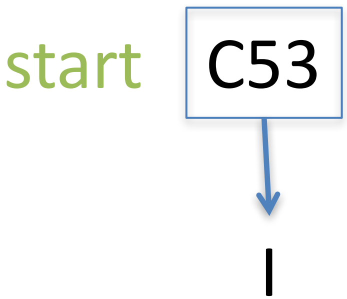
HMM Review

start

start C53


start
C53
I

start $\boxed{\text{C53}}$ C23

start C53 $\boxed{\text{C23}}$

I want

start C53 C23 C2

I want

start C53 C23 C2

I want a

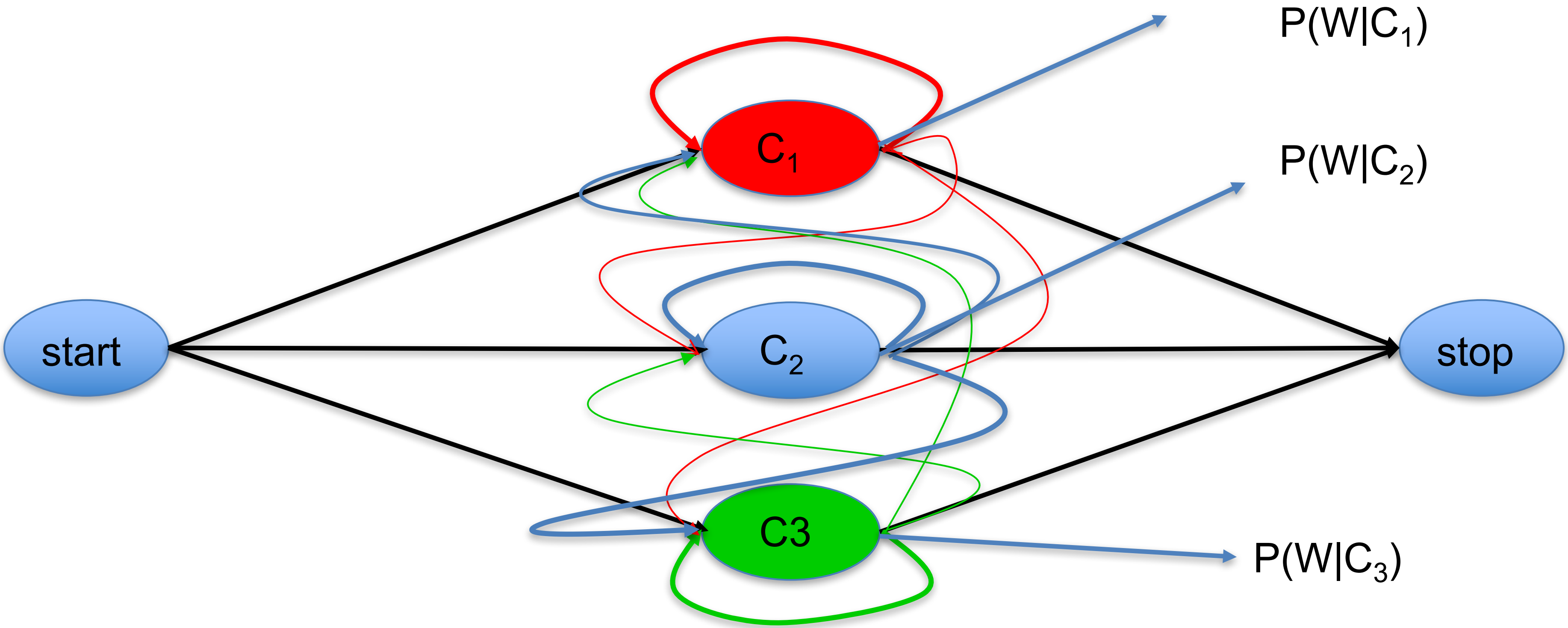
start C53 C23 $\boxed{\text{C2}}$ C5

I want a

start C53 C23 C2 $\boxed{\text{C5}}$

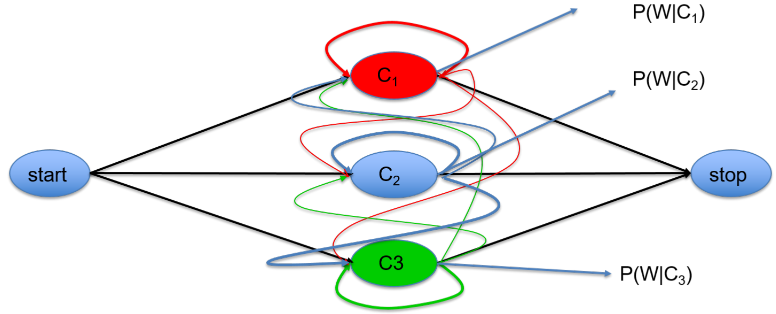
I want a flight

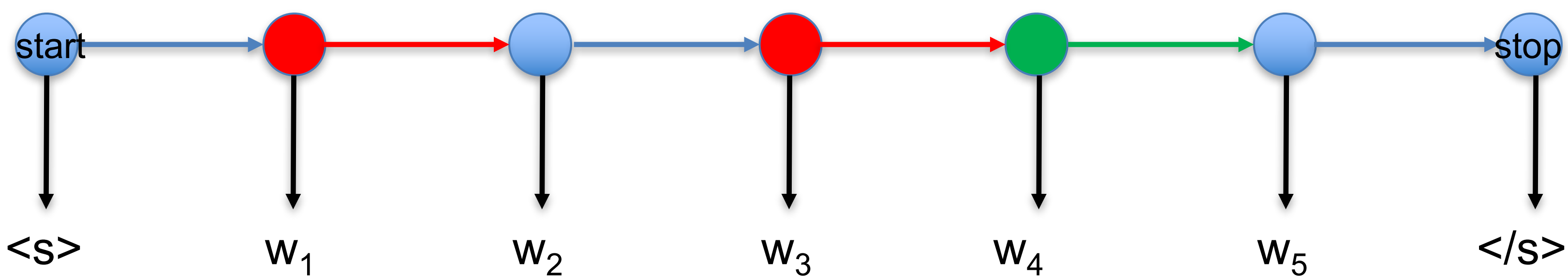
HMM as FSA



- Each time a class is visited, draw a word from the class

HMM

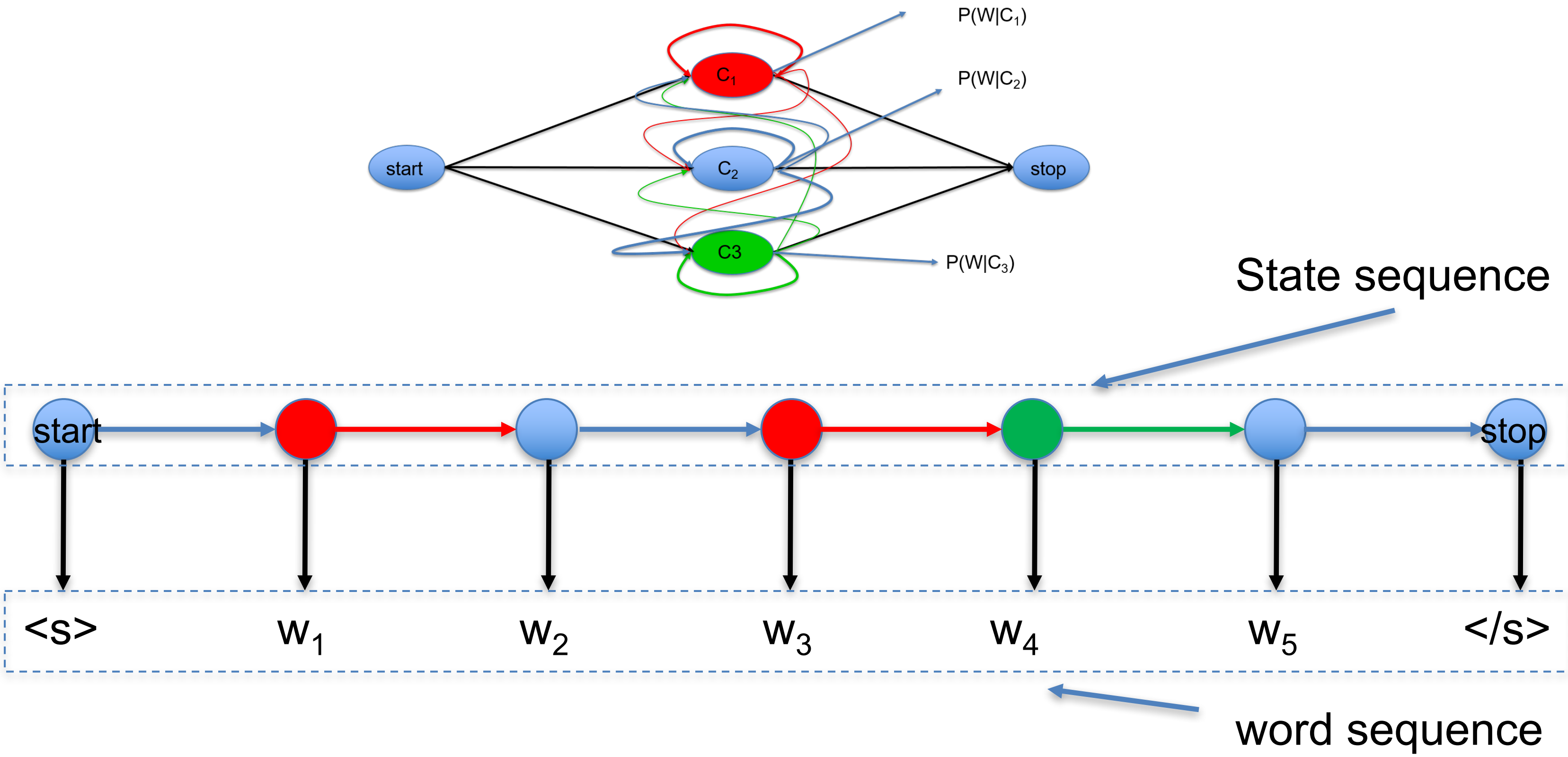




Parameters of the HMM

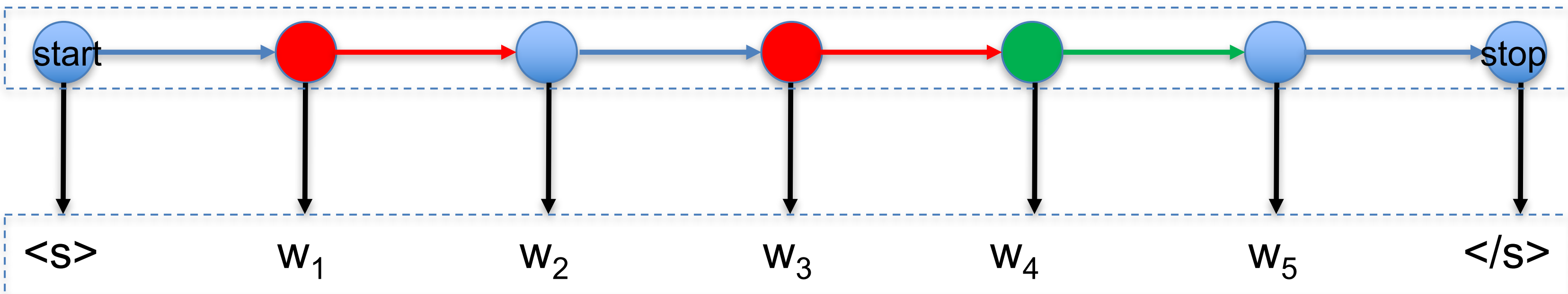
- The state transition probabilities of the underlying Markov chain on latent states
  - $P(S_i|S_j)$
- *Initial state probabilities*
  - What is the probability that *at the very first instant*, the process will be in state S_j
  - Often denoted by $\pi(S_j)$
- *Emission probabilities*
  - $P(w_i|s_j)$

HMM



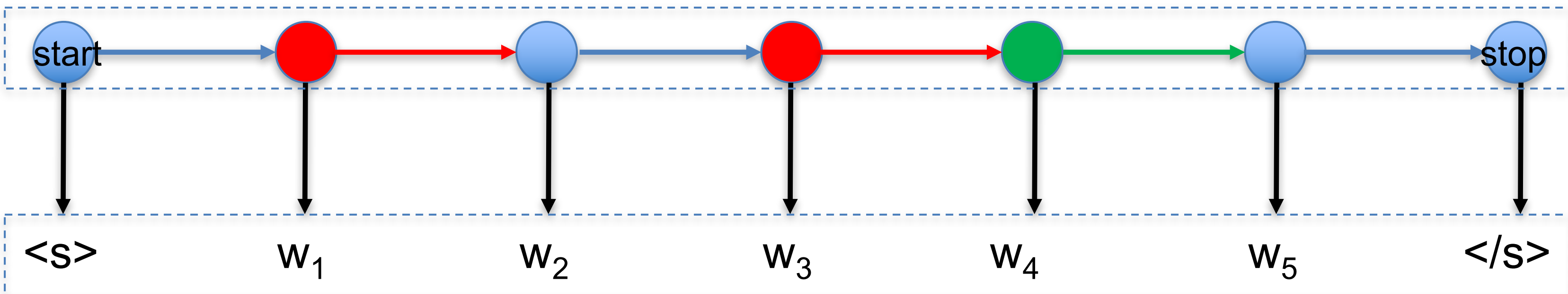
- We only observe the word sequence -- the state sequence is a latent variable

Decoding the state sequence



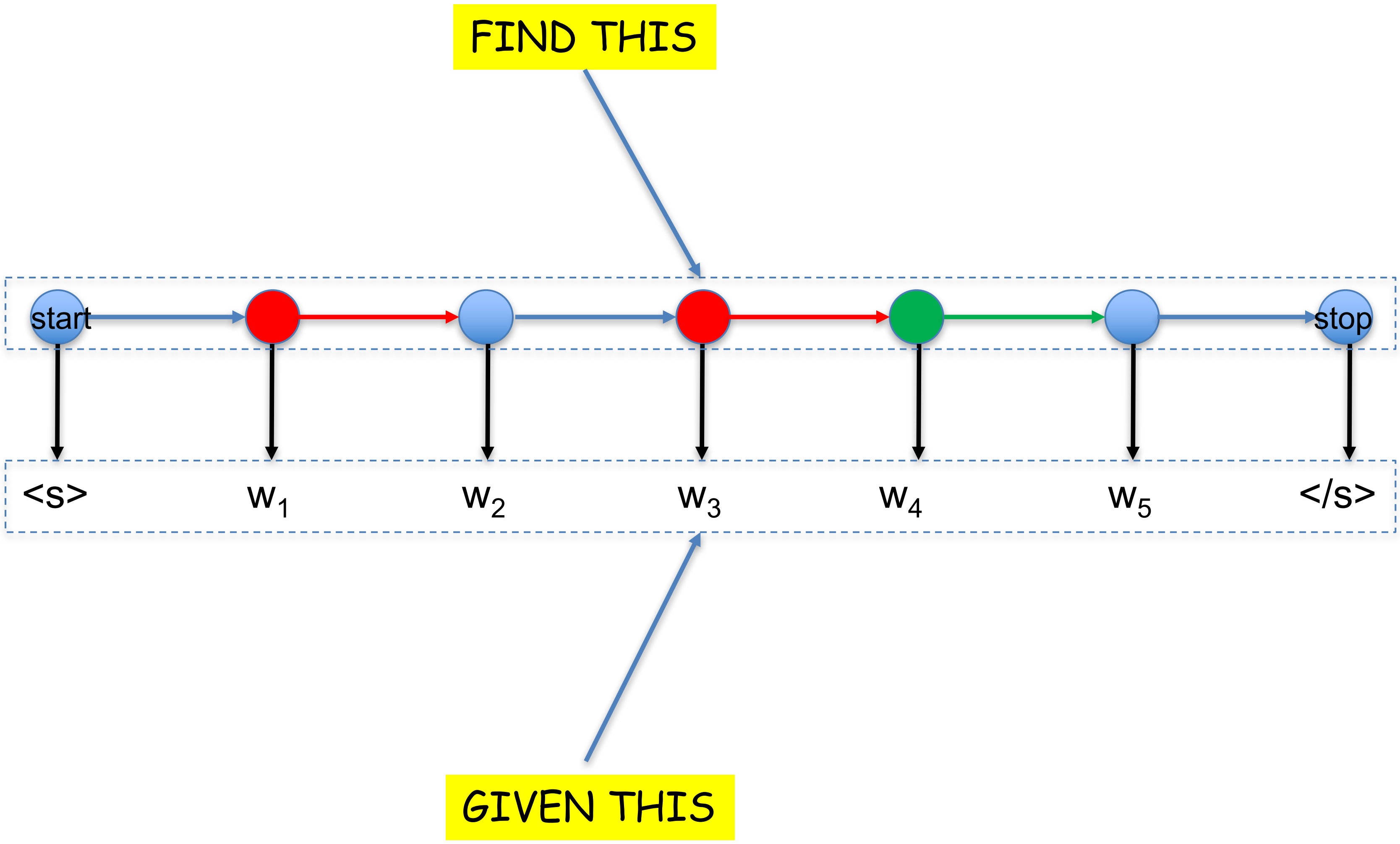
- *Preliminary: Given* all parameters of the HMM
  - Transition probabilities, initial state probabilities, emission probabilities
- *Problem: Given a word sequence <s> w_1 w_2… </s>, find the underlying state sequence*

Decoding the state sequence

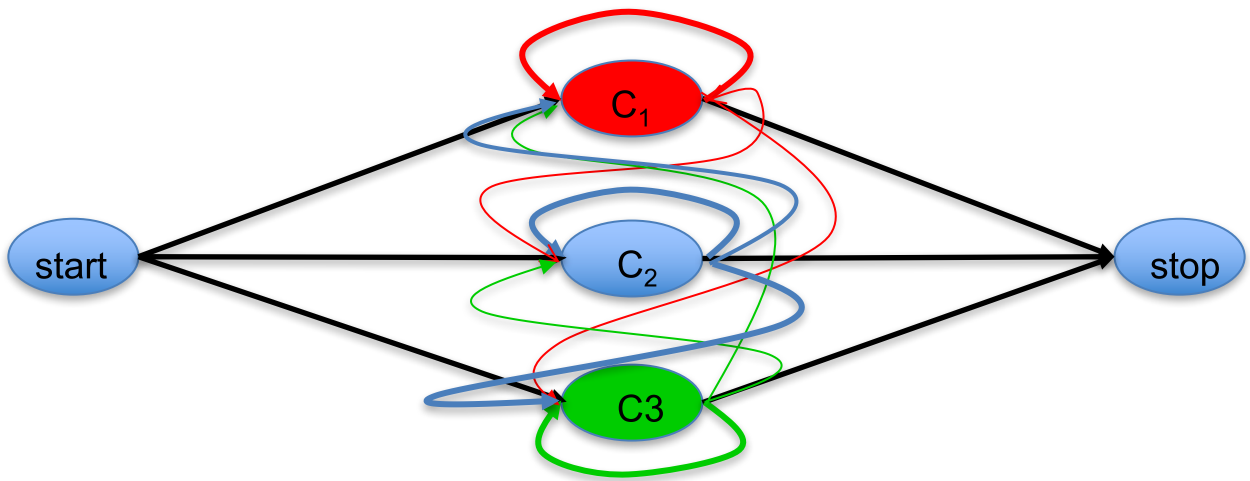


- *Preliminary: Given* all parameters of the HMM
  - Transition probabilities, initial state probabilities, emission probabilities
- *Problem: Given a word sequence <s> w_1 w_2… </s>, find the state sequence with highest probability*

