CSC6203/CIE6021:
Large Language Model

Lecture 2: language model and beyond

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

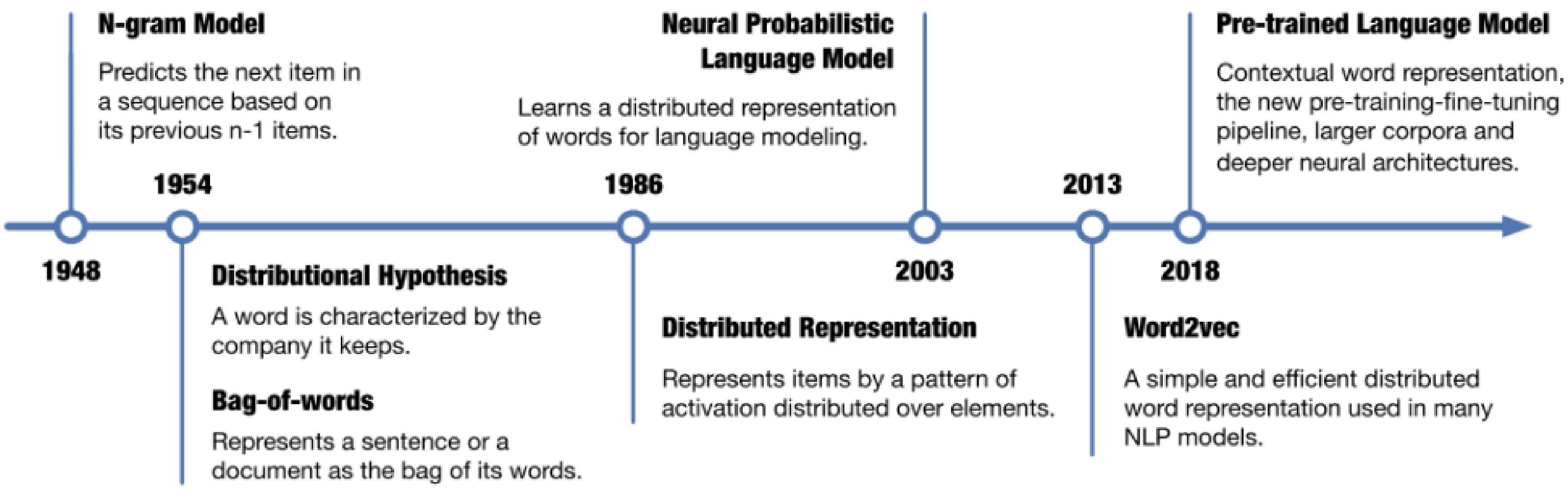
Winter 2023

Benyou Wang
School of Data Science

To recap…

Background

- language model



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

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?

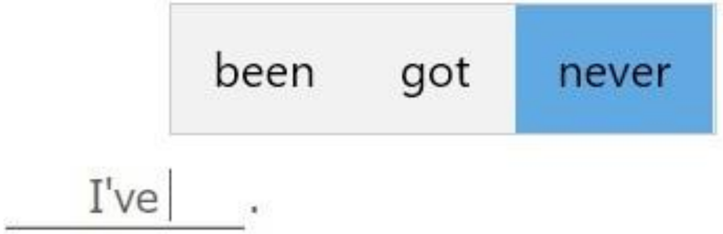
A **language model** assigns a probability to a N-gram

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

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

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

And $\boldsymbol{p(w_n|w_1} \cdot$ $= \dfrac{f(w_1 \cdots w_n)}{f(w_1 \cdots w_{n-1})}$



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

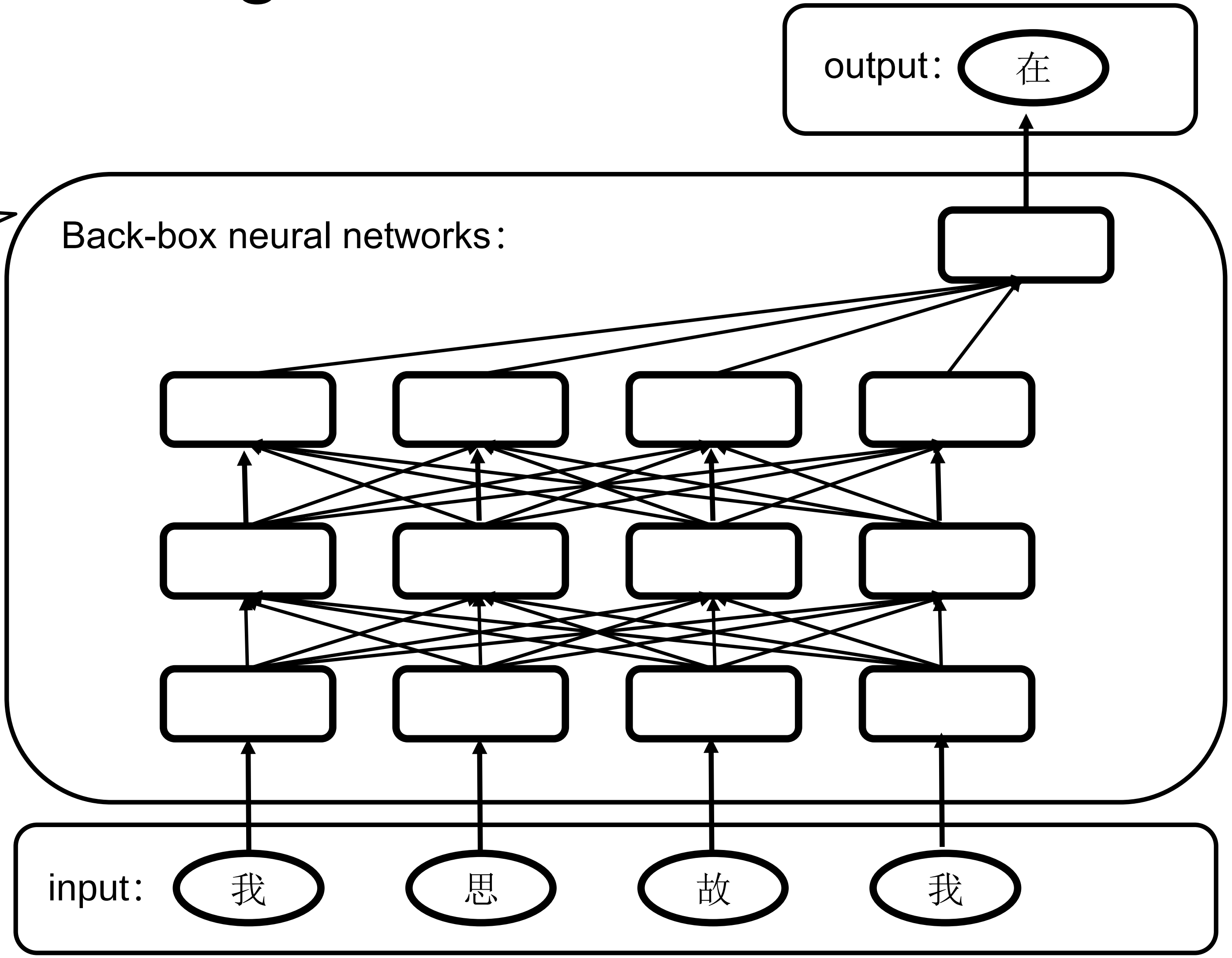
$$g: (V^{n-1}, V) \rightarrow R^+$$

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

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

Language model using neural networks

GPT-3/ChatGPT/GPT4 have 175B+ parameters
Humans have 100B+ neurons



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"

$$
\begin{aligned}
P(\text{the cat sat on the mat}) = {} & P(\text{the}) * P(\text{cat}|\text{the}) * P(\text{sat}|\text{the cat}) \\
& * P(\text{on}|\text{the cat sat}) * P(\text{the}|\text{the cat sat on}) \\
& * P(\text{mat}|\text{the cat sat on the})
\end{aligned}
$$

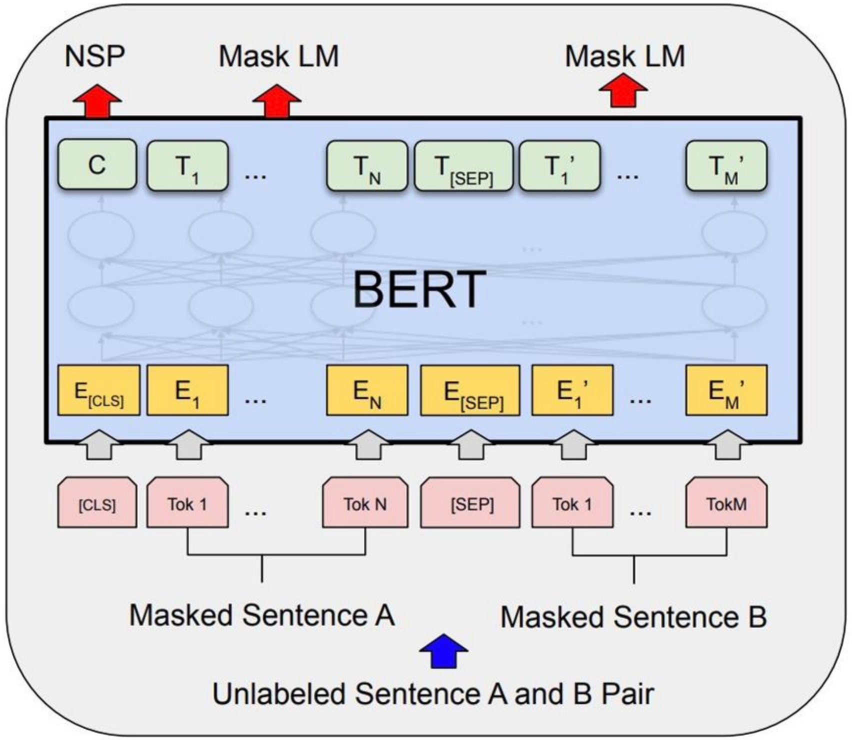
Implicit order

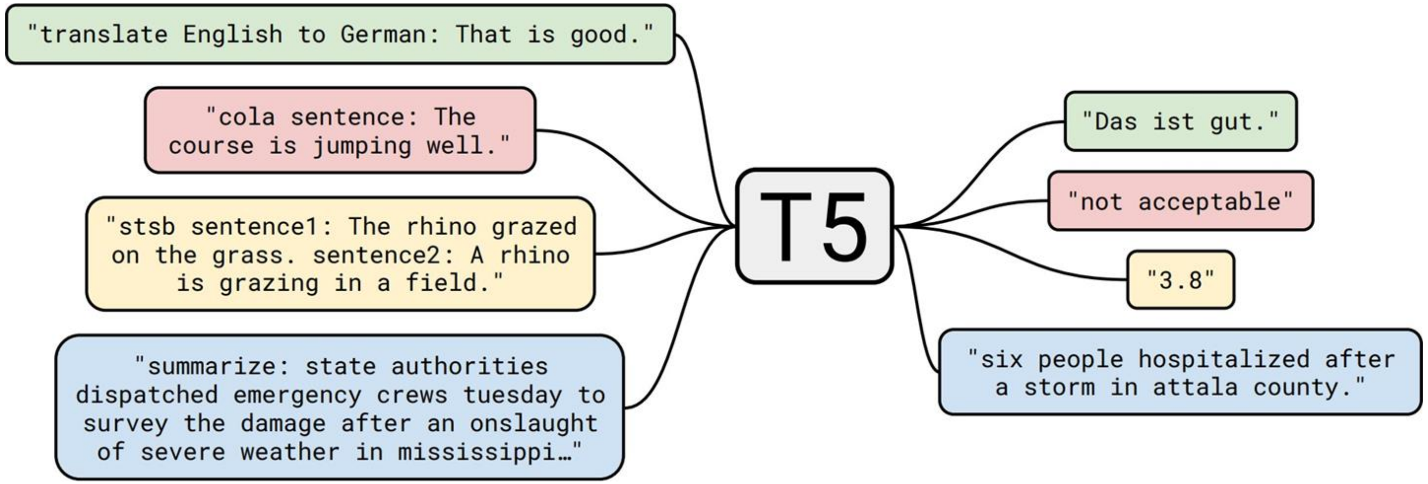
GPT-3 still acts in this way but the model is implemented as a very large neural network of 175-billion parameters!

