CS769 Advanced NLP

Morphology & Sequence Labeling

Junjie Hu

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Slides adapted from Luke, Yulia, Bob

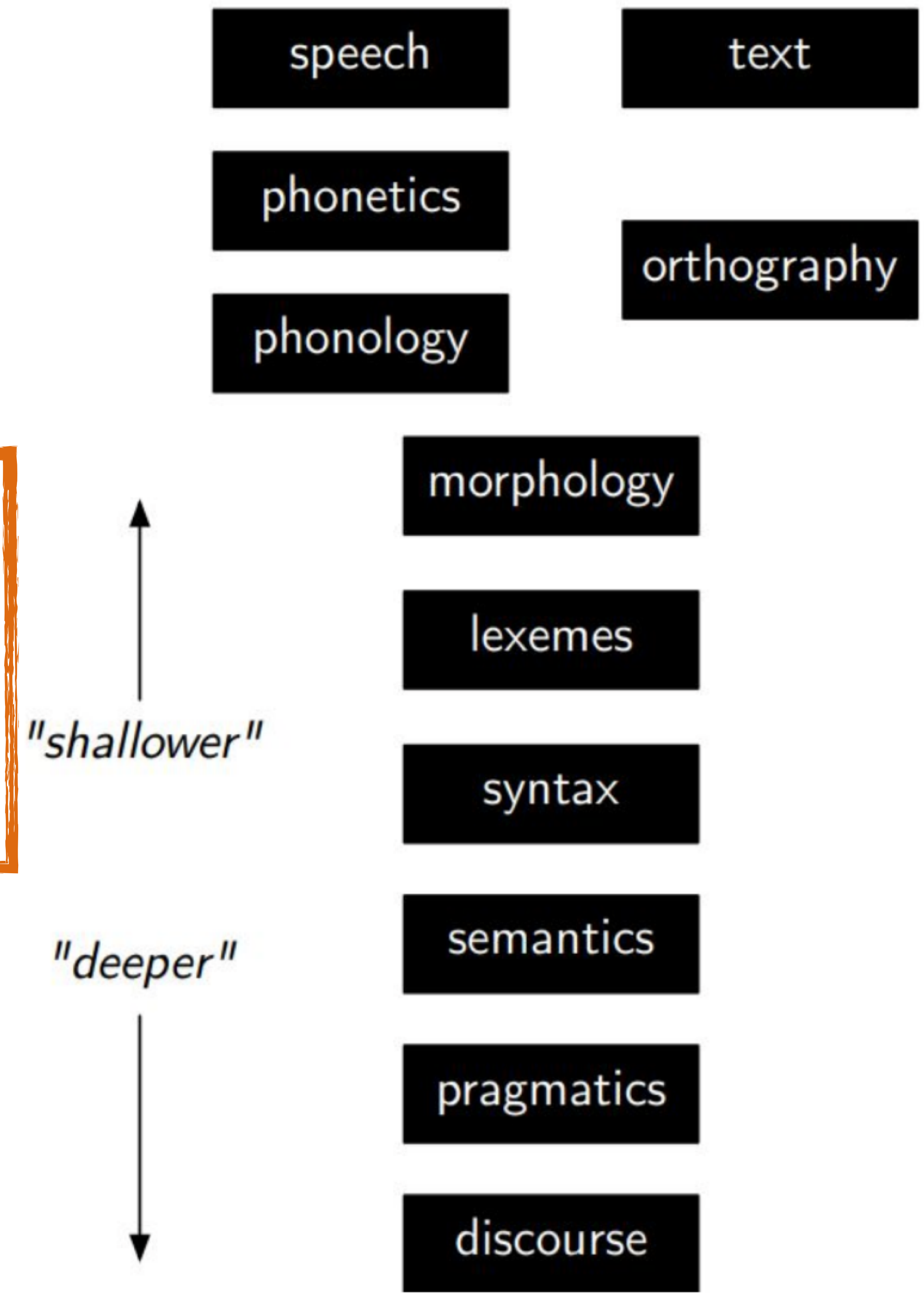
https://junjiehu.github.io/cs769-fall23/

Goals for Today

- **Subword** Tokenization: BPE
- (**Generative**) Sequence Labeling: **Hidden Markov Model**
- (**Discriminative**) Sequence Labeling: **Conditional Random Field**

Levels of Linguistic Knowledge

Phonetics	The study of the sounds of human language
Phonology	The study of sound systems in human language
Morphology	The study of the formation and internal structure of words
Syntax	The study of the formation and internal structure of sentences
Semantics	The study of the meaning of sentences
Pragmatics	The study of the way sentences with their semantic meanings are used for particular communicative goals



Morphology
& Word Tokenization

Tokenization (Example)

Input raw text

```
Dr. Smith said tokenization of English is “harder than you’ve thought.”
When in New York, he paid $12.00 a day for lunch and wondered what it would
be like to work for AT&T or Google, Inc.
```

Output from Stanford Parser with Part-of-Speech tags: http://nlp.stanford.edu:8080/parser/index.jsp

```
Dr./NNP Smith/NNP said/VBD tokenization/NN of/IN English/NNP
is/VBZ ``/`` harder/JJR than/IN you/PRP 've/VBP thought/VBN ./.
''/''
When/WRB in/IN New/NNP York/NNP ,/, he/PRP paid/VBD $/$ 12.00/CD
a/DT day/NN for/IN lunch/NN and/CC wondered/VBD what/WP it/PRP
would/MD be/VB like/JJ to/TO work/VB for/IN AT&T/NNP or/CC
Google/NNP ,/, Inc./NNP ./.
```

Subword Tokenization

- Neural systems typically use a **relatively small fixed** vocabulary
- Real world contains many words
  - New words all the time
  - For morphologically rich languages, even more so
  - But most words are rare (Zipf's Law)
- Note that rare words do not have good corpus statistics
- So, tokenize words into more frequent subword segments

Unsupervised Subword Algorithms

- Use the data to tell us how to tokenize
- Three common algorithms:
  - **Byte-Pair Encoding (BPE)** [Sennrich et al., 2016]
  - **WordPiece** [Schuster and Nakajima, 2012]
  - **Unigram language modeling tokenization** (Unigram) [Kudo, 2018]
- Learnable tokenizer:
  - Training: takes a raw training corpus and induces a vocabulary
  - Segmentation: tokenizes a raw test sentence according to the induced vocabulary

BPE: https://github.com/rsennrich/subword-nmt
SentencePiece: https://github.com/google/sentencepiece

Byte-Pair Encoding

- Add a special end-of-word symbol "▁" (U+2581) or </w> at the end of each word in training corpus
- Convert words into a set of characters, create an initial vocabulary
- Iteratively merge the most frequent pair of adjacent tokens for k times

```
function BYTE-PAIR ENCODING(strings C, number of merges k) returns vocab V

  V ← all unique characters in C             # initial set of tokens is characters
  for i = 1 to k do                          # merge tokens til k times
    t_L, t_R ← Most frequent pair of adjacent tokens in C
    t_NEW ← t_L + t_R                        # make new token by concatenating
    V ← V + t_NEW                            # update the vocabulary
    Replace each occurrence of t_L, t_R in C with t_NEW    # and update the corpus
  return V
```

Byte-Pair Encoding (Example)

Example — training corpus:

low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new

low_ low_ low_ low_ low_ lowest_ lowest_ newer_ newer_ newer_ newer_ newer_ newer_ wider_ wider_ wider_ new_ new_

corpus

5	l o w _
2	l o w e s t _
6	n e w e r _
3	w i d e r _
2	n e w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w

Byte-Pair Encoding (Example)

corpus		**vocabulary**
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

Merge e r to er

corpus		**vocabulary**
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2	l o w e s t _	
6	n e w er _	
3	w i d er _	
2	n e w _	

Byte-Pair Encoding (Example)

corpus		**vocabulary**
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2	l o w e s t _	
6	n e w er _	
3	w i d er _	
2	n e w _	

Merge er _ to er_

corpus		**vocabulary**
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

Byte-Pair Encoding (Example)

corpus		**vocabulary**
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

Merge n e to ne

corpus		**vocabulary**
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne
2	l o w e s t _	
6	ne w er_	
3	w i d er_	
2	ne w _	

Byte-Pair Encoding (Example)

- The next merges are:

Merge	**Current Vocabulary**
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

+: Usually include frequent words,
and frequent subwords which are often morphemes, e.g., *-est* or *-er*

Syntax
& Sequence Labeling

Sequence labeling problems

- Map **a sequence of words** to **a sequence of labels**
  - Part-of-speech tagging (Church, 1988; Brants, 2000)
  - Named entity recognition (Bikel et al., 1990)
  - Text chunking and shallow parsing (Ramshaw and Marcus, 1995)
  - Word alignment of parallel text (Vogel et al., 1996)
  - Compression (Conroy and O'Leary, 2001)
  - Acoustic models, discourse segmentation, etc.

Part of Speech Tagging

- Penn treebank tagset (Marcus et al., 1993)

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coordinating conjunction	*and, but, or*	PDT	predeterminer	*all, both*	VBP	verb non-3sg present	*eat*
CD	cardinal number	*one, two*	POS	possessive ending	*'s*	VBZ	verb 3sg pres	*eats*
DT	determiner	*a, the*	PRP	personal pronoun	*I, you, he*	WDT	wh-determ.	*which, that*
EX	existential 'there'	*there*	PRP$	possess. pronoun	*your, one's*	WP	wh-pronoun	*what, who*
FW	foreign word	*mea culpa*	RB	adverb	*quickly*	WP$	wh-possess.	*whose*
IN	preposition/ subordin-conj	*of, in, by*	RBR	comparative adverb	*faster*	WRB	wh-adverb	*how, where*
JJ	adjective	*yellow*	RBS	superlatv. adverb	*fastest*	$	dollar sign	$
JJR	comparative adj	*bigger*	RP	particle	*up, off*	#	pound sign	#
JJS	superlative adj	*wildest*	SYM	symbol	+,%, &	“	left quote	‘ or “
LS	list item marker	*1, 2, One*	TO	“to”	*to*	”	right quote	’ or ”
MD	modal	*can, should*	UH	interjection	*ah, oops*	(	left paren	[, (, {, <
NN	sing or mass noun	*llama*	VB	verb base form	*eat*	)	right paren	],), }, >
NNS	noun, plural	*llamas*	VBD	verb past tense	*ate*	,	comma	,
NNP	proper noun, sing.	*IBM*	VBG	verb gerund	*eating*	.	sent-end punc	. ! ?
NNPS	proper noun, plu.	*Carolinas*	VBN	verb past part.	*eaten*	:	sent-mid punc	: ; ... – -

POS tagging (Example)

- System outputs:
  - The/DT grand/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.
  - There/EX are/VBP 70/CD children/NNS there/RB
  - Preliminary/JJ findings/NNS were/VBD reported/VBN in/IN today/NN 's/POS New/NNP England/NNP Journal/NNP of/IN Medicine/NNP ./.

Universal Dependencies for All Languages

Universal Dependencies

Universal Dependencies (UD) is a framework for consistent annotation of grammar (parts of speech, morphological features, and syntactic dependencies) across different human languages. UD is an open community effort with over 300 contributors producing more than 150 treebanks in 90 languages. If you're new to UD, you should start by reading the first part of the Short Introduction and then browsing the annotation guidelines.

