School of Computer Science
Carnegie Mellon

10-701 Introduction to Machine Learning

HMMs and CRFs

Readings:
Bishop 13.1-13.2
Bishop 8.3-8.4
Sutton & McCallum (2006)
Lafferty et al. (2001)

Matt Gormley
Lecture 19
November 14, 2016

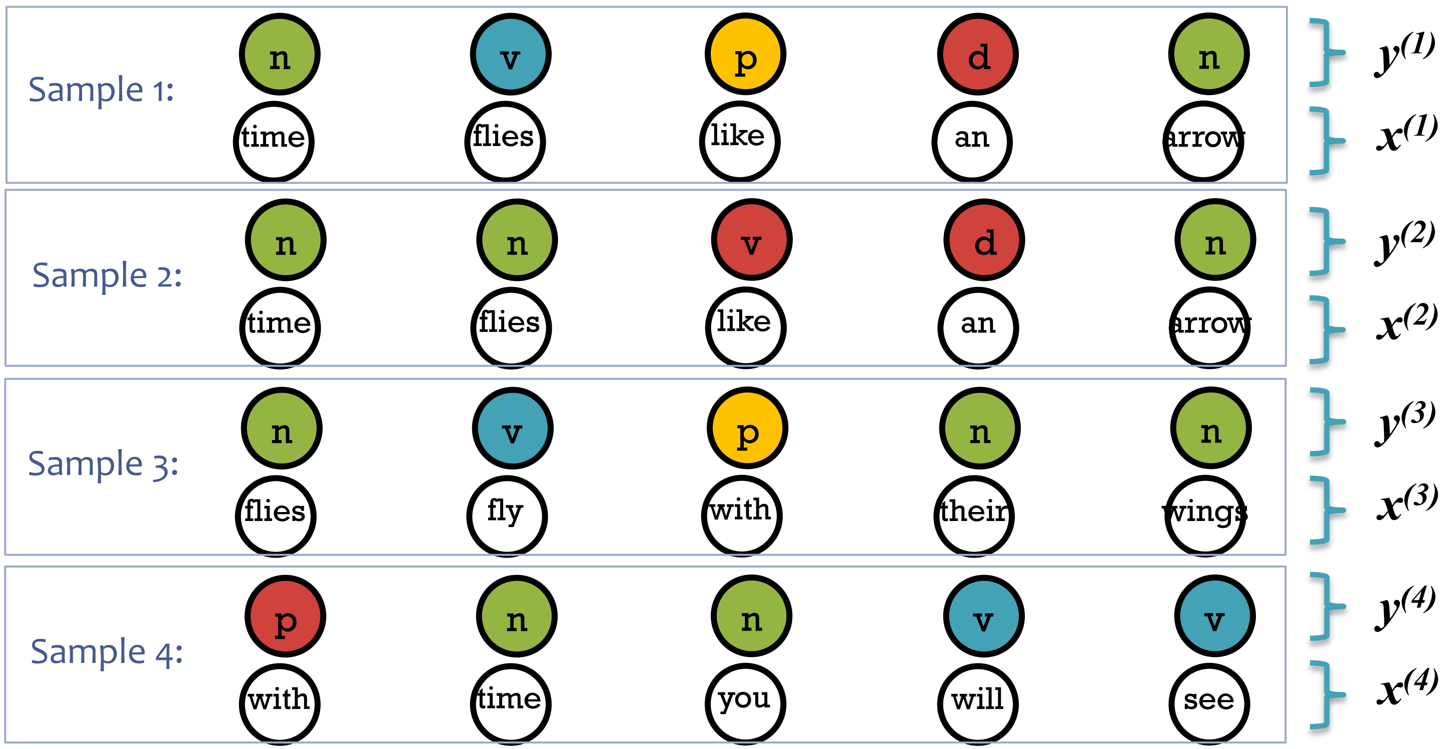
Reminders

- Homework 4
  – deadline extended to Wed, Nov. 16th
  – 10 extra points for submitting by Mon, Nov. 14th
- Poster Sessions
  – two sessions on Fri, Dec. 2nd
  – session 1: 8 - 11:30 am
  – session 2: 2 - 6 pm

HIDDEN MARKOV MODEL (HMM)

Dataset for Supervised Part-of-Speech (POS) Tagging

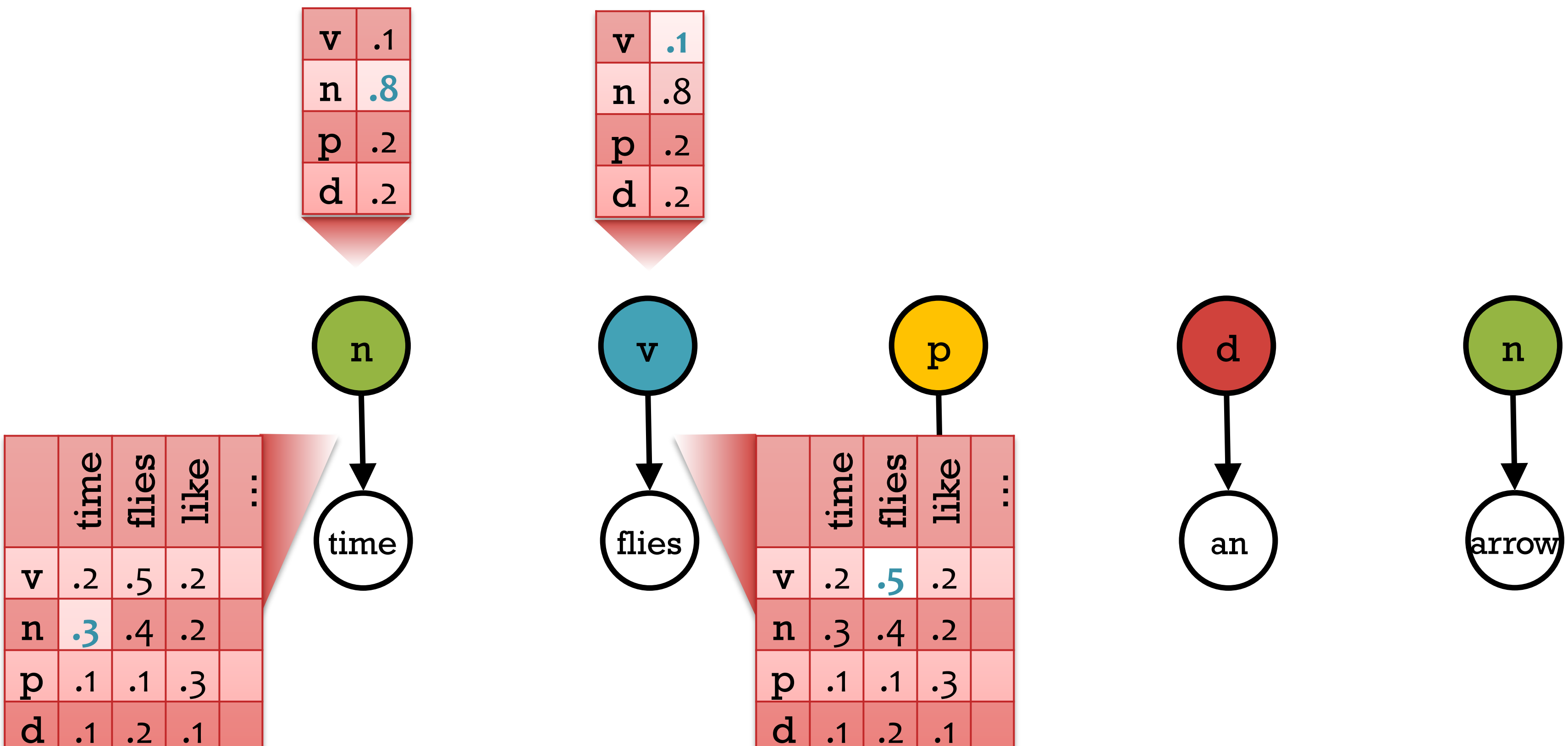
Data: $\mathcal{D} = \{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\}_{n=1}^{N}$



Naïve Bayes for Time Series Data

We could treat each word-tag pair (i.e. token) as independent. This corresponds to a Naïve Bayes model with a single feature (the word).

$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) = (.3 * .8 * .1 * .5 * \ldots)$$



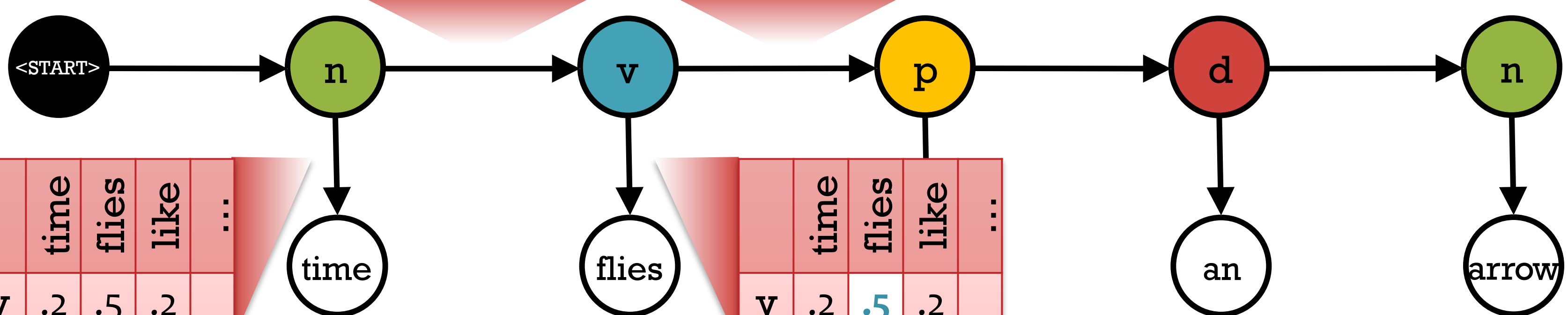
Hidden Markov Model

A Hidden Markov Model (HMM) provides a joint distribution over the the sentence/tags with an assumption of dependence between adjacent tags.

$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) = (.3 * .8 * .2 * .5 * \ldots)$$

	v	n	p	d
v	.1	.4	.2	.3
n	.8	.1	.1	0
p	.2	.3	.2	.3
d	.2	.8	0	0

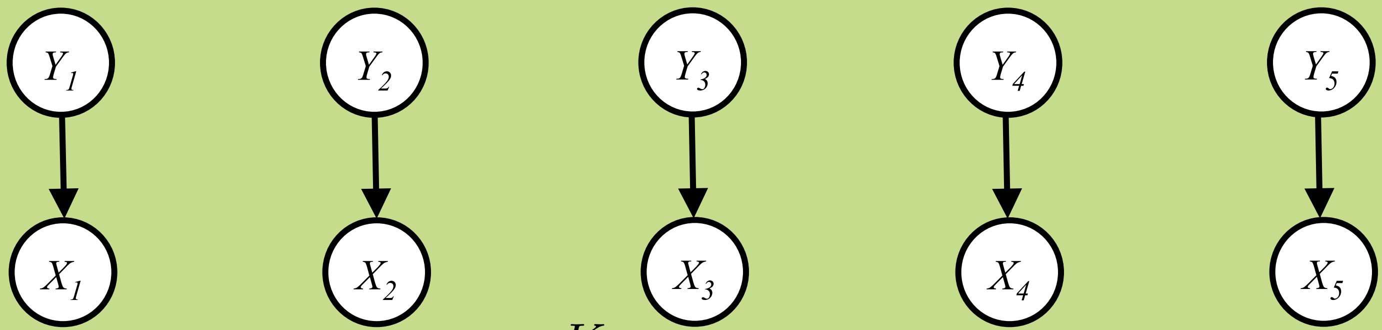
	v	n	p	d
v	.1	.4	.2	.3
n	.8	.1	.1	0
p	.2	.3	.2	.3
d	.2	.8	0	0



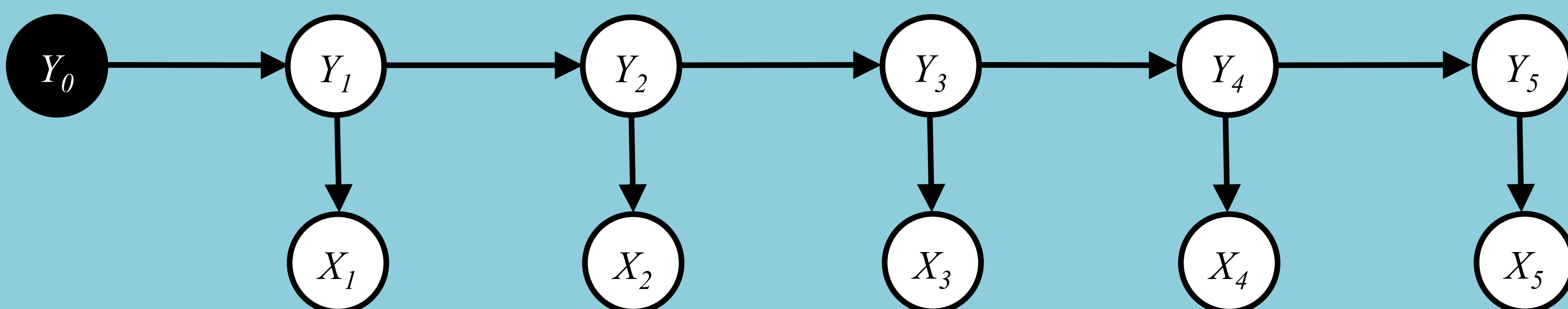
	time	flies	like	…
v	.2	.5	.2	
n	.3	.4	.2	
p	.1	.1	.3	
d	.1	.2	.1	

	time	flies	like	…
v	.2	.5	.2	
n	.3	.4	.2	
p	.1	.1	.3	
d	.1	.2	.1	

From NB to HMM



"Naïve Bayes": $$P(\mathbf{X}, \mathbf{Y}) = \prod_{k=1}^{K} P(X_k|Y_k)p(Y_k)$$



HMM: $$P(\mathbf{X}, \mathbf{Y}) = \prod_{k=1}^{K} P(X_k|Y_k)p(Y_k|Y_{k-1})$$

Hidden Markov Model

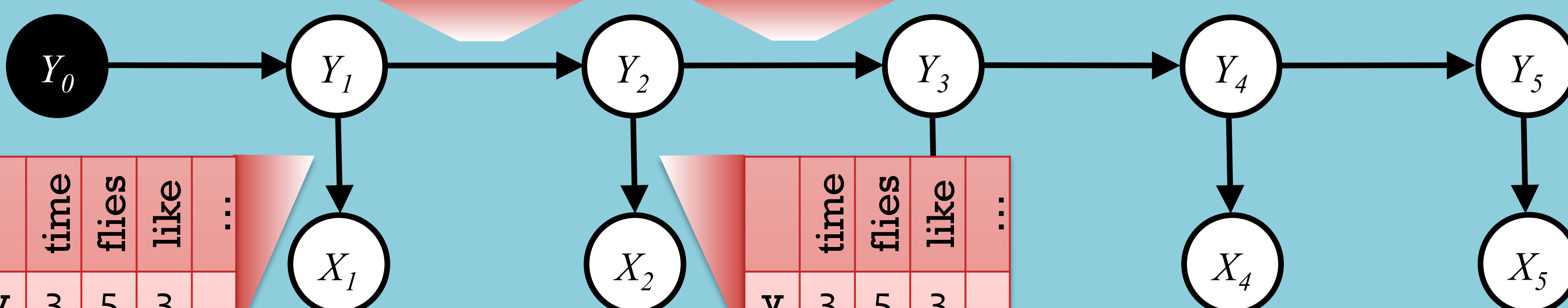
HMM Parameters:

Emission matrix, $\mathbf{A}$, where $P(X_k = w | Y_k = t) = A_{t,w}, \forall k$

Transition matrix, $\mathbf{B}$, where $P(Y_k = t | Y_{k-1} = s) = B_{s,t}, \forall k$

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0



	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

Hidden Markov Model

HMM Parameters:

Emission matrix, $\mathbf{A}$, where $P(X_k = w | Y_k = t) = A_{t,w}, \forall k$

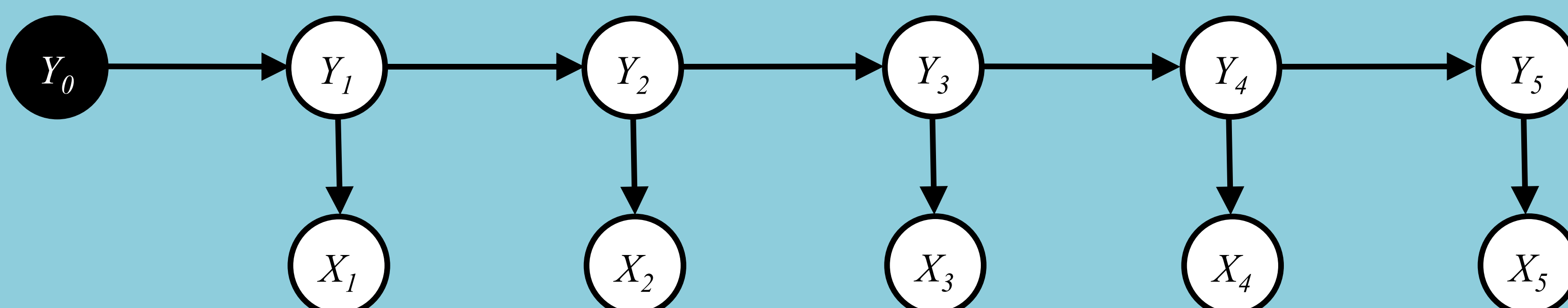
Transition matrix, $\mathbf{B}$, where $P(Y_k = t | Y_{k-1} = s) = B_{s,t}, \forall k$

Assumption: $y_0 = \text{START}$

Generative Story:

$$Y_k \sim \text{Multinomial}(\mathbf{A}_{Y_{k-1}}) \; \forall k$$

$$X_k \sim \text{Multinomial}(\mathbf{B}_{Y_k}) \; \forall k$$

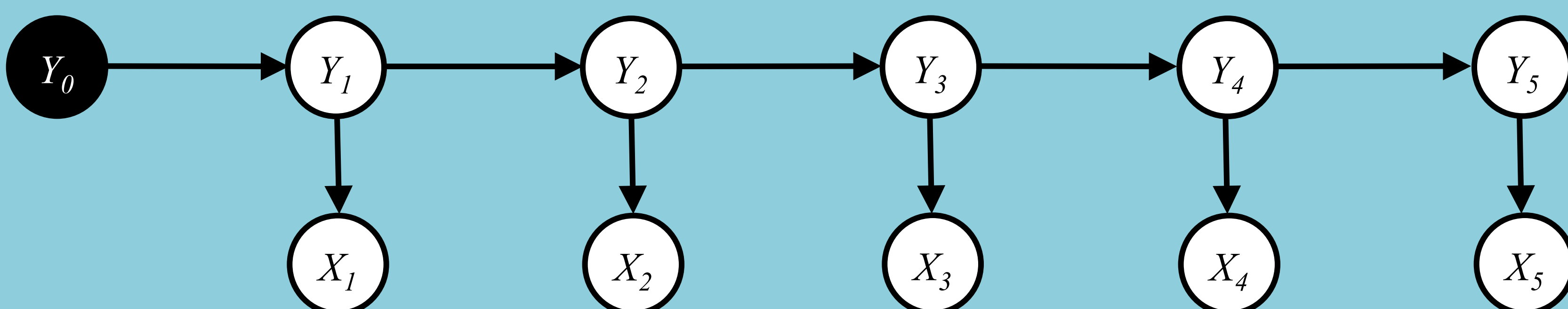


Hidden Markov Model

Joint Distribution:

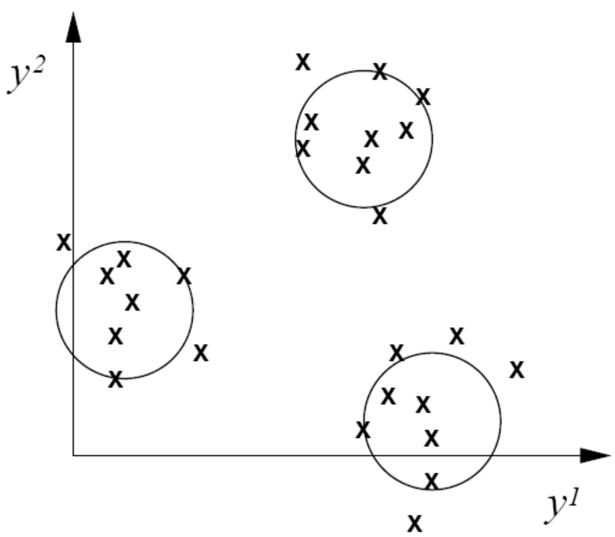
$$p(\mathbf{x}, \mathbf{y}) = \prod_{k=1}^{K} p(x_k | y_k) p(y_k | y_{k-1})$$

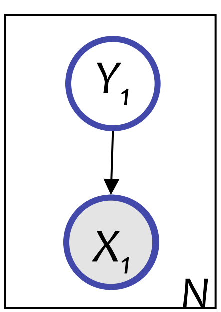
$$= \prod_{k=1}^{K} A_{y_k, x_k} B_{y_{k-1}, y_k}$$



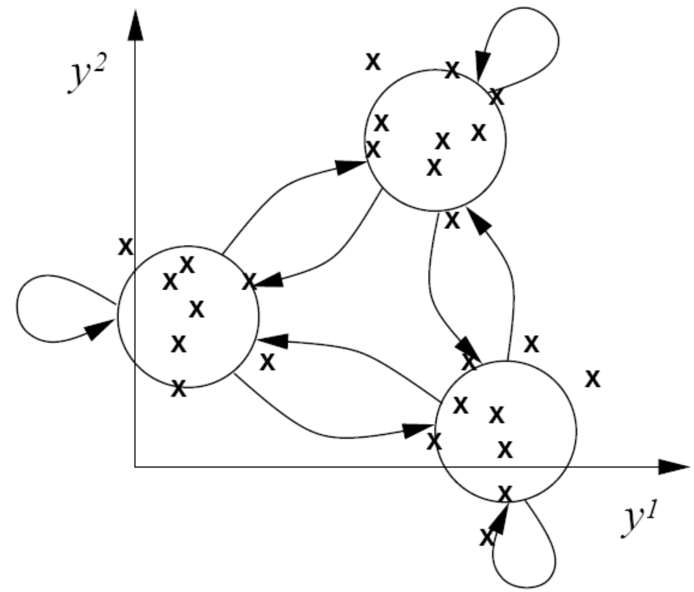
From static to dynamic mixture models

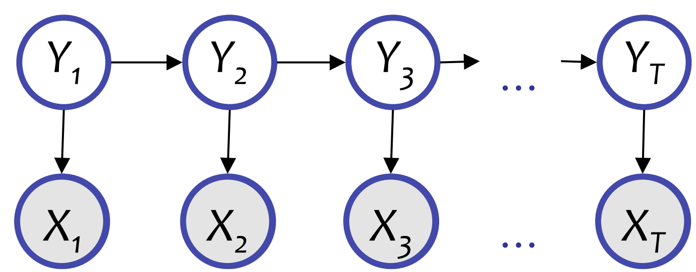
Static mixture





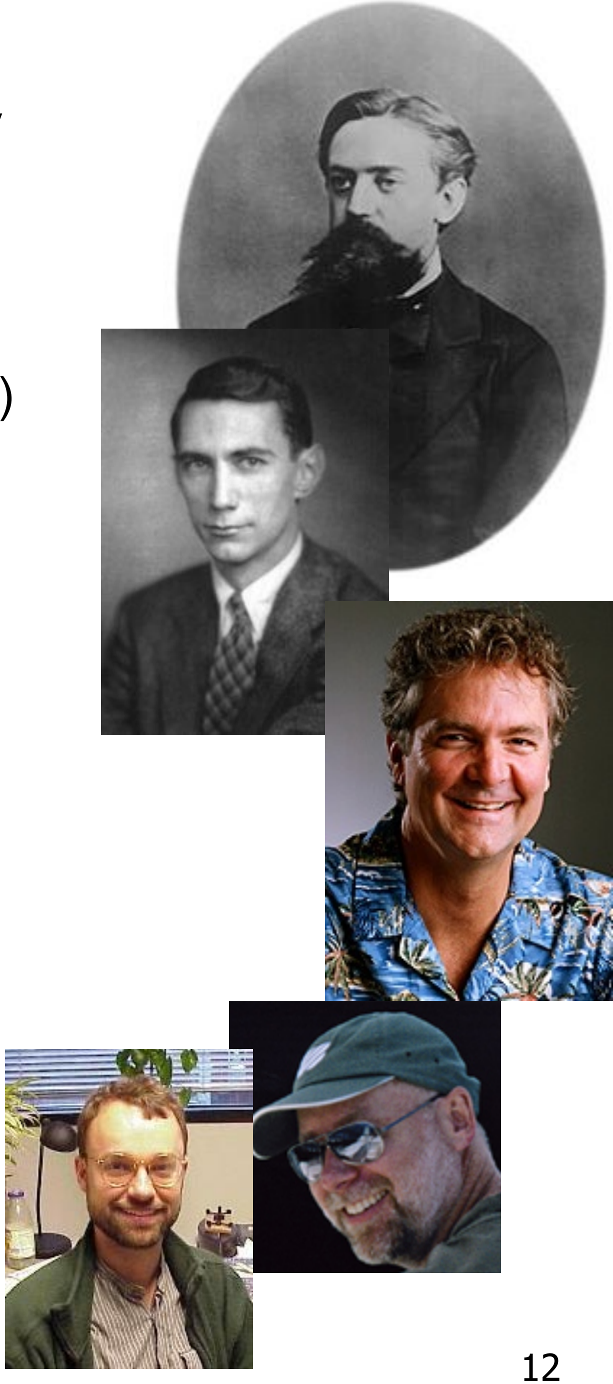
Dynamic mixture





HMMs: History

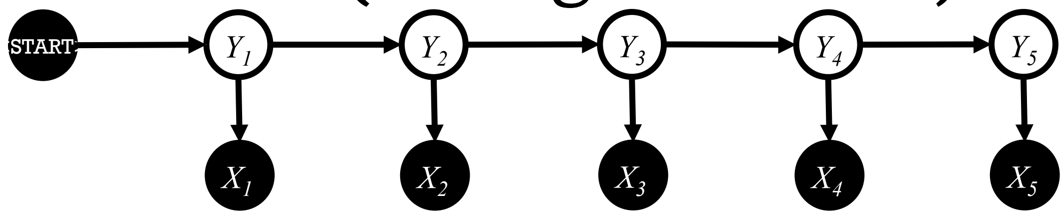
- Markov chains: Andrey Markov (1906)
  - Random walks and Brownian motion
- Used in Shannon's work on information theory (1948)
- Baum-Welsh learning algorithm: late 60's, early 70's.
  - Used mainly for speech in 60s-70s.
- Late 80's and 90's: David Haussler (major player in learning theory in 80's) began to use HMMs for modeling biological sequences
- Mid-late 1990's: Dayne Freitag/Andrew McCallum
  - Freitag thesis with Tom Mitchell on IE from Web using logic programs, grammar induction, etc.
  - McCallum: multinomial Naïve Bayes for text
  - With McCallum, IE using HMMs on CORA
- …



