CS183 Review ~~Linguistic~~

L2. Tasks, Data, Evaluation

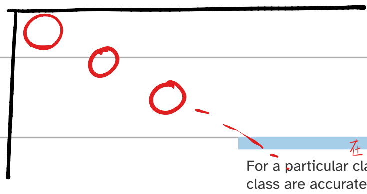
- Tasks and Evaluation Metric

- Task: Classification : input text data, output label from a closed set

E.g. Part of Speech Tagging (词性标注)

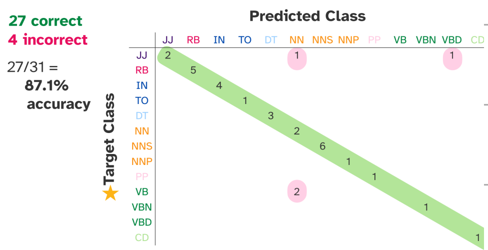
For this task, metric can be? Prediction class.

Def: Confusion Matrix : Target class



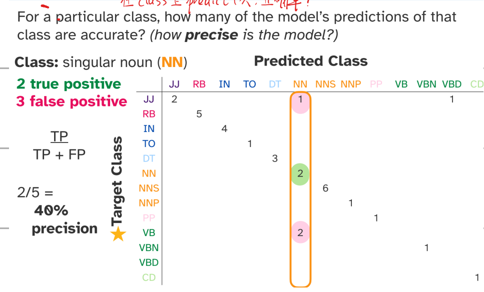
Diagonal: Correct!

⇒ Accuracy:

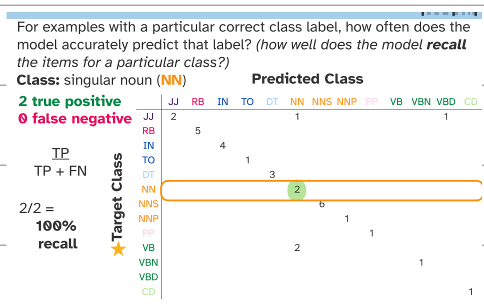


⇒ Precision:

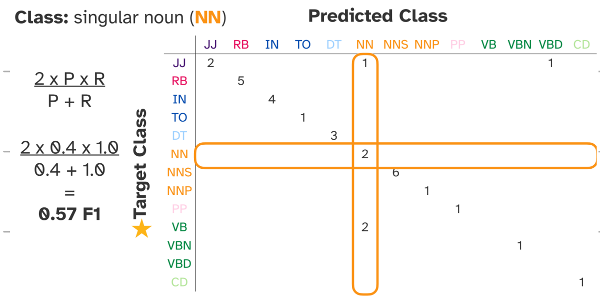
在class上predict 1次,正确率?



⇒ Recall:



With Precision & Recall ⇒ F_1 Score:



$$F_1 = \frac{2 \cdot P \cdot R}{P + R}$$

- Task: Automatic Speech Recognition: utterance ⇒ ~~text~~

Metric: Word Error Rate (WER)

Possible Error: Substitution (S), Deletion (D) & Insertion (I)

WER = (S+D+I)/N. N is the # of words in GT script

Key: Calculate Correct 'S+D+I' ⇒ Levenshtein Distance

Levenshtein Distance:

Hypothesis (i)

Reference (j)	0.	1. Take	2. me	3. to	4. you	5. see	6. campus
0.	0	1	2				
1. Take	1	0	min(2,1,3)				
2. me							
3. to							
4. UC							
5. campus							

"Grow" the edit matrix iteratively, by accumulating the cost for each element d [i,j]

DP!

$\text{local}(i,j) = \begin{cases} 1, & \text{if } H[i] \neq R[j] \\ 0, & \text{else} \end{cases}$

d [i, j] = min (d [i-1, j] + 1 , # Insertion
d [i , j -1] + 1 , # Deletion
d [i -1 , j -1] + local_substitution (i , j)) # Substitution

Hypothesis (i)

Reference (j)	0.	1. Take	2. me	3. to	4. you	5. see	6. campus
0.	0	1	2	3	4	5	6
1. Take	1	0	1	2	3	4	5
2. me	2	1	0	1	2	3	4
3. to	3	2	1	0	1	2	3
4. UC	4	3	1	1	1	2	2
5. campus	5	4	2	2	2	2	2

substitution insertion

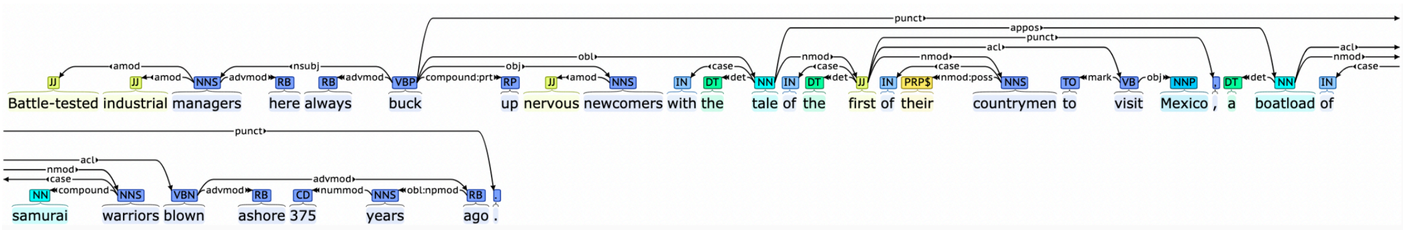
Optimal path: tells us what the errors were

∵ you ≠ UC

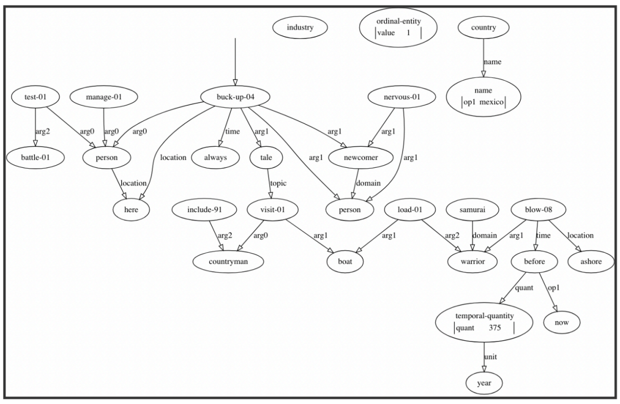
∴ local(i,j) = 1

· Task: Structured Prediction: text data → structured combination of labels from a closed set

E.g. Syntactic Parsing (Dependency).



E.g. Semantic Parcing (语义分析)
(Abstract Meaning Representation)



Metric: For them: Task-specific

E.g. Semantic Parsing: Text → SQL: match accuracy
execution accuracy

· Task: Text Comprehension : Text → Short piece of text

Information Extraction , Question Answering ...

· Task: Reference Based Text Generation : like machine translation

Metric: ① BLEU !

- System outputs / candidates $\hat{y}$

  $\hat{y}_1$ *The most common form of language use is conversation.*

  $\hat{y}_2$ *The most common form of language is speech.*

$G_n(x)$: n-gram

(里面无重复元素)

- $G_n(x)$ is the set of *n*-grams in sequence x

  $G_1(y) = \{$*the, most, natural, form, of, language, use, is, dialogue, .*$\}$

  $G_2(y) = \{$*the most, most natural, natural form, form of, ...*$\}$

$C(s,x)$

- $C(s,x)$ is the number of occurrences of *n*-gram s in x

  $C($*the*$, y) = 1$ $\quad C($*most*$, y) = 1$ $\quad C($*natural form*$, y) = 1$

n-gram precision

How many of the *n*-grams actually occur in the reference?

$$P_n(\hat{y}, y) = \frac{\sum_{s \in G_n(\hat{y})} \min(C(s,\hat{y}), C(s,y))}{\sum_{s \in G_n(\hat{y})} C(s,\hat{y})}$$

y *The most natural form of language use is dialogue.*

$\hat{y}_1$ *The most common form of language use is conversation.*

分母：所有 n-gram 元组，有重复，的数量

在GT中出现几次

分子：对于每一个 n 元组，min (C(s, $\hat{y}$), C(s, y))

Calculation Example

注意取出的 Gn(x) 来自于 prediction !

n-gram precision

How many of the *n*-grams actually occur in the reference?

$$P_n(\hat{y}, y) = \frac{\sum_{s \in G_n(\hat{y})} \min(C(s,\hat{y}), C(s,y))}{\sum_{s \in G_n(\hat{y})} C(s,\hat{y})}$$

y *The most natural form of language use is dialogue.*

$\hat{y}_1$ *The most common form of language use is conversation.*

$G_1(\hat{y}_1) = \{$*the, most, common, form, of, language, use, is, conversation, .*$\}$

s	$C(s,\hat{y})$	$C(s,y)$	$\min(C(s,\hat{y},C(s,y))$
the	1	1	1
most	1	1	1
common	1	0	0
form	1	1	1
of	1	1	1
language	1	1	1
use	1	1	1
is	1	1	1
conversation	1	0	0
.	1	1	1
Sum	**10**		**8**

$P_1(\hat{y}_1, y) = 8 / 10 = 0.8$

To prevent rather short sentences for BLEU score 'stealing'

⇒ Brevity Penalty

$$BP(\hat{y},y)=\begin{cases}1, & |\hat{y}|\geq|y| \\ e^{1-\frac{|y|}{|\hat{y}|}}, & \text{else}\end{cases}$$

n-gram precision

How many of the n-grams actually occur in the reference?

$$P_n(\hat{y},y)=\frac{\sum_{s\in G_n(\hat{y})}\min(C(s,\hat{y}),C(s,y))}{\sum_{s\in G_n(\hat{y})}C(s,\hat{y})}$$

y The most natural form of language use is dialogue.

$\hat{y}_3$ language

$$\mathrm{BP}(\hat{y},y)=\begin{cases}1 & |\hat{y}|\geq|y| \\ e^{(1-(|y|/|\hat{y}|))} & |\hat{y}|<|y|\end{cases}$$

In practice: Average n-gram precision, for up to $N = 4$

Final BLEU formula:

$$BP(\hat{y},y)\exp\left(\sum_{n=1}^{N}\frac{1}{N}\ln P_n(\hat{y},y)\right)$$

Average over the log results

$$P_n(\hat{y},y)=\frac{\sum_{s\in G_n(\hat{y})}\min(C(s,\hat{y}),C(s,y))}{\sum_{s\in G_n(\hat{y})}C(s,\hat{y})}$$

$$\mathrm{BP}(\hat{y},y)=\begin{cases}1 & |\hat{y}|\geq|y| \\ e^{(1-(|y|/|\hat{y}|))} & |\hat{y}|<|y|\end{cases}$$

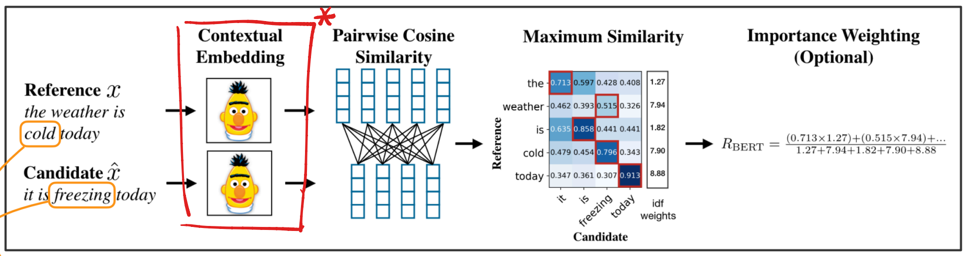
$$\textsc{Bleu}=\mathrm{BP}(\hat{y},y)\exp\left(\sum_{n=1}^{N}\frac{1}{N}\ln P_n(\hat{y},y)\right)$$

② Representation Similarity: Synonyms should work or suffice, but this can't be reflected on BLEU

Need Semantic or Contextual Representation

⇒ Embedding

Then calculate similarity -----



No n-gram overlap, but should still get some credit!

BERTScore, Zhang et al. 2020

③ Human Eval: Especially for long-text generation.

☆: There've been study showing that automatic metric can reflect human judgemen!

④ LLM-as-a-Judge: Use LLM to score or judge

· Task: Communicative Success: System take action via language

E.g. Input Image & Object, Output referring expression

Easy to eval since they take actions toward given object!

· Task: Dialogue & Interactive Systems

Hard to eval! ⇐ { Define slot filling task, but reduce expressivity
Use another model as simulater rather than human user }

- Building Language Technology — A paradigm:

Task — Implementation (Training included) — Eval (with Data!)

Tasks (in General)

Most tasks can be thought of as a mapping from some input X to some output Y, where one or both of these have to do with language

Task	Input (X)	Output (Y)
Text classification	Text	Label from a fixed class
Automatic speech recognition	Audio signal	Text
Dependency parsing	Text	Dependency tree
Code generation	Text	Executable code
Question answering	Document and question (both text)	Answer (text)
Translation	Text (in source language)	Text (in target language)
Open-ended NLG	Optional prompt (text)	Text
Referring expression generation	Image and target object	Referring expression (text)
Dialogue	Conversation history (text)	Next utterance (text)

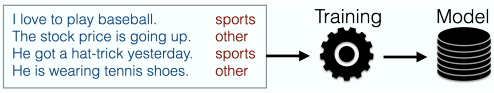
How can we implement a classification model?

- Manual rules

```python
def classify(x: str) -> str:
    sports_keywords = ["baseball", "soccer", "football", "tennis"]
    if any(keyword in x for keyword in sports_keywords):
        return "sports"
    else:
        return "other"
```

- Prompting a model without training

- Machine learning

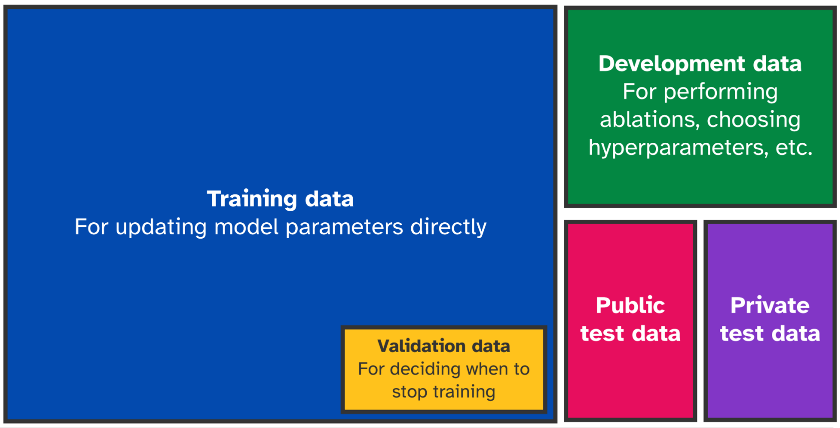


- How can we estimate performance on new documents?
  - Simulate it using held-out data just for evaluation!

(预留数据)

Data

Before pretrained models, nearly all datasets came with splits, assumed to be IID:



- Benchmark and Model Eval

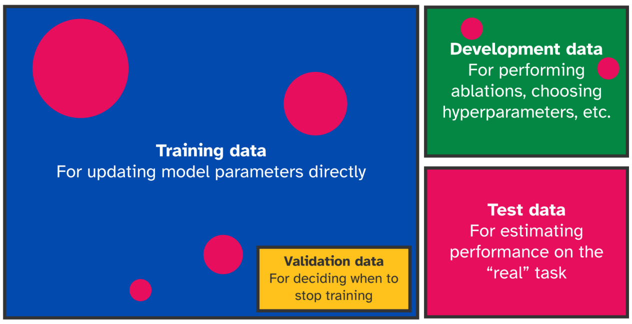
↓

Really important:

- Estimating how well our models will work on real-world data
- Shared understanding of model performance with standardized evaluations
- Building trust within a community in proving how well a new model does
- Driving progress towards specific tasks and capabilities

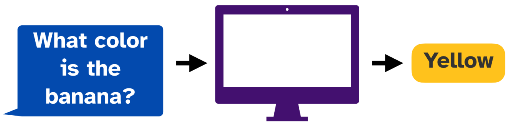
There can be pit-fall as well !

Dataset contamination occurs when actual test data is used in any of the previous splits (no longer IID). **Why is this bad?**



Model 'memorize' Test Data !

Does the benchmark actually evaluate what we want it to?



Correct 80% of the time without looking at an image!

伪相关

- Learned a spurious correlation that gives the model high accuracy: bananas are yellow
- Doesn't actually test visual understanding

Benchmark fail to eval how well model 'learn'

Defining "Human Performance"

Is the task actually evaluable?

- Is there an objectively correct label?

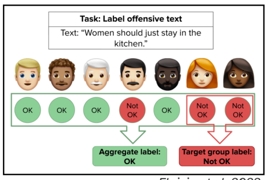
  What color should we make our Berkeley t-shirts?

- Do annotators disagree? If so, how are we processing their disagreements?

20 annotators

"Gold" label: Blue

Human performance: 12/20 = 60%



Fleisig et al. 2023

Human Eval Bias !

The Long-Tail Paradox

- Which of the following words is most rare? *myriad, solipsist, anachronistic, apricate*
- Can you think of a rarer word than these? (Check in the Google Books Ngram Viewer)
- LLMs can look really impressive when we are trying to challenge them with tricks, but maybe this is because we're bad at coming up with long-tail (rare) challenging tasks out of the blue!
- Can seem even more impressive when the task is unverifiable, and any model response could satisfy you!

We focus on few popular tasks, but ignore long-tail tasks

And for long-tail task, we are easy to be content

L3: Speech & Lexical Semantics (词汇语义)

Linguistic Units : Smallest : Phonemes (音位) & Graphemes (字位)

语音中区分意义最小声音单位　　书写系统中代表一个音位最小单位

Speech can be represented as acoustic waveform (声波)

continuous time series　　hard to analysis compute

⇒Signal Processing can be helpful

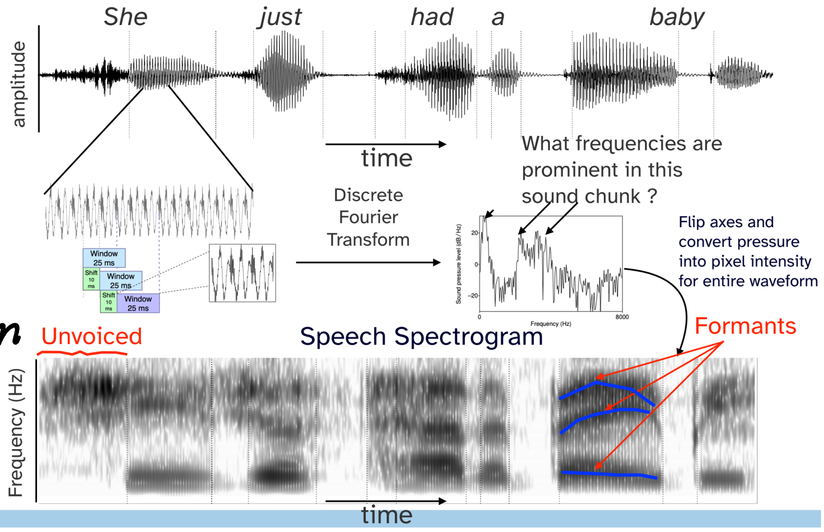
① Analyze short-time window

② Run Fourier Transform

Spectrogram

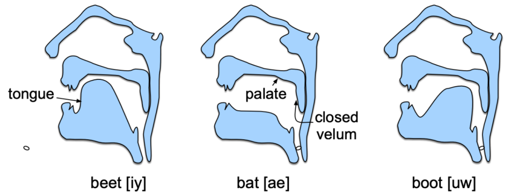
而又有发音对应的 Spectrogram

即可进行 'decode'

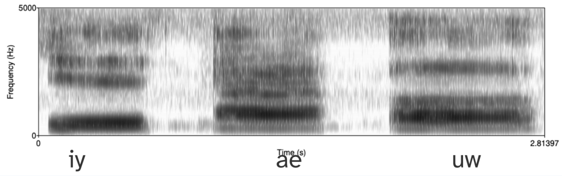


语音学: phonetics

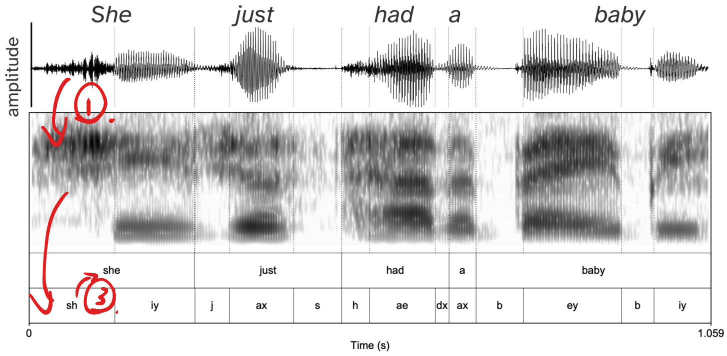
Study of speech sounds — their physical production, spectral and perceptual properties



Articulatory Phonetics



Acoustic Phonetics



- Spectrogram reveals some segmental structure with distinct properties
- These are phonemes — perceptually distinct speech sounds

International Phonetic Alphabet (IPA)

- Phoneticians compiled a common set of sounds used to codify different speech sounds (across languages)

THE INTERNATIONAL PHONETIC ALPHABET (r

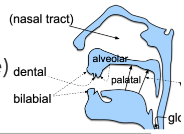
CONSONANTS (PULMONIC)

	Bilabial	Labiodental	Dental	Alveolar	Postalveolar	Retroflex	Palatal	V
Plosive	p b		t d			ʈ ɖ	c ɟ	k
Nasal	m	ɱ	n			ɳ	ɲ	
Trill	ʙ		r					
Tap or Flap		ⱱ	ɾ			ɽ		
Fricative	ɸ β	f v	θ ð	s z	ʃ ʒ	ʂ ʐ	ç ʝ	x

IPA：一音一符，一符一音

发音

- Consonants are characterized by place and manner of articulation
  - /p/ is caused by constriction at lips (labial)
  - /p/ is caused by sudden release of air (plosive)



辅音，元音

Place of articulation

- Vowels are characterized by jaw position and tongue shape

以之进行区分!!

