Lecture 2: Probability Distributions on Graphs

Note Title 1/17/2010

Graphs enable us to represent probability distributions compactly by exploiting the dependencies between variables.

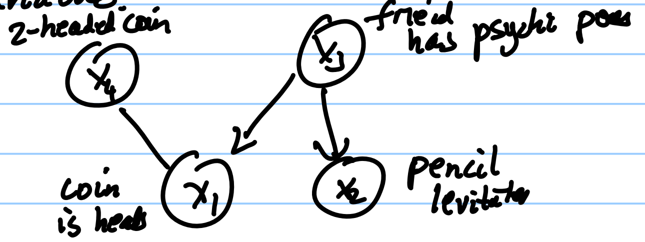
- This is helpful for understanding the structure of the data described by the distribution - enables transferring distributions from one domain to another.
- It also makes it easier to perform inference and to do learning

Example : $P(X_1, X_2, X_3, X_4)$ suppose $X_i \in \{0, 1\}$

this distribution has $2^4 - 1 = 15$ entries

A general distribution $P(X_1 ... , X_N)$ has $2^N - 1$ entries, which are far too many to learn unless we have an enormous amount of data

But suppose we know which variables directly influence other variables.



Model the situation

friend claims psychic powers which can predict coin toss is head

→ but coin toss can also be explained by 2-headed coin

→ an additional test - can friend levitate pencil - if successful can "explain away" the coin toss, and justify the 2-headed coin explanation

$P(X_1, X_2, X_3, X_4)$

$= P(X_1 | X_4, X_3) P(X_2 | X_3) P(X_4) P(X_3)$

- specified by fewer ($4+2+1+1 = 8$ numbers)

Knowing this structure gives:

(i) knowledge about the problem — explaining away

(ii) fewer numbers needed to describe distribution — less data needed to learn model

(iii) faster inference.

(2) Inference —

— normally takes $2^3=8$ operations.

to compute $P(x_1=1) = \sum_{x_2}\sum_{x_3}\sum_{x_4} P(x_1=1, x_2, x_3, x_4)$

exploiting graph structure.

$$= \sum_{x_2}\sum_{x_3}\sum_{x_4} P(x_1=1|x_3,x_4)P(x_2|x_3)P(x_3)P(x_4)$$

$$= \sum_{x_3}\sum_{x_4} P(x_1=1|x_3,x_4)P(x_3)P(x_4)\underbrace{\sum_{x_2} P(x_2|x_3)}_{1}$$

$$= \sum_{x_3}\sum_{x_4} P(x_1=1|x_3,x_4)P(x_3)P(x_4)$$

only $2^2=4$ operations required

General Result if the graph structure has no closed loops, then we can use dynamic programming to compute any property of interest (e.g. $P(x_1)$) in polynomial time in no. of nodes and no. of states.

E.g. $x_1 \to x_2 \to x_3 \to x_4$ rapid (polynomial) inference.

Intuition for dynamic programming — exploit the "linear structure" to break problem down into subcomponents.
E.g. to find the shortest path from Los Angeles to Boston which goes via Chicago, it is necessary only to find the shortest path from LA to Chicago and from Chicago to Boston independently.

More Techically — suppose we want to minimize
$\varphi(x_1 \dots x_N) = \varphi_{12}(x_1,x_2) + \varphi_{23}(x_2,x_3) + \dots + \varphi_{N-1N}(x_{N-1},x_N)$

$x_1 - x_2 - \cdot - \cdot - \cdot - x_N$. $x_i \in \{1,\dots,k\}$

Forward Pass of DP:

For each x_2, compute $h_2(x_2) = \min_{x_1} \varphi_{12}(x_1,x_2)$ shortest cost to x_2

For each x_3, compute $h_3(x_3) = \min_{x_1 x_2}\{\varphi_{12}(x_1,x_2)+\varphi_{23}(x_2,x_3)\}$,

the clever bit $\longrightarrow$ $h_3(x_3) = \min_{x_2}\{h_2(x_2) + \varphi_{23}(x_2,x_3)\}$,

In general, $h_m(x_m) = \min_{x_{m-1}}\{h_{m-1}(x_{m-1}) + \varphi_{m-1m}(x_{m-1},x_m)\}$,

So can compute $h_N(x_N)$ is poly time $\sim Nk^2$ operators

(3) Backward Pass of DP.

solve $\hat{x}_N = \arg\min h_N(x_N)$ $\Rightarrow$ $h_N(\hat{x}_N)$ is smallest cost of $\varphi(x_1 \ldots x_N)$

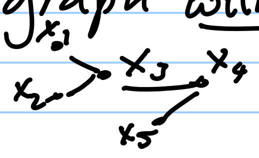
$\hat{x}_{N-1} = \arg\min \{ h_{N-1}(x_{N-1}) + \varphi_{N,N-1}(\hat{x}_N) \}$

$\vdots$

recovers the states $\hat{x}_N, \hat{x}_{N-1}, \ldots \hat{x}_1$ which give the shortest cost.

