Discrete Sequential Models
+
General CRF

Kayhan Batmanghelich

Slides Credit:

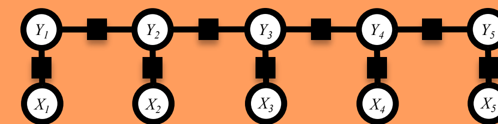
Matt Gormley (2016)

1. Data

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

2. Model

$$p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C)$$



3. Objective

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

5. Inference

1. Marginal Inference

$$p(\boldsymbol{x}_C) = \sum_{\boldsymbol{x}' : \boldsymbol{x}'_C = \boldsymbol{x}_C} p(\boldsymbol{x}' \mid \boldsymbol{\theta})$$

2. Partition Function

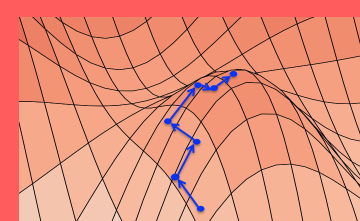
$$Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C)$$

3. MAP Inference

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmax}}\; p(\boldsymbol{x} \mid \boldsymbol{\theta})$$

4. Learning

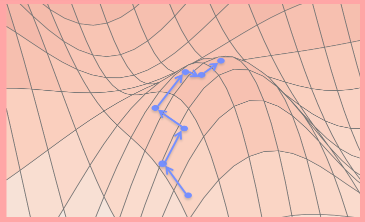
$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\; \ell(\boldsymbol{\theta}; \mathcal{D})$$



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



Outline

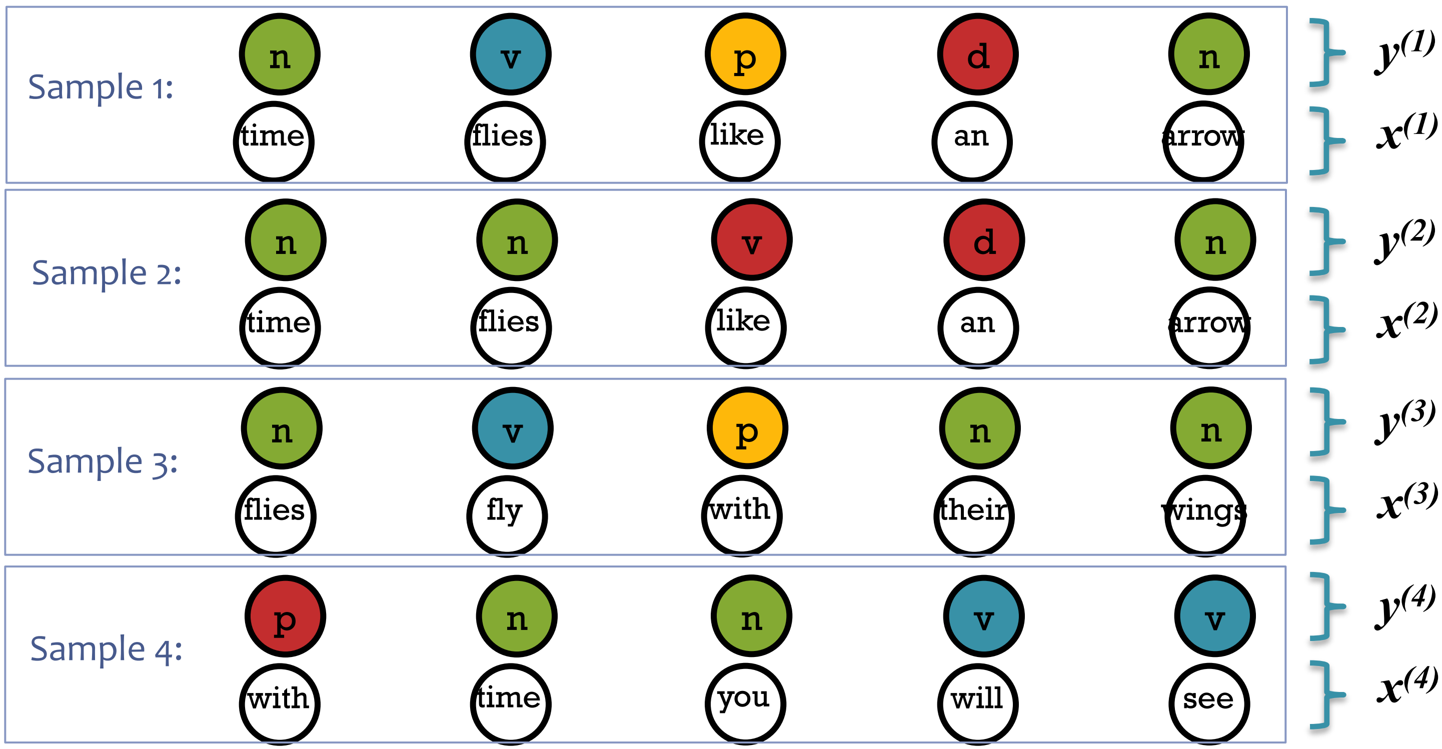
1. Case Study: Supervised Part-of-speech tagging (NLP)
   – Hidden Markov Model (HMM)
   – Maximum-Entropy Markov Model (MEMM)
   – Linear-chain CRF
   – *Digression*: Minimum Bayes Risk (MBR) Decoding
   – *Digression*: Generative vs. Discriminative
2. Case Study: Image Segmentation (Computer Vision)
   – 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: $\mathcal{D} = \{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\}_{n=1}^{N}$



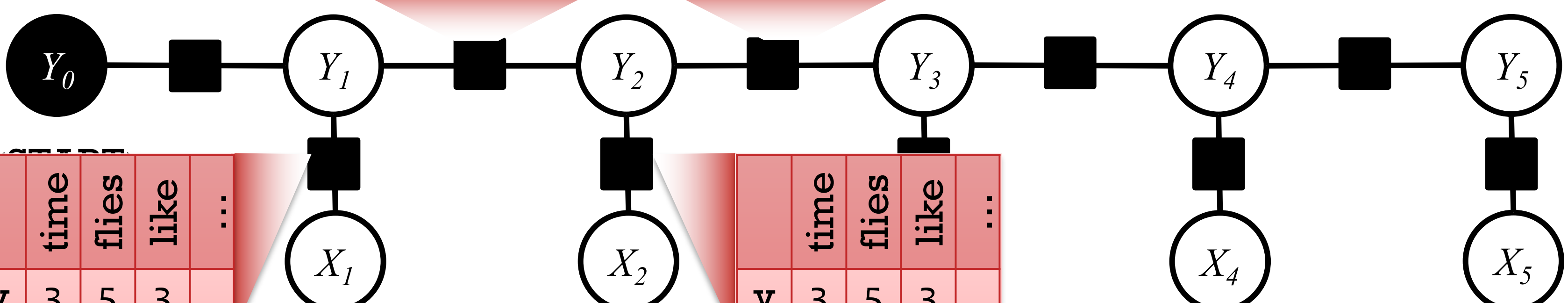
Factors have local opinions (≥ 0)

Each black box looks at *some* of the tags Y_i and words X_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.



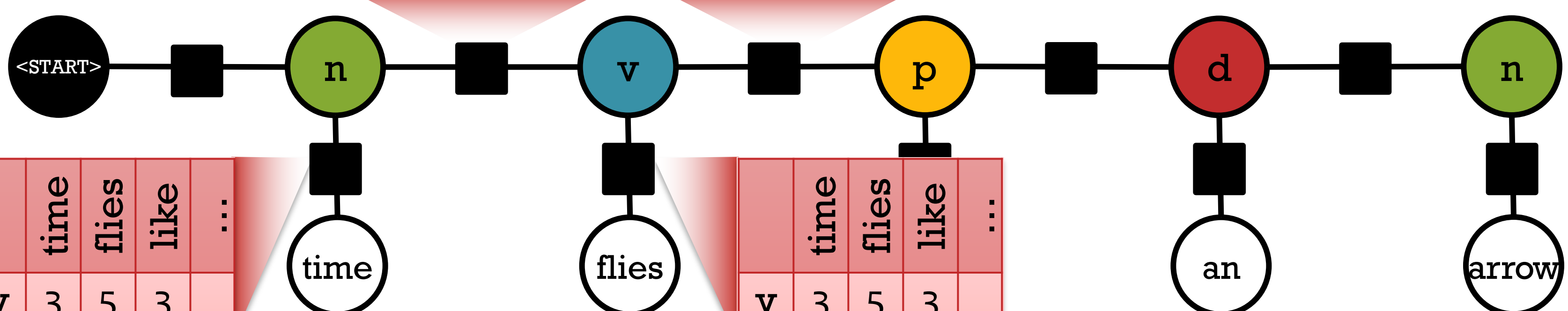
	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

	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 Y_i and words X_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 Y_i and words X_i

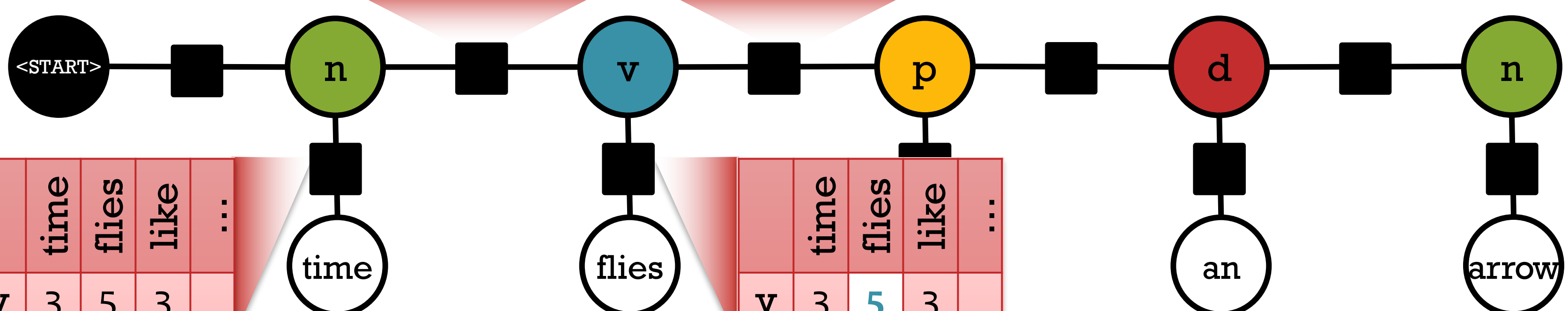
$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) = \frac{1}{Z}(4 * 8 * 5 * 3 * \ldots)$$

	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	

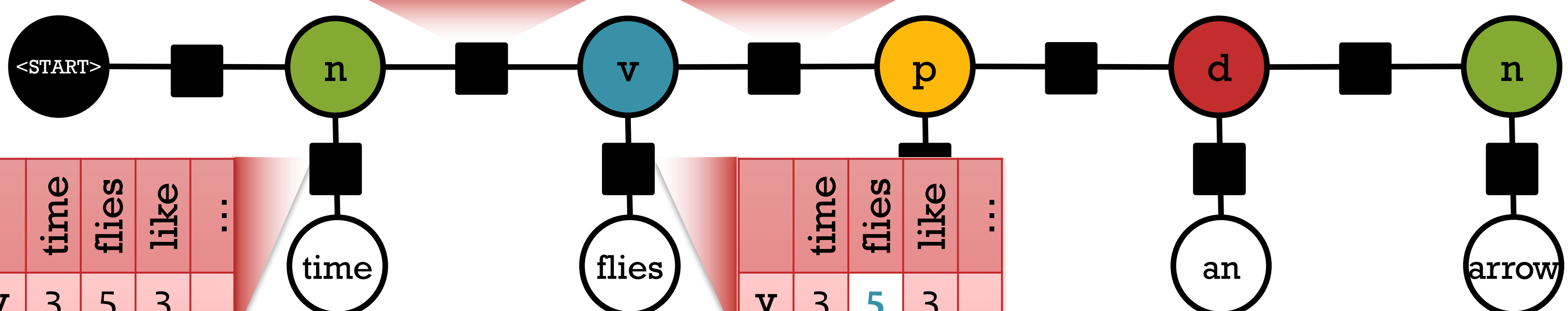
Markov Random Field (MRF)

Joint distribution over tags Y_i and words X_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



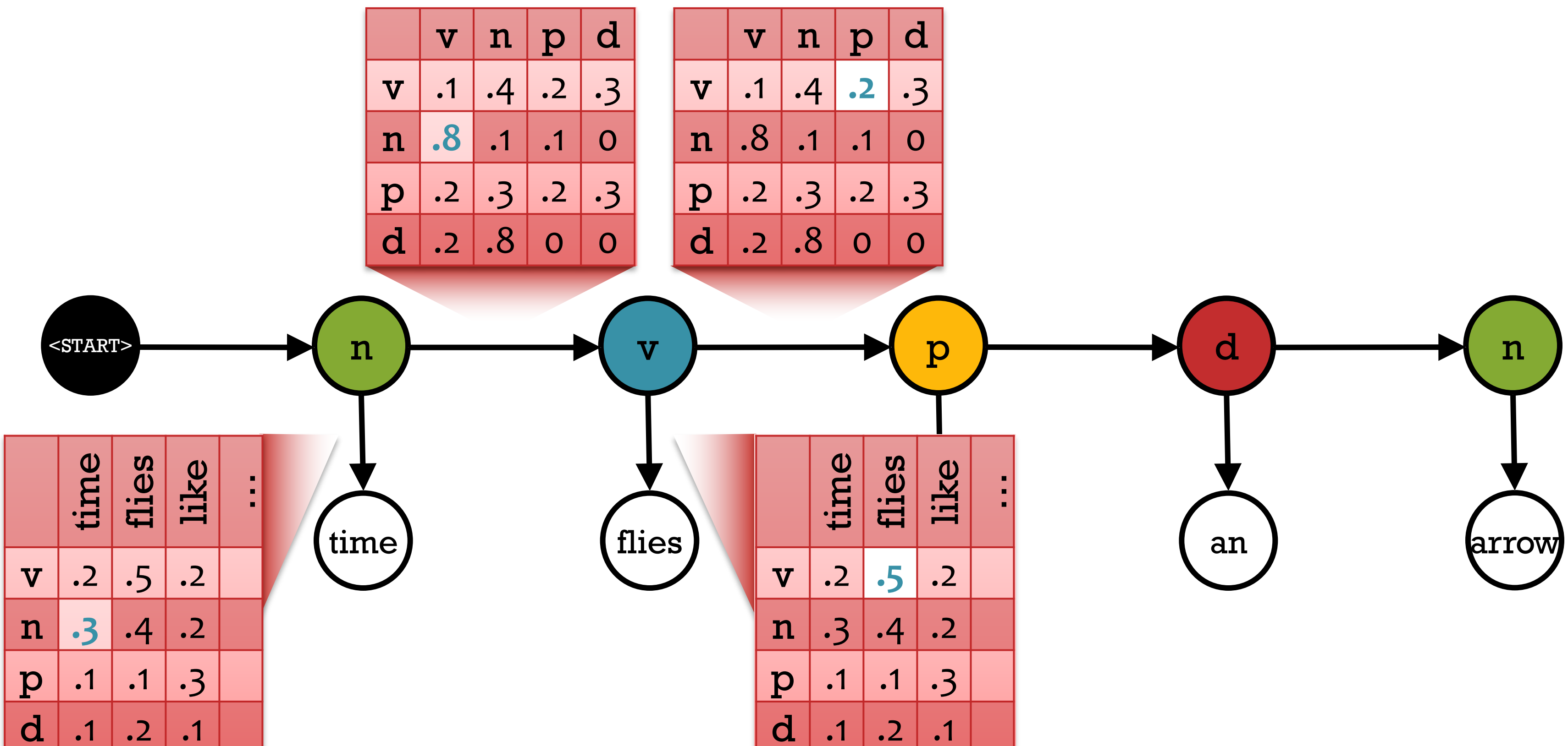
	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

	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

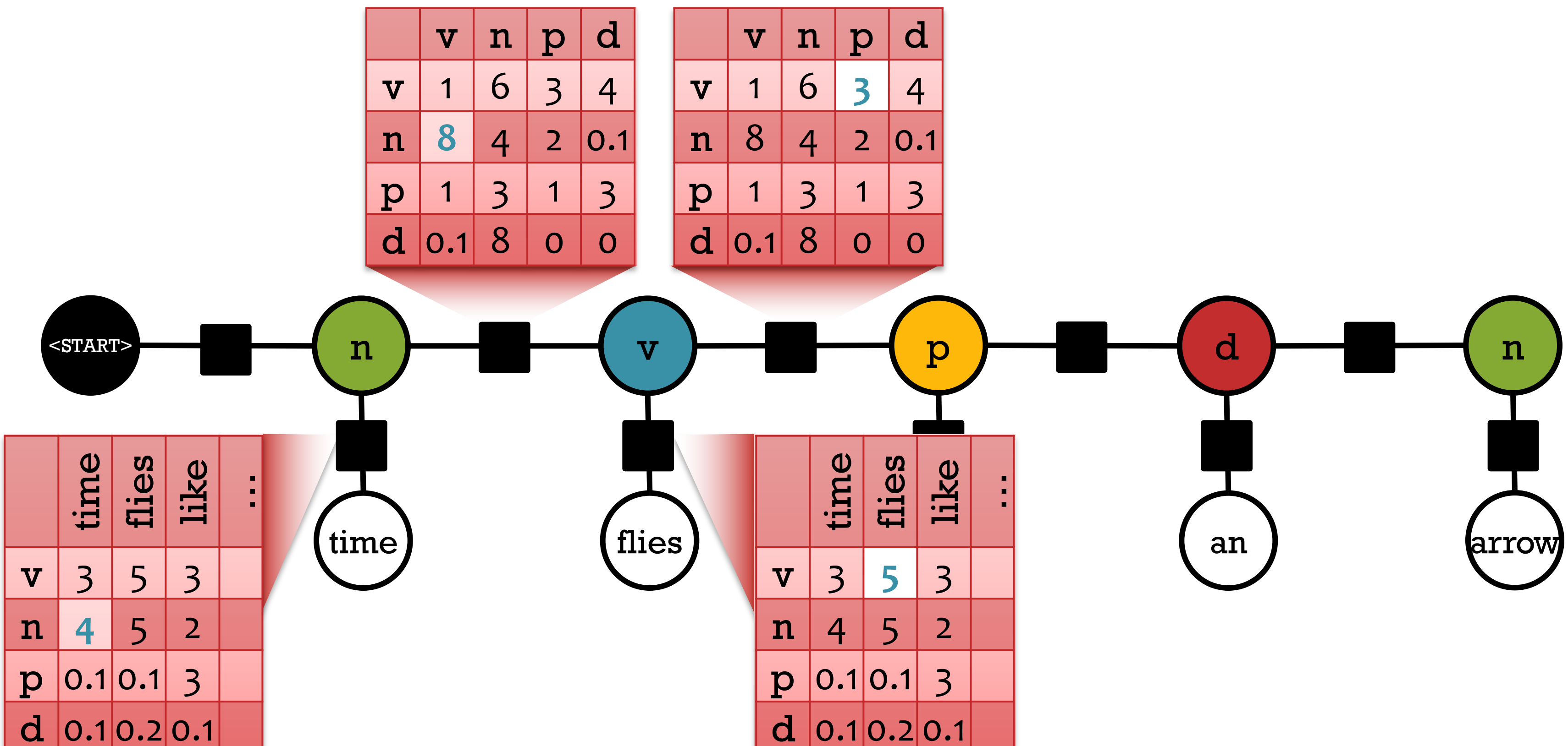
	time	flies	like	…
v	.2	.5	.2	
n	.3	.4	.2	
p	.1	.1	.3	
d	.1	.2	.1	

	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 Y_i and words X_i

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



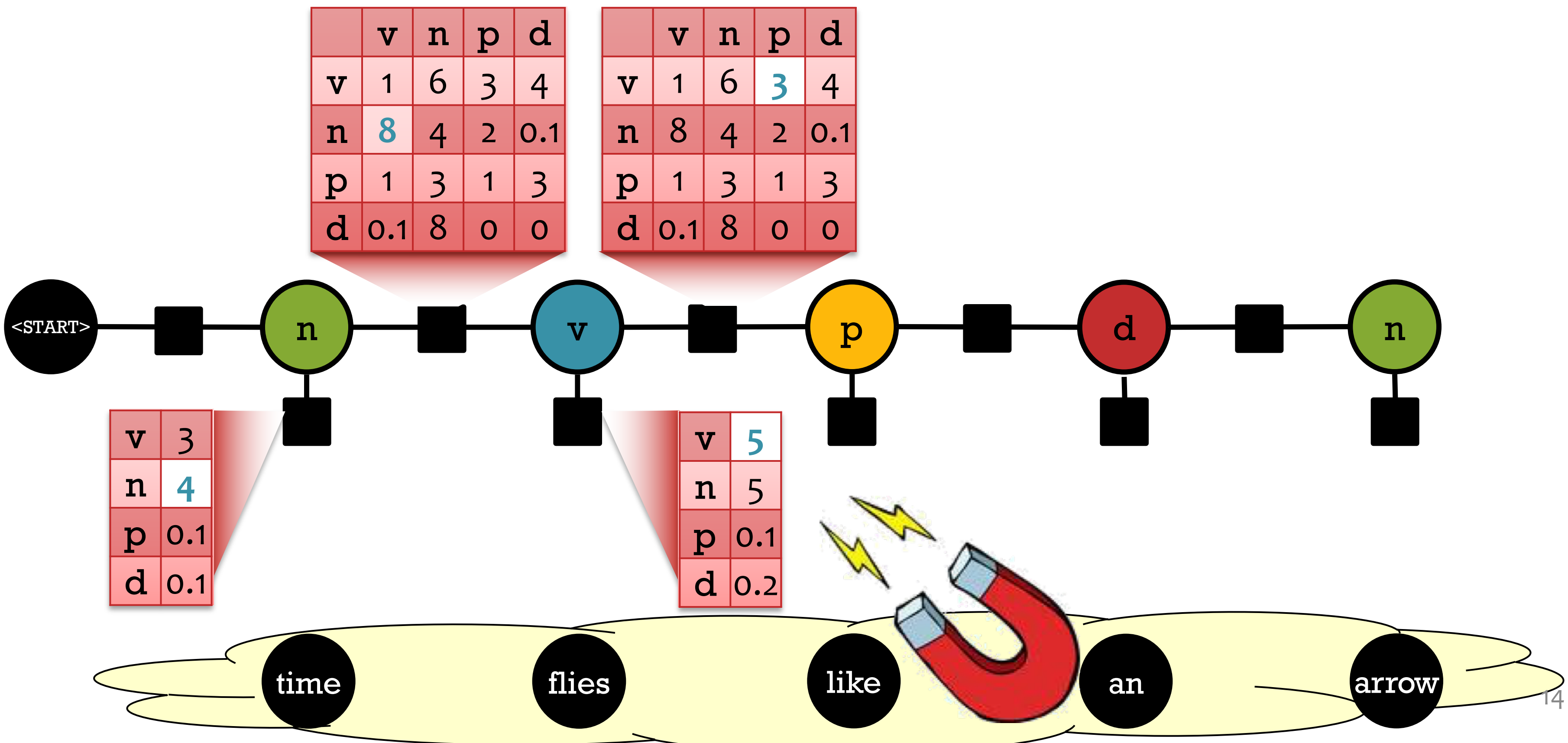
	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	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 Y_i given words x_i.
The factors and Z are now specific to the sentence $\boldsymbol{x}$.

