

Discrete Sequential Models + General CRF

Kayhan Batmanghelich

Slides Credit:

Matt Gormley (2016)

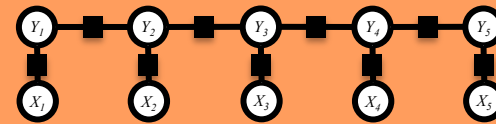
1. Data

$$\mathcal{D} = \{x^{(n)}\}_{n=1}^N$$

Sample 1:	n time	v flies	p like	d an	n arrow
Sample 2:	n time	n flies	v like	d an	n arrow
Sample 3:	n flies	v fly	p with	n their	n wing
Sample 4:	p with	n time	n you	v will	v see

2. Model

$$p(x | \theta) = \frac{1}{Z(\theta)} \prod_{C \in \mathcal{C}} \psi_C(x_C)$$



3. Objective

$$\ell(\theta; \mathcal{D}) = \sum_{n=1}^N \log p(x^{(n)} | \theta)$$

5. Inference

1. Marginal Inference

$$p(x_C) = \sum_{x': x'_C = x_C} p(x' | \theta)$$

2. Partition Function

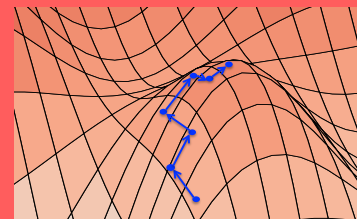
$$Z(\theta) = \sum_x \prod_{C \in \mathcal{C}} \psi_C(x_C)$$

3. MAP Inference

$$\hat{x} = \operatorname{argmax}_x p(x | \theta)$$

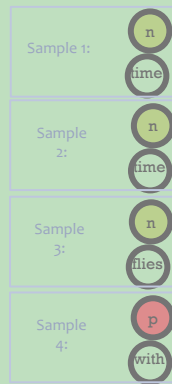
4. Learning

$$\theta^* = \operatorname{argmax}_{\theta} \ell(\theta; \mathcal{D})$$



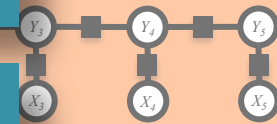
1. Data

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



2. Model

$$p(x) = \prod_{C \in \mathcal{C}} \psi_C(x_C)$$



$$\log p(x^{(n)} | \theta)$$

Today's Lecture...

...is really about Conditional Random Fields (CRFs), but in the guise of two case studies:

1. Part-of-speech (POS) tagging
2. Image segmentation

5.

1. Marginal Inference

$$x' : x'_C = x_C$$

2. Partition Function

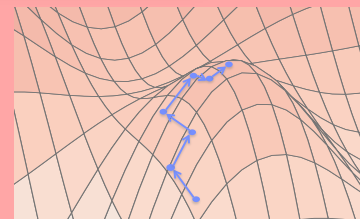
$$Z(\theta) = \sum_x \prod_{C \in \mathcal{C}} \psi_C(x_C)$$

3. MAP Inference

$$\hat{x} = \operatorname{argmax}_x p(x | \theta)$$

Learning

$$\max_{\theta} \ell(\theta; \mathcal{D})$$



Outline

















































1. Case Study: Supervised Part-of-speech tagging
(NLP)
 - Hidden Markov Model (HMM)
 - Maximum-Entropy Markov Model (MEMM)
 - Linear-chain CRF
 - *Digression*: Minimum Bayes Risk (MBR) Decoding
 - *Digression*: Generative vs. Discriminative
2. Case Study: Image Segmentation
(Computer Vision)
 - General CRF (e.g. grid)
 - Hidden-state CRF (HCRF)

HMMs, MEMMs, Linear-chain CRFs

1. CASE STUDY: SUPERVISED PART-OF-SPEECH TAGGING (NLP)

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

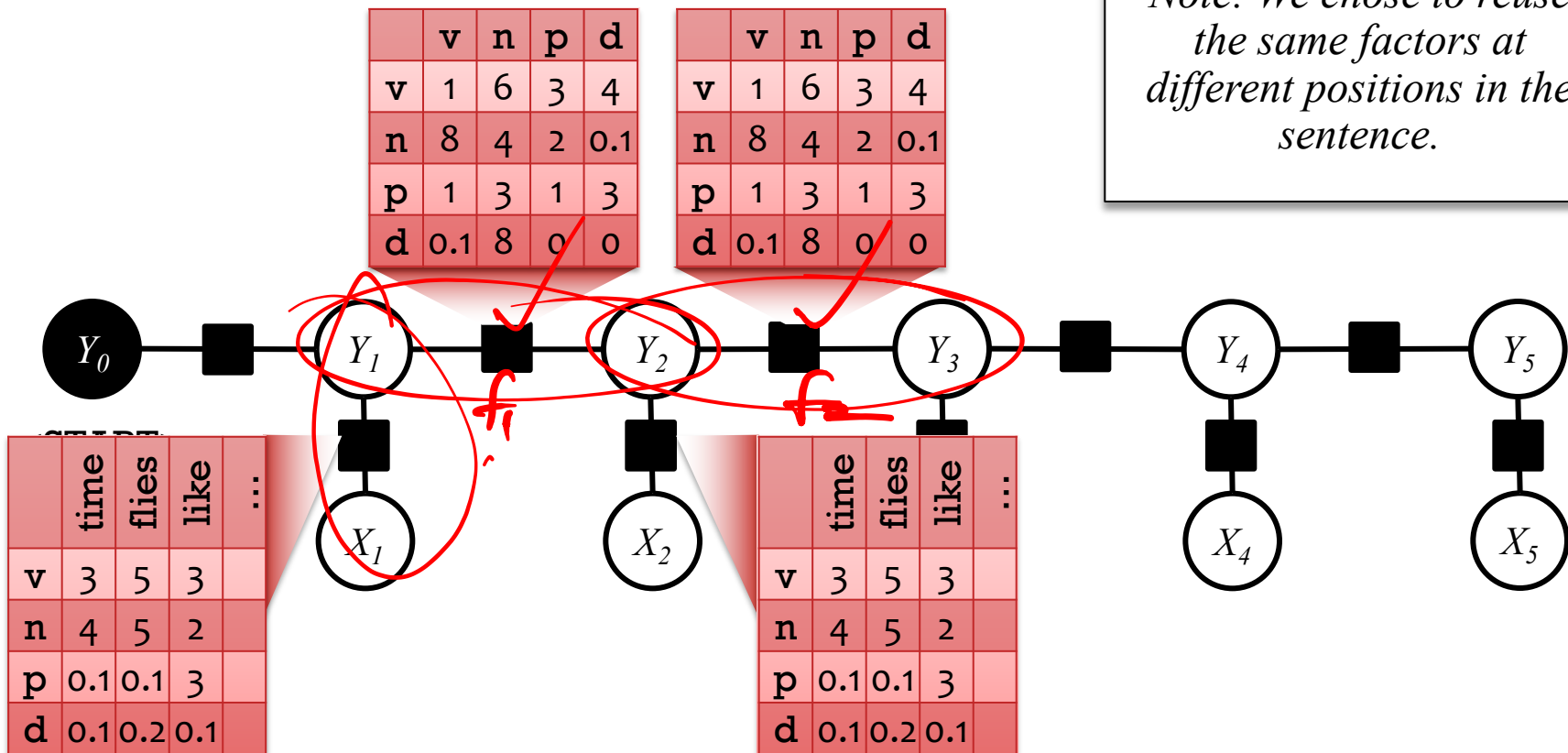
Data: $\mathcal{D} = \{\mathbf{x}^{(n)}, \mathbf{y}^{(n)}\}_{n=1}^N$

Sample 1:							$y^{(1)}$
							$x^{(1)}$
Sample 2:							$y^{(2)}$
							$x^{(2)}$
Sample 3:							$y^{(3)}$
							$x^{(3)}$
Sample 4:							$y^{(4)}$
							$x^{(4)}$

Factors have local opinions (≥ 0)

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

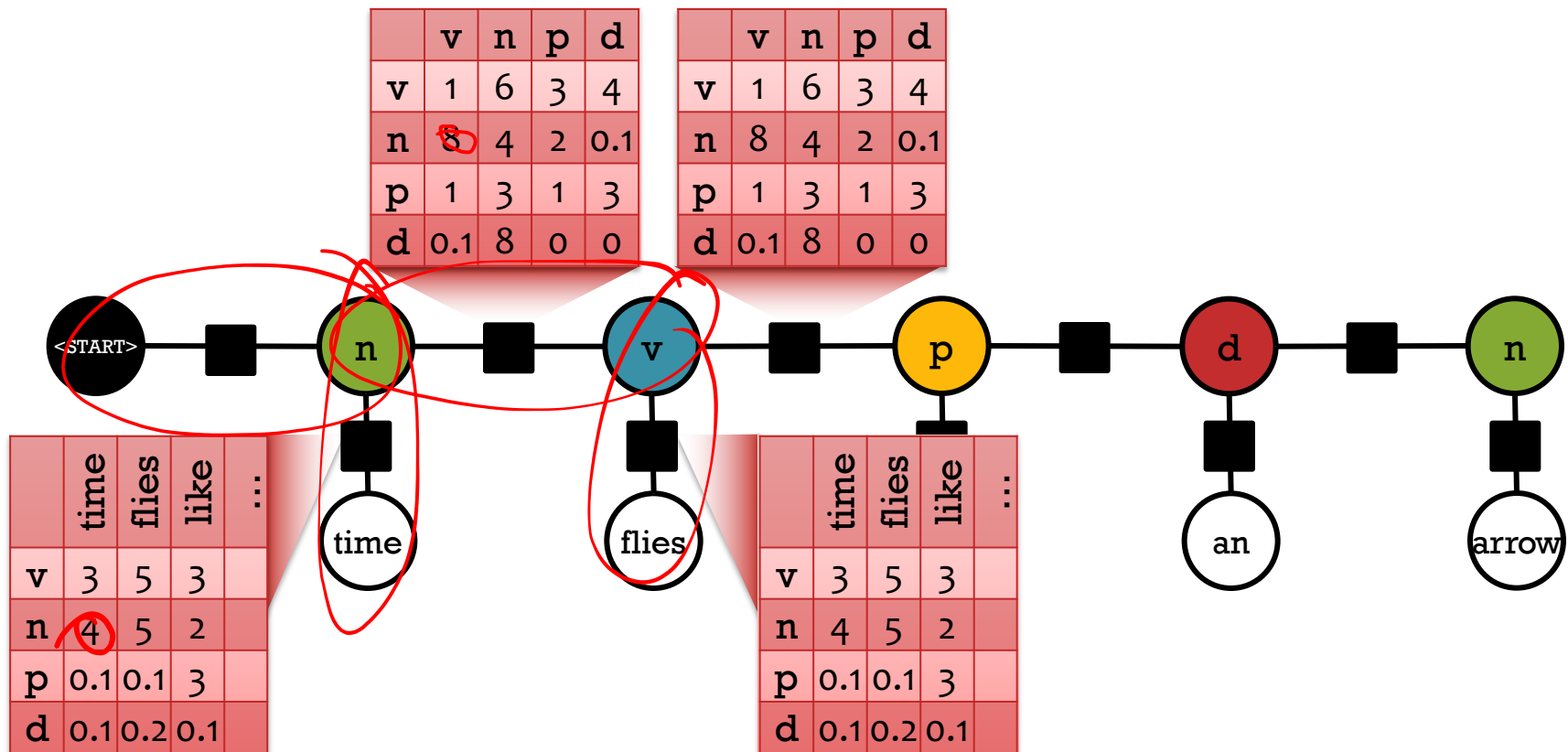
Note: We chose to reuse the same factors at different positions in the sentence.



Factors have local opinions (≥ 0)

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

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



Global probability = product of local opinions

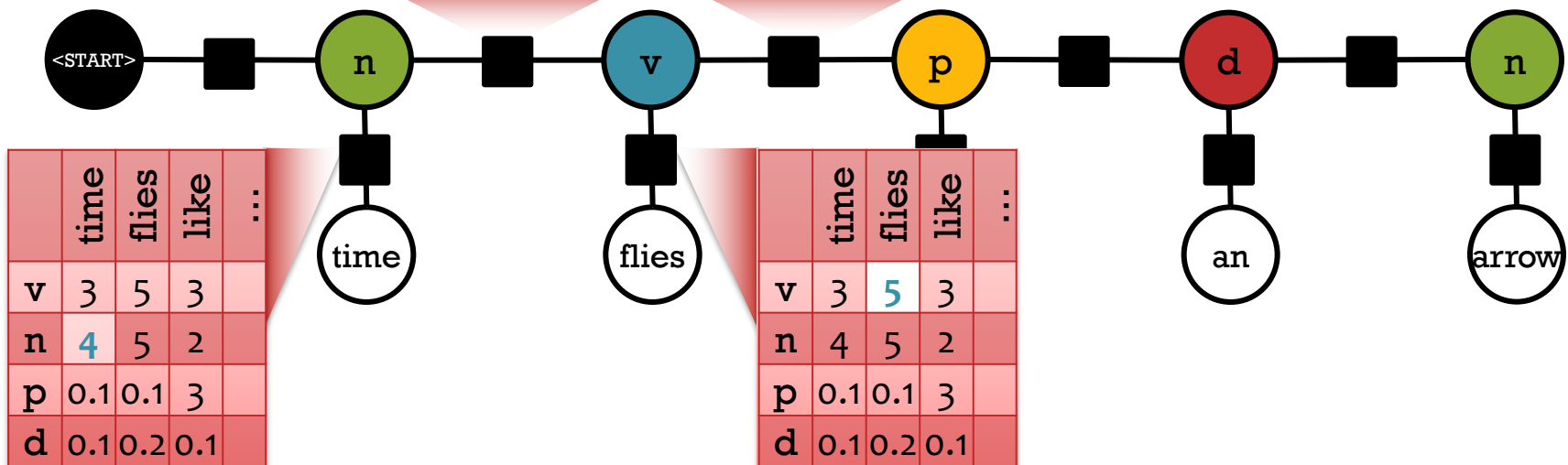
Each black box looks at some of the tags Y_i and words X_i

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

	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

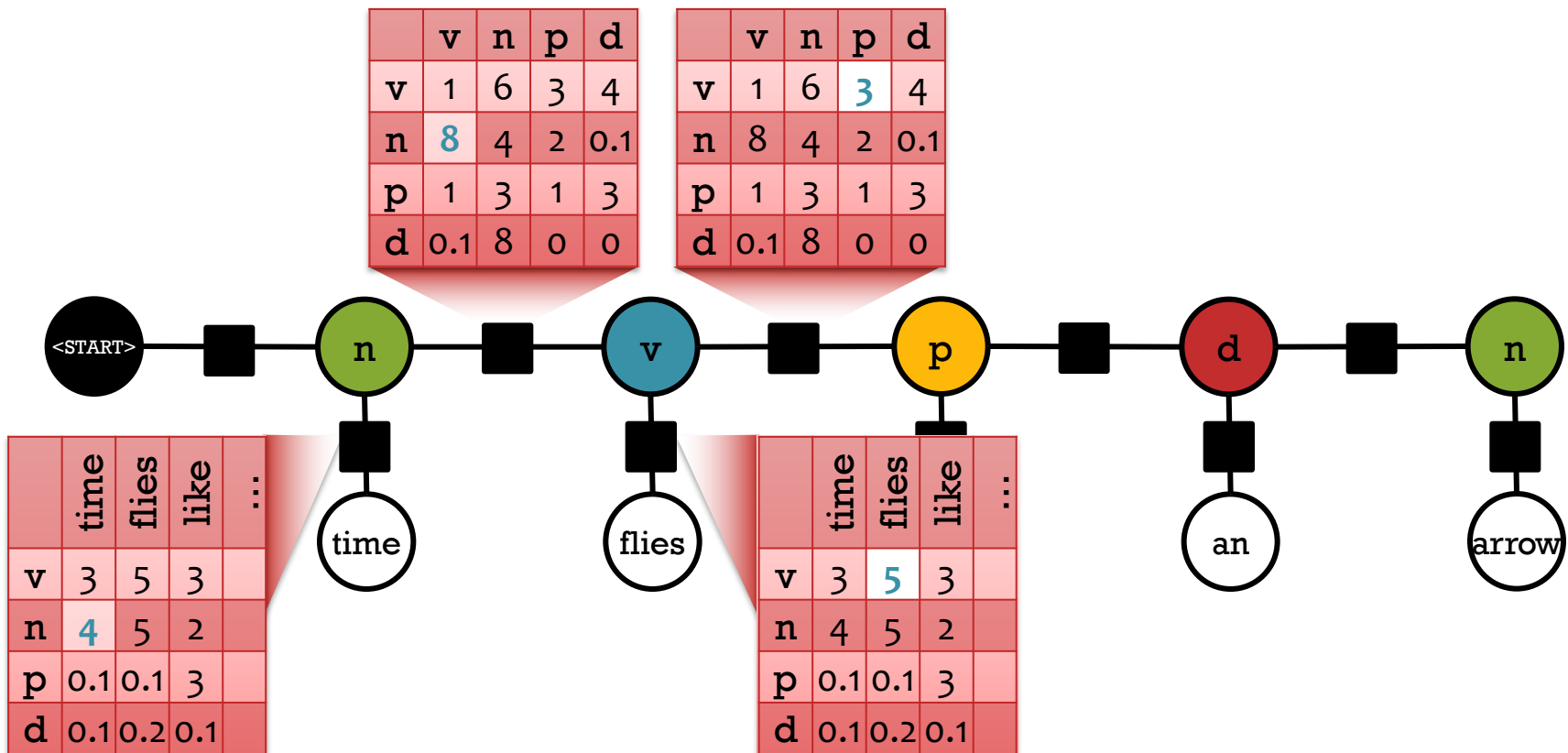
*Uh-oh! The probabilities of the various assignments sum up to $Z > 1$.
So divide them all by Z .*



Markov Random Field (MRF)

Joint distribution over tags Y_i and words X_i
 The individual factors aren't necessarily probabilities.

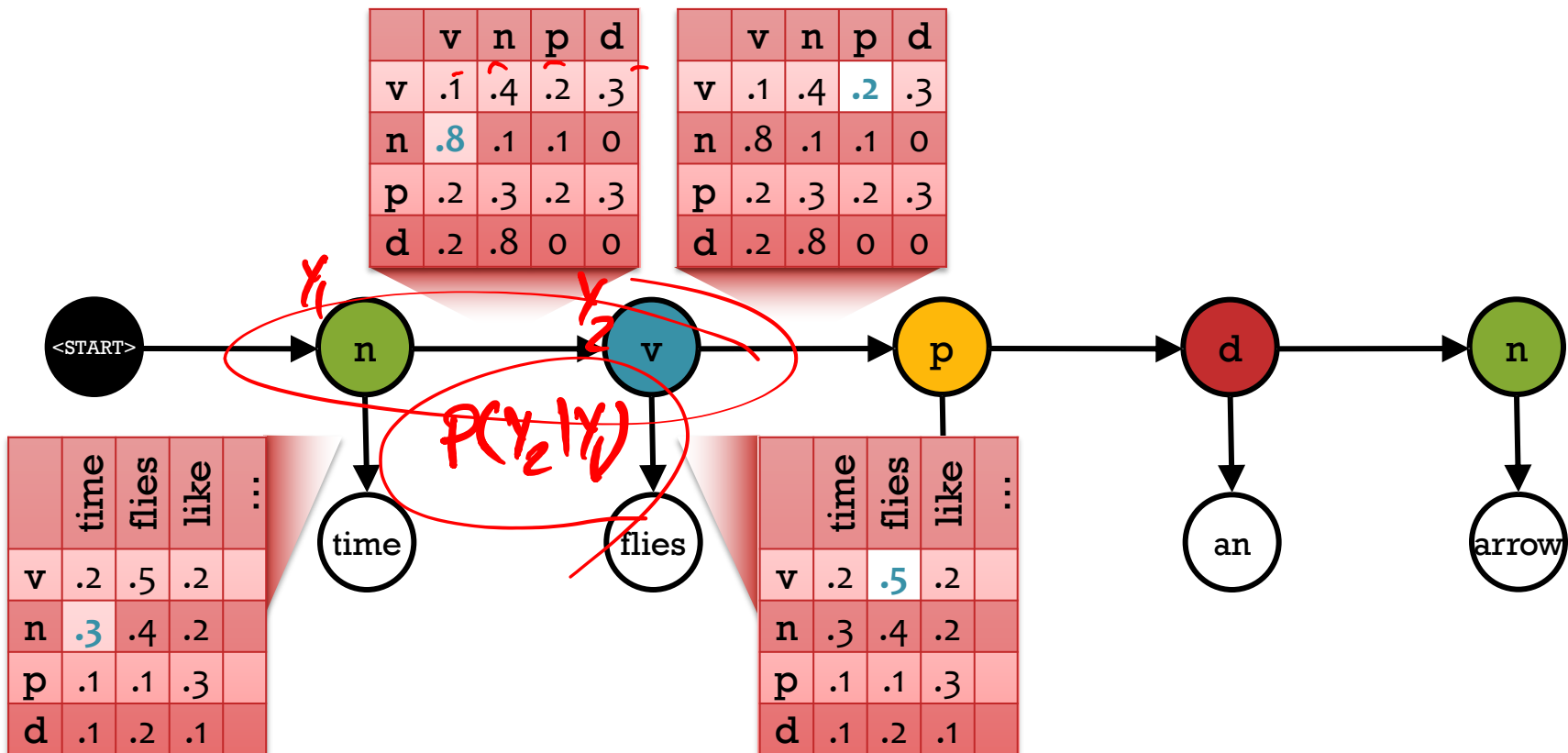
$$p(n, v, p, d, n, \text{time, flies, like, an, arrow}) = \frac{1}{Z} (4 * 8 * 5 * 3 * \dots)$$



Bayesian Networks $\sum P(Y_2|Y_1) = 1$

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

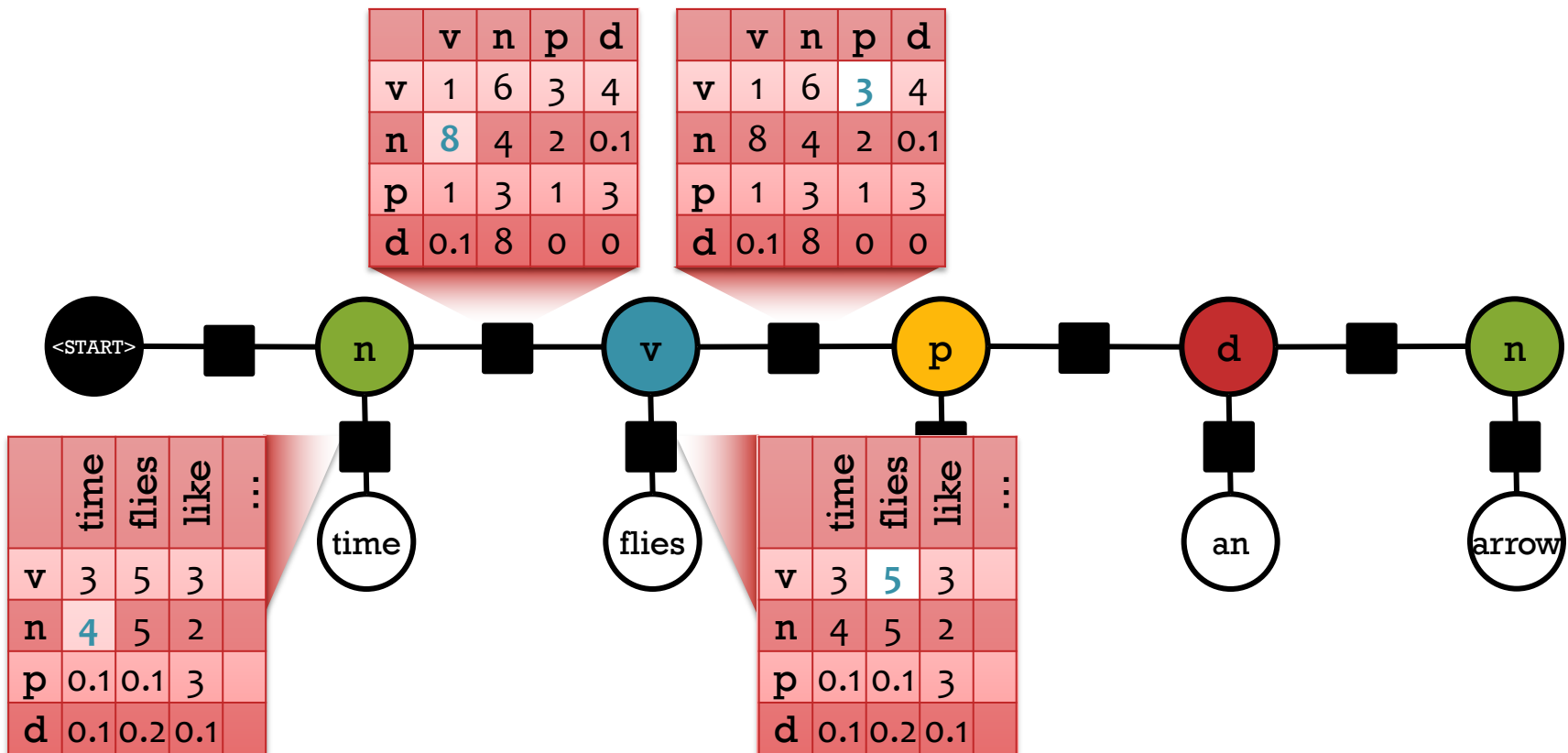
$$p(n, v, p, d, n, \text{time}, \text{flies}, \text{like}, \text{an}, \text{arrow}) = \frac{1}{Z} (.3 * .8 * .2 * .5 * \dots)$$



Markov Random Field (MRF)

Joint distribution over tags Y_i and words X_i

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



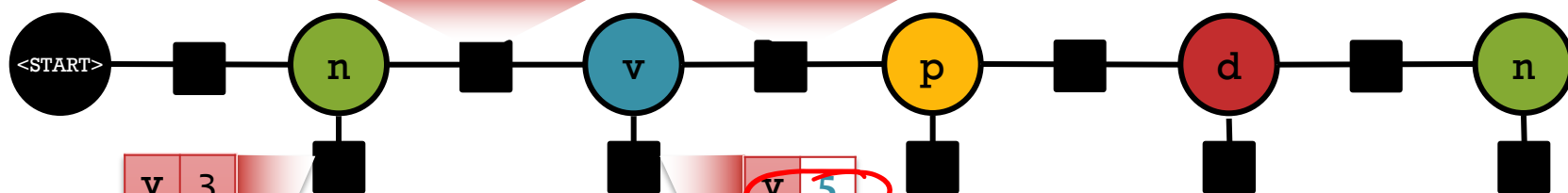
~~$P(y|x)$~~ Conditional Random Field (CRF)

Conditional distribution over tags Y_i given words x_i .
The factors and Z are now specific to the sentence x .

