k, j) = score(k, t-1)A(k, j)B(j, \omega_t).$$

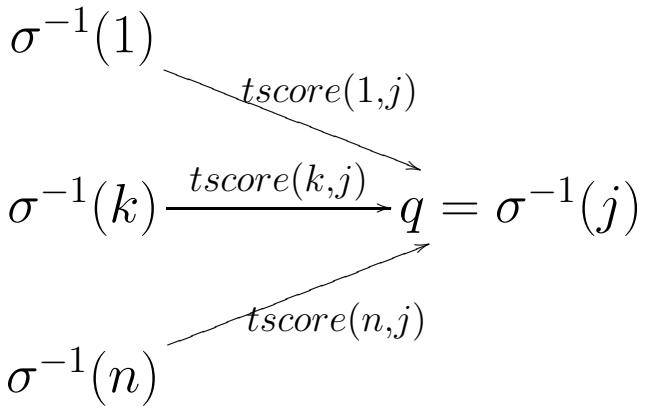
See the illustration below:



So to *maximize* this probability, we just have to find the *maximum of the probabilities* $tscore(k,j)$ *over all* k, that is, we must have

$$score(j,t) = \max_k tscore(k,j).$$

See the illustration below:



To get started, we set $score(j, 1) = \pi(j)B(j, \omega_1)$ for $j = 1, \ldots, n$.

The algorithm goes through a *forward phase* for $t = 1, \ldots, T$, during which it computes the probabilities $score(j, t)$ for $j = 1, \ldots, n$.

When $t = T$, we pick an index j such that $score(j, T)$ is maximal.

The machine learning community is fond of the notation

$$j = \arg\max_k score(k, T)$$

to express the above fact. Typically, the smallest index j corresponding to the largest value of $score(k, T)$ is returned.

This gives us the *last state* $q_T = \sigma^{-1}(j)$ in an optimal sequence that yields the output sequence $\mathcal{O}$.

The algorithm then goes through a *path retrieval phase*.

To to this, when we compute

$$score(j,t) = \max_k tscore(k,j),$$

we also record the index $k = \sigma(q_{t-1})$ of the state q_{t-1} in the best sequence $(q_1, \ldots, q_{t-1}, q_t)$ for which $tscore(k,j)$ is *maximal* (with $j = \sigma(q_t)$), as $pred(j,t) = k$.

The index k is often called the *backpointer* of j at time t.

This state may not be unique, we just pick one of them. Typically, the smallest index k corresponding to the largest value of $tscore(k,j)$ is returned.

Again, this can be expressed by

$$pred(j,t) = \arg\max_k tscore(k,j).$$

The predecessors $pred(j,t)$ are only defined for $t = 2, \ldots, T$, but we can let $pred(j,1) = 0$.

Observe that the path retrieval phase of the Viterbi algorithm is very similar to the phase of *Dijkstra's algorithm* for finding a shortest path that follows the $prev$ array.

The forward phase of the Viterbi algorithm is quite different from the Dijkstra's algorithm, and the Viterbi algorithm is actually simpler.

The Viterbi algorithm, invented by Andrew Viterbi in 1967, is shown below.

The input to the algorithm is $M = (Q, \mathbb{O}, \pi, A, B)$ and the sequence of indices $\omega(\mathcal{O}) = (\omega_1, \ldots, \omega_T)$ associated with the observed sequence $\mathcal{O} = (O_1, O_2, \ldots, O_T)$ of length T, with $\omega_t = \omega(O_t)$ for $t = 1, \ldots, T$.

The output is a sequence of states $(q_1, \ldots, q_T)$. This sequence is determined by the sequence of indices $(I_1, \ldots, I_T)$; namely, $q_t = \sigma^{-1}(I_t)$.

The Viterbi Algorithm

begin

for $j = 1$ **to** n **do**

$score(j, 1) = \pi(j)B(j, \omega_1)$

endfor;

(* forward phase to find the best (highest) scores *)

for $t = 2$ **to** T **do**

for $j = 1$ **to** n **do**

for $k = 1$ **to** n **do**

$tscore(k) = score(k, t-1)A(k, j)B(j, \omega_t)$

endfor;

$score(j, t) = \max_k tscore(k)$;

$pred(j, t) = \arg\max_k tscore(k)$

endfor

endfor;

(* second phase to retrieve the optimal path *)

$I_T = \arg\max_j score(j, T)$;

$q_T = \sigma^{-1}(I_T)$;

for $t = T$ **to** 2 **by** -1 **do**

$I_{t-1} = pred(I_t, t)$;

$q_{t-1} = \sigma^{-1}(I_{t-1})$

endfor

end

If we run the Viterbi algorithm on the output sequence (S, M, S, L) of Example 4.2, we find that the sequence (Cold, Cold, Cold, Hot) has the highest probability, 0.00282, among all sequences of length four.

