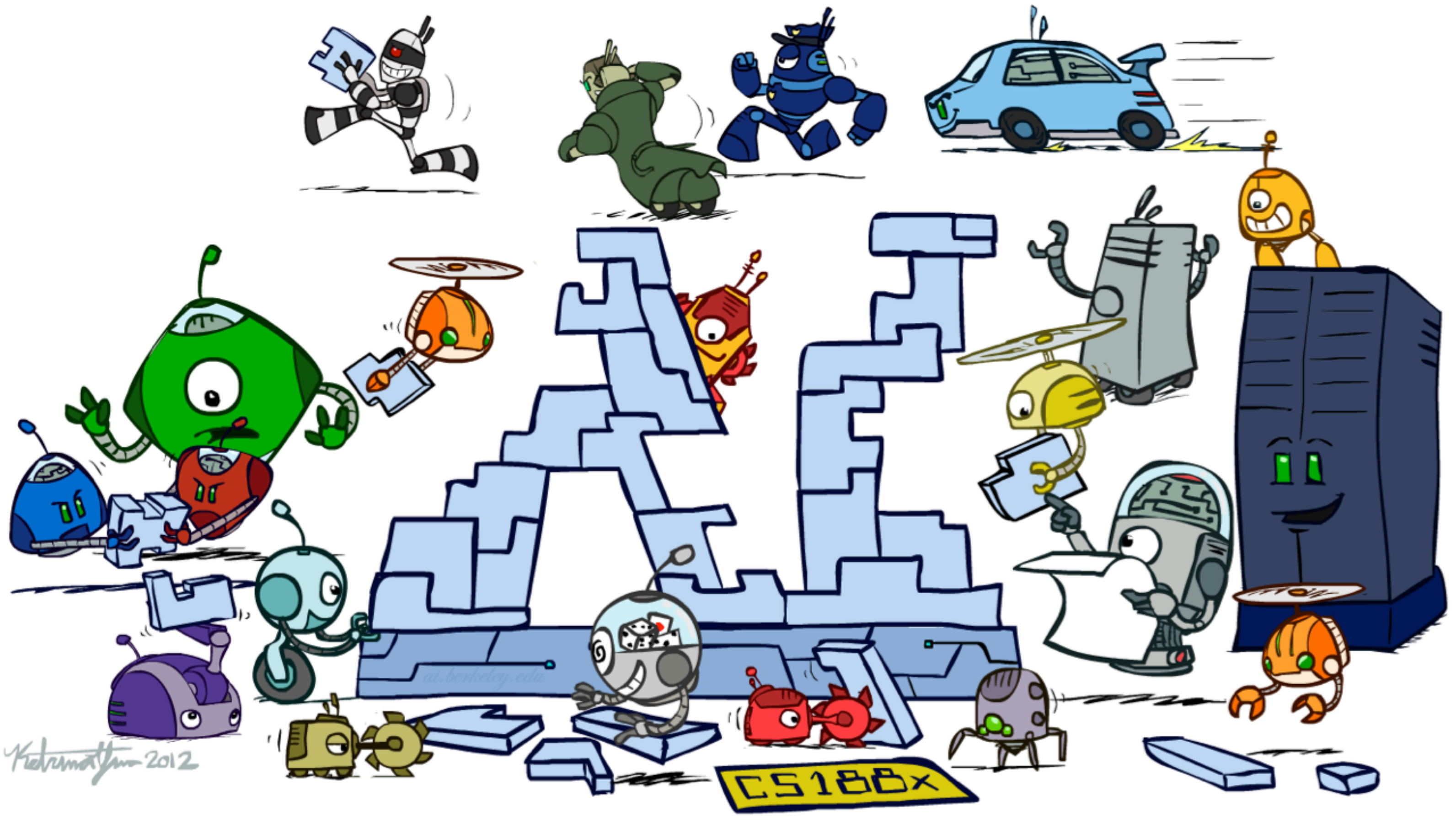
CS 188: Artificial Intelligence

Final Course Summary



Instructor: Oliver Grillmeyer --- University of California, Berkeley

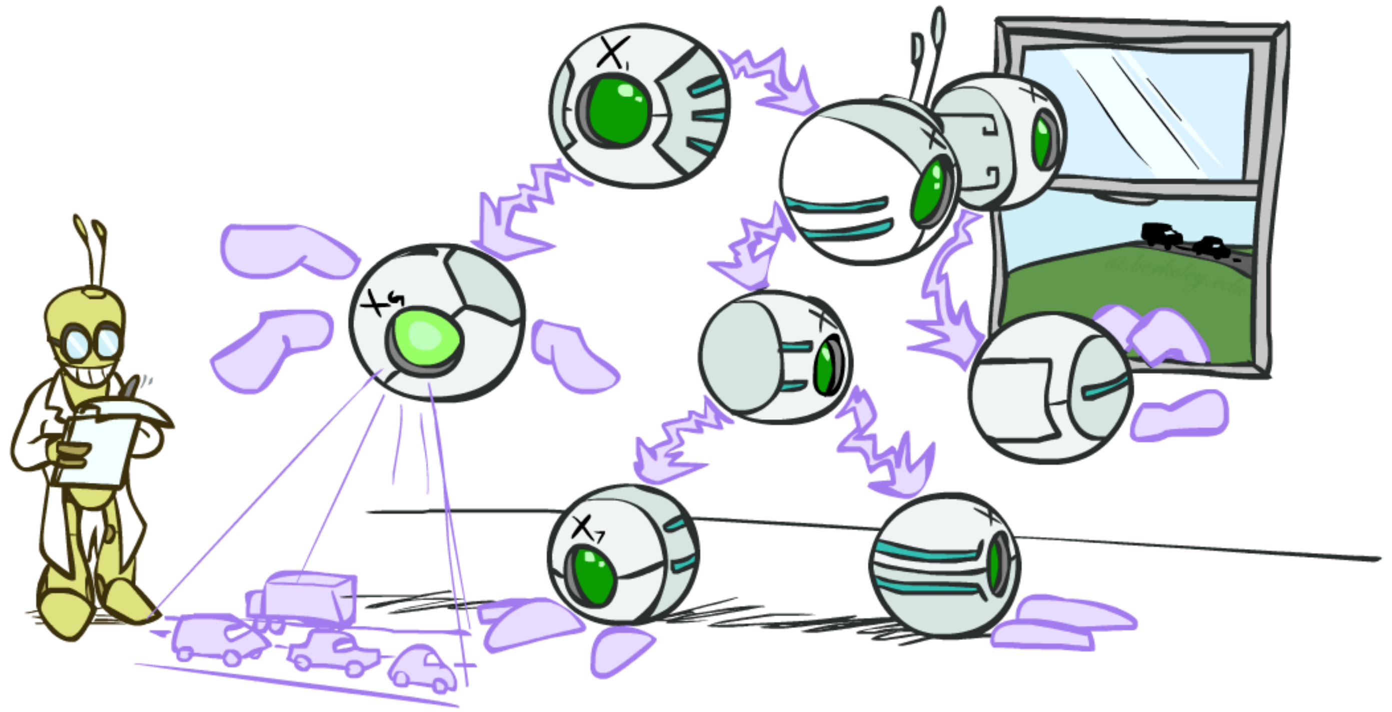
[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Announcements

- Final Exam is **Wednesday, August 13**, 7-10 PM PT in 2050 VLSB
- Please fill out course evaluations

CS 188: Artificial Intelligence

Bayes' Nets: Inference



Instructors: Oliver Grillmeyer and Charlie Snell — University of California, Berkeley

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Inference

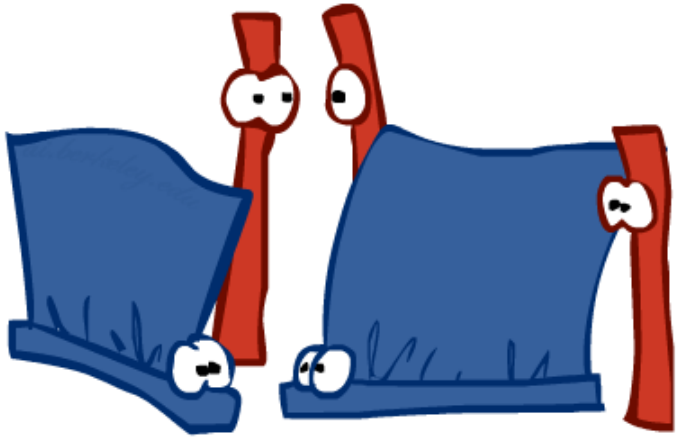
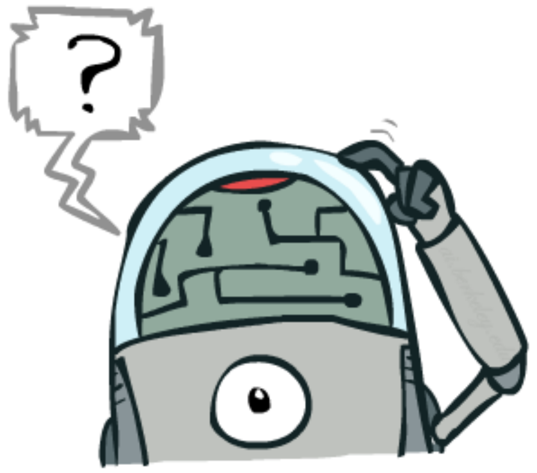
- Inference: calculating some useful quantity from a joint probability distribution

- Examples:
  - Posterior probability

    $P(Q|E_1 = e_1, \ldots E_k = e_k)$

  - Most likely explanation:

    $\text{argmax}_q \; P(Q = q|E_1 = e_1 \ldots)$

Inference by Enumeration

- General case:
  - Evidence variables: $E_1 \ldots E_k = e_1 \ldots e_k$
  - Query* variable: Q
  - Hidden variables: $H_1 \ldots H_r$

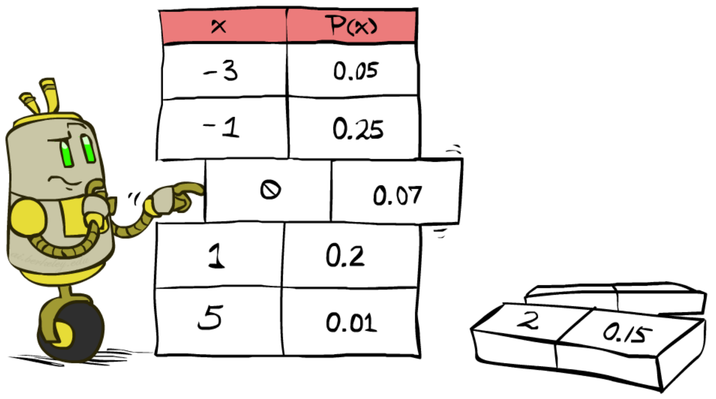
$\left.\right\}$ $X_1, X_2, \ldots X_n$ *All variables*

- We want:

** Works fine with multiple query variables, too*

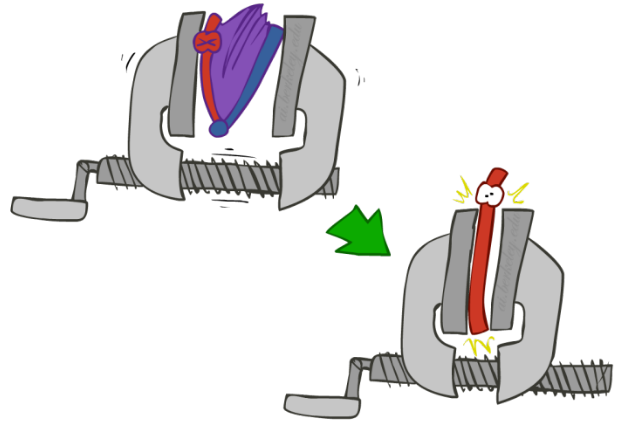
$$P(Q|e_1 \ldots e_k)$$

- Step 1: Select the entries consistent with the evidence



x	P(x)
-3	0.05
-1	0.25
0	0.07
1	0.2
5	0.01
2	0.15

- Step 2: Sum out H to get joint of Query and evidence



$$P(Q, e_1 \ldots e_k) = \sum_{h_1 \ldots h_r} \underbrace{P(Q, h_1 \ldots h_r, e_1 \ldots e_k)}_{X_1, X_2, \ldots X_n}$$

- Step 3: Normalize

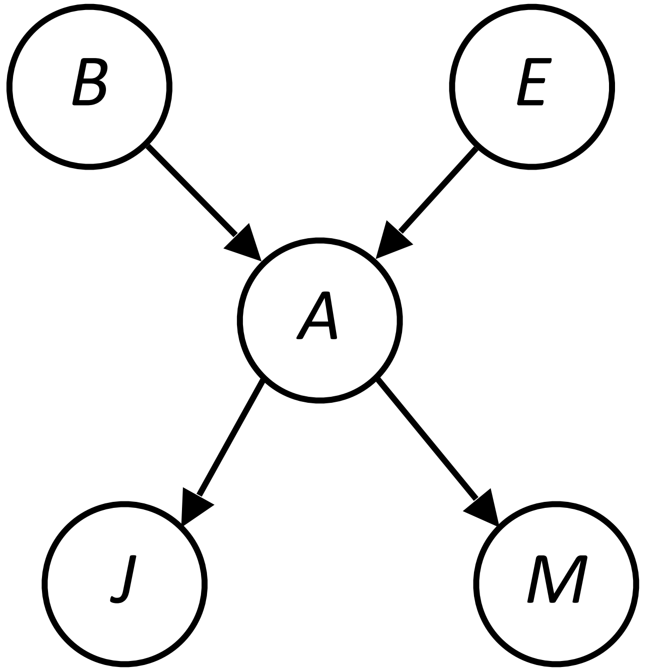
$$\times \frac{1}{Z}$$

$$Z = \sum_q P(Q, e_1 \cdots e_k)$$

$$P(Q|e_1 \cdots e_k) = \frac{1}{Z} P(Q, e_1 \cdots e_k)$$

Inference by Enumeration in Bayes' Net

- Given unlimited time, inference in BNs is easy
- Reminder of inference by enumeration by example:



$$P(B \mid +j, +m) \;\propto_B\; P(B, +j, +m)$$

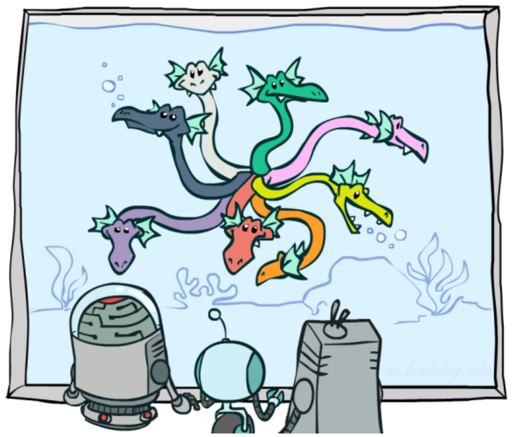
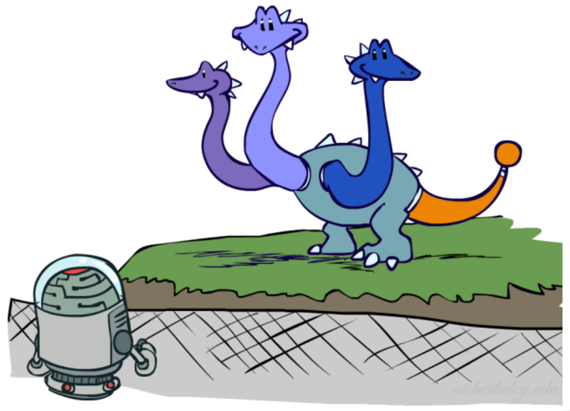
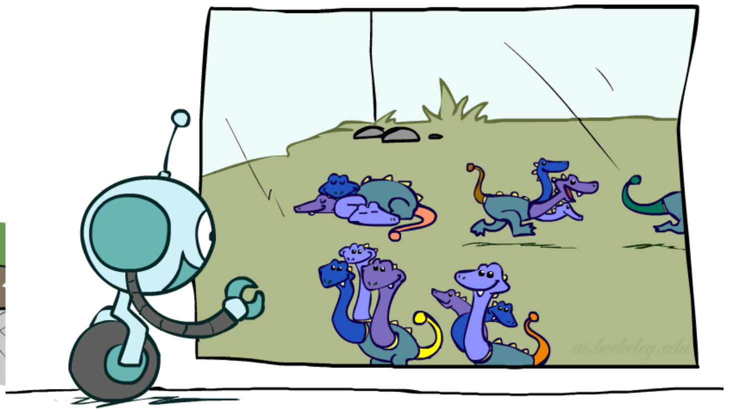
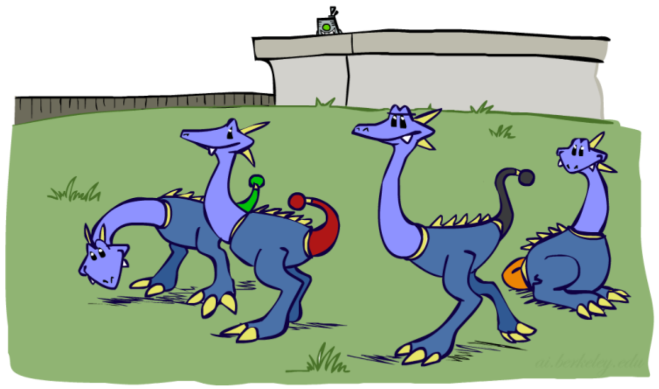
$$= \sum_{e,a} P(B, e, a, +j, +m)$$

$$= \sum_{e,a} P(B)P(e)P(a|B, e)P(+j|a)P(+m|a)$$

$$= P(B)P(+e)P(+a|B, +e)P(+j| + a)P(+m| + a) + P(B)P(+e)P(-a|B, +e)P(+j| - a)P(+m| - a)$$
$$P(B)P(-e)P(+a|B, -e)P(+j| + a)P(+m| + a) + P(B)P(-e)P(-a|B, -e)P(+j| - a)P(+m| - a)$$

Factor Summary

- In general, when we write $P(Y_1 \ldots Y_N \mid X_1 \ldots X_M)$
  - It is a "factor," a multi-dimensional array
  - Its values are $P(y_1 \ldots y_N \mid x_1 \ldots x_M)$
  - Any assigned (=lower-case) X or Y is a dimension selected from the array

Inference by Enumeration: Procedural Outline

- Track objects called factors
- Initial factors are local CPTs (one per node)

$P(R)$

+r	0.1
-r	0.9

$P(T|R)$

+r	+t	0.8
+r	-t	0.2
-r	+t	0.1
-r	-t	0.9

$P(L|T)$

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

- Any known values are selected
  - E.g. if we know $L = +\ell$, the initial factors are

$P(R)$

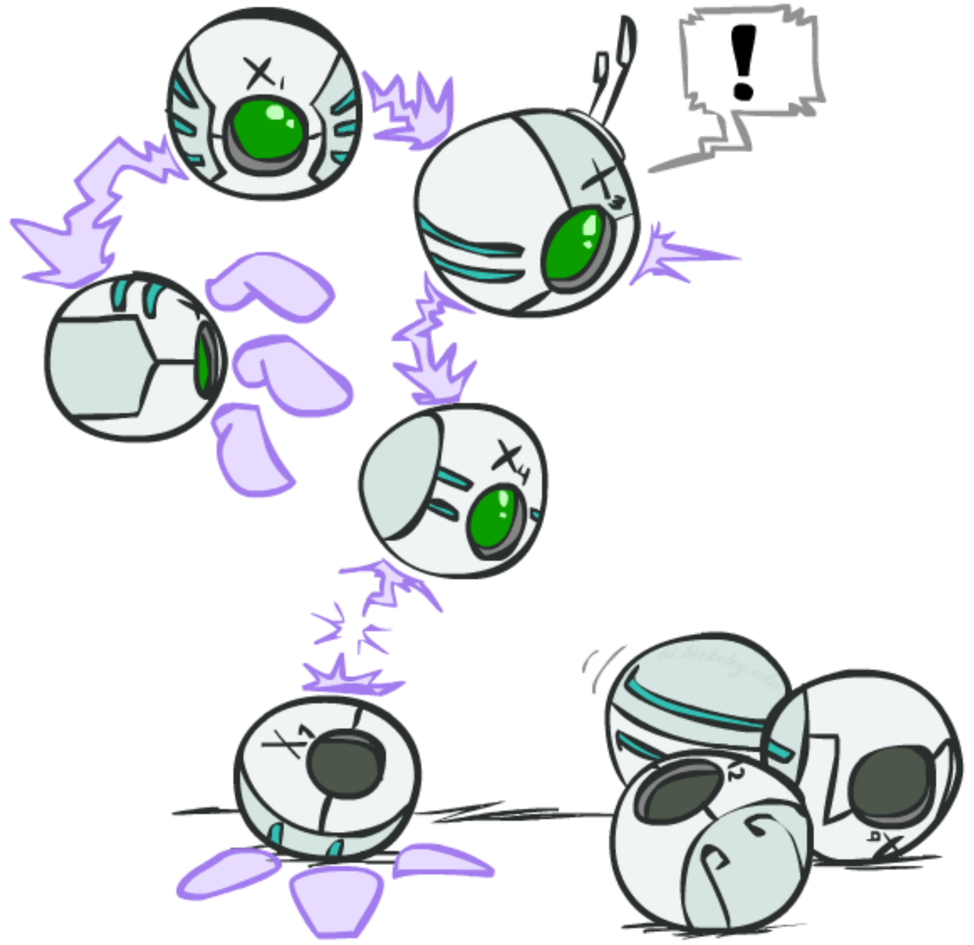
+r	0.1
-r	0.9

$P(T|R)$

+r	+t	0.8
+r	-t	0.2
-r	+t	0.1
-r	-t	0.9

$P(+\ell|T)$

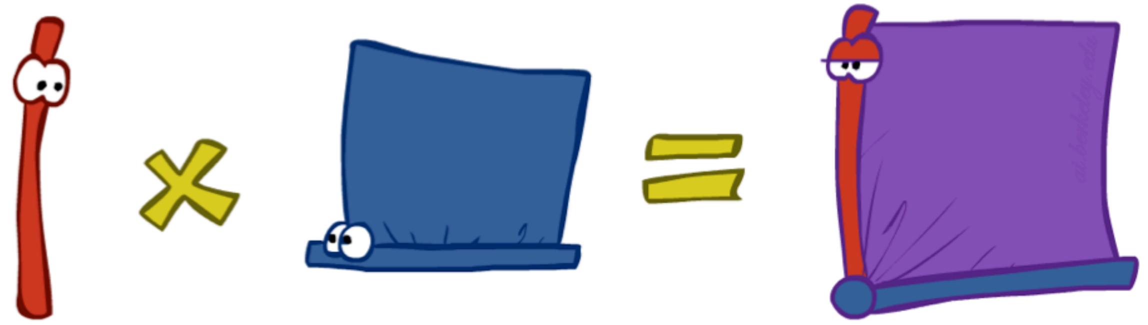
+t	+l	0.3
-t	+l	0.1



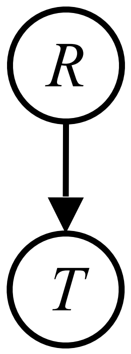
- Procedure: Join all factors, eliminate all hidden variables, normalize

Operation 1: Join Factors

- First basic operation: joining factors
- Combining factors:
  - Just like a database join
  - Get all factors over the joining variable
  - Build a new factor over the union of the variables involved



- Example: Join on R

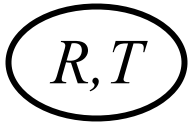


$P(R) \times P(T|R) \rightarrow P(R,T)$

+r	0.1
-r	0.9

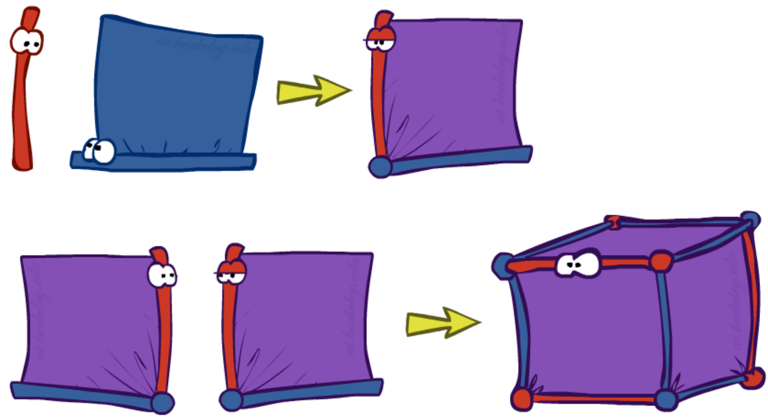
+r	+t	0.8
+r	-t	0.2
-r	+t	0.1
-r	-t	0.9

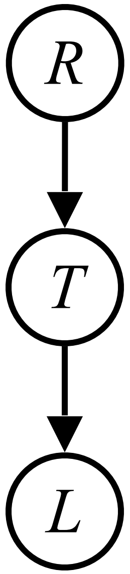
+r	+t	0.08
+r	-t	0.02
-r	+t	0.09
-r	-t	0.81



- Computation for each entry: pointwise products $\forall r, t: \quad P(r,t) = P(r) \cdot P(t|r)$

Example: Multiple Joins





$P(R)$

+r	0.1
-r	0.9

$P(T|R)$

+r	+t	0.8
+r	-t	0.2
-r	+t	0.1
-r	-t	0.9

$P(L|T)$

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

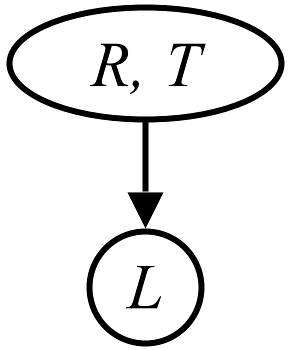
Join R

$P(R,T)$

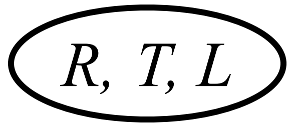
+r	+t	0.08
+r	-t	0.02
-r	+t	0.09
-r	-t	0.81

$P(L|T)$

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9



Join T



$P(R,T,L)$

+r	+t	+l	0.024
+r	+t	-l	0.056
+r	-t	+l	0.002
+r	-t	-l	0.018
-r	+t	+l	0.027
-r	+t	-l	0.063
-r	-t	+l	0.081
-r	-t	-l	0.729

Operation 2: Eliminate

- Second basic operation: marginalization
- Take a factor and sum out a variable
  - Shrinks a factor to a smaller one
  - A projection operation
- Example:

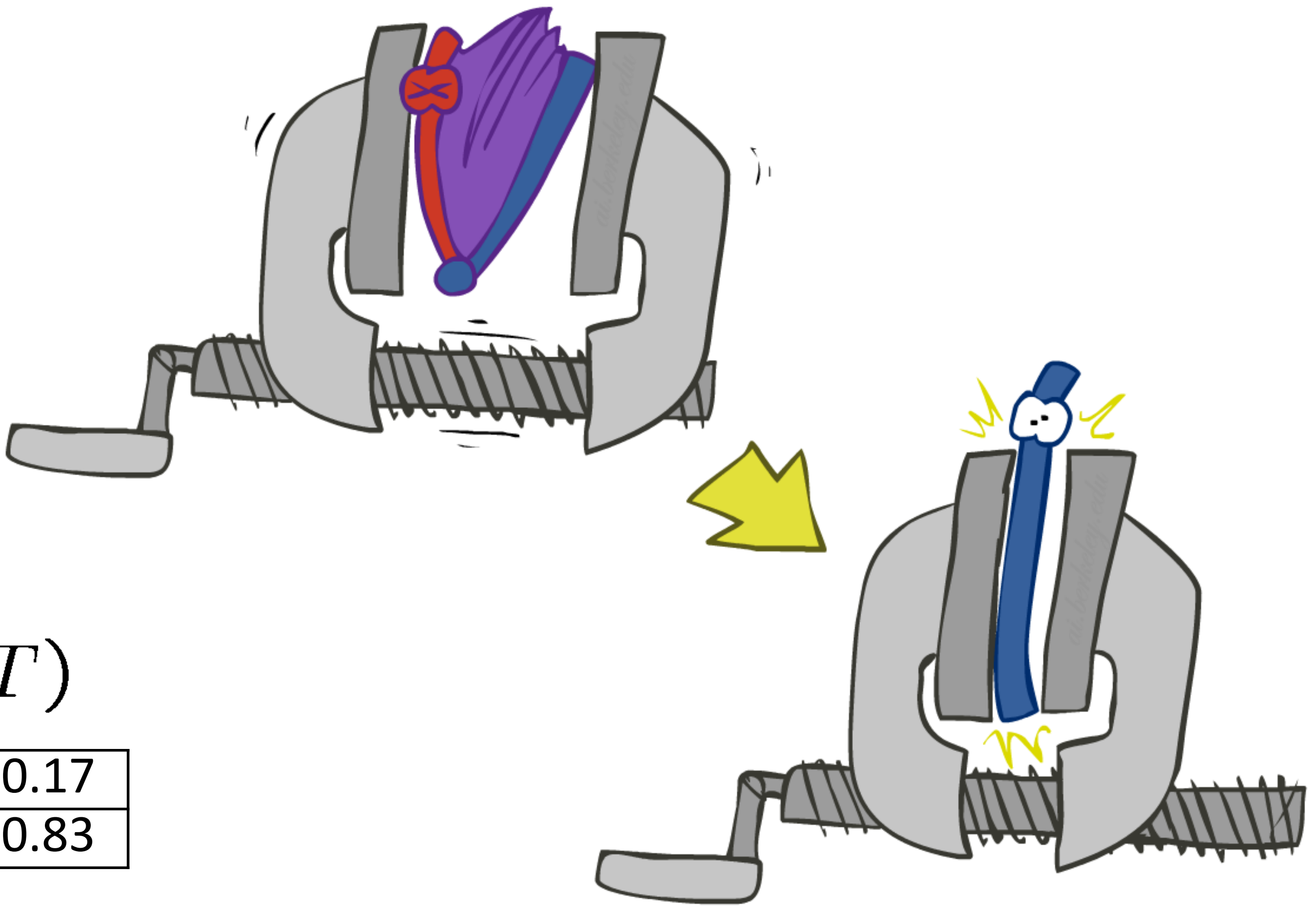
$P(R,T)$

+r	+t	0.08
+r	-t	0.02
-r	+t	0.09
-r	-t	0.81

$\xrightarrow{\text{sum } R}$

$P(T)$

+t	0.17
-t	0.83



Multiple Elimination

R, T, L

$P(R, T, L)$

+r	+t	+l	0.024
+r	+t	-l	0.056
+r	-t	+l	0.002
+r	-t	-l	0.018
-r	+t	+l	0.027
-r	+t	-l	0.063
-r	-t	+l	0.081
-r	-t	-l	0.729

Sum out R $\rightarrow$

T, L

$P(T, L)$

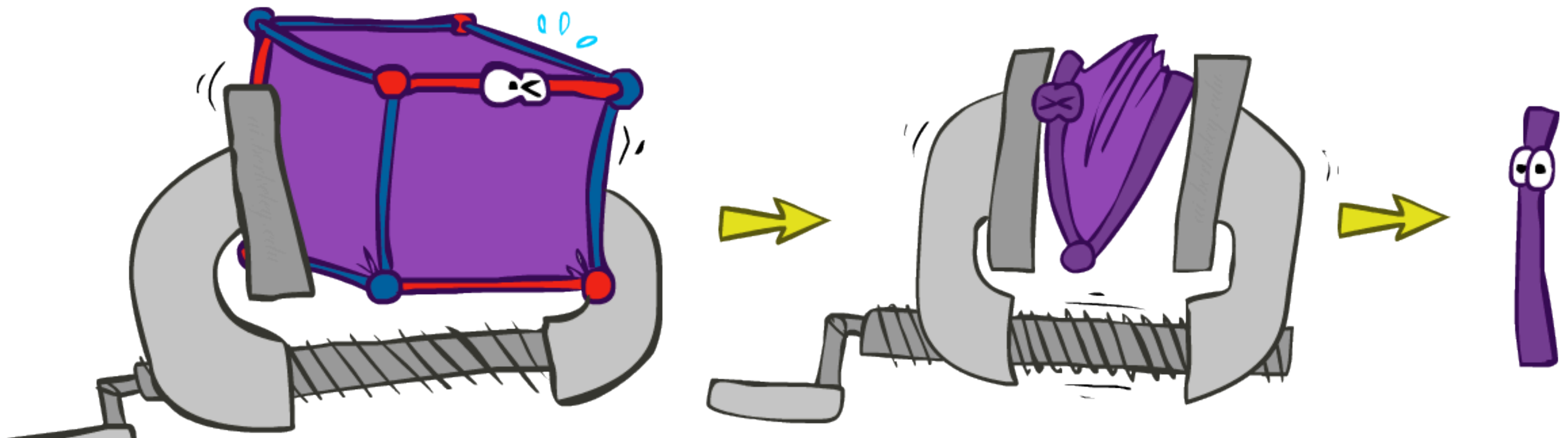
+t	+l	0.051
+t	-l	0.119
-t	+l	0.083
-t	-l	0.747

Sum out T $\rightarrow$

L

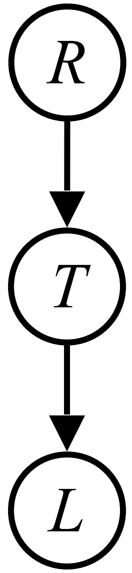
$P(L)$

+l	0.134
-l	0.886



Marginalizing Early! (aka VE)

Join R → Sum out R → Join T → Sum out T



$P(R)$

+r	0.1
-r	0.9

$P(T|R)$

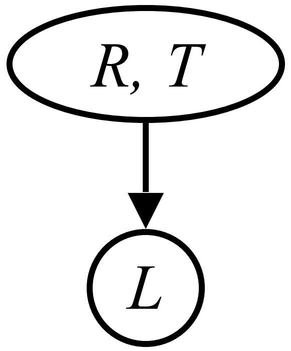
+r	+t	0.8
+r	-t	0.2
-r	+t	0.1
-r	-t	0.9

$P(L|T)$

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

$P(R,T)$

+r	+t	0.08
+r	-t	0.02
-r	+t	0.09
-r	-t	0.81

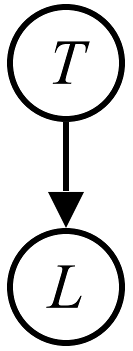


$P(L|T)$

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

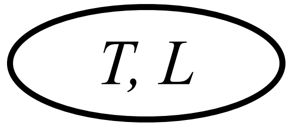
$P(T)$

+t	0.17
-t	0.83



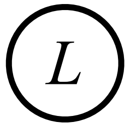
$P(L|T)$

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9



$P(T,L)$

+t	+l	0.051
+t	-l	0.119
-t	+l	0.083
-t	-l	0.747

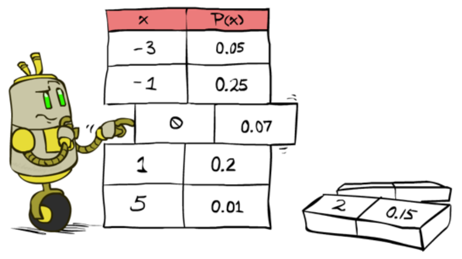


$P(L)$

+l	0.134
-l	0.866

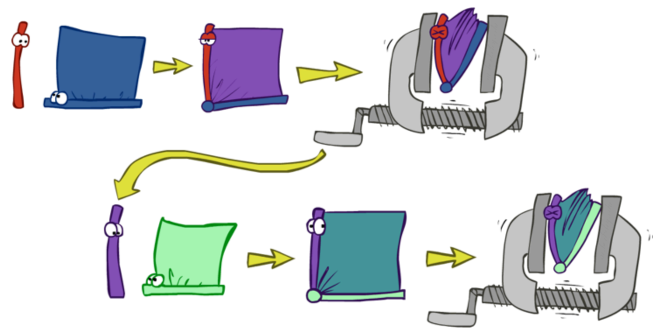
General Variable Elimination

- Query: $P(Q|E_1 = e_1, \dots E_k = e_k)$



- Start with initial factors:
  - Local CPTs (but instantiated by evidence)

- While there are still hidden variables (not Q or evidence):
  - Pick a hidden variable H_i
  - Join all factors mentioning H_i
  - Eliminate (sum out) H_i

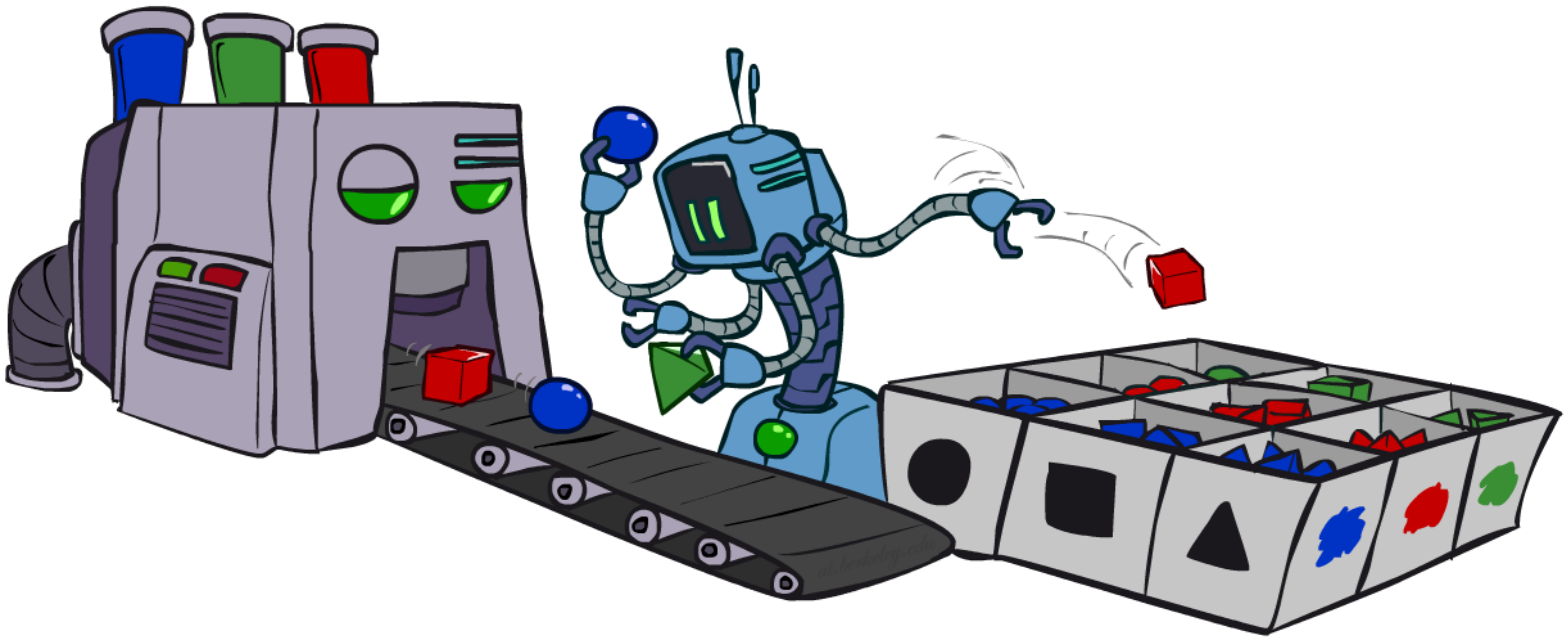


- Join all remaining factors and normalize

$$\times \frac{1}{Z}$$

CS 188: Artificial Intelligence

Bayes' Nets: Sampling



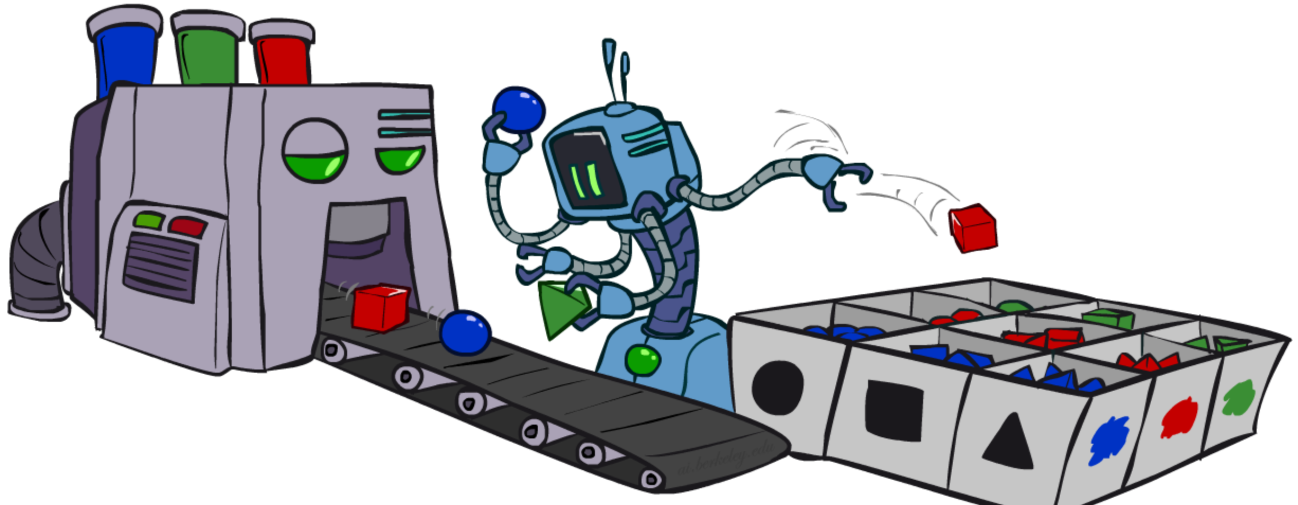
Instructor: Oliver Grillmeyer --- University of California, Berkeley

