NLP 201: CRFs Continued

Jeffrey Flanigan

November 9, 2021

University of California Santa Cruz
jmflanig@ucsc.edu

Many slides and figures from David Bamman and Noah Smith

Plan for Today

- Maximum entropy markov models (MEMMs) and the label bias problem
- Conditional random fields (again)
- Neural CRFs
- Markov Random Fields (MRFs)

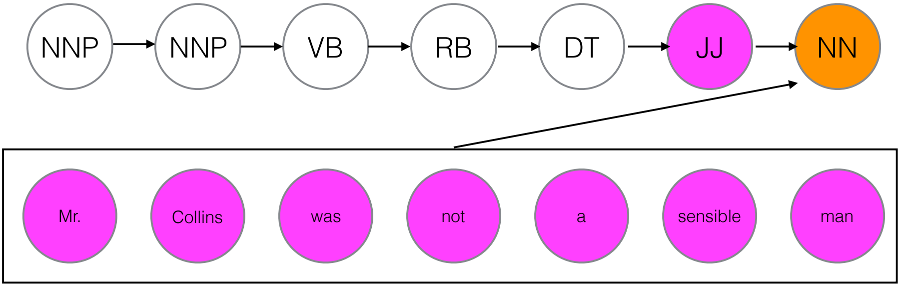
Motivation: Conditional Random Fields (CRFs)

Last time I motivated CRFs as an extension of logistic regression to sequences.

This time I will motivate CRFs from a different angle: the label bias problem

Maximum-Entropy Markov Models (MEMMs) and the label bias problem

MEMM



Features

$$f(y_i, y_{i-1}; x_1, \ldots, x_n)$$

Features are scoped over the previous predicted tag and the entire observed input

feature	example
x_i = man	1
y_{i-1} = JJ	1
i=n (last word of sentence)	1
x_i ends in -ly	0

Training

$$\prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)$$

For all training data, we want probability of the true label y_i conditioned on the previous true label y_{i-1} to be high.

This is simply multiclass logistic regression

Decoding

- With logistic regression, our prediction is simply the argmax y:

$$P(y \mid x, \beta)$$

- With an MEMM, we know the true y_{i-1} during training but we never of course know it at test time

$$P(y_i \mid y_{i-1}, x, \beta)$$

Greedy decoding

- A i=1, predict the argmax given START:

$$P(y_1 \mid START, x, \beta)$$

- For each subsequent time step, condition on the y just predicted during the step before

$$P(y_i \mid y_{i-1}, x, \beta)$$

Viterbi decoding

Viterbi for HMM: max joint probability

$$P(y)P(x \mid y) = P(x, y)$$

$$v_t(y) = \max_{u \in \mathcal{Y}} \left[v_{t-1}(u) \times P(y_t = y \mid y_{t-1} = u) P(x_t \mid y_t = y) \right]$$

Viterbi for MEMM: max conditional probability

$$P(y \mid x)$$

$$v_t(y) = \max_{u \in \mathcal{Y}} [v_{t-1}(u) \times P(y_t = y \mid y_{t-1} = u, x, \beta)]$$

MEMM Training

$$\prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)$$

For all training data, we want probability of the true label y_i conditioned on the previous true label y_{i-1} to be high.

This is simply multiclass logistic regression

MEMM Training

$$\prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)$$

Locally normalized — at each time step, each conditional distribution sums to 1

Label bias

$$\prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)$$

- For a given conditioning context, the probability of a tag (e.g., VBZ) only competes against other tags with that same context (e.g., NN)

Bottou 2001; Lafferty et al. 2001

Label bias

NN	TO	VB

will to fight

	NN	MD
x_i=will	10	40
y_{i-1}=START	-1	7
BIAS	7	-2

Modals show up much more frequently at the start of the sentence than nouns do (e.g., questions)

Toutanova et al. 2003

Label bias

NN TO VB

will to fight

But we know that MD + TO is very rare

- *can to eat
- *would to eat
- *could to eat
- *may to eat

Toutanova et al. 2003

Label bias

NN TO VB

will to fight

	TO
x_i=to	10000000
y_{i-1}=NN	0
y_{i-1}=MD	0

to is relatively deterministic (almost always TO) so it doesn't matter what tag precedes it.

Toutanova et al. 2003

Label bias

NN TO VB

will to fight

$$\prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)$$

Because of this local normalization, P(TO | context) will always be 1 if x=“to”

Toutanova et al. 2003

Label bias

NN TO VB

will to fight

That means our prediction for *to* can't help us disambiguate *will*. We lose the information that MD + TO sequences rarely happen.

