Structured Prediction

Sequence Labeling

Recipe for multiclass classification

- Collect a training set (hopefully with correct labels)

- Define feature representations for inputs $(\mathbf{x} \in \Re^n)$
  - And, **y** in {book, penguin, dog}
- Linear functions to score labels

$$\arg\max_{\mathbf{y} \in \{\text{book, dog, penguin}\}} \mathbf{w}_{\mathbf{y}}^T \mathbf{x}$$

Recipe for multiclass classification

- Train weight vectors $\mathbf{w}_y$ so that it scores correctly

  eg, for an input of type "book", we want

  $\mathbf{w}_{book}^T\mathbf{x} > \mathbf{w}_{dog}^T\mathbf{x}$ and $\mathbf{w}_{book}^T\mathbf{x} > \mathbf{w}_{penguin}^T\mathbf{x}$, and so on…

- Prediction: $\arg\max_{y \in \{book,\ dog,\ penguin\}} \mathbf{w}_y^T\mathbf{x}$

  – Easy to predict

  – Iterate over the output list, find the highest scoring one

What if the space of outputs is much larger? Say trees, or in general, graphs. Let's look at examples.

Example 1: Semantic Role Labeling

- Based on the dataset PropBank [Palmer et. al. 05]
  - Large human-annotated corpus of verb semantic relations
- The task: To predict arguments of verbs

Given a sentence, identifies who does what to whom, where and when.

The bus was ***heading*** for Nairobi in Kenya

Example 1: Semantic Role Labeling

- Based on the dataset PropBank [Palmer et. al. 05]
  - Large human-annotated corpus of verb semantic relations
- The task: To predict arguments of verbs

Given a sentence, identifies who does what to whom, where and when.

The bus was ***heading*** for Nairobi in Kenya

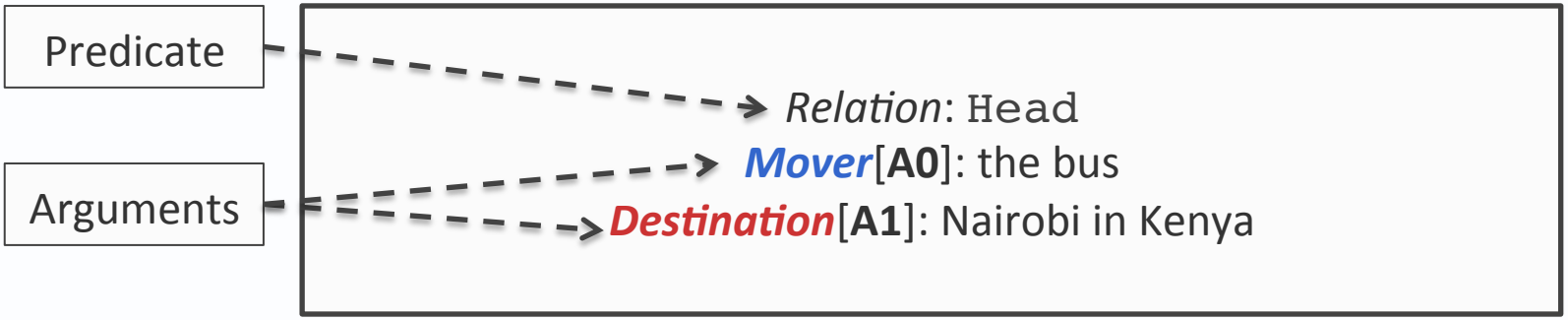
Relation: `Head`
Mover[**A0**]: the bus
Destination[**A1**]: Nairobi in Kenya

Example 1: Semantic Role Labeling

- Based on the dataset PropBank [Palmer et. al. 05]
  - Large human-annotated corpus of verb semantic relations
- The task: To predict arguments of verbs

Given a sentence, identifies who does what to whom, where and when.

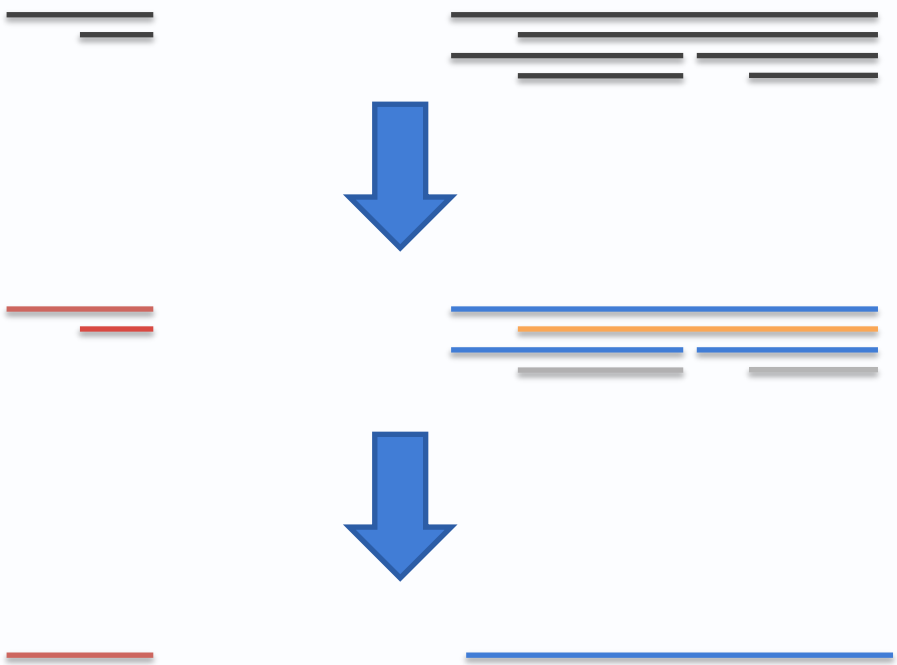
The bus was ***heading*** for Nairobi in Kenya



Predicting verb arguments

The bus was **heading** for Nairobi in Kenya.

1. Identify candidate arguments for verb using parse tree
   - Filtered using a binary classifier
2. Classify argument candidates
   - Multi-class classifier (one of multiple labels per candidate)
3. Inference
   - Using probability estimates from argument classifier
   - Must respect structural and linguistic constraints
     - Eg: No overlapping arguments



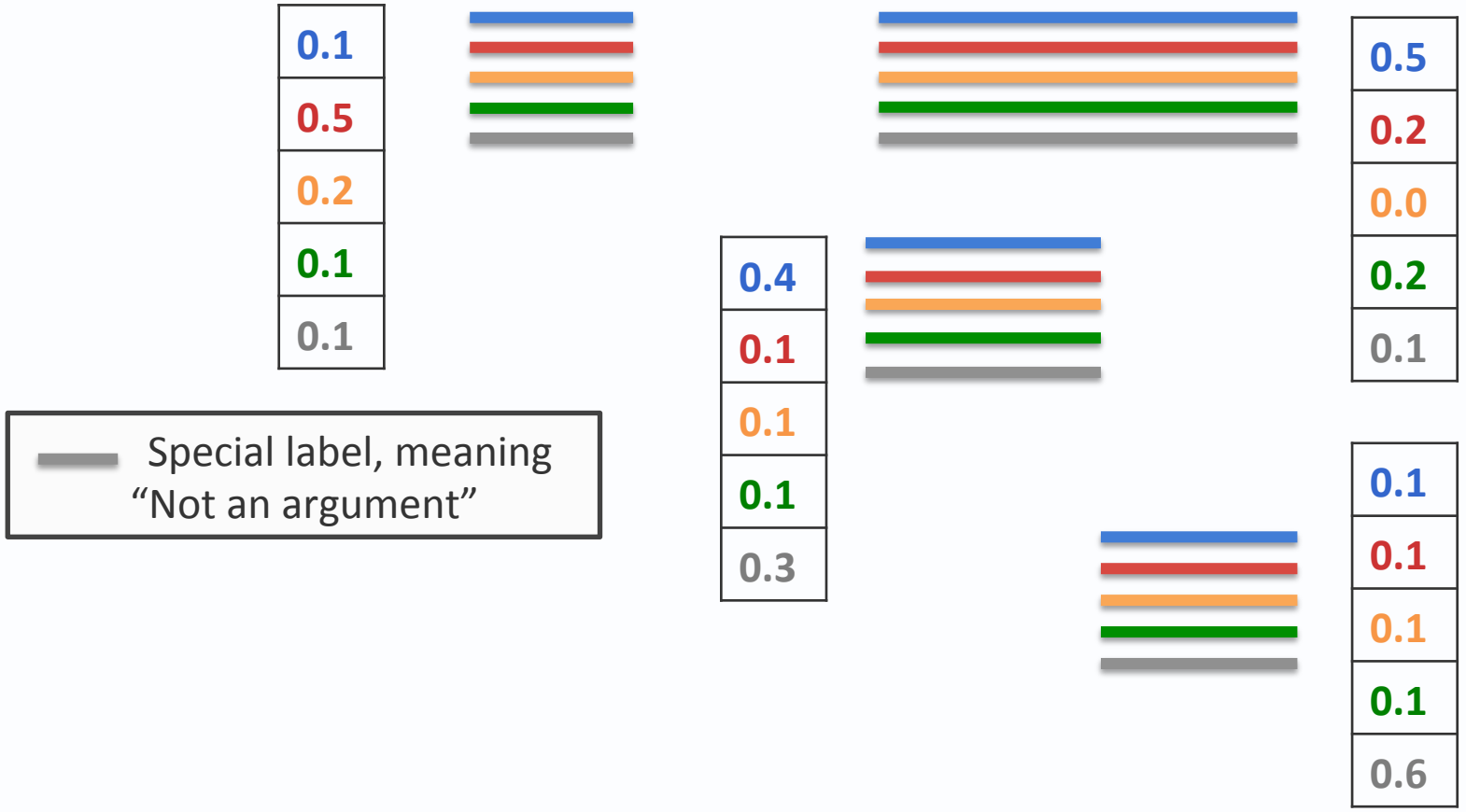
Inference: verb arguments

The bus was **heading** for Nairobi in Kenya.

Special label, meaning "Not an argument"

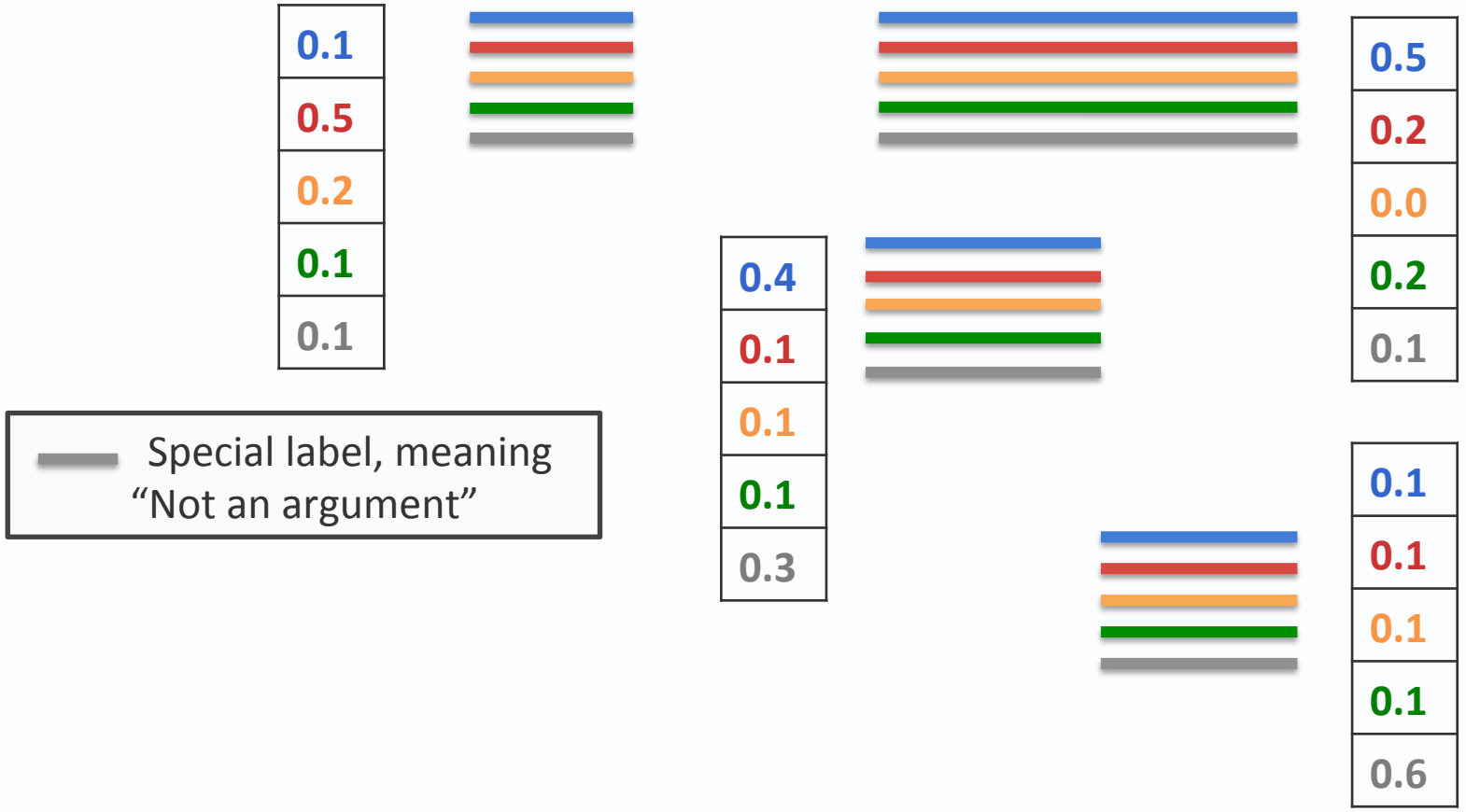
Inference: verb arguments

The bus was **heading** for Nairobi in Kenya.



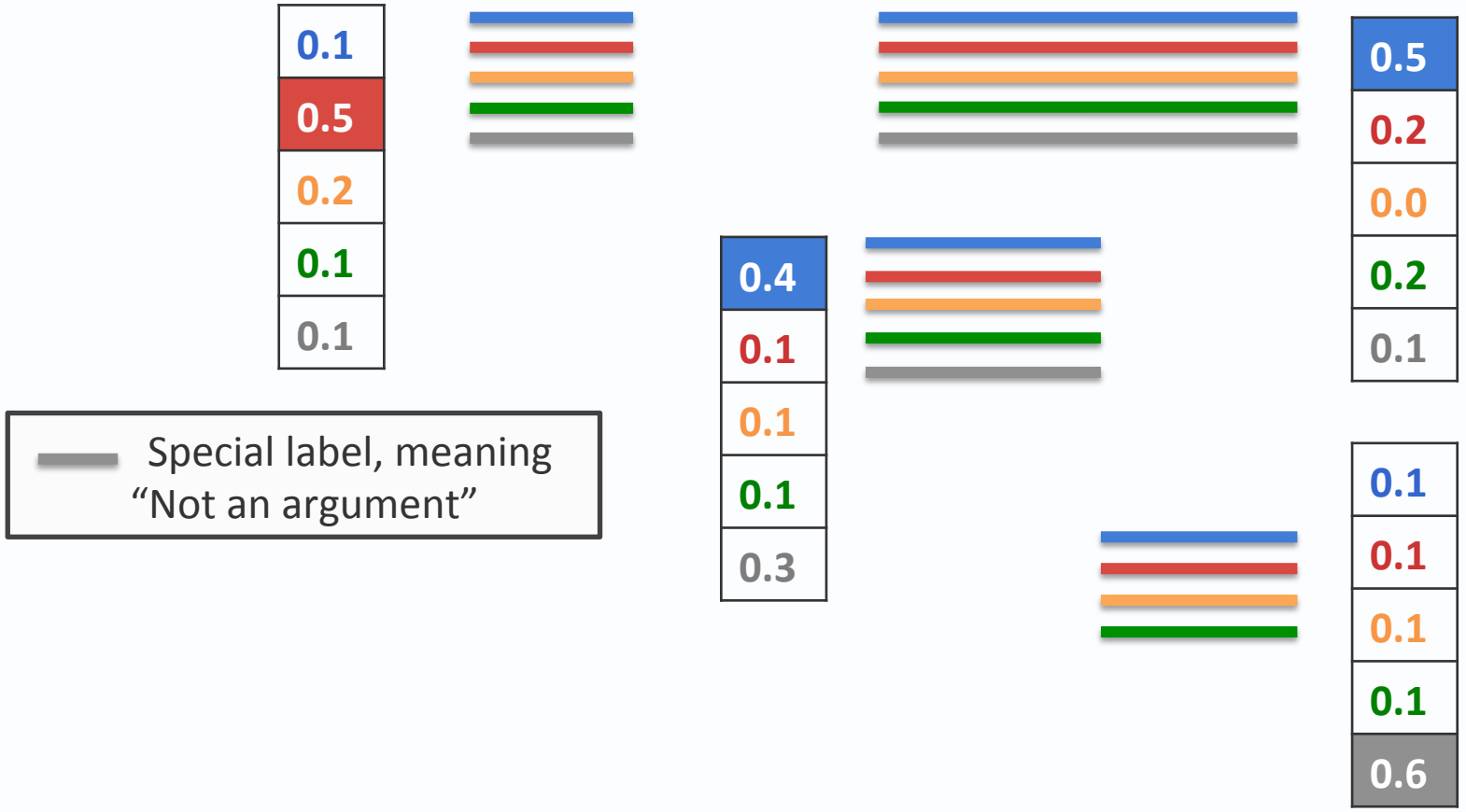
Inference: verb arguments

The bus was **heading** for Nairobi in Kenya.



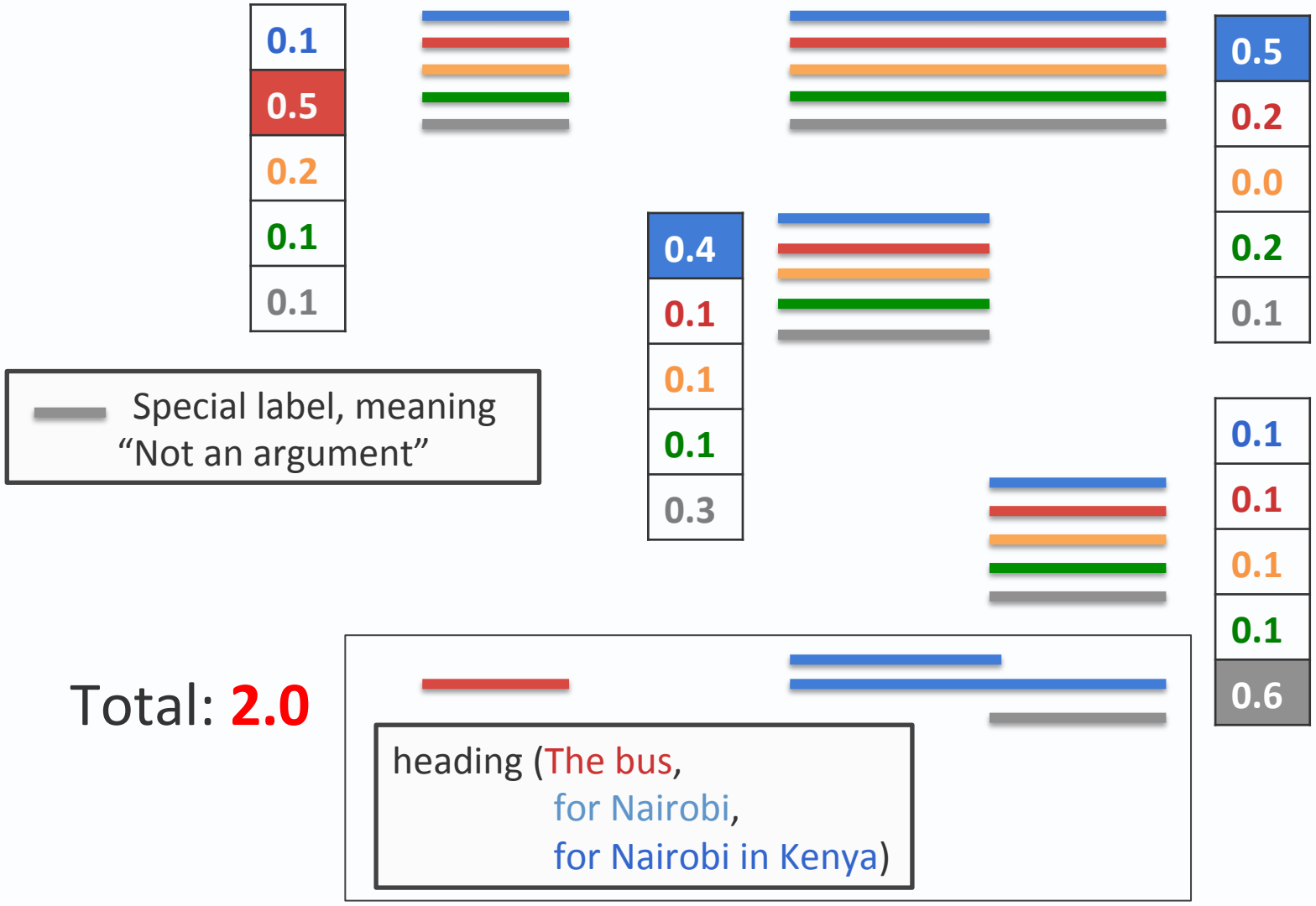
Inference: verb arguments

The bus was **heading** for Nairobi in Kenya.

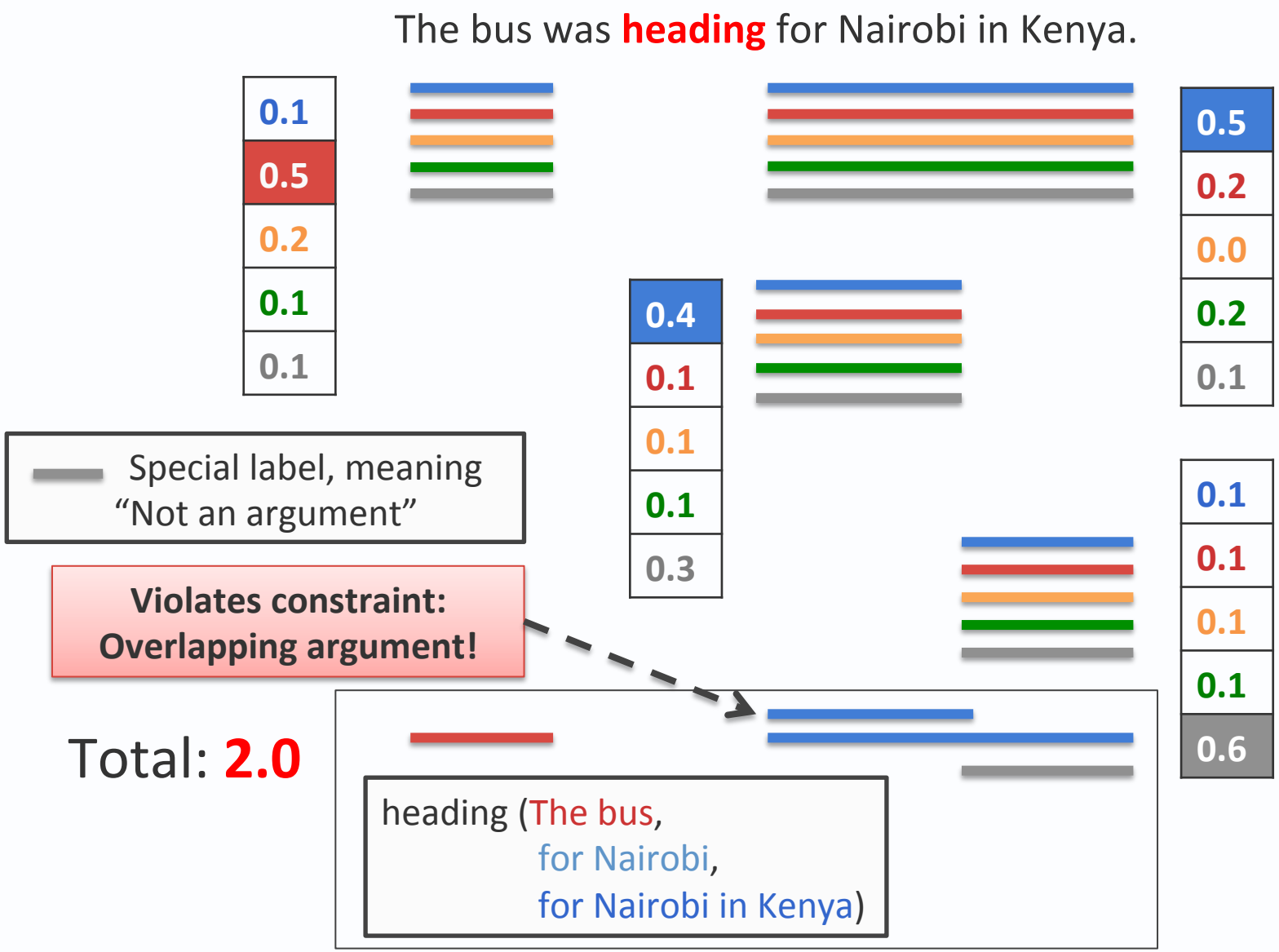


Inference: verb arguments

The bus was **heading** for Nairobi in Kenya.

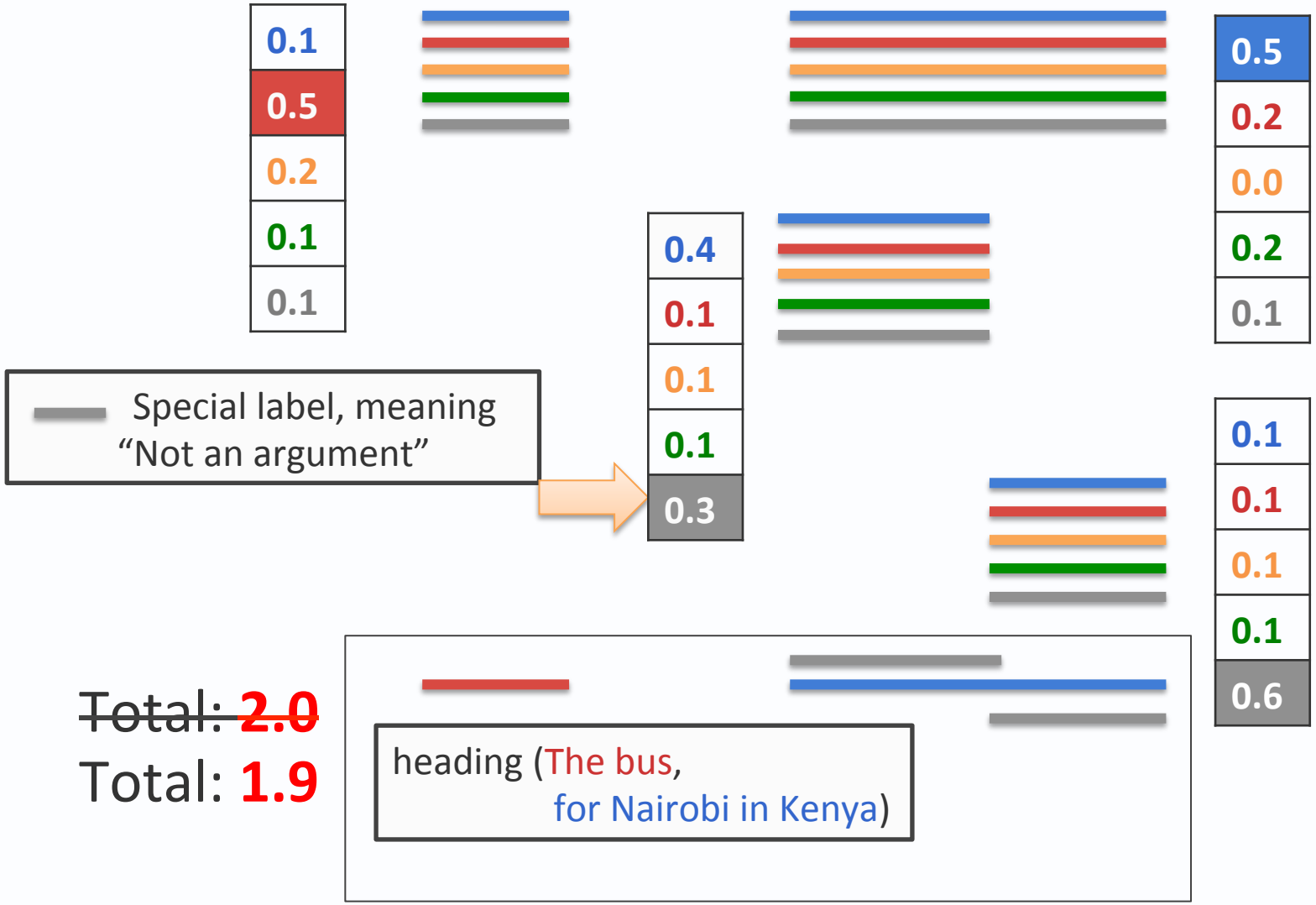


Inference: verb arguments



Inference: verb arguments

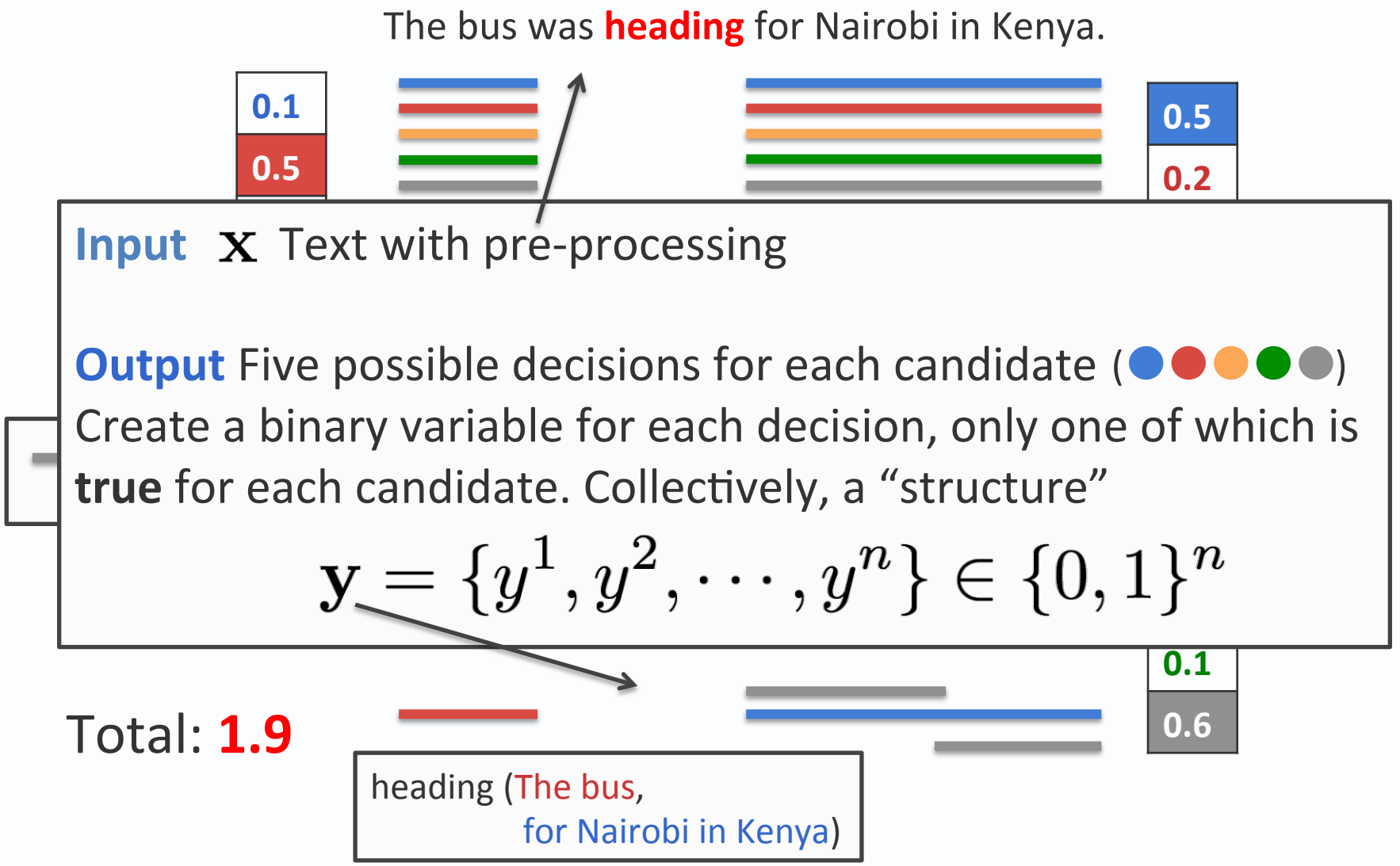
The bus was **heading** for Nairobi in Kenya.