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Sampling

- Sampling is a lot like repeated simulation
  - Predicting the weather, basketball games, …
- Basic idea
  - Draw N samples from a sampling distribution S
  - Compute an approximate posterior probability
  - Show this converges to the true probability P

- Why sample?
  - Learning: get samples from a distribution you don't know
  - Inference: getting a sample is faster than computing the right answer (e.g. with variable elimination)



Sampling

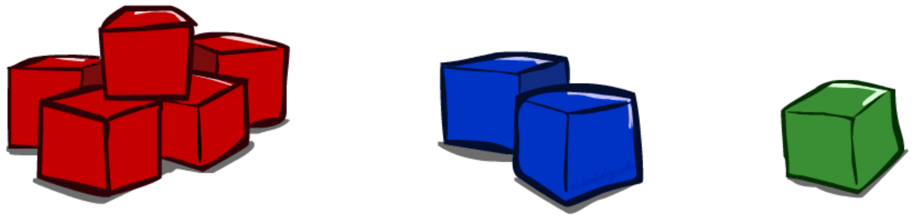
- Sampling from given distribution
  - Step 1: Get sample *u* from uniform distribution over [0, 1)
    - E.g. random() in python
  - Step 2: Convert this sample *u* into an outcome for the given distribution by having each target outcome associated with a sub-interval of [0,1) with sub-interval size equal to probability of the outcome

- Example

C	P(C)
red	0.6
green	0.1
blue	0.3

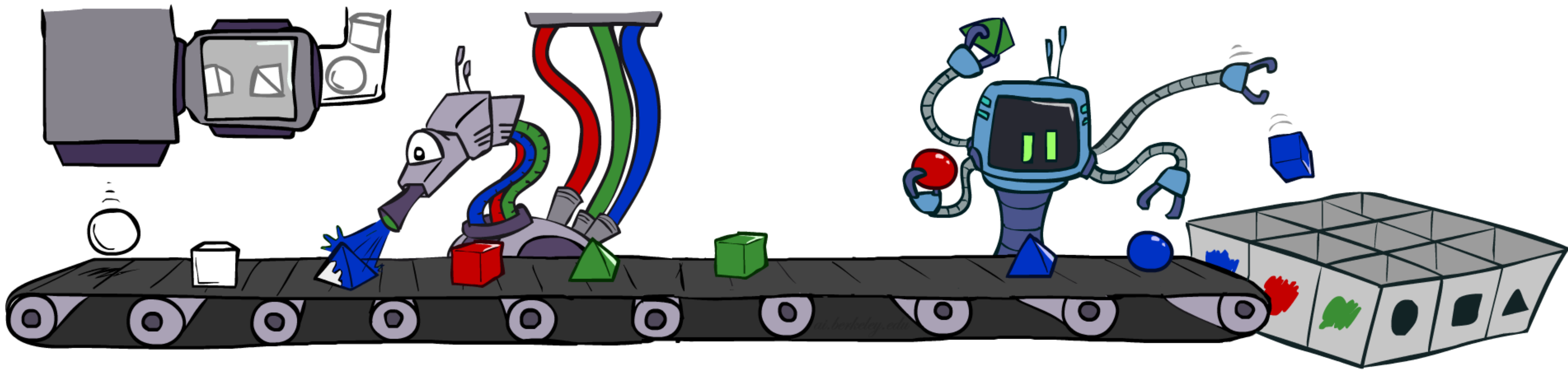
$$0 \leq u < 0.6, \rightarrow C = red$$
$$0.6 \leq u < 0.7, \rightarrow C = green$$
$$0.7 \leq u < 1, \rightarrow C = blue$$

- If random() returns *u* = 0.83, then our sample is *C* = blue
- E.g, after sampling 8 times:



Prior Sampling

- For $i = 1, 2, \ldots, n$
  - Sample x_i from $P(X_i \mid \text{Parents}(X_i))$
- Return $(x_1, x_2, \ldots, x_n)$


ai.berkeley.edu

Prior Sampling

$P(C)$

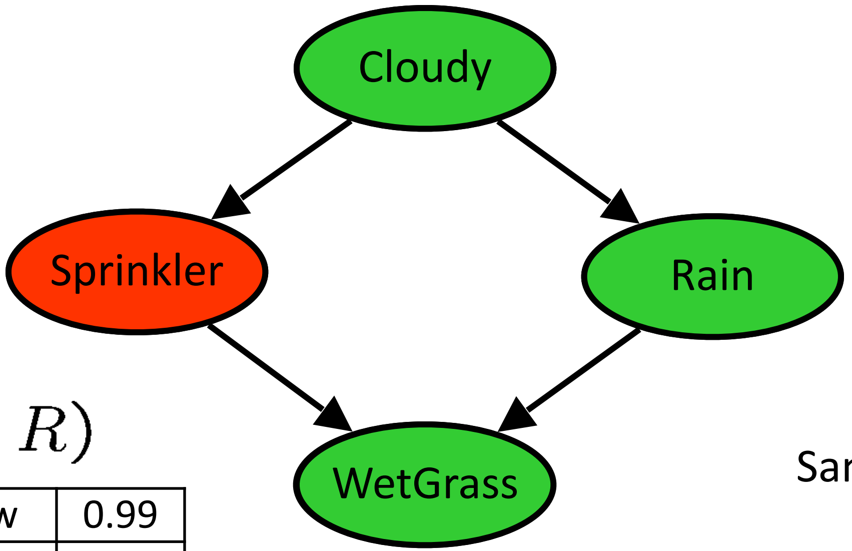
+c	0.5
-c	0.5

$P(S|C)$

+c	+s	0.1
	-s	0.9
-c	+s	0.5
	-s	0.5

$P(R|C)$

+c	+r	0.8
	-r	0.2
-c	+r	0.2
	-r	0.8



$P(W|S,R)$

+s	+r	+w	0.99
		-w	0.01
	-r	+w	0.90
		-w	0.10
-s	+r	+w	0.90
		-w	0.10
	-r	+w	0.01
		-w	0.99

Samples:

+c, -s, +r, +w

-c, +s, -r, +w

…

Example

- We'll get a bunch of samples from the BN:

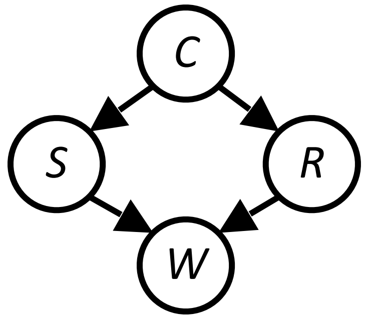
  +c, -s, +r, +w

  +c, +s, +r, +w

  -c, +s, +r, -w

  +c, -s, +r, +w

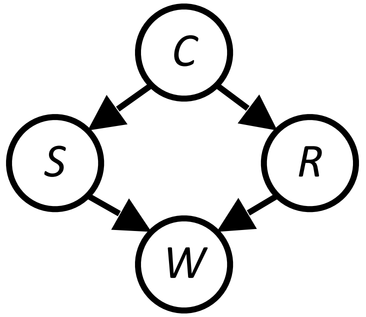
  -c, -s, -r, +w



- If we want to know P(W)
  - We have counts <+w:4, -w:1>
  - Normalize to get P(W) = <+w:0.8, -w:0.2>
  - This will get closer to the true distribution with more samples
  - Can estimate anything else, too
  - What about P(C | +w)?   P(C | +r, +w)? P(C | -r, -w)?
  - Fast: can use fewer samples if less time (what's the drawback?)

Rejection Sampling

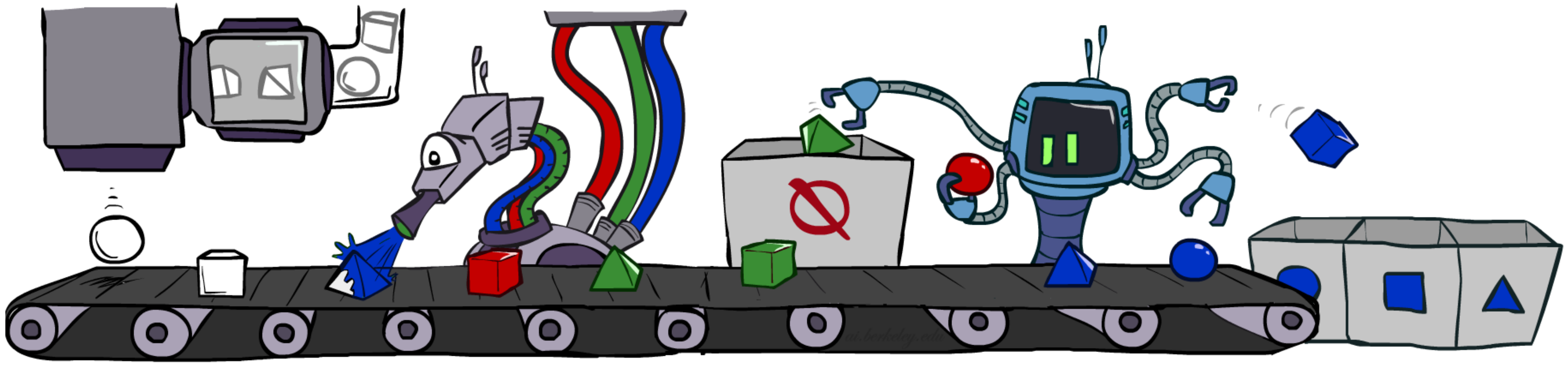
- Let’s say we want P(C)
  - No point keeping all samples around
  - Just tally counts of C as we go
- Let’s say we want P(C | +s)
  - Same thing: tally C outcomes, but ignore (reject) samples which don’t have S=+s
  - This is called rejection sampling
  - It is also consistent for conditional probabilities (i.e., correct in the limit)


$$\begin{tikzcd} & C \arrow[dl] \arrow[dr] & \\ S \arrow[dr] & & R \arrow[dl] \\ & W & \end{tikzcd}$$

+c, -s, +r, +w
+c, +s, +r, +w
-c, +s, +r, -w
+c, -s, +r, +w
-c, -s, -r, +w

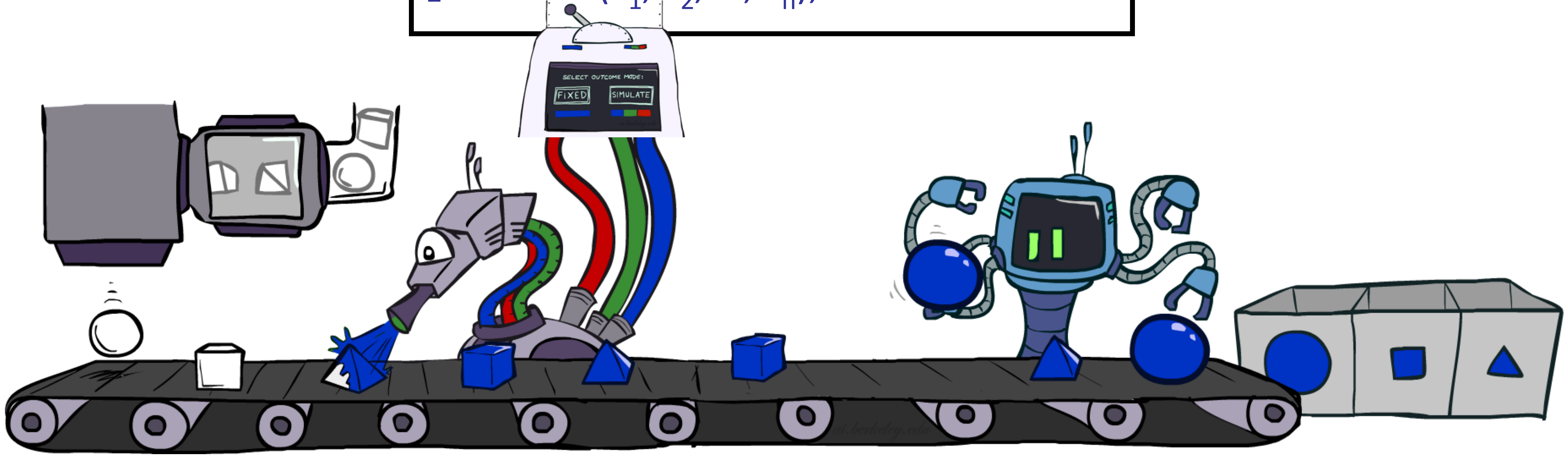
Rejection Sampling

```
- Input: evidence instantiation
- For i = 1, 2, …, n
    - Sample $x_i$ from $P(X_i \mid Parents(X_i))$
    - If $x_i$ not consistent with evidence
        - Reject: return – no sample is generated in this cycle
- Return $(x_1, x_2, …, x_n)$
```



Likelihood Weighting

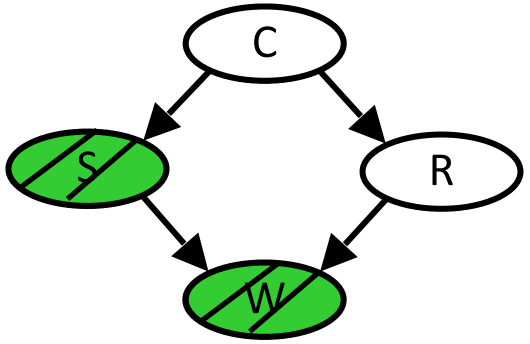
```
Input: evidence instantiation
w = 1.0
for i = 1, 2, …, n
    if $X_i$ is an evidence variable
        $X_i$ = observation $x_i$ for $X_i$
        Set $w = w * P(x_i \mid Parents(X_i))$
    else
        Sample $x_i$ from $P(X_i \mid Parents(X_i))$
return $(x_1, x_2, …, x_n)$, w
```



Likelihood Weighting

- Sampling distribution if z sampled and e fixed evidence

$$S_{WS}(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(z_i|\text{Parents}(Z_i))$$



- Now, samples have weights

$$w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{m} P(e_i|\text{Parents}(E_i))$$

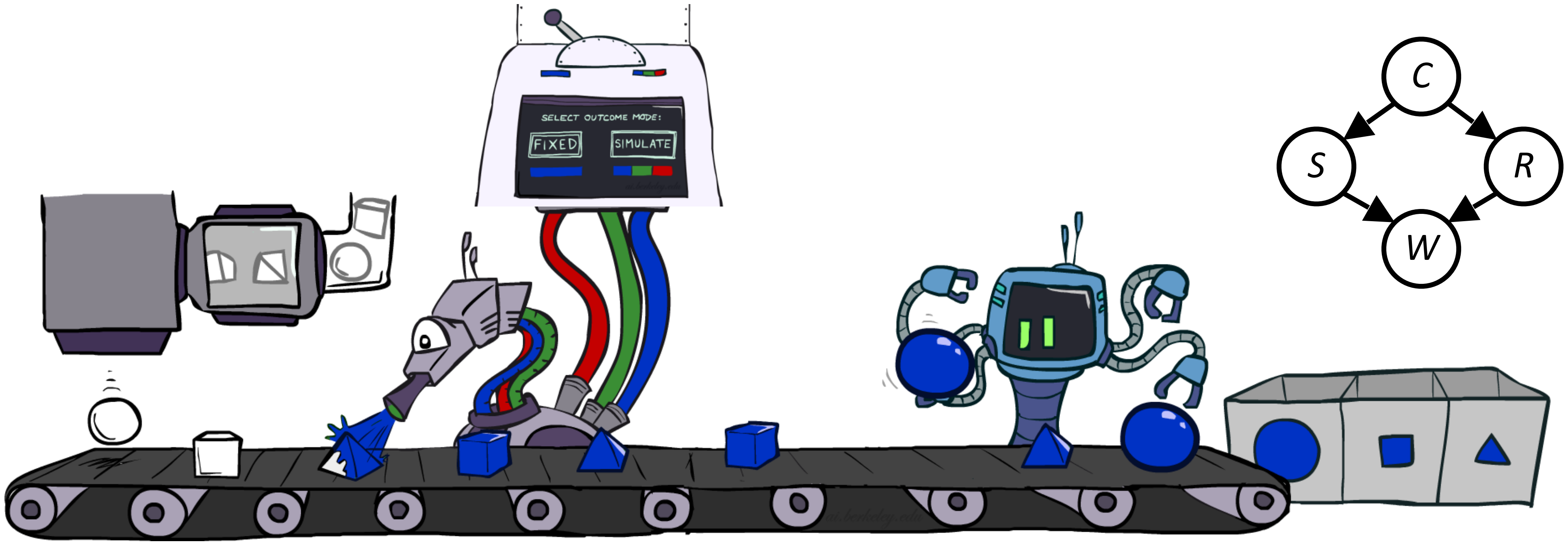
- Together, weighted sampling distribution is consistent

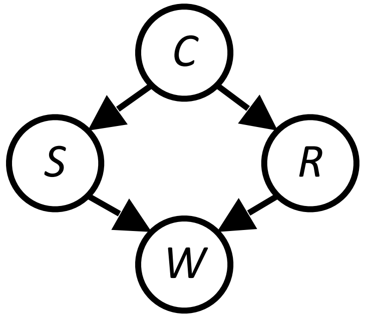
$$S_{\text{WS}}(z, e) \cdot w(z, e) = \prod_{i=1}^{l} P(z_i|\text{Parents}(z_i)) \prod_{i=1}^{m} P(e_i|\text{Parents}(e_i))$$

$$= P(\mathbf{z}, \mathbf{e})$$

Likelihood Weighting

- Likelihood weighting is good
  - We have taken evidence into account as we generate the sample
  - E.g. here, W's value will get picked based on the evidence values of S, R
  - More of our samples will reflect the state of the world suggested by the evidence
- Likelihood weighting doesn't solve all our problems
  - Evidence influences the choice of downstream variables, but not upstream ones (C isn't more likely to get a value matching the evidence)
- We would like to consider evidence when we sample every variable (leads to Gibbs sampling)




$$\begin{tikzcd} & C \arrow[dl] \arrow[dr] & \\ S \arrow[dr] & & R \arrow[dl] \\ & W & \end{tikzcd}$$

Gibbs Sampling

- *Procedure:* keep track of a full instantiation $x_1, x_2, \ldots, x_n$. Start with an arbitrary instantiation consistent with the evidence. Sample one variable at a time, conditioned on all the rest, but keep evidence fixed. Keep repeating this for a long time.

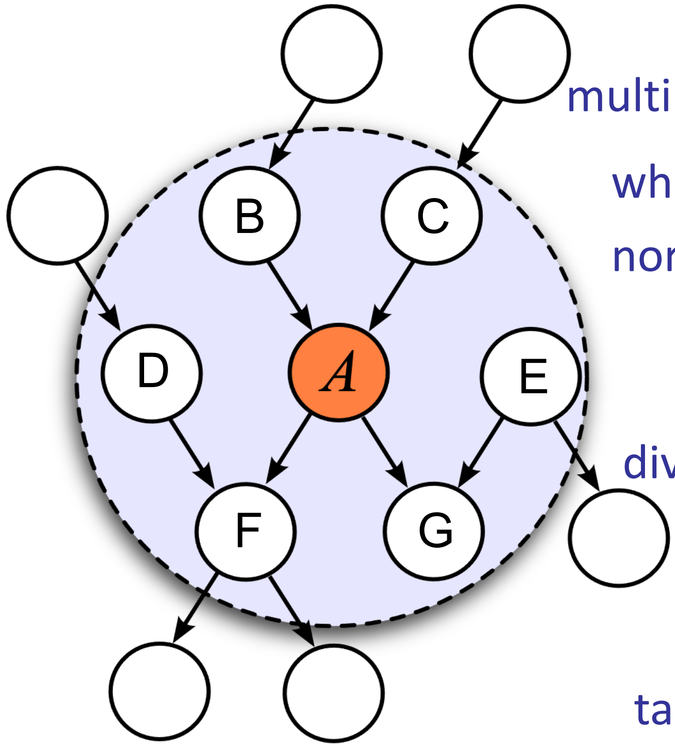
- *Property:* in the limit of repeating this infinitely many times the resulting samples come from the correct distribution (i.e. conditioned on evidence).

- *Rationale*: both upstream and downstream variables condition on evidence.

- In contrast: likelihood weighting only conditions on upstream evidence, and hence weights obtained in likelihood weighting can sometimes be very small. Sum of weights over all samples is indicative of how many "effective" samples were obtained, so we want high weight.

Gibbs Sampling: Node probability

- To sample, we first compute a factor that includes all node CPTs that depend on A.
- Then we normalize it to get a conditional probability for A.
- Finally we sample to get a new value for A.



multiply CPTs with A: $p(A \mid b, c)\, p\left(f \mid A, d\right)\, p(g \mid A, e)$

which is the factor: $p(A, f, g \mid b, c, d, e)$

normalize (sum over A and divide):

$$Z = \sum_a P(a, f, g \mid b, c, d, e) = P(f, g \mid b, c, d, e)$$

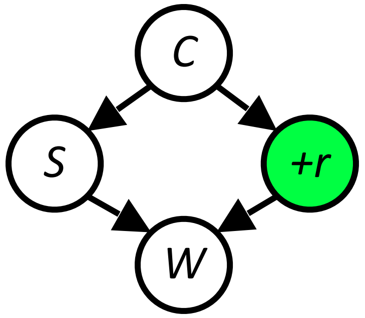
divide:

$$p\left(A \mid b, c, d, e, f, g\right) = p\left(A, f, g \mid b, c, d, e\right) / Z$$

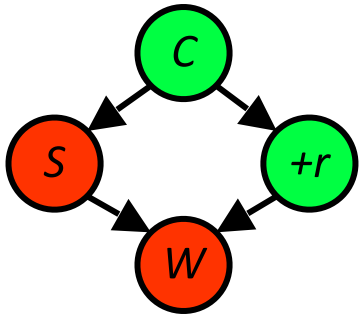
take a sample of A from this distribution.

Gibbs Sampling Example: P(S | +r)

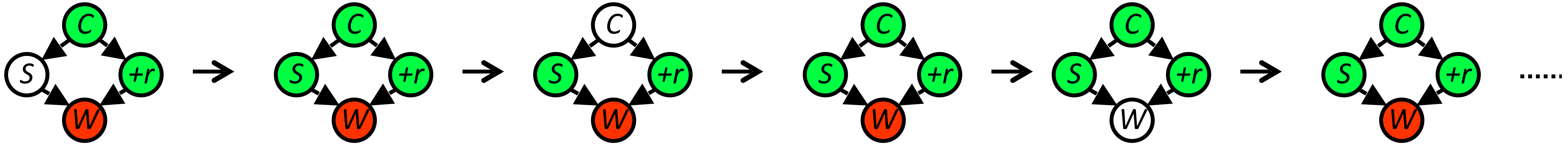
- Step 1: Fix evidence
  - R = +r



- Step 2: Initialize other variables
  - Randomly



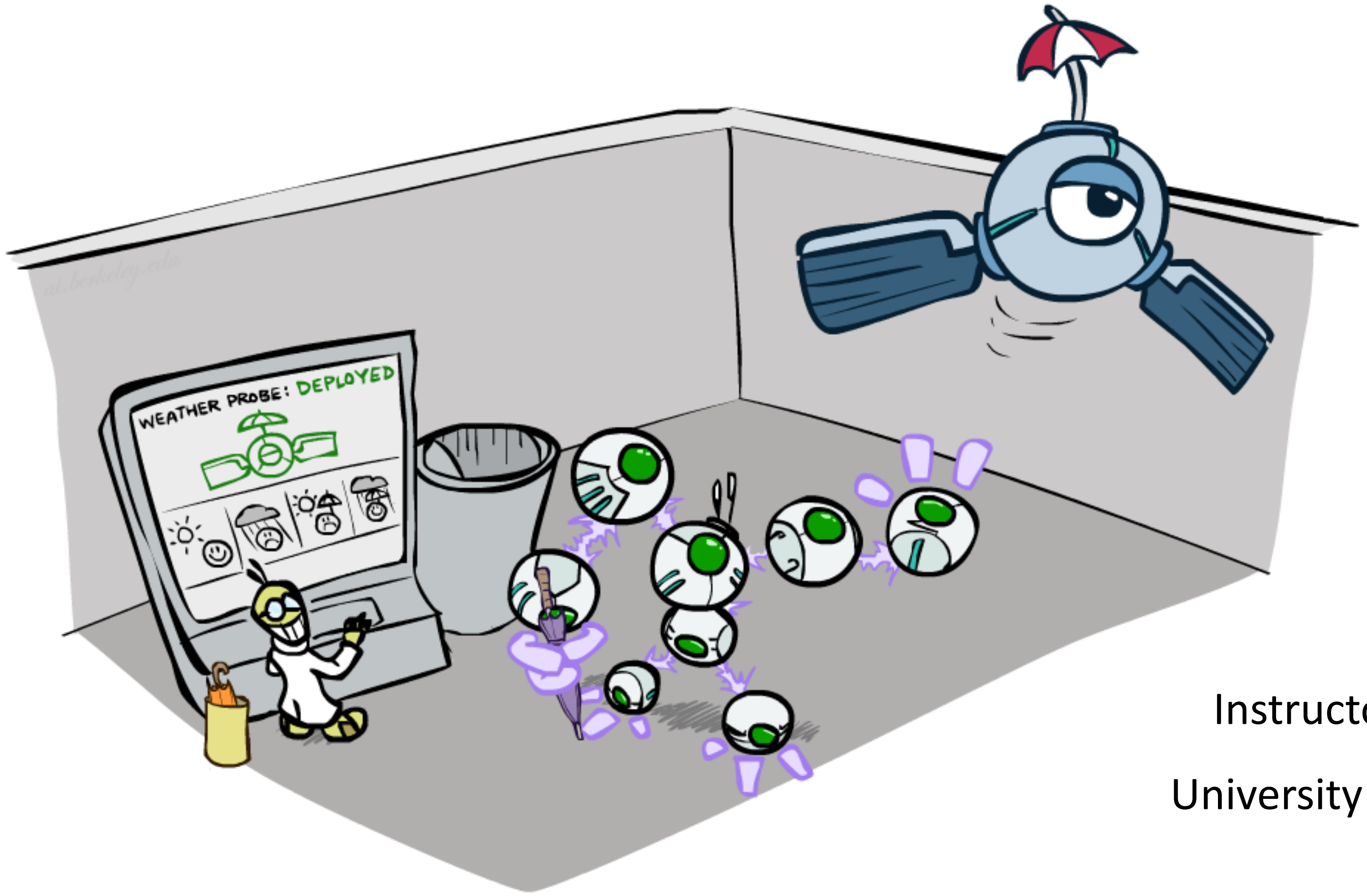
- Steps 3: Repeat
  - Choose a non-evidence variable X
  - Resample X from P(X | all other variables)



Sample from $P(S| + c, -w, +r)$ Sample from $P(C| + s, -w, +r)$ Sample from $P(W| + s, +c, +r)$

CS 188: Artificial Intelligence

Decision Networks and Value of Information

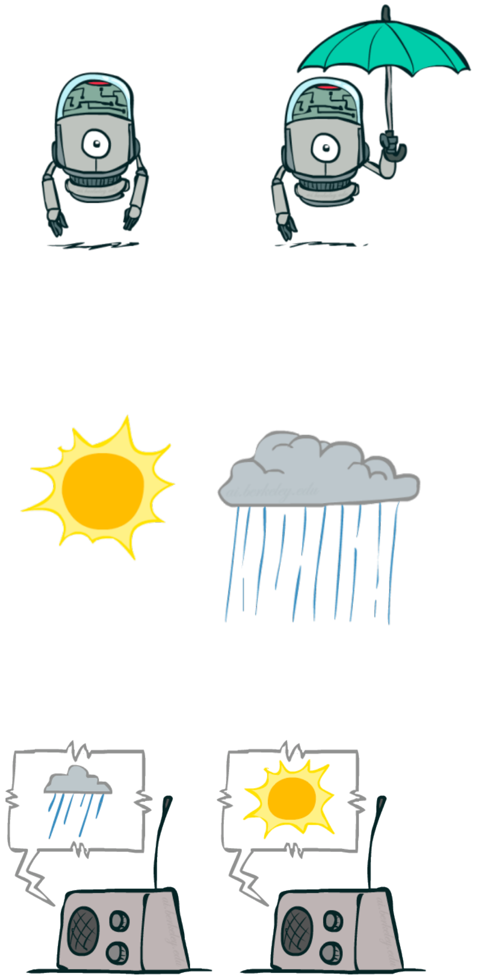


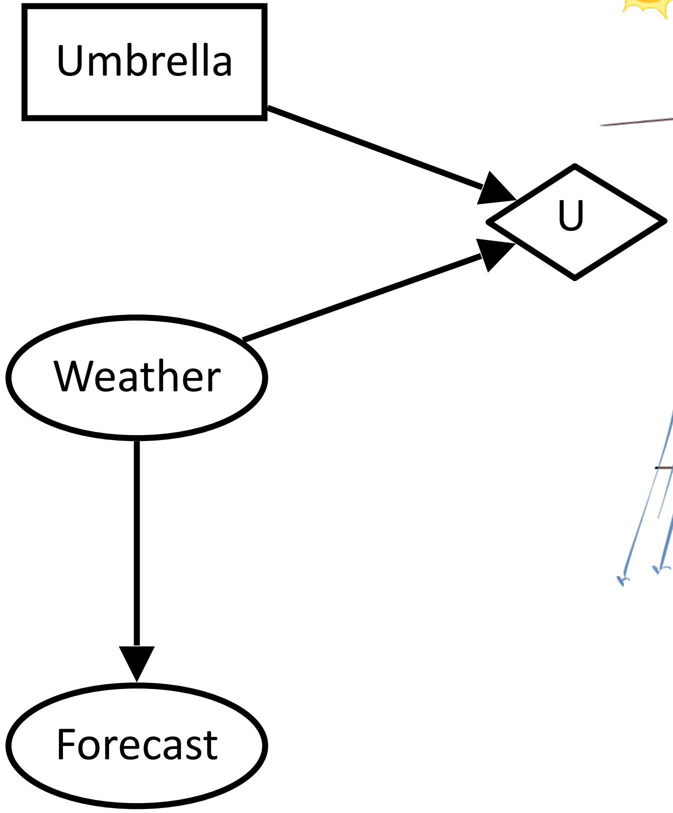
Instructor: Oliver Grillmeyer

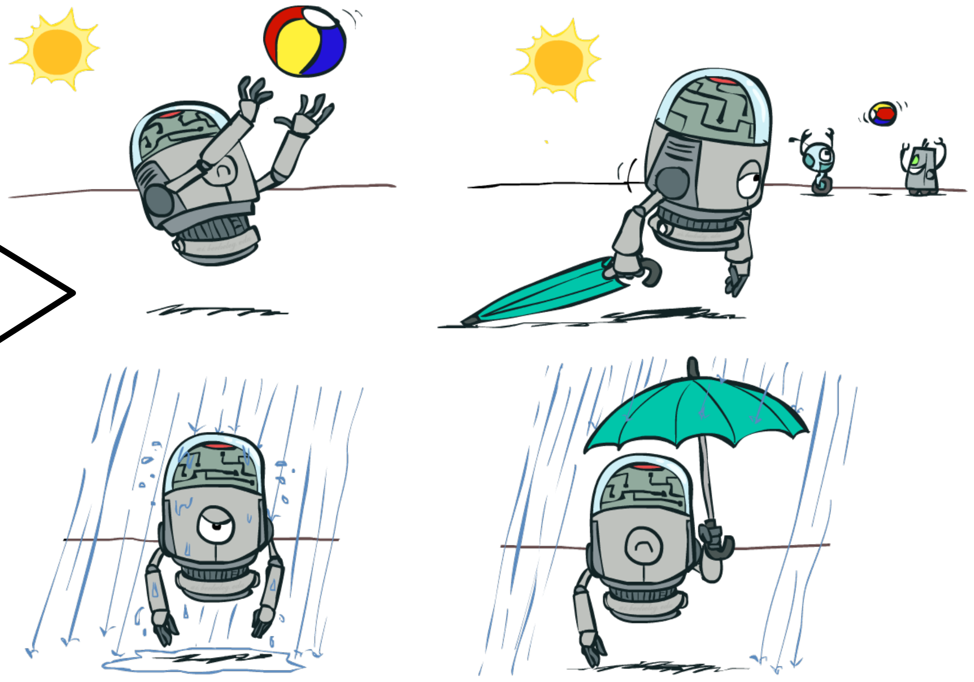
University of California, Berkeley

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Decision Networks



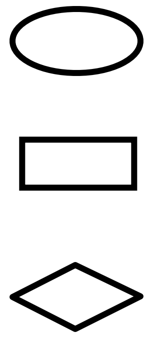


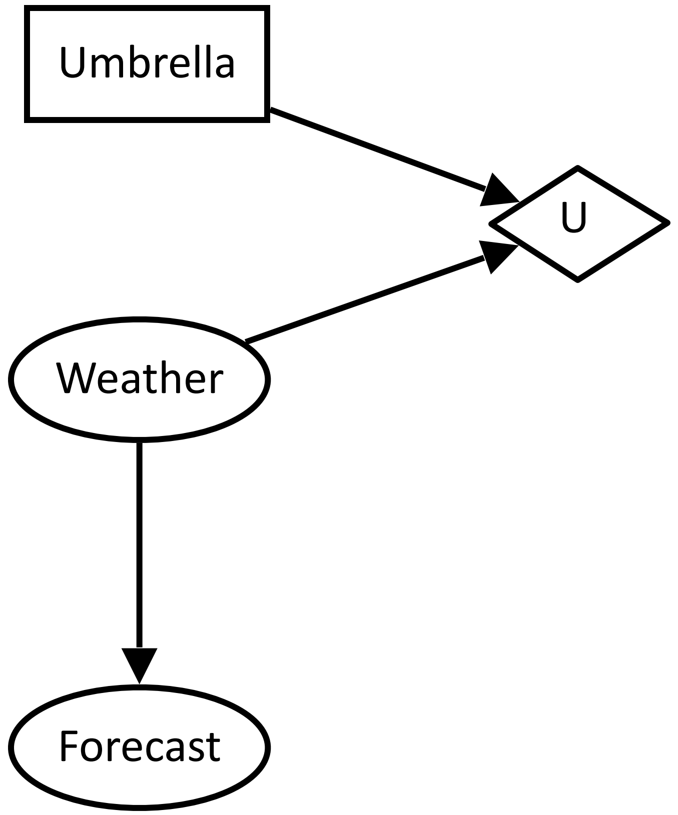


Decision Networks

- **MEU: choose the action which maximizes the expected utility given the evidence**

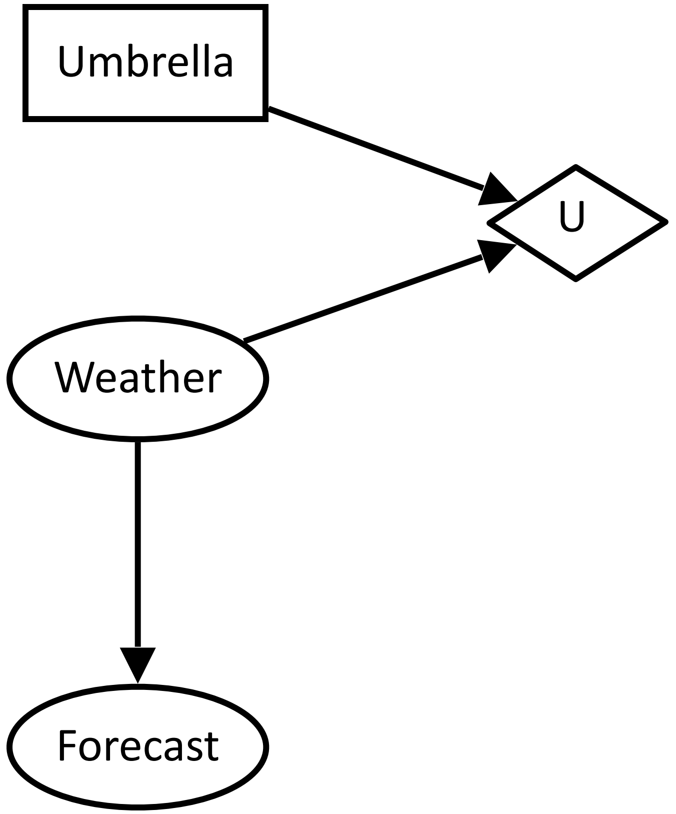
- Can directly operationalize this with decision networks
  - Bayes nets with nodes for utility and actions
  - Lets us calculate the expected utility for each action
- New node types:
  - Chance nodes (just like BNs)
  - Actions (rectangles, cannot have parents, act as observed evidence)
  - Utility node (diamond, depends on action and chance nodes)





Decision Networks

- Action selection
  - Instantiate all evidence
  - Set action node(s) each possible way
  - Calculate posterior for all parents of utility node, given the evidence
  - Calculate expected utility for each action
  - Choose maximizing action



