Sequence Tagging

CSE 5525: Foundations of Speech and Natural Language Processing

https://shocheen.github.io/courses/cse-5525-spring-2025

THE OHIO STATE UNIVERSITY

Slide Credits: Ana Marasović

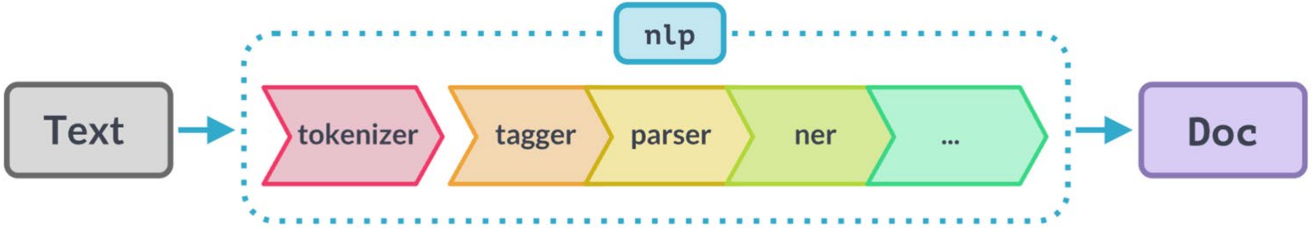
Logistics

- Homework 3 is due tonight.
  - Use all your late/slip days if you need now.
  - No late days for final project deadlines.

Lectures so far

1. Warm-up (4 lectures): Practical Intro to Machine/Deep Learning w/ an NLP application.
2. Modern LM fundaments (12 lectures): Background, key ingredients.
3. **Linguistic Structure Prediction (2 lectures).**
4. Rest of the lectures: Modern LMs in practice.

“Classical” NLP Pipeline



NAME	COMPONENT	CREATES	DESCRIPTION
tokenizer	`Tokenizer`	`Doc`	Segment text into tokens.
processing pipeline			
tagger	`Tagger`	`Token.tag`	Assign part-of-speech tags.
parser	`DependencyParser`	`Token.head`, `Token.dep`, `Doc.sents`, `Doc.noun_chunks`	Assign dependency labels.
ner	`EntityRecognizer`	`Doc.ents`, `Token.ent_iob`, `Token.ent_type`	Detect and label named entities.
lemmatizer	`Lemmatizer`	`Token.lemma`	Assign base forms.
textcat	`TextCategorizer`	`Doc.cats`	Assign document labels.
custom	custom components	`Doc._.xxx`, `Token._.xxx`, `Span._.xxx`	Assign custom attributes, methods or properties.

Source: https://spacy.io/usage/processing-pipelines

spaCy

Why should you know classic NLP tasks focused on aspects of the language system? [[Optiz, Wein, Schneider, 2024]]

Resources:
Creating lexicons and corpora with sensitivity to language, dialect, genre, & style variations

Evaluation:
Designing human evaluations, interrogating automatic metrics, & analyzing linguistic challenges for systems

Low-resource settings:
Better understand why approaches that work well for English or French might not work well for Swahili or Arapaho

80% languages have no corpus for supervised machine learning or LM pretraining

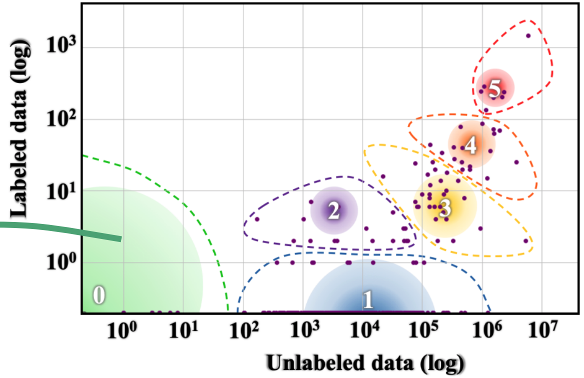
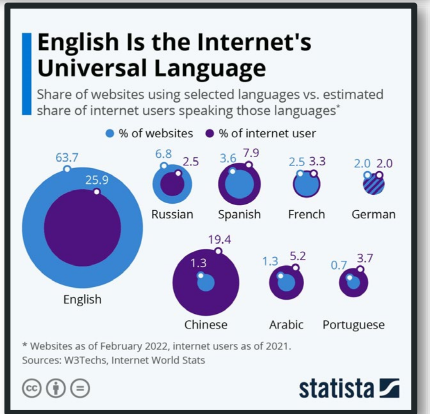


Figure 2: Language Resource Distribution: The size of the gradient circle represents the number of languages in the class. The color spectrum VIBGYOR, represents the total speaker population size from low to high. Bounding curves used to demonstrate covered points by that language class.

"3 in 4 users are unable to understand more than 60% of all websites, at least without a translation tool."



Why should you know classic NLP tasks focused on aspects of the language system? (cont.) [Optiz, Wein, Schneider, 2024]

Interpretability:
We would lack appropriate meta-language for describing many observed patterns

An illustration of this for the question of whether LLMs understand meaning:

- Learning a language: learning how surface forms (language text) connect to underlying structures
- When people and machines do this explicitly ⇒ parsing (part-of-speech tagging, named entity recognition, phrase chunking, coreference resolution)
- How is this core problem addressed in deep learning systems (and in human sentence processing)?

Study of language:
Corpus linguistics, documentary and historical linguistics, ...

Goals of Today's Lecture

Goal: Learn two classic sequence labeling tasks and a non-neural supervised approach to solving them

- Part-of-Speech (PoS) Tagging
- Named Entity Recognition (NER)
- Hidden Markov Model: Formulation
- Hidden Markov Model: Parameter estimation
- Viterbi algorithm

Parts of Speech (POS)

Part of Speech (POS) are categories of words based on:

- their **grammatical relationship** with neighboring words

  or

- **morphological** properties about their affixes
  - The stem is the part of the word that carries its primary lexical meaning; often a root or base form
  - Affixes: Morphemes that are attached to a word stem to form a new word or word form
    - Prefix: Pre- in Preview
    - Suffix: -ed in Played
  - Morphological: Relating to the forms of words
    - Walk → Walking (suffix -ing changes the form and grammatical function)
    - Happy → Unhappy (prefix un- changes the meaning)

Two classes of words: Open vs. Closed

Closed class words

- Relatively fixed membership, meaning new words in this class are rarely coined
- **Function words:** Short, frequent words with grammatical function
  - Determiners: a, an, the
  - Pronouns: she, he, I
  - Prepositions: on, under, over, near, by,…
- PoS tags of such words are deterministic

Open class words

- Continually being created or borrowed
- Nouns (including proper nouns), verbs, adjectives, adverbs, & interjections

Jurafsky & Martin Section 17.1 define many classes of words that you should familiarize yourself with: different types of nouns, adverbs, verbs, pronouns; particle, article, conjunction, complementizer, copula, modals, etc.

