CSCE 638 Natural Language Processing Foundation and Techniques

Lecture 4: Tokenization, Language Modeling, and Decoding Methods

Kuan-Hao Huang

Spring 2026

(Some slides adapted from Karthik Narasimhan, Danqi Chen, Graham Neubig, Greg Durrett, and Vivian Chen)

Assignment 0

Assignment 0

RELEASE DATE: 01/20/2026

DUE DATE: 01/29/2026 11:59pm on Gradescope

LaTeX Template: https://www.overleaf.com/read/pzhhcsmdfyst#557346

Name: First-Name Last-Name UIN: 000000000

This assignment consists of two parts: a writing section and a programming section. For the writing section, please use the provided LaTeX template to prepare your solutions and remember to fill in your name and UIN. For the programming section, please follow the instructions carefully.

Discussions with others on course materials and assignment solutions are encouraged, and the use of AI tools as assistance is permitted. However, you must ensure that ***the final solutions are written in your own words****. It is your responsibility to avoid excessive similarity to others' work. Additionally, please clearly* ***indicate any parts where AI tools were used*** *as assistance.*

If you have any question, please send an email to csce638-ta-26s@list.tamu.edu

Assignment 1

Assignment 1

RELEASE DATE: 01/28/2026

DUE DATE: 02/10/2026 11:59pm on Gradescope

LaTeX Template: https://www.overleaf.com/read/tsrvwjgzwjrw#942964

Name: First-Name Last-Name UIN: 000000000

This assignment consists of two parts: a writing section and a programming section. For the writing section, please use the provided LaTeX template to prepare your solutions and remember to fill in your name and UIN. For the programming section, please follow the instructions carefully.

Discussions with others on course materials and assignment solutions are encouraged, and the use of AI tools as assistance is permitted. However, you must ensure that ***the final solutions are written in your own words****. It is your responsibility to avoid excessive similarity to others' work. Additionally, please clearly* ***indicate any parts where AI tools were used*** *as assistance.*

If you have any question, please send an email to csce638-ta-26s@list.tamu.edu

Lecture Plan

- Tokenization
  - Subwords
  - Byte-Pair Encoding
- Language Models
  - Definition of Language Models
  - N-Gram Language Models
  - Language Model Decoding Methods
  - Neural Language Models

Recap: Word Vectors

Map each word to a vector!

$$\text{similarity}(word1, word2) = \frac{\mathbf{v}_{word1} \cdot \mathbf{v}_{word2}}{\|\mathbf{v}_{word1}\| \times \|\mathbf{v}_{word2}\|}$$

$$\mathbf{v}_{great} = [0.12, 0.38, -0.91, 0.57, -0.64]$$

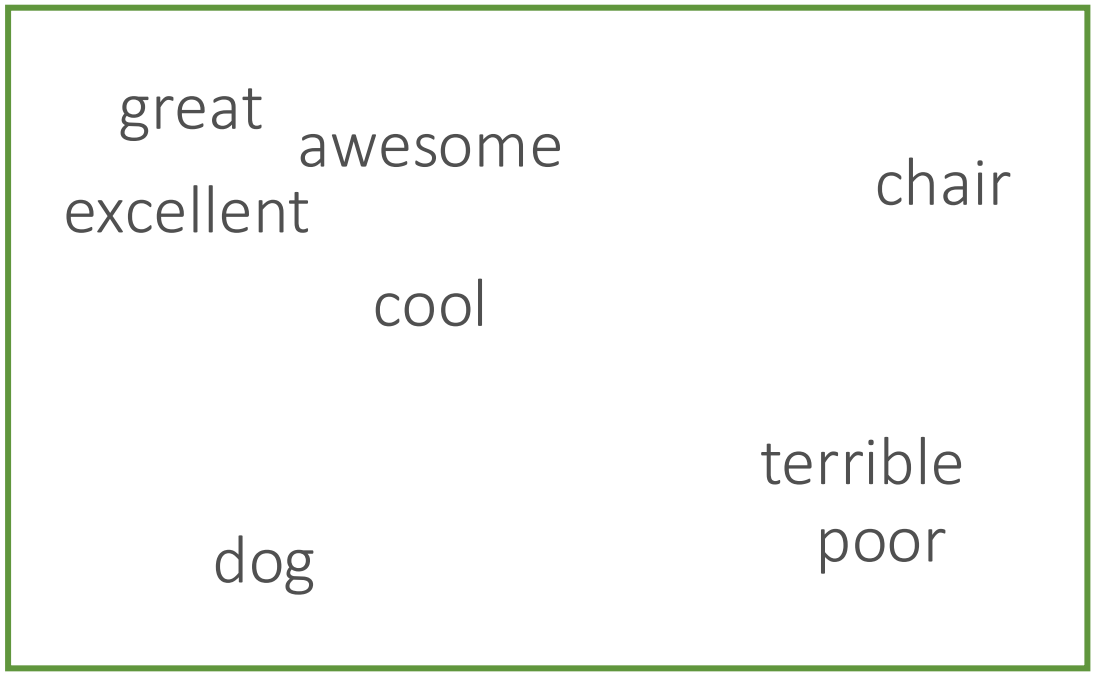
$$\mathbf{v}_{excellent} = [0.16, 0.47, -0.87, 0.50, -0.55]$$

