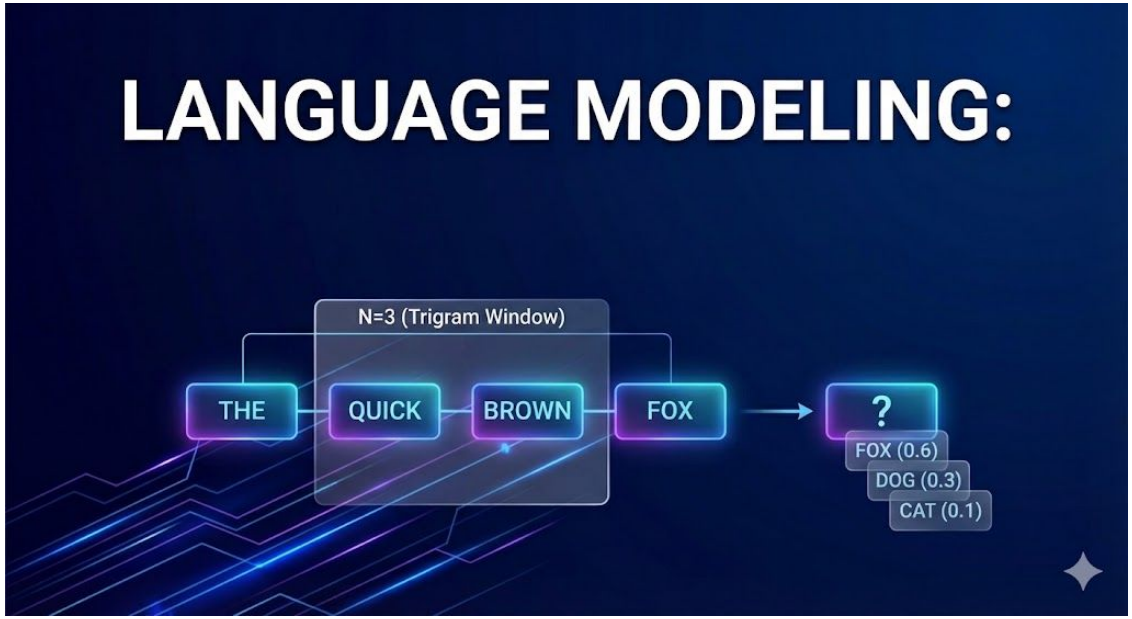


Natural Language Understanding

CSCI 4907/6515

Lecture 5: February 17, 2026

Aya Zirikly

Quiz 2

02/24/2026

Week 3, 4, and 5

- Please know your materials from week 1 & 2

Word Sense Disambiguation

- Task: automatically select the correct sense of a word
  - Input: a word in context
  - Output: sense of the word

- Motivated by many applications:
  - Information retrieval
  - Machine translation
  - …

How big is the problem?

- **Most words in English have only one sense**
  - 62% in Longman's Dictionary of Contemporary English
  - 79% in WordNet
- But the others tend to have several senses
  - Average of 3.83 in LDOCE
  - Average of 2.96 in WordNet

- **Ambiguous words are more frequently used**
  - In the British National Corpus, 84% of instances have more than one sense
- **Some senses are more frequent than others**

Baseline Performance

- Baseline: most frequent sense
  - Equivalent to “take first sense” in WordNet
  - Does surprisingly well!

Freq	Synset	Gloss
338	plant[1], works, industrial plant	buildings for carrying on industrial labor
207	plant[2], flora, plant life	a living organism lacking the power of locomotion
2	plant[3]	something planted secretly for discovery by another
0	plant[4]	an actor situated in the audience whose acting is rehearsed but seems spontaneous to the audience

62% accuracy in this case!

Upper Bound Performance (ceiling)

- Upper bound
  - Fine-grained WordNet sense: 75-80% human agreement
  - Coarser-grained inventories: 90% human agreement possible

(Social Role): Includes Chair #2 (Professor) and Chair #3 (Moderator).

