Machine Learning
the Fundamentals

William J. Deuschle

Harvard College
Cambridge, Massachusetts
April, 2019

Contents

Chapter 10

Hidden Markov Models

Many of the techniques we've considered so far in this book have been motivated by the *types* of data we could expect to work with. For example, the supervised learning techniques (forms of regression, neural networks, support vector machines, etc.) were motivated by the fact that we had labelled training data. We ventured into clustering to group unlabelled data and discussed dimensionality reduction to handle overly high-dimensional data. In the previous chapter, we examined techniques for managing *incomplete* data with latent variable models. In this chapter we turn to a technique for handling data indexed by *time*.

10.1 Motivation

One major type of data we have not yet paid explicit attention to is *time series* data. Most of the information we record comes with some sort of a timestamp. For example, any time we take an action online, there is a high probability that the database storing that data also tracks it with a timestamp. Physical sensors in the real world always record timestamps because it would be very difficult to make sense of their information if it is not indexed by time. When we undergo medical exams, the results are recorded along with a timestamp. It's almost inconceivable at this point that we would record information without also keeping track of when that data was generated, or at the very least when we saved that data.

There are a variety of reasons to treat time as a special type of data. As we've already discussed, it's a near-universal feature because we so frequently record timestamp information. It's difficult to imagine a data set that couldn't come with timestamps attached to the individual data points. Time also encodes a lot of information that we take for granted about the physical and digital worlds. For example, if the sensors on a plane record the position of the plane at a specific point in time, we would expect the surrounding data points to be relatively similar, or at least move in a consistent direction. In a more general sense, we expect that time constrains other aspects of the data it is attached to in specific ways.

Because we know time posseses these unique properties, it follows that we should exploit them to create more expressive models. In this chapter, we will focus on one such model known as a **Hidden Markov Model** or **HMM**. At a high level, the goal of an HMM is to model the state of an entity over time, with the caveat that we never actually observe the state itself. Instead, we observe a data point $\mathbf{x}_t$ at each time step (often called an 'emission') that is some function of the true state $\mathbf{s}_t$. For example, we could model the position of a robot over time given a noisy estimation of the robot's current position at each time step. Furthermore, we have some belief about how one state $\mathbf{s}_t$ transitions to the next state $\mathbf{s}_{t+1}$. Graphically, an HMM looks like Figure

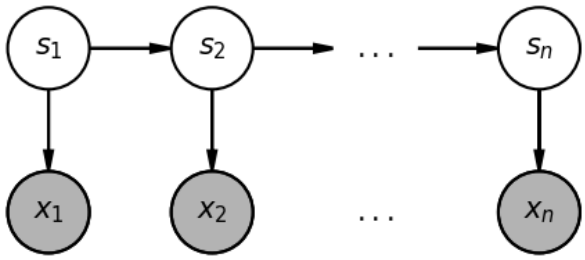

Figure 10.1: The directed graphical model for an HMM.

10.1, which encodes the relationships between emissions and hidden states.

We will probe HMMs in more detail over the course of the chapter, but for now let's consider their high-level properties.

ML Framework Cube: Hidden Markov Models

First, HMMs handle discrete states, and for the purpose of this text, we will only consider discrete-valued emissions as well. Second, since an HMM does not have access to the true states even at training time, it works on an unsupervised basis to approximate the hidden states. Finally, it's most common to probabilistically describe both the relationship between hidden states and observed emissions as well as the transitions between hidden states.

Domain	***Training***	***Probabilistic***
Discrete	Unsupervised	Yes

⋆ Models that treat continuous state variables are commonly referred to as dynamical systems.

10.2 Applications

Unsurprisingly, there are many applications for models like HMMs that explicitly account for time and unobserved states, especially those that relate to the physical world.

1. Robot position when movements are non-deterministic and sensor readings are noisy.

2. Speech recognition.

3. Analyzing sequences that occur in the natural world, such as DNA.

10.3 HMM Data, Model, and Parameterization

As explained above, HMMs model the state of an entity over time given some noisy observations, as shown in Figure 10.1.