$$\mathbf{v}_{awesome} = [0.08, 0.28, -0.90, 0.61, -0.54]$$

$$\mathbf{v}_{terrible} = [0.92, -0.36, 0.11, -0.24, 0.14]$$

$$\mathbf{v}_{poor} = [0.85, -0.40, 0.02, -0.31, 0.23]$$

Semantic meaning of words

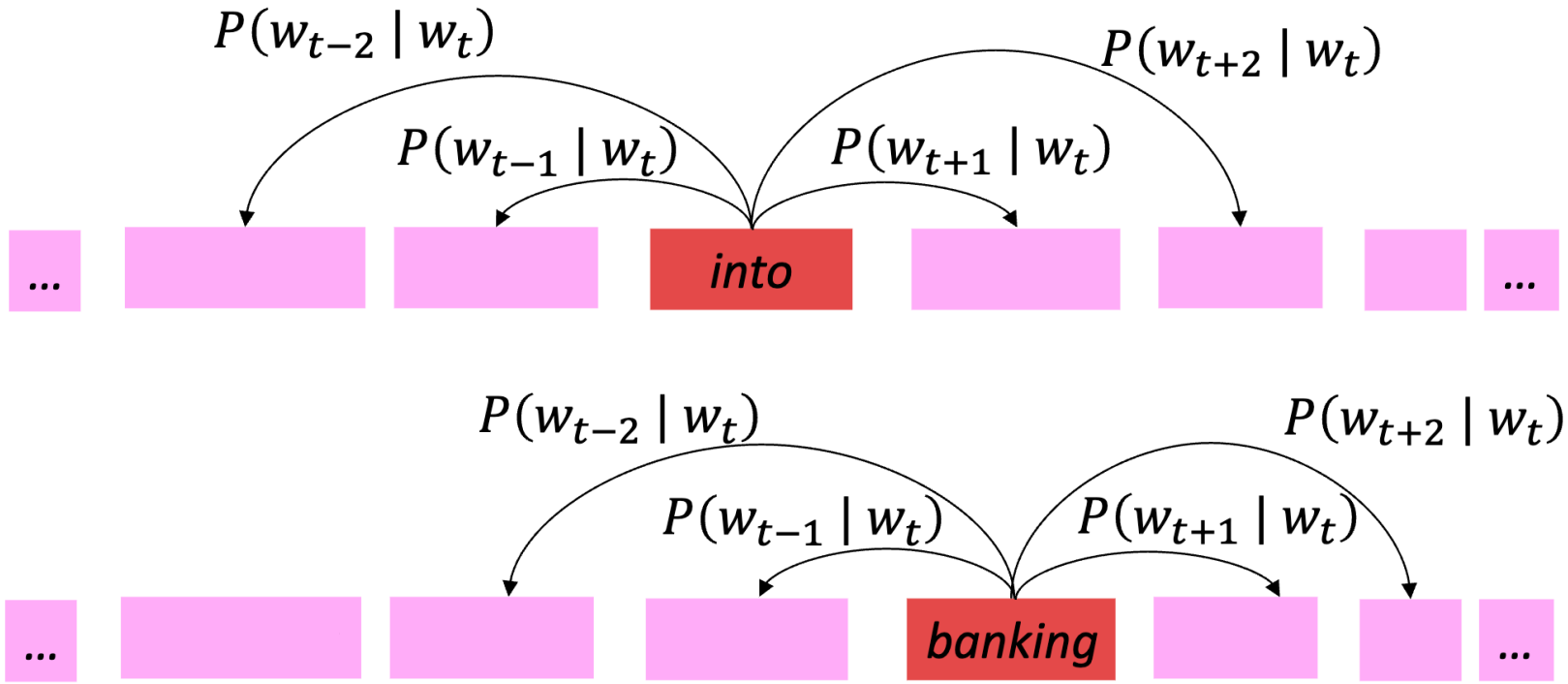


Semantic relationship between words

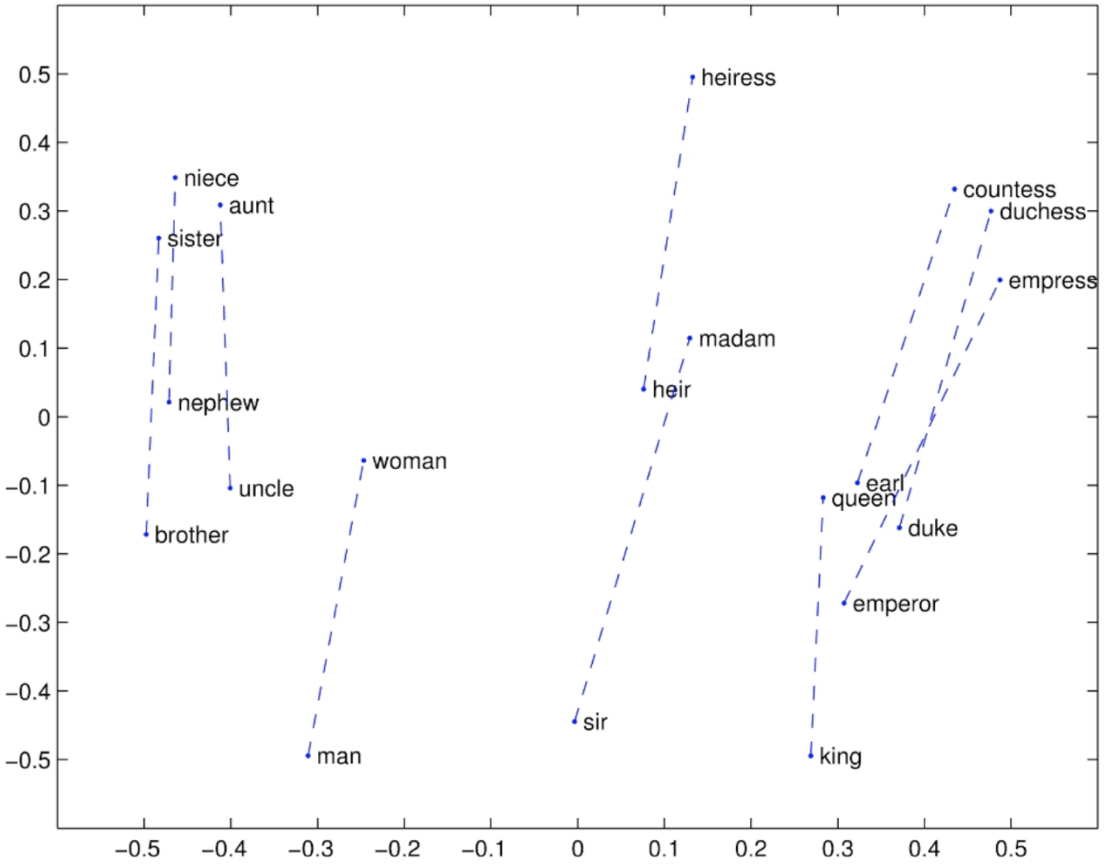
Recap: Word2Vec

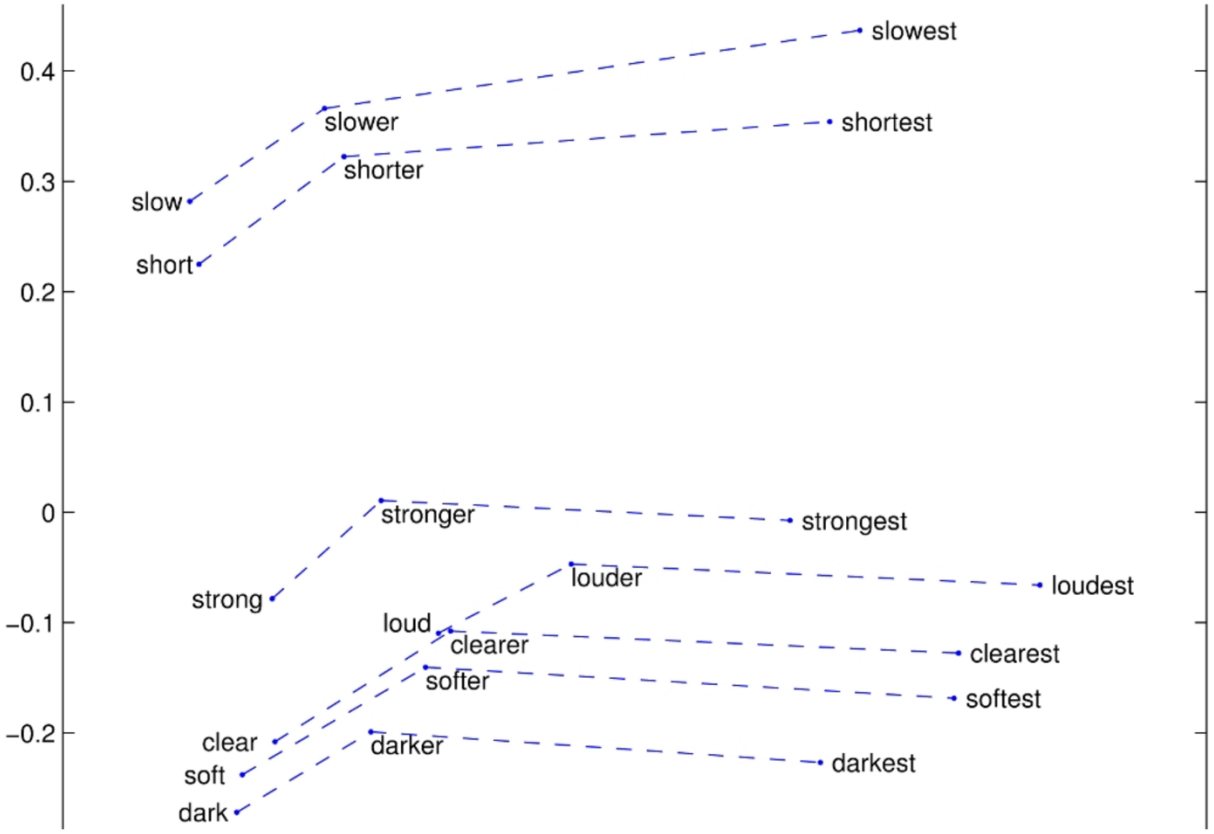
- **Main idea:** we want to use words to predict their context words
- Context: a fixed window of size m

Use center word w_t to predict context words w_{t-m} to w_{t+m}



Recap: Word2Vec





Tokenization

- Currently, we use word (and punctuation) as the basic unit to tokenize a text
  - I like this movie so much. → I + like + this + movie + so + much + .

What is the size of word embeddings (how many words)?

Size of Vocabulary

- The larger, the better?
- Storage? Computation?
- Do we need to consider all the words?
  - zcvahu
  - #$^&*
  - Low frequency words

Unknown Token

- We create an unknown token for all the words that have never been seen or low frequency words
  - <UNK>
- <UNK> has its own embedding
  - I like this movie &*# so much → I + like + this + movie + <UNK> + so + much + .
  - I like this movie sooooo much. → I + like + this + movie + <UNK> + much + .
- We can reduce the size of vocabulary
- We can handle unseen words

Is There A Better Way?

- We can guess the meaning of some unknown words
  - soooooooo
  - taaaasty
  - Transformerify
- Some words share the same prefix or suffix
  - happy, happier, happiest
  - drive, driving, driven
  - unlikely, unhappy, unhealthy
  - beautiful, trustful, grateful

Subword Tokenization

- We use subword (and punctuation) as the basic unit to tokenize a text
- Subword: parts of words
  - happy, happier, happiest: happ-, -y, -ier, -iest
  - drive, driving, driven: driv-, -e, -ing, -en
  - beautiful, trustful, grateful: -ful

Byte-Pair Encoding

- Byte-Pair Encoding (BPE) is a simple method to decide subword
  - Originally designed for compression
  - Use fewer subwords to cover more words
- Motivation: discover the most common pair of consecutive bytes of data
  - Start with a vocabulary containing only characters and a “end-of-word” symbol
  - Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary
  - Replace instances of the character pair with the new subword; repeat until desired vocabulary size

