

香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

CSC5051/MDS5110/CSC6052 : Natural Language Processing

Lecture 3: Language Models

Fall 2026

Benyou Wang
School of Data Science

Today's Lecture

- **Language Models in a Nutshell (and a recap)**
- Why Language Models?
- Language Models in a *Narrow* Sense
- Language Models in a *Broad* Sense

What Is Language Modeling?

A **language model** assigns a probability to a N-gram

$$f: V^n \rightarrow R^+$$

What Is Language Modeling?

A **language model** assigns a probability to a N-gram

$$f: V^n \rightarrow R^+$$



Sfklkljf fskjhfkjsh kjfs fs kjhkjhs fsjhfkshkjfh

Low probability



ChatGPT is all you need

high probability

What Is Language Modeling?

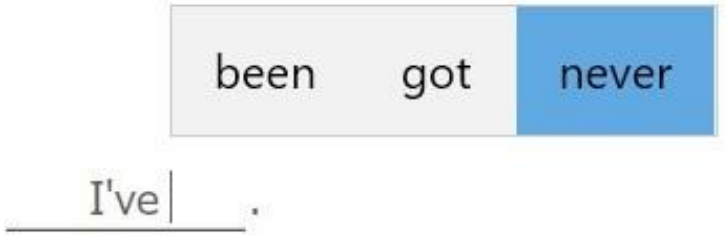
A **language model** assigns a probability to a N-gram

$$f: V^n \rightarrow R^+$$

A **conditional language model** assigns a probability of a word given some conditioning context

$$g: (V^{n-1}, V) \rightarrow R^+$$

And $$p(w_n | w_1 \cdots w_{n-1}) = g(w_1 \cdots w_{n-1}, w) = \frac{f(w_1 \cdots w_n)}{f(w_1 \cdots w_{n-1})}$$



What Is Language Modeling?

A **language model** assigns a probability to a N-gram

$$f: V^n \rightarrow R^+$$

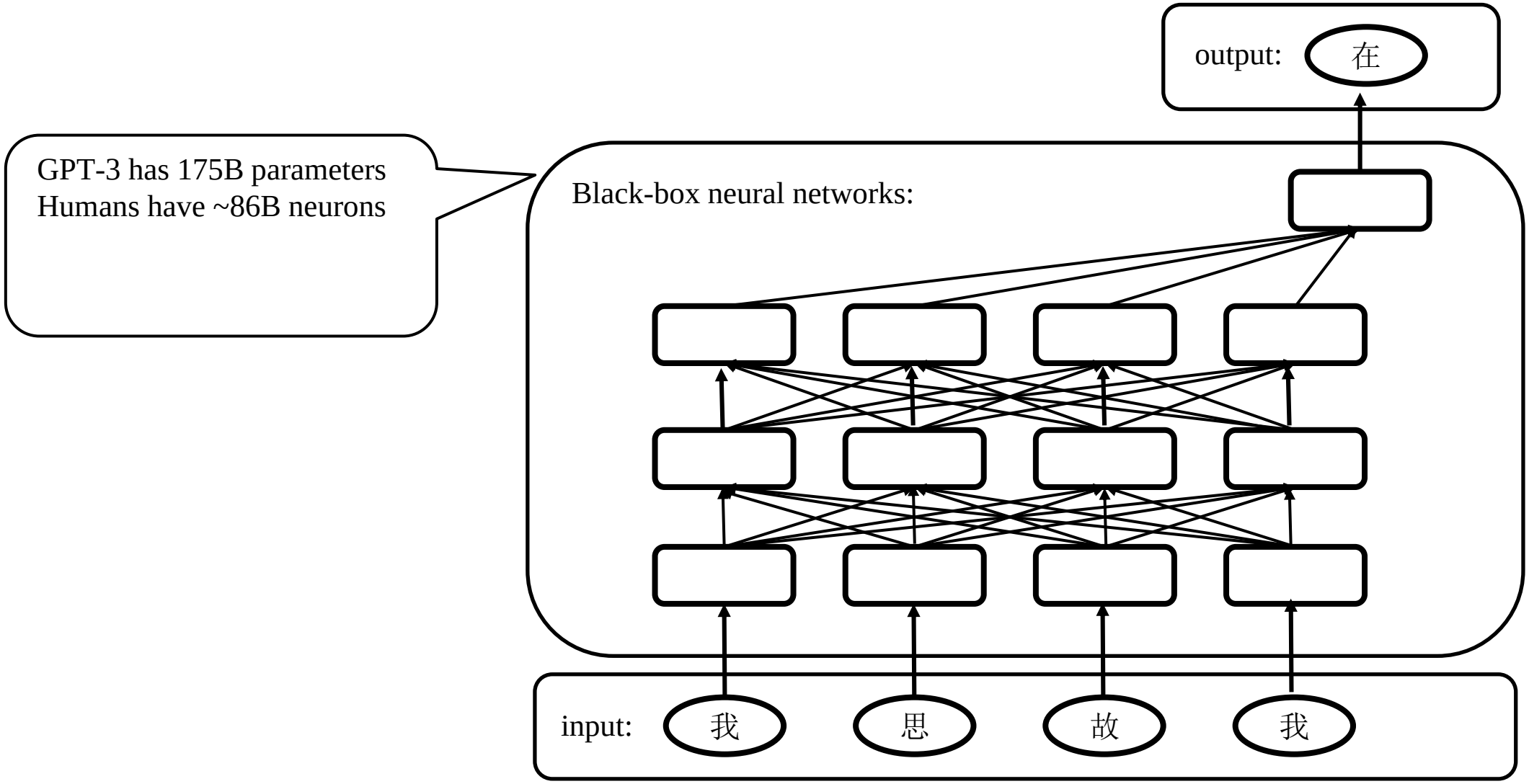
A **conditional language model** assigns a probability of a word given some conditioning context

$$g:(V^{n-1}, V) \rightarrow R^+$$

And $$p(w_n|w_1 \cdots w_{n-1}) = g(w_1 \cdots w_{n-1}, w) = \frac{f(w_1 \cdots w_n)}{f(w_1 \cdots w_{n-1})}$$

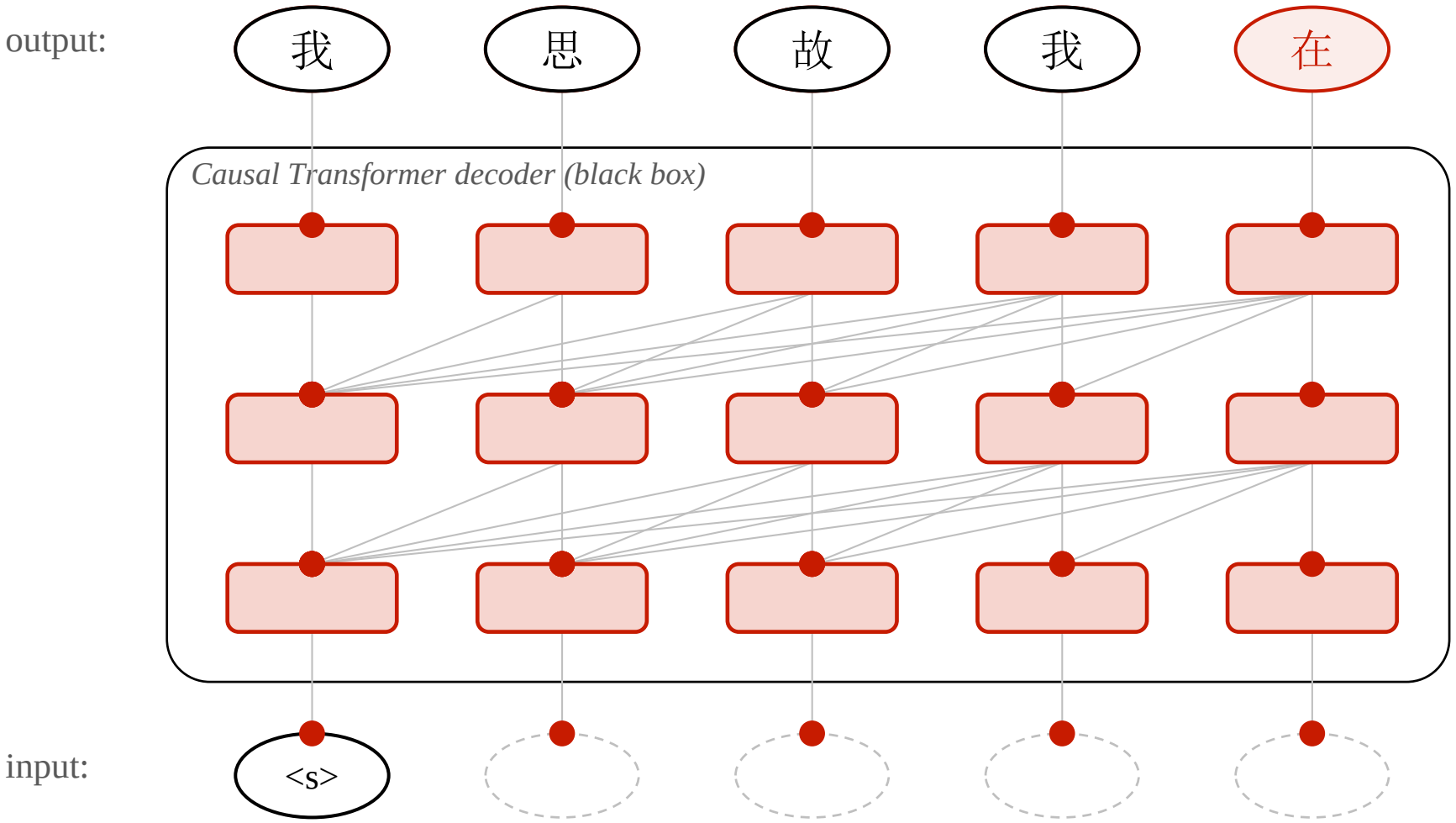
$p(w_n|w_1 \cdots w_{n-1})$ is the foundation of **modern large language models** (GPT, ChatGPT, etc.)

Language Model Using Neural Networks



Autoregressive Generation with a Neural LM: 我思故我在

Generated: 我思故我在 *"I think, therefore I am."*



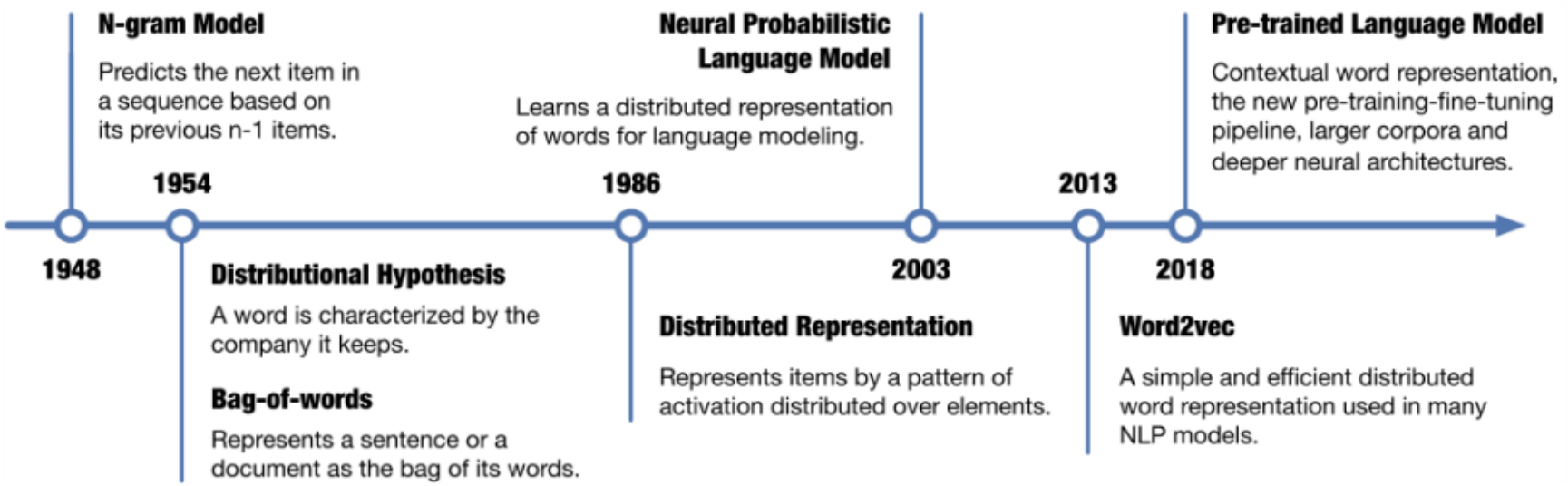
Each click = one step

1. Read all tokens so far
2. Information flows up; each position only looks left (causal)
3. The last position predicts the next character
4. Append it to the input and repeat

Lit columns stay lit: their keys/values are cached and reused (KV cache).

Background

- Language model (from *counting* to *prediction*)



Liu et al., Representation Learning for Natural Language Processing, Springer, 2020

Language Models: Narrow Sense

A probabilistic model that assigns a probability to every finite sequence (grammatical or not)

Sentence: “the cat sat on the mat”

$$P(\text{the cat sat on the mat}) = P(\text{the})\, P(\text{cat} \mid \text{the})\, P(\text{sat} \mid \text{the cat}) \times P(\text{on} \mid \text{the cat sat})\, P(\text{the} \mid \text{the cat sat on}) \times P(\text{mat} \mid \text{the cat sat on the})$$

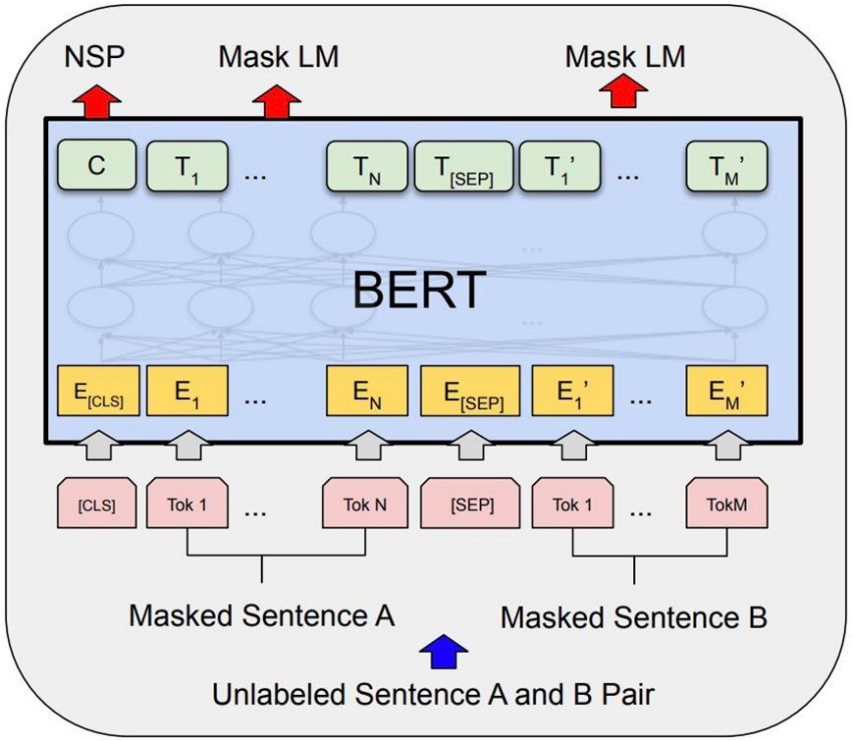
Implicit order: each word is conditioned on the words before it.

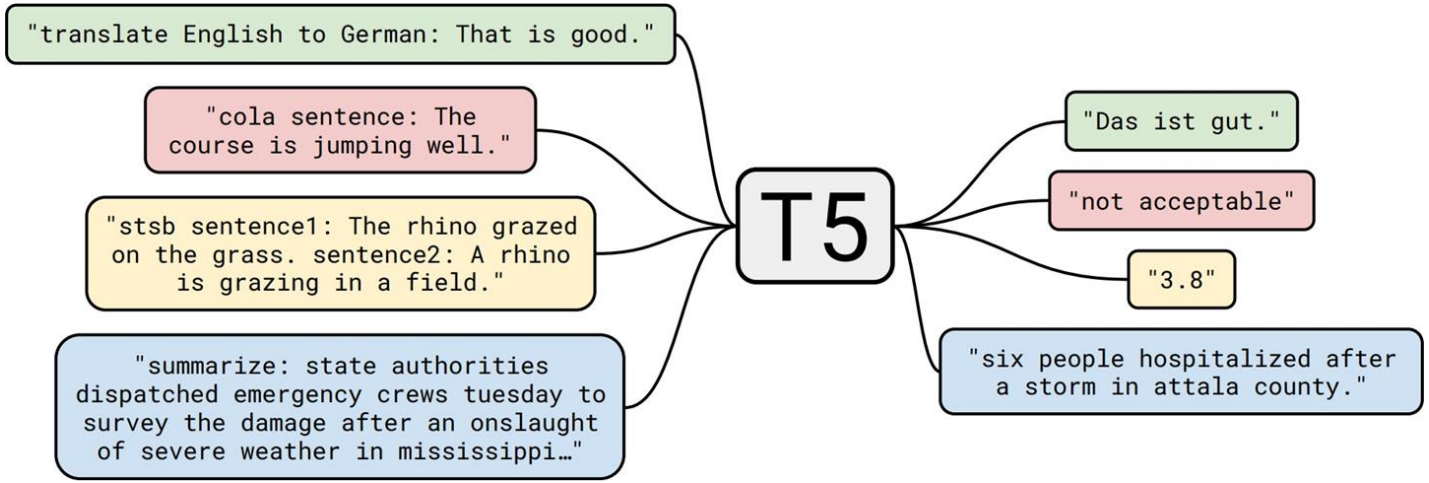
GPT-3 still acts in this way but the model is implemented as a very large neural network of 175-billion parameters!

Language Models: Broad Sense

- Decoder-only models (GPT-x models)
- Encoder-only models (BERT, RoBERTa, ELECTRA)
- Encoder-decoder models (T5, BART)

The latter two usually involve a different **pre-training** objective.





Today's Lecture

- Language Models in a Nutshell
- **Why Language Models?**
- Language Models in a *Narrow* Sense
- Language Models in a *Broad* Sense

Why Language Models?

- LM (next word prediction) is scalable
- LM does not need annotations (data is scalable)
- Large LM generalizes to some extent (from task A and task B, e.g. few-shot learning)

- LM is efficient to store **knowledge** as a knowledge base
  - imagine recording knowledge with images instead?
- A t**hinking** format
  - Imagine that animals (without rich language) cannot reason too much.
- Humans do LM **everyday** (do next-word/ next-second prediction)
  - There naturally exists a lot of text data for LM
- **Communication** Interface between humans and agent/computers
  - Also describe *actions/instructions to* robots!

What Can We Learn from Reconstructing the Input?

I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, ____

34

Fibonacci sequence

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture9-pretraining.pdf

What Can We Learn from Reconstructing the Input?

Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was ___.

Boring or great?
Sentiment analysis

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture9-pretraining.pdf

What Can We Learn from Reconstructing the Input?

The woman walked across the street, checking for traffic over ___ shoulder.

her ~~or his~~?

Context understanding

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture9-pretraining.pdf

What Can We Learn from Reconstructing the Input?

whales、**dolphins**?
Commonsense knowledge

I went to the ocean to see the fish, turtles, seals, and _____.

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture9-pretraining.pdf

Today's Lecture

- Language Models in a Nutshell
- Why Language Models?
- **Language Models in a *Narrow* Sense**
  (Probability theory, N-gram language model)
- Language Models in a Broad Sense

Recap: Basic Probability Theory

Why Do We Need Language Models?

Many NLP tasks require **natural language output**:

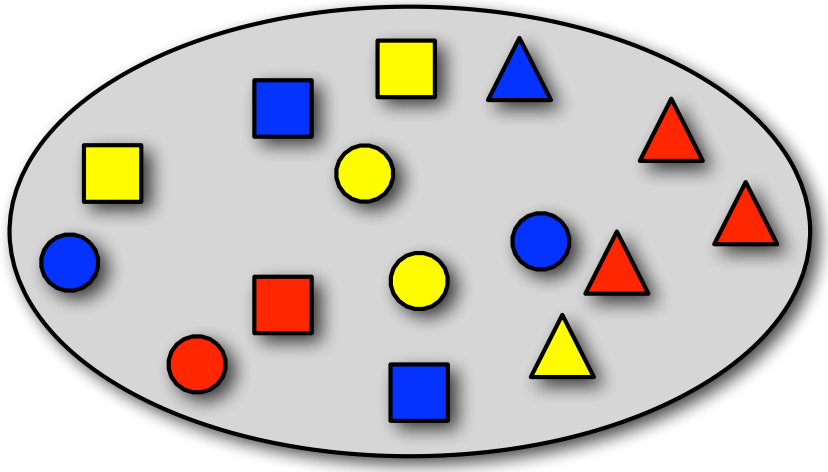
- **Machine translation**: return text in the target language
- **Speech recognition**: return a transcript of what was spoken
- **Natural language generation**: return natural language text
- **Spell-checking**: return corrected spelling of input

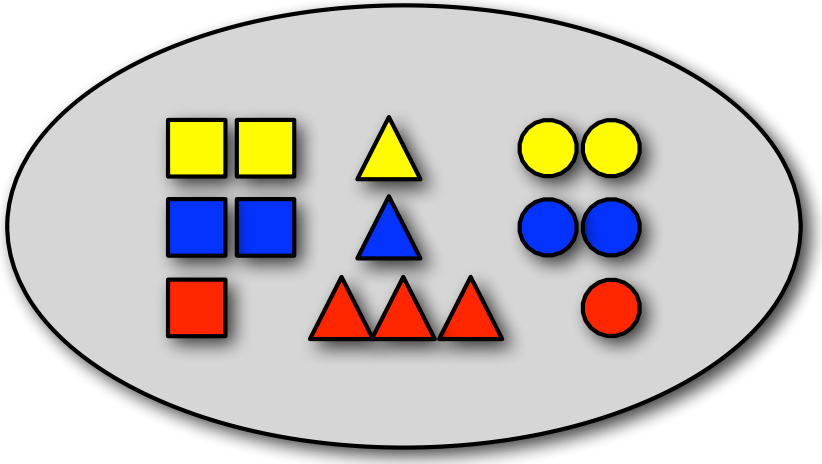
Language models define **probability distributions over (natural language) strings or sentences**.

➔We can use a language model to **score possible output strings** so that we can choose the best (i.e. most likely) one: if $P_{LM}(A) > P_{LM}(B)$, return A, not B

Sampling with Replacement

Pick a random shape, then put it back in the bag.





$P(\blacksquare) = 2/15$

$P(\text{blue}) = 5/15$

$P(\text{blue} \mid \square) = 2/5$

$P(\blacksquare) = 1/15$

$P(\text{red}) = 5/15$

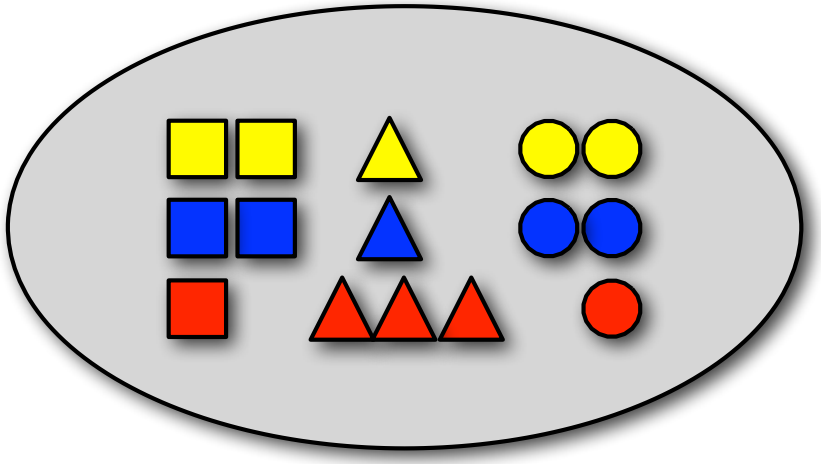
$P(\square) = 5/15$

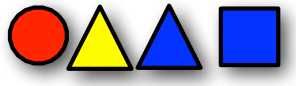
$P(\blacksquare \text{ or } \blacktriangle) = 2/15$

$P(\triangle \mid \text{red}) = 3/5$

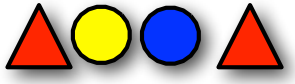
Sampling with Replacement

Pick a random shape, then put it back in the bag.
What sequence of shapes will you draw?