~~Total: 2.0~~

Total: **1.9**

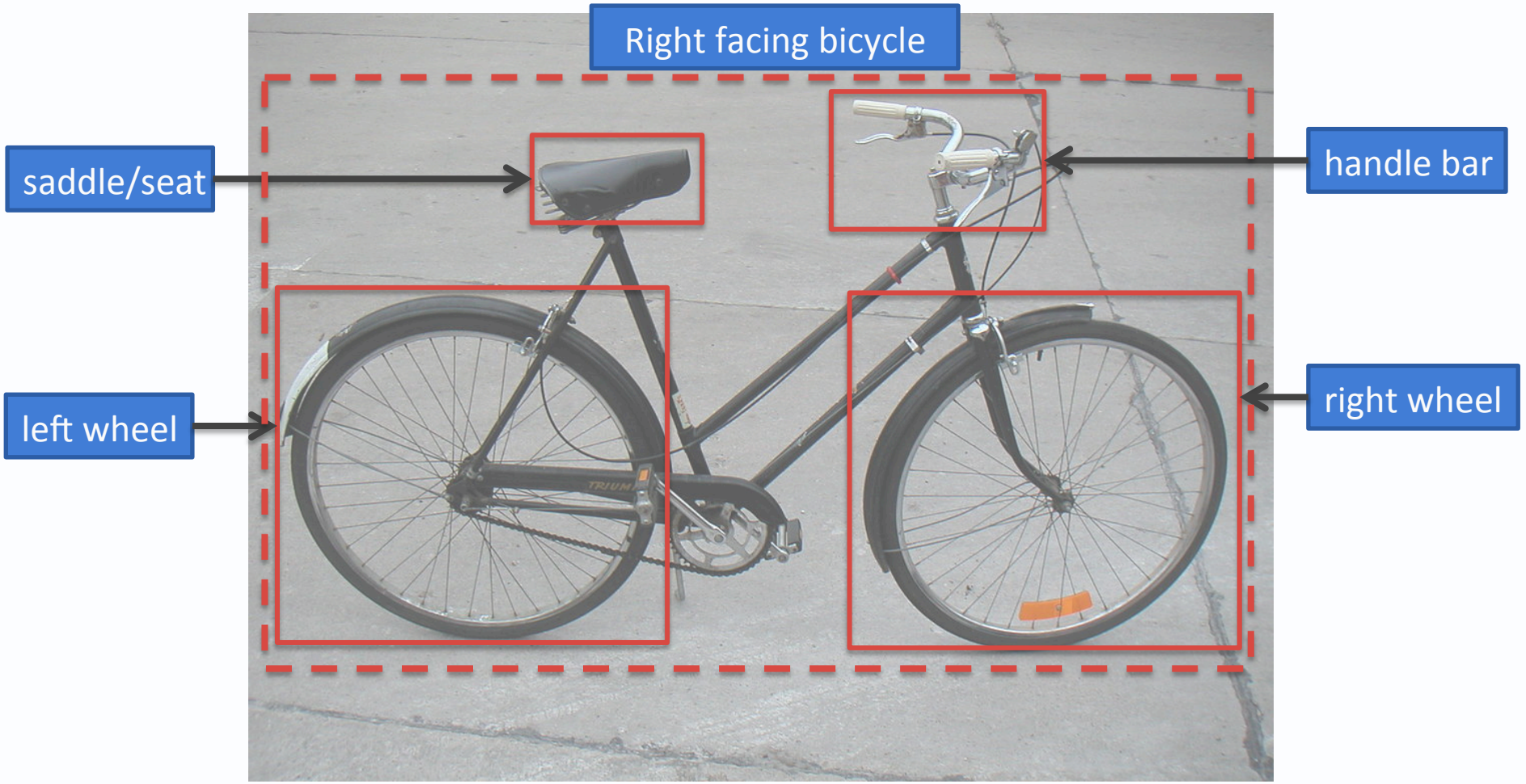
Inference: verb arguments



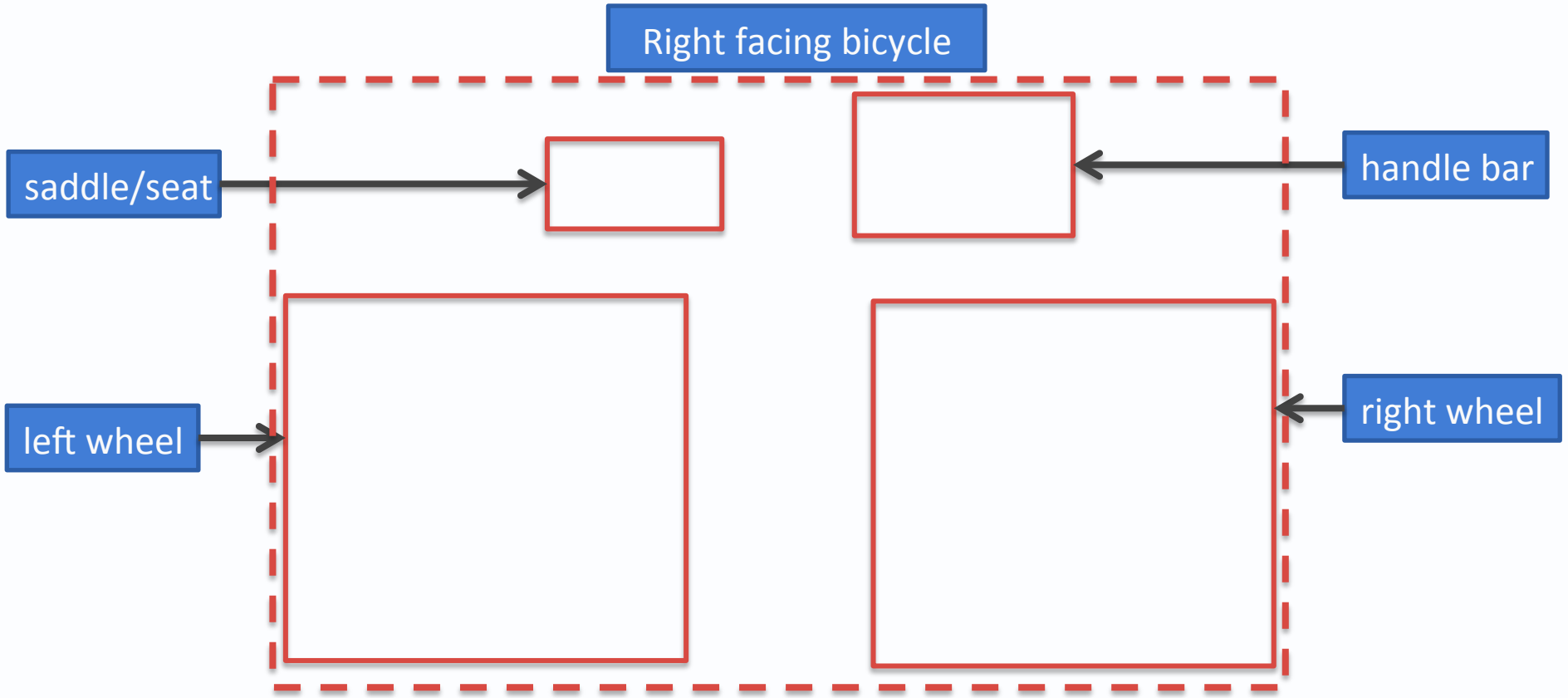
Example 2: What is in this picture?

Object detection

Object detection



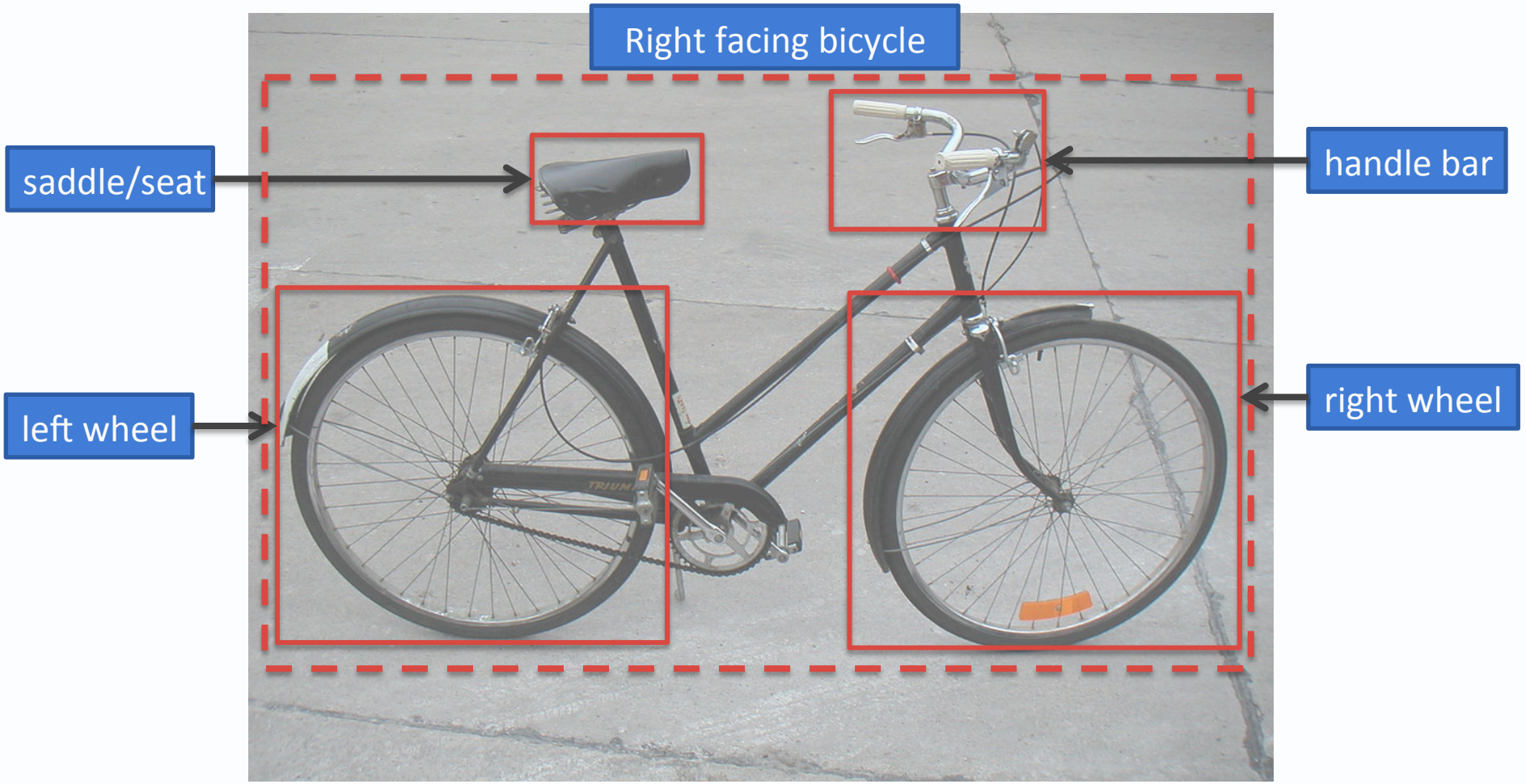
The output: A schematic showing the parts and their relative layout



Once again, a structure

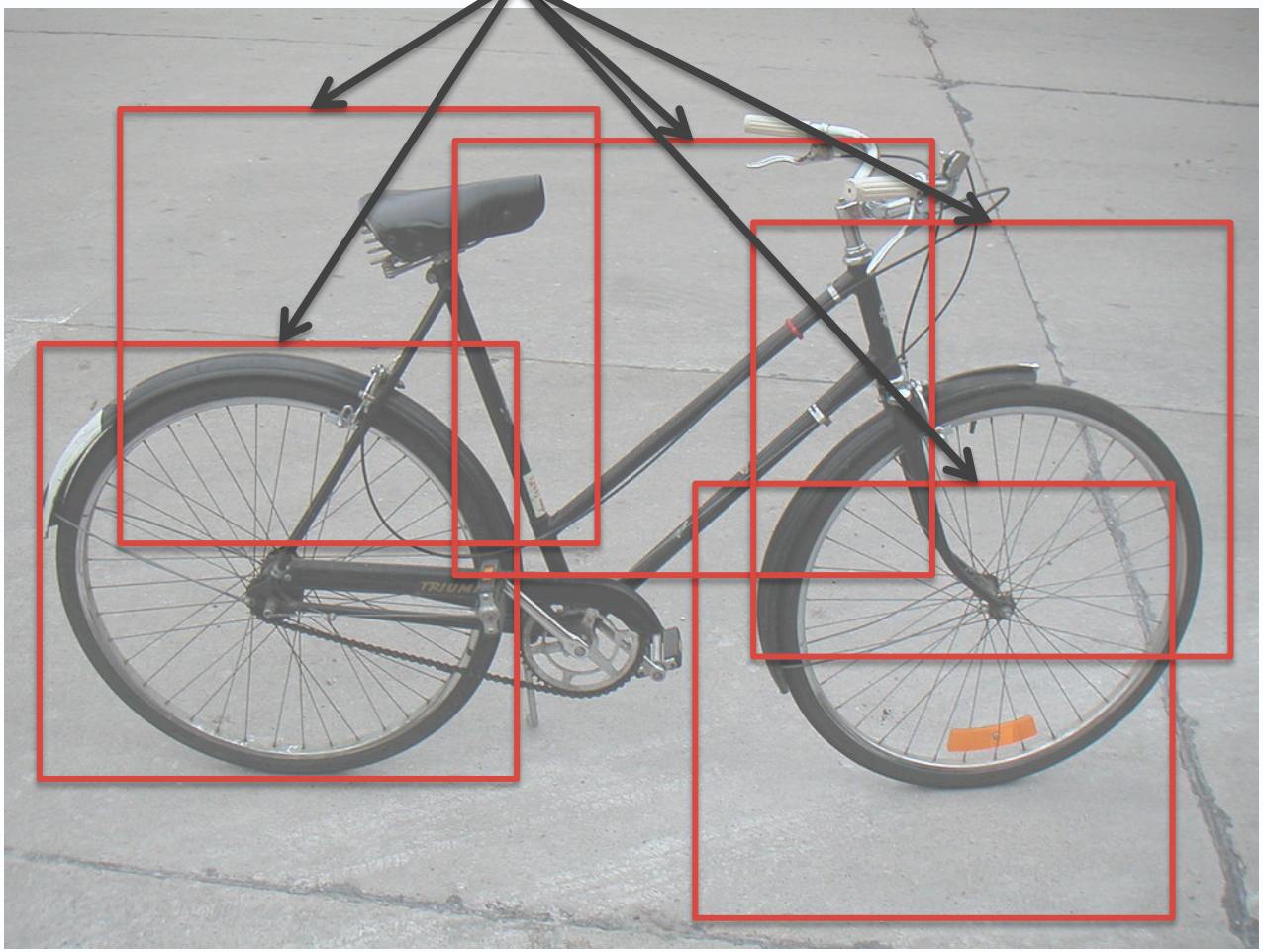
Object detection

How would you design a predictor that labels all the parts?



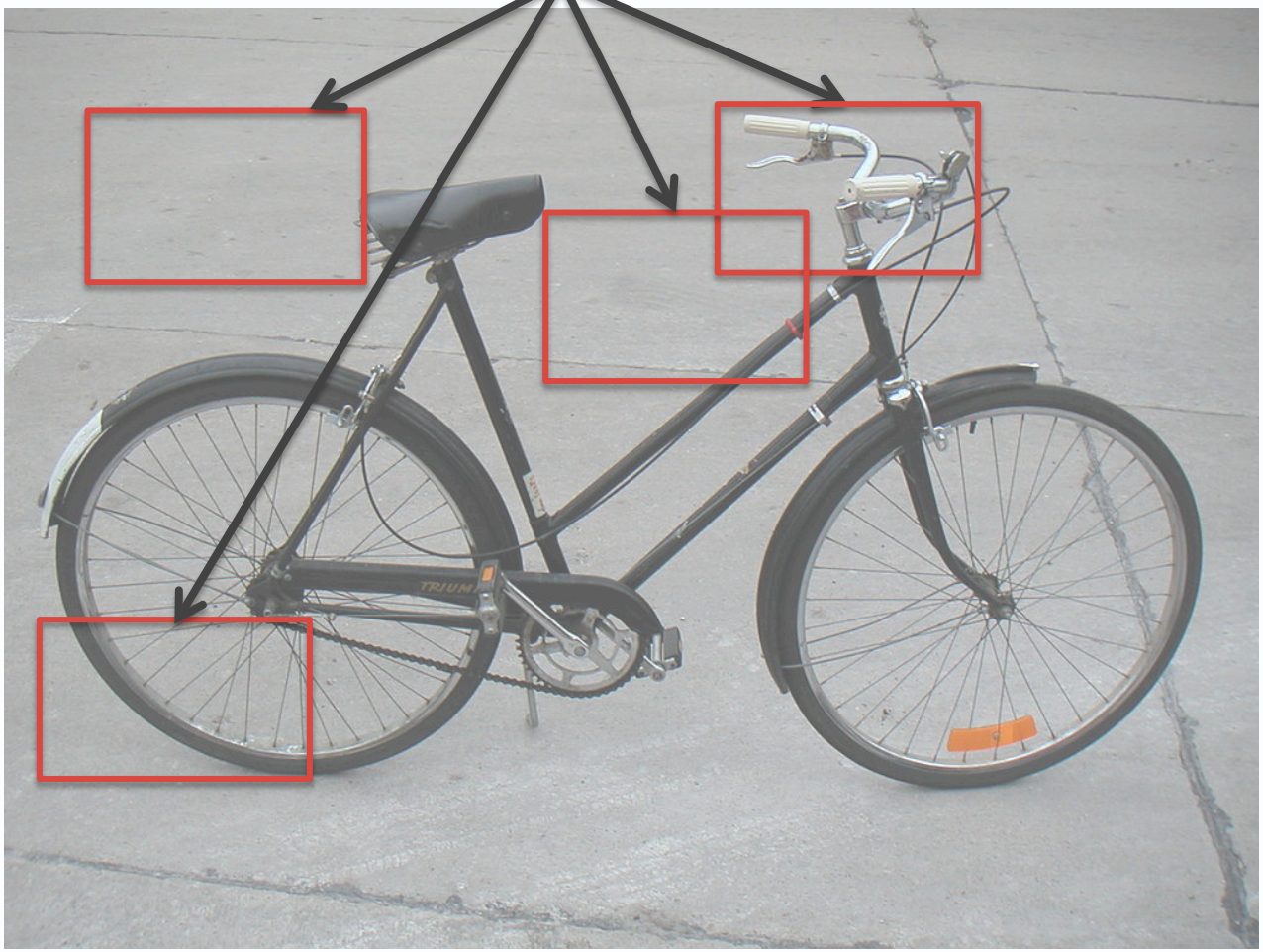
One approach to build this structure

Left wheel detector: Is there a wheel in this box? Binary classifier



One approach to build this structure

Handle bar detector: Is there a handle bar in this box? Binary classifier



One approach to build this structure

1. Left wheel detector

2. Right wheel detector

3. Handle bar detector

4. Seat detector

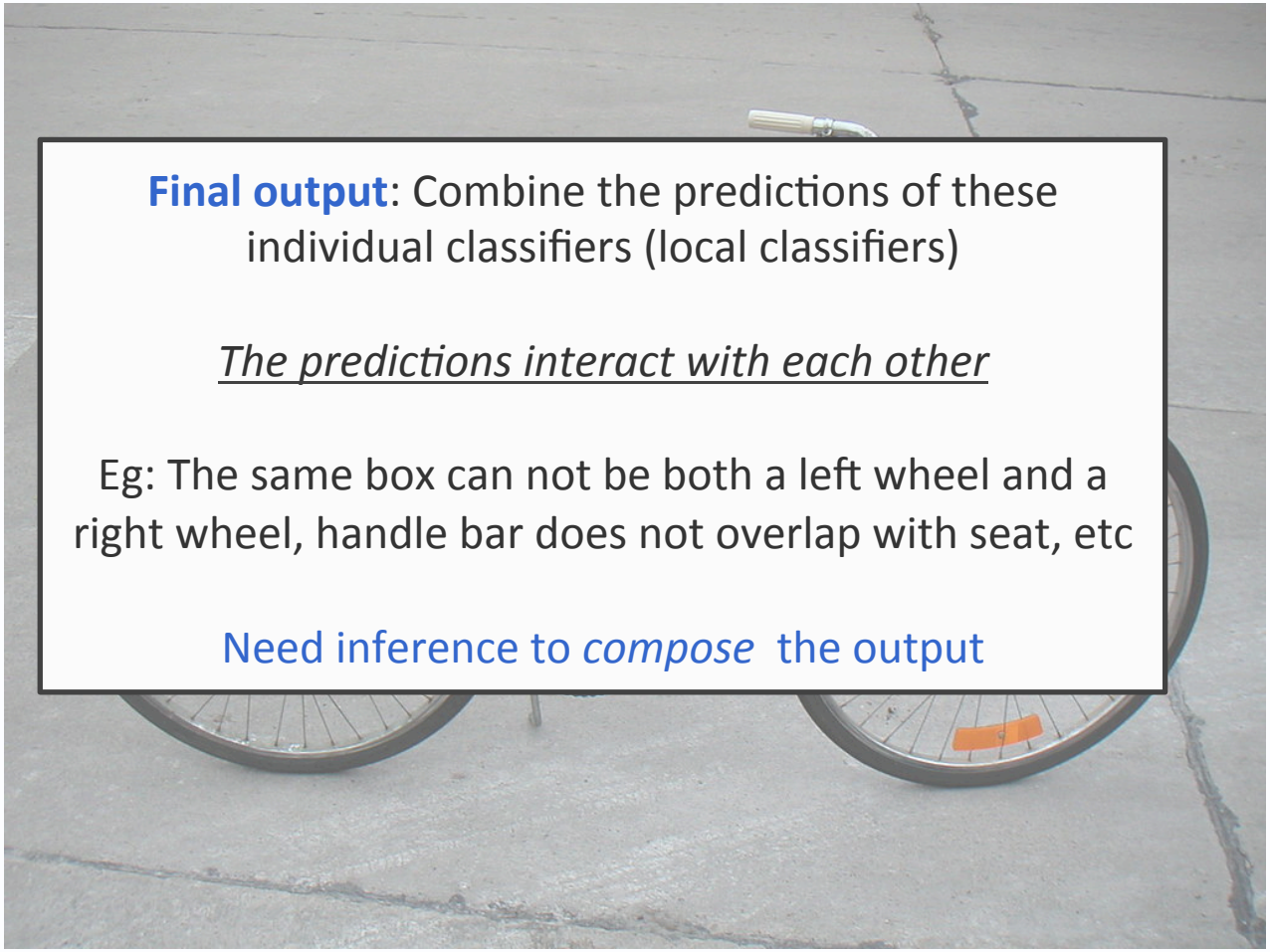
One approach to build this structure

1. Left wheel detector

2. Right wheel detector

3. Handle bar detector

4. Seat detector



Final output: Combine the predictions of these individual classifiers (local classifiers)

The predictions interact with each other

Eg: The same box can not be both a left wheel and a right wheel, handle bar does not overlap with seat, etc

Need inference to *compose* the output

Example 3: Sequence labeling

- Input: A sequence of *tokens* (like words)
- Output: A sequence of labels of same length as input

Eg: Part-of-speech tagging:

Given a sentence, find parts-of-speech of all the words

Example 3: Sequence labeling

More on this in next lecture

- Input: A sequence of *tokens* (like words)
- Output: A sequence of labels of same length as input

Eg: Part-of-speech tagging:

Given a sentence, find parts-of-speech of all the words

The	Fed	raises	interest	rates

Example 3: Sequence labeling

More on this in next lecture

- Input: A sequence of *tokens* (like words)
- Output: A sequence of labels of same length as input

Eg: Part-of-speech tagging:

Given a sentence, find parts-of-speech of all the words

The	Fed	raises	interest	rates
Determiner	Noun	Verb	Noun	Noun

Example 3: Sequence labeling

More on this in next lecture

- Input: A sequence of *tokens* (like words)
- Output: A sequence of labels of same length as input

Eg: Part-of-speech tagging:

Given a sentence, find parts-of-speech of all the words

The	Fed	raises	interest	rates
`Determiner`	`Noun`	`Verb`	`Noun`	`Noun`

Other tags possible in different contexts

`Verb` (I *fed* the dog)

`Verb` (Poems don't *interest* me)

`Verb` (He *rates* movies online)

Part-of-speech tagging

Given a word, its label depends on :

- The identity and characteristics of the word
  - Eg. Raises is a `Verb` because it ends in –es (among other reasons)
- Its grammatical context
  - Fed in “The Fed” is a `Noun` because it follows a `Determiner`
  - Fed in “I fed the..” is a `Verb` because it follows a `Pronoun`

Part-of-speech tagging

Given a word, its label depends on :

- The identity and characteristics of the word
  - Eg. Raises is a `Verb` because it ends in –es (among other reasons)
- Its grammatical context
  - Fed in “The Fed” is a `Noun` because it follows a `Determiner`
  - Fed in “I fed the..” is a `Verb` because it follows a `Pronoun`

One possible model:

Each output label is dependent on its neighbors in addition to the input

Part-of-speech tagging

Given a word, its label depends on :

- The identity and characteristics of the word
  - Eg. Raises is a `Verb` because it ends in –es (among other reasons)
- Its grammatical context
  - Fed in “The Fed” is a `Noun` because it follows a `Determiner`
  - Fed in “I fed the..” is a `Verb` because it follows a `Pronoun`

One possible model:

Each output label is dependent on its neighbors in addition to the input

Two kinds of scoring functions

1. Score for label associating with a particular word in context
2. Score for a pair of labels following each other

Part-of-speech tagging

Given a word, its label depends on :

- The identity and characteristics of the word
  - Eg. Raises is a `Verb` because it ends in –es (among other reasons)
- Its grammatical context
  - Fed in "The Fed" is a `Noun` because it follows a `Determiner`
  - Fed in "I fed the.." is a `Verb` because it follows a `Pronoun`

One possible model:

Each output label is dependent on its neighbors in addition to the input

Two kinds of scoring functions

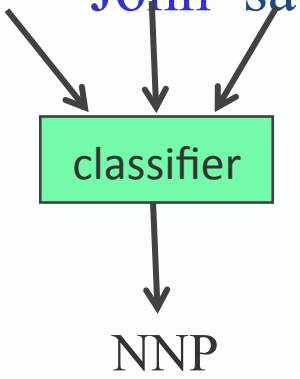
1. Score for label associating with a particular word in context
2. Score for a pair of labels following each other

What we want: Find a sequence of labels that maximizes the sum/product of these scores

Sequence Labeling as Classification

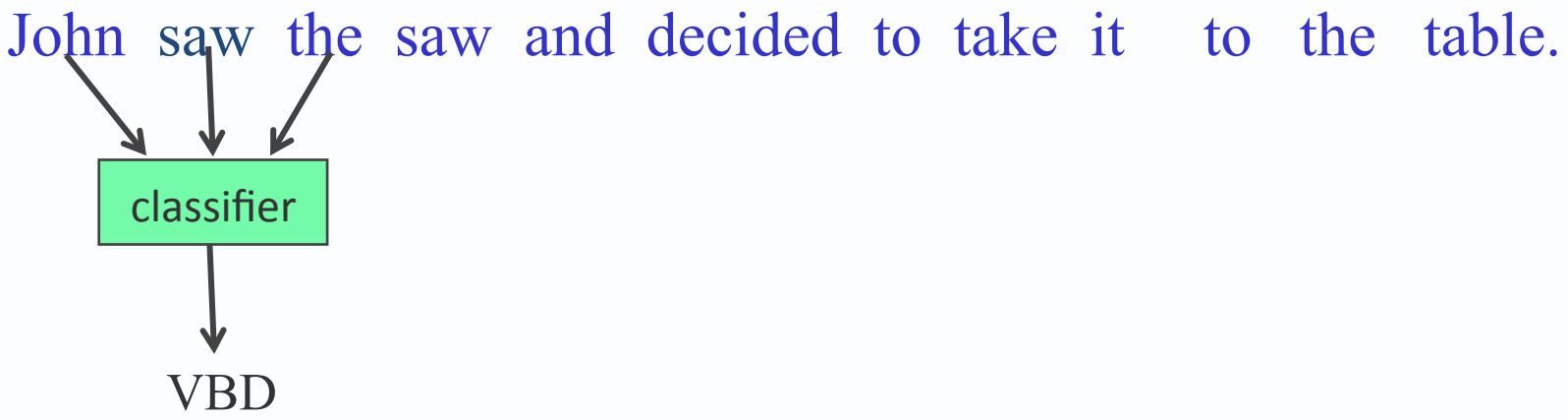
- Classify each token independently but use as input features, information about the surrounding tokens (sliding window).

John saw the saw and decided to take it to the table.



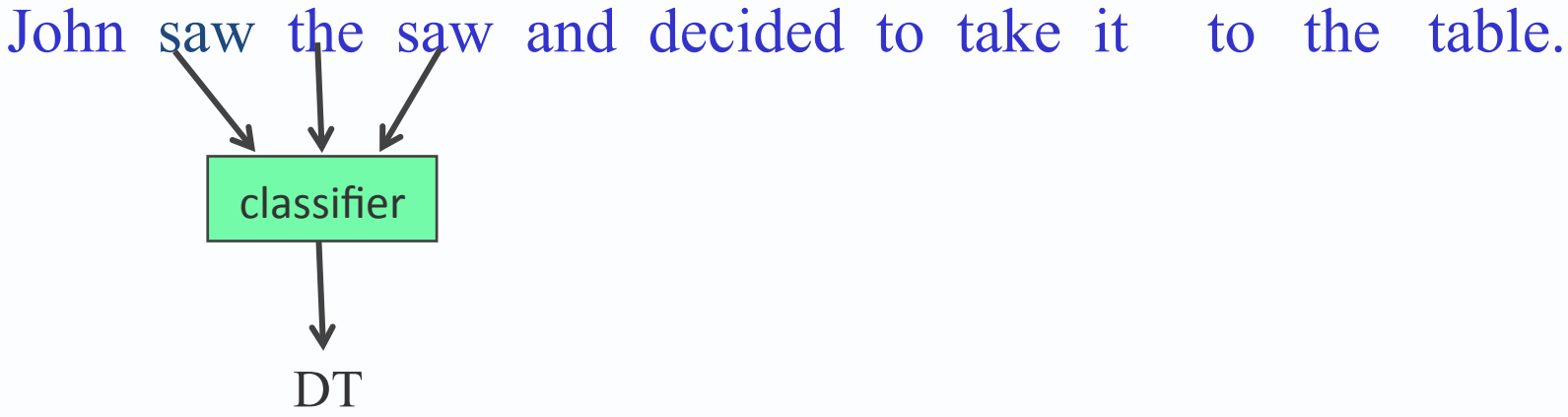
Sequence Labeling as Classification

- Classify each token independently but use as input features, information about the surrounding tokens (sliding window).



