

## Chapter 11: Learning

- **Lecture 1** Learning Issues.
- **Lecture 2** Decision-tree learning.
- **Lecture 3** Neural network learning.
- **Lecture 4** Case-Based reasoning.
- **Lecture 5** Learning under uncertainty.



# Learning

Learning is the ability to improve one's behavior based on experience.

- The range of behaviors is expanded: the agent can do more.
- The accuracy on tasks is improved: the agent can do things better.
- The speed is improved: the agent can do things faster.

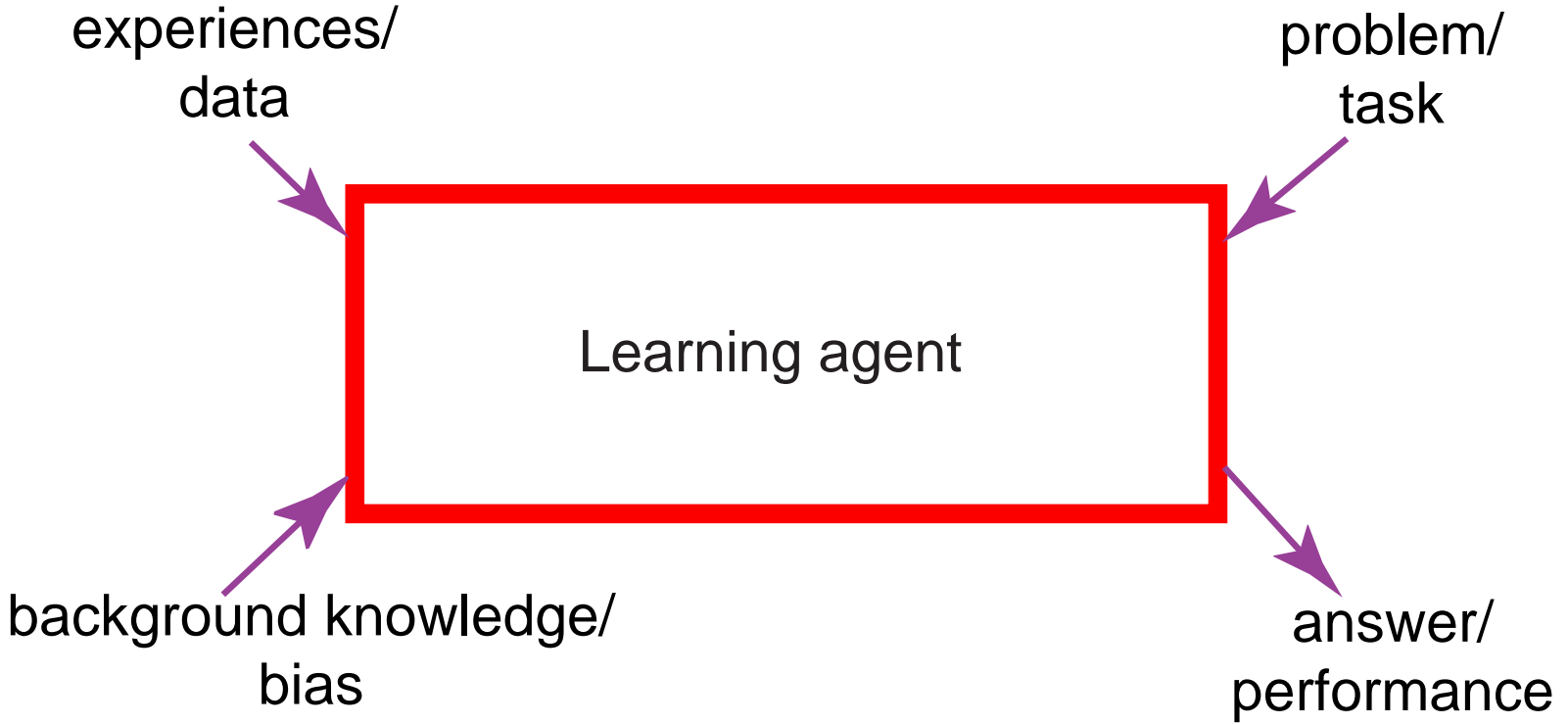


# Components of a learning problem

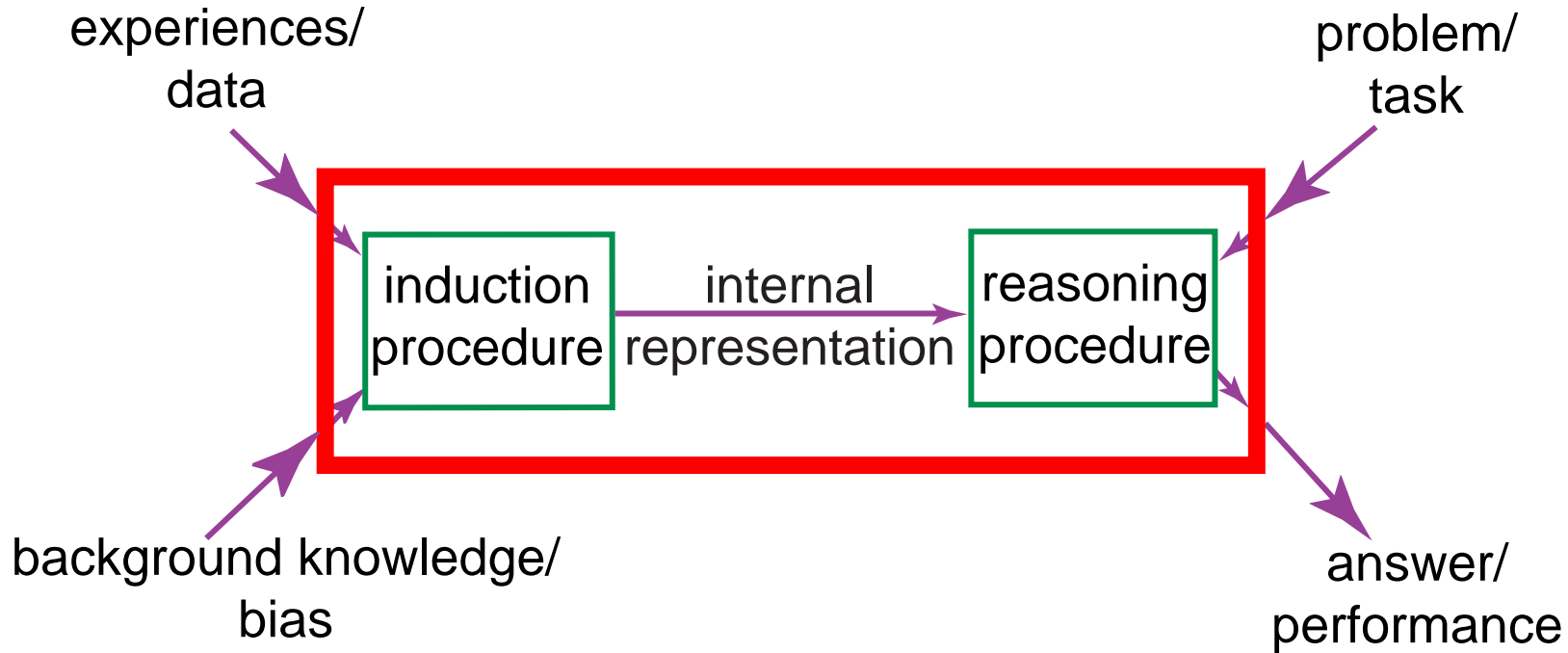
The following components are part of any learning problem:

- **task** The behavior or task that's being improved.  
For example: classification, acting in an environment
- **data** The experiences that are being used to improve performance in the task.
- **measure of improvement** How can the improvement be measured?  
For example: increasing accuracy in prediction, new skills that were not present initially, improved speed. 3

# Learning task



# Learning architecture



# Choosing a representation

- The richer the representation, the more useful it is for subsequent problem solving.
- The richer the representation, the more difficult it is to learn.