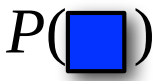


$P($  $)$

$= 1/15 \times 1/15 \times 1/15 \times 2/15 = 2/50625$

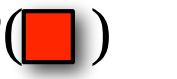
$P($  $)$

$= 3/15 \times 2/15 \times 2/15 \times 3/15 = 36/50625$

$P($  $) = 2/15$

$P(\text{blue}) = 5/15$

$P(\text{blue} \mid$  $) = 2/5$

$P($  $) = 1/15$

$P(\text{red}) = 5/15$

$P($  $) = 5/15$

$P($  $) = 2/15$

$P($ $\mid \text{red}) = 3/5$

Sampling with Replacement

```
Alice was beginning to get very tired of
sitting by her sister on the bank, and of
having nothing to do: once or twice she
had peeped into the book her sister was
reading, but it had no pictures or
conversations in it, 'and what is the use
of a book,' thought Alice 'without
pictures or conversation?'
```

$P(\texttt{of}) = 3/66$
$P(\texttt{Alice}) = 2/66$
$P(\texttt{was}) = 2/66$
$P(\texttt{to}) = 2/66$

$P(\texttt{her}) = 2/66$
$P(\texttt{sister}) = 2/66$
$P(\texttt{,}) = 4/66$
$P(\texttt{'}) = 4/66$

Probability Theory: Terminology

Trial (aka "experiment")

Picking a shape, predicting a word

Sample space Ω:

The **set of all possible outcomes**
(all shapes; all words in *Alice in Wonderland*)

Event $\omega \subseteq \Omega$:

An **actual outcome** (a subset of Ω)
(predicting '*the*', picking a triangle)

Random variable $X: \Omega \rightarrow T$

A function from the sample space (often the identity function)
Provides a 'measurement of interest' from a trial/experiment (Did we pick 'Alice'/a noun/a word starting with "x"/…?)

What Is a Probability Distribution?

$P(\omega)$ defines a **distribution** over Ω iff

1) Every event ω has a probability $P(\omega)$ between 0 and 1:

$$0 \leq \mathbb{P}(\omega \subseteq \Omega) \leq 1$$

2) The *null* event $\emptyset$ has probability $P(\emptyset) = 0$:

$$\mathbb{P}(\emptyset) = 0$$

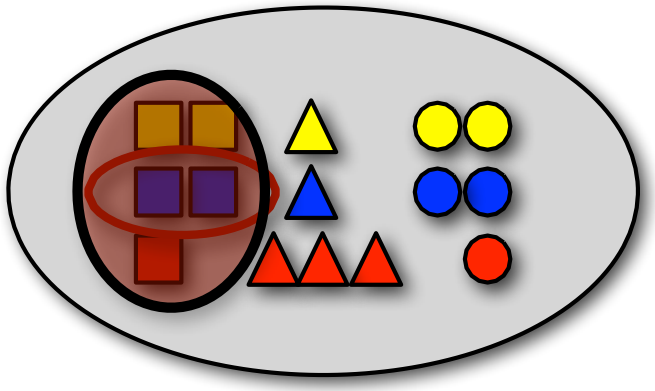
3) And the probability of all *disjoint* events sums to 1.

$$\sum_{\omega_i \subseteq \Omega} P(\omega_i) = 1 \quad \text{if } \forall j \neq i : \omega_i \cap \omega_j = \emptyset \text{ and } \bigcup_i \omega_i = \Omega$$

Joint and Conditional Probability

The conditional probability of X given Y, $P(X \mid Y)$, is defined in terms of the probability of Y, $P(Y)$, and the joint probability of X and Y, $P(X,Y)$:

$$P(X \mid Y) = \frac{P(X,Y)}{P(Y)}$$



$P(\text{blue} \mid \blacksquare) = 2/5$

The Chain Rule

The joint probability $P(X,Y)$ can also be expressed in terms of the conditional probability $P(X \mid Y)$

$$P(X, Y) = P(X \mid Y) P(Y)$$

This leads to the so-called chain rule

$$P(X_1,X_2,\ldots,X_n) = P(X_1)\, P(X_2 \mid X_1)\, P(X_3 \mid X_2,X_1) \cdots P(X_n \mid X_1,\ldots,X_{n-1})$$

$$= P(X_1) \prod_{i=2}^{n} P(X_i \mid X_1,\ldots,X_{i-1})$$

Independence

Two random variables X and Y are independent if

$$P(X, Y) = P(X)\,P(Y)$$

If X and Y are independent, then $P(X \mid Y) = P(X)$:

$$P(X \mid Y) = \frac{P(X,Y)}{P(Y)} = \frac{P(X)\,P(Y)}{P(Y)} = P(X) \qquad (X, Y \text{ independent})$$

Probability Models

Building a probability model consists of two steps:

1. Defining the model
2. Estimating the model's parameters (= training/learning)

Models (almost) always make independence *assumptions.*

That is, even though X and Y are not actually independent, our model may treat them as independent.

This reduces the number of model parameters that we need to estimate (e.g. from n^2 to $2n$)

Language Modeling with N-grams

Counting-based instead of prediction-based in Neural Networks

Language Modeling with N-grams

A language model over a vocabulary V assigns probabilities to strings drawn from V*.

Recall the **chain rule**:

$$P(w^{(1)} \ldots w^{(i)}) = P(w^{(1)}) \cdot P(w^{(2)} | w^{(1)}) \cdot \ldots \cdot P(w^{(i)} | w^{(i-1)}, \ldots, w^{(1)})$$

An **n-gram** language model assumes each word depends only on **the last n−1 words**:

$$P_{ngram}(w^{(1)} \ldots w^{(i)}) = P(w^{(1)}) \cdot P(w^{(2)} | w^{(1)}) \cdot \ldots \cdot P(w^{(i)} | w^{(i-1)}, \ldots, w^{(i-n+1)})$$

N-gram Models

N-gram models *assume* each word (event) depends only on the previous n−1 words (events):

$$\text{Unigram model: } P\left(w^{(1)}\ldots w^{(N)}\right)=\prod_{i=1}^{N} P\left(w^{(i)}\right)$$

$$\text{Bigram model: } P\left(w^{(1)}\ldots w^{(N)}\right)=\prod_{i=1}^{N} P\left(w^{(i)} \mid w^{(i-1)}\right)$$

$$\text{Trigram model: } P\left(w^{(1)}\ldots w^{(N)}\right)=\prod_{i=1}^{N} P\left(w^{(i)} \mid w^{(i-1)},w^{(i-2)}\right)$$

Such independence assumptions are called Markov assumptions (of order n−1).

A Unigram Model for Alice

```
beginning by, very Alice but was and?
reading no tired of to into sitting
sister the, bank, and thought of without
her nothing: having conversations Alice
once do or on she it get the book her had
peeped was conversation it pictures or
sister in, 'what is the use had twice of
a book''pictures or' to
```

$P(\texttt{of}) = 3/66$
$P(\texttt{Alice}) = 2/66$
$P(\texttt{was}) = 2/66$
$P(\texttt{to}) = 2/66$

$P(\texttt{her}) = 2/66$
$P(\texttt{sister}) = 2/66$
$P(\texttt{,}) = 4/66$
$P(\texttt{'}) = 4/66$

In this model, $P(\textit{English sentence}) = P(\textit{word salad})$

A Bigram Model for Alice

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?'

$P(w^{(i)} = \textbf{of} \mid w^{(i-1)} = \textbf{tired}) = 1$

$P(w^{(i)} = \textbf{of} \mid w^{(i-1)} = \textbf{use}) = 1$

$P(w^{(i)} = \textbf{sister} \mid w^{(i-1)} = \textbf{her}) = 1$

$P(w^{(i)} = \textbf{beginning} \mid w^{(i-1)} = \textbf{was}) = 1/2$

$P(w^{(i)} = \textbf{reading} \mid w^{(i-1)} = \textbf{was}) = 1/2$

$P(w^{(i)} = \textbf{bank} \mid w^{(i-1)} = \textbf{the}) = 1/3$

$P(w^{(i)} = \textbf{book} \mid w^{(i-1)} = \textbf{the}) = 1/3$

$P(w^{(i)} = \textbf{use} \mid w^{(i-1)} = \textbf{the}) = 1/3$

How to Estimate Probabilities in LM?

Learning (Estimating) a Language Model

Where do we get the parameters of our model (its actual probabilities) from?

$$P(w^{(i)} = \text{`the'} \mid w^{(i-1)} = \text{`on'}) = \textbf{???}$$

We need (a large amount of) text as training data to estimate the parameters of a language model.

The most basic parameter estimation technique: relative frequency estimation (= counts)

$$P(w^{(i)} = \text{`the'} \mid w^{(i-1)} = \text{`on'}) = C(\text{`on the'}) / C(\text{`on'})$$

Also called Maximum Likelihood Estimation (MLE)

NB: MLE assigns *all* probability mass to events that occur in the training corpus.

Are N-gram Models Actual Language Models?

How Do N-gram Models Define P(L)?

An n-gram model defines $P_{ngram}(w^{(1)} \ldots w^{(N)})$ in terms of the probability of predicting each word:

$$P_{\text{bigram}}\left(w^{(1)} \ldots w^{(N)}\right) = \prod_{i=1}^{N} P\left(w^{(i)} \mid w^{(i-1)}\right)$$

With a fixed vocabulary V, it's easy to make sure $P(w^{(i)} \mid w^{(i-1)})$ is a distribution:

$$\sum_{i=1}^{|V|} P\left(w_i \mid w_j\right) = 1 \quad \text{and} \quad \forall i,j: 0 \leq P\left(w_i \mid w_j\right) \leq 1$$

If $P(w^{(i)}|w^{(i-1)})$ is a distribution, this model defines *one* distribution (over all strings) *for each length* N

But the strings of a language L ***don't all have the same length***

English = {*"yes!", "I agree", "I see you", ...*}

And there is *no* N_{max} that limits how long strings in L can get.

Solution: the **EOS (end-of-sentence)** token!

How Do N-gram Models Define P(L)?

Think of a language model as a stochastic process:

- At each time step, randomly pick one more word.
- Stop generating more words when the word you pick is a special end- of-sentence (EOS) token.

To be able to pick the EOS token, we have to modify our training data so that each sentence ends in EOS.

This means our vocabulary is now $V^{EOS} = V \cup \{EOS\}$

We then get an actual language model, i.e. a distribution over strings of *any* length

Technically, this is only true because P(EOS | …) will be high enough that we are always guaranteed to stop after having generated a finite number of words

Why do we care about having one model for all lengths? We can now compare the probabilities of strings of different lengths, because they're computed by the same distribution.

A Couple More Modifications…

Handling Unknown Words: UNK

Training:

-Assume a fixed vocabulary (e.g. all words that occur at least *n* times in the training corpus)

-Replace all other words in the corpus by a token <UNK>

-Estimate the model on this modified training corpus.

Testing (e.g to compute probability of a string):

-Replace any words not in the vocabulary by <UNK>

Refinements:

use different UNK tokens for different types of words (numbers, etc.).

What About the Beginning of the Sentence?

In a trigram model

$$P\left(w^{(1)}w^{(2)}w^{(3)}\right) = P\left(w^{(1)}\right)\, P\left(w^{(2)} \mid w^{(1)}\right)\, P\left(w^{(3)} \mid w^{(2)}, w^{(1)}\right)$$

only the third term $P(w^{(3)}|w^{(2)},w^{(1)})$ is an actual trigram probability. What about $P(w^{(1)})$ and $P(w^{(2)}|w^{(1)})$?

If this bothers you:

Add n–1 **beginning-of-sentence** (BOS) symbols to each sentence for an n–gram model:

$\mathtt{BOS}_1\ \mathtt{BOS}_2\ \mathtt{Alice\ was}\ \ldots$

Now the unigram and bigram probabilities involve only BOS symbols.

Using N-gram Models to Generate Language

Which language model generates *more fluent* text?

A. Unigram
B. Bigram
C. Trigram
D. N-gram with a larger N

Generating from a Distribution

How do you generate text from an *n*-gram model?

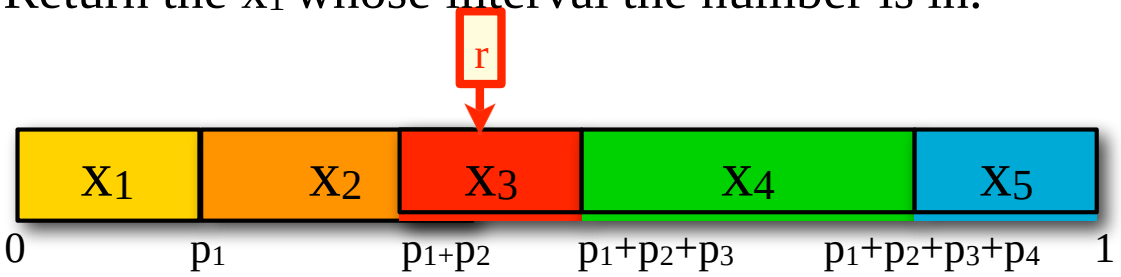
That is, how do you sample from a distribution $P(X \mid Y=y)$?

-Assume X has N possible outcomes (values): $\{x_1, \ldots, x_N\}$ and $P(X=x_i \mid Y=y) = p_i$

- Divide the interval [0,1] into N smaller intervals according to the probabilities of the outcomes

-Generate a random number r between 0 and 1.

-Return the x_1 whose interval the number is in.





Generating with a Bigram Model: Spin the Wheel (1/5)

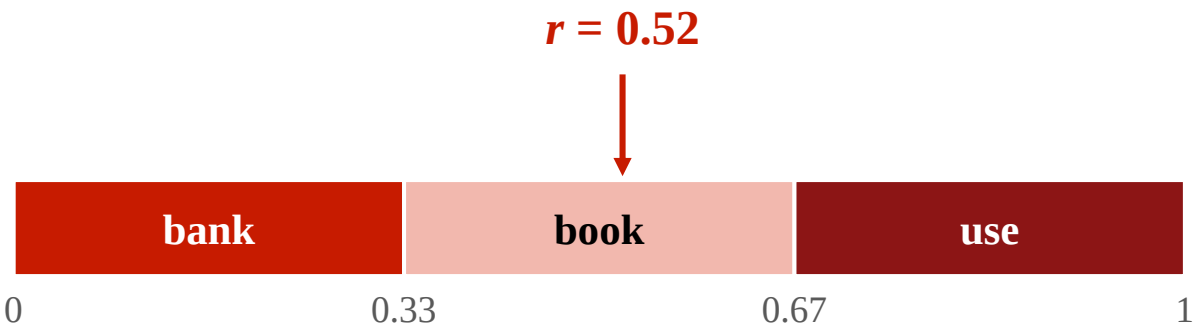
Generated: **the** **book**



click to spin

Next word ~ $P(w \mid \textbf{the})$

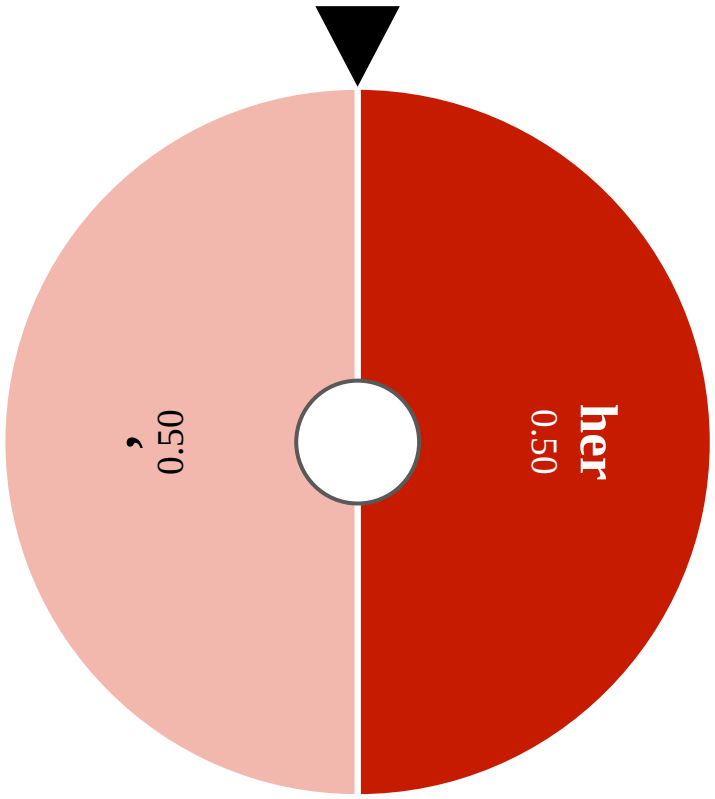
bank C(the bank) / C(the) = 1/3 = 0.33
book C(the book) / C(the) = 1/3 = 0.33
use C(the use) / C(the) = 1/3 = 0.33



r = 0.52 falls in [0.33, 0.67)
→ next word: **book**

Generating with a Bigram Model: Spin the Wheel (2/5)

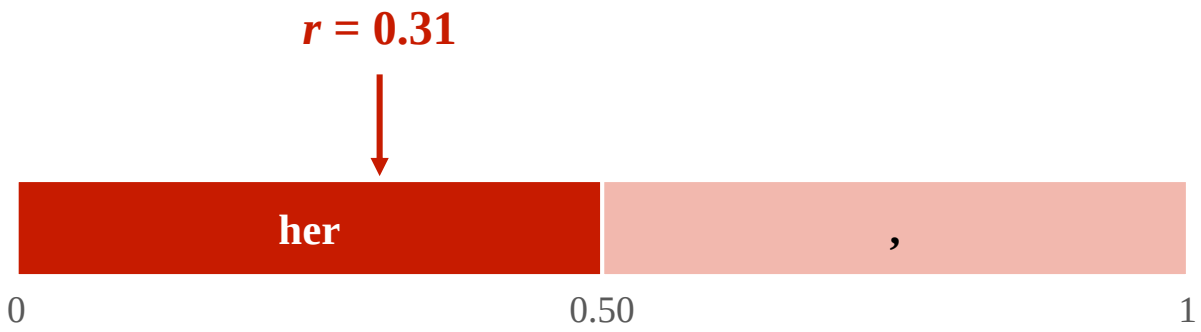
Generated: the | **book** | **her**



click to spin

Next word ~ $P(w \mid \textbf{book})$

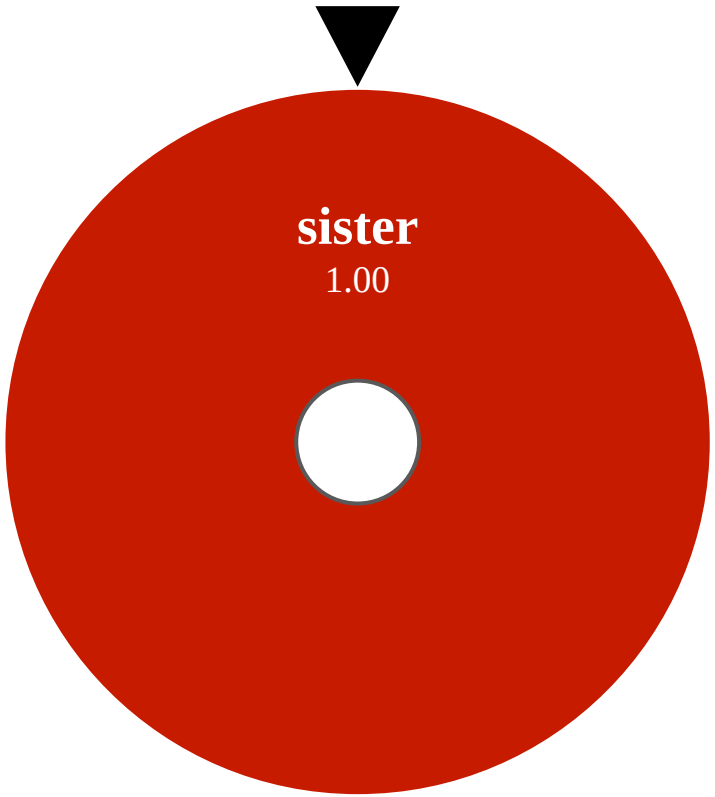
her C(book her) / C(book) = 1/2 = 0.50
, C(book ,) / C(book) = 1/2 = 0.50



r = 0.31 falls in [0.00, 0.50)
→ next word: **her**

Generating with a Bigram Model: Spin the Wheel (3/5)

Generated: the | book | **her** | **sister**



click to spin

Next word ~ $P(w \mid \textbf{her})$

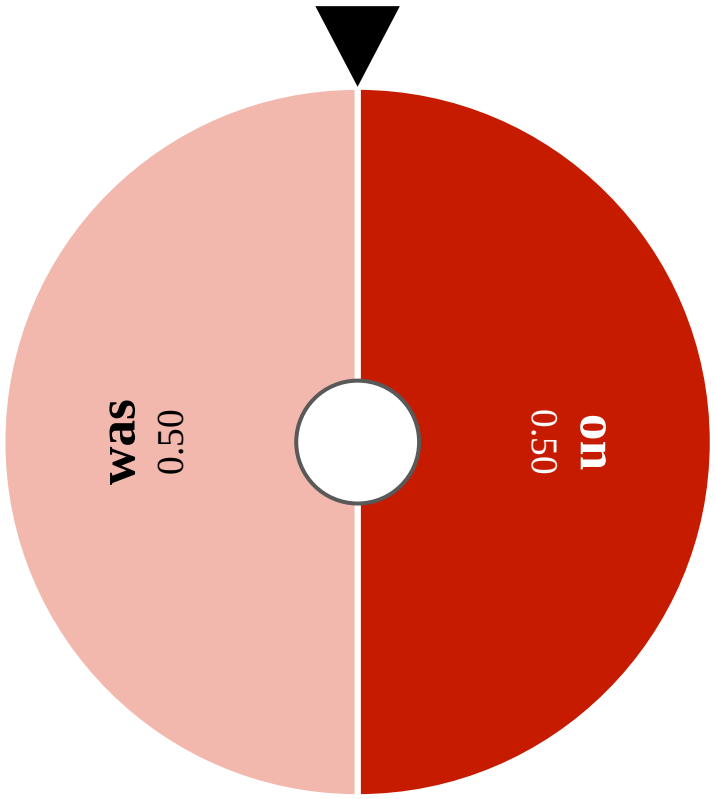
sister C(her sister) / C(her) = 2/2 = 1.00



r = 0.74 falls in [0.00, 1.00)
→ next word: **sister**

Generating with a Bigram Model: Spin the Wheel (4/5)

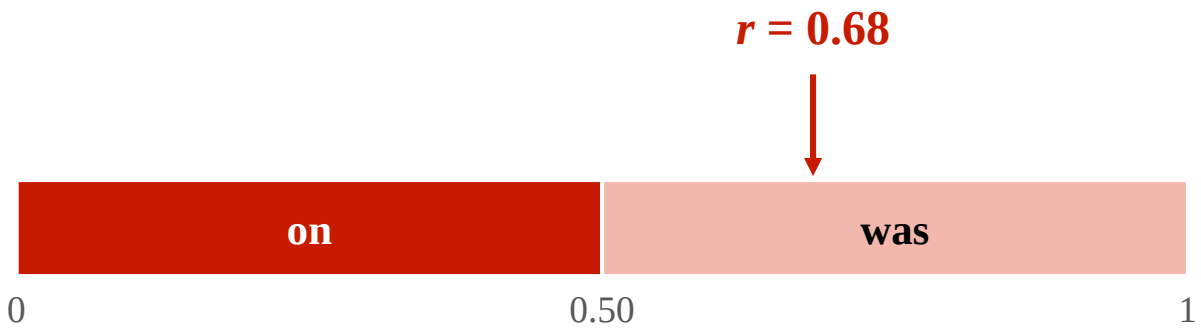
Generated: the | book | her | **sister** | **was** |



click to spin

Next word ~ $P(w \mid \textbf{sister})$

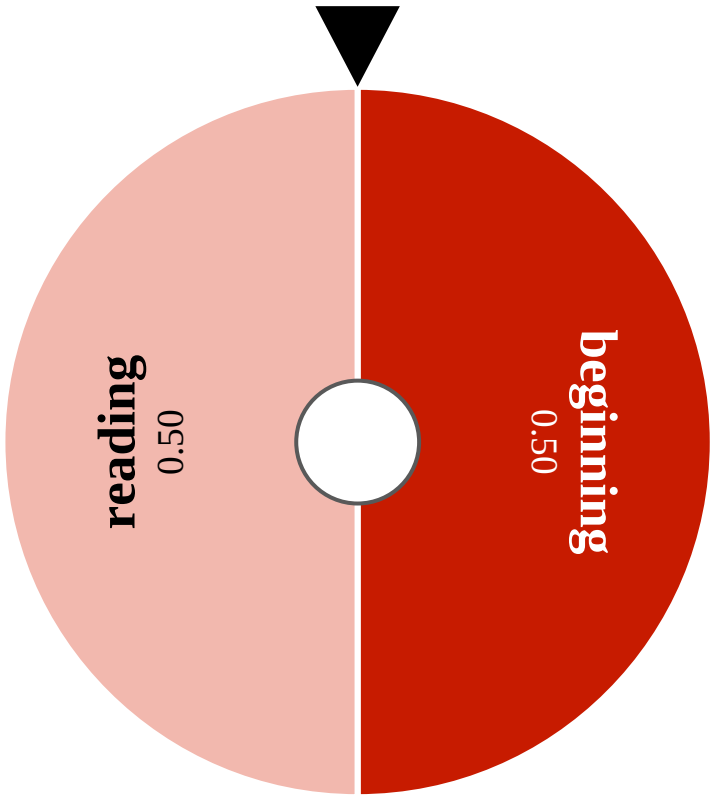
on C(sister on) / C(sister) = 1/2 = 0.50
was C(sister was) / C(sister) = 1/2 = 0.50



r = 0.68 falls in [0.50, 1.00)
→ next word: **was**

Generating with a Bigram Model: Spin the Wheel (5/5)

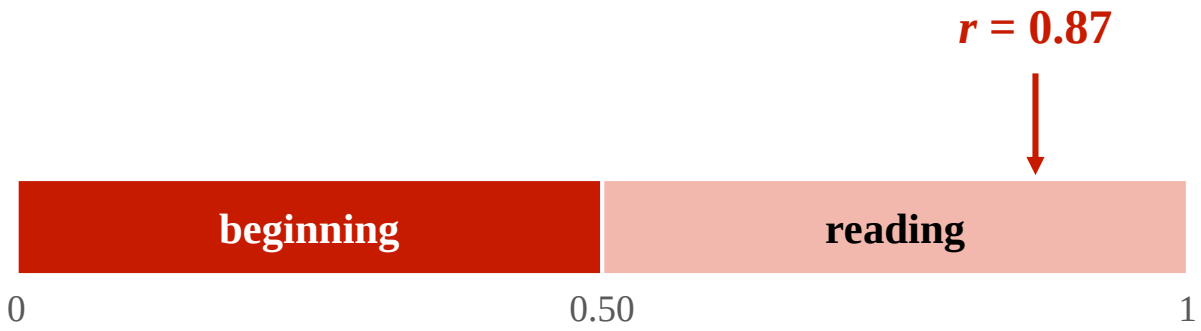
Generated: the | book | her | sister | **was** | **reading**



click to spin

Next word ~ $P(w \mid \textbf{was})$

beginning C(was beginning) / C(was) = 1/2 = 0.50
reading C(was reading) / C(was) = 1/2 = 0.50

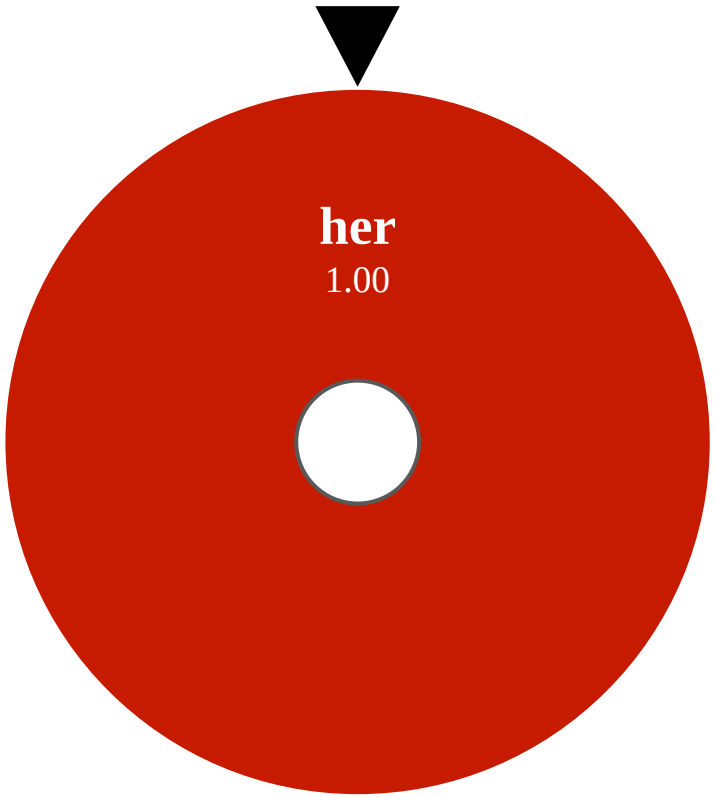


r = 0.87 falls in [0.50, 1.00)
→ next word: **reading**

Generated sentence: *"the book her sister was reading"* — every step is one spin of a different wheel P(w | previous word).