Toutanova et al. 2003

Label bias

Viterbi decoding doesn't help in this case

$$v_t(y) = \max_{u \in \mathcal{Y}} [v_{t-1}(u) \times P(y_t = y \mid y_{t-1} = u, x, \beta)]$$

$$P(y_t = \text{TO} \mid y_{t-1} = \text{MD}, x, \beta) = 1$$

$$P(y_t = \text{TO} \mid y_{t-1} = \text{NN}, x, \beta) = 1$$

Conditional random fields

- We can solve this problem using global normalization (over the entire sequences) rather than locally normalized factors.

MEMM

$$P(y \mid x, \beta) = \prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)$$

CRF

$$P(y \mid x, \beta) = \frac{\exp(\Phi(x, y)^\top \beta)}{\sum_{y' \in \mathcal{Y}} \exp(\Phi(x, y')^\top \beta)}$$

Conditional random fields

$$P(y \mid x, \beta) = \frac{\exp(\Phi(x,y)^\top \beta)}{\sum_{y' \in \mathcal{Y}} \exp(\Phi(x,y')^\top \beta)}$$

Feature vector scoped over the entire input and label sequence

$$\Phi(x,y) = \sum_{i=1}^{n} \phi(x, i, y_i, y_{i-1})$$

ϕ is the same feature vector we used for local predictions using MEMMs

Features

$$\phi(x, i, y_i, y_{i-1})$$

Features are scoped over the previous predicted tag and the entire observed input

feature	example
x_i = man	1
y_{i-1} = JJ	1
i=n (last word of sentence)	1
x_i ends in -ly	0

In a CRF, we use features from the entire sequence (by summing the individual features at each time step)

	will $\phi(x, 1, y_1, y_0)$	to $\phi(x, 2, y_2, y_1)$	fight $\phi(x, 3, y_3, y_2)$	$\mathbf{\Phi}$(x, NN TO VB)
x_i=will ^ y_i = NN	1	0	0	1
y_{i-1}=START ^ y_i = NN	1	0	0	1
x_i=will ^ y_i = MD	0	0	0	0
y_{i-1}=START ^ y_i = MD	0	0	0	0
…				
x_i=to ^ y_i = TO	0	1	0	1
y_{i-1}=NN ^ y_i = TO	0	1	0	1
y_{i-1}=MD ^ y_i = TO	0	0	0	0
…				
x_i=fight ^ y_i = VB	0	0	1	1
y_{i-1}=TO ^ y_i = VB	0	0	1	1

Implementation: Features in Linear Models

In linear models, the easiest way to represent feature vectors is a **dictionary from strings to features values**.

	will $\phi(x, 1, y_1, y_0)$	to $\phi(x, 2, y_2, y_1)$	fight $\phi(x, 3, y_3, y_2)$	Φ(x, NN TO VB)
x_i=will ^ y_i = NN	1	0	0	1
y_{i-1}=START ^ y_i = NN	1	0	0	1
x_i=will ^ y_i = MD	0	0	0	0
y_{i-1}=START ^ y_i = MD	0	0	0	0
…				
x_i=to ^ y_i = TO	0	1	0	1
y_{i-1}=NN ^ y_i = TO	0	1	0	1
y_{i-1}=MD ^ y_i = TO	0	0	0	0
…				
x_i=fight ^ y_i = VB	0	0	1	1
y_{i-1}=TO ^ y_i = VB	0	0	1	1

Conditional random fields

$$P(y \mid x, \beta) = \frac{\exp(\Phi(x,y)^\top \beta)}{\sum_{y' \in \mathcal{Y}} \exp(\Phi(x,y')^\top \beta)}$$

- In MEMMs, we normalize over the set of 45 POS tags
- CRFs are globally normalized, but the normalization complexity is huge — every possible sequence of labels of length n.

Forward algorithm (CRF)

$$P(y \mid x, \beta) = \frac{\exp(\Phi(x,y)^\top \beta)}{\sum_{y' \in \mathcal{Y}} \exp(\Phi(x,y')^\top \beta)}$$

- Calculating the denominator naively would involve a summation over K^N terms
- But we can do this efficiently in NK^2 time using the forward algorithm

For details, see: Collins, "The Forward-Backward Algorithm"

Forward algorithm (CRF)

END							
DT							
NNP							
VB							
NN							
MD							
START							
	^	Janet	will	back	the	bill	$

$$\alpha(1, y) = \exp\left(\phi(x, i, y, \text{START})^\top \beta\right)$$

$$\alpha(i, y) = \sum_{y' \in \mathcal{S}} \alpha(i-1, y') \times \exp\left(\phi(x, i, y, y')^\top \beta\right)$$