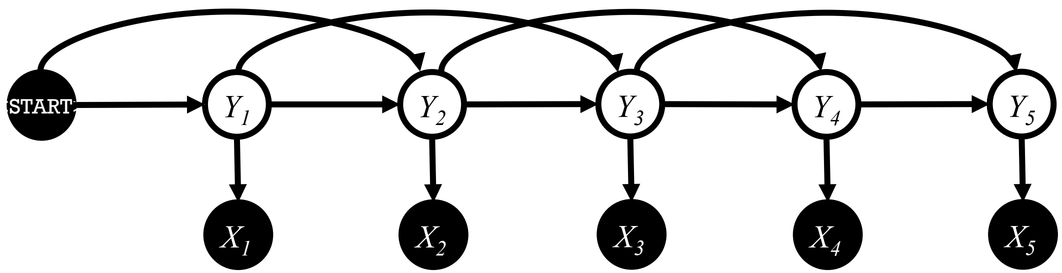
Slide from William Cohen

Higher-order HMMs

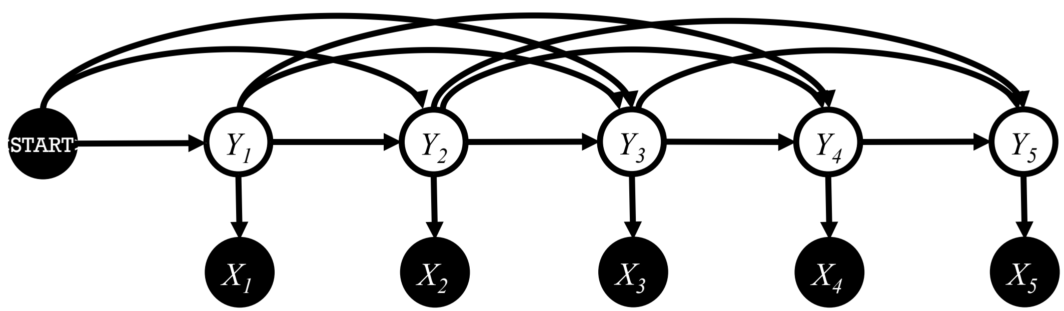
- 1^{st}-order HMM (i.e. bigram HMM)



- 2^{nd}-order HMM (i.e. trigram HMM)



- 3^{rd}-order HMM



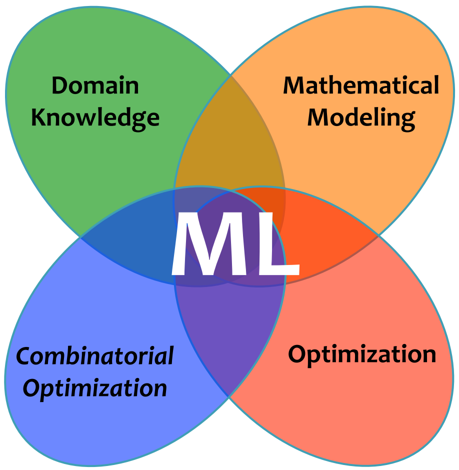
SUPERVISED LEARNING FOR BAYES NETS

Machine Learning

The **data** inspires the structures we want to predict

Our **model** defines a score for each structure

It also tells us what to optimize



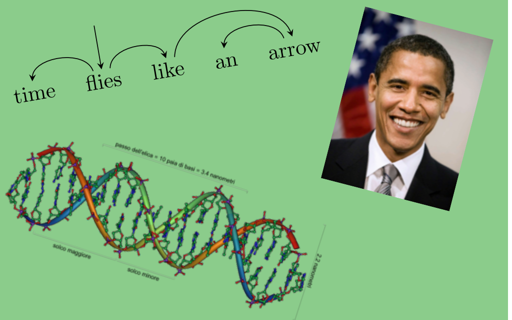
Inference finds {best structure, marginals, partition function} for a new observation

(**Inference** is usually called as a subroutine in learning)

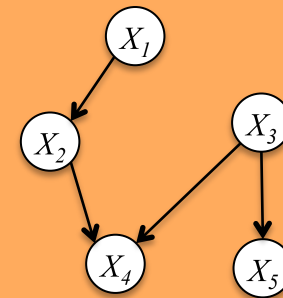
Learning tunes the parameters of the model

Machine Learning

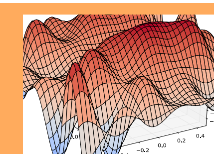
Data



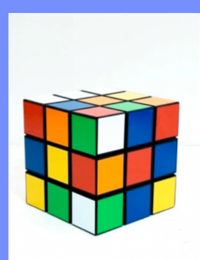
Model



Objective

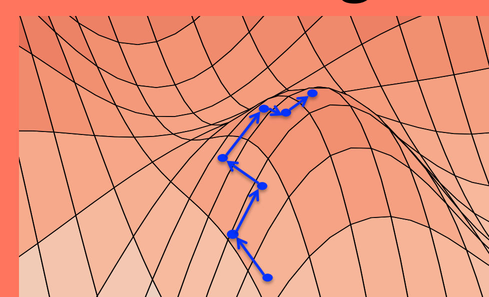


Inference

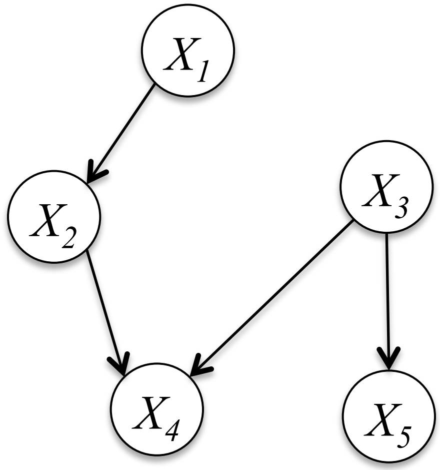


(**Inference** is usually called as a subroutine in learning)

Learning



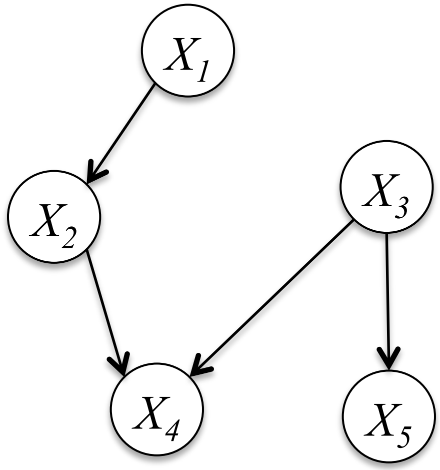
Learning Fully Observed BNs



$$p(X_1, X_2, X_3, X_4, X_5) =$$
$$p(X_5|X_3)p(X_4|X_2, X_3)$$
$$p(X_3)p(X_2|X_1)p(X_1)$$

Learning Fully Observed BNs

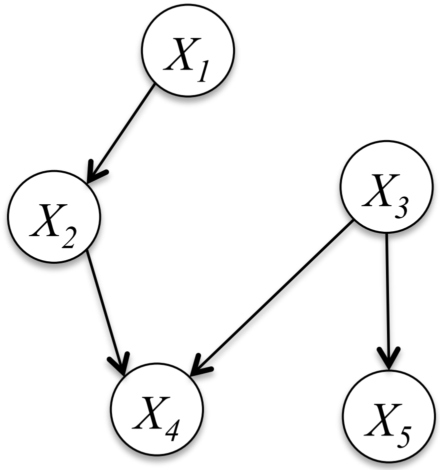
Recall...



$$p(X_1, X_2, X_3, X_4, X_5) =$$
$$p(X_5|X_3)p(X_4|X_2, X_3)$$
$$p(X_3)p(X_2|X_1)p(X_1)$$

Recall…

Learning Fully Observed BNs



$$p(X_1, X_2, X_3, X_4, X_5) =$$
$$p(X_5|X_3)p(X_4|X_2, X_3)$$
$$p(X_3)p(X_2|X_1)p(X_1)$$

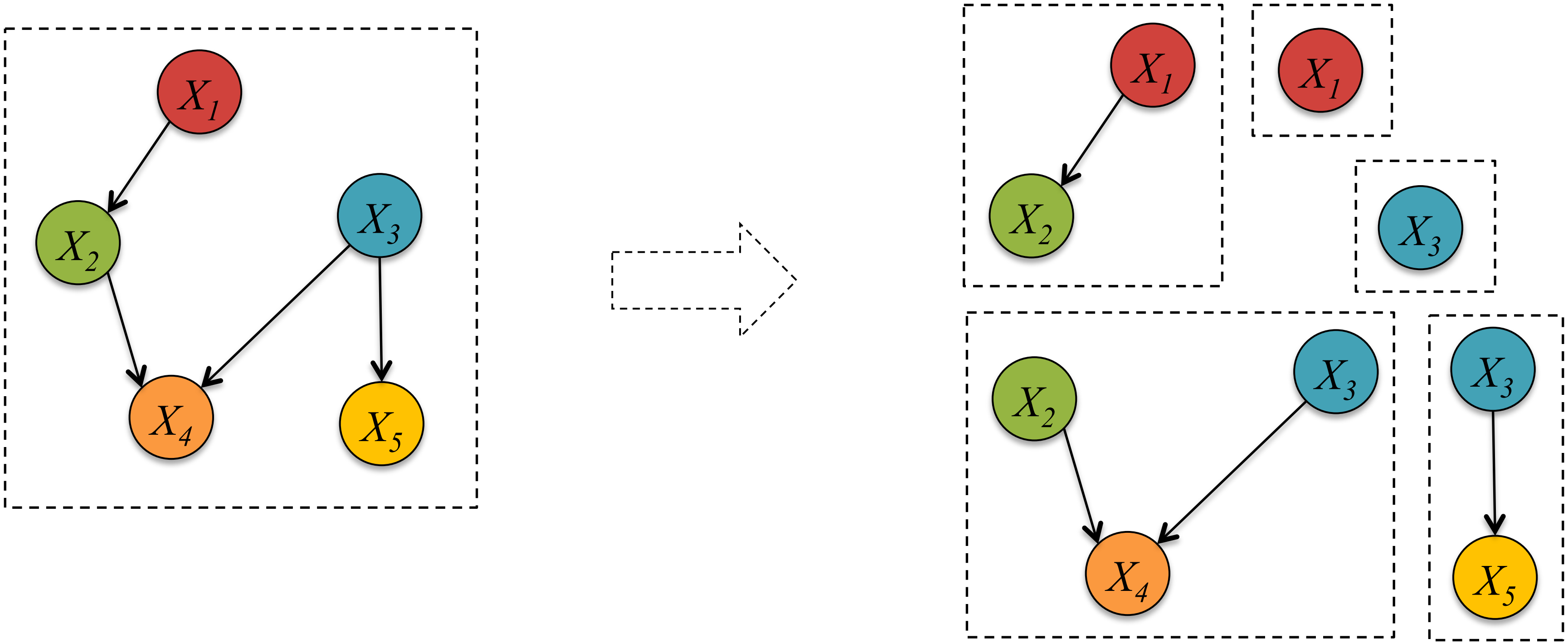
How do we learn these conditional and marginal distributions for a Bayes Net?

Learning Fully Observed BNs

Recall...

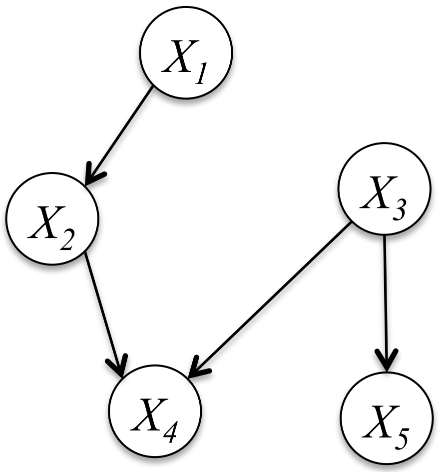
Learning this fully observed Bayesian Network is **equivalent** to learning five (small / simple) independent networks from the same data

$$p(X_1, X_2, X_3, X_4, X_5) = p(X_5|X_3)p(X_4|X_2, X_3) p(X_3)p(X_2|X_1)p(X_1)$$



Learning Fully Observed BNs

How do we **learn** these conditional and marginal distributions for a Bayes Net?



$$\begin{aligned}\boldsymbol{\theta}^* &= \underset{\boldsymbol{\theta}}{\text{argmax}} \log p(X_1, X_2, X_3, X_4, X_5) \\ &= \underset{\boldsymbol{\theta}}{\text{argmax}} \log p(X_5 | X_3, \theta_5) + \log p(X_4 | X_2, X_3, \theta_4) \\ &\qquad + \log p(X_3 | \theta_3) + \log p(X_2 | X_1, \theta_2) \\ &\qquad + \log p(X_1 | \theta_1)\end{aligned}$$

$$\theta_1^* = \underset{\theta_1}{\text{argmax}} \log p(X_1 | \theta_1)$$

$$\theta_2^* = \underset{\theta_2}{\text{argmax}} \log p(X_2 | X_1, \theta_2)$$

$$\theta_3^* = \underset{\theta_3}{\text{argmax}} \log p(X_3 | \theta_3)$$

$$\theta_4^* = \underset{\theta_4}{\text{argmax}} \log p(X_4 | X_2, X_3, \theta_4)$$

$$\theta_5^* = \underset{\theta_5}{\text{argmax}} \log p(X_5 | X_3, \theta_5)$$

SUPERVISED LEARNING FOR HMMS

Hidden Markov Model

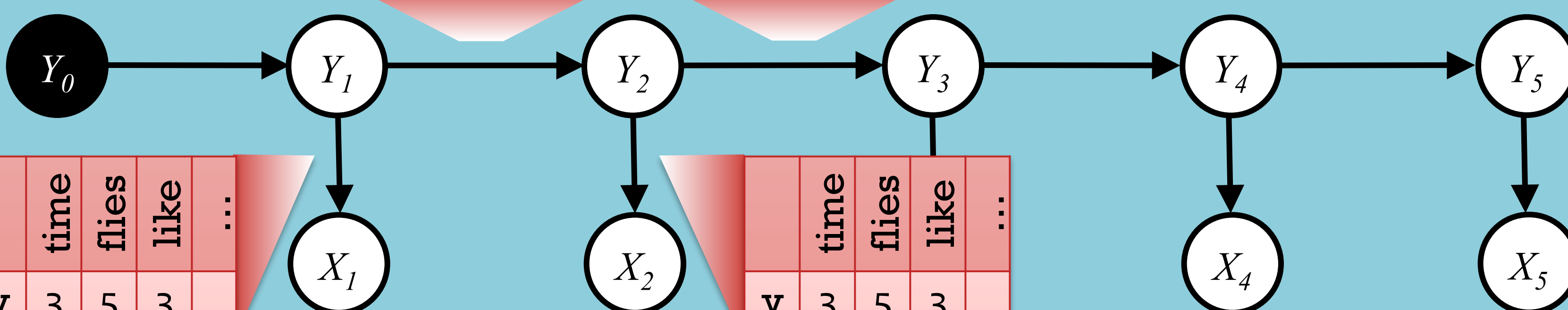
HMM Parameters:

Emission matrix, $\mathbf{A}$, where $P(X_k = w | Y_k = t) = A_{t,w}, \forall k$

Transition matrix, $\mathbf{B}$, where $P(Y_k = t | Y_{k-1} = s) = B_{s,t}, \forall k$

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0



	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

Hidden Markov Model

HMM Parameters:

Emission matrix, $\mathbf{A}$, where $P(X_k = w | Y_k = t) = A_{t,w}, \forall k$

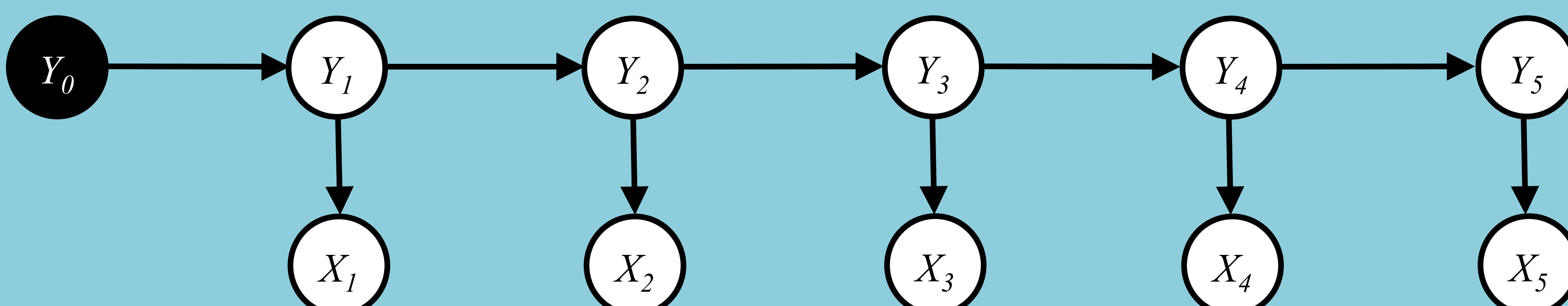
Transition matrix, $\mathbf{B}$, where $P(Y_k = t | Y_{k-1} = s) = B_{s,t}, \forall k$

Assumption: $y_0 =$ START

Generative Story:

$$Y_k \sim \text{Multinomial}(\mathbf{A}_{Y_{k-1}}) \ \forall k$$

$$X_k \sim \text{Multinomial}(\mathbf{B}_{Y_k}) \ \forall k$$

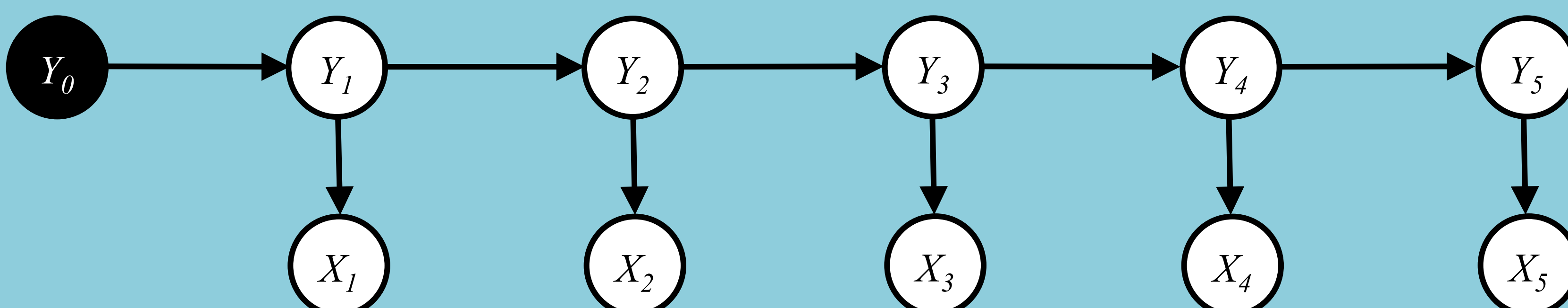


Hidden Markov Model

Joint Distribution:

$$p(\mathbf{x}, \mathbf{y}) = \prod_{k=1}^{K} p(x_k | y_k) p(y_k | y_{k-1})$$

$$= \prod_{k=1}^{K} A_{y_k, x_k} B_{y_{k-1}, y_k}$$



Whiteboard

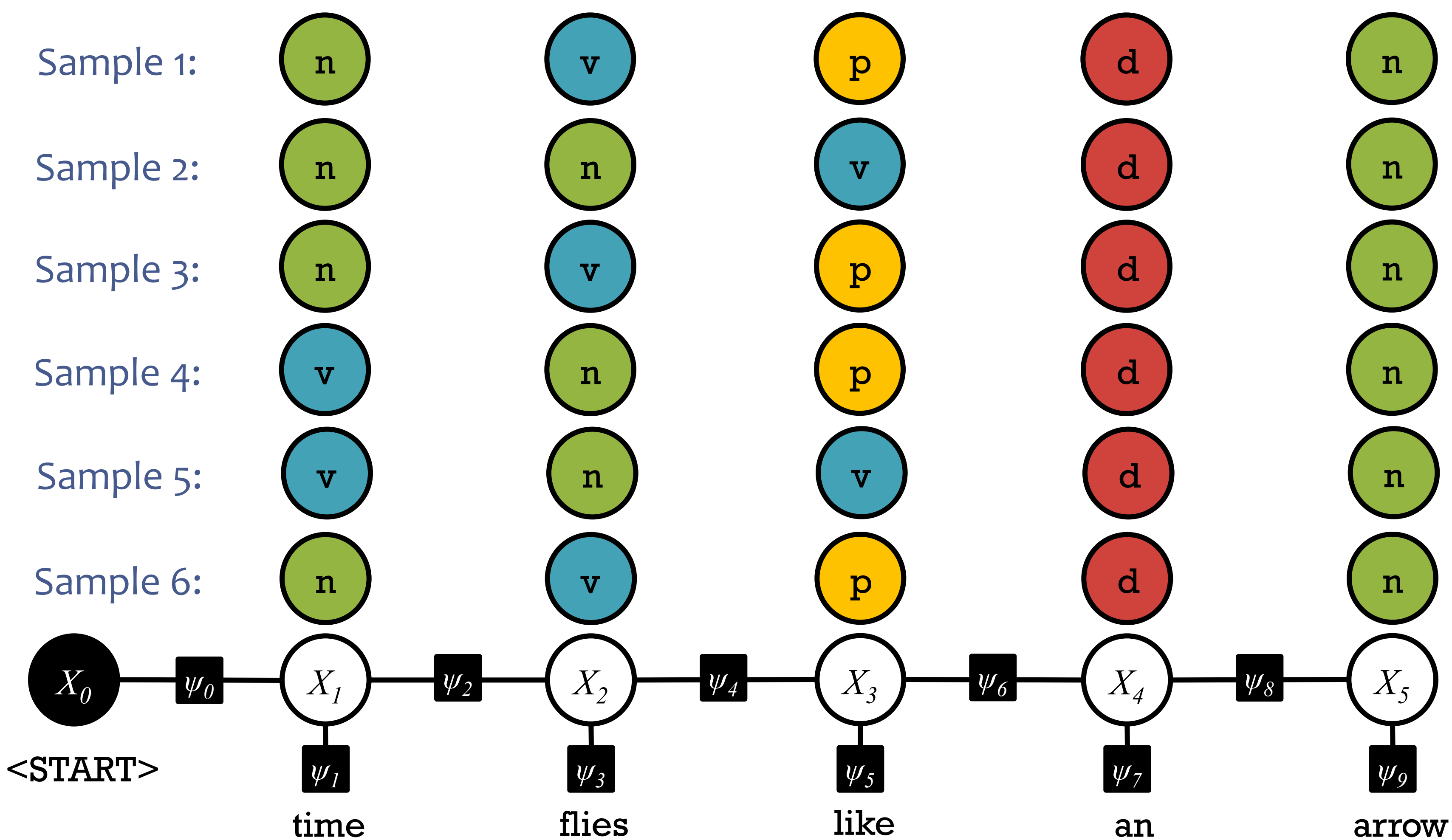
- MLEs for HMM

Representation of both directed and undirected graphical models

FACTOR GRAPHS

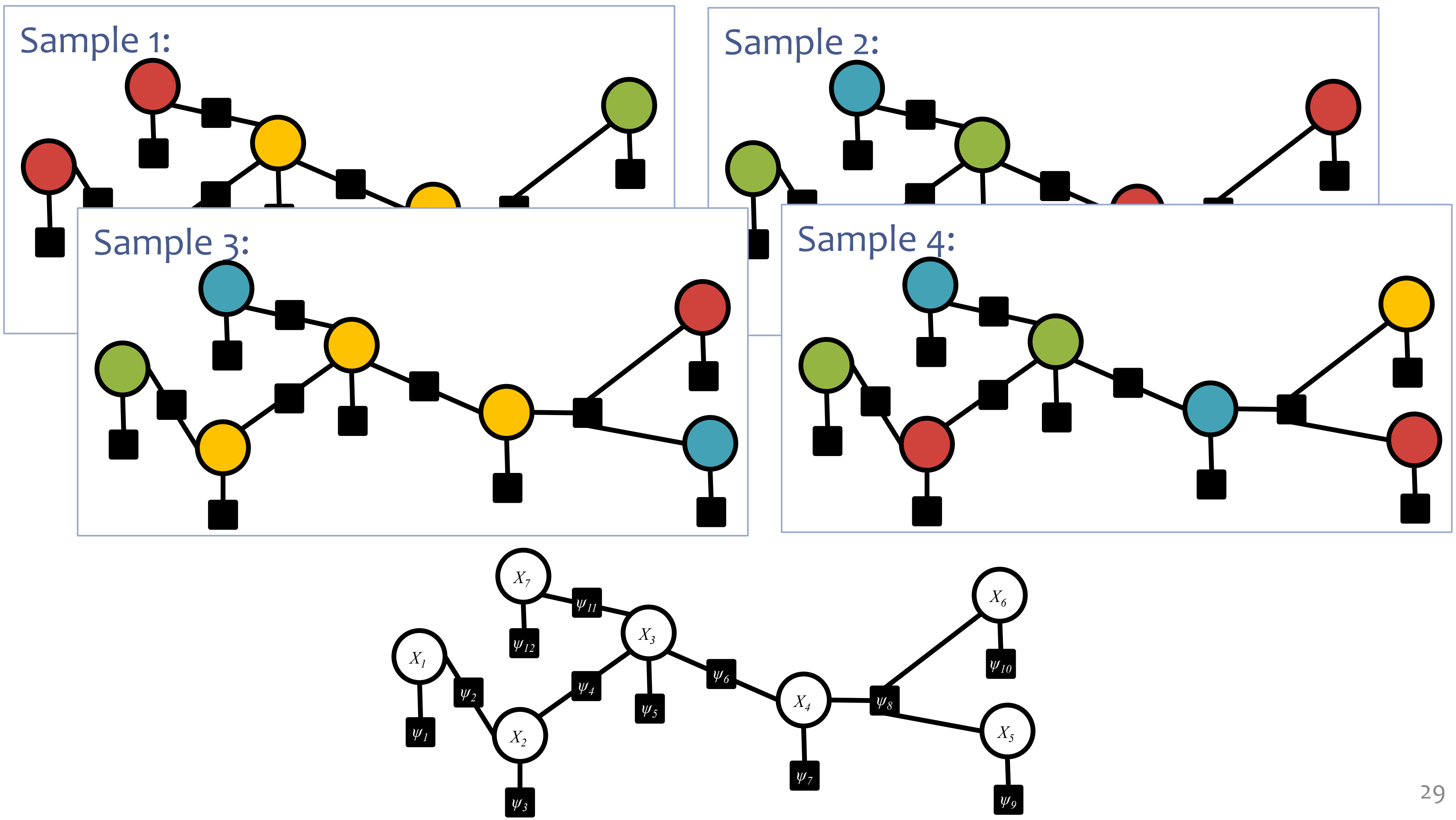
Sampling from a Joint Distribution

A **joint distribution** defines a probability $p(\boldsymbol{x})$ for each assignment of values $\boldsymbol{x}$ to variables $\boldsymbol{X}$.
This gives the **proportion** of samples that will equal $\boldsymbol{x}$.



