# CSCE 638 Natural Language Processing Foundation and Techniques

## Lecture 4: Word Representations, Tokenization, and Language Modeling

Kuan-Hao Huang

Spring 2025

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

# Assignment 0

- https://khhuang.me/CSCE638-S25/assignments/assignment0_0122.pdf
- Due: 1/29/2025 11:59pm
- Summit a .zip file to Canvas
  - submission.pdf for the writing section
  - submission.py and submission.ipynb for the coding section
- For questions
  - Discuss on Canvas
  - Send an email to csce638-ta-25s@list.tamu.edu

# Assignment 1

- https://khhuang.me/CSCE638-S25/assignments/assignment1_0127.pdf
- Due: 2/17/2025 11:59pm
- Summit a .zip file to Canvas
  - submission.pdf for the writing section
  - submission4.py and submission4.ipynb for the problem 4
  - submission5.py and submission5.ipynb for the problem 5
- For questions
  - Discuss on Canvas
  - Send an email to csce638-ta-25s@list.tamu.edu

# Lecture Plan

- Word Representations
  - Variants
  - Evaluation
- Tokenization
  - Subwords
  - Byte-Pair Encoding
- Language Models
  - Definition of Language Models
  - N-Gram Language Models
  - Neural Language Models

# Recap: Words as Vectors

*Bob* *likes* *Alice* *very* *much*

$$W = \begin{bmatrix} | & | & | & | & | \\ w_{bob} & w_{likes} & w_{Alice} & w_{very} & w_{much} \\ | & | & | & | & | \end{bmatrix}$$

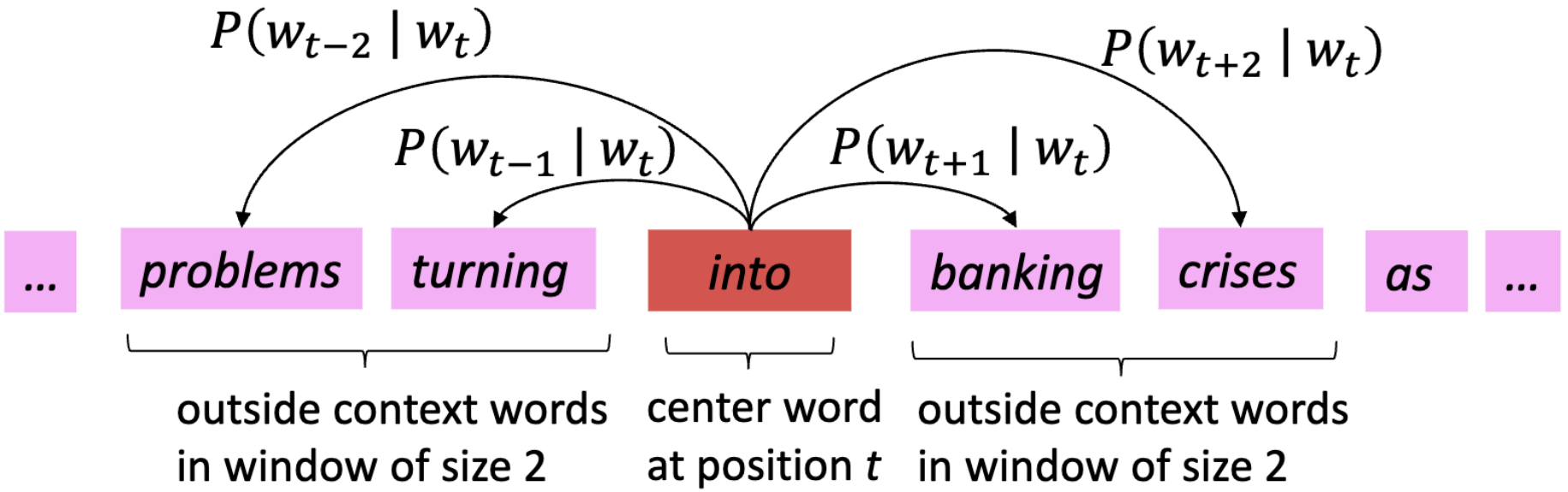
Use ***one vector*** to represent ***each word***

Text = A list of vectors

# 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 $w_{t+m}$




Words that occur in similar contexts tend to have similar meanings

# Recap: Probability for Predicting Context Wrod

We have two sets of vectors for each word in the vocabulary

$\mathbf{u}_w \in \mathbb{R}^d$: word vector when $w$ is a center word

$\mathbf{v}_w \in \mathbb{R}^d$: word vector when $w$ is a context word

$$P(w_{t+j} | w_t ; \theta) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

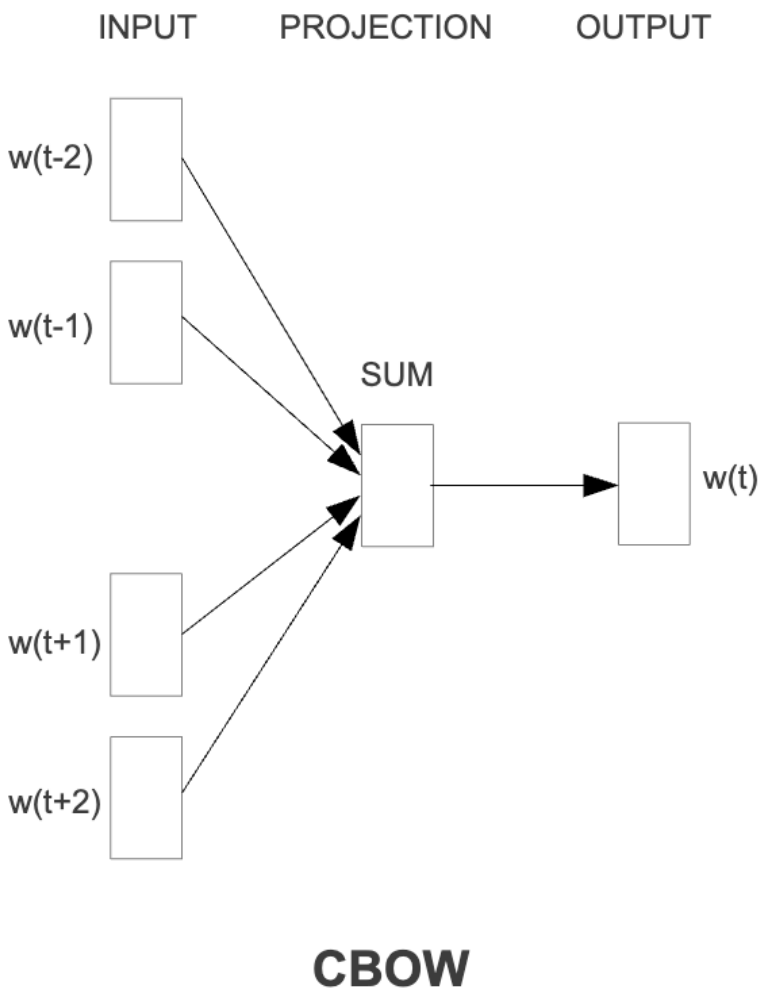
Normalize over entire vocabulary to give probability distribution