Forward algorithm (CRF)

END							
DT							
NNP							
VB							
NN							
MD							
START							
	^	Janet	will	back	the	bill	$

$$Z = \sum_{y' \in \mathcal{Y}} \exp\left(\Phi(x, y')^\top \beta\right) = \sum_{s \in \mathcal{S}} \alpha(n, s)$$

Conditional random fields

With a CRF, we have exactly the same parameters as we do with an equivalent MEMM; but we learn the best values of those parameters that leads to the best probability of the sequence overall (in our training data)

MEMM

	TO
x_i=to ^ y_i = TO	10000000
y_{i-1}=NN ^ y_i = TO	0
y_{i-1}=MD ^ y_i = TO	0

CRF

	TO
x_i=to ^ y_i = TO	7.8
y_{i-1}=NN ^ y_i = TO	1.4
y_{i-1}=MD ^ y_i = TO	-5.8

Learning for CRFs

To train a CRF, we minimize

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} - \log P(\mathcal{D}, \beta) = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} - \log P(y_i | x_i, \beta)$$

$$P(y|x, \beta) = \frac{\exp(\Phi(x, y)^T \beta)}{\sum_{y' \in Y} \exp(\Phi(x, y')^T \beta)}$$

$$\begin{aligned} \log P(y|x, \beta) &= \Phi(x, y)^T \beta - \log(\sum_{y' \in Y} \exp(\Phi(x, y')^T \beta)) \\ &= \Phi(x, y)^T \beta - \log(Z) \end{aligned}$$

Z is called the **normalizer** or **partition function**.

Learning for CRFs

For CRFs implemented with a NN library, you can just compute

$$Z = \sum_{y' \in Y} \exp(\Phi(x, y')^T \beta))$$

using the Forward algorithm, and use automatic differentiation to compute the gradient.

Learn the parameters using stochastic gradient descent, Adam, etc.

Why does this work? Viterbi with arbitrary scoring functions

- Viterbi algorithm can find the exact argmax

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}} \sum_{i=1}^{n} s_\theta(\mathbf{x}, i, y_i, y_{i-1})$$

- Forward algorithm can find the exact sum

$$\hat{\mathbf{y}} = \sum_{\mathbf{y}} \prod_{i=1}^{n} s_\theta(\mathbf{x}, i, y_i, y_{i-1})$$

- **Works for any scoring function and any semiring**
- We can use the Viterbi algorithm to make predictions for a CRF!
- We can use the Forward algorithm to compute Z for a CRF!

CRF with Linear Scoring Function

$$
\begin{aligned}
\log(P(\boldsymbol{y}|\boldsymbol{x})) &= \log(\prod_{i=0}^{n} s_\theta(\boldsymbol{x}, i, y_i, y_{i+1})) - \log(Z) \\
&= \sum_{i=0}^{n} \beta \cdot \phi(\boldsymbol{x}, i, y_i, y_{i+1}) - \log(Z) \\
&= \beta \cdot \sum_{i=0}^{n} \phi(\boldsymbol{x}, i, y_i, y_{i+1}) - \log(Z) \\
&= \beta \cdot \Phi(\boldsymbol{x}, \boldsymbol{y}) - \log(Z)
\end{aligned}
$$

$\phi(\boldsymbol{x}, i, y_i, y_{i+1})$ are **local features**

$\Phi(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=0}^{n} \phi(\boldsymbol{x}, i, y_i, y_{i+1})$ is the total feature vector

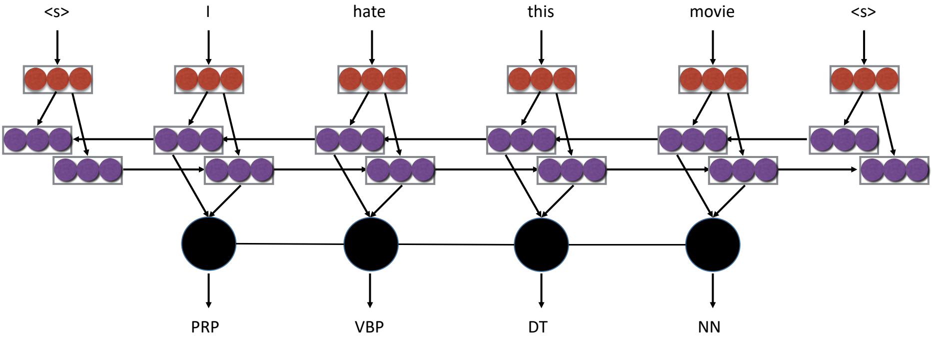
CRFs with Neural Models

Learning CRFs with a neural model:

- Compute Z(X) (the partition function) with the forward algorithm
- Compute derivative of log likelihood with automatic differentiation

Neural CRF Example

BiLSTM-CRF for Sequence Labeling



Potential Functions

- $\psi_i(y_{i-1}, y_i, X) = \exp\left(W^T T(y_{i-1}, y_i, X, i) + U^T\ S(y_i, X, i) + b_{y_{i-1},y_i}\right)$
  - Using neural features in DNN:
    $\psi_i(y_{i-1}, y_i, X) = \exp\left(W_{y_{i-1},y_i}^T F(X, i) + U_{y_i}^T\ F(X, i) + b_{y_{i-1},y_i}\right)$
    - Number of parameters: $O(|Y|^2 d_F)$
  - Simpler version:
    $\psi_i(y_{i-1}, y_i, X) = \exp\left(W_{y_{i-1},y_i} + U_{y_i}^T F(X, i) + b_{y_{i-1},y_i}\right)$
    - Number of parameters: $O(|Y|^2 + |Y| d_F)$

CRF Training & Decoding

- $P(Y|X) = \frac{\prod_{i=1}^{L} \psi_i(y_{i-1}, y_i, X)}{\sum_{Y'} \prod_{i=1}^{L} \psi_i(y'_{i-1}, y'_i, X)} = \frac{\prod_{i=1}^{L} \psi_i(y_{i-1}, y_i, X)}{Z(X)}$
- Training: computing the partition function Z(X)

$$Z(X) = \sum_{Y} \prod_{i=1}^{L} \psi_i(y_{i-1}, y_i, X)$$

- Decoding

$$y^* = argmax_Y P(Y|X)$$

Go through the output space of Y which grows exponentially with the length of the input sequence.

Viterbi Algorithm

- $\pi_t(y|\mathrm{X})$ is the partition of sequence with length equal to t and end with label y:

$$\pi_t(y|X) = \sum_{y_i,\dots,y_{t-1}} \left(\prod_{i=1}^{t-1} \psi_i(y_{i-1}, y_i, X) \right) \psi_t(y_{t-1}, y_t = y, X)$$

$$= \sum_{y_{t-1}} \psi_t(y_{t-1}, y_t = y, X) \sum_{y_i,\dots,y_{t-2}} \left(\prod_{i=1}^{t-2} \psi_i(y_{i-1}, y_i, X) \right) \psi_{t-1}(y_{t-2}, y_{t-1}, X)$$

$$= \sum_{y_{t-1}} \psi_t(y_{t-1}, y_t = y, X) \pi_{t-1}(y_{t-1}|X)$$

- Computing partition function $Z(X) = \sum_y \pi_L(y|X)$

Viterbi Algorithm

- Decoding is performed with similar dynamic programming algorithm
- Calculating gradient: $l_{ML}(X, Y; \theta) = -\log P(Y|X; \theta)$

$$\frac{\partial l_{ML}(X, Y; \theta)}{\partial \theta} = F(Y, X) - E_{P(Y|X;\theta)}[F(Y, X)]$$

  - Forward-backward algorithm (Sutton and McCallum, 2010)
    - Both $P(Y|X; \theta)$ and $F(Y, X)$ can be decomposed
    - Need to compute the marginal distribution:

$$P(y_{i-1} = y', y_i = y|X; \theta) = \frac{\alpha_{i-1}(y'|X)\psi_i(y', y, X)\beta_i(y|X)}{Z(X)}$$

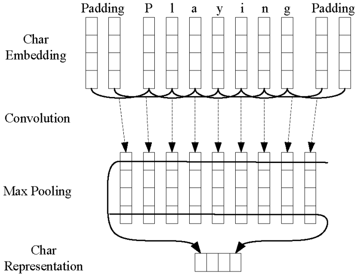
  - Not necessary if using DNN framework (auto-grad)

Case Study: BiLSTM-CNN-CRF for Sequence Labeling (Ma et al, 2016)

- Goal: Build a truly end-to-end neural model for sequence labeling task, requiring no feature engineering and data pre-processing.
- Two levels of representations
  - Character-level representation: CNN
  - Word-level representation: Bi-directional LSTM

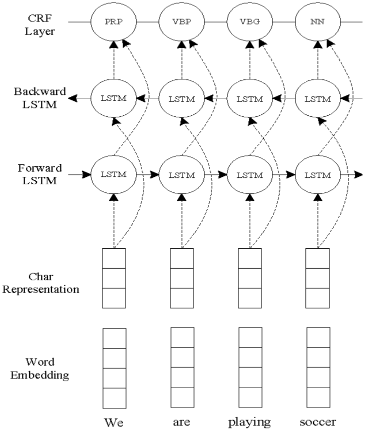
CNN for Character-level representation

- We used CNN to extract morphological information such as prefix or suffix of a word



Bi-LSTM-CNN-CRF

- We used Bi-LSTM to model word-level information.
- CRF is on top of Bi-LSTM to consider the co-relation between labels.