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



	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)

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

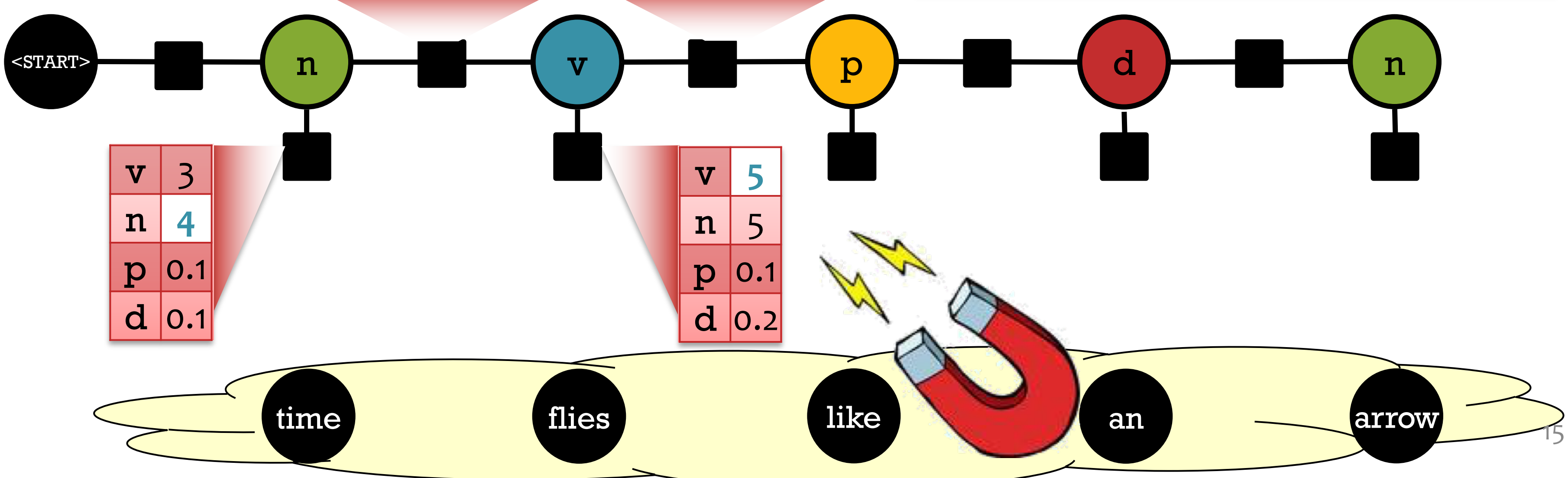
$$p(\text{n, v, p, d, n} \mid \text{time, flies, like, an, arrow}) = \frac{1}{Z}(4 * 8 * 5 * 3 * \ldots)$$

	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

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 $X_i = x_i$



v	3
n	4
p	0.1
d	0.1

v	5
n	5
p	0.1
d	0.2

Forward-Backward Algorithm

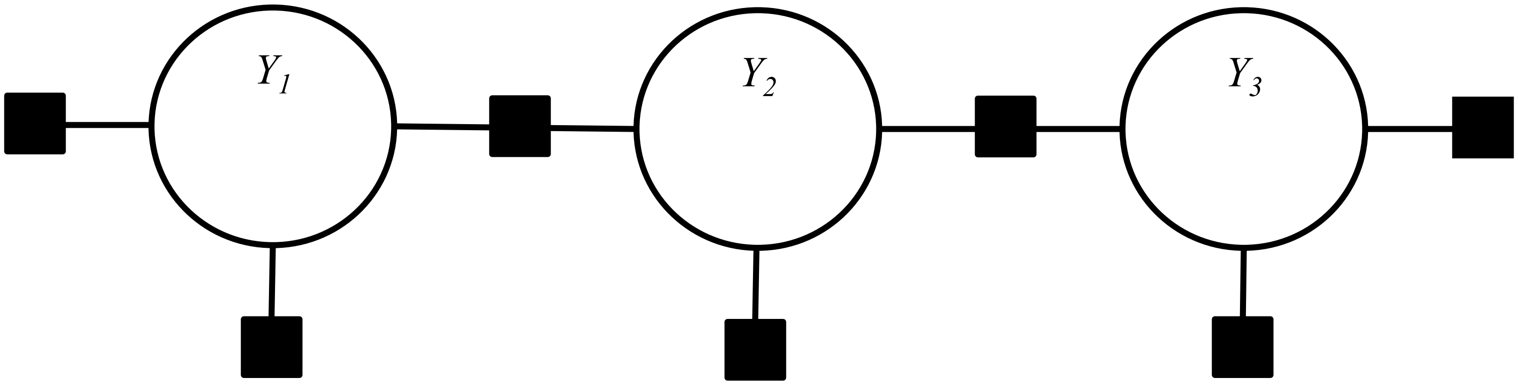
- Sum-product BP on an HMM is called the **forward-backward algorithm**
- Max-product BP on an HMM is called the **Viterbi algorithm**

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

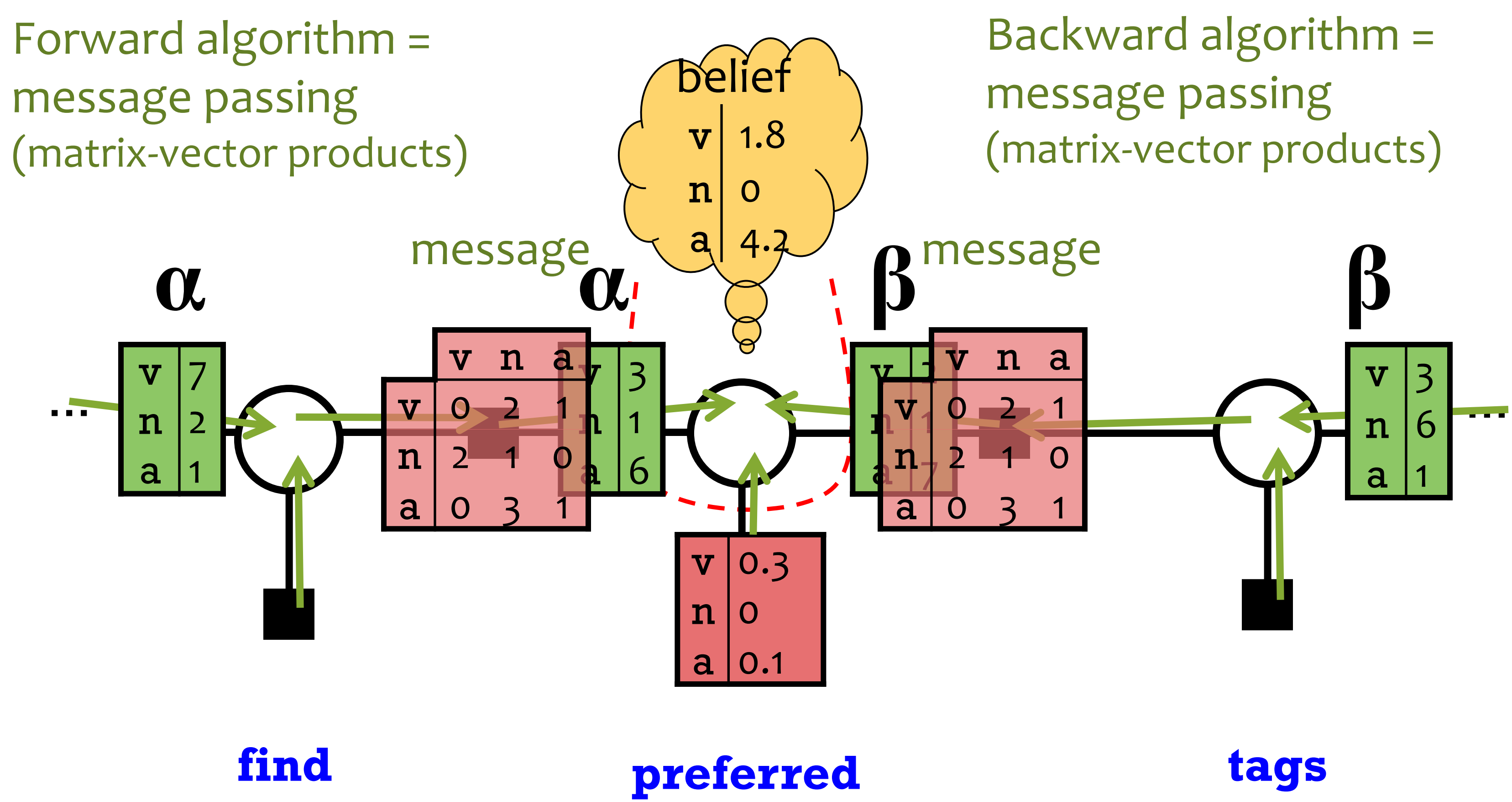
CRF Tagging Model



find | preferred | tags

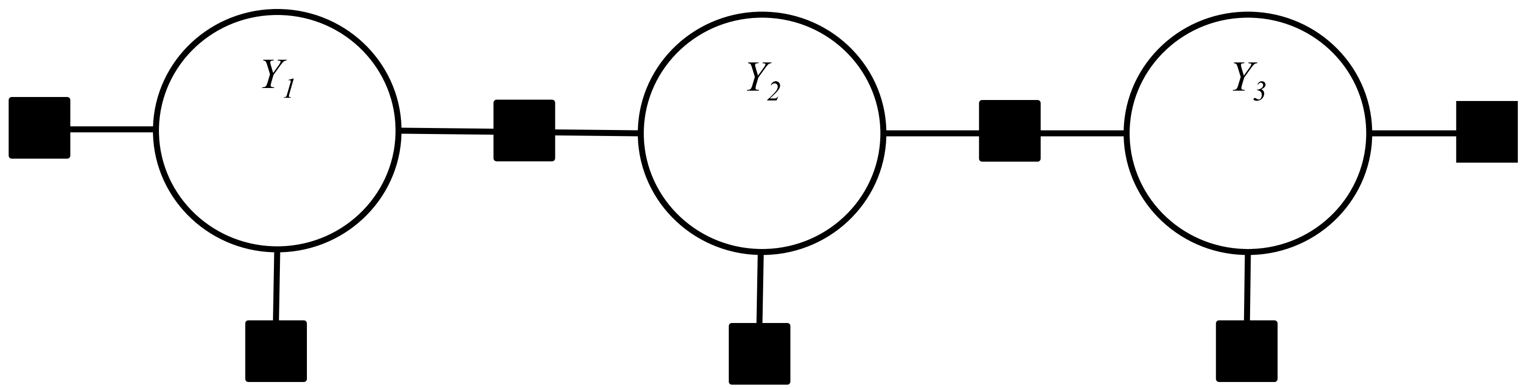
Could be verb or noun | *Could be adjective or verb* | *Could be noun or verb*

CRF Tagging by Belief Propagation



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

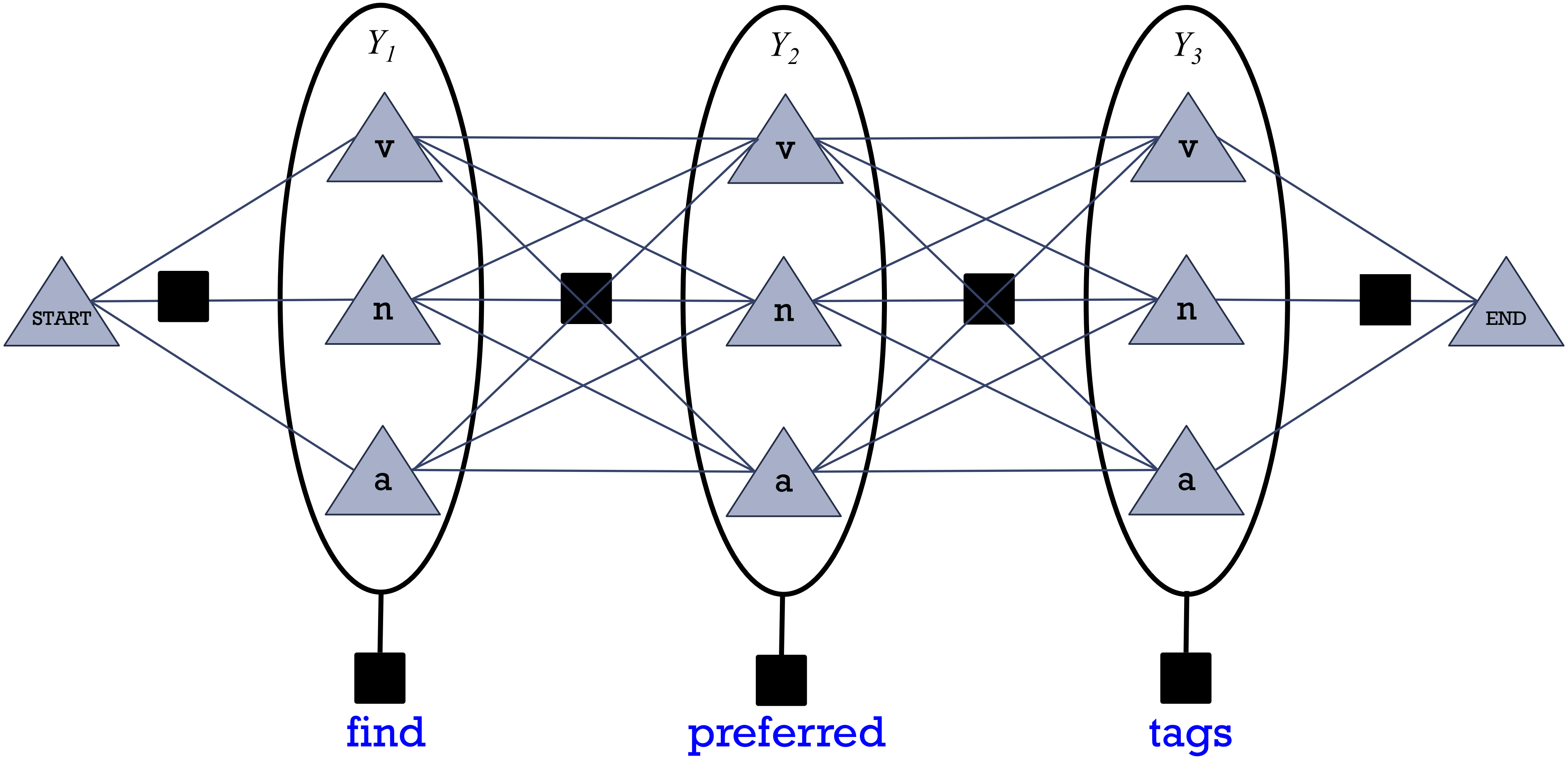
So Let's Review Forward-Backward …



find preferred tags

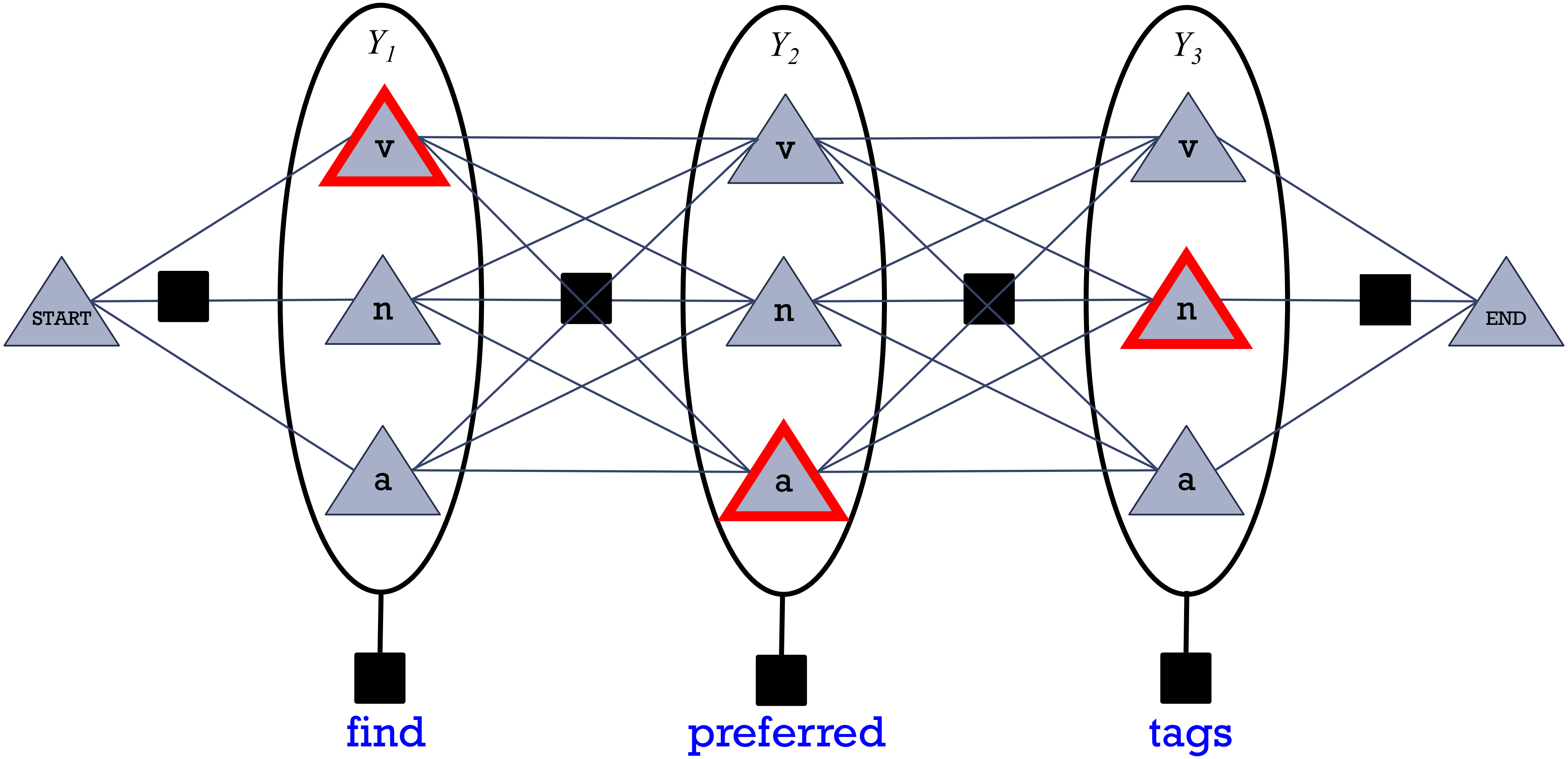
Could be verb or noun *Could be adjective or verb* *Could be noun or verb*

So Let's Review Forward-Backward …



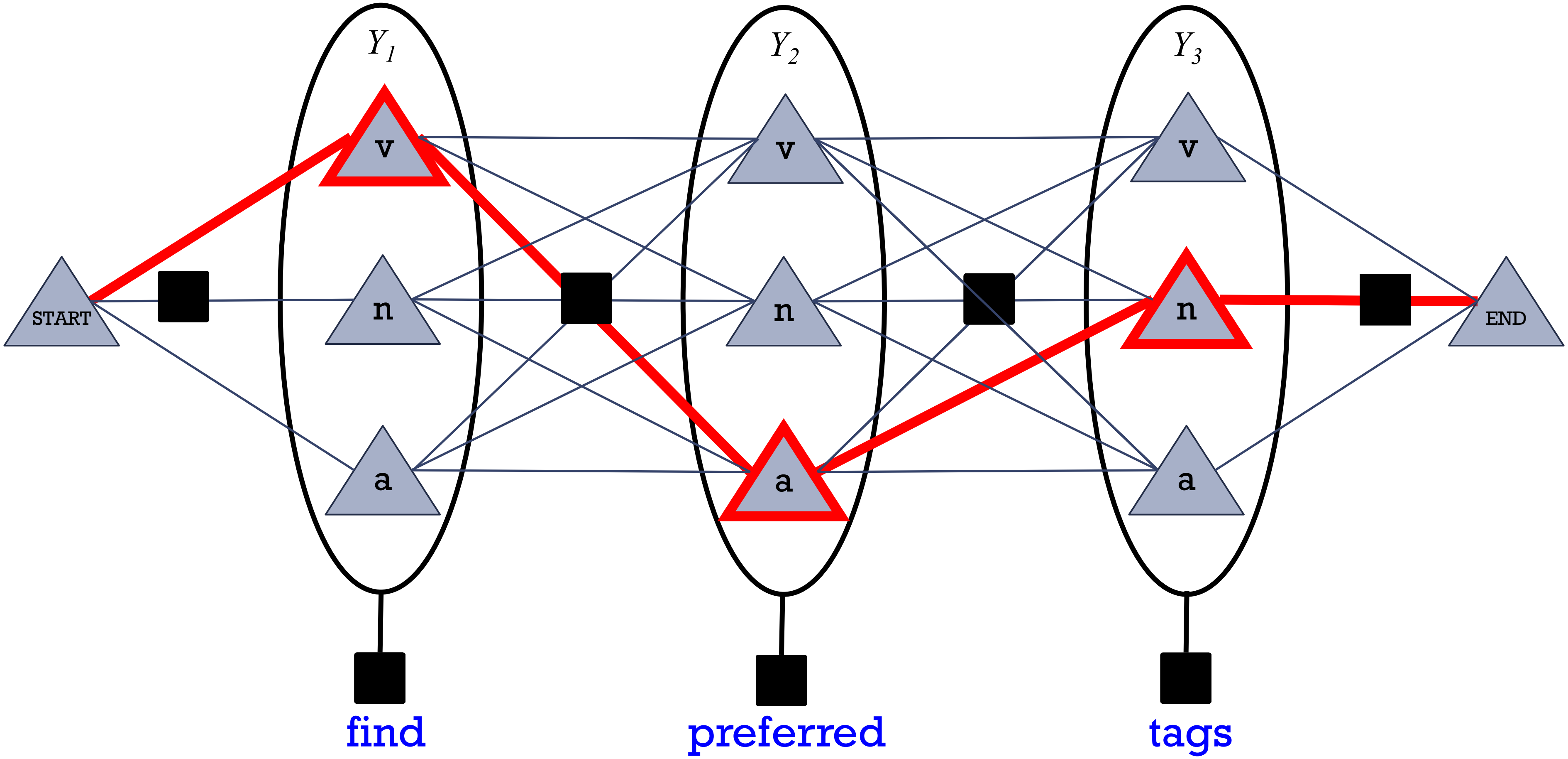
- Show the possible *values* for each variable

So Let's Review Forward-Backward …



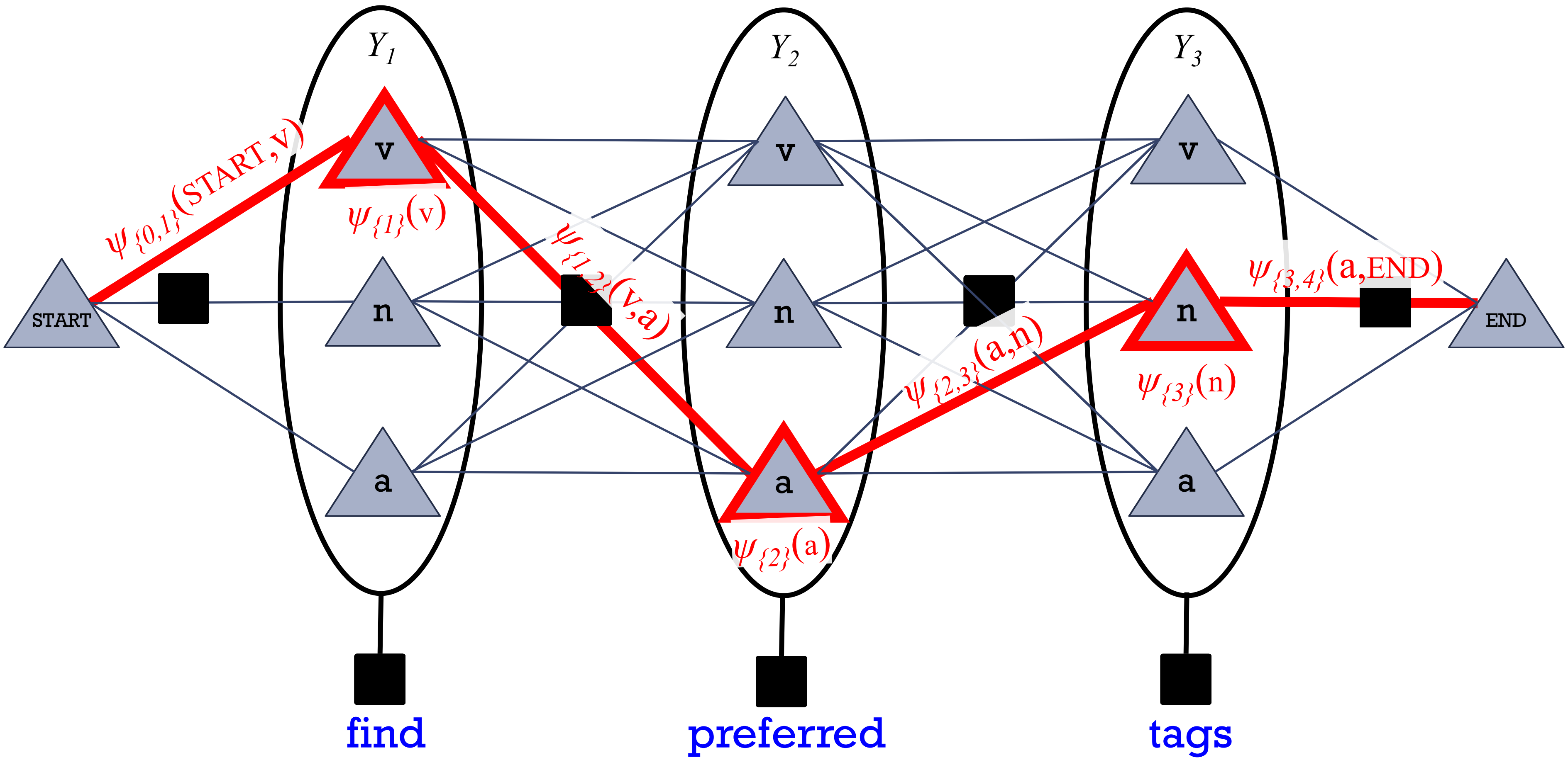
- Let's show the possible *values* for each variable
- One possible assignment