10.3.1 HMM Data

A complete data point for an HMM consists of the sequence of one-hot encoded states $\mathbf{s}_1, ..., \mathbf{s}_n$ as well as the corresponding sequence of observed emissions $\mathbf{x}_1, ..., \mathbf{x}_n$. Each state corresponds to one of K possible options, meaning $\mathbf{s}_t \in \mathbb{R}^K$, and each emission corresponds to one of M possible options meaning $\mathbf{x}_t \in \mathbb{R}^M$. Note that we don't have access to the complete data points, but only the observed emissions. It's our goal to infer the hidden states.

⋆ In general the observed emissions don't have to be discrete, but for the sake of being explicit, we present the discrete interpretation here.

Note that a single *data point* consists of a sequence of n emissions $\mathbf{x}_1, ..., \mathbf{x}_n$. A data set is said to have N data points, meaning N sequences where each sequence is composed of n emissions. To summarize:

- A data set consists of N sequences.
- Each sequence is composed of n observed emissions $\mathbf{x}_1, ..., \mathbf{x}_n$.
- It's our goal to infer the hidden states $\mathbf{s}_1, ..., \mathbf{s}_n$ for each sequence.
- Each emission $\mathbf{x}_t$ takes on one of M possible options.
- Each hidden state $\mathbf{s}_t$ take on one of K possible options.

10.3.2 HMM Model Assumptions

One goal of an HMM is to optimize the joint distribution over hidden states and observed emissions given by:

$$p(\mathbf{s}_1, ..., \mathbf{s}_n, \mathbf{x}_1, ..., \mathbf{x}_n) = p(\mathbf{s}_1, ..., \mathbf{s}_n)p(\mathbf{x}_1, ..., \mathbf{x}_n|\mathbf{s}_1, ..., \mathbf{s}_n) \tag{10.1}$$

It's not immediately obvious how we should go about optimizing the model for this complex joint distribution. Fortunately, HMMs make this problem easier via the following assumptions:

1. State $\mathbf{s}_{t+1}$ depends only on the previous state $\mathbf{s}_t$:

   $$p(\mathbf{s}_{t+1}|\mathbf{s}_1, ..., \mathbf{s}_t, \mathbf{x}_1, ..., \mathbf{x}_t) = p(\mathbf{s}_{t+1}|\mathbf{s}_t)$$

   This is the *Markov Property*, and it means that given information at the previous time step, we can ignore all earlier time steps.

2. At each time step t, the observed emission $\mathbf{x}_t$ depends only on the current state $\mathbf{s}_t$.

   $$p(\mathbf{x}_t|\mathbf{s}_1, ..., \mathbf{s}_t, \mathbf{x}_1, ..., \mathbf{x}_{t-1}) = p(\mathbf{x}_t|\mathbf{s}_t)$$

⋆ The Markovian assumption for transitions, as well as the fact that we don't observe the true states, gives rise to the *Hidden Markov Model* name.

Notice that these two assumptions allow us to factorize the large joint distribution given by Equation 10.1 as follows:

$$p(\mathbf{s}_1, ..., \mathbf{s}_n)p(\mathbf{x}_1, ..., \mathbf{x}_n|\mathbf{s}_1, ..., \mathbf{s}_n) = p(\mathbf{s}_1)\prod_{t=1}^{n-1} p(\mathbf{s}_{t+1}|\mathbf{s}_t)\prod_{t=1}^{n} p(\mathbf{x}_t|\mathbf{s}_t) \tag{10.2}$$

This factorization will prove important for making HMM training and inference tractable.

10.3.3 HMM Parameterization

Now that we understand the form of the data as well as the modeling assumptions made by an HMM, we can specify the model parameterization explicitly. Referencing the factorized joint distribution from Equation 10.2, it's clear that we will need three distinct sets of parameters.

