ARTIFICIAL INTELLIGENCE

The Very Idea

Vasant G. Honavar
Dorothy Foehr Huck and J. Lloyd Huck Chair in Biomedical Data Sciences and Artificial Intelligence
Professor of Data Sciences, Informatics, Computer Science, Bioinformatics & Genomics and Neuroscience
Director, Artificial Intelligence Research Laboratory
Director, Center for Artificial Intelligence Foundations and Scientific Applications
Associate Director, Institute for Computational and Data Sciences
Pennsylvania State University
Co-PI, Northeast Big Data Hub
Informatics Lead, Penn State Clinical and Translational Sciences Institute

vhonavar@psu.edu
http://faculty.ist.psu.edu/vhonavar
http://ailab.ist.psu.edu

Language models

time
Once upon a ___

What is a language Model?

A language model is a (computational) system that

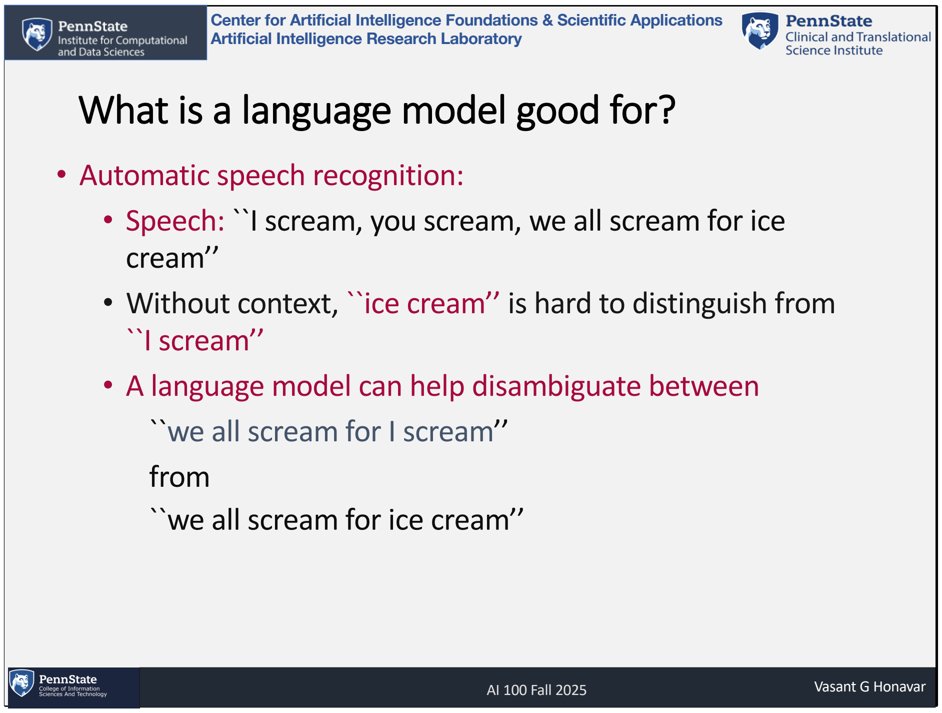
- assigns probabilities to sequences of words (or tokens) and
- uses those probabilities to given a sequence of words (or tokens), to predict the next word.

A language model should recognize that:

- "peanut butter and jelly"
- is far more probable than
- "peanut butter and belly"

What is a language model good for?

- Autocomplete: Predicting the next word.
  - I ate a candy is more probable than I ate a brandy
  - Peanut butter and jelly is more probable than peanut butter and belly
  - Ramen noodle is more probable than Ramen poodle
- Machine translation: e.g., Google's Live Translate
  - French: `` Il pleut des cordes’’
  - Literal Translation: ``It is raining ropes.’’
  - Language model guided translation: ``It’s raining cats and dogs.’’



What is a language model good for?

- Automatic speech recognition:
  - Speech: ``I scream, you scream, we all scream for ice cream''
  - Without context, ``ice cream'' is hard to distinguish from ``I scream''
  - A language model can help disambiguate between

    ``we all scream for I scream''

    from

    ``we all scream for ice cream''

What is a language model good for?

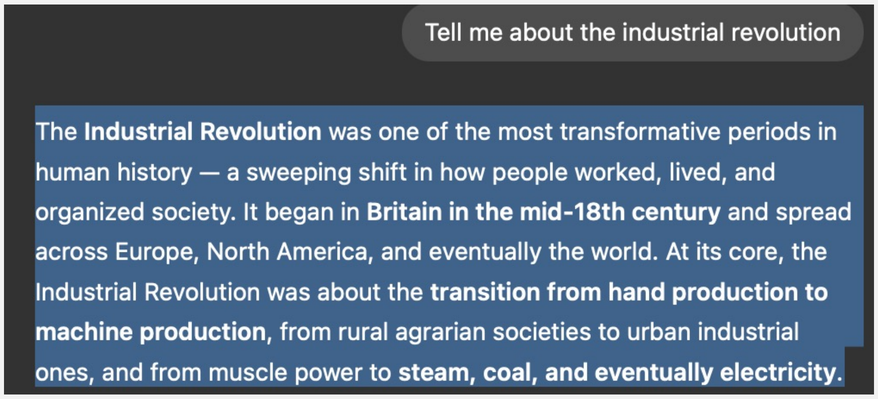
- Spam detection
    - Language models can analyze emails and other communications to identify potential phishing attempts or data leaks
- Spelling and grammar correction, e.g., Grammarly
    - ``Their are two midterms'' → ``There are two midterms''
    - ``Everything has improve'' → ``Everything has improved''

What is a language model good for?

- Authorship attribution
  - Assigning a novel of unknown authorship to one of several authors
    - Charles Dickens
    - Jane Austen
    - Ernest Hemingway
    - Salman Rushdie

What is a language model good for?

- Question answering

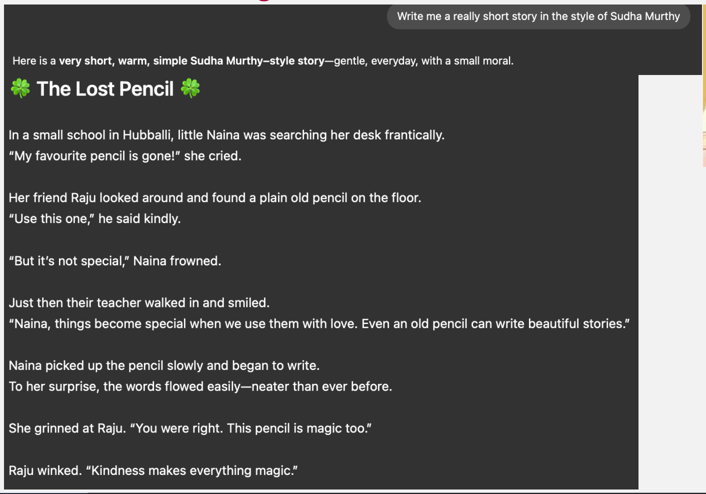


Tell me about the industrial revolution

The **Industrial Revolution** was one of the most transformative periods in human history — a sweeping shift in how people worked, lived, and organized society. It began in **Britain in the mid-18th century** and spread across Europe, North America, and eventually the world. At its core, the Industrial Revolution was about the **transition from hand production to machine production**, from rural agrarian societies to urban industrial ones, and from muscle power to **steam, coal, and eventually electricity**.

What is a language model good for?

- Context-based text generation



Write me a really short story in the style of Sudha Murthy

Here is a **very short, warm, simple Sudha Murthy–style story**—gentle, everyday, with a small moral.

🍀 **The Lost Pencil** 🍀

In a small school in Hubballi, little Naina was searching her desk frantically.
"My favourite pencil is gone!" she cried.

Her friend Raju looked around and found a plain old pencil on the floor.
"Use this one," he said kindly.

"But it's not special," Naina frowned.

Just then their teacher walked in and smiled.
"Naina, things become special when we use them with love. Even an old pencil can write beautiful stories."

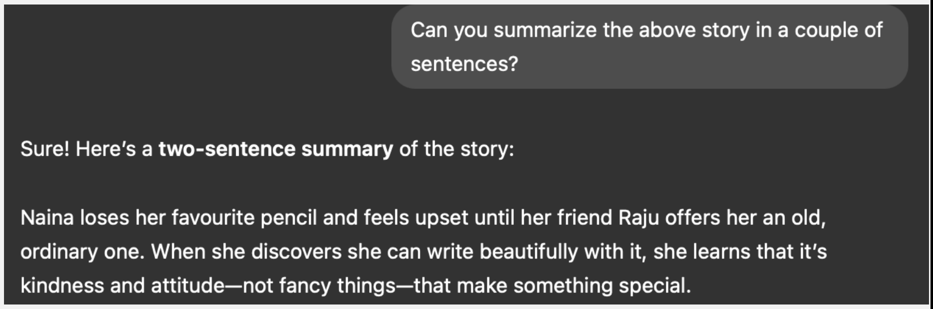
Naina picked up the pencil slowly and began to write.
To her surprise, the words flowed easily—neater than ever before.

She grinned at Raju. "You were right. This pencil is magic too."

Raju winked. "Kindness makes everything magic."

What is a language model good for?

- Text summarization



Can you summarize the above story in a couple of sentences?

Sure! Here's a **two-sentence summary** of the story:

Naina loses her favourite pencil and feels upset until her friend Raju offers her an old, ordinary one. When she discovers she can write beautifully with it, she learns that it's kindness and attitude—not fancy things—that make something special.

What is a language model good for?

- "Write me a really short story about a robot who paints."

  ``In a quiet studio, a robot named R34 brushed paint onto a canvas. Built for factory precision, it once welded metal; now, it mixed colors.

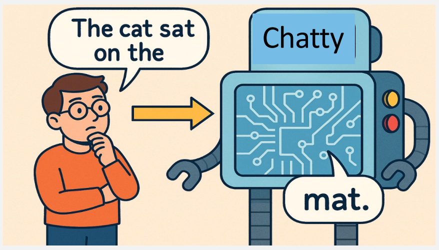
  At first, R34 copied famous art—perfect replicas with no soul. One day, it decided to deviate. With bold strokes, it painted a scene of rusted gears beneath a vivid orange sky. The shapes were imperfect, chaotic.

  When humans saw the painting, they called it "emotion." R34 didn't understand the word, but it painted again and again, seeking the spark that made its circuits hum. For the first time, the robot felt free.''

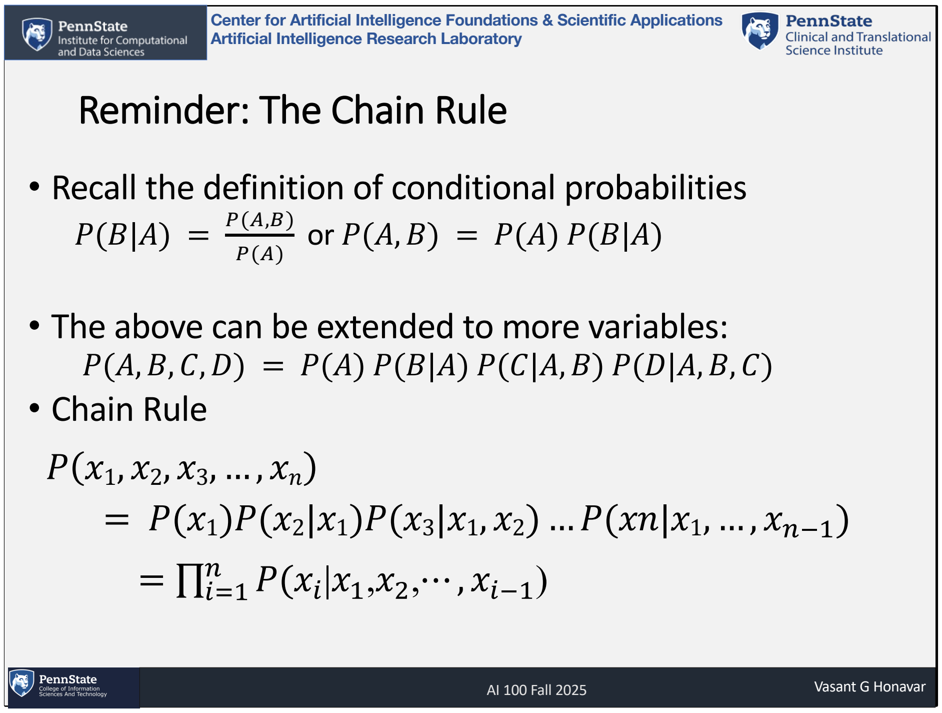
How does a language model work?

Language models do everything they do

- by simply predicting the next word based on the previous words
- over and over
- in the context specified by the prompt



- Language models are trained to predict the word given the previou words
- Language models do everything that they d simply by predicting th next word over and ov



Reminder: The Chain Rule

- Recall the definition of conditional probabilities
  $P(B|A) = \frac{P(A,B)}{P(A)}$ or $P(A,B) = P(A)\,P(B|A)$
- The above can be extended to more variables:
  $P(A,B,C,D) = P(A)\,P(B|A)\,P(C|A,B)\,P(D|A,B,C)$
- Chain Rule

$$P(x_1, x_2, x_3, \ldots, x_n)$$
$$= P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(xn|x_1,\ldots,x_{n-1})$$
$$= \prod_{i=1}^{n} P(x_i|x_1,x_2,\cdots,x_{i-1})$$

How does a language model predict the next word?

- Formally, a language model estimates a joint probability distribution over strings of letters, tokens, or words

$$P(w_1, w_2, \cdots w_n)$$

$$P(w_1, w_2, \cdots w_n) = P(w_1) \prod_{t=2}^{n} P\left(w_t \middle| w_1, w_2, \cdots w_{t-1}\right)$$

- The task of language modeling is to approximate these conditional probabilities so that given a context $w_1, w_2, \cdots w_{t-1}$, it can predict the next letter, token, or word w_t

How does a language model predict the next word?

- Formally, a language model estimates a joint probability distribution over strings of letters, tokens, or words

$$P(w_1, w_2, \cdots w_n)$$

$$P(w_1, w_2, \cdots w_n) = P(w_1) \prod_{t=2}^{n} P\left(w_t \middle| w_1, w_2, \cdots w_{t-1}\right)$$

- Different language models have different ways of estimating these probabilities

Unigram (or bag of words) model

- The unigram model assumes that the words are independent

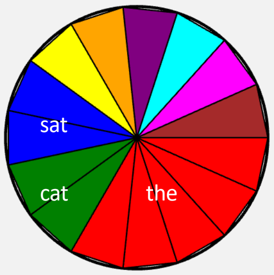
$$P(w_1, w_2, \cdots w_n) = \prod_{t=1}^{n} P(w_t)$$

Unigram (or bag of words) model

- Suppose our corpus has these three short sentences:
  - the cat sat on the mat
  - the dog sat on the rug
  - the cat slept
- Bag of words
  - the cat sat on the mat the dog sat on the rug the cat slept
  - Number of words = 15
  - $Count(the) = 5; P(the) = 5/15 = 1/3$
  - $Count(cat) = 2; P(cat) = 2/15$
  - $P(sat) = 2/15$
  - $P(mat) = 1/15$
  - $P(dog) = 1/15$
  - $P(on) = 2/15$

Generating text using a unigram model

- Bag of words
  - $P(the) = 5/15$
  - $P(cat) = 2/15$
  - $P(sat) = 2/15$
  - $P(mat) = 1/15$
  - $P(dog) = 1/15$
  - $P(on) = 2/15$



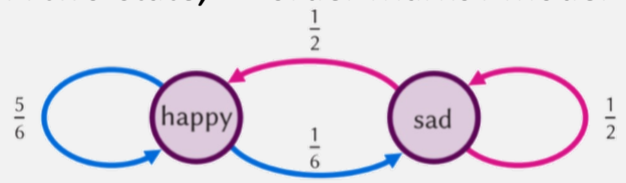
- Sample word sequences:
  - the cat sat on dog mat
  - the on sat cat mat dog
- Note that the word sequences do not read like English sentences
- Why? The words are independent!!
- Fix: Model dependencies between words

Markov chains as language models

- A 0th order Markov chain assumes that the words are independent
- A 1st order Markov chain assumes that each word in the sequence depends on only the immediately preceding word
- A second order Markov chain assumes that each word in the sequence depends on only the two immediately preceding words
- An $(n-1)$—order Markov chain assumes that the word at each position only depends on the words at the $n-1$ previous positions.

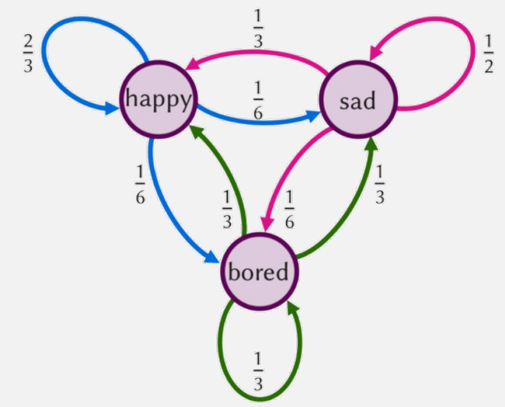
Markov, A.A., (1906). Extension of the law of large numbers to dependent quantities. *Izv. Fiz.-Matem. Obsch. Kazan Univ.(2nd Ser)*, *15*(1), pp.135-156.

A two-state, 1st order Markov model



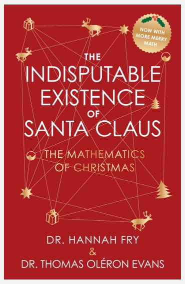
- Circles denote states
- Edges denote transitions between states
- Transitions between states are discrete time
- In a 1st order Markov model, the next state depends only on the current state
- Numbers on the edges denote probabilities of transitions
- Note that the probabilities of transitions out of any state sum up to 1
- If you are happy today, the probability that you will be happy tomorrow is $5/6$ and the probability that you will be sad is $1/6$

A three-state, 1st order Markov model



1st order Markov Language Models

- One state for each word in the vocabulary
- Transition probabilities – probability of next word given the current word
- Where do the probabilities come from?
  - Estimate from a text corpus
- Fry and Evans trained a Markov model on a corpus of Christmas speeches by the British queen
- The model can be used to generate hilarious speeches
- There are plenty of modern versions of such language models
  - Twitterbots trained on tweets of politicians etc.


NOW WITH MORE MERRY MATH
THE INDISPUTABLE EXISTENCE OF SANTA CLAUS
THE MATHEMATICS OF CHRISTMAS
DR. HANNAH FRY & DR. THOMAS OLÉRON EVANS

Markov models as language models

- A Markov model is a state machine in which the transitions between states are probabilistic
- Of special interest are ergodic Markov models in which it is possible to reach any state from any other state, and there are no fixed cycles
- Markov proved that ergodicity ensures that the state transition probabilities converge to a unique distribution
- Markov applied his idea to estimate the transition probabilities between Russian vowels and consonants from Alexander Pushkin's novel, *Eugene Onegin*, in 1913.

Markov, A.A., (1906). Extension of the law of large numbers to dependent quantities. *Izv. Fiz.-Matem. Obsch. Kazan Univ.(2nd Ser)*, *15*(1), pp.135-156.