Simplest WSD algorithm: Lesk's Algorithm

- Intuition: note word overlap between context and dictionary entries
  - **Unsupervised,** but knowledge rich

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"

Lesk's Algorithm

- Simplest implementation:
  - Count overlapping content words between glosses and context
- Lots of variants:
  - Include the examples in dictionary definitions
  - Include hypernyms and hyponyms
  - Give more weight to larger overlaps (e.g., bigrams)
  - Give extra weight to infrequent words
  - …

Terminology

- **Unsupervised learning:** Learning from unlabeled data
- **Supervised learning:** Learning from labeled data
- **Self-supervised learning:** Learning from labeled data (that is not generated by annotation; i.e., use the data itself as supervisory signal)

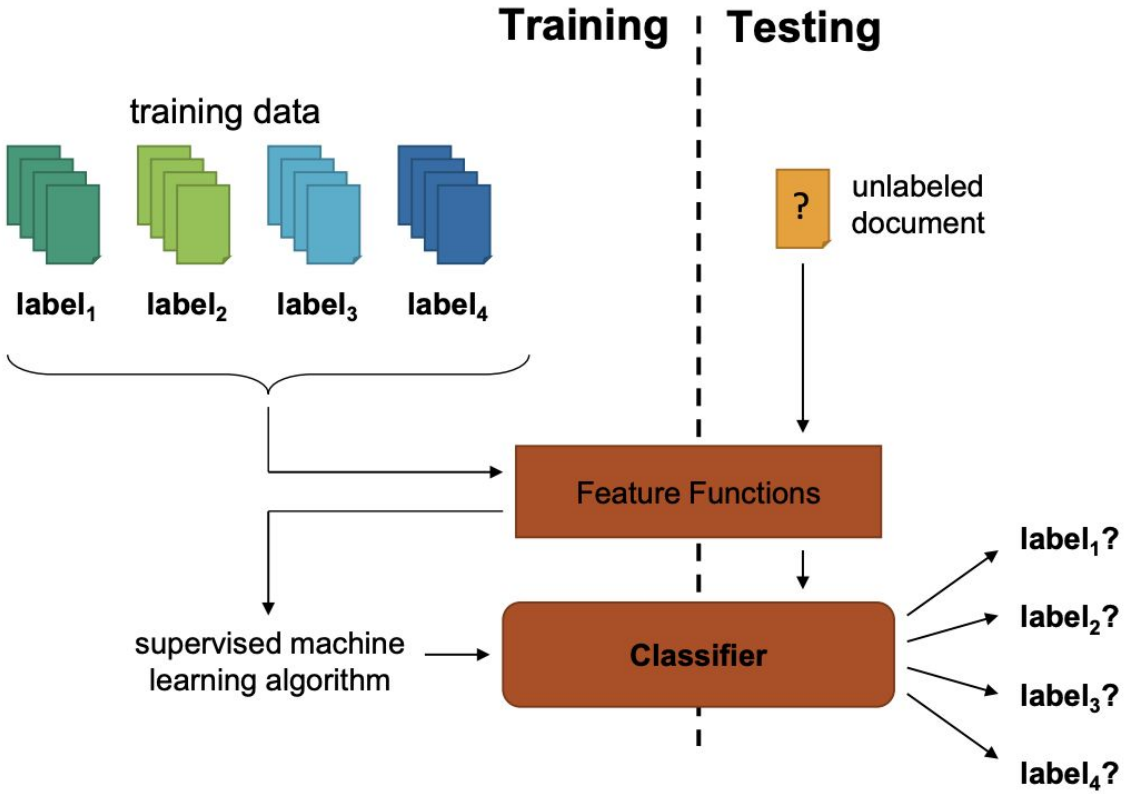
Lesk Algorithm?

Lesk Algorithm

Unsupervised/Knowledge-based

- Requires **Zero** training data
- You can run Lesk on a sentence immediately without ever showing the model a single example of "correct" disambiguation
- **Input:** Sentence + Dictionary
- **Output:** Best Sense

Alternative: WSD as **Supervised** Classification



WSD as supervised classification – Workflow

Input Take the sentence ***"The fisherman approached the bank."***
Feature Extraction: Convert the context into a numerical vector.

