Structured Prediction

EECS 183/283a: Natural Language Processing
(many slides adapted from Greg Durrett)

Structured Prediction

- **General problem:** we want to learn a (possibly conditional) distribution over some output space with <u>structure</u> to it
  - Structure: output space is a composition of variables whose values depend on each other's values

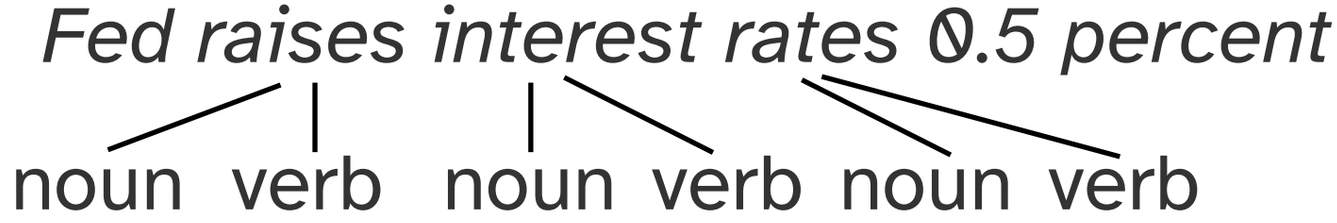
- Not structured prediction: simple classification tasks
  - Document classification (output space is set of classes)
  - Next-token prediction (output space is vocabulary)

Structured Prediction Problems

- Multi-class classification:
  - Output is a set of classes
  - Classes may not be independent: some classes are likely to co-occur, and others never co-occur
- Language modeling:
  - Output is a sequence of tokens
  - There might be long-distance dependencies between words across the sequence

Structured Prediction Problems

- Part-of-speech (POS) tagging
  - Output is a sequence of POS tags
  - Tags may depend on each other under lexical ambiguity



Structured Prediction Problems

- Part-of-speech (POS) tagging
  - Output is a sequence of POS tags
  - Tags may depend on each other under lexical ambiguity

Structured Prediction Problems

- Part-of-speech (POS) tagging
  - Output is a sequence of POS tags
  - Tags may depend on each other under lexical ambiguity

Fed raises interest rates 0.5 percent

noun **verb** **noun** verb **noun** verb

- Code generation
  - Output is a sequence of tokens
  - But they must be syntactically valid and compilable according to the programming language’s grammar

Challenges and Opportunities with Structured Prediction

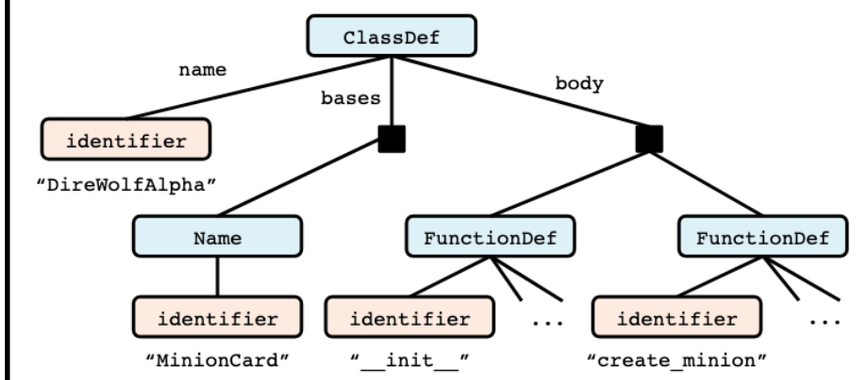
- Output space might be exponentially large, which makes search difficult at inference time

- But, known dependence between output variables might help narrow down the search at inference time