- Short introduction to UD
- UD annotation guidelines
- More information on UD:
  - How to contribute to UD
  - Tools for working with UD
  - Discussion on UD
  - UD-related events
- Query UD treebanks online:
  - SETS treebank search maintained by the University of Turku
  - PML Tree Query maintained by the Charles University in Prague
  - Kontext maintained by the Charles University in Prague
  - Grew-match maintained by Inria in Nancy
  - INESS maintained by the University of Bergen
- Download UD treebanks

Open class words	Closed class words	Other
ADJ	ADP	PUNCT
ADV	AUX	SYM
INTJ	CCONJ	X
NOUN	DET	
PROPN	NUM	
VERB	PART	
	PRON	
	SCONJ	

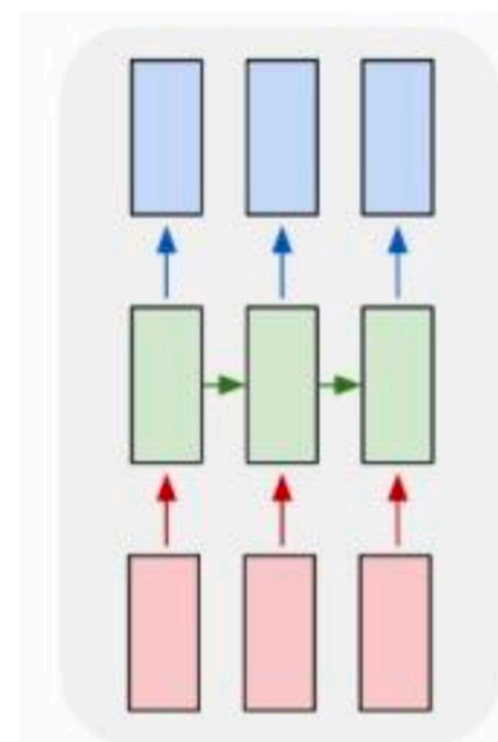
Sequence labeling as text classification

- **Generative** Model: Learn joint probability $P(X, Y)$
  - Hidden Markov Models

$$\hat{Y} = \arg \max_{\substack{y_1 \cdots y_n \\ \forall y_i \in \mathcal{C}}} P(x_1 \cdots x_n, y_1 \cdots y_n)$$

- **Discriminative** Model: Learn conditional probability $P(Y|X)$
  - Conditional Random Fields
  - Neural network-based methods

$$\hat{Y} = \arg \max_{Y} P(Y|X)$$



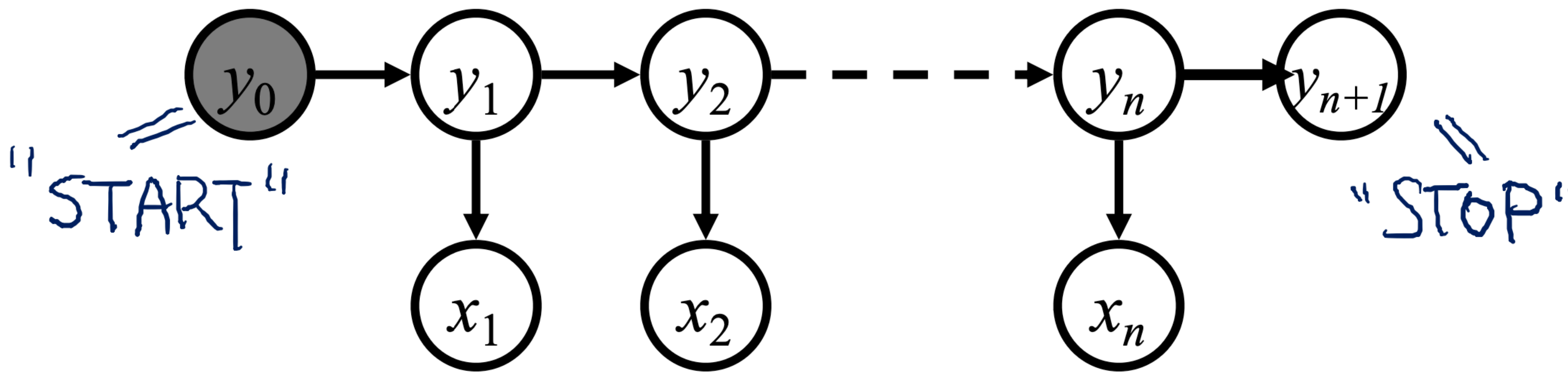
- Both trained via Maximum Likelihood Estimation

Hidden Markov Model

(Sequential Version of Naive Bayes)

Classic Solution: HMMs

- We want a model of unobservable (hidden) sequences y and observations x



$$p(x_1 ... x_n, y_1 ... y_{n+1}) = q(\text{STOP}|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

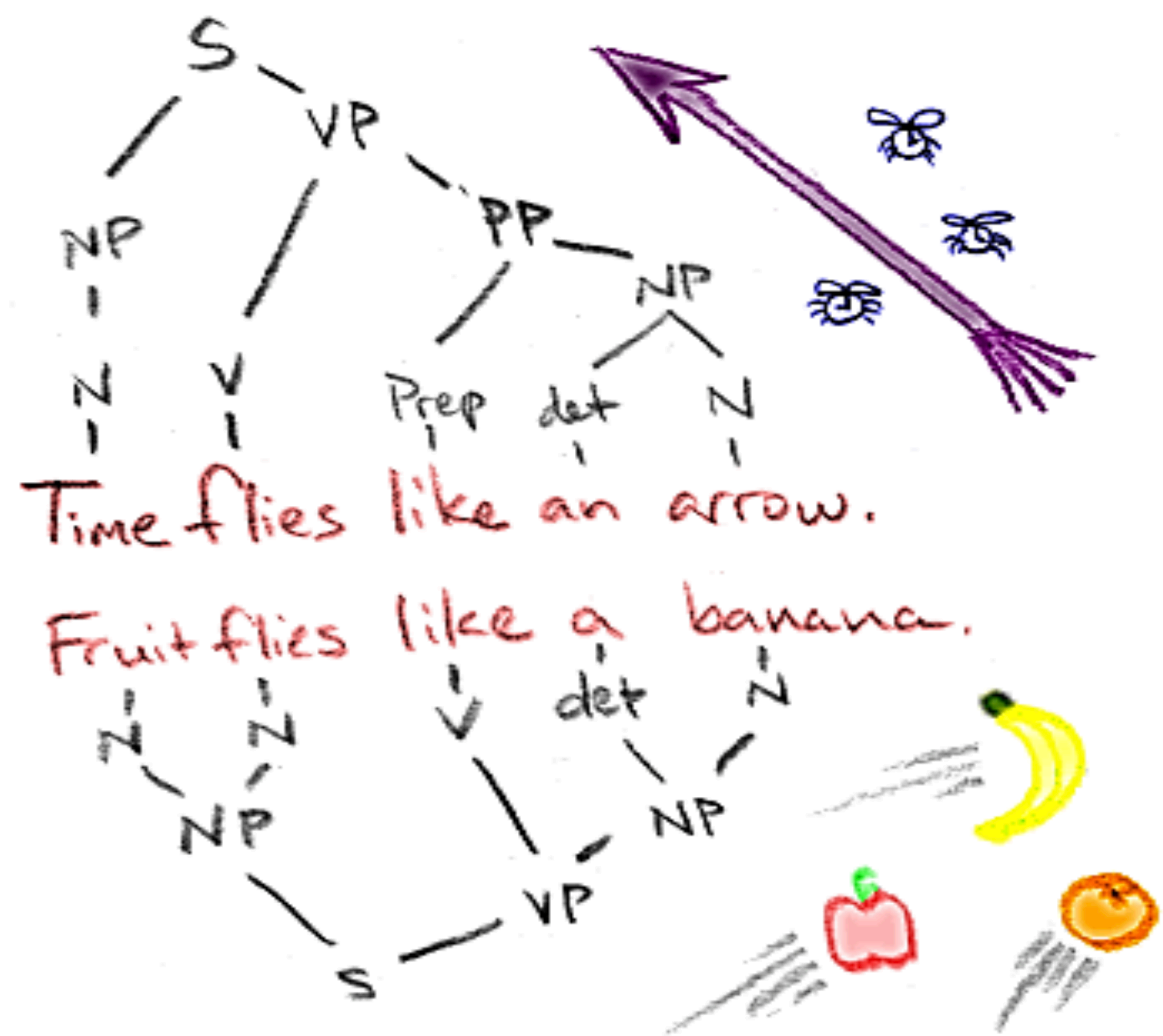
where $y_0 = \text{START}$ and we call $q(y'|y)$ the transition distribution and $e(x|y)$ the emission (or observation) distribution.

Assumptions:

- Tag/state sequence is generated by a Markov model
- Words are chosen independently, conditioned only on the tag/state
- These are totally broken assumptions: why?

Tag predictions depends on context

- Time flies like an arrow
- Fruit flies like a banana



HMM Learning and Inference

- **Learning** by **maximum likelihood estimation**: transition $q(y'|y)$ and emissions $e(x|y)$

$$p(x_1 \ldots x_n, y_1 \ldots y_{n+1}) = q(\text{STOP}|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

- **Inference** (linear time in sentence length!)
  - Viterbi:

$$y* = \operatorname*{argmax}_{y_1 \ldots y_n} p(x_1 \ldots x_n, y_1 \ldots y_{n+1})$$
$$\text{where } y_{n+1} = \text{STOP}$$

  - Forward Backward:

$$p(x_1 \ldots x_n, y_i) = \sum_{y_1 \ldots y_{i-1}} \sum_{y_{i+1} \ldots y_n} p(x_1 \ldots x_n, y_1 \ldots y_n)$$

Learning: Maximum Likelihood

- Supervised Learning
  - Assume m fully labeled training examples:

$$\{(x^{(i)}, y^{(i)}) | i = 1 \cdots m\}$$

where $x^{(i)} = x_1 \cdots x_n$ and $y^{(i)} = y_1 \cdots y_n$

- What's the maximum likelihood estimate?

$$p(x_1 ... x_n, y_1 ... y_{n+1}) = q(\text{STOP}|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

$q_{ML}(y_i|y_{i-1})$ $e_{ML}(x_i|y_i)$

Learning: Maximum Likelihood

- MLE: counting the co-occurrence of the event

$$q_{ML}(y_i|y_{i-1}) = \frac{c(y_{i-1}, y_i)}{c(y_{i-1})} \qquad e_{ML}(x|y) = \frac{c(y,x)}{c(y)}$$

- Will these estimates be high quality?
  - Which is likely to be more sparse, *q* or *e*?
  - The emission function, because $c(y,x)$ is more likely to have sparse values.

- Can use all the same smoothing tricks we used for counting-based language models!

- Other approaches: Map low-frequency words to a small, finite set of units (e.g., prefixes, word classes), and run MLE on new sequences

Inference (Decoding)

- Problem: find the most likely (Viterbi) sequence under the model

$$y* = \underset{y_1 \ldots y_n}{\text{argmax}} \; p(x_1 \ldots x_n, y_1 \ldots y_{n+1})$$

- Given model parameters, we can score any sequence pair

NNP	VBZ	NN	NNS	CD	NN	.
Fed	raises	interest	rates	0.5	percent	.

