EE-508: Hardware Foundations for Machine Learning
An Introduction to NLP

University of Southern California

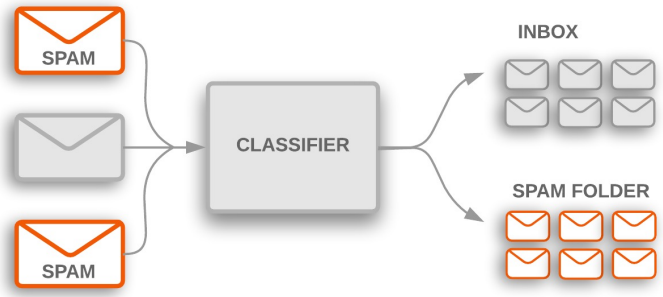
Ming Hsieh Department of Electrical and Computer Engineering

Instructors:
Arash Saifhashemi

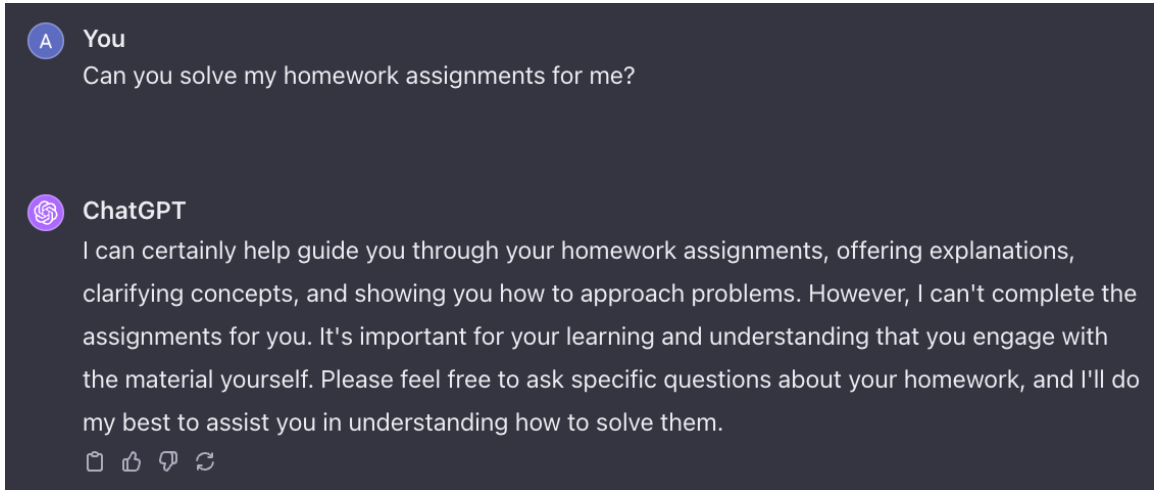
An Introduction to NLP

Introduction to Natural Language Processing (NLP)

- NLP is a branch of AI that enables computers to understand, interpret, and generate human language.
  - **Main Tasks**:
    - Text Classification (e.g., Spam Detection, Sentiment Analysis)
    - Machine Translation
    - Speech Recognition
    - Information Extraction
    - Question Answering
    - Text Generation



Source: developers.google.com/machine-learning/guides/text-classification



Source: ChatGPT

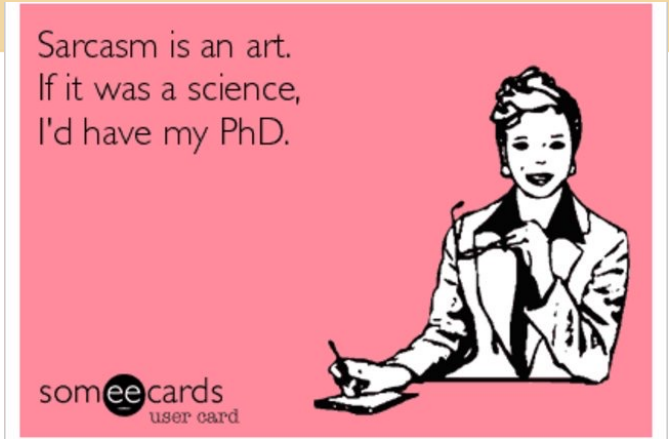
Challenges

Ambiguity and Context in Language

- I saw a man with a telescope.