# Common Learning Tasks

- **Supervised classification** Given a set of pre-classified training examples, classify a new instance.
- **Unsupervised learning** Find natural classes for examples.
- **Reinforcement learning** Determine what to do based on rewards and punishments.
- **Analytic learning** Reason faster using experience.
- **Inductive logic programming** Build richer models in terms of logic programs.



# Example Classification Data

	Action	Author	Thread	Length	Where
e1	skips	known	new	long	home
e2	reads	unknown	new	short	work
e3	skips	unknown	old	long	work
e4	skips	known	old	long	home
e5	reads	known	new	short	home
e6	skips	known	old	long	work

We want to classify new examples on property *Action* based on the examples' *Author*, *Thread*, *Length*, and *Where*.





# Feedback

Learning tasks can be characterized by the feedback given to the learner.

- **Supervised learning** What has to be learned is specified for each example.
- **Unsupervised learning** No classifications are given; the learner has to discover categories and regularities in the data.
- **Reinforcement learning** Feedback occurs after a sequence of actions.



# Measuring Success

- The measure of success is not how well the agent performs on the training examples, but how well the agent performs for new examples.
- Consider two agents:
  - $P$  claims the negative examples seen are the only negative examples. Every other instance is positive.
  - $N$  claims the positive examples seen are the only positive examples. Every other instance is negative.
- Both agents correctly classify every training example, but disagree on every other example.



# Bias

- The tendency to prefer one hypothesis over another is called a **bias**.
- Saying a hypothesis is better than  $N$ 's or  $P$ 's hypothesis isn't something that's obtained from the data.
- To have any inductive process make predictions on unseen data, you need a bias.
- What constitutes a good bias is an empirical question about which biases work best in practice.



# Learning as search

- Given a representation and a bias, the problem of learning can be reduced to one of search.
- Learning is search through the space of possible representations looking for the representation or representations that best fits the data, given the bias.
- These search spaces are typically prohibitively large for systematic search. Use **hill climbing**.
- A learning algorithm is made of a search space, an evaluation function, and a search method.



# Noise

- Data isn't perfect:
  - some of the attributes are assigned the wrong value
  - the attributes given are inadequate to predict the classification
  - there are examples with missing attributes
- **overfitting** occurs when a distinction appears in the data, but doesn't appear in the unseen examples. This occurs because of random correlations in the training set.



# Characterizations of Learning

- Find the best representation given the data.
- Delineate the class of consistent representations given the data.
- Find a probability distribution of the representations given the data.



# Learning Decision Trees

- Representation is a decision tree.
- Bias is towards simple decision trees.
- Search through the space of decision trees, from simple decision trees to more complex ones.



# Decision trees

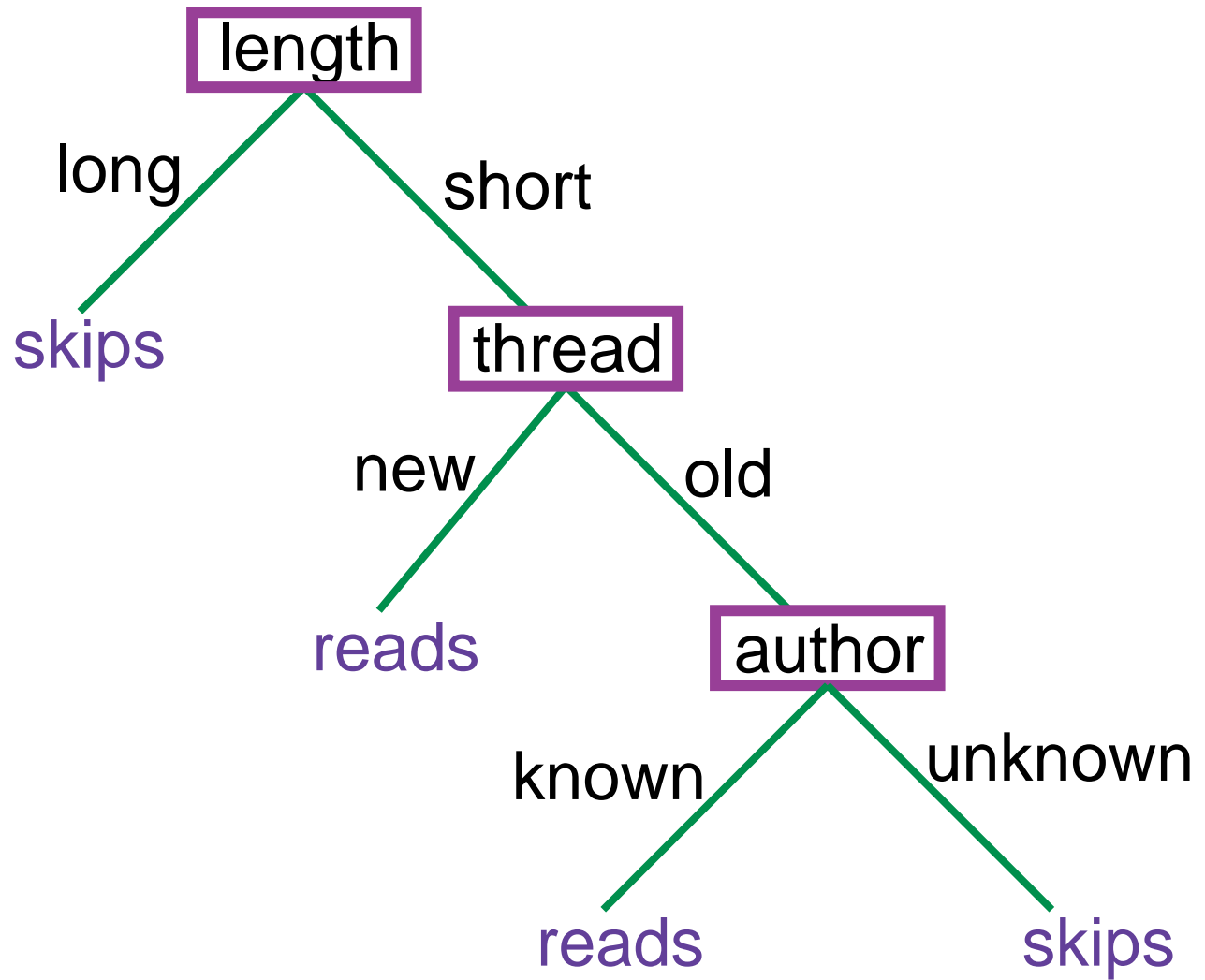
A **decision tree** is a tree where:

- The nonleaf nodes are labeled with attributes.
- The arcs out of a node labeled with attribute  $A$  are labeled with each of the possible values of the attribute  $A$ .
- The leaves of the tree are labeled with classifications.