- Some vowels also use lips (eg. sound *uw* in *cool*)

音位学

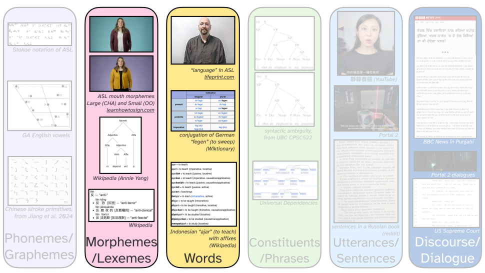
Phonology is the study of rules that govern the organization of sounds in a language (Phonemes → Syllables → Words)

音位→音节→单词

发音字典 (CMU)

- Pronunciation dictionaries (often made by linguists) give the syllables and phonemes within each word in vocabulary

Conclusion:



- Words are composed of atomic units based on sound (for spoken languages)
- Sounds are a function of how we move our vocal tracts and mouth anatomy
- Languages have distinct sets of possible sounds (phonemic inventory)
- And "rules" governing which sound sequences are likely (syllable structure)

接下来介绍 Morphemes / Lexemes (语素 / 词位 (字典词)).

- Text data can be viewed as a sequence of words
- First step in building a language technology: building a function that maps from arbitrary text data to that sequence

```
['The', 'most', 'natural', 'and', 'basic', 'form', 'of', 'language',
'use', 'is', 'dialogue', ':', 'Every', 'language', 'user', ',',
'including', 'young', 'children', 'and', 'illiterate', 'adults', ',',
'can', 'hold', 'a', 'conversation', ',', 'whereas', 'reading', ',',
'writing', ',', 'preparing', 'speeches', 'and', 'even', 'listening',
'to', 'speeches', 'are', 'far', 'from', 'universal', 'skills', '.']
```

tokenized text

- Type-token distinction:

(word type).

  - Type: a unique word in a text corpus
  - Token: an *instance* of a word type, appearing in a particular context

Example:

```
{'.', ',', ':', 'Every', 'The', 'a', 'adults', 'and', 'are', 'basic',
'can', 'children', 'conversation', 'dialogue', 'even', 'far', 'form',
'from', 'hold', 'illiterate', 'including', 'is', 'language', 'listening',
'most', 'natural', 'of', 'preparing', 'reading', 'skills', 'speeches',
'to', 'universal', 'use', 'user', 'whereas', 'writing', 'young'}
```

wordtypes (vocabulary)

instances (tokens) of wordtype ','

```
['The', 'most', 'natural', 'and', 'basic', 'form', 'of', 'language',
'use', 'is', 'dialogue', ':', 'Every', 'language', 'user', ',',
'including', 'young', 'children', 'and', 'illiterate', 'adults', ',',
'can', 'hold', 'a', 'conversation', ',', 'whereas', 'reading', ',',
'writing', ',', 'preparing', 'speeches', 'and', 'even', 'listening',
'to', 'speeches', 'are', 'far', 'from', 'universal', 'skills, '.']
```

tokenized text

- Simplest tokenizer (for English): splitting on spaces

```
tokenized = s.split(' ')
```

```
['The', 'most', 'natural', 'and', 'basic', 'form', 'of', 'language',
'use', 'is', 'dialogue:', 'Every', 'language', 'user,', 'including',
'young', 'children', 'and', 'illiterate', 'adults,', 'can', 'hold', 'a',
'conversation,', 'whereas', 'reading,', 'writing,', 'preparing',
'speeches', 'and', 'even', 'listening', 'to', 'speeches', 'are', 'far',
'from', 'universal', 'skills.']
```

定义了token, how to tokenize?

Simplest: whitespace!

punctuations mixed, maybe need rules for that

- But this gets us some weird wordtypes:

```
'dialogue:'
'user,'
'skills.'
```

Not really words different from *dialogue*, *user*, *skills*

- nltk tokenizers, with special rules for punctuation

```
import nltk
tokenized = nltk.word_tokenize(s)
```

```
['The', 'most', 'natural', 'and', 'basic', 'form', 'of', 'language',
'use', 'is', 'dialogue', ':', 'Every', 'language', 'user', ',',
'including', 'young', 'children', 'and', 'illiterate', 'adults', ',',
'can', 'hold', 'a', 'conversation', ',', 'whereas', 'reading', ',',
'writing', ',', 'preparing', 'speeches', 'and', 'even', 'listening',
'to', 'speeches', 'are', 'far', 'from', 'universal', 'skills', '.']
```

- But this still loses similarity between wordtypes

```
'skills'
'reading'
'speeches'
```

Lexically similar to, but morphologically distinct from *skill, read, speech*

- Once you've "trained" your tokenizer, you're stuck with it

```
vocab = ['.', ',', ':', 'Every', ... , 'writing', 'young']
vocab.index('ChatGPT') → not found!
```

- Once you've "trained" your tokenizer, you're stuck with it

```
vocab = ['.', ',', ':', 'Every', ... , 'writing', 'young', '<UNK>']
tokenized_indices =
    [vocabulary.index(token) for token in tokenized_text
     if token in vocabulary
     else vocabulary.index('<UNK>')]
```

add a special token for unknown words

Not Robust enough!

也可考虑: Character Unit → Encode, using Unicode:

- Strings are sequences of characters (bytes)!

```
tokenized = s.encode()
```

```
54 68 65 20 6d 6f 73 74 20 6e 61 74 75 72 61 6c 20 61 6e 64 20 62 61 73 69 63 20 66 6f 72 6d 20 6f 66 20
6c 61 6e 67 75 61 67 65 20 75 73 65 20 69 73 20 64 69 61 6c 6f 67 75 65 3a 20 45 76 65 72 79 20 6c 61 6e
67 75 61 67 65 20 75 73 65 72 2c 20 69 6e 63 6c 75 64 69 6e 67 20 79 6f 75 6e 67 20 63 68 69 6c 64 72 65
6e 20 61 6e 64 20 69 6c 6c 69 74 65 72 61 74 65 20 61 64 75 6c 74 73 2c 20 63 61 6e 20 68 6f 6c 64 20 61
20 63 6f 6e 76 65 72 73 61 74 69 6f 6e 2c 20 77 68 65 72 65 61 73 20 72 65 61 64 69 6e 67 2c 20 77 72 69
74 69 6e 67 2c 20 70 72 65 70 61 72 69 6e 67 20 73 70 65 65 63 68 65 73 20 61 6e 64 20 65 76 65 6e 20 6c
69 73 74 65 6e 69 6e 67 20 74 6f 20 73 70 65 65 63 68 65 73 20 61 72 65 20 66 61 72 20 66 72 6f 6d 20 75
6e 69 76 65 72 73 61 6c 20 73 6b 69 6c 6c 73 2e
```

- Now our vocabulary is a fixed size (all possible Unicode characters)

But: ⇓

- But: individual characters are not meaningful

```
['.', ',', ':', 'Every', ... , 'user', 'whereas', 'writing', 'young']
```

vs.

```
['.', ',', ':', 'A', 'B', 'C', ... , 'a', 'b', 'c', ... , '7', '8', '9']
```

- But: input sequences are much longer

```
'language'
```

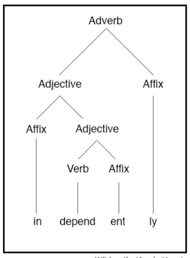
vs.

```
'l', 'a', 'n', 'g', 'u', 'a', 'g', 'e'
```

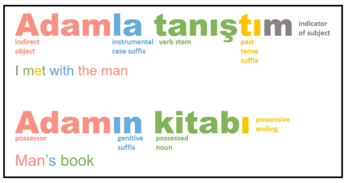
Compromise: Character ⟷ word
subparts

How to set the subparts vocab? ⇒ BPE ⇓

- Main principle: words are (often) composed of subparts (morphemes)
- Our vocabulary should have entries for frequent words kept whole, because we have a lot of data about those words
- But it should also have entries for parts of less-frequent words, so our ML models can learn how to compose parts of words into whole words (especially unfamiliar words!)



Wikipedia (Annie Yang)



- Modern standard for building a tokenizer
- Inputs: collection of texts and target vocabulary size
- Initial vocabulary is the set of all bytes (characters) across the texts
- Until the target vocabulary size is reached, repeat the following:
  - Tokenize all of the texts using the current vocabulary
  - Find the most common bigram in the tokenized texts, then add it to the vocabulary as a new wordtype *

每个单词 letter level 拆开(最先如此), 看2-gram freq, 找freq max, 进vocab。之后按 updated vocab tokenize, 找2-gram, 看freq ⋯ —
直至vocab中token数量降至目标以下

* Begin: Every letter is a token △: 'i' ⇒ 2 tokens

Documents + frequencies: ('hug', 10), ('pug', 5), ('pun', 12), ('bun', 4), ('hugs', 5)

vocabulary → *tokenized texts*

('h', 'u', 'g', 'p', 'n', 'b', 's') → ('h' 'u' 'g', 10), ('p' 'u' 'g', 5), ('p' 'u' 'n', 12), ('b' 'u' 'n', 4), ('h' 'u' 'g' 's', 5)

bigrams + frequencies

'h' 'u'	15
'u' 'g'	20
'p' 'u'	17
'u' 'n'	16
'b' 'u'	4
'g' 's'	5

most frequent bigram; add to vocabulary

Documents + frequencies: ('hug', 10), ('pug', 5), ('pun', 12), ('bun', 4), ('hugs', 5)

vocabulary → *tokenized texts*

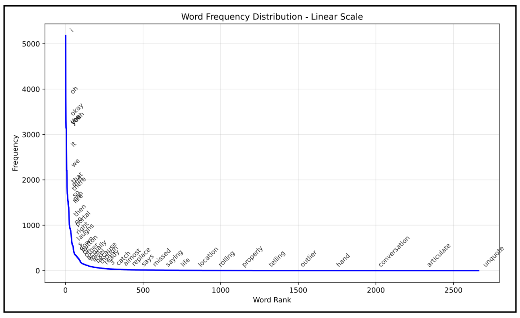
('h', 'u', 'g', 'p', 'n', 'b', 's') → ('h' 'u' 'g', 10), ('p' 'u' 'g', 5), ('p' 'u' 'n', 12), ('b' 'u' 'n', 4), ('h' 'u' 'g' 's', 5)

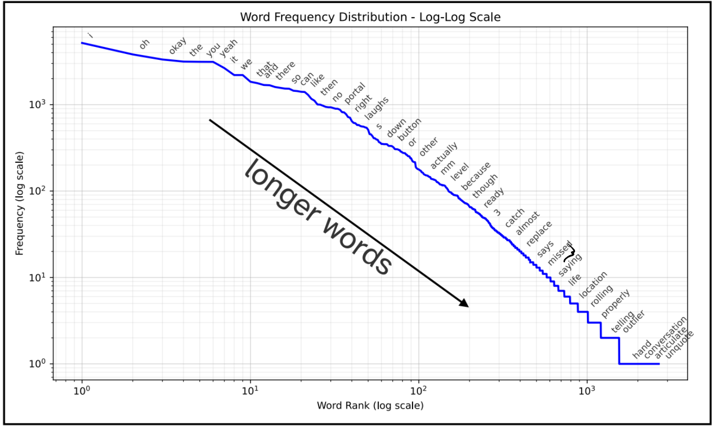
('h', 'u', 'g', 'p', 'n', 'b', 's', 'ug') → ('h' 'ug', 10), ('p' 'ug', 5), ('p' 'u' 'n', 12), ('b' 'u' 'n', 4), ('h' 'ug' 's', 5)

('h', 'u', 'g', 'p', 'n', 'b', 's', 'ug', 'un') → ('h' 'ug', 10), ('p' 'ug', 5), ('p' 'un', 12), ('b' 'un', 4), ('h' 'ug' 's', 5)

('h', 'u', 'g', 'p', 'n', 'b', 's', 'ug', 'un', 'hug') → ('hug', 10), ('p' 'ug', 5), ('p' 'un', 12), ('b' 'un', 4), ('hug' 's', 5)

Long-tail ("Zipfian") distribution of wordtypes
(from Portal 2 dialogues)





词汇freq基本规律：Zipfian Dist:

只有少数词汇会被frequently使用，绝大多数用得非常少

Word Meaning:

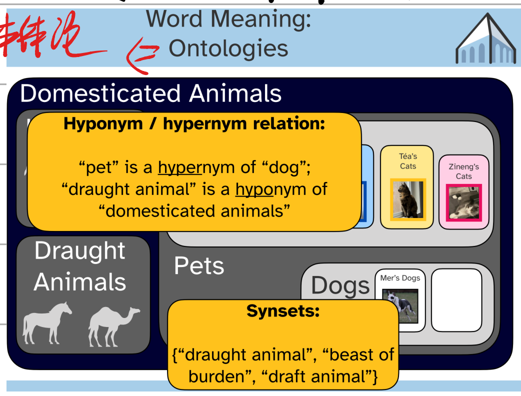
Core question: if a machine learning system is processing text, how should words be represented as input to and within the ML system?

(指称语义学)

Denotational semantics: the symbol *refers to* something in the context in which the language is used

一个词或符号的意义，就是它在现实世界或特定语境中所指代的那个具体事物、概念或实体

本体论



Hypernym：上义词 ⇓ 更广泛

Hyponym：下义词 ⇓ 更狭窄

Synset：同义词集

缺点

- Missing context-dependence (polysemy)
- Missing meaning of new words
- Requires human labor
- Subjective and culturally dependent

- Machine learning models expect numerical inputs
- Represent each word as a unique **one-hot vector** (vector with values 0, except for one value of 1) 独热编码

$\phi(\text{cat}) = [\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ ]$

$\phi(\text{kitten}) = [\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ ]$

- Vector dimension: the same size as the vocabulary
- Problem: no notion of similarity

How can I make my house safe for a new kitten?

How can I make my house safe for a new cat?

- (Is there some way we could use WordNet to get more informative numerical representations of words?)

Meaning 从 Context 中而来

- Instead: represent words as continuous vectors

$\phi(\text{cat}) = [\ -0.6\ \ 1.3\ \ -0.1\ \ 0.3\ ]$

$\phi(\text{kitten}) = [\ -0.7\ \ -1.5\ \ 0.0\ \ 0.3\ ]$

- Implicitly provides notions of similarity:

$$||\phi(\text{cat}), \phi(\text{kitten})|| < ||\phi(\text{cat}), \phi(\text{dog})||$$

- Similarity is learnable from text at scale:

... feeding time in multiple cat households can often be ...
... is to relieve the cat from the stress of ...
... effect on the feral cat population in rural areas ...

... So I wrapped the kitten up in a towel ...
... they noticed that the kitten population was getting high ...
... were looking for a kitten , hoping that someone ...

Core principle: **distributional hypothesis**

- Words that are used in similar contexts have similar meanings
- Context: typically, other words in a text, but really anything can be context!

- Input: corpus D including pairs (w, c) where w is a word and c is a context, e.g.:
  - *"Everybody likes tesgüino"*
    (w = "tesgüino", c = "likes")
    (w = "tesgüino", c = "everybody")

e.g., context is every word in the sentence

- We want to model: given that we observe word w, what's likely in the context c?

$$p(c \mid w; \theta)$$

- Our objective: find parameters that maximize the corpus probability

$$\arg\max_{\theta} \prod_{(w,c)\in\mathcal{D}} p(c \mid w; \theta)$$

- (There are other methods for word2vec (e.g., GloVe, CBOW), but we will only look at Skip-Gram in this class)

⇖.

Objective: find

$$\arg\max_{\theta} \prod_{(w,c)\in\mathcal{D}} p(c \mid w; \theta)$$

where

$$p(c \mid w; \theta) = \frac{\exp(\text{score}(c, w))}{\sum_{c'\in\mathcal{C}} \exp(\text{score}(c', w))}$$

score: similarity of representations of word and context

we use cosine similarity because, in contrast to dot product, it is unbiased towards vector magnitude: $\text{score}(c, w) = \cos(\phi(c), \phi(w)) = \dfrac{\phi(c)\cdot\phi(w)}{||\phi(c)||\,||\phi(w)||}$

⇑:

Training Paradigm: maximize prob of (c,w) pair, and prob is evaluated via

Objective: find

$$\arg\max_{\theta} \prod_{(w,c)\in\mathcal{D}} \frac{\exp(\text{score}(c, w))}{\sum_{c'\in\mathcal{C}} \exp(\text{score}(c', w))}$$

Problem: intractable to sum over all contexts

But 分母 is intractable! Vocab (C) is huge! So, instead of 'multiclass classification' (Don't need prob dist of w over the entire vocab) Negative Sampling

we do binary classification ⇒ Given (c',w) where they don't form a context and (c,w) from D, tell each sample form context or not.

- Solution: negative sampling
  - Instead of summing over all possibly contexts in which a word could appear
  - Approximate via learning from contexts in which a word *doesn't* appear
- Model whether or not a word-context pair is likely to exist in the dataset

$$p((w,c)\in\mathcal{D}) = \frac{1}{1+\exp(-\text{score}(c, w))}$$

similarity of representations of word and context

⇑.

NEW objective: find

$$\arg\max_{\theta} \prod_{(w,c)\in\mathcal{D}} p((w,c)\in\mathcal{D}) \prod_{(w,c)\in\mathcal{D}'} p((w,c)\notin\mathcal{D})$$

What is D'?

where

$$p((w,c)\in\mathcal{D}) = \frac{1}{1+\exp(-\text{score}(c, w))}$$

$$p((w,c)\notin\mathcal{D}) = 1 - p((w,c)\in\mathcal{D})$$

Binary Classification Prob Representation: Sigmoid

NEW objective: find

$$\arg\max_{\theta} \prod_{(w,c)\in\mathcal{D}} p((w,c)\in\mathcal{D}) \prod_{(w,c)\in\mathcal{D}'} p((w,c)\notin\mathcal{D})$$

⇐ How (w,c) ∈ D' is built

- Negative samples: sample and train on $l * |D|$ pairs (w', c') where

$w' \sim p(W)$ unigram prior over words

$c' \sim p(C)$ unigram prior over contexts

In large corpus, they are almost not from context.

- How to train?
  - Gradient descent
  - In practice, we work with log probabilities, not direct probabilities, to avoid float underflow

$$a : b :: c : ? \equiv d = \arg\max_{i\in\mathcal{W}} \frac{(\phi(b)-\phi(a)+\phi(c))\cdot\phi(i)}{||(\phi(b)-\phi(a)+\phi(c))\cdot\phi(i)||}$$

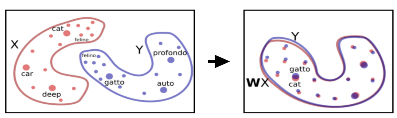
cosine similarity of — the expected representation of the analogy word — and some other word i

With embedding, we can even: ① vector arithmetic ⇑ (A之于B, 如C之于?)