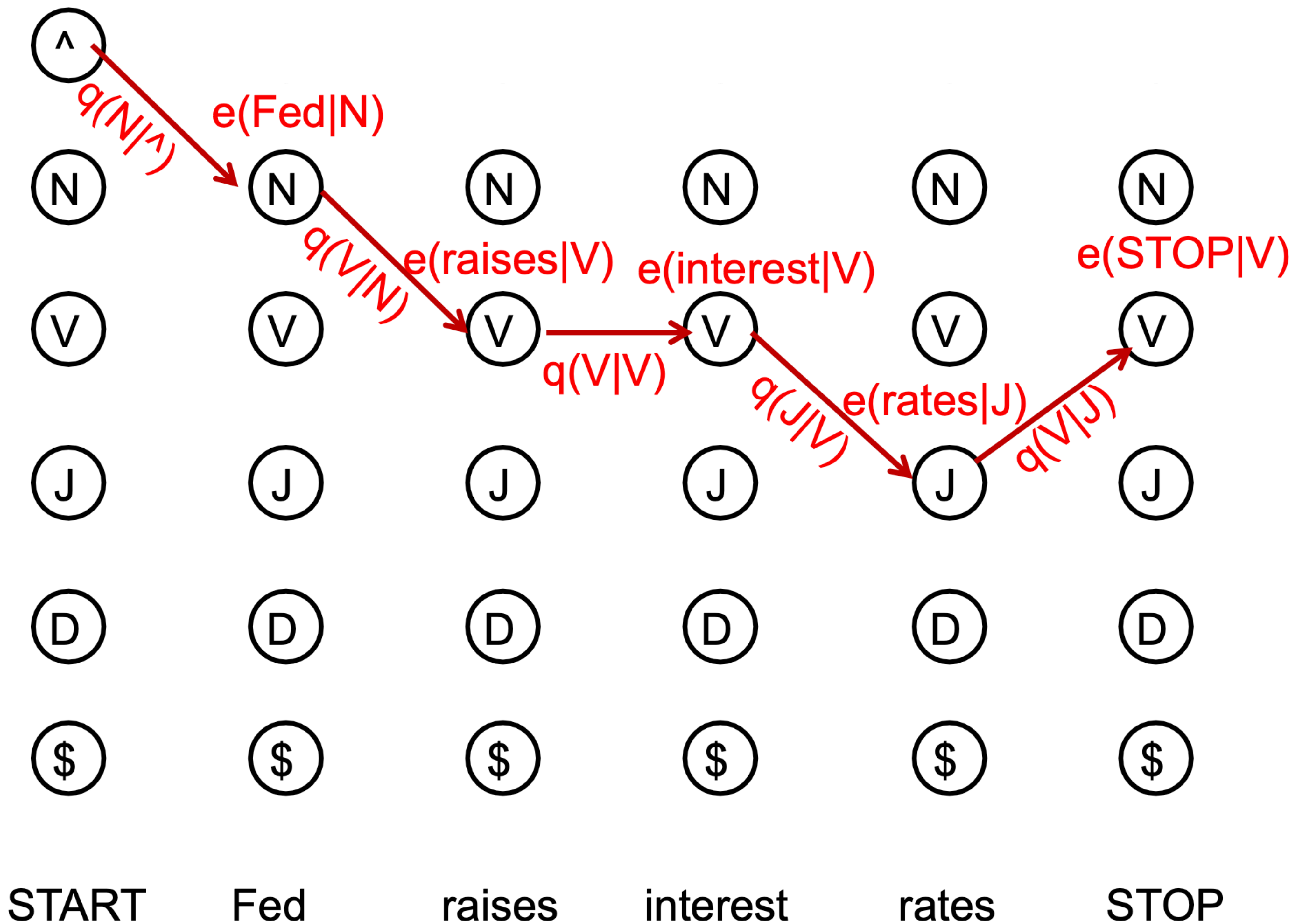
- In principle, we can list all possible tag sequences, score each one, and pick **the best one (a.k.a. the Viterbi state sequence)**

NNP VBZ NN NNS CD NN → logP = -23

NNP NNS NN NNS CD NN → logP = -29

NNP VBZ VB NNS CD NN → logP = -27

The State Lattice/Trellis: Viterbi



- Brute force approach: enumerate n^K possible tag sequences

Dynamic Programming!

- Focus on max, consider special case of n=2

- Define $\pi(i, y_i)$ to be the max score of a sequence of length i ending in tag y_i

$$\max_{y_1, y_2} q(STOP|y_2)q(y_2|y_1)e(x_2|y_2)q(y_1|START)e(x_1|y_1)$$

$$= \max_{y_2} q(STOP|y_2)e(x_2|y_2) \max_{y_1} q(y_1|START)q(y_2|y_1)e(x_1|y_1)$$

$$= \max_{y_2} q(STOP|y_2)e(x_2|y_2)\pi(2, y_2)$$

given that $\pi(2, y_2) = \max_{y_1} q(y_1|START)q(y_2|y_1)e(x_1|y_1)$

- What about the general case? (Consider n=3, etc…)

Dynamic Programming!

- General case

- Define $\pi(i, y_i)$ to be the max score of a sequence of length i ending in tag y_i

$$\pi(i, y_i) = \max_{y_1 \dots y_{i-1}} p(x_1 \dots x_i, y_1 \dots y_i)$$

$$= \max_{y_{i-1}} e(x_i | y_i) q(y_i | y_{i-1}) \max_{y_1 \dots y_{i-2}} p(x_1 \dots x_{i-1}, y_1 \dots y_{i-1})$$

$$= \max_{y_{i-1}} e(x_i | y_i) q(y_i | y_{i-1}) \quad \pi(i-1, y_{i-1})$$

- We now have an efficient algorithm. Start with i=*0* and work your way to the end of the sentence!

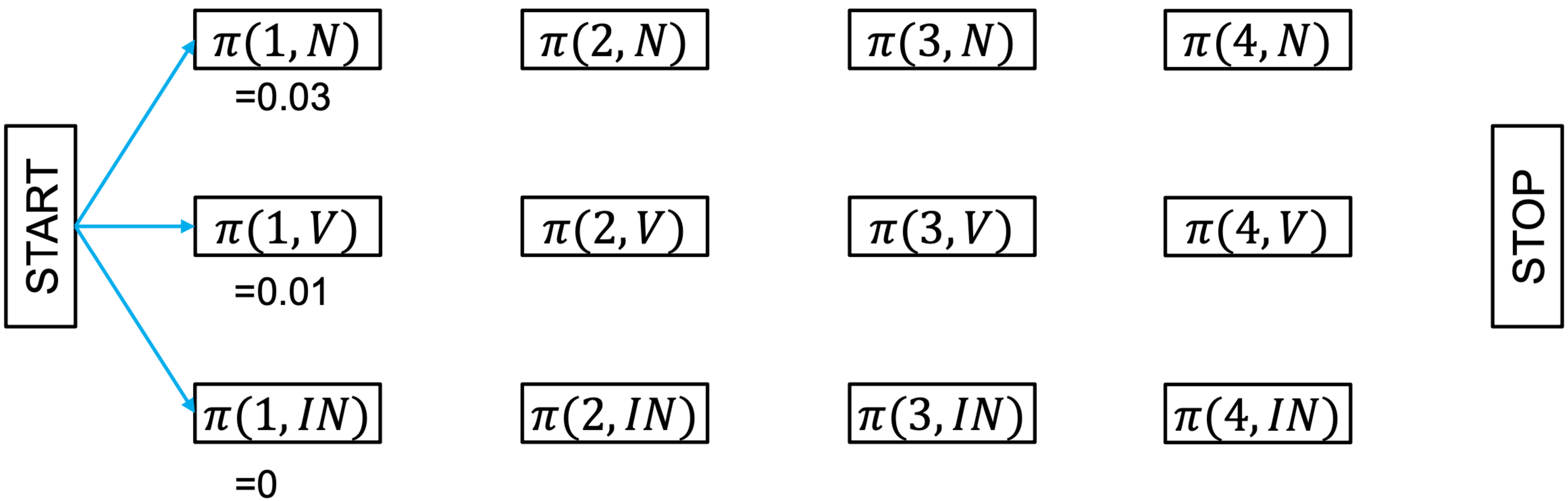
Viterbi (Example)

	Fruit	Flies	Like	Bananas	
	$\pi(1, N)$	$\pi(2, N)$	$\pi(3, N)$	$\pi(4, N)$	
START	$\pi(1, V)$	$\pi(2, V)$	$\pi(3, V)$	$\pi(4, V)$	STOP
	$\pi(1, IN)$	$\pi(2, IN)$	$\pi(3, IN)$	$\pi(4, IN)$	

$$\pi(i, y_i) = \max_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1})\pi(i-1, y_{i-1})$$

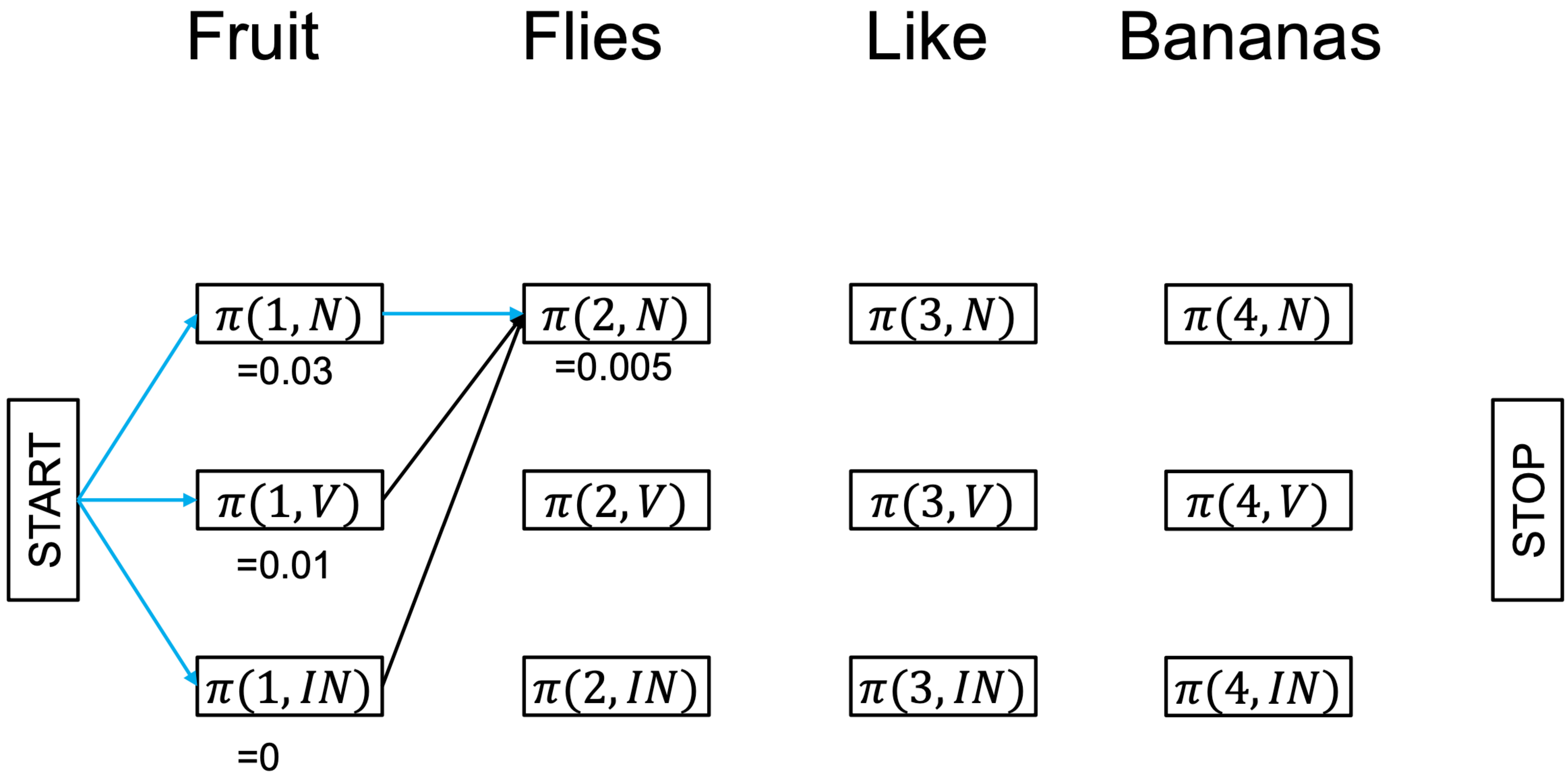
Viterbi (Example)