- *Is "fish" present?* [1]
- *Is "money" present?* [0]
- *Is previous word "the"?* [1]

Classifier: Feed this vector into an algorithm (Naive Bayes, SVM, or Neural Network).
Output: The classifier predicts the probability for each sense.

WSD as supervised classification – Workflow

The fisherman approached the bank

$P(\text{Sense} \mid \text{Context}) \propto P(\text{Context} \mid \text{Sense}) \cdot P(\text{Sense})$

Context: "fisherman", "approached"

Comparison:

- *Likelihood ("fisherman" | River Bank):* **High**
- *Likelihood ("fisherman" | Financial Bank):* **Low**

Result: Predict **Sense 2 (River)**

Other “tricks”

- One sense per discourse
  - If a polysemous word appears multiple times in a single document, it is extremely likely to have the same meaning every time.
- Word-aligned bilingual corpora

Summary

- Many words (lemmas) have multiple **senses** (meanings)
- The static vector embeddings we've seen so far do not take senses into account.
  - We get only one vector per word
- For many of our applications, we need to know which sense is being referred to (BERT)
  - For instance: how to translate "bank" to another language?
- Word sense disambiguation: given a word in context, output which sense is being used
  - Unsupervised and supervised approaches

Probabilistic Language Models

Course trajectory

Up until now, we've primarily been dealing with individual words

- Text categorization based on bag of words
- Vector space models and word embeddings

Now: Language as a sequence – how do words combine?

- Language modeling
- Parsing

Probabilistic Language Models

Today’s goal: Assign a probability to a sentence / string of words

Probabilistic Language Models

Today's goal: Assign a probability to a sentence / string of words

Why would probabilities for word sequences be useful? What kinds of tasks could this help with?

“Rick is starting a tornado garden”

“Rick is starting at a NATO garden”

“Rickets art innate omit a carton”

“Rick is starting a tomato garden”

“Rick is starting a tornado garden”

“Rick is starting at a NATO garden”

“Rickets art innate omit a carton”

“Rick is starting a tomato garden”

Why?

- Speech recognition
  - P(I saw a van) > P(eyes awe of an)
- Machine translation
  - P(‘**high** winds expected’) > P(‘**large** winds expected’)
- Spelling correction
  - The office is about fifteen minuets from my house
  - P(‘fifteen minutes’ > P(‘fifteen minuets’)
- Summarization, question-answering, …

Probabilistic Language Models

Assign a probability to a sentence / string of words

- **Word Embeddings (2013):** We used to initialize models with random noise. Then we realized we could **pre-train vectors** (Word2Vec) on a huge dataset and *reuse* them. This gave us a "generalized representation of **words**."
- **Language Models (2018+):** We realized that just knowing word definitions isn't enough; we need to know **grammar, syntax, and context**. So now, we pre-train entire networks (BERT/GPT) on massive text. This gives us a "generalized representation of **language**."

Predicting upcoming words

I like my coffee with cream and _____.

I like my coffee with cream and sugar.

I like my coffee with cream and socks.

He caught the pass and scored a touchdown. There was nothing he loved more than a game of _____.

He caught the pass and scored a touchdown. There was nothing he loved more than a game of football.

He caught the pass and scored a touchdown. There was nothing he loved more than a game of monopoly.

Probabilistic Language Modeling

- Goal: Compute the probability of a string of words
  - P(W) = P(w1, w2, w3, w4, w5)
- Related goal: Compute the probability of an upcoming word
  - P(w5 | w1, w2, w3, w4)
- A model that computes either of these is called a language model (LM) or grammar.

Language Modeling: The One Task to Rule Them All

Goal: We want to predict what the next word in a sentence will be

In other words, we want to get a probability distribution over all of the words in the vocabulary

1. It's useful/necessary for: question-answering, summarization, machine translation, etc.
2. We can frame other problems in this way e.g., sentiment analysis/classification, e.g.:

The sentiment of "I loved that movie" is _____.

