**10-708: Probabilistic Graphical Models, Spring 2020**

# Lecture 6: Case Studies: HMM and CRF

*Lecturer: Eric P. Xing* *Scribe: Tejas Srinivasan, Ruochi Zhang, Hongyu Zheng*

## 1 Hidden Markov Models (HMMs)

### 1.1 Introduction

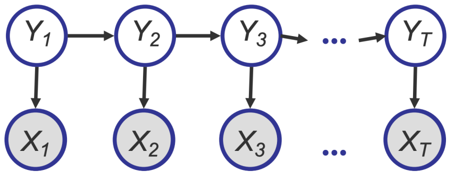

Figure 1: Hidden Markove Models (HMMs)

Hidden Markov Models (HMMs) are one of the common ways of modeling time series data. It is used in traditional speech recognition systems, natural language processing, computational biology, etc. HMMs allow us to deal with both observed and hidden events. Most of the common HMMs follow the first order markov assumption, which states that when predicting the future the past doesn't matter, only the present.

Hidden Markov Models consist of hidden and observed states ($y_{1...T}$ and $x_{1...T}$, respectively), and the dependencies are captured in the network structure seen in Figure 1. The observation and hidden space may be discrete or continuous; for the sake of simplicity, we assume that both are discrete in this case.

We define the following:

- Observation sequence: $\mathbf{x} = x_1, x_2, ..., x_T$
- Hidden sequence (unobserved): $\mathbf{y} = y_1, y_2, ..., y_T$
- Set of observations: $C = \{c_1, c_2, ..., c_K\}$
- Set of hidden states: $I = \{1, 2, ..., M\}$
- Prior probabilities: $p(y_1) \sim \texttt{Multinomial}(\pi_1, \pi_2, ..., \pi_M)$
- Transition probabilities (between hidden states): $p(y_t^j = 1 | y_{t-1}^i = 1) = a_{i,j} \implies p(y_t | y_{t-1}) = \mathbf{A}$
- Emission probabilities: $p(x_t | y_t^i = 1) \sim \texttt{Multinomial}(b_{i,1}, b_{i,2}, ..., b_{i,K}\ )$

## 1.2 Joint Probability Estimation: Forward-Backward Algorithm

Given a sequence of observed states $\mathbf{x} = x_1, x_2, ..., x_T$ and hidden states $\mathbf{y} = y_1, y_2, ..., y_T$, we want to estimate the joint probability $p(\mathbf{x}, \mathbf{y})$

$$\begin{aligned} p(\mathbf{x}, \mathbf{y}) &= p(y_1)p(x_1|y_1)p(y_2|y_1)p(x_2|y_2)...p(y_T|y_{T-1})p(x_T|y_T) \\ &= \pi_{y_1} \prod_{t=2}^{T} p(y_{t-1}|y_t) \prod_{t=1}^{T} p(x_t|y_t) \\ &= p(y_1, ..., y_T)p(x_1, ..., x_T|y_1, ..., y_T) \\ &= p(\mathbf{y})p(\mathbf{x}|\mathbf{y}) \end{aligned}$$

### 1.2.1 Forward Algorithm

We want to estimate $p(\mathbf{x})$, the likelihood of the observed sequence, given the HMM

$p(\mathbf{x}) = \sum_y p(\mathbf{x}, \mathbf{y})$

We define $\alpha(y_t^k = 1) = \alpha_t^k = P(x_1, ..., x_t, y_t^k = 1)$

$$\begin{aligned} \alpha_t^k &= \sum_i p(x_t|y_t^k = 1)p(y_t^k = 1|y_{t-1}^i = 1)p(x_1, ..., x_{t-1}, y_{t-1}^i = 1) \\ &= \sum_i p(x_t|y_t^k = 1)a_{i,k}\alpha_{t-1}^i \\ &= p(x_t|y_t^k = 1)\sum_i a_{i,k}\alpha_{t-1}^i \\ p(\mathbf{x}) &= \sum_k p(x_1, ..., x_T, y_T^k = 1) \\ &= \sum_k \alpha_T^k \end{aligned}$$

The forward probabilities $\alpha_t^k = P(x_1, ..., x_t, y_t^k = 1)$ can be computed recursively.

