University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 19: November 10

Lecturer: Alexander Schwing Scribe: Jingning Tang

1 Overview

This class contains two parts. At first, we combined the structured learning in lecture 12 to 14 with latent variable and EM algorithms talked about in lecture 16 to 18. In the second part, we briefly mentioned Hidden Markov Model (HMM). In this scribe, I will try to walk through the derivation of the first part and then add some key points which I found helpful to understand HMM and some similar models. The notations of latent variable model follows the slides while the notations of HMM will use those from [5]

2 Recap

2.1 Structured Learning with full observations

In lecture 12 to 14, we tried to solve the problem of inference and learning for structured data pairs $(x^{(i)}, \mathbf{y}^{(i)})$, where $\mathbf{y}^{(i)}$ is fully observed. To reduce the search space, the scoring function was decomposed to the summation of individual scoring functions over a restricted set $r \subseteq \{1, ..., D\}$. The formulation of the inference problem thus becomes

$$\arg\max_{\hat{y}} F(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}}) = \arg\max_{\hat{y}} \sum_r f_r(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}}_r) \tag{1}$$

If we penalize whenever maximum is within a margin $L(\hat{\mathbf{y}}, \mathbf{y}^{(i)})$ over ground truth, we will have

$$\max_{\hat{y}}(F(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}}) + L(\hat{\mathbf{y}}, \mathbf{y}^{(i)})) \geq F(\mathbf{w}, x^{(i)}, \mathbf{y}^{(i)}) \tag{2}$$

Our loss-augmented learning framework then becomes

$$\min_{\mathbf{w}} \frac{C}{2}\|\mathbf{w}\|_2^2 + \sum_{i \in \mathcal{D}} \left(\max_{\hat{y}}(F(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}}) + L(\hat{\mathbf{y}}, \mathbf{y}^{(i)})) - F(\mathbf{w}, x^{(i)}, \mathbf{y}^{(i)}) \right) \tag{3}$$

This is further generalized to

$$\min_{\mathbf{w}} \frac{C}{2}\|\mathbf{w}\|_2^2 + \sum_{i \in \mathcal{D}} \epsilon \ln \sum_{\hat{y}} \exp \frac{L(\mathbf{y}^{(i)}, \hat{\mathbf{y}}) + F(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}})}{\epsilon} - F(\mathbf{w}, x^{(i)}, \mathbf{y}^{(i)}) \tag{4}$$

In this equation, we can obtain logistic regression by setting $\epsilon = 1$ and SVM with $\epsilon = 0$. For the scoring function, it can be linear, depicted by $\mathbf{w}^T f(x^{(i)}, \mathbf{y}^{(i)})$, or modeled by deep learning. To optimize it, we use gradient descent to update parameter $\mathbf{w}$ with its gradient. With few simplifications, we derived

$$\nabla_{\mathbf{w}} L = C\mathbf{w} + \sum_{i \in \mathcal{D}, \hat{\mathbf{y}}} (p(\hat{\mathbf{y}}; x^{(i)}, \mathbf{w})) \nabla_{\mathbf{w}} F(\mathbf{w}, x^{(i)}, \hat{\mathbf{y}}) \tag{5}$$

repeat
  repeat
    update marginals b_r
  until *convergence*;
  update parameter $\mathbf{w}$
until *convergence*;

Algorithm 1: General optimization algorithm for structured prediction

There is no exact way to compute $p(\hat{\mathbf{y}}; x^{(i)}, \mathbf{w})$ (also called as belief in the slide, $b_r(\hat{\mathbf{y}}; x^{(i)}, \mathbf{w})$), so we use methods such as LP relaxation or message passing to approximate that.
Note that even though we have not formally introduced EM algorithms in lecture 14, the opimization steps are very similar.

2.2 Latent Variable Model

In Gaussian mixture model (GMM), we used a linear superposition of Gaussian model. To optimize it, we could minimize its negative log-liklihood with constraints

$$\min_{\pi,\mu,\sigma} -\log \prod_{i\in\mathcal{D}} p(x^{(i)}|\pi,\mu,\sigma) \coloneqq -\sum_{i\in\mathcal{D}} \log \sum_{k=1}^{K} \pi_k \mathcal{N}(x^{(i)}|\mu_k,\sigma_k) \quad \mathbf{s.t.} \sum_{k=1}^{K} \pi_k = 1 \tag{6}$$