1. One set of parameters is for the prior over our initial hidden state $p(\mathbf{s}_1)$. It will be denoted $\boldsymbol{\theta} \in \mathbb{R}^K$, such that:

$$p(\mathbf{s}_1 = k) = \theta_k$$

2. Another set of parameters is for the transition probabilities between states $p(\mathbf{s}_{t+1}|\mathbf{s}_t)$. It will be denoted $\mathbf{T} \in \mathbb{R}^{K\times K}$, such that:

$$p(\mathbf{s}_{t+1} = j|\mathbf{s}_t = i) = T_{i,j}$$

   where $T_{i,j}$ is the probability of transitioning from state i to state j.

3. Finally, we have a set of parameters for the conditional probability of the observed emission given the hidden state, meaning $p(\mathbf{x}_t|\mathbf{s}_t)$. It will be denoted $\boldsymbol{\pi} \in \mathbb{R}^{K\times M}$, such that:

$$p(\mathbf{x}_t = m|\mathbf{s} = k) = \pi_{k,m}$$

   A natural interpretation of the parameter matrix $\boldsymbol{\pi}$ is that each possible state (of which there are K options) has an M-dimensional vector describing the probability of an emission given that state.

In sum, we have three sets of parameters $\boldsymbol{\theta} \in \mathbb{R}^K$, $\mathbf{T} \in \mathbb{R}^{K\times K}$, and $\boldsymbol{\pi} \in \mathbb{R}^{K\times M}$ that we need to learn from our data set. Then, using this trained model, we will be able to perform several types of inference over our hidden states, which are detailed down below.

10.4 EM for HMMs and the Forward-Backward Algorithm

Recall the motivation for the Expectation-Maximization algorithm from the previous chapter: we had parameters we wished to optimize, but the presence of unobserved variables made direct optimization of those parameters intractable. We're faced with a similar problem in the context of HMMs.

Given a data set of observed emissions $\{\mathbf{x}^i\}_{i=1}^N$ where each data point $\mathbf{x}^i$ represents the sequence $(\mathbf{x}_1^i, ..., \mathbf{x}_n^i)$, our goal is to estimate the parameters $\boldsymbol{\theta}, \mathbf{T}, \boldsymbol{\pi}$ described in the previous section. If we knew the hidden states, it would be possible for us to write the joint probability $p(\mathbf{s}^i, \mathbf{x}^i)$ directly, and maximizing our parameters would be no issue. However, the true states are latent variables, and

thus we would need to sum over their possible values, which is what makes the direct maximization of our parameters intractable.

Instead, we will use the Expectation-Maximization algorithm we developed in the previous chapter. This amounts to computing the expectation over our hidden states in the E-step, and then based on those fixed expectations, we can update our parameters via a maximization of the expected complete-data in the M-step. As usual, we perform these operations iteratively until convergence.

So far, there is no clear departure from the same exact EM algorithm we saw in the last chapter. However, before we go about detailing the E and M steps, we need to consider something known as the **Forward-Backward Algorithm**.

10.4.1 Forward-Backward Algorithm

Let's consider what would actually be required of us to perform the E-step for the hidden states of an HMM. To compute the expected value of state $\mathbf{s}_t$ using the full joint distribution given by:

$$p(\mathbf{s}_1, ..., \mathbf{s}_n, \mathbf{x}_1, ..., \mathbf{x}_n) = p(\mathbf{s}_1) \prod_{t=1}^{n-1} p(\mathbf{s}_{t+1}|\mathbf{s}_t) \prod_{t=1}^{n} p(\mathbf{x}_t|\mathbf{s}_t)$$

would require marginalizing over all the unobserved states other than $\mathbf{s}_t$ as follows:

$$\mathbb{E}[\mathbf{s}_t|\mathbf{s}_1, ..., \mathbf{s}_{t-1}, \mathbf{s}_{t+1}, ..., \mathbf{s}_n, \mathbf{x}_1, ..., \mathbf{x}_n] = p(\mathbf{s}_t|\mathbf{s}_1, ..., \mathbf{s}_{t-1}, \mathbf{s}_{t+1}, ..., \mathbf{s}_n, \mathbf{x}_1, ..., \mathbf{x}_n) =$$
$$\sum_{k^{(1)} \in K} ... \sum_{k^{(t-1)} \in K} \sum_{k^{(t+1)} \in K} ... \sum_{k^{(n)} \in K} p(\mathbf{s}_t|\mathbf{s}_1 = k^{(1)}, ..., \mathbf{s}_{t-1} = k^{(t-1)}, \mathbf{s}_{t+1} = k^{(t+1)}, ..., \mathbf{s}_n = k^{(n)}, \mathbf{x}_1, ..., \mathbf{x}_n)$$

If we were to do this naively by marginalizing over all states at each time step, it would be a very expensive and wasteful computation. For example, computing the expected value of the state at time step 2 would require summing over all possible states for $\mathbf{s}_1, \mathbf{s}_3, ..., \mathbf{s}_n$. Then, to get the expected value of the state at time step 3, we would need to sum over all possible states for $\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_4, ..., \mathbf{s}_n$. Notice that this approach duplicates a lot of work. Rather than performing these summations over and over again, we can instead memoize (or reuse) these summations using the Forward-Backward algorithm.

The Forward-Backward algorithm is an example of a *message-passing* scheme, which means we compute information once and then pass it around our model in the form of compact, reusable messages. The goal is to avoid duplicating work, instead using the messages directly. If you've encounted dynamic programming before, the Forward-Backward Algorithm is an example. Ultimately, the purpose of the Foward-Backward algorithm is to make the expectation step over our HMM more efficient.

This algorithm passes messages, unsurprisingly, forwards and backwards through 'time', meaning up and down the chain shown in the graphical model representation given by Figure 10.1. The forward messages are defined at each state as $\alpha_t(\mathbf{s}_t)$, while the backward messages are defined at each state as $\beta_t(\mathbf{s}_t)$. Let's define these α and β values explicitly.

The α_t's represent the joint probability of all our observed emissions from time $1, ..., t$ as well as the state exactly at time t:

$$\alpha_t(\mathbf{s}_t) = p(\mathbf{x}_1, ..., \mathbf{x}_t, \mathbf{s}_t) \tag{10.3}$$

Graphically, this means that the α_t's are capturing the portion of the DGM shown in Figure 10.2.

Note that Equation 10.3 is an unfactorized joint probability over observed emissions and a hidden state. We can factorize this joint probability using what we know about the conditional

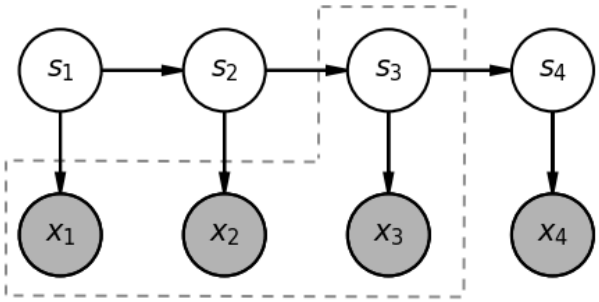

Figure 10.2: α_t's capture the joint probability for the boxed portion of this DGM.

independence properties of HMMs as follows:

$$\begin{aligned}\alpha_t(\mathbf{s}_t) &= p(\mathbf{x}_1, ..., \mathbf{x}_t, \mathbf{s}_t) && (10.4)\\ &= p(\mathbf{x}_t|\mathbf{s}_t) \sum_{\mathbf{s}_{t-1}} p(\mathbf{s}_t|\mathbf{s}_{t-1}) p(\mathbf{x}_1, ..., \mathbf{x}_{t-1}, \mathbf{s}_{t-1}) && (10.5)\\ &= p(\mathbf{x}_t|\mathbf{s}_t) \sum_{\mathbf{s}_{t-1}} p(\mathbf{s}_t|\mathbf{s}_{t-1}) \alpha_{t-1}(\mathbf{s}_{t-1}) && (10.6)\end{aligned}$$

