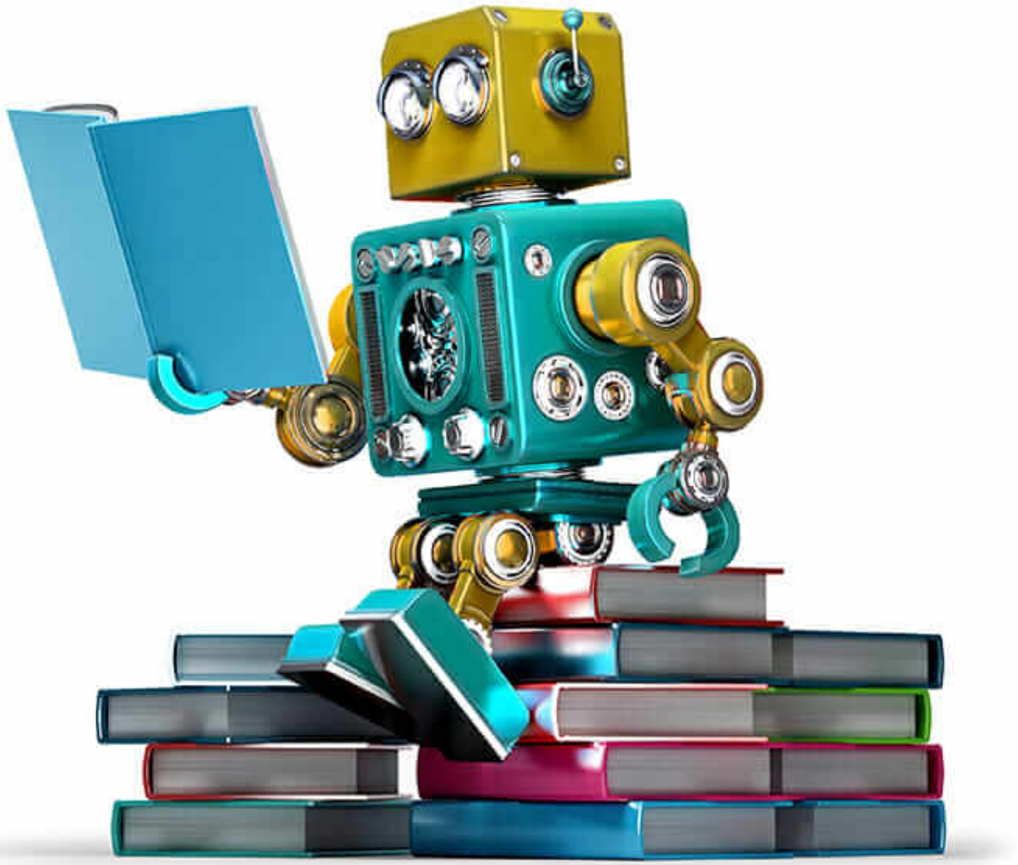


JENNIFER NEVILLE
DEPARTMENTS OF COMPUTER SCIENCE AND STATISTICS
PURDUE UNIVERSITY

INTRODUCTION TO MACHINE LEARNING

MACHINE LEARNING 101

1 Data representation

Choose

2 Knowledge representation

Choose

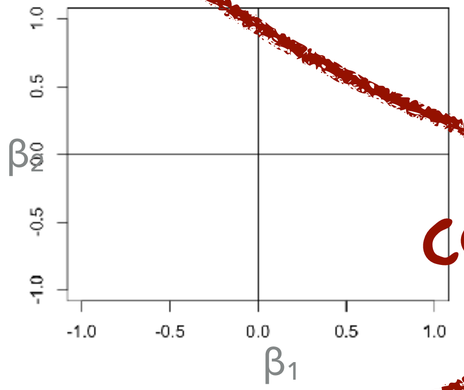
$$y = \beta_1 x_1 + \beta_2 x_2 \ldots + \beta_0$$

defines

MACHINE LEARNING 101

Model space

defines



Combine

3 objective function

$$L_{sq}(D) = \frac{1}{N} \sum_{i=1}^{|D|} (f(\mathbf{x}_i) - y_i)^2$$

Choose

MACHINE LEARNING 101

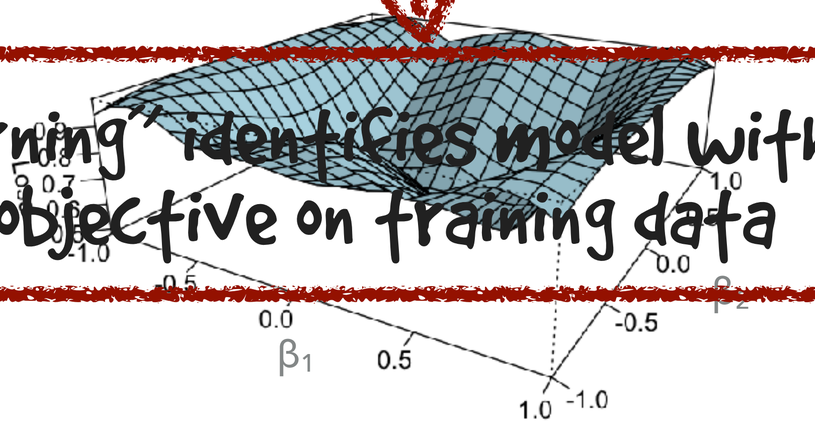
4

combine

Search algorithm

(eg. optimization)

"Learning" identifies model with best objective on training data



Assumption: if learned model is applied to (test) data drawn from same distribution, it will generalize

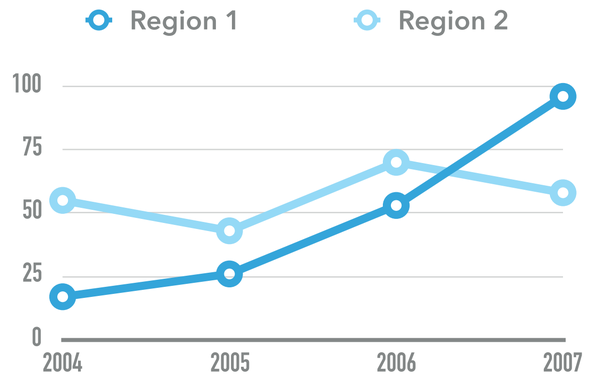
X

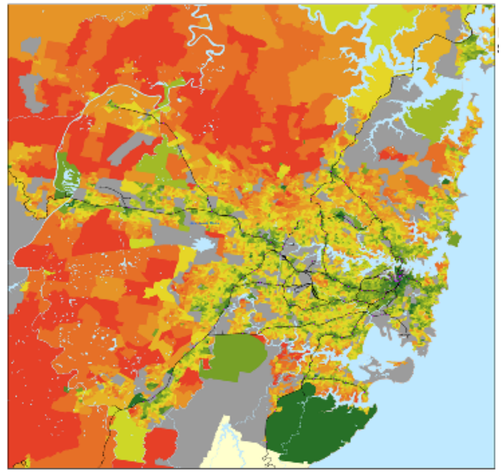
β_1=0.5, β_2=0.1

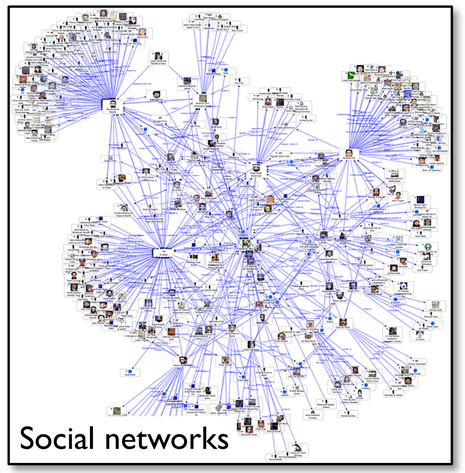
MACHINE LEARNING 101

1. Data representation
2. Knowledge representation
3. objective function
4. Search algorithm

Fraud	Age	Degree	StartYr	Series7
+	22	Y	2005	N
-	25	N	2003	Y
-	31	Y	1995	Y
-	27	Y	1999	Y
+	24	N	2006	N
-	29	N	2003	N







MACHINE LEARNING TASKS

- **Supervised learning**:
  Given examples with inputs/outputs (*X*, *y*), learn a function to map *X* to *y*, i.e. *f(X)=y*
  Goal is to **learn** *f* from training data, and evaluate it on new (test) data.
  - Classification: *y* is discrete (y are **class labels**, X refers to **attributes**)
  - Regression: *y* is continuous
- **Unsupervised learning**:
  Given examples with only inputs *X*, learn a function *f(X)* to **simplify** the data and map to unknown *y*
  - Clustering: y is discrete (y are **cluster memberships**)
  - Matrix factorization: y are continuous (y are **embeddings**)

DESCRIPTIVE VS. PREDICTIVE MODELING

- Descriptive models **summarize** the data
  - Provide insights into the domain
  - Focus on modeling joint distribution P(X)
  - May be used for classification, but prediction is not the primary goal

- Predictive models **predict** the value of one variable of interest given known values of other variables
  - Focus on modeling the conditional distribution P(Y | X) or on modeling the decision boundary for Y