# Example Decision Tree



# Equivalent Logic Program

$prop(Obj, user\_action, skips) \leftarrow$   
 $prop(Obj, length, long).$

$prop(Obj, user\_action, reads) \leftarrow$   
 $prop(Obj, length, short) \wedge prop(Obj, thread, new).$

$prop(Obj, user\_action, reads) \leftarrow$   
 $prop(Obj, length, short) \wedge prop(Obj, thread, old) \wedge$   
 $prop(Obj, author, known).$

$prop(Obj, user\_action, skips) \leftarrow$   
 $prop(Obj, length, short) \wedge prop(Obj, thread, old) \wedge$   
 $prop(Obj, author, unknown).$



# Issues in decision-tree learning

- Given some data, which decision tree should be generated? A decision tree can represent any discrete function of the inputs.
- You need a **bias**. Example, prefer the smallest tree. Least depth? Fewest nodes? Which trees are the best predictors of unseen data?
- How should you go about building a decision tree? The space of decision trees is too big for systematic search for the smallest decision tree.



# Searching for a Good Decision Tree

- The input is a target attribute (the *Goal*), a set of examples, and a set of attributes.
- Stop if all examples have the same classification.
- Otherwise, choose an attribute to split on,
  - for each value of this attribute, build a subtree for those examples with this attribute value.



# Decision tree learning: Boolean attributes

*% dtlearn(Goal, Examples, Attributes, DT)* given *Examples*

*%* and *Attributes* construct decision tree *DT* for *Goal*.

*dtlearn(Goal, Exs, Atts, Val) ←*

*all\_examples\_agree(Goal, Exs, Val).*

*dtlearn(Goal, Exs, Atts, if(Cond, YT, NT)) ←*

*examples\_disagree(Goal, Exs) ∧*

*select\_split(Goal, Exs, Atts, Cond, Rem\_Atts) ∧*

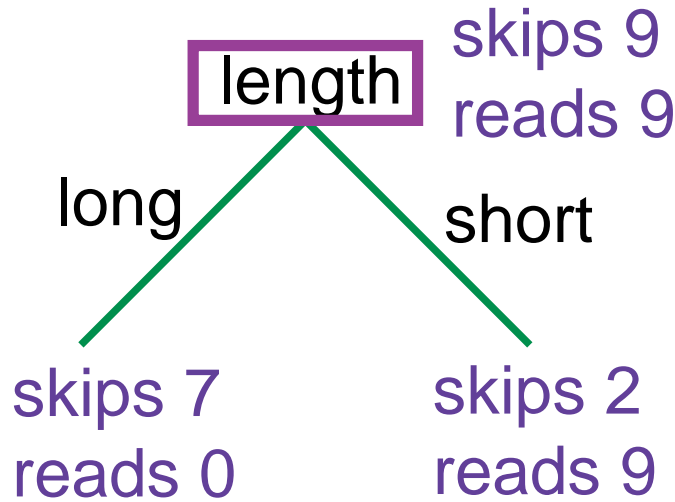
*split(Exs, Cond, Yes, No) ∧*

*dtlearn(Goal, Yes, Rem\_Atts, YT) ∧*

*dtlearn(Goal, No, Rem\_Atts, NT).*



# Example: possible splits



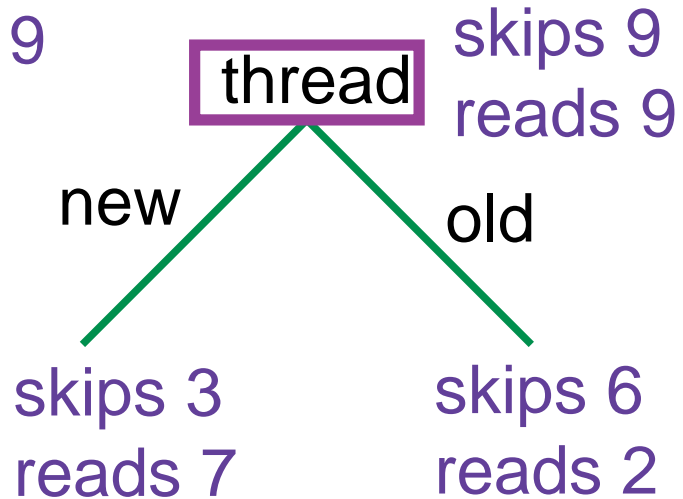
skips 9  
reads 9

long

short

skips 7  
reads 0

skips 2  
reads 9



skips 9  
reads 9

new

old

skips 3  
reads 7

skips 6  
reads 2



# Using this algorithm in practice

- Attributes can have more than two values. This complicates the trees.
- This assumes attributes are adequate to represent the concept. You can return probabilities at leaves.
- Which attribute to select to split on isn't defined. You want to choose the attribute that results in the smallest tree. Often we use information theory as an evaluation function in hill climbing.
- Overfitting is a problem.



# Handling Overfitting

- This algorithm gets into trouble overfitting the data. This occurs with noise and correlations in the training set that are not reflected in the data as a whole.
- To handle overfitting:
  - You can restrict the splitting, so that you split only when the split is useful.
  - You can allow unrestricted splitting and prune the resulting tree where it makes unwarranted distinctions.





# Neural Networks

- These representations are inspired by neurons and their connections in the brain.
- Artificial neurons, or **units**, have inputs, and an output. The output can be connected to the inputs of other units.
- The output of a unit is a parameterized non-linear function of its inputs.
- Learning occurs by adjusting parameters to fit data.
- Neural networks can represent an approximation to any function.