1. We can also use it to build "representations" (think vectors, embeddings) that will allow us to do a good job at predicting what comes next. And then use those representations to decide e.g., the sentiment of a text

Goal: Computer p(w)

- How to compute this joint probability?

  P(W) = P(its, water, is, so, transparent, that)?

- In the past, we've gone to a corpus. Could we get a big corpus and estimate the probability of P(its, water, is, so transparent, that), by counting how many "its water is so transparent that" occurs?

The “Direct Counting”

- **Can we calculate** the probability of the entire sentence W = "its water is so transparent that"?
- **Formula:** P(its, water, is, so, transparent, that) = Count("its water is so transparent that") / N
  (Where N is the total number of sentences in our corpus)

The Problem: Data Sparsity

- Even in a massive corpus (like the entire internet), the specific sequence *"its water is so transparent that"* might not appear $\boldsymbol{P(W) = 0}$
- The model thinks this sentence is "impossible," which is obviously false. This is called **Data Sparsity**.

Chain Rule

To solve sparsity, we break the long sentence down into small steps using the **Chain Rule of Probability**.

Chain Rule

To solve sparsity, we break the long sentence down into small steps using the **Chain Rule of Probability**.

Conditional probability

$$P(B|A) = P(A,B) / P(A)$$

$$P(A|B) = P(A,B) / P(B)$$

Chain Rule

Conditional probability

$$P(B|A) = P(A,B) / P(A)$$

$$P(A|B) = P(A,B) / P(B)$$

Can be rewritten as $P(A,B) = P(A)P(B|A) = P(B)P(A|B)$ [**chain rule**]

Chain Rule

The General Rule $P(A, B, C, D) = P(A) \times P(B|A) \times P(C|A,B) \times P(D|A,B,C)$

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})$$

Chain Rule

The General Rule $P(A, B, C, D) = P(A) \times P(B|A) \times P(C|A,B) \times P(D|A,B,C)$

P(its) × P(water | its) × P(is | its, water) × P(so | its, water, is) × P(transparent | its, water, is, so)

How to calculate these probabilities

P(its) P(water|its) P(is|its water) P(so|its water is)
P(transparent | its water is so)

Could we count in a large text corpus and divide?

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{Count(\text{its water is so transparent that the})}{Count(\text{its water is so transparent that})}$$

How to calculate these probabilities

P(its) P(water|its) P(is|it

P(transparent

No! We'll never see enough data to estimate this. There are too many possible sentences!

Could we count in a large text

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{Count(\text{its water is so transparent that the})}{Count(\text{its water is so transparent that})}$$

Markov Assumption to the Rescue

Simplifying assumption: We can calculate the probability of future events without looking too far into the past

Markov Assumption to the Rescue

Simplifying assumption: We can calculate the probability of future events without looking too far into the past

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

Markov Assumption to the Rescue

Simplifying assumption: We can calculate the probability of future events without looking too far into the past

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

Or maybe

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

Markov Assumption to the Rescue

Simplifying assumption: We can calculate the probability of future events without looking too far into the past

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

Or maybe

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

we can approximate the true probability and make the calculation much easier

The Approximation (Markov Assumption)

1. The Assumption

- We approximate the full history with just the last 1 word (Bigram) or 2 words (Trigram).
- **Approximation:** P(is | its, water) ≈ P(is | water)

2. The New Calculation Instead of counting one massive sentence, we multiply many small probabilities:

- $\boldsymbol{P(W) \approx P(\text{its}) \times P(\text{water}|\text{its}) \times P(\text{is}|\text{water}) \times P(\text{so}|\text{is}) \times P(\text{transparent}|\text{so}) \ldots}$

3. The Benefit

- The pair "water is" is **very common**.
- The pair "is so" is **very common**.
- We can now calculate a valid, non-zero probability for the long sentence even if we have never seen it before!