Sarcasm and Figurative Language

- Oh, great! Another Monday.



Language Diversity and Evolution

- Y'all come back now, ya hear?
- The tweet went viral, and now it's trending worldwide.

Two Pillars of NLP

Representation

How to represent the language to machines?

Techniques: Tokenization, One-Hot Encoding, Word Embeddings (Word2Vec, BERT)

Modeling

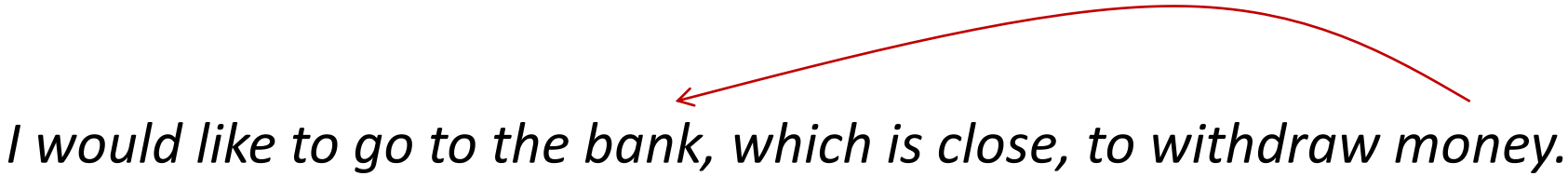
How to model languages statistically?

Techniques: n-gram models, RNNs, Transformers (BERT, GPT)

Representation

I would like to go to the bank, which is close, to withdraw money.

Representation



Representation

I would like to go to the bank, which is close, to withdraw money.

1 bank /ˈbæŋk/ *noun*

plural **banks**

Britannica Dictionary definition of BANK

[count]

1 : a business where people keep their money, borrow money, etc., or the building where such a business operates

- Our paychecks are deposited in/into the *bank* automatically.
- How much money do you have in the *bank*?
- My cousin works in/at a *bank*.
- I have to go to the *bank* today.

— often used before another noun

- *bank* customers
- How much money do you have in your *bank* account?

— see also SAVINGS BANK

3 bank /ˈbæŋk/ *noun*

plural **banks**

Britannica Dictionary definition of BANK

[count]

1 : the higher ground that is along the edge of a river, stream, etc.

- We sat on the *bank* of the river [=on the riverbank] to watch the boats.
- The stream overflowed its *banks*.

2 a : a steep slope **:** the side of a hill

- We planted bushes all along the *bank* in front of the house.
- They climbed a steep *bank* to get to the terrace.

— see also SANDBANK

b : a small hill that is built next to a road along a curve in order to make driving on that section of road safer