Decision Networks

Umbrella = leave

$$\mathrm{EU}(\mathrm{leave}) = \sum_w P(w)U(\mathrm{leave}, w)$$

$$= 0.7 \cdot 100 + 0.3 \cdot 0 = 70$$

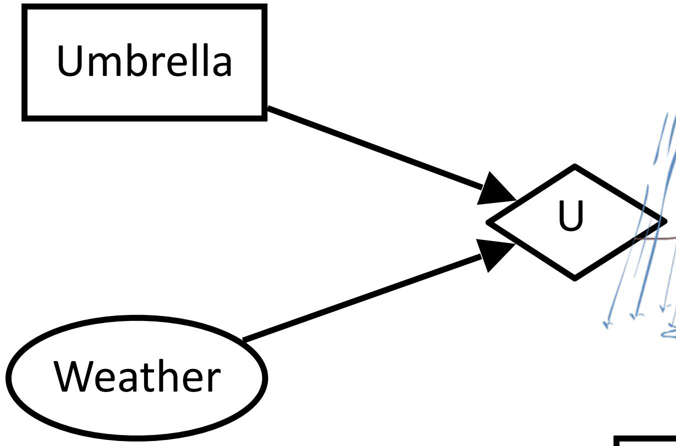
Umbrella = take

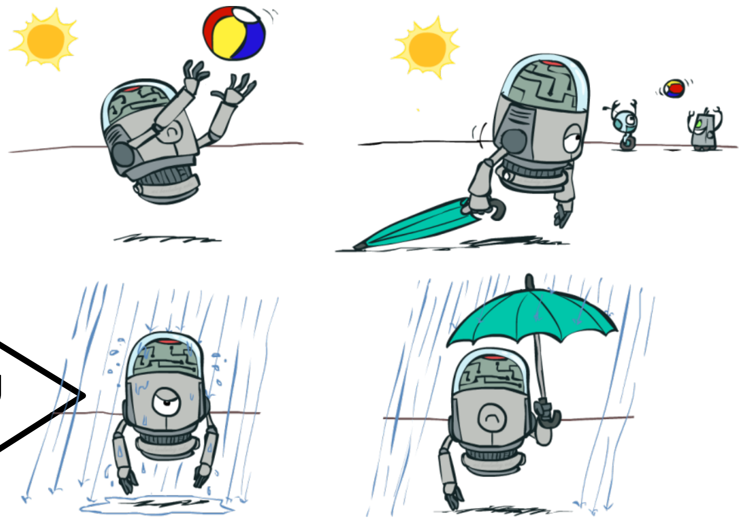
$$\mathrm{EU}(\mathrm{take}) = \sum_w P(w)U(\mathrm{take}, w)$$

$$= 0.7 \cdot 20 + 0.3 \cdot 70 = 35$$

Optimal decision = leave

$$\mathrm{MEU}(\emptyset) = \max_a \mathrm{EU}(a) = 70$$





W	P(W)
sun	0.7
rain	0.3

A	W	U(A,W)
leave	sun	100
leave	rain	0
take	sun	20
take	rain	70

Example: Decision Networks

Umbrella = leave

$$\mathrm{EU}(\mathrm{leave}|\mathrm{bad}) = \sum_w P(w|\mathrm{bad})U(\mathrm{leave}, w)$$

$$= 0.34 \cdot 100 + 0.66 \cdot 0 = 34$$

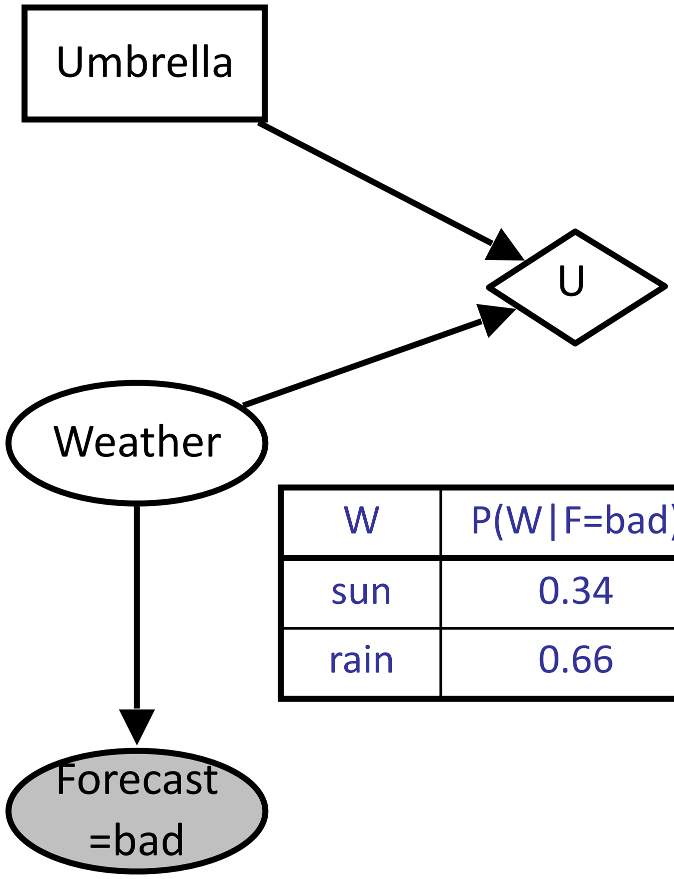
Umbrella = take

$$\mathrm{EU}(\mathrm{take}|\mathrm{bad}) = \sum_w P(w|\mathrm{bad})U(\mathrm{take}, w)$$

$$= 0.34 \cdot 20 + 0.66 \cdot 70 = 53$$

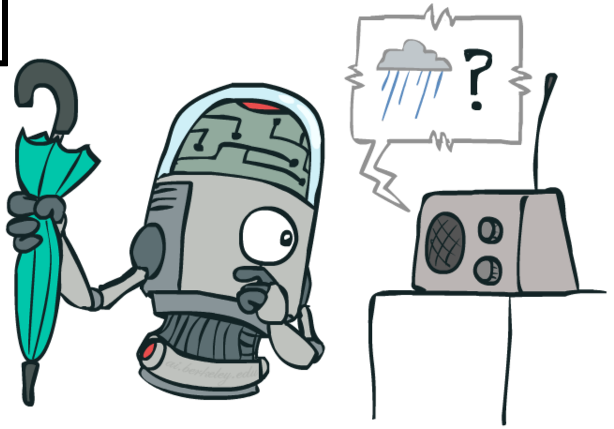
Optimal decision = take

$$\mathrm{MEU}(F = \mathrm{bad}) = \max_a \mathrm{EU}(a|\mathrm{bad}) = 53$$



A	W	U(A,W)
leave	sun	100
leave	rain	0
take	sun	20
take	rain	70

W	P(W|F=bad)
sun	0.34
rain	0.66



VPI Example: Weather

MEU with no evidence

$$\mathrm{MEU}(\emptyset) = \max_a \mathrm{EU}(a) = 70$$

MEU if forecast is bad

$$\mathrm{MEU}(F = \mathrm{bad}) = \max_a \mathrm{EU}(a|\mathrm{bad}) = 53$$

MEU if forecast is good

$$\mathrm{MEU}(F = \mathrm{good}) = \max_a \mathrm{EU}(a|\mathrm{good}) = 95$$

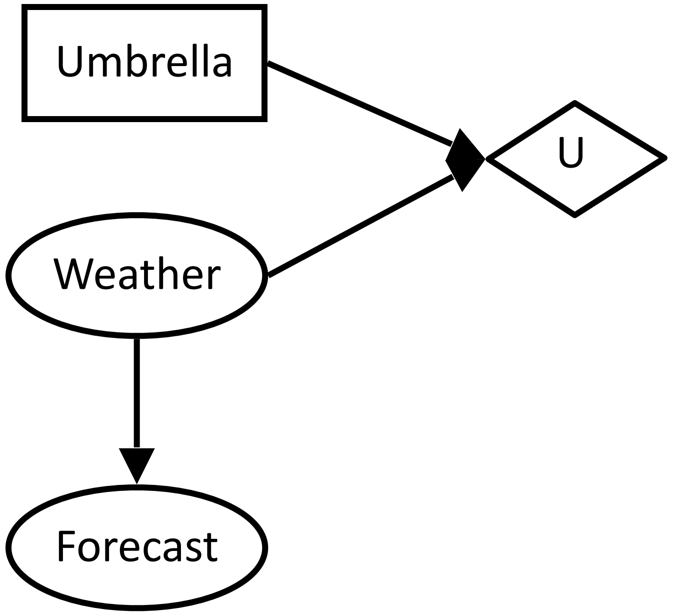
Forecast distribution

F	P(F)
good	0.59
bad	0.41

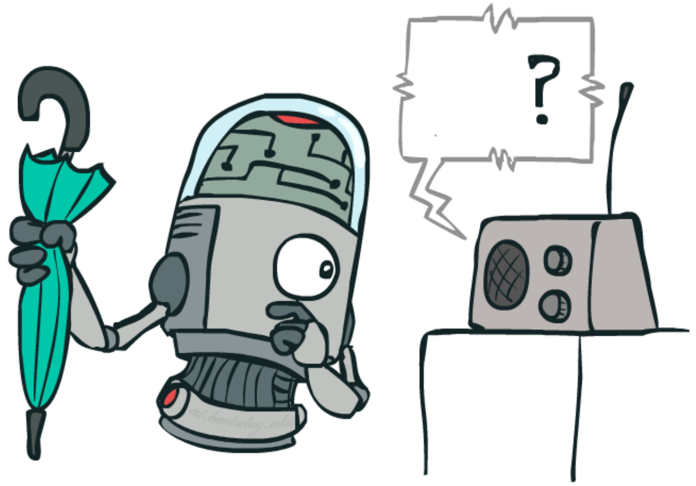
$$\Rightarrow \quad 0.59 \cdot (95) + 0.41 \cdot (53) - 70$$

$$77.8 - 70 = 7.8$$

$$\mathrm{VPI}(E'|e) = \left(\sum_{e'} P(e'|e)\mathrm{MEU}(e, e')\right) - \mathrm{MEU}(e)$$



A	W	U
leave	sun	100
leave	rain	0
take	sun	20
take	rain	70



VPI Properties

- Nonnegative

$$\forall E', e : \text{VPI}(E'|e) \geq 0$$

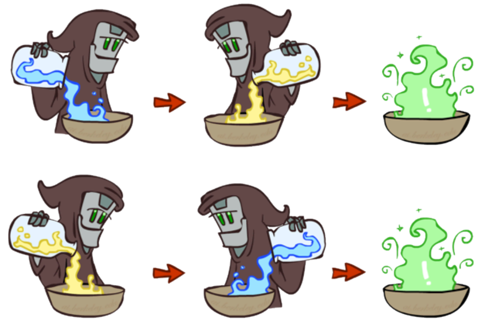
- Nonadditive

  (think of observing E_j twice)

$$\text{VPI}(E_j, E_k|e) \neq \text{VPI}(E_j|e) + \text{VPI}(E_k|e)$$

- Order-independent

$$\begin{aligned}\text{VPI}(E_j, E_k|e) &= \text{VPI}(E_j|e) + \text{VPI}(E_k|e, E_j) \\ &= \text{VPI}(E_k|e) + \text{VPI}(E_j|e, E_k)\end{aligned}$$



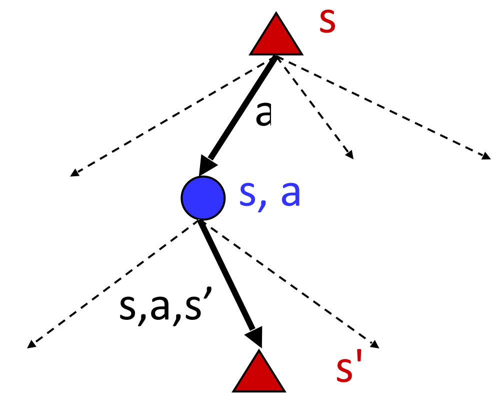
POMDPs

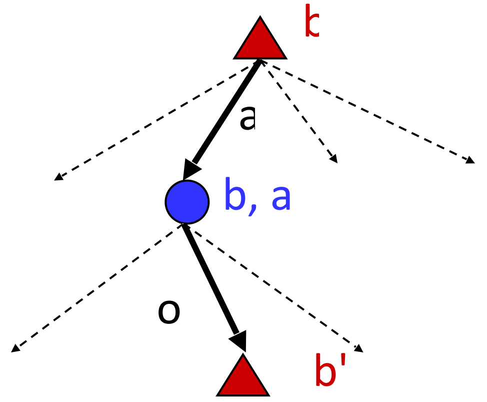
- MDPs have:
  - States S
  - Actions A
  - Transition function P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s')

- POMDPs add:
  - Observations O
  - Observation function P(o|s) (or O(s,o))

- POMDPs are MDPs over belief states b (distributions over S)

- We'll be able to say more in a few lectures





CS 188: Artificial Intelligence

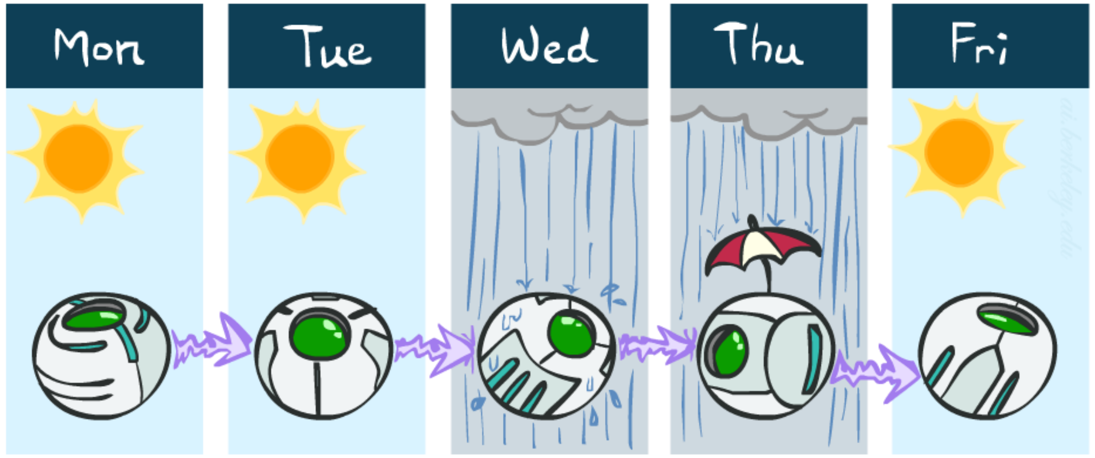
Hidden Markov Models

Instructors: Oliver Grillmeyer, Kaylo Littlejohn — University of California, Berkeley

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Example Markov Chain: Weather

- States: X = {rain, sun}

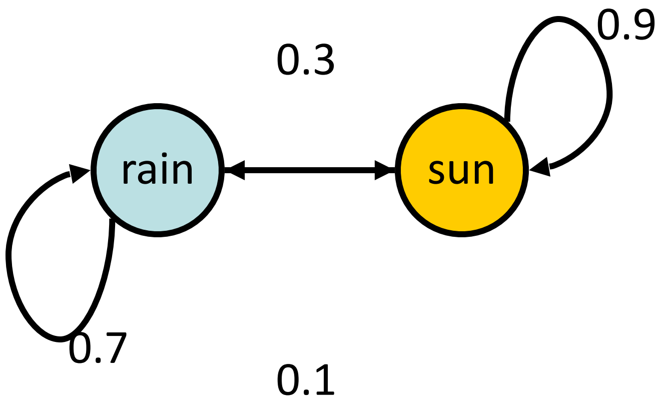


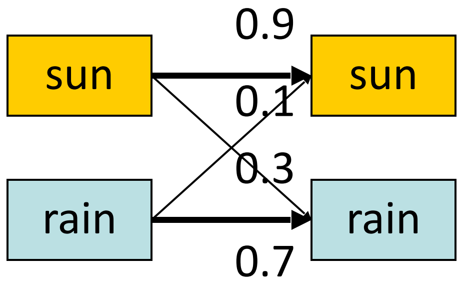
- Initial distribution: 1.0 sun

- CPT $P(X_t \mid X_{t-1})$:

X_{t-1}	X_t	$P(X_t \mid X_{t-1})$
sun	sun	0.9
sun	rain	0.1
rain	sun	0.3
rain	rain	0.7

Two new ways of representing the same CPT





Example Markov Chain: Weather

- Initial distribution: 1.0 sun
  - We know: $P(X_1) \quad P(X_t|X_{t-1})$

X_{t-1}	X_t	$P(X_t\|X_{t-1})$
sun	sun	0.9
sun	rain	0.1
rain	sun	0.3
rain	rain	0.7

- What is the probability distribution after one step?

$$P(X_2 = sun) = \sum_{x_1} P(x_1, X_2 = sun) = \sum_{x_1} P(X_2 = sun|x_1)P(x_1)$$

$$= \quad P(X_2 = \text{sun}|X_1 = \text{sun})P(X_1 = \text{sun}) + P(X_2 = \text{sun}|X_1 = \text{rain})P(X_1 = \text{rain})$$

$$0.9 \cdot 1.0 + 0.3 \cdot 0.0 = 0.9$$

Example Run of Mini-Forward Algorithm

- From initial observation of sun

$$\underset{P(X_1)}{\left\langle \begin{matrix} 1.0 \\ 0.0 \end{matrix} \right\rangle} \quad \underset{P(X_2)}{\left\langle \begin{matrix} 0.9 \\ 0.1 \end{matrix} \right\rangle} \quad \underset{P(X_3)}{\left\langle \begin{matrix} 0.84 \\ 0.16 \end{matrix} \right\rangle} \quad \underset{P(X_4)}{\left\langle \begin{matrix} 0.804 \\ 0.196 \end{matrix} \right\rangle} \Rightarrow \underset{P(X_\infty)}{\left\langle \begin{matrix} 0.75 \\ 0.25 \end{matrix} \right\rangle}$$

- From initial observation of rain

$$\underset{P(X_1)}{\left\langle \begin{matrix} 0.0 \\ 1.0 \end{matrix} \right\rangle} \quad \underset{P(X_2)}{\left\langle \begin{matrix} 0.3 \\ 0.7 \end{matrix} \right\rangle} \quad \underset{P(X_3)}{\left\langle \begin{matrix} 0.48 \\ 0.52 \end{matrix} \right\rangle} \quad \underset{P(X_4)}{\left\langle \begin{matrix} 0.588 \\ 0.412 \end{matrix} \right\rangle} \Rightarrow \underset{P(X_\infty)}{\left\langle \begin{matrix} 0.75 \\ 0.25 \end{matrix} \right\rangle}$$

- From yet another initial distribution $P(X_1)$:

$$\underset{P(X_1)}{\left\langle \begin{matrix} p \\ 1-p \end{matrix} \right\rangle} \quad \ldots \quad \Rightarrow \underset{P(X_\infty)}{\left\langle \begin{matrix} 0.75 \\ 0.25 \end{matrix} \right\rangle}$$

[Demo: L13D1,2,3]

Stationary Distributions

- For most chains:
  - Influence of the initial distribution gets less and less over time.
  - The distribution we end up in is independent of the initial distribution

- Stationary distribution:
  - The distribution we end up with is called the stationary distribution P_∞ of the chain
  - It satisfies

$$P_\infty(X) = P_{\infty+1}(X) = \sum_x P(X|x)P_\infty(x)$$

Example: Stationary Distributions

- Question: What's P(X) at time t = infinity?

$$\begin{tikzcd} X_1 \arrow[r] & X_2 \arrow[r] & X_3 \arrow[r] & X_4 \arrow[r, dashed] & \; & X_\infty \end{tikzcd}$$?

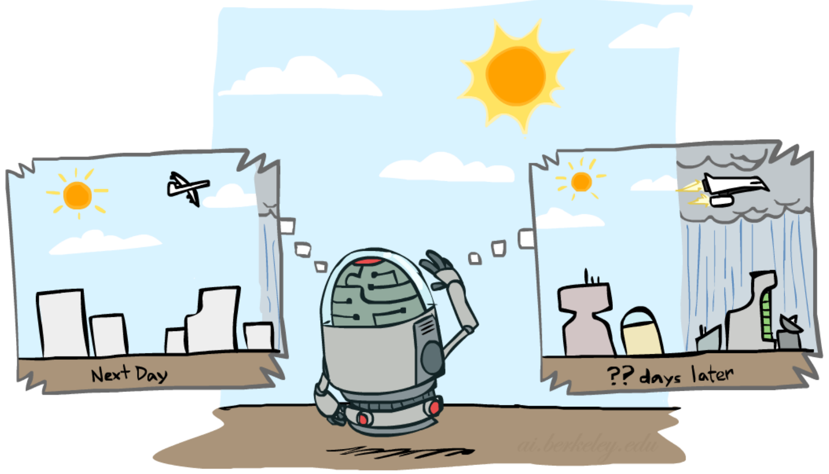
$$P_\infty(X) = P_{\infty+1}(X) = \sum_x P(X|x)P_\infty(x)$$

$$P_\infty(sun) = P(sun|sun)P_\infty(sun) + P(sun|rain)P_\infty(rain)$$
$$P_\infty(rain) = P(rain|sun)P_\infty(sun) + P(rain|rain)P_\infty(rain)$$

$$P_\infty(sun) = 0.9P_\infty(sun) + 0.3P_\infty(rain)$$
$$P_\infty(rain) = 0.1P_\infty(sun) + 0.7P_\infty(rain)$$

$$P_\infty(sun) = 3P_\infty(rain)$$

Also: $P_\infty(sun) + P_\infty(rain) = 1$ $\Rightarrow$ $P_\infty(sun) = 3/4$, $P_\infty(rain) = 1/4$



X_{t-1}	X_t	$P(X_t\|X_{t-1})$
sun	sun	0.9
sun	rain	0.1
rain	sun	0.3
rain	rain	0.7

- Alternatively: run simulation for a long (ideally infinite) time

Hidden Markov Models

- Markov chains OK for games, weak for real robots

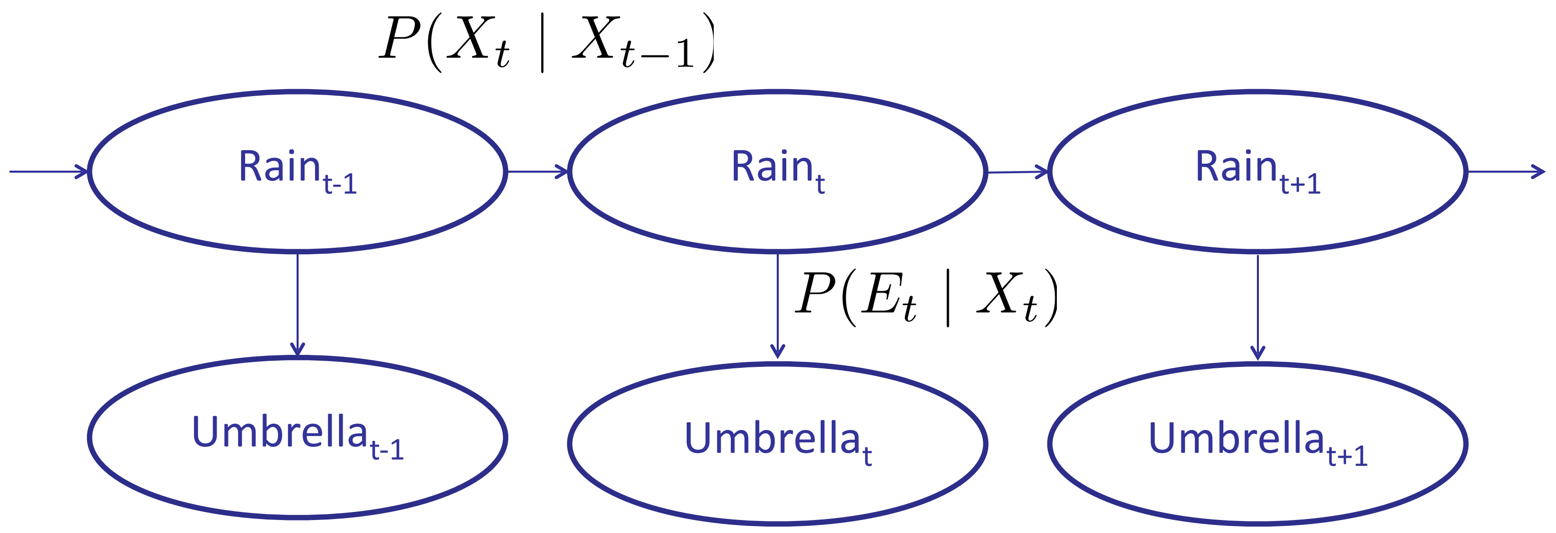
$$\begin{tikzcd} X_1 \arrow[r] & X_2 \arrow[r] & X_3 \arrow[r] & X_4 \arrow[r, dashed] & {} \end{tikzcd}$$

  - Need observations to update your beliefs

- Hidden Markov models (HMMs)
  - Underlying Markov chain over states X
  - You observe outputs (effects) at each time step

$$\begin{tikzcd} X_1 \arrow[r] \arrow[d] & X_2 \arrow[r] \arrow[d] & X_3 \arrow[r] \arrow[d] & X_4 \arrow[r, dashed] \arrow[d] & {} \\ E_1 & E_2 & E_3 & E_4 & \end{tikzcd}$$

Example: Weather HMM



- An HMM is defined by:
  - Initial distribution: $P(X_1)$
  - Transitions: $P(X_t \mid X_{t-1})$
  - Emissions: $P(E_t \mid X_t)$

Transitions

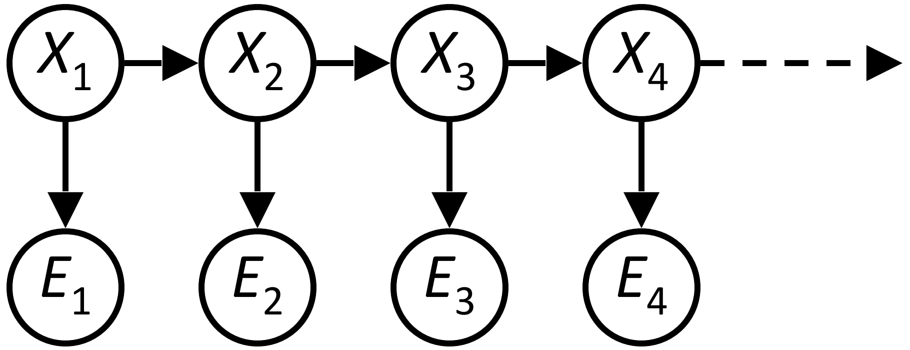
R_{t-1}	R_t	$P(R_t \mid R_{t-1})$
+r	+r	0.7
+r	-r	0.3
-r	+r	0.3
-r	-r	0.7

Emissions

R_t	U_t	$P(U_t \mid R_t)$
+r	+u	0.9
+r	-u	0.1
-r	+u	0.2
-r	-u	0.8

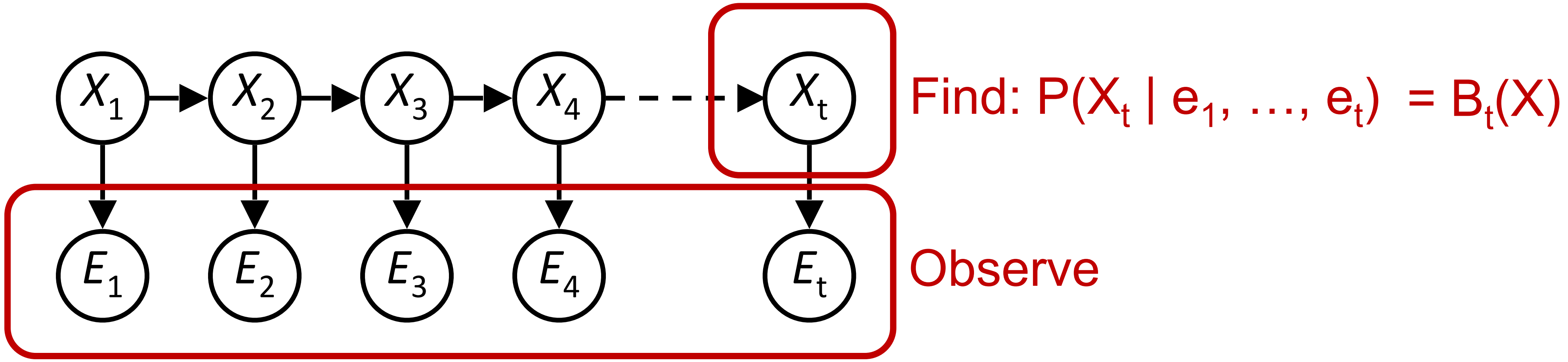
Conditional Independence

- HMMs have two important independence properties:
  - Markov hidden process: future depends on past via the present
  - Current observation independent of all else given current state



- Does this mean that evidence variables are guaranteed to be independent?
  - No, they are correlated by the hidden state

Inference in HMMs: Filtering

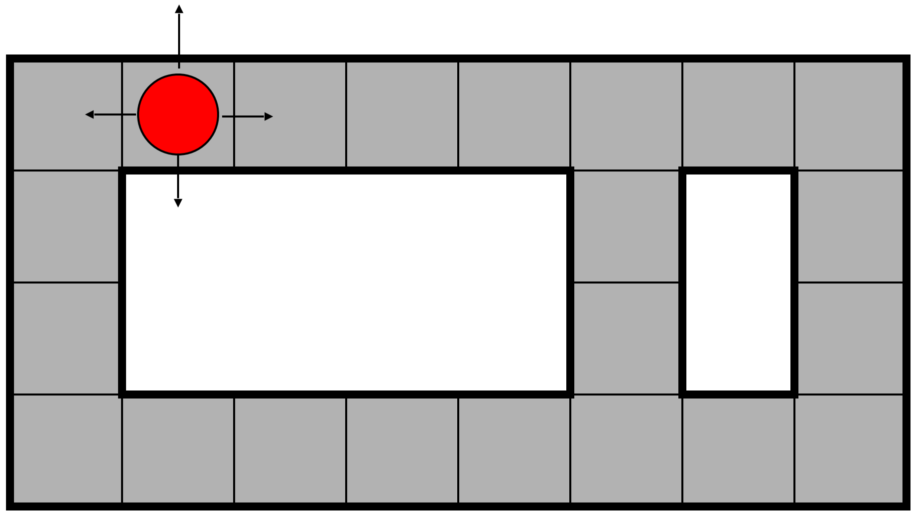


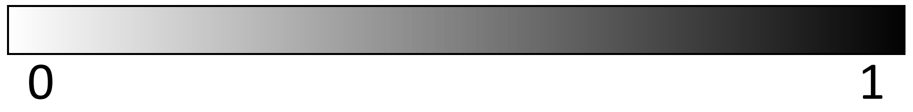
- Filtering, or monitoring, is the task of tracking the distribution $B_t(X) = P_t(X_t \mid e_1, \ldots, e_t)$ (the belief state) over time
- We start with $B_1(X)$ in an initial setting, usually uniform
- As time passes, or we get observations, we update $B(X)$

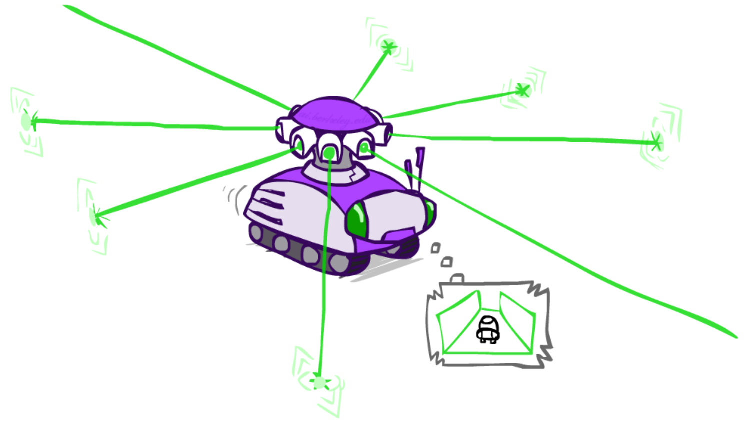
- The Kalman filter was invented in the 60's and first implemented as a method of trajectory estimation for the Apollo program

Example: Robot Localization

Example from
Michael Pfeiffer





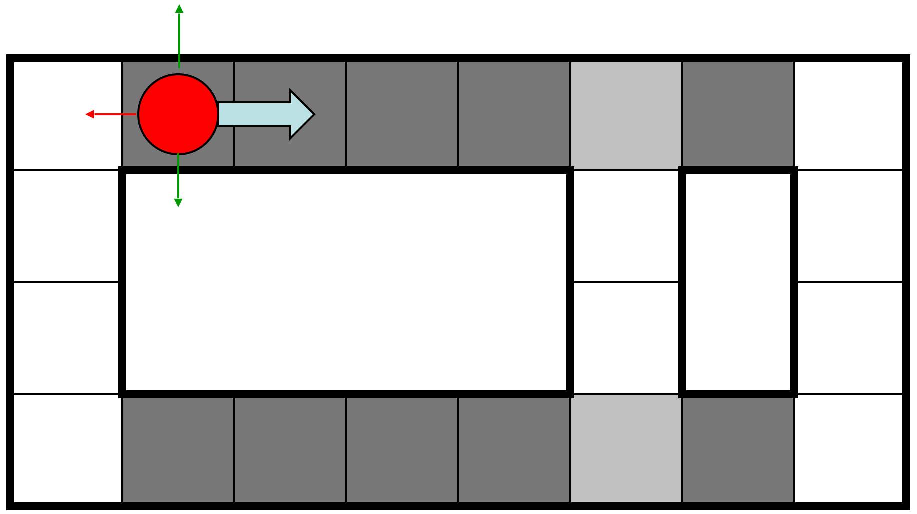


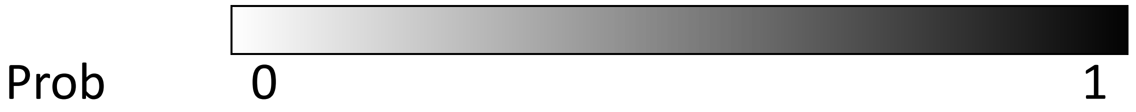
t=0

Sensor model: can read in which directions there is a wall, never more than 1 mistake

Motion model: may not execute action with small prob.

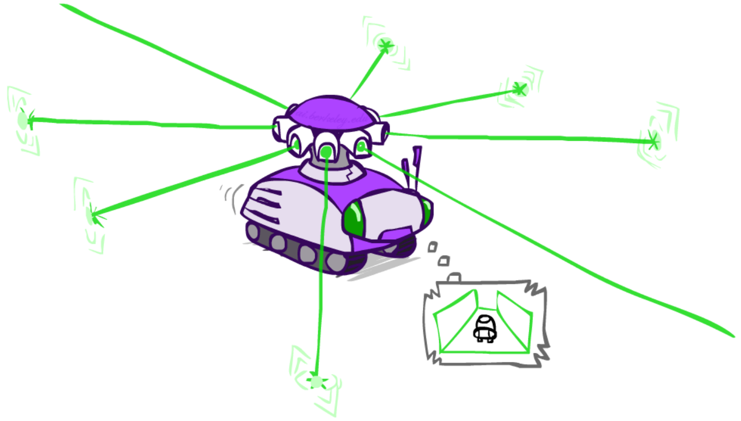
Example: Robot Localization



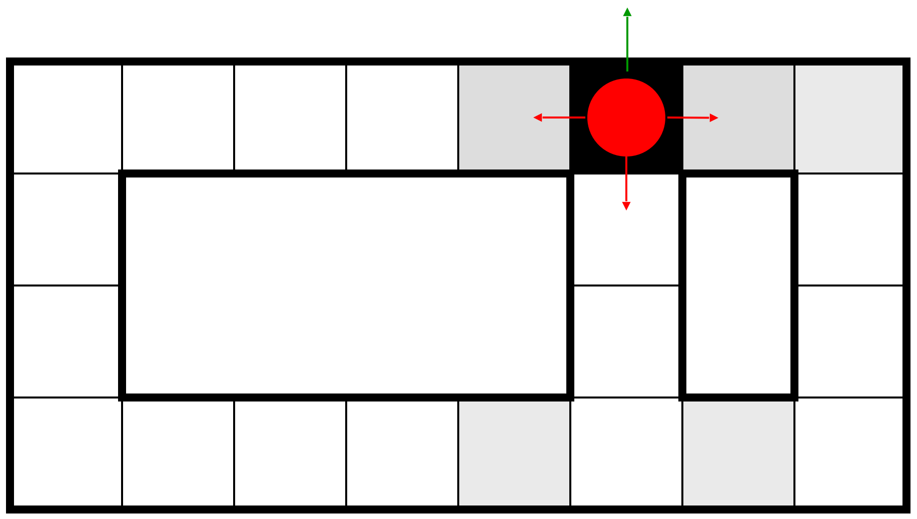


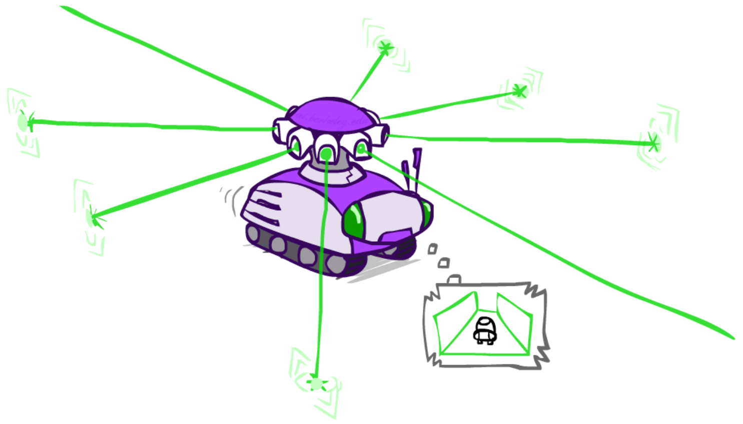
t=1

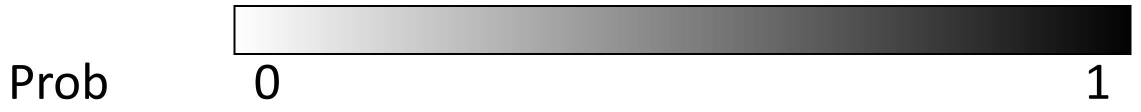
Lighter grey: was possible to get the reading, but less likely b/c required 1 mistake



Example: Robot Localization







t=5

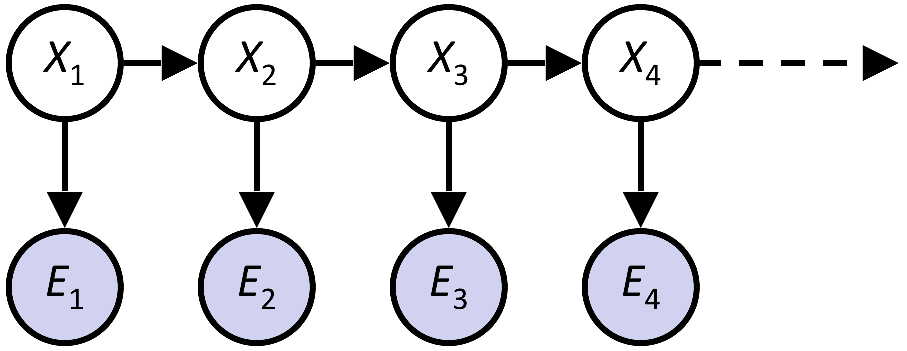
HMM Inference: Find State Given Evidence

- We are given evidence at each time and want to know

$$B_t(X) = P(X_t \mid e_{1:t})$$

- Idea: start with $P(X1)$ and derive $B_t(X)$ in terms of $B_{t-1}(X)$
  - Two steps: **Passage of Time** & **Observation**

$B'_4(X) = P(X_4 \mid e_{1:3})$



$B_3(X)$ $\quad$ $B_4(X) = P(X_4 \mid e_{1:4})$

Observation: General Case

- Assume we have current belief P(X | previous evidence):

$$B'(X_{t+1}) = P(X_{t+1}|e_{1:t})$$

- Then, after evidence comes in:

$$P(X_{t+1}|e_{1:t+1}) = P(X_{t+1}, e_{t+1}|e_{1:t})/P(e_{t+1}|e_{1:t})$$

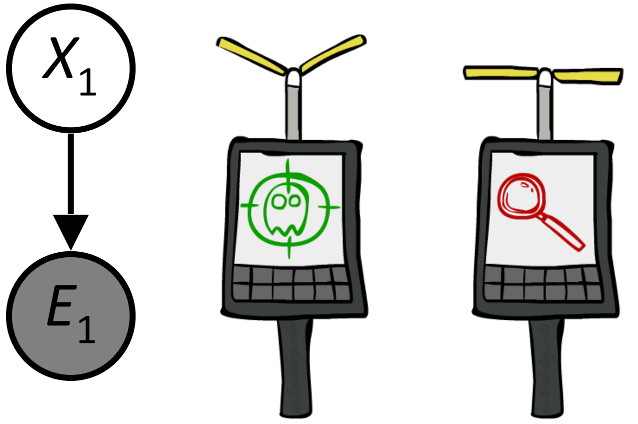
$$\propto_{X_{t+1}} P(X_{t+1}, e_{t+1}|e_{1:t})$$

$$= P(e_{t+1}|e_{1:t}, X_{t+1})P(X_{t+1}|e_{1:t})$$

$$= P(e_{t+1}|X_{t+1})P(X_{t+1}|e_{1:t})$$

- Or, compactly:

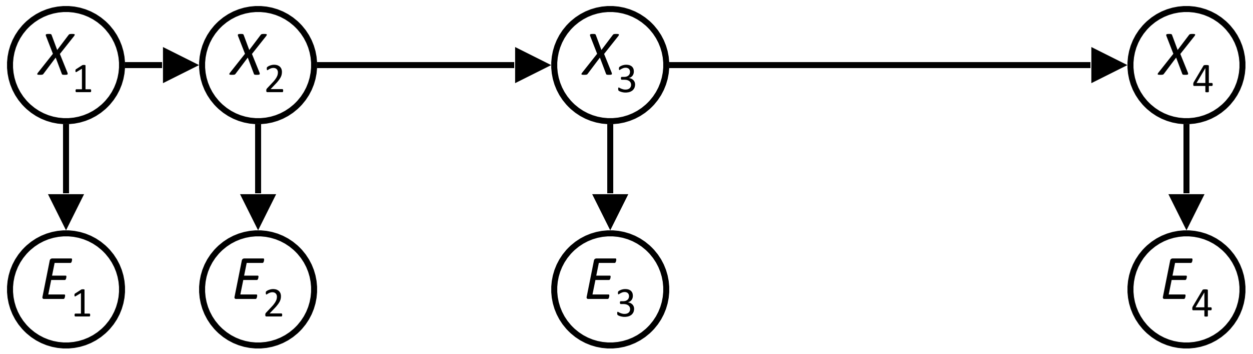
$$B(X_{t+1}) \propto_{X_{t+1}} P(e_{t+1}|X_{t+1})B'(X_{t+1})$$



- Basic idea: beliefs “reweighted” by likelihood of evidence
- Unlike passage of time, we have to renormalize