Fruit Flies Like Bananas



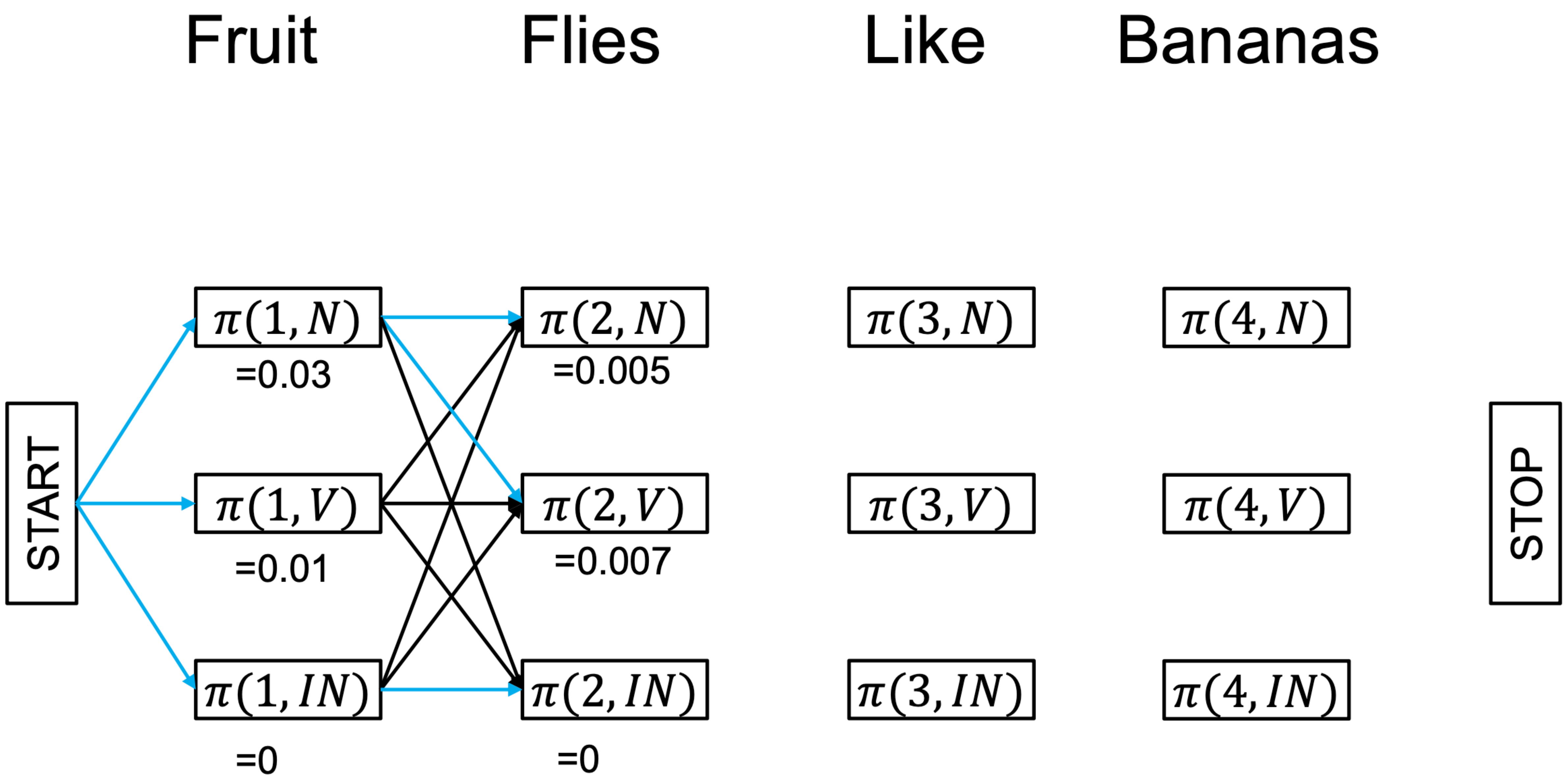
$$\pi(i, y_i) = \max_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1})\pi(i-1, y_{i-1})$$

Viterbi (Example)



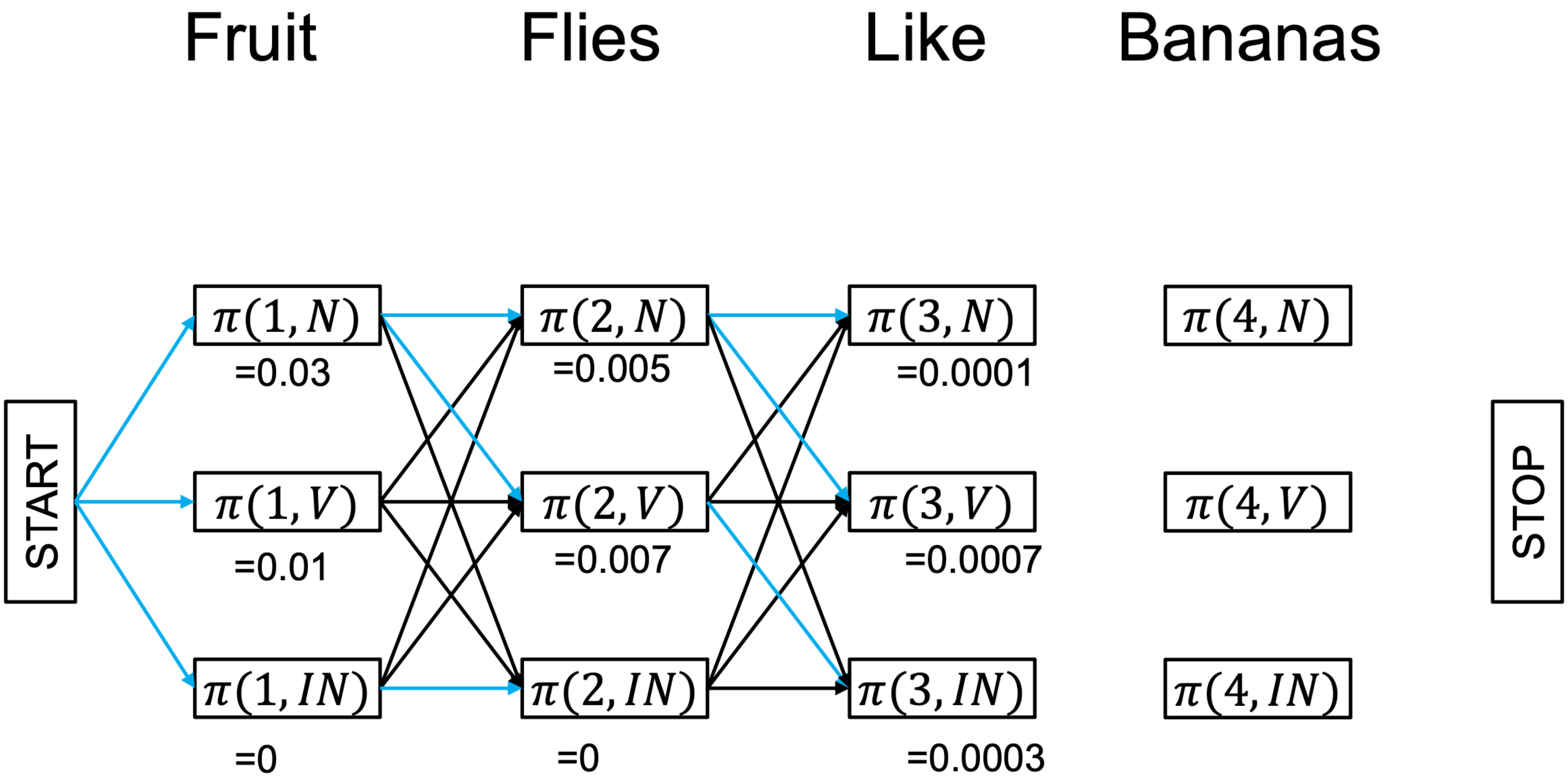
$$\pi(i, y_i) = \max_{y_{i-1}} e(x_i | y_i) q(y_i | y_{i-1}) \pi(i-1, y_{i-1})$$

Viterbi (Example)