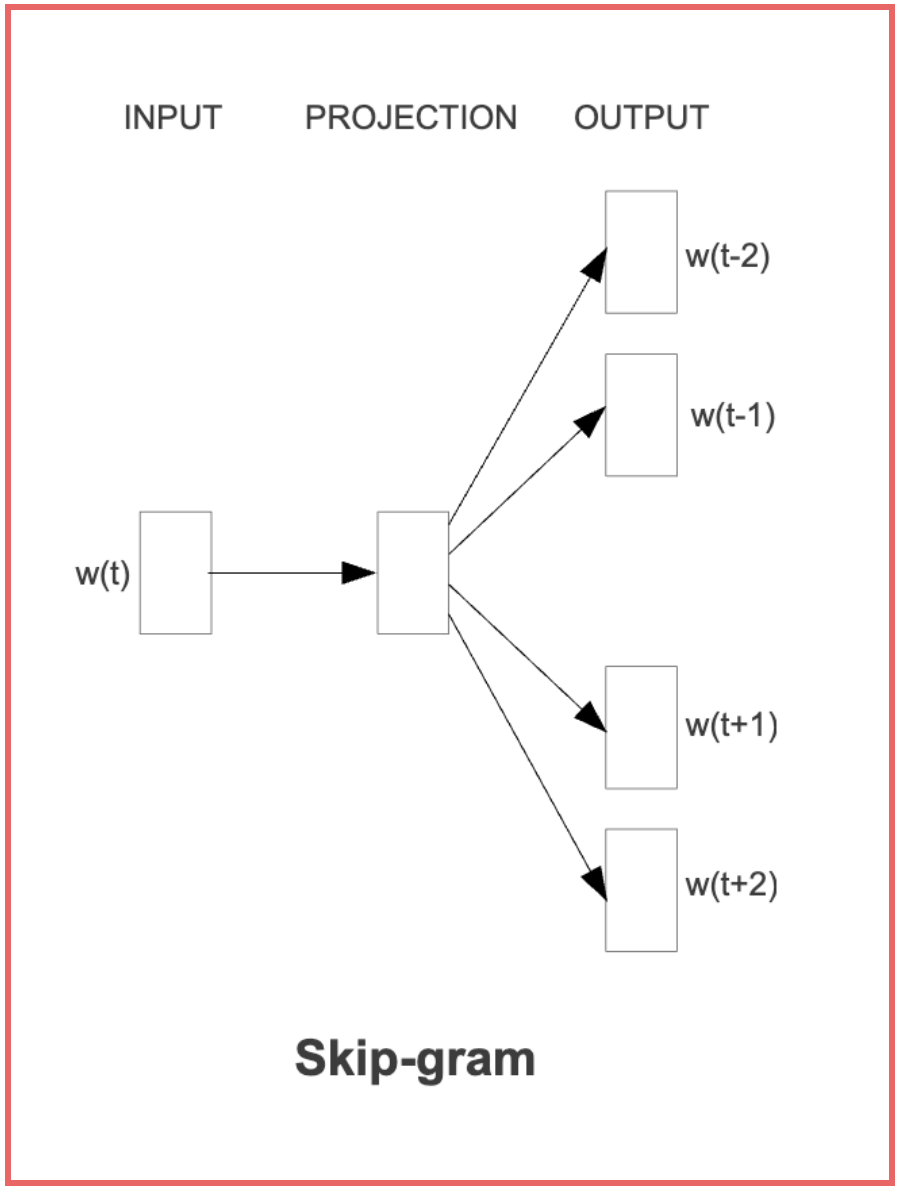
The score to indicate how likely the context word $w_{t+j}$ appears with the center word $w_t$

**Softmax function:** mapping arbitrary values to a probability distribution

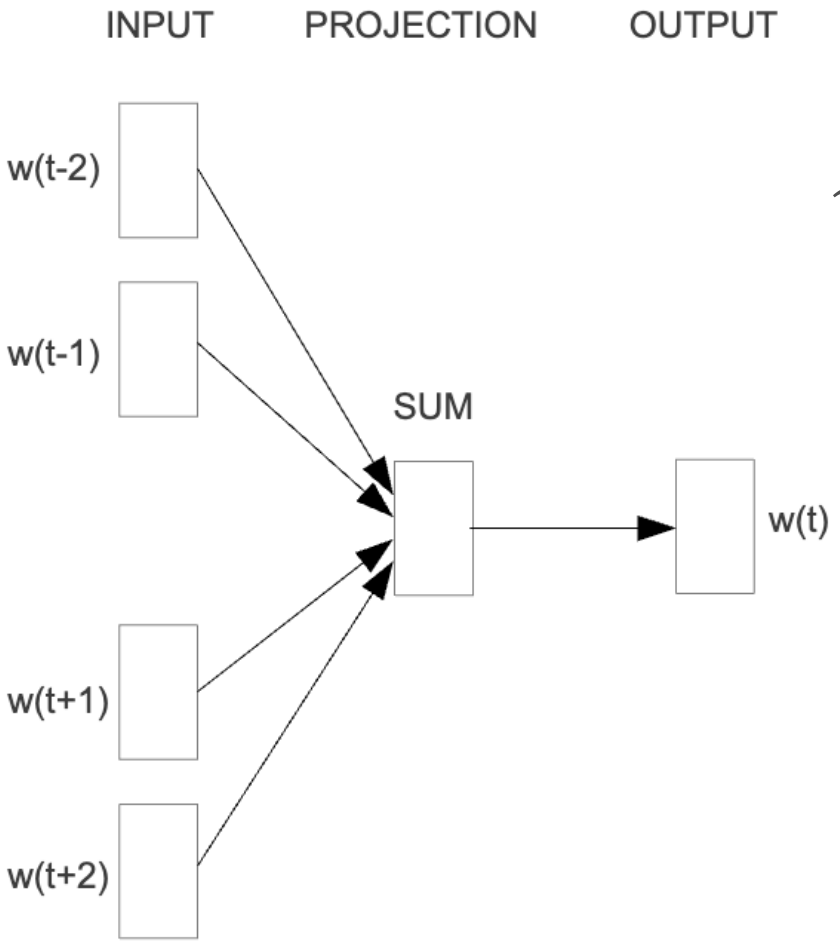
$$\text{softmax}(t) = \frac{e^t}{\sum_c e^c}$$