So Let's Review Forward-Backward …



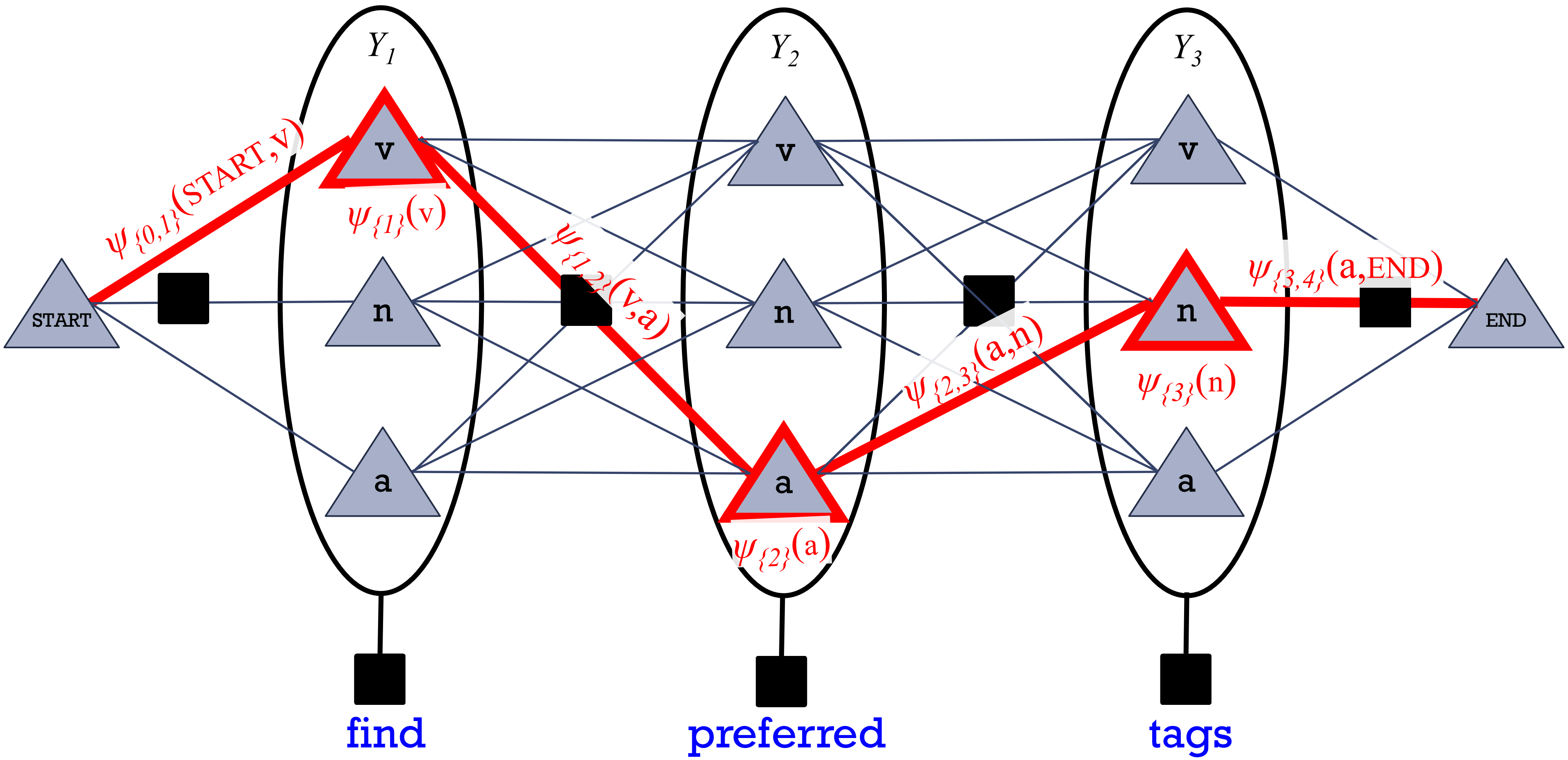
- Let's show the possible *values* for each variable
- One possible assignment
- And what the 7 factors think of it …

Viterbi Algorithm: Most Probable Assignment



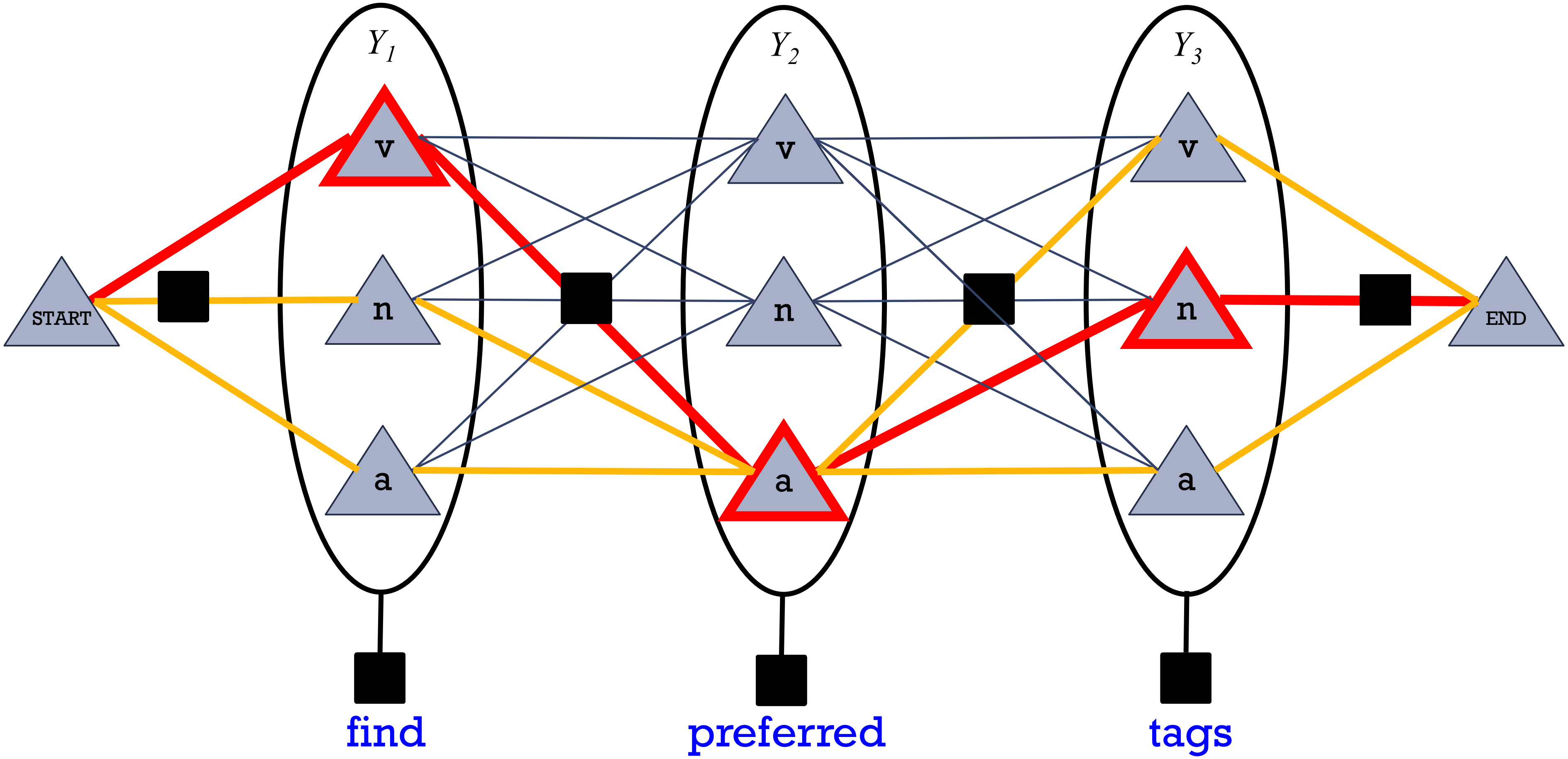
- So p(**v a n**) = (1/Z) * product of 7 numbers
- Numbers associated with edges and nodes of path
- Most probable assignment = **path with highest product**

Viterbi Algorithm: Most Probable Assignment



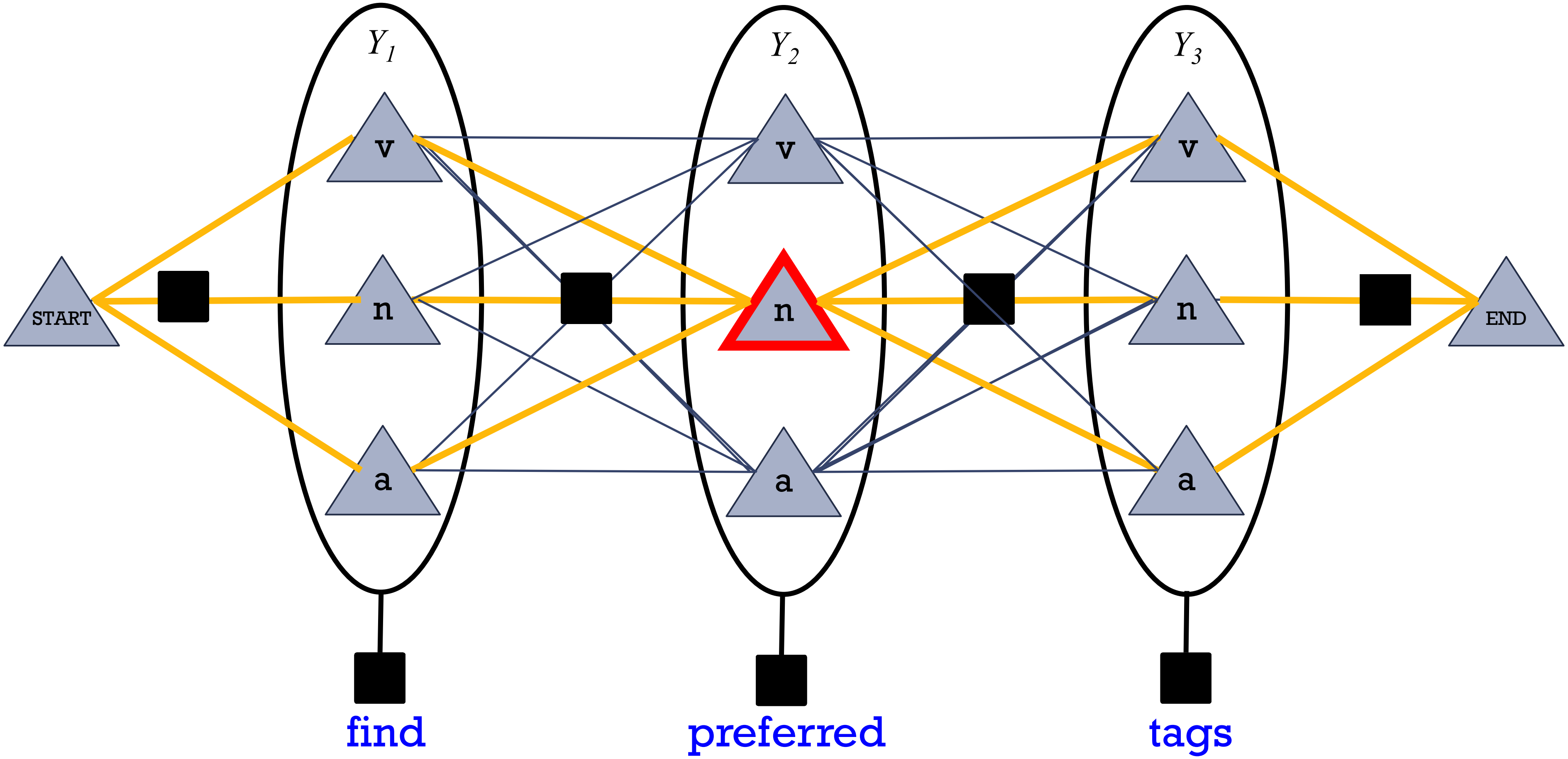
- So p(**v a n**) = (1/Z) * product weight of one path

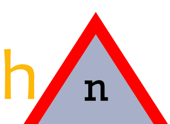
Forward-Backward Algorithm: Finds Marginals



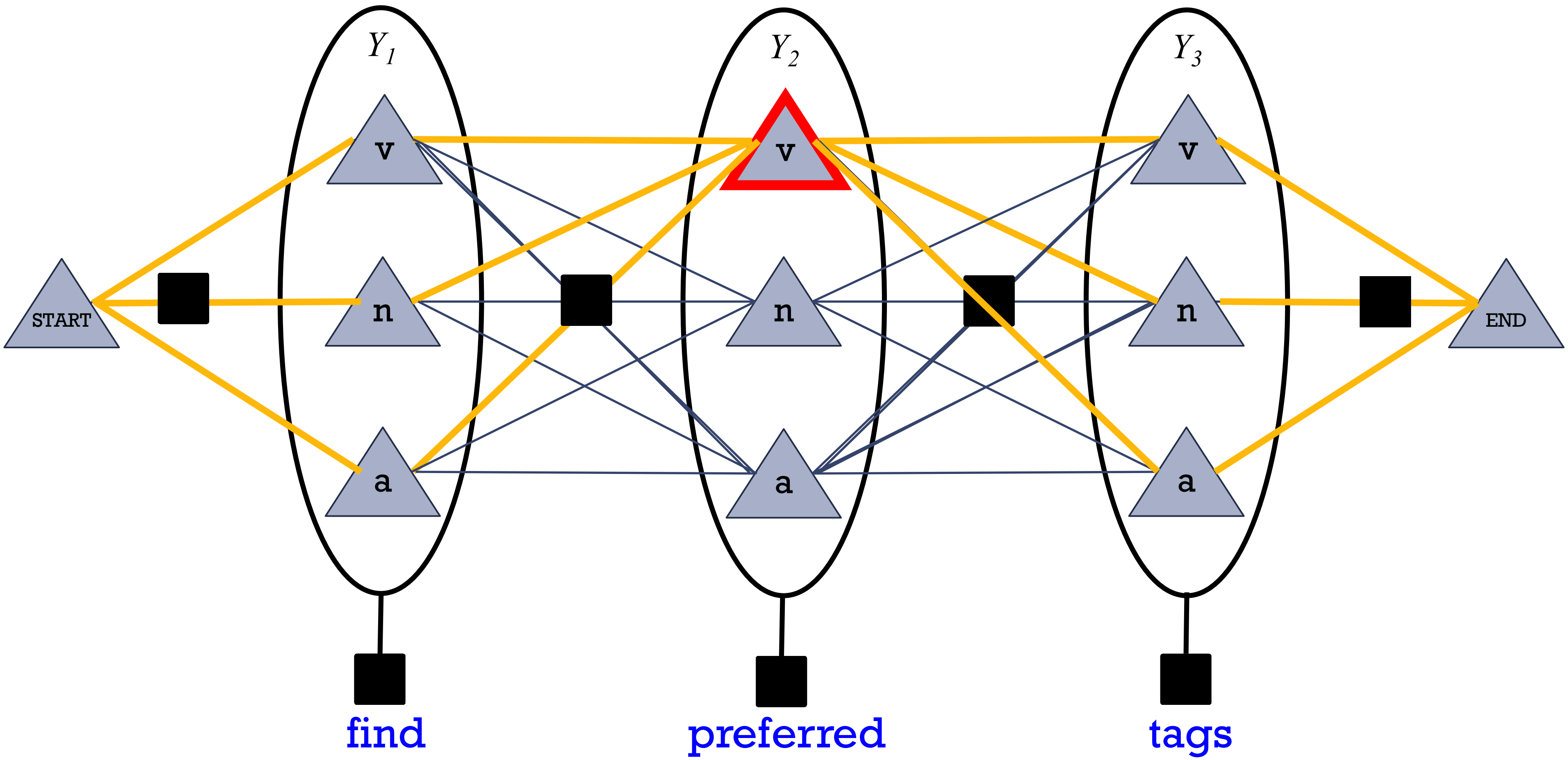
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability $p(Y_2 = \mathrm{a})$
  = (1/Z) * total weight of *all* paths through a

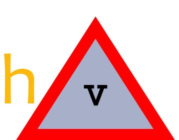
Forward-Backward Algorithm: Finds Marginals



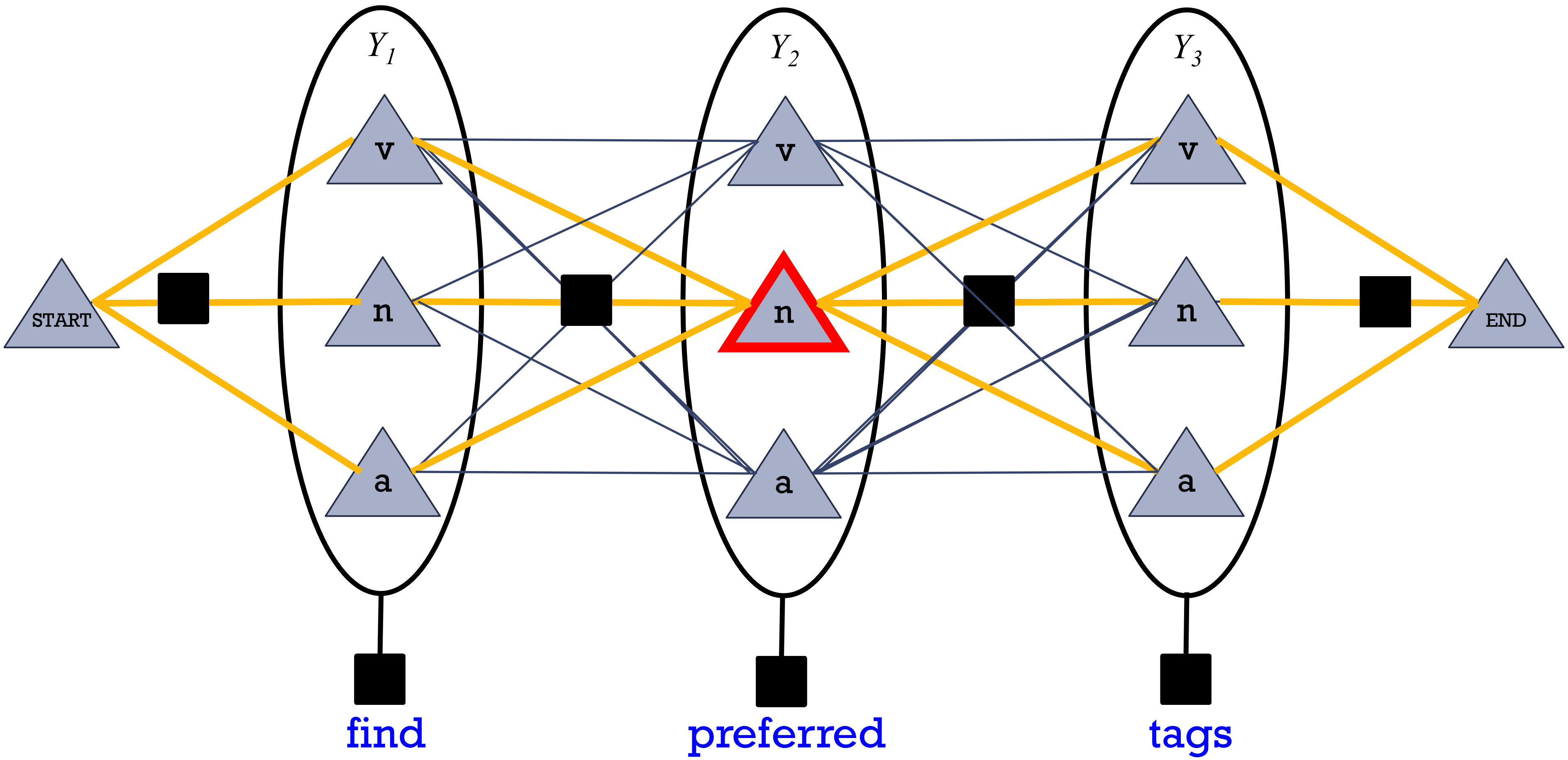
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability $p(Y_2 = a)$
  = (1/Z) * total weight of *all* paths through 

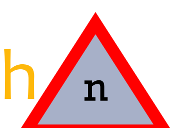
Forward-Backward Algorithm: Finds Marginals



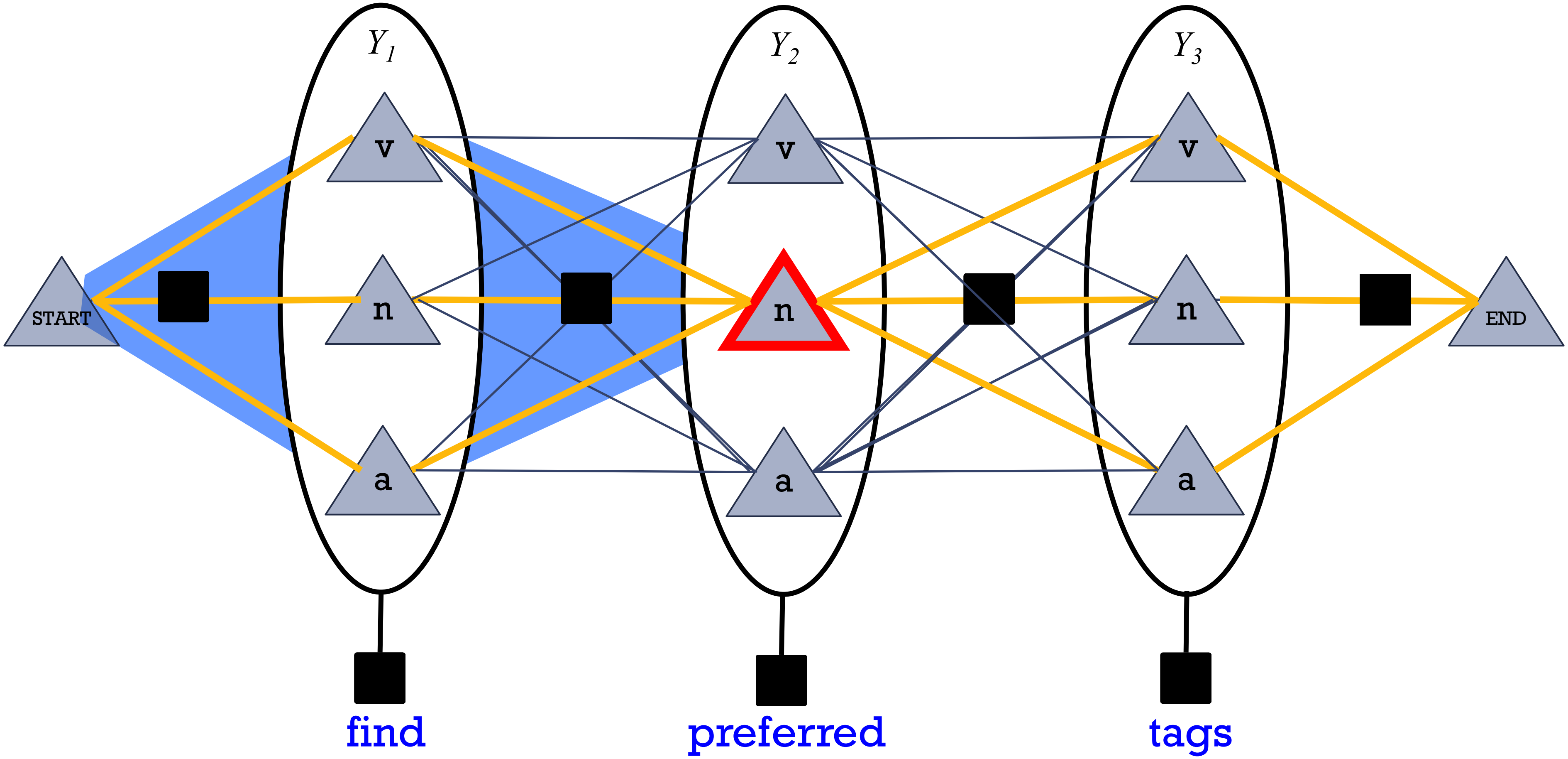
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability $p(Y_2 = a)$
  = (1/Z) * total weight of *all* paths through 