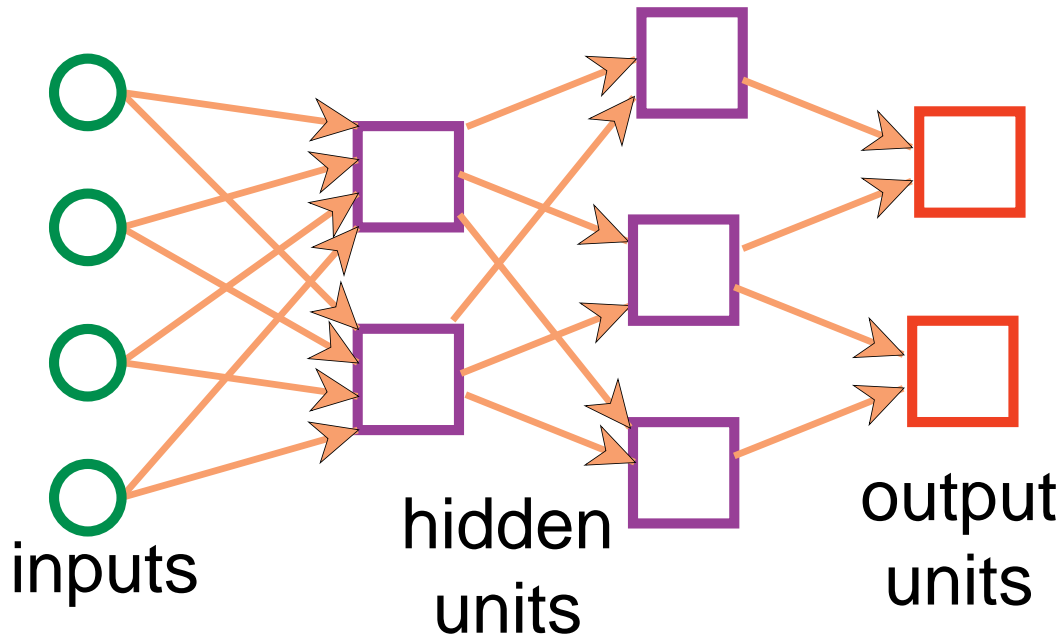
# Why Neural Networks?

- As part of neuroscience, in order to understand real neural systems, researchers are simulating the neural systems of simple animals such as worms.
- It seems reasonable to try to build the functionality of the brain via the mechanism of the brain (suitably abstracted).
- The brain inspires new ways to think about computation.
- Neural networks provide a different measure of simplicity as a learning bias.



# Feed-forward neural networks

- Feed-forward neural networks are the most common models.
- These are directed acyclic graphs:



# The Units

A unit with  $k$  inputs is like the parameterized logic program:

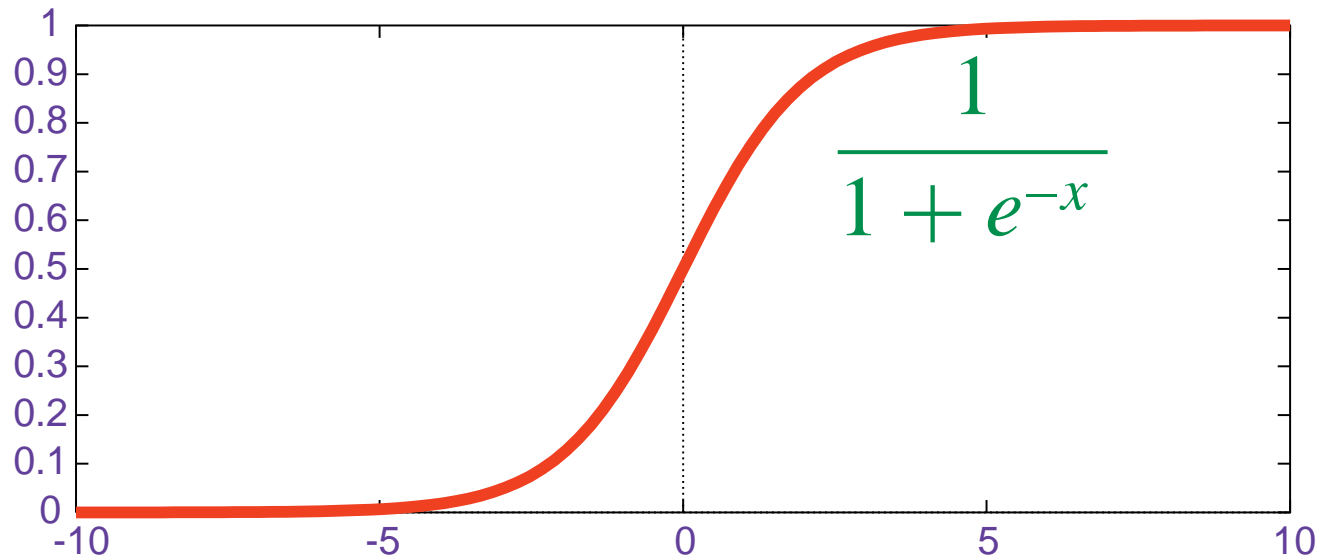
$$\text{prop}(\text{Obj}, \text{output}, V) \leftarrow$$
$$\text{prop}(\text{Obj}, \text{in}_1, I_1) \wedge$$
$$\text{prop}(\text{Obj}, \text{in}_2, I_2) \wedge$$
$$\dots$$
$$\text{prop}(\text{Obj}, \text{in}_k, I_k) \wedge$$
$$V \text{ is } f(w_0 + w_1 \times I_1 + w_2 \times I_2 + \dots + w_k \times I_k).$$

- $I_j$  are real-valued inputs.
- $w_j$  are adjustable real parameters.
- $f$  is an activation function.



# Activation function

A typical activation function is the **sigmoid** function:

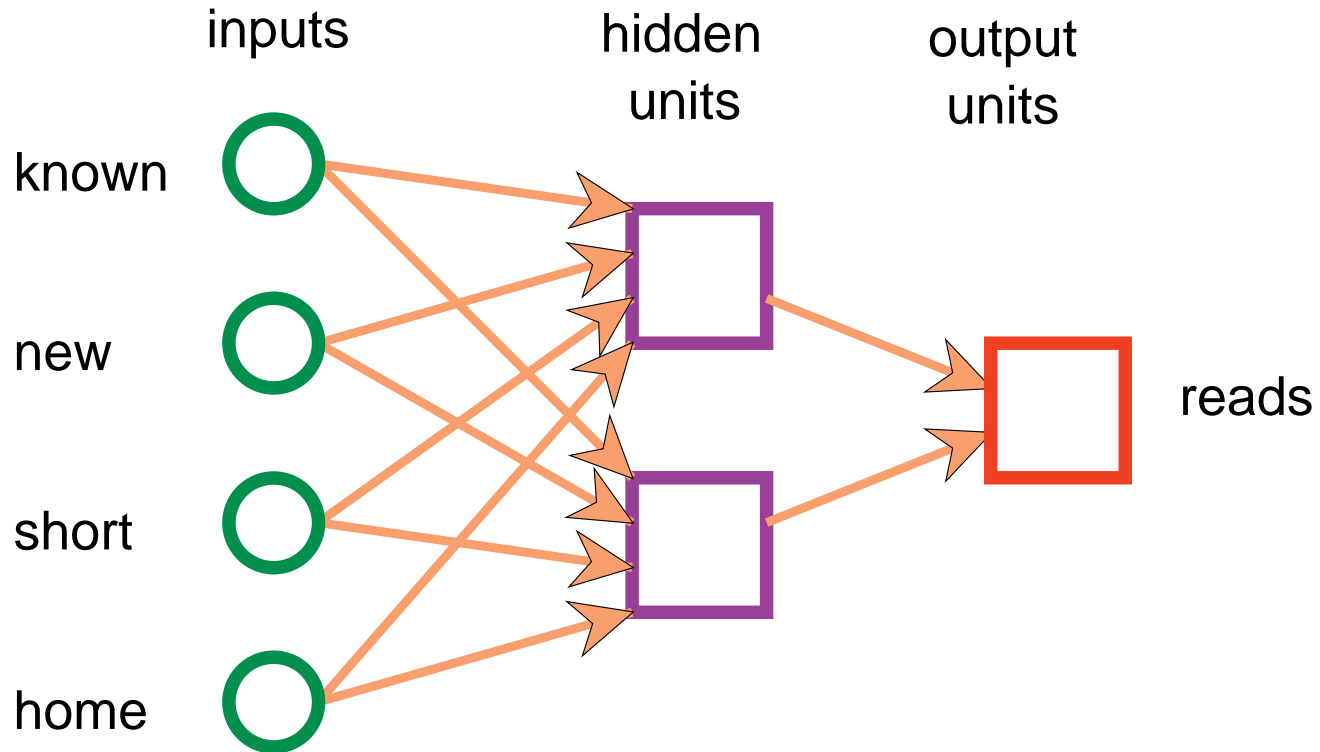


$$f(x) = \frac{1}{1 + e^{-x}}$$

$$f'(x) = f(x)(1 - f(x))$$



# Neural Network for the news example



# Axiomatizing the Network

- The values of the attributes are real numbers.
- Thirteen parameters  $w_0, \dots, w_{12}$  are real numbers.
- The attributes  $h_1$  and  $h_2$  correspond to the values of hidden units.
- There are 13 real numbers to be learned. The hypothesis space is thus a 13-dimensional real space.
- Each point in this 13-dimensional space corresponds to a particular logic program that predicts a value for *reads* given *known*, *new*, *short*, and *home*.



$\text{predicted\_prop}(\text{Obj}, \text{reads}, V) \leftarrow$

$\text{prop}(\text{Obj}, h_1, I_1) \wedge \text{prop}(\text{Obj}, h_2, I_2) \wedge$

$V \text{ is } f(w_0 + w_1 \times I_1 + w_2 \times I_2).$

$\text{prop}(\text{Obj}, h_1, V) \leftarrow$

$\text{prop}(\text{Obj}, \text{known}, I_1) \wedge \text{prop}(\text{Obj}, \text{new}, I_2) \wedge$

$\text{prop}(\text{Obj}, \text{short}, I_3) \wedge \text{prop}(\text{Obj}, \text{home}, I_4) \wedge$

$V \text{ is } f(w_3 + w_4 \times I_1 + w_5 \times I_2 + w_6 \times I_3 + w_7 \times I_4).$

$\text{prop}(\text{Obj}, h_2, V) \leftarrow$

$\text{prop}(\text{Obj}, \text{known}, I_1) \wedge \text{prop}(\text{Obj}, \text{new}, I_2) \wedge$

$\text{prop}(\text{Obj}, \text{short}, I_3) \wedge \text{prop}(\text{Obj}, \text{home}, I_4) \wedge$

$V \text{ is } f(w_8 + w_9 \times I_1 + w_{10} \times I_2 + w_{11} \times I_3 + w_{12}^{32} \times I_4)$



# Prediction Error

- For particular values for the parameters  $\bar{w} = w_0, \dots, w_m$  and a set  $E$  of examples, the **sum-of-squares error** is

$$Error_E(\bar{w}) = \sum_{e \in E} (p_e^{\bar{w}} - o_e)^2,$$

- $p_e^{\bar{w}}$  is the predicted output by a neural network with parameter values given by  $\bar{w}$  for example  $e$
- $o_e$  is the observed output for example  $e$ .
- The aim of neural network learning is, given a set of examples, to find parameter settings that minimize the error.



# Neural Network Learning

- Aim of neural network learning: given a set of examples, find parameter settings that minimize the error.
- **Back-propagation learning** is gradient descent search through the parameter space to minimize the sum-of-squares error.



# Backpropagation Learning



## Inputs:

- A network, including all units and their connections
- Stopping Criteria
- Learning Rate (constant of proportionality of gradient descent search)
- Initial values for the parameters
- A set of classified training data



**Output:** Updated values for the parameters



# Backpropagation Learning Algorithm

- Repeat
  - evaluate the network on each example given the current parameter settings
  - determine the derivative of the error for each parameter
  - change each parameter in proportion to its derivative
- until the stopping criteria is met



# Gradient Descent for Neural Net Learning

- At each iteration, update parameter  $w_i$

$$w_i \leftarrow \left( w_i - \eta \frac{\partial error(w_i)}{\partial w_i} \right)$$

$\eta$  is the learning rate

- You can compute partial derivative:

- numerically: for small  $\Delta$

$$\frac{error(w_i + \Delta) - error(w_i)}{\Delta}$$

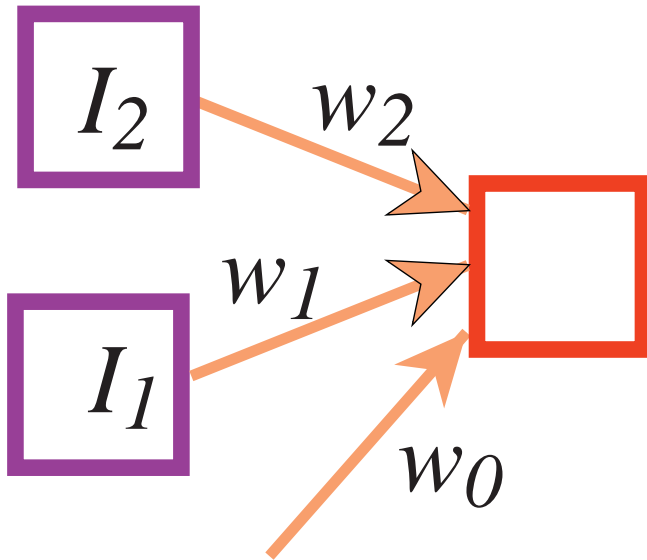
- analytically:  $f'(x) = f(x)(1 - f(x)) + \text{chain rule}$ <sub>37</sub>

# Simulation of Neural Net Learning

Parameter	iteration 0		iteration 1	iteration 80
	Value	Deriv	Value	Value
$w_0$	0.2	0.768	-0.18	-2.98
$w_1$	0.12	0.373	-0.07	6.88
$w_2$	0.112	0.425	-0.10	-2.10
$w_3$	0.22	0.0262	0.21	-5.25
$w_4$	0.23	0.0179	0.22	1.98
Error:	4.6121		4.6128	0.178



# What Can a Neural Network Represent?



$w_0$	$w_1$	$w_2$	Logic
-15	10	10	and
-5	10	10	or
5	-10	-10	nor

Output is  $f(w_0 + w_1 \times I_1 + w_2 \times I_2)$ .