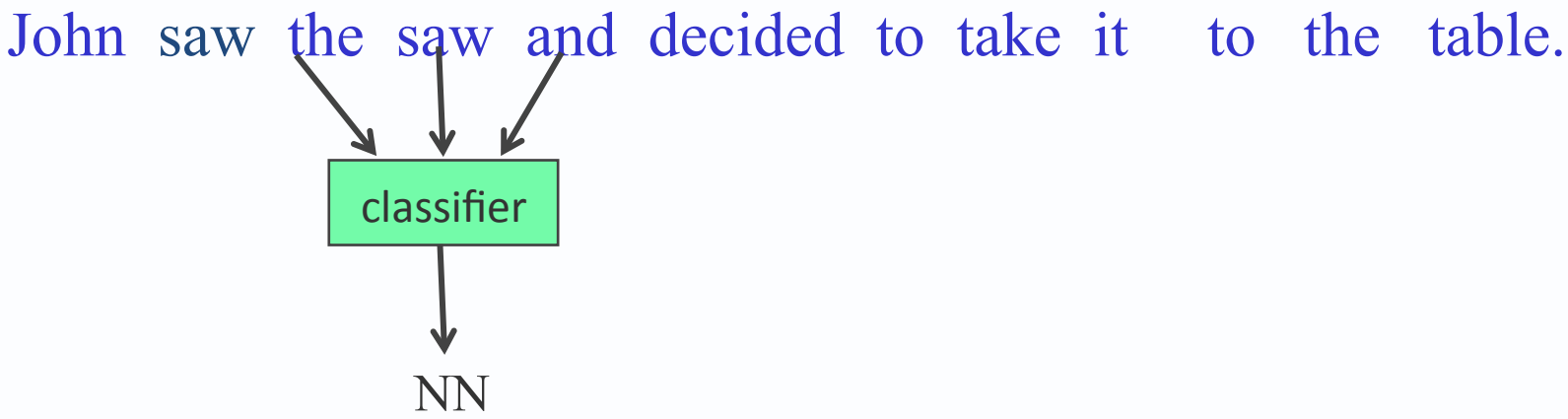
Sequence Labeling as Classification

- Classify each token independently but use as input features, information about the surrounding tokens (sliding window).



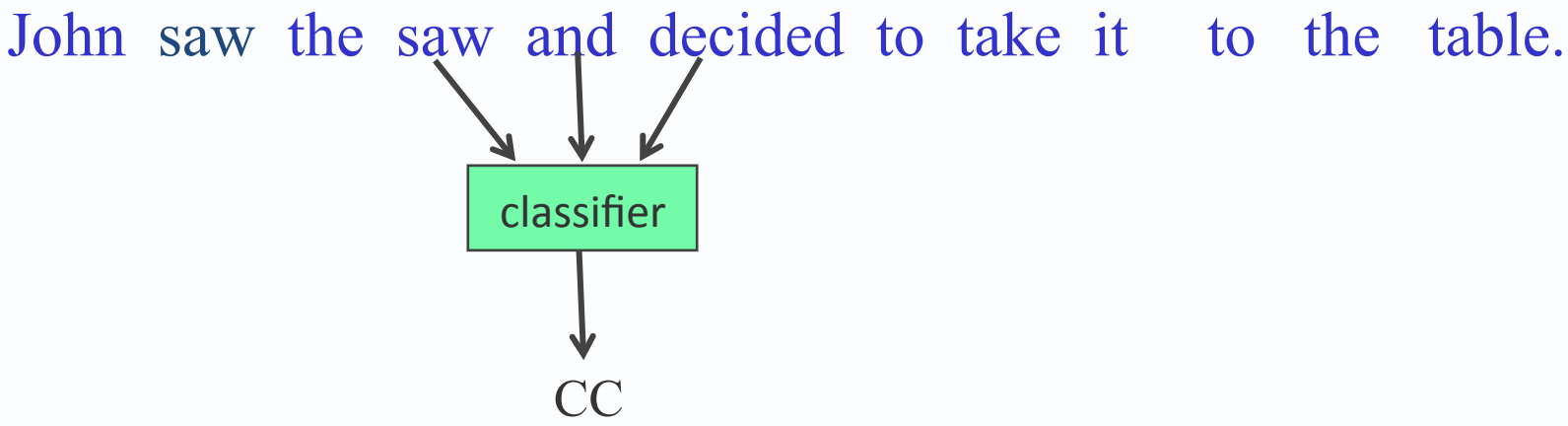
Sequence Labeling as Classification

- Classify each token independently but use as input features, information about the surrounding tokens (sliding window).



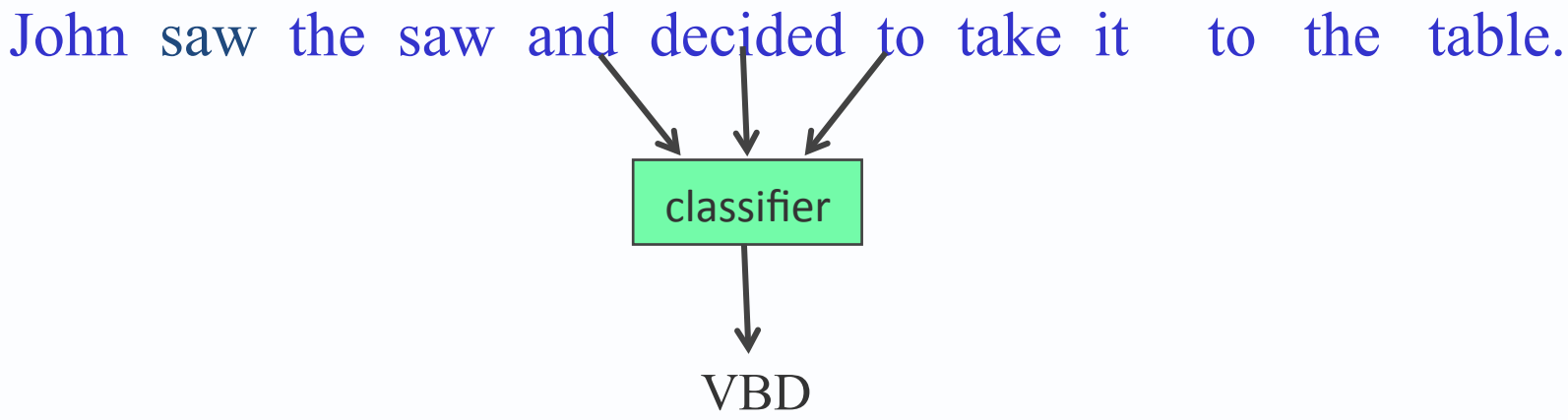
Sequence Labeling as Classification

- Classify each token independently but use as input features, information about the surrounding tokens (sliding window).



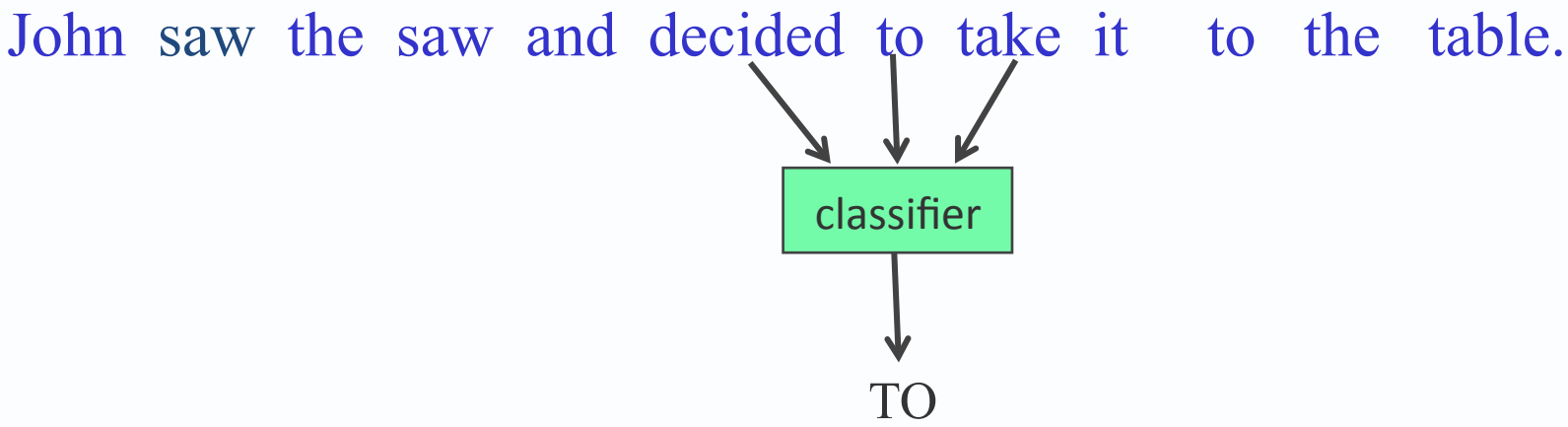
Sequence Labeling as Classification

- Classify each token independently but use as input features, information about the surrounding tokens (sliding window).



Sequence Labeling as Classification

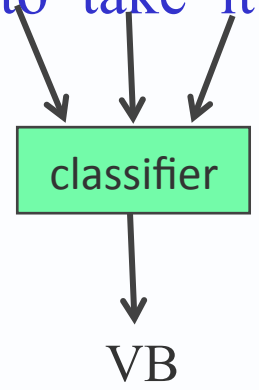
- Classify each token independently but use as input features, information about the surrounding tokens (sliding window).



Sequence Labeling as Classification

- Classify each token independently but use as input features, information about the surrounding tokens (sliding window).

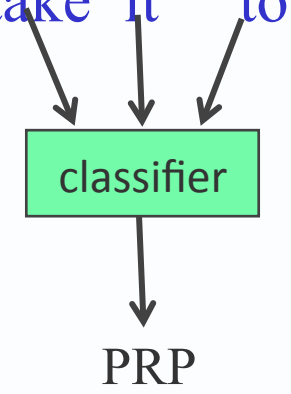
John saw the saw and decided to take it to the table.



Sequence Labeling as Classification

- Classify each token independently but use as input features, information about the surrounding tokens (sliding window).

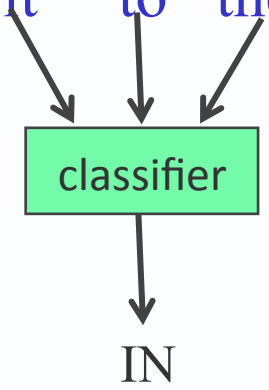
John saw the saw and decided to take it to the table.



Sequence Labeling as Classification

- Classify each token independently but use as input features, information about the surrounding tokens (sliding window).

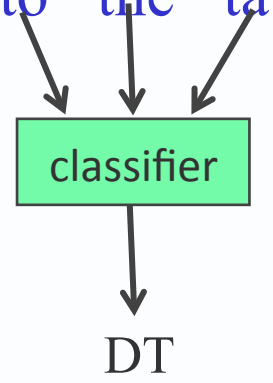
John saw the saw and decided to take it to the table.



Sequence Labeling as Classification

- Classify each token independently but use as input features, information about the surrounding tokens (sliding window).

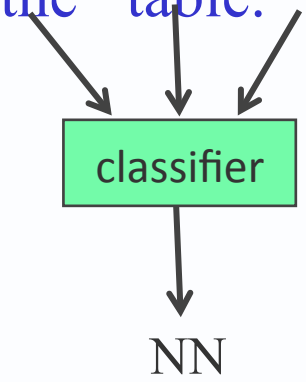
John saw the saw and decided to take it to the table.



Sequence Labeling as Classification

- Classify each token independently but use as input features, information about the surrounding tokens (sliding window).

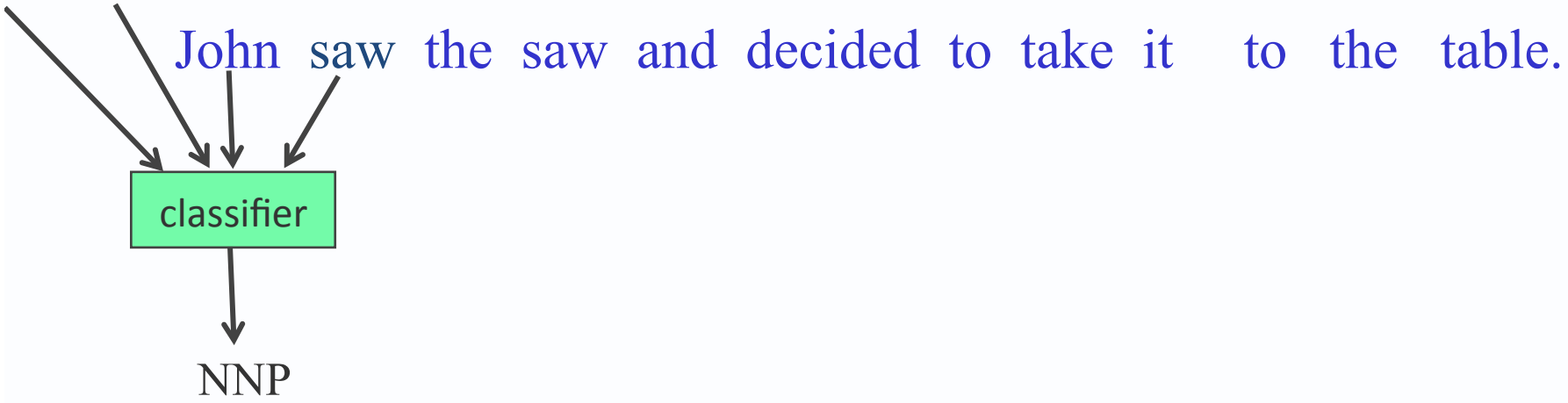
John saw the saw and decided to take it to the table.



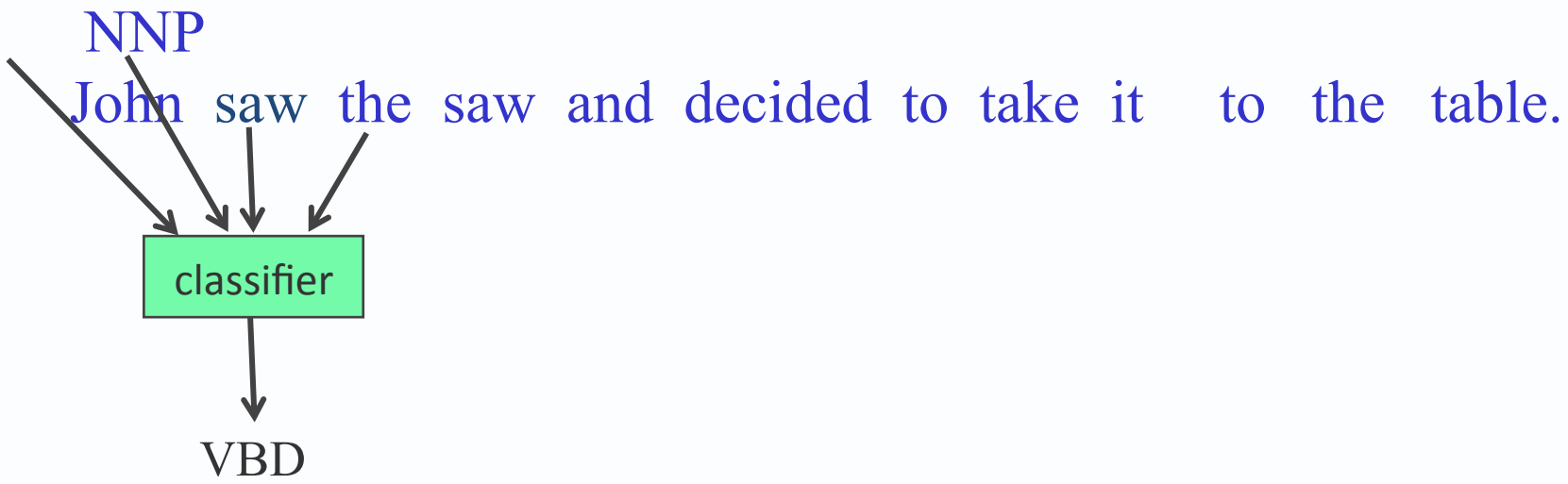
Sequence Labeling as Classification
Using Outputs as Inputs

- Better input features are usually the categories of the surrounding tokens, but these are not available yet.
- Can use category of either the preceding or succeeding tokens by going forward or back and using previous output.

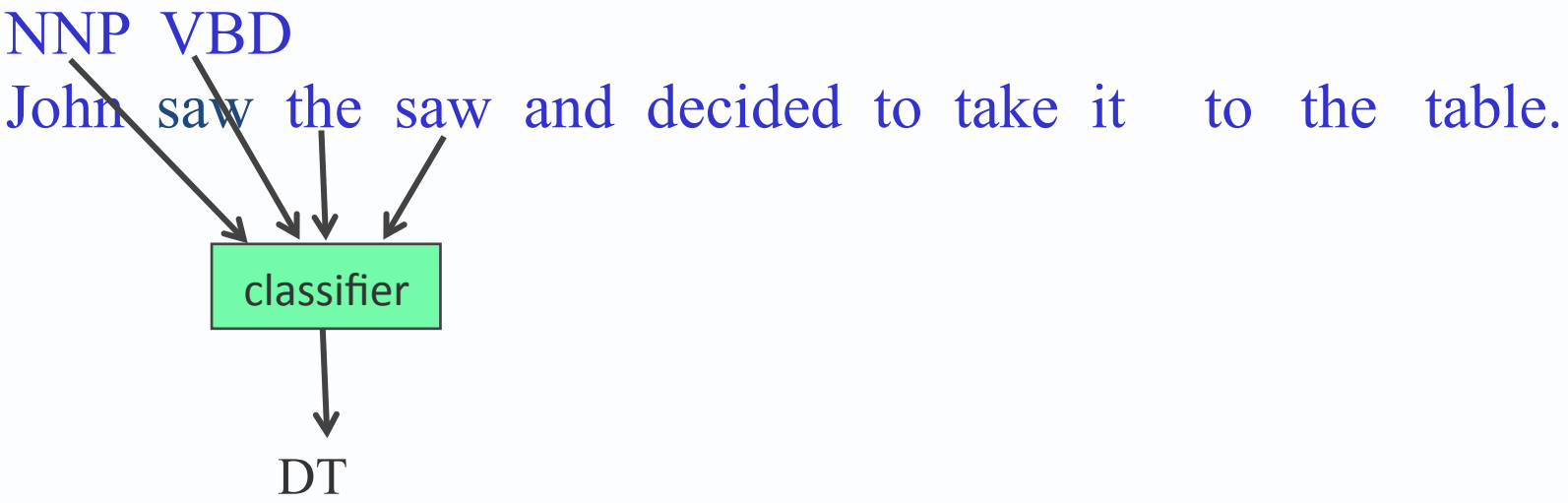
Forward Classification



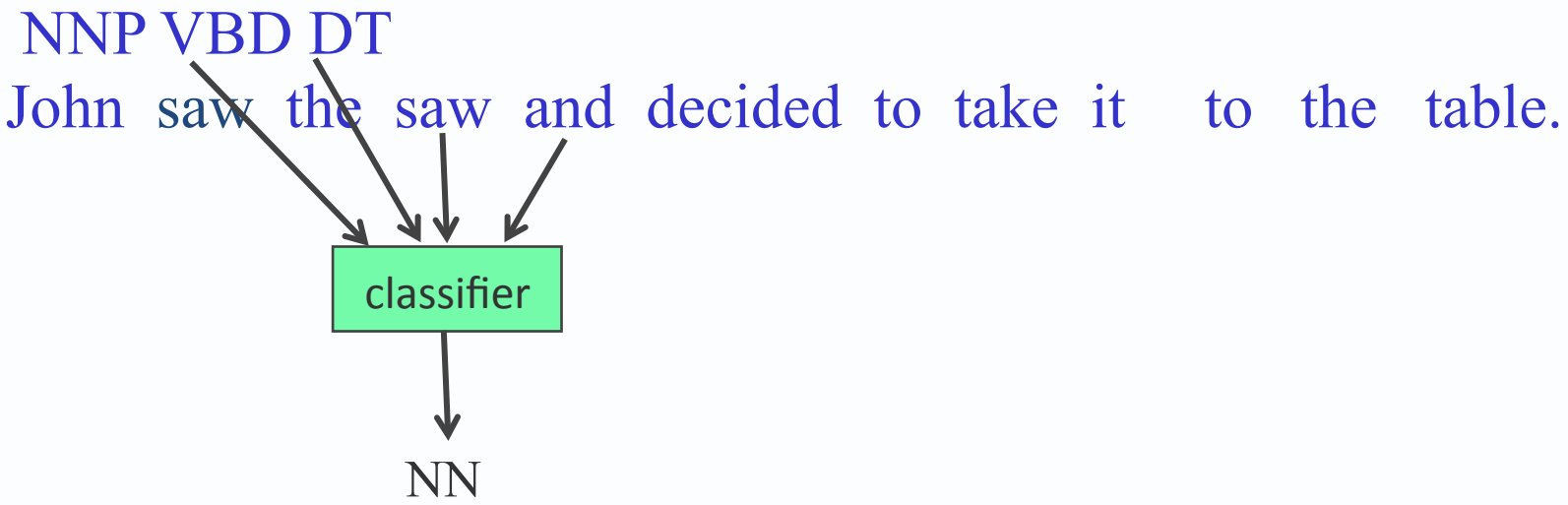
Forward Classification



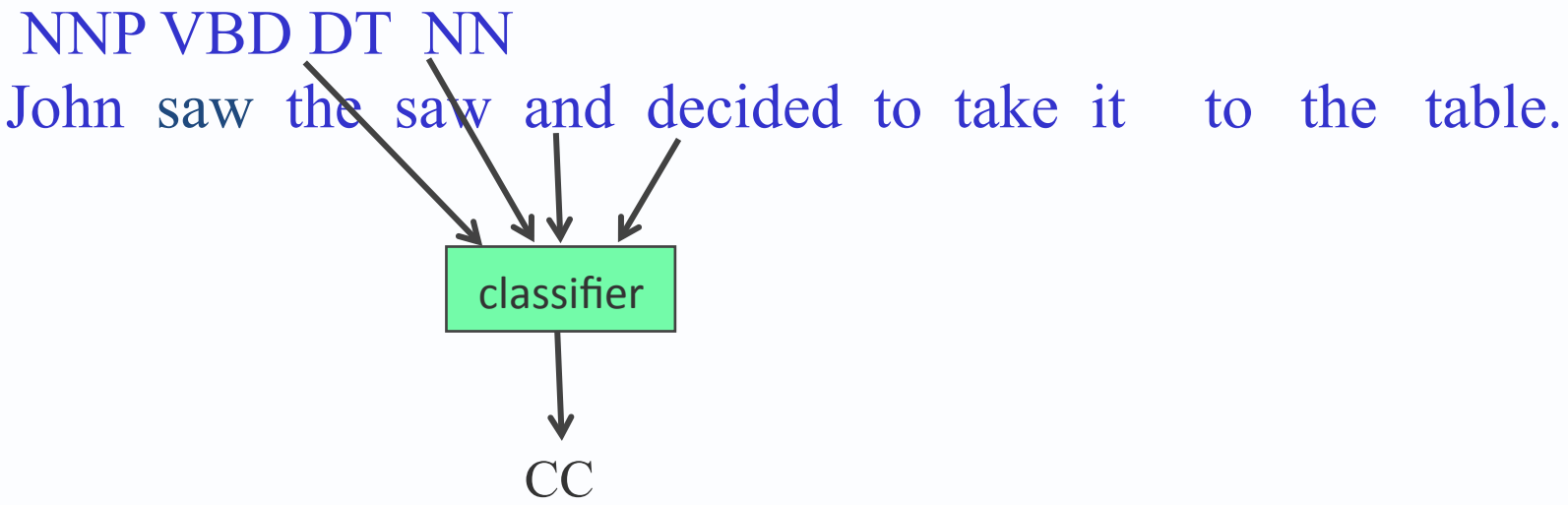
Forward Classification



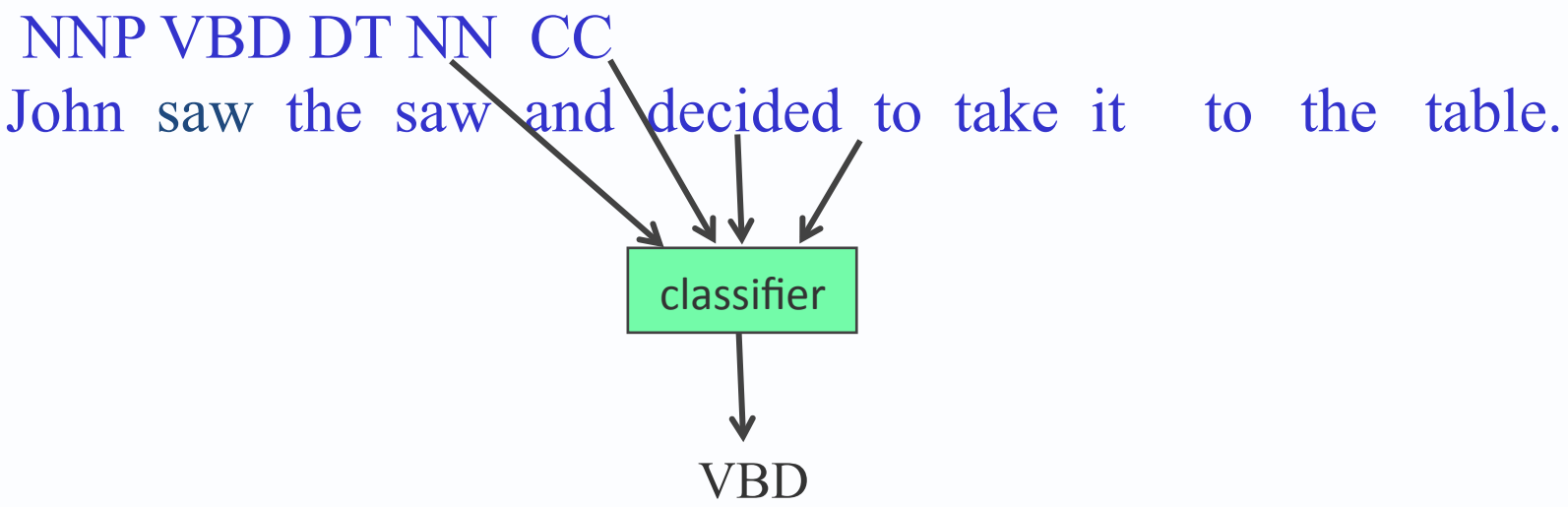
Forward Classification



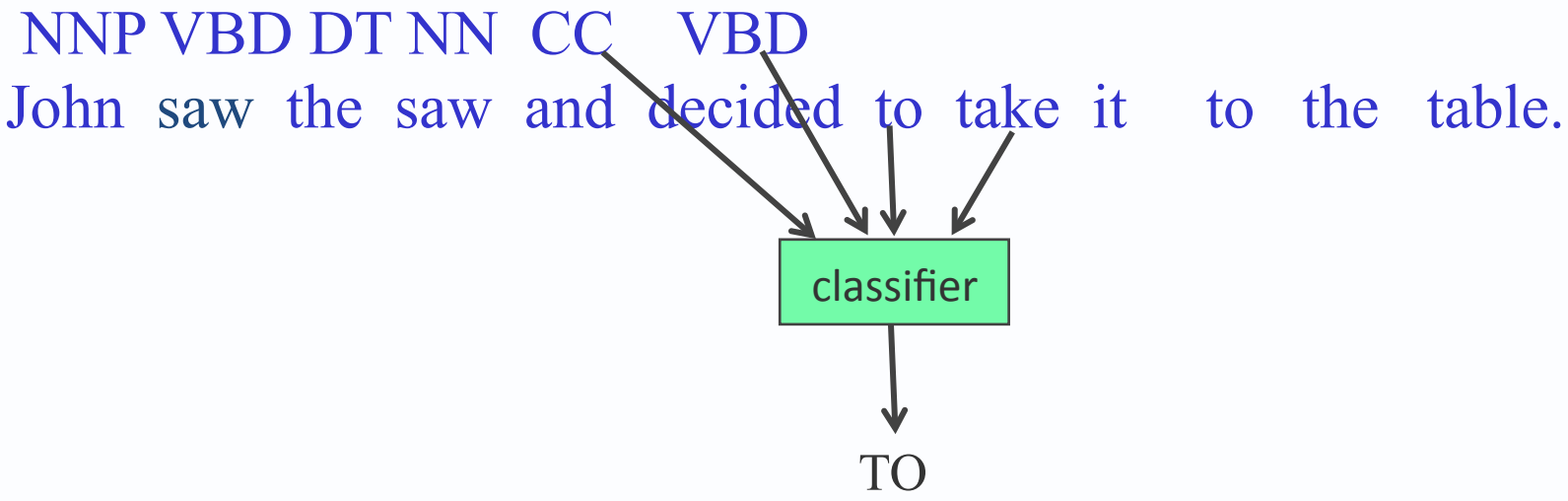
Forward Classification



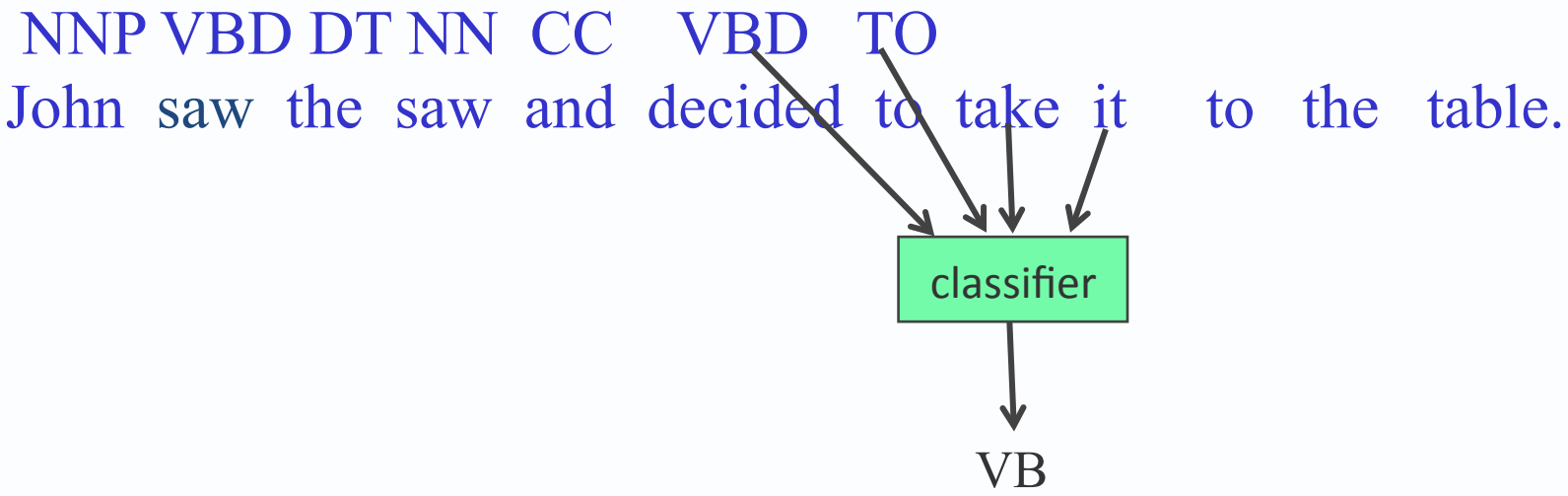
Forward Classification



Forward Classification

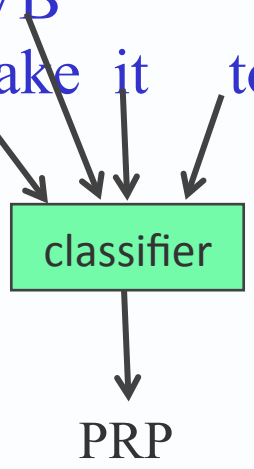


Forward Classification



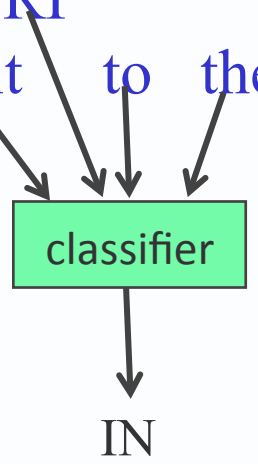
Forward Classification

NNP VBD DT NN CC VBD TO VB
John saw the saw and decided to take it to the table.