Advantage: efficiency, $\sim k^2 N$ operators – instead of considering the total k^N possible states of $\varphi(x_1 \ldots x_N)$

Note: (1) DP can be applied to any graph without closs loops $\rightarrow$ e.g. extend to



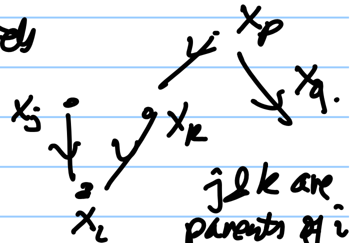
(2) DP can be modified to compute other properties of interest.

Lauritzen & Spiegelhalter. (3) DP can be extended to graphs with closed loops to give the junction tree algorithm – this includes a procedure for converting a graph to a tree. But for many graphs the tree is so large that computation on it is impractical.

Back to graphs

In general, Directed Graphs / Bayes Nets

$p(x_1 \ldots x_N) = \prod_i p(x_i | Pa(x_i))$

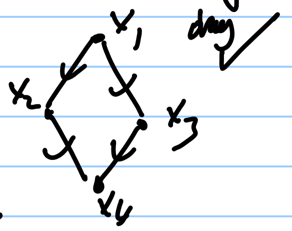


$Pa(x_i)$ are the parents of x_i, the nodes which have directed arcs directly into x_i

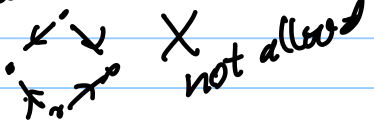
j & k are parents of i
p & q are not.

DAG's capture some of the causal structure of the variables in the problem.

This can include closed loops like this. $\longrightarrow$



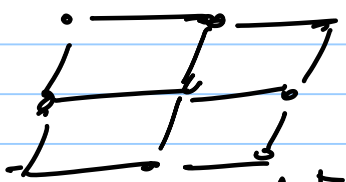
provided the arrows are consistent.



(4) Undirected Graphs

$G = (V, \mathcal{E})$ — here the edges are not directed

$\underset{\text{vertices}}{V}$, $\underset{\text{edges}}{\mathcal{E}}$.



$$P(\underline{x}) = \frac{1}{Z}\prod_{ij\in\mathcal{E}} \psi_{ij}(x_i,x_j) \prod_{i\in V}\psi_i(x_i) \rightarrow \underline{\text{potentials}}.$$

Z: normalization constant.

potential

often write $\psi_{ij}(x_i,x_j) = e^{-\phi_{ij}(x_i,x_j)}$, -ve

$$P(\underline{x}) = \frac{1}{Z} e^{-\{\sum_{ij\in\mathcal{E}}\phi_{ij}(x_i,x_j)+\sum_i \phi_i(x_i)\}}$$

Undirected Graphs include Direct Graphs as a special case — just drop the arrows.

E.g. you can convert $A \rightarrow B \rightarrow C$ to $A - B - C$

to $A \leftarrow B \leftarrow C$.

For real causality — intervention by graph pruning can distinguish between them $A \rightarrow B \rightarrow C$ and $A \leftarrow B \leftarrow C$.

(1) $P(x_A,x_B,x_C) = \frac{1}{Z} e^{-\{\psi_A(x_A)+\psi_{AB}(x_A,x_B)+\psi_{BC}(x_B,x_C)\}}$

(2) $P(x_C|x_B)P(x_B|x_A)P(x_A)$ directed, or $P(x_A|x_B)P(x_B|x_C)P(x_C)$

Can translate from (1) to (2) using dynamic programming

Translate from (1) to (2) set $\psi_A(x_A) = -\log P(x_A)$

$\psi_{AB}(x_A,x_B) = -\log P(x_B|x_A)$, $\psi_{BC}(x_B,x_C) = -\log P(x_C|x_B)$

Latent/Hidden Variables

For both Directed and Undirected graphs some variables can be observed directly and so are 'observable', while the others are 'latent', 'hidden'.

