Machine Learning 10-601

Tom M. Mitchell
Machine Learning Department
Carnegie Mellon University

October 23, 2017

Today:

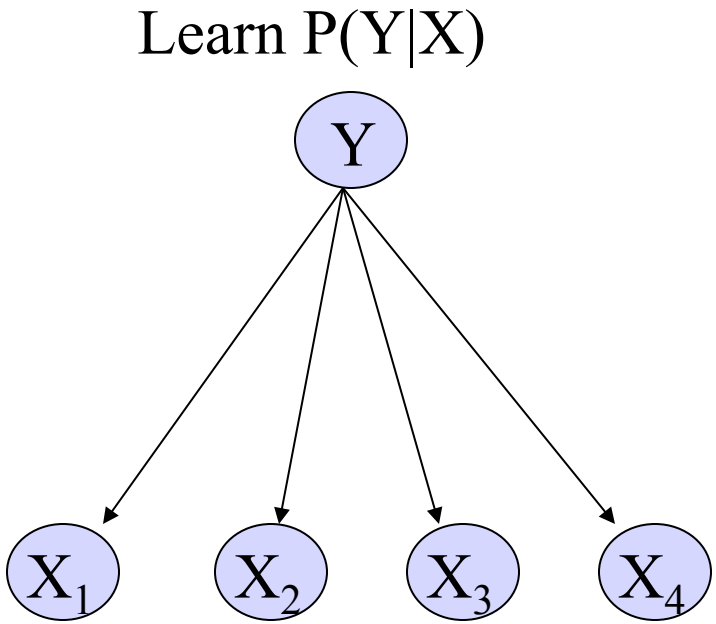
- Mixture models, clustering
- Midterm review

Midterm Exam: Oct 25

- Pittsburgh: 6:30-9:00pm
- SV: 5:30-8:30pm

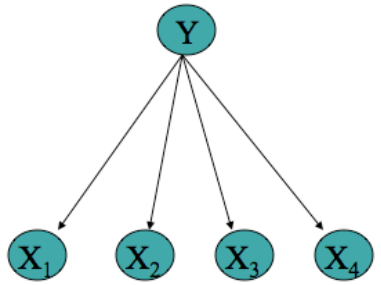
- No electronic anything allowed
- Bring one sheet of self-written notes (one side only)
- Will be graded on a curve
- Some easy questions, some hard

Using Unlabeled Data to Help Train Naïve Bayes Classifier



Y	X1	X2	X3	X4
1	0	0	1	1
0	0	1	0	0
0	0	0	1	0
?	0	1	1	0
?	0	1	0	1

EM and estimating θ



Given observed continuous X, unobserved set Y of boolean values

E step: Calculate for each training example, X(k)

the expected value of each unobserved variable Y

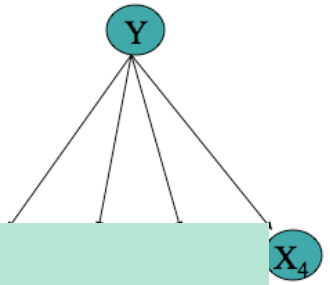
$$E_{P(Y|X_1 \ldots X_N)}[y(k)] = P(y(k)=1|x_1(k), \ldots x_N(k); \theta) = \frac{P(y(k)=1)\prod_i P(x_i(k)|y(k)=1)}{\sum_{j=0}^{1} P(y(k)=j)\prod_i P(x_i(k)|y(k)=j)}$$

M step: Calculate estimates similar to MLE, but replacing each count by its expected count

$$\theta_{ij|m} = \hat{P}(X_i = j|Y = m) = \frac{\sum_k P(y(k)=m|x_1(k) \ldots x_N(k))\ \delta(x_i(k)=j)}{\sum_k P(y(k)=m|x_1(k) \ldots x_N(k))}$$

MLE would be: $\hat{P}(X_i = j|Y = m) = \frac{\sum_k \delta((y(k)=m) \wedge (x_i(k)=j))}{\sum_k \delta(y(k)=m)}$

EM and estimating θ



Q: what if our X_i are continuous instead of discrete?

E step: Calculate for each training example, k

the expected value of each unobserved variable Y

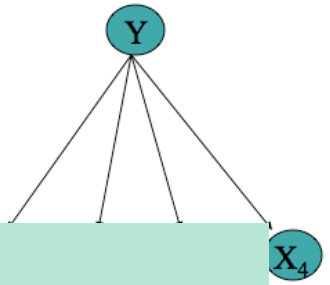
$$E_{P(Y|X_1 \ldots X_N)}[y(k)] = P(y(k)=1|x_1(k), \ldots x_N(k); \theta) = \frac{P(y(k)=1)\prod_i P(x_i(k)|y(k)=1)}{\sum_{j=0}^{1} P(y(k)=j)\prod_i P(x_i(k)|y(k)=j)}$$

M step: Calculate estimates similar to MLE, but replacing each count by its expected count

$$\theta_{ij|m} = \hat{P}(X_i = j|Y = m) = \frac{\sum_k P(y(k)=m|x_1(k)\ldots x_N(k))\ \delta(x_i(k)=j)}{\sum_k P(y(k)=m|x_1(k)\ldots x_N(k))}$$

MLE would be: $\hat{P}(X_i = j|Y = m) = \frac{\sum_k \delta((y(k)=m) \wedge (x_i(k)=j))}{\sum_k \delta(y(k)=m)}$

EM and estimating θ



Q: what if our X_i are continuous instead of discrete?

A: then we could use Gaussian Naïve Bayes instead

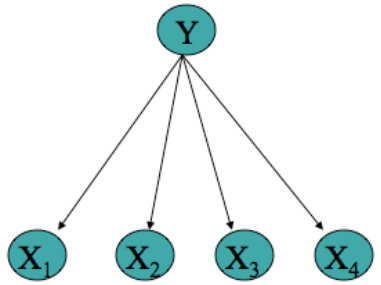
$$E_{P(Y|X_1 \ldots X_N)}[y(k)] = P(y(k)=1|x_1(k), \ldots x_N(k); \theta) = \frac{P(y(k)=1)\prod_i P(x_i(k)|y(k)=1)}{\sum_{j=0}^{1} P(y(k)=j)\prod_i P(x_i(k)|y(k)=j)}$$

M step: Calculate estimates similar to MLE, but replacing each count by its expected count

$$\theta_{ij|m} = \hat{P}(X_i = j|Y = m) = \frac{\sum_k P(y(k)=m|x_1(k) \ldots x_N(k))\ \delta(x_i(k)=j)}{\sum_k P(y(k)=m|x_1(k) \ldots x_N(k))}$$

MLE would be: $\hat{P}(X_i = j|Y = m) = \frac{\sum_k \delta((y(k)=m) \wedge (x_i(k)=j))}{\sum_k \delta(y(k)=m)}$

EM and estimating θ



Given observed continuous X, unobserved set Y of boolean values

E step: Calculate for each training example, X(k)

the expected value of each unobserved variable Y

$$E_{P(Y|X_1 \ldots X_N)}[y(k)] = P(y(k)=1|x_1(k), \ldots x_N(k); \theta) = \frac{P(y(k)=1)\prod_i P(x_i(k)|y(k)=1)}{\sum_{j=0}^{1} P(y(k)=j)\prod_i P(x_i(k)|y(k)=j)}$$

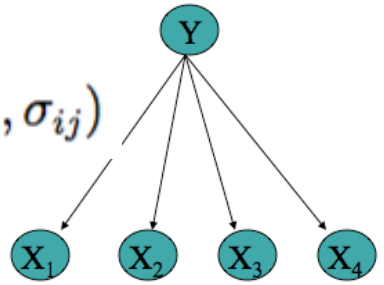
M step: Calculate estimates similar to MLE, but replacing each count by its expected count

$$\theta_{ij|m} = \hat{P}(X_i = j|Y = m) = \frac{\sum_k P(y(k)=m|x_1(k) \ldots x_N(k))\ \delta(x_i(k)=j)}{\sum_k P(y(k)=m|x_1(k) \ldots x_N(k))}$$

MLE would be: $\hat{P}(X_i = j|Y = m) = \frac{\sum_k \delta((y(k)=m) \wedge (x_i(k)=j))}{\sum_k \delta(y(k)=m)}$

EM and estimating θ

$$P(X_i = x || Y = j) = N(x; \mu_{ij}, \sigma_{ij})$$



Given observed continuous X, unobserved set Y of boolean values

E step: Calculate for each training example, X(k)

the expected value of each unobserved variable Y

$$E_{P(Y|X_1 \ldots X_N)}[y(k)] = P(y(k) = 1 | x_1(k), \ldots x_N(k); \theta) = \frac{P(y(k) = 1) \prod_i P(x_i(k) | y(k) = 1)}{\sum_{j=0}^{1} P(y(k) = j) \prod_i P(x_i(k) | y(k) = j)}$$

M step: Calculate estimates similar to MLE, but replacing each count by its expected count

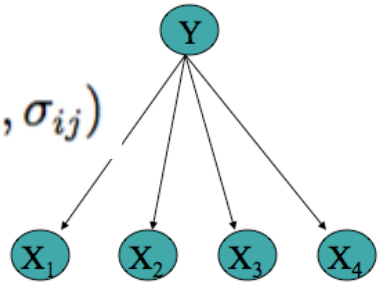
$$\mu_{ij} = \frac{\sum_k x_i(k)\ P(y(k) = j | x_1(k) \ldots x_N(k))}{\sum_k P(y(k) = j | x_1(k) \ldots x_N(k))}$$

MLE would be:

$$\mu_{ij} = \frac{\sum_k x_i(k)\ \delta(y(k) = j)}{\sum_k \delta(y(k) = j)}$$

EM and estimating θ

$$P(X_i = x || Y = j) = N(x; \mu_{ij}, \sigma_{ij})$$



Given observed continuous X, unobserved set Y of boolean values

E step: Calculate for each training example, X(k)

the expected value of each unobserved variable Y

$$E_{P(Y|X_1 \ldots X_N)}[y(k)] = P(y(k) = 1 | x_1(k), \ldots x_N(k); \theta) = \frac{P(y(k) = 1) \prod_i P(x_i(k) | y(k) = 1)}{\sum_{j=0}^{1} P(y(k) = j) \prod_i P(x_i(k) | y(k) = j)}$$

M step: Calculate estimates similar to MLE, but replacing each count by its expected count

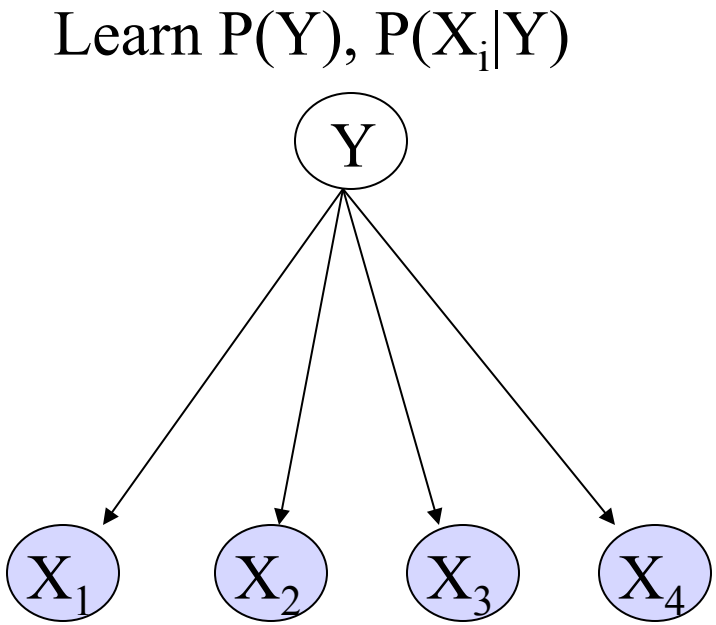
$$\mu_{ij} = \frac{\sum_k x_i(k) \; P(y(k) = j | x_1(k) \ldots x_N(k))}{\sum_k P(y(k) = j | x_1(k) \ldots x_N(k))}$$

$$P(Y = j) = \frac{1}{K} \sum_{k=1}^{k=K} P(y(k) = j)$$

Usupervised clustering

Just extreme case for EM with zero labeled examples…

Mixture Distribution to Cluster Points <X1, X2, X3, X4>



Y	X1	X2	X3	X4
?	0	0	1	1
?	0	1	0	0
?	0	0	1	0
?	0	1	1	0
?	0	1	0	1

Mixture Distributions

Model joint $P(X_1 \ldots X_n)$ as mixture of multiple distributions.

Use discrete-valued random var Z to indicate which distribution is being use for each random draw

So

$$P(X_1 \ldots X_n) = \sum_i P(Z = i) \;\; P(X_1 \ldots X_n | Z)$$

Mixture Distributions

Model joint $P(X_1 \ldots X_n)$ as mixture of multiple distributions.