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

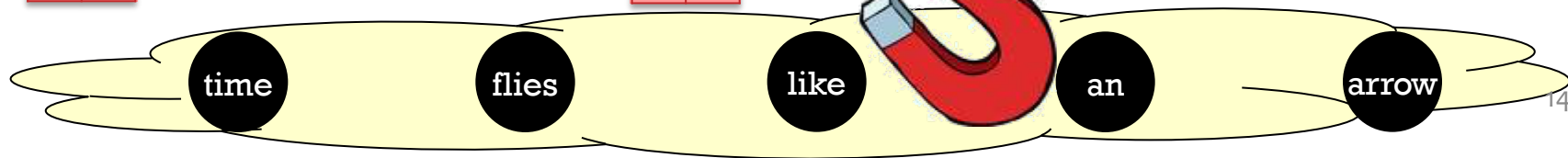
	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	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)

$p(y|x)$ $\hat{y} = x\beta + \varepsilon$ $y \leftarrow f(x)$

Conditional distribution over tags Y_i given words x_i .
The factors and Z are now specific to the sentence x .

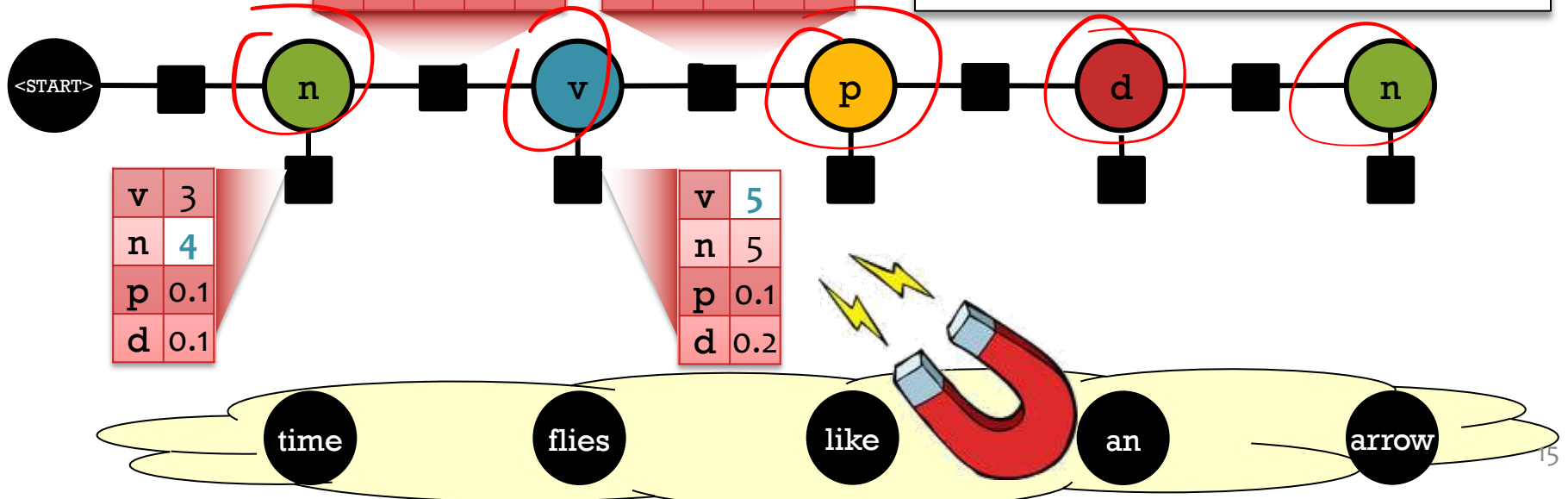
$$p(n, v, p, d, n \mid \text{time, flies, like, an, arrow}) = \frac{1}{Z} (4 * 8 * 5 * 3 * \dots)$$

	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

We say the variables X_i have been “clamped” to their values x_i .

This is equivalent to multiplying in an “evidence potential” which is a point mass with all its weight on $X_i = x_i$



Forward-Backward Algorithm

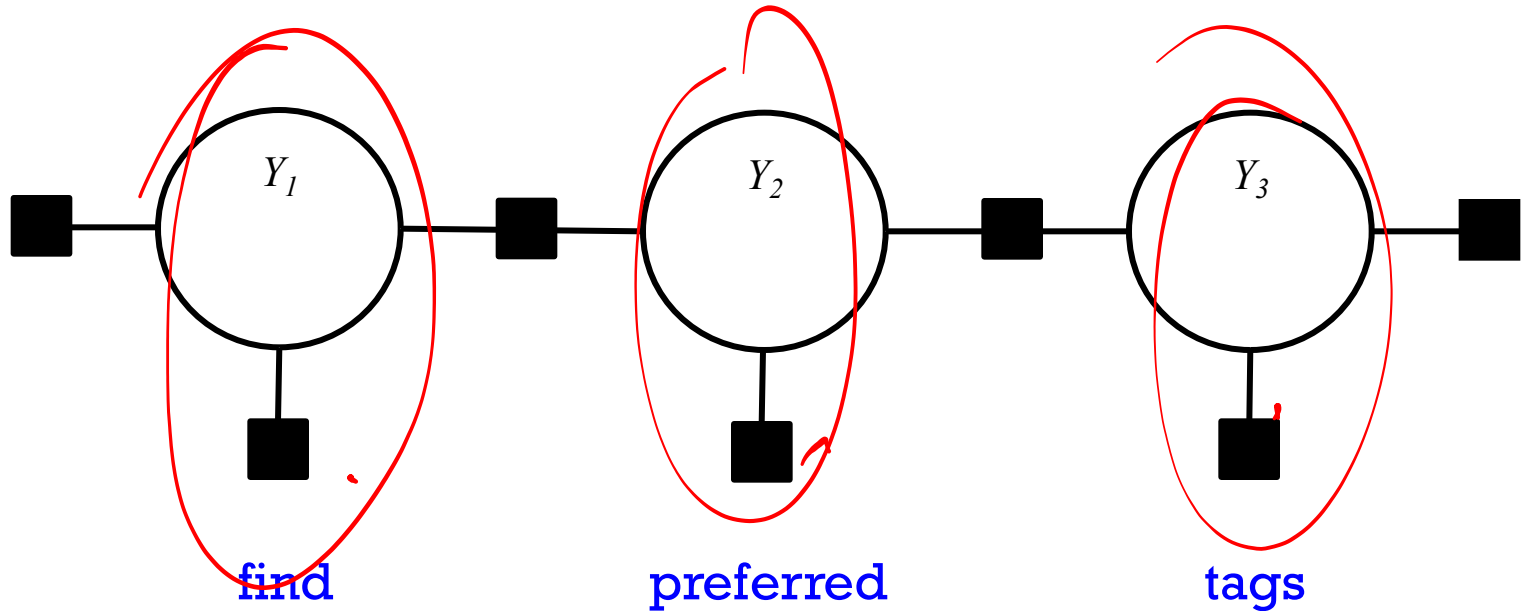
- Sum-product BP on an HMM is called the **forward-backward algorithm**
- Max-product BP on an HMM is called the **Viterbi algorithm**

Learning and Inference Summary

For discrete variables:

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

CRF Tagging Model



Could be verb or noun

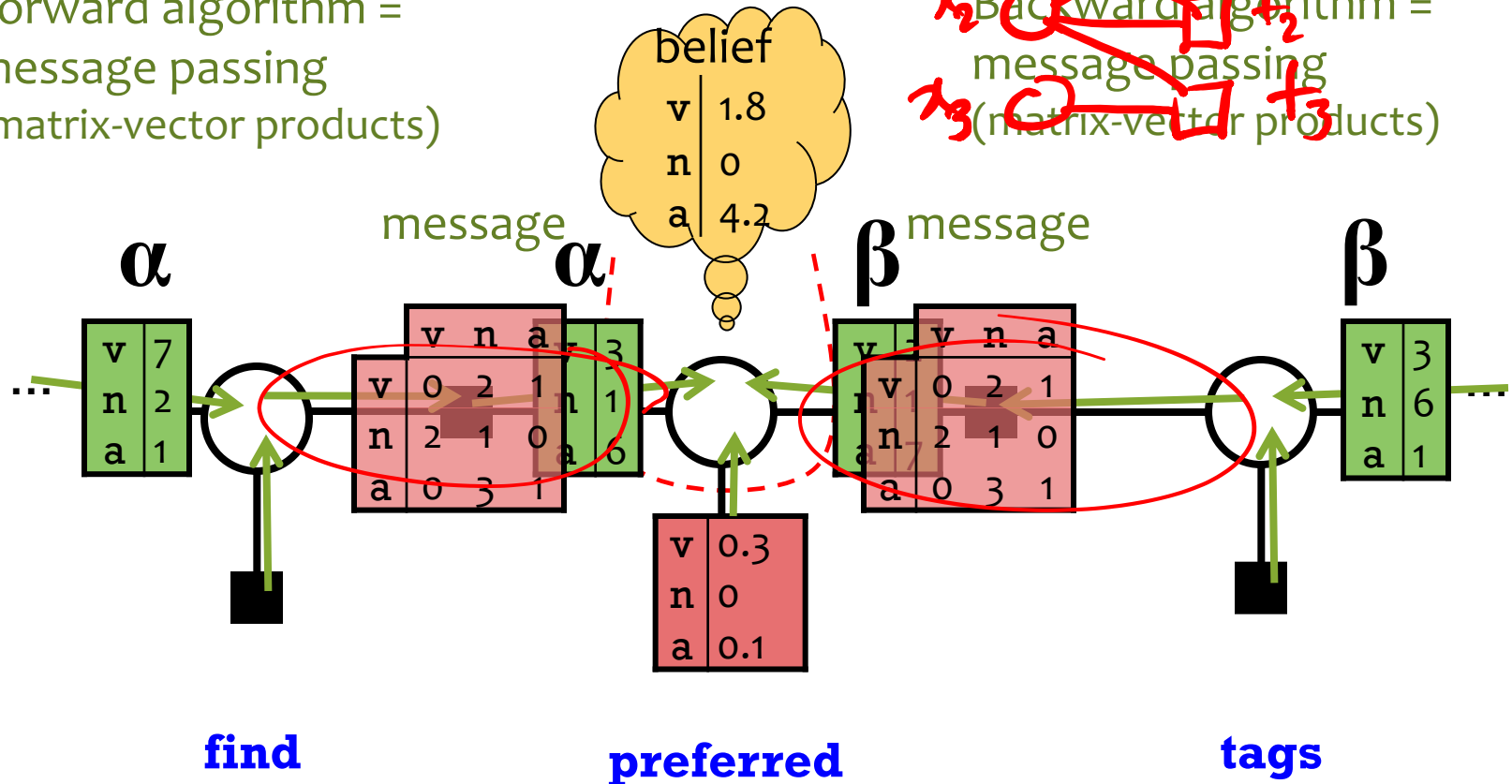
Could be adjective or verb

Could be noun or verb

CRF Tagging by Belief Propagation

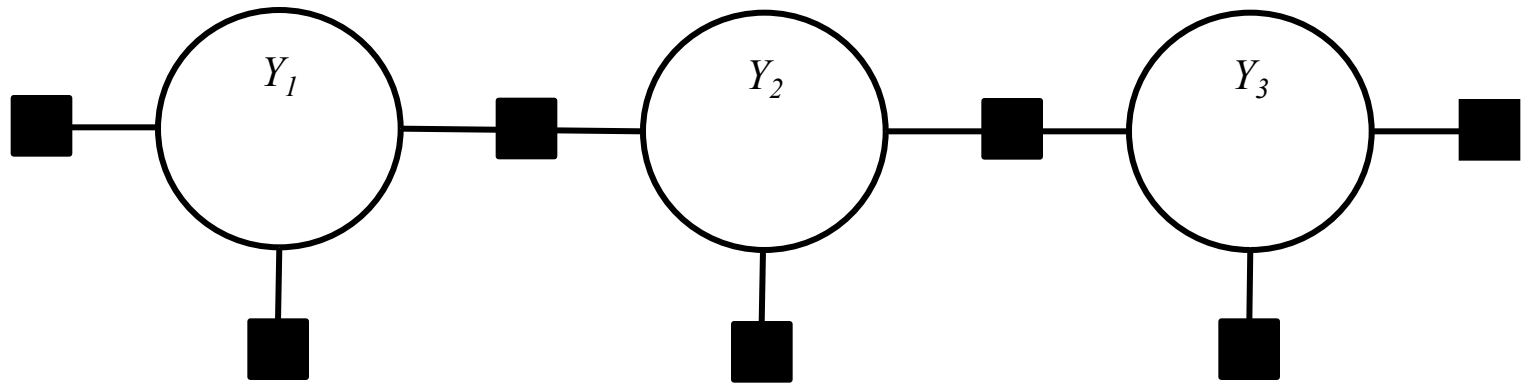
Forward algorithm =
message passing
(matrix-vector products)

~~Backward algorithm =
message passing
(matrix-vector products)~~



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

So Let's Review Forward-Backward ...



find

preferred

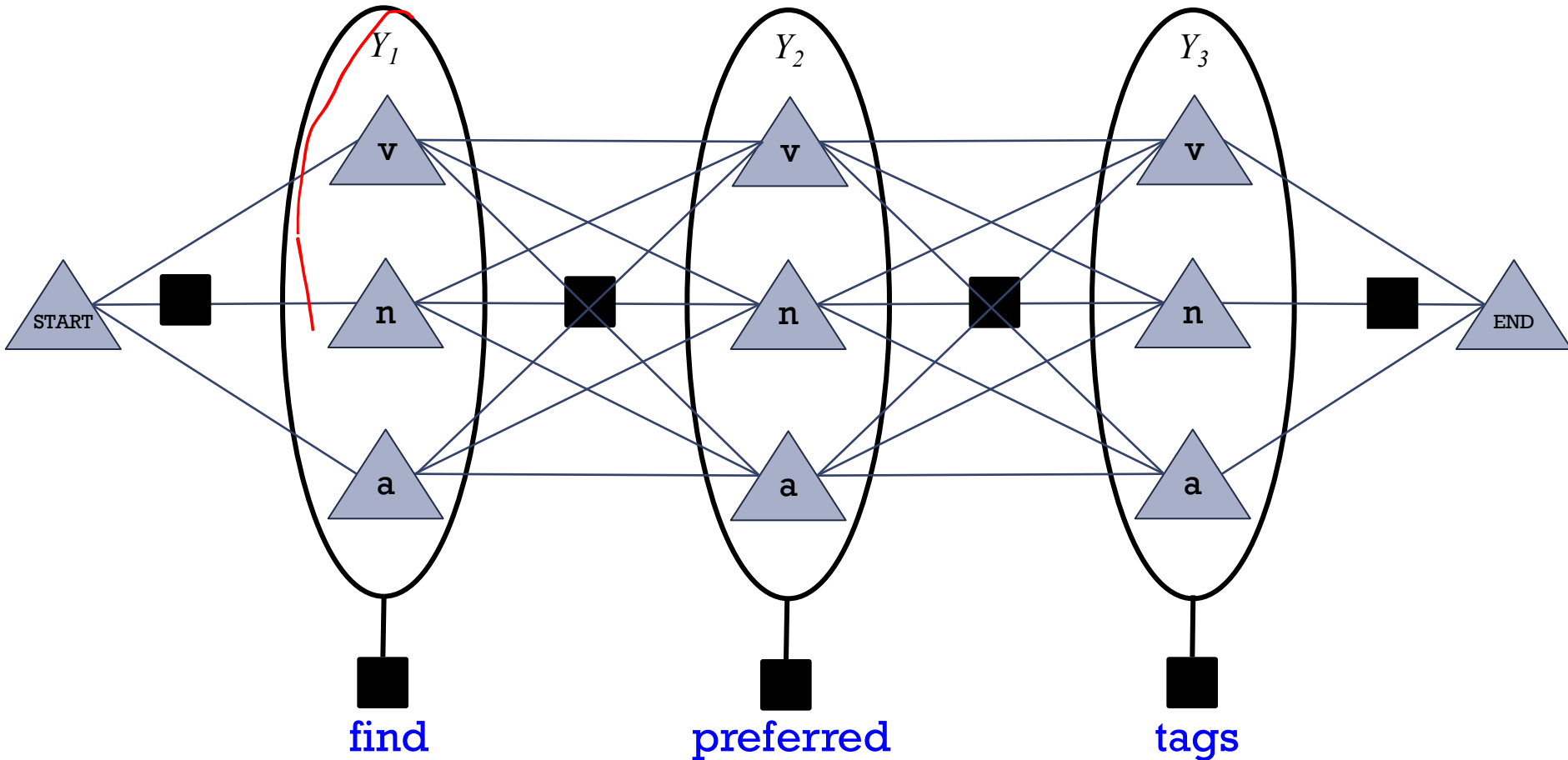
tags

Could be verb or noun

Could be adjective or verb

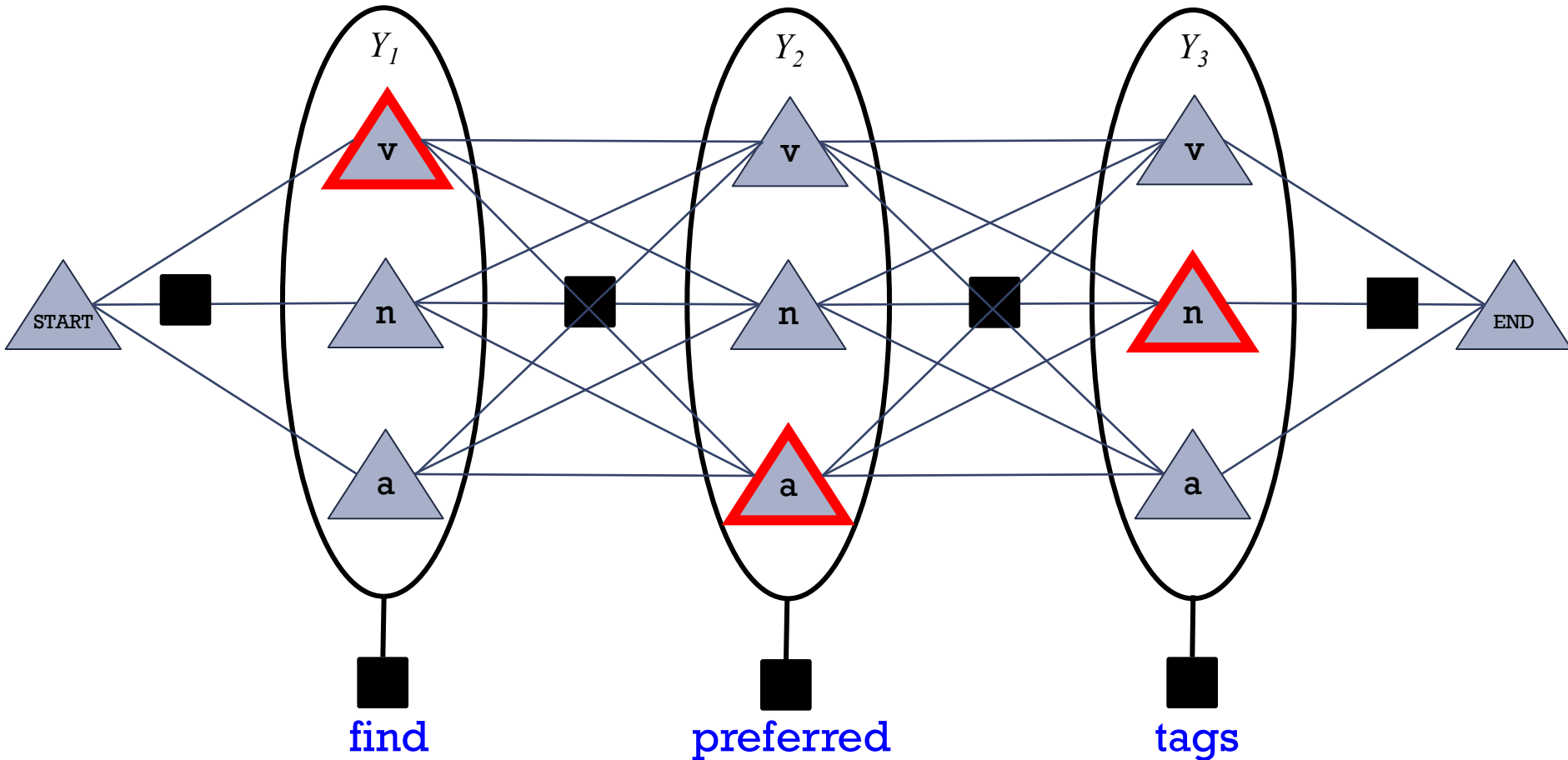
Could be noun or verb

So Let's Review Forward-Backward ...



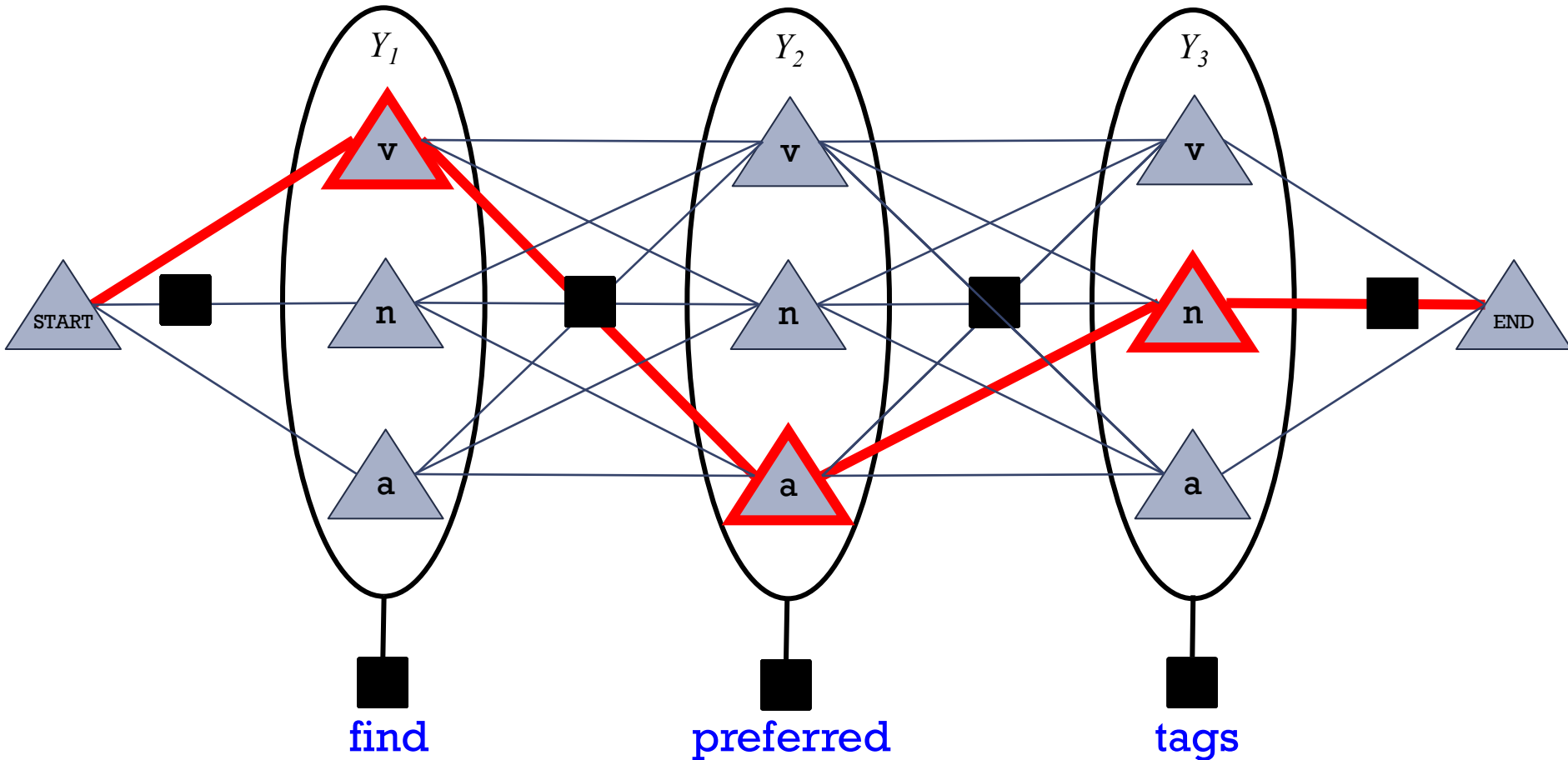
- Show the possible *values* for each variable

So Let's Review Forward-Backward ...