Decoding the state sequence

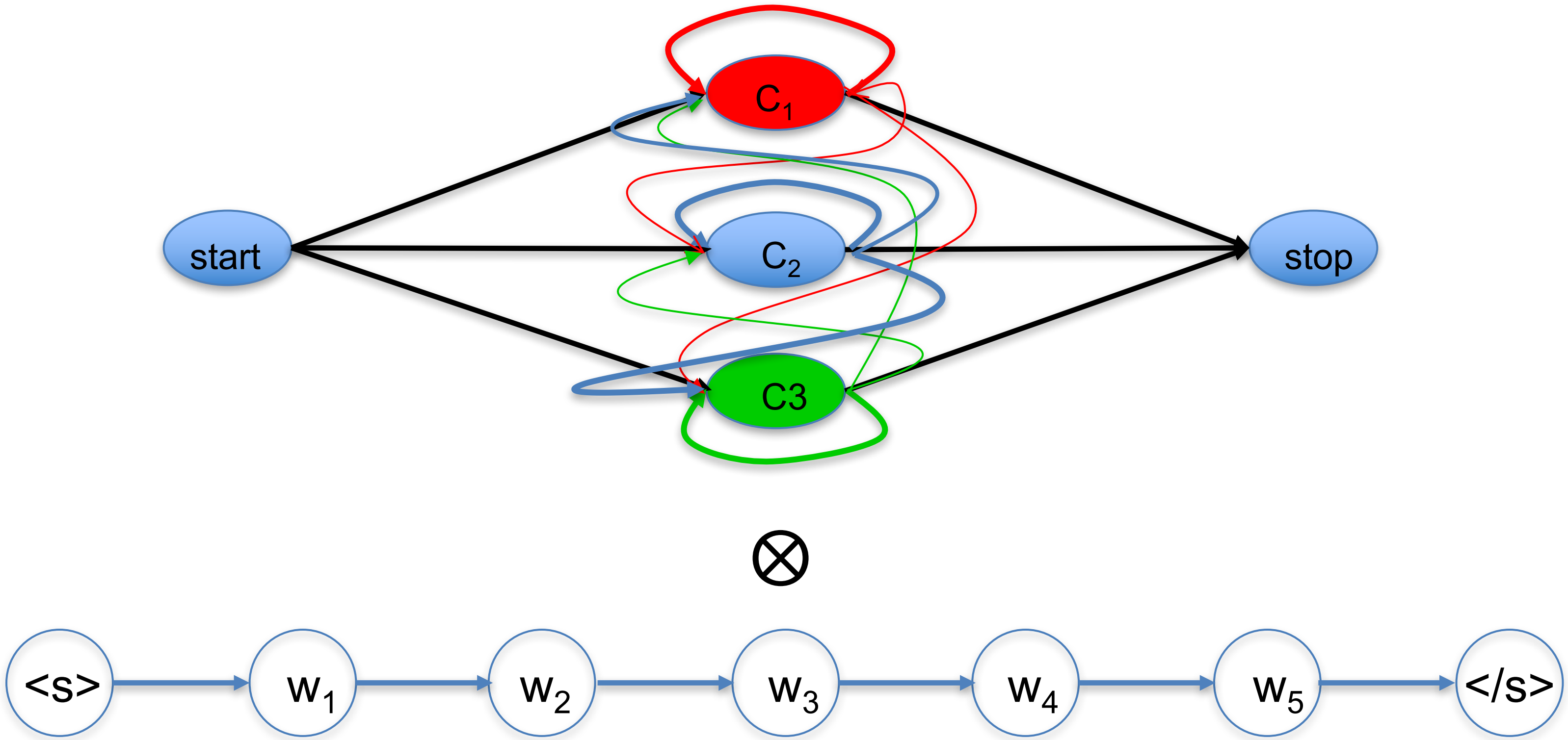


The graph view of the problem



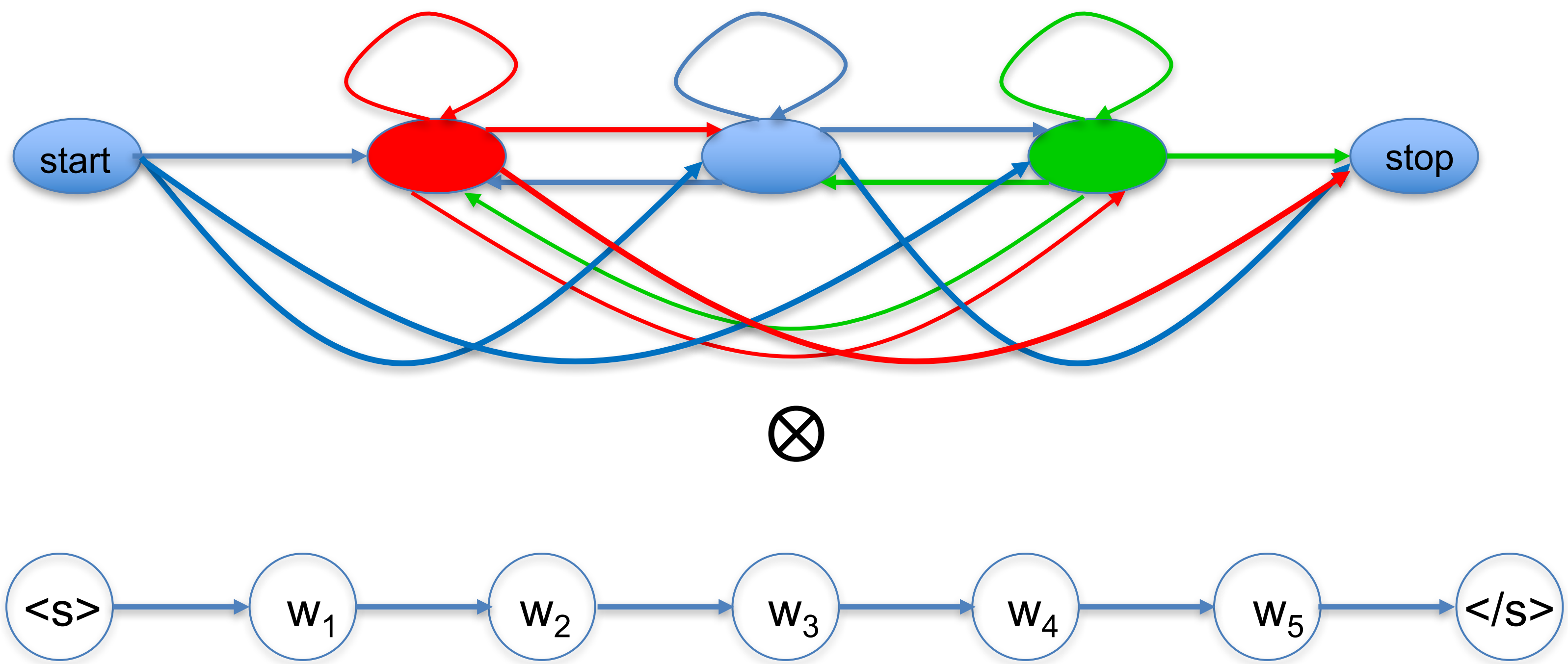
- Any valid state sequence *must* conform to the transition structure imposed by the Markov model
  - It *must* be a valid path through the Markov graph (i.e. no zero prob transitions) and it will be scored by the Markov model's probs
- At the same time, the *productions* from the state sequence must conform to the structure of the observation
  - i.e. w_i *must* be followed by w_{i+1} with probability 1 and each emission will be scored with the corresponding emission prob

The graph view of the problem



- The set of all combination of states and words can be represented as a combined graph that conforms to the restrictions of *both* graphs
  - i.e. the composition of both graphs, which is a trellis..

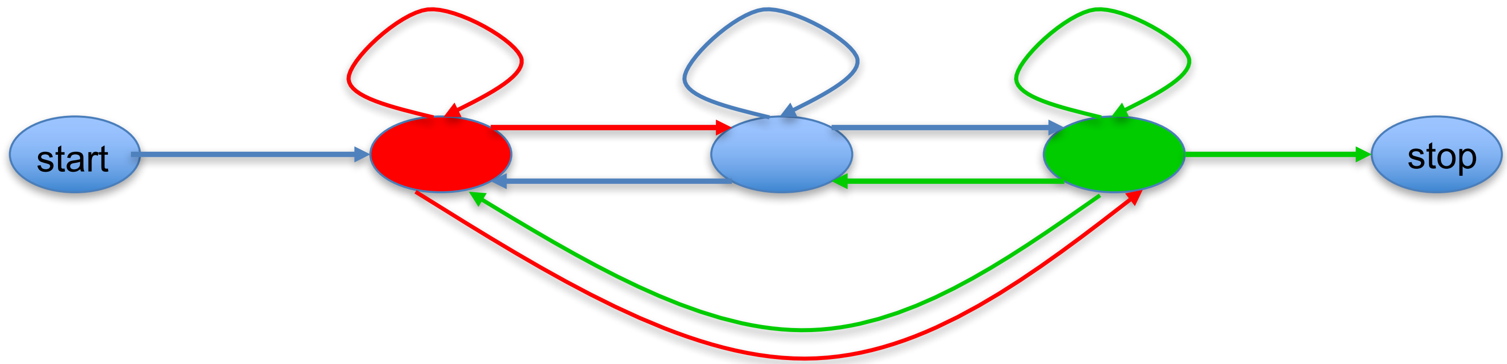
The graph view of the problem



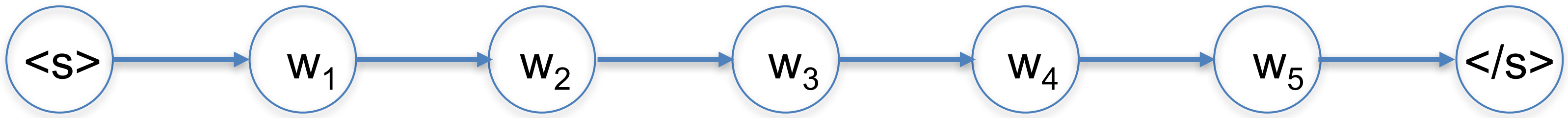
- The set of all combination of states and words can be represented as a combined graph that conforms to the restrictions of *both* graphs
  - I.e. the composition of both graphs, which is a trellis..

The graph view of the problem

Assuming a simpler model for clarity of illustration (first word *must* be from red state, last word *must* be from green state)

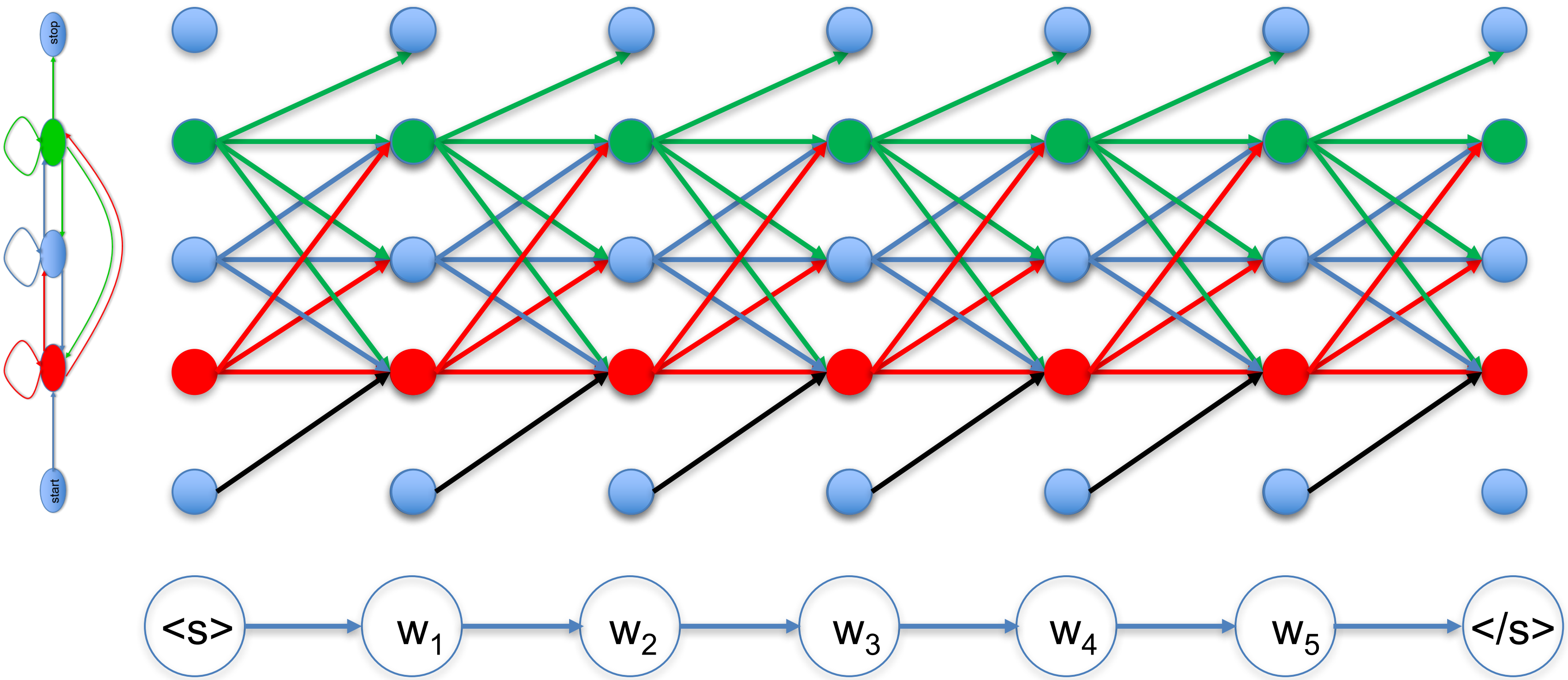


$\otimes$



- The set of all combination of states and words can be represented as a combined graph that conforms to the restrictions of *both* graphs
  - I.e. the composition of both graphs, which is a trellis..

The graph view of the problem

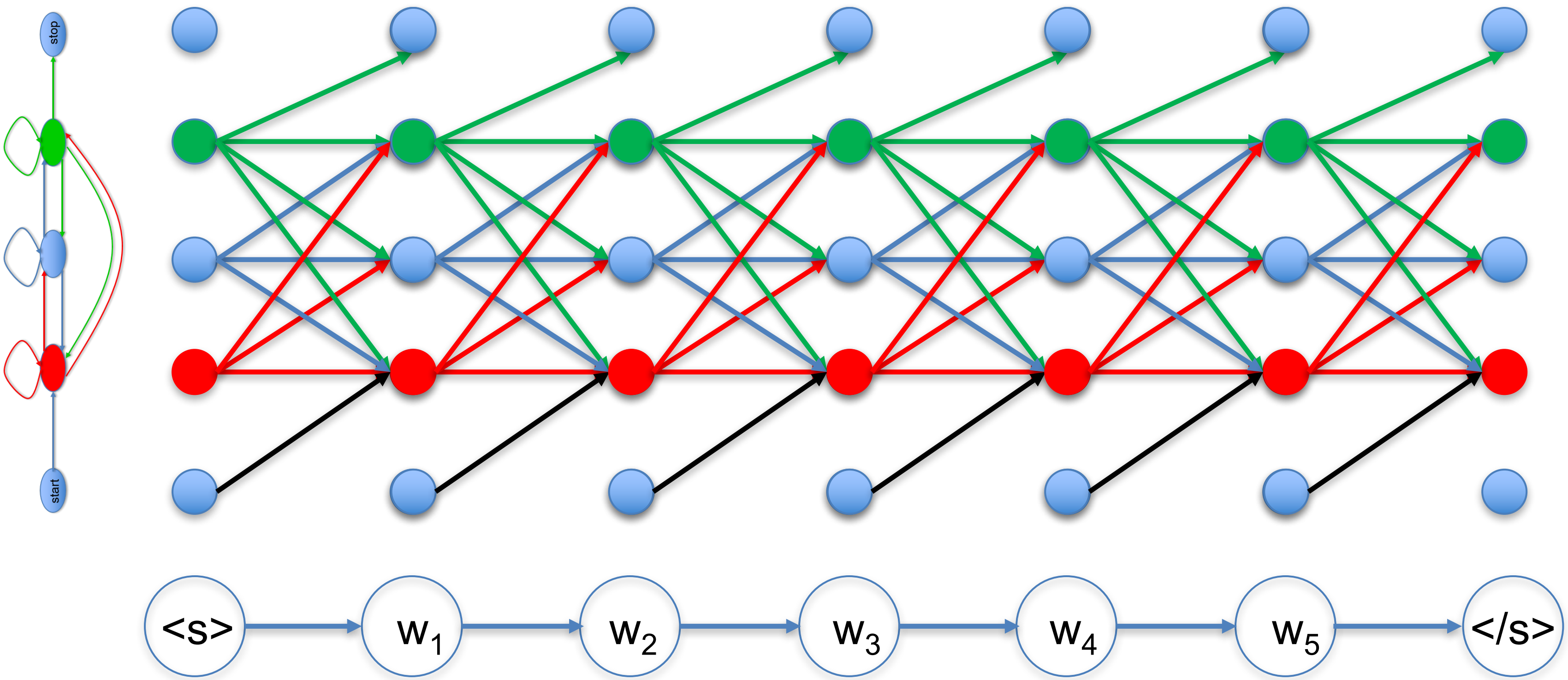


- The Trellis that composes the state graph and the observation graph
- Every state sequence through this trellis conforms to both, the Markov graph over states and the linear ordering of words

Probabilities on the Trellis

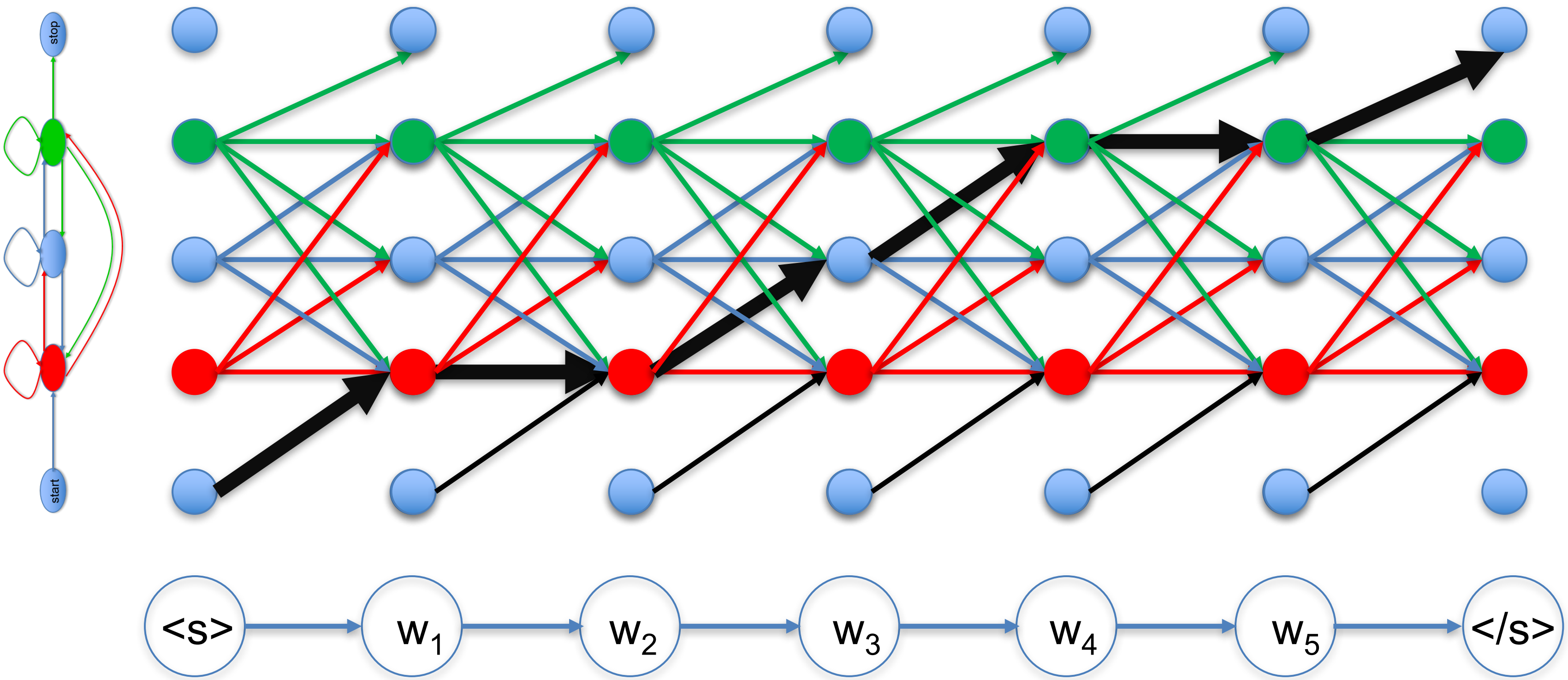
- The "score" for combining a state and a word is the probability of emitting that word from the state
- The "score" for an edge is the product of the probabilities associated with edges in both graphs
- The "score" for a path through the trellis is now obtained by *multiplying* component node and edge probabilities

The graph view of the problem



- Trellis: $NodeScore(s, w) \; = \; P(w|s)$
- Trellis $Edgescore(s_i, s_j) = \; P(s_j|s_i)$

The graph view of the problem



- As defined, the score associated with a path in the trellis is its joint prob under the generative model:

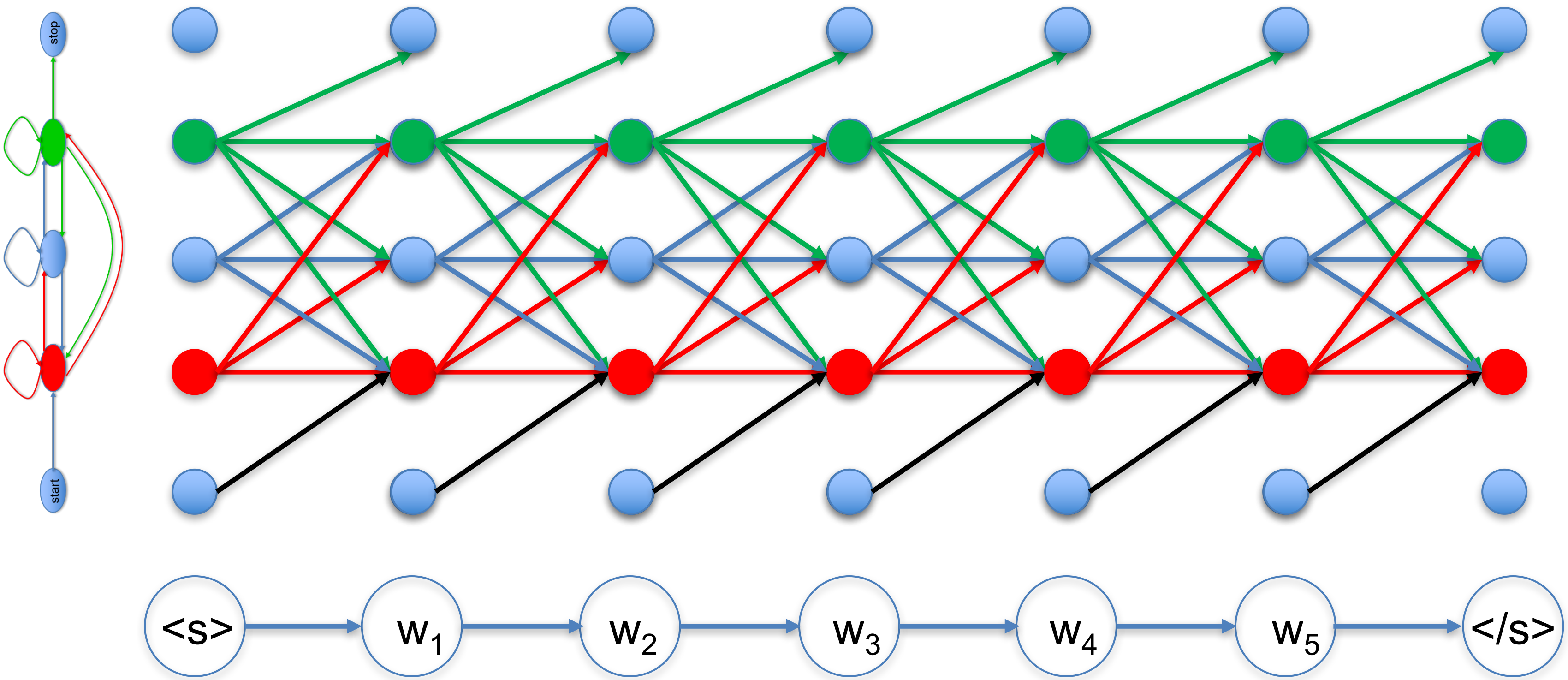
$$P(start, < s >, s_1, w_1, s_2, w_2, \dots, stop, </s >)$$