Byte-Pair Encoding Example

- Start with a vocabulary containing only characters and a "end-of-word" symbol

End-of-word symbol

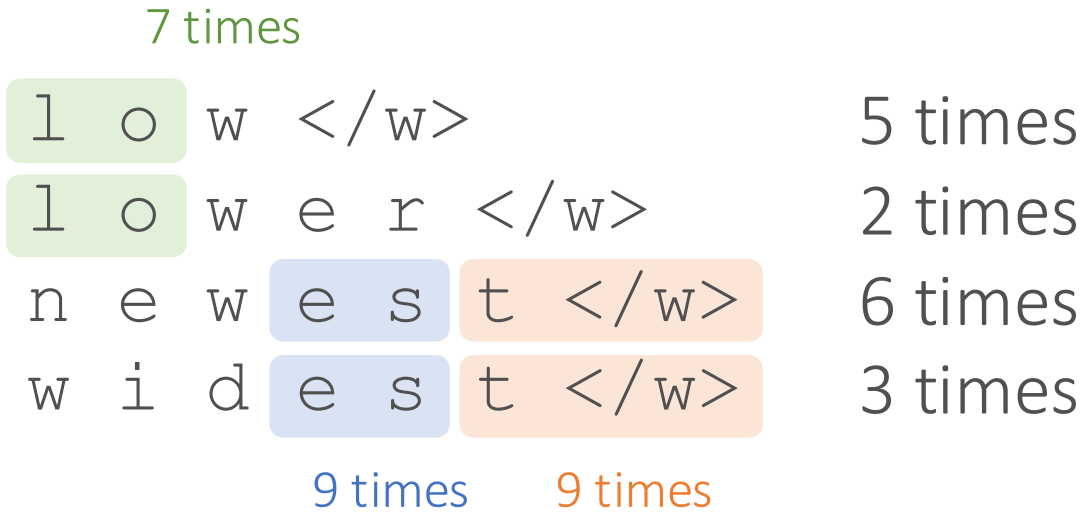
l o w </w>	5 times
l o w e r </w>	2 times
n e w e s t </w>	6 times
w i d e s t </w>	3 times

Vocabulary

</w> l o w e
r n s t i d

Byte-Pair Encoding Example

- Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary



Vocabulary

</w> l o w e
r n s t i d
es

Byte-Pair Encoding Example

- Replace instances of the character pair with the new subword

l o w </w>	5 times
l o w e r </w>	2 times
n e w es t </w>	6 times
w i d es t </w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es

Byte-Pair Encoding Example

- Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary

l o w </w>	5 times
l o w e r </w>	2 times
n e w es t </w>	6 times
w i d es t </w>	3 times

9 times

Vocabulary

</w> l o w e
r n s t i d
es est

Byte-Pair Encoding Example

- Replace instances of the character pair with the new subword

l o w </w>	5 times
l o w e r </w>	2 times
n e w est </w>	6 times
w i d est </w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est

Byte-Pair Encoding Example

- Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary

l o w </w>	5 times
l o w e r </w>	2 times
n e w est </w>	6 times
w i d est </w>	3 times

9 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>

Byte-Pair Encoding Example

- Replace instances of the character pair with the new subword

l o w </w>	5 times
l o w e r </w>	2 times
n e w est</w>	6 times
w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>

Byte-Pair Encoding Example

- Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary

7 times	
l o w </w>	5 times
l o w e r </w>	2 times
n e w est</w>	6 times
w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo

Byte-Pair Encoding Example

- Replace instances of the character pair with the new subword

lo w </w>	5 times
lo w e r </w>	2 times
n e w est</w>	6 times
w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo

Byte-Pair Encoding Example

- Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary

7 times

lo w </w>	5 times
lo w e r </w>	2 times
n e w est</w>	6 times
w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo low

Byte-Pair Encoding Example

- Replace instances of the character pair with the new subword

low </w>	5 times
low e r </w>	2 times
n e w est</w>	6 times
w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo low

Byte-Pair Encoding Example

- Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary

	low </w>	5 times
	low e r </w>	2 times
6 times	n e w est</w>	6 times
	w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo low ne

Byte-Pair Encoding Example

- Replace instances of the character pair with the new subword

low </w>	5 times
low e r </w>	2 times
ne w est</w>	6 times
w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo low ne

Byte-Pair Encoding Example

- Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary

	low </w>	5 times
	low e r </w>	2 times
6 times	ne w est</w>	6 times
	w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo low ne new

Byte-Pair Encoding Example

- Replace instances of the character pair with the new subword

low </w>	5 times
low e r </w>	2 times
new est</w>	6 times
w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo low ne new

Byte-Pair Encoding Example

- Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary

	low </w>	5 times
	low e r </w>	2 times
6 times	new est</w>	6 times
	w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo low ne new
newest</w>

Byte-Pair Encoding Example

- Replace instances of the character pair with the new subword

low </w>	5 times
low e r </w>	2 times
newest</w>	6 times
w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo low ne new
newest</w>

Byte-Pair Encoding Example

- Find the most common pair of adjacent characters “x” and “y”; add subword “xy” to the vocabulary

5 times	low </w>	5 times
	low e r </w>	2 times
	newest</w>	6 times
	w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo low ne new
newest</w>
low</w>

Byte-Pair Encoding Example

- Replace instances of the character pair with the new subword

low</w>	5 times
low e r </w>	2 times
newest</w>	6 times
w i d est</w>	3 times

Vocabulary

</w> l o w e
r n s t i d
es est est</w>
lo low ne new
newest</w>
low</w>

Byte-Pair Encoding Example

MERGES

```
e + s => es
es + t => est
est + </w> => est</w>
l + o => lo
lo + w => low
n + e => ne
ne + w => new
new + est</w> => newest</w>
low + </w> => low</w>
```

Vocabulary

```
</w> l o w e
r n s t i d
es est est</w>
lo low ne new
newest</w>
low</w>
```

Byte-Pair Encoding Example

MERGES

```
e + s => es
es + t => est
est + </w> => est</w>
l + o => lo
lo + w => low
n + e => ne
ne + w => new
new + est</w> => newest</w>
low + </w> => low</w>
```

Vocabulary

```
</w> l o w e
r n s t i d
es est est</w>
lo low ne new
newest</w>
low</w>
```

New unseen token: lowest → `low est</w>`

Byte-Pair Encoding Example

MERGES

```
e + s => es
es + t => est
est + </w> => est</w>
l + o => lo
lo + w => low
n + e => ne
ne + w => new
new + est</w> => newest</w>
low + </w> => low</w>
```

Vocabulary

```
</w> l o w e
r n s t i d
es est est</w>
lo low ne new
newest</w>
low</w>
```

New unseen token: powest → `<UNK> o w est</w>`

Subword Tokenization

- We use subword (and punctuation) as the basic unit to tokenize a text
- Subword: parts of words
  - happy, happier, happiest: happ-, -y, -ier, -iest
  - drive, driving, driven: driv-, -e, -ing, -en
  - beautiful, trustful, grateful: -ful
- A more effective way to construct vocabulary

Lecture Plan

- Tokenization
  - Subwords
  - Byte-Pair Encoding
- Language Models
  - Definition of Language Models
  - N-Gram Language Models
  - Language Model Decoding Methods
  - Neural Language Models

What are Language Models?

- A probabilistic model of a sequence of words
  - Evaluate the probability of whether a text is acceptable
- How likely are the following sentences?

The dog is barking at the stranger in the yard.

Yesterday, I went to the park and saw a group of children playing soccer.

The sky colorful because painted an artist.

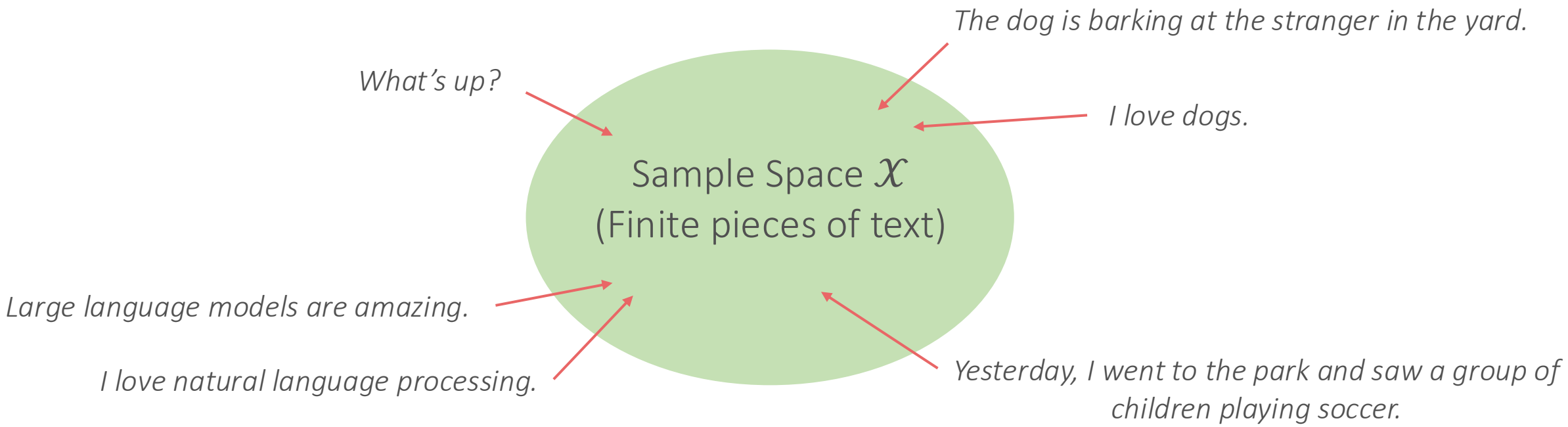
Cats upon they chairs sleeping their dreams fall.

Plorp zix flanned the quibble through treemunk.