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 model in a narrow sense**
  (Probability theory, N-gram language model)

- Language model in broad sense

- More thoughts on language model

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

Hmmm, but…

… what does it mean for a language model to “define a probability distribution”?

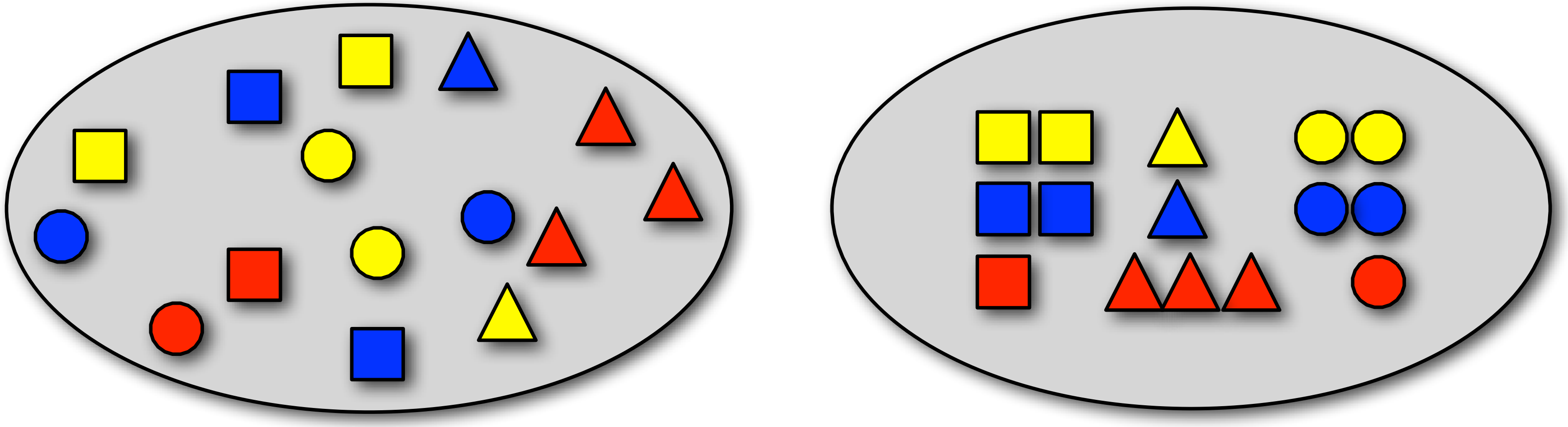
… *why* would we want to define probability distributions over languages?

… how can we construct a language model such that it *actually* defines a probability distribution?

Reminder:
Basic Probability Theory

Sampling with replacement

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



$P(■) = 2/15$

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

$P(\text{blue} \mid □) = 2/5$

$P(■) = 1/15$

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

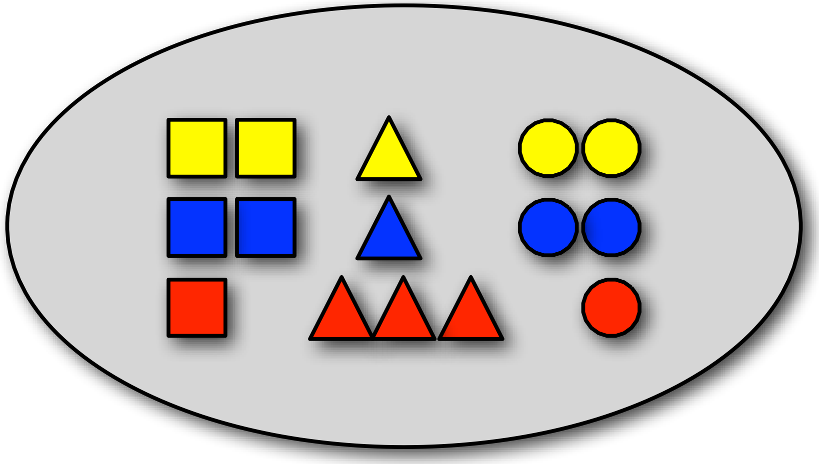
$P(□) = 5/15$

$P(■ \text{ or } ▲) = 2/15$

$P(△ \mid \text{red}) = 3/5$

Sampling with replacement

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



$P(●▲▲■)$
$= 1/15 \times 1/15 \times 1/15 \times 2/15$
$= 2/50625$

$P(▲●●▲)$
$= 3/15 \times 2/15 \times 2/15 \times 3/15$
$= 36/50625$

$P(■) = 2/15$
$P(\text{blue}) = 5/15$
$P(\text{blue} \mid □) = 2/5$

$P(■) = 1/15$
$P(\text{red}) = 5/15$
$P(□) = 5/15$

$P(■ \text{ or } ▲) = 2/15$
$P(△ \mid \text{red}) = 3/5$

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?'
```

$P(\texttt{of}) = 3/66$

$P(\texttt{Alice}) = 2/66$

$P(\texttt{was}) = 2/66$

$P(\texttt{to}) = 2/66$

$P(\texttt{her}) = 2/66$

$P(\texttt{sister}) = 2/66$

$P(\texttt{,}) = 4/66$

$P(\texttt{'}) = 4/66$

Sampling with replacement

```
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, $P(\mathit{English\ sentence}) = P(\mathit{word\ salad})$

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: $\Omega \rightarrow \mathrm{T}$

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:

$$0 \leq P(\omega \subseteq \Omega) \leq 1$$

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

$$P(\emptyset) = 0$$

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

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

$$\text{and } \bigcup_i \omega_i = \Omega$$

Discrete probability distributions: single trials

'Discrete': a fixed (often finite) number of outcomes

Bernoulli distribution (two possible outcomes)

Defined by the probability of success (= head/yes)

The probability of *head* is p. The probability of *tail* is $1-p$.

Categorical distribution (*N* possible outcomes $c_1 \dots c_N$)

The probability of category/outcome c_i is p_i ($0 \leq p_i \leq 1$; $\sum_i p_i = 1$).

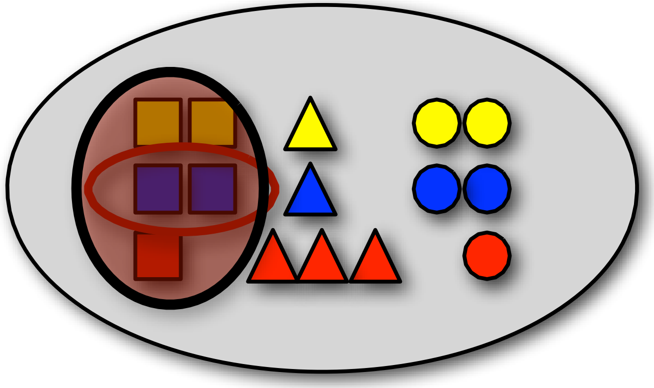
- e.g. the probability of getting a six when rolling a die once
- e.g. the probability of the next word (picked among a vocabulary of N words)

(NB: Most of the distributions we will see in this class are categorical.
Some people call them multinomial distributions, but those refer to *sequences* of trials, e.g. the probability of getting five sixes when rolling a die ten times)

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

$$\begin{aligned} P(X_1, X_2, \ldots, X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_2, X_1)....P(X_n|X_1, ...X_{n-1}) \\ &= P(X_1)\prod_{i=2}^{n} P(X_i|X_1 \ldots X_{i-1}) \end{aligned}$$

Independence

Two random variables X and Y are independent if

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

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

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

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

$$= P(X)$$

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

Language modeling with N-grams

A language model over a vocabulary V
assigns probabilities to strings drawn from $\mathrm{V}*$.

Recall the **chain rule**:
$P(w^{(1)} \ldots w^{(i)}) = P(w^{(1)}) \cdot P(w^{(2)} \mid w^{(1)}) \cdot \ldots \cdot P(w^{(i)} \mid w^{(i-1)}, \ldots, w^{(1)})$

An **n-gram** language model assumes each word
depends only on **the last n−1 words**:
$P_{ngram}(w^{(1)} \ldots w^{(i)}) = P(w^{(1)}) \cdot P(w^{(2)} \mid w^{(1)}) \cdot \ldots \cdot P(w^{(i)} \mid w^{(i-1)}, \ldots, w^{(1-(n+1))})$

N-gram models

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

Unigram model: $P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)})$

Bigram model: $P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} \mid w^{(i-1)})$

