Course: Model, Learning, and Inference: Lecture 5

Alan Yuille
Department of Statistics, UCLA
Los Angeles, CA 90095
yuille@stat.ucla.edu

Abstract

Probability distributions on structured representation. Dynamic Programming. Learning with EM. NOTE: NOT FOR DISTRIBUTION!!

1 Introduction

We discuss how to define probabilistic models that use richly structured probability distributions and describe how *graphical models* can be used to represent the dependencies among a set of variables.

Then we describe dynamic programming and EM for learning.

2 Representing structured probability distributions

A probabilistic model defines the joint distribution for a set of random variables. For example, imagine that a friend of yours claims to possess psychic powers – in particular, the power of psychokinesis. He proposes to demonstrate these powers by flipping a coin, and influencing the outcome to produce heads. You suggest that a better test might be to see if he can levitate a pencil, since the coin producing heads could also be explained by some kind of sleight of hand, such as substituting a two-headed coin. We can express all possible outcomes of the proposed tests, as well as their causes, using the binary random variables X_1, X_2, X_3, and X_4 to represent (respectively) the truth of the coin being flipped and producing heads, the pencil levitating, your friend having psychic powers, and the use of a two-headed coin. Any set of beliefs about these outcomes can be encoded in a joint probability distribution, $P(x_1, x_2, x_3, x_4)$. For example, the probability that the coin comes up heads ($x_1 = 1$) should be higher if your friend actually does have psychic powers ($x_3 = 1$).

Once we have defined a joint distribution on X_1, X_2, X_3, and X_4, we can reason about the implications of events involving these variables. For example, if flipping the coin produces heads ($x_1 = 1$), then the probability distribution over the remaining variables is

$$P(x_2, x_3, x_4|x_1 = 1) = \frac{P(x_1 = 1, x_2, x_3, x_4)}{P(x_1 = 1)}. \quad (1)$$

This equation can be interpreted as an application of Bayes' rule, with X_1 being the data, and X_2, X_3, X_4 being the hypotheses. However, in this setting, as with most probabilistic models, any variable can act as data or hypothesis. In the general case, we use probabilistic inference to compute the probability distribution over a set of *unobserved* variables (here, X_2, X_3, X_4) conditioned on a set of *observed* variables (here, X_1).

Another common pattern of influence is *explaining away*. Imagine that your friend flipped the coin, and it came up heads ($x_1 = 1$). The propositions that he has psychic powers ($x_3 = 1$) and that it is a two-headed coin ($x_4 = 1$) might both become more likely. However, while these two variables were independent before seeing the outcome of the coinflip, they are now dependent: if you were to go on to discover that the coin has two heads, the hypothesis of psychic powers would return to its baseline probability – the evidence for psychic powers was "explained away" by the presence of the two-headed coin.

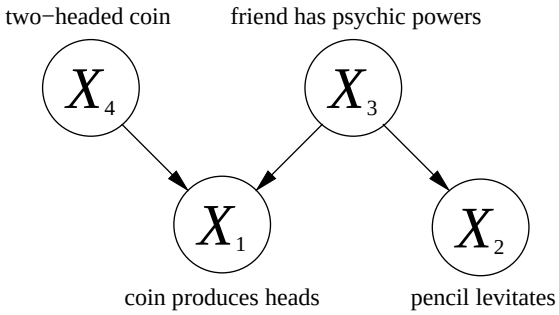

Figure 1: Directed graphical model (Bayes net) showing the dependencies among variables in the "psychic friend" example discussed in the text.

2.1 Directed graphical models

Directed graphical models, also known as Bayesian networks or Bayes nets, consist of a set of nodes, representing random variables, together with a set of directed edges from one node to another, which can be used to identify statistical dependencies between variables. Typically, nodes are drawn as circles, and the existence of a directed edge from one node to another is indicated with an arrow between the corresponding nodes. If an edge exists from node A to node B, then A is referred to as the "parent" of B, and B is the "child" of A. This genealogical relation is often extended to identify the "ancestors" and "descendants" of a node.