Probabilities on the Trellis

- Instead of probabilities, we will often work with *log* probabilities (this is one way of dealing with underflow)

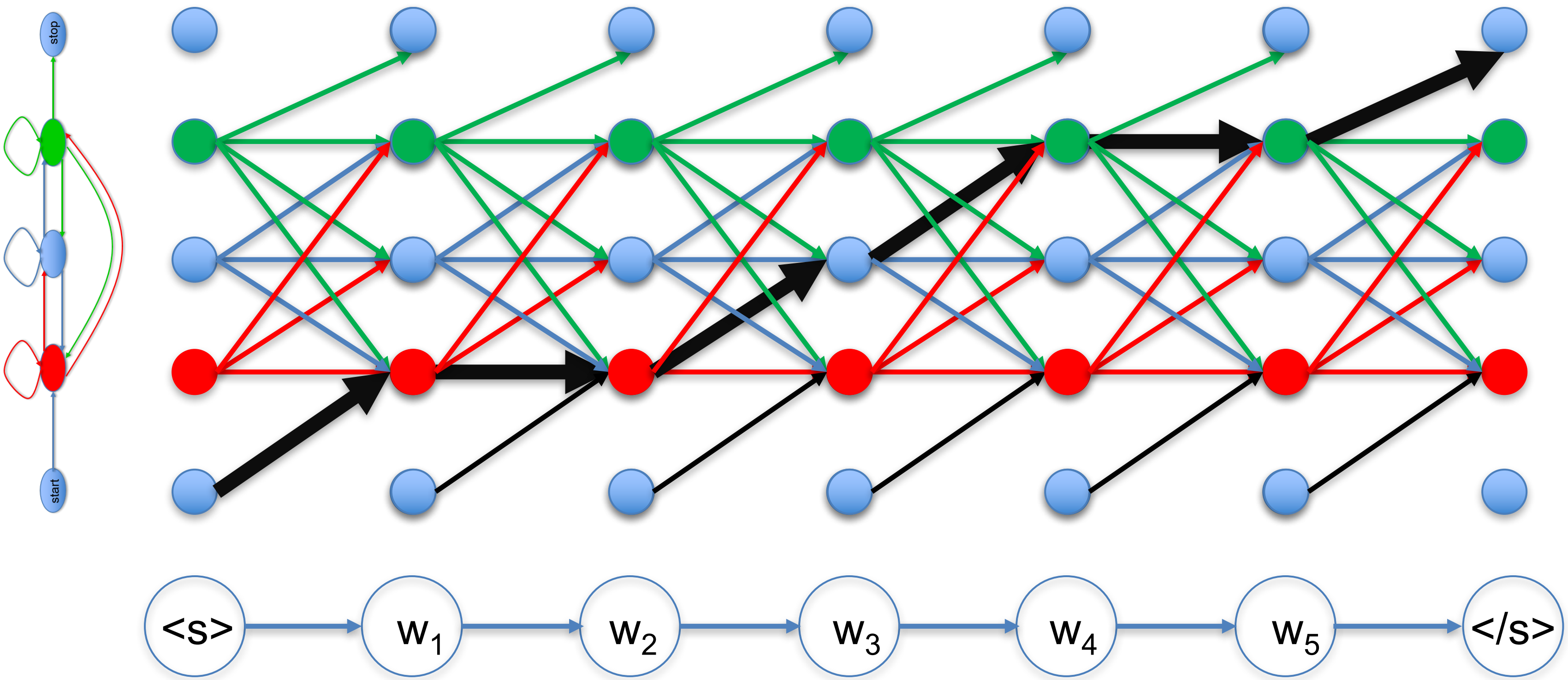
- So…. instead of multiplying components along the paths, we add them

The graph view of the problem



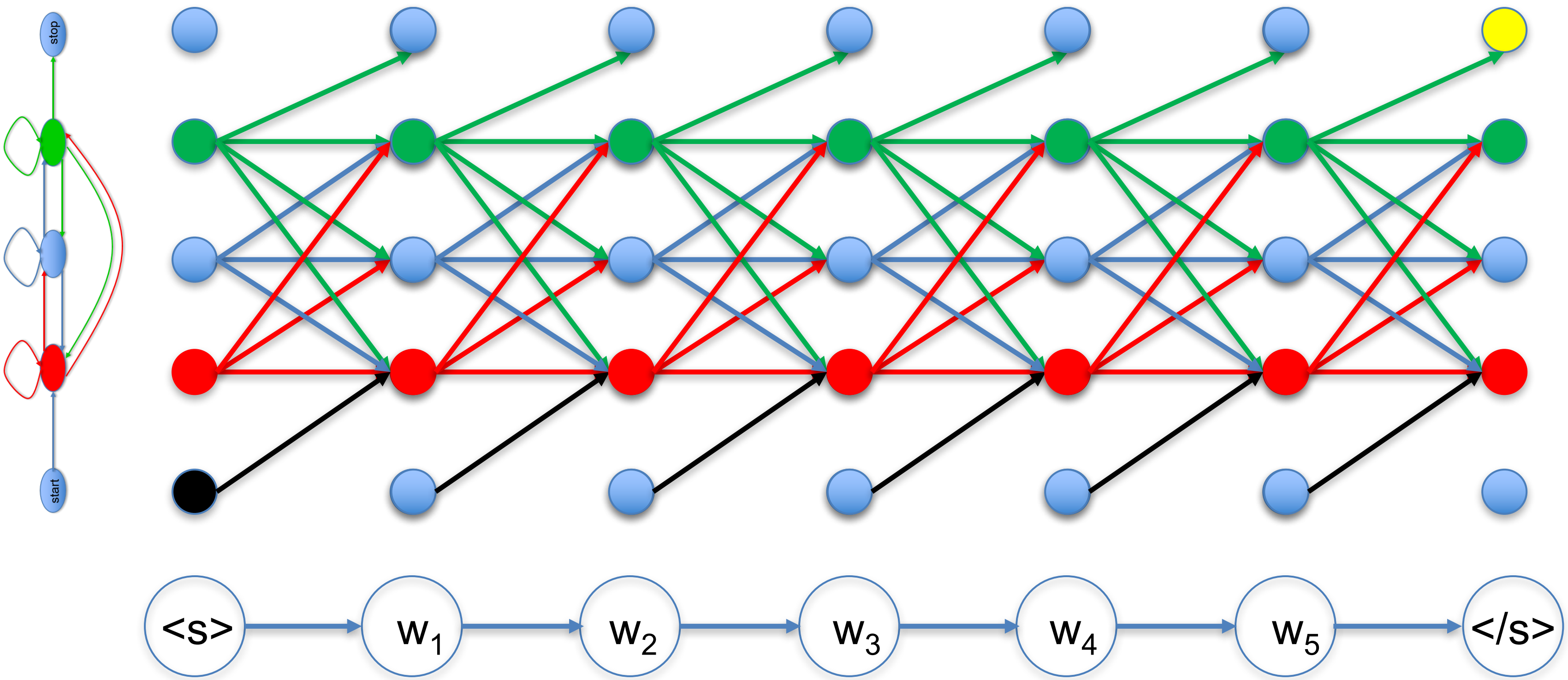
- Trellis: $NodeScore(s, w) = \log P(w|s)$
- Trellis $Edgescore(s_i, s_j) = \log P(s_j|s_i)$

The graph view of the problem



- Path score = $\log P(start, < s >, s_1, w_1, s_2, w_2, \ldots, stop, </s >)$

Finding the state sequence



- Problem: Find the most probable state sequence given the word sequence
- Equivalent problem: Find the highest scoring path from the start (black) node to the final (yellow) node
- For this we can now use the Viterbi algorithm

Viterbi algorithm

- Initialize:

```
    Score[1:M, 1:N] = -infty
    Bestpredecessor[1:M, 1:N] = null
```

- Algorithm:

```
Score[1,1] = nodescore(node(1,1))
for i = 2:M
    for j = 1:N
        BP = argmax_k(Score[i-1,k] + edgescore((i-1,k),(i,j)))
        Score[i,j] = Score[i-1,BP] + edgescore((i-1,BP),(i,j))
                                   + nodescore(i,j)
        Bestpredecessor[i,j] = BP
```

- Final overall cost:

```
BestScore = Score[M,N]
```

- Actual sequence of states (from parent 1):

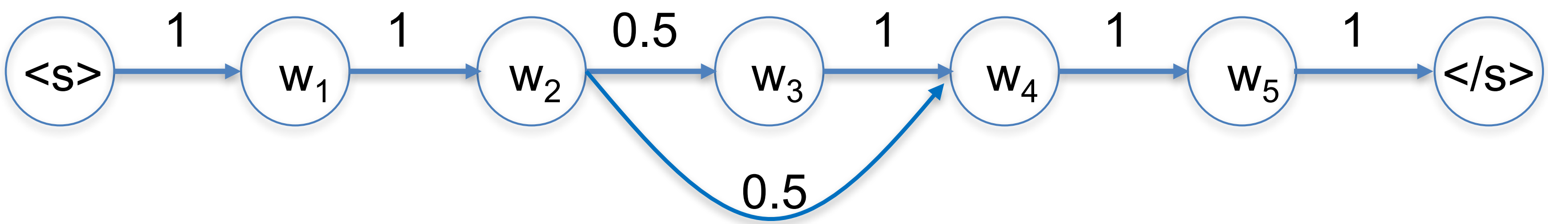
```
State[M] = N
for i = M downto 2
    State[i-1] = Bestpredecessor(i, State[i])
```

Generalizing the approach

<s> w_1 w_2 [w_3] w_4 w_5 </s>

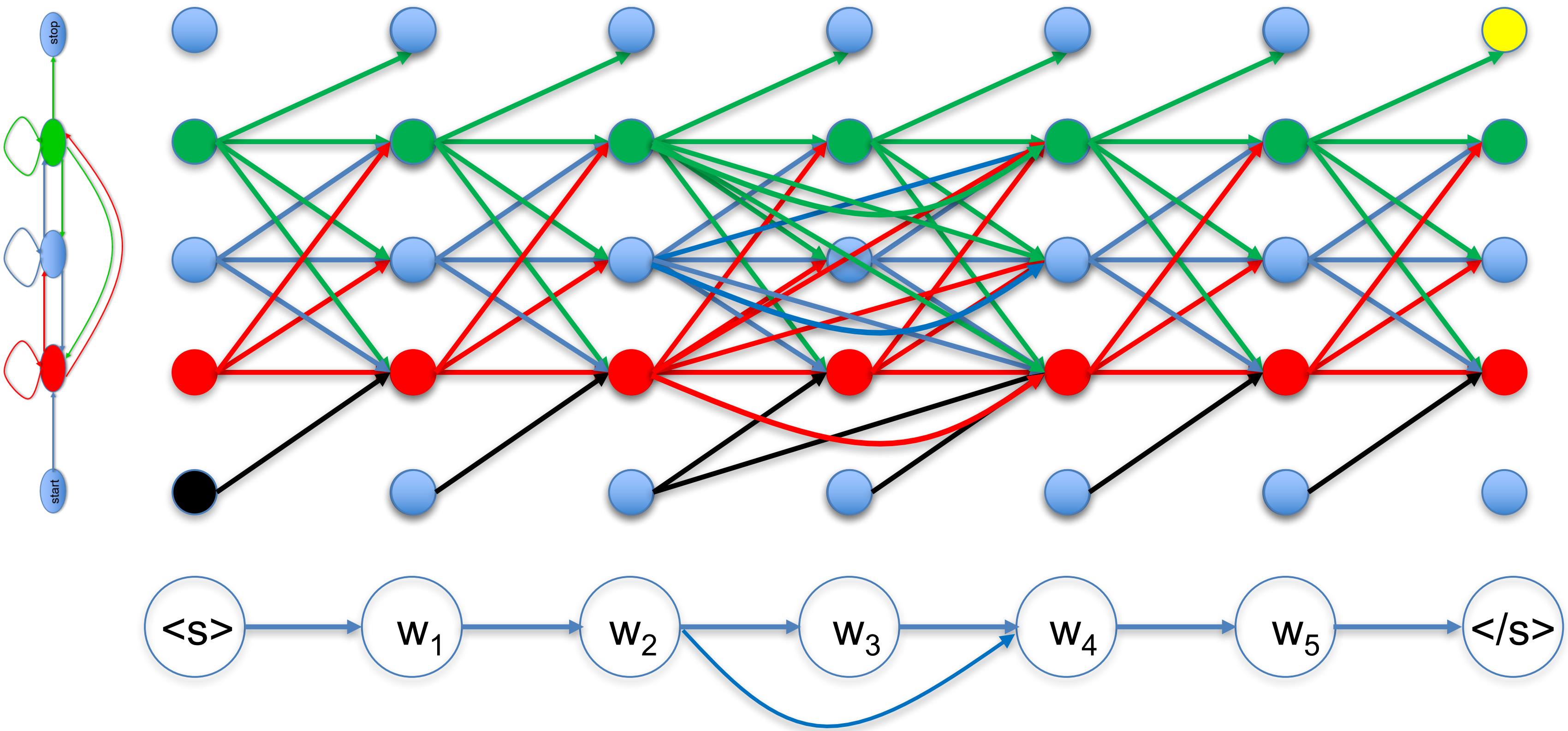
- Consider the case where the observed word sequence is uncertain
  - Uncertain whether w_3 was said or not
  - But the presence or absence of w_3 changes the interpretation of the sentence
  - How to find the most likely state sequence

The uncertain observation graph



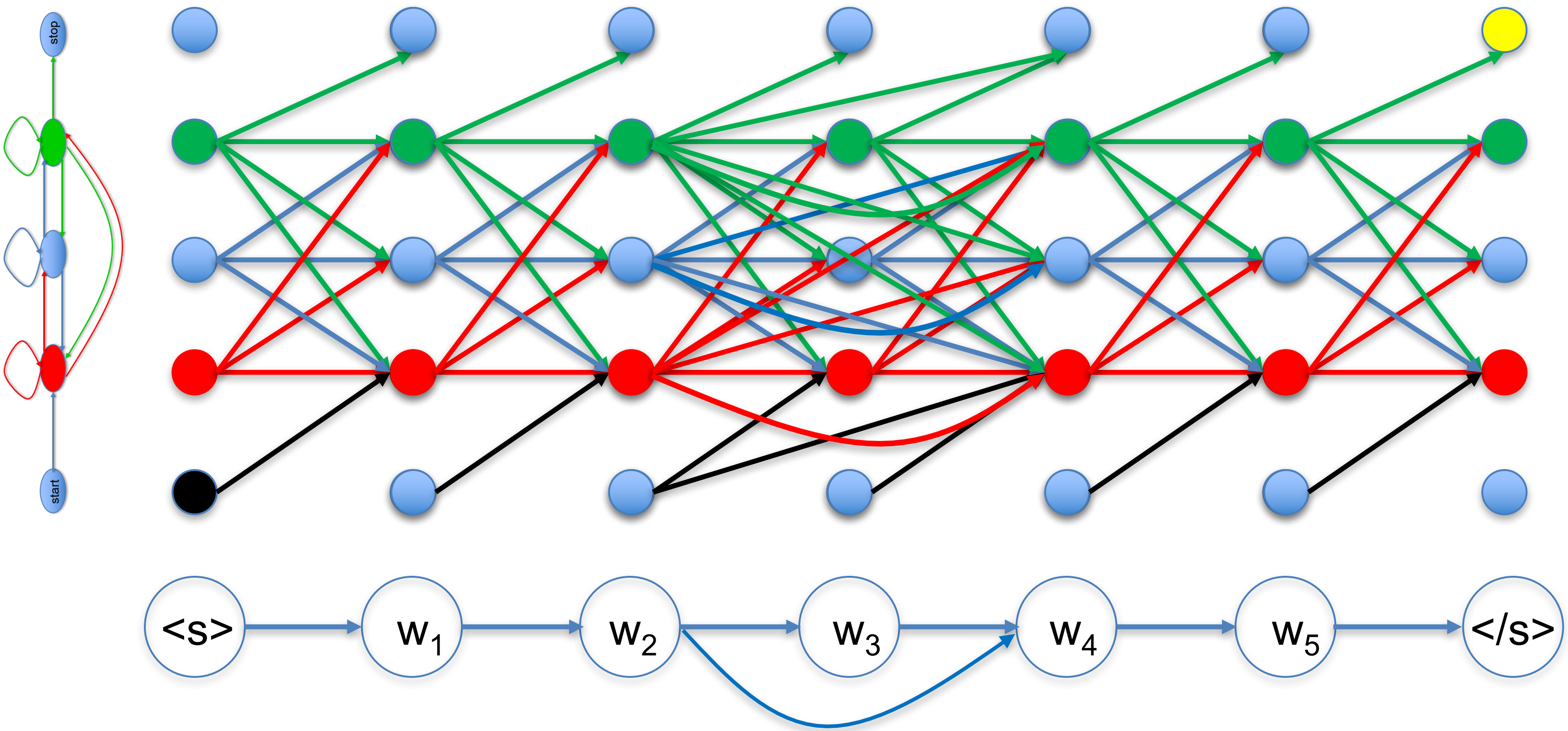
- The observation sequence can now be modeled by this modified graph
  - Note the probabilities
    - The 0.5 may be replaced by any other value indicative of our certainty in the occurrence of the word

The modified trellis



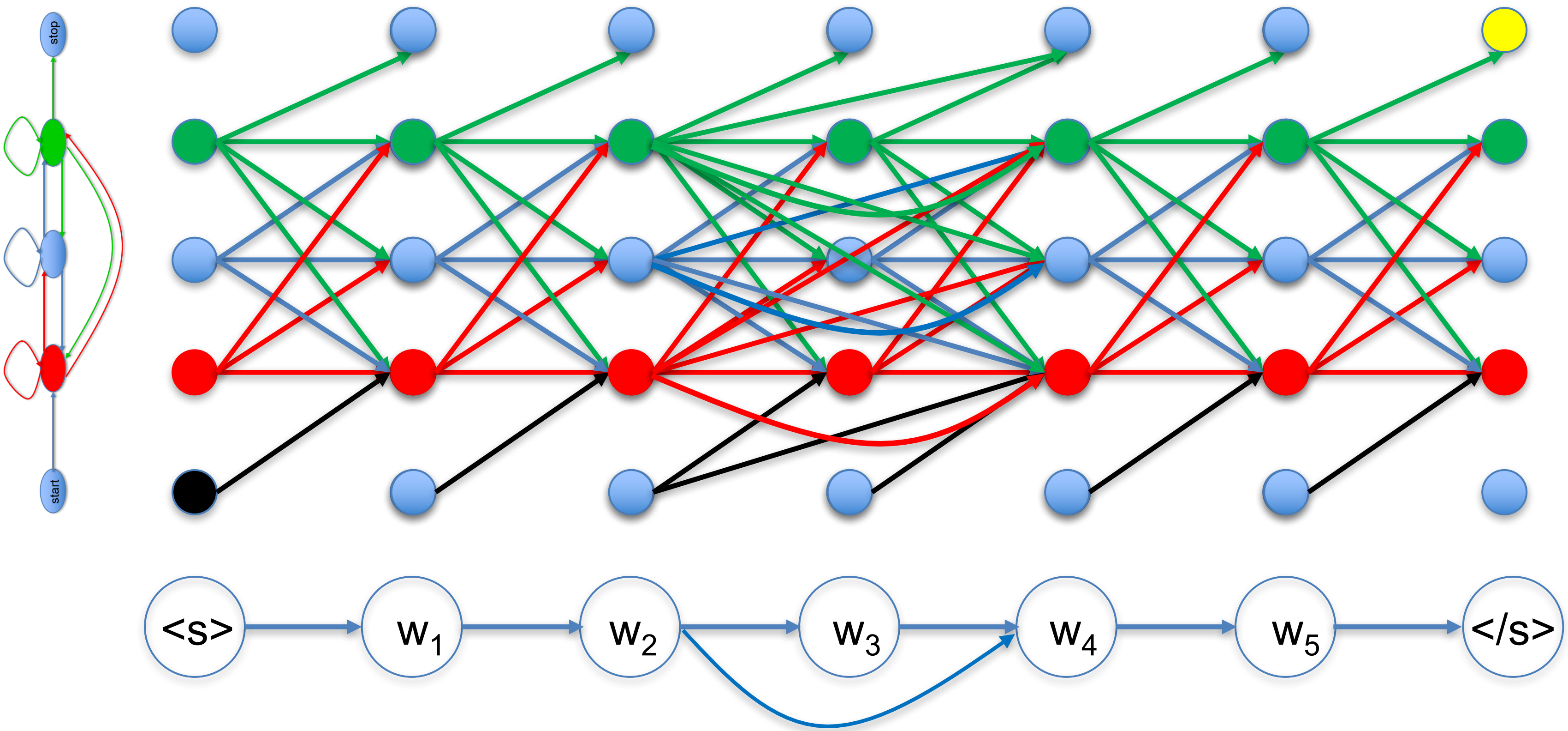
- Trellis obtained by composing Markov graph and observation graph
- Permits state sequences that skip the uncertain word

The modified trellis



- $NodeScore(s, w) = \log P(w|s)$
- $Edgescore\left((s_i, s_j), (w_k, w_l)\right) = \log P(s_j|s_i) + \log P(w_k|w_l)$
  - Note: $\log P(w_k|w_l) = -\infty$ for words that are not connected

The modified trellis



- *The Viterbi algorithm can be modified to solve this problem*

Generalizing the approach

Spare him not , kill him OR Spare him , not kill him

- What is the word graph for this problem?

Uses of HMMs in NLP

- Part-of-speech tagging (Church, 1988; Brants, 2000)
- Named entity recognition (Bikel et al., 1999) and other information extraction tasks
- Text chunking and shallow parsing (Ramshaw and Marcus, 1995)
- Word alignment in parallel text (Vogel et al., 1996)

- Also popular in computational biology and central to speech recognition.

Part of Speech Tagging

After paying the medical bills , Frances was nearly broke .

RB VBG DT JJ NNS , NNP VBZ RB JJ .