Since there is no closed form solution, we introduced the latent variable $z_{ik} \in \{0,1\}$
with $\sum_{k=1}^{K} z_{ik} = 1 \; \forall i$. With the help of the latent variable, we can found an equivalent formulation of $p(x^{(i)}|\pi,\mu,\sigma)$ as the following

$$p(x) = \sum_{\mathbf{z}_i} p(\mathbf{z}_i) p(x^{(i)}|\mathbf{z}_i) = \sum_{\mathbf{z}_i} (\prod_{k=1}^{K} \pi_k^{z_{ik}}) (\prod_{k=1}^{K} \mathcal{N}(x^{(i)}|\mu_k,\sigma_k) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x^{(i)}|\mu_k,\sigma_k) \tag{7}$$

This reformulation of the GMM does not seem to be useful at the first sight, but now we are able to work with the joint distribution $p(x,z)$ [1]. Instead of the marginal distribution $p(x)$. With the help of posterior, denoted as $r_{ik} \coloneqq p(z_{ik}=1|x^{(i)})$, applying EM algorithm on GMM will be significantly simplified.
We also learned concave-convex procedure (CCCP) in generalizing EM algorithms. The CCCP follows the step as below. Keep in mind of the decomposition of $\ln \sum_{\mathbf{z}} \exp F(x^{(i)}, \mathbf{z}, \theta)$. This is useful in the later part for structure latent variable models

Initialize θ ;
repeat
- Decompose concave part into 'convex + concave' at current θ
- Solve convex program

$$\min_{\theta} \ln Z(\theta) - \ln \sum_{\mathbf{z}} \exp F(x^{(i)}, \mathbf{z}, \theta) \tag{8}$$

  Where

$$\ln \sum_{\mathbf{z}} \exp F(x^{(i)}, \mathbf{z}, \theta) = \max_{q(\mathbf{z})} \left(\sum_{\mathbf{z}} q(\mathbf{z}) F(x^{(i)}, \mathbf{z}, \theta) + H(q(\mathbf{z})) \right) \tag{9}$$

until *convergence*;

Algorithm 2: General procedures for CCCP

3 Structure Latent Variable Models

The aforementioned latent variable z_{ik} can be generalized beyond GMM into the structured prediction. Here z can be either unobserved data or explicitly introduced latent variable. In the given example below, only the important objects are labeled in the image, making it a weakly labeled image. Now the complete data now becomes $\mathbf{y} = (\mathbf{s}, \mathbf{z})$

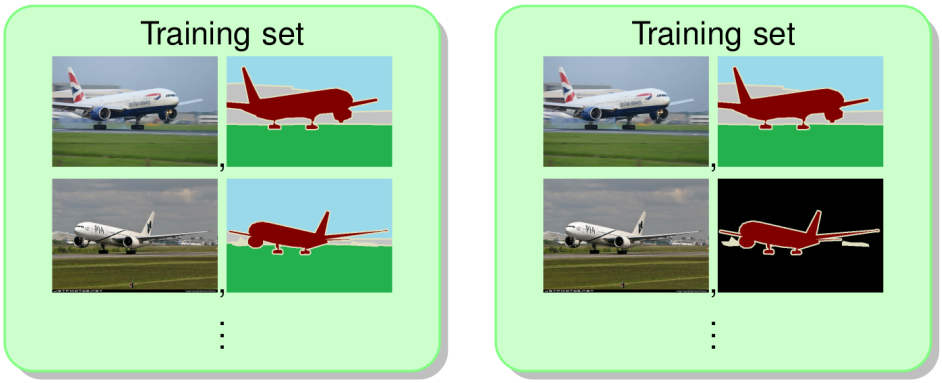

Figure 1: Left images are fully labeled whereas right images only label the airplane and leave ground, sky and buildings as blank

With similar fashion as the equation 2, we use hinge loss for augmentation. However, this time we penalize whenever best overall prediction exceeds best prediction with annotation being clamped

$$\max_{\hat{\mathbf{s}},\hat{\mathbf{z}}} F(\mathbf{w}, x^{(i)}, \hat{\mathbf{s}}, \hat{\mathbf{z}}) \geq \max_{\hat{\mathbf{z}}} F(\mathbf{w}, x^{(i)}, \mathbf{s}^{(i)}, \hat{\mathbf{z}}) \tag{10}$$

Then we have Latent Structured SVM (LSSVM)

$$\min_{\mathbf{w}} \frac{C}{2}\|\mathbf{w}\|_2^2 + \sum_{i\in\mathcal{D}} \max_{\hat{\mathbf{s}},\hat{\mathbf{z}}} F(\mathbf{w}, x^{(i)}, \hat{\mathbf{s}}, \hat{\mathbf{z}}) - \sum_{i\in\mathcal{D}} \max_{\hat{\mathbf{z}}} F(\mathbf{w}, x^{(i)}, \mathbf{s}^{(i)}, \hat{\mathbf{z}}) \tag{11}$$