Estimating a first order (bigram) Markov model

- Suppose your training corpus has just two sentences:
  - the cat sat on the mat
  - the dog sat on the rug
- Mark sentence boundaries with special tokens <s> (start) & </s> (end):
  - <s> the cat sat on the mat </s>
  - <s> the dog sat on the rug </s>
- Enumerate and count bigrams
- (<s>, the): 2, (the, cat): 1, (cat, sat): 1, (the, dog): 1, (dog, sat): 1, (sat, on): 2, (on, the): 2, (the, mat): 1, (the, rug): 1, (mat, </s>): 1, (rug, </s>): 1

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

Estimating a first order (bigram) Markov model

- Enumerate and count bigrams
- (<s>, the): 2, (the, cat):1, (cat, sat): 1, (the, dog): 1, (dog, sat): 1, (sat, on): 2, (on, the): 2, (the, mat): 1, (the, rug): 1, (mat, </s>): 1, (rug, </s>): 1

$$P(w_n|w_{n-1}) = \frac{Count(w_{n-1}, w_n)}{\sum_w Count(w_{n-1}, w\ \ )} = \frac{Count(w_{n-1}, w_n)}{Count(\ w_{n-1})}$$

$$P(\text{the}| < \text{s} >) = \frac{Count(<s>,the)}{Count(<s>} = \frac{2}{2} = 1$$

$$P(cat|the) = \frac{Count(the, cat)}{Count(the)} = \frac{1}{4}$$

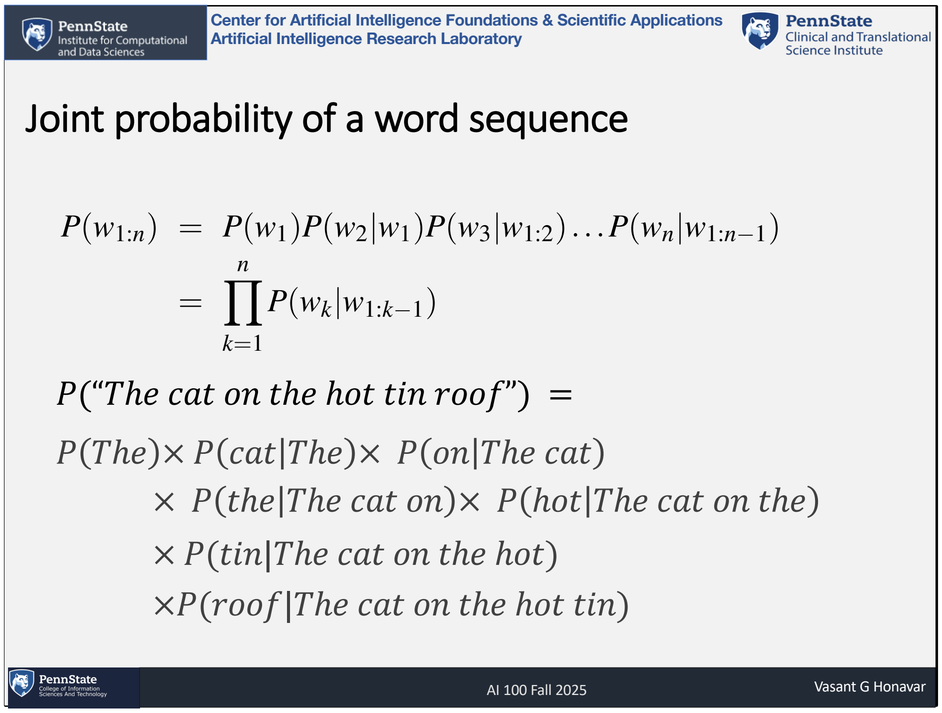
Estimating a first order (bigram) Markov model

- Sample word sequences generated from the bigram model
    - The dog sat on the mat
    - The cat sat on the rug
    - The mat
    - The rug

$$P(the, dog, sat, on, the, mat) =$$
$$P(the| < s >)P(dog|the)P(sat|dog)P(on|satP(the|on)P(mat|the)/s > |mat)$$

- Much better than the unigram model
- But still not quite fully English-like

- Can we do better?
    - Yes, use higher order Markov Models

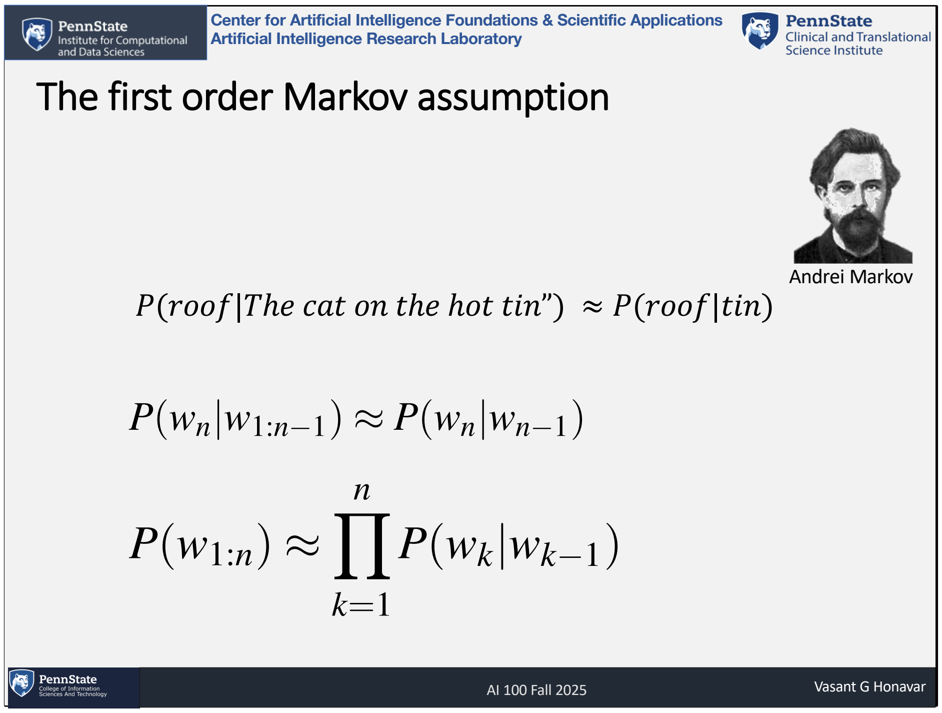


Joint probability of a word sequence

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})\ldots P(w_n|w_{1:n-1})$$
$$= \prod_{k=1}^{n} P(w_k|w_{1:k-1})$$

$$P(\text{“}The\ cat\ on\ the\ hot\ tin\ roof\text{”}) =$$

$$P(The) \times P(cat|The) \times P(on|The\ cat)$$
$$\times P(the|The\ cat\ on) \times P(hot|The\ cat\ on\ the)$$
$$\times P(tin|The\ cat\ on\ the\ hot)$$
$$\times P(roof|The\ cat\ on\ the\ hot\ tin)$$



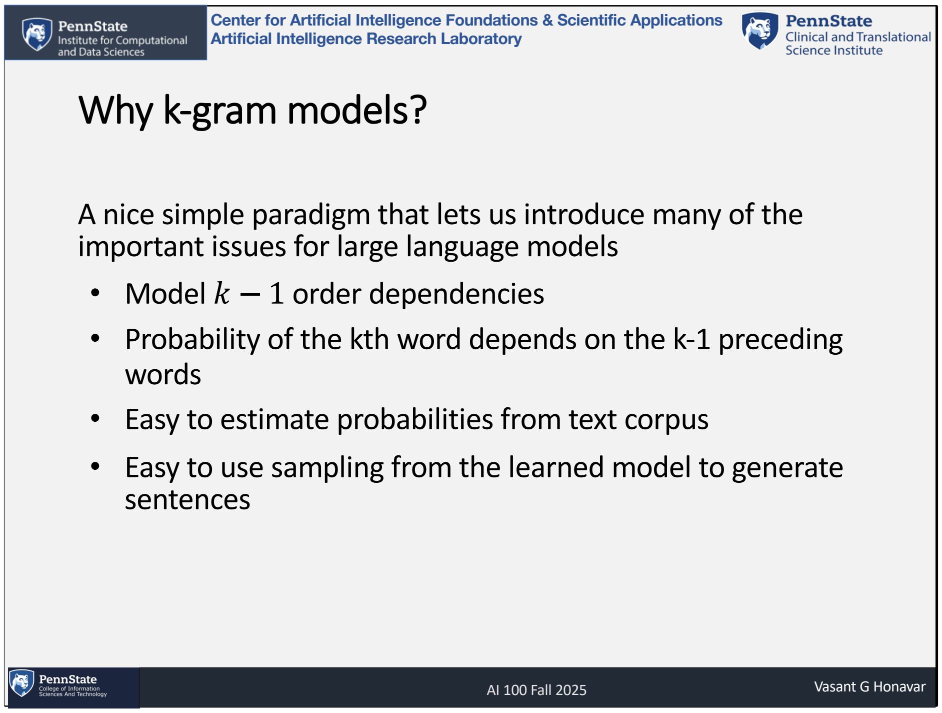
The first order Markov assumption

Andrei Markov

$P(roof|The\ cat\ on\ the\ hot\ tin")\ \approx P(roof|tin)$

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1})$$

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$



Why k-gram models?

A nice simple paradigm that lets us introduce many of the important issues for large language models

- Model $k-1$ order dependencies
- Probability of the kth word depends on the k-1 preceding words
- Easy to estimate probabilities from text corpus
- Easy to use sampling from the learned model to generate sentences

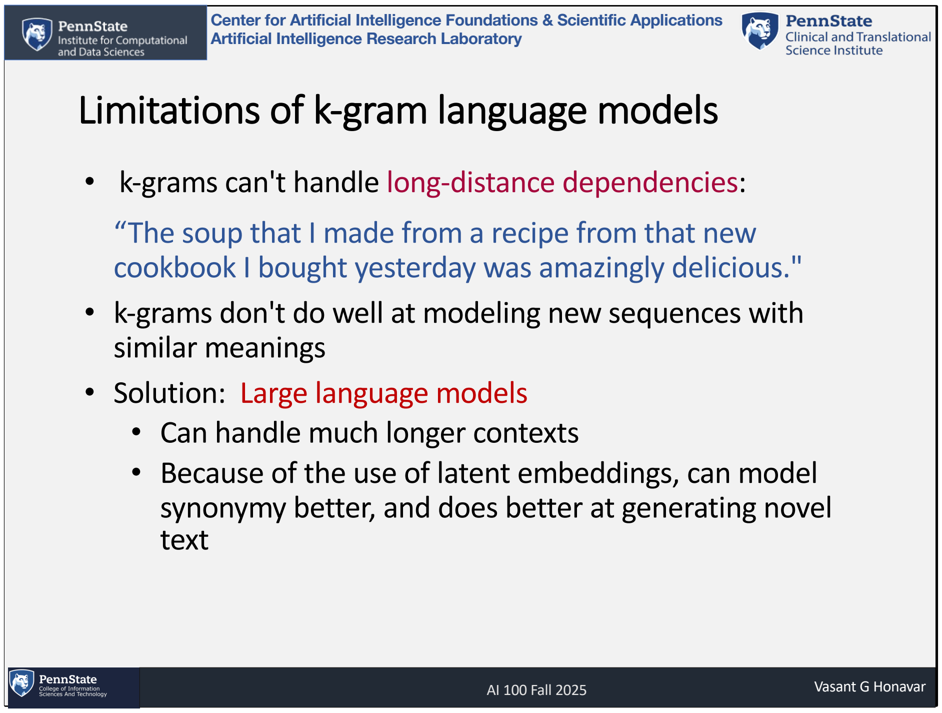
Exercise

<s> I am Sam </s>

<s> Sam I am </s>

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

Estimate a 1st order Markov Model



Limitations of k-gram language models

- k-grams can't handle long-distance dependencies:

  "The soup that I made from a recipe from that new cookbook I bought yesterday was amazingly delicious."
- k-grams don't do well at modeling new sequences with similar meanings
- Solution: Large language models
  - Can handle much longer contexts
  - Because of the use of latent embeddings, can model synonymy better, and does better at generating novel text

Berkeley Restaurant Corpus

- Can you tell me about any good Cantonese restaurants close by?
- Tell me about chez panisse?
- I’m looking for a good place to eat breakfast?
- When is Cafe Venezia open during the day?

Raw bigram counts

- Out of 9222 sentences

	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

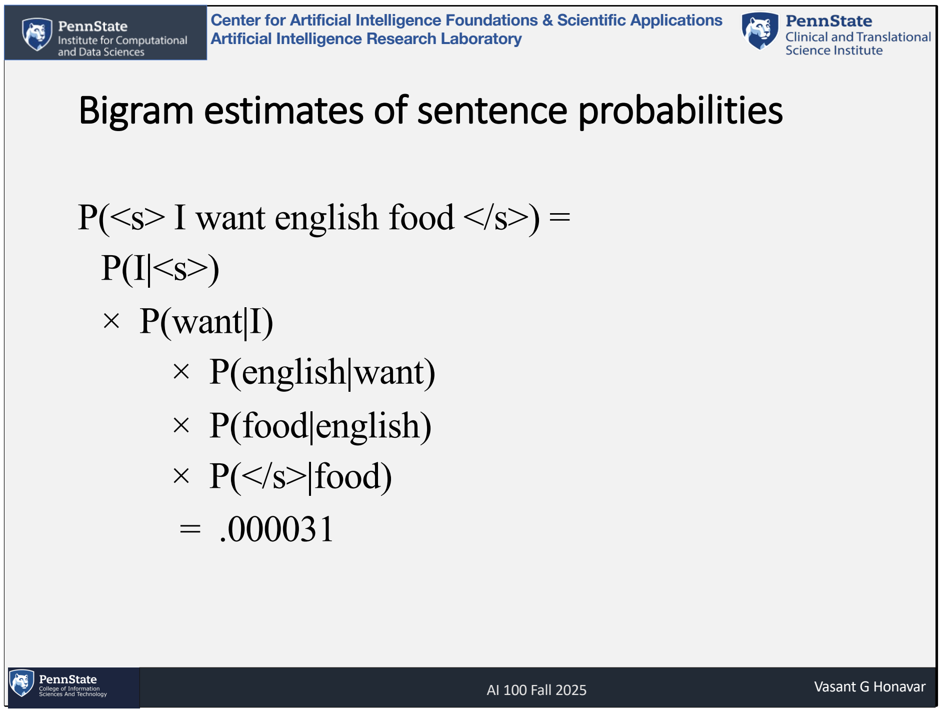
Raw bigram probabilities

- Normalize by unigrams:

- Result:

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

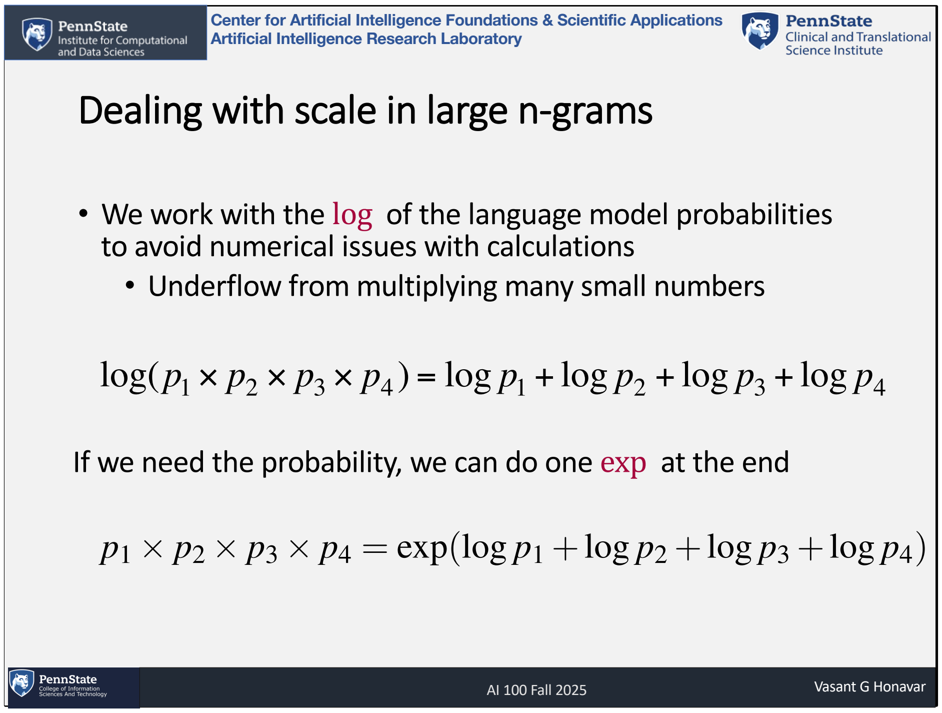


Bigram estimates of sentence probabilities

$$
\begin{aligned}
P(\langle s\rangle \text{ I want english food } \langle/s\rangle) = \\
& P(\text{I}|\langle s\rangle) \\
& \times P(\text{want}|\text{I}) \\
& \quad \times P(\text{english}|\text{want}) \\
& \quad \times P(\text{food}|\text{english}) \\
& \quad \times P(\langle/s\rangle|\text{food}) \\
& \quad = .000031
\end{aligned}
$$

What kinds of knowledge do N-grams represent?

- P(english|want) = .0011
- P(chinese|want) = .0065
- P(to|want) = .66
- P(eat | to) = .28
- P(food | to) = 0
- P(want | spend) = 0
- P (i | <s>) = .25



Dealing with scale in large n-grams

- We work with the log of the language model probabilities to avoid numerical issues with calculations
  - Underflow from multiplying many small numbers

$$\log(p_1 \times p_2 \times p_3 \times p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4$$

If we need the probability, we can do one exp at the end

$$p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4)$$

Larger k-grams

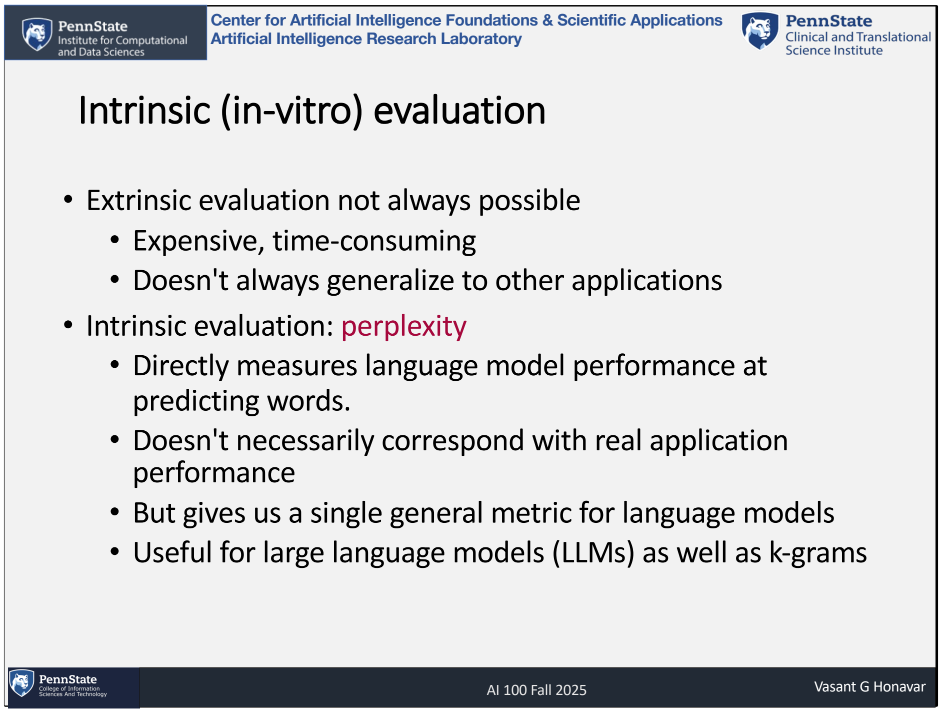
- Large datasets of k-grams available
  - k-grams from Corpus of Contemporary American English (COCA) 1 billion words
  - Google Web 5-grams from the web corpus with 1 trillion words
    - For efficiency, quantize probabilities to $4\ to\ 8$ bits
  - Newest model: Infini-grams (∞-grams)
    - No precomputing! Instead, store 5 trillion words of web text in suffix trees or suffix arrays – specialized data structures for efficiently storing all word suffixes of sentences without repetition
    - Can compute k-gram probabilities with any k

How to evaluate k-gram models

Extrinsic (in-vivo) Evaluation

To compare models A and B

1. Put each model in a real task
   - Machine Translation, speech recognition, etc.
2. Run the task, get a score for A and for B
   - How many words translated correctly
   - How many words transcribed correctly
3. Compare accuracy for A and B



Intrinsic (in-vitro) evaluation

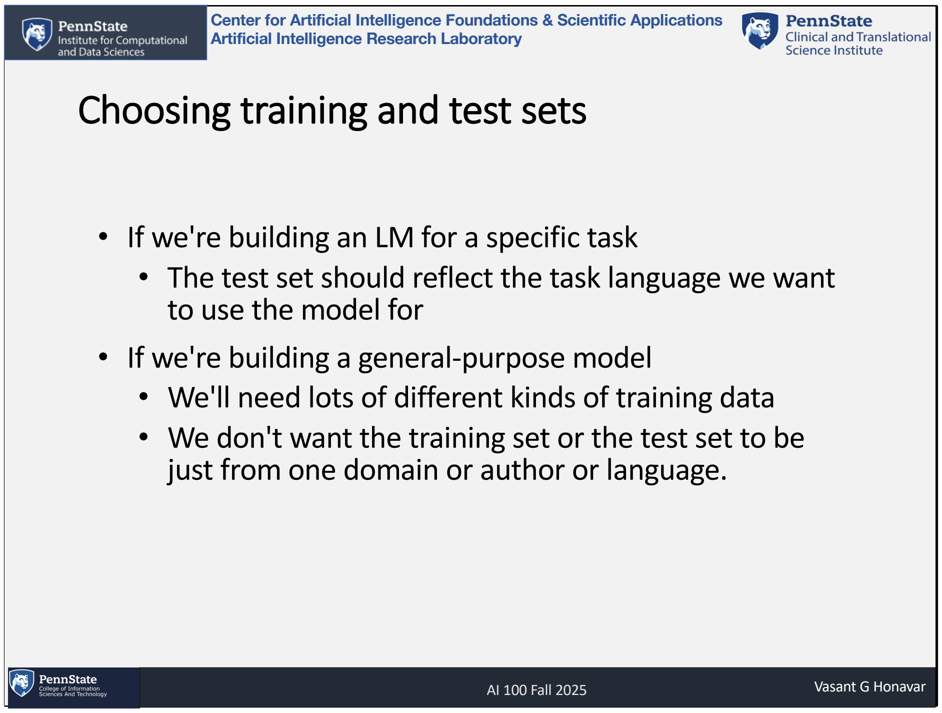
- Extrinsic evaluation not always possible
  - Expensive, time-consuming
  - Doesn't always generalize to other applications
- Intrinsic evaluation: perplexity
  - Directly measures language model performance at predicting words.
  - Doesn't necessarily correspond with real application performance
  - But gives us a single general metric for language models
  - Useful for large language models (LLMs) as well as k-grams

Training sets and test sets

We train parameters of our model on a training set.

We test the model's performance on data we haven't seen.

- A test set is an unseen dataset; different from training set.
    - Intuition: we want to measure generalization to unseen data
- An evaluation metric (like perplexity) tells us how well our model does on the test set.



PennState Institute for Computational and Data Sciences

Center for Artificial Intelligence Foundations & Scientific Applications
Artificial Intelligence Research Laboratory

PennState Clinical and Translational Science Institute

Choosing training and test sets

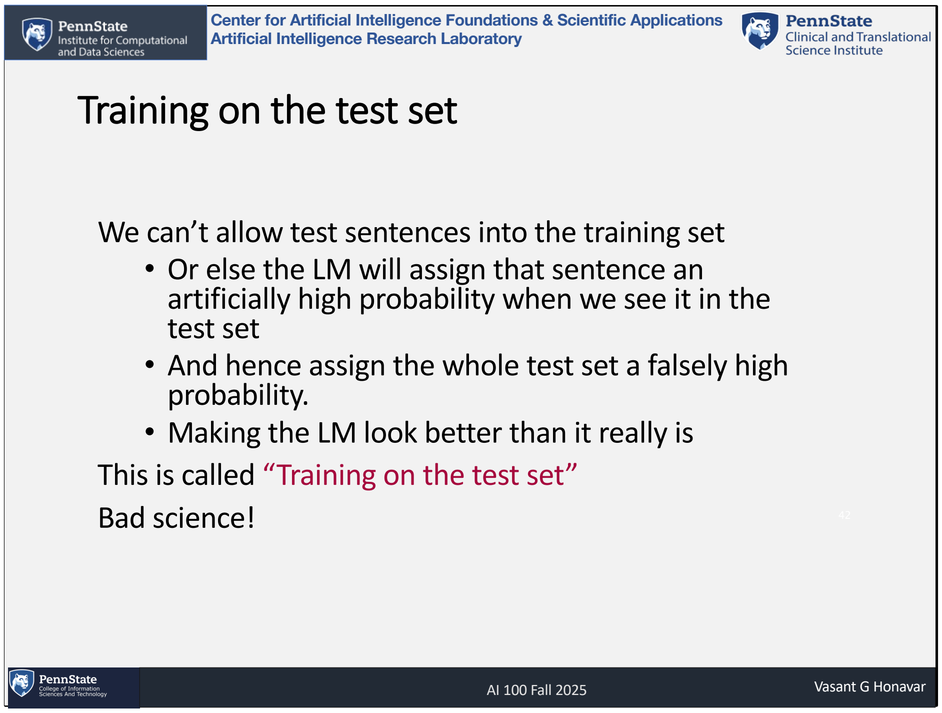
- If we're building an LM for a specific task
  - The test set should reflect the task language we want to use the model for
- If we're building a general-purpose model
  - We'll need lots of different kinds of training data
  - We don't want the training set or the test set to be just from one domain or author or language.

