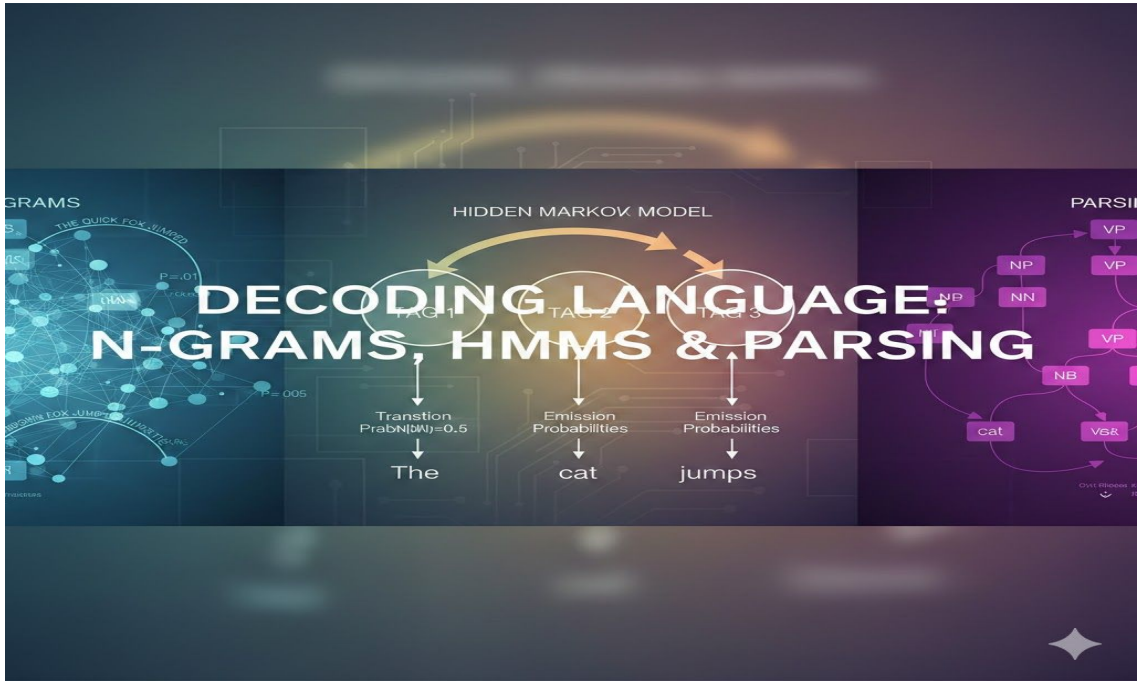


Natural Language Understanding

CSCI 4907/6515

Lecture 7: February 24, 2026

Aya Zirikly

Evaluation: how good is our model

- Does our language model prefer good sentences to bad ones? Does it assign higher probability to real or frequently encountered sentences than ungrammatical or rarely observed ones?
- We train the parameters of our model on a training set
- Evaluate on data we haven't seen
  - Test set: unseen dataset that is different from our training set, totally unseen
  - Evaluation metric: tells us how well our model does on the test set

Intrinsic vs. extrinsic evaluation

- Extrinsic evaluation: Plug our model into a down the line system (e.g., speech recognizer, machine translation system)
  - To compare models A and B, see which results in better performance on a task (e.g., how many words correctly identified? Correctly translated?)
    - Pros: What we care about
    - Cons: Time-consuming; can take days or weeks
- Intrinsic evaluation: Testing the quality of the model itself (without a specific task)
  - E.g., perplexity
  - Pros: often correlates with extrinsic performance and much easier to run
  - Cons: can be a bad approximation unless training and test sets are very similar

Intuition of Perplexity

- The Shannon Game:
  - How well can we predict the next word?

Intuition of Perplexity

- The Shannon Game:
  - How well can we predict the next word?

- I always order pizza with cheese and ____
  The 33rd President of the US was ____
  I saw a ____

mushrooms 0.1
pepperoni 0.1
anchovies 0.01
….
fried rice 0.0001
….
and 1e-100

Intuition of Perplexity

- The Shannon Game:
  - How well can we predict the next word?
  - I always order pizza with cheese and ____

    The 33rd President of the US was ____

    I saw a ____
  - Unigrams are terrible at this game. (Why?)

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

….

fried rice 0.0001

….

and 1e-100

Intuition of Perplexity

- The Shannon Game:
  - How well can we predict the next word?
  - I always order pizza with cheese and ____
  - The 33rd President of the US was ____
  - I saw a ____
  - Unigrams are terrible at this game. (Why?)

mushrooms 0.1
pepperoni 0.1
anchovies 0.01
….
fried rice 0.0001
….
and 1e-100

- A better model of a text
  - is one which assigns a higher probability to the word that actually occurs

Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest P(sentence)

Perplexity is the inverse probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Chain rule:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

For bigrams:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1})}}$$

Minimizing perplexity is the same as maximizing probability

Ex: let's take a sentence consisting of random digits (0-9)

0 0 0 0 0 0 0 3 0 0

- What is the perplexity of this sentence according to a model that assigned equal probability to all digits?
- What is the perplexity of this sentence according to a model that was trained on the following corpus?
  - 100 total digits; 91 were 0's, 1 of each of the remaining 9 digits

Ex: let's take a sentence consisting of random digits (0-9)

0 0 0 0 0 0 0 3 0 0

- What is the perplexity of this sentence according to a model that assigned equal probability to all digits?
- What is the perplexity of this sentence according to a model that was trained on the following corpus?
  - 100 total digits; 91 were 0's, 1 of each of the remaining 9 digits

Calculating perplexity

0000000300

1) What is the perplexity of this sentence according to a model that assigned equal probability to all digits (0-9) (Perplexity with Equal Probability)?

10 **Why?**

Calculating perplexity

0000000300

1) What is the perplexity of this sentence according to a model that assigned equal probability to all digits (0-9) (Perplexity with Equal Probability)?

10

When a model has no specific information, the perplexity is equal to the size of the vocabulary. Since there are 10 digits (0 through 9), each has a 1/10 (0.1) chance of occurring. The inverse of this probability (1 / 0.1) is 10

Calculating perplexity with trained corpus

0000000300

2) What is the perplexity of this sentence according to a model trained on a corpus of 100 digits where 91 were "0"s and 1 was each of the remaining 9 digits?

Because this model "expected" many zeros, it is much less "perplexed" by this sentence than the uniform model was.

Calculating perplexity with trained corpus

0000000300
2) What is the perplexity of this sentence according to a model trained on a corpus of 100 digits where 91 were "0"s and 1 was each of the remaining 9 digits?

Probabilities: Based on the corpus, the probability of a "0" is 91/100 (**0.91**) and the probability of a "3" is 1/100 (**0.01**).
Sentence Probability: The test sentence "0000000300" contains nine 0s and one 3. Its probability is: 0.91^9 * 0.01 = **0.00428**.
Perplexity is the inverse probability of the sentence, normalized by the number of digits (N = 10):

- PP = (0.00428)^(-1/10)
- PP = 1 / (0.00428)^(0.1)
- **PP = 1.73**

I want chinese food

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray.

Here are a few other useful probabilities:

$P(\texttt{i}\,|\,\texttt{<s>}) = 0.25$ $\quad$ $P(\texttt{english}\,|\,\texttt{want}) = 0.0011$

$P(\texttt{food}\,|\,\texttt{english}) = 0.5$ $\quad$ $P(\texttt{</s>}\,|\,\texttt{food}) = 0.68$

Intuition of perplexity

- The highest perplexity is the size of the vocabulary. This means that you had basically no information about which words were more likely to occur
- A perplexity of 3 = you had roughly a ⅓ chance of guessing each word correctly.
- The larger the perplexity, the less likely you can guess which word will occur

Perplexity

Metric Level	Interpretation	Performance
High Perplexity	Model is very confused; many equally likely words.	Poor / Random
Low Perplexity	Model is confident; predicts the correct word often.	Good / Accurate
Perplexity = 1	Model predicts the next word with 100% certainty every time.	Perfect (or Overfit)

Lower perplexity = better model

- Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Limitation to perplexity

- Does not necessarily correlate with human perception of quality (e.g., “the cat sat on the mat, the cat sat on the mat”
- Doesn’t correlate with practical, applied performance
- Doesn’t take into account context / long-distance dependencies
- Generic, redundant responses; short sentences
- If interested, check the relation to cross-entropy

Dealing with Sparsity

- Smoothing
- Back-off
- Interpolation

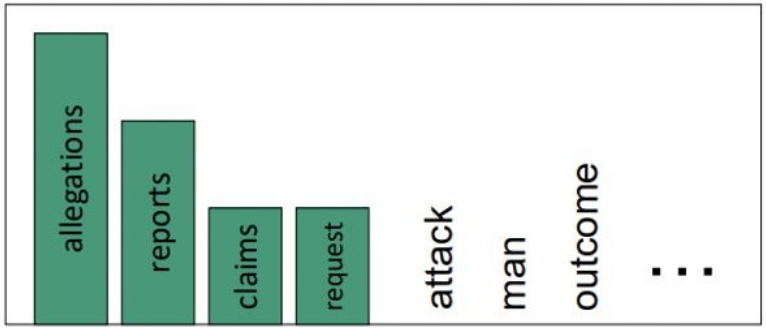
Dealing with Sparsity

- Smoothing
- Back-off
- Interpolation

The intuition of smoothing (from Dan Klein)

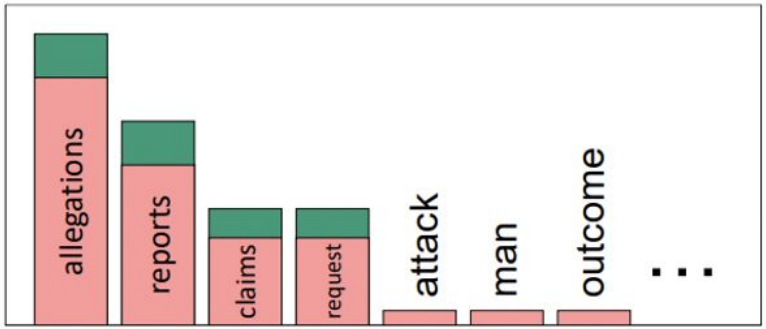
- When we have sparse statistics:

  P(w | denied the)
  3 allegations
  2 reports
  1 claims
  1 request
  7 total



- Steal probability mass to generalize better

  P(w | denied the)
  2.5 allegations
  1.5 reports
  0.5 claims
  0.5 request
  2 other
  7 total



Add-one estimation

- Also called Laplace smoothing
- Pretend we saw each word one more time than we did
- Just add one to all the counts!