### 1.2.2 Backward Algorithm

We want to compute $P(y_t^k = 1|\mathbf{x})$, the posterior disributuon of the $t^{th}$ hidden state, given $\mathbf{x}$. We start by estimating $P(y_t^k = 1, \mathbf{x})$

$$\begin{aligned} P(y_t^k = 1, \mathbf{x}) &= P(x_1, ..., x_t, y_t^k = 1, x_{t+1}, ..., x_T) \\ &= P(x_1, ..., x_t, y_t^k = 1)P(x_{t+1}, ..., x_T|x_1, ..., x_t, y_t^k = 1) \\ &= P(x_1, ..., x_t, y_t^k = 1)P(x_{t+1}, ..., x_T|y_t^k = 1) \end{aligned}$$

$P(x_1, ..., x_t, y_t^k = 1)$ is the forward probability. We define the backward probability, $\beta_t^k = P(x_{t+1}, ..., x_T|y_t^k = 1)$.

$$\begin{aligned}
\beta_t^k &= P(x_{t+1}, ..., x_T | y_t^k = 1) \\
&= \sum_i P(x_{t+2}, ..., x_T | y_{t+1}^i = 1) P(x_{t+1} | y_{t+1}^i = 1) P(y_{t+1}^i = 1 | y_t^k = 1) \\
&= \sum_i \beta_{t+1}^i P(x_{t+1} | y_{t+1}^i = 1) a_{k,i}
\end{aligned}$$

We compute $\beta_t^k$ recursively. $P(y_t^k = 1, \mathbf{x}) \propto \alpha_t^k \beta_t^k$

## 1.3 Posterior Decoding

We can now compute the posterior probabilities of the hidden state, $P(y_t^k = 1|\mathbf{x})$.

$$\begin{aligned}
P(y_t^k = 1|\mathbf{x}) &= \frac{P(y_t^k = 1, \mathbf{x})}{P(\mathbf{x})} \\
&= \frac{\alpha_t^k \beta_t^k}{P(\mathbf{x})} \\
&= \frac{\alpha_t^k \beta_t^k}{\sum_k \alpha_T^k}
\end{aligned}$$

## 1.4 Viterbi Decoding

What if we want to find the joint probability of the whole sequence (and the most likely sequence of states)?

$$\vec{y^*} = \text{argmax}_{\vec{y}} P(\vec{y}|\vec{x})$$

This differs from what posterior decoding trying to solve which is the most likely state at position $t$ given the sequence of $\vec{x}$. And it's hard for posterior decoding to solve this problem as it's impossible to store all the probabilities which amount grows exponentially to the size of sequences.

To solve this, the Viterbi algorithm is proposed based on dynamic programming. The recursion for $V_t^k$ which stands for the probability of the most likely sequence of states ending at state $y_t = k$ can be written as:

$$V_t^k = p(x_t | y_t^K = 1) \max_i a_{i,k} V_{t-1}^i$$

One technical issue might encounter is underflows. The probability scores become extremely small after certain times steps. The solution for that would be to take the logarithmic transformation of all values.

## 1.5 Computational Complexity and Implementation Details

The sum-product algorithm is polynomial. The time complexity is $O(K^2N)$, the space complexity is $O(KN)$, where $K$ is the number of states and $N$ number of time steps.

For the Viterbi algorithm, as mentioned above, we can solve the underflow problem via the logarithmic transformation and use the sum of logs instead of multiplication of probabilities.

For the forward/backward algorithm, we can solve the underflow problem by re-scaling the $\alpha$, $\beta$ term with a constant.