- Adverb (RB)
- Verb (VBG, VBZ, and others)
- Determiner (DT)
- Adjective (JJ)
- Noun (NN, NNS, NNP, and others)
- Punctuation (., ,, and others)

Named Entity Recognition

With Commander Chris Ferguson at the helm ,

Atlantis touched down at Kennedy Space Center .

Named Entity Recognition

O B-person I-person I-person O O O O

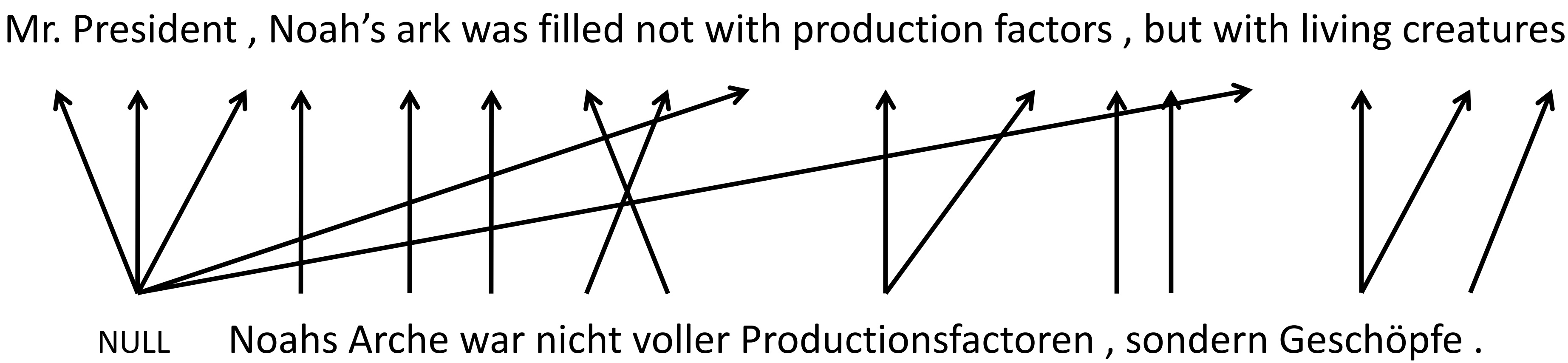
With Commander Chris Ferguson at the helm ,

B-space-shuttle O O O B-place I-place I-place O

Atlantis touched down at Kennedy Space Center .

- What makes this hard?

Word Alignment



Decoding / Inference

Hidden Markov Model

- A model over sequences of symbols, but there is missing information associated with each symbol: its "state."
  - Assume a finite set of possible states, Λ.

$$p(\text{start}, s_1, w_1, s_2, w_2, \ldots, s_n, w_n \text{stop}) \quad = \quad \prod_{i=1}^{n+1} \eta(w_i \mid s_i) \times \gamma(s_i \mid s_{i-1})$$

- A *joint* model over the observable symbols and their hidden/latent/unknown classes.

Key Algorithms for HMMs

Given the HMM and a sequence:

1. The most probable state sequence?
2. The probability of the word sequence?
3. The probability distribution over states, for each word?
4. Minimum risk sequence

Given states and sequences, or just states:

5. The parameters of the HMM ($\boldsymbol{\gamma}$ and $\boldsymbol{\eta}$)?

Problem 1:
Most Likely State Sequence

- Input: HMM (**$\boldsymbol{\gamma}$** and **$\boldsymbol{\eta}$**) and symbol sequence **w**.
- Output: $\arg\max_{\boldsymbol{s}} p(\boldsymbol{s} \mid \boldsymbol{w}, \boldsymbol{\gamma}, \boldsymbol{\eta})$

- Statistics view: maximum *a posteriori* inference
- Computational view: discrete, combinatorial optimization

Example

I	suspect	the	present	forecast	is	pessimistic	.
CD	JJ	DT	JJ	NN	NNS	JJ	.
NN	NN	JJ	NN	VB	VBZ		
NNP	VB	NN	RB	VBD			
PRP	VBP	NNP	VB	VBN			
		VBP	VBP	VBP			
4	4	5	5	5	2	1	1

4,000 possible state sequences!

Naïve Solutions

- List all the possibilities in Λ^n.
  - Correct.
  - Inefficient.
- Work left to right and greedily pick the best s_i at each point, based on s_{i-1} and w_i.
  - Not correct; solution may not be equal to:

    $$\arg\max_{\boldsymbol{s}} p(\boldsymbol{s} \mid \boldsymbol{w}, \boldsymbol{\gamma}, \boldsymbol{\eta})$$
  - But fast!

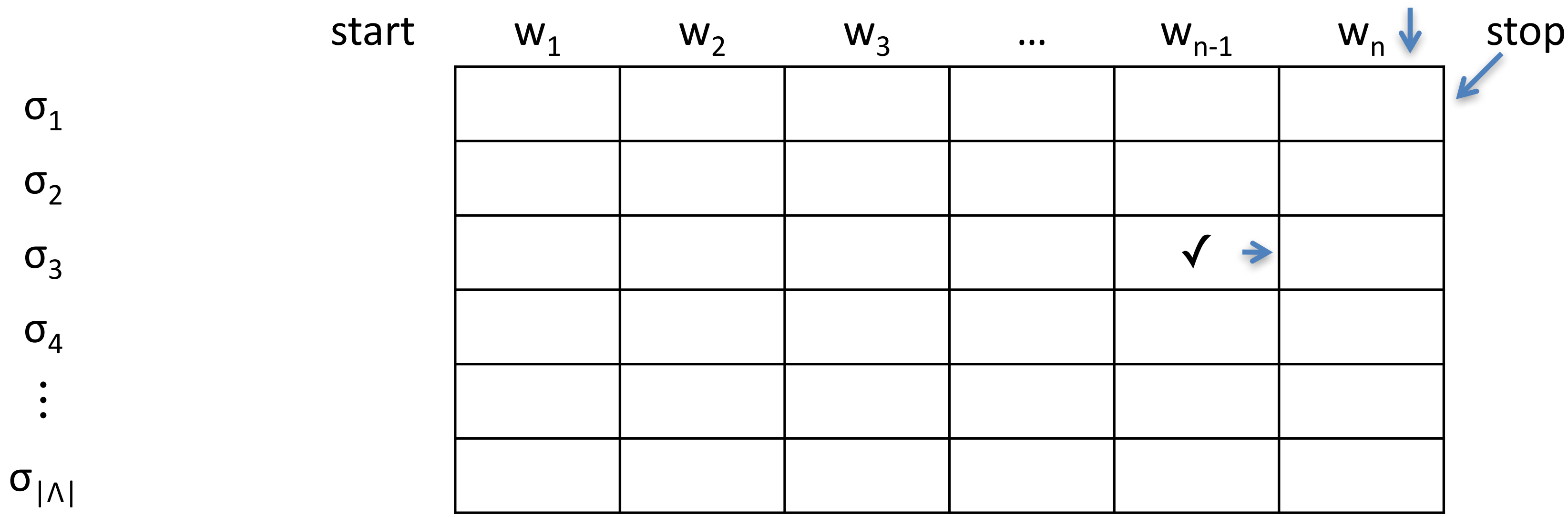
Interactions

- Each word’s label depends on the word, and nearby labels.
- But given *adjacent* labels, others do not matter.

I	suspect	the	present	forecast	is	pessimistic	.
CD	JJ	DT	JJ	NN	NNS	JJ	.
NN	NN	JJ	NN	VB	VBZ		
NNP	VB	NN	RB	VBD			
PRP	VBP	NNP	VB	VBN			
		VBP	VBP	VBP			

(arrows show *most preferred* label by each neighbor)

Base Case: Last Label



$$\text{score}_n(\sigma) \quad = \quad \gamma(\text{stop} \mid \sigma) \times \eta(w_n \mid \sigma) \times \gamma(\sigma \mid s_{n-1})$$

Of course, we do not actually know s_{n-1}!

Recurrence

- If I knew the score of every sequence $s_1 \ldots s_{n-1}$, I could reason easily about s_n.
  - But my decision about s_n would only depend on s_{n-1}!
- So I really only need to know the score of the best sequence ending in each s_{n-1}.
- Think of that as some "precalculation" that happens before I think about s_n.

Recurrence

- Assume we have the scores for all prefixes of the current state sequence.
  – One score for each possible last-state of the prefix.

$$\text{score}_n(\sigma) \quad = \quad \gamma(\text{stop} \mid \sigma) \times \eta(w_n \mid \sigma) \times \max_{\sigma'} \gamma(\sigma \mid \sigma') \times \text{score}_{n-1}(\sigma')$$

Recurrence

- The recurrence "bottoms out" at start.
- This leads to a simple algorithm for calculating all the scores.

$$
\begin{aligned}
\mathrm{score}_n(\sigma) &= \gamma(\mathrm{stop} \mid \sigma) \times \eta(w_n \mid \sigma) \times \max_{\sigma'} \gamma(\sigma \mid \sigma') \times \mathrm{score}_{n-1}(\sigma') \\
\mathrm{score}_{n-1}(\sigma) &= \eta(w_{n-1} \mid \sigma) \times \max_{\sigma'} \gamma(\sigma \mid \sigma') \times \mathrm{score}_{n-2}(\sigma') \\
\mathrm{score}_{n-2}(\sigma) &= \eta(w_{n-2} \mid \sigma) \times \max_{\sigma'} \gamma(\sigma \mid \sigma') \times \mathrm{score}_{n-3}(\sigma') \\
&\vdots \\
\mathrm{score}_1(\sigma) &= \eta(w_1 \mid \sigma) \times \gamma(\sigma \mid \mathrm{start})
\end{aligned}
$$

Viterbi Algorithm (Scores Only)

- For every σ in Λ, let:

$$\text{score}_1(\sigma) \quad = \quad \eta(w_1 \mid \sigma) \times \gamma(\sigma \mid \text{start})$$

- For i = 2 to n – 1, for every σ in Λ:

$$\text{score}_i(\sigma) \quad = \quad \eta(w_i \mid \sigma) \times \max_{\sigma' \in \Lambda} \gamma(\sigma \mid \sigma') \times \text{score}_{i-1}(\sigma')$$

- For every σ in Λ:

$$\text{score}_n(\sigma) \quad = \quad \gamma(\text{stop} \mid \sigma) \times \eta(w_n \mid \sigma) \times \max_{\sigma' \in \Lambda} \gamma(\sigma \mid \sigma') \times \text{score}_{n-1}(\sigma')$$

- Claim:

$$\max_{\boldsymbol{s}} p(\boldsymbol{s}, \boldsymbol{w} \mid \boldsymbol{\gamma}, \boldsymbol{\eta}) = \max_{\sigma \in \Lambda} \text{score}_n(\sigma)$$

Exploiting Distributivity

$$\begin{aligned}
\max_{\sigma\in\Lambda} \mathrm{score}_n(\sigma) &= \max_{\sigma\in\Lambda} \gamma(\mathrm{stop} \mid \sigma) \times \eta(w_n \mid \sigma) \times \max_{\sigma'\in\Lambda} \gamma(\sigma \mid \sigma') \times \mathrm{score}_{n-1}(\sigma') \\
&= \max_{\sigma\in\Lambda} \gamma(\mathrm{stop} \mid \sigma) \times \eta(w_n \mid \sigma) \times \max_{\sigma'\in\Lambda} \gamma(\sigma \mid \sigma') \\
&\quad \times \eta(w_{n-1} \mid \sigma') \times \max_{\sigma''\in\Lambda} \gamma(\sigma' \mid \sigma'') \times \mathrm{score}_{n-2}(\sigma'') \\
&= \max_{\sigma\in\Lambda} \gamma(\mathrm{stop} \mid \sigma) \times \eta(w_n \mid \sigma) \times \max_{\sigma'\in\Lambda} \gamma(\sigma \mid \sigma') \\
&\quad \times \eta(w_{n-1} \mid \sigma') \times \max_{\sigma''\in\Lambda} \gamma(\sigma' \mid \sigma'') \\
&\quad \times \eta(w_{n-2} \mid \sigma'') \times \max_{\sigma'''\in\Lambda} \gamma(\sigma'' \mid \sigma''') \times \mathrm{score}_{n-3}(\sigma''') \\
&= \max_{\sigma,\sigma',\sigma'',\sigma'''} \gamma(\mathrm{stop} \mid \sigma) \times \eta(w_n \mid \sigma) \times \gamma(\sigma \mid \sigma') \\
&\quad \times \eta(w_{n-1} \mid \sigma') \times \gamma(\sigma' \mid \sigma'') \\
&\quad \times \eta(w_{n-2} \mid \sigma'') \times \gamma(\sigma'' \mid \sigma''') \times \mathrm{score}_{n-3}(\sigma''') \\
&= \max_{\boldsymbol{s}\in\Lambda^n} \prod_{i=1}^{n+1} \gamma(s_i \mid s_{i-1}) \times \eta(w_i \mid s_i)
\end{aligned}$$

$$\max_{\boldsymbol{s}} p(\boldsymbol{s}, \boldsymbol{w} \mid \boldsymbol{\gamma}, \boldsymbol{\eta}) = \max_{\sigma\in\Lambda} \mathrm{score}_n(\sigma)$$

	I	suspect	the	present	forecast	is	pessimistic	.
CD	3E-7							
DT			3E-8					
JJ		1E-9	1E-12	3E-12			7E-23	
NN	4E-6	2E-10	1E-13	6E-13	4E-16			
NNP	1E-5		4E-13					
NNS						1E-21		
PRP	4E-3							
RB				2E-14				
VB		6E-9		3E-15	2E-19			
VBD					6E-18			
VBN					4E-18			
VBP		5E-7	4E-14	4E-15	9E-19			
VBZ						6E-18		
.								2E-24
	1	2	3	4	5	6	7	8

Not Quite There

- As described, this algorithm only lets us calculate the *probability* of the best label sequence.
- It does not recover the best sequence!

Understanding the Scores

- $\text{score}_i(\sigma)$ is the score of the best sequence labeling up through w_i, ignoring what comes later.

$$\text{score}_i(\sigma) = \max_{s_1,\ldots,s_{i-1}} p(s_1, w_1, s_2, w_2, \ldots, s_i = \sigma, w_i)$$

- Similar trick as before: if I know what s_{i+1} is, then I can use the scores to choose s_i.
- Solution: keep backpointers.

	I	suspect	the	present	forecast	is	pessimistic	.
CD	3E-7							
DT			3E-8					
JJ		1E-9	1E-12	3E-12			7E-23	
NN	4E-6	2E-10	1E-13	6E-13	4E-16			
NNP	1E-5		4E-13					
NNS						1E-21		
PRP	4E-3							
RB				2E-14				
VB		6E-9		3E-15	2E-19			
VBD					6E-18			
VBN					4E-18			
VBP		5E-7	4E-14	4E-15	9E-19			
VBZ						6E-18		
.								2E-24

	I	suspect	the	present	forecast	is	pessimistic	.
CD	3E-7							
DT			3E-8					
JJ		1E-9	1E-12	3E-12			7E-23	
NN	4E-6	2E-10	1E-13	6E-13	4E-16			
NNP	1E-5		4E-13					
NNS						1E-21		
PRP	4E-3							
RB				2E-14				
VB		6E-9		3E-15	2E-19			
VBD					6E-18			
VBN					4E-18			
VBP		5E-7	4E-14	4E-15	9E-19			
VBZ						6E-18		
.								2E-24

Viterbi Algorithm

- For every σ in Λ, let:

$$\text{score}_1(\sigma) \quad = \quad \eta(w_1 \mid \sigma) \times \gamma(\sigma \mid \text{start})$$

- For i = 2 to n – 1, for every σ in Λ:

$$\text{score}_i(\sigma) \quad = \quad \eta(w_i \mid \sigma) \times \max_{\sigma' \in \Lambda} \gamma(\sigma \mid \sigma') \times \text{score}_{i-1}(\sigma')$$

$$\text{bp}_i(\sigma) = \arg\max_{\sigma' \in \Lambda} \gamma(\sigma \mid \sigma') \times \text{score}_{i-1}(\sigma')$$

- For every σ in Λ:

$$\text{score}_n(\sigma) \quad = \quad \gamma(\text{stop} \mid \sigma) \times \eta(w_n \mid \sigma) \times \max_{\sigma' \in \Lambda} \gamma(\sigma \mid \sigma') \times \text{score}_{n-1}(\sigma')$$

$$\text{bp}_n(\sigma) = \arg\max_{\sigma' \in \Lambda} \gamma(\sigma \mid \sigma') \times \text{score}_{n-1}(\sigma')$$

Viterbi Algorithm: Backtrace

- After calculating all score and bp values, start by choosing s_n to maximize $score_n$.
- Then let $s_{n-1} = bp_n(s_n)$.

- In general, $s_{i-1} = bp_i(s_i)$.

Another Example

	time	flies	like	an	arrow	.
DT				10e-15	6e-21	
IN			8e-13		1e-19	
JJ			6e-14		2e-16	
NN	2e-4				3e-16	
NNP					1e-16	
VB	2e-7		1e-14		1e-19	
VBP			8e-16		4e-19	
VBZ		2e-9			3e-18	
.					1e-21	3e-17
,				4e-20	5e-22	

Another Example

	time	flies	like	an	arrow	.
DT				10e-15	6e-21	
IN			8e-13		1e-19	
JJ			6e-14		2e-16	
NN	2e-4				3e-16	
NNP					1e-16	
VB	2e-7		1e-14		1e-19	
VBP			8e-16		4e-19	
VBZ		2e-9			3e-18	
.					1e-21	3e-17
,				4e-20	5e-22	

Lecture Outline

✓ Viterbi algorithm

2. Decoding more generally
3. Five views

Inference

- Eventually, you need to run your structured predictor on test data!
- For sequence labeling and segmentation models with very local interactions, decoding is usually accomplished by something "like" Viterbi algorithm.

Random Variables

- A variable whose value depends on chance
- Denoted by capital letters: X, Y, Z
- Associated with sets of possible values: Val(X)
- A single possible value: $x \in$ Val(X)
- Probabilistic modeling: defining distributions over r.v.s

- There's more than one way to map your structured prediction problem to random variables!

Probabilistic Inference Problems

Given values for some random variables ($\mathbf{X} \subset \mathbf{V}$) …

- *Most Probable Explanation: what are the *most probable* values of the *rest* of the r.v.s $\mathbf{V} \setminus \mathbf{X}$?

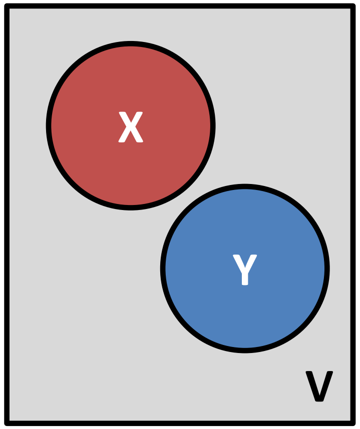
(More generally …)

- *Maximum *A Posteriori* (MAP): what are the most probable values of *some* other r.v.s, $\mathbf{Y} \subset (\mathbf{V} \setminus \mathbf{X})$?

These do not need to be probabilistic! Change "most probable" to "maximum scoring."

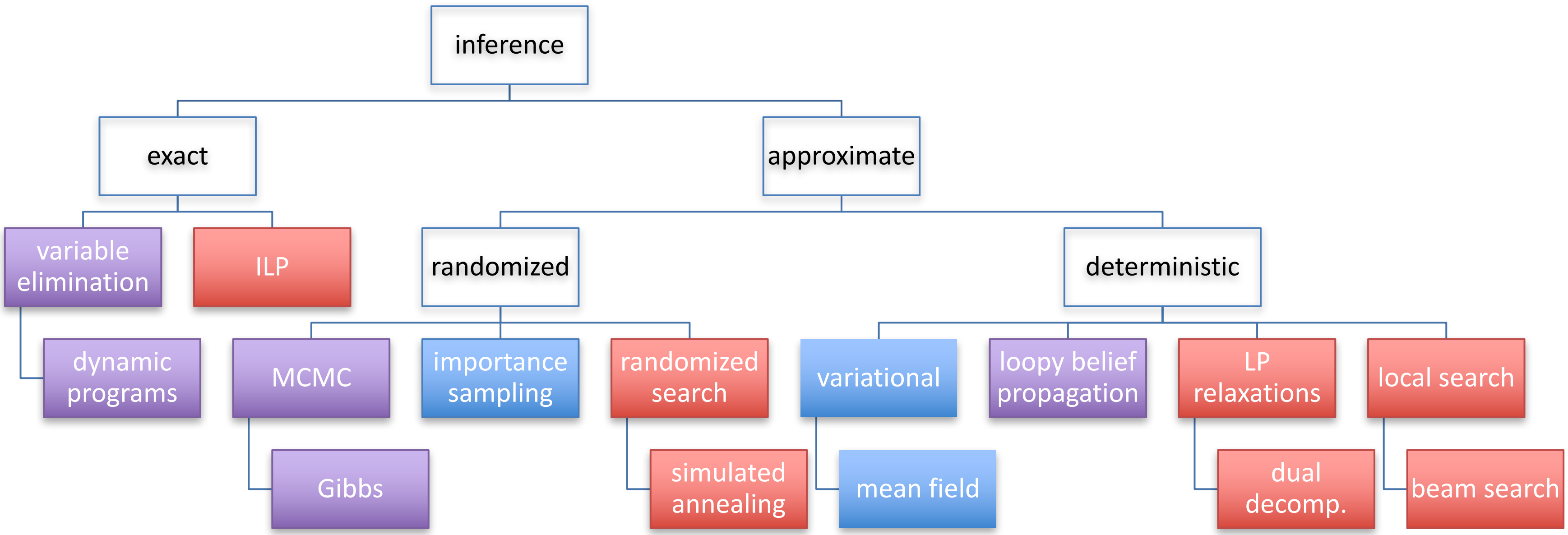
- Random sampling from the posterior over values of **Y**
- Full posterior over values of **Y**
- Marginal probabilities from the posterior over **Y**

- *Minimum Bayes risk: What is the **Y** with the lowest expected cost?
- *Cost-augmented decoding: What is the most *dangerous* value of **Y**, compared to true **y***?



