COMPSCI 688: Probabilistic Graphical Models

Lecture 10: Learning in MRFs

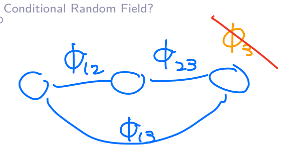
Dan Sheldon

Manning College of Information and Computer Sciences
University of Massachusetts Amherst

Partially based on materials by Benjamin M. Marlin (marlin@cs.umass.edu) and Justin Domke (domke@cs.umass.edu)

Learning in MRFs

Learning in Pairwise MRFs



Let's consider the problem of learning in a pairwise MRF with only edge potentials:

$$p_\theta(\mathbf{x}) = \frac{1}{Z(\theta)} \prod_{(i,j)\in E} \phi_{ij}(x_i, x_j; \theta), \qquad Z(\theta) = \sum_{\mathbf{x}} \prod_{(i,j)\in E} \phi_{ij}(x_i, x_j; \theta)$$

Parameterized as

$$\phi_{ij}(a, b; \theta) = \exp(\theta_{ij}^{ab})$$

x_i	x_j	ϕ_{ij}
0	0	$\exp(\theta_{ij}^{00})$
0	1	$\exp(\theta_{ij}^{01})$
1	0	$\vdots$
1	1	

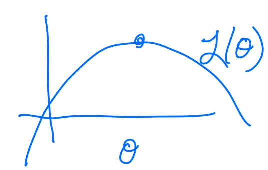
Learning in Pairwise MRFs

$(x_1^{(1)}, \ldots, x_d^{(1)})$

The learning problem is: given a data set $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}$, find θ to maximize

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \log p_\theta(\mathbf{x}^{(n)})$$

To solve this, we need to compute derivatives of $\mathcal{L}(\theta)$.



Log-Likelihood of Single Datum

$\frac{1}{Z(\theta)} \prod_{i,j} \phi_{ij}(x_i, x_j; \theta)$

Let's start by reformulating the log-likelihood of a single datum $\mathbf{x}$. Write

energy = − log prob

$$p_\theta(\mathbf{x}) = \frac{1}{Z(\theta)} \exp(-E_\theta(\mathbf{x}))$$

where $-E_\theta(\mathbf{x})$ is the *negative energy*:

$\sum_{(i,j)} \log \phi_{ij}(x_i, x_j; \theta) = \sum_{(i,j)} \log \exp(\theta_{ij}^{x_i x_j})$

$$-E_\theta(\mathbf{x}) = \log \prod_{(i,j)\in E} \phi_{ij}(x_i, x_j; \theta) = \sum_{(i,j)\in E} \theta_{ij}^{x_i x_j}$$

The log-likelihood of datum $\mathbf{x}$ is:

$$\log p_\theta(\mathbf{x}) = \underbrace{-E_\theta(\mathbf{x})}_{\text{linear in } \theta} - \underbrace{\log Z(\theta)}_{\text{nonlinear}}$$

The derivative with respect to a generic parameter θ_{uv}^{ab} is

Z = "normalizing constant" = "partition function"

$$\frac{\partial}{\partial \theta_{uv}^{ab}} \log p_\theta(\mathbf{x}) = \frac{\partial}{\partial \theta_{uv}^{ab}} (-E_\theta(\mathbf{x})) - \frac{\partial}{\partial \theta_{uv}^{ab}} \log Z(\theta)$$

We'll treat each term separately.

Negative Energy Derivative

$$\begin{array}{ccccc} 0 & & 0 & & 1 \\ x_1 & - & x_2 & - & x_3 \end{array}$$

$-E_\theta(x) = \theta_{12}^{00} + \theta_{23}^{01}$

Recall the negative energy definition:

$$-E_\theta(\mathbf{x}) = \sum_{(i,j)\in E} \theta_{ij}^{x_i x_j}.$$

$\frac{\partial}{\partial \theta_{12}^{01}}\left(\theta_{12}^{00} + \theta_{23}^{01}\right) = 0$

$\frac{\partial}{\partial \theta_{12}^{00}}\left(\theta_{12}^{00} + \theta_{23}^{01}\right) = 1$

Its derivative is easy, because it is linear in the parameters

$$\frac{\partial}{\partial \theta_{uv}^{ab}} (-E_\theta(\mathbf{x})) = \frac{\partial}{\partial \theta_{uv}^{ab}} \sum_{(i,j)\in E} \theta_{ij}^{x_i x_j} = \mathbb{I}[x_u = a, x_v = b]$$

Log-Partition Function Derivative

$Z(\theta) = \sum_{x'} \exp(-E_\theta(x'))$

The derivative of the log-partition function has a special form.