A single unit can't represent *xor*.



# Bias in neural networks and decision trees

- It's easy for a neural network to represent “at least two of  $I_1, \dots, I_k$  are true”:

$$\begin{array}{cccc} w_0 & w_1 & \cdots & w_k \\ \hline -15 & 10 & \cdots & 10 \end{array}$$

This concept forms a large decision tree.

- Consider representing a conditional: “If  $c$  then  $a$  else  $b$ ”:
  - Simple in a decision tree.
  - Needs a complicated neural network to represent  $(c \wedge a) \vee (\neg c \wedge b)$ .





# Neural Networks and Logic

- Meaning is attached to the input and output units.
- There is no a priori meaning associated with the hidden units.
- What the hidden units actually represent is something that's learned.



# Case-based Reasoning

- **Idea:** experiences themselves are stored. These are called **cases.**
- Given a new example, the most appropriate case(s) in the knowledge base are found and these are used to predict properties of the new example.



# Extremes of Case-based Reasoning

- The cases are simple and for each new example the agent has seen many identical instances. Use the statistics of the cases.
- The cases are simple but there are few exact matches. Use a distance metric to find the closest cases.
- The cases are complex, there are no matches. You need sophisticated reasoning to determine why an old case is like the new case.

**Examples:** legal reasoning, case-based planning.



# $k$ -nearest Neighbors

- Need a distance metric between examples.
- Given a new example, find the  $k$  nearest neighbors of that example.
- Predict the classification by using the mode, median, or interpolating between the neighbors.
- Often want  $k > 1$  because there can be errors in the case base.



# Euclidean Distance

- Define a metric for each dimension (convert the values to a numerical scale).
- The **Euclidean distance** between examples  $x$  and  $y$  is:

$$d(x, y) = \sqrt{\sum_A w_A (x_A - y_A)^2}$$

- $x_A$  is the numerical value of attribute  $A$  for example  $x$
- $w_A$  is a nonnegative real-valued parameter that specifies the relative weight of attribute  $A$ .



## *kd-tree*

- Like a decision tree, but examples are stored at the leaves.
- The aim is to build a balanced tree; so a particular example can be found in  $\log n$  time when there are  $n$  examples.
- Not all leaves will be an exact match for a new example.
- Any exact match can be found in  $d = \log n$  time
- All examples that miss on just one attribute can be found in  $O(d^2)$  time.



# Learning Under Uncertainty

- We want to learn models from data.

$$P(model|data) = \frac{P(data|model) \times P(model)}{P(data)}.$$

- The **likelihood**,  $P(data|model)$ , is the probability that this model would have produced this data.
- The **prior**,  $P(model)$ , encodes the learning bias



# Bayesian Learning of Probabilities

- Suppose there are two outcomes  $A$  and  $\neg A$ . We would like to learn the probability of  $A$  given some data.
- We can treat the probability of  $A$  as a real-valued random variable on the interval  $[0, 1]$ , called  $probA$ .

$$P(probA=p|data) = \frac{P(data|probA=p) \times P(probA=p)}{P(data)}$$

- Suppose the data is a sequence of  $n$   $A$ 's out of independent  $m$  trials,

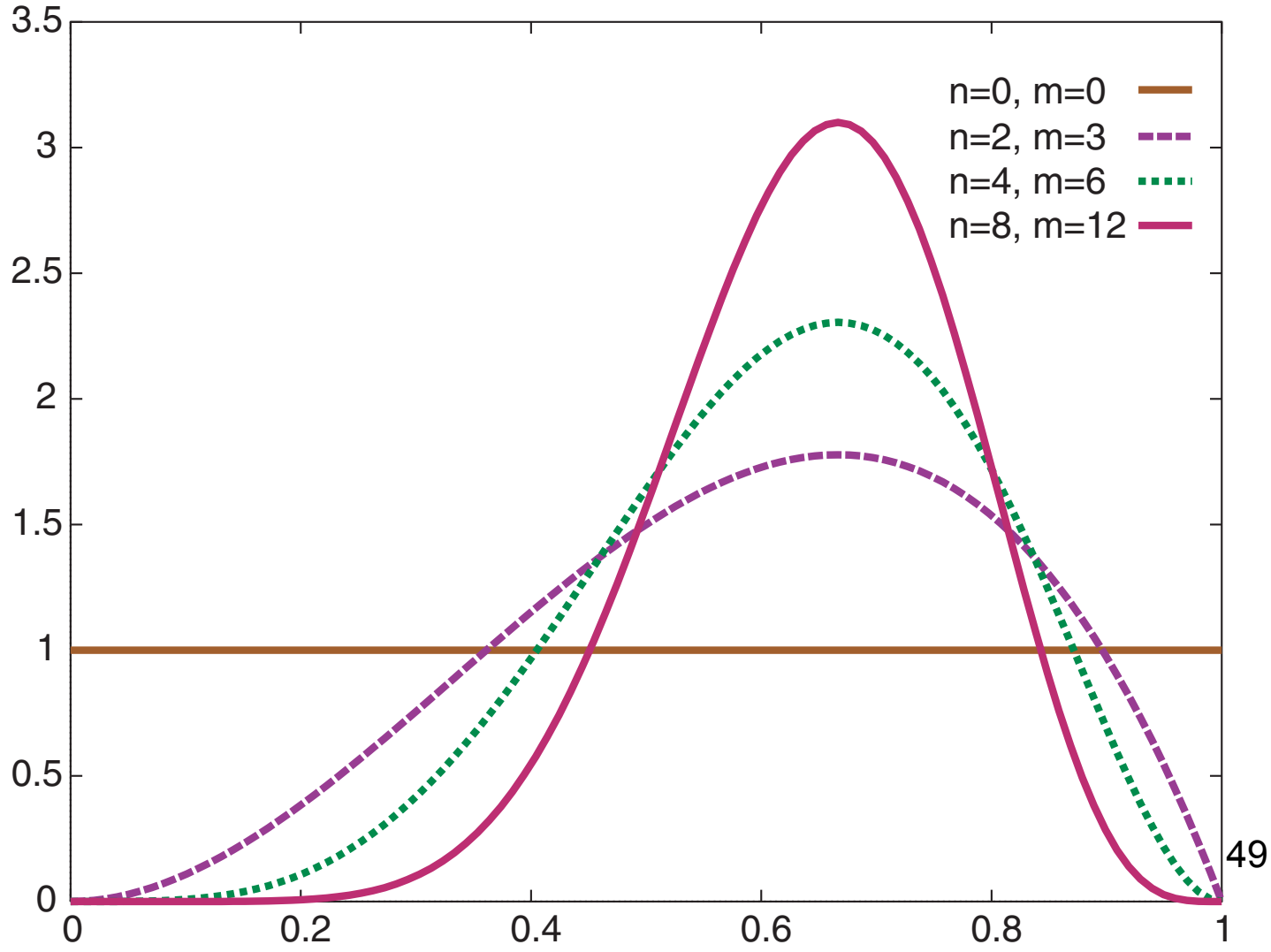
$$P(data|probA=p) = p^n \times (1 - p)^{m-n}$$

- Uniform prior:  $P(probA=p) = 1$  for all  $p \in [0, 1]$ .





