EECS 391
Introduction to
Artificial Intelligence

Review for Final Exam

Thu Dec 7, 2017

Final exam time and location

Tue Dec 12
8:00 AM to 11:00 AM
White 411

Note: Final is in our original classroom.

Final exam

- closed book no notes, will not need a calculator
- covers all main topics in the course
  - Problem Solving by Search
  - Optimal Game Playing
  - Constraint Satisfaction
  - Probabilistic Reasoning and Bayes' Rule
    (with discrete and continuous variables)
  - Bayesian Belief Networks
  - Unsupervised Learning and Clustering
  - Neural Networks
  - Probabilistic Reasoning Over Time
- There will more coverage of 2nd half.

L	Date	Topic	Chap	Notes
0	Tue, Aug 29	Course Overview		
1	Thu, Aug 31	Introduction	1	
2	Tue, Sep 5	Intelligent Agents	2	W1 out
3	Thu, Sep 7	Problem Solving by Search	3	P1 out
4	Tue, Sep 12	Uniform Cost Search, Informed Search	3	
5	Thu, Sep 14	A* Search, Hill Climbing	4	
6	Tue, Sep 19	Optimal Game Play, Minimax, α-β Pruning	5	W1 due; W2
7	Thu, Sep 21	Evaluation Functions, Stochastic Games	5	
8	Tue, Sep 26	Constraint Satisfaction Problems	6	
9	Thu, Sep 28	Constraint Propagation, Local Search for CSPs	6	P1 due
10	Tue, Oct 3	Probability and Uncertainty	13	W2 due; W3 out
11	Thu, Oct 5	Probabilistic Reasoning	13	
12	Tue, Oct 10	Reasoning with Bayes' Rule	14	
	Thu, Oct 12	Midterm Review		W3 due
	Tue, Oct 17	**Midterm Exam**		

search

game play

CSPs

Reasoning

L	Date	Topic	Chap	Notes
13	Thu, Oct 19	Bayesian Belief Networks	14	W4 out
	Tue, Oct 24	*Fall Break - no class*		
14	Thu, Oct 26	Inference in Bayes Nets	14	W4 o
15	Tue, Oct 31	Inference in Bayes Nets		
16	Thu, Nov 2	Deep Belief Networks	20	
17	Tue, Nov 7	Reasoning with Continuous Variables	20	W4 due
18	Thu, Nov 9	Learning from Examples	18	
19	Tue, Nov 14	Unsupervised Learning and Clustering	20	
20	Thu, Nov 16	Neural Networks	20	P2 out; W5 o
	Nov 21,23	*(no class - Thanksgiving Holidays)*		
21	Tue, Nov 28	Applications of Neural Networks		
22	Thu, Nov 30	Models for Sequential Data	15	
23	Tue, Dec 5	Probabilistic Reasoning Over Time	15	W5 due
	Thu, Dec 7	Final Review		P2 due (Sat Dec 9)
	Tue, Dec 12	**Final Exam 8:00AM - 11:00AM**		**in White 411**

Bayesian belief networks

probability

learning

neural networks

MMs & HMMs

some selected lecture slides

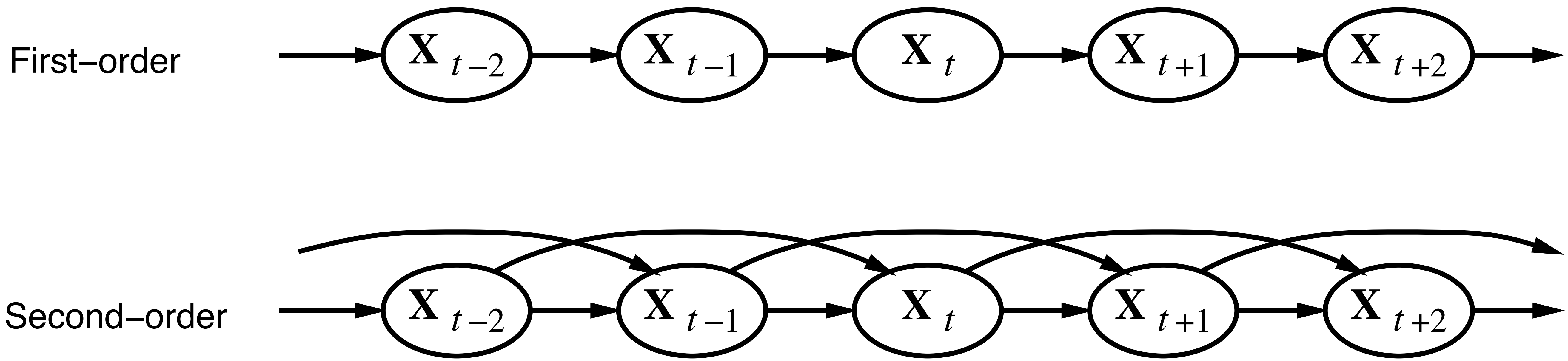
(not an exhaustive list)

Markov processes (Markov chains)

Construct a Bayes net from these variables: parents?

Markov assumption: $\mathbf{X}_t$ depends on **bounded** subset of $\mathbf{X}_{0:t-1}$

First-order Markov process: $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1})$
Second-order Markov process: $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-2}, \mathbf{X}_{t-1})$



Sensor Markov assumption: $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_{0:t}, \mathbf{E}_{0:t-1}) = \mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$

Stationary process: transition model $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1})$ and
sensor model $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$ fixed for all t

Understand HMMs and how they are defined and applied

Hidden Markov Models (HMMs)

Eg: words — hidden causal states

acoustic features — observable evidence

$$\begin{tikzcd} X_0 \arrow[r] \arrow[d] & \cdots & X_{t-1} \arrow[r] \arrow[d] & X_t \arrow[r] \arrow[d] & \cdots \\ E_0 & & E_{t-1} & E_t & \cdots \end{tikzcd}$$

Joint prob is same as Bayes Net:

$$P(X_{0:T}, E_{0:T})$$

$$= P(E_0) \prod_{i=1}^{t} \boxed{P(E_i \mid X_i) P(X_i \mid X_{i-1})}$$

What do these mean?

$\boxed{?}$ How would you model spelling correction with HMMs?

Or auto-correct

The general classification/regression problem

desired output

$$\mathbf{y} = \{y_1, \ldots, y_K\}$$

$\Uparrow$

model

$$\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_M\}$$

$\Uparrow$

Data

$$\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$$

$$\mathbf{x}_i = \{x_1, \ldots, x_N\}_i$$

for classification:

$$y_i = \begin{cases} 1 & \text{if } \mathbf{x}_i \in C_i \equiv \text{class } i, \\ 0 & \text{otherwise} \end{cases}$$

regression for arbitrary y.

model (e.g. a decision tree) is defined by M parameters, **e.g. a multi-layer neural network.**

input is a set of T observations, each an N-dimensional vector (binary, discrete, or continuous)

Given data, we want to learn a model that can correctly classify novel observations **or map the inputs to the outputs**

Computing the gradient ("learning rule") for a neural network

- Idea: minimize error by gradient descent
- Take the derivative of the objective function wrt the weights:

$$
\begin{aligned}
E &= \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^T\mathbf{x}_n - c_n)^2 \\
\frac{\partial E}{w_i} &= \frac{2}{2}\sum_{n=1}^{N}(w_0 x_{0,n} + \cdots + w_i x_{i,n} + \cdots + w_M x_{M,n} - c_n)x_{i,n} \\
&= \sum_{n=1}^{N}(\mathbf{w}^T\mathbf{x}_n - c_n)x_{i,n}
\end{aligned}
$$

- And in vector form:

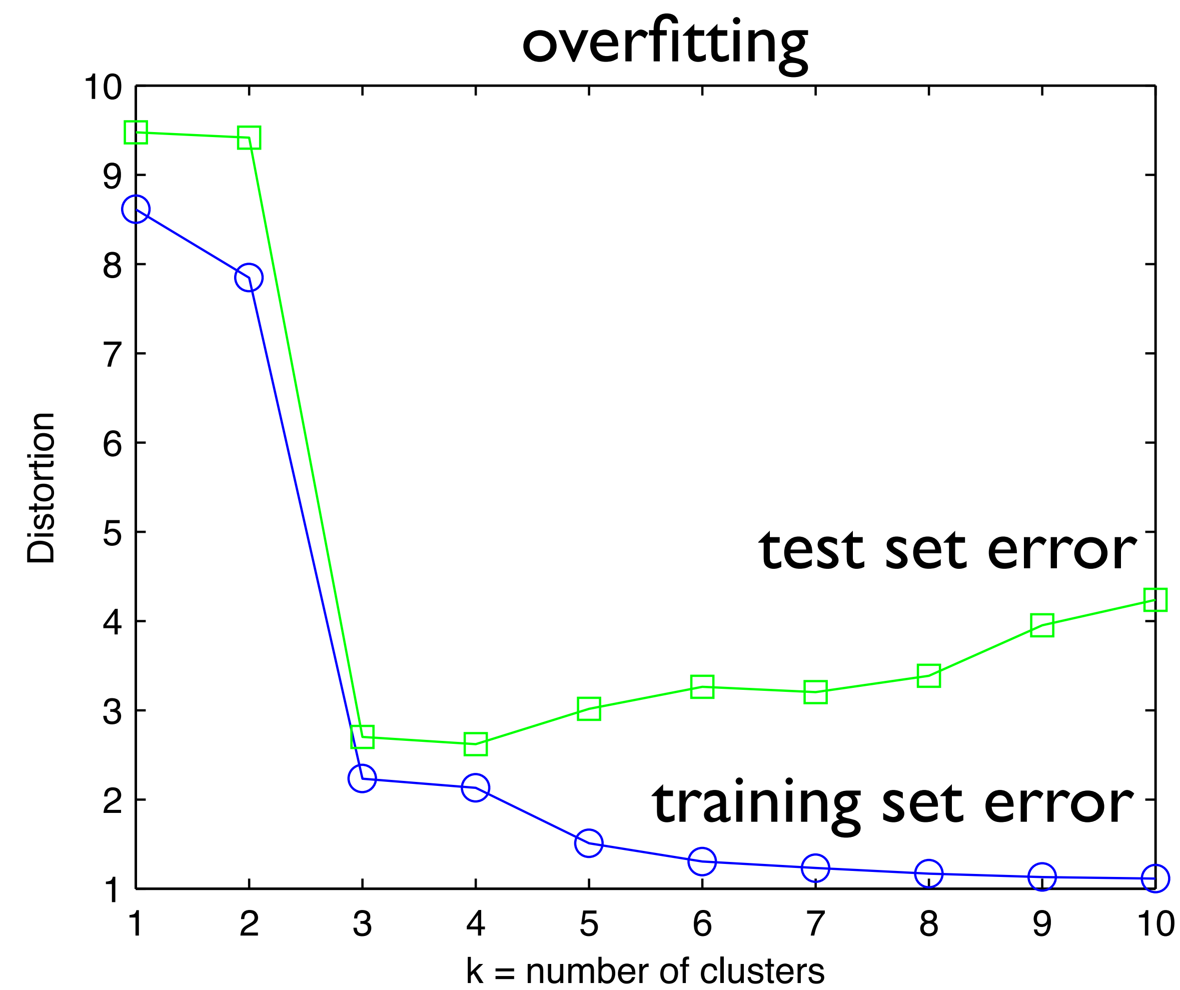
$$
\frac{\partial E}{\mathbf{w}} = \sum_{n=1}^{N}(\mathbf{w}^T\mathbf{x}_n - c_n)\mathbf{x}_n
$$

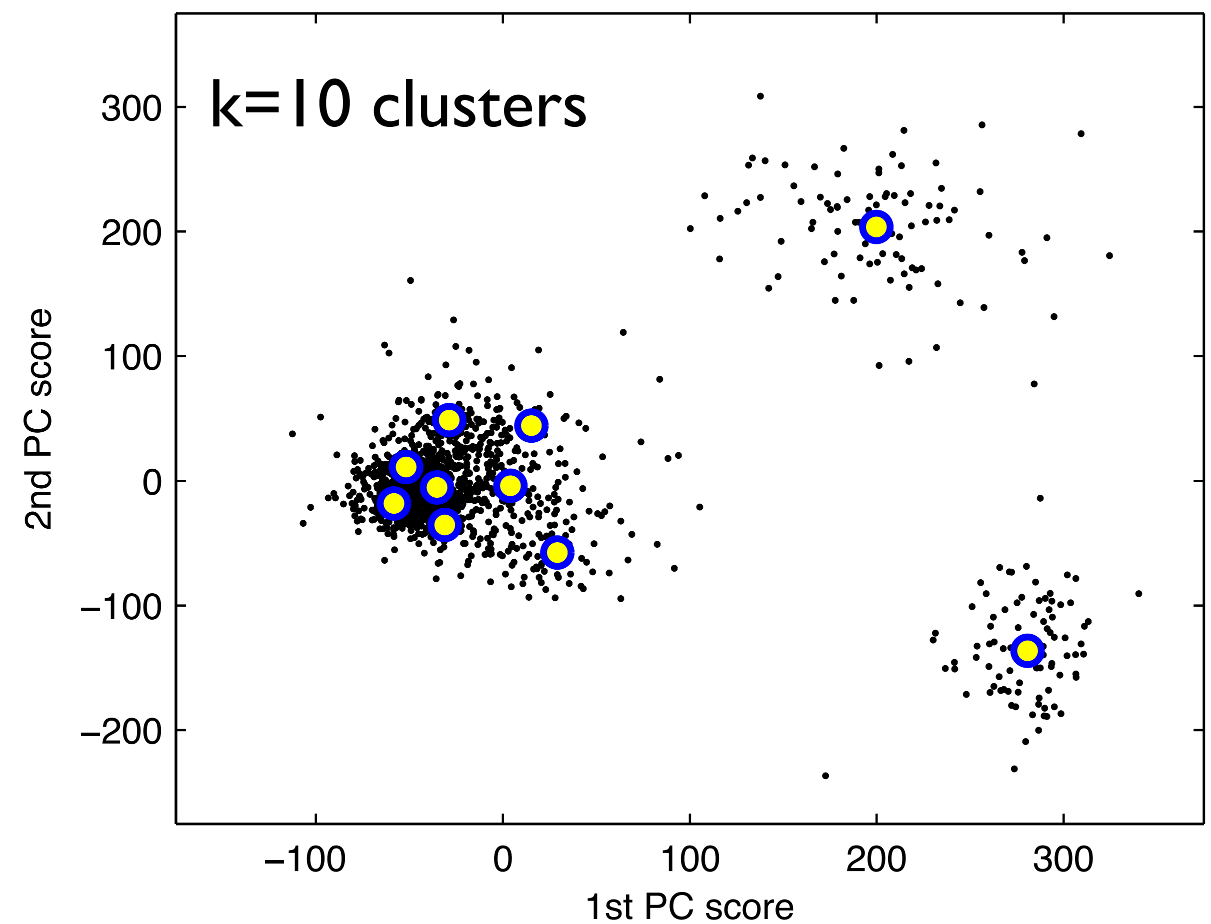
How do we choose k?

- *Increasing* k, will always *decrease* our distortion. This will ***overfit*** the data.
  - How can we avoid this?
  - Or how do we choose the best k?
- One way: ***cross validation***
- Use our distortion metric:

$$D = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \parallel \mathbf{x}_n - \boldsymbol{\mu}_k \parallel^2$$

- Then just measure the distortion on a *test data set*, and stop when we reach a minimum.





A general multi-layer neural network

- Error function is defined as before, where we use the target vector t_n to define the desired output for network output y_n.

$$E = \frac{1}{2}\sum_{n=1}^{N}(\mathbf{y}_n(\mathbf{x_n}, \mathbf{W_{1:L}}) - \mathbf{t}_n)^2$$

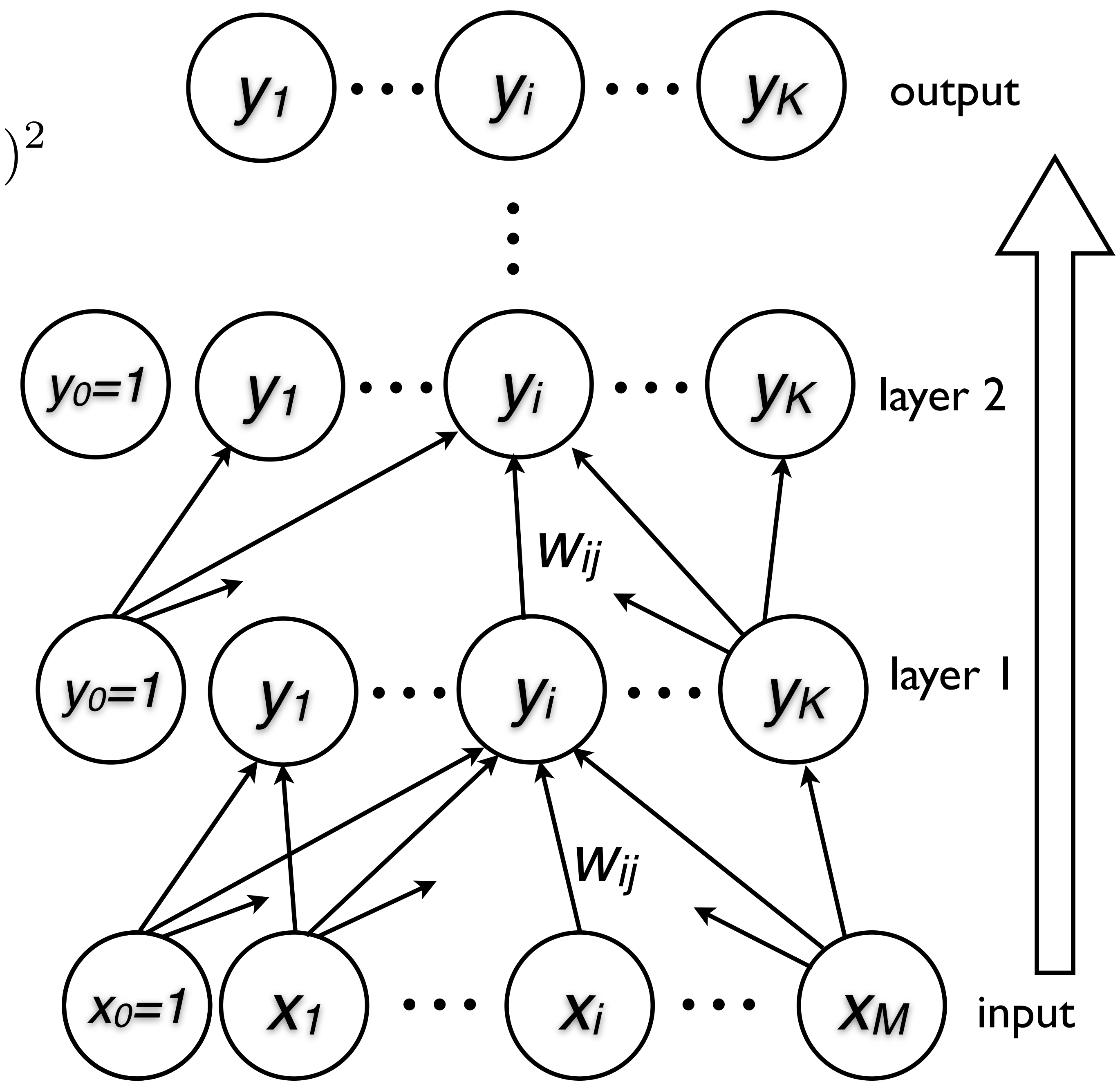
- The "forward pass" computes the outputs at each layer:

$$y_j^l = f(\sum_i w_{i,j}^l \, y_j^{l-1})$$

$$l = \{1, \dots, L\}$$

$$\mathbf{x} \equiv \mathbf{y}^0$$

$$\text{output} = \mathbf{y}^L$$



Deriving a learning rule for k-means clustering

- Our objective function is:

$$D = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \parallel \mathbf{x}_n - \boldsymbol{\mu}_k \parallel^2$$