Fun facts about other languages 🙂

- In **Korean**, the words corresponding to English adjectives act as a subclass of verbs, so what is in English an adjective “beautiful” acts in Korean like a verb meaning “to be beautiful”
- While many scholars believe that all human languages have the categories of noun and verb, others have argued that some languages, such as **Riau Indonesian** and **Tongan**, don’t even make this distinction

Ambiguity Resolution

I	will	book	a	room	at	the	hotel
PRP	MD	VB	DT	NN	IN	DT	NN

She	is	reading	an	interesting	book
PRP	VBZ	VBG	DT	JJ	**NN**

Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

Figure source: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

POS Tagging

The task of assigning a part-of-speech to each word in a text

Input: A sequence $x_1, x_2, \ldots, x_n$ of (tokenized) words and a **tagset**

Output: A sequence $y_1, y_2, \ldots, y_n$ of tags, each output y_i corresponding exactly to one x_i

An example of **sequence labelling** or **sequence tagging:**
The task of assigning a label to each unit in a sequence, thus mapping a sequence of observations to a sequence of labels of the same length

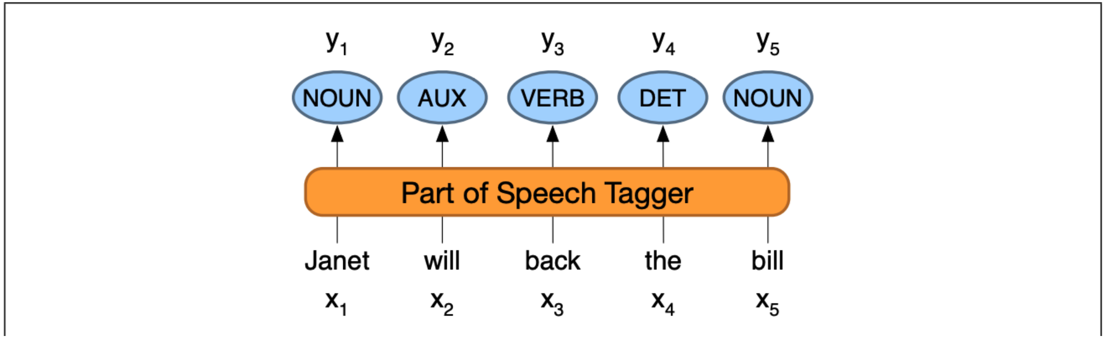


Figure source: Jurafsky & Martin

Why POS tagging?

Can be useful for other NLP tasks

- Parsing: POS tagging can improve syntactic parsing [the next step in the classical NLP pipeline]
  - Whether a word is a noun or a verb tells us about likely neighboring words (nouns in English are preceded by determiners and adjectives, verbs by nouns)
  - It also tells us about syntactic structure (verbs have dependency links to nouns)
- MT: reordering of adjectives and nouns
  - Say from Spanish where adjectives come after the nouns to English where they precede the nouns
- Sentiment or affective tasks:
  - May want to distinguish adjectives or other POS
- Text-to-speech
  - How do we pronounce "lead" or "object"?

Useful for linguistic or language-analytic computational tasks

- Need to control for POS when studying linguistic change like creation of new words, or meaning shift
- Or control for POS in measuring meaning similarity or difference

Slide source: Jurafsky & Martin

How difficult is POS tagging in English?

Roughly 15% of word *types* are ambiguous

- Janet is always PROPN, hesitantly is always ADV

But those 15% tend to be very common, so ∼60% of word *tokens* are ambiguous

E.g. type "*back*"

*earnings growth took a <u>back</u>/**ADJ** seat*

*a small building in the <u>back</u>/**NOUN***

*a clear majority of senators **back/VERB** the bill*

*enable the country to buy **back/PART** debt*

*I was twenty-one <u>back</u>/**ADV** then*

Not all possible tags for a given word are equally likely!

Majority baseline (useful beyond PoS tagging):
Given a word, predict the PoS tag which is most frequent in the training corpus

The most-frequent-tag baseline has an accuracy of ∼92% on the sections 22-24 of the WSJ corpus, only 5% lower than the SOTA and human ceiling

Slide source: Jurafsky & Martin

Sources of information for POS tagging

Prior probabilities of word/tag

- "will" is usually an AUX

Identity of neighboring words

- "the" means the next word is probably not a verb

Morphology and wordshape:

- Prefixes
  - *unable*: un- ⇒ ADJ
- Suffixes
  - *importantly*: -ly ⇒ ADJ
- Capitalization
  - *Janet*: CAP ⇒ PROPN

Slide source: Jurafsky & Martin

Standard algorithms for POS tagging

Supervised Machine Learning:

- **Hidden Markov Models [today]**
- Conditional Random Fields (CRF) / Maximum Entropy Markov Models (MEMM)
- Neural sequence models (RNNs or Transformers)
- Pretrained (Large) Language Models (like BERT), finetuned

All required a hand-labeled training set, all about equal performance (97% on English)

All make use of information sources we discussed

- Via human created features: HMMs and CRFs
- Via representation learning: Neural LMs

Slide source: Jurafsky & Martin

Goals of Today’s Lecture

Goal: Learn two classic sequence labeling tasks and a non-neural supervised approach to solving them

- Part-of-Speech (PoS) Tagging
- **Named Entity Recognition (NER)**
- Hidden Markov Model: Formulation
- Hidden Markov Model: Parameter estimation
- Viterbi algorithm

Named Entities (NEs)

Named entity typically means anything that can be referred to with a proper name

Most common 4 tags:

- PER (Person): “Marie Curie”
- LOC (Location): “New York City”
- ORG (Organization): “Stanford University”
- GPE (Geo-Political Entity): "Boulder, Colorado"

But the term is also extended to things that aren't entities, e.g., dates, times, prices

Entity classes can be specialized to a domain, e.g., GENE, PROTEIN, DISEASE, ...

They may range in specificity, e.g., ANIMAL, MAMMAL, DOG_BREED, ...

Often **multi-word phrases** ⇒ **Segmentation** problem: Where does the NE start/end?