Two Steps: Passage of Time + Observation

$$B(X_t) = P(X_t|e_{1:t})$$



$$B'(X_{t+1}) = \sum_{x_t} P(X'|x_t)B(x_t)$$

$$B(X_{t+1}) \propto_{X_{t+1}} P(e_{t+1}|X_{t+1})B'(X_{t+1})$$

Example: Weather HMM

B(+r) = 0.5
B(-r) = 0.5

B'(+r) = 0.5
B'(-r) = 0.5

B(+r) = 0.818
B(-r) = 0.182

B'(+r) = 0.627
B'(-r) = 0.373

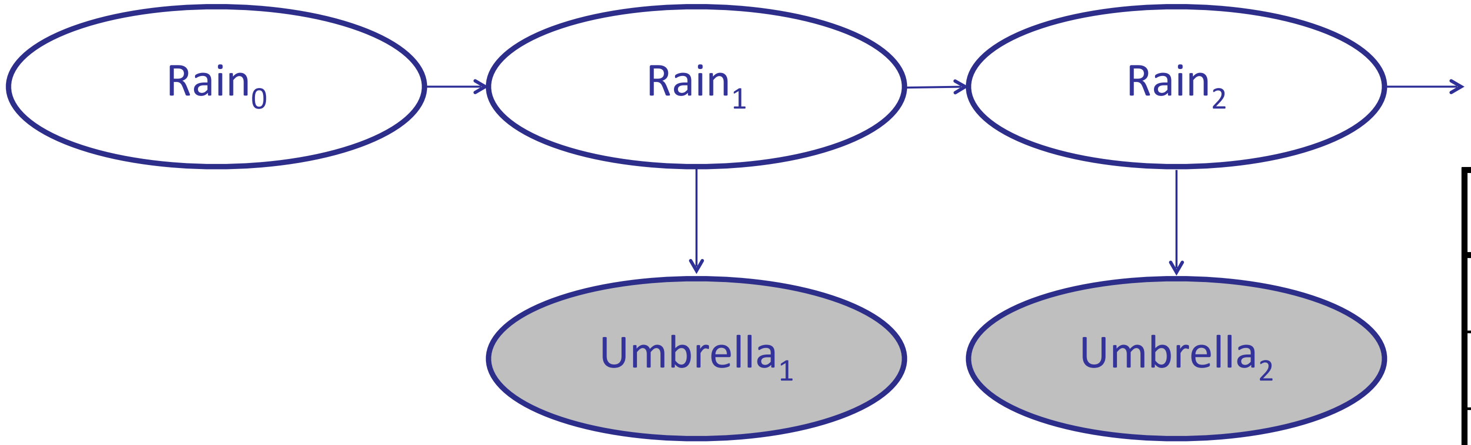
B(+r) = 0.883
B(-r) = 0.117

Passage of Time:

$$B'(X_{t+1}) = \sum_{x_t} P(X_{t+1}|x_t)B(x_t)$$

Observation:

$$B(X_{t+1}) \propto_{X_{t+1}} P(e_{t+1}|X_{t+1})B'(X_{t+1})$$



$P(X_{t+1}|X_t)$

R$_t$	R$_{t+1}$	P(R$_{t+1}$|R$_t$)
+r	+r	0.7
+r	-r	0.3
-r	+r	0.3
-r	-r	0.7

$P(E_t|X_t)$

R$_t$	U$_t$	P(U$_t$|R$_t$)
+r	+u	0.9
+r	-u	0.1
-r	+u	0.2
-r	-u	0.8

CS 188: Artificial Intelligence

HMMs, Particle Filters, and Applications

Instructors: Oliver Grillmeyer, Cheol Jun Cho

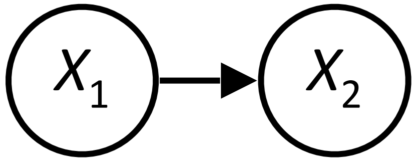
University of California, Berkeley

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Passage of Time

- Assume we have current belief P(X | evidence to date)

$$B(X_t) = P(X_t|e_{1:t})$$



- Then, after one time step passes:

$$P(X_{t+1}|e_{1:t}) = \sum_{x_t} P(X_{t+1}, x_t|e_{1:t})$$

$$= \sum_{x_t} P(X_{t+1}|x_t, e_{1:t})P(x_t|e_{1:t})$$

$$= \sum_{x_t} P(X_{t+1}|x_t)P(x_t|e_{1:t})$$

- Or compactly:

$$B'(X_{t+1}) = \sum_{x_t} P(X_{t+1}|x_t)B(x_t)$$

- Basic idea: beliefs get "pushed" through the transitions
  - With the "B" notation, we have to be careful about what time step t the belief is about, and what evidence it includes

Observation

- Assume we have current belief P(X | previous evidence):

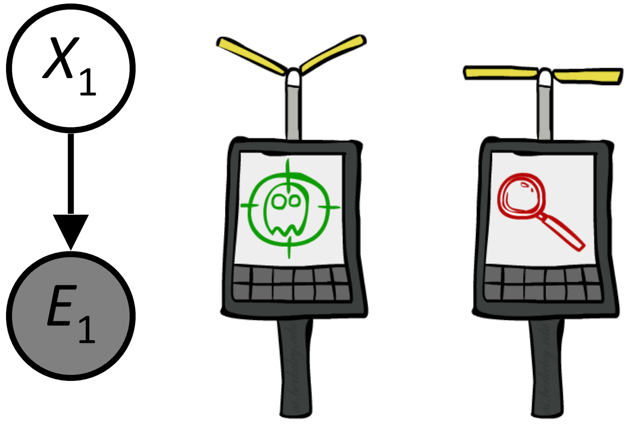
$$B'(X_{t+1}) = P(X_{t+1}|e_{1:t})$$

- Then, after evidence comes in:

$$P(X_{t+1}|e_{1:t+1}) = P(X_{t+1}, e_{t+1}|e_{1:t})/P(e_{t+1}|e_{1:t})$$
$$\propto_{X_{t+1}} P(X_{t+1}, e_{t+1}|e_{1:t})$$
$$= P(e_{t+1}|e_{1:t}, X_{t+1})P(X_{t+1}|e_{1:t})$$
$$= P(e_{t+1}|X_{t+1})P(X_{t+1}|e_{1:t})$$

- Or, compactly:

$$B(X_{t+1}) \propto_{X_{t+1}} P(e_{t+1}|X_{t+1})B'(X_{t+1})$$



- Basic idea: beliefs "reweighted" by likelihood of evidence
- Unlike passage of time, we have to renormalize

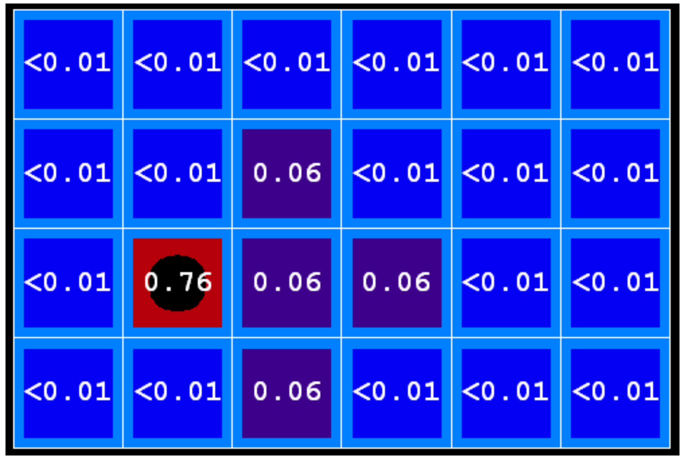
Filtering

Elapse time: compute P(X_t | $e_{1:t-1}$)

$$P(x_t|e_{1:t-1}) = \sum_{x_{t-1}} P(x_{t-1}|e_{1:t-1}) \cdot P(x_t|x_{t-1})$$

Observe: compute P(X_t | $e_{1:t}$)

$$P(x_t|e_{1:t}) \propto P(x_t|e_{1:t-1}) \cdot P(e_t|x_t)$$



<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
<0.01	<0.01	0.06	<0.01	<0.01	<0.01
<0.01	0.76	0.06	0.06	<0.01	<0.01
<0.01	<0.01	0.06	<0.01	<0.01	<0.01

$$\begin{tikzcd} X_1 \arrow[r] \arrow[d] & X_2 \arrow[d] \\ E_1 & E_2 \end{tikzcd}$$

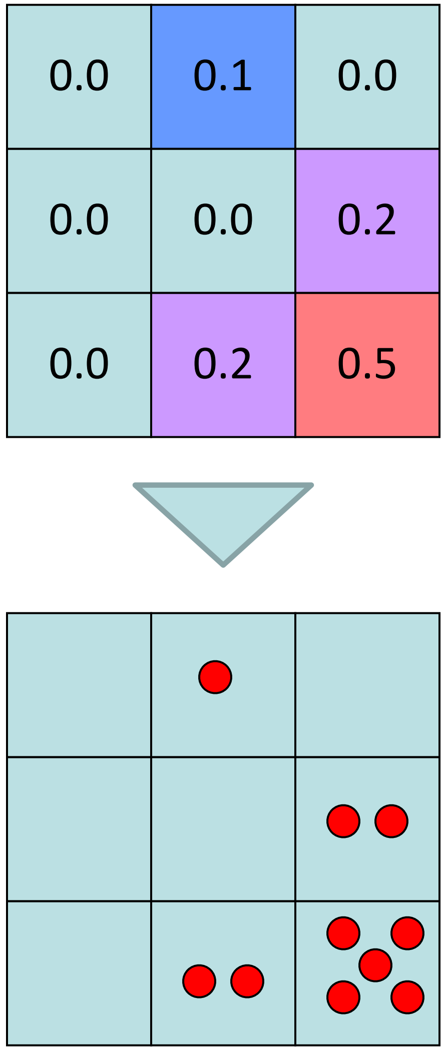
Belief: <P(rain), P(sun)>

$P(X_1)$	<0.5, 0.5>	*Prior on X_1*
$P(X_1 \mid E_1 = umbrella)$	<0.82, 0.18>	*Observe*
$P(X_2 \mid E_1 = umbrella)$	<0.63, 0.37>	*Elapse time*
$P(X_2 \mid E_1 = umb, E_2 = umb)$	<0.88, 0.12>	*Observe*

[Demo: Ghostbusters Exact Filtering (L15D2)]

Particle Filtering

- Filtering: approximate solution
- Sometimes |X| is too big to use exact inference
  - |X| may be too big to even store B(X)
  - E.g. X is continuous
- Solution: approximate inference
  - Track samples of X, not all values
  - Samples are called particles
  - Time per step is linear in the number of samples
  - But: number needed may be large
  - In memory: list of particles, not states
- This is how robot localization works in practice
- Particle is just new name for sample



Particle Filtering: Elapse Time

- Each particle is moved by sampling its next position from the transition model

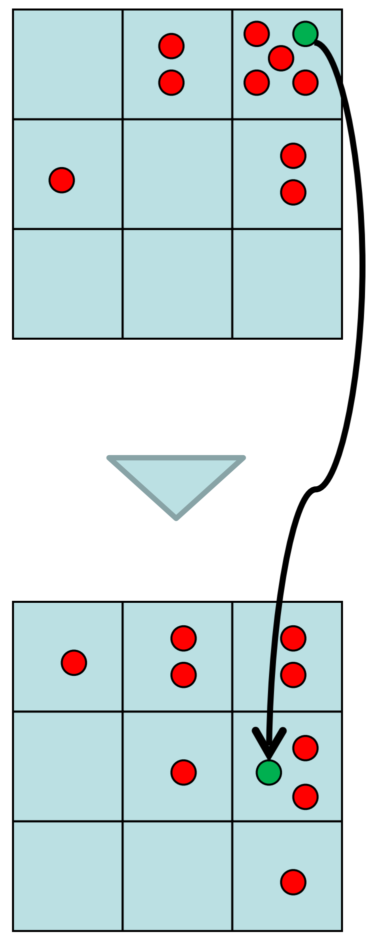
$$x' = \text{sample}(P(X'|x))$$

  - This is like prior sampling – samples' frequencies reflect the transition probabilities
  - Here, most samples move clockwise, but some move in another direction or stay in place

- This captures the passage of time
  - If enough samples, close to exact values before and after (consistent)

Particles:
(3,3)
(2,3)
(3,3)
(3,2)
(3,3)
(3,2)
(1,2)
(3,3)
(3,3)
(2,3)

Particles:
(3,2)
(2,3)
(3,2)
(3,1)
(3,3)
(3,2)
(1,3)
(2,3)
(3,2)
(2,2)



Particle Filtering: Observe

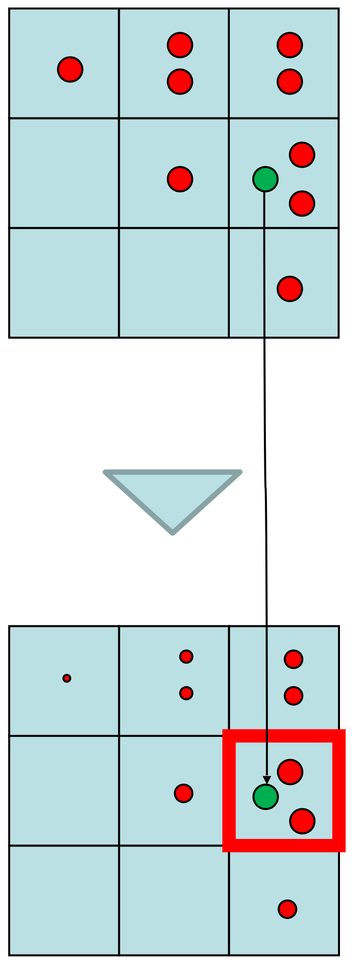
- Slightly trickier:
  - Don’t sample observation, fix it
  - Similar to likelihood weighting, downweight samples based on the evidence

$$w(x) = P(e|x)$$

$$B(X) \propto P(e|X)B'(X)$$

  - As before, the probabilities don’t sum to one, since all have been down-weighted (in fact they now sum to (N times) an approximation of P(e))

Particles:
(3,2)
(2,3)
(3,2)
(3,1)
(3,3)
(3,2)
(1,3)
(2,3)
(3,2)
(2,2)



Particles:

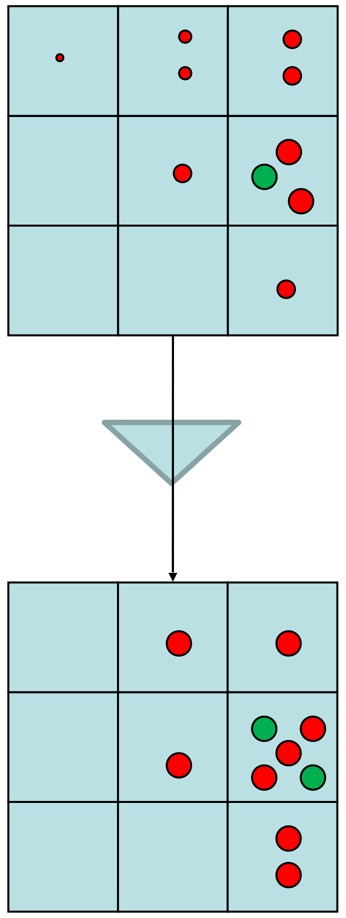
(3,2)	w=.9
(2,3)	w=.2
(3,2)	w=.9
(3,1)	w=.4
(3,3)	w=.4
(3,2)	w=.9
(1,3)	w=.1
(2,3)	w=.2
(3,2)	w=.9
(2,2)	w=.4

Particle Filtering: Resample

- Rather than tracking weighted samples, we resample
- N times, we choose from our weighted sample distribution (i.e. draw with replacement)
- This is equivalent to renormalizing the distribution
- Now the update is complete for this time step, continue with the next one

Particles:

(3,2) w=.9
(2,3) w=.2
(3,2) w=.9
(3,1) w=.4
(3,3) w=.4
(3,2) w=.9
(1,3) w=.1
(2,3) w=.2
(3,2) w=.9
(2,2) w=.4

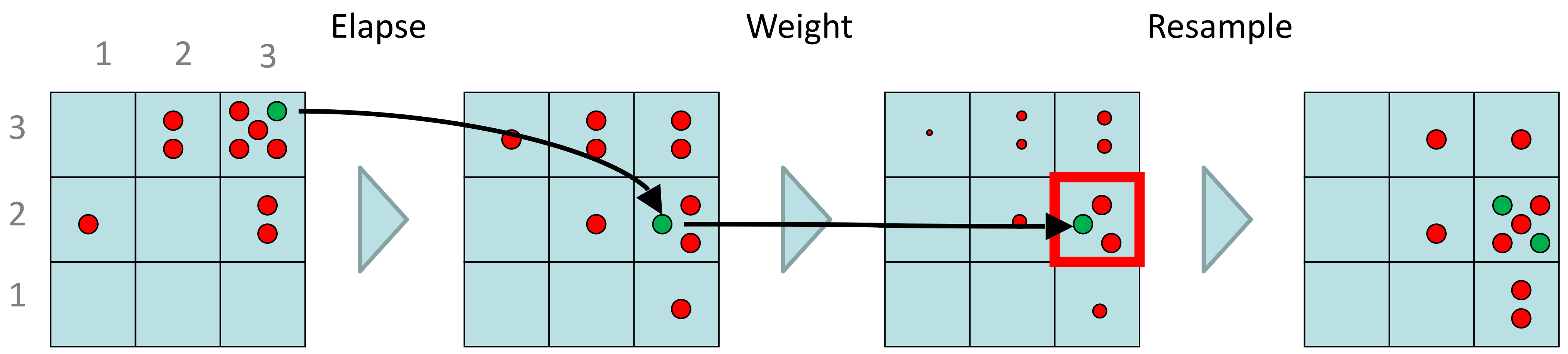


(New) Particles:

(3,2)
(2,2)
(3,2)
(2,3)
(3,3)
(3,2)
(1,3)
(2,3)
(3,2)
(3,2)

Recap: Particle Filtering

- Particles: track samples of states rather than an explicit distribution

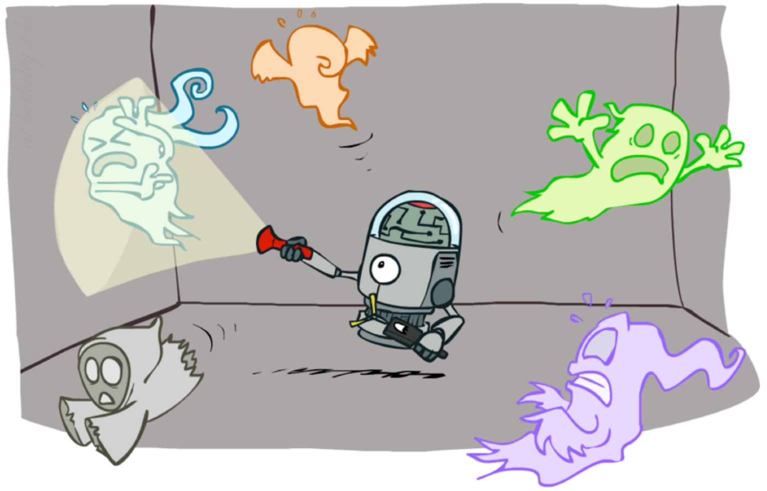


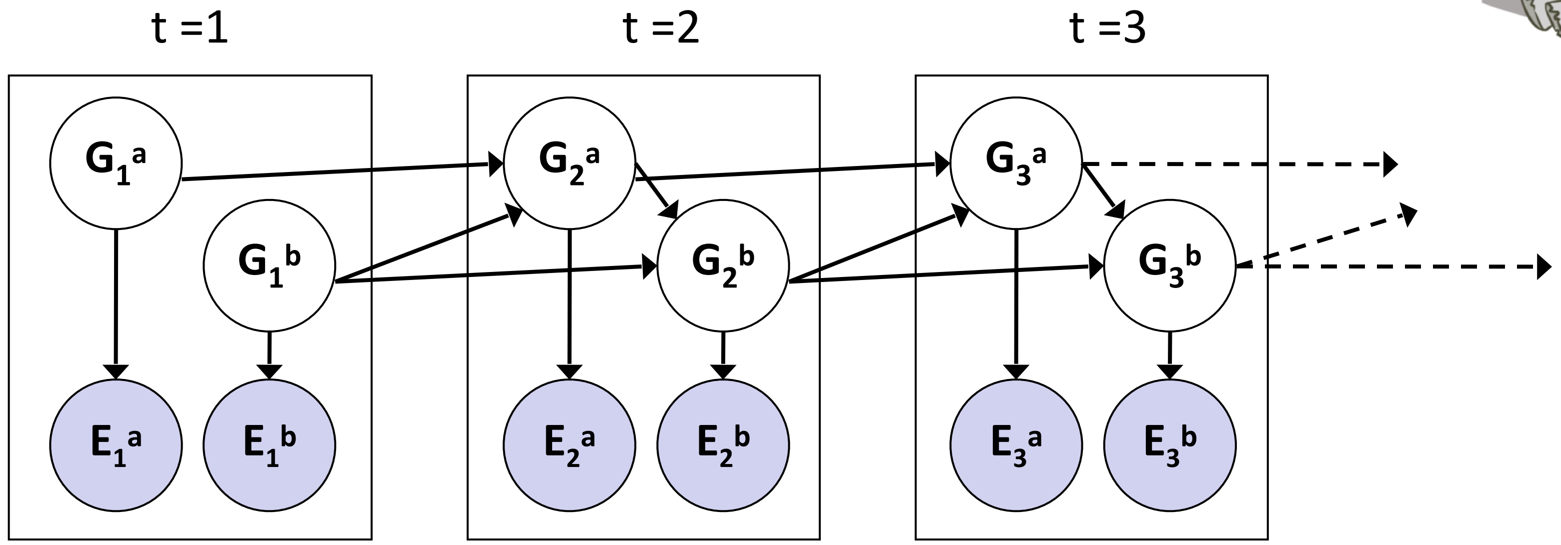
Particles:	Particles:	Particles:	(New) Particles:
(3,3)	(3,2)	(3,2) w=.9	(3,2)
(2,3)	(2,3)	(2,3) w=.2	(2,2)
(3,3)	(3,2)	(3,2) w=.9	(3,2)
(3,2)	(3,1)	(3,1) w=.4	(2,3)
(3,3)	(3,3)	(3,3) w=.4	(3,3)
(3,2)	(3,2)	(3,2) w=.9	(3,2)
(1,2)	(1,3)	(1,3) w=.1	(1,3)
(3,3)	(2,3)	(2,3) w=.2	(2,3)
(3,3)	(3,2)	(3,2) w=.9	(3,2)
(2,3)	(2,2)	(2,2) w=.4	(3,2)

[Demos: ghostbusters particle filtering (L15D3,4,5)]

Dynamic Bayes Nets (DBNs)

- We want to track multiple variables over time, using multiple sources of evidence
- Idea: Repeat a fixed Bayes net structure at each time
- Variables from time *t* can condition on those from *t-1*



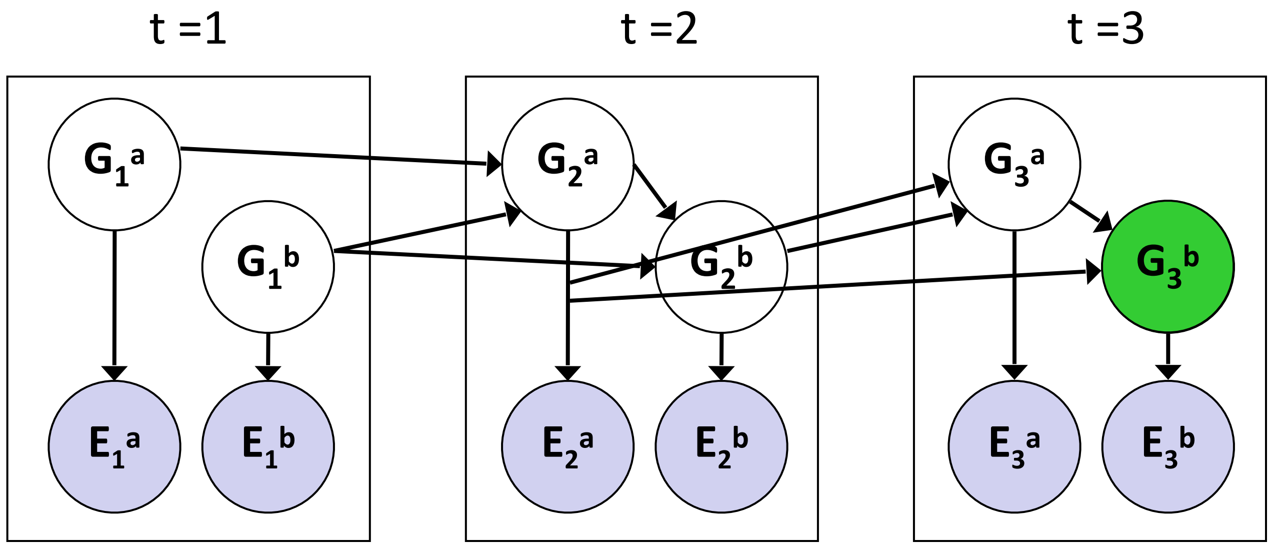


- Dynamic Bayes nets are a generalization of HMMs

[Demo: pacman sonar ghost DBN model (L15D6)]

Exact Inference in DBNs

- Variable elimination applies to dynamic Bayes nets
- Procedure: "unroll" the network for T time steps, then eliminate variables until $P(X_T|e_{1:T})$ is computed



- Online belief updates: Eliminate all variables from the previous time step; store factors for current time only

DBN Particle Filters

- A particle is a complete sample for a time step

- **Initialize**: Generate prior samples for the t=1 Bayes net
  - Example particle: $\mathbf{G_1^a}$ = (3,3) $\mathbf{G_1^b}$ = (5,3)

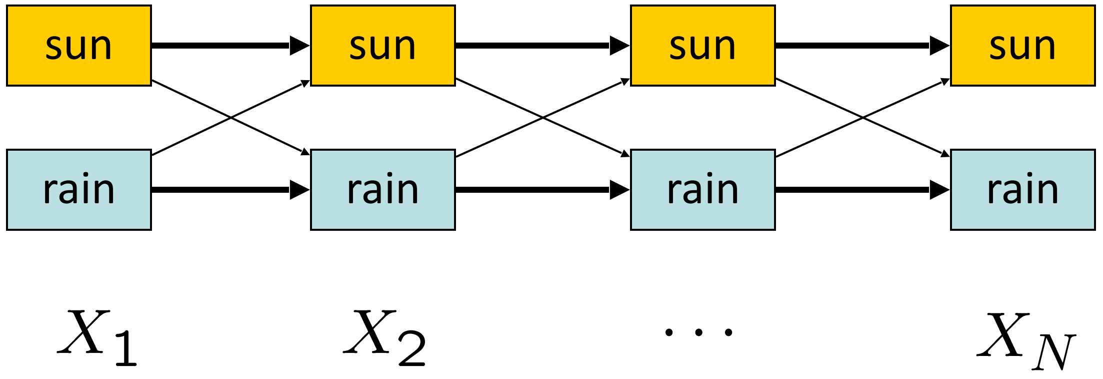
- **Elapse time**: Sample a successor for each particle
  - Example successor: $\mathbf{G_2^a}$ = (2,3) $\mathbf{G_2^b}$ = (6,3)

- **Observe**: Weight each *entire* sample by the likelihood of the evidence conditioned on the sample
  - Likelihood: $P(\mathbf{E_1^a} \mid \mathbf{G_1^a}) * P(\mathbf{E_1^b} \mid \mathbf{G_1^b})$

- **Resample:** Select prior samples (tuples of values) in proportion to their likelihood

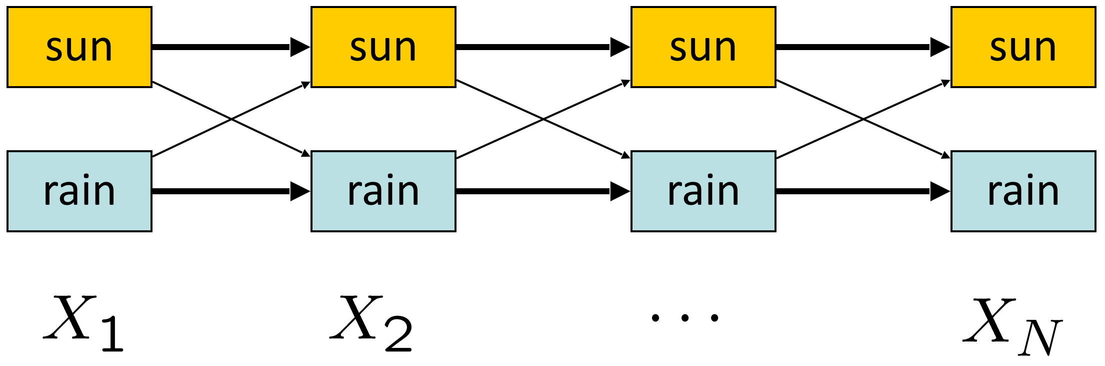
State Trellis

- State trellis: graph of states and transitions over time



- Each arc represents some transition $x_{t-1} \rightarrow x_t$
- Each arc has weight $P(x_t|x_{t-1})P(e_t|x_t)$
- Each path is a sequence of states
- The product of weights on a path is that sequence's probability along with the evidence
- Forward algorithm computes sums of paths, Viterbi computes best paths

Forward / Viterbi Algorithms



Forward Algorithm (Sum)

$$f_t[x_t] = P(x_t, e_{1:t})$$

$$= P(e_t|x_t) \sum_{x_{t-1}} P(x_t|x_{t-1}) f_{t-1}[x_{t-1}]$$

Viterbi Algorithm (Max)

$$m_t[x_t] = \max_{x_{1:t-1}} P(x_{1:t-1}, x_t, e_{1:t})$$

$$= P(e_t|x_t) \max_{x_{t-1}} P(x_t|x_{t-1}) m_{t-1}[x_{t-1}]$$

CS 188: Artificial Intelligence

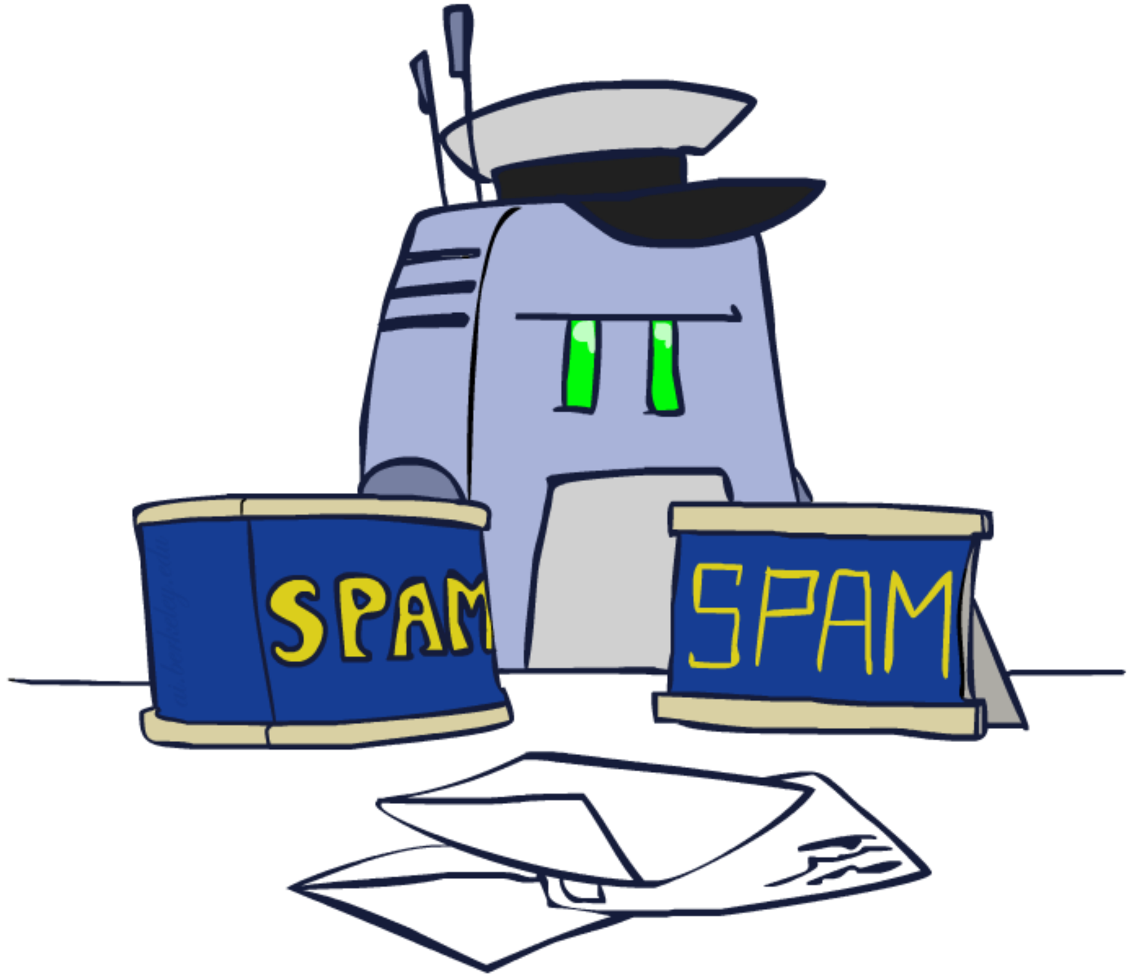
Naïve Bayes

Instructor: Oliver Grillmeyer --- University of California, Berkeley

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Model-Based Classification

- Model-based approach
  - Build a model (e.g. Bayes' net) where both the output label and input features are random variables
  - Instantiate any observed features
  - Query for the distribution of the label conditioned on the features
- Challenges
  - What structure should the BN have?
  - How should we learn its parameters?



Inference for Naïve Bayes

- Goal: compute posterior distribution over label variable Y
  - Step 1: get joint probability of label and evidence for each label

$$\begin{tikzcd}
P(Y, f_1 \ldots f_n) = \begin{bmatrix} P(y_1, f_1 \ldots f_n) \\ P(y_2, f_1 \ldots f_n) \\ \vdots \\ P(y_k, f_1 \ldots f_n) \end{bmatrix} \arrow[r, Rightarrow] & \begin{bmatrix} P(y_1) \prod_i P(f_i|y_1) \\ P(y_2) \prod_i P(f_i|y_2) \\ \vdots \\ P(y_k) \prod_i P(f_i|y_k) \end{bmatrix} \arrow[d, bend left, "+"] \\
 & \overline{P(f_1 \ldots f_n)} \arrow[d, Rightarrow] \\
 & P(Y|f_1 \ldots f_n)
\end{tikzcd}$$

  - Step 2: sum to get probability of evidence
  - Step 3: normalize by dividing Step 1 by Step 2

Naïve Bayes for Text

- Bag-of-words Naïve Bayes:
  - Features: W_i is the word at position i
  - As before: predict label conditioned on feature variables (spam vs. ham)
  - As before: assume features are conditionally independent given label
  - New: each W_i is identically distributed

- Generative model: $P(Y, W_1 \dots W_n) = P(Y) \prod_i P(\underset{\curvearrowright}{W_i}|Y)$

Word at position i, not ith word in the dictionary!

- "Tied" distributions and bag-of-words
  - Usually, each variable gets its own conditional probability distribution P(F|Y)
  - In a bag-of-words model
    - Each position is identically distributed
    - All positions share the same conditional probs P(W|Y)
    - Why make this assumption?
  - Called "bag-of-words" because model is insensitive to word order or reordering

Example: Spam Filtering

- Model: $P(Y, W_1 \ldots W_n) = P(Y) \prod_i P(W_i|Y)$
- What are the parameters?

$P(Y)$

ham :	0.66
spam:	0.33

$P(W|\text{spam})$

the :	0.0156
to :	0.0153
and :	0.0115
of :	0.0095
you :	0.0093
a :	0.0086
with:	0.0080
from:	0.0075
...	

$P(W|\text{ham})$

the :	0.0210
to :	0.0133
of :	0.0119
2002:	0.0110
with:	0.0108
from:	0.0107
and :	0.0105
a :	0.0100
...	

- Where do these tables come from?

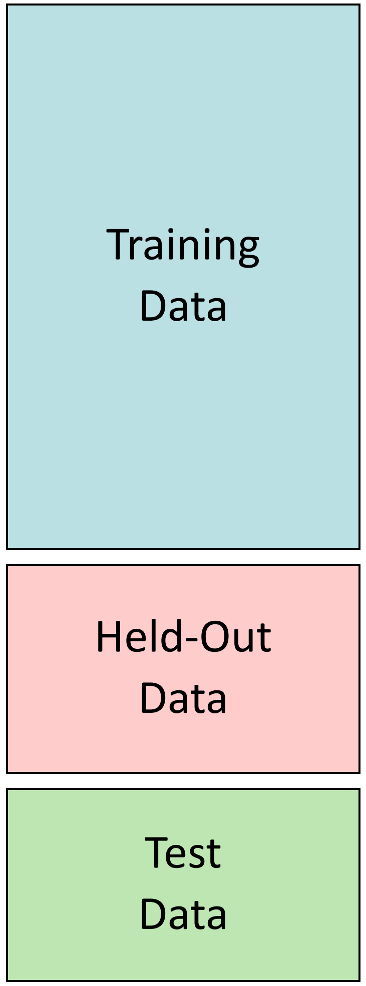
Spam Example

Word	P(w|spam)	P(w|ham)	Tot Spam	Tot Ham
(prior)	0.33333	0.66666	-1.1	-0.4

P(spam | w) = 98.9

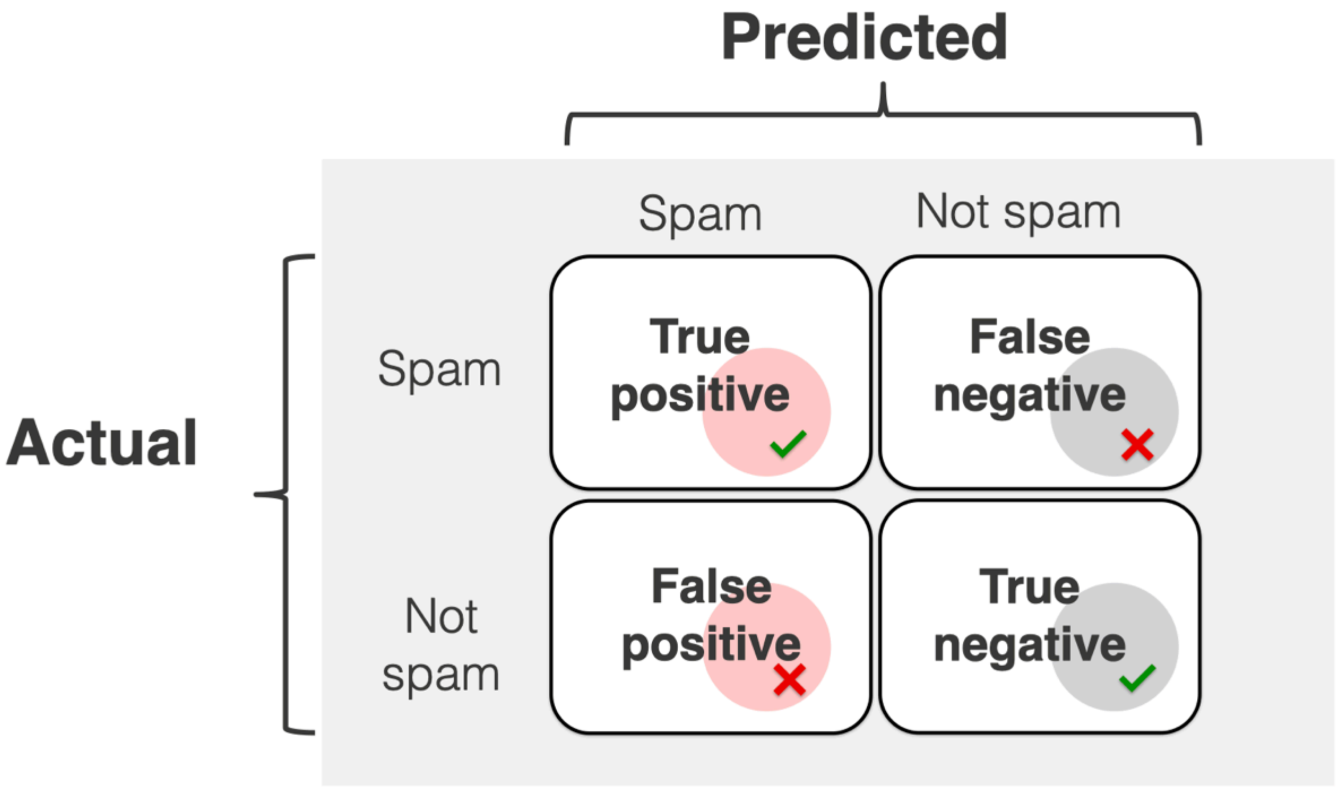
Important Concepts

- Data: labeled instances (e.g. emails marked spam/ham)
  - Training set
  - Held out set
  - Test set
- Features: attribute-value pairs which characterize each x
- Experimentation cycle
  - Learn parameters (e.g. model probabilities) on training set
  - (Tune hyperparameters on held-out set)
  - Compute accuracy of test set
  - Very important: never “peek” at the test set!
- Evaluation (many metrics possible, e.g. accuracy)
  - Accuracy: fraction of instances predicted correctly
- Overfitting and generalization
  - Want a classifier which does well on *test* data
  - Overfitting: fitting the training data very closely, but not generalizing well
  - We’ll investigate overfitting and generalization formally in a few lectures