Then just as equation 5, we generalize it into

$$\min_{\mathbf{w}} \frac{C}{2}\|\mathbf{w}\|_2^2 + \sum_{i\in\mathcal{D}} \epsilon \ln \sum_{\hat{\mathbf{s}},\hat{\mathbf{z}}} \exp \frac{F(\mathbf{w}, x^{(i)}, \hat{\mathbf{s}}, \hat{\mathbf{z}})}{\epsilon} - \sum_{i\in\mathcal{D}} \epsilon \ln \sum_{\hat{\mathbf{s}},\hat{\mathbf{z}}} \exp \frac{F(\mathbf{w}, x^{(i)}, \mathbf{s}^{(i)}, \hat{\mathbf{z}})}{\epsilon} \tag{12}$$

such that when $\epsilon = 0$, we will have the hinge-loss and lead to LSSVM and when $\epsilon = 1$, the loss will become log-loss and we will obtain hidden conditional random field (HCRF). When the loss augmentation is also considered, the lraning framework can be expanded again, thus becoming the most universal learning objective we can learn in this class.

$$\begin{aligned}\min_{\mathbf{w}} \frac{C}{2}\|\mathbf{w}\|_2^2 &+ \sum_{i\in\mathcal{D}} \epsilon \ln \sum_{\hat{\mathbf{s}},\hat{\mathbf{z}}} \exp \frac{F(\mathbf{w}, x^{(i)}, \hat{\mathbf{s}}, \hat{\mathbf{z}}) + L_i(\hat{\mathbf{s}}, \hat{\mathbf{z}})}{\epsilon} \\ &- \sum_{i\in\mathcal{D}} \epsilon \ln \sum_{\hat{\mathbf{s}},\hat{\mathbf{z}}} \exp \frac{F(\mathbf{w}, x^{(i)}, \mathbf{s}^{(i)}, \hat{\mathbf{z}}) + L_i^c(\mathbf{s}^{(i)}, \hat{\mathbf{z}})}{\epsilon}\end{aligned} \tag{13}$$

Using the decomposition mentioned in the CCCP, we can decompose the third term into

$$\max_{q_i(\mathbf{z})} \left(\sum_{\mathbf{z}} F(\mathbf{w}, x^{(i)}, \mathbf{s}^{(i)}, \hat{\mathbf{z}}) + L_i^c(\mathbf{s}^{(i)}, \hat{\mathbf{z}}) + \epsilon H(q_i(\mathbf{z})) \right) \tag{14}$$

so that the optimization will follow the same procedures as the EM algorithm, alternating between $q(\mathbf{z})$ and $\mathbf{w}$.

4 Hidden Markov Model

4.1 Overview

Hidden Markov models (HMM) contain two main ideas: Markov chains and hidden states. The Markov chains built on the idea that given the current state X_t, the future prediction is independent of the past. If we assume discrete time steps, then we will have the joint distribution

$$p(X_{1:T}) = p(X_1)p(X_2|X_1)p(X_3|X_2)... = p(X_1)\prod_{t=2}^{T} p(X_t|X_{t-1}) \tag{15}$$

With the addition of the hidden states, we model both the transitions between different states and also observation given the current state. The corresponding joint distribution has the form

$$p(\mathbf{z}_{1:T}, \mathbf{x}_{1:T}) = p(\mathbf{z}_{1:T})p(\mathbf{x}_{1:T}|\mathbf{z}_{1:T}) = \left[p(z_1)\prod_{t=2}^{T} p(z_t|z_{t-1})\right]\left[\prod_{t=2}^{T} p(x_t|z_t)\right] \tag{16}$$

The observation probabilities are also called *emission* probabilities in some textbooks since the observed data x emits from the corresponding state z. The following shows a graphical model representation of the HMM, where $z_{i,j}$ models the transition between state i to state j and z_i, y_i models the emission from state i to observation i.