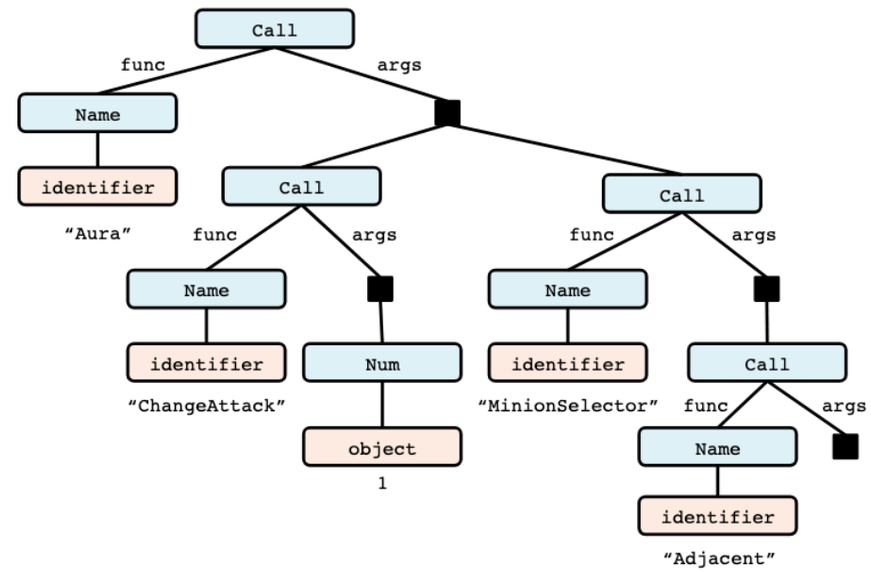
- E.g., constrained decoding for code generation


```
name: [
  'D', 'i', 'r', 'e', ' ',
  'W', 'o', 'l', 'f', ' ',
  'A', 'l', 'p', 'h', 'a']
cost: ['2']
type: ['Minion']
rarity: ['Common']
race: ['Beast']
class: ['Neutral']
description: [
  'Adjacent', 'minions', 'have',
  '+', '1', 'Attack', '.']
health: ['2']
attack: ['2']
durability: ['-1']
```

```
class DireWolfAlpha(MinionCard):
  def __init__(self):
    super().__init__(
      "Dire Wolf Alpha", 2, CHARACTER_CLASS.ALL,
      CARD_RARITY.COMMON, minion_type=MINION_TYPE.BEAST)
  def create_minion(self, player):
    return Minion(2, 2, auras=[
      Aura(ChangeAttack(1), MinionSelector(Adjacent()))
    ])
```



(a) The root portion of the AST.



(b) Excerpt from the same AST, corresponding to the code snippet `Aura(ChangeAttack(1),MinionSelector(Adjacent()))`.

Rabinovich et al. 2017

Challenges and Opportunities with Structured Prediction

- How can we use domain knowledge to improve search?
  - Constrained decoding
  - Reranking
  - Structured modeling
  - Learn it through training on lots of data
- If we have lots of data available to train on, why do we need prior domain knowledge at all?
  - Can still make learning or inference more efficient
  - Can also guarantee validity of outputs

Challenges and Opportunities with Structured Prediction

- Sometimes we just don’t have data available!
- Example tasks:
  - Generating JSON objects for a complex custom annotation schema
  - “Translating” from natural language proofs to executable Lean code
  - Reasoning tasks where answers can be generated by composing several structured reasoning traces

Case Study: Sequence Tagging

- Sequence tagging problems: $f : \mathcal{V}^N \rightarrow \mathcal{C}^N$
- Where we also have a prior over possible output tag sequences $p \in \Delta^{\mathcal{C}^N}$
- Sequence tagging tasks:
  - Part-of-speech tagging
  - Named entity recognition
  - Modern tagging tasks?

Case Study: Part-of-Speech (POS) Tagging

To start:

- Modern LLMs are quite good at POS tagging

- We also don’t need to do POS tagging anymore to build good language technologies

- But, it’s a well-studied task that will provide us with a good intuition about structured prediction problems in general

Part-of-Speech Tagging

Open class tags (English)

- Nouns
  - Proper nouns: *IBM, Italy*
  - Common nouns: *cat, snow*
- Verbs: *see, repost*
- Adjectives: *brown, cooked*
- Adverbs: *swiftly*