Many, 'neural network' models can be expressed using hidden variables. E.g. Boltzmann Machine.

$$p(\underline{y},\underline{x}|\omega) = \frac{1}{Z} e^{-E[\underline{y},\underline{x}|\omega]}$$

x_i — observed
y_j — hidden.

$$E[\underline{y},\underline{x},\omega] = \sum_{ij}\omega^o_{ij}x_iy_j + \sum_{ij}\omega^h_{ij}y_iy_j$$

(5.) Graphical Models can also have variable topology and no. of nodes.

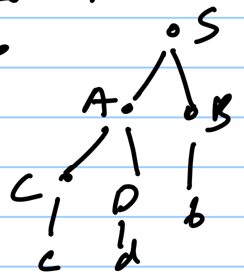
E.g. SCFG Stochastic Context Free Grammar

Production Rules: $A \to (B, C)$ A, B, C, .. Non-Terminal Nodes

$A \to a$ a, b, c terminal node

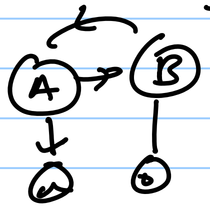
assign probability to rules.

root node



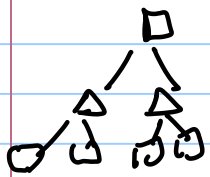
— probability distribution over the structure (see later in the course)

Hidden Markov Models:



→ states A, B can produce outputs
→ probabilities for transitions between states and producing outputs.

AND/OR Graphs:



□ AND NODE
△ OR NODES

The OR nodes acts as 'switch variables'
Graph topology changes when we select the switch

For Brief Description of the models → see Griffiths & Yuille handout
→ or wait for descriptions later in the course

Inference Algorithms

There are a range of inference algorithms that we will describe in the next few lectures.

Stochastic Sampling
Dynamic Programming
Steepest Descent.
Free Energy Methods
Graph Cuts.

These algorithms will exploit the graph structure

(6) <u>What</u> do we want to compute?

← data.

<u>Suppose</u> we have $p(x_1 \dots x_N | \text{data})$

we may want to compute the marginal distribution

$$p(x_i|\text{data}) = \sum_{x_j \neq i} p(x_1 \dots x_N|\text{data}).$$

or estimate $\hat{\underline{x}} = \cancel{\arg}\max_{\underline{x}} p(\underline{x}|\text{data})$

If there are hidden variables $\underline{y}$, we may want to compute $p(\underline{x}|\text{data}) = \sum_{\underline{y}} p(\underline{x},\underline{y}|\text{data})$

or, compute $\hat{\underline{x}} = \text{ARGMAX}\, p(\underline{x}|\text{data})$, knowing $P(\underline{x},\underline{y}|\text{data})$

→ requires the EM algorithm.

<u>Also</u>, we may want to estimate parameters of the model:

→ e.g. Boltzmann Machine $P(\underline{x},\underline{y}|\underline{\omega}) = e^{\sum_{ij} \omega_{ij}^{o} x_i y_j + \sum_{ij} \omega_{ij}^{s} y_i y_j}$

<u>estimate</u>: the $\underline{\omega}$ from a series of observations $\underline{x}^\mu$

<u>maximize</u> w.r.t. $\hat{\underline{\omega}}$ $\prod_\mu P(\underline{x}^\mu|\underline{\omega}) = \prod_\mu \sum_{\underline{y}} P(\underline{x}^\mu, \underline{y}|\underline{\omega})$

something called inference — but in the course we will call it learning.

<u>easier</u> if no hidden variables, e.g. estimate the probability that a coin yields "heads" from a set of sample coin tosses

<u>Note</u>: most learning assumes that the form of the model is known — eg the graph structure. — it is usually much harder to learn the structure. But we will give some examples later in the course

<u>Finally</u>, model selection and Occam's factor.

Consider two models $P(\underline{x},\underline{h}|M_1)$ $P(\underline{x},\tilde{\underline{h}}|M_2)$ $\underline{h}, \tilde{\underline{h}}$ hidden variables

the prob of data $P(\underline{x}) = \sum_{\underline{h}} P(\underline{x},\underline{h}|M_1)$ for M_1

$= \sum_{\tilde{\underline{h}}} P(\underline{x},\tilde{\underline{h}}|M_2)$ for M_2.

This penalizes complex models and favours more precise models which fit the data.