$$\begin{aligned} \frac{\partial}{\partial \theta_{uv}^{ab}} \log Z(\theta) &= \frac{1}{Z(\theta)} \cdot \frac{\partial}{\partial \theta_{uv}^{ab}} Z(\theta) \\ &= \frac{1}{Z(\theta)} \cdot \frac{\partial}{\partial \theta_{uv}^{ab}} \sum_{x'} \exp(-E_\theta(x')) \\ &= \frac{1}{Z(\theta)} \sum_{x'} \frac{\partial}{\partial \theta_{uv}^{ab}} \exp(-E_\theta(x')) \\ &= \frac{1}{Z(\theta)} \sum_{x'} \exp(-E_\theta(x')) \cdot \frac{\partial}{\partial \theta_{uv}^{ab}}(-E_\theta(x')) \\ &= \sum_{x'} \frac{\exp(-E_\theta(x'))}{Z(\theta)} \cdot \mathbb{I}[x'_u = a, x'_v = b] \end{aligned}$$

$$= \sum_{x'} p_\theta(x') \cdot \mathbb{I}[x'_u = a, x'_v = b]$$

$$= P_\theta(X_u = a, X_v = b)$$

Takeaways:

- derivative of log-partition function is a marginal probability
- example of more general phenomenon (exponential family models)

Put Together

Put together, the derivative of the log-likelihood of a single datum is

$$\frac{\partial}{\partial \theta_{uv}^{ab}} \log p_\theta(\mathbf{x}) = \mathbb{I}[x_u = a, x_v = b] - P_\theta(X_u = a, X_v = b)$$

Log-Likelihood of N Data Points

With N data points, the derivative of the log-likelihood is

$$\boxed{\frac{\partial}{\partial \theta_{uv}^{ab}}} \mathcal{L}(\theta) = \frac{\partial}{\partial \theta_{uv}^{ab}} \frac{1}{N} \sum_{n=1}^{N} \log p_\theta(\mathbf{x}^{(n)}) = \frac{1}{N} \sum_{n=1}^{N} \left(\mathbb{I}[x_u^{(n)} = a, x_v^{(n)} = b] - P_\theta(X_u = a, X_v = b) \right)$$

$$= \left(\frac{1}{N} \sum_{n=1}^{N} \mathbb{I}[x_u^{(n)} = a, x_v^{(n)} = b] \right) - P_\theta(X_u = a, X_v = b)$$

$$= \frac{\#(X_u = a, X_v = b)}{N} - P_\theta(X_u = a, X_v = b)$$

The derivative is data marginal minus a model marginal.

$p_\theta(x) = \prod_c \phi_c(x_c)$

$$\frac{\partial}{\partial \theta_c^a} \log p_\theta(x) = \frac{\#(X_c = a)}{N} - P_\theta(X_c = a)$$

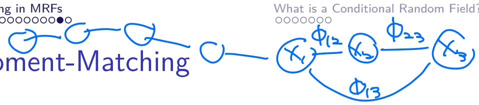
Computing the Derivatives

$$\boxed{\frac{\partial}{\partial \theta_{uv}^{ab}} \mathcal{L}(\theta) = \frac{\#(X_u = a, X_v = b)}{N} - P_\theta(X_u = a, X_v = b)} = 0$$

How do we compute the derivative?

- first term: counting, easy, iterate through data
- second term: compute a marginal in MRF w/ params θ
  inference! message-passing / variable elimination
  → key subroutine

Moment-Matching



Each partial derivative must be zero at a maximum. This gives the *moment-matching* condition, which asserts the data marginal should match the model marginal:

"Solve" for θ

$$\boxed{\frac{\#(X_u = a, X_v = b)}{N} = P_\theta(X_u = a, X_v = b)}$$

$\forall (u,v) \in E$
$\forall a \in Val(X_u)$
$\forall b \in Val(X_v)$

This is similar to counting in Bayes net learning, but **the marginal** $P_\theta(X_u = a, X_v = b)$ **depends on *all* parameters**, not just the "local parameters" $\theta_{uv}^{\cdot}$, because of the global normalization constant $Z(\theta)$.

The moment matching conditions for all parameters form a system of equations. It has a "unique" solution (the distribution is unique, not the parameters), but it's not easy to solve directly.

Learning via Optimization

Instead, we can numerically maximize the log-likelihod, for example by gradient ascent:

- Initialize θ (e.g. $\theta \leftarrow 0$)
- Repeat
  - $\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}(\theta)$

vector of all partials

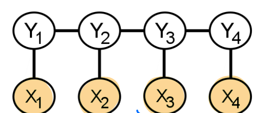
We saw above how to compute the entries of the gradient $\nabla_\theta L(\theta)$.

The key subroutine is inference in the MRF.

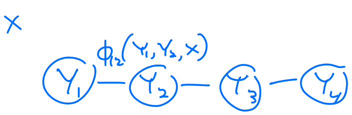
What is a Conditional Random Field?

What is a Conditional Random Field?

Before we describe a CRF informally as an MRF where the $\mathbf{x}$ variables are always observed.