Pay careful attention to the last line of Equation 10.4. Notice that our expression for $\alpha_t(\mathbf{s}_t)$ actually includes the expression for $\alpha_{t-1}(\mathbf{s}_{t-1})$, which is the α from the previous time step. This is significant because it means we can define our messages *recursively*. After we've computed the α's at one time step, instead of having to recompute those values to get the alphas at the next time step, we can simply *pass* them forwards along the chain. In other words, we compute the α at state $\mathbf{s}_1$, then pass that message along to compute the α at state $\mathbf{s}_2$, on and on until we've reached the end of the chain and have all the α's in hand.

⋆ These α values will be useful when performing the expectation step during training, as well as for general inference once we've finished training our HMM.

At this point, we've now handled the forward messages, which send information from the beginning to the end of the chain. We also need to send information from the end of the chain back to the beginning, which constitutes the backwards portion of the algorithm. This is where we will compute our β values.

The β_t's represent the joint probability over all the observed emissions from time $t+1, ..., n$ conditioned on the state at time t:

$$\beta_t(\mathbf{s}_t) = p(\mathbf{x}_{t+1}, ..., \mathbf{x}_n|\mathbf{s}_t) \tag{10.7}$$

Graphically, this means that the β_t's are capturing the portion of the DGM shown in Figure 10.3.

We can factorize Equation 10.7 in a similar way to how we factorized the distribution described

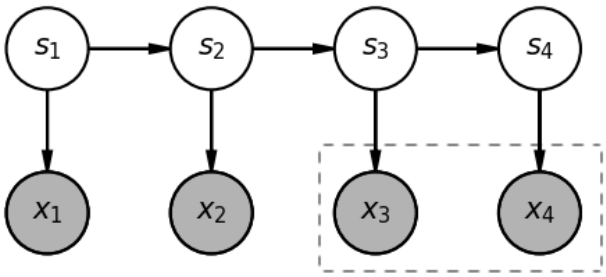

Figure 10.3: β_t's capture the joint probability for the boxed portion of this DGM.

by the α's:

$$\begin{aligned}
\beta_t(\mathbf{s}_t) &= p(\mathbf{x}_{t+1}, ..., \mathbf{x}_n | \mathbf{s}_t) && (10.8)\\
&= \sum_{\mathbf{s}_{t+1}} p(\mathbf{s}_{t+1}|\mathbf{s}_t) p(\mathbf{x}_{t+1}, ..., \mathbf{x}_n | \mathbf{s}_{t+1}) && (10.9)\\
&= \sum_{\mathbf{s}_{t+1}} p(\mathbf{s}_{t+1}|\mathbf{s}_t) p(\mathbf{x}_{t+1}|\mathbf{s}_{t+1}) p(\mathbf{x}_{t+2}, ..., \mathbf{x}_n | \mathbf{s}_{t+1}) && (10.10)\\
&= \sum_{\mathbf{s}_{t+1}} p(\mathbf{s}_{t+1}|\mathbf{s}_t) p(\mathbf{x}_{t+1}|\mathbf{s}_{t+1}) \beta_{t+1}(\mathbf{s}_{t+1}) && (10.11)
\end{aligned}$$

Again, as we saw with our calculation of the α's, we have our β's defined recursively. Once again, this means that we can propagate messages efficiently. In this case, we start at the end of the chain, and compute our β's for each state by passing messages back toward the front.