## 1.6 Learning HMM

We first make an assumption called "parameter sharing" to further simplify the HMM learning problem. It means we only consider a time-invariant 1st-order Markov model, where we have:

$$\begin{aligned}\pi_k &= p(X_1^K = 1)\\ A_{ij} &= p(X_t^j = 1 | X_{t-1}^i) = 1\end{aligned}$$

### 1.6.1 Supervised Learning

Both sequences of observation and sequences of true hidden states are known. The maximum likelihood parameters are:

$$\begin{aligned}a_{ij} &= \frac{\sum_n \sum_{t=2}^T y_{n,t-1}^i y_{n,t}^j}{\sum_n \sum_{t=2}^T y_{n,t-1}^i}\\ b_{ik} &= \frac{\sum_n \sum_{t=2}^T y_{n,t-1}^i x_{n,t}^k}{\sum_n \sum_{t=2}^T y_{n,t-1}^i}\end{aligned}$$

where $a_{ij}$ is the corresponding entry in the transition matrix from state $i$ to state $j$ and $b_{ik}$ is the corresponding entry in the emission matrix for state $i$ to produce observation $k$.

As shown above, the MLE estimation is actually a "counting scheme" which leads to its drawback. When there's little data, this may be over-fitting. For unobserved transitions or unobserved emissions the probability would be assigned as 0, which is really bad. (Sparse data problem).

This can be resolved by some "smoothing techniques". One of them is called "Pseudocounts". It would mean for each count above, we would add some pseudo counts to make sure it's non-zero. The pseudo count for each entry represents our prior belief, while the total pseudo counts represent the "strength" of prior belief. This is actually equivalent to Bayesian estimation under a uniform prior with "parameter strength" equals to the pseudo counts. Similar techniques have been used in the Bayesian language model where a Dirichlet prior is assigned to each row of the transition matrix.

### 1.6.2 Unsupervised Learning

When the true state path is unknown and only the observation is known. It can be solved by the Baum Welch algorithm (i.e., EM algorithm). It is guaranteed to increase the log-likelihood of the model after each iteration and it will converge to local optimum, depending on initial conditions.

Because in the unsupervised learning setting we cannot optimize over the log-likelihood directly, we optimize over the expected log-likelihood function. As compared to the log-likelihood, we replace the variables related to the unobserved hidden states with sufficient statistics of them. In brief, the E-step of the EM algorithm calculates the sufficient statistics based on the forward and backward algorithm.

$$\gamma^i_{n,t} = p(y^i_{n,t} = 1|x_n)$$
$$\xi^{i,j}_{n,t} = p(y^i_{n,t-1} = 1, y^j_{n,t} = 1|x_n)$$

The M-step then maximize the expected log likelihood which is symbolically identical to MLE.

$$\pi_i = \frac{\sum_n \gamma^i_{n,1}}{N}$$
$$a_{ij} = \frac{\sum_n \sum_{t=2}^{T} \xi^{i,j}_{n,t}}{\sum_n \sum_{t=1}^{T-1} \gamma^i_{n,t}}$$
$$b_{ik} = \frac{\sum_n \sum_{t=1}^{T} \gamma^i_{n,t} x^k_{n,t}}{\sum_n \sum_{t=1}^{T-1} \gamma^i_{n,t}}$$

## 2 Conditional Random Fields (CRFs)

### 2.1 Caveats of HMM

There are two shortcomings of HMM in general:

- HMMs are unable to capture global knowledge. Transitions and emissions in HMMs are all local ("I saw a word last time, and I want to predict the next word with just that"). Higher-order Markov models can alleviate this problem but the cost scales up quickly. This is also prevalent in NLP researches, as emotional and sentimental analysis are likely very non-local.

- A philosophical concern is that HMM optimizes $P(X, Y)$ the joint probability, but what we are more interested in is the conditional probability $P(Y \mid X)$.

A real and challenging example is as follows.

Locally, for every state at every step, State 2 is always the preferred next step and State 1 is never more probable. However in HMM, each step is independent and we can observe a very counter-intuitive phenomenon: The most probable path under HMM is $1-1-1-1$, the path that is never favored locally at every step, with probability 0.09 (in contrast, the probability for path $2-2-2-2$ is only 0.018). This is analogous to the "rib versus rob" example in the famous Lafferty and McCallum paper [1], which was written at Carnegie Mellon.

