Lecture 5 — April 14

Lecturer: Lester Mackey *Scribe: Brian Do and Robin Jia*

5.1 Discrete Hidden Markov Models

5.1.1 Recap

In the last lecture, we introduced discrete HMMs as a way of modeling a sequence of datapoints $x = (x_0, x_1, \ldots, x_T)$ as emissions from dependent hidden states $z = (z_0, z_1, \ldots, z_T)$ with each $z_t \in \{1, \ldots, k\}$. More precisely, we developed the following generative model for HMMs:

1. Sample the first state: $z_0 \sim \text{Mult}(\pi, 1)$.
2. Sample each new state from a transition distribution determined by the prior state: $z_t \mid z_{t-1} = j \overset{ind}{\sim} \text{Mult}(a_j, 1)$.
3. Sample datapoints $x_t \mid z_t \overset{ind}{\sim} p(x_t|z_t, \eta)$ from a state-dependent emission distribution.

Recall that $\pi \in \mathbb{R}^k$ is the initial state probability vector, each $a_j \in \mathbb{R}^k$ is the transition probability vector from state j to all other states, and η represents parameters for each conditional distribution given z_t.

Last time, we assumed that all model parameters were known $\theta = (\pi, A, \eta)$ and developed recursive schemes for efficient probabilistic inference in this model. Specifically, we developed α and β recursions to compute the single state conditional probabilities

$$p(z_t|x) = \frac{p(x_{0:t}, z_t)p(x_{t+1:T}|z_t)}{p(x)} \triangleq \frac{\alpha(z_t)\beta(z_t)}{p(x)}$$

for all timepoints t in $\text{O}(k^2T)$ total time. While this permits us to make inferences about hidden states in isolation, it is not sufficient to make joint inferences about several hidden states. For example, since the hidden states are dependent conditioned on x, typically $p(z_t, z_{t-1}|x) \neq p(z_t|x)p(z_{t-1}|x)$. In this lecture, we will learn to leverage our α-β recursions to compute these co-occurrence probabilities and other inferential quantities of interest.

5.1.2 Calculating co-occurrence probabilities

Let us try to calculate the co-occurrence probability $p(z_t, z_{t+1}|x)$ using Bayes' rule and knowledge of the HMM conditional independence structure:

$$\begin{aligned} p(z_t, z_{t+1}|x) &= \frac{p(x|z_t, z_{t+1})p(z_{t+1}, z_t)}{p(x)} = \frac{p(x|z_t, z_{t+1})p(z_{t+1}|z_t)p(z_t)}{p(x)} \\ &= \frac{p(x_{0:t}|z_t)p(x_{t+1}|z_{t+1})p(x_{t+2:T}|z_{t+1})a_{z_t,z_{t+1}}p(z_t)}{p(x)}. \end{aligned}$$

In the final equation, we have broken $p(x|z_t, z_{t+1})$ into $p(x_{0:t}|z_t)$ (the probability of the past and present observations given the present state), $p(x_{t+1}|z_{t+1})$ (the probability of the next observation given the next state), and $p(x_{t+2:T}|z_{t+1})$ (the probability of all future observations given the next state). We notice then that $p(x_{0:t}|z_t)p(z_t) = p(x_{0:t}, z_t) = \alpha(z_t)$ and $p(x_{t+2:T}|z_{t+1}) = \beta(z_{t+1})$. Therefore, we can compute the co-occurrence probabilities

$$p(z_t, z_{t+1}|x) = \frac{\alpha(z_t)\beta(z_{t+1})p(x_{t+1}|z_{t+1})a_{z_t,z_{t+1}}}{p(x)}$$

using only the results of our α-β recursion and the known transition and emission probabilities. This follows as, for any t, $p(x) = \sum_{z_t} \alpha(z_t)\beta(z_t)$. A derivation of the conditional distribution of any subsequence of z given all observations x proceeds similarly.

