CS 5034/6134 Natural Language Processing
Fall 2024

Lecture 2: What is NLP?

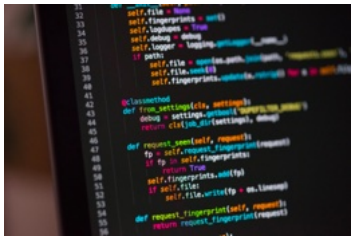
University of CINCINNATI

What is NLP?

How do we communicate with computers?

How we imagine talking to computers

How do we communicate with computers?



How we actually talk to computers

How do we communicate with computers?

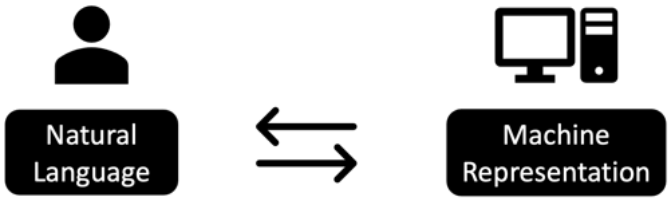
One of the ultimate goals

What is natural language processing (NLP)?

The process of building computational models for understanding natural language.

Sometimes also called natural language understanding and computational linguistics.

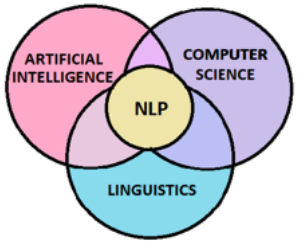
What is natural language processing (NLP)?



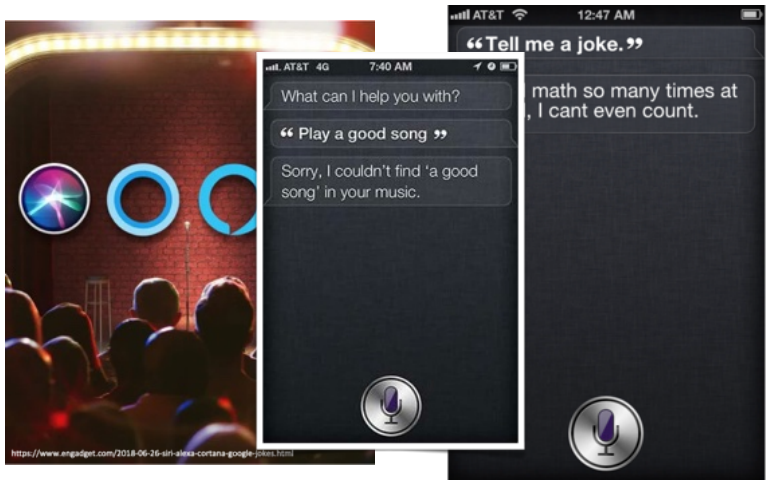
We want a program that can understand and reason about the world via language.

What is natural language processing (NLP)?

Natural Language Processing (NLP) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language.

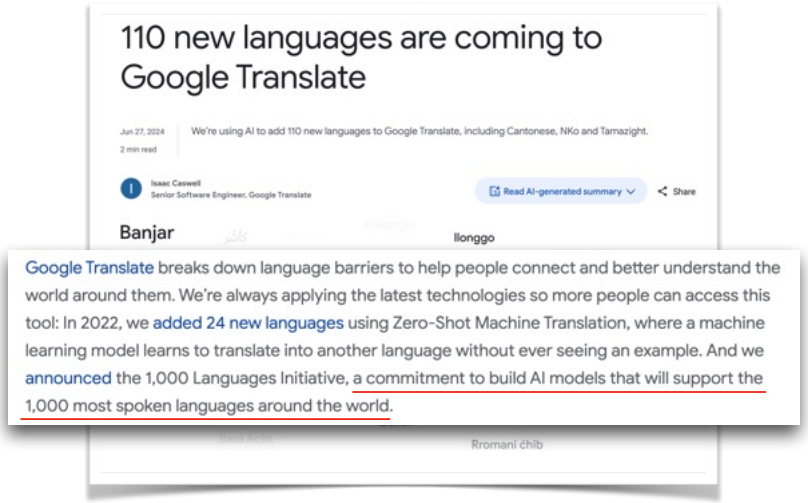


Virtual Assistants

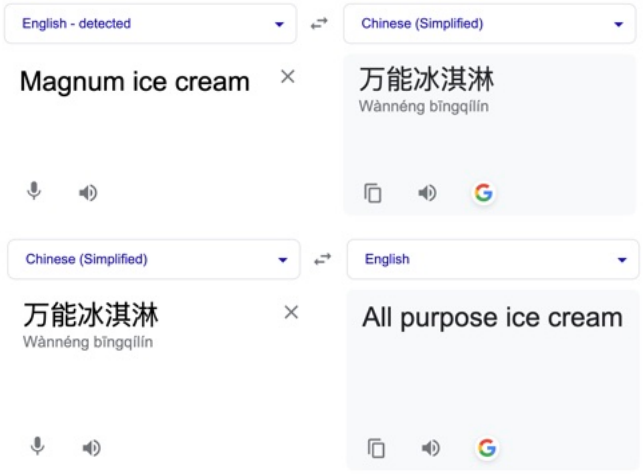


Machine Translation

Machine Translation

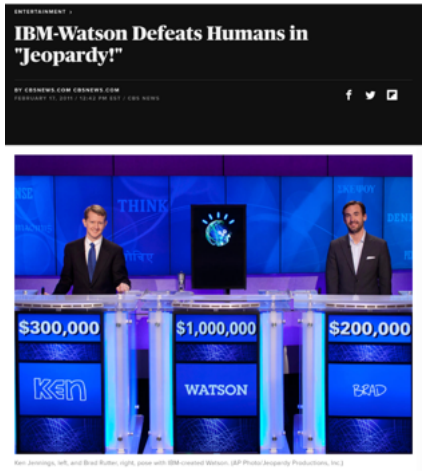


Machine Translation



Google Translate result in 2023-12

Question Answering



> Watson, specifically, is a "question answering machine" of a type that artificial intelligence researchers have struggled with for decades — a computer akin to the one on "Star Trek" that can understand questions posed in natural language and answer them.

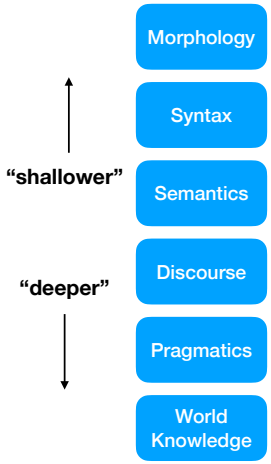
https://www.nytimes.com/2011/02/17/science/17jeopardy-watson.html

Other Applications

- Text Classification
- Information Extraction
- Event Extraction
- Text Summarization
- Sentiment Analysis
- Integrating Language and Vision
- ……

Levels of Analysis and Knowledge

- **Morphology**: how words are constructed; prefixes & suffixes
- **Syntax**: structural relationships between words in a sentence.
- **Semantics**: meanings of words, phrases, and expressions
- **Discourse**: relationships across different sentences or thoughts; contextual effects
- **Pragmatics**: the purpose of a statement; how we use language to communicate
- **World Knowledge**: facts about the world; common sense



Morphology

一隻憂鬱的小烏龜在孤獨的蕩鞦韆

様々な著書を通して、多くの人に数学を広める上でも貢献した。

I went to Mr. Shabu, the best place for hot pot.

Some languages like Chinese and Japanese don't have spaces between words.

Even in English, this cannot be done deterministically.

Morphology

- kick, kicks, kicked, kicking
- sit, sits, sat, sitting
- murder, murders

But it's not as simple as naively adding and deleting endings...

- gorge, gorgeous
- glass, glasses
- arm, army

Syntax: Part-of-Speech Tagging

Tars tears the door off the plane.

Verb, adjective, or noun?

tear

- verb: tear it apart
- noun: wipe the tear

plane

- verb: plane the surface
- adjective: plane surface
- noun: travel by plane

Syntax: Part-of-Speech Tagging

Tars tears the door off the plane.

NOUN VERB ART NOUN PREP ART NOUN

Syntax: Structural Ambiguity

Time flies like an arrow.

Metaphor:

Time/**NOUN** flies/**VERB** like/**PREP** an/**ART** arrow/**NOUN**

New Fly Species:

Time/**NOUN** flies/**NOUN** like/**VERB** an/**ART** arrow/**NOUN**

Stopwatch Imperative:

Time/**VERB** flies/**NOUN** like/**PREP** an/**ART** arrow/**NOUN**

Syntax: Structural Ambiguity

- I saw the Rocky Mountains flying to Seattle.
- I watered the plant with yellow leaves.
- I saw the man on the hill with the telescope.

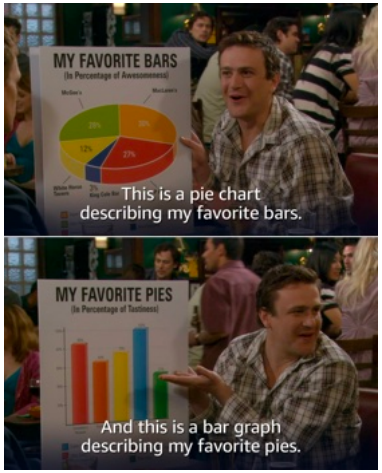
But syntax doesn't tell us much about meaning…

- Blue green ant lifted pink yellow elephant.
- fire … match … arson … hotel
- plastic cat food can cover

Semantics: Lexical Ambiguity

- What does "shot" mean?
- I walked to the bank …
  - of the river.
  - to get money.
- The bug in the room ...
  - was planted by spies.
  - flew out the window.
- I work for John Hancock ...
  - and he is a good boss.
  - which is a good company

Semantics: Word Sense Ambiguity



How I Met Your Mother, Season 4, Episode 22

Semantics: Frame Semantic Parsing

Cooking_creation

Definition: This frame describes food and meal preparation. A **Cook** creates a **Produced_food** from (raw) **Ingredients**. The **Heating_instrument** and/or the **Container** may also be specified.

Caitlin BAKED **some cookies** from the **pre-packaged dough**.

Discourse: Coreference

Terry Horton has been a landlord in Cincinnati for more than a decade… Mr. Horton began eyeing a new vacant property in the South Cumminsville neighborhood… The landlord says an appraiser, who is white, used unfair comparisons to assess the worth of his apartment building.

https://www.nytimes.com/2023/03/18/realestate/appraisal-racial-discrimination-cincinnati.html

Discourse: Coreference

Terry Horton has been a landlord in Cincinnati for more than a decade… **Mr. Horton** began eyeing a new vacant property in the South Cumminsville neighborhood… **The landlord** says an appraiser, who is white, used unfair comparisons to assess the worth of **his** apartment building.

Sometimes, it requires world knowledge to make the inference. We will see that later.

Pragmatics: Intentions of Utterances

Rules of Conversation

- Can you pass me the paper?
- Do you have a watch?

Speech Acts

- Will you marry me?
- I bet you $50 that the Lakers will win tonight.
- Excuse me.

Pragmatics: Intentions of Utterances

On a very hot day, I said in class in Japan:

Me: How hot!

Student: It is really hot! (attention to the semantic meaning)

In Brazil, the reaction was different:

Me: How hot!

Student: Do you want me to open the door? (attending to the pragmatic meaning)

De Oliveira, Ulisses Tadeu Vaz, Sumiko Nishitani Ikeda, and Marcelo Saparas. "EFFECTIVE WAYS OF TEACHING PRAGMATICS: HUMOR IN THE CLASSROOM." *PragMATIZES-Revista Latino-Americana de Estudos em Cultura* (2019): 225-237.

Pragmatics Humor

https://7452131672170885 68.weebly.com/pragmatics--discourse.html

Pragmatics Humor

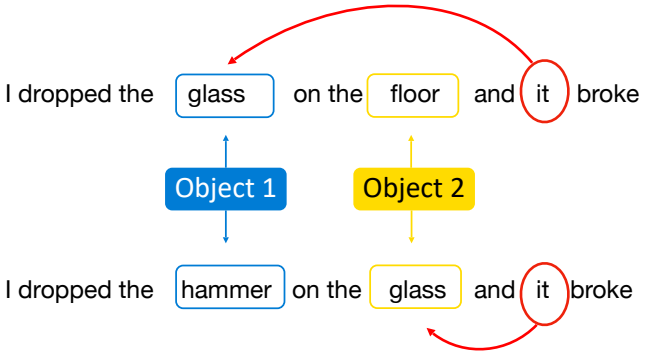
https://www.eatliver.com/coffee-or-tea/

World Knowledge

Explicitly Expressed

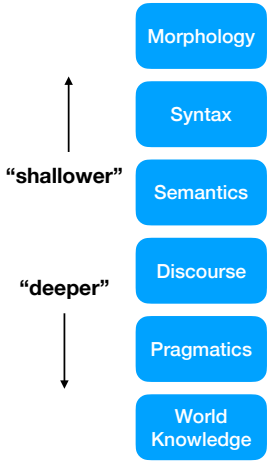
Implicitly Expressed

World Knowledge



Levels of Analysis and Knowledge

- **Morphology**: how words are constructed; prefixes & suffixes
- **Syntax**: structural relationships between words in a sentence.
- **Semantics**: meanings of words, phrases, and expressions
- **Discourse**: relationships across different sentences or thoughts; contextual effects
- **Pragmatics**: the purpose of a statement; how we use language to communicate
- **World Knowledge**: facts about the world; common sense



Why NLP is Hard: Ambiguity

So far, we have shown natural languages can have different levels of knowledge, which also brings different levels of ambiguity:

- **Structural ambiguity**

  "I eat sushi with chopsticks" vs "I eat sushi with wasabi"

- **Lexical (word sense) ambiguity**

  "The market is active". Financial market or physical market?

- **Referential ambiguity**

  "John saw Terry. But he walked away." Who walked away?

Why NLP is Hard: Ambiguity

So far, we have shown natural languages can have different levels of knowledge, which also brings different levels of ambiguity:

- **Phonological ambiguity**

  knight, night

  meat, meet

  son, sun

  for, four

I want to gamble.	wǒ 我	xiǎng 想	yào 要	dǔ 赌	bó 博
I want to pursue a Ph.D.	wǒ 我	xiǎng 想	yào 要	dú 读	bó 博

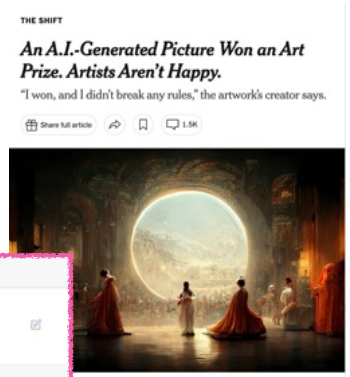
Dealing with Ambiguity

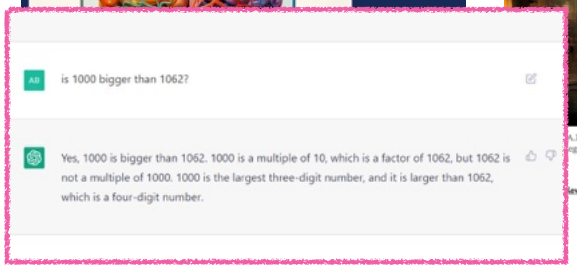
Statistical models are one of the main methods to handle ambiguity in NLP tasks.

These models need to be trained (estimated, learned) before they can be tested (used, evaluated).

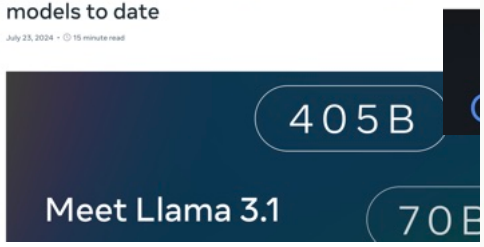
Where are we now?

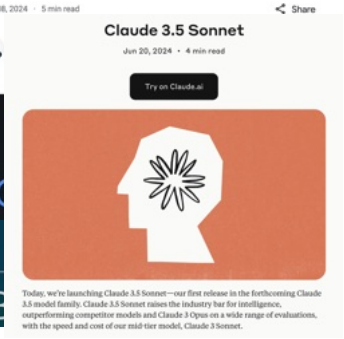
Where are we now?





Where are we now?





CS 5034/6134 Natural Language Processing
Fall 2024

Lecture 3: Corpus and Morphology

University of CINCINNATI

Traditional NLP Pipeline

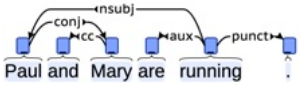
- **Tokenizer/Segmenter**
  to identify words and sentences
- **Morphological analyzer/POS-tagger**
  to identify the part of speech and structure of words
- **Word sense disambiguation**
  to identify the meaning of words
- **Syntactic/semantic Parser**
  to obtain the structure and meaning of sentences
- **Coreference resolution**
  to keep track of the various entities mentioned

Traditional NLP Pipeline

- **Representation:**
  Each step in the NLP pipeline requires its own representations
  Part-of-speech tagging requires a POS tag set

NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural

  Dependency parsing requires dependency labels



Traditional NLP Pipeline

- **Representation:**
  - Each step in the NLP pipeline requires its own representations.
  - The representations should capture linguistically appropriate generalizations or abstractions.
  - They often require linguistic expertise for designing.
  - Sometimes it is not easy to tell if a representation is good or not.

Traditional NLP Pipeline

→ Apply a condition & learn from it.

- **Modeling:**
  - Each step in the NLP pipeline relies on a model (rule-based or statistical) to generate the most appropriate representations.
    → different from representation.
  - Annotation is time-consuming and expensive (and they are not perfect either!)
  - The models are not perfect. The errors can accumulate at each step.

Traditional NLP Pipeline Outdated?

- Many current neural methods are “end to end”
- With “large” amounts of training data, they can often outperform traditional pipelines.
- But
  - What if we don’t have enough data?
  - How can we incorporate explicit knowledge, reasoning into these models?
  - Interpretability?

Corpus

- A corpus (plural: corpora) is a collection of natural language data.
- To understand and model how language works, we need empirical evidence. Ideally, naturally-occurring corpora serve as realistic samples of a language.
  → large chunk of human-made sentences / data.
- **Raw corpora:** raw text with minimal or no processing.
- **Annotated corpora:** humans have read the text and marked categories or linguistic structures describing their syntax and/or meaning.

Corpus Information

- **Basic statistics:** corpus size, word frequency
- **Mode of communication:** speech, writing
- **Genre:** news, fiction, poetry, tweet
- **Topic:** politics, sports, music
- **How was the text produced:** web crawls, real meetings recorded
- **Intellectual property:** copyright, licenses
- **Annotation process:** annotation guidelines, quality assessment

Corpus Selection

- It is a complicated decision and depends on many factors.
  - What is your task?
  - What do you want your system to achieve?
- Statistical approaches typically assume that training and test data are sampled from the same distribution.
- Your corpus selection determines your system's working "domain".

↓

Make sure training data & test data sets are similar to achieve a higher performance.

Why do we need corpora?

- To evaluate our systems

  Good science requires controlled experimentation.

  Good engineering requires benchmarks.

- To improve our systems

  Unsupervised learning, supervised learning, semi-supervised learning…

What are the first things to do when we have a raw corpus?

Text Normalization

- Transforming text data into a standardized or canonical form.
- One of the data preprocessing steps before using a corpus.
- It makes the data clean, and more convenient to use.

Tokenization

- When processing text in English, we typically identify word boundaries ("word segmentation") based on white space and punctuation marks.

  A word is any sequence of alphabetical characters between whitespaces that's not a punctuation mark?

- This works to a first approximation, but what about:
  - Abbreviations: Mr. D.C.
  - Contractions: did n't would n't should n't

    cases like can't might not be split, or might be split into can 't
  - Complex names: New York
  - Other languages that have no whitespace
- Sometimes numbers and other special symbols (e.g., $, emoticons, hashtags) are specially tokenized as well.

Punctuation Matters

Sentence Splitting

- After tokenization, documents are often split into sentences.
- Recognizing sentence boundaries is surprisingly non-trivial!
  - I went to **Mr. Shabu**, the best place for hot pot.
  - She met her son's teacher, **Mr.** Hal **K.** Jordan.
  - He sold XYZ **Inc.** for **$50.2** million on **Sept.** 2, 2020.
  - He was infected with **E. coli** and **C. difficile**.
- Recommendation: do not try to write your own sentence splitter! A simple regex rarely performs well — download one that others have taken time to develop.

NLTK

- The Natural Language Toolkit (https://nltk.org) is a Python library for NLP.
  - Open-source, community-built
  - Wrappers for many corpora and lexical resources
  - Easy-to-use interface
  - Might not be the state-of-the-art, but good for normal purposes

```
>>> import nltk
>>> sentence = """At eight o'clock on Thursday morning
... Arthur didn't feel very good."""
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning',
'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']
```

spaCy

- The spaCy (https://spacy.io) is another widely used Python library for NLP.
  - Open-source
  - Easy-to-use interface
  - Neural based models, takes more space and time!

Morphology

- After tokenization, another common following step is normalizing word formats, which requires the knowledge of morphology.
- Morphology is concerned with the construction of words and the meaning of their parts.
- A morpheme is the smallest meaning-bearing unit of a word.
- Words are composed of stems and affixes (prefix and suffix).

Morphemes: stems, affixes

dis-count-able
prefix-stem-suffix

- Stems are often **free** morphemes.

  (free morphemes can occur by themselves as words)

- Affixes are usually **bound** morphemes.

  (bound morphemes have to combine with others to form words)

Inflectional vs. Derivational

- **Inflectional:** variants that have essentially the same meaning and the same general part-of-speech.

  E.g., kick, kicks, kicked, kicking

  book, books

- **Derivational:** variants that represent a different concept. The part-of-speech may be the same or different.

  E.g., friend, friendly, friendliness, friendship

  grace, disgrace, disgraceful, disgracefully

Examples

Root	Morphological Variants
walk	walks, walked, walking (I)
noise	noisy, noisily (D)
order	reorder, orderly (D)
elect	reelect, reelection (D)
view	preview, previewer, previewers (D)

Inflectional vs. Derivational?

Word Normalization

- Reducing a word to a standardized "root" form is useful for word normalization.

  E.g., *hire, hires, hired,* and *hiring* should represent the same concept for tasks like information retrieval and question answering.

- **Lemma** is the linguistic term for a word's root form in, and **lemmatization** is the process of identifying a word's root.

- **Case folding** is another kind of word normalization.

  E.g., *Pushing* and *pushing* should be treated as the same term.

  But this can cause problems too,

  E.g., *Apple* (company) and *apple* (fruit) are not the same thing.

Blindly stripping affixes can produce strange results…

preempt	→	empt
news	→	new
pretend	→	tend
hardly	→	hard
glasses	→	glass
Mrs	→	Mr

Morphology Humor

Wow, you read one book on morphology and then spend your time walking around wondering why it is possible to **blacken**, **whiten**, or **redden** something but not **yellowen**, **bluen**, or **orangen** it.

Why Word Normalization?

- Suppose you want to search the news for Kennedy's assassination.
- What keywords should you use?

  assassination, assassinate, assassinating?

Inverted Files

- Traditional information retrieval (IR) systems use an inverted file data structure to represent the documents in a text collection.
- An inverted file is an index of normalized words (typically stems) paired with a set of documents that contain the word.

  E.g.,

assassinat	(doc1, doc4, doc35, doc56, …)
conspire	(doc3, doc50, doc90, …)
murder	(doc3, doc7, doc36, …)
Kennedy	(doc24, doc27, doc29, …)

- An IR system can simply collect the document sets for query terms and apply Boolean operators to match the query (e.g., AND).

Proximity Information

- Information about the location of words in each document can also be stored to allow for phrasal matching.

  E.g., doc8 contains an instance of "hot dog"

hot	(doc8 [99], doc55 [6, 101], …)
dog	(doc8 [100], doc98 [12], …)

- Adding proximity information dramatically increases the size of the inverted file.
- Extremely common words can blow up the size of the inverted file (e.g., imagine indexing every instance of "the" !). For this reason, **stopwords** (typically closed class function words) often are not indexed.

Stemming

- Lemmatization algorithms can be complex. For this reason we sometimes make use of a simpler but cruder method.
- **Stemming** is used to approximately recognize word variants in documents. Stemming is widely applied in information retrieval.
- Most stemmers just chop off common prefixes and suffixes using heuristics.

assassination	→	assasinat + ion
assassinations	→	assasinat + ions
assassinate	→	assasinat + e
assassinates	→	assasinat + es
assassinated	→	assasinat + ed
assassinating	→	assasinat + ing

The Porter Stemmer

- The Porter Stemmer is a widely used stemming algorithm that uses a cascaded set of rewrite rules. These rules simply chop off and add characters without considering parts-of-speech or a dictionary of known root forms.
- Sample suffix rules:

Rule	Example
ational → ate	relational → relate
ing → (empty)	motoring → motor
sses → ss	grasses → grass

- More details can be found here.

Stemming in Action

Original Text

This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things-names and heights and soundings-with the single exception of the red crosses and the written notes.

Stemmed Text

Thi wa not the map we found in Billi Bone s chest but an accur copi complet in all thing name and height and sound with the singl except of the red cross and the written note

Limitations of Stemming

- Simple stemmers can be useful in cases where we need to collapse across different variants of the same lemma. Nonetheless, they do tend to commit errors of both over- and under-generalizing.

Errors of Commission	**Errors of Omission**
organization->organ	European/Europe
doing->doe	analyzes/analysis
numerical->numerous	noisy/noise
policy->police	sparsity/sparse

(Krovetz, 1993)

Morphological Analysis as Search

- Input: word w, dictionary, and morphology rules
- Process:

  `IF` w is in the dictionary, return its definition

  `ELSE` apply all of the rules exhaustively and return all derivations of w

  For each rule, strip off the affix and add the replacement chars to produce a candidate root. If the candidate root is in the dictionary with the appropriate POS, then success!
  - If success, return the derived word with the POS and properties assigned by the rule.
  - If failure, then recursively apply the rule set to the candidate root to see if it can be derived.

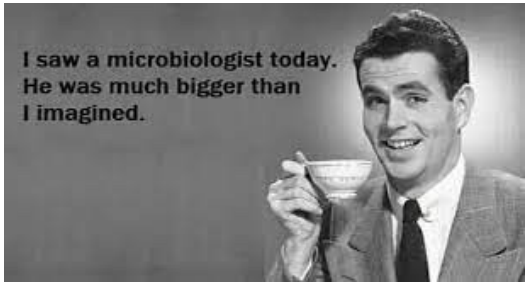
Morphological Rule Examples

IF ⟶ **THEN**

Rule ID	Prefix	Suffix	Replace Chars	Root POS	Derived POS	Prop-erties
1		s		noun	noun	plural
2		s		verb	verb	present tense
3		ed		verb	verb	past tense
4		ed		verb	adj	
5		y		noun	adj	
6		y	e	noun	adj	
7	un			adj	adj	
8	anti			noun	adj	

Derived Word	Root
cats	cat
sings	sing
armed	arm
armed	arm
hairy	hair
noisy	noise
unfair	fair
antiwar	war

Derivation Order Matters!



Derivation #1
micro- & biology → microbiology
microbiology & -ist → microbiologist

Derivation #2
biology & -ist → biologist
micro- & biology → microbiologist

Exhaustive Depth-first Search

Rules

Rule ID	Prefix	Suffix	Replace Chars	Root POS	Derived POS
1		ed		verb	adj
2		ed		verb	verb
3	re			verb	verb

Dictionary

view (noun)
view (verb)

Derivations for "reviewed"?

Exhaustive Depth-first Search

Rules

Rule ID	Prefix	Suffix	Replace Chars	Root POS	Derived POS
R1		ed		verb	adj
R2		ed		verb	verb
R3	re			verb	verb

Dictionary

view (noun)
view (verb)

Derivations

R1 + R3 → adj
R2 + R3 → verb
R3 + R2 → verb

reviewed (adj)	R1	review (verb)	R1	×	
			R2	×	
			R3	view (verb)	✓
reviewed (verb)	R2	review (verb)	R1	×	
			R2	×	
			R3	view (verb)	✓
reviewed (verb)	R3	viewed (verb)	R1	×	
			R2	view (verb)	✓
			R3	×	

Text Normalization: Summary

Before almost any natural language processing of a text, the text needs to be normalized. We have covered three common steps:

- Tokenizing words
- Normalizing word formats
- Segmenting sentences

CS 5034/6134 Natural Language Processing
Fall 2024

Lecture 4:
N-gram Language Models

University of CINCINNATI

Language Models

- How will a machine know what is a plausible language?

  我like自然XXXYUYAN$%^PROcessing^_^

- How to distinguish between word salad (random words) and normal sentences?

  “lamb apple water marry” vs. “mary had a little lamb”

- How to automatically correct spelling errors or grammatical mistakes?

  “fantastci” vs. “fantastic”

- How to generate human-like languages?

  I love natural language ___

Language Models

Language models define probability distributions over the sequences of words in a language.

Example: Imagine a spelling checker, if it knows

Prob(“fantastci”) << Prob(“fantastic”)

Then it will make a correction suggestion.

The question is, how to define these probabilities?

Probability Theory Recap

- Sample Space

  The sample space of a random experiment is defined as the set of all possible outcomes of an experiment.

- Sample point

  A sample point is an element of a sample space.

- Event

  An event is a subset of the sample space.

Probability Theory Recap

Think about tossing a coin, with two possible outcomes: heads (H) and tails (T).

- If I toss it once, what is the sample space?
  - {H, T}
- If I toss it twice, what is the sample space?
  - {HH, HT, TH, TT}
- If I toss it twice, what is the event that I have different outcomes?
  - {HT, TH}

Probability Theory Recap

Probability of an event is the sum of the probabilities of the individual sample point of which it is composed.

If I toss a "fair" coin twice, what is the probability that I have different outcomes?

Sample space: {HH, HT, TH, TT}

Event A: {HT, TH}

Prob(A) = Prob(HT) + Prob(TH) = 1/4 + 1/4 = 1/2

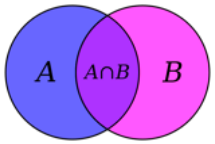
Probability Theory Recap

- Joint Probability

$P(A \cap B)$ sometimes also written as $P(A, B)$

- Combining events

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$



- Conditional Probability

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Conditional Probability

- Conditional Probability

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Suppose I flip a "fair" coin twice. Given that at least one head was obtained, what is the probability of obtaining two heads?

Sample space: {HH, HT, TH, TT}

Event A - two heads: {HH}

Event B - at least one head: {HT, HH, TH}

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{1/4}{3/4} = \frac{1}{3}$$

Chain Rule

The joint probability $P(A, B)$ can also be expressed in terms of the conditional probability $P(A \mid B)$:

$$P(A, B) = P(A \mid B)P(B)$$

Generalizing this to N joint events in a general form is called the **chain rule**:

$$P(A_1, A_2, \ldots, A_n) = P(A_1)P(A_2 \mid A_1)P(A_3 \mid A_1, A_2) \ldots P(A_n \mid A_1, \ldots A_{n-1})$$
$$= \prod_{i=1}^{n} P(A_i \mid A_1, \ldots, A_{i-1})$$

Independence

Two events A and B are independent iff

$$P(A \mid B) = P(A)$$

By substituting it in the joint probability:

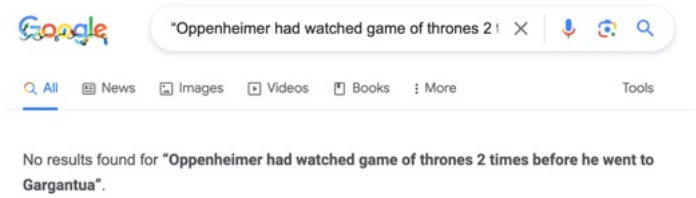
$$P(A, B) = P(A \mid B)P(B) = P(A)P(B)$$

Chain rule for n independent events:

$$P(A_1, A_2, \ldots, A_n) = P(A_1)P(A_2 \mid A_1)P(A_3 \mid A_1, A_2) \ldots P(A_n \mid A_1, \ldots A_{n-1})$$
$$= \prod_{i=1}^{n} P(A_i)$$

Language Models in NLP

- In many scenarios, it will be difficult to know how likely a sentence is to occur in “natural language”, like English.



Language Models in NLP

- But we can define a language model that give us good approximations.
- N-gram models are the simplest and most common kind of language model.
- Let’s see how these models are defined, how to learn their parameters and what they can do.

N-grams

An n-gram is a word sequence of length n.

1-gram or **unigram**

2-gram or **bigram**

3-gram or **trigram**

Mary had a little lamb

unigrams: Mary, had, a, little, lamb

bigrams: Mary had, had a, a little, little lamb

trigrams: Mary had a, had a little, a little lamb

N-gram Language Model

Assume we have some **training data**: a large corpus of general English text.

The set of all words in this corpus is called its **vocabulary**.

We want to have a language model over the vocabulary V that estimate the probability of any given sequence.

How to compute the probability of a sequence of words "$w_1\ w_2\ \ldots\ w_n$"?

What is the most intuitive way?

Using Frequency Counts

We estimate the probability of an N-gram using the training data.

$$P(w_1, \ldots, w_n) = \frac{C(w_1, \ldots, w_n)}{\#\ of\ all\ sentences}$$

where $C(x)$ is the count of x in our text corpus, the denominator is the total number of sentences in the text corpus.

In the previous example, we have

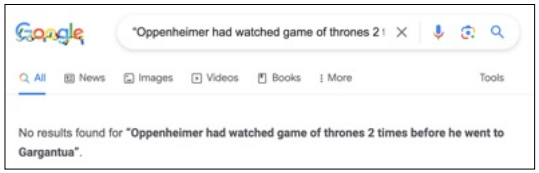
$$P(Mary\ had\ a\ little\ lamb) = \frac{C(Mary\ had\ a\ little\ lamb)}{\#\ of\ all\ sentences}$$

N-gram Language Model

VS.

N-gram Language Model

- What if the text corpus does not contain this sentence, i.e., $C(x) = 0$?



- Recall the **chain rule**:

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_1, \ldots, w_{i-1})$$

- Yet still, many of these conditional probabilities can still be zero! (what is the last term?) => P(n | all previous terms)

Independence Assumption

So we make an independence assumption: the probability of a word only depends on the most recent past words.

unigram model: $P(w_i \mid w_1, \ldots, w_{i-1}) \approx P(w_i)$

bigram model: $P(w_i \mid w_1, \ldots, w_{i-1}) \approx P(w_i \mid w_{i-1})$

trigram model: $P(w_i \mid w_1, \ldots, w_{i-1}) \approx P(w_i \mid w_{i-2}, w_{i-1})$

This is also called Markov assumption.

Independence Assumption

So now we have:

unigram model: $P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i)$

bigram model: $P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-1})$

trigram model: $P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-2}, w_{i-1})$

N-gram Language Model Example

Mary had a little lamb

unigram model: P = P(Mary)*P(had)*P(a)*P(little)*P(lamb)

bigram model: P = P(Mary)*P(had | Mary)*P(a | had)*P(little | a)*P(lamb|little)

trigram model: P = P(Mary)*P(had | Mary)*P(a | Mary, had)*P(little | had, a)*P(lamb | a, little)

How to get these conditional probabilities?

Computing N-gram Probabilities

For the unigram probability,

In our example, we estimate

$$P(lamb) = \frac{C(lamb)}{N}$$

where $C(x)$ is the count of x in our text corpus, $N = \sum_{x'} C(x')$ is the total number of items in the dataset.

More generally, for any unigram, we have

$$P(w_i) = \frac{C(w_i)}{\sum_{x'} C(x')}$$

Computing N-gram Probabilities

For the conditional probability,

In our example, we estimate

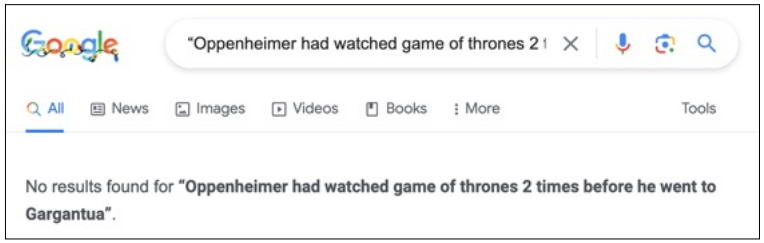
$$P(lamb \mid a, little) = \frac{C(a, little, lamb)}{C(a, little)}$$

where $C(x)$ is the count of x in our text corpus.

More generally, for any trigram, we have

$$P(w_i \mid w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})}$$

Computing N-gram Probabilities



Now to estimate the probability of this sentence, we only need to find out the probabilities of a set of n-grams. If n=2, then we need:

P(Oppenheimer), P(had), P(watched), …

P(Oppenheimer, had), P(had, watched), P(watched, game), …

Practical Details 1: Beginning/End of Sequence

Trigram model assumes two word history:

$$P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2 \mid w_1)\prod_{i=3}^{n} P(w_i \mid w_{i-2}, w_{i-1})$$

But consider this pair:

	w_1	w_2	w_3	w_4
(1)	Mary	had	a	little
(2)	had	a	little	lamb

What is wrong?