Training Details

- Optimization Algorithm:
  - SGD with momentum (0.9)
  - Learning rate decays with rate 0.05 after each epoch.
- Dropout Training:
  - Applying dropout to regularize the model with fixed dropout rate 0.5
- Parameter Initialization:
  - Parameters: Glorot and Bengio (2010)
  - Word Embedding: Stanford's GloVe 100-dimentional embeddings
  - Character Embedding: uniformly sampled from $[-\sqrt{\frac{3}{dim}}, +\sqrt{\frac{3}{dim}}]$, where $dim = 30$

Experiments

Model	POS		NER					
	Dev	Test	Dev			Test		
	Acc.	Acc.	Prec.	Recall	F1	Prec.	Recall	F1
BRNN	96.56	96.76	92.04	89.13	90.56	87.05	83.88	85.44
BLSTM	96.88	96.93	92.31	90.85	91.57	87.77	86.23	87.00
BLSTM-CNN	97.34	97.33	92.52	93.64	93.07	88.53	90.21	89.36
BLSTM-CNN-CRF	97.46	97.55	94.85	94.63	94.74	91.35	91.06	91.21

End

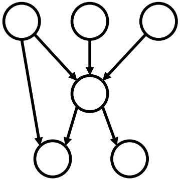
We stopped here.

Graphical Models

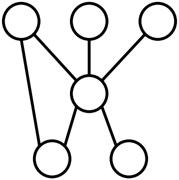
- Nodes represent random variables
- Edges represent dependence between random variables
- Usually trained to maximize joint or conditional probability, but can be trained with any loss function (ex: Max-Margin Markov Nets)

Three Types of Graphical Models

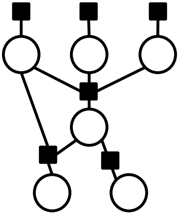
Directed Graphical Model



Undirected Graphical Model



Factor Graph

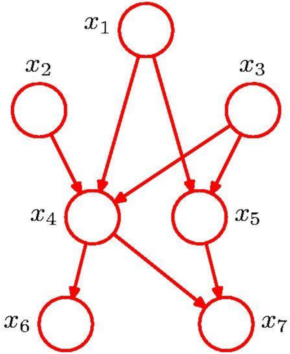


Today: Undirected Graphical Models

But first: Markov blanket (review)

Bayesian Networks

$p(x_1, \ldots, x_7)$



General Factorization

$$p(\mathbf{x}) = \prod_{k=1}^{K} p(x_k | \mathrm{pa}_k)$$

Definition: Markov Blanket

Given a node X, the **Markov blanket** for X is the minimal set of nodes that makes X conditionally independent of all the other nodes given the Markov blanket.

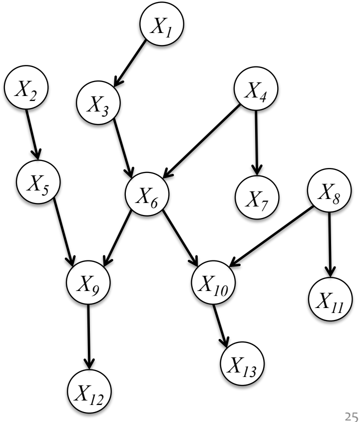
Let G be a graph, and let B be Markov blanket for X.

$$X \perp\!\!\!\perp (G - \{X\} - B) | B$$

Markov Blanket (Directed)

Def: the **co-parents** of a node are the parents of its children

Def: the **Markov Blanket** of a node in a directed graphical model is the set containing the node's parents, children, and co-parents.

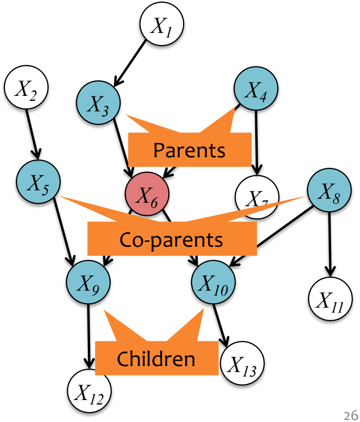


Markov Blanket (Directed)

Def: the **co-parents** of a node are the parents of its children

Def: the **Markov Blanket** of a node in a directed graphical model is the set containing the node's parents, children, and co-parents.