- MLE estimate:

$$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

- Add-1 estimate:

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)+1}{c(w_{i-1})+V}$$

What is +1 smoothing -Laplace

The "Zero" Problem

- In standard MLE, if an N-gram never appears in the training data, its count is **0**.
- => probability becomes **0** (multiplying by 0 makes the entire sentence probability 0, and dividing by 0 causes errors).

2. The +1 Solution (Laplace)

- We pretend we have seen every possible word combination at least **one extra time**.
- We add **1** to the numerator (the count of the specific N-gram).
- To keep the probabilities summing to 1, we must add the entire Vocabulary Size ($|V|$) to the denominator.

3. The Formula (Bigram Example)

- **Standard MLE:** $P(w_i \mid w_{i-1})$ = Count(w_{i-1}, w_i) / Count(w_{i-1})
- **Smoothed:** $P(w_i \mid w_{i-1})$ **= (Count(w_{i-1}, w_i) + 1) / (Count(w_{i-1}) + $|V|$)**
    - *Where $|V|$ is the total number of unique words in your vocabulary.*

When is +1 smoothing used?

Whenever we rely on discrete counts of data and need to account for unseen events.

N-Gram Language Models

- **Use Case:** Preventing a language model from assigning a probability of 0 to a perfectly valid sentence just because a specific 2-word combination was missing from the training text.

Naive Bayes Classifiers (Spam Filters)

- **Use Case:** Imagine a spam filter trained on thousands of emails. A new spam email arrives containing the word *"cryptocurrency"*, which the model has never seen before.
- **The Fix:** Without smoothing, the probability of the email being spam becomes 0. With +1 smoothing, *"cryptocurrency"* gets a small, non-zero probability, allowing the model to look at the rest of the words to make a decision.

Information Retrieval / Basic search engines

- **Use Case:** Query Likelihood Models. If a user searches for *"blue canvas shoes"* and the document is missing the word *"canvas"*, smoothing ensures the document isn't completely disqualified from the search results.

Why is it *noisy* for search

While it works, +1 smoothing has a major flaw in search engines: **Document Length Bias**.

- In a search engine, you are comparing a 10-word snippet to a 5,000-word article.
- If you add +1 to every word in a tiny document, you are fundamentally changing its profile much more than you are for a large document.
  - "Simple" search engines accidentally ranking short, irrelevant documents higher just because the smoothing math "inflated" their scores.

Beyond Laplace: Why Dirichlet Smoothing?

The Motivation: While +1 (Laplace) smoothing treats all unseen words as equally likely, Dirichlet smoothing incorporates **prior knowledge** from the entire document collection (corpus).
The Bayesian Perspective: It treats the document as a multinomial distribution and uses the **Dirichlet distribution** as a "conjugate prior."

- Think of it as adding "pseudo-counts" to a document based on how common a word is across all documents.

$$\hat{P}(w|D) = \frac{count(w, D) + \mu \cdot P(w|C)}{|D| + \mu}$$

count(w, D): Times word w appears in document D.
μ: The smoothing parameter (typically the average document length, e.g., 2000).
P(w|C): Probability of word w in the entire **Corpus** (Total counts in C / Size of C).
|D|: Total words in document D.

Beyond Laplace: Why Dirichlet Smoothing?

The Motivation: While +1 (Laplace) smoothing treats all unseen words as equally likely, Dirichlet smoothing incorporates **prior knowledge** from the entire document collection (corpus).

- Adding "pseudo-counts" to a document based on how common a word is across all documents.

$$\hat{P}(w|D) = \frac{count(w, D) + \mu \cdot P(w|C)}{|D| + \mu}$$

count(w, D): Times word w appears in document D.
μ: The smoothing parameter (typically the average document length, e.g., 2000).
P(w|C): Probability of word w in the entire **Corpus** (Total counts in C / Size of C).
|D|: Total words in document D.

Dealing with Sparsity

- Smoothing
- Back-off
- Interpolation

Back-off

- If a model encounters an N-gram it has never seen before (count is 0), it doesn't just guess blindly or assign a 0 probability.

- Instead, it "backs off" to a shorter history. If the Trigram fails, try the Bigram. If the Bigram fails, try the Unigram.

Back-off

we want to predict the next word in the phrase: *"I want Chinese ____"*

- **Target:** P(food | want, Chinese)
- **Step 1 (Check Trigram):** Does the exact phrase *"want Chinese food"* exist in the corpus?
  - If **Yes** ➔ Use the Trigram probability.
  - If **No** (Count = 0) ➔ Proceed to Step 2.
- **Step 2 (Backoff to Bigram):** Drop the oldest word ("want"). Does the phrase *"Chinese food"* exist?
  - If **Yes** ➔ Use the Bigram probability: P(food | Chinese) × α
  - If **No** ➔ Proceed to Step 3.
- **Step 3 (Backoff to Unigram):** Drop the next oldest word ("Chinese").
  - Use the Unigram probability: P(food) × α
  - **discount weight (α)**. This is a small penalty applied because we are relying on less specific context.

Dealing with Sparsity

- Smoothing
- Back-off
- Interpolation

Simple (linear) interpolation

- Instead of choosing *between* a Trigram, Bigram, or Unigram, we calculate the probability for all three and blend them together.
- We assign a fixed weight (**λ**, "Lambda") to each model to decide how much we trust it.

To find the **predicted probability (P)** of word w_n given the two previous words:

$$P(w_n \mid w_{n-2}, w_{n-1}) = \lambda_1 P(w_n \mid w_{n-2}, w_{n-1}) + \lambda_2 P(w_n \mid w_{n-1}) + \lambda_3 P(w_n)$$

The Rules

- The weights must always add up to exactly 1: $\lambda_1 + \lambda_2 + \lambda_3 = 1$
- **Example Weights:** We usually trust the Trigram the most, so we might set $\lambda_1 = 0.7$, $\lambda_2 = 0.2$, and $\lambda_3 = 0.1$.
- **The Benefit:** Even if the Trigram count is 0, the Bigram and Unigram will usually provide a small, non-zero probability, completely solving the sparsity/zero-frequency problem.

Conditional interpolation (context dependent)

The Problem with "Simple" Interpolation

- In simple interpolation, the weights (λ) are static across the entire dataset.
- But what if the context is extremely rare? If we have only seen the phrase *"squid ink ___"* once in our entire corpus, we probably shouldn't trust the Trigram with a massive 0.7 weight. We should rely more heavily on the lower-order models.

The Concept

- We make the weights **conditional** based on the specific history (context) we are looking at.
- If the history (w_{-2}, w_{-1}) has a high frequency in the training data, we give λ_1 a high value.
- If the history has a low frequency, we give λ_2 or λ_3 higher values.
- Lambdas are now functions of the specific context history

$$P(w \mid w_{-2}, w_{-1}) = \lambda_1(w_{-2}, w_{-1})P(w \mid w_{-2}, w_{-1}) + \lambda_2(w_{-2}, w_{-1})P(w \mid w_{-1}) + \lambda_3(w_{-2}, w_{-1})P(w)$$

How do we calculate lambda

- Divide data into training, dev, and test set
- Estimate probabilities of unigrams, bigrams, and trigrams on training data
- Use dev set to pick lambdas that maximize probability of dev set sentences

Google Book N-Grams corpus

What is it?

- A massive dataset released by Google containing the frequency counts of N-grams (from 1-grams up to 5-grams) found in millions of digitized books printed between 1500 and the present.
- It is one of the largest linguistic datasets ever created, containing hundreds of billions of words.

Because the data is tagged by **year**, researchers do not just use it to train language models. They use it to study human history and culture through the lens of language.
You can literally watch the birth, peak, and death of ideas, technologies, and slang.

https://books.google.com/ngrams/

Google N-Gram Release, August 2006

AUG
3

All Our N-gram are Belong to You

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects,

...

That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

Google N-Gram Release

- serve as the incoming 92
- serve as the incubator 99
- serve as the independent 794
- serve as the index 223
- serve as the indication 72
- serve as the indicator 120
- serve as the indicators 45
- serve as the indispensable 111
- serve as the indispensible 40
- serve as the individual 234

https://blog.research.google/2006/08/all-our-n-gram-are-belong-to-you.html

Google Book N-Grams corpus

N-gram Tracking

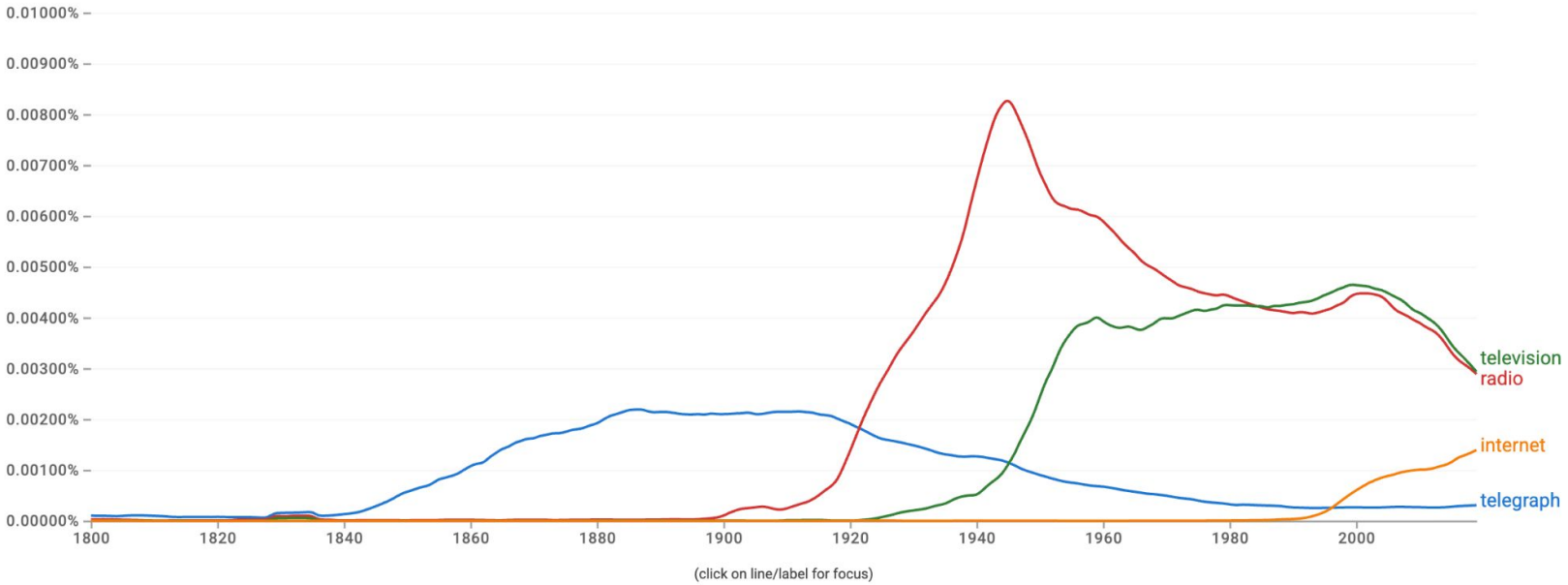
- **Technology:** Graphing the 1-grams *"telegraph"*, *"radio"*, *"television"*, and *"internet"* shows distinct, overlapping bell curves corresponding to their historical dominance.
- **Grammar Evolution:** Tracking the Bigrams *"burnt out"* vs. *"burned out"* shows exactly when the regular "-ed" spelling overtook the irregular "-t" spelling in English literature.
- **Censorship:** Researchers have used the corpus to detect sudden, unnatural drops in the frequency of specific names (e.g., political dissidents) in countries with state-controlled publishing