*Different kinds of **decoding**.

Approaches to Inference



red = hard inference
blue = soft inference
purple = both

Hidden Markov Model

- **X** and **Y** are both sequences of symbols
  - **X** is a sequence from the vocabulary Σ
  - **Y** is a sequence from the state space Λ

$$p(\boldsymbol{Y} = \boldsymbol{s}, \boldsymbol{X} = \boldsymbol{w}) = \\ p(\text{start}, s_1, w_1, s_2, w_2, \ldots, s_n, w_n \text{stop}) \quad = \quad \prod_{i=1}^{n+1} \eta(w_i \mid s_i) \times \gamma(s_i \mid s_{i-1})$$

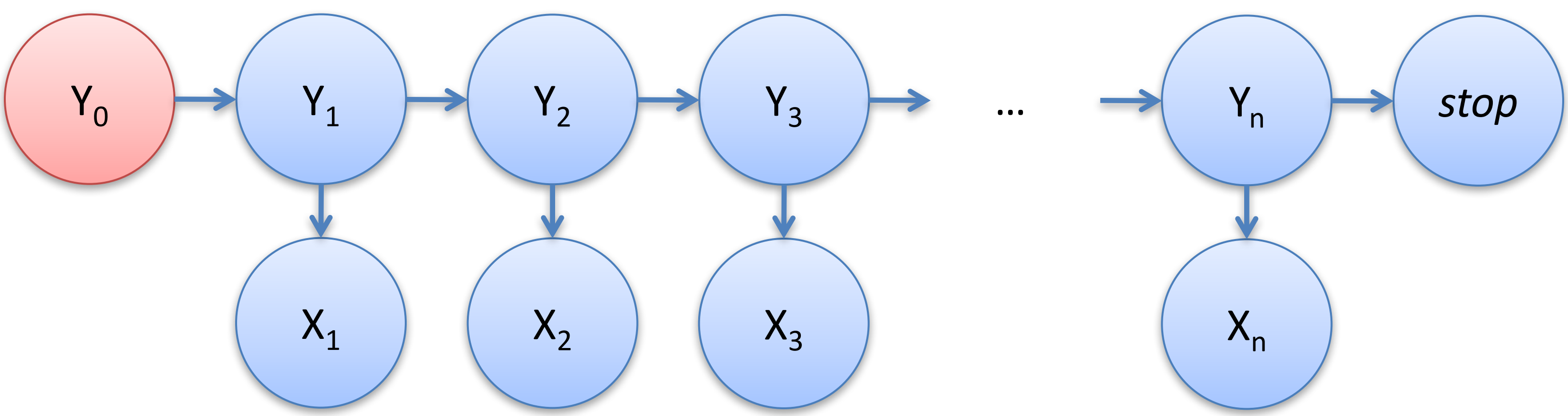
- Parameters:
  - Transitions **γ** including γ(*stop* | s), γ(s | *start*)
  - Emissions **η**

Hidden Markov Model

- The joint model's independence assumptions are easy to capture with a Bayesian network.

$$p(\boldsymbol{Y} = \boldsymbol{s}, \boldsymbol{X} = \boldsymbol{w}) =$$

$$p(\text{start}, s_1, w_1, s_2, w_2, \ldots, s_n, w_n \text{stop}) = \prod_{i=1}^{n+1} \eta(w_i \mid s_i) \times \gamma(s_i \mid s_{i-1})$$

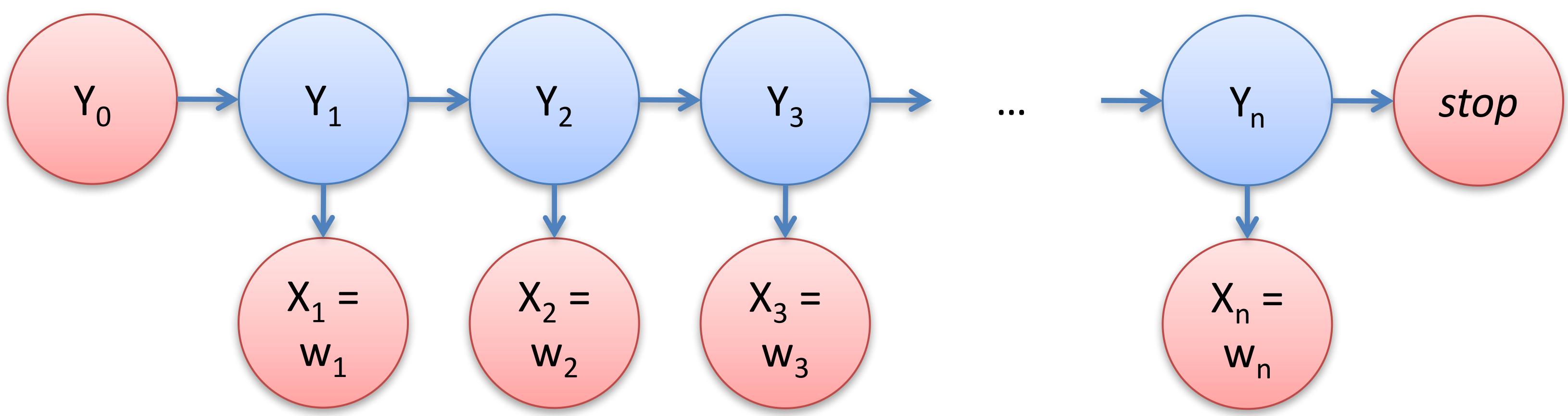


Hidden Markov Model

- The MPE/MAP inference problem is to find the most probable value of **Y** given **X = x**.

$$p(\boldsymbol{Y} = \boldsymbol{s}, \boldsymbol{X} = \boldsymbol{w}) =$$

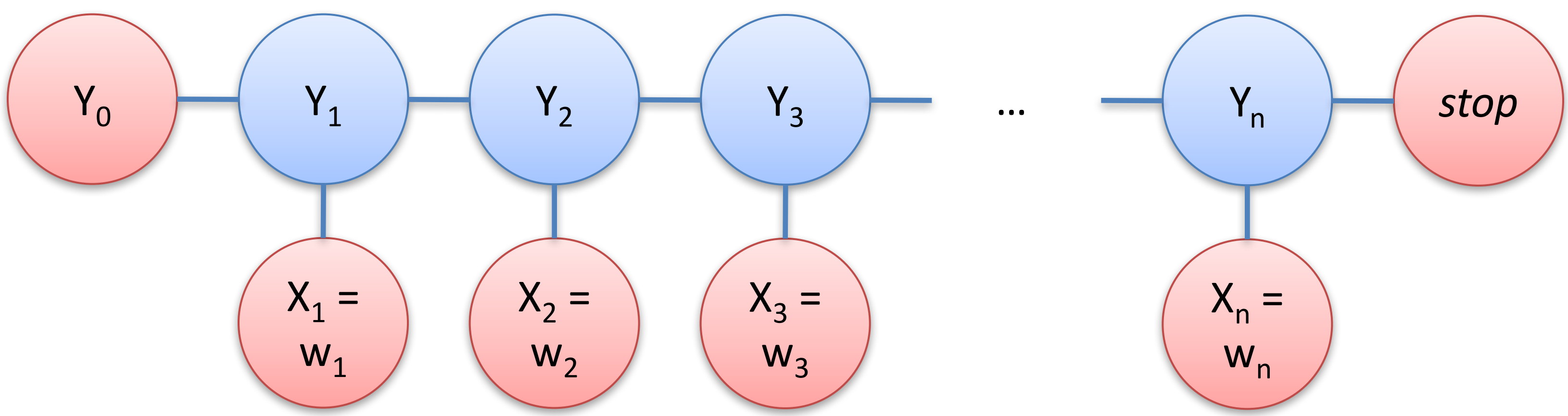
$$p(\text{start}, s_1, w_1, s_2, w_2, \ldots, s_n, w_n \text{stop}) \quad = \quad \prod_{i=1}^{n+1} \eta(w_i \mid s_i) \times \gamma(s_i \mid s_{i-1})$$



Hidden Markov Model

- The MPE/MAP inference problem is to find the most probable value of **Y** given **X** = **x**.

- Markov network:



Markov Network

- A different graphical model representation; undirected. Vertices are still r.v.s.
- Every clique C in the graph gets a *local* scoring function ϕ_C that maps assignments to values.

$$mulscore(\boldsymbol{x}, \boldsymbol{y}) = \prod_{C \in \mathcal{C}} \phi_C(\Pi_C(\boldsymbol{x}, \boldsymbol{y}))$$

$$addscore(\boldsymbol{x}, \boldsymbol{y}) = \sum_{C \in \mathcal{C}} \log \phi_C(\Pi_C(\boldsymbol{x}, \boldsymbol{y}))$$

- This score can be *globally* renormalized to obtain a probabilistic interpretation. (Not today.)

Restriction #1

1. The score function needs to *factor locally*.
   – The more locally, the better!

$$score(\boldsymbol{x}, \boldsymbol{y}) = \sum_{C \in \mathcal{C}} \log \phi_C(\Pi_C(\boldsymbol{x}, \boldsymbol{y}))$$

Linear Models

- Define a feature vector function **g** that maps (**x**, **y**) pairs into d-dimensional real space.
- Score is linear in **g**(**x**, **y**).

$$\begin{aligned} score(\boldsymbol{x}, \boldsymbol{y}) &= \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) \\ \boldsymbol{y}^* &= \arg\max_{\boldsymbol{y} \in \mathcal{Y}_{\boldsymbol{x}}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) \end{aligned}$$

- Results:
  - decoding seeks **y** to maximize the score.
  - learning seeks **w** to … do something we' ll talk about later.
- Extremely general!

Generic Noisy Channel as Linear Model

$$
\begin{aligned}
\hat{\boldsymbol{y}} &= \arg\max_{\boldsymbol{y}} \log \left(p(\boldsymbol{y}) \cdot p(\boldsymbol{x} \mid \boldsymbol{y}) \right) \\
&= \arg\max_{\boldsymbol{y}} \log p(\boldsymbol{y}) + \log p(\boldsymbol{x} \mid \boldsymbol{y}) \\
&= \arg\max_{\boldsymbol{y}} w_{\boldsymbol{y}} + w_{\boldsymbol{x}|\boldsymbol{y}} \\
&= \arg\max_{\boldsymbol{y}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y})
\end{aligned}
$$

- Of course, the two probability terms are typically composed of “smaller” factors; each can be understood as an exponentiated weight.

Max Ent Models as Linear Models

$$
\begin{aligned}
\hat{\boldsymbol{y}} &= \arg\max_{\boldsymbol{y}} \log p(\boldsymbol{y} \mid \boldsymbol{x}) \\
&= \arg\max_{\boldsymbol{y}} \log \frac{\exp \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y})}{z(\boldsymbol{x})} \\
&= \arg\max_{\boldsymbol{y}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) - \log z(\boldsymbol{x}) \\
&= \arg\max_{\boldsymbol{y}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y})
\end{aligned}
$$

HMMs as Linear Models

$$
\begin{aligned}
\hat{\boldsymbol{y}} &= \arg\max_{\boldsymbol{y}} \log p(\boldsymbol{x}, \boldsymbol{y}) \\
&= \arg\max_{\boldsymbol{y}} \left(\sum_{i=1}^{n} \log p(x_i \mid y_i) + \log p(y_i \mid y_{i-1}) \right) + \log p(stop \mid y_n) \\
&= \arg\max_{\boldsymbol{y}} \left(\sum_{i=1}^{n} w_{y_i \downarrow x_i} + w_{y_{i-1} \rightarrow y_i} \right) + w_{y_n \rightarrow stop} \\
&= \arg\max_{\boldsymbol{y}} \sum_{y,x} w_{y \downarrow x} freq(y \downarrow x; \boldsymbol{y}, \boldsymbol{x}) + \sum_{y,y'} w_{y \rightarrow y'} freq(y \rightarrow y'; \boldsymbol{y}) \\
&= \arg\max_{\boldsymbol{y}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y})
\end{aligned}
$$

Restrictions #1, #2

1. The score function needs to *factor locally*.
   – The more locally, the better!

$$score(\boldsymbol{x}, \boldsymbol{y}) = \sum_{C \in \mathcal{C}} \log \phi_C(\Pi_C(\boldsymbol{x}, \boldsymbol{y}))$$

2. The local scoring functions need to be linear in features.

$$score(\boldsymbol{x}, \boldsymbol{y}) = \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y})$$

$$score(\boldsymbol{x}, \boldsymbol{y}) = \sum_{C \in \mathcal{C}} \mathbf{w}^\top \mathbf{f}(\Pi_C(\boldsymbol{x}, \boldsymbol{y}))$$

Running Example

		1	2	3	4	5	6	7	8	9	10
$\boldsymbol{x}$	=	Britain	sent	warships	across	the	English	Channel	Monday	to	rescue
$\boldsymbol{y}$	=	B	O	O	O	O	B	I	B	O	O
$\boldsymbol{y}'$	=	O	O	O	O	O	B	I	B	O	O

11	12	13	14	15	16	17	18	19	20
Britons	stranded	by	Eyjafjallajökull	's	volcanic	ash	cloud	.	
B	O	O	B	O	O	O	O	O	O
B	O	O	B	O	O	O	O	O	O

- IOB sequence labeling, here applied to NER
- Often solved with HMMs, CRFs, M^{3}Ns …

feature function $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$		$g(\boldsymbol{x}, \boldsymbol{y})$	$g(\boldsymbol{x}, \boldsymbol{y}')$
bias:	count of i s.t. $y_i = \mathsf{B}$	5	4
	count of i s.t. $y_i = \mathsf{I}$	1	1
	count of i s.t. $y_i = \mathsf{O}$	14	15
lexical:	count of i s.t. $x_i = \textit{Britain}$ and $y_i = \mathsf{B}$	1	0
	count of i s.t. $x_i = \textit{Britain}$ and $y_i = \mathsf{I}$	0	0
	count of i s.t. $x_i = \textit{Britain}$ and $y_i = \mathsf{O}$	0	1
downcased:	count of i s.t. $lc(x_i) = \textit{britain}$ and $y_i = \mathsf{B}$	1	0
	count of i s.t. $lc(x_i) = \textit{britain}$ and $y_i = \mathsf{I}$	0	0
	count of i s.t. $lc(x_i) = \textit{britain}$ and $y_i = \mathsf{O}$	0	1
	count of i s.t. $lc(x_i) = \textit{sent}$ and $y_i = \mathsf{O}$	1	1
	count of i s.t. $lc(x_i) = \textit{warships}$ and $y_i = \mathsf{O}$	1	1
shape:	count of i s.t. $\textit{shape}(x_i) = \textit{Aaaaaaa}$ and $y_i = \mathsf{B}$	3	2
	count of i s.t. $\textit{shape}(x_i) = \textit{Aaaaaaa}$ and $y_i = \mathsf{I}$	1	1
	count of i s.t. $\textit{shape}(x_i) = \textit{Aaaaaaa}$ and $y_i = \mathsf{O}$	0	1
prefix:	count of i s.t. $\textit{pre}_1(x_i) = B$ and $y_i = \mathsf{B}$	2	1
	count of i s.t. $\textit{pre}_1(x_i) = B$ and $y_i = \mathsf{I}$	0	0
	count of i s.t. $\textit{pre}_1(x_i) = B$ and $y_i = \mathsf{O}$	0	1
	count of i s.t. $\textit{pre}_1(x_i) = s$ and $y_i = \mathsf{O}$	2	2
	count of i s.t. $\textit{shape}(\textit{pre}_1(x_i)) = A$ and $y_i = \mathsf{B}$	5	4
	count of i s.t. $\textit{shape}(\textit{pre}_1(x_i)) = A$ and $y_i = \mathsf{I}$	1	1
	count of i s.t. $\textit{shape}(\textit{pre}_1(x_i)) = A$ and $y_i = \mathsf{O}$	0	1
	$[\![\textit{shape}(\textit{pre}_1(x_1)) = A \wedge y_1 = \mathsf{B}]\!]$	1	0
	$[\![\textit{shape}(\textit{pre}_1(x_1)) = A \wedge y_1 = \mathsf{O}]\!]$	0	1
gazetteer:	count of i s.t. x_i is in the gazetteer and $y_i = \mathsf{B}$	2	1
	count of i s.t. x_i is in the gazetteer and $y_i = \mathsf{I}$	0	0
	count of i s.t. x_i is in the gazetteer and $y_i = \mathsf{O}$	0	1
	count of i s.t. $x_i = \textit{sent}$ and $y_i = \mathsf{O}$	1	1

(What is *Not* A Linear Model?)

- Probabilistic models with hidden variables, requiring general MAP inference:

$$\arg\max_{\boldsymbol{y}} p(\boldsymbol{y} \mid \boldsymbol{x}) = \arg\max_{\boldsymbol{y}} \sum_{\boldsymbol{z}} p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})$$

- Models based on non-linear kernels

$$\arg\max_{\boldsymbol{y}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) \quad = \quad \arg\max_{\boldsymbol{y}} \sum_{i=1}^{N} \alpha_i K\left(\langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle, \langle \boldsymbol{x}, \boldsymbol{y} \rangle\right)$$

Lecture Outline

✓ Viterbi algorithm

✓ Decoding more generally

3. Five views

1. Probabilistic Graphical Models

- View the linguistic structure as a collection of random variables that are interdependent.
- Represent interdependencies as a directed or undirected graphical model.
- Conditional probability tables (BNs) or factors (MNs) encode the probability distribution.
- Use standard techniques from PGMs to decode.

Inference in Graphical Models

- General algorithm for exact MPE inference: variable elimination.
  - Iteratively solve for the best values of each variable conditioned on values of "preceding" neighbors.
  - Then trace back.
  - Challenge: order the r.v.s for efficiency!

The Viterbi algorithm is an instance of max-product variable elimination!

MAP is Linear Decoding

- Bayesian network:

$$\sum_i \log p(x_i \mid \text{parents}(X_i)) + \sum_j \log p(y_j \mid \text{parents}(Y_j))$$

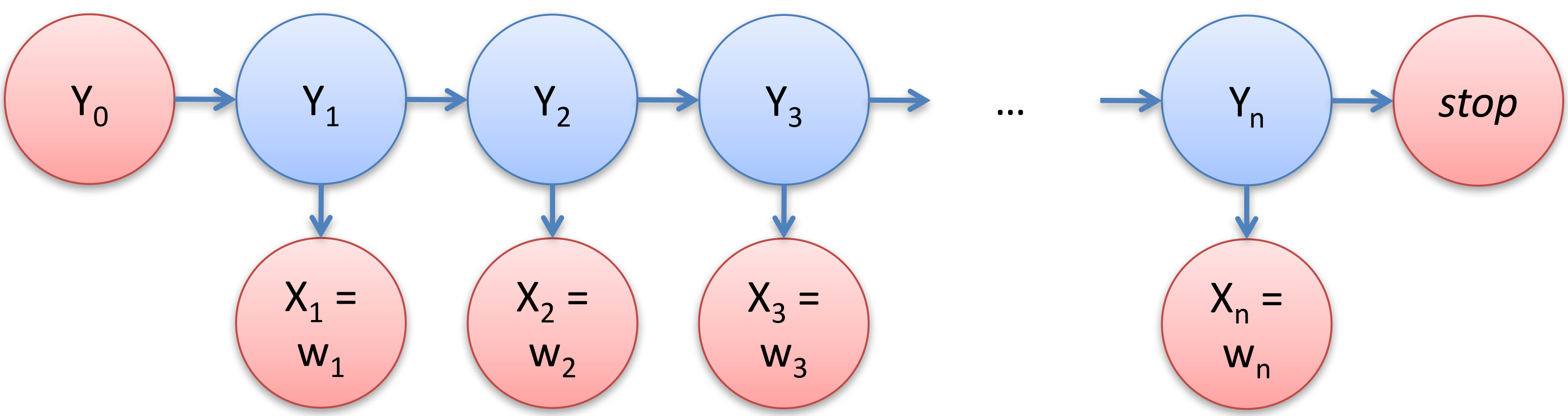
- Markov network:

$$\sum_C \log \phi_C\left(\{x_i\}_{i \in C}, \{y_j\}_{j \in C}\right)$$

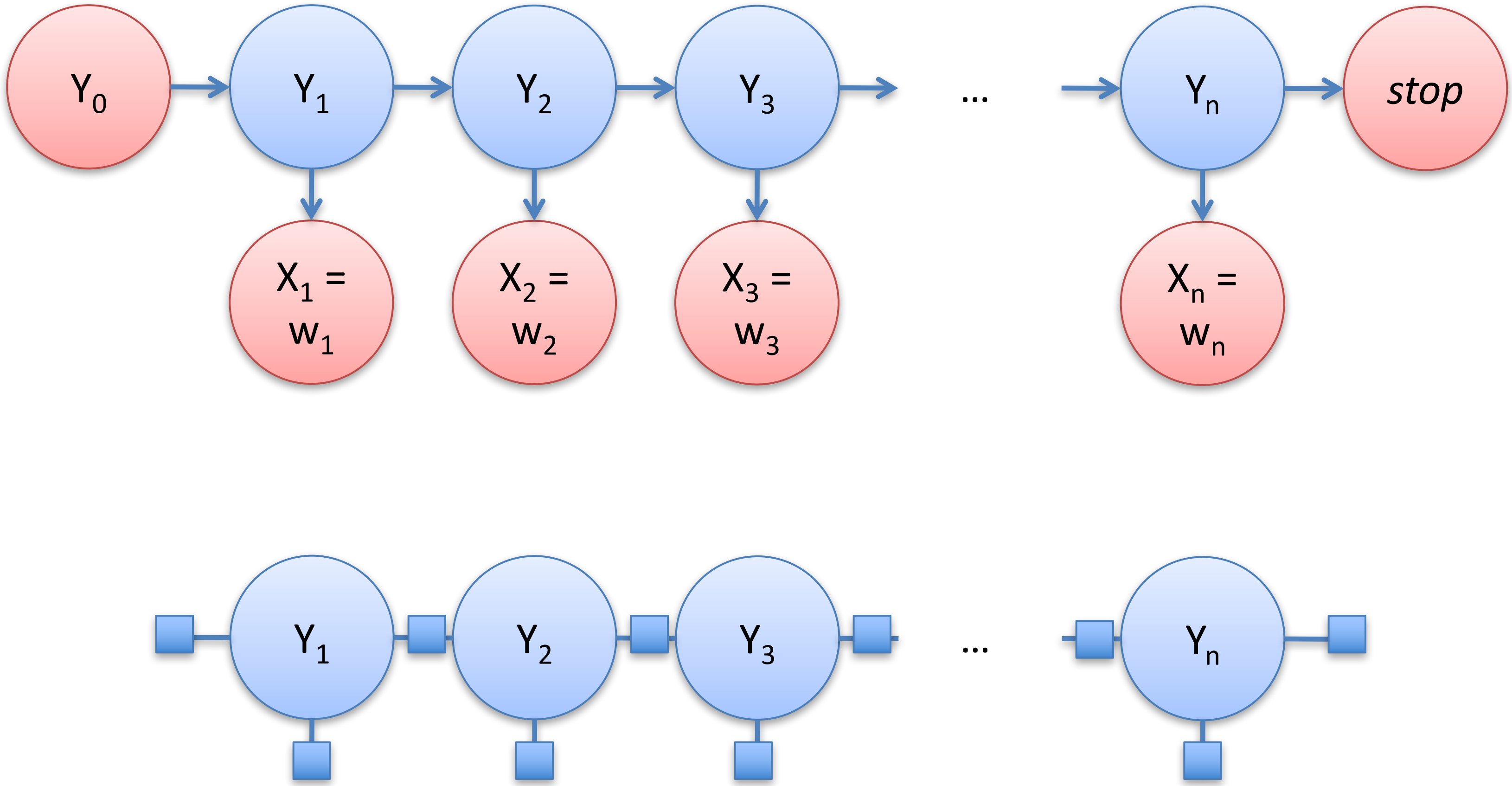
- This works if every variable is in **X** or **Y**.