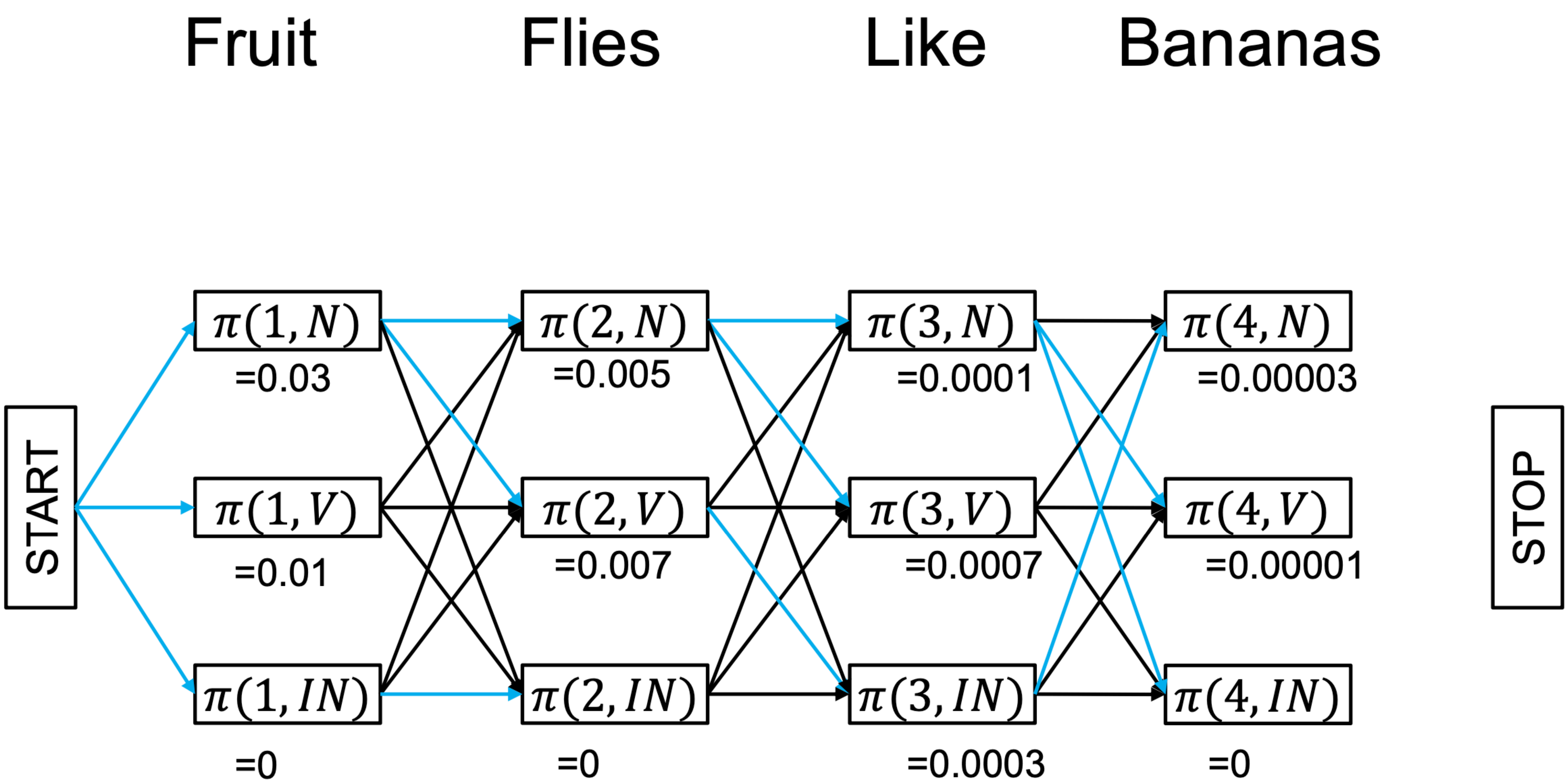
$$\pi(i, y_i) = \max_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1})\pi(i-1, y_{i-1})$$

Viterbi (Example)



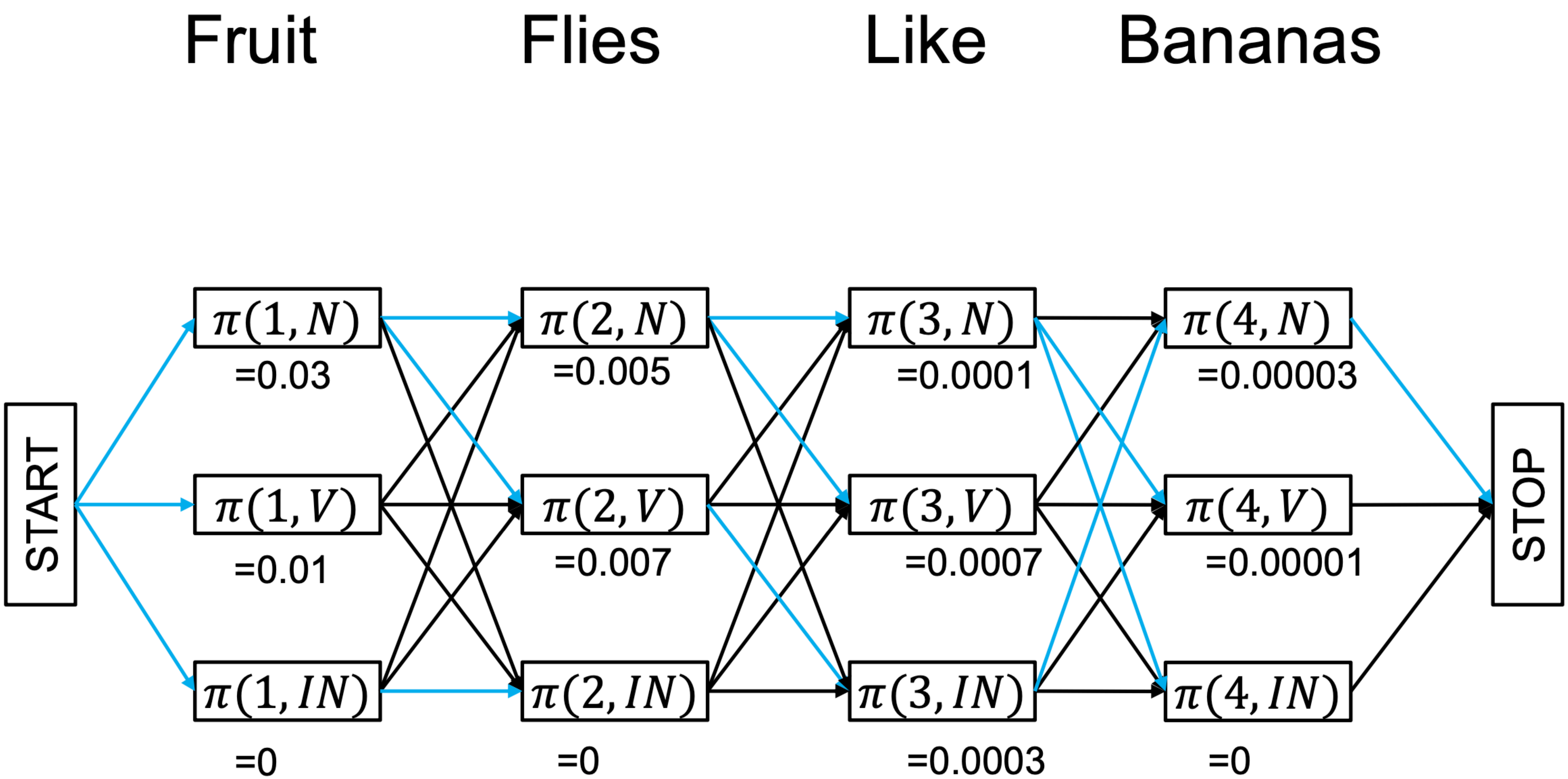
$$\pi(i, y_i) = \max_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1})\pi(i-1, y_{i-1})$$

Viterbi (Example)



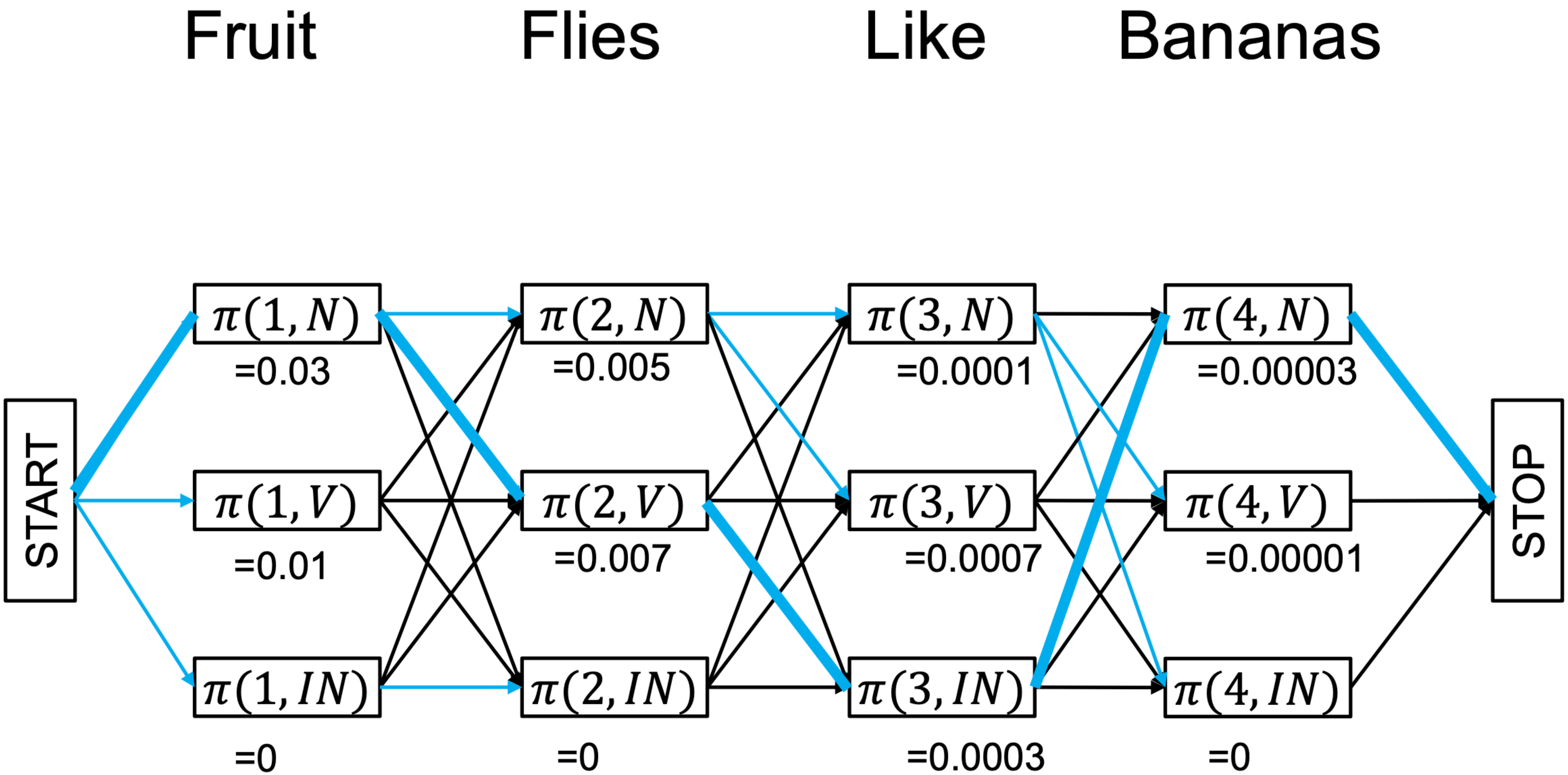
$$\pi(i, y_i) = \max_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1})\pi(i-1, y_{i-1})$$

Viterbi (Example)



$$\pi(i, y_i) = \max_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1})\pi(i-1, y_{i-1})$$

Viterbi (Example)



$$bp(i, y_i) = \arg\max_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1})\pi(i-1, y_{i-1})$$

Why is this not a greedy algorithm? Why does this find max P(.)?