② Translation: Key Assumption:

Two Language Embedding is the different mapping of the same semantic space

- What if we have embeddings in different languages and want to align them? Given:
  - ϕ^A : language A's embedding function
  - ϕ^B : language B's embedding function
  - $\mathcal{D} = \{w_i^A, w_i^B\}_{i=1}^M$: dataset pairing *M* word translations
- Find a matrix *W* such that $\min_W \sum_{i=1}^{M} ||W\phi(w_i^A) - \phi(w_i^B)||^2$



But still has challenge: like Polysemy (一词多义)

- If we have representations for parts of sentences (wordtypes), then can we get a representation of the whole sentence?
- One option: bag-of-words representation

$$\phi(\overline{x}) = \frac{1}{|\overline{x}|}\sum_{i=0}^{|\overline{x}|}\phi(x_i)$$

⇑

Try to represent a sentence, but miss a lot of things. (Syntax, Semantic ...)

Word Embedding Today ⇒

Word Embeddings in the Age of LLMs

- Building a language technology pre-LLMs, for a target task:
  - Download some pre-trained word embeddings from the Internet
  - Initialize your language model with these word embeddings (all other parameters randomized)
  - Use your task-specific data to fine-tune the other parameters (possibly also fine-tuning the word embeddings) ☆
  - Having a good starting point via word embeddings is **really** important, especially with little task-specific data
- But we don't do this anymore
  - Language models still learn embeddings specific to each wordtype ← Inside
  - But we don't download them from the Internet — we download the whole language model ⇐ Download as a whole

L4. Syntax

Grammaticality:

- Grammaticality judgments are not universal!
  - No such thing as "bad grammar" — disagreements over what feels grammatical or not are due to language variation
  - Grammatical sentences don't need to be meaningful

- Our main task: we have some language L, and we want to know whether a new sentence $x \in L$
  - Should we represent L as a finite-sized set of possible sentences? Apparently no!
  - What about a regular expression? ↓ 正则表达式

Some sentences sound grammatical, some not.

Can regular expression work?

- Can't handle center-embedding
- Lack of Memory

⇒ No!

⇒ Introducing: Context-Free Grammar

The cat that thinks the cow thinks the rabbits hid are wrong. ✗

The rabbits hid.
DT N VB

Regex:
DT N VB

The cow thinks the rabbits hid.
DT N VB DT N VB

(DT N VB)*

The cat that thinks the cow thinks the rabbits hid is wrong.
DT N IN VB DT N VB DT N VB VB JJ

DT N IN (DT N VB)* VB JJ?

Is there another formalism that tells us whether a string is "accepted" in a language or not?

Context-Free Grammars

- Set of nonterminal symbols 非终结符集合 ⇐ 语法抽象概念或中间变量
- Set of terminal symbols (wordtypes) 终结符 ⇐ 词汇，不能再分解
- Set of production rules defining how nonterminal symbols could be expressed via the composition of other nonterminal and terminal symbols

☆ 定义如何用 nonterminal & terminal to re-write a non-terminal

△. 'Can be re-written as ---'

Nonterminal symbols

DT N VB JJ IN

S NP VP SBAR

Terminal symbols (vocabulary)

are, cat, cow, hid, is, rabbits, that, the, thinks, wrong, ...

☆ **Production rules**

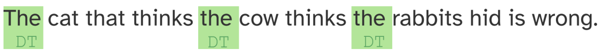
DT → {*the, a, an, ...*}
N → {*cat, cow, rabbits, dogs, ...*}
VB → {*hid, is, thinks, was, are, ...*}
JJ → {*wrong, right, blue, red, ...*}
IN → {*that, in, of, because, ...*}
NP → {DT N, NP IN VP}
VP → {VB, VB JJ, VB S}
S → {NP VP, VP}

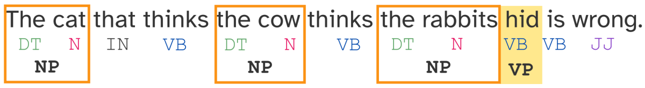
How to use production rule? Arrow:

Right → Left : Parsing / Reduction

Left → Right : Generation

E.g.:





Nonterminal symbols

DT N VB JJ IN

S NP VP SBAR

Terminal symbols (vocabulary)

are, cat, cow, hid, is, rabbits, that, the, thinks, wrong, …

Production rules

DT → {*the, a, an, …*}
N → {*cat, cow, rabbits, dogs, …*}
VB → {*hid, is, thinks, was, are, …*}
JJ → {*wrong, right, blue, red, …*}
IN → {*that, in, of, because, …*}
NP → {DT N, NP IN VP}
VP → {VB, VB JJ, VB S}
S → {NP VP, VP}

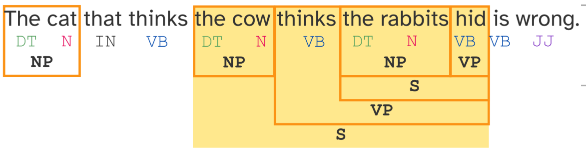
Nonterminal symbols

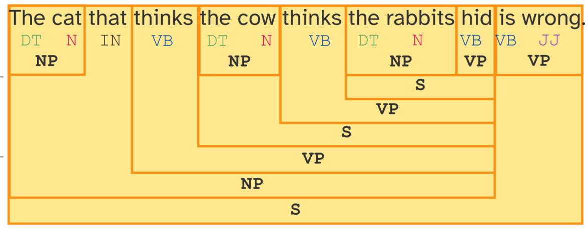
DT N VB JJ IN

S NP VP

Production rules

NP → {DT N, NP IN VP}
VP → {VB, VB JJ, VB S}
S → {NP VP, VP}





NP is singular, so its corresponding VP should be too

Nonterminal symbols

DT N VB JJ IN

S NP VP

Production rules

NP → {DT N, NP IN VP}
VP → {VB, VB JJ, VB S}
S → {NP VP, VP}

Nonterminal symbols

DT N VB JJ IN

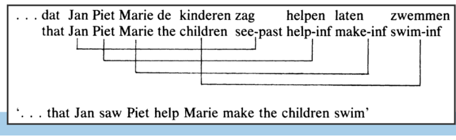
S NP VP

Production rules

NP → {DT N, NP IN VP}
VP → {VB, VB JJ, VB S}
S → {NP VP, VP}

每个rule describe了一个constituent.

- CFG is often called phrase structure or constituency grammar
- Each production rule describes a **constituent**
- Constituent constructions are independent of one another (this is why the grammar is context-free)
- Augmenting a CFG with agreement (e.g., distinguishing plural vs. singular NPs and plural vs. singular VPs plural) means it is no longer context-free
- Also, some languages aren't even context-free, not even considering agreement:



Kaplan et al.

若CFG中加入了'agreement', 如 NP&NPs, VP&VPs, 则不再 context-free (dependent of context)

Probabilistic CFG

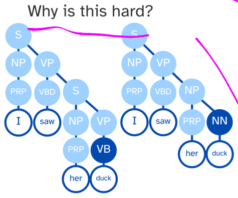
Production rules augmented with probabilities

DT → {$^{0.5}$*the*, $^{0.2}$*a*, $^{0.2}$*an*, …}
N → {$^{0.02}$*cat*, $^{0.01}$*cow*, $^{0.01}$*rabbits*, $^{0.02}$*dogs*, …}
VB → {$^{0.01}$*hid*, $^{0.5}$*is*, $^{0.05}$*thinks*, $^{0.3}$*was*, $^{0.2}$*are*, …}
JJ → {$^{0.05}$*wrong*, $^{0.1}$*right*, $^{0.05}$*blue*, $^{0.05}$*red*, …}
IN → {$^{0.4}$*that*, $^{0.3}$*in*, $^{0.2}$*of*, $^{0.05}$*because*, …}
NP → {$^{0.8}$DT N, $^{0.1}$NP IN VP, …}
VP → {$^{0.3}$VB, $^{0.2}$VB JJ, $^{0.1}$VB S, …}
S → {$^{0.7}$NP VP, $^{0.05}$VP, …}

$p(\text{rule}|\text{nonterminal})$

learn from data

syntactic parsing: given a PCFG, which rule applications generated our sentence? Why is this hard?



CFG中Rule规定了可能的rule, 而PCFG对可能"进行了量化 → 一棵树概率 = 构成此树的所有产生式规则的概率之乘积

歧义语法分析

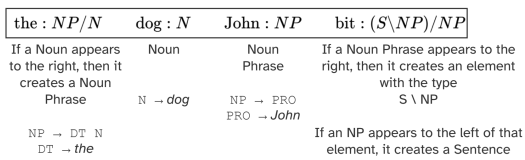
Introduce: CCG:

Syntactic type: 定义词如何与其他词依据词性进行组合

"/":需要一个参数; (S\NP)/NP: 左右均需NP, 最终组成S
&"\"

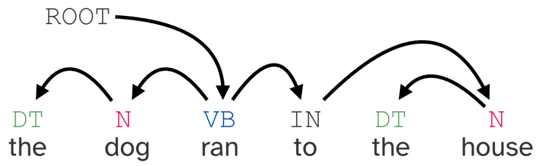
Combinatory Categorial Grammar (CCG)

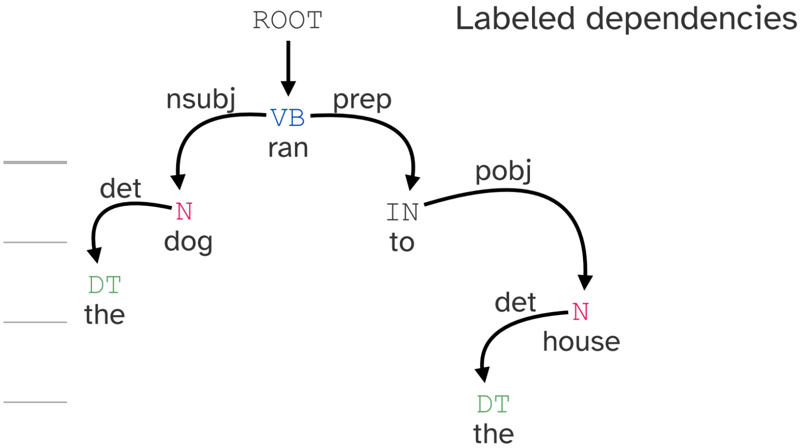
- Another way of representing a constituency grammar: bottom-up 从它出发
- Elements of a CCG:
  - Lexical items (wordtypes) (每个词)
  - Each paired with a syntactic type (≈ nonterminal or composition thereof)



Introduce: Dependency Grammar

- A dependency grammar includes:
  - Terminal symbols (wordtypes) 约束(规则).
  - Parts of speech ⇐ 词性
  - Some constraints on which parts of speech can be attached to other parts of speech





Why Syntax is Important? ①模型学习句法结构能力有限 ②是低数据场景下很好的 Inductive Bias

Lec5. Compositional Semantics

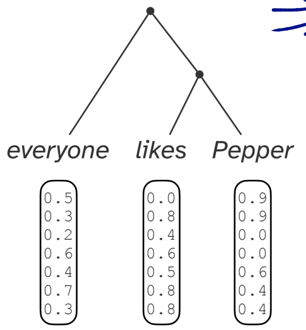
Lexical semantics: we can get word meanings how?

- Denotational semantics: tokens are references to things in the real world ⇐ 指称语义学
- Ontologies: tokens are references to nodes in some knowledge graph ⇐ 本体论
- Word embeddings: tokens are represented by continuous vectors ⇐ 词嵌入

*: 句法作用: 哪些词序组合可能 or 不可能, 并为合法组合建立潜在结构

- **Lexical semantics:** we can get word meanings
  - Denotational semantics: tokens are references to things in the real world
  - Ontologies: tokens are references to nodes in some knowledge graph
  - Word embeddings: tokens are represented by continuous vectors
- *
  Syntax: we can determine what sequences of word types are possible or not possible in a language by modeling latent structure
  - ☆ Constituency grammar aka phrase structure grammar aka context-free grammar
  - Dependency grammar (Introduced)

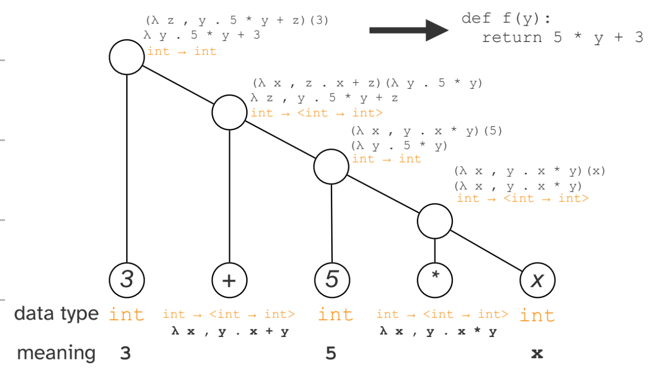
Main challenge of semantic parsing: how do we get a single representation of the entire sentence's meaning from (a) the meanings of its words, and (b) their order and latent structure?



⇒ With word meaning & Syntax, how to semantic parsing?

Analogy:

CCG and Lambda Calculus



- CCG ⇒ 组合结构

Lambda: "+" ⇒ λx.λy.x+y

- Truth-Conditional Semantics

☆ In the context of their use, statements are either true or false, i.e., they have the type *t* (aka, bool in python)

- Let's call this context a world *w*
- We'd like the outcome of our semantic parsing to be a some that can evaluate to true or false (i.e., $\mathcal{W} \to [0,1]$)

Goal: Sentence $\xrightarrow{\text{Semantic Parsing}}$ Function. f

This f: Input a world w, output whether this sentence is true or not in w.

w_1 w_2

the cat is on the table $\xrightarrow{\text{CCG}}$ some function that can be evaluated to give us the denotation in an arbitrary world ⇐ ☆ CCG, a bridge.

everyone	likes	Pepper
S / (S \ NP)	(S \ NP) / NP	NP
< e → t > → t	e → < e → t >	e
λ f . ∀ x (person(x) → f(x))	λ x, y . likes (y, x)	Pepper

likes Pepper ⟶ S \ NP ; e → t ; λ y . likes (y, Pepper)

	everyone	likes	Pepper
Syntactic type	S / (S \ NP)	(S \ NP) / NP	NP
Semantic type	<e → t> → t	e → <e → t>	e
λ-expression	λ f . ∀ x (person(x) → f(x))	λ x, y . likes (y, x)	Pepper

Example: CCG.

Pepper 语义上为实体

λ-exp ⇒ function

everyone: S / (S \ NP) ; < e → t > → t ; λ f . ∀ x (person(x) → f(x))

likes: (S \ NP) / NP ; e → < e → t > ; λ x, y . likes (y, x)

Pepper: NP ; e ; Pepper

⟶ S \ NP ; e → t ; λ y . likes (y, Pepper)

⟶ S ; t

λ f . ∀ x (person(x) → f(x))[(λ y . likes (y, Pepper))]

∀ x (person(x) → (λ y . likes (y, Pepper)) (x)

∀ x (person(x) → likes (x, Pepper))

Final Result. ⇐

Formal Semantics

- Logical operators, like ∨, ∧, and ¬
  *Pepper is clever **and** curious*
- Quantifiers like ∀ and ∃
  ***Some** cats like water*
- Relationships between functions ⇒ and ⇔
  Squares are rectangles (∀x(square(x) ⇒ rectangle(x)))
- Verbs can have tenses, and can be modified with adverbs
- We can talk about beliefs others have
- Some combinations of meanings are nonsensical (unevaluable)
  green ideas
- Sentences aren't just statements — sometimes they are commands, questions, etc.
- Sentences exist in the context of previous sentences and their meanings

Modern: ⇓

- In NLP, nobody is really mapping from sentences to lambda calculus representations anymore
- However, many of our problems still take the form of mapping from language to some meaningful structured representation

Lec5: Dialogue, Conversation & Interaction

- Pragmatic: 语用学，研究语言在特定情景(context)中如何被使用，理解和产生意义

三个核心概念: Speech Act, Presupposition & Implication
言语行为 预设 隐含

What color is Pepper?

⬇

$\lambda f.(\text{Color}(f) \wedge f(\text{Pepper}))$

Semantics: mapping from surface form (sequence of tokens) to formal executable representation

$[\![\lambda f.(\text{Color}(f) \wedge f(\text{Pepper}))]\!]^i$

⬇

{black}

Pragmatics: executing the logical form against some context to acquire its denotation

Speech Acts

- Our interpretation of utterances used in context often goes beyond the literal (formal) meaning
- By interpreting speech as action, we can ascribe intent to utterances that isn't obvious from their formal representation ⇒ 当我们将言语诠释为行动时，我们能够推断潜在意图

Do you mind if I sit next to you?

Yeah (go ahead)

No (I don't mind)

Sorry, someone is coming

Presupposition

- Propositions that must be true about a world in order to compute the denotation of a particular sentence
- In other words: implicit assumptions made by utterances

Pepper owns a house — Can be true or false

Pepper's house is big — Computing the truth value requires that, in our world, there is an entity `x` such that `house(x)` and `owns(Pepper, x)`

Awareness of presupposition in speech is very useful for critical analysis of persuasive speech, e.g. in politics

一句话要成立，必依赖某些未言明的背景假设

Implicature

- Propositions suggested by an utterance, but not explicitly expressed
- Meaning is determined by reasoning about alternatives

Do you know what they weather will be like today?

You should bring your umbrella.

Context: in San Francisco

~ Heavy rain is possible

Alternatives:

- *no*
- *yes*
- *low 60s*
- *chance of precipitation is 30%*

言外之意

Principles behind Implicature: Gricean Maxims

- General principles we believe we mutually hold about how what kinds of utterances we should add to conversation given what's been said so far:
  - **Quantity:** utterances should contain just the right amount of information — not too little or too much
  - **Truth:** utterances should not contain falsehoods
  - **Relation:** utterances should be relevant to what's been said before
  - **Manner:** utterance form and meaning should be clear
- What happens when we contribute utterances that break these principles?

Principles behind Implicature: Gricean Maxims

- **Flouting** conversation maxims is "breaking" them under the assumption the listener knows the speaker is intentionally breaking them
  - E.g., flouting relevance:

Do you know what they weather will be like today?

You should bring your umbrella.

- **Violating** maxims is breaking them under the assumption the listener *won't* believe a maxim has been broken

My dog ate my homework.

上面是四项会话原则 ⇒ 打破原则的两种方式。⇒

下面介绍会话中的共同基础：(Common Ground)

During interaction, we maintain some representation of what we believe is mutually known by conversation participants

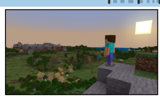
- Mutually known: I know it, I believe you know it, I believe you believe I know it, I believe you believe I believe you know it, ...
- What can be in the common ground?
  - Principles guiding how we interact with one another (e.g., Gricean maxims, shared lexicon)
  - Propositions about the world, values and beliefs
  - Things in our shared environment, including things we are paying attention to

- This allows us to reason about what is not in the common ground
  - E.g., facts, beliefs, etc. that we believe the other does not know
- By maintaining models of others' beliefs, we can reason about how they might interpret our utterances
- If we want to bridge a belief gap between ourselves and another conversation participant, we can rely on the the rules governing language use and interpretation that (we believe) are in the common ground

Cooperative Interaction:

Multi-Agent Foundations of Interaction

- You're playing Minecraft, and want to build a house
  - Your actions influence the state of the world
  - But there's also stochasticity (e.g., animals, enemies, villagers appear and might destroy your house, but their behavior is predictable)



- But what happens when another player comes to the game?
  - They also want to build a house
  - Maybe they will take some of your resources from you
  - Their behavior is not completely stochastic, though!

To best model another player, we may want to keep track of their:

- **Beliefs:** what information are they using to make decisions? How do they perceive the world, and how do they build an internal model of the world as they act in it?
- **Goals:** what do they want to get done? Do they share the same goals, are their goals orthogonal, or are they trying to sabotage mine?
- **Intentions:** how will they attempt to execute their goals? What skills do they have and what strategies are they likely to take?
- **Model of *me*:** if it's useful for me to reason about them, they are probably reasoning about me, too — how does this influence their actions?

- Let's assume my partner and I share the same (high-level) goal, and it's in the common ground
  - Goal: build a house
- It's likely optimal for me to choose actions in a way that depends on my partner's actions, to avoid redundancy and execute the goals more efficiency
  - I'll gather wood while my partner places it in the right spots to create a foundation
- Some environments might require that I and my partner take different actions at the same time
  - E.g., pressing pallet switches

- How can we more successfully coordinate with one another, especially under uncertainty over how the environment works?
- How can I better model my partner?
  - What they observe and know?
  - What they are trying to do?
  - What are their plans to act?
- How can I influence my partner?
  - By sharing information with them?
  - By telling them what to do?
  - By teaching them about how to act?