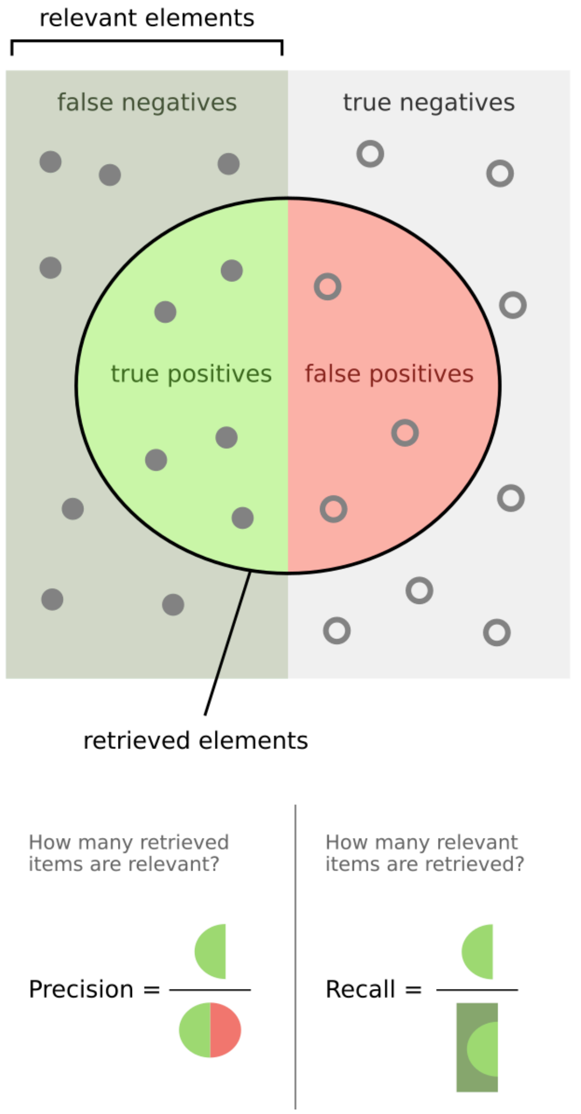


Confusion Matrix

- Used to show space of actual and predicted values



From evidentlyai.com

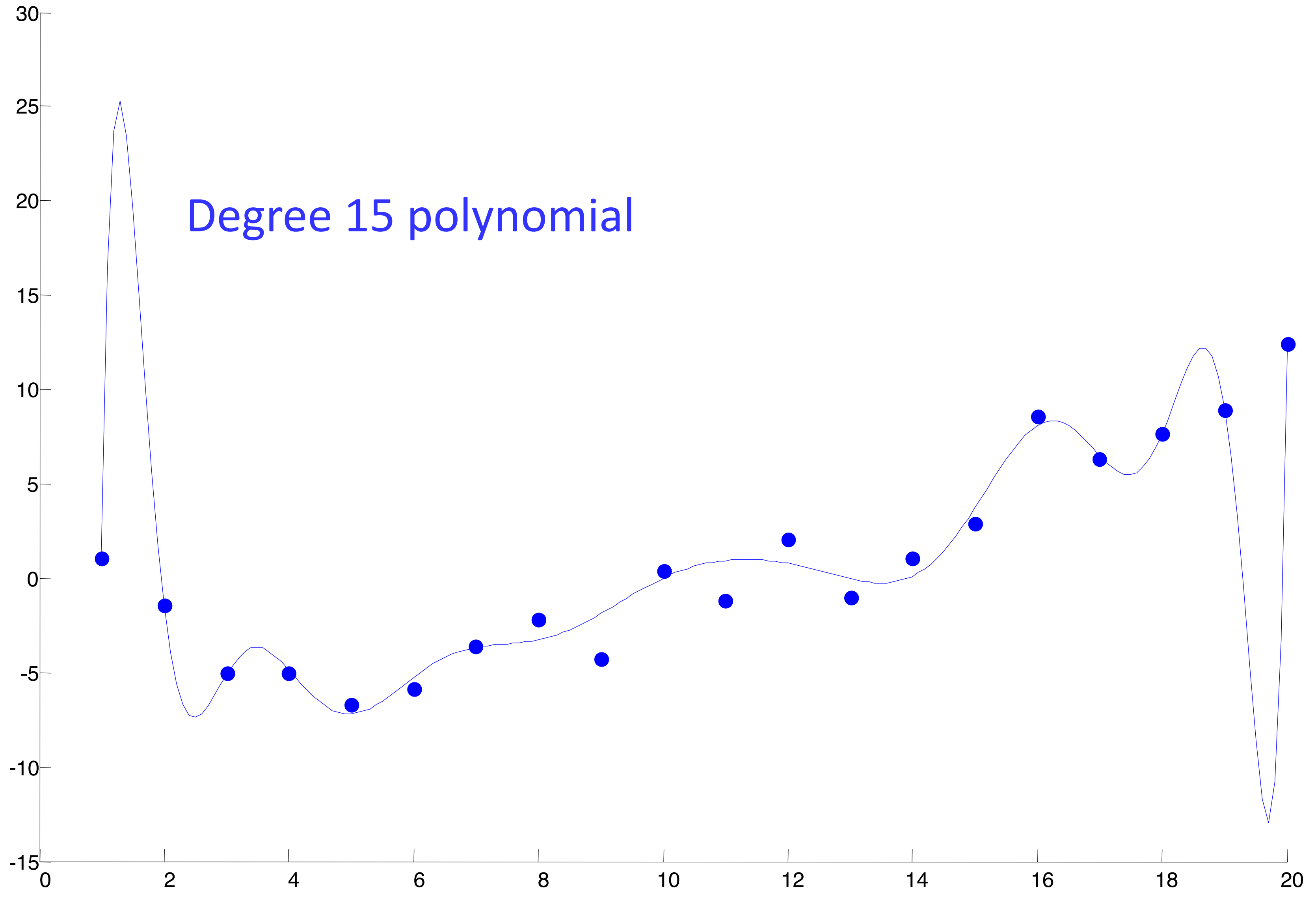


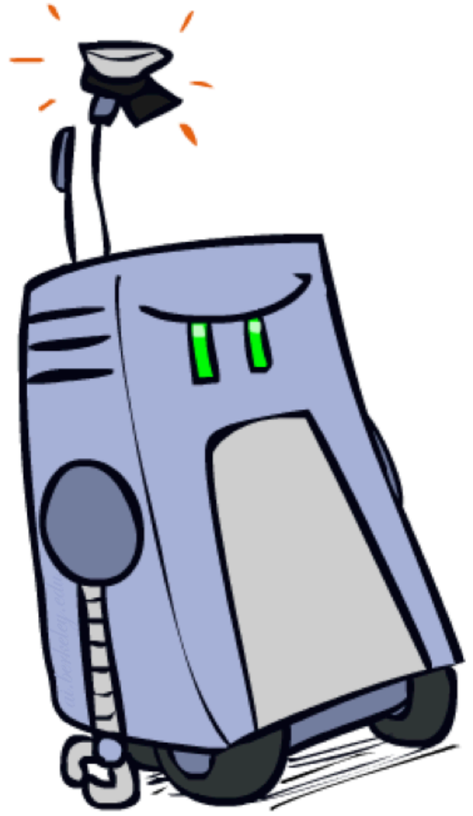
From Wikipedia

Performance Metrics

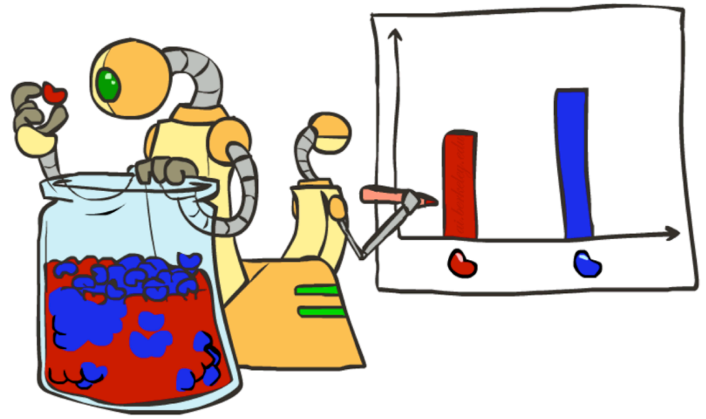
- Accuracy
  - Number of correct predictions from the entire data set:
    (TP + TN) / (TP + FP + TN + FN)
- Precision
  - Number of correct positive predictions from total positive predictions:
    TP / (TP + FP)
- Recall
  - Number of correct positive predictions from the actual positive samples:
    TP / (TP + FN)
- F-score
  - Harmonic mean of Precision and Recall: 2TP / (2TP + FP + FN)

Overfitting



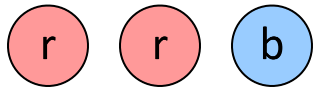


Parameter Estimation



- Estimating the distribution of a random variable
- *Elicitation:* ask a human (why is this hard?)
- *Empirically:* use training data (learning!)
  - E.g.: for each outcome x, look at the *empirical rate* of that value:

$$P_{\text{ML}}(x) = \frac{\text{count}(x)}{\text{total samples}}$$



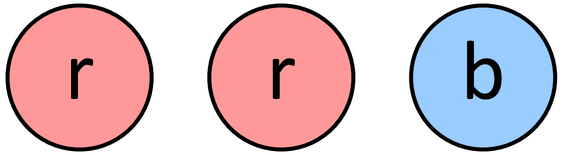
$$P_{\text{ML}}(\text{r}) = 2/3$$

  - This is the estimate that maximizes the *likelihood of the data*

$$L(x,\theta) = \prod_i P_\theta(x_i)$$

Laplace Smoothing

- Laplace's estimate:
  - Pretend you saw every outcome once more than you actually did



$$P_{LAP}(x) = \frac{c(x)+1}{\sum_x [c(x)+1]}$$

$$= \frac{c(x)+1}{N+|X|}$$

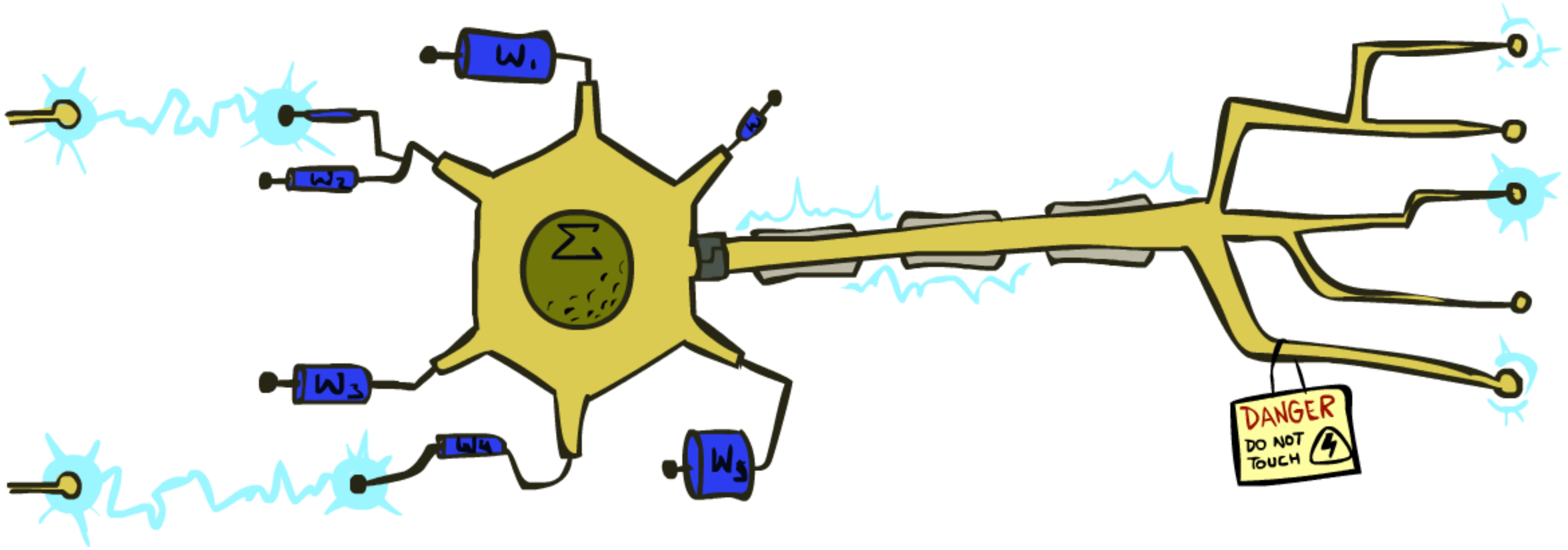
$$P_{ML}(X) =$$

$$P_{LAP}(X) =$$

  - Can derive this estimate with *Dirichlet priors* (see cs281a)

CS 188: Artificial Intelligence

Perceptrons and Logistic Regression



Instructor: Oliver Grillmeyer —- University of California, Berkeley

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Feature Vectors

$$x \qquad f(x) \qquad y$$

```
Hello,

Do you want free printr
cartriges?  Why pay more
when you can get them
ABSOLUTELY FREE!  Just
```

$\Rightarrow$

```
# free       : 2
YOUR_NAME    : 0
MISSPELLED   : 2
FROM_FRIEND  : 0
...
```

$\Rightarrow$

SPAM
or
+

$\Rightarrow$

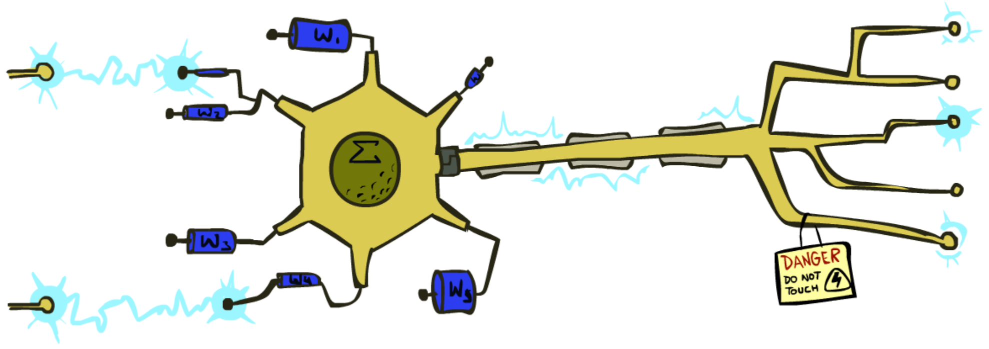
```
PIXEL-7,12   : 1
PIXEL-7,13   : 0
...
NUM_LOOPS    : 1
...
```

$\Rightarrow$

"2"

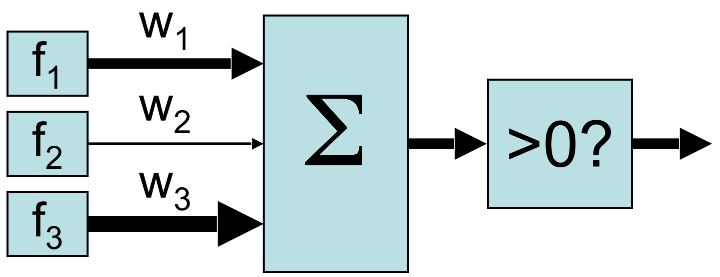
Linear Classifiers

- Inputs are feature values
- Each feature has a weight
- Sum is the activation



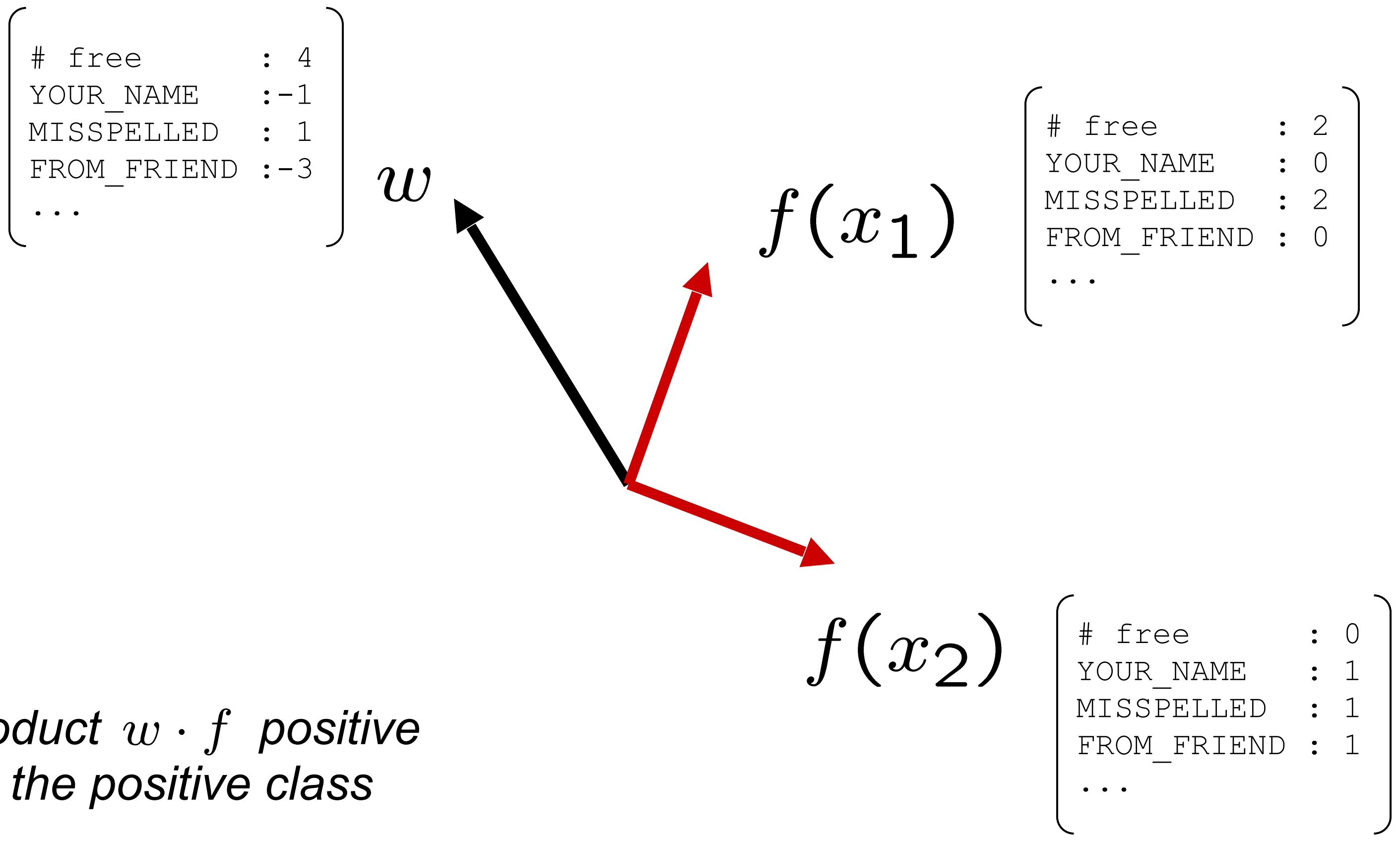
$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- If the activation is:
  - Positive, output +1
  - Negative, output -1



Weights

- Binary case: compare features to a weight vector
- Learning: figure out the weight vector from examples



```
# free      : 4
YOUR_NAME   :-1
MISSPELLED  : 1
FROM_FRIEND :-3
...
```

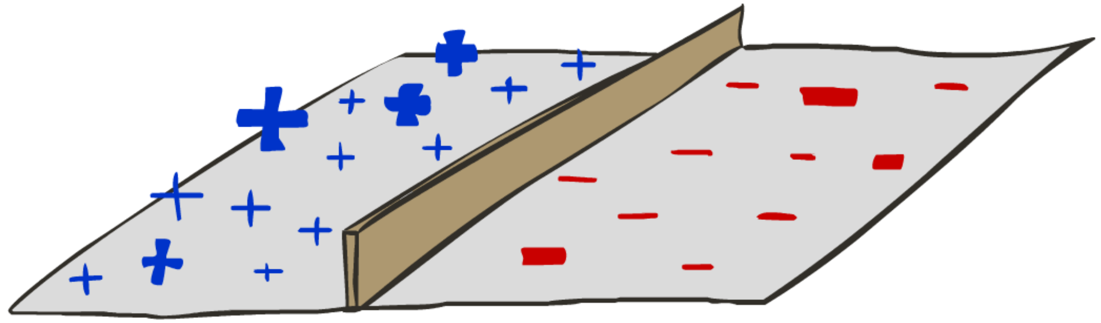
```
# free      : 2
YOUR_NAME   : 0
MISSPELLED  : 2
FROM_FRIEND : 0
...
```

```
# free      : 0
YOUR_NAME   : 1
MISSPELLED  : 1
FROM_FRIEND : 1
...
```

Dot product $w \cdot f$ *positive means the positive class*

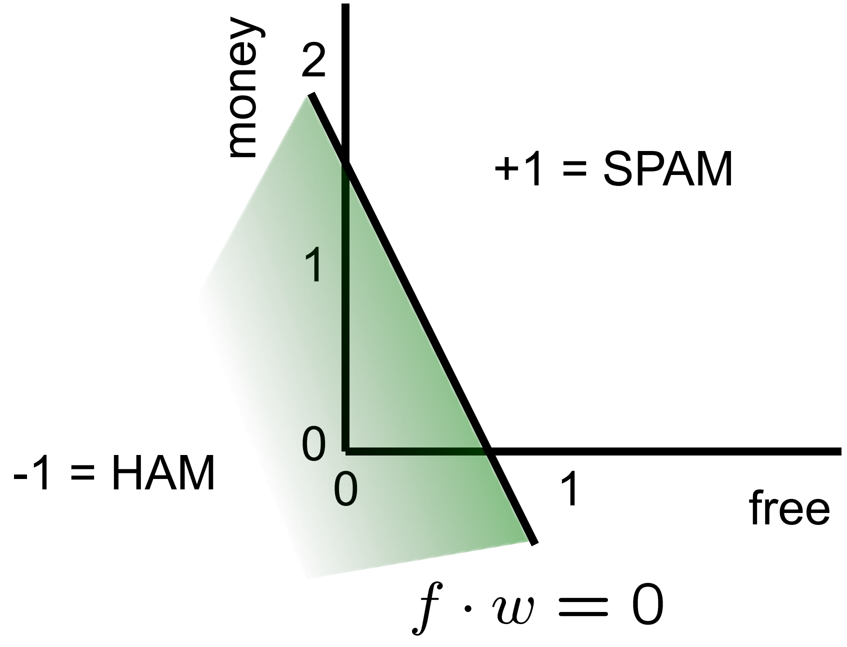
Binary Decision Rule

- In the space of feature vectors
  - Examples are points
  - Any weight vector is a hyperplane
  - One side corresponds to Y=+1
  - Other corresponds to Y=-1



w

```
BIAS  : -3
free  :  4
money :  2
...
```



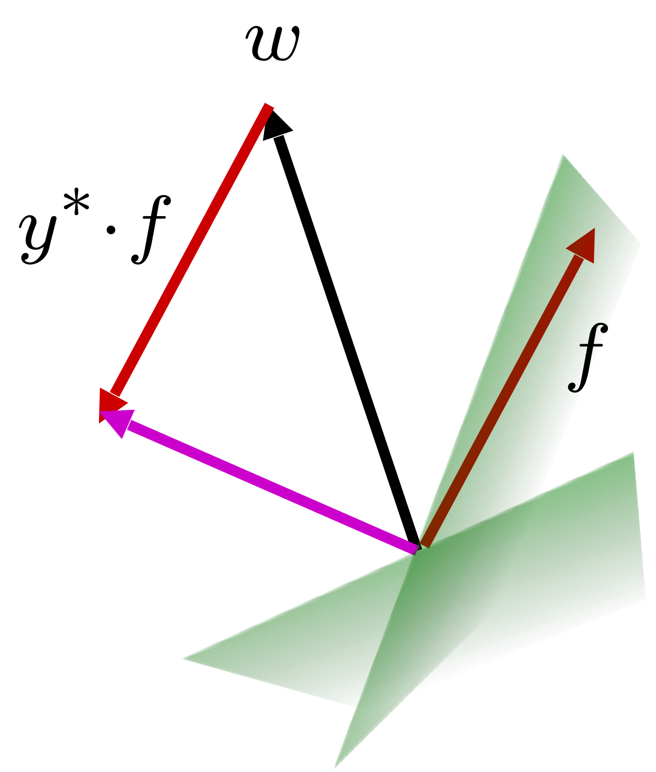
Learning: Binary Perceptron

- Start with weights = 0
- For each training instance:
  - Classify with current weights

$$y = \begin{cases} +1 & \text{if} \quad w \cdot f(x) \geq 0 \\ -1 & \text{if} \quad w \cdot f(x) < 0 \end{cases}$$

  - If correct (i.e., y=y*), no change!
  - If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if y* is -1.

$$w = w + y^* \cdot f$$



Multiclass Decision Rule

- If we have multiple classes:
  - A weight vector for each class:

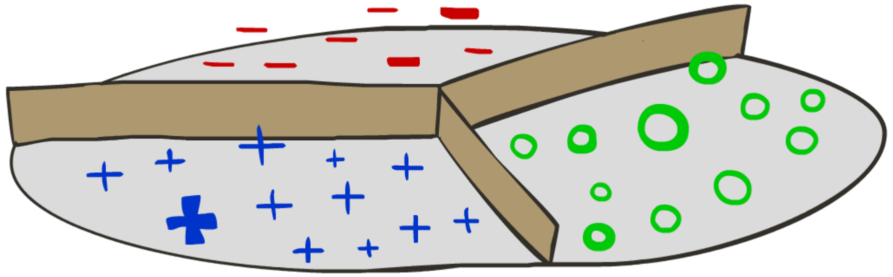
$$w_y$$

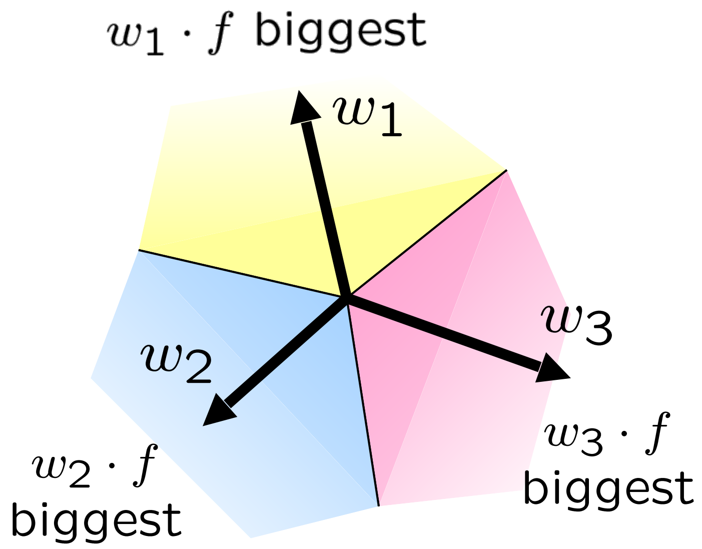
  - Score (activation) of a class y:

$$w_y \cdot f(x)$$

  - Prediction highest score wins

$$y = \arg\max_y \ w_y \cdot f(x)$$





Binary = multiclass where the negative class has weight zero

Learning: Multiclass Perceptron

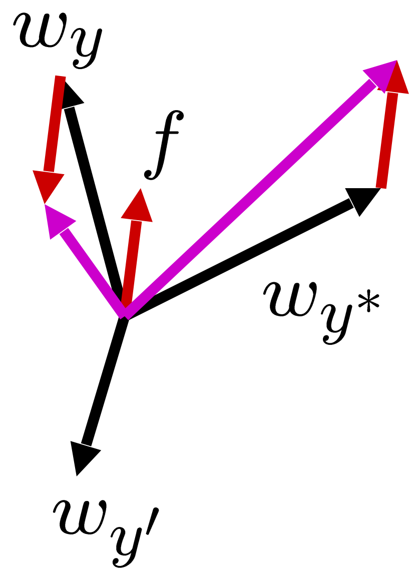
- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights

$$y = \arg\max_y \; w_y \cdot f(x)$$

- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

$$w_y = w_y - f(x)$$

$$w_{y^*} = w_{y^*} + f(x)$$



Example: Multiclass Perceptron

"win the vote"

"win the election"

"win the game"

```
Sample A
BIAS : 1
win  : 1
game : 0
vote : 1
the  : 1
```

wS . fA = 1; wP . fA = 0; wT . fA = 0

```
Sample B
BIAS : 1
win  : 1
game : 0
vote : 0
the  : 1
```

wS . fB = -2; wP . fB = 3; wT . fB = 0

```
Sample C
BIAS : 1
win  : 1
game : 1
vote : 0
the  : 1
```

wS . fC = -2; wP . fC = 3; wT . fC = 0

w_{SPORTS}

```
                  fA     wS      fC     wS
BIAS   : 1        1       0       1      1
win    : 0        1      -1       1      0
game   : 0    -   0   =   0   +   1  =   1
vote   : 0        1      -1       0     -1
the    : 0        1      -1       1      0
...                      ...            ...
```

$w_{POLITICS}$

```
                  fA     wP      fC     wP
BIAS   : 0        1       1       1      0
win    : 0        1       1       1      0
game   : 0    +   0   =   0   -   1  =  -1
vote   : 0        1       1       0      1
the    : 0        1       1       1      0
...                      ...            ...
```

w_{TECH}

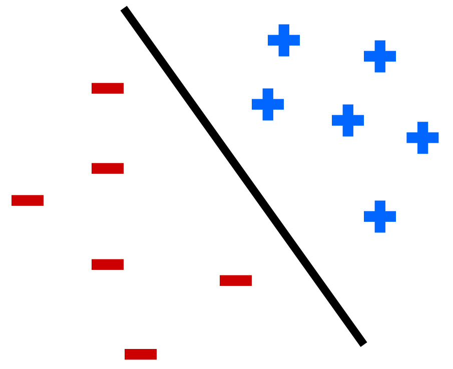
```
BIAS   : 0
win    : 0
game   : 0
vote   : 0
the    : 0
...
```

Properties of Perceptrons

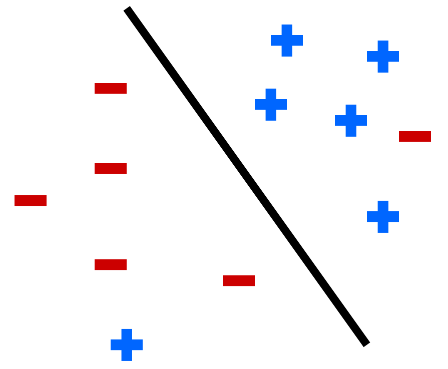
- Separability: true if some parameters get the training set perfectly correct
- Convergence: if the training is separable, perceptron will eventually converge (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

$$\text{mistakes} < \frac{k}{\delta^2}$$

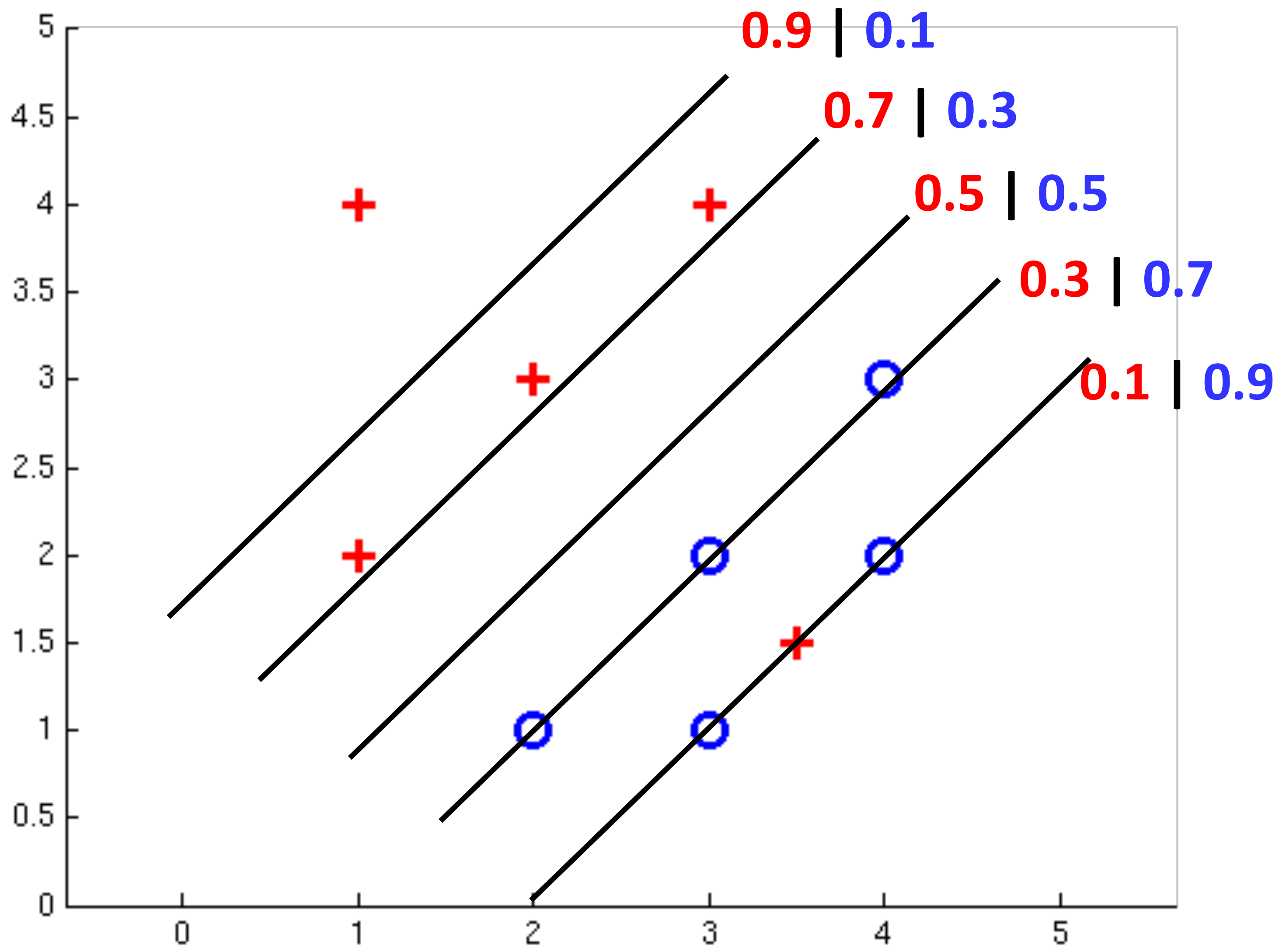
Separable



Non-Separable



Non-Separable Case: Probabilistic Decision

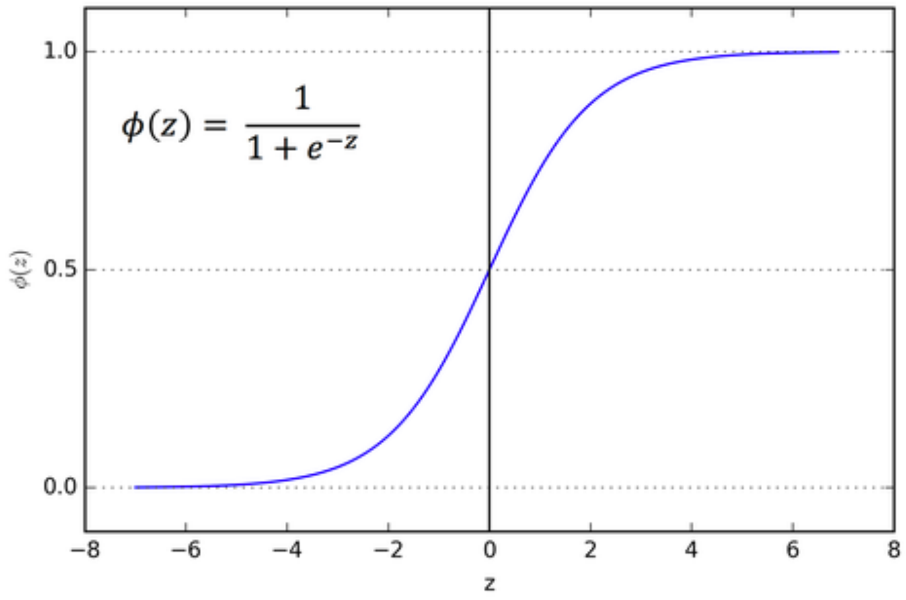


How to get probabilistic decisions?

- Perceptron scoring: $z = w \cdot f(x)$
- If $z = w \cdot f(x)$ very positive → want probability going to 1
- If $z = w \cdot f(x)$ very negative → want probability going to 0

- Sigmoid function

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



Best w?

- Maximum likelihood estimation:

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)};w)$$

with:

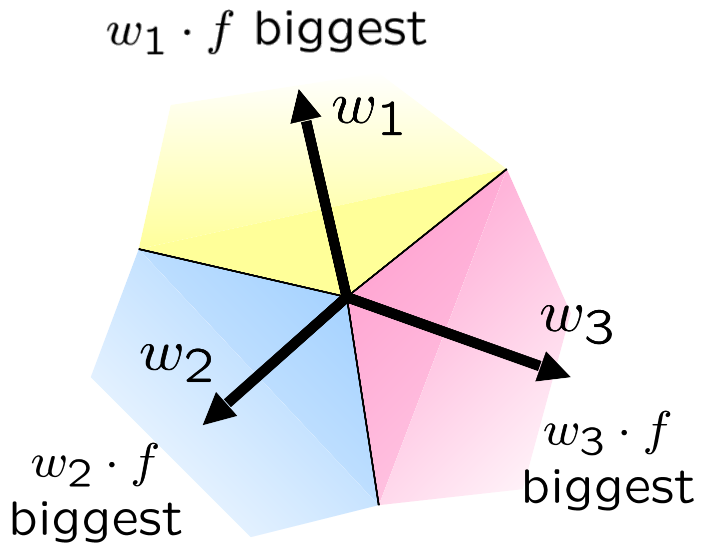
$$P(y^{(i)} = +1|x^{(i)};w) = \frac{1}{1+e^{-w\cdot f(x^{(i)})}}$$

$$P(y^{(i)} = -1|x^{(i)};w) = 1 - \frac{1}{1+e^{-w\cdot f(x^{(i)})}}$$

= Logistic Regression

Multiclass Logistic Regression

- Recall Perceptron:
  - A weight vector for each class: w_y
  - Score (activation) of a class y: $w_y \cdot f(x)$
  - Prediction highest score wins $y = \arg\max_y \; w_y \cdot f(x)$



- How to make the scores into probabilities?

$$\underbrace{z_1, z_2, z_3}_{\text{original activations}} \rightarrow \underbrace{\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}, \frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}, \frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}}_{\text{softmax activations}}$$

Best w?