https://books.google.com/ngrams/

Google Books Ngram Viewer

telegraph,radio,television,internet

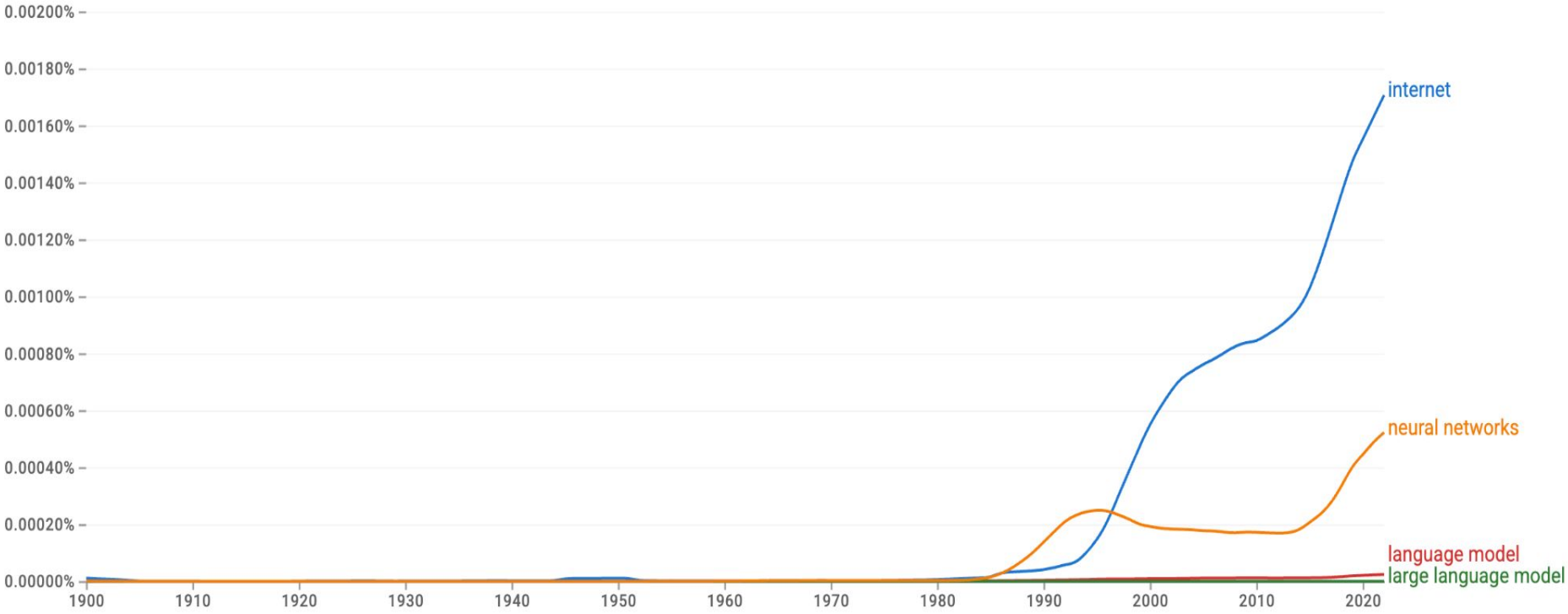
1800 - 2019 | English (2019) | Case-Insensitive | Smoothing



Google Books Ngram Viewer

internet,language model,large language model,neural networks

1900 - 2022 | English | Case-Insensitive | Smoothing

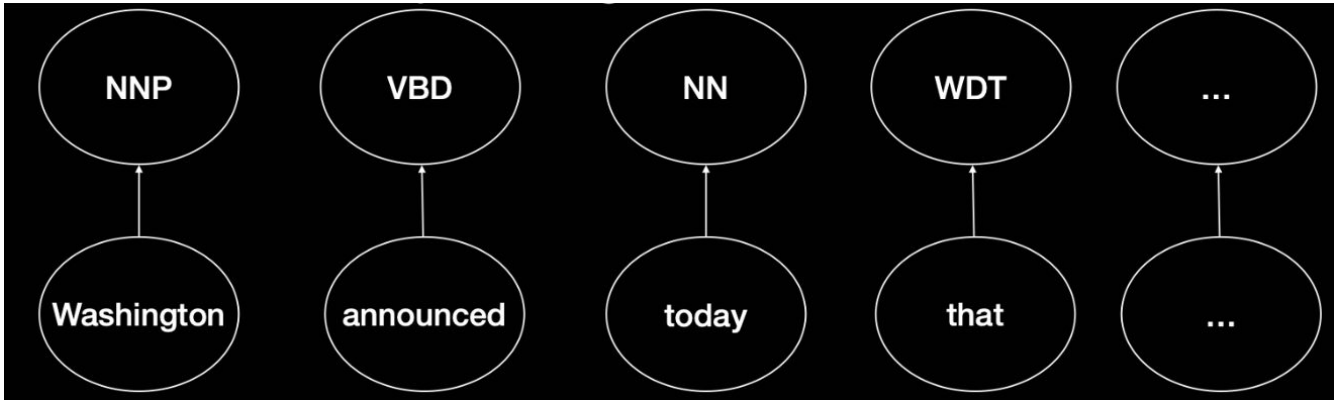


Summary

- Language models offer a way to calculate the probability of a string of words and to predict upcoming words
- N-Gram models are a type of language model that predict upcoming words based on a fixed window of previous words
- Simple approach that often works decently well, but clearly missing a lot of linguistic information and “understanding”
- Extrinsic vs. intrinsic (perplexity) evaluation
- Dealing with sparsity

Part-of-Speech (POS) Tagging

Goal: Take a text and assign a part-of-speech (noun, verb, adjective…) to each word in the text

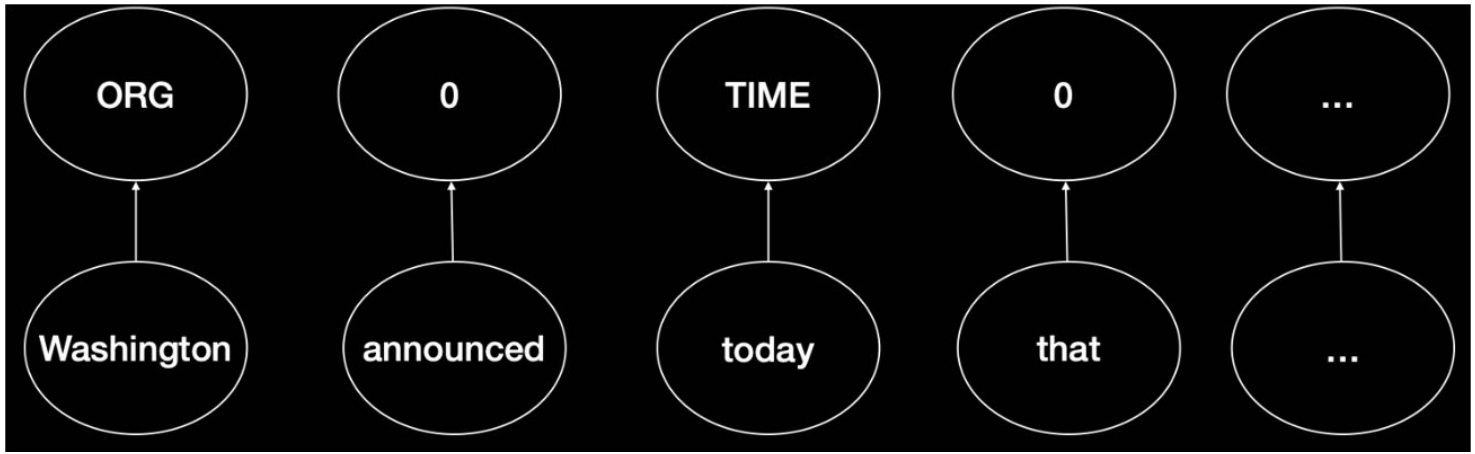


Why?

- Useful for determining sentence structure and meaning (i.e., for higher-level processing)
- Knowing whether word is noun/verb/adjective tells us about likely neighboring words
- Linguistics: key piece of describing how a language works
- NLP: often useful as a feature (language modeling and QA can get derailed if they don't understand POS)
- Data science: can help with analysis, filtering, processing (e.g., maybe you only care about pulling out the names or verbs from a piece of text)

One instance of a sequence labeling task

Given an input of length N, you want to assign it label of length N



How can we define parts of speech?

To assign a Part of Speech, we don't just look at what a word *means*; we look at its **shape** and its **neighborhood**.

The Semantic Definition (Meaning)

- **The Concept:** Defining a word by the concept it represents.
- **Examples:** Nouns are "people, places, or things." Verbs are "actions." Adjectives are "properties."
- **The NLP Problem:** This is often too fuzzy for computers. For example, "assassination" is an action, but grammatically it acts as a noun.