Example: The Markov Blanket of X_6 is $\{X_3, X_4, X_5, X_8, X_9, X_{10}\}$



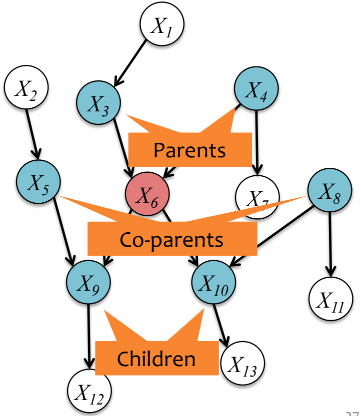
Markov Blanket (Directed)

Def: the **co-parents** of a node are the parents of its children

Def: the **Markov Blanket** of a node in a directed graphical model is the set containing the node's parents, children, and co-parents.

Theorem: a node is **conditionally independent** of every other node in the graph given its **Markov blanket**

Example: The Markov Blanket of X_6 is $\{X_3, X_4, X_5, X_8, X_9, X_{10}\}$



Undirected graphical models

- An alternative representation for joint distributions is as an **undirected graphical model**
- As in BNs, we have one node for each random variable
- Rather than CPDs, we specify (non-negative) **potential functions** over sets of variables associated with cliques C of the graph,

$$p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(\mathbf{x}_c)$$

  Z is the **partition function** and normalizes the distribution:

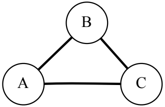
$$Z = \sum_{\hat{x}_1, \ldots, \hat{x}_n} \prod_{c \in C} \phi_c(\hat{\mathbf{x}}_c)$$

- Like CPD's, $\phi_c(\mathbf{x}_c)$ can be represented as a table, but it is *not normalized*
- Also known as **Markov random fields** (MRFs) or Markov networks

Undirected graphical models

$$p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(\mathbf{x}_c), \qquad Z = \sum_{\hat{x}_1, \ldots, \hat{x}_n} \prod_{c \in C} \phi_c(\hat{\mathbf{x}}_c)$$

Simple example (potential function on each edge encourages the variables to take the same value):



$\phi_{A,B}(a,b) =$

A \ B	0	1
0	10	1
1	1	10

$\phi_{B,C}(b,c) =$

B \ C	0	1
0	10	1
1	1	10

$\phi_{A,C}(a,c) =$

A \ C	0	1
0	10	1
1	1	10

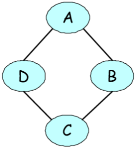
$$p(a,b,c) = \frac{1}{Z} \phi_{A,B}(a,b) \cdot \phi_{B,C}(b,c) \cdot \phi_{A,C}(a,c),$$

where

$$Z = \sum_{\hat{a}, \hat{b}, \hat{c} \in \{0,1\}^3} \phi_{A,B}(\hat{a}, \hat{b}) \cdot \phi_{B,C}(\hat{b}, \hat{c}) \cdot \phi_{A,C}(\hat{a}, \hat{c}) = 2 \cdot 1000 + 6 \cdot 10 = 2060.$$

Hair color example as a MRF

- We now have an **undirected** graph:



- The joint probability distribution is parameterized as

$$p(a, b, c, d) = \frac{1}{Z}\phi_{AB}(a, b)\phi_{BC}(b, c)\phi_{CD}(c, d)\phi_{AD}(a, d)\ \phi_A(a)\phi_B(b)\phi_C(c)\phi_D(d)$$

- **Pairwise potentials** enforce that no friend has the same hair color:

$$\phi_{AB}(a, b) = 0 \text{ if } a = b, \quad \text{and 1 otherwise}$$

- **Single-node potentials** specify an affinity for a particular hair color, e.g.

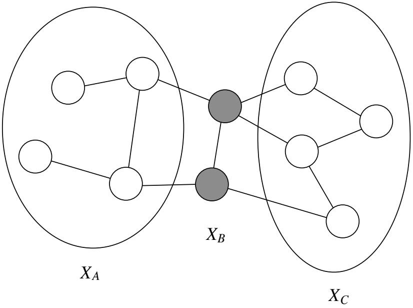
$$\phi_D(\text{“red”}) = 0.6, \quad \phi_D(\text{“blue”}) = 0.3, \quad \phi_D(\text{“green”}) = 0.1$$

The normalization Z makes the potentials **scale invariant**! Equivalent to

$$\phi_D(\text{“red”}) = 6, \quad \phi_D(\text{“blue”}) = 3, \quad \phi_D(\text{“green”}) = 1$$

Markov network structure implies conditional independencies

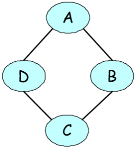
- Let G be the undirected graph where we have one edge for every pair of variables that appear together in a potential
- Conditional independence is given by **graph separation**!