Hidden Markov Model

- When we eliminate Y_1, we take a product of three relevant factors.
  - $\gamma(Y_1 \mid \textit{start})$
  - $\eta(w_1 \mid Y_1)$, reduced to the observed value w_1
  - $\gamma(Y_2 \mid Y_1)$

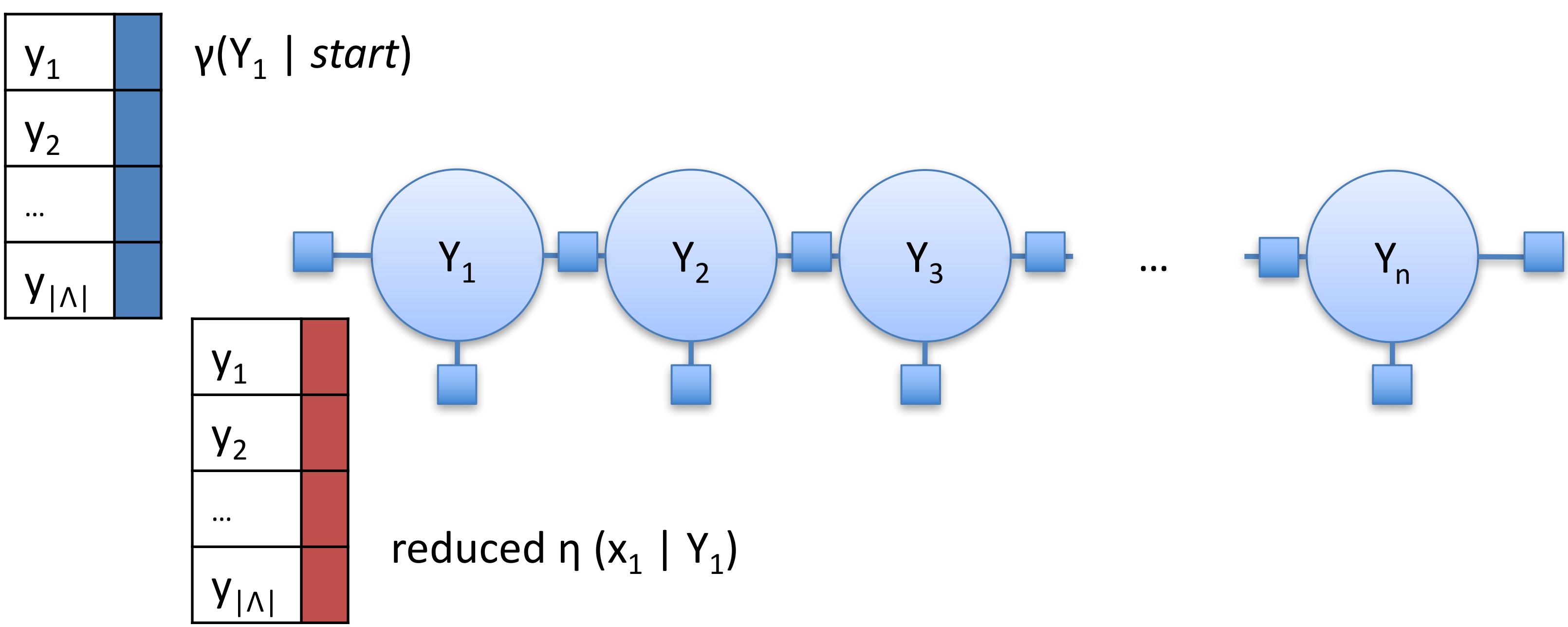


Factor Representation



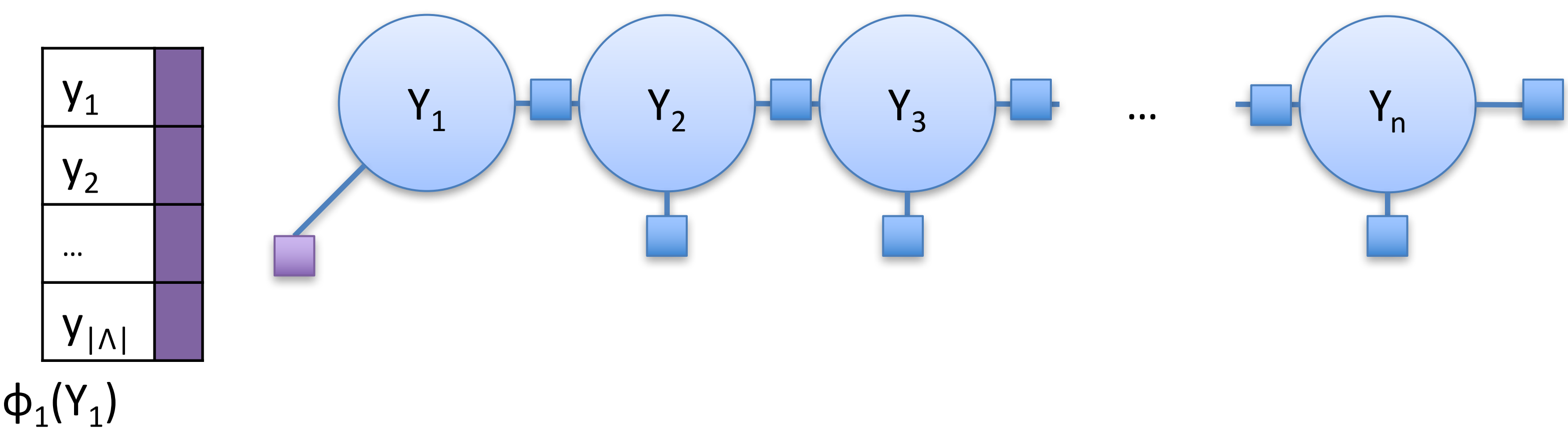
Hidden Markov Model

- When we eliminate Y_1, we first take a product of two factors that only involve Y_1.



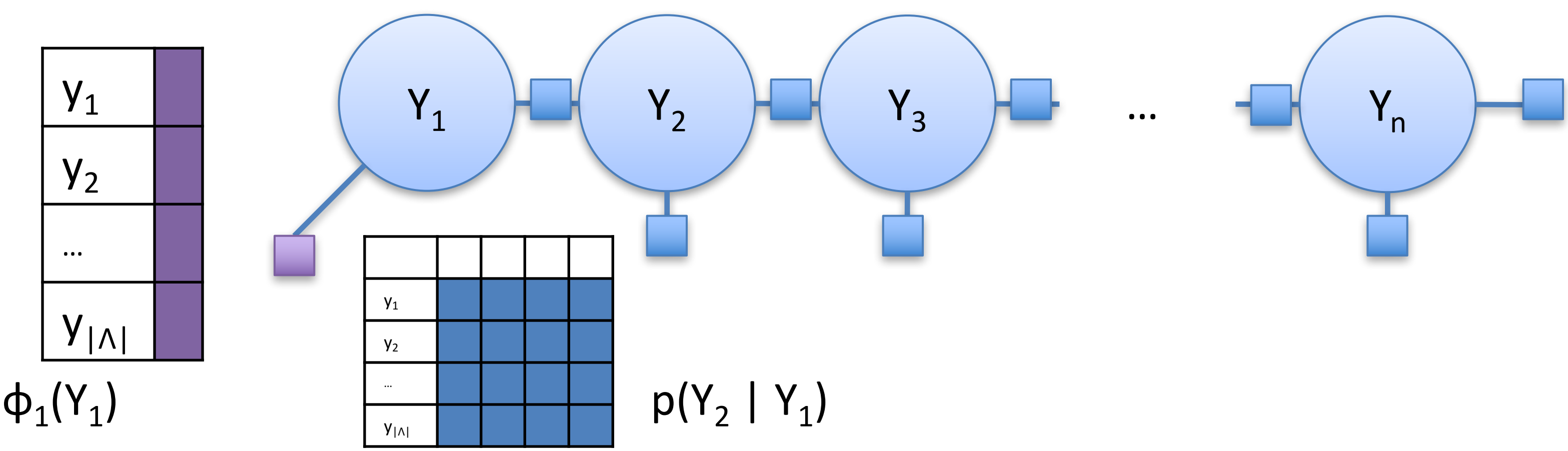
Hidden Markov Model

- When we eliminate Y_1, we first take a product of two factors that only involve Y_1.
- This is the Viterbi probability vector for Y_1.



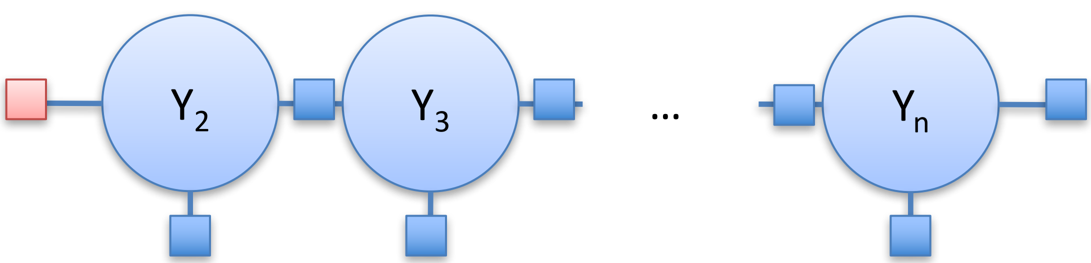
Hidden Markov Model

- When we eliminate Y_1, we first take a product of two factors that only involve Y_1.
- This is the Viterbi probability vector for Y_1.
- Eliminating Y_1 equates to solving the Viterbi probabilities for Y_2.



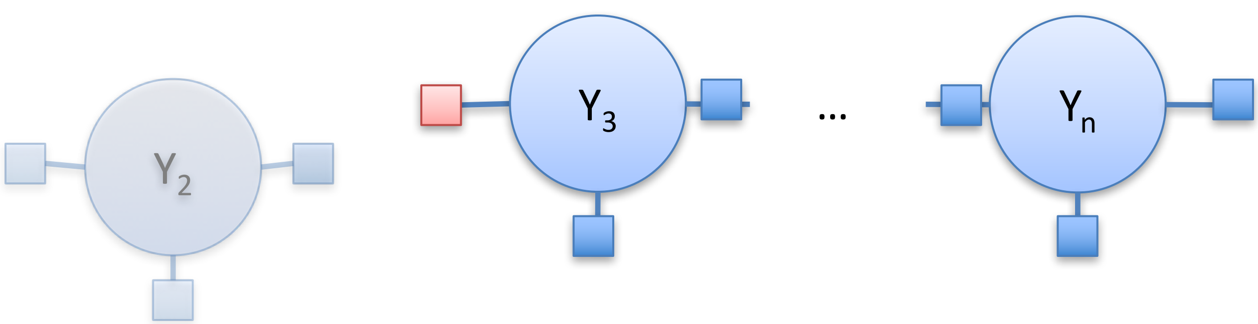
Hidden Markov Model

- Product of all factors involving Y_1, then reduce.
  - $\phi_2(Y_2) = \max_{y \in Val(Y_1)} (\phi_1(y) \times p(Y_2 \mid y))$
  - This factor holds Viterbi probabilities for Y_2.



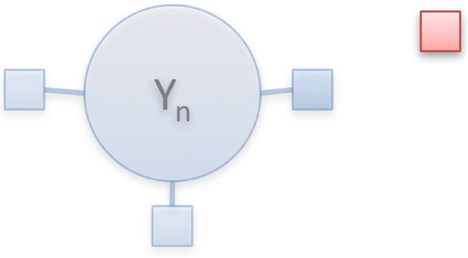
Hidden Markov Model

- When we eliminate Y_2, we take a product of the analogous two relevant factors.
- Then reduce.
  - $\phi_3(Y_3) = \max_{y \in Val(Y_2)} (\phi_2(y) \times p(Y_3 \mid y))$



Hidden Markov Model

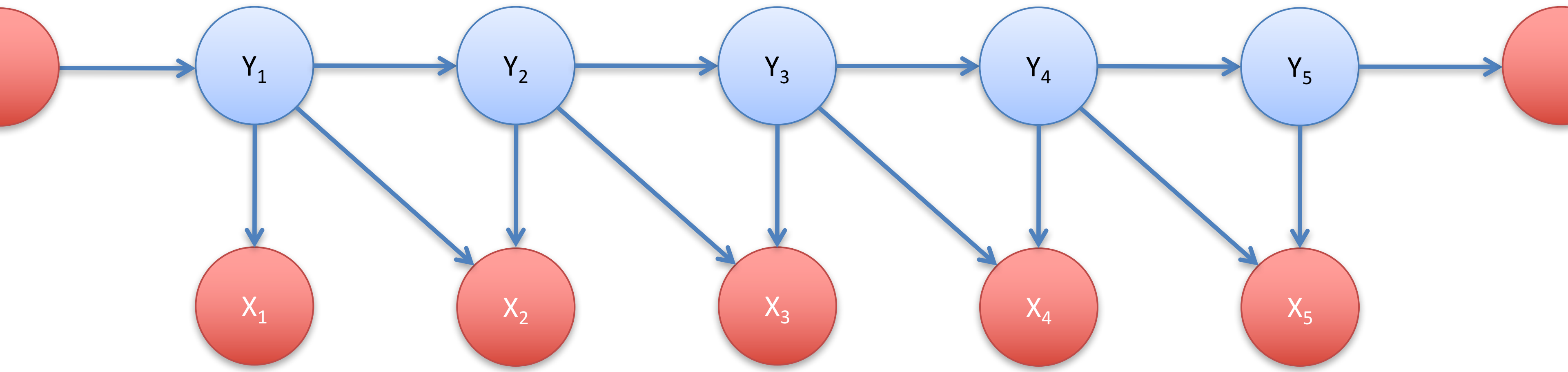
- At the end, we have one final factor with one row, ϕ_{n+1}.
- This is the score of the best sequence.
- Use backtrace to recover values.



Why Think This Way?

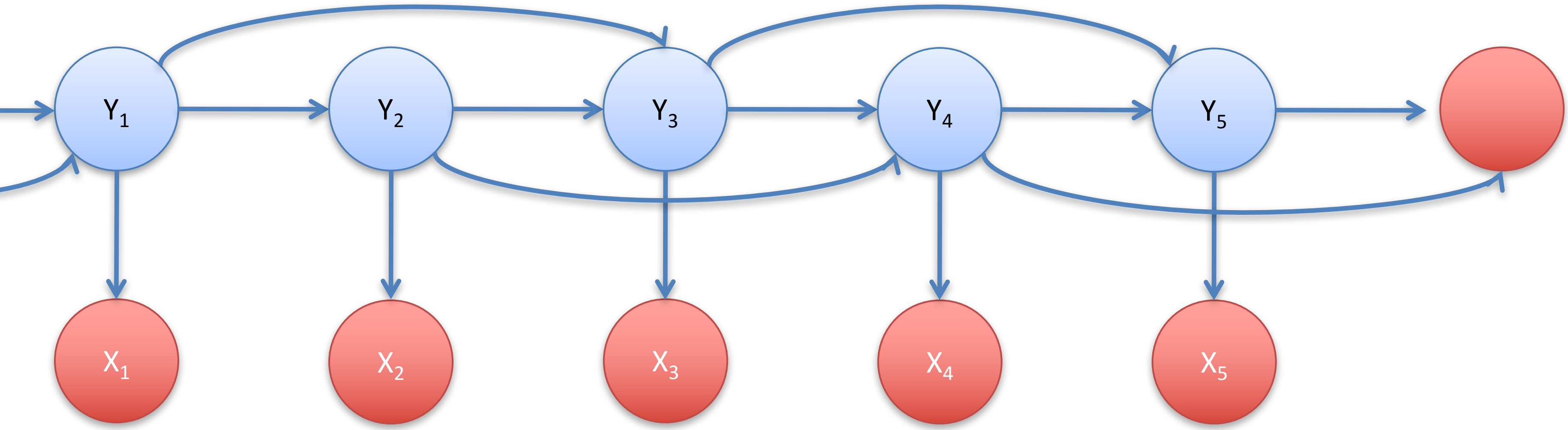
- Easy to see how to generalize HMMs.
  - More evidence
  - More factors
  - More hidden structure
  - More dependencies
- Probabilistic interpretation of factors is *not* central to finding the "best" **Y** …
  - Many factors are not conditional probability tables.

Generalization Example 1



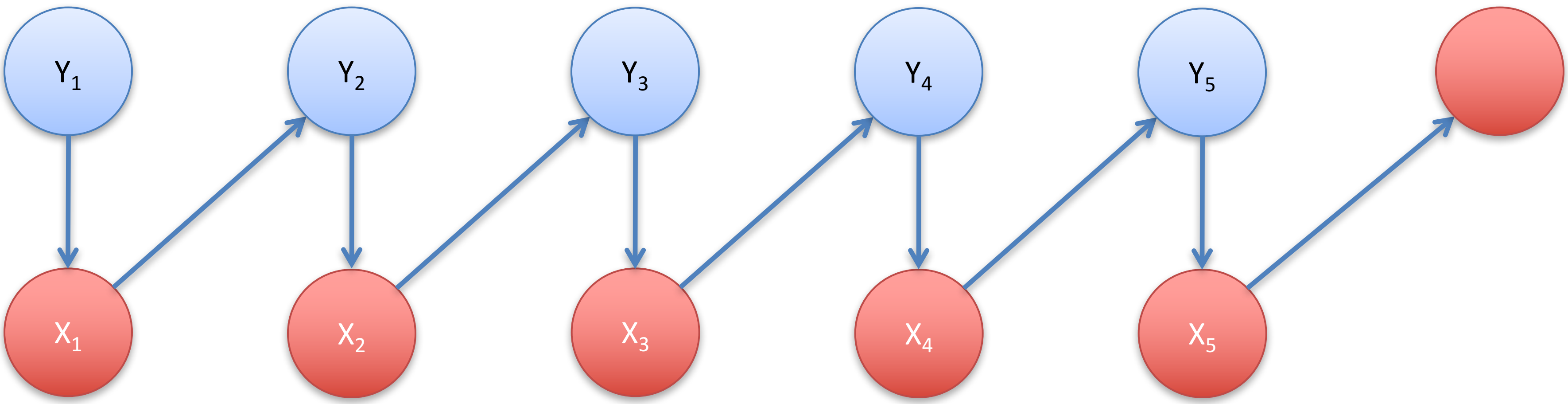
- Each word also depends on previous state.