Language Models

- Learn the probability distribution over texts $x = [w_1, w_2, \ldots, w_l] \in \mathcal{X}$

$$P(x) = P(w_1, w_2, \ldots, w_l)$$



What Can Language Models Do?

- Score texts

  $P(\textit{The dog is barking at the stranger in the yard.})$ → High

  $P(\textit{Cats upon they chairs sleeping their dreams fall.})$ → Low

- Generate texts

$$\tilde{x} \sim P(\mathcal{X})$$

Auto-Regressive Language Models

$$P(w_1, w_2, w_3, \dots, w_l) = P(w_1)P(w_2, w_3, \dots, w_l|w_1)$$
$$= P(w_1)P(w_2|w_1)(w_3, \dots, w_l|w_1, w_2)$$
$$= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)(w_4, \dots, w_l|w_1, w_2, w_3)$$
$$= \prod_{i=1}^{l} P(w_i|w_1, w_2, \dots, w_{i-1})$$

$P(\textit{She likes to go hiking}) = P(\textit{She}) \cdot P(\text{likes}|\text{She}) \cdot P(\text{to}|\text{She likes})$
$\cdot P(\text{go}|\text{She likes to}) \cdot P(\text{hiking}|\text{She likes to go})$

Auto-Regressive Language Models

$$P(w_1, w_2, w_3, \dots, w_l) = \prod_{i=1}^{l} P(w_i | w_1, w_2, \dots, w_{i-1})$$

Next Token

Context

Next token prediction problem based on context

Challenge: How to predict $P(w_i | w_1, w_2, \dots, w_{i-1})$?

Unigram Language Models

Assumption: $P(w_i|w_1, w_2, \ldots, w_{i-1}) \approx P(w_i)$

$$P(w_1, w_2, w_3, \ldots, w_l) \approx P(w_1)P(w_2)P(w_3) \ldots P(w_l)$$

Similar to the concept of bag-of-words!

How to calculate $P(w_i)$?

A count-based solution: Collect training corpus and count

$$P(w_i) = \frac{C_{train}(w_i)}{\sum_t C_{train}(w_t)}$$

Bigram Language Models

Assumption: $P(w_i|w_1, w_2, \ldots, w_{i-1}) \approx P(w_i|w_{i-1})$

$$P(w_1, w_2, w_3, \ldots, w_l) \approx P(w_1)P(w_2|w_1)P(w_3|w_2)P(w_4|w_3) \ldots P(w_l|w_{l-1})$$

The prediction of the next token depends only on the previous token

A count-based solution: Collect training corpus and count

$$P(w_i|w_{i-1}) = \frac{C_{train}(w_{i-1}, w_i)}{C_{train}(w_{i-1})}$$

Trigram Language Models

Assumption: $P(w_i|w_1, w_2, \dots, w_{i-1}) \approx P(w_i|w_{i-2}, w_{i-1})$

$$P(w_1, w_2, w_3, \dots, w_l) \approx P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)P(w_4|w_2, w_3) \dots P(w_l|w_{l-2}, w_{l-1})$$

The prediction of the next token depends on the previous **two** tokens

A count-based solution: Collect training corpus and count

$$P(w_i|w_{i-1}, w_{i-2}) = \frac{C_{train}(w_{i-2}, w_{i-1}, w_i)}{C_{train}(w_{i-2}, w_{i-1})}$$

N-Gram Language Models

Assumption: $P(w_i|w_1, w_2, \ldots, w_{i-1}) \approx P(w_i|w_{i-n+1}, \ldots, w_{i-1})$

$$P(w_1, w_2, w_3, \ldots, w_l) \approx \prod_i P(w_i|w_{i-n+1}, \ldots, w_{i-1})$$

The prediction of the next token depends on the previous **n** tokens

A count-based solution: Collect training corpus and count

$$P(w_i|w_{i-n+1}, \ldots, w_{i-1}) = \frac{C_{train}(w_{i-n+1}, \ldots, w_{i-1}, w_i)}{C_{train}(w_{i-n+1}, \ldots, w_{i-1})}$$

How to Evaluate Language Models?

- A good language model should assign higher probability to typical, grammatically correct sentences
- Train a language model on a suitable training corpus
  - Assumption: observed sentences ≈ good sentences
- Test on different, unseen corpus
  - The higher probability that the language model assigns to the test set, the better (why?)
- Evaluation metric: Perplexity

Perplexity (PPL)

For a corpus $\mathcal{X}$ with sentences $\{x_1, x_2, \ldots, x_n\}$

Likelihood $$P(\mathcal{X}) = \prod_{i=1}^{n} P(x_i)$$ The higher, the better

Log-Likelihood $$\log P(\mathcal{X}) = \sum_{i=1}^{n} \log P(x_i)$$ The higher, the better

Per-Word Log-Likelihood $$WLL(\mathcal{X}) = \frac{1}{W} \sum_{i=1}^{n} \log P(x_i)$$ The higher, the better

The number of words in the test corpus

Perplexity (PPL)

For a corpus $\mathcal{X}$ with sentences $\{x_1, x_2, \dots, x_n\}$

Perplexity $e^{-WLL(\mathcal{X})}$

The lower, the better

$$WLL(\mathcal{X}) = \frac{1}{W}\sum_{i=1}^{n} \log P(x_i)$$

Computed by language model

Unigram Language Model

$$P(w_j | w_1, w_2, \dots, w_{j-1}) \approx P(w_j)$$

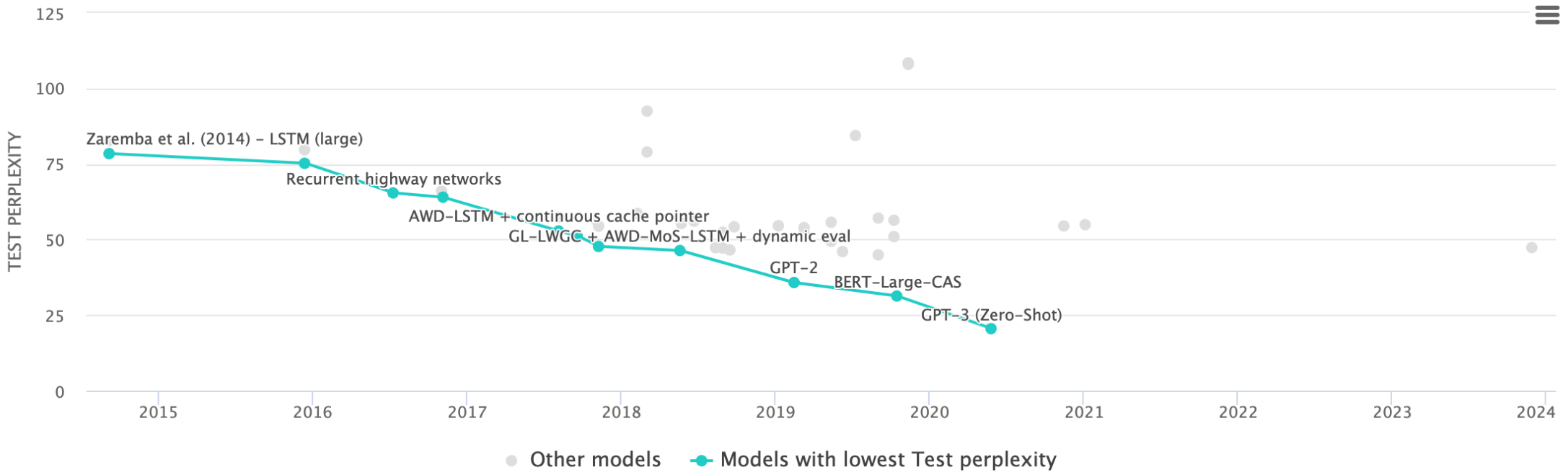
$$P(x) = \prod_j P(w_j)$$

Unigram Language Model

$$WLL(\mathcal{X}) = \frac{1}{W}\sum_i \sum_j \log P(w_{i,j})$$

Minimizing perplexity → maximizing probability of corpus

Perplexity (PPL)



https://paperswithcode.com/sota/language-modelling-on-penn-treebank-word

Text Generation with Language Models

Trigram Language Model

$$P(w_1, w_2, w_3, \dots, w_l) \approx \prod_i P(w_i | w_{i-2}, w_{i-1})$$

- Generate the first word $w_1 \sim P(w)$
- Generate the second word $w_2 \sim P(w|w_1)$
- Generate the third word $w_3 \sim P(w|w_1, w_2)$
- Generate the fourth word $w_4 \sim P(w|w_2, w_3)$
- Generate the fifth word $w_5 \sim P(w|w_3, w_4)$
- …
- Until the end of the sentence <eos>

Generation Examples

Unigram

release millions See ABC accurate President of Donald Will cheat them a CNN megynkelly experience @ these word out- the

Bigram

Thank you believe that @ ABC news, Mississippi tonight and the false editorial I think the great people Bill Clinton . ''

Trigram

We are going to MAKE AMERICA GREAT AGAIN! #MakeAmericaGreatAgain https: //t.co/DjkdAzT3WV

Typical LMs are not sufficient to handle long-range dependencies

"Alice/Bob could not go to work that day because she/he had a doctor's appointment"