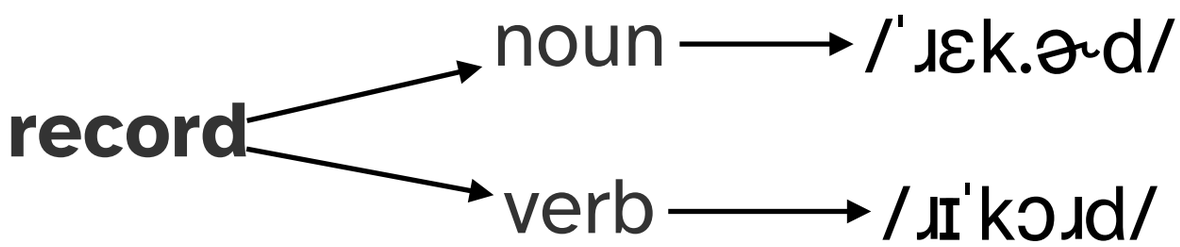
Part-of-Speech Tagging

“Closed” class tags (English)

- Determiners: *the*, *an*, *some*, *my*, etc.
- Conjunctions: *and*, *or*
- Pronouns: *they*, *us*
- Auxiliaries and modalities: *had* [VERB], *could*
- Prepositions: *up*, *to*, *in*
- Particles: combinations of prepositions with verbs into a single verbal phrase, e.g., *made up*

Part-of-Speech Tagging

Lexical ambiguity



Fed raises interest rates 0.5 percent

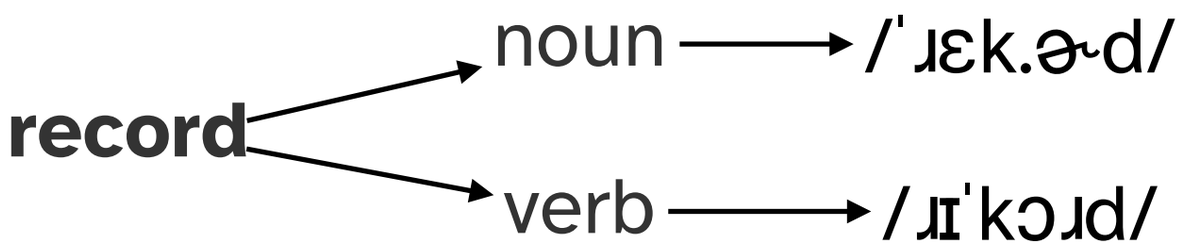
NNP	NNS	NN	NNS	CD	NN
VBD	VBZ	VB	VBZ		
VBN		VBP			