- Differentiate w.r.t. to the mean (the parameter we want to estimate):

$$\frac{\partial D}{\partial \boldsymbol{\mu}_k} = 2 \sum_{n=1}^{N} r_{nk}(\mathbf{x}_n - \boldsymbol{\mu}_k)$$

- We know the optimum is when

$$\frac{\partial D}{\partial \boldsymbol{\mu}_k} = 2 \sum_{n=1}^{N} r_{nk}(\mathbf{x}_n - \boldsymbol{\mu}_k) = 0$$

- Here, we can solve for the mean:

$$\boldsymbol{\mu}_k = \frac{\sum_n r_{nk} \mathbf{x}_n}{\sum_n r_{nk}}$$

- This is simply a weighted mean for each cluster.

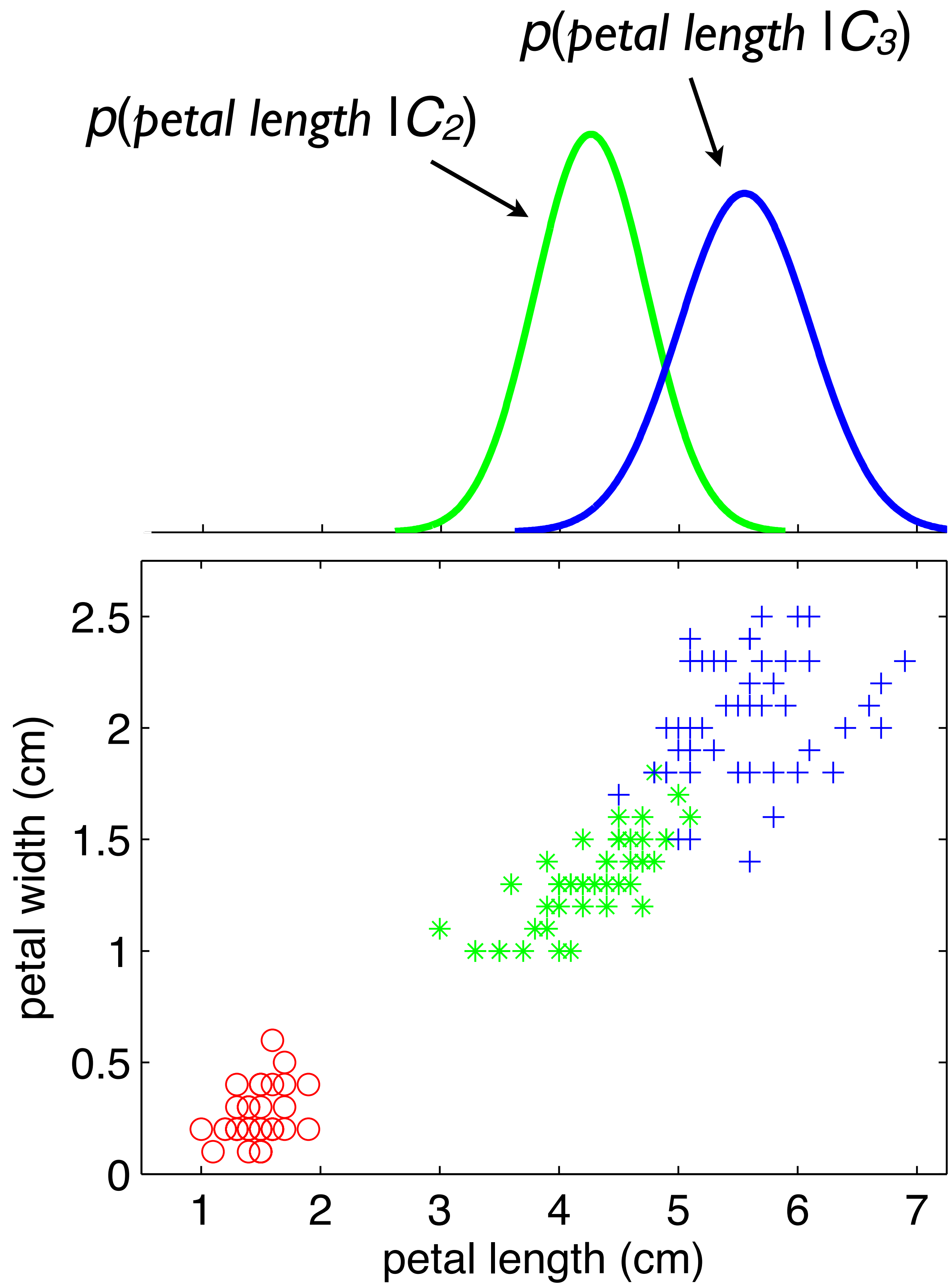
- Thus we have a simple estimation algorithm (*k-means clustering*)
  1. select k points at random
  2. estimate (update) means
  3. repeat until converged

- convergence (to a local minimum) is guaranteed

What classifier would give "optimal" performance?

- Consider the iris data.
- How would we minimize the number of *future* mis-classifications?
- We would need to know the true *distribution* of the classes.
- Assume they follow a Gaussian distribution.
- The number of samples in each class is the same (50), so (assume) $p(C_k)$ is equal for all classes.
- Because $p(\mathbf{x})$ is the same for all classes we have:

$$
\begin{aligned}
p(C_k|\mathbf{x}) &= \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})} \\
&\propto p(\mathbf{x}|C_k)p(C_k)
\end{aligned}
$$



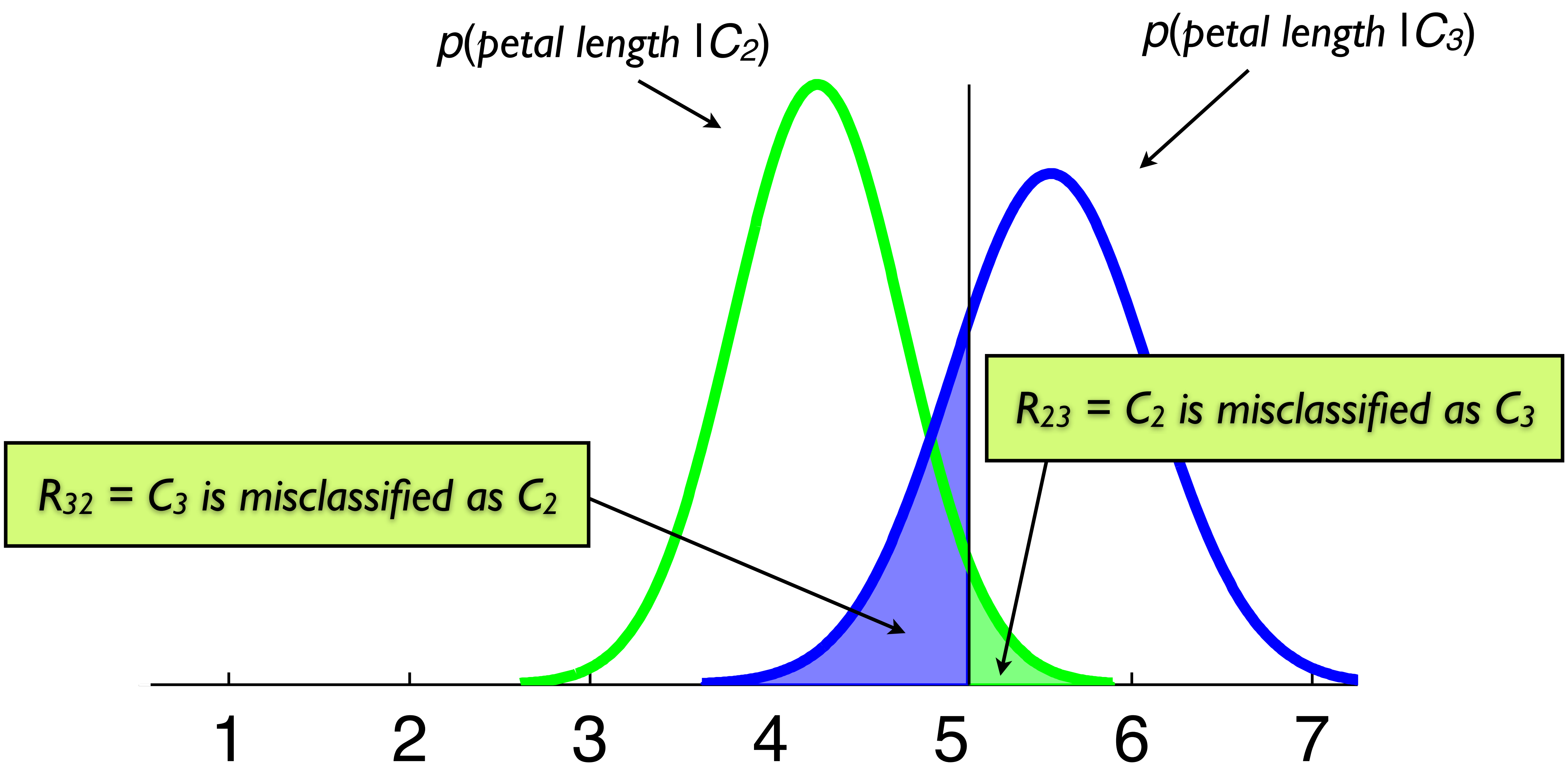
Where do we put the boundary?

