Probability Distributions on Graphs

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 distributions – enables transferring distributions from one domain to another
- It also makes it easier to perform inference and to do learning

Example: $P(x_1, x_2, x_3, x_4)$ Let $x_i \in \{0,1\}$

This distribution has 2^4-1 =15 entries

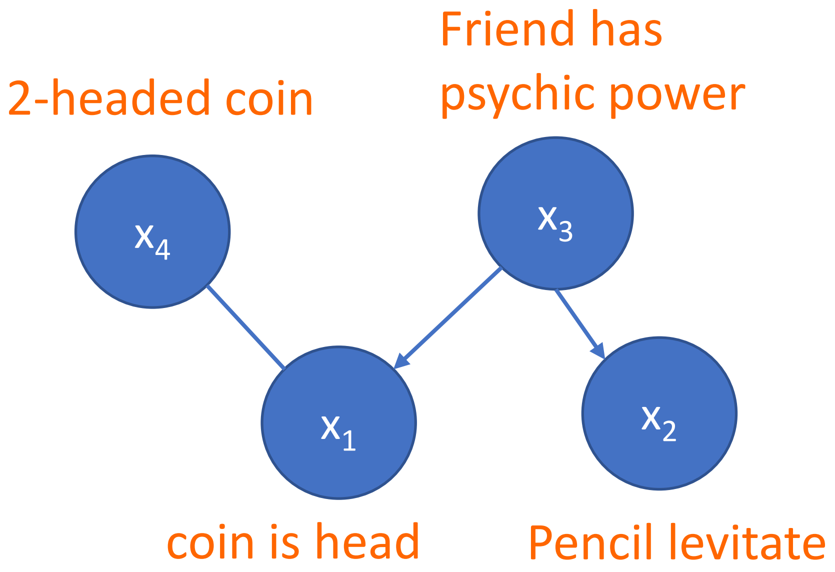
A general distribution $P(x_1, \ldots, x_N)$ has 2^N-1 entries, which are far too many to learn unless we have an enormous amount of data

But suppose we know which variables directly influence other variables

Probability Distributions on Graphs

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



$$P(x_1, x_2, x_3, x_4)$$
$$= P(x_1 \mid x_4, x_3) P(x_2 \mid x_3) P(x_4) P(x_3)$$

~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 distributions – less data needed to learn model

(iii) Faster inference

JOHNS HOPKINS UNIVERSITY

Inference

to compute $P(x_1=1)=\sum_{x_2}\sum_{x_3}\sum_{x_4}P(x_1=1,x_2,x_3,x_4)$ normally takes 2^3=8 operations

exploiting graph structure

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

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

$$=\underbrace{\sum_{x_3}\sum_{x_4}P(x_1=1|x_3,x_4)P(x_3)P(x_4)}_{\text{Only } 2^2=4 \text{ operations requires}}$$

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

E.G. 

Rapid(polynomial) inference

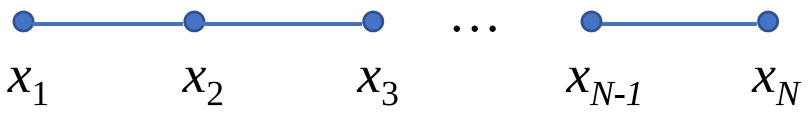
Intuition for dynamic programming

Exploit the “linear structure” to break problems down into subcomponents

E.G. to find the shortest path from Los Angeles to Boston which gives via Chicago, it is necessary only to find the shortest path from LA to Chicago and from Chicago to Boston

More Technically, suppose we want to minimize

$$\varphi(x_1,\ldots,x_N) = \varphi_{12}(x_1,x_2) + \varphi_{23}(x_2,x_3) + \cdots + \varphi_{N-1N}(x_{N-1},x_N)$$



$$x_i \in \{1,\ldots,k\}$$

Intuition for dynamic programming

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

Forward 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→ $h_3(x_3) = \min_{x_1, x_2} \{h_2(x_2) + \varphi_{23}(x_2, x_3)\}$