To summarize, the Forward-Backward algorithm is an optimization that will boost efficiency for the EM algorithm as well as for inference over a trained HMM. We calculate the α and β values as follows:

$$\alpha_t(\mathbf{s}_t) = \begin{cases} p(\mathbf{x}_t|\mathbf{s}_t) \sum_{\mathbf{s}_{t-1}} p(\mathbf{s}_t|\mathbf{s}_{t-1}) \alpha_{t-1}(\mathbf{s}_{t-1}) & 1 < t \leq n \\ p(\mathbf{x}_1|\mathbf{s}_1) p(\mathbf{s}_1) & \text{otherwise} \end{cases}$$

$$\beta_t(\mathbf{s}_t) = \begin{cases} \sum_{\mathbf{s}_{t+1}} p(\mathbf{s}_{t+1}|\mathbf{s}_t) p(\mathbf{x}_{t+1}|\mathbf{s}_{t+1}) \beta_{t+1}(\mathbf{s}_{t+1}) & 1 \leq t < n \\ 1 & \text{otherwise} \end{cases}$$

⋆ Notice that the base case for the β's is 1. This is just a quirk of our indexing, and it ensures we have valid messages when we pass messages back from the final state $\mathbf{s}_n$.

10.4.2 Using α's and β's for Training and Inference

Now that we know how to compute these α and β values, we need to consider how we actually use them for training and inference within our HMM setup. Indeed, it's not immediately clear how the distributions defined by the α's and β's are useful.

Let's begin by assuming we randomly initialized our model parameters $\boldsymbol{\theta}, \mathbf{T}, \boldsymbol{\pi}$, and then used these to compute all the α and β values by passing messages up and down the chain (using the

recurrences defined at the end of the last section). We now have α and β values defined at every time step in our HMM. Consider the product of the α and β value at a specific time t:

$$\alpha_t(\mathbf{s}_t)\beta_t(\mathbf{s}_t) = p(\mathbf{x}_1, ..., \mathbf{x}_t, \mathbf{s}_t)p(\mathbf{x}_{t+1}, ..., \mathbf{x}_n|\mathbf{s}_t) = p(\mathbf{x}_1, ..., \mathbf{x}_n, \mathbf{s}_t)$$

which is just the joint distribution over all our observed emissions and the state at time t. Using this as a building block, there are many distributions over the variables in our HMM that we can evaluate. For example, we might like to evaluate the joint distribution over our observed emissions.

Joint Distribution Over Emissions

$$p(\mathbf{x}_1, ..., \mathbf{x}_n) = \sum_{\mathbf{s}_t \in K} \alpha_t(\mathbf{s}_t)\beta_t(\mathbf{s}_t) \tag{10.12}$$

where we can sum over the possible state values at any time step $1, ..., n$.

Another useful distribution is that of the state at time t given all the observed emissions. This has the name *smoothing*, which in some contexts means updating our beliefs given observed evidence.

Smoothing

$$p(\mathbf{s}_t|\mathbf{x}_1, ..., \mathbf{x}_n) \propto p(\mathbf{x}_1, ..., \mathbf{x}_t, \mathbf{s}_t) = \alpha_t(\mathbf{s}_t)\beta_t(\mathbf{s}_t) \tag{10.13}$$

Notice that the proportionality of the smoothing operation means we can normalize to recover probabilities by taking the softmax over the possible values for the state $\mathbf{s}_t$.

Another common task is predicting the value of the next emission given the previous emissions.

Prediction

$$p(\mathbf{x}_{n+1}|\mathbf{x}_1, ..., \mathbf{x}_n) \tag{10.14}$$

To compute this we can sum over the final state $\mathbf{s}_n$ as well as the next state $\mathbf{s}_{n+1}$ as follows:

$$p(\mathbf{x}_{n+1}|\mathbf{x}_1, ..., \mathbf{x}_n) \propto \sum_{\mathbf{s}_n} \sum_{\mathbf{s}_{n+1}} p(\mathbf{x}_1, ..., \mathbf{x}_n, \mathbf{s}_n)p(\mathbf{s}_{n+1}|\mathbf{s}_n)p(\mathbf{x}_{n+1}|\mathbf{s}_{n+1}) \tag{10.15}$$

$$\propto \sum_{\mathbf{s}_n} \sum_{\mathbf{s}_{n+1}} \alpha_n(\mathbf{s}_n)p(\mathbf{s}_{n+1}|\mathbf{s}_n)p(\mathbf{x}_{n+1}|\mathbf{s}_{n+1}) \tag{10.16}$$

Again, we recover a proportional result from this operation. We can use the softmax over the emissions states to normalize.