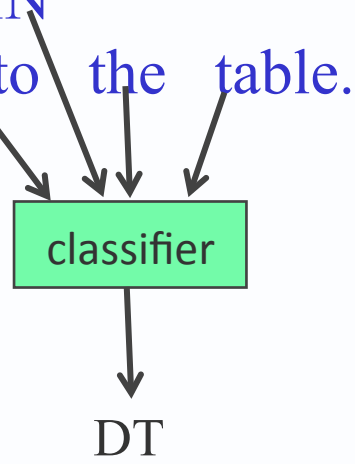
Forward Classification

NNP VBD DT NN CC VBD TO VB PRP
John saw the saw and decided to take it to the table.



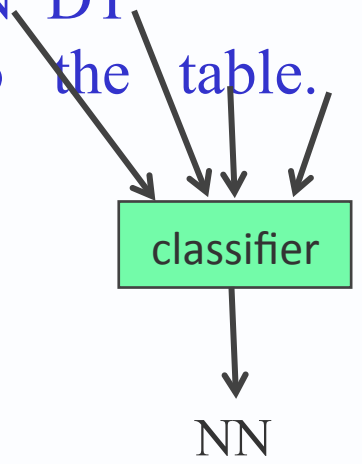
Forward Classification

NNP VBD DT NN CC VBD TO VB PRP IN
John saw the saw and decided to take it to the table.



Forward Classification

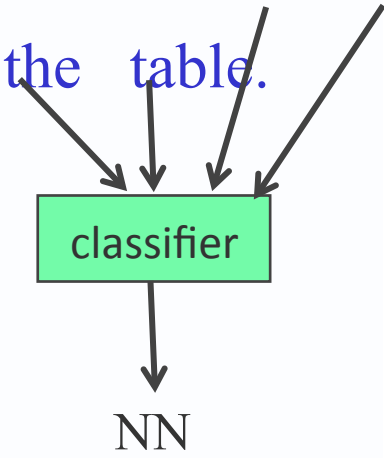
NNP VBD DT NN CC VBD TO VB PRP IN DT
John saw the saw and decided to take it to the table.



Backward Classification

- Disambiguating “to” in this case would be even easier backward.

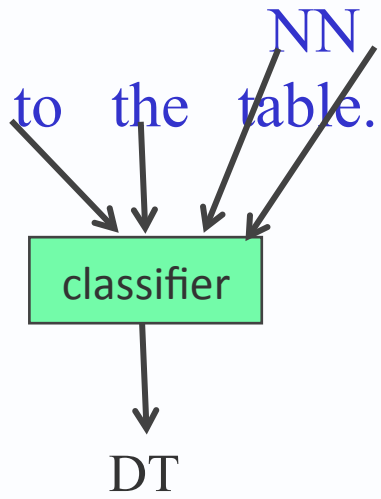
John saw the saw and decided to take it to the table.



Backward Classification

- Disambiguating “to” in this case would be even easier backward.

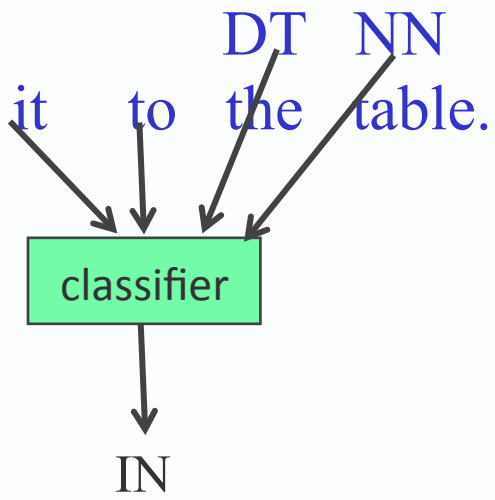
John saw the saw and decided to take it to the table.



Backward Classification

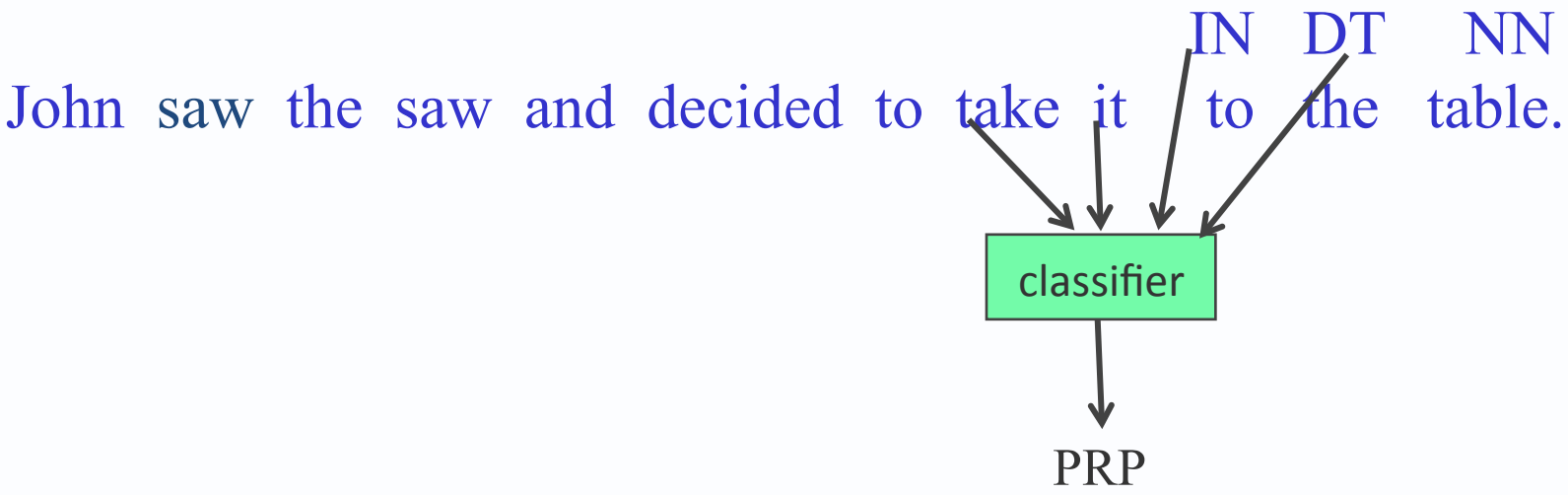
- Disambiguating “to” in this case would be even easier backward.

John saw the saw and decided to take it to the table.



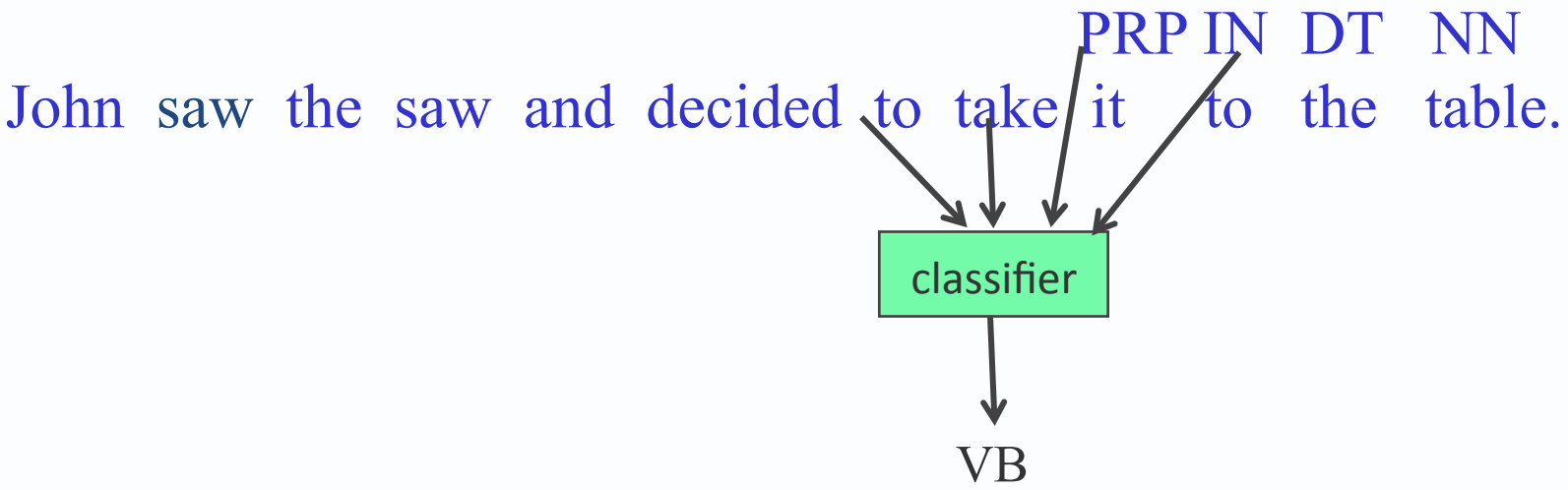
Backward Classification

- Disambiguating “to” in this case would be even easier backward.



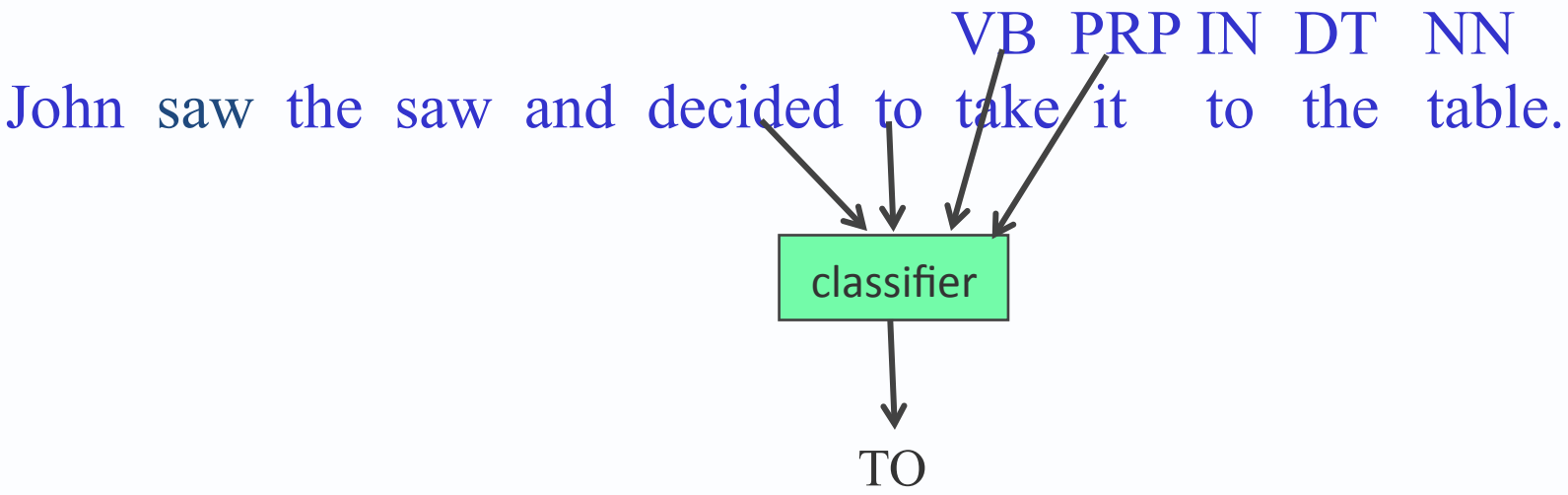
Backward Classification

- Disambiguating “to” in this case would be even easier backward.



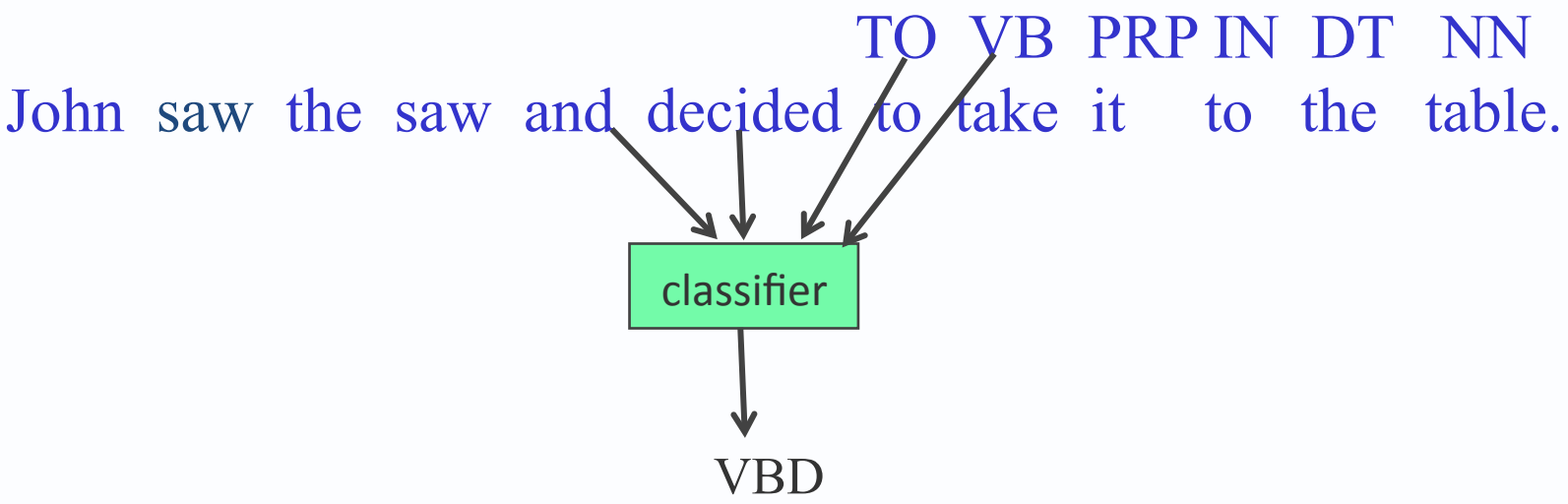
Backward Classification

- Disambiguating “to” in this case would be even easier backward.



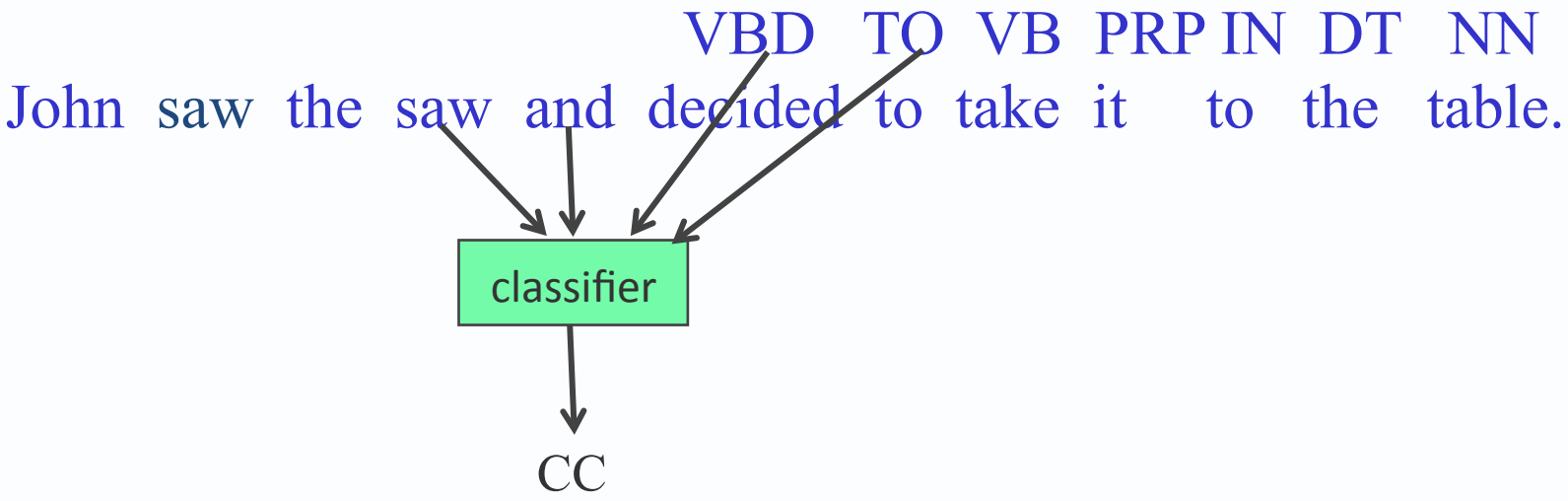
Backward Classification

- Disambiguating “to” in this case would be even easier backward.



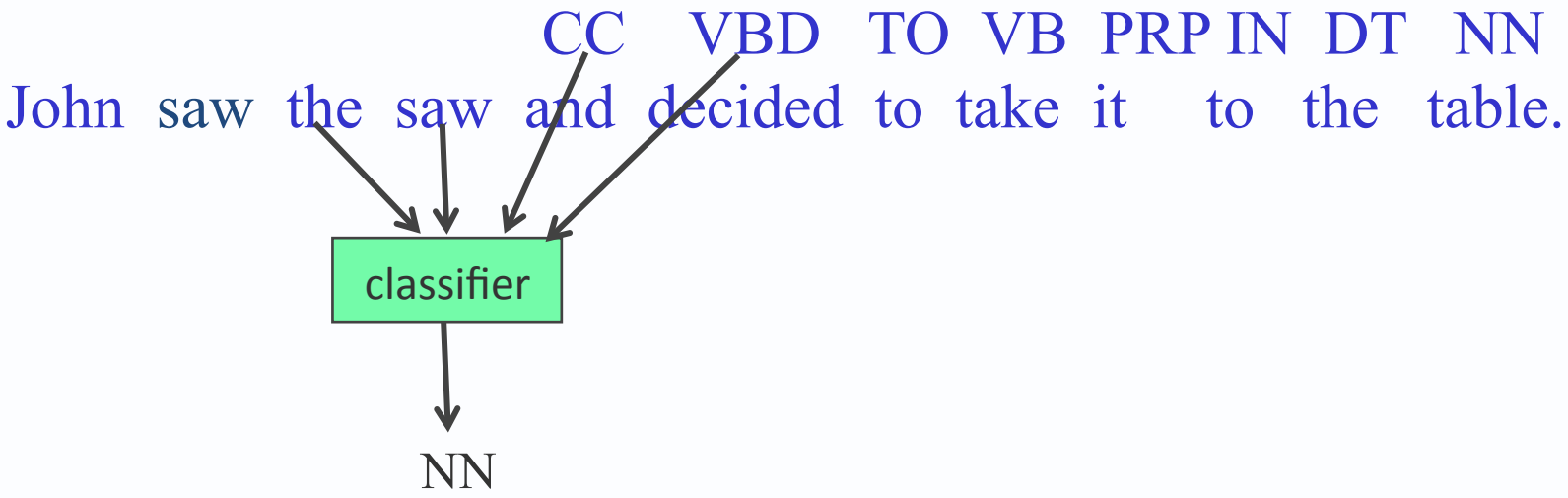
Backward Classification

- Disambiguating “to” in this case would be even easier backward.



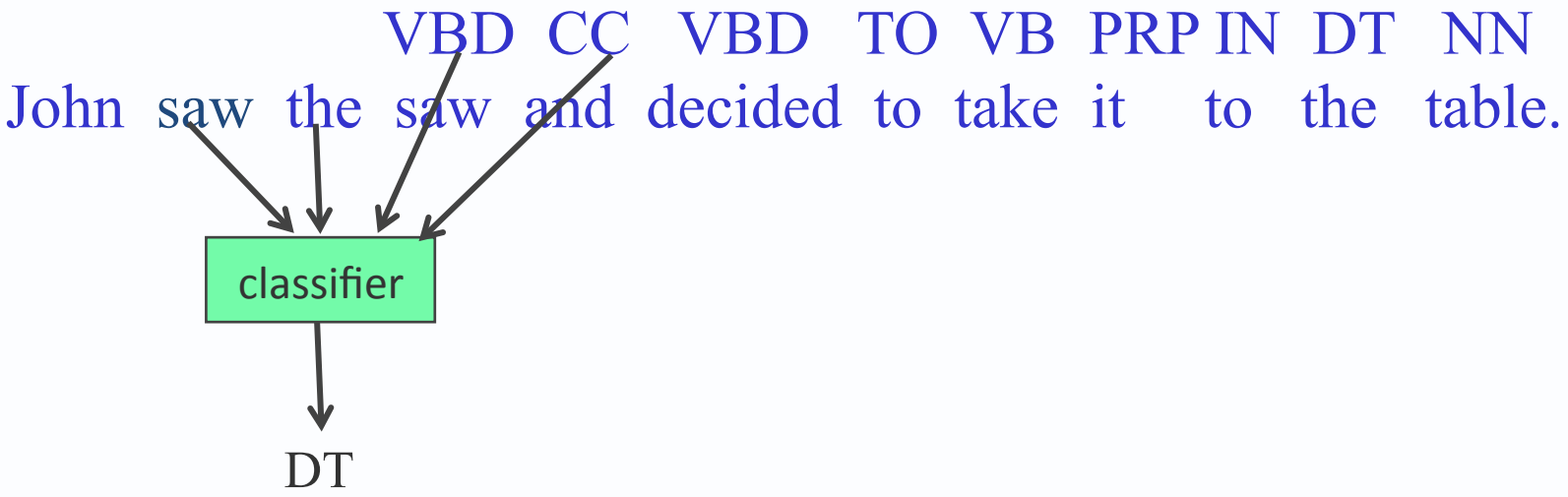
Backward Classification

- Disambiguating “to” in this case would be even easier backward.



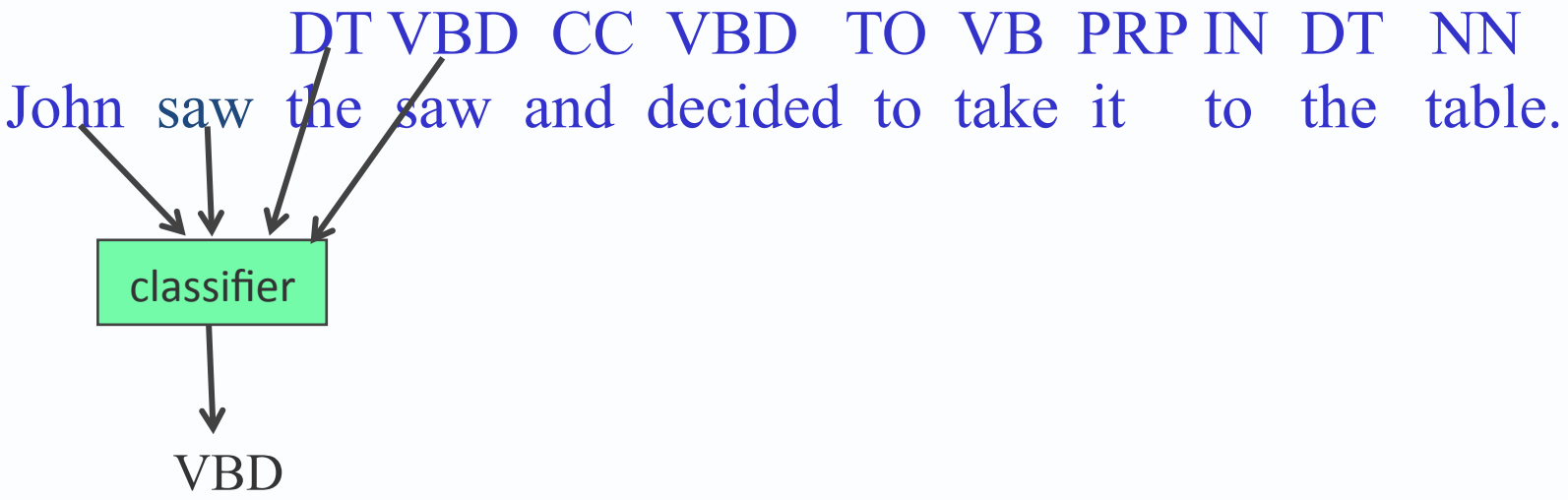
Backward Classification

- Disambiguating “to” in this case would be even easier backward.



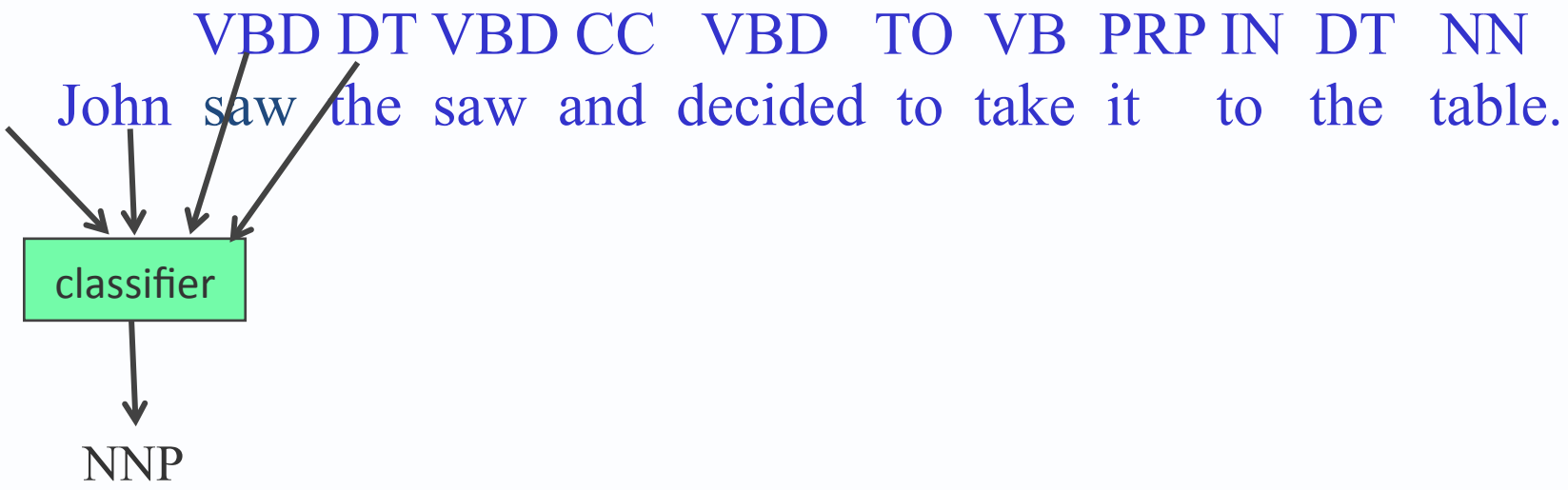
Backward Classification

- Disambiguating “to” in this case would be even easier backward.



Backward Classification

- Disambiguating “to” in this case would be even easier backward.

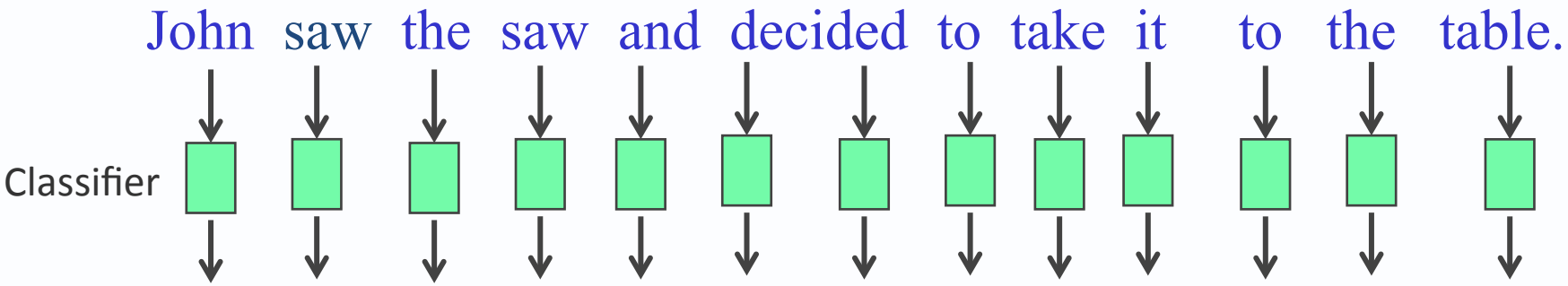


Problems with Sequence Labeling as Classification

- Not easy to integrate information from category of tokens on both sides.
- Difficult to propagate uncertainty between decisions and “collectively” determine the most likely joint assignment of categories to all of the tokens in a sequence.

Inference

- Disambiguating “to” in this case would be even easier backward.



NNP	.3	.5	.2	.7	.4	.3	.1	.4	.1	.3	.7
VB	.1	.5	.3	.1	.1	.1	0	0	0	.5	.2
DT	0	0	.1	0	.3	.5	0	0	.7	.4	0
NN	0	0	.1	0	.2	.2	.7	.4	0	0	.1
TO	.1	0	.1	.2	.1	0	.3	.1	.2	.1	0
…											

Probabilistic Sequence Models

- Probabilistic sequence models allow integrating uncertainty over multiple, interdependent classifications and collectively determine the most likely global assignment.
- Two standard models
  - Hidden Markov Model (HMM)
  - Conditional Random Field (CRF)

Predicting Sequences: Hidden Markov Models

Sequences

- Sequences of states
  - Text is a sequence of words or even letters
  - A video is a sequence of frames
- If there are K unique states, the set of unique state sequences is infinite
- Our goal: Define probability distributions over sequences
- If $x_1, x_2, \cdots, x_n$ is a sequence that has n tokens, we want to be able to define $P(x_1, x_2, \cdots, x_n)$

  …for all values of n

A history-based model

$$P(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} P(x_i | x_1, x_2, \cdots, x_{i-1})$$

- Each token is dependent on all the tokens that came before it
  - Simple conditioning
  - Each $P(x_i \mid \ldots)$ is a multinomial probability distribution over the tokens

Example: A Language model

It was a bright cold day in April.

$$P(\text{It was a bright cold day in April}) =$$

Probability of a word starting a sentence

Probability of a word following “It”

Probability of a word following “It was”

Probability of a word following “It was a”

A history-based model

$$P(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} P(x_i | x_1, x_2, \cdots, x_{i-1})$$

- Each token is dependent on all the tokens that came before it
  - Simple conditioning
  - Each $P(x_i \mid \ldots)$ is a multinomial probability distribution over the tokens
- What is the problem here?
  - How many parameters do we have?
    - Grows with the size of the sequence!