5.1.3 Soft inference for the full hidden sequence z

Suppose that our object of inference was instead the entire sequence of hidden states. Interestingly, calculation of $p(z|x)$ is even more straightforward. Observe that, by Bayes' rule,

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} = \frac{\pi(z_0)\Pi_{t=0}^{T-1} a_{z_t,z_{t+1}} \Pi_t p(x_t|z_t)}{p(x)}.$$

Since $p(x)$ is a function of α and β, this expression depends only on known quantities.

5.1.4 Hard inference for z via the Viterbi algorithm

Thus far, we have discussed soft inference, computing the conditional probabilities of hidden states, but often we are interested in hard assignments that associate each datapoint with a single state. In particular, as in the mixture modeling setting, we would like to find the hidden sequence that is most likely given the observations x. However, unlike in the mixture model setting where hidden states were independent, it is not the case that the most probable sequence of states $\text{argmax}_z\, p(z|x)$ is equal to the sequence of most probable individual states $\text{argmax}_{z_t}\, p(z_t|x)$. The distinction can be illustrated in the handwriting recognition setting: an ambiguous set of characters considered in isolation may seem to most likely correspond to the word "the" but in the context of a surrounding sentence, "I put on a ___ today," be more sensibly decoded as "tie." Thus, we must account for the dependence.

Since we know how compute $p(z|x)$, we could use brute force to consider all k^T possible sequences z, but that quickly becomes intractable. Instead, we will compute $\max_z p(z|x)$ and its maximizer in an efficient recursive manner analogous to our α recursion. Note that since $p(z|x) = \frac{p(z,x)}{p(x)}$ and $p(x)$ does not depend on the specific sequence z, the maximizer of the conditional probabillity is the same as the maximizer of the joint probability $p(z, x)$ (over z), and so we will focus on finding $\max_z p(z, x)$ and its maximizer using a recursive procedure called the **Viterbi algorithm**. Significantly, this algorithm is *identical* to the forward pass α-recursion except that we will replace summation over states with maximization over states.

The Viterbi algorithm

We begin by defining our targets for recursion

$$v(z_t) = \max_{z_{0:t-1}} p(z_{0:t}, x_{0:t}) = \max_{z_{0:t-1}} p(z_{0:t-1}, x_{0:t}, z_t).$$

Note the similarity to $\alpha(z_t)$. Instead of summing over past states that may have led to present hidden state z_t and observation x_t, we are maximizing over them. Note moreover that

$$\max_{z_T} v(z_T) = \max_{z_T} \max_{z_{0:T-1}} p(z_{0:T-1}, x_{0:T}, z_T) = \max_{z} p(z, x),$$

so if we can compute $v(z_T)$ efficiently, we can also compute our target maximum probability.

The Viterbi algorithm accomplishes this computation via a recursive forward pass through time:

$$\begin{aligned}
v(z_0) &= p(x_0, z_0) = p(z_0)p(x_0|z_0) \quad \text{(Base case)} \\
v(z_{t+1}) &= \max_{z_{0:t}} p(z_{0:t}, x_{0:t+1}, z_{t+1}) \\
&= \max_{z_t} (\max_{z_{0:t-1}} p(z_{0:t-1}, x_{0:t}, x_t)p(x_{t+1}|z_{t+1})p(z_{t+1}|z_t)) \\
&= \max_{z_t} v(z_t)a_{z_t,z_{t+1}}p(x_{t+1}, z_{t+1}) \quad \text{(Recursion for all } t \geq 0) \\
\max_z p(z, x) &= \max_{z_T} v(z_T) \quad \text{(Final step)} \quad .
\end{aligned}$$

The equation for $v(z_0)$ is just equal to the joint probability of the current state and current observation occurring together; there are no prior states over which to maximize. For the derivation of $v(z_{t+1})$, we can break up the max over all past states into a max over the proximal state, $v(z_t)$, and a max over more distant past states. These distant past states ($z_{0:t-1}$) are accounted for in $v(z_t)$, which we calculated in the prior recursive step. Altogether, these establish the base case and the recursive step. As with the α-recursion, calculating $v(z_t)$ for all t takes $O(k^2T)$ work.