Generating with a Trigram Model: Spin the Wheel (1/6)

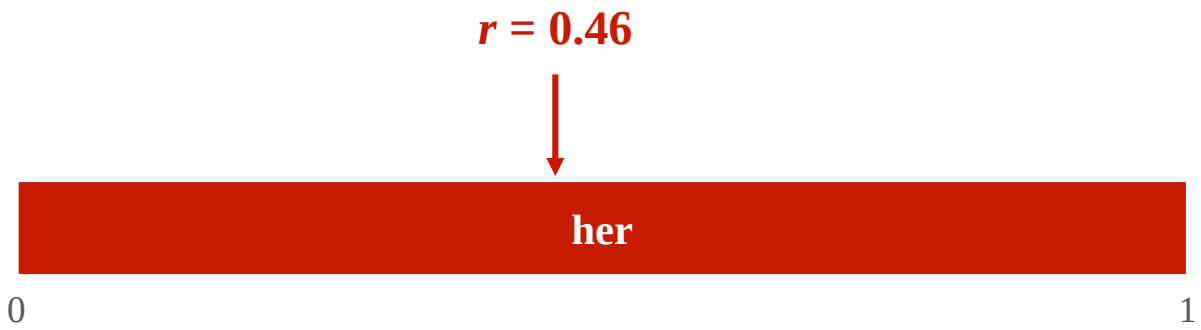
Generated: **the** **book** **her**



click to spin

Next word ~ $P(w \mid$ **the book**$)$

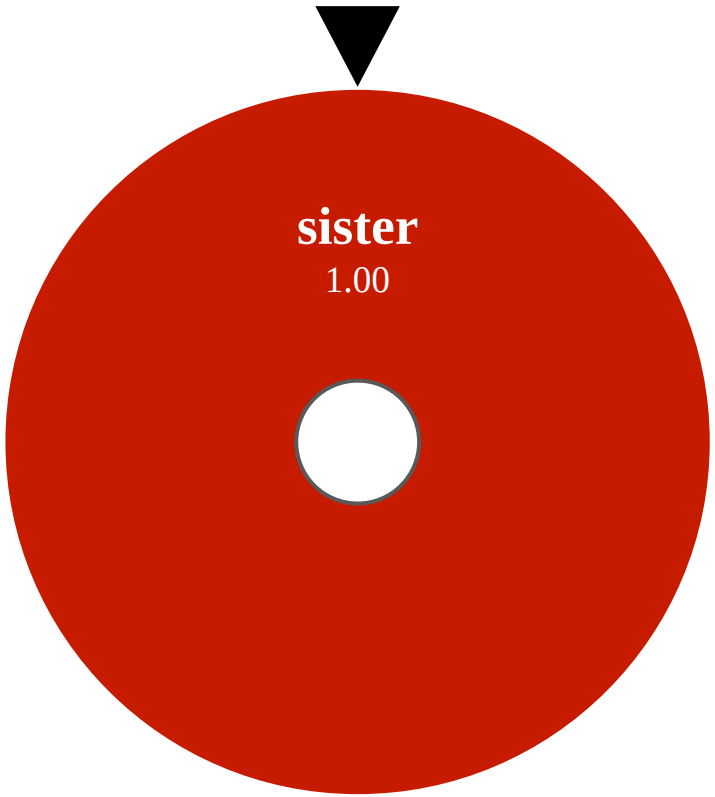
her C(the book her) / C(the book) = 1/1 = 1.00



r = 0.46 falls in [0.00, 1.00)
→ next word: **her**

Generating with a Trigram Model: Spin the Wheel (2/6)

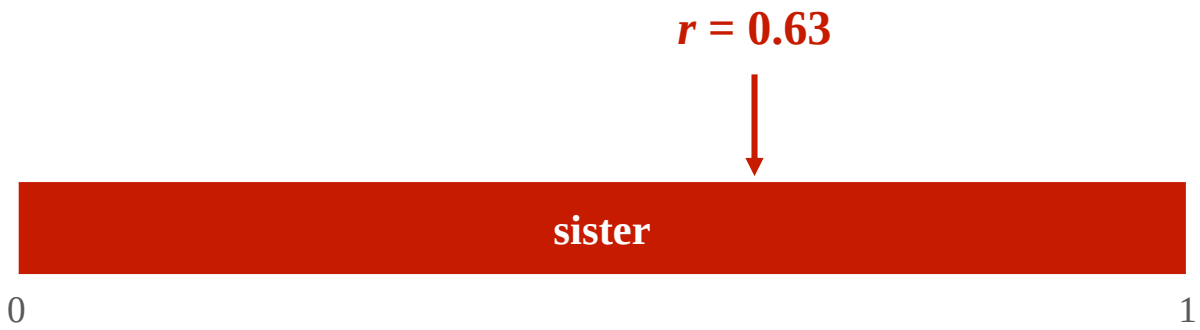
Generated: the | **book** | **her** | **sister**



click to spin

Next word ~ $P(w \mid$ **book her**$)$

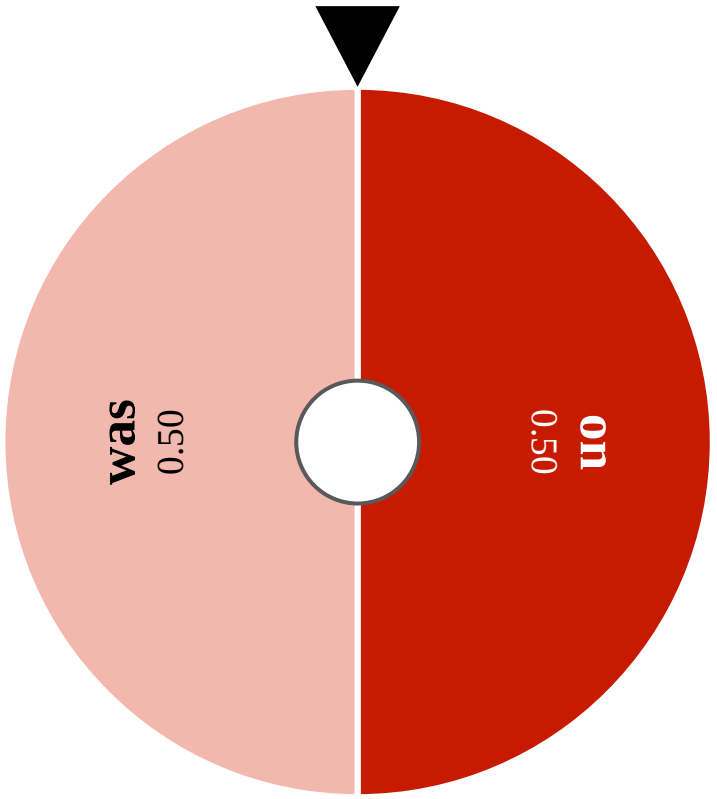
sister C(book her sister) / C(book her) = 1/1 = 1.00



r = 0.63 falls in [0.00, 1.00)
→ next word: **sister**

Generating with a Trigram Model: Spin the Wheel (3/6)

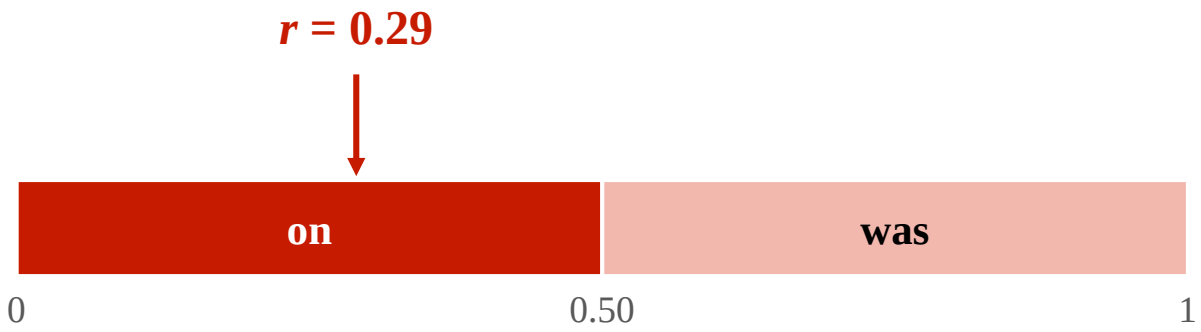
Generated: the | book | **her** | **sister** | **on**



click to spin

Next word ~ $P(w \mid$ **her sister**$)$

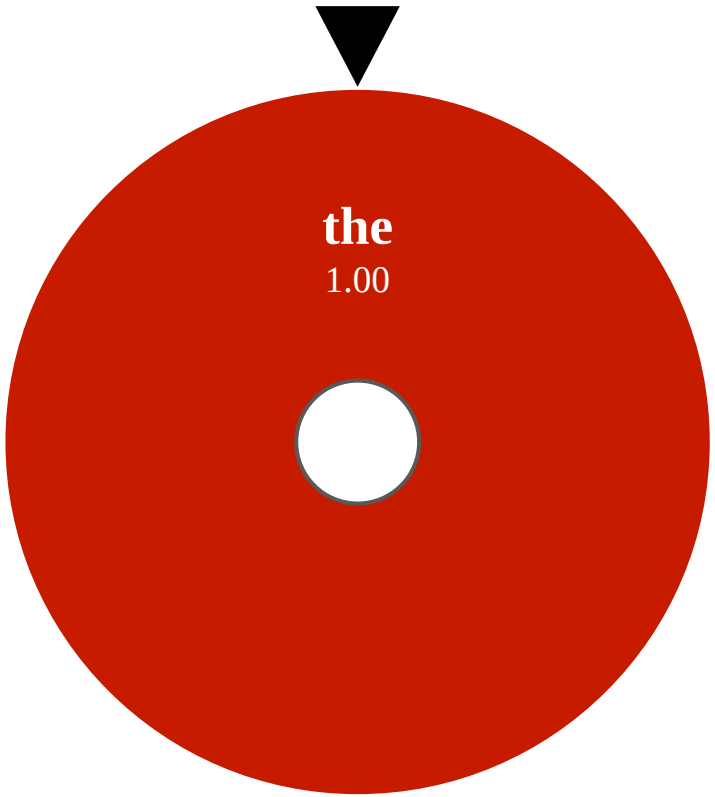
on C(her sister on) / C(her sister) = 1/2 = 0.50
was C(her sister was) / C(her sister) = 1/2 = 0.50



r = 0.29 falls in [0.00, 0.50)
→ next word: **on**

Generating with a Trigram Model: Spin the Wheel (4/6)

Generated: | the | book | her | **sister** | **on** | **the** | | |



click to spin

Next word ~ $P(w \mid$ **sister on**$)$

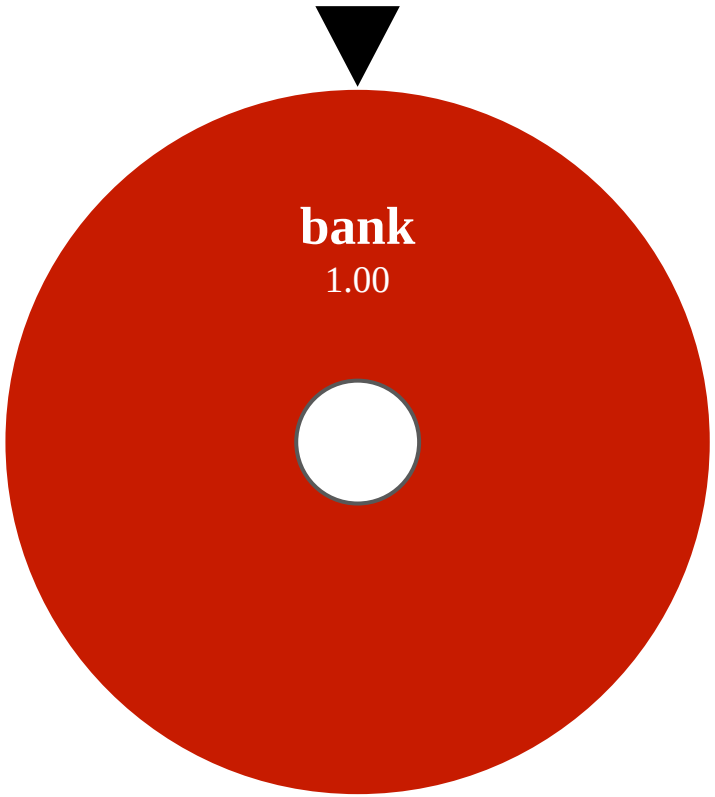
the C(sister on the) / C(sister on) = 1/1 = 1.00



r = 0.81 falls in [0.00, 1.00)
→ next word: **the**

Generating with a Trigram Model: Spin the Wheel (5/6)

Generated: | the | book | her | sister | **on** | **the** | **bank** |



click to spin

Next word ~ $P(w \mid \textbf{on the})$

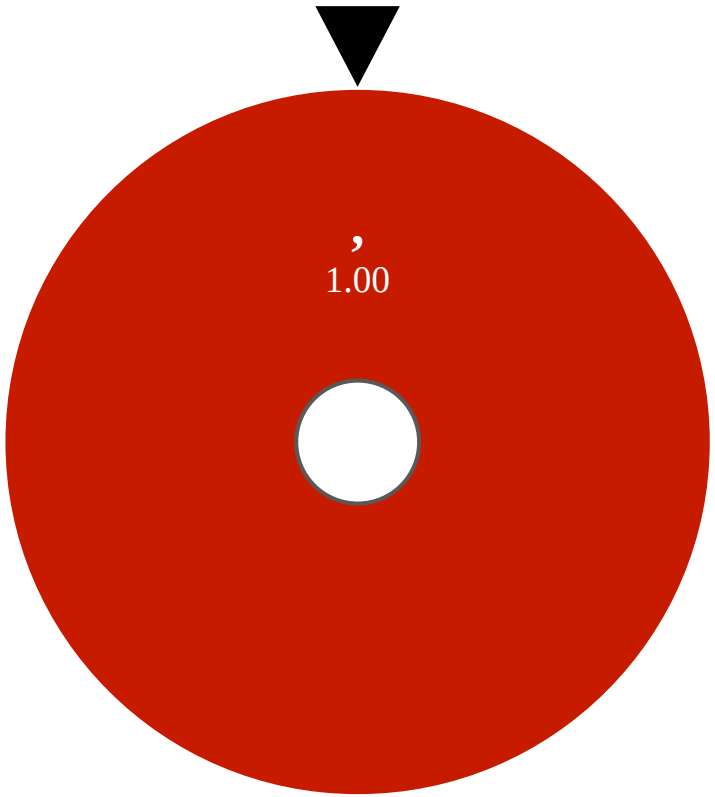
bank C(on the bank) / C(on the) = 1/1 = 1.00



r = 0.12 falls in [0.00, 1.00)
→ next word: **bank**

Generating with a Trigram Model: Spin the Wheel (6/6)

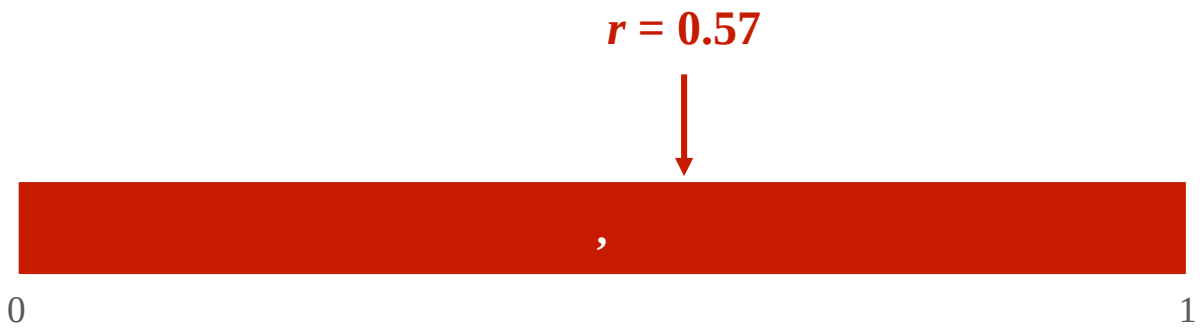
Generated: the | book | her | sister | on | **the** | **bank** | **,**



click to spin

Next word ~ $P(w \mid \textbf{the bank})$

, C(the bank ,) / C(the bank) = 1/1 = 1.00



r = 0.57 falls in [0.00, 1.00)
→ next word: **,**

Generated: *"the book her sister on the bank,"* — a new sentence, yet 5 of the 6 wheels had a single slice: larger n, sparser counts.

Generating the Wall Street Journal

unigram: Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

bigram: 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

trigram: 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

Generating Shakespeare

Unigram	• To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have • Every enter now severally so, let • Hill he late speaks; or! a more to leg less first you enter • Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like
Bigram	• What means, sir. I confess she? then all sorts, he is trim, captain. •Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. •What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman? •Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt
Trigram	• Sweet prince, Falstaff shall die. Harry of Monmouth's grave. • This shall forbid it should be branded, if renown made it empty. • Indeed the duke; and had a very good friend. • Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.
Quadrigram	• King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; • Will you not tell me who I am? • It cannot be but so. • Indeed the short and the long. Marry, 'tis a noble Lepidus.

What is the drawback of a large N for a N-gram LM?

Space complexity
Less diversity?

Shakespeare as Corpus

The Shakespeare corpus consists of N=884,647 word **tokens** and a vocabulary of V=29,066 word **types**

Shakespeare produced 300,000 bigram types out of V^2= 844 million possible bigram types.

99.96% of possible bigrams don't occur in the corpus.

Our relative frequency estimate assigns non-zero probability to only 0.04% of the possible bigrams That percentage is even lower for trigrams, 4-grams, etc.

Use data from https://huggingface.co/datasets/Trelis/tiny-shakespeare or https://raw.githubusercontent.com/karpathy/char-rnn/master/data/tinyshakespeare/input.txt

MLE Doesn't Capture Unseen Events

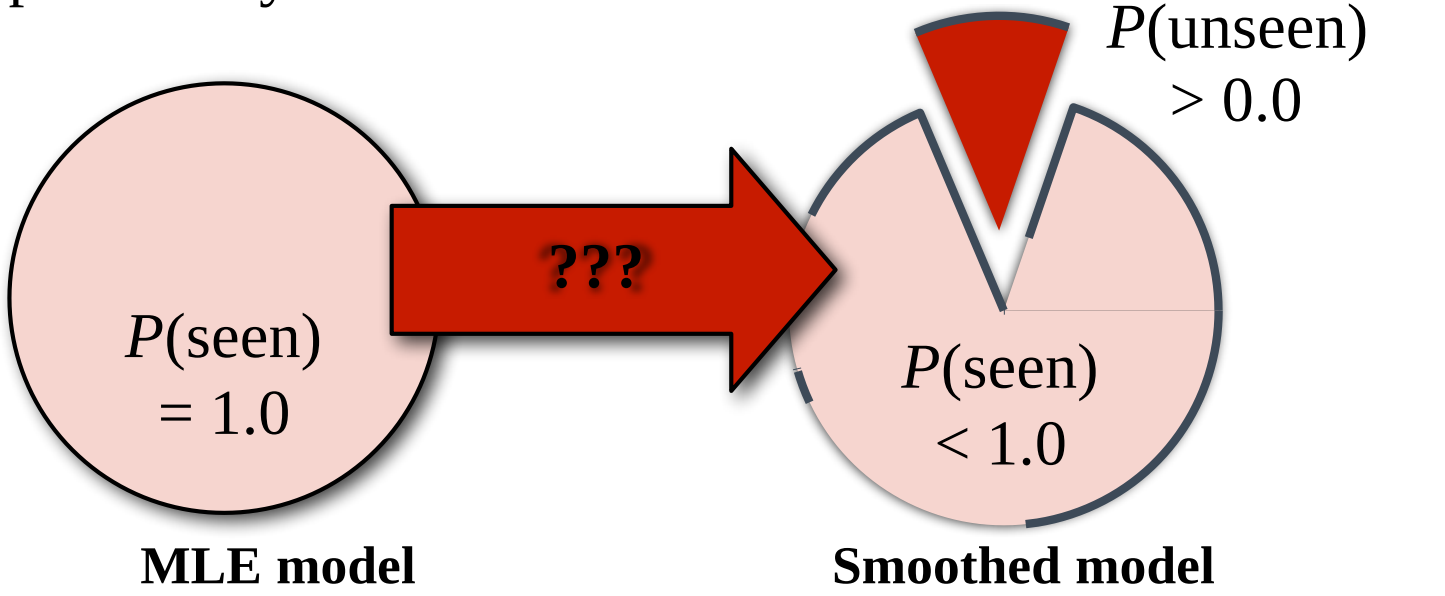
We estimated a model on 440K word tokens, but:

Fact 1: *Only 30,000 word types occur in the training data.*
Any word that does not occur in the training data has zero probability!

Fact 2: *Only 0.04% of all possible bigrams (over 30K word types) occur in the training data*
Any bigram that does not occur in the training data has *zero* probability (even if we have seen both words in the bigram)

How Do We Assign Non-zero Probability to Unseen Events?

We have to “smooth” our distributions to assign some probability mass to unseen events



Smoothing Methods

Add-one smoothing:
Hallucinate counts that didn't occur in the data

Linear interpolation:
$\tilde{P}(w \mid w', w'') = \lambda \hat{P}(w \mid w', w'') + (1-\lambda)\tilde{P}(w \mid w')$
Interpolate n-gram model with (n–1)-gram model.