Different Decoding Strategies

Random Sampling

$$w_3 \sim P(w|w_1, w_2)$$

Greedy Decoding

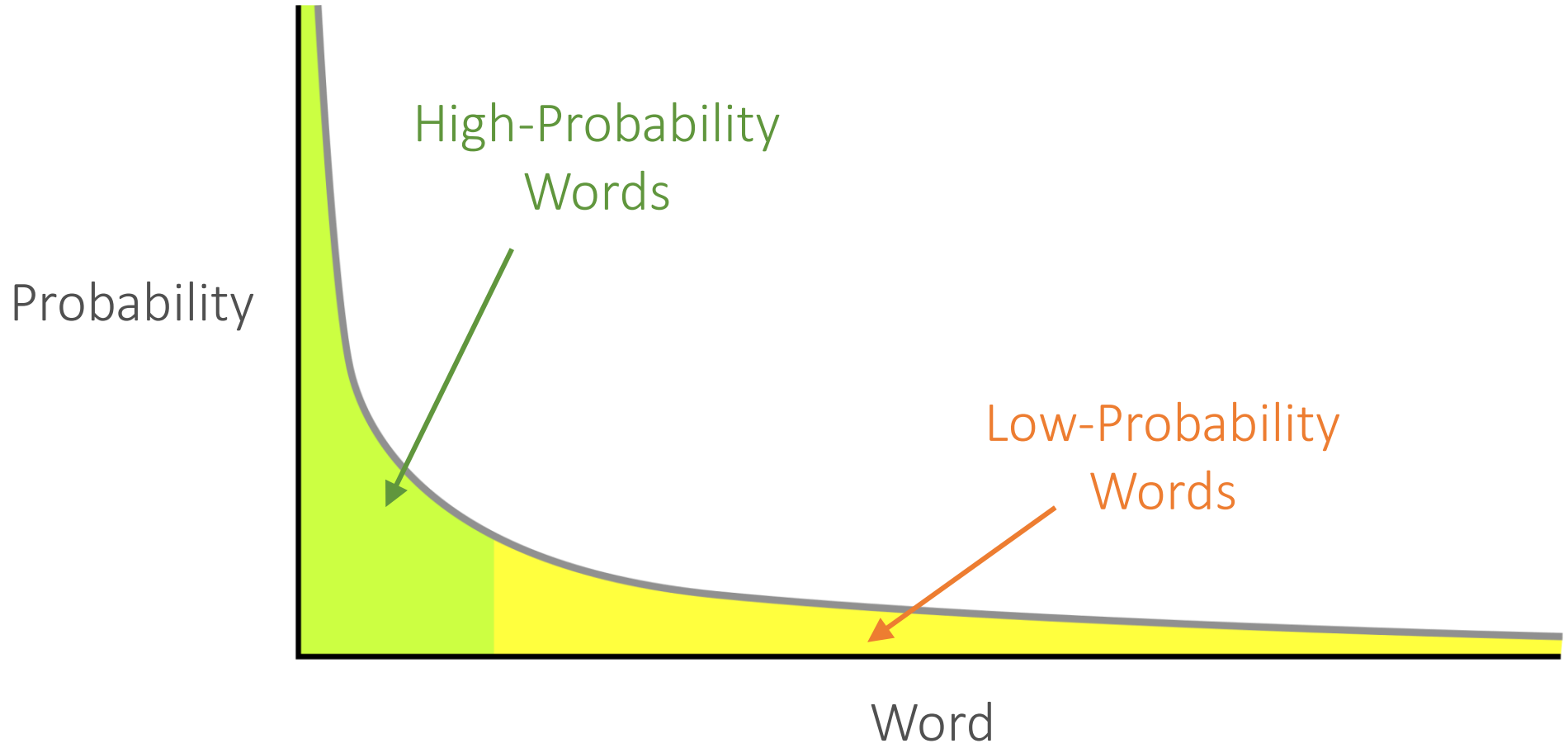
$$w_3 = \arg\max_{w \in \mathcal{V}} P(w|w_1, w_2)$$

Temperature Scaling

$$P(w_i) = \text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

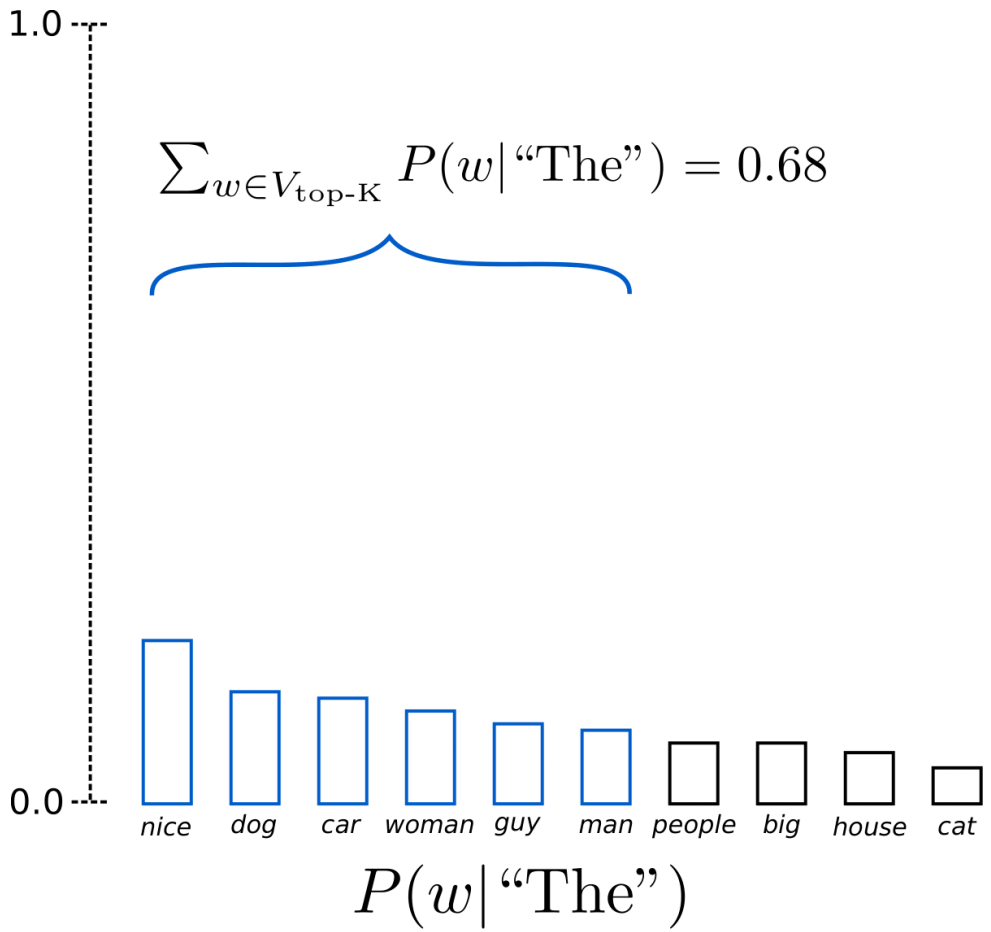
$$P(w_i) = \text{softmax}(z_i) = \frac{e^{\frac{z_i}{T}}}{\sum_j e^{\frac{z_j}{T}}}$$