To recover the maximizing sequence $\text{argmax}_z\, p(z, x)$ as well, it suffices to store pointers to maximizing states in each recursive step of the Viterbi algorithm. That is, the recursive computation of $v_{z_{t+1}}$ involves only a maximization over z_t, so one can keep track of the maximizer $m(z_{t+1}) = z_t^*$. Once the Viterbi computation is complete, one can follow these pointers backwards from z_T^* to obtain the entire sequence that maximizes $p(z, x)$ and hence $p(z|x)$.

5.1.5 EM for Hidden Markov Models

Our discussion of HMMs so far has assumed that the parameters $\theta = (\pi, A, \eta)$ are known, but, typically, we do not know the model parameters in advance. As usual (and as is most often done in practice), we will turn to the EM to learn model parameters that approximately maximize the likelihood of our observations.[1] Along the way, we will see that many of the inference routines we have derived will be useful for implementing the EM algorithm.

[1] In the HMM setting, the EM algorithm is also called the **Baum-Welch algorithm**.

Note that we defined our HMMs in terms of generic emission distributions, and the exact M-step updates will depend on the emission distributions chosen. To provide a concrete example of a complete EM algorithm, we will adopt a Gaussian emission model,

$$p(x_t \mid z_t = j; \eta) = \phi(x_t; \mu_j, \Sigma_j).$$

Complete log likelihood

As usual, we begin our EM derivation by writing down the complete log likelihood. Let $\theta = (\pi, A, \eta)$ denote all the parameters of the model. We have

$$\begin{aligned}\ell_c(\theta) = \log(p(x, z; \theta)) &= \log \pi_{z_0} + \sum_{t=0}^{T-1} \log a_{z_t z_{t+1}} + \sum_{t=0}^{T} \log \phi(x_t; \mu_{z_t} \Sigma_{z_t}) \\ &= \sum_{j=1}^{k} \mathbb{I}[z_0 = j] \log \pi_j + \sum_{t=0}^{T-1} \sum_{j=1}^{k} \sum_{l=1}^{k} \mathbb{I}[z_t = j, z_{t+1} = l] \log a_{jl} + \sum_{t=0}^{T} \sum_{j=1}^{k} \mathbb{I}[z_t = j] \log \phi(x_i; \mu_j, \Sigma_j).\end{aligned}$$

E-Step

In the E-Step, we compute the expected complete log likelihood under the distribution $z \sim p(z \mid x; \theta)$. We can see from the form of the complete log likelihood that it is sufficient to compute two types of statistics:

1. $\gamma_t(j) = \mathbb{E}[\mathbb{I}[z_t = j] \mid x; \theta] = p(z_t = j \mid x; \theta)$ for each $t \in \{0, \ldots, T\}$ and $j \in \{1, \ldots, k\}$
2. $\xi_t(j, l) = \mathbb{E}[\mathbb{I}[z_t = j, z_{t+1} = l] \mid x; \theta] = p(z_t = j, z_{t+1} = l \mid x; \theta)$ for each $t \in \{0, \ldots, T\}$ and $(j, l) \in \{1, \ldots, k\}^2$.

Fortunately, we have already learned how to compute these quantities using the α and β recursions. This completes the E-Step.