Slide source: Jurafsky & Martin

Named Entity Recognition (NER)

The task of identify all spans in the text that denote some category of entities

As sequence labeling:

Input: A sequence $x_1, x_2, \ldots, x_n$ of (tokenized) words and the **set of entity types**

Output: A sequence $y_1, y_2, \ldots, y_n$ of **(B)IO(ES)** entity tags, each output y_i corresponding exactly to one x_i

(B)IO(ES) Tagging

- B: token that *begins* a span
- I: tokens *inside* a span
- O: tokens outside of any span
- E: token that *ends* a span
- S: singleton

Words	IO Label	BIO Label	BIOES Label
Jane	I-PER	B-PER	B-PER
Villanueva	I-PER	I-PER	E-PER
of	O	O	O
United	I-ORG	B-ORG	B-ORG
Airlines	I-ORG	I-ORG	I-ORG
Holding	I-ORG	I-ORG	E-ORG
discussed	O	O	O
the	O	O	O
Chicago	I-LOC	B-LOC	S-LOC
route	O	O	O
.	O	O	O

Ambiguity Resolution

Type	Tag	Sample Categories	Example sentences
People	PER	people, characters	**Turing** is a giant of computer science.
Organization	ORG	companies, sports teams	The **IPCC** warned about the cyclone.
Location	LOC	regions, mountains, seas	**Mt. Sanitas** is in **Sunshine Canyon**.
Geo-Political Entity	GPE	countries, states	**Palo Alto** is raising the fees for parking.

Washington was born into slavery on the farm of James Burroughs.	PER
Washington went up 2 games to 1 in the four-game series.	ORG
Blair arrived in **Washington** for what may well be his last state visit.	LOC
In June, **Washington** passed a primary seatbelt law.	GPE

Examples from: Jurafsky & Martin

Why NER?

Can be useful for other NLP tasks

- Targeted aspect-based sentiment analysis: Consumer's sentiment toward a particular company or person?
- Question Answering: Answer questions about an entity?
- Information Extraction: Extracting facts about entities from text

Or linguistic or language-analytic computational tasks

- For example, you may want to study how British novels depicted India during the colonial period
- Using a corpus of novels, you could use NER to:
    a. Extract **locations** like Calcutta, Delhi, and Himalayas to track representations of geography
    b. Identify **people** such as Rama or Lord Cornwallis to study how individuals were characterized
    c. Detect **dates/events** like 1857 (Indian Rebellion) for cultural context

Slide source: Jurafsky & Martin

Standard algorithms for NER

Supervised Machine Learning given a human-labeled training set of text annotated with tags

- **Hidden Markov Models [today]**
- Conditional Random Fields (CRF) / Maximum Entropy Markov Models (MEMM)
- Neural sequence models (RNNs or Transformers)
- Pretrained (Large) Language Models (like BERT), finetuned

Slide source: Jurafsky & Martin

Goals of Today's Lecture

Goal: Learn two classic sequence labeling tasks and a non-neural supervised approach to solving them

- Part-of-Speech (PoS) Tagging
- Named Entity Recognition (NER)
- **Hidden Markov Model: Formulation**
- Hidden Markov Model: Parameter estimation
- Viterbi algorithm

Hidden Markov Model (HMM) is a probabilistic sequence model: given a sequence of units (words, letters, morphemes, sentences, whatever), it computes a probability distribution over possible sequences of labels and chooses the best label sequence.

Can you recall when we mentioned Markov before?
When did we last calculate a probability distribution for some potential sequences?

Components of a (Discrete) Markov chain

$Q = q_1 q_2 \cdots q_N$ … a set of N **states=observations** [random variables which can take on values from some set]

Components of a (Discrete) Markov chain (cont.)

$Q = q_1 q_2 \ldots q_N$ … a set of N **states=observations** [random variables which can take on values from some set]

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{N1} \\ \vdots & \vdots & \ldots & \vdots \\ a_{N1} & a_{N2} & \ldots & a_{NN} \end{bmatrix}$$

… a **transition probability** matrix A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$

Components of a (Discrete) Markov chain (cont.)

$Q = q_1 q_2 \ldots q_N$ … a set of N **states=observations** [random variables which can take on values from some set]

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{N1} \\ \vdots & \vdots & \ldots & \vdots \\ a_{N1} & a_{N2} & \ldots & a_{NN} \end{bmatrix}$$

… a **transition probability** matrix A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$

$\pi = \pi_1, \pi_2, \ldots, \pi_N$ … an **initial probability distribution** over states

π_i … is the probability that the Markov chain will start in state i $\quad \sum_{i=1}^{N} \pi_i = 1$

Components of a (Discrete) Markov chain (cont.)

$Q = q_1 q_2 \ldots q_N$ … a set of N **states=observations** [random variables which can take on values from some set]

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{N1} \\ \vdots & \vdots & \ldots & \vdots \\ a_{N1} & a_{N2} & \ldots & a_{NN} \end{bmatrix}$$

… a **transition probability** matrix A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$

$\pi = \pi_1, \pi_2, \ldots, \pi_N$ … an **initial probability distribution** over states

π_i … is the probability that the Markov chain will start in state i $\quad \sum_{i=1}^{N} \pi_i = 1$

Some states j may have $\pi_j = 0$, meaning that they cannot be initial states

(1st order) **Markov assumption:** $\mathbb{P}(q_i = x | q_1 \ldots q_{i-1}) = \mathbb{P}(q_i = x | q_{i-1})$

Components of a **Hidden** Markov Model

Observable vs. **hidden** events: the former we observe directly, the latter must be inferred

- We see words, and must infer the part-of-speech tags from the word sequence

We build HMM from the same components as the Markov chain and introduce:

$$B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1V} \\ \vdots & \vdots & \cdots & \vdots \\ b_{N1} & b_{N2} & \cdots & b_{NV} \end{bmatrix}$$

... a matrix of **emission probabilities**, each expressing the probability of an output observation o_t (drawn from a vocabulary $V = v_1, v_2, ..., v_V$) being generated from a hidden state q_i

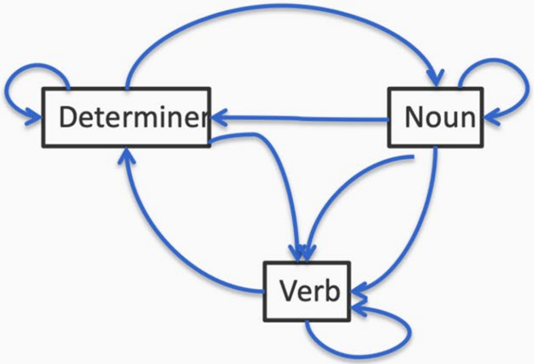
And we make another assumption besides the Markov assumption:

- **Output independence**: The probability of an output observation o_i depends only on the state that produced the observation q_i and not on any other states or any other observation

$$\mathbb{P}(o_i | q_1, \ldots, q_T, o_1, \ldots, o_T) = \mathbb{P}(o_i | q_i)$$

Toy part of speech example

Transitions



Emissions

P(The | Determiner) = 0.5
P(A | Determiner) = 0.3
P(An | Determiner) = 0.1
P(Fed | Determiner) = 0
…

P(Fed| Noun) = 0.001
P(raises| Noun) = 0.04
P(interest| Noun) = 0.07
P(The| Noun) = 0
…

Each edge here is associated with a ***transition probability***

Emission probabilities: *Given that the system is in a certain state, these are probabilities that it will emit a certain observation*