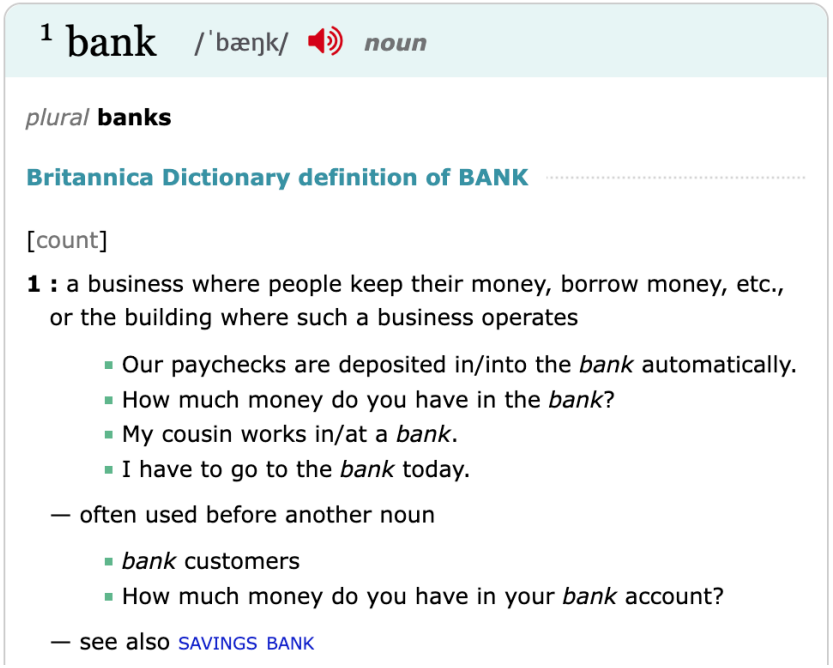
3 : a thick mass of clouds or fog

- a fog *bank*
- A *bank* of dark clouds entered the region.

Representation

Unfortunately, words are not enough. The meaning often comes from the context.

I would like to go to the bank, which is close, to withdraw money.



1 bank /ˈbæŋk/ *noun*

plural **banks**

Britannica Dictionary definition of BANK

[count]

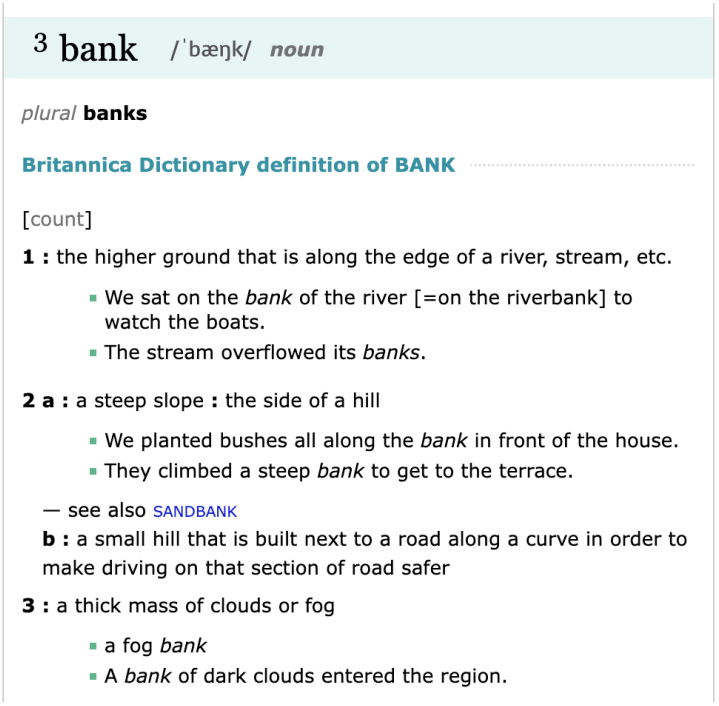
1 : a business where people keep their money, borrow money, etc., or the building where such a business operates

- Our paychecks are deposited in/into the *bank* automatically.
- How much money do you have in the *bank*?
- My cousin works in/at a *bank*.
- I have to go to the *bank* today.

— often used before another noun

- *bank* customers
- How much money do you have in your *bank* account?

— see also SAVINGS BANK



3 bank /ˈbæŋk/ *noun*

plural **banks**

Britannica Dictionary definition of BANK

[count]

1 : the higher ground that is along the edge of a river, stream, etc.

- We sat on the *bank* of the river [=on the riverbank] to watch the boats.
- The stream overflowed its *banks*.

2 a : a steep slope **:** the side of a hill

- We planted bushes all along the *bank* in front of the house.
- They climbed a steep *bank* to get to the terrace.