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

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?'
```

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

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

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

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

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

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

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

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

Using a bigram model for Alice

English

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?'

Word Salad

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

Now, $P(\textit{English}) \gg P(\textit{word salad})$

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

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

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

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

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

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

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

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

Where do we get the probabilities from?

Learning (estimating) a language model

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

$$P(w^{(i)} = \text{‘}the\text{’} \mid w^{(i-1)} = \text{‘}on\text{’}) = ???$$

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)

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

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 $P_{ngram}(w^{(1)} \dots w^{(N)})$ in terms of the
probability of predicting each word: $P_{bigram}(w^{(1)} \dots w^{(N)}) = \prod_{i=1 \dots N} P(w^{(i)} \mid w^{(i-1)})$

With a fixed vocabulary V, it's easy to make sure $P(w^{(i)} \mid w^{(i-1)})$
is a distribution: $\sum_{i=1 \dots |V|} P(w_i \mid w_j) = 1$ and $\forall_{i,j}\, 0 \leq P(w_i \mid w_j) \leq$

If $P(w^{(i)} \mid w^{(i-1)})$ 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

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

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

If this bothers you:

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

`BOS1 BOS2 Alice was …`

Now the unigram and bigram probabilities involve only BOS symbols.

To recap…

Estimating a bigram models with BOS (<s>), EOS (</s>) and UNK using MLE

1. Replace all rare words in training corpus with UNK
2. Bracket each sentence by special start and end symbols:

   `<s> Alice was beginning to get very tired… </s>`

3. Vocabulary V’ = all tokens in modified training corpus

   (all common words, UNK, <s>, </s>)

4. Count the frequency of each bigram….

   C(<s> Alice) = 1, *C*(Alice was) = 1, …

5. .... and normalize these frequencies to get probabilities:

$$P(\textit{was} \mid \textit{Alice}) = \frac{C(\textit{Alice was})}{\sum_{w_i \in V'} C(\textit{Alice } w_i)}$$

Using language models

How do we use language models?

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:

$$\text{argmax}_{S_{Out}} P(S_{Out} \mid \text{Input}) = \text{argmax}_{S_{Out}} P(\text{Input} \mid S_{Out}) P(S_{Out})$$

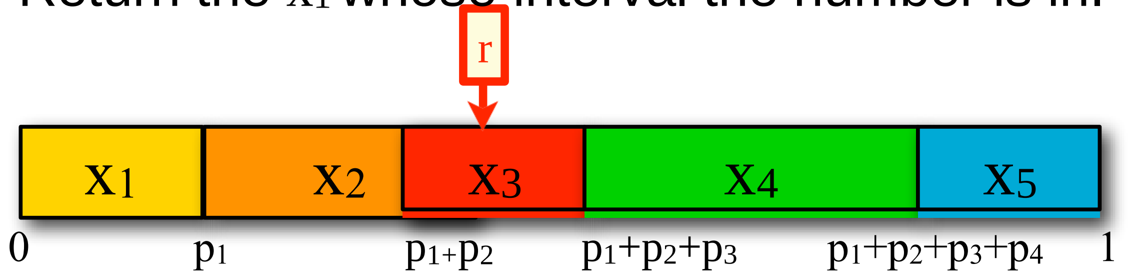
Using n-gram models to generate language

Generating from a distribution

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

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

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

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.