3 x 2 x 3 x 2 x 1 x 1

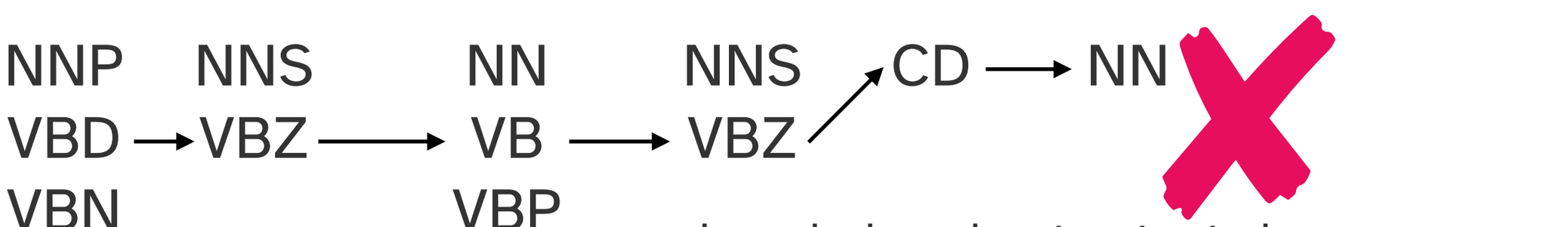
= 36 possible sequences

Part-of-Speech Tagging

Lexical ambiguity



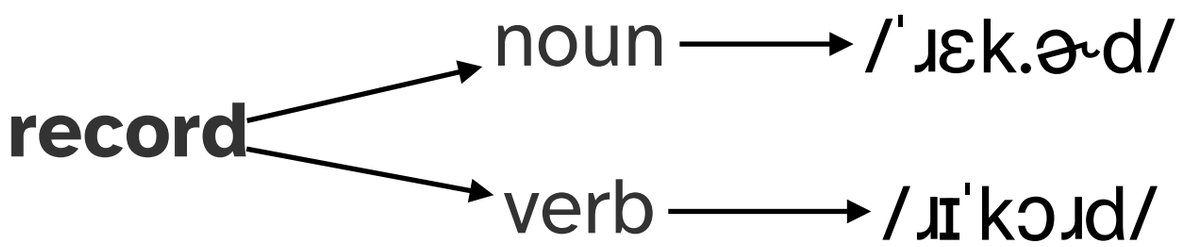
Fed raises interest rates 0.5 percent



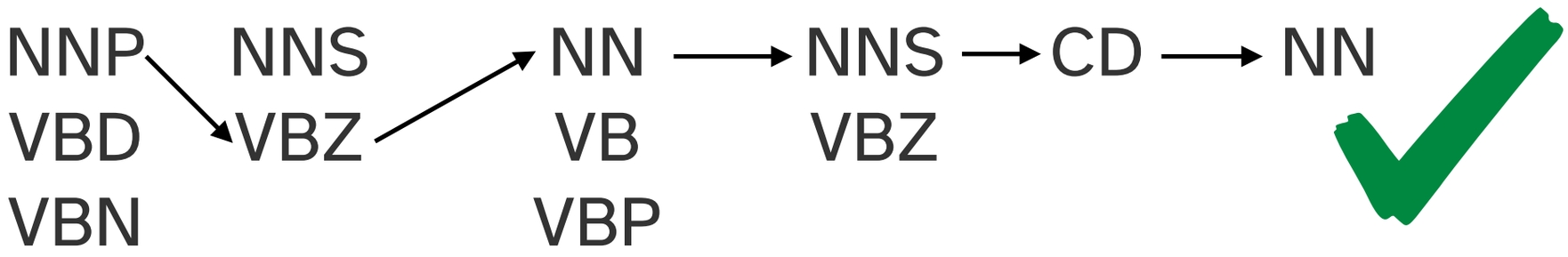
knowledge about output classes:
a long sequence of verbs is very unlikely

Part-of-Speech Tagging

Lexical ambiguity

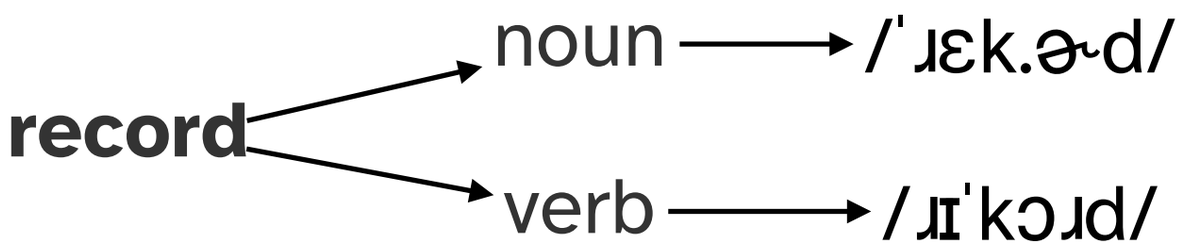


Fed raises interest rates 0.5 percent



Part-of-Speech Tagging

Lexical ambiguity



Fed raises interest rates 0.5 percent

NNP VBD VBN → NNS VBZ → NN VB VBP → NNS VBZ → CD → NN

prior knowledge about the relationship between input and output:
“rates” cannot be the object of “interest”

Part-of-Speech Tagging

- Sequence tagging problems: $f : \mathcal{V}^N \rightarrow \mathcal{P}^N$
- Where we also have a prior over possible output tag sequences $p \in \Delta^{\mathcal{P}^N}$
- **Simplification:** independently predict each POS tag: $p(Y_i \mid \overline{x}, i)$
  - E.g., embed token $\phi(x_i) \in \mathbb{R}^d$
  - Then classify each embedding $p(Y_i) = f(\phi(x_i))$ $f : \mathbb{R}^d \rightarrow \Delta^{\mathcal{P}}$
  - *interest* —> 55% NN, 35% VB, 10% VBN