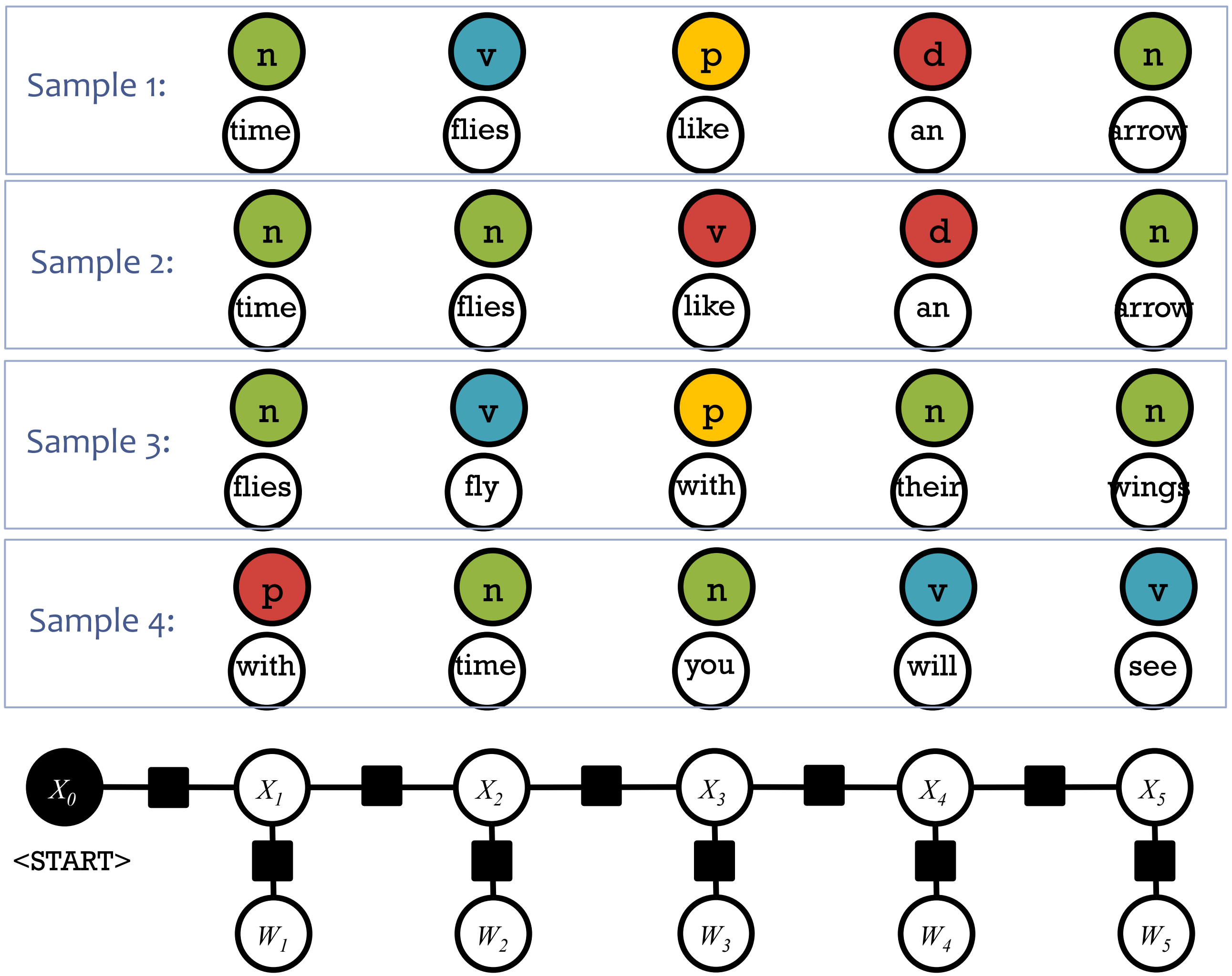
Sampling from a Joint Distribution

A **joint distribution** defines a probability $p(\boldsymbol{x})$ for each assignment of values $\boldsymbol{x}$ to variables $\boldsymbol{X}$. This gives the **proportion** of samples that will equal $\boldsymbol{x}$.



Sampling from a Joint Distribution

A **joint distribution** defines a probability $p(\boldsymbol{x})$ for each assignment of values $\boldsymbol{x}$ to variables $\boldsymbol{X}$. This gives the **proportion** of samples that will equal $\boldsymbol{x}$.



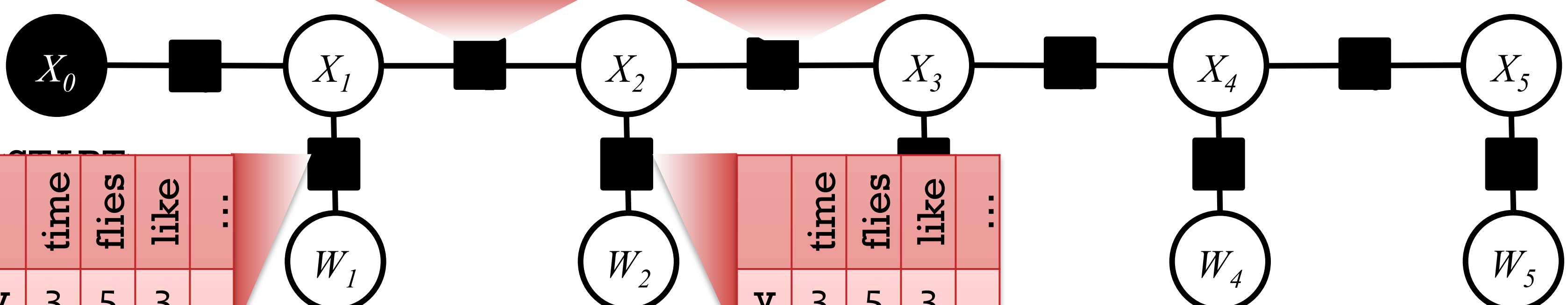
Factors have local opinions (≥ 0)

Each black box looks at *some* of the tags X_i and words W_i

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

Note: We chose to reuse the same factors at different positions in the sentence.



	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

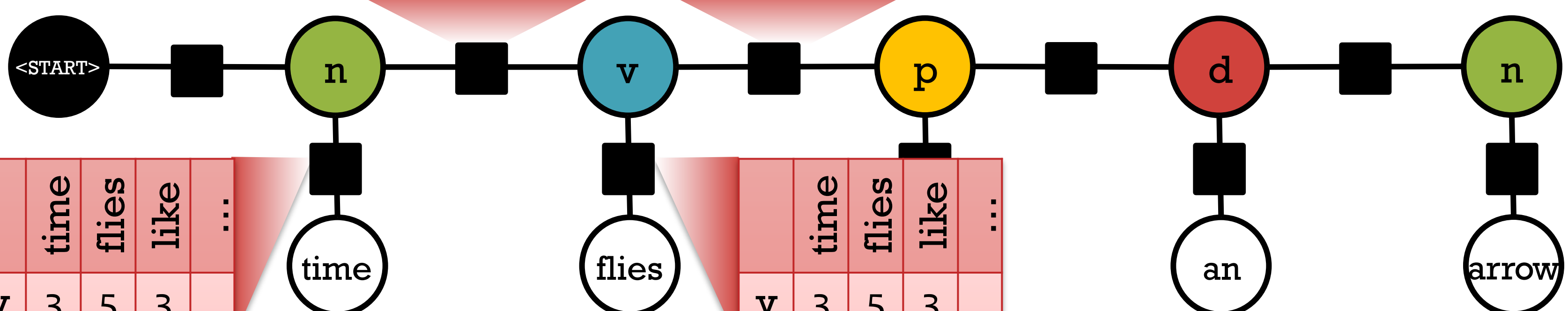
Factors have local opinions (≥ 0)

Each black box looks at *some* of the tags X_i and words W_i

$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) \quad = \quad ?$$

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0



	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

Global probability = product of local opinions

Each black box looks at *some* of the tags X_i and words W_i

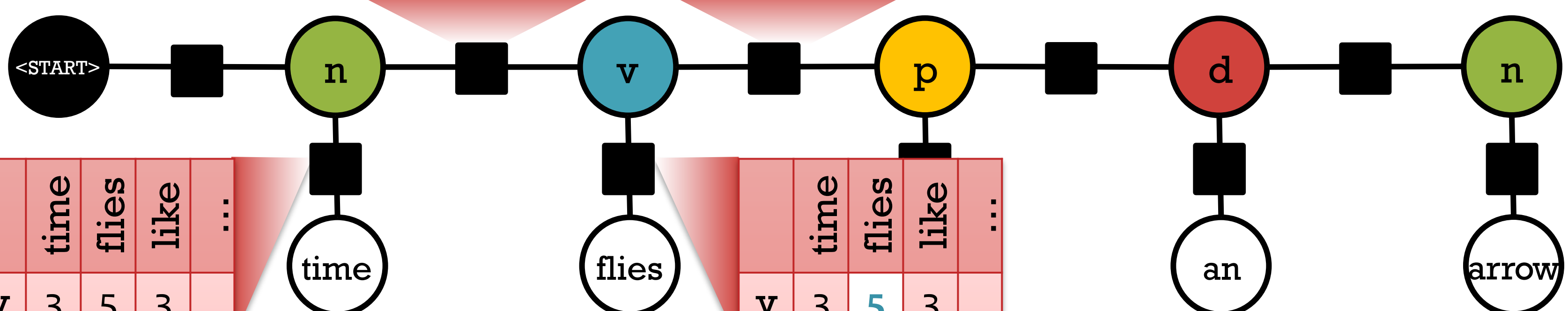
$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) = \frac{1}{Z}(4 * 8 * 5 * 3 * \ldots)$$

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

Uh-oh! The probabilities of the various assignments sum up to $Z > 1$.

So divide them all by Z.



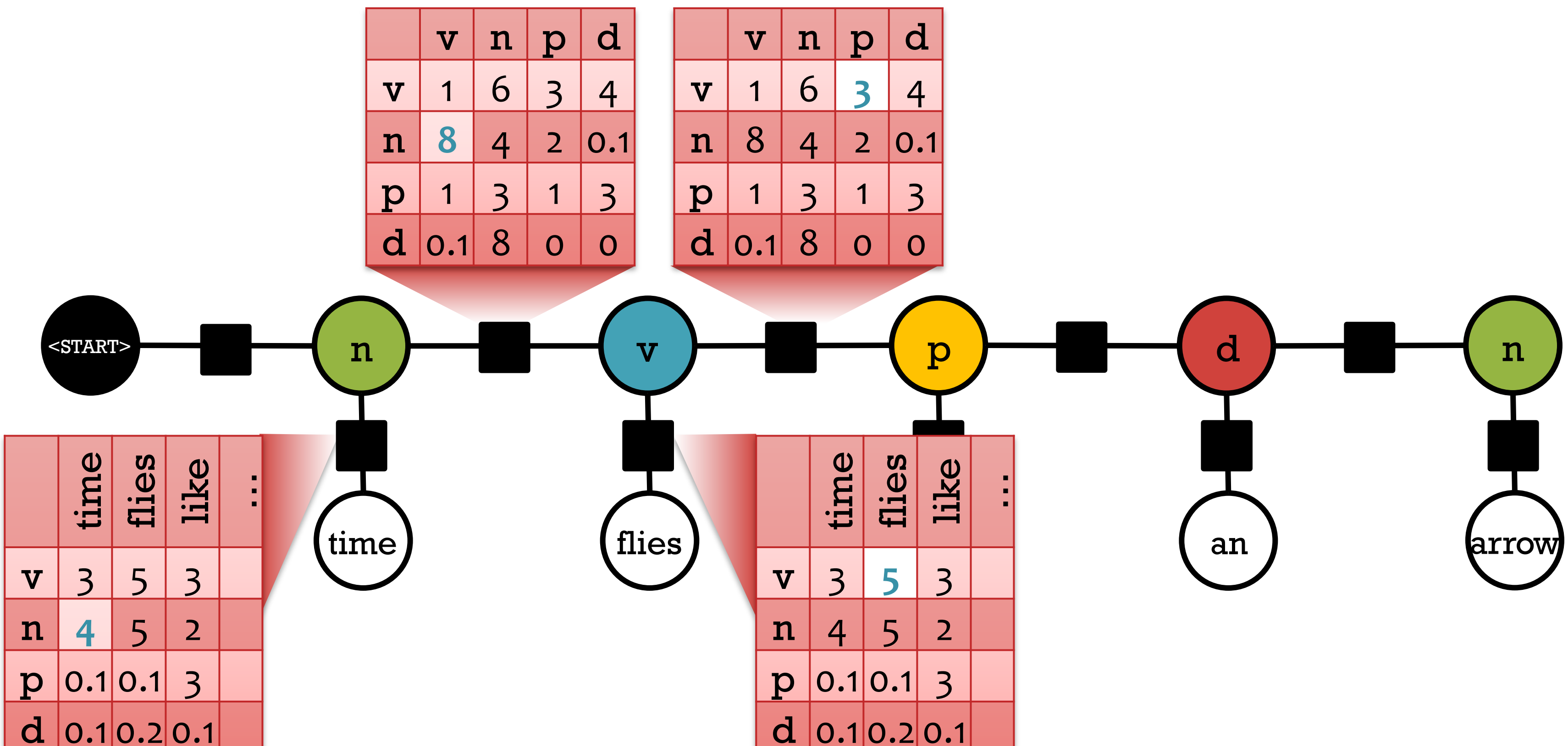
	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

Markov Random Field (MRF)

Joint distribution over tags X_i <u>and</u> words W_i
The individual factors aren't *necessarily* probabilities.

$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) = \frac{1}{Z}(4 * 8 * 5 * 3 * \ldots)$$



	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

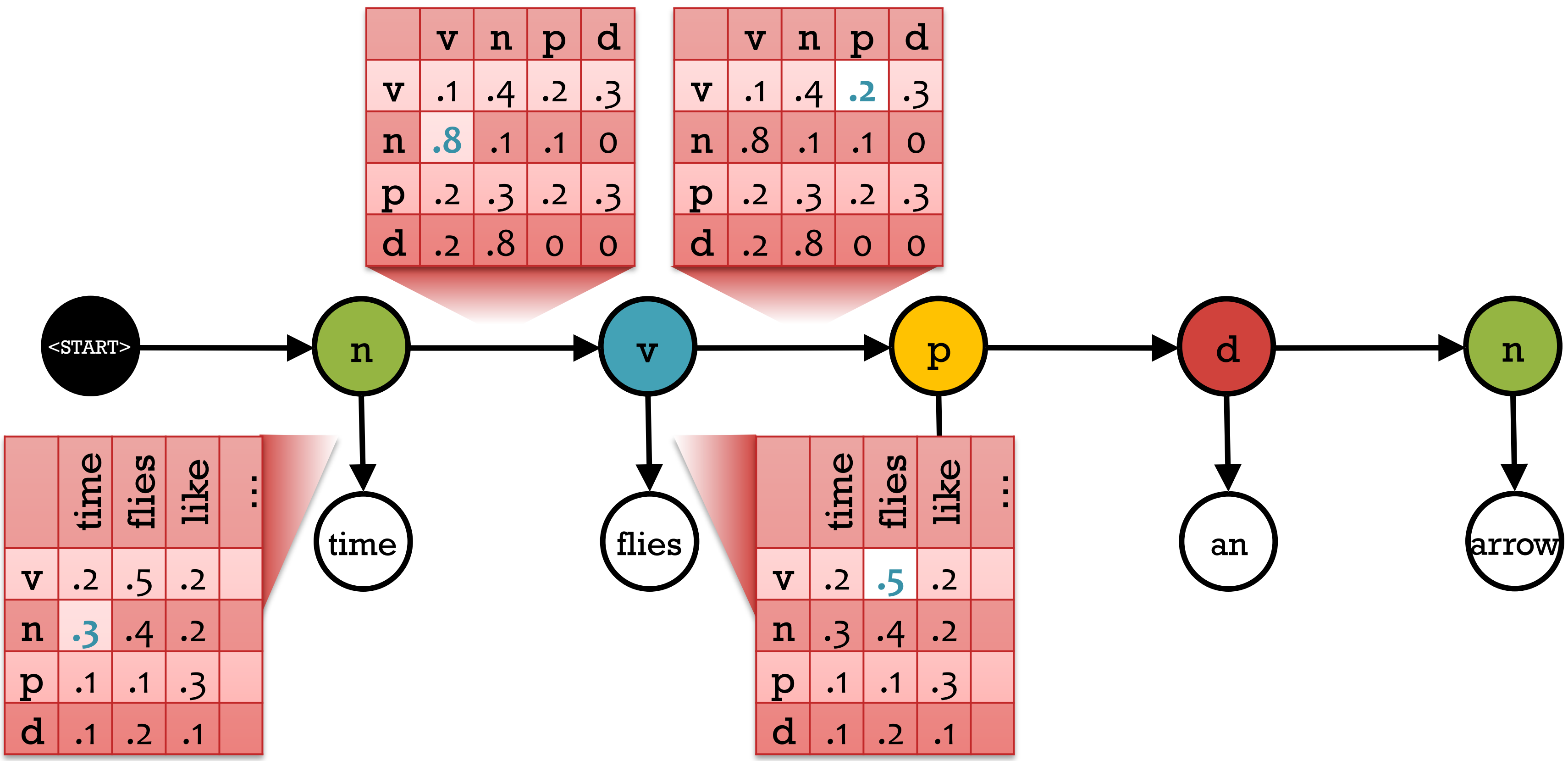
	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

Bayesian Networks

But sometimes we *choose* to make them probabilities.
Constrain each row of a factor to sum to one. Now $Z = 1$.

$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) = \cancel{\frac{1}{Z}}(.3 * .8 * .2 * .5 * \ldots)$$



	v	n	p	d
v	.1	.4	.2	.3
n	.8	.1	.1	0
p	.2	.3	.2	.3
d	.2	.8	0	0

	v	n	p	d
v	.1	.4	.2	.3
n	.8	.1	.1	0
p	.2	.3	.2	.3
d	.2	.8	0	0

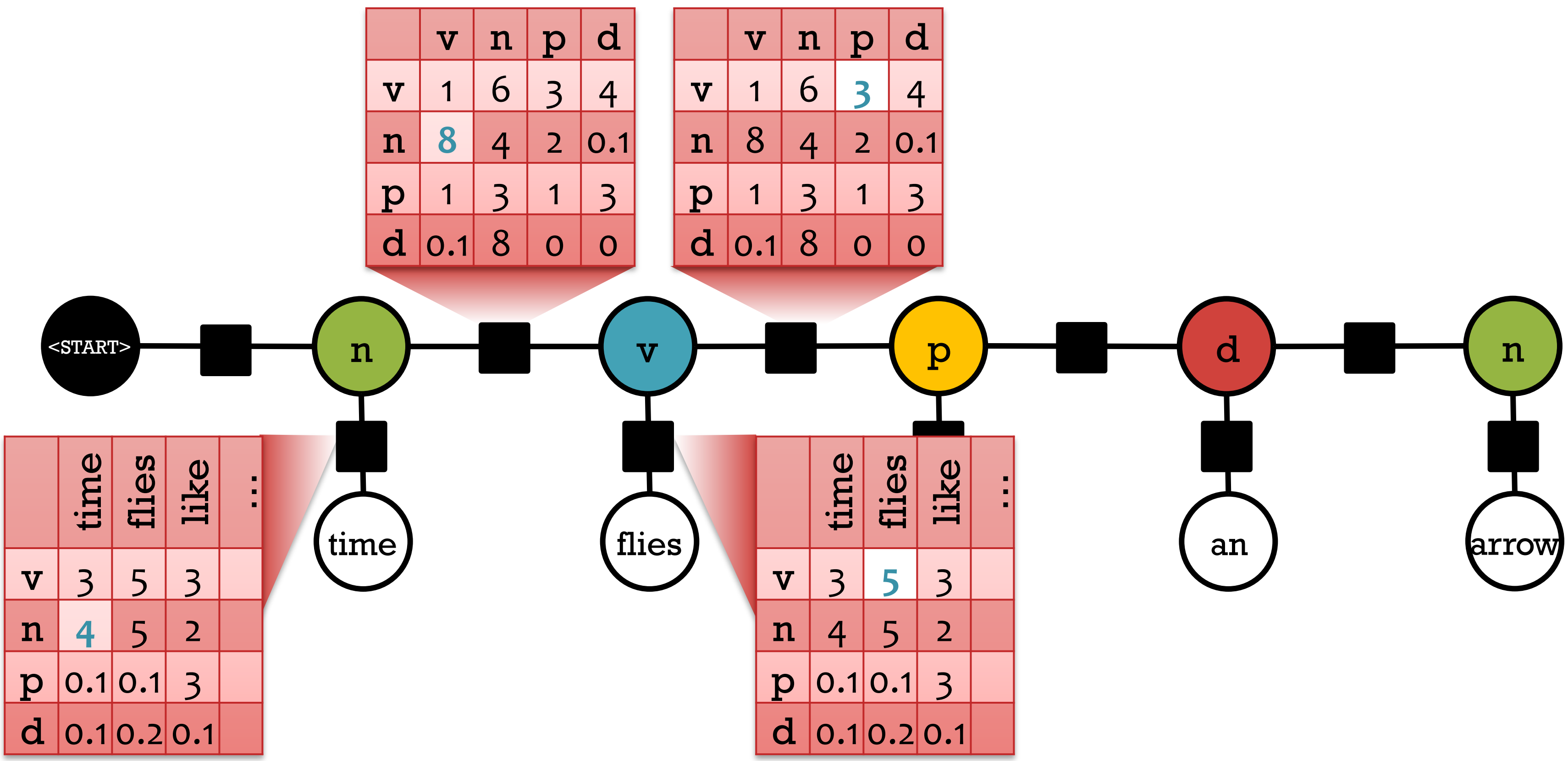
	time	flies	like	…
v	.2	.5	.2	
n	.3	.4	.2	
p	.1	.1	.3	
d	.1	.2	.1	

	time	flies	like	…
v	.2	.5	.2	
n	.3	.4	.2	
p	.1	.1	.3	
d	.1	.2	.1	

Markov Random Field (MRF)

Joint distribution over tags X_i <u>and</u> words W_i

$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) = \frac{1}{Z}(4 * 8 * 5 * 3 * \ldots)$$



	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

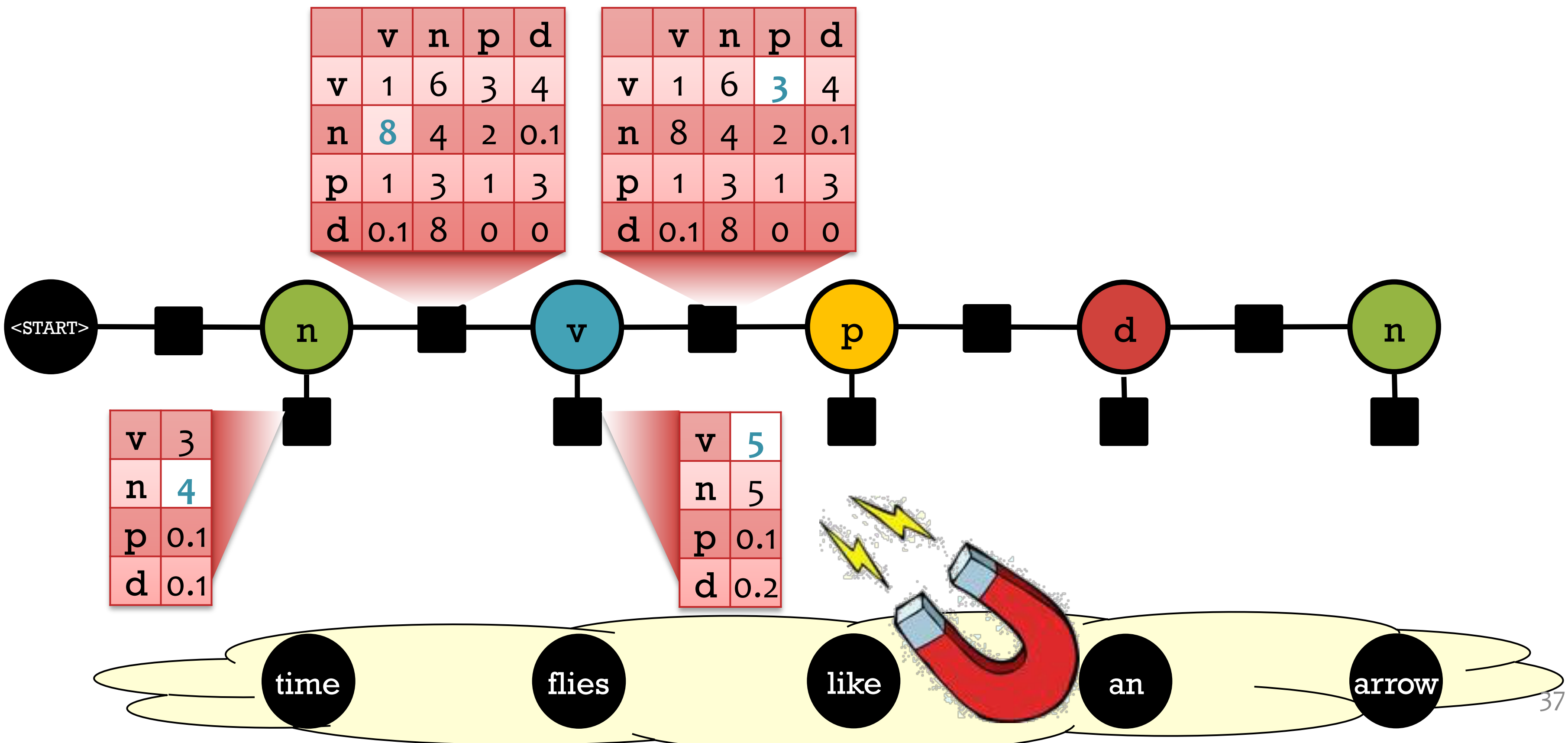
	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

Conditional Random Field (CRF)

Conditional distribution over tags X_i given words w_i.
The factors and Z are now specific to the sentence w.

$$p(\text{n, v, p, d, n} \mid \text{time, flies, like, an, arrow}) = \frac{1}{Z}(4 * 8 * 5 * 3 * \ldots)$$



	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

v	3
n	4
p	0.1
d	0.1

v	5
n	5
p	0.1
d	0.2

How General Are Factor Graphs?