Solution: Lose the history

Discrete Markov Process

- A system can be in one of K states at a time
- State at time t is x_t
- First-order Markov assumption

  The state of the system at any time is ***independent*** of the full sequence history given the previous state

$$P(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} P(x_i | x_{i-1})$$

  – Defined by two sets of probabilities:
    - Initial state distribution: $P(x_1 = S_j)$
    - State transition probabilities: $P(x_i = S_j \mid x_{i-1} = S_k)$

Example: Another language model

It was a bright cold day in April

$$P(\text{It was a bright cold day in April}) =$$

- Probability of a word starting a sentence
- Probability of a word following “It”
- Probability of a word following “was”
- Probability of a word following “a”

If there are K tokens/states, how many parameters do we need? $O(K^2)$

m^{th} order Markov Model

- A generalization of the first order Markov Model
  - Each state is only dependent on m previous states
  - More parameters
  - But still less than storing entire history

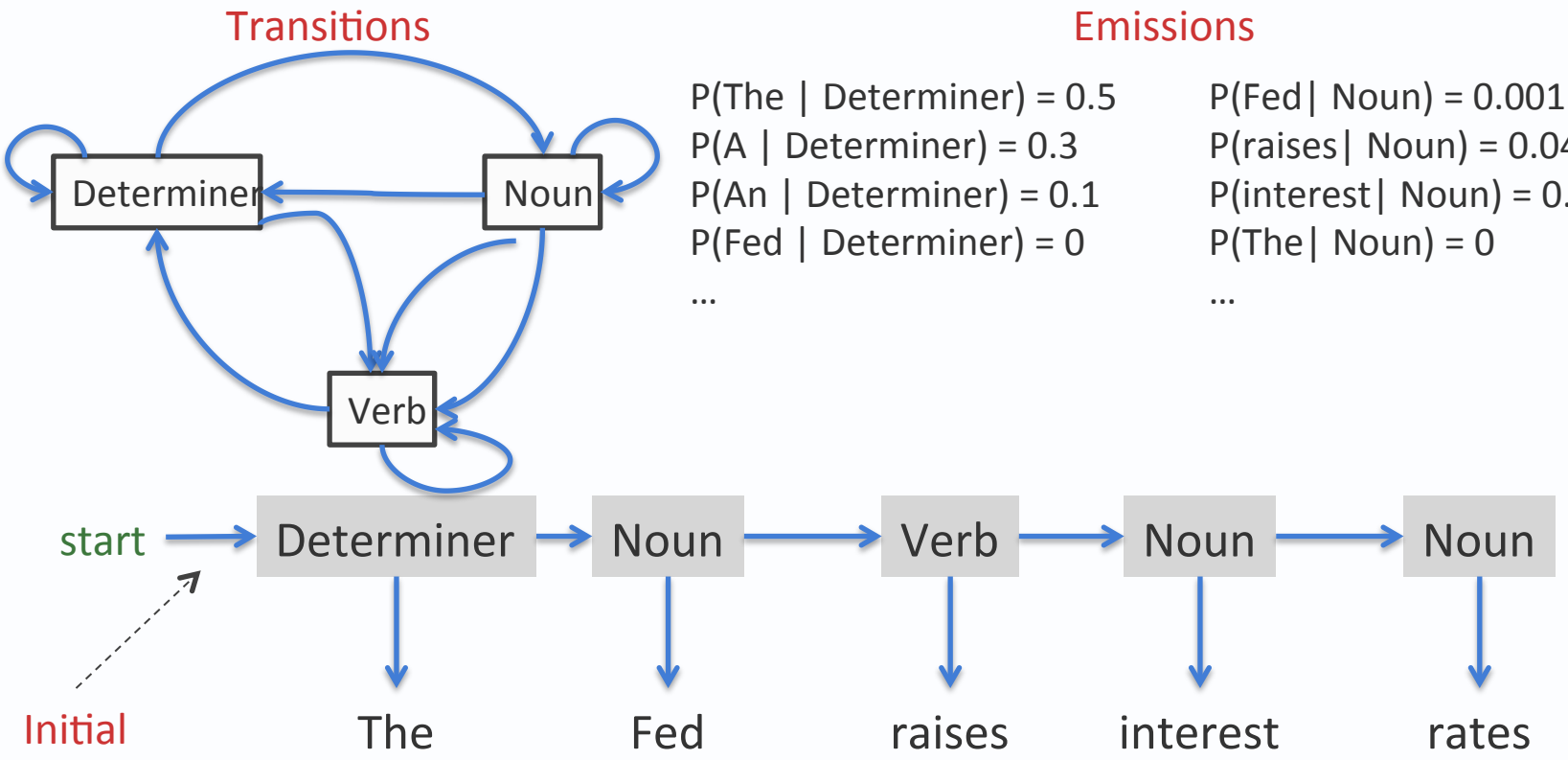
Hidden Markov Model

- Discrete Markov Model:
  - States follow a Markov chain
  - *Each state is an observation*

- Hidden Markov Model:
  - States follow a Markov chain
  - States are not observed
  - Each state stochastically emits an observation

Toy part-of-speech example

The Fed raises interest rates



Joint model over states and observations

- Notation
  - Number of states = K, Number of observations = M
  - π: Initial probability over states (K dimensional vector)
  - A: Transition probabilities (K x K matrix)
  - B: Emission probabilities (K x M matrix)

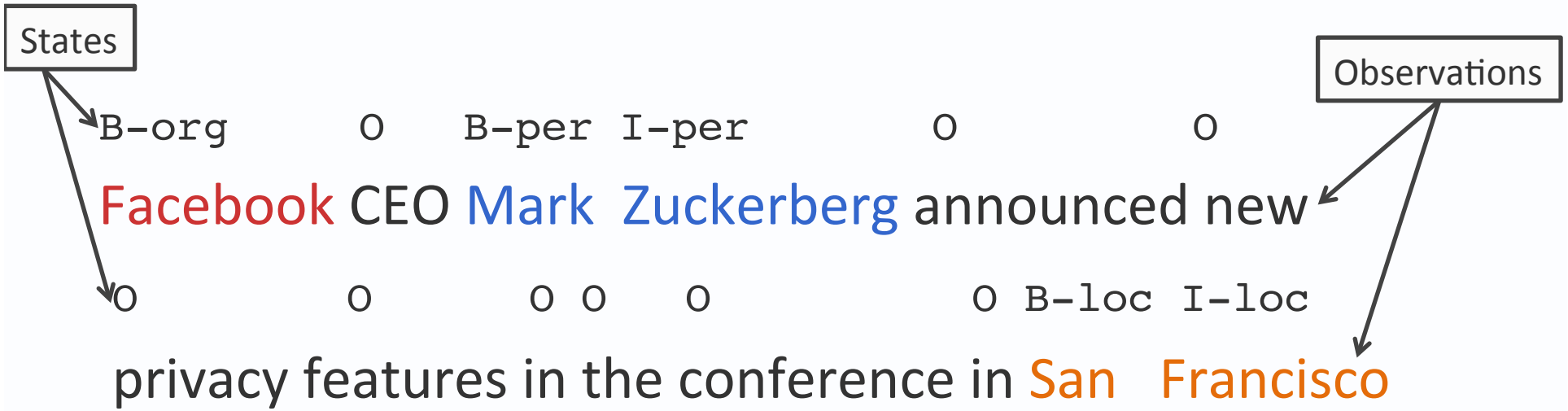
- Probability of states and observations
  - Denote states by $y_1, y_2, \cdots$ and observations by $x_1, x_2, \cdots$

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$= \pi_{y_1} \prod_{i=1}^{n-1} A_{y_i, y_{i+1}} \prod_{i=1}^{n} B_{y_i, x_i}$$

Example: Named Entity Recognition

Goal: To identify persons, locations and organizations in text



Other applications

- Speech recognition
  - Input: Speech signal
  - Output: Sequence of words
- NLP applications
  - Information extraction
  - Text chunking
- Computational biology
  - Aligning protein sequences
  - Labeling nucleotides in a sequence as exons, introns, etc.

Three questions for HMMs

[Rabiner 1999]

1. Given an observation sequence, $x_1, x_2, \cdots x_n$ and a model (π, A, B), how to efficiently calculate the probability of the observation?

2. Given an observation sequence, $x_1, x_2, \cdots, x_n$ and a model (π, A, B), how to efficiently calculate the most probable state sequence?

3. How to calculate (π, A, B) from observations?

Most likely state sequence

- Input:
  - A hidden Markov model (π, A, B)
  - An observation sequence $\mathbf{x} = (x_1, x_2, \cdots, x_n)$
- Output: A state sequence $\mathbf{y} = (y_1, y_2, \cdots, y_n)$ that corresponds to $\arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}, \pi, A, B)$
  - Maximum *a posteriori* inference (MAP inference)
- Computationally: combinatorial optimization

MAP inference

- We want $\arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}, \pi, A, B)$
- We have defined

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

- But, P(**y** | **x**, π, A, B) $\propto$ P(**x**, **y** |π, A, B)
  - And we don't care about P(**x**) we are maximizing over **y**
- So, $\arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}, \pi, A, B) = \arg\max_{\mathbf{y}} P(\mathbf{y}, \mathbf{x}|\pi, A, B)$

How many possible sequences?

The	Fed	raises	interest	rates

List of allowed tags for each word

Determiner	Verb Noun	Verb Noun	Verb Noun	Verb Noun
1	2	2	2	2

In this simple case, 16 sequences (1 x 2 x 2 x 2 x 2)

How many possible sequences?

Observations	x_1	x_2	...	x_n

List of allowed states for each observation

s_1	s_1	...	s_1
s_2	s_2		s_2
s_3	s_2		s_3
.	.		.
.	.		.
s_K	s_K		s_K

Output: One state per observation $y_i = s_j$

K^n possible sequences to consider in $\arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}, \pi, A, B)$

Naïve approaches

1. Try out every sequence
   - Score the sequence **y** as $P(\mathbf{y}|\mathbf{x}, \pi, A, B)$
   - Return the highest scoring one
   - What is the problem?
     - Correct, but slow, $O(K^n)$
2. Greedy search
   - Construct the output left to right
   - For each i, elect the best y_i using y_{i-1} and x_i
   - What is the problem?
     - Incorrect but fast, O(n)

Solution: Use the independence assumptions

Recall: The first order Markov assumption

The state at token i is only influenced by the previous state, the next state and the token itself

Given the adjacent labels, the others do not matter

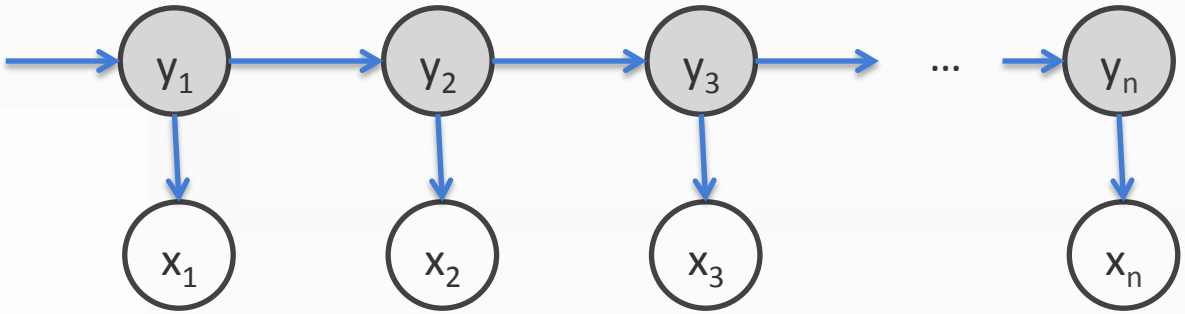
Suggests a recursive algorithm

Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

Transition probabilities Emission probabilities Initial probability



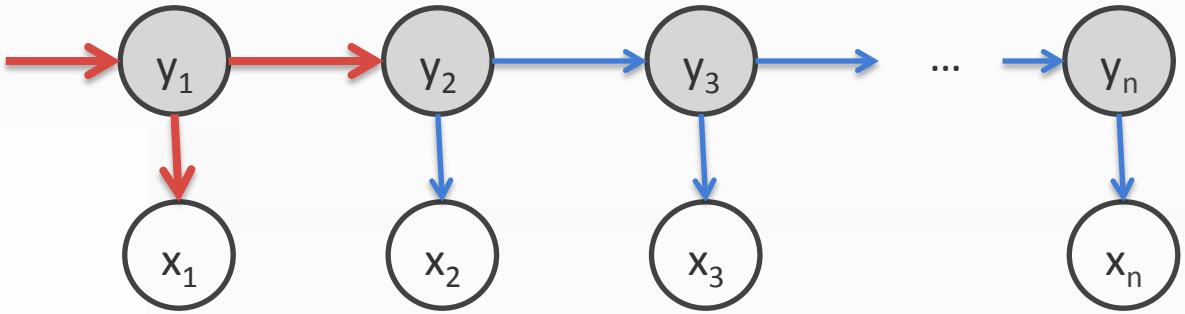
Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots P(y_2|y_1) P(x_2|y_2) P(y_1) P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1) P(x_2|y_2) P(y_1) P(x_1|y_1)$$

The only terms that depend on y_1



Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

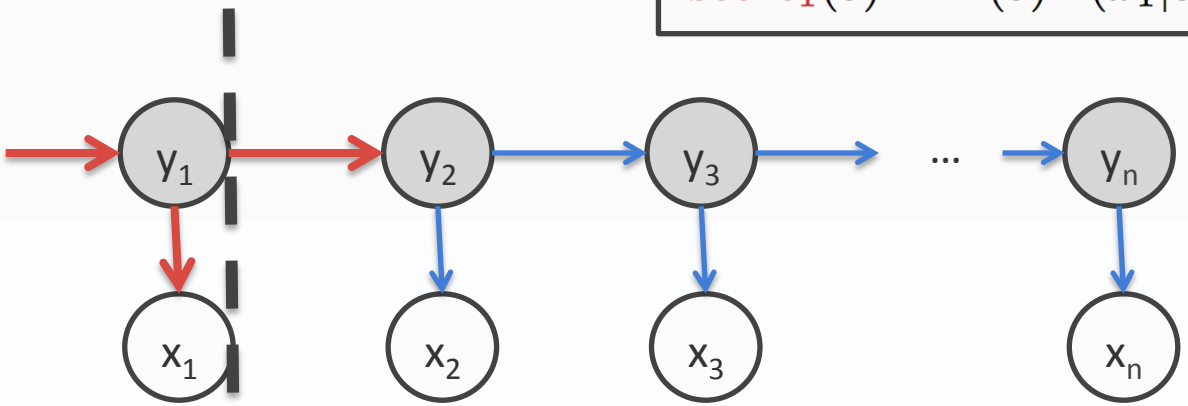
$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)\text{score}_1(y_1)$$

Abstract away the score for all decisions till here into score

$$\text{score}_1(s) = P(s)P(x_1|s)$$



Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

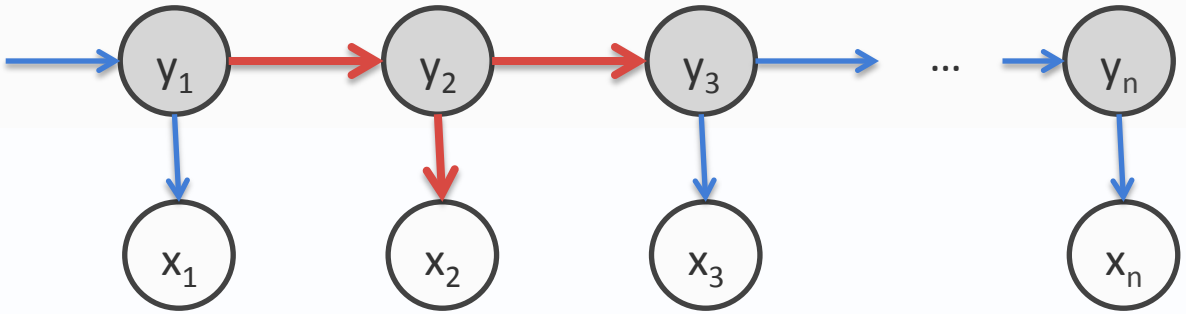
$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots P(y_2|y_1) P(x_2|y_2) P(y_1) P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1) P(x_2|y_2) P(y_1) P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1) P(x_2|y_2) \text{score}_1(y_1)$$

$$= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots \max_{y_2} P(y_3|y_2) P(x_3|y_3) \max_{y_1} P(y_2|y_1) P(x_2|y_2) \text{score}_1(y_1)$$

Only terms that depend on y_2

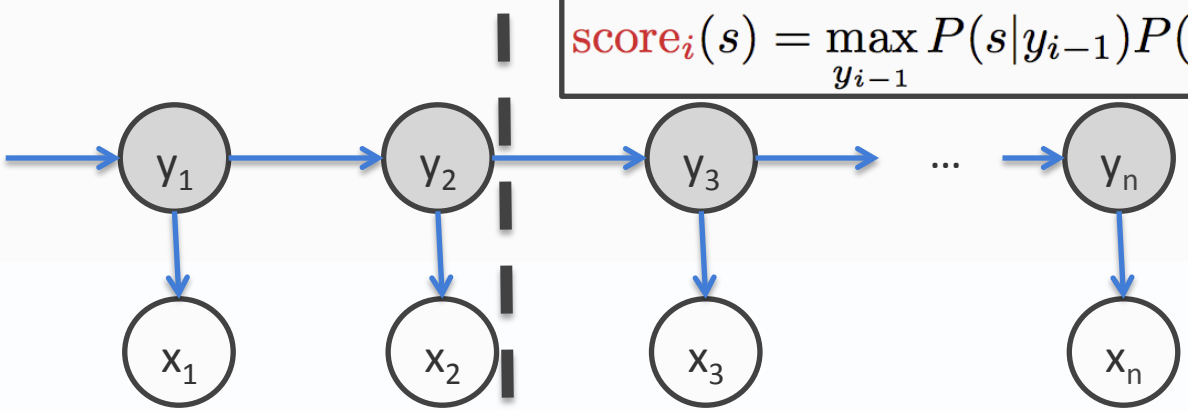


Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$\begin{aligned}
&\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1) \\
&= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1) \\
&= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)\text{score}_1(y_1) \\
&= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_2} P(y_3|y_2)P(x_3|y_3) \max_{y_1} P(y_2|y_1)P(x_2|y_2)\text{score}_1(y_1) \\
&= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_2} P(y_3|y_2)P(x_3|y_3)\text{score}_2(y_2)
\end{aligned}$$

$$\text{score}_i(s) = \max_{y_{i-1}} P(s|y_{i-1})P(x_i|s)\text{score}_{i-1}(y_{i-1})$$



Abstract away the score for all decisions till here into score

Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots P(y_2|y_1) P(x_2|y_2) P(y_1) P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1) P(x_2|y_2) P(y_1) P(x_1|y_1)$$

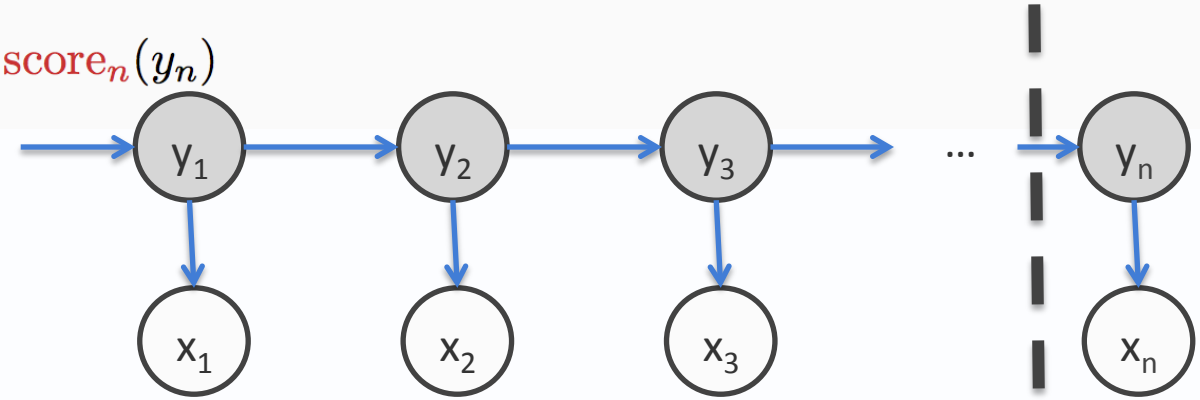
$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1) P(x_2|y_2) \text{score}_1(y_1)$$

$$= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots \max_{y_2} P(y_3|y_2) P(x_3|y_3) \max_{y_1} P(y_2|y_1) P(x_2|y_2) \text{score}_1(y_1)$$

$$= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots \max_{y_2} P(y_3|y_2) P(x_3|y_3) \text{score}_2(y_2)$$

$$\vdots$$

$$= \max_{y_n} \text{score}_n(y_n)$$



Abstract away the score for all decisions till here into score

Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$\begin{aligned}
&\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1) \\
&= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1) \\
&= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)\text{score}_1(y_1) \\
&= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_2} P(y_3|y_2)P(x_3|y_3) \max_{y_1} P(y_2|y_1)P(x_2|y_2)\text{score}_1(y_1) \\
&= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_2} P(y_3|y_2)P(x_3|y_3)\text{score}_2(y_2) \\
&\quad \vdots \\
&= \max_{y_n} \text{score}_n(y_n)
\end{aligned}$$

$$\text{score}_1(s) = P(s)P(x_1|s)$$

$$\text{score}_i(s) = \max_{y_{i-1}} P(s|y_{i-1})P(x_i|s)\text{score}_{i-1}(y_{i-1})$$

Viterbi algorithm

Max-product algorithm for first order sequences

π: Initial probabilities
A: Transitions
B: Emissions

1. Initial: For each state s, calculate

$$\text{score}_1(s) = P(s)P(x_1|s) = \pi_s B_{x_1,s}$$

2. Recurrence: For i = 2 to n, for every state s, calculate

$$\begin{aligned}\text{score}_i(s) &= \max_{y_{i-1}} P(s|y_{i-1})P(x_i|s)\text{score}_{i-1}(y_{i-1}) \\ &= \max_{y_{i-1}} A_{y_{i-1},s}B_{s,x_i}\text{score}_{i-1}(y_{i-1})\end{aligned}$$

3. Final state: calculate

$$\max_{\mathbf{y}} P(\mathbf{y}, \mathbf{x}|\pi, A, B) = \max_s \text{score}_n(s)$$

This only calculates the max. To get final answer (*argmax*),

- keep track of which state corresponds to the max at each step
- build the answer using these back pointers

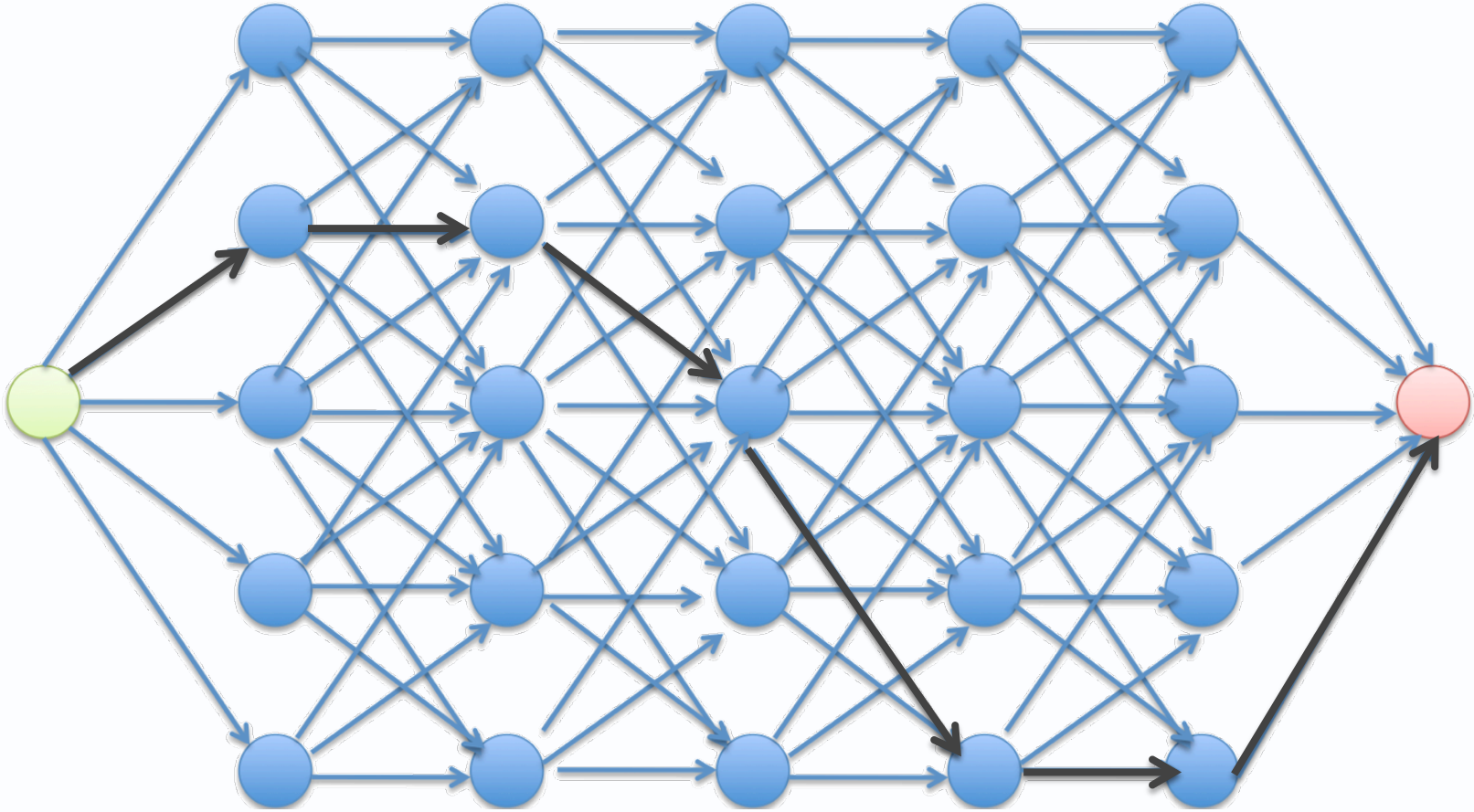
General idea

- Dynamic programming
  - The best solution for the full problem relies on best solution to sub-problems
  - Memoize partial computation

- Examples
  - Viterbi algorithm
  - Dijkstra's shortest path algorithm
  - CKY
  - …

Viterbi algorithm as best path

Goal: To find the highest scoring path in this trellis



Complexity of inference

- Complexity parameters
  - Input sequence length: n
  - Number of states: K
- Memory
  - Storing the table: nK (scores for all states at each position)
- Runtime
  - At each step, go over pairs of states
  - $O(nK^2)$

Example: POS tagging

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	*and, but, or*	SYM	symbol	+,%, &
CD	cardinal number	*one, two*	TO	"to"	*to*
DT	determiner	*a, the*	UH	interjection	*ah, oops*
EX	existential 'there'	*there*	VB	verb base form	*eat*
FW	foreign word	*mea culpa*	VBD	verb past tense	*ate*
IN	preposition/sub-conj	*of, in, by*	VBG	verb gerund	*eating*
JJ	adjective	*yellow*	VBN	verb past participle	*eaten*
JJR	adj., comparative	*bigger*	VBP	verb non-3sg pres	*eat*
JJS	adj., superlative	*wildest*	VBZ	verb 3sg pres	*eats*
LS	list item marker	*1, 2, One*	WDT	wh-determiner	*which, that*
MD	modal	*can, should*	WP	wh-pronoun	*what, who*
NN	noun, sing. or mass	*llama*	WP$	possessive wh-	*whose*
NNS	noun, plural	*llamas*	WRB	wh-adverb	*how, where*
NNP	proper noun, sing.	*IBM*	$	dollar sign	*$*
NNPS	proper noun, plural	*Carolinas*	#	pound sign	*#*
PDT	predeterminer	*all, both*	"	left quote	' or "
POS	possessive ending	*'s*	"	right quote	' or "
PRP	personal pronoun	*I, you, he*	(	left parenthesis	[, (, {, <
PRP$	possessive pronoun	*your, one's*	)	right parenthesis	],), }, >
RB	adverb	*quickly, never*	,	comma	,
RBR	adverb, comparative	*faster*	.	sentence-final punc	. ! ?
RBS	adverb, superlative	*fastest*	:	mid-sentence punc	: ; ... – -
RP	particle	*up, off*			

J&M Fig 5.6: Penn Treebank POS tags

Example: POS tagging

$t_{i-1} \backslash t_i$	NNP	MD	VB	JJ	NN	. . .
<s>	0.2767	0.0006	0.0031	0.0453	0.0449	. . .
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	. . .
MD	0.0008	0.0002	0.7968	0.0005	0.0008	. . .
VB	0.0322	0.0005	0.0050	0.0837	0.0615	. . .
JJ	0.0306	0.0004	0.0001	0.0733	0.4509	. . .
. . .	. . .	. . .	. . .	. . .	. . .	. . .

- Probabilities estimated from tagged WSJ corpus, showing, e.g.:
  - Proper nouns (NNP) often begin sentences: $P(\texttt{NNP}|\texttt{<s>}) \approx 0.28$
  - Modal verbs (MD) nearly always followed by bare verbs (VB).
  - Adjectives (JJ) are often followed by nouns (NN).

Table excerpted from J&M draft 3rd edition, Fig 8.5

Example: POS tagging

$t_{i-1} \backslash t_i$	NNP	MD	VB	JJ	NN	. . .
<s>	0.2767	0.0006	0.0031	0.0453	0.0449	. . .
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	. . .
MD	0.0008	0.0002	0.7968	0.0005	0.0008	. . .
VB	0.0322	0.0005	0.0050	0.0837	0.0615	. . .
JJ	0.0306	0.0004	0.0001	0.0733	0.4509	. . .
. . .	. . .	. . .	. . .	. . .	. . .	. . .

- This table is incomplete!
- In the full table, every row must sum up to 1 because it is a **distribution** over the next state (given previous).

Table excerpted from J&M draft 3rd edition, Fig 8.5

Example: POS tagging

$t_i \backslash w_i$	Janet	will	back	the	. . .
NNP	0.000032	0	0	0.000048	. . .
MD	0	0.308431	0	0	. . .
VB	0	0.000028	0.000672	0	. . .
DT	0	0	0	0.506099	. . .
. . .	. . .	. . .	. . .	. . .	. . .

- MLE probabilities from tagged WSJ corpus, showing, e.g.:
  - 0.0032% of proper nouns are *Janet*: $P(\texttt{Janet}|\texttt{NNP}) = 0.000032$
  - About half of determiners (DT) are *the*.
  - *the* can also be a proper noun. (Annotation error?)
- Again, in full table, rows would sum to 1.

From J&M draft 3rd edition, Fig 8.6

Example: POS tagging

What's the probability of this tagged sentence?

$$\text{This/DET is/VB a/DET simple/JJ sentence/NN}$$

- First, add begin- and end-of-sentence <s> and </s>. Then:

$$\begin{aligned} p(S,T) &= \prod_{i=1}^{n} P(t_i|t_{i-1})P(w_i|t_i) \\ &= P(\text{DET}|\text{<s>})P(\text{VB}|\text{DET})P(\text{DET}|\text{VB})P(\text{JJ}|\text{DET})P(\text{NN}|\text{JJ})P(\text{</s>}|\text{NN}) \\ &\quad \cdot P(\text{This}|\text{DET})P(\text{is}|\text{VB})P(\text{a}|\text{DET})P(\text{simple}|\text{JJ})P(\text{sentence}|\text{NN}) \end{aligned}$$

- Then, plug in the probabilities we estimated from our corpus.