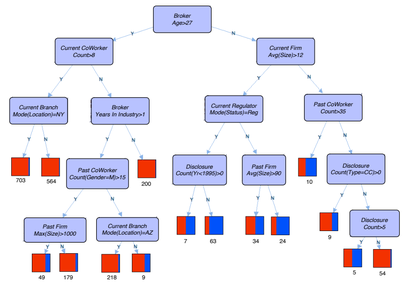
KNOWLEDGE REPRESENTATION (AKA MODEL FAMILY)

- **Underlying structure of the model or patterns that we seek from the data**
  - Specifies the models/patterns that could be returned as the results of the machine learning algorithm
  - Defines space of possible models/patterns for algorithm to search over

- Examples:
  - **If-then rule**
    If short closed car
    then toxic chemicals
  - **Regression model**

$$y = \beta_1 x_1 + \beta_2 x_2 \ldots + \beta_0$$

  - **Decision tree**



LEARNING TECHNIQUE

- **Method to construct model or patterns from data**
- **Model space**
  - Choice of knowledge representation defines a set of possible models or patterns
- **Objective function**
  - Associates a numerical value (score) with each member of the set of models/patterns
- **Search technique**
  - Defines a method for generating members of the set of models/patterns, determining their score, and identifying the ones with the “best” score

OBJECTIVE FUNCTION

- **A numeric score assigned to each possible model in a search space, given a reference/input dataset**
  - Used to judge the quality of a particular model for the domain

- Score function are **statistics**—estimates of a population parameter based on a sample of data
- Examples:
  - Misclassification
  - Squared error
  - Likelihood

EXAMPLE LEARNING PROBLEM

TASK: DEVISE A RULE TO CLASSIFY ITEMS BASED ON THE ATTRIBUTE X

Knowledge representation:

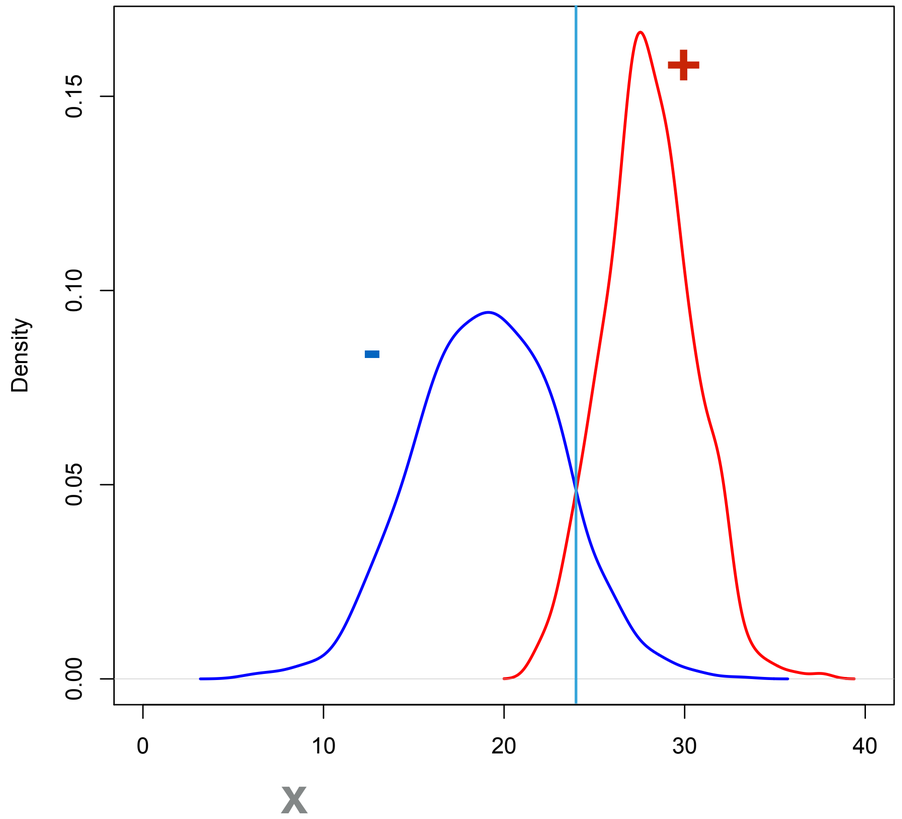
If-then rules
Example rule:

If x > 25 then **+**

Else **-**

What is the model space?

All possible thresholds



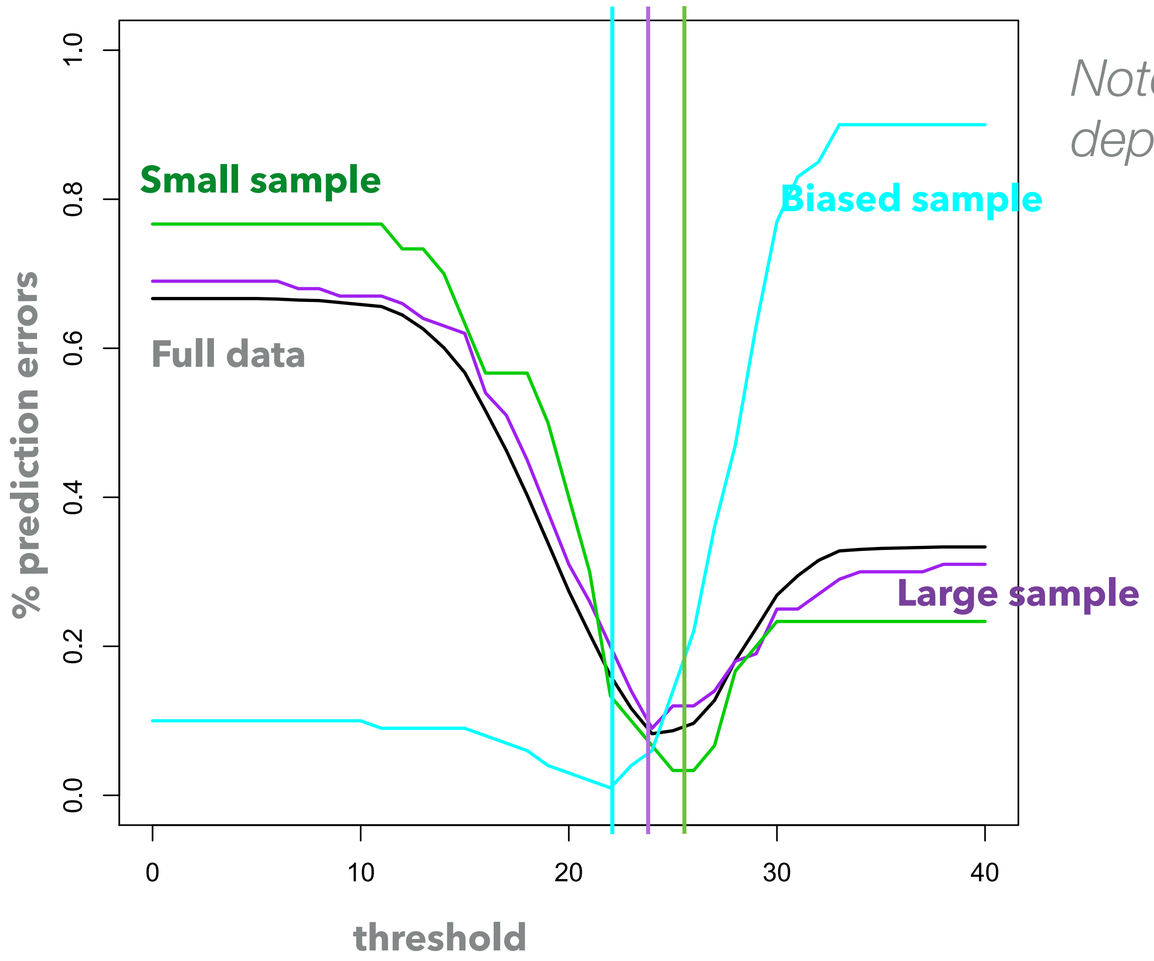
What is the objective function?