- Factor graphs can be used to describe
  - Markov Random Fields (undirected graphical models)
    - i.e., log-linear models over a tuple of variables
  - Conditional Random Fields
  - Bayesian Networks (directed graphical models)

- *Inference* treats all of these interchangeably.
  - Convert your model to a factor graph first.
  - Pearl (1988) gave key strategies for *exact* inference:
    - **Belief propagation**, for inference on *acyclic* graphs
    - **Junction tree algorithm**, for making *any* graph acyclic (by merging variables and factors: blows up the runtime)

Factor Graph Notation

- **Variables:**

$$\mathcal{X} = \{X_1, \ldots, X_i, \ldots, X_n\}$$

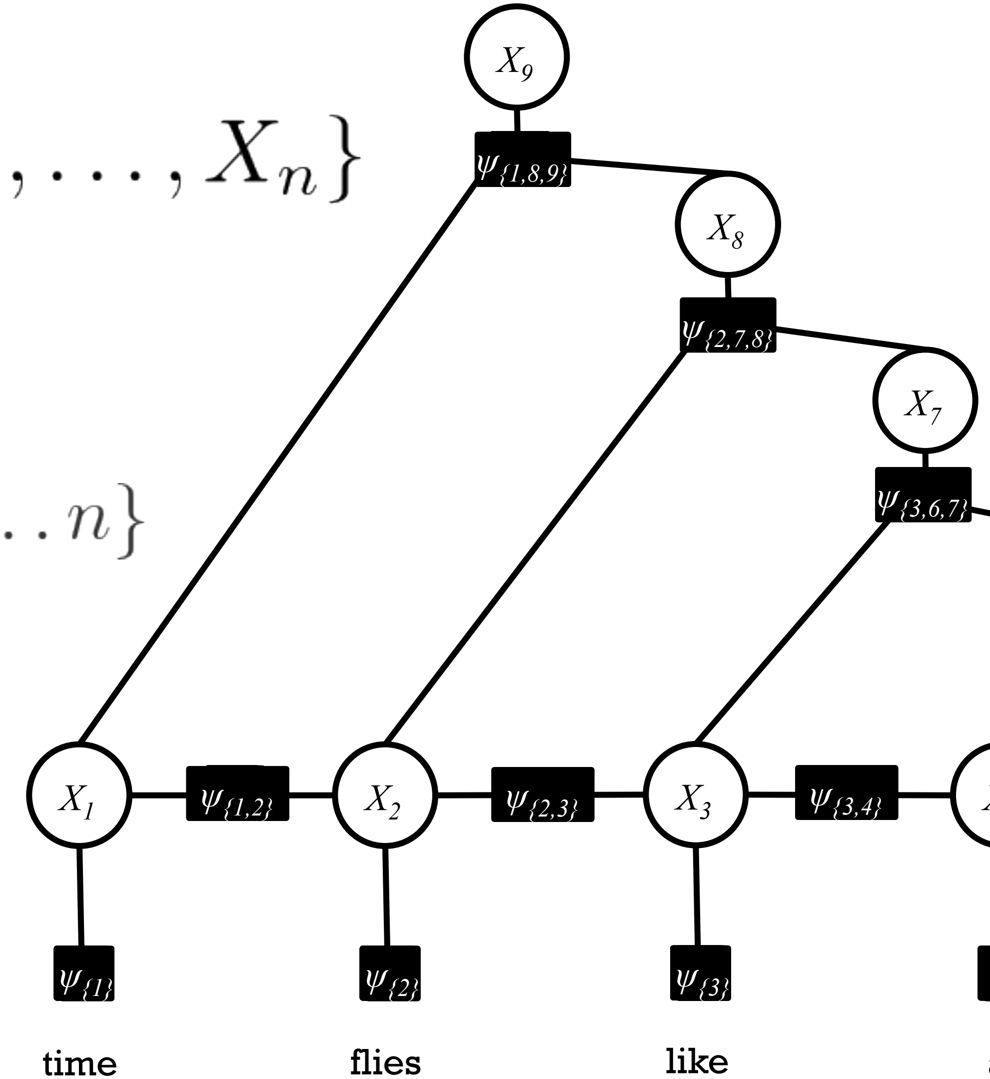
- **Factors:**

$$\psi_\alpha, \psi_\beta, \psi_\gamma, \ldots$$

where $\alpha, \beta, \gamma, \ldots \subseteq \{1, \ldots n\}$

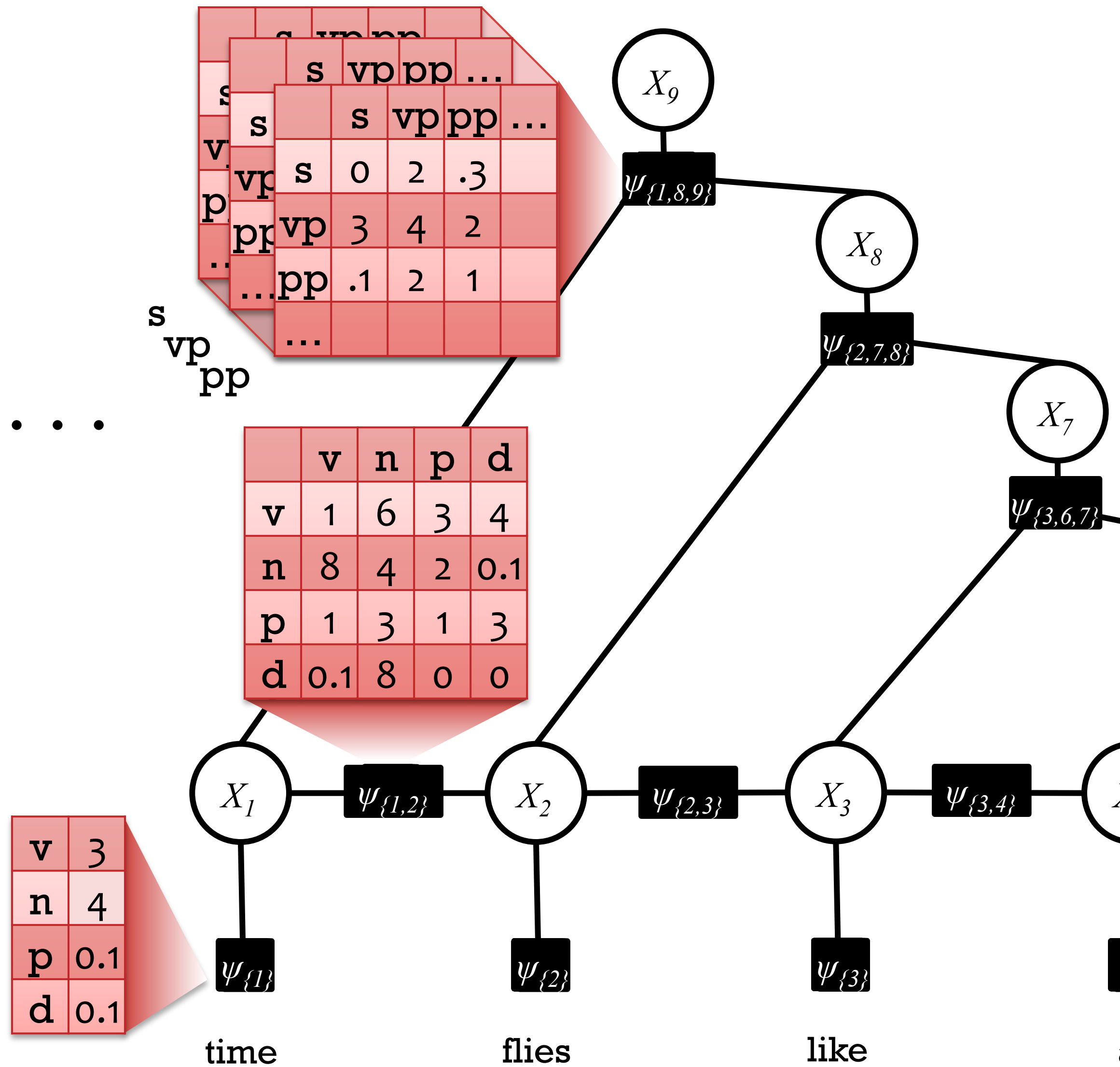
Joint Distribution

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_\alpha \psi_\alpha(\boldsymbol{x}_\alpha)$$



Factors are Tensors

- Factors:
  $$\psi_\alpha, \psi_\beta, \psi_\gamma, \ldots$$



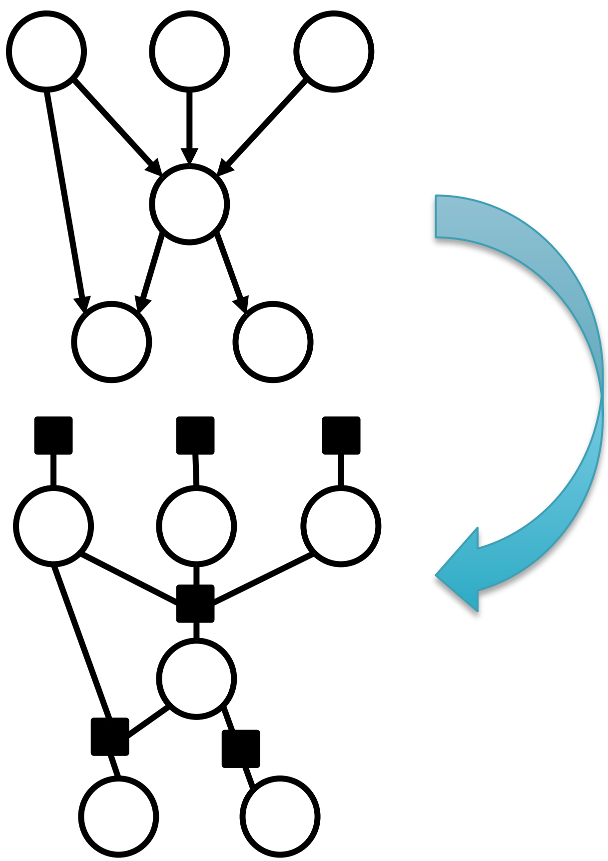
	s	vp	pp	…
s	0	2	.3	
vp	3	4	2	
pp	.1	2	1	
…				

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

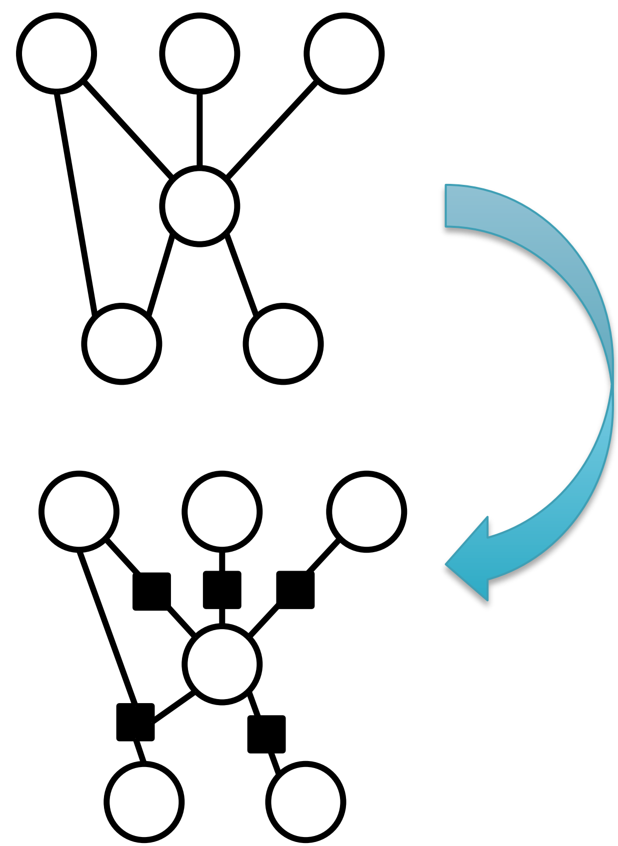
v	3
n	4
p	0.1
d	0.1

Converting to Factor Graphs

Each conditional and marginal distribution in a **directed GM** becomes a factor



Each clique in an **undirected GM** becomes a factor



Equivalence of directed and undirected trees

- Any undirected tree can be converted to a directed tree by choosing a root node and directing all edges away from it
- A directed tree and the corresponding undirected tree make the same conditional independence assertions
- Parameterizations are essentially the same.
  - Undirected tree:
  - Directed tree:
  - Equivalence:

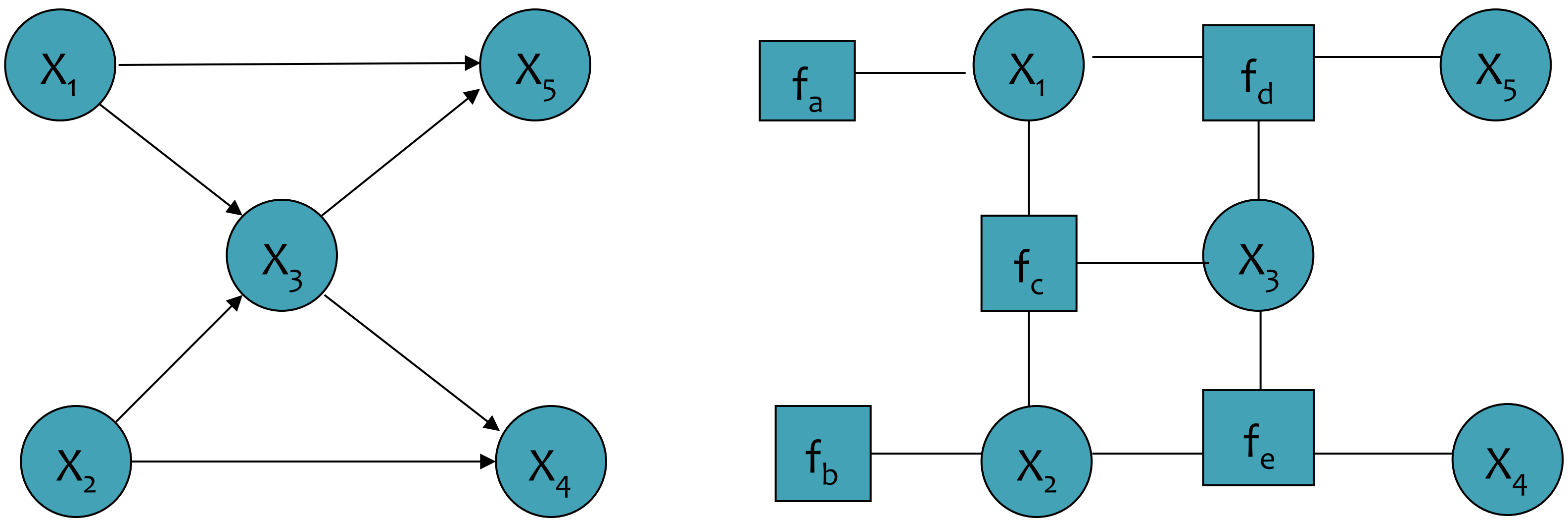
$$p(x) \quad = \quad \frac{1}{Z}\left(\prod_{i \in V} \psi(x_i) \prod_{(i,j) \in E} \psi(x_i, x_j)\right)$$

$$p(x) = p(x_r) \prod_{(i,j) \in E} p(x_j | x_i)$$

$$\psi(x_r) = p(x_r); \quad \psi(x_i, x_j) = p(x_j | x_i);$$
$$Z = 1, \quad \psi(x_i) = 1$$

Factor Graph Examples

- Example 1



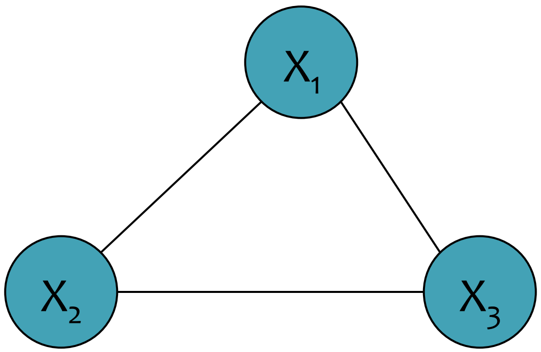
$P(X_1)$ $P(X_2)$ $P(X_3|X_1,X_2)$ $P(X_5|X_1,X_3)$ $P(X_4|X_2,X_3)$

↓ ↓ ↓ ↓ ↓

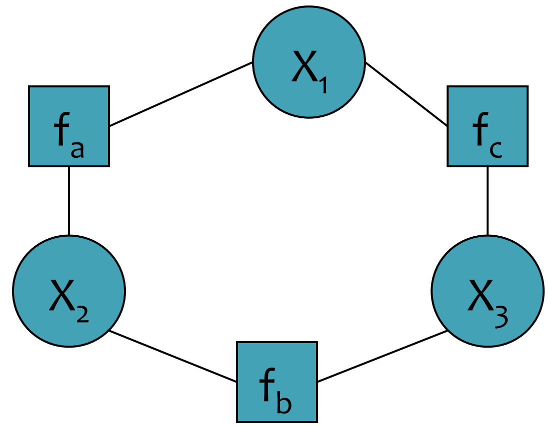
$f_a(X_1)$ $f_b(X_2)$ $f_c(X_3,X_1,X_2)$ $f_d(X_5,X_1,X_3)$ $f_e(X_4,X_2,X_3)$

Factor Graph Examples

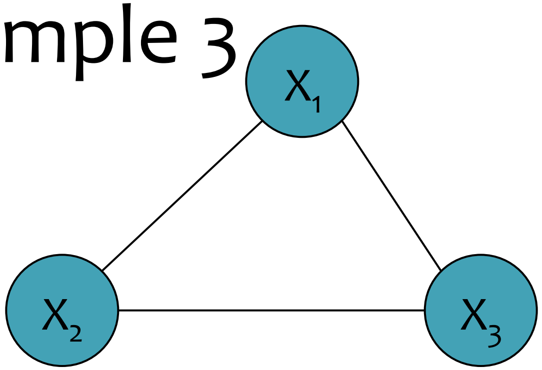
- Example 2



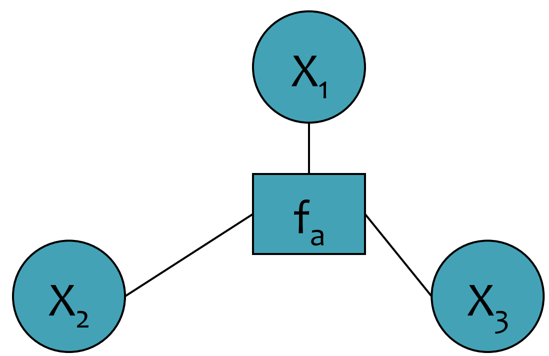
$\psi(x_1,x_2,x_3) = f_a(x_1,x_2)f_b(x_2,x_3)f_c(x_3,x_1)$



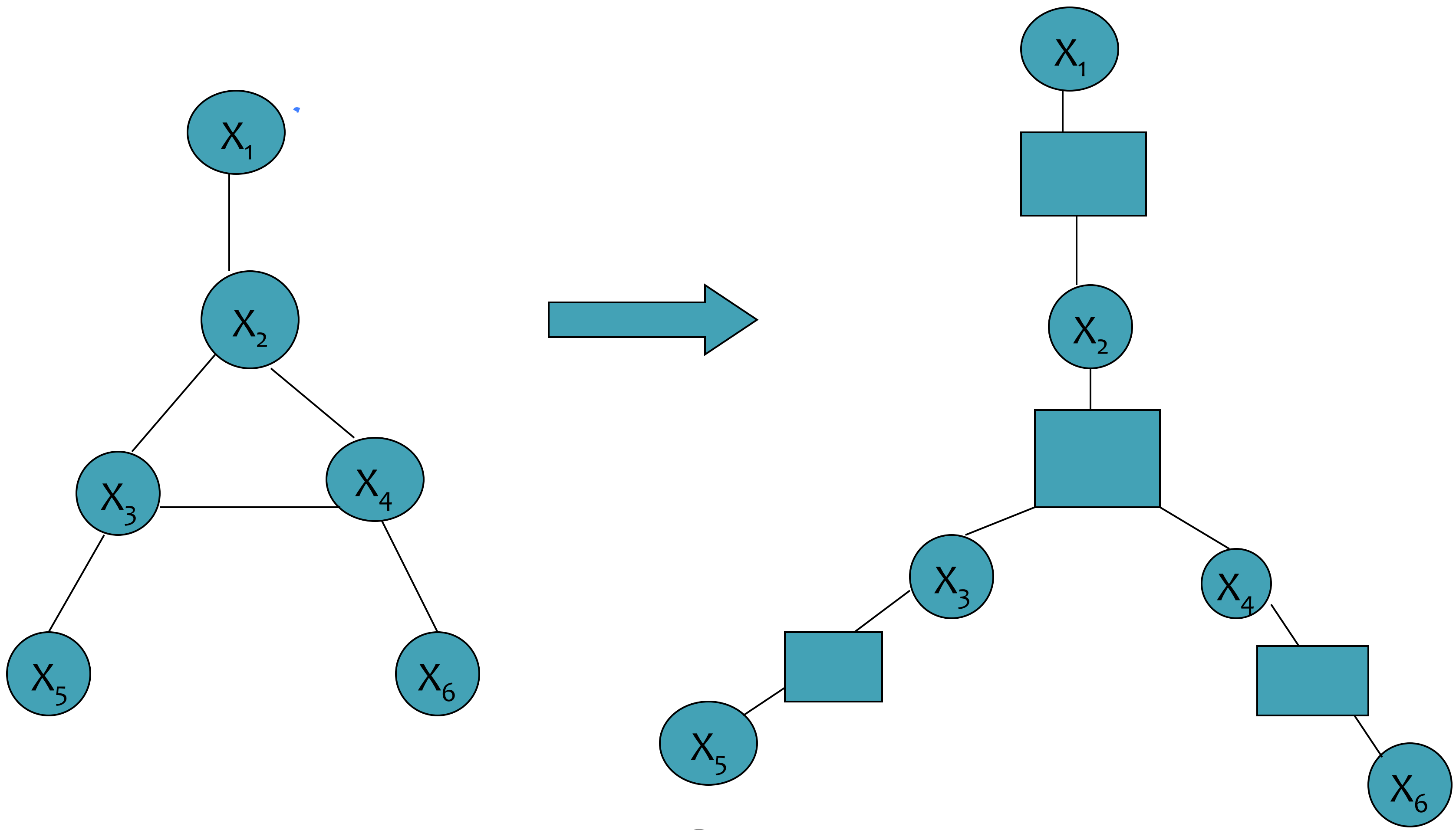
- Example 3



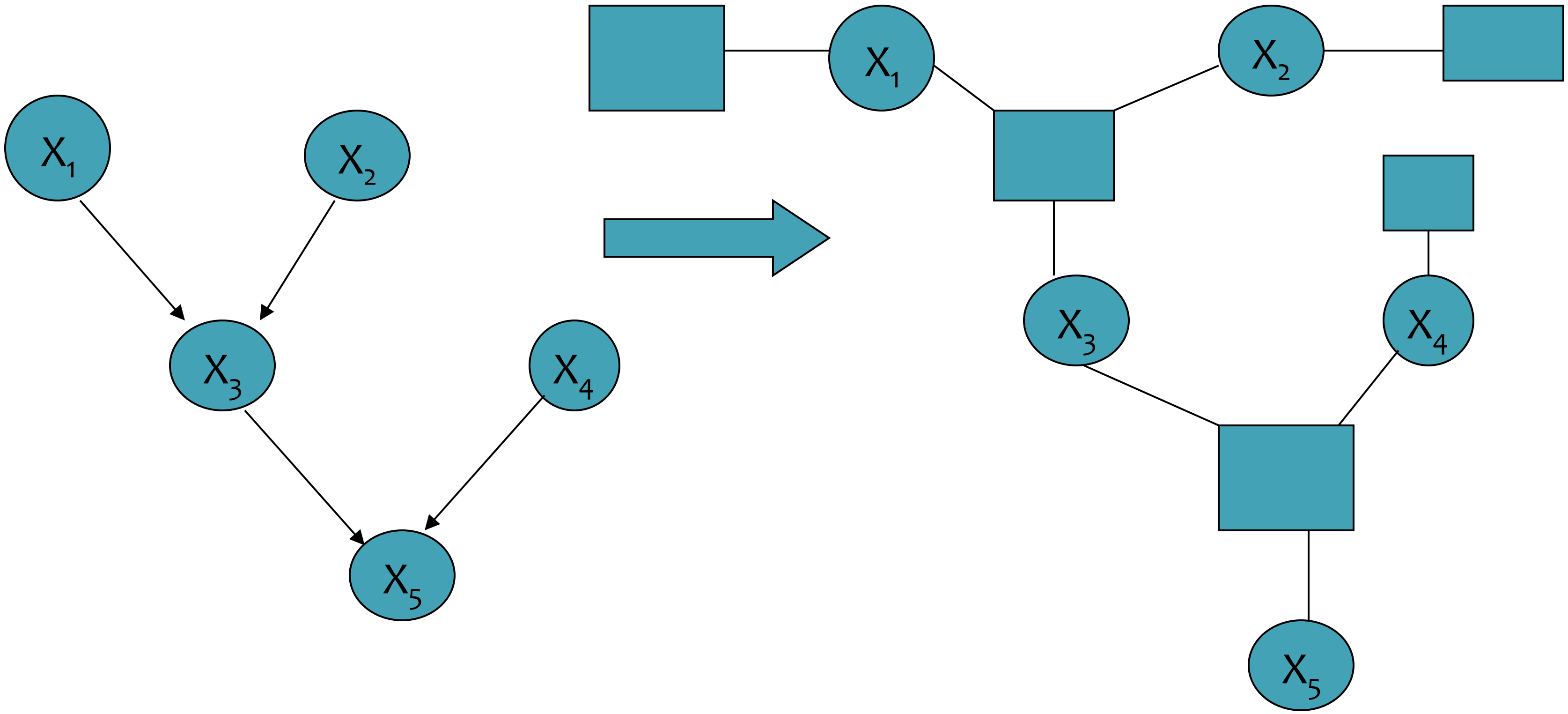
$\psi(x_1,x_2,x_3) = f_a(x_1,x_2,x_3)$



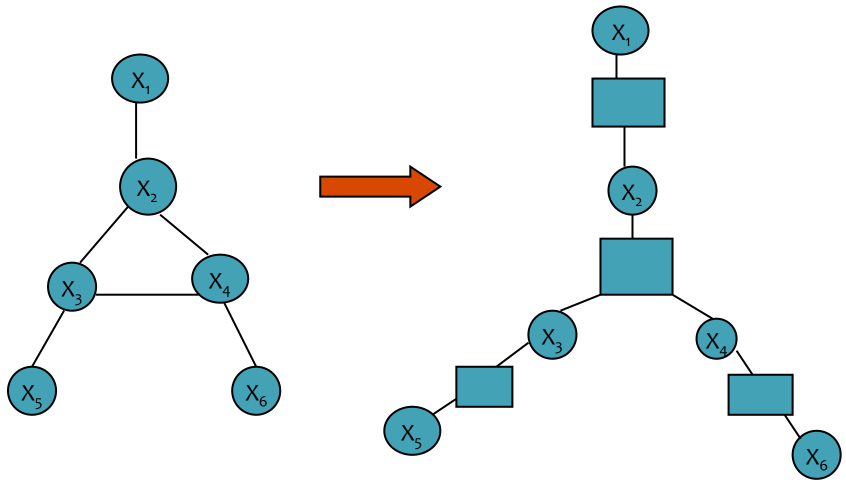
Tree-like Undirected GMs to Factor Trees

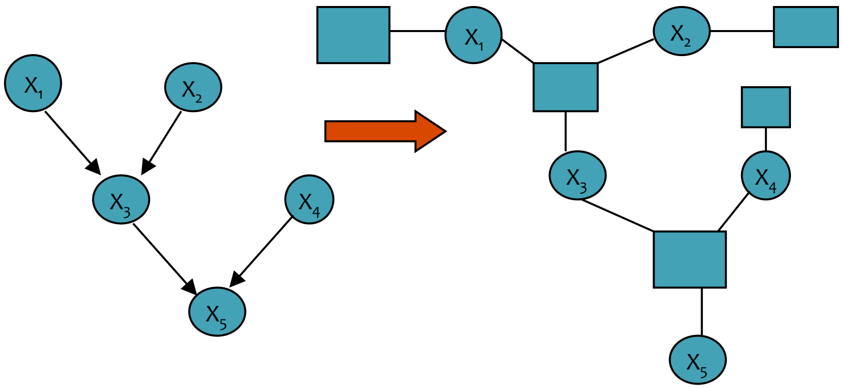


Poly-trees to Factor trees



Why factor graphs?





- Because FG turns tree-like graphs to factor trees,
- Trees are a data-structure that guarantees correctness of BP !