Practical Details 1: Beginning/End of Sequence

To capture behavior at beginning/end of sequences, we manually define a beginning and end pseudo word:

	w_{-1}	w_0	w_1	w_2	w_3	w_4	w_5
(1)	<s>	<s>	Mary	had	a	little	</s>
(2)	<s>	<s>	had	a	little	lamb	</s>

Now P(had | <s>, <s>) and P(</s> | a, little) are low, so we know they are not good sentences.

Our equation becomes

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n+1} P(w_i \mid w_{i-2}, w_{i-1})$$

Practical Details 2: Log Probability

- Word probabilities are typically very small.
- The longer the sentence, the smaller the probability will become (more probabilities multiply together).
- Multiplying lots of small numbers will lead to numerical underflow, even using double precision floating point.
- So we use log space instead of linear space.

$$p_1 * p_2 * p_3 * p_4 = e^{\log p_1 + \log p_2 + \log p_3 + \log p_4}$$

just compute $\rightarrow \log p_1 + \log p_2 + \log p_3 + \log p_4$

Practical Details 3: Unknown Words

- What about words we have never seen before (not in training set but pops up in test set)?
- Imagine we pick the corpus as the collection of Shakespeares's plays. Now we want to compute the probability for the sentence:

  "I play computer games"

  However, "computer" does not exist in Shakespeare's works!
- We call them **unknown words**, or out of vocabulary (**OOV**) words.
- To prevent zero probability, we can add a pseudo **<UNK>** word to represent any unknown words in the test set.

Practical Details 3: Unknown Words

- How to define the probabilities of <UNK> words?

  1. Select a prior vocabulary, anything in the training corpus but not in the vocabulary are considered as <UNK>

     OR

  2. Set a frequency threshold, anything in the training corpus below this threshold are considered as <UNK>

Practical Details 4: Smoothing

- Estimating probabilities works well when you have a lot of data. But no matter how "big" a corpus is, there will always be rare words.

  For example: the Brown corpus contains about 1 million words. But it contains only 49,000 unique words. And over 40,000 of those words occur <= 5 times!

- A particularly difficult problem is when word combinations **never** appears in the training data.

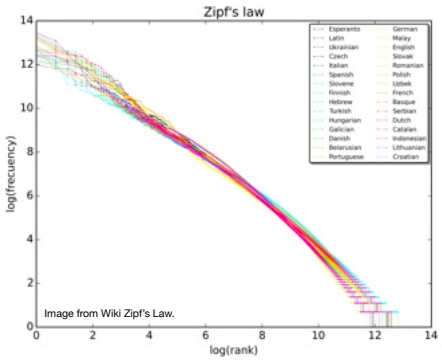
- How do we prevent zero probability in this case?

Zipf's Law

- How "common" are common words and how "rare" are rare words?

- For example, in the Brown Corpus,
  - the word "the" is the most common word, which accounts for nearly 7% of all word occurrences (69,971 out of 1 million).
  - the word "of" is the second most frequent word, about 3.5% of words (36,411)
  - third rank "and" (28,852)

Zipf's Law



$$\text{word frequency} \propto \frac{1}{\text{word rank}^{\alpha}}$$

Zipf's Law

- A corpus will have a number of **common** words. We see them often enough to know (almost) anything about them.

- Regardless of how large our corpus is, there will be a lot of **rare** words and **unknown** words.

- This means we need to find clever ways to estimate probabilities for things we have rarely or never seen.

Zipf's Law in other Contexts

Heaps' Law

- What is the relation between corpus size and vocabulary size?
- A corpus of N tokens has a vocabulary size:

$$|V| \propto N^{\beta}$$

where $0 < \beta < 1$.

- Both Zipf's law and Heaps' law are empirical.

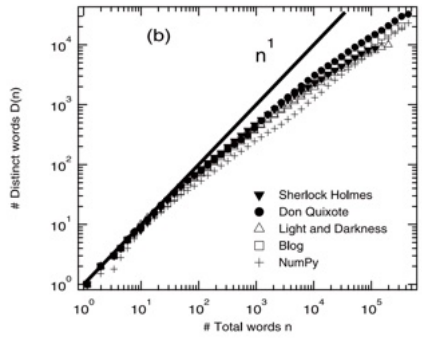


Image from: Sano, Yukie; Takayasu, Hideki; Takayasu, Misako (2012). "Zipf's Law and Heaps' Law Can Predict the Size of Potential Words". Progress of Theoretical Physics Supplement. 194: 202–209

Practical Details 4: Smoothing

- We apply a "modification" to the probabilities, which is called **smoothing**.
- To put it in an intuitive way, smoothing methods address the unseen combination (**NOT** unknown words) problem by stealing probability mass from seen events and reallocating it to unseen events.
- Different ways of smoothing.

Add-One (Laplace) Smoothing

- Assume we add 1 to the count of every possible n-gram
- Example, for unigram,

$$P(w_i) = \frac{C(w_i)}{N}$$

where $N = \sum_{x'} C(x')$

- After smoothing:

$$P(w_i) = \frac{C(w_i) + 1}{N + |V|}$$

Add-One (Laplace) Smoothing

- Assume we add 1 to the count of every possible n-gram
- Example, for bigram,

$$P(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

After smoothing:

$$P(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + |V|}$$

vocabulary size

Why?

- We still need to make sure

$$\sum_{w'} P(w' \mid w_{i-1}) = 1$$

Add-One (Laplace) Smoothing

- However, add-one smoothing is not friendly to "common" n-grams.
- Suppose "little" occur 100 times, "little, lamb" occurs 10 times, and vocabulary size = 20,000.

Before smoothing:

$$P(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})} = \frac{C(little, lamb)}{C(little)} = \frac{10}{100} = 0.1$$

After add-one smoothing:

$$P(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + |V|} = \frac{10 + 1}{100 + 20{,}000} \approx 0.0005$$

Add-α Smoothing

- Assume we add α ($\alpha < 1$) to the count of every possible n-gram
- Example, for bigram:

$$P(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

After smoothing:

$$P(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha |V|}$$

Linear Interpolation Smoothing

- Interpolate n-gram model with k-gram model (k<=n)
- bigram

$$\hat{P}(w_i \mid w_{i-1}) = \lambda P(w_i) + (1 - \lambda) P(w_i \mid w_{i-1})$$

- trigram

$$\begin{aligned} \hat{P}(w_i \mid w_{i-2}, w_{i-1}) = {} & \lambda_1 P(w_i) && \text{unigram} \\ & + \lambda_2 P(w_i \mid w_{i-1}) && \text{bigram} \\ & + (1 - \lambda_1 - \lambda_2) P(w_i \mid w_{i-2}, w_{i-1}) && \text{trigram} \end{aligned}$$

N-gram Model Summary

- To estimate the probability of a sequence of words $w_1, w_2, \ldots, w_n$, we need:
  - a training corpus
  - chain rule
  - independence assumption
- As a result, we get (here we use trigram):

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_1, \ldots, w_{i-1}) = \prod_{i=1}^{n} P(w_i \mid w_{i-2}, w_{i-1})$$

- Then estimate each trigram probability from the training corpus:

$$P(w_i \mid w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})}$$

- Beginning/end symbol, UNK symbol, using logarithms, smoothing

Generating text with Language Models

- How do we use language models: a random sentence generator.
- Suppose we already learned a bigram language model, how to generate a sequence?

$$P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-1})$$

  - Generate the 1st word $w_1 \sim P(w \mid <s>)$
  - Generate the 2nd word $w_2 \sim P(w \mid w_1)$
  - Generate the 3nd word $w_3 \sim P(w \mid w_2)$
  - …
  - Until </s> is generated.

Generating text with Language Models

- Trigram language model:

$$P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-2}, w_{i-1})$$

  - Generate the 1st word $w_1 \sim P(w \mid <s>, <s>)$
  - Generate the 2nd word $w_2 \sim P(w \mid <s>, w_1)$
  - Generate the 3nd word $w_3 \sim P(w \mid w_1, w_2)$
  - …
  - Until </s> is generated.

Generating text with Language Models

- How do we select w_3 based on $P(w \mid w_1, w_2)$?
- Greedy

  Idea: choose the most likely word

$$w_3 = \arg\max_{w \in V} P(w \mid w_1, w_2)$$

…a major problem is a major problem is a major problem is a major problem is a major problem is a major problem…

Generating text with Language Models

- How do we select w_3 based on $P(w \mid w_1, w_2)$?
- Greedy

  Idea: choose the most likely word

$$w_3 = \arg\max_{w \in V} P(w \mid w_1, w_2)$$

- Sampling

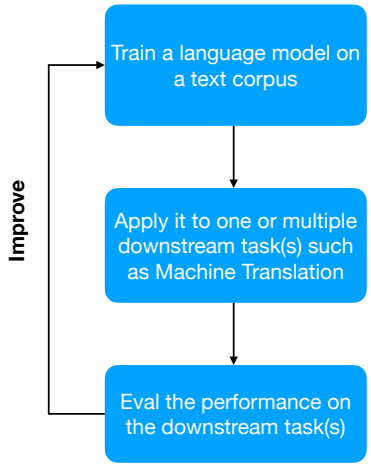
  Top-k: randomly select from top-k words with the highest probability

  Top-p: randomly select from the smallest set of tokens for which the cumulative probability reach a specified value p

Evaluating Language Models

- How do we know if one language model is better than another?
- Two types of evaluation in NLP
  1. **Intrinsic Evaluation**: design a measure that is inherent to the current task
  2. **Extrinsic Evaluation**: measure performance on a downstream application

Extrinsic Evaluation



- The most reliable evaluation.
- Can be more time consuming.
- Directly target downstream task.

Word Error Rate

- Originally developed for speech recognition.
- How much does the predicted sequence of words differ from the actual sequence of words in the correct transcript?

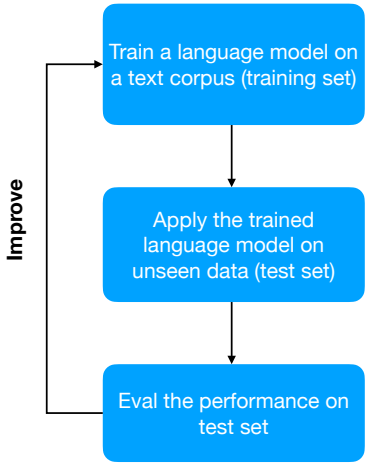
$$\text{WER} = \frac{\text{Insertions} + \text{Deletions} + \text{Substitutions}}{\text{Actual words in transcript}}$$

Insertions: “had little lamb” → “had **a** little lamb”

Deletions: “go **to** home” → “go home”

Substitutions: “the **star** night” → “the **starry** night”

Intrinsic Evaluation



- Can be much quicker in the development cycle.
- Intrinsic improvement may not guarantee extrinsic improvement.
- Both intrinsic and extrinsic eval require an evaluation metric that allow us to compare the performance of different models.
- It is not always obvious how to design the evaluation metric.

Intrinsic Evaluation of Language Models: Perplexity

- Suppose we have 2 language model that can assign probabilities to sequence of words.
- How do we know which is better?
- We have a training set, and a test set.
- We trained the 2 models on the training set, and compute the probability of the test set. Whichever model has a high probability is better.

Intrinsic Evaluation of Language Models: Perplexity

- We have a language model that can assign probabilities to sequence of words.
- The **perplexity** of this model on a sequence of words $w_1, \ldots, w_n$ is defined as

$$\begin{aligned}\text{Perplexity}(w_1, \ldots, w_n) &= P(w_1, \ldots, w_n)^{-\frac{1}{n}} \\ &= \sqrt[n]{\frac{1}{P(w_1, \ldots, w_n)}} \\ &= \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i \mid w_1, \ldots, w_{i-1})}}\end{aligned}$$

Intrinsic Evaluation of Language Models: Perplexity

$$\text{Perplexity}(w_1, \ldots, w_n) = P(w_1, \ldots, w_n)^{-\frac{1}{n}}$$

- unigram vs. bigram vs. trigram?
- Lower perplexity is better (higher probability)
- Perplexity (PPL) of unigram > PPL of bigram > PPL of trigram

	unigram	bigram	trigram
Perplexity	962	170	109

Train and test on the Wall Street Journal corpus

Intrinsic Evaluation of Language Models: Perplexity

- If my language model gets a perplexity on some dataset as 20. Is it good or not?
- Two language models' perplexity can only be compared if they use the same vocabularies.
  - Some language might be more predictable

Imagine a language with vocabulary size $|V|$. Suppose each word has equal probability, i.e., $P(w) = \frac{1}{|V|}$. What is the perplexity of the unigram model?

Intrinsic Evaluation of Language Models: Perplexity

Imagine a language with vocabulary size $|V|$. Suppose each word has equal probability, i.e., $P(w) = \frac{1}{|V|}$. What is the perplexity of the unigram model?

$$\begin{aligned}\text{Perplexity}(w_1, \ldots, w_n) &= \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i \mid w_1, \ldots, w_{i-1})}} \\ &= \sqrt[n]{\prod_{i=1}^{n} \frac{1}{1/|V|}} \\ &= |V|\end{aligned}$$

Intrinsic Evaluation of Language Models: Perplexity

- If my language model gets a perplexity on some dataset as 20. Is it good or not?
- Two language models' perplexity can only be compared if they use the same vocabularies.
  - Some languages might be more predictable ("easy")
- Test data must be disjoint from training data. Knowledge of the test set can cause the perplexity to be artificially low.

Intrinsic Evaluation of Language Models: Perplexity

- Use logarithms to prevent underflow:

$$\begin{aligned}\text{Perplexity}(w_1, \ldots, w_n) &= P(w_1, \ldots, w_n)^{-\frac{1}{n}} \\ &= \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i \mid w_1, \ldots, w_{i-1})}} \\ &= \exp\Big(-\frac{1}{n}\sum_{i=1}^{n} \log P(w_i \mid w_1, \ldots, w_{i-1})\Big)\end{aligned}$$

Entropy

- In Information Theory, the entropy of a discrete random variable X is defined as:

$$H(X) = -\sum_x p(x)\log p(x)$$

- It is also referred to as Shannon entropy.
- It describes the level of “information” or “uncertainty”.
- Here we use base 2 for $\log$, which provides **unit of bits**.

Entropy Example

- Suppose we flip a fair coin. The entropy of the outcome:

$$\begin{aligned} H(X) &= -\sum_x p(x)\log p(x) \\ &= -\sum_{i=1}^{2} \frac{1}{2}\log_2 \frac{1}{2} \\ &= 1 \end{aligned}$$

Entropy Example

- Suppose we flip an unfair coin. P(heads)=0.9 and P(tails)=0.1. The entropy of the outcome:

$$\begin{aligned} H(X) &= -\sum_x p(x)\log p(x) \\ &= -0.9\log_2 0.9 - 0.1\log_2 0.1 \\ &= 0.469 < 1 \end{aligned}$$

Entropy Example

- Suppose we flip an unfair coin. P(heads)=1 and P(tails)=0. The entropy of the outcome:

$$\begin{aligned} H(X) &= -\sum_x p(x)\log p(x) \\ &= -1\log_2 1 \\ &= 0 \end{aligned}$$

Entropy Example

- Suppose we roll a fair 6-sided dice. The entropy of the outcome:

$$\begin{aligned} H(X) &= -\sum_x p(x)\log p(x) \\ &= -\sum_{i=1}^{6} \frac{1}{6}\log_2 \frac{1}{6} \\ &= 2.585 \end{aligned}$$

- Uniform probability yields maximum uncertainty and therefore maximum entropy.

Entropy of a Language

- Entropy over a sequence $W = \{w_1, \ldots, w_n\}$ from a language L:

$$H(w_1, \ldots, w_n) = -\sum_{W \in L} p(w_1, \ldots, w_n)\log p(w_1, \ldots, w_n)$$

- This will depend on how long the sequence is. To have a more meaningful measure, we get the average, also called entropy rate:

$$\frac{1}{n}H(w_1, \ldots, w_n) = -\frac{1}{n}\sum_{W \in L} p(w_1, \ldots, w_n)\log p(w_1, \ldots, w_n)$$

Entropy of a Language

- Define entropy of the language L:

$$H(L) = -\frac{1}{n}\lim_{n\to\infty}\sum_{W \in L} p(w_1, \ldots, w_n)\log p(w_1, \ldots, w_n)$$

- This can be simplified (Shannon-McMillan-Breiman theorem) to:

$$H(L) = -\frac{1}{n}\lim_{n\to\infty}\log p(w_1, \ldots, w_n)$$

- But we don't know the true probability distribution p.

Cross-entropy

- In practice, we don't know the true probability distribution p for language L, only an estimated distribution $\hat{p}$ from a language model.
- Define cross-entropy as

$$H(p, \hat{p}) = -\sum_x p(x)\log \hat{p}(x)$$

- According to Gibb's inequality, entropy is less than or equal to its cross-entropy, i.e.,

$$-\sum_x p(x)\log p(x) \leq -\sum_x p(x)\log \hat{p}(x)$$

Cross-entropy

- According to Gibb's inequality, entropy is less than or equal to its cross-entropy, i.e.,

$$-\sum_x p(x)\log p(x) \leq -\sum_x p(x)\log \hat{p}(x)$$

- This means that if we have two language models, the more accurate model will have a lower cross-entropy.

Cross-entropy

- Following Shannon-McMillan-Breiman theorem,

$$H(p,\hat{p}) = -\frac{1}{n}\lim_{n\to\infty}\log \hat{p}(w_1,\ldots,w_n)$$

- For a language model, lower $H(p,\hat{p})$ is better.

- Recall perplexity, perplexity is simply $2^{\text{cross-entropy}}$.

$$\text{Perplexity}(w_1,\ldots,w_n) = P(w_1,\ldots,w_n)^{-\frac{1}{n}}$$

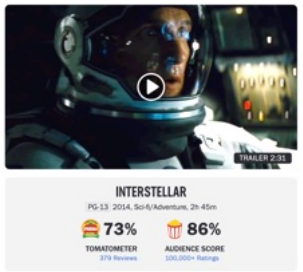
$$2^{\text{cross-entropy}} = 2^{-\frac{1}{n}\log_2 P(w_1,\ldots,w_n)} = P(w_1,\ldots,w_n)^{-\frac{1}{n}}$$

CS 5134/6034 Natural Language Processing
Fall 2024

Lecture 6: Text Classification - Naive Bayes

University of CINCINNATI

What is Text Classification?

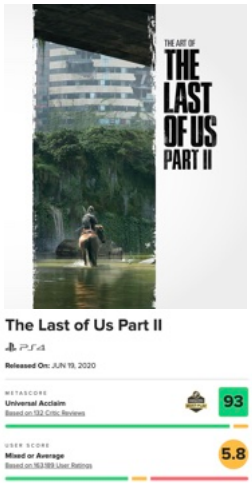


★★★★★ Jan 15, 2024
One of the greatest movies of all time. The ending took my breath away. Christopher Nolan is a filmmaking genius. Unbelievably good

★★ Dec 2, 2023
Robot makes a film about humans. Meh.

from rottentomatoes

What is Text Classification?



10 JAN 14, 2024
This is not a game, this is a life, and the story was great, I don't know why some people didn't like it, but the story was great, the gameplay was very eventful, and the music, I can't tell you anymore.

4 JAN 1, 2024
It's like the first game, but without any likable characters and constant convoluted and boring flashbacks and unskippable scenes to remove any enjoyment you might have had.

from metacritic

What is Text Classification?

- **Text classification**: assigning a class to a given text.
- For example, **sentiment analysis** is a type of text classification. It can be a **binary classification** task, which assign one of two labels `{positive, negative}` to the input (e.g., a movie or game review).
- It can also be a **multiclass classification** task: assign one of K labels (e.g., `{positive, negative, neutral}`) to the input.
- A **classifier** is a function that maps input $x \in X$ to a pre-defined set of class labels $y \in Y$.

Text Classification: Hand-coded Rules

- Rules based on combinations of words or other patterns. For example, pre-define sets of words:

  positive: {good, great, like, amazing, …} ⇒ +1 for each +ve term

  negative: {bad, boring, hate, weak, …} ⇒ -1 for each -ve term

- Simplest rule: count positive and negative words in the text. Predict which is greater.

- What are the problems with simple counting?

Machine Learning for NLP

- Until the 1990s, NLP systems primarily consisted of manually written dictionaries, grammars, and rules.

- Manually creating resources can be very effective, allowing hand-tailoring for specific domains and tasks. (Good)

- But hand-crafting resources is time-consuming and the resources are prone to errors, prone to omissions, and tend to be brittle. (Bad)

- Today, most NLP systems use statistical methods or machine learning, yielding more robust technology.

Why? ↳ Way more data.

↳ Much better computing power.

Types of Machine Learning

Machine Learning (ML) systems come in three flavors:

- **Supervised ML**: "gold standard" annotated texts are used for training (typically labeled by humans).

- **Weakly Supervised / Semi-Supervised ML**: usually a small number of "gold" annotated texts plus a large set of unannotated texts are used for training. (Sometimes a large set of noisily labeled data is used for training.)

- **Unsupervised ML:** predictions are made entirely based on unannotated texts, usually with some type of clustering.

Machine Learning Classifiers

- Most machine learning algorithms train a classification model (classifier) using labeled data as input. The resulting classifier then takes unlabeled examples as input and predicts labels for them.

- Each instance is typically represented as **a feature vector**. Each feature is an attribute that can take a range of values.

- The goal is to generalize from the training examples to learn how to accurately label **new** examples.

  – Too much generalization produces errors. (Overfit)

  – Too little generalization overfits, creating overly specific models. (Underfit)

Data Sets

- NLP experiments typically rely on three subsets of annotated data:
- **Training Data:** used for acquiring statistics or training a machine learning algorithm. System developers can also inspect the data to help design and improve their system.
- **Development (tuning) Data:** used as pseudo test data to evaluate the system during development and determine good parameter settings.
- **Test Data:** blind (unseen) data used for evaluation. The developer should never look at this data or the validity of the experimental results will be compromised.

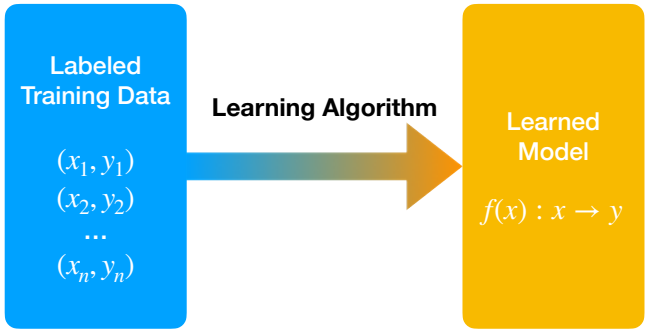
Supervised Machine Learning

- The supervised learning paradigm:

  Given gold (correctly) labeled training data, obtain a classifier that predict the labels as accurately as possible.
- It is called "supervised" because the learning algorithm can get **feedback** about how accurate its predictions are from the labels in the training data.

Supervised Machine Learning: Training

Input is in vector form.

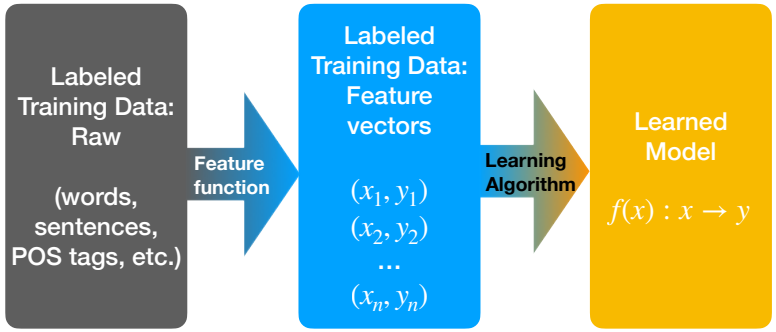


Feature Engineering

- How do we get these x_i?
- The data we want to classify could be anything, e.g., movie reviews, emails, conversations, words, POS tags, etc.
- We want to be able to build **a vector** x_i for any data point, which is called **a feature vector**.

Feature Engineering

- Before we train a classifier on our data, we need to build feature vectors for them.



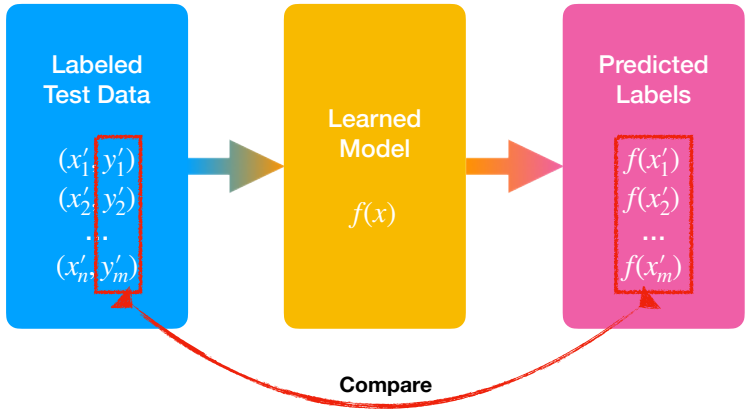
Feature Engineering

- Our classification algorithms typically require numerical representations of data.
- Before we training a classifier, we first define **a suitable feature function** that maps raw data points to vectors.
- We denote each data point's feature vector as x_i
  - Each element in x_i is one **feature**.
  - The dimension (# of features) of x_i is fixed, and could be very large.
  - Feature vector x_i needs to capture key attributes of the data that the classifier needs.
- In practice, **feature engineering** (design suitable feature functions) is very important for accurate classification.

Feature Engineering

- The performance of any ML system depends crucially on the features used to represent the examples.
- The choice of features can be more important than the choice of learning algorithm!
  - – given inadequate features, no ML algorithm will perform well.
  - – given strong features, most ML algorithms will perform well.
- Many ML algorithms exist with different strengths and weaknesses, but defining good features is critically important!

Supervised Machine Learning: Testing



Supervised Machine Learning

- The supervised learning task:

  Given gold (correctly) labeled training data $\{(x_i, y_i)\}$, where feature vector x_i is a representation of the input data point, y_i belongs to a fixed set of class labels.

  Our goal is to obtain a classifier $f(x)$ that predict the labels as accurately as possible: $f(x_i) \rightarrow y_i$

- To make this work, we need:
  - what **class of functions** $f(x)$ to consider
  - what **learning algorithm** we apply to learn $f(x)$

Probabilistic Classifiers

- Given input x, we want to find the most likely class y, i.e.,

$$y^* = \arg\max_y P(Y = y \mid X = x)$$

$$y^* = \arg\max_y g(y)$$

y^* is the y that maximizes $g(y)$

$$P(Y = y \mid X = x)$$

The probability that the class label is y when the input is x

Modeling $P(Y \mid X)$ with Bayes Rule

- Bayes Rule

$$\underbrace{P(Y \mid X)}_{\text{Posterior}} = \frac{P(X, Y)}{P(X)} = \frac{P(X \mid Y)P(Y)}{P(X)} \propto \underbrace{P(X \mid Y)}_{\text{Likelihood}}\underbrace{P(Y)}_{\text{Prior}}$$

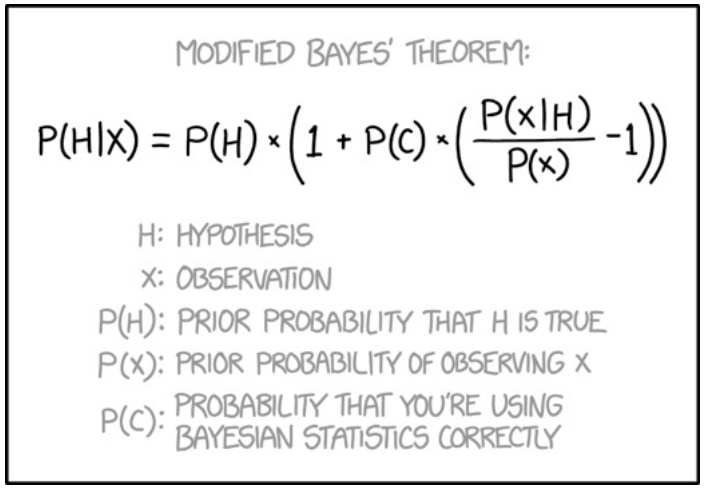
- The posterior $P(Y \mid X)$ is proportional to the likelihood $P(X \mid Y)$ times the prior $P(Y)$.

Using Bayes Rule for our Classifier

$$\begin{aligned} y^* &= \arg\max_y P(Y = y \mid X = x) \\ &= \arg\max_y \frac{P(X = x \mid Y = y)P(Y = y)}{P(X = x)} \\ &= \arg\max_y P(X = x \mid Y = y)P(Y = y) \end{aligned}$$

$P(X)$ does not affect "$\arg\max_y$"

Humor



Modeling $P(Y = y)$

- $P(Y = y)$ is the "prior" class probability.

  We can estimate this as the fraction of examples in the training data that have class y:

$$P(Y = y) = \frac{\text{\# of examples that have class } y}{\text{\# of all examples}}$$

- "Relative frequency" in the training data. For example,
  - Nominator: number of reviews labeled as y
  - Denominator: number of all reviews

Modeling $P(X = x \mid Y = y)$

- $P(X = x \mid Y = y)$ is the "likelyhood" of input x.
- $x = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ is a vector. Each α_i is a feature.
- Let's make a (**naive**) independence assumption:

$$P(X = (\alpha_1, \alpha_2, \ldots, \alpha_n) \mid Y = y) = \prod_{i=1}^{n} P(X_i = \alpha_i \mid Y = y)$$

The Naive Bayes Classifier

- Assign class y^* to input $x = (\alpha_1, \ldots, \alpha_n)$ if

$$y^* = \arg\max_y P(Y = y) \prod_{i=1}^{n} P(X_i = \alpha_i \mid Y = y)$$

$P(Y = y)$ is the prior class probability

(estimated as the fraction of items in the training data with class y)

$P(X_i = \alpha_i \mid Y = y)$ is the (class-conditional) likelihood of the feature α_i conditioned on the class y.

Naive Bayes for Text Classification

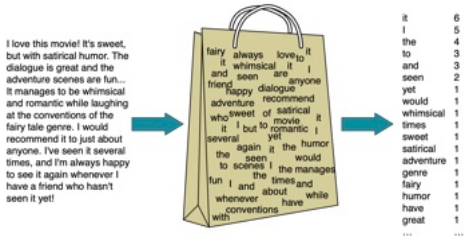
- Modeling $P(Y = y)$ based on the relative frequency.
- Modeling $P(X = x \mid Y = y)$. There are many different ways to define it.
- Here we show one example. Suppose x is a sequence of words:

  $P(X = x \mid Y = y) = P(w_1, w_2, \ldots, w_n \mid y)$

  Let's make an assumption.

Modeling $P(X = x \mid Y = y)$ for Text Classification

- **Bag-of-words (BoW)**: the positions of words do not matter!
- BoW models are often used to classify documents. They treat a document as an unordered collection of words with frequency counts.



from the J&M textbook

Modeling $P(X = x \mid Y = y)$ for Text Classification

- The **naive Bayes** assumption: the probabilities $P(w_i \mid y)$ are independent and hence can be "naively" multiplied:

$$P(w_1, w_2, \ldots, w_n \mid y) = P(w_1 \mid y)P(w_2 \mid y) \ldots P(w_n \mid y)$$

- With this independence assumption, now we only need to define all $P(w_i \mid y)$.

Naive Bayes Classifier for Text Classification

- With this assumption, we can have a **Naive Bayes classifier** that selects the class by:

$$\begin{aligned} y^* &= \arg\max_y P(Y = y \mid X = x) \\ &= \arg\max_y P(X = x \mid Y = y)P(Y = y) \\ &= \arg\max_y P(w_1 \mid y)P(w_2 \mid y) \ldots P(w_n \mid y)P(y) \\ &= \arg\max_y P(y) \prod_{i=1}^{n} P(w_i \mid y) \end{aligned}$$

- Use log probabilities to prevent underflow!

Computing $P(w_i \mid y)$

- Let's see how to define $P(w_i \mid y)$.
- Suppose this is a sentiment analysis task for movie reviews. Each w_i is a word in the training corpus vocabulary V. The label set is {+, -}.
- We concatenate all positive (+) reviews into one big positive text, all negative (-) reviews into one big negative text.

$$P(w_i \mid +) = \frac{\text{count}(w_i, +)}{\sum_{w \in V} \text{count}(w, +)} \qquad P(w_i \mid -) = \frac{\text{count}(w_i, -)}{\sum_{w \in V} \text{count}(w, -)}$$

where $\text{count}(w_i, +)$ means the number of times w_i appear in the big positive text, similarly for $\text{count}(w_i, -)$.

Smoothing

↳ Only for words present in the training data.

- What if some $\text{count}(w_i, +)$ or $\text{count}(w_i, -)$ are 0?
- Add a small amount to observed frequency counts.
- Laplace (add-1) smoothing. Use $P(w_i \mid +)$ as an example:

$$P(w_i \mid +) = \frac{\text{count}(w_i, +) + 1}{\sum_{w \in V} (\text{count}(w, +) + 1)} = \frac{\text{count}(w_i, +) + 1}{\sum_{w \in V} \text{count}(w, +) + |V|}$$

Unknown Words

- Recall how we deal with unknown words in n-gram language models?
- Create an <UNK> token.
- Or we can also ignore them: remove them from the test document.

Stop Words

- Some systems ignore stop words.
- Stop words: very frequent words like `the` and `a`.
  - Sort the vocabulary by word frequency in training set
  - Put the top 10 or 50 words into a stop word list.
  - Then remove all stop words from both training and test sets.

Naive Bayes Example for Sentiment Analysis

- Suppose we have a training corpus of movie reviews and their sentiment (positive + and negative -).

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable with no fun

- The prior probability $P(Y = y)$:

$$P(+) = \frac{2}{5} \qquad P(-) = \frac{3}{5}$$

Naive Bayes Example for Sentiment Analysis

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable ~~with~~ no fun

# of words in "big positive text": 9
# of words in "big negative text": 14
$|V|$: 20
Suppose "with" is in the stop list and we ignore stop words.

- The likelyhood probability $P(X = x \mid Y = y)$:

$$P(\text{predictable} \mid +) = \frac{0+1}{9+20} \qquad P(\text{predictable} \mid -) = \frac{1+1}{14+20}$$

$$P(\text{no} \mid +) = \frac{0+1}{9+20} \qquad P(\text{no} \mid -) = \frac{1+1}{14+20}$$

$$P(\text{fun} \mid +) = \frac{1+1}{9+20} \qquad P(\text{fun} \mid -) = \frac{0+1}{14+20}$$

Naive Bayes Example for Sentiment Analysis

- The prior probability $P(Y = y)$: $P(+) = \frac{2}{5}$ $P(-) = \frac{3}{5}$
- The likelyhood probability $P(X = x \mid Y = y)$:

$$P(\text{predictable} \mid +) = \frac{0+1}{9+20} \qquad P(\text{predictable} \mid -) = \frac{1+1}{14+20}$$

$$P(\text{no} \mid +) = \frac{0+1}{9+20} \qquad P(\text{no} \mid -) = \frac{1+1}{14+20}$$

$$P(\text{fun} \mid +) = \frac{1+1}{9+20} \qquad P(\text{no} \mid -) = \frac{0+1}{14+20}$$

- For the test sentence S = "predictable with no fun":

$$P(+)P(S \mid +) = \frac{2}{5} * \frac{1*1*2}{29^3} = 3.3 \times 10^{-5} \qquad P(-)P(S \mid -) = \frac{3}{5} * \frac{2*2*1}{34^3} = 6.1 \times 10^{-5}$$

So the model predicts the class **"negative"**.

Naive Bayes Summary

- Naive Bayes can be applied to a wide variety of classification tasks, such as text categorization, spam detection, language identification, etc.
- This approach can also be used with character or word sequences (n-grams), rather than just with individual words.
- Naive Bayes may not perform as well as more sophisticated ML algorithms, but it often serves as a good baseline for text classification.

CS 5134/6034 Natural Language Processing
Fall 2024

Lecture 7: Logistic Regression

University of CINCINNATI

Evaluating Classifiers

Cross-Validation

- Evaluation setup:

  Split into separate training, development and test sets.

- Alternative: n-fold cross-validation
  - Split data into n partitions of equal size called **folds**.

  - Run n experiments, each using fold i for testing and the remaining folds for training.
  - The results are then averaged.

- Cross-validation allows **ALL** of the data to be used for both training and testing, but never at the same time.

Baselines

- It is important to understand how difficult a problem is, and how well simple or conventional methods perform.

  E.g., choosing the most frequent POS tag yields ~90% accuracy

- Simple techniques sometimes work surprisingly well. Don't assume that a fancy technique will be better!