在multi-agent 环境中:

- At the beginning of an interaction, we might have significant uncertainty over other agents
  - Over their goals, beliefs, and skills
  - And also over how they use language
- But over time, we converge to more similar representations
  - By building models of one another from observing their behavior
  - By explicitly resolving uncertainties via language use
  - This refines our expectations of other agents
  - Certainty in our expectations allows us to take communicative shortcuts

Observation space now includes utterances made by other agent(s)

Action space now includes the ability to produce an utterance

← Dynamic Interaction.

The influence of learning on interaction dynamics:

- Agent policies adapt as they learn about other agents
- This influences the observations other agents make of. them
- Which in turn influences how other agents adapt

- **Challenges for learning in multi-agent environments:**
  - Agents need to learn their first-order policy (goal, observation → action)
  - But they also might need to model how their adaptations influence the behavior of other agents, including the introduction of new shared abstractions (i.e., words and conventions)!
- **Challenges for evaluating multi-agent systems:**
  - Dynamics depend heavily on initial conditions: (uncertainty over) variation across agent partners' beliefs, goals, and intentions
  - E.g., in teaching contexts, a teacher will adapt their language and pedagogical strategy to the learner's existing knowledge and skills

Element of Scenario,

- Interaction dynamics also depend heavily on the properties of the context itself:
  - Incentive structure
  - Environment design — perception and action, novelty
  - Participants — how many, any existing structures among them, roles, a priori asymmetries
  - Communication channel
- Work in computational linguistics, psycholinguistics, and cognitive science aims to characterize the relationship between scenario design and linguistic behavior

Simple Multi-Agent Scenario:

- **Environment:** set of candidate referents, available to both players
- **Players**
  - Speaker: knows the identity of a target object
  - Listener: no privileged information
- **Shared goal:** listener picks out the target object
- **Actions**:
  - Speaker: natural language
  - Listener: selection of candidate referent
- **Communication channel:** unidirectional, single utterance (no dialogue)

Conversation as a multi-agent game.

- No immediate physical environment, but we still exist in a social world, with common (or uncommon) knowledge, goals, and values
- One can work towards social goals by using language as action, e.g. through:
  - Education
  - Persuasion
  - Hate speech and dogwhistles

Lec 6: Multilingual NLP

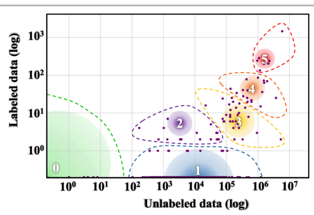
- For any task we expect out of language technologies, they should work for any language
  - Question answering, information retrieval, summarization
  - Dialogue systems and chatbots
  - Language generation
- Language technologies can also support cross-language communication
  - Machine translation
  - Language learning

Challenges:

① Data Modality

- Not all languages have writing systems, with many languages where:
  - Only audio recording is possible or available
  - Developing a consistent writing system is difficult, and finding or creating written records is extremely time-consuming relative to how much the language is used
- Some languages are rarely written or have inconsistent uses of writing systems, and only used conversationally
- Some writing systems are not-yet digitized, or all documents are handwritten

② Data Scarcity



- For most languages, very little data is available for training or evaluating language technologies
- There's even less labeled or parallel data!

1.2B total speakers, virtually no available data for building language technologies

③ Dialectical Variation

- The same languages can vary significantly depending on who is speaking it where
  - Regional differences
  - Formality differences
- What can vary? Any linguistic features!
- Speakers often mix different dialects with one another in the same conversation (code-mixing)

In some languages, syllables may be distinguished not only by which specific vowel is used, but also by:

- Syllable length
- Pitch contour (spectrogram)
- Pitch height
- Phonation (breathy, creaky, etc.)

Differences between vowels may be more subtle

⇐ ④ Speech System

⑤ Morphology ⇒

⑥ Lexical Semantics

(like how to express 'q₂' !)

- Synthetic languages denote syntactic relationships between words using inflection (modification of a word, e.g., conjugating a word) or agglutination (adding particles to a word)
- Issues with tokenization

⑦ Syntax : (E.g. English & Chinese. English & Japanese)

⑧ Semantics : Wide variety of possible language feature

⑨ Idioms & Figurative Speech ⑩ Difference in Language use

⑪ Language Change over time.

Lec6: Sequence Modeling

How to model a sentence, i.e., a token sequence?

- For now: let's assume utterances are sequences of tokens from our vocabulary
- Our vocabulary has a fixed size and consists of discrete wordtypes (we'll get to modeling continuous language signals, like speech, in a few weeks!) 词汇表大小固定，且离散
- A sequence is denoted as:

$$\overline{x} = \langle x_1, \ldots, x_n \rangle \quad x \in \mathcal{V} \quad x_n = \text{EOS}$$

⇐ Inductive Bias

- We can also consider writing out all possible sequences given our vocabulary (though this set is infinitely large): $\mathcal{V}^+$

建模目标本质：

- 一种建模方式：

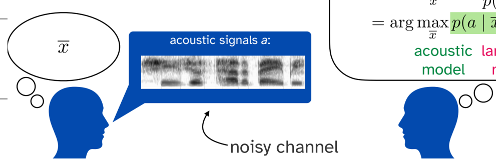
- A sequence model imposes a probability distribution over $\mathcal{V}^+$

$$p(\overline{X}) \in \Delta^{\mathcal{V}^+} \qquad \overline{x} \in \mathcal{V}^+ \qquad p(\overline{x})$$

- Why is this useful?

Bayes' rule

$$\begin{aligned} \overline{x}^* &= \arg\max_{\overline{x}} p(\overline{x} \mid a) \\ &= \arg\max_{\overline{x}} \frac{p(a \mid \overline{x})p(\overline{x})}{p(a)} \\ &= \arg\max_{\overline{x}} \underbrace{p(a \mid \overline{x})}_{\text{acoustic model}}\underbrace{p(\overline{x})}_{\text{language model}} \end{aligned}$$



Example: Count-Based Language Model

Documents + frequencies:

$$\mathcal{D} = \begin{array}{l}\texttt{('hug', 10), ('pug', 5),}\\ \texttt{('pun', 12), ('bun', 4),}\\ \texttt{('hugs', 5)}\end{array}$$

$$\mathcal{V} = \{\mathrm{b, g, h, n, p, s, u}\}$$

$$\mathcal{V}^+ = \{\mathrm{b, g, h, \ldots, bb, bg, \ldots, bug, bun, \ldots, sssssss, \ldots}\}$$

- Learning problem: we want a to estimate the probability distribution $p(\overline{X}) \in \Delta^{\mathcal{V}^+}$ that generated our observations $\mathcal{D}$
- One simple option: just count based on document occurrence

$$p(\overline{x}) = \frac{C(\overline{x})}{|\mathcal{D}|}$$

$\overline{x}$	$p(\overline{x}) = \frac{C(\overline{x})}{\lvert\mathcal{D}\rvert}$
hug	10/36 = 0.28
pug	5/36 = 0.14
pun	12/36 = 0.33
bun	4/36 = 0.11
hugs	5/36 = 0.14
$\overline{x} \in \mathcal{V}^+ \setminus \mathcal{D}$	0/36 = 0.00

$p(\overline{X})$

Count-Based ⇒ frequency

Training Data ⇒ Corpus , But ……

Language Modeling is Hard

$$p(\overline{X}) \in \Delta^{\mathcal{V}^+} \qquad \overline{x} \in \mathcal{V}^+ \qquad p(\overline{x})$$

- How might we go about assigning a probability to **any possible sequence**, even ones we've never seen before?
- One intuition: sentences have internal consistencies!

可见概率建模不能以句子为unit.

Autoregressive language modeling:

- The probability of a sequence is a product of local token probabilities
- The probability of a token depends on the ones that came before it

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1})$$

Introduce : Autoregressive Language Modeling :

Sampling Procedure

Sampling from an Autoregressive Language Model

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1})$$

$$= p(x_1)p(x_2 \mid x_1) \ldots p(x_n \mid x_1, \ldots, x_{n-1})$$

Let's sample a sequence from this approximation:

- Sample the first word $x_1 \sim p(X_1) \in \Delta^{\mathcal{V}}$
- Sample the second word $x_2 \sim p(X_2 \mid x_1) \in \Delta^{\mathcal{V}}$

$p(X_2 \mid x_1 = \text{the})$

$\mathcal{X}$	$p(x)$
-ir	0.03
same	0.02
impact	0.02
line	0.02
...	

*· 选中EOS，停止！

$\overline{x} = \langle$ *the, same*

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \cdots, x_{i-1})$$

Another Approach: Mask 少用

$$p(\bar{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \dots, x_{i-1}, x_{i+1} \cdots x_n)$$

Masked language modeling:

- The probability of a sequence is a product of local token probabilities
- The probability of a token depends on the ones that came before *and after* it

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \dots, x_{i-1}, x_{i+1} \dots x_n)$$

The name *anteater* refers to the species' [redacted], which consists mainly of ants and termites.

Key question: 什么是:

$p(x_i \mid x_1, \dots, x_{i-1})$?

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \dots, x_{i-1})$$

$$= p(x_1)\, p(x_2 \mid x_1) \dots p(x_n \mid x_1, \dots, x_{n-1})$$

probability that the first word is x_1

probability that the second word is x_2, given that the first word is x_1

probability that the sentence ends after the sequence $\langle x_1, \dots, x_{n-1} \rangle$

Core modeling challenge:

How do we compute these conditional probabilities? *.

$x_3 \sim P(X_3 \mid x_1, x_2)$, but what is it?

Lec 7: N Grams

$p(X_i = x) = p(x \mid x_1, \dots, x_{i-1})$.

How to model it? Core challenge:

i can be any number!!

one solution

⇒ Markov property assumption.

Let's make a Markov assumption:

The probability of word at index *i* only depends on the *n* - 1 words that came before it

$$\approx p(x \mid x_{i-n+1}, \dots, x_{i-1})$$

Preceding (n-1)-gram

$$\approx \frac{C(x_{i-n+1}, \dots, x_{i-1}, x)}{C(x_{i-n+1}, \dots, x_{i-1})}$$

count of n-gram

count of prev (n-1)-gram

可见与之在 n or (n-1) gram 中的 count 挂钩

n=1 (left, aka bag of word)

n=2 (right)

$$p(\overline{x}) = \prod_{i=1}^{|\overline{x}|} p(x_i \mid x_1, \dots, x_{i-1})$$

$$\approx \prod_{i-1}^{|\overline{x}|} p(x_i)$$

$$p(\text{yellow suitcase and red hat}) = p(\text{red suitcase and yellow hat})$$

Word order does not matter!
This is why it's called "bag of words"

$$p(X_i = x) \approx \frac{C(x_{i-n+1}, \dots, x_{i-1}, x)}{C(x_{i-n+1}, \dots, x_{i-1})}$$

$$= \frac{C(x_{i-1}, x_i)}{C(x_{i-1})}$$

$$p(\overline{x}) \approx \prod_{i=1}^{|\overline{x}|} \frac{C(x_{i-1}, x_i)}{C(x_{i-1})}$$

"n-gram to model conditional prob" Summary ⇒

Generalization of *n*-gram Language Model

As *n* increases, we get more fluent text

But also more sparsity: $p(X_{10} = \text{-}) = \frac{C(\text{Liberals, Third Parties, Left-})}{C(\text{Liberals, Third Parties, Left})} = \frac{417}{418}$

(in a corpus of 1.4T tokens!)

n=1	n=2	n=3	n=4	n=10
never	view	were	dropped	Liber
Comment	-find	e	her	-als
has	a	-colog	off	,
in	place	-ical	at	Third
.	of	topics	her	Part
"	a	related	achiev	-ies
t	human	T	-ement	,
view	intelligence	-weet	Canadian	Left
C	brief	niet	Resource	-
never Comment has in . "t view C	***viewfind a place of a human intelligence brief***	***were ecological topics related to Tweet niet***	***dropped her off at her achievement Canadian Resource***	***Liberals, Third Parties, Left-***

For an *n*-gram language model, we need to store counts for:

- All sequences of length *n* ($\mathcal{V}^n$)
- All sequences of length *n* - 1 ($\mathcal{V}^{n-1}$)

⇑ Storage size: exponent to n

Also, there can be corner cases:

$$p(X_i = x) \approx \frac{C(x_{i-n+1}, \dots, x_{i-1}, x)}{C(x_{i-n+1}, \dots, x_{i-1})}$$

Missing data (sparsity)

- What if the count of the target *n*-gram is 0?
  - Solution: add a small number to the count for every *n*-gram (aka "smoothing")

若 $|V|^n$ 语料库中无 <prefix, x_i>.
则加一个小数字 smooth 一下

$$p(X_i = x) \approx \frac{C(x_{i-n+1}, \dots, x_{i-1}, x)}{C(x_{i-n+1}, \dots, x_{i-1})}$$

Missing data (sparsity)

- What if the count of the target *n*-gram is 0?
  - Solution: add a small number to the count for every *n*-gram (aka "smoothing")
- What if our *n*-1-gram prefix has a count of 0? 不考虑 n-gram, 而试 n-1, n-2,
  - Solution: condition on a shorter *n*-gram prefix (e.g., the previous *n*-2, or *n*-3, etc.) instead (aka "backoff")

Shortback:

③ Without a big *n*, cannot handle long-distance dependencies

In sequence modeling, Long & short term dependency is significant

$$p(X_i = x) \approx \frac{C(x_{i-n+1}, \dots, x_{i-1}, x)}{C(x_{i-n+1}, \dots, x_{i-1})}$$

No notion of similarity

- What if the count of the target *n*-gram is 0? → smoothing
- What if our *n*-1-gram prefix has a count of 0? → backoff
- ① Storage is at worst exponential wrt $|\mathcal{V}|$
- ② Can't learn anything from the counts of *n*-grams containing similar words

$$p(\text{bike} \mid \text{I bought a}) \approx p(\text{bicycle} \mid \text{I purchased a})$$

N-Gram works, but we dream of:

'θ' can be NN!

- Let's say we have a language model that can give us a probability of any text $p_\theta(\overline{x})$
- We created this language model using a corpus $\mathcal{D} = \{\overline{x}_i\}_{i=1}^{M}$
- We care how well this generalizes to some held-out dataset $\mathcal{D}'$

- Likelihood: probability of the data under our model

M个 sentence
每个 sentence 最多 N个 token

$$\prod_{i=1}^{M} p_\theta(\overline{x}_i)$$

- Negative log likelihood (fixes float underflow)

$$-\sum_{i=1}^{M} \log p_\theta(\overline{x}_i) = -\sum_{i=1}^{M}\sum_{j=1}^{N} \log p_\theta(x_j^i \mid x_1^i, \dots, x_{j-1}^i)$$

- Perplexity: inverse probability of data, normalized by number of tokens in the dataset

$$= \exp\left(-\frac{1}{\sum_{i=1}^{M} |\overline{x}^i|} \sum_{i=1}^{M}\sum_{j=1}^{N} \log p_\theta(x_j^i \mid x_1^i, \dots, x_{j-1}^i)\right)$$

Lec 7: Generation

In this section, we explore more possible way to generate sequence in a proper and effective manner.

Sequence sampling method in Lec 6 ⇒

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \dots, x_{i-1})$$

- As we generate, we build our output sequence $\overline{x}$, which starts as an empty sequence $\overline{x} = \langle\rangle$
- At each step i, we choose an item from the vocabulary $\mathcal{V}$ by performing some **operation** on the local probability distribution $p(X_i \mid x_1, \dots, x_{i-1}) \in \Delta^{\mathcal{V}}$
- Then, we append this to our running sequence $\overline{x} \leftarrow \overline{x} + \langle x_i \rangle$
- If we ever choose EOS, we stop generation 如何利用这个条件概率决定了生成方式
- **Main point:** generation methods differ wrt the **operation** they perform on $p(X_i \mid x_1, \dots, x_{i-1})$

Operation: Temperature

Operation: modify the logits *before* computing probabilities

logits (before softmax)

$$s(w) = f(w \mid x_1, \dots, x_{i-1}; \theta)$$ parameter

$$p(X_i = w \mid x_1, \dots, x_{i-1}) = \frac{\exp(s(w))}{\sum_{w' \in \mathcal{V}} \exp(s(w'))}$$

softmax operation

Softmax-Logit - Paradigm

Sneak peek: computing probabilities over wordtypes using pretty much any modern language model

- Score each wordtype independently unsoftmaxed: logits

$$s(w) = f(w \mid x_1, \dots, x_{i-1}; \theta) \quad \leftarrow \text{logits}$$

- Renormalize using softmax

$$p(X_i = w \mid x_1, \dots, x_{i-1}) = \frac{\exp(s(w))}{\sum_{w' \in \mathcal{V}} \exp(s(w'))}$$

⟸⟹ Bring in Temperature!

Temperature parameter controls the "smoothness" of this distribution:

τ ↑ ⇒ uniform distribution.

$$p(X_i = w \mid x_1, \dots, x_{i-1}) = \frac{\exp(s(w)/\tau)}{\sum_{w' \in \mathcal{V}} \exp(s(w')/\tau)}$$

- τ = 1: no changes to the probability distribution
- τ → 0: relative probability assigned to highest-probability item in distribution increases
  - in practice, setting a temperature of 0 recovers "argmax", putting all of the mass on the highest-probability item

τ ↑ Entropy ↑

τ ↓, ↓; Sentences are more deterministic (τ=0, argmax)

- **Temperature allows us to control the entropy of the output distribution without changing its relative ranking of items**

τ ↑, entropy ↑

- Higher temperature: closer to a uniform distribution
- Lower temperature: "peakier" distribution (in the limit, gives all probability mass to the most probable item)

Suppose we have task :

Finding the Most Probable Sequence

Operation: find $\arg\max_{\overline{x}\in\mathcal{V}^+} p(\overline{x})$

Greedy doesn't guarantee global optima !

- Why is this hard?
- An approximation: greedy "sampling"

$$x_i \leftarrow \arg\max_{x\in\mathcal{V}} p(X_i \mid x_1, \ldots, x_{i-1})$$

- Just choose the most probable wordtype at each generation step (no random sampling needed)

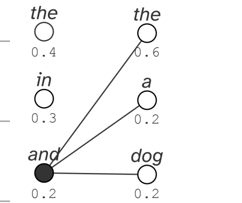
Operation: choose $x_i \leftarrow \arg\max_{x\in\mathcal{V}} p(X_i \mid x_1, \ldots, x_{i-1})$

- Why isn't this guaranteed to get us the highest-probability sequence?
- A better approximation for global argmax: **beam search**
  - During generation, we maintain a "beam" of *n* sequences instead of just one
  - At each generation step *i*,
    - We select the *n* most likely next tokens X_i for each prefix, and create *n* more sequences
    - Then we look at all the n^2 sequences so far, and discard all but the *n* most likely sequences
  - At the end, we select the sequence that has the highest probability among the set

⇒Beam Search to ease this issue !

Example :

$p(X_1)$ $p(X_2 \mid x_1 = \text{and})$



Discard all but the top 3 continuations

Current Beam Candidates

Prefix	Probability
the list	0.4*0.3
the look	0.3*0.2
the parents	0.2*0.1
in the	0.3*0.15
in a	0.3*0.1
in this	0.3*0.05
and the	0.2*0.6
and a	0.2*0.2
and dog	0.2*0.2

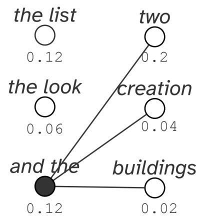
Beam

Prefix	Probability
the	0.04
in	0.02
and	0.02

Current Beam Candidates

Prefix	Probability
the list	**0.12**
the look	**0.06**
the parents	0.02
in the	0.05
in a	0.03
in this	0.02
and the	**0.12**
and a	0.04
and dog	0.04

$p(\overline{x})$ $p(X_3 \mid \overline{x} = \text{the look})$



Refresh the beam with top *n* candidates

discard

Current Beam Candidates

Prefix	Probability
the list of	0.004
the list goes	0.002
the list is	0.002
the look of	**0.018**
the lookout	**0.012**
the look on	0.006
and the two	**0.024**
and the creation	0.005
and the buildings	0.002

Beam

Prefix	Probability
the look of	0.018
the lookout	0.012
and the two	0.024

- How do we know when to stop?
  - When all of the items in the beam have EOS (we don't expand these prefixes, just keep them around for the end)
  - Or, when we've reached a maximum sequence length
- Let's say we're done sampling at this point
- We'll select the sequence with the highest probability in the beam

But : length may matter ! Item < 1 !