Use discrete-valued random var Z to indicate which distribution is being use for each random draw

So

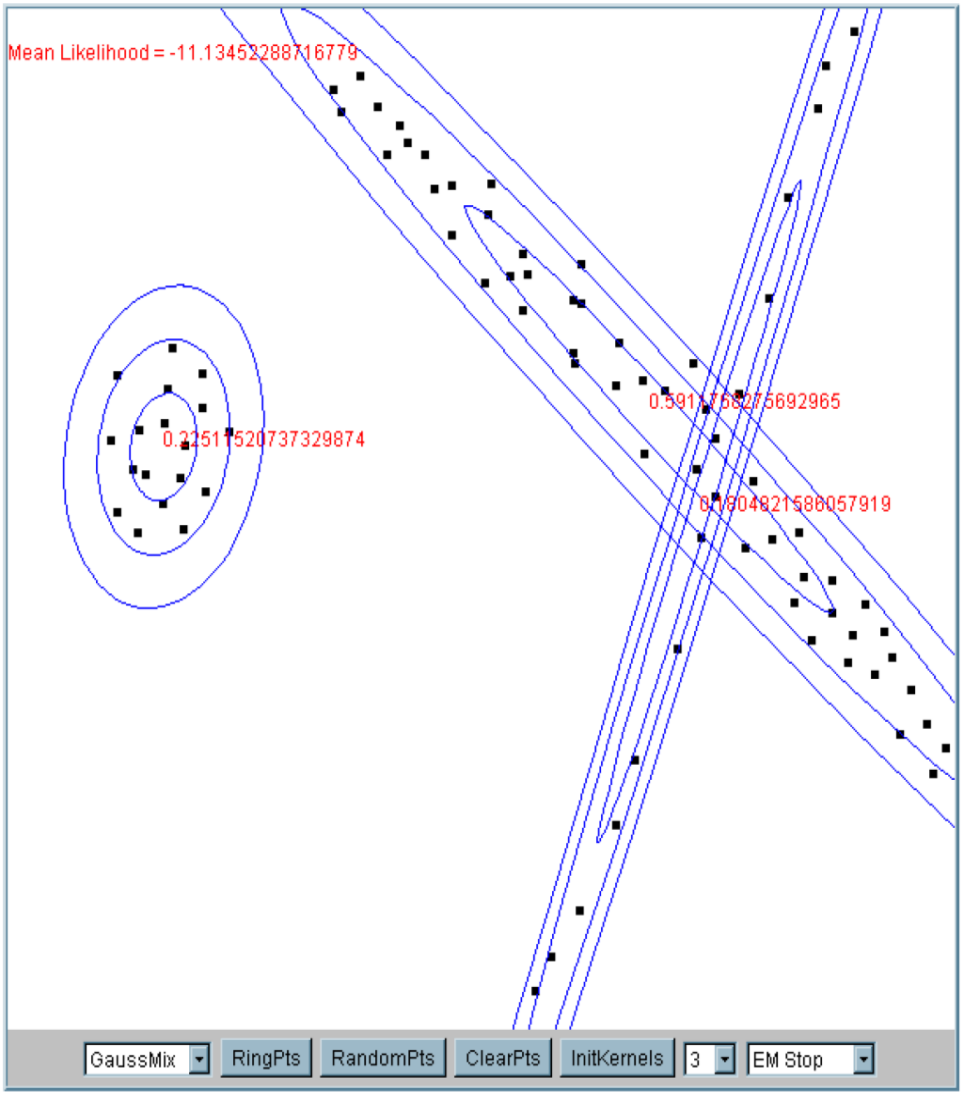
$$P(X_1 \ldots X_n) = \sum_i P(Z = i) \;\; P(X_1 \ldots X_n | Z)$$

Mixture of *Gaussians*:

- Assume each data point X=<X1, … Xn> is generated by one of several Gaussians, as follows:

1. randomly choose Gaussian i, according to P(Z=i)
2. randomly generate a data point <x1,x2 .. xn> according to $N(\mu_i, \Sigma_i)$

Mixture of Gaussians



What you should know about EM

- For learning from partly unobserved data
- MLE of θ = $\arg\max_\theta \log P(data|\theta)$
- EM estimate: θ = $\arg\max_\theta E_{Z|X,\theta}[\log P(X, Z|\theta)]$

  Where X is observed part of data, Z is unobserved
- Nice case is Bayes net of boolean vars:
  - M step is like MLE, with with unobserved values replaced by their expected values, given the other observed values
- EM for training Bayes networks
- Can also develop MAP version of EM
- Can also derive your own EM algorithm for your own problem
  - write out expression for $E_{Z|X,\theta}[\log P(X, Z|\theta)]$
  - E step: for each training example X^k, calculate $P(Z^k \mid X^k, \theta)$
  - M step: chose new θ to maximize

What you should know

Decision Trees: You should know:

- Well posed function approximation problems:
  - Instance space, X
  - Sample of labeled training data { $\langle x^{(i)}, y^{(i)} \rangle$ }
  - Hypothesis space, H = { f: X→Y }

- Learning is a search/optimization problem over H
  - Various objective functions to define the goal
    - minimize training error (0-1 loss)
    - minimize validation error (0-1 loss)
    - among hypotheses that minimize error, select smallest (?)

- Decision tree learning
  - Greedy top-down learning of decision trees (ID3, C4.5, ...)
  - Overfitting and post-pruning
  - Extensions… to continuous values, probabilistic classification
  - Widely used commercially: decision *forests*

Overfitting

Consider a hypothesis h and its

- Error rate over training data: $error_{train}(h)$
- True error rate over all data: $error_{true}(h)$

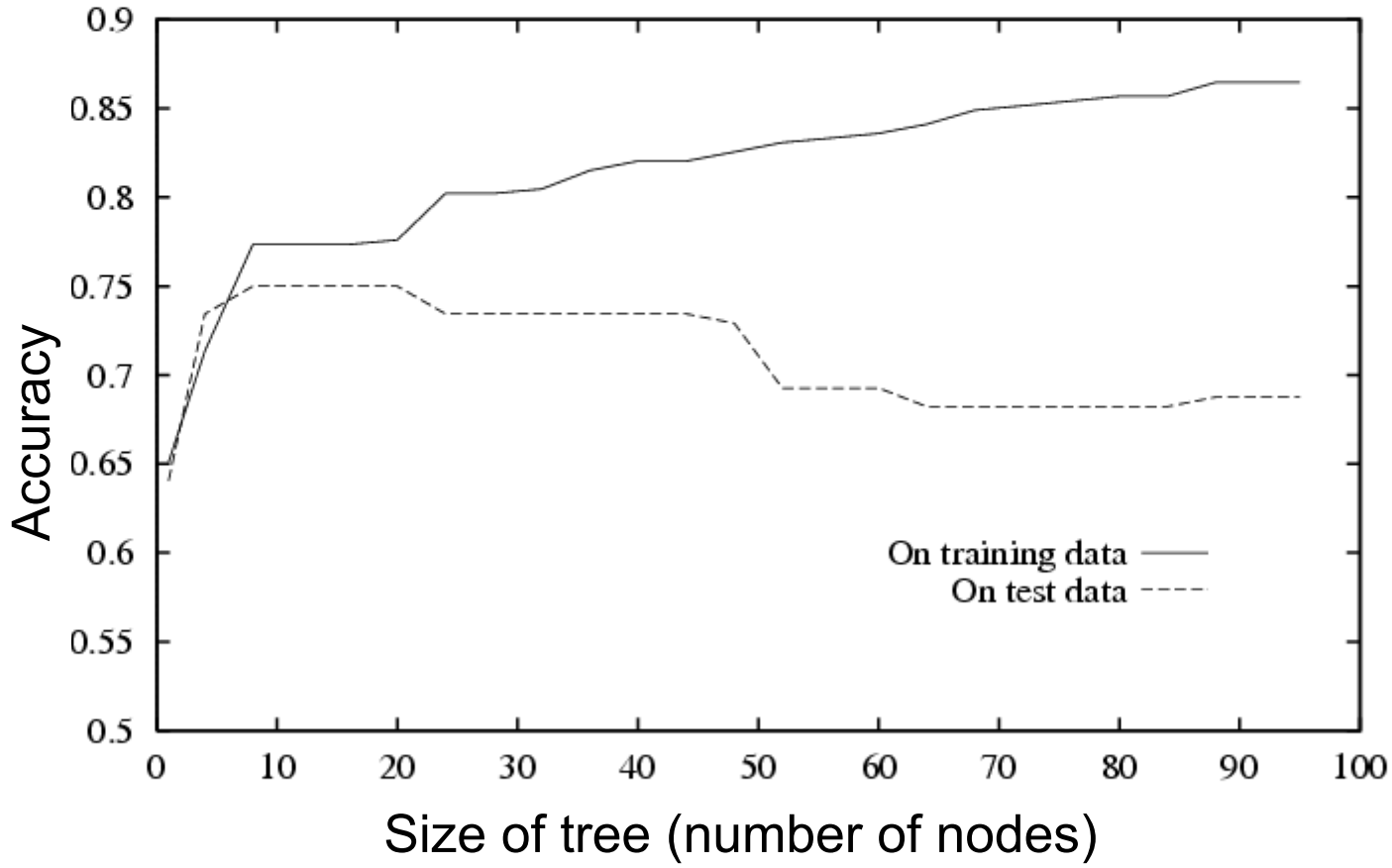
We say h overfits the training data if

$$error_{true}(h) > error_{train}(h)$$

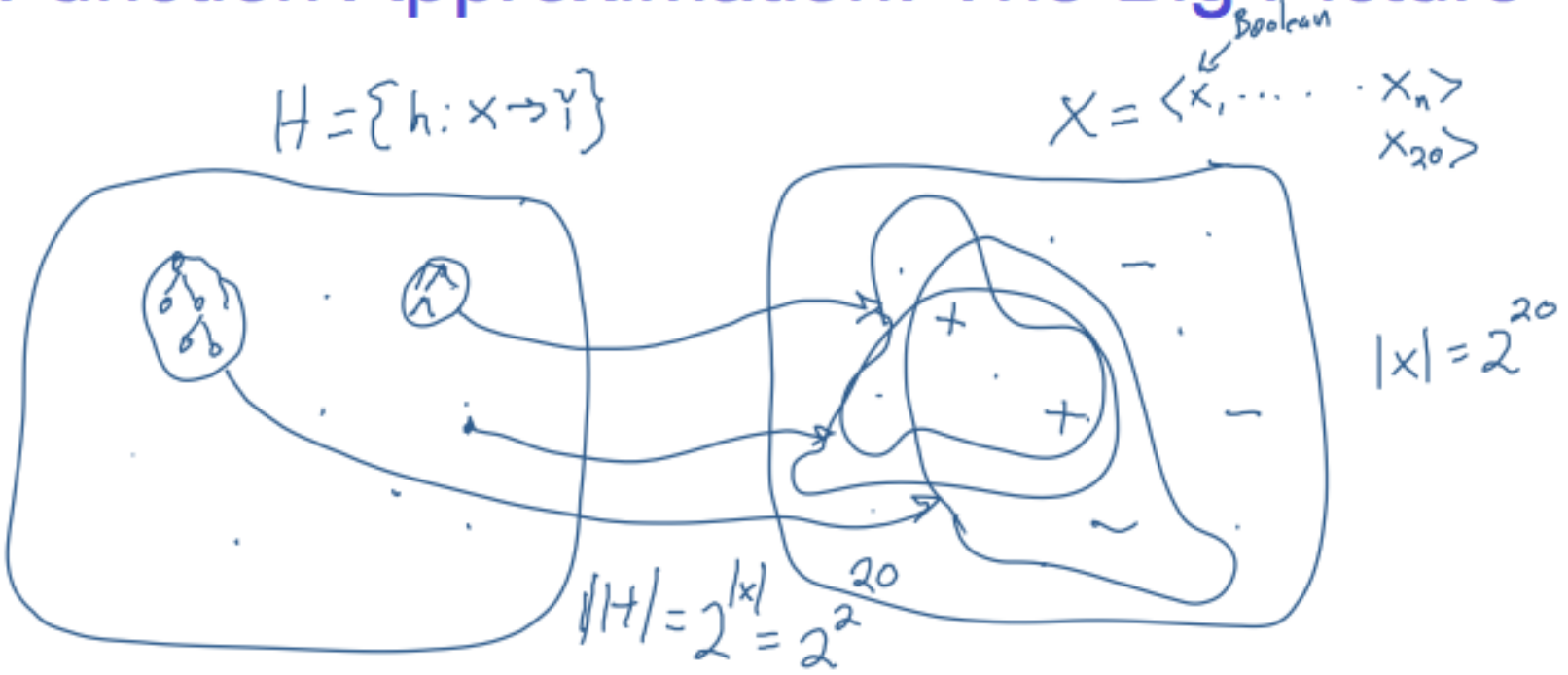
Amount of overfitting =

$$error_{true}(h) - error_{train}(h)$$

Overfitting in Decision Tree Learning



Function Approximation: The Big Picture



How many labeled examples are needed in order to determine which of the $2^{2^{20}}$ hypotheses is the correct one?

All 2^{20} instances in X must be labeled!