— see also SANDBANK

b : a small hill that is built next to a road along a curve in order to make driving on that section of road safer

3 : a thick mass of clouds or fog

- a fog *bank*
- A *bank* of dark clouds entered the region.

Representation: Word Indexing

- **Word Indexing (Integer Encoding):**
  - Map each word to a **unique** integer
  - Convert text into **sequences** of integers.

- **Example**:
  - **Sentence**: *"The cat sat on the mat"*
  - **Unique Words**:
    - *"The", "cat", "sat", "on", "mat"*
  - **Integer Assignment**:
    - *"The" → 1*
    - *"cat" → 2*
    - *"sat" → 3*
    - *"on" → 4*
    - *"mat" → 5*
  - Encoded: [1, 2, 3, 4, 1, 5]

Representation: Word Indexing

- **Word Indexing (Integer Encoding):**
  - Map each word to a **unique** integer
  - Convert text into **sequences** of integers.
- **Advantages**:
  - **Simple** and efficient.
  - **Compatible** with various neural network architectures.

- **Example**:
  - **Sentence**: *"The cat sat on the mat"*
  - **Unique Words**:
    - *"The", "cat", "sat", "on", "mat"*
  - **Integer Assignment**:
    - *"The" → 1*
    - *"cat" → 2*
    - *"sat" → 3*
    - *"on" → 4*
    - *"mat" → 5*
  - Encoded: [1, 2, 3, 4, 1, 5]

Representation: Word Indexing

- **Word Indexing (Integer Encoding):**
  - Map each word to a **unique** integer
  - Convert text into **sequences** of integers.
- **Advantages**:
  - **Simple** and efficient.
  - **Compatible** with various neural network architectures.
- **Limitations**:
  - Does not capture **semantic** relationships.
  - The integer assignment is **arbitrary**.
  - **Numeric closeness** doesn't imply semantic similarity.

- **Example**:
  - **Sentence**: *"The cat sat on the mat"*
  - **Unique Words**:
    - *"The", "cat", "sat", "on", "mat"*
  - **Integer Assignment**:
    - *"The" → 1*
    - *"cat" → 2*
    - *"sat" → 3*
    - *"on" → 4*
    - *"mat" → 5*
  - Encoded: [1, 2, 3, 4, 1, 5]

Representation: One-Hot

- **One-Hot Encoding:**
  - Each word is transformed into a vector with a '1' in its index position and '0's elsewhere.

- **Example**:
  - **Sentence**: *"The cat sat on the mat"*
  - **Unique Words**:
    - *"The", "cat", "sat", "on", "mat"*
- **One-Hot Vectors**:
  - "The" → [1, 0, 0, 0, 0]
  - "cat" → [0, 1, 0, 0, 0]
  - "sat" → [0, 0, 1, 0, 0]
  - "on" → [0, 0, 0, 1, 0]
  - "mat" → [0, 0, 0, 0, 1]
- **Sentence Representation**:
  - Sequence of One-Hot Vectors:
  - ["The", "cat", "sat", "on", "the", "mat"]
  - [[1, 0, 0, 0, 0],
    [0, 1, 0, 0, 0],
    [0, 0, 1, 0, 0],
    [0, 0, 0, 1, 0],
    [1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1]]

Representation: One-Hot

- **One-Hot Encoding:**
  - Each word is transformed into a vector with a '1' in its index position and '0's elsewhere.
- **Advantages**:
  - **Simplicity**.
  - Each word is uniquely represented.

