Lecture 1:

Probabilistic Graphical Models

Sam Roweis

Monday July 24, 2006
Machine Learning Summer School, Taiwan

Building Intelligent Computers

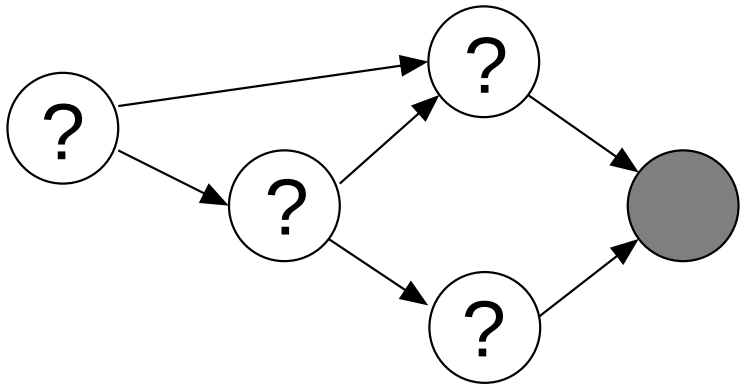
- We want intelligent, adaptive, robust behaviour in computers.

Picture:  $\Rightarrow$ Name: Sam Roweis

- Often hand programming not possible.
- Solution? Get the computer to program itself, by showing it examples of the behaviour we want!
  This is the *learning* approach to AI.
- Really, we write the structure of the program and the computer tunes many internal parameters.

Uncertainty and Artificial Intelligence (UAI)

- Statistical Models are "Probabilistic Databases"
  - traditional DB technology cannot answer queries about items that were never loaded into the dataset; probabilistic methods can:
  - make decisions given partial information about the world
  - account for noisy sensors or actuators
  - explain phenomena not part of our models
  - describe inherently stochastic behaviour in the world
- Automatic System Building
  - old expert systems needed hand coding of knowledge and of output semantics
  - learning automatically constructs rules and supports all types of queries



Applications of Probabilistic Learning

- Automatic speech recognition & speaker verification
- Printed and handwritten text parsing
- Face location and identification
- Tracking/separating objects in video
- Search and recommendation (e.g. `google`, `amazon`)
- Financial prediction, fraud detection (e.g. credit cards)
- Insurance premium prediction, product pricing
- Medical diagnosis/image analysis (e.g. pneumonia, pap smears)
- Game playing (e.g. backgammon)
- Scientific analysis/data visualization (e.g. galaxy classification)
- Analysis/control of complex systems (e.g. freeway traffic, industrial manufacturing plants, space shuttle)
- Troubleshooting and fault correction

Canonical Tasks

- *Supervised Learning*: given examples of inputs and corresponding desired outputs, predict outputs on future inputs.
  Ex: classification, regression, time series prediction
- *Unsupervised Learning*: given only inputs, automatically discover representations, features, structure, etc.
  Ex: clustering, outlier detection, compression
- *Rule Learning*: given multiple measurements, discover very common joint settings of subsets of measurements.
- *Reinforcement Learning*: given sequences of inputs, actions from a fixed set, and scalar rewards/punishments, learn to select action sequences in a way that maximizes expected reward.
  [Last two will not be covered in these lectures.]

Representation

- Key issue: how do we represent information about the world? (e.g. for an image, do we just list pixel values in some order?)

 $\rightarrow$ 127,254,3,18,...

- We must pick a way of numerically representing things that exploits regularities or structure in the data.
- To do this, we will rely on probability and statistics, and in particular on *random variables*.
- A *random variable* is like a variable in a computer program that represents a certain quantity, but its value changes depending on which data our program is looking at. The value a random variable is often unknown/uncertain, so we use probabilities.