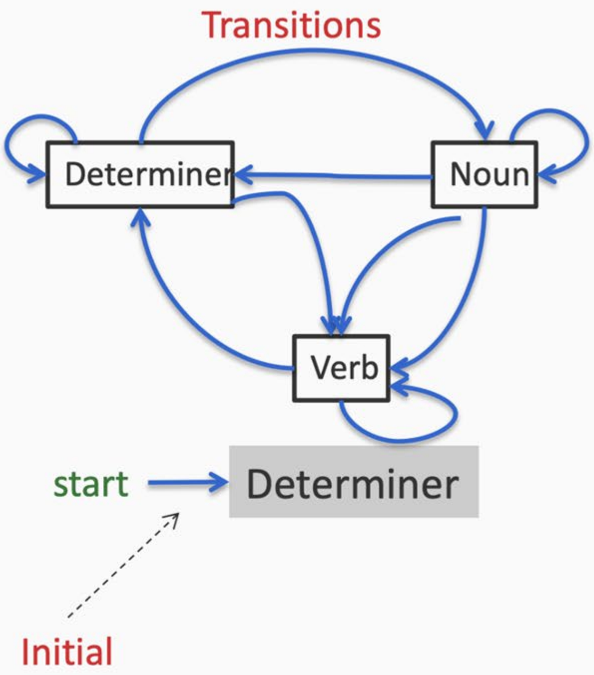
Initial

P(Determiner) = 0.9
P(Noun) = 0.08
P(Verb) = 0.02

Initial probabilities*: What is the probability that the sequence starts in a certain state?*

Slide credit: Vivek Srikumar

Toy part of speech example



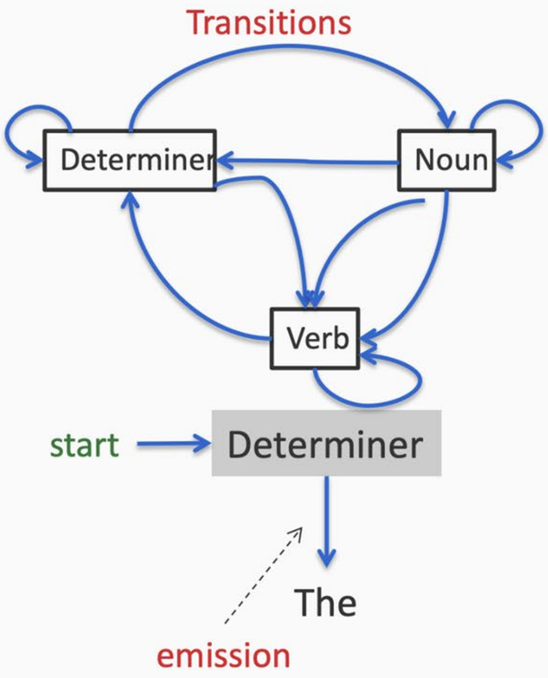
Emissions

P(The | Determiner) = 0.5
P(A | Determiner) = 0.3
P(An | Determiner) = 0.1
P(Fed | Determiner) = 0
...

P(Fed| Noun) = 0.001
P(raises| Noun) = 0.04
P(interest| Noun) = 0.07
P(The| Noun) = 0
...

Slide credit: Vivek Srikumar

Toy part of speech example



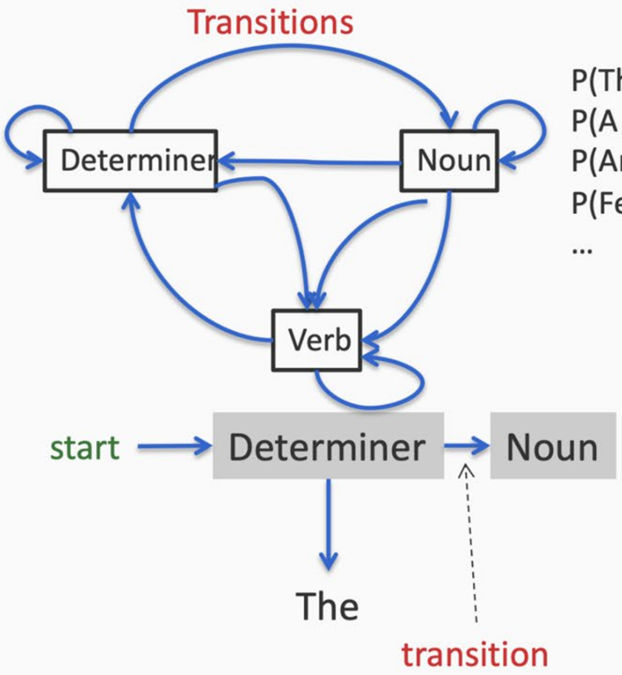
Emissions

P(The | Determiner) = 0.5
P(A | Determiner) = 0.3
P(An | Determiner) = 0.1
P(Fed | Determiner) = 0
...

P(Fed| Noun) = 0.001
P(raises| Noun) = 0.04
P(interest| Noun) = 0.07
P(The| Noun) = 0
...

Slide credit: Vivek Srikumar

Toy part of speech example



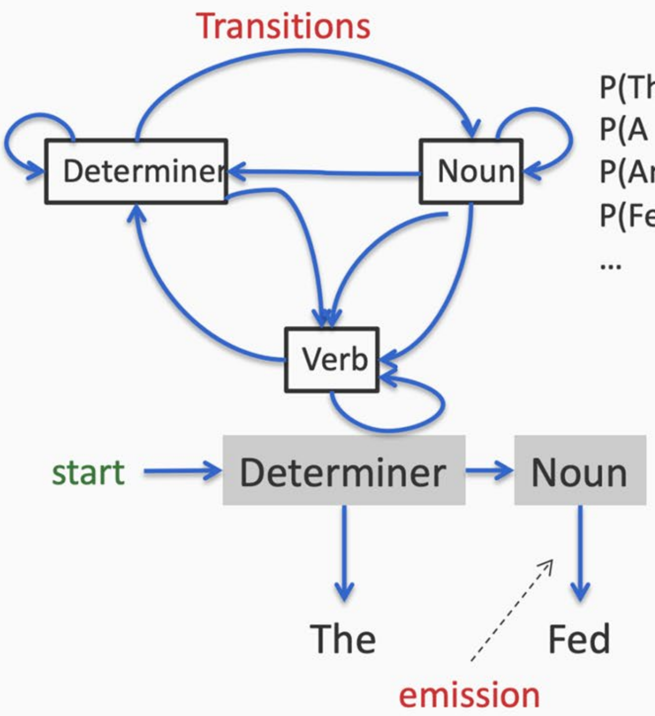
Emissions

P(The | Determiner) = 0.5
P(A | Determiner) = 0.3
P(An | Determiner) = 0.1
P(Fed | Determiner) = 0
...