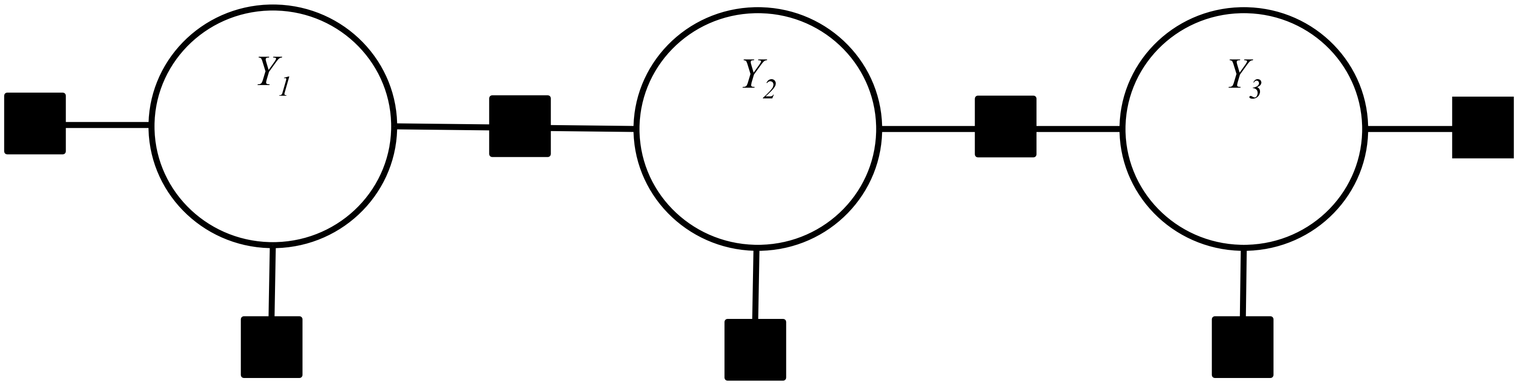
THE FORWARD-BACKWARD ALGORITHM

Learning and Inference Summary

For discrete variables:

	Learning	Marginal Inference	MAP Inference
HMM		Forward-backward	Viterbi
Linear-chain CRF		Forward-backward	Viterbi

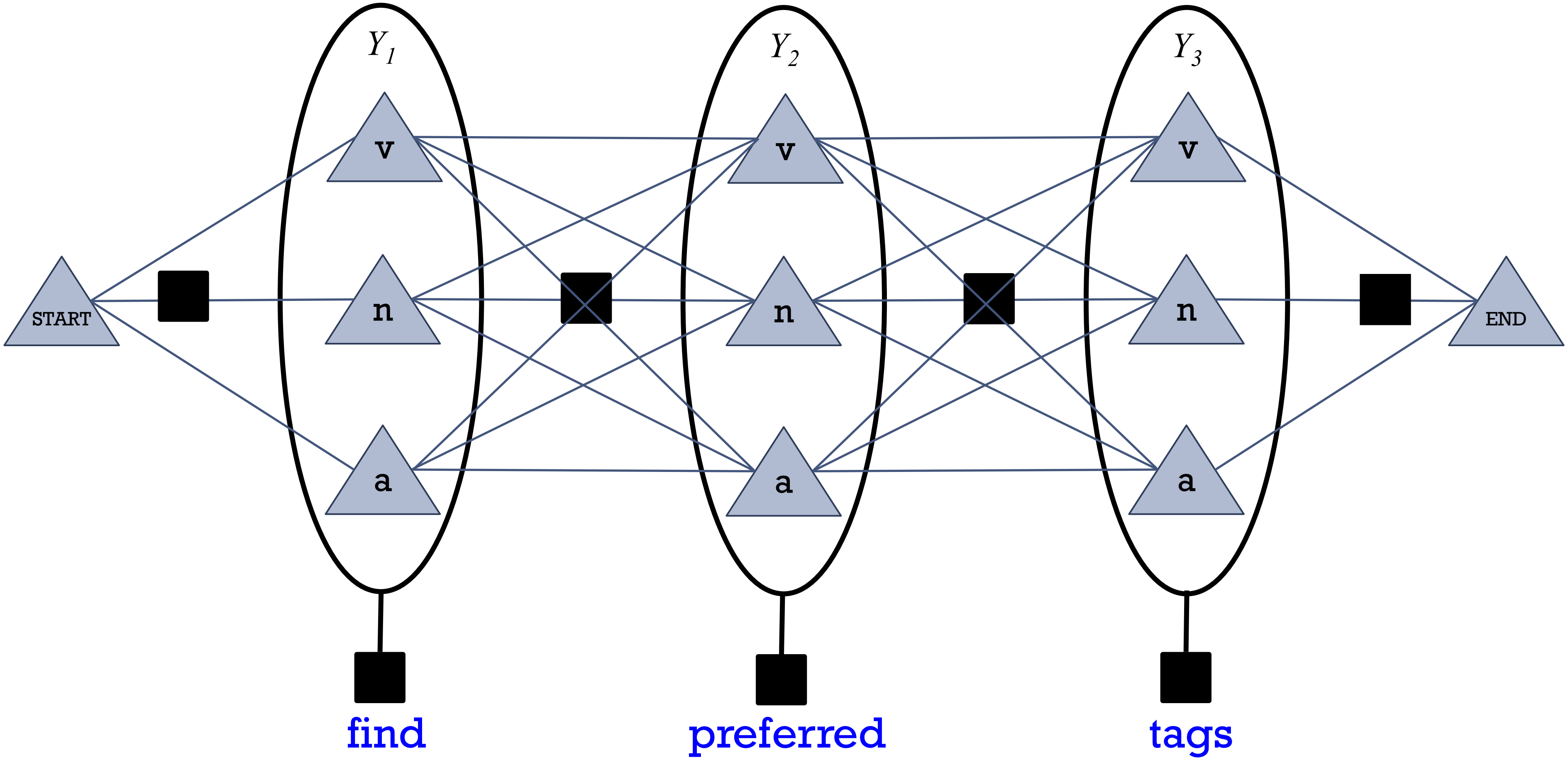
Forward-Backward Algorithm



find | preferred | tags

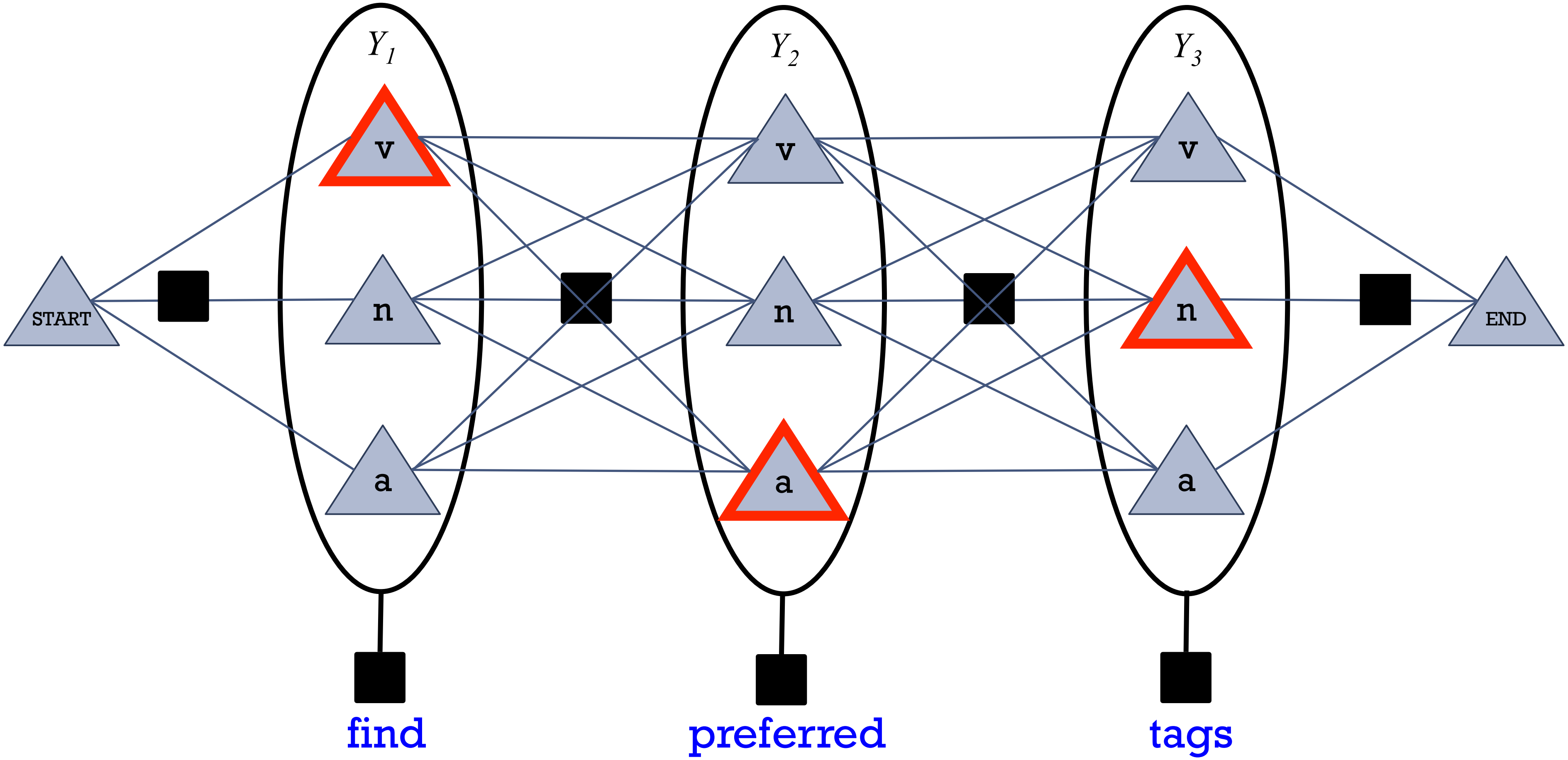
Could be verb or noun | *Could be adjective or verb* | *Could be noun or verb*

Forward-Backward Algorithm



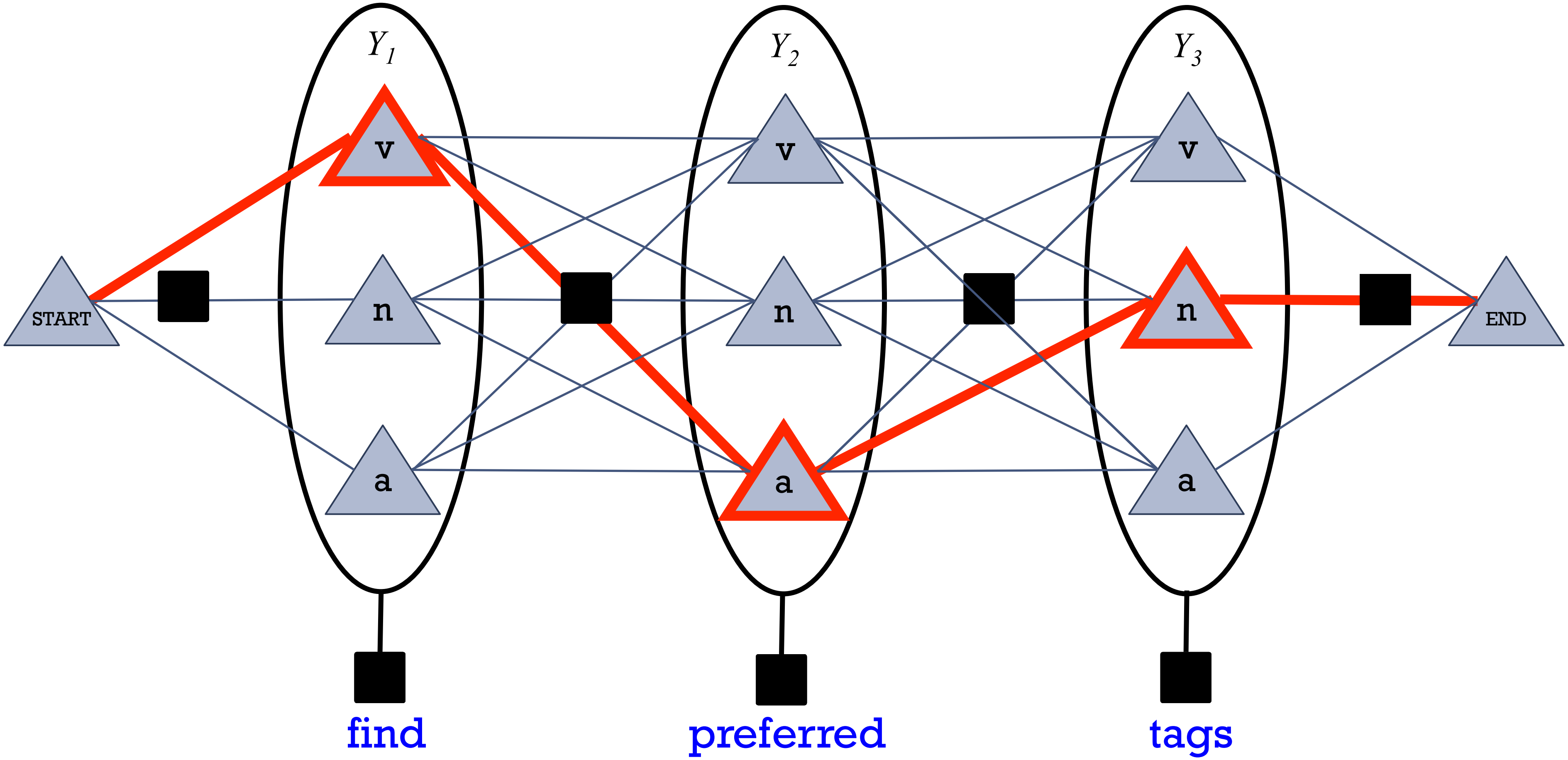
- Show the possible *values* for each variable

Forward-Backward Algorithm



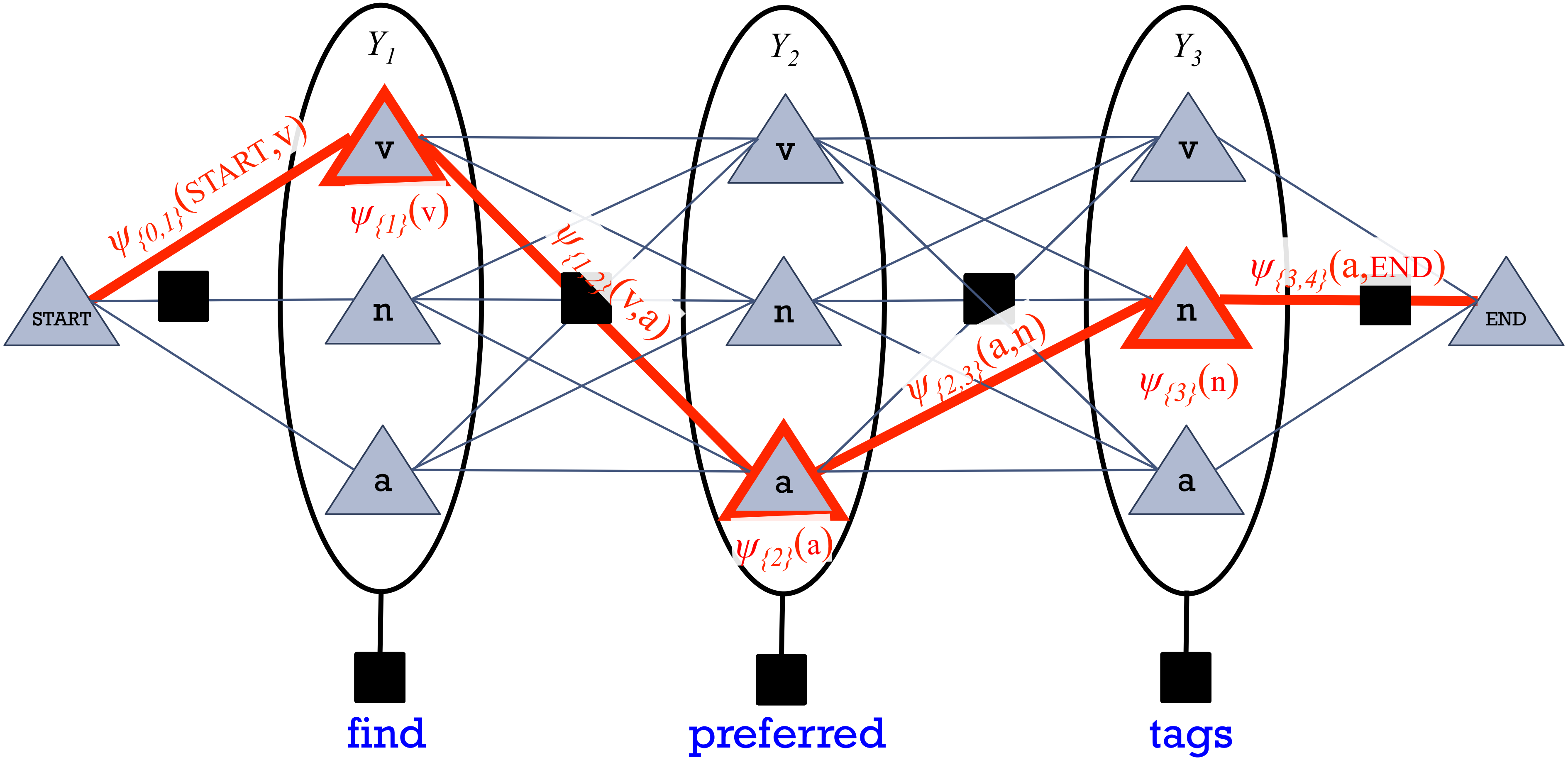
- Let's show the possible *values* for each variable
- One possible assignment

Forward-Backward Algorithm



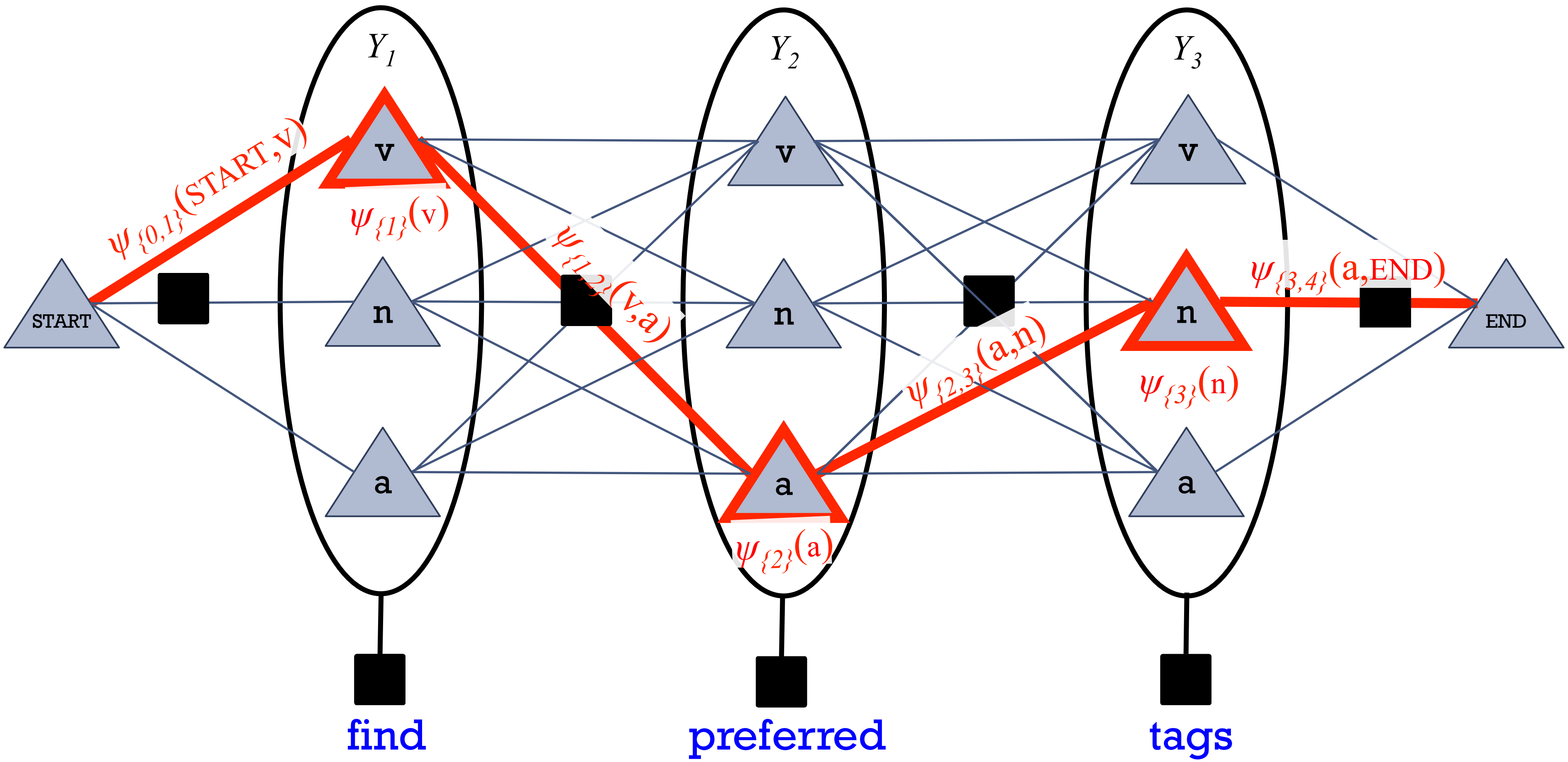
- Let's show the possible *values* for each variable
- One possible assignment
- And what the 7 factors think of it ...

Viterbi Algorithm: Most Probable Assignment



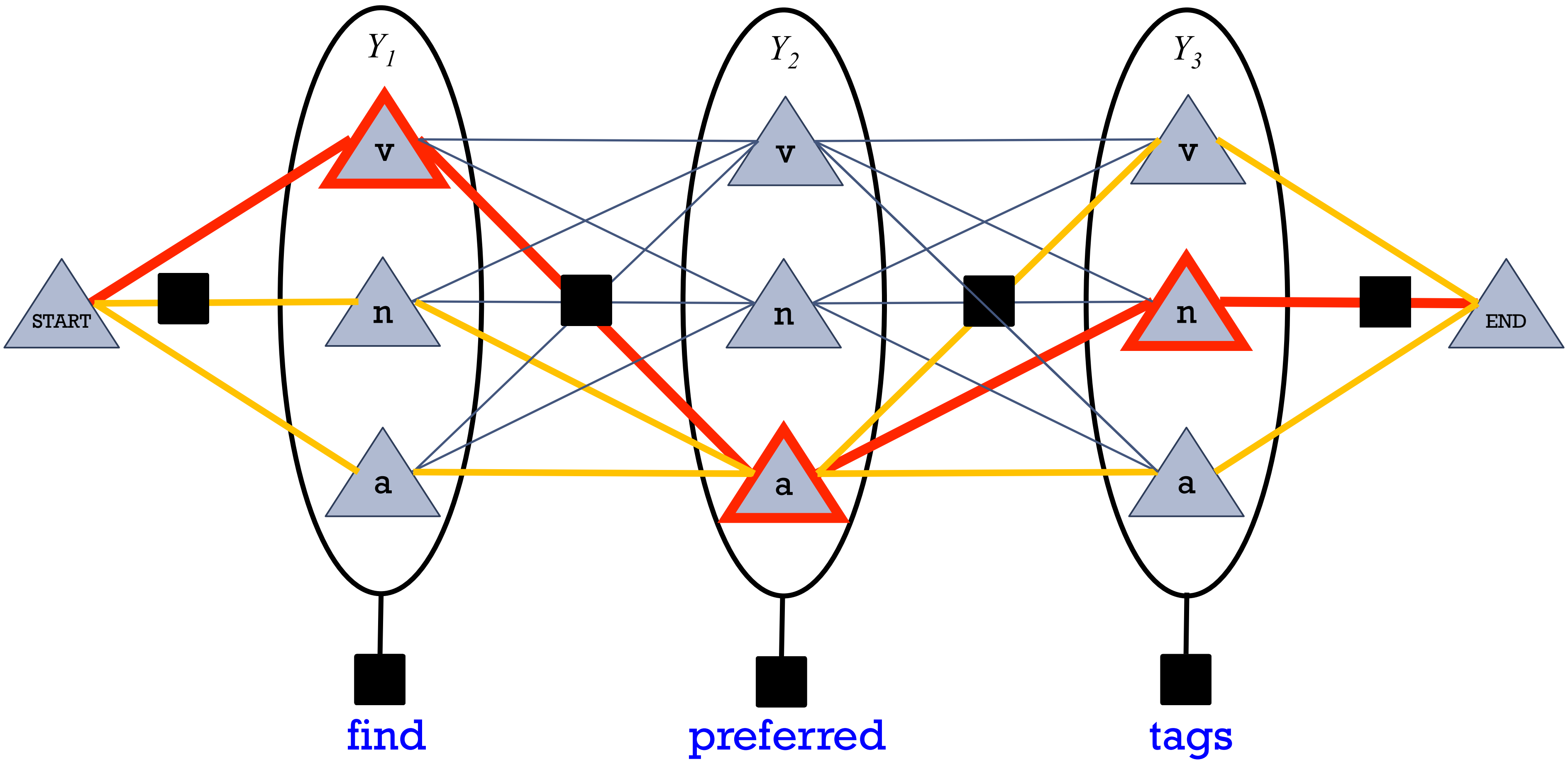
- So p(**v a n**) = (1/Z) * product of 7 numbers
- Numbers associated with edges and nodes of path
- Most probable assignment = **path with highest product**

Viterbi Algorithm: Most Probable Assignment



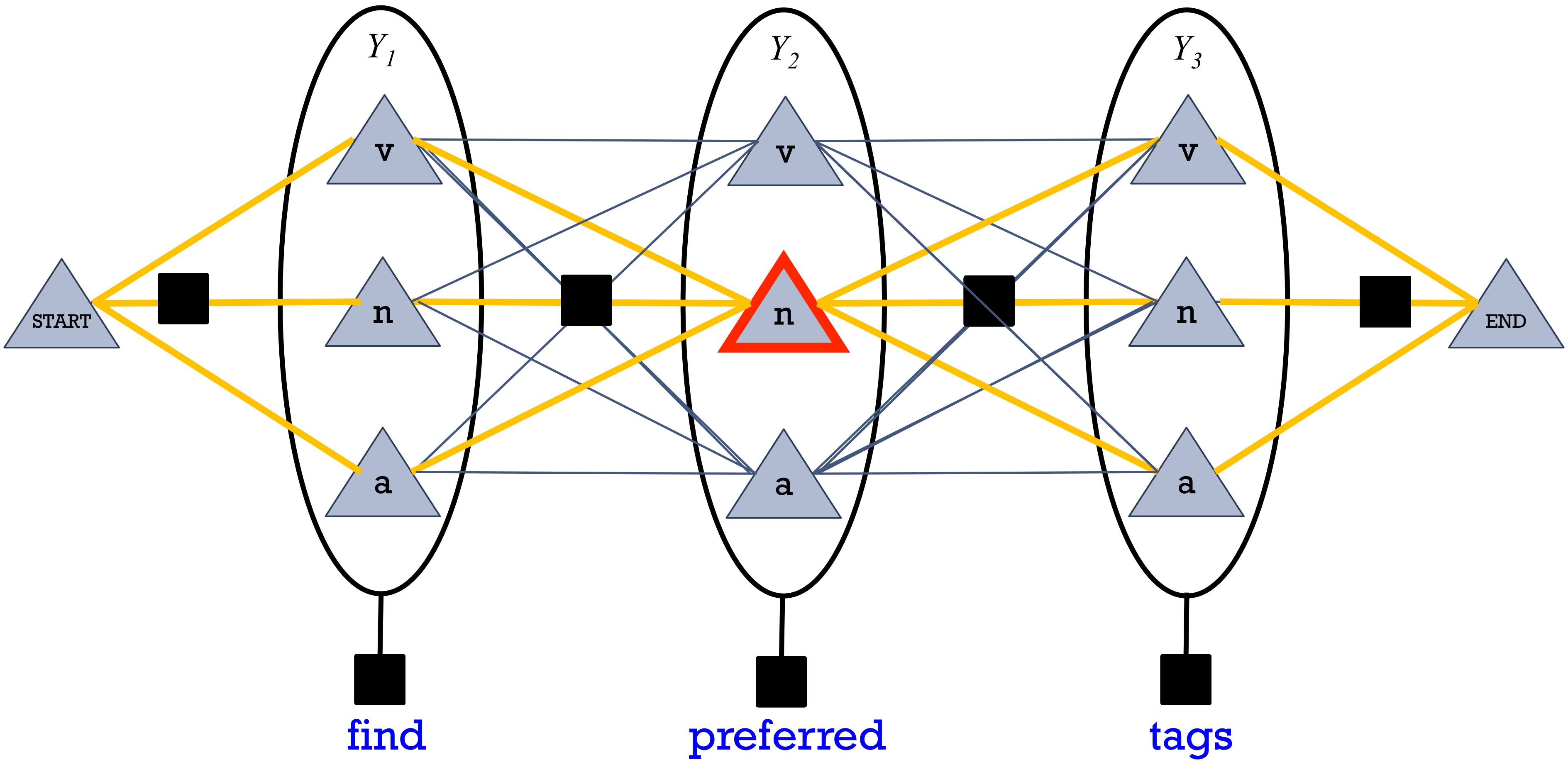
- So p(**v a n**) = (1/Z) * product weight of one path

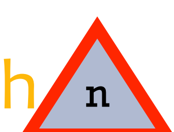
Forward-Backward Algorithm: Finds Marginals