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

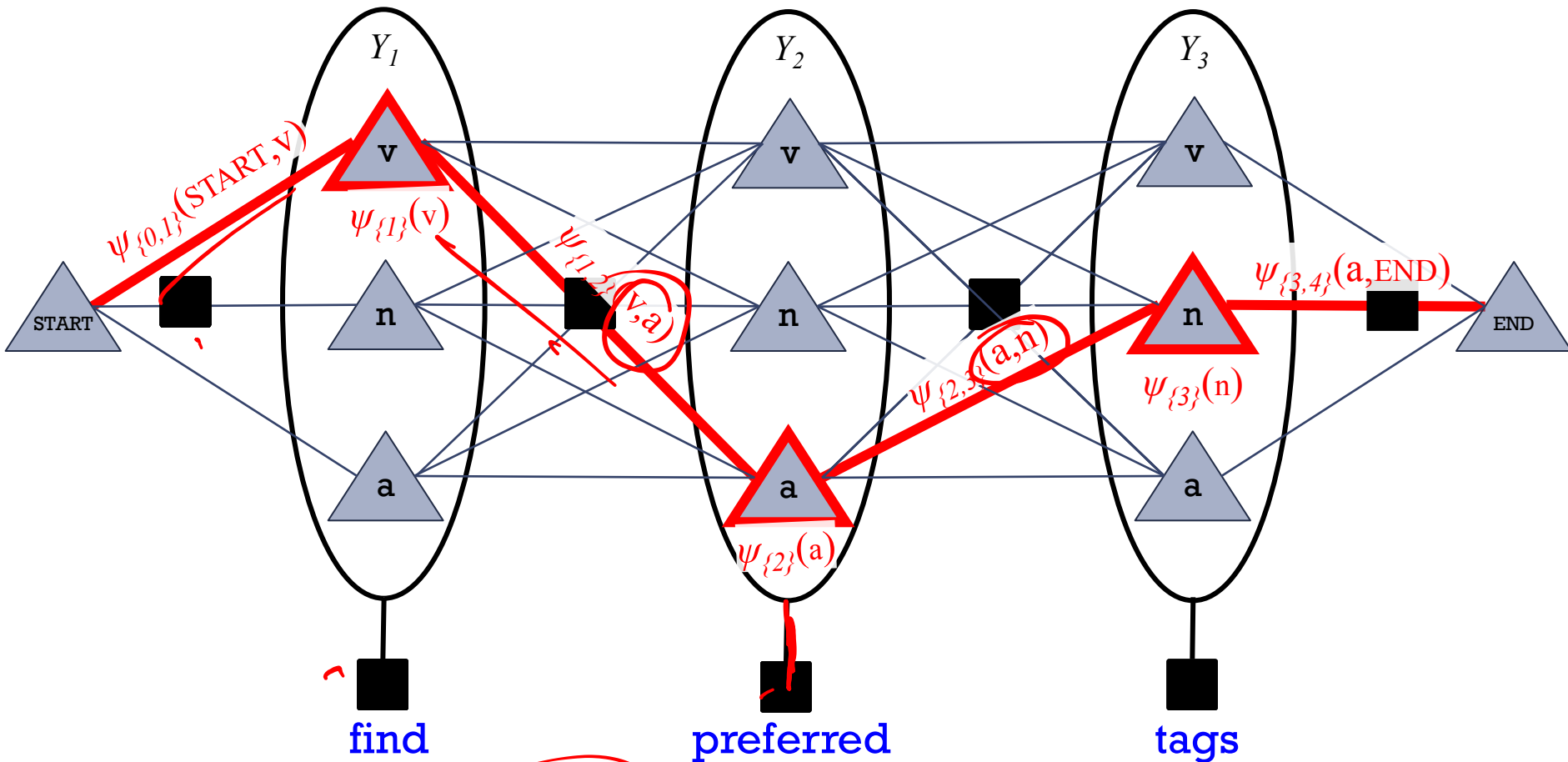
So Let's Review Forward-Backward ...



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

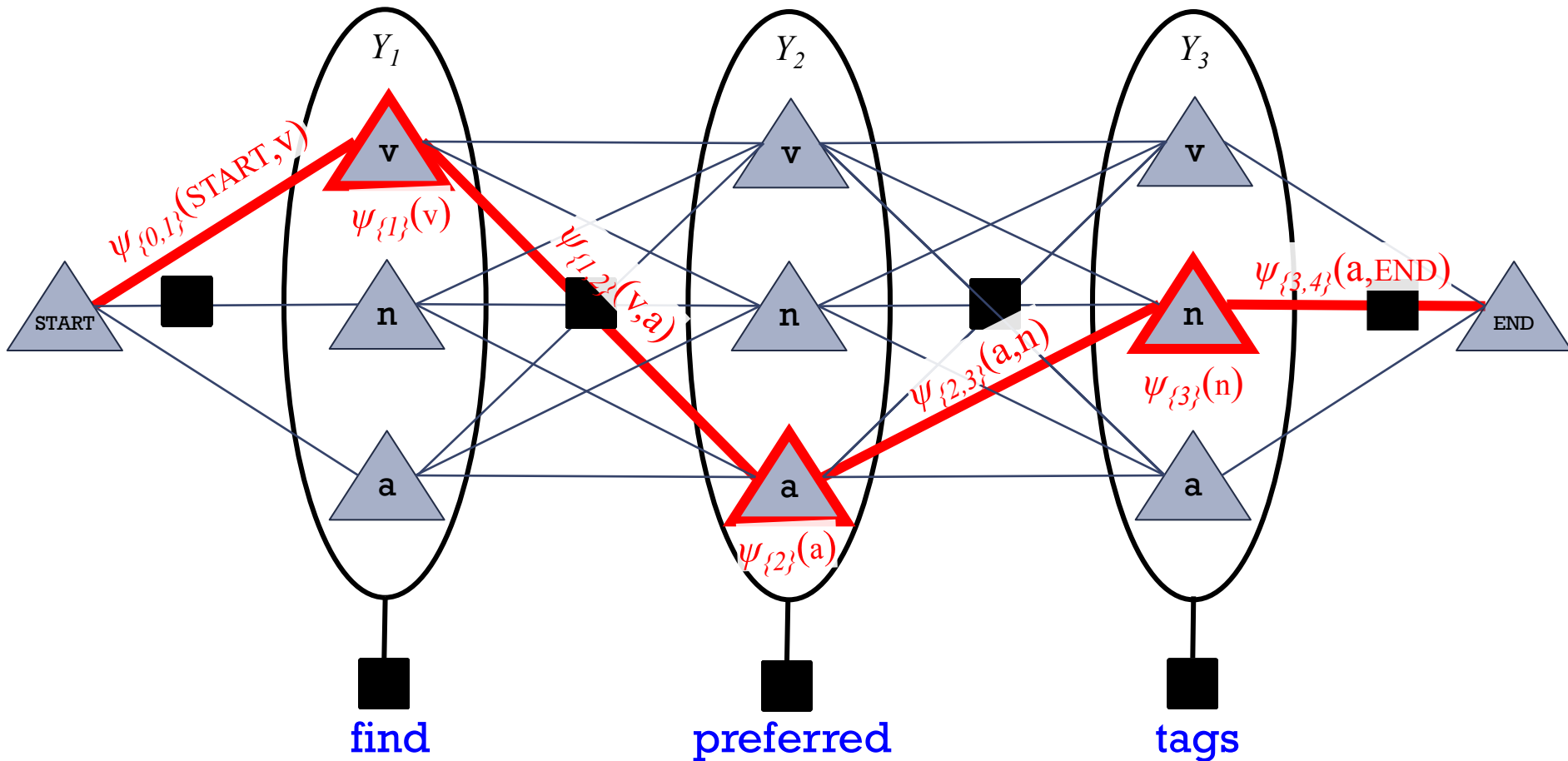
Viterbi Algorithm: Most Probable Assignment

$$P(X) = \frac{1}{Z} \prod_c \psi_c(x_c)$$



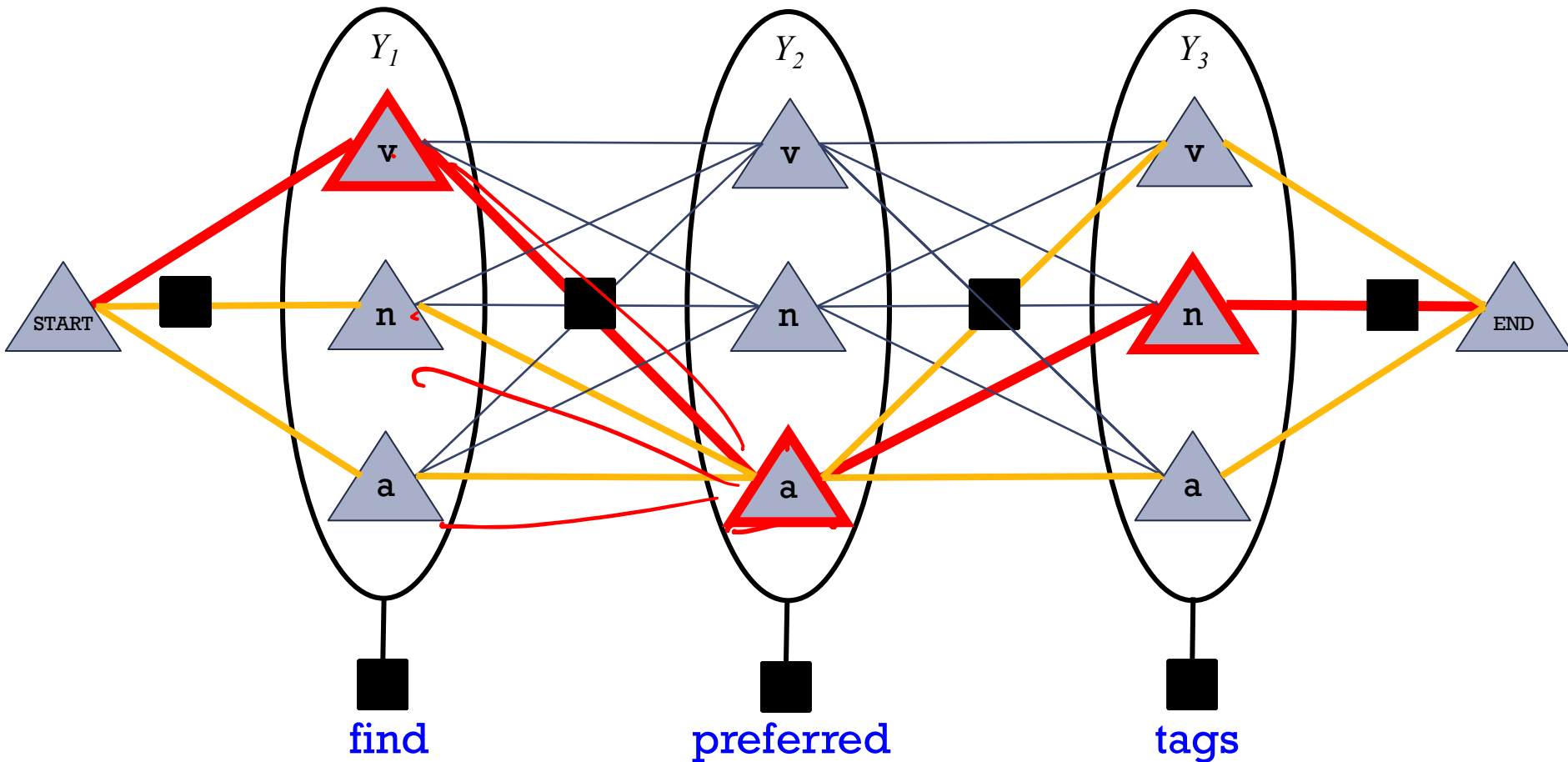
- So $p(v \ a \ n) = (1/Z) * \text{product of 7 numbers}$
- Numbers associated with edges and nodes of path
- Most probable assignment = **path with highest product**

Viterbi Algorithm: Most Probable Assignment

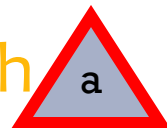


- So $p(v \ a \ n) = (1/Z) * \text{product weight of one path}$

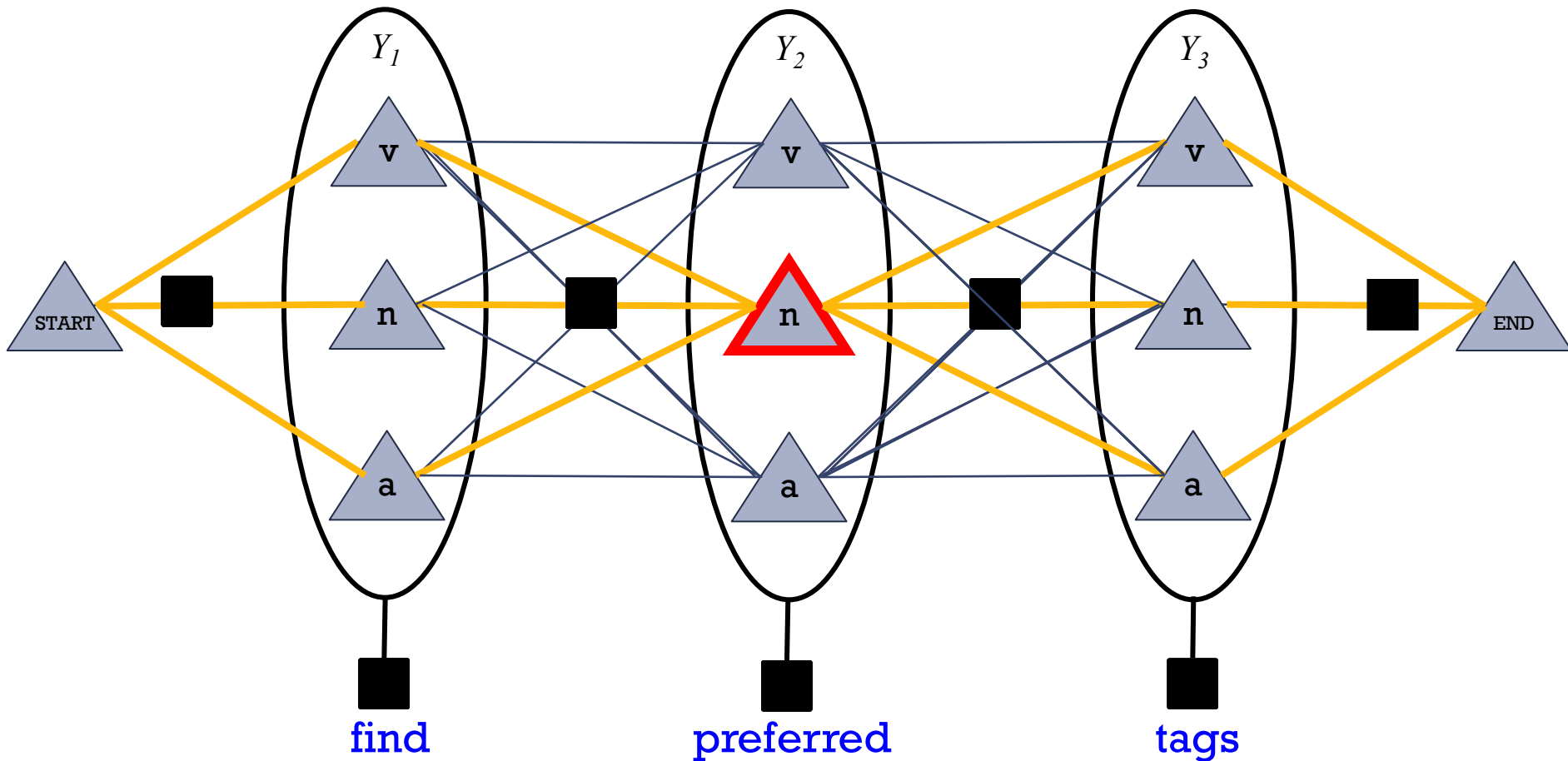
Forward-Backward Algorithm: Finds Marginals



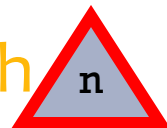
- So $p(\mathbf{v} \mathbf{a} \mathbf{n}) = (1/Z) * \text{product weight of one path}$
- Marginal probability $p(Y_2 = a)$
 $= (1/Z) * \text{total weight of all paths through}$



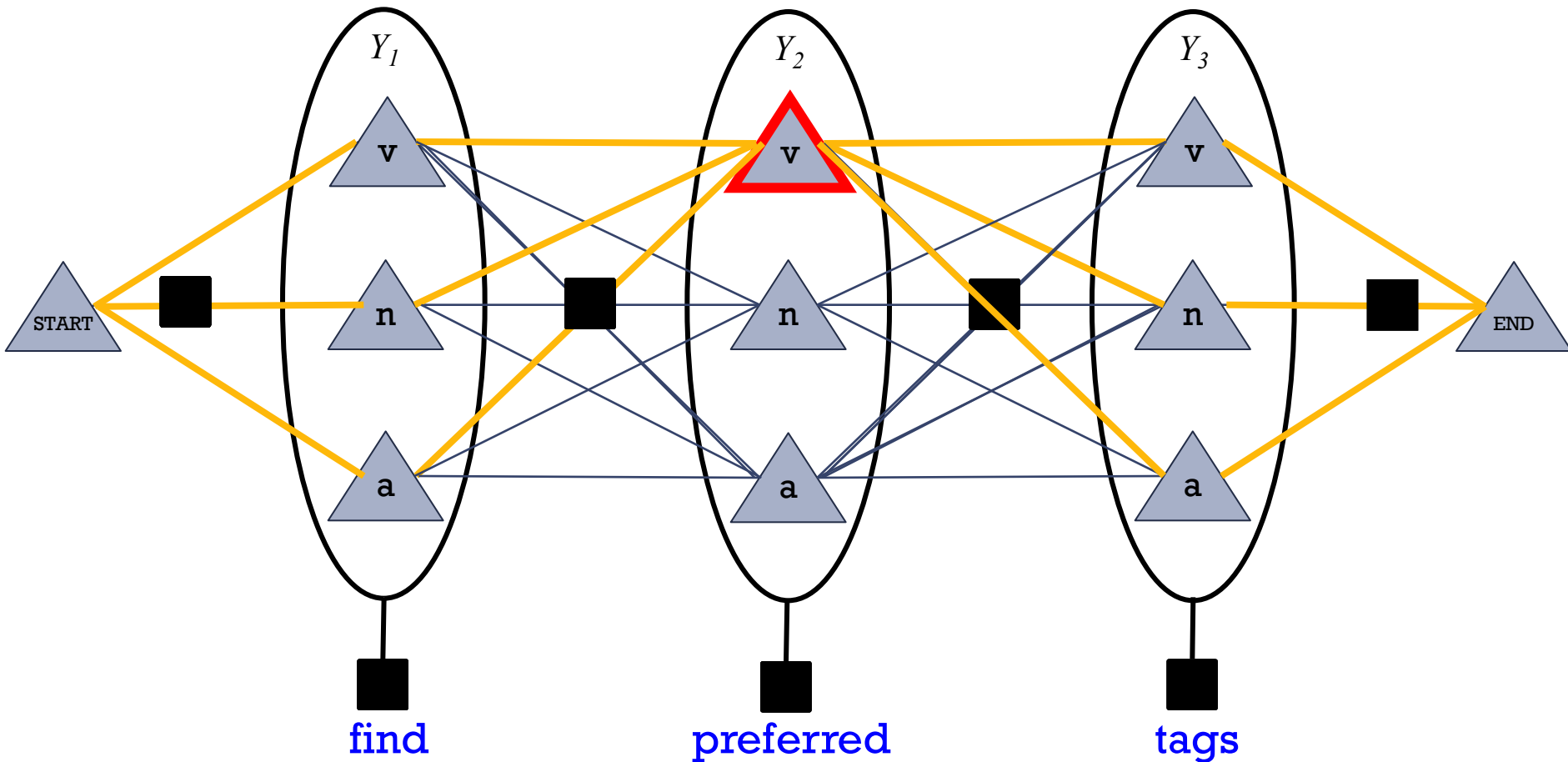
Forward-Backward Algorithm: Finds Marginals



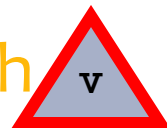
- So $p(v \ a \ n) = (1/Z) * \text{product weight of one path}$
- Marginal probability $p(Y_2 = a)$
 $= (1/Z) * \text{total weight of all paths through}$



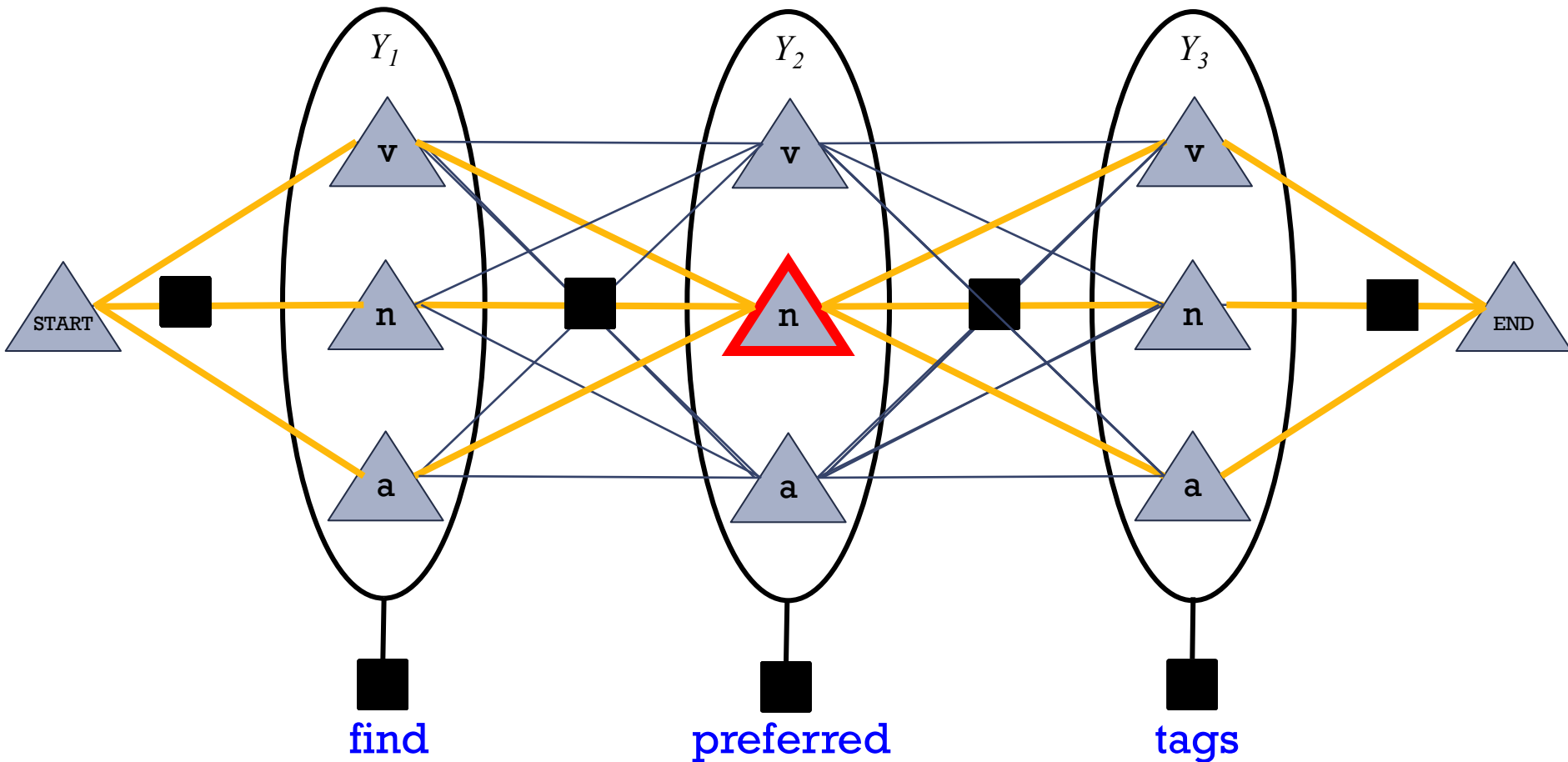
Forward-Backward Algorithm: Finds Marginals



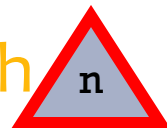
- So $p(\mathbf{v} \mathbf{a} \mathbf{n}) = (1/Z) * \text{product weight of one path}$
- Marginal probability $p(Y_2 = \mathbf{a})$
 $= (1/Z) * \text{total weight of all paths through}$



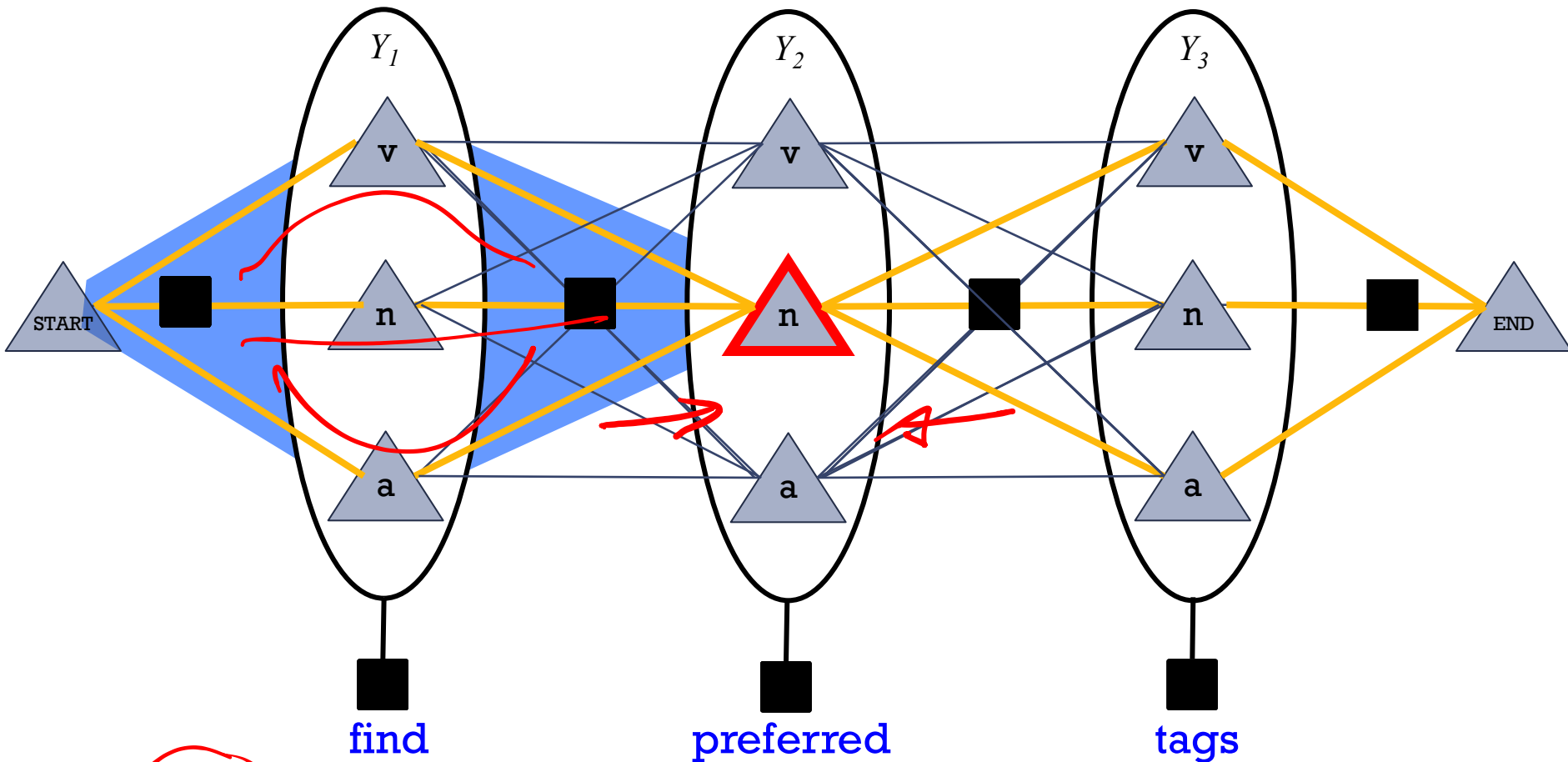
Forward-Backward Algorithm: Finds Marginals