$p(x,y)$ $\Rightarrow$



$p(y|x)$

Here's a better definition. A CRF defines an MRF over $\mathbf{y}$ *for every fixed value of* $\mathbf{x}$:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{c \in \mathcal{C}} \phi_c(\mathbf{x}, \mathbf{y}_c), \qquad Z(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{c \in \mathcal{C}} \phi_c(\mathbf{x}, \mathbf{y}_c)$$

Example
Chain model $p(y|x) = \frac{1}{Z(x)} \prod_{i=1}^{N} \phi_i(x_i, y_i) \cdot \prod_{i=1}^{N-1} \Phi_{i,i+1}(y_i, y_{i+1})$

Notes:

- No distribution over $\mathbf{x}$
- Normalized separately for each $\mathbf{x}$
- Each potential ϕ_c can depend arbitrarily on $\mathbf{x}$ (often designed with "local" connections to selected entries of $\mathbf{x}$, but not necessary)
- Cliques c are subsets of the $\mathbf{y}$ indices

Learning in CRFs

$(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})$ $\quad \sum_n \log p_\theta(x^{(n)}, y^{(n)})$

In CRFs, we maximize the *conditional log-likelihood*:

$$\max_\theta \frac{1}{N} \sum_{n=1}^{N} \log p_\theta(\mathbf{y}^{(n)} | \mathbf{x}^{(n)})$$

Some aspects are similar to learning in MRFs. A key difference is that the "model marginals" are different for each data case, because the normalization constant $Z(\mathbf{x}^{(n)})$ is different.

(see HW2, HW3)

Discussion

discriminative CRF $p(y|x)$

generative learning MRF $p(x,y)$

Why CRFs?

- It's often better not to learn a model for $p(\mathbf{x})$ if it is not needed, e.g., if you only want to predict $p(\mathbf{y}|\mathbf{x})$. This is especially true if we have lots of data.
- But it may be better to use an MRF and learn a full model $p(\mathbf{x}, \mathbf{y})$ for the joint distribution, especially if the model is "correct" and with smaller data sets. (Intuition: the $\mathbf{x}$ data can help you learn the correct model faster.)

bias / variance

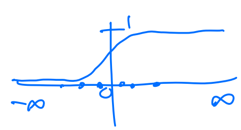
Example: Logistic Regression

$\mathbb{R}^d$ $\quad x \to y$ $\quad \{0,1\}$

Logistic regression is a simple CRF with $y \in \{0, 1\}$.

$$\log p_\theta(y|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp(\underbrace{\theta^\top \mathbf{x} \cdot \mathbb{I}[y=1]}_{\phi(x,y;\theta)}) = \begin{cases} 1 & y=0 \\ \exp(\theta^\top x) & y=1 \end{cases}$$

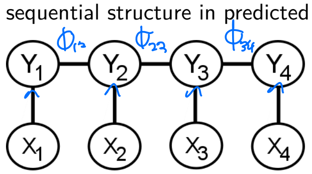
$$Z(\mathbf{x}) = \exp(\theta^\top \mathbf{x}) + 1$$

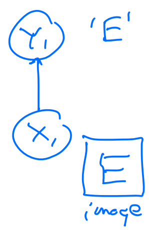


$$p_\theta(y=1|\mathbf{x}) = \frac{\exp(\theta^\top \mathbf{x})}{1 + \exp \theta^\top \mathbf{x}} = \text{sigmoid}(\theta^\top x)$$

Example: Chain CRF

One way to view a chain-structured CRF is as a sequence of logistic regression models, with pairwise connctions between adjacent y variables to encourage a particular sequential structure in predicted labels:





Message-Passing Implementation

Overflow/Underflow and Log-Sum-Exp

$p(x) = \frac{1}{Z} \prod_c \phi_c(x_c)$

- When factor values are small or large, or with many factors, messages can underflow or overflow since they are products of many terms. A common solution is to manipulate all factors and messages in log space.
- **Example**: consider the common factor manipulation

$$A(x) = \sum_y \overbrace{B(x,y)C(y)}^{\exp(\lambda(x,y))}$$

Let's compute $\alpha(x) = \log A(x)$ from $\beta(x,y) = \log B(x,y)$ and $\gamma(y) = \log C(y)$

- **Step 1**: multiplication of factors is addition of log-factors

$$\lambda(x,y) := \log(B(x,y)C(y)) = \beta(x,y) + \gamma(y)$$

- **Step 2**: marginalization requires exponentiation ("log-sum-exp")

$$\alpha(x) = \log\left(\overbrace{\sum_y \exp \lambda(x,y)}^{A(x)}\right)$$

$\lambda(x,\cdot)$

Numerically Stable log-sum-exp

Before exponentiating, we need to be careful to shift values to avoid overflow/underflow

logsumexp$(a_1, \ldots, a_k)$: $\log \sum_{i=1}^{k} \exp a_i$

- $c \leftarrow \max_i a_i$
- return $c + \log \sum_i \exp(a_i - c)$

See `scipy.special.logsumexp`

(Comment: log-space implementation probably not needed in HW2, probably needed in HW3.)