- **Example**:
  - **Sentence**: *"The cat sat on the mat"*
  - **Unique Words**:
    - *"The", "cat", "sat", "on", "mat"*
- **One-Hot Vectors**:
  - "The" → [1, 0, 0, 0, 0]
  - "cat" → [0, 1, 0, 0, 0]
  - "sat" → [0, 0, 1, 0, 0]
  - "on" → [0, 0, 0, 1, 0]
  - "mat" → [0, 0, 0, 0, 1]
- **Sentence Representation**:
  - Sequence of One-Hot Vectors:
  - ["The", "cat", "sat", "on", "the", "mat"]
  - [[1, 0, 0, 0, 0],
    [0, 1, 0, 0, 0],
    [0, 0, 1, 0, 0],
    [0, 0, 0, 1, 0],
    [1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1]]

Representation: One-Hot

- **One-Hot Encoding:**
  - Each word is transformed into a vector with a '1' in its index position and '0's elsewhere.
- **Advantages**:
  - Simplicity.
  - Each word is uniquely represented.
- **Limitations**:
  - Vectors become **very large** with extensive vocabularies.
  - Most elements in the vector are **zeros**.
  - All words are **equally dissimilar**!

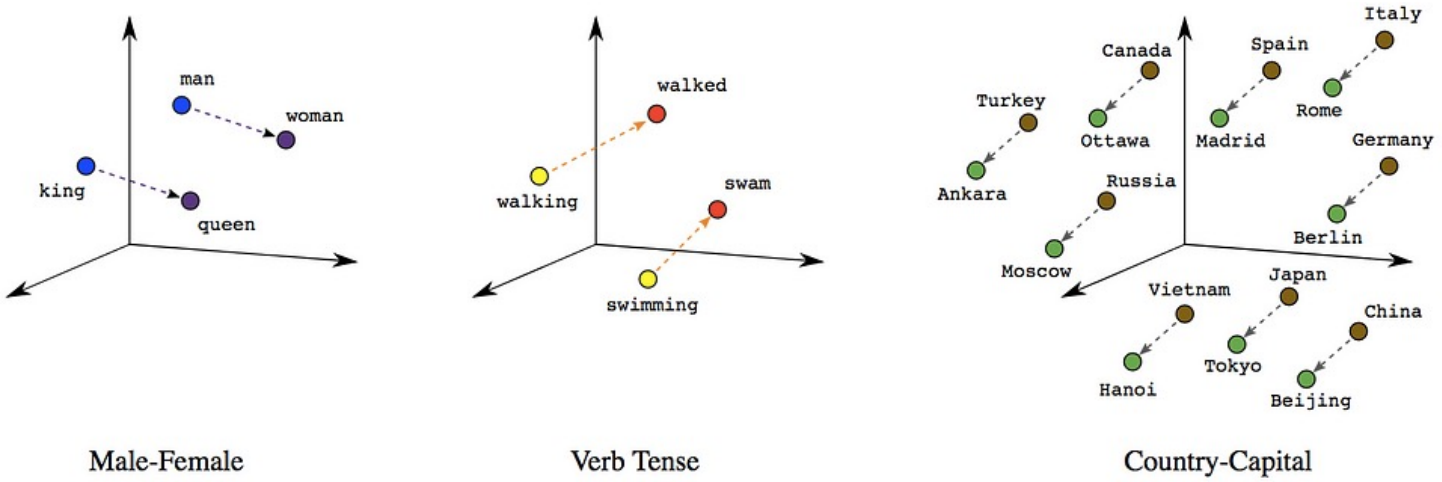
- **Example**:
  - **Sentence**: *"The cat sat on the mat"*
  - **Unique Words**:
    - *"The", "cat", "sat", "on", "mat"*
- **One-Hot Vectors**:
  - `"The" → [1, 0, 0, 0, 0]`
  - `"cat" → [0, 1, 0, 0, 0]`
  - `"sat" → [0, 0, 1, 0, 0]`
  - `"on" → [0, 0, 0, 1, 0]`
  - `"mat" → [0, 0, 0, 0, 1]`
- **Sentence Representation**:
  - Sequence of One-Hot Vectors:
    - `["The", "cat", "sat", "on", "the", "mat"]`
    -
```
[[1, 0, 0, 0, 0],
 [0, 1, 0, 0, 0],
 [0, 0, 1, 0, 0],
 [0, 0, 0, 1, 0],
 [1, 0, 0, 0, 0],
 [0, 0, 0, 0, 1]]
```