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



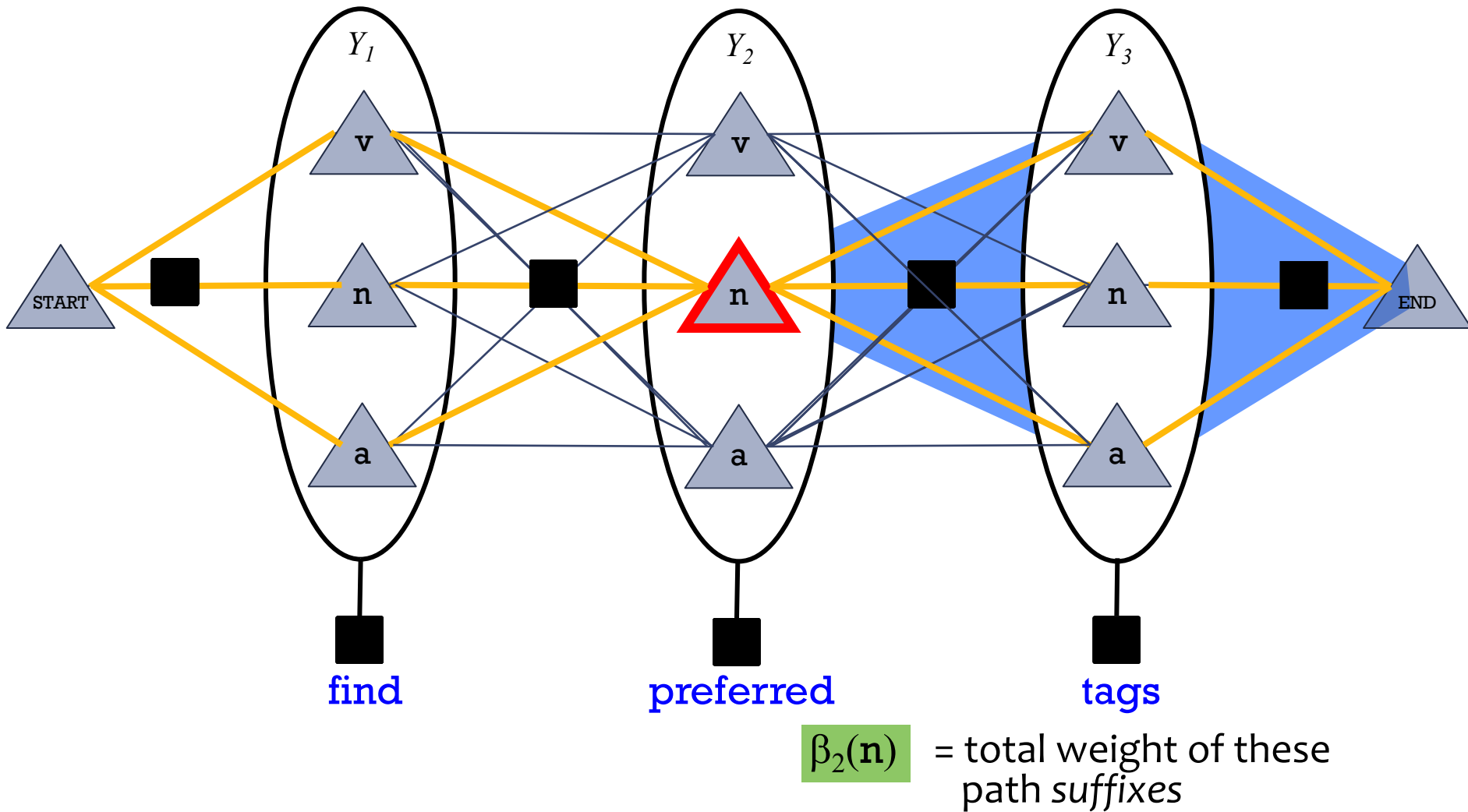
Forward-Backward Algorithm: Finds Marginals



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

(found by dynamic programming: matrix-vector products)

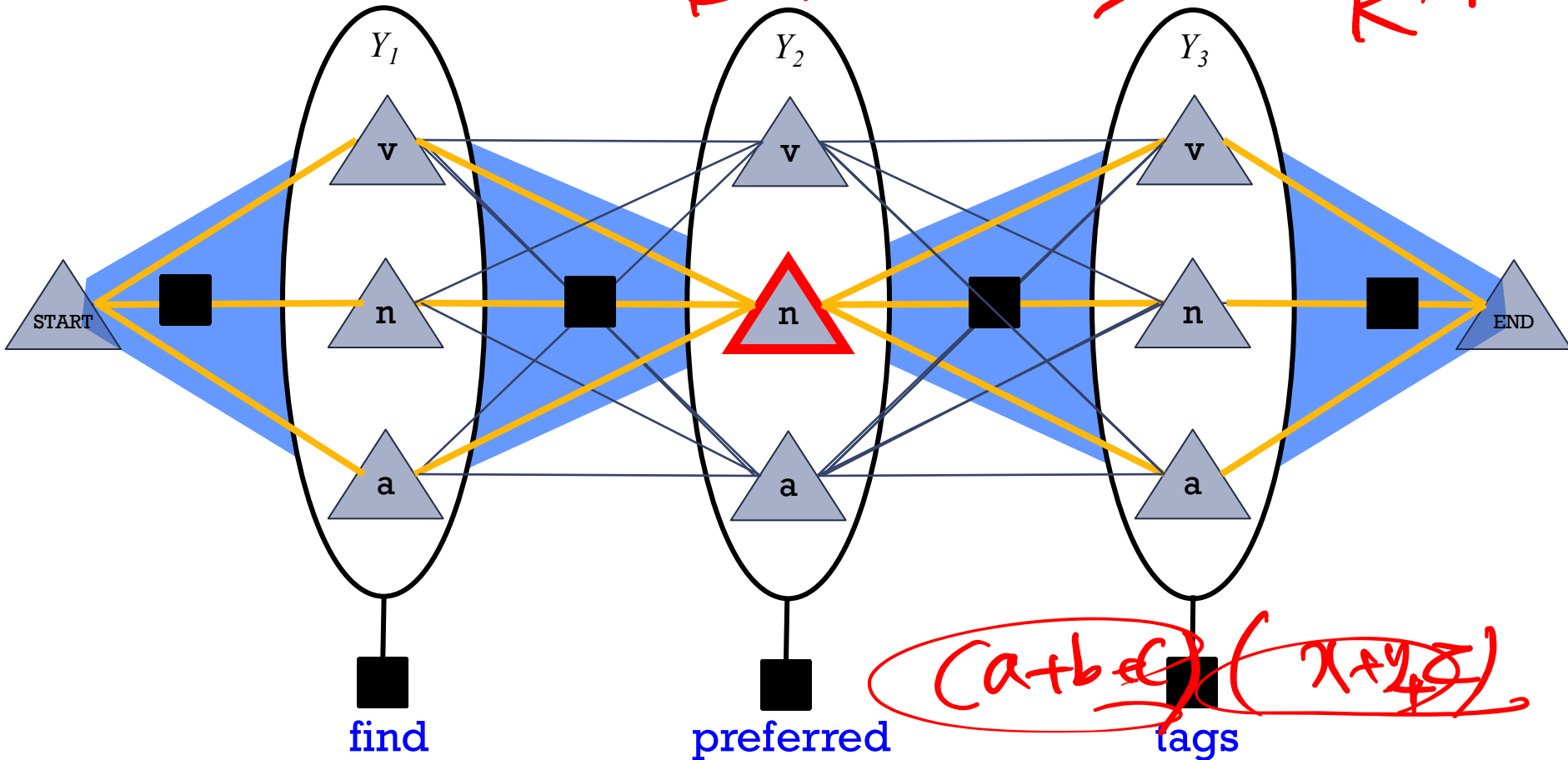
Forward-Backward Algorithm: Finds Marginals



(found by dynamic programming: matrix-vector products)

Forward-Backward Algorithm: Finds Marginals

~~$\alpha_1(n)$~~ ~~$\beta_1(n)$~~ $K \rightarrow K^M$



$\alpha_2(n)$ = total weight of these path prefixes $(a+b+c)$
 $a \quad b \quad c$

$\beta_2(n)$ = total weight of these path suffixes $(x+y+z)$
 $x \quad y \quad z$

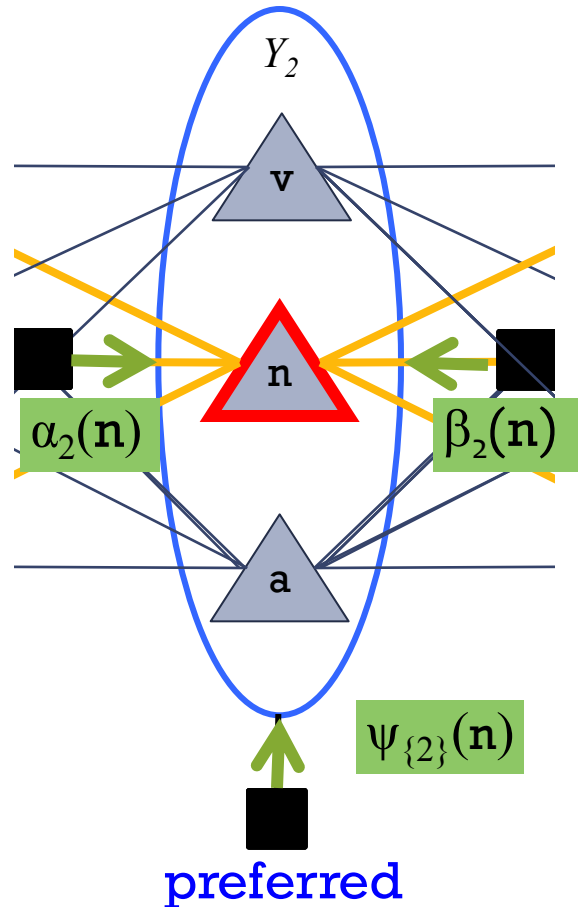
Product gives $ax+ay+az+bx+by+bz+cx+cy+cz$ = total weight of paths

$$P(Y_2 = n) = \alpha_2(n) \beta_2(n) \psi_2(n)$$

Forward-Backward Algorithm: Finds Marginals

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

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

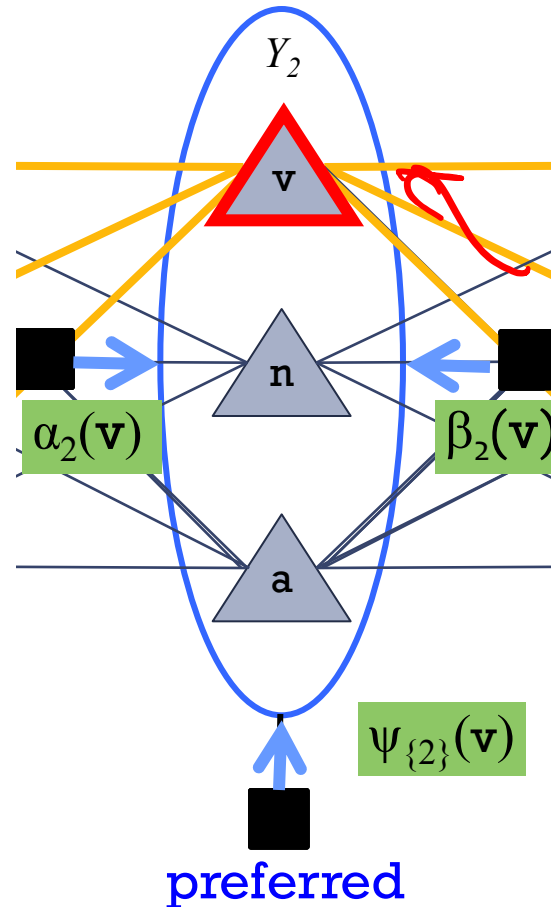


“belief that $Y_2 = n$ ”

total weight of *all paths through* 

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

Forward-Backward Algorithm: Finds Marginals



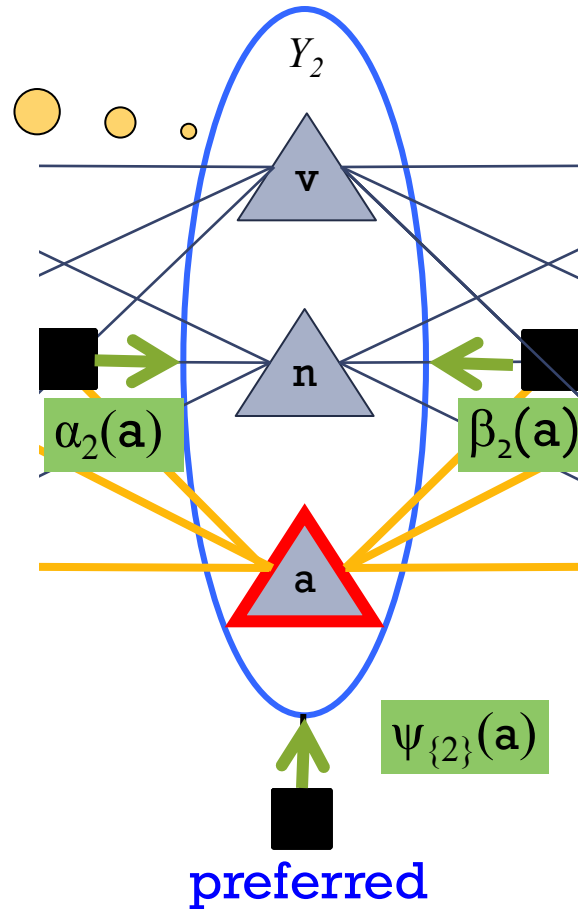
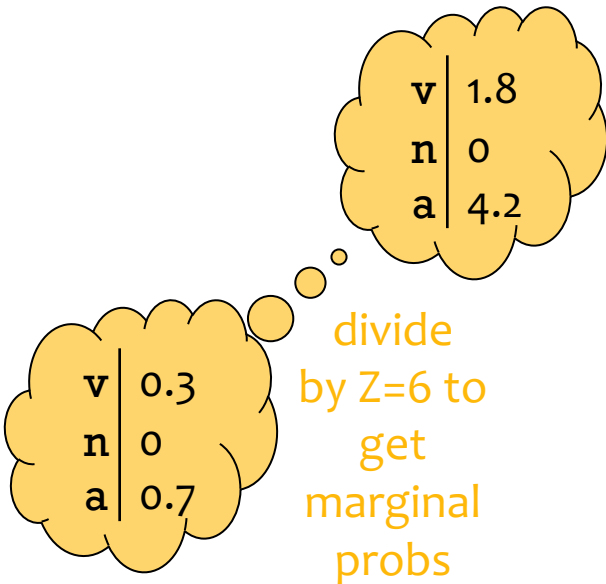
“belief that $Y_2 = v$ ”

“belief that $Y_2 = n$ ”

total weight of *all paths* through 

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

Forward-Backward Algorithm: Finds Marginals



“belief that $Y_2 = \mathbf{v}$ ”

“belief that $Y_2 = \mathbf{n}$ ”

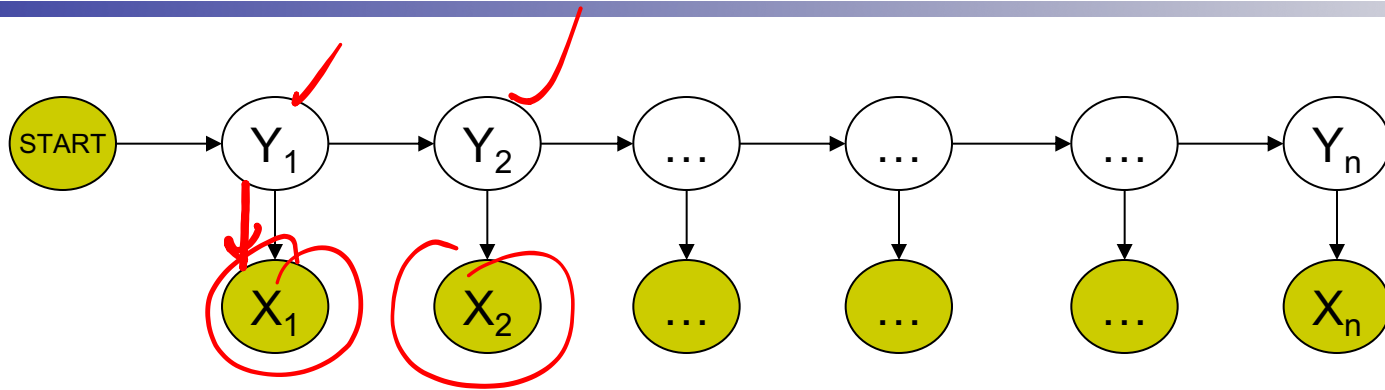
“belief that $Y_2 = \mathbf{a}$ ”

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

total weight of *all* paths through 

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

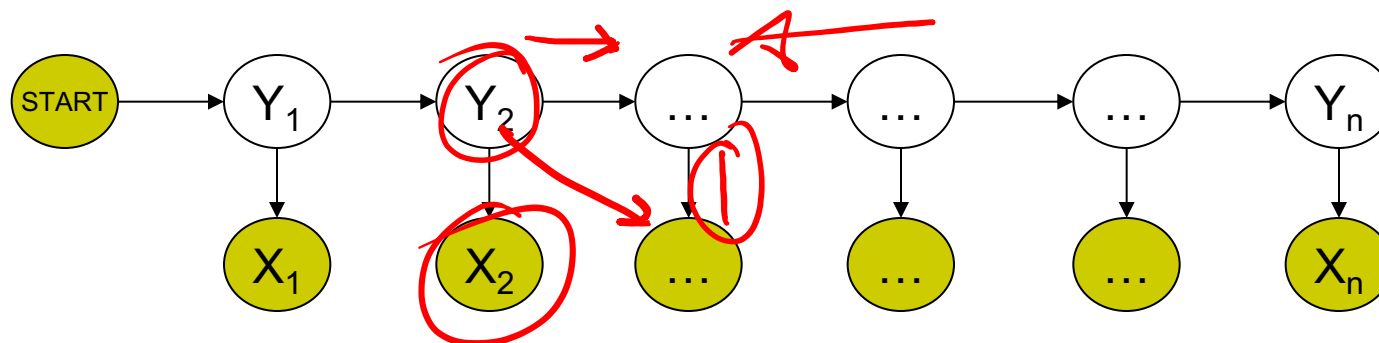
Hidden Markov Model



$$P(\mathbf{x}_{1:n}, \mathbf{y}_{1:n}) = \prod_{i=1}^n P(x_i | y_i) P(y_i | y_{i-1})$$

Handwritten red annotations include a large circle around the entire equation, a circle around $P(y_i | y_{i-1})$, and a circle around $P(x_i | y_i)$. A red arrow points to the n in the product index.

Shortcomings of Hidden Markov Model (1): locality of features

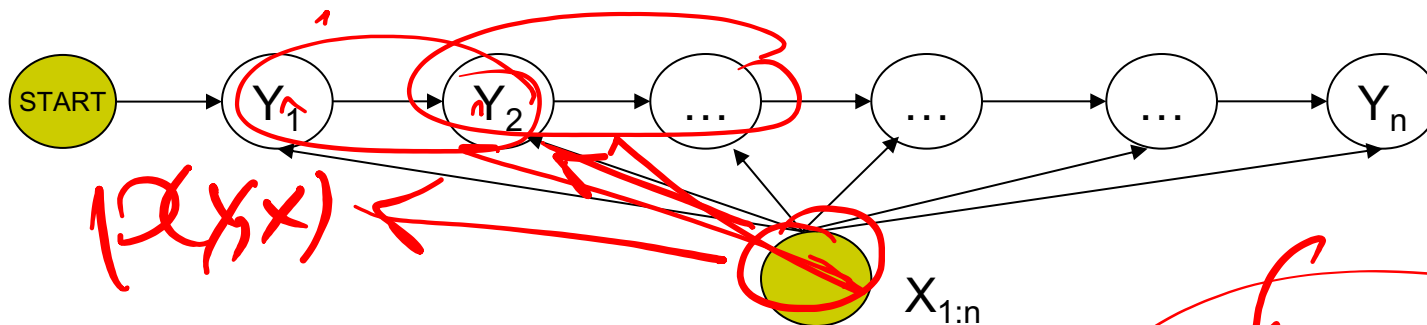


- HMM models capture dependences between each state and **only** its corresponding observation
 - NLP example: In a sentence segmentation task, each segmental state may depend not just on a single word (and the **adjacent segmental stages**), but also on the **(non-local) features** of the whole line 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 $P(Y|X)$

A Solution:

Maximum Entropy Markov Model (MEMM)

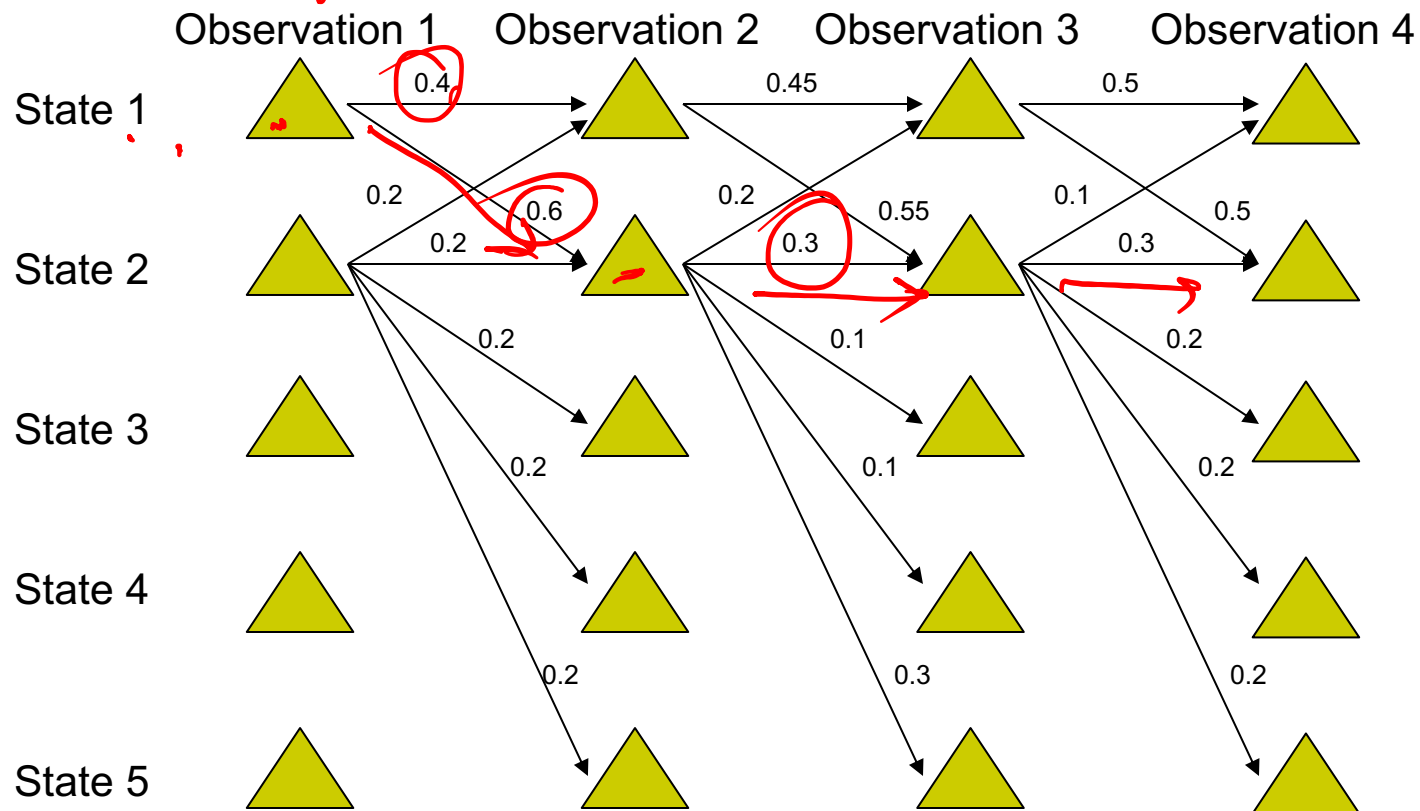
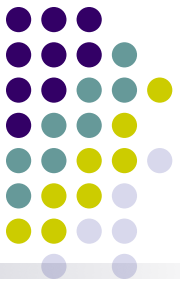
$X_{1:n}$ = find preferred tag



$$P(\mathbf{y}_{1:n} | \mathbf{x}_{1:n}) = \prod_{i=1}^n P(y_i | y_{i-1}, \mathbf{x}_{1:n}) = \prod_{i=1}^n \frac{\exp(\mathbf{w}^T \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_{1:n}))}{Z(y_{i-1}, \mathbf{x}_{1:n})}$$

- Why not providing the full observation sequence explicitly
 - More expressive than HMMs (not the direction of arrow – no causal interpretation, X is just covariates)
- Discriminative model
 - Completely ignores modeling $P(\mathbf{X})$: saves modeling effort
 - Learning objective function consistent with predictive function: $P(\mathbf{Y} | \mathbf{X})$