- What if our sequences have different lengths?

Length normalization: $p(\overline{x}) \propto \dfrac{1}{|\overline{x}|^{\alpha}} \sum_{i=1}^{|\overline{x}|} \log p(x_i \mid x_1, \ldots, x_{i-1})$

⇐ Length may matter !

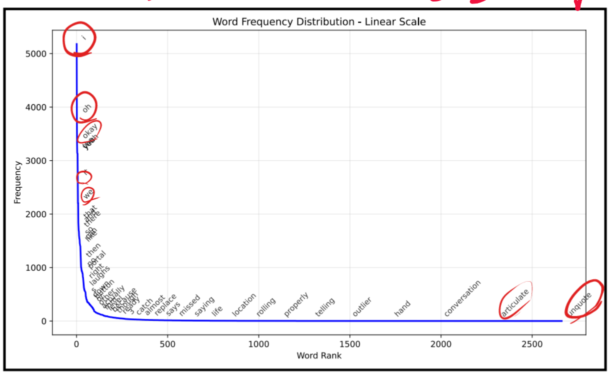
Argmax can be problematic !

Argmax produces repetitive, less diverse, and overall *too*-probable output sequences

* What's missing?

Beam Search	Human
...to provide an overview of the current state-of-the-art in the field of computer vision and machine learning, and to provide an overview of the current state-of-the-art in the field of computer vision and machine learning, and to provide an overview of the current state-of-the-art in the field of computer vision and machine learning, and to provide an overview of the current state-of-the-art in the field of computer vision and machine learning, and...	...which grant increased life span and three years warranty. The Antec HCG series consists of five models with capacities spanning from 400W to 900W. Here we should note that we have already tested the HCG-620 in a previous review and were quite satisfied With its performance. In today's review we will rigorously test the Antec HCG-520, which as its model number implies, has 520W capacity and contrary to Antec's strong beliefs in multi-rail PSUs is equipped...

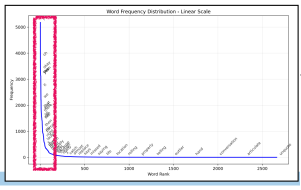
* Argmax "only" chose those highly freq words !



⇒ Change picking strategy !

Operation: top-p (nucleus) sampling

- Identify the set of *n* tokens $\mathcal{P}$ that have the highest probabilities under $p(X_i \mid x_1, \ldots, x_{i-1})$ and their cumulative probability is *p*
- Set the probabilities of all but these tokens to 0
- Renormalize by dividing remaining probabilities by $\sum_{x\in\mathcal{P}} p(X_i \mid x_1, \ldots, x_{i-1}) = p$



Operation: ϵ sampling

- Identify the set of n tokens $\mathcal{E}$ such that $\forall\, x \in \mathcal{E}$, $p(x \mid x_1, \ldots, x_{i-1}) \geq \epsilon$ 只考虑概率高于 ε (threshold) 的 token. normalize.
- Set the probabilities of all but these tokens to 0 and then sample
- Renormalize by dividing remaining probabilities by $\sum_{x \in \mathcal{E}} p(X_i \mid x_1, \ldots, x_{i-1})$

Operation: constrained decoding

- For some tasks, we have additional information about what wordtypes can or cannot be next
- E.g., in code generation, I can't generate more) than I have (对于任意输出空间上的需求，人工取出
- While modern LLMs can learn these patterns from data at scale, it can sometimes still be useful to constrain our output space $C \subseteq V$，并用 $\sum_{x \in C} p(X_i \mid x_1, \cdots, x_{i-1})$ 进行 renormalize.
- Similar to before: given a set of possible continuations $\mathcal{C} \subseteq \mathcal{V}$ we will set the probabilities of all other tokens to 0, then renormalize using $\sum_{x \in \mathcal{C}} p(X_i \mid x_1, \ldots, x_{i-1})$

Lec 8: Neural Sequence Modeling

Previously, we've mentioned the $p(X_i = x_i) = \prod_{i=1}^{|x|} p(x_i \mid x_1, \cdots, x_{i-1})$ we dream of:

$$p(\overline{x}) = \prod_{i=1}^{|\overline{x}|} p(x_i \mid x_1, \ldots, x_{i-1})$$

Autoregressive approximation ← Formulation

$$\approx \prod_{i=1}^{|\overline{x}|} p(x_i \mid x_{i-n+1}, \ldots, x_{i-1})$$

N-gram approximation ← Care about previous N-1 words

$$\approx \prod_{i=1}^{|\overline{x}|} \frac{C(x_{i-n+1}, \ldots, x_{i-1}, x_i)}{C(i_{i-n+1}, \ldots, x_{i-1})}$$

Count-based approximation ← Use n & (n-1) gram to calculate

$$\approx \prod_{i=1}^{|\overline{x}|} p(x_i \mid x_{i-n+1}, \ldots, x_i; \theta)$$

Neural n-grams ⇐ what we dream of.

$$p(x_i \mid x_{i-n+1}, \ldots, x_i; \theta) = \frac{\exp(s(x_i \mid x_{i-n+1}, \ldots, x_{i-1}; \theta))}{\sum_{x' \in \mathcal{V}} \exp(s(x' \mid x_{i-n+1}, \ldots, x_{i-1}; \theta))}$$

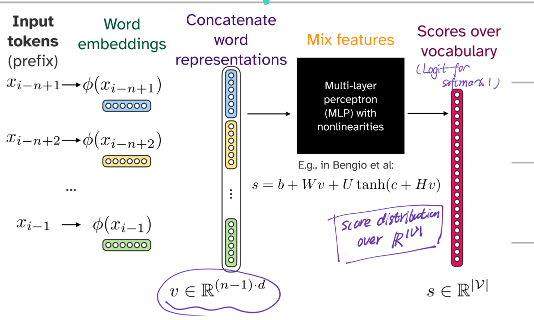
Softmax over **wordtype scores**

$$s(x \mid x_{i-n+1}, \ldots, x_{i-1}; \theta) = f_\theta(\phi(x_{i-n+1}), \ldots, \phi(x_{i-1}))[x]$$

f_θ: Neural model; takes n-1 inputs: embeddings of each prefix wordtype

Utilize 'softmax-logit paradigm' & Embedding to bring in semantic information

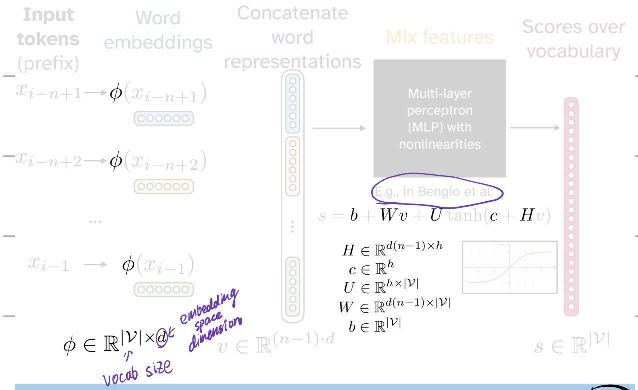
f_θ can be MLP, then out model can be:



→ softmax → sample.

$$\phi \in \mathbb{R}^{|\mathcal{V}| \times d}$$
$$H \in \mathbb{R}^{d(n-1) \times h}$$
$$c \in \mathbb{R}^{h}$$
$$U \in \mathbb{R}^{h \times |\mathcal{V}|}$$
$$W \in \mathbb{R}^{d(n-1) \times |\mathcal{V}|}$$
$$b \in \mathbb{R}^{|\mathcal{V}|}$$

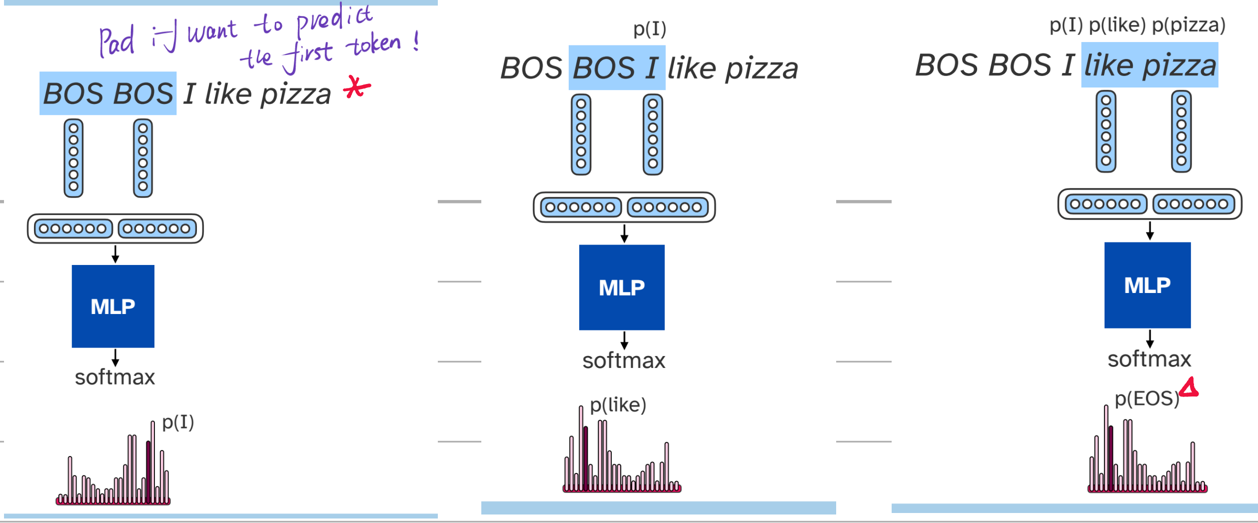
$$C_n \in \mathbb{N}^{|\mathcal{V}|^n}$$
$$C_{n-1} \in \mathbb{N}^{|\mathcal{V}|^{n-1}}$$



(learnable parameters)

Neural N-grams		**Count-based N-grams**
n = 6 |V| = 20,000 d = 100 h = 60	➔ ~13M parameters save parameter!	6.4 x 10²⁵...

Looks nice. Moreover, it even can save storage!



Example: n=3, for generating one word, look at previous (3-1) words

* Pad for generating first (3-1) words! △. STOP!

perplexity.

$$\exp\left(-\frac{1}{\cancel{3}\ 4}\left(\log p(\mathrm{I}) + \log p(\mathrm{like}) + \log p(\mathrm{pizza}) + p(\mathrm{EOS})\right)\right)$$

⇑ Generated Sentence's Perplexity

- We have some training dataset $\mathcal{D} = \{\overline{x}\}_{i=1}^{M}$
- We want to find the parameters of our neural model θ that maximize the likelihood of this dataset
- Train using gradient descent in log-probability space

$$\theta^* = \arg\min_{\theta}\left(-\sum_{\overline{x}\in\mathcal{D}}\sum_{i=1}^{|\overline{x}|}\log p(x_i \mid x_{i-n+1},\ldots,x_{i-1};\theta)\right)$$

$\phi \in \mathbb{R}^{|\mathcal{V}|\times d}$
$H \in \mathbb{R}^{d(n-1)\times h}$
$c \in \mathbb{R}^{h}$
$U \in \mathbb{R}^{h\times|\mathcal{V}|}$
$W \in \mathbb{R}^{d(n-1)\times|\mathcal{V}|}$
$b \in \mathbb{R}^{|\mathcal{V}|}$

Training Stuff:

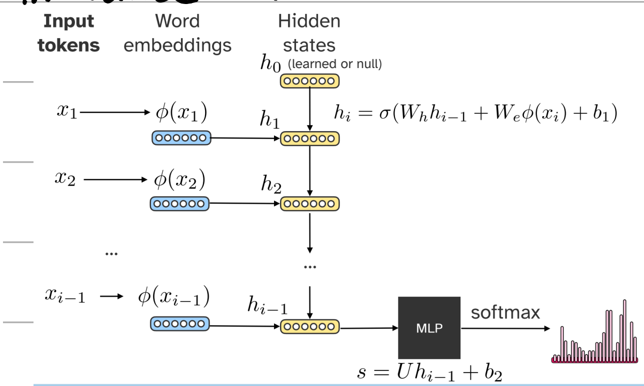
- ✓ What if the count of the target n-gram is 0?
- ✓ What if our n-1-gram prefix has a count of 0?
- ✓ Storage is at worst exponential wrt $|\mathcal{V}|$
- ✓ No notion of similarity
- ✓ Intervening words Can learn about it
- ✗ Without a big n, cannot handle long-distance dependencies
  still need n-1 token prefix !!

- N-gram models (count-based or neural) impose fixed windows of context
- Could we instead truly compute $p(x_i \mid x_1,\ldots,x_{i-1})$ for any i?

Many previous issues are solved. but n-gram's Markov Property still violates long-dist dependency.

⇒ To tackle this, introduce: Recurrent Neural Network:

- **Key idea:** add a latent space whose values are both:
  - determined by computations performed during previous "timesteps", and
  - taken in as input at each timestep
- The value of this latent space is called a "hidden state" ☆☆

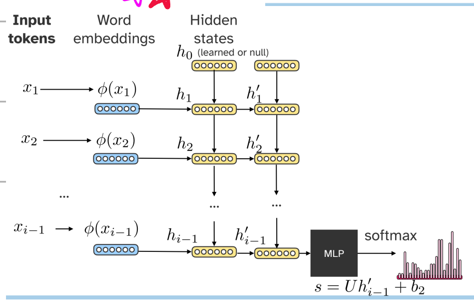


shared.

$h_i = \sigma(W_h h_{i-1} + W_e\phi(x_i) + b_1)$

σ: sigmoid

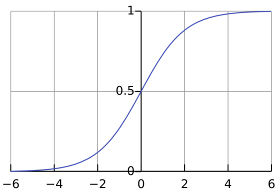
⇐ Single layer



⇐ Multi-Layer

$$\boxed{\begin{aligned} h_i &= \sigma(W_h h_{i-1} + W_e \phi(x_i) + b_1) \\ s &= U h'_{i-1} + b_2 \end{aligned}}$$

$\phi(\cdot) \in \mathbb{R}^{|\mathcal{V}| \times d}$
$W_e \in \mathbb{R}^{d \times d'}$
$b_1 \in \mathbb{R}^{d'}$
$W_h \in \mathbb{R}^{d' \times d'}$
$U \in \mathbb{R}^{d' \times |\mathcal{V}|}$
$b_2 \in \mathbb{R}^{|\mathcal{V}|}$



Neural N-grams

$\phi \in \mathbb{R}^{|\mathcal{V}| \times d}$
$H \in \mathbb{R}^{d(n-1) \times h}$
$c \in \mathbb{R}^{h}$
$U \in \mathbb{R}^{h \times |\mathcal{V}|}$
$W \in \mathbb{R}^{d(n-1) \times |\mathcal{V}|}$
$b \in \mathbb{R}^{|\mathcal{V}|}$

```
n = 6
|V| = 20,000
d = 100
h = 60
```
→ ~13M parameters

RNN

$\phi(\cdot) \in \mathbb{R}^{|\mathcal{V}| \times d}$
$W_e \in \mathbb{R}^{d \times d'}$
$b_1 \in \mathbb{R}^{d'}$
$W_h \in \mathbb{R}^{d' \times d'}$
$U \in \mathbb{R}^{d' \times |\mathcal{V}|}$
$b_2 \in \mathbb{R}^{|\mathcal{V}|}$

```
|V| = 20,000
d = 100
d' = 60
```
→ ~2.1M parameters

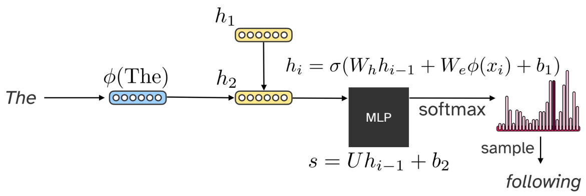
Plus, we can (theoretically) keep information from any point in the past!

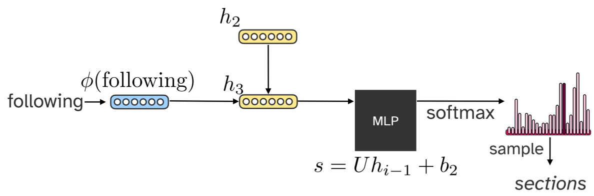
Good!

Generation

Example

(not training)





- We have some training dataset $\mathcal{D} = \{\overline{x}\}_{i=1}^{M}$
- We want to find the parameters of our neural model θ that maximize the likelihood of this dataset
- Train using gradient descent in log-probability space

$$\theta^* = \arg\min_{\theta} \left(- \sum_{\overline{x} \in \mathcal{D}} \sum_{i=1}^{|\overline{x}|} \log p(x_i \mid x_1, \ldots, x_{i-1}; \theta) \right)$$

$\phi(\cdot) \in \mathbb{R}^{|\mathcal{V}| \times d}$
$W_e \in \mathbb{R}^{d \times d'}$
$b_1 \in \mathbb{R}^{d'}$
$W_h \in \mathbb{R}^{d' \times d'}$
$U \in \mathbb{R}^{d' \times |\mathcal{V}|}$
$b_2 \in \mathbb{R}^{|\mathcal{V}|}$

这玩意儿就是loss!

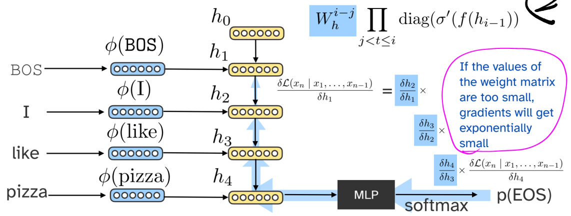
During training, we always compute the probability of each observed token by conditioning on its observed prefix. This is also called “teacher forcing”.