Representation: Word Embedding

- Dense vector that captures semantic meaning and relationships

Representation: Word Embedding

- Dense vector that captures semantic meaning and relationships

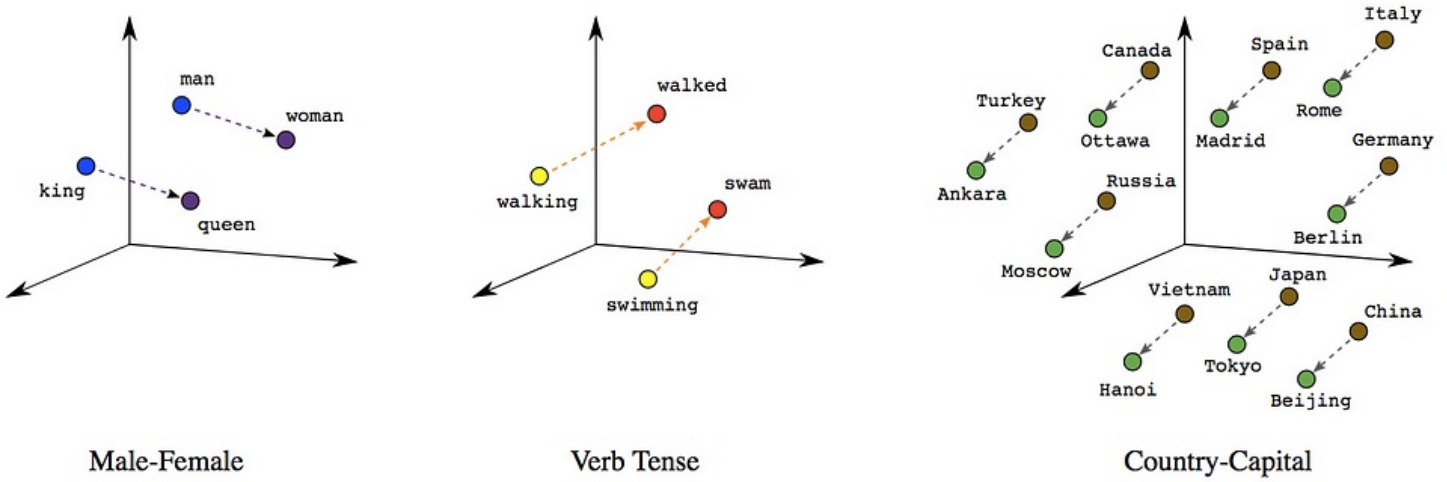


Male-Female Verb Tense Country-Capital

Source: https://developers.google.com/machine-learning/crash-course/embeddings/translating-to-a-lower-dimensional-space

Representation: Word Embedding

- Dense vector that captures semantic meaning and relationships
- **Advantages**:
  - Reduces dimensionality and sparsity.
  - Able to represent synonyms, antonyms, and varying degrees of semantic similarity.
  - Context-Aware: Reflects polysemy – words with multiple meanings based on context.