The directed graph used in a Bayes net has one node for each random variable in the associated probability distribution. The edges express the statistical dependencies between the variables in a fashion consistent with the *Markov condition*: conditioned on its parents, each variable is independent of all other variables except its descendants. This has an important implication: a Bayes net specifies a canonical factorization of a probability distribution into the product of the conditional distribution for each variable conditioned on its parents. Thus, for a set of variables $X_1, X_2, \ldots, X_M$, we can write $P(x_1, x_2, \ldots, x_M) = \prod_i P(x_i|\mathrm{Pa}(X_i))$ where $\mathrm{Pa}(X_i)$ is the set of parents of X_i.

Figure 1 shows a Bayes net for the example of the friend who claims to have psychic powers. This Bayes net identifies a number of assumptions about the relationship between the variables involved in this situation. For example, X_1 and X_2 are assumed to be independent given X_3, indicating that once it was known whether or not your friend was psychic, the outcomes of the coin flip and the levitation experiments would be completely unrelated. By the Markov condition, we can write $P(x_1, x_2, x_3, x_4) = P(x_1|x_3, x_4)P(x_2|x_3)P(x_3)P(x_4)$. This factorization allows us to use fewer numbers in specifying the distribution over these four variables: we only need one number for each variable, conditioned on each set of values taken on by its parents. In this case, this adds up to 8 numbers rather than 15.

2.2 Undirected graphical models

Undirected graphical models, also known as Markov Random Fields (MRFs), consist of a set of nodes, representing random variables, and a set of undirected edges, defining neighbourhood structure on the graph which indicates the probabilistic dependencies of the variables at the nodes. Each set of fully-connected neighbors as associated with a *potential* function, which varies as the associated random variables take on different values. When multiplied together, these potential functions give the probability distribution over all the variables. Unlike directed graphical models, there need be no simple relationship between these potentials and the local conditional probability distributions. Moreover, undirected graphical models usually have closed loops (if they do not, then they can be reformulated as directed graphical models).

In this section we will use X_i to refer to variables whose values can be directed observed and Y_i to refer to latent, or hidden, variables whose values can only be inferred, see figure (4). We will use the vector notation $\vec{x}$ and $\vec{y}$ to represent the values taken by these random variables, being $x_1, x_2, \ldots$ and $y_1, y_2, \ldots$ respectively.

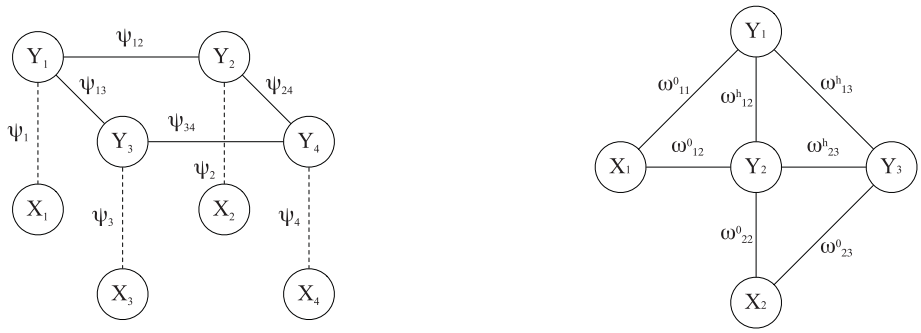

Figure 2: The left panel illustrates an MRF model where $\Lambda = \{(1,2),(2,3),(3,4),(4,1)\}$, see text for more details. The right panel is a Boltzmann Machine, see text for details.