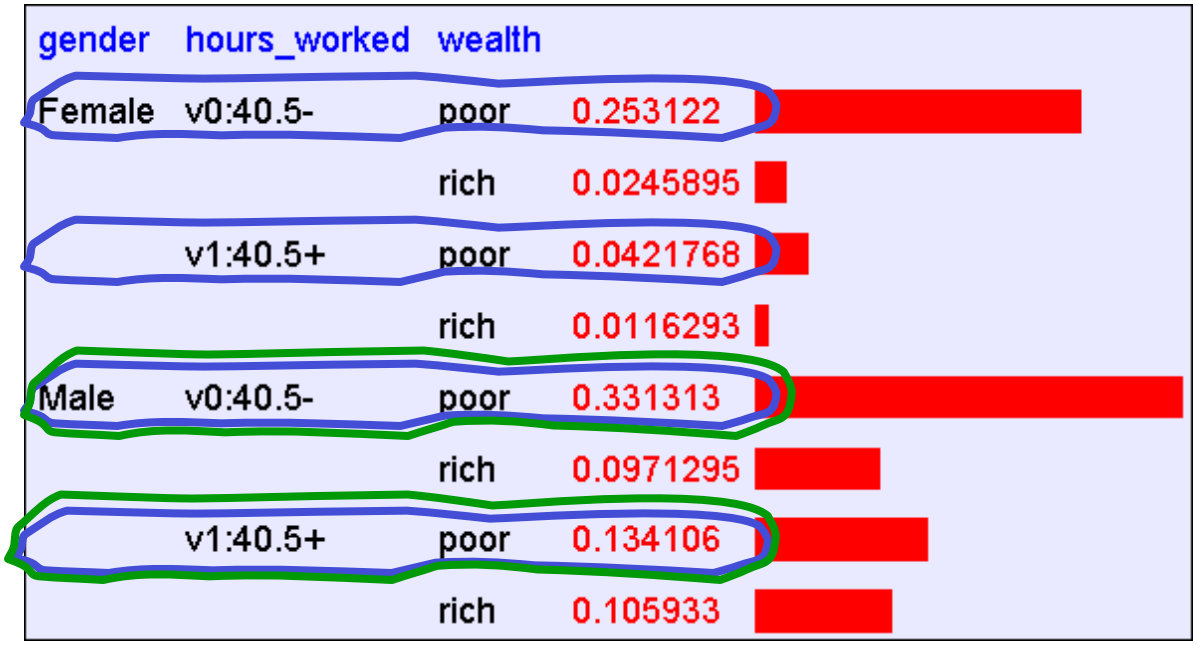
There is no free lunch!

Inductive inference – generalizing beyond the training data is impossible unless we add more assumptions (eg. priors over H)

You should know

- Probability basics
  - random variables, conditional probs, …
  - Bayes rule
  - Joint probability distributions
  - calculating probabilities from the joint distribution
- Estimating parameters from data
  - maximum likelihood estimates
  - maximum a posteriori estimates
  - distributions – Bernoulli, Binomial, Beta, Dirichlet, …
  - conjugate priors

Inference with the Joint



$$P(E_1 \mid E_2) = \frac{P(E_1 \wedge E_2)}{P(E_2)} = \frac{\sum_{\text{rows matching } E_1 \text{ and } E_2} P(\text{row})}{\sum_{\text{rows matching } E_2} P(\text{row})}$$

P(Male | Poor) = 0.4654 / 0.7604 = 0.612

sounds like the solution to
learning F: X → Y,
or P(Y | X).

Main problem: learning P(Y|X)
can require more data than we have

consider learning Joint Dist. with 100 attributes
# of rows in this table? $2^{100} > 10^{30}$
# of people on earth? 10^{10}
fraction of rows with 0 training examples? 99.99

What to do?

1. Be smart about how we estimate probabilities from sparse data
   - maximum likelihood estimates
   - maximum a posteriori estimates

2. Be smart about how to represent joint distributions
   - Bayes networks, graphical models, conditional independencies

Principles for Estimating Probabilities

- Maximum Likelihood Estimate (MLE): choose θ that maximizes probability of observed data $\mathcal{D}$

$$\hat{\theta} = \arg\max_{\theta} P(\mathcal{D} \mid \theta)$$

- Maximum a Posteriori (MAP) estimate: choose θ that is most probable given prior probability and observed data

$$\begin{aligned}\hat{\theta} &= \arg\max_{\theta} P(\theta \mid \mathcal{D}) \\ &= \arg\max_{\theta} \frac{P(\mathcal{D} \mid \theta) P(\theta)}{P(\mathcal{D})} \\ &= \arg\max_{\theta} P(\mathcal{D} \mid \theta) P(\theta)\end{aligned}$$

Principles for Estimating Probabilities

Principle 1 (maximum likelihood):

- choose parameters θ that maximize **P(data | θ)**
- result in our case: $\hat{\theta}^{MLE} = \frac{\alpha_1}{\alpha_1 + \alpha_0}$

Principle 2 (maximum a posteriori probability):

- choose parameters θ that maximize **P(θ | data)**
- result in our case:

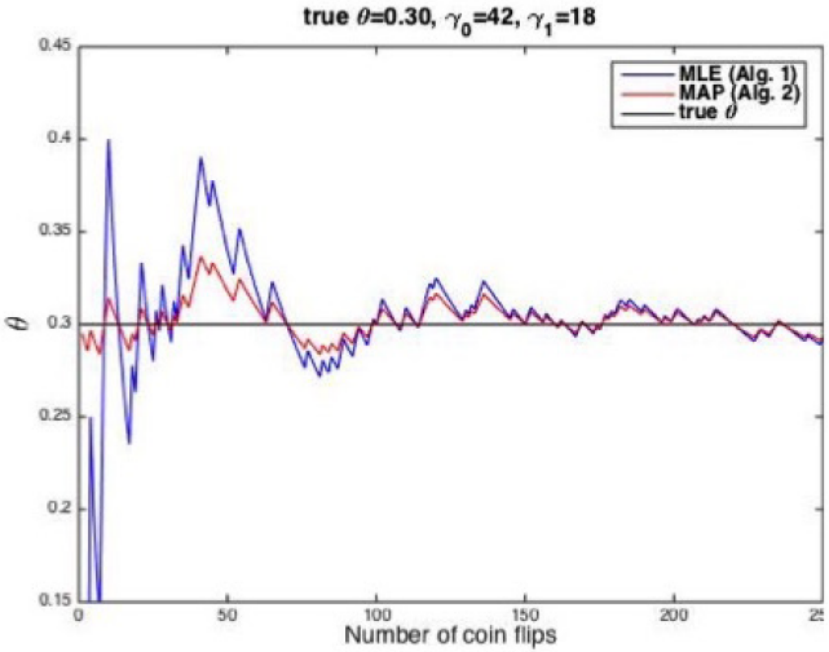
$$\hat{\theta}^{MAP} = \frac{\alpha_1 + \#\text{hallucinated\_1s}}{(\alpha_1 + \#\text{hallucinated\_1s}) + (\alpha_0 + \#\text{hallucinated\_0s})}$$

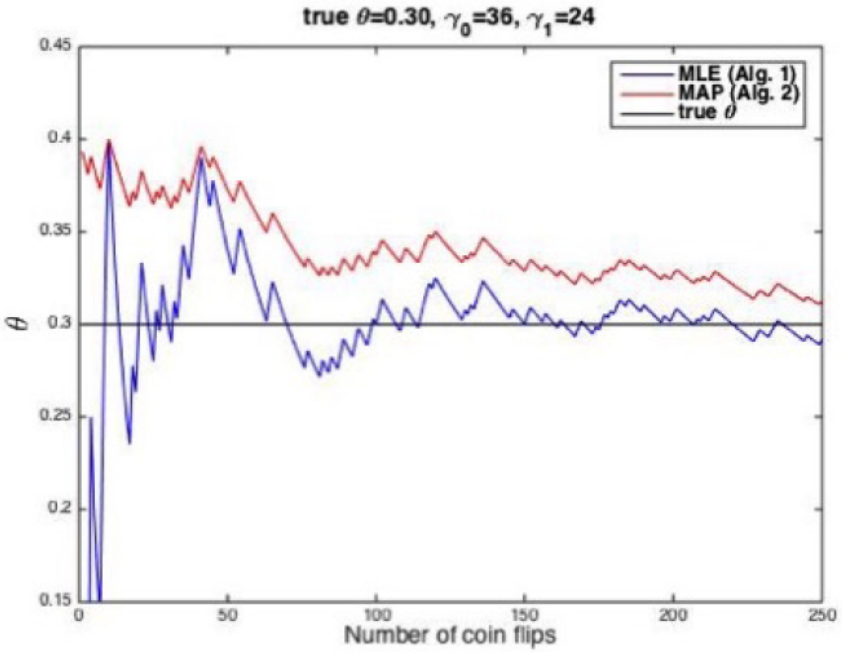
Correct MAP Priors

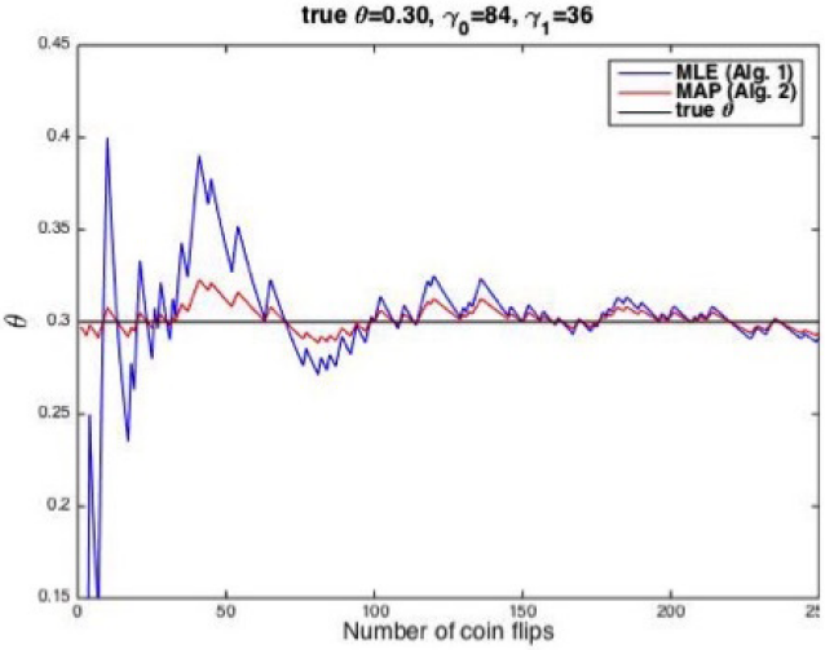
Incorrect MAP Priors

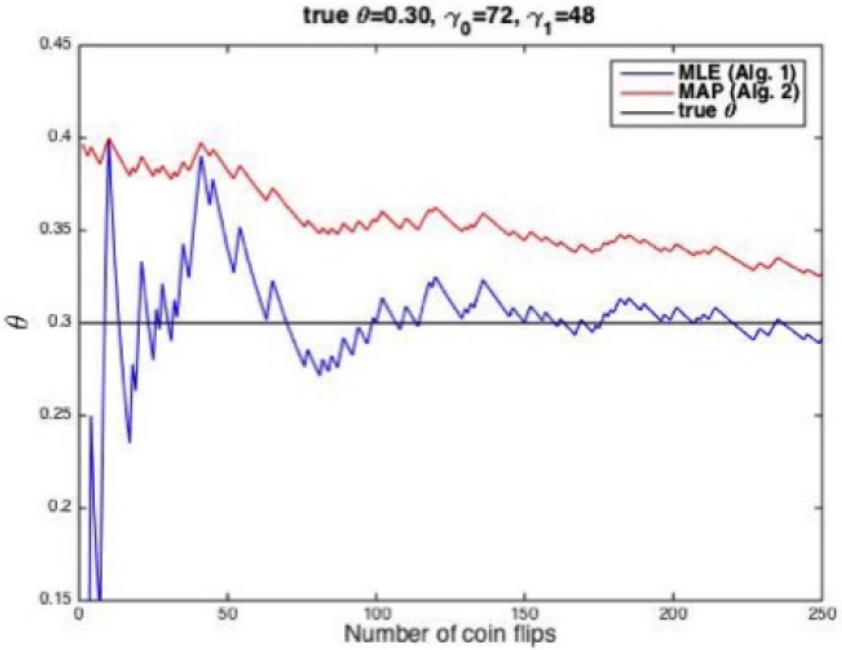
Low Confidence Priors

High Confidence Priors









Naïve Bayes: What you should know

- Designing classifiers based on Bayes rule
- Conditional independence
  - What it is
  - Why it's important
- Naïve Bayes assumption and its consequences
  - Which (and how many) parameters must be estimated under different generative models (different forms for P(X|Y))
    - and why this matters
- How to train Naïve Bayes classifiers
  - MLE and MAP estimates
  - with discrete and/or continuous inputs X_i

Conditional Independence

Definition: X is conditionally independent of Y given Z, if the probability distribution governing X is independent of the value of Y, given the value of Z

$$(\forall i, j, k) P(X = x_i | Y = y_j, Z = z_k) = P(X = x_i | Z = z_k)$$

Which we often write

$$P(X|Y, Z) = P(X|Z)$$

E.g.,

$$P(Thunder|Rain, Lightning) = P(Thunder|Lightning)$$

Naïve Bayes in a Nutshell

Bayes rule:

$$P(Y = y_k | X_1 \dots X_n) = \frac{P(Y = y_k) P(X_1 \dots X_n | Y = y_k)}{\sum_j P(Y = y_j) P(X_1 \dots X_n | Y = y_j)}$$

Assuming conditional independence among X_i's:

$$P(Y = y_k | X_1 \dots X_n) = \frac{P(Y = y_k) \prod_i P(X_i | Y = y_k)}{\sum_j P(Y = y_j) \prod_i P(X_i | Y = y_j)}$$

So, to pick most probable Y for $X^{new} = < X_1, \dots, X_n >$

$$Y^{new} \leftarrow \arg\max_{y_k} \; P(Y = y_k) \prod_i P(X_i^{new} | Y = y_k)$$

Example: Live in Sq Hill? P(S|G,D,E)

- S=1 iff live in Squirrel Hill
- G=1 iff shop at SH Giant Eagle
- W =1 iff Walk or Bike to CMU
- B=1 iff Birthday is before July 1

.26 P(S=1) :

.175 P(W=1 | S=1) :

.71 P(W=1 | S=0) :

.95 P(G=1 | S=1) :

.35 P(G=1 | S=0) :

.45 P(B=1 | S=1) :

.35 P(B=1 | S=0) :

P(S=0) : .74

P(W=0 | S=1) : .825

P(W=0 | S=0) : .29

P(G=0 | S=1) : .05

P(G=0 | S=0) : .65

P(B=0 | S=1) : .55

P(B=0 | S=0) : .65

$S_{new} = \arg\max_{0,1} P(S=S_{new}) P(W=W_{new}|S=S_{new}) P(G=1|S) P(B=1|S)$

$P(S_{new}|G,W,B)$

$S=0 \Rightarrow .74 \quad .29 \quad .35 \quad .35 = .026$

$P(S=1|\ldots) = .26$

$S=1 \Rightarrow .26 \quad .825 \quad .95 \quad .45 = .091$

$$\frac{.026}{.026 + .091}$$

K-Fold Cross Validation

Idea: train multiple times, leaving out a disjoint subset of data each time for test. Average the test set accuracies.