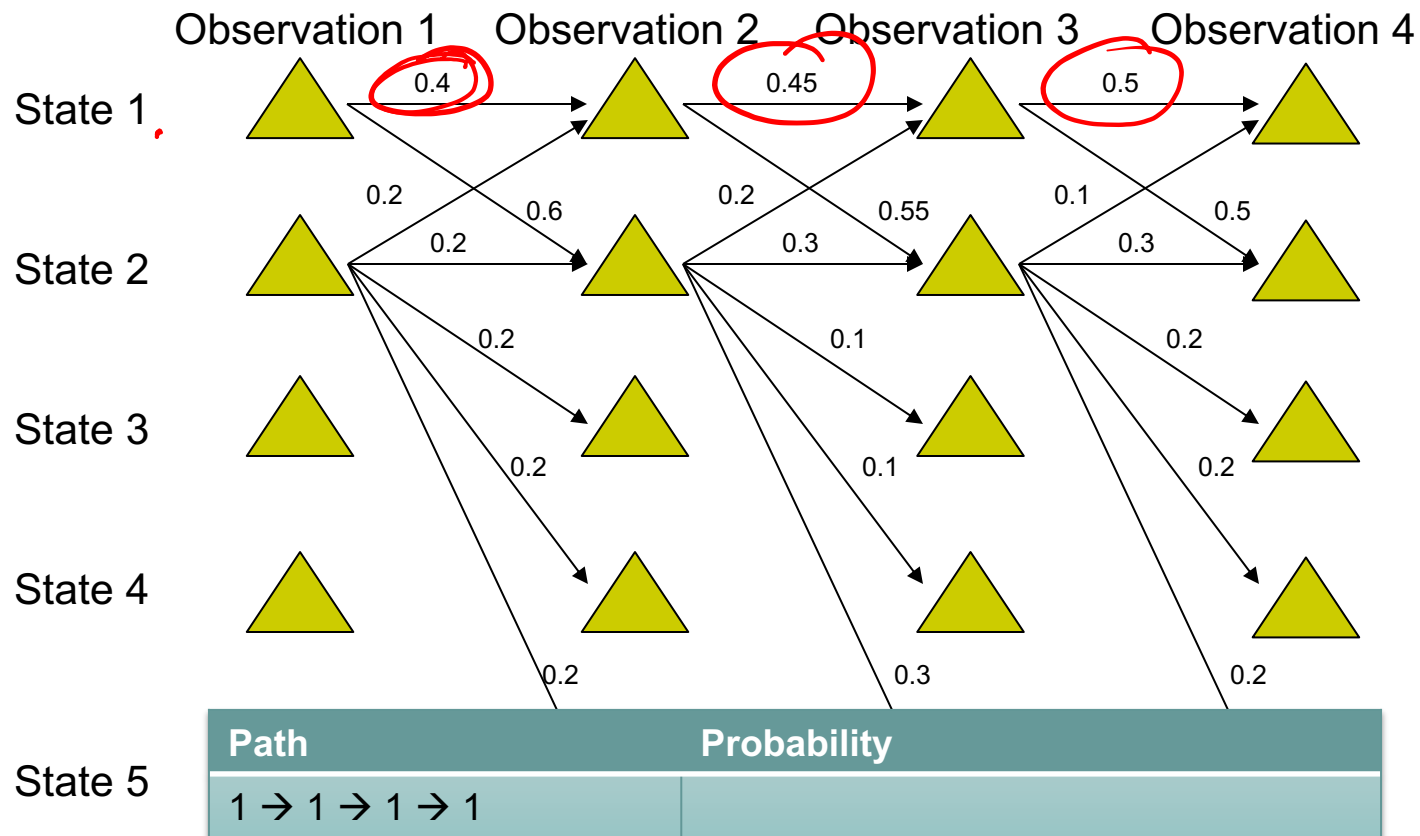
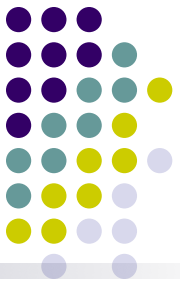
Then, shortcomings of MEMM (and HMM) (2): the Label bias problem

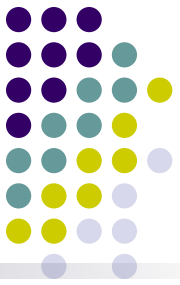


What the local transition probabilities say:

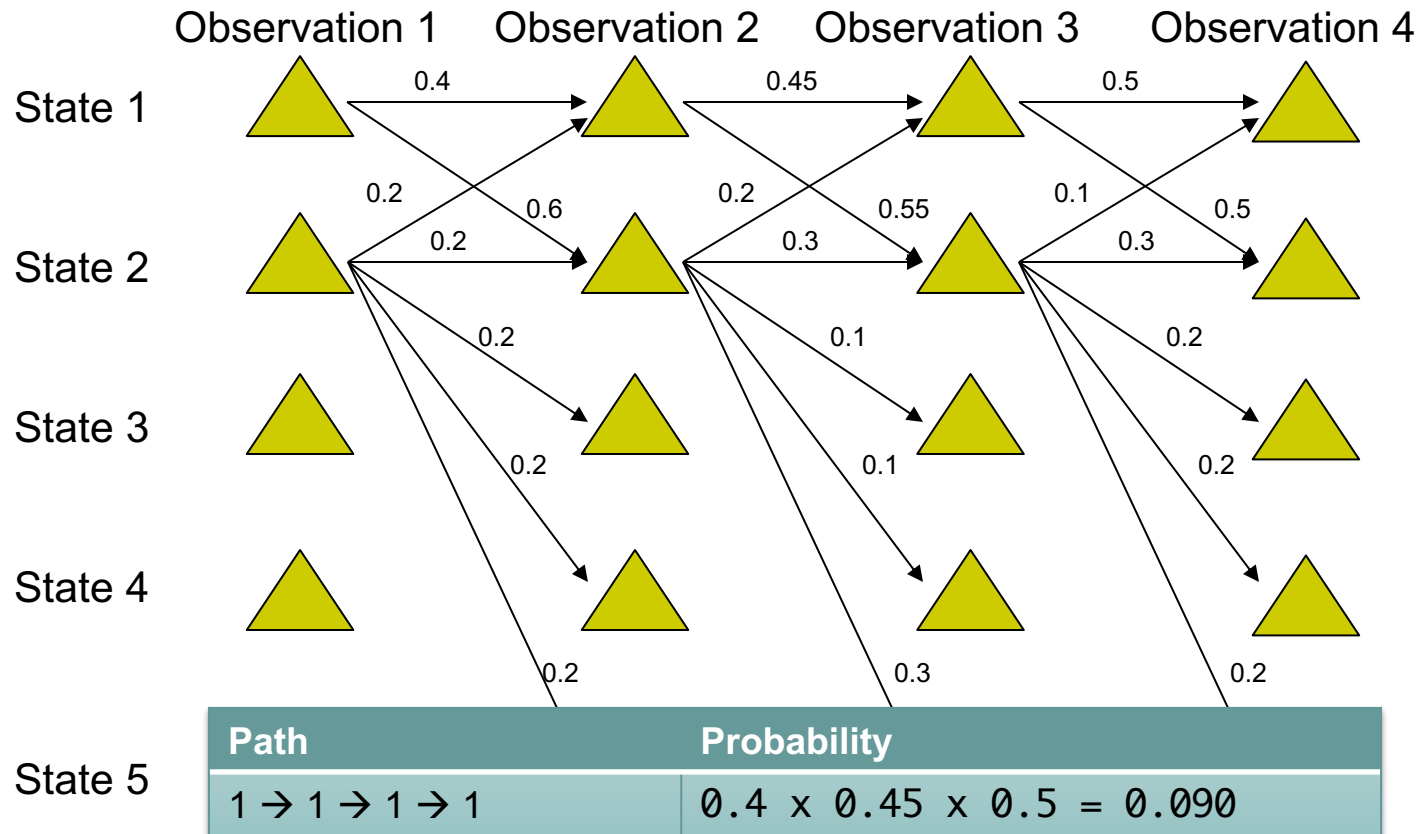
- State 1 almost always prefers to go to state 2
- State 2 almost always prefers to stay in state 2

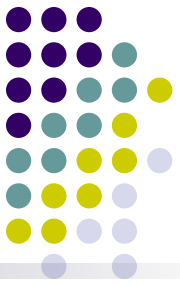
MEMM: the Label bias problem



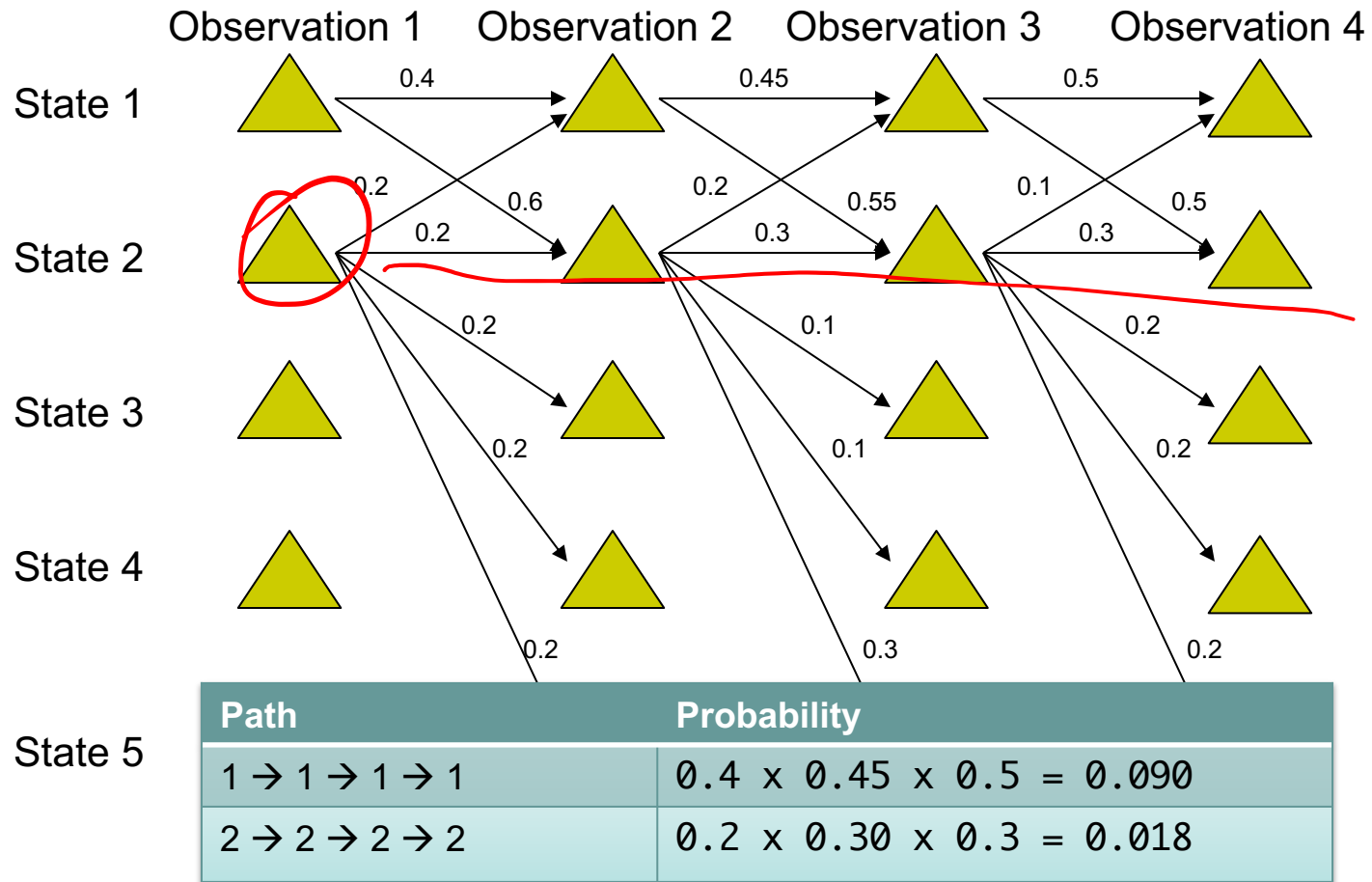


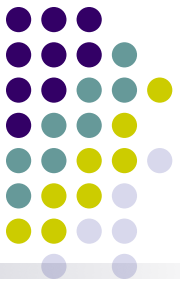
MEMM: the Label bias problem



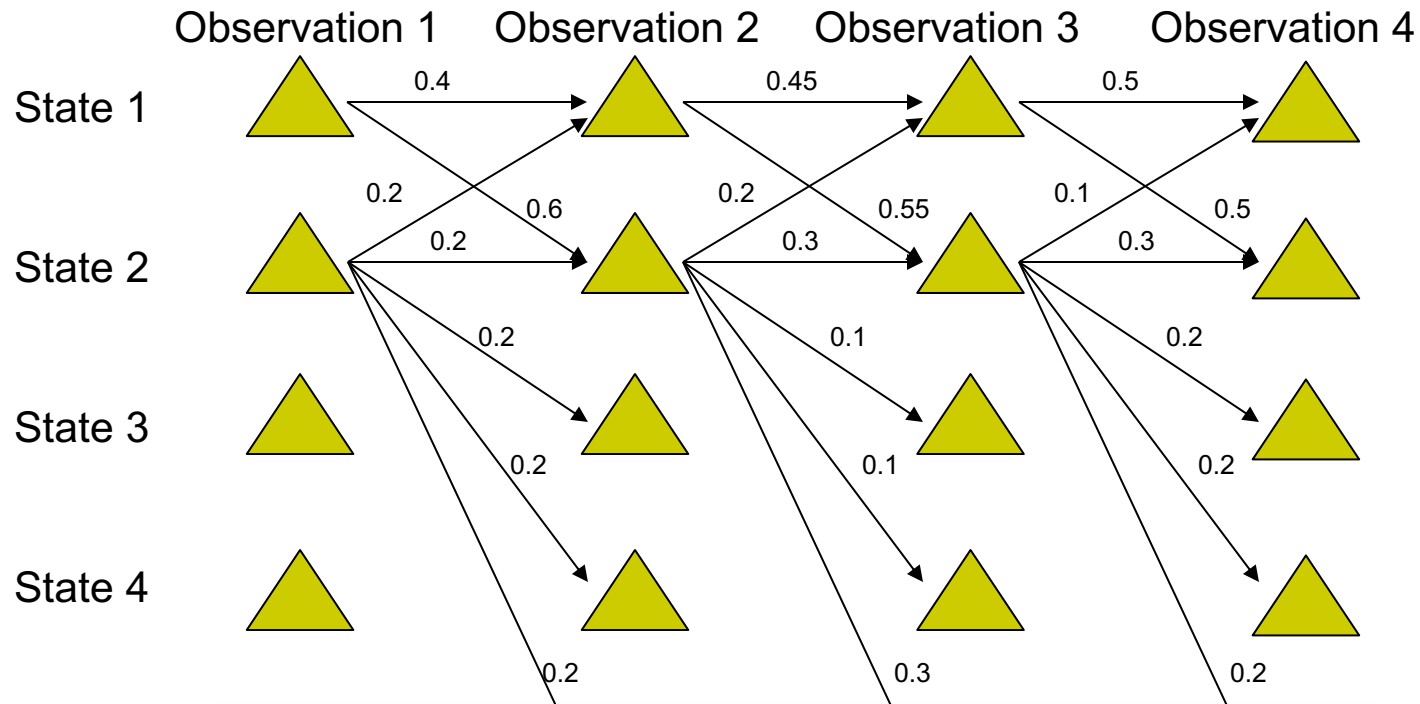


MEMM: the Label bias problem

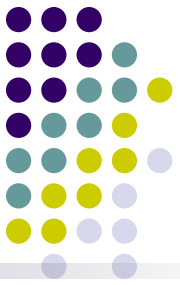




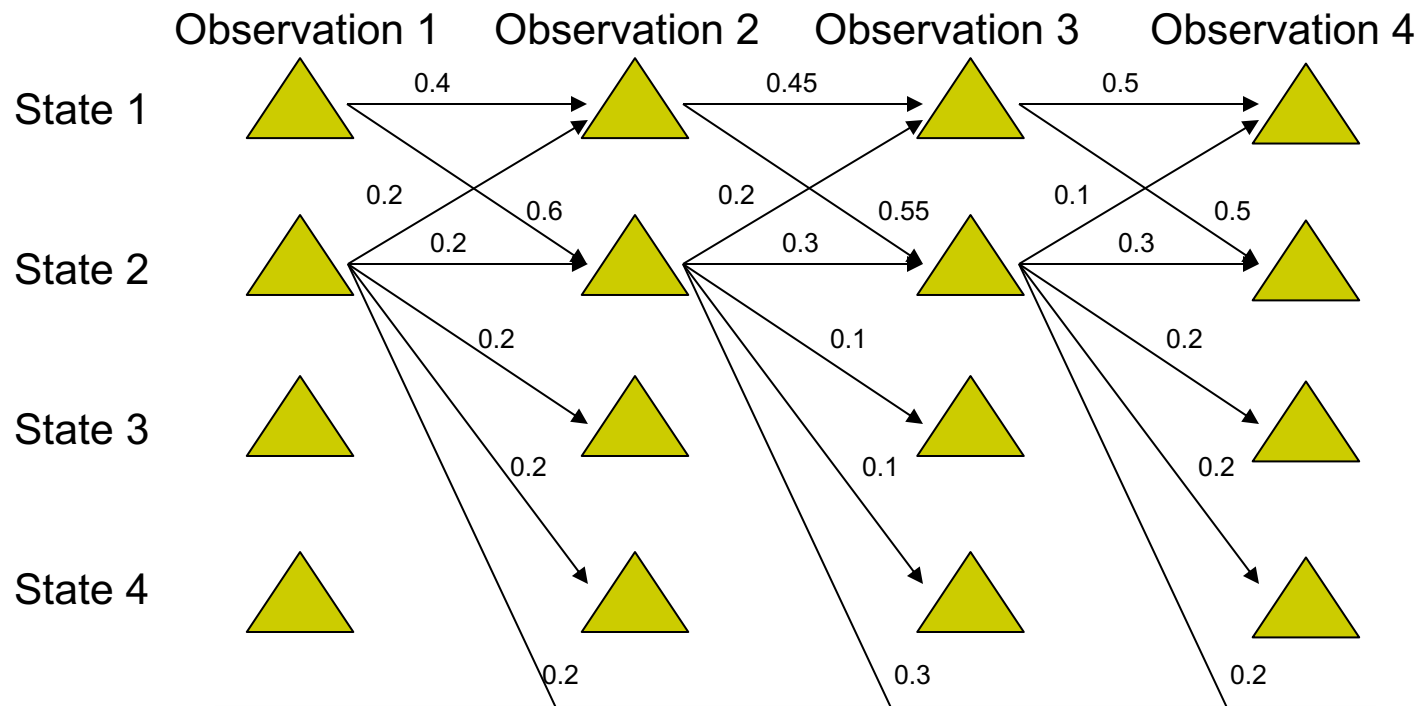
MEMM: the Label bias problem



Path	Probability
1 → 1 → 1 → 1	$0.4 \times 0.45 \times 0.5 = 0.090$
2 → 2 → 2 → 2	$0.2 \times 0.30 \times 0.3 = 0.018$
1 → 2 → 1 → 2	$0.6 \times 0.20 \times 0.5 = 0.060$

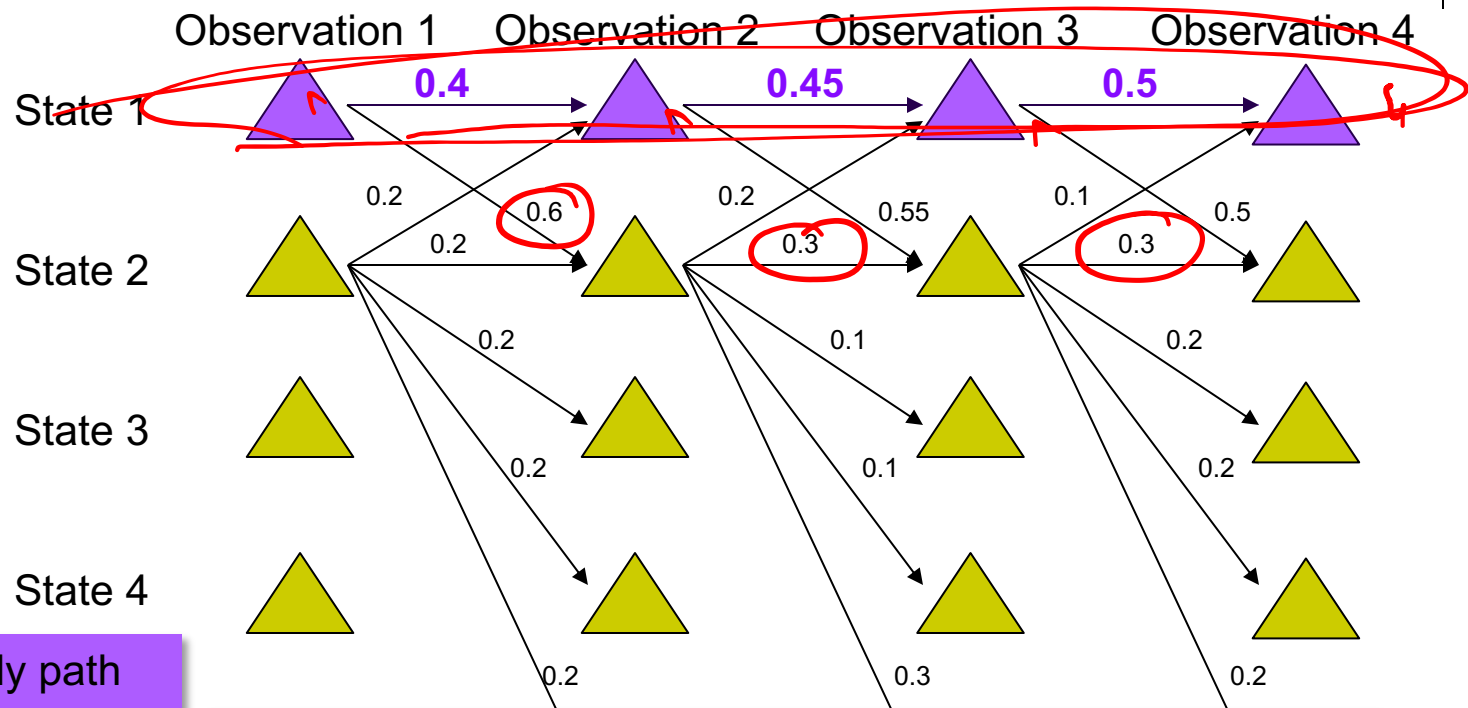
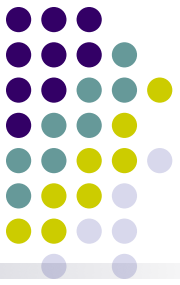


MEMM: the Label bias problem

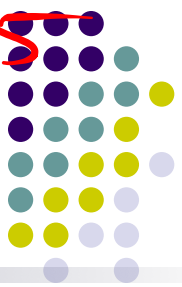


State 5	Path	Probability
State 5	1 → 1 → 1 → 1	$0.4 \times 0.45 \times 0.5 = 0.090$
	2 → 2 → 2 → 2	$0.2 \times 0.30 \times 0.3 = 0.018$
	1 → 2 → 1 → 2	$0.6 \times 0.20 \times 0.5 = 0.060$
	1 → 1 → 2 → 2	$0.4 \times 0.55 \times 0.3 = 0.066$

MEMM: the Label bias problem

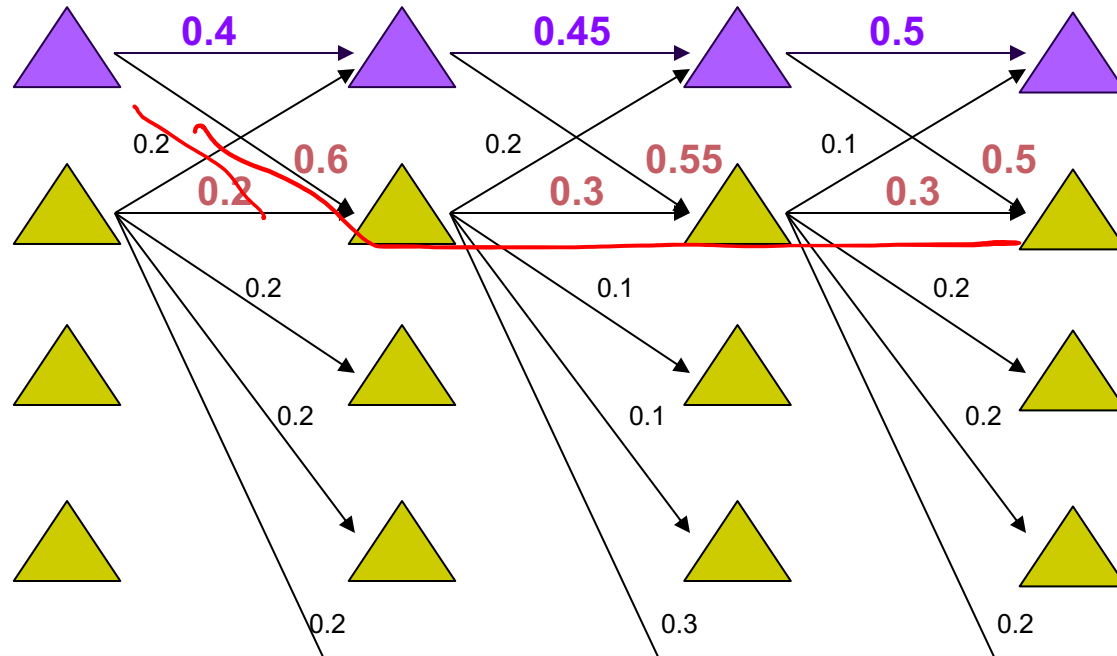


Path	Probability
1 → 1 → 1 → 1	$0.4 \times 0.45 \times 0.5 = 0.090$
2 → 2 → 2 → 2	$0.2 \times 0.30 \times 0.3 = 0.018$
1 → 2 → 1 → 2	$0.6 \times 0.20 \times 0.5 = 0.060$
1 → 1 → 2 → 2	$0.4 \times 0.55 \times 0.3 = 0.066$

$$0.09 \times 0.6 = 0.045$$


MEMM: the Label bias problem

Observation 1 Observation 2 Observation 3 Observation 4



Yet **locally** it seems state 1 wants to go to state 2 and state 2 wants to remain in state 2.

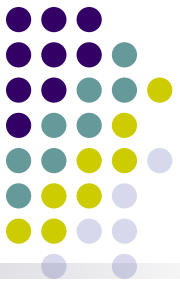
State 3

State 4

Most likely path

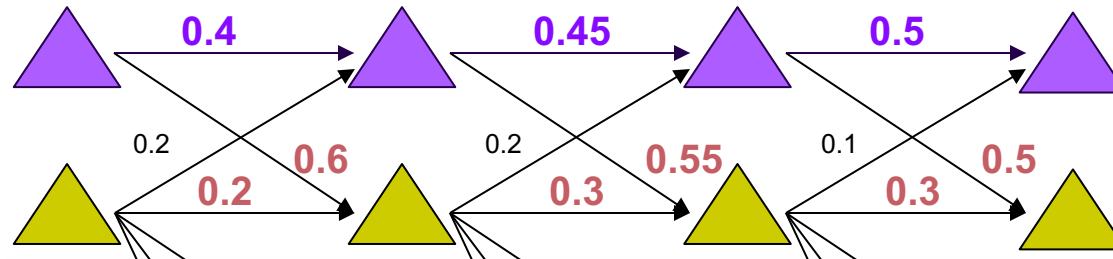
State 5

Path	Probability
1 → 1 → 1 → 1	$0.4 \times 0.45 \times 0.5 = 0.090$
2 → 2 → 2 → 2	$0.2 \times 0.30 \times 0.3 = 0.018$
1 → 2 → 1 → 2	$0.6 \times 0.20 \times 0.5 = 0.060$
1 → 1 → 2 → 2	$0.4 \times 0.55 \times 0.3 = 0.066$



MEMM: the Label bias problem

Observation 1 Observation 2 Observation 3 Observation 4



Yet **locally** it seems state 1 wants to go to state 2 and state 2 wants to remain in state 2.