- Maximum likelihood estimation:

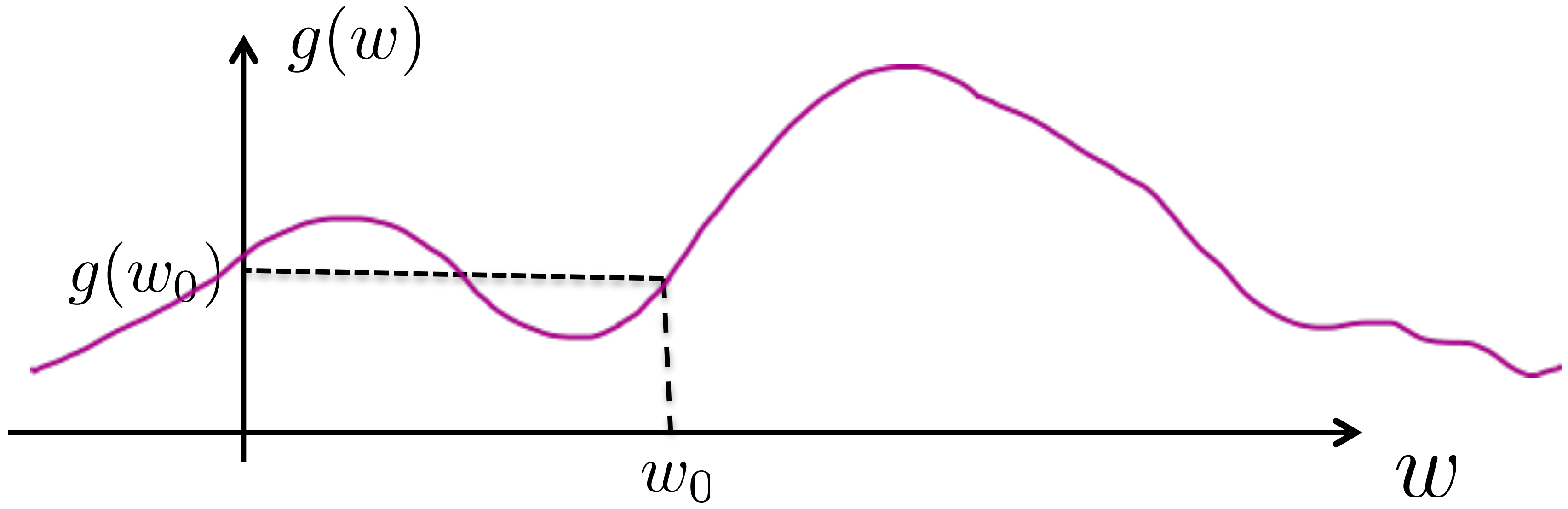
$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

with:

$$P(y^{(i)}|x^{(i)}; w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$$

= Multi-Class Logistic Regression

1-D Optimization



- Could evaluate $g(w_0 + h)$ and $g(w_0 - h)$
  - Then step in best direction

- Or, evaluate derivative: $$\frac{\partial g(w_0)}{\partial w} = \lim_{h \to 0} \frac{g(w_0 + h) - g(w_0 - h)}{2h}$$
  - Tells which direction to step into

Optimization Procedure: Gradient Ascent

```
▪ init $w$
▪ for iter = 1, 2, …
      $w \leftarrow w + \alpha * \nabla g(w)$
```

- α: learning rate --- hyperparameter that needs to be chosen carefully
- How? Try multiple choices
  - Crude rule of thumb: update changes w about 0.1 – 1 %

Batch Gradient Ascent on the Log Likelihood Objective

$$\max_w \quad ll(w) = \max_w \quad \underbrace{\sum_i \log P(y^{(i)}|x^{(i)};w)}_{g(w)}$$

```
▪ init $w$
▪ for iter = 1, 2, …
    $w \leftarrow w + \alpha * \sum_i \nabla \log P(y^{(i)}|x^{(i)};w)$
```

Stochastic Gradient Ascent on the Log Likelihood Objective

$$\max_w \; ll(w) = \max_w \; \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

Observation: once gradient on one training example has been computed, might as well incorporate before computing next one

```
▪ init $w$
▪ for iter = 1, 2, …
    ▪ pick random j
      $w \leftarrow w + \alpha * \nabla \log P(y^{(j)} | x^{(j)}; w)$
```

Mini-Batch Gradient Ascent on the Log Likelihood Objective

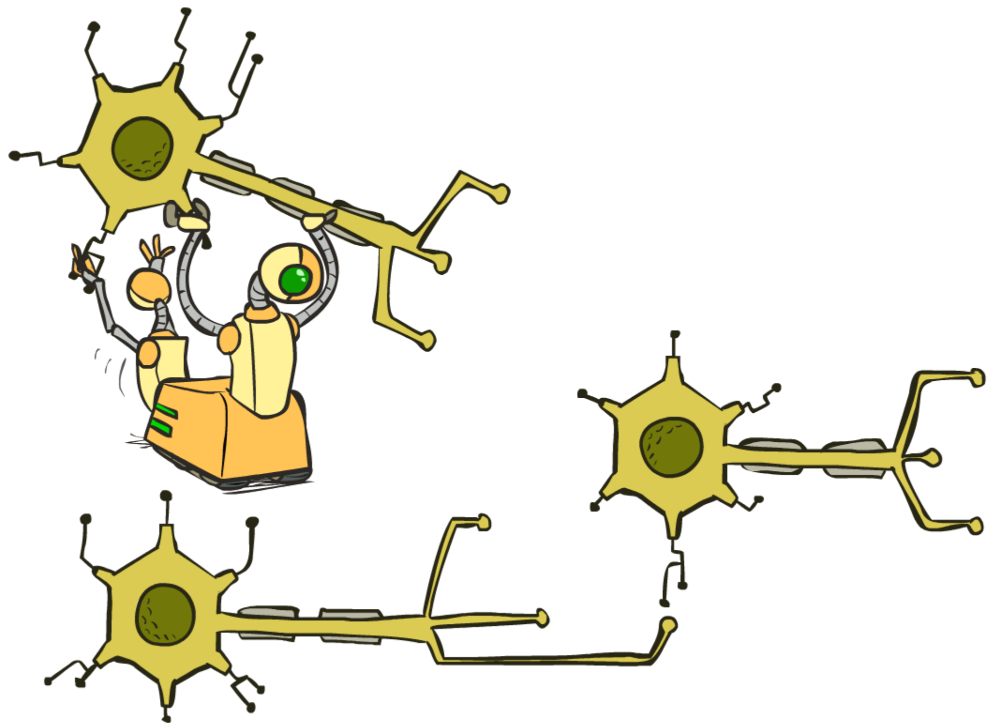
$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

Observation: gradient over small set of training examples (=mini-batch) can be computed in parallel, might as well do that instead of a single one

```
- init $w$
- for iter = 1, 2, …
    - pick random subset of training examples J
      $w \leftarrow w + \alpha * \sum_{j \in J} \nabla \log P(y^{(j)}|x^{(j)}; w)$
```

CS 188: Artificial Intelligence

Neural Nets and Applications

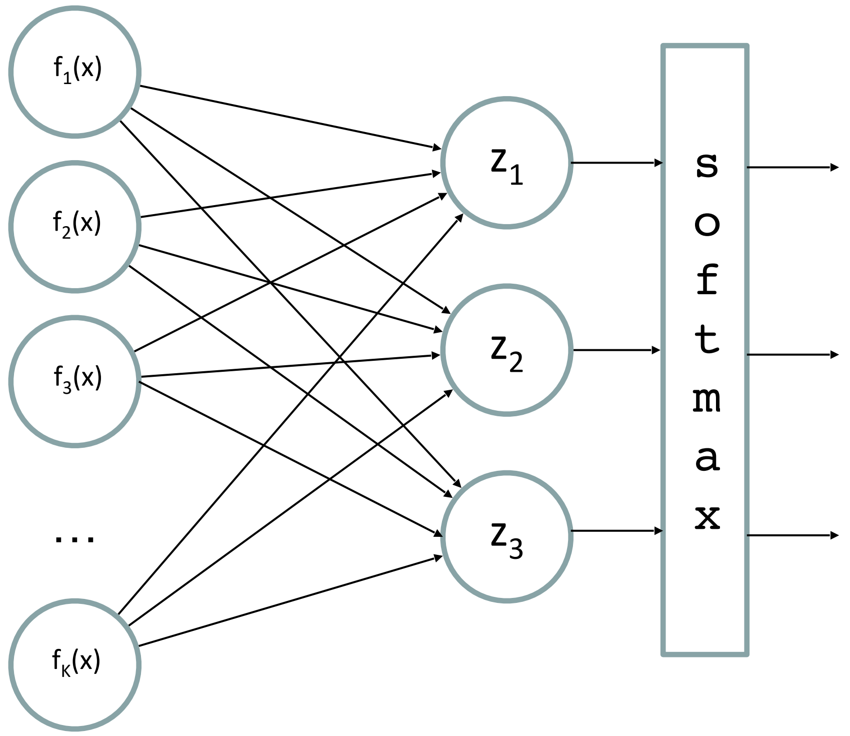


Instructor: Oliver Grillmeyer --- University of California, Berkeley

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Multi-class Logistic Regression

- = special case of neural network

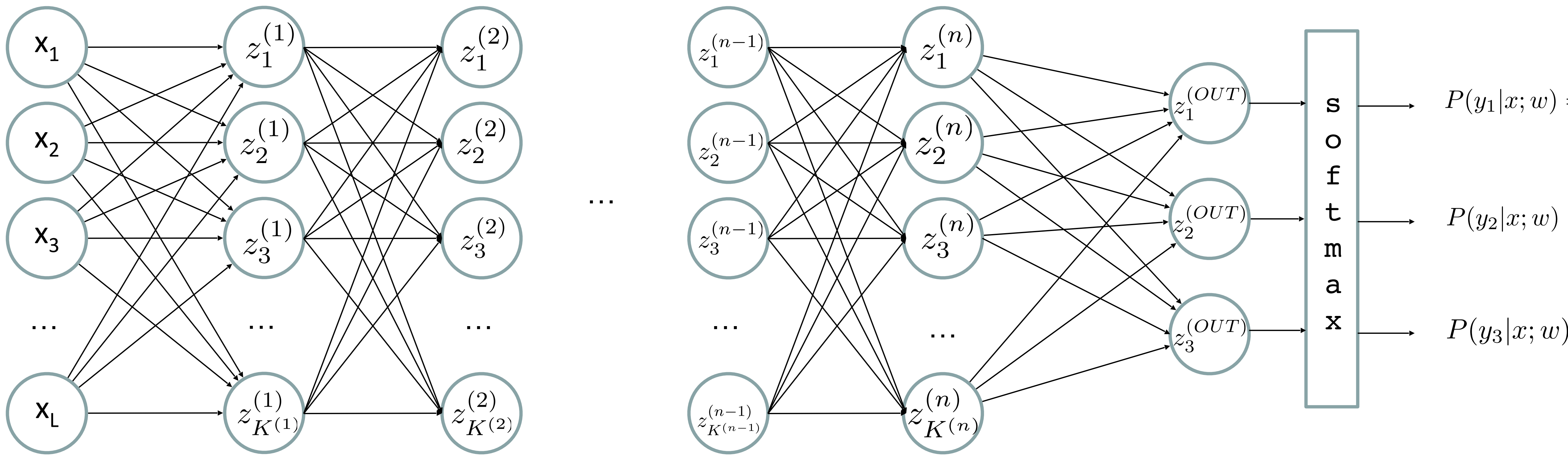


$$P(y_1|x;w) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$P(y_2|x;w) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$P(y_3|x;w) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

Deep Neural Network = Also learn the features!

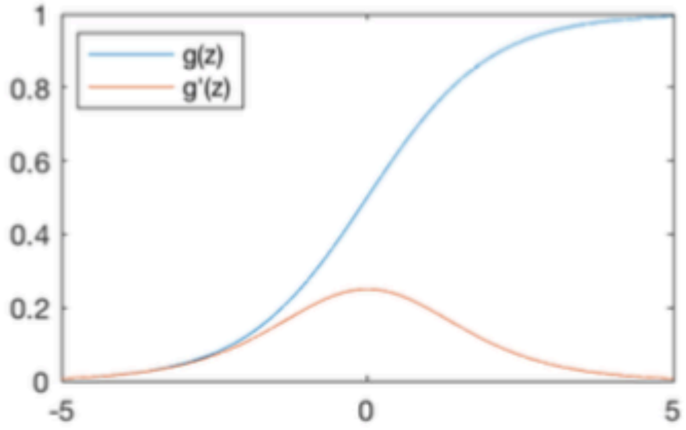


$$z_i^{(k)} = g(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)})$$

g = nonlinear activation function

Common Activation Functions

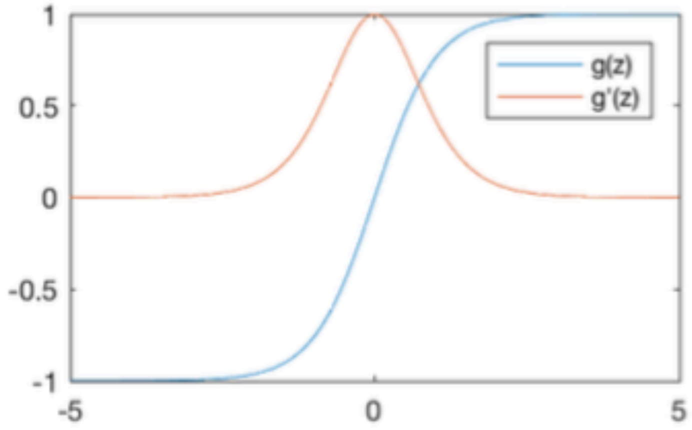
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

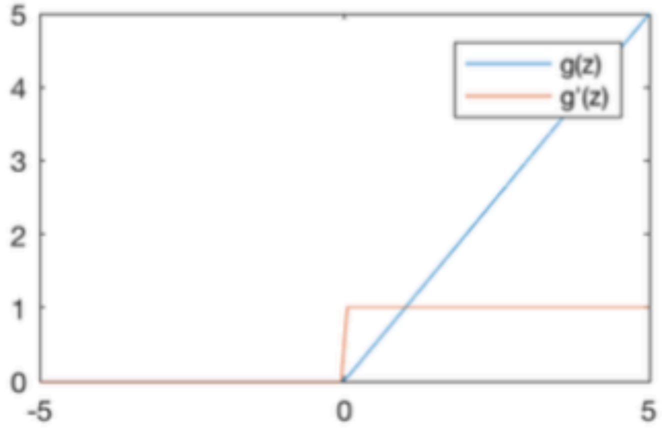
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

[source: MIT 6.S191 introtodeeplearning.com]

Deep Neural Network: Also Learn the Features!

- Training the deep neural network is just like logistic regression:

$$\max_w \; ll(w) = \max_w \; \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

- Much larger weight vector to learn
- Keep training (adjust weights with gradient ascent) until we meet our performance criteria or validation set performance starts decreasing

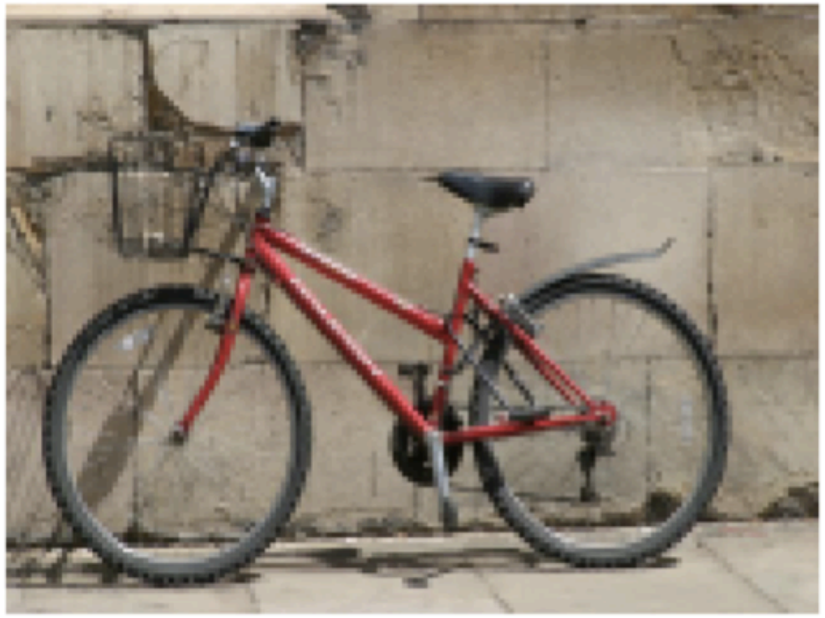
Summary of Key Ideas

- Optimize probability of label given input $\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)};w)$

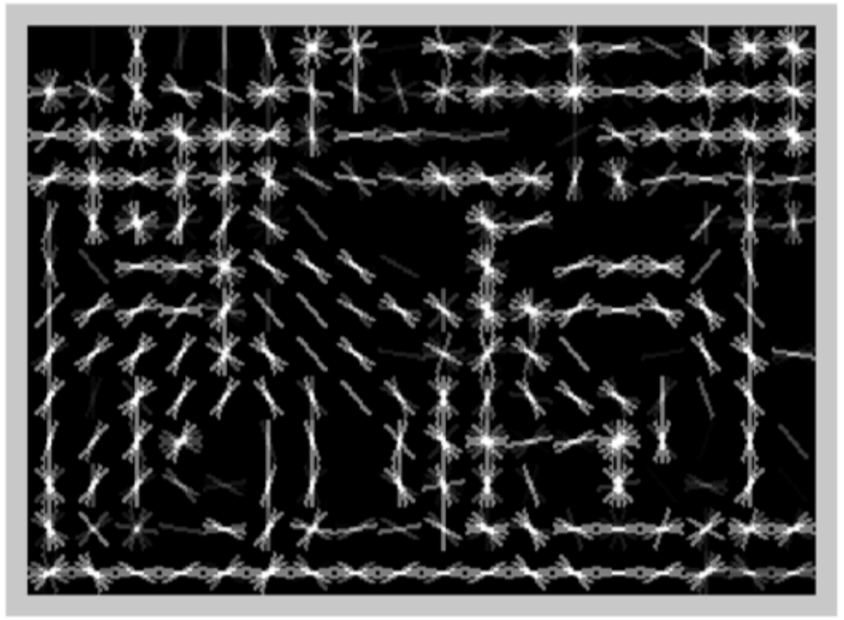
- Continuous optimization
  - Gradient ascent:
    - Compute steepest uphill direction = gradient (= just vector of partial derivatives)
    - Take step in the gradient direction
    - Repeat (until held-out data accuracy starts to drop = "early stopping")

- Deep neural nets
  - Last layer = still logistic regression
  - Now also many more layers before this last layer
    - = computing the features
    - → the features are learned rather than hand-designed
  - Universal function approximation theorem
    - `If` neural net is large enough
    - `Then` neural net can represent any continuous mapping from input to output with arbitrary accuracy
    - But remember: need to avoid overfitting / memorizing the training data → early stopping!
  - Automatic differentiation gives the derivatives efficiently (how? = outside of scope of 188)

Features and Generalization

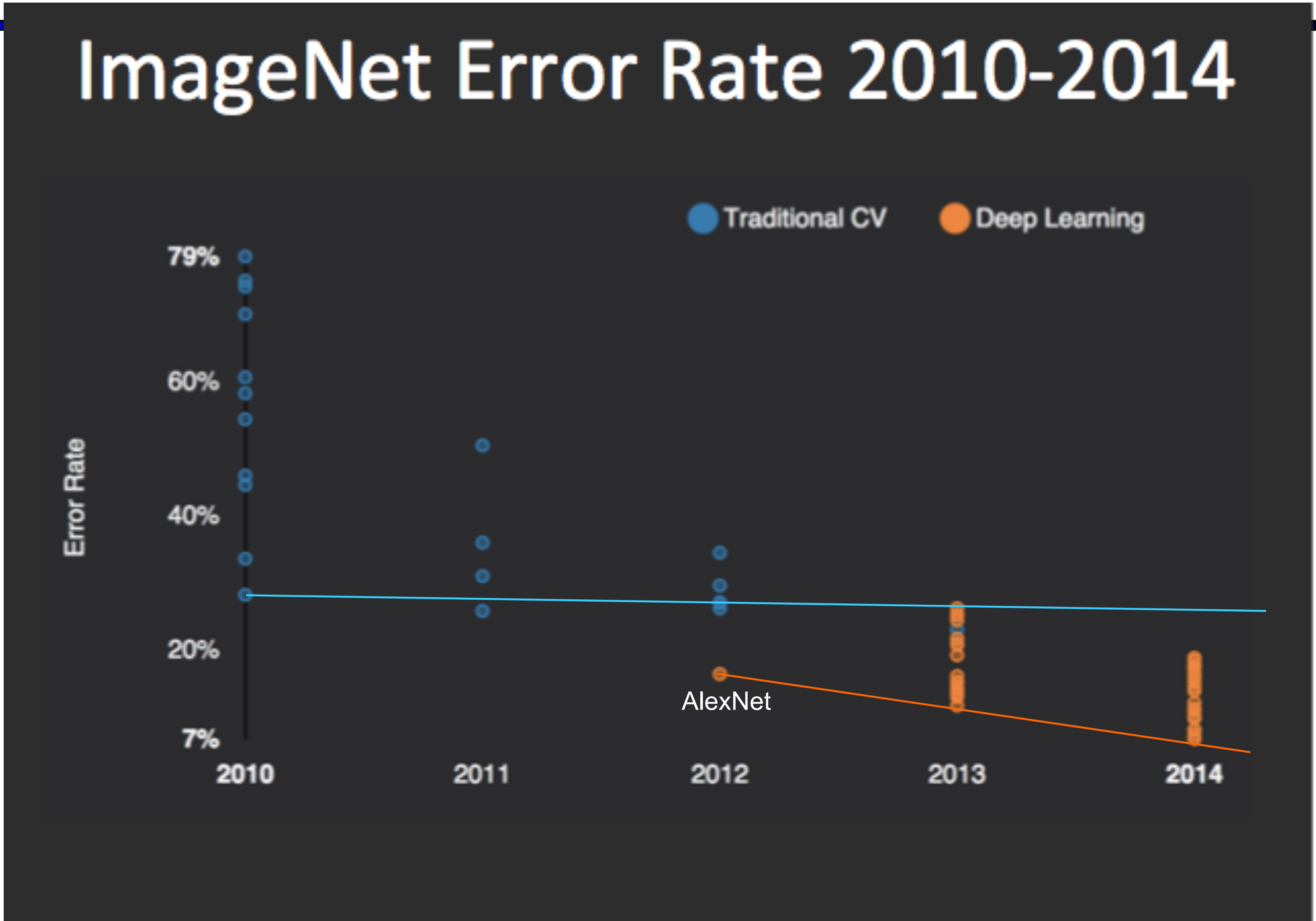


Image



HoG

Performance



graph credit Matt Zeiler, Clarifai

CS 188: Artificial Intelligence

Neural Nets (wrap-up) and Decision Trees

Instructor: Oliver Grillmeyer --- University of California, Berkeley

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

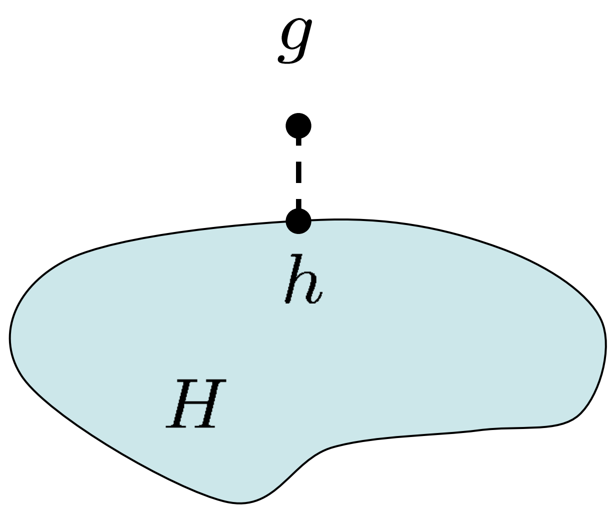
Inductive Learning (Science)

- Simplest form: learn a function from examples
  - A target function: g
  - Examples: input-output pairs $(x, g(x))$
  - E.g. x is an email and $g(x)$ is spam / ham
  - E.g. x is a house and $g(x)$ is its selling price

- Problem:
  - Given a hypothesis space H
  - Given a training set of examples x_i
  - Find a hypothesis $h(x)$ such that $h \sim g$

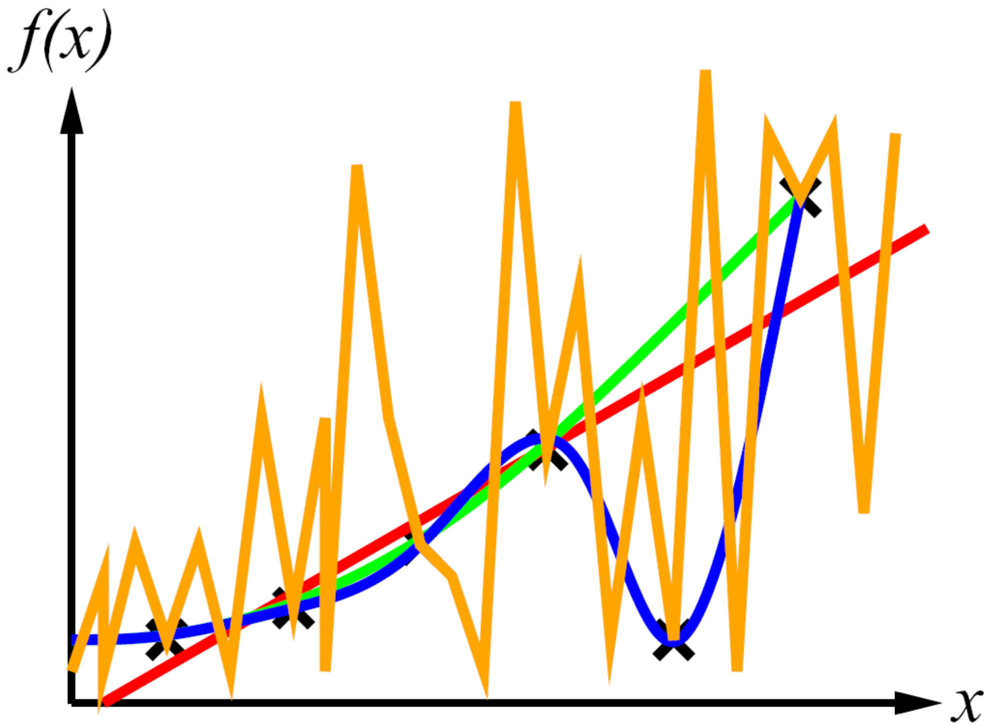
- Includes:
  - Classification (outputs = class labels)
  - Regression (outputs = real numbers)

- How do perceptron and naïve Bayes fit in? $(H, h, g$, etc.)



Inductive Learning

- Curve fitting (regression, function approximation):



- Consistency vs. simplicity
- Ockham's razor

Features

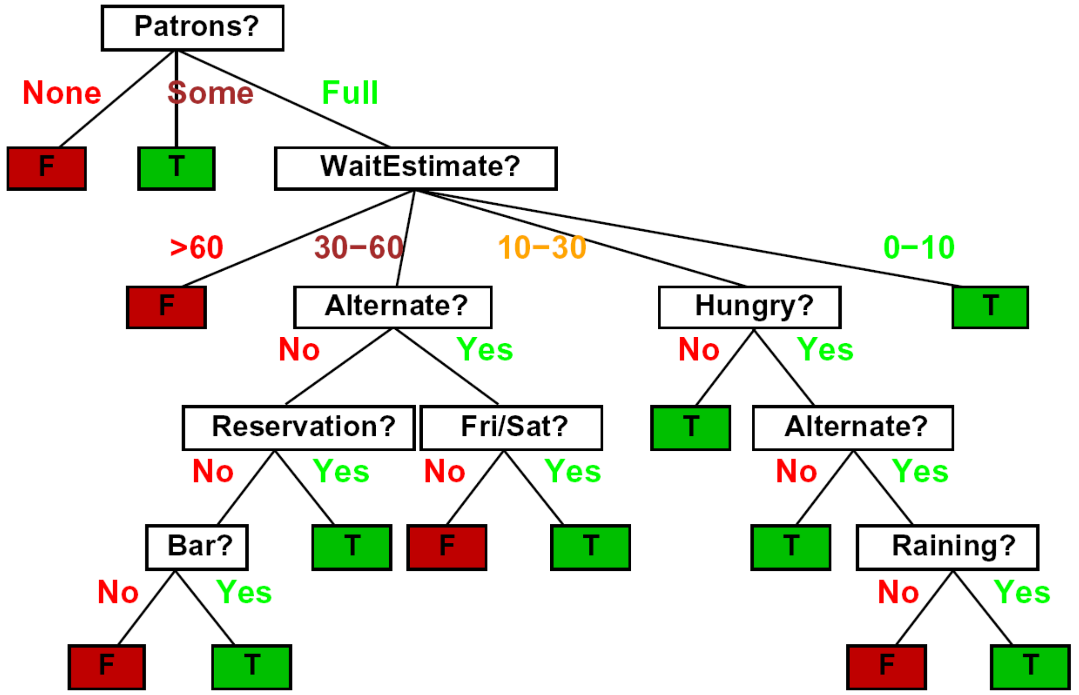
- Features, aka attributes
  - Sometimes: TYPE=French
  - Sometimes: $f_{\text{TYPE=French}}(x) = 1$

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0–10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30–60	T

Decision Trees

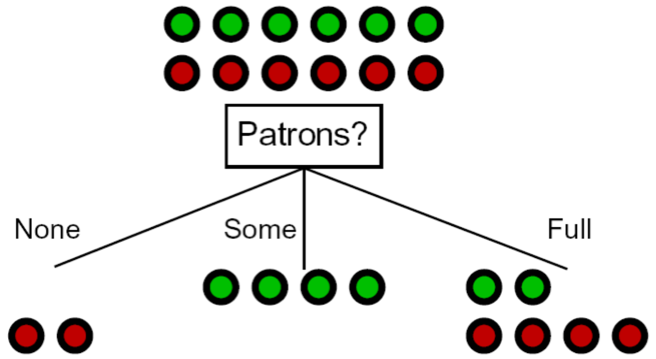
- Compact representation of a function:
  - Truth table
  - Conditional probability table
  - Regression values

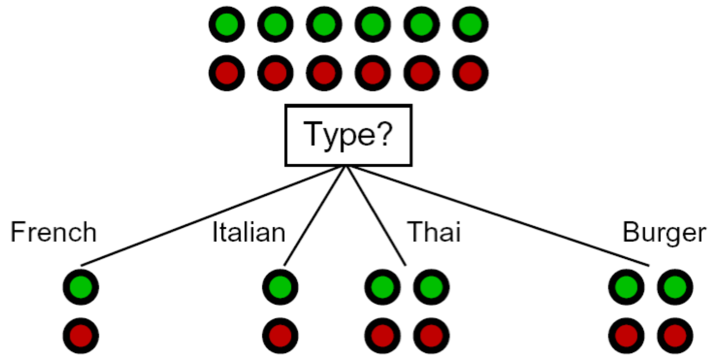
- True function
  - Realizable: in H



Choosing an Attribute

- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"





- So: we need a measure of how "good" a split is, even if the results aren't perfectly separated out

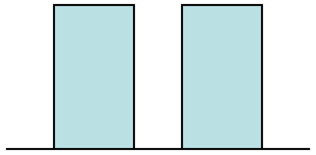
Entropy

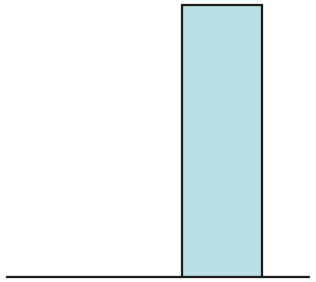
- General answer: if prior is $\langle p_1, \ldots, p_n \rangle$:
  - Information is the expected code length

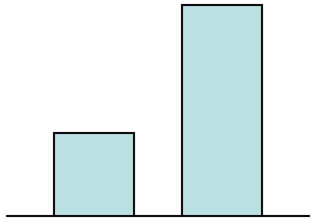
$$H(\langle p_1, \ldots, p_n \rangle) = E_p \log_2 1/p_i$$

$$= \sum_{i=1}^{n} -p_i \log_2 p_i$$

- Also called the entropy of the distribution
  - More uniform = higher entropy
  - More values = higher entropy
  - More peaked = lower entropy
  - Rare values almost "don't count"

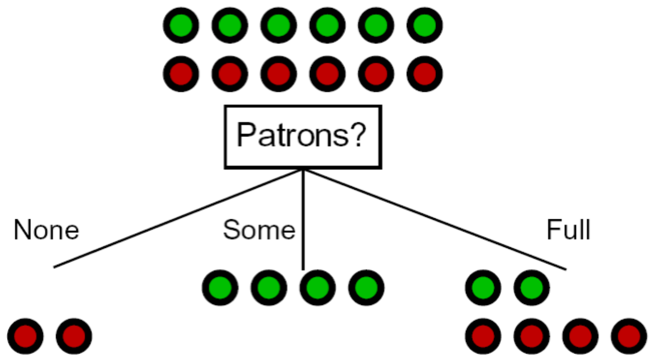

1 bit

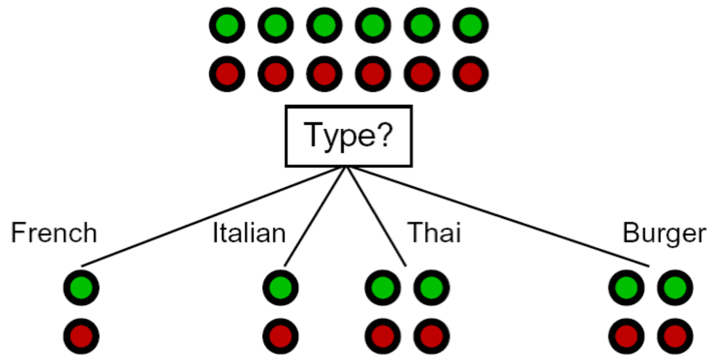

0 bits


0.5 bit

Information Gain

- Back to decision trees!
- For each split, compare entropy before and after
  - Difference is the information gain
  - Problem: there's more than one distribution after split!

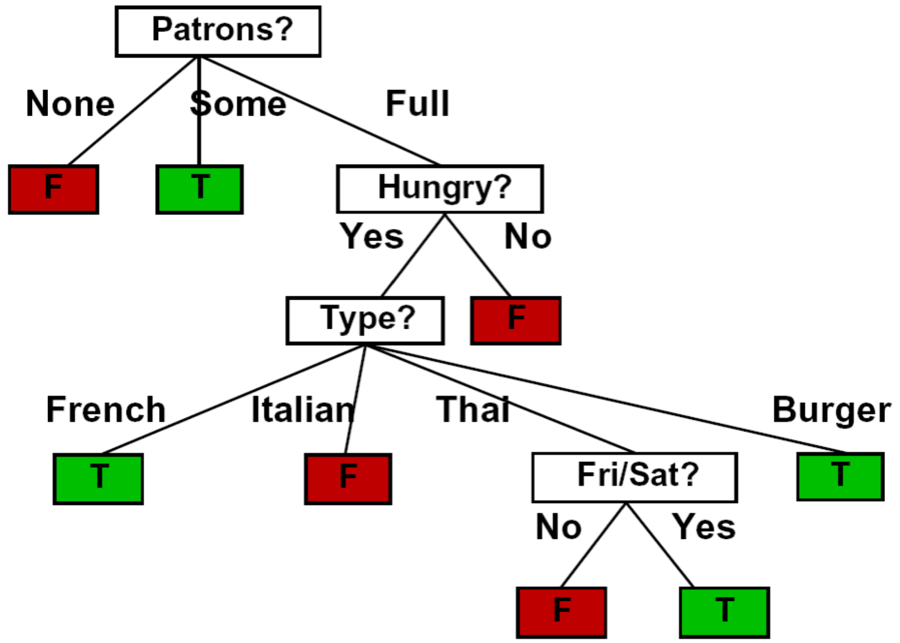




  - Solution: use expected entropy, weighted by the number of examples

Example: Learned Tree

- Decision tree learned from these 12 examples:



- Substantially simpler than "true" tree
  - A more complex hypothesis isn't justified by data
- Also: it's reasonable, but wrong

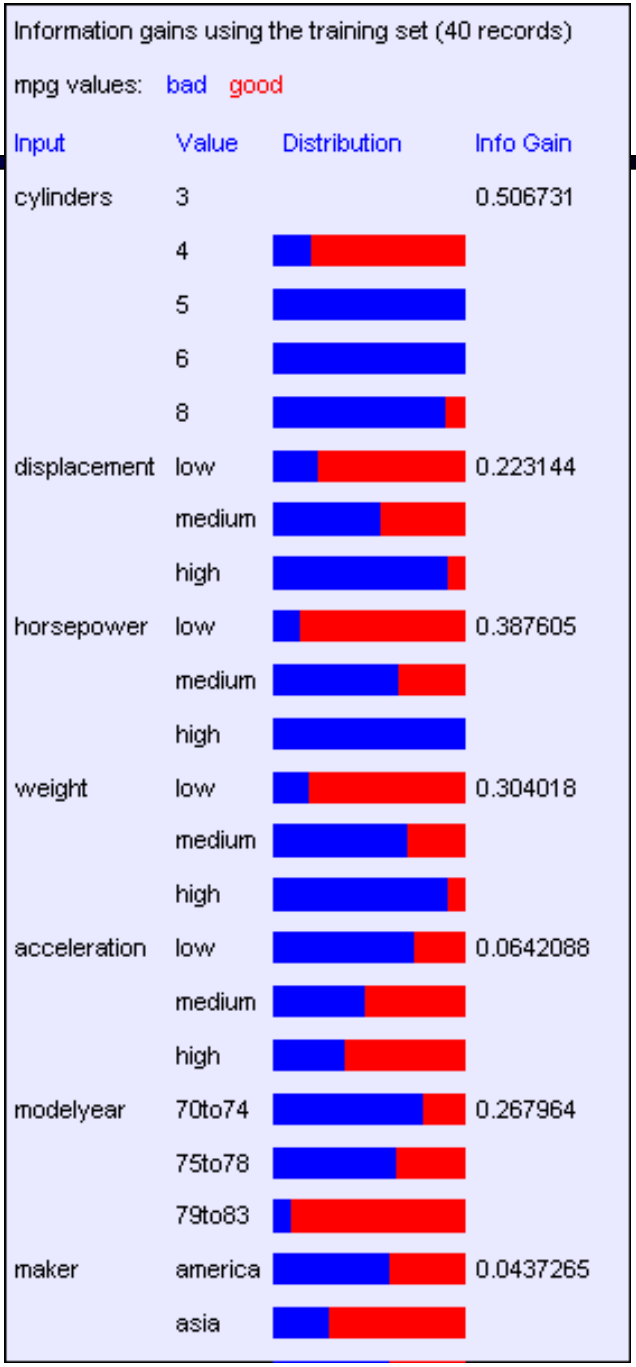
Example: Miles Per Gallon

40 Examples

mpg	cylinders	displacement	horsepower	weight	acceleration	modelyear	maker
good	4	low	low	low	high	75to78	asia
bad	6	medium	medium	medium	medium	70to74	america
bad	4	medium	medium	medium	low	75to78	europe
bad	8	high	high	high	low	70to74	america
bad	6	medium	medium	medium	medium	70to74	america
bad	4	low	medium	low	medium	70to74	asia
bad	4	low	medium	low	low	70to74	asia
bad	8	high	high	high	low	75to78	america
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
bad	8	high	high	high	low	70to74	america
good	8	high	medium	high	high	79to83	america
bad	8	high	high	high	low	75to78	america
good	4	low	low	low	low	79to83	america
bad	6	medium	medium	medium	high	75to78	america
good	4	medium	low	low	low	79to83	america
good	4	low	low	medium	high	79to83	america
bad	8	high	high	high	low	70to74	america
good	4	low	medium	low	medium	75to78	europe
bad	5	medium	medium	medium	medium	75to78	europe

Find the First Split

- Look at information gain for each attribute
- Note that each attribute is correlated with the target!
- What do we split on?