Partition data into K disjoint subsets

For k=1 to K

testData = kth subset

h ← classifier trained on all data except for testData

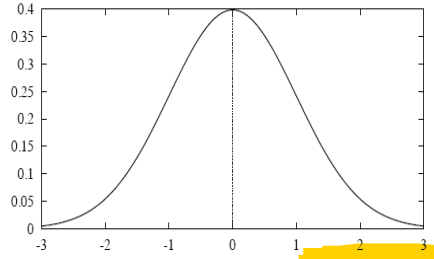
accuracy(k) = accuracy of h on testData

end

FinalAccuracyEstimate = mean of the K recorded testset accuracies

What if we have continuous X_i ?

Gaussian Naïve Bayes (GNB): assume



$$p(X_i = x | Y = y_k) = \frac{1}{\sqrt{2\pi\sigma_{ik}^2}} e^{-\frac{1}{2}\left(\frac{x-\mu_{ik}}{\sigma_{ik}}\right)^2}$$

Sometimes assume variance σ_{ik}

- is independent of Y (i.e., σ_i),
- or independent of X_i (i.e., σ_k)
- or both (i.e., σ)

Gaussian Naïve Bayes Algorithm – continuous X_i
(but still discrete Y)

Suppose $X = \langle X_1 \ldots X_n \rangle$

- Train Naïve Bayes (examples)

  for each value y_k

  estimate* $\pi_k \equiv P(Y = y_k)$

  for each attribute X_i estimate $P(X_i | Y = y_k)$

  - class conditional mean μ_{ik}, variance σ_{ik}

Boolean $X_i \Rightarrow$ # params $= 1 + 2n$

Continuous $X_i \Rightarrow$ # params $= 1 + 4n$

$P(X=1|Y=1)$
$X=0|Y=1$
$X=1|Y=0$
$X=0|Y=0$

- Classify (X^{new})

$$Y^{new} \leftarrow \arg\max_{y_k} \; P(Y = y_k) \prod_i P(X_i^{new} | Y = y_k)$$

$$Y^{new} \leftarrow \arg\max_{y_k} \; \pi_k \prod_i \mathcal{N}(X_i^{new}; \mu_{ik}, \sigma_{ik})$$

* probabilities must sum to 1, so need estimate only n-1 parameters...

What you should know:

- Logistic regression
  - Functional form follows from Naïve Bayes assumptions
    - For Gaussian Naïve Bayes assuming variance $\sigma_{i,k} = \sigma_i$
    - For discrete-valued Naïve Bayes too
  - But training procedure picks parameters without making conditional independence assumption
  - MLE training: pick W to maximize P(Y | X, W)
  - MAP training: pick W to maximize P(W | X,Y)
    - 'regularization'
    - helps reduce overfitting

- Gradient ascent/descent
  - General approach when closed-form solutions unavailable

Derive form for $P(Y|X)$ for Gaussian $P(X_i|Y=y_k)$ assuming $\sigma_{ik} = \sigma_i$

$$P(Y=1|X) = \frac{P(Y=1)P(X|Y=1)}{P(Y=1)P(X|Y=1) + P(Y=0)P(X|Y=0)}$$

$$= \frac{1}{1 + \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)}}$$

$$= \frac{1}{1 + \exp(\ln \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)})}$$

$$= \frac{1}{1 + \exp(\ (\ln \frac{1-\pi}{\pi}) + \sum_i \ln \frac{P(X_i|Y=0)}{P(X_i|Y=1)})}$$

$$P(x \mid y_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} \; e^{\frac{-(x-\mu_{ik})^2}{2\sigma_{ik}^2}}$$

$$\sum_i \left(\frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2} X_i + \frac{\mu_{i1}^2 - \mu_{i0}^2)}{2\sigma_i^2} \right)$$

$$P(Y=1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^{n} w_i X_i)}$$

Very convenient!

$$P(Y=1|X=<X_1,...X_n>)=\frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

implies

$$P(Y=0|X=<X_1,...X_n>)=\frac{exp(w_0+\sum_i w_i X_i)}{1+exp(w_0+\sum_i w_i X_i)}$$

implies

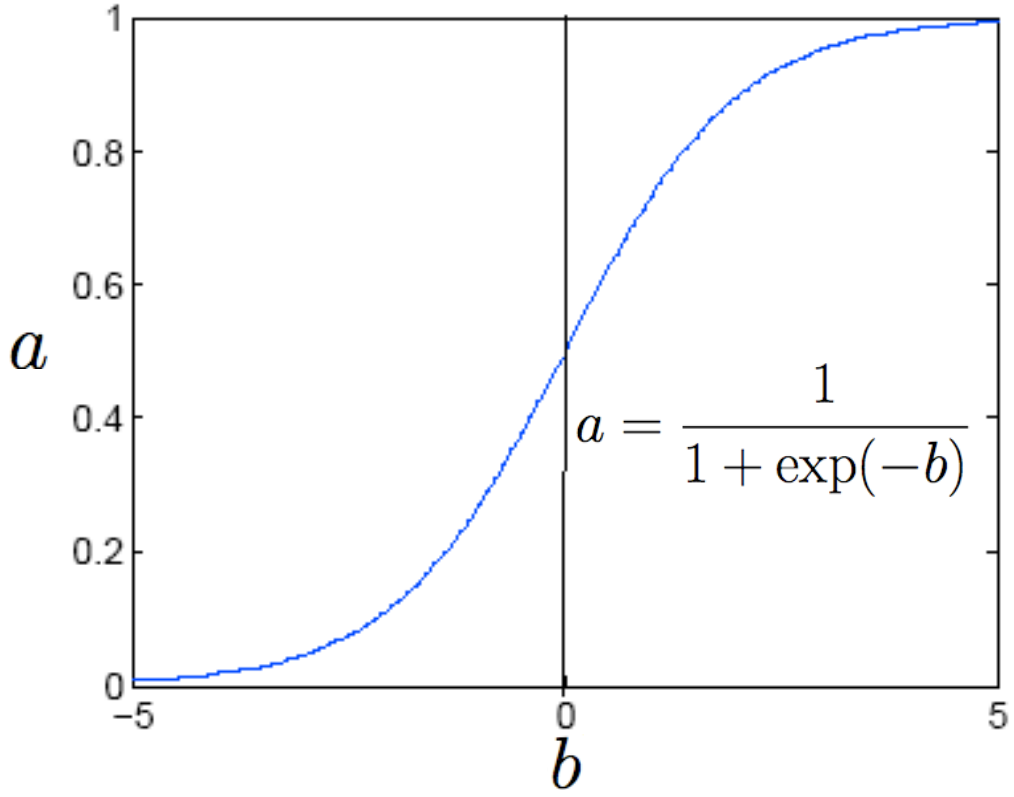
$$\frac{P(Y=0|X)}{P(Y=1|X)}=exp(w_0+\sum_i w_i X_i)$$

linear classification rule!

implies

$$\ln\frac{P(Y=0|X)}{P(Y=1|X)}=w_0+\sum_i w_i X_i$$

Logistic function (also called Sigmoid)



$$P(Y = 1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^{n} w_i X_i)}$$

Training Logistic Regression: MCLE

- Choose parameters W=<w_0, ... w_n> to <u>maximize conditional likelihood</u> of training data

where

$$P(Y = 0|X, W) = \frac{1}{1 + exp(w_0 + \sum_i w_i X_i)}$$

$$P(Y = 1|X, W) = \frac{exp(w_0 + \sum_i w_i X_i)}{1 + exp(w_0 + \sum_i w_i X_i)}$$

- Training data D = $\{\langle X^1, Y^1\rangle, \ldots \langle X^L, Y^L\rangle\}$
- Data likelihood = $\prod_l P(X^l, Y^l|W)$
- Data <u>conditional</u> likelihood = $\prod_l P(Y^l|X^l, W)$

$$W_{MCLE} = \arg\max_W \prod_l P(Y^l|W, X^l)$$

Maximizing Conditional Log Likelihood

$$P(Y = 0|X, W) = \frac{1}{1 + exp(w_0 + \sum_i w_i X_i)}$$

$$P(Y = 1|X, W) = \frac{exp(w_0 + \sum_i w_i X_i)}{1 + exp(w_0 + \sum_i w_i X_i)}$$

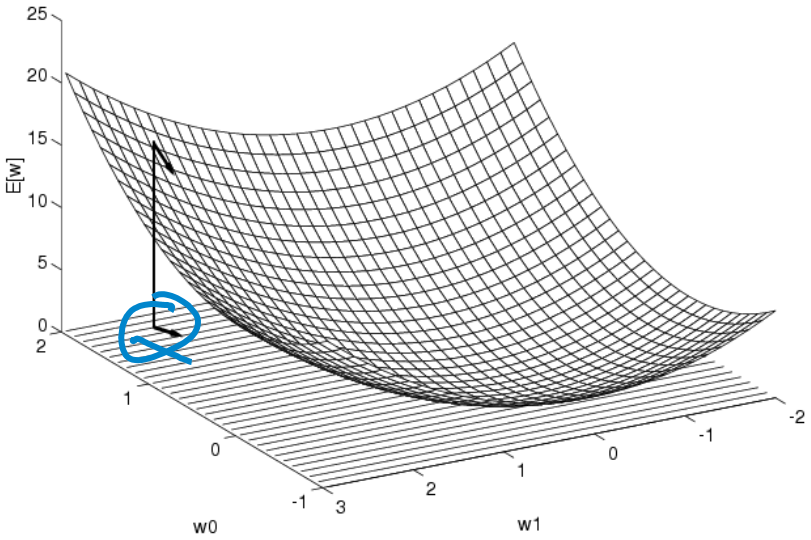
$$l(W) \equiv \ln \prod_l P(Y^l|X^l, W) = \sum_l \ln P(Y^l|X^l, W)$$

$$= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + exp(w_0 + \sum_i^n w_i X_i^l))$$

Good news: $l(W)$ is concave function of W

Bad news: no closed-form solution to maximize $l(W)$

Gradient Descent



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

MLE vs MAP

- Maximum conditional likelihood estimate

$$W \leftarrow \arg\max_W \ \ln \prod_l P(Y^l|X^l, W)$$

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))$$

- Maximum a posteriori estimate with prior W~N(0,σI)

$$W \leftarrow \arg\max_W \ \ln[P(W) \ \prod_l P(Y^l|X^l, W)]$$

$$w_i \leftarrow w_i - \eta\lambda w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))$$

Gradient Descent:

Batch gradient: use error $E_D(\mathbf{w})$ over entire training set D

Do until satisfied:

1. Compute the gradient $\nabla E_D(\mathbf{w}) = \left[\frac{\partial E_D(\mathbf{w})}{\partial w_0} \ldots \frac{\partial E_D(\mathbf{w})}{\partial w_n}\right]$
2. Update the vector of parameters: $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E_D(\mathbf{w})$

Stochastic gradient: use error $E_d(\mathbf{w})$ over single examples $d \in D$

Do until satisfied:

1. Choose (with replacement) a random training example $d \in D$
2. Compute the gradient just for d: $\nabla E_d(\mathbf{w}) = \left[\frac{\partial E_d(\mathbf{w})}{\partial w_0} \ldots \frac{\partial E_d(\mathbf{w})}{\partial w_n}\right]$
3. Update the vector of parameters: $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E_d(\mathbf{w})$

Stochastic approximates Batch arbitrarily closely as $\eta \rightarrow 0$

Stochastic can be much faster when D is very large

Intermediate approach: use error over subsets of D

MAP estimates and Regularization

- Maximum a posteriori estimate with prior W~N(0,σI)

$$W \leftarrow \arg\max_W \ \ln[P(W) \ \prod_l P(Y^l|X^l,W)]$$

$$w_i \leftarrow w_i - \eta\lambda w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))$$