Viterbi Algorithm

- Dynamic programming (for all i)

$$\pi(i, y_i) = \max_{y_1 \cdots y_{i-1}} p(x_1 \ldots x_i, y_1 \ldots y_i)$$

- Iterative computation

$$\pi(0, y_0) = \begin{cases} 1 \text{ if } y_0 == START \\ 0 \text{ otherwise} \end{cases}$$

For i = 1 … n:

$$\pi(i, y_i) = \max_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1})\pi(i-1, y_{i-1})$$

- Store back pointers:

$$bp(i, y_i) = \arg\max_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1})\pi(i-1, y_{i-1})$$

- What is the final solution? $bp(n+1, STOP)$

Viterbi Algorithm: Time complexity

- Linear in sentence length *n*

- Polynomial in the number of possible tags $\mathcal{K}$

$$\pi(i, y_i) = \underbrace{\max_{y_{i-1}}}_{\text{iterate over all possible tags}} e(x_i|y_i)q(y_i|y_{i-1})\pi(i-1, y_{i-1})$$

- Specifically:

$$O(n|\mathcal{K}|) \text{ entries in } \pi(i, y_i)$$

$$O(|\mathcal{K}|) \text{ time to compute each } \pi(i, y_i)$$

- Total runtime:

$$O(n|\mathcal{K}|^2)$$

Conditional Random Fields

(Sequential Version of Logistic Regression)

Recap: Logistic Regression (Log Linear Models)

- **Text classification:** $X = \{x_1 \cdots, x_n\}, y \in \{1 \cdots C\}$

$F(X, y = c)$ Scoring function

$$P(y = c|X) = \frac{\boxed{\exp(w_c^T f(X) + b_c)}}{\boxed{\sum_k \exp(w_k^T f(X) + b_k)}}, \qquad w_c, f(X) \in \mathbb{R}^d$$

$Z(X)$ Normalization constant or partition function

- **"Log-linear" assumption:**
  - The features of the input is "log-linear" to the output

$$\log P(y = c|X) = F(y = c, X) - \log Z(X)$$

  - Very flexible to include hand-crafted features (or learned features by neural networks)

Linear chain Conditional Random Fields
(“Log-Linear” 1st order Sequential Model)

- **Sequence labeling** $X = \{x_1 \cdots x_n\}, Y = \{y_1 \cdots y_n, \text{STOP}\}$:

$$P(Y|X) = \frac{1}{\boxed{Z(X)}} \exp \boxed{\left(\sum_{i=2}^{n+1} \lambda \cdot q(y_{i-1}, y_i, X) + \sum_{i=1}^{n} \mu \cdot g(y_i, X) \right)}$$

$$Z(X) = \sum_Y \exp(F(Y, X))$$

d1 features scoring transitions

d2 features scoring each state w/ input sequence

$$F(Y, X) = w \cdot f(Y, X) = \sum_{i=1}^{n} w \cdot f(y_i, y_{i+1}, X), \quad w, f(Y, X) \in \mathbb{R}^d$$

$$f(y_i, y_{i+1}, X) = [q(y_i, y_{i+1}, X);\ g(y_i, X)]$$

$$w = [\lambda;\ \mu], \lambda \in \mathbb{R}^{d_1}, \mu \in \mathbb{R}^{d_2}$$

ICML “test-of-time” paper: Lafferty et al. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data

CRF: Learning

- **Learning**: maximize the log-likelihood over the training data

$$
\begin{aligned}
\mathcal{L}(w) &= \sum_{(X,Y)\sim\mathcal{D}_{\mathrm{train}}} \log P(Y|X) \\
&= \sum_{(X,Y)\sim\mathcal{D}_{\mathrm{train}}} w^\top f(Y,X) - \log \boxed{Z(X)}
\end{aligned}
$$

$$w^* = \arg\max_w \mathcal{L}(w)$$

Sum over all possible outputs *Y* for an input *X* — Brute force solution: score n^C outputs
Can we do faster?

- **Update:** stochastic gradient descent to move in a direction that decreases the loss

$$w \leftarrow w - \alpha \frac{\partial \mathcal{L}(w)}{\partial w}$$

Dynamic Programing

- **Learning**: maximize the log-likelihood over the training data

$$\frac{\partial \log Z(X)}{\partial w_j} = \mathbb{E}_Y \left[\sum_{i=1}^{n} f_j(y'_i, y'_{i+1}, X) \right]$$

$$= \sum_{i=1}^{n} \mathbb{E}_{y'_i, y'_{i+1}} \left[P(y'_i, y'_{i+1} | X) f_j(y'_i, y'_{i+1}, X) \right]$$

$$= \sum_{i=1}^{n} \sum_{y'_i, y'_{i+1}} P(y'_i, y'_{i+1} | X) f_j(y'_i, y'_{i+1}, X)$$

$P(y'_i, y'_{i+1} | X)$ can be computed by dynamic programing (forward-backward algorithm) — sum production algorithm, basically replace the max operation in Viterbi algorithm by sum operation

CRF Decoding: Viterbi

- Same as HMM decoding

- Viterbi (max-production algorithm): define the recursive function to compute the max value of the past partial sequence

$$
\begin{aligned}
Y^* &= \arg\max_Y \log P(Y|X) \\
&= \arg\max_Y w \cdot f(Y, X) - \underline{\log Z(X)} \\
&= \arg\max_Y \sum_{i=1}^{n} w \cdot f(y_i, y_{i+1}, X)
\end{aligned}
$$

Decoding output doesn't depend on the second term

Feature functions

- Feature functions based on **possible combination of words and tags**, or other information such as POS tag (if given), whether the word is capitalized or not

$$q_1(y_{i-1}, y_i, X) = \begin{cases} 1 & \text{if } y_{i-1} = \text{OTHER and } y_i = \text{PERSON} \\ 0 & \text{otherwise} \end{cases}$$

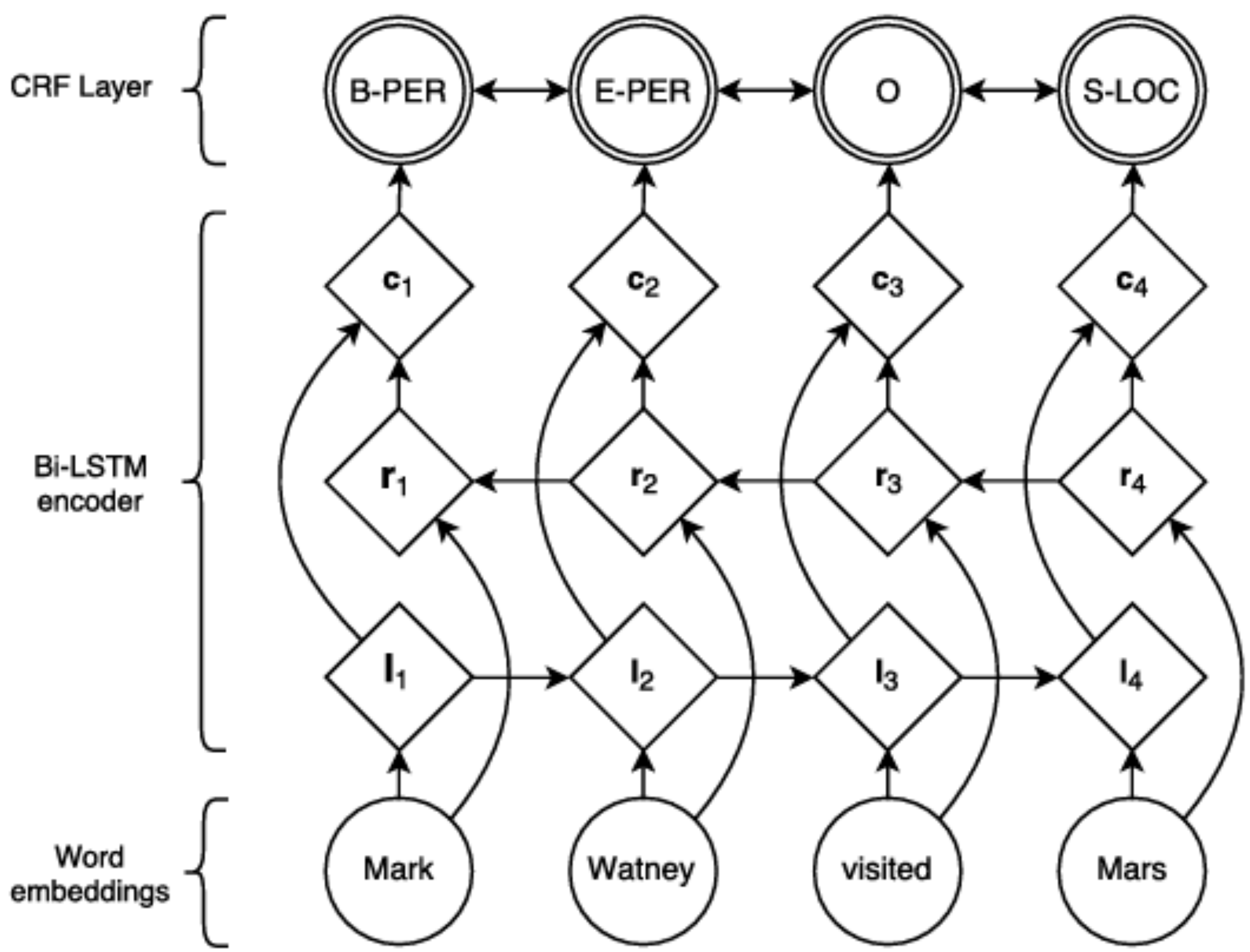
$$g_2(y_i, X) = \begin{cases} 1 & \text{if } y_i = \text{PERSON and } x_i = \text{John} \\ 0 & \text{otherwise} \end{cases}$$

Feature values are not limited to just binary values, can be real-values too.
Number of features can be tens of thousands or more.

Neural Conditional Random Fields

Neural CRF

- Rather than hand-crafted features, let's use NN to learn features.



$$p(\mathbf{y}|\mathbf{X}) = \frac{e^{s(\mathbf{X},\mathbf{y})}}{\sum_{\widetilde{\mathbf{y}} \in \mathbf{Y_X}} e^{s(\mathbf{X},\widetilde{\mathbf{y}})}}$$

$$s(\mathbf{X},\mathbf{y}) = \sum_{i=0}^{n} A_{y_i,y_{i+1}} + \sum_{i=1}^{n} P_{i,y_i}$$

Lampple et. al 2016 Neural Architectures for Named Entity Recognition

Learned Feature

- P_{i,y_i} : the output of the bi-LSTM model followed by a linear projection layer. $P \in \mathbb{R}^{n \times C}$
- $A \in \mathbb{R}^{C+2 \times C+2}$: is the transition matrix from one state (tag) to the other state, including the start/end states (so *C+2*).