Example: POS tagging

Let's now tag the newspaper headline:

deal talks fail

Note that each token here could be a noun (N) or a verb (V).
We'll use a toy HMM given as follows:

	to N	to V
from start	.8	.2
from N	.4	.6
from V	.8	.2

Transitions

	deal	fail	talks
N	.2	.05	.2
V	.3	.3	.3

Emissions

Example: POS tagging

	deal	talks	fail
N			
V			

	to N	to V
from start	.8	.2
from N	.4	.6
from V	.8	.2

Transitions

	deal	fail	talks
N	.2	.05	.2
V	.3	.3	.3

Emissions

$$\text{score}_1(s) = P(s)P(x_1|s) = \pi_s B_{x_1,s}$$

$$\text{score}_i(s) = \max_{y_{i-1}} P(s|y_{i-1})P(x_i|s)\text{score}_{i-1}(y_{i-1})$$

$$= \max_{y_{i-1}} A_{y_{i-1},s} B_{s,x_i} \text{score}_{i-1}(y_{i-1})$$

Example: POS tagging

	to N	to V
from start	.8	.2
from N	.4	.6
from V	.8	.2

Transitions

	deal	fail	talks
N	.2	.05	.2
V	.3	.3	.3

Emissions

	deal	talks	fail
N	.8x.2 = .16	← .16x.4x.2 = .0128 (since .16x.4 > .06x.8)	↙ .0288x.8x.05 = .001152 (since .0128x.4 < 0.0288x.8)
V	.2x.3 = .06	↖ .16x.6x.3 = .0288 (since .16x.6 > .06x.2)	↖ .0128x.6x.3 = .002304 (since .0128x.6 > 0.0288x.2)

Looking at the highest probability entry in the final column and chasing the backpointers, we see that the tagging N N V wins.

$$\text{score}_1(s) = P(s)P(x_1|s) = \pi_s B_{x_1,s}$$

$$\text{score}_i(s) = \max_{y_{i-1}} P(s|y_{i-1})P(x_i|s)\text{score}_{i-1}(y_{i-1})$$

$$= \max_{y_{i-1}} A_{y_{i-1},s} B_{s,x_i} \text{score}_{i-1}(y_{i-1})$$

Learning HMM parameters

- Assume we know the number of states in the HMM
- Two possible scenarios
  1. We are given a data set D = {<$\mathbf{x}_i$, $\mathbf{y}_i$>} of sequences labeled with states

And we have to learn the parameters of the HMM (π, A, B)

Supervised learning with complete data

  2. We are given only a collection of sequences D = {$\mathbf{x}_i$}

And we have to learn the parameters of the HMM (π, A, B)

Unsupervised learning, with incomplete data

EM algorithm

Supervised learning of HMM

- We are given a dataset D = {<$\mathbf{x}_i$, $\mathbf{y}_i$>}
  - Each $\mathbf{x}_i$ is a sequence of observations and $\mathbf{y}_i$ is a sequence of states that correspond to $\mathbf{x}_i$

  Goal: Learn initial, transition, emission distributions (π, A, B)

- How do we learn the parameters of the probability distribution?
  - The maximum likelihood principle

$$(\hat{\pi}, \hat{A}, \hat{B}) = \max_{\pi,A,B} P(D|\pi, A, B) = \max_{\pi,A,B} \prod_i P(\mathbf{x}_i, \mathbf{y}_i|\pi, A, B)$$

And we know how to write this in terms of the parameters of the HMM

Supervised learning details

$$(\hat{\pi}, \hat{A}, \hat{B}) = \max_{\pi,A,B} P(D|\pi, A, B) = \max_{\pi,A,B} \prod_i P(\mathbf{x}_i, \mathbf{y}_i|\pi, A, B)$$

π, A, B can be estimated separately just by counting

- Makes learning simple and fast

[Exercise: Derive the following using derivatives of the log likelihood. Requires Lagrangian multipliers.]

$$\pi_s = \frac{\text{count}(\text{start} \rightarrow s)}{n}$$

Number of instances where the first state is s

Number of examples

Initial probabilities

$$A_{s',s} = \frac{\text{count}\,(s \rightarrow s')}{\text{count}\,(s)}$$

Transition probabilities

$$B_{s,x} = \frac{\text{count}\begin{pmatrix} s \\ \downarrow \\ x \end{pmatrix}}{\text{count}\,(s)}$$

Emission probabilities

Priors and smoothing

- Maximum likelihood estimation works best with lots of annotated data
  - Never the case
- Priors inject information about the probability distributions
  - Dirichlet priors for multinomial distributions
- Effectively additive smoothing
  - Add small constants to the counts

Conditional Models and Local Classifiers

HMM

- The independence assumption

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

HMM redux

- The independence assumption

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

- Training via maximum likelihood

$$(\hat{\pi}, \hat{A}, \hat{B}) = \max_{\pi, A, B} P(D|\pi, A, B) = \max_{\pi, A, B} \prod_i P(\mathbf{x}_i, \mathbf{y}_i|\pi, A, B)$$

HMM redux

- The independence assumption

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

- Training via maximum likelihood

$$(\hat{\pi}, \hat{A}, \hat{B}) = \max_{\pi,A,B} P(D|\pi, A, B) = \max_{\pi,A,B} \prod_i P(\mathbf{x}_i, \mathbf{y}_i|\pi, A, B)$$

We are optimizing joint likelihood of the input and the output for training

HMM redux

- The independence assumption

Probability of input given the prediction!

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

- Training via maximum likelihood

$$(\hat{\pi}, \hat{A}, \hat{B}) = \max_{\pi,A,B} P(D|\pi, A, B) = \max_{\pi,A,B} \prod_i P(\mathbf{x}_i, \mathbf{y}_i|\pi, A, B)$$

We are optimizing joint likelihood of the input and the output for training

HMM redux

- The independence assumption

Probability of input given the prediction!

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

- Training via maximum likelihood

$$(\hat{\pi}, \hat{A}, \hat{B}) = \max_{\pi,A,B} P(D|\pi, A, B) = \max_{\pi,A,B} \prod_i P(\mathbf{x}_i, \mathbf{y}_i|\pi, A, B)$$

We are optimizing joint likelihood of the input and the output for training

At prediction time, we only care about the probability of output given the input. Why not directly optimize this *conditional likelihood* instead?

Modeling next-state directly

- Instead of modeling the joint distribution P(**x**, **y**) only focus on P(**y**|**x**)
  - Which is what we care about eventually anyway

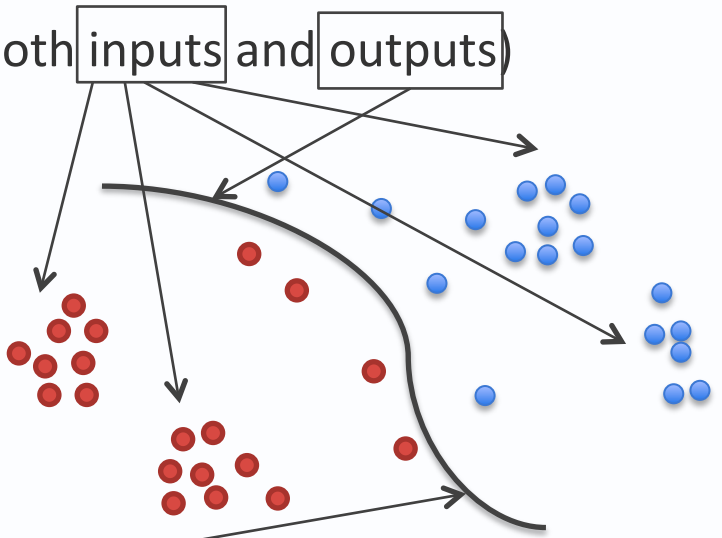
- For sequences, different formulations
  - Maximum Entropy Markov Model [McCallum, et al 2000]
  - Projection-based Markov Model [Punyakanok and Roth, 2001]

  (other names: discriminative/conditional markov model, …)

Generative vs Discriminative models

- Generative models
  - learn P(x, y)
  - Characterize how the data is generated (both inputs and outputs)
  - Eg: Naïve Bayes, Hidden Markov Model

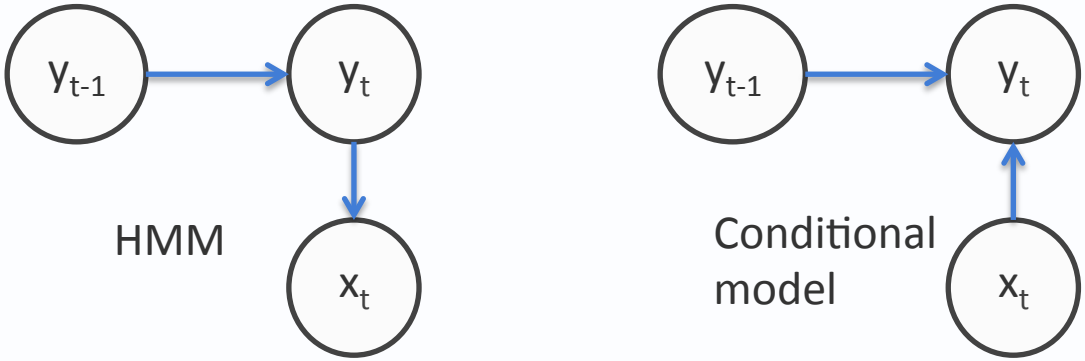
A generative model tries to characterize the distribution of the inputs, a discriminative model doesn't care



- Discriminative models
  - learn P(y | x)
  - Directly characterizes the decision boundary only
  - Eg: Logistic Regression, Conditional models (several names)

Another independence assumption

$$P(y_i|y_{i-1}, y_{i-2}, \cdots, x_i, x_{i-1}, \cdots) = P(y_i|y_{i-1}, x_i)$$

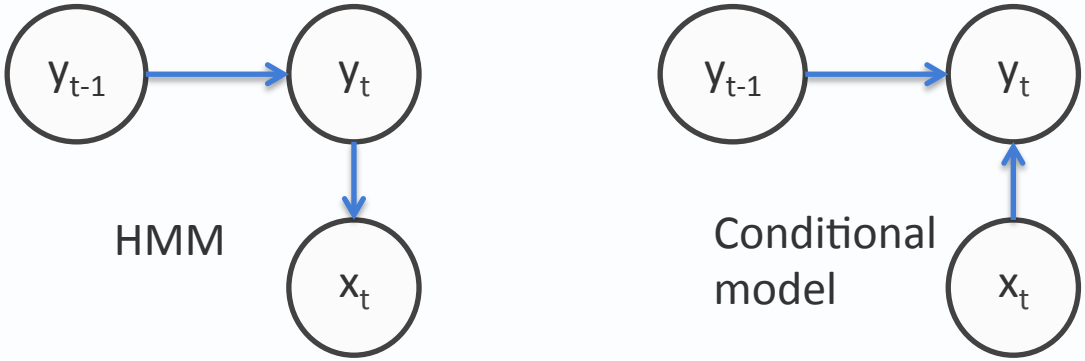


This assumption lets us write the conditional probability of the output as

$$P(\mathbf{y}|\mathbf{x}) = \prod_i P(y_i|y_{i-1}, x_i)$$

Another independence assumption

$$P(y_i|y_{i-1}, y_{i-2}, \cdots, x_i, x_{i-1}, \cdots) = P(y_i|y_{i-1}, x_i)$$



This assumption lets us write the conditional probability of the output as

$$P(\mathbf{y}|\mathbf{x}) = \prod_i \boxed{P(y_i|y_{i-1}, x_i)}$$

We need to learn this function

Modeling $P(y_i \mid y_{i-1}, x_i)$

- Different approaches possible
  1. Train a *maximum entropy* / log-linear classifier

$$P(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \frac{e^{\mathbf{w}^T \phi(\mathbf{x},\mathbf{y})}}{\sum_{y'} e^{\mathbf{w}^T \phi(\mathbf{x},\mathbf{y})}}$$

Interpretation: Score for label, converted to a well-formed probability distribution by exponentiating + normalizing

  2. Ignore the fact that we are predicting a probability, we only care about maximizing some *score.* Train any classifier, using say the perceptron algorithm

- For both cases*:*
  - Use rich features that depend on input and previous state
  - We can increase the dependency to arbitrary neighboring x_i's
    - Eg. Neighboring words influence this words POS tag

Modeling ~~$P(y_i \mid y_{i-1}, x_i)$~~ $P(y_i \mid y_{i-1}, \mathbf{x})$

- Different approaches possible
  1. Train a *maximum entropy* / log-linear classifier

$$P(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \frac{e^{\mathbf{w}^T \phi(\mathbf{x},\mathbf{y})}}{\sum_{y'} e^{\mathbf{w}^T \phi(\mathbf{x},\mathbf{y})}}$$

Interpretation: Score for label, converted to a well-formed probability distribution by exponentiating + normalizing

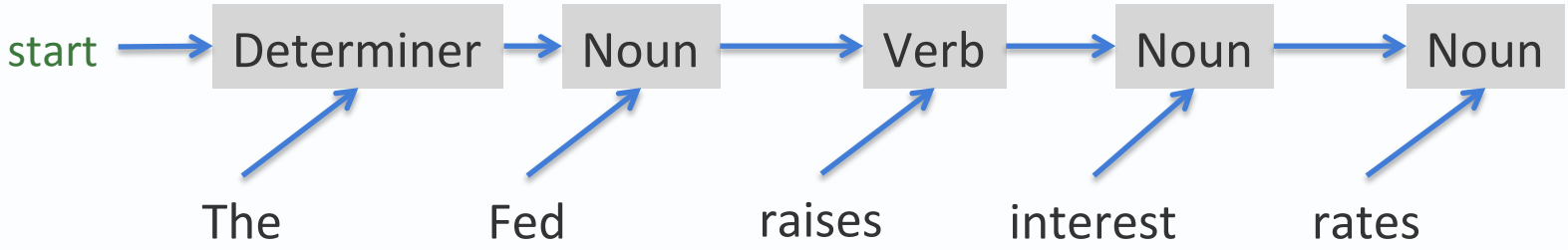
  2. Ignore the fact that we are predicting a probability, we only care about maximizing some *score.* Train any classifier, using say the perceptron algorithm

- For both cases*:*
  - Use rich features that depend on input and previous state
  - We can increase the dependency to arbitrary neighboring x_i's
    - Eg. Neighboring words influence this words POS tag

Maximum Entropy Markov Model

Goal: Compute P(**y** | **x**)

$$P(y_i|y_{i-1}, \mathbf{x}) \propto \exp\left(\mathbf{w}^T \phi(\mathbf{x}, i, y_i, y_{i-1})\right)$$

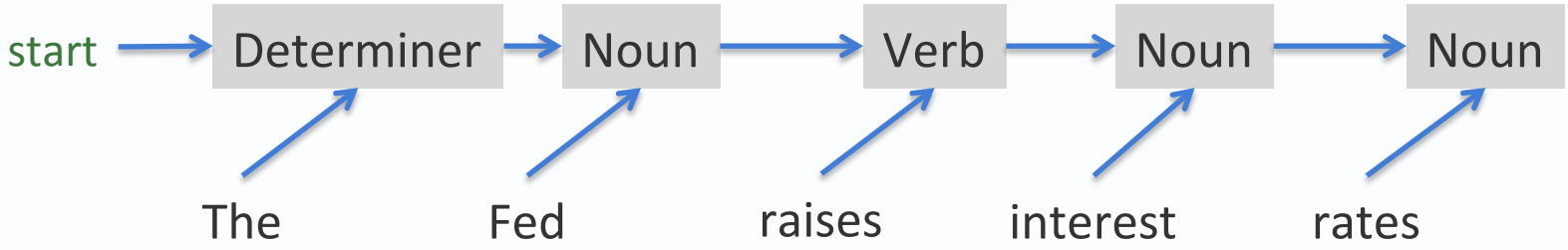


The prediction task: Using the entire input and the current label, predict the next label

Maximum Entropy Markov Model

Goal: Compute P(**y** | **x**)

$$P(y_i|y_{i-1}, \mathbf{x}) \propto \exp\left(\mathbf{w}^T \phi(\mathbf{x}, i, y_i, y_{i-1})\right)$$



word

Caps

-es

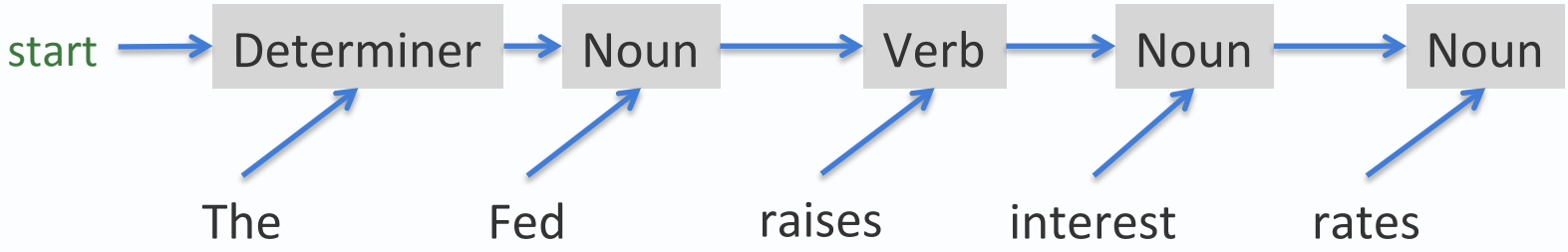
Previous

To model the probability, first, we need to define features for the current classification problem

Maximum Entropy Markov Model

Goal: Compute P(**y** | **x**)

$$P(y_i|y_{i-1}, \mathbf{x}) \propto \exp\left(\mathbf{w}^T \phi(\mathbf{x}, i, y_i, y_{i-1})\right)$$



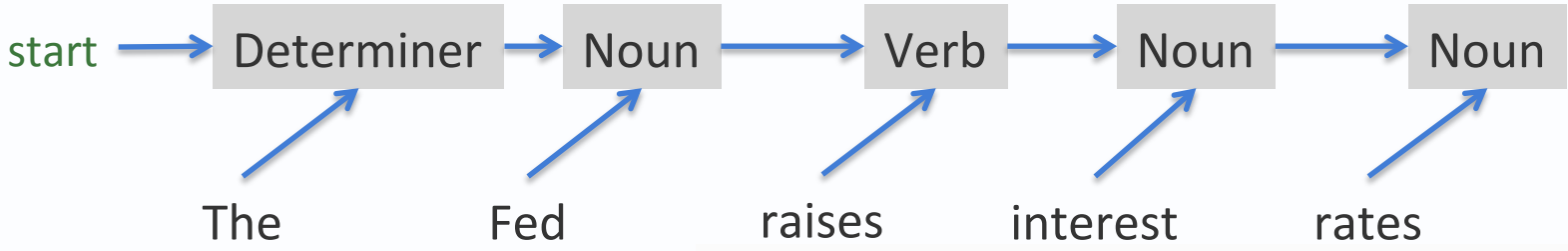
word	The
Caps	Y
-es	N
Previous	start

ϕ(x, 0, start, y_0)

Maximum Entropy Markov Model

Goal: Compute P(**y** | **x**)

$$P(y_i|y_{i-1}, \mathbf{x}) \propto \exp\left(\mathbf{w}^T \phi(\mathbf{x}, i, y_i, y_{i-1})\right)$$



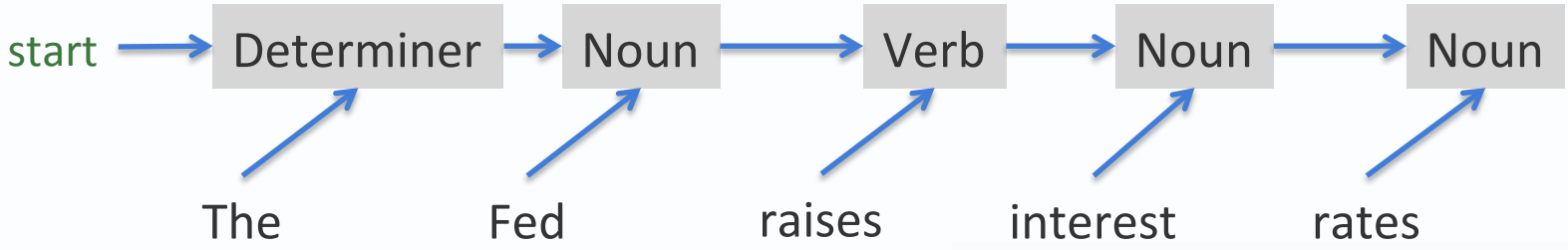
word	The	Fed
Caps	Y	Y
-es	N	N
Previous	start	Determiner

ϕ(x, 0, start, y_0) ϕ(x, 1, y_0, y_1)

Maximum Entropy Markov Model

Goal: Compute P(**y** | **x**)

$$P(y_i|y_{i-1}, \mathbf{x}) \propto \exp\left(\mathbf{w}^T \phi(\mathbf{x}, i, y_i, y_{i-1})\right)$$



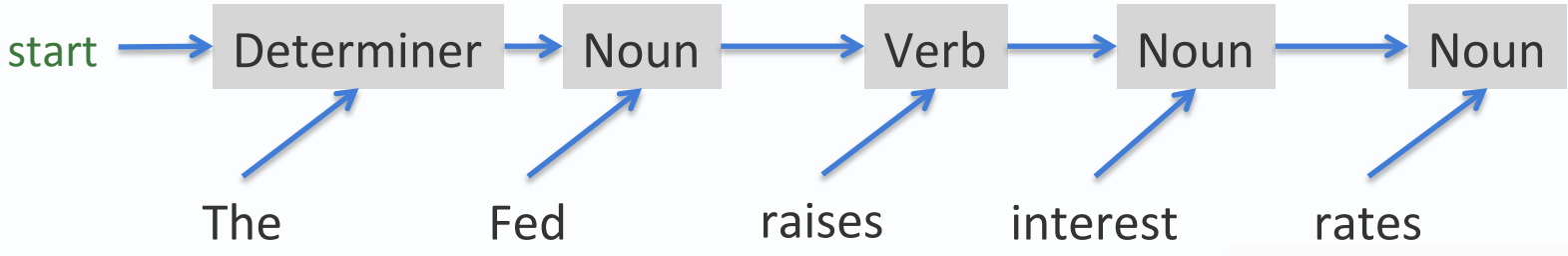
word	The	Fed	raises
Caps	Y	Y	N
-es	N	N	Y
Previous	start	Determiner	Noun

ϕ(x, 0, start, y_0) ϕ(x, 1, y_0, y_1) ϕ(x, 2, y_1, y_2)

Maximum Entropy Markov Model

Goal: Compute P(**y** | **x**)

$$P(y_i|y_{i-1}, \mathbf{x}) \propto \exp\left(\mathbf{w}^T \phi(\mathbf{x}, i, y_i, y_{i-1})\right)$$



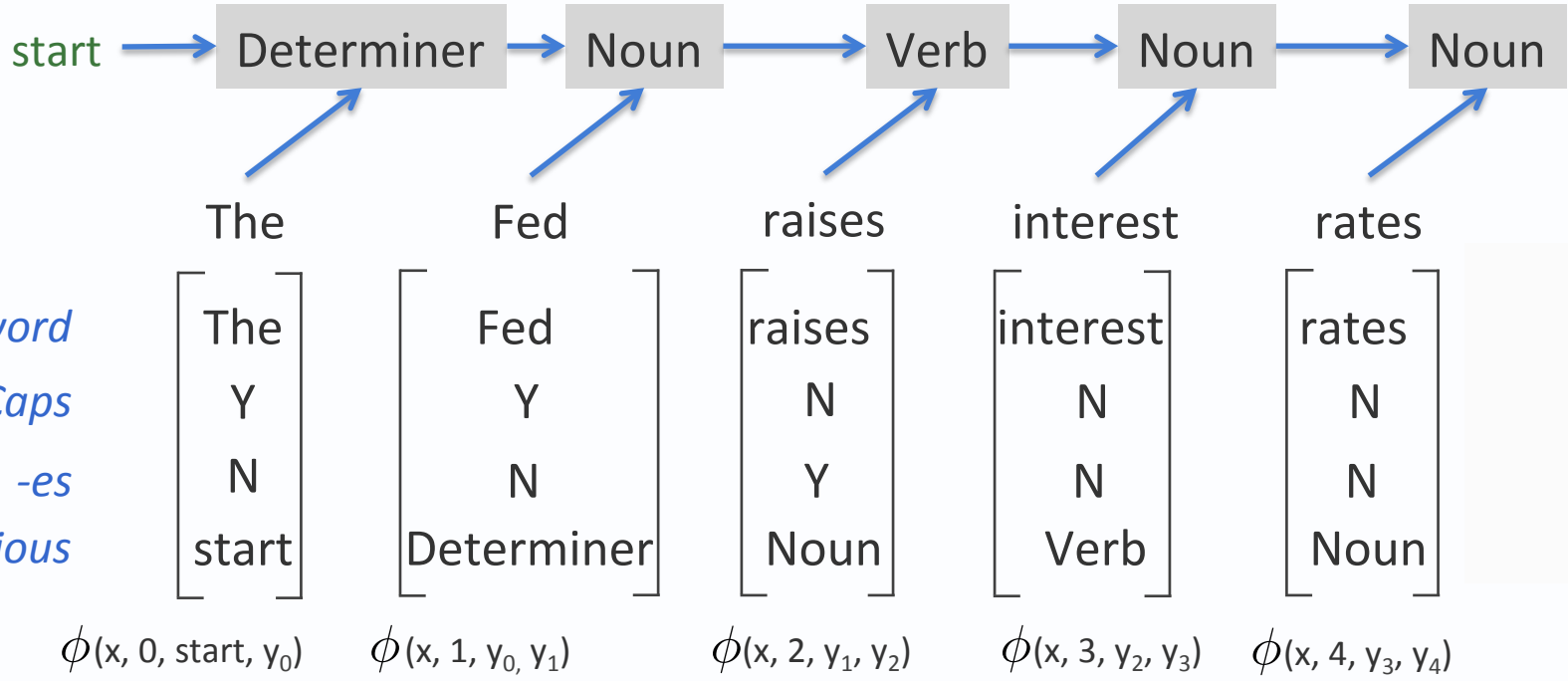
word	The	Fed	raises	interest
Caps	Y	Y	N	N
-es	N	N	Y	N
Previous	start	Determiner	Noun	Verb

ϕ(x, 0, start, y_0) ϕ(x, 1, y_0, y_1) ϕ(x, 2, y_1, y_2) ϕ(x, 3, y_2, y_3)

Maximum Entropy Markov Model

Goal: Compute P(**y** | **x**)

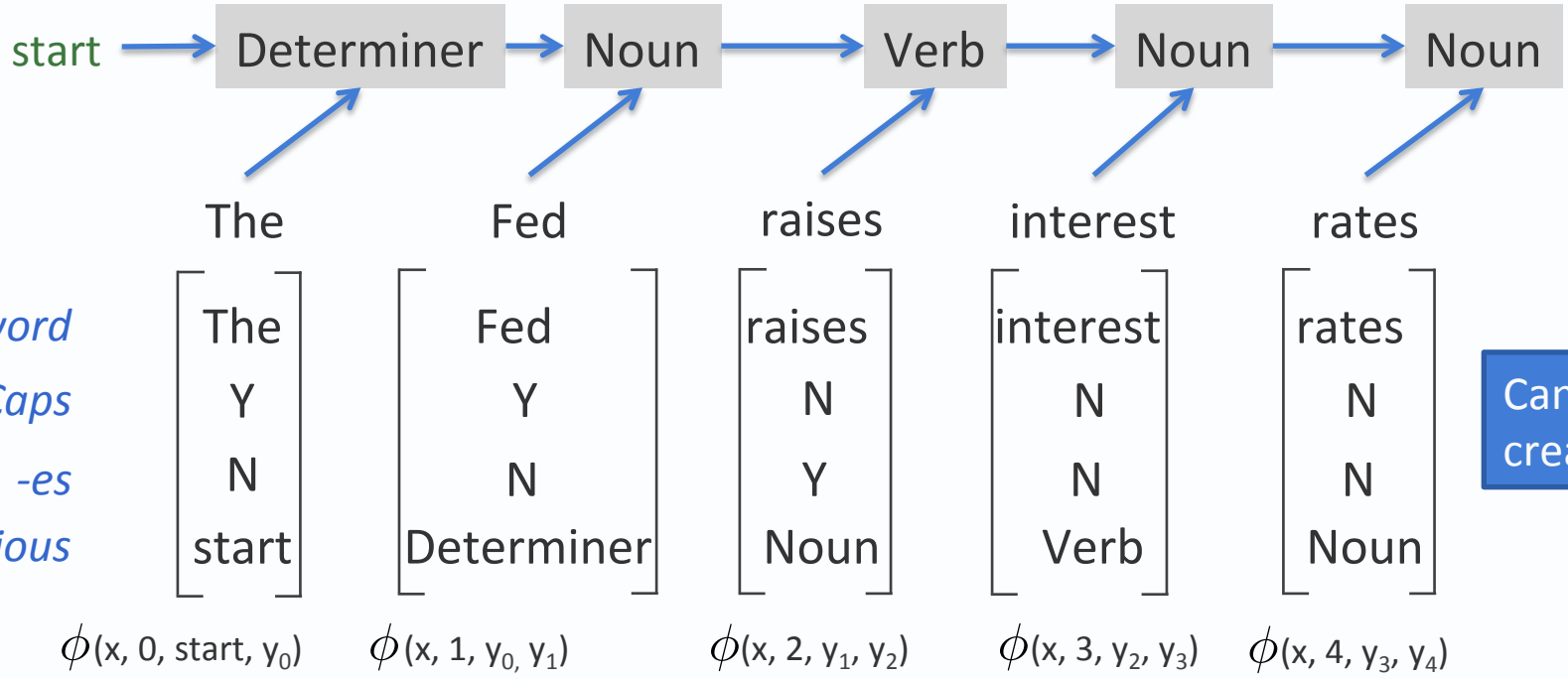
$$P(y_i|y_{i-1}, \mathbf{x}) \propto \exp\left(\mathbf{w}^T \phi(\mathbf{x}, i, y_i, y_{i-1})\right)$$



Maximum Entropy Markov Model

Goal: Compute P(**y** | **x**)

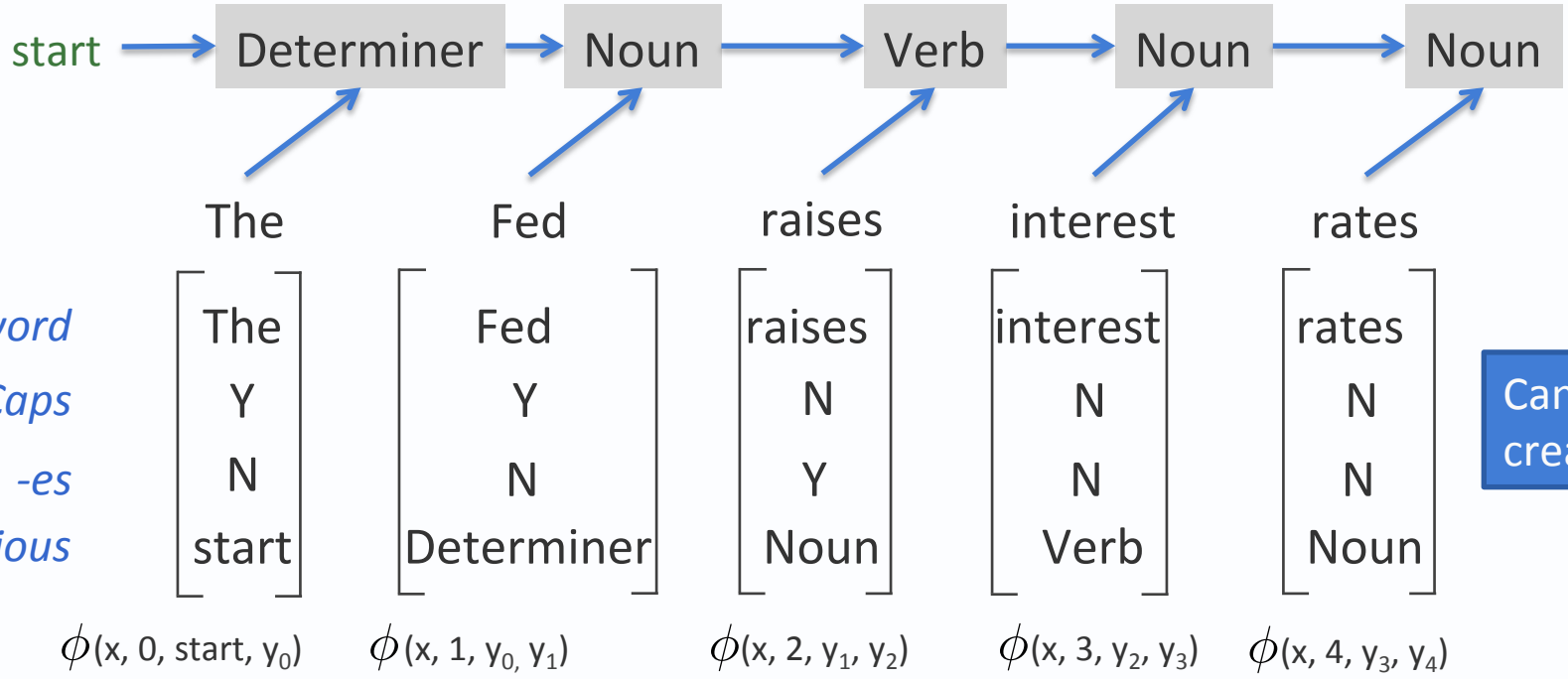
$$P(y_i|y_{i-1}, \mathbf{x}) \propto \exp\left(\mathbf{w}^T \phi(\mathbf{x}, i, y_i, y_{i-1})\right)$$