4-grams *look* like Shakespeare because they *are* Shakespeare!

MLE doesn’t capture unseen events

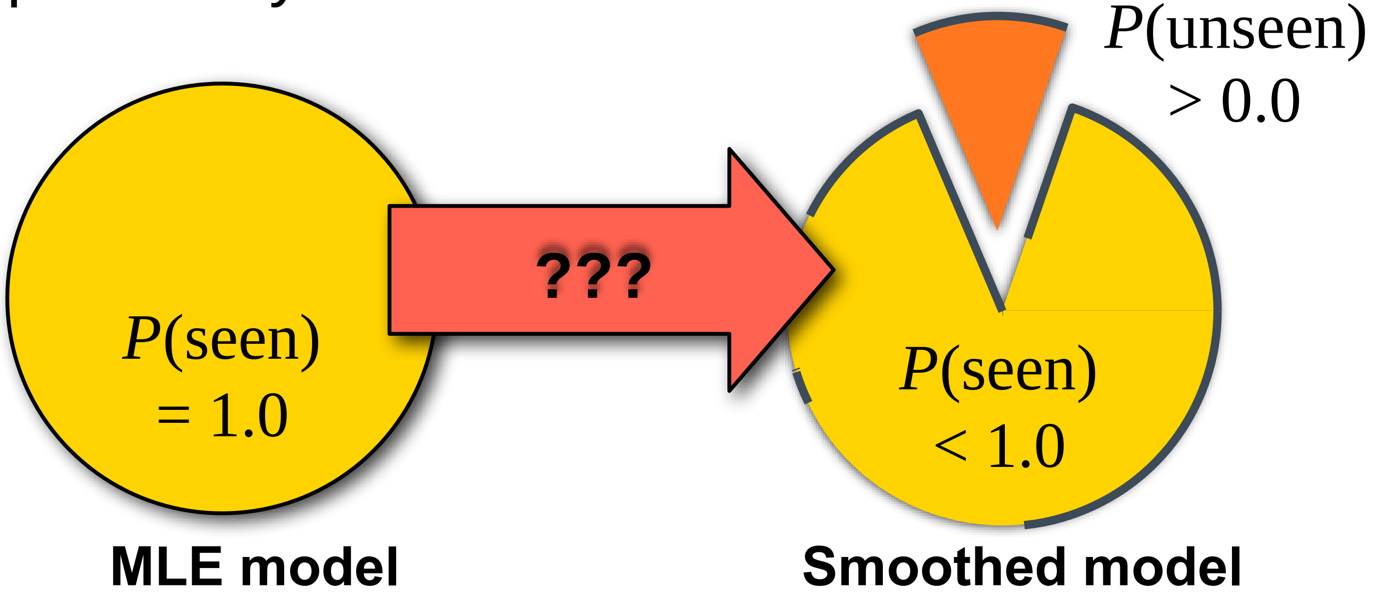
We estimated a model on 440K word tokens, but:

Only 30,000 word types occur in the training data
Any word that does not occur in the training data
has zero probability!

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 we assign non-zero probability to unseen events?

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



We won’t talk much about smoothing this year.

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

Good-Turing: Use probability of rare events to estimate probability of unseen events

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 models is defined as the inverse ($\frac{1}{P(\ldots)}$) of the probability of the test set, normalized ($\sqrt[N]{\ldots}$) by the # 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

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

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

$$PP(w_1 \dots w_N) =_{def} \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1},\dots,w_{i-n+1})}}$$

with

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

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.

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?

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

Insertions: "eat lunch" → "eat ***a*** lunch"

Deletions: "see ***a*** movie" → "see movie"

Substitutions: "drink ***ice*** tea" → "drink ***nice*** tea"

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

Contents

- **Language model in a narrow sense**
  (Probability theory, N-gram language model)

- **Language model in broad sense**
  (BERT and beyond)

- More thoughts on language model

More on N-gram LMs

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: $x^{(\&'!)}$ depends only on the preceding n-1 words

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}})$$

(assumption)

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{\boxed{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}}{\boxed{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}}$$

(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!

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}$$

(statistical approximation)

N-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

~~*as the proctor started the clock, the*~~ $\underbrace{\textit{students opened their}}_{\text{condition on this}}$ __

discard

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

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

Should we have discarded the "proctor" context?

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*.

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

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

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

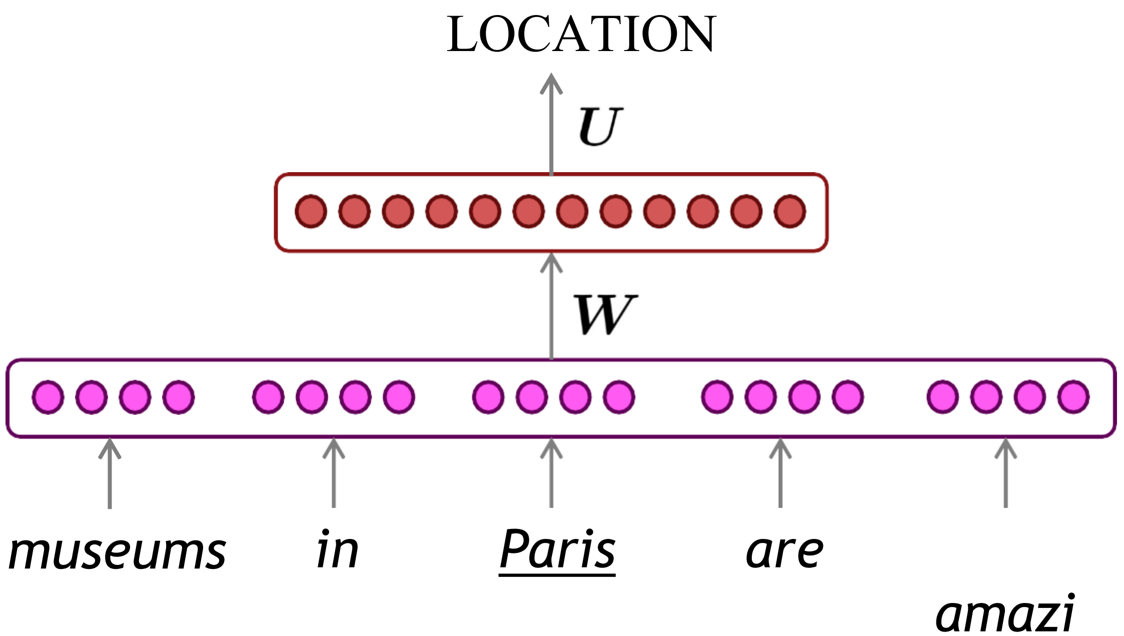
$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\boxed{\text{count}(\text{students opened their } \boldsymbol{w})}}{\text{count}(\text{students opened their})}$$

Increasing *n* or increasing corpus size 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 in Lecture 2:



A fixed-window neural Language Model

output distribution

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

hidden layer

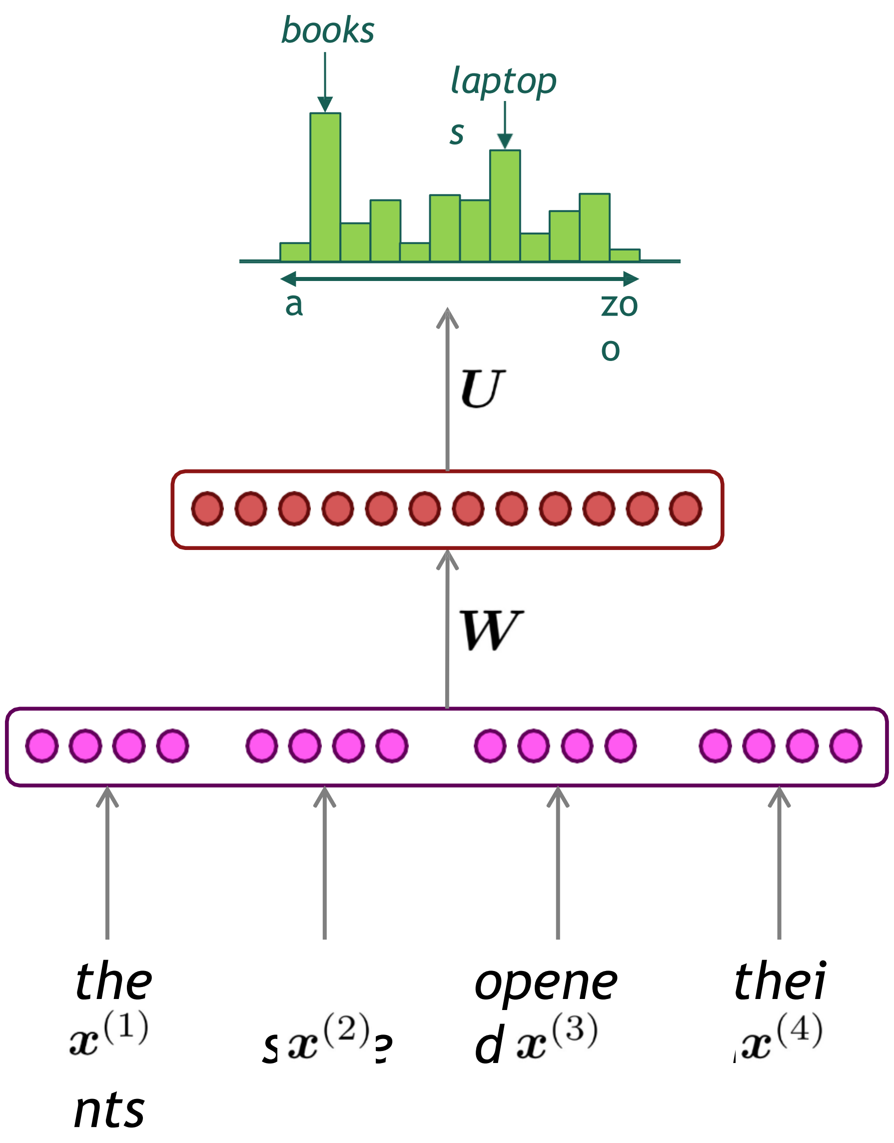
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

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

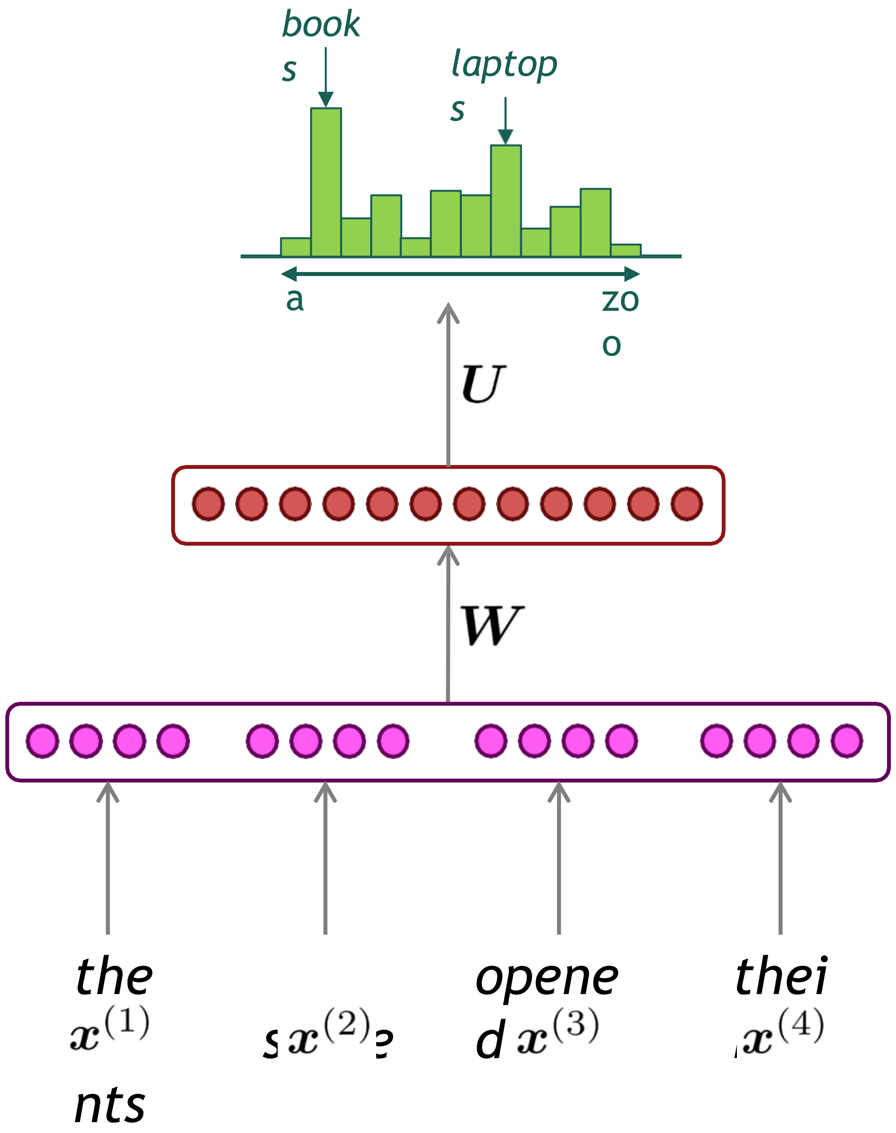