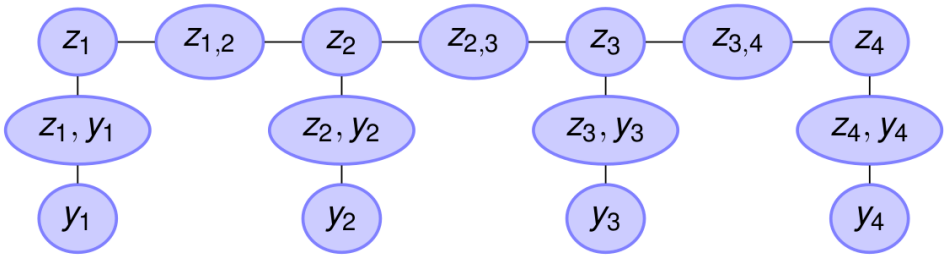

Figure 2: Hidden Markov Model

4.2 Inference

There are multiple methods to infer the hidden states, given the observation and learned parameters, such as forward algorithms, forward-backward algorithms and Viterbi algorithms. These three different methods are generally used based on the type of inference. Below are some notable types of inference.

- **Filtering** means to compute **belief state** $p(z_t|\mathbf{x}_{1:t})$ online, or recursively, as the data streams in. Usually forward algorithm is used here.

Initialization: $t = 0$, transition probability $p(z_t|z_{t-1})$, emission probability $p(y_j|z_i)$ and observation $y_1, ..., y_t$;
for $t = 2$ *to* T **do**
  $\alpha_t(z_t) = p(y_t|x_t) \sum_{x_{t-1}} p(x_t|x_{(t-1)})\alpha_{t-1}(x_{t-1})$
end
Output: $p(y(1:t)) = \alpha_T$

Algorithm 3: Forward algorithm

- **Smoothing** means to compute $p(z_t|\mathbf{x}_{1:t})$ offline, given all the evidence. The forward-backward algorithm is often used. Below is an example of Python implementation of forward-backward algorithm from.

Inputs: Length m, set of possible states $\mathcal{S}$, function $\psi(s, s', j)$. Define * to be a special initial state.

Initialization (forward terms): For all $s \in \mathcal{S}$,

$$\alpha(1, s) = \psi(*, s, 1)$$

Recursion (forward terms): For all $j \in \{2 \dots m\}, s \in \mathcal{S}$,

$$\alpha(j, s) = \sum_{s' \in \mathcal{S}} \alpha(j-1, s') \times \psi(s', s, j)$$

Initialization (backward terms): For all $s \in \mathcal{S}$,

$$\beta(m, s) = 1$$

Recursion (backward terms): For all $j \in \{1 \dots (m-1)\}, s \in \mathcal{S}$,

$$\beta(j, s) = \sum_{s' \in \mathcal{S}} \beta(j+1, s') \times \psi(s, s', j+1)$$

Calculations:

$$Z = \sum_{s \in \mathcal{S}} \alpha(m, s)$$

For all $j \in \{1 \dots m\}, a \in \mathcal{S}$,

$$\mu(j, a) = \alpha(j, a) \times \beta(j, a)$$

For all $j \in \{1 \dots (m-1)\}, a, b \in \mathcal{S}$,

$$\mu(j, a, b) = \alpha(j, a) \times \psi(a, b, j+1) \times \beta(j+1, b)$$

Figure 3: forward-backward algorithm [2]

- **MAP estimation** means computing $\arg\max_{\mathbf{z}_{1:t}} p(\mathbf{z}_{1:t}|\mathbf{x}_{1:t})$, which is the most probable state sequence given all the observations. Viterbi decoding is usually applied on this task. The details of Viterbi algorithms will be omitted in this scribe, but there is a PyTorch tutorial implementing Bi-LSTM CRF with Viterbi decoding posted on ref and is highly recommended for practice.

function VITERBI(*observations* of len T,*state-graph* of len N) **returns** *best-path*

create a path probability matrix *viterbi*$[N+2,T]$
for each state s **from** 1 **to** N **do** ; initialization step
viterbi$[s,1] \leftarrow a_{0,s} * b_s(o_1)$
backpointer$[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
for each state s **from** 1 **to** N **do**
viterbi$[s,t] \leftarrow \max_{s'=1}^{N}$ *viterbi*$[s',t-1] * a_{s',s} * b_s(o_t)$
backpointer$[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N}$ *viterbi*$[s',t-1] * a_{s',s}$
viterbi$[q_F,T] \leftarrow \max_{s=1}^{N}$ *viterbi*$[s,T] * a_{s,q_F}$; termination step
backpointer$[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N}$ *viterbi*$[s,T] * a_{s,q_F}$; termination step
return the backtrace path by following backpointers to states back in time from *backpointer*$[q_F,T]$

Figure 4: viterbi algorithm [4]

4.3 Learning

The parameters of HMM contain $\theta = (\pi, A, B)$, where π is the initial state distribution, A is the transition matrix and B is the emission matrix.