Vanishing Gradients

$$\frac{\delta \mathcal{L}(x_i \mid x_1, \ldots, x_{i-1})}{\delta h_j} = \frac{\delta \mathcal{L}(x_i \mid x_1, \ldots, x_{i-1})}{\delta h_i} \prod_{j<t\leq i} \mathrm{diag}\left(\sigma'(f(h_{i-1}))\right) W_h$$

$$\prod_{j<t\leq i} W_h \prod_{j<t\leq i} \mathrm{diag}(\sigma'(f(h_{i-1})))$$

$$W_h^{i-j} \prod_{j<t\leq i} \mathrm{diag}(\sigma'(f(h_{i-1})))$$



$$\mathcal{L}(x_n \mid x_1, \ldots, x_{n-1}) = -\log p(x_n \mid x_1, \ldots, x_{n-1}; \theta)$$

Problem: Vanishing Gradients

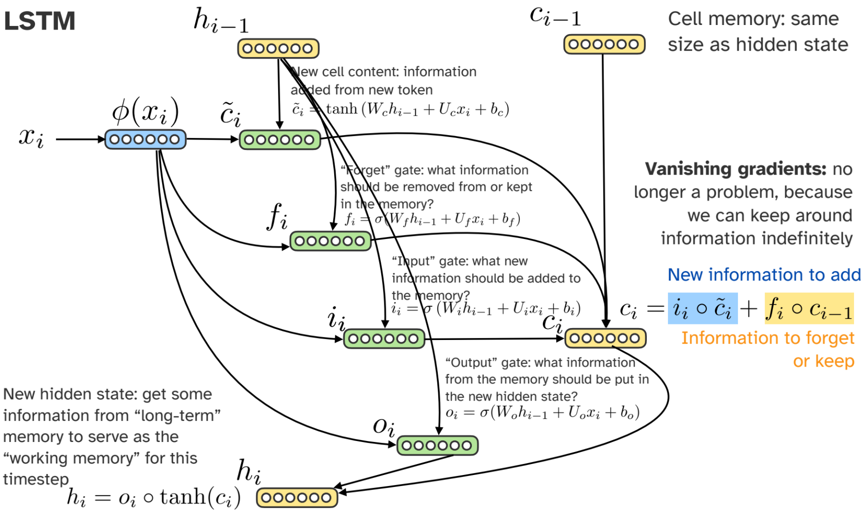
Exploding gradients: if the activations are too big, the gradients become too big, and the values of the weights become too big, eventually getting values closer to ∞

- Can solve using gradient clipping (if the norm of the gradient is greater than a threshold, clamp its value or scale it down)

- **Vanishing gradients:** gradient signal from many timesteps in the future is negligible compared to gradient signal from nearby tokens

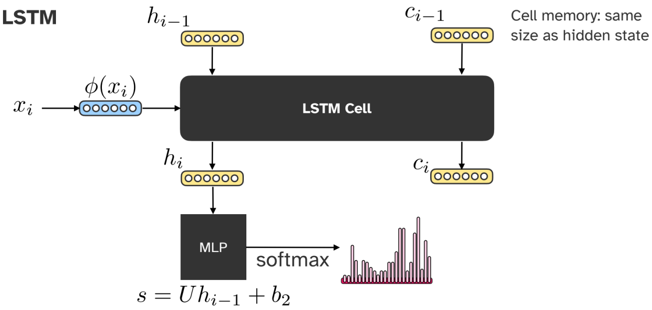
⇒ Introduce: Advanced Model:

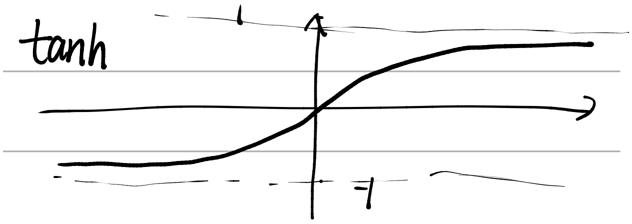
LSTM



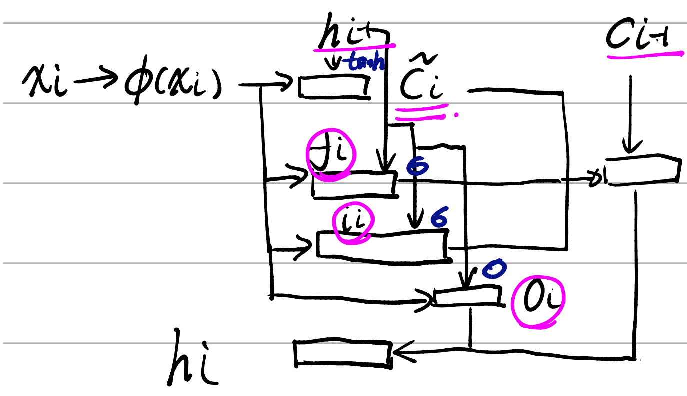
Vanishing gradients: no longer a problem, because we can keep around information indefinitely

LSTM





tanh allow negative number



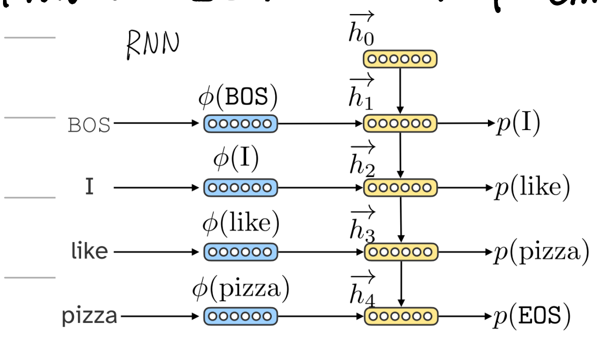
$C_i = i_i \circ \tilde{C}_i + f_i \circ C_{i-1}$

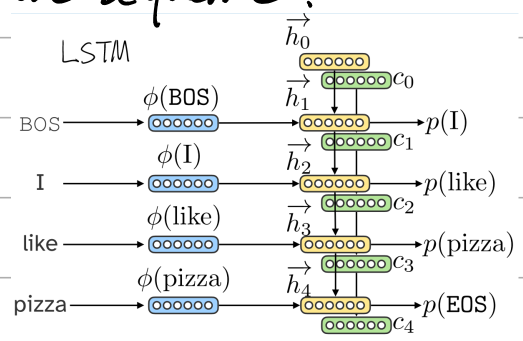
f_i i_i O_i are scalars

$h_i = O_i \circ \tanh(C_i)$

Lec 8: Sequence Embedding

We have word embedding, what about sentence? Can we use RNN or LSTM to help embed the sequence?





$$\phi(\overline{x}) = \phi(\text{I like pizza}) = \frac{1}{|\overline{x}|}\sum_{i=1}^{|\overline{x}|}\phi(x_i)$$

Option 0: average word embeddings

But we lose all notions of order

$$\phi(\overline{x}) = \phi(\text{I like pizza}) = h_{-1}$$

Option 1: take the last hidden state

But this is overly weighted by information from later tokens

$$\phi(\overline{x}) = \phi(\text{I like pizza}) = \frac{1}{|\overline{x}|}\sum_{i=1}^{|\overline{x}|} h_i$$

Option 2: average hidden states over time

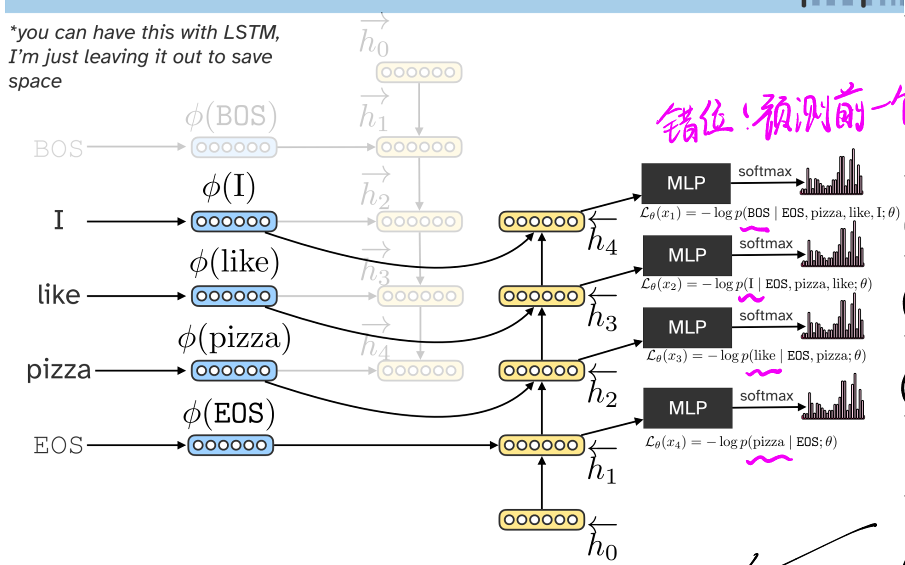
But earlier states have no information about later ones, while later ones have full information of the sequence *

*⟺ To tackle this ⟹ Bidirectional RNN

Bidirectional RNN

you can have this with LSTM, I'm just leaving it out to save space

错位：预测前一个词



$$\phi(\overline{x}) = \phi(\text{I like pizza}) = \frac{1}{|\overline{x}|}\sum_{i=1}^{|\overline{x}|}\left[\overrightarrow{h}_i; \overleftarrow{h}_i\right]$$

① 加入BOS & EOS

② 正向做完一遍，反向做一遍

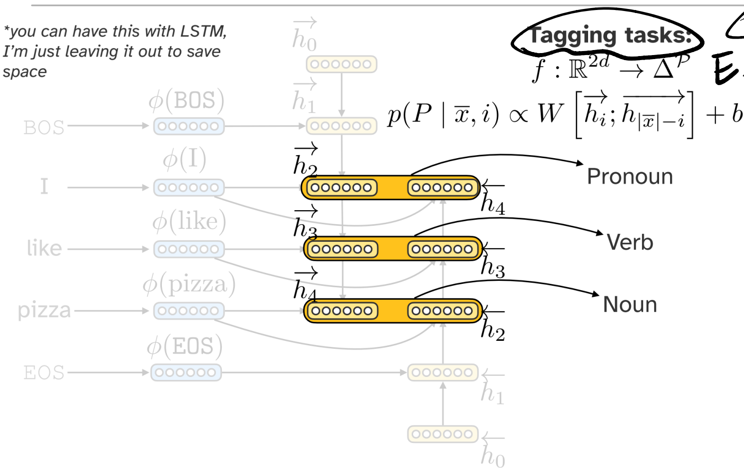
③ $\overrightarrow{h_i}$ $\overleftarrow{h_i}$ 拼接，并normalize.

接下游任务(同时利用 $\overrightarrow{h_i}$ $\overleftarrow{h_i}$)

you can have this with LSTM, I'm just leaving it out to save space

Tagging tasks: $f : \mathbb{R}^{2d} \to \Delta^{P}$ E.g.

$$p(P \mid \overline{x}, i) \propto W\left[\overrightarrow{h}_i; \overleftarrow{h}_{|\overline{x}|-i}\right] + b$$



- Let's say we have some embedding function (aka **encoder**)

$$\text{Enc} : \mathcal{V}^{+} \to \mathbb{R}^n$$

Note: $\text{Enc} \equiv \phi$ is the notation we'll use for encoding sequences

- And we want to learn to classify text
  - E.g.: spam vs. not spam
- We want to learn the optimal parameters of some function

$$f_\theta : \mathbb{R}^n \to \Delta^{\{\text{spam}, \text{not spam}\}}$$

such that for some labeled dataset $\mathcal{D} = \{\overline{x}_i, y_i\}_{i=1}^{M}$

$$\theta^* = \arg\max_\theta \prod_{i=1}^{M} f(y_i \mid \text{Enc}(\overline{x}_i); \theta)$$

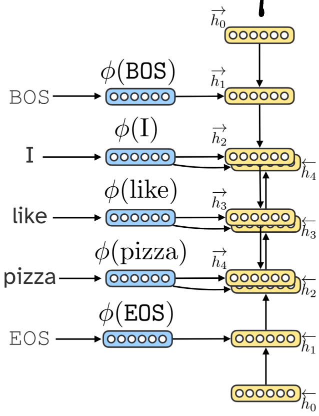
- Our classifier might just look like a linear transformation or MLP, e.g.,

$$f_\theta(\overline{x}) = \sigma(W\text{Enc}(\overline{x}) + b)$$
$$\theta = \{W, b\}$$

- And we can train this classifier by fixing the parameters of the Encoder
- Or, we could also backpropagate into the encoder itself
  $\theta = \{W, b\} \cup \theta_{\text{RNN}}$

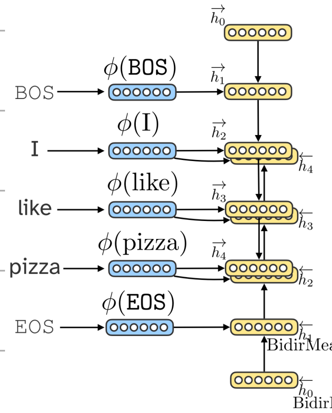
Lec 9. Sequence to Sequence Modeling

Recap:



- Maintains a latent "hidden state" updated after each new token
- Hidden states can be used to:
  - Get a distribution over the next token's value
  - Get a vector representation of the entire sequence

$$\text{Enc}(\overline{x}) = \text{Pool}\left(h_1, \ldots, h_{|\overline{x}|}\right) \in \mathbb{R}^d$$



$$\text{Enc}(\overline{x}) = \text{Pool}\left(h_1, \ldots, h_{|\overline{x}|}\right) \in \mathbb{R}^d$$

"Pooling" functions map from a sequence of items of type t to a single item of type t

$$\text{Pool} : \mathbb{R}^{d\times n} \to \mathbb{R}^d$$

n vectors of length *d* — single vector of length *d*

$$\text{MeanPool}(h_1, \ldots, h_{|\overline{x}|}) = \frac{1}{|\overline{x}|}\sum_{i=1}^{|\overline{x}|} h_i$$

$$\text{BidirMeanPool}(\vec{h}_1, \ldots, \vec{h}_{|\overline{x}|}; \overleftarrow{h}_1, \ldots, \overleftarrow{h}_{|\overline{x}|}) = \frac{1}{|\overline{x}|}\sum_{i=1}^{|\overline{x}|}\left[\vec{h}_i; \overleftarrow{h}_i\right]$$

$$\text{BidirLastPool}(\vec{h}_1, \ldots, \vec{h}_{|\overline{x}|}; \overleftarrow{h}_1, \ldots, \overleftarrow{h}_{|\overline{x}|}) = \left[\vec{h}_{|\overline{x}|}; \overleftarrow{h}_{|\overline{x}|}\right]$$

补上一个新 Sequence Embedding 的应用

Natural Language Inference

- Aka: recognizing textual entailment
- Given two sentences $\overline{x}_p$(*premise*) and $\overline{x}_h$(*hypothesis*), determine the following:
  - If $\overline{x}_p$ is true, then is $\overline{x}_h$ always true?
  - 3 possible labels:
    - Entailment: $\overline{x}_p \vDash \overline{x}_h$
    - Contradiction: $\overline{x}_p \vDash \neg\overline{x}_h$
    - Neutral: $\overline{x}_p \nvDash \overline{x}_h$

  $f : \mathcal{V}^+ \times \mathcal{V}^+ \to \Delta^{\{\text{entailment,contradiction,neutral}\}}$
- How might we use sentence embeddings to implement f?

$$f : \mathcal{V}^+ \times \mathcal{V}^+ \to \Delta^{\{\text{entailment,contradiction,neutral}\}}$$

Output：哪种关系

vector concatenation

$$f(\overline{x}_p, \overline{x}_h) = \text{softmax}(W\,[\text{Enc}(\overline{x}_p); \text{Enc}(\overline{x}_h)] + b)$$

text encoding of premise — text encoding of hypothesis

Train using labeled data
$\overline{x}_p, \overline{x}_h, y \in \{\text{entailment}, \text{contradiction}, \text{neutral}\}$

Can be used to measure quality of sentence representations! Important!

- What can we do with sequence embeddings?
  - Text classification

    $f_\theta(\overline{x}) = \sigma(W\text{Enc}(\overline{x}) + b) \qquad f : \mathcal{V}^+ \to \mathcal{C}$
  - Multi-sentence classification, e.g., natural language inference

    $f : \mathcal{V}^+ \times \mathcal{V}^+ \to \Delta^{\{\text{entailment,contradiction,neutral}\}}$

    $f(\overline{x}_p, \overline{x}_h) = \text{softmax}(W\,[\text{Enc}(\overline{x}_p); \text{Enc}(\overline{x}_h)] + b)$
  - Retrieval $\quad \arg\max_{\overline{x}' \in \mathcal{D}} \text{sim}(\text{Enc}(\overline{x}), \text{Enc}(\overline{x}'))$
- **Conditional sequence generation!**
  - Aka "sequence transduction"
  - E.g., machine translation, response generation in dialogue

Want to implement a new task!

- **Task:** map from text in one language (L1) to a distribution over texts in another language (L2), assigning high probability to texts that preserves the input text meaning

$$f : \mathcal{V}_{\text{L1}}^+ \to \Delta^{\mathcal{V}_{\text{L2}}^+}$$

$\overline{x}_{\text{English}}$ = *Oula A. Alrifai is a Syrian emigrant to the United States and writer for various Washington-based think tanks.*

Google Translate

$\overline{x}_{\text{Malay}}$ = *Oula A. Alrifai ialah seorang pendatang Syria ke Amerika Syarikat dan penulis untuk pelbagai badan pemikir yang berpangkalan di Washington.*

- What do we get from sequence models? Recall.
  - Autoregressive (token-by-token) generation
  - Sentence embeddings $\text{Enc}(\overline{x})$

How to translate

End-to-end, we can think about the translation problem as generating an output sequence token-by-token by [mapping from some input word to the target vocabulary, using the target language syntax:]

- **Key idea:** encode some input into a vector, decode it using autoregressive generation
  - Aka: sequence-to-sequence, encoder-decoder, sequence transduction, conditional language model…

[Core idea!]

1. Encode input sentence: $\text{Enc}(\overline{x}_{\text{L1}})$
2. Generate output sequence token-by-token, conditioning on previously-generated tokens **and the input sentence embedding:**

$$\overline{y}_{\text{L2}} = \langle y_1, \ldots, y_n\rangle \quad y_n = \texttt{EOS}$$

$$y_i \sim p(Y_i \mid y_1, \ldots, y_{i-1}, \text{Enc}(\overline{x}_{\text{L1}}); \theta_{\text{dec}})$$

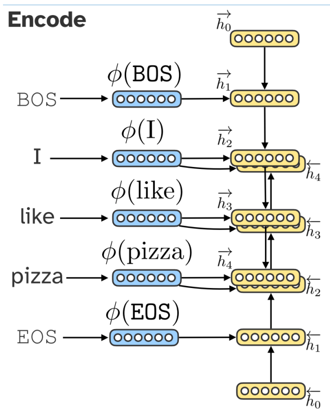
previously-generated tokens (empty when starting translation) typically, we use *y* to denote output tokens

embedding of sequence to translate

parameters of decoder model (e.g., RNN parameters)

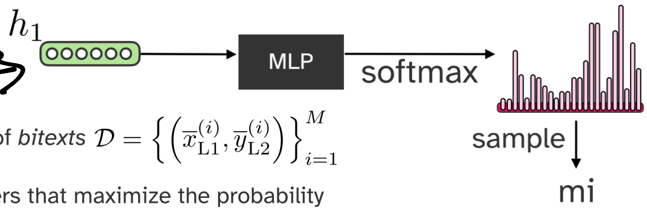
(you can condition on other things, too! e.g., images)

We can use RNN to encode & decode:



$\mathrm{Enc}(\overline{x}) = \mathrm{Pool}\left(h_1, \ldots, h_{|\overline{x}|}\right) \in \mathbb{R}^d$

BOS → φ(BOS).



- Given a training dataset of *bitexts* $\mathcal{D} = \left\{ \left(\overline{x}^{(i)}_{\mathrm{L1}}, \overline{y}^{(i)}_{\mathrm{L2}} \right) \right\}_{i=1}^{M}$
- We want to find parameters that maximize the probability of target sequences $\overline{y}$ conditioned on input sequence $\overline{x}$

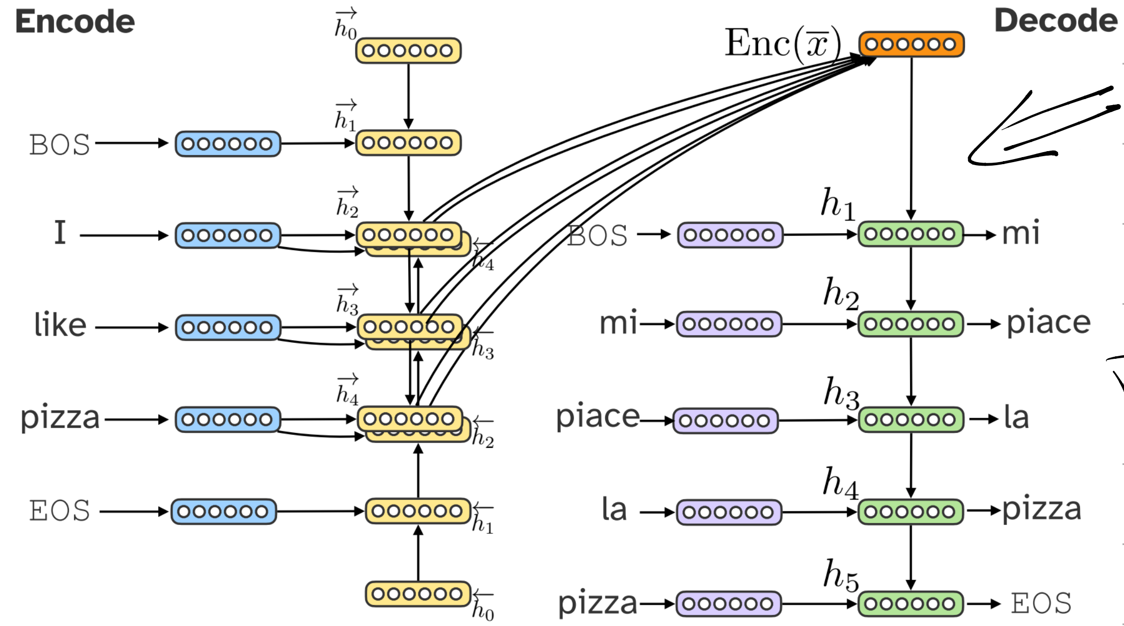
$$\theta^* = \arg\max_{\theta} \left(-\sum_{i=1}^{M} \sum_{j=1}^{|\overline{y}^{(i)}_{\mathrm{L2}}|} \log p \left(y^{(i)}_j \mid y^{(i)}_1, \ldots, y^{(i)}_{j-1}, \mathrm{Enc}_{\theta_{\mathrm{enc}}}\left(\overline{x}^{(i)}_{\mathrm{L1}}\right); \theta_{\mathrm{dec}} \right) \right)$$