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 polynomial time operators

Backward Pass of DP

Solve $\hat{x}_N = \arg\min h_N(x_N)$

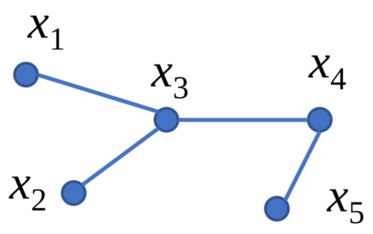
$$\hat{x}_{N-1} = \arg\min \left\{ h_{N-1}(x_{N-1}) + \varphi_{N,N-1}(\hat{x}_N) \right\}$$

$$\vdots$$

Recovers the states x_N, $x_{N\text{-}1}$, …, x_1 which give the shortest cost

Advantage: efficiently, ~k^2N operations – instead of considering the total k^N possible states of $\varphi(x_1, x_2, \ldots, x_N)$

Note: (1) DP can be applied to any graph without closed loops → e.g. extended to



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

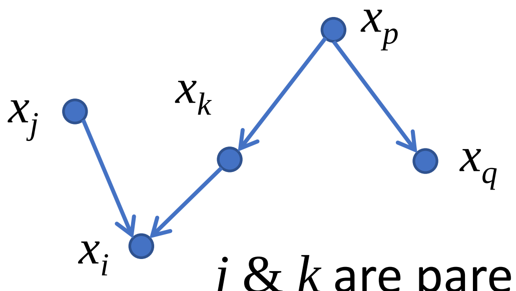
(3) DP can be extended to graphs with closed loops to give the junction free algorithm – this includes a procedure for converting a graph to a tree. But for many graphs the tree is so large that computation on it is impractical (Lauritzen & Spiegelhalter)

Back to graphs

In general, Directed Graphs / Bayes Nets

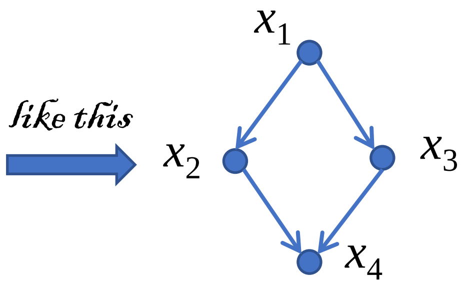
$$P(x_1,\ldots,x_N) = \prod_i P\left(x_i \mid P_a(x_i)\right)$$

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

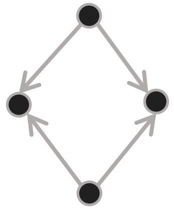


j & k are parents of i
p & q are not

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



This can include closed loops provided the arrows are consistent

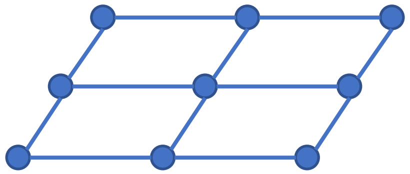


not allowed

Undirected Graphs

$G = (V, E)$:here the edges are not directed

V: vertices, E: Edges



$$P(\mathbf{x}) = \frac{1}{\boxed{Z}} \prod_{(i,j)\in E} \psi_{ij}(x_i, x_j) \prod_{i\in V} \psi_i(x_i)$$

normalization constant

potentials

Often write $\psi_{ij}(x_i, x_j) = e^{-\phi_{ij}(x_i, x_j)}$ $\Rightarrow$ $P(\mathbf{x}) = \frac{1}{Z} e^{-\left\{\sum_{(i,j)\in E} \phi_{ij}(x_i, x_j) + \sum_i \phi_i(x_i)\right\}}$

Undirected Graphs include Direct Graphs as a special case – just drop the arrow

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 $A \leftarrow B \leftarrow C$

Undirected Graphs

(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) $P(x_C \mid x_B)P(x_B \mid x_A)P(x_A)$ directed, or $P(x_A \mid x_B)P(x_B \mid x_C)P(x_C)$