The Approximation (Markov Assumption)

1. The Assumption

N-Grams

- We approximate the full history with just the last 1 word (Bigram) c).
- **Approximation:** P(is | its, water) ≈ P(is | water)

2. The New Calculation Instead of counting one massive sentence, we multiply many small probabilities:

- $P(W)$ ≈ P(its) × P(water|its) × P(is|water) × P(so|is) × P(transparent|so) ...

3. The Benefit

- The pair "water is" is **very common**.
- The pair "is so" is **very common**.
- We can now calculate a valid, non-zero probability for the long sentence even if we have never seen it before!

Markov Assumption

N-gram assumption

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$$

Markov Assumption

N-gram assumption

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$$

Goal 1: Calculate probability of a sentence

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})$$

Simplest case?

Simplest case: Unigram model (“Bag of words”)

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

Simplest case: Unigram model (“Bag of words”)

P(its water is so transparent)

= P(its) P(water|its) P(is|its water) P(so|its water is)
P(transparent | its water is so)

Simplest case: Unigram model (“Bag of words”)

$$P(\text{its water is so transparent})$$

$$= P(\text{its})\, P(\text{water}|\text{its})\, P(\text{is}|\text{its water})\, P(\text{so}|\text{its water is})\, P(\text{transparent} \mid \text{its water is so})$$

$$\approx P(\text{its})\, P(\text{water})\, P(\text{is})\, P(\text{so})\, P(\text{transparent})$$

Simplest case: Unigram model (“Bag of words”)

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

Unigram model

1. The Classic Example (Jurafsky & Martin) This was generated by a Unigram model trained on the Wall Street Journal:

> *"fifth, an, of, futures, the, an, incorporated, a, a, the, inflation, most, dollars, quarter, in, is, mass"*

2. The "Shannon" Example (1948) Claude Shannon, the father of information theory, generated this using a Unigram model:

> *"REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TO OF"*

Unigram model

What do you notice?

Unigram model

What do you notice?

- **No Grammar:** "a a the" (articles repeating).
- **Common Words:** It is mostly "the", "of", "in", "is" (Stop Words).
- **No Meaning:** It reads like a dictionary exploded.

Bigram model

Condition on previous word

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

Bigram model

1. **The Classic Example (Jurafsky & Martin)** Trained on the Wall Street Journal:

   "What means, sir. I confess she? then all sorts of great to the huge."

2. **The "Shakespeare" Example** Trained on Shakespeare's works:

   "The duke's a fine. The duke's a fine. The duke's a fine." (This happens because "The duke's" is a common start, and "a fine" is a common end, creating a loop).

Bigram model

What do you notice?

Bigram model

What do you notice?

- **Local Grammar:** "The huge" makes sense. "I confess" makes sense.
- **Global Nonsense:** The sentence as a whole means nothing.
- **Short Phrases:** It looks like someone glued together 2-word phrases:
  - {What means}
  - {means, sir}
  - {sir. I}
  - {I confess}

Bigram model

P(its water is so transparent)

= P(its) P(water|its) P(is|its water) P(so|its water is) P(transparent | its water is so)

≈ P(its|<s>) P(water|its) P(is|water) P(so|is) P(transparent|so)

Bigram model - why is it better?

The Bigram model is no longer rolling a random die. It is looking at a **Transition Matrix**.

Current Word: "The"

Next Options:

- ❏ "cat" (30%)
- ❏ "dog" (20%)
- ❏ "of" (0.1%) *(Big difference from Unigram!)*
- ❏ "the" (0.01%) *(Prevents stuttering!)*

You rarely get *"the the"* or *"is is"* because P(the | the) is very low. The model knows that a determiner is usually followed by a noun or adjective.

Bigram model - The garden path problem

Since the model only sees **1 step back**, it often walks into a linguistic dead end.

- **Start:** "The"
- **Next:** "bank" (Valid)
- **Next:** "is" (Valid)
- **Next:** "closed" (Valid)
- **Next:** "minded" (Wait... "closed-minded" is valid, but "The bank is closed-minded" is nonsense).

Bigrams produce **grammatically correct phrases** but **semantically incoherent sentences**.