Prediction error rate

OBJECTIVE FUNCTION OVER MODEL SPACE

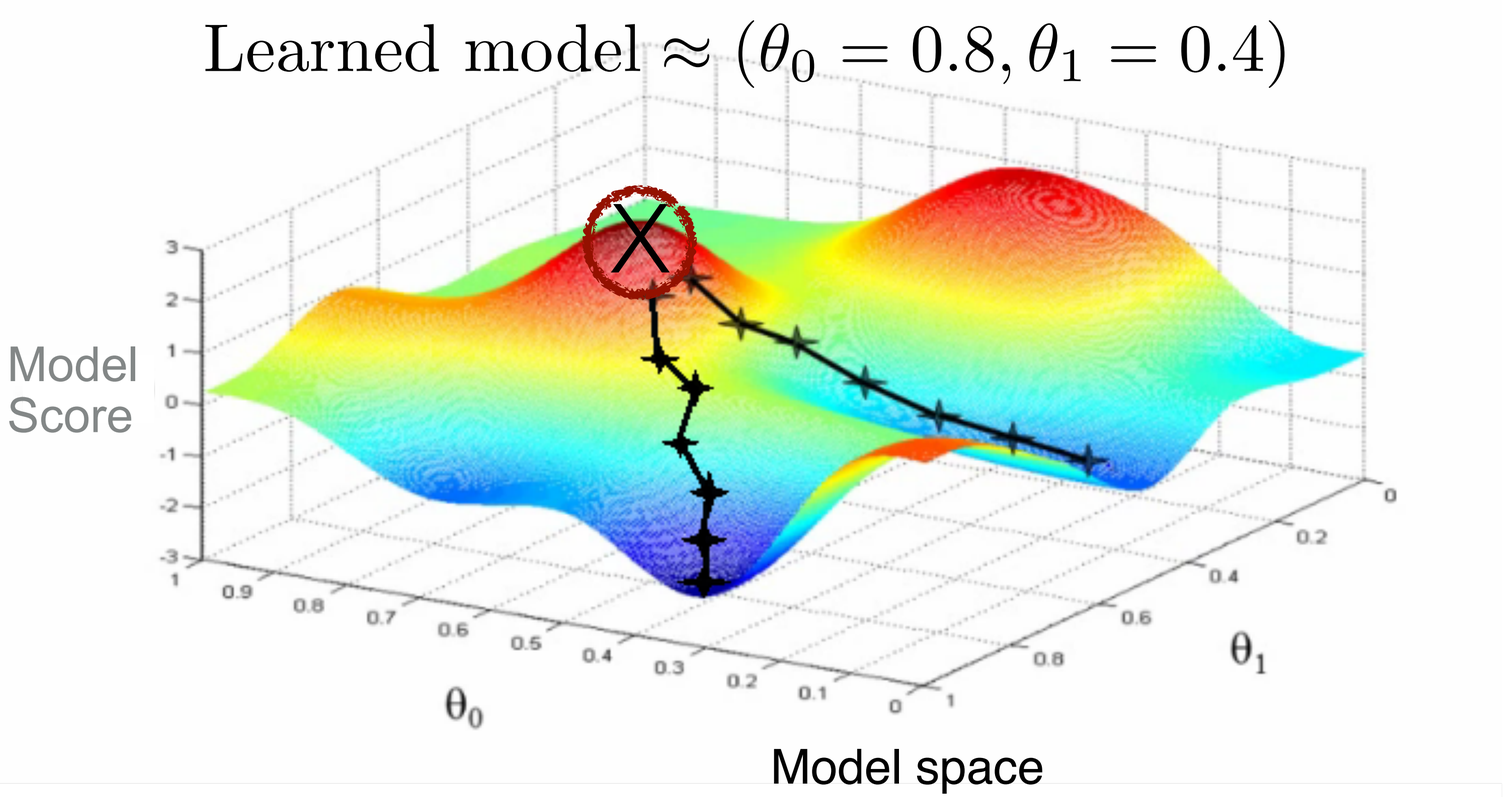
Search procedure?

Try all thresholds, select one with lowest score



*Note: learning result depends on **data***

WHAT SPACE ARE WE SEARCHING?



Alex Holehuse, Notes from Andrew Ng's Machine Learning Class, http://www.holehouse.org/mlclass/01_02_Introduction_regression_analysis_and_gr.html

SEARCHING OVER MODELS/PATTERNS

- Consider a **space** of possible models $M=\{M_1, M_2, ..., M_k\}$ with parameters θ
- Search could be over model structures or parameters, e.g.:
  - **Parameters**: In a linear regression model, find the regression coefficients ($\boldsymbol{\beta}$) that minimize squared loss on the training data
  - **Model structure**: In a decision trees, find the tree structure that maximizes accuracy on the training data

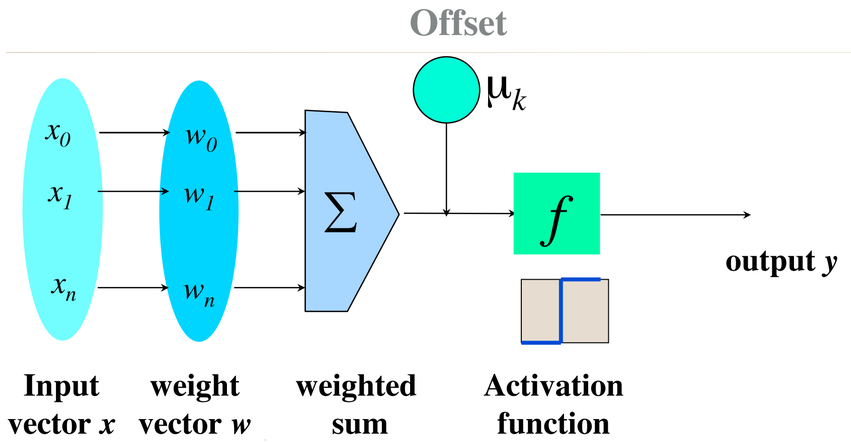
OPTIMIZATION OVER SCORE FUNCTIONS

- **Smooth** functions:
  - If a function is *smooth*, it is differentiable and the derivatives are continuous, then we can use gradient-based optimization
    - If function is *convex*, we can solve the minimization problem in closed form: $\nabla S(\theta)$ using **convex optimization**
    - If function is smooth but non-linear, we can use iterative search over the surface of S to find a local minimum (e.g., hill-climbing)
- **Non-smooth** functions:
  - If the function is *discrete*, then traditional optimization methods that rely on smoothness are not applicable. Instead we need to use **combinatorial optimization**

EXAMPLE: NEURAL NETWORKS

NEURON

- First learning algorithm in 1959 (Rosenblatt)
  - Perceptron learning rule
  - Provide target outputs with inputs for a single neuron
  - Incrementally update weights to learn to produce outputs



PERCEPTRON

Model: $f(x) = \sum_{i=1}^{m} w_i x_i + b$ *Offset*

$y = sign[f(x)]$

Activation function

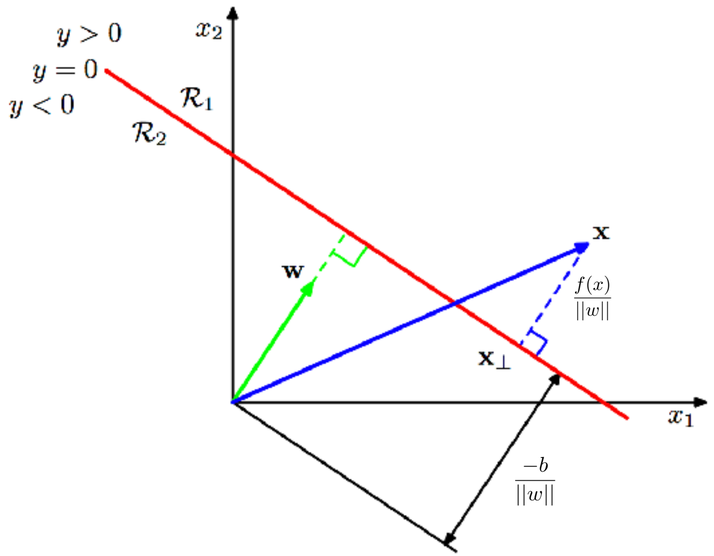


Figure: C. Bishop

Dot product is product of:

(i) projection of x onto w (i.e., ||x|| cos θ), and (ii) the length of w

Dot product is 0 if x is perpendicular to w

Add b, if >0 then positive class, if <0 then negative class

PERCEPTRON

Model: $f(x) = \sum_{i=1}^{m} w_i x_i + b$

$$y = sign[f(x)]$$

Learning: $if \;\; y(j)(\sum_{i=1}^{m} w_i x_i(j) + b) \leq 0$

$$then \;\; w \leftarrow w + \eta y(j) x(j)$$

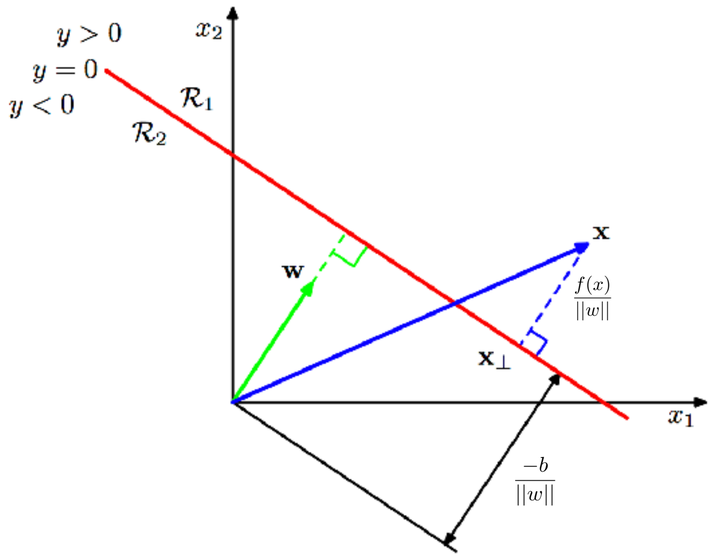
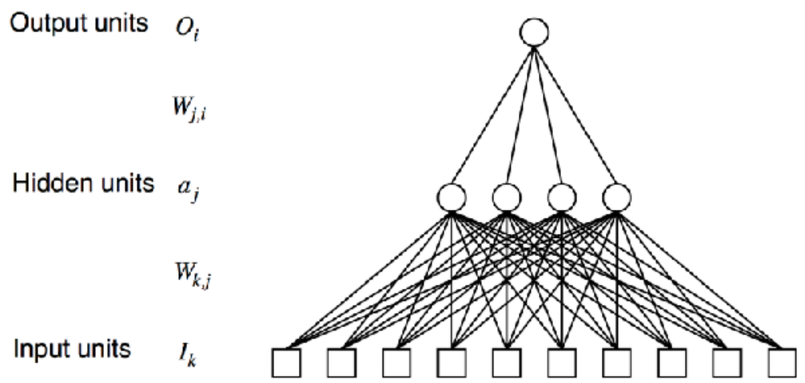