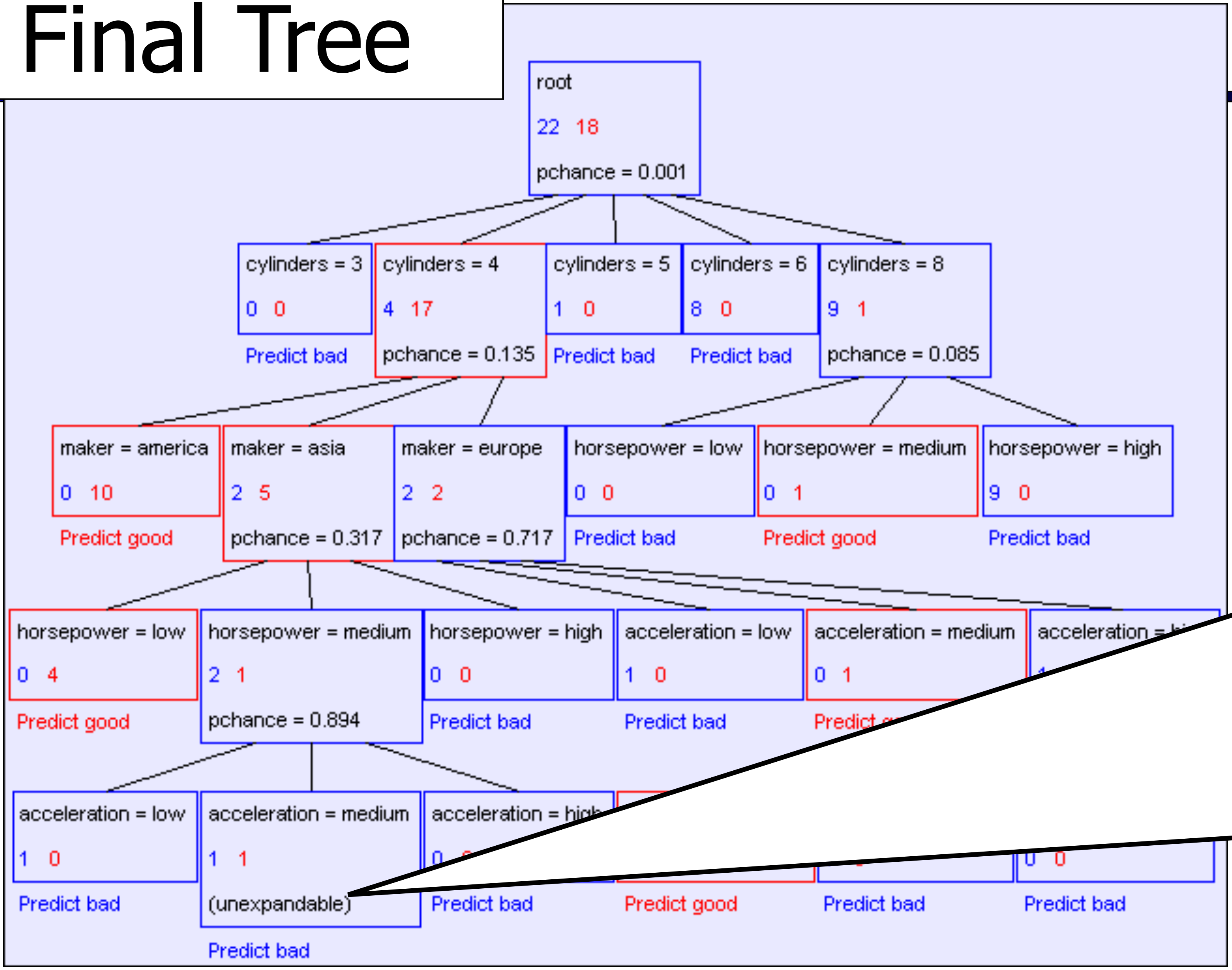


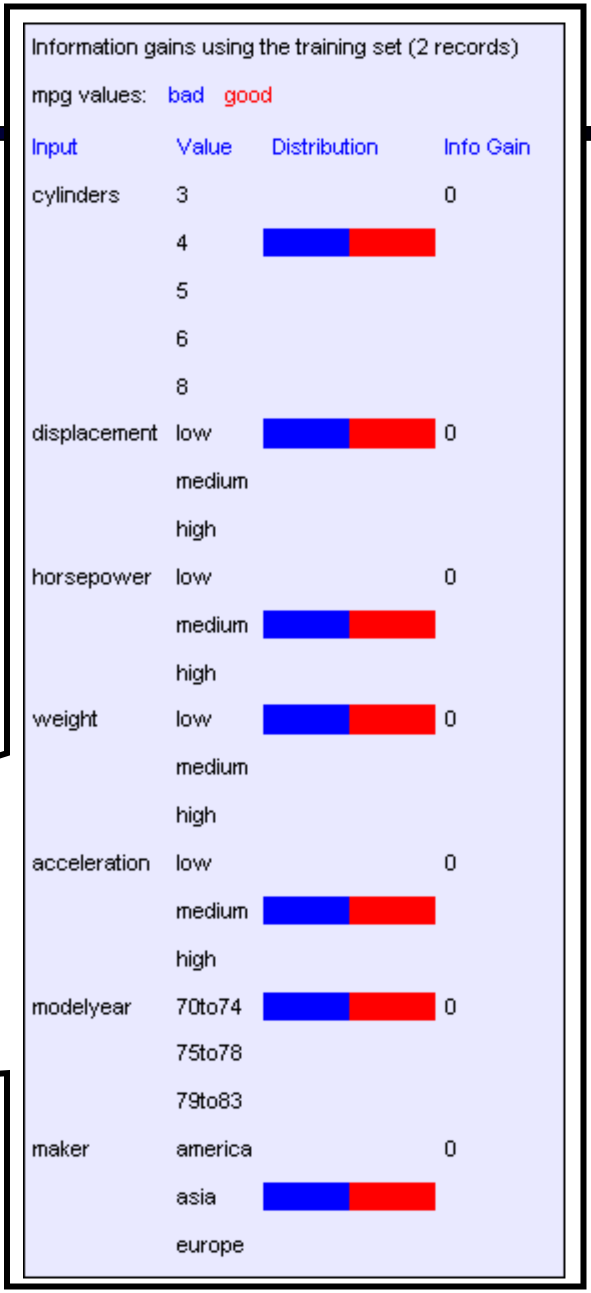
Information gains using the training set (40 records)

mpg values: bad good

Input	Value	Distribution	Info Gain
cylinders	3		0.506731
	4		
	5		
	6		
	8		
displacement	low		0.223144
	medium		
	high		
horsepower	low		0.387605
	medium		
	high		
weight	low		0.304018
	medium		
	high		
acceleration	low		0.0642088
	medium		
	high		
modelyear	70to74		0.267964
	75to78		
	79to83		
maker	america		0.0437265
	asia		

Final Tree





Information gains using the training set (2 records)

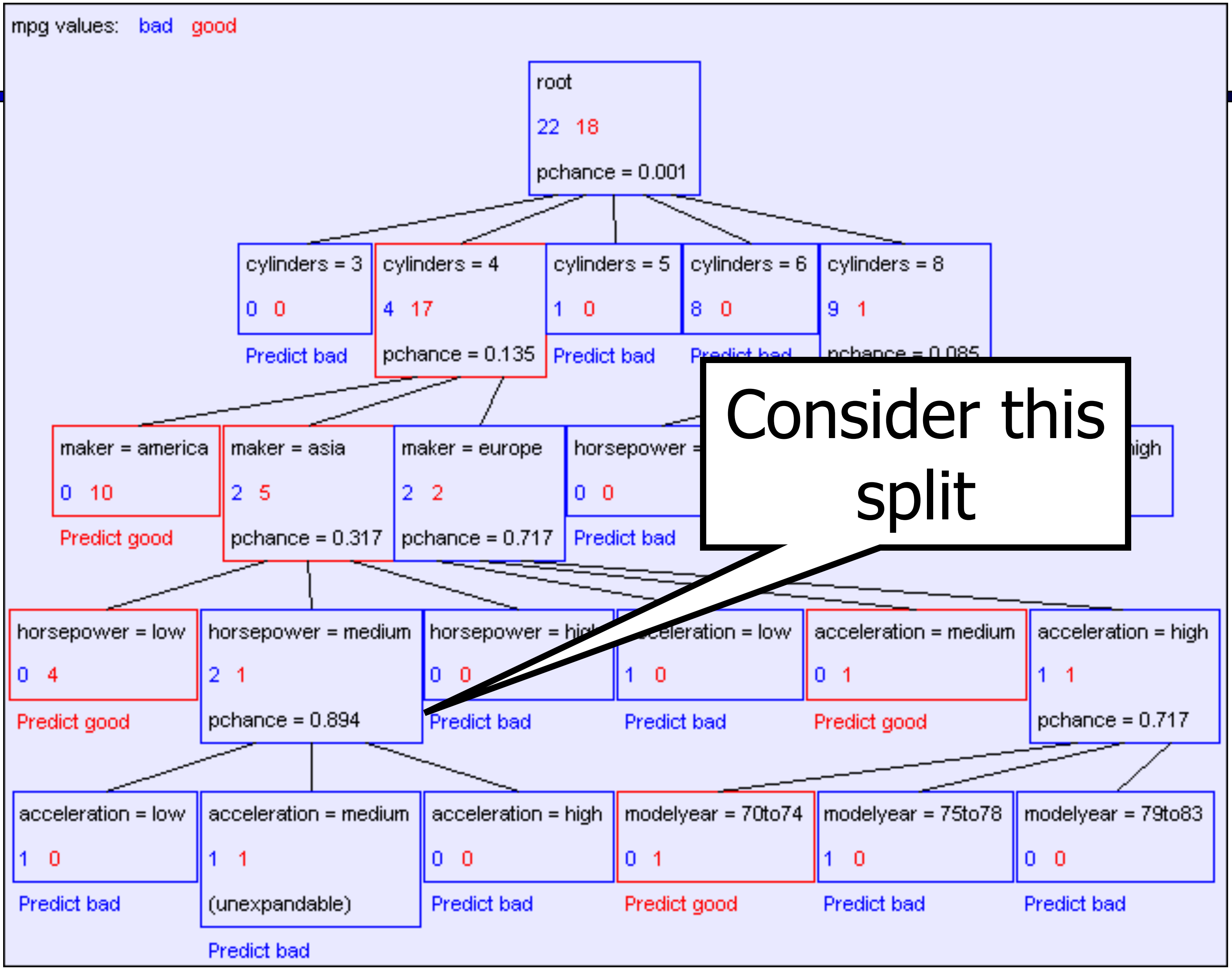
mpg values: bad good

Input	Value	Distribution	Info Gain
cylinders	3		0
	4	🟦🟥	
	5		
	6		
	8		
displacement	low	🟦🟥	0
	medium		
	high		
horsepower	low		0
	medium	🟦🟥	
	high		
weight	low	🟦🟥	0
	medium		
	high		
acceleration	low		0
	medium	🟦🟥	
	high		
modelyear	70to74	🟦🟥	0
	75to78		
	79to83		
maker	america		0
	asia	🟦🟥	
	europe		

Reminder: Overfitting

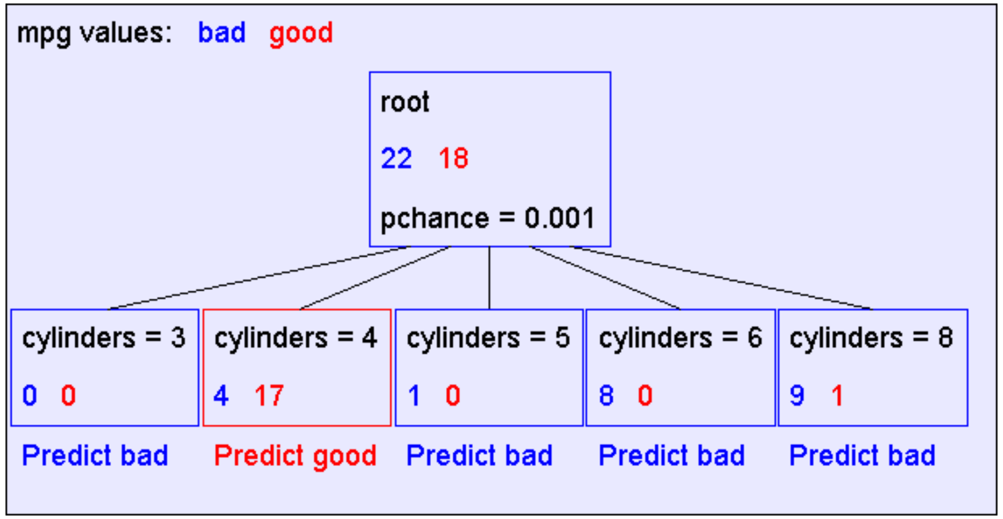
- Overfitting:
  - When you stop modeling the patterns in the training data (which generalize)
  - And start modeling the noise (which doesn’t)

- We had this before:
  - Naïve Bayes: needed to smooth
  - Perceptron: early stopping

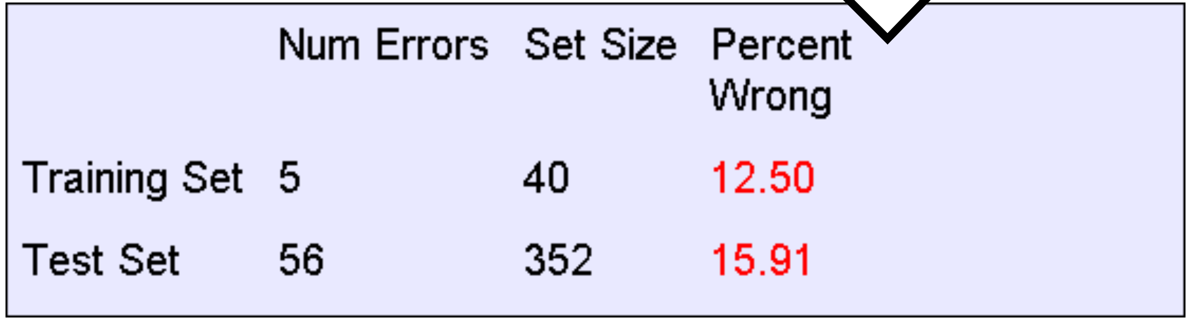

mpg values: bad good
root
22 18
pchance = 0.001
cylinders = 3
0 0
Predict bad
cylinders = 4
4 17
pchance = 0.135
cylinders = 5
1 0
Predict bad
cylinders = 6
8 0
Predict bad
cylinders = 8
9 1
pchance = 0.085
maker = america
0 10
Predict good
maker = asia
2 5
pchance = 0.317
maker = europe
2 2
pchance = 0.717
horsepower =
0 0
Predict bad
high
horsepower = low
0 4
Predict good
horsepower = medium
2 1
pchance = 0.894
horsepower = high
0 0
Predict bad
acceleration = low
1 0
Predict bad
acceleration = medium
0 1
Predict good
acceleration = high
1 1
pchance = 0.717
acceleration = low
1 0
Predict bad
acceleration = medium
1 1
(unexpandable)
Predict bad
acceleration = high
0 0
Predict bad
modelyear = 70to74
0 1
Predict good
modelyear = 75to78
1 0
Predict bad
modelyear = 79to83
0 0
Predict bad
Consider this split

Pruning example

- With $\text{MaxP}_{\text{CHANCE}} = 0.1$:



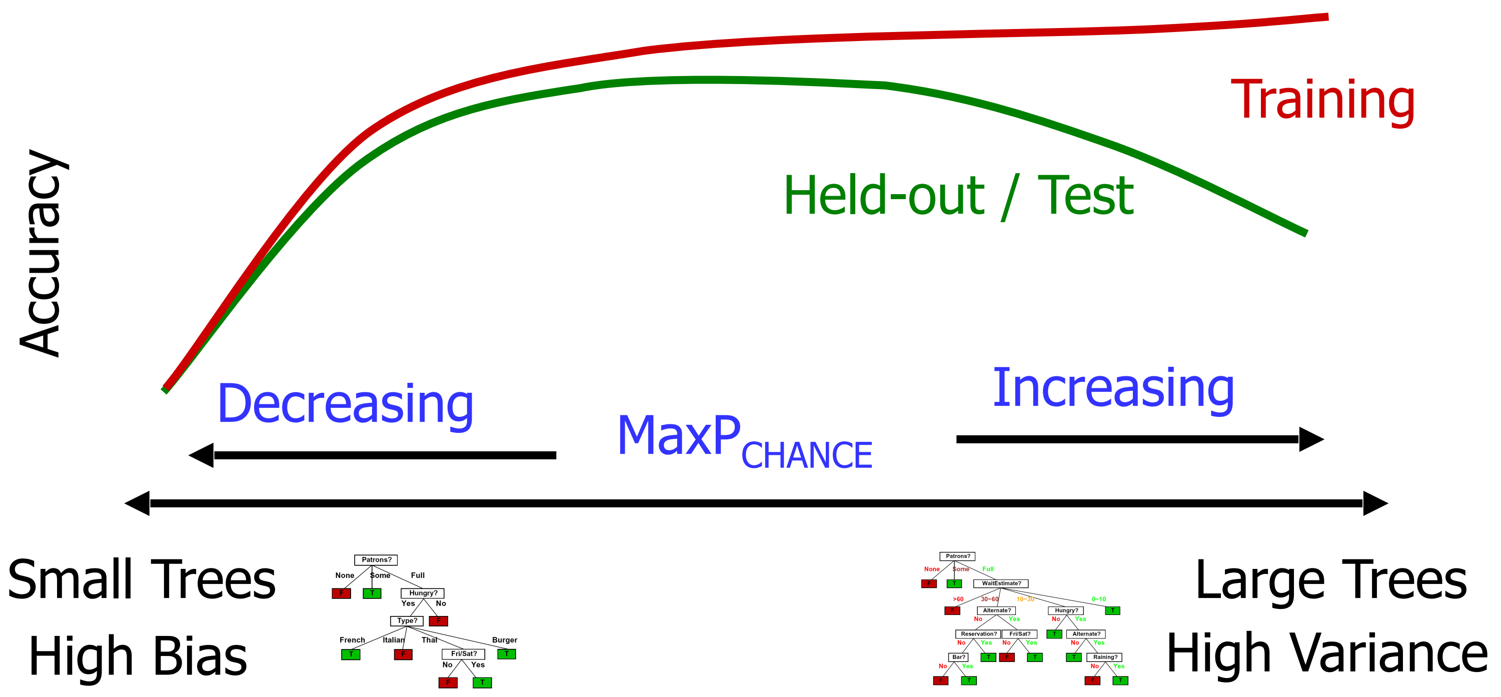
Note the improved test set accuracy compared with the unpruned tree



	Num Errors	Set Size	Percent Wrong
Training Set	5	40	12.50
Test Set	56	352	15.91

Regularization

- $\mathrm{MaxP}_{\mathrm{CHANCE}}$ is a regularization parameter
- Generally, set it using held-out data (as usual)

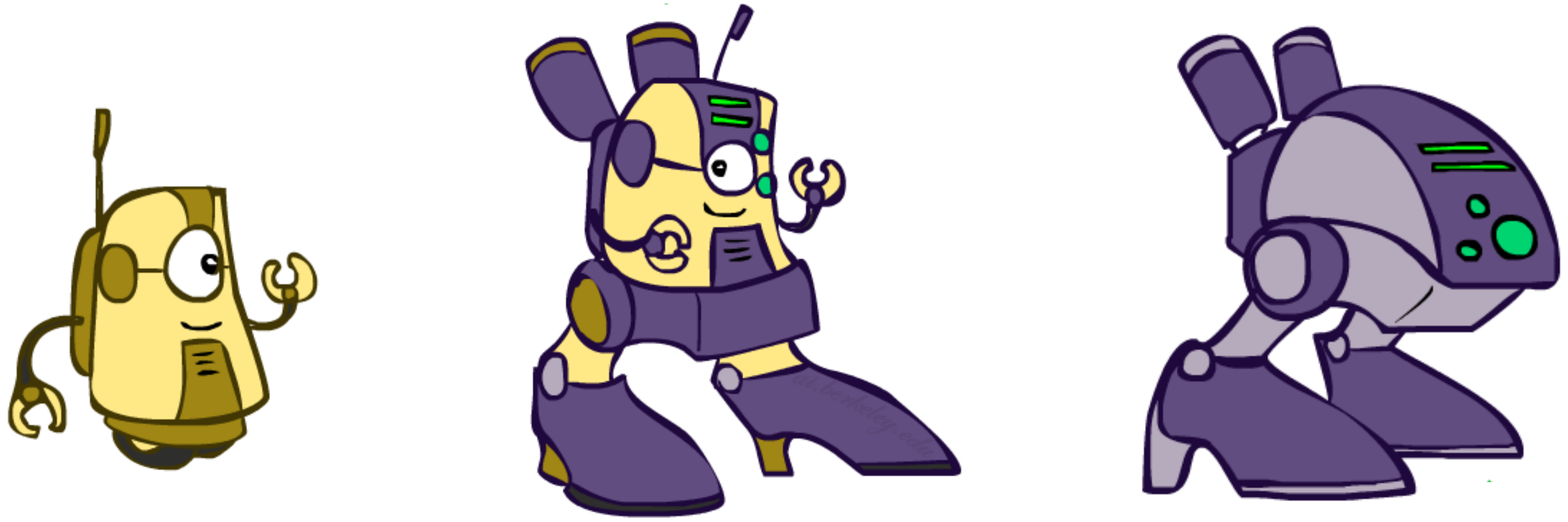


Two Ways of Controlling Overfitting

- Limit the hypothesis space
  - E.g. limit the max depth of trees
  - Easier to analyze

- Regularize the hypothesis selection
  - E.g. chance cutoff
  - Skip most of the hypotheses unless data is clear
  - Usually done in practice

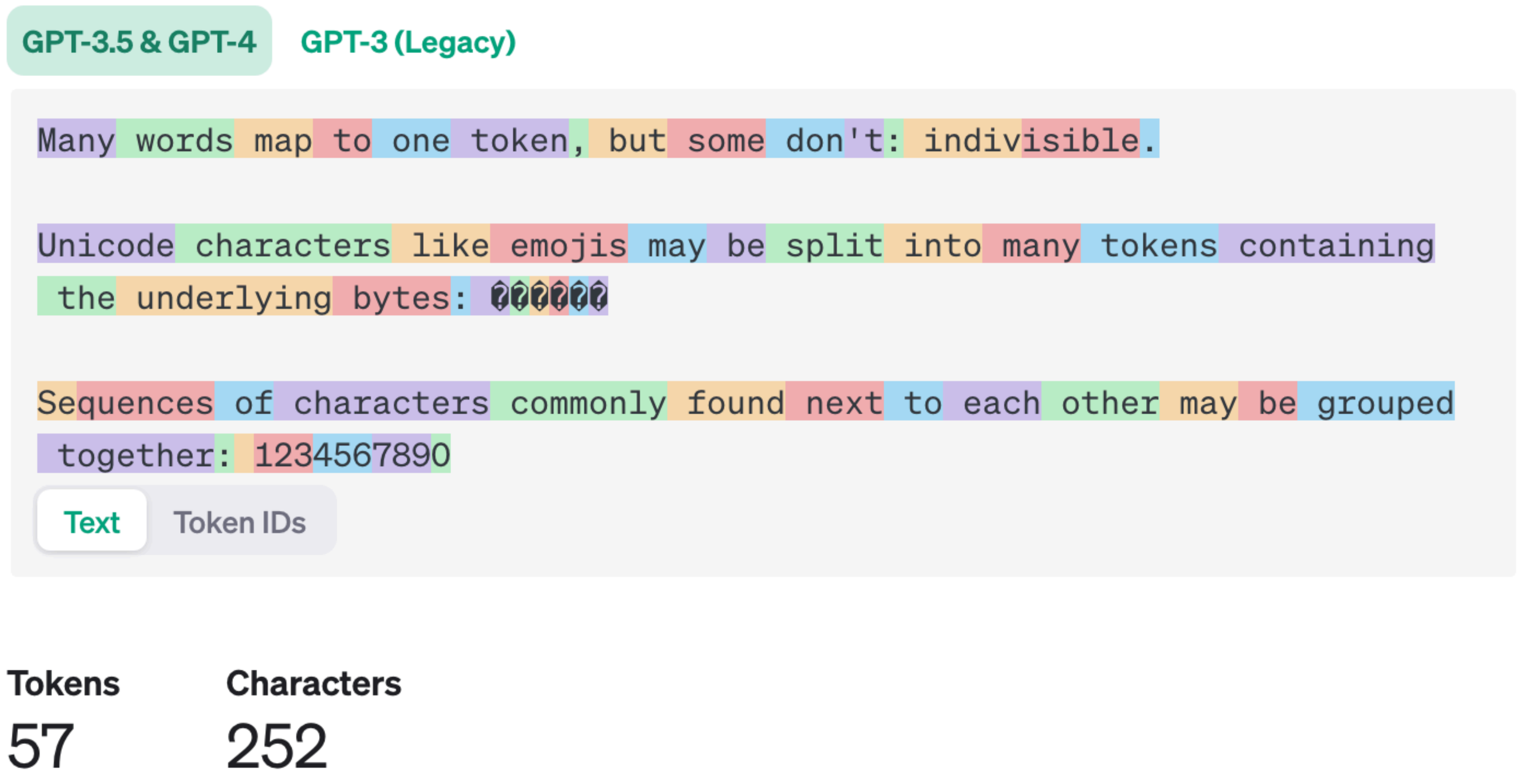
Large Language Model Transformers



Large Language Models

- Feature engineering
  - Text tokenization
  - Word embeddings
- Deep neural networks
  - Autoregressive models
  - Self-attention mechanisms
  - Transformer architecture
- Multi-class classification
- Supervised learning
  - Self-supervised learning
  - Instruction tuning
- Reinforcement learning
  - … from human feedback (RLHF)
- Policy search
  - Policy gradient methods
- Beam search

Text Tokenization


GPT-3.5 & GPT-4 GPT-3 (Legacy)

Many words map to one token, but some don't: indivisible.

Unicode characters like emojis may be split into many tokens containing the underlying bytes: ������

Sequences of characters commonly found next to each other may be grouped together: 1234567890

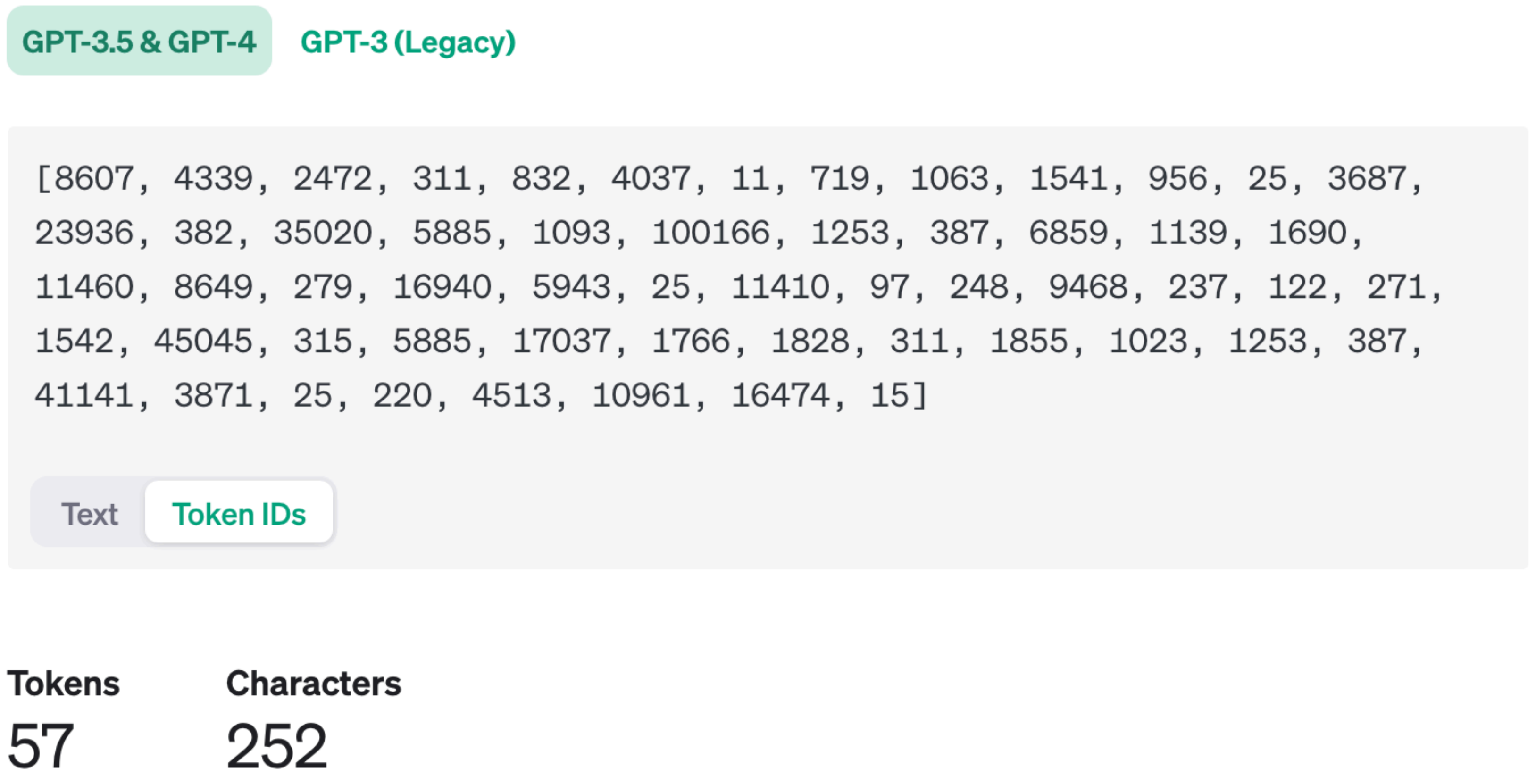
Text Token IDs

Tokens
57

Characters
252

https://platform.openai.com/tokenizer

Text Tokenization



```
[8607, 4339, 2472, 311, 832, 4037, 11, 719, 1063, 1541, 956, 25, 3687,
23936, 382, 35020, 5885, 1093, 100166, 1253, 387, 6859, 1139, 1690,
11460, 8649, 279, 16940, 5943, 25, 11410, 97, 248, 9468, 237, 122, 271,
1542, 45045, 315, 5885, 17037, 1766, 1828, 311, 1855, 1023, 1253, 387,
41141, 3871, 25, 220, 4513, 10961, 16474, 15]
```

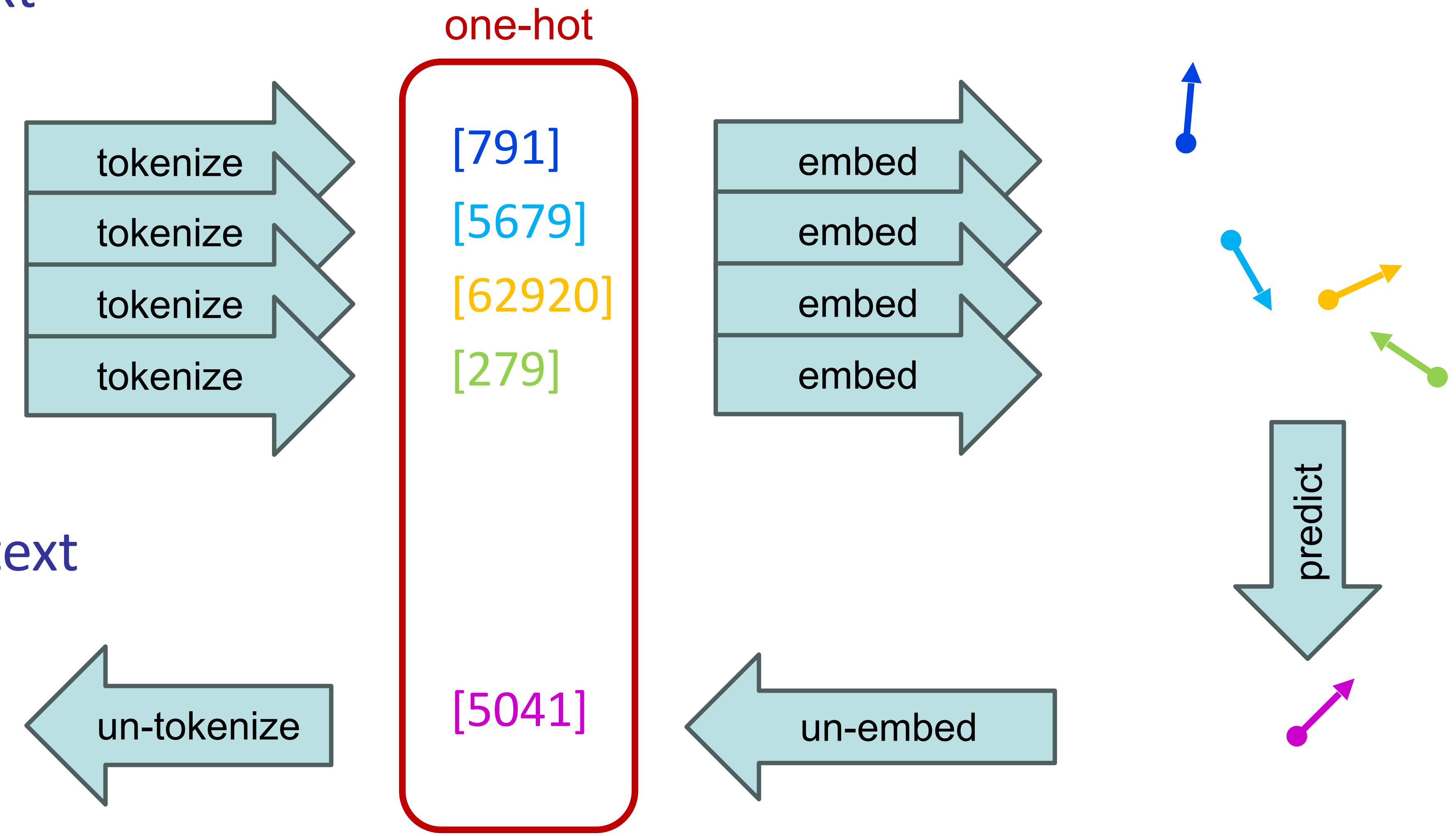
Tokens
57

Characters
252

https://platform.openai.com/tokenizer

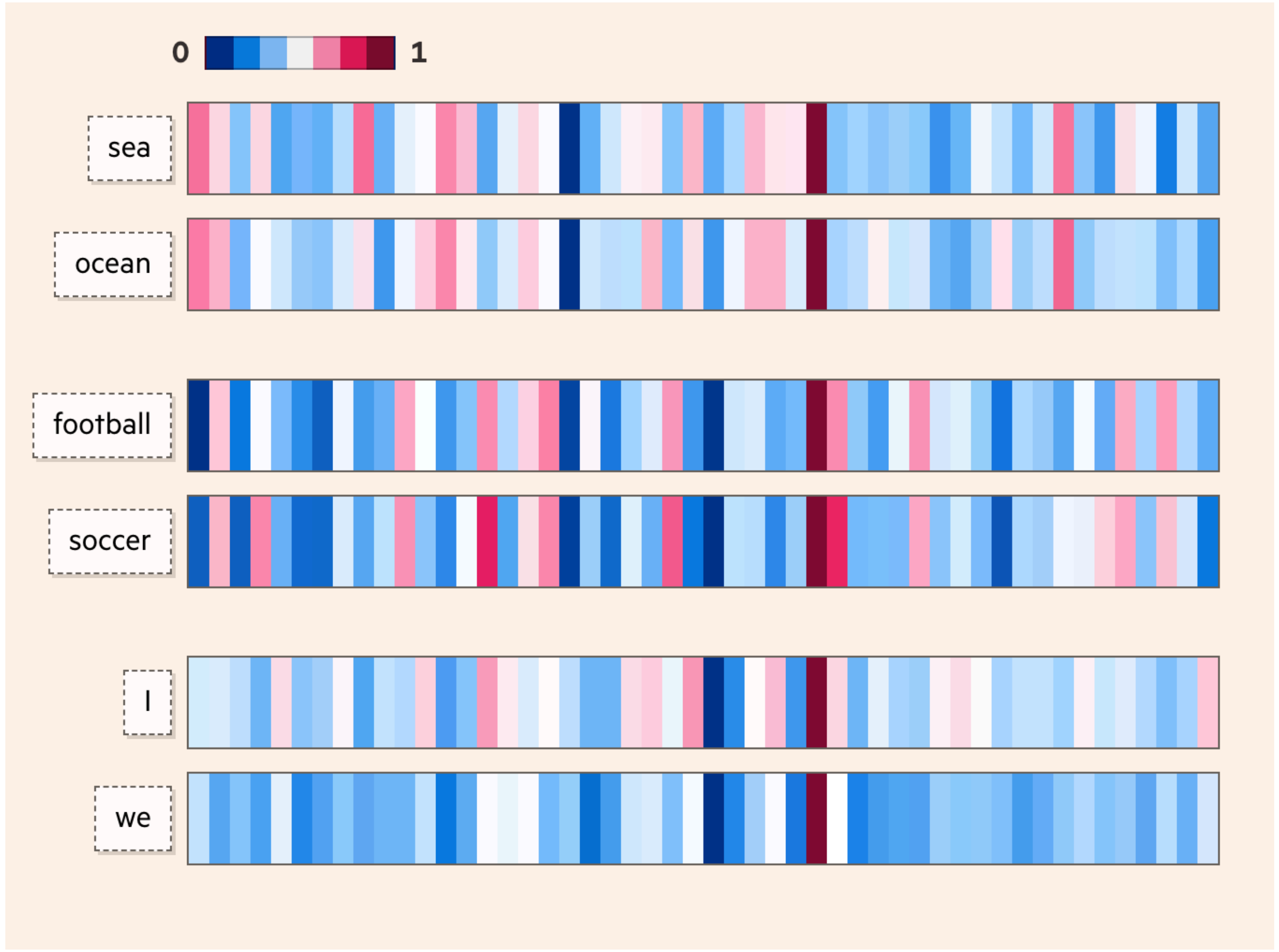
Word Embeddings

- Input: some text
  - "The"
  - " dog"
  - " chased"
  - " the"
- Output: more text
  - " ball"



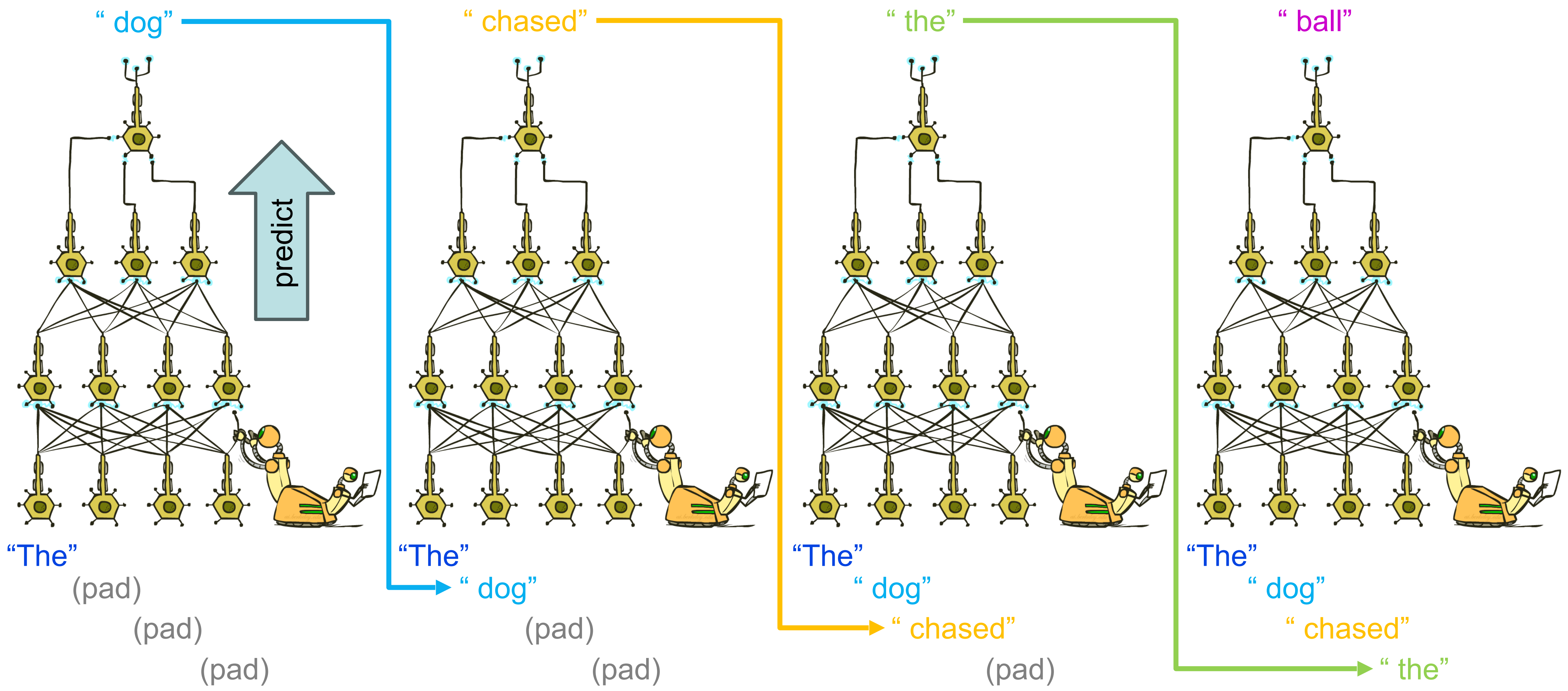
What do word embeddings look like?