Here is an intuitive explanation. Imagine State 1 is the introvert who only has two friends, and even $1 \to 1$ is not the most popular one, it is still a very probable choice with probability 0.4. State 2 is the popular guy who likes $2 \to 2$ most, but the probability is still only 0.3. In HMM we compare these numbers, but this is like comparing "friendship" or "stickness" from different people. Intuitively, it should not even be compared from the start.

This Label Bias problem (preference for states with a lower number of transitions) is intrinsic to HMM because we need to normalize the emission probability $P(Y \mid X)$ for every $X$, locally. Our solution is to normalize the probabilities globally instead, called the potentials, and use a global partition function. This step also turns the model into an undirected graphical model.

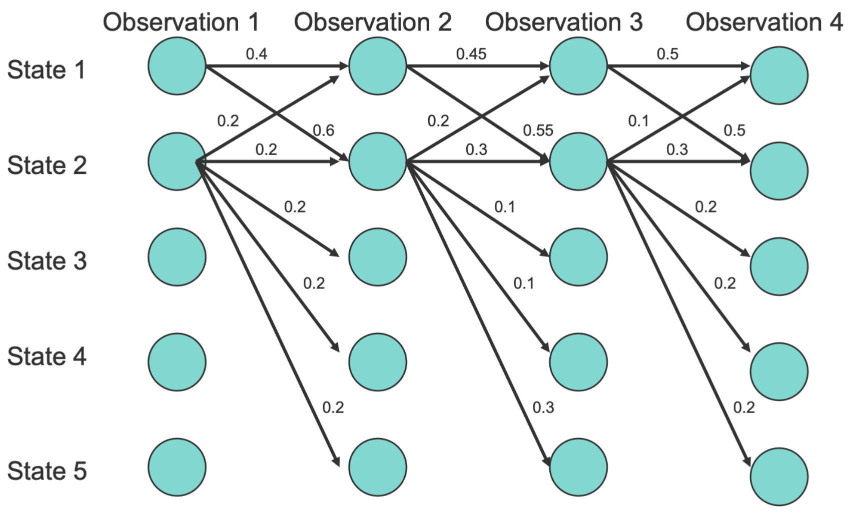

Figure 2: The Label Bias example. Edge denotes probability of transition.

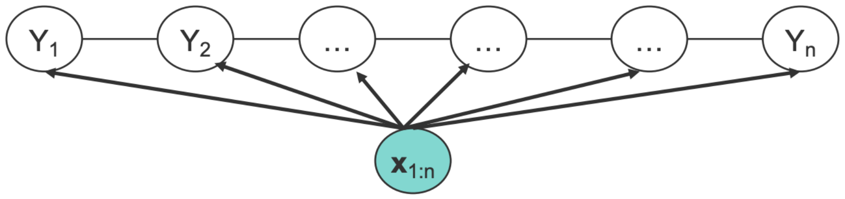

Figure 3: The Conditional Random Field (CRF) model.

## 2.2 The CRF Model

We directly define the conditional distribution with the weighted potential functions, without caring about local normalization, as follows.

$$\begin{aligned} P(\mathbf{y}_{1:n} \mid \mathbf{x}_{1:n}) &= \frac{1}{Z(\mathbf{x}_{1:n})} \prod_{i=1}^{n} \phi(y_i, y_{i-1}, \mathbf{x}_{1:n}) \\ &= \frac{1}{Z(\mathbf{x}_{1:n}, \mathbf{w})} \prod_{i=1}^{n} \exp(\mathbf{w}^T \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_{1:n})) \end{aligned}$$

The potential functions can also be directly defined on cliques of the model, which are now single states and state pairs including the observations (now as features). It is very flexible, utilizing both local and global information.

$$P(\mathbf{y}_{1:n} \mid \mathbf{x}_{1:n}) = \frac{1}{Z(\mathbf{x}_{1:n}, \lambda, \mu)} \exp(\sum_{i=1}^{n} (\lambda^T \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_{1:n}) + \mu^T \mathbf{g}(y_i, \mathbf{x}_{1:n}))$$

This formulation is usually called a feature-based model, with **f** and **g** as edge features and node features.