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 

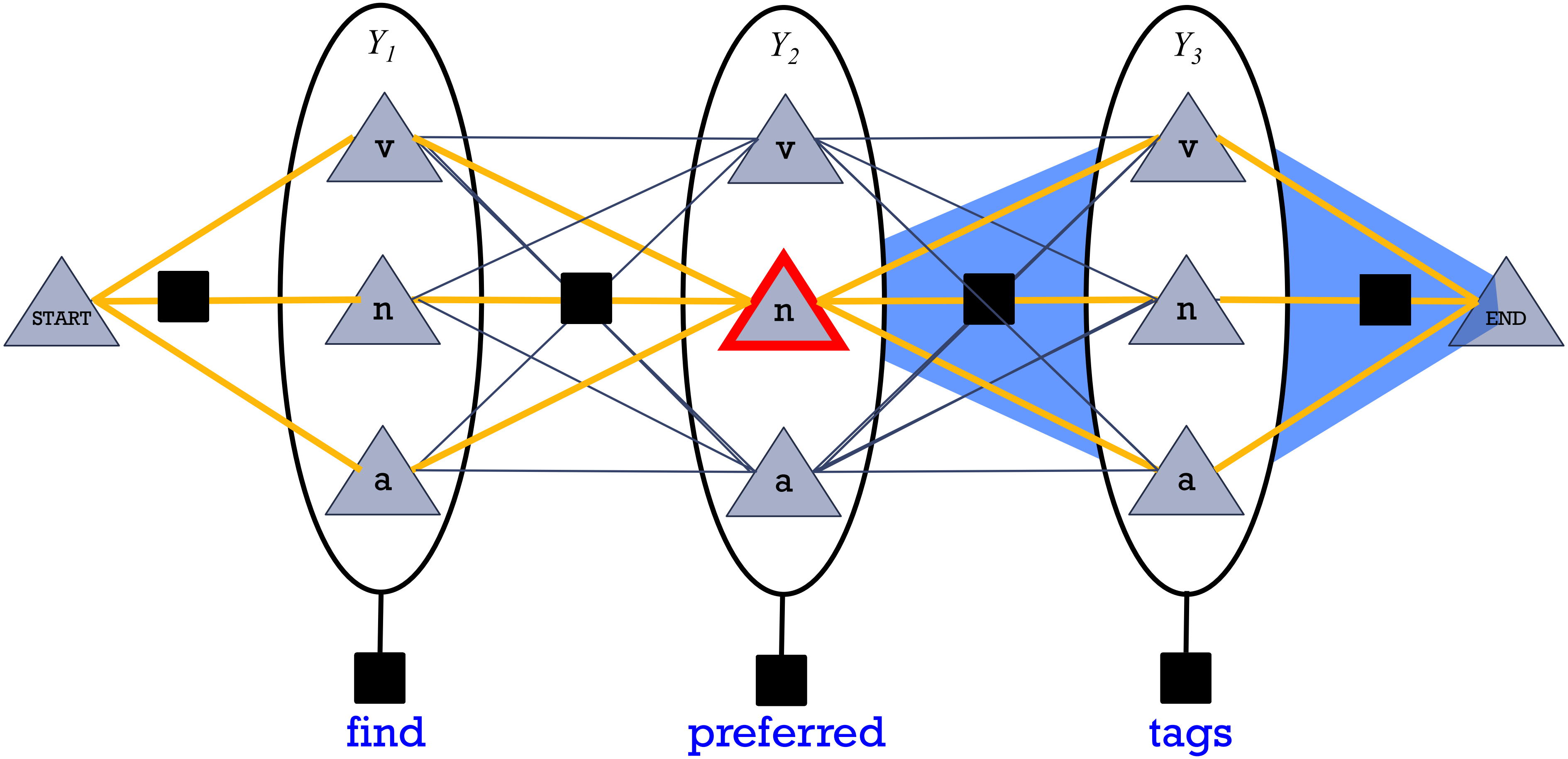
Forward-Backward Algorithm: Finds Marginals



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

(found by dynamic programming: matrix-vector products)

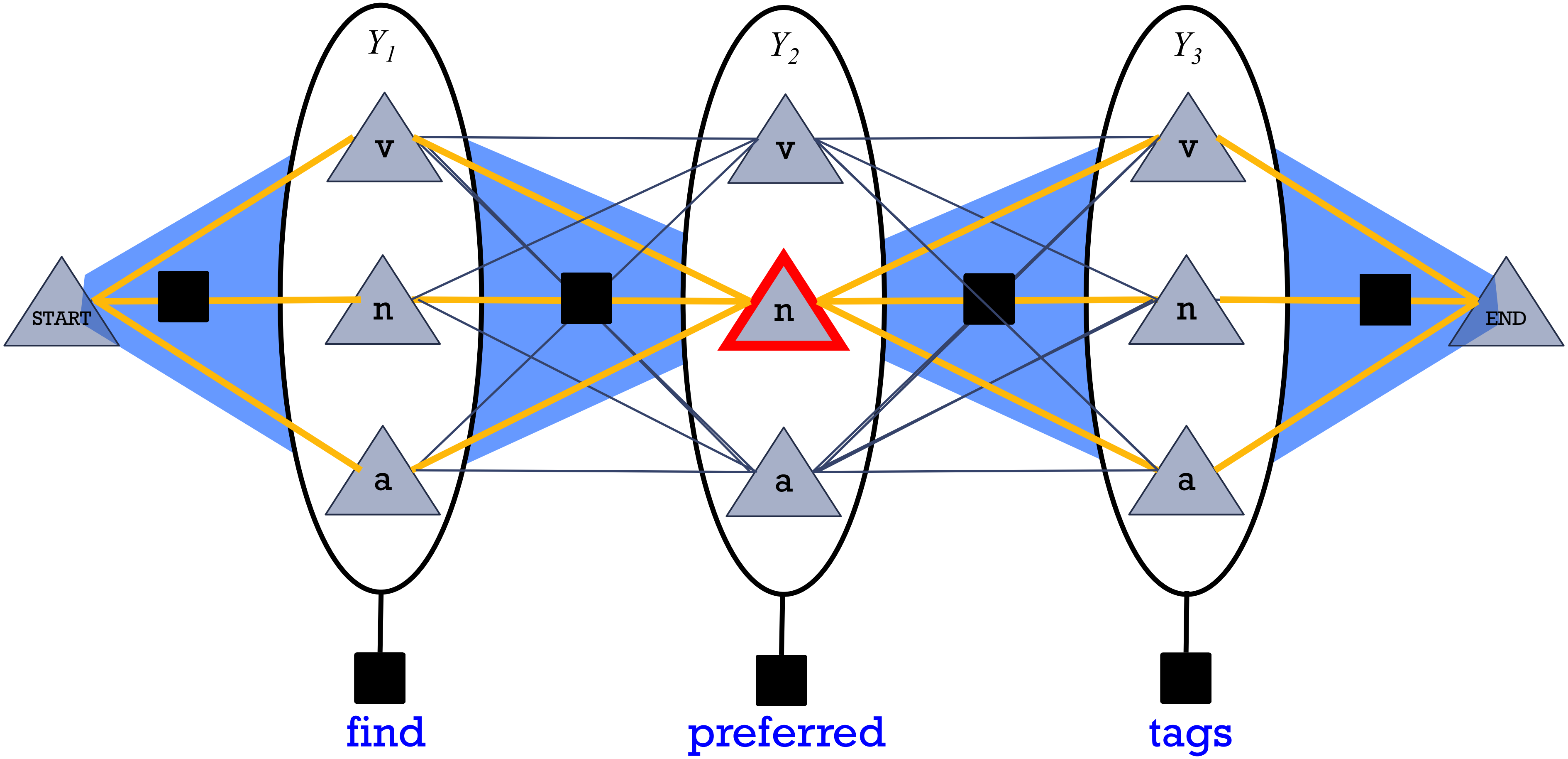
Forward-Backward Algorithm: Finds Marginals



$\beta_2(\mathbf{n})$ = 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)

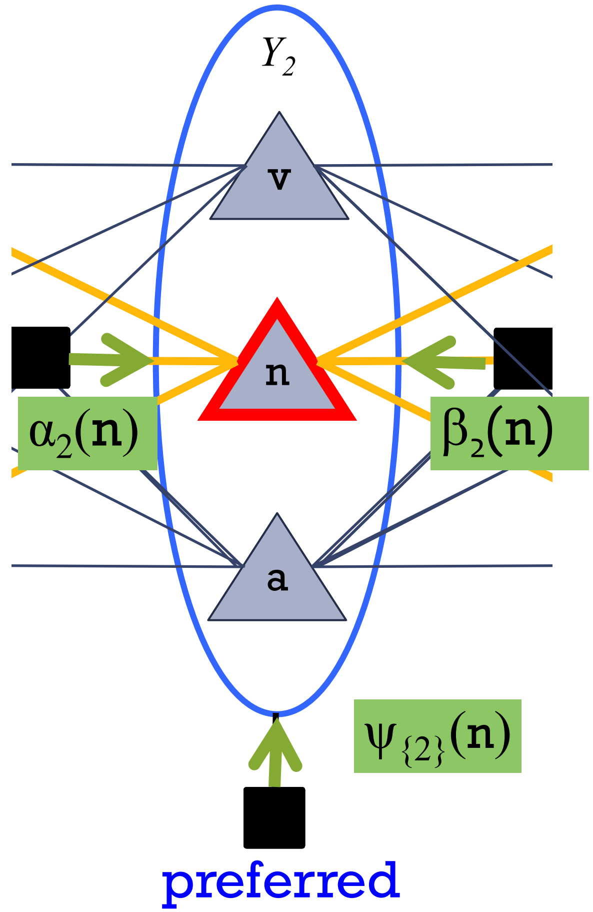
$\beta_2(\mathbf{n})$ = total weight of these path *suffixes* (x + y + z)

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

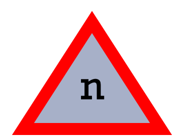
Forward-Backward Algorithm: Finds Marginals

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.

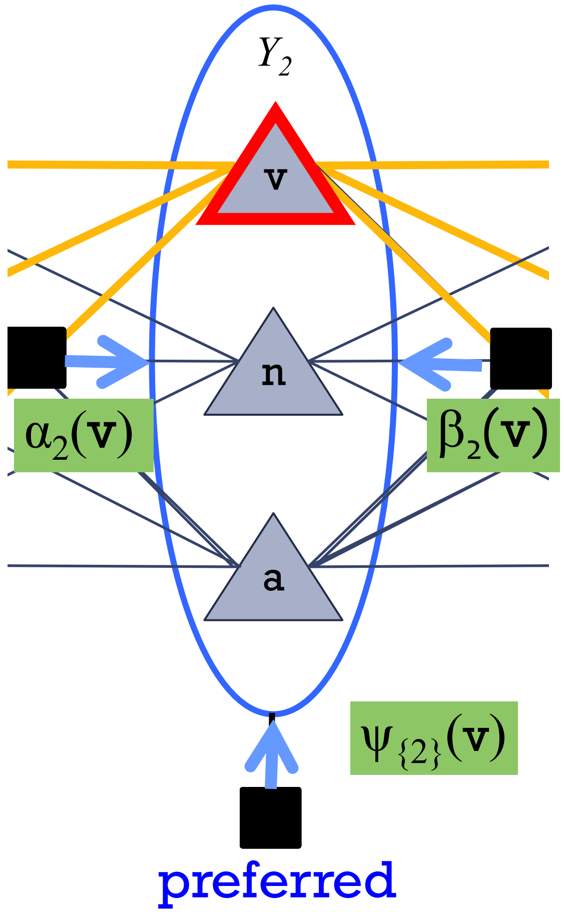


"belief that Y_2 = **n**"

total weight of *all* paths through 

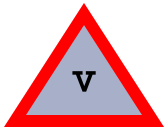
= $\alpha_2(\mathbf{n})$ $\psi_{\{2\}}(\mathbf{n})$ $\beta_2(\mathbf{n})$

Forward-Backward Algorithm: Finds Marginals



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

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

total weight of *all* paths through 

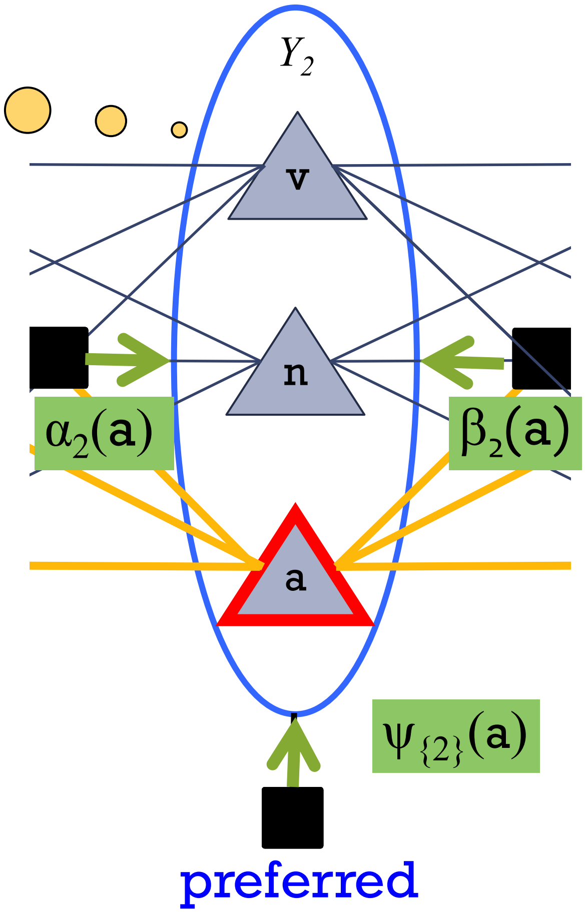
$= \alpha_2(\mathbf{v}) \; \psi_{\{2\}}(\mathbf{v}) \; \beta_2(\mathbf{v})$

Forward-Backward Algorithm: Finds Marginals

v	1.8
n	0
a	4.2

v	0.3
n	0
a	0.7

divide by Z=6 to get marginal probs



"belief that Y_2 = **v**"

"belief that Y_2 = **n**"

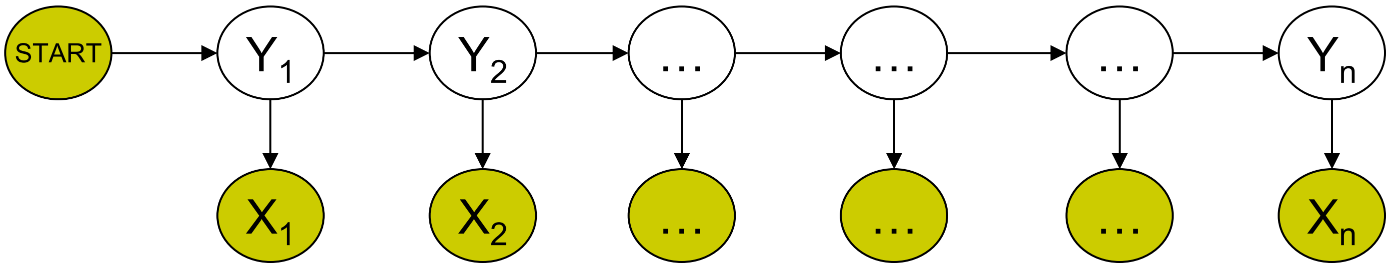
"belief that Y_2 = **a**"

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

total weight of *all* paths through **a**

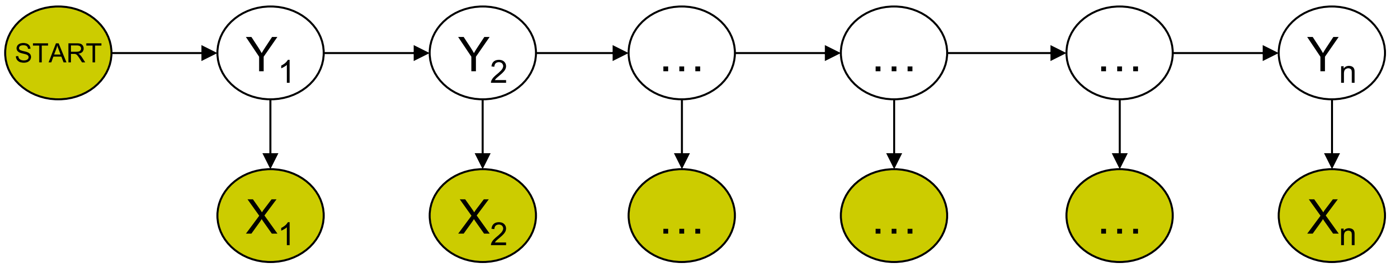
= $\alpha_2(\mathbf{a})\ \psi_{\{2\}}(\mathbf{a})\ \beta_2(\mathbf{a})$

Hidden Markov Model



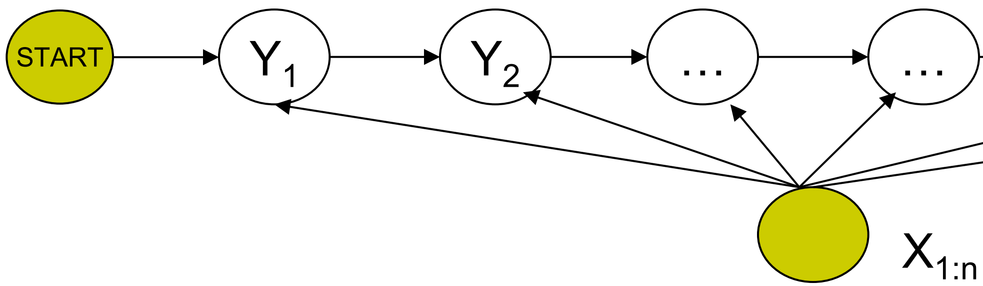
$$P(\boldsymbol{x}_{1:n}, \boldsymbol{y}_{1:n}) = \prod_{i=1}^{n} P(x_i|y_i)P(y_i|y_{i-1})$$

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**)

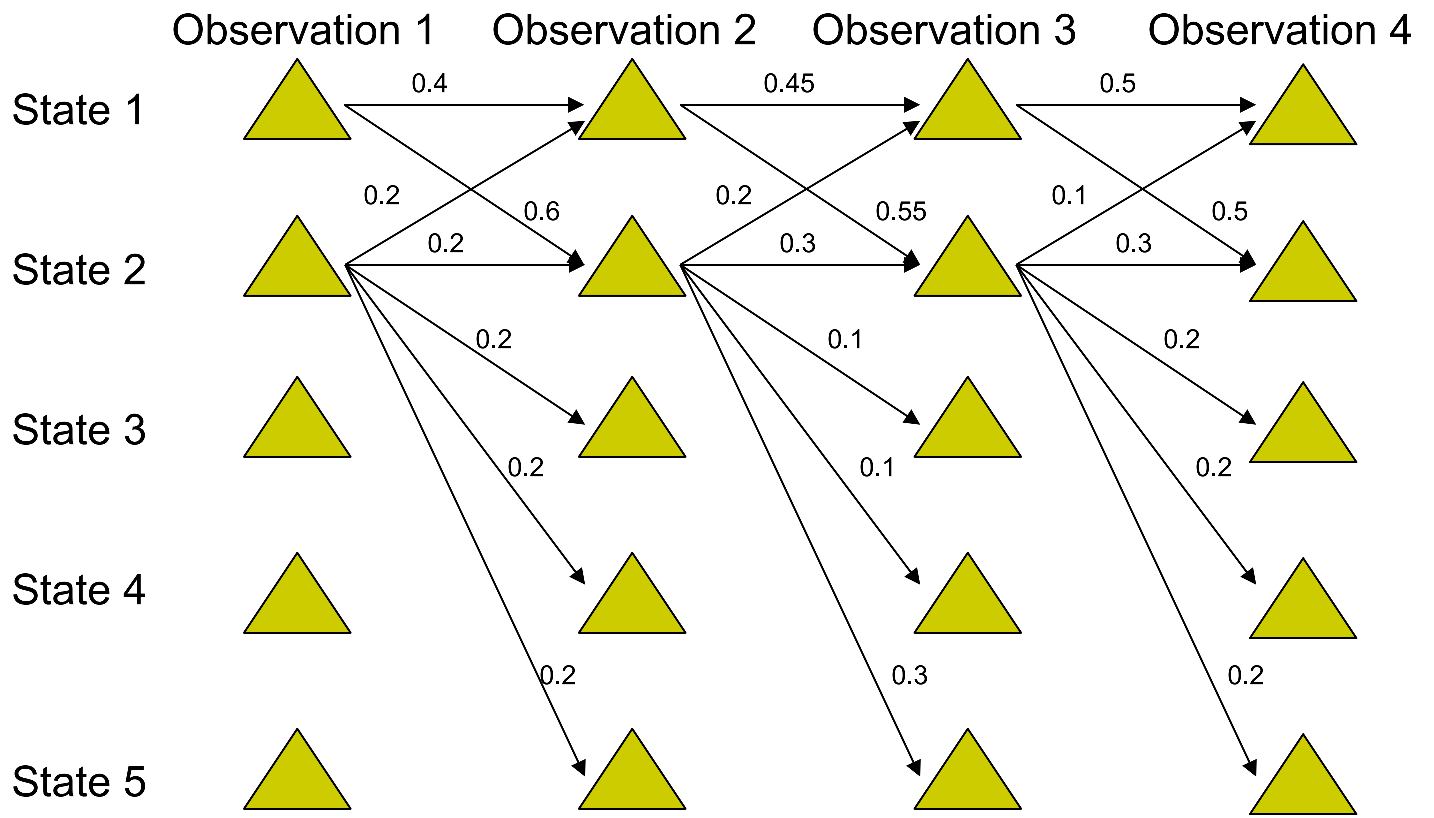
A Solution:
Maximum Entropy Markov Model (MEMM)



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

- 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(**X**): saves modeling effort
  - Learning objective function consistent with predictive function: P(**Y**|**X**)