How can we define parts of speech?

To assign a Part of Speech, we don't just look at what a word *means*; we look at its **shape** and its **neighborhood**.

The Morphological Definition (Shape/Form)

- **The Concept:** Defining a word by its internal structure, specifically its prefixes and suffixes (morphemes).
- If a word ends in **-tion**, **-ness**, or **-ity**, it is almost certainly a Noun.
    - If it ends in **-ed** or **-ing**, it is usually a Verb.
    - If it ends in **-ly**, it is likely an Adverb.
- **The NLP Value:** This is highly useful for guessing the POS of unknown or out-of-vocabulary (OOV) words.

How can we define parts of speech?

The Distributional/Syntactic Definition (Context)

- **The Concept:** Defining a word by where it physically sits in a sentence and what words are allowed to go next to it.
- **Examples:**
  - If a word comes immediately after "the", "a", or "an" (Determiners), it is usually a Noun or an Adjective.
  - *The* ____ *is running.* (Only a noun fits here: *dog*, *program*, *boy*).
- **The NLP Value:** This is the **most powerful** definition for algorithms. Models like Hidden Markov Models (HMMs) rely almost entirely on this distributional context to tag words accurately.

How can we define parts of speech?

Typically combination of syntactic and morphology

The Open Class Problem (Handling the Unknown)

1. Closed Class Words

- **What they are:** Grammatical glue words like Prepositions (*in, on*), Determiners (*the, a*), and Conjunctions (*and, or*).
- **Why they are "Closed":** Languages rarely invent new ones. You can literally write down a complete list of every English preposition and give it to a computer to memorize.

2. Open Class Words (The Moving Target)

- **What they are:** Nouns, Verbs, Adjectives, and Adverbs. These carry the actual meaning of the sentence.
- **Why they are "Open":** We are constantly inventing new ones! Think of words like *"doomscrolling"*, *"Googling"*, *"yeet"*, or *"Bitcoin"*.

The Open Class Problem (Handling the Unknown)

The Problem: Out-Of-Vocabulary (OOV) Words

- The "Open Class Problem" occurs because **no matter how large your training dictionary is, your tagger will eventually encounter a word it has never seen before.**
- If an NLP model relies entirely on memorizing a dictionary (e.g., "run" = Verb), it will crash or fail when it sees a new slang word or a typo.
- Because the word is unknown, the model must deduce that it belongs to an open class (usually a Noun or Verb) by relying strictly on **Morphology** (does it end in *-ed*?) and **Context** (does it come right after a determiner?).

Open class POS

Four major open classes in English

- Nouns
- Verbs
- Adjectives
- Adverbs

All languages have noun and verbs… but may not have the other two

Nouns

Nouns

- Open class
  - New inventions all the time: muggle, webinar, ...

Nouns

- Open class
  - New inventions all the time: muggle, webinar, ...
- Semantics:
  - Generally, words for people, places, things
  - But not always (bandwidth, energy, ...)

Nouns

- Open class
  - New inventions all the time: muggle, webinar, ...
- Semantics:
  - Generally, words for people, places, things
  - But not always (bandwidth, energy, ...)
- Syntactic environment:
  - Occurring with determiners
  - Pluralizable, possessivizable
- Other characteristics:
  - Mass vs. count nouns

Verbs

- Open class
  - New inventions all the time: google, tweet, …

Verbs

- Open class
  - New inventions all the time: google, tweet, …
- Semantics
  - Generally, denote actions, processes, etc.

Verbs

- Open class
  - New inventions all the time: google, tweet, …
- Semantics
  - Generally, denote actions, processes, etc.
- Syntactic environment
  - E.g., Intransitive, transitive

Verbs

- Open class
  - New inventions all the time: google, tweet, ...
- Semantics
  - Generally, denote actions, processes, etc.
- Syntactic environment
  - E.g., Intransitive, transitive
- Other characteristics
  - Main vs. auxiliary verbs
  - Gerunds (verbs behaving like nouns)
  - Participles (verbs behaving like adjectives)

Adjectives and Adverbs

- Adjectives
  - Generally modify nouns, e.g., *tall* girl

Adjectives and Adverbs

- Adjectives
  - Generally modify nouns, e.g., *tall* girl
- Adverbs
  - A semantic and formal hodge-podge…
  - Sometimes modify verbs, e.g., sang *beautifully*
  - Sometimes modify adjectives, e.g., *extremely* hot

Closed Class POS

- Prepositions
  - In English, occurring before noun phrases
  - Specifying some type of relation (spatial, temporal, …)
  - Examples: *on* the shelf, *before* noon

Closed Class POS

- Prepositions
  - In English, occurring before noun phrases
  - Specifying some type of relation (spatial, temporal, …)
  - Examples: *on* the shelf, *before* noon
- Particles
  - Resembles a preposition, but used with a verb (“phrasal verbs”)
  - Examples: find *out*, turn *over*, go *on*

Particles vs. Prepositions

He came *by* the office in a hurry	(by = preposition)
He came *by* his fortune honestly	(by = particle)
We ran *up* the phone bill	(up = particle)
We ran *up* the small hill	(up = preposition)
He lived *down* the block	(down = preposition)
He never lived *down* the nicknames	(down = particle)

More Closed Class POS

- Determiners
  - Establish reference for a noun
  - Examples: *a*, *an*, *the* (articles), *that*, *this*, *many*, *such*, …

More Closed Class POS

- Determiners
  - Establish reference for a noun
  - Examples: *a*, *an*, *the* (articles), *that*, *this*, *many*, *such*, …
- Pronouns
  - Refer to person or entities: *he*, *she*, *it*
  - Possessive pronouns: *his*, *her*, *its*
  - Wh-pronouns: *what*, *who*

Closed class POS

- Coordinating conjunctions
  - Join two elements of “equal status”
  - Examples: cats *and* dogs, salad *or* soup

Closed class POS

- Coordinating conjunctions
  - Join two elements of “equal status”
  - Examples: cats *and* dogs, salad *or* soup
- Subordinating conjunctions
  - Join two elements of “unequal status”
  - Examples: We’ll leave *after* you finish eating. *While* I was waiting in line, I saw my friend.
  - Complementizers are a special case: I think *that* you should finish your assignment

Beyond English

Chinese

No verb/adjective distinction!

漂亮: beautiful/to be beautiful

Riau Indonesian/Malay

No Articles
No Tense Marking
3rd person pronouns neutral to both gender and number
No features distinguishing verbs from nouns

Ayam (chicken) *Makan* (eat)

The chicken is eating
The chicken ate
The chicken will eat
The chicken is being eaten
Where the chicken is eating
How the chicken is eating
Somebody is eating the chicken
The chicken that is eating

Treebanks

- Corpora that have been annotated for POS tags and syntactic structure
- Usually parsed automatically and hand-corrected
- **Penn Treebank:**
  - 7 million words with POS tags
  - ~2-3 millions words with parses
- Different tagsets: balance informativity with how time-intensive

How much deep grammatical information does the tag capture? Does it just say "Verb," or does it say "Verb, past-tense, plural, third-person"?

Machine learning models require humans to manually label thousands of sentences to create training data.

Part of Speech Tags

Penn Treebank Tags

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coord. conj.	*and, but, or*	NNP	proper noun, sing.	*IBM*	TO	"to"	*to*
CD	cardinal number	*one, two*	NNPS	proper noun, plu.	*Carolinas*	UH	interjection	*ah, oops*
DT	determiner	*a, the*	NNS	noun, plural	*llamas*	VB	verb base	*eat*
EX	existential 'there'	*there*	PDT	predeterminer	*all, both*	VBD	verb past tense	*ate*
FW	foreign word	*mea culpa*	POS	possessive ending	*'s*	VBG	verb gerund	*eating*
IN	preposition/ subordin-conj	*of, in, by*	PRP	personal pronoun	*I, you, he*	VBN	verb past participle	*eaten*
JJ	adjective	*yellow*	PRP$	possess. pronoun	*your, one's*	VBP	verb non-3sg-pr	*eat*
JJR	comparative adj	*bigger*	RB	adverb	*quickly*	VBZ	verb 3sg pres	*eats*
JJS	superlative adj	*wildest*	RBR	comparative adv	*faster*	WDT	wh-determ.	*which, that*
LS	list item marker	*1, 2, One*	RBS	superlatv. adv	*fastest*	WP	wh-pronoun	*what, who*
MD	modal	*can, should*	RP	particle	*up, off*	WP$	wh-possess.	*whose*
NN	sing or mass noun	*llama*	SYM	symbol	+,%, &	WRB	wh-adverb	*how, where*

Figure 8.2 Penn Treebank part-of-speech tags.

Token	spaCy Tag (Penn Treebank)	What it means in this context
Will	**MD**	**Modal auxiliary:** A helper verb expressing possibility or future intent.
it	**PRP**	**Personal Pronoun:** Refers to the object in question (e.g., a satellite).
just	**RB**	**Adverb:** Modifies the verb "cease".
cease	**VB**	**Verb, base form:** The primary action (to stop).
to	**TO**	**Infinitive marker:** The word "to" preceding a base verb.
function	**VB**	**Verb, base form:** The action being ceased.
and	**CC**	**Coordinating Conjunction:** Connects "function" and "orbit".
orbit	**VB**	**Verb, base form:** The second action connected by the conjunction.
the	**DT**	**Determiner:** Specifies the upcoming noun.
earth	**NN**	**Noun, singular:** The object being orbited.
forever	**RB**	**Adverb:** Describes how long the orbiting will happen.

Universal dependencies tagset

The Problem with the Penn Treebank The Penn Treebank is fantastic, but it was built specifically for English. If you try to use its 36 tags on heavily inflected languages like Arabic, Finnish, or Japanese, it completely breaks down.

2. The Solution: One Tagset for Every Language The Universal Dependencies project created a simplified, standardized set of exactly **17 tags** designed to work across *all* human languages.