## 2.3 CRF Inference

With specified model, we are interested in inference: Given $\lambda$ and $\mu$, find the most probable states given observation $\mathbf{x}_{1:n}$:

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} \exp(\sum_{i=1}^{n}(\lambda^T \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_{1:n}) + \mu^T \mathbf{g}(y_i, \mathbf{x}_{1:n}))$$

This can be done by running the message passing algorithm on the junction tree of CRF (which is a chain).

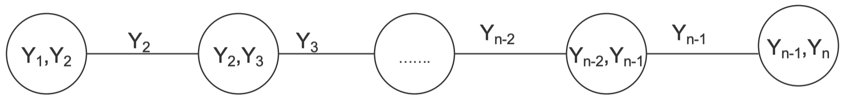

Figure 4: Junction Tree of the CRF.

Its formulation is the same as the Viterbi decoding algorithms used in HMMs.

## 2.4 CRF Learning

For simplicity assume the model is fully observed and we want to find the most probable $\lambda$ and $\mu$. Formally, given $N$ observations $\{\mathbf{x}_d, \mathbf{y}_d\}$, we want to find the following:

$$\begin{aligned}\lambda^*, \mu^* &= \arg\max_{\lambda,\mu} \prod_{i=1}^{N} P(\mathbf{y}_d \mid \mathbf{x}_d, \lambda, \mu) \\ &= \arg\max_{\lambda,\mu} \sum_{d=1}^{N}(\sum_{i=1}^{n}(\lambda^T \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) + \mu^T \mathbf{g}(y_{d,i}, \mathbf{x}_d) - \log Z(\mathbf{x}_d, \lambda, \mu))\end{aligned}$$

It appears we can do this in a one-shot manner (counting frequencies), but unfortunately it does not work for CRF. This is one of the disadvantages of CRF compared to HMM.

Next, we compute the gradient with respect to $\lambda$:

$$\nabla_\lambda L(\lambda, \mu) = \sum_{d=1}^{N}(\sum_{i=1}^{n}(\mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) - \sum_{\mathbf{y}} P(\mathbf{y} \mid \mathbf{x}_d) \sum_{i=1}^{n} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d))$$

The first term here is straightforward from the observations. However to calculate the second term, we assume $\mathbf{y}$ are not observed and take expectation of the same term over all possible $\mathbf{y}$. Their difference defines the gradient. This means even if we observe all $\mathbf{y}$s already, we will pretend we do not know it in one of the steps. In general, learning undirected graphical models requires inference.

It turns out the expectation term is, in fact, tractable as we can decompose it into sums over marginal distributions of neighboring nodes:

$$\begin{aligned}\sum_{\mathbf{y}} P(\mathbf{y} \mid \mathbf{x}_d) \sum_{i=1}^{n} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d)) &= \sum_{i=1}^{n} \sum_{\mathbf{y}} P(\mathbf{y} \mid \mathbf{x}_d) \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d)) \\ &= \sum_{i=1}^{n} \sum_{y_i, y_{i-1}} P(y_i, y_{i-1} \mid \mathbf{x}_d) \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d))\end{aligned}$$

In the end, this is an EM algorithm where we calculate the expectations, then update the parameters with gradient ascent.

### 2.5 Concluding Remarks

After all this work, do we really get a powerful model compared to HMM? In fact, the earlier papers are somewhat disappointing in experimental results. Comparison of error rates on synthetic data (where the label bias is baked into the datasets) shows that CRF is only marginally better than HMM with the presence of label bias, and it is actually slightly worse in some tests without significant label bias. If you show similar results in your ICML/(other ML conference) submissions now, it will be outright rejected. But 15 to 20 years ago people were paying more attention to the mathematical beauty, clarity, and potential of the model rather than benchmarks. Nowadays, HMMs and CRFs are still the backbones of many machine learning algorithms, but when it started the paper is a disaster: hard to read and mediocre numbers. In the end, what is the catch? People do need to publish papers, but at least for this class, to evaluate the course projects, good formulations and clear designs will be more important than “good numbers”, so don’t worry if you don’t get a good result.

## References

[1] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.