# Posterior Probabilities for Different Data



# MAP model

- The **maximum a posteriori probability** (MAP) model is the model that maximizes  $P(model|data)$ . That is, it maximizes:

$$P(data|model) \times P(model)$$

- Thus it minimizes:

$$(-\log P(data|model)) + (-\log P(model))$$

which is the number of bits to send the data given the model plus the number of bits to send the model.



# Information theory overview

- A **bit** is a binary digit.
- 1 bit can distinguish 2 items
- $k$  bits can distinguish  $2^k$  items
- $n$  items can be distinguished using  $\log_2 n$  bits
- Can you do better?



# Information and Probability

Let's design a code to distinguish elements of  $\{a, b, c, d\}$  with

$$P(a) = \frac{1}{2}, P(b) = \frac{1}{4}, P(c) = \frac{1}{8}, P(d) = \frac{1}{8}$$

Consider the code:

$a$  0             $b$  10             $c$  110             $d$  111

This code sometimes uses 1 bit and sometimes uses 3 bits.

On average, it uses

$$\begin{aligned} &P(a) \times 1 + P(b) \times 2 + P(c) \times 3 + P(d) \times 3 \\ &= \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{3}{8} = 1\frac{3}{4} \text{ bits.} \end{aligned}$$

The string  $aacabbda$  has code 00110010101110.



# Information Content

- To identify  $x$ , you need  $-\log_2 P(x)$  bits.
- If you have a distribution over a set and want to identify a member, you need the expected number of bits:

$$\sum_x -P(x) \times \log_2 P(x).$$

This is the **information content** or **entropy** of the distribution.

- The expected number of bits it takes to describe a distribution given evidence  $e$ :

$$I(e) = \sum_x -P(x|e) \times \log_2 P(x|e).$$



# Information Gain

If you have a test that can distinguish the cases where  $\alpha$  is true from the cases where  $\alpha$  is false, the **information gain** from this test is:

$$I(\text{true}) - (P(\alpha) \times I(\alpha) + P(\neg\alpha) \times I(\neg\alpha)).$$

- $I(\text{true})$  is the expected number of bits needed before the test
- $P(\alpha) \times I(\alpha) + P(\neg\alpha) \times I(\neg\alpha)$  is the expected number of bits after the test.



# Averaging Over Models

- **Idea:** Rather than choosing the most likely model, average over all models, weighted by their posterior probabilities given the data.
- If you have observed  $n$   $A$ 's out of  $m$  trials
  - the most likely value (MAP) is  $\frac{n}{m}$
  - the expected value is  $\frac{n+1}{m+2}$



# Learning a Belief Network

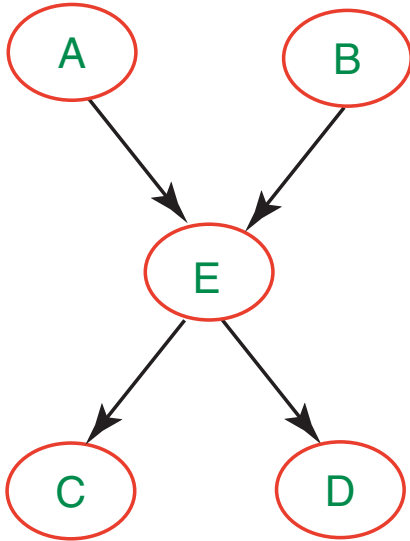
- If you
  - know the structure
  - have observed all of the variables
  - have no missing data
- you can learn each conditional probability separately.





# Learning belief network example

Model



Data

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
<i>t</i>	<i>f</i>	<i>t</i>	<i>t</i>	<i>f</i>
<i>f</i>	<i>t</i>	<i>t</i>	<i>t</i>	<i>t</i>
<i>t</i>	<i>t</i>	<i>f</i>	<i>t</i>	<i>f</i>
...				

→ Probabilities

$P(A)$

$P(B)$

$P(E|A, B)$

$P(C|E)$

$P(D|E)$



# Learning conditional probabilities

- Each conditional probability distribution can be learned separately:
- For example:

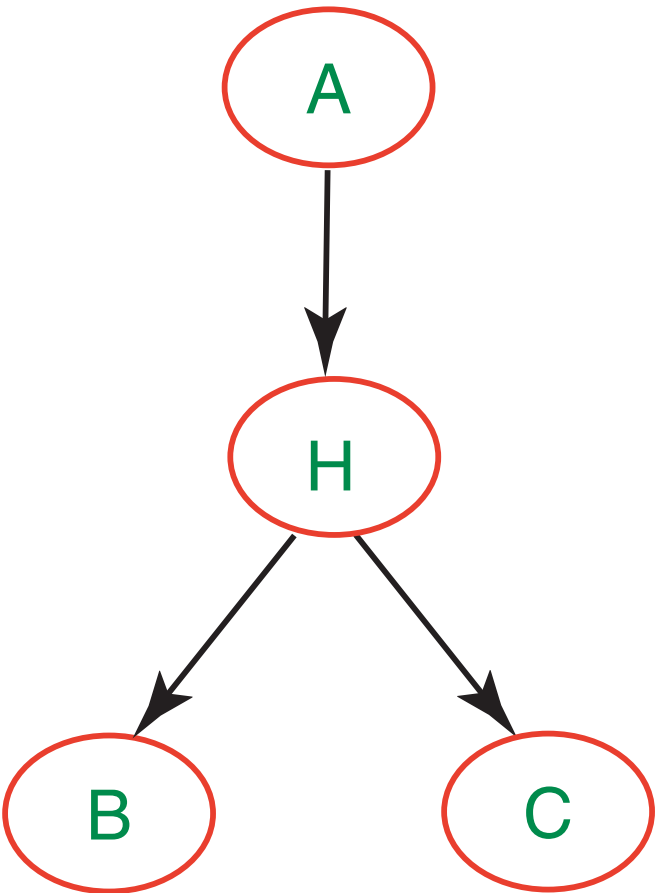
$$P(E = t | A = t \wedge B = f) \\ = \frac{(\text{\#examples: } E = t \wedge A = t \wedge B = f) + m_1}{(\text{\#examples: } A = t \wedge B = f) + m}$$

where  $m_1$  and  $m$  reflect our prior knowledge.

- There is a problem when there are many parents to a node as then there is little data for each probability estimate.



# Unobserved Variables



➤ What if we had only observed values for  $A, B, C$ ?