# Continuous Bag of Words (CBOW) vs Skip-Grams



# Continuous Bag of Words (CBOW)




$$\mathcal{L}(\theta) = \prod_{t=1}^{T} P(w_t | \{w_{t+j}\}), -m \leq j \leq m, j \neq 0$$

$$P(w_t | \{w_{t+j}\}) = \frac{\exp(\mathbf{u}_{w_t} \cdot \bar{\mathbf{v}}_t)}{\sum_{k \in V} \exp(\mathbf{u}_k \cdot \bar{\mathbf{v}}_t)}$$

$$\bar{\mathbf{v}}_t = \frac{1}{2m} \sum_{-m \leq j \leq m, j \neq 0} \mathbf{v}_{t+j}$$

# GloVe: Global Vectors

GloVe: Global Vectors for Word Representation (Pennington et al. 2014)

**Idea:** capture ratios of co-occurrence probabilities as linear meaning components in a word vector space

Log-bilinear model $w_i \cdot w_j = \log P(i|j)$

Vector difference $w_i \cdot (w_a - w_b) = \frac{\log P(x|a)}{\log P(x|b)}$

$$J = \sum_{i,j=1}^{V} f(X_{ij})\left(w_i^{\top}\tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

Global co-occurrence statistics

Training faster and scalable to very large corpora!

# FastText: Sub-Word Embeddings

Enriching Word Vectors with Subword Information (Bojanowski et al. 2017)

Similar as Skip-gram, but break words into n-grams with n = 3 to 6

where

3-grams: <wh, whe, her, ere, re>

4-grams: <whe, wher, here, ere>

5-grams: <wher, where, here>

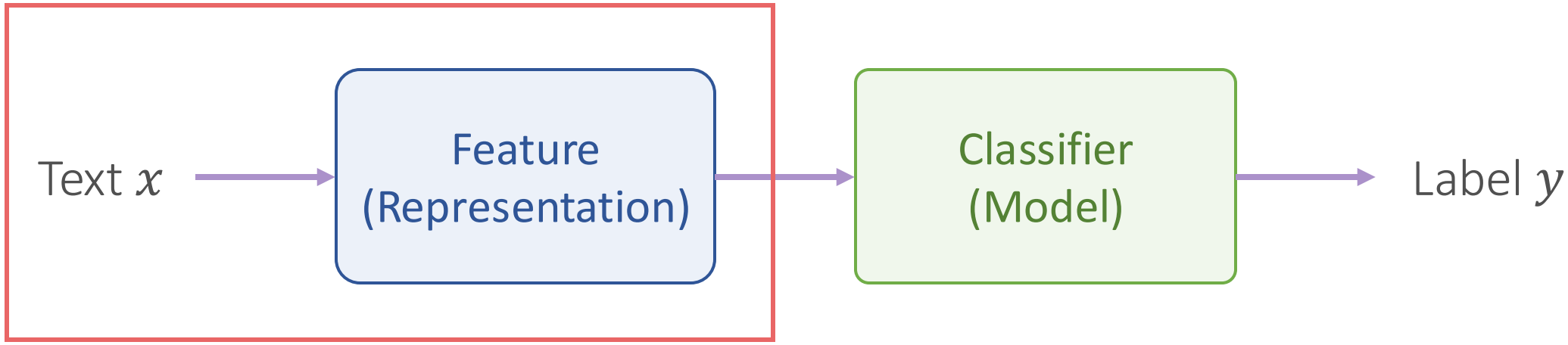
6-grams: <where, where>

Replace $\mathbf{u}_i \cdot \mathbf{v}_j$ with $\sum_{g \in n-grams(w_i)} \mathbf{u}_g \cdot \mathbf{v}_j$

# Trained Word Vectors Are Available

- Word2Vec: https://code.google.com/archive/p/word2vec/
- GloVe: https://nlp.stanford.edu/projects/glove/
- FastText: https://fasttext.cc/

# Prediction-Based Word Vectors




- Learn word vectors directly from text
  - Word2Vec (Skip-Gram and CBOW)
  - GloVe
  - FastText

# How to Evaluate the Quality of Word Embeddings?

- Intrinsic evaluation
  - Measures the quality of word embeddings by assessing their performance on specific linguistic or semantic tasks
- Extrinsic evaluation
  - Measures the quality of word embeddings by testing their impact on downstream and real-world tasks

# Intrinsic Evaluation: Word Similarity

**Word similarity**

Example dataset: wordsim-353
353 pairs of words with human judgement
http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/

| Word 1 | Word 2 | Human (mean) |
|---|---|---|
| tiger | cat | 7.35 |
| tiger | tiger | 10 |
| book | paper | 7.46 |
| computer | internet | 7.58 |
| plane | car | 5.77 |
| professor | doctor | 6.62 |
| stock | phone | 1.62 |
| stock | CD | 1.31 |
| stock | jaguar | 0.92 |

Cosine similarity:

$$\cos(\boldsymbol{u}_i, \boldsymbol{u}_j) = \frac{\boldsymbol{u}_i \cdot \boldsymbol{u}_j}{||\boldsymbol{u}_i||_2 \times ||\boldsymbol{u}_j||_2}.$$

Metric: Spearman rank correlation

# Intrinsic Evaluation: Word Similarity

| Model | Size | WS353 | MC | RG | SCWS | RW |
|---|---|---|---|---|---|---|
| SVD | 6B | 35.3 | 35.1 | 42.5 | 38.3 | 25.6 |
| SVD-S | 6B | 56.5 | 71.5 | 71.0 | 53.6 | 34.7 |
| SVD-L | 6B | 65.7 | 72.7 | 75.1 | 56.5 | 37.0 |
| CBOW† | 6B | 57.2 | 65.6 | 68.2 | 57.0 | 32.5 |
| SG† | 6B | 62.8 | 65.2 | 69.7 | 58.1 | 37.2 |
| GloVe | 6B | 65.8 | 72.7 | 77.8 | 53.9 | 38.1 |
| SVD-L | 42B | 74.0 | 76.4 | 74.1 | 58.3 | 39.9 |
| GloVe | 42B | **75.9** | **83.6** | **82.9** | **59.6** | **47.8** |
| CBOW* | 100B | 68.4 | 79.6 | 75.4 | 59.4 | 45.5 |