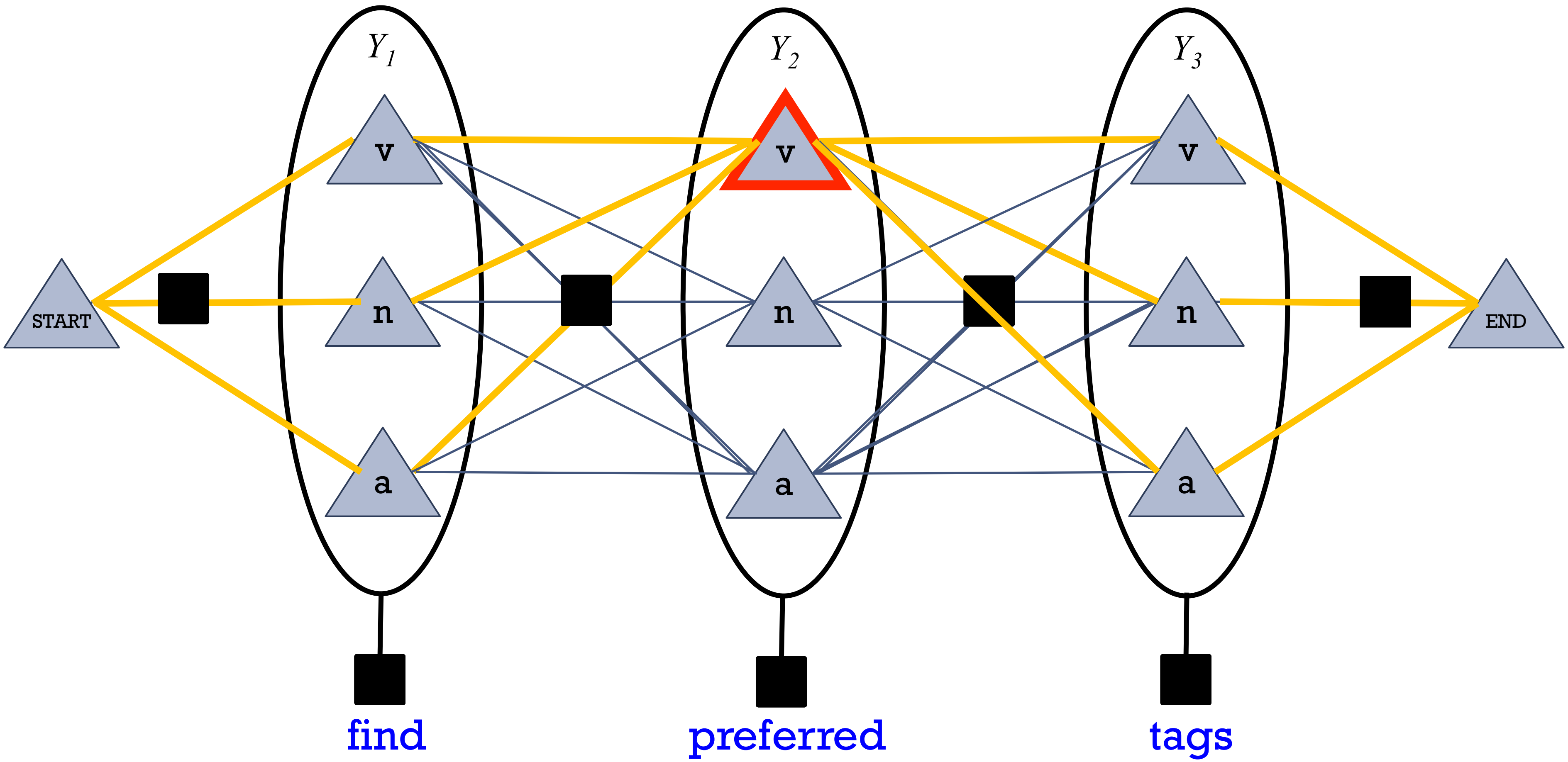
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability p(Y_2 = a)
  = (1/Z) * total weight of *all* paths through a

Forward-Backward Algorithm: Finds Marginals



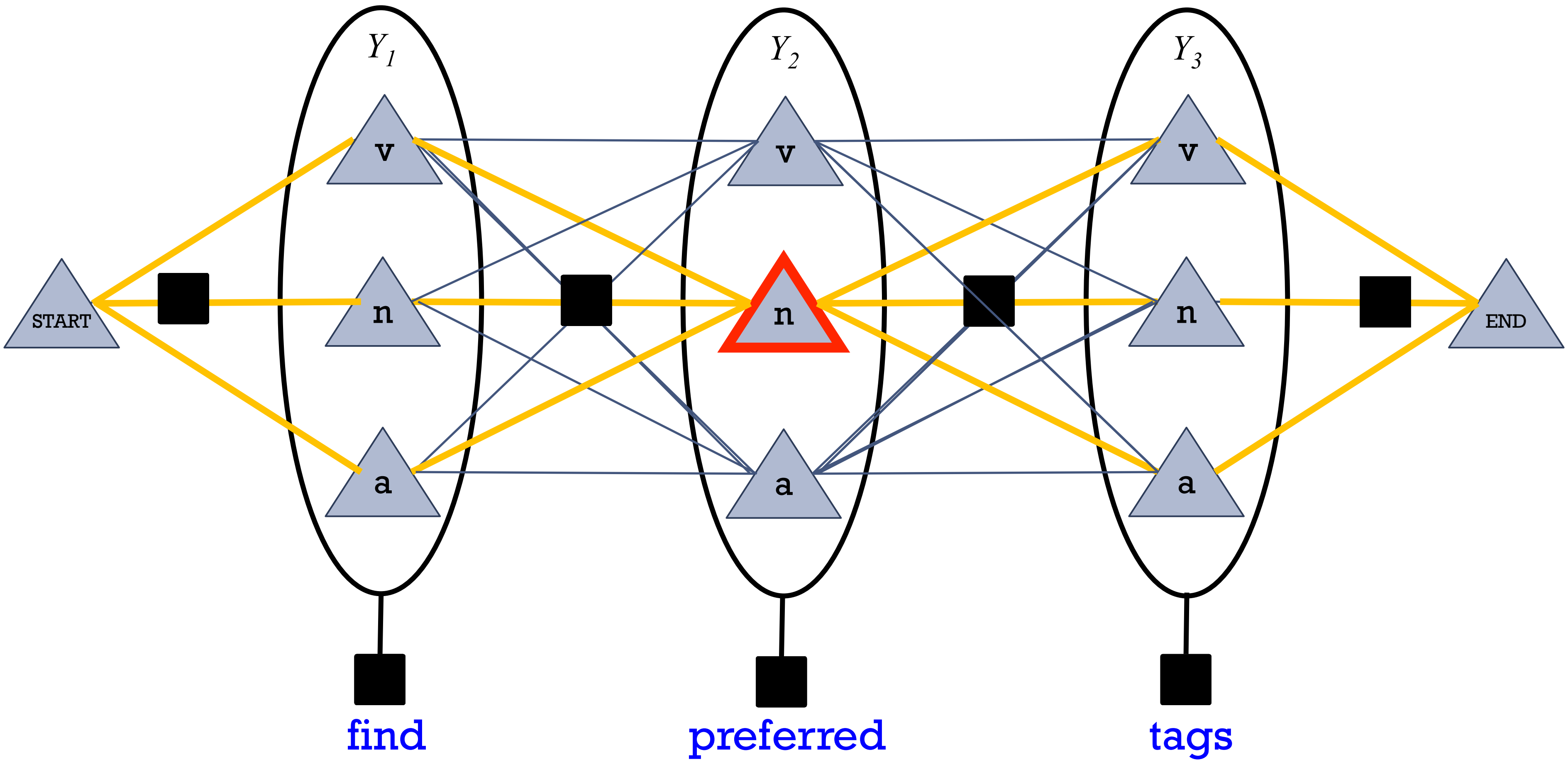
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability $p(Y_2 = a)$
  = (1/Z) * total weight of *all* paths through 

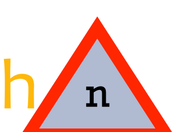
Forward-Backward Algorithm: Finds Marginals



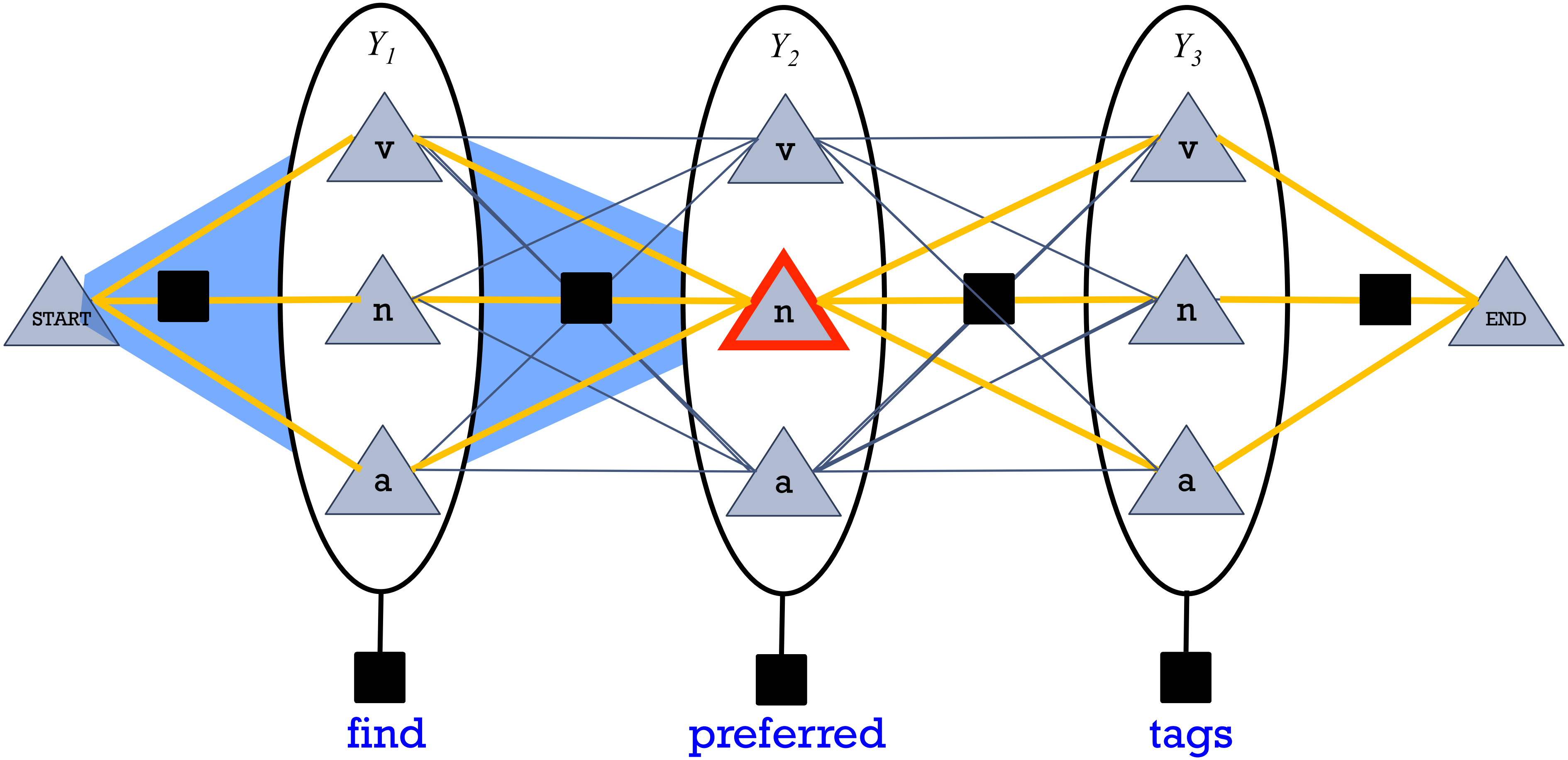
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability $p(Y_2 = a)$
  = (1/Z) * total weight of *all* paths through **v**

Forward-Backward Algorithm: Finds Marginals



- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability p(Y_2 = a)
  = (1/Z) * total weight of *all* paths through 

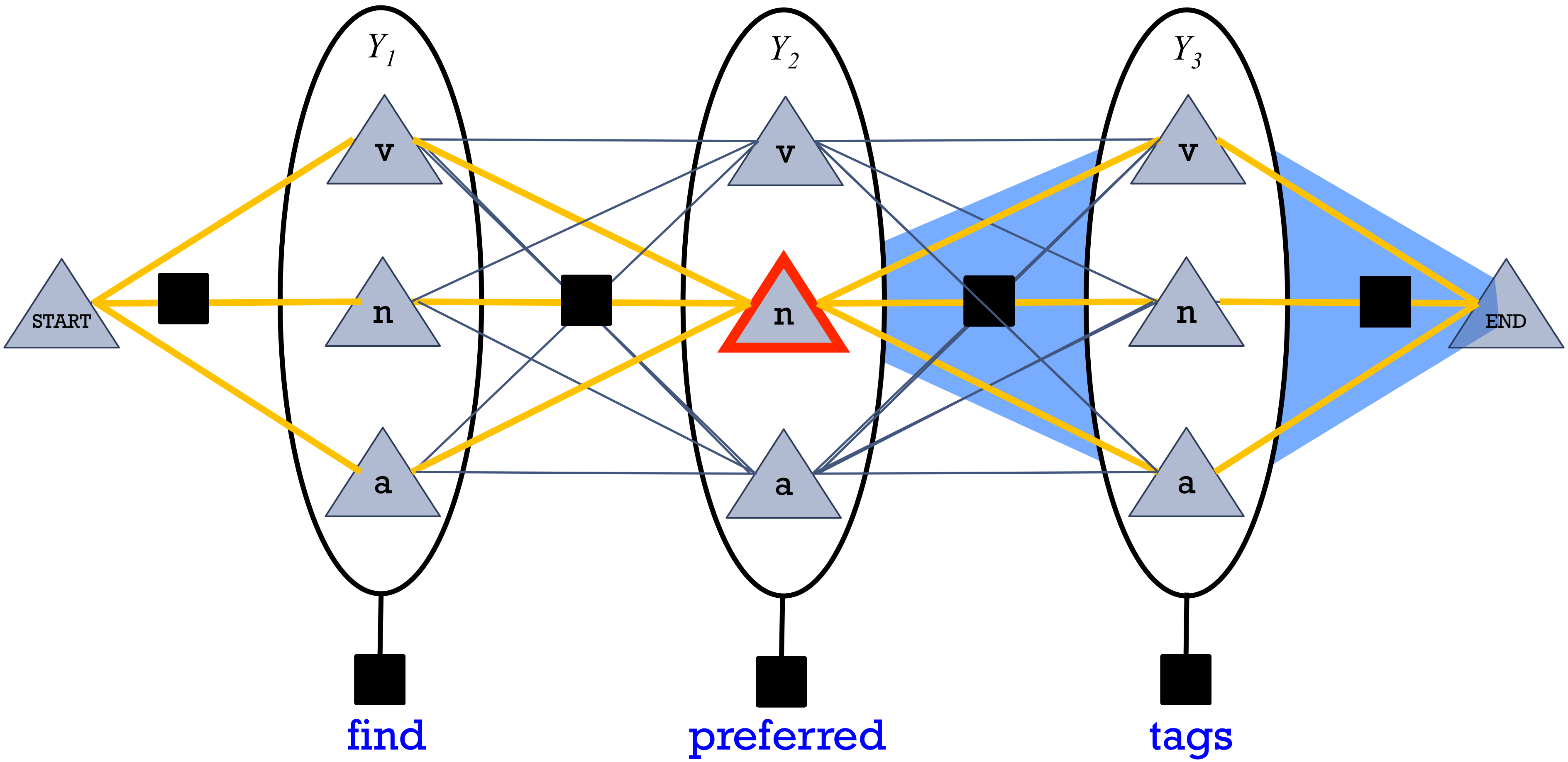
Forward-Backward Algorithm: Finds Marginals



$\alpha_2(\mathbf{n})$ = total weight of these path *prefixes*

(found by dynamic programming: matrix-vector products)

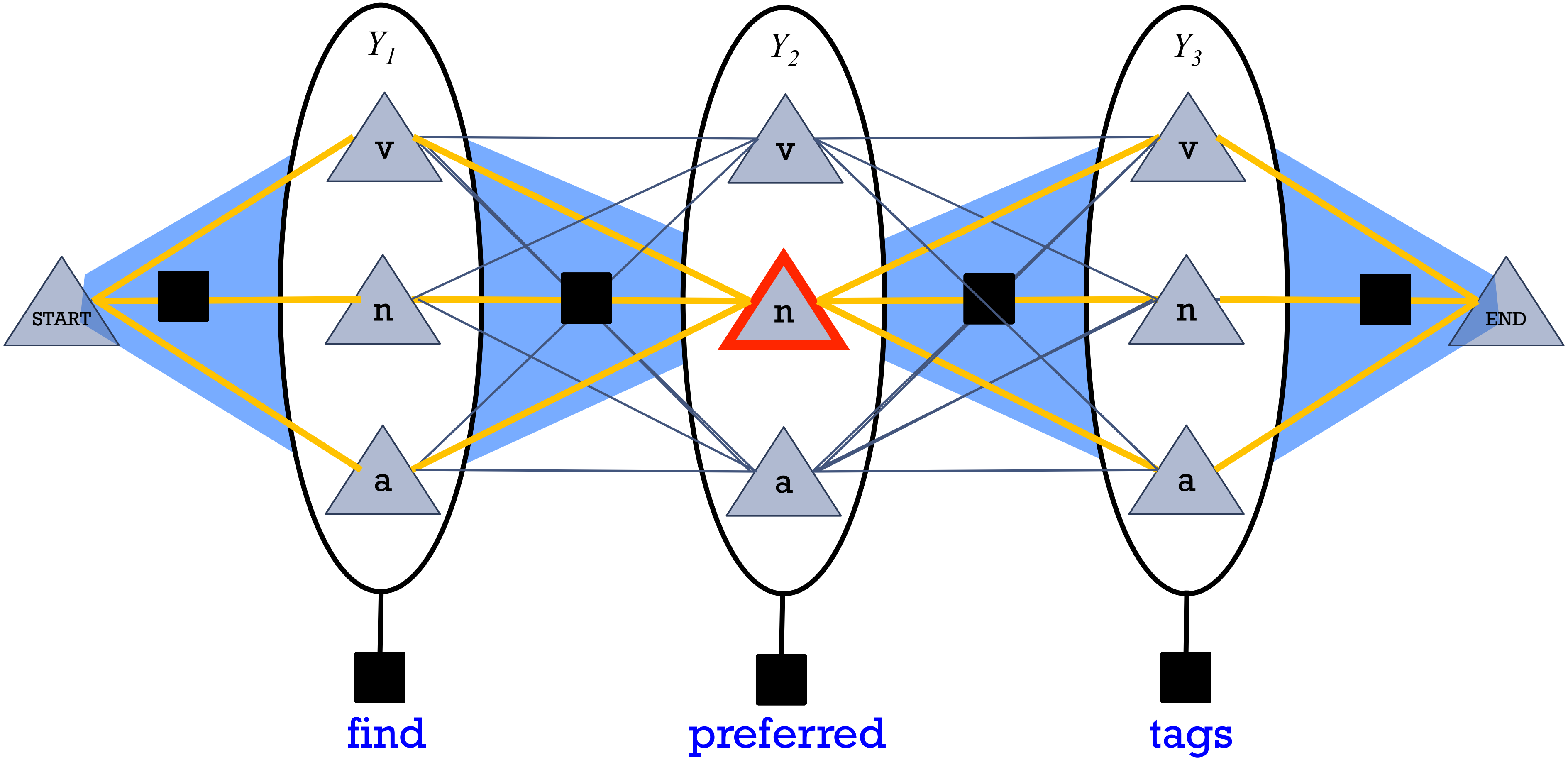
Forward-Backward Algorithm: Finds Marginals



$\beta_2(\mathbf{n})$ = total weight of these path *suffixes*

(found by dynamic programming: matrix-vector products)

Forward-Backward Algorithm: Finds Marginals



$\alpha_2(\mathbf{n})$ = total weight of these path *prefixes* (a + b + c)

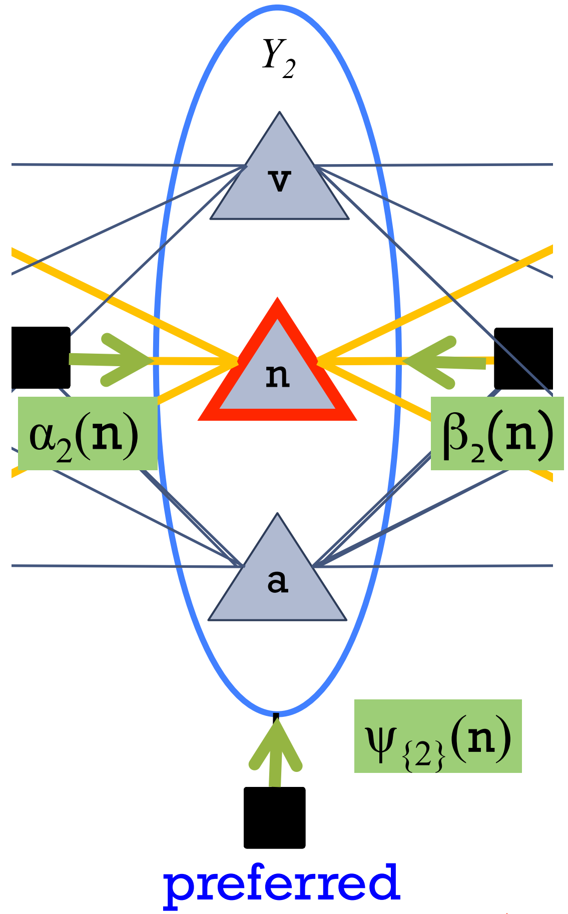
$\beta_2(\mathbf{n})$ = total weight of these path *suffixes* (x + y + z)

Product gives ax+ay+az+bx+by+bz+cx+cy+cz = total weight of paths

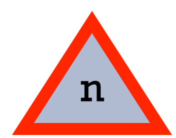
Forward-Backward Algorithm: Finds Marginals

Oops! The weight of a path through a state also includes a weight at that state.
So $\alpha(\mathbf{n})\cdot\beta(\mathbf{n})$ isn't enough.

The extra weight is the opinion of the unigram factor at this variable.

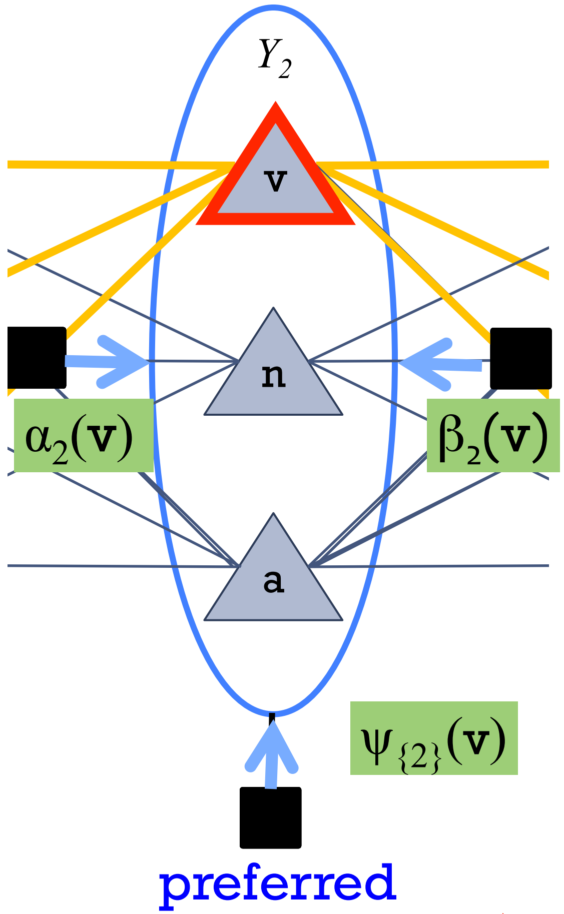


"belief that $Y_2 = \mathbf{n}$"

total weight of *all* paths through 

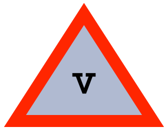
$= \alpha_2(\mathbf{n}) \; \psi_{\{2\}}(\mathbf{n}) \; \beta_2(\mathbf{n})$

Forward-Backward Algorithm: Finds Marginals



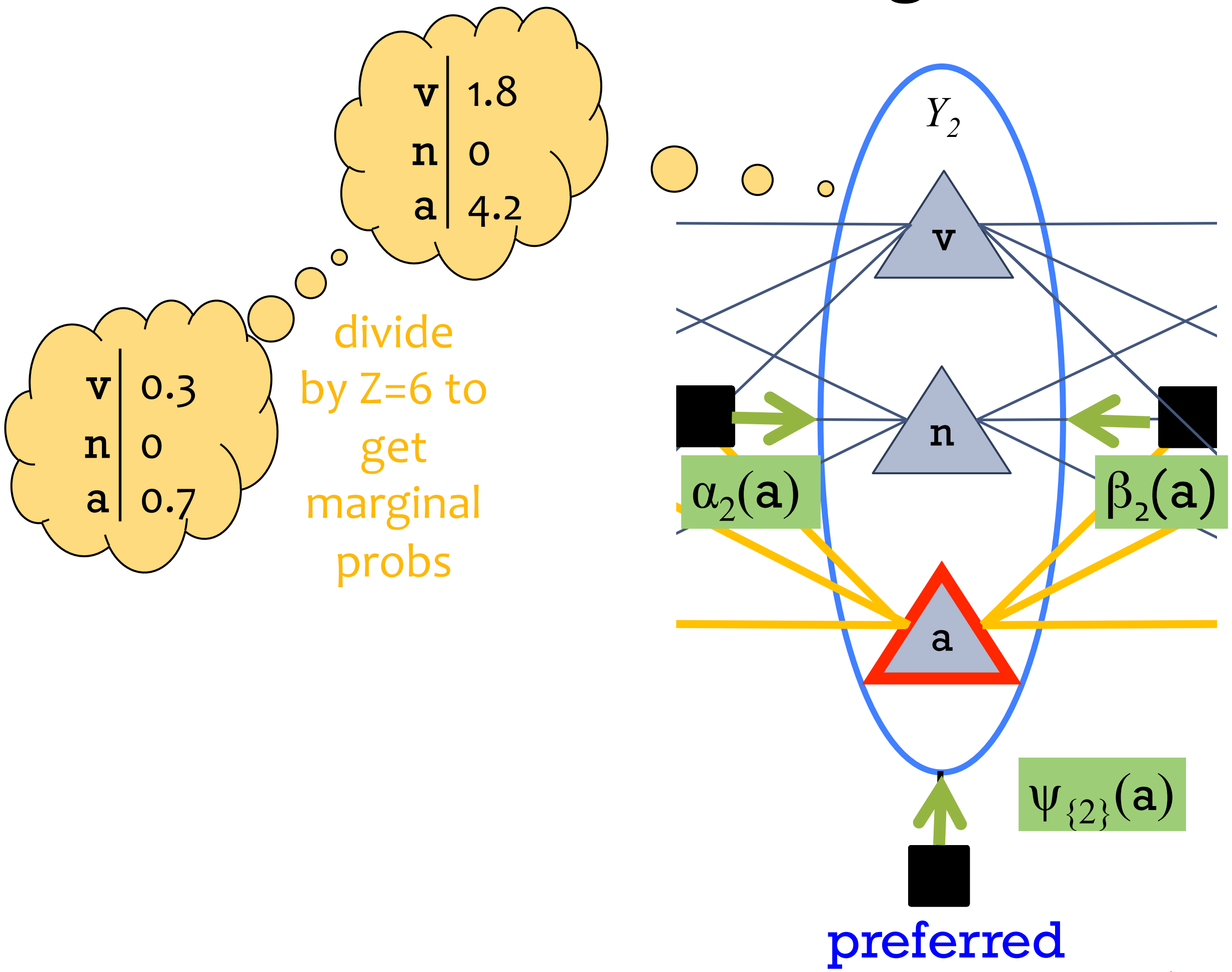
"belief that $Y_2 = \mathbf{v}$"

"belief that $Y_2 = \mathbf{n}$"

total weight of *all* paths through 

= $\alpha_2(\mathbf{v})$ $\psi_{\{2\}}(\mathbf{v})$ $\beta_2(\mathbf{v})$

Forward-Backward Algorithm: Finds Marginals



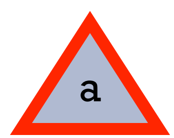
"belief that Y_2 = **v**"

"belief that Y_2 = **n**"

"belief that Y_2 = **a**"

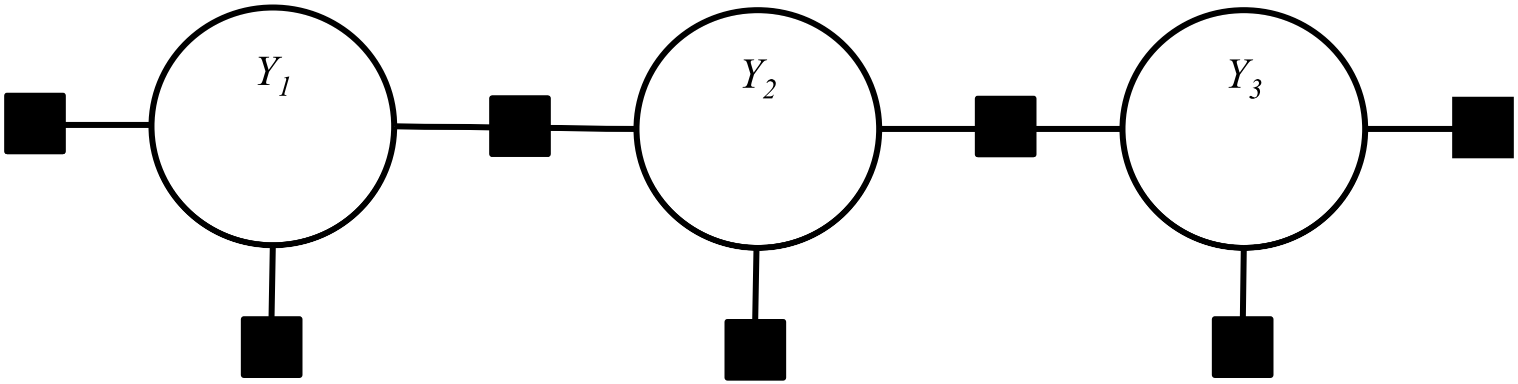
---

sum = Z
(total probability of *all* paths)

total weight of *all* paths through 

= $\alpha_2(\mathtt{a})$ $\psi_{\{2\}}(\mathtt{a})$ $\beta_2(\mathtt{a})$

CRF Tagging Model



find preferred tags

Could be verb or noun *Could be adjective or verb* *Could be noun or verb*

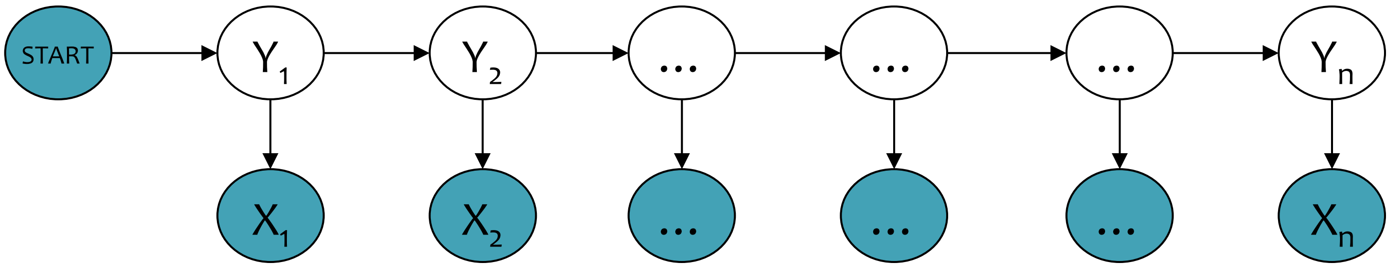
Whiteboard

- Forward-backward algorithm
- Viterbi algorithm

Conditional Random Fields (CRFs) for time series data

LINEAR-CHAIN CRFS

Shortcomings of Hidden Markov Models



- HMM models capture dependences between each state and only its corresponding observation
  - NLP example: In a sentence segmentation task, each segmental state may depend not just on a single word (and the adjacent segmental stages), but also on the (non-local) features of the whole line such as line length, indentation, amount of white space, etc.
- Mismatch between learning objective function and prediction objective function
  - HMM learns a joint distribution of states and observations P(**Y**, **X**), but in a prediction task, we need the conditional probability P(**Y**|**X**)

Conditional Random Field (CRF)

Conditional distribution over tags X_i given words w_i.
The factors and Z are now specific to the sentence w.