Figure: C. Bishop

ITERATE OVER TRAINING EXAMPLES FOR FIXED NUMBER OF ITERATIONS OR UNTIL ERROR IS BELOW A PRE-SPECIFIED THRESHOLD

TWO LAYERED NEURAL NETWORK

- Combine multiple perceptrons into ensemble
- Each perceptron output is a hidden unit, which are then aggregated into a final output
- Objective function: squared error, cross entropy



Output $O_i = g(\sum_j W_{j,i} a_j)$

Hidden units $a_j = g(\sum_k W_{k,j} I_k)$

Figure: M. Velosa

LEARNING NEURAL NETWORKS

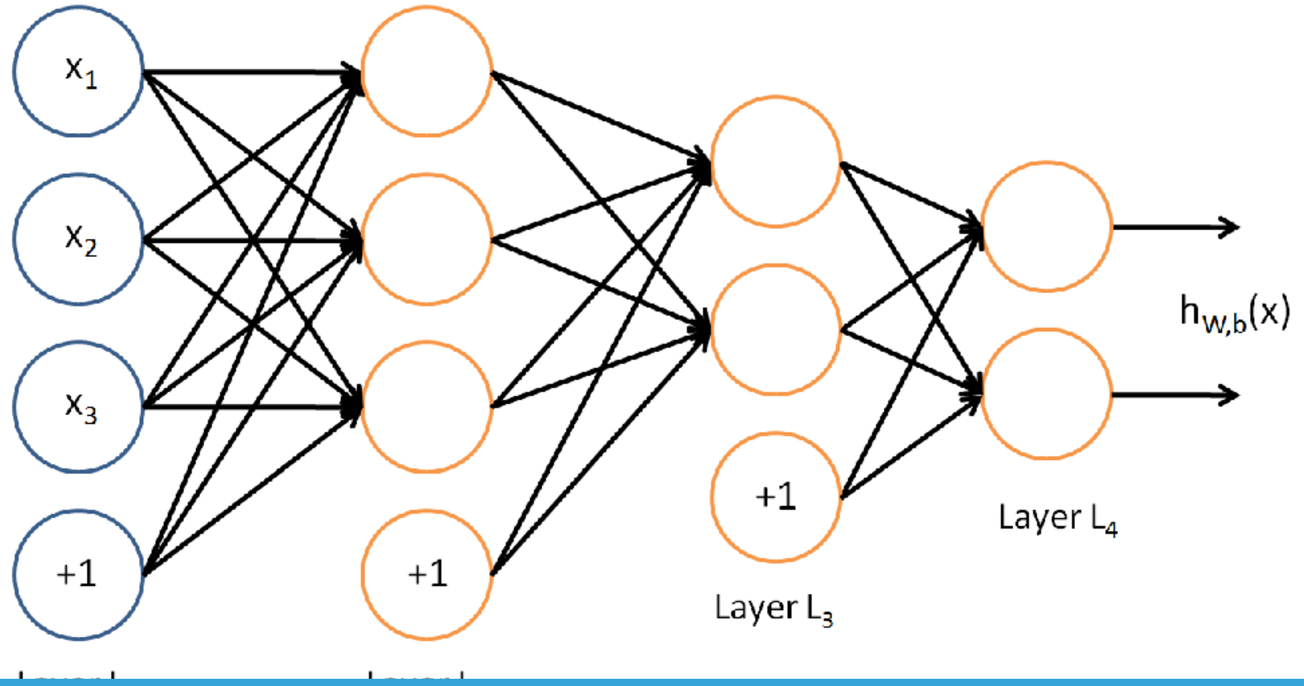
- Backpropagate error to each of the units in the network
- Assume activation function (g) is differentiable,
  then take partial derivative of the error with respect to each weight
  (using the chain rule)
- Update weights in similar way as for perceptron, e.g.,

$$\text{If } \Delta_i = Err_i g'(in_i) \text{ then } W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times I_k \times \Delta_j$$

FROM NEURAL NETWORKS TO DEEP LEARNING



ADDING LAYERS IN NEURAL NETWORKS GIVES THE MODEL MORE FLEXIBILITY —TRIED IN 1980S BUT DID NOT IMPROVE PERFORMANCE SUBSTANTIALLY BECAUSE BACK PROP ESTIMATION WOULD GET STUCK IN (SUBPAR) LOCAL MAXIMA

EXAMPLE: NAIVE BAYES CLASSIFIER

NAIVE BAYES CLASSIFIER

$$P(C|\mathbf{X}) \propto P(\mathbf{X}|C)P(C)$$

BAYES RULE

$$\propto \prod_{i=1}^{m} P(X_i|C)P(C)$$

NAIVE ASSUMPTION

Assumption: Attributes are *conditionally independent* given the class

MAXIMUM LIKELIHOOD ESTIMATION

- Widely used method of parameter estimation
- "Learn" the best parameters by finding the values of θ that maximizes likelihood:

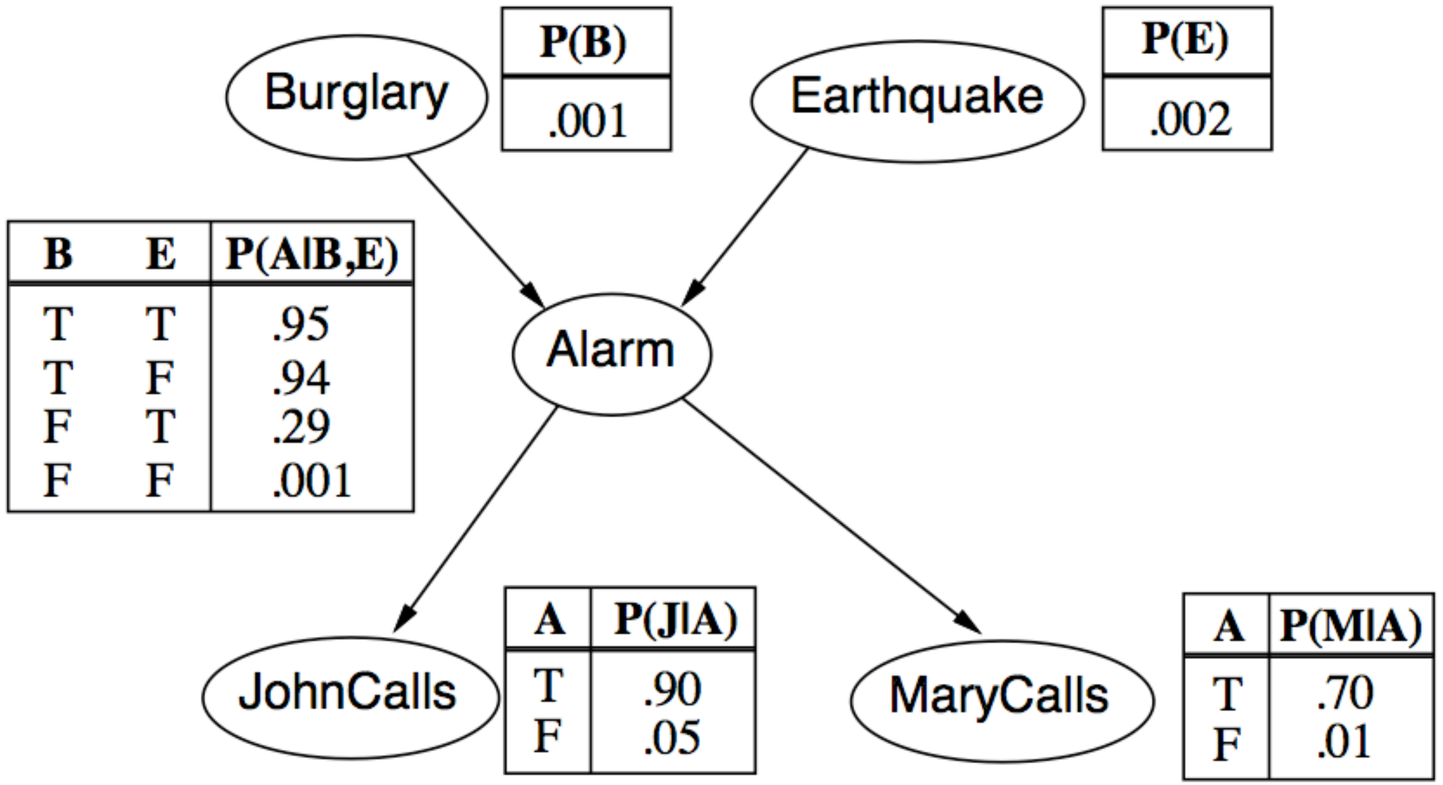
$$\hat{\theta}_{MLE} = \arg\max_{\theta} L(\theta)$$