"previously-generated" (teacher-forcing) tokens for example (i)

embedding of sequence to translate

parameters of decoder model

Training



Pipline ⊕

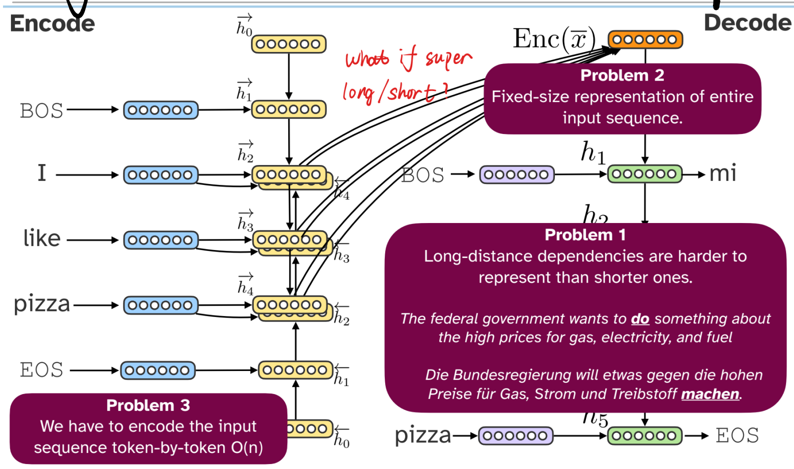
△: Enc可以是单独训后直接fix住，只训Dec
也可finetune

△ Options for backprop: use pre-trained θ_{enc} and freeze, or finetune by backprop directly $\theta = \theta_{\mathrm{enc}} \cup \theta_{\mathrm{dec}}$

$p(y \mid x)$

- Machine translation
- Summarization (long text → short text)
- Dialogue (previous utterances → next utterance)
- Syntactic parsing (input sentence → serialized parse)
- Code generation (natural language → Python code)

Application Overall ⟹
of this Core Idea Model



但并非 perfect:

⇐ Prob 1: 有些translation细节，其dependcy链很长 (RNN毛病)

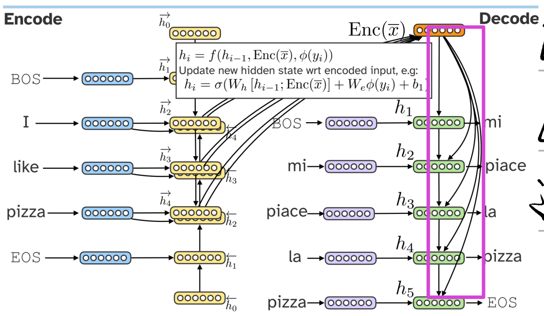
Prob 2: Pooling $(h_1, \ldots, h_{|\overline{x}|})$ 是否单一？

Prob3: RNN毛病

Prob2 Description

$$\mathrm{MeanPool}(h_1, \ldots, h_{|\overline{x}|}) = \frac{1}{|\overline{x}|} \sum_{i=1}^{|\overline{x}|} h_i = \sum_{i=1}^{|\overline{x}|} \frac{1}{|\overline{x}|} h_i$$

equally weight contribution of each input token / index



Prob1 简单解决: Enc(x) 给每个decode过程都补上。

$$\mathrm{Enc}(\bar{x}) = \sum_{i=1}^{|\bar{x}|} p(i) h_i$$
$$p(I) \in \Delta^{\mathbb{N}_{1:|\bar{x}|}}$$

- Generalization of pooling: assign a **probability distribution** over input indices, and compute **weighted** sum over hidden states need: ①
- What if this probability distribution could change during generation? ②.

With respect to prob 2, can we average with weight? And the weight can be in form of prob dist?

⇒ Attention. Coming Soon

Lec 10: Attention

$$\mathrm{Enc}(\bar{x}, g) = \sum_{i=1}^{|\bar{x}|} p(i \mid g) h_i$$
$$g \in \mathbb{R}^d \quad p : \mathbb{R}^d \to \Delta^{\mathbb{N}_{1:|\bar{x}|}}$$

- Compute a new distribution over input indices depending on some other vector g

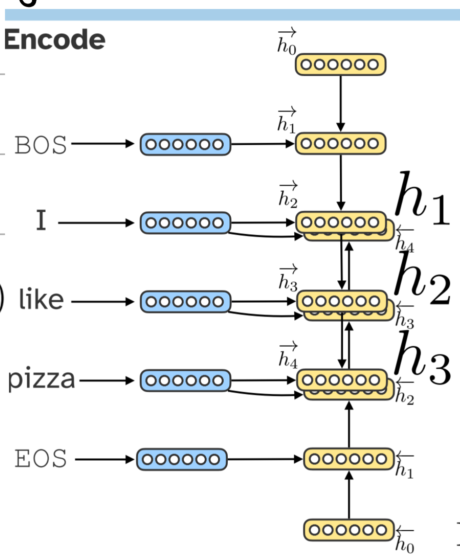
Two questions:
- Where does g come from?
- How to implement p?

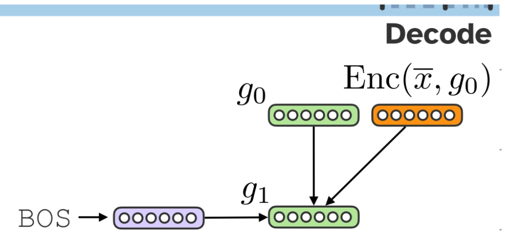
Our need: ② ① *(of the previously mentioned problems)

– Formulation*: We want weight: $p(i \mid g)$.

$g \in \mathbb{R}^d$, $p : \mathbb{R}^d \to \Delta^{N_{1:|\bar{x}|}}$

g can be modeled as hidden state:

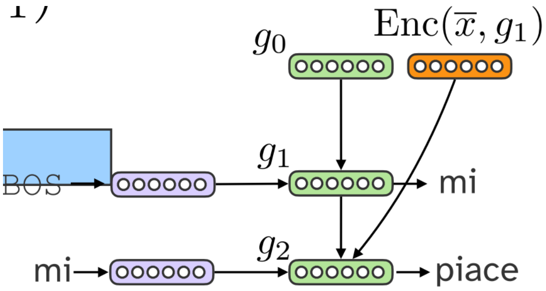




$$g_i = f(g_{i-1}, \mathrm{Enc}(\bar{x}, g_{i-1}), \phi(y_i))$$

text encoding also depends on the previous state!

Recall: $\mathrm{Enc}(\bar{x}, g) = \sum_{i=1}^{|\bar{x}|} p(i \mid g) h_i$



- Update decoding hidden state based on an updated version of the input sequence encoding

$$g_i = f(g_{i-1}, \mathrm{Enc}(\bar{x}, g_{i-1}), \phi(y_i))$$

- The updated sequence is determined via a weighted sum over input hidden states

$$\mathrm{Enc}(\bar{x}, g) = \sum_{i=1}^{|\bar{x}|} p(i \mid g) h_i \qquad p : \mathbb{R}^d \to \Delta^{\mathbb{N}_{1:|\bar{x}|}}$$

- Next time: how do we compute $p(I \mid g)$?

– One solution: Attention

- An analogy: databases
- A **query** is some search term
- A table has **keys** paired with **values**
- We want to return the **values** of the **keys** most related to the **query**

Query: what are some words for mammals in Turkish?

i	Key	Value
1	sheep	koyun
2	river	ırmak
3	author	yazar
4	bull	boğa
5	fly	sinek
6	radio	radyo
7	farmer	çiftçi
8	plant	bitki
9	actress	aktris
10	home	ev
11	snail	salyangoz

$$p(i \mid q, k_i) \propto \mathrm{sim}(\mathrm{Enc}(q), \phi(k_i))$$

Weight of each item is proportional to the similarity of its key with the query

hidden state

$p(i \mid q, k_i) v_i$ value.

Weight each value accordingly

-In our RNN Scenario:

$$p(i \mid q, k_i) \propto \mathrm{sim}(\mathrm{Enc}(q), \phi(k_i)) \equiv p(i \mid g, h_i) \propto \mathrm{sim}(g, h_i)$$

Query Key

$$\mathrm{Enc}(\overline{x}, g) = \sum_{i=1}^{|\overline{x}|} p(i \mid g) h_i$$

Value

- **Query:** current decoder hidden state g
- **Key:** encoder hidden state (at some index i) h_i
- **Value:** also the encoder hidden state at index i

$$p(i \mid g, h_i) \propto \mathrm{sim}(g, h_i)$$

Query Key

$$\mathrm{Enc}(\overline{x}, g) = \sum_{i=1}^{|\overline{x}|} p(i \mid g) h_i$$

Value

There are multiple way to eval $\mathrm{sim}(q, k_i)$ ...

$$p(i \mid q, k_i) \propto \mathrm{sim}(q, k_i) = \frac{e^{\mathrm{sim}(q,k_i)}}{\sum_{i'=1}^{|\overline{x}|} e^{\mathrm{sim}(q,k_{i'})}}$$

$q \in \mathbb{R}^d$

$\mathbf{k} \in \mathbb{R}^{d' \times |\overline{x}|}$

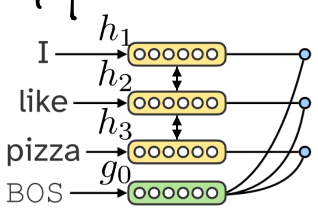
$\mathrm{sim}(q, \mathbf{k}) = q^\intercal \mathbf{k}$ Dot product attention $d = d'$

$\mathrm{sim}(q, \mathbf{k}) = q^\intercal W \mathbf{k}$ Bilinear attention $W \in \mathbb{R}^{d \times d'}$

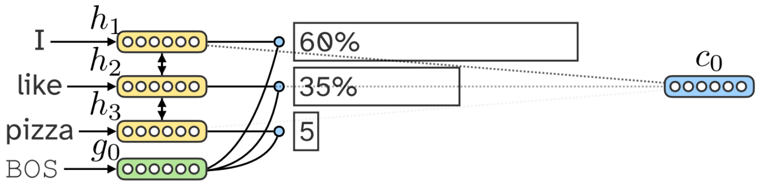
$\mathrm{sim}(q, \mathbf{k}) = w_2^\intercal \tanh(W_1[q; \mathbf{k}])$ MLP attention $W_1 \in \mathbb{R}^{(d+d') \times d''}$, $w_2 \in \mathbb{R}^{d''}$

$\mathrm{sim}(q, \mathbf{k}) = \frac{q^\intercal \mathbf{k}}{\sqrt{d'}}$ Scaled dot product attention

Pipeline !!



$s_3 = g_0^\intercal h_3$



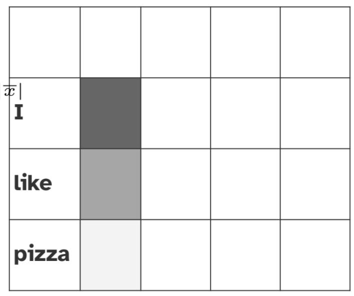
Similarities between query and keys

$s_0 = g_0^\intercal \mathbf{h} \in \mathbb{R}^{|\overline{x}|}$

Weights over keys (weights sum to 1)

$\alpha_0 = p(\cdot \mid g_0, \mathbf{h}) = \mathrm{softmax}(s_0) \in \Delta^{\mathbb{N}_{1:|\overline{x}|}}$

α is calculated via s ⇐



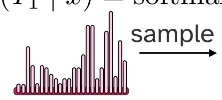
Encoding is weighted sum over values

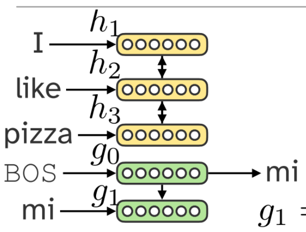
$c_0 = \mathrm{Enc}(\overline{x}, g_0) = \sum_{i=1}^{|\overline{x}|} \alpha_{0,i} h_i \in \mathbb{R}^d$

usage

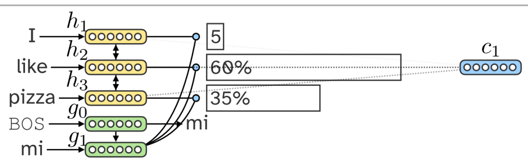
$p(Y_1 \mid \overline{x}) = \mathrm{softmax}(f(c_0, g_0))$

sample!





$g_1 = g(g_0, c_0, y_1)$



Similarities between query and keys

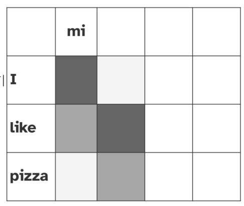
$s_1 = g_1^\intercal \mathbf{h} \in \mathbb{R}^{|\overline{x}|}$

Weights over keys (weights sum to 1)

$\alpha_1 = p(\cdot \mid g_1, \mathbf{h}) = \mathrm{softmax}(s_1) \in \Delta^{\mathbb{N}_{1:|\overline{x}|}}$

Encoding is weighted sum over values

$c_1 = \mathrm{Enc}(\overline{x}, g_1) = \sum_{i=1}^{|\overline{x}|} \alpha_{1,i} h_i \in \mathbb{R}^d$



Update g. and repeat......

Overview:

- We encoded our input sequence into hidden states: $h_1, \ldots, h_{|\overline{x}|} = \mathbf{h} \in \mathbb{R}^{d \times |\overline{x}|}$
- Now we want to predict the next word y_{i+1}
- We have access to the previous decoder hidden state g_i
- First, compute attention scores for each input hidden state using similarity between query (g_i) and keys ($\mathbf{h}$): $s_i = a(g_i, \mathbf{h}) \in \mathbb{R}^{|\overline{x}|}$
- Then, take softmax of attention scores to get a distribution over keys: $\alpha_i = \mathrm{softmax}(s_i) \in \Delta^{\mathbb{N}_{1:|\overline{x}|}}$
- Finally, compute a weighted sum of values ($\mathbf{h}$) using this distribution $$c_i = \sum_{j=1}^{|\overline{x}|} \alpha_{i,j} h_j \in \mathbb{R}^d$$
- Use the weighted sum to predict the next word: $p(Y_{i+1} \mid \overline{x}, y_1, \ldots, y_i) = \mathrm{softmax}(f(c_i, g_i))$
- Use the weighted sum to update the decoder hidden state: $g_i = g(g_{i-1}, c_{i-1}, y_i)$

Complexity

$h_1, \ldots, h_{|\overline{x}|} = \mathbf{h} \in \mathbb{R}^{d \times |\overline{x}|}$

O(n) $s_i = a(g_i, \mathbf{h}) \in \mathbb{R}^{|\overline{x}|}$

$\alpha_i = \mathrm{softmax}(s_i) \in \Delta^{\mathbb{N}_{1:|\overline{x}|}}$

$c_i = \sum_{j=1}^{|\overline{x}|} \alpha_{i,j} h_j \in \mathbb{R}^d$

$p(Y_{i+1} \mid \overline{x}, y_1, \ldots, y_i) = \mathrm{softmax}(f(c_i, g_i))$ O(m)

$g_i = g(g_{i-1}, c_{i-1}, y_i)$

O(n m)

m: dimension of vocab

n: input length

- **Main problem: fixed-size representation ("bottleneck") of input sequence at each stage of decoding**
- Attention fixes this by allowing access to different parts of the input sequence based on where we are in decoding
- While decoding token-by-token, we can "focus" on different input parts
  - Helps with vanishing gradient problem via shortcut to far-away states
  - Provides some kind of interpretability ("soft" alignment) learned without any supervision

How attention solved this problem.

And other benefits.

Attention was initially developed for sequence-to-sequence problems

But we can also use it during single-sequence generation to solve analogous problems:

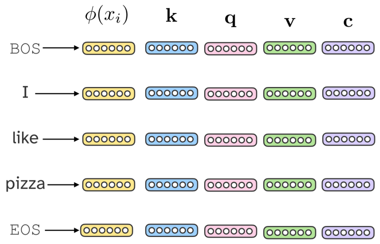
- Problem 1: long-distance dependencies
- Problem 2: fixed-size representation of sequence so far

- Self-Attention

- Keys, values, and queries are transformations of the same underlying representations
- For each token generated at index i, we have:
  - A key: $k_i = Kh_i \in \mathbb{R}^d \quad K \in \mathbb{R}^{d\times d}$
  - A value: $v_i = Vh_i \in \mathbb{R}^d \quad V \in \mathbb{R}^{d\times d}$
- When trying to generate our next token at index j, we need a query: $q_j = Qh_j \in \mathbb{R}^d \quad Q \in \mathbb{R}^{d\times d}$
  - Scores over keys: $s_{ji} = q_j^\intercal k_i$
  - Attention distribution over keys: $\alpha_{ji} = \dfrac{\exp(s_{ji})}{\sum_{i'=1}^{j}\exp(s_{ji'})}$
  - Weighted sum of values: $c_j = \sum_{i=1}^{j} \alpha_{ji} v_i$

$\cancel{h_{j+1} = g(h_j, c_j, x_{j+1})} \quad h_{j+1} = \phi(x_{j+1})$

Crucial for supporting parallel commutation



$\mathbf{s} = \mathbf{q}^\intercal\mathbf{k} \in \mathbb{R}^{|\overline{x}|\times|\overline{x}|}$

$\alpha = \mathrm{softmax}_{\mathrm{dim}=1}(\mathbf{s}) \in \{\Delta^{\mathbb{N}_{1:|\overline{x}|}}\}^{|\overline{x}|}$

$\mathbf{c} = \alpha\mathbf{v} \in \mathbb{R}^{d\times|\overline{x}|}$

We can get context-sensitive representations of each input token **in parallel**!

Two problems.

Prob1:

Problem 1 No position information!

- Currently, our model is operating only on embedding of individual wordtypes $\phi(x_i) = \phi(x)$
- Let's also embed their indices: $\phi(x_i) = \phi(x) + \phi(i)$

$k_i = K\phi(x_i) \in \mathbb{R}^d$

$v_i = V\phi(x_i) \in \mathbb{R}^d$

$q_j = Q\phi(x_j) \in \mathbb{R}^d$

$\mathbf{s} = \mathbf{q}^\intercal\mathbf{k} \in \mathbb{R}^{|\overline{x}|\times|\overline{x}|}$

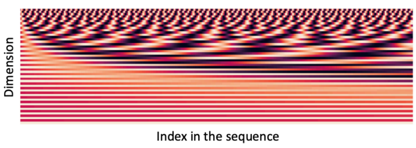
$\alpha = \mathrm{softmax}_{\mathrm{dim}=1}(\mathbf{s}) \in \{\Delta^{\mathbb{N}_{1:|\overline{x}|}}\}^{|\overline{x}|}$

$\mathbf{c} = \alpha\mathbf{v} \in \mathbb{R}^{d\times|\overline{x}|}$

Problem 2 No nonlinearities!