Finally, we may wish to approximate the transition probabilities between states $\mathbf{s}_t$ and $\mathbf{s}_{t+1}$ given all the observed evidence.

Transition

$$p(\mathbf{s}_t, \mathbf{s}_{t+1}|\mathbf{x}_1, ..., \mathbf{x}_n) \propto p(\mathbf{x}_1, ..., \mathbf{x}_t, \mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t)p(\mathbf{x}_{t+1}|\mathbf{s}_{t+1})p(\mathbf{x}_{t+2}, ..., \mathbf{x}_n|\mathbf{s}_{t+1}) \tag{10.17}$$

$$\propto \alpha_t(\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t)p(\mathbf{x}_{t+1}|\mathbf{s}_{t+1})\beta_{t+1}(\mathbf{s}_{t+1}) \tag{10.18}$$

$$\tag{10.19}$$

where we again have proportionality results that can be normalized if necessary.

Now that we understand how exactly α and β values can be used to evaluate different distributions of interest for an HMM, we can finally turn to using EM to optimize the parameters of our model.

10.4.3 E-Step

Recall that the goal of the expectation step is to compute the expected values of the hidden variables in our model given a fixed set of parameters. Our parameters are $\boldsymbol{\theta}, \mathbf{T}, \boldsymbol{\pi}$. Note that the α and β values described in the previous section are exact functions of these parameters, so at the start of the E-step, we should compute them using the efficient Forward-Backward algorithm. It will become clear why we need the α and β values as we explain the E-step.

Under the HMM setup, our hidden variables are the states $\mathbf{s}_1, ..., \mathbf{s}_n$. Since we have N emission sequences $\mathbf{x}^i = (\mathbf{x}_1^i, ..., \mathbf{x}_n^i) \quad \forall i \in N$ in our data set, we will end up with N expected sequences over our hidden states:

$$\mathbf{q}^i = \mathbb{E}[(\mathbf{s}_1^i, ..., \mathbf{s}_n^i)|(\mathbf{x}_1^i, ..., \mathbf{x}_n^i)] \quad \forall i \in N$$

Where $\mathbf{q}^i \in \mathbb{R}^{n \times K}$, corresponding to the expectation of each state at each time step. Let's consider $q_{t,k}^i$ which is the expectation over a single state value k at a single point in time t. We can write this as:

$$\begin{aligned} q_{t,k}^i &= \mathbb{E}[s_{t,k}^i|(\mathbf{x}_1^i, ..., \mathbf{x}_n^i)] && (10.20) \\ &= p(\mathbf{s}_t^i = k|(\mathbf{x}_1^i, ..., \mathbf{x}_n^i)) && (10.21) \end{aligned}$$

This calculates the probability of a single state given all of our observed emissions for a single data point. This is exactly the smoothing operation described in the previous section! That is why we calculated our α and β values at the beginning of the E-step using our fixed parameters, because we can now easily use the smoothing operation given by Equation 10.13 to compute our $\mathbf{q}_t^i$ values.

Ordinarily, we'd be done with the E-step after computing the expectations of our hidden variables, but under the HMM setup, we need to take one more expectation before moving on to the M-step. Recall that we have transition probabilities between our hidden states given by the parameter matrix $\mathbf{T}$. Because these are transitions between latent variables, we also need to compute the expectation over the joint distribution between pairs of latent variables next to each other in time, which we will denote $\mathbf{Q}_{t,t+1}^i \in \mathbb{R}^{K \times K}$. Formally:

$$\mathbf{Q}_{t,t+1}^i = \mathbb{E}[\mathbf{s}_t^i, \mathbf{s}_{t+1}^i|(\mathbf{x}_1^i, ..., \mathbf{x}_n^i)] \tag{10.22}$$