A standard model used in vision is of form: $P(\vec{x}|\vec{y}) = (\prod_i P(x_i|y_i))\, P(\vec{y})$ where the prior distribution on the latent variables is an MRF,

$$P(\vec{y}) = \frac{1}{Z} \prod_{i,j \in \Lambda} \psi_{ij}(y_i, y_j) \prod_i \psi_i(y_i) \tag{2}$$

where Z is a normalizing constant ensuring that the resulting distribution sums to 1. Here, $\psi_{ij}(\cdot,\cdot)$ and $\psi_i(\cdot)$ are the potential functions, and the underlying graph is a lattice, with Λ being the set of connected pairs of nodes (see Figure 2). This model has many applications. For example, $\vec{x}$ can be taken to be the observed intensity values of a corrupted image and $\vec{y}$ the true image intensity, with $P(x_i|y_i)$ modeling the corruption of these intensity values. The prior $P(\vec{y})$ is used to put prior probabilities on the true intensity, for example that neighbouring intensity values are similar (e.g. that the intensity is spatially smooth). A similar model can be used for binocular stereopsis, where the $\vec{x}$ correspond to the image intensities in the left and right eyes and $\vec{y}$ denotes the depth of the surface in space that generates the two images. The prior on $\vec{y}$ can assume that the depth is a spatially smoothly varying function.

The neighbourhood structure, given by Λ above, will typically correspond to a graph structure with closed loops, see figure. A graph with no closed loops is called a tree. It can easily be shown that an undirected graphical model on a tree can be reformulated as a directed graphical model on the same tree. This will make it significantly easier to perform inference, see next section. Tree models occur naturally for problems whose structure is one-dimensional such as speech recognition. See Hidden Markov Models in figure (5). For vision problems, however, undirected graphical models will be defined on graphs with closed loops. In this case inference becomes more difficult and requires more complex algorithms that we describe in the next section.

Another example of an MRF is the Boltzmann Machine, which has been very influential in the neural network community. In this model the components x_i and y_i of the observed and latent variables $\vec{x}$ and $\vec{y}$ all take on values 0 or 1. The standard model is

$$P(\vec{y}, \vec{x}|\vec{\omega}) = \frac{1}{Z} \exp\{-E(\vec{y}, \vec{x}, \vec{\omega})/T\} \tag{3}$$

where T is a parameter reflecting the "temperature" of the system, and E depends on unknown parameters $\vec{\omega}$ which are weighted connections ω^h_{ij} between hidden variables y_i, y_j and ω^o_{ij} between observed and hidden variables x_i, y_j,

$$E(\vec{y}, \vec{x}, \vec{\omega}) = \sum_{ij} \omega^o_{ij} x_i y_j + \sum_{ij} \omega^h_{ij} y_i y_j. \tag{4}$$

In this model, the potential functions are of the form $\exp\{-\omega^o_{ij} x_i y_j\}$ and $\exp\{-\omega^h_{ij} y_i y_j\}$, and the underlying graph connects pairs of observed and hidden variables and pairs of hidden variables (see Figure 2). Training the Boltzmann Machine involves identifying the correct potential functions, learning the parameters $\vec{\omega}$ from training examples. Viewing Equation 3 as specifying the likelihood of a statistical model, inferring $\vec{\omega}$ can be formulated as a problem of Bayesian inference of the kind discussed above.

This learning is just another form of Bayesian inference. Suppose we have a set $\{\vec{x}^\mu : \mu = 1, ..., M\}$ of training samples. We first must sum out over the latent variables to obtain $P(\vec{x}|\vec{w}) = \sum_{\vec{y}} P(\vec{x}, \vec{y}|\vec{\omega})$. Then we calculate the

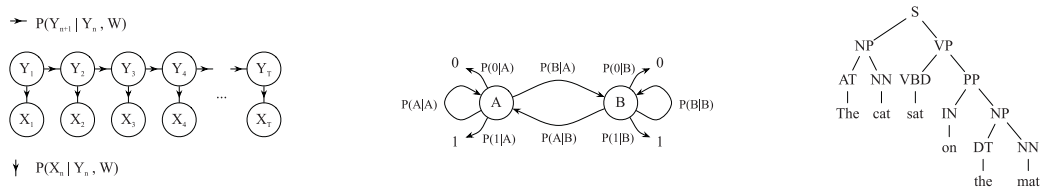

Figure 3: The left and center panels illustrate Hidden Markov Models. The right panel is a parsing tree from a Probabilistic Context Free Grammar. See text for details.