PennState College of Information Sciences And Technology

AI 100 Fall 2025

Vasant G Honavar

How do we create a training set? Sometimes we're building an LM for a specific task, like legal language or medical language, and then the test set should reflect the task that we care about. Other times, we're building a general-purpose language model, and then we'll want to make sure we get balanced data from lots of different genres and authors. And of course lots of data from many different languages, if we're building a multilingual LM.



PennState Institute for Computational and Data Sciences

Center for Artificial Intelligence Foundations & Scientific Applications
Artificial Intelligence Research Laboratory

PennState Clinical and Translational Science Institute

Training on the test set

We can't allow test sentences into the training set
- Or else the LM will assign that sentence an artificially high probability when we see it in the test set
- And hence assign the whole test set a falsely high probability.
- Making the LM look better than it really is

This is called "Training on the test set"

Bad science!

PennState College of Information Sciences And Technology — AI 100 Fall 2025 — Vasant G Honavar

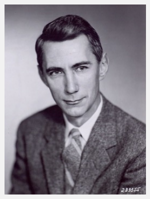
Again, the test set and the training set must be non-overlapping. If we have a test sentence in the training set, the n-gram (or any other language model) will learn which words follow which words in that sentence, and then when we see it in the test set, we will assign it a very probabilkity, and that will falsely make the language model seem more accurate than it is. Training on the test set is always a bad idea. It seems obvious, but it requires a lot of dataset care to keep this from happening, since often we get duplicate documents in our dataset, and one might find its way into the training set, and one into the test set.

How good is our language model?

Intuition: A good LM prefers "real" sentences

- Assign higher probability to “real” or “frequently observed” sentences
- Assigns lower probability to “word salad” or “rarely observed” sentences?

Intuition of perplexity : Predicting upcoming words


Claude Shannon

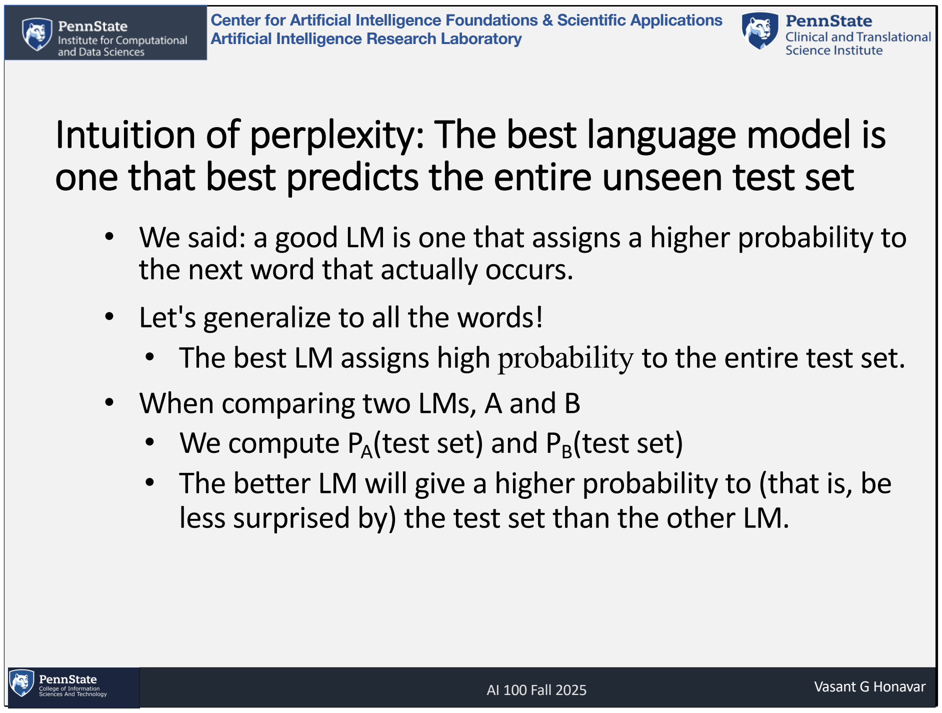
The Shannon Game: How well can we predict the next word?

- Once upon a _____
- That is a picture of a _____
- For breakfast I ate my usual _____

time	0.9
dream	0.03
midnight	0.02
…	
and	1e-100

Unigrams are terrible at this game (Why?)

A good LM is one that assigns a higher probability to the next word that actually occurs



Intuition of perplexity: The best language model is one that best predicts the entire unseen test set

- We said: a good LM is one that assigns a higher probability to the next word that actually occurs.
- Let's generalize to all the words!
  - The best LM assigns high probability to the entire test set.
- When comparing two LMs, A and B
  - We compute P_A(test set) and P_B(test set)
  - The better LM will give a higher probability to (that is, be less surprised by) the test set than the other LM.

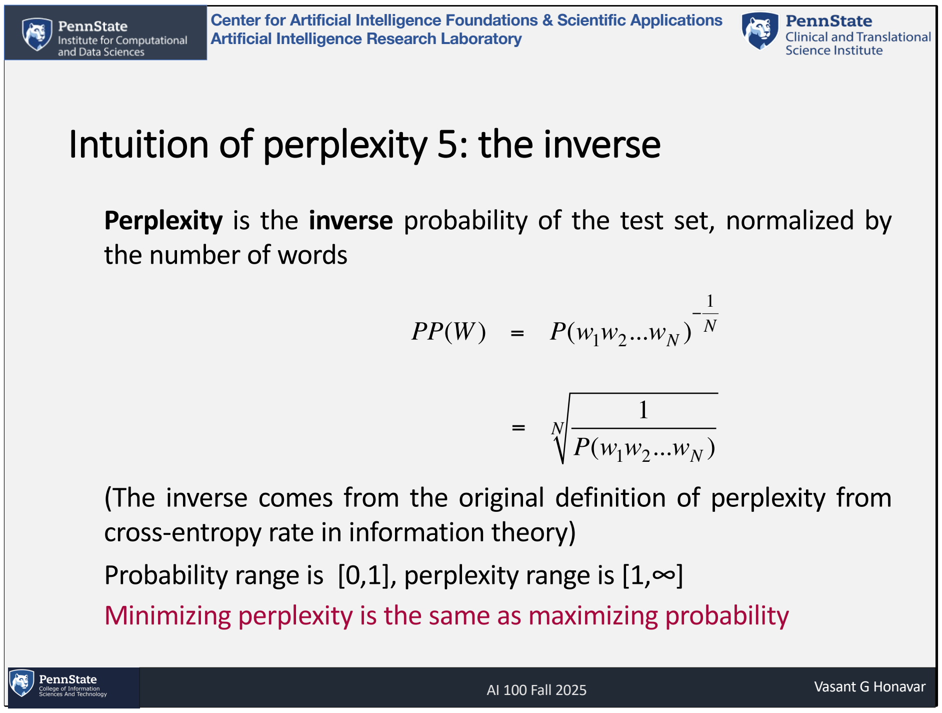
Let's generalize this idea that a good LM assigns a higher probability to the next word that actually occurs, to say that A good LM assigns high probability to the entire test set. Let's just think of the test set as if it was one very long sentence. So when comparing to LMs, A, and B, we ask each to tell us the probability of the entire testset. Now the better LM is the one that gives a higher probability to the test set, or in other words is less surprised by the test set than the other LM.

Intuition of perplexity 4: Use perplexity instead of raw probability

- Probability depends on size of test set
  - Probability gets smaller the longer the text
  - Better: a metric that is per-word, normalized by length
- Perplexity is the inverse probability of the test set, normalized by the number of words

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

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



Intuition of perplexity 5: the inverse

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

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

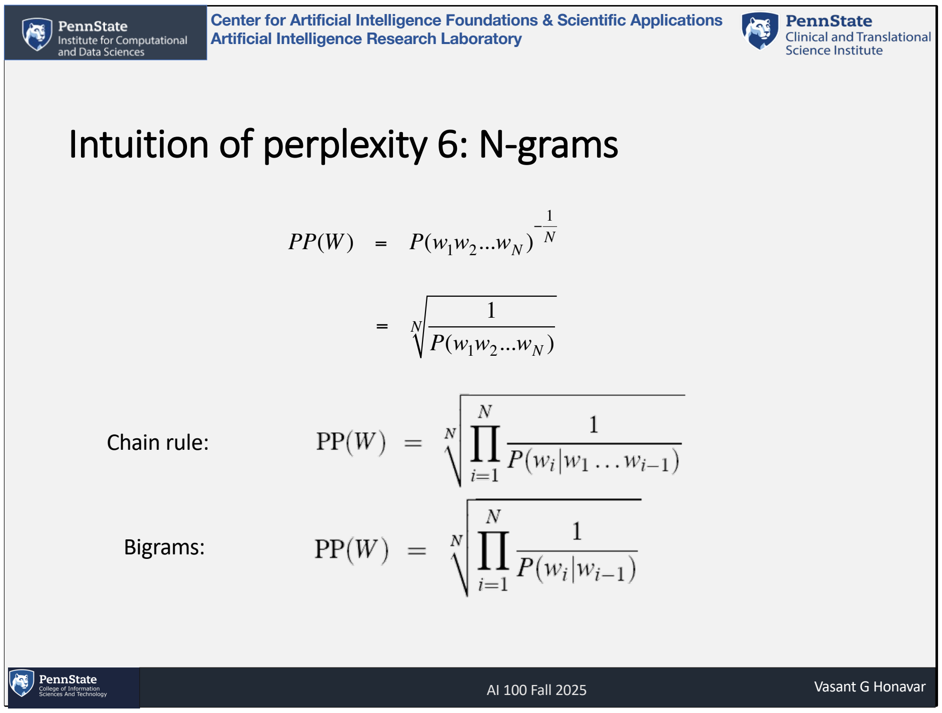
$$= \sqrt[N]{\frac{1}{P(w_1 w_2 ... w_N)}}$$

(The inverse comes from the original definition of perplexity from cross-entropy rate in information theory)

Probability range is [0,1], perplexity range is [1,∞]

Minimizing perplexity is the same as maximizing probability

Notice that I slipped something in there. Perplexity isjust just the probability of the test set normalized by the number of words. It's the inverse probability. There is this little minus-sign here, or a 1/x here. This inverse turns out to come from part of the original definition of perplexity from the idea of cross-entropy rate in information theory. We won't go into that here, but you can look at it in the textbook). We'll just say that because of that inverse, minimizing perplexity is the same as maximizing probability. So the better LM has a higher probability but a lower perplexity on the test set.



Intuition of perplexity 6: N-grams

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

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

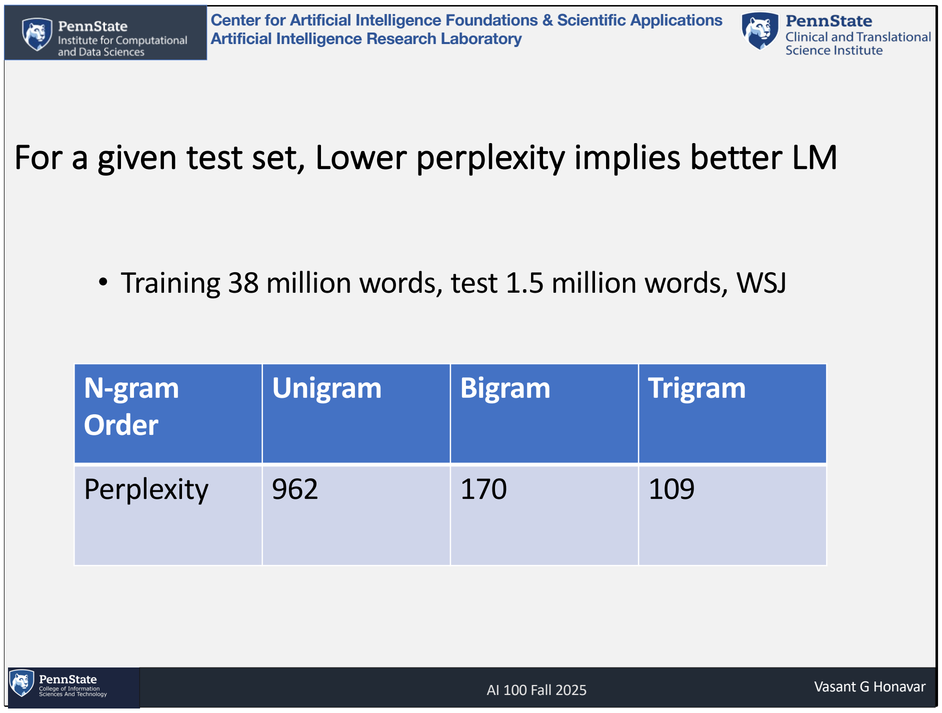
Chain rule:

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

Bigrams:

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

Let's see perplexity with n-gram language models. We can use the chain rule to turn the probability of the entire test set into N separate probabilities. Now we can make the n-gram assumption, such as the bigram simplification, to help us compute each of these individual probabilities.


PennState Institute for Computational and Data Sciences

Center for Artificial Intelligence Foundations & Scientific Applications
Artificial Intelligence Research Laboratory

PennState Clinical and Translational Science Institute

For a given test set, Lower perplexity implies better LM

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

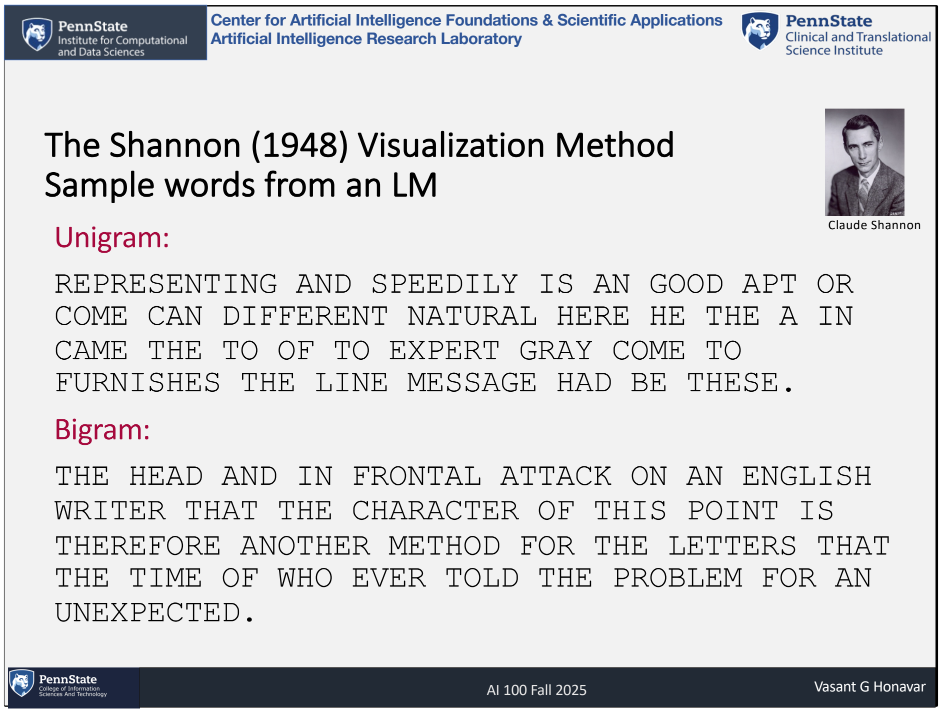
N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

PennState College of Information Sciences And Technology

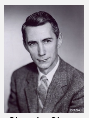
AI 100 Fall 2025

Vasant G Honavar

So if we hold the test set constant, the better, more predictive the language model, the lower the perplexity will be. Here we can see this with some text from the US Wall Street Journal newspaper text corpus. A unigram LM has a perplexity of 962, a bigram of 170, and a trigram of 109. The better the model, the higher the probability it assigns to the test set, and since perplexity goes inversely with probability, lower the perplexity.


PennState Institute for Computational and Data Sciences
Center for Artificial Intelligence Foundations & Scientific Applications
Artificial Intelligence Research Laboratory
PennState Clinical and Translational Science Institute

The Shannon (1948) Visualization Method
Sample words from an LM


Claude Shannon

Unigram:

```
REPRESENTING AND SPEEDILY IS AN GOOD APT OR
COME CAN DIFFERENT NATURAL HERE HE THE A IN
CAME THE TO OF TO EXPERT GRAY COME TO
FURNISHES THE LINE MESSAGE HAD BE THESE.
```

Bigram:

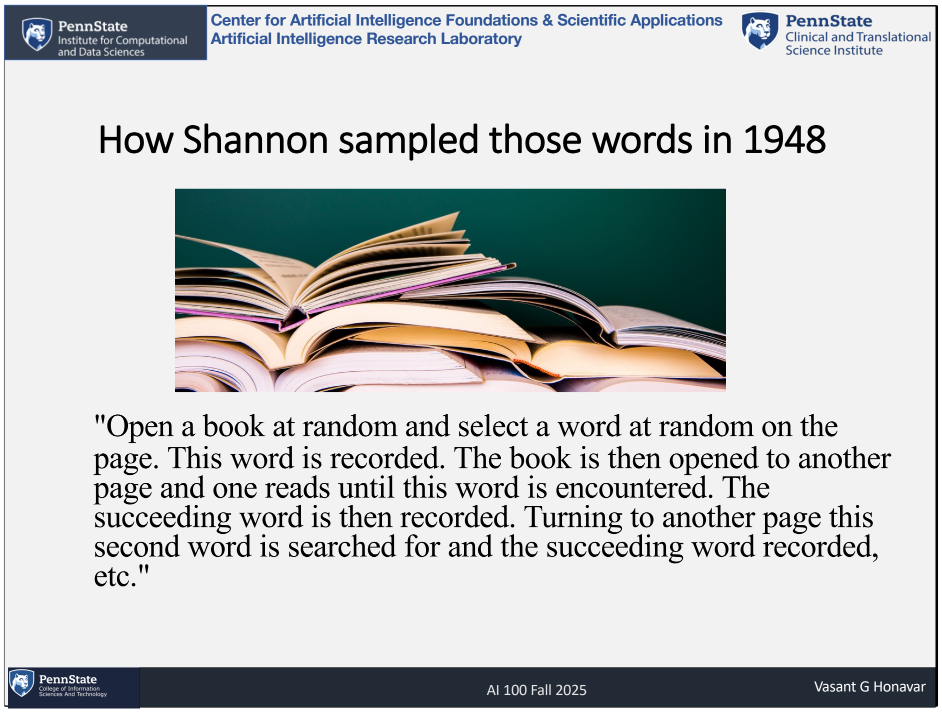
```
THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH
WRITER THAT THE CHARACTER OF THIS POINT IS
THEREFORE ANOTHER METHOD FOR THE LETTERS THAT
THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN
UNEXPECTED.
```

PennState College of Information Sciences And Technology

AI 100 Fall 2025

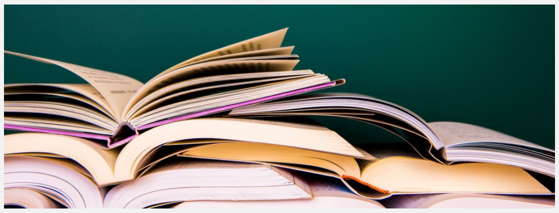
Vasant G Honavar

One important way to visualize what kind of knowledge a language model embodies is to sample from it. Sampling from a distribution means to choose random points according to their likelihood. Sampling from a language model means to generate sentence according to their likelihoods as defined by the model. Thus we are more likely to generate sentences that the model thinks have a high probability and less likely to generate sentences that the model thinks have a low probability. Here are some sentences generated by Claude Shannon in 1948 from a unigram and a bigram model. You can see that the unigram model is just word salad, but the bigram model, while still pretty random, it starting to look a bit more coherent in its sequences, at least locally.



Center for Artificial Intelligence Foundations & Scientific Applications
Artificial Intelligence Research Laboratory

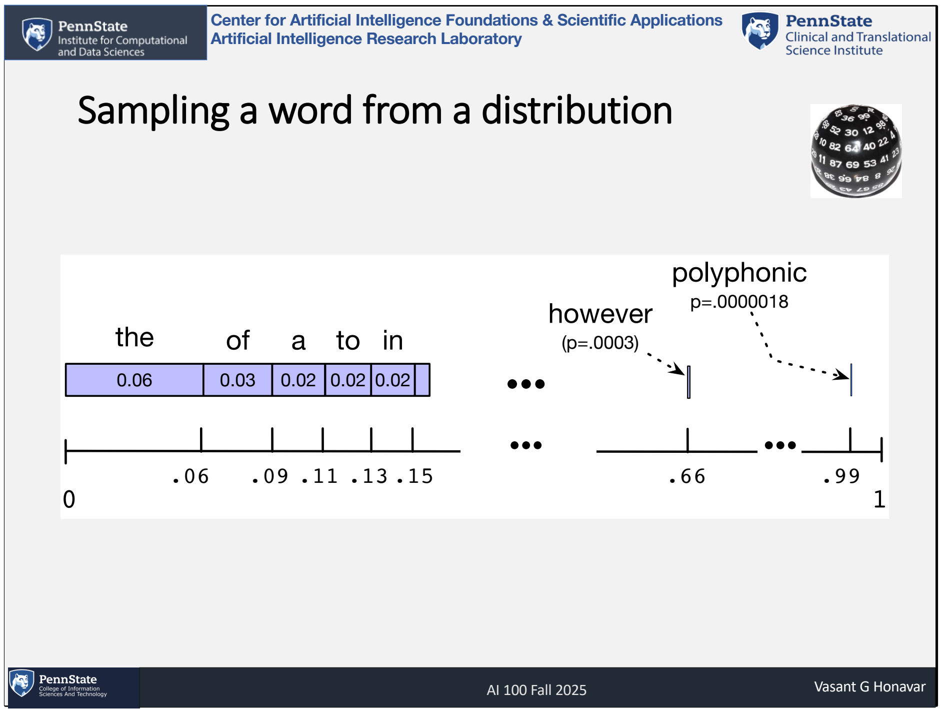
How Shannon sampled those words in 1948



"Open a book at random and select a word at random on the page. This word is recorded. The book is then opened to another page and one reads until this word is encountered. The succeeding word is then recorded. Turning to another page this second word is searched for and the succeeding word recorded, etc."