- The misclassification error is defined by

$$p(\text{error}) = \int_{R_{32}} p(\mathbf{x}|C_3)P(C_3)d\mathbf{x} + \int_{R_{23}} p(\mathbf{x}|C_2)P(C_2)d\mathbf{x}$$

corrected

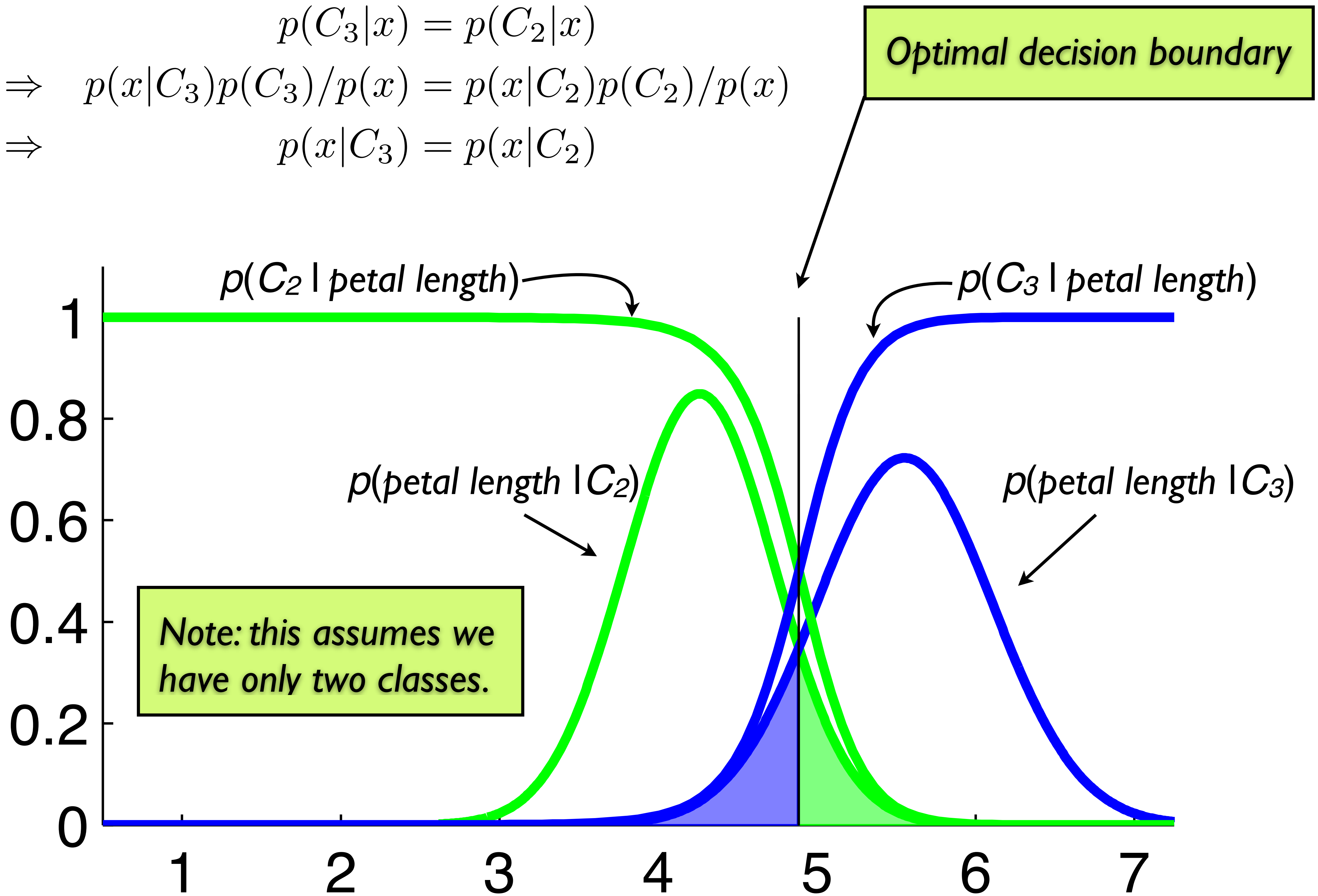
- which in our case is proportional to the data likelihood



The optimal decision boundary

- The minimal misclassification error at the point where

$$
\begin{aligned}
& p(C_3|x) = p(C_2|x) \\
\Rightarrow \quad & p(x|C_3)p(C_3)/p(x) = p(x|C_2)p(C_2)/p(x) \\
\Rightarrow \quad & p(x|C_3) = p(x|C_2)
\end{aligned}
$$



Bayesian inference with continuous variables

likelihood prior

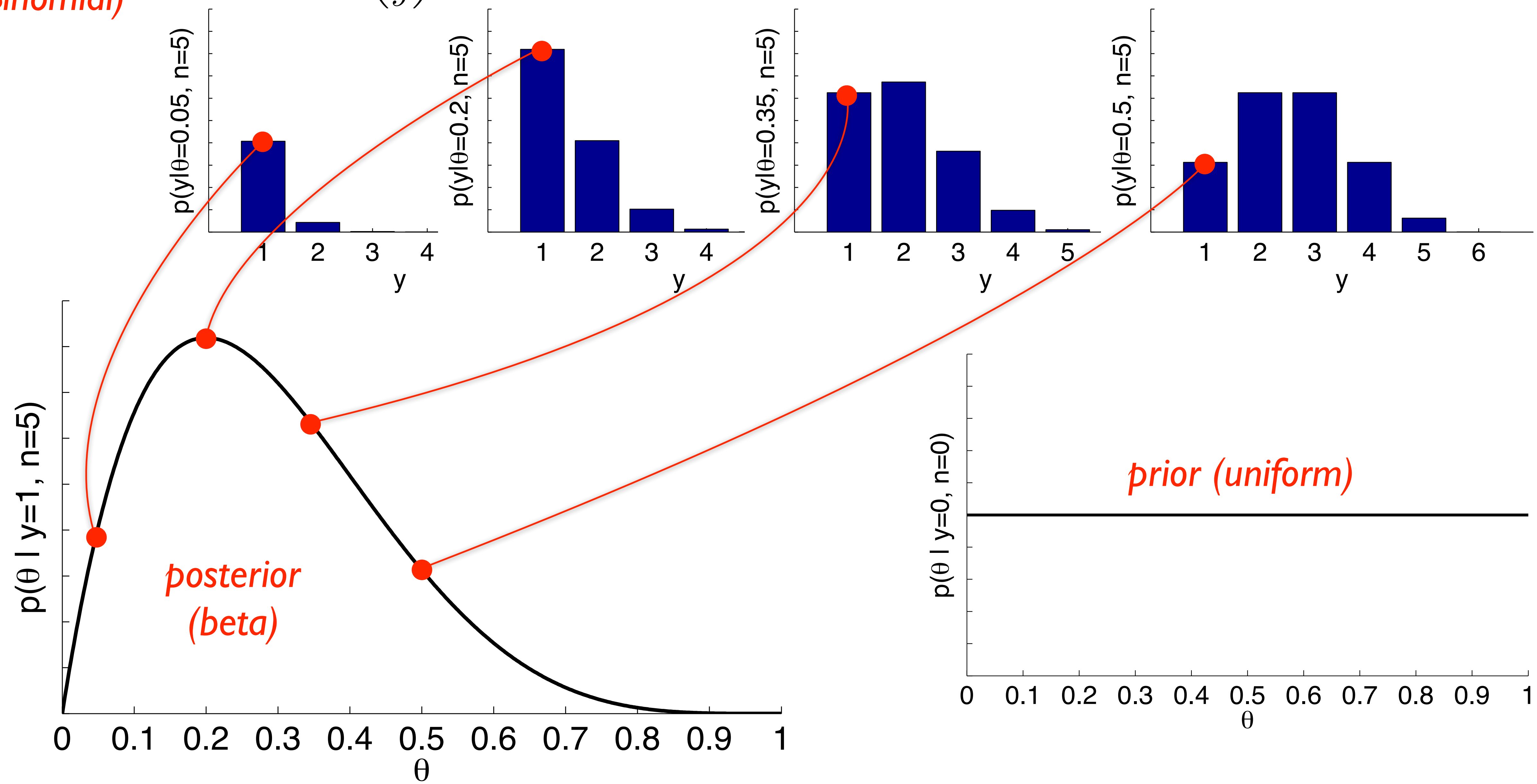
$$p(\theta|y,n) = \frac{p(y|\theta,n)p(\theta|n)}{p(y|n)}$$

posterior

$$p(y|n) = \int p(y|\theta,n)p(\theta|n)d\theta$$

normalizing constant

likelihood (Binomial) $p(\theta|y,n) \propto \binom{n}{y}\theta^{y}(1-\theta)^{n-y}$