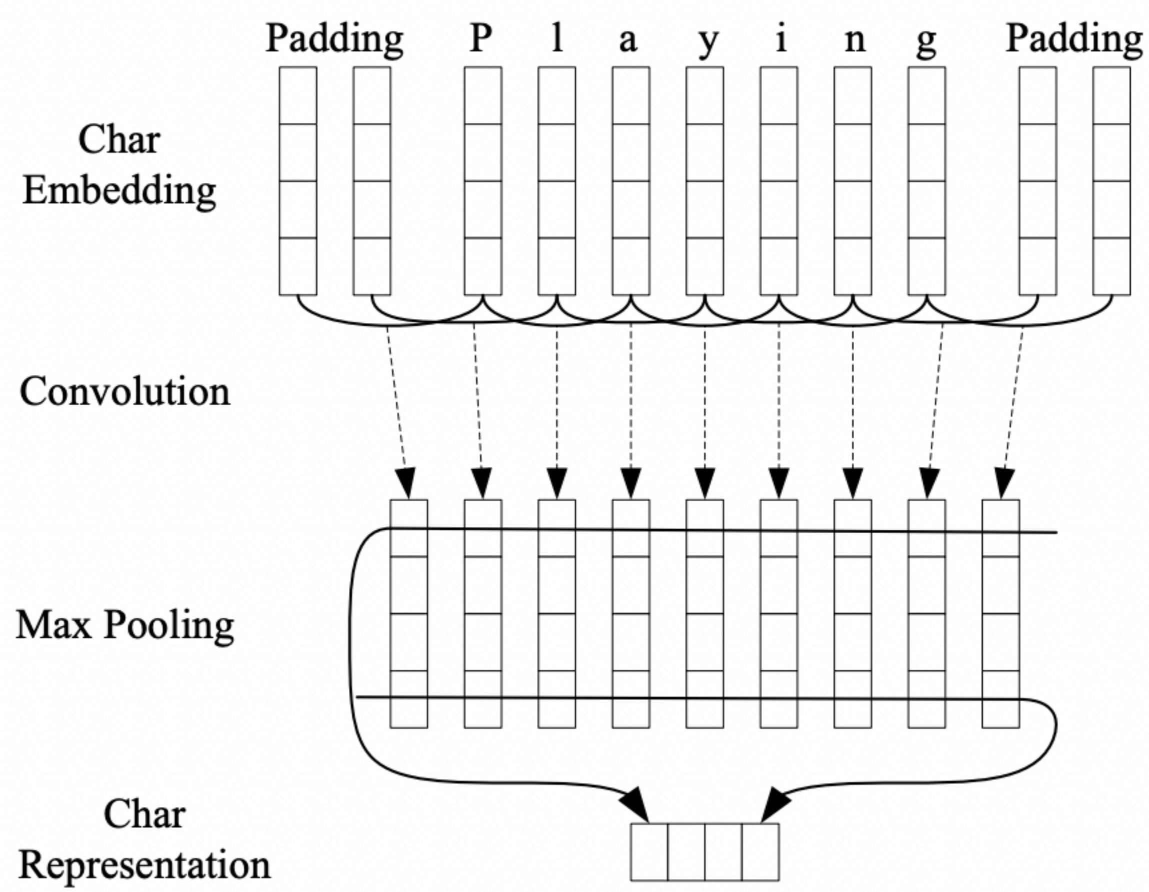
$$s(\mathbf{X}, \mathbf{y}) = \sum_{i=0}^{n} A_{y_i, y_{i+1}} + \sum_{i=1}^{n} P_{i, y_i}$$

Scoring the transition

Scoring the association
Of tag yi w/ the input X

BiLSTM-CNN CRF

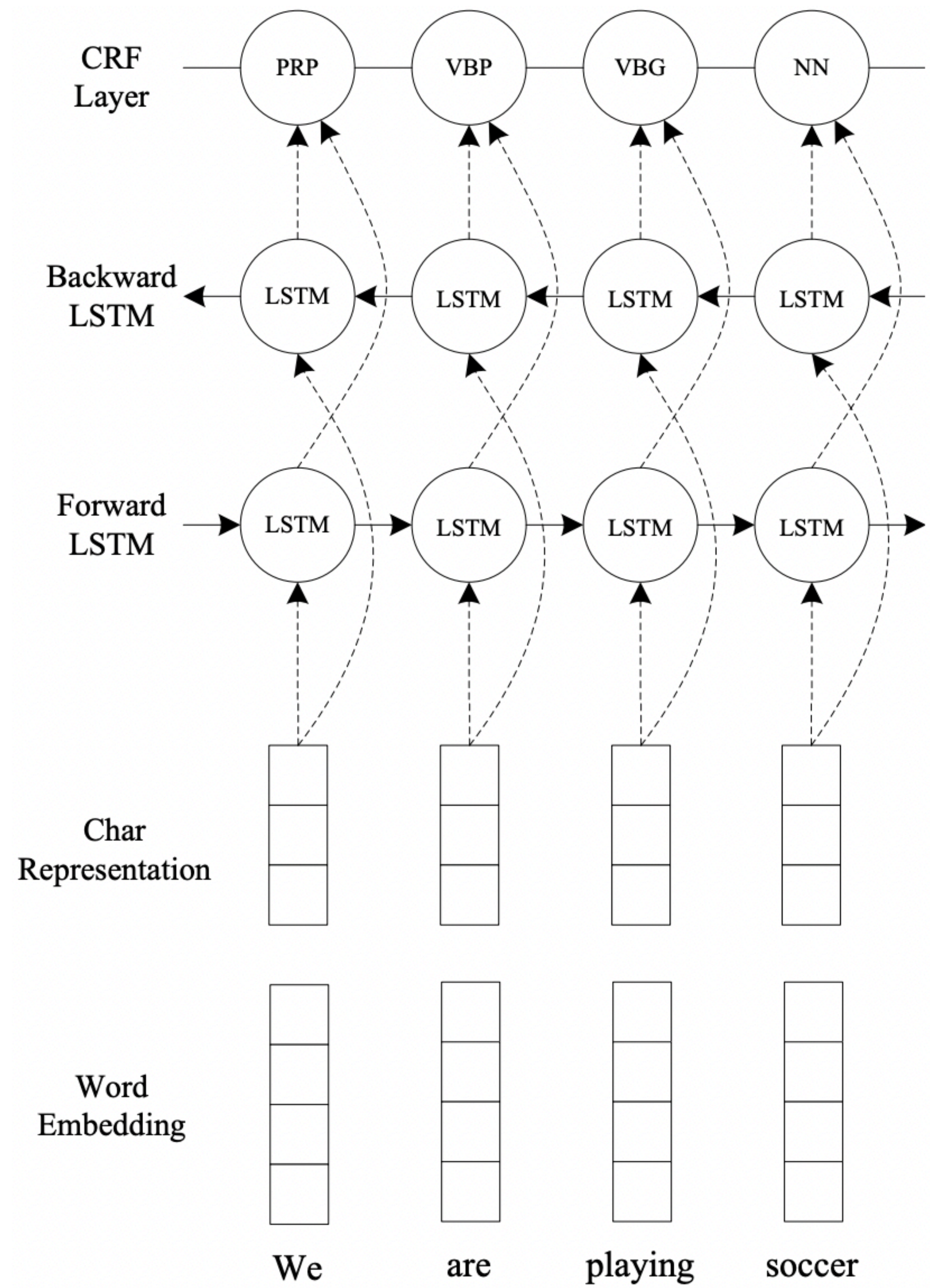
- Use CNN to encode character embeddings



Ma et al. 2016. End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF

BiLSTM-CNN CRF

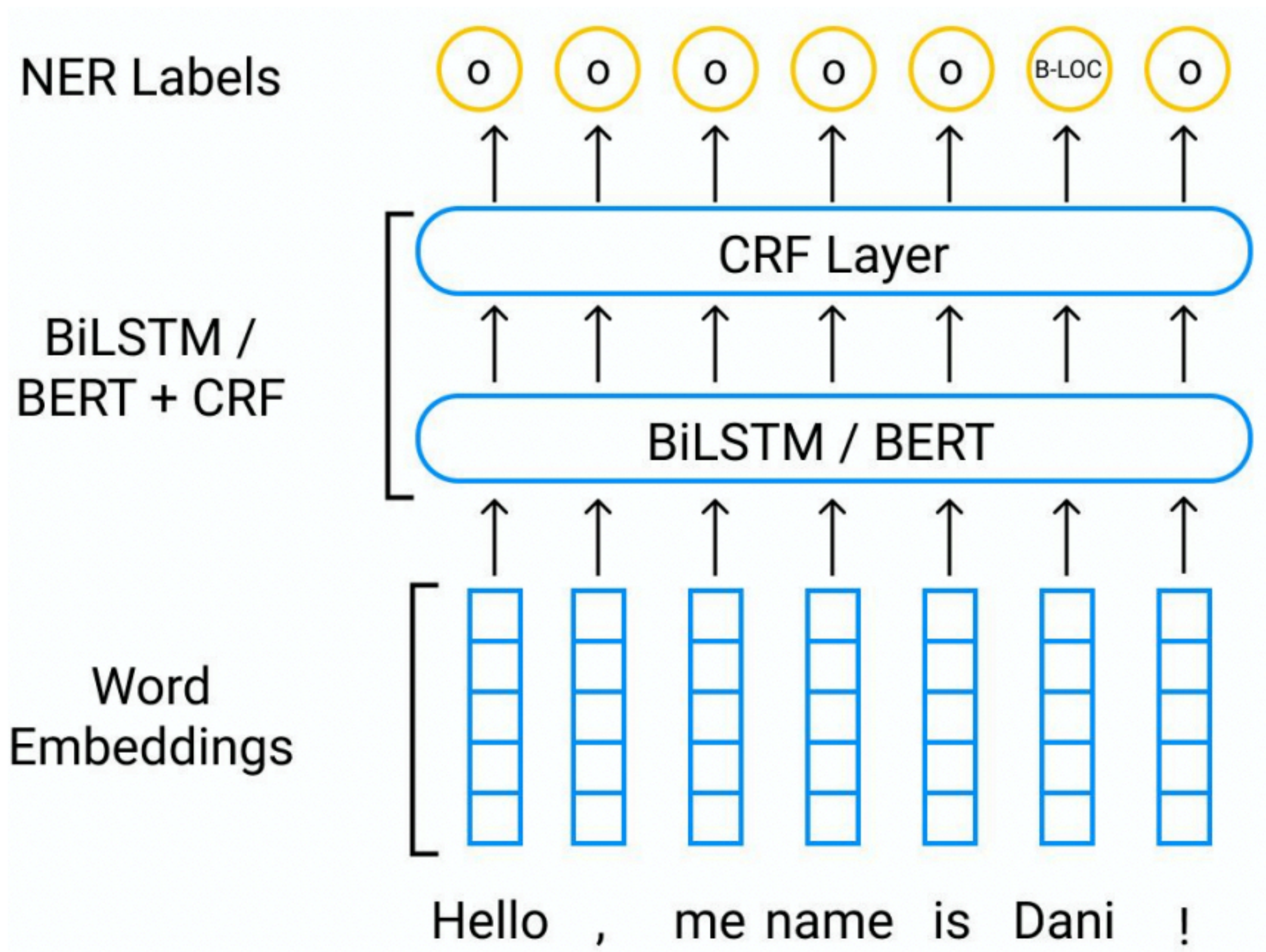
- Use CNN to encode character embeddings
- Combine char and word embeddings together
- Further encode by BiLSTM model to learn the sequence representations
- Add a CRF layer



Ma et al. 2016. End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF

BERT-CRF

- Replace BiLSTM with a BERT encoder



Comparison: Naive Bayes -> HMM
Logistic Regression -> CRF

Recap: Naive Bayes & HMMs

- Naive Bayes (for text classification):

$$P(X, y) = P(X|y)P(y) = \left(\prod_{x_i} P(x_i|y)\right) P(y)$$

- Hidden Markov Models (for sequence labeling):

$$P(X, Y) = q(\text{STOP}|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)$$

$$= \left(q(\text{STOP}|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}\right) \left(\prod_{i=1}^{n} e(x_i|y_i)\right)$$

$$= P(Y) \left(\prod_{i=1}^{n} P(x_i|y_i)\right)$$

HMMs ≈ sequence version of Naive Bayes!
Both are generative models.