Absolute Discounting: Subtract constant count from frequent events and add it to rare events

Kneser-Ney: AD with modified unigram probabilities

Add-One (Laplace) Smoothing

A really simple way to do smoothing:
Increment the actual observed count of every *possible* event (e.g. bigram) by a hallucinated count of 1 (or by a hallucinated count of some k with 0<k<1).

Shakespeare bigram model (roughly):

0.88 million actual bigram counts + 844.xx million hallucinated bigram counts

Oops. Now almost none of the counts in our model come from actual data. We're back to word salad.

K needs to be really small. But it turns out that that still doesn't work very well.

Evaluation

Intrinsic vs Extrinsic Evaluation

How do we know whether one language model is better than another?

There are two ways to evaluate models:

- intrinsic evaluation captures how well the model captures what it is supposed to capture (e.g. probabilities)
- extrinsic (task-based) evaluation captures how useful the model is in a particular task.

Both cases require an evaluation metric that allows us to measure and compare the performance of different models.

Intrinsic Evaluation of Language Models: Perplexity

Perplexity

The perplexity of a language model is the inverse of the probability of the test set, normalized by the number of tokens N in the test set. If a LM assigns probability $P(w_1, \ldots, w_N)$ to a test corpus $w_1 \ldots w_N$, the LM's perplexity, $PP(w_1 \ldots w_N)$, is

$$ppl(w_1, w_2, \cdots w_n) = \sqrt[n]{\frac{1}{p(w_1, \ldots, w_N)}}$$

A LM with lower perplexity is better because it assigns a higher probability to the unseen test corpus.

LM_1 and LM_2's perplexity can only be compared if they use the same vocabulary

- Trigram models have lower perplexity than bigram models; Bigram models have lower perplexity than unigram models, etc.

Practical Issues

- Since language model probabilities are very small, multiplying them together often yields to underflow.
- It is often better to use logarithms instead, so replace

$$PP(w_1 \ldots w_N) \stackrel{\text{def}}{=} \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_{i-1},\ldots,w_{i-n+1})}}$$

with

$$PP(w_1 \ldots w_N) \stackrel{\text{def}}{=} \exp\left(-\frac{1}{N} \sum_{i=1}^{N} \log P(w_i \mid w_{i-1},\ldots,w_{i-n+1}) \right)$$

Extrinsic (Task-Based) Evaluation of LMs: Word Error Rate

Intrinsic vs. Extrinsic Evaluation

Perplexity tells us which LM assigns a higher probability to unseen text

This doesn't necessarily tell us which LM is better for our task (i.e. is better at scoring candidate sentences)

Task-based evaluation:

- -Train model A, plug it into your system for performing task T
- -Evaluate performance of system A *on task T*.
- -Train model B, plug it in, evaluate system B on same task T.
- -Compare scores of system A and system B on task T.

Language Models for Downstream Tasks

Independently of any application, we can use a language model as a random sentence generator (i.e we sample sentences according to their language model probability)

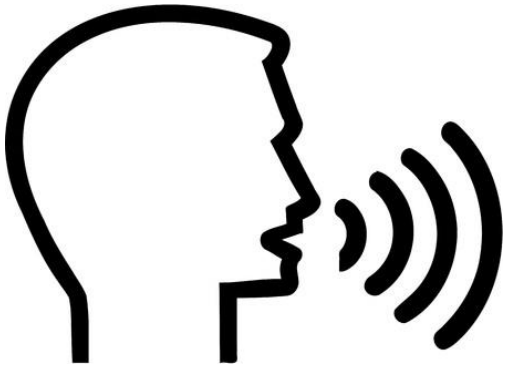
Systems for applications such as machine translation, speech recognition, spell-checking, generation, often produce **multiple candidate** sentences as output.

- We prefer output sentences S_{Out} that have a higher probability
- We can use a language model $P(S_{Out})$ to score and rank these different candidate output sentences, e.g. as follows:

$$\text{Argmax}_{SOut}\, P(S_{Out} \mid \text{Input}) = \text{Argmax}_{SOut}\, P(\text{Input} \mid S_{Out}) P(S_{Out})$$

Example: language model in information retrieval.

An Example of Using Language Models in ASR



这儿有**周杰伦**演唱会（There is a Jay Chou concert!）

- Acoustic Model: **周杰轮？ 周捷伦？**
- Acoustic Model + language models: **周杰伦**

Word Error Rate (WER)

Originally developed for speech recognition.

How much does the *predicted* sequence of words differ from the *actual* sequence of words in the correct transcript?

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

Insertions: "eat lunch" → "eat ***a*** lunch"
Deletions: "see ***a*** movie" → "see movie"
Substitutions: "drink ***ice*** tea" → "drink ***nice*** tea"

Which One is Better ? Using Language Models in MT

I think therefore I am

Translator

我思故我在

我思，故我在。

我思考，所以我存在。

思，则在。

思考让我成为“我”。

Let the Language Model Decide

I think therefore I am

Translator

我思故我在

我思，故我在。

我思考，所以我存在。

思，则在。

思考让我成为“我”。

Language models

Numeric results

High
Middle-high
Low

For example，我思故我在 is with the highest probability.

It is most fluent, not necessarily 信达雅

To Recap….

Key Concepts in Summary

- N-gram language models
  - Independence assumptions
  - Getting from n-grams to a distribution over a language
  - Relative frequency (maximum likelihood) estimation Smoothing
  - Intrinsic evaluation: Perplexity, Extrinsic evaluation: WER

Today's Lecture

- Language Models in a Nutshell
- Why Language Models?
- Language Models in a *Narrow* Sense
- **Language Models in a Broad Sense**
  (N-gram, Word2Vec, BERT and beyond)

N-gram Language Models

the students opened their

- **Question**: How to learn a Language Model?
- **Answer** (pre- Deep Learning): learn an *n*-gram Language Model!

- **Definition:** An *n*-gram is a chunk of *n* consecutive words.
  - unigrams: “the”, “students”, “opened”, ” their”
  - bigrams: “the students”, “students opened”, “opened their”
  - trigrams: “the students opened”, “students opened their”
  - four-grams: “the students opened their”

- **Idea:** Collect statistics about how frequent different n-grams are and use these to predict next word.

N-gram Language Models

- First we make a Markov assumption: $x^{(n)}$ depends only on the preceding n-1 words

$$P\left(\mathbf{x}^{(t+1)} \mid \mathbf{x}^{(t)},\ldots,\mathbf{x}^{(1)}\right) = P\left(\mathbf{x}^{(t+1)} \mid \mathbf{x}^{(t)},\ldots,\mathbf{x}^{(t-n+2)}\right) \quad \text{(Markov assumption)}$$

$$= \frac{P\left(\mathbf{x}^{(t+1)},\mathbf{x}^{(t)},\ldots,\mathbf{x}^{(t-n+2)}\right)}{P\left(\mathbf{x}^{(t)},\ldots,\mathbf{x}^{(t-n+2)}\right)} \quad \text{(definition of conditional prob.)}$$

- **Question:** How do we get these n-gram and (n-1)-gram probabilities?
- **Answer:** By counting them in some large corpus of text!

$$\approx \frac{\text{count}\left(\mathbf{x}^{(t+1)},\mathbf{x}^{(t)},\ldots,\mathbf{x}^{(t-n+2)}\right)}{\text{count}\left(\mathbf{x}^{(t)},\ldots,\mathbf{x}^{(t-n+2)}\right)} \quad \text{(statistical approximation)}$$

N-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

as the proctor started the clock, the students opened their ____

discard

condition on this

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - P(exams | students opened their) = 0.1

Sparsity Problems with N-gram Language Models

Sparsity Problem 1

Problem: What if *"students opened their w"* never occurred in data? Then *w* has probability 0!

(Partial) Solution: Add small δ to the count for every $w \in V$. This is called *smoothing*.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any w*!

(Partial) Solution: Just condition on *"opened their"* instead. This is called *backoff*.

Note: Increasing n makes sparsity problems worse. Typically, we can't have n bigger than 5.

Storage Problems with N-gram Language Models

Storage: Need to store count for all *n*-grams you saw in the corpus.

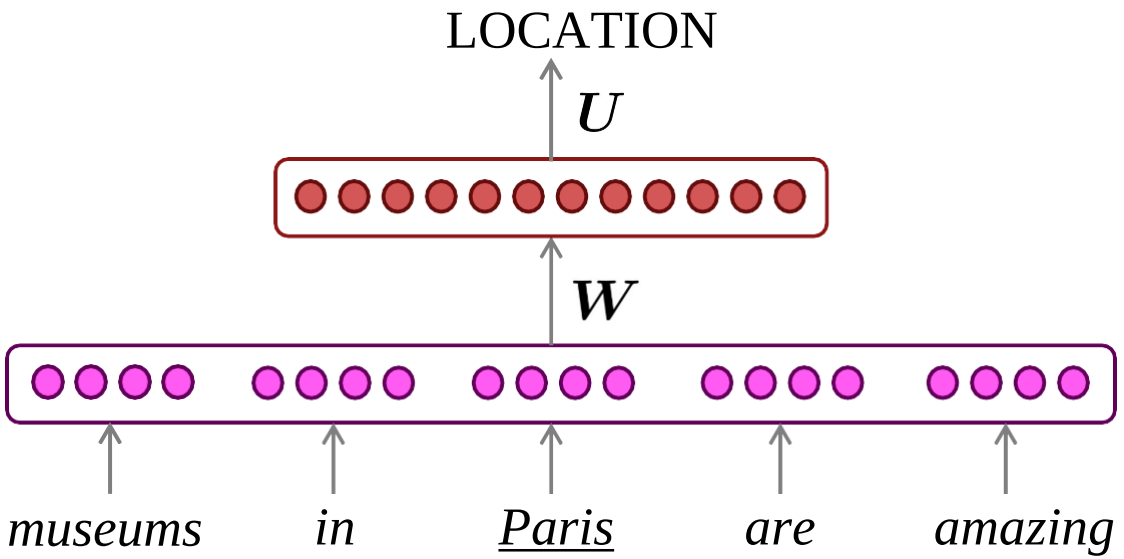
$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

Increasing *n* or increasing corpus increases model size!

How to Build a *Neural* Language Model?

- Recall the Language Modeling task:
  - Input: sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$
  - Output: prob. dist. of the next word $P(\boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$

- How about a window-based neural model?
  - We saw this applied to Named Entity Recognition :



A Fixed-window Neural Language Model

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

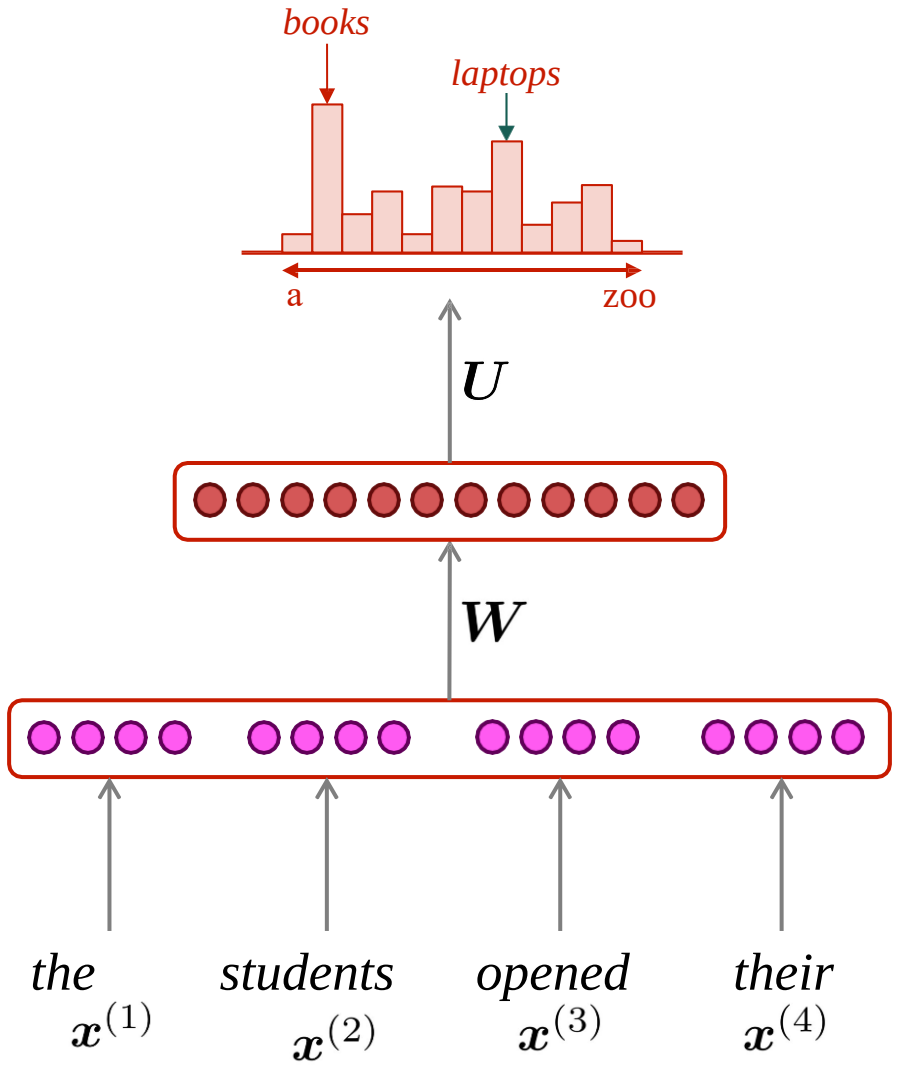
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



A Fixed-window Neural Language Model

Approximately: Y. Bengio, et al. (2000/2003): A Neural Probabilistic Language Model

Improvements over *n*-gram LM:

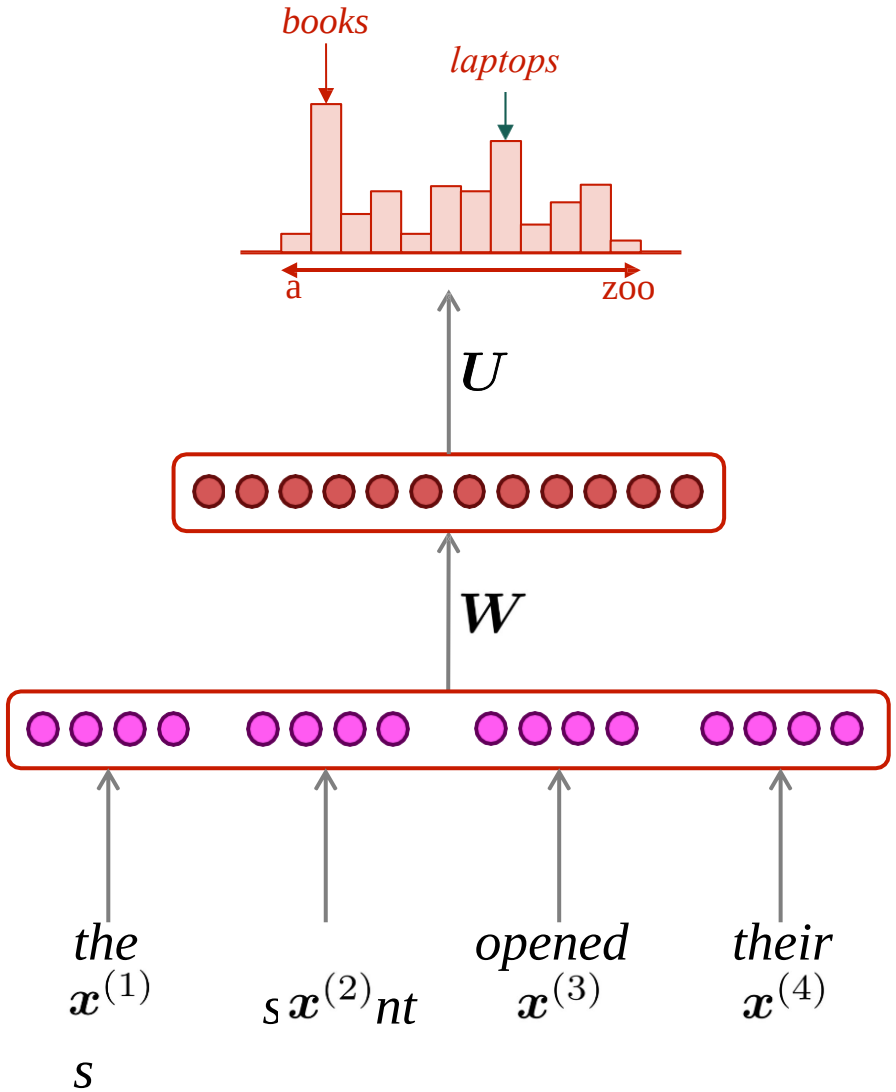
- No sparsity problem
- Don't need to store all observed *n*-grams

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges *W*
- Window can never be large enough!
- $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in *W*. No symmetry in how the inputs are processed.

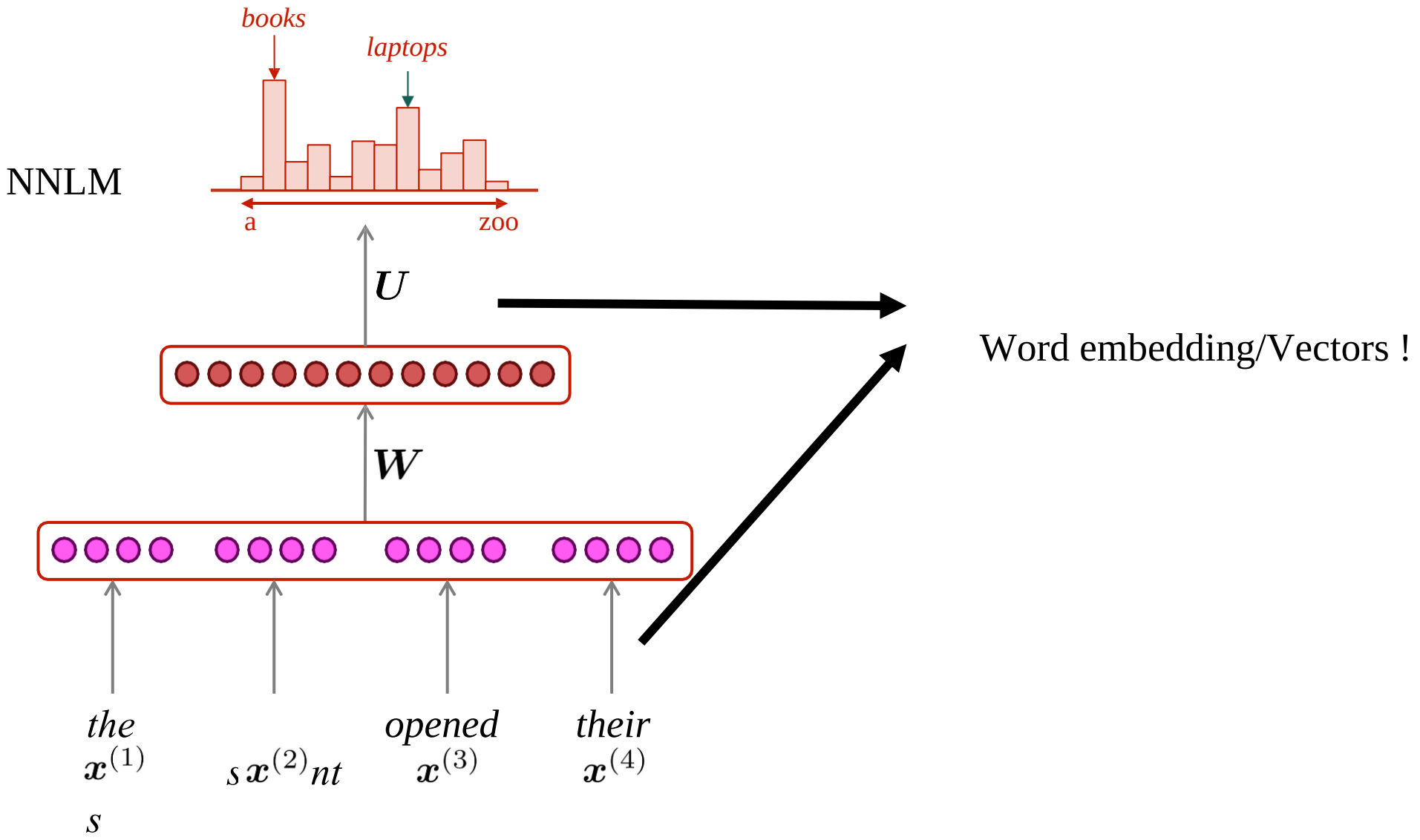
We need a neural architecture that can process *any length input*

Recurrent NN is the solution !



From N-gram LMs to Word Vectors

By-products of NNLM: Word Embedding



Representing Words by Their Context

Distributional semantics: **A word’s meaning is given by the words that frequently appear close-by**

- “*You shall know a word by the company it keeps*” (J. R. Firth 1957: 11)
- **One of the most successful ideas of modern statistical NLP!**

- When a word *w* appears in a text, its context is the set of words that appear nearby (within a fixed-size window).
- We use the many contexts of w to build up a representation of *w*

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*



These context words will represent ***banking***

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture01-wordvecs1.pdf

Word2vec: Objective Function

- We want to minimize the objective function:

$$J(\theta)=-\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m,\ j\neq 0}\log P\left(w_{t+j}\mid w_t;\theta\right)$$

- Question: How to calculate $P\left(w_{t+j}\mid w_t;\theta\right)$

Answer: We will use two vectors per word *w*:

- v_w when w is a center word
- u_w when w is a context word

Then for a center word *c* and *a* context word *o:* (softmax)

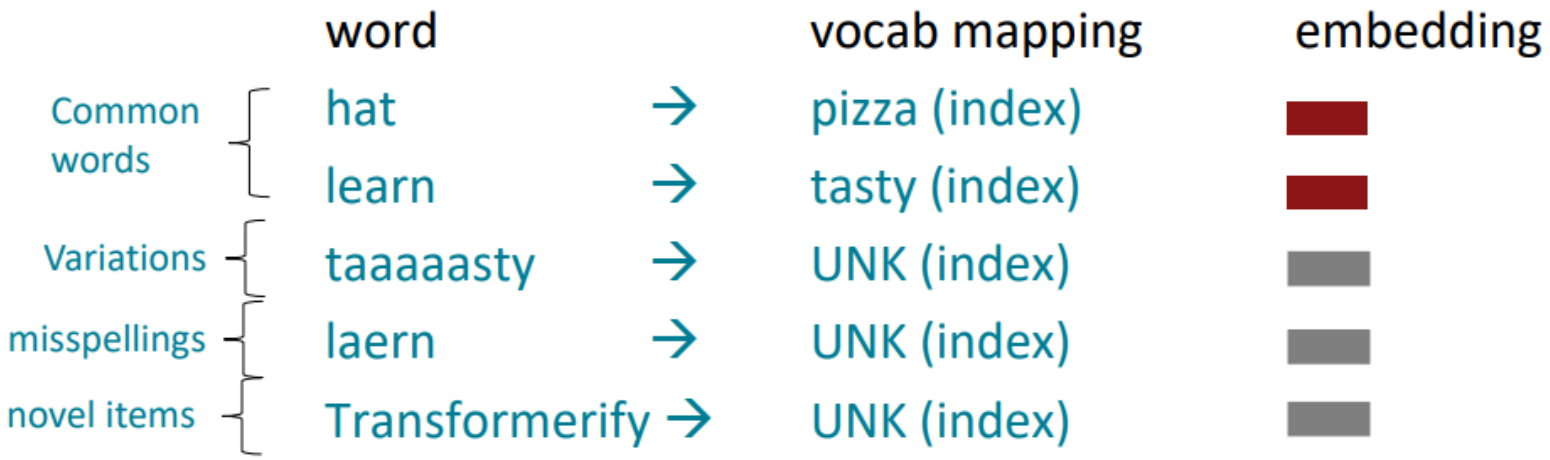
$$P(o\mid c)=\frac{\exp\left(u_o^T v_c\right)}{\sum_{w\in V}\exp\left(u_w^T v_c\right)}$$

"max" because amplifies probability of largest
"soft" because still assigns some probability to smaller

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture01-wordvecs1.pdf

Word Structure and Subword Models

We assume a fixed vocab of tens of thousands of words, built from the training set. All novel words seen at test time are mapped to a single UNK.



Finite vocabulary assumptions make even less sense in many languages.

- Many languages exhibit complex morphology, or word structure.
- The effect is more word types, each occurring fewer times.