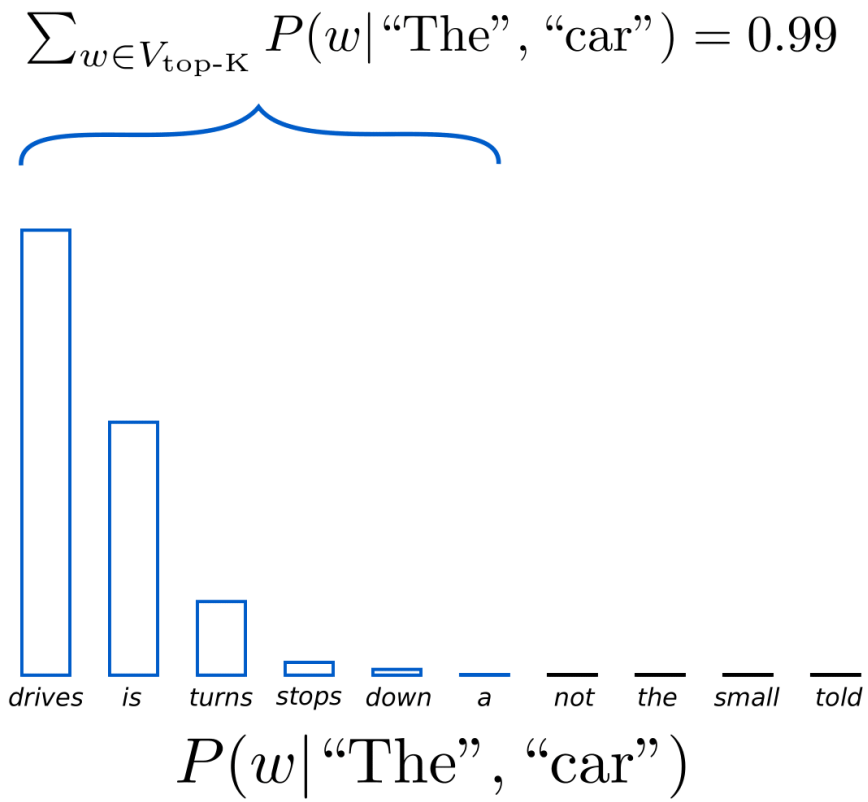
Long-Tail Word Distribution



https://en.wikipedia.org/wiki/Long_tail

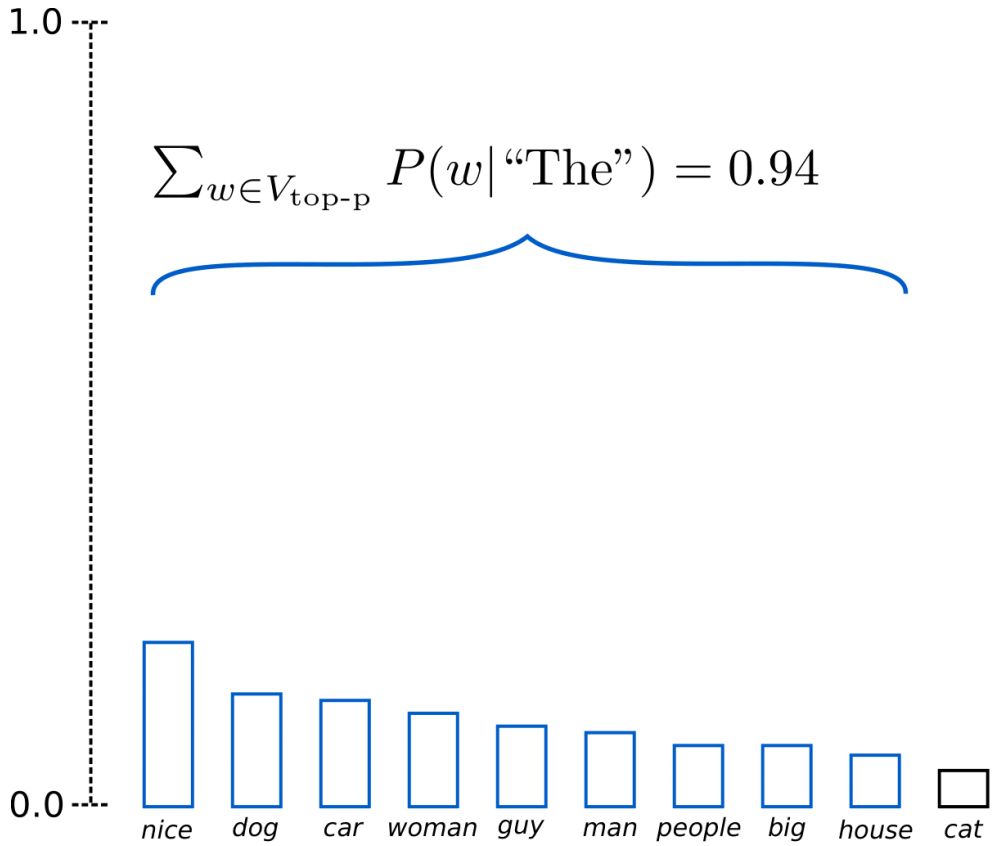
Top-K Sampling

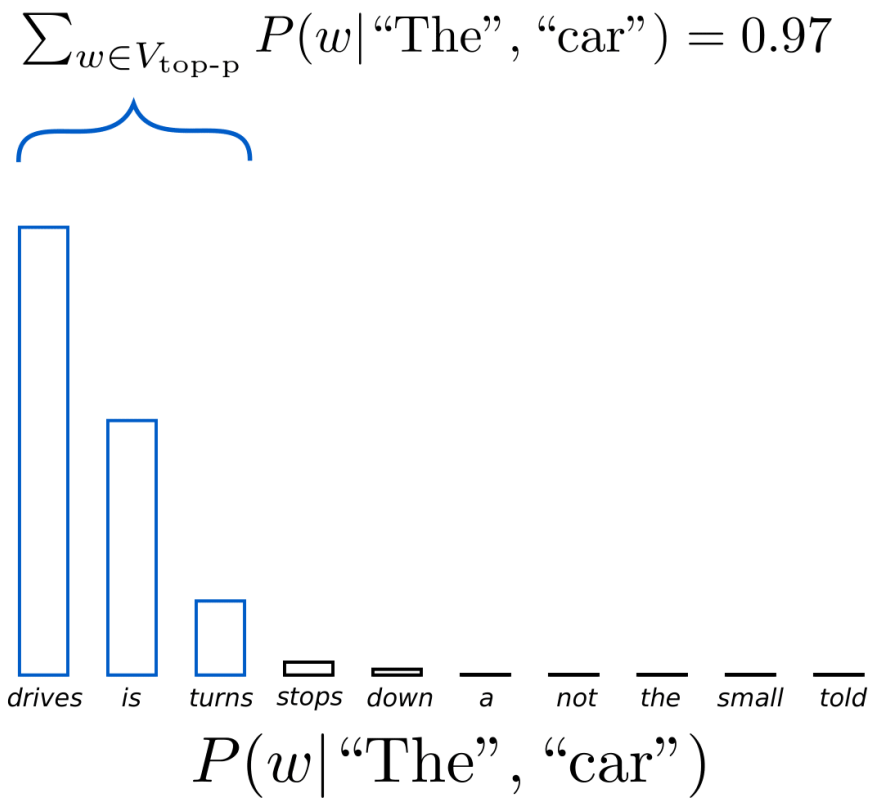




https://huggingface.co/blog/how-to-generate

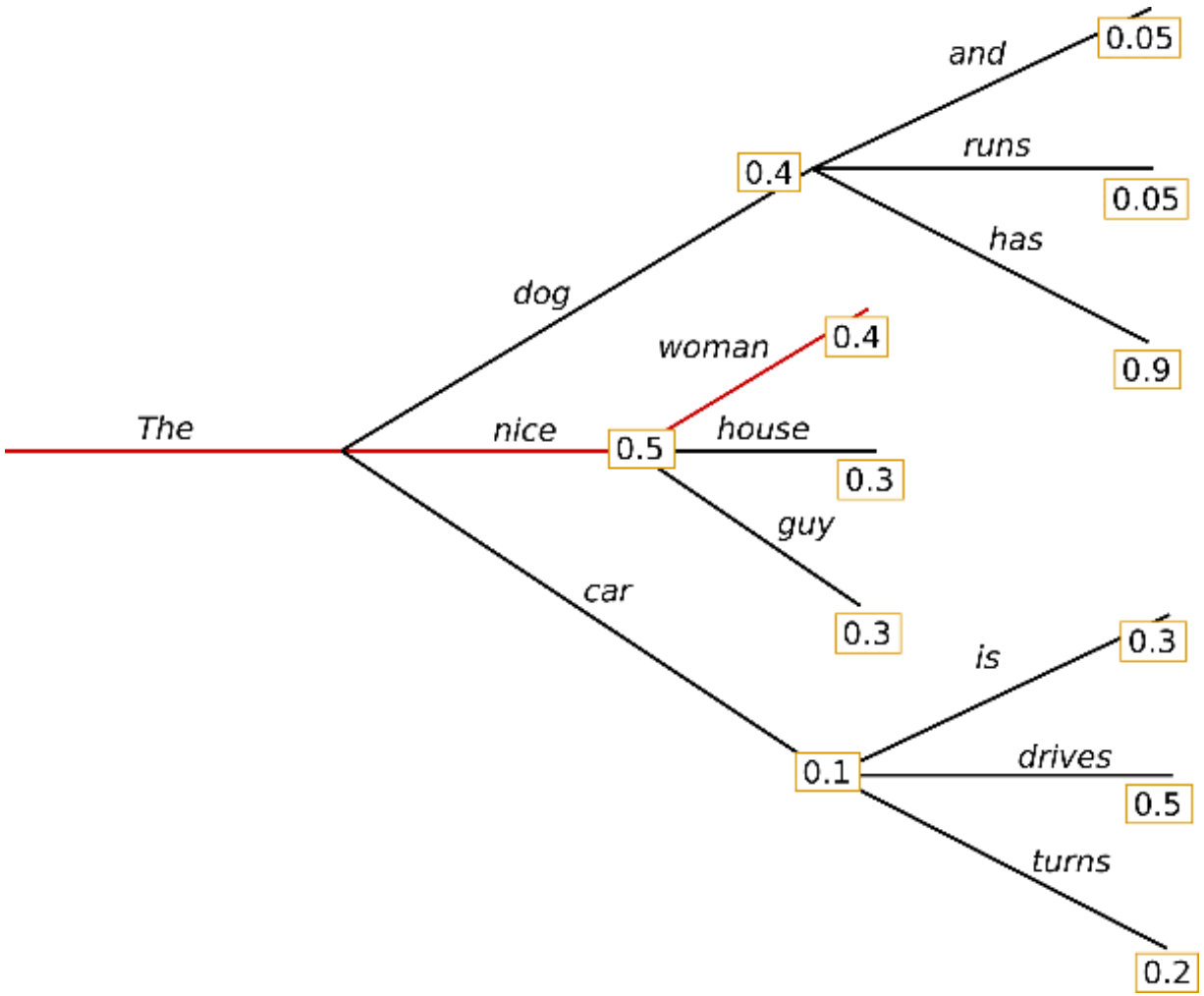
Top-P Sampling





https://huggingface.co/blog/how-to-generate

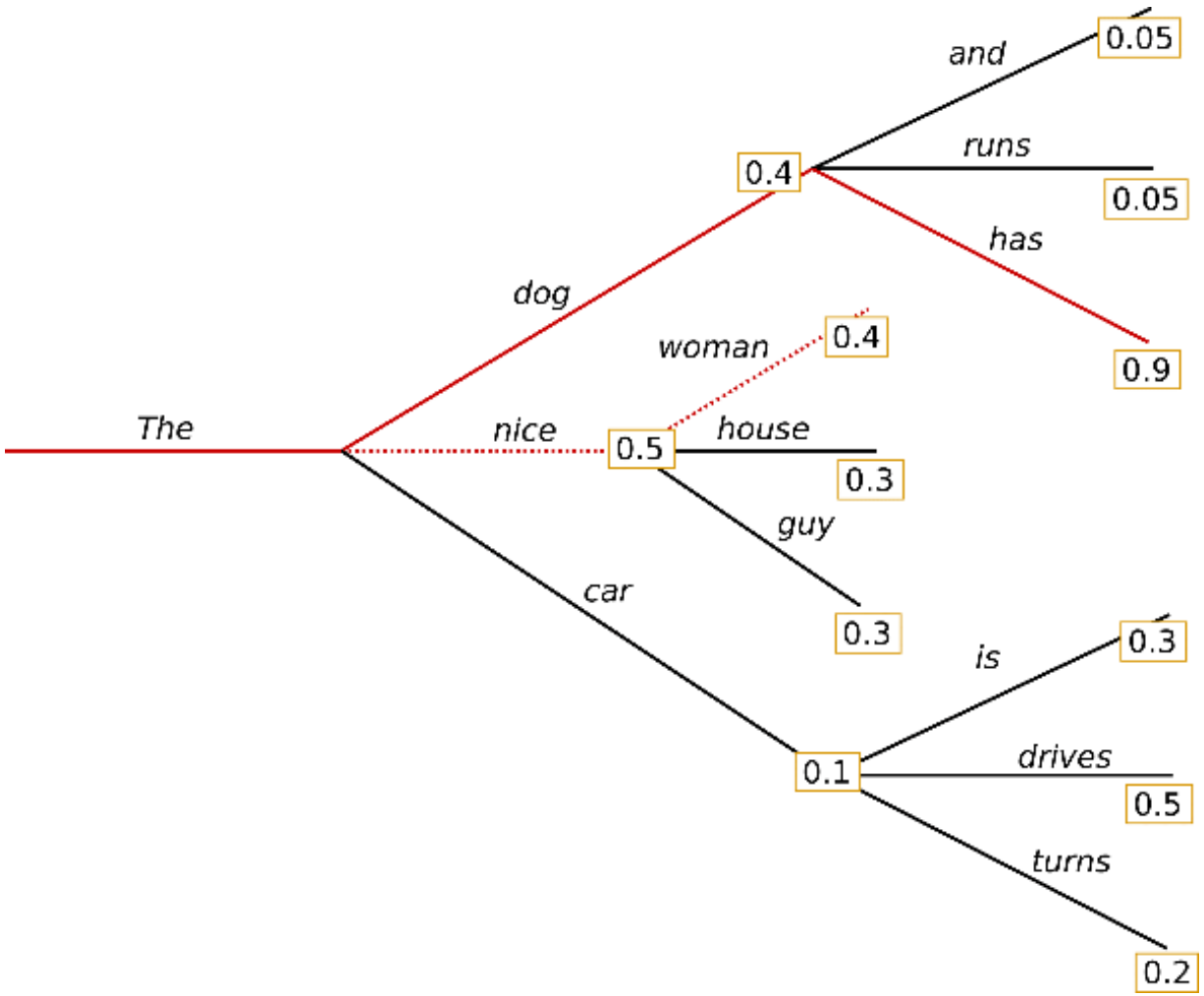
Beam Search



Sequential sampling does not always lead to the optimal sequence

https://huggingface.co/blog/how-to-generate

Beam Search

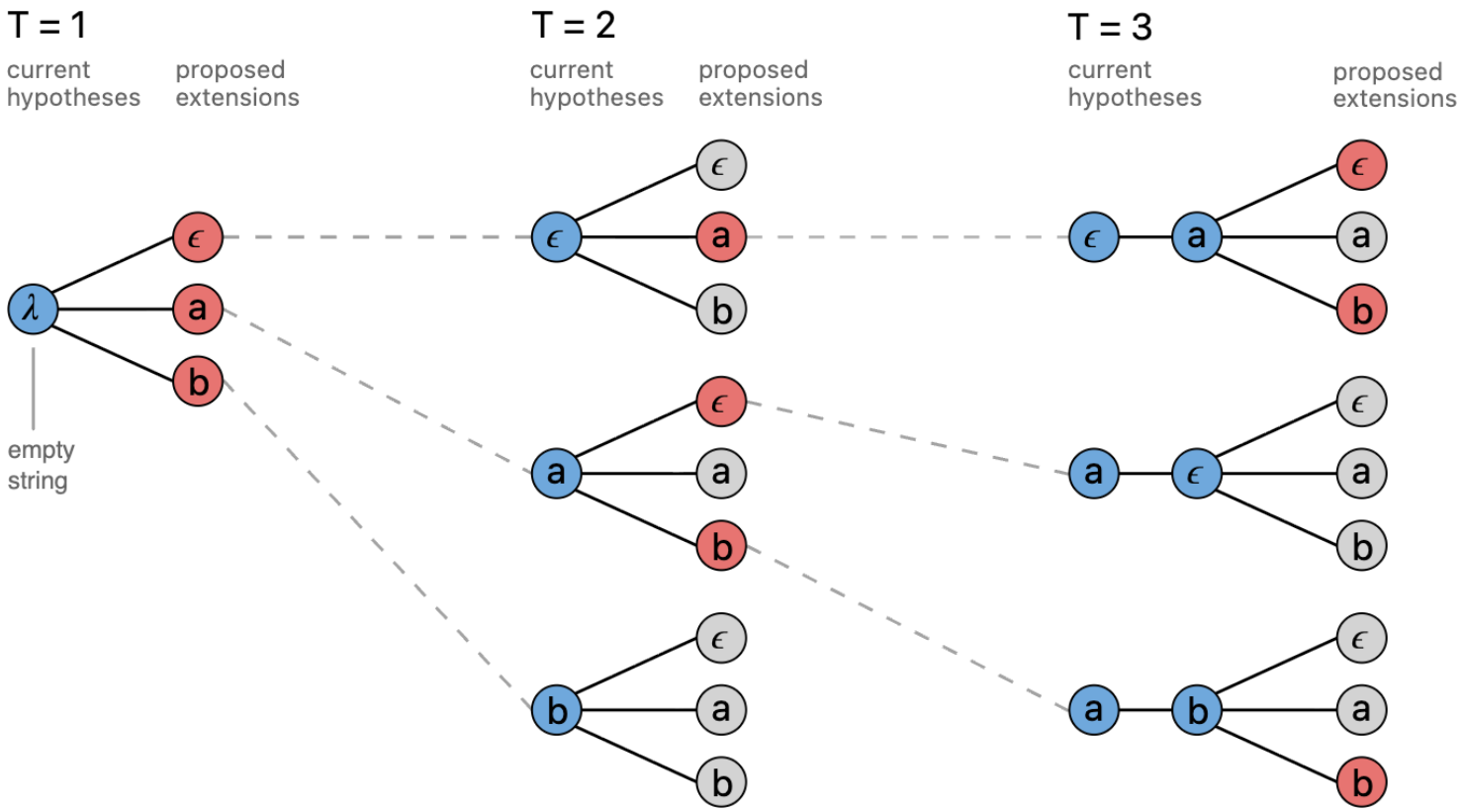


Sequential sampling does not always lead to the optimal sequence

https://huggingface.co/blog/how-to-generate

Beam Search

- Beam size m: Keep m best paths in the queue



https://distill.pub/2017/ctc/

What Can Language Models Do?

- Score texts

$P(\textit{The dog is barking at the stranger in the yard.}) \rightarrow$ High

$P(\textit{Cats upon they chairs sleeping their dreams fall.}) \rightarrow$ Low

Compute perplexity

- Generate texts

$$\tilde{x} \sim P(\mathcal{X})$$

Sample from word distribution

N-Gram Language Models

Assumption: $P(w_i|w_1, w_2, \dots, w_{i-1}) \approx P(w_i|w_{i-n+1}, \dots, w_{i-1})$

$$P(w_1, w_2, w_3, \dots, w_l) \approx \prod_i P(w_i|w_{i-n+1}, \dots, w_{i-1})$$

The prediction of the next token depends on the previous **n** tokens

A count-based solution: Collect training corpus and count

$$P(w_i|w_{i-n+1}, \dots, w_{i-1}) = \frac{C_{train}(w_{i-n+1}, \dots, w_{i-1}, w_i)}{C_{train}(w_{i-n+1}, \dots, w_{i-1})}$$

Any Problems?