One may have noticed that the numbers involved, being products of probabilities, become quite small.

Indeed, underflow may arise in dynamic programming. Fortunately, there is a simple way to avoid underflow by taking logarithms.

It immediately verified that the time complexity of the Viterbi algorithm is $O(n^2T)$.

Let us now to turn to the second problem, the *evaluation problem* (or *likelyhood problem*).

This time, given a finite collection $\{M_1, \ldots, M_L\}$ of HMM's with the same output alphabet $\mathbb{O}$, for any observed output sequence $\mathcal{O} = (O_1, O_2, \ldots, O_T)$ of length T, *find which model M_ℓ is most likely to have generated $\mathcal{O}$*.

More precisely, given any model M_k, we compute the probability $tprob_k$ that M_k could have produced $\mathcal{O}$ *along any path*.

Then we pick an HMM M_ℓ for which $tprob_\ell$ is maximal.

It is easy to adapt the Viterbi algorithm to compute $tprob_k$. This algorithm is called the *forward algorithm*.

Since we are not looking for an explicity path, *there is no need for the second phase*, and during the forward phase, going from $t-1$ to t, rather than finding the maximum of the scores $tscore(k)$ for $k = 1, \ldots, n$, we just set $score(j, t)$ to the *sum* over k of the temporary scores $tscore(k)$.

At the end, $tprob_k$ is the sum over j of the probabilities $score(j, T)$.

The input to the algorithm is $M = (Q, \mathbb{O}, \pi, A, B)$ and the sequence of indices $\omega(\mathcal{O}) = (\omega_1, \ldots, \omega_T)$ associated with the observed sequence $\mathcal{O} = (O_1, O_2, \ldots, O_T)$ of length T, with $\omega_t = \omega(O_t)$ for $t = 1, \ldots, T$.

The output is the probability *tprob*.

The Foward Algorithm

begin
  for $j = 1$ **to** n **do**
    $score(j, 1) = \pi(j)B(j, \omega_1)$
  endfor;
  for $t = 2$ **to** T **do**
    for $j = 1$ **to** n **do**
      for $k = 1$ **to** n **do**
        $tscore(k) = score(k, t-1)A(k, j)B(j, \omega_t)$
      endfor;
      $score(j, t) = \sum_k tscore(k)$
    endfor
  endfor;
  $tprob = \sum_j score(j, T)$
end

We can now run the above algorithm on $M_1, \ldots, M_L$ to compute $tprob_1, \ldots, tprob_L$, and we pick the model M_ℓ for which $tprob_\ell$ is *maximum*.

As for the Viterbi algorithm, the time complexity of the forward algorithm is $O(n^2T)$.

Underflow is also a problem with the forward algorithm.

At first glance it looks like taking logarithms does not help because there is no simple expression for $\log(x_1 + \cdots + x_n)$ in terms of the $\log x_i$.

Fortunately, we can use the *log-sum exp trick*; see the notes.

Example 4.3. To illustrate the forward algorithm, assume that our observant student also recorded the drinking behavior of a professor at Harvard, and that he came up with the HHM shown in Figure 4.4.

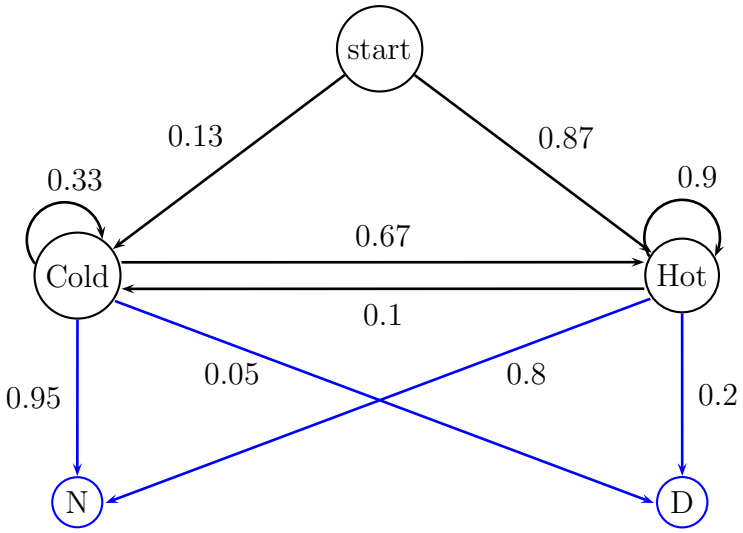


Figure 4.4: Example of an HMM modeling the "drinking behavior" of a professor at Harvard.

However, the student can't remember whether he observed the sequence NNND at Penn or at Harvard.

So he runs the forward algorithm on both HMM's to find the most likely model. Do it!