SG: Skip-Gram



GloVe: Global Vectors for Word Representation, EMNLP 2014

# Intrinsic Evaluation: Word Analogy

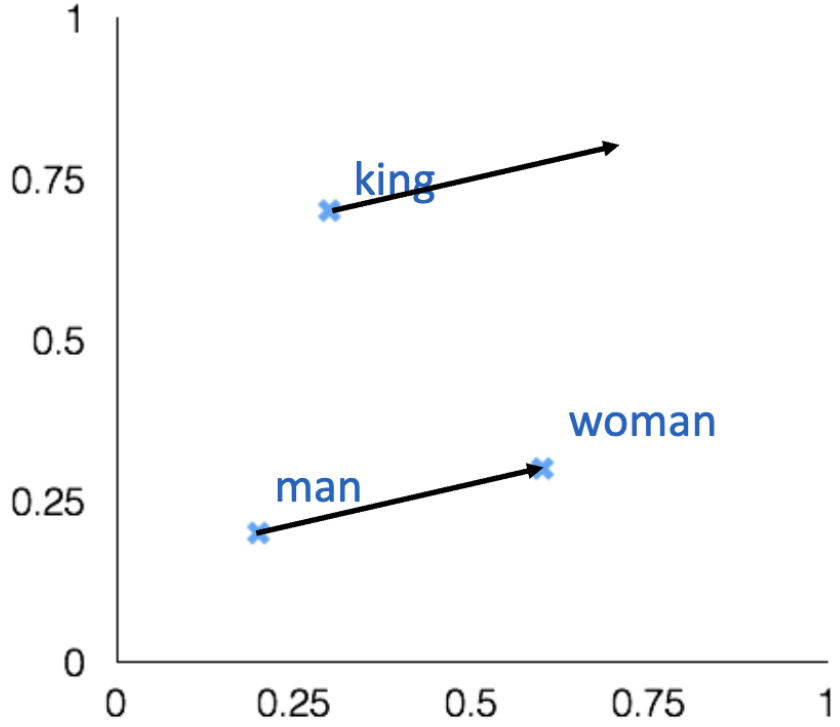
**Word analogy**

man: woman ≈ king: ?

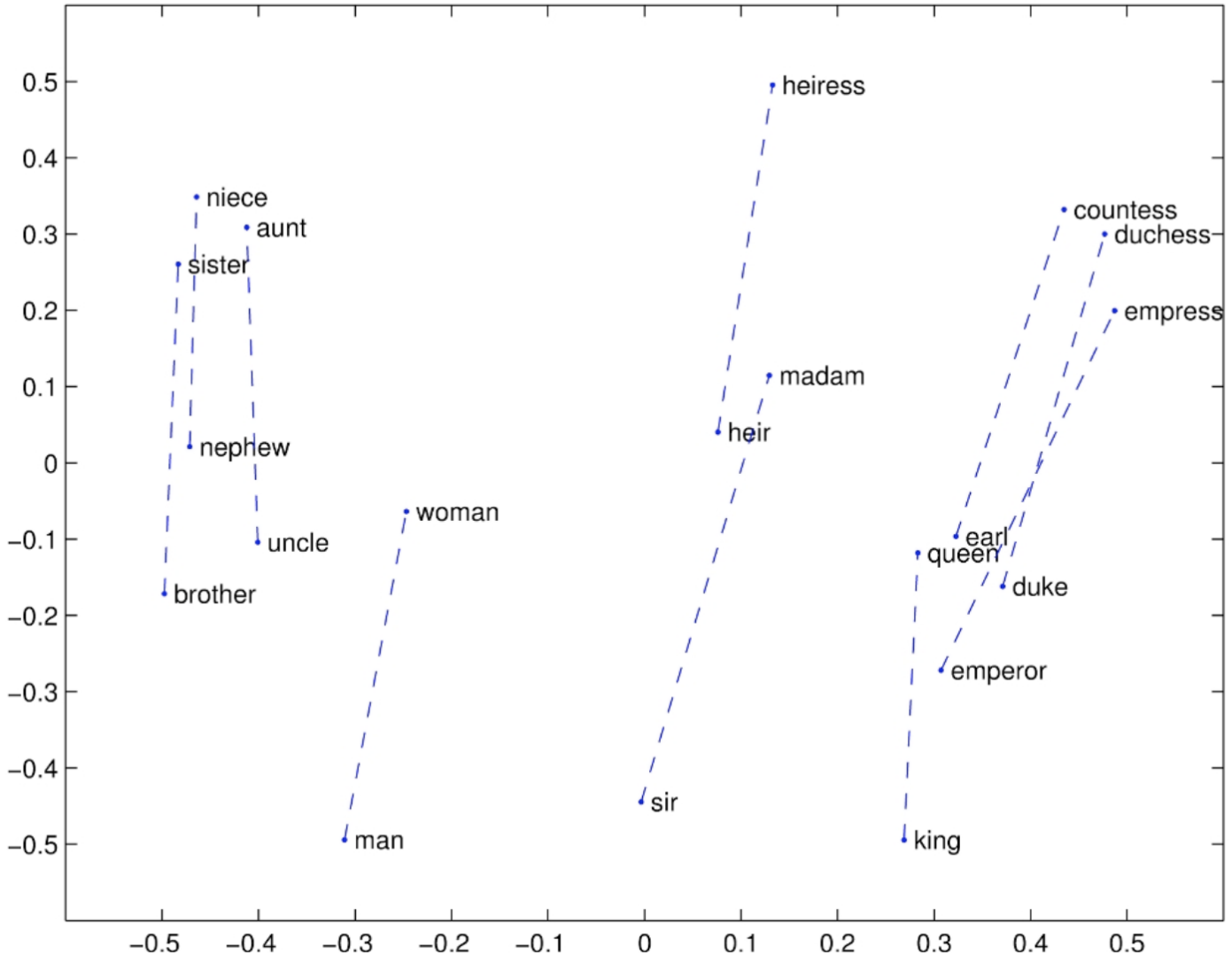
Paris: France ≈ London: ?

bad: worst ≈ cool: ?