- Often easier to work with loglikelihood:

$$\begin{aligned} l(\theta|D) &= log\ L(\theta|D) \\ &= log \prod_{i=1}^{n} p(x(i)|\theta) \\ &= \sum_{i=1}^{n} logp(x(i)|\theta) \end{aligned}$$

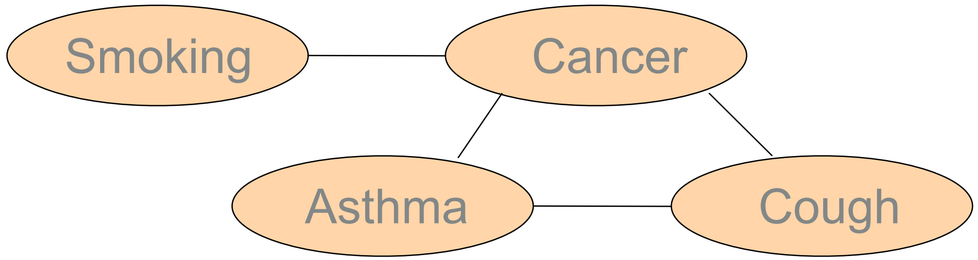
GRAPHICAL MODELS

BAYESIAN NETWORKS



$$P(\mathbf{X} = \mathbf{x}) = \prod_{i=1}^{p} P(X_i | parents(X_i))$$

MARKOV NETWORKS



Log-linear model:

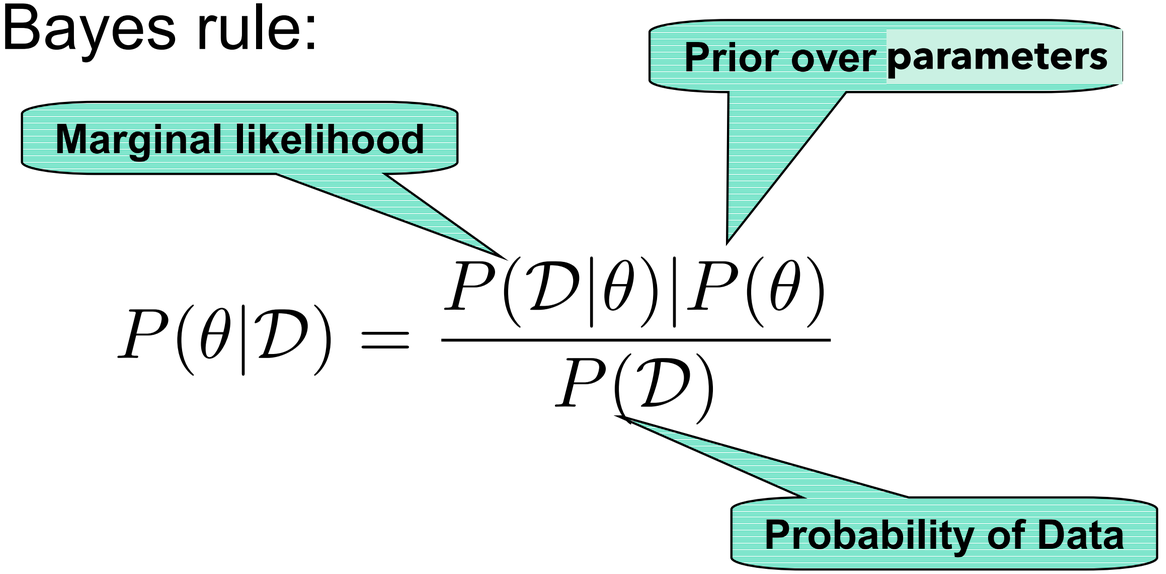
$$P(X = x) = \frac{1}{Z} exp(\sum_i w_i f_i(x))$$

Weight of Feature i Feature i

$$f_1(Smoking, Cancer) = \begin{cases} 1 & if \neg Smoking \vee Cancer \\ 0 & otherwise \end{cases}$$

OBJECTIVE FUNCTION

Using Bayes rule:



INFERENCE

- Bayesian networks (and Markov networks) are compact representations of probability distributions
- Each network describes a unique probability distribution *P*
- How do we answer queries about *P*?
- We use inference to refer to the process of computing answers to such queries

QUERIES

- There are many types of queries we might ask
- Most of these involve evidence
  - An evidence e is an assignment of values to a set of E variables in the domain
  - Simplest query: compute the probability of observing the evidence

$$P(e) = \sum_{x_1} ... \sum_{x_k} P(x_1, ..., x_k, e)$$

QUERIES

- We also may be interested in the conditional probability of a variable given the evidence

$$P(X|e) = \frac{P(X, e)}{P(e)}$$

- It is used for:
  - Prediction: what is the probability of an outcome given the starting condition? (target is descendant)
  - Diagnosis: what is the probability of disease given the symptoms? (target is ancestor)
- Direction between the variables does not restrict the directions of the queries

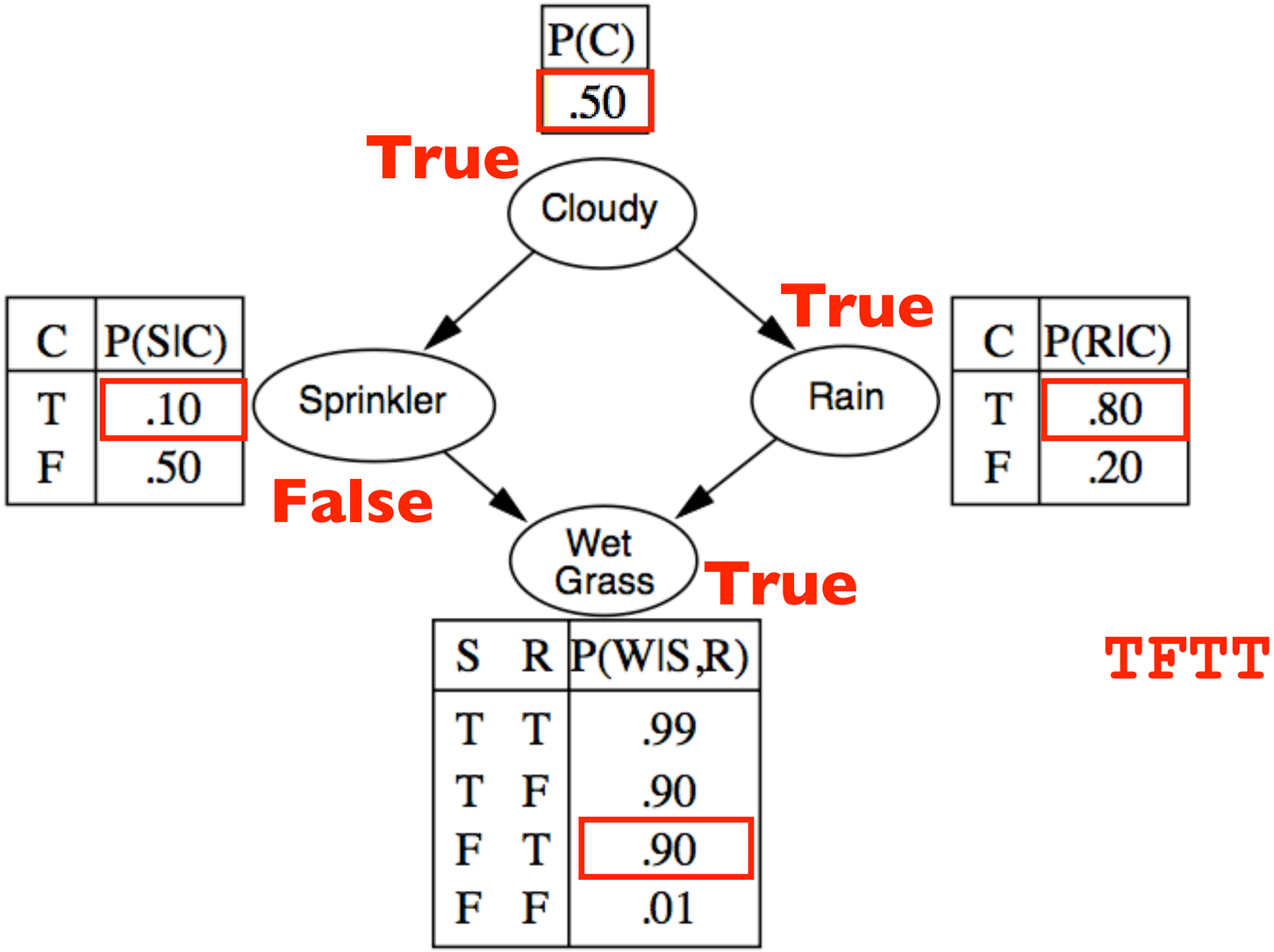
INFERENCE ALGORITHMS

- Exact inference methods
  - Variable elimination
  - Belief propagation (aka sum-product algorithm, message passing)
  - Clique tree propagation
  - Junction tree algorithm
- Approximate inference methods
  - Stochastic sampling and Markov Chain Monte Carlo (MCMC)
  - Variational methods
  - Loopy belief propagation