Approximately: Y. Bengio, et al. (2000/2003): A Neural Probabilistic Language Model

Improvements over *n*-gram LM:

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

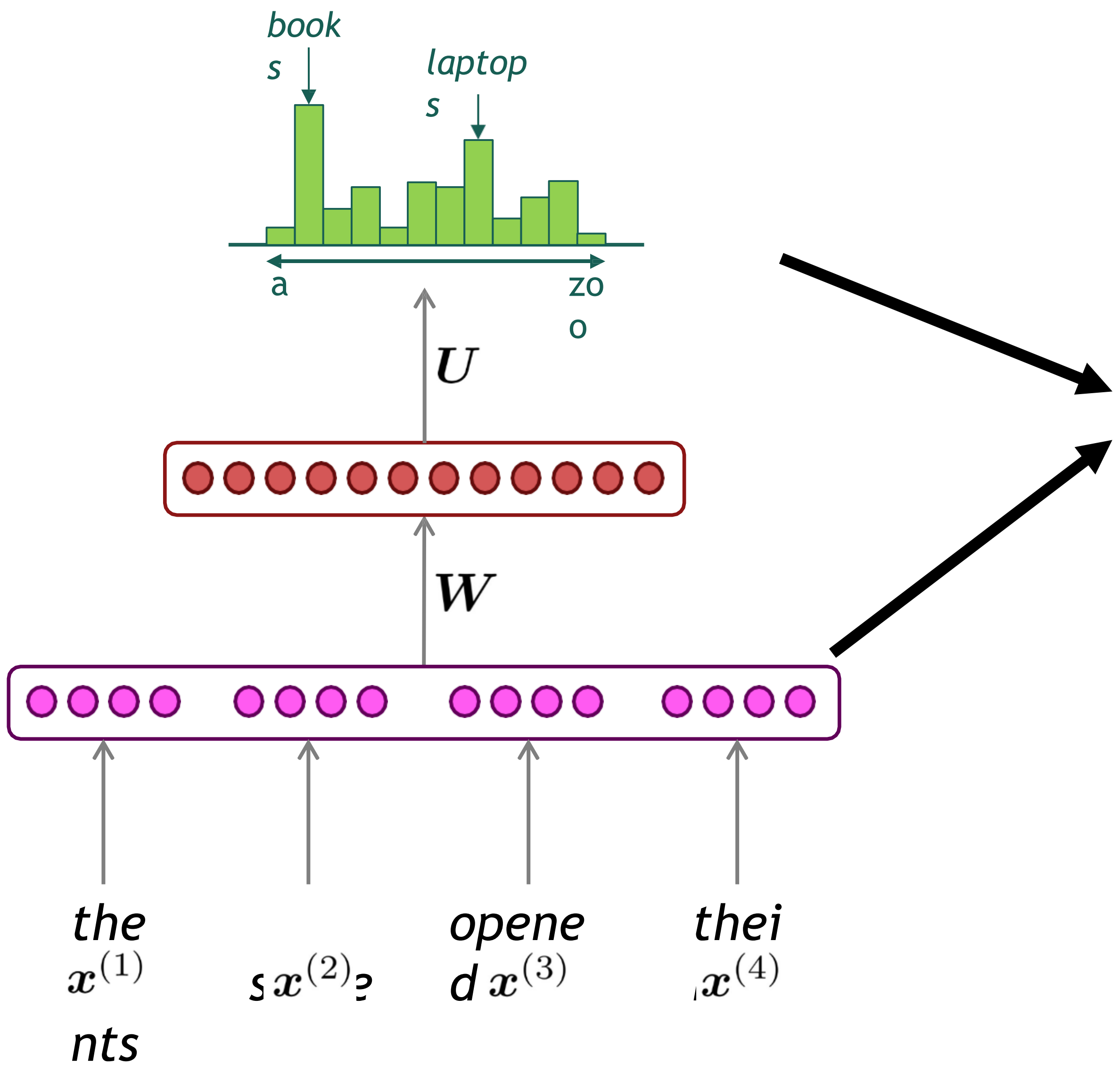
Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges W
- Window can never be large enough!
- $x^{(!)}$ and $x^{(")}$ 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*

From N-gram LMs to Word vectors



Word embedding/Vectors !

How do we represent the meaning of a word?

Definition: meaning (Webster dictionary)

- ❑ the idea that is represented by a word, phrase, etc.
- ❑ the idea that a person wants to express by using words, signs, etc.
- ❑ the idea that is expressed in a work of writing, art, etc.

Commonest linguistic way of thinking of meaning:

- ❑ signifier (symbol) ⇔ signified (idea or thing)
  = denotational semantics
- ❑ Tree ⇔ {🌳, 🌲, 🌴, …}

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

Representing words as discrete symbols

- In traditional NLP, we regard words as discrete symbols:
  hotel, conference, motel – a localist representation

- Such symbols for words can be represented by one-hot vectors:
  motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
  hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

- Vector dimension = number of words in vocabulary (e.g., 500,000+)

These two vectors are orthogonal

There is no natural notion of similarity for one-hot vectors!

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

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***

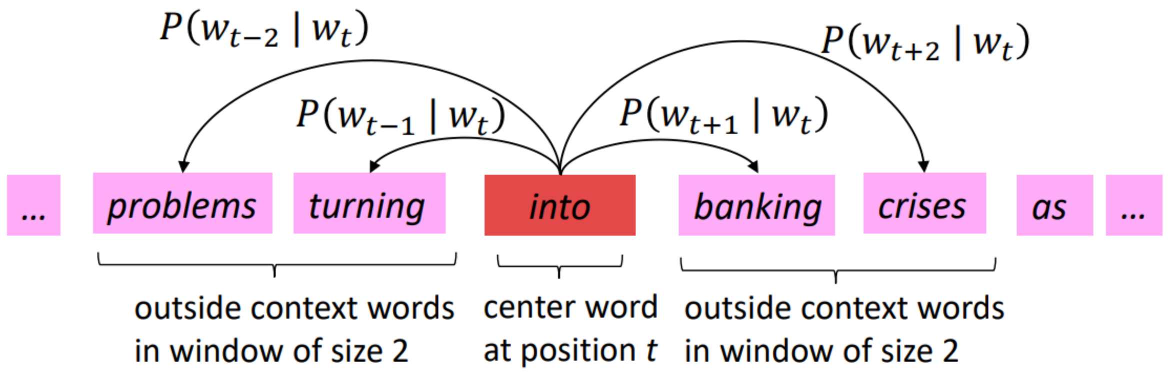
https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture01-wordvecs1.pdf

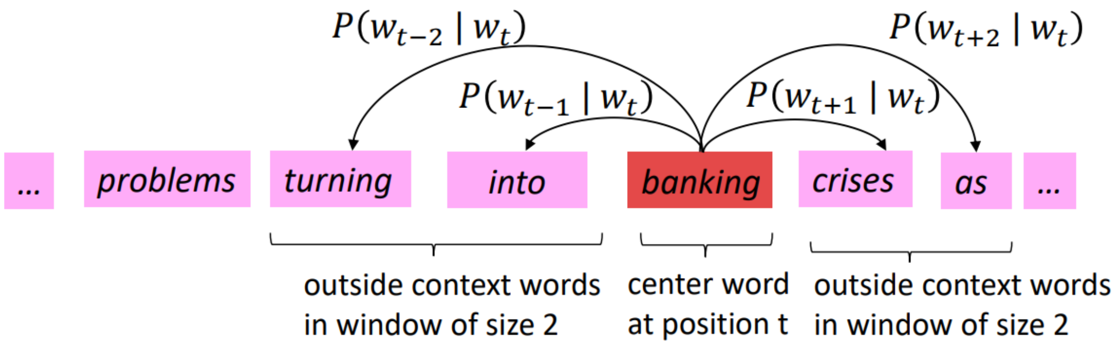
Word2Vec Overview

Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

Idea:

- We have a large corpus ("body") of text: a long list of words
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context ("outside") words o
- Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
- Keep adjusting the word vectors to maximize this probability





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

Word2vec: objective function

- We want to minimize the objective function:

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

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

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)

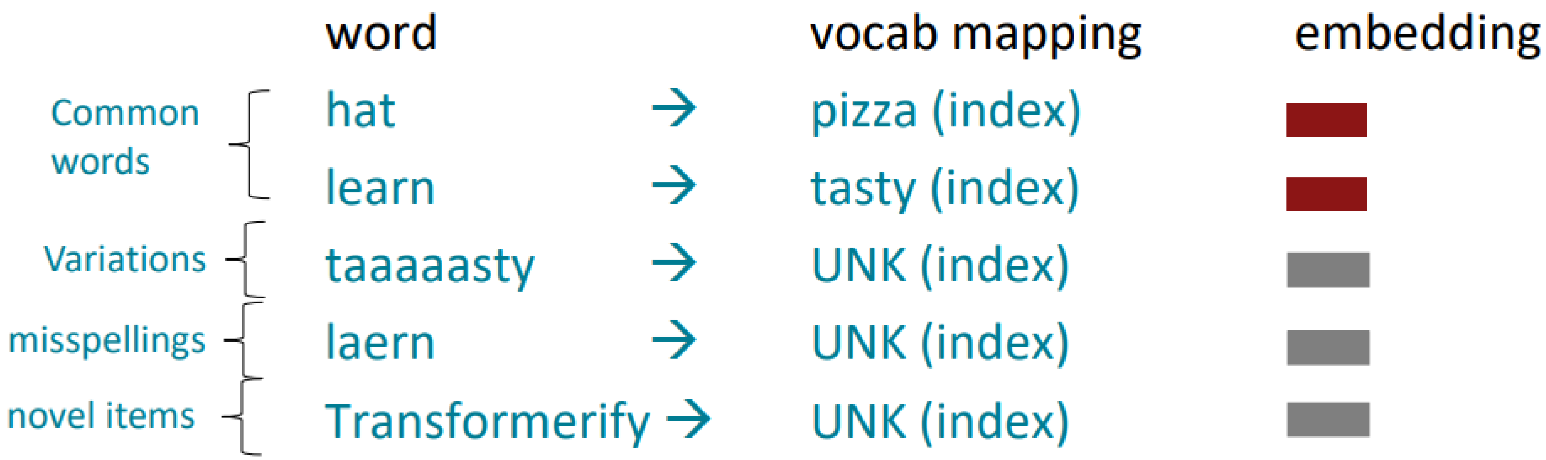
$$P(o \mid c) = \frac{\exp\left(u_o^T v_c\right)}{\sum_{w \in V} \exp\left(u_w^T v_c\right)}$$

“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

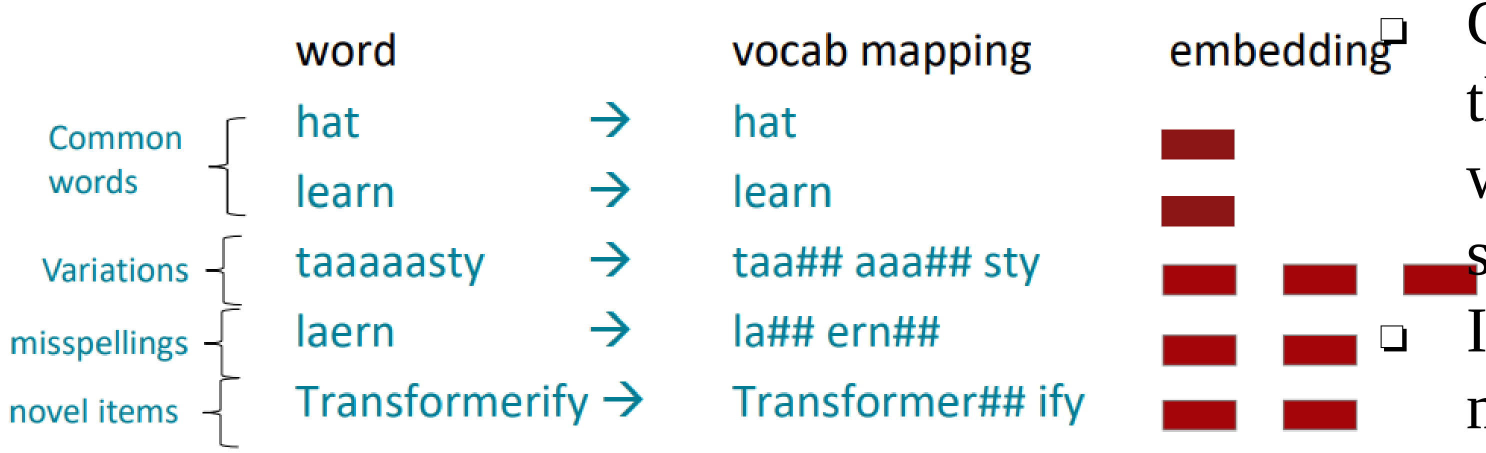
Word structure and subword models

Subword modeling in NLP encompasses a wide range of methods for reasoning about structure below the word level. (Parts of words, characters, bytes.)

- The dominant modern paradigm is to learn a vocabulary of **parts of words (subword tokens)**.
- At training and testing time, each word is split into a sequence of known subwords.

Byte-pair encoding is a simple, effective strategy for defining a subword vocabulary.

1. Start with a vocabulary containing only characters and an “end-of-word” symbol.
2. Using a corpus of text, find the most common adjacent characters “a,b”; add “ab” as a subword.
3. Replace instances of the character pair with the new subword; repeat until desired vocab size.



- ❑ Common words end up being a part of the subword vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components.