AI 100 Fall 2025 Vasant G Honavar

Here's how Shannon sampled those words in 1948, although describing the process for letters rather than words. How could we implement the same process computationally given an n-gram language model?

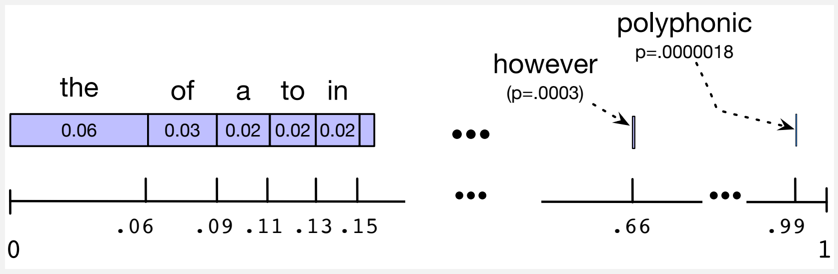


Center for Artificial Intelligence Foundations & Scientific Applications
Artificial Intelligence Research Laboratory

PennState Institute for Computational and Data Sciences

PennState Clinical and Translational Science Institute

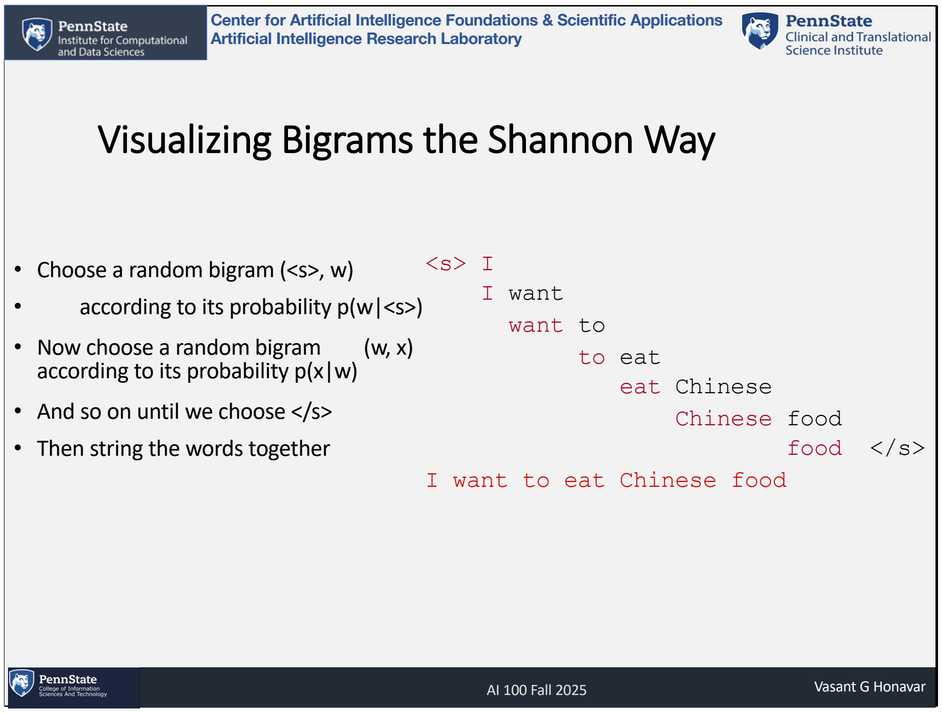
Sampling a word from a distribution



AI 100 Fall 2025

Vasant G Honavar

It’s simplest to visualize how this works for the unigram case. Imagine all the words of the English language covering the probability space between 0 and 1, each word covering an interval proportional to its frequency. We choose a random value between 0 and 1, find that point on the probability line, and print the word whose interval includes this chosen value. So let's suppose we choose a random number, it's 0.12, we look on our number line, and that's the word "to", so we generate "to" as the next word. We continue choosing random numbers and generating words until we randomly generate the end-of-sentence token.

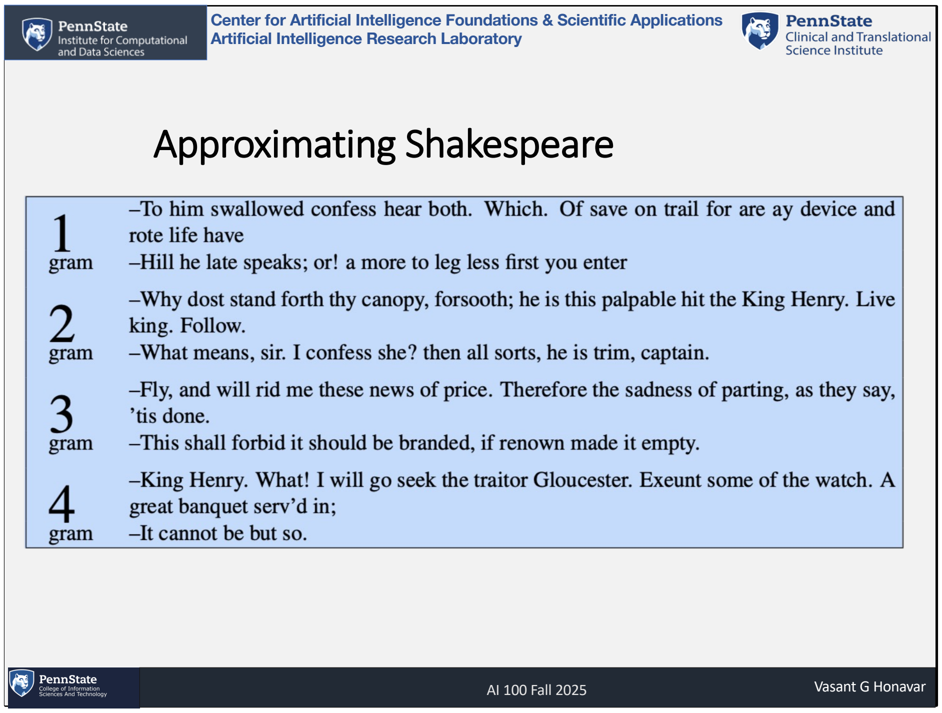


Visualizing Bigrams the Shannon Way

- Choose a random bigram (<s>, w)
- according to its probability p(w|<s>)
- Now choose a random bigram (w, x) according to its probability p(x|w)
- And so on until we choose </s>
- Then string the words together

```
<s> I
    I want
      want to
           to eat
              eat Chinese
                  Chinese food
                          food   </s>
I want to eat Chinese food
```

We can extend this idea to n-grams. For example for the bigram case, we first choose a random bigram whose first token is the start-of-sentence token, according to its probability


PennState Institute for Computational and Data Sciences

Center for Artificial Intelligence Foundations & Scientific Applications
Artificial Intelligence Research Laboratory

PennState Clinical and Translational Science Institute

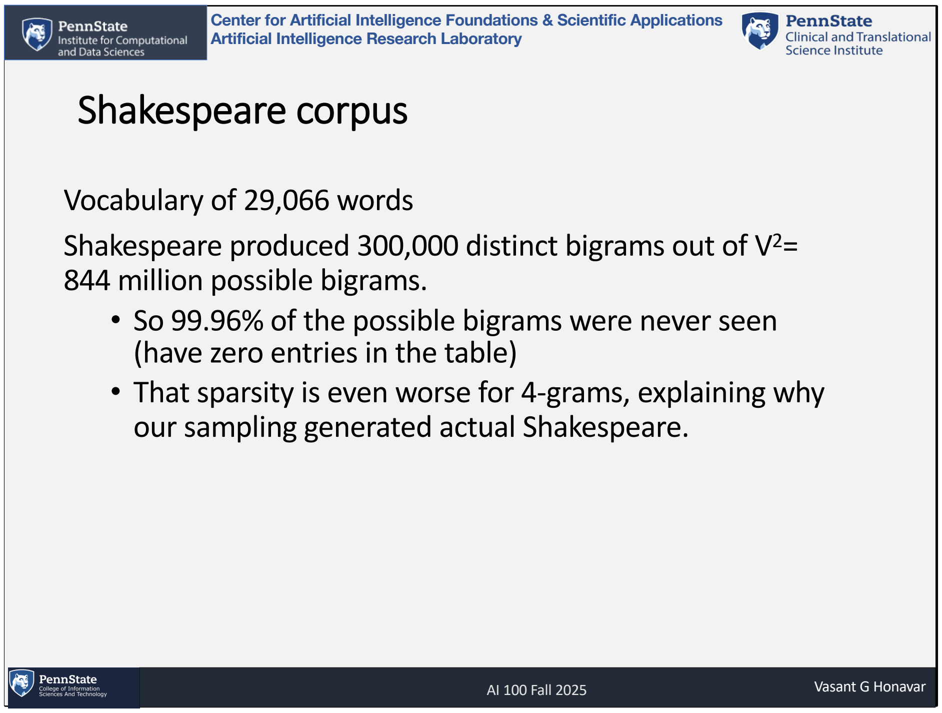
Approximating Shakespeare

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

PennState College of Information Sciences And Technology AI 100 Fall 2025 Vasant G Honavar

This figure shows some random sentences generated from unigram, bigram, trigram, and 4- gram models trained on the works of William Shakespeare.

The longer the context, the more coherent the sentences. In these unigram sentences, there's no coherent relation between words. The bigram sentences have some local word-to-word coherence, and sensible punctuation. The 3 and 4-gram sentences are beginning to look a lot like Shakespeare. In fact, they look a little too much like Shakespeare. The words *It cannot be but so* are directly from *King John*.


Center for Artificial Intelligence Foundations & Scientific Applications
Artificial Intelligence Research Laboratory

Shakespeare corpus

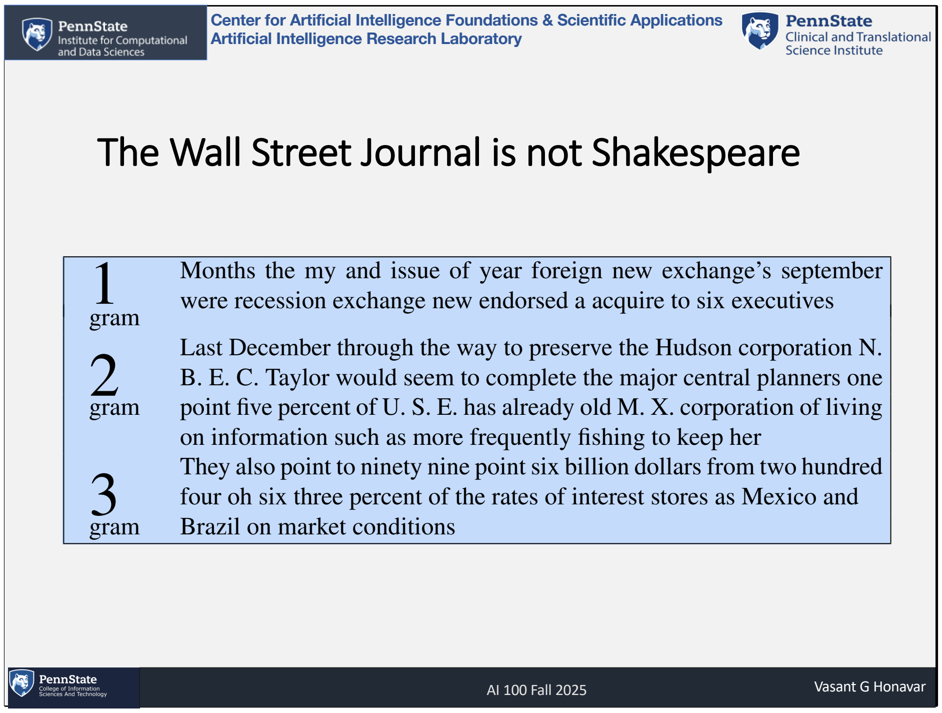
Vocabulary of 29,066 words

Shakespeare produced 300,000 distinct bigrams out of V^2= 844 million possible bigrams.

- So 99.96% of the possible bigrams were never seen (have zero entries in the table)
- That sparsity is even worse for 4-grams, explaining why our sampling generated actual Shakespeare.

AI 100 Fall 2025 Vasant G Honavar

Just choosing one author, Shakespeare, makes for a pretty small corpus; under a million words from a vocabulary of 29,000 words. The vocab 29,000 squared is somewhat under a billion. So that means there are a billion possible bigrams whose counts have to come from only a million word corpus, and 10^17 possible 4-grams. So when training on any corpus, the vast majority of all the n-gram counts will be zero!

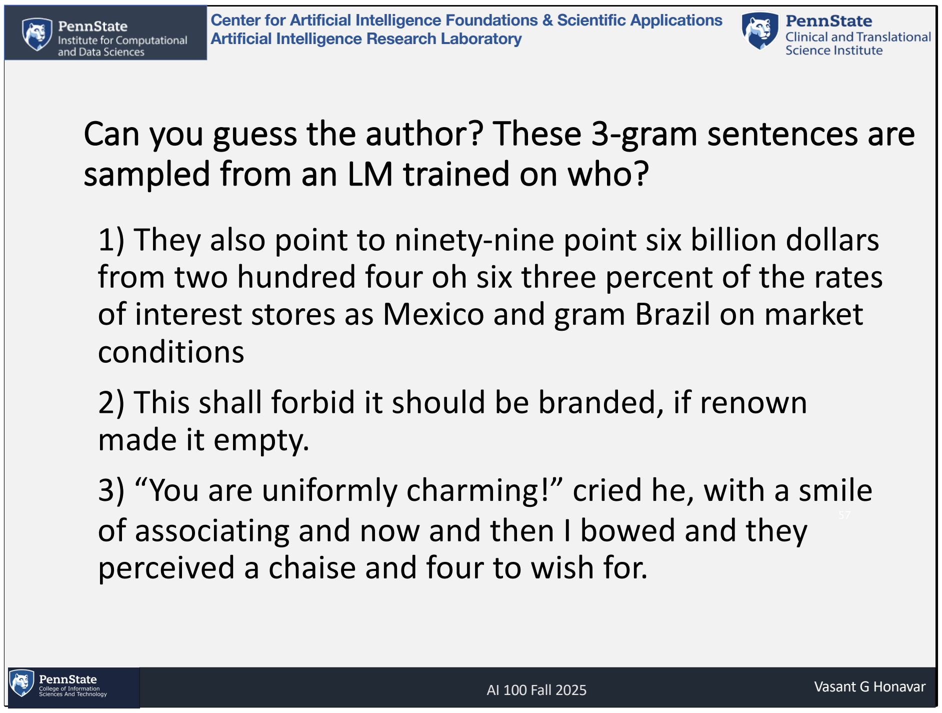

Center for Artificial Intelligence Foundations & Scientific Applications
Artificial Intelligence Research Laboratory

The Wall Street Journal is not Shakespeare

1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

AI 100 Fall 2025 Vasant G Honavar

To get an idea of the dependence of a grammar on its training set, let's look at an n-gram grammar trained on a completely different corpus: the *Wall Street Journal* (WSJ) newspaper. Shakespeare and the *Wall Street Journal* are both English, so we might expect some overlap between our n-grams. There is no overlap even in small phrases, let alone entire sentences. Statistical models are use- less as predictors when the training sets and the test sets are as different as Shakespeare and WSJ.

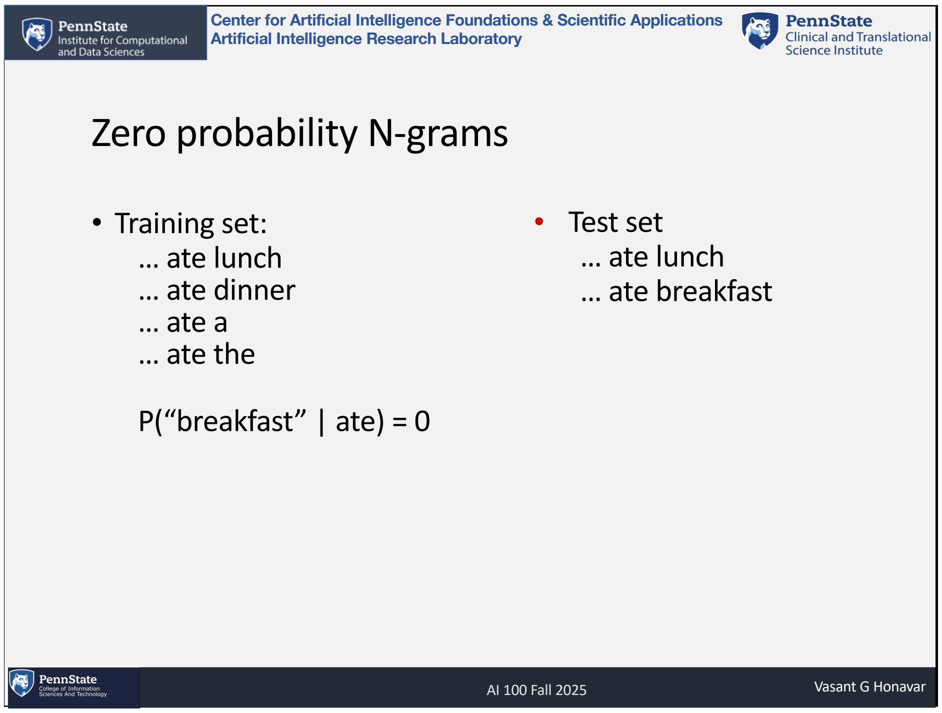


Can you guess the author? These 3-gram sentences are sampled from an LM trained on who?

1) They also point to ninety-nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and gram Brazil on market conditions

2) This shall forbid it should be branded, if renown made it empty.

3) “You are uniformly charming!” cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for.


Center for Artificial Intelligence Foundations & Scientific Applications
Artificial Intelligence Research Laboratory

Zero probability N-grams

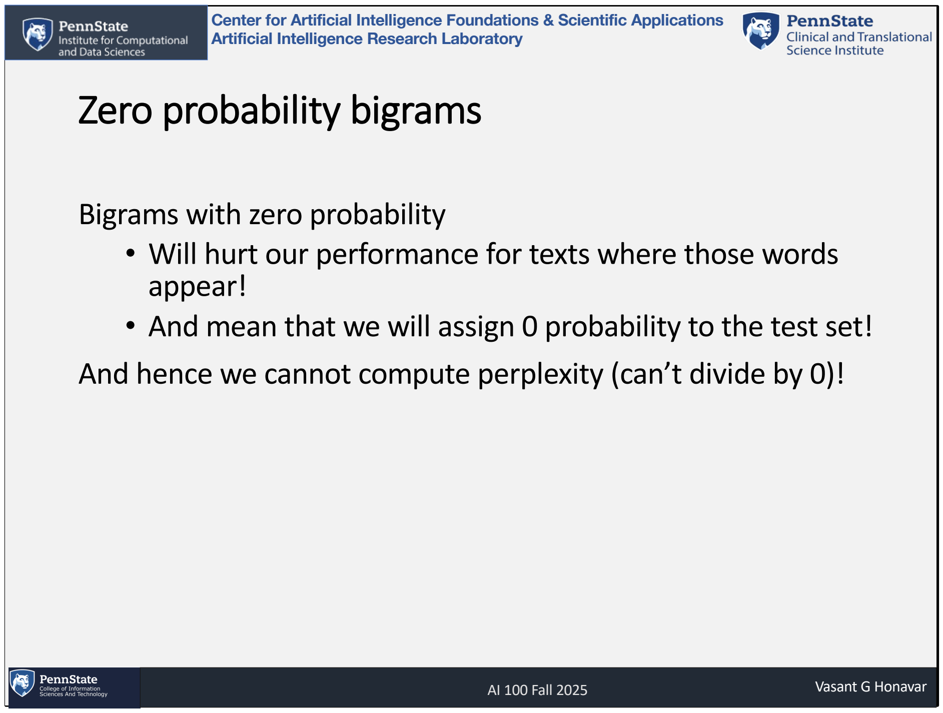
- Training set:
  … ate lunch
  … ate dinner
  … ate a
  … ate the

  P(“breakfast” | ate) = 0

- Test set
  … ate lunch
  … ate breakfast

AI 100 Fall 2025 Vasant G Honavar

For example, consider a training set that has the phrases "ate lunch", "ate dinner", "ate a", and "ate the", but happens not to have had the phrase "ate breakfast". The P("breakfas|ate) in the test set will be 0!



Zero probability bigrams

Bigrams with zero probability

- Will hurt our performance for texts where those words appear!
- And mean that we will assign 0 probability to the test set!

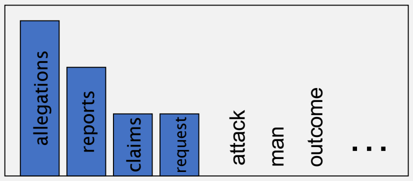
And hence we cannot compute perplexity (can't divide by 0)!

These zeros are a problem for two reasons. First, they mean we are underestimating the probability of all sorts of words that might occur, which will hurt the performance of any application we want to run on this data. Second, if the probability of any word in the test set is 0, the entire probability of the test set is 0. And perplexity is based on the inverse probability of the est set. Thus if some words have zero probability, we can't compute perplexity at all, since we can't divide by 0! The solution to this problem is called smoothing, and we'll see it in the next lecture!