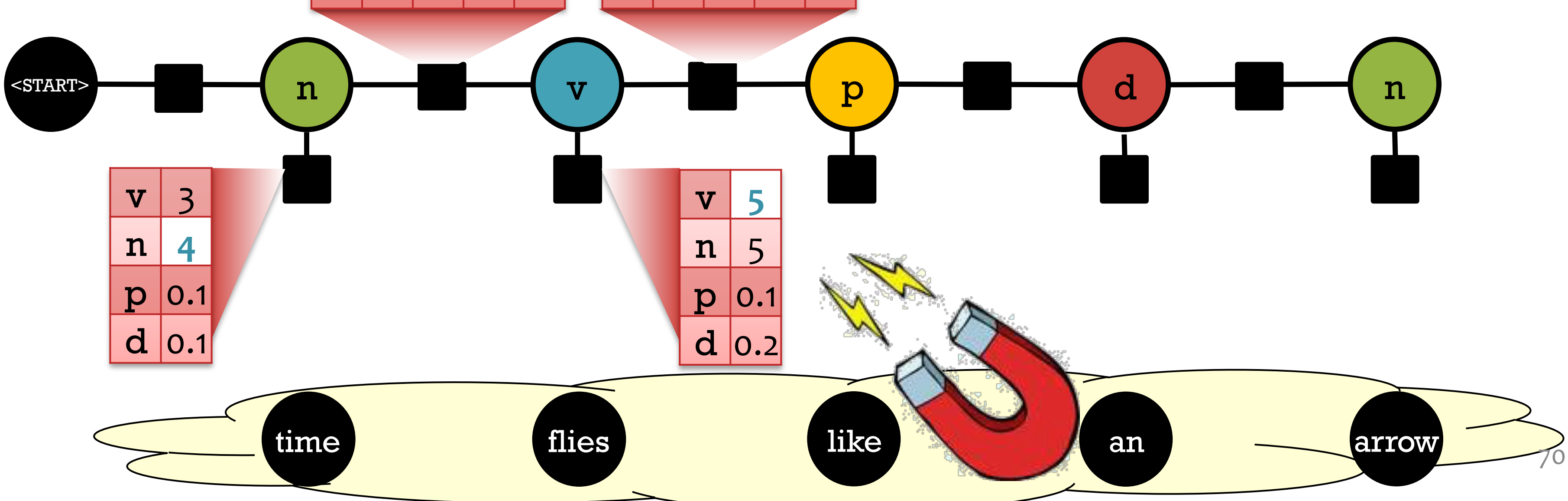
$$p(\text{n, v, p, d, n} \mid \text{time, flies, like, an, arrow}) = \frac{1}{Z}(4 * 8 * 5 * 3 * \ldots)$$

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

v	3
n	4
p	0.1
d	0.1

v	5
n	5
p	0.1
d	0.2



Conditional Random Field (CRF)

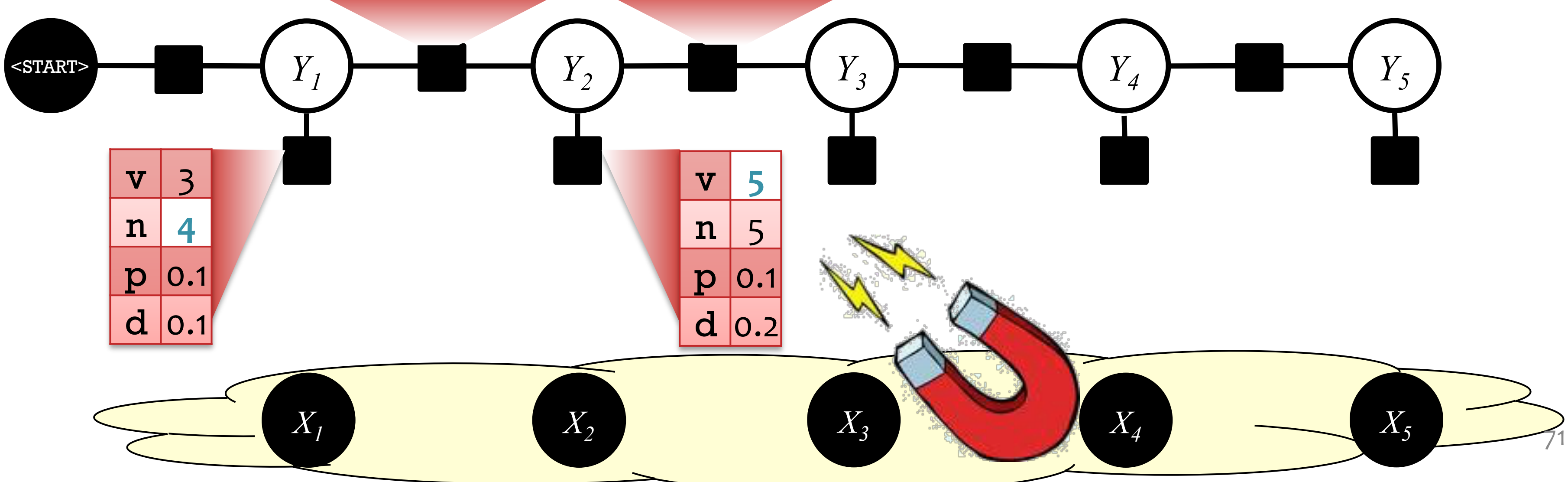
Recall: Shaded nodes in a graphical model are **observed**

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

v	3
n	4
p	0.1
d	0.1

v	5
n	5
p	0.1
d	0.2

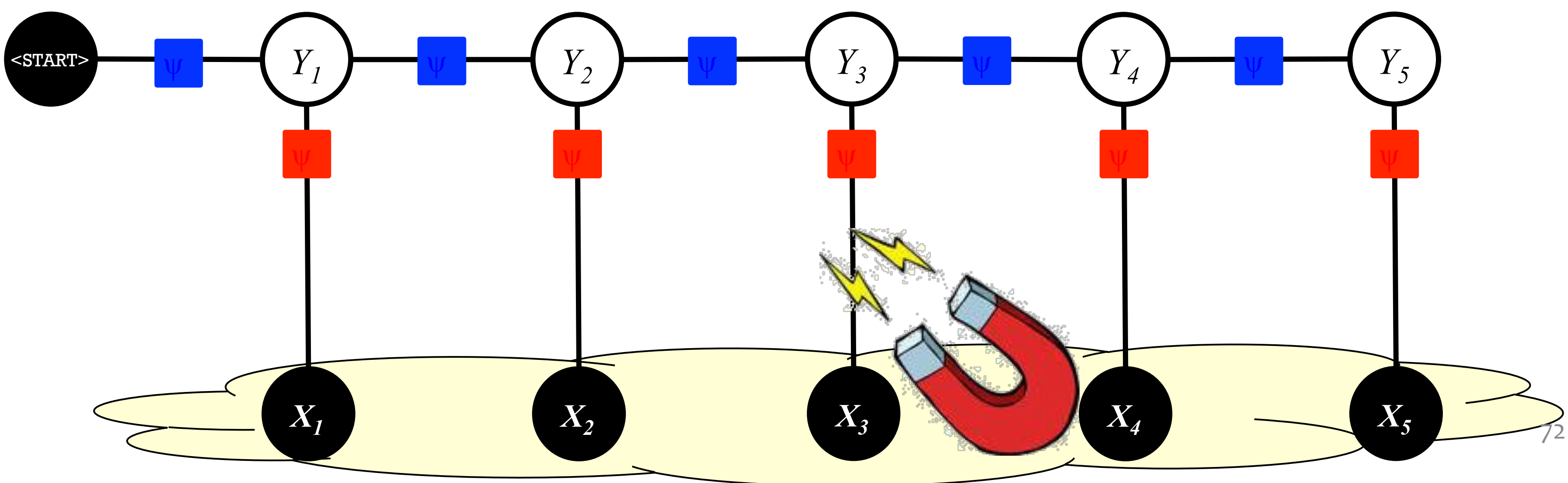


Conditional Random Field (CRF)

This **linear-chain CRF** is just **like an HMM,** except that its factors are **not** necessarily probability distributions

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \psi_{\text{em}}(y_k, x_k) \psi_{\text{tr}}(y_k, y_{k-1})$$

$$= \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\text{em}}(y_k, x_k)) \exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\text{tr}}(y_k, y_{k-1}))$$



Quiz

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \psi_{\text{em}}(y_k, x_k) \psi_{\text{tr}}(y_k, y_{k-1})$$

$$= \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\text{em}}(y_k, x_k)) \exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\text{tr}}(y_k, y_{k-1}))$$

Multiple Choice: Which model does the above distribution share the most in common with?

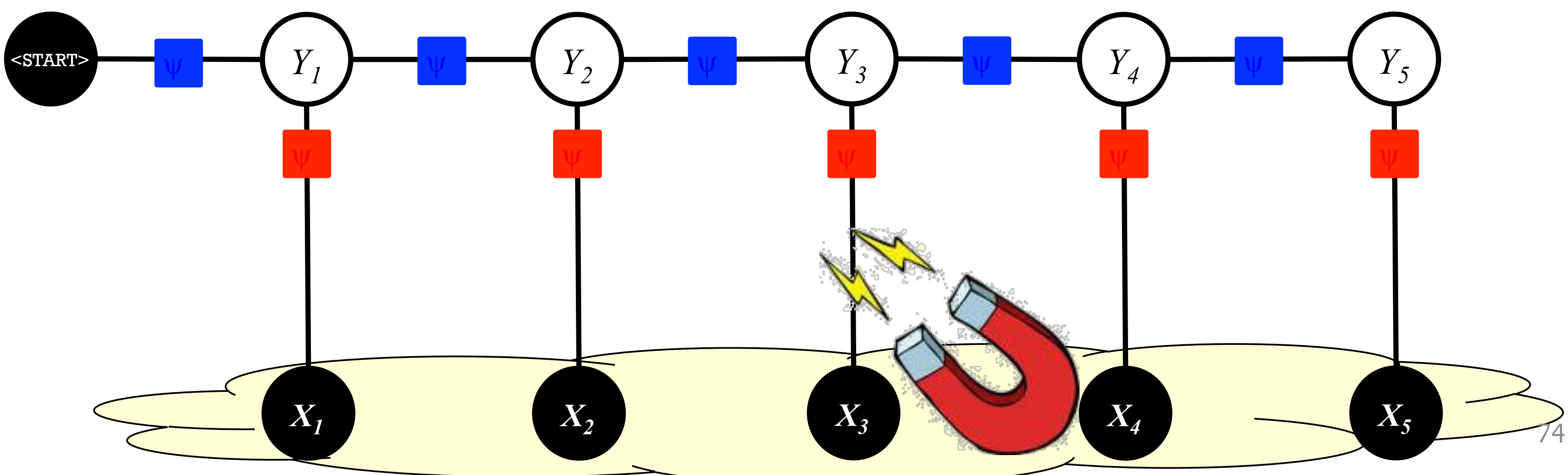
A. Hidden Markov Model
B. Bernoulli Naïve Bayes
C. Gaussian Naïve Bayes
D. Logistic Regression

Conditional Random Field (CRF)

This **linear-chain CRF** is just **like an HMM,** except that its factors are **not** necessarily probability distributions

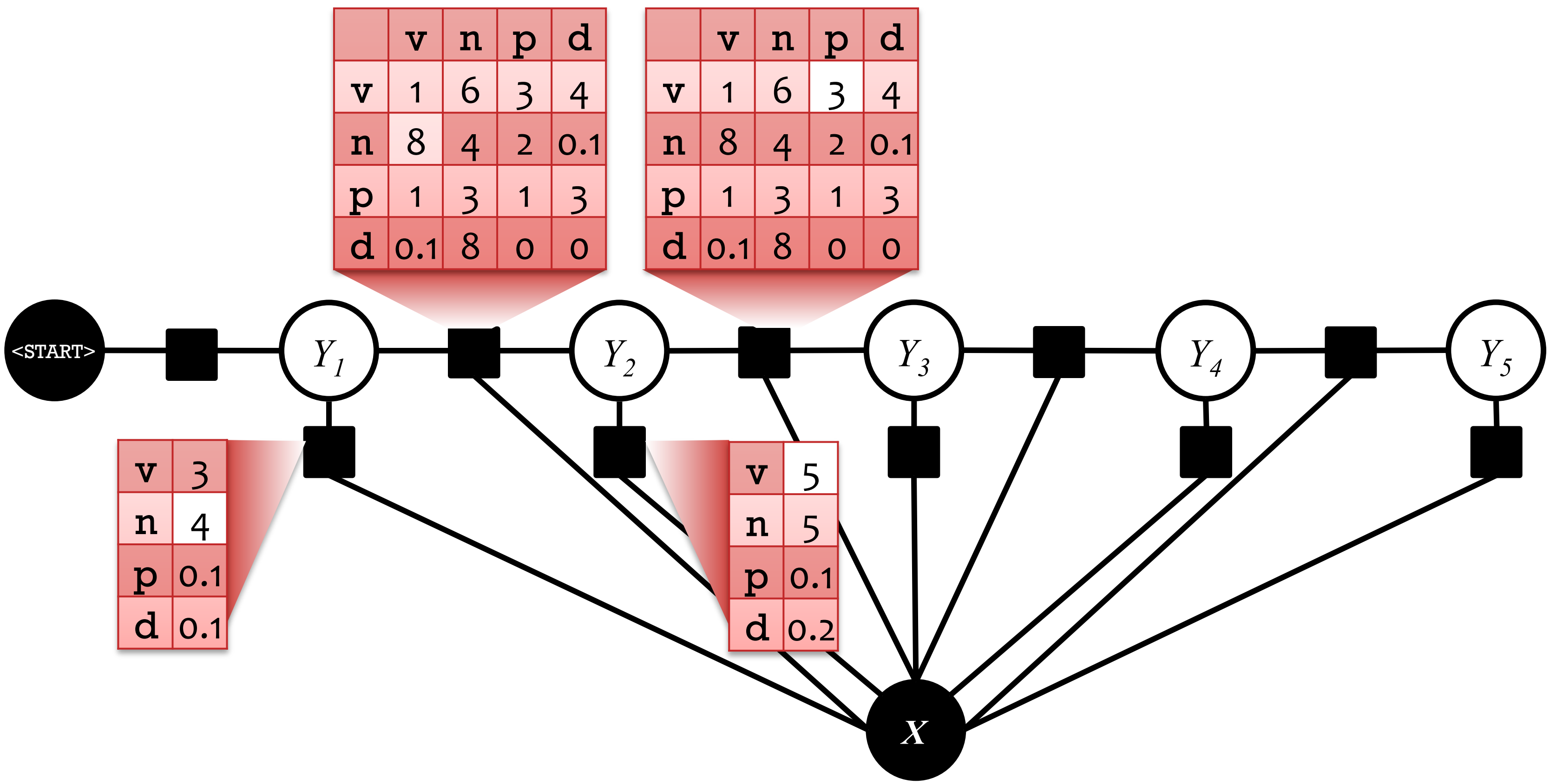
$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \psi_{\text{em}}(y_k, x_k)\psi_{\text{tr}}(y_k, y_{k-1})$$

$$= \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\text{em}}(y_k, x_k))\exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\text{tr}}(y_k, y_{k-1}))$$



Conditional Random Field (CRF)

- That is the **vector** X
- Because it's observed, we can condition on it for free
- Conditioning is how we converted from the MRF to the CRF (i.e. when taking a slice of the emission factors)



	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

v	3
n	4
p	0.1
d	0.1

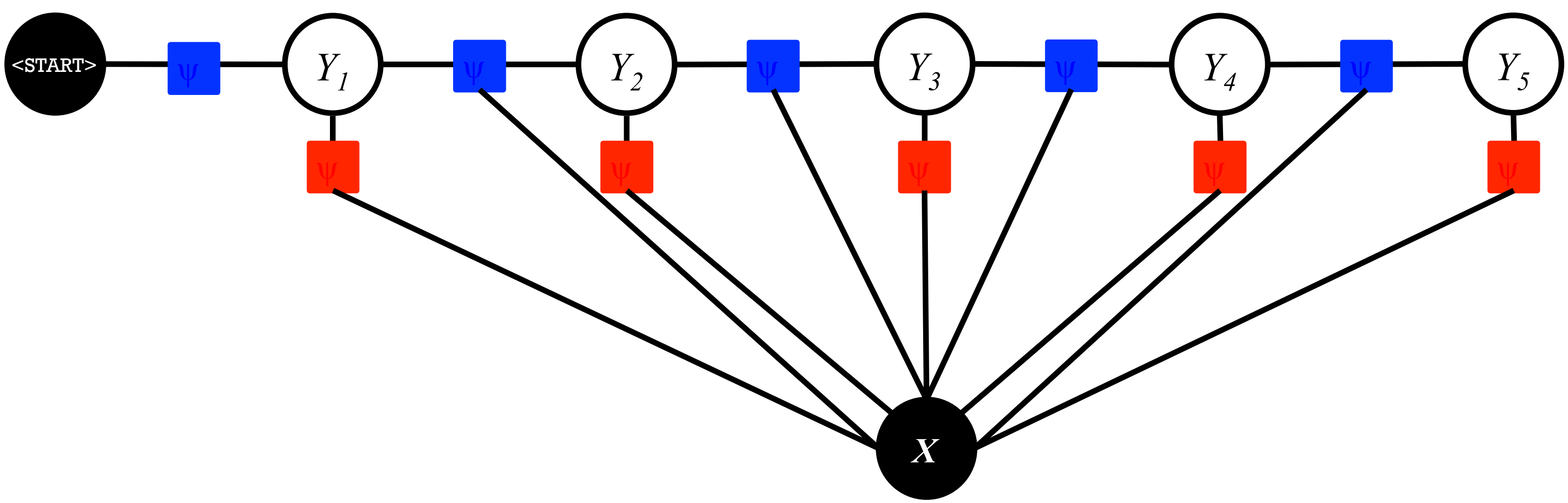
v	5
n	5
p	0.1
d	0.2

Conditional Random Field (CRF)

- This is the **standard** linear-chain CRF definition
- It permits rich, overlapping features of the vector $\boldsymbol{X}$

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \psi_{\text{em}}(y_k, \mathbf{x}) \psi_{\text{tr}}(y_k, y_{k-1}, \mathbf{x})$$

$$= \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\text{em}}(y_k, \mathbf{x})) \exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\text{tr}}(y_k, y_{k-1}, \mathbf{x}))$$

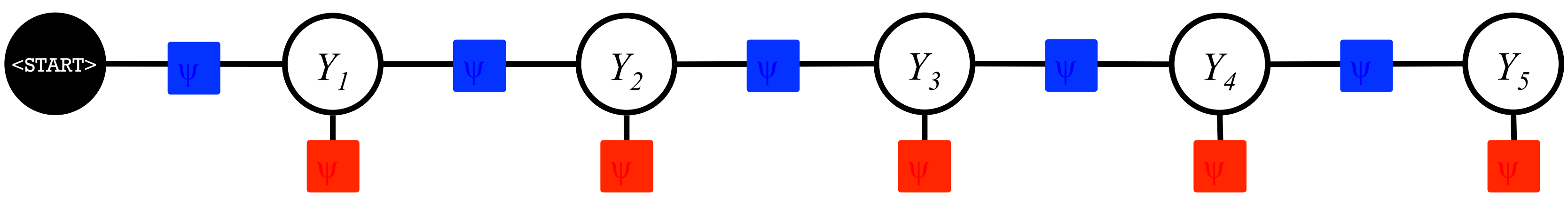


Conditional Random Field (CRF)

- This is the **standard** linear-chain CRF definition
- It permits rich, overlapping features of the vector $\boldsymbol{X}$

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \psi_{\text{em}}(y_k, \mathbf{x}) \psi_{\text{tr}}(y_k, y_{k-1}, \mathbf{x})$$

$$= \frac{1}{Z(\mathbf{x})} \prod_{k=1}^{K} \exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\text{em}}(y_k, \mathbf{x})) \exp(\boldsymbol{\theta} \cdot \mathbf{f}_{\text{tr}}(y_k, y_{k-1}, \mathbf{x}))$$



Visual Notation: Usually we draw a CRF **without** showing the variable corresponding to $\boldsymbol{X}$

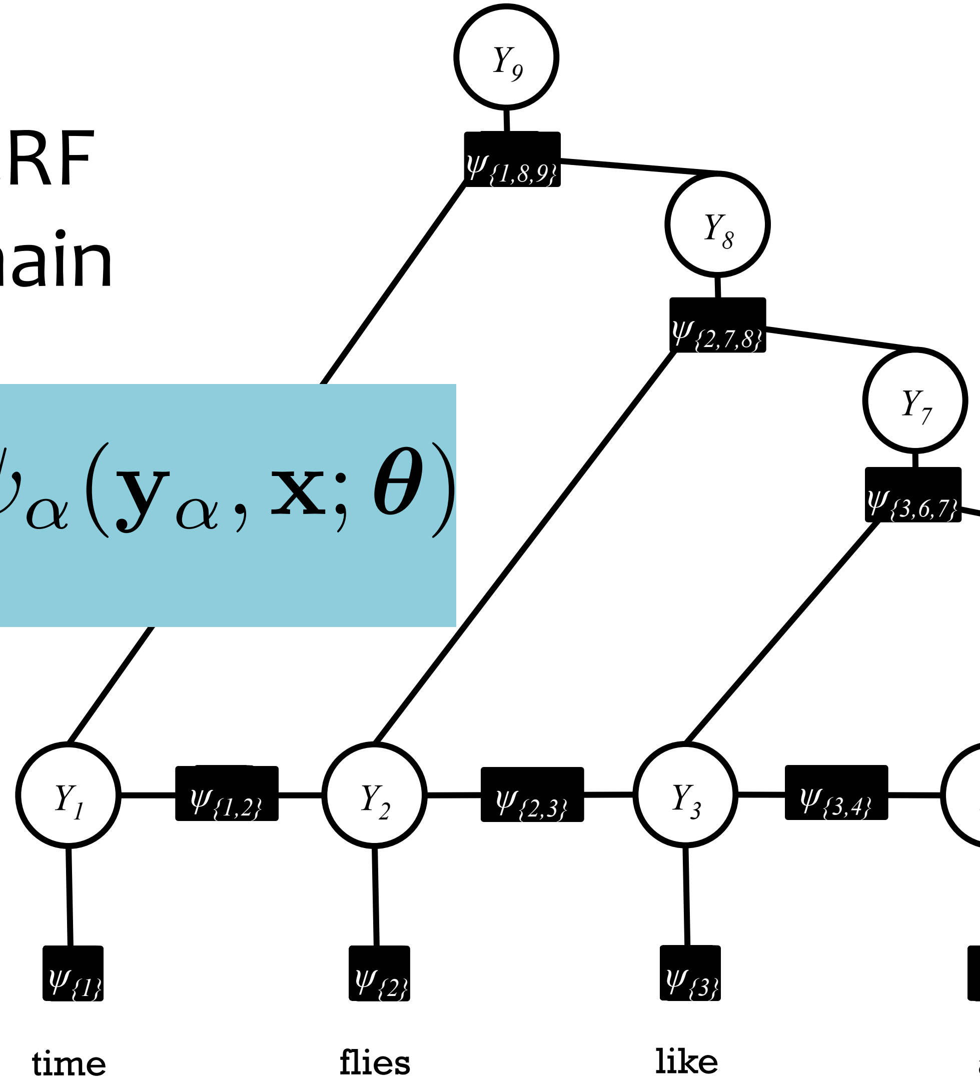
Whiteboard

- Forward-backward algorithm for linear-chain CRF

General CRF

The topology of the graphical model for a CRF doesn't have to be a chain

$$p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{\alpha} \psi_{\alpha}(\mathbf{y}_{\alpha}, \mathbf{x}; \boldsymbol{\theta})$$



Standard CRF Parameterization

$$p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{\alpha} \psi_\alpha(\mathbf{y}_\alpha, \mathbf{x}; \boldsymbol{\theta})$$

Define each potential function in terms of a fixed set of feature functions:

$$\psi_\alpha(\mathbf{y}_\alpha, \mathbf{x}; \boldsymbol{\theta}) = \exp(\boldsymbol{\theta} \cdot \mathbf{f}_\alpha(\mathbf{y}_\alpha, \mathbf{x}))$$