Using random variables to represent the world

- We will use mathematical random variables to encode everything we know about the task: inputs, outputs and internal states.
- Random variables may be *discrete/categorical* or *continuous/vector*.
  Discrete quantities take on one of a fixed set of values,
  e.g. $\{$`0,1`$\}$, $\{$`email,spam`$\}$, $\{$`sunny,overcast,raining`$\}$.
  Continuous quantities take on real values.
  e.g. `temp=12.2, income=38231, blood-pressure=58.9`
- Generally have repeated measurements of same quantities.
  Convention: $i, j, \ldots$ indexes components/variables/dimensions;
  $n, m, \ldots$ indexes cases/records, x are inputs, y are outputs.
  – x_i^n is the value of the i^{th} input variable on the n^{th} case
  – y_j^m is the value of the j^{th} output variable on the m^{th} case
  $\mathbf{x}_n$ is a vector of all inputs for the n^{th} case
  $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n, \ldots, \mathbf{x}_N\}$ are all the inputs

Structure of Learning Machines

- Given some inputs, expressed in our representation, how do we calculate something about them (e.g. this is Sam's face)?
- Our computer program uses a *mathematical function* $z = f(x)$
  x is the representation of our input (e.g. face)
  z is the representation of our output (e.g. Sam)
- Hypothesis Space and Parameters:
  We don't just make up functions out of thin air. We select them from a carefully specified set, known as our *hypothesis space*.
- Generally this space is indexed by a set of *parameters* θ which are knobs we can turn to create different machines:
  $\mathcal{H} : \{f(\mathbf{z}|\mathbf{x}, \theta)\}$
- Hardest part of doing probabilistic learning is deciding how to represent inputs/outputs and how to select hypothesis spaces.

Loss Functions for Tuning Parameters

- Let inputs=$\mathbf{X}$, correct answers=$\mathbf{Y}$, outputs of our machine=$\mathbf{Z}$.
- Once we select a representation and hypothesis space, how do we set our parameters θ?
- We need to quantify what it means to do well or poorly on a task.
- We can do this by defining a loss function $L(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ (or just $L(\mathbf{X}, \mathbf{Z})$ in unsupervised case).
- Examples:

  Classification: $z_n(\mathbf{x}_n)$ is predicted class. $\quad L = \sum_n [y_n \neq z_n(\mathbf{x}_n)]$

  Regression: $\mathbf{z}_n(\mathbf{x}_n)$ is predicted output. $\quad L = \sum_n \|\mathbf{y}_n - \mathbf{z}_n(\mathbf{x}_n)\|^2$

  Clustering: $\mathbf{z}_c$ is mean of all cases assigned to cluster c.

  $$L = \sum_n \min_c \|\mathbf{x}_n - \mathbf{z}_c\|^2$$

- Now set parameters to minimize average loss function.

Training vs. Testing

- *Training data*: the $\mathbf{X}, \mathbf{Y}$ we are given.
  Testing data: the $\mathbf{X}, \mathbf{Y}$ we will see in future.
- *Training error*: the average value of loss on the training data.
  Test error: the average value of loss on the test data.
- What is our real goal? To do well on the data we have seen already? Usually not. We already have the answers for that data. We want to perform well on *future unseen data*. So ideally we would like to minimize the test error. How to do this if we don't have test data?
- Probabilistic framework to the rescue!

Sampling Assumption

- Imagine that our data is created randomly, from a joint probability distribution $p(\mathbf{x}, \mathbf{y})$ which we don't know.
- We are given a finite (possibly noisy) training sample: $\{\mathbf{x}_1, \mathbf{y}_1, \ldots, \mathbf{x}_n, \mathbf{y}_n, \ldots, \mathbf{x}_N \mathbf{y}_N\}$ with members n generated *independently and identically distributed* (iid).
- Looking only at the training data, we construct a machine that generates outputs $\mathbf{z}$ given inputs. (Possibly by trying to build machines with small training error.)
- Now a new sample is drawn from the same distribution as the training sample.
- We run our machine on the new sample and evaluate the loss; this is the test error.
- Central question: by looking at the machine, the training data and the training error, what if anything can be said about test error?