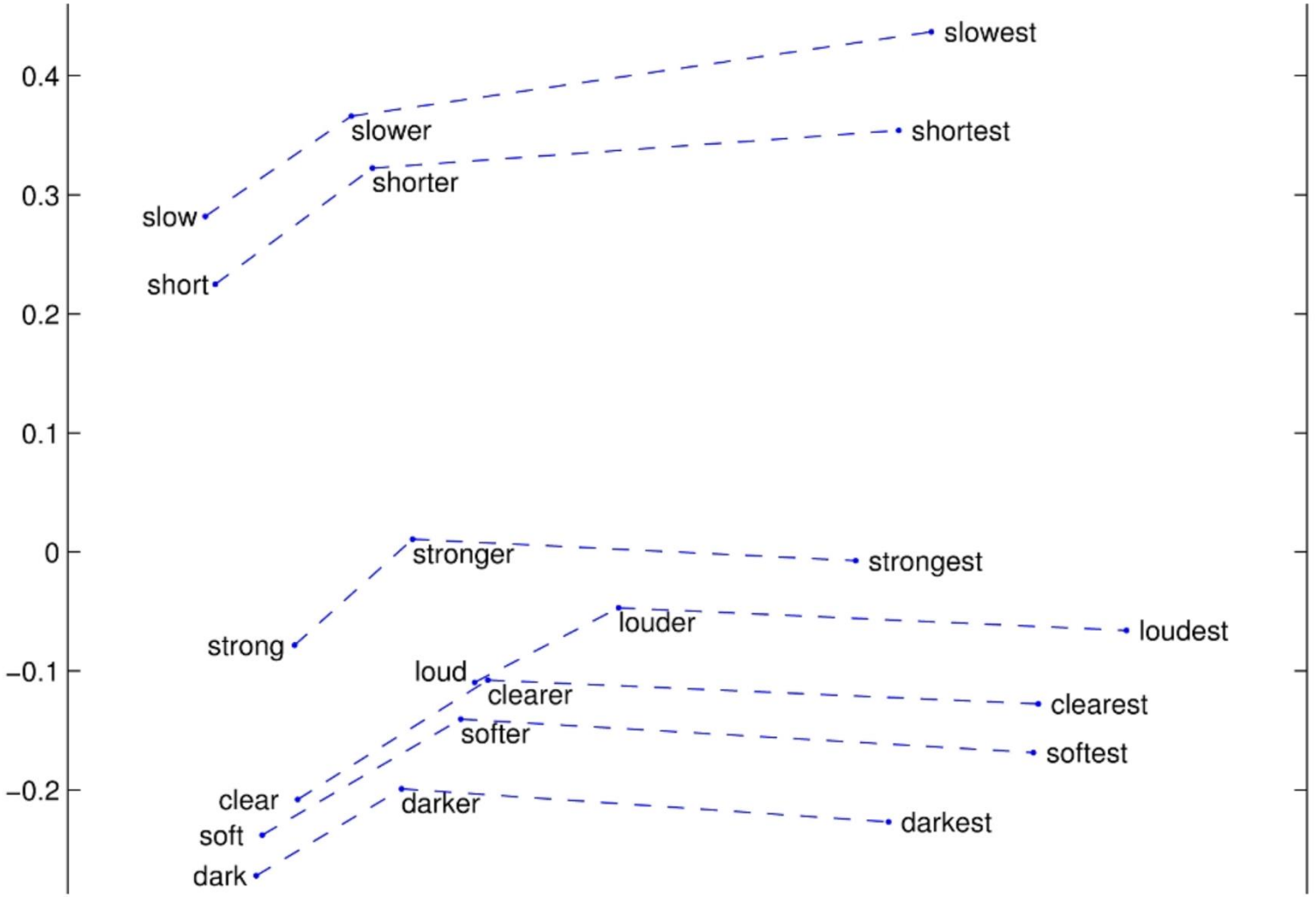
$$\arg\max_{w}\left(\cos(\mathbf{u}_w, \mathbf{u}_{woman} - \mathbf{u}_{man} + \mathbf{u}_{king})\right)$$