3. The 17 Universal Tags (Categorized) Instead of getting bogged down in tense or plurality, UPOS keeps it broad and structural:

- **Open Class Words:** `NOUN`, `VERB`, `ADJ` (Adjective), `ADV` (Adverb), `PROPN` (Proper Noun), `INTJ` (Interjection)
- **Closed Class Words:** `ADP` (Adposition/Preposition), `AUX` (Auxiliary Verb), `DET` (Determiner), `NUM` (Numeral), `PART` (Particle), `PRON` (Pronoun)
- **Other:** `PUNCT` (Punctuation), `SYM` (Symbol), `X` (Other/Foreign/Unknown)

Universal dependencies tagset

The Biggest Benefit

Cross-Lingual NLP Because the tags are universal, an NLP engineer can write a single piece of code that says *"extract all the* `PROPN` *(Proper Nouns)"* and it will successfully pull the names of people and cities out of texts in 100 different languages without needing to learn 100 different tagging systems.

	Tag	Description	Example
Open Class	**ADJ**	Adjective: noun modifiers describing properties	*red, young, awesome*
	ADV	Adverb: verb modifiers of time, place, manner	*very, slowly, home, yesterday*
	NOUN	words for persons, places, things, etc.	*algorithm, cat, mango, beauty*
	VERB	words for actions and processes	*draw, provide, go*
	PROPN	Proper noun: name of a person, organization, place, etc..	*Regina, IBM, Colorado*
	INTJ	Interjection: exclamation, greeting, yes/no response, etc.	*oh, um, yes, hello*
Closed Class Words	**ADP**	Adposition (Preposition/Postposition): marks a noun's spacial, temporal, or other relation	*in, on, by, under*
	AUX	Auxiliary: helping verb marking tense, aspect, mood, etc.,	*can, may, should, are*
	CCONJ	Coordinating Conjunction: joins two phrases/clauses	*and, or, but*
	DET	Determiner: marks noun phrase properties	*a, an, the, this*
	NUM	Numeral	*one, two, first, second*
	PART	Particle: a preposition-like form used together with a verb	*up, down, on, off, in, out, at, by*
	PRON	Pronoun: a shorthand for referring to an entity or event	*she, who, I, others*
	SCONJ	Subordinating Conjunction: joins a main clause with a subordinate clause such as a sentential complement	*that, which*
Other	**PUNCT**	Punctuation	; , ()
	SYM	Symbols like $ or emoji	$, %
	X	Other	asdf, qwfg

Figure 8.1 The 17 parts of speech in the Universal Dependencies tagset (Nivre et al., 2016a). Features can be added to make finer-grained distinctions (with properties like number, case, definiteness, and so on).

```
import spacy

# Load the English language model
nlp = spacy.load("en_core_web_sm")

text = "Will it just cease to function and orbit the earth forever"
doc = nlp(text)

for token in doc:
    print(f"{token.text} | {token.pos_} | {token.tag_}")
```

```
Will     | AUX   | MD
it       | PRON  | PRP
just     | ADV   | RB
cease    | VERB  | VB
to       | PART  | TO
function | VERB  | VB
and      | CCONJ | CC
orbit    | VERB  | VB
the      | DET   | DT
earth    | NOUN  | NN
forever  | ADV   | RB
```

Differences between tagsets

	There	are	70	children	there	.
Penn Treebank	EX	VBP	CD	NNS	RB	.
Universal Dependency	PRO	VERB	NUM	NOUN	ADV	PUNC

Tag	The Definition	Examples
NN	Noun, singular or mass	*dog, water, city, software*
NNS	Noun, plural	*dogs, oceans, cities, computers*
NNP	Proper noun, singular	*London, Microsoft, Sarah, Earth*
NNPS	Proper noun, plural	*Americans, iPhones, Dakotas*

Try tagging

- The back door
- On my back
- Win the voters back
- Promised to back the bill

- I hope that she wins
- That day was nice
- You can go that far

What is the challenge in POS?

- Ambiguity!
- In English
  - ~11.5% of word types are ambiguous in Brown corpus
  - 40-60% of word tokens are ambiguous
  - Annotator disagreement in Penn Treebank: 3.5%
- But usually predictable

- earnings growth took a **back/JJ** seat
- a small building in the **back/NN**
- a clear majority of senators **back/VBP** the bill
- Dave began to **back/VB** toward the door
- enable the country to buy **back/RP** debt
- I was twenty-one **back/RB** then

POS tagging

- Baseline?
  - Pick most frequent tag for each word type
  - ~90% accuracy if train + test sets are drawn from Penn Treebank
- Upper bound performance:
  - Humans achieve roughly 97% accuracy

Generative Approach
Hidden Markov Models

Markov Chains

- A Markov Chain is a mathematical system that hops from one "state" to another based on probabilities.

- The golden rule of this system is the **Markov Assumption (Memorylessness)**: The probability of moving to the next state depends *only* on your current state. The system has absolutely zero memory of how it got there.

Markov Chains

The Bigram Connection A Bigram language model is literally just a **first-order Markov Chain** applied to human language.

- **The States:** Every word in your vocabulary is a "state."
- **The Transitions:** The probability of moving from one word to the next. For example, if you are currently in the state *"New"*, there is a high transition probability to the state *"York"*, and a low transition probability to the state *"apple"*.

$$P(w_n \mid w_1, w_2, \ldots, w_{n-1}) \approx P(w_n \mid w_{n-1})$$

Markov assumption

Markov Chains

Example: Bigram Language Model

	i	want	to	eat	food	lunch
i	0.01	0.98		0.01		
want			0.98		0.01	0.01
to			0.01	0.98		0.01
eat			0.04		0.04	0.91
food	0.44		0.44		0.12	
lunch	0.67				0.33	

$$P(w_n \mid w_{n-1}) = \frac{\#(w_{n-1}w_n)}{\#w_{n-1}}$$

$$P(w_0 \ldots w_n) \approx P(w_0 \mid \text{<s>}) \times P(w_1 \mid w_0) \times P(w_2 \mid w_1) \times \ldots \times P(w_n \mid w_{n-1})$$

Markov chain: formal definition

A Set of States (Q)

- **Symbol:** Q = {q_1, q_2, ..., q_N}
- **Definition:** A finite set of N possible states in the model.
- **In a Bigram Model:** Each state q_i represents a specific word in your vocabulary.

A Transition Probability Matrix (A)

- **Symbol:** A = [a_{i}□]
- **Definition:** An N × N matrix where each element a_{i}□ represents the probability of moving from state i to state j.
- **The Probability Formula:** a_{i}□ = P(q□ = j | q□$_{-1}$ = i)

Markov chain: formal definition

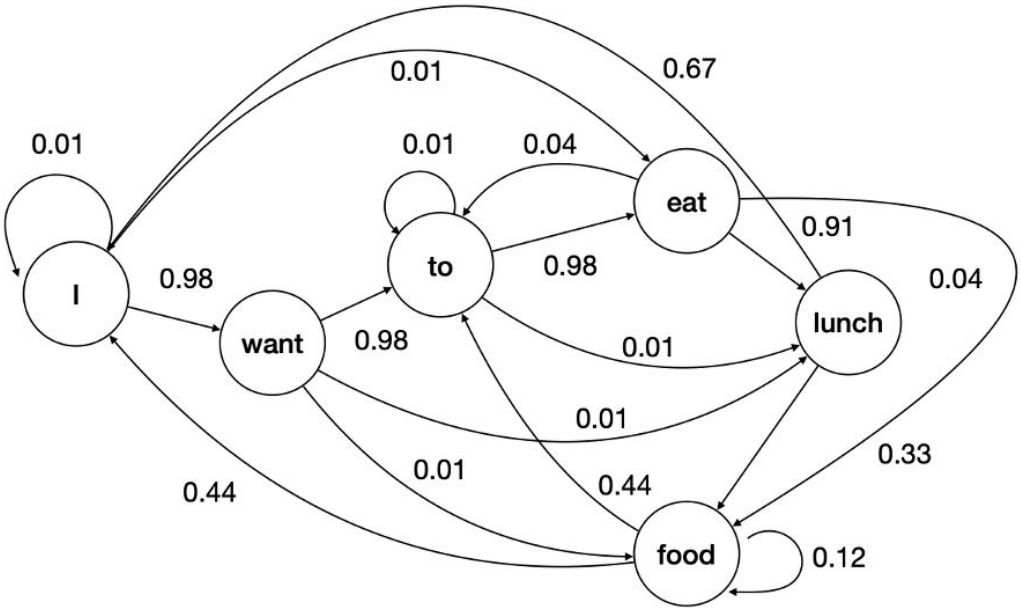
An Initial Probability Distribution (π)

- **Symbol:** $\pi = [\pi_1, \pi_2, \ldots, \pi\_N]$
- **Definition:** The probability that the Markov Chain will start in state i.
- **The Probability Formula:** $\pi_i = P(q_1 = i)$

Markov Chains

Example: Bigram Language Model

	i	want	to	eat	food	lunch
i	0.01	0.98		0.01		
want			0.98		0.01	0.01
to			0.01	0.98		0.01
eat			0.04		0.04	0.91
food	0.44		0.44		0.12	
lunch	0.67				0.33	



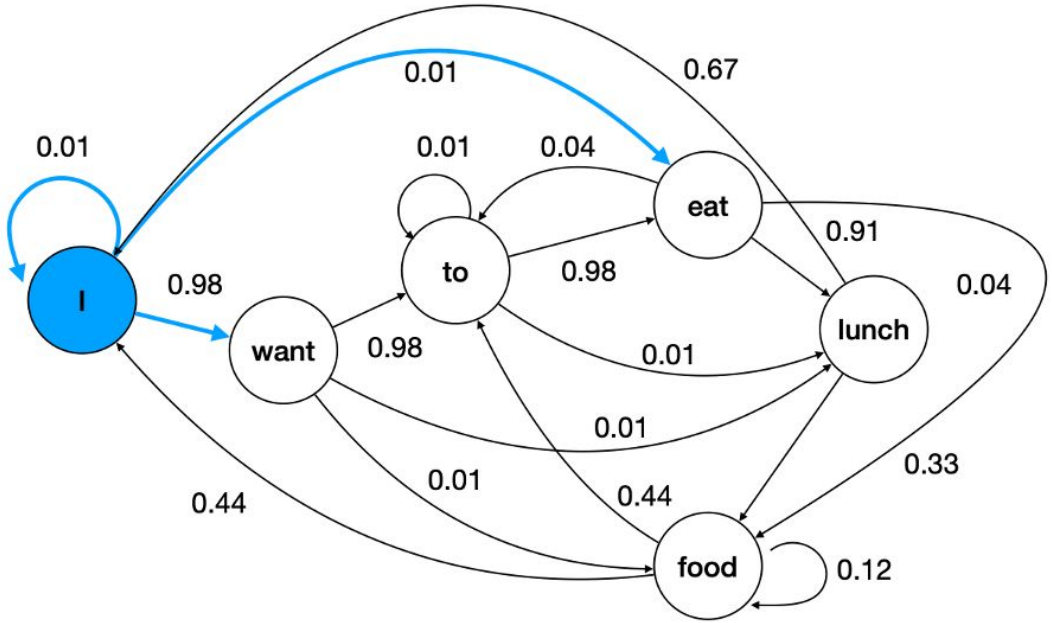
$$P(w_n \mid w_{n-1}) = \frac{\#(w_{n-1}w_n)}{\#w_{n-1}}$$

$$P(w_0 \ldots w_n) \approx P(w_0 \mid \text{<s>}) \times P(w_1 \mid w_0) \times P(w_2 \mid w_1) \times \ldots \times P(w_n \mid w_{n-1})$$