Latent variables and mixture models

In many problems, the observable data are believed to reflect some kind of underlying latent structure. For example, in a clustering problem, we might only see the location of each point, but believe that each point was generated from one of a small number of clusters. Associating the observed data with random variables X_i and the latent variables with random variables Y_i, we might want to define a probabilistic model for X_i that explicitly takes into account the latent structure Y_i. Such a model can ultimately be used to make inferences about the latent structure associated with new datapoints, as well as providing a more accurate model of the distribution of X_i.

A simple and common example of a latent variable model is a *mixture model*, in which the distribution of X_i is assumed to be a mixture of several other distributions. For example, in the case of clustering, we might believe that our data were generated from two clusters, each associated with a different Gaussian (i.e. normal) distribution. If we let y_i denote the cluster from which the datapoint x_i was generated, and assume that there are K such clusters, then the probability distribution over x_i is

$$P(x_i) = \sum_{k=1}^{K} P(x_i|y_i = k)P(y_i = k) \tag{5}$$

where $P(x_i|y_i = k)$ is the distribution associated with cluster k, and $P(y_i = k)$ is the probability that a point would be generated from that cluster. If we can estimate the parameters that characterize these distributions, we can infer the probable cluster membership (y_i) for any datapoint (x_i).

Figure 4: Latent Variables.

probability of generating the samples $\prod_{\mu=1}^{M} P(\vec{x}^{\mu}|\vec{\omega})$. Next we put a prior $P(\vec{\omega})$ on the weights and compute the posterior $P(\vec{\omega}|\{\vec{x}^{\mu} : \mu = 1, ..., M\} \propto P(\vec{\omega}) \prod_{\mu=1}^{M} P(\vec{x}^{\mu}|\vec{\omega})$. Learning the $\vec{\omega}$ can be performed by MAP estimation of $\vec{\omega}$. For large number M of training data the choice of prior $P(\vec{\omega})$ becomes unimportant because the data dominates.

2.3 Probabilistic Distributions on Grammars

We can also define probability distributions on more complex structures. For example, probabilistic/stochastic grammars have become important in computational linguistics where the structure of the representation is itself a random variable so that it depends on the input data, see Figures 3,**??**. This is desirable, for example, when segmenting images into objects because the number of objects in an image can be variable. Another important new class of models defines probability distributions on relations between objects. This allows the properties of an object to depend probabilistically both on other properties of that object and on properties of related objects. One of the great strengths of probabilistic models is this capacity to combine structured representations with statistical methods, providing a set of tools that can be used to explore how structure and statistics are combined in human cognition.

Hidden Markov Models

Hidden Markov models (or HMMs) are an important class of one-dimensional graphical models that have been used for problems such as speech and language processing. For example, see Figure 3 (left panel), the HMM model for a word W assumes that there are a sequence of T observations $\{x_t : t = 1, ..., T\}$ (taking L values) generated by a set of hidden states $\{y_t : t = 1, ..., T\}$ (taking K values). The joint probability distribution is defined by $P(\{y_t\}, \{x_t\}, W) = P(W)P(y_1|W)P(x_1|y_1, W) \prod_{t=2}^{T} P(y_t|y_{t-1}, W)P(x_t|y_t, W)$. The HMM for W is defined by the probability distributions $P(y_1|W)$, the $K \times K$ probability transition matrix $P(y_t|y_{t-1}, W)$, the $K \times L$ observation probabilty matrix $P(x_t|y_t, W)$, and the prior probability of the word $P(W)$. Applying HMM's to recognize the word requires algorithms, based on dynamic programming and EM, to solve three related inference tasks. Firstly, we need to learn the models (i.e. $P(x_t|y_t, W)$ and $P(y_t|y_{t-1}, W)$) for each word W. Secondly, we need to evaluate the probability $P(\{x_t\}, W) = \sum_{\{y_t\}} P(\{y_t\}, \{x_t\}, W)$ for the observation sequence $\{x_t\}$ for each word W. Thirdly, we must recognize the word by model selection to estimate $W^* = \arg\max_W \sum_{\{y_t\}} P(\{y_t\}, W|\{x_t\})$.