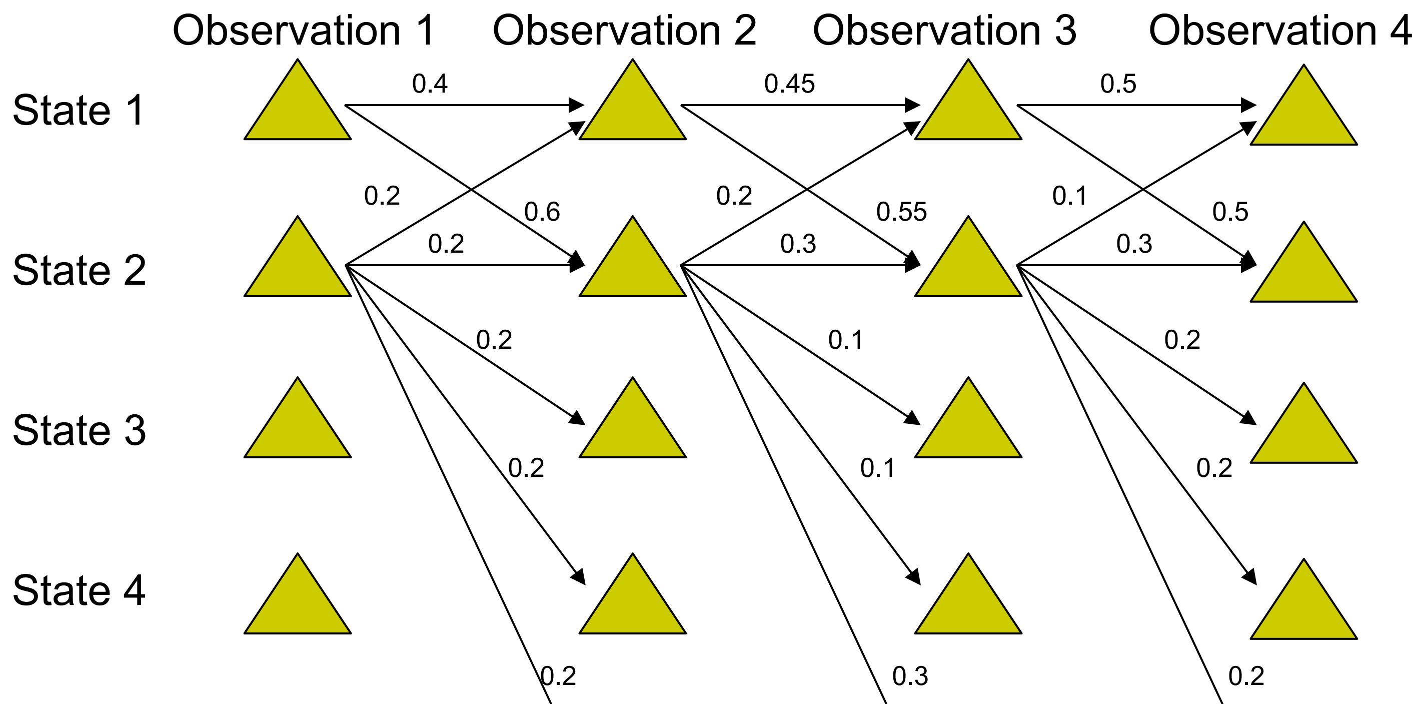
Then, shortcomings of MEMM (and HMM) (2): the Label bias problem



What the local transition probabilities say:

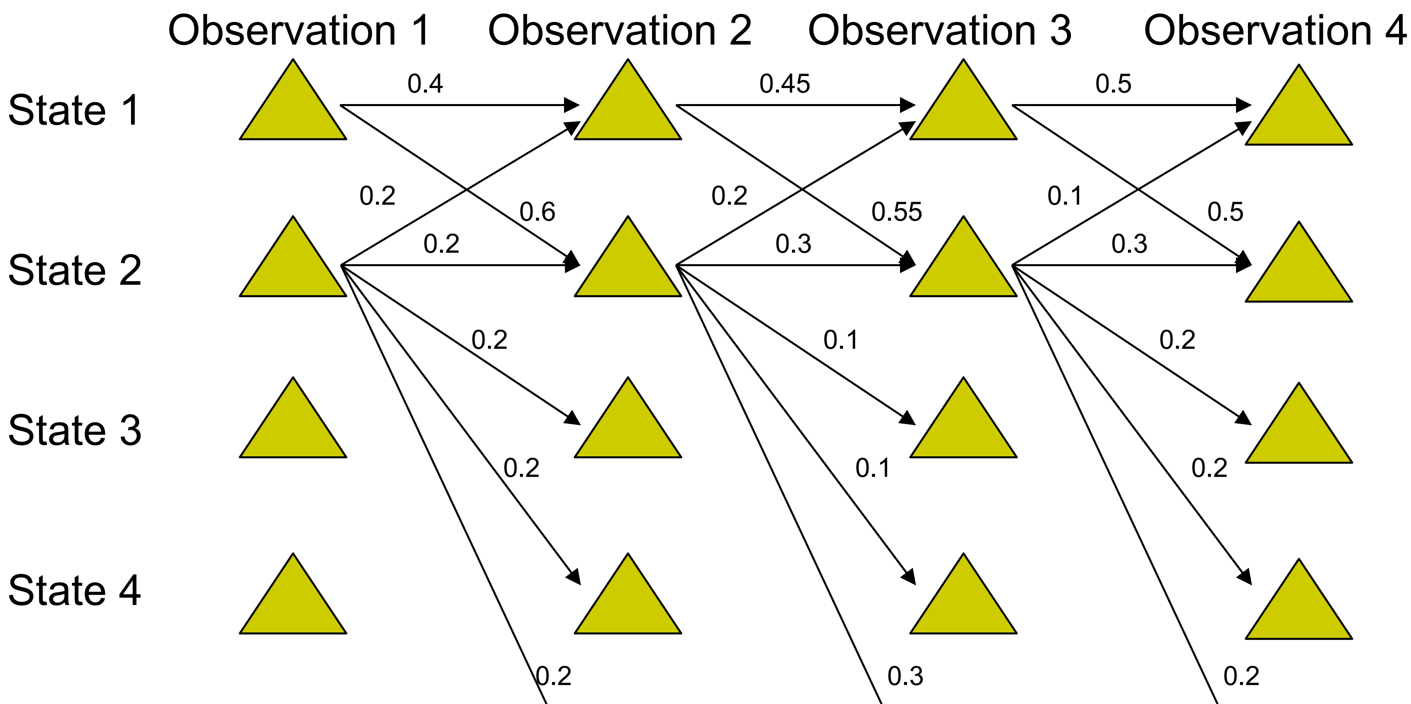
- State 1 almost always prefers to go to state 2
- State 2 almost always prefers to stay in state 2

MEMM: the Label bias problem



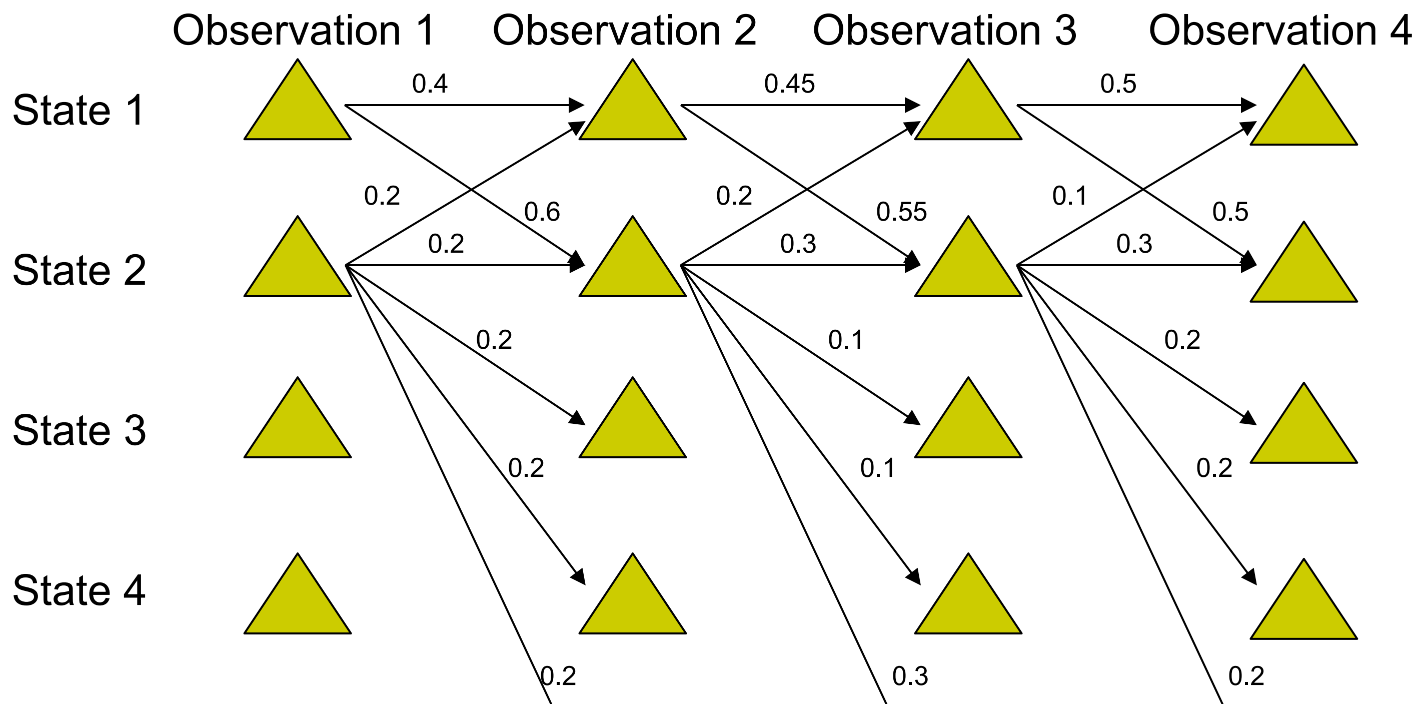
Path	Probability
1 → 1 → 1 → 1	

MEMM: the Label bias problem



Path	Probability
1 → 1 → 1 → 1	0.4 x 0.45 x 0.5 = 0.090

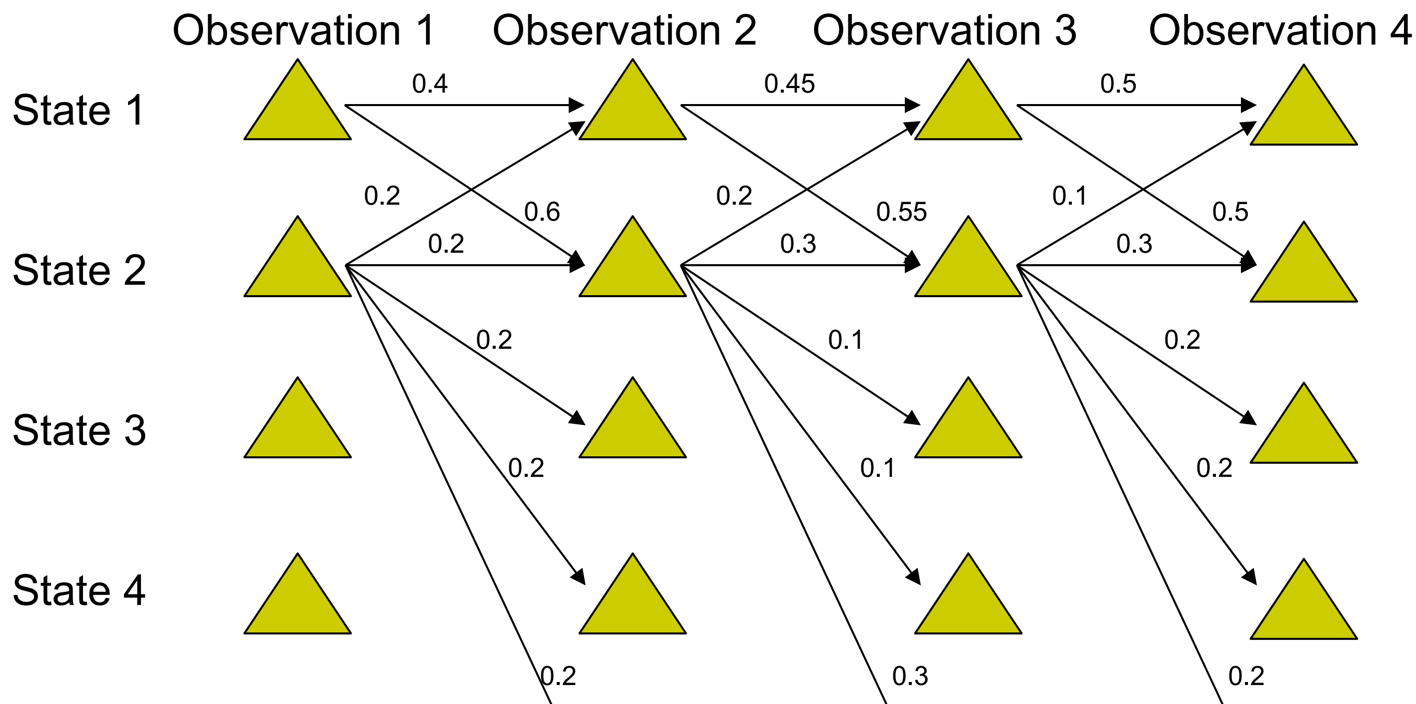
MEMM: the Label bias problem



State 5

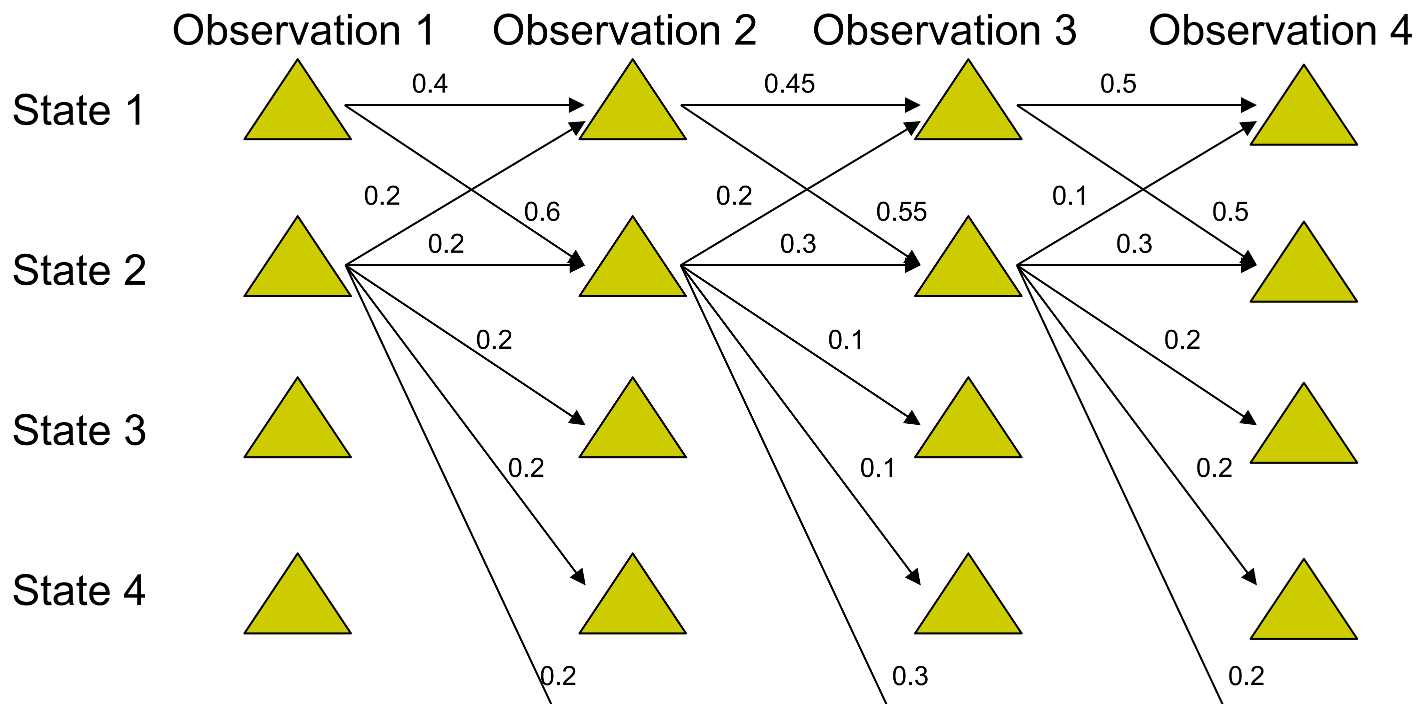
Path	Probability
1 → 1 → 1 → 1	0.4 x 0.45 x 0.5 = 0.090
2 → 2 → 2 → 2	0.2 x 0.30 x 0.3 = 0.018

MEMM: the Label bias problem



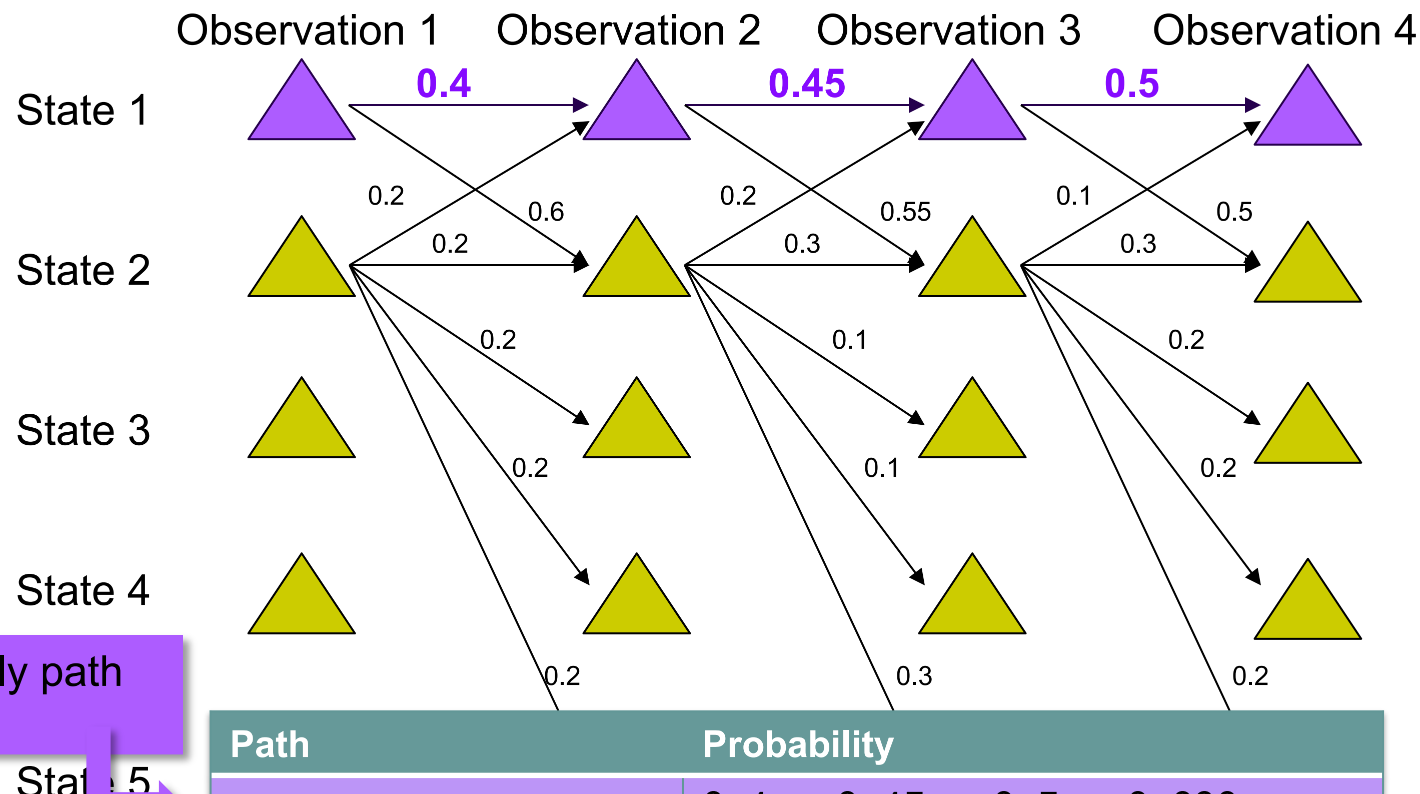
Path	Probability
1 → 1 → 1 → 1	0.4 x 0.45 x 0.5 = 0.090
2 → 2 → 2 → 2	0.2 x 0.30 x 0.3 = 0.018
1 → 2 → 1 → 2	0.6 x 0.20 x 0.5 = 0.060

MEMM: the Label bias problem



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

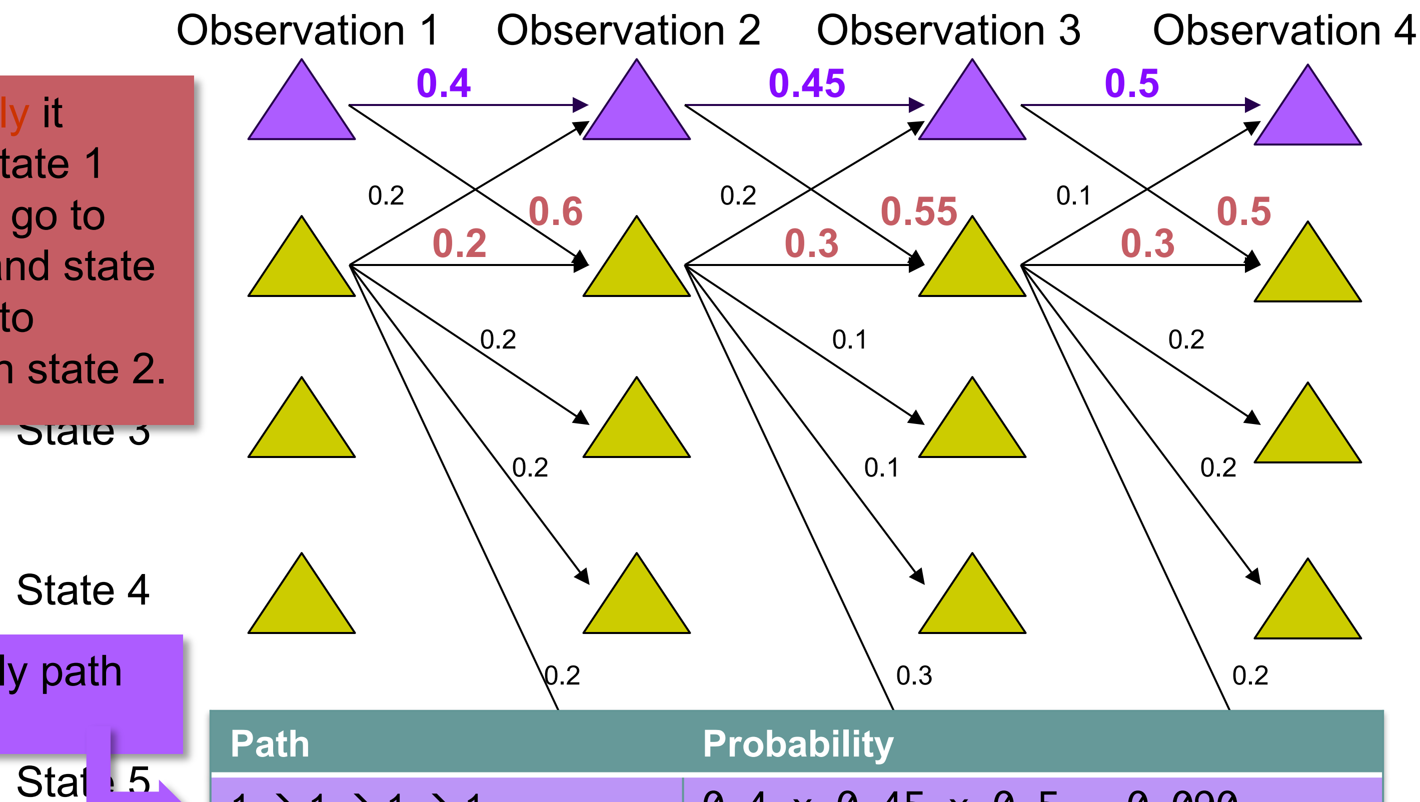
MEMM: the Label bias problem



Most likely path

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

MEMM: the Label bias problem

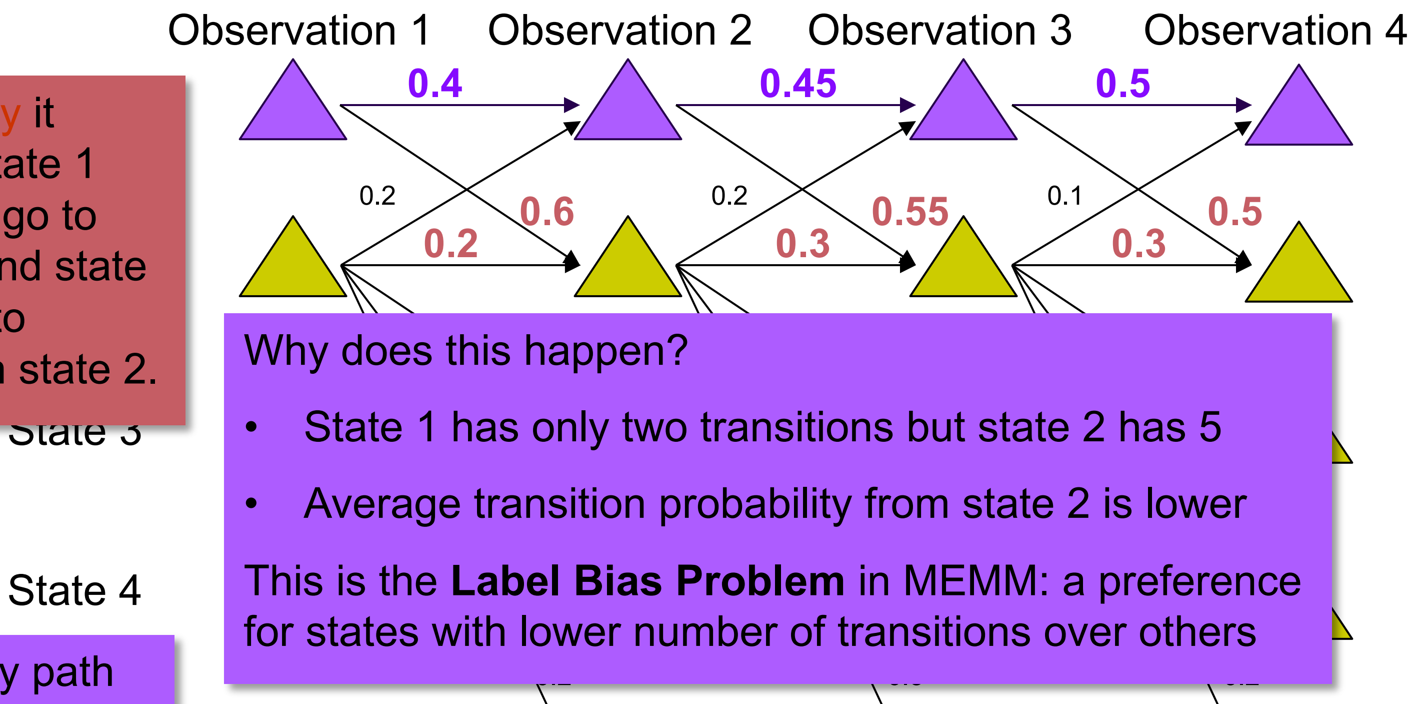


Yet locally it seems state 1 wants to go to state 2 and state 2 wants to remain in state 2.