Let's consider $Q_{t,t+1,k,l}^i$ which is the transition from state k at time step t to state l at time step $t+1$:

$$\begin{aligned} Q_{t,t+1,k,l}^i &= \mathbb{E}[\mathbf{s}_t^i = k, \mathbf{s}_{t+1}^i = l|(\mathbf{x}_1^i, ..., \mathbf{x}_n^i)] && (10.23) \\ &= p(\mathbf{s}_t^i = k, \mathbf{s}_{t+1}^i = l|(\mathbf{x}_1^i, ..., \mathbf{x}_n^i)) && (10.24) \\ & && (10.25) \end{aligned}$$

This is calculating the probability of a specific latent variable transition at a specific time step, which is exactly the transition operation described in the previous section. We can directly use our α and β values in the transition operation given by Equation 10.17.

With our $\mathbf{q}^i$ and $\mathbf{Q}^i$ values as expectations over the hidden states and hidden joint states respectively (with the current setting of our parameters fixed), we are ready to move on to the maximization step.

10.4.4 M-Step

The M-step mirrors what we saw in the previous chapter. In the E-step we calculated expectations over our hidden variables, which allows us to set up our expected complete-data log likelihood for a single data point:

$$\begin{aligned}\mathbb{E}_{\mathbf{q},\mathbf{Q}}[\log p(\mathbf{x}^i, \mathbf{s}^i; \boldsymbol{\theta}, \mathbf{T}, \boldsymbol{\pi})] &= \mathbb{E}_{\mathbf{q},\mathbf{Q}}\Big[\log p(\mathbf{x}^i|\mathbf{s}^i; \boldsymbol{\pi}) + \log p(\mathbf{s}^i; \boldsymbol{\theta}, \mathbf{T})\Big] \\ &= \sum_{t=1}^{n}\sum_{k=1}^{K} q_{t,k}^i \sum_{m=1}^{M} \log \pi_{k,x_{t,m}^i} + \sum_{k=1}^{K} q_k^i \log \theta_k + \sum_{t=1}^{n-1}\sum_{k=1}^{K}\sum_{l=1}^{K} Q_{t,t+1,k,l}^i \log T_{k,l}\end{aligned}$$

Applying the appropriate Lagrange multipliers and maximizing with respect to each of the parameters of interest, we recover the following update equations:

$$\theta_k = \frac{\sum_{i=1}^{N} q_{1,k}^i}{N} \tag{10.26}$$

$$T_{k,l} = \frac{\sum_{i=1}^{N}\sum_{t=1}^{n-1} Q_{t,t+1,k,l}^i}{\sum_{i=1}^{N}\sum_{t=1}^{n-1} q_{t,k}^i} \tag{10.27}$$

$$\pi_{k,m} = \frac{\sum_{i=1}^{N}\sum_{t=1}^{n} q_{t,k}^i x_{t,m}^i}{\sum_{i=1}^{N}\sum_{t=1}^{n} q_{t,k}^i} \tag{10.28}$$

After updating our parameter matrices $\boldsymbol{\theta}, \mathbf{T}, \boldsymbol{\pi}$, we switch back to the E-step, continuing in this way until convergence of our parameters.

10.4.5 From HMM Training to Inference

In the preceding sections, we described the process for training an HMM. In doing so, we've actually built up all the machinery needed to perform inference on our trained HMM for free. Specifically, we've described four types of queries we can execute over a trained HMM:

- Joint Estimation
- Prediction
- Smoothing
- Transition

We analyzed these already because we needed them to train the HMM, but once we've completed training, we can use our optimized parameters along with the α and β values that we compute with the Forward-Backward algorithm to query our HMM.

10.5 Conclusion

The Hidden Markov Model is a type of latent variable model motivated by the combination of time series and discrete state space data. We relied on the Expectation-Maximization algorithm described in the previous chapter, and developed the efficient Forward-Backward algorithm to make both training and inference feasible for HMMs. Many of the ideas developed in this chapter will offer good intuition for dynamical systems and other time series models.