State 3

State 4

Why does this happen?

- State 1 has only two transitions but state 2 has 5
- Average transition probability from state 2 is lower

This is the **Label Bias Problem** in MEMM: a preference for states with lower number of transitions over others

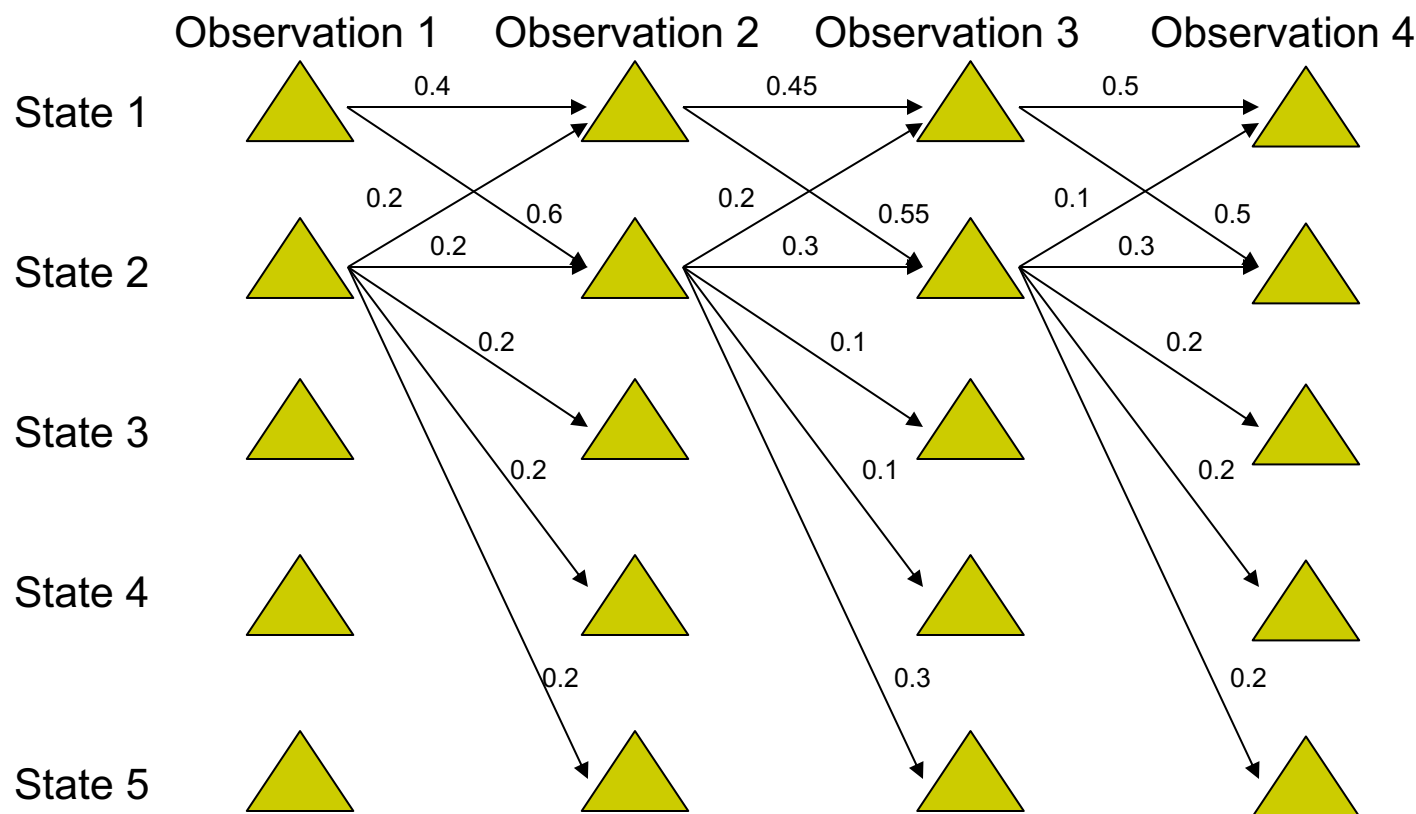
Most likely path

State 5

Path	Probability
1 → 1 → 1 → 1	$0.4 \times 0.45 \times 0.5 = 0.090$
2 → 2 → 2 → 2	$0.2 \times 0.30 \times 0.3 = 0.018$
1 → 2 → 1 → 2	$0.6 \times 0.20 \times 0.5 = 0.060$
1 → 1 → 2 → 2	$0.4 \times 0.55 \times 0.3 = 0.066$

Solution:

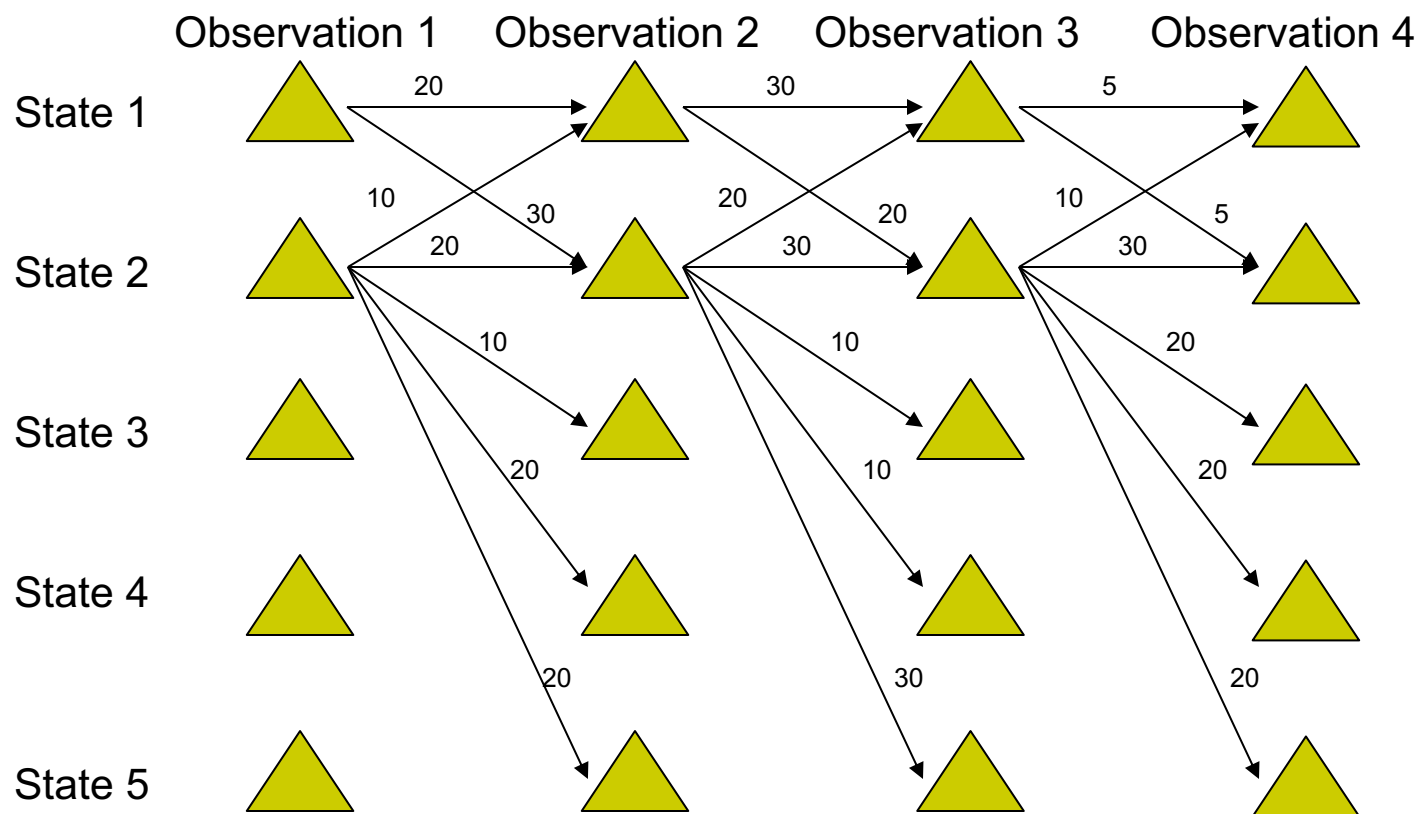
Do not normalize probabilities locally



From local probabilities...

Solution:

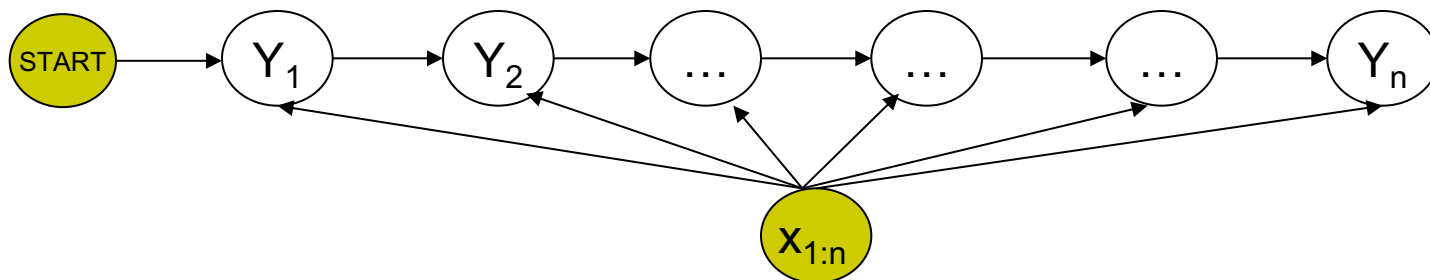
Do not normalize probabilities locally



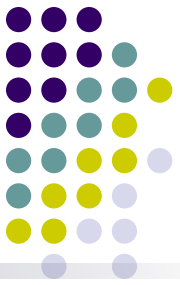
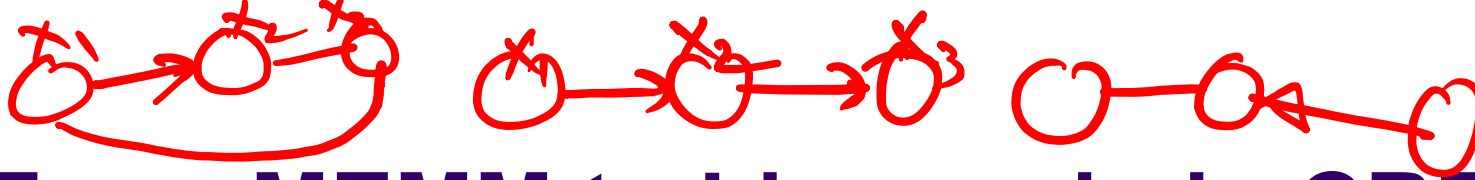
From local probabilities to local potentials!

States with lower transitions do not have an unfair advantage!

From MEMM

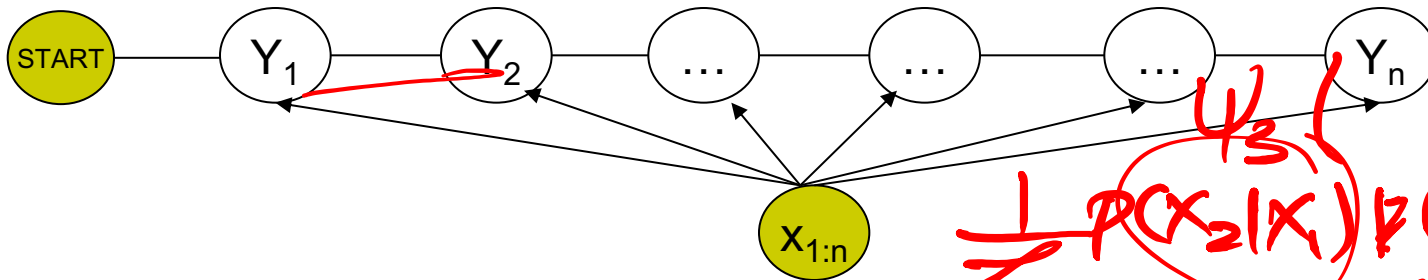


$$P(\mathbf{y}_{1:n}|\mathbf{x}_{1:n}) = \prod_{i=1}^n P(y_i|y_{i-1}, \mathbf{x}_{1:n}) = \prod_{i=1}^n \frac{\exp(\mathbf{w}^T \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_{1:n}))}{Z(y_{i-1}, \mathbf{x}_{1:n})}$$



From MEMM to Linear-chain CRF

$$P(x_1, x_2, x_3) = P(x_1) P(x_2 | x_1) P(x_3 | x_2)$$



$$P(\mathbf{y}_{1:n} | \mathbf{x}_{1:n}) = \frac{1}{Z(\mathbf{x}_{1:n})} \prod_{i=1}^n \phi(y_i, y_{i-1}, \mathbf{x}_{1:n}) = \frac{1}{Z(\mathbf{x}_{1:n}, \mathbf{w})} \prod_{i=1}^n \exp(\mathbf{w}^T \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_{1:n}))$$

- CRF is a **partially** directed model

- Discriminative model like MEMM
- Unlike MEMM, each **factor is not normalized**. Hence, usage of global $Z(\mathbf{x})$ overcomes the label bias problem of MEMM
- Models the dependence between each state and the entire observation sequence (like MEMM)

~~X + Y~~ $\frac{1}{Z(\theta)} \exp(\theta^T T(x) - A(\theta)) h(x)$
chain CRF $g(y_i, x)$
 $p(y_i, x)$

- $$y, y_{i-1}, y_{i-2}$$



$$[\lambda \mu]^T [g]$$

$$\lambda^T f(y_i, y_{i-1}, x) + \nu^T h(y_i, y_{i-1}, y_{i-2}, x) -$$

Whiteboard

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

(side-by-side with MRF)

Learning and Inference Summary

For discrete variables:

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

Features

General idea:

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

Features for tagging ...

$f(x, y)$

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 ...



- 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 ...

$f(y_i, y_{i-1}, x)$

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 ...

$f(y_i, y_{i-1}, y_{i-2})$

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.
 - Why?** 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.

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 (\mathbf{x}, \mathbf{y}) ?

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

E.g., template 7 might be $(\text{tag}(i-1), \text{tag}(i), \text{suffix2}(i+1))$.

At each position of (\mathbf{x}, \mathbf{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 $\text{score}(\mathbf{x}, \mathbf{y})$

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 $(\text{tag}(i-1), \text{tag}(i), \text{suffix2}(i+1))$.

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 $\text{score}(x,y)$

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 $(\text{tag}(i-1), \text{tag}(i), \text{suffix2}(i+1))$.

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 $\text{score}(x,y)$

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 $(\text{tag}(i-1), \text{tag}(i), \text{suffix2}(i+1))$.

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 $\text{score}(x,y)$

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

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

E.g., template 7 might be $(\text{tag}(i-1), \text{tag}(i), \text{suffix2}(i+1))$.

At each position of (\mathbf{x}, \mathbf{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= $(\mathbf{D}, \mathbf{N}, -)$ ”
so we add one copy of that feature’s weight to $\text{score}(\mathbf{x}, \mathbf{y})$

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

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

E.g., template 7 might be $(\text{tag}(i-1), \text{tag}(i), \text{suffix2}(i+1))$.

This template gives rise to *many* features, e.g.:

$$\begin{aligned} \text{score}(\mathbf{x}, \mathbf{y}) = & \dots \\ & + \theta[\text{“template7}=(\mathbf{P}, \mathbf{D}, \text{-ow})\text{”}] * \text{count}(\text{“template7}=(\mathbf{P}, \mathbf{D}, \text{-ow})\text{”}) \\ & + \theta[\text{“template7}=(\mathbf{D}, \mathbf{D}, \text{-xx})\text{”}] * \text{count}(\text{“template7}=(\mathbf{D}, \mathbf{D}, \text{-xx})\text{”}) \\ & + \dots \end{aligned}$$

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 (\mathbf{x}, \mathbf{y}) ?

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

E.g., template 7 might be $(\text{tag}(i-1), \text{tag}(i), \text{suffix2}(i+1))$.

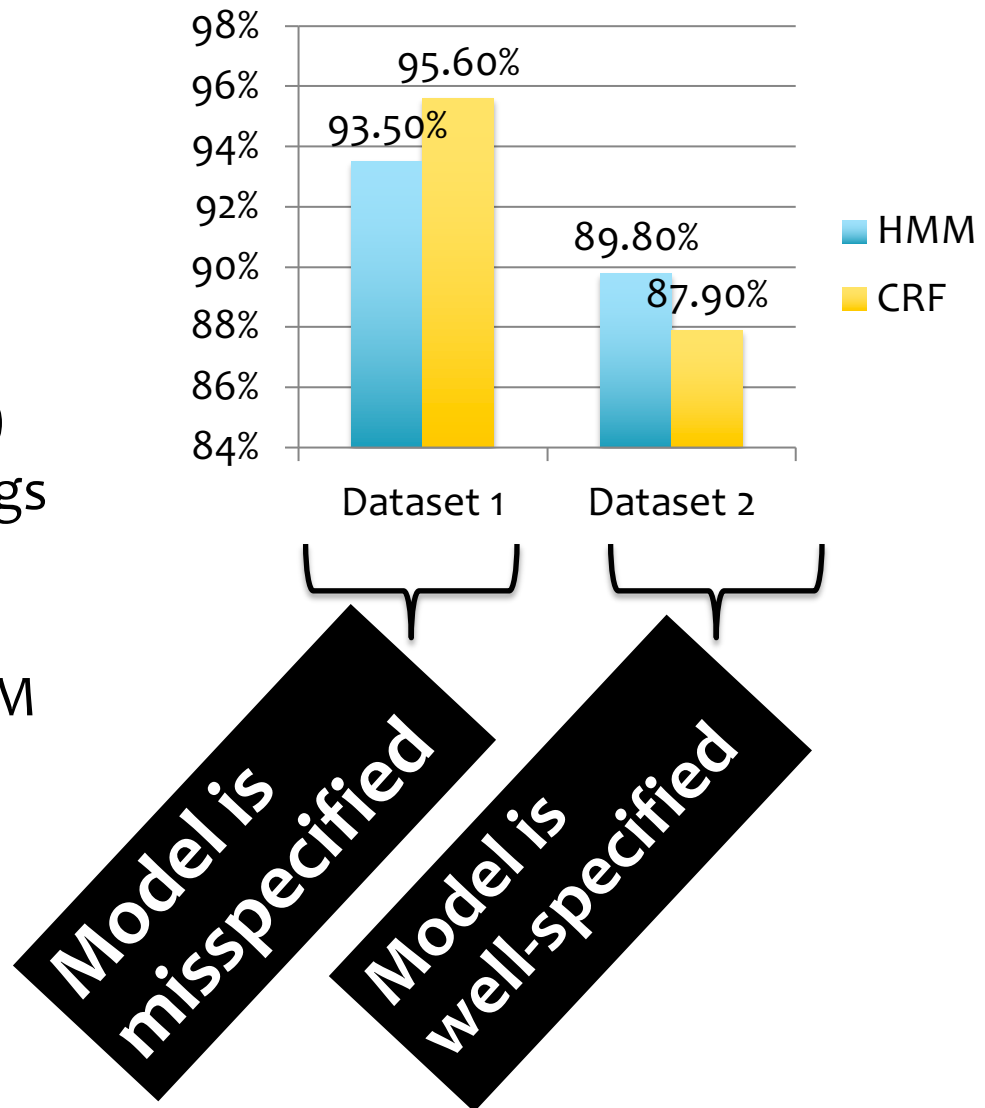
Note: Every template should mention at least some blue.

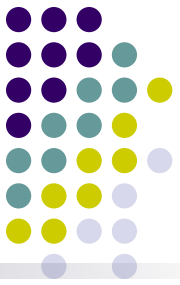
- Given an input \mathbf{x} , a feature that only looks at red will contribute the same weight to $\text{score}(\mathbf{x}, \mathbf{y}_1)$ and $\text{score}(\mathbf{x}, \mathbf{y}_2)$.
- So it can't help you choose between outputs $\mathbf{y}_1, \mathbf{y}_2$.

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

<i>model</i>	<i>error</i>	<i>oov error</i>
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 \geq CRF $>$ MEMM
- Using additional overlapping features: CRF⁺ $>$ MEMM⁺ \gg HMM

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

arg $p(\mathbf{y}|\mathbf{x})$

$$h_{\theta}(\mathbf{x}) = \operatorname{argmin}_{\hat{\mathbf{y}}} \mathbb{E}_{\mathbf{y} \sim p_{\theta}^{\tau}(\cdot|\mathbf{x})} [\ell(\hat{\mathbf{y}}, \mathbf{y})]$$

θ^{τ}

$$= \operatorname{argmin}_{\hat{\mathbf{y}}} \sum_{\mathbf{y}} p_{\theta}(\mathbf{y} | \mathbf{x}) \ell(\hat{\mathbf{y}}, \mathbf{y})$$

Minimum Bayes Risk Decoding

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

Consider some example loss functions:

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

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

(Handwritten red arrow points from \hat{y} to y)

The MBR decoder is:

$$\begin{aligned} h_{\theta}(x) &= \operatorname{argmin}_{\hat{y}} \sum_y p_{\theta}(y | x) (1 - \mathbb{I}(\hat{y}, y)) \\ &= \operatorname{argmax}_{\hat{y}} p_{\theta}(\hat{y} | x) \end{aligned}$$

(Handwritten red circles and arrows highlight the minimization over \hat{y} and the maximization over \hat{y} in the second line, and the probability term $p_{\theta}(y | x)$ in the first line.)

which is exactly the MAP inference problem!

Minimum Bayes Risk Decoding

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

Consider some example loss functions:

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

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

The MBR decoder is:

$$\hat{y}_i = h_{\theta}(\mathbf{x})_i = \operatorname{argmax}_{\hat{y}_i} p_{\theta}(\hat{y}_i | \mathbf{x})$$

This decomposes across variables and requires the variable marginals.

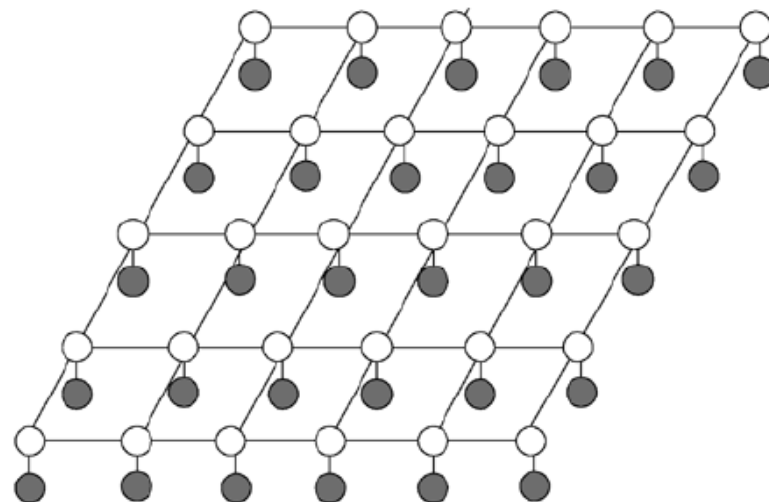
General CRFs, Hidden-state CRFs

2. CASE STUDY: IMAGE SEGMENTATION (COMPUTER VISION)

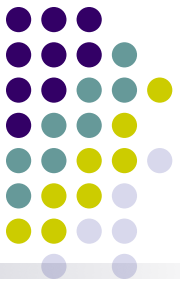
Other CRFs



- So far we have discussed only 1-dimensional chain CRFs
 - Inference and learning: exact
- We could also have CRFs for arbitrary graph structure
 - E.g: Grid CRFs
 - Inference and learning no longer tractable
 - Approximate techniques used
 - MCMC Sampling
 - Variational Inference
 - Loopy Belief Propagation
 - We will discuss these techniques soon



Applications of CRF in Vision



Stereo Matching

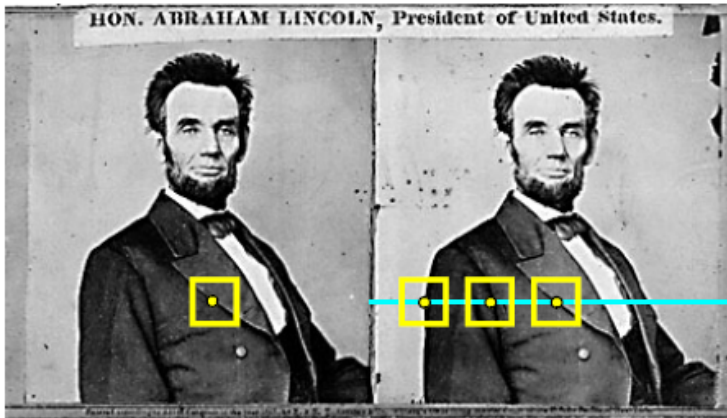


Image Restoration

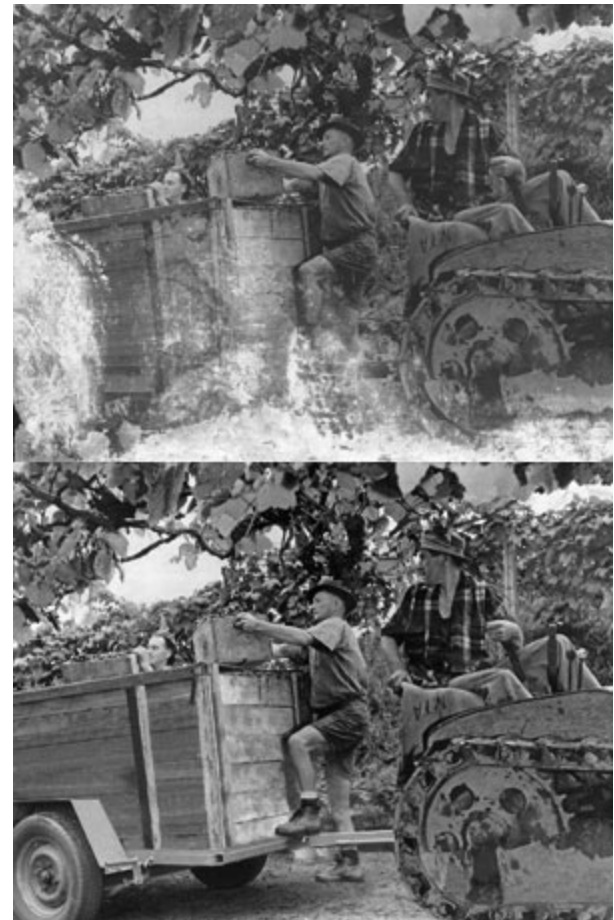


Image Segmentation



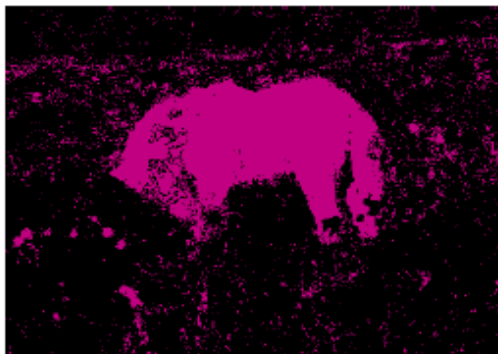
Application: Image Segmentation



$\phi_i(y_i, x) \in \mathbb{R}^{\approx 1000}$: local image features, e.g. bag-of-words
→ $\langle w_i, \phi_i(y_i, x) \rangle$: local classifier (like logistic-regression)
 $\phi_{i,j}(y_i, y_j) = \llbracket y_i = y_j \rrbracket \in \mathbb{R}^1$: test for same label
→ $\langle w_{ij}, \phi_{ij}(y_i, y_j) \rangle$: penalizer for label changes (if $w_{ij} > 0$)
combined: $\operatorname{argmax}_y p(y|x)$ is smoothed version of local cues