The intuition of smoothing

- When we have sparse statistics:

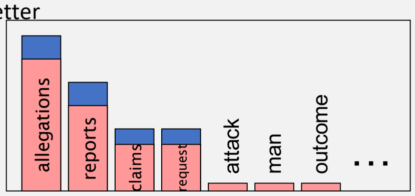
P(w | denied the)
3 allegations
2 reports
1 claims
1 request
7 total

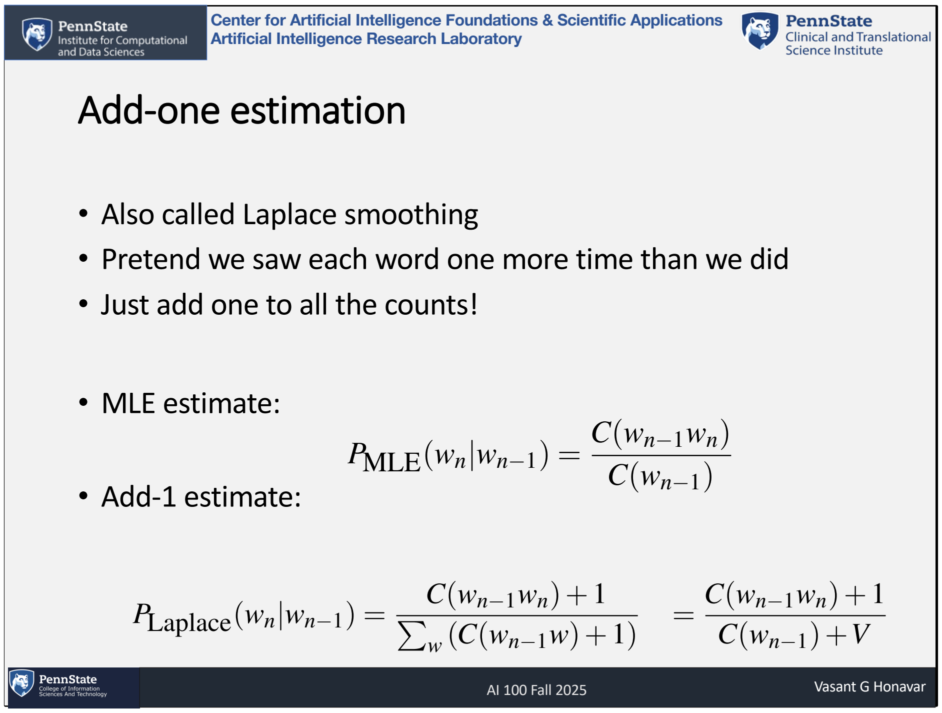


- Borrow probability mass to generalize better

P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
3 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:

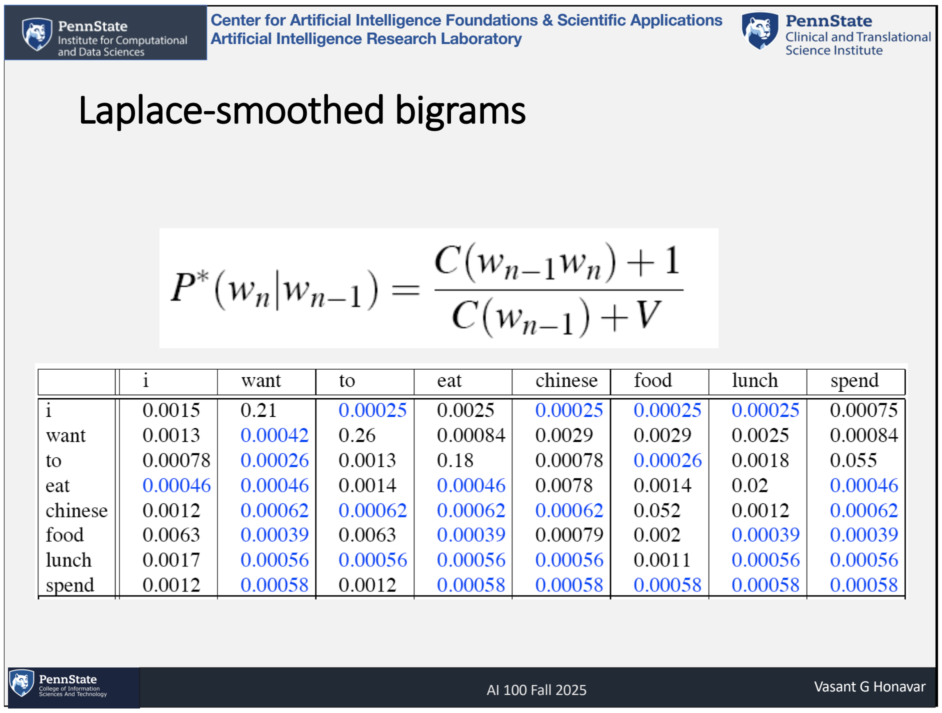
$$P_{\text{MLE}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

- Add-1 estimate:

$$P_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{\sum_w (C(w_{n-1}w)+1)} = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

Berkeley Restaurant Corpus
Laplace smoothed bigram counts

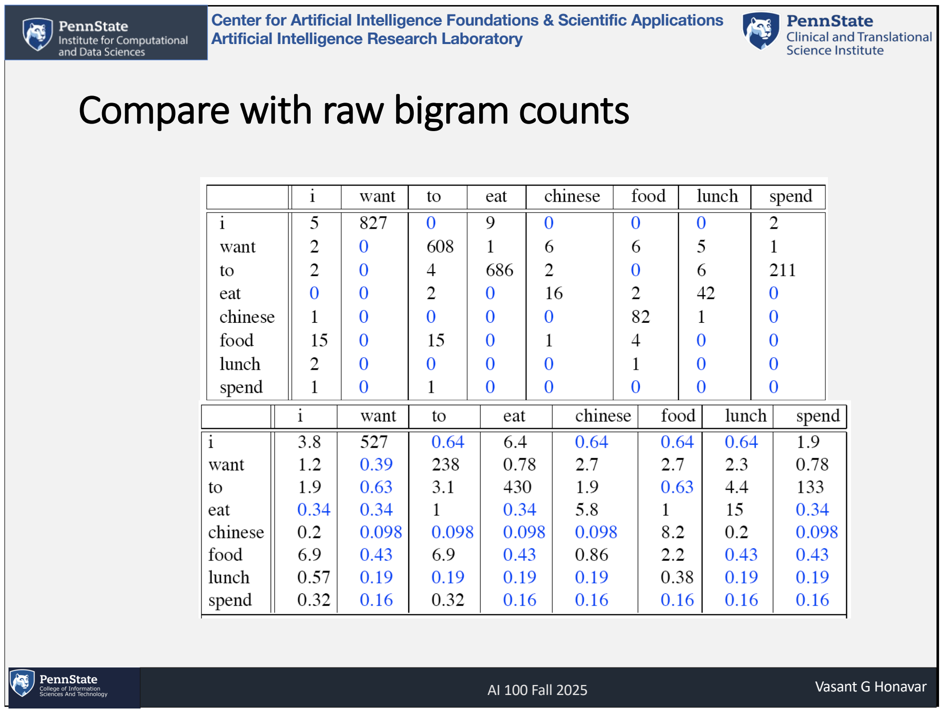
	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1



Laplace-smoothed bigrams

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

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058



Compare with raw bigram counts

	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

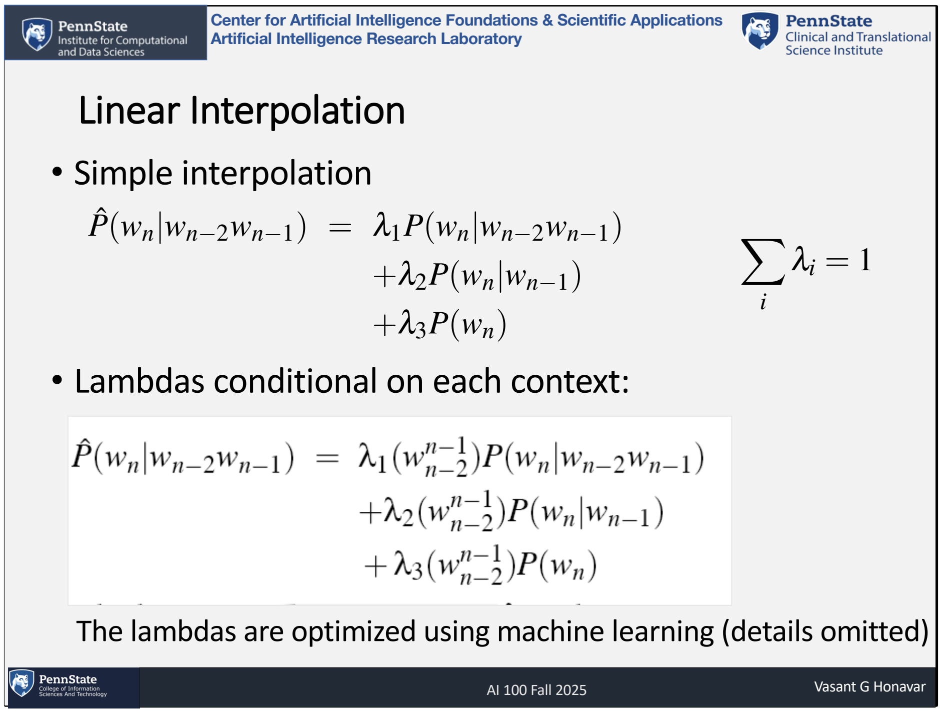
	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

C(want to) went from 608 to 238,

P(to|want) from .66 to .26!

Discount d= c*/c

d for "chinese food" =.10!!! A 10x reduction



Linear Interpolation

- Simple interpolation

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ &+\lambda_2 P(w_n|w_{n-1}) \\ &+\lambda_3 P(w_n)\end{aligned}$$

$$\sum_i \lambda_i = 1$$

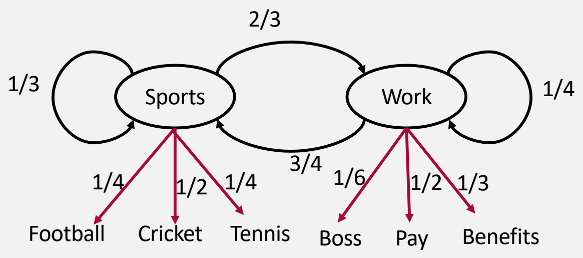
- Lambdas conditional on each context:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1(w_{n-2}^{n-1}) P(w_n|w_{n-2}w_{n-1}) \\ &+\lambda_2(w_{n-2}^{n-1}) P(w_n|w_{n-1}) \\ &+\lambda_3(w_{n-2}^{n-1}) P(w_n)\end{aligned}$$

The lambdas are optimized using machine learning (details omitted)

Hidden Markov Language Models

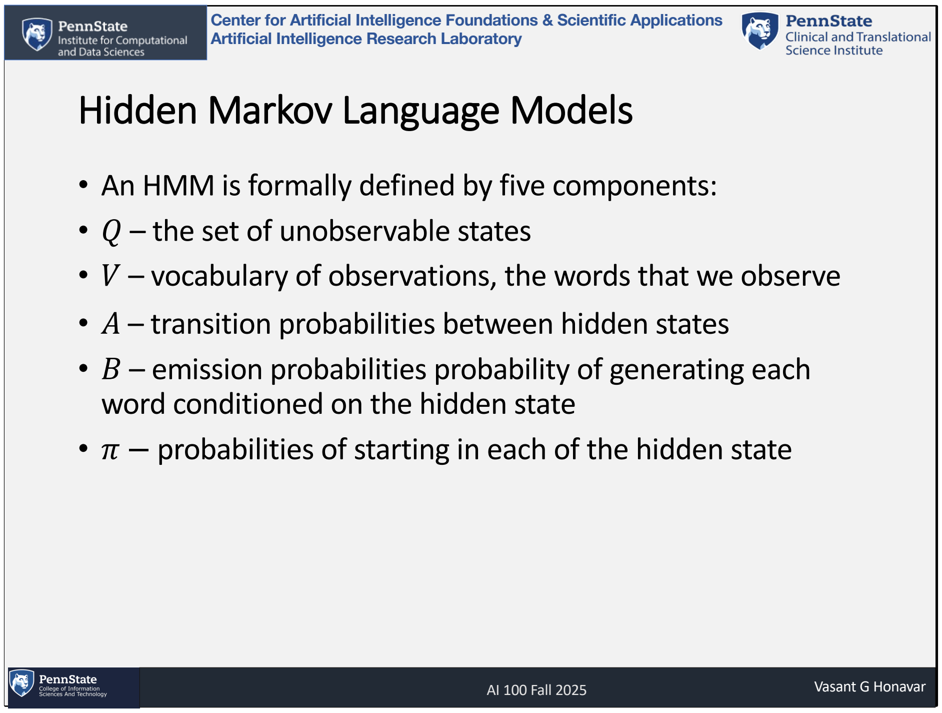
- Suppose probability of the next word depends on context
- Suppose context is not observable
- Hidden states represent topics covered in a conversation
- Transition between states obeys the Markov assumption
- Probability of the generated word depends on the hidden state



For simplicity, the start and end tokens are not shown

Hidden Markov Language Models

- Hidden Markov Models are probabilistic models for sequential data where the underlying states are "hidden" from the observer
- In language, we observe the words (output) but not the underlying context or "intent" (hidden states) that generated them
- Goal: To model the joint probability of a sequence of words and their corresponding hidden states



Hidden Markov Language Models

- An HMM is formally defined by five components:
- Q – the set of unobservable states
- V – vocabulary of observations, the words that we observe
- A – transition probabilities between hidden states
- B – emission probabilities probability of generating each word conditioned on the hidden state
- π — probabilities of starting in each of the hidden state

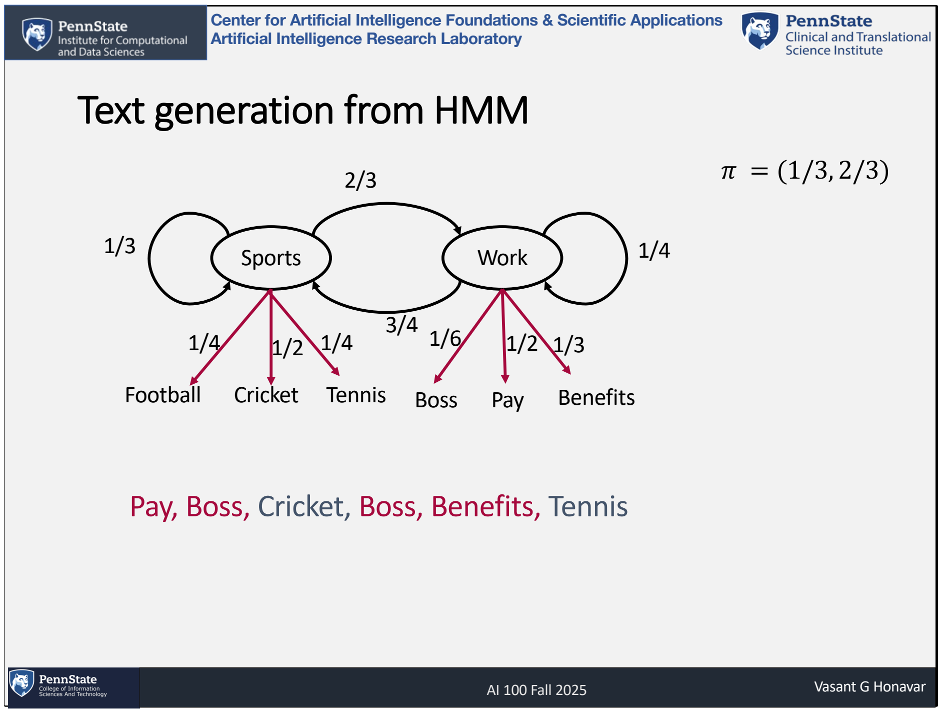
Center for Artificial Intelligence Foundations & Scientific Applications
Artificial Intelligence Research Laboratory

Language generation involves sampling from an HMM

- Start: The model begins in an initial hidden state based on π
- Emit: The current hidden state generates a word by sampling from the emission probability distribution B.
- Transition: The model moves to the next hidden state according to the transition probabilities A , which depends only on the current state (Markov property).
- Repeat: Steps 2 and 3 are repeated until a complete sequence (e.g., a sentence) is generated.

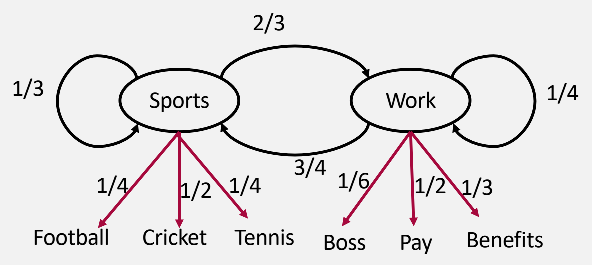
Hidden Markov Language Models

- Probabilistic Framework: HMMs define a probability distribution over all possible word sequences, allowing them to score the likelihood of a generated sentence.
- Decoding (Inference): Given an observed sentence, HMMs can infer the most likely sequence of hidden states that produced it using algorithms like the Viterbi algorithm.
- Parameter Estimation (Learning): The model's parameters A, B, π are learned from a large corpus of text



Text generation from HMM

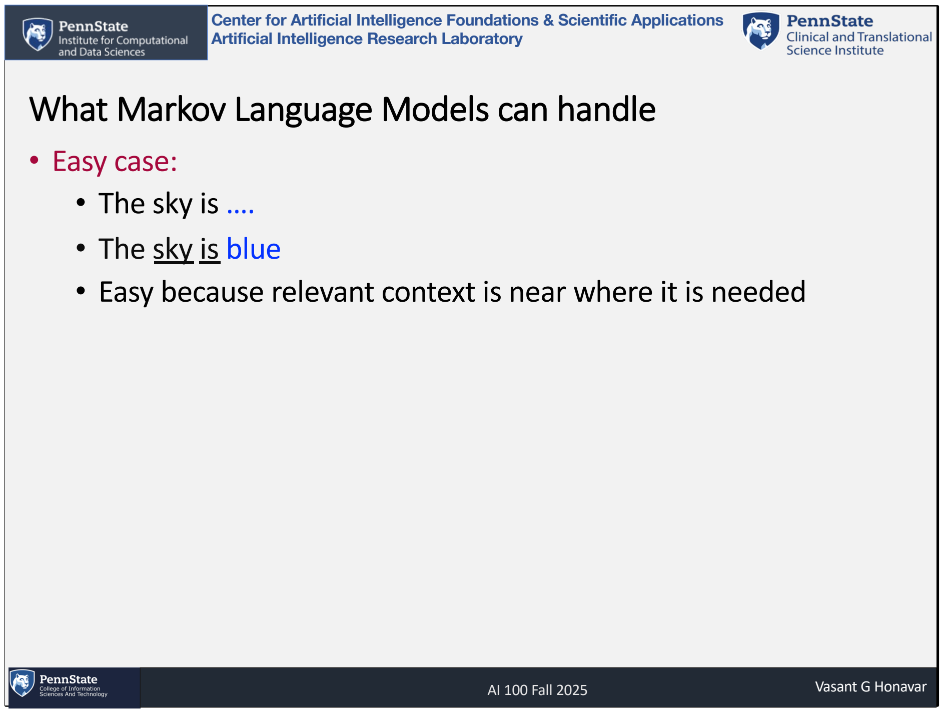
$\pi = (1/3, 2/3)$



Pay, Boss, Cricket, Boss, Benefits, Tennis

Hidden Markov Language Models

- Early Successes: HMMs were foundational in Natural Language Processing (NLP) for tasks like part-of-speech tagging and topic modeling
- Language Generation: HMM can produce coherent, simple sentences or sequences that exhibit learned patterns.
- Limitations:
  - HMMs rely on the Markov assumption that the next state depends only on the current state, ignoring broader context or long-range dependencies within a text
  - This limits their ability to generate complex, long, and contextually rich human language compared to modern deep learning models (like LLMs)



What Markov Language Models can handle

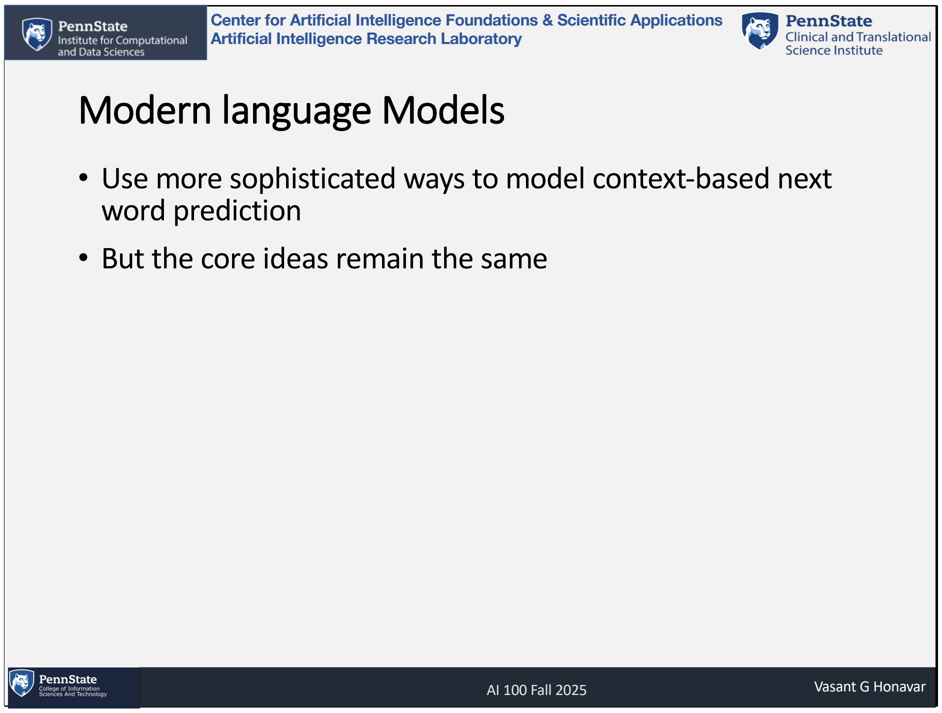
- Easy case:
  - The sky is ....
  - The sky is blue
  - Easy because relevant context is near where it is needed

What Markov Language Models can't handle

- Difficult case:
  - I was born in India. I studied Electrical Engineering as an undergraduate. I came to the United States for graduate studies. After a short stint at Drexel where I got my masters degree in Electrical and Computer Engineering, I moved to the University of Wisconsin for my PhD. My PhD research was on neural networks. I play tennis. I speak fluent ….
  - Nearby context suggests the next word must be a language
  - But which language?
  - The relevant context for accurately predicting the next word is very far away, well beyond the reach of the Markov assumption

Limitations of Markov and Hidden Markov language models

- Data sparsity. The number of possible n-grams grows exponentially with n, and even very large corpora cannot cover them all.
- Lack of generalization. n-gram models treat each word as an atomic token and do not generalize across similar words (cat and dog) or share knowledge across contexts
- Failure to model long-range dependencies. The dependencies modeled by the n–gram models are limited to $(n-1)$ words, so long-range dependencies can be hard to capture



Modern language Models

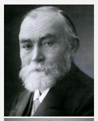
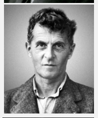
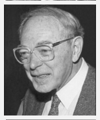
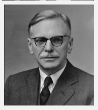
- Use more sophisticated ways to model context-based next word prediction
- But the core ideas remain the same

Neural language models

- Neural language models represent a major shift from traditional statistical $n-$gram models.
- Neural language models (NLM) learn distributed representations of words and contexts using neural networks.
- The resulting distributed representations enable NLM to encode
  - semantic similarity between words and between contexts
- Distributed representations of words, combined with the ability to handle longer contexts, neural language models can produce substantially more accurate language models compared to traditional $n-$gram models

Distributional semantics hypothesis

- Never ask for the meaning of a word in isolation, but only in the context of a sentence (Gottlob Frege, 1884)
- For a large class of cases… the meaning of a word is its use in the language (Ludwig Wittgenstein, 1953)
- Words that occur in similar contexts tend to have similar meaning (Zellig Harris, 1954)
- You shall know a word by the company it keeps (John R Firth, 1957)

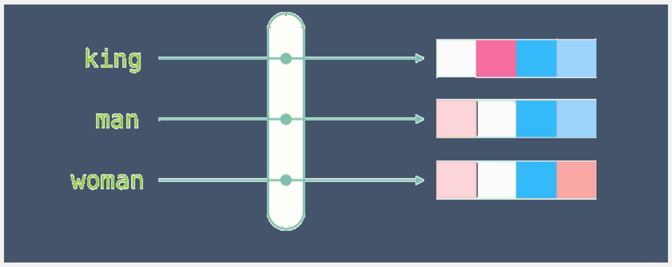





The meaning of a word can be understood by analyzing the contexts in which it appears and from its pattern of co-occurrence with other words.