We can extend to

Unigram model

Bigram model

Trigram model

4-gram model

5-gram model

…

Model	Memory	Example Output
Unigram	None	*"fifth, an, of, futures, the, an, incorporated"*
Bigram	Last 1 Word	*"What means, sir. I confess she? then all sorts"*
Trigram	Last 2 Words	*"Fly, and will rid me these news of price."*
4-Gram	Last 3 Words	*"The computer which I had just put into the machine"*

Computing probabilities
Maximum Likelihood Estimation (MLE)

- **Maximum Likelihood Estimation (MLE)** is a method for estimating probabilities by maximizing the likelihood of the observed data.
- In simple terms: We assume that the training corpus is a "perfect" representation of the language, so we calculate probabilities based *exactly* on what we see.

The Intuition

- If the word "cat" appears 10 times in a 100-word text

  MLE probability is **10/100 = 0.1**

- We maximize the likelihood that our model generates the training data.

Computing probabilities -MLE

- Bigram

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

- General n-gram

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})}$$

Computing probabilities -MLE

- Bigram

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

- General n-gram

$$P(w_n|w_{n-N+1}^{n-1}) = C$$

Count(w_{i-1}, w_i): How many times the pair "Previous Word + Current Word" appears.

Count(w_{i-1}): How many times the "Previous Word" appears in total.

MLE example

Corpus: "I read a book. I read a magazine."

Task: Calculate P(book | a)

Count("a book"): 1

Count("a"): 2

Result: 1 / 2 = **0.5**

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$P(\text{I}|\text{<s>})$ $P(\text{Sam}|\text{<s>})$ $P(\text{am}|\text{I})$

$P(\text{</s>}|\text{Sam})$ $P(\text{Sam}|\text{am})$ $P(\text{do}|\text{I})$

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

???

$P(\text{I}|\text{<s>})$ $P(\text{Sam}|\text{<s>})$ $P(\text{am}|\text{I})$

$P(\text{</s>}|\text{Sam})$ $P(\text{Sam}|\text{am})$ $P(\text{do}|\text{I})$

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$P(\text{I}|\text{<s>})$ $P(\text{Sam}|\text{<s>})$ $P(\text{am}|\text{I})$

$P(\text{</s>}|\text{Sam})$ $P(\text{Sam}|\text{am})$ $P(\text{do}|\text{I})$

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$P(\text{I}|\text{<s>}) = \frac{2}{3} = .67$ $P(\text{Sam}|\text{<s>}) = \frac{1}{3} = .33$ $P(\text{am}|\text{I}) = \frac{2}{3} = .67$

$P(\text{</s>}|\text{Sam}) = \frac{1}{2} = 0.5$ $P(\text{Sam}|\text{am}) = \frac{1}{2} = .5$ $P(\text{do}|\text{I}) = \frac{1}{3} = .33$

p(i|i)?

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

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

P(<s> i want chinese food </s>)?

2nd word

1st word

	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.

$$P(\texttt{i}\,|\,\texttt{<s>}) = 0.25 \qquad P(\texttt{english}|\texttt{want}) = 0.0011$$
$$P(\texttt{food}|\texttt{english}) = 0.5 \qquad P(\texttt{</s>}|\texttt{food}) = 0.68$$

Predicting future words: chain rule

- N-gram assumption

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$$

Putting it together in an example -Unigram prediction

He caught the pass and scored a touchdown. There was nothing he loved more than a game of _____.

Context Seen: ∅ (Nothing).
Logic: $P(w_i)$
The model ignores the sentence entirely. It simply picks the most frequent word in the English language.

There was nothing he loved more than a game of **the**.

the
off
and

Putting it together in an example -bigram prediction

He caught the pass and scored a touchdown. There was nothing he loved more than a game of _____.

Context Seen: "of"
Logic: $P(w_i \mid \text{"of"})$
The model looks at the last word and asks: "What usually follows 'of'?"

There was nothing he loved more than a game of **the**.