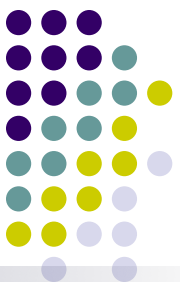
original



local classification



local + smoothness

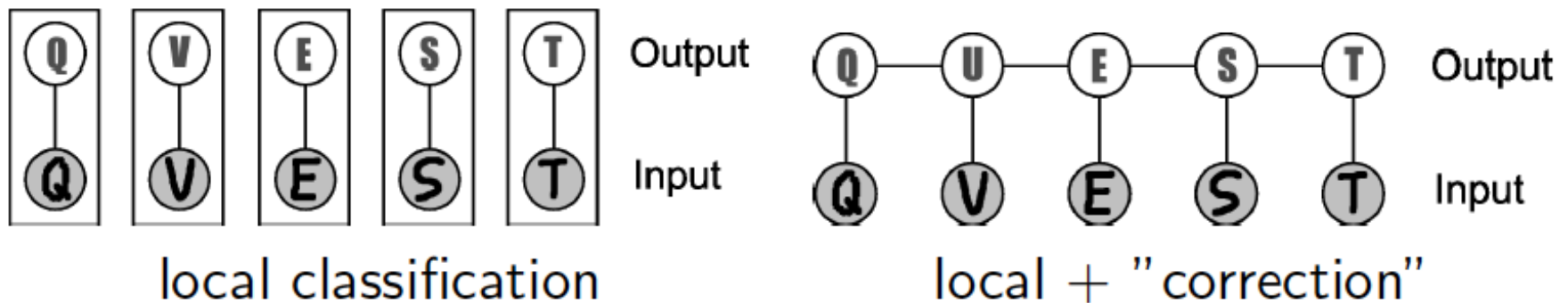


Application: Handwriting Recognition

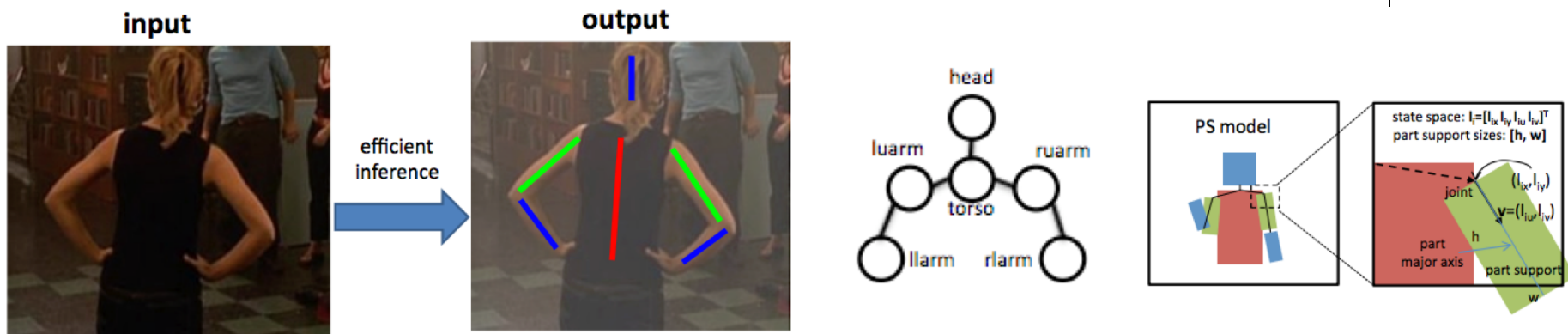
$\phi_i(y_i, x) \in \mathbb{R}^{\approx 1000}$: image representation (pixels, gradients)
 $\rightarrow \langle w_i, \phi_i(y_i, x) \rangle$: local classifier if x_i is letter y_i

$\phi_{i,j}(y_i, y_j) = e_{y_i} \otimes e_{y_j} \in \mathbb{R}^{26 \cdot 26}$: letter/letter indicator
 $\rightarrow \langle w_{ij}, \phi_{ij}(y_i, y_j) \rangle$: encourage/suppress letter combinations

combined: $\operatorname{argmax}_y p(y|x)$ is "corrected" version of local cues



Application: Pose Estimation

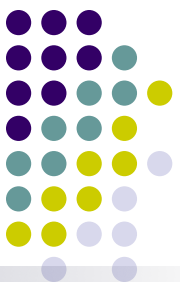


$$p(l|x) \propto \exp \left[\sum_{ij} \theta_{ij}^T \phi_{ij}(l_i, l_j, x) + \sum_i \theta_i^T \phi_i(l_i, x) \right] = e^{\theta^T \phi(l, x)}.$$

Penalizes unrealistic poses

Local classifier for each part

$\operatorname{argmax}_y p(y|x)$ is cleaned up version of local prediction



Feature Functions for CRF in Vision

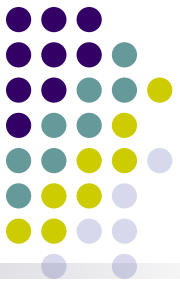
$\phi_i(y_i, x)$: local representation, high-dimensional
 $\rightarrow \langle w_i, \phi_i(y_i, x) \rangle$: local classifier

$\phi_{i,j}(y_i, y_j)$: prior knowledge, low-dimensional
 $\rightarrow \langle w_{ij}, \phi_{ij}(y_i, y_j) \rangle$: penalize outliers

learning adjusts parameters:

- ▶ unary w_i : learn local classifiers and their importance
- ▶ binary w_{ij} : learn importance of smoothing/penalization

$\operatorname{argmax}_y p(y|x)$ is cleaned up version of local prediction



Case Study: Image Segmentation

- Image segmentation (FG/BG) by modeling of interactions btw RVs
 - Images are noisy.
 - Objects occupy continuous regions in an image.

[Nowozin, Lampert 2012]



Input image



Pixel-wise separate optimal labeling



Locally-consistent joint optimal labeling

$$Y^* = \arg \max_{y \in \{0,1\}^n} \left[\overbrace{\sum_{i \in S} V_i(y_i, X)}^{\text{Unary Term}} + \overbrace{\sum_{i \in S} \sum_{j \in N_i} V_{i,j}(y_i, y_j)}^{\text{Pairwise Term}} \right].$$

Y : labels

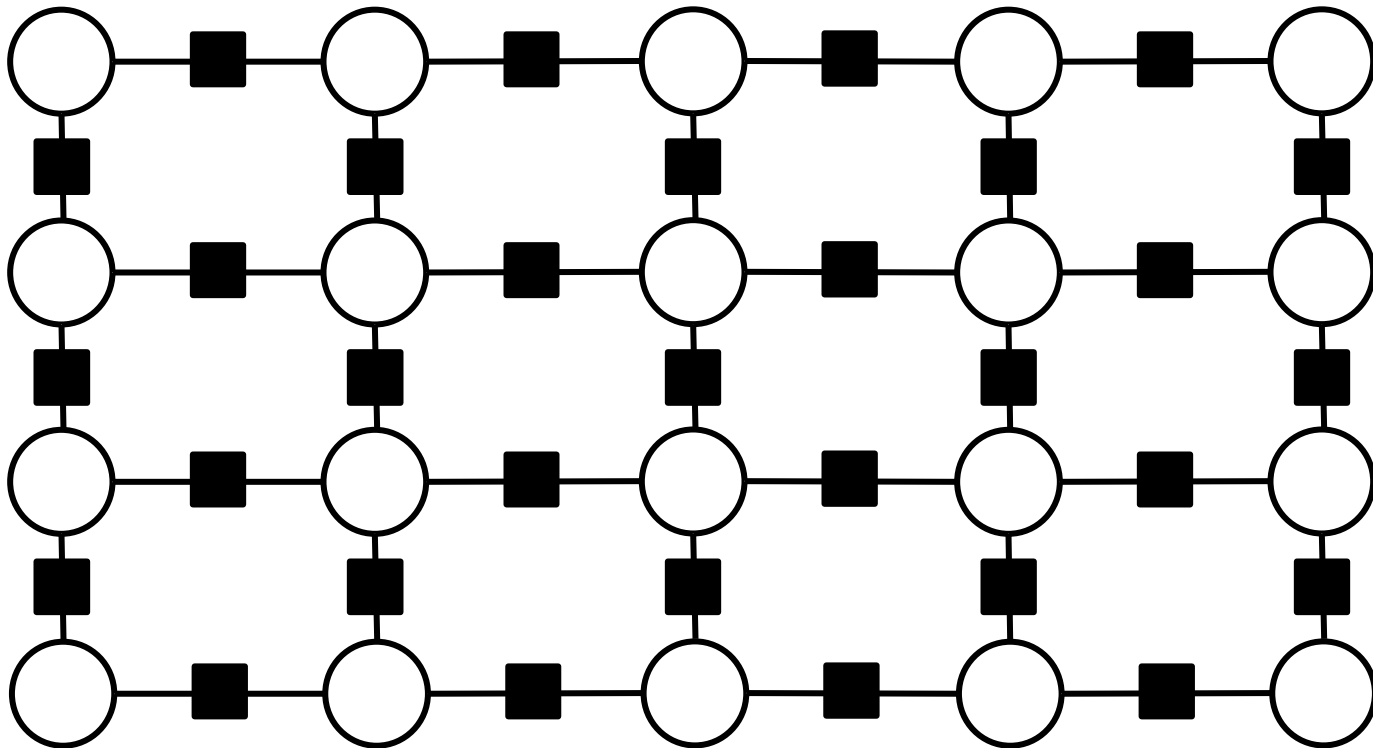
X : data (features)

S : pixels

N_i : neighbors of pixel i

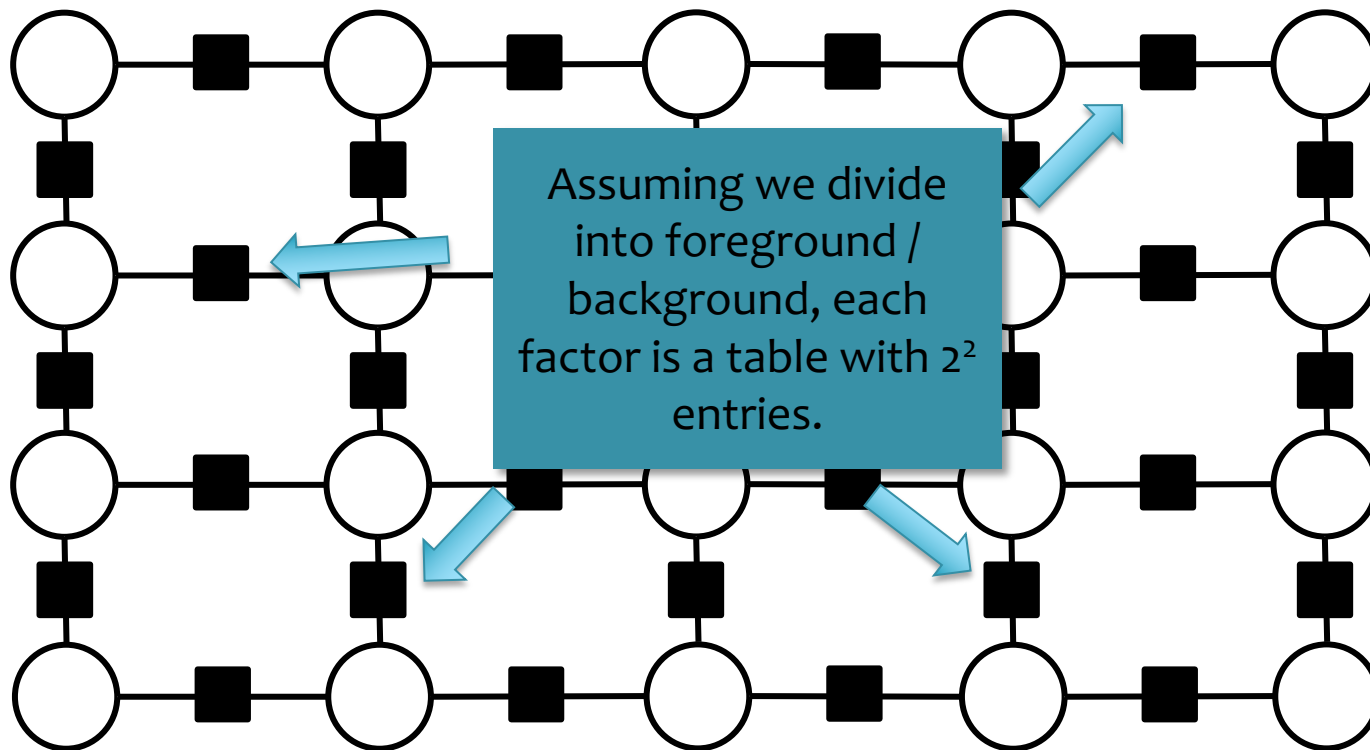
Grid CRF

- Suppose we want to image segmentation using a grid model



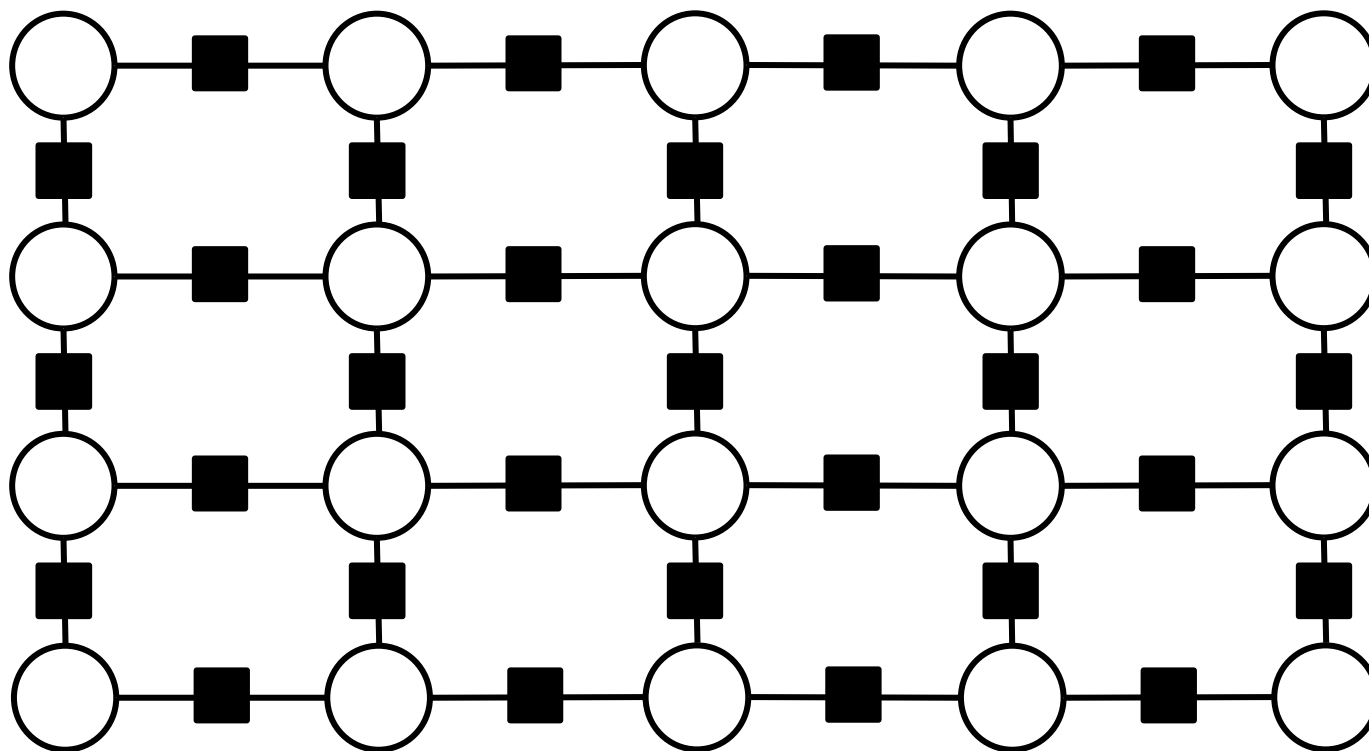
Grid CRF

- Suppose we want to image segmentation using a grid model



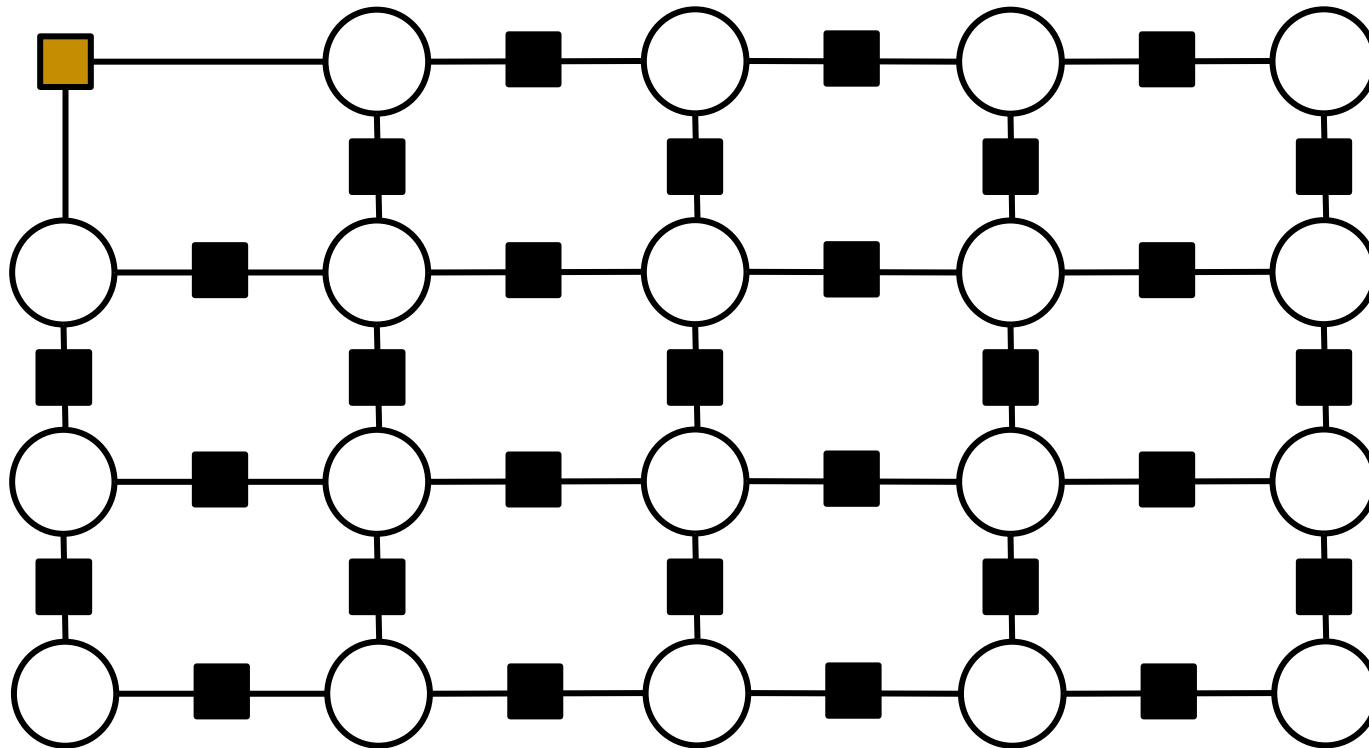
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?



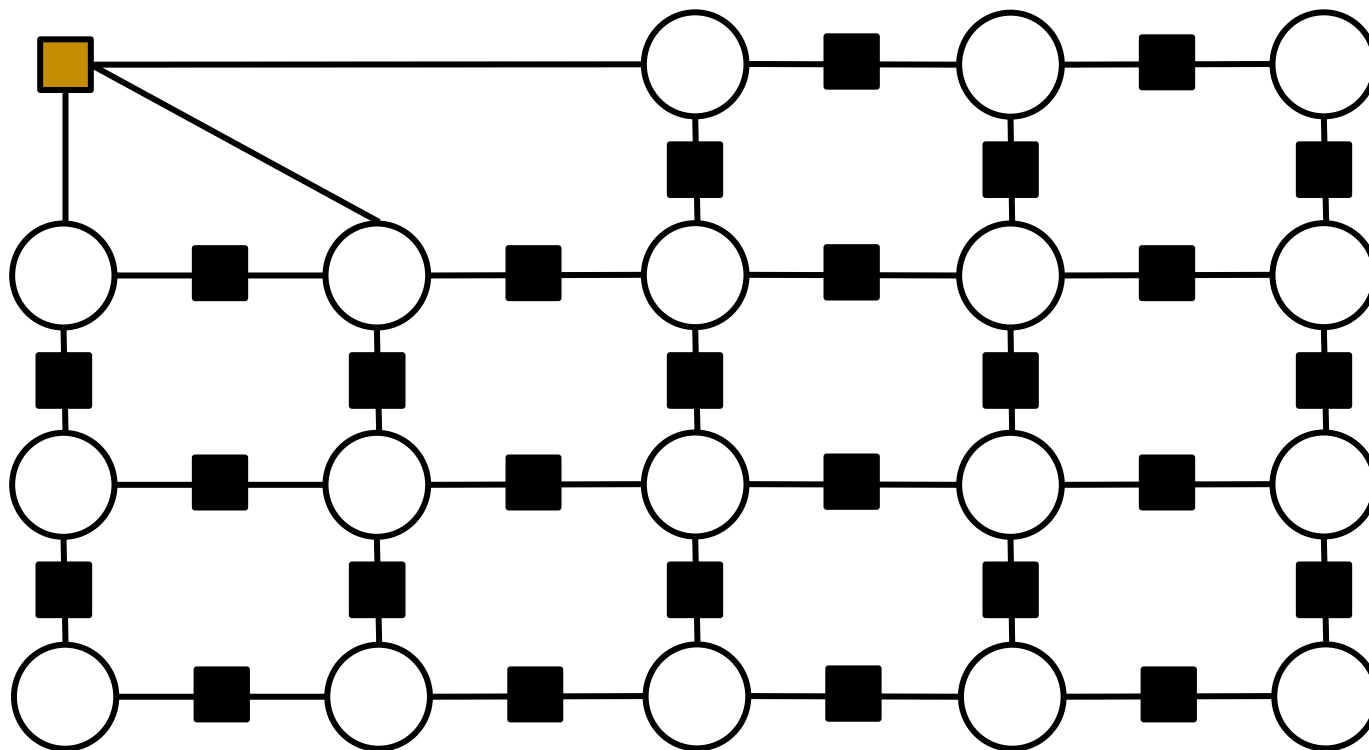
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?



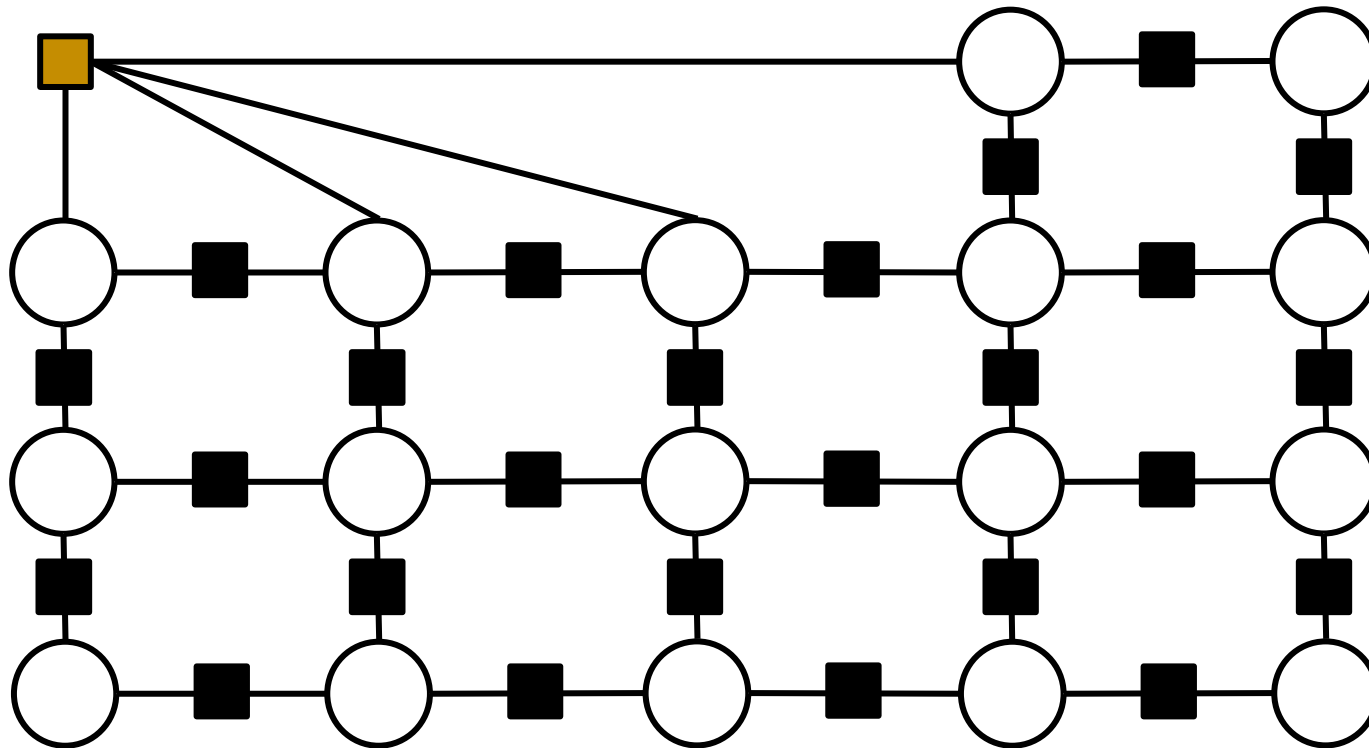
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?