- Features learned in language models:

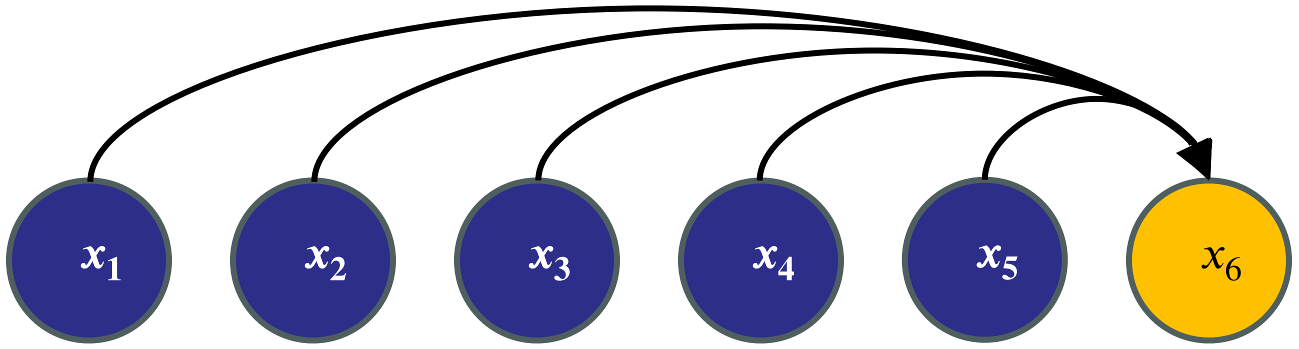


ig.ft.com/generative-ai

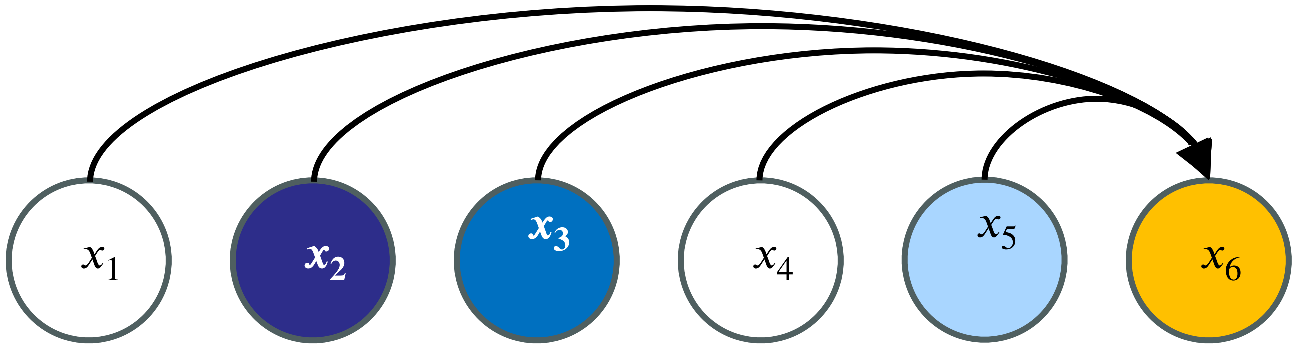
Autoregressive Models



Self-Attention Mechanisms

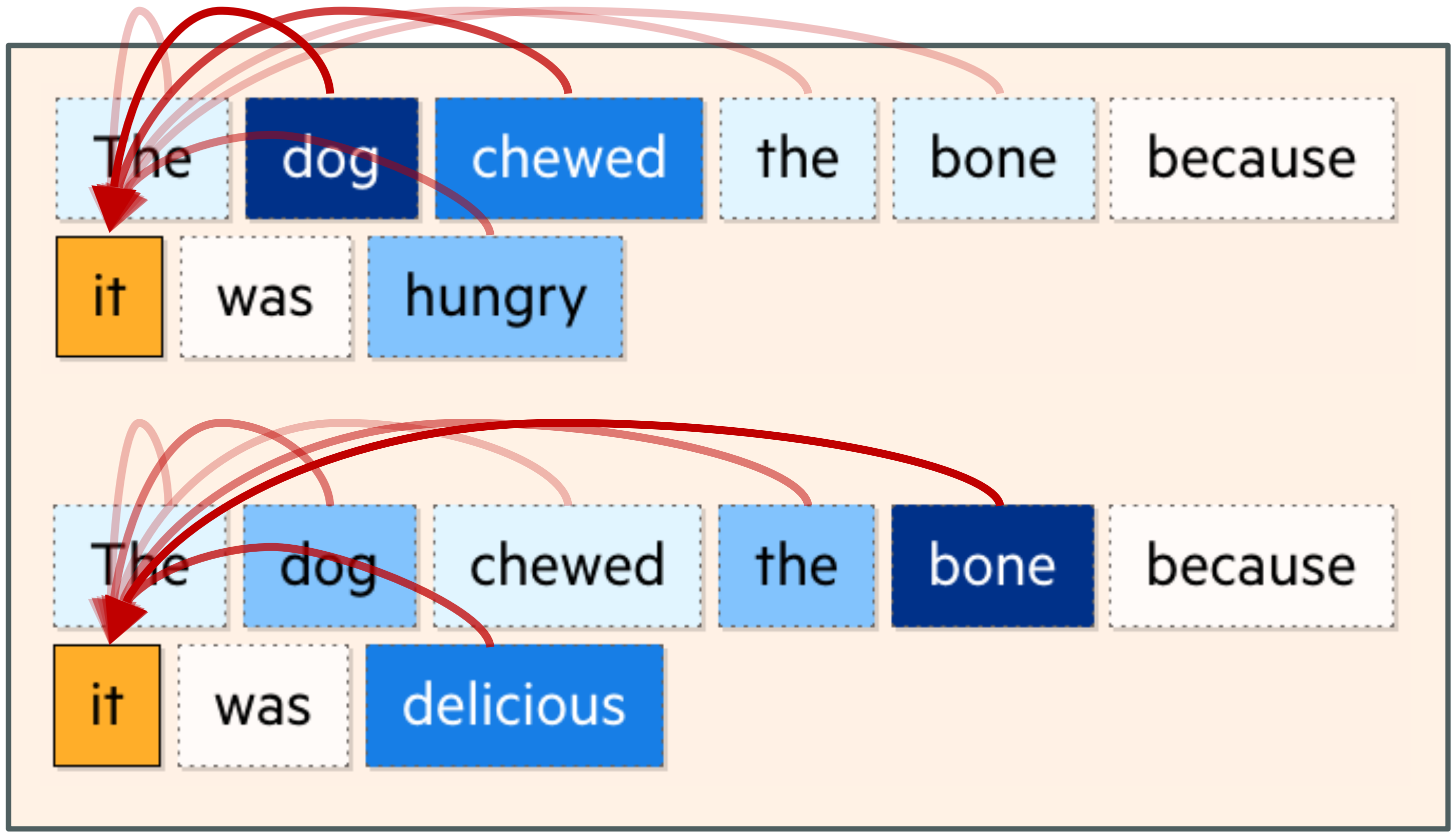


- Instead of conditioning on *all* input tokens equally…

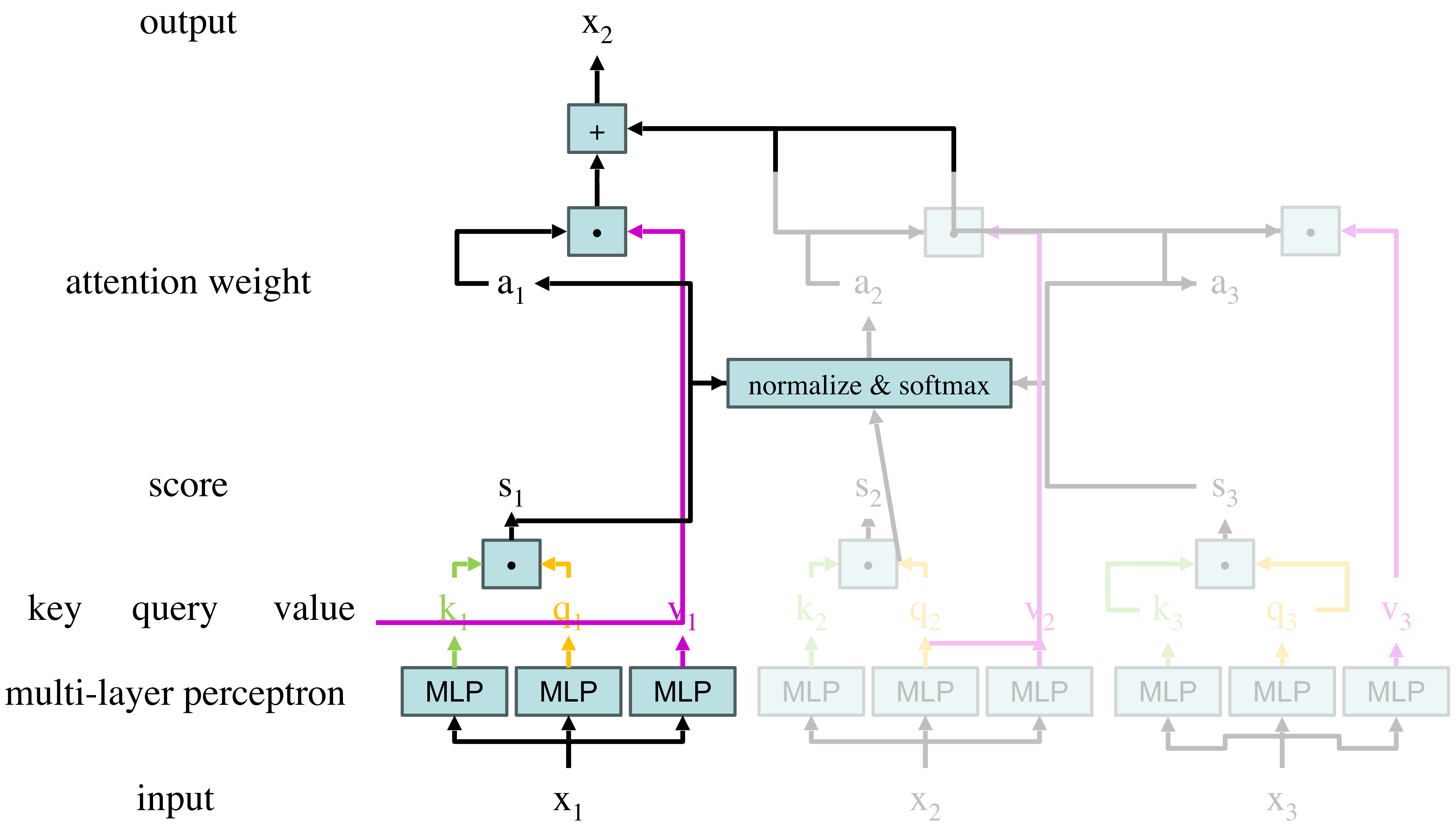


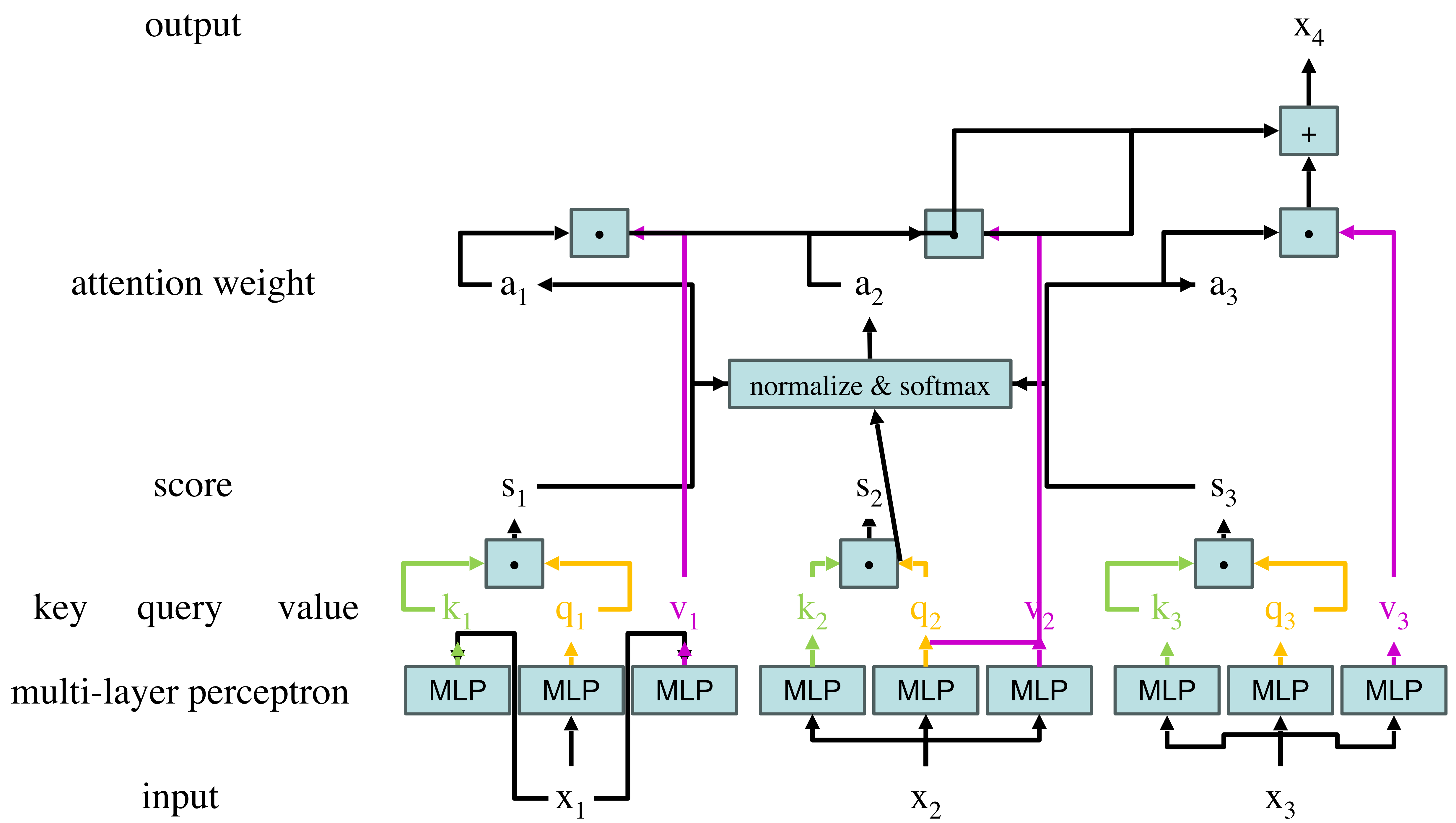
- Pay more attention to relevant tokens!

Self-Attention Mechanisms



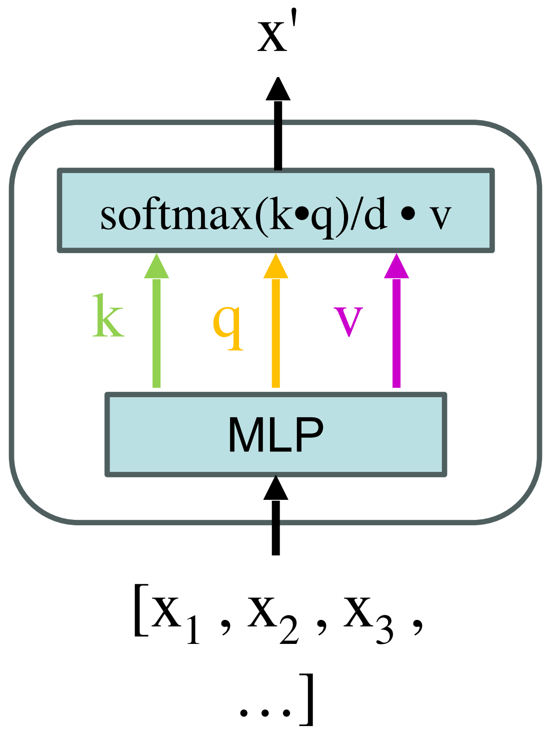
ig.ft.com/generative-ai


output
x_2
+
attention weight
a_1
a_2
a_3
normalize & softmax
score
s_1
s_2
s_3
key
query
value
k_1
q_1
v_1
k_2
q_2
v_2
k_3
q_3
v_3
multi-layer perceptron
MLP
input
x_1
x_2
x_3

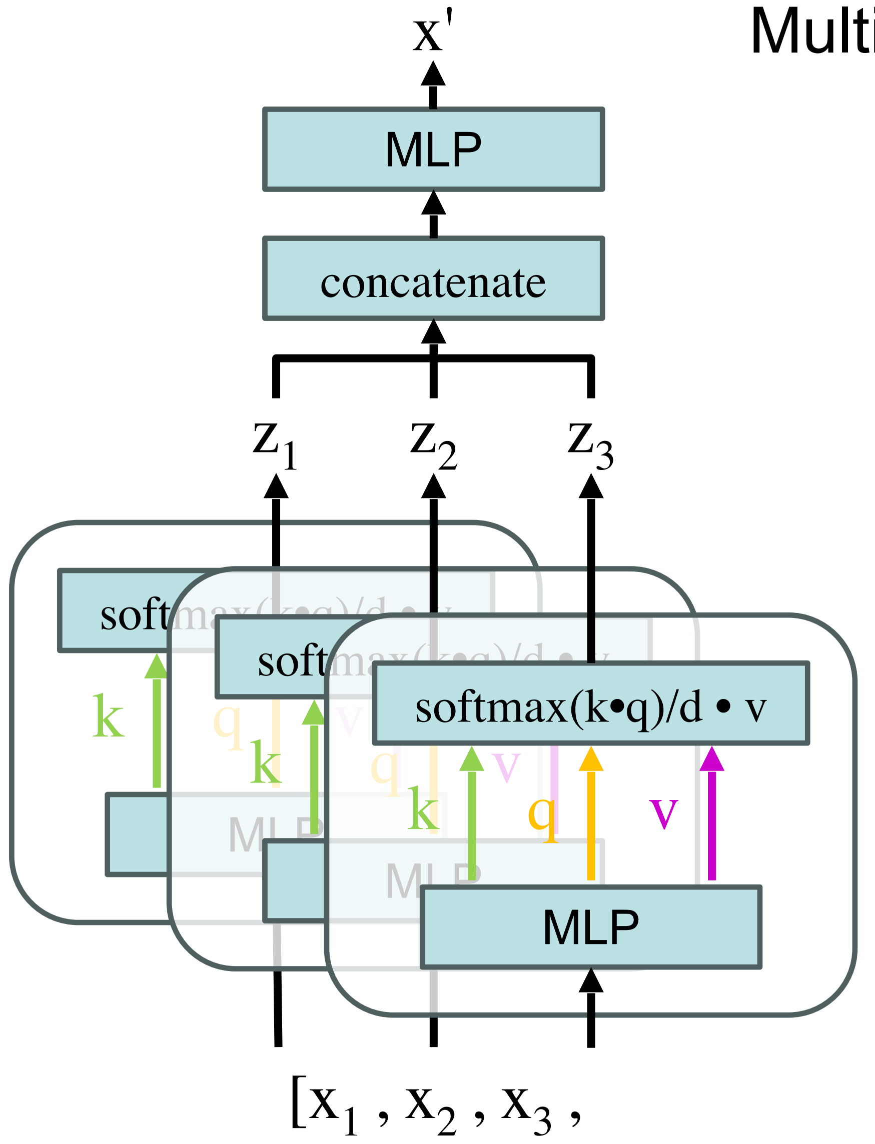

output
x_4
+
attention weight
a_1
a_2
a_3
normalize & softmax
score
s_1
s_2
s_3
key
query
value
k_1
q_1
v_1
k_2
q_2
v_2
k_3
q_3
v_3
multi-layer perceptron
MLP
input
x_1
x_2
x_3

Multi-Headed Attention

Single-headed

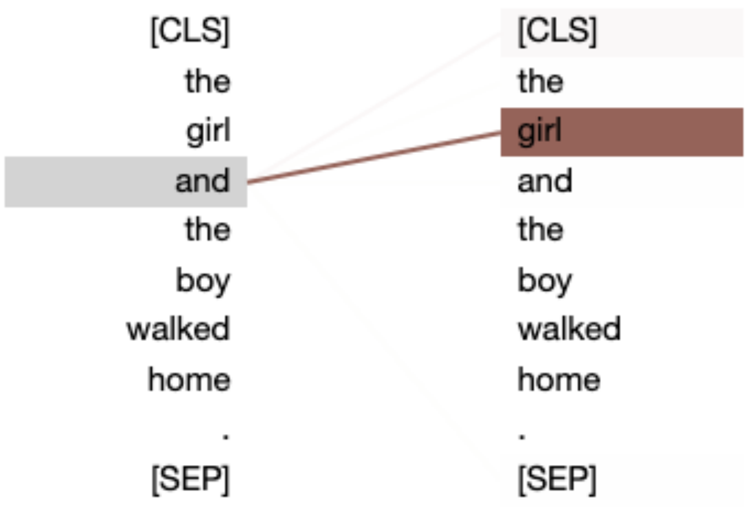


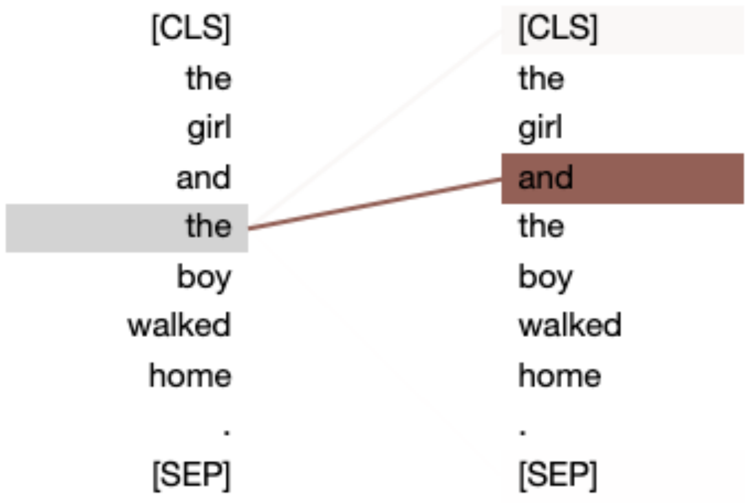
Multi-headed

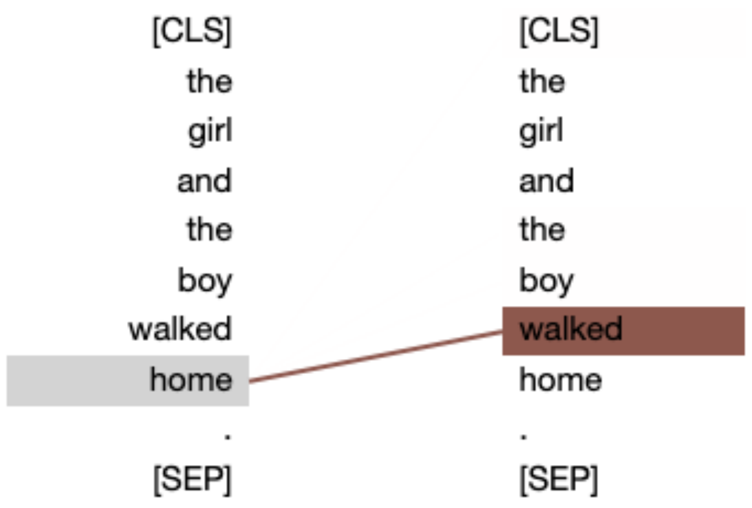


Multi-Headed Attention

Head 6: previous word



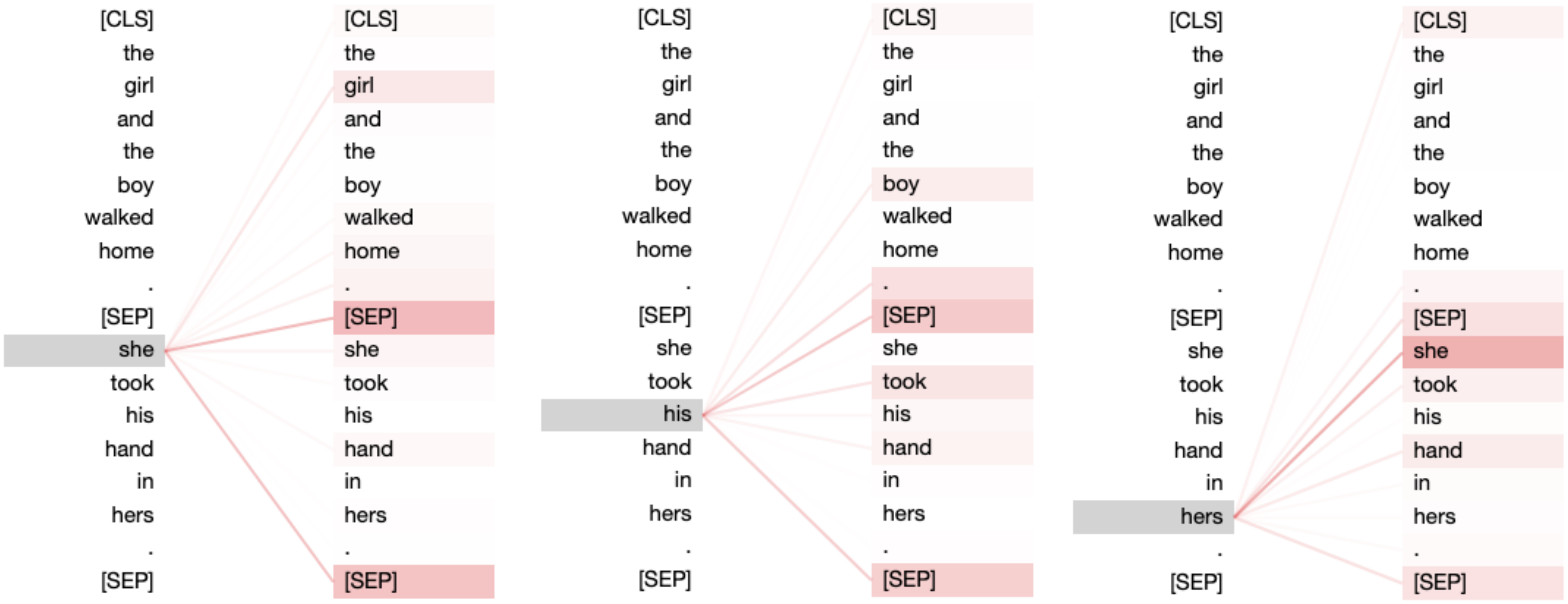




https://github.com/jessevig/bertviz

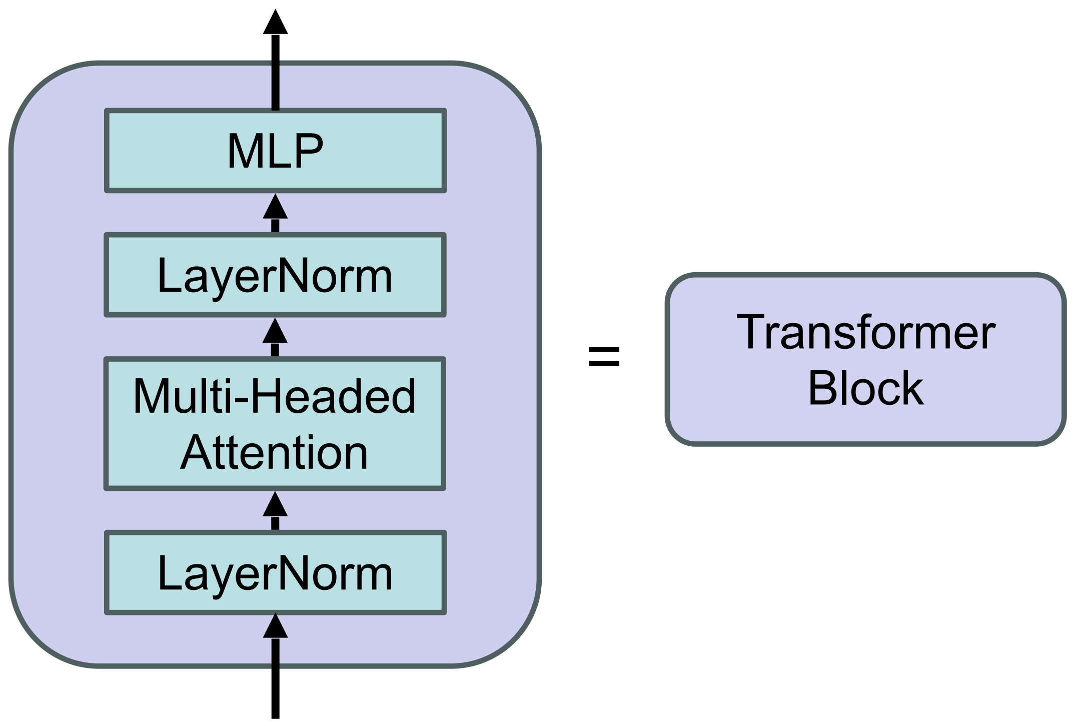
Multi-Headed Attention

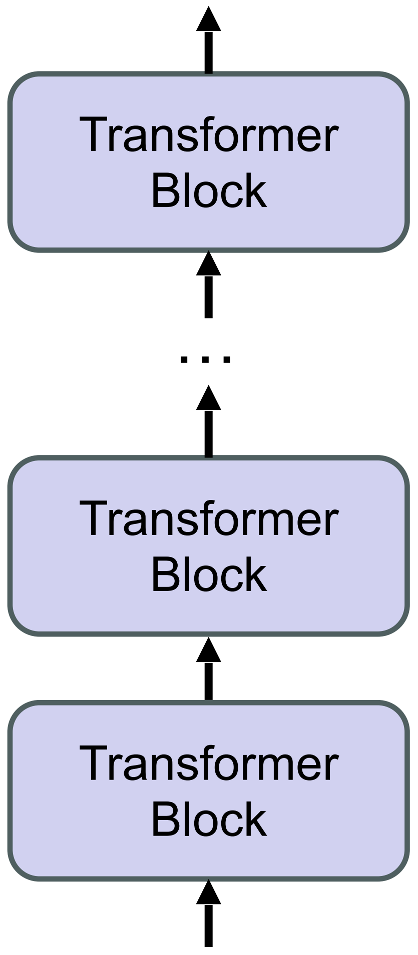
Head 4: pronoun references



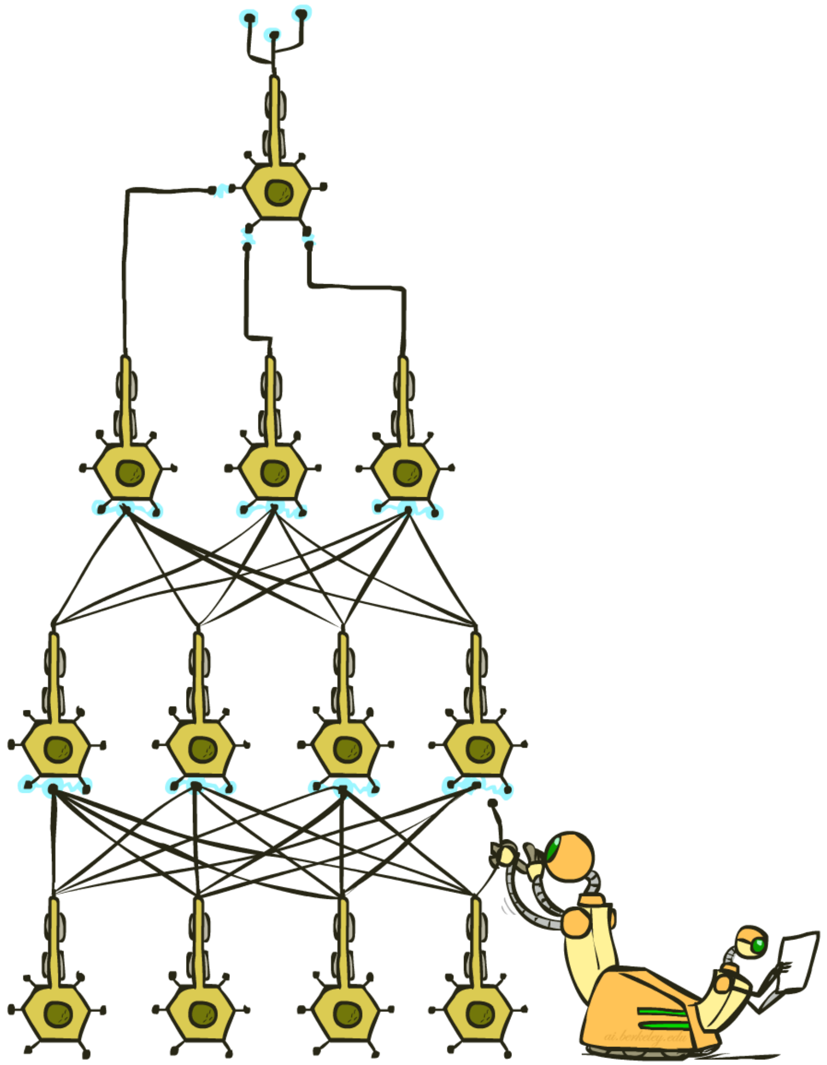
https://github.com/jessevig/bertviz

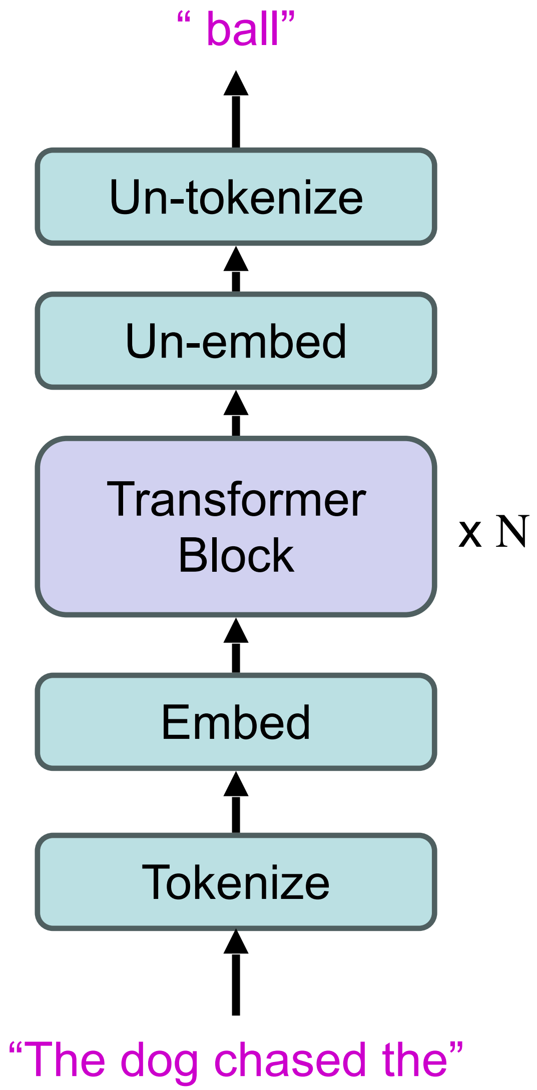
Transformer Architecture





Transformer Architecture





Pre-Training and Fine-Tuning

1. **Pre-Train:** train a large model with a lot of data on a self-supervised task
   - Predict next word / patch of image
   - Predict missing word / patch of image
   - Predict if two images are related (contrastive learning)
2. **Fine-Tune:** continue training the same model on task you care about

Instruction Tuning

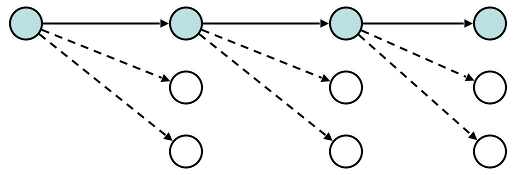
- Task 1 = predict next word (learns to mimic human-written text)
  - Query: `"What is population of Berkeley?"`
  - Human-like completion: `"This question always fascinated me!"`

- Task 2 = generate **helpful** text
  - Query: `"What is population of Berkeley?"`
  - Helpful completion: `"It is 117,145 as of 2021 census."`

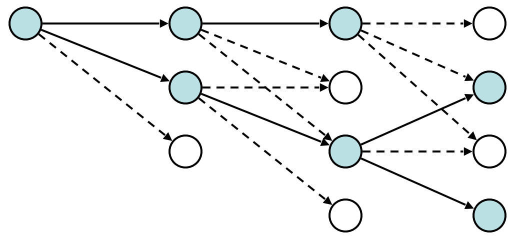
- Fine-tune on collected examples of helpful human conversations
- Also can use Reinforcement Learning

Beam Search

Like greedy hillclimbing search, but keep K states at all times:



Greedy Search



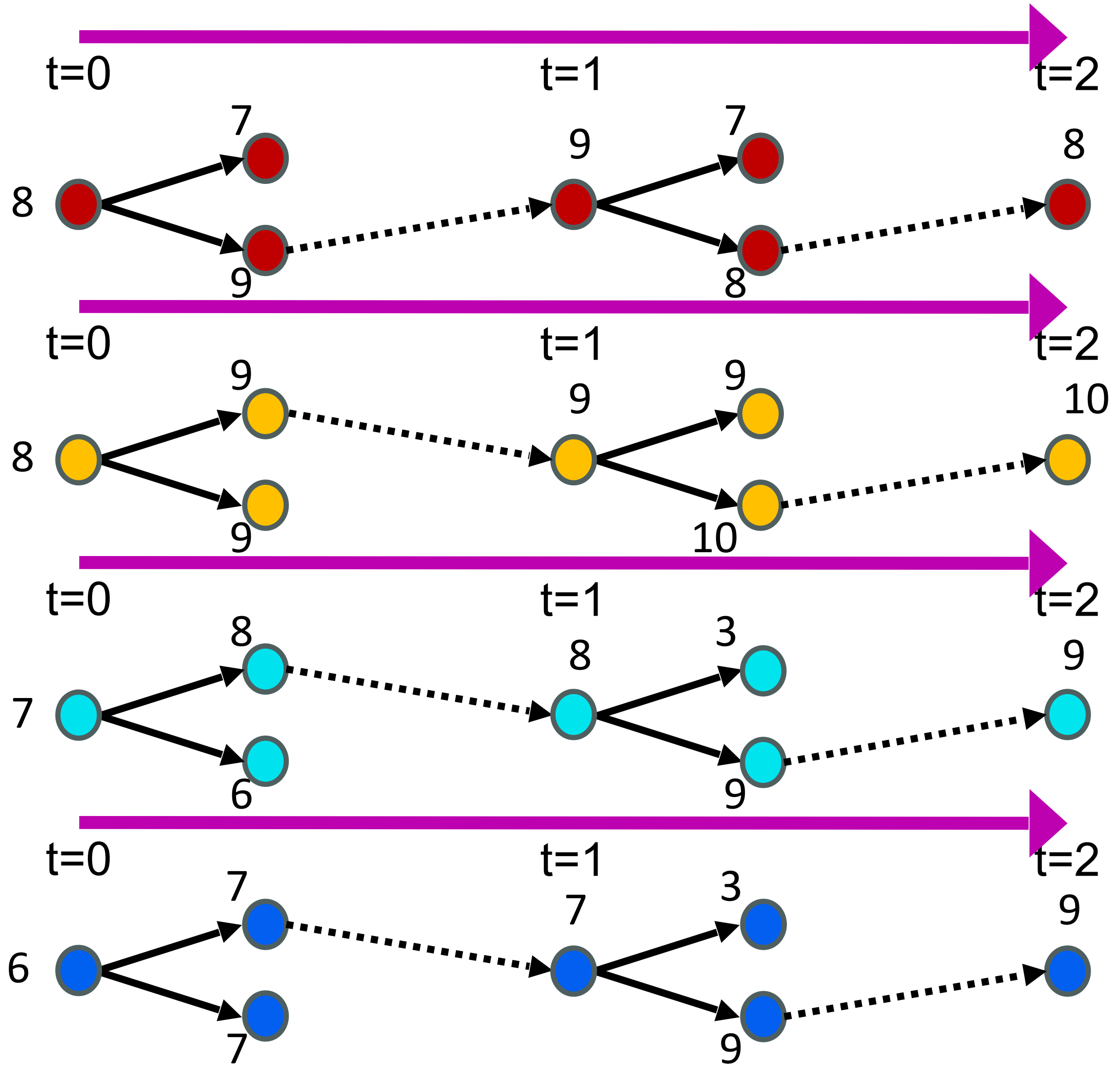
Beam Search

Variables: beam size, encourage diversity?

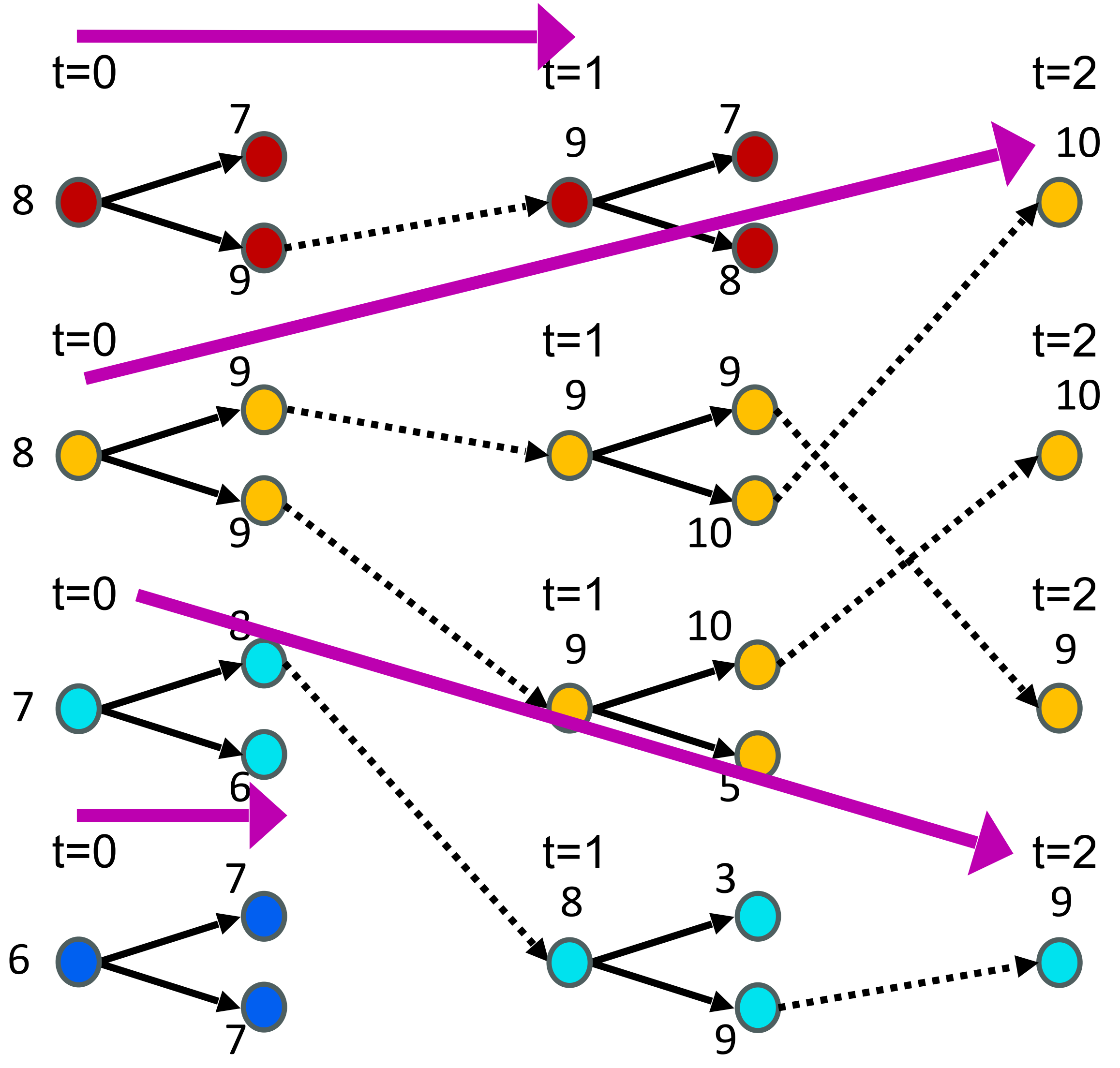
The best choice in MANY practical settings

Complete? Optimal?

Beam Search



Parallel search



Beam search