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture9-pretraining.pdf

Interesting Characters/Words

- 忝 《广韵》《集韵》并以冉切，音琰 (yan3)。物上大下小也。又《集韵》他刀切，音叨(tao1)。进也。
- LGUer
- Looong

looongLLaVA: Scaling Multi-modal LLMs to 1000 Images Efficiently via a Hybrid Architecture

Xidong Wang†, Dingjie Song†, Shunian Chen, Chen Zhang, Benyou Wang*
The Chinese University of Hong Kong, Shenzhen
Shenzhen Research Institute of Big Data
https://github.com/FreedomIntelligence/LongLLaVA

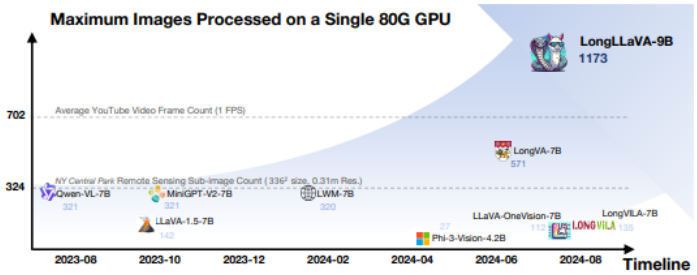


Figure 1: Comparison of the maximum images processed by MLLMs on a single 80GB GPU (Int8 Quantization), and plotted against their release dates. Our model, LongLLaVA, leads the way with the ability to handle up to 1173 images, demonstrating its superior processing capability. Res refers to resolution. Although these baseline models are capable of processing these images as input, their performance often deteriorates significantly (Song et al., 2024) with more images.

A Paper from Our Group: MorphTE

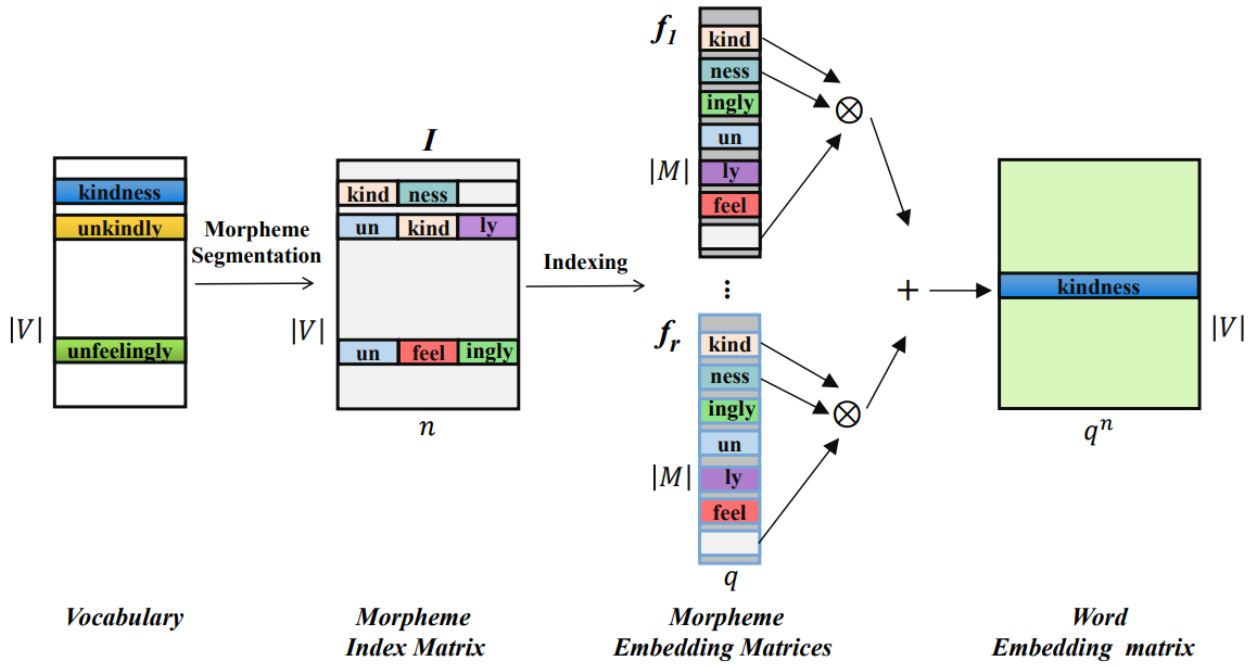


Figure 3: The workflow of MorphTE. ***n*** is the order (number of morphemes for a word). ***q*** is the size of morpheme vectors. |***V***| and |***M***| denote the size of word vocabulary and morpheme vocabulary.

Guobing Gan, Peng Zhang, Sunzhu Li, Xiuqing Lu, **Benyou Wang.** MorphTE: Injecting Morphology in Tensorized Embeddings. NeurIPS 2022

From Static Word Vector to Contextualized Word Vectors

What's Wrong with Word2vec?

One vector for each word type

$$v(\text{bank}) = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix}$$

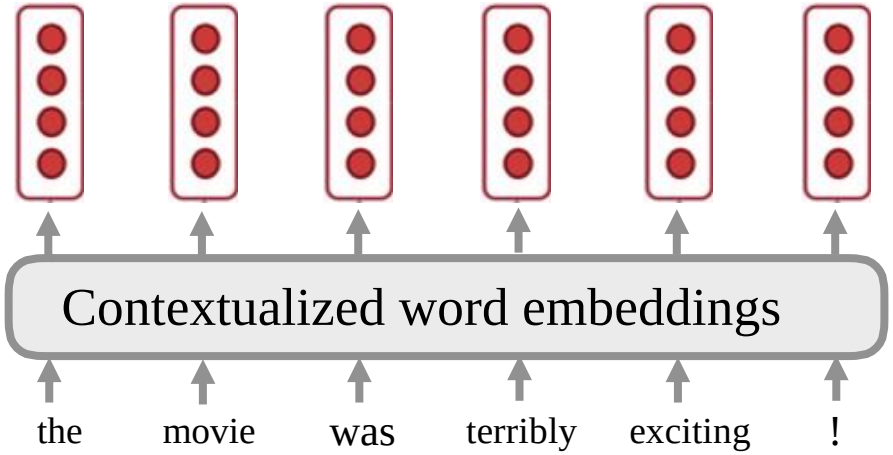
Complex characteristics of word use: semantics, syntactic behavior, and connotations

Polysemous words, e.g., bank, mouse

mouse^{1} : .... a *mouse* controlling a computer system in 1968.
mouse^{2} : .... a quiet animal like a *mouse*
bank^{1} : ...a *bank* can hold the investments in a custodial account ...
bank^{2} : ...as agriculture burgeons on the east *bank*, the river ...

Contextualized Word Embeddings

Let's build a vector for each word conditioned on its **context**!



$$f : (w_1, w_2, \ldots, w_n) \longrightarrow \mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$$

ELMo

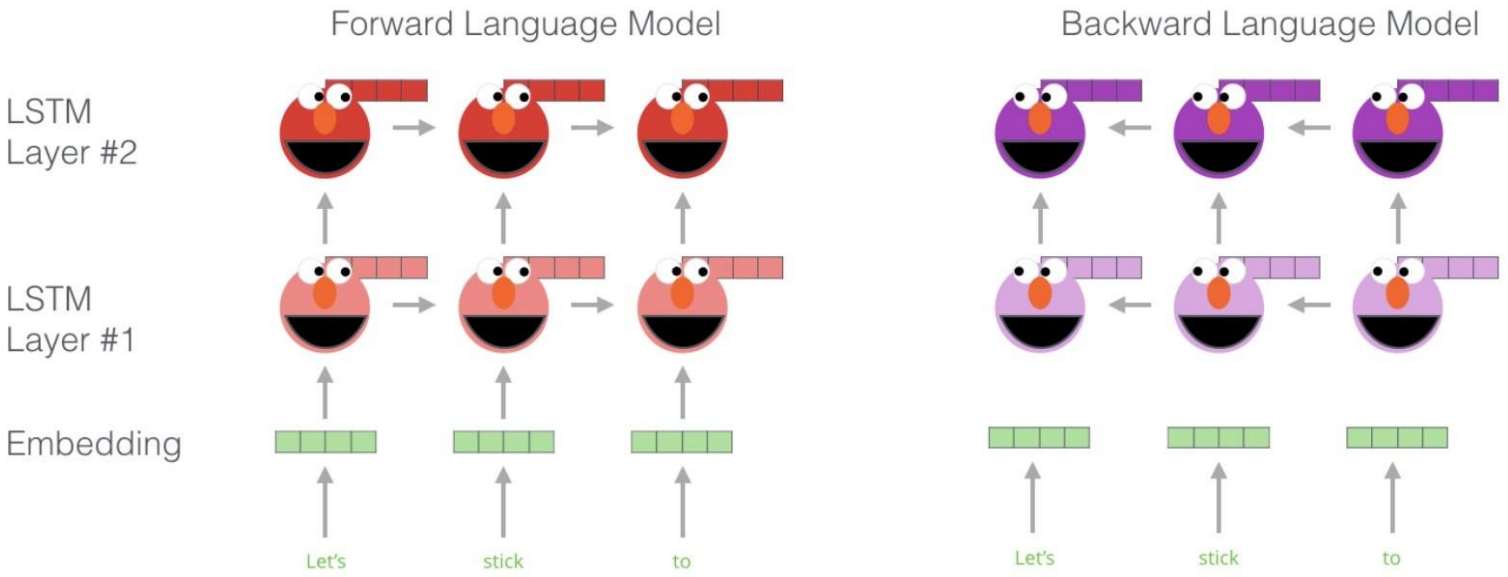
NAACL'18: Deep contextualized word representations

Key idea:

- Train an LSTM-based language model on some large corpus
- Use the hidden states of the LSTM for each token to compute a vector representation of each word



ELMo



$$-\sum_{k=1}^{N} \left(\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s) \right)$$

words in the sentence

input

softmax

How to Use ELMo?

$$
\begin{aligned}
R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\
&= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\},
\end{aligned}
$$

← # of layers

$$
\mathbf{h}_{k,0}^{lM} = \mathbf{x}_k^{LM}, \mathbf{h}_{k,j}^{LM} = [\overrightarrow{\mathbf{h}}_{k,j}^{LM}; \overleftarrow{\mathbf{h}}_{k,j}^{LM}]
$$

$$
\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}
$$

- γ^{task}: allows the task model to scale the entire ELMo vector
- s_j^{task}: softmax-normalized weights across layers
- Plug ELMo into any (neural) NLP model: freeze all the LMs weights and change the input representation to:

$$
[\mathbf{x}_k; \mathbf{ELMo}_k^{task}]
$$

(could also insert into higher layers)

Use ELMo in Practice

https://allennlp.org/elmo

Pre-trained ELMo Models

Model	Link(Weights/Options File)		# Parameters (Millions)	LSTM Hidden Size/Output size	# Highway Layers>
Small	weights	options	13.6	1024/128	1
Medium	weights	options	28.0	2048/256	1
Original	weights	options	93.6	4096/512	2
Original (5.5B)	weights	options	93.6	4096/512	2

```
from allennlp.modules.elmo import Elmo, batch_to_ids

options_file = "https://allennlp.s3.amazonaws.com/models/elmo/2x409
weight_file = "https://allennlp.s3.amazonaws.com/models/elmo/2x4096_

# Compute two different representation for each token.
# Each representation is a linear weighted combination for the
# 3 layers in ELMo (i.e., charcnn, the outputs of the two BiLSTM))
elmo = Elmo(options_file, weight_file, 2, dropout=0)

# use batch_to_ids to convert sentences to character ids
sentences = [['First', 'sentence', '.'], ['Another', '.']]
character_ids = batch_to_ids(sentences)

embeddings = elmo(character_ids)
```

Also available in TensorFlow

BERT

First released in Oct 2018.

NAACL'19: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

How is BERT different from ELMo?

#1. Unidirectional context vs bidirectional context

#2. LSTMs vs Transformers (will talk later)

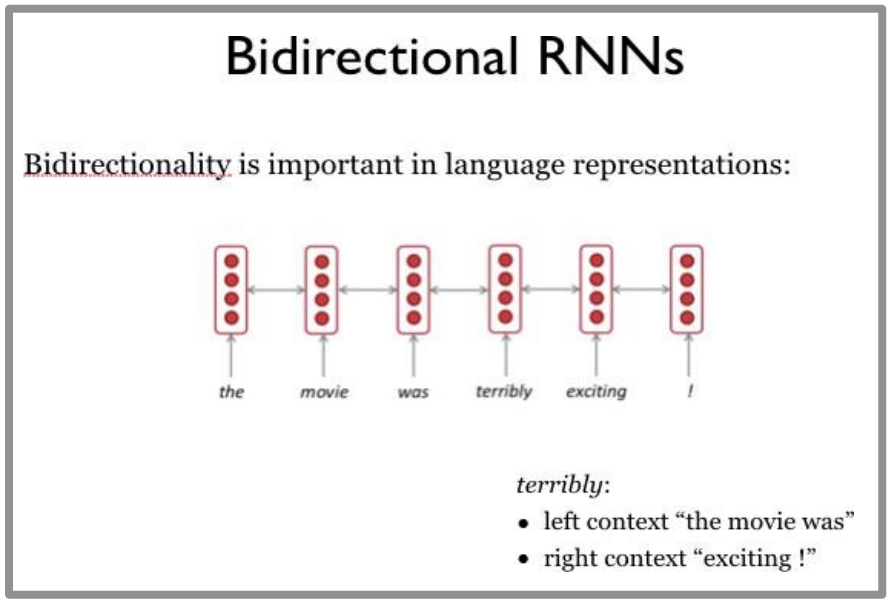
#3. The weights are not frozen, called fine-tuning



Bidirectional Encoders

Language models only use left context or right context (although ELMo used two independent LMs from each direction).

Language understanding is bidirectional

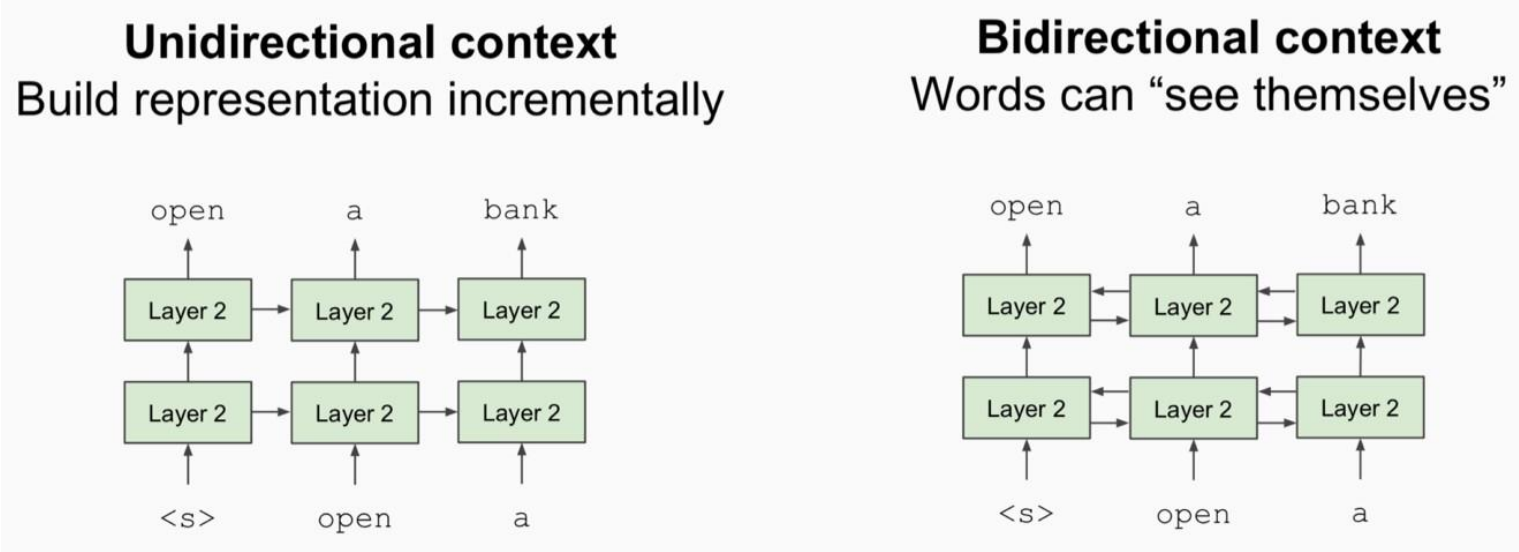


Why are LMs unidirectional?

Bidirectional Encoders

Language models only use left context or right context (although ELMo used two independent LMs from each direction).

Language understanding is bidirectional



Objective I: Masked Language Models (MLMs)

Solution: Mask out 15% of the input words, and then predict the masked words



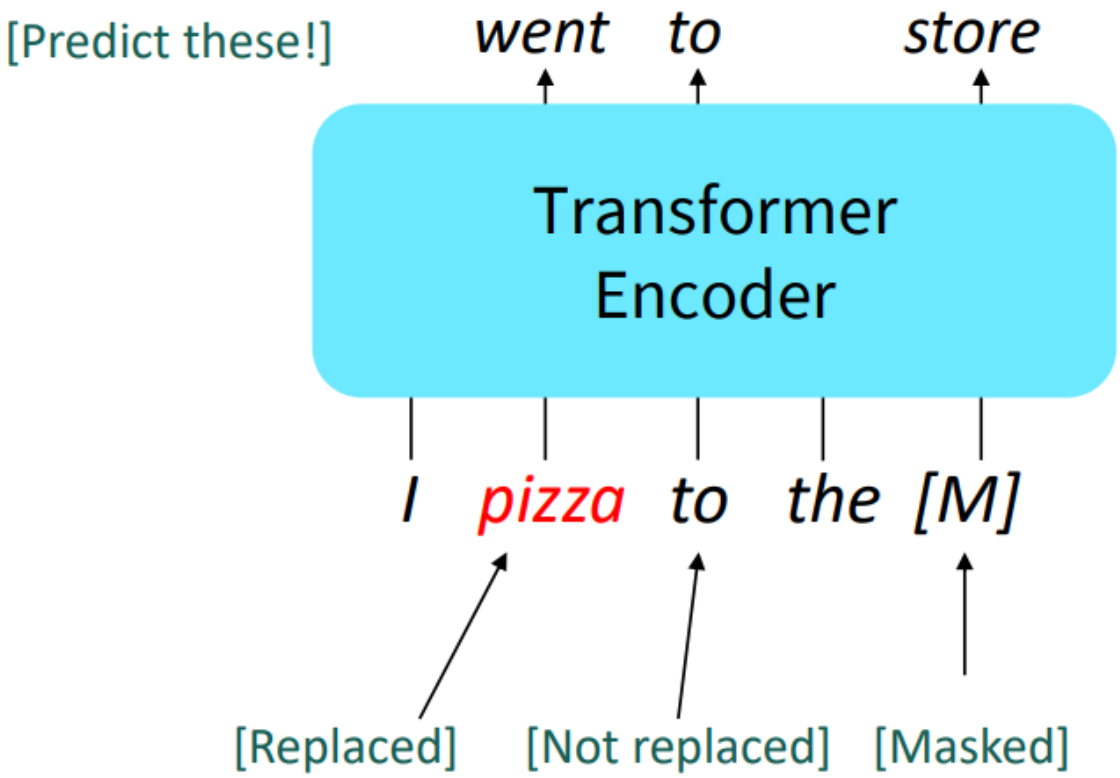
Too little masking: too expensive to train Too much masking: not enough context

Objective I: Masked Language Models (MLMs)

A little more complication:

- Rather than *always* replacing the chosen words with `[MASK]`, the data generator will do the following:
- 80% of the time: Replace the word with the `[MASK]` token, e.g., `my dog is hairy` → `my dog is [MASK]`
- 10% of the time: Replace the word with a random word, e.g., `my dog is hairy` → `my dog is apple`
- 10% of the time: Keep the word unchanged, e.g., `my dog is hairy` → `my dog is hairy`. The purpose of this is to bias the representation towards the actual observed word.

Because [MASK] will not be used when BERT is used in downstream tasks



[Devlin et al., 2018]

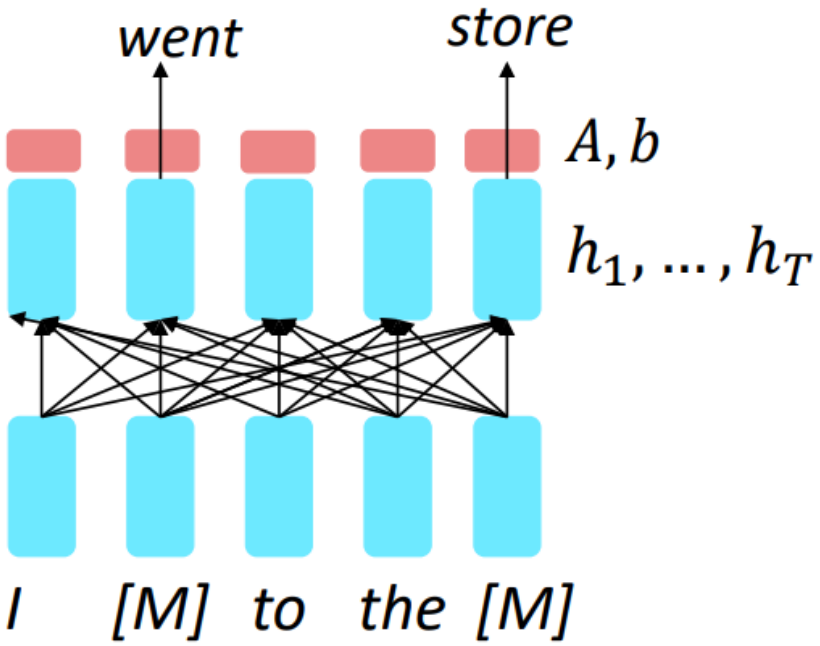
Pretraining Encoders: What Pretraining Objective to Use?

So far, we've looked at language model pretraining. But **encoders get bidirectional context**, so we can't do language modeling!

Idea: replace some fraction of words in the input with a special [MASK] token; predict these words.

$$h_1,\ldots,h_T = \text{Encoder}(w_1,\ldots,w_T)$$

$$y_i \sim Aw_i+b$$



Only add loss terms from words that are "masked out." If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(x|\tilde{x})$. Called **Masked LM.**

[Devlin et al., 2018]

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture9-pretraining.pdf

Objective II: Next Sentence Prediction (NSP)

Always sample two sentences, predict whether the second sentence is followed after the first one.