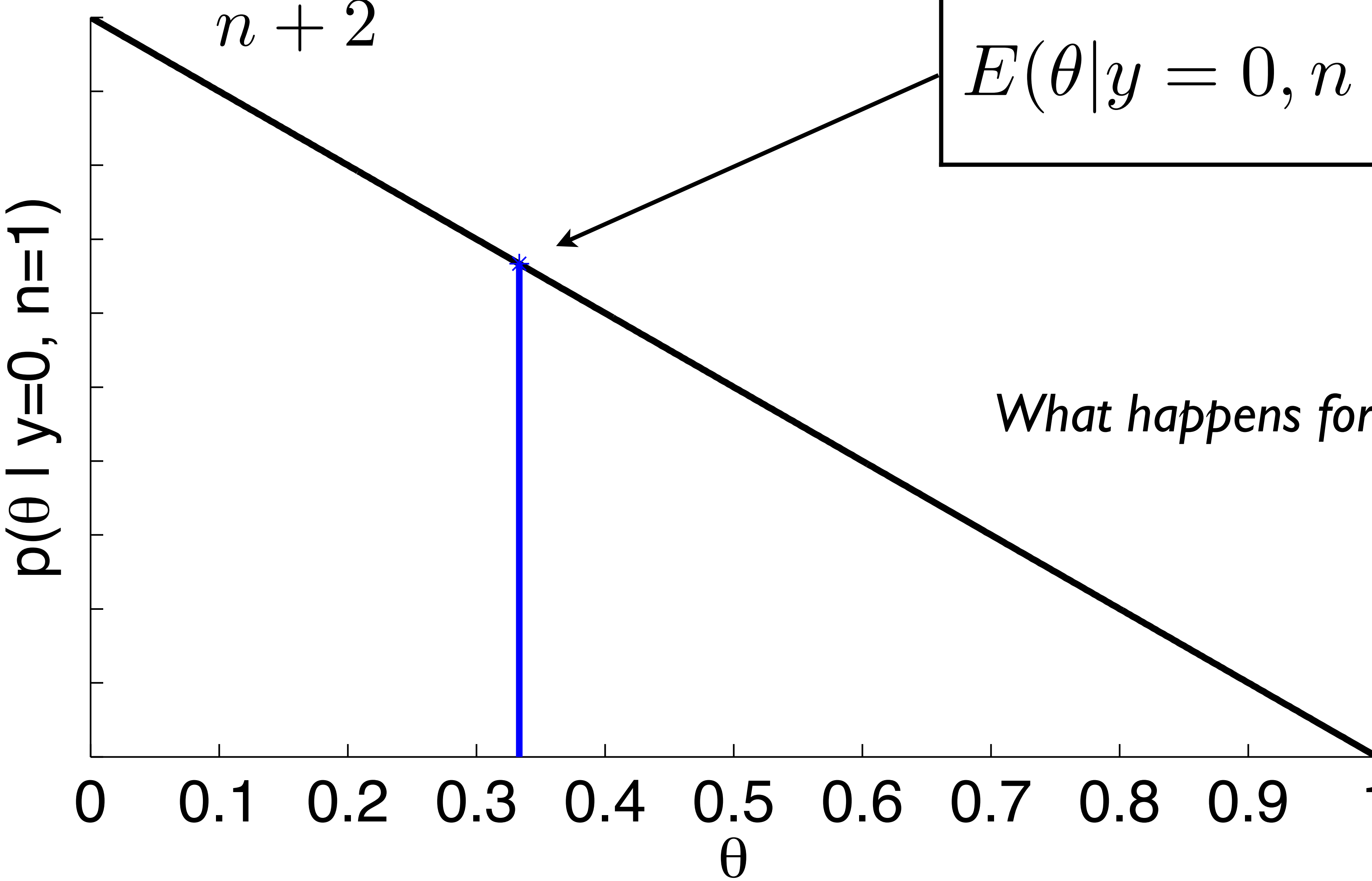
The expected value estimate

- The expected value of a pdf is:

$$E(\theta|y,n) = \int_0^1 \theta p(\theta|y,n)d\theta$$
$$= \frac{y+1}{n+2}$$

This is called *"smoothing" or "regularization"*

$$E(\theta|y=0,n=1) = \frac{1}{3}$$



What happens for zero trials?

Simplifying with "Naïve" Bayes

- What if we assume the features are independent?

$$
\begin{aligned}
p(\mathbf{x}|C_k) &= p(x_1, \ldots, x_N|C_k) \\
&= \prod_{n=1}^{N} p(x_n|C_k)
\end{aligned}
$$

- We know that's not precisely true, but it might make a good approximation.
- Now we only need to specify N different likelihoods:

$$
p(x_i = v_i|C_k = k) = \frac{\text{Count}(x_i = v_i \wedge C_k = k)}{\text{Count}(C_k = k)}
$$

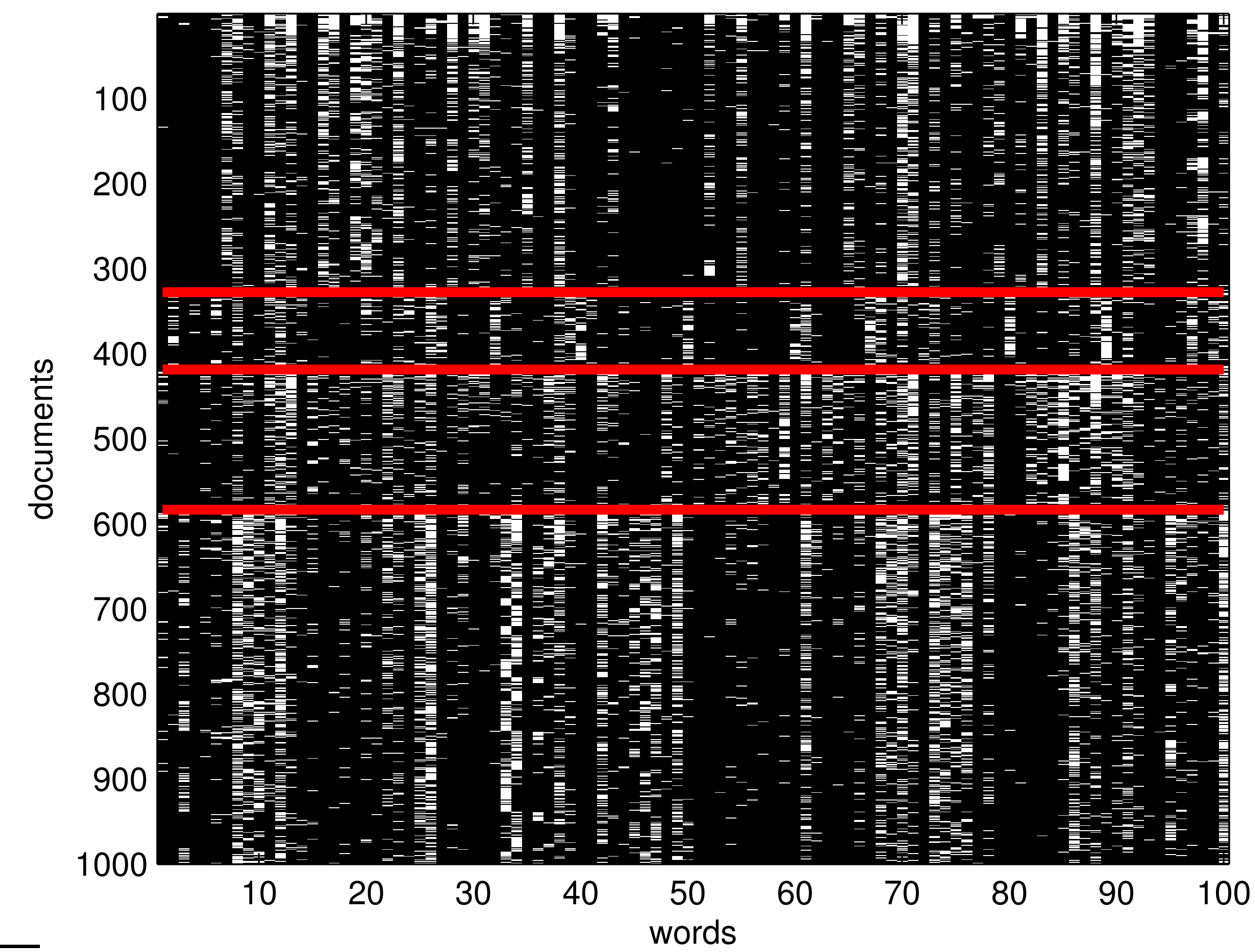
- Huge savings in number of of parameters

Text classification with the *bag of words* model

- Each row is a document represented as a bag-of-words vector.
- The different classes are different newsgroups.
- The differences in word frequencies are readily apparent.
- We can use mixture models and naïve Bayes to classify the documents

$$p(C_k|\mathbf{x}) = \frac{p(C_k)\prod_n p(x_n|C_k)}{\sum_k p(C_k)\prod_n p(x_n|C_k)}$$

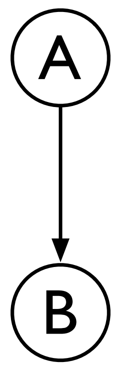
- We only replace the data likelihood with our bag-of-words model.
- This is a common way to build a spam filter or classify web pages.