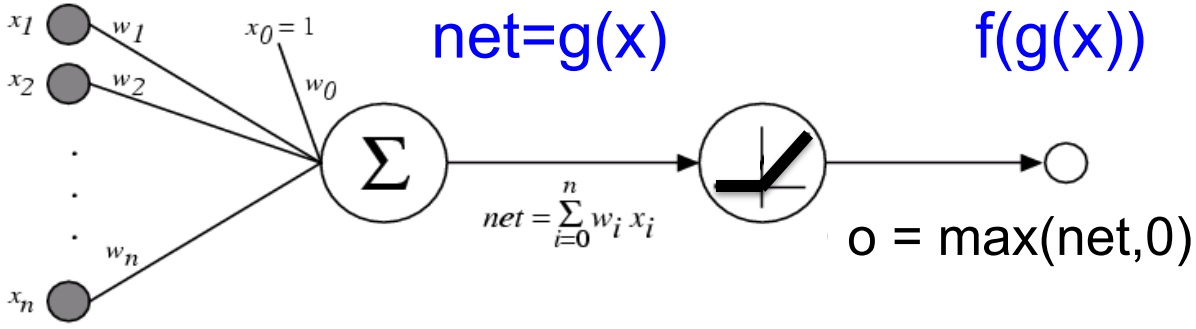
called a "regularization" term

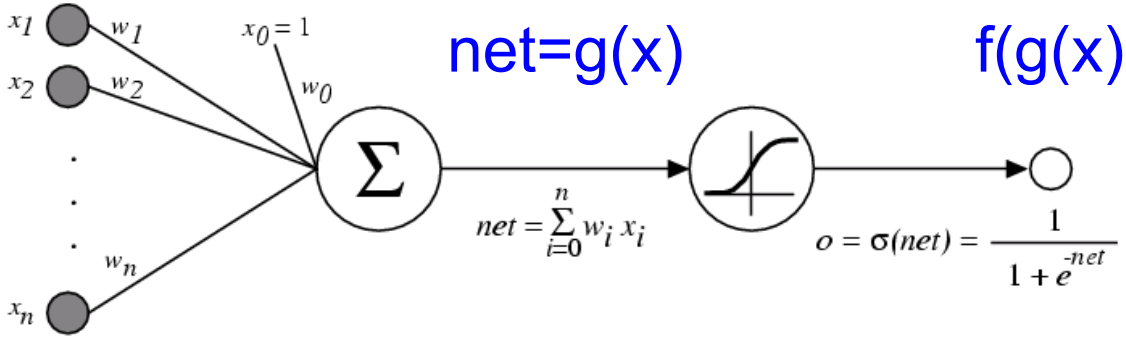
- helps reduce overfitting
- if P(W) is Gaussian, then encourages W to be near the mean of P(W) : zero here, but can easily use any mean
- used very frequently in Logistic Regression

What you should know:

- Backpropagation algorithm
  - training network with gradient descent
  - how to derive gradient for simple cost functions, units
  - relationship to logistic regression
- Neural nets learn internal representations
- Convolutional neural networks
  - convolution operation
  - see new Homework

Units often composed so output o = f(g(x))
f is called the “activation function”





Many types of parameterized units

- Sigmoid units
- ReLU
- Leaky ReLU (fixed non-zero slope for input<0)
- Parametric ReLU (trainable slope)
- Max Pool
- Inner Product
- GRU's
- LSTM's
- Matrix multiply
- …. no end in sight

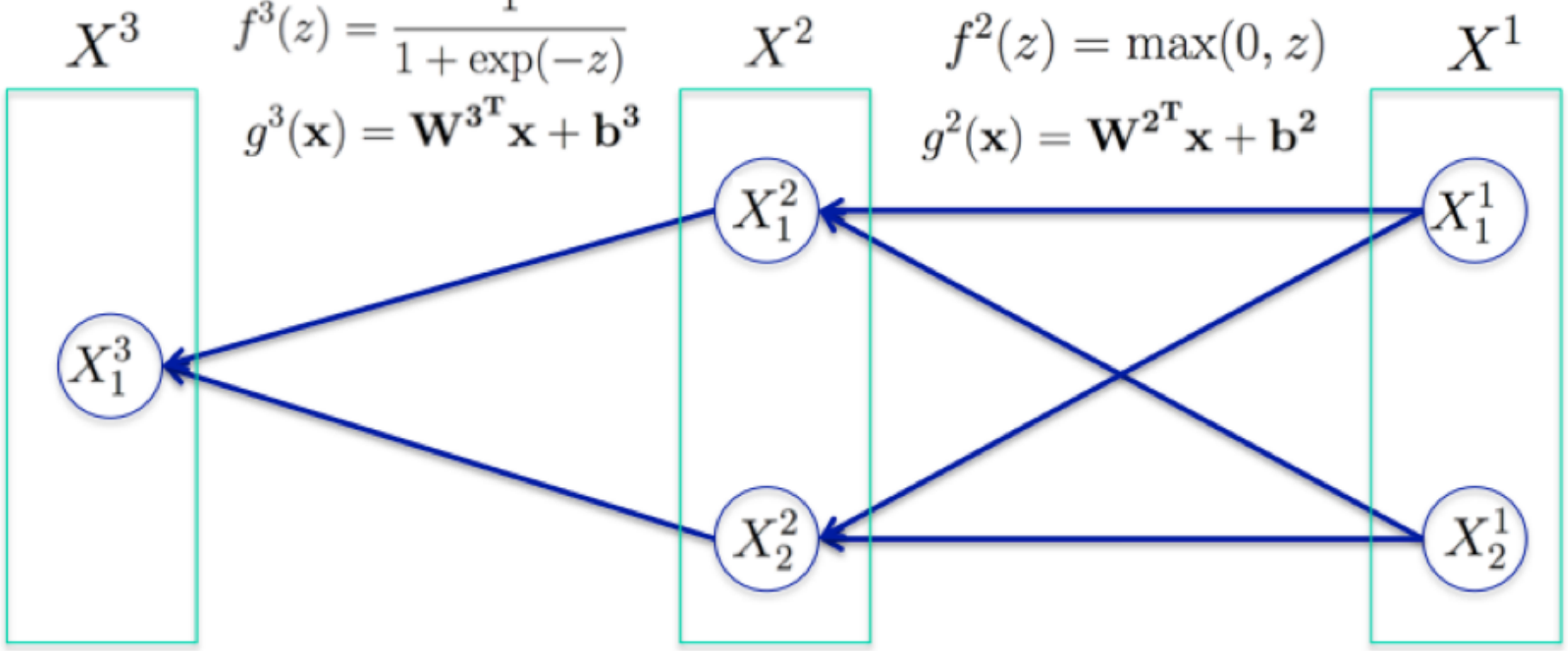
Any unit h(X; W) that is differentiable w.r.t. X and W

Training Deep Nets

1. Choose loss function $J(\theta)$ to optimize
   - sum of squared errors for y continuous: $\Sigma\ (y - h(x; \theta))^2$
   - maximize conditional likelihood: $\Sigma \log P(y|x; \theta)$
   - MAP estimate: $\Sigma \log P(y|x; \theta)\ P(\theta)$
   - ~~0/1 loss. Sum of classification errors: $\Sigma\ \delta(y = h(x; \theta)$~~
   - …
2. Design network architecture
   - Network of layers (ReLU's, sigmoid, convolutions, ...)
   - Widths of layers
   - Fully or partly interconnected
   - ...
3. Training algorithm
   - Derive gradient formulas Ê
   - Choose gradient descent method, including stopping condition
   - Experiment with alternative architectures

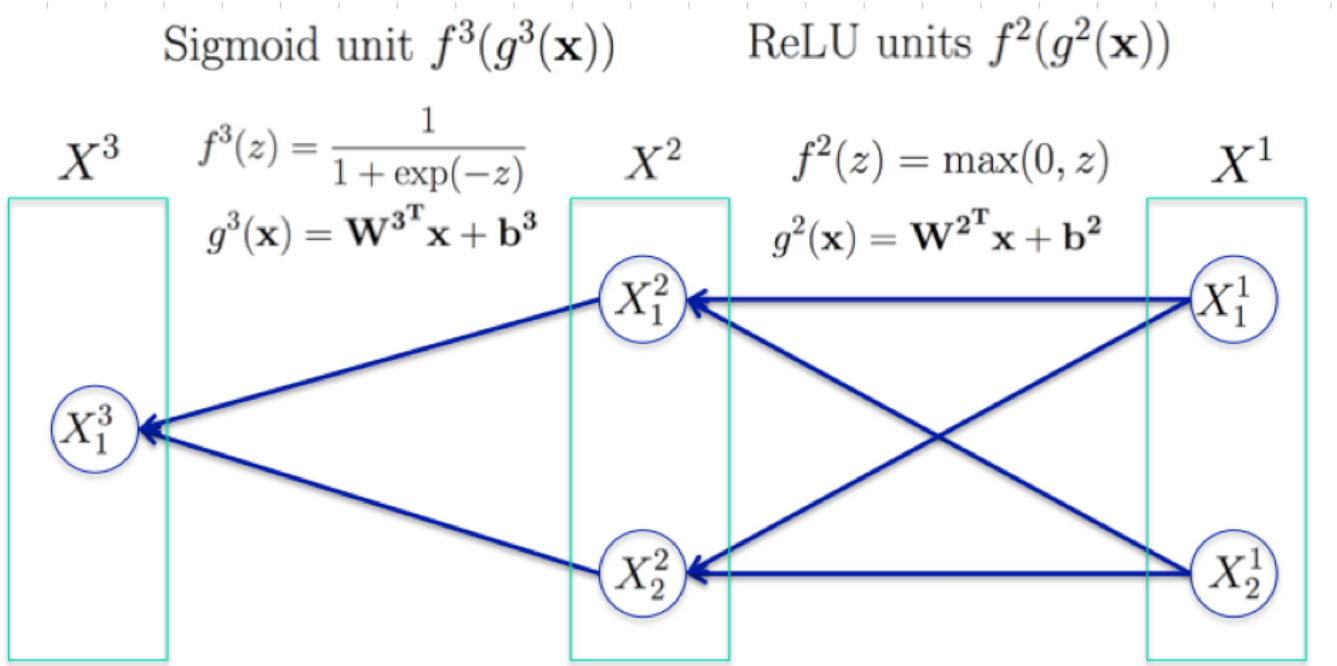
Feed Forward

Sigmoid unit $f^3(g^3(\mathbf{x}))$ ReLU units $f^2(g^2(\mathbf{x}))$



X^3	$g^3(X^2)$	W^{3T}		b^3	X^2	$g^2(X^1)$	W^{2T}		b^2	X^1
0.53	0.12	0.10	-0.09	0.10	0.20	0.20	0.10	-0.10	0.10	1
					0.00	-0.25	-0.20	0.10	-0.05	0
					1					*1*

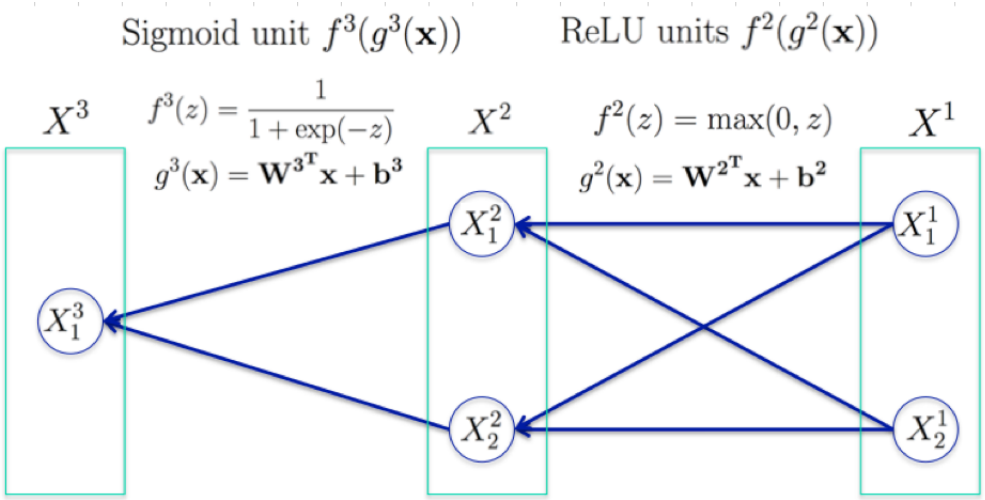
Back propagation



Ytrue	1		
dJ/dX3	-1.89		
dJ/dg3	-0.47		
dJ/dW3T	-0.09	0.00	-0.47
dJ/dX2	-0.05		
	0.04		
dJ/dg2	-0.05		
	0.00		
dJ/dW2T	-0.05	0.00	-0.05
	0.00	0.00	0.00

X^3	$g^3(X^2)$	W^{3T}		b^3	X^2	$g^2(X^1)$	W^{2T}		b^2	X^1
0.53	0.12	0.10	-0.09	0.10	0.20	0.20	0.10	-0.10	0.10	1
					0.00	-0.15	-0.20	0.10	0.05	0
					1					***1***

update each parameter according to $\theta_i \leftarrow \quad \theta_i - \eta \frac{\partial J(\theta)}{\partial \theta_i}$



$$\frac{\partial J(\theta)}{\partial g_1^3} = \frac{\partial J(\theta)}{\partial X_1^3}\frac{\partial X_1^3}{\partial g^3} = \frac{\partial J(\theta)}{\partial X_1^3} X_1^3(1-X_1^3)$$

$$\frac{\partial J(\theta)}{\partial w_i^3} = \frac{\partial J(\theta)}{\partial g^3}\frac{\partial g^3}{\partial w_i^3} = \frac{\partial J(\theta)}{\partial g^3} X_i^2$$