P(Fed| Noun) = 0.001
P(raises| Noun) = 0.04
P(interest| Noun) = 0.07
P(The| Noun) = 0
...

Slide credit: Vivek Srikumar

Toy part of speech example



Emissions

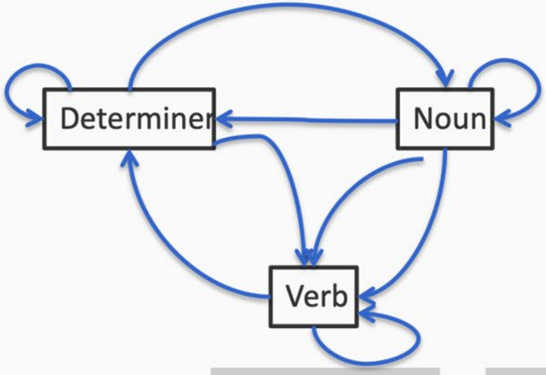
P(The | Determiner) = 0.5
P(A | Determiner) = 0.3
P(An | Determiner) = 0.1
P(Fed | Determiner) = 0
…

P(Fed| Noun) = 0.001
P(raises| Noun) = 0.04
P(interest| Noun) = 0.07
P(The| Noun) = 0
…

Slide credit: Vivek Srikumar

Toy part of speech example

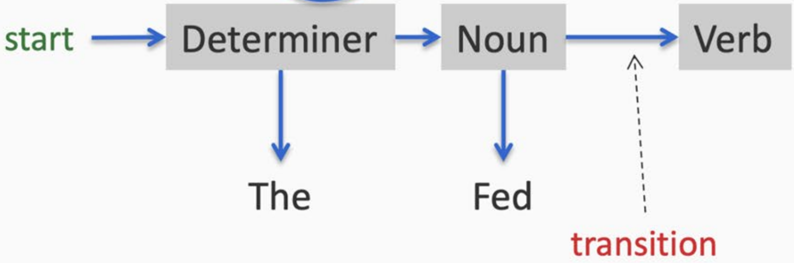
Transitions



Emissions

P(The | Determiner) = 0.5
P(A | Determiner) = 0.3
P(An | Determiner) = 0.1
P(Fed | Determiner) = 0
…

P(Fed| Noun) = 0.001
P(raises| Noun) = 0.04
P(interest| Noun) = 0.07
P(The| Noun) = 0
…



Slide credit: Vivek Srikumar

HMM Tagger

$w_1, w_2, \ldots, w_N$ ... a sequence of observed words

$$\hat{t}_{1:N} = \text{argmax}_{t_{1:N}} \mathbb{P}(t_1, \ldots, t_N | w_1, \ldots, w_N)$$

Bayes rule to turn to the process we've just seen with the toy example

$$= \text{argmax}_{t_{1:N}} \frac{\mathbb{P}(w_1, \ldots, w_N | t_1, \ldots, t_N)\mathbb{P}(t_1, \ldots, t_N)}{\mathbb{P}(w_1, \ldots, w_N)}$$

The denominator independent of tags

$$= \text{argmax}_{t_{1:N}} \mathbb{P}(w_1, \ldots, w_N | t_1, \ldots, t_N)\mathbb{P}(t_1, \ldots, t_N)$$

Markov and output independence assumptions

$$\approx \text{argmax}_{t_{1:N}} \prod_{i=1}^{N} \mathbb{P}(w_i | t_i) \cdot \mathbb{P}(t_1) \prod_{i=2}^{N} \mathbb{P}(t_i | t_{i-1})$$

Let's use the notation we introduced

$$\approx \text{argmax}_{t_{1:N}} \prod_{i=1}^{N} B_{t_i, w_i} \cdot \pi_{t_1} \prod_{i=2}^{N} A_{t_{i-1}, t_i}$$

HMM Tagger

$w_1, w_2, \ldots, w_N$ a sequence of observed words

$$\hat{t}_{1:N} = \text{argmax}_{t_{1:N}} \mathbb{P}(t_1, \ldots, t_N | w_1, \ldots, w_N)$$

Bayes rule to turn to the process we've just seen with the toy example

$$= \text{argmax}_{t_{1:N}} \frac{\mathbb{P}(w_1, \ldots, w_N | t_1, \ldots, t_N)\mathbb{P}(t_1, \ldots, t_N)}{\mathbb{P}(w_1, \ldots, w_N)}$$

The denominator independent of tags

$$= \text{argmax}_{t_{1:N}} \mathbb{P}(w_1, \ldots, w_N | t_1, \ldots, t_N)\mathbb{P}(t_1, \ldots$$

Markov and output independence assumptions

$$\approx \text{argmax}_{t_{1:N}} \prod_{i=1}^{N} \mathbb{P}(w_i | t_i) \cdot \mathbb{P}(t_1) \prod_{i=2}^{N} \mathbb{P}(t_i | t_{i-1})$$

Let's use the notation we introduced

$$\approx \text{argmax}_{t_{1:N}} \prod_{i=1}^{N} B_{t_i, w_i} \cdot \pi_{t_1} \prod_{i=2}^{N} A_{t_{i-1}, t_i}$$

Once we have estimated emission, initial, and transition probabilities, all we need to do to get the most probable sequence of tags for a given sequence of words is to plug the probabilities into this equation for every possible sequence of tags and return the sequence that maximizes the equation's value

How to calculate π, A, B from observations?

Two possible scenarios:

1. We are given a dataset of sequences labeled with states…

   …and we have to learn the parameters of the HMM

   - Supervised learning

2. We are given only a collection of sequences…

   …and we have to learn the parameters of the HMM

   - Unsupervised learning
   - Baum–Welch algorithm: a special case of the expectation–maximization (EM) algorithm used to find the unknown parameters of a hidden Markov model (HMM)

Slide based on slides by Vivek Srikumar

Supervised learning of HMM

The maximum likelihood principle

a sequence of words

$$\mathcal{D} = (\boldsymbol{w}^{(\boldsymbol{k})}, \boldsymbol{t}^{(\boldsymbol{k})})_{k=1}^{D}$$

a sequence of tags

$$\text{argmax}_{\pi,A,B} \prod_{k=1}^{D} \mathbb{P}(\boldsymbol{w}^{(\boldsymbol{k})}, \boldsymbol{t}^{(\boldsymbol{k})})$$

$$\approx \text{argmax}_{\pi,A,B} \prod_{k=1}^{D} \prod_{i=1}^{N} B_{t_i^{(k)}, w_i^{(k)}} \cdot \pi_{t_1^{(k)}} \prod_{i=2}^{N} A_{t_{i-1}^{(k)}, t_i^{(k)}}$$

π, A, B can be estimated separately by counting using a *tagged* training corpus

Slide based on slides by Vivek Srikumar

Supervised learning of HMM

$$\mathcal{D} = (\boldsymbol{w}^{(\boldsymbol{k})}, \boldsymbol{t}^{(\boldsymbol{k})})_{k=1}^{D}$$

a special start token

$$\pi_t = \frac{\mathrm{count}(start \rightarrow t)}{|\mathcal{D}|}$$

$$A_{t',t} = \frac{\mathrm{count}(t \rightarrow t')}{\mathrm{count}(t)}$$

$$B_{t,w} = \frac{\mathrm{count}(t \rightarrow w)}{\mathrm{count}(t)}$$

for every possible tag `t` in the taggest and word `w` in the vocabulary

Add small constants to the counts to avoid zero probabilities (smoothing)

Slide based on slides by Vivek Srikumar

How many possible sequences?