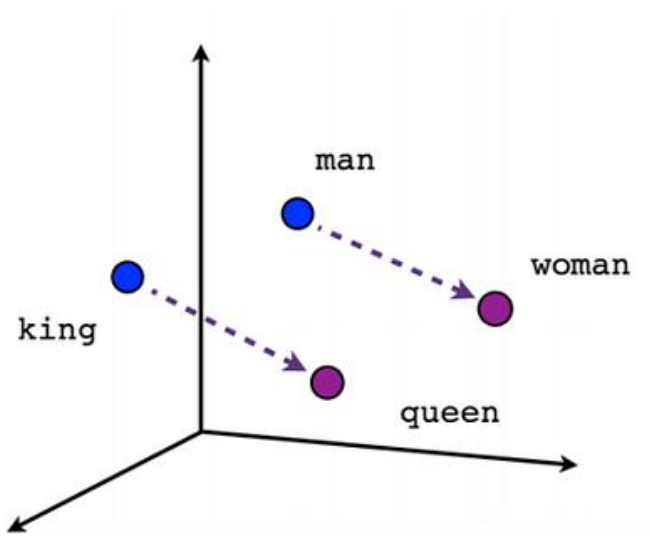
# Intrinsic Evaluation: Word Analogy



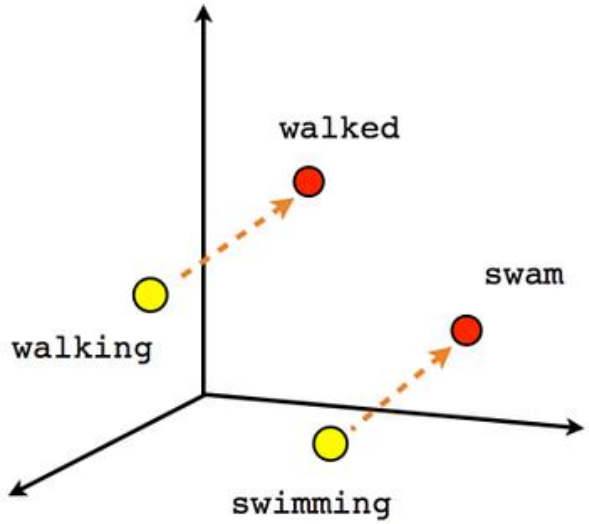
# Intrinsic Evaluation: Word Analogy



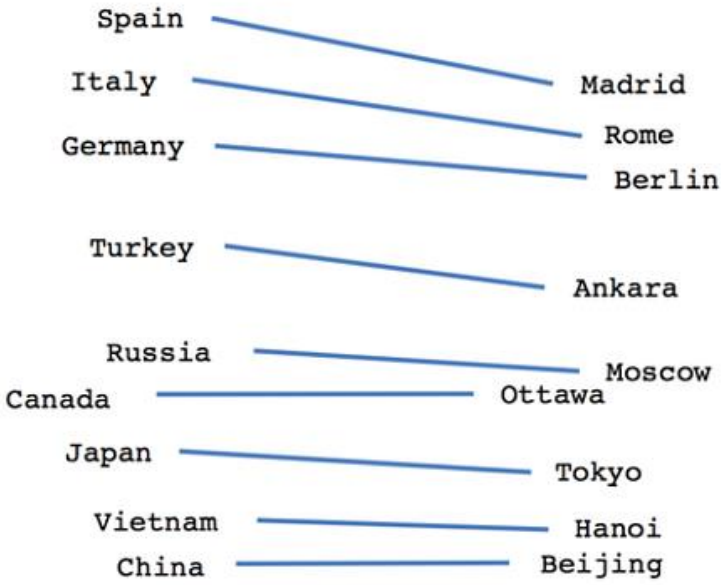
# Intrinsic Evaluation: Word Analogy




Male-Female




Verb tense

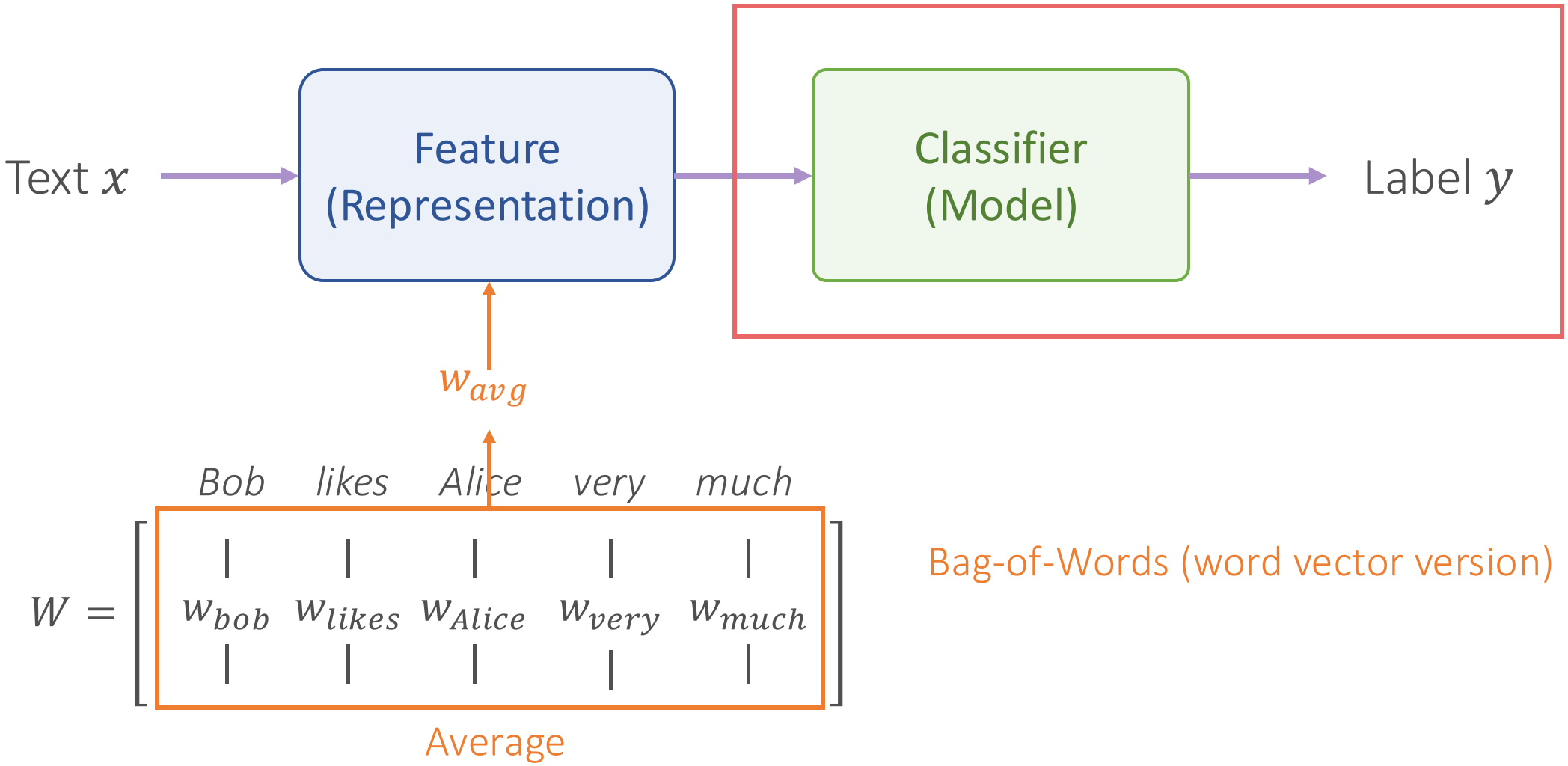



Country-Capital



https://sanjayc.medium.com/word2vec-analogical-reasoning-d47d3a66b9fb

# Extrinsic Evaluation: Downstream Performance



# Lecture Plan

- Word Representations
  - Variants
  - Evaluation
- Tokenization
  - Subwords
  - Byte-Pair Encoding
- Language Models
  - Definition of Language Models
  - N-Gram Language Models
  - Neural Language Models

# Recap: Word Embeddings as Learning Problem

- Learning vectors (also called embeddings) from text for representing words
- Input:
  - A large text corpus
    - Wikipedia + Gigaword 5: 6B tokens
    - Twitter: 27B tokens
    - Common Crawl: 840B tokens
  - Vocabulary $\mathcal{V}$ How to decide the vocabulary?
  - Vector dimension $d$ (e.g., 300)
- Output:
  - Mapping function $f: \mathcal{V} \rightarrow \mathbb{R}^d$

$$v_{apple} = \begin{pmatrix} -0.224 \\ 0.479 \\ 0.871 \\ -0.231 \\ 0.101 \end{pmatrix}$$

$$v_{digital} = \begin{pmatrix} 0.257 \\ 0.587 \\ -0.972 \\ -0.456 \\ -0.002 \end{pmatrix}$$