Part-of-Speech Tagging

- Sequence tagging problems: $f : \mathcal{V}^N \rightarrow \mathcal{P}^N$
- Where we also have a prior over possible output tag sequences $p \in \Delta^{\mathcal{P}^N}$

- **Simplification:** independently predict each POS tag: $p(Y_i \mid \overline{x}, i)$
  - E.g., embed sequence $\mathrm{Emb}(\overline{x}) \in \mathbb{R}^{d \times N}$
  - Then classify each hidden state $p(Y_i) = f(h_i)$ $f : \mathbb{R}^d \rightarrow \Delta^{\mathcal{P}}$
  - But POS tags are still predicted independently of each other — no guarantee we'll satisfy prior $p \in \Delta^{\mathcal{P}^N}$

Part-of-Speech Tagging

Tradeoff between validity and efficiency

- Independently predicting output components (ignoring implicit structure) is fast (O(n)), but might result in invalid or very low-probability outputs

- Considering only the possible outputs (or reranking by their probability) improves quality and validity, but requires enumerating all possible outputs (potentially exponential)

- Can we do better? —> structured modeling and dynamic programming!

Hidden Markov Models

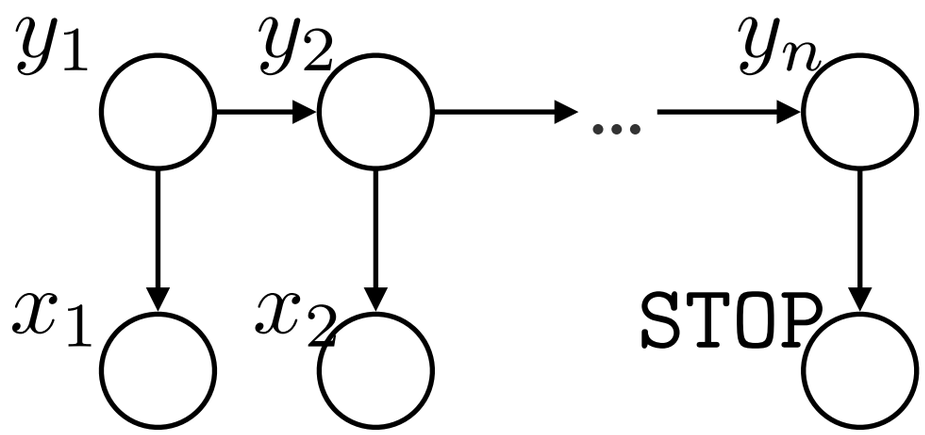
- **Preliminary: generative vs. discriminative models**
  - Generative model: parameterizes a joint distribution over random variables X and Y
  - Discriminative model: parameterizes a conditional distribution of target Y given observation X
- Classification: X = instance features, Y = instance class
- Language modeling:
  - “True” language modeling: Y = sequence of tokens
  - Autoregressive approximation: X = previous tokens, Y = next token
- Part-of-speech tagging: X = sequence of tokens, Y = part of speech tags

Hidden Markov Models

- Tags: $y_i \in \mathcal{P}$
- Words: $x_i \in \mathcal{V}$
- HMM generative model gives us $P(\overline{x}, \overline{y})$

$$p(y_1)p(x_1 \mid y_1)p(y_2 \mid y_1)p(x_2 \mid y_2)\ldots p(\texttt{STOP} \mid y_n)$$

Generative story:



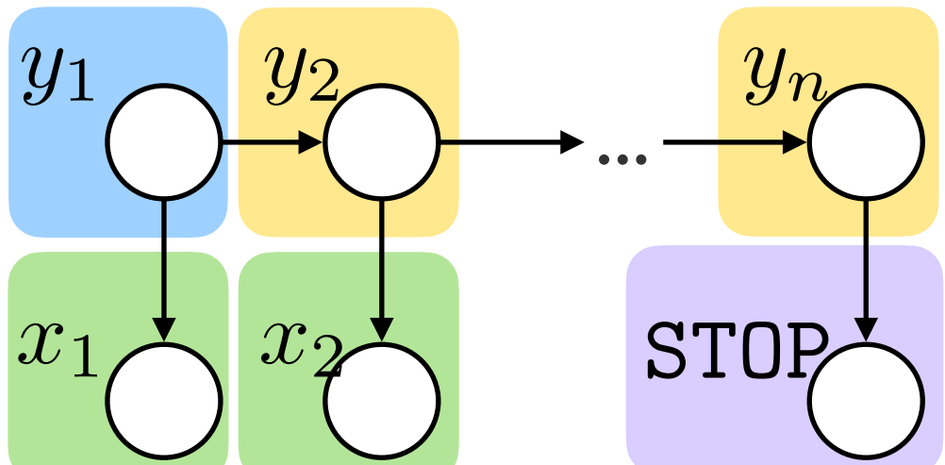
Markov process: y_i is conditionally independent of $y_1 \cdots y_{i-2}$ given y_{i-1}

Words are conditionally independent of one another given their parts of speech

Hidden Markov Models

$$p(y_1)p(x_1 \mid y_1)p(y_2 \mid y_1)p(x_2 \mid y_2)\ldots p(\texttt{STOP} \mid y_n)$$

Generative story:



Distribution over first-token POS tag $p(y_1) \in \Delta^P$

Emission probabilities $E \in \Delta^{\mathcal{P} \times \mathcal{V}}$

Transition probabilities $T \in \Delta^{\mathcal{P} \times \mathcal{P}}$

Probability of stopping after seeing POS tag $p(\texttt{STOP} \mid y) \in \mathbb{R}^{|\mathcal{P}|}_{[0:1]}$

Two core questions:

- how to determine these probabilities? (parameter estimation)
- how to use this model to map from a sentence to a sequence of POS tags? (inference)

HMM Parameter Estimation

- Labeled training data: $\left\{\left(\overline{x}^{(i)}, \overline{y}^{(i)}\right)\right\}_{i=1}^{N}$
- Training objective: maximize **joint** likelihood of training data (maximum likelihood estimation)

$$\sum_i \log P\left(\overline{y}^{(i)}, \overline{x}^{(i)}\right)$$

$$= \sum_{i=1}^{N} \left(\log P\left(y_1^{(i)}\right) + \sum_{j=1}^{|\overline{x}^{(i)}|} \log P\left(x_j^{(i)} \mid y_j^{(i)}\right) \right.$$

$$\left. + \sum_{j=1}^{|\overline{x}^{(i)}|} \log P\left(y_j^{(i)} \mid y_{j-1}^{(i)}\right) + \log P\left(\texttt{STOP} \mid y_{|\overline{x}^{(i)}|}^{(i)}\right) \right)$$

First-token POS tag
Emission probabilities
Transition probabilities
Probability of stopping

Count-Based MLE

Similar to count-based n-gram models: just count and normalize occurrences:

- How often does the first token have POS tag p?
- How often does the sentence stop if we just had tag p?
- If our current POS tag is p, how often is it followed by p'?
- If our current POS tag is p, how often does it "emit" word x?