- ❑ In the worst case, words are split into as many subwords as they have characters.

From static word vector to contextualized word vectors

What's wrong with word2vec?

- One vector for each word type

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

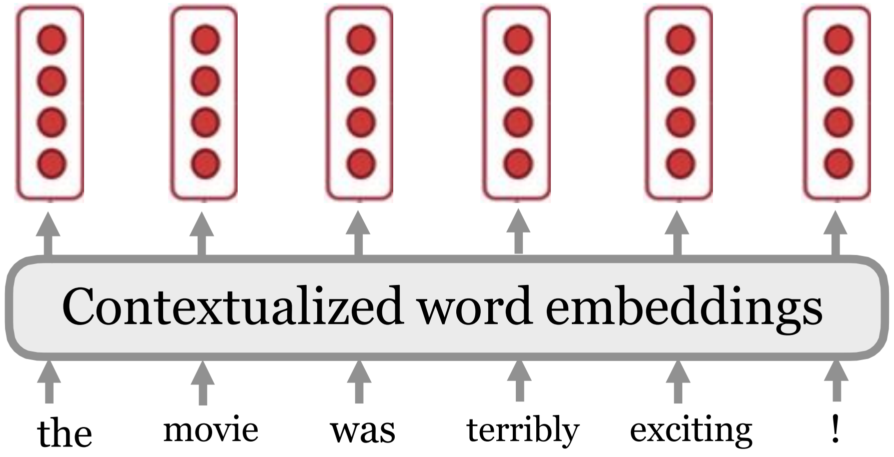
- Complex characteristics of word use: semantics, syntactic behavior, and connotations

- 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**!



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

Contextualized word embeddings

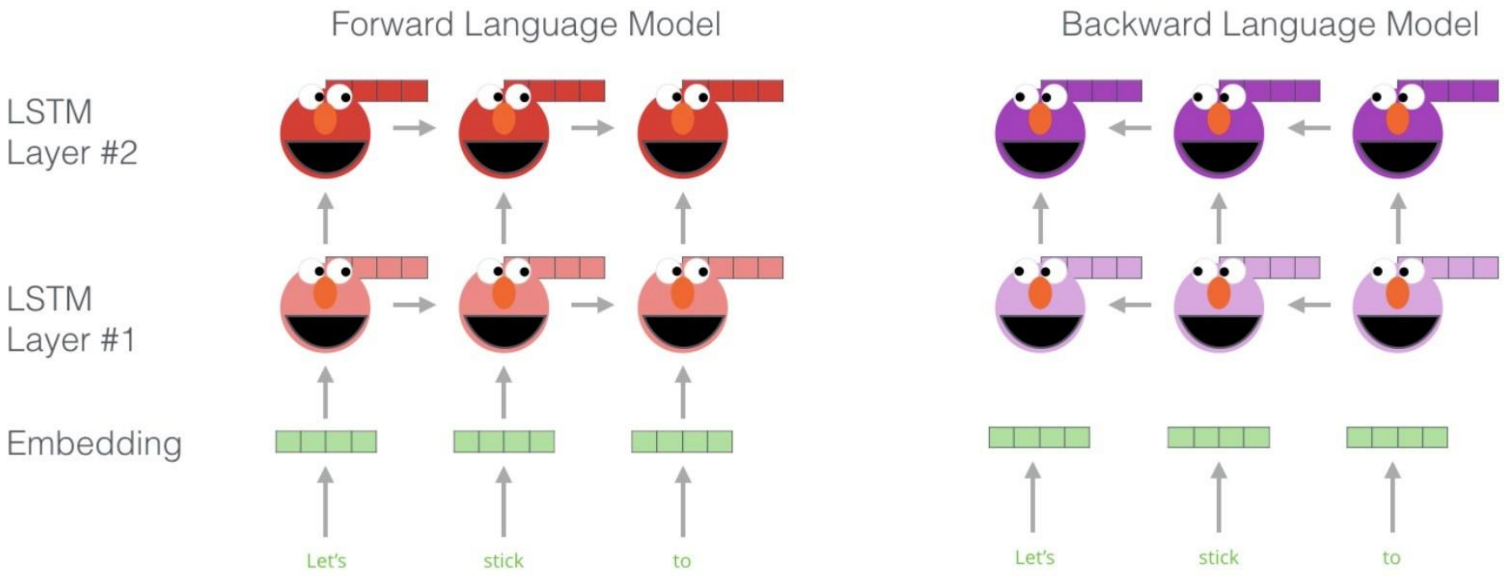
	Source	Nearest Neighbors
GloVe	play	playing, game, games, played, players, plays, player, Play, football, multiplayer
biLM	Chico Ruiz made a spectacular play on Alusik 's grounder $\{\ldots\}$	Kieffer , the only junior in the group , was commended for his ability to hit in the clutch , as well as his all-round excellent play .
	Olivia De Havilland signed to do a Broadway play for Garson $\{\ldots\}$	$\{\ldots\}$ they were actors who had been handed fat roles in a successful play , and had talent enough to fill the roles competently , with nice understatement .

(Peters et al, 2018): Deep contextualized word representations

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



words in the sentence

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

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

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

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

- 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

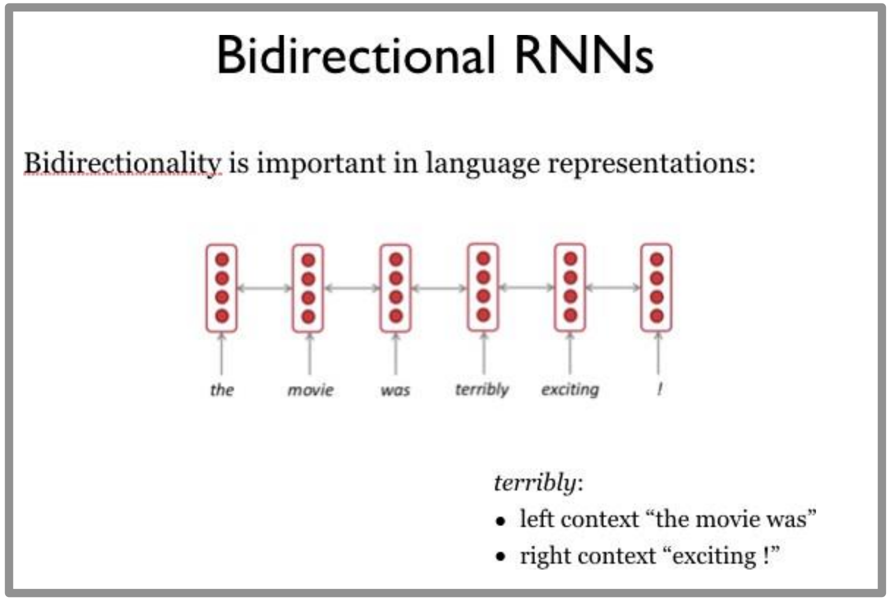
#2. LSTMs vs Transformers (will talk later)

#3. The weights are not freezed, 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

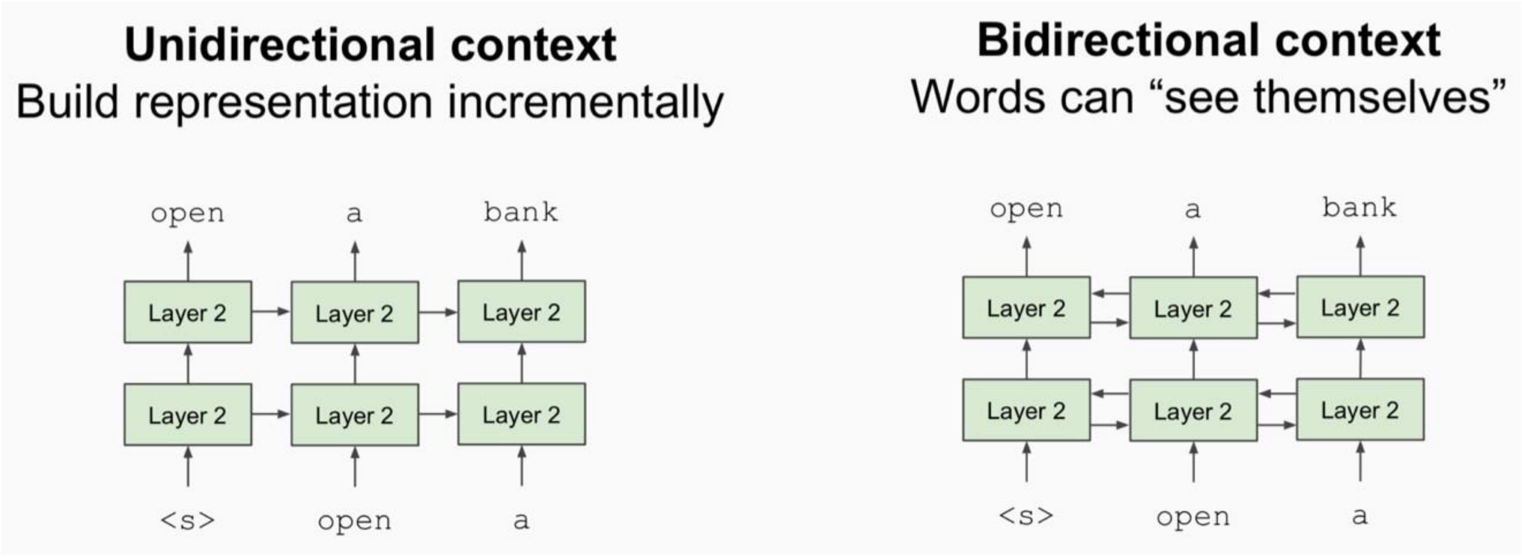
Lecture 9:



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



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

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] is never seen when BERT is used…

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]

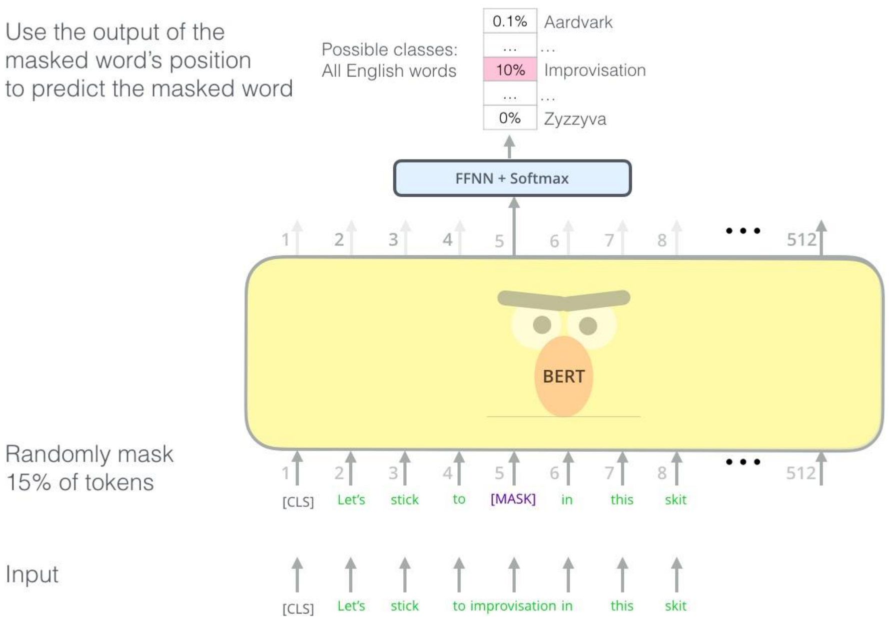
penguin [MASK] are flight ##less birds [SEP]

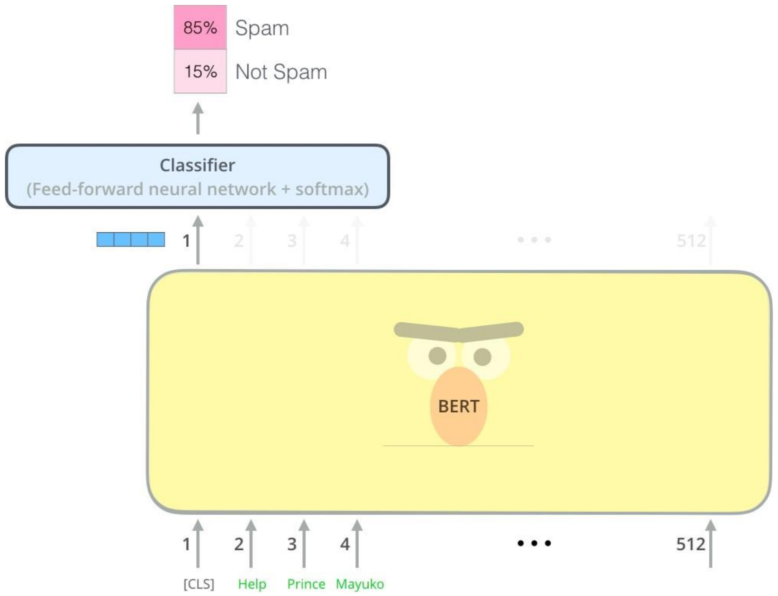
Label = NotNext

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

Pre-training and fine-tuning



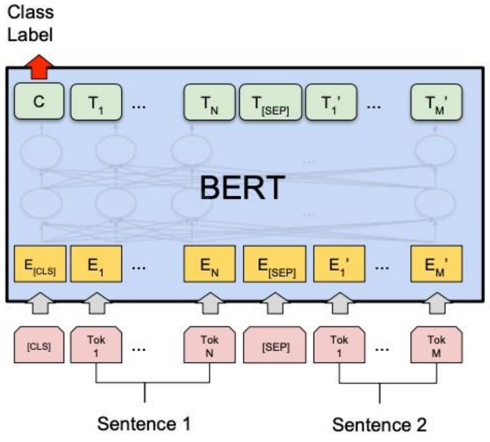


Pre-training

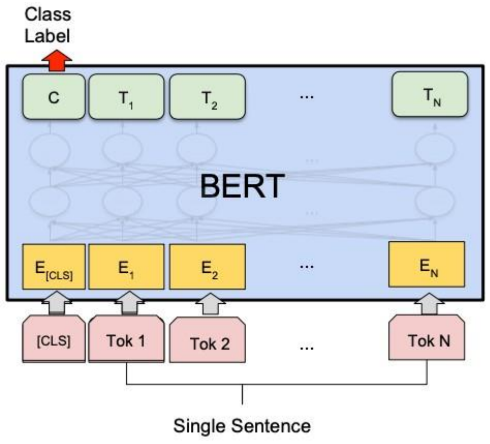
Fine-tuning

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

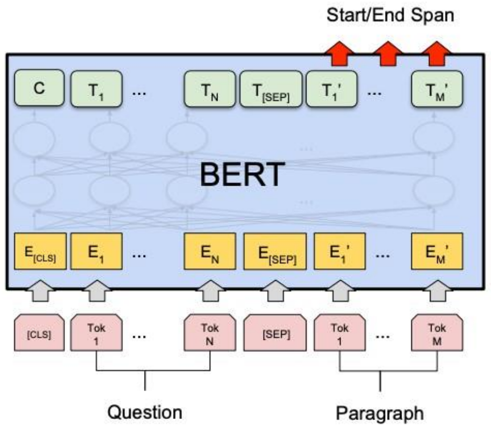
Applications



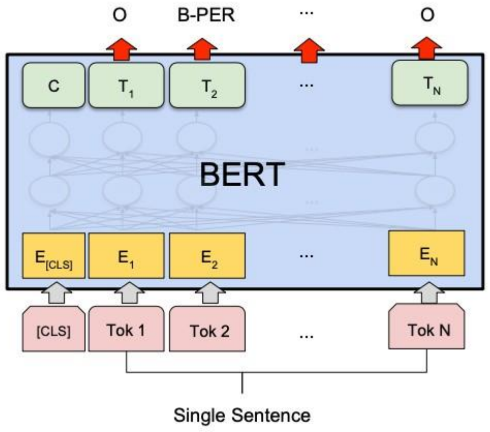
(a) Sentence Pair Classification Tasks:
MNLI, QQP, QNLI, STS-B, MRPC,
RTE, SWAG



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



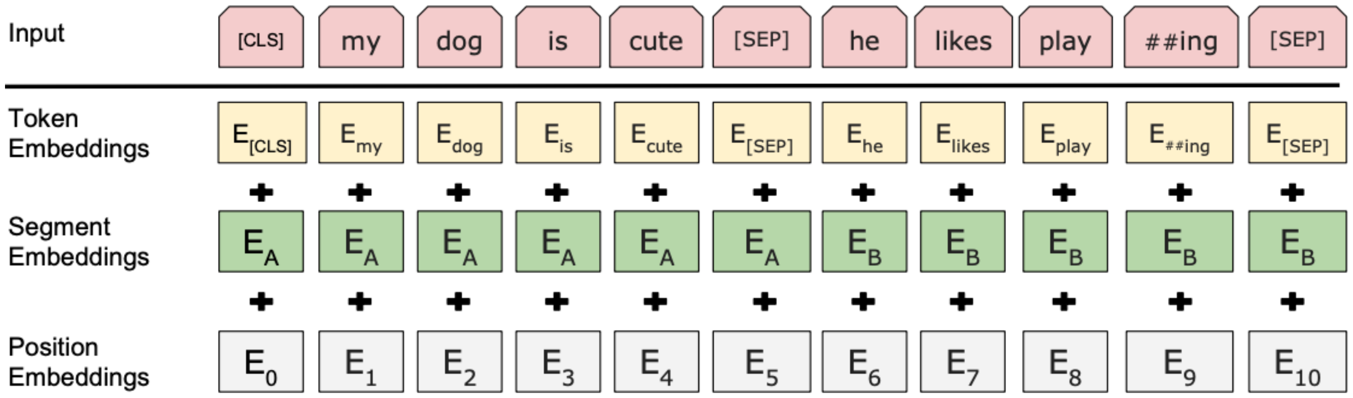
(c) Question Answering Tasks:
SQuAD v1.1



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

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

Variants of contextualized word vectors

Where we were: pretrained word embeddings

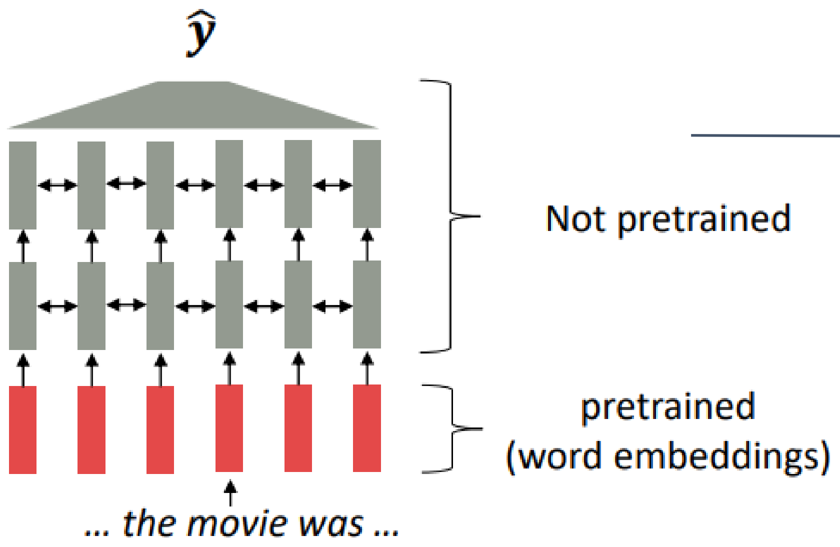
Some issues to think about:

- The training data we have for our downstream task (like question answering) must be sufficient to teach all contextual aspects of language.
- Most of the parameters in our network are randomlv initialized!

Where we're going: pretraining whole models

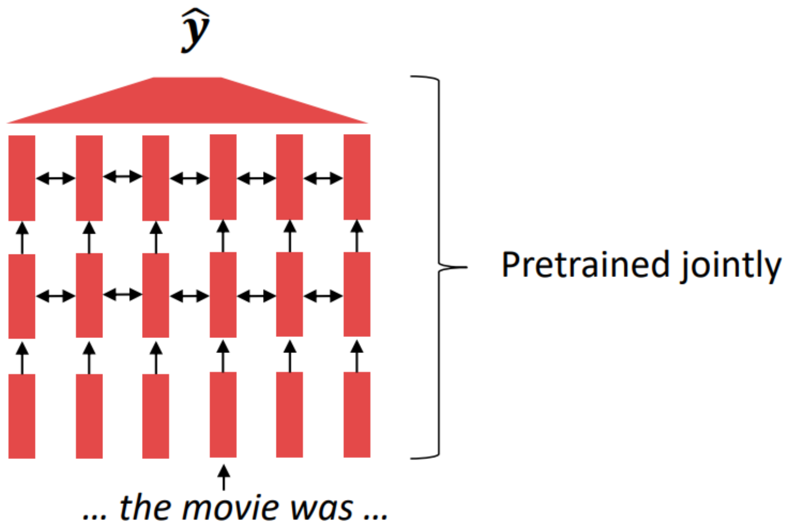
In modern NLP:

- All (or almost all) parameters in NLP networks are initialized via pretraining.
- Pretraining methods hide parts of the input from the model, and train the model to reconstruct those parts.