Generalization Example 2



- "Trigram" HMM

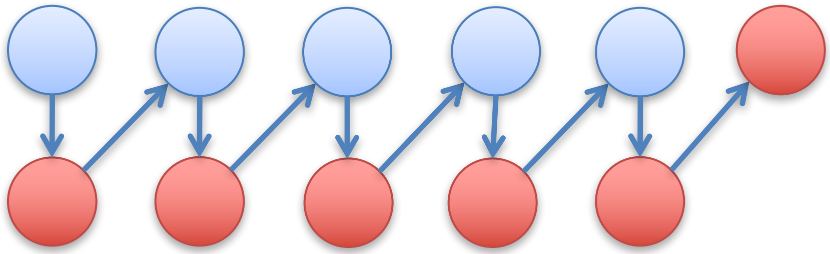
Generalization Example 3

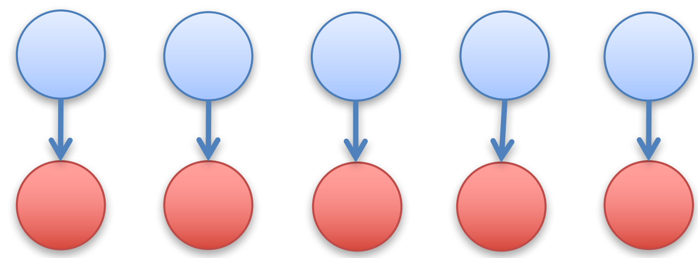


- Aggregate bigram model (Saul and Pereira, 1997)

Inference in Graphical Models

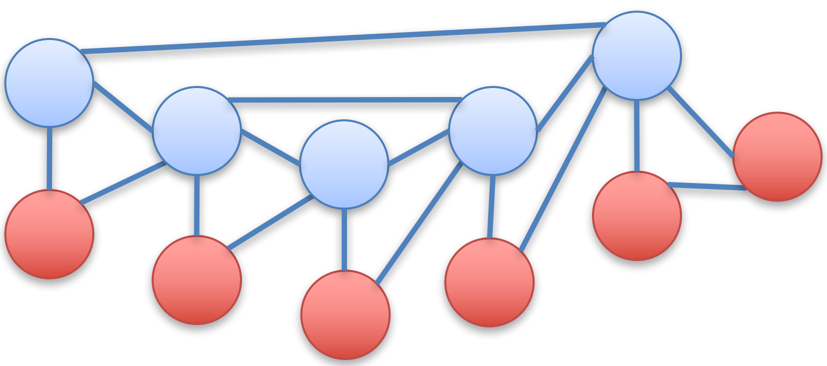
- Remember: more edges make inference more expensive.
  - Fewer edges means stronger independence.
- Really pleasant:

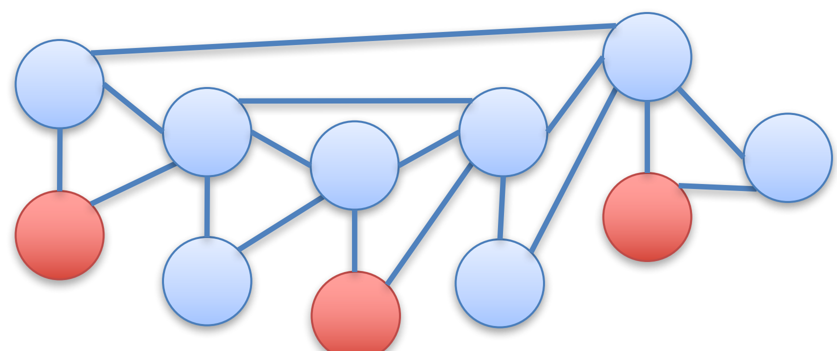




Inference in Graphical Models

- Remember: more edges make inference more expensive.
  - Fewer edges means stronger independence.
- Really unpleasant:





Decoding, Continued

September 5, 2013

Lecture Outline

- ✓ Viterbi algorithm
- ✓ Decoding more generally

3. Five views
   - ✓ MPE/MAP inference in a graphical model

2. Polytopes

“Parts”

- Assume that feature function **g** breaks down into local parts.

$$\mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) \quad = \quad \sum_{i=1}^{\#parts(\boldsymbol{x})} \mathbf{f}(\Pi_i(\boldsymbol{x}, \boldsymbol{y}))$$

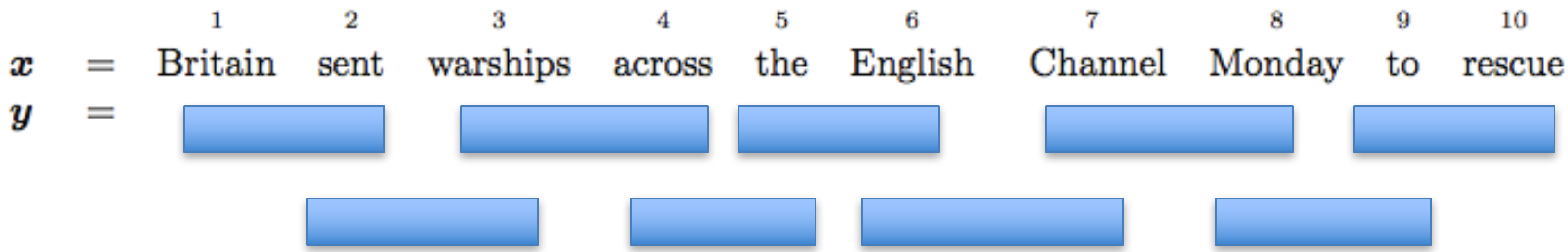
- Each part has an alphabet of possible values.
  – Decoding is choosing values for all parts, with consistency constraints.
  – (In the graphical models view, a part is a clique.)

Example

		1	2	3	4	5	6	7	8	9	10
$\boldsymbol{x}$	$=$	Britain	sent	warships	across	the	English	Channel	Monday	to	rescue
$\boldsymbol{y}$	$=$										

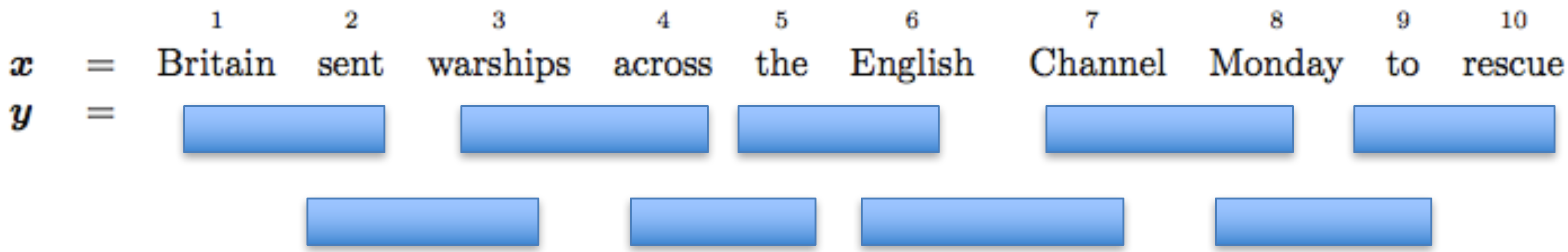
- One part per word, each is in $\{\mathrm{B}, \mathrm{I}, \mathrm{O}\}$
- No features look at multiple parts
  - Fast inference
  - Not very expressive

Example



- One part per bigram, each is in {BB, BI, BO, IB, II, IO, OB, OO}
- Features and constraints can look at pairs
  - Slower inference
  - A bit more expressive

Geometric View



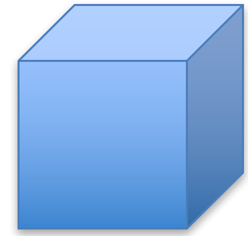
- Let $z_{i,\boldsymbol{\pi}}$ be 1 if part i takes value $\boldsymbol{\pi}$ and 0 otherwise.
- $\mathbf{z}$ is a vector in $\{0, 1\}^N$
  - N = total number of localized part values
  - Each $\mathbf{z}$ is a vertex of the unit cube

Score is Linear in $\mathbf{z}$

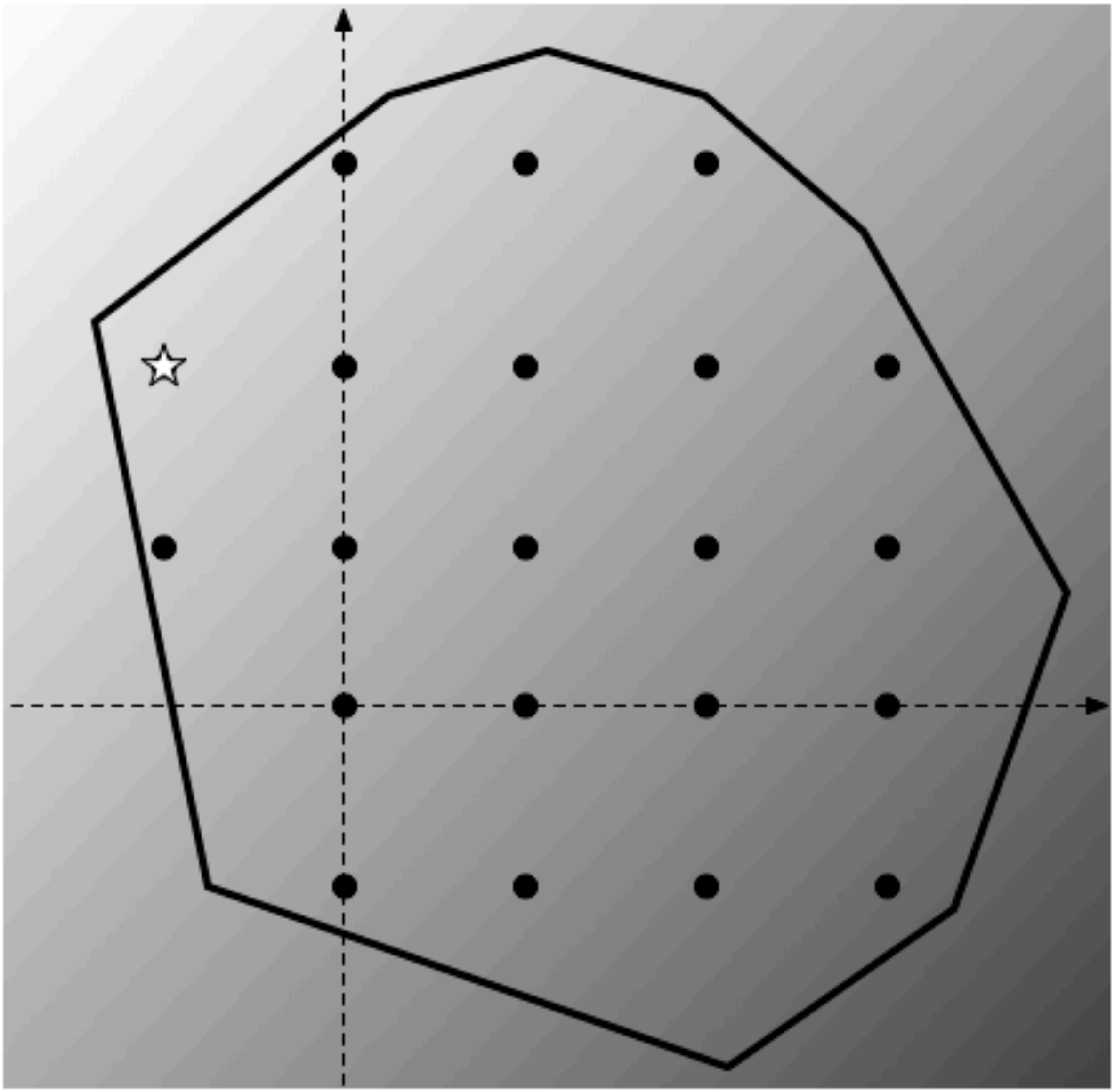
$$
\begin{aligned}
\arg\max_{\boldsymbol{y}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) &= \arg\max_{\boldsymbol{y}} \mathbf{w}^\top \sum_{i=1}^{\#parts(\boldsymbol{x})} \mathbf{f}(\Pi_i(\boldsymbol{x}, \boldsymbol{y})) \\
&= \arg\max_{\boldsymbol{y}} \mathbf{w}^\top \sum_{i=1}^{\#parts(\boldsymbol{x})} \sum_{\boldsymbol{\pi} \in \mathrm{Values}(\Pi_i)} \mathbf{f}(\boldsymbol{\pi}) \mathbf{1}\{\Pi_i(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{\pi}\} \\
&= \arg\max_{\mathbf{z} \in \mathcal{Z}_{\boldsymbol{x}}} \mathbf{w}^\top \sum_{i=1}^{\#parts(\boldsymbol{x})} \sum_{\boldsymbol{\pi} \in \mathrm{Values}(\Pi_i)} \mathbf{f}(\boldsymbol{\pi}) z_{i,\boldsymbol{\pi}} \\
&= \arg\max_{\mathbf{z} \in \mathcal{Z}_{\boldsymbol{x}}} \mathbf{w}^\top \mathbf{F}_{\boldsymbol{x}} \mathbf{z} \\
&= \arg\max_{\mathbf{z} \in \mathcal{Z}_{\boldsymbol{x}}} \left(\mathbf{w}^\top \mathbf{F}_{\boldsymbol{x}}\right) \mathbf{z}
\end{aligned}
$$

not really equal; need to transform back to get **y**

Polyhedra

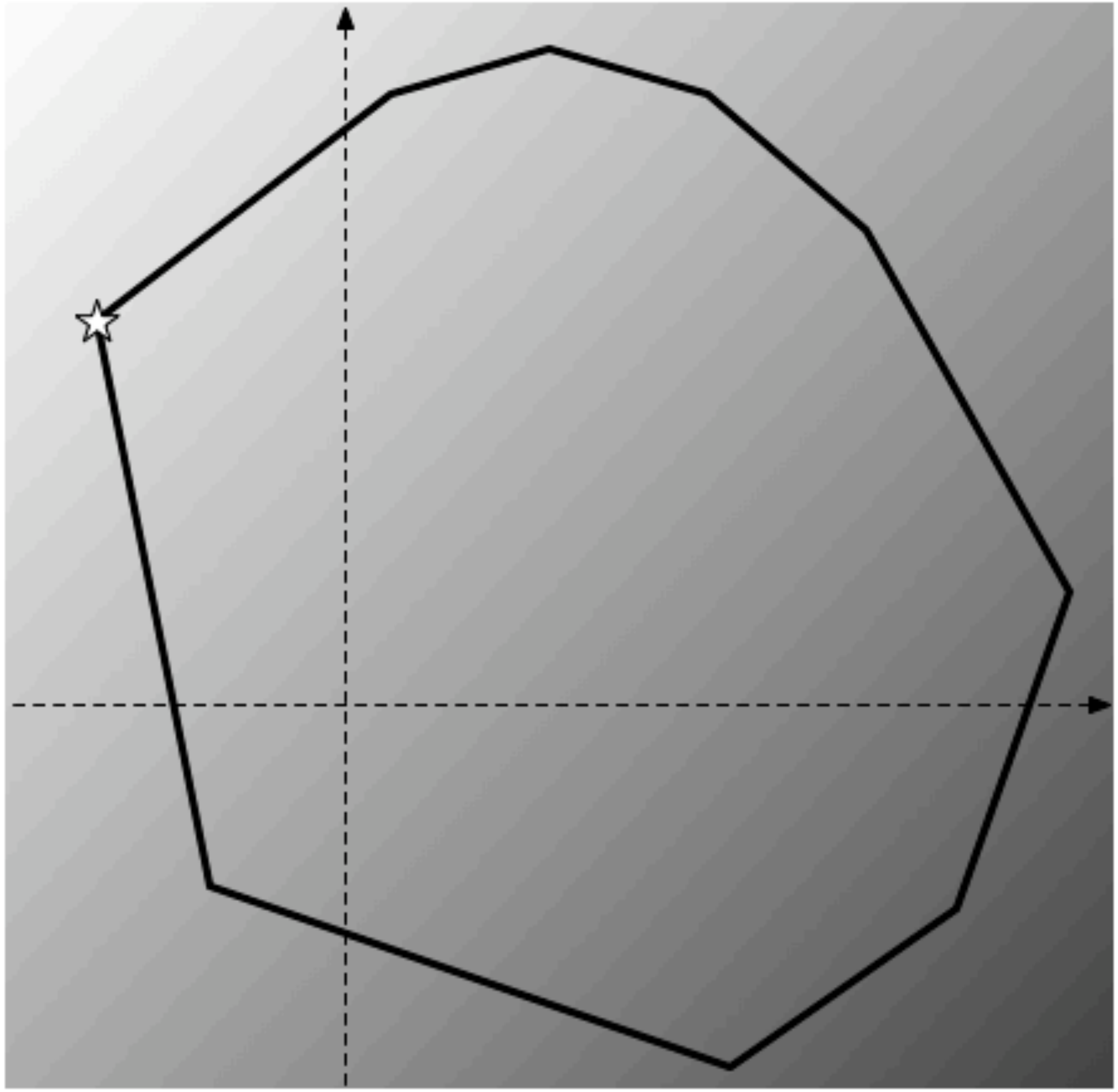


- Not all vertices of the *N*-dimensional unit cube satisfy the constraints.
  - E.g., can't have $z_{1,\mathrm{BI}} = 1$ and $z_{2,\mathrm{BI}} = 1$
- Sometimes we can write down a small (polynomial number) of linear constraints on **z**.
- Result: linear objective, linear constraints, integer constraints …



Integer Linear Programming

- Very easy to add new constraints and non-local features.
- Many decoding problems have been mapped to ILP (sequence labeling, parsing, …), but it's *not* always trivial.
- NP-hard in general.
  - But there are packages that often work well in practice (e.g., CPLEX)
  - Specialized algorithms in some cases
  - LP relaxation for approximate solutions



Remark

- Graphical models assumed a probabilistic interpretation
  - Though they are not always learned using a probabilistic interpretation!

- The polytope view is agnostic about how you interpret the weights.
  - It only says that the decoding problem is an ILP.

3. Weighted Parsing

Grammars

- Grammars are often associated with natural language parsing, but they are extremely powerful for imposing constraints.
- We can add weights to them.
  - HMMs are a kind of weighted regular grammar (closely connected to WFSAs)
  - PCFGs are a kind of weighted CFG
  - Many, many more.
- Weighted parsing: find the maximum-weighted derivation for a string **x**.

Decoding as Weighted Parsing

- Every valid **y** is a grammatical derivation (parse) for **x**.
  - HMM: sequence of "grammatical" states is one allowed by the transition table.
- Augment parsing algorithms with weights and find the best parse.

The Viterbi algorithm is an instance of recognition by a weighted grammar!

BIO Tagging as a CFG

$$
\begin{array}{lcl lcl lcl lcl}
N_{\Rightarrow} & \rightarrow & B\ R_B & R_B & \rightarrow & B\ R_B & R_I & \rightarrow & B\ R_B & R_O & \rightarrow & B\ R_B \\
N_{\Rightarrow} & \rightarrow & O\ R_O & R_B & \rightarrow & O\ R_O & R_I & \rightarrow & O\ R_O & R_O & \rightarrow & O\ R_O \\
 & & & R_B & \rightarrow & I\ R_I & R_I & \rightarrow & I\ R_I & & & \\
 & & & R_B & \rightarrow & \epsilon & R_I & \rightarrow & \epsilon & R_O & \rightarrow & \epsilon \\
 & & & & & & & & & & & \\
 & & \forall x \in \Sigma, & B & \rightarrow & x & I & \rightarrow & x & O & \rightarrow & x
\end{array}
$$

- Weighted (or probabilistic) CKY is a dynamic programming algorithm very similar in structure to classical CKY.

4. Paths and Hyperpaths

Best Path

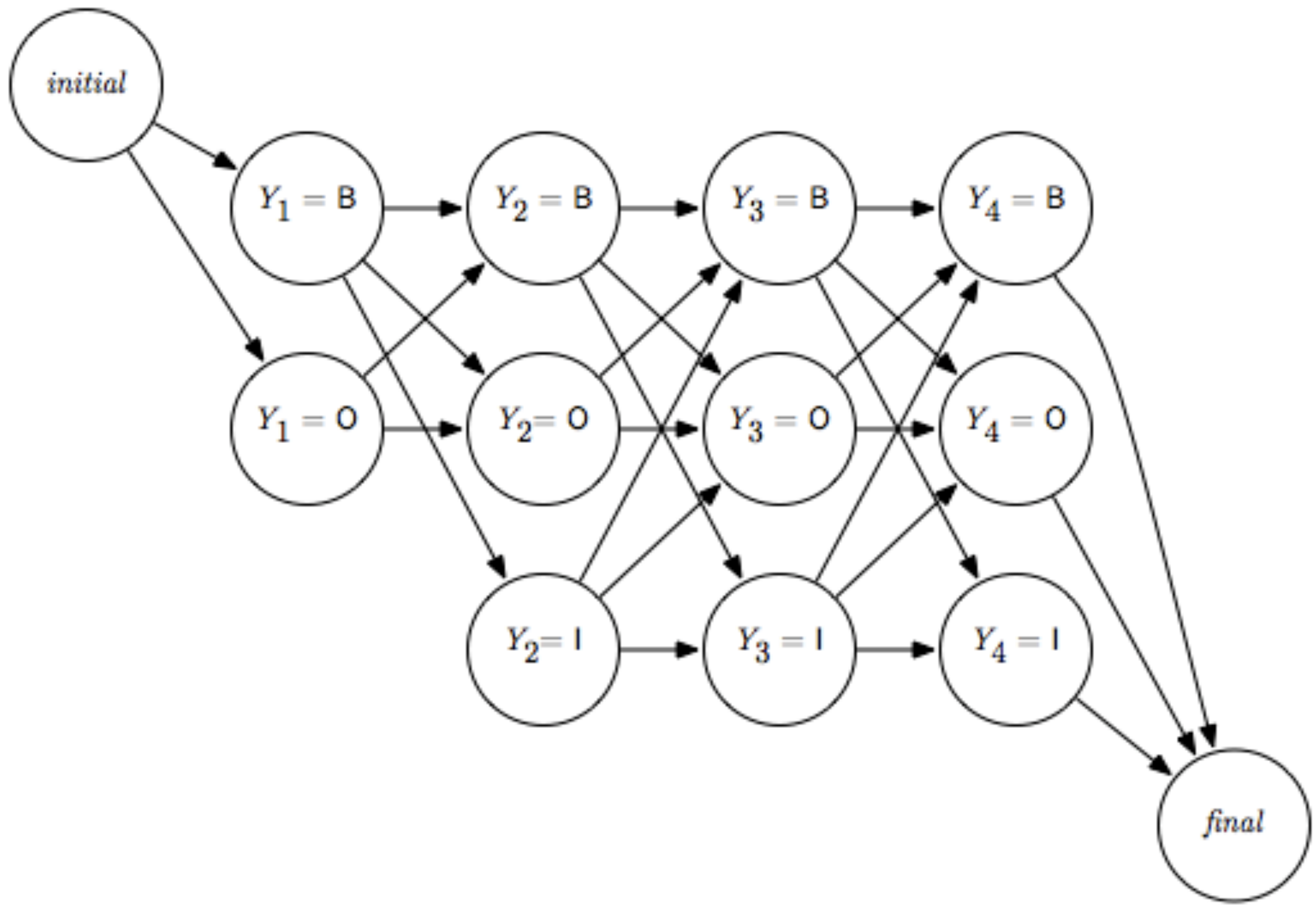
- General idea: take $\boldsymbol{x}$ and build a graph.
- Score of a path factors into the edges.

$$\arg\max_{\boldsymbol{y}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) \quad = \quad \arg\max_{\boldsymbol{y}} \mathbf{w}^\top \sum_{e \in \text{Edges}} \mathbf{f}(e) \mathbf{1}\{e \text{ is crossed by } \boldsymbol{y}\text{'s path}\}$$

- Decoding is finding the *best* path.

The Viterbi algorithm is an instance of finding a best path!