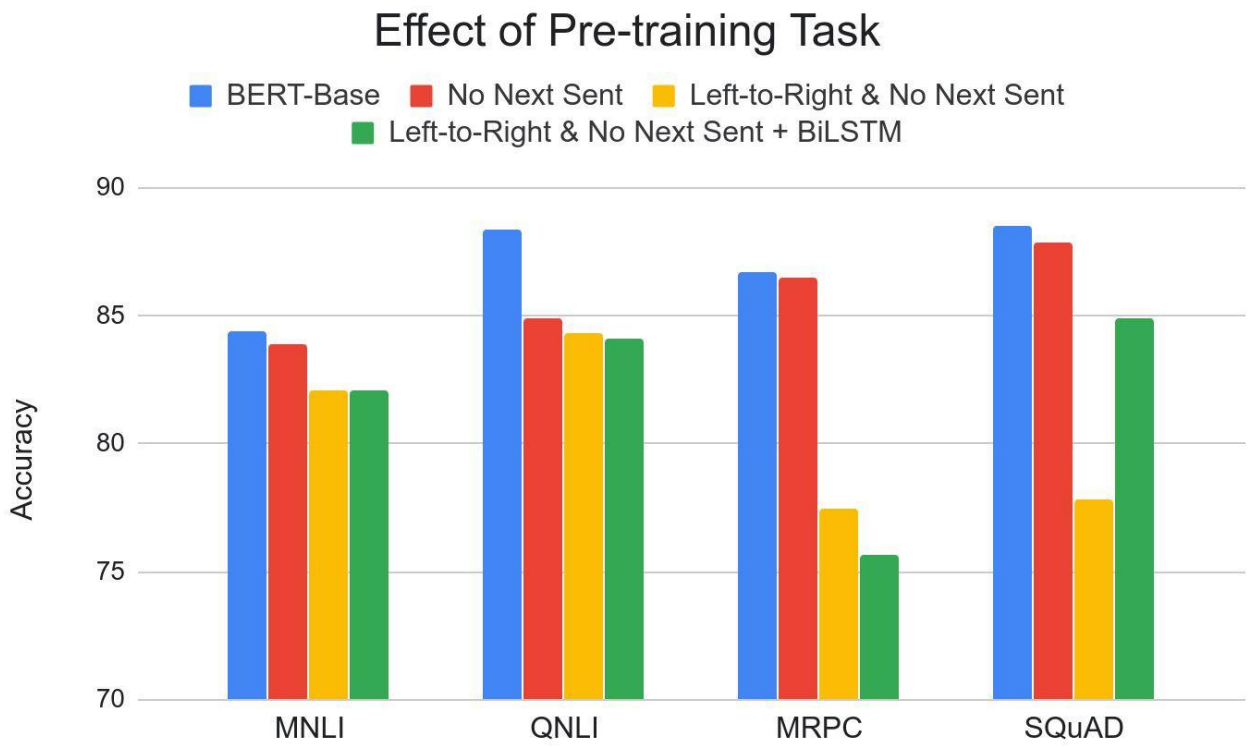
Input = `[CLS] the man went to [MASK] store [SEP]`
`he bought a gallon [MASK] milk [SEP]`
Label = `IsNext`

Input = `[CLS] the man [MASK] to the store [SEP]`
`penguin [MASK] are flight ##less birds [SEP]`
Label = `NotNext`

Recent papers show that NSP is not necessary…

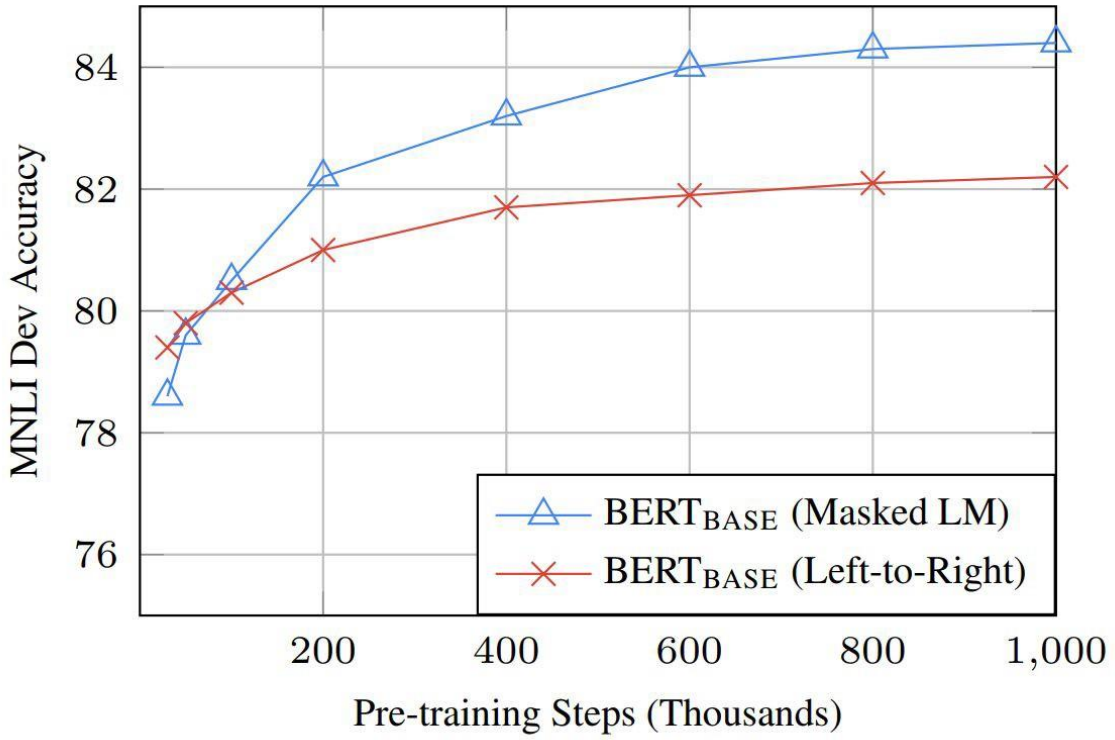
(Joshi*, Chen* et al, 2019) :SpanBERT: Improving Pre-training by Representing and Predicting Spans (Liu et al, 2019): RoBERTa: A Robustly Optimized BERT Pretraining Approach

On BERT Pre-training



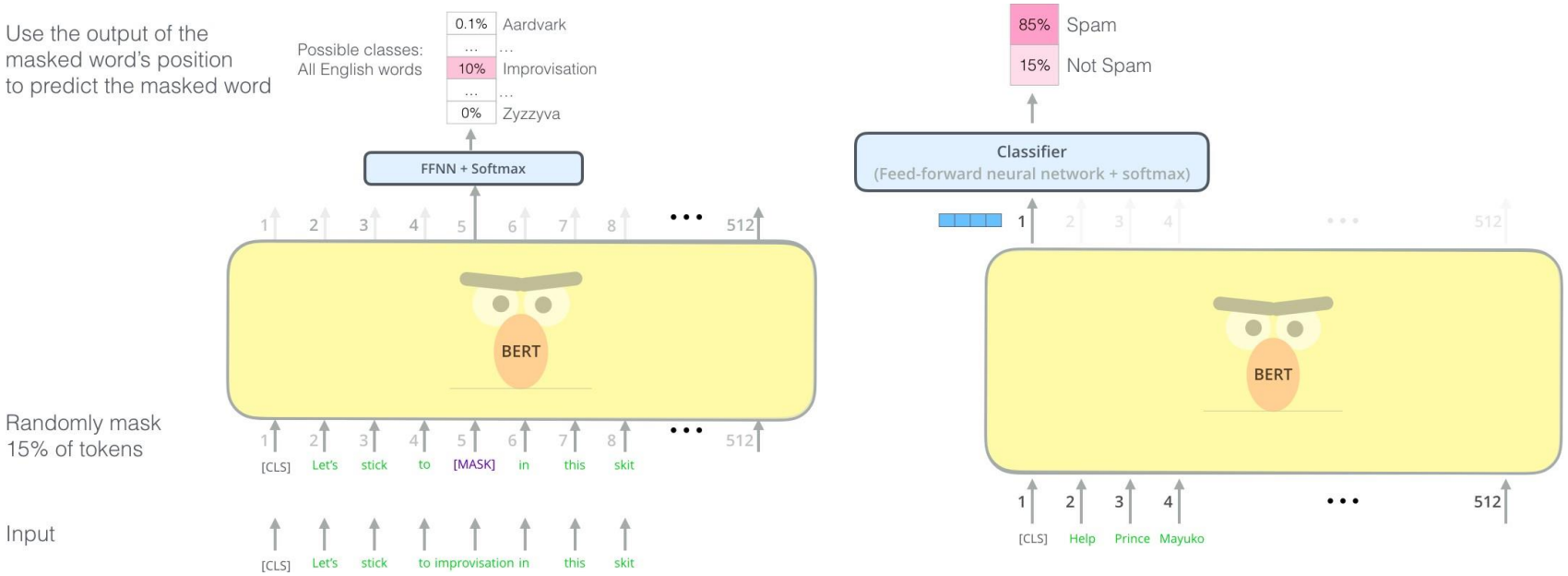
- Masked LM (compared to left-to-right LM) is very important on some tasks, Next Sentence Prediction is important on other tasks.
- Left-to-right model does very poorly on word-level task (SQuAD), although this is mitigated by BiLSTM

Directionality Helps



- Masked LM takes slightly longer to converge because it only predicts 15% instead of 100%
- But absolute results are much better almost immediately

Pre-training and Fine-tuning



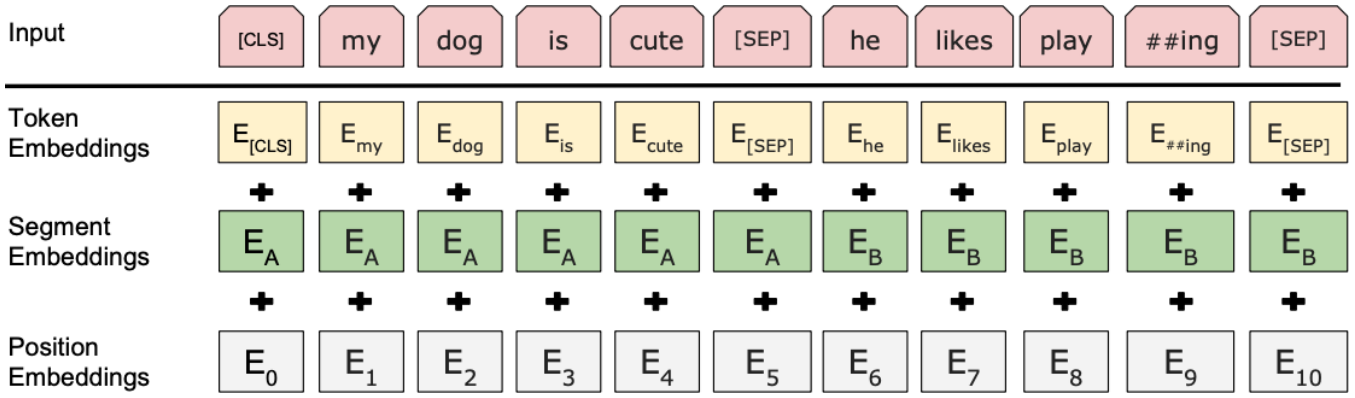
Pre-training

Fine-tuning

Key idea: all the weights are fine-tuned on downstream tasks

More Details

- Input representations



- Use word pieces instead of words: playing => play ##ing ← Assignment 4
- Trained 40 epochs on Wikipedia (2.5B tokens) + BookCorpus (0.8B tokens)
- Released two model sizes: BERT_base, BERT_large

BERT Step 1: Building a Training Example

Raw text: Sentence A *"the man went to the store"* Sentence B *"he bought milk"*

① Add special tokens and segments [CLS] starts the input, [SEP] ends each sentence; A/B tells the sentences apart

NSP label: IsNext

[CLS]	the	man	went	to	the	store	[SEP]	he	bought	milk	[SEP]
A0	A1	A2	A3	A4	A5	A6	A7	B8	B9	B10	B11

② Pick ~15% of the tokens to predict (3 shown here for illustration)

③ Corrupt the selected tokens 80% → [MASK], 10% → random word, 10% → unchanged

[CLS]	the	man	**went**	to	the	**[MASK]**	[SEP]	he	bought	**apple**	[SEP]
			10%: keep			**80%: [MASK]**				**10%: random**	

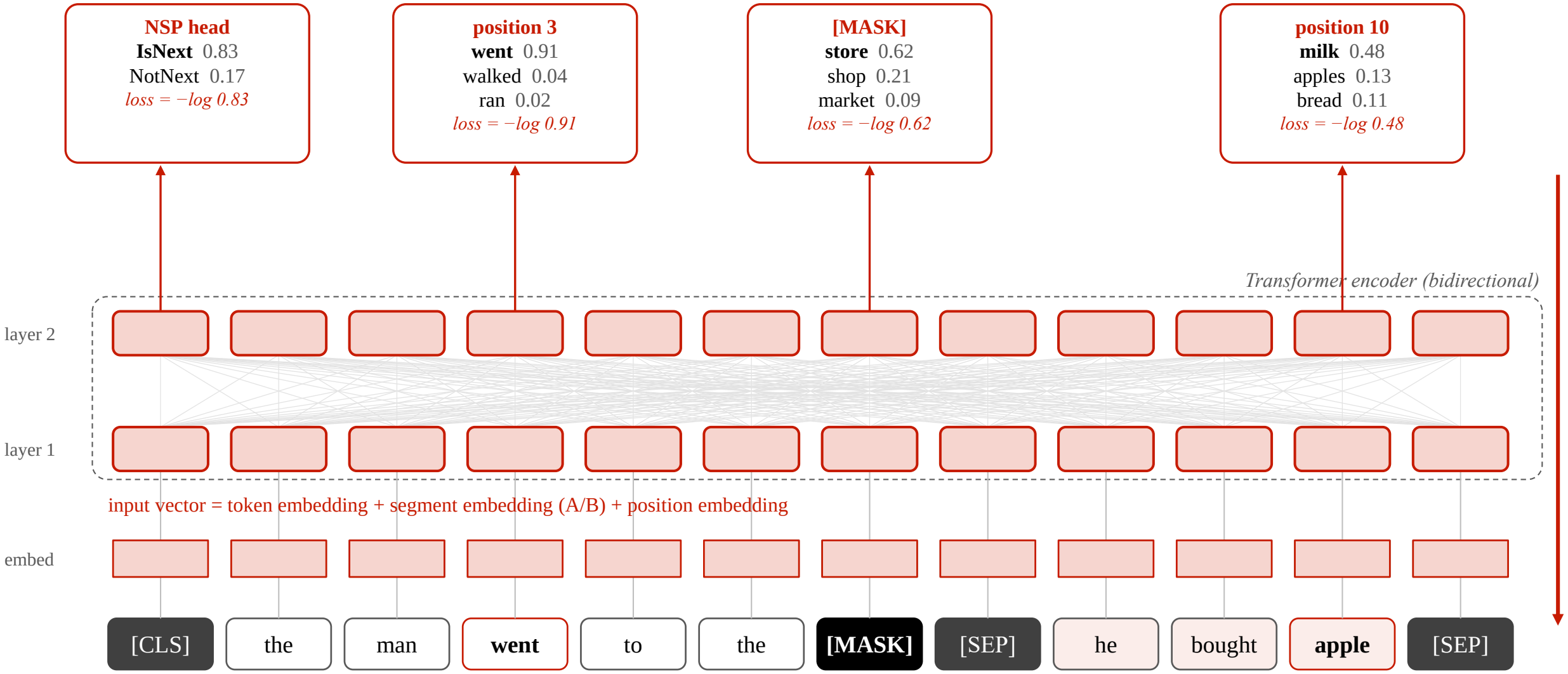
④ Targets: the model must recover the original tokens (and the NSP label)

IsNext	**went**	**store**	**milk**

Why not always [MASK]? [MASK] never appears when BERT is used later, so sometimes the model sees a random or the real word and must still predict the original.

50% of pairs use the real next sentence (IsNext), 50% a random sentence from the corpus (NotNext).

BERT Step 2: Pre-training (Masked LM + NSP)

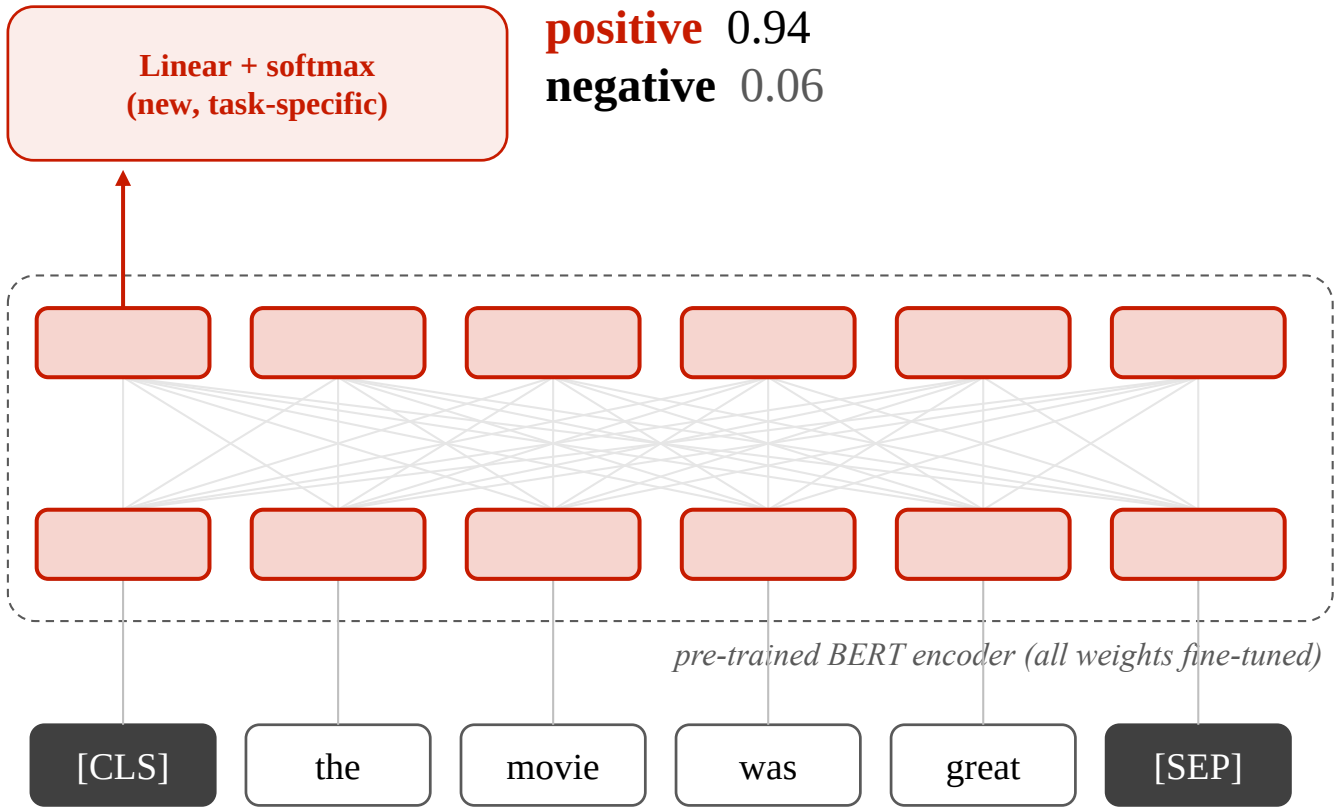


Total loss = MLM loss on the 3 selected positions + NSP loss → backpropagate and update all weights

BERT Step 3: Inference and Fine-tuning

Fine-tuned for sentiment classification

Only the [CLS] vector feeds the classifier; the whole sentence is read in **one forward pass**.



Without fine-tuning: fill in the blank

"Paris is the **[MASK]** of France."

capital 0.97 city 0.01 heart 0.01

The MLM head from pre-training predicts the masked word using both sides of the context.

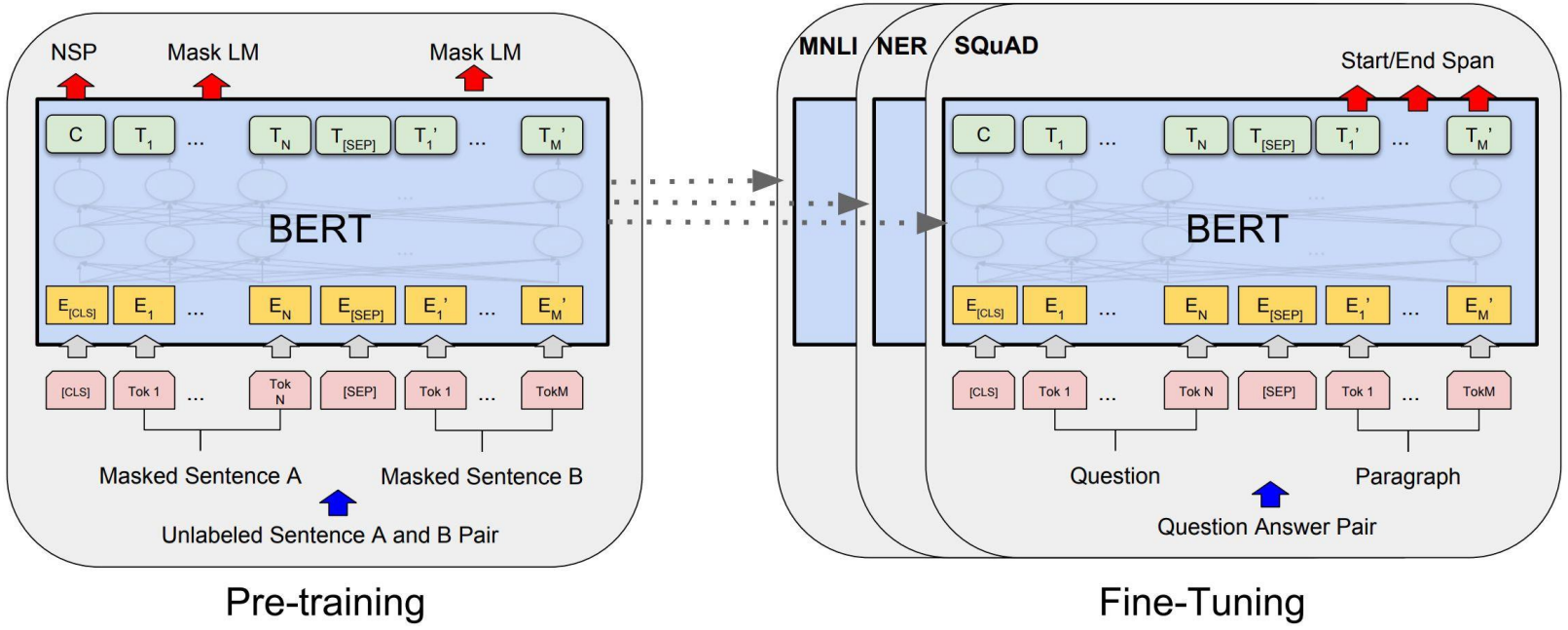
BERT vs. GPT at inference

BERT: one pass, every token sees every other token → great for understanding (classification, QA, embeddings).

GPT: a loop, one new token per pass, each token sees only the left → great for generation.

Numbers on this slide are illustrative.

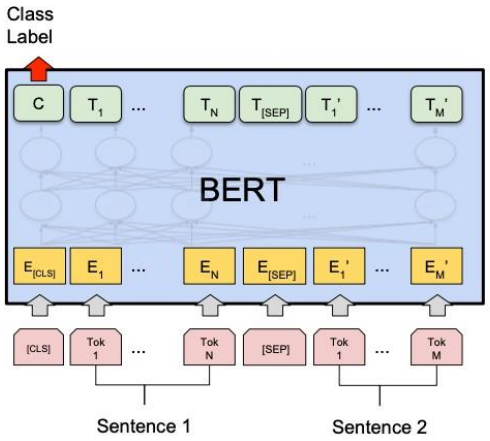
One for All



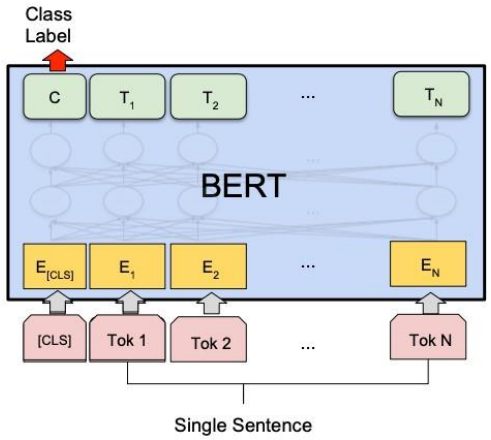
Maybe this is one of the first popular **Foundation model**

On the Opportunities and Risks of Foundation Models. https://arxiv.org/abs/2108.07258

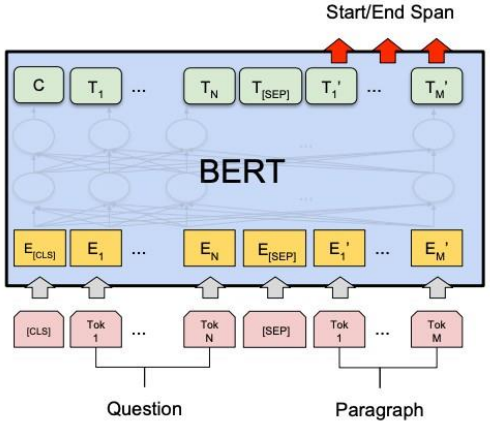
Applications



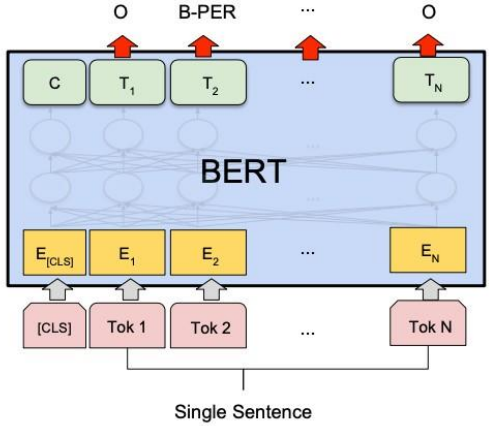
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(b) Single Sentence Classification Tasks: SST-2, CoLA