  E.g., a POS tagger that achieves < 90% accuracy is not very good!

- It is essential to know the underlying class distributions in your data.

- Given labeled data, a common good baseline is a supervised ML system with bag-of-word features.

Evaluation Metrics

- **Accuracy:** the % of instances assigned a correct label
- It is easy to get high accuracy if one class is very common.
- Suppose a 3-class classification task. 90% of the examples are label-1. If a classifier always predicts label-1, the accuracy is 90%!

Precision and Recall

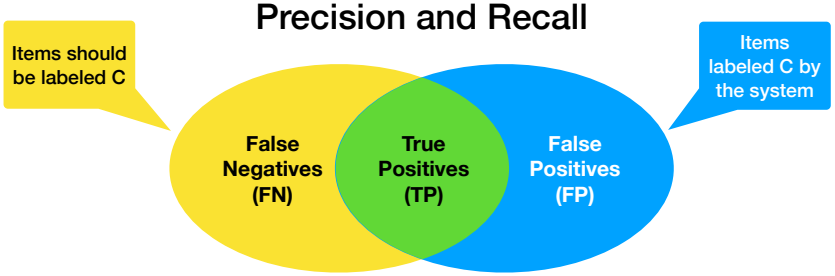
- **Precision:** for a class C, the % of instances assigned the label C by the system that actually have label C in the test data

$$\frac{\text{\# correctly labeled as C}}{\text{\# labeled as C}}$$

- **Recall:** for a class C, the % of instances that have label C in the test data get assigned label C by the system.

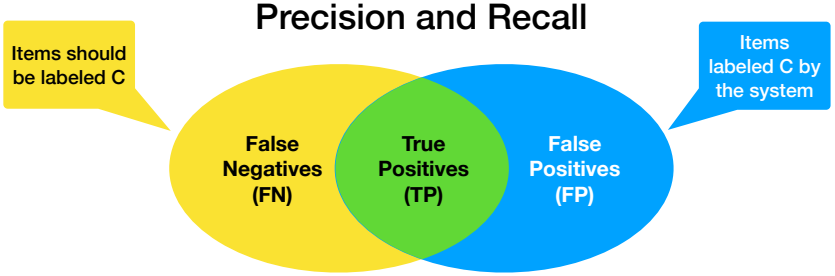
$$\frac{\text{\# correctly labeled as C}}{\text{\# true instances of C}}$$

Precision and Recall



- **True positives:** Items that were labeled C by the system, and should be labeled C.
- **False positives:** Items that were labeled C by the system, but should not be labeled C.
- **False negatives:** Items that were not labeled C by the system, but should be labeled C.

Precision and Recall



- Precision: P = TP / (TP + FP)
- Recall: R = TP / (TP + FN)
- F-score: $F = \frac{2}{\frac{1}{P} + \frac{1}{R}} = 2*P*R / (P+R)$

Confusion Matrix

- A confusion matrix presents how many items that have label C in the test set are labeled with C' by the system.

True Labels

System Output	Action	Comedy	Romance
Action	18	12	6
Comedy	5	19	10
Romance	4	0	8

Confusion Matrix

- A confusion matrix presents how many items that have label C in the test set are labeled with C' by the system.

True Labels

System Output	Action	Comedy	Romance
Action	18	12	6
Comedy	5	19	10
Romance	4	0	8
	$Recall_{action}$= 18/27	$Recall_{comedy}$= 19/31	$Recall_{romance}$= 8/24

Confusion Matrix

- A confusion matrix presents how many items that have label C in the test set are labeled with C' by the system.

True Labels

System Output	Action	Comedy	Romance	
Action	18	12	6	$Precision_{action}$=18/36
Comedy	5	19	10	$Precision_{comedy}$=19/34
Romance	4	0	8	$Precision_{romance}$=8/12

Confusion Matrix

- A confusion matrix presents how many items that have label C in the test set are labeled with C' by the system.

True Labels

System Output	Action	Comedy	Romance	
Action	18	12	6	$Precision_{action}$=18/36
Comedy	5	19	10	$Precision_{comedy}$=19/34
Romance	4	0	8	$Precision_{romance}$=8/12
	$Recall_{action}$= 18/27	$Recall_{comedy}$= 19/31	$Recall_{romance}$= 8/24	

Macro-average

- Macro-average: average **over all n classes**

Macro-averaged Precision =

$$\frac{1}{3} \times (\text{Precision}_{action} + \text{Precision}_{comedy} + \text{Precision}_{romance})$$

Macro-averaged Recall =

$$\frac{1}{3} \times (\text{Recall}_{action} + \text{Recall}_{comedy} + \text{Recall}_{romance})$$

Micro-average

- Micro-average: average **over all examples**

Micro-averaged Precision =

$$\frac{\text{TP}_{action} + \text{TP}_{comedy} + \text{TP}_{romance}}{\text{TP}_{action} + \text{TP}_{comedy} + \text{TP}_{romance} + \text{FP}_{action} + \text{FP}_{comedy} + \text{FP}_{romance}}$$

Micro-averaged Recall =

$$\frac{\text{TP}_{action} + \text{TP}_{comedy} + \text{TP}_{romance}}{\text{TP}_{action} + \text{TP}_{comedy} + \text{TP}_{romance} + \text{FN}_{action} + \text{FN}_{comedy} + \text{FN}_{romance}}$$

Macro-averaged Precision

- Macro-averaged precision: average the precision **over all n classes**

True Labels

System Output	Action	Comedy	Romance	
Action	18	12	6	$\text{Precision}_{action}$=18/36
Comedy	5	19	10	$\text{Precision}_{comedy}$=19/34
Romance	4	0	8	$\text{Precision}_{romance}$=8/12

Macro-averaged Precision = $\frac{1}{3} \times (\frac{18}{36} + \frac{19}{34} + \frac{8}{12})$

Micro-averaged Precision

- Micro-averaged precision: average the precision **over all instances**

True Labels

System Output	Action	Comedy	Romance	
Action	18	12	6	$\text{Precision}_{action}$=18/36
Comedy	5	19	10	$\text{Precision}_{comedy}$=19/34
Romance	4	0	8	$\text{Precision}_{romance}$=8/12

Micro-averaged Precision = $\frac{1}{3} \times \frac{18 + 19 + 8}{(18 + 19 + 8) + (12 + 6 + 5 + 10 + 4 + 0)}$

Macro-average vs. Micro-average

- Which average should you report?
- **Macro**-averaging is useful if **all classes** are equally important, especially for an **imbalanced dataset**.
- **Micro**-averaging is useful when you want to account for the total number of misclassifications in the dataset. For multi-class classification, micro-averaged precision=micro averaged recall=accuracy (think why?).

Probabilistic Classifiers

- A probabilistic classifier returns the most likely class y for input x,

$$y^* = \arg\max_y P(Y = y \mid X = x)$$

- Naive Bayes uses Bayes Rule:

$$y^* = \arg\max_y P(y \mid x) = \arg\max_y P(x \mid y)P(y)$$

Naive Bayes models the **joint distribution** of the class and the data:

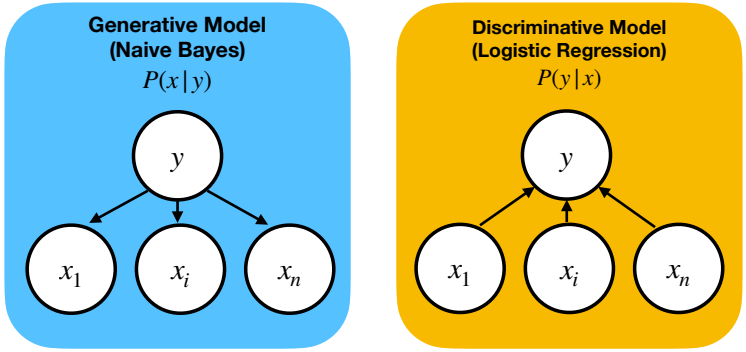
$$P(x \mid y)P(y) = P(x, y)$$

Joint models are also called **generative** models.

- **Discriminative** (also called **conditional**) models try to model $P(y \mid x)$ directly.

Generative vs. Discriminative Models

- For a classification task, given data point $x = (x_1, x_2, \ldots, x_n)$, we want a model to predict label y.



How to model $P(y \mid x)$ for any x?

- Bag-of-words Generative:

$$P(x, y) = P(y)\prod_{i=1}^{n} P(x_i \mid y) = \frac{c(y)}{\sum_{y'} c(y')}\prod_{i=1}^{n}\frac{c(x_i, y)}{\sum_{x'} c(x', y)}$$

- Bag-of-words Discriminative:

$$P(y \mid x) = ?$$

How to model $P(y \mid x)$ for any x?

- What if we have never seen x before?
- Even if we could define a probability distribution

$$P(Y = y \mid X_i = x_i) \text{ s.t. } \sum_{y'} P(Y = y' \mid X_i = x_i) = 1$$

for any single feature x_i

- We can't just multiply these probabilities together to get one distribution over all y' for a given x because

$$\sum_{y'} P(Y = y' \mid X = x) = \sum_{y'} \prod_{i=1}^{n} P(Y = y' \mid X_i = x_i) < 1$$

How to model $P(y \mid x)$ for any x?

- Instead, define model that calculates probability directly based on parameters θ

$$P(y \mid x; \theta)$$

- We don't know how important each feature x_i of $x = (x_1, x_2, \ldots, x_n)$ for our classification task is.
- Given a set of observations (training), **linear regression** models produce a regression line that best fits the observed data.

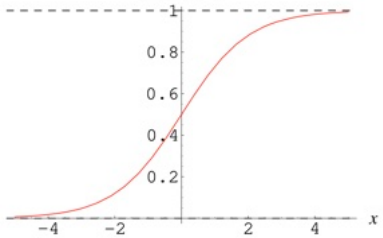
$$y' = \theta_0 + \sum_{i=1}^{n} \theta_i \cdot x_i$$

- But we need a probability.

Sigmoid Function

- The **sigmoid function** (also called **logistic function**) $\sigma(x)$ maps any real number to the range (0,1):

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Turning Probability into Classifier

- Now we can define:

$$P(y = 1 \mid x; \theta) = \sigma(\theta_0 + \sum_{i=1}^{n} \theta_i \cdot x_i)$$

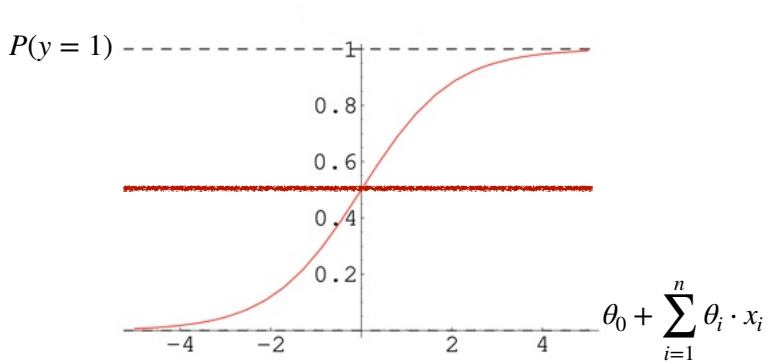
$$P(y = 0 \mid x; \theta) = 1 - P(y = 1 \mid x; \theta)$$

- Turning a probability into a classifier:

$$y^* = \begin{cases} 1 & \text{if } P(y = 1 \mid x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Here 0.5 is called the **decision boundary**.

Turning Probability into Classifier



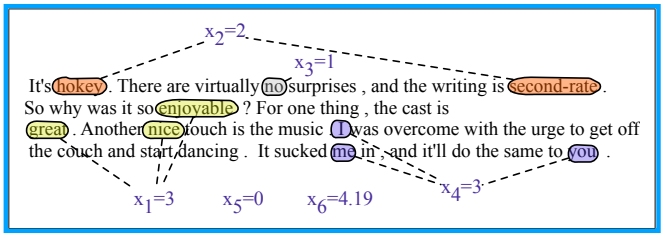
$$y^* = \begin{cases} 1 & \text{if } P(y=1 \mid x) > 0.5 \\ 0 & \text{otherwise} \end{cases} \quad \begin{array}{l} \text{if } \theta_0 + \sum_{i=1}^{n} \theta_i \cdot x_i > 0 \\ \text{if } \theta_0 + \sum_{i=1}^{n} \theta_i \cdot x_i \leq 0 \end{array}$$

Logistic Regression for Sentiment Analysis

- Does $y = 1$ or $y = 0$?

It's hokey . There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable ? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

Building Features



Var	Definition	Value
x_1	count(positive lexicon) ∈ doc)	3
x_2	count(negative lexicon) ∈ doc)	2
x_3	1 if "no" ∈ doc 0 otherwise	1
x_4	count(1st and 2nd pronouns ∈ doc)	3
x_5	1 if "!" ∈ doc 0 otherwise	0
x_6	log(word count of doc)	ln(66) = 4.19

Classifying Sentiment

- So we have $x = (3, 2, 1, 3, 0, 4.9)$
- Suppose $\theta = (2.5, -5.0, -1.2, 0.5, -2.0, 0.7)$, $\theta_0 = 0.1$

$$\begin{aligned} P(+ \mid x) &= \sigma(\theta_0 + \sum_{i=1}^{n} \theta_i \cdot x_i) \\ &= \sigma((2.5, -5.0, -1.2, 0.5, -2.0, 0.7) \cdot (3, 2, 1, 3, 0, 4.9) + 0.1) \\ &= \sigma(0.833) \\ &= 0.7 \end{aligned}$$

$$\begin{aligned} P(- \mid x) &= 1 - \sigma(\theta_0 + \sum_{i=1}^{n} \theta_i \cdot x_i) \\ &= 0.3 \end{aligned}$$

Logistic Regression: Binary Classification

- **Task:** Model $P(y \in \{0,1\} \mid x)$ for any input vector $x = (x_1, \ldots, x_n)$.
- **Idea:** Learn feature weights θ to capture how important each feature x_i is for predicting $y = 1$.
- For **binary** classification, logistic regression uses the **sigmoid** function:

$$P(y = 1 \mid x) = \sigma(\theta_0 + \sum_{i=1}^{n} \theta_i \cdot x_i)$$

Parameters to learn: θ

What about multi-class classification?

- In NLP, we often have tasks that involve many classes, such as POS tagging or semantic categories.
- **Multinomial logistic regression** (also called **maximum entropy modeling** or MaxEnt) generalizes to multiple classes.
- It uses a so-called **softmax function** to achieve probability distributions.

Multinomial Logistic Regression

- **Task:** Model $P(y \in \{y^{(1)}, \ldots, y^{(K)}\} \mid x)$ for any input vector $x = (x_1, \ldots, x_n)$.
- **Idea:** Learn feature weights $\theta^{(j)}$ to capture how important each feature x_i is for predicting $y^{(j)}$.

$$P(y^{(j)} \mid x) = f(\theta_0^{(j)} + \sum_{i=1}^{n} \theta_i^{(j)} \cdot x_i)$$

- We need to make a probability distribution:

$$0 \leq P(y^{(j)} \mid x) \leq 1 \text{ and } \sum_{j=1}^{K} P(y^{(j)} \mid x) = 1$$

Multinomial Logistic Regression

- For **multi-class** classification, **multinomial logistic regression** uses the **softmax** function:

$$\begin{aligned} P(y^{(j)} \mid x) &= \text{softmax}(z)^{(j)} \\ &= \frac{\exp(z^{(j)})}{\sum_{k=1}^{K} \exp(z^{(k)})} \\ &= \frac{\exp(-(\theta_0^{(j)} + \sum_{i=1}^{n} \theta_i^{(j)} \cdot x_i))}{\sum_{k=1}^{K} \exp(-(\theta_0^{(k)} + \sum_{i=1}^{n} \theta_i^{(k)} \cdot x_i))} \end{aligned}$$

Parameters to learn: $\theta^{(1)}, \ldots, \theta^{(K)}$

Softmax Function

- The softmax functions turns any vector of reals $z = (z_1, \ldots, z_K)$ into a discrete probability distribution $p = (p_1, \ldots, p_K)$ where $0 \leq p_i \leq 1$ and $\sum_{i=1}^{K} p_i = 1$

$$p_i = \text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{k=1}^{K} \exp(z_k)}$$

- Logistic regression applies the softmax to a linear combination of the input features.
- We will see the softmax function again when we talk about neural networks.

Softmax Function

- Suppose $z = (1,0,2)$

$$p_1 = \frac{\exp(z_1)}{\sum_{k=1}^{3} \exp(z_k)} = \frac{e}{e + 1 + e^2} = 0.245$$

$$p_2 = \frac{\exp(z_2)}{\sum_{k=1}^{3} \exp(z_k)} = \frac{1}{e + 1 + e^2} = 0.090$$

$$p_3 = \frac{\exp(z_3)}{\sum_{k=1}^{3} \exp(z_k)} = \frac{e^2}{e + 1 + e^2} = 0.665$$

- So, the new probability distribution $p = (0.245,\ 0.090,\ 0.665)$

Binary vs. Multinomial Logistic Regression

- Let $z_j = \exp(-(\theta_0^{(j)} + \sum_{i=1}^{n} \theta_i^{(j)} \cdot x_i))$, and $z' = \exp(-(\theta_0 + \sum_{i=1}^{n} \theta_i \cdot x_i))$
- Binary logistic regression:

$$P(y = 1 \mid x) = \frac{1}{1 + z'} \qquad P(y = 0 \mid x) = \frac{z'}{1 + z'}$$

- Multinomial logistic regression:

$$P(y = 1 \mid x) = \frac{z_1}{z_1 + z_0} \qquad P(y = 0 \mid x) = \frac{z_0}{z_1 + z_0}$$

- Binary logistic regression is a special case of multinomial logistic regression over two class (let $z_1 = 1, z_0 = z'$)

Discriminative Model Training

- The model calculates probability directly based on input x and parameters θ:

$$P(y \mid x; \theta)$$

- Define a loss function that is lower if the model is better, such as negative log likelihood

$$L_{train}(\theta) = -\sum_{(x,y) \in D_{train}} \log P(y \mid x; \theta)$$

- **Optimize** the parameters to minimize loss

$$\theta^* = \arg\max_{\theta} L_{train}(\theta)$$

Gradient Descent

- The gradient of the loss function with respect to the parameters:

$$\frac{\partial L_{train}(\theta)}{\partial \theta}$$

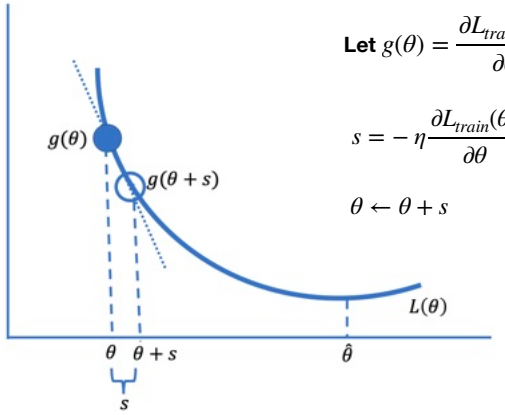
It indicates the direction of steepest increase in $L_{train}(\theta)$

- Move in the opposite direction to decrease $L_{train}(\theta)$

$$\theta \leftarrow \theta - \eta \frac{\partial L_{train}(\theta)}{\partial \theta}$$

η is called the **learning rate**

Gradient Descent



Let $g(\theta) = \frac{\partial L_{train}(\theta)}{\partial \theta}$

$$s = -\eta \frac{\partial L_{train}(\theta)}{\partial \theta}$$

$$\theta \leftarrow \theta + s$$

Stochastic Gradient Descent

- **Gradient descent:**

  Compute loss for entire dataset before updating weights

- **Stochastic gradient descent:**

  Compute loss for one (randomly sampled) training example before updating weights

$\theta \leftarrow 0$
repeat til done # see caption
  For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)
    1. Optional (for reporting): # How are we doing on this tuple?
      Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # What is our estimated output $\hat{y}$?
      Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$ from the true output $y^{(i)}$?
    2. $g \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss?
    3. $\theta \leftarrow \theta - \eta\, g$ # Go the other way instead
return θ

CS 5134/6034 Natural Language Processing
Fall 2024

Lecture 9: Part-of-Speech Tagging

University of CINCINNATI

Part-of-Speech

Hold the plane down when I hand plane it.

plane

- verb: to make smooth or even
- noun: a flat surface

What are the Classes?

- **Nouns**
  - **common nouns**: book, water
  - **count nouns** have a plural (books)
  - **mass nouns** (water) don't have a plural
  - **proper nouns** (Names): Cincinnati, Mary
- **Verbs**
  - wear, eat, live, travel
  - verbs have different morphological forms
- **Adjectives**
  - **attributive** use (modifying a noun): a great book
  - **predicative** use (as arguments of be): the book is boring
  - many **gradable** adjectives also have a **comparative form**: greater, better; and a **superlative form**: greatest, best.
- **Adverbs**
  - **manner** adverbs: quickly, slowly
  - **degree** adverbs: very, highly
  - **directional** and **locative** adverbs: left, downstairs
  - **temporal** adverbs: yesterday
- **Pronouns**
  - I, you, she, her, myself, …
- **Determiners**
  - the, a, this, that, some, …
- **Prepositions**
  - by, at, from, as, against, below, …
- **Conjunctions**
  - and, or, neither, but, ...
- **Modal auxiliaries**
  - will, may, could, can, …
- …

Fine-Grained Classes

- Noun
  - singular (NN): apple
  - plural (NNS): apples
  - proper (NNP, NNPS): iPhone, iPhones
- Verb
  - base (VB): eat
  - past tense (VBD): ate
  - gerund (VBG or present participle): eating
  - past participle (VBN): eaten
  - 3rd person singular present (VBZ): eats

…

Open vs. Closed Classes

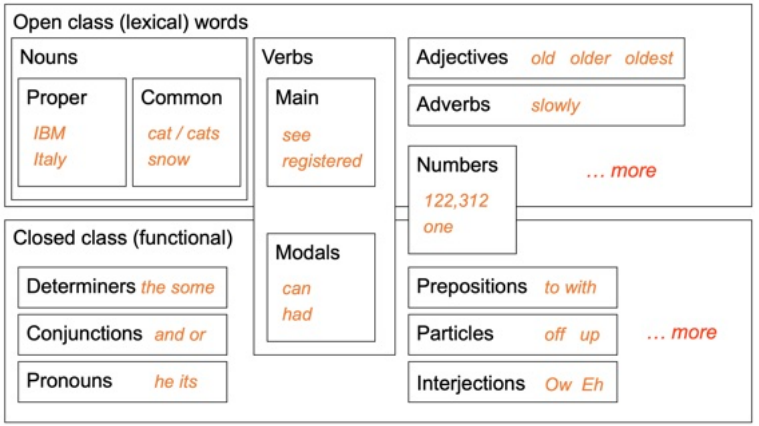
- Open classes

  e.g., nouns, verbs, adjectives, adverbs

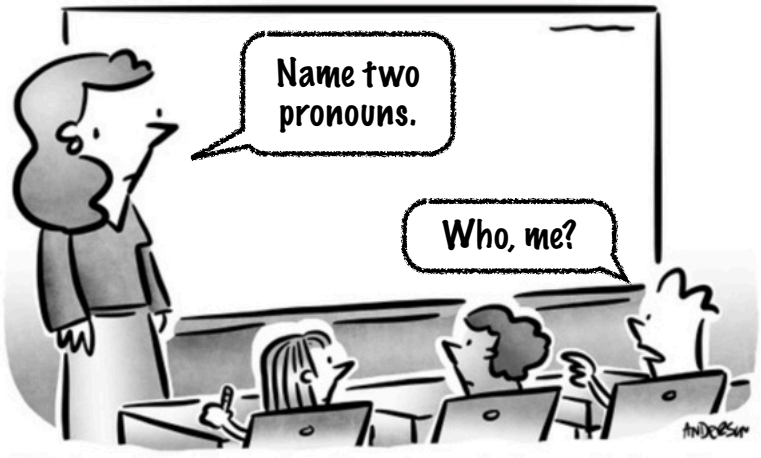
- Closed classes

  e.g., determiners, pronouns, prepositions

Open vs. Closed Classes



Humor



Part-of-Speech Tagging Task

- Input: a sequence of word tokens
- Output: a sequence of part-of-speech tags, one per word

Tars tears the door off the plane

NOUN VERB ART NOUN PREP ART NOUN

Why POS Tagging?

- POS tagging is one of the first steps in the traditional NLP pipeline (after tokenization, segmentation).
- POS tagging is traditionally viewed as a prerequisite for further analysis:
  - Syntactic Parsing:

    What words are in the sentence?
  - Information extraction:

    Find names, dates, relations, etc.

How many POS tags

- Commonly used tagsets for English usually have 40-100 tags.
  - Original Brown corpus used a large set of 87 POS tags.
  - Most common in NLP today is the Penn Treebank set of 45 tags.
  - British National Corpus Basic Target (C5) used 61 tags.

Penn Treebank POS Tags

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	*and, but, or*	SYM	symbol	*+,%, &*
CD	cardinal number	*one, two, three*	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, singular	*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*			

Ambiguity in POS Tagging

- tear/VERB off vs. tear/NOUN ran down
- glass of water/NOUN vs. water/VERB the plants
- a round/ADJ table vs. round/VERB the numbers
- abstract/ADJ art vs. paper abstract/NOUN

How much Ambiguity

- Common POS ambiguities in English

  noun—verb: tear, water, count

  adj—verb: round, average, lost

  noun—adj: abstract, chief, half

- A word is ambiguous if it has more than one POS

  How do we know that? Dictionary? Corpus?

How much Ambiguity

- Word types: unique words
- Word tokens: all occurrences
- Brown corpus:

  11% of the word types are ambiguous wrt. part of speech

  40% of the word tokens are ambiguous!

Creating a POS Tagger

- For a new language

  Step 0: Define a POS tagset

  Step 1: Annotate a corpus with these tags

Annotation Consistency is Essential

- Inter-annotator agreement (IAA) measure the consistency of different people when annotating data.
- High IAA is important to ensure that the task is well-defined and that the annotations have high integrity.
- To achieve high IAA, you must:

  — precisely define the annotation task, which usually requires detailed annotation guidelines that include examples and discuss how to handle boundary cases.

  — train the annotators, iteratively, while refining the guidelines.

  — measure IAA using an appropriate statistical measure.

The Kappa Statistic

- The Cohen's Kappa (κ) statistic measures the degree to which annotations by different people agree, adjusted for agreement due to chance.

$$\kappa = \frac{\text{P(agree) - P(expected)}}{\text{1 - P(expected)}}$$

where:

— P(agree) = proportion of times the annotators agree

— P(expected) = proportion of times the annotators are expected to agree by chance

Kappa for Two Annotators

$$\text{P(expected)} = \sum_{c \in C} P(c \mid A_1) \times P(c \mid A_2)$$

where

— C is the set of all possible classes (e.g., POS tagset)

— A_1 represents annotator #1

— A_2 represents annotator #2

A Simple Example

A1	Y	Y	N	Y	N	Y	N	N	Y	Y
A2	Y	Y	N	N	Y	Y	Y	N	Y	Y

$$P(\text{agree}) = 7/10 = 0.7$$

$$\begin{aligned} P(\text{expected}) &= P(Y \mid A_1) * P(Y \mid A_2) + P(N \mid A_1) * P(N \mid A_2) \\ &= 6/10 * 7/10 + 4/10 * 3/10 \\ &= 0.54 \end{aligned}$$

$$\kappa = \frac{\text{P(agree) - P(expected)}}{\text{1 - P(expected)}} = \frac{0.7 - 0.54}{1 - 0.54} = 0.348$$

An Exercise

A \ B	**Yes**	**No**
Yes	45	15
No	25	15

P(agree) = 60/100 = 0.6

P(expected) = 0.6*0.7+0.4*0.3 = 0.54

$$\kappa = \frac{\text{P(agree)} - \text{P(expected)}}{1 - \text{P(expected)}} = \frac{0.6 - 0.54}{1 - 0.54} = 0.1304$$

Creating a POS Tagger

- For a new language
  - Define a POS tagset
  - Annotate a corpus with these tags
- For a well-studied language
  - obtain a POS-tagged corpus
- Then build your POS tagging model…

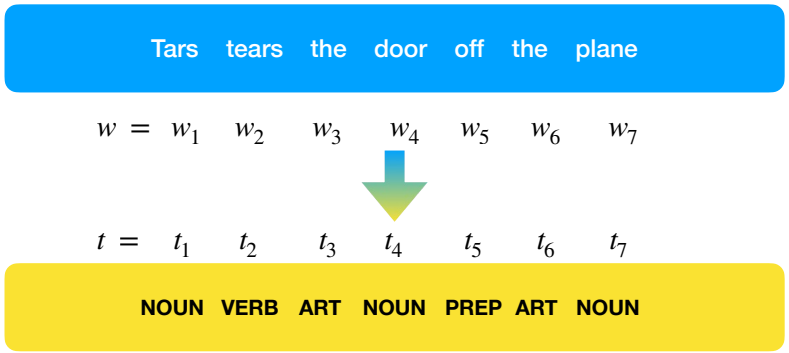
Rule-based POS Tagging

- Rule-based taggers rely on a dictionary and morphological analysis to provide possible POS tags for a word, and/or rules can be generated from training data.
- Manually developed disambiguation rules can perform reasonably well, but statistical methods generally work better IF you have annotated data.

Example rules:
- If preceding word = ART, then disambiguate {NOUN,VERB} as NOUN.
- If a possible verb does not agree in number or person with the preceding NP, then eliminate the verb tag.
- If the preceding word is an ADV, then it is tagged as VERB.

Statistical POS Tagging

- Statistical part-of-speech taggers identify the most likely sequence of tags for the words in a sentence.



Statistical POS Tagging

- Statistical part-of-speech taggers identify the most likely sequence of tags for the words in a sentence.
- What is the most likely POS tag sequence $t = (t_1, \ldots, t_n)$ for the given sentence $w = (w_1, \ldots, w_n)$?

$$t^* = \arg\max_t P(t \mid w)$$

- What can help us decide the tags?

What is useful for POS Tagging

- The word itself
  - some words may only be nouns
  - some words are ambiguous
  - one tag is more probable than others?
- Tags of surrounding words
  - two determiners rarely follow each other
  - two base form verbs rarely follow each other
  - determiner is almost always followed by adjective or noun
- A generative model?

POS tagging with Generative Models

$$\begin{aligned} t^* &= \arg\max_t P(t \mid w) \\ &= \arg\max_t \frac{P(t, w)}{P(w)} \\ &= \arg\max_t P(t, w) \\ &= \arg\max_t P(t)P(w \mid t) \end{aligned}$$

- $P(t, w)$ is the joint distribution of the POS tag sequence t we want to predict and the observed sentence w.
- Why do we decompose $P(t, w)$ into $P(t)$ and $P(w \mid t)$?

Modeling

- Let's make some assumptions
  - Each tag depends only on the previous tag: a bigram tag model

$$P(t) = \prod_{i=1}^{n} P(t_i \mid t_{i-1})$$

Define $t_0 = \phi$ as the beginning of sequence.

  - words are independent given tags

$$P(w \mid t) = \prod_{i=1}^{n} P(w_i \mid t_i)$$

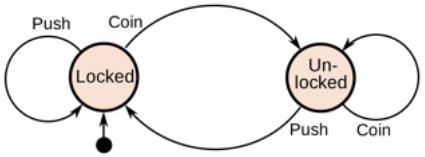
Modeling

- Under the first-order Markov assumption:

$$\begin{aligned} t^* &= \arg\max_t P(t)P(w \mid t) \\ &= \arg\max_t \prod_{i=1}^{n} P(t_i \mid t_{i-1})P(w_i \mid t_i) \end{aligned}$$

Finite-state Machine (FSM)

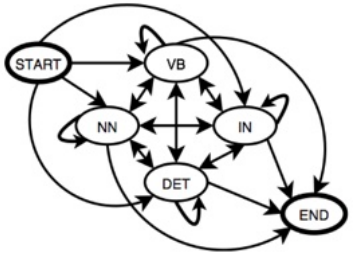
- A finite-state machine (FSM) or finite-state automaton (FSA), or simply a state machine, is a mathematical model of computation.
- The FSM can change from one state to another in response to some inputs; the change from one state to another is called a **transition**.



State diagram for a turnstile. From Wiki/Finite-state_machine.

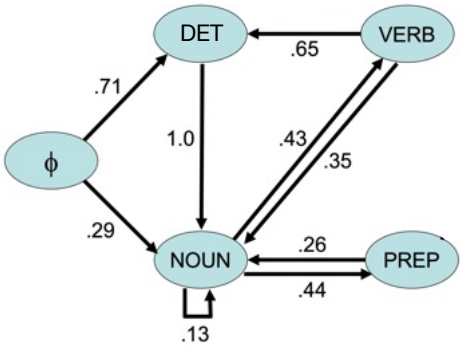
Probabilistic Finite-state Machine

- Another angle: sentences are generated by walking through **states** in a graph. Each state represents a tag.



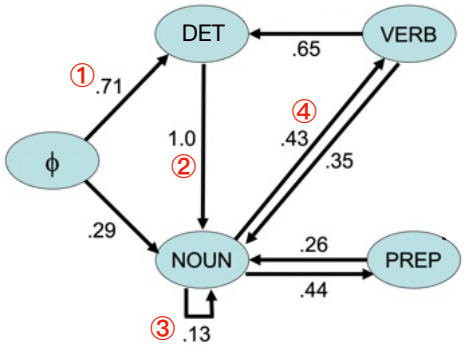
- Probability of moving from state s to s' is called "**transition probability**": $P(t_i = s' \mid t_{i-1} = s)$

Transition Probability



What is the probability of "DET NOUN NOUN VERB"?

Transition Probability



P(DET NOUN NOUN VERB) = 0.71*1.0*0.13*0.43

Example of Transition Probabilities

$t_{i-1} \backslash t_i$	NNP	MD	VB	JJ	NN	. . .
ϕ	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
  - Proper nouns (NNP) often begin sentences: P(NNP|ϕ) = 0.2767
  - Modal verbs (MD) are nearly always followed by base verbs (VB): P(VB|MD)=0.7968
  - Adjectives (JJ) are often followed by nouns (NN): P(NN|JJ) = 0.4509

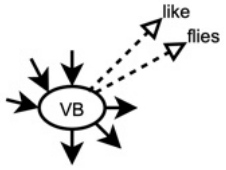
Example of Transition Probabilities

$t_{i-1} \backslash t_i$	NNP	MD	VB	JJ	NN	. . .
ϕ	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 (distribution) must sum up to 1.

Probabilistic Finite-state Machine: Outputs

- When passing through each state, emit a word.



- Probability of emitting w from state s is called the emission probability: $P(w_i = w \mid t_i = s)$

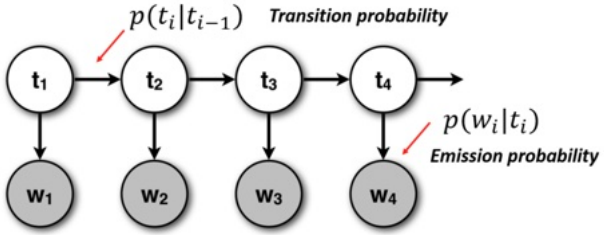
Example of Emission Probabilities

$t_i \backslash w_i$	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	. . .
. . .	. . .	. . .	. . .	. . .	. . .

- Probabilities estimated from tagged WSJ corpus
  - 0.0032% of proper nouns are Janet: P(Janet|NNP) = 0.000032
  - About half of determiners (DT) are “the”: P(the|DT)=0.506099
- Again, in full table, each row sum should equal to 1.

Graphical Model Diagram

- In **graphical model** notation, circles = random variables, and each arrow = a conditional probability factor in the joint likelihood:



→ = a lookup in the transition distribution,
↓ = a lookup in the emission distribution.

What can we do with this model?

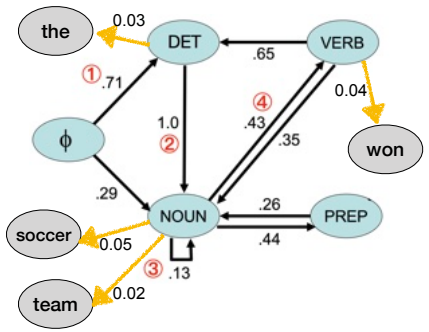
- If we know the transition and emission probabilities, we can compute the probability of a tagged sentence!
- Suppose we have sentence $w = (w_1, \ldots, w_n)$, and POS tags $t = (t_1, \ldots, t_n)$, what is the probability that our probabilistic finite-state machine would generate the exact (w, t) if we stepped through at random?
- This is the joint probability

$$P(w,t) = \prod_{i=1}^{n} P(t_i \mid t_{i-1}) P(w_i \mid t_i)$$

Example of Computing the Joint Prob

- What is the probability of this tagged sentence?