M-Step

According to our E-step computations, the expected complete log likelihood takes the form

$$\mathbb{E}[\ell_c(\theta)] = \sum_{j=1}^{k} \gamma_0(j) \pi_j + \sum_{t=0}^{T-1} \sum_{j=1}^{k} \sum_{l=1}^{k} \xi_t(j, l) \log a_{jl} + \sum_{t=0}^{T} \sum_{j=1}^{k} \gamma_t(j) \log \phi(x_i; \mu_j, \Sigma_j).$$

In the M-Step, we maximize the ECLL with respect to θ. As it turns out, there are closed form update formulas:

$$\begin{aligned}\pi_j &= \gamma_0(j) \\ a_{jl} &= \frac{\sum_{t=0}^{T-1} \xi_t(j, l)}{\sum_{t=0}^{T-1} \gamma_t(j)}\end{aligned}$$

The μ_j and Σ_j updates, as you might expect, are exactly the same as in the GMM case, with $\gamma_t(j)$ in place of τ_{ij}. The formulas for π_j and a_{jl} are also intuitive. For π, $\gamma_0(j)$ is just the conditional probability that $z_0 = j$. For A, the numerator $\sum_{t=0}^{T-1} \xi_t(j, l)$ is the (conditionally) expected number of transitions from state j to l, while the denominator $\sum_{t=0}^{T-1} \gamma_t(j)$ is the (conditionally) expected number of times the system is in hidden state j, before time T.

5.1.6 Beyond HMMs

HMMs are widely used and widely applicable, but they are still limited by a specific form of dependence structure. Fortunately, efficient generalizations of the α-β recursions (known as **belief propagation** or the **sum-product algorithm**) exist for carrying out probabilistic inference in models with more general tree-like dependence structures.

For many dependence structures (e.g., those with cyclic dependencies), exact probabilistic inference is computationally intractable. However, approximate inference can be carried out efficiently by repeatedly applying related **loopy belief propagation** procedures.

5.2 Beyond Flat Clustering

Thus far in the course we have considered only "flat" or structureless approaches to clustering. Each method has been designed to capture similarity within a cluster, but none have modeled the relationships that may exist amongst clusters. Moreover, these methods all have the often undesirable feature that clusters may change arbitrarily as the number of clusters k varies. That is, it is not necessarily the case that clusters produced when $k = 4$ are subsets of those produced when $k = 3$. **Hierarchical clustering** techniques address these issues by producing a *nested* series of clusterings that reflect the similarity between clusters.

The most ubiquitous versions of hierarchical clustering are model-free and depend on a user-specified dissimilarity measure d between any two datapoints x_i and x_j. Common choices Euclidean distance $||x_i - x_j||_2$ and negative Pearson correlation, but any dissimilarity measure is supported. The measure is used by the hierarchical clustering algorithm to induce dissimilarity measures between and within clusters. As with k-medoids, the standard hierarchical clustering algorithms require access only to the matrix of pairwise datapoint dissimilarities $d_{ij} = d(x_i, x_j)$; the original data need not be retained.

The standard output of a hierarchical clustering run is a *dendrogram*. A dendrogram is a binary tree in which each node represents a cluster. Each leaf (terminal) node is a cluster containing a single datapoint, each parent node represents the cluster formed by merging its two daughter clusters, and the root node represents a single cluster containing the entire dataset. One useful characteristic of a dendrogram is that the height of each node in the dendrogram represents the dissimilarity between its 2 daughter nodes. All terminal nodes lie at height 0 because they have no daughter nodes. This yields an interpretable visualization of the clustering sequence. Moreover, a user can slice a dendrogram at any intercluster dissimilarity level to obtain a single clustering with a number of clusters that varies between 1 (at the root node level) to n, the number of datapoints, (at the zero dissimilarity level).

Remark. The dendrogram height interpretation is only appropriate when the hierarchical clustering algorithm maintains a certain monotonicity property ensuring that daughter dissimilarity associated with a parent node is never smaller than the daughter dissimilarity associated with a child node. However, this property is maintained by the most widely used hierarchical clustering approaches.

There are two principal hierarchical clustering paradigms: *agglomerative* (bottom-up, starting from individual datapoints and merging similar clusters) and *divisive* (top-down,

starting with the entire dataset and gradually dividing into dissimilar clusters). In the next lecture, these two paradigms will be discussed in detail.