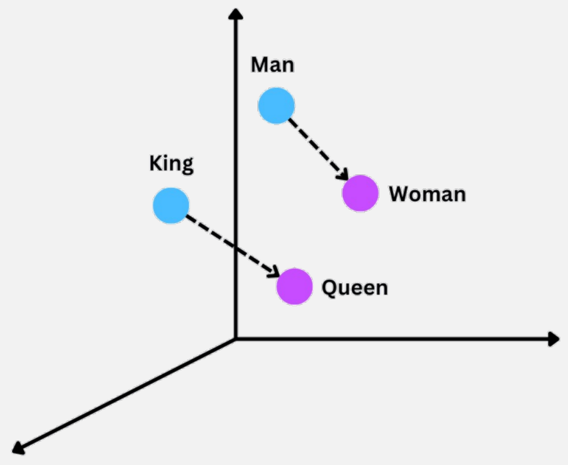
Distributional semantics to word embeddings

Word embeddings

- Represent words as low-dimensional vectors whose geometry reflects the semantic and syntactic features of the word (or language token)
- Ensure that semantically similar words have similar vector representations

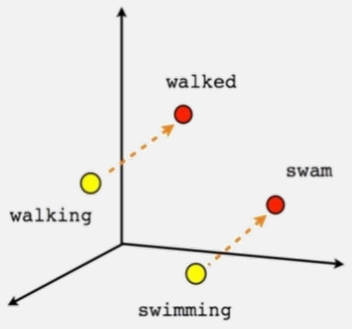


Word embeddings preserve semantic relations

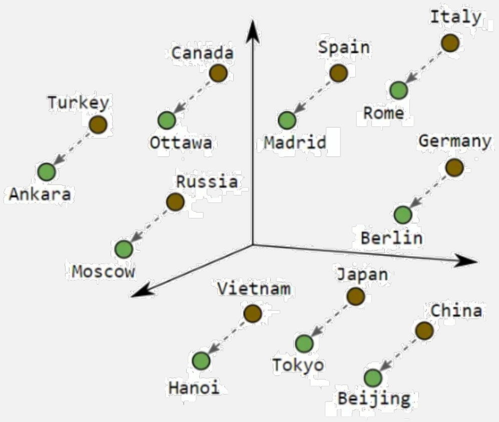


Gender

Word embeddings preserve semantic relations

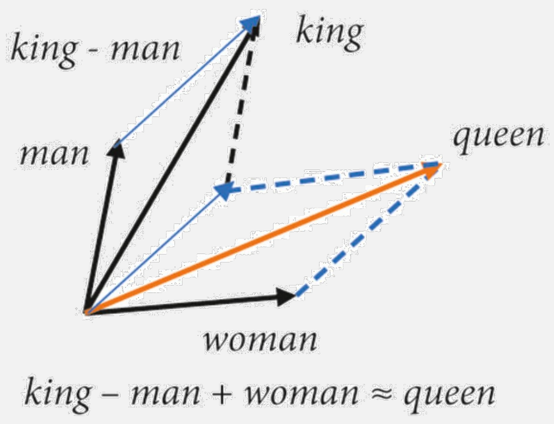


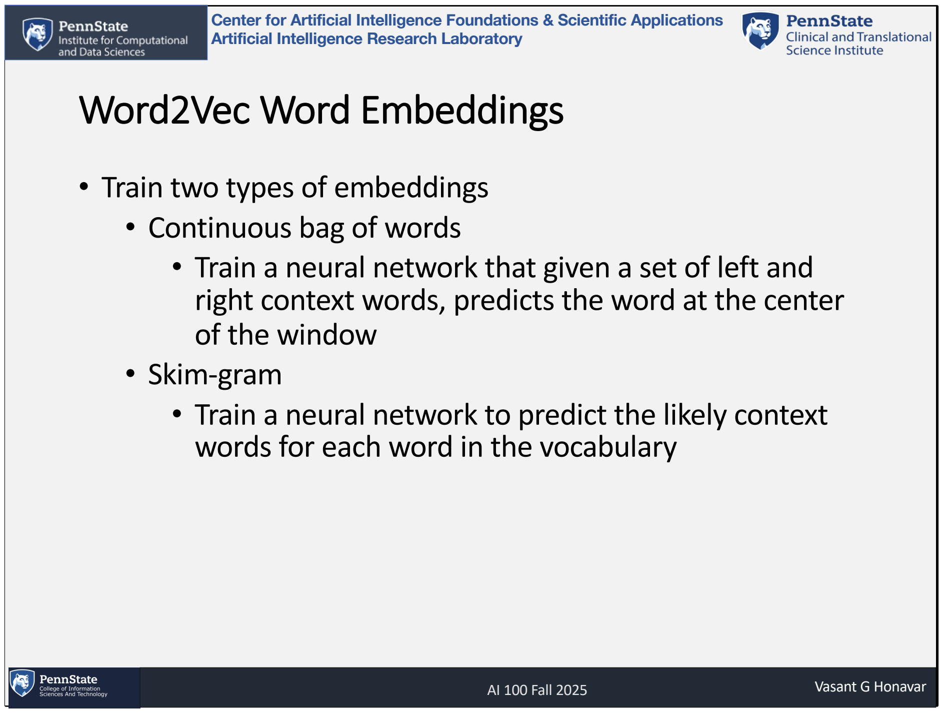
Word embeddings preserve semantic relations



Country-capital

Magic of Word embeddings



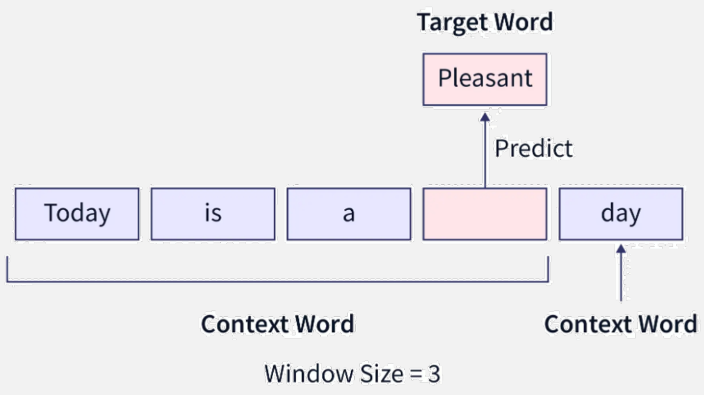


Word2Vec Word Embeddings

- Train two types of embeddings
  - Continuous bag of words
    - Train a neural network that given a set of left and right context words, predicts the word at the center of the window
  - Skim-gram
    - Train a neural network to predict the likely context words for each word in the vocabulary

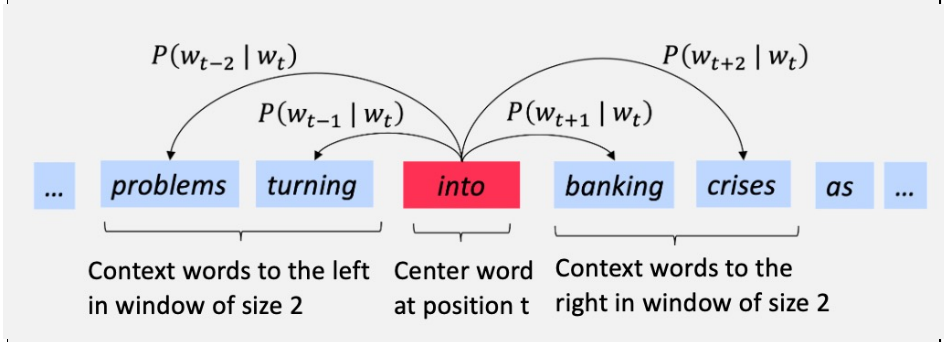
Word2Vec Continuous Bag of Words (CBOW)

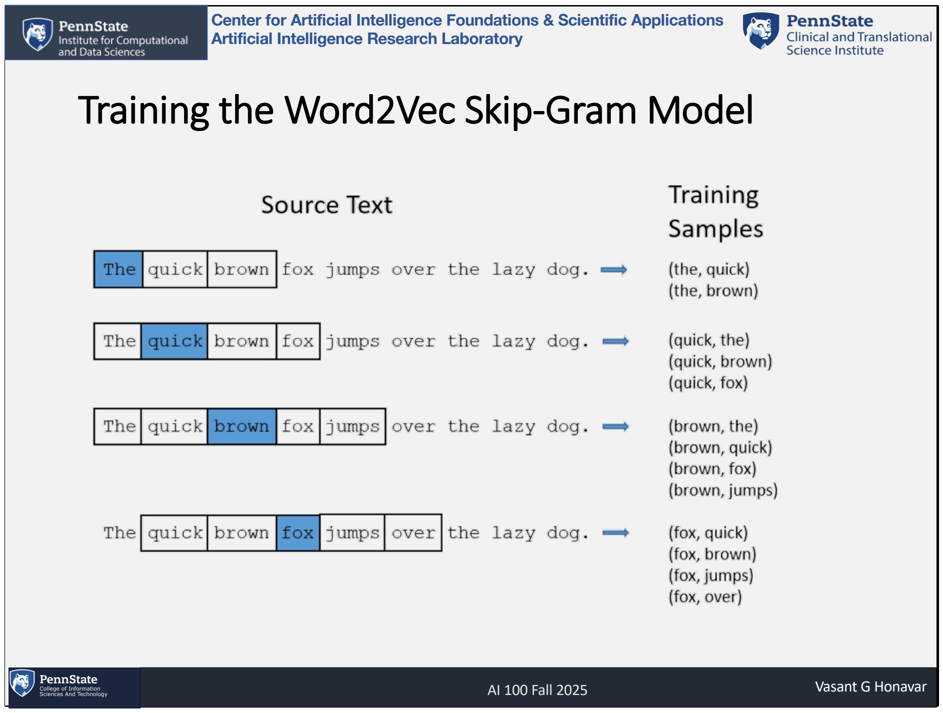
- CBOW is trained to predict the center word from surrounding context



Word2Vec Skip-Gram Model

- Skip-gram model is trained to predict the probability of context words given the center word





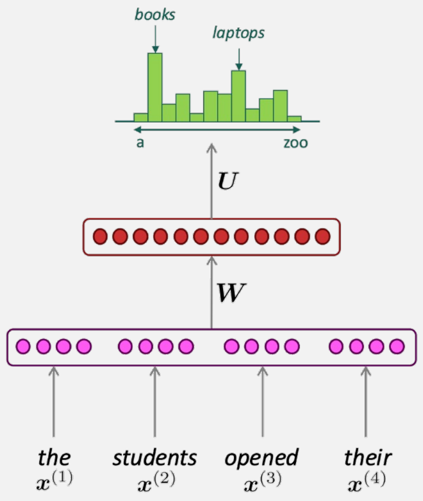
Training the Word2Vec Skip-Gram Model

Source Text		Training Samples
[The] [quick] [brown] fox jumps over the lazy dog.	⟹	(the, quick) (the, brown)
[The] **[quick]** [brown] [fox] jumps over the lazy dog.	⟹	(quick, the) (quick, brown) (quick, fox)
[The] [quick] **[brown]** [fox] [jumps] over the lazy dog.	⟹	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The [quick] [brown] **[fox]** [jumps] [over] the lazy dog.	⟹	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

Kinds of embeddings

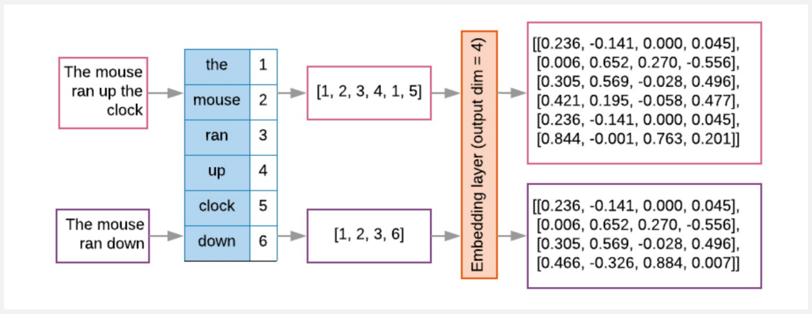
- The way we train the word embeddings depends on the application
- For next word prediction, we only use left context
- For predicting the missing word, we use both left and right context

Neural language model with fixed context size



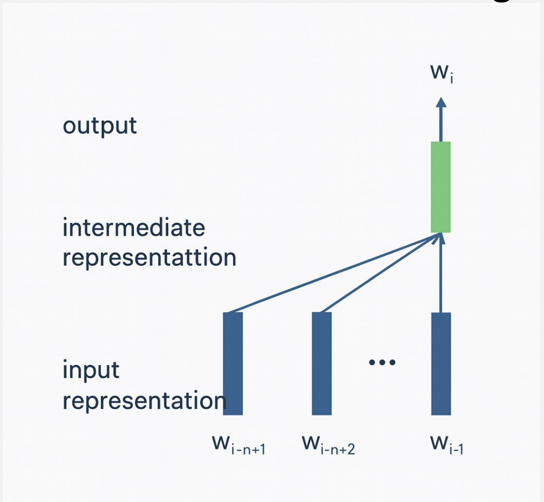
- Feed-forward neural language model is trained to predict the next word from embeddings of context

Data flow in a neural language model

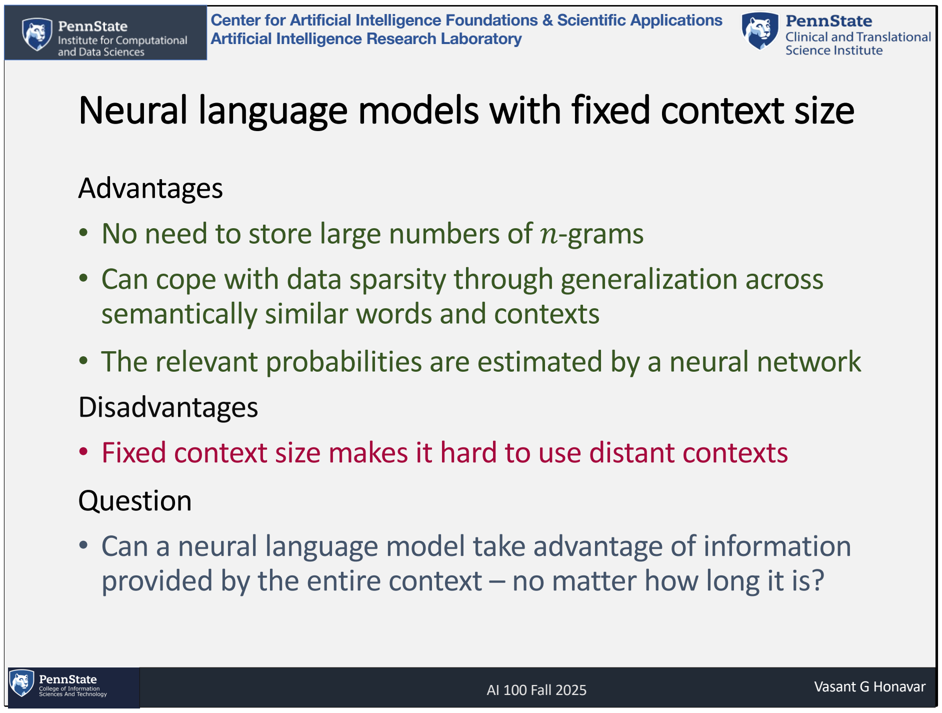


- The words are mapped to embeddings before being input to the neural network

Representation of context in a neural language model



Source: Hang Li. 2022. ACM 65, 7, 56–63. https://doi.org/10.1145/3490443



Neural language models with fixed context size

Advantages

- No need to store large numbers of n-grams
- Can cope with data sparsity through generalization across semantically similar words and contexts
- The relevant probabilities are estimated by a neural network

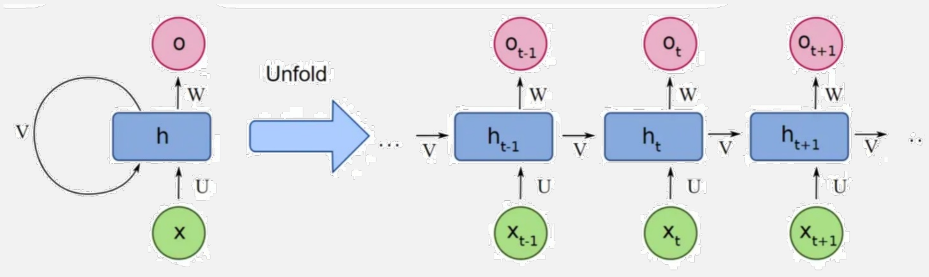
Disadvantages

- Fixed context size makes it hard to use distant contexts

Question

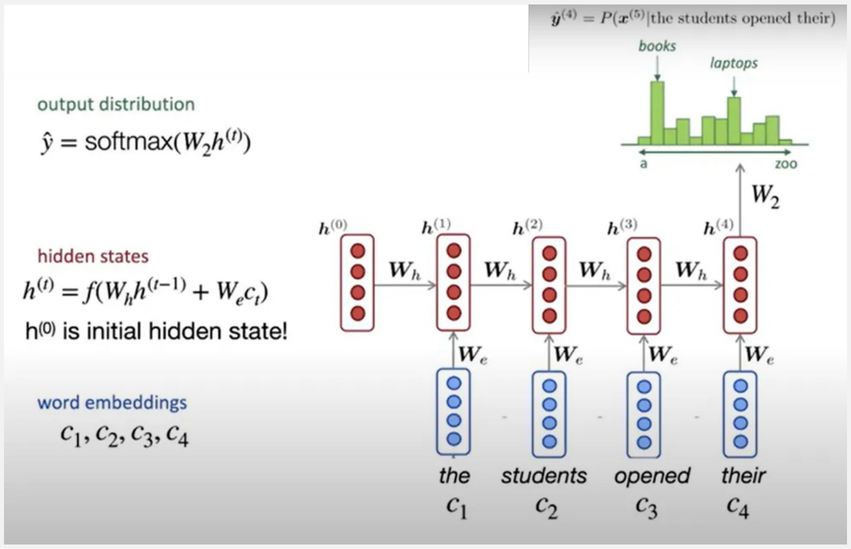
- Can a neural language model take advantage of information provided by the entire context – no matter how long it is?