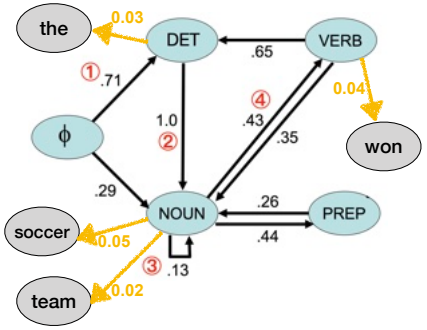
the/DET **soccer**/NOUN **team**/NOUN **won**/VERB



Example of Computing the Joint Prob

- What is the probability of this tagged sentence?

the/DET **soccer**/NOUN **team**/NOUN **won**/VERB



$$P(w,t) = \prod_{i=1}^{n} P(t_i \mid t_{i-1}) P(w_i \mid t_i)$$
$$= P(\text{DET} \mid \phi) P(\text{the} \mid \text{DET}) \cdot P(\text{NOUN} \mid \text{DET}) P(\text{soccer} \mid \text{NOUN}) \cdot P(\text{NOUN} \mid \text{NOUN}) P(\text{team} \mid \text{NOUN}) \cdot P(\text{VERB} \mid \text{NOUN}) P(\text{won} \mid \text{VERB})$$
$$= 0.71 * 0.03 * 1.0 * 0.05 * 0.13 * .02 * 0.43 * 0.04$$

Hidden Markov Model

- A **Hidden Markov Model (HMM)** is used to find the most likely sequence of hidden events given a sequence of observable events.
- The input tokens are the observed events.
- The class labels (states) are the **hidden** events.

  It is called hidden because at test time, we only see the words (emissions). The tags (states) are hidden variables.
- FSM view: given a sequence of words, what is the most probable state path that generated them?

Hidden Markov Model

Elements of an HMM:

- a set of states (here: tags)
- an output alphabet (here: words)
- initial state (here: beginning of sentence ϕ)
- state transition probabilities (here: $P(t_i \mid t_{i-1})$)
- symbol emission probabilities (here: $P(w_i \mid t_i)$)

Relationship to Previous Models

- **N-gram model**:

  a model for sequences that also makes a Markov assumption, but has no hidden variables.
- **Naive Bayes**:

  a model with hidden variables (the classes) but no sequential dependencies.
- **HMM**:

  a model for sequences with hidden variables.

Formalizing the Tagging Problem

- Find the best tag sequence t for an untagged sentence w

$$t^* = \arg\max_t P(t)P(w \mid t)$$
$$= \arg\max_t \prod_{i=1}^{n} P(t_i \mid t_{i-1})P(w_i \mid t_i)$$

- $P(t_i \mid t_{i-1})$ are the state transition probabilities
- $P(w_i \mid t_i)$ are the emission probabilities

Search for the Best Tag Sequence

- We have defined a model, but how do we use it?
  - given: word sequence w
  - wanted: best tag sequence t^*
- For any specific tag sequence t, it is easy to compute $P(t)P(w \mid t)$.
- Can we enumerate all possible t, compute their probabilities, and choose the best one?

Enumeration won't work

- Suppose we have C possible tags for each of the L words in the sentence.
- How many possible tag sequences?
- There are C^L possible tag sequences: the number grows exponentially in the length L.
- For all but small L, too many sequences to efficiently enumerate.

CS 5134/6034 Natural Language Processing
Fall 2024

Lecture 10: Viterbi

University of CINCINNATI

Formalizing the Tagging Problem

- Find the best tag sequence t for an untagged sentence w

$$t^* = \arg\max_t P(t)P(w \mid t)$$
$$= \arg\max_t \prod_{i=1}^{n} P(t_i \mid t_{i-1})P(w_i \mid t_i)$$

- $P(t_i \mid t_{i-1})$ are the state transition probabilities
- $P(w_i \mid t_i)$ are the emission probabilities

Hidden Markov Model

- A **Hidden Markov Model (HMM)** is used to find the most likely sequence of hidden events given a sequence of observable events.
- The input tokens are the **observed** events.
- The class labels (states) are the **hidden** events.

  It is called hidden because at test time, we only see the words (emissions). The tags (states) are hidden variables.
- FSM view: given a sequence of words, what is the most probable state path that generated them?

Hidden Markov Model

Elements of an HMM:

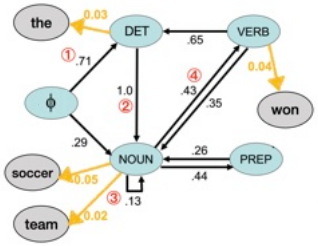
- a set of states (here: tags)
- an output alphabet (here: words)
- initial state (here: beginning of sentence ϕ)
- state transition probabilities (here: $P(t_i \mid t_{i-1})$)
- symbol emission probabilities (here: $P(w_i \mid t_i)$)

Using HMMs

We can use HMMs in two ways:

- **Likelihood**: given a sequence of observed events, compute the probability of the observed events

  E.g., given a sentence, compute the probability of the sentence, taking all tag possibilities into account.



$$P(w,t) = \prod_{i=1}^{n} P(t_i \mid t_{i-1}) P(w_i \mid t_i)$$
$$= P(\text{DET} \mid \text{<s>}) P(\text{the} \mid \text{DET}) \cdot$$
$$P(\text{NOUN} \mid \text{DET}) P(\text{soccer} \mid \text{NOUN}) \cdot$$
$$P(\text{NOUN} \mid \text{NOUN}) P(\text{team} \mid \text{NOUN}) \cdot$$
$$P(\text{VERB} \mid \text{NOUN}) P(\text{won} \mid \text{VERB})$$
$$= 0.71 * 0.03 * 1.0 * 0.05 * 0.13 * .02 * 0.43 * 0.04$$

Using HMMs

We can use HMMs in two ways:

- **Decoding**: given a sequence of observed events, find the most likely sequence of hidden events (tags)

  E.g., consider an HMM where words are the observed events and part-of-speech tags are the hidden events.

  Given a sentence, find the most likely sequence of part-of-speech tags for the words.

Search for the Best Tag Sequence

- We have defined a model, but how do we use it?
  - given: word sequence w
  - wanted: best tag sequence t^*
- For any specific tag sequence t, it is easy to compute $P(t)P(w \mid t)$.
- What are the naive methods?

Enumeration won't work

- Suppose we have C possible tags for each of the L words in the sentence.
- How many possible tag sequences?
- There are C^L possible tag sequences: the number grows exponentially in the length L.
- For all but small L, too many sequences to efficiently enumerate.

Greedy Algorithm won't Work

- Suppose for the i-th word w_i , we select $t_i = t_i^*$ such that

$$t_i^* = \arg\max_{t_i} P(t_i \mid t_{i-1}) P(w_i \mid t_i)$$

- Is this equivalent to

$$t^* = \arg\max_{t} \; P(t)P(w \mid t)$$
$$= \arg\max_{t} \; \prod_i P(t_i \mid t_{i-1}) P(w_i \mid t_i) \; ?$$

- Greedy algorithm may only achieve local optimal, not global.

Greedy vs. Dynamic Programming

- The greedy algorithm is fast. Runtime complexity?

  — $O(CL)$, with C tags, length-L sentence

- But it commits to a tag before seeing subsequent tags, so the results is likely to be suboptimal.

- The decoding problem has an **optimal substructure**.

  — "optimal substructure" means that an optimal solution can be constructed from optimal solutions of its subproblems.

- We can use **dynamic programming** to achieve an optimal solution.

The Viterbi Algorithm

- Intuition: the best path of length i ending in state t must include one of the best paths of length i-1 of different ending states. So,

- Find the best paths (why plural? ending in different states!) of length i-1 ending in all possible states

- Extending each of those by 1 step, to state t

- Select the best path of length i ending in state t

The Viterbi Algorithm

- Use a chart to store partial results as we go

  —C*L table, where $v(t, i)$ is the probability of the best state (tag) sequence for $(w_1, \ldots, w_i)$ that ends in state t

- Fill in columns from left to right, with

$$v(t, i) = \max_{t'} \; v(t', i-1) \cdot P(t \mid t') \cdot P(w_i \mid t)$$

- Store back pointers to remember which state we came from.

The Viterbi Algorithm

initial state prob $P(tag_s \mid \phi)$

function VITERBI(*observations* of len T,*state-graph* of len N) **returns** *best-path*, *path-prob*

create a path probability matrix *viterbi*[*N*,*T*]
for each state s **from** 1 **to** N **do** ; initialization step
  $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
  $backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$; termination step
$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T]$; termination step
$bestpath \leftarrow$ the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*

emission prob $b_s(o_t) = P(word_t \mid tag_s)$

transition prob $a_{s,s'} = P(tag_s \mid tag_{s'})$

Figure from the textbook

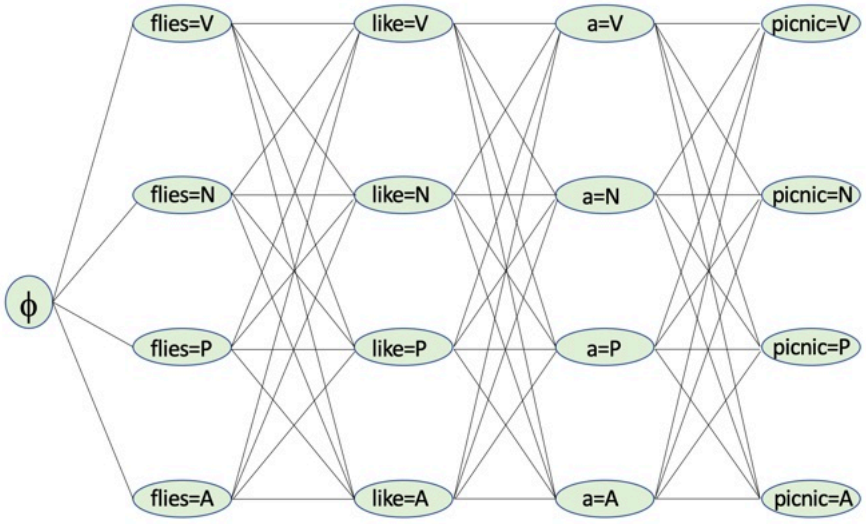
A Real Example of Viterbi

- Suppose we have 4 part-of-speech tags:

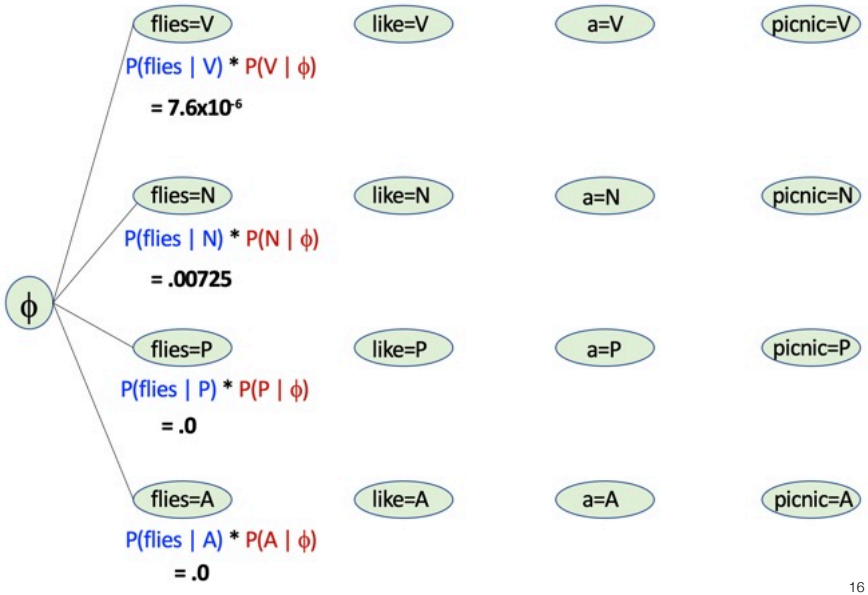
  N=noun, V=verb, P=preposition, A=article

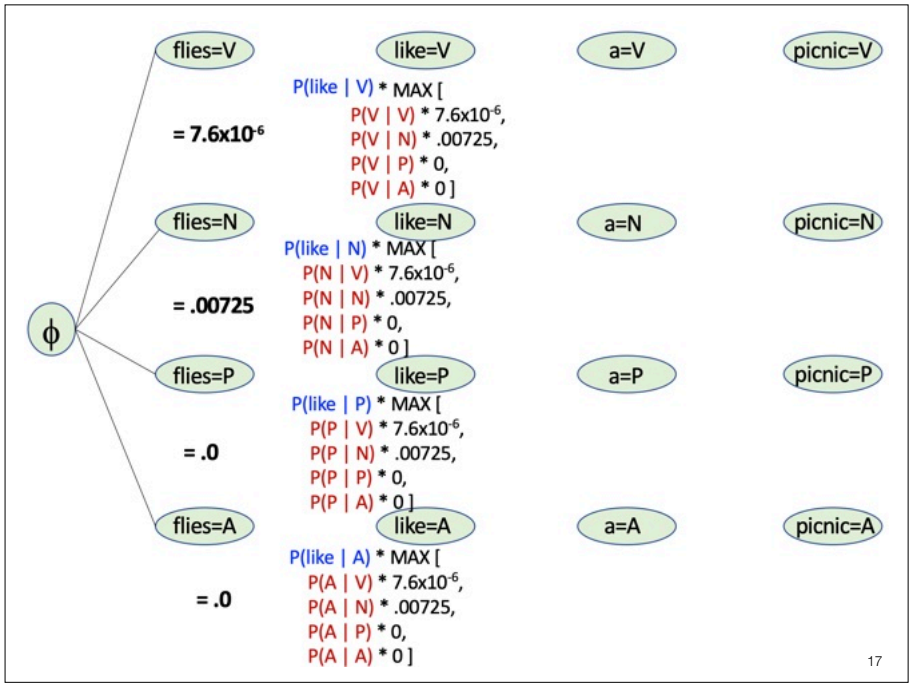
- The numbers are based on transition and emission probabilities that were made up for this example. They are not meant to be realistic.

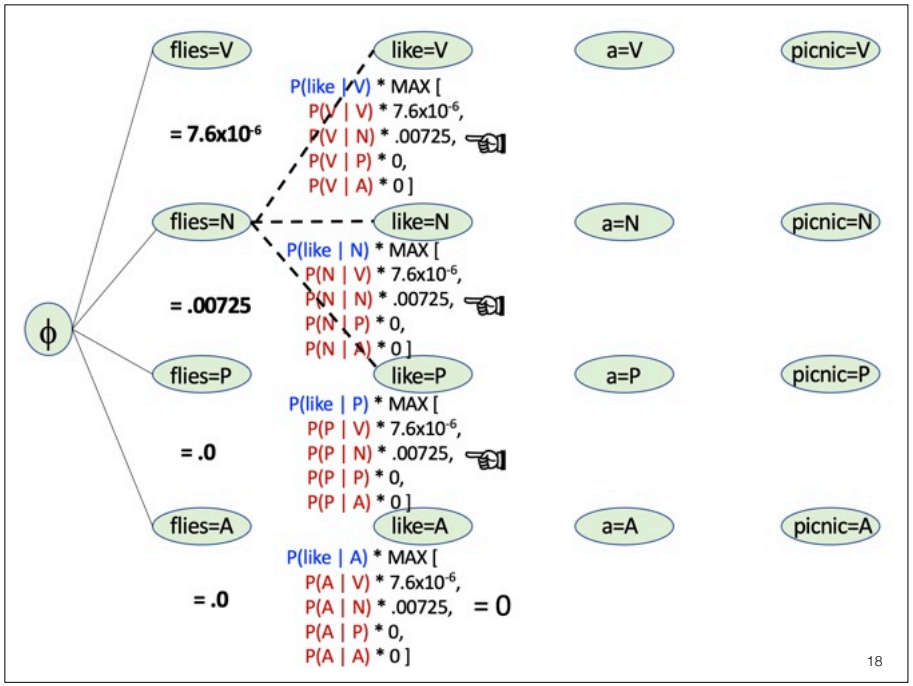
  In particular, some zero emission probabilities were included to keep the example simple, but in the real world you would use smoothing and not have zero probabilities.

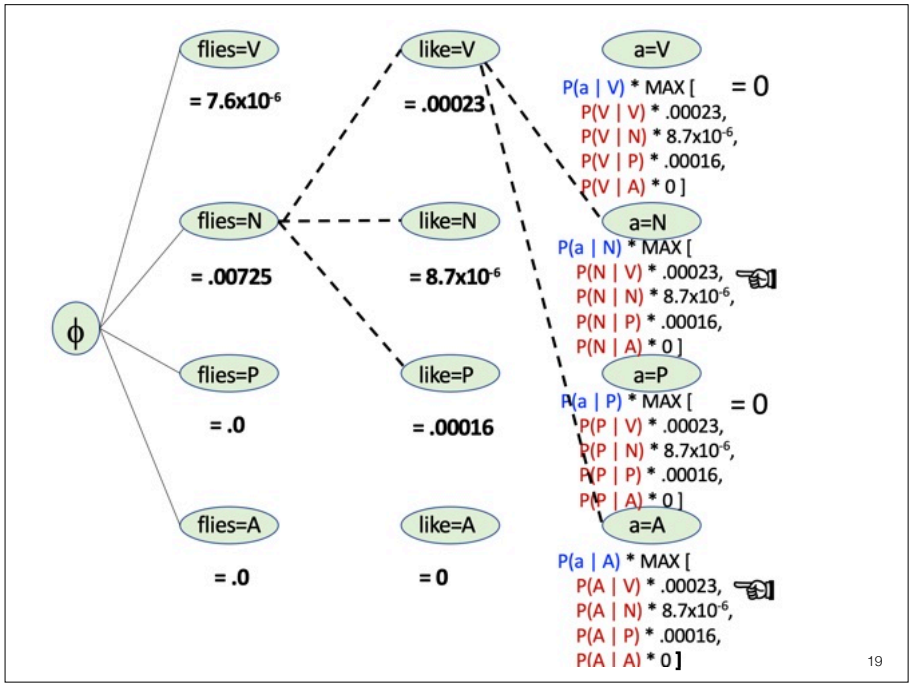


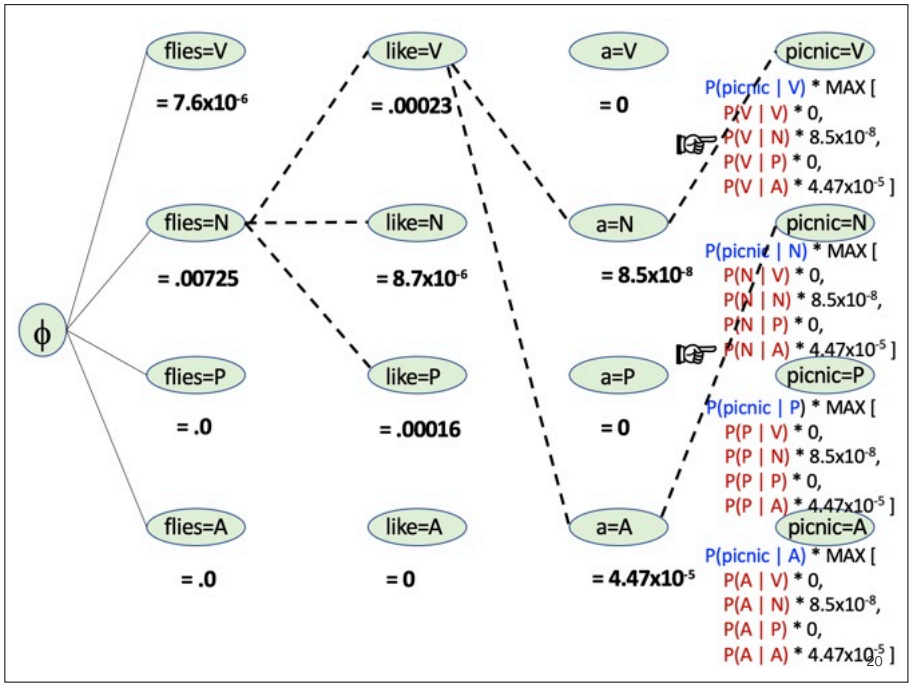
This represents our traversal through the HMM as we process each word!

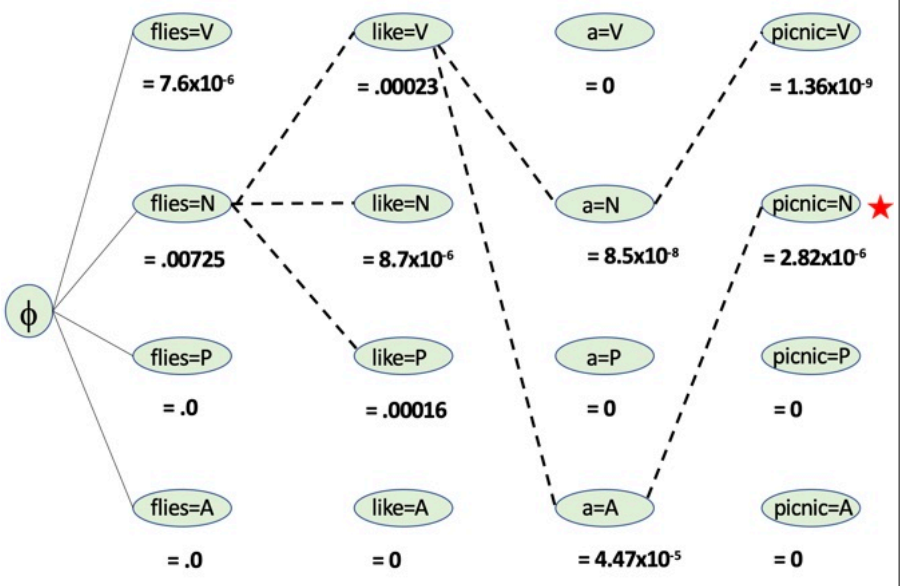


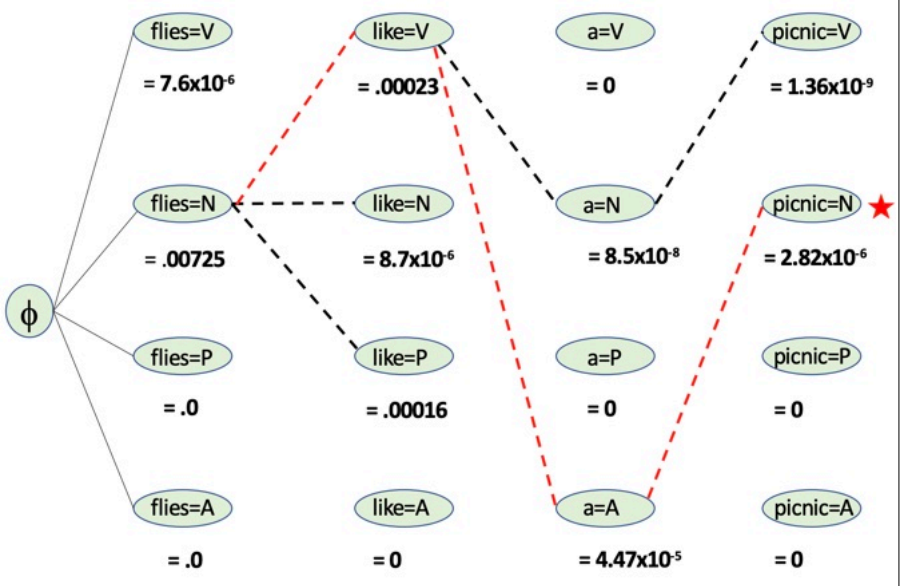

flies=V
like=V
a=V
picnic=V
= 7.6x10-6
P(like | V) * MAX [
P(V | V) * 7.6x10-6,
P(V | N) * .00725,
P(V | P) * 0,
P(V | A) * 0]
flies=N
like=N
a=N
picnic=N
= .00725
P(like | N) * MAX [
P(N | V) * 7.6x10-6,
P(N | N) * .00725,
P(N | P) * 0,
P(N | A) * 0]
flies=P
like=P
a=P
picnic=P
= .0
P(like | P) * MAX [
P(P | V) * 7.6x10-6,
P(P | N) * .00725,
P(P | P) * 0,
P(P | A) * 0]
flies=A
like=A
a=A
picnic=A
= .0
P(like | A) * MAX [
P(A | V) * 7.6x10-6,
P(A | N) * .00725,
P(A | P) * 0,
P(A | A) * 0]
17


flies=V
like=V
a=V
picnic=V
= 7.6x10-6
P(like | V) * MAX [
P(V | V) * 7.6x10-6,
P(V | N) * .00725,
P(V | P) * 0,
P(V | A) * 0]
flies=N
like=N
a=N
picnic=N
= .00725
P(like | N) * MAX [
P(N | V) * 7.6x10-6,
P(N | N) * .00725,
P(N | P) * 0,
P(N | A) * 0]
flies=P
like=P
a=P
picnic=P
= .0
P(like | P) * MAX [
P(P | V) * 7.6x10-6,
P(P | N) * .00725,
P(P | P) * 0,
P(P | A) * 0]
flies=A
like=A
a=A
picnic=A
= .0
P(like | A) * MAX [
P(A | V) * 7.6x10-6,
P(A | N) * .00725,
P(A | P) * 0,
P(A | A) * 0]
= 0
18


flies=V
like=V
a=V
= 7.6x10-6
= .00023
P(a | V) * MAX [
P(V | V) * .00023,
P(V | N) * 8.7x10-6,
P(V | P) * .00016,
P(V | A) * 0]
= 0
flies=N
like=N
a=N
= .00725
= 8.7x10-6
P(a | N) * MAX [
P(N | V) * .00023,
P(N | N) * 8.7x10-6,
P(N | P) * .00016,
P(N | A) * 0]
flies=P
like=P
a=P
= .0
= .00016
P(a | P) * MAX [
P(P | V) * .00023,
P(P | N) * 8.7x10-6,
P(P | P) * .00016,
P(P | A) * 0]
= 0
flies=A
like=A
a=A
= .0
= 0
P(a | A) * MAX [
P(A | V) * .00023,
P(A | N) * 8.7x10-6,
P(A | P) * .00016,
P(A | A) * 0]
19


flies=V
like=V
a=V
picnic=V
= 7.6x10-6
= .00023
= 0
P(picnic | V) * MAX [
P(V | V) * 0,
P(V | N) * 8.5x10-8,
P(V | P) * 0,
P(V | A) * 4.47x10-5]
flies=N
like=N
a=N
picnic=N
= .00725
= 8.7x10-6
= 8.5x10-8
P(picnic | N) * MAX [
P(N | V) * 0,
P(N | N) * 8.5x10-8,
P(N | P) * 0,
P(N | A) * 4.47x10-5]
flies=P
like=P
a=P
picnic=P
= .0
= .00016
= 0
P(picnic | P) * MAX [
P(P | V) * 0,
P(P | N) * 8.5x10-8,
P(P | P) * 0,
P(P | A) * 4.47x10-5]
flies=A
like=A
a=A
picnic=A
= .0
= 0
= 4.47x10-5
P(picnic | A) * MAX [
P(A | V) * 0,
P(A | N) * 8.5x10-8,
P(A | P) * 0,
P(A | A) * 4.47x10-5]
20





The Viterbi Algorithm

- The Viterbi algorithm is a general method for decoding tag (label) sequences.
- What is the time complexity if we have C tags, length-L sentence?

  $O(C^2L)$
- This algorithm is efficient because we only need to know the best sequence leading to the previous word thanks to the Markov assumption.

Using a POS Tagger in a Pipeline

- We could use a statistical POS tagger to determine the most likely part-of-speech for each word and give them to the parser as the definitive answer.
- This approach would maximally reduce the amount of lexical ambiguity that the parser needs to deal with because it would only receive one part-of-speech per word.
- But if the tagger made any mistakes, then the parser will be stuck with them!
- Remember that even if a tagger gets 95% accuracy, that still means 1 in 20 words is mistagged.

Probabilistic Tag Assignment

- Instead of labeling each word with exactly one tag, we could assign a set of possible tags to each word with associated probabilities for the downstream system to use.
- The simplest approach would be to use frequency counts to estimate probabilities:

$$P(t_i \mid w_i) = \frac{\text{\# times } w_i \text{ occurs with } t_i}{\text{\# times } w_i \text{ occurs with any tag}}$$

- A better approach considers how likely a tag is for a word given the **surrounding** words in the sentence.

The Forward Probability

- The forward probability $\alpha_t(s)$ is the probability of being in state s after seeing the first t observed events.
- Intuitively, it is the sum of the probabilities over all state sequences that could lead to state s for observations (words) $o_1, \ldots, o_t$:

$$\alpha_t(s) = P(o_1, \ldots, o_t, \text{state}(o_t) = s)$$

- We sequentially process the input, sweeping over each possible state for o_t . For each state s , we sum over all possible prior states of (the forward probability of the prior state) * (the transition between the states) * (the emission probability of o_t):

$$\alpha_t(s) = \sum_{s'=1}^{N} \alpha_{t-1}(s') a_{s',s} b_s(o_t)$$

The Forward Probability

function FORWARD(*observations* of len T, *state-graph* of len N) **returns** *forward-prob*

create a probability matrix *forward*[N,T]
for each state s **from** 1 **to** N **do** ; initialization step
  $forward[s,1] \leftarrow \pi_s * b_s(o_1)$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**
    $forward[s,t] \leftarrow \sum_{s'=1}^{N} forward[s',t-1] * a_{s',s} * b_s(o_t)$
$forwardprob \leftarrow \sum_{s=1}^{N} forward[s,T]$; termination step
return *forwardprob*

SUM, not MAX

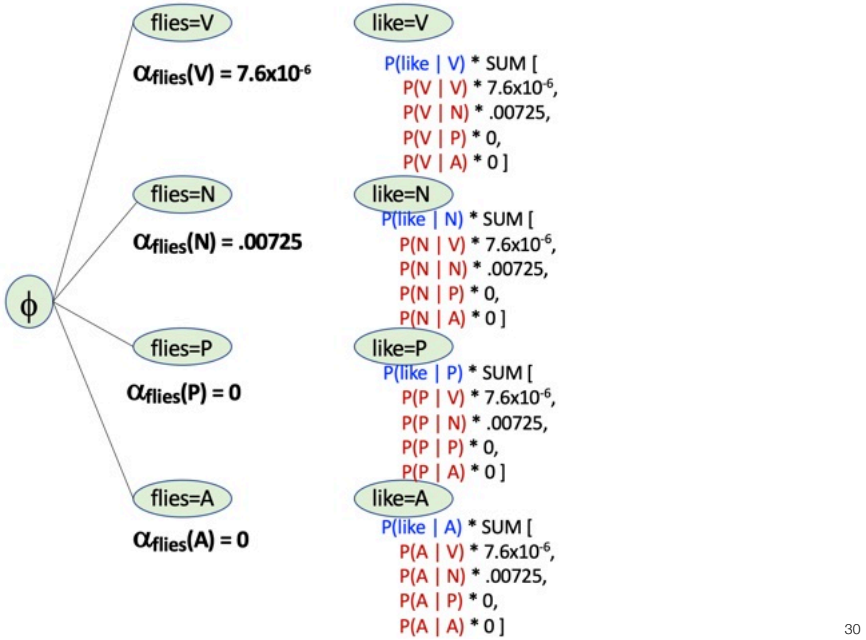
This is nearly identical to the Viterbi algorithm, but sums over the probabilities instead of taking the Max.

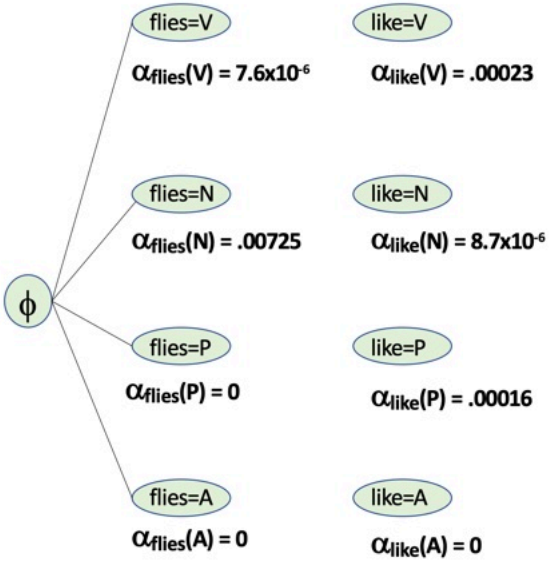
Normalization for Final Probabilities

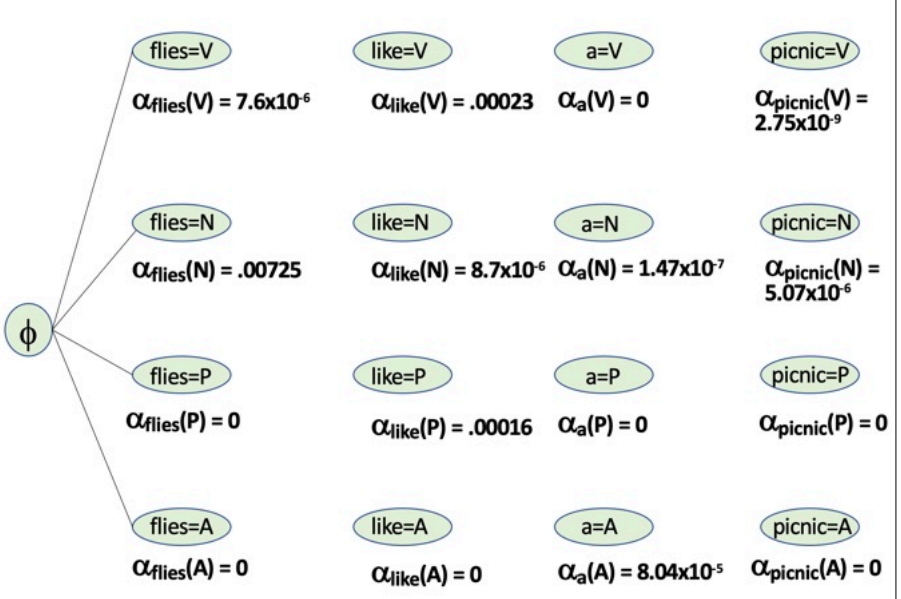
- To compute the final probability of each tag, we normalize the forward probabilities in each column of the matrix so the probabilities sum to 1.
- The probability of a word o_t having state s would be computed as follows:

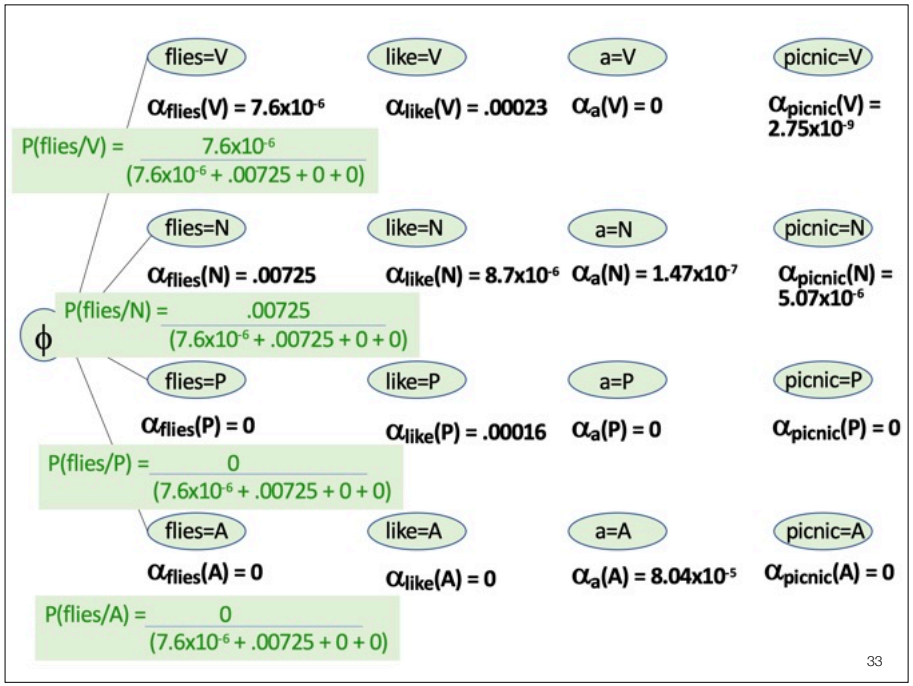
$$P(\text{state}(o_t) = s) = \frac{\alpha_t(s)}{\sum_{s'=1}^{N} \alpha_t(s')}$$