Figure 3 (center panel) shows a simple example of a Hidden Markov Model HMM consists if two coins, one biased and the other fair, with the coins switched occasionally. The observable $0, 1$ is whether the coin is heads or tails. The hidden state A, B is which coin is used. There are (unknown) transition probabilities between the hidden states A and B, and (unknown) probabilities for the obervations $0, 1$ conditioned on the hidden states. Given this graph, the learning, or training, task of the HMM is to estimate the probabilities from a sequence of measurements. The HMM can then be used to estimate the hidden states and the probability that the model generated the data (so that it can be compared to alternative models for classification).

Figure 5: Hidden Markov Models.

Box 5: Probabilistic Context Free Grammars

A probabilistic context free grammar (or PCFG) is a context free grammar that associates a probability with each of its production rules. A PCFG can be used to generate sentences and to parse them. The probability of a sentence, or a parse, is defined to be the product of the probabilities of the production rules used to generate the sentence.

For example, we define a PCFG to generate a parse tree as follows, see Figure 3 (right panel). We define non-terminal nodes $S, NP, VP, AT, NNS, VBD, PP, IN, DT, NN$ where S is a sentence, VP is a verb phrase, VBD is a verb,NP is a noun phrase, NN is a noun, and so on. The terminal nodes are words from a dictionary (e.g. "the", "cat", "sat", "on", "mat".) We define production rules which are applied to non-terminal nodes to generate child nodes (e.g. $S \mapsto NP, VP$ or $NN \mapsto$ "cat"). We specify probabilities for the production rules.

These production rules enable us to generate a sentence starting from the root node S. We sample to select a production rule and apply it to generate child nodes. We repeat this process on the child nodes and stop when all the nodes are terminal (i.e. all are words). To parse an input sentence, we use dynamic programming to compute the most probable way the sentence could have been generated by the production rules.

We can learn the production rules, and their probabilities, from training data of sentences. This can be done in a supervised way, where the correct parsing the sentences is known. Or in an unsupervised way, as described by Klein and Manning.

Probabilistic context free grammars have several important features. Firstly, the number of nodes in these graphs are variable (unlike other models where the number of nodes is fixed). Moreover, they have independence properties so that different parts of the tree are independent. This is, at best, an approximation for natural language and more complex models are needed.

Figure 6: Probabilistic Context Free Grammars.

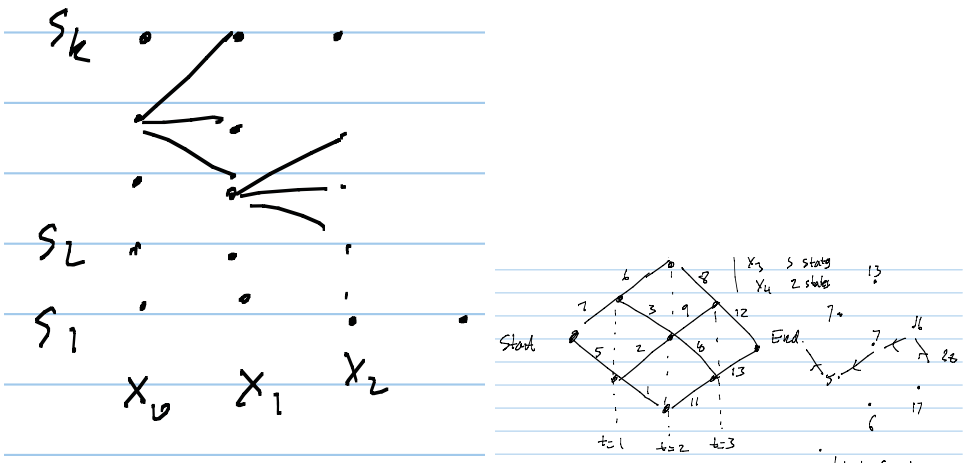

Figure 7: Examples of Dynamic Programming.