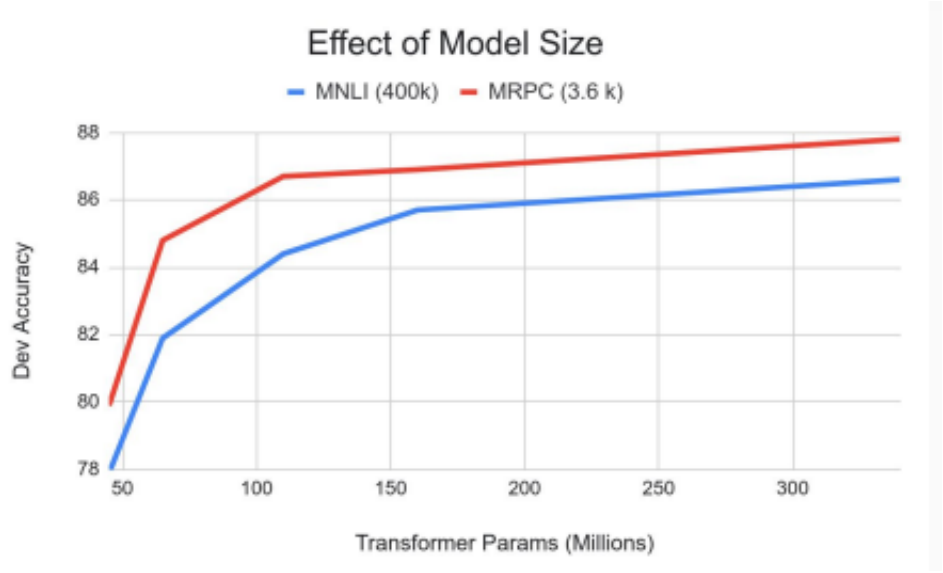


(c) Question Answering Tasks: SQuAD v1.1



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

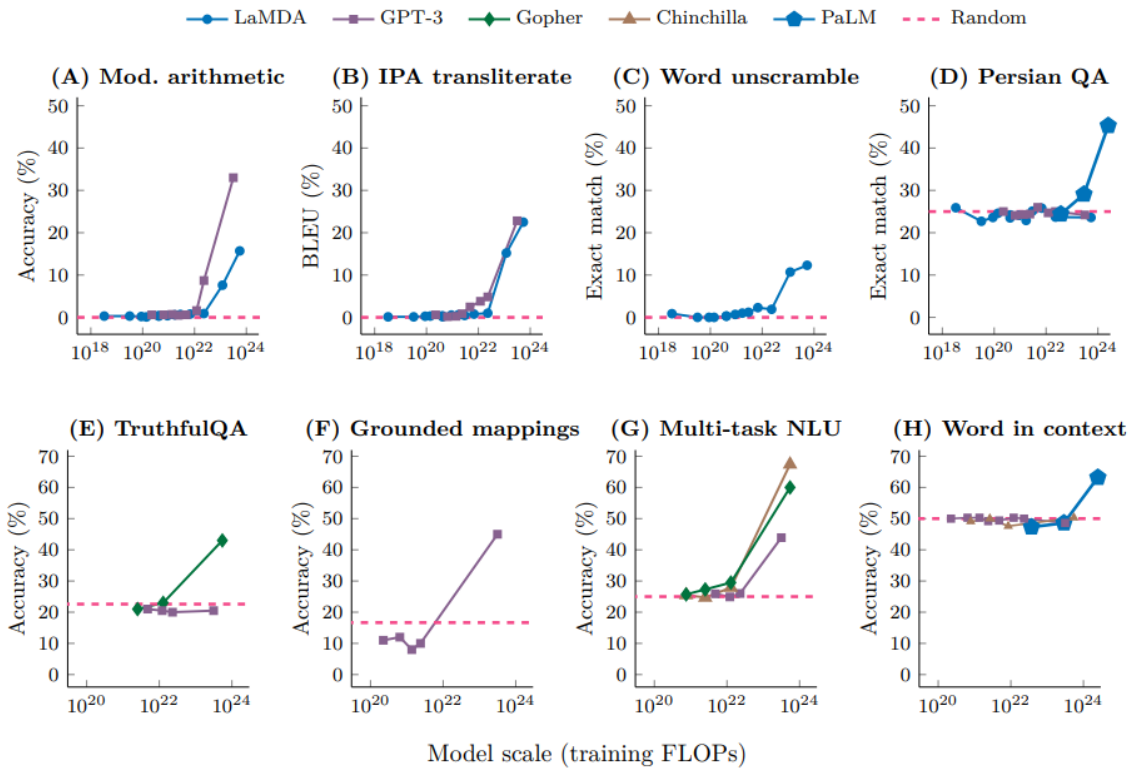
Scalability: BERT



It seems BERT cannot benefit that much from scaling;

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1204/slides/Jacob_Devlin_BERT.pdf

Scalability: GPT



GPT scales better !
Reason: generation scales, but not for discrimination

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1204/slides/Jacob_Devlin_BERT.pdf

Variants of Contextualized Word Vectors

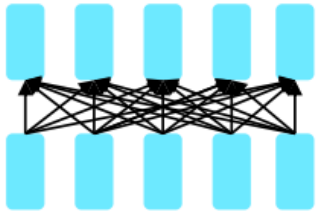
Overview

Model	Type	Architecture	Task
NLM [25]	static	1-layer MLP	$(a, b) \rightarrow c$ predicting the next word
Skip-Gram [200]	static	1-layer MLP	$b \rightarrow c, \quad b \rightarrow a$ predicting neighboring words
CBow [200]	static	1-layer MLP	$(a, c) \rightarrow b$ predicting central words
Glove [227]	static	1-layer MLP	$\vec{w_i}^T \vec{w_j} \propto logp(\#(w_i\, w_j))$ predicting the log co-occurrence count
ELMO [230]	contextualized	LSTM	$(a, b, c, d) \rightarrow e, \quad (e, d, c, b) \rightarrow a$ bi-directional language model
BERT [66], Roberta [185] ALBERT [154],XLNET [351]	contextualized	Transformers or Transformer-XL	$(a, [\texttt{mask}], c) \rightarrow (_, b, _)$ predicting masked words
Electra [54]	contextualized	Transformer	$(a, \hat{b}, c, \hat{d}) \rightarrow (0, 1, 0, 1)$ replaced token prediction
T5 [241] BART [158]	contextualized	Transformers	$(a, b, c,) \rightarrow (d, e)$ predicting the sequence
GPT [240]	contextualized	Transformers	$(a, b, c, d) \rightarrow e$ autoregressively predicting the next word

Benyou Wang et al. Pre-trained Language Models in Biomedical Domain: A Systematic Survey. ACM Computing Survey.

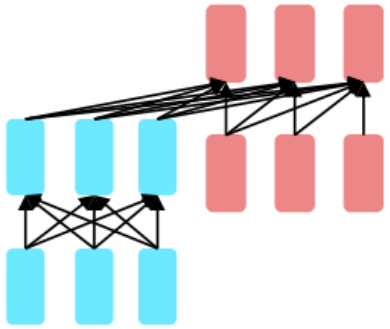
Pretraining for Three Types of Architectures

The neural architecture influences the type of pretraining, and natural use cases.



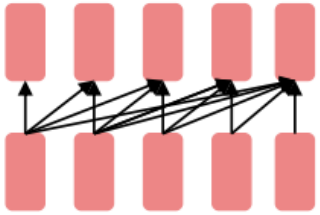
Encoders

- **Gets bidirectional context – can condition on future!**
- **How do we train them to build strong representations?**



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?



Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture9-pretraining.pdf

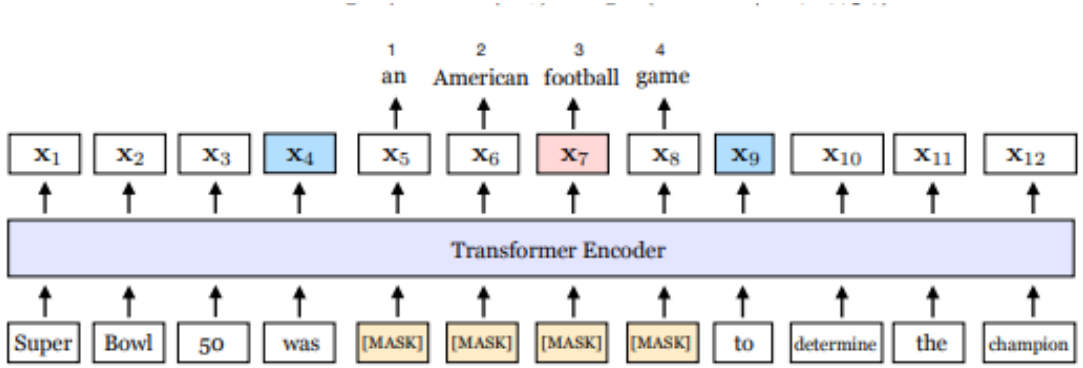
RoBERTa

- *RoBERTa: A Robustly Optimized BERT Pretraining Approach* (Liu et al, University of Washington and Facebook, 2019)
- Trained BERT for more epochs and/or on more data
  - Showed that more epochs alone helps, even on same data
  - More data also helps
- Improved masking and pre-training data slightly

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
Single-task single models on dev										
$\text{BERT}_{\text{LARGE}}$	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
$\text{XLNet}_{\text{LARGE}}$	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	**90.2/90.2**	**94.7**	**92.2**	**86.6**	**96.4**	**90.9**	**68.0**	**92.4**	**91.3**	-

SpanBERT

- *SpanBERT: Improving Pre-training by Representing and Predicting Spans* (Joshi et al., 2019)
- Mask a whole span
- Span masking helps



	SQuAD 1.1 EM	SQuAD 1.1 F1	SQuAD 2.0 EM	SQuAD 2.0 F1
Human Perf.	82.3	91.2	86.8	89.4
Google BERT	84.3	91.3	80.0	83.3
Our BERT	86.5	92.6	82.8	85.9
Our BERT-1seq	87.5	93.3	83.8	86.6
SpanBERT	**88.8**	**94.6**	**85.7**	**88.7**

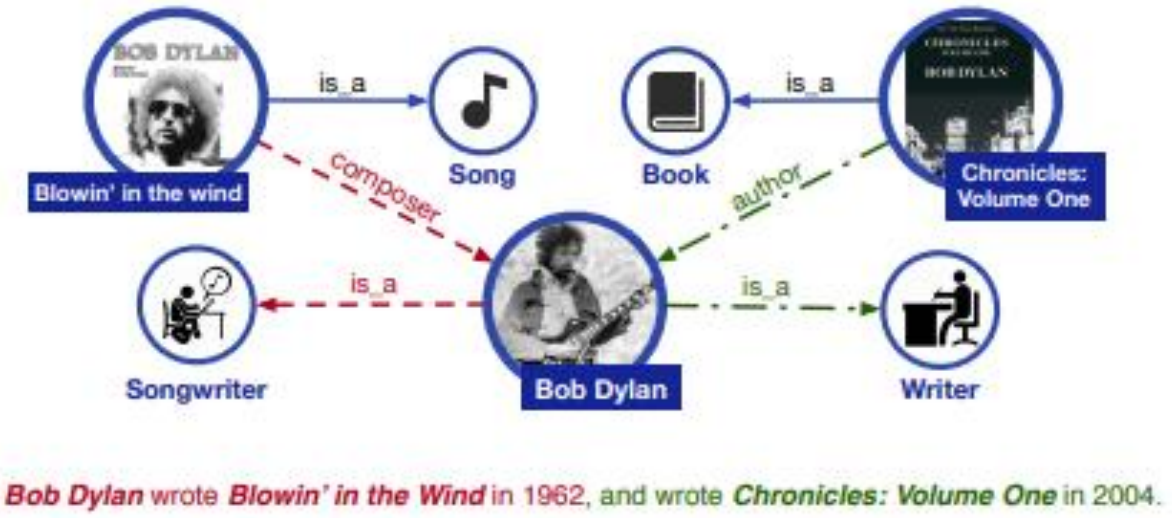
BERT-wwm

- Pre-Training with Whole Word Masking for Chinese, Cui et al. 2019
- Mask a whole Chinese word

	Chinese	English
Original Sentence	使用语言模型来预测下一个词的概率。	we use a language model to predict the probability of the next word.
+ CWS	语言 模型 来 预测 下 一个 词 的 概率 。	-
+ BERT Tokenizer	语 言 模 型 来 预 测 下 一 个 词 的 概 率 。	we use a language **model** to **pre ##di ##ct** the **pro ##ba ##bility** of the next word .
Original Masking	语 言 **[M]** 型 来 **[M]** 测 下 一 个 词 的 概 率 。	we use a language **[M]** to **[M] ##di ##ct** the **pro [M] ##bility** of the next word .
+ WWM	语 言 **[M] [M]** 来 **[M] [M]** 下 一 个 词 的 概 率 。	we use a language **[M]** to **[M] [M] [M]** the **[M] [M] [M]** of the next word .
++ N-gram Masking	**[M] [M] [M] [M]** 来 **[M] [M]** 下 一 个 词 的 概 率 。	we use a **[M] [M]** to **[M] [M] [M]** the **[M] [M] [M] [M] [M]** next word .
+++ Mac Masking	语 法 建 模 来 预 见 下 一 个 词 的 几 率 。	we use a **text system** to **ca ##lc ##ulate** the **po ##si ##bility** of the next word .

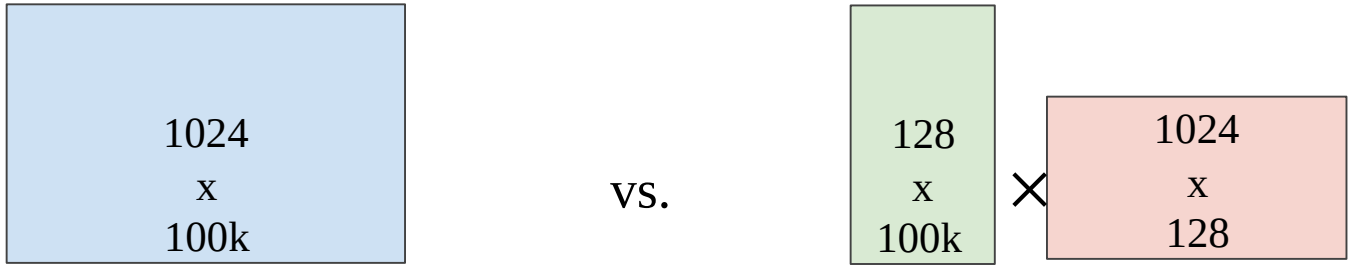
ERNIE

- ERNIE: Enhanced Language Representation with Informative Entities. Zhang et al. 2019
- Mask Informative Entities



ALBERT

- *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations* (Lan et al, Google and TTI Chicago, 2019)
- Innovation #1: Factorized embedding parameterization
  - Use a small embedding size (e.g., 128) and then project it to Transformer hidden size (e.g., 1024) with parameter matrix



ALBERT

- Innovation #2: Cross-layer parameter sharing
  - Share all parameters between Transformer layers
- Results:

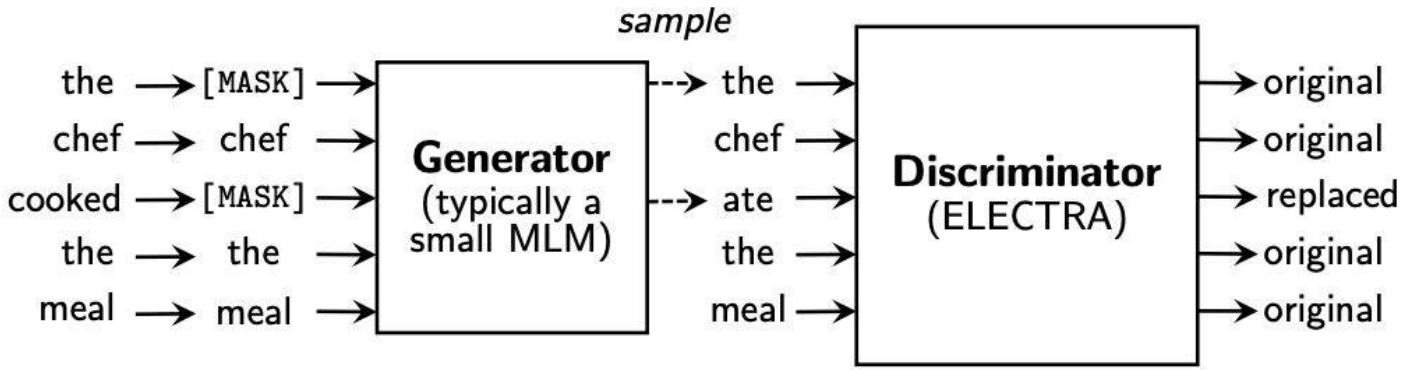
Models	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS
Single-task single models on dev								
BERT-large	86.6	92.3	91.3	70.4	93.2	88.0	60.6	90.0
XLNet-large	89.8	93.9	91.8	83.8	95.6	89.2	63.6	91.8
RoBERTa-large	90.2	94.7	**92.2**	86.6	96.4	**90.9**	68.0	92.4
ALBERT (1M)	90.4	95.2	92.0	88.1	96.8	90.2	68.7	92.7
ALBERT (1.5M)	**90.8**	**95.3**	**92.2**	**89.2**	**96.9**	**90.9**	**71.4**	**93.0**

- ALBERT is light in terms of *parameters*, not *speed*

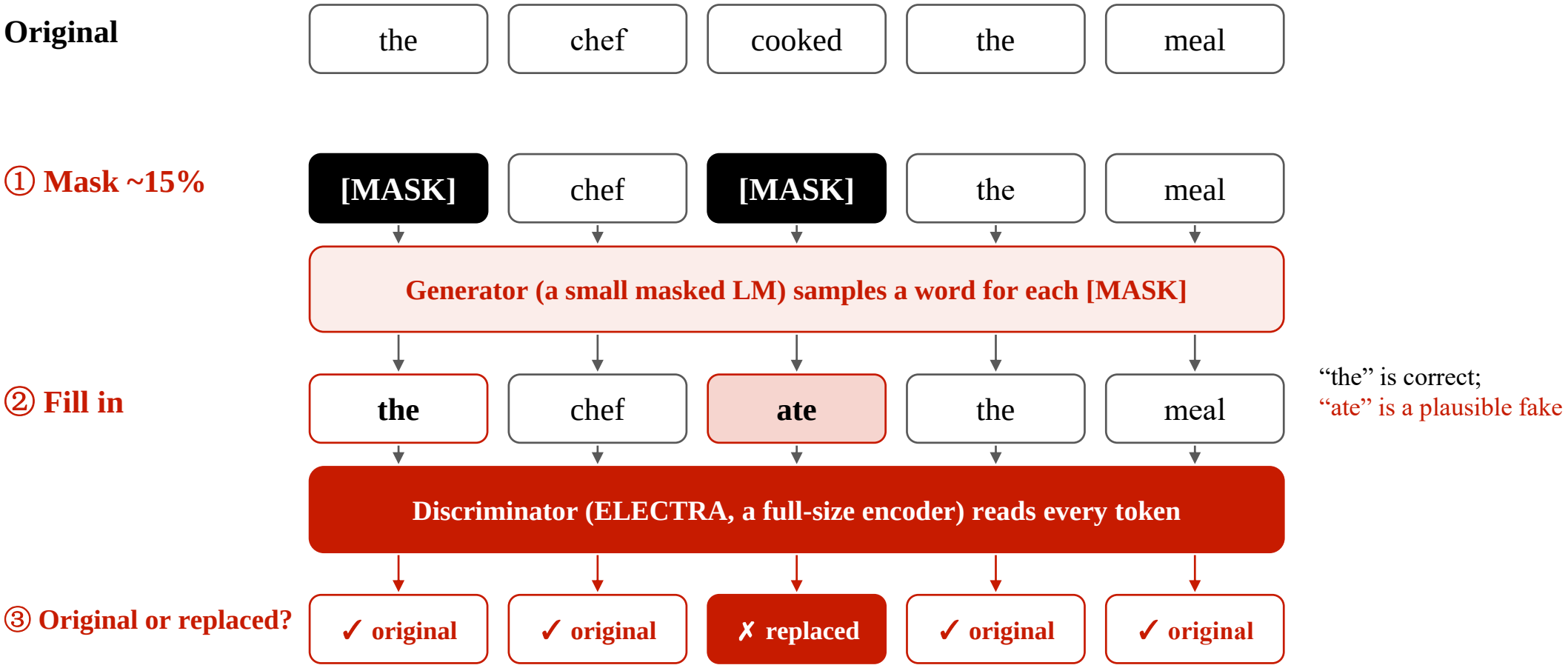
Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT	base	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3	4.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	1.0
ALBERT	base	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1	5.6x
	large	18M	90.6/83.9	82.3/79.4	83.5	91.7	68.5	82.4	1.7x
	xlarge	60M	92.5/86.1	86.1/83.1	86.4	92.4	74.8	85.5	0.6x
	xxlarge	235M	**94.1/88.3**	**88.1/85.1**	**88.0**	**95.2**	**82.3**	**88.7**	0.3x

ELECTRA

- *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators* (Clark et al, 2020)
- Train model to discriminate locally plausible text from real text



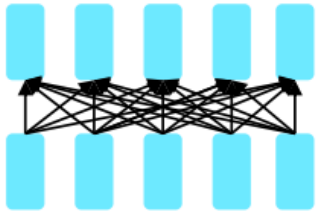
ELECTRA: Replaced Token Detection, Step by Step



Why it works: the loss covers all tokens (100%), not just the 15% that were masked, so every token gives a training signal. After pre-training, the generator is thrown away and the discriminator is fine-tuned.

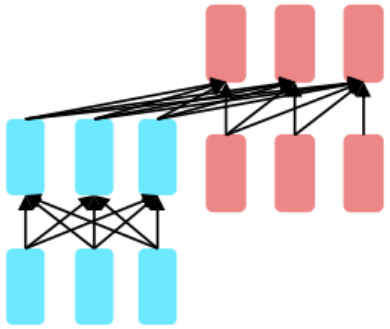
Pretraining for Three Types of Architectures

The neural architecture influences the type of pretraining, and natural use cases.



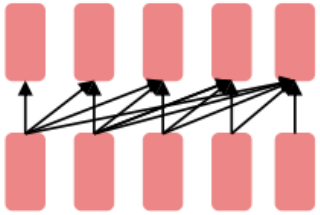
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

- **Good parts of decoders and encoders?**
- **What's the best way to pretrain them?**



Decoders

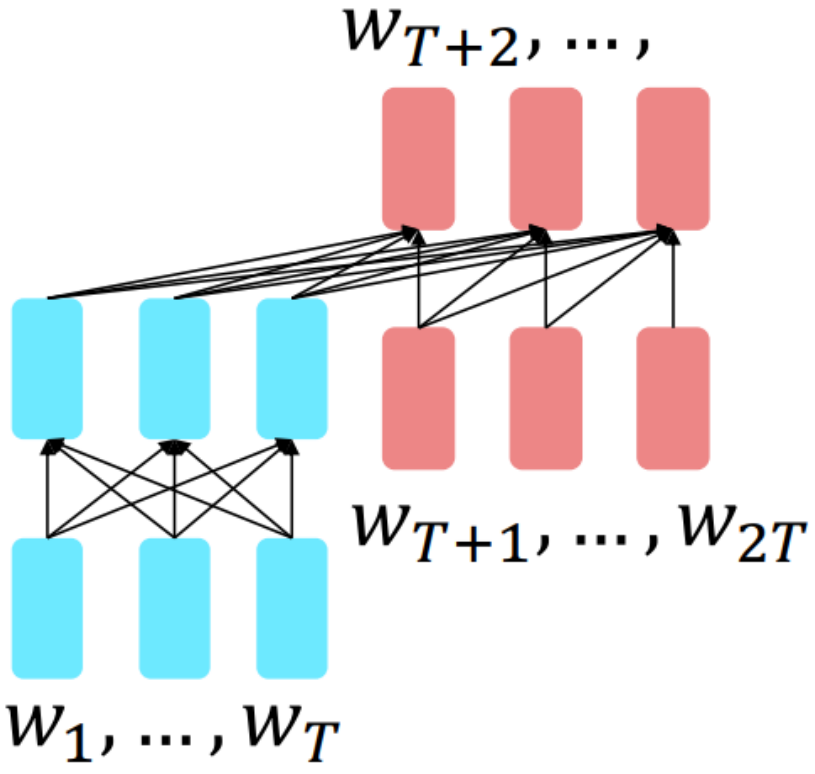
- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture9-pretraining.pdf

Pretraining Encoder-decoders: What Pretraining Objective to Use?

For **encoder-decoders**, we could do something like **language modeling**, but where a prefix of every input is provided to the encoder and is not predicted.

$$
\begin{aligned}
h_1,\ldots,h_T &= \text{Encoder}(w_1,\ldots,w_T) \\
h_{T+1},\ldots,h_{2T} &= \text{Decoder}(w_1,\ldots,w_T,h_1,\ldots,h_T) \\
y_i &\sim Ah_i+b, \quad i>T
\end{aligned}
$$



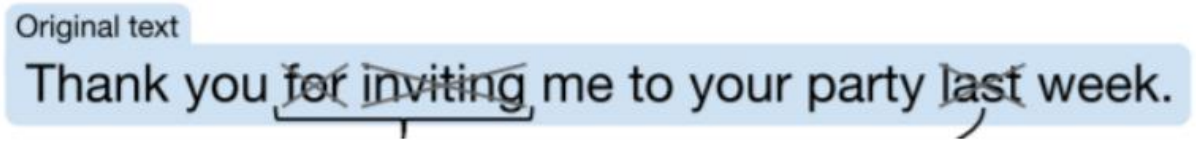
The **encoder** portion benefits from bidirectional context;
The **decoder** portion is used to train the whole model through language modeling.