Count-Based MLE

training data

they can
N V

they fish
N V

start probabilities

	Count	$P(y_1)$
N	2	1.0
V	0	0.0

transition probabilities

	N	V	**STOP**
N	0	2	0
V	0	0	2

$p(y_j \mid y_{j-1})$

	N	V	**STOP**
N	0.0	1.0	0.0
V	0.0	0.0	1.0

emission probabilities

	they	can	fish
N	2	0	0
V	0	1	1

$p(x_j \mid y_j)$

	they	can	fish
N	1.0	0.0	0.0
V	0.0	0.5	0.5

Smoothing

training data

they can they fish

N V N V

start probabilities

	Count	$P(y_1)$
N	3	0.75
V	1	0.25

transition probabilities

	N	V	**STOP**
N	1	3	1
V	1	1	3

$p(y_j \mid y_{j-1})$

	N	V	**STOP**
N	0.2	0.6	0.2
V	0.2	0.2	0.6

emission probabilities

	they	can	fish
N	3	1	1
V	1	2	2

$p(x_j \mid y_j)$

	they	can	fish
N	0.6	0.2	0.2
V	0.2	0.4	0.4

Joint Probability of a Tagged Sequence

training data

they can they fish
N V N V

start probabilities

	Count	$P(y_1)$
N	3	0.75
V	1	0.25

transition probabilities

$p(y_j \mid y_{j-1})$

	N	V	**STOP**
N	0.2	0.6	0.2
V	0.2	0.2	0.6

emission probabilities

$p(x_j \mid y_j)$

	they	can	fish
N	0.6	0.2	0.2
V	0.2	0.4	0.4

they can fish $P(\overline{y}, \overline{x}) =$
N V V

Decoding

$$P(\overline{x}, \overline{y}) = p(y_1)p(x_1 \mid y_1)p(y_2 \mid y_1)p(x_2 \mid y_2) \ldots p(\texttt{STOP} \mid y_n)$$

- Problem: find $\arg\max_{\overline{y}} P(\overline{y} \mid \overline{x}) = \arg\max_{\overline{y}} \frac{P(\overline{y}, \overline{x})}{P(\overline{x})}$ (Bayes rule)

$$= \arg\max_{\overline{y}} P(\overline{y}, \overline{x})$$

$$= \arg\max_{\overline{y}} \log P(\overline{y}, \overline{x})$$

$$= \arg \max_{\tilde{y}_1, \ldots, \tilde{y}_n} (\log P(\tilde{y}_1) + \log P(x_1 \mid \tilde{y}_1) + \log P(\tilde{y}_2 \mid \tilde{y}_1) \ldots)$$

Problem: can't just search over all possible $\tilde{y}$

Viterbi Algorithm

- Main principle: dynamic programming
- Define $v \in \mathbb{R}^{|\overline{x}| \times |\mathcal{P}|}$, where $v_i(\tilde{y})$ is the score of the best path ending in $\tilde{y}$ at time i
- Base case: $v_1(\tilde{y}) = \log P(x_1 \mid \tilde{y}) + \log P(\tilde{y})$

Viterbi Algorithm

- Main principle: dynamic programming
- Define $v \in \mathbb{R}^{|\overline{x}| \times |\mathcal{P}|}$, where $v_i(\tilde{y})$ is the score of the best path ending in $\tilde{y}$ at time i
- Base case: $v_1(\tilde{y}) = \log P(x_1 \mid \tilde{y}) + \log P(\tilde{y})$
- Recurrence:

$$v_i(\tilde{y}) = \log P(x_i \mid \tilde{y}) + \max_{\tilde{y}_{\text{prev}}} \left(\log P(\tilde{y} \mid \tilde{y}_{\text{prev}}) + v_{i-1}(\tilde{y}_{\text{prev}}) \right)$$

Viterbi Algorithm

$$v_1(\tilde{y}) = \log P(x_1 \mid \tilde{y}) + \log P(\tilde{y})$$

$$v_i(\tilde{y}) = \log P(x_i \mid \tilde{y}) + \max_{\tilde{y}_{\text{prev}}} \left(\log P(\tilde{y} \mid \tilde{y}_{\text{prev}}) + v_{i-1}(\tilde{y}_{\text{prev}})\right)$$

- Why does this work?
  - The best tag sequence has the score:

$$\max_{\tilde{y}_n, \ldots, \tilde{y}_1} \left(\log P(\texttt{STOP} \mid \tilde{y}_n) + \log P(x_n \mid \tilde{y}_n) + \cdots + \log P(x_1 \mid \tilde{y}_1) + \log P(\tilde{y}_1)\right)$$

$$= \max_{\tilde{y}_n, \ldots, \tilde{y}_2} \left(\log P(\texttt{STOP} \mid \tilde{y}_n) + \log P(x_n \mid \tilde{y}_n) + \ldots\right) + \max_{\tilde{y}_1} \left(\log P(x_1 \mid \tilde{y}_1) + \log P(\tilde{y}_1)\right)$$

Viterbi Algorithm

$$v_1(\tilde{y}) = \log P(x_1 \mid \tilde{y}) + \log P(\tilde{y})$$

$$v_i(\tilde{y}) = \log P(x_i \mid \tilde{y}) + \max_{\tilde{y}_{\text{prev}}} \left(\log P(\tilde{y} \mid \tilde{y}_{\text{prev}}) + v_{i-1}(\tilde{y}_{\text{prev}}) \right)$$