Logistic Regression & CRF

- Logistic Regression (for text classification):

$$P(Y=c|X) \propto w_c \cdot F(X, Y=c)$$

- Conditional Random Field (for sequence labeling):

$$P(Y|X) = \prod_{i=1}^{T} P(Y_i|X_i, Y_{i-1}), \quad Y_0 = \text{[START]}$$

$$\begin{aligned} P(Y_i=c|X_i, Y_{i-1}) &\propto w_c \cdot f(Y_i=c, Y_{i-1}, X_i) \\ &= \lambda \cdot q(Y_i=c, Y_{i-1}, X_i) + \mu \cdot g(Y_i=c, X_i) \end{aligned}$$

CRF ≈ sequence version of Logistic Regression!
Both are discriminative models.

Generative v.s. Discriminative

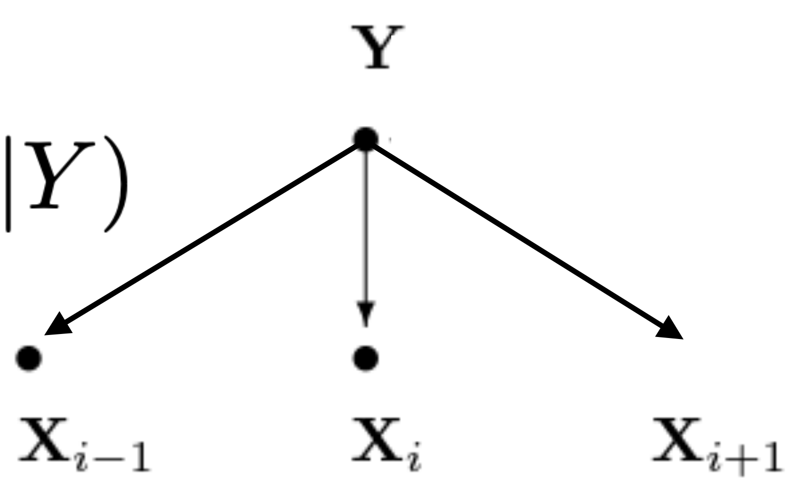
- **Generative Models**:
  - Joint probability: $P(X, Y)$
  - Make prediction by $\arg\max_Y P(X, Y)$
  - Can generate new samples (X, Y)
  - Examples: **HMMs, Naive Bayes**
- **Discriminative Models**:
  - Conditional probability: $P(Y|X)$
  - Can directly predict $\arg\max_Y P(Y|X)$
  - Examples: **Conditional Random Fields, Logistic Regression**
- Both trained via Maximum Likelihood Estimation

Compare Naive Bayes and Logistic Regression

- Directed graphical model vs undirected graphical model

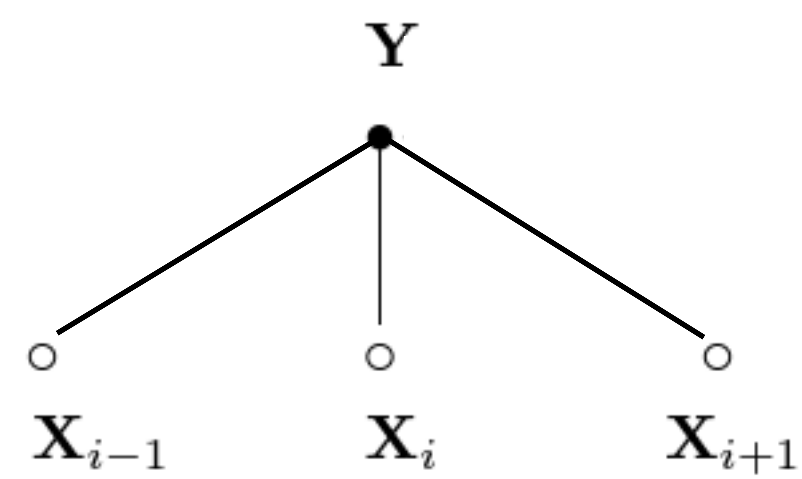
$$Y \sim P(Y)$$

$$P(X|Y)$$



Naive Bayes
(Generative)

$$P(Y = c|X) \propto w_c \cdot F(X, Y = c)$$



Logistic Regression
(Discriminative)

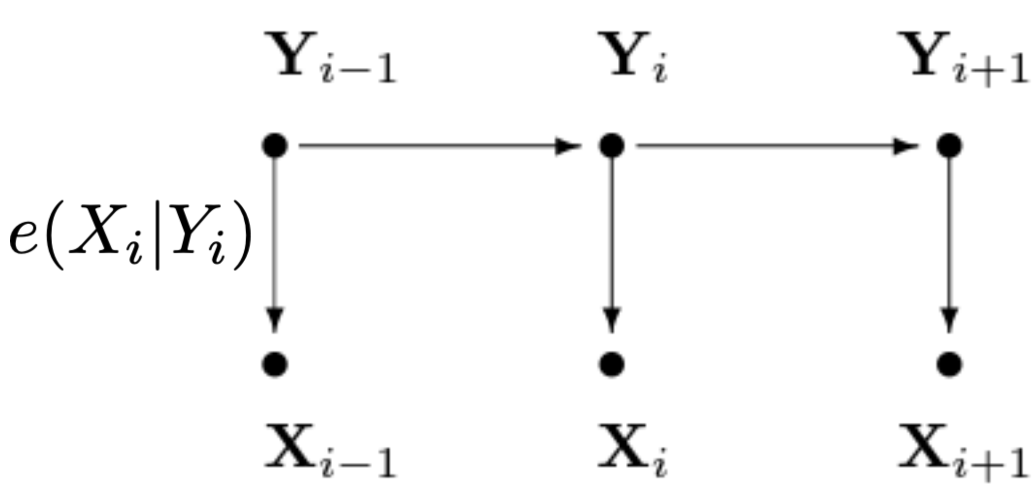
An open circle indicates that the variable is not generated by the model.

Compare HMM and linear chain CRF

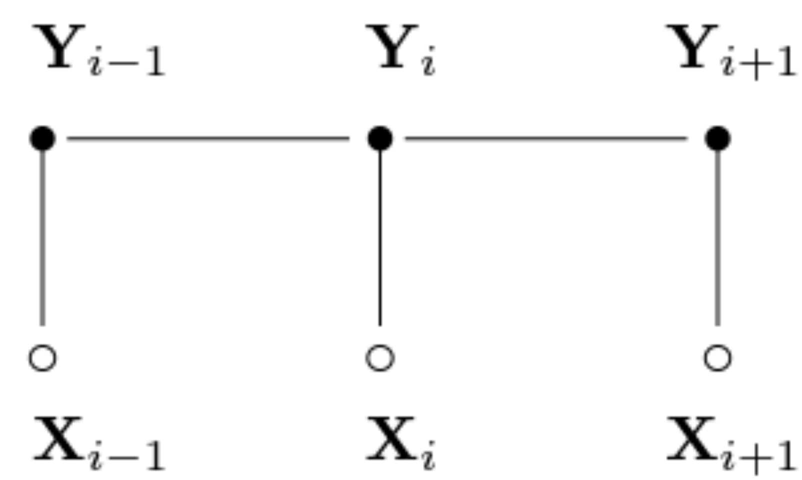
- Directed graphical model vs undirected graphical model

$$P(Y_i = c|X_i, Y_{i-1}) \propto w_c \cdot f(Y_i = c, Y_{i-1}, X_i)$$
$$= \lambda \cdot q(Y_i = c, Y_{i-1}, X_i) + \mu \cdot g(Y_i = c, X_i)$$

$q(Y_i|Y_{i-1})$



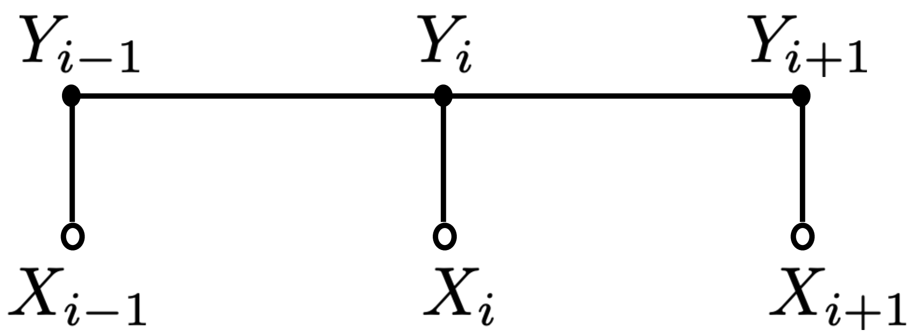
HMM
(Generative)



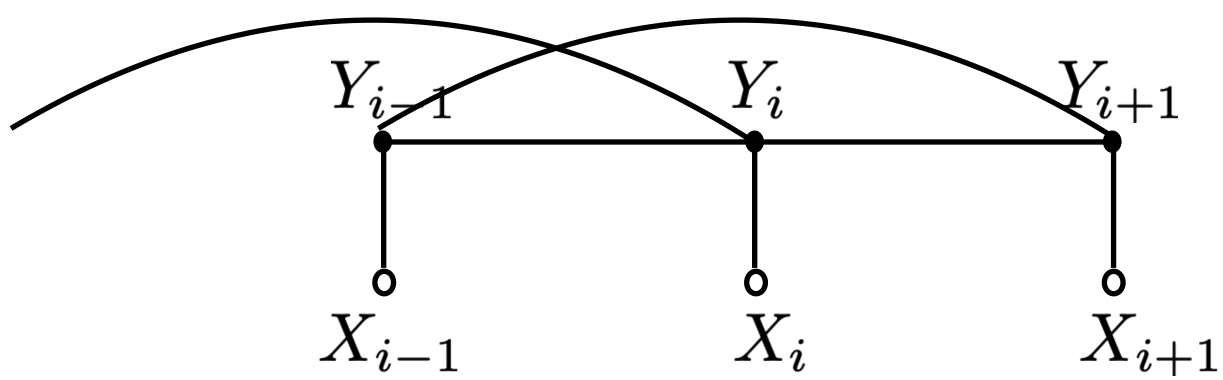
Chain-structure CRF
(Discriminative)

An open circle indicates that the variable is not generated by the model.

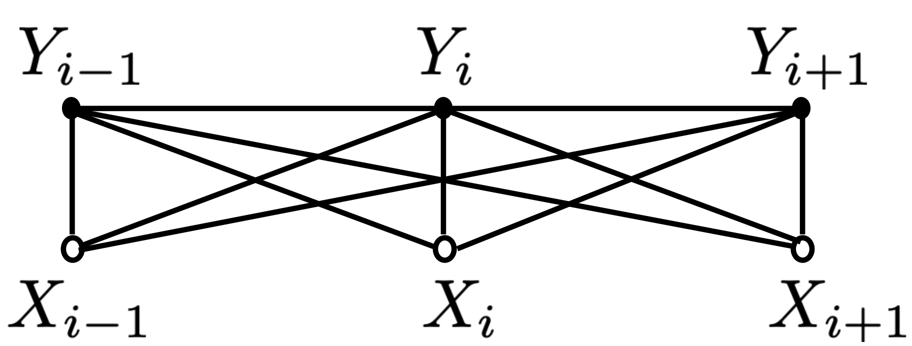
Variants of CRF Layers



- 1th order linear chain



- 2nd order linear chain



- Local vs. Global context

Questions?