[Raffel et al., 2020]

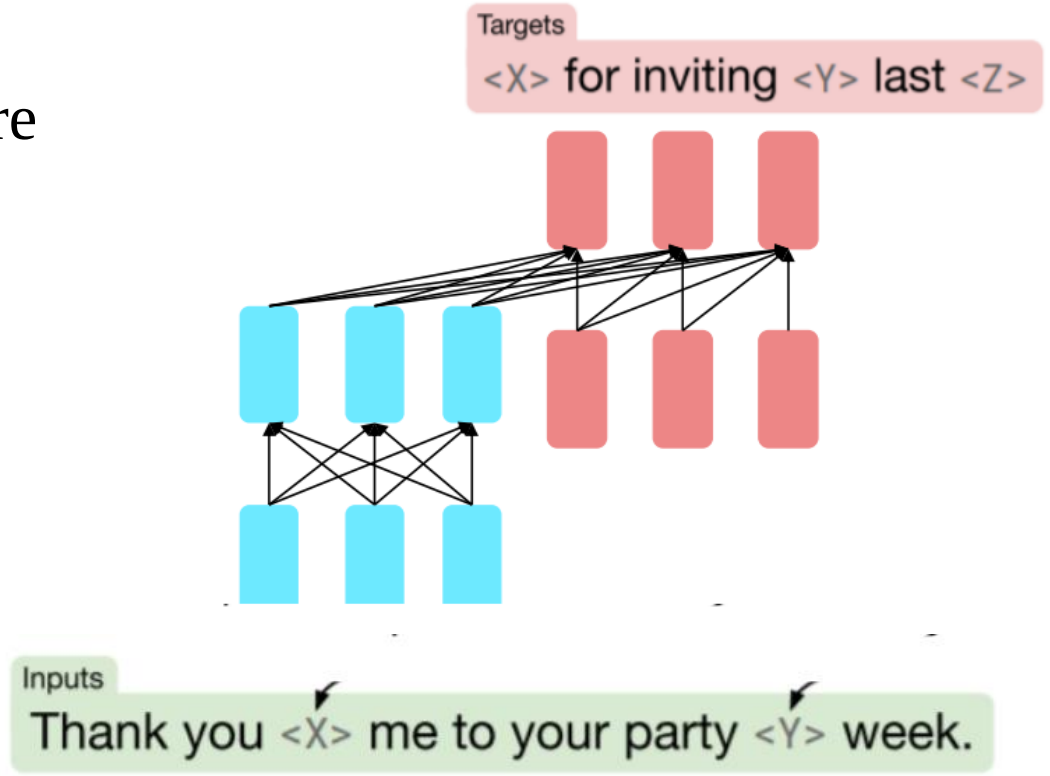
Pretraining Encoder-decoders: What Pretraining Objective to Use?

What Raffel et al., 2020 found to work best was span corruption. Their model: T5.

Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!



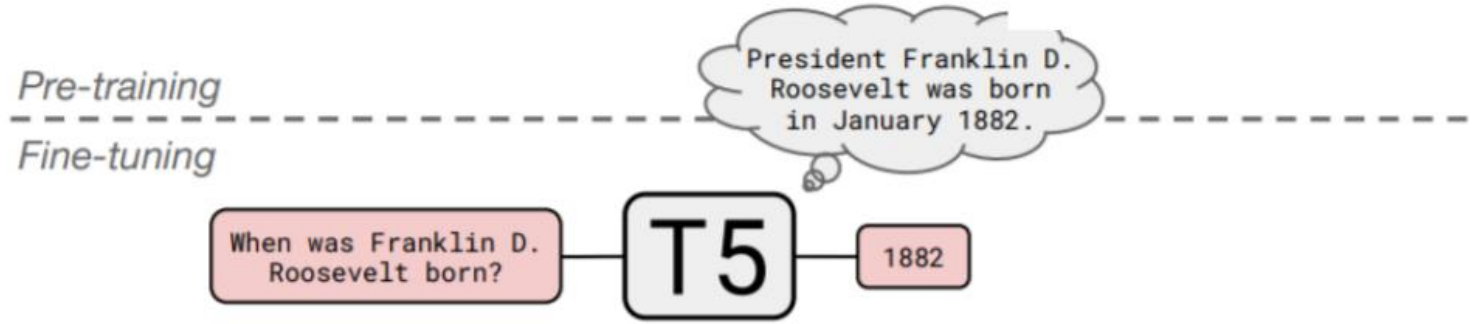
This is implemented in text preprocessing: it's still an objective that looks like **language modeling** at the decoder side.



[Raffel et al., 2020]

Pretraining Encoder-decoders: What Pretraining Objective to Use?

A fascinating property of T5: it can be finetuned to answer a wide range of questions, retrieving knowledge from its parameters.



	NQ	WQ	TQA dev	TQA test	
Karpukhin et al. (2020)	**41.5**	42.4	**57.9**	–	
T5.1.1-Base	25.7	28.2	24.2	30.6	**220 million params**
T5.1.1-Large	27.3	29.5	28.5	37.2	**770 million params**
T5.1.1-XL	29.5	32.4	36.0	45.1	**3 billion params**
T5.1.1-XXL	32.8	35.6	42.9	52.5	**11 billion params**
T5.1.1-XXL + SSM	35.2	**42.8**	51.9	**61.6**	

NQ: Natural Questions
WQ: WebQuestions
TQA: Trivia QA

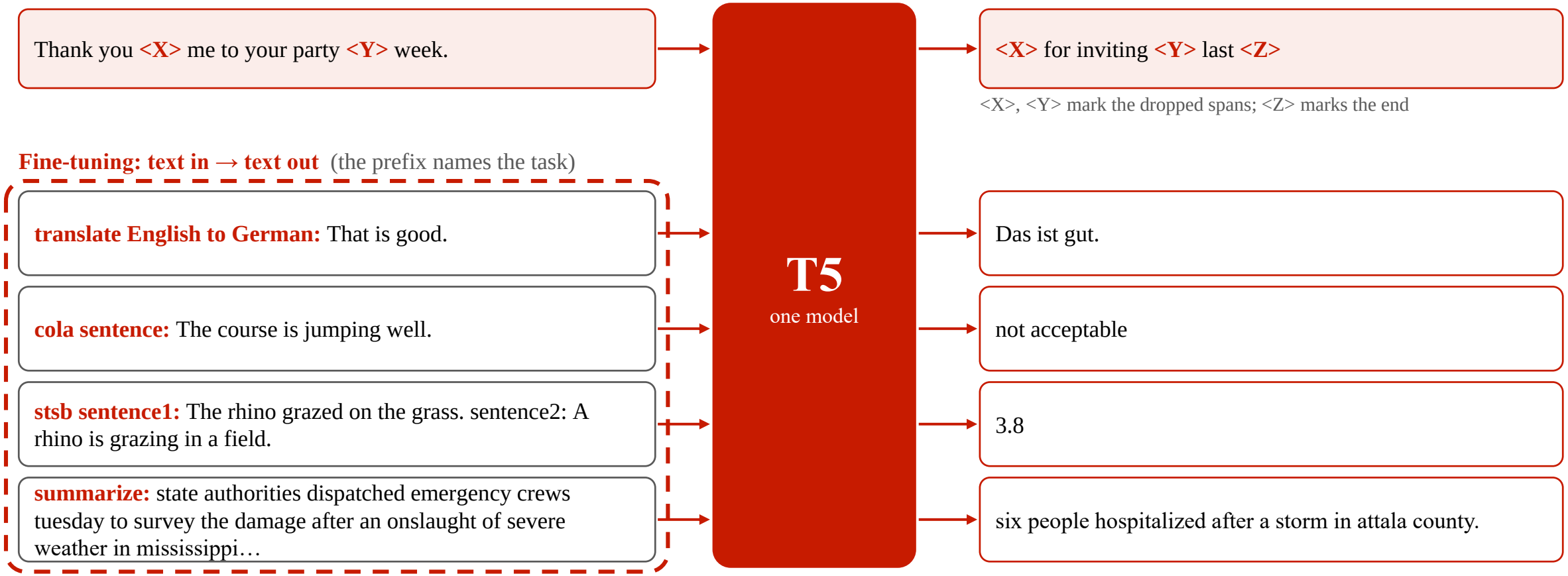
All “open-domain” versions

[Raffel et al., 2020]

T5: Many Tasks, One Text-to-Text Model

Pre-training: span corruption (no task prefix)

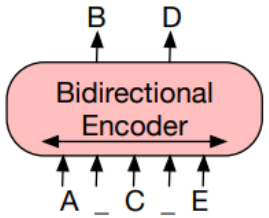
Original: Thank you **for inviting** me to your party **last** week.



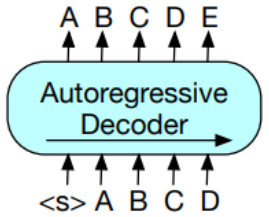
All tasks are mixed into one training set: same model, same loss (predict the output text), same decoding.

Pretraining Encoder-decoders: What Pretraining Objective to Use?

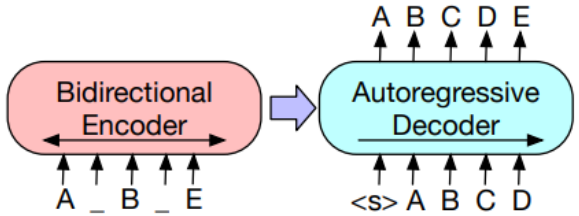
BART: **Denoising** Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. https://aclanthology.org/2020.acl-main.703.pdf



(a) BERT: Random tokens are replaced with masks, and the document is encoded bidirectionally. Missing tokens are predicted independently, so BERT cannot easily be used for generation.



(b) GPT: Tokens are predicted auto-regressively, meaning GPT can be used for generation. However words can only condition on leftward context, so it cannot learn bidirectional interactions.



(c) BART: Inputs to the encoder need not be aligned with decoder outputs, allowing arbitary noise transformations. Here, a document has been corrupted by replacing spans of text with a mask symbols. The corrupted document (left) is encoded with a bidirectional model, and then the likelihood of the original document (right) is calculated with an autoregressive decoder. For fine-tuning, an uncorrupted document is input to both the encoder and decoder, and we use representations from the final hidden state of the decoder.

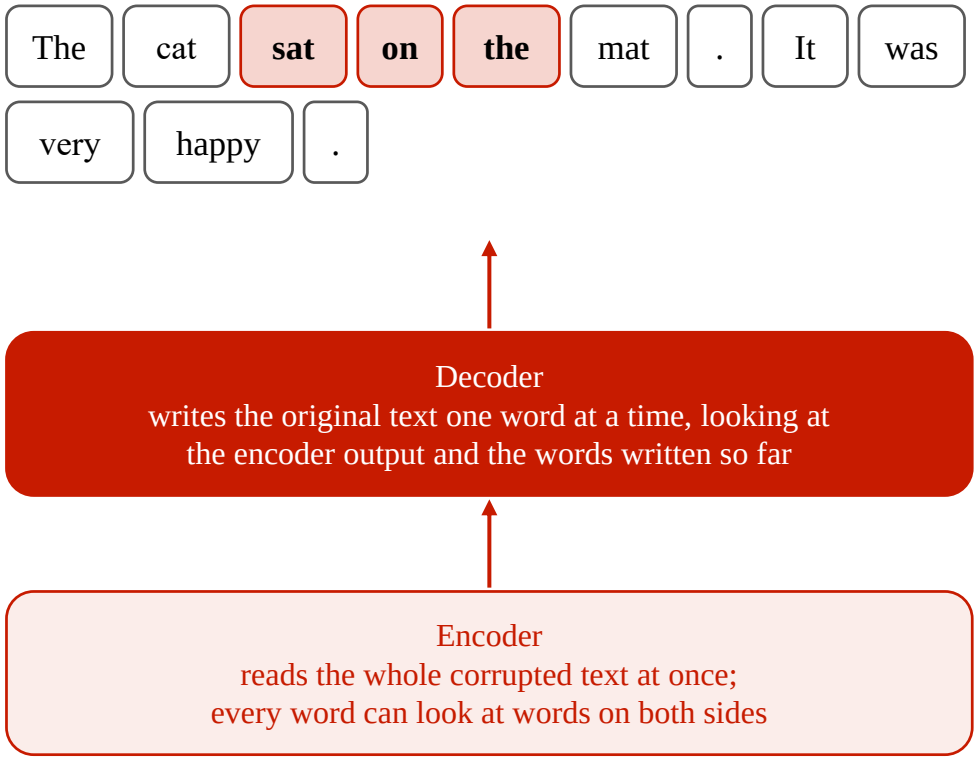
Figure 1: A schematic comparison of BART with BERT (Devlin et al., 2019) and GPT (Radford et al., 2018).

[Lewis et al., 2020]

BART: Corrupt the Text, Then Reconstruct It

Original	**The cat sat on the mat. It was very happy.**
Token masking	The cat **[MASK]** on the mat. It was **[MASK]** happy.
Token deletion	The cat on the mat. It was happy.
Text infilling	The cat **[MASK]** mat. It **[MASK]** was very happy.
Sentence permutation	It was very happy. The cat sat on the mat.
Document rotation	on the mat. It was very happy. The cat sat

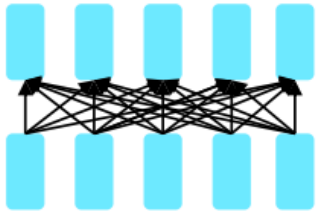
Text infilling: one [MASK] can hide several words (“sat on the”) or none at all, so the model must also guess how many words are missing.



Training signal: compare every word the decoder writes with the original text (cross-entropy). Here it had to restore “sat on the” for the first [MASK] and nothing for the second.

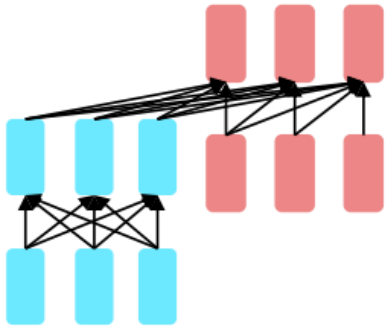
Pretraining for Three Types of Architectures

The neural architecture influences the type of pretraining, and natural use cases.



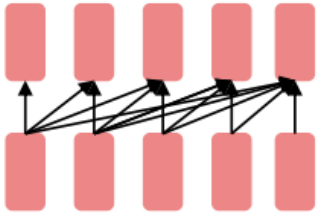
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?



Decoders

- **Language models! What we've seen so far.**
- **Nice to generate from; can't condition on future words**

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture9-pretraining.pdf

Encoder vs. Decoder vs. Encoder-Decoder: Trade-offs

	Encoder-only (BERT)	Decoder-only (GPT, LLaMA)	Encoder-decoder (T5, BART)
Attention	Bidirectional: every token sees the whole input	Causal: each token sees only earlier tokens	Bidirectional encoder + causal decoder (cross-attention)
Pre-training	Masked LM: learns from ~15% of tokens	Next-token prediction: learns from every token	Span corruption / denoising
Best at	Understanding: classification, retrieval, embeddings	Open-ended generation, in-context learning, chat	Input → output tasks: translation, summarization
KV cache	✗ No incremental cache: if the input changes, re-encode all of it	✓ Cache K, V of all past tokens; prompt and output share one cache; reusable across turns (prefix caching)	△ Decoder can cache; encoder runs once per input, but a new turn changes the bidirectional encoding → re-encode
Main cost	Cannot generate text natively	KV cache grows with context length	Two stacks; parameters split between input and output

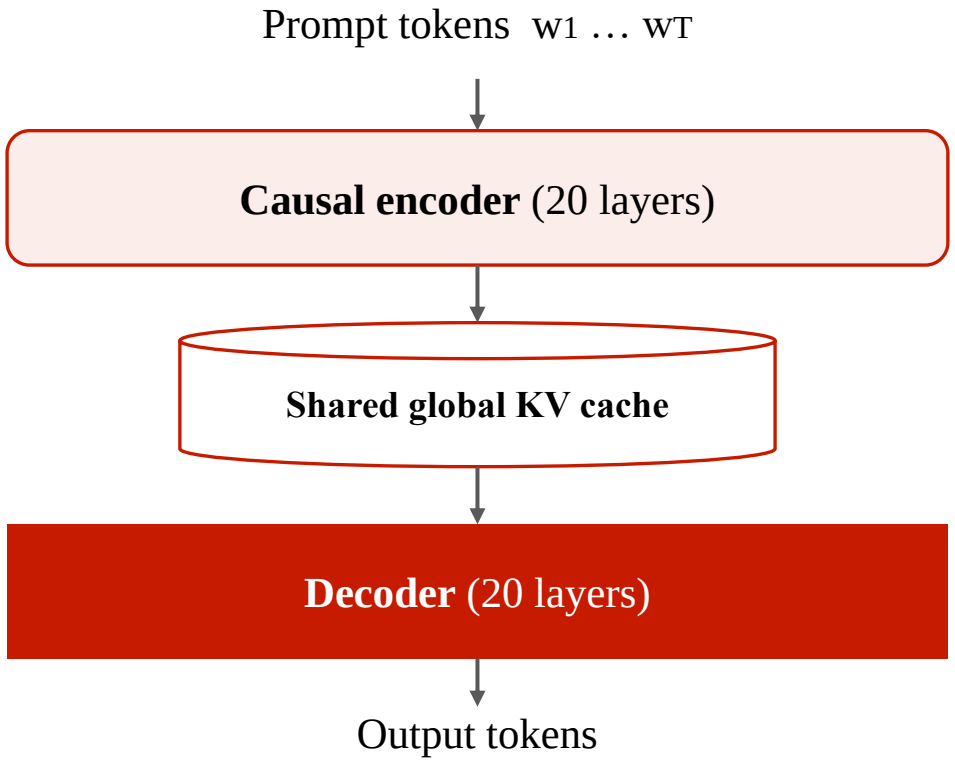
Why decoder-only models dominate today

- **One simple objective on raw text** — every token is a training signal, so data and compute scale easily.
- **One interface for every task** — text in, text out: prompting, in-context learning and chat, no task-specific heads.
- **Cheap generation with a KV cache** — past tokens never change under causal attention, so their keys/values are computed once.
- **Predictable scaling, shared tooling** — scaling laws, FlashAttention, paged KV cache, speculative decoding all target this design.

Is Encoder-Decoder Back? DeepSeek-V4.1-Flash

What DeepSeek released (Sept 10, 2026)

- 552B-parameter MoE model, 1M-token context
- New **"Causal Encoder–Decoder"**: 40 layers = 20-layer causal encoder + 20-layer decoder
- Reading input (prefill) uses only the encoder: **8B** active parameters; writing output uses the full stack: **16B**
- The decoder reads one shared KV cache projected from the encoder output → KV cache needs **1/4 the GPU memory** of V4-Flash



Verdict: the name is back, the T5 recipe is not

- Unlike T5, the encoder is **causal**: earlier tokens never depend on later ones, so cached states stay valid as the prompt grows.
- It is still a left-to-right language model trained on next-token prediction; conceptually close to YOCO's "decoder-decoder" design (Sun et al., 2024).
- What returns is the encoder-decoder idea of **splitting work between reading and writing**, used here to shrink the KV cache and cheapen long prompts.

Sources: DeepSeek V4.1-Flash release note (Sept 10, 2026); NVIDIA model card. Some internal details come from third-party analyses.

Back to the Language Model (Next-word Prediction)

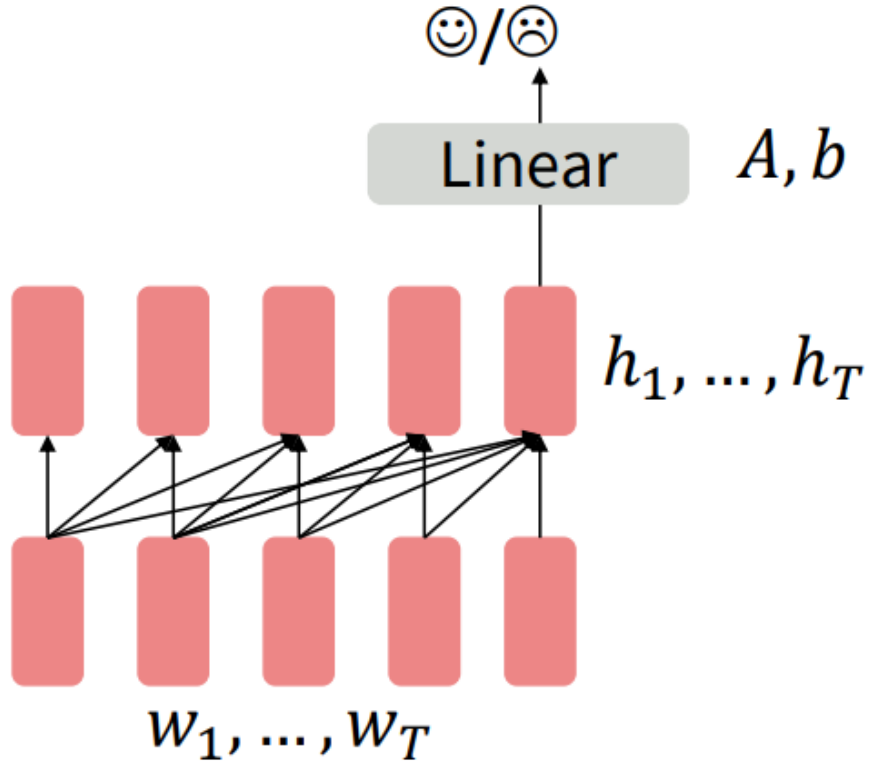
Pretraining Decoders

When using language model pretrained decoders, we can ignore that they were trained to model $p(w_t \mid w_{1:t-1})$

We can finetune them by training a classifier on the last word's hidden state.

$$h_1, \ldots, h_T = \text{Decoder}(w_1, \ldots, w_T)$$
$$y \sim A h_T + b$$

Where A and b are randomly initialized and specified by the downstream task.
Gradients backpropagate through the whole network.



[Note how the linear layer hasn't been pretrained and must be learned from scratch.]

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture9-pretraining.pdf

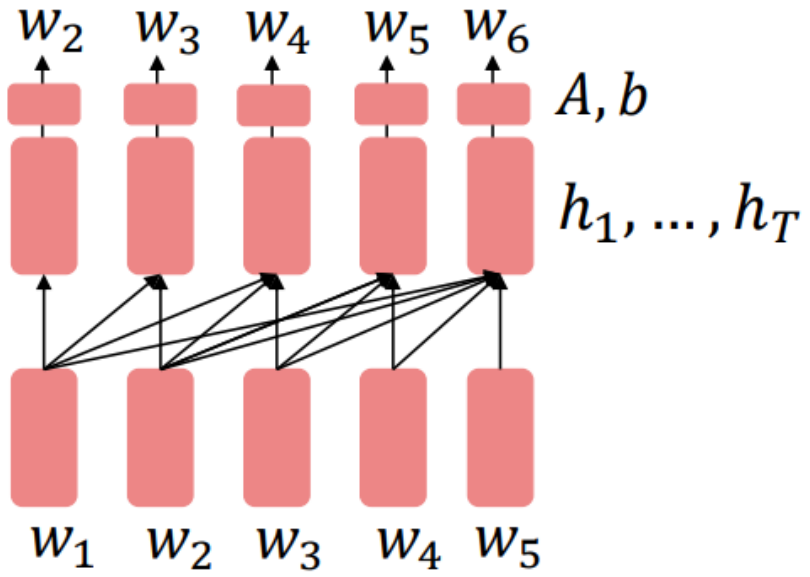
Pretraining Decoders

It’s natural to pretrain decoders as language models and then use them as generators, finetuning their $p_\theta(w_t \mid w_{1:t-1})$

This is helpful in tasks **where the output is a sequence** with a vocabulary like that at pretraining time!

- Dialogue (context=dialogue history)
- Summarization (context=document)

$$h_1,\ldots,h_T = \text{Decoder}(w_1,\ldots,w_T)$$
$$w_t \sim Ah_{t-1}+b$$



Where A, b were pretrained in the language model!

[Note how the linear layer has been pretrained.]

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture9-pretraining.pdf

Increasingly Convincing Generations (GPT2) [Radford et al., 2019]

We mentioned how pretrained decoders can be used in their capacities as language models. GPT-2, a larger version (1.5B) of GPT trained on more data, was shown to produce relatively convincing samples of natural language.

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.
GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

GPT-3, In-context Learning, and Very Large Models

So far, we've interacted with pretrained models in two ways:

- Sample from the distributions they define (maybe providing a prompt)
- Fine-tune them on a task we care about, and take their predictions.

Very large language models seem to perform some kind of learning **without gradient steps** simply from examples you provide within their contexts.

GPT-3 is the canonical example of this. The largest T5 model had 11 billion parameters.
GPT-3 has 175 billion parameters.

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture9-pretraining.pdf

LLaMA, Open-Source Models

Meta hopes to advance NLP research through LLaMA, particularly in the **academic exploration** of large language models.

LLaMA can be customized for **a variety of use cases**, especially in **research and non-commercial projects** where it demonstrates greater suitability.

Through architectural optimizations, LLaMA can achieve performance similar to GPT-3 while using fewer computational resources.

Llama 2: Open Foundation and Fine-Tuned Chat Models. https://arxiv.org/pdf/2307.09288

Final questions to be answered

- Why LM is so much powerful in AI?
- Why use Decoder-only (Autoregressive) LM?

Acknowledgement

- Princeton COS 484: Natural Language Processing. Contextualized Word Embeddings. Fall 2019
- CS447: Natural Language Processing. Language Models. *http://courses.engr.illinois.edu/cs447*

Claude Opus 5.5 is all you need!

How do you generate text from an *n*-gram model?

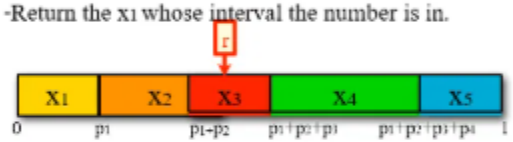
That is, how do you sample from a distribution $P(X|Y=y)$?

-Assume X has N possible outcomes (values): $\{x_1, \ldots, x_N\}$ and $P(X=x_i|Y=y) = p_i$

- Divide the interval [0,1] into N smaller intervals accordingto the probabilities of the outcomes

-Generate a random number r between 0 and 1.

-Return the x_1 whose interval the number is in.





用俄罗斯轮盘赌，做一个生成五个词的动画

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?'

用上面这个例子