Advanced topic: Bayesian belief networks

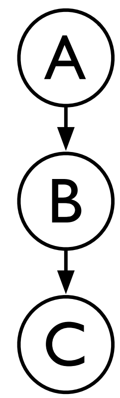
- A common way to represent probabilistic relationships is with Bayesian belief networks

Direct cause



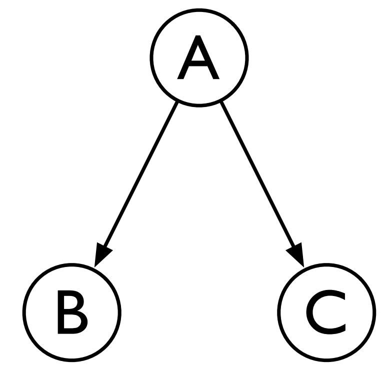
P(B|A)

Indirect cause



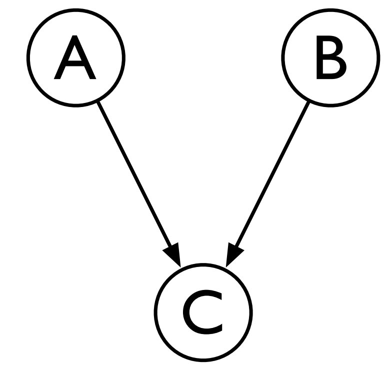
P(B|A)
P(C|B)

Common cause

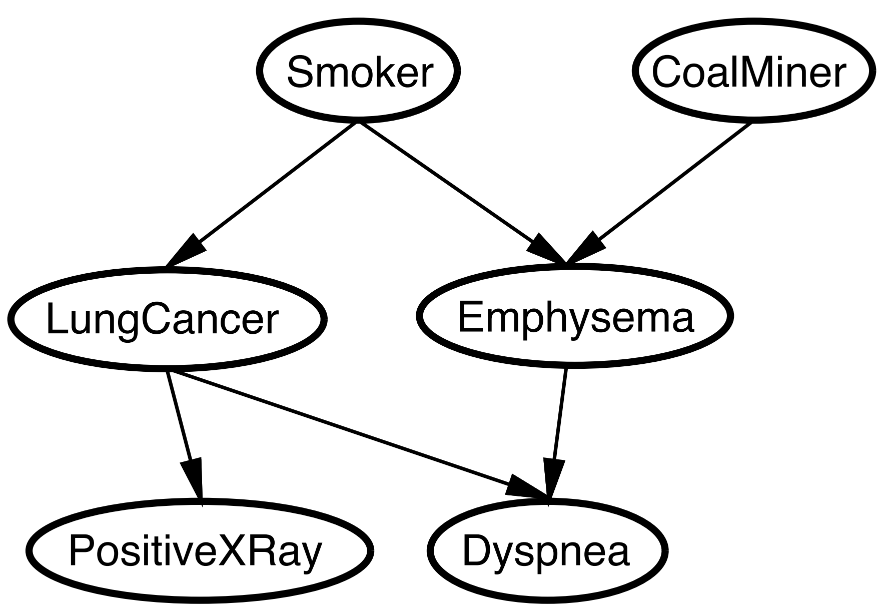


P(B|A)
P(C|A)

Common effect



P(C|A,B)



	S C	S ~C	~S C	~S~C
E	0.9	0.3	0.5	0.1
~E	0.1	0.7	0.5	0.9

Emphysema

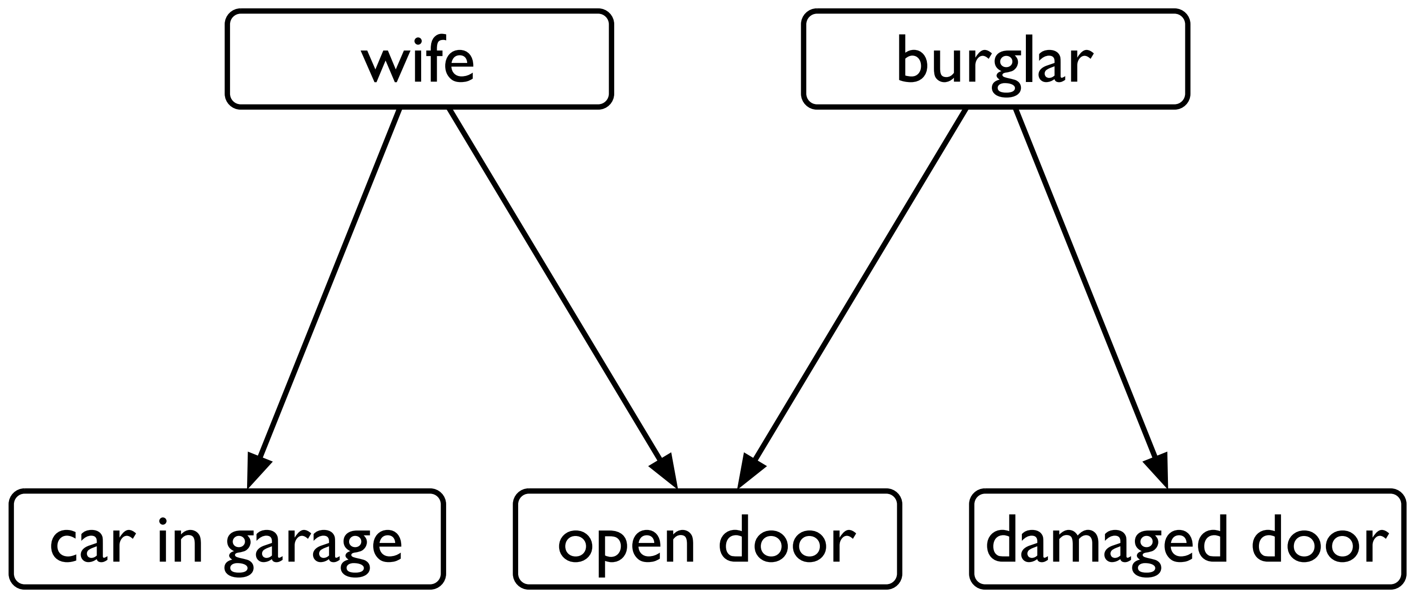
The joint probability of the network is specified in terms of the **conditional probabilities**

Recap (cont'd)

W	B	P(O|W,B)
F	F	0.01
F	T	0.25
T	F	0.05
T	T	0.75

P(W)
0.05

P(B)
0.001



W	P(C|W)
F	0.01
T	0.95

B	P(D|B)
F	0.001
T	0.5

- The structure of this model allows a simple expression for the joint probability

$$
\begin{aligned}
P(x_1, \ldots, x_n) &\equiv P(X_1 = x_1 \wedge \ldots \wedge X_n = x_n) \\
&= \prod_{i=1}^{n} P(x_i | \text{parents}(X_i))
\end{aligned}
$$

$$\Rightarrow P(o, c, d, w, b) = P(c|w)P(o|w,b)P(d|b)P(w)P(b)$$

Summary of inference with the joint probability distribution

- The complete (probabilistic) relationship between variables is specified by the joint probability:

$$P(X_1, X_2, \ldots, X_n)$$
$$= P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$$

- All conditional and marginal distributions can be derived from this using the basic rules of probability, the sum rule and the product rule

$$P(X) = \sum_Y P(X,Y)$$ sum rule, "marginalization"

$$P(X,Y) = P(Y|X)P(X) = P(X|Y)P(Y)$$ product rule

$$P(Y|X) = \frac{P(X,Y)}{P(X)}$$ corollary, conditional probability

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$ corollary, Bayes rule

Inference in Bayesian networks

- For queries in Bayesian networks, we divide variables into three classes:
  - evidence variables: $e = \{e_1, \ldots, e_m\}$ what you know
  - query variables: $x = \{x_1, \ldots, x_n\}$ what you want to know
  - non-evidence variables: $y = \{y_1, \ldots, y_l\}$ what you don't care about
- The complete set of variables in the network is $\{e \cup x \cup y\}$.
- Inferences in Bayesian networks consist of computing $p(x|e)$, the posterior probability of the query given the evidence:

$$p(x|e) = \frac{p(x,e)}{p(e)} = \alpha\, p(x,e) = \alpha \sum_y p(x,e,y)$$

- This computes the marginal distribution $p(x,e)$ by summing the joint over all values of y.
- Recall that the joint distribution is defined by the product of the conditional pdfs:

$$p(z) = \prod_{i=1} P(z_i|\text{parents}(z_i))$$

where the product is taken over all variables in the network.

Variable elimination on the burglary network

- As we mentioned in the last lecture, we could do straight summation:

$$\begin{aligned} p(b|o) &= \alpha\, p(o, w, b, c, d) \\ &= \alpha \sum_{w,c,d} p(o|w, b)p(c|w)p(d|b)p(w)p(b) \end{aligned}$$

- But: the number of terms in the sum is *exponential* in the non-evidence variables.
- This is bad, and we can do much better.
- We start by observing that we can pull out many terms from the summation.