Generalization and Overfitting

- Crucial concepts: *generalization*, *capacity*, *overfitting*.
- What's the danger in the above setup? That we will do well on training data but poorly on test data. This is called *overfitting*.
- Example: just memorize training data and give random outputs on all other data.
- Key idea: you can't learn anything about the world without making some assumptions.
  (Although you can memorize what you have seen).
- Both representation and hypothesis class (model choice) represent assumptions we make.
- The ability to achieve small loss on test data is *generalization*.

Capacity: Complexity of Hypothesis Space

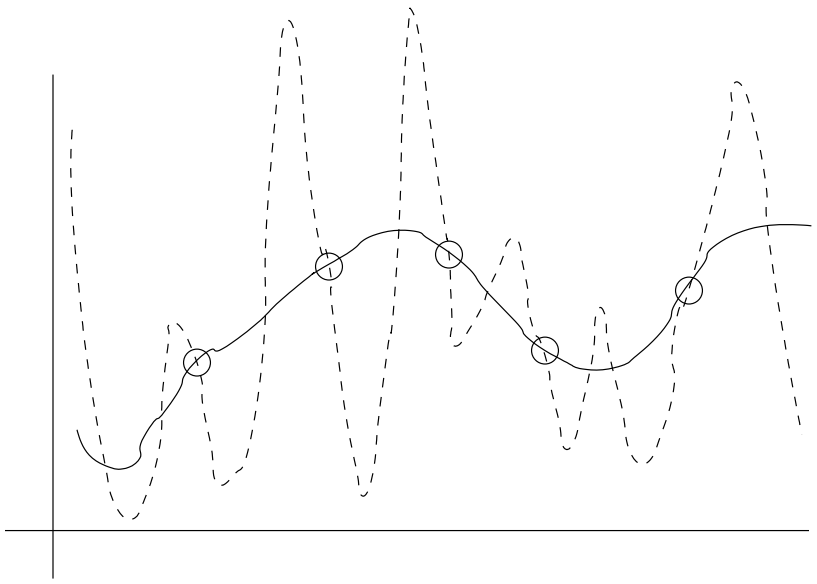
- Learning == Search in Hypothesis Space
- Inductive Learning Hypothesis: Generalization is possible.
  If a machine performs well on most training data AND it is not too complex, it will probably do well on similar test data.
- Amazing fact: in many cases this can actually be proven, using *learning theory*. If our hypothesis space is not too complicated/flexible (has a low *capacity* in some formal sense), and if our training set is large enough then we can bound the probability of performing much worse on test data than on training data.
- For now, control capacity by adding a penalty to the loss:

$$\Phi(\mathbf{X}, \theta) = L(\mathbf{X}|\theta) + P(\theta)$$

- This says that it is good to fit the data well (get low training loss) but it is also good to bias ourselves towards simpler models to avoid overfitting.

Inductive Bias

- The converse of the Inductive Learning Hypothesis is that generalization only possible if we make some assumptions, or introduce some priors. We need an *Inductive Bias*.
- No Free Lunch Theorems: an unbiased learner can never generalize.
- Consider: arbitrarily wiggly functions or random truth tables or non-smooth distributions.



0 0 0	0
0 0 1	?
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	?
1 1 0	1
1 1 1	?

Probabilistic Approach

- Given the above setup, we can think of learning as estimation of joint probability density functions given samples from the functions.
- Classification and Regression: conditional density estimation $p(\mathbf{y}|\mathbf{x})$
- Unsupervised Learning: density estimation $p(\mathbf{x})$
- *The central object of interest is the joint distribution and the main difficulty is compactly representing it and robustly learning its shape given noisy samples.*
- *Our inductive bias is expresses as prior assumptions about these joint distributions.*
- *The main computations we will need to do during the operation of our algorithms are to efficiently calculate marginal and conditional distributions from our compactly represented joint model.*

Joint Probabilities

- Goal 1: represent a joint distribution $\mathsf{P}(\mathbf{X}) = \mathsf{P}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ compactly even when there are many variables.
- Goal 2: efficiently calculate marginal and conditionals of such compactly represented joint distributions.
- Notice: for n discrete variables of arity k, the naive (table) representation is HUGE: it requires k^n entries.
- We need to make some *assumptions* about the distribution.
  One simple assumption: independence $==$ complete factorization:
  $$\mathsf{P}(\mathbf{X}) = \prod_i \mathsf{P}(\mathbf{x}_i)$$
- But the independence assumption is too restrictive.
  So we make *conditional independence* assumptions instead.