Most likely path

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

MEMM: the Label bias problem



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

Why does this happen?

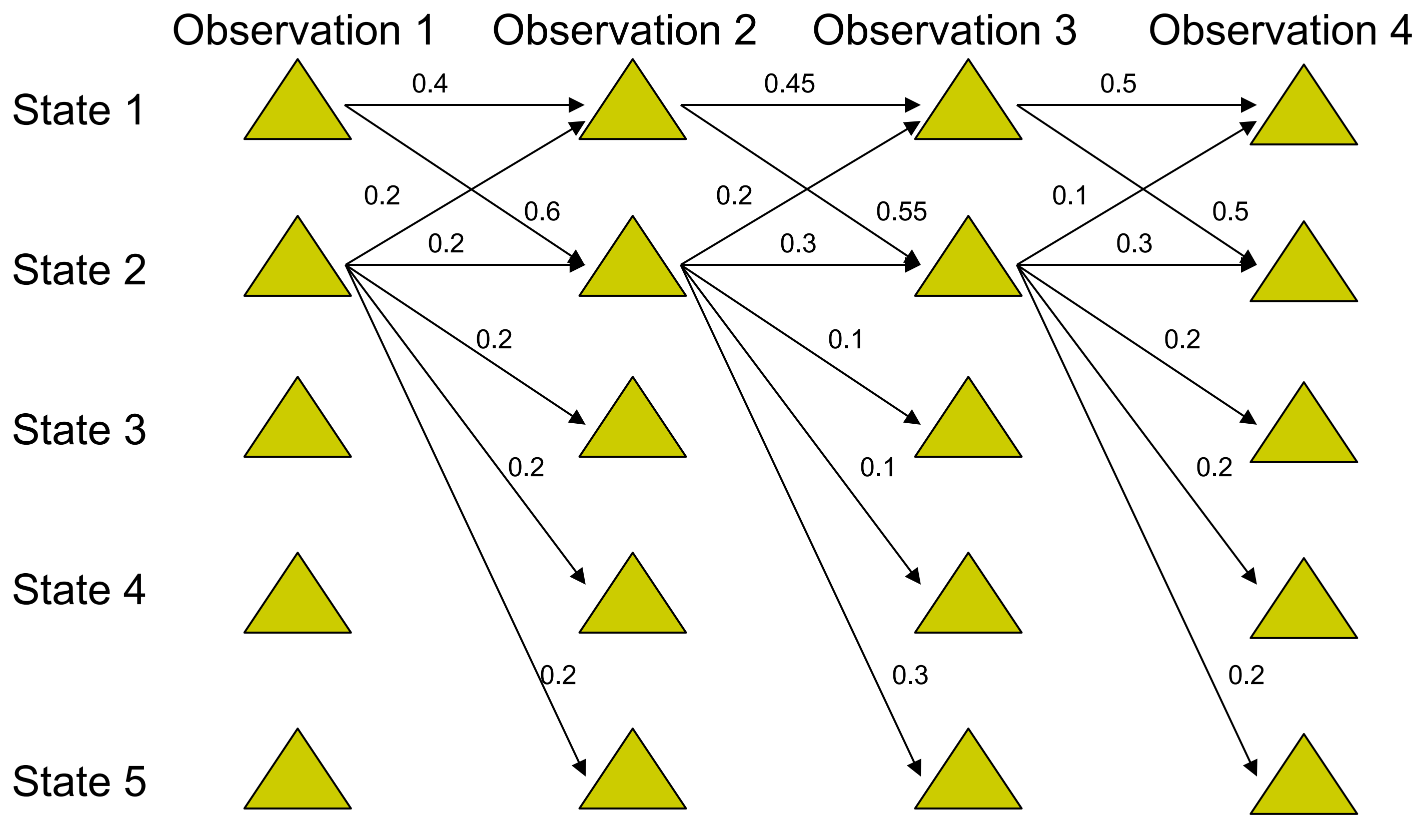
- State 1 has only two transitions but state 2 has 5
- Average transition probability from state 2 is lower

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

Most likely path

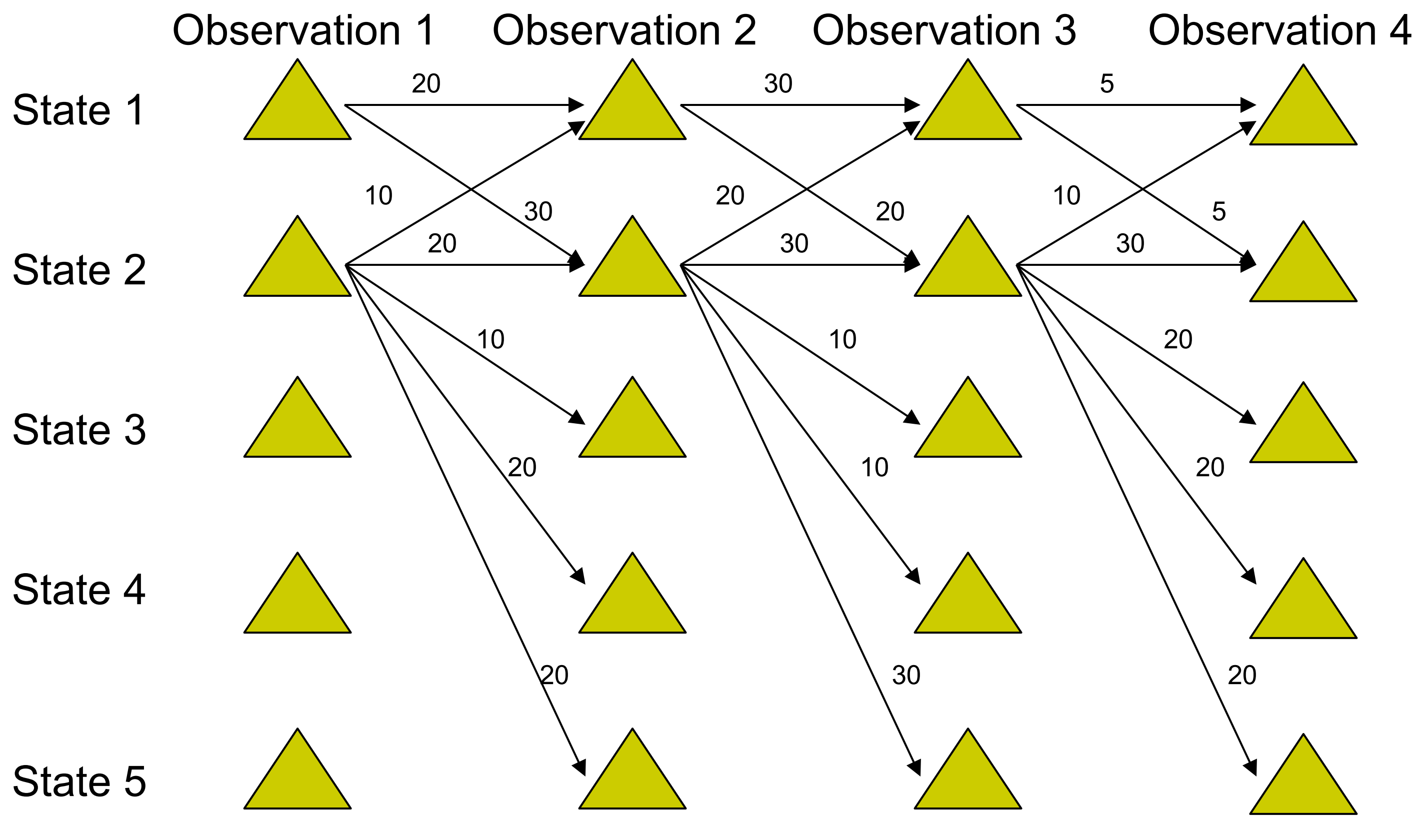
Path	Probability
1 → 1 → 1 → 1	0.4 x 0.45 x 0.5 = 0.090
2 → 2 → 2 → 2	0.2 x 0.30 x 0.3 = 0.018
1 → 2 → 1 → 2	0.6 x 0.20 x 0.5 = 0.060
1 → 1 → 2 → 2	0.4 x 0.55 x 0.3 = 0.066

Solution:
Do not normalize probabilities locally



From local probabilities…

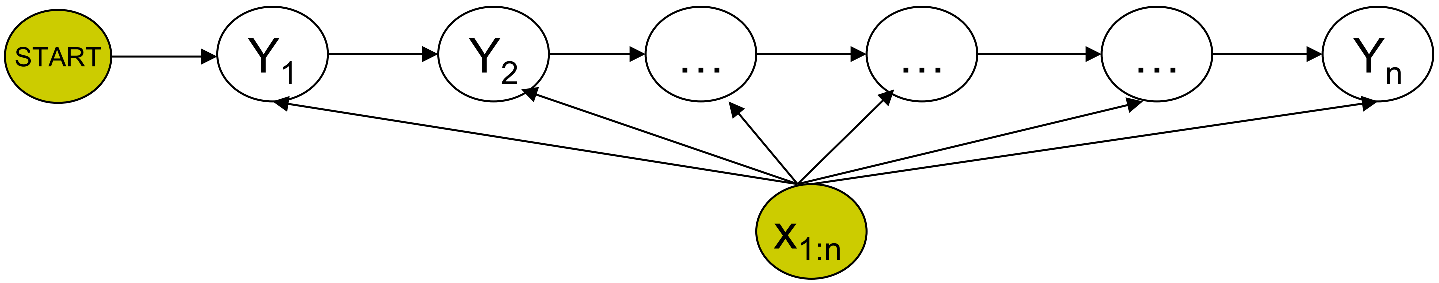
Solution:
Do not normalize probabilities locally



From local probabilities to local potentials!

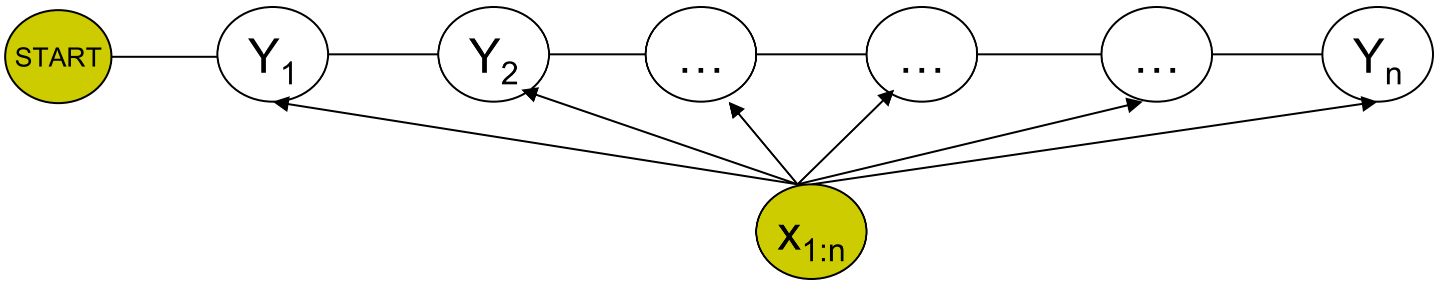
States with lower transitions do not have an unfair advantage!

From MEMM ….



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

From MEMM to Linear-chain CRF

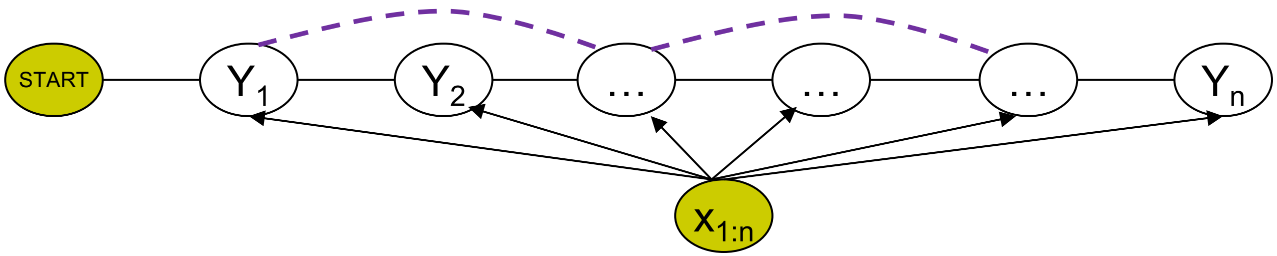


$$P(\mathbf{y}_{1:n}|\mathbf{x}_{1:n}) = \frac{1}{Z(\mathbf{x}_{1:n})}\prod_{i=1}^{n}\phi(y_i, y_{i-1}, \mathbf{x}_{1:n}) = \frac{1}{Z(\mathbf{x}_{1:n}, \mathbf{w})}\prod_{i=1}^{n}\exp(\mathbf{w}^T\mathbf{f}(y_i, y_{i-1}, \mathbf{x}_{1:n}))$$

- CRF is a partially directed model
  - Discriminative model like MEMM
  - Unlike MEMM, each factor is not normalized. Hence, usage of global Z(**x**) overcomes the label bias problem of MEMM
  - Models the dependence between each state and the entire observation sequence (like MEMM)

Linear-chain CRF

- Linear-chain Conditional Random Field parametric form:



$$
\begin{aligned}
P(\mathbf{y}|\mathbf{x}) &= \frac{1}{Z(\mathbf{x},\lambda,\mu)}\exp\Big(\sum_{i=1}^{n}\big(\sum_{k}\lambda_k f_k(y_i,y_{i-1},\mathbf{x}) + \sum_{l}\mu_l g_l(y_i,\mathbf{x})\big)\Big) \\
&= \frac{1}{Z(\mathbf{x},\lambda,\mu)}\exp\Big(\sum_{i=1}^{n}\big(\lambda^T \overset{\text{binary}}{\boxed{\mathbf{f}(y_i,y_{i-1},\mathbf{x})}} + \mu^T \overset{\text{unary}}{\boxed{\mathbf{g}(y_i,\mathbf{x})}}\big)\Big)
\end{aligned}
$$

$$
\text{where } Z(\mathbf{x},\lambda,\mu) = \sum_{\mathbf{y}}\exp\Big(\sum_{i=1}^{n}\big(\lambda^T\mathbf{f}(y_i,y_{i-1},\mathbf{x}) + \mu^T\mathbf{g}(y_i,\mathbf{x})\big)\Big)
$$

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 $f_k(x,y)$ 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

$_0$ Time $_1$ flies $_2$ like $_3$ an $_4$ arrow $_5$

- 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.
  - 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 (x,y)?

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

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

At <u>each position</u> 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)).

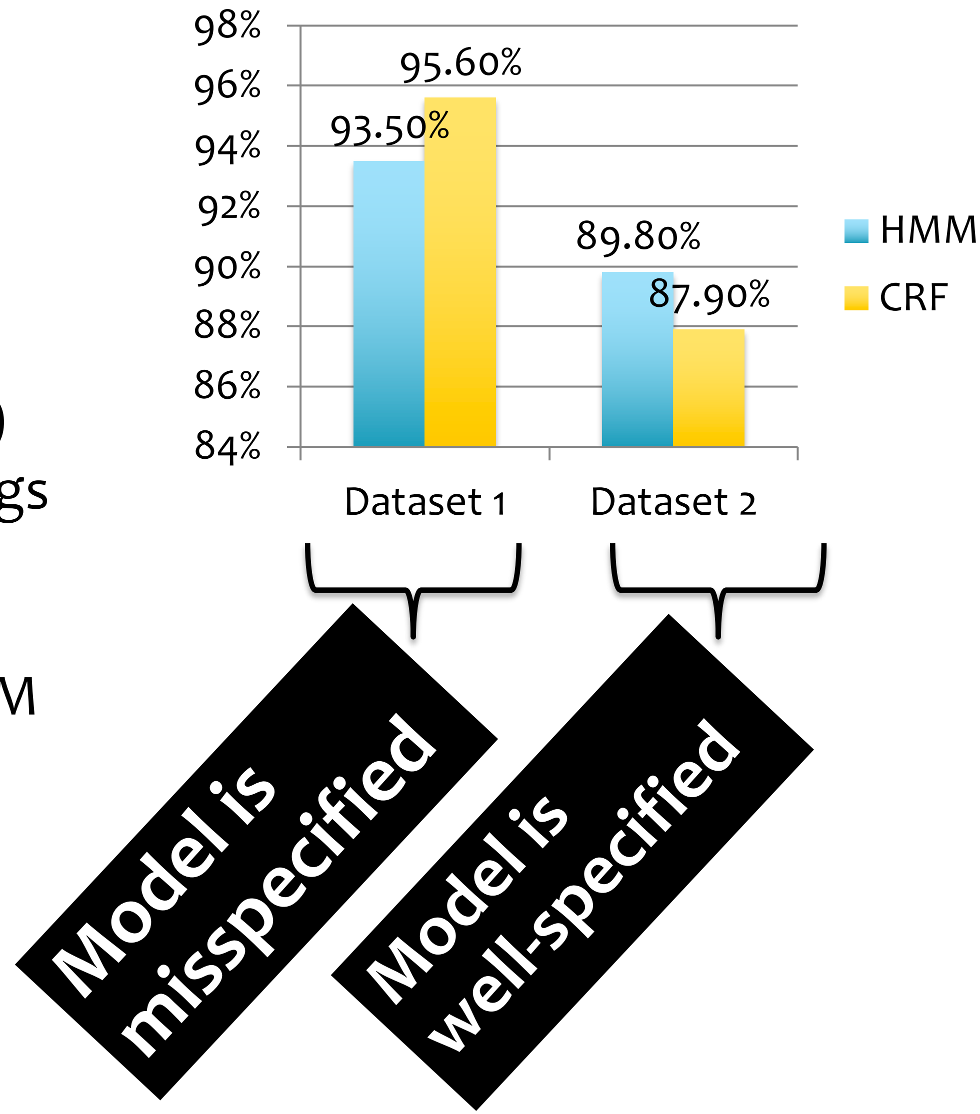
Note: Every template should mention at least some blue.

- 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.

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

Minimum Bayes Risk Decoding

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

$$\begin{aligned} h_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \underset{\hat{\boldsymbol{y}}}{\operatorname{argmin}} \; \mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})}[\ell(\hat{\boldsymbol{y}}, \boldsymbol{y})] \\ &= \underset{\hat{\boldsymbol{y}}}{\operatorname{argmin}} \; \sum_{\boldsymbol{y}} p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) \ell(\hat{\boldsymbol{y}}, \boldsymbol{y}) \end{aligned}$$

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:

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

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:

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

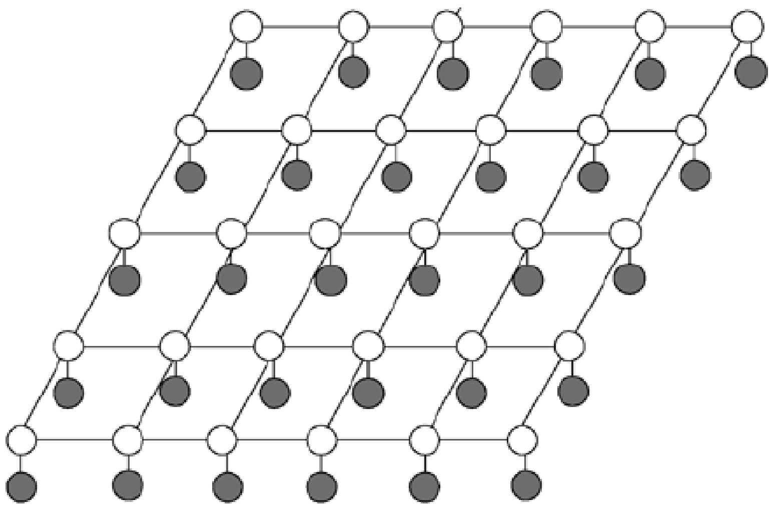
This decomposes across variables and requires the variable marginals.