➡ can translate from (1) to (2) using dynamic programming

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

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

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 $P(\mathbf{y}, \mathbf{x}, \mathbf{w}) = \frac{1}{Z} e^{-E[\mathbf{y},\mathbf{x},\mathbf{w}]}$

$$E[\mathbf{y}, \mathbf{x}, \mathbf{w}] = \sum_{i,j} w_{ij}^0 x_i y_j + \sum_{i,j} w_{ij}^h y_i y_j$$

x_i: observed
y_i: hidden

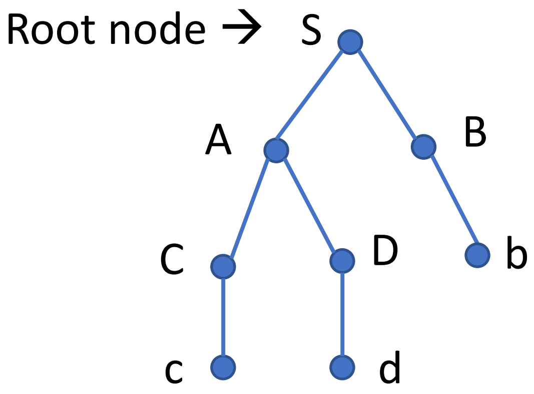
Graphical Models can also have variable topology and no. of nodes

E.G. SCFG (Stochastic Context-Free Grammar)

Prediction Rules: A → (B, C)
A → a

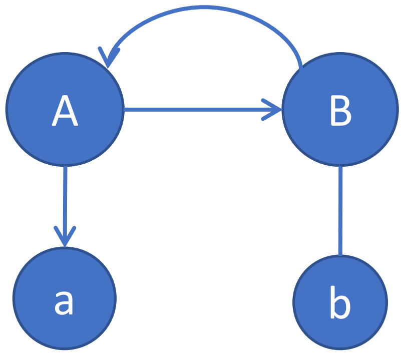
A, B, C, … : non-terminal nodes
a, b, c : terminal nodes

Assign probability to rules



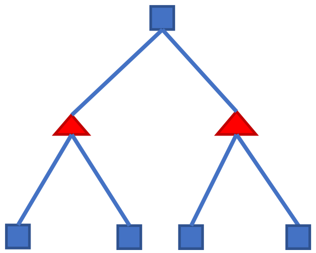
Probability distribution over the structure (see later in the course)

Hidden Markov Models:



- ➔ States A, B can product output
- ➔ Probabilities for transitioning between states and producing outputs

AND/OR Graph



■ AND Node

▲ OR Node

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 the description 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 Programing
Steepest Decent
Free Energy Methods
Graph Cuts

These algorithms will exploit the graph structure

What do we want to compute?

Suppose we have $P(x_1, \ldots, x_N \mid \text{data})$

We may want to compute the marginal distributions

$$P(x_i \mid \text{data}) = \sum_{j \neq i} P(x_1, \ldots, x_j, \ldots, x_N \mid \text{data})$$

or estimate $\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} P(\mathbf{x} \mid \text{data})$

If there are hidden variable y, we may want to compute

$$P(\mathbf{x} \mid \text{data}) = \sum_{\mathbf{y}} P(\mathbf{x}, \mathbf{y} \mid \text{data})$$

or compute $\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} P(\mathbf{x} \mid \text{data})$ knowing $P(\mathbf{x}, \mathbf{y} \mid \text{data})$

➔ Require the EM algorithm

Also, we may want to estimate parameters of the model:

e.g. Boltzmann machine

Boltzmann machine $P(\mathbf{x},\mathbf{y}\,|\,\mathbf{w}) = e^{\sum_{i,j} w^0_{ij} x_i y_j + \sum_{i,j} w_{ij} y_i y_j}$

Estimate: the $\mathbf{w}$ form a series of absolute $\mathbf{x}^\mu$

$$\max_{\mathbf{w}} \prod_\mu P(\mathbf{x}^\mu\,|\,\mathbf{w}) = \prod_\mu \sum_{\mathbf{y}} P(\mathbf{x}^\mu, \mathbf{y}\,|\,\mathbf{w})$$

Sometimes called inference – but in this course, we will call it learning

Easier if no hidden variables, e.g. estimate the probability that a coin yields "heads" from a set of sample coin tosses

Note: most learning assumes that the form of the model is known – e.g. the graph structure – it is usually mech harder to learn the structure. But we will give some examples later in the course

Finally model selected and occam's(?) factor (?)

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

The probability of data $P(\mathbf{x}) = \sum_{\mathbf{h}} P(\mathbf{x},\mathbf{h}\,|\,M_1)$ for M_1

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

This penalizes(?) complex models and harm(?) more precise models which fit the data