Conditional Independence

- Notation: $\mathbf{x}_A \perp \mathbf{x}_B | \mathbf{x}_C$
  Definition: two (sets of) variables $\mathbf{x}_A$ and $\mathbf{x}_B$ are conditionally independent given a third $\mathbf{x}_C$ if:

$$\mathsf{P}(\mathbf{x}_A, \mathbf{x}_B | \mathbf{x}_C) = \mathsf{P}(\mathbf{x}_A | \mathbf{x}_C)\mathsf{P}(\mathbf{x}_B | \mathbf{x}_C) \qquad \forall \mathbf{x}_C$$

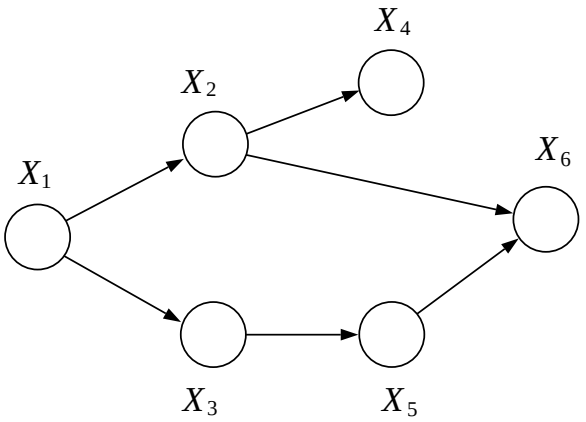
  which is equivalent to saying

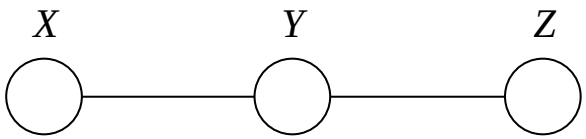
$$\mathsf{P}(\mathbf{x}_A | \mathbf{x}_B, \mathbf{x}_C) = \mathsf{P}(\mathbf{x}_A | \mathbf{x}_C) \qquad \forall \mathbf{x}_C$$

- Only a subset of all distributions respect any given (nontrivial) conditional independence statement. The subset of distributions that respect all the CI assumptions we make is the *family of distributions consisitent with our assumptions*.
- Probabilistic graphical models are a powerful, elegant and simple way to specify such a family.

Probabilistic Graphical Models

- Probabilistic graphical models represent large joint distributions compactly using a set of "local" relationships specified by a graph.
- Each random variable in our model corresponds to a graph node.
- There are directed/undirected *edges* between the nodes which tell us qualitatively about the *factorization* of the joint probability.
- There are *functions* stored at the nodes which tell us the quantitative details of the pieces into which the distribution factors.





- Graphical models are also known as Bayes(ian) (Belief) Net(work)s.

Directed Graphical Models

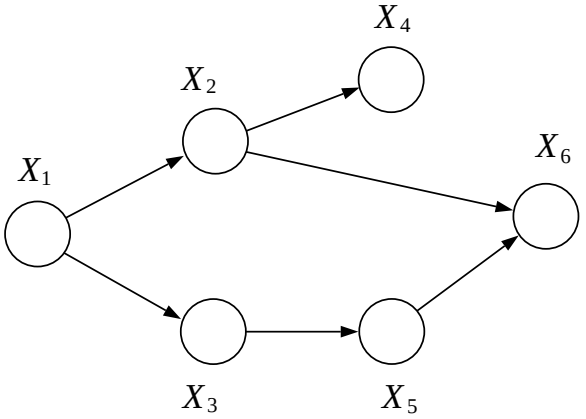
- Consider *directed acyclic graphs* over n variables.
- Each node has (possibly empty) set of parents π_i.
- Each node maintains a function $f_i(x_i; \mathbf{x}_{\pi_i})$ such that $f_i > 0$ and $\sum_{x_i} f_i(x_i; \mathbf{x}_{\pi_i}) = 1 \, \forall \pi_i$.
- Define the joint probability to be:

$$\mathsf{P}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) = \prod_i \mathsf{P}(x_i | \mathbf{x}_{\pi_i})$$

- Factorization of the joint in terms of *local conditional probabilities*. Exponential in "fan-in" of each node instead of in total variables n.

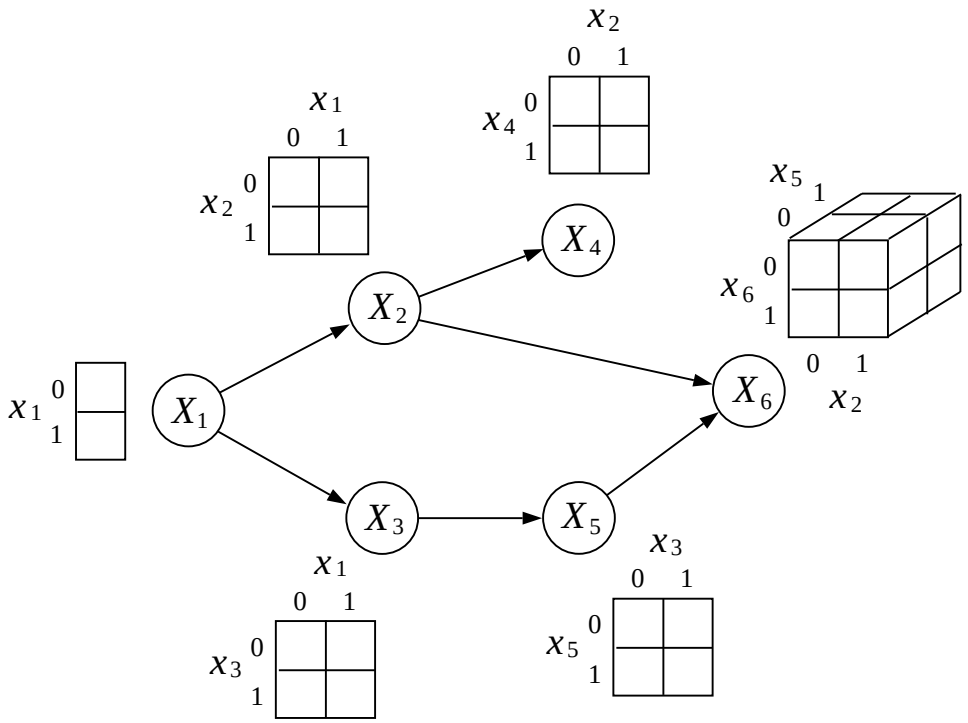
Example DAG

- Consider this six node network:



The joint probability is now:

$$\mathsf{P}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6) = \mathsf{P}(\mathbf{x}_1)\mathsf{P}(\mathbf{x}_2|\mathbf{x}_1)\mathsf{P}(\mathbf{x}_3|\mathbf{x}_1) \mathsf{P}(\mathbf{x}_4|\mathbf{x}_2)\mathsf{P}(\mathbf{x}_5|\mathbf{x}_3)\mathsf{P}(\mathbf{x}_6|\mathbf{x}_2, \mathbf{x}_5)$$



Conditional Independence and Missing Edges in DAGs

- Key point about directed graphical models:
  Missing edges imply conditional independence
- Remember, that by the chain rule we can always write the full joint as a product of conditionals, given an ordering:

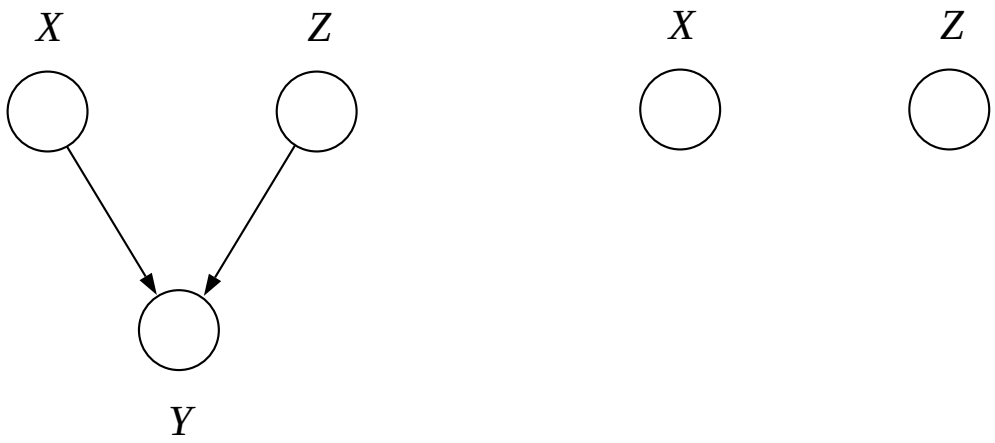
  $$\mathsf{P}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \ldots) = \mathsf{P}(\mathbf{x}_1)\mathsf{P}(\mathbf{x}_2|\mathbf{x}_1)\mathsf{P}(\mathbf{x}_3|\mathbf{x}_1, \mathbf{x}_2)\mathsf{P}(\mathbf{x}_4|\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)\ldots$$

- In general, the DAG is telling us that each variable is *conditionally independent of its non-descendants given its parents*:

  $$\{x_i \perp \mathbf{x}_{\widetilde{\pi}_i}|\mathbf{x}_{\pi_i}\}\forall i$$

  where $\mathbf{x}_{\widetilde{\pi}_i}$ are non-descendants of x_i that are not its parents.
- Removing an edge into node i eliminates an argument from the conditional probability factor $p(x_i|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{i-1})$
- Remember: the graph alone represents a *family of joint distributions consistent with its CI assumptions*, not any specific distribution.

Explaining Away



- Q: When we condition on $\mathbf{y}$, are $\mathbf{x}$ and $\mathbf{z}$ independent?

$$\mathrm{P}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \mathrm{P}(\mathbf{x})\mathrm{P}(\mathbf{z})\mathrm{P}(\mathbf{y}|\mathbf{x}, \mathbf{z})$$