$A$	$B$	$C$
$t$	$f$	$t$
$f$	$t$	$t$
$t$	$t$	$f$
	...	



# EM Algorithm

## Augmented Data

<i>A</i>	<i>B</i>	<i>C</i>	<i>H</i>
<i>t</i>	<i>f</i>	<i>t</i>	<i>t</i>
<i>f</i>	<i>t</i>	<i>t</i>	<i>f</i>
<i>t</i>	<i>t</i>	<i>f</i>	<i>t</i>
	...		

## Probabilities

$$P(A)$$

$$P(H|A)$$


$$P(B|H)$$

$$P(C|H)$$

E-step



M-step

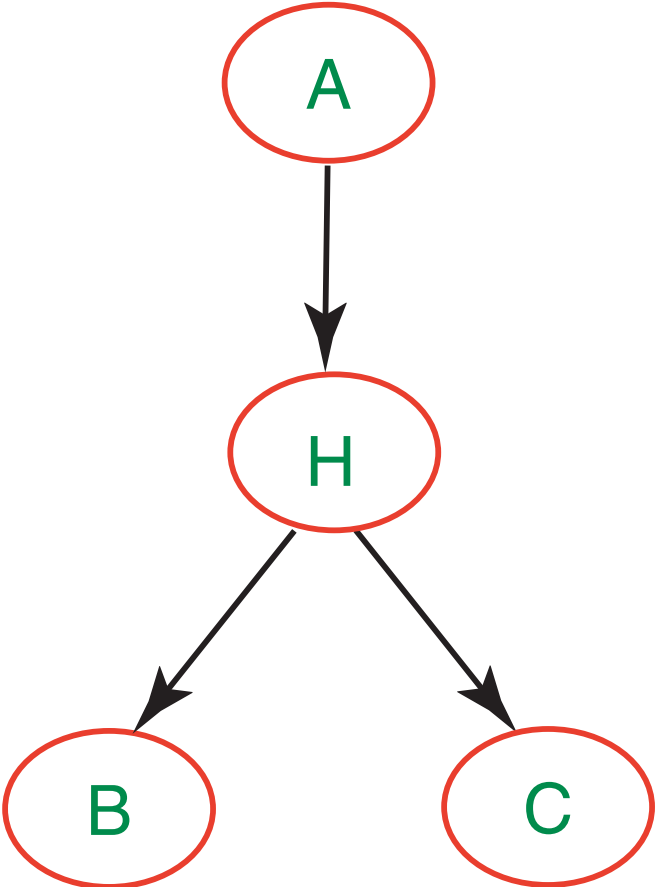


# EM Algorithm

- Repeat the following two steps:
  - **E-step** give the expected number of data points for the unobserved variables based on the given probability distribution.
  - **M-step** infer the (maximum likelihood) probabilities from the data. This is the same as the full observable case.
- Start either with made-up data or made-up probabilities.
- EM will converge to a local maxima.



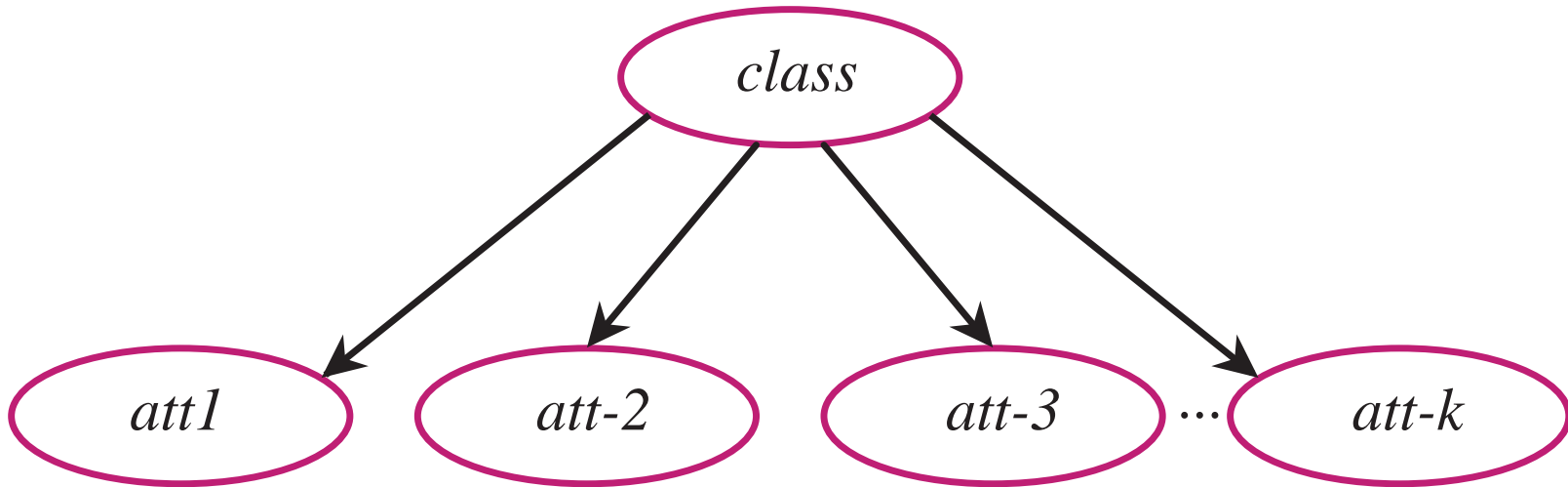
# Example Data



A	B	C	Count
t	t	t	143
t	t	f	329
t	f	t	57
t	f	f	271
f	t	t	87
f	t	f	66
f	f	t	23
f	f	f	24



# Naive Bayesian Classifier



# Unsupervised Learning

- Given a collection of data, find natural classifications.
- This can be seen as the naive Bayesian classifier with the classification unobserved.
- EM can be used to learn classification.





# Bayesian learning of decision trees

$$P(\text{model}|\text{data}) = \frac{P(\text{data}|\text{model}) \times P(\text{model})}{P(\text{data})}.$$

- A model here is a decision tree
- We allow for decision trees with probabilities at the leaves
- A bigger decision tree can always fit the data better
- $P(\text{model})$  lets us encode a preference for smaller decision trees.



# Data for decision tree learning

<i>att</i> <sub>1</sub>	<i>att</i> <sub>2</sub>	<i>class</i>	count
t	t	c1	10
t	t	c2	3
t	f	c1	5
t	f	c2	12
f	t	c1	7
f	t	c2	14
f	f	c1	8
f	f	c2	1



# Probabilities From Experts

- Bayes rule lets us combine expert knowledge with data

$$P(model|data) = \frac{P(data|model) \times P(model)}{P(data)}.$$

- The experts prior knowledge of the model (i.e.,  $P(model)$ ) can be expressed as a pair  $\langle n, m \rangle$  that can be interpreted as though they had observed  $n$   $A$ 's out of  $m$  trials.
- This estimate can be combined with data.
- Estimates from multiple experts can be combined together.