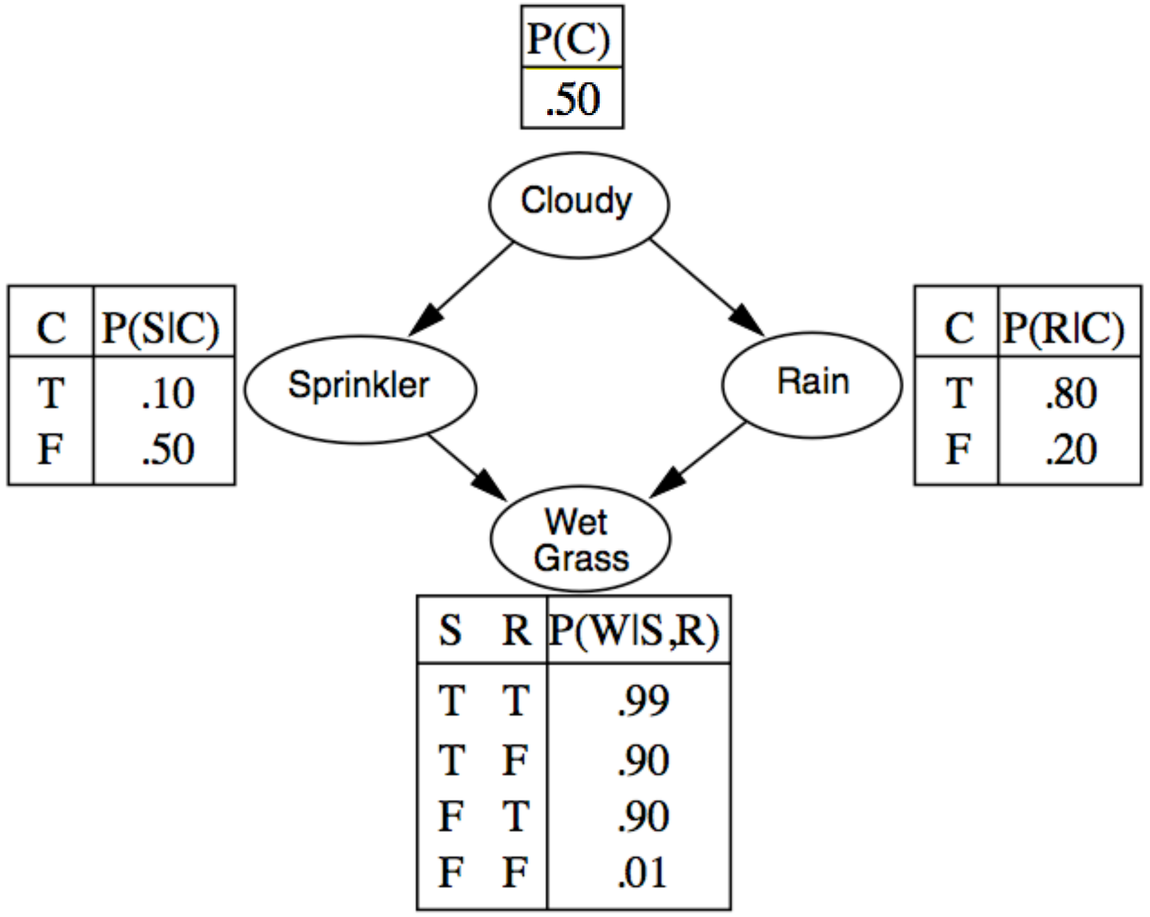
INFERENCE BY STOCHASTIC SIMULATION

- Core idea
  - Draw samples from a sampling distribution defined by the network
  - Compute an approximate posterior probability in a way that converges to the true probability
- Methods
  - Simple sampling from an empty network
  - Rejection sampling – reject samples that don't agree with the evidence
  - Likelihood weighting – weight samples based on evidence
  - Markov chain Monte Carlo – sample from a stochastic process whose stationary distribution is the true posterior

EXAMPLE


P(C)
.50
True
Cloudy
True
Sprinkler
Rain
False
Wet Grass
True
C P(S|C)
T .10
F .50
C P(R|C)
T .80
F .20
S R P(W|S,R)
T T .99
T F .90
F T .90
F F .01
TFTT

GENERATE EMPIRICAL SAMPLING DISTRIBUTION



TFTT

TTTT

FFTF

FFFF

TFTF

FTTT

FFTT

TTFT

TTFF

TFTT

FTFF

...

WHAT IF WE HAVE EVIDENCE OBSERVED IN SOME NODES?

REJECTION SAMPLING

- Sample the network as before...
- ...but discard samples that don't correspond with the evidence.
- Similar to real-world estimation procedures, which use observation
- However, it is hopelessly expensive for large networks where P(e) is small

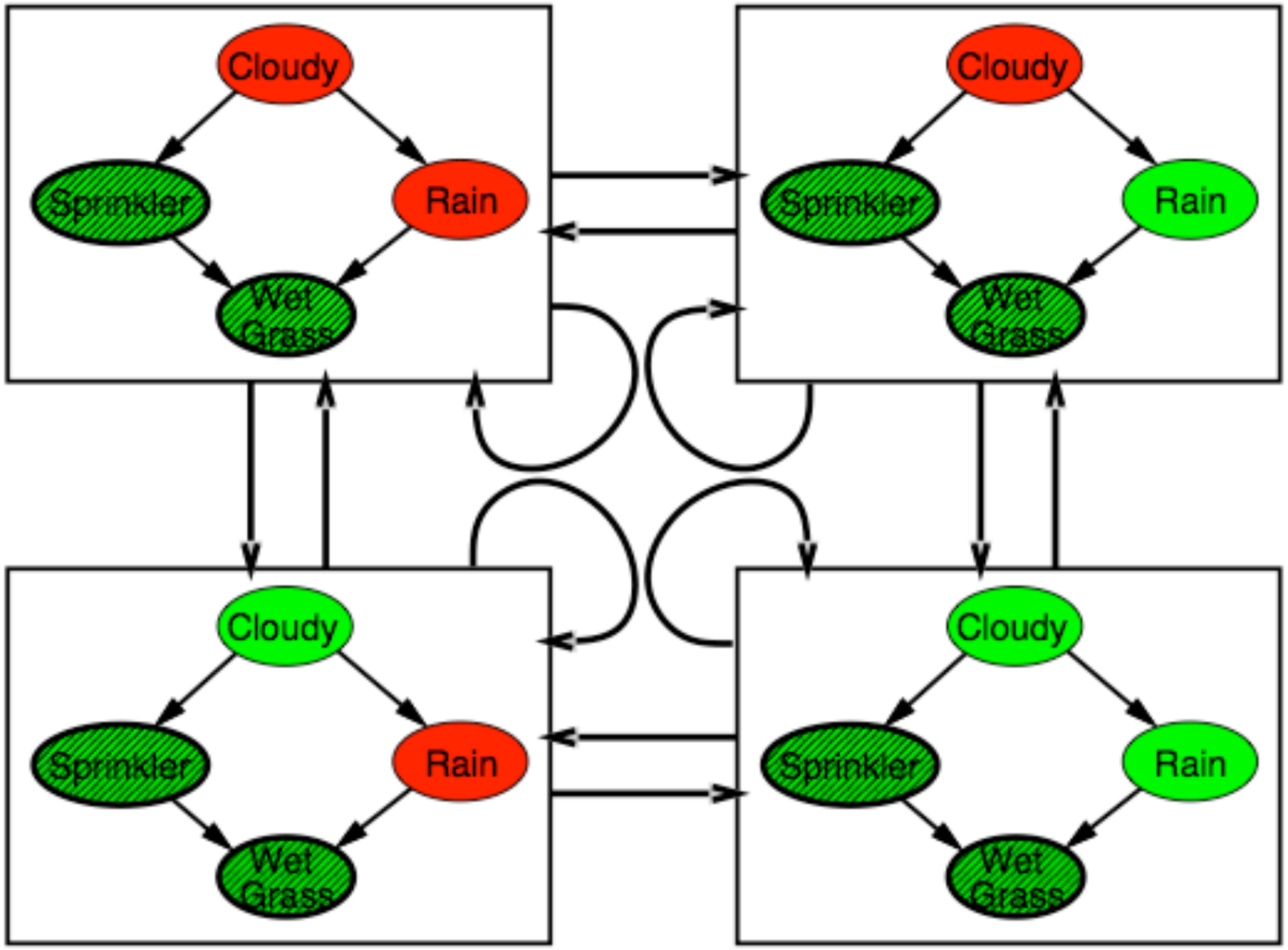
LIKELIHOOD WEIGHTING

- Do simple sampling as before...
  - But only generate samples that are consistent with the evidence
  - And weight the likelihood of each sample based on the evidence
- More efficient than rejection-based sampling since we use all the samples generated
- ... but performance degrades as number of evidence variables increases

MCMC

- The “state” of the system is the current assignment of all variables
- Algorithm
  - Initialize all variables randomly
  - Generate next state by sampling one variable given its Markov blanket
  - Sample each variable in turn, keeping evidence fixed
  - After “burn-in” the samples will be drawn from the posterior, use set of samples to determine probabilities of interest

MARKOV CHAIN



VARIATIONAL INFERENCE

- In variational inference, the posterior distribution over a set of query variables X given some evidence E is approximated by a variational distribution:

$$P(X|E) \approx Q(X)$$

- The variational distribution Q(X) is restricted to belong to a family of distributions of simpler form than P(X | E)
- The difference between Q and the true posterior is measured in terms of a dissimilarity function d(Q;P) and hence inference is performed by selecting the distribution Q that minimizes d.
- Example: for a multiply-connected network consider polytrees over same variables

CONNECTION TO PROBABILISTIC PROGRAMMING

HOW TO WRITE A BAYESIAN MODELING PAPER

1. Write down a generative model in an afternoon

2. Get 2 grad students to implement inference for a month

3. Use technical details of inference to pad half of the paper

Slide: David Duvenaud and James Llyod, University of Cambridge

CAN WE DO BETTER?