# 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 soooooo 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

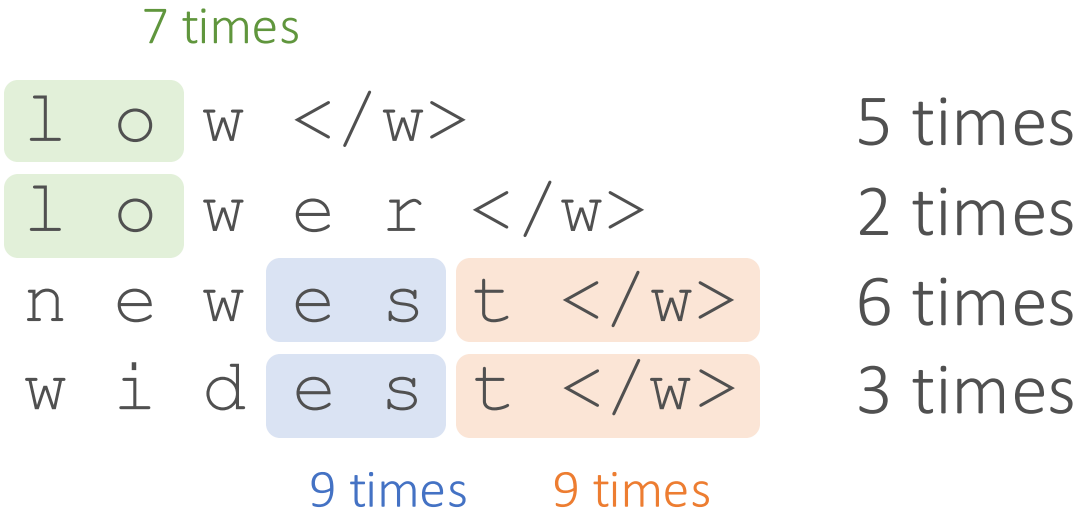
| | |
|---|---|
| l o w </w> | 5 times |
| l o w e r </w> | 2 times |
| n e w e s t </w> | 6 times |
| w i d e s t </w> | 3 times |