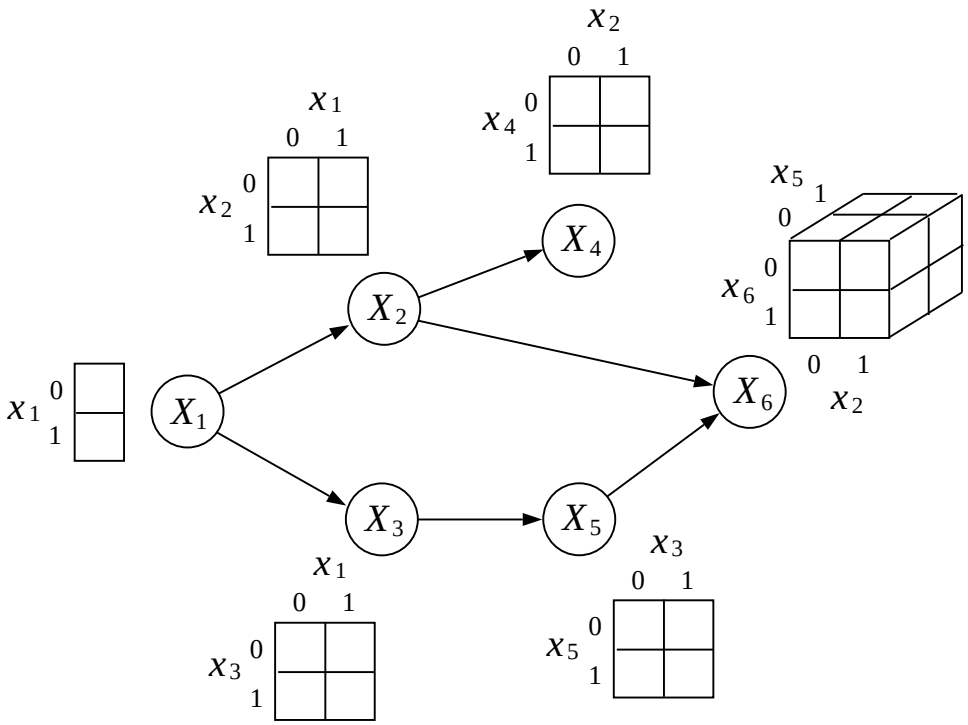
- $\mathbf{x}$ and $\mathbf{z}$ are *marginally independent*, but given $\mathbf{y}$ they are *conditionally dependent*.
- This important effect is called *explaining away* (Berkson's paradox.)
- For example, flip two coins independently; let $\mathbf{x}$=coin1,$\mathbf{z}$=coin2. Let $\mathbf{y}$=1 if the coins come up the same and $\mathbf{y}$=0 if different.
- $\mathbf{x}$ and $\mathbf{z}$ are independent, but if I tell you $\mathbf{y}$, they become coupled!

What's Inside the Nodes/Cliques?

- For directed models we need prior functions $p(x_i)$ for root nodes and parent-conditionals $p(x_i|\mathbf{x}_{\pi_i})$ for interior nodes.
- We'll consider various types of nodes: binary/discrete (categorical), continuous, interval, and integer counts.
- We'll see some basic *probability models* (parametrized families of distributions); these models live inside nodes of directed models.
- Notice that for specific (numerical) choices of factors at the nodes there may be even more conditional independencies, but when we talk about the structure of the model, we are only concerned with statements that are always true of every member of the family of distributions, no matter what specific factors live at the nodes.

Probability Tables & CPTs

- For discrete (categorical) variables, the most basic parametrization is the probability table which lists $p(x = k^{th} \text{ value})$.
- Since PTs must be nonnegative and sum to 1, for k-ary nodes there are $k - 1$ free parameters.
- If a discrete node has discrete parent(s) we make one table for each setting of the parents: this is a *conditional probability table* or CPT.



Exponential Family

- For a numeric random variable $\mathbf{x}$

$$p(\mathbf{x}|\eta) = h(\mathbf{x}) \exp\{\eta^\top T(\mathbf{x}) - A(\eta)\}$$
$$= \frac{1}{Z(\eta)} h(\mathbf{x}) \exp\{\eta^\top T(\mathbf{x})\}$$

  is an exponential family distribution with *natural parameter* η.

- Function $T(\mathbf{x})$ is a *sufficient statistic*.
- Function $A(\eta) = \log Z(\eta)$ is the log normalizer.
- Key idea: all you need to know about the data in order to estimate parameters is captured in the summarizing function $T(\mathbf{x})$.
- Examples: Bernoulli, binomial/geometric/negative-binomial, Poisson, gamma, multinomial, Gaussian, ...

Nodes with Parents

- When the parent is discrete, we just have one probability model for each setting of the parent. Examples:
  – table of natural parameters (exponential model for cts. child)
  – table of tables (CPT model for discrete child)
- When the parent is numeric, some or all of the parameters for the child node become *functions* of the parent's value.
- A very common instance of this for regression is the "linear-Gaussian": $p(\mathbf{y}|\mathbf{x}) = \mathrm{gauss}(\theta^\top \mathbf{x}; \Sigma)$.
- For classification, often use Bernoulli/Multinomial densities whose parameters π are some function of the parent: $\pi_j = f_j(\mathbf{x})$.