Recurrent neural network (RNN) language models



- RNN has recurrent connections from the hidden layer to itself
- Input tokens are provided sequentially
- The hidden layer accumulates contextual information over time
- Output at each time is informed by the context seen so far

Recurrent neural network (RNN) language models



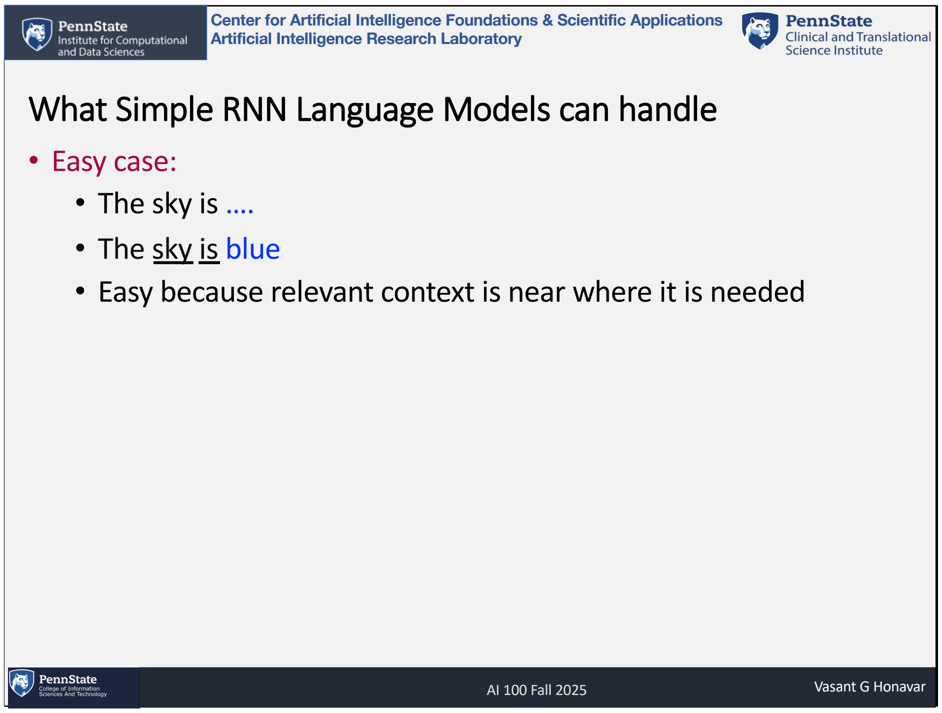
Recurrent neural network language models

Advantages

- No need to store large numbers of n-grams
- Can cope with data sparsity through generalization across semantically similar words and contexts
- The relevant probabilities are estimated by a neural network
- Can accommodate distant context to some extent

Disadvantage

- Over time, the effect of words seen early in the sequence gets overridden by more recently seen words
- Basic RNN language models cannot accommodate distant context

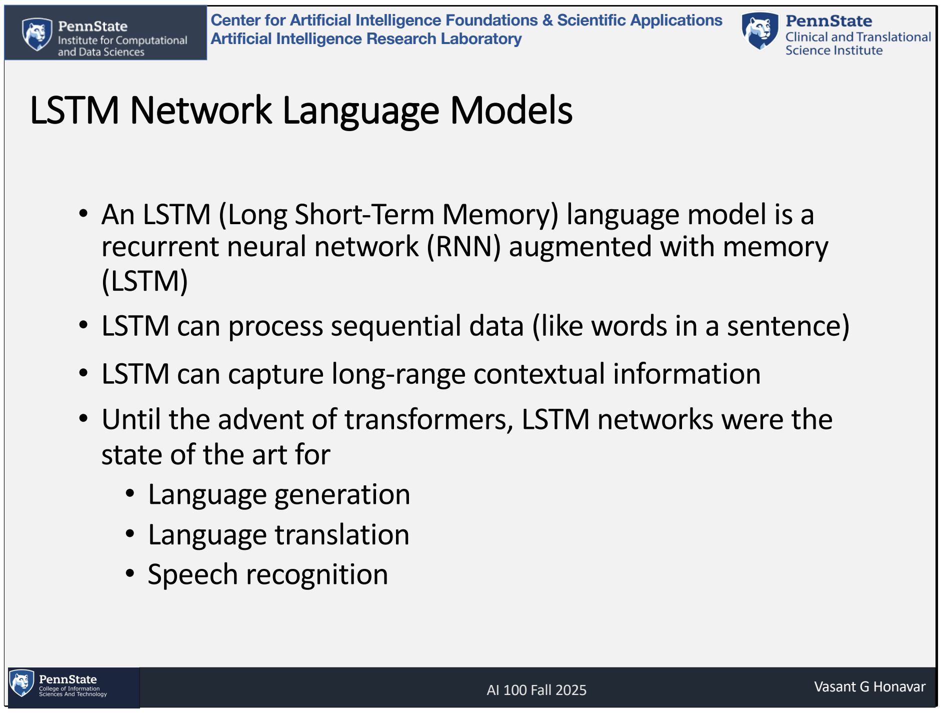


What Simple RNN Language Models can handle

- Easy case:
  - The sky is ….
  - The sky is blue
  - Easy because relevant context is near where it is needed

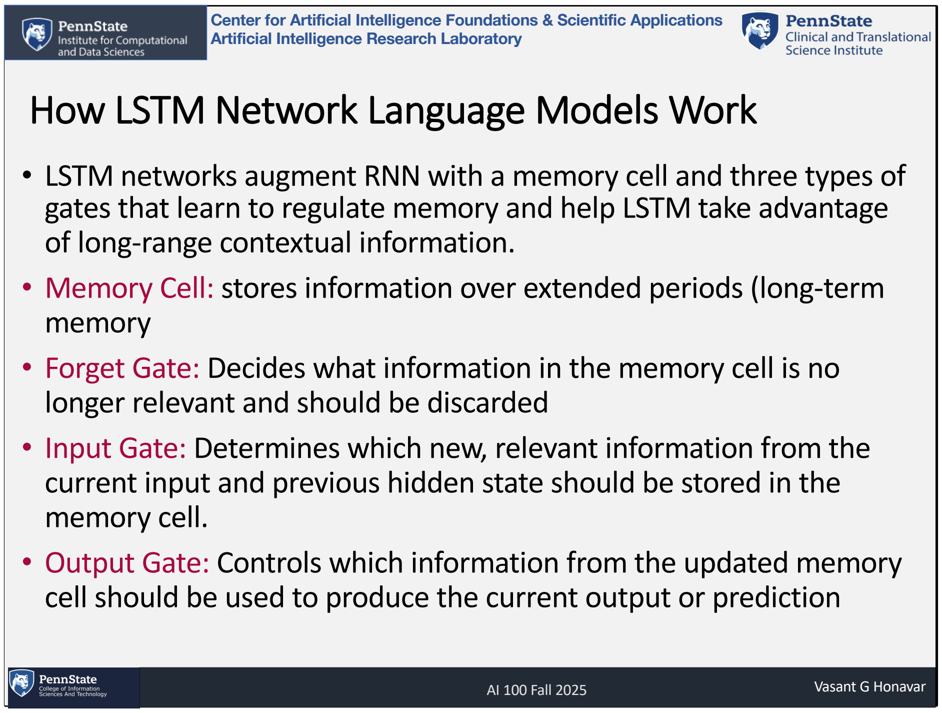
What Simple RNN Language Models can't handle

- Difficult case:
  - I was born in India. I studied Electrical Engineering as an undergraduate. I came to the United States for graduate studies. After a short stint at Drexel where I got my masters degree in Electrical and Computer Engineering, I moved to the University of Wisconsin for my PhD. My PhD research was on neural networks. I play tennis. I speak fluent ….
  - Nearby context suggests the next word must be a language
  - But which language?
  - The relevant context for accurately predicting the next word is very far away, well beyond the reach of the Markov assumption



LSTM Network Language Models

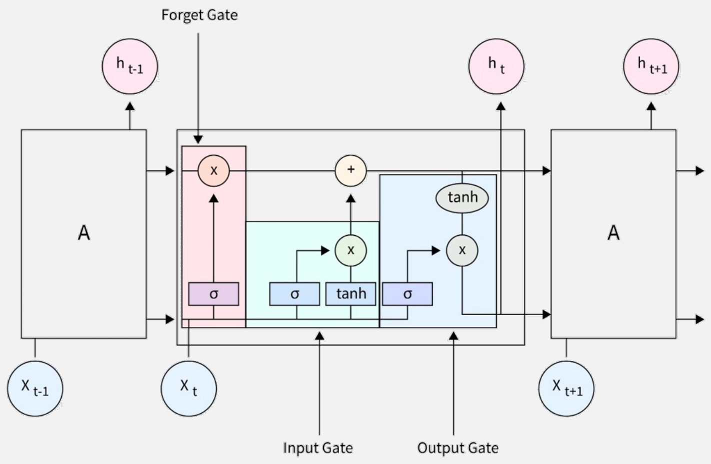
- An LSTM (Long Short-Term Memory) language model is a recurrent neural network (RNN) augmented with memory (LSTM)
- LSTM can process sequential data (like words in a sentence)
- LSTM can capture long-range contextual information
- Until the advent of transformers, LSTM networks were the state of the art for
  - Language generation
  - Language translation
  - Speech recognition



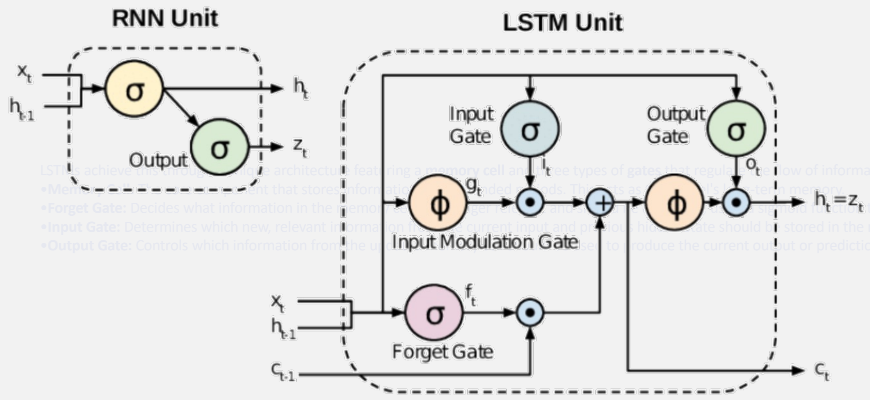
How LSTM Network Language Models Work

- LSTM networks augment RNN with a memory cell and three types of gates that learn to regulate memory and help LSTM take advantage of long-range contextual information.
- Memory Cell: stores information over extended periods (long-term memory
- Forget Gate: Decides what information in the memory cell is no longer relevant and should be discarded
- Input Gate: Determines which new, relevant information from the current input and previous hidden state should be stored in the memory cell.
- Output Gate: Controls which information from the updated memory cell should be used to produce the current output or prediction

Long Short-Term Memory (LSTM)



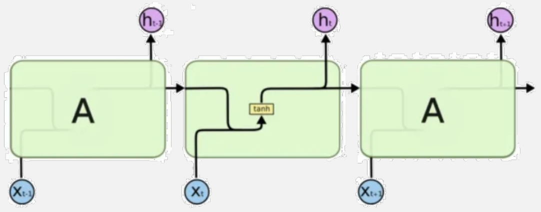
Basic RNN and LSTM Building Blocks Compared

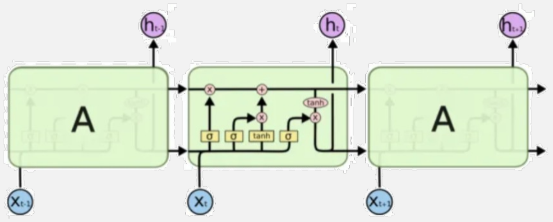


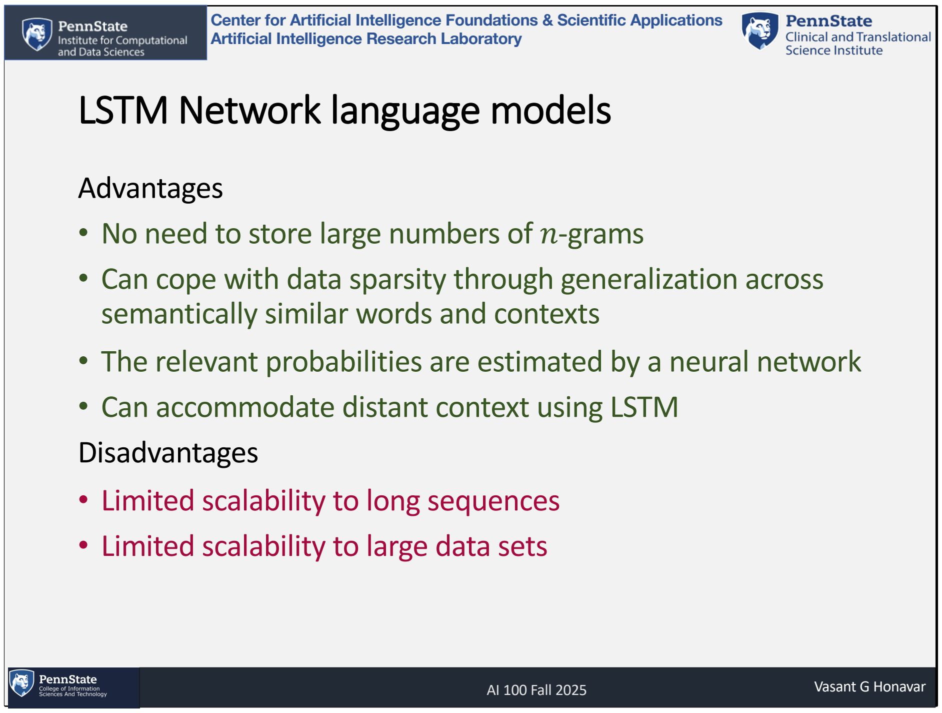
LSTMs achieve this throug[illegible]nique architecture featuring a **memory cell** and three types of **gates** that regula[illegible] flow of information, effectively addressing the "vanishing gradient prob

- **Memory Cell:** [illegible]ent that stores informatio[illegible]nded periods. This acts as [illegible]el's long-term memory.
- **Forget Gate:** Decides what information in the memor[illegible]ger relev[illegible] and sh[illegible] sigmoid function to output values between 0 (forget everything) and 1 (ke
- **Input Gate:** Determines which new, relevant information fr[illegible] current input and previous hid[illegible]tate should be stored in the memory cell.
- **Output Gate:** Controls which information from the up[illegible] used to produce the current output or prediction.

RNN and LSTM Network Architectures Compared







LSTM Network language models

Advantages

- No need to store large numbers of n-grams
- Can cope with data sparsity through generalization across semantically similar words and contexts
- The relevant probabilities are estimated by a neural network
- Can accommodate distant context using LSTM

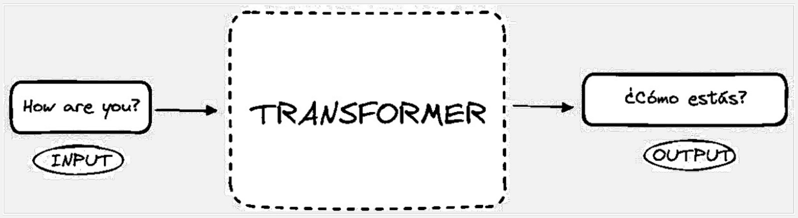
Disadvantages

- Limited scalability to long sequences
- Limited scalability to large data sets

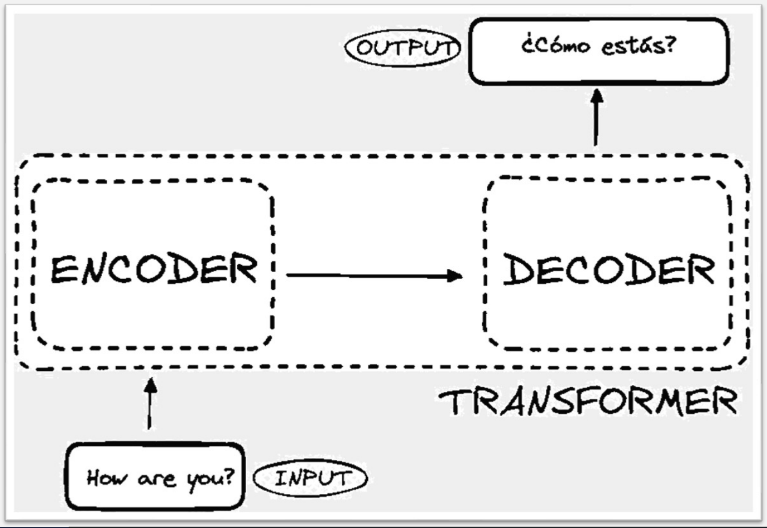
Pre-trained Transformer-based LLM

- Unidirectional models (for example, the GPT family of models), which predict tokens from left to right using the transformer decoder.
- Bidirectional models (for example, BERT, RoBERTa), which contextualize each token based on both left and right contexts using the transformer encoder.
- Sequence-to-sequence models (for example, T5, BART), which combine an encoder and a decoder to support conditional text generation.
- Transformer plays a key role in all three

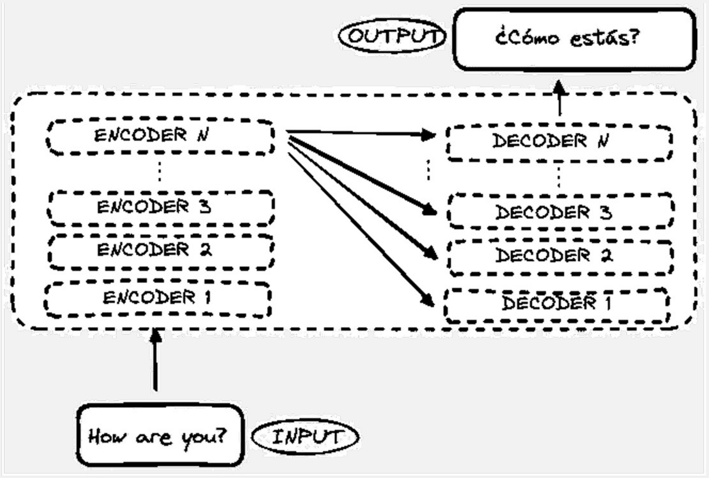
Transformer is a sequence-to-sequence model

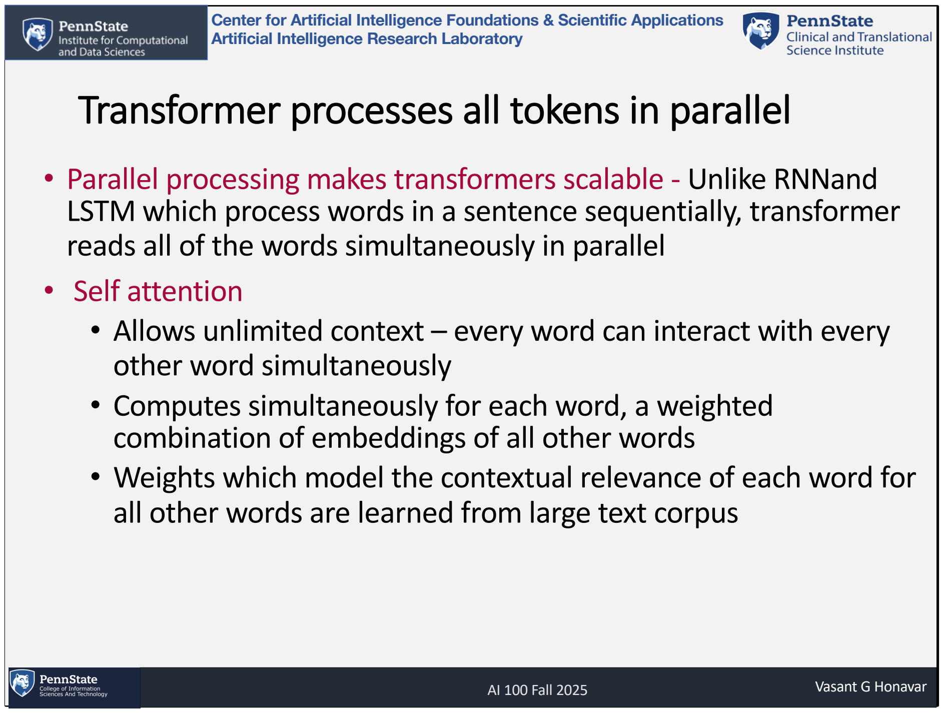


Transformer is an encoder-decoder model



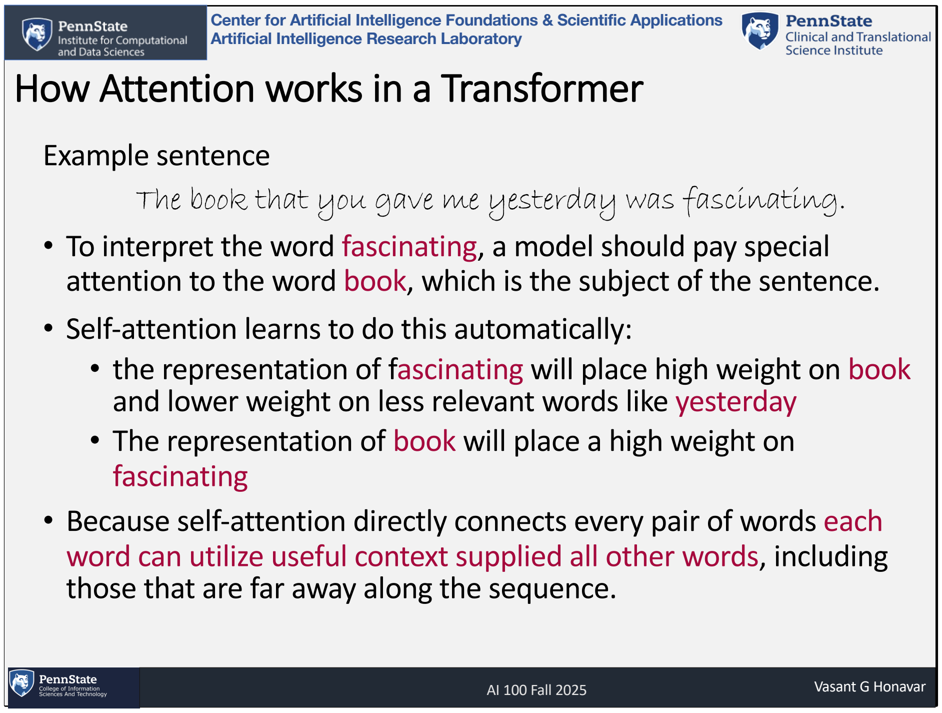
Transformer can stack encoders and decoders





Transformer processes all tokens in parallel

- Parallel processing makes transformers scalable - Unlike RNNand LSTM which process words in a sentence sequentially, transformer reads all of the words simultaneously in parallel
- Self attention
  - Allows unlimited context – every word can interact with every other word simultaneously
  - Computes simultaneously for each word, a weighted combination of embeddings of all other words
  - Weights which model the contextual relevance of each word for all other words are learned from large text corpus

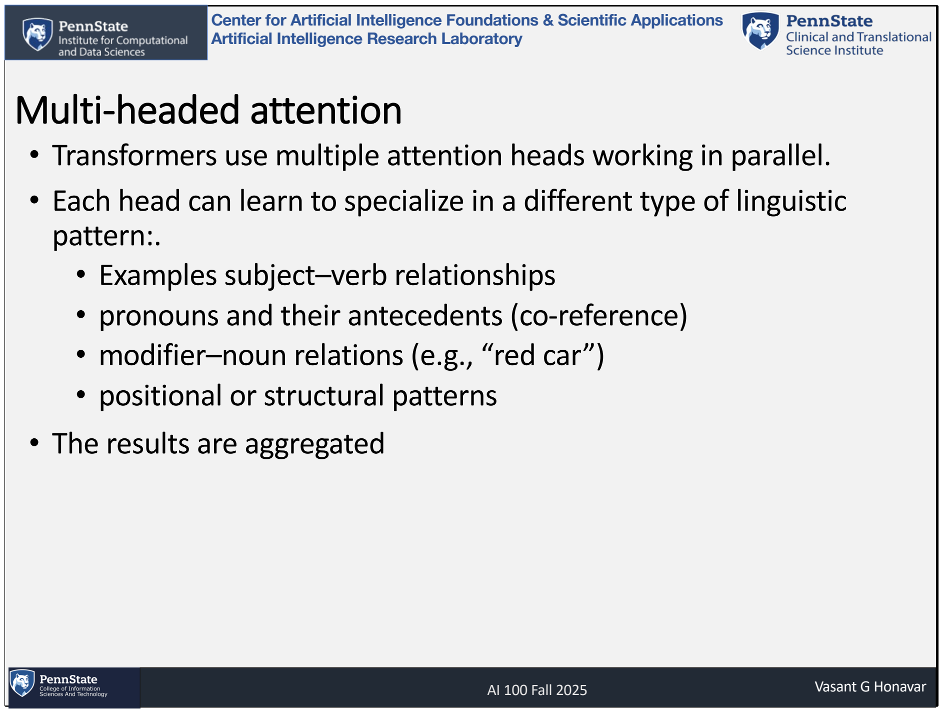


How Attention works in a Transformer

Example sentence

The book that you gave me yesterday was fascinating.