Example: Graphical Models

Application Papers

1. Write down a graphical model
2. Perform inference using general-purpose software
3. Apply to some new problem

Inference papers

1. Identify common structures in graphical models (e.g. chains)
2. Develop efficient inference method
3. Implement in a general-purpose software package

Modeling and inference have been disentangled

Slide: David Duvenaud and James Llyod, University of Cambridge

EX: INDIAN BUFFET HAWKES PROCESSES (TAN, RAO, N UAI'18)

Initialize the F uniform particle weights.

for *each observation* $\mathbf{y}_i = \{t_i, \mathcal{T}_i\}$, $i = 1, \ldots, n$ **do**

  for *each particle* $\mathbf{z}_i^f = \{\mathbf{K}_i, \mathbf{V}_i\}$ *of observation* $\mathbf{y}_i$, $f \in \{1, \ldots, F\}$ **do**

    - Sample the auxiliary variables $\mathbf{w}_i, \mathbf{c}_i$ and latent factor particles $\mathbf{z}_i^f = \{\mathbf{K}_i, \mathbf{V}_i\}$.
    - Sample the model parameters $\mathbf{\Theta} = \{\lambda_0, \{\beta_l\}, \{\tau_l\}\}$.
    - Update the triggering kernels.
    - Update the particle weights $\mathbf{u}_i^f$.

  end

  Normalize the particle weights.

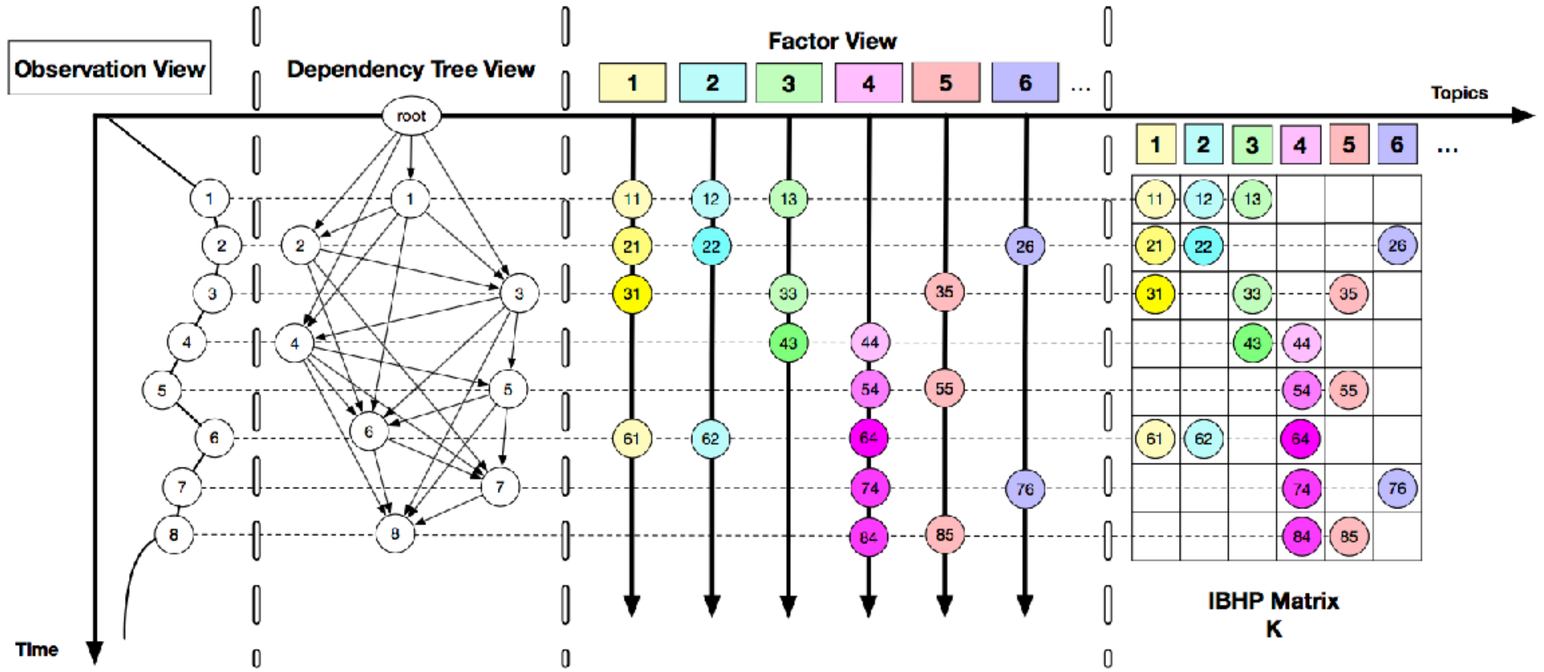
  if $\|\mathbf{u}_i\|_2^{-2} <$ *threshold, i.e., the effective number of particles is too low* **then**

    Resample particles with replacement based on the particle weights.

  end

end

Algorithm 2: SMC inference algorithm for IBHP.



Inference: sequential Monte Carlo using particle filtering

EX: CONSTRAINED SAMPLING OF ATTRIBUTED GRAPHS (ROBLES, MORENO, N KDD'16)

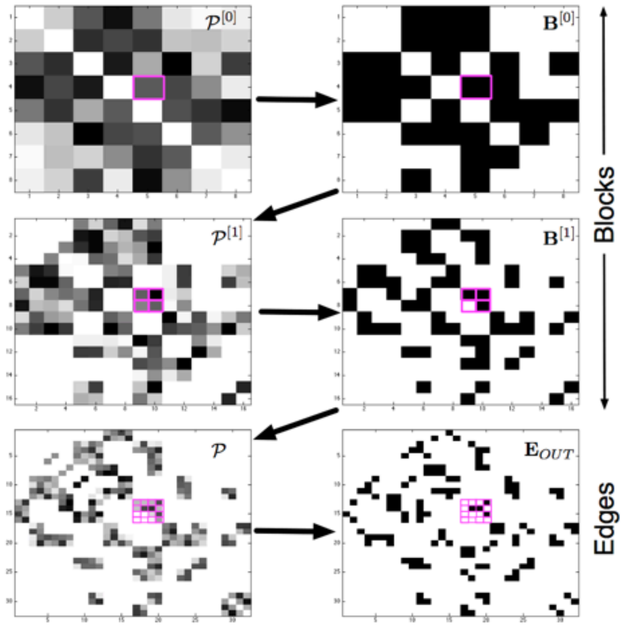


Figure 1: mKPGM sampling process with $b = 2$, $\ell = 3$, $K = 5$. $\mathcal{P}^{[0]}$ is generated as a KPGM. Left: Matrices of probabilities $\mathcal{P}^{[l]}$ (white: $\mathcal{P}_{ij} = 0$ black: $\mathcal{P}_{ij} = 1$). Right: Adjacency matrices $\mathbf{B}^{[0]} \dots \mathbf{B}^{[K-\ell-1]}, E_{OUT}$ (black $\Rightarrow$ block/edge sampled).

Algorithm *GraphSampling*

Input: $G_{IN} = (\mathbf{V}, \mathbf{E})$, node attrs $\mathbf{X}_{IN}$; GNM $\mathcal{M}$, error ϵ
Ouput: $G_{OUT}, \mathbf{X}_{OUT}, \rho_{OUT}$
$[\Psi, \beta, \Theta^{\mathbf{X}}, \Theta^{G}] \leftarrow$ *LearnParameters*$(G_{IN}, \mathbf{X}_{IN})$
Sample $\mathbf{X}_{OUT}$ from $P(\mathbf{X}|\Theta^{\mathbf{X}})$
Initialize $\rho_{OUT} = \infty$ and $l_o = K - \ell - 1$
while $(|\rho_{IN} - \rho_{OUT}| > \epsilon)$ AND $(l_o \geq 0)$ **do**
  Sample $\mathbf{B}^{[l_o]}_{sample} \sim \mathcal{M}(\Theta^{G})$ using *basic sampling* [2]
  for $l = l_o + 1$ **to** $K - \ell - 1$ **do**
    $\mathbf{B}^{[l+1]}_{sample} \leftarrow$ *LPBlockSearch*$(\mathcal{M}, \Theta^{G}, \mathbf{B}^{[l]}_{sample}, \Psi, \beta)$
  $G_{OUT} \leftarrow$ *MEEdgeSampling*$(\mathcal{M}, \Theta^{G}, \mathbf{B}^{[K-\ell-1]}_{sample}, \Psi, \beta)$
  Calculate ρ_{OUT} using G_{OUT} and $\mathbf{X}_{OUT}$
  $l_o = l_o - 1$