GLMs and Canonical Links

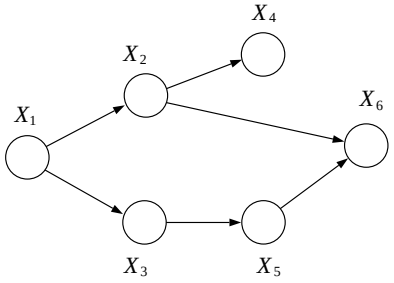
- Generalized Linear Models: $p(\mathbf{y}|\mathbf{x})$ is exponential family with conditional mean $\mu_i = f_i(\theta^\top \mathbf{x})$.
- The function f is called the *response function*.
- If we chose f to be the inverse of the mapping b/w conditional mean and natural parameters then it is called the *canonical response function* or *canonical link*:

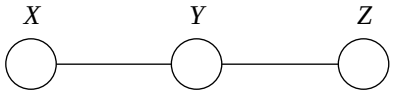
$$\eta = \psi(\mu)$$
$$f(\cdot) = \psi^{-1}(\cdot)$$

- Example: logistic function is canonical link for Bernoulli variables; softmax function is canonical link for multinomials

Review: Goal of Graphical Models

- Graphical models aim to provide *compact factorizations* of large joint probability distributions.
- These factorizations are achieved using *local functions* which exploit *conditional independencies* in the models.
- The graph tells us a basic set of *conditional independencies* that must be true. From these we can derive more that also must be true. These independencies are crucial to developing efficient algorithms valid for *all* numerical settings of the local functions.
- Local functions tell us the quantitative details of the distribution.
- Certain numerical settings of the distribution may have more independencies present, but these do not come from the *graph*.





Learning Graphical Models from Data

- In AI the bottleneck is often knowledge acquisition.
- Human experts are rare, expensive, unreliable, slow.
  But we have lots of machine readable data.
- Want to build systems automatically based on data and a small amount of prior information (e.g. from experts).


$\Rightarrow$ Sam Roweis


$\Rightarrow$ John Langford

- For now, our "systems" will be probabilistic graphical models.
- Assume the prior information we have specifies type & structure of the GM, as well as the mathematical form of the parent-conditional distributions or clique potentials.
- In this case learning $\equiv$ setting parameters.
  ("Structure learning" is also possible but we won't consider it now.)

Basic Statistical Problems

- Let's remind ourselves of the basic problems we discussed: *density estimation*, *clustering*, *classification* and *regression*.
- We can always do joint density estimation and then condition:
  Regression: $p(\mathbf{y}|\mathbf{x}) = p(\mathbf{y},\mathbf{x})/p(\mathbf{x}) = p(\mathbf{y},\mathbf{x})/\int p(\mathbf{y},\mathbf{x})d\mathbf{y}$
  Classification: $p(c|\mathbf{x}) = p(c,\mathbf{x})/p(\mathbf{x}) = p(c,\mathbf{x})/\sum_c p(c,\mathbf{x})$
  Clustering: $p(c|\mathbf{x}) = p(c,\mathbf{x})/p(\mathbf{x})$ c unobserved
  Density Estimation: $p(\mathbf{y}|\mathbf{x}) = p(\mathbf{y},\mathbf{x})/p(\mathbf{x})$ $\mathbf{x}$ unobserved

In general, if certain nodes are *always* observed we may not want to model their density:

Regression/Classification

If certain nodes are *always* unobserved they are called *hidden* or *latent* variables (more later):

Clustering/Density Est.

Multiple Observations, Complete Data, IID Sampling

- A single observation of the data $\mathbf{X}$ is rarely useful on its own.
- Generally we have data including many observations, which creates a *set of random variables*: $\mathcal{D} = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^M\}$
- We will assume two things (for now):
  1. Observations are independently and identically distributed according to joint distribution of graphical model: IID samples.
  2. We observe all random variables in the domain on each observation: complete data.
- We shade the nodes in a graphical model to indicate they are observed. (Later you will see unshaded nodes corresponding to missing data or latent variables.)