“Lattice” View of Viterbi



A Generic Best Path Algorithm

- Input: directed graph $G = (V, E)$, $\text{cost} : E \rightarrow \mathbb{R}$, start vertex v_0
- Output: $d : V \rightarrow \mathbb{R}$ (shortest path function) and back pointers $b : V \rightarrow V$

for all $v \in V \setminus \{v_0\}$, $d(v) := \infty$ and $b(v) := \varnothing$

set $d(v_0) = 0$

while d has not converged:

    pick an arbitrary edge (u, v)

    if $d(u) + \text{cost}(u, v) < d(v)$:

        $d(v) := d(u) + \text{cost}(u, v)$

        $b(v) := u$

Ordering Updates

- Naïve ways of choosing edges will lead to cyclic updating and gross inefficiency!
- Before considering various ways of doing it, let's consider how the Viterbi algorithm is essentially solving the same problem.

Viterbi Algorithm (In the Style of A Best Path Algorithm)

- Input:
  - directed graph $G = (V, E)$ where
    each vertex $v = (q, t)$, $q \in Q \cup \{\varnothing\}$, $t \in \{-1, 0, 1, \ldots, n\}$
    and each edge $(u, v) = ((q, t), (q', t + 1))$
  - cost : $E \to \mathbb{R}$, defined by
    $\text{cost}((q, t), (q', t + 1)) = -\log \gamma(q' \mid q) - \log \eta(s_{t+1} \mid q) - \log (1 - \xi(q))$
    $\text{cost}((q, n - 1), (q', n)) = -\log \gamma(q' \mid q) - \log \eta(s_{t+1} \mid q) - \log \xi(q')$
    $\text{cost}((\varnothing, -1), (q, 0)) = -\log \pi(q)$
  - fixed start vertex $v_0 = (\varnothing, -1)$
- Output: $d : V \to \mathbb{R}$ (shortest path function) and back pointers $b : V \to V$

for all $v \in V \setminus \{v_0\}$, $d(v) := \infty$ and $b(v) := \varnothing$
set $d(v_0) = 0$
perform a topological sort on V
~~while d has not converged:~~ for each v in top-sort order:
    ~~pick an arbitrary edge (u, v)~~
    for each $(u, v) \in E$:
        if $d(u) + \text{cost}(u, v) < d(v)$:
            $d(v) := d(u) + \text{cost}(u, v)$
            $b(v) := u$
    // d(v) and b(v) are now known

The Viterbi Trick

- From a “best path” perspective, Viterbi is:
  - defining the vertices and edges to have special structure (state/time step)
  - assigning costs based on HMM weights and the specific input string $s_1 \ldots s_n$
  - ordering the edges cleverly to make things efficient
- Note also: Viterbi's graph has no cycles.

Another Variant: "Forward" Updating

- After topological sort, can also choose all edges going *out of* current node.

for all $v \in V \setminus \{v_0\}$, $d(v) := \infty$ and $b(v) := \varnothing$
set $d(v_0) = 0$
perform a topological sort on V
for each u in top-sort order:
    for each $(u, v) \in E$:
      if $d(u) + cost(u, v) < d(v)$:
          $d(v) := d(u) + cost(u, v)$
          $b(v) := u$

Memoized Recursion

- Input: directed graph $G = (V, E)$, cost : $E \rightarrow \mathbb{R}$, start vertex v_0, target vertex v_t
- Output: $d : V \rightarrow \mathbb{R}$ (shortest path function) and back pointers $b : V \rightarrow V$

```
for all v ∈ V \ {v0}, d(v) := ∅ and b(v) := ∅
set d(v0) = 0
memoize(vt)

memoize(v):
   // guaranteed to return best-cost path score for v
   if d(v) = ∅:
     d(v) := ∞
     for each (u, v) ∈ E:
          if memoize(u) + cost(u, v) < d(v):
               d(v) := d(u) + cost(u, v)
               b(v) := u
   return d(v)
```

A Generic Best Path Algorithm

- Input: directed graph $G = (V, E)$, cost : $E \rightarrow \mathbb{R}$, start vertex v_0
- Output: $d : V \rightarrow \mathbb{R}$ (shortest path function) and back pointers $b : V \rightarrow V$

for all $v \in V \setminus \{v_0\}$, $d(v) := \infty$ and $b(v) := \varnothing$

set $d(v_0) = 0$

while d has not converged:

  pick an arbitrary edge (u, v)

  if $d(u) + \text{cost}(u, v) < d(v)$:

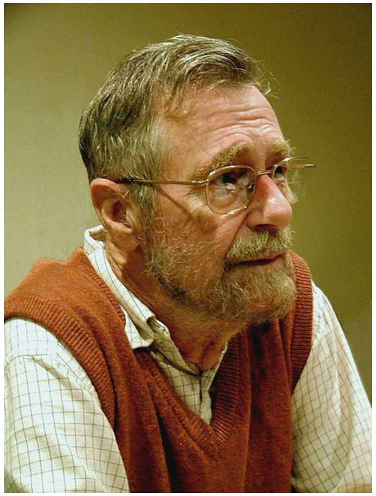
   $d(v) := d(u) + \text{cost}(u, v)$

   $b(v) := u$

Dijkstra's Algorithm

- Input: directed graph $G = (V, E)$, cost : $E \to \mathbb{R}_{\geq 0}$ (important!), start vertex v_0
- Output: $d : V \to \mathbb{R}$ (shortest path function) and back pointers $b : V \to V$

for all $v \in V \setminus \{v_0\}$, $d(v) := \infty$ and $b(v) := \varnothing$
set $d(v_0) = 0$
Q := priority queue on V ordered by d (lower cost = higher priority)
~~while d has not converged:~~ while Q is not empty:
    ~~pick an arbitrary edge (u, v)~~
    u := extract-min(Q)
    for each $(u, v) \in E$:
      if $d(u) + \text{cost}(u, v) < d(v)$:
        $d(v) := d(u) + \text{cost}(u, v)$
        $b(v) := u$
        update v's priority in Q



A* Algorithm

- Input: directed graph $G = (V, E)$, cost : $E \to \mathbb{R}_{\geq 0}$, start vertex v_0, target vertex v_t, heuristic $h : V \to \mathbb{R}_{\geq 0}$ such that $h(v) \leq$ best-cost(v, v_t)
- Output: $d : V \to \mathbb{R}$ (shortest path function) and back pointers $b : V \to V$

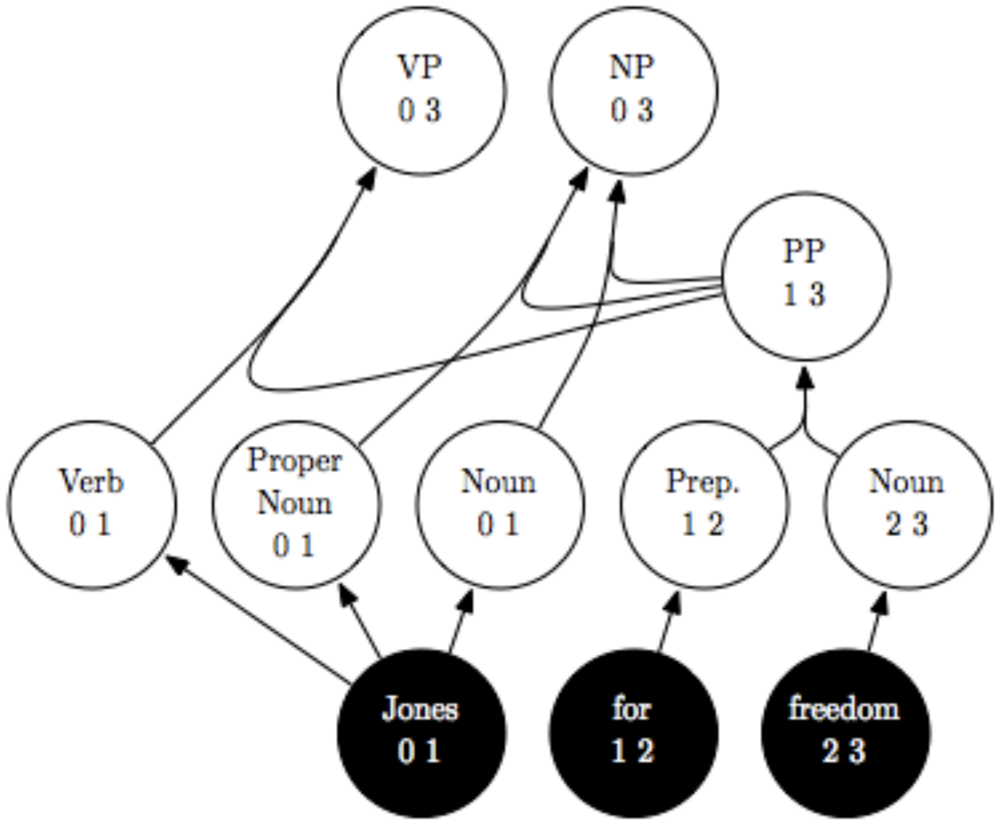
```
for all v ∈ V \ {v0}, d(v) := ∞ and b(v) := ∅
set d(v0) = 0
Q := priority queue on V ordered by d + h (lower cost = higher priority)
while Q is not empty:
    u := extract-min(Q)
    for each (u, v) ∈ E:
      if d(u) + cost(u, v) < d(v):
          d(v) := d(u) + cost(u, v)
          b(v) := u
          update v's priority in Q
```

Minimum Cost Hyperpath

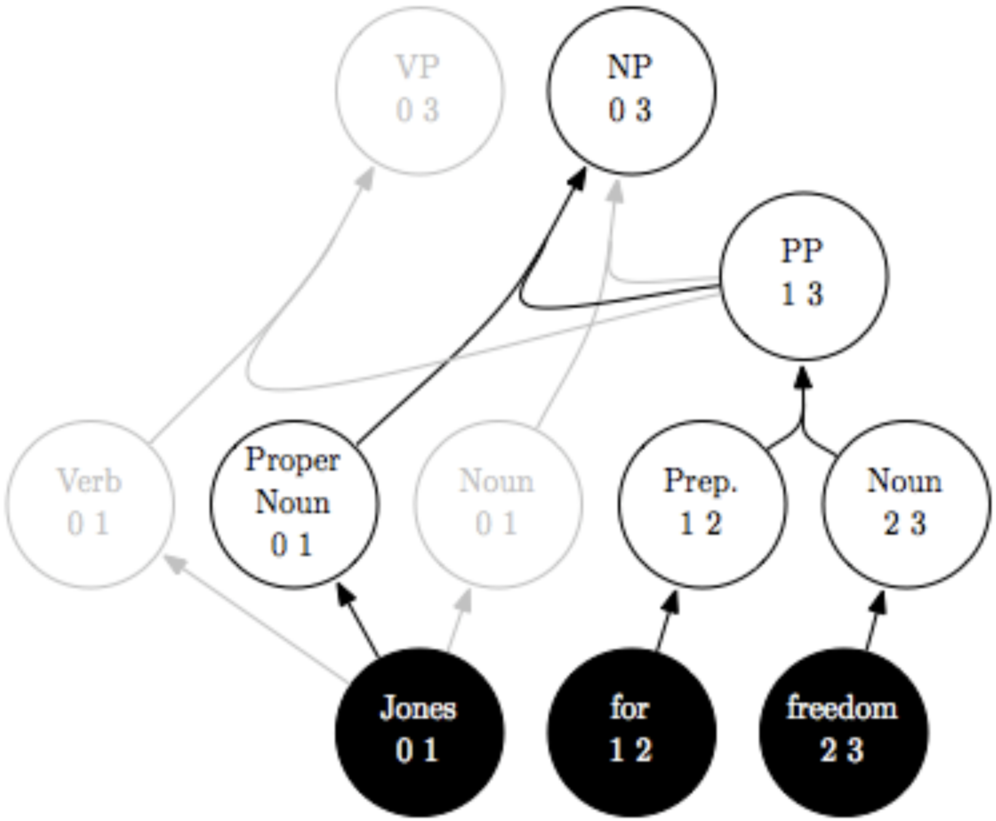
- General idea: take ***x*** and build a hypergraph.
- Score of a hyperpath factors into the hyperedges.
- Decoding is finding the best *hyperpath*.

- This connection was elucidated by Klein and Manning (2002).

Parsing as a Hypergraph

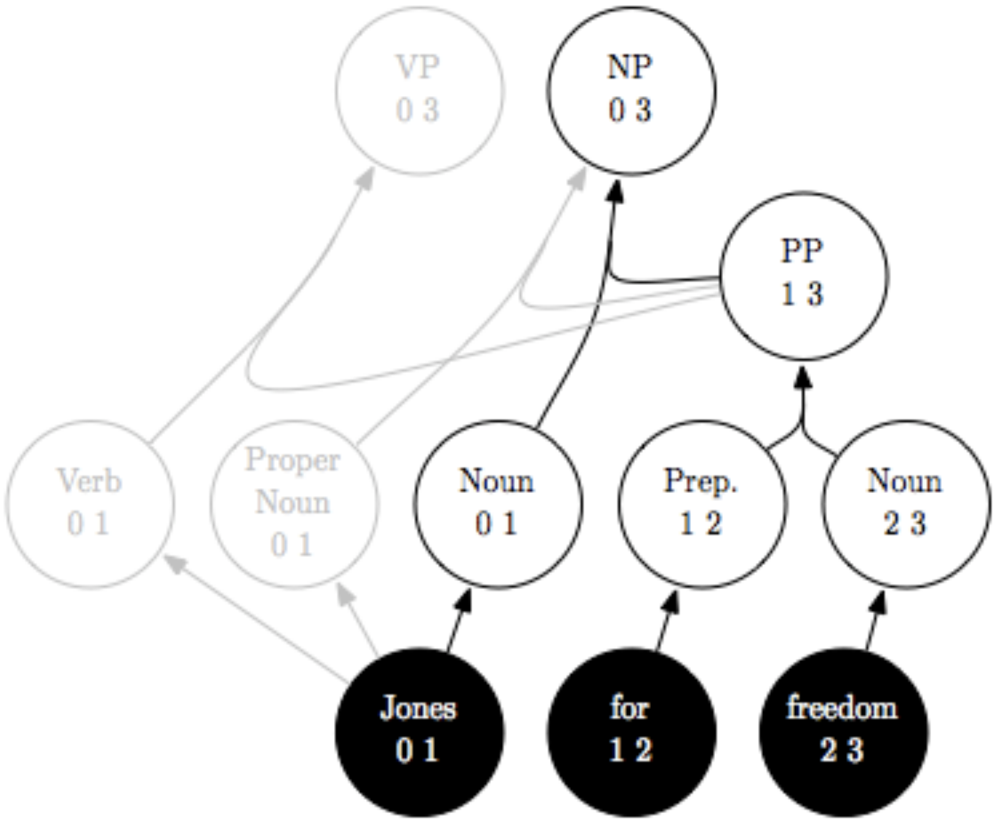


Parsing as a Hypergraph



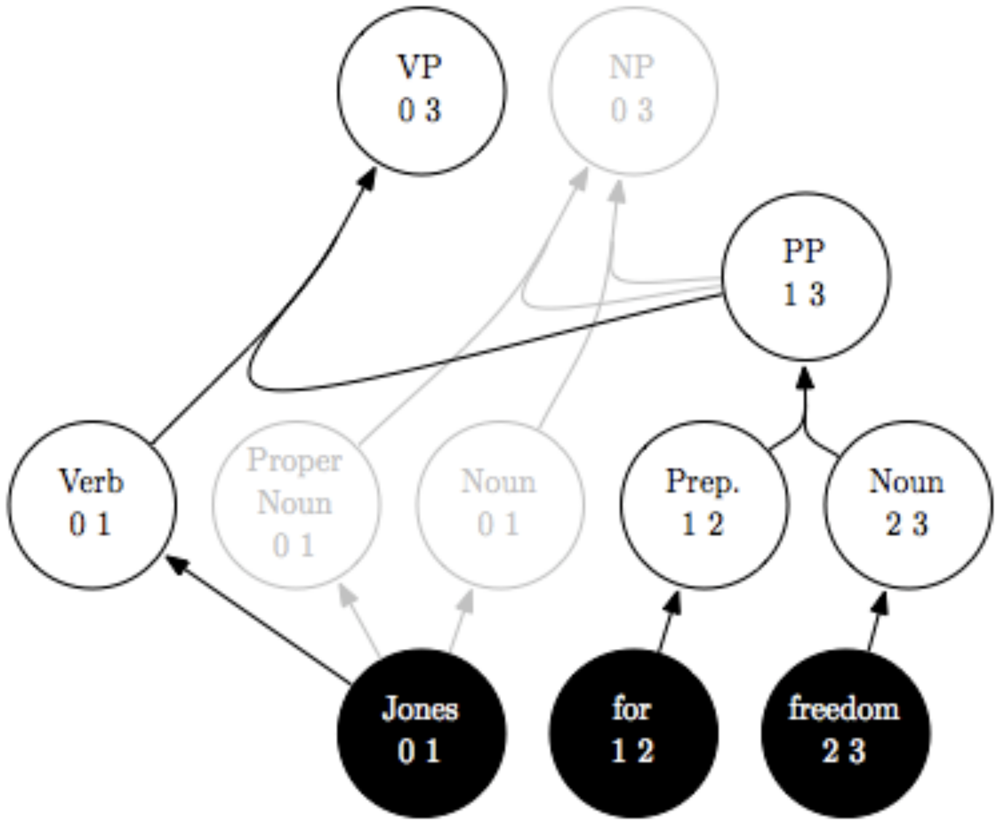
cf. “Dean for democracy”

Parsing as a Hypergraph



Forced to work on his thesis, sunshine streaming in the window, Mike experienced a …

Parsing as a Hypergraph



Forced to work on his thesis, sunshine streaming in the window, Mike began to …

Why Hypergraphs?

- Useful, compact encoding of the hypothesis space.
  - Build hypothesis space using local features, maybe do some filtering.
  - Pass it off to another module for more fine-grained scoring with richer or more expensive features.

5. Weighted Logic Programming

Logic Programming

- Start with a set of axioms and a set of inference rules.

$$\begin{array}{lrcl} \forall A, C, & \text{ancestor}(A, C) & \Leftarrow & \text{parent}(A, C) \\ \forall A, C, & \text{ancestor}(A, C) & \Leftarrow & \bigvee_B \text{ancestor}(A, B) \wedge \text{parent}(B, C) \end{array}$$

- The goal is to prove a specific theorem, goal.
- Many approaches, but we assume a *deductive* approach.
  - Start with axioms, iteratively produce more theorems.

$$
\begin{array}{rl}
 & \mathsf{label\text{-}bigram}(\text{“B”}, \text{“B”}) \\
 & \mathsf{label\text{-}bigram}(\text{“B”}, \text{“I”}) \\
 & \mathsf{label\text{-}bigram}(\text{“B”}, \text{“O”}) \\
 & \mathsf{label\text{-}bigram}(\text{“I”}, \text{“B”}) \\
 & \mathsf{label\text{-}bigram}(\text{“I”}, \text{“I”}) \\
 & \mathsf{label\text{-}bigram}(\text{“I”}, \text{“O”}) \\
 & \mathsf{label\text{-}bigram}(\text{“O”}, \text{“B”}) \\
 & \mathsf{label\text{-}bigram}(\text{“O”}, \text{“O”}) \\
\forall x \in \Sigma, & \mathsf{labeled\text{-}word}(x, \text{“B”}) \\
\forall x \in \Sigma, & \mathsf{labeled\text{-}word}(x, \text{“I”}) \\
\forall x \in \Sigma, & \mathsf{labeled\text{-}word}(x, \text{“O”})
\end{array}
$$

$$
\begin{array}{rcl}
\forall \ell \in \Lambda, \ \ \mathsf{v}(\ell, 1) & = & \mathsf{labeled\text{-}word}(x_1, \ell) \\
\forall \ell \in \Lambda, \ \ \mathsf{v}(\ell, i) & = & \bigvee_{\ell' \in \Lambda} \mathsf{v}(\ell', i-1) \wedge \mathsf{label\text{-}bigram}(\ell', \ell) \wedge \mathsf{labeled\text{-}word}(x_i, \ell) \\
\mathsf{goal} & = & \bigvee_{\ell \in \Lambda} \mathsf{v}(\ell, n)
\end{array}
$$

Weighted Logic Programming

- Twist: axioms have weights.
- Want the proof of goal with the best score:

$$\arg\max_{\boldsymbol{y}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) \quad = \quad \arg\max_{\boldsymbol{y}} \mathbf{w}^\top \sum_{a \in \text{Axioms}} \mathbf{f}(a) \mathit{freq}(a; \boldsymbol{y})$$

- Note that axioms can be used more than once in a proof (**y**).

Whence WLP?

- Shieber, Schabes, and Pereira (1995): many parsing algorithms can be understood in the same deductive logic framework.
- Goodman (1999): add weights in a semiring, get many useful NLP algorithms.
- Eisner, Goldlust, and Smith (2004, 2005): semiring-generic algorithms, Dyna.

Dynamic Programming

- Most views (exception is polytopes) can be understood as DP algorithms.
  - The low-level *procedures* we use are often DP.
  - Even DP is too high-level to know the best way to implement.
- Break a problem into slightly smaller problems with **optimal substructure**.
  - Best path to v depends on best paths to all u such that $(u,v) \in E$.
- **Overlapping subproblems**: each subproblem gets used repeatedly, and there aren't too many of them.

Dynamic Programming

- Three main strategies for DP:
  - Viterbi, Levenshtein edit distance, CKY: predefined, “clever” ordering.
  - Memoization
  - Agenda (Dijkstra’s algorithm, A*)
- Things to remember in general:
  - The hypergraph may too big to represent explicitly; exhaustive calculation may be too expensive.
  - The hypergraph may or may not have properties that make “clever” orderings possible.
  - DP does *not* imply polynomial time and space! Most common approximations when the desired state space is too big: beam search, cube pruning, agendas with early stopping, ...

Summary

- Decoding is the general problem of choosing a complex structure.
  - Linguistic analysis, machine translation, speech recognition, …
  - Statistical models are usually involved (not necessarily probabilistic).
- No perfect general view, but much can be gained through a combination of views.
- First question: can I solve it exactly with DP?