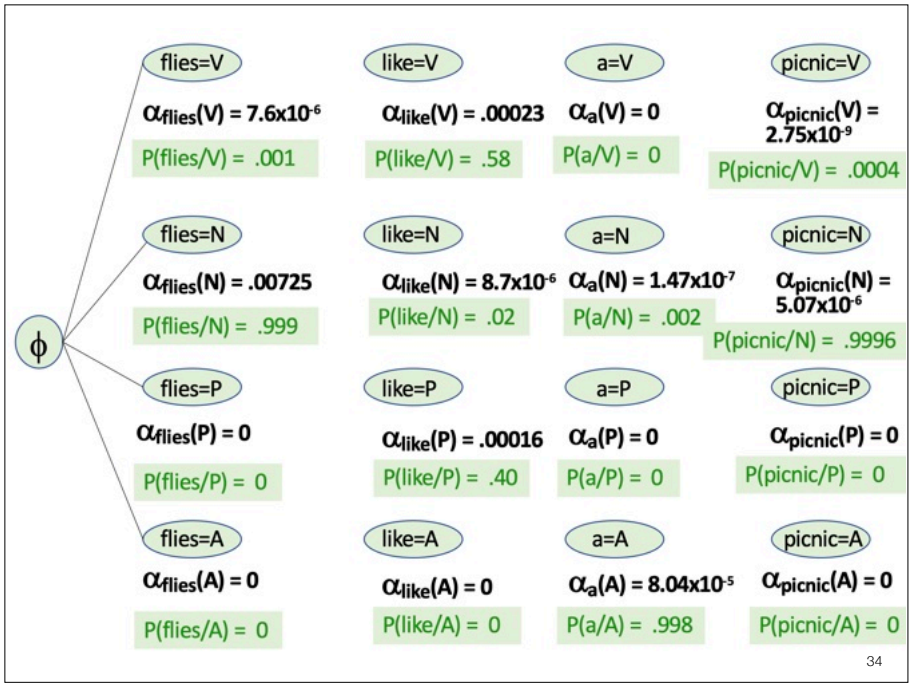

flies=V
P(flies | V) * P(V | ϕ)
$\alpha_{flies}(V) = 7.6x10^{-6}$
flies=N
P(flies | N) * P(N | ϕ)
$\alpha_{flies}(N) = .00725$
flies=P
P(flies | P) * P(P | ϕ)
$\alpha_{flies}(P) = 0$
flies=A
P(flies | A) * P(A | ϕ)
$\alpha_{flies}(A) = 0$
ϕ


flies=V
$\alpha_{flies}(V) = 7.6x10^{-6}$
like=V
P(like | V) * SUM [
P(V | V) * $7.6x10^{-6}$,
P(V | N) * .00725,
P(V | P) * 0,
P(V | A) * 0]
flies=N
$\alpha_{flies}(N) = .00725$
like=N
P(like | N) * SUM [
P(N | V) * $7.6x10^{-6}$,
P(N | N) * .00725,
P(N | P) * 0,
P(N | A) * 0]
flies=P
$\alpha_{flies}(P) = 0$
like=P
P(like | P) * SUM [
P(P | V) * $7.6x10^{-6}$,
P(P | N) * .00725,
P(P | P) * 0,
P(P | A) * 0]
flies=A
$\alpha_{flies}(A) = 0$
like=A
P(like | A) * SUM [
P(A | V) * $7.6x10^{-6}$,
P(A | N) * .00725,
P(A | P) * 0,
P(A | A) * 0]
ϕ


flies=V
$\alpha_{flies}(V) = 7.6x10^{-6}$
like=V
$\alpha_{like}(V) = .00023$
flies=N
$\alpha_{flies}(N) = .00725$
like=N
$\alpha_{like}(N) = 8.7x10^{-6}$
flies=P
$\alpha_{flies}(P) = 0$
like=P
$\alpha_{like}(P) = .00016$
flies=A
$\alpha_{flies}(A) = 0$
like=A
$\alpha_{like}(A) = 0$
ϕ


flies=V
$\alpha_{flies}(V) = 7.6x10^{-6}$
like=V
$\alpha_{like}(V) = .00023$
a=V
$\alpha_a(V) = 0$
picnic=V
$\alpha_{picnic}(V) = 2.75x10^{-9}$
flies=N
$\alpha_{flies}(N) = .00725$
like=N
$\alpha_{like}(N) = 8.7x10^{-6}$
a=N
$\alpha_a(N) = 1.47x10^{-7}$
picnic=N
$\alpha_{picnic}(N) = 5.07x10^{-6}$
flies=P
$\alpha_{flies}(P) = 0$
like=P
$\alpha_{like}(P) = .00016$
a=P
$\alpha_a(P) = 0$
picnic=P
$\alpha_{picnic}(P) = 0$
flies=A
$\alpha_{flies}(A) = 0$
like=A
$\alpha_{like}(A) = 0$
a=A
$\alpha_a(A) = 8.04x10^{-5}$
picnic=A
$\alpha_{picnic}(A) = 0$
ϕ





The Backward Probability

- The forward probability $\beta_t(s)$ is the probability of seeing the observations from time t+1 to the end, given that we are in state s at time t.
- Intuitively, it is the sum of the probabilities over all state sequences that could lead to state s for observations (words) $o_{t+1}, \ldots, o_T$:

$$\beta_t(s) = P(o_{t+1}, \ldots, o_T \,|\, \text{state}(o_t) = s)$$

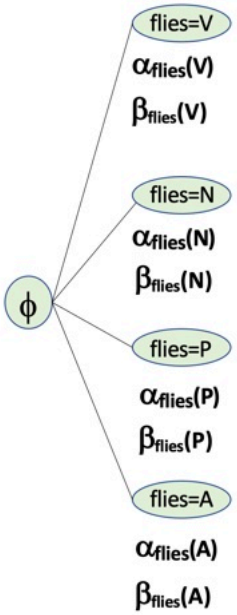
- Initialization from the end of sequence! Similar to beginning of sentence ϕ, we can define end of sentence Ω and initialize $\beta_T(s) = a_{s,\Omega}$.
- The algorithm for computing backward probabilities is analogous to algorithm for forward probabilities, except we sweep from right to left:

$$\beta_t(s) = \sum_{s'=1}^{N} \beta_{t+1}(s') a_{s,s'} b_s(o_{t+1})$$

Using Both Forward and Backward Probabilities

- The most accurate estimates will come from using both the forward and backward probabilities.
- Intuitively, this approach takes into account both the context prior to a word and the context following it.
- Compute all the forward and backward probabilities as described earlier, then normalize the nodes in each column so they sum to 1.

$$P(\text{state}(o_t) = s) = \frac{\alpha_t(s)\beta_t(s)}{\sum_{s'=1}^{N} \alpha_t(s')\beta_t(s')}$$

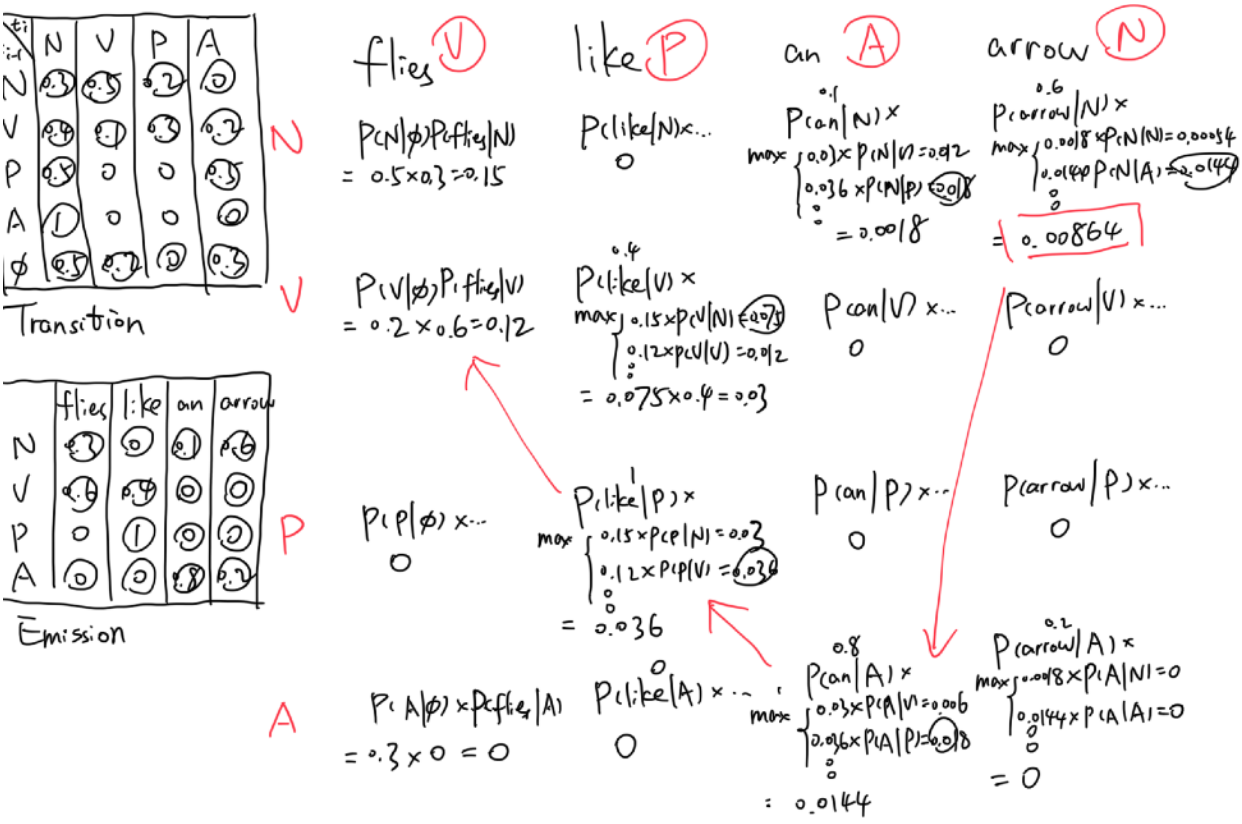


$$P(flies/V) = \frac{\alpha_{flies}(V) * \beta_{flies}(V)}{(\alpha_{flies}(V) * \beta_{flies}(V) + \alpha_{flies}(N) * \beta_{flies}(N) + \alpha_{flies}(P) * \beta_{flies}(P) + \alpha_{flies}(A) * \beta_{flies}(A))}$$

$$P(flies/N) = \frac{\alpha_{flies}(N) * \beta_{flies}(N)}{(\alpha_{flies}(V) * \beta_{flies}(V) + \alpha_{flies}(N) * \beta_{flies}(N) + \alpha_{flies}(P) * \beta_{flies}(P) + \alpha_{flies}(A) * \beta_{flies}(A))}$$

$$P(flies/P) = \frac{\alpha_{flies}(P) * \beta_{flies}(P)}{(\alpha_{flies}(V) * \beta_{flies}(V) + \alpha_{flies}(N) * \beta_{flies}(N) + \alpha_{flies}(P) * \beta_{flies}(P) + \alpha_{flies}(A) * \beta_{flies}(A))}$$

$$P(flies/A) = \frac{\alpha_{flies}(A) * \beta_{flies}(A)}{(\alpha_{flies}(V) * \beta_{flies}(V) + \alpha_{flies}(N) * \beta_{flies}(N) + \alpha_{flies}(P) * \beta_{flies}(P) + \alpha_{flies}(A) * \beta_{flies}(A))}$$


Transition
Emission
flies
like
an
arrow
N
V
P
A
P(N|ϕ)P(flies|N) = 0.5×0.3 = 0.15
P(like|N)×... = 0
P(can|N)× max{0.0×P(N|V)=0.92, 0.036×P(N|P)=0.018, ...} = 0.0018
P(arrow|N)× max{0.0018×P(N|N)=0.00054, 0.0144×P(N|A)=0.0144, ...} = 0.00864
P(V|ϕ)P(flies|V) = 0.2×0.6 = 0.12
P(like|V)× max{0.15×P(V|N)=0.075, 0.12×P(V|V)=0.012, ...} = 0.075×0.4 = 0.03
P(can|V)×... = 0
P(arrow|V)×... = 0
P(P|ϕ)×... = 0
P(like|P)× max{0.15×P(P|N)=0.03, 0.12×P(P|V)=0.036, ...} = 0.036
P(can|P)×... = 0
P(arrow|P)×... = 0
P(A|ϕ)×P(flies|A) = 0.3×0 = 0
P(like|A)×... = 0
P(can|A)× max{0.03×P(A|V)=0.006, 0.036×P(A|P)=0.018, ...} = 0.0144
P(arrow|A)× max{0.0018×P(A|N)=0, 0.0144×P(A|A)=0, ...} = 0

CS 5134/6034 Natural Language Processing
Fall 2024

Lecture 12: Sequence Labeling

University of CINCINNATI

Shallow Parsing

- Instead of generating a complete parse tree for a sentence, **shallow parsers** generate fragments representing local syntactic constituents. (This approach is sometimes also called **partial parsing** or **syntactic chunking**.)
- Shallow parsers typically try to identify NPs, VPs, and PPs (and occasionally other constituents).
- These local syntactic constituents can be identified (relatively) reliably using simple grammar rules and heuristics.
- Some shallow parsers use finite state machines to recognize a regular grammar.

Shallow Parsing Example

- Shallow parsers usually produce a flat syntactic representation of **non-recursive** constituents (sometimes called "chunks").
- You can build a shallow parser based on rules, or train a classifier using **BIO labeling**.

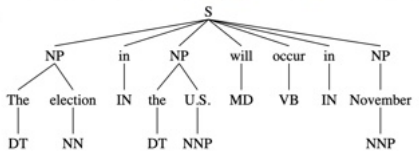
The election in the U.S. will occur in November

1. [NP: The election] in [NP: the U.S.] will occur in [NP: November]
2. [NP: The election] [PP: in the U.S.] will occur [PP: in November]
3. [NP: The election] [PP: in the U.S.] [VP: will occur] [PP: in November]

Rule-based: NP Only

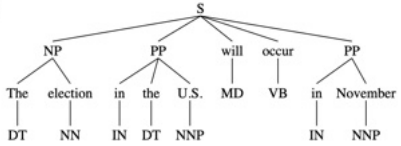
```
[1]: import nltk
     sentence = 'The election in the U.S. will occur in November'

[2]: nltk.word_tokenize(sentence)

[2]: ['The', 'election', 'in', 'the', 'U.S.', 'will', 'occur', 'in', 'November']

[3]: nltk.pos_tag(nltk.word_tokenize(sentence))

[3]: [('The', 'DT'),
      ('election', 'NN'),
      ('in', 'IN'),
      ('the', 'DT'),
      ('U.S.', 'NNP'),
      ('will', 'MD'),
      ('occur', 'VB'),
      ('in', 'IN'),
      ('November', 'NNP')]

[4]: grammar = """NP: {<DT>?<JJ>*<NN.*>}"""
     shallow_parser = nltk.RegexpParser(grammar)
     shallow_parser.parse(nltk.pos_tag(nltk.word_tokenize(sentence)))
```

[4]:



Rule-based: NP + PP

```
[1]: import nltk
     sentence = 'The election in the U.S. will occur in November'

[2]: nltk.word_tokenize(sentence)

[2]: ['The', 'election', 'in', 'the', 'U.S.', 'will', 'occur', 'in', 'November']

[3]: nltk.pos_tag(nltk.word_tokenize(sentence))

[3]: [('The', 'DT'),
      ('election', 'NN'),
      ('in', 'IN'),
      ('the', 'DT'),
      ('U.S.', 'NNP'),
      ('will', 'MD'),
      ('occur', 'VB'),
      ('in', 'IN'),
      ('November', 'NNP')]

[4]: grammar = """
     PP: {<IN><DT>?<JJ>*<NN.*>}
     NP: {<DT>?<JJ>*<NN.*>}"""
     shallow_parser = nltk.RegexpParser(grammar)
     shallow_parser.parse(nltk.pos_tag(nltk.word_tokenize(sentence)))

[4]:
```



Benefits of Shallow Parsing

- Shallow parsers are usually much faster than full parsers.
- Shallow parsers are more robust with ungrammatical input.

  E.g., twitter, speech.
- Recognizing deep syntactic structure may not be necessary for some NLP applications.
  - Some ambiguity issues can be ignored if they are not critical for identifying the syntactic chunks (e.g., PP attachment).
  - Some structural issues can be delayed and left for semantic analysis.

Weaknesses of Shallow Parsing

- Attachments are usually not attempted.
- Does not represent embedded relative clauses.

  E.g., *I gave the boy **that was sick** some medicine.*
- Often has trouble recognizing reduced relative clauses.

  E.g., *The woman **killed last night** was an important diplomat.*
- Syntactic roles (e.g., subject, direct object) generally are not assigned. Full parsers do not always assign these roles either, but it is usually straightforward to do so with post-processing with heuristics.

Named Entity Recognition

Named Entity Recognition (NER) systems identify specific types of entities, primarily proper named entities and special categories:

- **Proper Names:** people, organizations, locations, etc.

  Elvis Presley, IBM, Department of the Interior, Centers for Disease Control
- **Dates & Times:** ubiquitous and surprisingly varied

  November 9, 1997, 11/9/97, 10:29 pm
- **Measures:** measurements with specific units

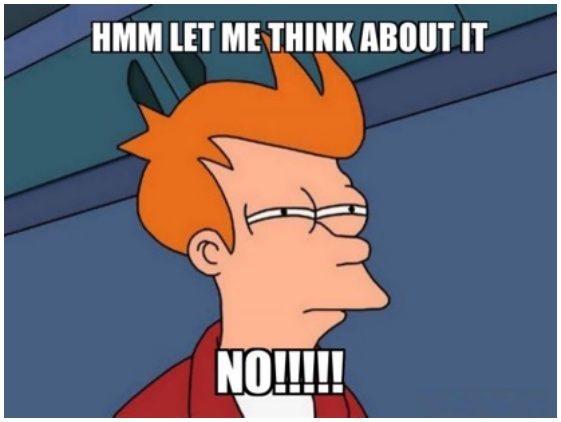
  45%, 5.3 lbs, 65 mph, $1.4 billion
- **Other:** Application-specific stylized terms.

  URLs, email addresses, phone numbers, social security numbers

Challenges

- No dictionary will contain all existing proper names. New names are constantly being created.
- Just finding the proper names isn't trivial! Why?
  1. The first word of every sentence is capitalized.
  2. Proper names can include lower case words (e.g., University **of** Cincinnati).
  3. Some texts are not written in mixed case (e.g., some headlines)
  4. Spoken language doesn't have case! (And some languages don't signal proper names with case.)
  5. Proper names are often abbreviations or acronyms. But not all acronyms are proper names (e.g., NLP, AI, CS, OS).

HUMOR



IS HMM A NAMED ENTITY?

Proper Name Ambiguity

- Many companies, organizations, and locations are named after people!

  Companies: Ford, ZEISS, Calvin Klein, Air Jordan

  Universities: Stanford, McGill, Purdue, National Sun Yat-sen University

  Locations: JFK (airport), Washington (capital, state, county, etc.)
- Acronyms can often refer to many different things

  ACL: Association for Computational Linguistics vs. *anterior cruciate ligament*

  UC: University of Cincinnati vs. University of California vs. Universal Control
- Many proper names can have multiple semantic types

  — *April, June, Georgia, Jordan*

Rule-based vs. Machine Learning

- Hand-coded rules

  Good: Can perform best, especially for specialized domains.

  Bad: Expensive to build. Domain-specific.
- Machine Learning

  Good: Can be easily adapted for new domains.

  Bad: Need annotated training corpus.

Common Types of Rules

- Matching against lists (e.g., common person names, organization lists, location gazetteers, etc.) **But very important to scan the lists and remove problematic entries!**
- Leading/trailing terms representing titles or designations (e.g., Prof., Mr., Ms., Jr., Co., Inc.) or special symbols (e.g., $ for money).
- Surface structure patterns, for instance:
  - dates (e.g., Month/Day/Year)
  - phone numbers (e.g., (###) ###-####)
  - email addresses (e.g., xxxxx@xxx.xxx.edu)
- Contextual patterns. For example, locations frequently occur in contexts such as "headquarters in X", "lives in X", "moved to X", etc.

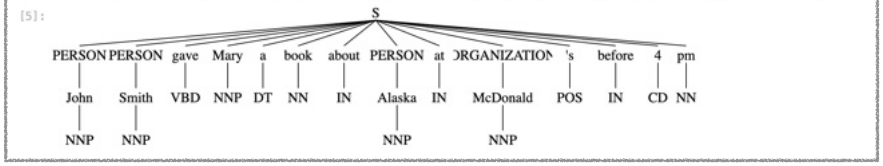
Machine Learning Models

- Many NLP problems can be formulated as **sequence labeling** tasks. Sequence labeling methods are appropriate for problems where the label of an item depends on other (typically nearby) items in the sequence.

  Example tasks: part-of-speech tagging, shallow parsing, named entity recognition, opinion extraction.
- NER models can be created with:
  - Hidden Markov Model (HMM)
  - Maximum-entropy Markov Model (MEMM)
  - Conditional Random Fields (CRF)

NER Example

```
[5]: nltk.ne_chunk(nltk.pos_tag(nltk.word_tokenize("John Smith gave Mary a book about Alaska at McDonald's before 4 pm")))
```




```
[6]: import spacy
     nlp = spacy.load("en_core_web_trf")

     text = "John Smith gave Mary a book about Alaska at McDonald's before 4 pm"
     doc = nlp(text)
     spacy.displacy.render(doc, style="ent")
```

John Smith PERSON gave Mary PERSON a book about Alaska GPE at McDonald's ORG before 4 pm TIME

Sequential Tagging for NER

- When named entity recognition is viewed as a classification or sequential tagging problem, every word is labeled based on whether it is part of a named entity.
- The most common labeling scheme is **BIO tagging (short for beginning, inside, outside)**.

  Example: "John Smith gave Mary a book about Alaska"

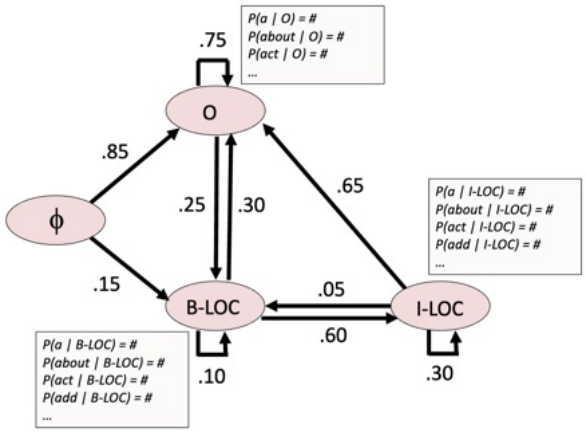
  John/B Smith/I gave/O Mary/B a/O book/O about/O Alaska/B
- A different classifier may be created for each entity type.

  The/O University/B_{ORG} of/I_{ORG} Cincinnati/I_{ORG} hired/O Dr./B_{PER} Susan/I_{PER} Miller/I_{PER} on/O Tuesday/B_{DATE} .

HMMs for NER

- Hidden Markov Models (HMMs) can be used for Named Entity Recognition, exactly as they were used for POS Tagging.
- Given a training corpus of text labeled with Named Entity (NE) tags, tables can be created for emission probabilities $P(word_i \mid tag_i)$ and transition probabilities $P(tag_i \mid tag_{i-1})$. The tags are the NE labels.
- The Viterbi algorithm can then be used for decoding to find the most probable sequence of NE labels.
- A limitation of this approach though is that it cannot use arbitrary features.

HMM Example for NER Tagging of Locations



Maximum Entropy Markov Models (MEMM)

- Given a sentence $w^{(1)}, \ldots, w^{(n)}$, we want to find the tag sequence $t^{(1)}, \ldots, t^{(n)}$ that maximize $P(t^{(1)}, \ldots, t^{(n)} \mid w^{(1)}, \ldots, w^{(n)})$ directly. We assume:

$$P\left(t^{(1)}, \ldots, t^{(n)} \mid w^{(1)}, \ldots, w^{(n)}\right) = \prod_{i=1}^{n} P\left(t^{(i)} \mid t^{(i-1)}, w^{(i)}\right)$$

- MEMMs use a logistic regression ("Maximum Entropy") classifier for each $P(t^{(\mathrm{i})} \mid w^{(\mathrm{i})}, t^{(\mathrm{i}-1)})$. Recall multinomial logistic regression?

$$P(y^{(j)} \mid x) = \text{softmax}\left(\theta_0^{(j)} + \sum_{i=1}^{n} \theta_i^{(j)} \cdot x_i\right)$$

Maximum Entropy Markov Models (MEMM)

- MEMMs use a logistic regression ("Maximum Entropy") classifier for each $P(t^{(\mathrm{i})} \mid w^{(\mathrm{i})}, t^{(\mathrm{i}-1)})$:

$$P\left(t^{(i)} = t_k \mid t^{(i-1)}, w^{(i)}\right) = \frac{\exp\left(\sum_j \theta_{jk} f_j\left(t^{(i-1)}, w^{(i)}\right)\right)}{\sum_l \exp\left(\sum_j \theta_{jl} f_j\left(t^{(i-1)}, w^{(i)}\right)\right)}$$

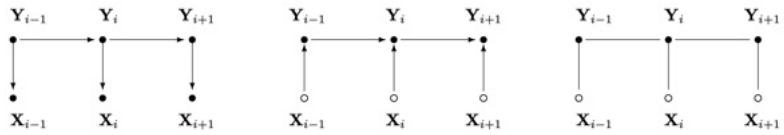
  where $t^{(i)}$ is the label of the i-th word vs. t_i is the i-th label in the label list.

- This requires the definition of a feature function $f\left(t^{(i-1)}, w^{(i)}\right)$ that returns an n-dimensional feature vector.
- Training weights λ_{jk} for each feature j used to predict label t_k .

Conditional Random Fields (CRF)

- MEMMs are trained **locally** to maximize the probability of each individual state (label).
- Conditional Random Fields have a similar mathematical definition as MEMMs, but trained **globally** to maximize the probability of the whole sequence.
- Training is slower, but avoid the label bias problem.

HMM vs. MEMM vs. CRF



HMM **MEMM** **CRF**

- An open circle indicates that the variable is not generated by the model.
- HMM is a generative model.
- MEMM and CRF are discriminative models.

A Maximum Entropy System for NER

- The MENERGI system is a nice example of a maximum entropy approach to NER:

  Named Entity Recognition: A Maximum Entropy Approach Using Global Information [Chieu & Ng, 2002]

- Good NER systems typically use a large set of local features based on properties of the target word and its neighboring words.
- This research also incorporates global features that capture information from across the document.

NER Types and Tagging Scheme

MENERGI recognizes 7 types of named entities, based on the MUC-6/7 NER task definition.

Person
Organization
Location
Date
Time
Money
Percent

Each class is subdivided into 4 sub-classes:

- Begin/Continue/End

the	University	of	Cincinnati
B_{ORG}	C_{ORG}	C_{ORG}	E_{ORG}

- Unique

Ohio
U_{LOC}

- How many classes in total?

  4 subclasses for each NE class 4*7

\+ one outside tag (not a NE) = 29

Illegal Tag Sequence

- One problem with NER classifiers is that they can produce inadmissible (illegal) tag sequences.

For example,

I	love	Ohio
O	O	E_{LOC}

- To eliminate this problem, they defined transition probabilities between classes $P(c_i \mid c_{i-1})$ to be 1 if the sequence is admissible or 0 if it is illegal.
- The probability of classes $c_1, \ldots, c_n$ assigned to the words in a sentence s in a document D is defined as:

$$P\left(c_1, \ldots, c_n \mid s, D\right) = \prod_{i=1}^{n} P\left(c_i \mid s, D\right) * P\left(c_i \mid c_{i-1}\right)$$

Feature Set

- The classifier uses one set of **local** features, which are based on properties of the target word w, the word on its left w_{-1}, and the word on its right w_{+1}.
- The classifier also uses a set of **global** features, which are extracted from instances of the same token that occur elsewhere in the document.
- Features that occur infrequently in the training set are discarded as a form of **feature selection**.

External Dictionaries

- Several external dictionaries were created by compiling lists of locations, companies, and person names.

Location Names	www.timeanddate.com
	www.cityguide.travel-guides.com
	www.worldtravelguide.net
Corporate Names	www.fmlx.com
Person Names	www.census.gov

- Large lists are easy to obtain and can really help an NER system.

Local Features

- strings of the target, previous, and next words
- zone of the word (headline, dateline, DD, or main text)
- capitalization-based features
- is it first word of the sentence?
- is it in WordNet? (OOV = out-of-vocabulary feature)
- presence of the target, previous, and next words in dictionaries
- is it a month, day, or number?
- is it preceded/followed by an NE class prefix/suffix term?
- 10 features that look for specific characters in the current word string (next slide)

Local Features

Token satisfies	Example	Feature
Starts with a capital letter, ends with a period	*Mr.*	*InitCap-Period*
Contains only one capital letter	*A*	*OneCap*
All capital letters and period	*CORP.*	*AllCaps-Period*
Contains a digit	*AB3, 747*	*Contain-Digit*
Made up of 2 digits	*99*	*TwoD*
Made up of 4 digits	*1999*	*FourD*
Made up of digits and slash	*01/01*	*Digit-slash*
Contains a dollar sign	*US$20*	*Dollar*
Contains a percent sign	*20%*	*Percent*
Contains digit and period	*$US3.20*	*Digit-Period*

Ambiguous Contexts

- Some entities occur in ambiguous sentences that can be confusing even for human readers without context.

McCann initiated a new global system.

The CEO of McCann announced... (=> McCann is a company)

The McCann family announced... (=> McCann is a person)

Liz Claiborne recently purchased Shoes R Us for $1.3 milllion.

She bought the shoe retailer to begin franchising it nationwide.

The company bought the shoe retailer to expand its product line.

Global Features

- Traditionally, NER systems classified each word/phrase independently of other instances of the same word/phrase in other parts of the document.
- But other contexts may provide valuable clues about what type of entity it is. For example:
  - capitalization is indicative if not the first word of a sentence
  - some contexts contain strong prefixes/suffixes in a phrase
  - some contexts contain strong preceding/following neighbors
  - acronyms can often be aligned with their expanded phrase

Global Features

- if another occurrence of the word appears in an unambiguous position (not first word), is it capitalized?

  E.g., “Apple was introduced …” + “… by Apple”

- do other occurrences of the word occur with a known named entity prefix/suffix?

  E.g., Mr. McCann, Apple Inc.

- if the word looks like an acronym, is there a capitalized sequence of words anywhere with these leading letters? If so, acronym features are assigned to the likely acronym word and the corresponding word sequence.

  E.g., FCC and Federal Communications Commission are both found in a document.

- for capitalized word sequences, the longest substrings that appear elsewhere are assigned features.

  E.g., in the sentence, “Even News Broadcasting Corp., noted for its accurate reporting, made the erroneous announcement”, we do not want “Even” in the name.

NER Summary

- Named Entity Recognition can perform quite well! NER is one of the most successful subproblems in NLP.
- But it's not perfect ... named entities are so common that even 95% accuracy means that there are still a lot of mistakes.
- And ... NER systems often need to be retrained for specific domains to perform well. Mistagging one common entity can be a disaster. (Imagine always tagging "Jordan" as a country in basketball stories)
- NER performs substantially worse if the text is not mixed case.
- Beware of downloading large lists without checking the entries. Ambiguous terms must be filtered or they will cause you grief!

CS 5134/6034 Natural Language Processing
Fall 2024

Lecture 13: Lexical Semantics

University of CINCINNATI

Word Meanings

- So far we have learned:

  the **structure** of words → morphology

  the **distribution** of words → n-gram language models

- In this lecture, we will focus on:

  the **meaning** of words/phrases → lexical semantics

How to get meanings of words?

- Dictionary

  The **lexicographic** approach understands words from dictionaries, lexicons, ontologies, etc.

- Large Raw Text

  The **distributional** approach understands words based on large text corpus.

Word Senses

- **Word forms**

  The inflected word as it appears in text

  e.g., read, reads, reading

- **Lemma**

  A basic word form used to represent all forms of the same word

- **Word Sense**

  A discrete representation of an aspect of a word's meaning

Homonymy and Polysemy

- **Homonymy**

  words having the same spelling or pronunciation but unrelated, different meanings

  plane[1]: airplane

  plane[2]: flat surface

- **Polysemy**

  A word is polysemous if it has different related meanings

  bank[1]: a financial institution

  bank[2]: the physical building where a financial institution offers services

Metonymy

Some senses of a word are related in a systematic way (systematic polysemy):

- container vs. content

  "sip the glass"

- producer vs. product

  "listen to Michael Jackson"

- organization vs. building

  "went to Walmart"

- capital city vs. government

  "Moscow announced a partial mobilization…"

  …

Synonym and Antonym

- **Synonym**

  Words with the **same** meaning in some contexts.

- **Antonym**

  Words with the **opposite** meaning.

- They are relations between **senses** rather than words.

  How big/large is that plane?

  John has a big sister →? John has a large sister

Hypernym and Hyponym

- One sense is a **hyponym** of another if the first sense is more specific, denoting a subclass of the other

  plane is a hyponym of vehicle

  sofa is a hyponym of furniture

- Conversely **hypernym**

  vehicle is a hypernym of car

  furniture is a hypernym of bed

- Hypernym/hyponym can also be called "IS-A" relation.

  sofa is a furniture

Meronym and Holonym

- Meronymy is a semantic relation between a **meronym** denoting a part and a **holonym** denoting a whole.

  “keycap” is a meronym of “keyboard”

  “keyboard” is a holonym of “keycap”

- Part-whole relation is also sometimes called “HAS-A” relation

  “keyboard” has a “keycap”

WordNet

- WordNet is a large lexical database of English:

  118k nouns, 12k verbs, 22k adjectives, 4.5k adverbs

  (wordnets for other languages: EuroWordNet, Global WordNet, …)

- Each word is associated with one or multiple senses with part-of-speech tags.

- Word senses are grouped into sets of synonyms (**synsets**), each expressing a distinct concept.

- Each synset is defined by a definition called **gloss**.

Synset

- The **synset** (synonym set) represents a distinct concept.

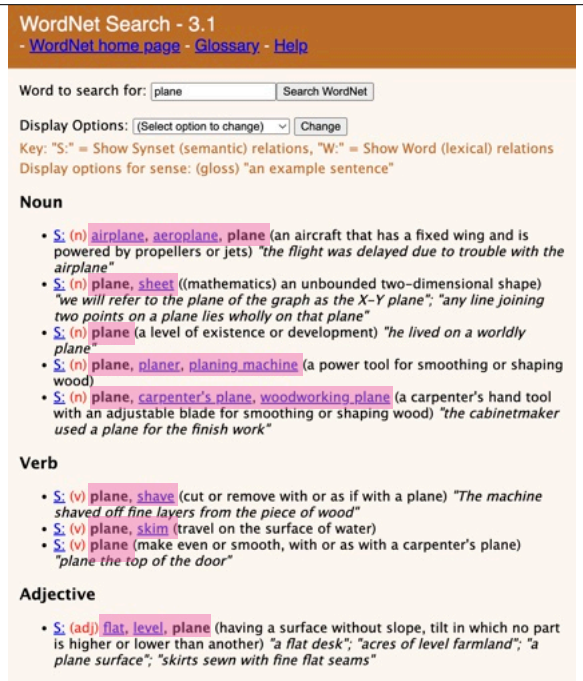
  plane.n.01: an aircraft that has a fixed wing and is powered by propellers or jets

  “n” means it is a noun

  “01” means it is the second sense of the word form “plane”

  It belongs to the **Synset(“airplane.n.01”)**, which includes

  plane.n.01, airplane.n.01 and aeroplane.n.01



Synset

- WordNet contains 81k noun synsets, 13k verb synsets, 19k adjective synsets, 3.5k adverb synsets.
- Synsets are connected in a graph with different types of “edges”:
  - **IS-A relation:** hypernyms and hyponyms.