3 Inference and Learning

Inference and learning algorithmic can exploit the structure of the underlying graph structure of the probability distribution. For example, in order to evaluate Equation 1, we need to compute

$$\begin{aligned} P(x_1 = 1) &= \sum_{x_2}\sum_{x_3}\sum_{x_4} P(x_1 = 1, x_2, x_3, x_4) \\ &= \sum_{x_2}\sum_{x_3}\sum_{x_4} P(x_1 = 1|x_3, x_4)P(x_2|x_3)P(x_3)P(x_4) \\ &= \sum_{x_3}\sum_{x_4} P(x_1 = 1|x_3, x_4)P(x_3)P(x_4) \end{aligned} \tag{6}$$

where we were able to sum out X_2 directly as a result of its independence from X_1 when conditioned on X_3.

3.1 Dynamic Programming for Inference

In particular, if the underlying graph is a tree (i.e. it has no closed loops), dynamic programming (DP) algorithms can be used to exploit this structure, see figure (**??**). The intuition behind DP can be illustrated by planning the shortest route for a trip from Los Angeles to Boston. To determine the cost of going via Chicago, you only need to calculate the shortest route from LA to Chicago and then, independently, from Chicago to Boston. Decomposing the route in this way, and taking into account the linear nature of the trip, gives an efficient algorithm with convergence rates which are polynomial in the number of nodes and hence are often feasible for computation. Equation (6) is one illustration for how the dynamic programming can exploit the structure of the problem to simplify the computation. These methods are put to particularly good use in hidden Markov models, see figure (5).

Dynamic Programming can be used when we can express the graph as a tree structure and hence order the nodes.

$$P(\vec{y}) = \frac{1}{Z}\prod_{i=1}^{N-1}\psi_i(y_i, y_{i+1})\prod_{i=1}^{N-1}\psi_i(y_i) \tag{7}$$

We maximize equation (7) with respect to y by using the max rule version of dynamic programming. This can be done recursively by defining a function $h_i(y_i)$ by a *forward pass*:

$$h_i(y_i) = \max_{y_{i-1}} h_{i-1}(y_{i-1})\psi_i(y_{i-1}, y_i)\psi_i(y_i), \tag{8}$$

where $h_1(y_1) = 1$.

The forward pass computes the maximum value of $P(\vec{y})$. The backward pass of dynamic programming compute the most probable value $\vec{y}^*$.

The computational complexity of the dynamic programming algorithm is $O(MN^K)$ where M is the number of cliques in the aspect model for the object, $K = 2$ is the size of the maximum clique.

NOTE: WE CAN ALSO USE DYNAMIC PROGRAMMING TO COMPUTE THINGS LIKE THE MARGINALS $P_i(y_i)$ and the PARTITION FUNCTION Z.

These are done by the sum-rule of dynamic programming – which replaces equation (8) by

$$h_i(y_i) = \sum_{y_{i-1}} h_{i-1}(y_{i-1})\psi_i(y_{i-1}, y_i)\psi_i(y_i). \tag{9}$$

EXAMPLE: ISING MODEL??

Dynamic programming can also be used for inference of probabilistic context free grammars (PCFG).This is because the key idea of dynamic programming is independence – and PCFGs produce trees where the different branches are independent of each other. Dynamic Programming can only be applied directly to problems which are defined on tree structures (because this allows us to order the nodes – the ordering does not have to be unique).

What if you do not have a tree structure (i.e., you have closed loops)? There is an approach called *junction trees* which shows that you can transform any probability distribution on a graph into a probability distribution on a tree by enhancing the variables. The basic idea is triangulation (Lauritzen and Spiegelhalter). But this, while useful, is limited because the resulting trees can be enormous.

In practice, people often use pruned DP to speed up the process. (SEE LATER). Also you can view DP as a special case of A*!!