Markov Chains

Example: Bigram Language Model

	i	want	to	eat	food	lunch
i	0.01	0.98		0.01		
want			0.98		0.01	0.01
to			0.01	0.98		0.01
eat			0.04		0.04	0.91
food	0.44		0.44		0.12	
lunch	0.67				0.33	



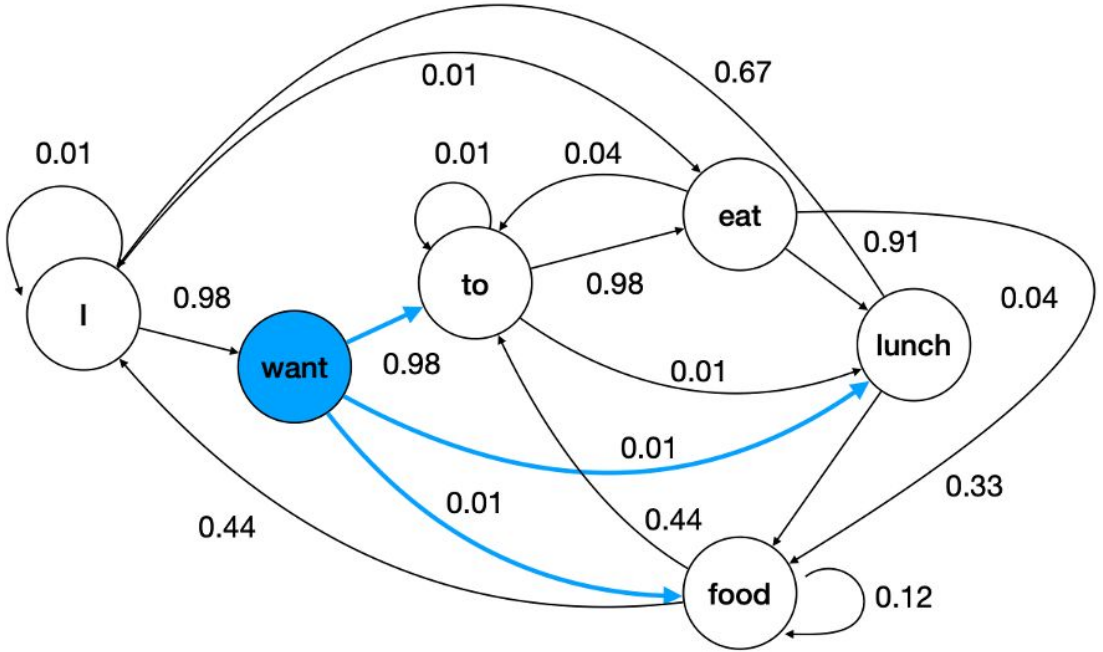
$$P(w_n \mid w_{n-1}) = \frac{\#(w_{n-1}w_n)}{\#w_{n-1}}$$

$$P(w_0 \ldots w_n) \approx P(w_0 \mid \text{<s>}) \times P(w_1 \mid w_0) \times P(w_2 \mid w_1) \times \ldots \times P(w_n \mid w_{n-1})$$

Markov Chains

Example: Bigram Language Model

	i	want	to	eat	food	lunch
i	0.01	0.98		0.01		
want			0.98		0.01	0.01
to			0.01	0.98		0.01
eat			0.04		0.04	0.91
food	0.44		0.44		0.12	
lunch	0.67				0.33	



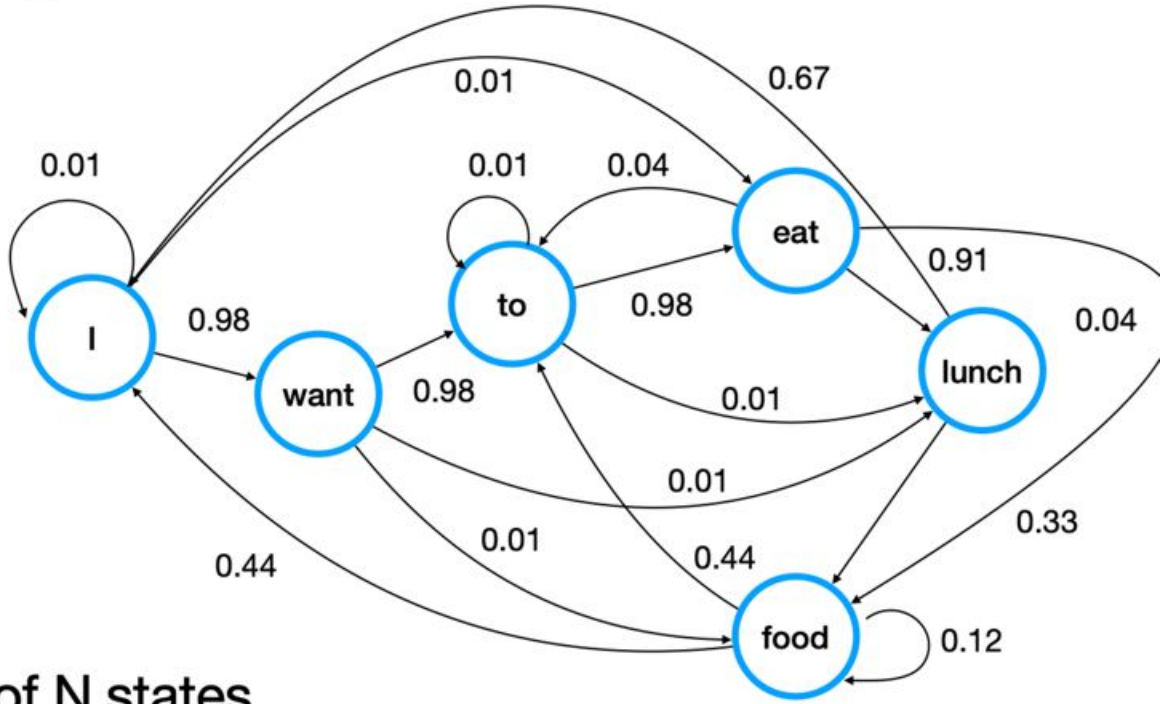
$$P(w_n \mid w_{n-1}) = \frac{\#(w_{n-1}w_n)}{\#w_{n-1}}$$

$$P(w_0 \ldots w_n) \approx P(w_0 \mid \text{<s>}) \times P(w_1 \mid w_0) \times P(w_2 \mid w_1) \times \ldots \times P(w_n \mid w_{n-1})$$

Markov Chains

Example: Bigram Language Model

	i	want	to	eat	food	lunch
i	0.01	0.98		0.01		
want			0.98		0.01	0.01
to			0.01	0.98		0.01
eat			0.04		0.04	0.91
food	0.44		0.44		0.12	
lunch	0.67				0.33	

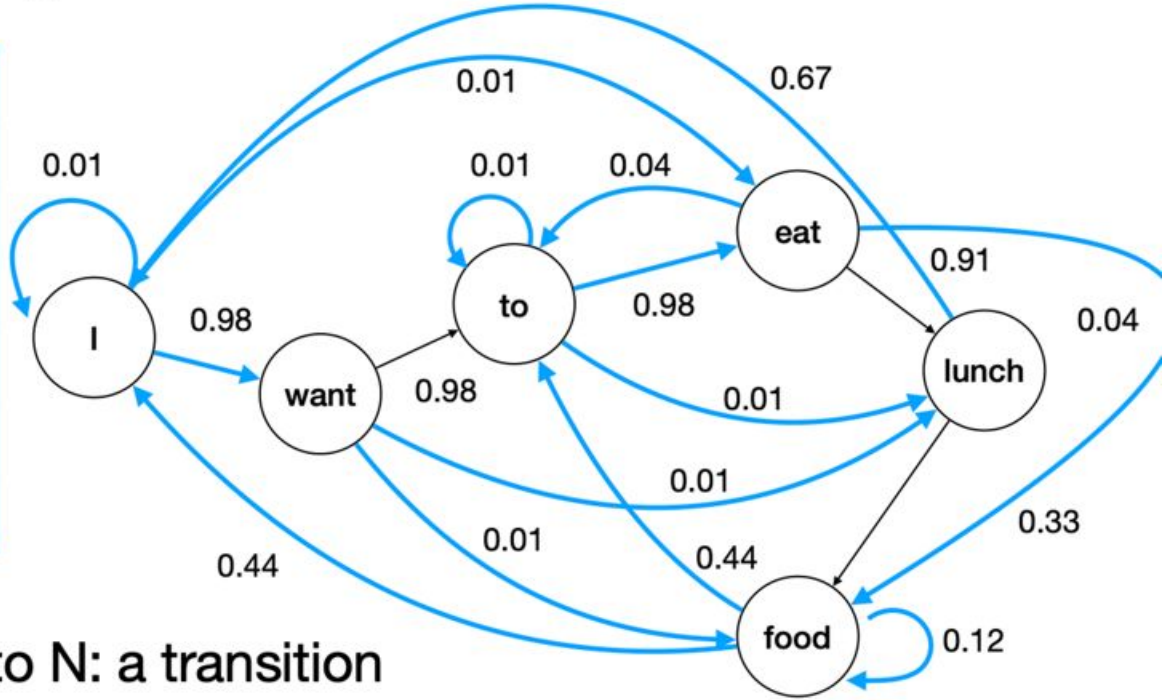


$Q = \{q_1, q_2, \ldots q_N\}$: a set of N states

Markov Chains

Example: Bigram Language Model

	i	want	to	eat	food	lunch
i	0.01	0.98		0.01		
want			0.98		0.01	0.01
to			0.01	0.98		0.01
eat			0.04		0.04	0.91
food	0.44		0.44		0.12	
lunch	0.67				0.33	