  It is **transitive**: armchair is a kind of chair, chair if a kind of furniture, then we know armchair is a kind of furniture.
  - **HAS-A relation:** meronymy, part-whole relation
  - **Antonym relation**

- S: (n) **airplane**, aeroplane, plane (an aircraft that has a fixed wing and is powered by propellers or jets)
  - ***direct hyponym*** / *full hyponym*
  - *part meronym*
  - *domain term category*
  - *direct hypernym* / *inherited hypernym* / *sister term*
    - S: (n) heavier-than-air craft (a non-buoyant aircraft that requires a source of power to hold it aloft and to propel it)
      - *direct hyponym* / *full hyponym*
      - *direct hypernym* / *inherited hypernym* / *sister term*
        - S: (n) aircraft (a vehicle that can fly)
          - *direct hyponym* / *full hyponym*
          - *member holonym*
          - *part meronym*
          - *domain term category*
          - *direct hypernym* / *inherited hypernym* / *sister term*
            - S: (n) craft (a vehicle designed for navigation in or on water or air or through outer space)
              - *direct hyponym* / *full hyponym*
              - *direct hypernym* / *inherited hypernym* / *sister term*
                - S: (n) vehicle (a conveyance that transports people or objects)
                  - *direct hyponym* / *full hyponym*
                  - *part meronym*
                  - *direct hypernym* / *inherited hypernym* / *sister term*
                    - S: (n) conveyance, transport (something that serves as a means of transportation)
                      - *direct hyponym* / *full hyponym*
                      - *direct hypernym* / *inherited hypernym* / *sister term*
                        - S: (n) instrumentality, instrumentation (an artifact (or system of artifacts) that is instrumental in accomplishing some end)


```
bass³, basso (an adult male singer with the lowest voice)
=> singer, vocalist, vocalizer, vocaliser
   => musician, instrumentalist, player
      => performer, performing artist
         => entertainer
            => person, individual, someone...
               => organism, being
                  => living thing, animate thing,
                     => whole, unit
                        => object, physical object
                           => physical entity
                              => entity
bass⁷ (member with the lowest range of a family of instruments)
=> musical instrument, instrument
   => device
      => instrumentality, instrumentation
         => artifact, artefact
            => whole, unit
               => object, physical object
                  => physical entity
                     => entity
```


WordNet in NLTK

```
[1]: from nltk.corpus import wordnet as wn

[2]: wn.synsets('plane')

[2]: [Synset('airplane.n.01'),
      Synset('plane.n.02'),
      Synset('plane.n.03'),
      Synset('plane.n.04'),
      Synset('plane.n.05'),
      Synset('plane.v.01'),
      Synset('plane.v.02'),
      Synset('plane.v.03'),
      Synset('flat.s.01')]

[3]: wn.synsets('plane', pos=wn.VERB)

[3]: [Synset('plane.v.01'), Synset('plane.v.02'), Synset('plane.v.03')]

[4]: wn.synset('plane.n.01').hypernyms()

[4]: [Synset('heavier-than-air_craft.n.01')]

[5]: wn.synset('plane.n.01').hyponyms()

[5]: [Synset('airliner.n.01'),
      Synset('amphibian.n.02'),
      Synset('biplane.n.01'),
      Synset('bomber.n.01'),
      Synset('delta_wing.n.01'),
      Synset('fighter.n.02'),
      Synset('hangar_queen.n.01'),
      Synset('jet.n.01'),
      Synset('monoplane.n.01'),
      Synset('multiengine_airplane.n.01'),
      Synset('propeller_plane.n.01'),
      Synset('reconnaissance_plane.n.01'),
      Synset('seaplane.n.01'),
      Synset('ski-plane.n.01'),
      Synset('tanker_plane.n.01')]
```

Word Similarity

- **Synonymy**: a binary relation

  Two words are either synonymous or not

- **Similarity**: a looser metric

  Two words are more similar if they share more features of meaning

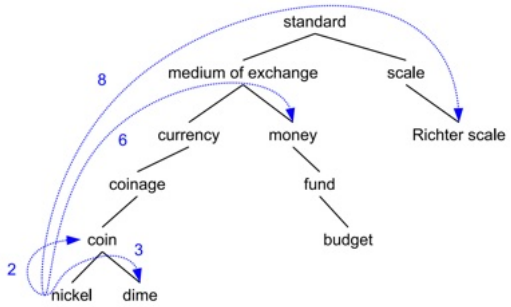
  sim(sofa, table) >? sim(sofa, car)

  sim(sofa, table) >? sim(sofa, soda)

- sofa and soda has a smaller **edit distance** but very different meanings!

Path-based Similarity

- Path length is just the distance between synsets.



Path-based Similarity

- Given a semantic hierarchy such as WordNet, semantic similarity can be estimated in various ways based on the path between two words. Let's define:
- pathlen(s_1, s_2) = 1 + number of edges in the shortest path in the hypernym graph between sense nodes s_1 and s_2
- simpath(s_1, s_2) = $\frac{1}{\text{pathlen}(s_1, s_2)}$
- wordsim(w_1, w_2) = $\max_{s_1 \in senses(w_1),\ s_2 \in senses(w_2)} \text{simpath}(s_1, s_2)$

Path-based Similarity

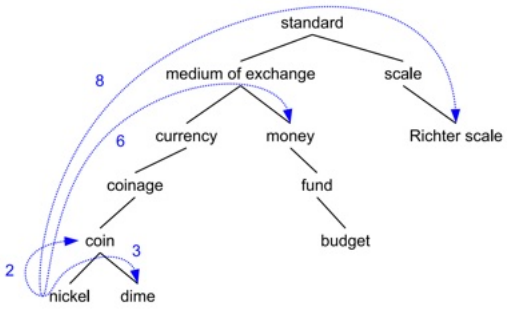
simpath(nickel, coin)=1/2

simpath(fund, budget)=1/2

simpath(nickel, currency)=1/4

simpath(nickel, money)=1/6

simpath(nickel, standard)=1/6



But do we really want simpath(nickel, money) = simpath(nickel, standard) ?

Dictionary-based Similarity

- Dictionaries typically contain a short definition (**gloss**) for each word.
- The term overlap between the glosses of words $w_1$and w_2 can be used as a similarity score.
- If the semantic resource also contains hyponym/hypernym relations, the overlap with them can be used as well.
- For example, if hypo(x) is concatenation of hyponym glosses,

  simdict(w_1, w_2) =

  overlap(gloss(w_1), gloss(w_2)) +

  overlap(gloss(hypo(w_1)), gloss(hypo(w_2))) +

  overlap(gloss(w_1), gloss(hypo(w_2))) +

  overlap(gloss(hypo(w_1)), gloss(w_2))

Problems with Path/Dictionary-based Similarity

- We need a high quality lexicon, which takes time and costs money!
- What if the word you are looking for is not in the database?
- We need a consistent hierarchy of hypernyms and hyponyms

  — Can we ever guarantee each link represents a uniform distance?
- Distributional similarity? We will discuss it in the next lecture.

Word Sense Ambiguity

- We have seen that many words have multiple possible meanings.
- When meeting them in text, how can a model know which sense to pick?

  I went to the **bank**. The water was inviting.

  I went to the **bank** to get some cash.

  We spotted the annoying **bug**. Then it flew away.

  We spotted the annoying **bug** and fixed it.

Humor

Humor

WordNet has 42 senses of the verb "take"!

- S: (v) **take** (carry out)
- S: (v) **take**, occupy, use up (require (time or space))
- S: (v) lead, **take**, direct, conduct, guide (take somebody somewhere)
- S: (v) **take**, get hold of (get into one's hands, take physically)
- S: (v) assume, acquire, adopt, take on, **take** (take on a certain form, attribute, or aspect)
- S: (v) **take**, read (interpret something in a certain way; convey a particular meaning or impression)
- S: (v) bring, convey, **take** (take something or somebody with oneself somewhere)
- S: (v) **take** (take into one's possession)
- S: (v) **take** (travel or go by means of a certain kind of transportation, or a certain route)
- S: (v) choose, **take**, select, pick out (pick out, select, or choose from a number of alternatives)
- S: (v) accept, **take**, have (receive willingly something given or offered)
- S: (v) fill, **take**, occupy (assume, as of positions or roles)
- S: (v) consider, **take**, deal, look at (take into consideration for exemplifying purposes)
- S: (v) necessitate, ask, postulate, need, require, **take**, involve, call for, demand (require as useful, just, or proper)
- S: (v) **take** (experience or feel or submit to)
- S: (v) film, shoot, **take** (make a film or photograph of something)
- S: (v) remove, **take**, take away, withdraw (remove something concrete, as by lifting, pushing, or taking off, or remove something abstract)
- S: (v) consume, ingest, take in, **take**, have (serve oneself to, or consume regularly)
- S: (v) **take**, submit (accept or undergo, often unwillingly)
- S: (v) **take**, accept (make use of or accept for some purpose)
- S: (v) **take** (take by force)
- S: (v) assume, **take**, strike, take up (occupy or take on)
- S: (v) accept, admit, **take**, take on (admit into a group or community)
- S: (v) **take** (ascertain or determine by measuring, computing or take a reading from a dial)
- S: (v) learn, study, read, **take** (be a student of a certain subject)
- S: (v) claim, **take**, exact (take as an undesirable consequence of some event or state of affairs)
- S: (v) **take**, make (head into a specified direction)
- S: (v) aim, **take**, train, take aim, direct (point or cause to go (blows, weapons, or objects such as photographic equipment) towards)
- S: (v) **take** (be seized or affected in a specified way)
- S: (v) carry, pack, **take** (have with oneself; have on one's person)
- S: (v) lease, rent, hire, charter, engage, **take** (engage for service under a term of contract)
- S: (v) subscribe, subscribe to, **take** (receive or obtain regularly)
- S: (v) **take** (buy, select)
- S: (v) **take** (to get into a position of having, e.g., safety, comfort)
- S: (v) **take**, have (have sex with; archaic use)
- S: (v) claim, **take** (lay claim to; as of an idea)
- S: (v) accept, **take** (be designed to hold or take)
- S: (v) contain, **take**, hold (be capable of holding or containing)
- S: (v) **take** (develop a habit)
- S: (v) drive, **take** (proceed along in a vehicle)
- S: (v) **take** (obtain by winning)
- S: (v) contract, **take**, get (be stricken by an illness, fall victim to an illness)

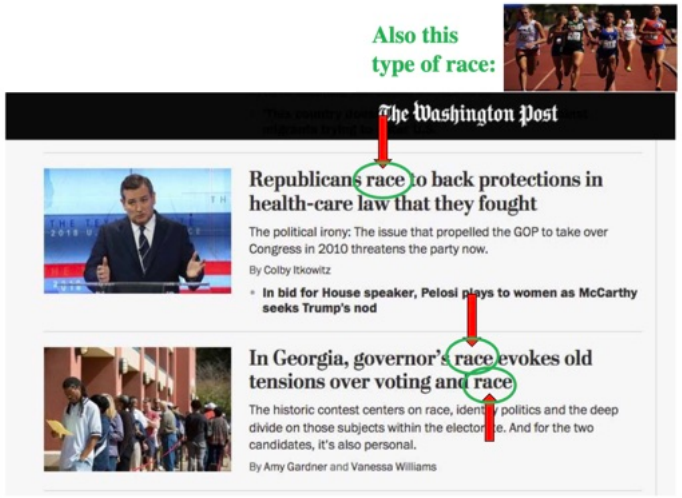
And verb + particle Expressions

take after	She takes after her mother.
take down	The rocket took down a plane.
take back	I take back what I said.
take on	He took on the tobacco lobby.
take it out	She took it out on her boyfriend.
take off	He took off for Bermuda.
take over	He plans to take over the company.
take to	The boy took to his new babysitter.
take up	The girl took up karate.
take up with	He took up with his neighbor.

Word Sense Disambiguation

- Word Sense Disambiguation (WSD) is the task of determining the correct sense for a polysemous word in a specific context.
- WSD systems typically assume the presence of a sense inventory (set of all possible senses) for each polysemous word using a lexical resource such as WordNet.
- Using the most common sense is a reasonable baseline approach. The first sense listed in WordNet is usually considered to be the most common sense.

Polysemy in the Wild



Dictionary-based WSD

- The Lesk algorithm is the oldest and most powerful knowledge-based WSD method.
- It uses a dictionary, such as WordNet.
- It disambiguates word by measuring the overlap between the words in its **context** and the words in each sense **definition**. The sense with the highest overlap wins.

The Simplified Lesk Algorithm

function SIMPLIFIED LESK(*word*, *sentence*) **returns** best sense of *word*

best-sense ← most frequent sense for *word*
max-overlap ← 0
context ← set of words in *sentence*
for each *sense* **in** senses of *word* **do**
  signature ← set of words in the gloss and examples of *sense*
  overlap ← COMPUTEOVERLAP(*signature*, *context*)
  if *overlap* > *max-overlap* **then**
    max-overlap ← *overlap*
    best-sense ← *sense*
end
return(*best-sense*)

The Simplified Lesk Algorithm

Context: The **bank** can guarantee deposits will eventually cover future tuition costs because it invests in adjustable-rate mortgage securities.

bank^1	Gloss:	a financial institution that accepts deposits and channels the money into lending activities
	Examples:	"he cashed a check at the bank", "that bank holds the mortgage on my home"
bank^2	Gloss:	sloping land (especially the slope beside a body of water)
	Examples:	"they pulled the canoe up on the bank", "he sat on the bank of the river and watched the currents"

bank^1 has two non-stopwords overlapping with the context: **deposits** and **mortgage**.

bank^2 has zero words.

Then bank^1 is selected.

Supervised Machine Learning for WSD

- Tag/Label: WordNet sense
- An annotated corpus. For example,

  SemCor: 234,000 words from Brown Corpus, manually tagged with WordNet senses.
- What type of classifier?

  naive bayes, logistic regression, neural models, …

Naive Bayes for WSD

- P(s) is the prior probability of that sense

  Counting in a labeled training set.
- P(w|s) conditional probability of a word given a particular sense

  P(w|s) = count(w,s)/count(s)
- We get both of these from a tagged corpus like SemCor.

A Real Example

- Suppose "bass" has two senses:

  bass-1: the lean flesh of a saltwater fish of the family Serranidae

  bass-2: the member with the lowest range of a family of musical instruments
- And we have this small corpus:

	Doc	Words (context of "bass")	Class
Training	1	fish smoked fish	bass-1
	2	fish line	bass-1
	3	fish haul smoked	bass-1
	4	guitar jazz line	bass-2
Test	5	line guitar jazz jazz	?

	Doc	Words (context of "bass")	Class
Training	1	fish smoked fish	bass-1
	2	fish line	bass-1
	3	fish haul smoked	bass-1
	4	guitar jazz line	bass-2
Test	5	line guitar jazz jazz	?

- Prior: P(class) = count(class) / sum(count(class'))

  P(bass-1) = P(bass-2)=
- Likelyhood: P(word | class) = (count(word, class)+1) / (sum(count(word', class))+|V|)

  P(line | bass-1) = P(line | bass-2) =

  P(guitar | bass-1) = P(guitar | bass-2) =

  P(jazz | bass-1) = P(jazz | bass-2) =
- Using naive bayes: argmax P(class | test) = argmax P(class) $\prod$ P(word | class)

  P(bass-1 | test) =

  P(bass-2 | test) =

	Doc	Words (context of "bass")	Class
Training	1	fish smoked fish	bass-1
	2	fish line	bass-1
	3	fish haul smoked	bass-1
	4	guitar jazz line	bass-2
Test	5	line guitar jazz jazz	?

# of words in bass-1 text: 8
# of words in bass-2 text: 3
$|V|$: 6

Vocab = {fish, smoked, line, haul, guitar, jazz}

- Prior: P(class) = count(class) / sum(count(class'))

  P(bass-1) = 3/4 P(bass-2)= 1/4

- Likelyhood: P(word | class) = (count(word, class)+1) / (sum(count(word', class))+|V|)

  P(line | bass-1) = (1+1)/(8+6) = 2/14 P(line | bass-2) = (1+1)/(3+6) = 2/9

  P(guitar | bass-1) = (0+1)/(8+6) = 1/14 P(guitar | bass-2) = (1+1)/(3+6) = 2/9

  P(jazz | bass-1) = (0+1)/(8+6) = 1/14 P(jazz | bass-2) = (1+1)/(3+6) = 2/9

- Using naive bayes: argmax P(class | test) = argmax P(class) $\prod$ P(word | class)

  P(bass-1 | test) = 3/4 * 2/14 * 1/14 * 1/14 * 1/14 = 3/(2*14^4)

  P(bass-2 | test) = 1/4 * 2/9 * 2/9 * 2/9 * 2/9 = 4/(9^4) → win!

Weak Supervision for WSD

- Yarowsky [1995] developed a weakly supervised bootstrapping method for word sense disambiguation, which exploits coarse distributional similarity.

- Two key ideas:

  1. One sense per **collocation**: Nearby (co-located) words provide strong and consistent clues to the sense of a word. (Different senses occur in recognizably different contexts.)

  2. One sense per **discourse**: The sense of a target word is highly consistent within a document.

Bootstrapping Algorithm

1. Collect the words in a context window around each instance of the target word.
2. For each sense, use **seed words** to label (some) instances as positive training examples for the sense. The remaining unlabeled instances are called the residual.
3. Train a supervised classifier with the labeled instances.
4. Apply the classifier to all instances in the residual. Apply a threshold to identify the most confidently labeled instances and add them to the training set.
5. [Optional:] Apply the one-sense-per-discourse heuristic to modify or add more labels.
6. If new instances were labeled, then return to Step 3.

Initial State

Sense	Training Examples		
?	...company said the	**plant**	is still operating
?	Although thousands of	**plant**	and animal species
?	...zonal distribution of	**plant**	life...
?	...to strain microscopic	**plant**	life from the...
?	vinyl chloride monomer	**plant**	, which is...
?	and Golgi apparatus of	**plant**	and animal cells
?	...computer disk drive	**plant**	located in...
?	...divide life into	**plant**	and animal kingdom
?	...close-up studies of	**plant**	life and natural
?	...Nissan car and truck	**plant**	in Japan is...
?	...keep a manufacturing	**plant**	profitable without

plant-A: a living organism lacking the power of locomotion
plant-B: buildings for carrying on industrial labor

Intermediate State

Sense	Training Examples		
A	...zonal distribution of	**plant**	life...
A	...close-up studies of	**plant**	life and natural...
A	...divide life into	**plant**	and animal kingdom
?	...vinyl chloride monomer	**plant**	, which is...
?	...Nissan car and truck	**plant**	in Japan is...
?	...and Golgi apparatus of	**plant**	and animal cells
?	...computer disk drive	**plant**	located in...
B	...keep a manufacturing	**plant**	profitable without...
B	...discovered at a St. Louis	**plant**	manufacturing...
B	...vast manufacturing	**plant**	in Fremont
B	...copper manufacturing	**plant**	found that they...

plant-A: a living organism lacking the power of locomotion
seeds: {life, living}
plant-B: buildings for carrying on industrial labor
seeds: {industrial, buildings, manufacturing}

One-sense-per-discourse Heuristic

Labeling previously untagged contexts:

Tag	Text	Training Examples		(from same discourse)
A → A	724	...the existence of	**plant**	and animal life...
A → A	724	...classified as either	**plant**	or animal...
? → A	724	Although bacterial and	**plant**	cells are enclosed

Error Correction:

Tag	Text	Training Examples		(from same discourse)
A → A	525	...contains a varied	**plant**	and animal life...
A → A	525	...the most common	**plant**	life, the...
A → A	525	...slight within Arctic	**plant**	species...
B → A	525	...are protected by	**plant**	parts remaining from...

How to get seeds?

Yarowsky experimented with 3 methods for choosing seeds:

1. Choose words that appear in the dictionary definition for the target word, and more frequently than in other definitions.

2. Manually define a single collocate for each sense.

   For example: "fish" and "play" for the 2 senses of bass.

3. Identify words that frequently co-occur with target word in a large text collection. A human must decide which sense each collocate is associated with.

   For example, "salmon" and "saxophone" might show up for the 2 senses of bass.

Pros and Cons

- Requires no annotated training data, just raw text!
- The one-sense-per-discourse heuristic can be applied after each iteration or at the end, and can be exploited for other discourse tasks as well.
- The one-sense-per-discourse heuristic allows the algorithm to escape from (some) misclassifications.
- Multiple senses associated with the same general domain can be difficult to distinguish (e.g., "bass" as a musical instrument vs. a singing voice).
- One limitation of this approach is coverage. Some contexts may not contain corroborating evidence for a sense.

CS 5134/6034 Natural Language Processing
Fall 2024

Lecture 14: Distributional Representations

University of CINCINNATI

How to get meanings of words?

- Dictionary

  The **lexicographic** approach understands words from dictionaries, lexicons, ontologies, etc.

- Large Raw Text

  The **distributional** approach understands words based on large text corpus.

Distributional Hypothesis

- "You shall know a word by the company it keeps!"

  (FIRTH 1957)

- "oculist and eye-doctor ... occur in almost the same environments ... If A and B have almost identical environments we say that they are synonyms."

  (HARRIS 1954)

- **Distributional hypothesis:**

  Words that occur in similar contexts tend to have similar meanings.

Intuition

What is "shuke"?

The shuke weighs 10,000 kg.

The shuke strengthens national security.

DOD plans to procure nearly 2,500 shukes.

The shukes can internally store up to six missiles.

The pilot was able to safely eject from the shuke.

The shuke can reach a top speed of Mach 1.6.

Though we don't know what exactly "shuke" is, we can tell from the sentences that it is likely a military plane.

Distributional Similarity

- From the distributional hypothesis, two words will have a high distributional similarity if their surrounding contexts are similar.
- Represent words as vectors such that
  - — each **element** corresponds to a different **context**
  - — the vector captures how strongly the word is associated with each context
- The similarity between words is computed as the similarity between their vectors.

Distributional Similarity

- Represent each word as a vector

$$w = (w_1, \ldots, w_n) \in \mathbb{R}^n$$

  in an n-dimensional space.
  - — each dimension corresponds to a particular **context** c_i
  - — w_i means how strongly the word is associated with c_i
- The similarity between words u and v is computed as the similarity of their vectors u and v.

Manhattan Distance

$$\text{ManhattanDistance}(v, u) = \sum_{i=1}^{n} |v_i - u_i|$$

Example:

$u = (1,2,3,4)$, $v = (2,3,4,5)$

$$\text{ManhattanDistance}(v, u) = \sum_{i=1}^{n} |v_i - u_i| = |1-2| + |2-3| + |3-4| + |4-5| = 4$$

- It is very sensitive to outliers, e.g., v=(2,3,4,1000).
- The distance “grows” as the vector gets longer.

Euclidean Distance

$$\text{EuclideanDistance}(v, u) = \sqrt{\sum_{i=1}^{n} (v_i - u_i)^2}$$

Example:

$u = (1,2,3,4)$, $v = (2,3,4,5)$

$$\text{EuclideanDistance}(v, u) = \sqrt{\sum_{i=1}^{n} (v_i - u_i)^2} = \sqrt{1^2 + 1^2 + 1^2 + 1^2} = 2$$

- It is also very sensitive to outliers.

Jaccard Similarity

- The Jaccard similarity metric assesses the amount of weighted overlap between features.

$$\text{Jaccard}(v,u) = \frac{\sum_{i=1}^{n} \min(v_i, u_i)}{\sum_{i=1}^{n} \max(v_i, u_i)}$$

Example:

$u = (1,2,3,4)$, $v = (2,3,4,5)$

$$\text{Jaccard}(v,u) = \frac{\sum_{i=1}^{n} \min(v_i, u_i)}{\sum_{i=1}^{n} \max(v_i, u_i)} = \frac{1+2+3+4}{2+3+4+5} = \frac{10}{14}$$

- Vector length does not matter now.
- But it is still very sensitive to outliers.

Cosine Similarity

- A consistently good similarity metric is the cosine between the vectors.
- The numerator is the dot product (from linear algebra), and the denominator normalizes by the length of each vector.

$$\text{Cosine}(v,u) = \frac{\sum_{i=1}^{n} v_i * u_i}{\sqrt{\sum_{i=1}^{n} v_i^2} * \sqrt{\sum_{i=1}^{n} u_i^2}}$$

Example:

$u = (1,2,3,4)$, $v = (2,3,4,5)$

$$\text{Cosine}(v,u) = \frac{\sum_{i=1}^{n} v_i * u_i}{\sqrt{\sum_{i=1}^{n} v_i^2} * \sqrt{\sum_{i=1}^{n} u_i^2}} = \frac{1*2+2*3+3*4+4*5}{\sqrt{1^2+2^2+3^2+4^2} * \sqrt{2^2+3^2+4^2+5^2}} = \frac{40}{\sqrt{30} * \sqrt{54}}$$

Cosine Similarity

$$\text{Cosine}(v,u) = \frac{\sum_{i=1}^{n} v_i * u_i}{\sqrt{\sum_{i=1}^{n} v_i^2} * \sqrt{\sum_{i=1}^{n} u_i^2}}$$

- Cosine similarity is very robust and widely used across NLP.
- Cosine(v, u)=1: v and u point in the same direction

  Cosine(v, u)=0: v and u are orthogonal

  Cosine(v, u)=-1: v and u point in the opposite direction

Define Meaning in a Vector Space

- Each word is represented by a high-dimensional vector.
- Similar words are “nearby in semantic space”.

How to Define Vectors: Context Features

- We assume the vector captures how strongly the word is associated with **each context**.
- What is a context?
  - — **the document containing the word**
  - — k words before and after the word
  - — grammatically related words

Term-Document Matrix

	Doc1	Doc2	Doc3	Doc4
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

A term-document matrix is a 2D table:

— Each **cell** is the frequency count of the term (word) t in document d, denoted as TF(t, d) , also called term frequency

— Each **column** is a vector of counts over words, representing a **document**

— Each **row** is a vector of counts over documents, representing a **word**

Term-Document Matrix

	Doc1	Doc2	Doc3	Doc4
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

— Each **column** is a vector of counts over words, representing a **document**

— Each **row** is a vector of counts over documents, representing a **word**

Two **words**/**documents** are similar if their vector are similar.

TF-IDF

	Doc1	Doc2	Doc3	Doc4	…
a	10	10	8	15	
soldier	10	0	0	0	

"a" appears in 30 documents
"soldier" appears in 1 document

Term Frequency (TF):

TF(t, d) = # times that term t occurs in document d

Document Frequency (DF):

DF(t) = # documents that contain term t

Inverse Document Frequency (IDF):

$$\text{IDF}(t) = \log \frac{N}{\text{DF}(t)}$$

where N is the number of documents in the text collection.

Then we define:

TF-IDF(t, d) = TF(t, d) * IDF(t)

TF-IDF

- TF-IDF is a widely used term-weighting scheme that originated in the information retrieval (IR) community.
  - finding the most relevant document for a query.
- What is the TF-IDF value for terms that occur in all documents?
  - log N/DF(t) = log 1 = 0.
- TF-IDF gives higher weights to terms that are less common, under the assumption that these words are particularly relevant when they do occur.

How to Define Vectors: Context Features

- We assume the vector captures how strongly the word is associated with **each context**.
- What is a context?
  - the document containing the word
  - **k words before and after the word**
  - grammatically related words

Word-Word Matrix

- Context defined by nearby words:

sugar, a sliced lemon, a tablespoonful of	**lemon**	preserve or jam, a pinch each of
their enjoyment. Cautiously she sampled her first	**pineapple**	and another fruit whose taste she likened
well suited to programming on the digital	**computer**	In finding the optimal R-stage policy from
for the purpose of gathering data and	**information**	necessary for the study authorized in the

...

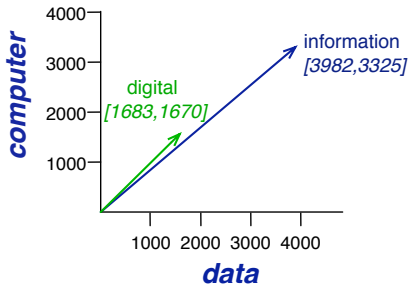
Context: ± 7 words

- Each cell is the frequency count of word w **co-occur** with context word c

context

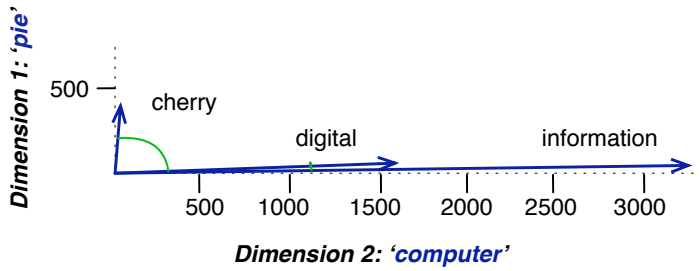
word	sugar	lemon	data	…
lemon	1	1	0	
pineapple	0	10	0	
computer	0	0	18	
information	0	0	20	

Similarity based on Word-Word Matrix



A 2D visualization of word vectors for “digital” and “information”, corresponding to only 2 context words “data” and “computer”.

Visualizing Cosine Similarity



Co-occurrence

- We can use different methods to define co-occurrence:
  - within a fixed window: c occurs within $\pm k$ words of w
  - within the same sentence
  - by grammatical relations

    For example, c occurs as a subject/object/… of verb w
- Representing co-occurrences:
  - binary features (1, 0): w does (not) co-occur with c
  - frequency count: # times w co-occur with c

Raw Frequency Count Issues

- Sometimes, low co-occurrences counts are very informative, and high co-occurrence counts are not.
- Very common words like “a”, “the” will always have high frequency.
- We want to identify when co-occurrence counts are higher than we would **expect by chance**, e.g., Riemann and zeta.
- So, we use a weighted function instead of raw counts!

Pointwise Mutual Information (PMI)

- **Pointwise Mutual Information (PMI)** measures association by comparing the observed joint distribution of two terms with the probability that they should occur together if they are independent:

$$\text{PMI}(w, c) = \log \frac{P(w, c)}{P(w)P(c)}$$

- On your own: work out the ways to define $P(w, c)$, $P(w)$ and $P(c)$.

Positive Pointwise Mutual Information (PPMI)

- PMI can be negative (co-occurring less often than expected by chance).
- People often use Positive Pointwise Mutual Information (PPMI) instead:

$$\text{PMI}(w, c) = \max(0, \log \frac{P(w, c)}{P(w)P(c)})$$

PMI Example

P(the) = 0.50 **P(the & dog) = 0.02**
P(dog) = 0.03 **P(barked & dog) = 0.02**
P(barked) = 0.02

$$\text{PMI(the, dog)} = \log \frac{P(\text{the \& dog})}{P(\text{the})P(\text{dog})} = \log_2 \frac{0.02}{0.50 * 0.03} = 0.42$$

$$\text{PMI(barked, dog)} = \log \frac{P(\text{barked \& dog})}{P(\text{barked})P(\text{dog})} = \log_2 \frac{0.02}{0.02 * 0.03} = 5.06$$

- PMI is biased towards infrequent events.

Sparse vs. Dense Vectors

- TF-IDF or PMI vectors are long and sparse.
  - long: dimension equals # documents (TF-IDF) or vocabulary size (PMI)
  - sparse: most elements are zero
- Alternative: learn vectors that are
  - short: length 50-1000
  - dense: most elements are non-zero

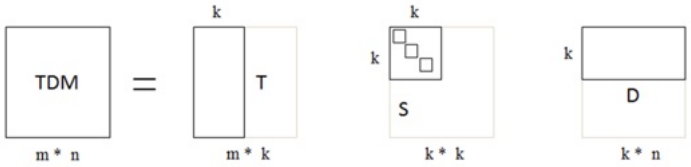
Sparse vs. Dense Vectors

Why do we want dense vectors?

- Dense vectors may be easier to use as features in machine learning (fewer weights to learn)
- Dense vectors may do better at capturing synonymy:
  - In PMI vectors, synonym words such as car and automobile, are distinct dimensions. However, a word with car as a neighbor and a word with automobile as a neighbor should be similar!

How to get short dense vectors?

- Count-based methods: singular value decomposition (SVD) of term-document matrix (TDM)



How to get short dense vectors?

- Prediction-based methods:

  — Train a binary classifier to predict if one word is likely to appear in the context of another.

  — The parameters of the classifier provide a dense vector for the target word.

  — The vectors are also called **word embeddings**.

  — It can be trained on large amounts of raw text in a **self-supervised** manner.

CS 5134/6034 Natural Language Processing
Fall 2024

Lecture 15: Word Embeddings

University of CINCINNATI

Sparse vs. Dense Vectors

- TF-IDF or PMI vectors are long and sparse.
  - long: dimension equals # documents (TF-IDF) or vocabulary size (PMI)
  - sparse: most elements are zero
- Alternative: learn vectors that are
  - short: length 50-1000
  - dense: most elements are non-zero

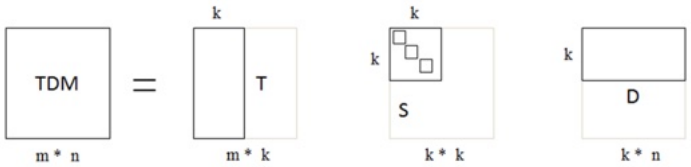
Sparse vs. Dense Vectors

Why do we want dense vectors?

- Dense vectors may be easier to use as features in machine learning (fewer weights to learn)
- Dense vectors may do better at capturing synonymy:
  - In PMI vectors, synonym words such as car and automobile, are distinct dimensions. However, a word with car as a neighbor and a word with automobile as a neighbor should be similar!

How to get short dense vectors?

- Count-based methods: singular value decomposition (SVD) of term-document matrix (TDM)



How to get short dense vectors?

- Prediction-based methods:

  — Train a binary classifier to predict if one word is likely to appear in the context of another.

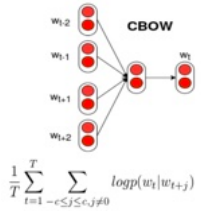
  — The parameters of the classifier provide a dense vector for the target word.

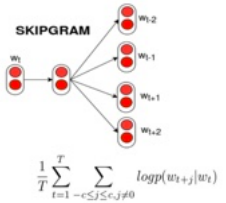
  — The vectors are also called **word embeddings**.

  — It can be trained on large amounts of raw text in a **self-supervised** manner.

Word2Vec

- Mikolov et al., 2013a: Efficient Estimation of Word Representations in Vector Space
- Mikolov et al., 2013b: Distributed Representations of Words and Phrases and their Compositionality
- Two ways to represent context:





Skip-gram

- Instead of counting how often each word occurs near “apricot"
  — Train a classifier on a binary prediction task:

  Is the given word likely to show up near “apricot"?

“…lemon, a [tablespoon of **apricot** jam, a] pinch…”

c1 c2 c3 c4

- Our goal is learn weights to maximize:

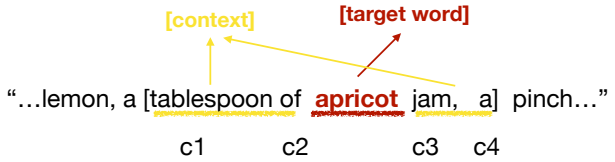
P(“tablespoon” near “apricot”) * P(“of” near “apricot”) * P(“jam” near “apricot”) * P(“a” near “apricot”) * … * P(“spaceship” is not near “apricot”) * P(“NLP” is not near “apricot”) * …

Skip-gram

- Treat the target word w and a neighboring context word c as **positive** examples.
- Randomly sample other words in the lexicon to get **negative** examples.
- Use **logistic regression** to train a classifier to distinguish those two cases.
- Use the learned weights as the **embeddings**.

Skip-gram Training Data

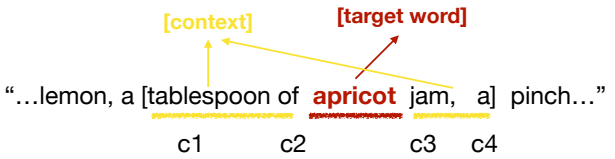
- Assume a ± 2 word window, given training sentence:



- Decide whether c is a real context word for the target word w:

$$P(+|w,c) > P(-|w,c) = 1 - P(+|w,c)$$

Skip-gram Training Data



- **Positive** examples:

  (apricot, tablespoon), (apricot, of), (apricot, jam), …

- **Negative** examples?

Sampling Negative Examples

- How should we get negative "contexts"?
- For each positive example (w, c), we randomly sample k words c' and add them as negative training examples (w, c').
- Words in the vocabulary can be sampled based on the their frequency.

positive examples +	
t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -			
t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

How to get $P(+|w,c)$

- **Intuition:**

  Words are likely to appear near similar words.

- **Idea:**

  Model similarity with a dot-product of vectors:

  Similarity(w, c) = $w \cdot c$

- **Normalization:**

  Use sigmoid function to get a probability:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Skip-gram Classifier

- Our classifier should be able to tell whether c is a real context word for the target word w:

$$P(+|w,c) \stackrel{?}{>} P(-|w,c)$$

- Recall logistic regression for binary classification:

$$P(y=1|x) = \sigma(\theta_0 + \sum_{i=1}^{n} \theta_i \cdot x_i)$$

Parameters to learn: θ

Skip-gram Classifier

- Logistic regression for binary classification:

$$P(y=1|x) = \sigma(\theta_0 + \sum_{i=1}^{n} \theta_i \cdot x_i)$$

- Here we have two weight vectors, instead of one feature vector and one weight vector.

$$P(+|w,c) = \sigma(\sum_{i=1}^{n} w_i \cdot c_i)$$

- That is,

$$P(+|w,c) = \frac{1}{1+e^{-w\cdot c}} \qquad P(-|w,c) = 1 - P(+|w,c)$$

Skip-gram: Training Objective

- We want to find a model that maximize the similarity of the target word with the actual context words, and minimize the similarity of the target with the k negative sampled non-neighbor words.