3.2 EM and Dynamic Programming for Learning

Dynamic programming can also be used as a component of learning. Suppose we have a parameterized distribution $P(\vec{x}, \vec{y}|\vec{\lambda})$ where $\vec{x}$ is observed, $\vec{y}$ are the hidden states, and $\vec{\lambda}$ are the model parameters (to be learnt). Given training data $\{\vec{x}_\mu\}$ we should estimate $\vec{\lambda}$ by maximum likelihood (ML) by computing:

$$\vec{\lambda}^* = \arg\max_{\vec{\lambda}} \prod_\mu P(\vec{x}_\mu|\vec{\lambda}). \tag{10}$$

This requires eliminating the hidden variables which is usually intractable analytically. Instead we can apply the EM algorithm to minimize (local minima) a free energy:

$$F(\{q_\mu\}, \vec{\lambda}) = \sum_\mu \sum_{\vec{y}^\mu} q_\mu(\vec{y}^\mu) \log q_\mu(\vec{y}^\mu) - \sum_\mu \sum_{\vec{y}^\mu} q_\mu(\vec{y}^\mu) \log P(\vec{x}_\mu, \vec{y}^\mu|\vec{\lambda}). \tag{11}$$

NOTE that we have distributions $q_\mu(.)$ for the states $\vec{y}_\mu$ of all training examples. But the parameters $\vec{\lambda}$ are common for all examples.

The EM algorithm for equation (11) has the following E-step and M-step:

$$\begin{aligned} \text{E} - \text{Step} \quad & q_\mu^{t+1}(\vec{y}_\mu) = P(\vec{y}_\mu|\vec{x}_\mu, \vec{\lambda}^t), \\ \text{M} - \text{step} \quad & \vec{\lambda}^{t+1} = \arg\min \sum_\mu \sum_{\vec{y}_\mu} q_\mu^{t+1}(\vec{y}_\mu) \log P(\vec{x}_\mu, \vec{y}_\mu)|\vec{\lambda}). \end{aligned} \tag{12}$$

There are two stages where dynamic programming helps makes the steps practical. Firstly, computing $P(\vec{y}_\mu|\vec{x}_\mu, \vec{\lambda}^t)$ from $P(\vec{y}, \vec{x}|\vec{\lambda})$ (E-step) is practical because we can set $P(\vec{y}_\mu|\vec{x}_\mu, \vec{\lambda}^t) \propto P(\vec{y}, \vec{x}|\vec{\lambda})$ and then use DP (sum-rule) to

compute the normalization constant. Secondly, the summation $\sum_{\vec{y}_\mu}$ (M-step) can also be computed using DP (sum-rule).

To make the EM practical we must perform the minimization of the RHS with respect to $\vec{\lambda}$. This can be done depending on the how the distribution depends on $\vec{\lambda}$. Suppose we assume the general form:

$$P(\vec{x}, \vec{y}|\vec{\lambda}) = \frac{1}{Z[\vec{\lambda}]} \exp\{\vec{\lambda} \cdot \vec{\phi}(\vec{x}, \vec{y})\}, \tag{13}$$

then the M-step reduces to solving:

$$\vec{\lambda}^{t+1} = \arg\min\{\sum_{\mu} \sum_{\vec{y}_\mu} q_\mu^{t+1}(\vec{y}_\mu)\vec{\lambda} \cdot \vec{\phi}(\vec{x}_\mu, \vec{y}_\mu) - N \log Z[\vec{\lambda}]. \tag{14}$$

Compare this to the task of learning a distribution without hidden variables. This requires solving $\vec{\lambda}^* = \arg\max_{\vec{\lambda}} P(\{\vec{x}_\mu\}|\vec{\lambda})$ which reduces to solving $\vec{\lambda}^* = \arg\min_{\vec{\lambda}}\{\vec{\lambda} \cdot \vec{\phi}(\vec{x}_\mu) - N \log Z[\vec{\lambda}]\}$. Recall that this can be solved by steepest descent or generalized iterative scaling (GIS) but requires that we can compute terms like $\sum_{\vec{x}} \vec{\phi}(\vec{x}) P(\vec{x}|\vec{\lambda})$ which may be very difficult. In short, *one iteration* of EM for learning parameters of a distribution with hidden variables is as difficult as estimating the parameters for a distribution which does not have any hidden variables.

Next lecture will give examples for learning Hidden Markov Models (HMMs) and Probabilistic Context Free Grammars (PCFGs).