- $X_{\mathbf{A}} \perp X_{\mathbf{C}} \mid X_{\mathbf{B}}$ if there is no path from $a \in \mathbf{A}$ to $c \in \mathbf{C}$ after removing all variables in $\mathbf{B}$

Example

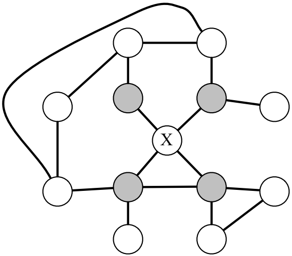
- Returning to hair color example, its undirected graphical model is:



- Since removing A and C leaves no path from D to B, we have $D \perp B \mid \{A, C\}$
- Similarly, since removing D and B leaves no path from A to C, we have $A \perp C \mid \{D, B\}$
- No other independencies implied by the graph

Markov blanket

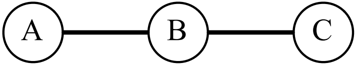
- A set $\mathbf{U}$ is a **Markov blanket** of X if $X \notin \mathbf{U}$ and if $\mathbf{U}$ is a minimal set of nodes such that $X \perp (\mathcal{X} - \{X\} - \mathbf{U}) \mid \mathbf{U}$
- In undirected graphical models, the Markov blanket of a variable is precisely its **neighbors** in the graph:



- In other words, X is independent of the rest of the nodes in the graph given its immediate neighbors

Proof of independence through separation

- We will show that $A \perp C \mid B$ for the following distribution:



$$p(a, b, c) = \frac{1}{Z}\phi_{AB}(a, b)\phi_{BC}(b, c)$$

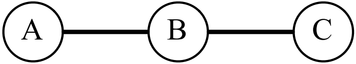
- First, we show that $p(a \mid b)$ can be computed using only $\phi_{AB}(a, b)$:

$$\begin{aligned}
p(a \mid b) &= \frac{p(a, b)}{p(b)} \\
&= \frac{\frac{1}{Z}\sum_{\hat{c}} \phi_{AB}(a, b)\phi_{BC}(b, \hat{c})}{\frac{1}{Z}\sum_{\hat{a},\hat{c}} \phi_{AB}(\hat{a}, b)\phi_{BC}(b, \hat{c})} \\
&= \frac{\phi_{AB}(a, b)\sum_{\hat{c}} \phi_{BC}(b, \hat{c})}{\sum_{\hat{a}} \phi_{AB}(\hat{a}, b)\sum_{\hat{c}} \phi_{BC}(b, \hat{c})} = \frac{\phi_{AB}(a, b)}{\sum_{\hat{a}} \phi_{AB}(\hat{a}, b)}.
\end{aligned}$$

- More generally, the probability of a variable conditioned on its Markov blanket depends *only* on potentials involving that node

Proof of independence through separation

- We will show that $A \perp C \mid B$ for the following distribution:



$$p(a,b,c) = \frac{1}{Z}\phi_{AB}(a,b)\phi_{BC}(b,c)$$

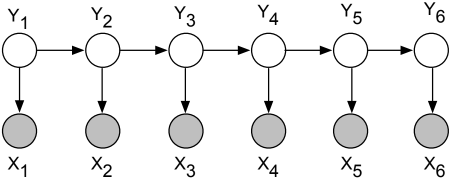
Proof.

$$\begin{aligned} p(a,c \mid b) = \frac{p(a,c,b)}{\sum_{\hat{a},\hat{c}} p(\hat{a},b,\hat{c})} &= \frac{\phi_{AB}(a,b)\phi_{BC}(b,c)}{\sum_{\hat{a},\hat{c}} \phi_{AB}(\hat{a},b)\phi_{BC}(b,\hat{c})} \\ &= \frac{\phi_{AB}(a,b)\phi_{BC}(b,c)}{\sum_{\hat{a}} \phi_{AB}(\hat{a},b)\sum_{\hat{c}} \phi_{BC}(b,\hat{c})} \\ &= p(a \mid b)p(c \mid b) \end{aligned}$$

$\square$

Converting BNs to Markov networks

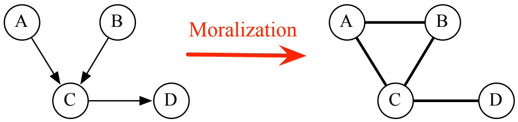
What is the equivalent Markov network for a hidden Markov model?