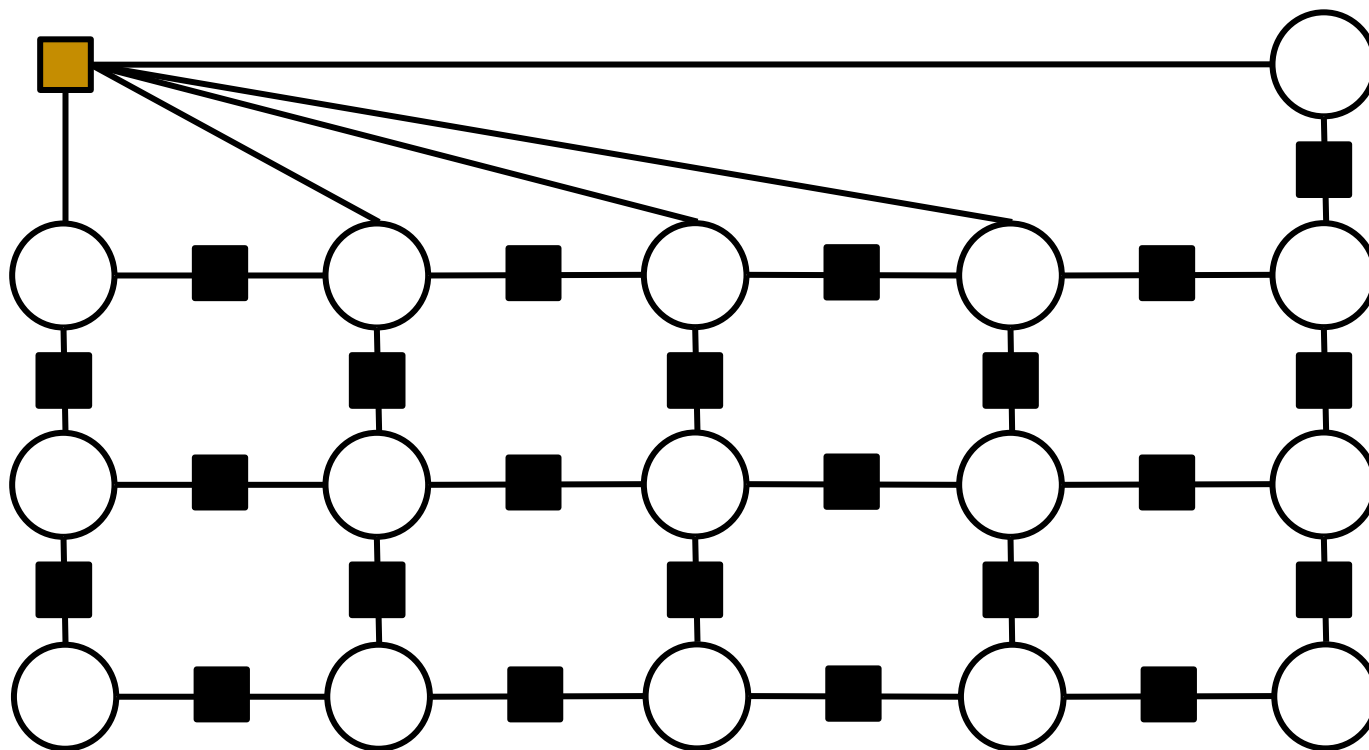
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?



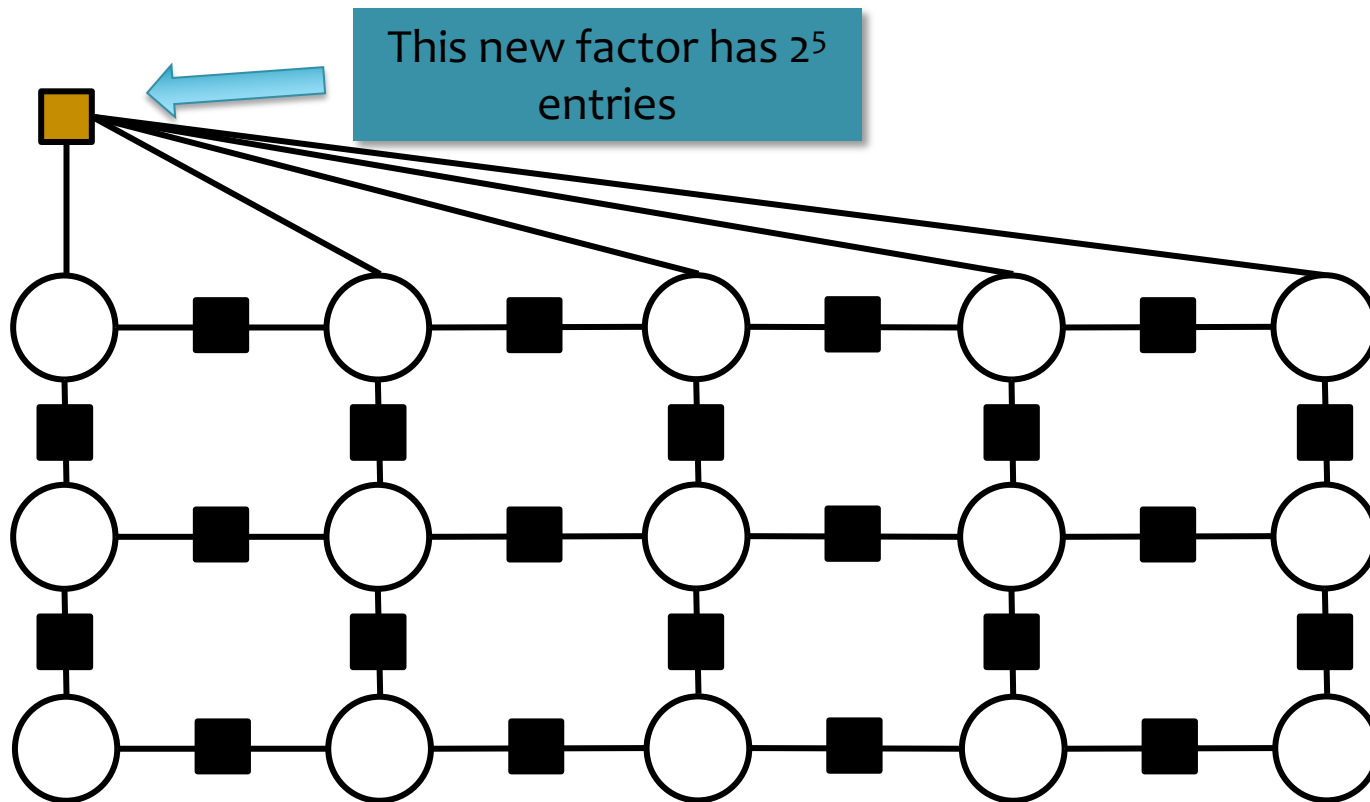
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?



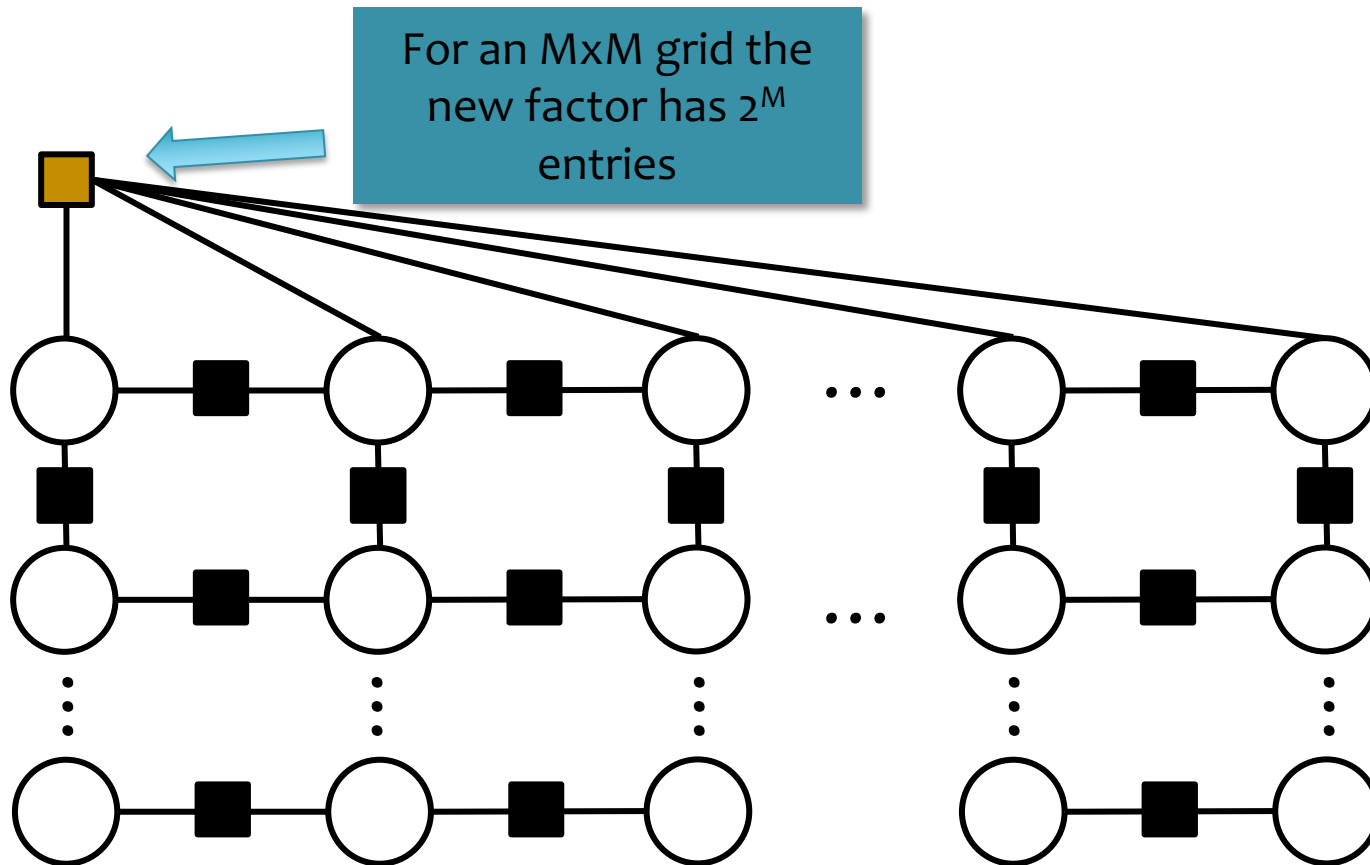
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?



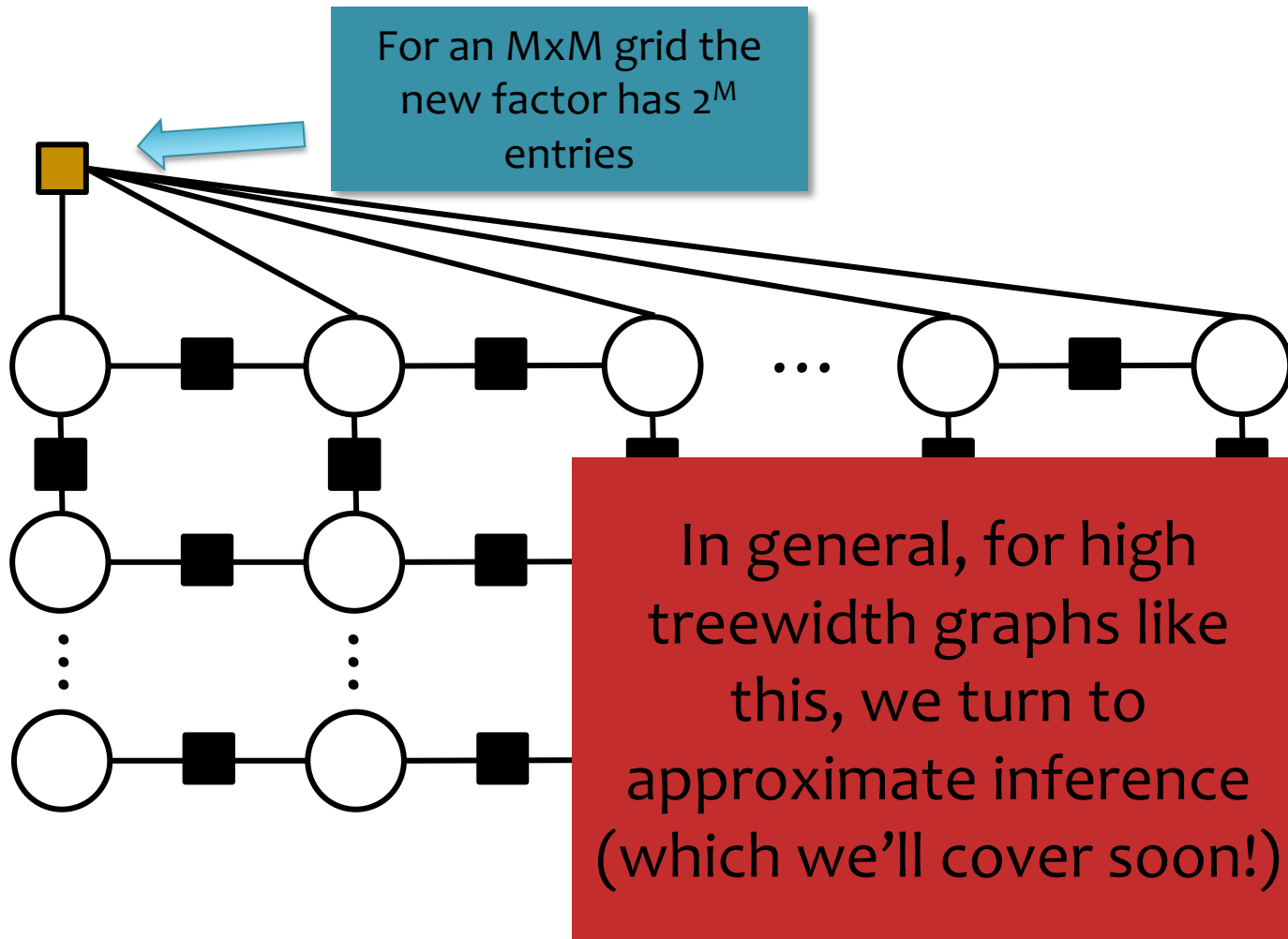
Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?



Grid CRF

- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?



Case Study: Object Recognition

Data consists of images x and labels y .



pigeon



rhinoceros



leopard



llama

Case Study: Object Recognition

Data consists of images x and labels y .

- Preprocess data into “patches”
- Posit a latent labeling z describing the object’s parts (e.g. head, leg, tail, torso, grass)
- Define graphical model with these latent variables in mind
- z is not observed at train or test time

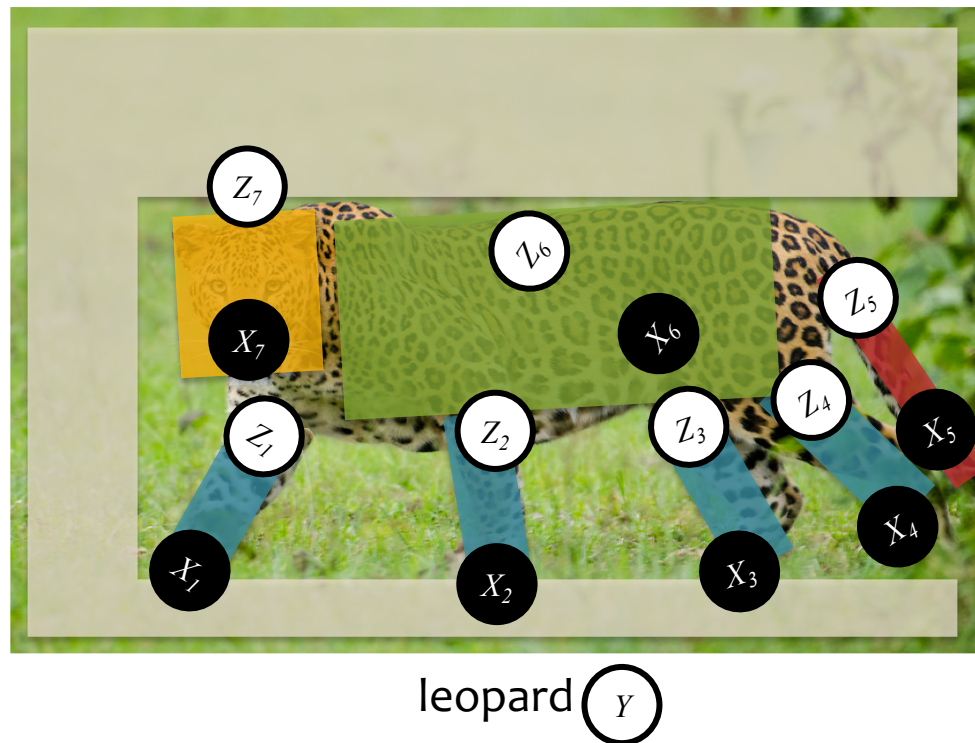


leopard

Case Study: Object Recognition

Data consists of images x and labels y .

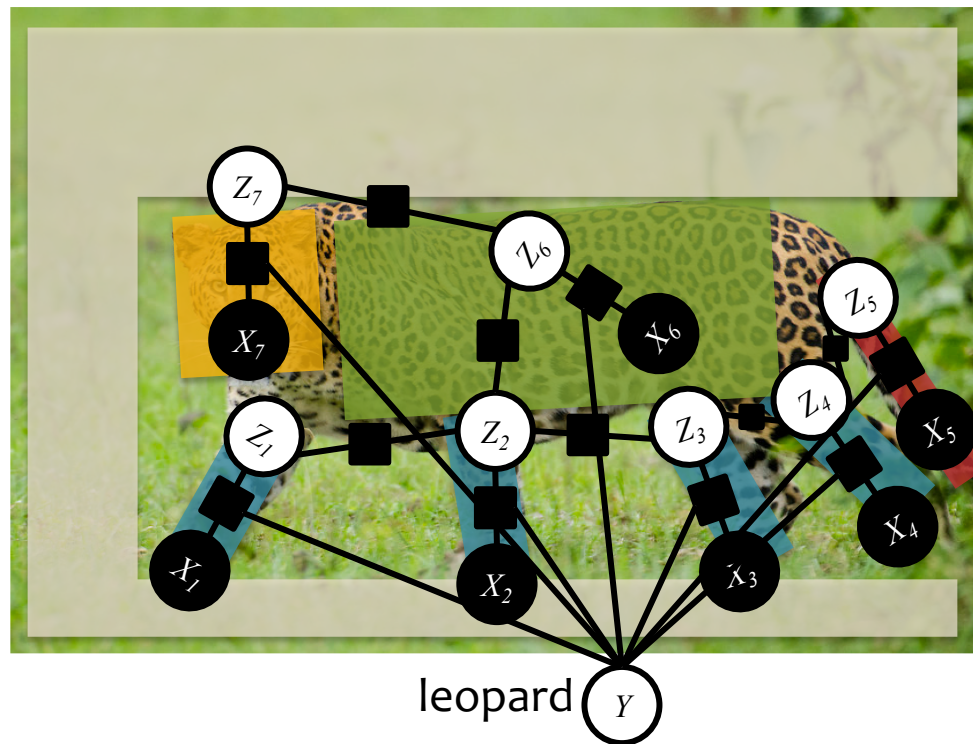
- Preprocess data into “patches”
- Posit a latent labeling z describing the object’s parts (e.g. head, leg, tail, torso, grass)
- Define graphical model with these latent variables in mind
- z is not observed at train or test time



Case Study: Object Recognition

Data consists of images x and labels y .

- Preprocess data into “patches”
- Posit a latent labeling z describing the object’s parts (e.g. head, leg, tail, torso, grass)
- Define graphical model with these latent variables in mind
- z is not observed at train or test time

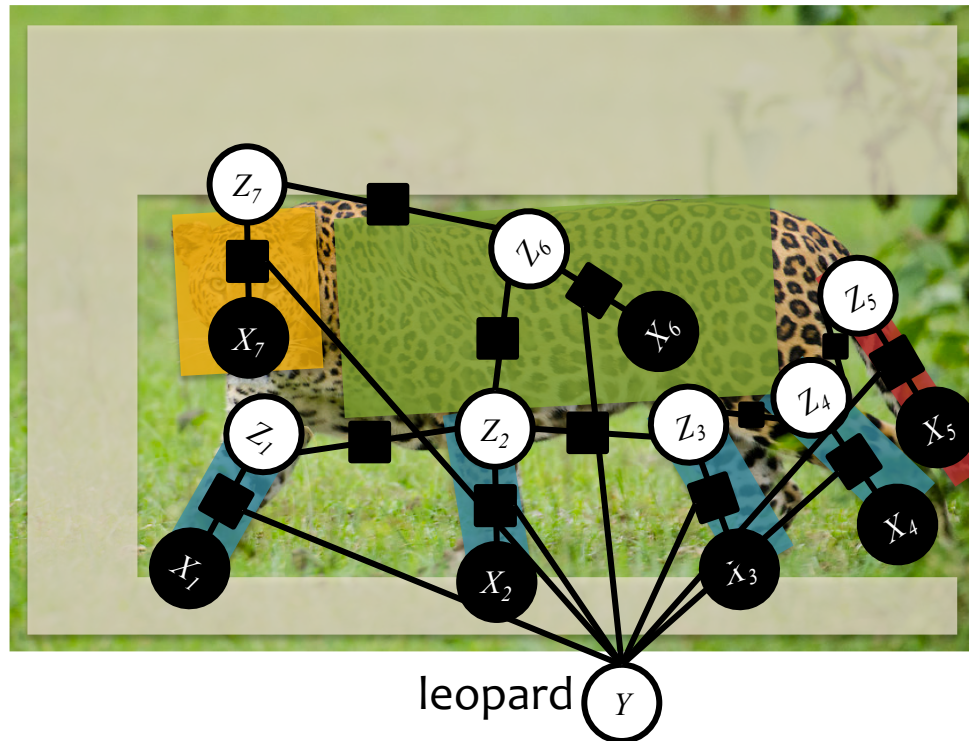


Hidden-state CRFs

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

Joint model: $p_{\theta}(\mathbf{y}, \mathbf{z} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x}, \theta)} \prod_{\alpha} \psi_{\alpha}(\mathbf{y}_{\alpha}, \mathbf{z}_{\alpha}, \mathbf{x})$

Marginalized model: $p_{\theta}(\mathbf{y} \mid \mathbf{x}) = \sum_{\mathbf{z}} p_{\theta}(\mathbf{y}, \mathbf{z} \mid \mathbf{x})$



Hidden-state CRFs

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

Joint model: $p_{\theta}(\mathbf{y}, \mathbf{z} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x}, \theta)} \prod_{\alpha} \psi_{\alpha}(\mathbf{y}_{\alpha}, \mathbf{z}_{\alpha}, \mathbf{x})$

Marginalized model: $p_{\theta}(\mathbf{y} \mid \mathbf{x}) = \sum_{\mathbf{z}} p_{\theta}(\mathbf{y}, \mathbf{z} \mid \mathbf{x})$

We can train using gradient based methods:
(the values \mathbf{x} are omitted below for clarity)

$$\begin{aligned} \frac{d\ell(\theta \mid \mathcal{D})}{d\theta} &= \sum_{n=1}^N \left(\mathbb{E}_{\mathbf{z} \sim p_{\theta}(\cdot \mid \mathbf{y}^{(n)})} [f_j(\mathbf{y}^{(n)}, \mathbf{z})] - \mathbb{E}_{\mathbf{y}, \mathbf{z} \sim p_{\theta}(\cdot, \cdot)} [f_j(\mathbf{y}, \mathbf{z})] \right) \\ &= \sum_{n=1}^N \sum_{\alpha} \left(\underbrace{\sum_{\mathbf{z}_{\alpha}} p_{\theta}(\mathbf{z}_{\alpha} \mid \mathbf{y}^{(n)}) f_{\alpha,j}(\mathbf{y}_{\alpha}^{(n)}, \mathbf{z}_{\alpha})}_{\text{Inference on clamped factor graph}} - \sum_{\mathbf{y}_{\alpha}, \mathbf{z}_{\alpha}} \underbrace{p_{\theta}(\mathbf{y}_{\alpha}, \mathbf{z}_{\alpha}) f_{\alpha,j}(\mathbf{y}_{\alpha}, \mathbf{z}_{\alpha})}_{\text{Inference on full factor graph}} \right) \end{aligned}$$

Learning and Inference Summary

	Learning	Marginal Inference	MAP Inference
HMM	Just counting	Forward-backward	Viterbi
MEMM	Gradient based – decomposes and doesn't require inference (GLIM)	Forward-backward	Viterbi
Linear-chain CRF	Gradient based – doesn't decompose because of $Z(\mathbf{x})$ and requires marginal inference	Forward-backward	Viterbi
General CRF	Gradient based – doesn't decompose because of $Z(\mathbf{x})$ and requires (approximate) marginal inference	(approximate methods)	(approximate methods)
HCRF	Gradient based – same as General CRF	(approximate methods)	(approximate methods)

Summary

- HMM:
 - Pro: Easy to train
 - Con: Misses out on rich features of the observations
- MEMM:
 - Pro: Fast to train and supports rich features
 - Con: Suffers (like the HMM) from the label bias problem
- Linear-chain CRF:
 - Pro: Defeats the label bias problem with support for rich features
 - Con: Slower to train
- MBR Decoding:
 - the principled way to account for a loss function when decoding from a probabilistic model
- Generative vs. Discriminative:
 - gen. is better if the model is well-specified
 - disc. is better if the model is misspecified
- General CRFs:
 - Exact inference won't suffice for high treewidth graphs
 - More general topologies can capture intuitions about variable dependencies
- HCRF:
 - Training looks very much like CRF training
 - Incorporation of hidden variables can model domain specific knowledge