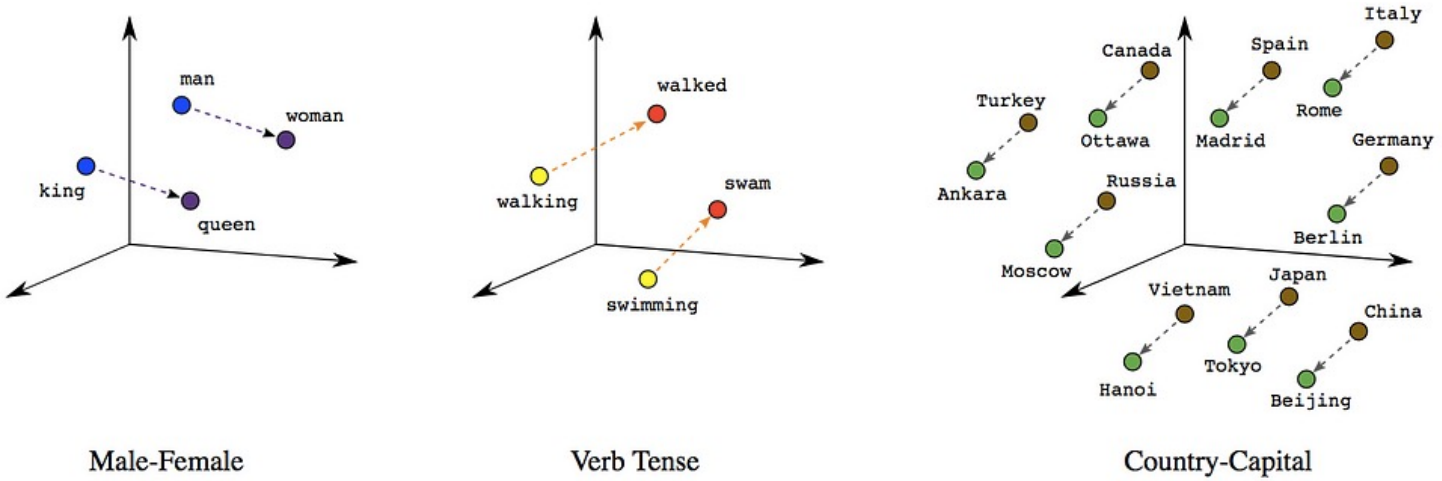


Male-Female Verb Tense Country-Capital

Source: https://developers.google.com/machine-learning/crash-course/embeddings/translating-to-a-lower-dimensional-space

Representation: Word Embedding

- Dense vector that captures semantic meaning and relationships
- **Advantages**:
  - Reduces dimensionality and sparsity.
  - Able to represent synonyms, antonyms, and varying degrees of semantic similarity.
  - Context-Aware: Reflects polysemy – words with multiple meanings based on context.
- **Limitations**:
  - Traditional embeddings assign a single vector per word, regardless of context.
    - For example, the word ‘bank’ has the same vector whether it refers to a river or a financial institution.
    - Contextual embeddings like BERT solve this by generating dynamic vectors based on sentence context.
  - Requires Pretraining: Quality depends on the corpus and pretraining process.



Source: https://developers.google.com/machine-learning/crash-course/embeddings/translating-to-a-lower-dimensional-space

Why Pretraining Matters in Word Embeddings

- **Word embeddings are pretrained** on large text corpora
- The **quality of embeddings** depends on:
  - Size of the training data
  - Relevance to your domain
  - Pretraining method (e.g., Skip-gram, CBOW)
- **Example:**
  - Medical corpus → “insulin” close to “glucose”
  - Movie reviews → No such relationship captured
  - *Low-quality or mismatched corpora → Poor embeddings!*