Many inference algorithms are more conveniently given for undirected models – this shows how they can be applied to Bayesian networks

Moralization of Bayesian networks

- Procedure for converting a Bayesian network into a Markov network
- The **moral graph** $\mathcal{M}[G]$ of a BN $G = (V, E)$ is an undirected graph over V that contains an undirected edge between X_i and X_j if
  1. there is a directed edge between them (in either direction)
  2. X_i and X_j are both parents of the same node

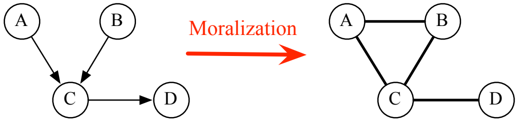


(term historically arose from the idea of "marrying the parents" of the node)

- The addition of the moralizing edges leads to the loss of some independence information, e.g., $A \rightarrow C \leftarrow B$, where $A \perp B$ is lost

Converting BNs to Markov networks

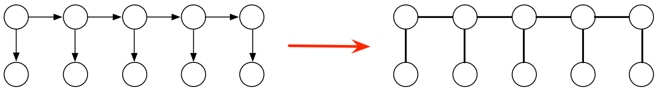
1. Moralize the directed graph to obtain the undirected graphical model:



2. Introduce one potential function for each CPD:

$$\phi_i(x_i, \mathbf{x}_{pa(i)}) = p(x_i \mid \mathbf{x}_{pa(i)})$$

- So, converting a hidden Markov model to a Markov network is simple:



- For variables having > 1 parent, factor graph notation is useful

Conditional random fields (CRFs)

- **Conditional random fields** are undirected graphical models of conditional distributions $p(\mathbf{Y} \mid \mathbf{X})$
  - $\mathbf{Y}$ is a set of **target variables**
  - $\mathbf{X}$ is a set of **observed variables**
- We typically show the graphical model using just the $\mathbf{Y}$ variables
- Potentials are a function of $\mathbf{X}$ and $\mathbf{Y}$

Formal definition

- A CRF is a Markov network on variables $\mathbf{X} \cup \mathbf{Y}$, which specifies the conditional distribution

$$P(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{c \in C} \phi_c(\mathbf{x}_c, \mathbf{y}_c)$$

  with partition function

$$Z(\mathbf{x}) = \sum_{\hat{\mathbf{y}}} \prod_{c \in C} \phi_c(\mathbf{x}_c, \hat{\mathbf{y}}_c).$$

- As before, two variables in the graph are connected with an undirected edge if they appear together in the scope of some factor
- The only difference with a standard Markov network is the normalization term – before marginalized over $\mathbf{X}$ and $\mathbf{Y}$, now only over $\mathbf{Y}$

Parameterization of CRFs

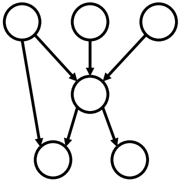
- Factors may depend on a large number of variables
- We typically parameterize each factor as a log-linear function,

$$\phi_c(\mathbf{x}_c, \mathbf{y}_c) = \exp\{\mathbf{w} \cdot \mathbf{f}_c(\mathbf{x}_c, \mathbf{y}_c)\}$$

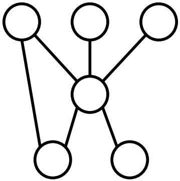
- $\mathbf{f}_c(\mathbf{x}_c, \mathbf{y}_c)$ is a feature vector
- $\mathbf{w}$ is a weight vector which is typically learned – we will discuss this extensively in later lectures

Three Types of Graphical Models

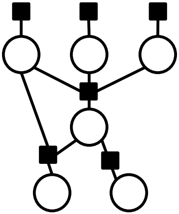
Directed Graphical Model



Undirected Graphical Model



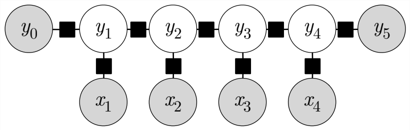
Factor Graph



Factor graphs

Factor Graphs

$$\hat{\mathbf{t}} = \underset{\mathbf{t}}{\operatorname{argmax}} \sum_{i=1}^{n} \log(p(w_i|t_i)) + \log(p(t_i|t_{i-1}))$$



- Like Bayesian networks, **factor graphs** are a graphical model
- Each box represents a **local factor**, which is a function that depends on the R.V.s it is connected to
- The total score is the sum (or product) of the factors