The	Fed	raises	interest	rates

Suppose each word allows only the following tags

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

Slide credit: Vivek Srikumar

How many possible sequences?

The	Fed	raises	interest	rates

Suppose each word allows only the following tags

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

In this simple case, $1 \times 2 \times 2 \times 2 \times 2 = 16$ possible sequences exist

Slide credit: Vivek Srikumar

Given an observed sequence, and a model (π, A, B), how to <u>efficiently</u> calculate the most probable state sequence?

$t_1, t_2, \ldots, t_N$ ... a sequence of tags

$w_1, w_2, \ldots, w_N$ ... a sequence of observed words

$$\hat{t}_{1:N} = \text{argmax}_{t_{1:N}} \mathbb{P}(t_1, \ldots, t_N | w_1, \ldots, w_N) \approx \text{argmax}_{t_{1:N}} \prod_{i=1}^{N} B_{t_i, w_i} \cdot \pi_{t_1} \prod_{i=2}^{N} A_{t_{i-1}, t_i}$$

Naïve approaches

1. Try out every sequence and return the highest scoring one
   - Correct, but slow, `O(num-possible-states`$^{\texttt{sequence-length}}$`)`
2. Greedy search
   - The best tag given the previously chosen tag and observed word
   - Incorrect, but fast `O(sequence-length)` $t_i = \arg\max_t \mathbb{P}(t \mid t_{i-1}) P(w_i \mid t)$

Viterbi algorithm: `O(sequence-length x num-possible-states`^{2}`)`

Solution: Use the independence assumptions

$t_1, t_2, \ldots, t_N$... a sequence of tags

$w_1, w_2, \ldots, w_N$... a sequence of observed words

$$\hat{t}_{1:N} = \text{argmax}_{t_{1:N}} \mathbb{P}(t_1, \ldots, t_N | w_1, \ldots, w_N) \approx \text{argmax}_{t_{1:N}} \prod_{i=1}^{N} B_{t_i, w_i} \cdot \pi_{t_1} \prod_{i=2}^{N} A_{t_{i-1}, t_i}$$

Take advantage of the first order Markov assumption

The state for any observation is only influenced by the previous state, the next state and the observation itself

Given the adjacent labels, the others do not matter

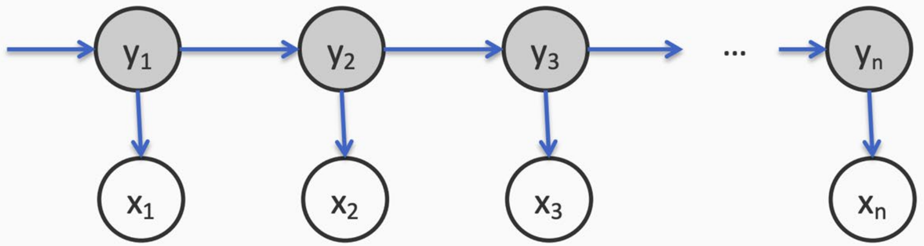
Suggests a recursive algorithm

Slide credit: Vivek Srikumar

Deriving the recursive algorithm

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

What we want: An assignment to all the y_i's that maximizes this product

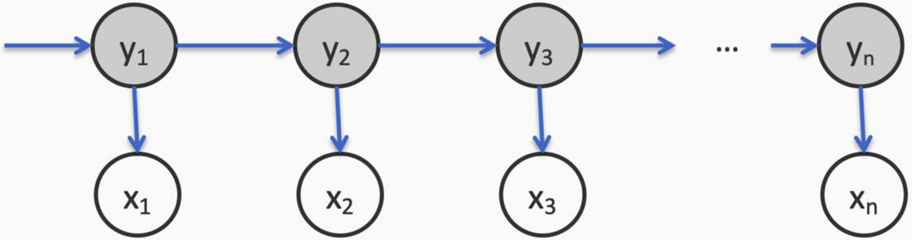


Slide credit: Vivek Srikumar

Deriving the recursive algorithm

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

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

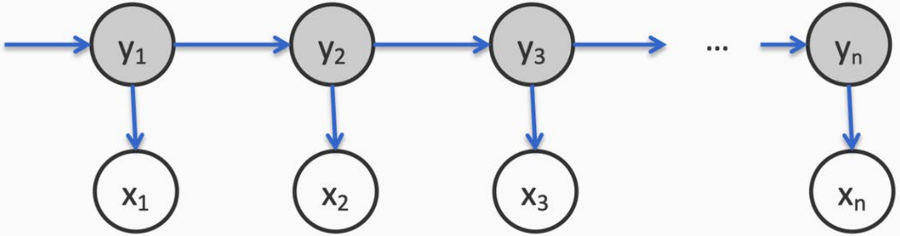


Deriving the recursive algorithm

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

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

Initial probability



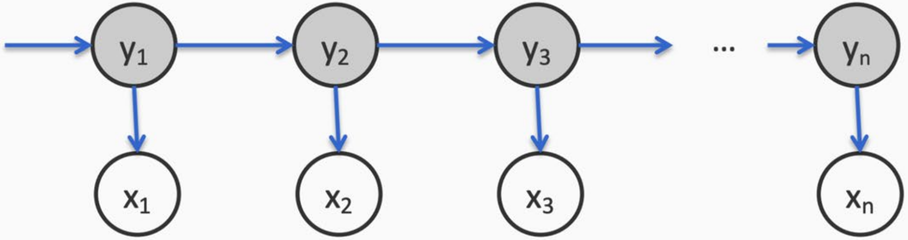
Deriving the recursive algorithm

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

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

Emission probabilities

Initial probability



Deriving the recursive algorithm

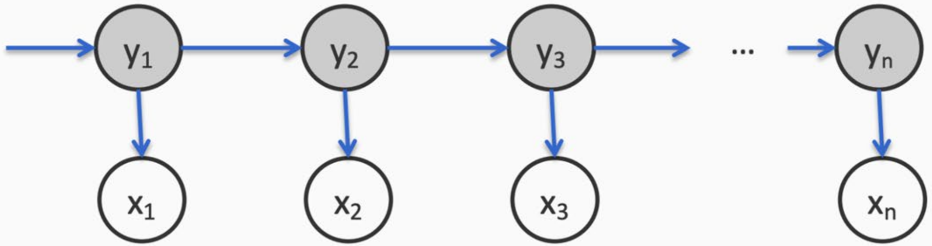
$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