Stronger:

- representations of language
- parameter initializations for strong NLP models.
- Probability distributions over language that we can sample from



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

[Recall, *movie* gets the same word embedding, no matter what sentence it shows up in]

[This model has learned how to represent entire sentences through pretraining]

What can we learn from reconstructing the input?

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

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

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

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

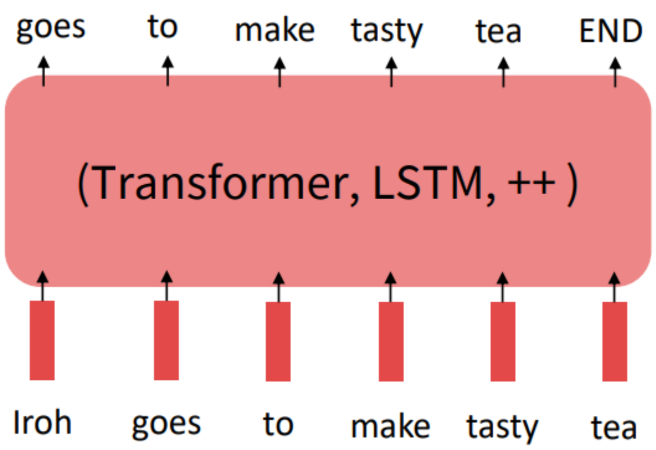
Pretraining through language modeling [Dai and Le, 2015]

Recall the language modeling task:
- Model $p_\theta(w_t \mid w_{1:t-1})$, the probability distribution over words given their past contexts.
- There's lots of data for this! (In English.)

Pretraining through language modeling:
- Train a neural network to perform language modeling on a large amount of text.
- Save the network parameters.

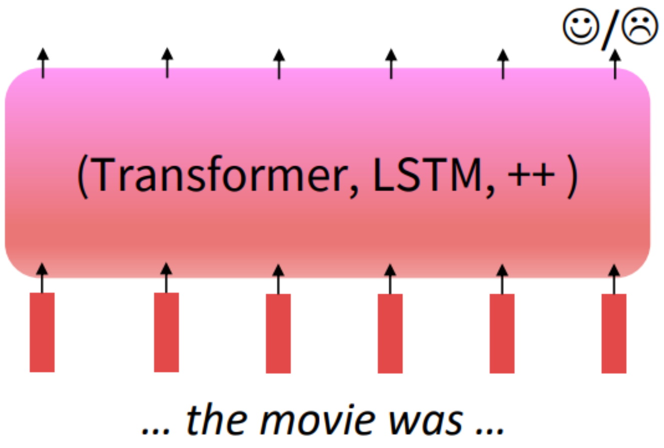
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



Step 2: Finetune (on your task)

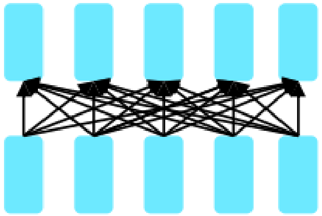
Not many labels; adapt to the task!



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

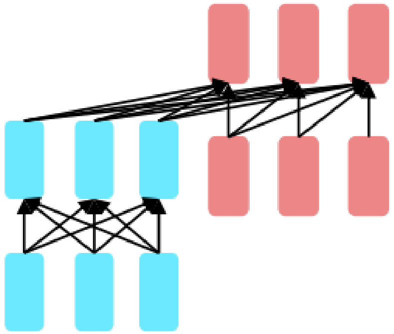
Pretraining for three types of architectures

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



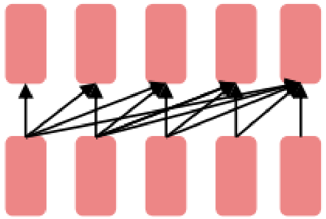
Encoders

- **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

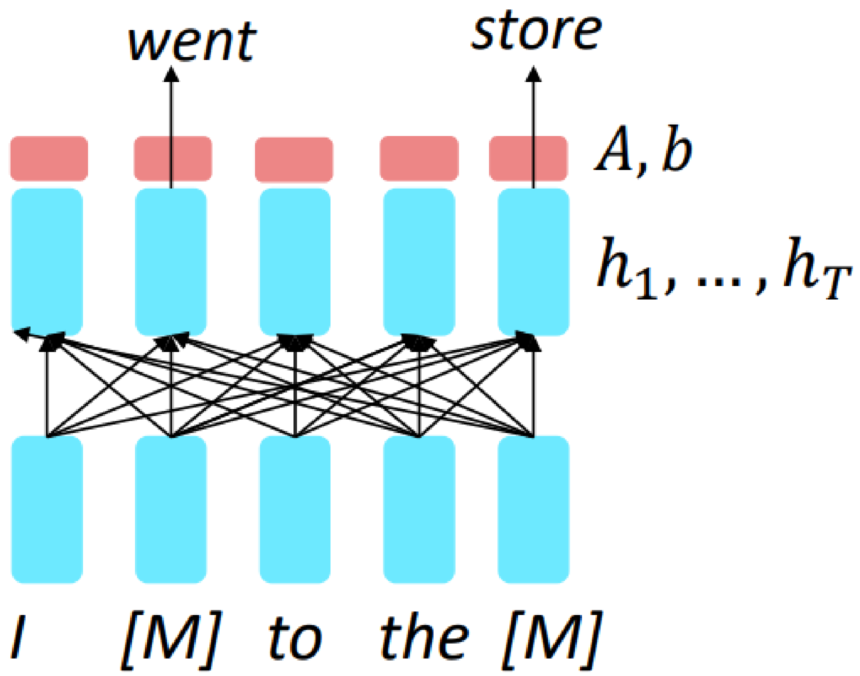
Pretraining encoders: what pretraining objective to use?

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 $p_\theta(x \mid \tilde{x})$. Called **Masked LM.**



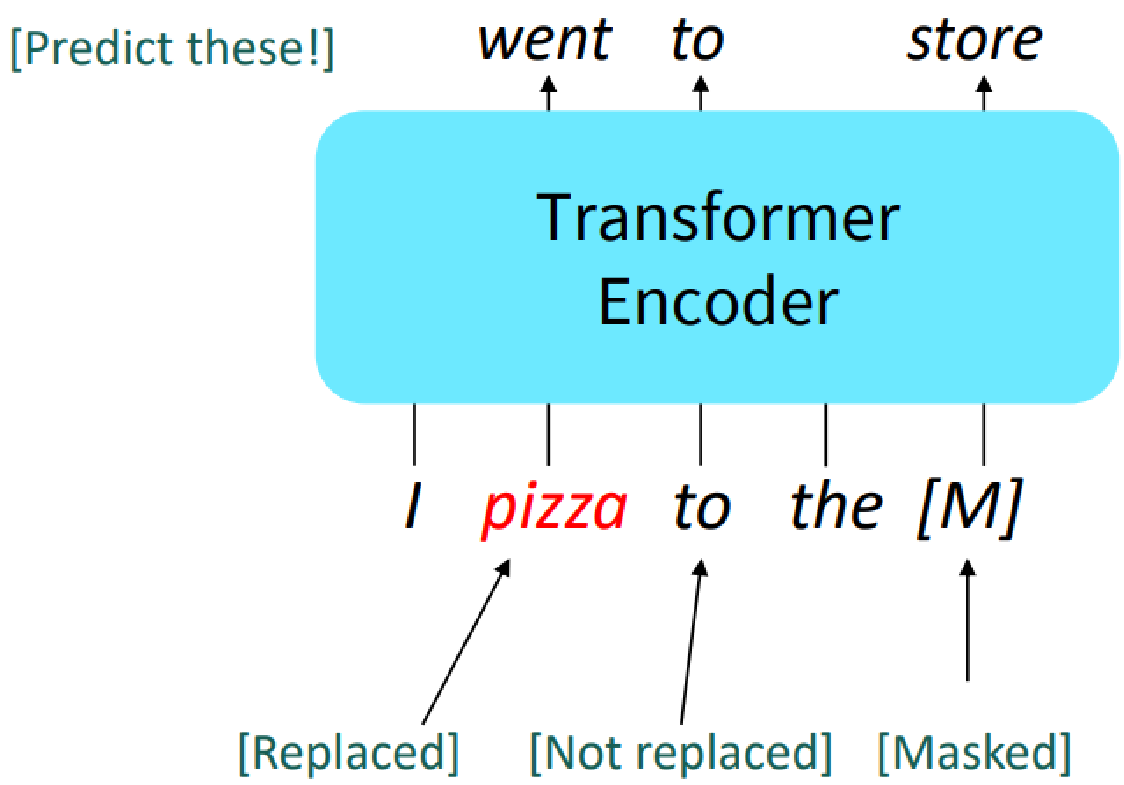
[Devlin et al., 2018]

BERT: Bidirectional Encoder Representations from Transformers

Devlin et al., 2018 proposed the “Masked LM” objective and released the weights of a pretrained Transformer, a model they labeled BERT.

Some more details about Masked LM for BERT:

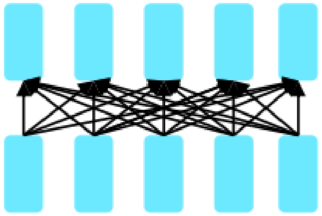
- Predict a random 15% of (sub)word tokens.
    - Replace input word with [MASK] 80% of the time
    - Replace input word with a random token 10% of the time
    - Leave input word unchanged 10% of the time (but still predict it!)
- Why? Doesn’t let the model get complacent and not build strong representations of non-masked words. (No masks are seen at fine-tuning time!)



[Devlin et al., 2018]

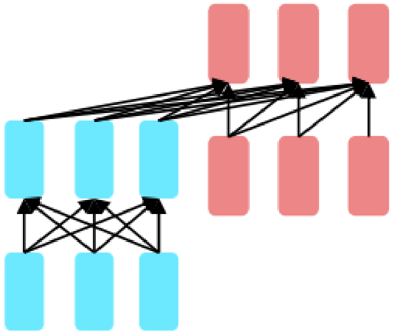
Pretraining for three types of architectures

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



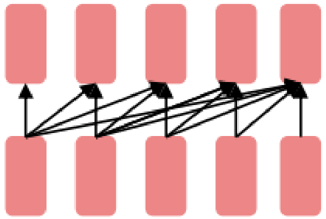
Encoders

- 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?**



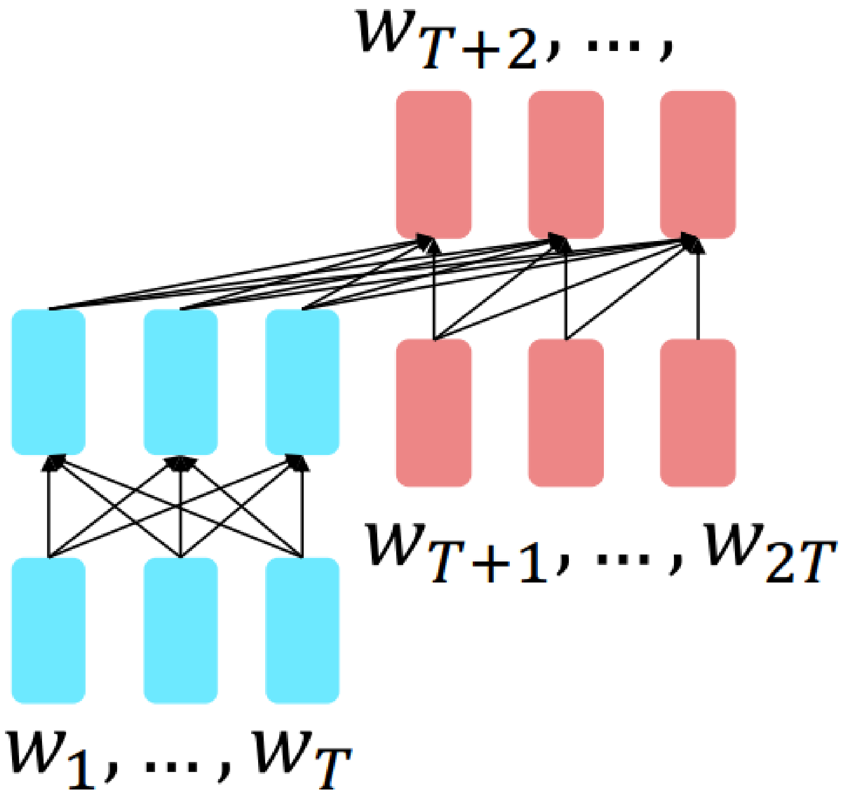
Decoders

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

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.

$$h_1, \ldots, h_T = \text{Encoder}\left(w_1, \ldots, w_T\right)$$
$$h_{T+1}, \ldots, h_2 = \text{Decoder}(w_1, \ldots, w_T, h_1, \ldots, h_T)$$
$$y_i \sim Ah_i + b, i > T$$



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

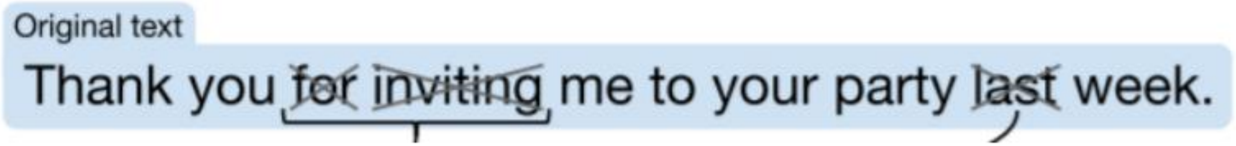
[Raffel et al., 2018]

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

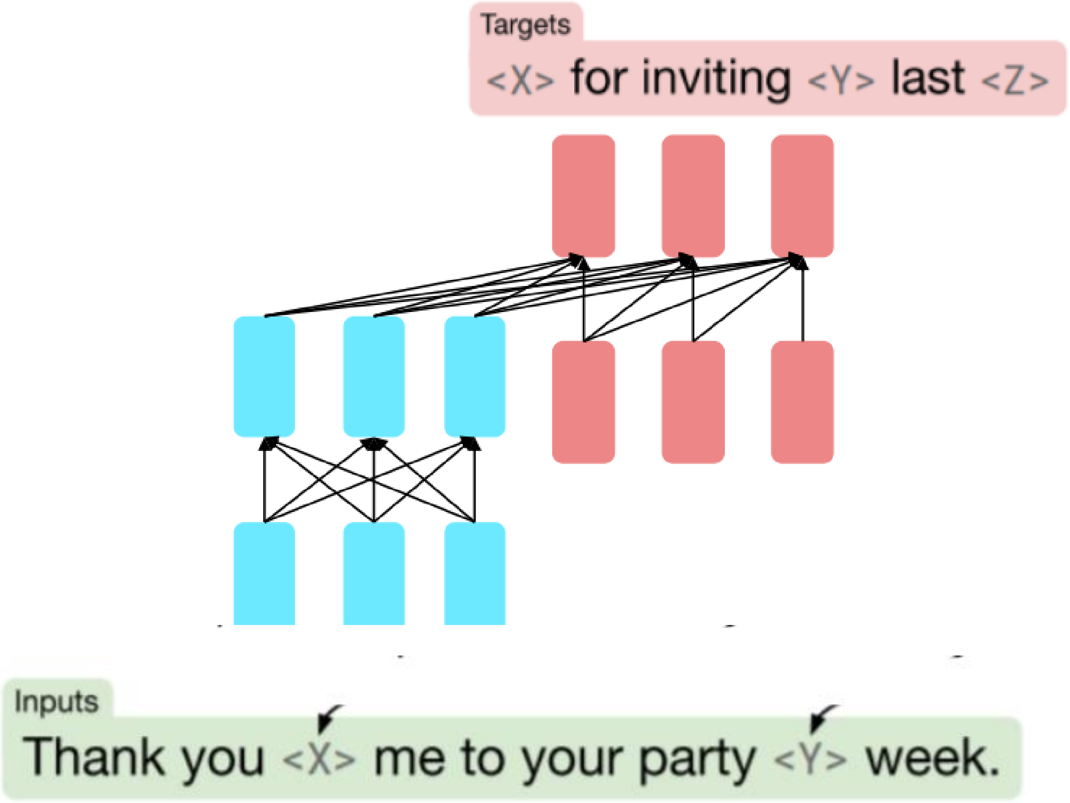
Pretraining encoder-decoders: what pretraining objective to use?