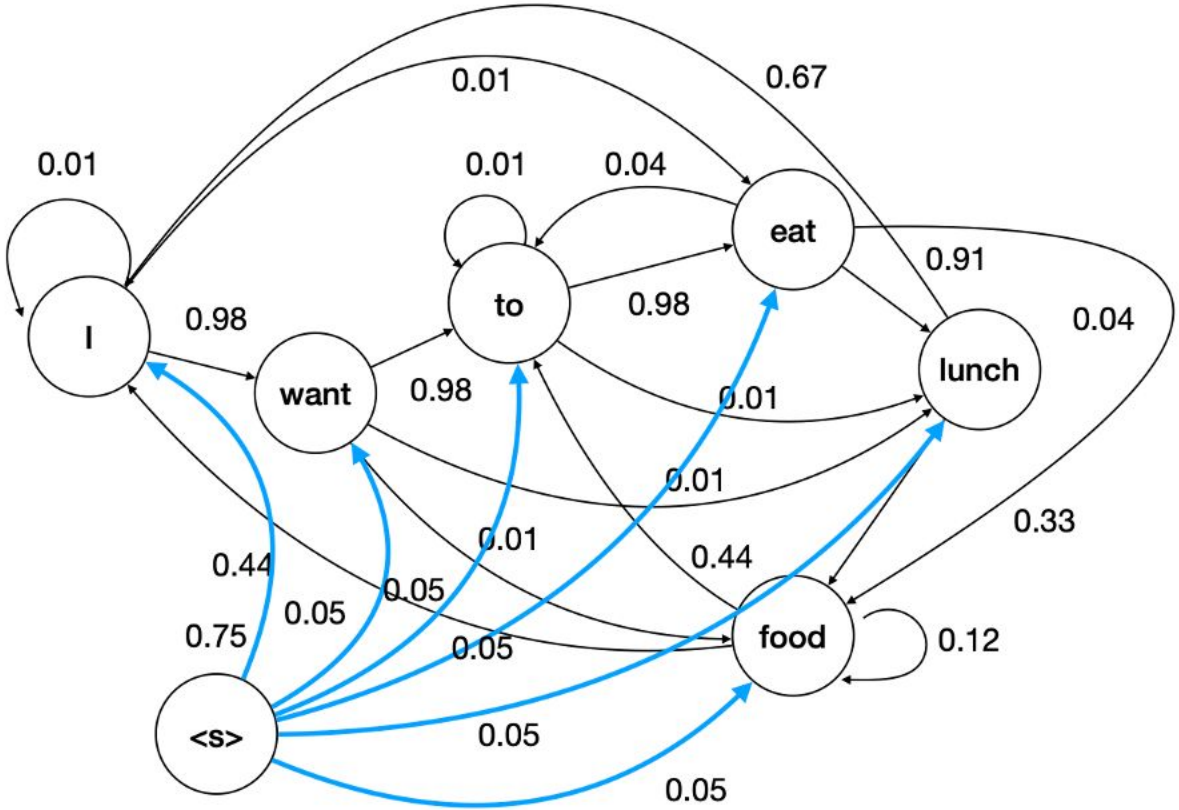
A = {aij} for all i,j from 1 to N: a transition probability matrix

Markov Chains

Example: Bigram Language Model

	i	want	to	eat	food	lunch
i	0.01	0.98		0.01		
want			0.98		0.01	0.01
to			0.01	0.98		0.01
eat			0.04		0.04	0.91
food	0.44		0.44		0.12	
lunch	0.67				0.33	

	i	want	to	eat	food	lunch
π	0.75	0.05	0.05	0.05	0.05	0.05



π = {π1, π2, …πN}: an initial probability distribution over states

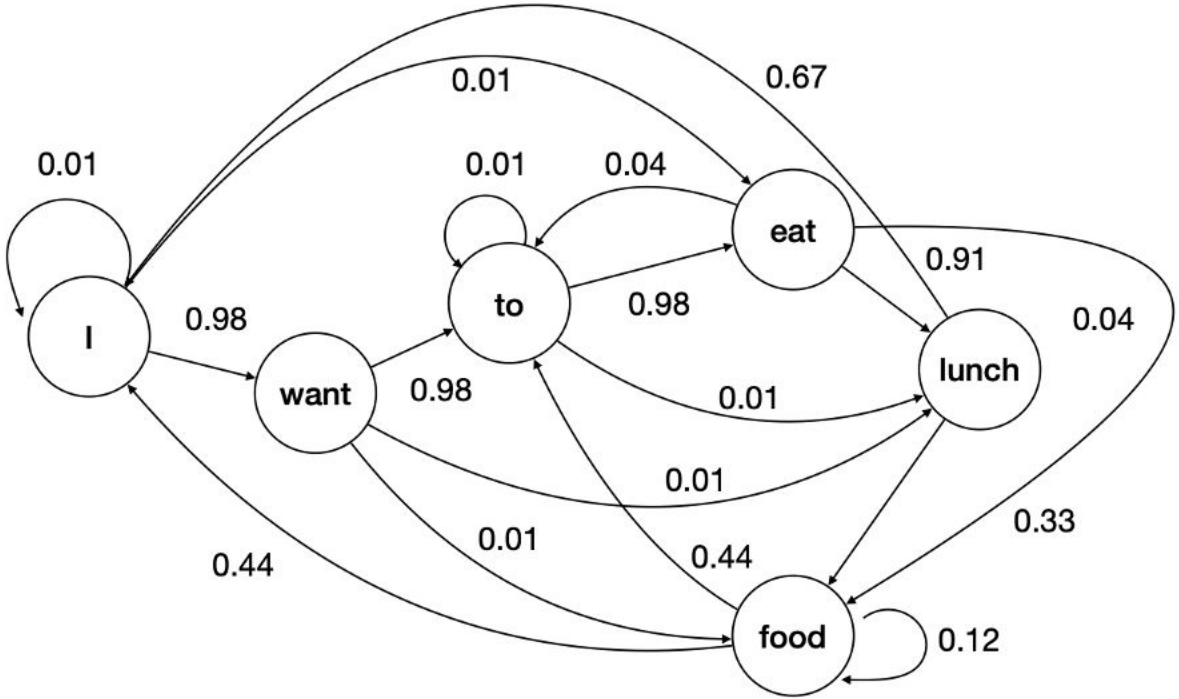
Markov Chains

Example: Bigram Language Model

	i	want	to	eat	food	lunch
π	0.75	0.05	0.05	0.05	0.05	0.05

	i	want	to	eat	food	lunch
i	0.01	0.98		0.01		
want			0.98		0.01	0.01
to			0.01	0.98		0.01
eat			0.04		0.04	0.91
food	0.44		0.44		0.12	
lunch	0.67				0.33	

P(I want food to eat)?

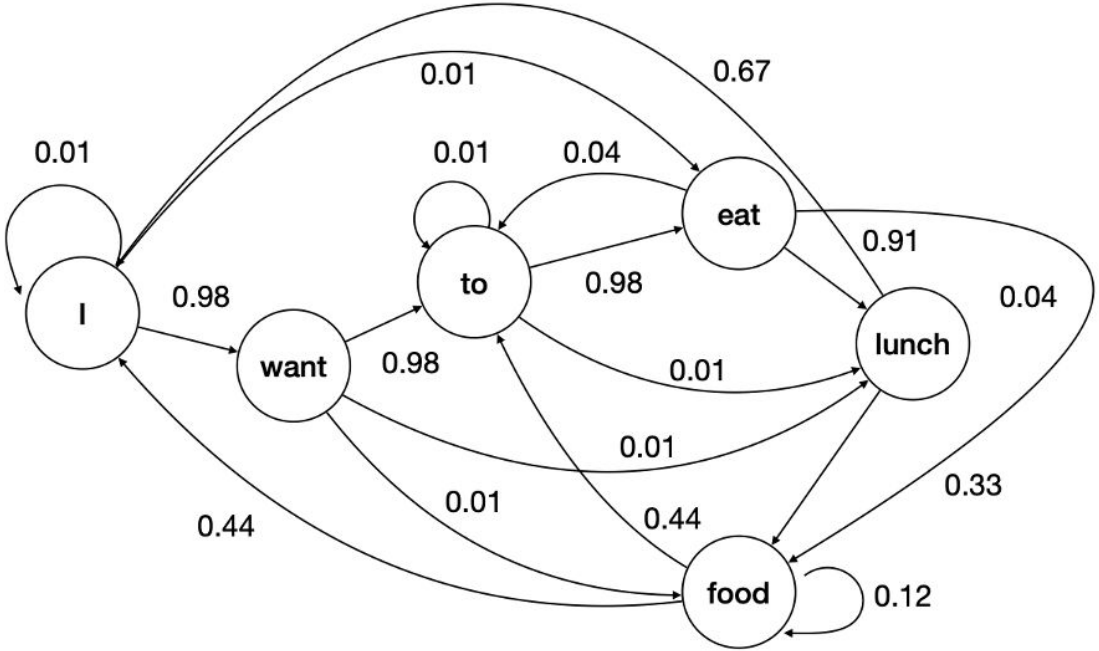


Markov Chains

Example: Bigram Language Model

	i	want	to	eat	food	lunch
π	0.75	0.05	0.05	0.05	0.05	0.05

	i	want	to	eat	food	lunch
i	0.01	0.98		0.01		
want			0.98		0.01	0.01
to			0.01	0.98		0.01
eat			0.04		0.04	0.91
food	0.44		0.44		0.12	
lunch	0.67				0.33	



P(I want food to eat)?