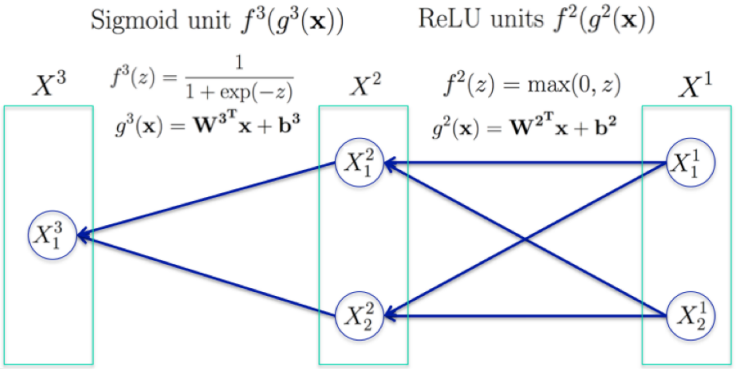
$$\frac{\partial J(\theta)}{\partial X_i^2} = \frac{\partial J(\theta)}{\partial g^3}\frac{\partial g^3}{\partial X_i^2} = \frac{\partial J(\theta)}{\partial g^3} w_i^3$$

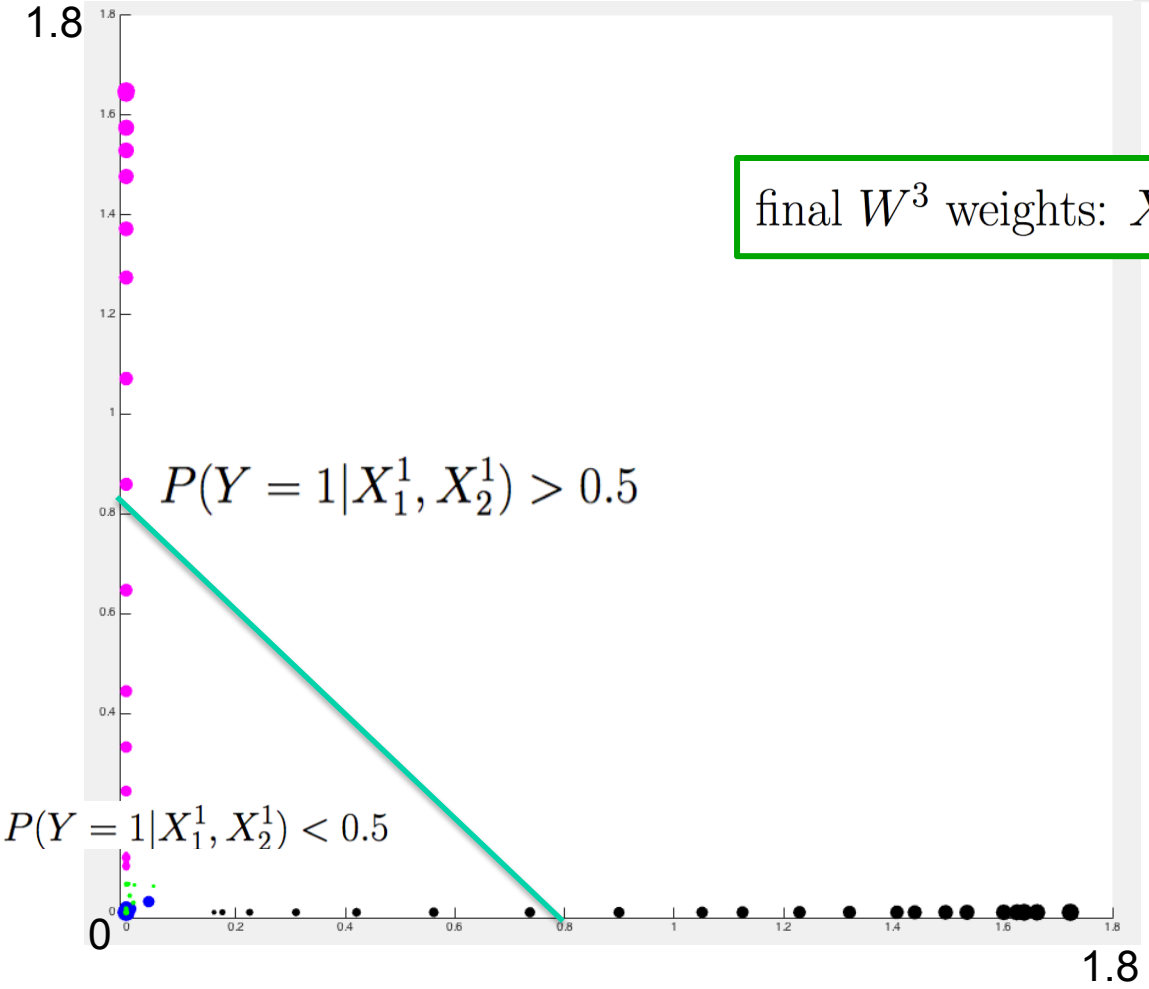
$$\frac{\partial J(\theta)}{\partial g_i^2} = \frac{\partial J(\theta)}{\partial X_i^2}\frac{\partial X_i^2}{\partial g_i^2} = \frac{\partial J(\theta)}{\partial X_i^2} \times [\text{if } g_i^2 > 0 \text{ then 1 else 0}]$$

$$\frac{\partial J(\theta)}{\partial w_{ik}^2} = \frac{\partial J(\theta)}{\partial g_i^2}\frac{\partial g_i^2}{\partial X_k^1} = \frac{\partial J(\theta)}{\partial g^3} X_k^1$$

Final decision surface in terms of X^2

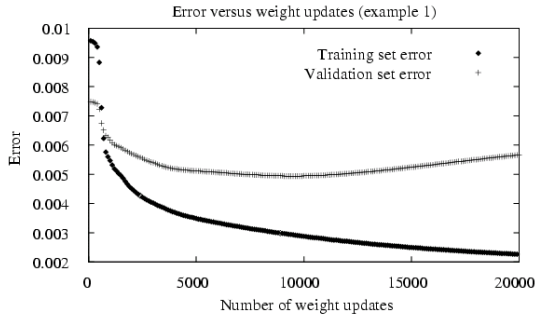
Input: [0 1] [1 1] [1 0] [0 0]





final W^3 weights: $X^3 = 2.417\ X_1^2 + 2.346\ X_2^2 - 1.908$

Dealing with Overfitting



Our learning algorithm involves a parameter

n=number of gradient descent iterations

How do we choose n to optimize future error or loss?

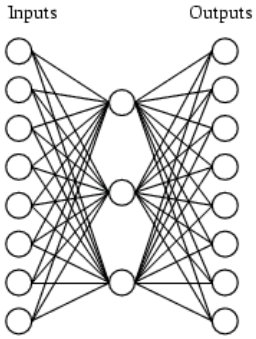
- Separate available data into <u>training</u> and <u>validation</u> set
- Use <u>training</u> to perform gradient descent
- n ← number of iterations that optimizes <u>validation</u> set error

→ *gives unbiased estimate of optimal n*

(but still an optimistically <u>biased</u> estimate of true error)

Learning Hidden Layer Representations

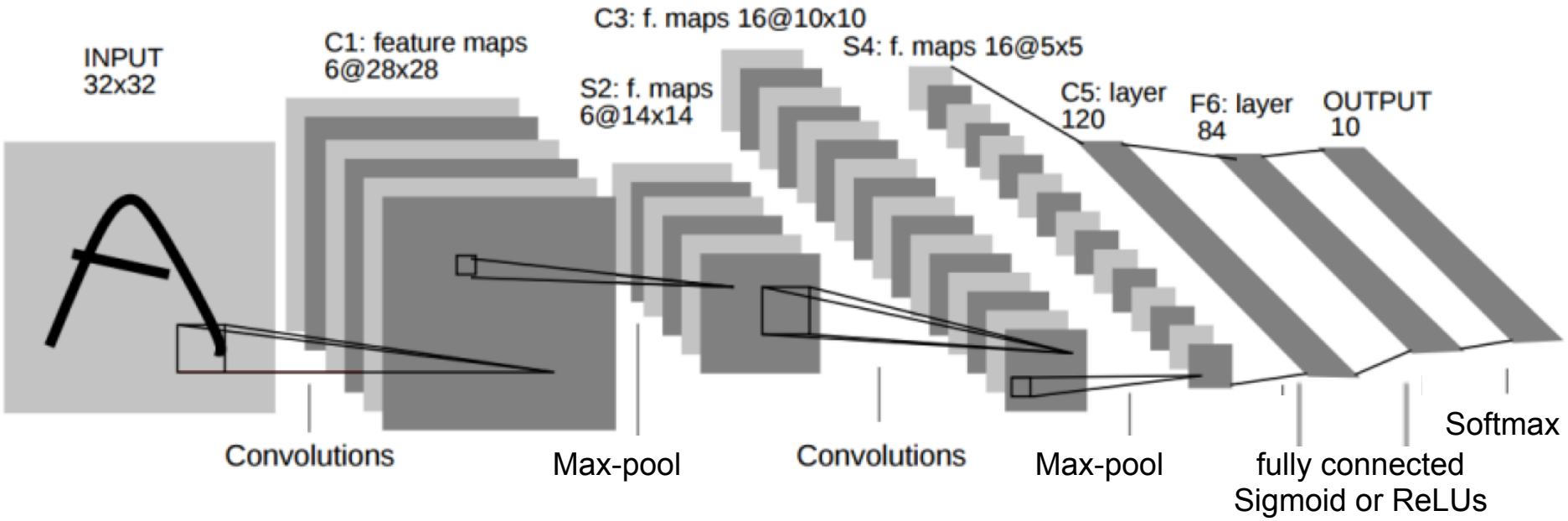
A network:



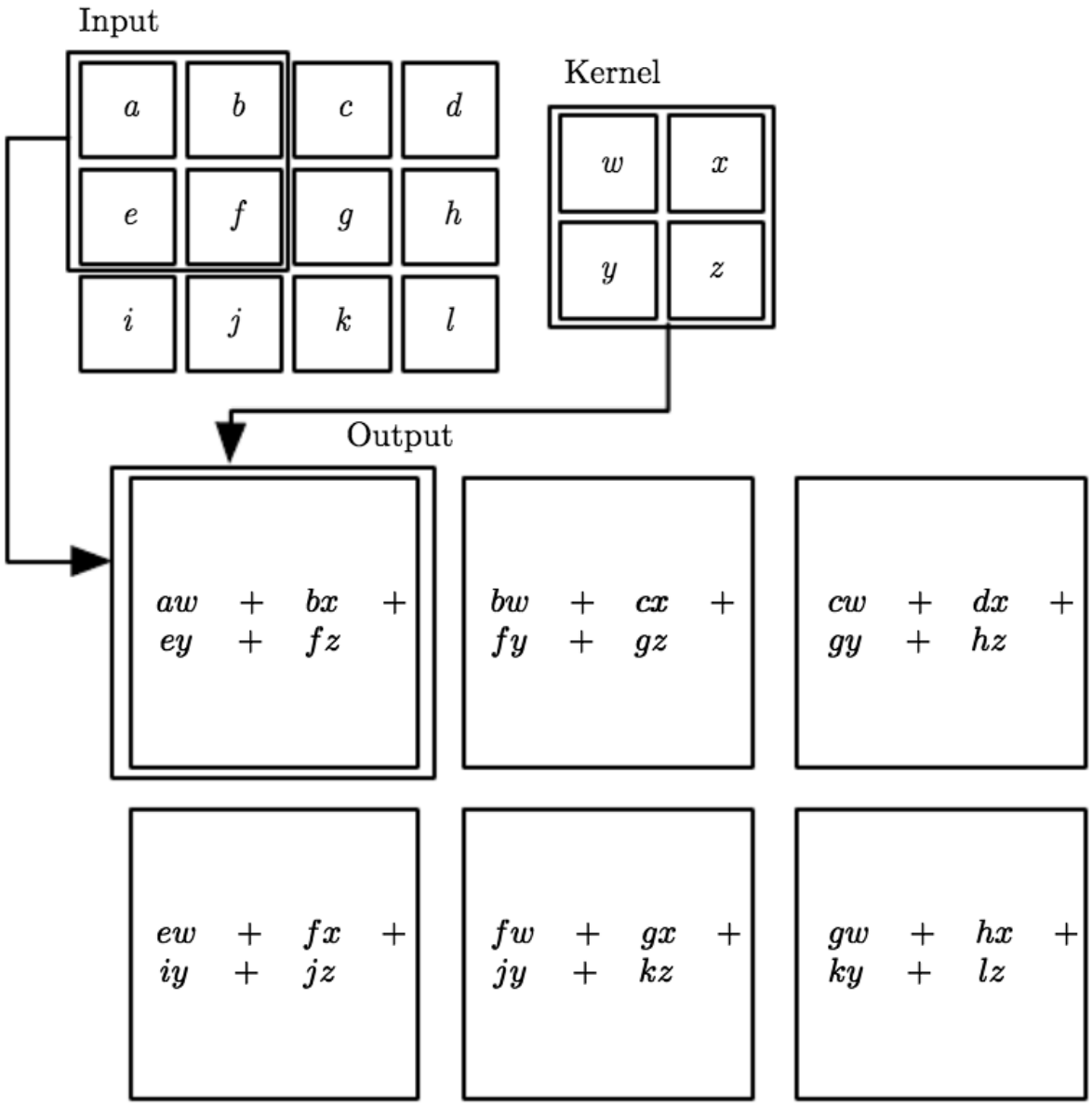
Learned hidden layer representation:

Input		Hidden Values				Output
10000000	$\rightarrow$	.89	.04	.08	$\rightarrow$	10000000
01000000	$\rightarrow$	.01	.11	.88	$\rightarrow$	01000000
00100000	$\rightarrow$	.01	.97	.27	$\rightarrow$	00100000
00010000	$\rightarrow$	.99	.97	.71	$\rightarrow$	00010000
00001000	$\rightarrow$	.03	.05	.02	$\rightarrow$	00001000
00000100	$\rightarrow$	.22	.99	.99	$\rightarrow$	00000100
00000010	$\rightarrow$	.80	.01	.98	$\rightarrow$	00000010
00000001	$\rightarrow$	.60	.94	.01	$\rightarrow$	00000001

A Convolutional Neural Net for Handwritten Digit recognition: LeNet



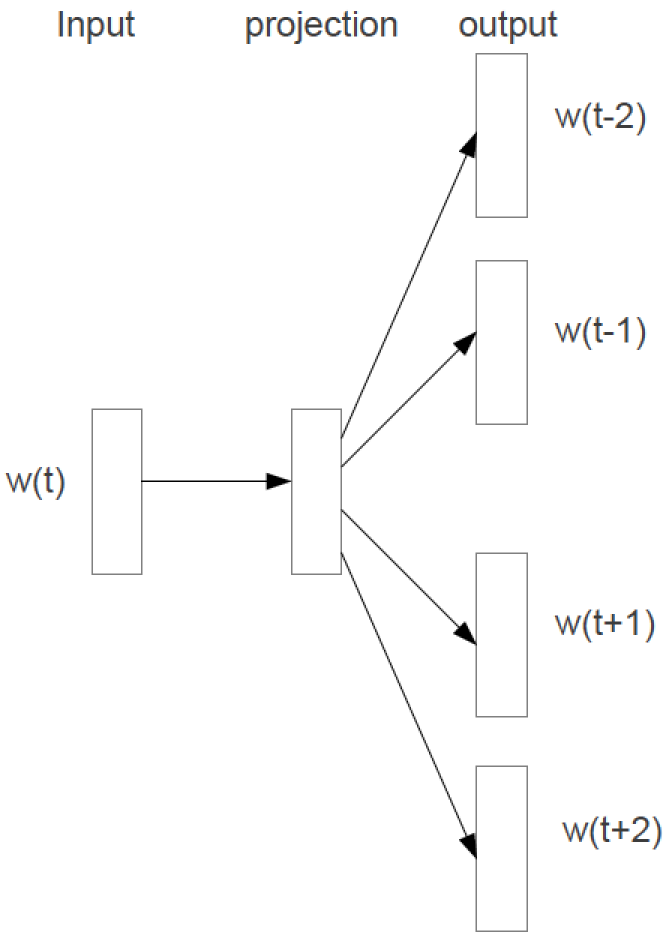
Convolution



$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

[from Goodfellow et al.]

Word2Vec Word Embeddings



basic skip gram model:

train to maximize:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

where

$$p(w_O|w_I)=\frac{\exp\left({v'_{w_O}}^{\top}v_{w_I}\right)}{\sum_{w=1}^{W}\exp\left({v'_w}^{\top}v_{w_I}\right)}$$

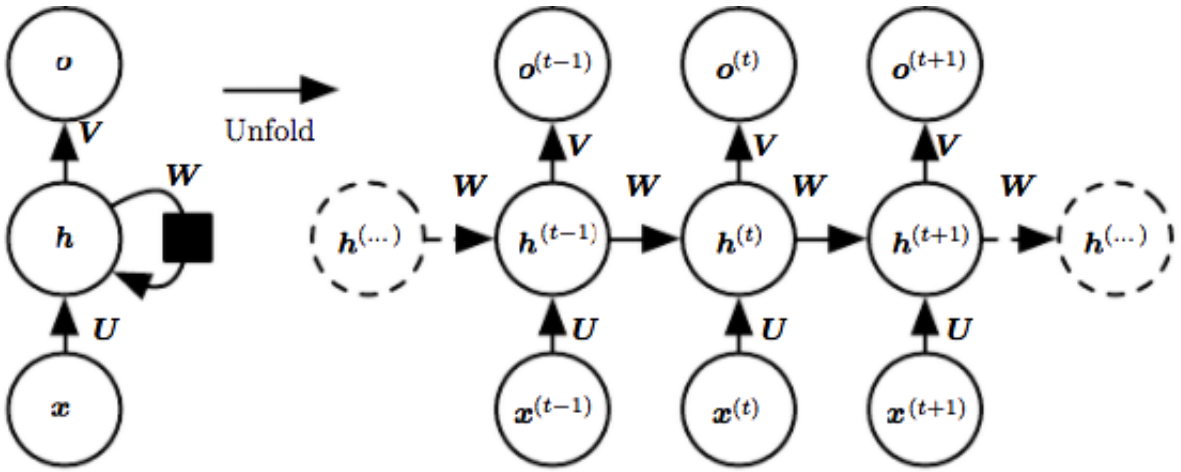
optimized…
+ hierarchical softmax
+ negative sampling
+ subsample frequent w's

[Mikolav et al., 2013]

Training Recurrent Networks

Key principle for training:

1. Treat as if unfolded in time, resulting in directed acyclic graph
2. Note shared parameters in unfolded net → sum the gradients



* problem: vanishing and/or exploding gradients

[Goodfellow et al., 2016]

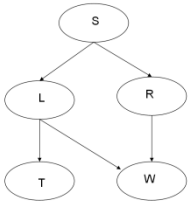
Graphical Models

- Key Idea:
  - Conditional independence assumptions useful
  - but Naïve Bayes is extreme!
  - Graphical models express sets of conditional independence assumptions via graph structure
  - Graph structure plus associated parameters define *joint probability distribution over set of variables*

- Two types of graphical models:
  - Directed graphs (aka Bayesian Networks)
  - Undirected graphs (aka Markov Random Fields)

our focus

Bayesian Networks Definition



Parents	P(W|Pa)	P(¬W|Pa)
L, R	0	1.0
L, ¬R	0	1.0
¬L, R	0.2	0.8
¬L, ¬R	0.9	0.1

A Bayes network represents the joint probability distribution over a collection of random variables

A Bayes network is a directed acyclic graph and a set of conditional probability distributions (CPD's)

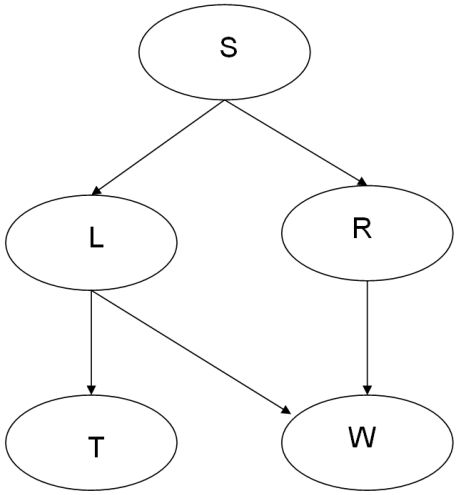
- Each node denotes a random variable
- Edges denote dependencies
- For each node X_i its CPD defines $P(X_i \mid Pa(X_i))$
- The joint distribution over all variables is defined to be

$$P(X_1 \ldots X_n) = \prod_i P(X_i | Pa(X_i))$$

Pa(X) = immediate parents of X in the graph

Bayesian Networks

- CPD for each node X_i describes $P(X_i \mid Pa(X_i))$



Parents	P(W|Pa)	P(¬W|Pa)
L, R	0	1.0
L, ¬R	0	1.0
¬L, R	0.2	0.8
¬L, ¬R	0.9	0.1

W

Chain rule of probability says that in general:

$$P(S,L,R,T,W) = P(S)P(L|S)P(R|S,L)P(T|S,L,R)P(W|S,L,R,T)$$

But in a Bayes net: $P(X_1 \ldots X_n) = \prod_i P(X_i|Pa(X_i))$

$P(S)\,P(L|S)\,P(R|S)\,P(T|L)\,P(W|L,R)$

What You Should Know

- Bayes nets are convenient representation for encoding dependencies / conditional independence
- BN = Graph plus parameters of CPD's
  - Defines joint distribution over variables
  - Can calculate everything else from that
  - Though inference may be intractable
- Reading conditional independence relations from the graph
  - Each node is cond indep of non-descendents, given only its parents
  - 'Explaining away'

X and Y are conditionally independent given Z, **if and only if** X and Y are D-separated by Z.

[Bishop, 8.2.2]

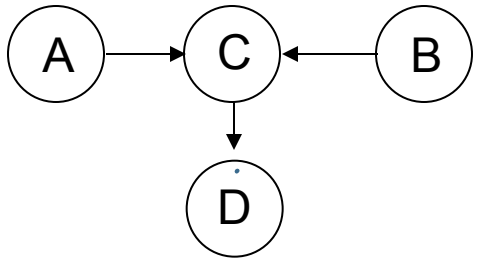
Suppose we have three sets of random variables: X, Y and Z

X and Y are **D-separated** by Z (and therefore conditionally indep, given Z) iff every path from every variable in X to every variable in Y is **blocked**

A path from variable X to variable Y is **blocked** if it includes a node such that *either* of the following holds:

1. arrows on the path meet either head-to-tail or tail-to-tail at the node and this node is in Z

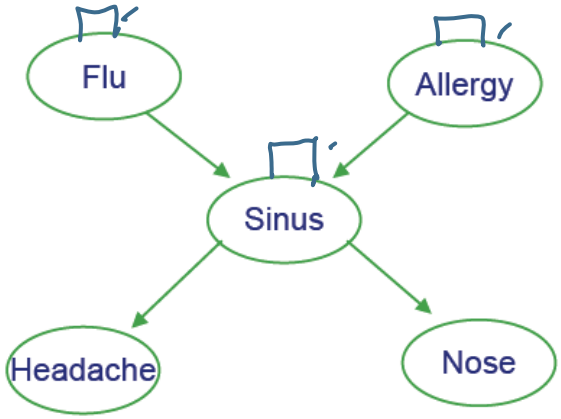
2. the arrows meet head-to-head at the node, and neither the node, nor any of its descendants, is in Z



What you should know: Inference in Bayes Nets

- In general, intractable (NP-complete)
- Probability for a given joint assignment
- Probability for one unobserved variable given all the others
- Conditional independence / D-separation
- Markov blanket
- How to use Markov blanket to simplify inference
- How to generate samples from joint distribution
  - and how to use samples to estimate anything
- Gibbs sampling

Prob. of joint assignment: easy



- Suppose we are interested in joint assignment <F=f,A=a,S=s,H=h,N=n>

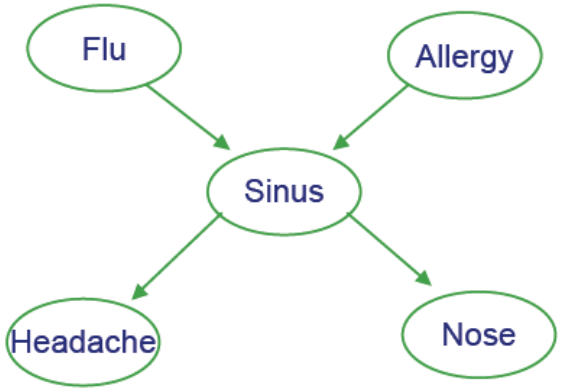
var value

What is P(f,a,s,h,n)?

$P(f)\,P(a)\,P(s|f,a)\,P(h|s)\,P(n|s)$

let's use p(a,b) as shorthand for p(A=a, B=b)

Generating a sample from joint distribution: easy



How can we generate random samples drawn according to P(F,A,S,H,N)?