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

Transition probabilities

Emission probabilities

Initial probability



Deriving the recursive algorithm

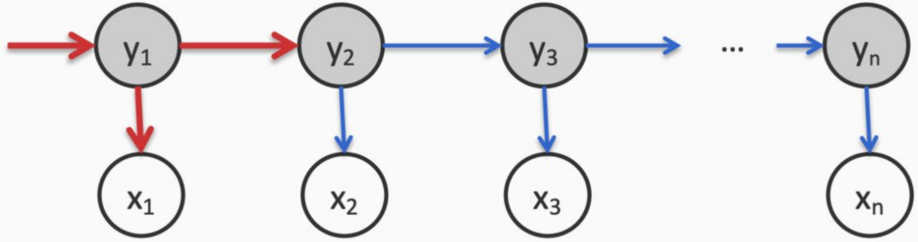
$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

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

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

Only a few factors depend on y_1 so we rearrange the product such that we place all those factors to the right

We can move max_{y_1} to the right too because other terms do not depend on y_1



Deriving the recursive algorithm

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

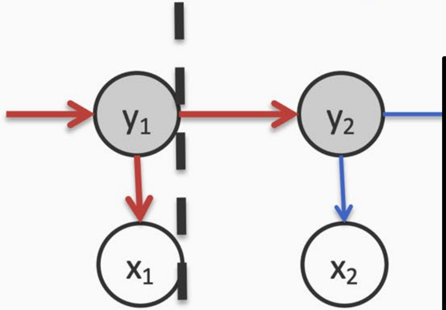
$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

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

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

Abstract away the score for all decisions till here into score_1

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



Abstract away the last two terms into something that we will give a special name `score_1`

`s` is a symbol for any state

Deriving the recursive algorithm

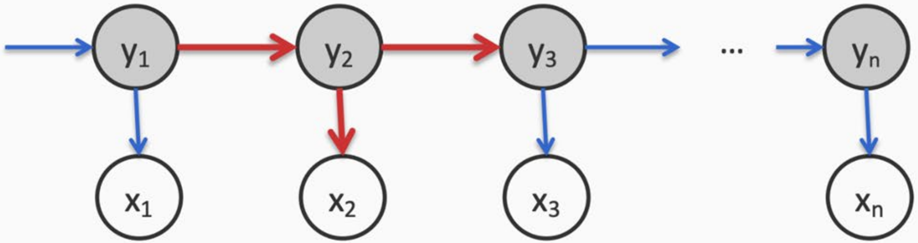
$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

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

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

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

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

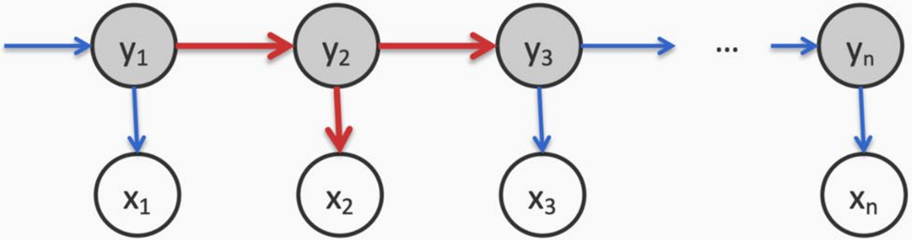


Deriving the recursive algorithm

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

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

Only terms that depend on y_2

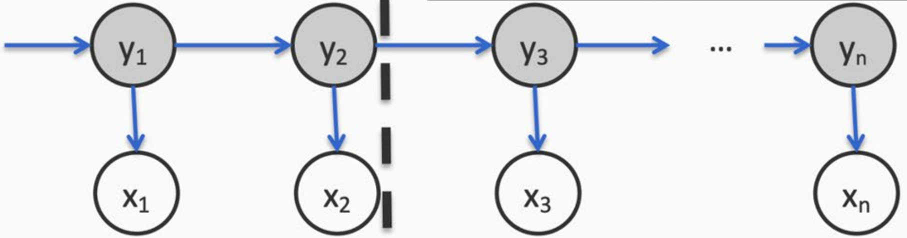


Deriving the recursive algorithm

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

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

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



Abstract away the score for all decisions till here into score

Deriving the recursive algorithm

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

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

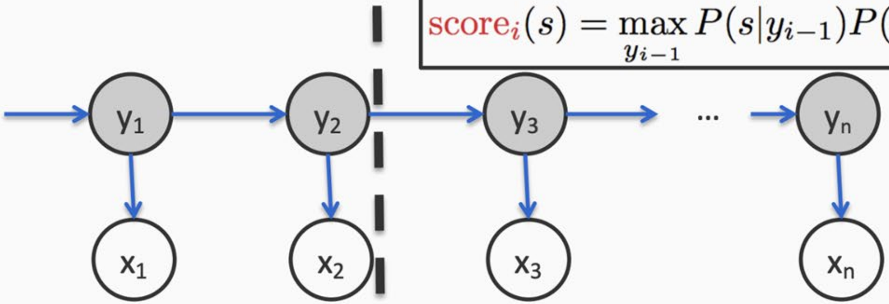
$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

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

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

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

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



Abstract away the score for all decisions till here into score

Deriving the recursive algorithm

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

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

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

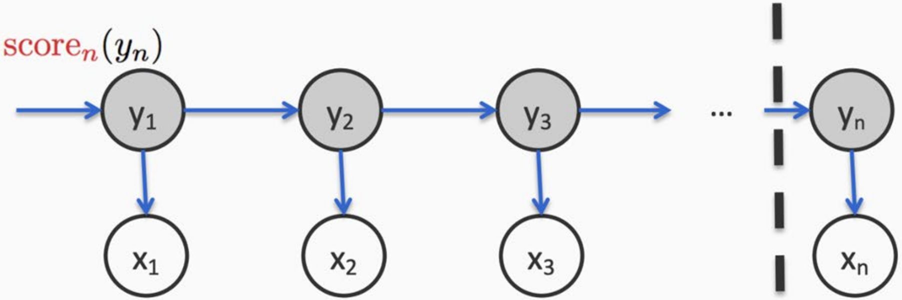
$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n)\cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)\mathrm{score}_1(y_1)$$

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

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

$$\vdots$$

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



Abstract away the score for all decisions till here into score

Deriving the recursive algorithm

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

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

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

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

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

$$\vdots$$

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

Viterbi algorithm

Max-product algorithm for first order sequences

1. Initial: For each state s, calculate

$$score_1(s) = P(s)P(\,x_1 \mid s\,)$$

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

$$score_i(s) = \max_{y_{i-1}} P(\,s \mid y_{i-1}\,)P(\,x_i \mid s\,)score_{i-1}(y_{i-1})$$