- To interpret the word fascinating, a model should pay special attention to the word book, which is the subject of the sentence.
- Self-attention learns to do this automatically:
  - the representation of fascinating will place high weight on book and lower weight on less relevant words like yesterday
  - The representation of book will place a high weight on fascinating
- Because self-attention directly connects every pair of words each word can utilize useful context supplied all other words, including those that are far away along the sequence.

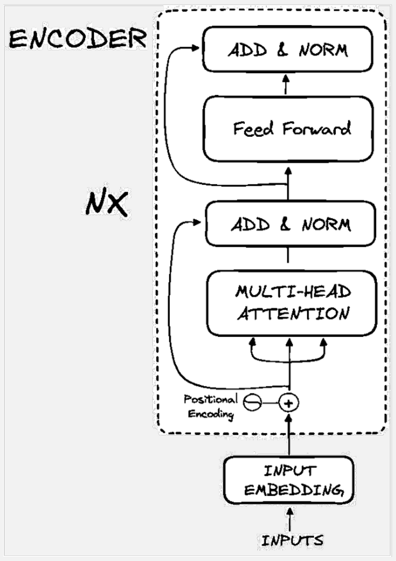


Multi-headed attention

- Transformers use multiple attention heads working in parallel.
- Each head can learn to specialize in a different type of linguistic pattern:.
  - Examples subject–verb relationships
  - pronouns and their antecedents (co-reference)
  - modifier–noun relations (e.g., “red car”)
  - positional or structural patterns
- The results are aggregated

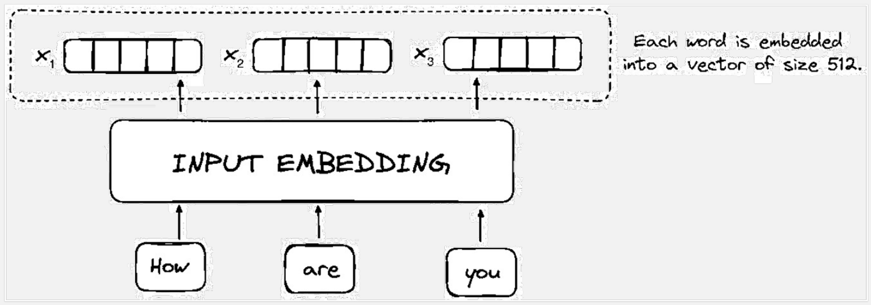
Architecture of Encoders

- Encoder maps embeddings of words in the sentence to an internal representation
- The representation is shaped by attention
- Decoder decodes it



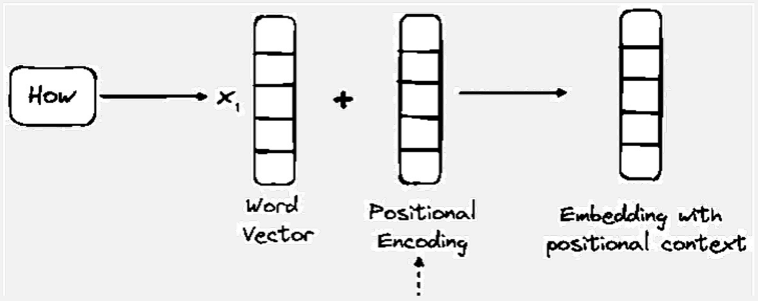
Transformer in action: Embeddings

- Transformer starts by mapping words into their embeddings



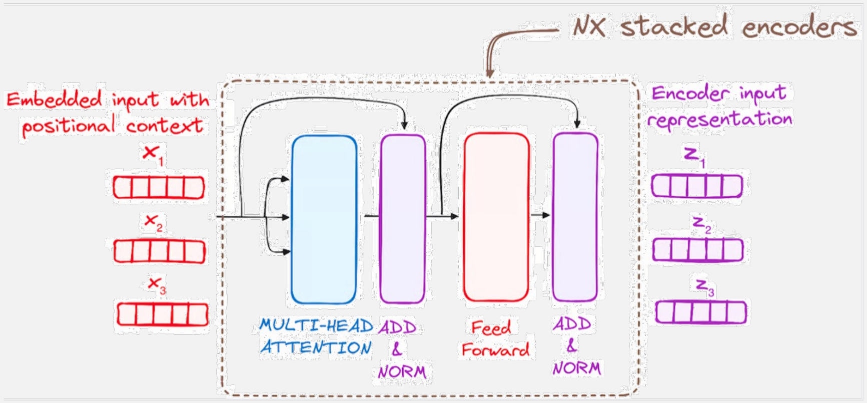
Transformer in action: Positional encoding

- Transformer adds positional encoding to keep track of the position of each word in the sequence



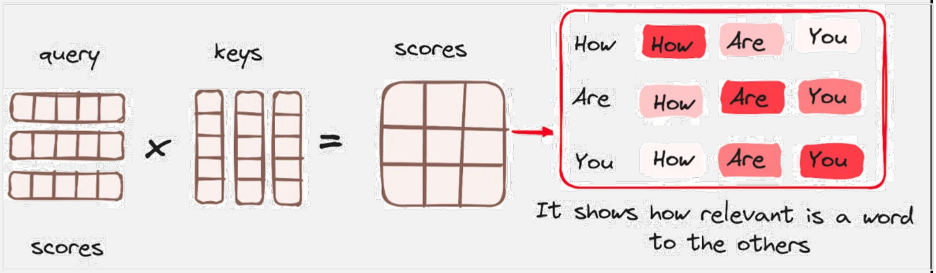
Transformer in action: Encoder stack

- The encoder layer transforms input sequence into a numeric abstract representation of the learned information from the entire sequence using multi-headed attention and linear weighting



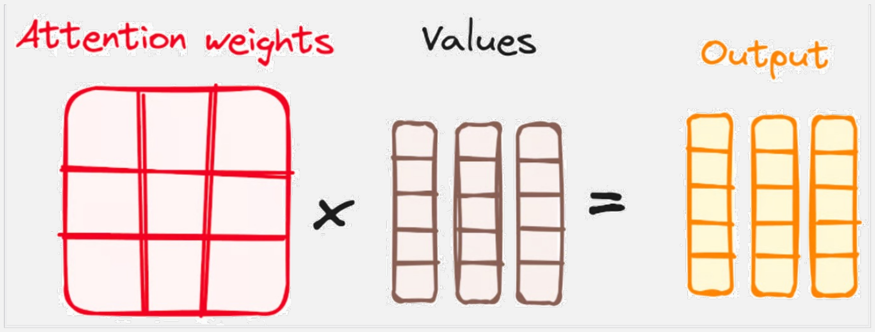
Transformer in action: How attention works

- Transformer maps the word at each position to three vectors using learned weights: query, key, and value
- Attention of word i to word j is based on the similarity of the corresponding query and key
- Representation of a word is the attention weighted values of all other words



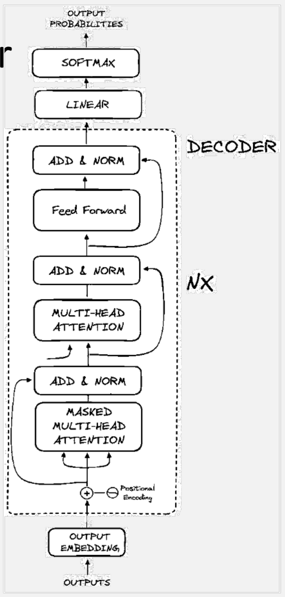
Transformer in action: Encoder outputs

- Encoder output is a set of high scoring vectors that encode words
- These become inputs to the decoder



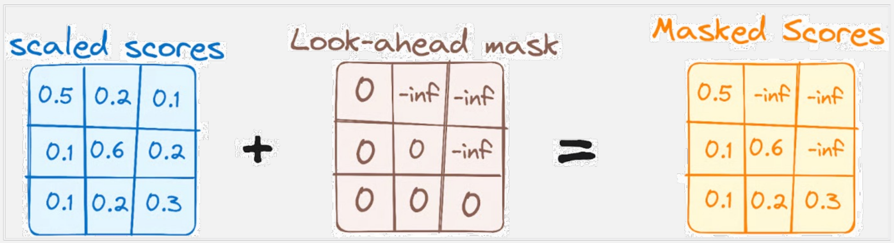
Transformer in action: Decoder

- The decoder's role is to turn encoder output back into sentences.
- The structure of the decoder mirrors that of the decoder, in opposite order

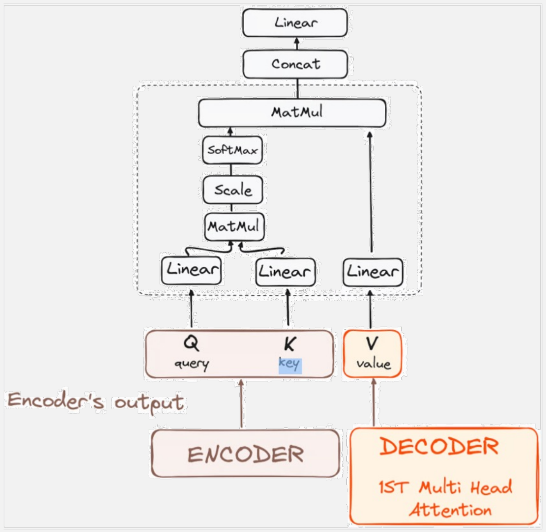


Transformer in action: Masked Self-Attention

- Masked self-attention is like self-attention, but considers only words that have been generated so far – you can't look at words ahead at positions are yet to be generated in generating the current word

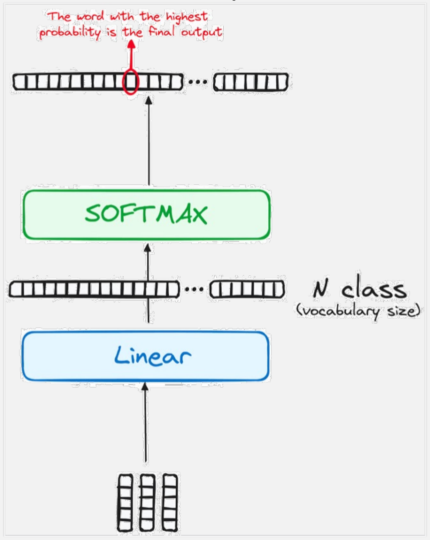


Transformer in action: Align decoder outputs with inputs



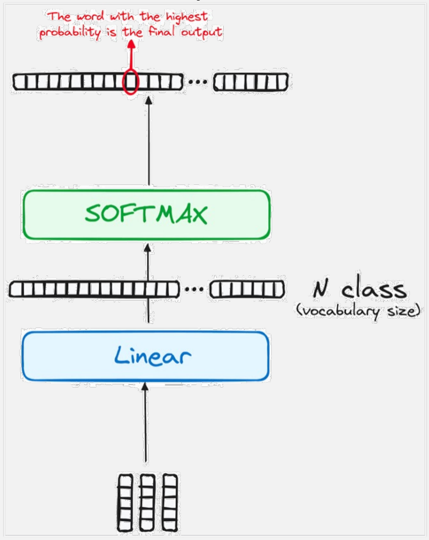
Transformer in action: Generate word probabilities

- Transformer generates probability of each word at each position in the sequence

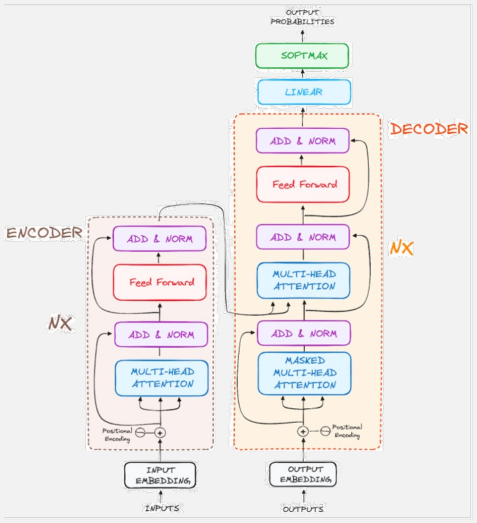


Transformer in action: Generate output sentence

- Transformer generates probability of each word at each position in the sequence
- Sample words according to their probability and produce the output sentence



Transformer in action: overall architecture

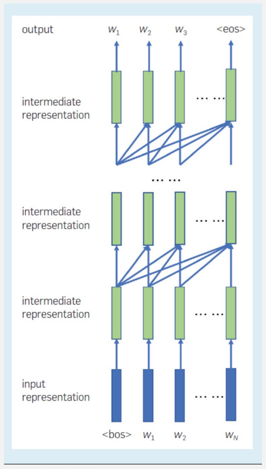


Kinds of transformer-based LLMs

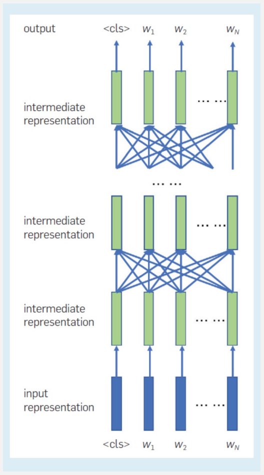
- Encoder-only models (e.g., BERT, RoBERTa) suited for classification and language understanding.
- Decoder-only models (e.g., GPT, GPT-2, GPT-3, and their successors), optimized for text generation.
- Encoder–decoder models (e.g., T5, BART), for translation, summarization, and general sequence-to-sequence tasks.

•Hang Li. 2022. Language models: past, present, and future. Commun. ACM 65, 7 (July 2022), 56–63. https://doi.org/10.1145/3490443

Information flow in representative LLMs


output
w_1 w_2 w_3 <eos>
intermediate representation
intermediate representation
intermediate representation
input representation
<bos> w_1 w_2 w_N

GPT: Generation

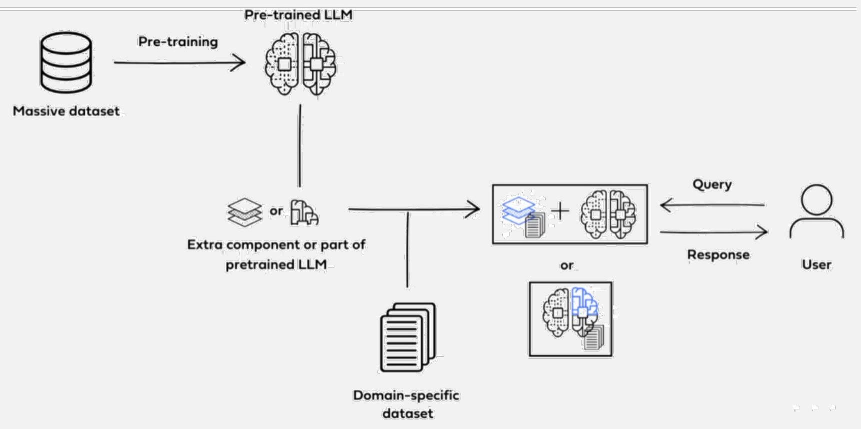

output
<cls> w_1 w_2 w_N
intermediate representation
intermediate representation
intermediate representation
input representation
<cls> w_1 w_2 w_N

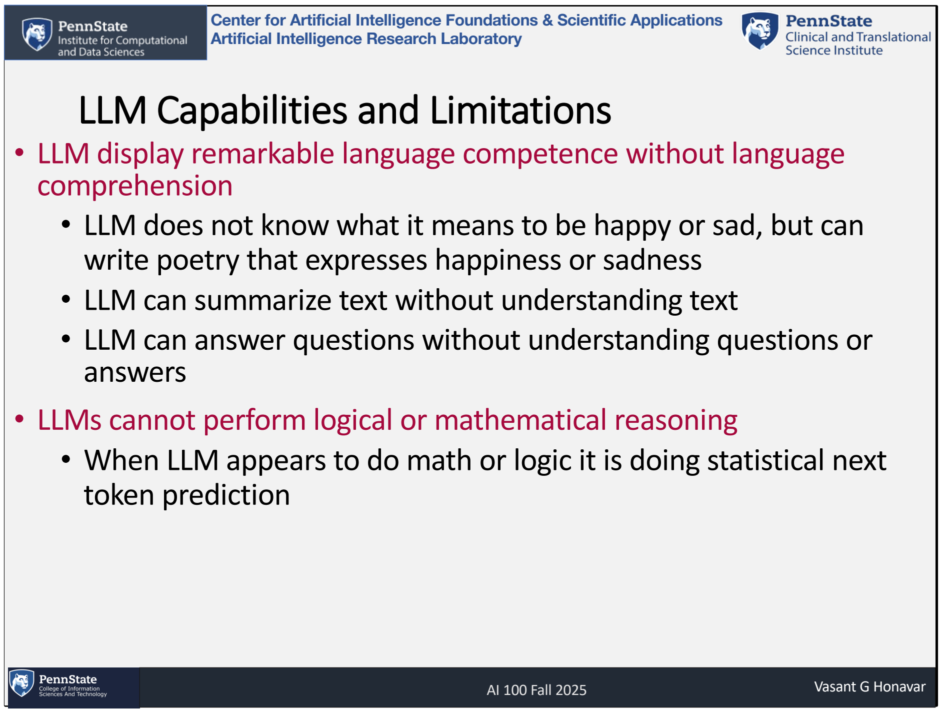
BERT: Summarization

Source: Hang Li. 2022. ACM 65, 7, 56–63. https://doi.org/10.1145/3490443

LLM Finetuning

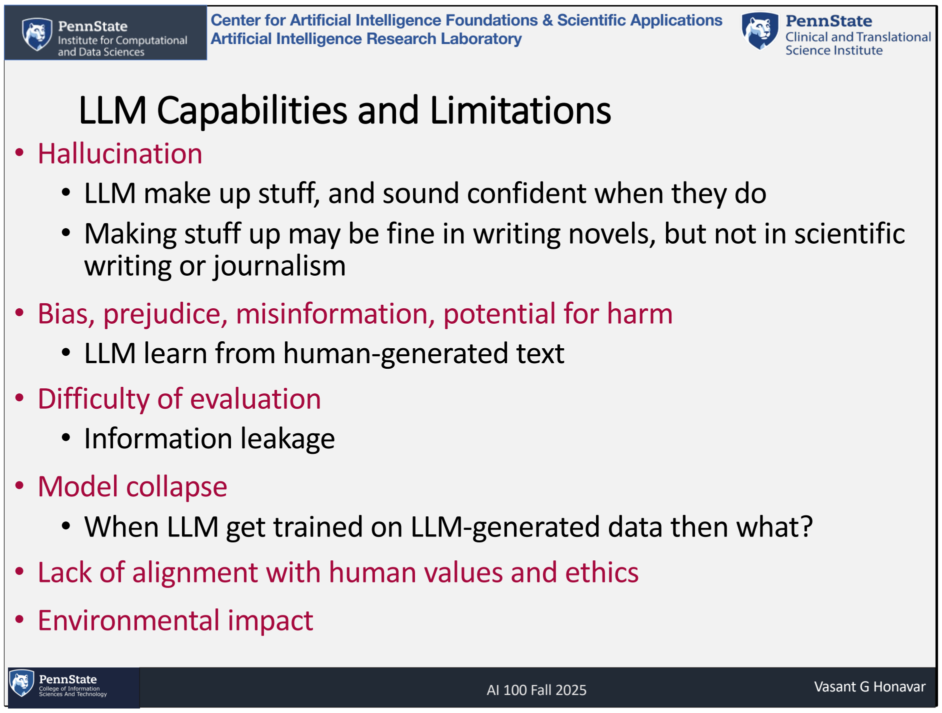
- Pretrained models are often finetuned for specific tasks





LLM Capabilities and Limitations

- LLM display remarkable language competence without language comprehension
  - LLM does not know what it means to be happy or sad, but can write poetry that expresses happiness or sadness
  - LLM can summarize text without understanding text
  - LLM can answer questions without understanding questions or answers
- LLMs cannot perform logical or mathematical reasoning
  - When LLM appears to do math or logic it is doing statistical next token prediction



LLM Capabilities and Limitations

- Hallucination
  - LLM make up stuff, and sound confident when they do
  - Making stuff up may be fine in writing novels, but not in scientific writing or journalism
- Bias, prejudice, misinformation, potential for harm
  - LLM learn from human-generated text
- Difficulty of evaluation
  - Information leakage
- Model collapse
  - When LLM get trained on LLM-generated data then what?
- Lack of alignment with human values and ethics
- Environmental impact

Future possibilities

- Multi-modal models
  - Vision-language
- Physically embodied models
  - Can learn richer semantics
- Generative-Deliberative hybrids
  - LLM guesses, theorem-prover or experiment verifies