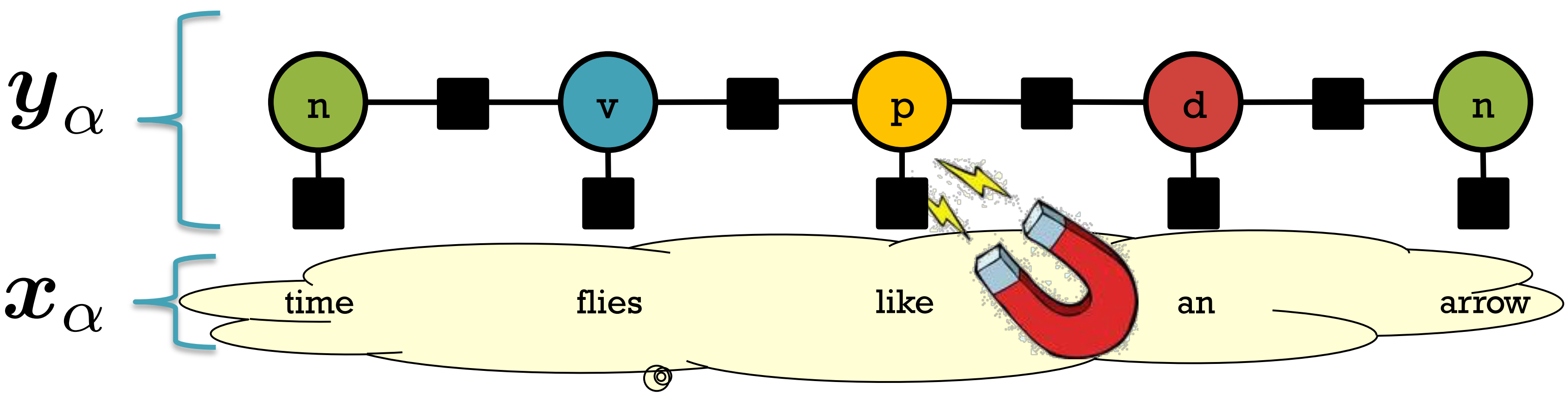
↑ Predicted variables

↑ Observed variables

Standard CRF Parameterization

Define each potential function in terms of a fixed set of feature functions:

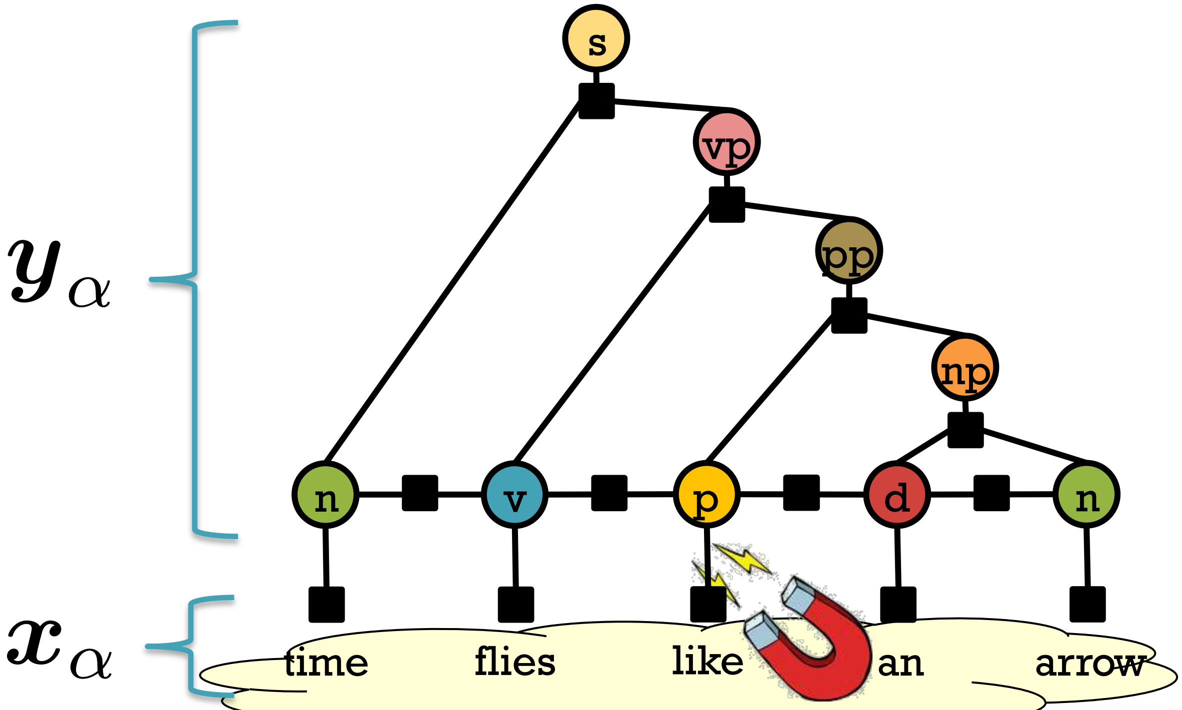
$$\psi_\alpha(\mathbf{y}_\alpha, \mathbf{x}; \boldsymbol{\theta}) = \exp(\boldsymbol{\theta} \cdot \mathbf{f}_\alpha(\mathbf{y}_\alpha, \mathbf{x}))$$



Standard CRF Parameterization

Define each potential function in terms of a fixed set of feature functions:

$$\psi_\alpha(\mathbf{y}_\alpha, \mathbf{x}; \boldsymbol{\theta}) = \exp(\boldsymbol{\theta} \cdot \mathbf{f}_\alpha(\mathbf{y}_\alpha, \mathbf{x}))$$



Exact inference for tree-structured factor graphs

BELIEF PROPAGATION

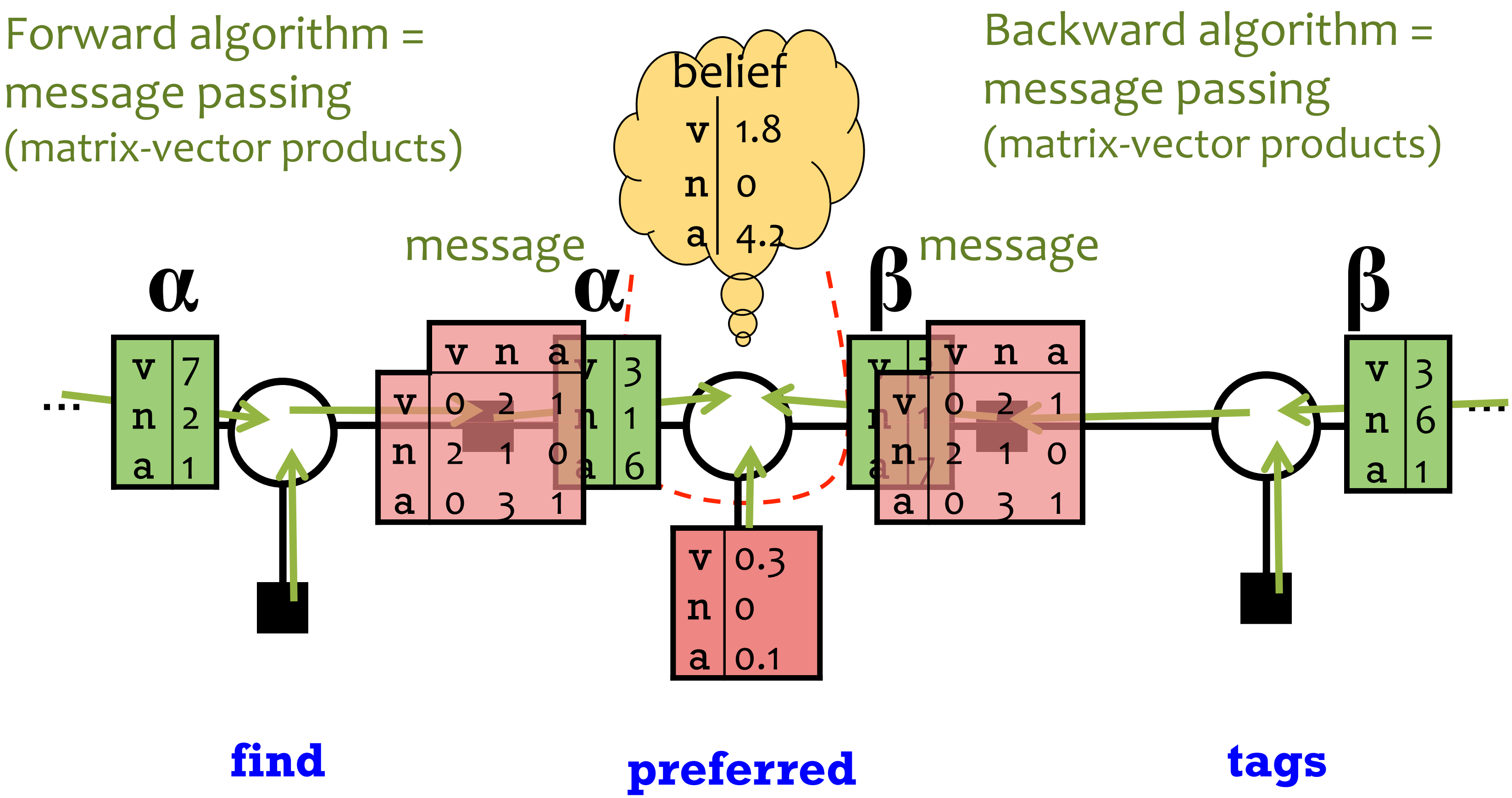
Inference for **HMMs**

- Sum-product BP on **an HMM** is called the **forward-backward algorithm**
- Max-product BP on **an HMM** is called the **Viterbi algorithm**

Inference for CRFs

- Sum-product BP on a CRF is called the **forward-backward algorithm**
- Max-product BP on a CRF is called the **Viterbi algorithm**

CRF Tagging by Belief Propagation



- Forward-backward is a message passing algorithm.
- It's the simplest case of belief propagation.

SUPERVISED LEARNING FOR CRFS

What is Training?

That’s easy:

Training = picking **good** model parameters!

But how do we know if the
model parameters are any “good”?

Machine Learning

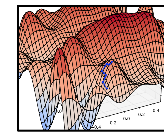
Log-likelihood Training

1. Choose **model**

$$p_\theta(\boldsymbol{y}) = \frac{1}{Z} \prod_\alpha \psi_\alpha(\boldsymbol{y}_\alpha)$$

2. Choose **objective:**
   Assign high probability to the things we observe and low probability to everything else

$$L(\theta) = \sum_{\boldsymbol{y} \in \mathcal{D}} \log p_\theta(\boldsymbol{y})$$



3. Compute derivative **by hand** using the chain rule

$$\frac{dL(\theta)}{d\theta_j} = \sum_{\boldsymbol{y} \in \mathcal{D}} \left(\sum_\alpha \left[f_{\alpha,j}(\boldsymbol{y}_\alpha) - \sum_{\boldsymbol{y}'} p_\theta(\boldsymbol{y}'_\alpha) f_{\alpha,j}(\boldsymbol{y}'_\alpha) \right] \right)$$

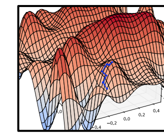
Log-likelihood Training

1. Choose **model**
   Such that derivative in #3 is easy

$$p_\theta(\boldsymbol{y}) = \frac{1}{Z} \prod_\alpha \exp(\theta \cdot \boldsymbol{f}_\alpha(\boldsymbol{y}_\alpha))$$

2. Choose **objective:**
   Assign high probability to the things we observe and low probability to everything else

$$L(\theta) = \sum_{\boldsymbol{y} \in \mathcal{D}} \log p_\theta(\boldsymbol{y})$$



3. Compute derivative **by hand** using the chain rule

$$\frac{dL(\theta)}{d\theta_j} = \sum_{\boldsymbol{y} \in \mathcal{D}} \left(\sum_\alpha \left[f_{\alpha,j}(\boldsymbol{y}_\alpha) - \sum_{\boldsymbol{y}'} p_\theta(\boldsymbol{y}'_\alpha) f_{\alpha,j}(\boldsymbol{y}'_\alpha) \right] \right)$$

4. Compute **the marginals** by exact inference

Note that these are **factor marginals** which are just the (normalized) **factor beliefs** from BP!

Recipe for Gradient-based Learning

1. Write down the objective function
2. Compute the partial derivatives of the objective (i.e. gradient, and maybe Hessian)
3. Feed objective function and derivatives into black box

4. Retrieve optimal parameters from black box

Optimization Algorithms

What is the black box?

- Newton's method
- Hessian-free / Quasi-Newton methods
  – Conjugate gradient
  – L-BFGS
- Stochastic gradient methods
  – Stochastic gradient descent (SGD)
  – Stochastic meta-descent
  – AdaGrad

Stochastic Gradient Descent

- Suppose we have N training examples s.t. $f(x) = \sum_{i=1}^{N} f_i(x)$.
- This implies that $\nabla f(x) = \sum_{i=1}^{N} \nabla f_i(x)$.

SGD Algorithm:

1. Choose a starting point x.
2. While not converged:
   - Choose a step size t.
   - Choose i so that it sweeps through the training set.
   - Update

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} + t \nabla f_i(\vec{x})$$

Whiteboard

- CRF model
- CRF data log-likelihood
- CRF derivatives

Practical Considerations for Gradient-based Methods

- Overfitting
  - L2 regularization
  - L1 regularization
  - Regularization by early stopping
- For SGD: Sparse updates

"Empirical" Comparison of Parameter Estimation Methods

- **Example NLP task:** CRF dependency parsing
- **Suppose:** Training time is dominated by inference
- **Dataset:** One million tokens
- **Inference speed:** 1,000 tokens / sec
- ➔ 0.27 hours per pass through dataset

	# passes through data to converge	**# hours to converge**
GIS	1000+	270
L-BFGS	100+	27
SGD	10	~3

FEATURE ENGINEERING FOR CRFS

Features

General idea:

- Make a list of interesting substructures.
- The feature $f_k(x,y)$ counts tokens of k^{th} substructure in (x,y).

Features for tagging …

N V P D N

Time flies like an arrow

- Count of tag P as the tag for "like"

Weight of this feature is like log of an emission probability in an HMM

Features for tagging …

N V P D N

Time flies like an arrow

- Count of tag P as the tag for "like"
- Count of tag P

Features for tagging …

N V P D N

Time flies like an arrow

0 1 2 3 4 5

- Count of tag P as the tag for "like"
- Count of tag P
- Count of tag P in the middle third of the sentence

Features for tagging …

N V P D N

Time flies like an arrow

- Count of tag P as the tag for "like"
- Count of tag P
- Count of tag P in the middle third of the sentence
- Count of tag bigram V P

Weight of this feature is like log of a transition probability in an HMM

Features for tagging …

N V P D N

Time flies like an arrow

- Count of tag P as the tag for "like"
- Count of tag P
- Count of tag P in the middle third of the sentence
- Count of tag bigram V P
- Count of tag bigram V P followed by "an"

Features for tagging …

N V P D N
Time flies like an arrow

- Count of tag P as the tag for "like"
- Count of tag P
- Count of tag P in the middle third of the sentence
- Count of tag bigram V P
- Count of tag bigram V P followed by "an"
- Count of tag bigram V P where P is the tag for "like"

Features for tagging …

N V P D N

Time flies like an arrow

- Count of tag P as the tag for "like"
- Count of tag P
- Count of tag P in the middle third of the sentence
- Count of tag bigram V P
- Count of tag bigram V P followed by "an"
- Count of tag bigram V P where P is the tag for "like"
- Count of tag bigram V P where both words are lowercase

Features for tagging ...

N V P D N

Time flies like an arrow

- Count of tag trigram N V P?
  - A bigram tagger can only consider within-bigram features: only look at 2 adjacent blue tags (plus arbitrary red context).
  - So here we need a trigram tagger, which is slower.
  - The forward-backward states would remember *two* previous tags.

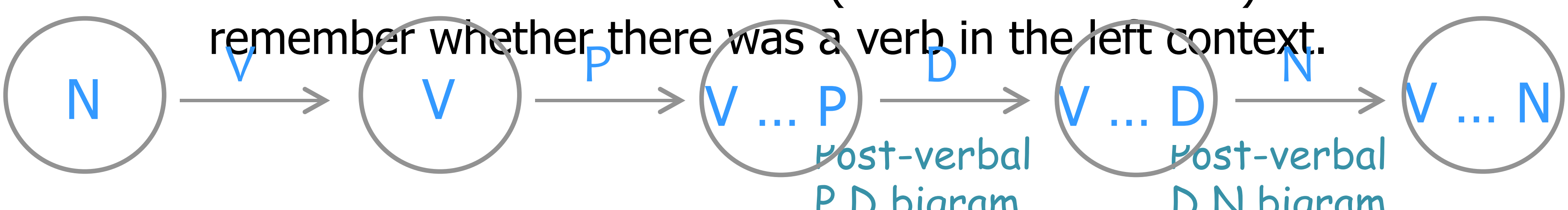
We take this arc once per N V P triple, so its weight is the total weight of the features that fire on that triple.

Features for tagging …

N V P D N

Time flies like an arrow

- Count of tag trigram N V P?
  - A bigram tagger can only consider within-bigram features: only look at 2 adjacent blue tags (plus arbitrary red context).
  - So here we need a trigram tagger, which is slower.
- Count of "post-verbal" nouns? ("discontinuous bigram" V N)
  - An n-gram tagger can only look at a narrow window.
  - Here we need a *fancier* model (finite state machine) whose states remember whether there was a verb in the left context.



How might you come up with the features that you will use to score (x,y)?

1. Think of some attributes ("basic features") that you can compute at each position in (x,y).

For position i in a tagging, these might include:

- Full name of tag i
- First letter of tag i (will be "N" for both "NN" and "NNS")
- Full name of tag i-1 (possibly BOS); similarly tag i+1 (possibly EOS)
- Full name of word i
- Last 2 chars of word i (will be "ed" for most past-tense verbs)
- First 4 chars of word i (why would this help?)
- "Shape" of word i (lowercase/capitalized/all caps/numeric/…)
- Whether word i is part of a known city name listed in a "gazetteer"
- Whether word i appears in thesaurus entry e (one attribute per e)
- Whether i is in the middle third of the sentence

How might you come up with the features that you will use to score (x,y)?

1. Think of some attributes ("basic features") that you can compute at each position in (x,y).
2. Now conjoin them into various "feature templates."

E.g., template 7 might be (tag(i-1), tag(i), suffix2(i+1)).

At each position of (x,y), exactly one of the many template7 features will fire:

N V P D N

Time flies like an arrow

At i=1, we see an instance of "template7=(BOS,N,-es)"
so we add one copy of that feature's weight to score(x,y)

How might you come up with the features that you will use to score (x,y)?

1. Think of some attributes ("basic features") that you can compute at each position in (x,y).
2. Now conjoin them into various "feature templates."

E.g., template 7 might be (tag(i-1), tag(i), suffix2(i+1)).

At each position of (x,y), exactly one of the many template7 features will fire:

N V P D N

Time flies like an arrow

At i=2, we see an instance of "template7=(N,V,-ke)"
so we add one copy of that feature's weight to score(x,y)

How might you come up with the features that you will use to score (x,y)?

1. Think of some attributes ("basic features") that you can compute at each position in (x,y).
2. Now conjoin them into various "feature templates."

E.g., template 7 might be (tag(i-1), tag(i), suffix2(i+1)).

At each position of (x,y), exactly one of the many template7 features will fire:

N V P D N

Time flies like an arrow

At i=3, we see an instance of "template7=(N,V,-an)"
so we add one copy of that feature's weight to score(x,y)

How might you come up with the features that you will use to score (x,y)?

1. Think of some attributes ("basic features") that you can compute at each position in (x,y).
2. Now conjoin them into various "feature templates."

E.g., template 7 might be (tag(i-1), tag(i), suffix2(i+1)).

At each position of (x,y), exactly one of the many template7 features will fire:

N V P D N

Time flies like an arrow

At i=4, we see an instance of "template7=(P,D,-ow)"
so we add one copy of that feature's weight to score(x,y)

How might you come up with the features that you will use to score (x,y)?

1. Think of some attributes ("basic features") that you can compute at each position in (x,y).
2. Now conjoin them into various "feature templates."

E.g., template 7 might be (tag(i-1), tag(i), suffix2(i+1)).

At each position of (x,y), exactly one of the many template7 features will fire:

N V P D N

Time flies like an arrow

At i=5, we see an instance of "template7=(D,N,-)"
so we add one copy of that feature's weight to score(x,y)

How might you come up with the features that you will use to score (x,y)?

1. Think of some attributes ("basic features") that you can compute at each position in (x,y).
2. Now conjoin them into various "feature templates."

E.g., template 7 might be (tag(i-1), tag(i), suffix2(i+1)).
This template gives rise to *many* features, e.g.:

score(x,y) = …
+ θ["template7=(P,D,-ow)"] * count("template7=(P,D,-ow)")
+ θ["template7=(D,D,-xx)"] * count("template7=(D,D,-xx)")
+ …

With a handful of feature templates and a large vocabulary, you can easily end up with millions of features.

How might you come up with the features that you will use to score (x,y)?

1. Think of some attributes ("basic features") that you can compute at each position in (x,y).
2. Now conjoin them into various "feature templates."

E.g., template 7 might be (tag(i-1), tag(i), suffix2(i+1)).

Note: Every template should mention at least some blue.

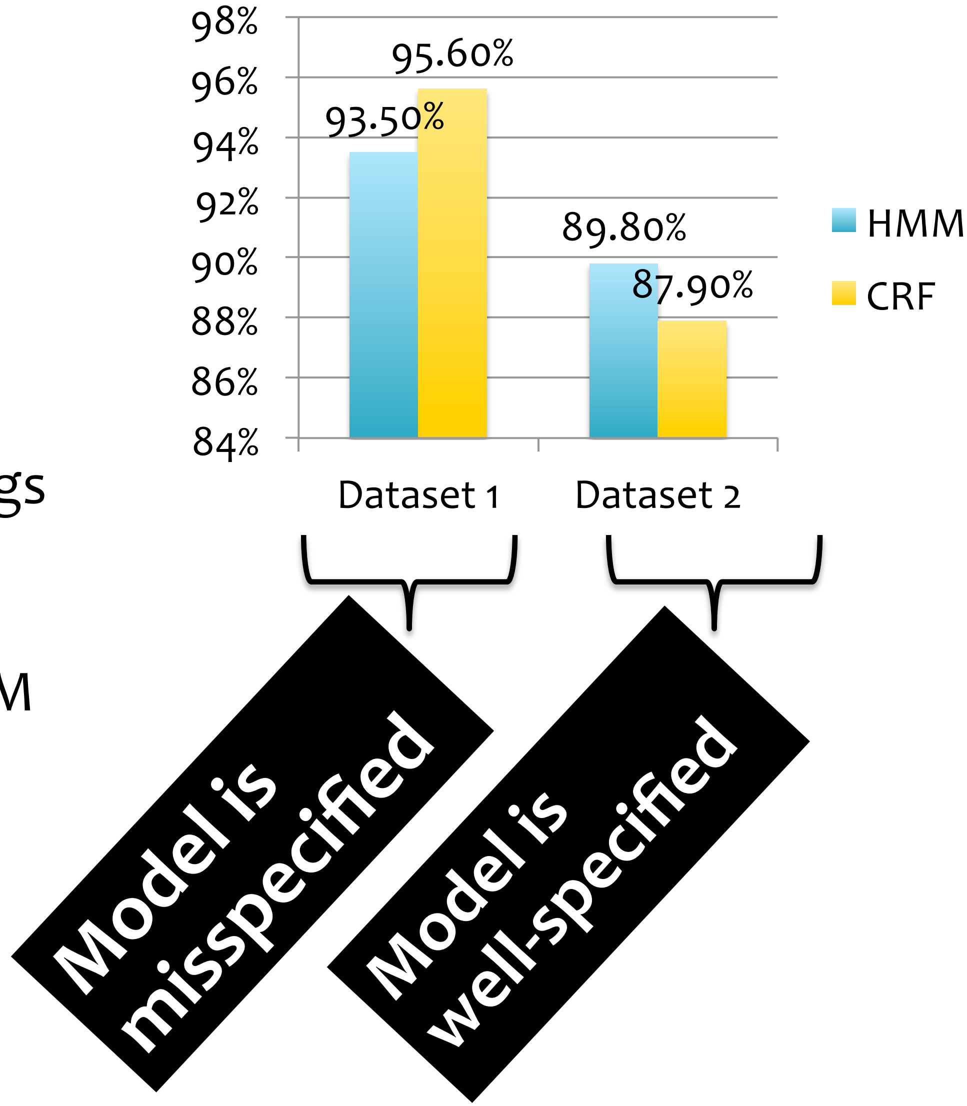
- Given an input x, a feature that only looks at red will contribute the same weight to score(x,y_1) and score(x,y_2).
- So it can't help you choose between outputs y_1, y_2.

HMMS VS CRFS

Generative vs. Discriminative

Liang & Jordan (ICML 2008) compares **HMM** and **CRF** with **identical features**

- Dataset 1: (Real)
  - WSJ Penn Treebank (38K train, 5.5K test)
  - 45 part-of-speech tags
- Dataset 2: (Artificial)
  - Synthetic data generated from HMM learned on Dataset 1 (1K train, 1K test)
- Evaluation Metric: Accuracy



CRFs: some empirical results

- Parts of Speech tagging

model	*error*	*oov error*
HMM	5.69%	45.99%
MEMM	6.37%	54.61%
CRF	5.55%	48.05%
$MEMM^+$	4.81%	26.99%
CRF^+	4.27%	23.76%

$^+$Using spelling features

  – Using same set of features: HMM >=< CRF > MEMM
  – Using additional overlapping features: CRF^+ > $MEMM^+$ >> HMM

MBR DECODING

Minimum Bayes Risk Decoding

- Suppose we given a loss function $l(\mathbf{y}', \mathbf{y})$ and are asked for a single tagging
- How should we choose just one from our probability distribution $p(\mathbf{y}|\mathbf{x})$?
- A minimum Bayes risk (MBR) decoder $h(\mathbf{x})$ returns the variable assignment with minimum **expected** loss under the model's distribution

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \underset{\hat{\boldsymbol{y}}}{\text{argmin}} \; \mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})}[\ell(\hat{\boldsymbol{y}}, \boldsymbol{y})]$$

$$= \underset{\hat{\boldsymbol{y}}}{\text{argmin}} \; \sum_{\boldsymbol{y}} p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) \ell(\hat{\boldsymbol{y}}, \boldsymbol{y})$$

Minimum Bayes Risk Decoding

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \underset{\hat{\boldsymbol{y}}}{\text{argmin}} \; \mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\cdot | \boldsymbol{x})}[\ell(\hat{\boldsymbol{y}}, \boldsymbol{y})]$$

Consider some example loss functions:

The ***0-1* loss function** returns *1* only if the two assignments are identical and *0* otherwise:

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}) = 1 - \mathbb{I}(\hat{\boldsymbol{y}}, \boldsymbol{y})$$

The MBR decoder is:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \underset{\hat{\boldsymbol{y}}}{\text{argmin}} \; \sum_{\boldsymbol{y}} p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})(1 - \mathbb{I}(\hat{\boldsymbol{y}}, \boldsymbol{y}))$$

$$= \underset{\hat{\boldsymbol{y}}}{\text{argmax}} \; p_{\boldsymbol{\theta}}(\hat{\boldsymbol{y}} \mid \boldsymbol{x})$$

which is exactly the MAP inference problem!

Minimum Bayes Risk Decoding

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \underset{\hat{\boldsymbol{y}}}{\text{argmin}} \; \mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\cdot | \boldsymbol{x})}[\ell(\hat{\boldsymbol{y}}, \boldsymbol{y})]$$

Consider some example loss functions:

The **Hamming loss** corresponds to accuracy and returns the number of incorrect variable assignments:

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \sum_{i=1}^{V} (1 - \mathbb{I}(\hat{y}_i, y_i))$$

The MBR decoder is:

$$\hat{y}_i = h_{\boldsymbol{\theta}}(\boldsymbol{x})_i = \underset{\hat{y}_i}{\text{argmax}} \; p_{\boldsymbol{\theta}}(\hat{y}_i \mid \boldsymbol{x})$$

This decomposes across variables and requires the variable marginals.

SUMMARY

Summary: Learning and Inference

For discrete variables:

	Learning	Marginal Inference	MAP Inference
HMM	MLE by counting	Forward-backward	Viterbi
Linear-chain CRF	Gradient based – doesn't decompose because of $Z(\mathbf{x})$ and requires marginal inference	Forward-backward	Viterbi

Summary: Models

	Classification	Structured Prediction
Generative	Naïve Bayes	HMM
Discriminative	Logistic Regression	CRF