3. At the final state: calculate

$$\max_{y_{i-1}} P(\,y, x \mid \pi, A, B\,) = \max_{s} score_n(s)$$

Viterbi algorithm

Max-product algorithm for first order sequences

π: Initial probabilities
A: Transitions
B: Emissions

1. Initial: For each state s, calculate

$$score_1(s) = P(s)P(x_1 \mid s) = \pi_s B_{x_1,s}$$

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

$$score_i(s) = \max_{y_{i-1}} P(s \mid y_{i-1})P(x_i \mid s)score_{i-1}(y_{i-1})$$

$$\boxed{= \max_{y_{i-1}} A_{y_{i-1},s} B_{s,x_i} score_{i-1}(y_{i-1})}$$

3. At the final state: calculate

$$\max_{y_{i-1}} P(y, x \mid \pi, A, B) = \max_{s} score_n(s)$$

Viterbi algorithm

Max-product algorithm for first order sequences

π: Initial probabilities
A: Transitions
B: Emissions

1. Initial: For each state s, calculate

$$score_1(s) = P(s)P(x_1 \mid s) = \pi_s B_{x_1,s}$$

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

$$score_i(s) = \max_{y_{i-1}} P(s \mid y_{i-1})P(x_i \mid s)score_{i-1}(y_{i-1})$$
$$= \max_{y_{i-1}} A_{y_{i-1},s} B_{s,x_i} score_{i-1}(y_{i-1})$$

3. At the final state: calculate

$$\max_{y_{i-1}} P(y, x \mid \pi, A, B) = \max_{s} score_n(s)$$

Runtime complexity:
O(sequence length x #possible states^2)

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

From sequence labeling to syntactic parsing

Syntax: The set of principles under which sequences of words are judged to be grammatically acceptable

- We've already learned one of the most basic syntactic concepts: the syntactic role of each word

🔜 Turning to thinking about word/phrase order

Grammar (informally) is the broader term that encompasses all implicit rules by which speakers intuitively judge which strings are well-formed and what they mean; including syntax, morphology, phonetics (sounds), semantics, and sometimes pragmatics (contextual use of language)

- Different from a **grammar formalism** that provides a set of mathematical rules or algorithms that can be used to generate the syntactic structures of a language

Syntactic parsing: The task of assigning a *syntactic structure* to a sequence of text

- Different theories of grammar propose different formalisms for describing the syntactic structure of sentences:
    - ➔ constituency grammars & dependency grammars

Why do we care?

Getting the **right interpretations of words**:

- *<u>Visiting</u> relatives can be annoying.*
- *<u>Visiting relatives</u> can be annoying.*

Gateway to thinking about recognizing **who is doing what to whom**:

- *The **cat** <u>chased</u> the dog.*

Machine translation from subject-verb-object (SVO) languages like English to verb-subject-object (VSO) languages like Welsh [https://en.wikipedia.org/wiki/Verb%E2%80%93subject%E2%80%93object_word_order]

Grammar checking: Sentences that cannot be parsed may have grammatical errors (or at least be hard to read)

Always useful for chunking text into phrases

Constituency parsing: Intro

Constituency parsing is a method that breaks a sentence down into its constituent parts

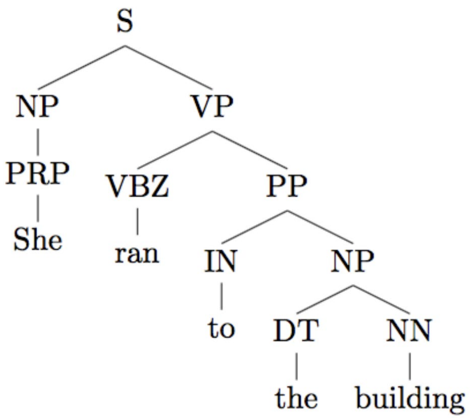
Constituents are words or groups of words that function as a single unit within a hierarchical structure

- **S**entence, **N**oun **P**hrase, **V**erb **P**hrase, **P**repositional **P**hrases
- Bottom layers in POS tags

Constituents are represented in a **parse tree**

- Not a binary tree
- Right branching in English

Constituency makes sense for a lot of languages but not all, e.g., those where the word order is free such as Latin



What’s hard about constituency parsing?

Structural ambiguity: When the grammar can assign more than one parse to a sentence

PP attachment ambiguity:

- *The children ate the cake with a spoon.*
- (S (NP (Det The) (N children)) (VP (V ate) (NP (Det the) (N cake)) (PP (P with) (NP (Det a) (N spoon)))))
- (S (NP (Det The) (N children)) (VP (V ate) (NP (Det the) (N cake) (PP (P with) (NP (Det a) (N spoon))))))
    - Same parse as “The children ate the cake with some icing”

Modifier scope

- *Plastic cup holder*
- (NP (Adj Plastic) (N (N Cup) (N Holder)))
- (NP (N (NP (Adj Plastic) (N Cup)) (N Holder)))

What’s hard about constituency parsing? Cont.

Structural ambiguity: When the grammar can assign more than one parse to a sentence

Complement structure:

- *The students complained to the professor that they didn’t understand.*
- (S
  (NP (Det The) (N students))
  (VP (V complained)
  (PP (P to) (NP (Det the) (N professor)))
  (SBAR (WHNP that) (S (NP they) (VP (V didn't) (VP understand))))))
- (S
  (NP (Det The) (N students))
  (VP (V complained)
  (PP (P to)
  (NP (Det the) (N professor)
  (SBAR (WHNP that) (S (NP they) (VP (V didn't) (VP understand))))))))

What’s hard about constituency parsing? Cont.

Structural ambiguity: When the grammar can assign more than one parse to a sentence

Coordination scope:

- *I saw the man with a telescope and a hat.*
- (S
  (NP (I))
  (VP (V saw)
  (NP (Det the) (N man)
  (PP (P with)
  (NP (NP (Det a) (N telescope))
  (CC and)
  (NP (Det a) (N hat))))))))

- (S
  (NP (I))
  (VP (V saw)
  (NP (NP (Det the) (N man)
  (PP (P with)
  (NP (Det a) (N telescope))))
  (CC and)
  (NP (Det a) (N hat)))))

Let’s parse!

Given a sentence, how do we find the highest scoring parse tree for it?

We’ll apply the **CKY algorithm** to *Probabilistic* **Context-Free Grammars**

Goals of Next Lecture

Learn how to produce a constituency parse using an non-neural algorithm

- Intro
- **Context-Free Grammars (CFGs)**
- Probabilistic CFGs
- CKY Algorithm
- Evaluation