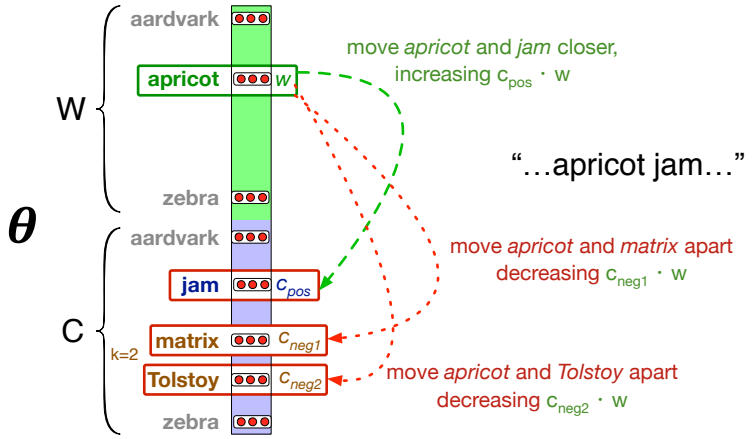
$$\begin{aligned} L_{CE} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\ &= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\ &= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\ &= -\left[\log \sigma(c_{pos}\cdot w) + \sum_{i=1}^{k}\log\sigma(-c_{neg_i}\cdot w)\right] \end{aligned}$$

$\sigma(x)+\sigma(-x)=1$

Skip-gram: Learning the Classifier

- Use stochastic gradient descent to learn the parameters W (matrix of target word vectors w) and C (matrix of context word vectors c).
- Here, we don't actually care about the binary classification like we do for the logistic regression model in text classification task.
- We care about the learned parameters W and C — they give us good vector representations of words!

Skip-gram: Learning the Classifier



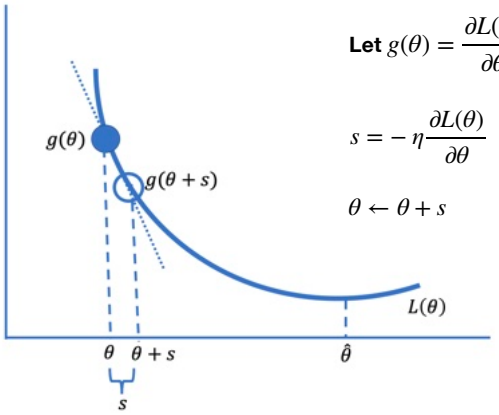
The intuition of one step of gradient descent.

Gradient Descent

$$\theta \leftarrow \theta - \eta \frac{\partial L(\theta)}{\partial \theta}$$

- **Direction:** We move in the reverse direction from the gradient of the loss function.
- **Magnitude:** We move the value of this gradient $\frac{\partial L(\theta)}{\partial \theta}$ weighted by a learning rate η.
- Higher learning rate means we change θ faster.

Gradient Descent



Let $g(\theta) = \frac{\partial L(\theta)}{\partial \theta}$

$$s = -\eta \frac{\partial L(\theta)}{\partial \theta}$$

$$\theta \leftarrow \theta + s$$

Derivatives of the Loss Function L_{CE}

$$L_{CE} = -\left[\log \sigma\left(c_{pos} \cdot w\right) + \sum_{i=1}^{k} \log \sigma\left(-c_{neg_i} \cdot w\right)\right]$$

, where $\sigma(x) = \frac{1}{1+e^{-x}}$.

$$\frac{\partial L_{CE}}{\partial c_{pos}} = ?$$

Derivatives of the Loss Function L_{CE}

$$L_{CE} = -\left[\log \sigma\left(c_{pos} \cdot w\right) + \sum_{i=1}^{k} \log \sigma\left(-c_{neg_i} \cdot w\right)\right]$$

$$\frac{\partial L_{CE}}{\partial c_{pos}} = \left[\sigma\left(c_{pos} \cdot w\right) - 1\right] w$$

$$\frac{\partial L_{CE}}{\partial c_{neg_i}} = \left[\sigma\left(c_{neg_i} \cdot w\right)\right] w$$

$$\frac{\partial L_{CE}}{\partial w} = \left[\sigma\left(c_{pos} \cdot w\right) - 1\right] c_{pos} + \sum_{i=1}^{k} \left[\sigma\left(c_{neg_i} \cdot w\right)\right] c_{neg_i}$$

Update Parameters

At time step t:

$$c_{pos}^{t+1} = c_{pos}^{t} - \eta \left[\sigma\left(c_{pos}^{t} \cdot w^{t}\right) - 1\right] w^{t}$$

$$c_{neg}^{t+1} = c_{neg}^{t} - \eta \left[\sigma\left(c_{neg}^{t} \cdot w^{t}\right)\right] w^{t}$$

$$w^{t+1} = w^{t} - \eta \left[\left[\sigma\left(c_{pos} \cdot w^{t}\right) - 1\right] c_{pos} + \sum_{i=1}^{k} \left[\sigma\left(c_{neg} \cdot w^{t}\right)\right] c_{neg_i}\right]$$

Two Sets of Embeddings

- Skip-gram learns two sets of embeddings

  Target embeddings matrix W

  Context embedding matrix C

- It's common to just add them together, representing the i-th word as the vector $W_i + C_i$

Summary of Skip-gram Embeddings

- For a vocabulary of size V: Start with V random vectors (typically 300-dimensional) as initial embeddings.
- Train a logistic regression classifier to distinguish words that co-occur in corpus from those that don't
  - —Pairs of words that co-occur are positive examples
  - —Pairs of words that don't co-occur are negative examples
- During training, target and context vectors of positive examples will become similar, and those of negative examples will become dissimilar.
- Keep two embedding matrices W and C, where each word in the vocabulary is mapped to a 300D vector.

Word Embeddings Evaluation

- Extrinsic Evaluation

  Use the word embeddings for downstream tasks.

  Recall logistic regression for text classification, we need to define feature vectors for the input. Now we can use word embeddings as feature vectors!

  Or use word embeddings for clustering.

- Intrinsic Evaluation

Intrinsic Evaluation: Word Similarity

- Compare to human scores on word similarity.

  WordSim-353 (Finkelstein et al., 2002)

  SimLex-999 (Hill et al., 2015)

  Stanford Contextual Word Similarity (SCWS) dataset (Huang et al., 2012)

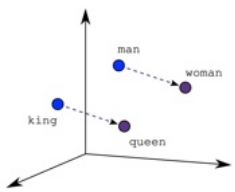
```
[1]: import torch
     import torchtext

[2]: glove = torchtext.vocab.GloVe(name="6B", dim=300)

[3]: torch.cosine_similarity(glove['cat'].unsqueeze(0),
                             glove['dog'].unsqueeze(0))

[3]: tensor([0.6817])
```


Intrinsic Evaluation: Word Analogy

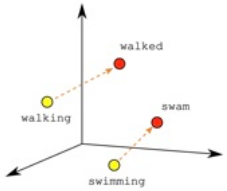
- To solve “man is to woman as king is to ___”

$$v_{man} - v_{women} \approx v_{king} - v_{queen}$$

$$v_{Paris} - v_{France} \approx v_{Rome} - v_{Italy}$$



Male-Female



Verb tense

Intrinsic Evaluation: Word Analogy

- To solve “man is to woman as king is to ___”

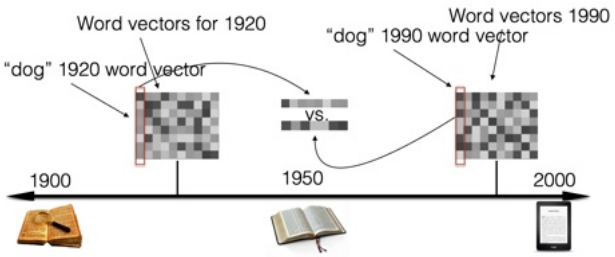
$$v_{man} - v_{women} \approx v_{king} - v_{queen}$$

$$v_{Paris} - v_{France} \approx v_{Rome} - v_{Italy}$$

```
[4]: def closest_words(vec, n=5):
         dists = torch.norm(glove.vectors - vec, dim=1)
         lst = sorted(enumerate(dists.numpy()), key=lambda x: x[1])
         for idx, difference in lst[1:n+1]:
             print(glove.itos[idx], difference)

[5]: closest_words(glove['king'] - glove['man'] + glove['woman'])

queen 5.955312
monarch 6.899857
mother 7.178615
princess 7.252288
daughter 7.277298
```

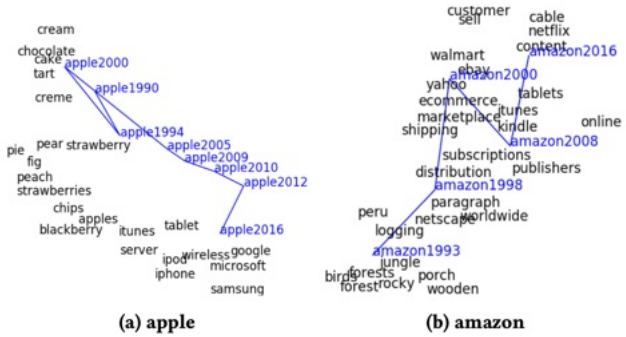
Embeddings can help study word history!

- Learn word embeddings for the same word separately from old publications vs. modern text.



Visualizing Changes

- For analysis and visualization purposes, we often project 300D vectors into 2D space.



Word Embeddings Resources

- Word2Vec:

  https://code.google.com/archive/p/word2vec/

  300-dimensional vectors trained on 100B tokens from Google News dataset.

- Glove:

  https://nlp.stanford.edu/projects/glove/

  300-dimensional vectors trained on 6B tokens from Wikipedia+Gigaword 5; 42-840B tokens from Common Crawl; 27B tokens Twitter.

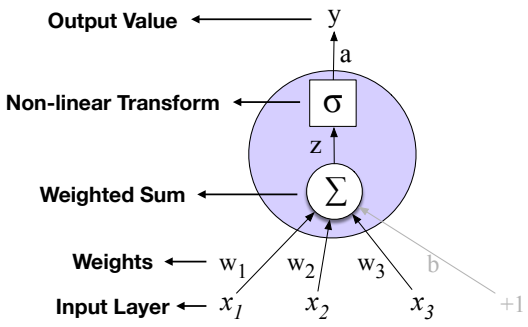
CS 5134/6034 Natural Language Processing
Fall 2024

Lecture 16: Neural Networks for NLP

University of CINCINNATI

Neural Network

A modern **neural network** is a network of small computing units, each of which takes a vector of input values and produces a single output value.



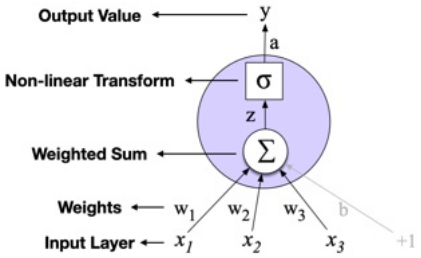
Neural Unit

- One neural unit takes a weighted sum of its inputs, with an additional term called **bias**.

$$z = w \cdot x + b = (\sum_i w_i x_i) + b$$

- Instead of generating z directly, another non-linear function f is often applied to z before output, which is called the **activation function:**

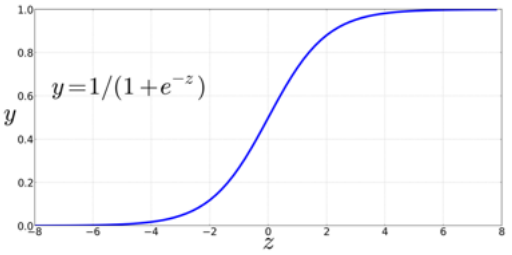
$$y = f(z)$$



Non-Linear Activation Functions

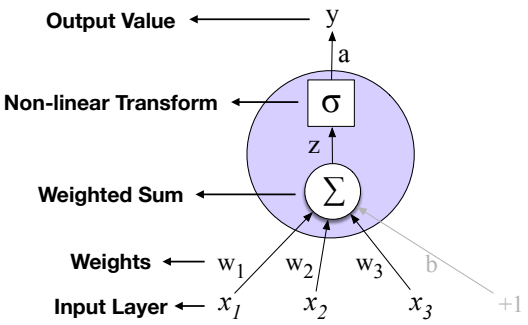
- Recall the sigmoid function for logistic regression:

$$\sigma(z) = \frac{1}{1 + e^{-x}}$$



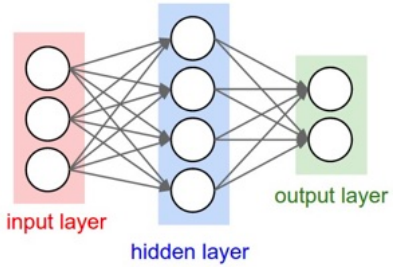
Neural Unit

$$\sigma(z) = \frac{1}{1 + e^{-(w \cdot x + b)}}$$



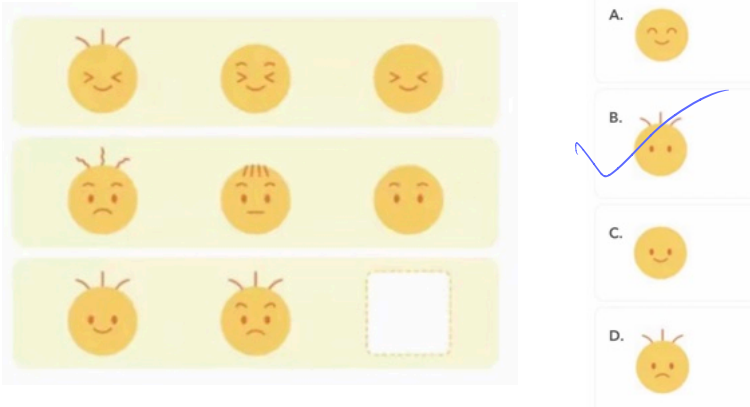
Neural Networks for NLP

- Neural networks are machine learning models typically with at least 3 layers of nodes(units):
  - input nodes
  - 1 or more layers of hidden units ("deep" neural nets can have many layers of hidden units)
  - output nodes



Neural Networks for NLP

- Neural networks have achieved great performance across many areas of AI, including NLP over the last decade.
- **Deep learning** refers to neural networks that have layers of hidden units.
- Neural nets are typically supervised learning models that perform best (relative to other methods) when trained with large amounts of data.



Logical Functions

- Can neural units compute simple logical functions?

AND			OR			XOR		
x1	x2	y	x1	x2	y	x1	x2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

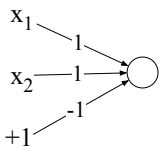
Perceptrons

- A perceptron is a neural unit that has a binary output:

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$

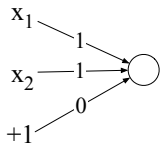
- Perceptrons can recognize **AND** and **OR** functions.

AND		
x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1



AND

OR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1



OR

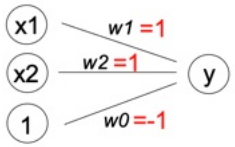
Perceptrons

- A perceptron is a neural unit that has a binary output:

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$

- Perceptrons to recognize the **AND** function:

AND		
x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1



x1	x2	y	
0	0	0	f(0*1 + 0*1 + 1*-1) = f(-1)
0	1	0	f(0*1 + 1*1 + 1*-1) = f(0)
1	0	0	f(1*1 + 0*1 + 1*-1) = f(0)
1	1	1	f(1*1 + 1*1 + 1*-1) = f(1)

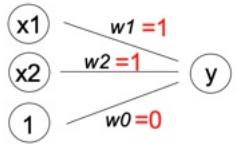
Perceptrons

- A perceptron is a neural unit that has a binary output:

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$

- Perceptrons to recognize the **OR** function:

OR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1



x1	x2	y	
0	0	0	f(0*1 + 0*1 + 1*0) = f(0)
0	1	1	f(0*1 + 1*1 + 1*0) = f(1)
1	0	1	f(1*1 + 0*1 + 1*0) = f(1)
1	1	1	f(1*1 + 1*1 + 1*0) = f(2)

Perceptrons

- A perceptron is a neural unit that has a binary output:

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$

- Perceptrons can recognize **AND** and **OR** functions.
- Can you come up with a perceptron that captures **XOR**?

XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Why? Perceptrons are linear classifiers!

- Perceptron equation given x_1 and x_2, is the equation of a line

$$w_1 x_1 + w_2 x_2 + b = 0$$

i.e.,

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$$

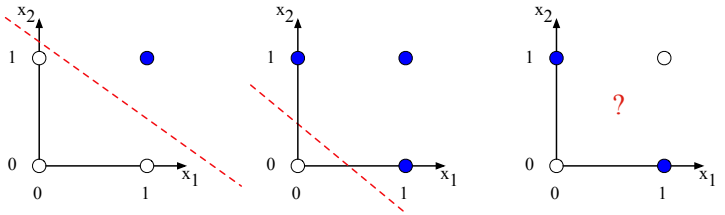
- This line acts as a decision boundary

  0 if input is on one side of the line

  1 if on the other side

Decision Boundaries

XOR is not a linearly separable function!



a) x_1 AND x_2 b) x_1 OR x_2 c) x_1 XOR x_2

AND		
x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

OR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

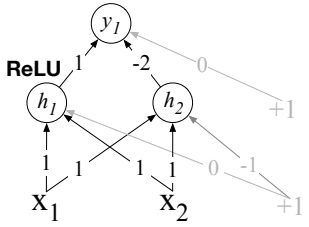
XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Neural Network for XOR

- If you create a neural network with a layer of hidden units and the ReLU activation function, then you can represent XOR.

ReLU: $f(z) = \max(0,z)$

XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



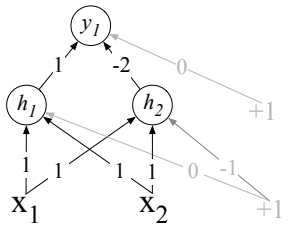
$h_1 = \max(x_1 + x_2, 0)$

$h_1 = \max(x_1 + x_2 - 1, 0)$

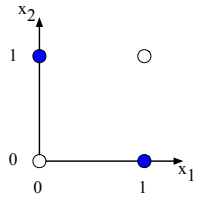
$y = h_1 - 2h_2$

x1	x2	h1	h2	y
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	2	1	0

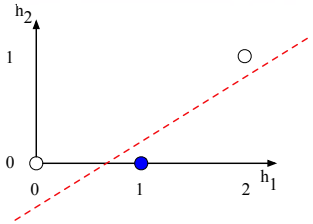
Hidden Layer h



x1	x2	h1	h2	y
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	2	1	0



a) The original x space



b) The new (linearly separable) h space

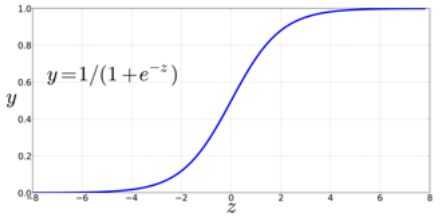
Activation Functions - Sigmoid

- Sigmoid

$$f(z) = \frac{1}{1 + e^{-x}}$$

- Derivative:

$$f'(z) = f(z) \times (1 - f(z))$$



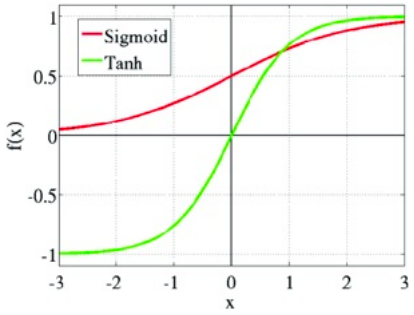
Activation Functions - Tanh

- Tanh

$$f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

- Derivative:

$$f'(z) = 1 - f(z)^2$$



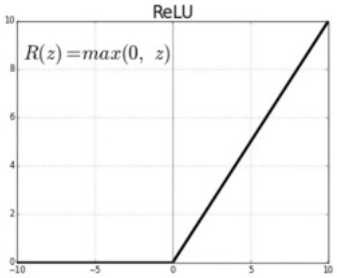
Activation Functions - ReLU

- ReLU (rectified linear unit)

$$f(z) = \max(0,\ z)$$

- Derivative:

$$f'(z) = \begin{cases} 1, & z > 0 \\ 0, & z < 0 \end{cases}$$



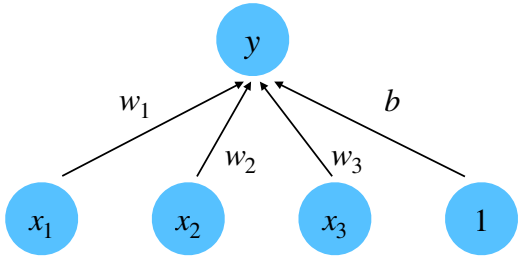
Activation Functions - Softmax

- For classification tasks, we often want a probability distribution over a set of classes. But the values across the output nodes are not a probability distribution.
- The softmax function can be applied to transform the real-valued output vector into a probability distribution by normalizing the values across the output vector.
- For a vector z with d dimensions, the formula is:

$$softmax_i(z) = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}} \quad 1 \le i \le d$$

Binary Logistic Regression as a Neural Net

- $y = \sigma(w \cdot x + b)$. d is input dimension.



Input layer: $x \in \mathbb{R}^d$

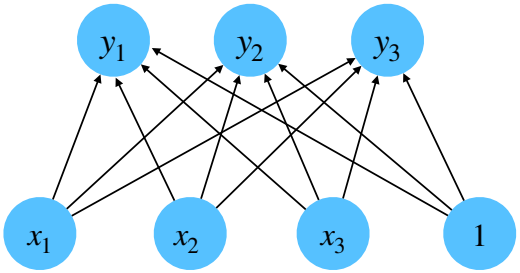
Output layer: $y \in \mathbb{R}$

Weight: $W \in \mathbb{R}^d$

Bias: $b \in \mathbb{R}$

Mutinomial Logistic Regression as a Neural Net

- $y = \text{softmax}(W \cdot x + b)$. C is the number of classes. d is input dimension.



Input layer: $x \in \mathbb{R}^d$

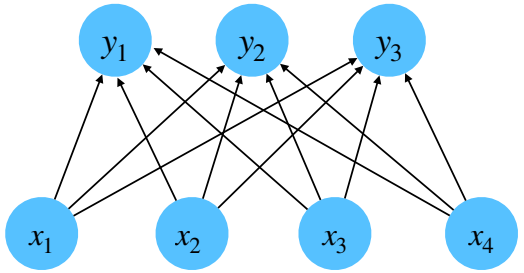
Output layer: $y \in \mathbb{R}^C$

Weight: $W \in \mathbb{R}^{C \times d}$

Bias: $b \in \mathbb{R}^C$

Replace the Bias Unit

- For notation simplicity, from now on, we will assume an extra element 1 in the input layer, and omit the bias term in the notation.



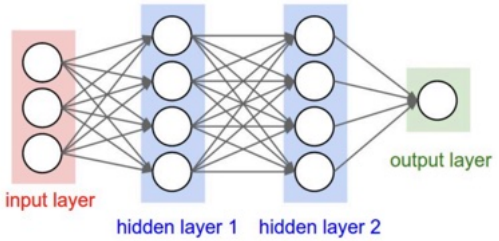
Input layer: $x \in \mathbb{R}^d$

Output layer: $y \in \mathbb{R}^C$

Weight: $W \in \mathbb{R}^{C \times d}$

Feedforward Neural Networks

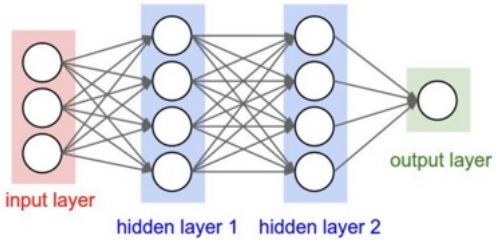
- **Feedforward neural nets** have at least one layer of hidden units and no cycles.
- It is also called **multi-layer perceptrons** (or MLPs).



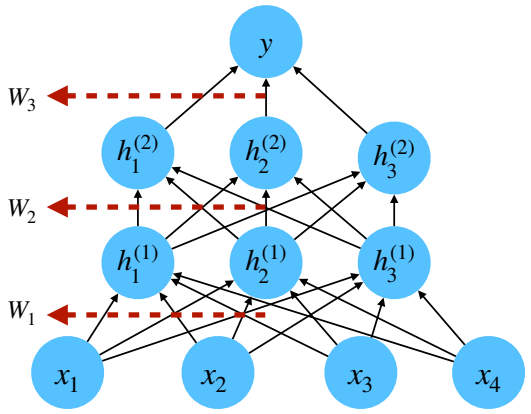
Feedforward Neural Networks

- Each layer is usually **fully-connected (FC)** to the next layer. We often talk about the weights as a matrix W, where the value for the hidden units are:

$$h = f(Wx)$$



Muti-layer Feedforward Neural Networks



- Input layer: $x \in \mathbb{R}^d$
- Hidden layer 1:

$$h^{(1)} = f(W_1 x),\ W_1 \in \mathbb{R}^{d_1 \times d}$$

- Hidden layer 2:

$$h^{(2)} = f(W_2 h^{(1)}),\ W_2 \in \mathbb{R}^{d_2 \times d_1}$$

- Output layer:

$$y = f(W_3 h^{(2)}) \in \mathbb{R}^C,\ W_3 \in \mathbb{R}^{C \times d_2}$$

Real Examples of Feedforward Networks

- Now let's see 2 examples of using feedforward neural networks for nlp tasks:

1. Text Classification
2. Language Modeling

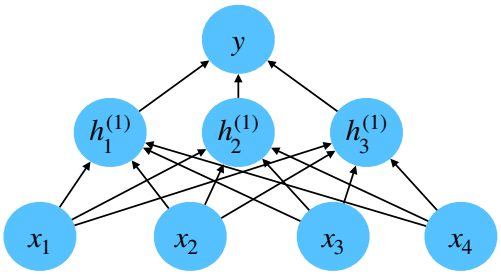
Feedforward Networks for Text Classification

- Let's re-consider the **sentiment analysis** task.
- Input is manually defined features.
- Output is 0 or 1.

Var	Definition
x_1	count(positive lexicon) $\in$ doc)
x_2	count(negative lexicon) $\in$ doc)
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns $\in$ doc)
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_6	log(word count of doc)

Add one layer to logistic regression

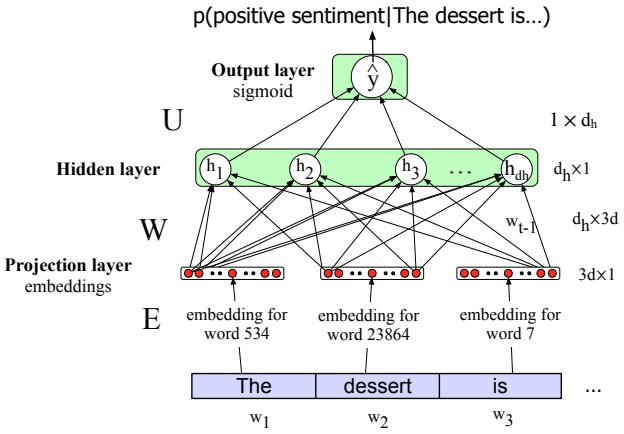
- Recall we have used logistic regression for sentiment analysis: $P(+|x) = \sigma(w \cdot x + b)$
- Let's add one hidden layer to it to allow non-linear interactions between features.



Word Embeddings as Features

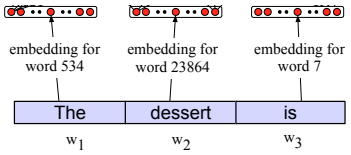
- The real power of deep learning comes from the ability to learn features from the data.
- Instead of using hand-built human-engineered features for classification,
- We can use learned representations like word embeddings!

Word Embeddings as Features



Issue: Text with Different Lengths

- This assumes a fixed size length (3).
- But text come in with different lengths.
- Some simple solutions:
  1. make the input length of the longest text
     I. if shorter, then pad with zero embeddings
     II. if longer (during test time), truncate them
  2. create a single embedding (the same dimension as one word)
     I. Take the mean of all word embeddings in the text
     II. Take the element-wise max of all the word embeddings



Neural N-gram Language Models

- Language modeling: computing the probability of the next word given previous words.

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_1, \ldots, w_{i-1})$$

- We have introduced n-gram language models, such as trigram:

$$P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-2}, w_{i-1}),$$

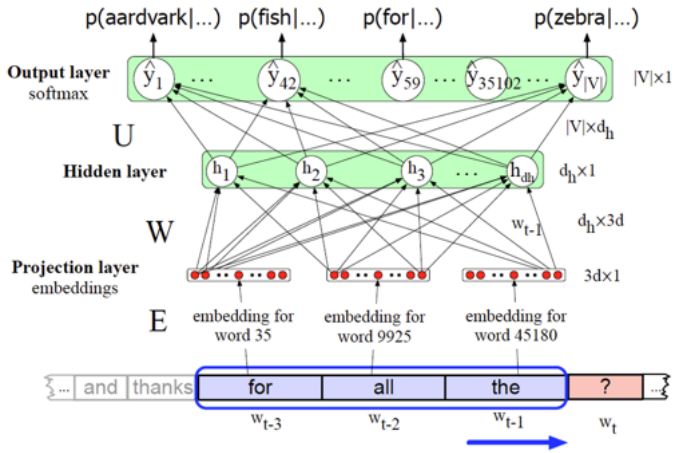
where $P(w_i \mid w_{i-2}, w_{i-1}) = \dfrac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})}$

- How can we model this with a neural net?

Neural N-gram Language Models

- How can we model this with a neural net?
  - input layer: N-1 word vectors
  - output layer: a softmax over $|V|$ units
- Key idea: replace raw probabilities from the corpus with estimation from a neural network!

Neural N-gram Language Models



Neural N-gram Language Models

- Goal:

  $\arg\max_w$ P(w | for all the) = ?

- Input layer:

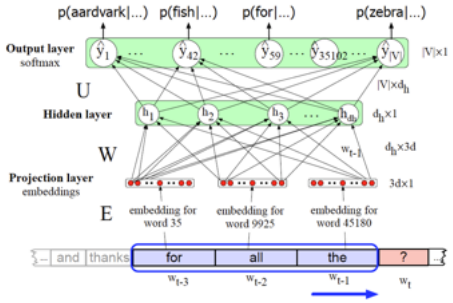
  $x = [e(\text{for}); e(\text{all}); e(\text{the})] \in \mathbb{R}^{3d}$

- Hidden layer:

  $h = \tanh(Wx) \in \mathbb{R}^{d_h}$

- Output layer:

  $z = Uh \in \mathbb{R}^{|V|}$, P(w | for all the) = $\text{softmax}_i(z) = \frac{e^{z_i}}{\sum_j e^{z_j}}$



Generalization Ability of Neural LM

- In the training data:

  You can find the answer in the book.

- In the test set:

  You can find the solution in ___ ?

- N-gram language models cannot predict something not seen before.

- Neural language models can use the similarity between “answer” and “solution” for prediction. Word embeddings provide **generalization** across similar contexts.

$U_{|V| \times 100}(W_{100 \times 900} x_{900 \times 1}) \rightarrow$ softmax.

$|V| \times 1$

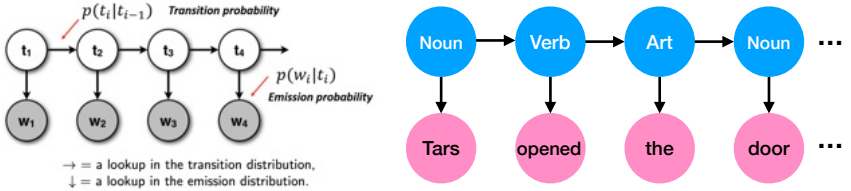
CS 5134/6034 Natural Language Processing
Fall 2024

Lecture 17: Recurrent Neural Networks

University of CINCINNATI

Neural Models for Sequence Labeling

- Recall we have built statistical model for sequence labeling tasks, such as part-of-speech tagging, named entity recognition, etc.



- How can we model sequences using neural networks?

Recurrent Neural Network (RNN)

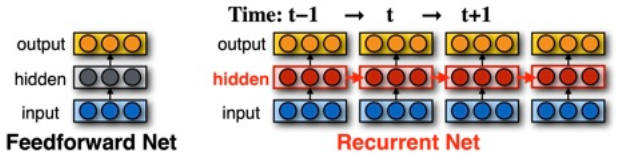
- Feedforward networks can only handle input and output that have a fixed size.
- Recurrent neural networks can allow **variable length sequences** (for both inputs and outputs)
- This is very important for NLP tasks because texts come in with different lengths.

Recurrent Neural Networks

- RNN processes a sequence of T inputs.
- **Recurrence**:

  The hidden state computed at the previous step $h^{(t-1)}$ is fed into the hidden state at the current step $h^{(t)}$.

- Each time step t corresponds to a feedforward network whose hidden layer $h^{(t)}$ gets input from the layer below — $x^{(t)}$ **AND** the hidden layer values from the previous time step — $h^{(t-1)}$.



Recurrent Neural Networks

- Each time step t corresponds to a feedforward network whose hidden layer $h^{(t)}$ gets input from the layer below — $x^{(t)}$ **AND** the hidden layer values from the previous time step — $h^{(t-1)}$.
- Hidden states at time step t:

$$h^{(t)} = g(Uh^{(t-1)} + Wx^{(t)})$$

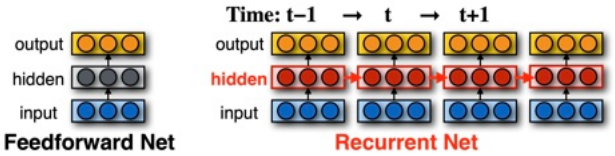
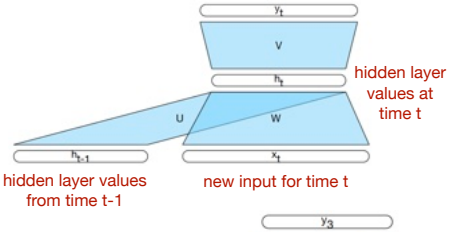
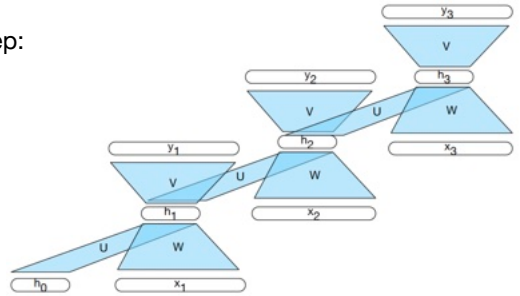


Illustration of the Recurrent Process

Picture of 2nd time step:

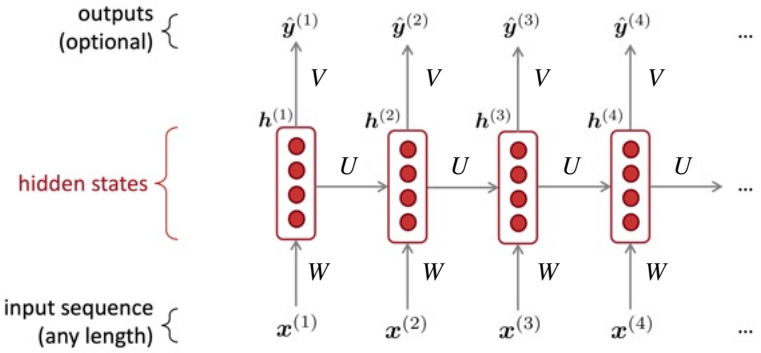


Picture of 4th time step:

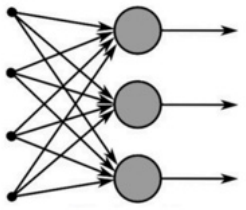


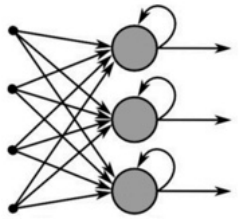
Recurrent Neural Networks

- Weight matrix W, U and V are applied repeatedly.



RNNs vs. Feedforward NNs


Feed-Forward Neural Network


Recurrent Neural Network

$$h = g(Wx)$$

$$h^{(t)} = g(Uh^{(t-1)} + Wx^{(t)})$$

RNNs for language modeling

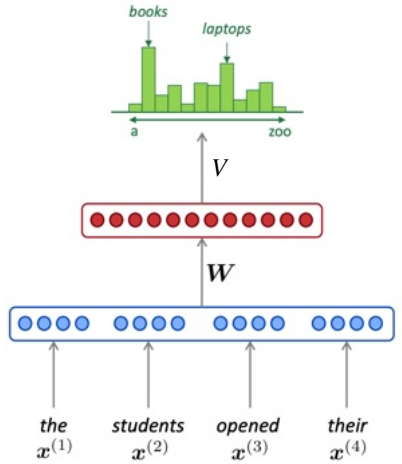
Recap: Feedforward Neural Language Model

output distribution
$y = \text{softmax}(Vh) \in \mathbb{R}^l$,
where l is the vocabulary size

hidden layer
$h = g(Wx) \in \mathbb{R}^{d_h}$

input layer
$x = [e^{(1)}, e^{(2)}; e^{(3)}; e^{(4)}] \in \mathbb{R}^{4d}$

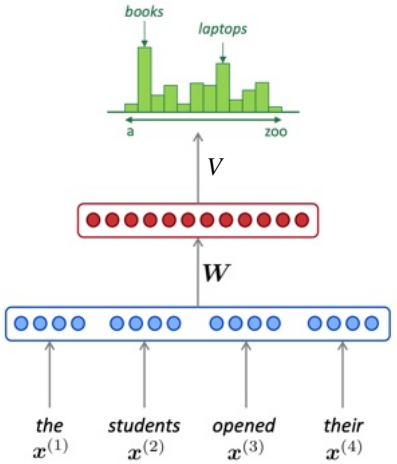
words
(fixed size context window)



Recap: Feedforward Neural Language Model

- Improvements over n-gram language model:
  - no sparsity problem
  - can use pre-trained word embeddings
  - can handle unseen words
  - better generalization
- Remaining problems:
  - Fixed context window

We need a neural model that can handle any length input!