"the" (High probability: "of the")
"a" (High probability: "of a")
"course" (Common phrase: "of course")

Putting it together in an example -trigram prediction

He caught the pass and scored a touchdown. There was nothing he loved more than a game of _____.

Context Seen: "game of"
Logic: $P(w_i \mid$ "game", "of")
The model looks up: "What completes the phrase 'game of'?"

There was nothing he loved more than a game of **Thrones**.

Dependent on corpus
Thrones" (If trained on modern internet data: "Game of Thrones")
"life" ("Game of Life")
"chance" ("Game of chance")

Putting it together in an example -trigram prediction

There was nothing he loved more than a game of **Thrones**.

Dependent on corpus
Thrones" (If trained on modern internet data: "Game of Thrones")
"life" ("Game of Life")
"chance" ("Game of chance")

Why not "Football"?

- The word **"touchdown"** is 13 words away.
- The Trigram model **cannot see it**. It has a "horizon" of 2 words.
- It knows "game of Thrones" is statistically more common than "game of Football" (people usually just say "a football game").
- The model ignores the sports context entirely.

Comparison with LLM

Just for contrast, here is what a modern model (like BERT or GPT) does.

- **Context Seen:** The entire paragraph, including "caught", "pass", and "touchdown".
- **Self-Attention:** The model attends to "touchdown" and "pass".
- It recognizes the semantic cluster: **{Sports, American Football}**.

*"There was nothing he loved more than a game of **football***

"Horizon Problem": why N-grams fail at long-distance context

Horizon problem

The **Horizon** is the maximum distance an N-gram model can "see" into the past.
Mathematically, for an N-gram model, the horizon is fixed at $\boldsymbol{N-1}$ **words**.
Any information that appears *before* this window is effectively invisible to the model.

Horizon problem -subject/verb agreement

"The **students** who studied for the exam in the library during the storm **are** happy."

- ❑ **Target Word: are** (Plural Verb).
- ❑ **Dependency:** It must agree with the subject **students** (Plural Noun).
- ❑ **Distance:** There are **10 words** between "students" and "are".

The N-Gram Failure

- **Trigram View (*N*=3):** It sees only *"the storm ___"*.
- **Prediction:** Since "storm" is singular, the model predicts **is**.
  - *Prediction:* "The storm **is** happy." (Grammatically correct locally, but wrong for the sentence).
- **Horizon:** The word "students" fell off the horizon 8 words ago.

Why not increase *N*

You might ask: *"Why not just use a 20-gram model?"*

The Explosion

- As you increase the Horizon (N), the number of possible combinations grows exponentially.

Formula: $|V|^N$

- Where $|V|$ is the Vocabulary size (e.g., 50,000 words).
- Where N is the gram size.

Even for a modest 4-gram model, you are looking for $50{,}000^4$ possible sequences. That is **6.25×10^{18}** combinations.

Why not increase *N*

If you look for a specific 20-word sequence (N=20) to predict the next word, you will almost certainly never find it in your training data.

Result: $P(w_t \mid w_{t-19} \ldots w_{t-1}) \approx 0$

The Dilemma:

- **Small N (Bigram):** High Bias (Dumb errors due to lack of context).
- **Large N (20-gram):** High Variance (Overfitting; the model thinks valid sentences are "impossible" because it hasn't seen them).

Solution

This problem was solved by moving away from "Counting" (Discrete) and toward "Memory" (Continuous).

Recurrent Neural Networks (RNNs)

- Instead of a fixed window, pass a "hidden state" (memory vector) forward through time.
- **Logic:** $h_t = f(h_{t-1}, x_t)$
  - The memory at time t depends on the memory at $t-1$.
- **Theoretical Horizon:** ∞ (Infinite).
- **Practical Horizon:** ~50 words (limited by the "Vanishing Gradient" problem).

Solution

Transformers (Attention)

- **Mechanism:** Self-Attention (Q, K, V).
- The model can "attend" to any word in the sequence (w_1) at time (w□), no matter how far apart they are.
- **Result:** The "Horizon" is effectively gone. The model sees the entire paragraph as a single, connected graph.

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