Maximum Entropy Markov Model

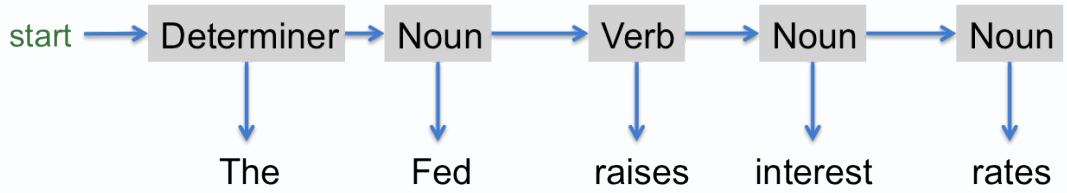
Goal: Compute P(**y** | **x**)

$$P(y_i|y_{i-1}, \mathbf{x}) \propto \exp\left(\mathbf{w}^T \phi(\mathbf{x}, i, y_i, y_{i-1})\right)$$



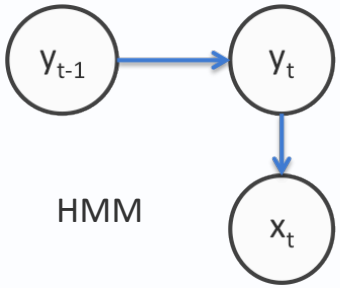
Can get very creative here

Compare to HMM: Only depends on the word and the previous tag



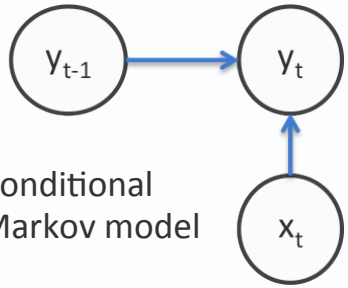
Using MEMM

- Training
  - Next-state predictor **locally** as maximum likelihood
    - Similar to any maximum entropy classifier
- Prediction/decoding
  - Modify the Viterbi algorithm for the new independence assumptions



HMM

$$\text{score}_i(s) = \max_{y_{i-1}} P(s|y_{i-1})P(x_i|s)\text{score}_{i-1}(y_{i-1})$$



Conditional Markov model

$$\text{score}_i(s) = \max_{y_{i-1}} P(s|y_{i-1}, x_i)\text{score}_{i-1}(y_{i-1})$$

Generalization: Any multiclass classifier

- **Viterbi decoding**: we only need a score for each decision
  - So far, probabilistic classifiers

- In general, use any learning algorithm to get a score for the label y_i given y_{i-1} and **x**
  - Multiclass versions of perceptron, SVM
  - Just like MEMM, these allow arbitrary features to be defined

Comparison to HMM

What we gain

1. Rich feature representation for inputs
   - Helps generalize better by thinking about properties of the input tokens rather than the entire tokens
   - Eg: If a word ends with –es, it might be a present tense verb (such as raises). Could be a feature; HMM cannot capture this

2. Discriminative predictor
   - Model P(**y** | **x**) rather than P(**y**, x)
   - *Joint* vs *conditional*

Global Models

…local classifiers! Label bias problem

Let's look at the independence assumption

$$P(y_i|y_{i-1}, y_{i-2}, \cdots, x_i, x_{i-1}, \cdots) = P(y_i|y_{i-1}, x_i)$$

"Next-state" classifiers are locally normalized

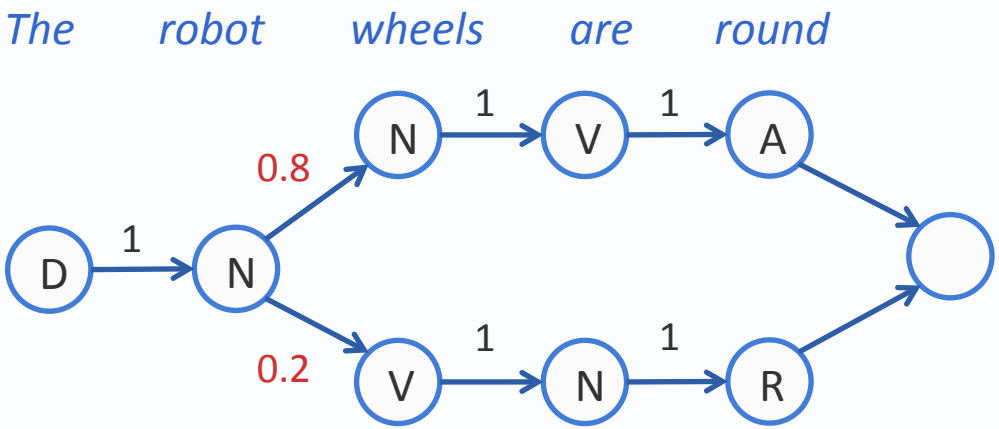
…local classifiers! Label bias problem

Let's look at the independence assumption

$$P(y_i|y_{i-1}, y_{i-2}, \cdots, x_i, x_{i-1}, \cdots) = P(y_i|y_{i-1}, x_i)$$

"Next-state" classifiers are locally normalized

Eg: Part-of-speech tagging the sentence



Suppose these are the only state transitions allowed

Example based on [Wallach 2002]

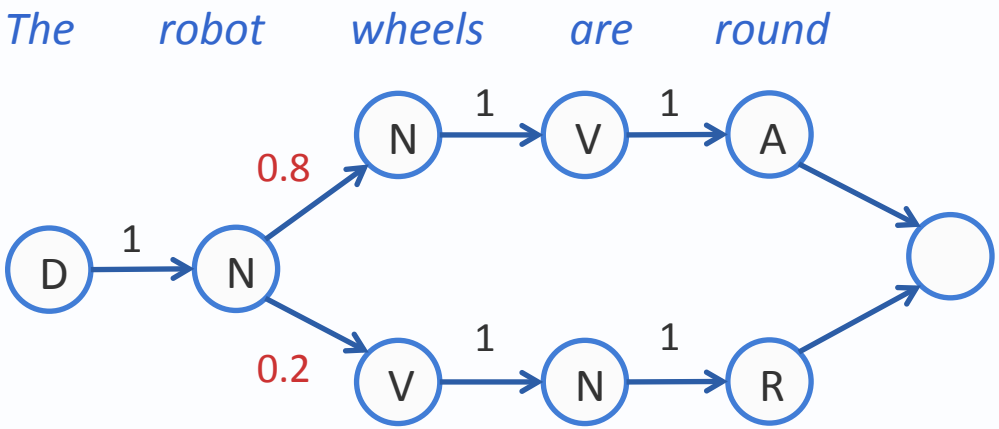
…local classifiers! Label bias problem

Let's look at the independence assumption

$$P(y_i|y_{i-1}, y_{i-2}, \cdots, x_i, x_{i-1}, \cdots) = P(y_i|y_{i-1}, x_i)$$

"Next-state" classifiers are locally normalized

Eg: Part-of-speech tagging the sentence



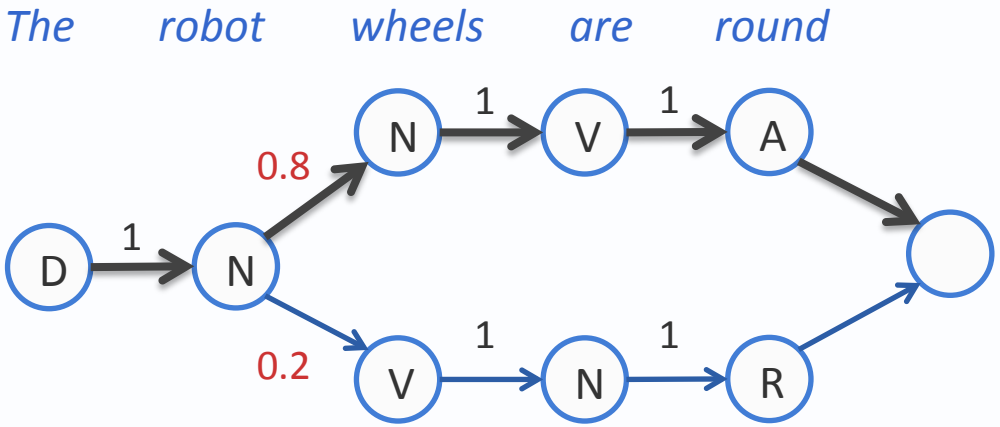
Option 1: P(D | The) .
P(N | D, robot) .
P(N | N, wheels) .
P(V | N, are) .
P(A | V, round)

Option 2: P(D | The) .
P(N | D, robot) .
P(V | N, wheels) .
P(N | V, are) .
P(R| N, round)

Suppose these are the only state transitions allowed

Example based on [Wallach 2002]

…local classifiers ! Label bias problem



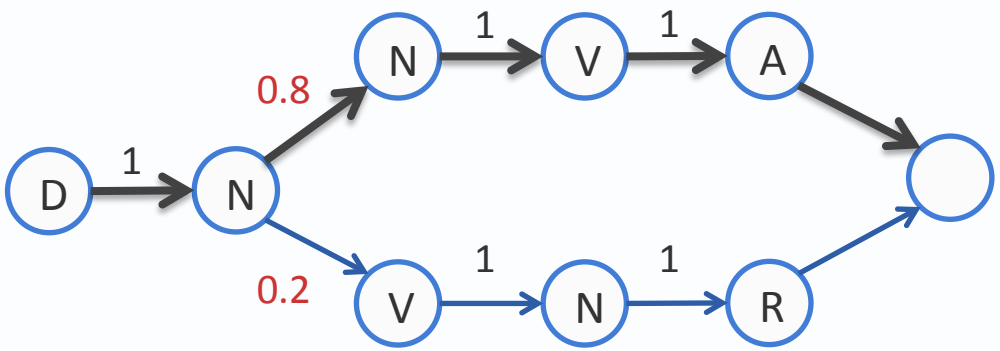
Suppose these are the only state transitions allowed

Option 1: P(D | The) .
P(N | D, robot) .
P(N | N, wheels) .
P(V | N, are) .
P(A | V, round)

Option 2: P(D | The) .
P(N | D, robot) .
P(V | N, wheels) .
P(N | V, are) .
P(R| N, round)

…local classifiers ! Label bias problem

The *robot* *wheels* *are* *round*



Option 1: P(D | The) .
P(N | D, robot) .
P(N | N, wheels) .
~~P(V | N, are)~~ . P(V | N, Fred) .
P(A | V, round)

Option 2: P(D | The) .
P(N | D, robot) .
P(V | N, wheels) .
~~P(N | V, are)~~ . P(N | V, Fred) .
P(R| N, round)

Suppose these are the only state transitions allowed

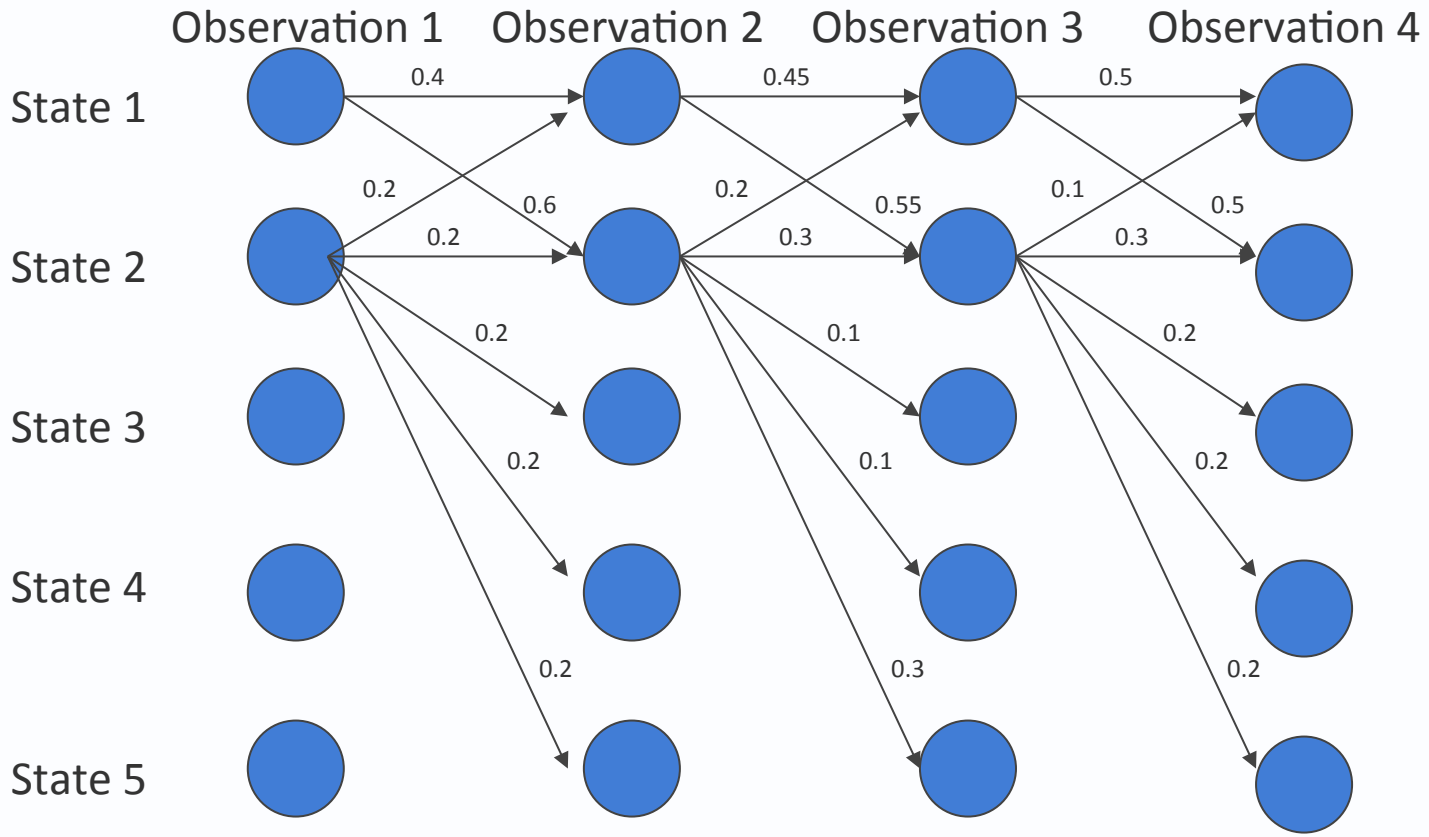
The *robot* *wheels* *Fred* *round*

The path scores are the same

Even if the word Fred is never observed as a verb in the data, it will be predicted as one

The input *Fred* does not influence the output at all

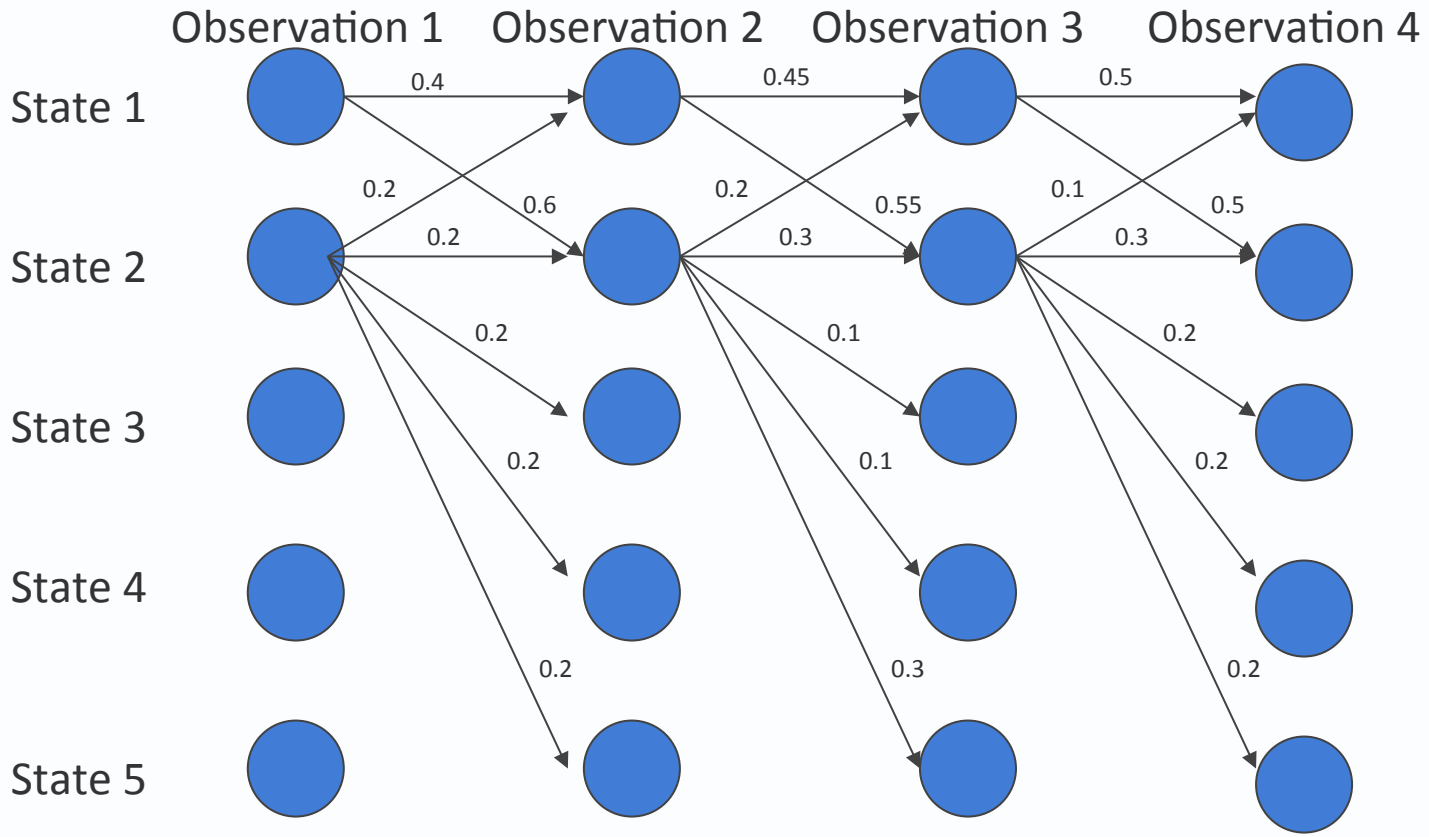
MEMM: Label bias problem



What the local transition probabilities say:

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

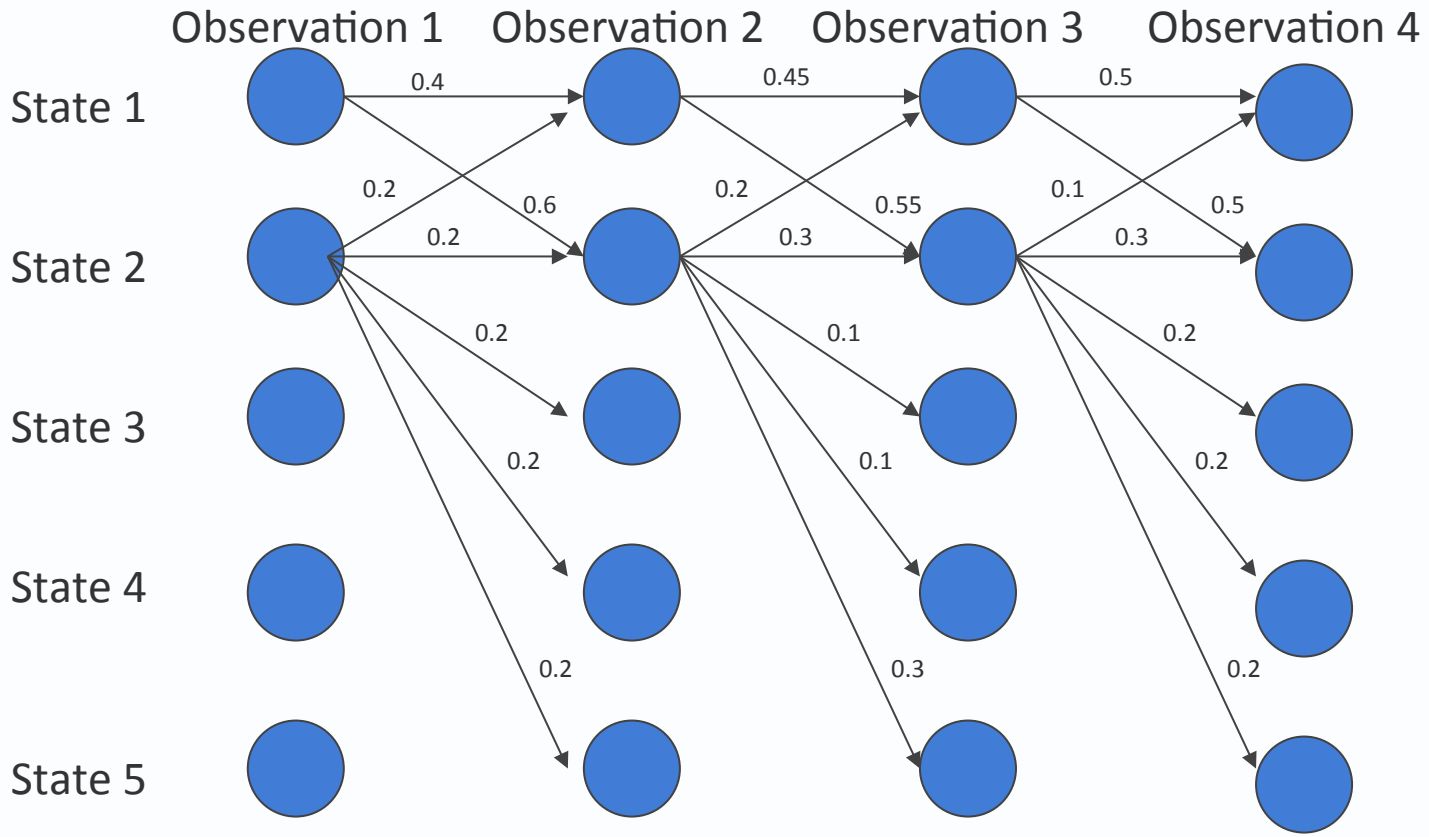
MEMM: Label bias problem



Probability of path 1-> 1-> 1-> 1:

- 0.4 x 0.45 x 0.5 = 0.09

MEMM: Label bias problem



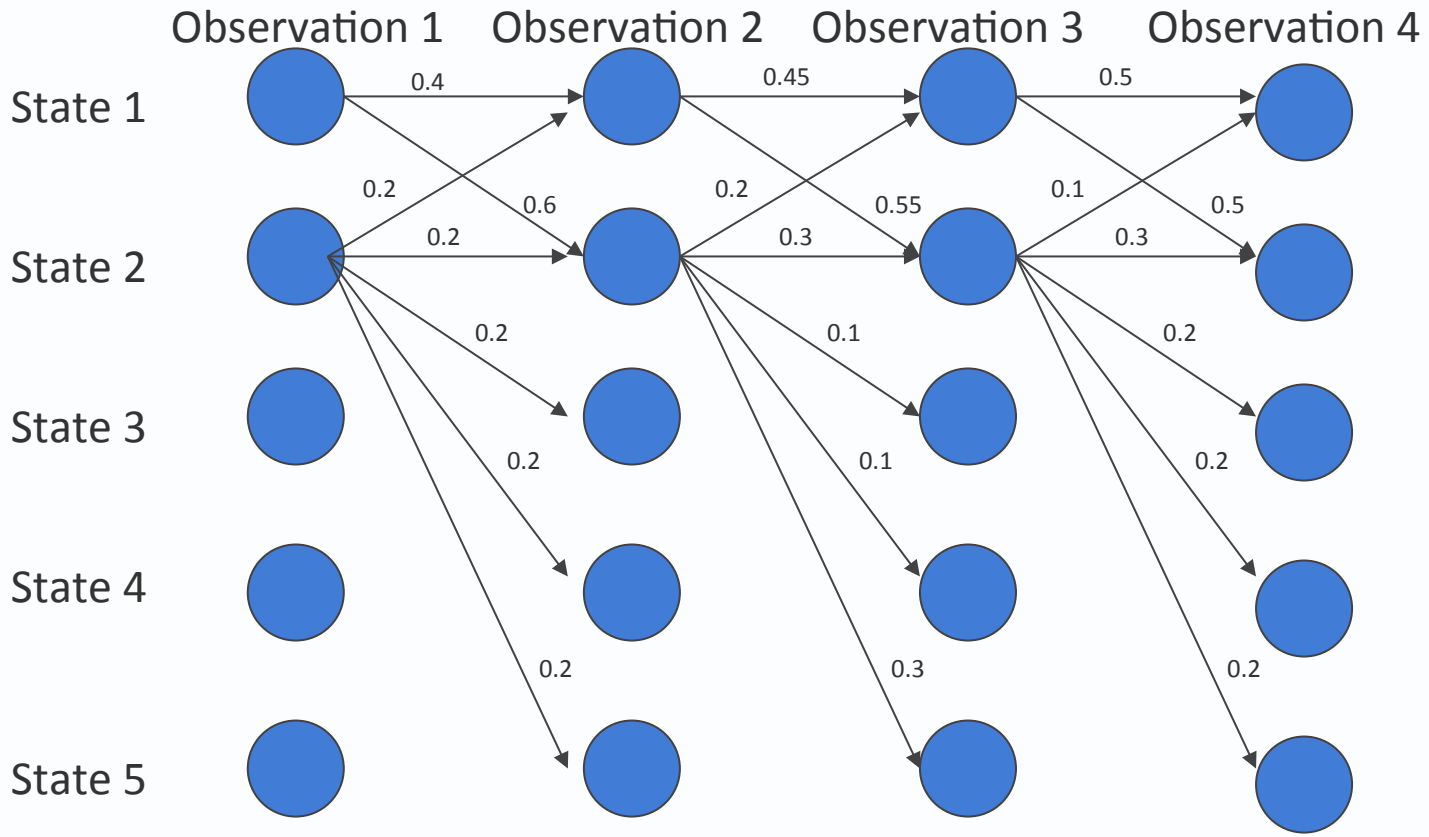
Probability of path 2->2->2->2 :

- 0.2 X 0.3 X 0.3 = 0.018

Other paths:

1-> 1-> 1-> 1: 0.09

MEMM: Label bias problem

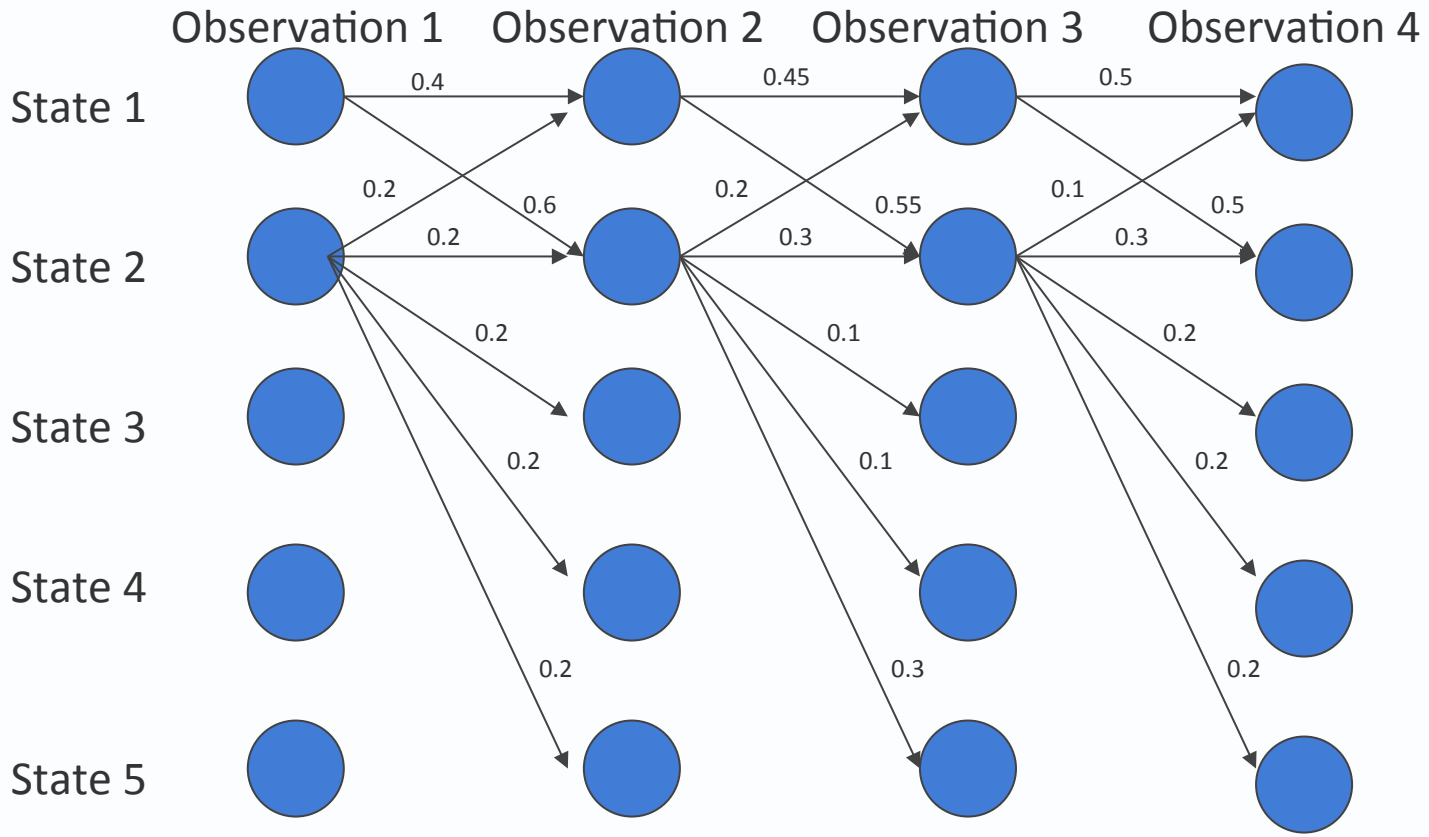


Probability of path 1->2->1->2:

- 0.6 X 0.2 X 0.5 = 0.06

Other paths:
1->1->1->1: 0.09
2->2->2->2: 0.018

MEMM: Label bias problem



Probability of path 1->1->2->2:

- 0.4 X 0.55 X 0.3 = 0.066

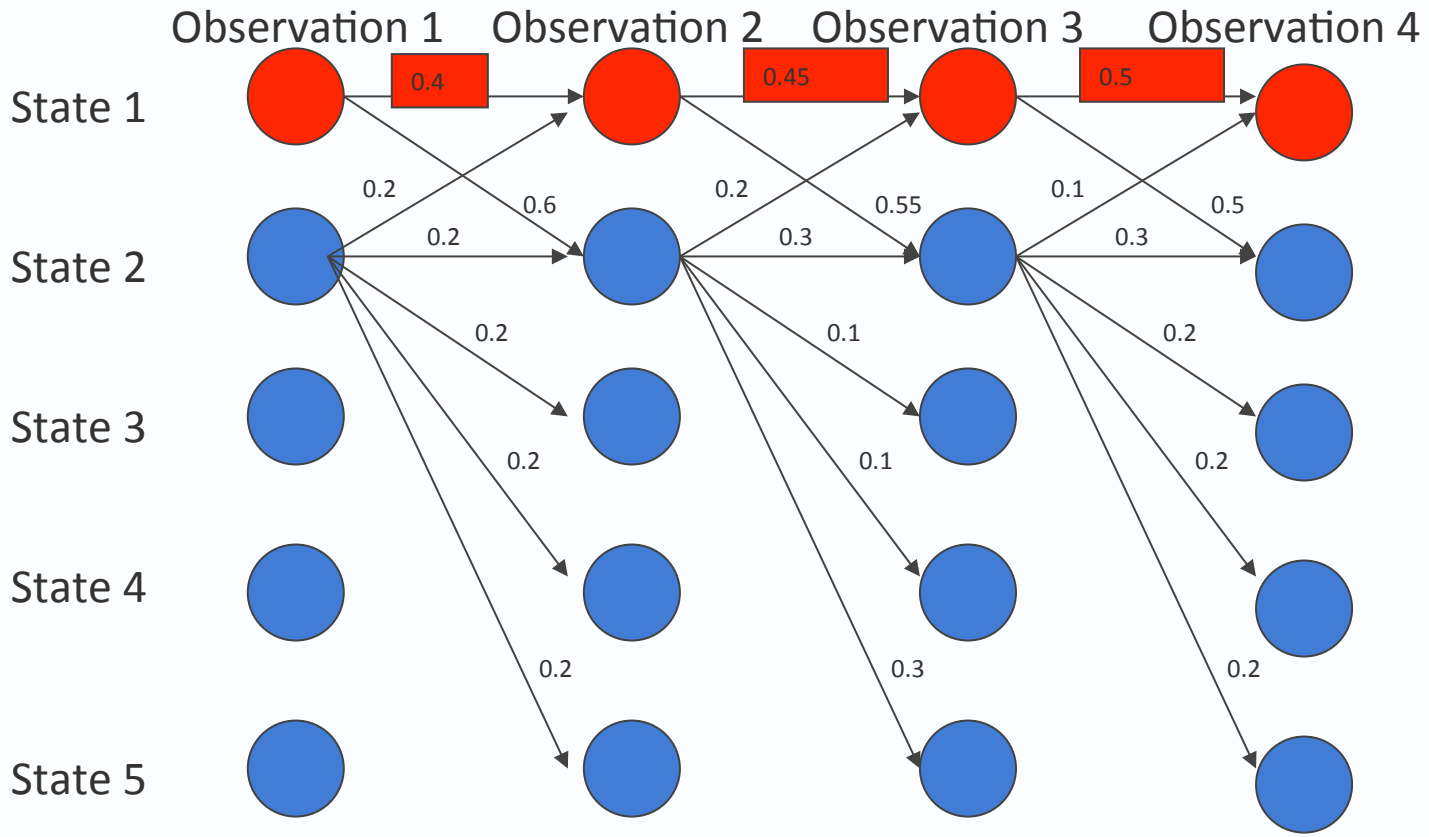
Other paths:

1->1->1->1: 0.09

2->2->2->2: 0.018

1->2->1->2: 0.06

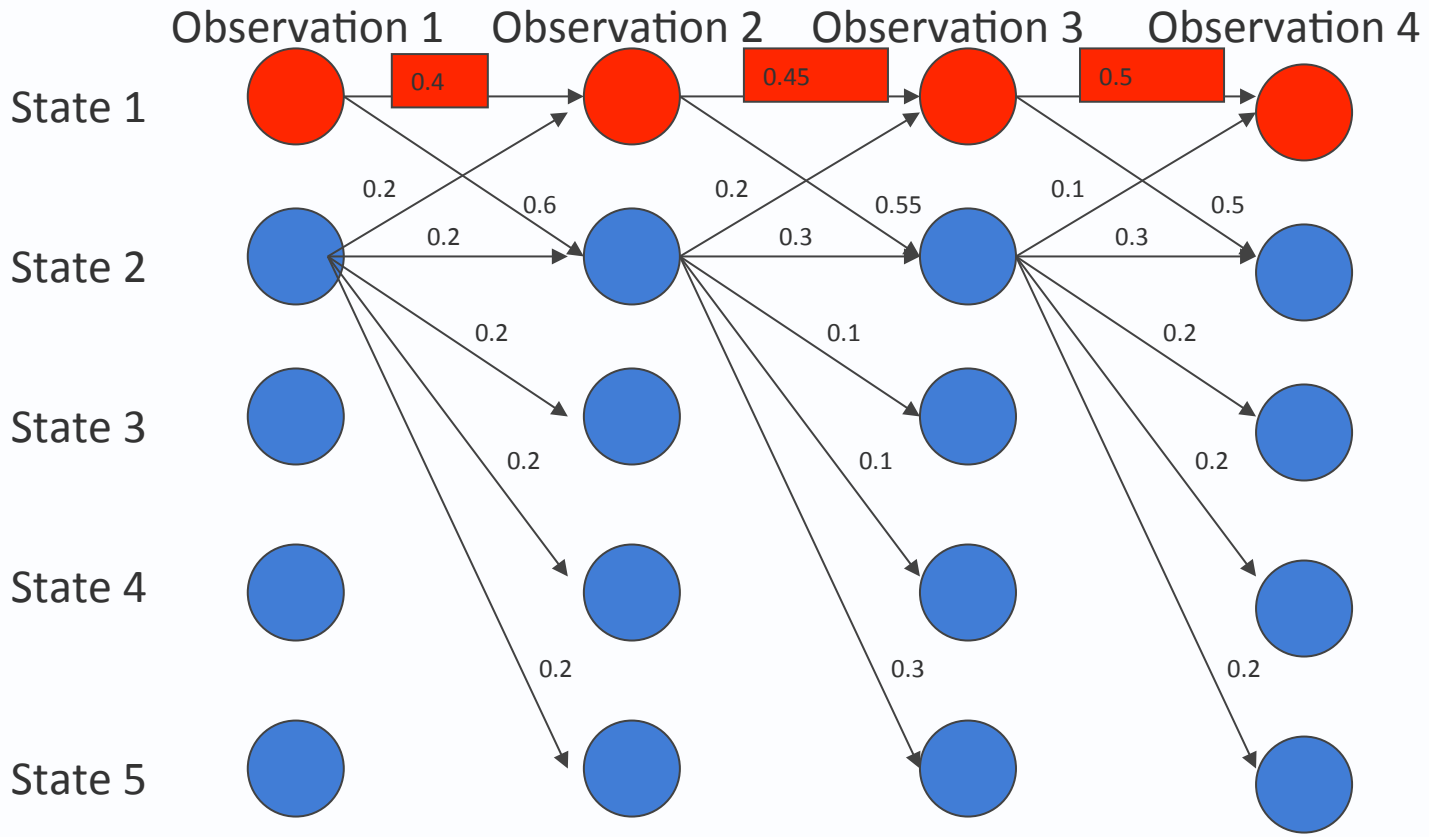
MEMM: Label bias problem



Most Likely Path: 1-> 1-> 1-> 1

- Although locally it seems state 1 wants to go to state 2 and state 2 wants to remain in state 2.
- **why?**

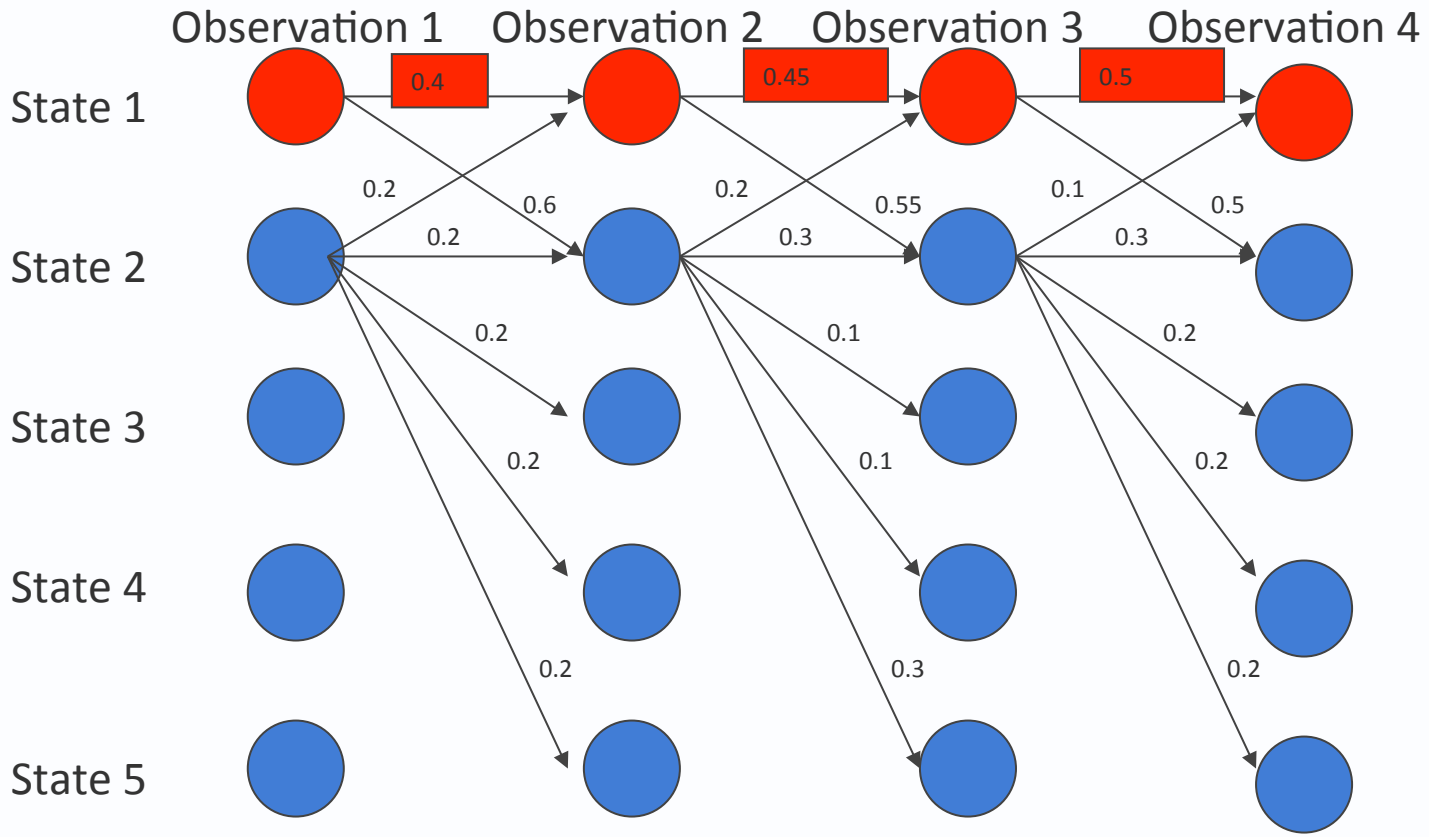
MEMM: Label bias problem



Most Likely Path: 1-> 1-> 1-> 1

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

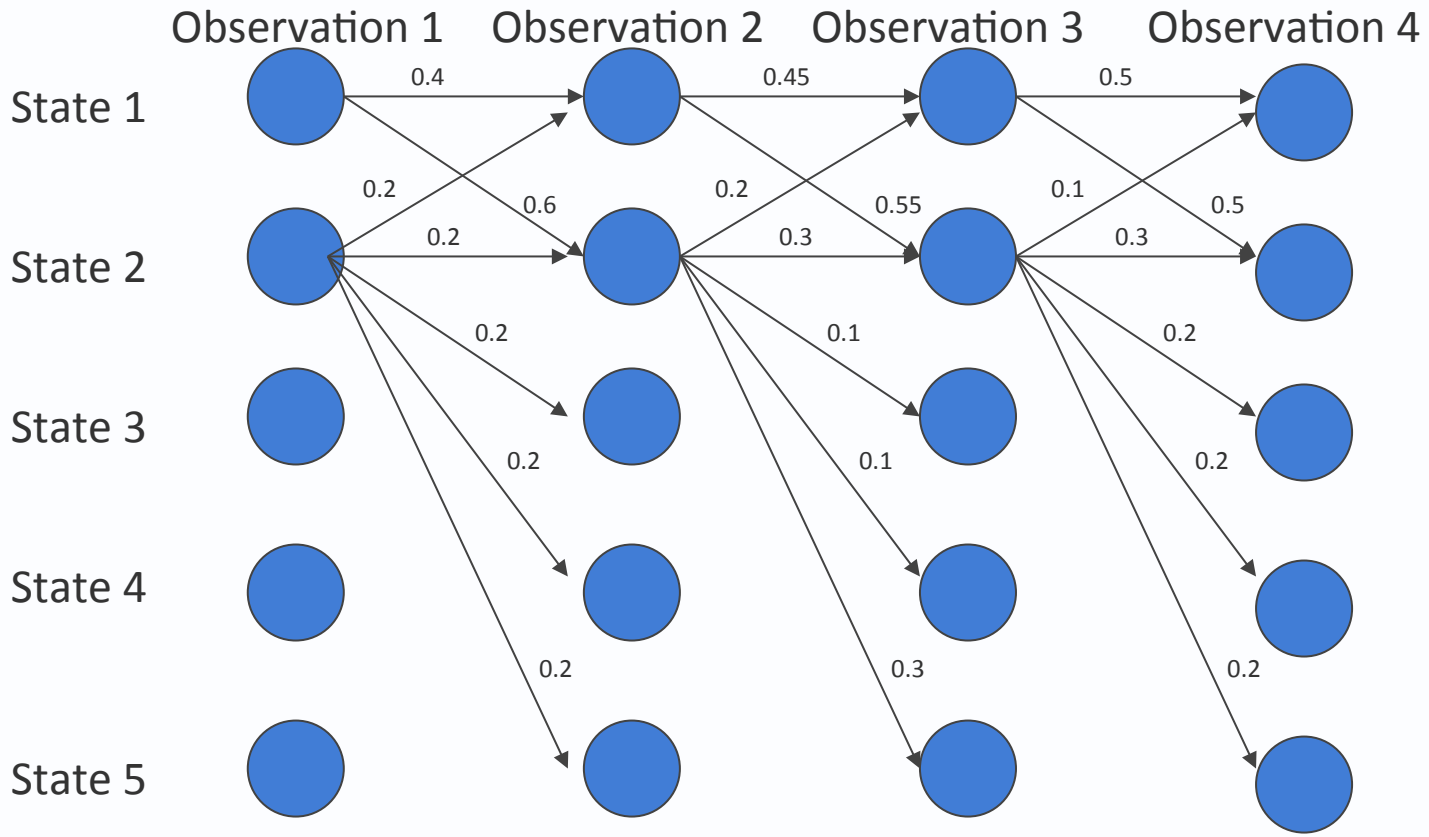
MEMM: Label bias problem



Label bias problem in MEMM:

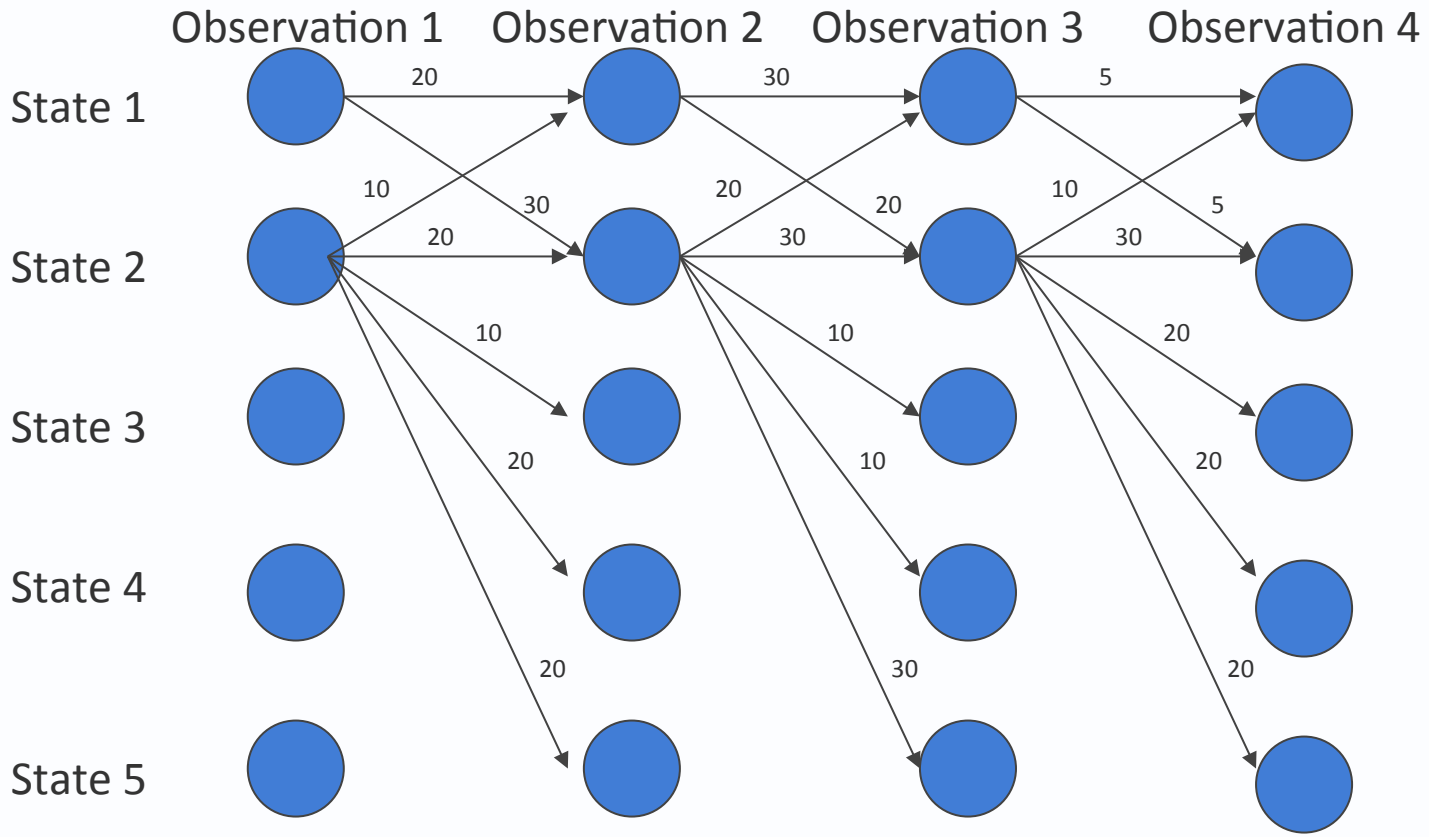
- Preference of states with lower number of transitions over others

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!

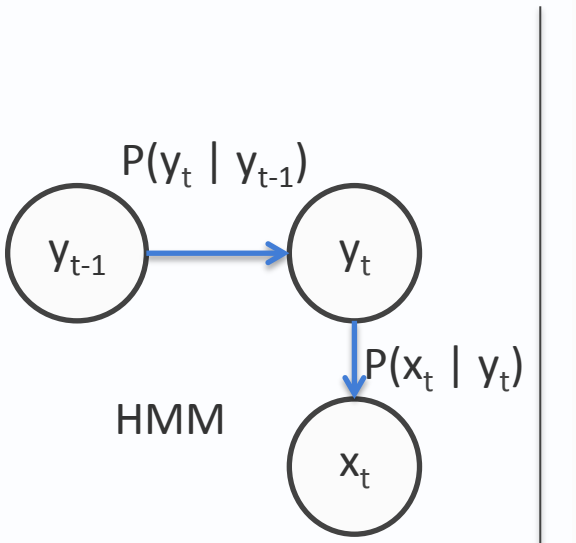
Label Bias

- States with a single outgoing transition effectively ignore their input
  - States with lower-entropy next states are less influenced by observations
- Why?
  - Because each the next-state classifiers are locally normalized
  - If a state has fewer next states, each of those will get a higher probability mass
    - …and hence preferred
- Side note: Surprisingly doesn't affect some tasks
  - Eg: part-of-speech tagging

Global models

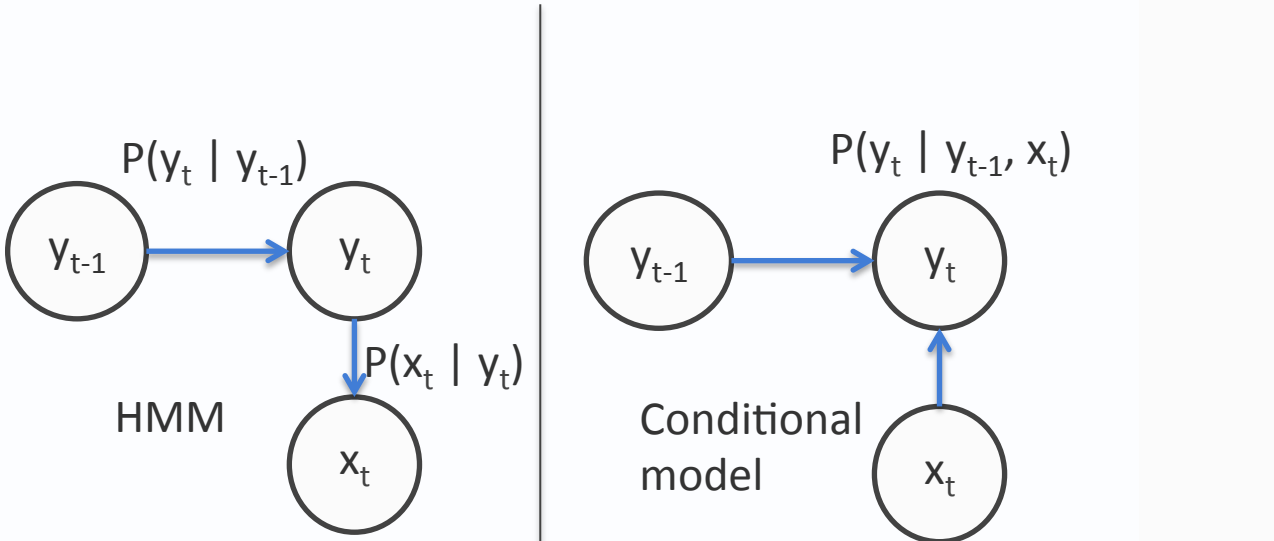
- Train the predictor globally
  - Instead of training local decisions independently
- *Normalize globally*
  - Make each edge in the model undirected
  - Not associated with a probability, but just a “score”
- Conditional Random Fields (CRF)

HMM vs. A local model vs. A global model



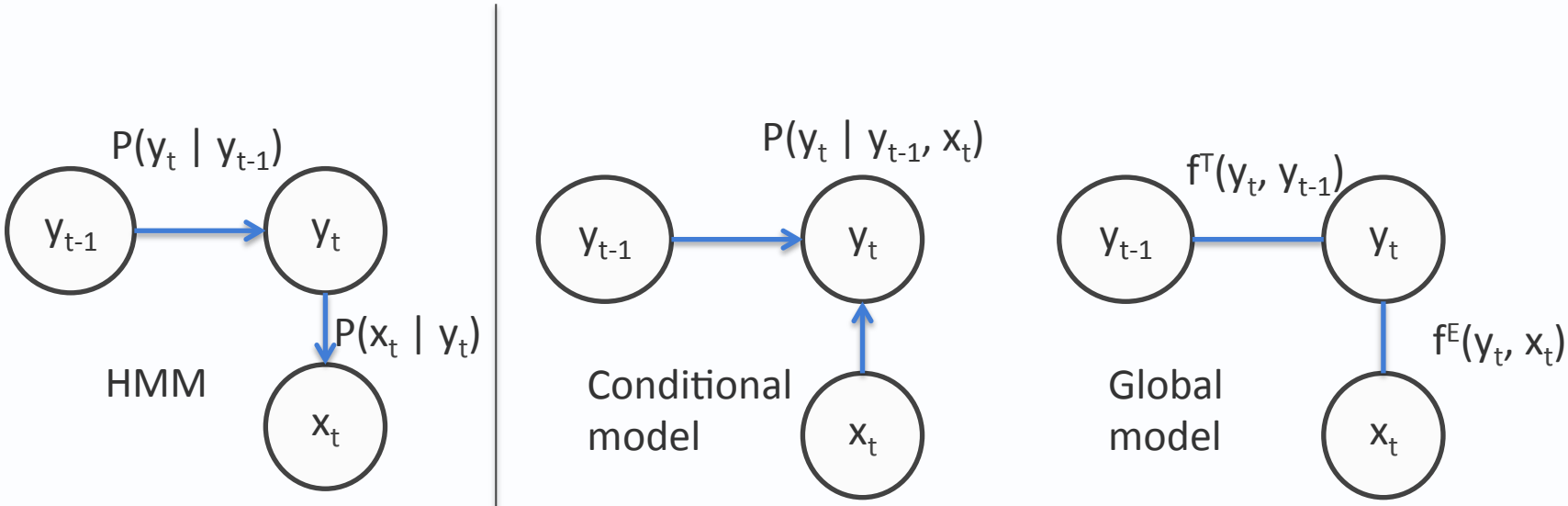
Generative

HMM vs. A local model vs. A global model



Local: P is locally normalized to add up to one for each t

HMM vs. A local model vs. A global model



Generative

Discriminative

Local: P is locally normalized to add up to one for each t

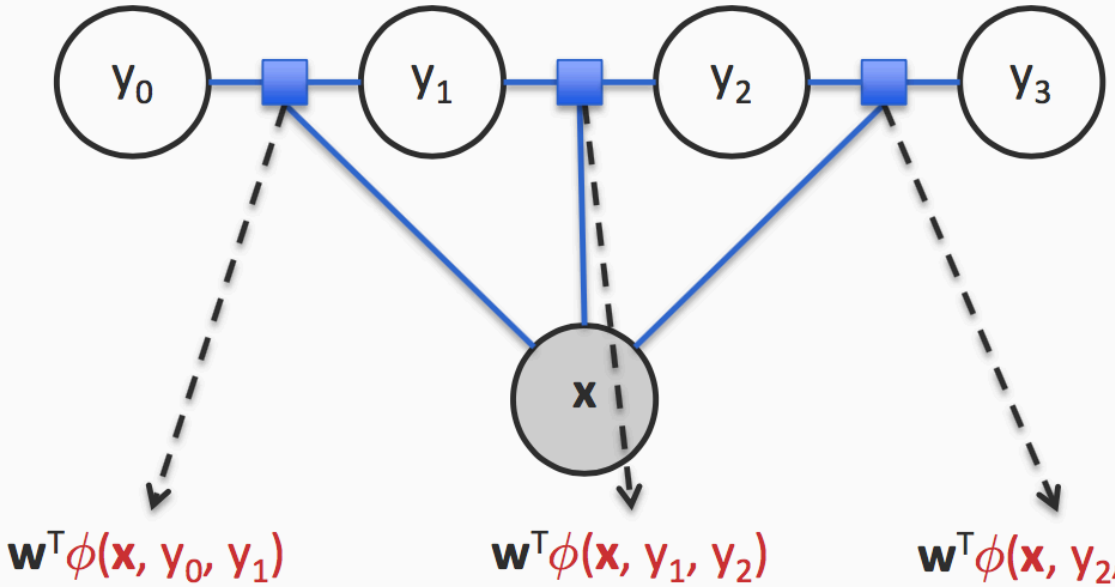
Global: The functions f^T and f^E are scores that are not normalized

Conditional Random Fields (CRF) for sequences

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z}\prod_i \exp\left(\mathbf{w}^T\phi(\mathbf{x}, y_i, y_{i-1})\right)$$

Z: Normalizing constant, sum over all sequences

$$Z = \sum_{\hat{\mathbf{y}}}\prod_i \exp\left(\mathbf{w}^T\phi(\mathbf{x}, \hat{y}_i, \hat{y}_{i-1})\right)$$

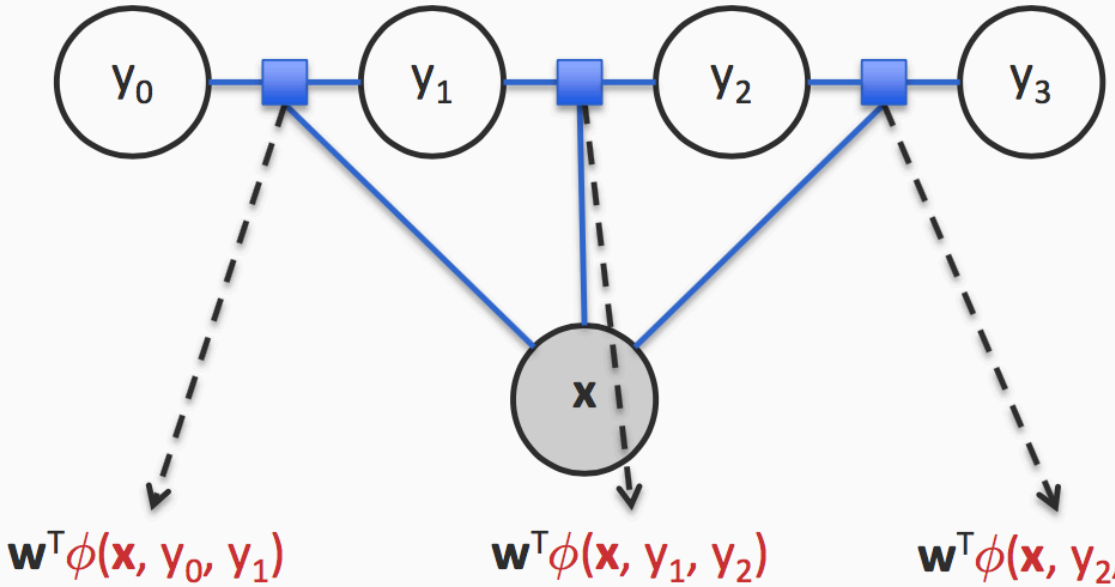


Conditional Random Fields (CRF) for sequences

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z}\prod_i \exp\left(\mathbf{w}^T\phi(\mathbf{x}, y_i, y_{i-1})\right)$$

Z: Normalizing constant, sum over all sequences

$$Z = \sum_{\hat{\mathbf{y}}}\prod_i \exp\left(\mathbf{w}^T\phi(\mathbf{x}, \hat{y}_i, \hat{y}_{i-1})\right)$$



CRF: A different view

- Input: **x**, Output: **y**

- Define a feature vector for the **entire** input and output sequence: $\phi(\mathbf{x}, \mathbf{y})$

- Define a giant log-linear model, P(**y** | **x**) parameterized by **w**

$$P(\mathbf{y}|\mathbf{x}) = \frac{\exp\left(\mathbf{w}^T\phi(\mathbf{x},\mathbf{y})\right)}{\sum_{\mathbf{y}'}\exp\left(\mathbf{w}^T\phi(\mathbf{x},\mathbf{y}')\right)} \propto \exp\left(\mathbf{w}^T\phi(\mathbf{x},\mathbf{y})\right)$$

  - Just like any other log-linear model, except
    - Space of **y** is the set of all possible sequences of the correct length
    - Normalization constant sums over all sequences

In an MEMM, probabilities were locally normalized

Prediction

Goal: To predict most probable sequence **y** an input **x**

$$\arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}) = \arg\max_{\mathbf{y}} \exp(\mathbf{w}^T \phi(\mathbf{x}, \mathbf{y})) = \arg\max_{\mathbf{y}} \mathbf{w}^T \phi(\mathbf{x}, \mathbf{y})$$

But the score decomposes as $\mathbf{w}^T \phi(\mathbf{x}, \mathbf{y}) = \sum_i \mathbf{w}^T \phi(\mathbf{x}, y_i, y_{i-1})$

Prediction via Viterbi (with sum instead of product)

$$\text{score}_0(s) = \mathbf{w}^T \phi(\mathbf{x}, y_0, \text{start})$$

$$\text{score}_i(s) = \max_{y_{i-1}} \left(\mathbf{w}^T \phi(\mathbf{x}, y_i, y_{i-1}) + \text{score}_{i-1}(y_{i-1}) \right)$$