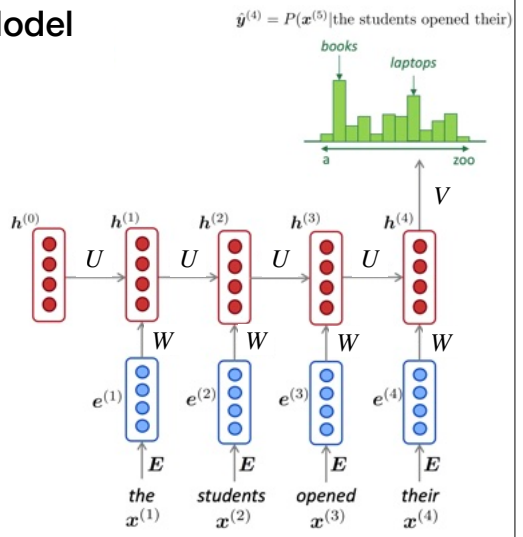
A RNN Language Model

output distribution
$y^{(t)} = \text{softmax}(Vh^{(t)}) \in \mathbb{R}^l$,
where l is the vocabulary size

hidden layer
$h^{(t)} = g(Uh^{(t-1)} + We^{(t)}) \in \mathbb{R}^{d_h}$
$h^{(0)}$ is the initial hidden state

input embeddings
$e^{(t)} = Ex^{(t)}$,
E is the embedding matrix

words (one-hot vectors)
$x^{(t)} \in \mathbb{R}^l$



This input sequence can be however long!

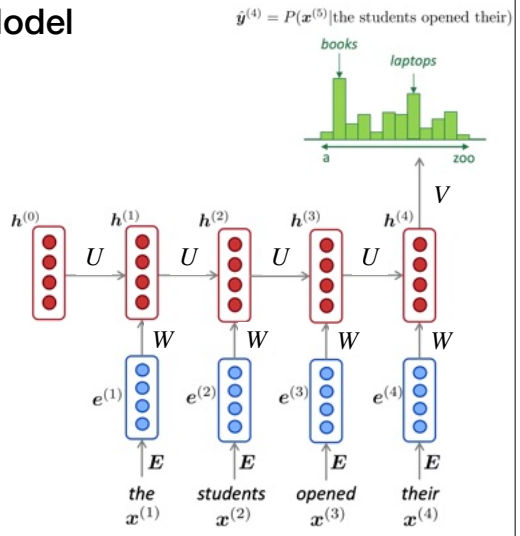
A RNN Language Model

- Improvements over feedforward neural n-gram language model:

  — can take any length input

  — capture longer dependence

  — model size will not increase as sequence length increases



This input sequence can be however long!

Training a RNN Language Model

- To train an RNN language model, we adopt the same self-supervision paradigm (no human labeled data).
- We use a big corpus of text, and at each time step t, we ask the model to predict the next word.
- Loss function is the cross-entropy between our predicted distribution and correct distribution.

$$L_{CE} = -\sum_{w \in V} y^{(t)}[w] \ \log \hat{y}^{(t)}[w]$$

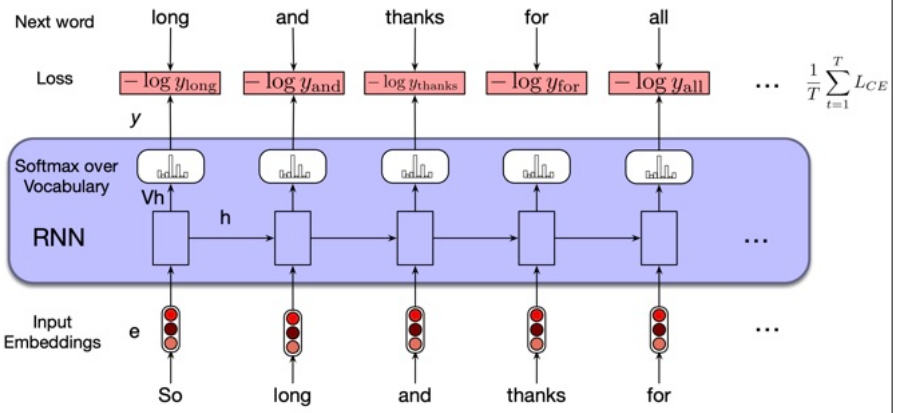
- Since the correct distribution is a one-hot vector where the entry for the actual next word is 1, we have:

$$L_{CE}(\hat{y}^{(t)}, y^{(t)}) = -\log \hat{y}^{(t)}[w_{t+1}]$$

- So we have the training loss:

$$L(\theta) = -\frac{1}{T}\sum_{t=1}^{T} \log \hat{y}^{(t)}[w_{t+1}]$$

Training a RNN Language Model



Weight Tying

$e^{(t)} = Ex^{(t)}$

$h^{(t)} = g(Uh^{(t-1)} + We^{(t)})$

$\hat{y}^{(t)} = \text{softmax}(Vh^{(t)})$

- Trainable parameters:

  $\theta = \{W, U, V, E\}$

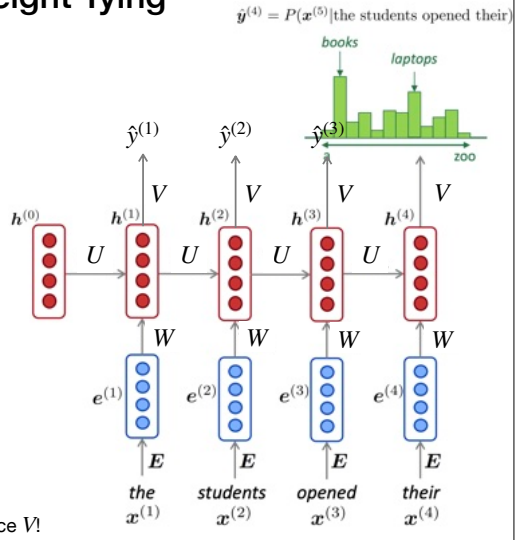
- Embedding matrix $E \in \mathbb{R}^{l \times d}$,

  where l is vocab size, d is embedding dimension

- Last layer matrix $V \in \mathbb{R}^{l \times d_h}$,

  where d_h is the hidden state size.

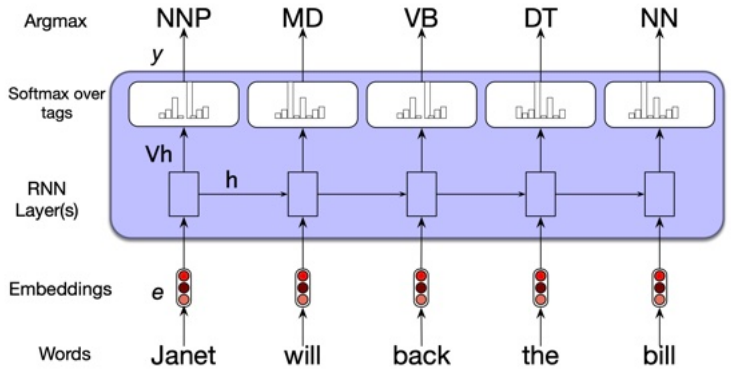
- Let $d = d_h$, then we can use E to replace V!



RNNs for Other NLP Tasks

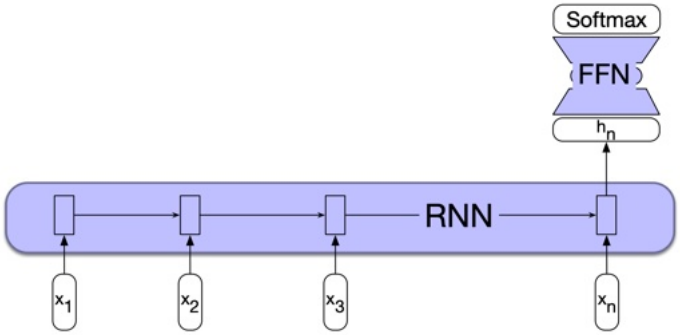
Sequence Labeling

- Inputs are word embeddings and outputs are tag probabilities from a softmax function.



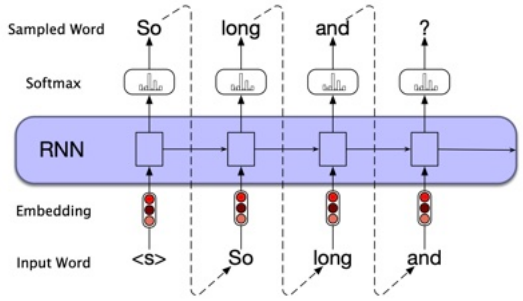
Text Classification

- We take the hidden layer for the last token of the text, to pass it to another feedforward network to choose a class.



Language Generation

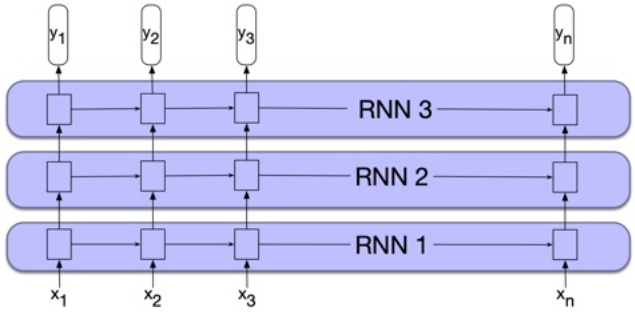
- Text generation, together with image/code/video generation, nowadays often called "**generative AI**".
- Using a language model to incrementally generate the next word based on previous choices is called **autoregressive generation**.



Stacked and Bidirectional RNNs

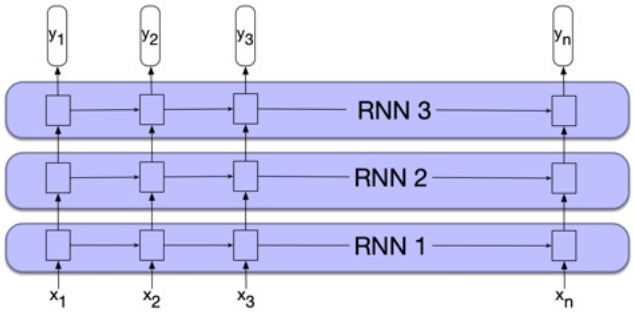
Stacked RNNs

- RNNs are already “deep” in one dimension (time step)
- We can make them deep in another dimension — stacking them like feedforward nets.



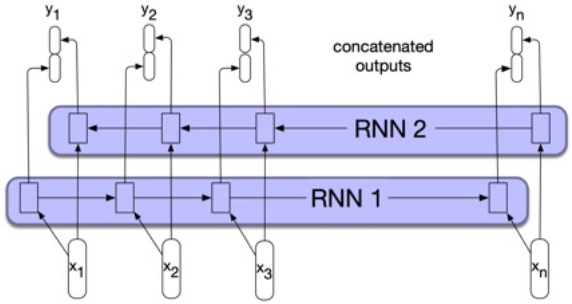
Stacked RNNs

- In practice, using 2 to 4 layers outperforms the single layer.
- Different layers captures different levels of abstractions.



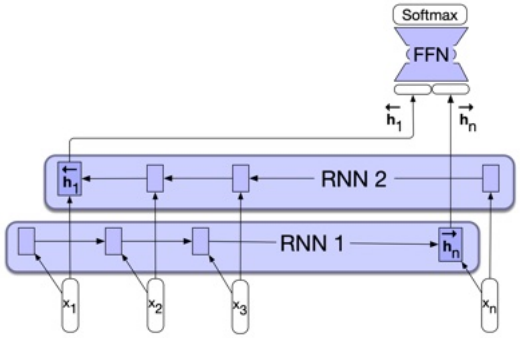
Bidirectional RNNs

- RNNs use information from left to right. But for tasks like text classification, we see the whole sentence at a time.
- A bidirectional RNN combines two independent RNNs, one from left to right, the other from the end to the start.



Bidirectional RNNs

- Then we concatenate the two representations into one vector.
- For sequence classification, we concatenate the last hidden states from the forward and backward pass ($\vec{h}_n$ and $\vec{h}_1$).



RNNs Variations

RNN Variants: LSTMs, GRUs

- **Long Short-Term Memory networks (LSTMs)** are RNNs with a more complex recurrent architecture.
- **Gated Recurrent Units (GRUs)** are a simplification of LSTMs.
- They use a mechanism called “gate” to control how much to forget or remember.

Vanishing and Exploding Gradient

- In simple RNNs (also called **vanilla RNN**), the current hidden state $h^{(t)}$ is a nonlinear function of the previous hidden state $h^{(t-1)}$ and the current input $x^{(t)}$:

$$h^{(t)} = g(Uh^{(t-1)} + Wx^{(t)})$$

- With $g = \tanh$, models suffer from the **vanishing gradient** problem: the gradients are eventually driven to zero as sequence get longer.
- With $g = \textbf{ReLU}$, models suffer from the **exploding gradient** problem: the gradients are becoming too large.

Gradient Clipping

- One solution for gradient exploding is called gradient clipping: we scale the gradient down if it is larger than some threshold.

Algorithm 1 Pseudo-code for norm clipping

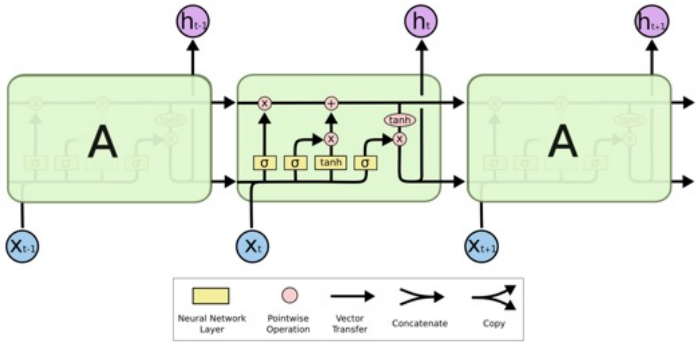
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

LSTMs

- LSTMs (Long Short-Term Memory networks) were introduced to overcome the vanishing gradient problem.
- LSTMs contain an additional **memory cell state** that also gets passed through the network and updated at each time step.
- There are three different **gates (input/forget/output)** that decide how much of the past hidden and cell states to keep.
  - **forget gate**: decides which part of the memory to drop
  - **input gate**: decides which part of the input to add to the memory
  - **output gate**: decides which part of the memory to use in the hidden state

LSTMs

- Key idea: turning **multiplication** into **addition** and using "gates" to control how much information to add/erase.



LSTMs

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

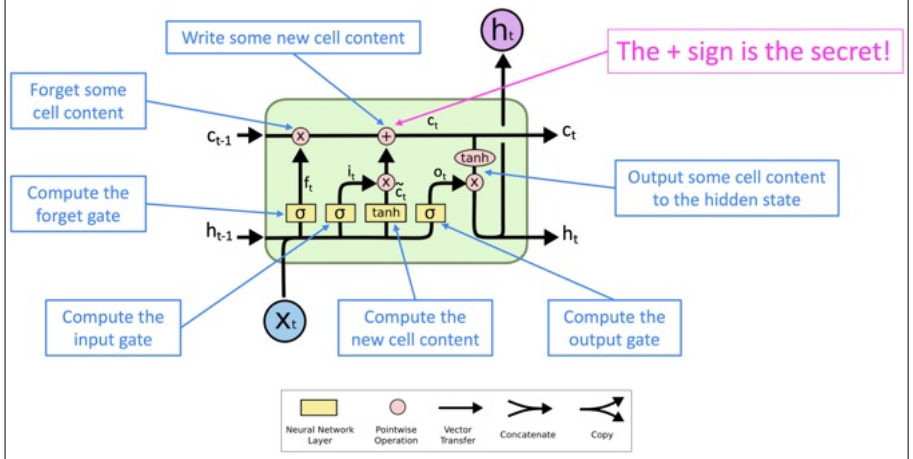
Sigmoid function: all gate values are between 0 and 1

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$
$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$
$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$
$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$
$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$
$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

All these are vectors of same length n

Gates are applied using element-wise (or Hadamard) product: $\odot$

LSTMs



How does LSTM solve vanishing gradients?

- LSTMs make it much easier as compared to RNNs to preserve information over many time steps.
- For example, if the forget get is set to 1 for a cell dimension and the input gate set of 0, then that cell is preserved indefinitely.
- However, LSTM doesn't guarantee that there is no vanishing/ exploding gradient.
- In practice, you get about 100 time steps rather than about 7 (vanilla RNNs).

CS 5134/6034 Natural Language Processing
Fall 2024

Lecture 18: Machine Translation and Seq2Seq Models

University of CINCINNATI

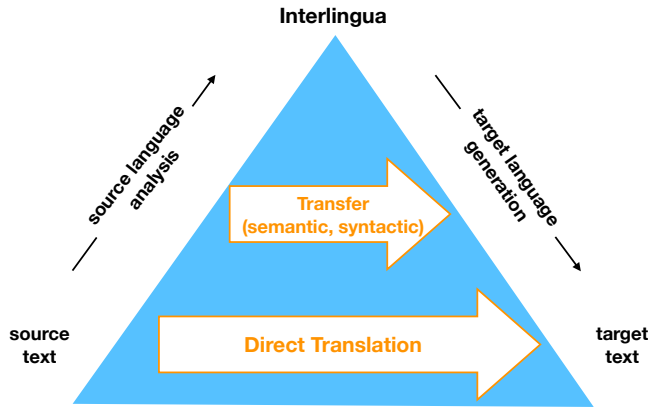
Machine Translation

- Machine Translation (MT) systems automatically translate one natural language (the **source** language) to another (the **target** language).
- High-quality MT could greatly enhance global communication.
- But ... MT is challenging because it requires full understanding as well as generation.
- Early MT systems ran into such difficulties that MT research stopped for many years! But MT has seen a resurgence.

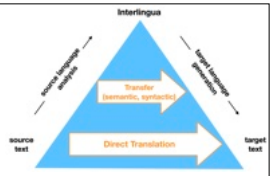
ALPAC Report

- ALPAC (Automatic Language Processing Advisory Committee) was a committee established in 1964 by the US government to evaluate the progress in computational linguistics, particularly machine translation.
- In 1966, the ALPAC report came out: Human translation is far cheaper and better.
- It marked the beginning of the first AI winter, killing machine translation for a long time.

The Vauquois Triangle

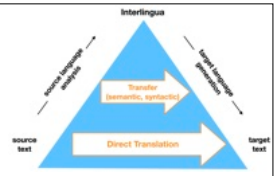


Direct Translation



- **Direct machine translation** translates one language directly into another without an intermediate representation.
- Direct MT systems typically use word for word translation based on a bilingual dictionary.
- Local word reordering can be used to smooth the target language output.
- Direct MT methods are inherently limited and usually don't perform very well.

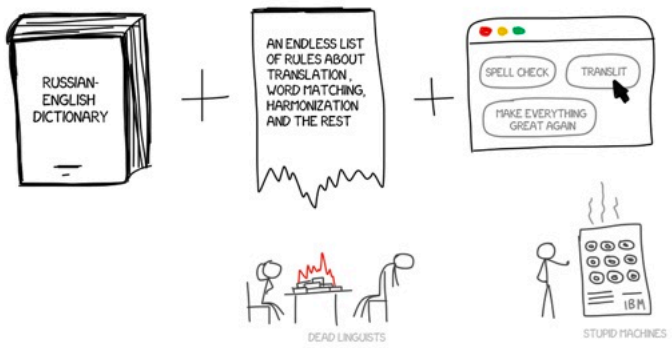
Direct Translation: Word for Word



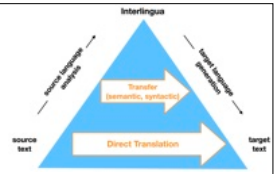
Many cheap commercial systems do word for word translation, despite its inherent problems, which include:

- **Lexical (Word Sense) Ambiguity**: shot, bat
- **Synonyms & Subtleties**: run, jog, sprint, trot, flee
- **Phrases**: hot dog, real estate, dry run, operating system
- **Idioms**: raining cats and dogs, buried the hatchet, kicked the bucket
- **Transposition**: Word order varies a lot, especially between different types of languages.

Rule-based Machine Translation



Indirect Translation



- **Indirect machine translation** uses an intermediate representation to capture meaning.
- An **interlingua** is a language-independent meaning representation. Analysis of the source language produces a representation in the interlingua, which is then used to produce output in the target language.
- Interlingua-based MT allows for multilingual systems that can translate to and from any pair of languages.
- Another approach is **transfer** systems that use language-dependent intermediate representations such as semantic and syntactic structures.

Machine Translation Evaluation

- What do we need to evaluate?
  - **Adequacy**: translation should remain the same meaning
  - **Fluency**: translation should be fluent in the target language
- We need appropriate evaluation metrics!
- **Human evaluation**:

  Expensive, and not easily reproducible or comparable across evaluations.
- **Automatic evaluation**:

  Inexpensive, can be done on a large scale, but may not be as accurate as human evaluation.

Automatic Evaluation: BLEU

- **BLEU** (Bilingual Evaluation Understudy) compares a **machine (candidate)** translation to **human (reference)** translation(s).
- It is based on N-gram precision p_n, i.e., how many n-grams in the candidate translation also occur in the reference translation.
- It also contains **a brevity penalty** that penalize short candidate translations.
- Finally the BLEU score is the production of the geometric mean of p_n $(n = 1..4)$ and the brevity penalty.

Statistical Machine Translation (SMT)

- SMT began development since 90s, and became popular in 2000s.
- Given a source language text S, we find to generate a target language text T:

$$\arg\max_T P(T|S)$$

- Using bayes' rule:

$$\arg\max_T P(S|T)\ P(T)$$

The translation model $P(S|T)$ captures the adequacy of the translation. Trained on a parallel corpus.

The language model $P(T)$ captures the fluency of the translation. Trained on the monolingual corpus.

Parallel Corpus

- SMT systems are trained with **parallel text corpora**, which consist of source language texts coupled with their manual translations into a target language.
- The Hansard parallel corpus is widely used and contains the proceedings of the Canadian Parliament in both French and English. The European Union also has parallel corpora for many European languages.
- The Web contains many sites that have multiple translations. However, the translations often are not parallel.
- Difficult to find them for low-resource languages.

Automatic Evaluation: BLEU

$$\text{BLEU} = \underbrace{\min\Big(1, \exp\big(1 - \frac{\text{reference-length}}{\text{output-length}}\big)\Big)}_{\text{brevity penalty}} \underbrace{\Big(\prod_{i=1}^{4} precision_i\Big)^{1/4}}_{\text{n-gram overlap}}$$

with

$$precision_i = \frac{\sum_{\text{snt} \in \text{Cand-Corpus}} \sum_{i \in \text{snt}} \min(m^i_{cand}, m^i_{ref})}{w^i_t = \sum_{\text{snt}' \in \text{Cand-Corpus}} \sum_{i' \in \text{snt}'} m^{i'}_{cand}}$$

where

- m^i_{cand} is the count of i-gram in candidate matching the reference translation
- m^i_{ref} is the count of i-gram in the reference translation
- w^i_t is the total number of i-grams in candidate translation

Alignment

- To get $P(S \mid T)$, we need to align source and target words.
- Even in parallel corpora, sentences and paragraphs do not always perfectly correspond. For example, one French sentence may be translated as two English sentences.
- Some information (even entire sentences or paragraphs!) may also be present in one but missing in the other.
- **Sentence alignment** algorithms attempt to align the corresponding sentences across parallel texts.
- **Word alignment** algorithms attempt to align the corresponding words across parallel sentences

1990s-2010s: Statistical Machine Translation

- SMT was a huge research field.
- The best systems are extremely complex.
- Systems have many separately-designed subcomponents

  — Lots of feature engineering

  — Need to design features to capture particular language phenomena

  — Require compiling and maintaining extra resources, such as tables of equivalent phrases

  — Lots of human effort to maintain — repeated effort for each language pair!

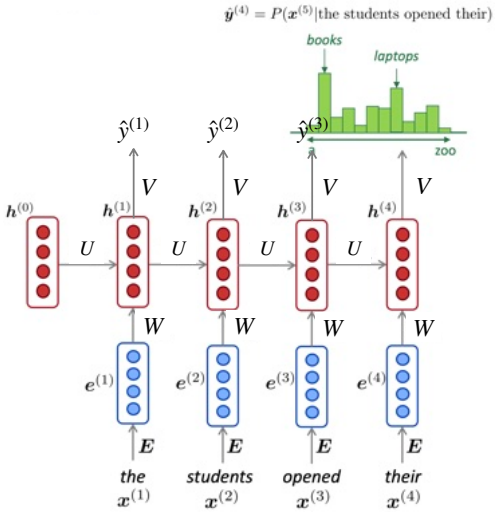
Neural Machine Translation (NMT)

- The neural machine translation is a way to do Machine Translation with a neural network.
- Machine translation inspired the creation of **sequence-to-sequence (seq2seq)** architectures:

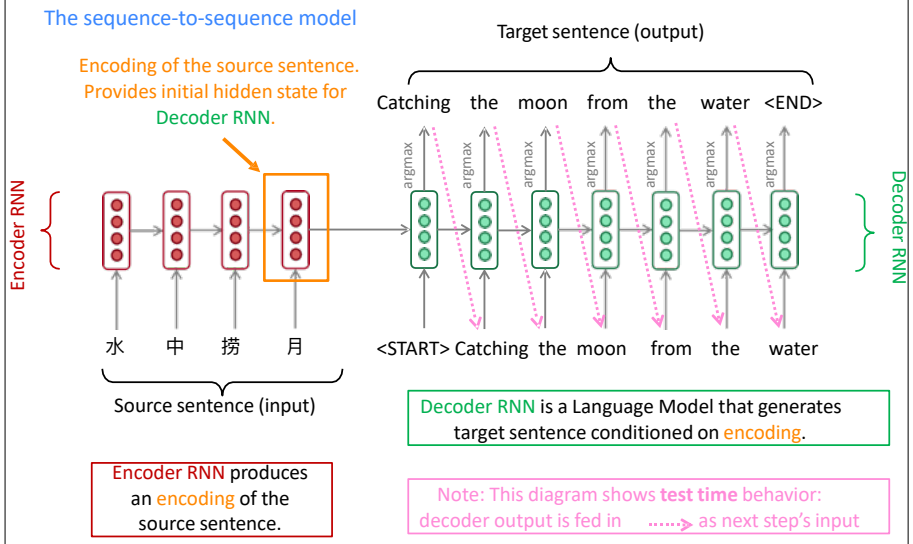
  The **encoder** reads in source language input,

  The **decoder** returns target language output (translation)
- The concept of word alignments inspired **attention** in seq2seq models.
- Seq2seq is also called **end-to-end** neural nets.

Recap: RNN Language Model



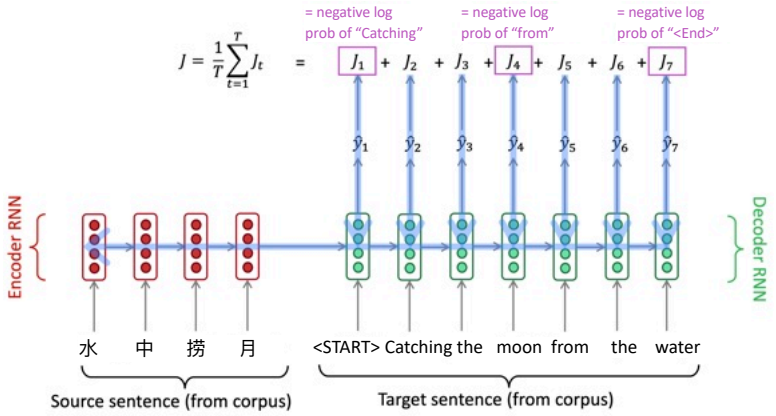
Encoder-Decoder Architecture



Seq2Seq Models

- The seq2seq model is an example of a conditional language model.
  - — **language model** because the decoder is predicting the next word of the target sentence
  - — **conditional** because its predictions are also conditioned on the source sentence
- NMT **directly** calculates $P(T \mid S)$
  - — no more bayes' rule

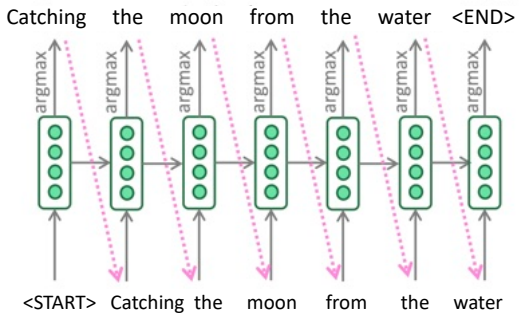
Training Seq2Seq Models



Seq2seq is optimized as a single system. Backpropagation operates *"end to end"*.

Decoding Seq2Seq Models

- Greedy decoding: at each step, take the most probable word.
- Better methods?



Decoding with Beam Search

- Instead of getting the argmax, we keep track of the k **most probable partial translations** at each step.
  - k is the **beam size** (in practice around 5 to 10)
  - not guaranteed to find optimal solution
  - but better than greedy decoding in practice

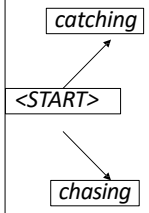
Beam Search Decoding Example

Beam size = 2

<START>

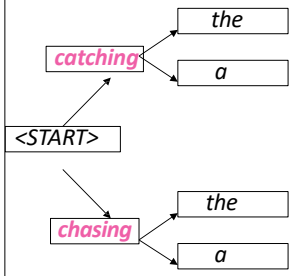
Beam Search Decoding Example

Beam size = 2



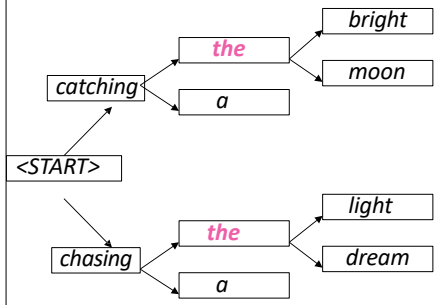
Beam Search Decoding Example

Beam size = 2



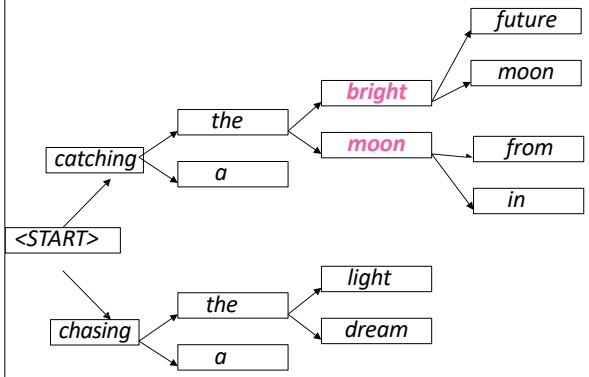
Beam Search Decoding Example

Beam size = 2



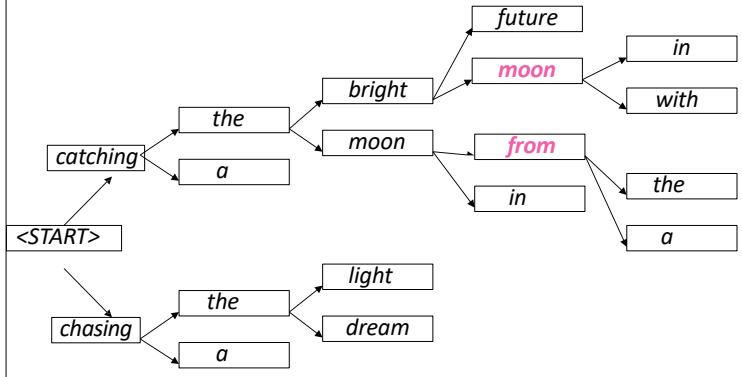
Beam Search Decoding Example

Beam size = 2



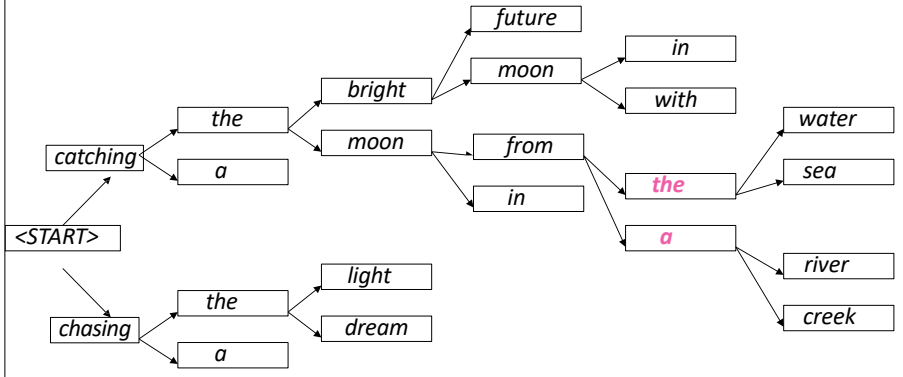
Beam Search Decoding Example

Beam size = 2



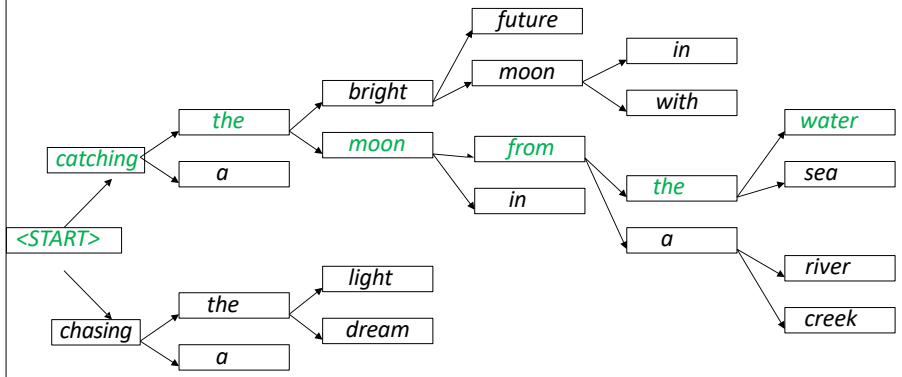
Beam Search Decoding Example

Beam size = 2



Beam Search Decoding Example

Beam size = 2



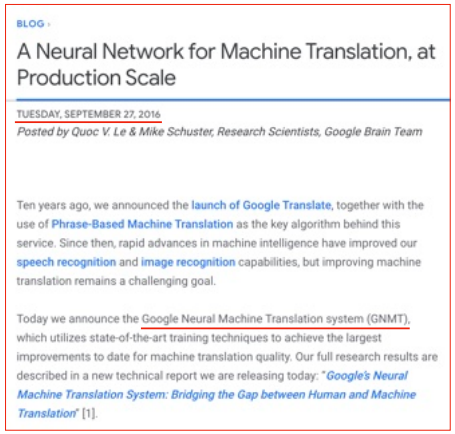
Backtrack

Pros and Cons of NMT

- Compared to SMT, NMT has many advantages:
  - — Better performance (more fluent, better use of context, better use of phrase similarities)
  - — One single neural network to be optimized end-to-end (no individual subcomponents)
  - — Less human engineering effort (no feature engineering; same method for all language pairs)
- Cons:
  - — NMT is less interpretable
  - — NMT is difficult to control (cannot easily specify rules for translation)

NMT: A Big Success of Deep Learning in NLP

- 2006: Google launched the web-based free-to-user translation service developed. It was a SMT system.
- 2014: First seq2seq paper published.
- 2016: Google Translate switches from SMT to NMT.
- SMT systems, built by hundreds of engineers over many years, outperformed by NMT systems trained by a small group of engineers in a few months!



BLOG

A Neural Network for Machine Translation, at Production Scale

TUESDAY, SEPTEMBER 27, 2016

Posted by Quoc V. Le & Mike Schuster, Research Scientists, Google Brain Team

Ten years ago, we announced the launch of Google Translate, together with the use of Phrase-Based Machine Translation as the key algorithm behind this service. Since then, rapid advances in machine intelligence have improved our speech recognition and image recognition capabilities, but improving machine translation remains a challenging goal.

Today we announce the Google Neural Machine Translation system (GNMT), which utilizes state-of-the-art training techniques to achieve the largest improvements to date for machine translation quality. Our full research results are described in a new technical report we are releasing today: "*Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*" [1].

NMT Research Continues

- In 2018, NMT research continues to thrive.
- Researchers have found many, many improvements to the "vanilla" seq2seq NMT system we've presented today.
- But one improvement is so integral that it is the new vanilla —

Attention