Variable elimination

- When we've pulled out all the redundant terms we get:

$$p(b|o) = \alpha\, p(b) \sum_d p(d|b) \sum_w p(w)p(o|w,b) \sum_c p(c|w)$$

- We can also note the last term sums to one. In fact, every variable that is not an ancestor of a query variable or evidence variable is *irrelevant* to the query, so we get

$$p(b|o) = \alpha\, p(b) \sum_d p(d|b) \sum_w p(w)p(o|w,b)$$

  which contains far fewer terms: In general, *complexity is **linear** in the # of CPT entries.*

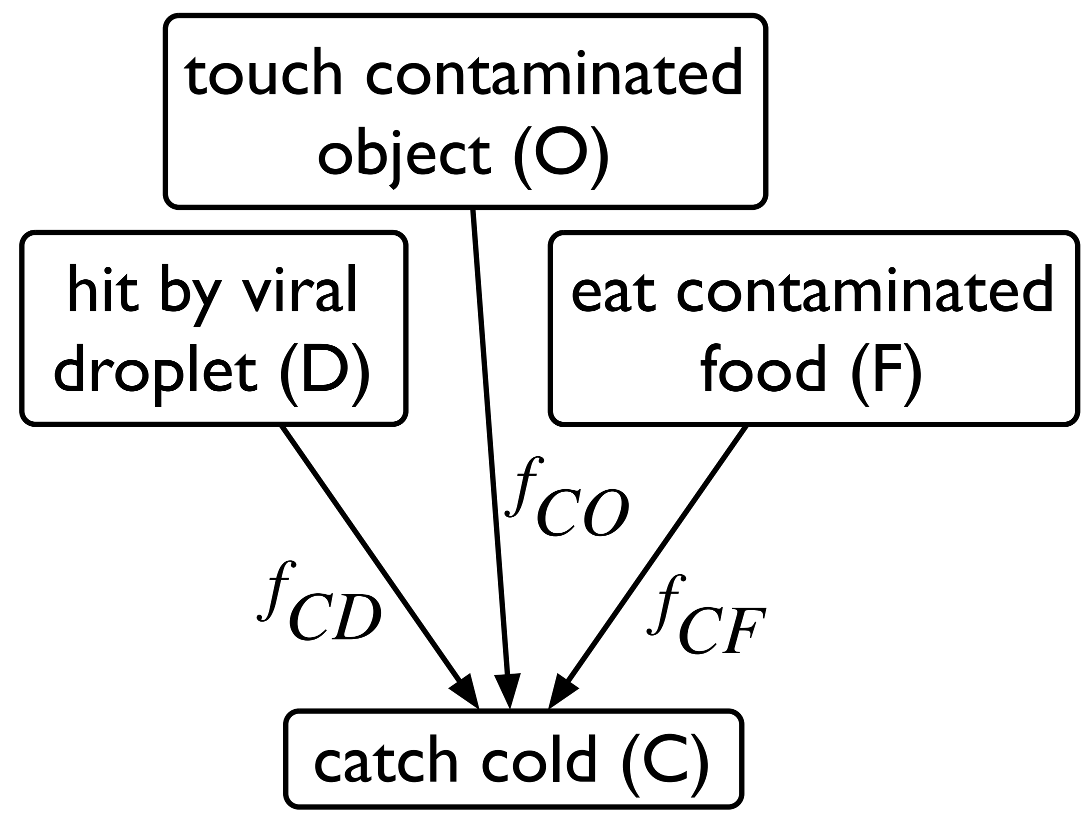
- This method is called *variable elimination*.
  - if # of parents is bounded, also linear in the number of nodes.
  - the expressions are evaluated in right-to-left order (bottom-up in the network)
  - intermediate results are stored
  - sums over each are done only for those expressions that depend on the variable

- Note: for multiply connected networks, variable elimination can have exponential complexity in the worst case.

Beyond tables: modeling causal relationships using Noisy-OR

- We assume each cause C_j can produce effect E_i with probability f_{ij}.
- The noisy-OR model assumes the parent causes of effect E_i contribute independently.
- The probability that none of them caused effect E_i is simply the product of the probabilities that each one *did not* cause E_i.
- The probability that any of them caused E_i is just one minus the above, i.e.

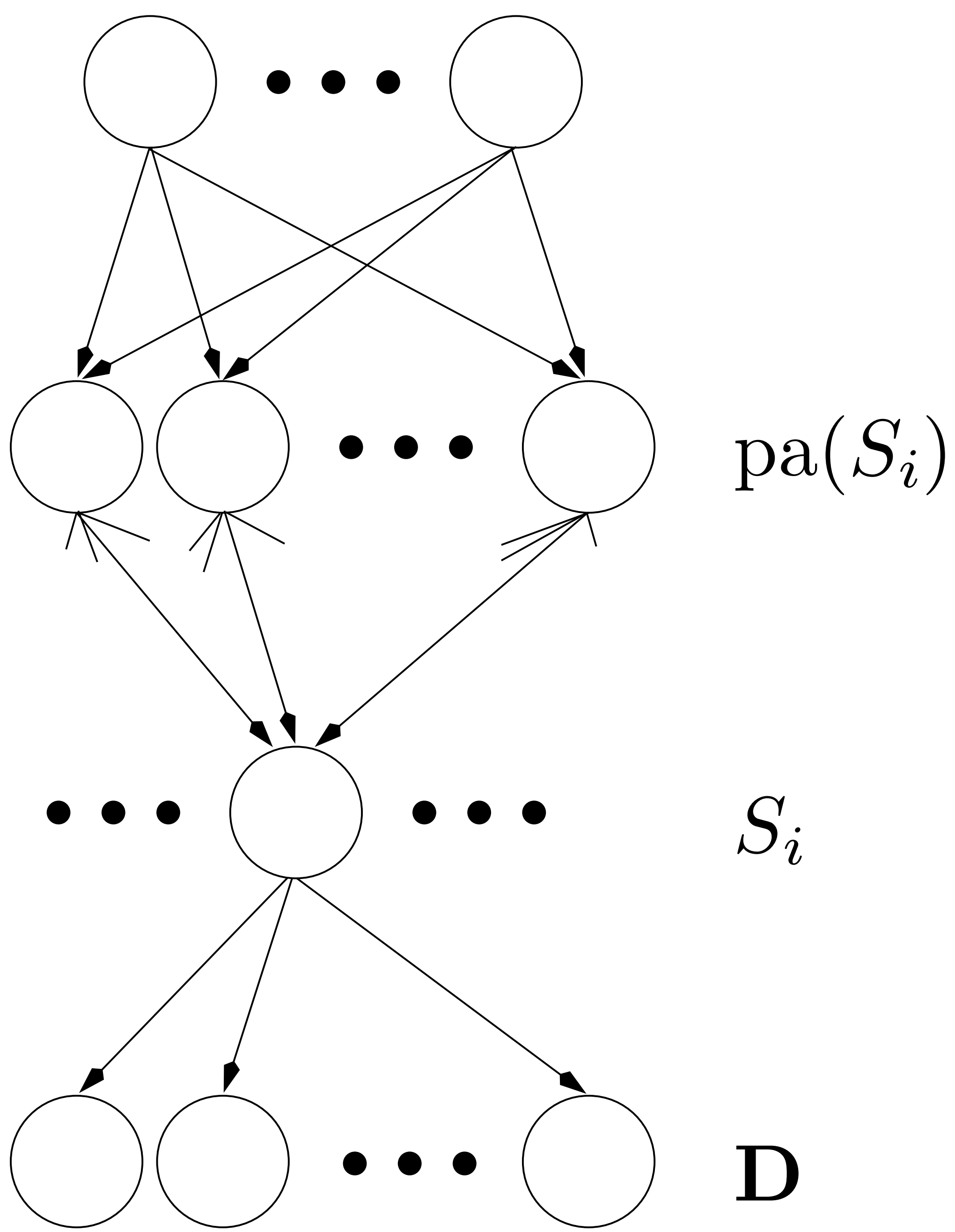
$$
\begin{aligned}
P(E_i|\mathrm{par}(E_i)) &= P(E_i|C_1, \ldots, C_n) \\
&= 1 - \prod_i (1 - P(E_i|C_j)) \\
&= 1 - \prod_i (1 - f_{ij})
\end{aligned}
$$



$$
P(C|D, O, F) = 1 - (1 - f_{CD})(1 - f_{CO})(1 - f_{CF})
$$

Hierarchical Statistical Models

A Bayesian belief network:



The joint probability of binary states is

$$P(\mathbf{S}|\mathbf{W}) = \prod_i P(S_i|\mathrm{pa}(S_i), \mathbf{W})$$

The probability S_i depends only on its parents:

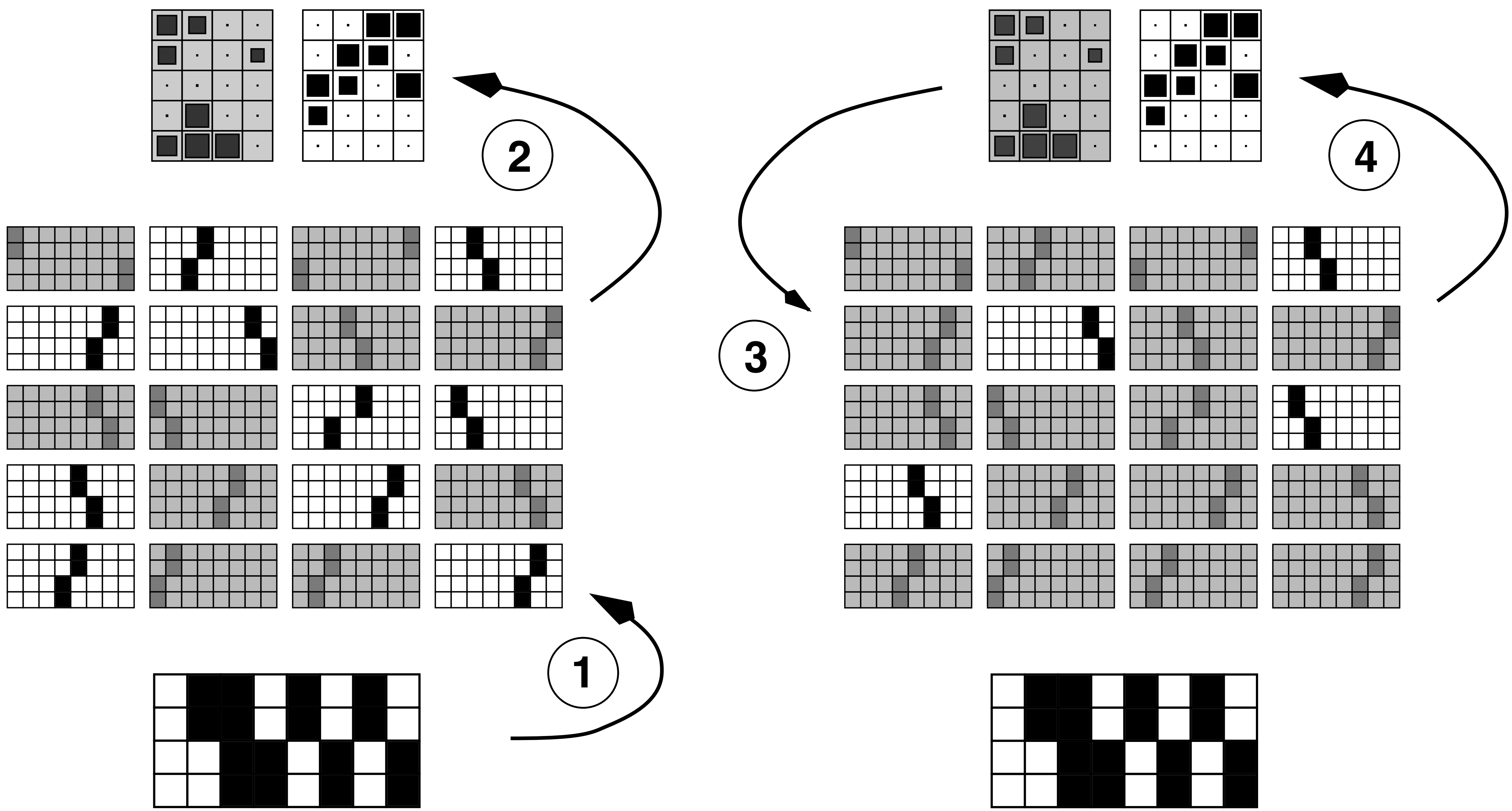
$$P(S_i|\mathrm{pa}(S_i), \mathbf{W}) = \begin{cases} h(\sum_j S_j w_{ji}) & \text{if } S_i = 1 \\ 1 - h(\sum_j S_j w_{ji}) & \text{if } S_i = 0 \end{cases}$$

The function h specifies how causes are combined, $h(u) = 1 - \exp(-u)$, $u > 0$.

Main points:

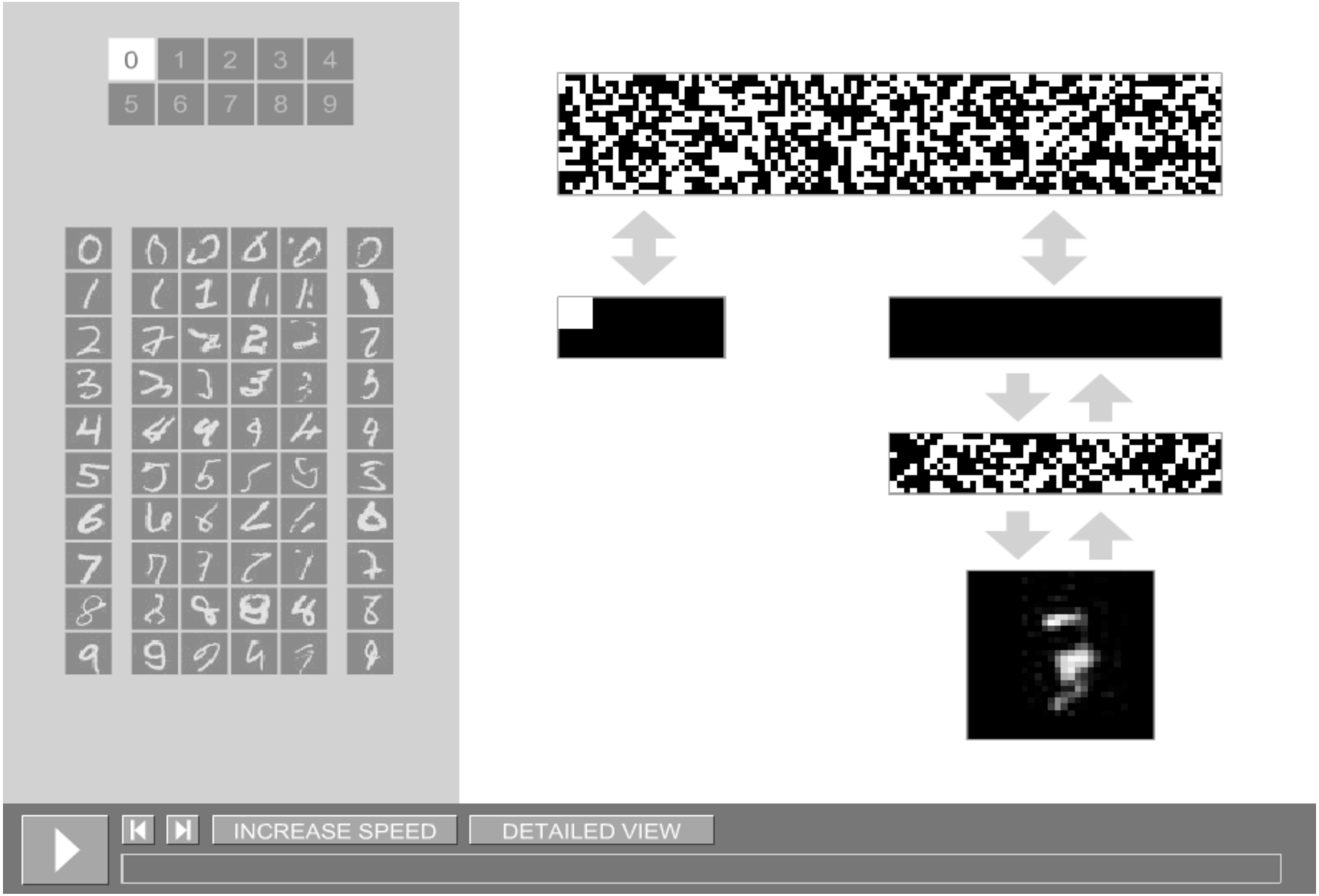
- hierarchical structure allows model to form high order representations
- upper states are priors for lower states
- weights encode higher order features

Gibbs sampling: feedback disambiguates lower-level states



One the structure learned, the Gibbs updating convergences in two sweeps.

Demo: deep-belief network model (Hinton)



www.cs.toronto.edu/~hinton/adi/index.htm

Constraint Satisfaction

- general definition, types of constraints, constraint graph and hyper graph
- constraint propagation, node consistency, forward checking, arc-consistency
- backtracking search
- heuristics: minimum remaining values, degree, least-constraining value
- local search for CSPs
- the exam will NOT cover:
  - continuous CSPs or constraint optimization problems
  - path consistency or k-consistency
  - material in section 6.3.3 or section 6.5
- Types of problems:
  - set up a problem as a CSP
  - solve a simple CSP problem using specific techniques

Optimal Game Play

- optimal game play, game trees, zero-sum games, utility function
- general concepts of deterministic vs stochastic, perfect vs imperfect information
- pruning, minimax, alpha-beta pruning
- stochastic games, imperfect information
- evaluation functions, weighted linear functions of game features
- the exam will NOT cover
  - topics in sections 5.4.2 to 5.4.4 and 5.6 to 5.8.

- Types of problems:
  - general concept questions
  - draw a game tree
  - work out the minimax value of each node
  - work out which branches are pruned using alpha-beta pruning

General concepts in search

- types of search problems, state space, actions, path cost, transition model
- problem solving performance:
  - completeness, optimality, time and space complexity
- uniformed vs informed search strategies
- tree search vs graph search, how to handle repeated states (loops)
- BFS, DFS, UCS, DLS, ID-DFS, Bidirectional search
- greedy best-first, A*, memory bounded approaches
- heuristic function, conditions for optimality, admissibility, completeness
- local search algorithms, optimization problems, hill-climbing, greedy local search, simulated annealing, local beam search, local minima and maxima
- the exam will NOT cover:
  - genetic algorithms, search in continuous spaces, or topics in sections 4.2 to 4.5

Search: examples of types of problems

- questions pertaining to the general concepts
- work out the steps of a specific search algorithm on a problem
- calculate the number of nodes expanded by different search algorithms
- determine if a heuristic is admissible