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

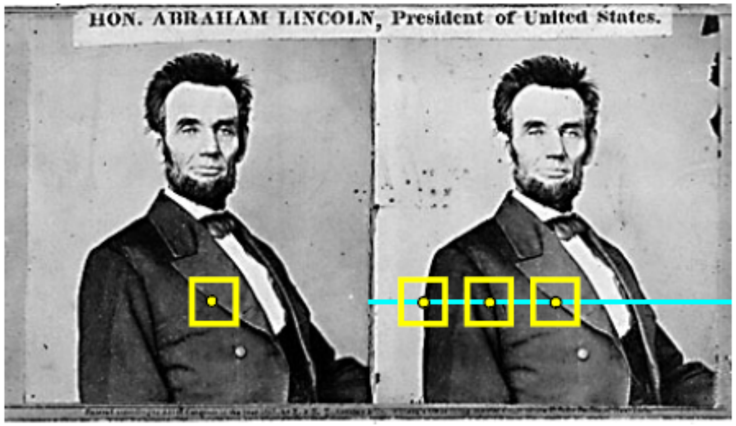


Image Restoration

Image Segmentation

Application: Image Segmentation

$\phi_i(y_i, x) \in \mathbb{R}^{\approx 1000}$: local image features, e.g. bag-of-words

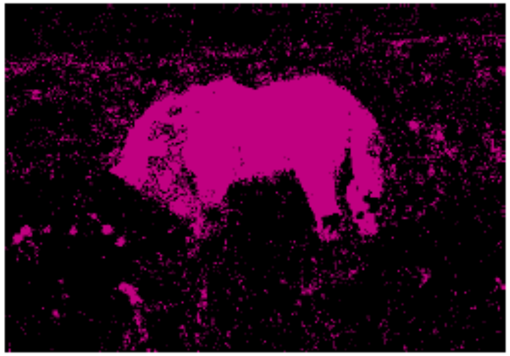
$\rightarrow \quad \langle w_i, \phi_i(y_i, x) \rangle$: local classifier (like logistic-regression)

$\phi_{i,j}(y_i, y_j) = [\![y_i = y_j]\!] \in \mathbb{R}^1$: test for same label

$\rightarrow \quad \langle w_{ij}, \phi_{ij}(y_i, y_j) \rangle$: 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

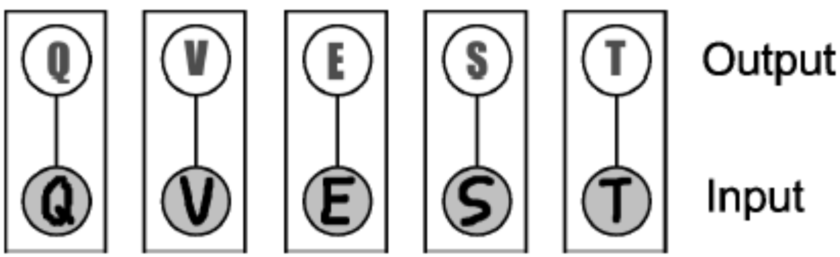
$\phi_i(y_i, x) \in \mathbb{R}^{\approx 1000}$: image representation (pixels, gradients)

$\rightarrow \quad \langle w_i, \phi_i(y_i, x) \rangle$: local classifier if x_i is letter y_i

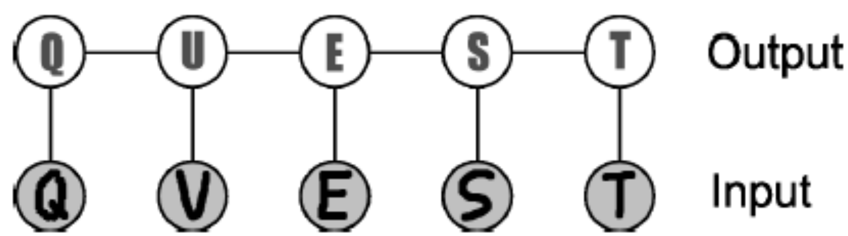
$\phi_{i,j}(y_i, y_j) = e_{y_i} \otimes e_{y_j} \in \mathbb{R}^{26 \cdot 26}$: letter/letter indicator

$\rightarrow \quad \langle w_{ij}, \phi_{ij}(y_i, y_j) \rangle$: encourage/suppress letter combinations

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

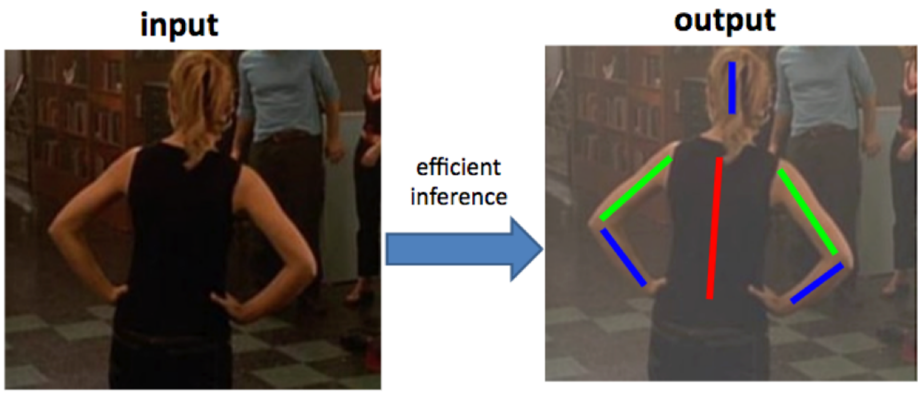


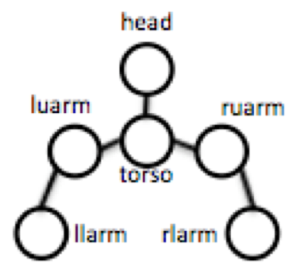
local classification

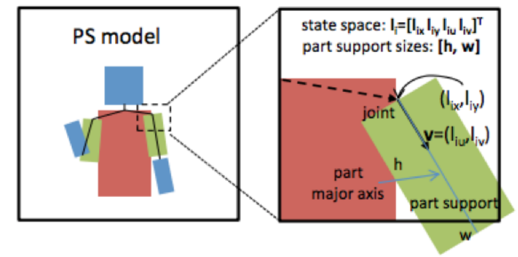


local + "correction"

Application: Pose Estimation







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

Penalizes unrealistic poses

Local classifier for each part

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

Cascaded Models for Articulated Pose Estimation, B. Sapp, A. Toshev, B. Taskar

Feature Functions for CRF in Vision

$\phi_i(y_i, x)$: local representation, high-dimensional

$\rightarrow \quad \langle w_i, \phi_i(y_i, x)\rangle$: local classifier

$\phi_{i,j}(y_i, y_j)$: prior knowledge, low-dimensional

$\rightarrow \quad \langle w_{ij}, \phi_{ij}(y_i, y_j)\rangle$: 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

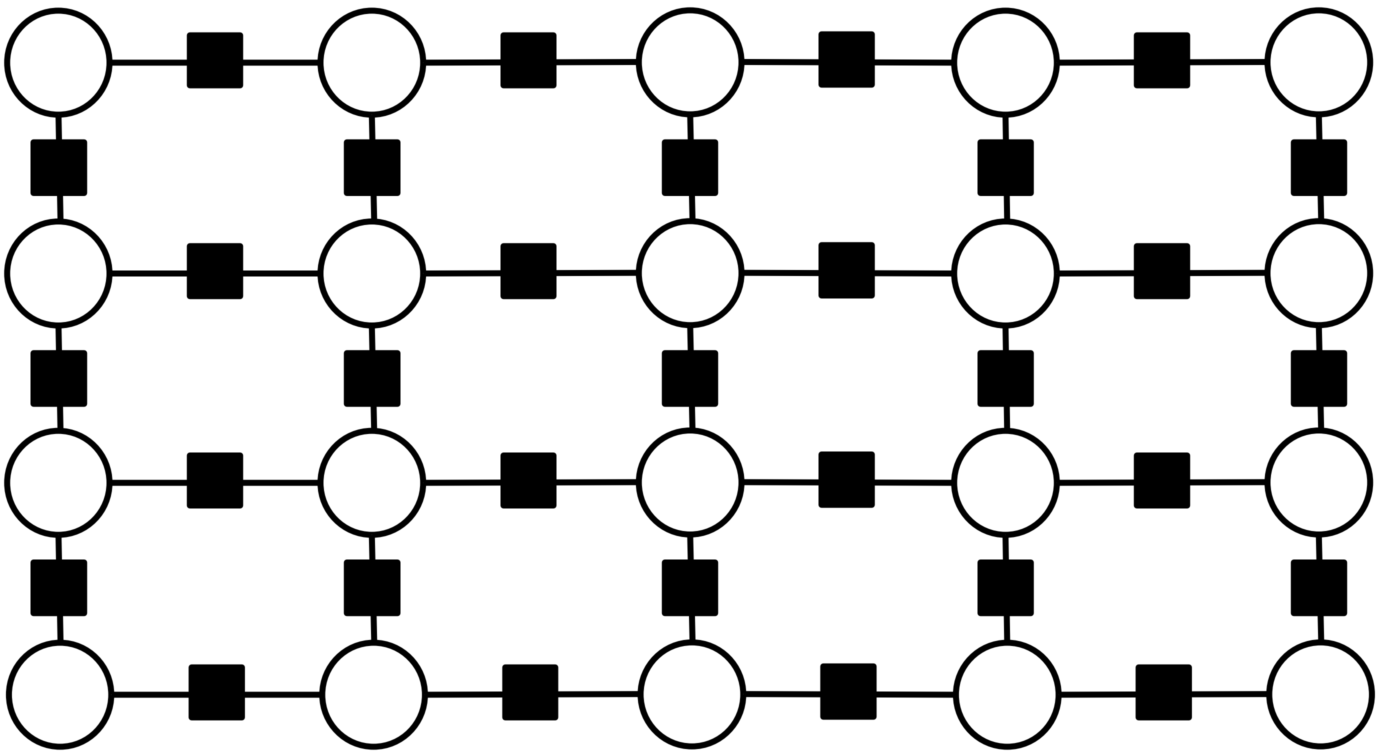
Locally-consistent joint optimal labeling