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>
l o w e r </w>
n e w est</w>
w i d est</w>
```

5 times
2 times
6 times
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

| | |
|---|---|
| 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
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

- Word Representations
  - Variants
  - Evaluation
- Tokenization
  - Subwords
  - Byte-Pair Encoding
- Language Models
  - Definition of Language Models
  - N-Gram Language Models
  - 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.*

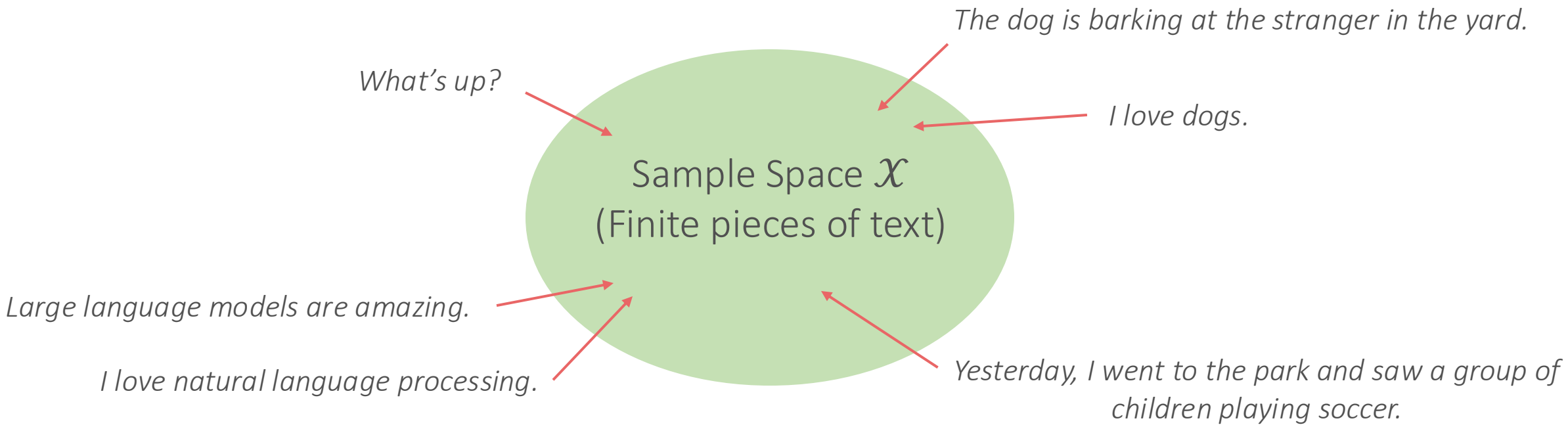
*Cats upon they chairs sleeping their dreams fall.*

*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

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

Next Token

Context

Next token prediction problem based on context

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

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

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

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

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

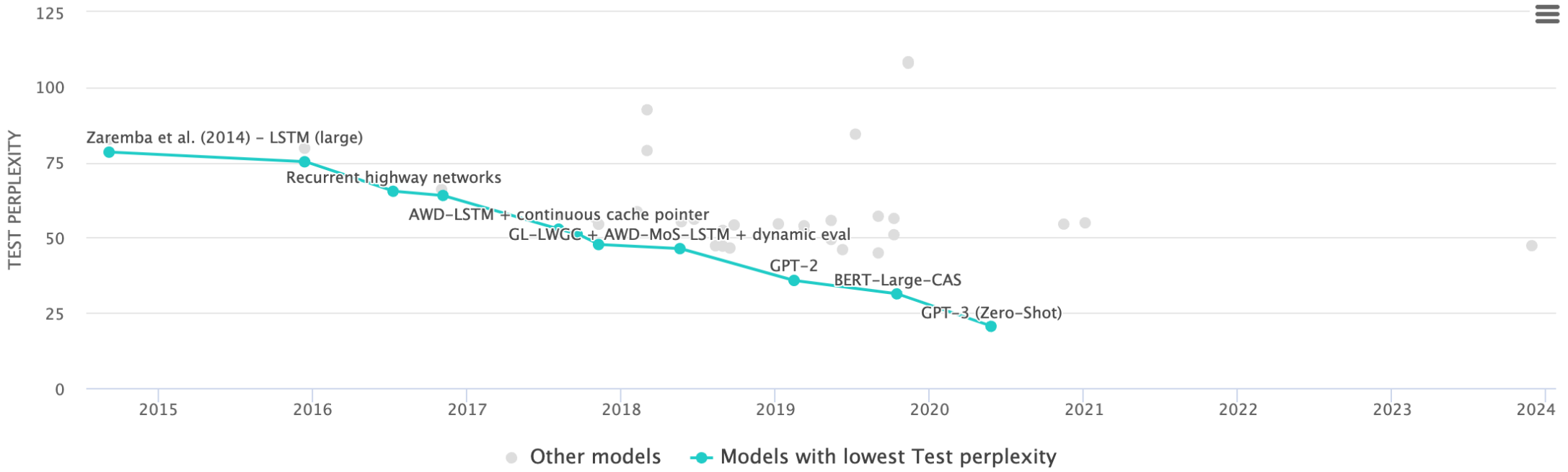
$$P(x) = \prod_j P(w_j)$$

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

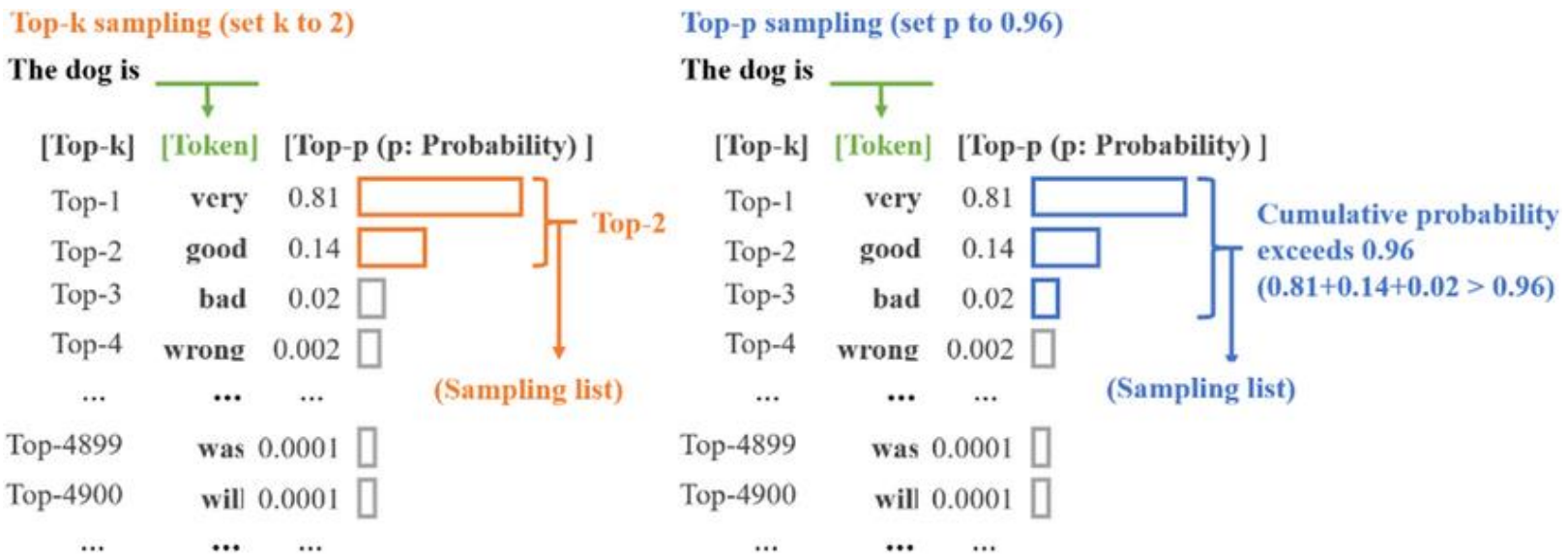
"Alice/Bob could not go to work that day because she/he had a doctor's appointment"

# Different Decoding Strategies

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

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

Top-k Sampling and Top-P Sampling






Determining the best feature combination through text and probabilistic feature analysis for GPT-2-based mobile app review detection, 2023

# 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

Compute perplexity

- Generate texts

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

Sample from word distribution

# N-Gram Language Models

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

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

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

A count-based solution: Collect training corpus and count

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

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 oranges
- Test set
  - I love apples
  - I like oranges
- 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 $\alpha$ 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

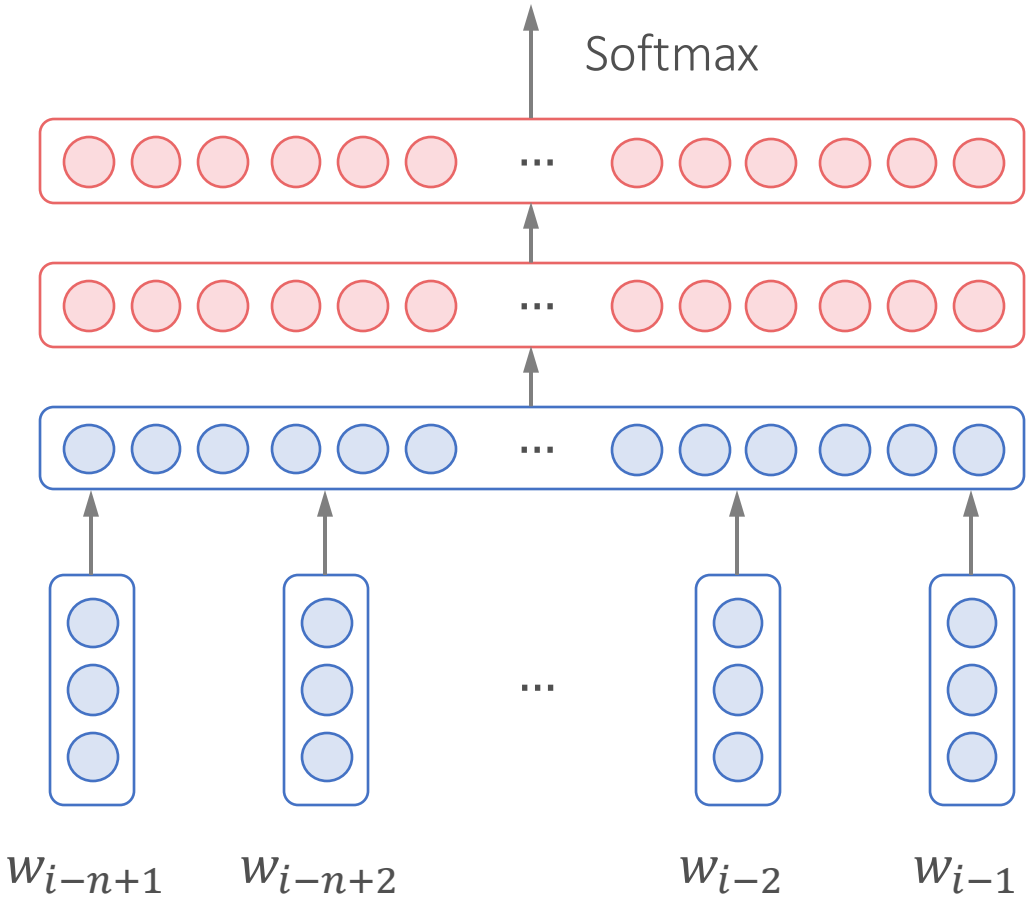
A count-based solution: Collect training corpus and count

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

Can we compute the probability in a different way?

# Neural Language Models

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




Training corpus

- I like apples
- I love oranges

Test set

- I love apples
- I like oranges

# 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

- Word Representations
  - Variants
  - Evaluation
- Tokenization
  - Subwords
  - Byte-Pair Encoding
- Language Models
  - Definition of Language Models
  - N-Gram Language Models
  - Neural Language Models

# Next Lecture

- Convolutional Neural Network
- Recurrent Neural Network