Algorithm *LPBlockSearch*

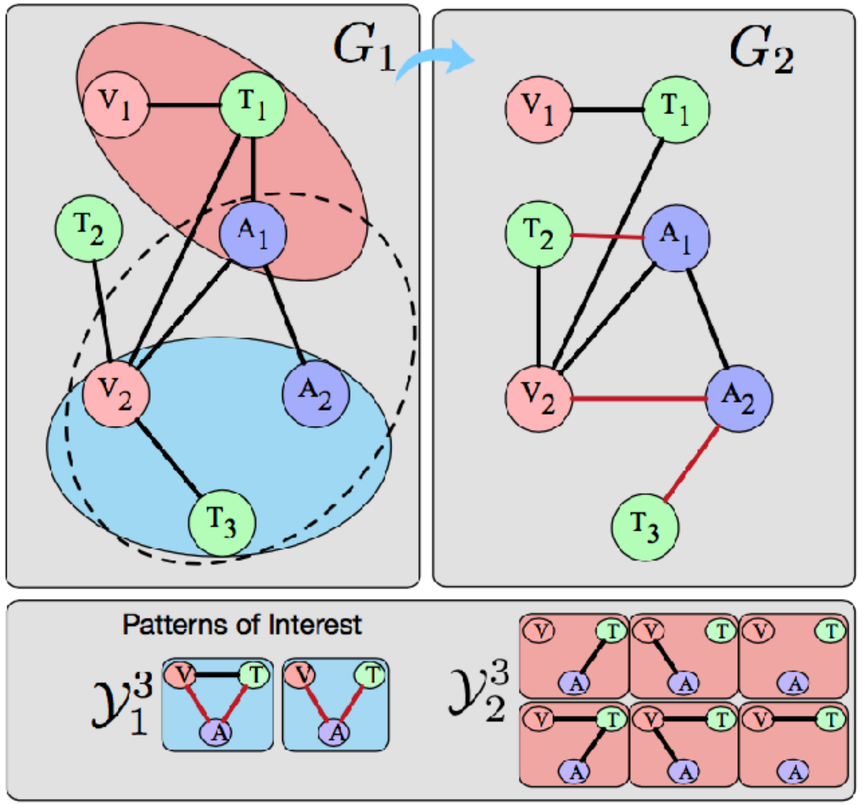
Input: $\mathcal{M}, \Theta^{G}, \mathbf{B}^{[l]}_{sample}, \Psi, \beta$
Ouput: $\mathbf{B}^{[l+1]}_{sample}$ sampled blocks in $l + 1$:
$(\mathbf{U}, \mathbf{T}) =$ *getUniquePr_BLocations*$(\mathcal{M}, \Theta^{G}, \mathbf{B}^{[l]}_{sample})$
for $u = 1$ **to** $|\mathbf{U}|$ **do**
  Draw $n_u \sim Bin(|\mathbf{T}_u|, \pi_u)$ {# of blocks to sample per π_u}
  for $j = 1$ **to** $|\Psi|$ **do**
    $\mathbf{e}_j = \beta_j \times n_u$ {fraction of possible edges leading to ρ_{IN}}
  Determine A_{jk} {# of descendent edges of type $\psi_j \in \Psi$ per position $t_k \in \mathbf{T}_u$}
  for $j = 1$ **to** N_Ω **do**
    $ub_j = \sum_{k=1}^{|\Psi|} A_{jk}$ {max # of sampled blocks per $A_{\cdot k}$}
  Solve the LP of eq 1: find min χ using n_u, $\mathbf{e}$, A, and ub
  for $j = 1$ **to** N_Ω **do**
    $\mathbb{B}'_{sample} =$ Randomly sample χ_j blocks from ub_j places
    $\mathbf{B}^{[l+1]}_{sample} = \mathbf{B}^{[l+1]}_{sample} \cup \mathbb{B}'_{sample}$

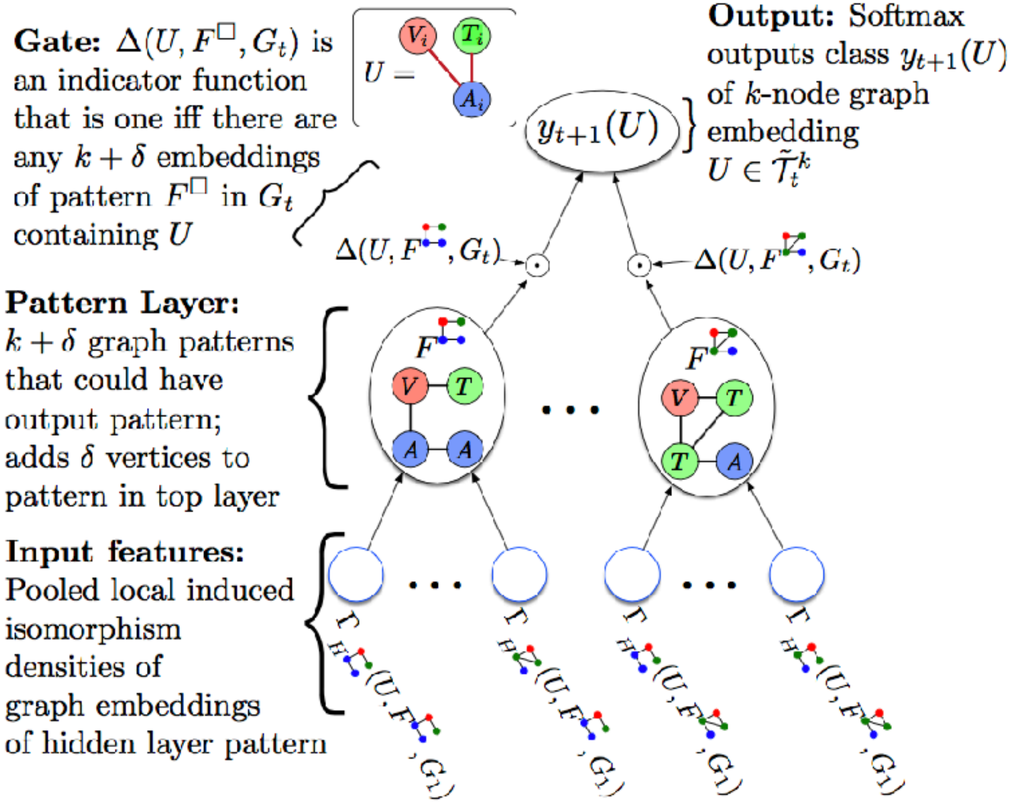
Algorithm *MEEdgeSampling*

Input: $\mathcal{M}, \Theta^{G}, \mathbf{B}^{[K-\ell-1]}_{sample}, \Psi, \beta$
Output: G_{OUT}
$(\mathbf{U}, \mathbf{T}) =$ *getUniquePr_ELocations*$(\mathcal{M}, \Theta^{G}, \mathbf{B}^{[l]}_{sample}, \Psi)$
for $u = 1$ **to** $|\mathbf{U}|$ **do**
  Draw $n_u \sim Bin(|\mathbf{T}_{u\cdot}|, \pi_u)$ {# edges per unique probabil}
  Set $N_e = N_e + n_u$ {Total # edges to be sample}
Draw $\mathbf{\Gamma} = [\gamma_1, \dots, \gamma_{|\Psi|}] \sim Mult(N_e; \beta)$ {# edges per edge-type to match ρ_{IN}}
for $u = 1$ **to** $|\mathbf{U}|$ **do**
  Draw $\mathbf{Y} = [Y_1, \dots, Y_{|\Psi|}] \sim Mult\left(n_u; \frac{\Gamma}{N_e}\right)$ {# edges per edge-type for π_u}
  for $j = 1$ **to** $|\Psi|$ **do**
    $\mathbf{E}' =$ Sampling Y_j edges at random from $\mathbf{T}_{uj}$ locations.
    $\mathbf{E}_{OUT} = \mathbf{E}_{OUT} \cup \mathbf{E}'$
    $\gamma_j = \gamma_j - Y_j$ {Adaptative process to match the moments}
    $N_e = N_e - Y_j$

Sampling: Two-stage constrained process, maximum entropy + linear programming

EX: SUBGRAPH PATTERN NEURAL NETWORKS (MENG, MOULI, RIBEIRO, N AAAI'18)





Implemented in Theano: challenge is dynamic construction of model architecture,
but also need to modify optimization to use distributions rather than independent samples

MACHINE LEARNING 101

1. Data representation
2. Knowledge representation
3. objective function
4. Search algorithm

SAMPLING WITH EVIDENCE

SAMPLING WITH CONSTRAINTS

SAMPLING COMPLEX DISTRIBUTIONS EFFICIENTLY (SPACE OR TIME)

APPROXIMATE VS EXACT INFERENCE (CORRECTNESS/EFFICIENCY TRADEOFF, FOR BOTH CONTINUOUS&DISCRETE)

OPTIMIZATION OVER SETS

SAMPLING FROM NON-GENERATIVE DISTRIBUTIONS