To generate a random sample for roots of network (F or A):

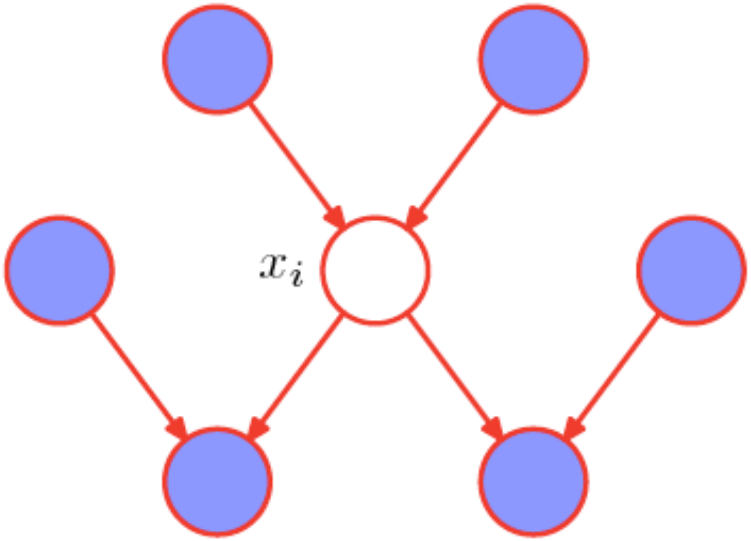
1. let θ = P(F=1) # look up from CPD
2. r = random number drawn uniformly between 0 and 1
3. if r<θ then output 1, else 0

To generate a random sample for S, given F,A:

1. let θ = P(S=1|F=f,A=a) # look up from CPD
2. r = random number drawn uniformly between 0 and 1
3. if r<θ then output 1, else 0

Markov Blanket

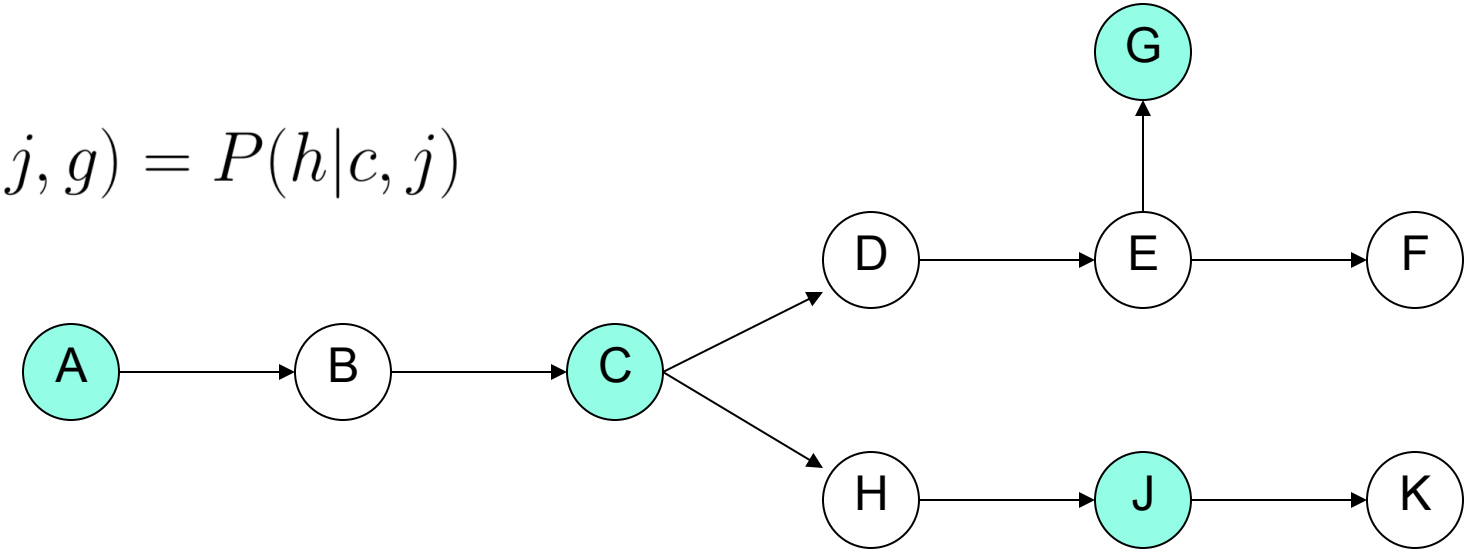
The Markov blanket of a node $\mathbf{x}_i$ comprises the set of parents, children and co-parents of the node. It has the property that the conditional distribution of $\mathbf{x}_i$, conditioned on all the remaining variables in the graph, is dependent only on the variables in the Markov blanket.



from [Bishop, 8.2]

Why Markov Blanket is Useful for Inference

$$P(h|a,c,j,g) = P(h|c,j)$$



$$P(h|c,j) = \frac{P(h,c,j)}{P(c,j)} = \frac{P(c)P(h|c)P(j|h)}{P(c)P(h|c)P(j|h) + P(c)P(\neg h|c)P(j|\neg h)}$$

$$= \frac{P(h|c)P(j|h)}{P(h|c)P(j|h) + P(\neg h|c)P(j|\neg h)}$$

let's use shorthand P(a) to represent P(A=a)

Learning of Bayes Nets

- Four categories of learning problems
  - Graph structure may be known/unknown
  - Variable values may be fully observed / partly unobserved

- Easy case: learn parameters when graph structure is *known*, and training data is *fully observed*

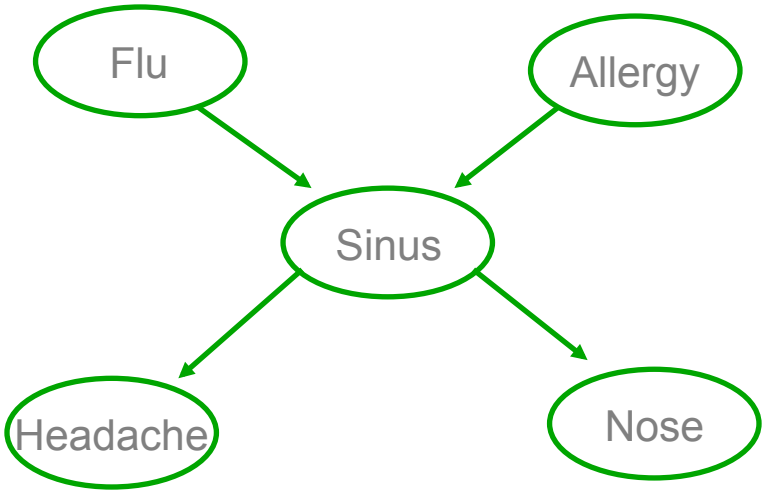
- Interesting case: graph *known*, data *partly observed*

- Gruesome case: graph structure *unknown*, data *partly unobserved*

Learning CPTs from Fully Observed Data

- Example: Consider learning the parameter

$$\theta_{s|ij} \equiv P(S = 1 | F = i, A = j)$$



- MLE (Max Likelihood Estimate) is

$$\theta_{s|ij} = \frac{\sum_{k=1}^{K} \delta(f_k = i, a_k = j, s_k = 1)}{\sum_{k=1}^{K} \delta(f_k = i, a_k = j)}$$

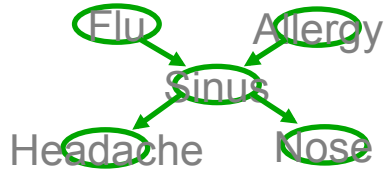
k^{th} training example

δ(X) = 1 if X is true
0 otherwise

- Remember why?

let's use a_k to represent value of A on the kth example

EM Algorithm - Informally



EM is a general procedure for learning from partly observed data

Given observed variables X, unobserved Z (X={F,A,H,N}, Z={S})

Begin with arbitrary choice for parameters θ

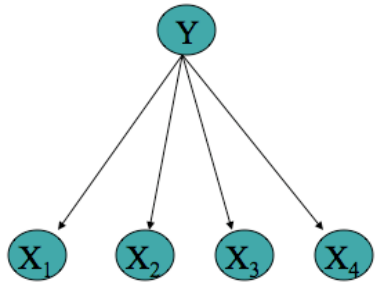
Iterate until convergence:

- E Step: use X, θ to estimate the unobserved Z values
- M Step: use X values and estimated Z values to derive a better θ

Guaranteed to find local maximum.

Each iteration increases $E_{P(Z|X,\theta)}[\log P(X, Z|\theta')]$

EM and estimating θ



Given observed set X, unobserved set Y of boolean values

E step: Calculate for each training example, k

the expected value of each unobserved variable Y

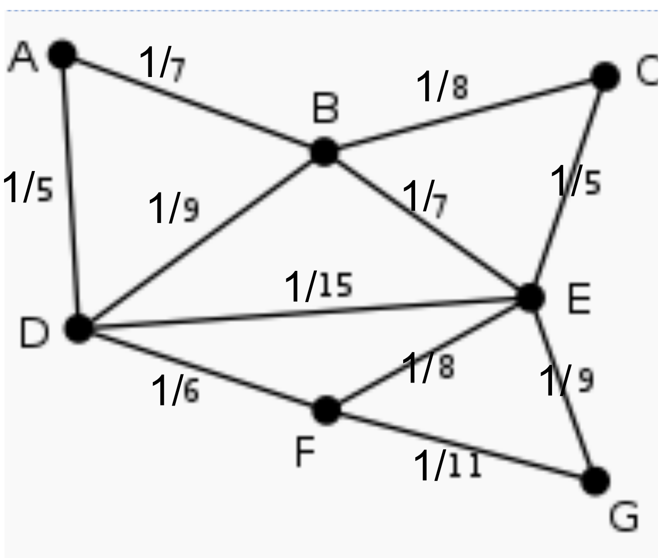
$$E_{P(Y|X_1 \ldots X_N)}[y(k)] = P(y(k)=1|x_1(k), \ldots x_N(k); \theta) = \frac{P(y(k)=1)\prod_i P(x_i(k)|y(k)=1)}{\sum_{j=0}^{1} P(y(k)=j)\prod_i P(x_i(k)|y(k)=j)}$$

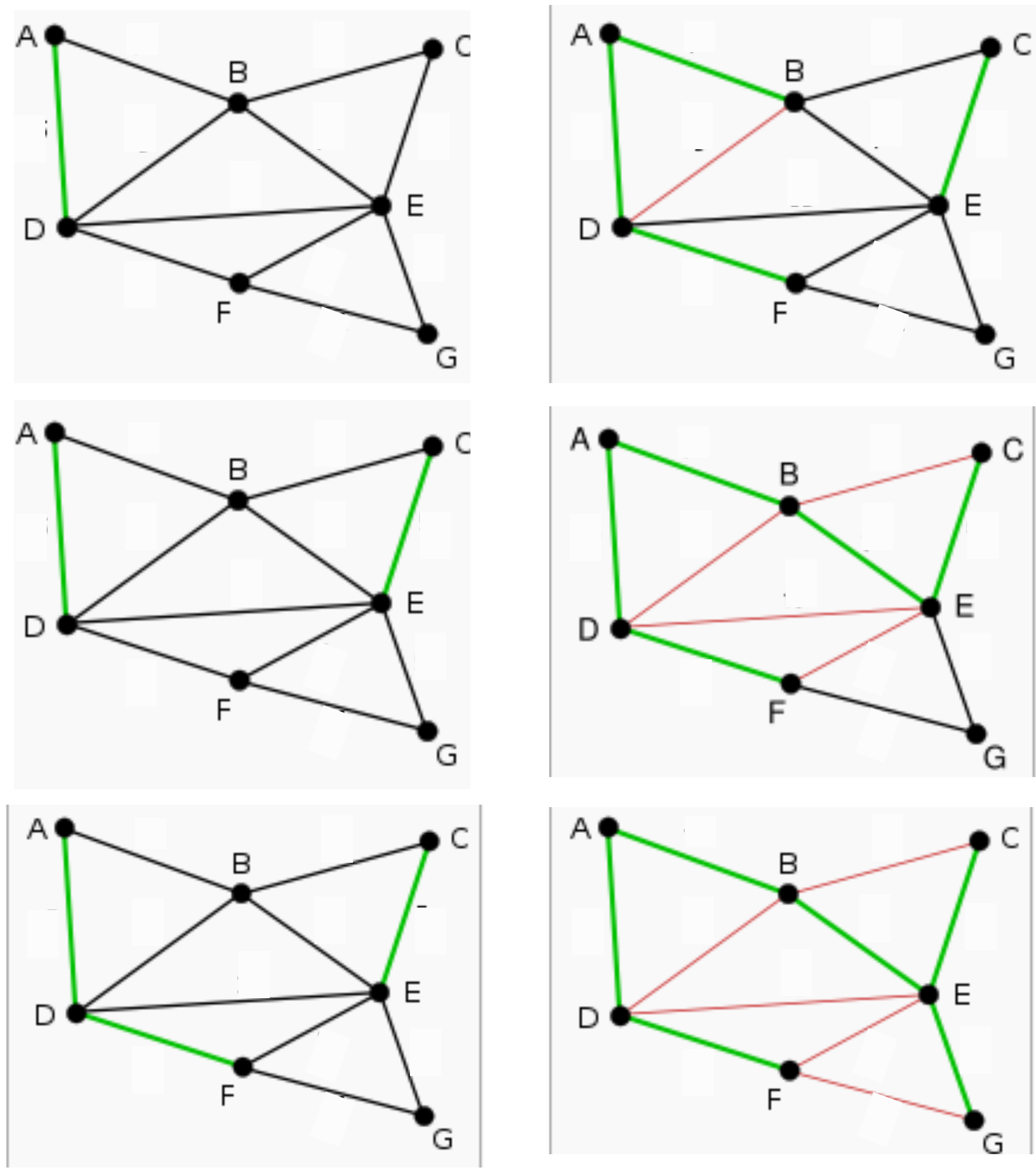
M step: Calculate estimates similar to MLE, but replacing each count by its expected count

$$\theta_{ij|m} = \hat{P}(X_i = j|Y = m) = \frac{\sum_k P(y(k)=m|x_1(k)\ldots x_N(k))\ \delta(x_i(k)=j)}{\sum_k P(y(k)=m|x_1(k)\ldots x_N(k))}$$

MLE would be: $\hat{P}(X_i = j|Y = m) = \frac{\sum_k \delta((y(k)=m) \wedge (x_i(k)=j))}{\sum_k \delta(y(k)=m)}$

Chow-Liu algorithm example
Greedy Algorithm to find Max-Spanning Tree





[courtesy A. Singh, C. Guestrin]

Bayes Nets – What You Should Know

- Representation
  - Bayes nets represent joint distribution as a DAG + Conditional Distributions
  - D-separation lets us decode conditional independence assumptions
- Inference
  - NP-hard in general
  - For some graphs, closed form inference is feasible
  - Approximate methods too, e.g., Monte Carlo methods, …
- Learning
  - Easy for known graph, fully observed data (MLE's, MAP est.)
  - EM for partly observed data, known graph
  - Learning graph structure: Chow-Liu for tree-structured networks
  - Hardest when graph unknown, data incompletely observed