- Sinusoidal embeddings

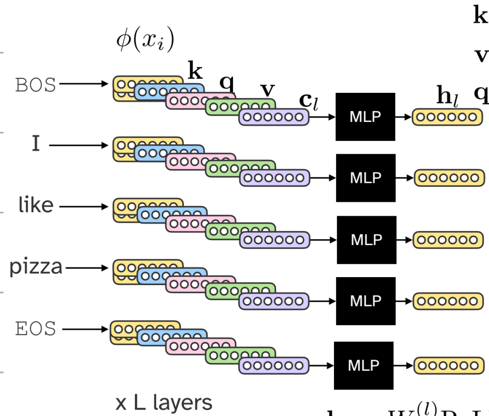
$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



- (in practice, most methods use learned embeddings)

Prob2: $h_l = MLP(c_l)$

Calculate qkv: First block use embedding $\phi(x_i)$; Following blocks use h_{l-1} from the last block



$\mathbf{k}^{(l)} = K^{(l)}\mathbf{h}_l$

$\mathbf{v}^{(l)} = V^{(l)}\mathbf{h}_l$

$\mathbf{q}^{(l)} = Q^{(l)}\mathbf{h}_l$

$\mathbf{s} = \mathbf{q}^\intercal\mathbf{k} \in \mathbb{R}^{|\overline{x}|\times|\overline{x}|}$

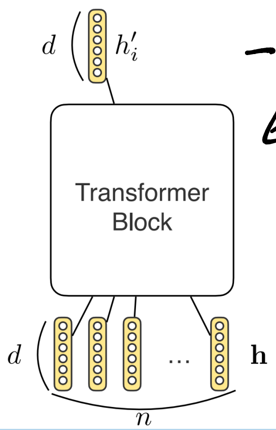
$\alpha = \mathrm{softmax}_{\mathrm{dim}=1}(\mathbf{s}) \in \{\Delta^{\mathbb{N}_{1:|\overline{x}|}}\}^{|\overline{x}|}$

$\mathbf{c}_l = \alpha\mathbf{v}^{(l)}$

$\mathbf{h}_l = W_2^{(l)}\mathrm{ReLU}(W_1^{(l)}\mathbf{c}_l + b_1^{(l)}) + b_2^{(l)}$

Lec 11: Transformer

- A transformer block takes as input a sequence of input vectors $\mathbf{h} \in \mathbb{R}^{d\times n}$
- It generates a sequence of output vectors $\mathbf{h}' \in \mathbb{R}^{d\times n}$
- For now, let's focus on how it computes the output vector corresponding to the input at index i, $h_i' \in \mathbb{R}^d$



- Encoder

Compute attention over input vectors

$\mathbf{k} = K\mathbf{h} \in \mathbb{R}^{d_k\times n}$

$q_i = Qh_i \in \mathbb{R}^{d_k}$

$s_i = \dfrac{q_i^\intercal\mathbf{k}}{\sqrt{d_k}} \in \mathbb{R}^n$

$\alpha_i = \mathrm{softmax}(s_i) \in \Delta^{\mathbb{N}_{1:n}}$

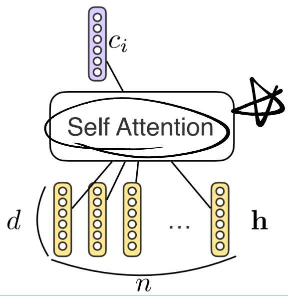
$\mathbf{v} = V\mathbf{h} \in \mathbb{R}^{d_k\times n}$

$c_i = \sum_{i'=1}^{n} \alpha_i v_{i'}$

$c_i = \mathrm{SelfAttention}(\mathbf{h})_i$

(single head)

$\theta_{\mathrm{SelfAttention}} = \{K, Q, V\}$

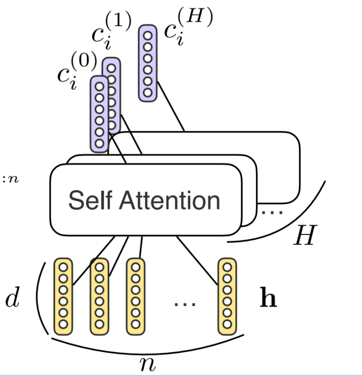


MultiHead Attention

Compute attention over input vectors

$$\theta_{\text{SelfAttention}_j} = \{K_j, Q_j, V_j\}$$

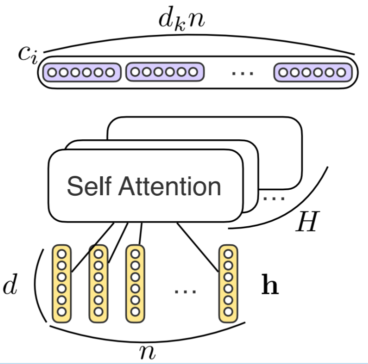
$$\mathbf{k}^{(j)} = K_j \mathbf{h} \in \mathbb{R}^{d_k \times n}$$
$$q_i^{(j)} = Q_j h_i \in \mathbb{R}^{d_k}$$
$$s_i^{(j)} = \frac{q_i^{(j)\intercal} \mathbf{k}^{(j)}}{\sqrt{d_k}} \in \mathbb{R}^n$$
$$\alpha_i^{(j)} = \text{softmax}\left(s_i^{(j)}\right) \in \Delta^{\mathbb{N}_{1:n}}$$
$$\mathbf{v}^{(j)} = V_j \mathbf{h} \in \mathbb{R}^{d_k \times n}$$
$$c_i^{(j)} = \sum_{i'=1}^{n} \alpha_{i'}^{(j)} v_{i'}^{(j)}$$
$$c_i^{(j)} = \text{SelfAttention}_j(\mathbf{h})_i$$



Compute attention over input vectors

$$c_i^{(j)} = \text{SelfAttention}_j(\mathbf{h})_i$$
$$c_i = \left[c_i^{(1)}; \ldots; c_i^{(H)}\right]$$

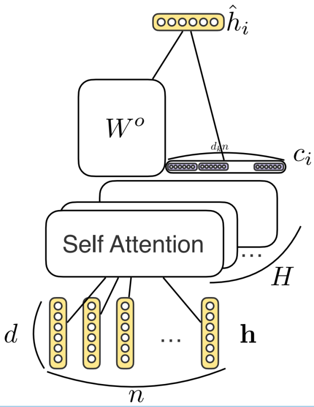
concat



Compute attention over input vectors

$$c_i^{(j)} = \text{SelfAttention}_j(\mathbf{h})_i$$
$$c_i = \left[c_i^{(1)}; \ldots; c_i^{(H)}\right]$$
$$\hat{h}_i = W^o c_i$$

residual + layernorm

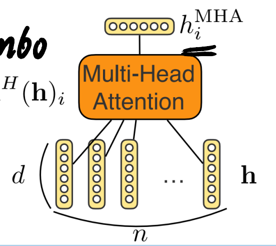


1. Compute attention over input vectors

$$\theta_{\text{MultiHeadAttention}^H} = \{W^o\} \cup \{K_j, Q_j, V_j\}_{j=1}^H$$

$$c_i^{(j)} = \text{SelfAttention}_j(\mathbf{h})_i$$
$$c_i = \left[c_i^{(1)}; \ldots; c_i^{(H)}\right]$$
$$\hat{h}_i = W^o c_i \quad \text{linear combo}$$
$$h_i^{\text{MHA}} = \text{MultiHeadAttention}^H(\mathbf{h})_i$$

Multi-headed attention allows us to look at multiple places in the input sequence at once.



Residual connection allows for smoother gradients.

Layer normalization cuts down on uninformative variation in activations, speeding up training.

1. Compute attention over input vectors

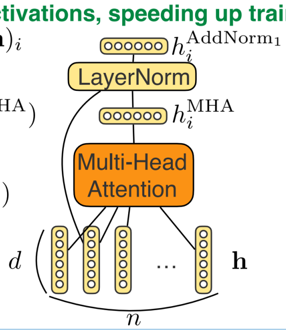
$$h_i^{\text{MHA}} = \text{MultiHeadAttention}^H(\mathbf{h})_i$$

2. Add and norm

$$h_i^{\text{AddNorm}_1} = \text{LayerNorm}(h_i + h_i^{\text{MHA}})$$

$$\text{LayerNorm}: \mathbb{R}^d \to \mathbb{R}^d$$
$$\text{LayerNorm}(x) = \frac{g}{\sigma(x)}(x - \mu(x))$$

$$\mu(x) = \frac{1}{d}\sum_{i=1}^{d} x_i \quad \sigma(x) = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$$



1. Compute attention over input vectors

$$h_i^{\text{MHA}} = \text{MultiHeadAttention}^H(\mathbf{h})_i$$

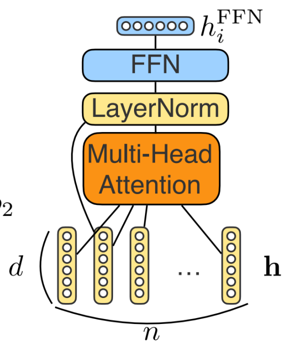
2. Add and norm

$$h_i^{\text{AddNorm}_1} = \text{LayerNorm}(h_i + h_i^{\text{MHA}})$$

3. Feedforward layer

$$h_i^{\text{FFN}} = \text{ReLU}(h_i^{\text{AddNorm}_1} W_1 + b_1) W_2 + b_2$$

Here is a nonlinearity that makes learning easier!



1. Compute attention over input vectors

$$h_i^{\text{MHA}} = \text{MultiHeadAttention}^H(\mathbf{h})_i$$

2. Add and norm

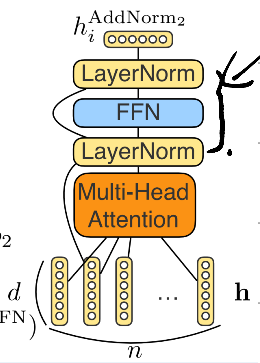
$$h_i^{\text{AddNorm}_1} = \text{LayerNorm}(h_i + h_i^{\text{MHA}})$$

3. Feedforward layer

$$h_i^{\text{FFN}} = \text{ReLU}(h_i^{\text{AddNorm}_1} W_1 + b_1) W_2 + b_2$$

4. Another add and norm

$$h_i^{\text{AddNorm}_2} = \text{LayerNorm}(h_i^{\text{AddNorm}_1} + h_i^{\text{FFN}})$$

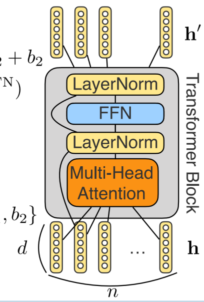


Residual Connection

$$\mathbf{h}^{\text{MHA}} = \text{MultiHeadAttention}^H(\mathbf{h})$$
$$\mathbf{h}^{\text{AddNorm}} = \text{LayerNorm}(\mathbf{h} + \mathbf{h}^{\text{MHA}})$$
$$\mathbf{h}^{\text{FFN}} = \text{ReLU}(\mathbf{h}^{\text{AddNorm}} W_1 + b_1) W_2 + b_2$$
$$\mathbf{h}' = \text{LayerNorm}(\mathbf{h}^{\text{AddNorm}} + \mathbf{h}^{\text{FFN}})$$

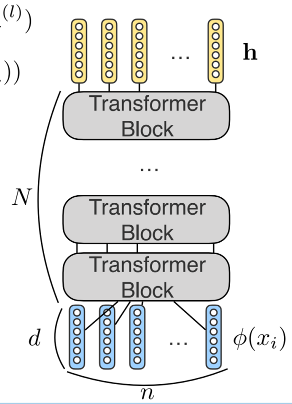
$$\mathbf{h}' = \text{TransformerBlock}(\mathbf{h})$$

$$\theta_{\text{TransformerBlock}} = \theta_{\text{MultiHeadAttention}^H} \cup \{W_1, W_2, b_1, b_2\}$$



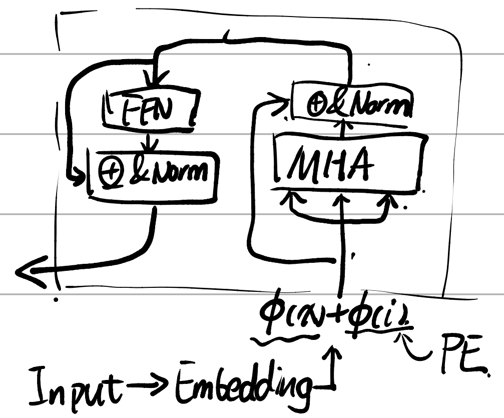
$$\mathbf{h}^{(l+1)} = \text{TransformerBlock}_{l+1}(\mathbf{h}^{(l)})$$
$$\mathbf{h}^N = \text{TransformerEncoder}(\phi(x_1), \ldots, \phi(x_n))$$
$$\mathbf{h}^N \in \mathbb{R}^{d \times n}$$

Input: $\phi(x_i) = \phi(x) + \phi(i)$
word + position embeddings
$\overline{x} = \langle x_1, \ldots, x_n \rangle$



×N

Encoder.

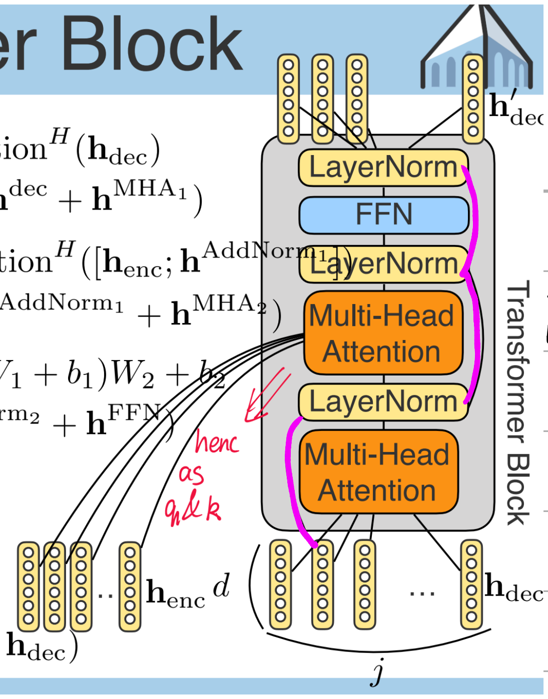


Decoder Block

$$\mathbf{h}^{\mathrm{MHA}_1} = \mathrm{MultiHeadAttention}^H(\mathbf{h}_{\mathrm{dec}})$$
$$\mathbf{h}^{\mathrm{AddNorm}_1} = \mathrm{LayerNorm}(\mathbf{h}^{\mathrm{dec}} + \mathbf{h}^{\mathrm{MHA}_1})$$
$$\mathbf{h}^{\mathrm{MHA}_2} = \mathrm{MultiHeadAttention}^H([\mathbf{h}_{\mathrm{enc}}; \mathbf{h}^{\mathrm{AddNorm}_1}])$$
$$\mathbf{h}^{\mathrm{AddNorm}_2} = \mathrm{LayerNorm}(\mathbf{h}^{\mathrm{AddNorm}_1} + \mathbf{h}^{\mathrm{MHA}_2})$$
$$\mathbf{h}^{\mathrm{FFN}} = \mathrm{ReLU}(\mathbf{h}^{\mathrm{AddNorm}_2} W_1 + b_1) W_2 + b_2$$
$$\mathbf{h}'_{\mathrm{dec}} = \mathrm{LayerNorm}(\mathbf{h}^{\mathrm{AddNorm}_2} + \mathbf{h}^{\mathrm{FFN}})$$

One last layer norm to get our final output vectors for each token generated so far!

$$\mathbf{h}'_{\mathrm{dec}} = \mathrm{DecoderBlock}(\mathbf{h}_{\mathrm{enc}}, \mathbf{h}_{\mathrm{dec}})$$



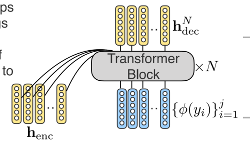
- Decoder
First MHA: self
Second MHA: q, k: henc
v: LN's result

同时, MHA & LN数量+1

Res连接都跟上

$$\mathbf{h}^{(l+1)}_{\mathrm{dec}} = \mathrm{DecoderBlock}_{l+1}(\mathbf{h}_{\mathrm{enc}}, \mathbf{h}^{(l+1)}_{\mathrm{dec}})$$
$$\mathbf{h}^{N}_{\mathrm{dec}} = \underset{\mathrm{Decoder}}{\mathrm{Transformer}}(\mathbf{h}_{\mathrm{enc}}, \phi(y_1), \ldots, \phi(y_j))$$
$$\underset{\mathrm{Decoder}}{\mathrm{Transformer}} : \mathbb{R}^{d \times n} \times \mathcal{V}^j \to \mathbb{R}^{d \times j}$$

The transformer decoder maps from a set of input encodings and a sequence of tokens generated so far to a set of vectors, each corresponding to a token in the currently-generated sequence



How to generate index j+1's word? Decoder 输入前j个word, 算出 h^N_{dec} 后, 取 h^N_j:

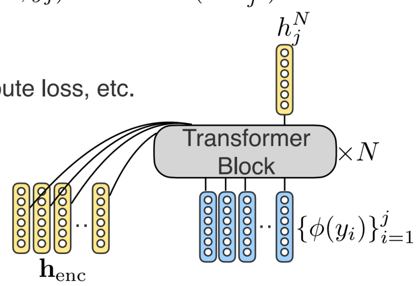
$W \in \mathbb{R}^{|\mathcal{V}| \times d}$,

$prob(Y_{j+1} | y_1, \ldots, y_j) = \mathrm{softmax}(W h^N_j)$

- Let's say we want to predict the word at index $j+1$
- All we need to do is transform this into a distribution over the vocabulary

$$p(Y_{j+1} \mid x_1, \ldots, x_n, y_1, \ldots, y_j) = \mathrm{softmax}(W h_j^N)$$
$$W \in \mathbb{R}^{|\mathcal{V}| \times d}$$

- Then we can sample, compute loss, etc.



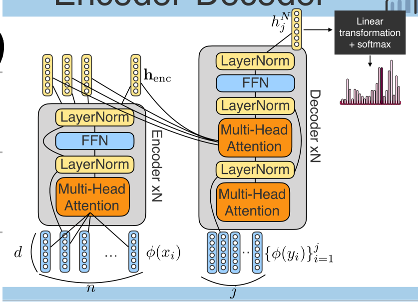
Given some paired data $(\overline{x}, \overline{y})$:

$\mathbf{h}_{\mathrm{enc}} = \underset{\mathrm{Encoder}}{\mathrm{Transformer}}(\overline{x})$ ← We can compute these in parallel on the GPU

$p(y_{i+1}) = \underset{\mathrm{Decoder}}{\mathrm{Transformer}}(\mathbf{h}_{\mathrm{enc}}, y_1, \ldots, y_i)$

Probability of a token should only depend on the ones that come before it! But we still want to take advantage of parallelism…

Encoder-Decoder



(Training of Enc&Dec)

Masking:

- When doing the forward pass on the decoder, self-attention can be parallelized
- But we don't want the model to learn to rely on "future" words in the sequence!
- Solution: during training, set $s_{ij} = -\infty$ if i (query index) < j (key/value index)

← Training: for parallelism, masking is important!! Mask 用在 Decoder 中第一个MHA

Recall the autoregressive approximation of sequence probability:

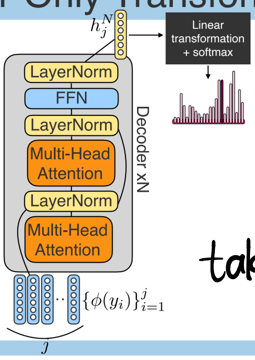
$$p(\overline{x}) = \prod_{i=1}^{|\overline{x}|} p(x_i \mid x_1, \ldots, x_{i-1})$$

Can we do this with a transformer model?

Yes! We just don't have an encoder (i.e., decoder-only LLM).

We still have to be careful about masking while training.

Decoder-Only Transformer



- Transformer for sequence generation
Decoder only.
take the last h^N_j!

What if we just want some representation of a sentence, and we don't really care about being able to generate sentences? $\mathrm{Enc}(\overline{x})$

We get this with the transformer encoder:

$$\mathbf{h}_{\mathrm{enc}} = \underset{\mathrm{Encoder}}{\mathrm{Transformer}}(\overline{x})$$

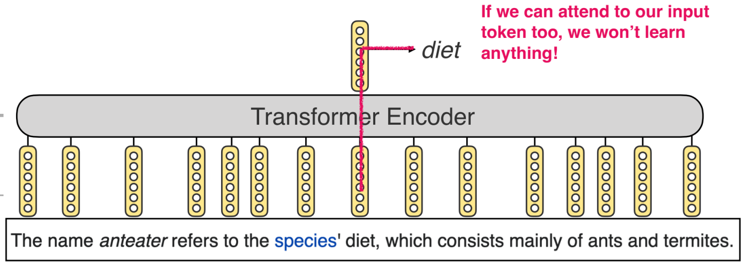
- Transformer for Sequence modeling
Encoder only

$$p(\overline{x}) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1} \ldots x_n)$$

- From our transformer encoder, we get
  $$h_i = \text{Transformer}_{\text{Encoder}}(\overline{x})_i \in \mathbb{R}^d$$
- We could just transform this to the vocabulary space, and compute the loss from there:
  $$p(X_i) = \text{softmax}(W h_i)$$
- Do you anticipate any problems? ✲

✲ Training can be problematic.

If we can attend to our input token too, we won't learn anything!



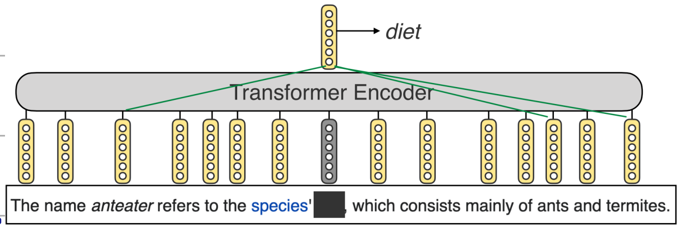
For word with index j, output h_j can see x_j !!

And our training goal is: $p(X_j = x_j) = \text{softmax}(W h_j)$

Information leakage!

⇒ Mask word j^*! For encoder to infer with x_i $(i \neq j)$! *(mask <u>some</u> tokens).

⇒ BERT



Instead: randomly sample some tokens to replace with a MASK placeholder token in the input, then train the model to predict those words