- If the hidden state sequences are observed, maximum likelihood can be simply applied to learn the HMM parameters. The details of B depend on the form of the observation model. For example, if each state has a Gaussian distribution associated with it, we will have the following MLEs:

$$\hat{\mu}_k = \frac{\bar{x}_k}{N_k}, \quad \hat{\Sigma}_k = \frac{\overline{xx}_k^T - N_k \hat{\mu}_k \hat{\mu}_k^T}{N_k} \tag{17}$$

  where the sufficient statistics are given by

$$\bar{x}_k \coloneqq \sum_{i=1}^{N} \sum_{i=1}^{T_i} \mathbb{I}(z_{i,k} = k) x_{i,k} \quad \overline{xx}_k^T - N_k \coloneqq \sum_{i=1}^{N} \sum_{i=1}^{T_i} \mathbb{I}(z_{i,k} = k) x_{i,k} x_{i,k}^T \tag{18}$$

- If the hidden state sequences are not observed, we can use the Baum-Welch algorithm to train HMM. Essentially it is the EM algorithm utilizing forward-backward algorithm.

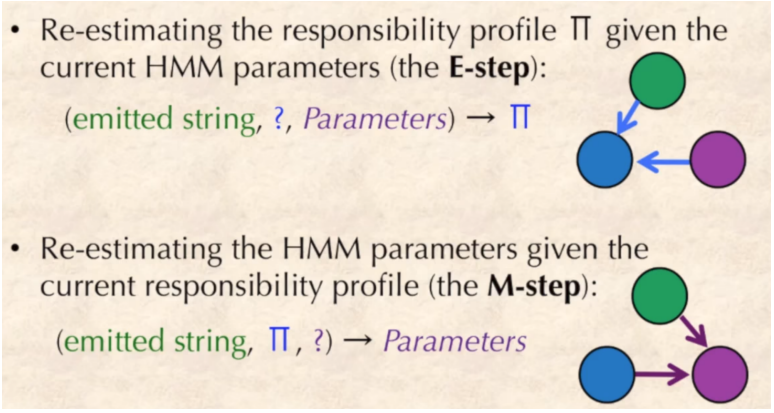

Figure 5: Baum-Welch Algorithm [3]

4.4 Application

There are wide varieties of HMM application, such as

- **Part of speech tagging**. Here x_t represents a word, and z_t represents its part of speech (noun, verb, adjective, etc.)

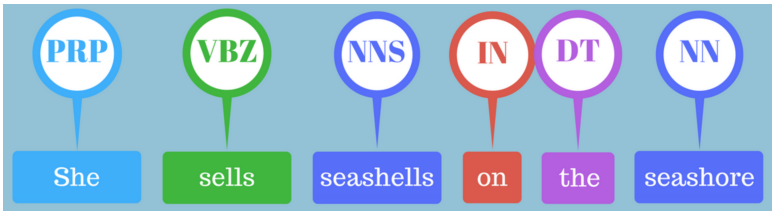

Figure 6: Part of Speech Tagging

- **Protein sequence alignment** arranges sequence of protein to identify regions of similarity. Here x_t represents an amino acid, and z_t represents whether this matches the latent consensus sequence at this location.

Figure 7: Protein Sequence Alignment

4.5 Misc

- While aforementioned problems are typical applications of HMM, in real world people generally use CRF instead of HMM to solve the problems. CRF can be intuitively understood as a "discriminative" version of HMM. By directly modeling the conditional distribution, CRF do not make conditional independence assumptions among x (observations). Think about the application part-of-speech tagging, the tag of certain word is definitely affected by its surrounding words, even length of the sentence, etc. Therefore, CRF usually outperforms HMM in the tasks above.
- Currently we have the powerful tool of deep learning, so we usually see RNN as the mainstream tool to perform the sequence tagging. Then the question arises, "whey people sometimes use LSTM + CRF to do the tagging instead of just either of them". This is because while LSTM can have better overall accuracy, it can sometimes lead to naive mistake such as a verb right after another verb in sequence tagging. The CRF can enforce such rule to avoid this mistake. Therefore, the deep model will learn many non-linear features while the CRF provides simple yet powerful rules to improve upon deep learning.

References

[1] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[2] M. Collins. *The Forward-Backward Algorithm*. New York, NY.

[3] P. Compeau. *Bioinformatics Algorithms An Active Learning Approach*. Active Learning Publishers, 2014.

[4] M. Dubremetz. *HMM Decoding: Viterbi Algorithm*. Uppsala, Sweden, 2011.

[5] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.