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

Y: labels
X: data (features)
S: pixels
N_i: neighbors of pixel i

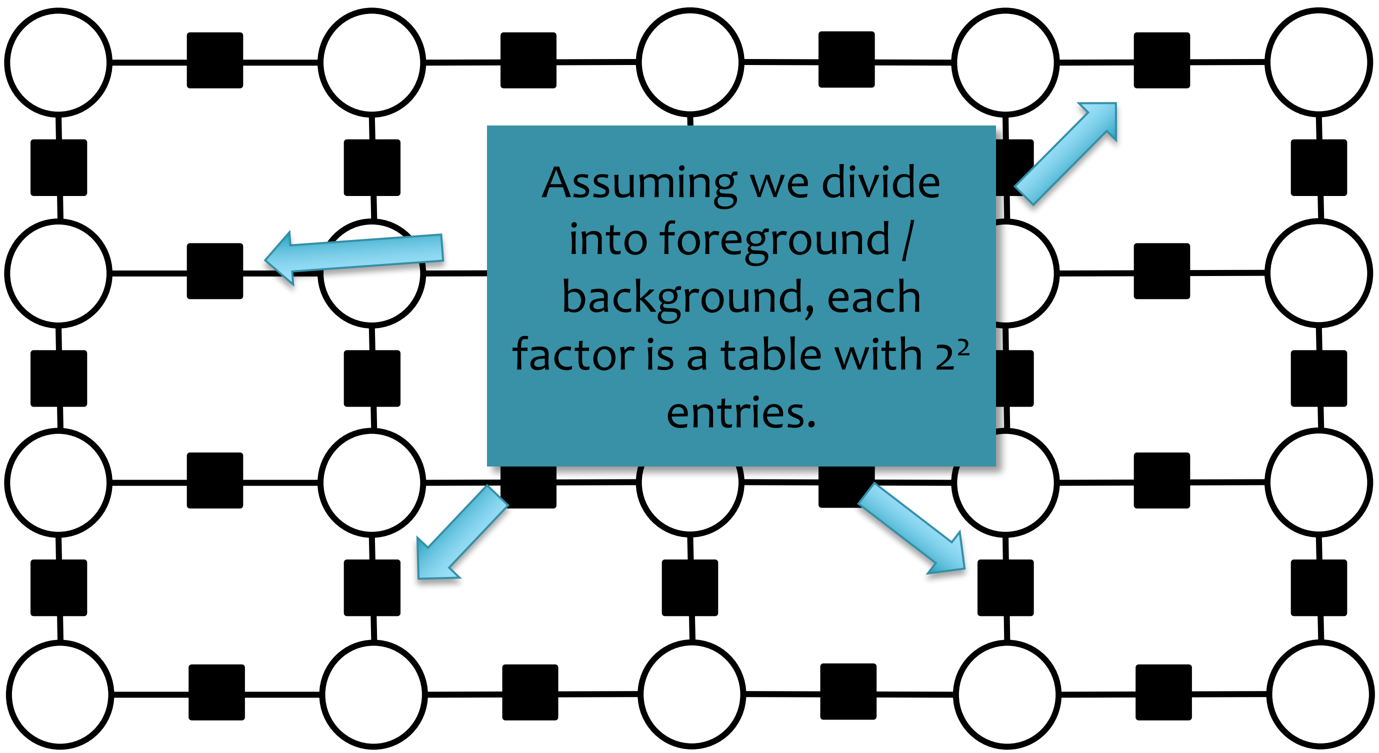
Grid CRF

- Suppose we want to image segmentation using a grid model



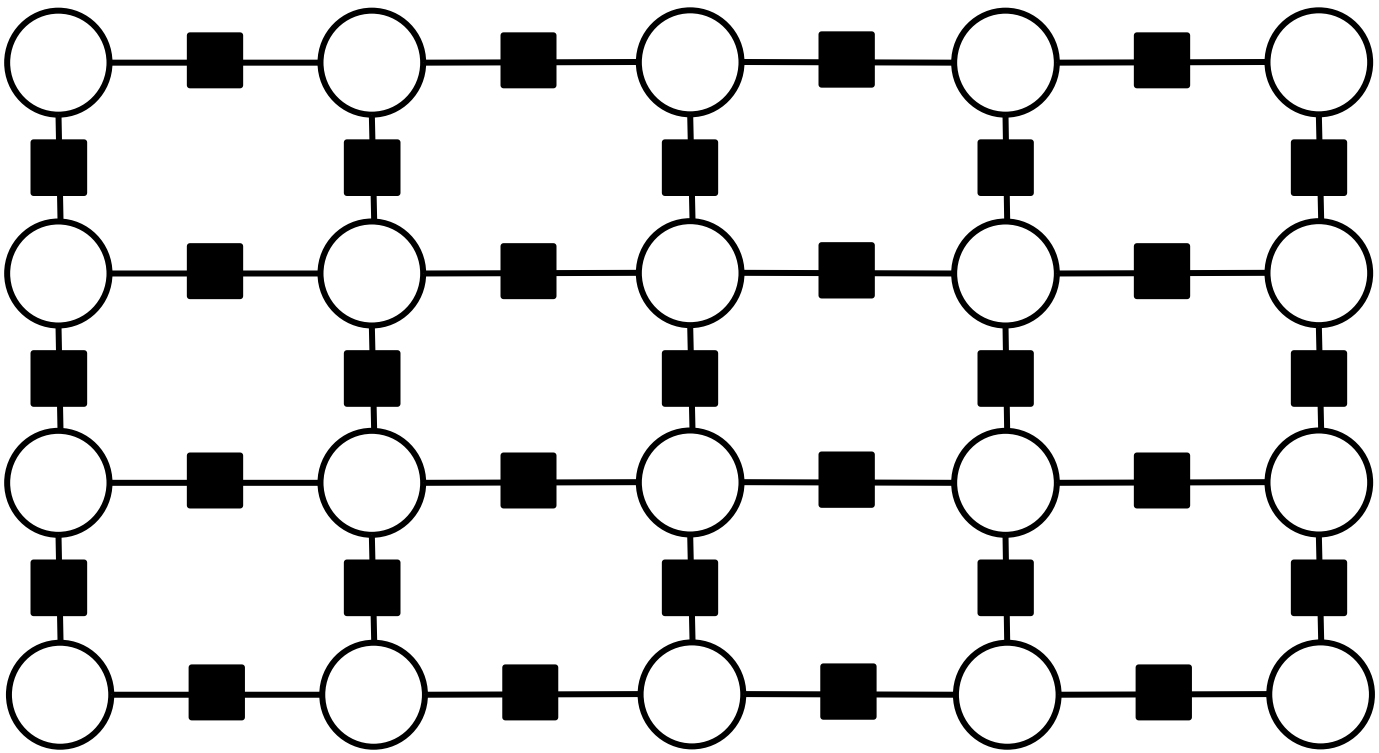
Grid CRF

- Suppose we want to image segmentation using a grid model



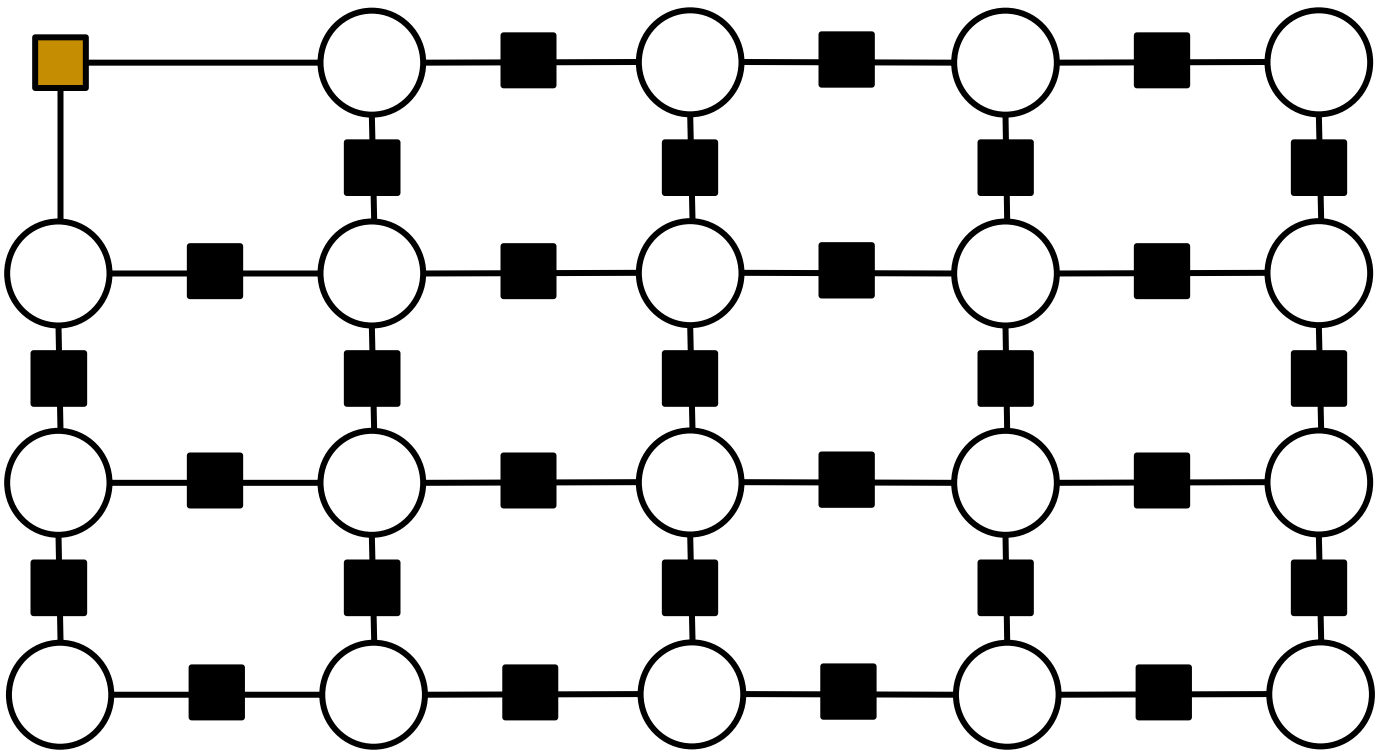
Grid CRF

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



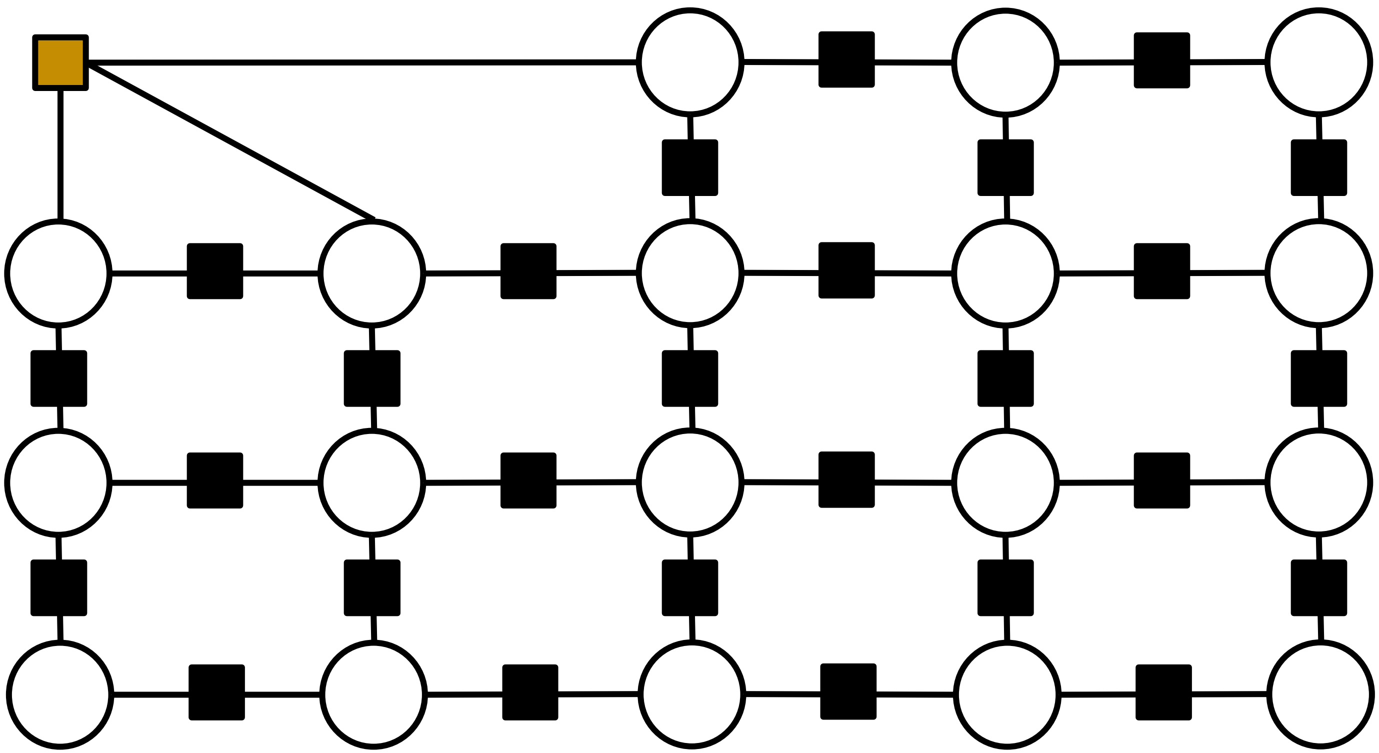
Grid CRF

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



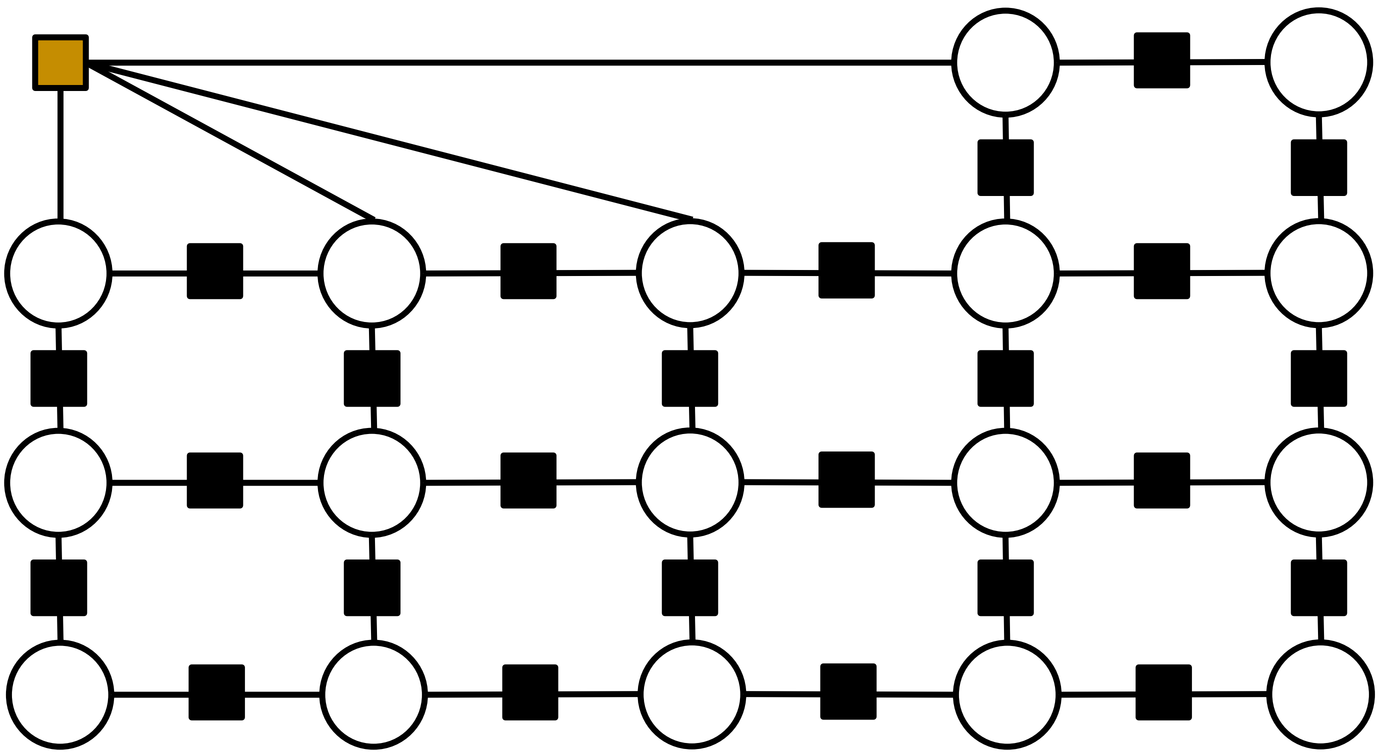
Grid CRF

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



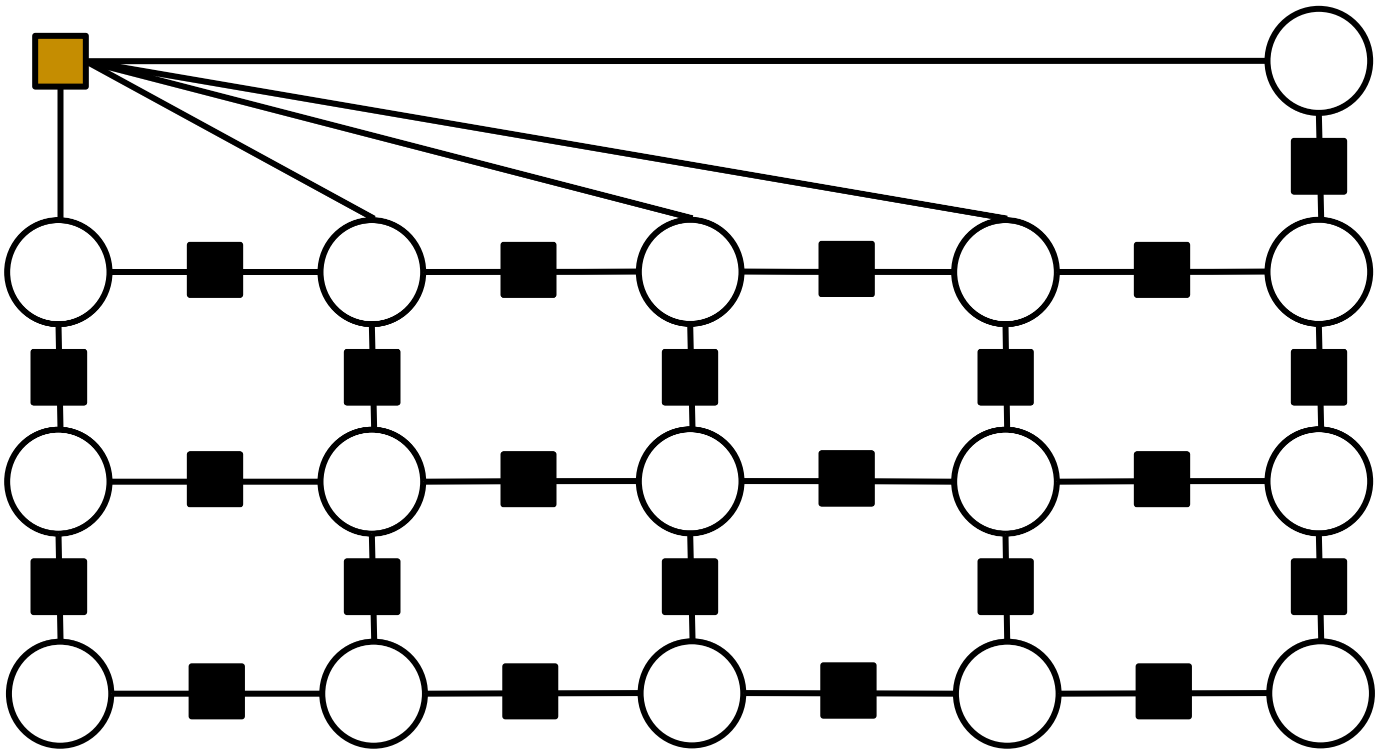
Grid CRF

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



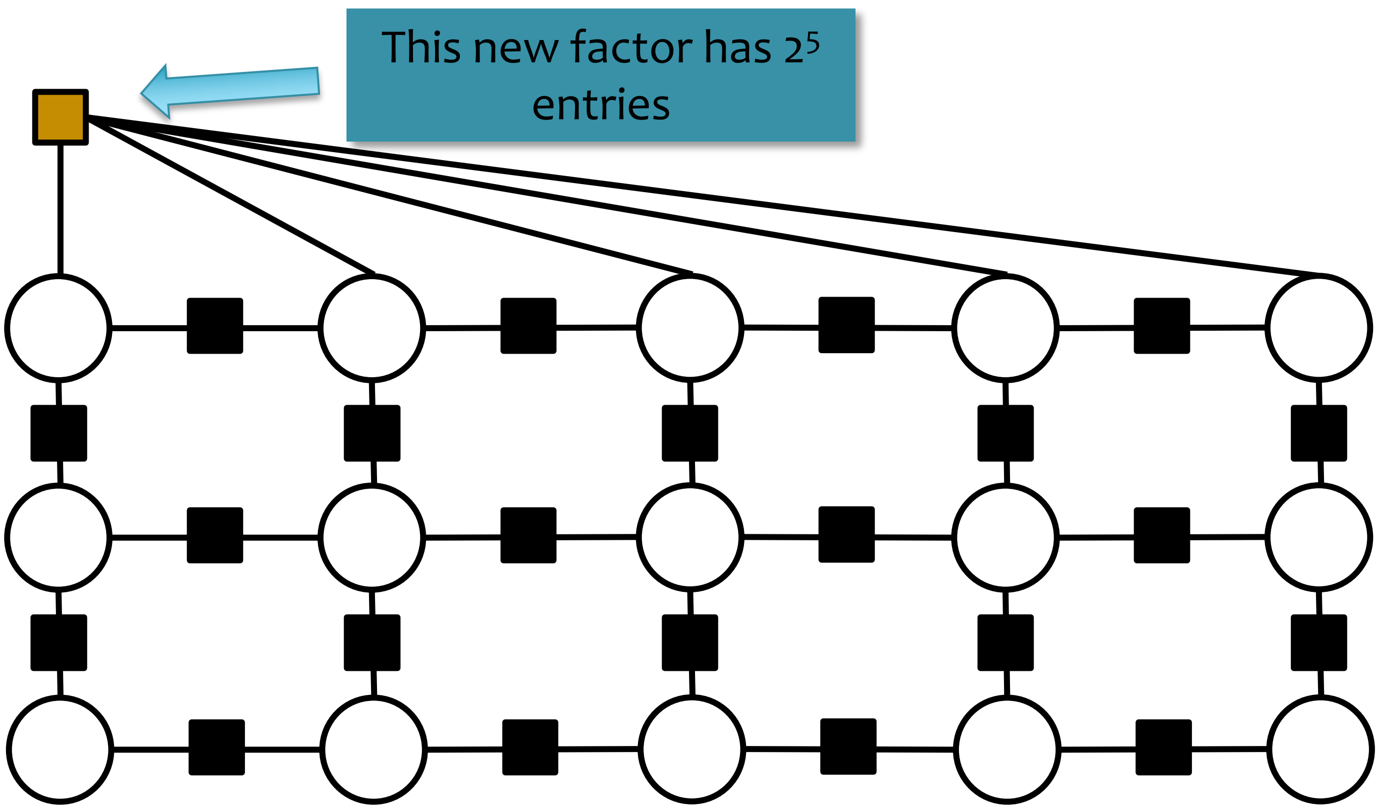
Grid CRF

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



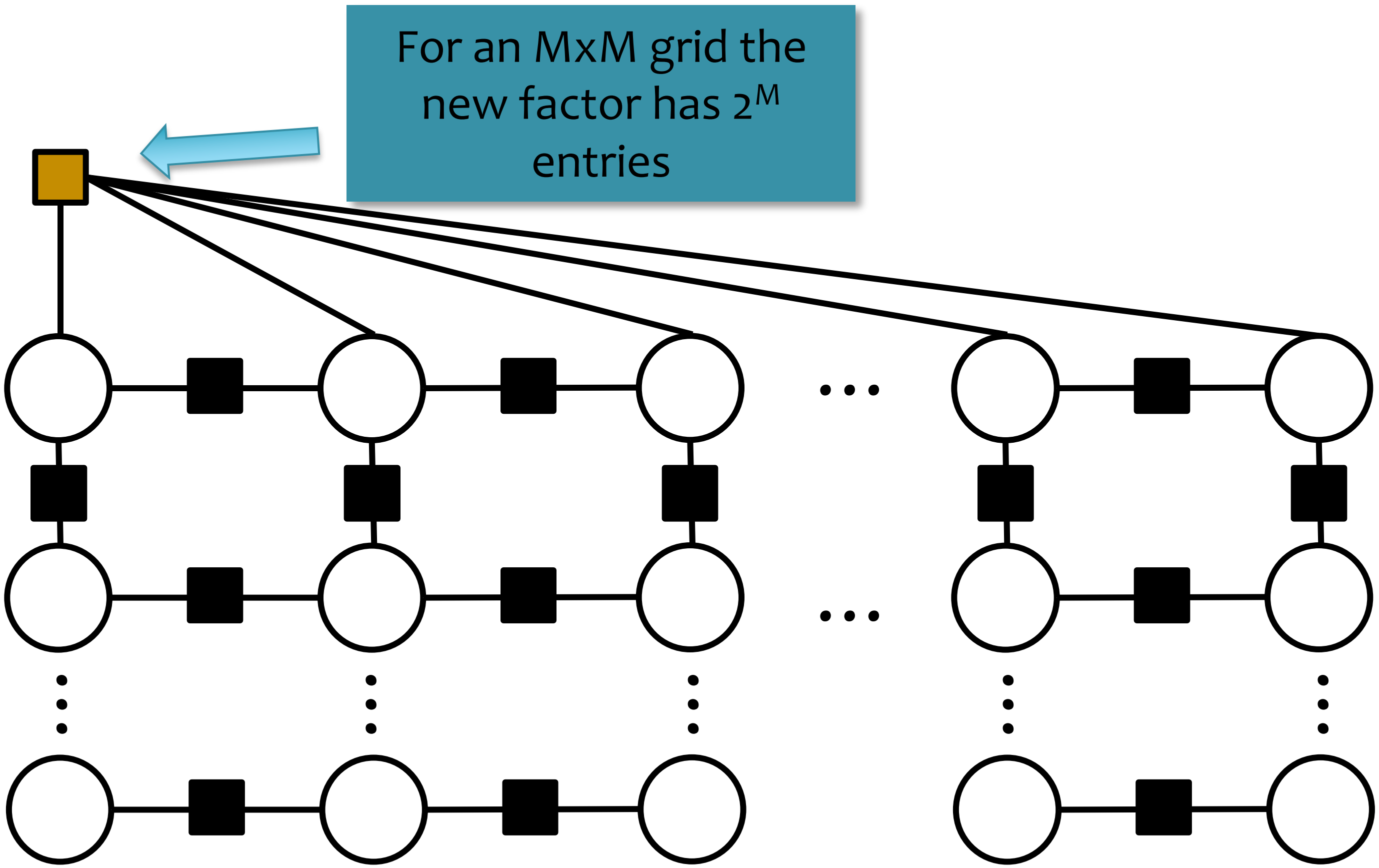
Grid CRF

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



Grid CRF

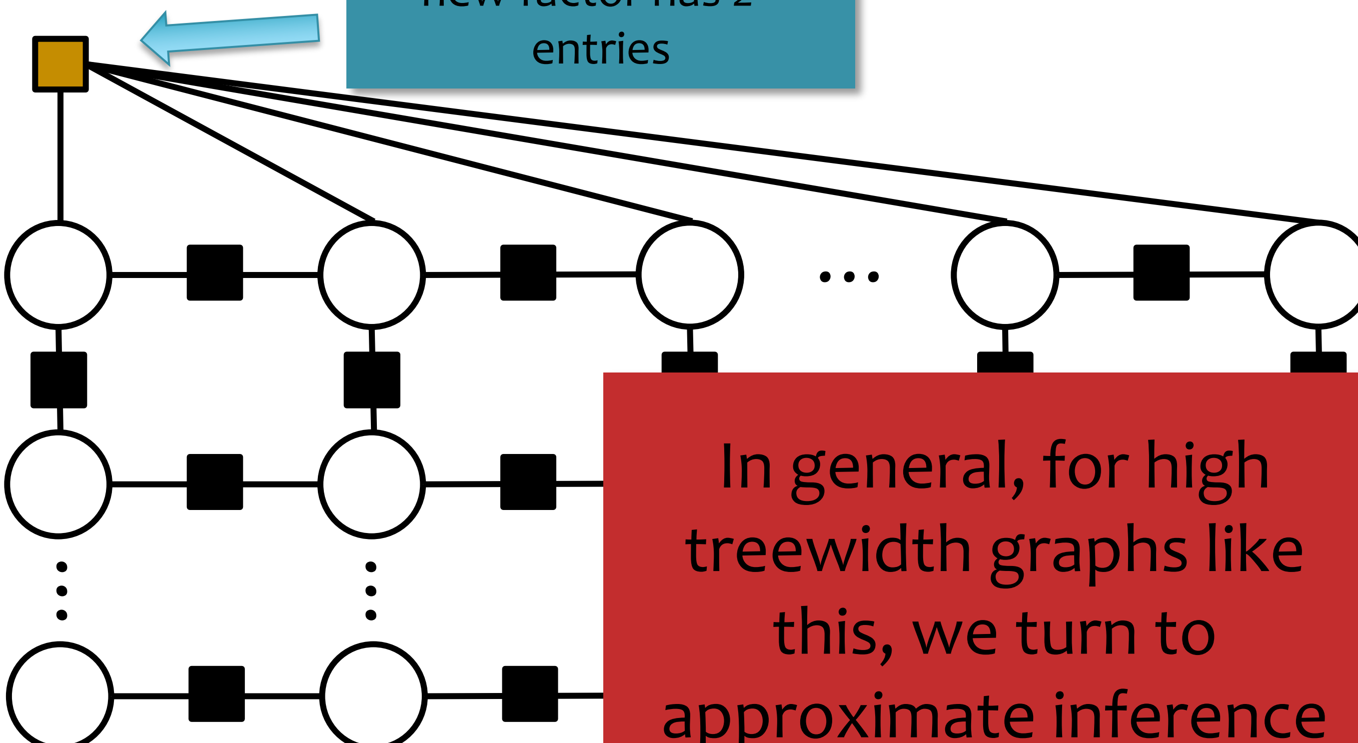
- Suppose we want to image segmentation using a grid model
- What happens when we run variable elimination?



Grid CRF

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

For an MxM grid the new factor has 2^M entries



In general, for high treewidth graphs like this, we turn to approximate inference (which we'll cover soon!)

Case Study: Object Recognition

Data consists of images $\boldsymbol{x}$ and labels y.

pigeon

rhinoceros

leopard

llama

Case Study: Object Recognition

Data consists of images x and labels y.

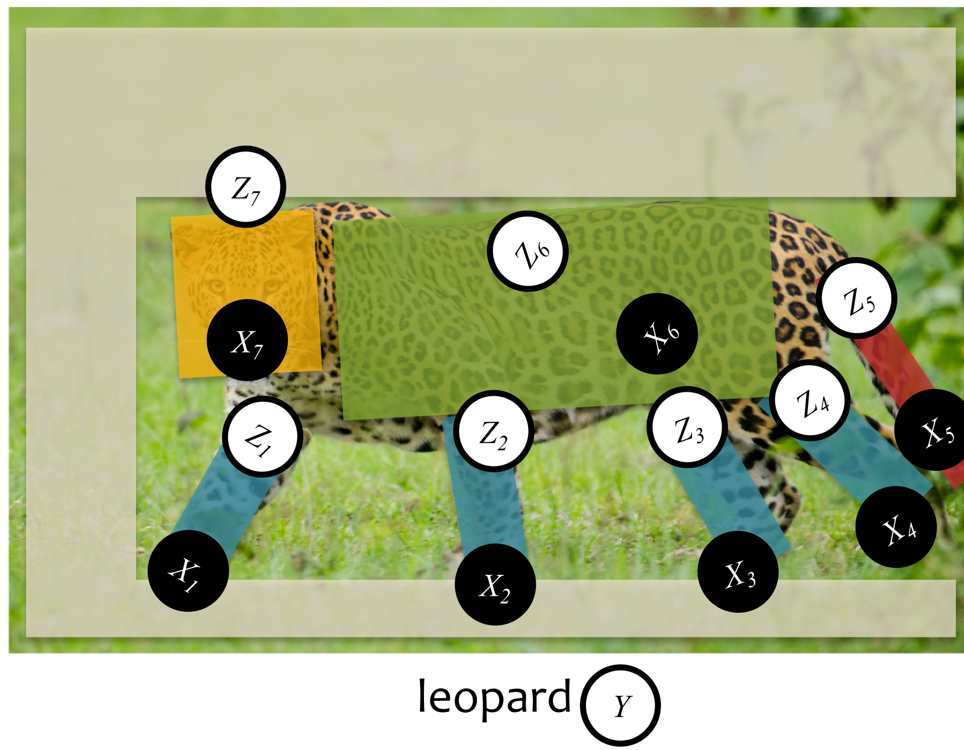
- Preprocess data into "patches"
- Posit a latent labeling z describing the object's parts (e.g. head, leg, tail, torso, grass)
- Define graphical model with these latent variables in mind
- z is not observed at train or test time

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

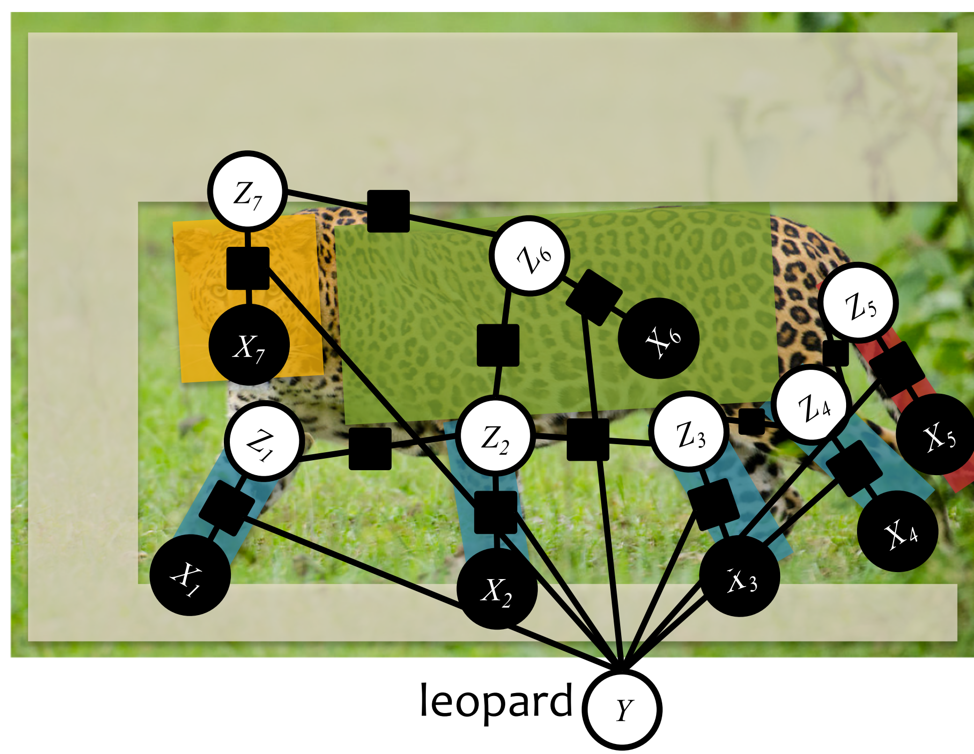


Case Study: Object Recognition

Data consists of images $\boldsymbol{x}$ 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

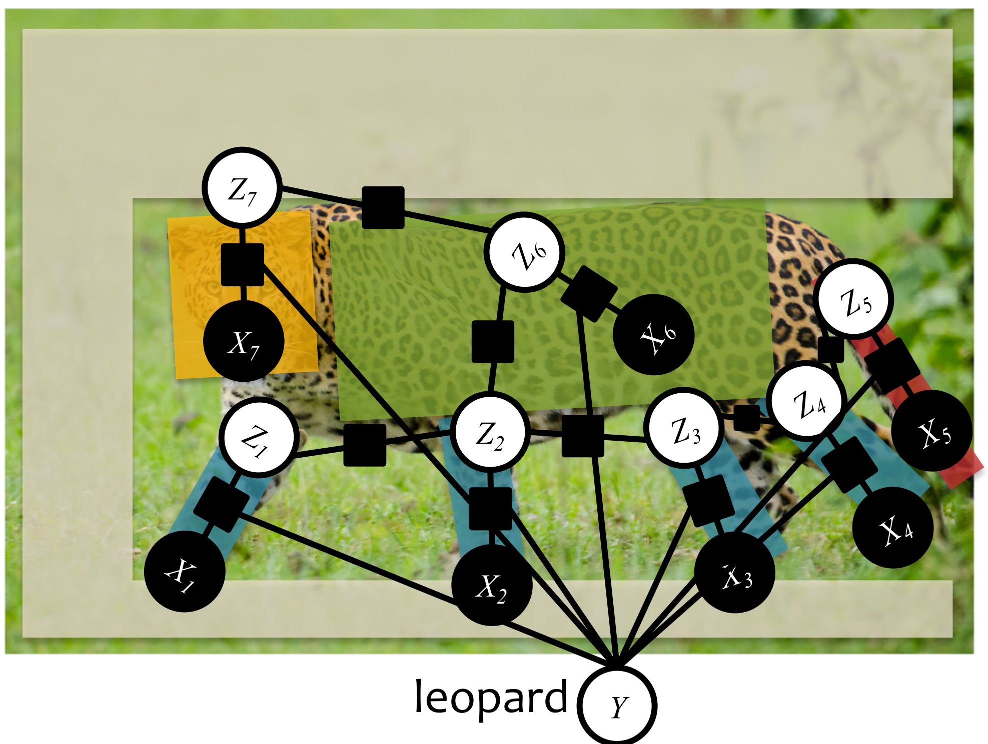


Hidden-state CRFs

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

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

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



Hidden-state CRFs

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

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

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

We can train using gradient based methods:

(the values $\boldsymbol{x}$ are omitted below for clarity)

$$\frac{d\ell(\boldsymbol{\theta}|\mathcal{D})}{d\boldsymbol{\theta}} = \sum_{n=1}^{N} \left(\mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{y}^{(n)})}[f_j(\boldsymbol{y}^{(n)}, \boldsymbol{z})] - \mathbb{E}_{\boldsymbol{y},\boldsymbol{z} \sim p_{\boldsymbol{\theta}}(\cdot,\cdot)}[f_j(\boldsymbol{y}, \boldsymbol{z})] \right)$$

$$= \sum_{n=1}^{N} \sum_{\alpha} \left(\sum_{\boldsymbol{z}_\alpha} \underbrace{p_{\boldsymbol{\theta}}(\boldsymbol{z}_\alpha \mid \boldsymbol{y}^{(n)})}_{\text{Inference on \textbf{clamped} factor graph}} f_{\alpha,j}(\boldsymbol{y}_\alpha^{(n)}, \boldsymbol{z}_\alpha) - \sum_{\boldsymbol{y}_\alpha, \boldsymbol{z}_\alpha} \underbrace{p_{\boldsymbol{\theta}}(\boldsymbol{y}_\alpha, \boldsymbol{z}_\alpha)}_{\text{Inference on \textbf{full} factor graph}} f_{\alpha,j}(\boldsymbol{y}_\alpha, \boldsymbol{z}_\alpha) \right)$$

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