- Why does this work?
  - The best tag sequence has the score:

$$\max_{\tilde{y}_n, \ldots, \tilde{y}_1} \left(\log P(\texttt{STOP} \mid \tilde{y}_n) + \log P(x_n \mid \tilde{y}_n) + \cdots + \log P(x_1 \mid \tilde{y}_1) + \log P(\tilde{y}_1) \right)$$

$$= \max_{\tilde{y}_n, \ldots, \tilde{y}_2} \left(\log P(\texttt{STOP} \mid \tilde{y}_n) + \log P(x_n \mid \tilde{y}_n) + \cdots + \max_{\tilde{y}} v_1(\tilde{y}) \right)$$

Viterbi Algorithm

$$v_1(\tilde{y}) = \log P(x_1 \mid \tilde{y}) + \log P(\tilde{y})$$

$$v_i(\tilde{y}) = \log P(x_i \mid \tilde{y}) + \max_{\tilde{y}_{\text{prev}}} \left(\log P(\tilde{y} \mid \tilde{y}_{\text{prev}}) + v_{i-1}(\tilde{y}_{\text{prev}}) \right)$$

- Why does this work?
  - The best tag sequence has the score:

$$\max_{\tilde{y}_n, \ldots, \tilde{y}_1} \left(\log P(\texttt{STOP} \mid \tilde{y}_n) + \log P(x_n \mid \tilde{y}_n) + \cdots + \log P(x_1 \mid \tilde{y}_1) + \log P(\tilde{y}_1) \right)$$

$$= \max_{\tilde{y}_n, \ldots, \tilde{y}_2} \Bigg(\log P(\texttt{STOP} \mid \tilde{y}_n) + \log P(x_n \mid \tilde{y}_n) + \cdots + \log P(x_2 \mid \tilde{y}_2) + \log P\left(\tilde{y}_2 \mid \arg\max_{\tilde{y}} v_1(\tilde{y}) \right) \Bigg)$$

$$+ \max_{\tilde{y}} v_1(\tilde{y})$$

Viterbi Algorithm: Step-by-Step

$$v_1(\tilde{y}) = \log P(x_1 \mid \tilde{y}) + \log P(\tilde{y})$$

$$v_i(\tilde{y}) = \log P(x_i \mid \tilde{y}) + \max_{\tilde{y}_{\text{prev}}} \left(\log P(\tilde{y} \mid \tilde{y}_{\text{prev}}) + v_{i-1}(\tilde{y}_{\text{prev}}) \right)$$

To compute $\max_{\tilde{y}} \log P(\overline{x}, \overline{y})$**:**

```
v = 0^{|x̄| × |𝒫|}
for i = 1 ... |x̄|
  for ỹ in 𝒫
    if i == 1:
      v_1(ỹ) = log P(x_1 | ỹ) + log P(ỹ)
    else:
      v_i(ỹ) = log P(x_i | ỹ) + max_{ỹ_prev} (log P(ỹ | ỹ_prev) + v_{i-1}(ỹ_prev))
return v_{|x̄|+1}(STOP)
```

keep track of this over time to reconstruct the maximum-probability sequence

$$\arg\max_{\tilde{y}} \log P(\overline{x}, \overline{y})$$

Viterbi Algorithm Example

$$v_1(\tilde{y}) = \log P(x_1 \mid \tilde{y}) + \log P(\tilde{y})$$

$$v_i(\tilde{y}) = \log P(x_i \mid \tilde{y}) + \max_{\tilde{y}_{\text{prev}}} \left(\log P(\tilde{y} \mid \tilde{y}_{\text{prev}}) + v_{i-1}(\tilde{y}_{\text{prev}})\right)$$

start logprobs

N	-1
V	-1

transition logprobs

	N	V	**STOP**
N	-2	-1	-1
V	-1	-1	-2

emission logprobs

	they	can	fish
N	-1	-3	-1
V	-3	-1	-1

v	N	V
they		
can		
can		
fish		
STOP		