Unseen Patterns in Training Corpus

- Not all n-grams in the test set will be observed in training corpus
- Training corpus
  - I like apples
  - I love bananas
- Test set
  - I like bananas
  - I love apples
- This problem becomes severe when n is large

Laplace Smoothing

- Handle sparsity by making sure all probabilities are non-zero in our model
  - Just add α to all counts and renormalize

Bigram language model before smoothing

$$P(w_i|w_{i-1}) = \frac{C_{train}(w_{i-1}, w_i)}{C_{train}(w_{i-1})}$$

Bigram language model after smoothing

$$P(w_i|w_{i-1}) = \frac{C_{train}(w_{i-1}, w_i) + \alpha}{C_{train}(w_{i-1}) + \alpha|\mathcal{V}|}$$

Linear Interpolation

$$\hat{P}(w_i|w_{i-1}, w_{i-2}) = \lambda_1 P(w_i|w_{i-1}, w_{i-2}) \quad \text{Trigram}$$

$$+\lambda_2 P(w_i|w_{i-1}) \quad \text{Bigram}$$

$$+\lambda_3 P(w_i) \quad \text{Unigram}$$

$$\sum_i \lambda_i = 1$$

Strong empirical performance!

N-Gram Language Models

Assumption: $P(w_i|w_1, w_2, \dots, w_{i-1}) \approx P(w_i|w_{i-n+1}, \dots, w_{i-1})$

$$P(w_1, w_2, w_3, \dots, w_l) \approx \prod_i P(w_i|w_{i-n+1}, \dots, w_{i-1})$$

The prediction of the next token depends on the previous **n** tokens

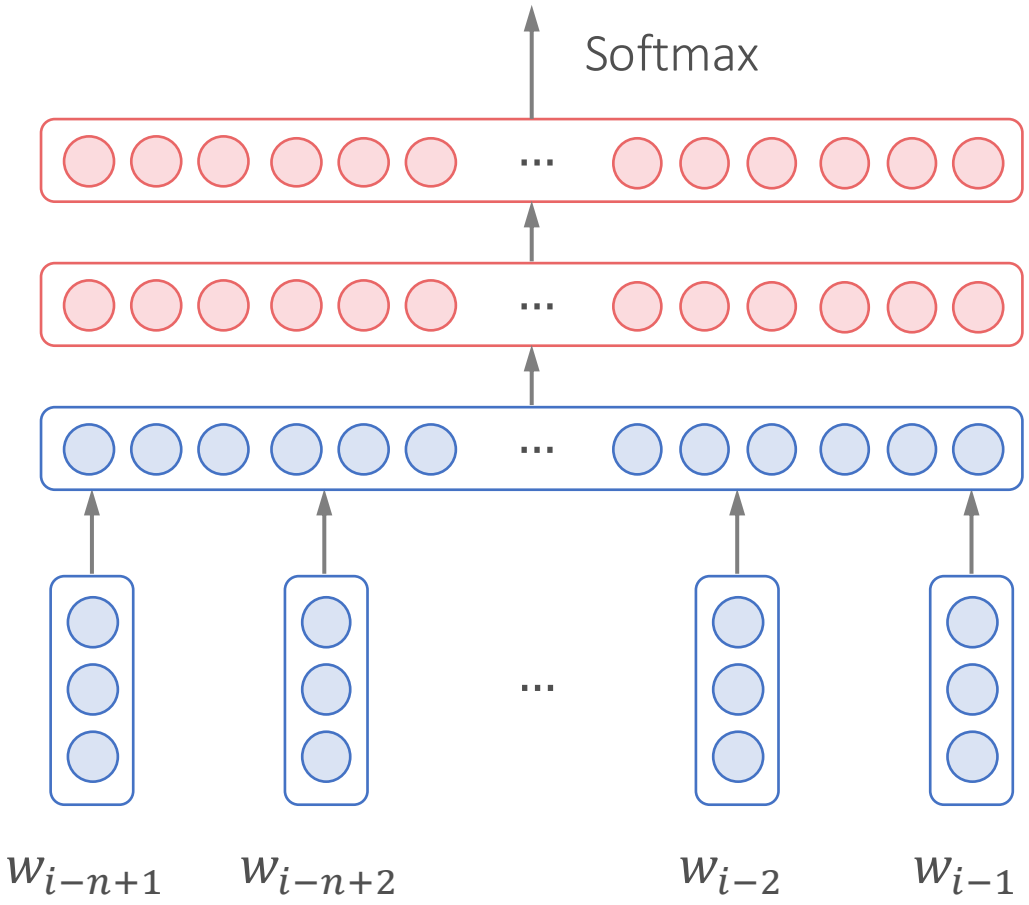
A count-based solution: Collect training corpus and count

$$P(w_i|w_{i-n+1}, \dots, w_{i-1}) = \frac{C_{train}(w_{i-n+1}, \dots, w_{i-1}, w_i)}{C_{train}(w_{i-n+1}, \dots, w_{i-1})}$$

Can we compute the probability in a different way?

Neural Language Models

$$P(w_i|w_{i-n+1}, \cdots, w_{i-1})$$



Training corpus

- I like apples
- I love bananas

Test set

- I love apples
- I like bananas

Auto-Regressive Language Models

$$P(w_1, w_2, w_3, \dots, w_l) = P(w_1)P(w_2, w_3, \dots, w_l|w_1)$$
$$= P(w_1)P(w_2|w_1)(w_3, \dots, w_l|w_1, w_2)$$
$$= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)(w_4, \dots, w_l|w_1, w_2, w_3)$$
$$= \prod_{i=1}^{l} P(w_i|w_1, w_2, \dots, w_{i-1})$$

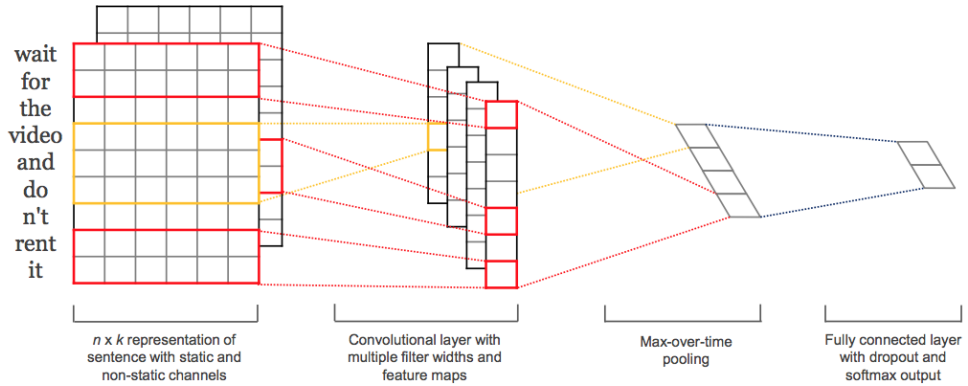
$P(\textit{She likes to go hiking}) = P(\textit{She}) \cdot P(\text{likes}|\text{She}) \cdot P(\text{to}|\text{She likes})$
$\cdot P(\text{go}|\text{She likes to}) \cdot P(\text{hiking}|\text{She likes to go})$

Lecture Plan

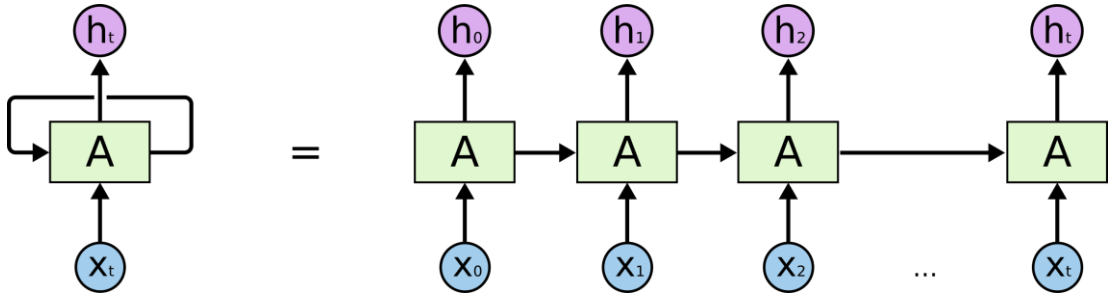
- Tokenization
  - Subwords
  - Byte-Pair Encoding
- Language Models
  - Definition of Language Models
  - N-Gram Language Models
  - Language Model Decoding Methods
  - Neural Language Models

Next Lecture

The dog bites the man ≠ The man bites the dog



Convolutional Neural Network



Recurrent Neural Network

https://colah.github.io/posts/2015-08-Understanding-LSTMs/
https://cezannec.github.io/CNN_Text_Classification/