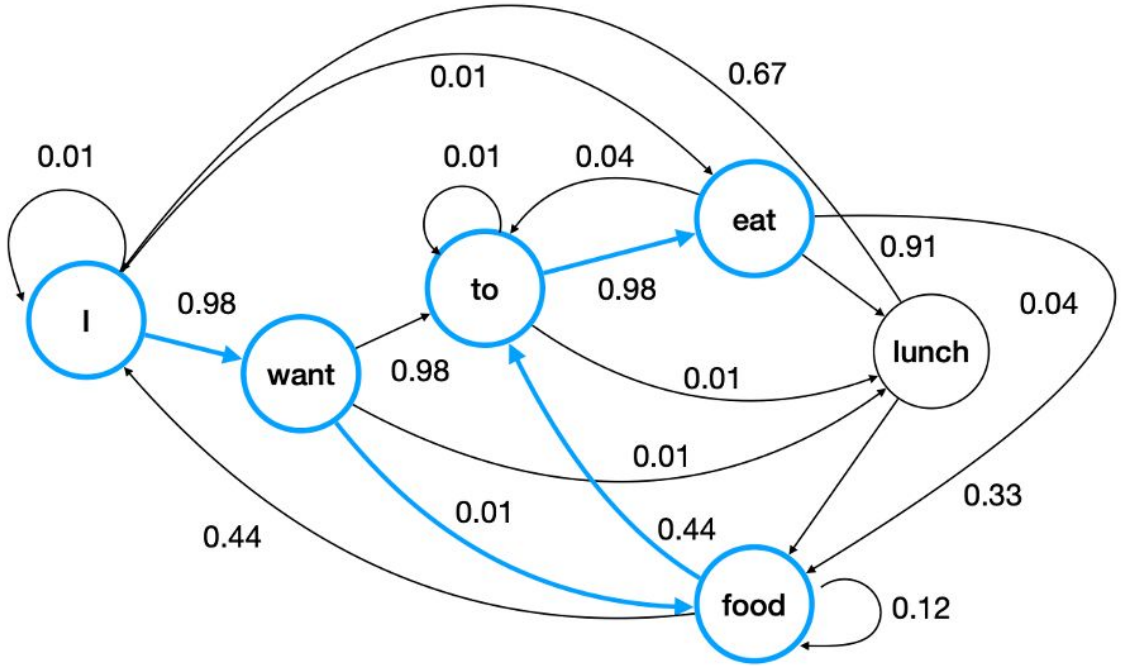
π(I) x P(want|I) x P(food|want) x P(to|food) x P(eat|to) x P(</s>|eat)

Markov Chains

Example: Bigram Language Model

	i	want	to	eat	food	lunch
π	0.75	0.05	0.05	0.05	0.05	0.05

	i	want	to	eat	food	lunch
i	0.01	0.98		0.01		
want			0.98		0.01	0.01
to			0.01	0.98		0.01
eat			0.04		0.04	0.91
food	0.44		0.44		0.12	
lunch	0.67				0.33	



P(I want food to eat)?

0.75 x 0.98 x 0.01 x 0.44 x 0.98 x P(</s>|eat)

Hidden Markov Models: Augmenting Markov Chains

What makes it "Hidden"?

In a standard Markov Chain (like a Bigram model), the "states" are the actual words in the sentence. You can see them perfectly. In HMM, we split the world into two layers: what we *can* see, and what we *cannot* see.

1. The Visible Layer (Observations)

- The sequence of words w_1, w_2, …, w_n (e.g., *"The dog ran"*).

2. The Hidden Layer (States)

- The underlying grammatical structure (e.g., DET → NOUN → VERB).
- The actual states hopping from one to the next are the POS tags. Because the text doesn't explicitly tell us what they are, they are **Hidden States**.

HMM: The Two Core Probabilities

HMM relies on two completely different sets of probabilities working together simultaneously.

1. Transition Probabilities (The Grammatical Rules)

- **What it does:** Measures the probability of jumping from one hidden tag to another.
- **The Question:** *"If my current hidden state is a Noun, what is the probability that my next hidden state will be a Verb?"*
- **Where it comes from:** Counting tag pairs in the training corpus (e.g., how often `NOUN` is followed by `VERB`).

HMM: The Two Core Probabilities

2. Emission / Observation Probabilities

- **What it does:** Measures the probability that a specific hidden tag will "emit" or generate the specific word you are looking at.
- **The Question:** *"If I know the hidden state is a Noun, what is the probability that the visible word it spits out is 'dog'?"*
- **Where it comes from:** Counting how often the word "dog" is tagged as a `NOUN` versus a `VERB` in the training data.

The Formal Definition of an HMM

To mathematically define an HMM, we take the standard Markov Chain components (Q, A, and π) and augment them with our new Observation layer.

The Augmentation

- **O = {o_1, o_2, ..., o_T}:** A sequence of T **observations** (the actual words in the sentence we are trying to tag).
- **V = {v_1, v_2, ..., v_V}:** The **Observation Vocabulary** (a list of every possible word the model knows).
- **B = [b□(k)]:** The **Emission Probability Matrix** (or Observation Probabilities). This defines the probability of seeing observation v□ given that we are currently in hidden state q□: **b□(k) = P(O□ = v□ | q□ = q□)**

HMM: decoding

Once an HMM has learned all its Transition and Emission probabilities, you hand it a brand new sentence (like *"The dog barks"*). The HMM figures out the most likely sequence of hidden tags: **Decoding**.

$$\hat{T} = \arg\max_{T} P(T \mid W)$$

- **W:** The sequence of words you gave it (*The, dog, barks*).
- **T:** A possible sequence of tags (e.g., DET -> NOUN -> VERB).
- **T_hat:** The winning/predicted, most likely sequence of tags.
- **argmax_T:** The function that searches through every possible sequence of T to find the one that yields the highest probability.

Max vs. argmax

Why do we use `argmax` instead of a regular `max` function?

- **If we used `max`:** calculate every possible tag combination, find the best one, and simply return the winning probability score. That is useless to us—we don't want the score, we want the tags!
- **Because we use `argmax`:** The function evaluates the scores, returns the **arguments** (the inputs) that *produced* that maximum score. It returns `[DET, NOUN, VERB]`.

Hidden Markov Models

$$\hat{q} = argmax_{q_1 \ldots q_n} P(o_1 \ldots o_n \mid q_1 \ldots q_n) P(q_1 \ldots q_n)$$

Hidden Markov Models

$$\hat{q} = argmax_{q_1 \ldots q_n} P(o_1 \ldots o_n \mid q_1 \ldots q_n) P(q_1 \ldots q_n)$$

output independence

$$P(o_1 \ldots o_n \mid q_1 \ldots q_n) \approx \prod_{i=1}^{n} P(o_i \mid q_i)$$

Hidden Markov Models

$$\hat{q} = argmax_{q_1 \ldots q_n} P(o_1 \ldots o_n \mid q_1 \ldots q_n) P(q_1 \ldots q_n)$$

output independence

$$P(o_1 \ldots o_n \mid q_1 \ldots q_n) \approx \prod_{i=1}^{n} P(o_i \mid q_i)$$

Markov

$$P(t_1 \ldots t_n) \approx \prod_{i=1}^{n} P(t_i \mid t_{i-1})$$

Hidden Markov Models

$$\hat{q} = argmax_{q_1 \ldots q_n} P(o_1 \ldots o_n \mid q_1 \ldots q_n) P(q_1 \ldots q_n)$$

output independence

$$P(o_1 \ldots o_n \mid q_1 \ldots q_n) \approx \prod_{i=1}^{n} P(o_i \mid q_i)$$

Markov

$$P(t_1 \ldots t_n) \approx \prod_{i=1}^{n} P(t_i \mid t_{i-1})$$

$$\hat{q} \approx \text{argmax}_{q_1 \ldots q_n} \prod_{i}^{n} P(o_i \mid q_i) P(q_i \mid q_{i-1})$$

Hidden Markov Models

$$\hat{q} = argmax_{q_1 \ldots q_n} P(o_1 \ldots o_n \mid q_1 \ldots q_n) P(q_1 \ldots q_n)$$

output independence

$$P(o_1 \ldots o_n \mid q_1 \ldots q_n) \approx \prod_{i=1}^{n} P(o_i \mid q_i)$$

Markov

$$P(t_1 \ldots t_n) \approx \prod_{i=1}^{n} P(t_i \mid t_{i-1})$$

Emission

Transition

$$\hat{q} \approx \text{argmax}_{q_1 \ldots q_n} \prod_{i}^{n} P(o_i \mid q_i) P(q_i \mid q_{i-1})$$

Hidden Markov Models

Part of Speech Tagging

- In language, POS is *latent*, it is not observed
- So, we can use an HMM to find the most likely POS sequence for a sentence

HMM working example

Imagine we have a tiny HMM trained on a corpus.

- **Observations (Words):** "The", "bank", "can".
- **Hidden States (Tags): DET** (Determiner), **NOUN**, **VERB**, **MD** (Modal verb like 'can' or 'will').

HMM working example

1. The Ambiguity Problem

- **"bank"** could be a **NOUN** (a building) or a **VERB** (to tilt an airplane).
- **"can"** could be a **NOUN** (a metal container), a **VERB** (to preserve food), or a **MD** (a modal verb meaning "to be able to").

HMM needs to look at the context and decide which tag is correct.

HMM working example

2. The Two Components of the "Score"

For every word, the HMM looks at two probabilities to find the best path:

- **Transition Probability:** How likely is it that tag B follows tag A?
  - *Example:* In English, a **NOUN** is very likely to follow a **DET** (The bank). A **VERB** is very unlikely to follow another **VERB** directly.
- **Emission Probability:** If the tag is X, how likely is it that we see word Y?
  - *Example:* If the tag is **MD**, the probability of seeing the word "can" is very high. If the tag is **DET**, the probability of seeing the word "can" is 0.

Walking Through the Sentence: "The bank can..."

"The"

- **Initial Probability:** Most sentences start with a **DET**.
- **Emission:** The word "The" is almost always a **DET**.
- **Winner:** The HMM starts with the state **DET**.

"bank"

- The HMM compares two paths:
  1. **DET -> NOUN:** High transition probability (Determiners usually lead to nouns).
  2. **DET -> VERB:** Low transition probability (Determiners rarely lead straight to verbs).
- **Winner:** Even though "bank" *could* be a verb, the HMM chooses **NOUN** because of the "The" before it.

"can"

- The HMM compares:
  1. **NOUN -> MD:** High transition (Nouns are often followed by modal verbs like "The bank can...").
  2. **NOUN -> NOUN:** Lower transition (Two nouns in a row like "bank can" is less common than a subject-verb relationship).

Decoding

The HMM "decodes" the sequence by finding the path with the highest total score (multiplying all the transitions and emissions together).

Final Answer:

- **The** | DET
- **bank** | NOUN
- **can** | MD