What [Raffel et al., 2018] 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!



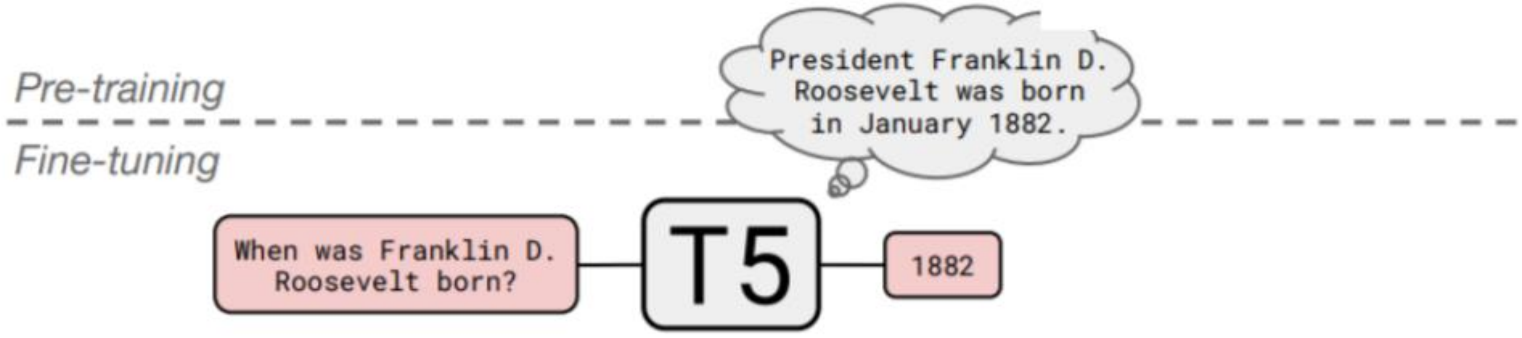
This is implemented in text preprocessing: it's still an objective that looks like **language modeling** at the decoder side.



[Raffel et al., 2018]

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: Natural Questions
WQ: WebQuestions
TQA: Trivia QA

All “open-domain” versions

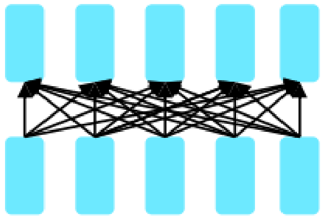
	NQ	WQ	TQA dev	TQA test	
Karpukhin et al. (2020)	**41.5**	42.4	**57.9**	–	
T5.1.1-Base	25.7	28.2	24.2	30.6	**220 million params**
T5.1.1-Large	27.3	29.5	28.5	37.2	**770 million params**
T5.1.1-XL	29.5	32.4	36.0	45.1	**3 billion params**
T5.1.1-XXL	32.8	35.6	42.9	52.5	**11 billion params**
T5.1.1-XXL +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

[Raffel et al., 2018]

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

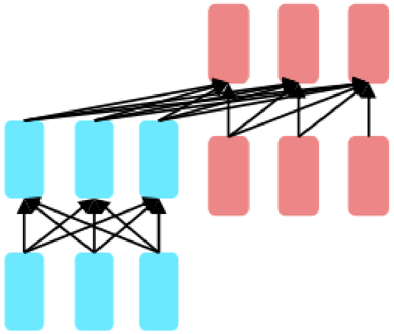
Pretraining for three types of architectures

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



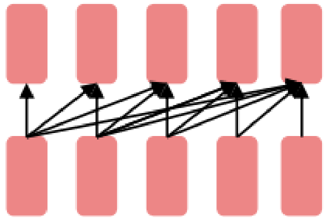
Encoders

- 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

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.

Back to the language model (next word predict)

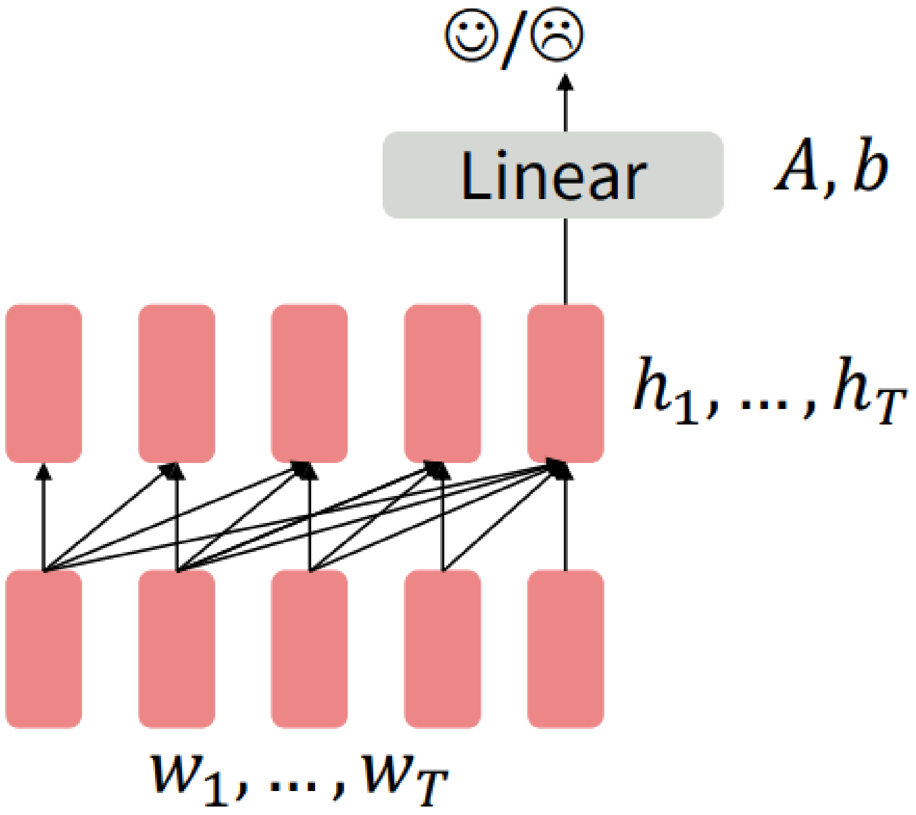
Pretraining decoders

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.

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

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.]

Pretraining decoders

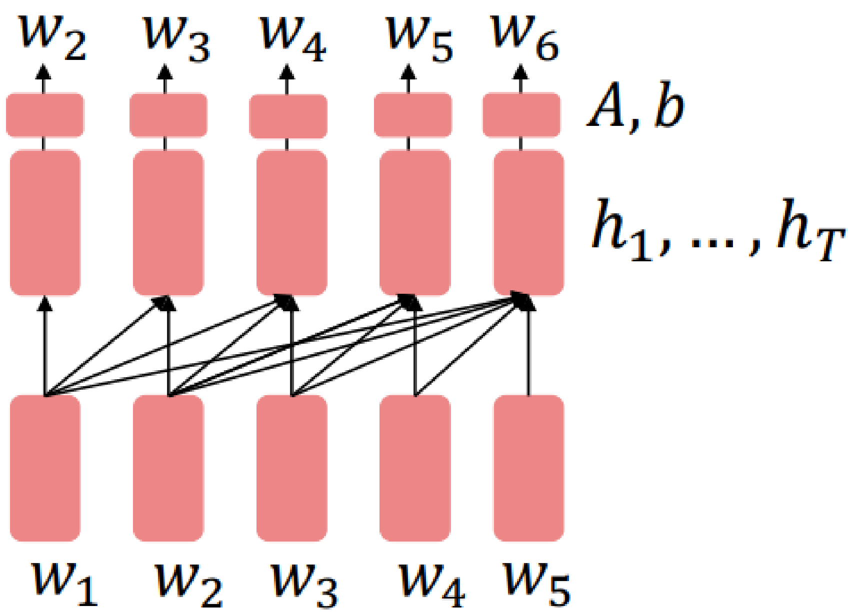
It's natural to pretrain decoders as language models and then use them as generators, finetuning their $p_\theta(w_t \mid w_{1:t-1})$

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.]

Increasingly convincing generations (GPT2) [Radford et al., 2018]

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.

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

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.**

Today's lecture

- **Language model in a narrow sense**
  (Probability theory, N-gram language model)

- Language model in broad sense

- **More thoughts on language model**

- LM (next word predict) is scalable
- LM does not need annotations
- LM is simple such that it is easily to adapt it many tasks
- LM could model human thoughts
- LM is efficient to capture knowledge (imagine use images to record knowledge?)
- Humans do LM everyday (do next-word/ next-second prediction)

Five-minute Tutorial

Python library

We provide a Python library, which you can install as follows:

```
$ pip install openai
```

https://platform.openai.com/docs/libraries/python-library main.py +

```python
import os
import openai

# Load your API key from an environment variable or secret management service
OPENAI_API_KEY = "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"

openai.api_key = os.getenv(OPENAI_API_KEY)

chat_completion = openai.ChatCompletion.create(model="gpt-3.5-turbo", messages=[{"role": "user", "content": "Hello world"}])
```

https://platform.openai.com/docs/libraries/python-library

Prompt Engineering

Related resource:

- ❖ https://www.promptingguide.ai/zh
- ❖ https://www.youtube.com/watch?v=dOxUroR57xs&ab_channel=ElvisSaravia
- ❖ https://github.com/dair-ai/Prompt-Engineering-Guide

Take some time!

- Use ChatGPT API by yourself.

Assignment 1: Using ChatGPT API

This will be released in the **next week**!
See updates in our BB system, WeChat and Emails.

Acknowledgement

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