Why Pretraining Matters in Contextual Embeddings

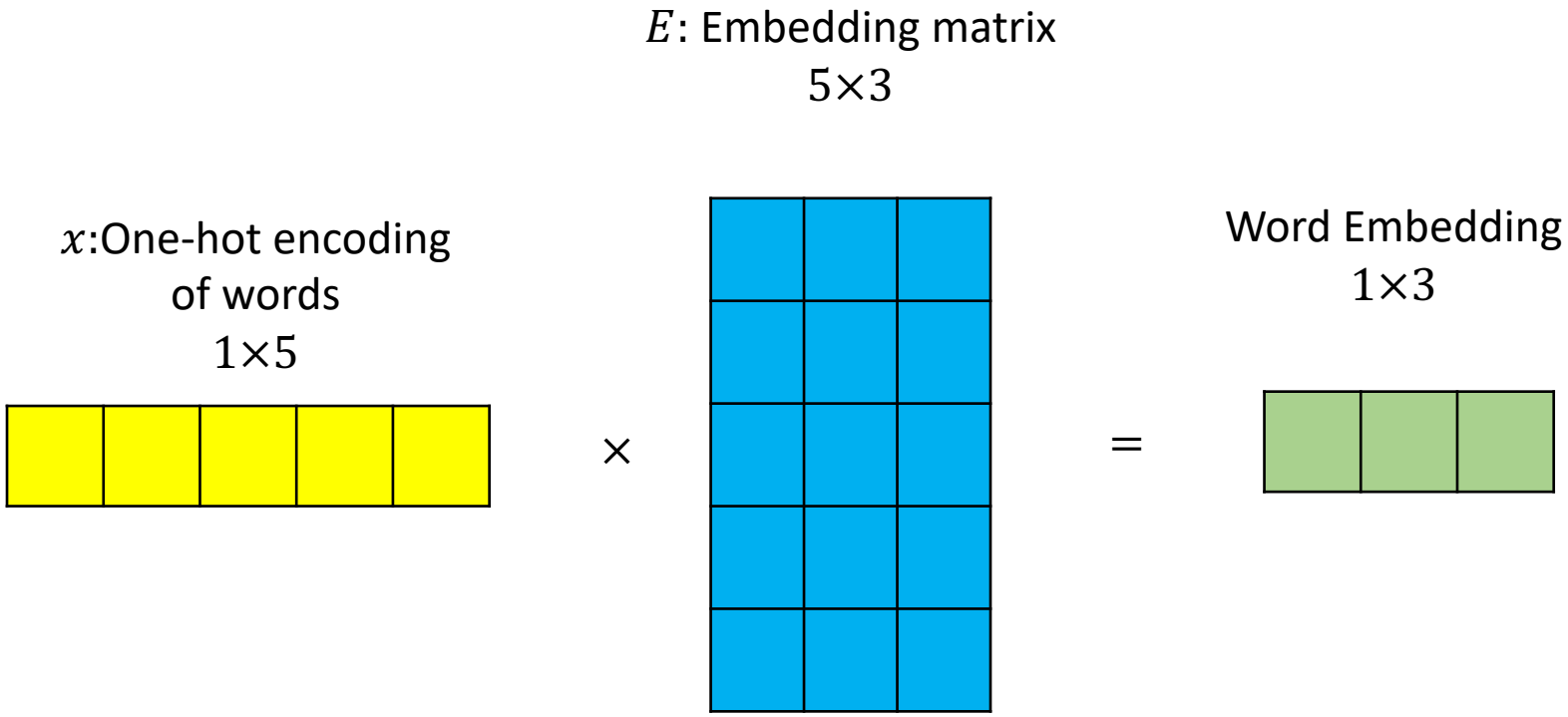
- **Contextual embeddings** (e.g., BERT, RoBERTa) generate **different vectors** for the same word based on its context.
- These models are **pretrained on massive corpora** using deep transformer architectures.
- **Quality depends on:**
  - The size and diversity of the corpus
  - The pretraining objectives (e.g., masked language modeling, next sentence prediction)
  - The number of parameters and training duration
- **Example:**
  - BERT trained on general web text understands:
    - “bank” (money) vs. “bank” (river) based on context
  - A domain-specific model (e.g., BioBERT) is better for biomedical text
- Using the wrong pretrained model for a task can lead to poor performance.

Representation: Word Embedding

- **Example**:
  - **Sentence**: *"The cat sat on the mat"*
  - **Unique Words**:
    - *"The", "cat", "sat", "on", "mat"*
- **Word Embedding**:
  - `"The" → [0.2, -0.4, 0.7, 0.1, 0.8]`
  - `"cat" → [0.5, 0.3, -0.6, 0.1, -0.2]`
  - `"sat" → [0.1, -0.3, 0.6, 0.5, 0.4]`
  - `"on" → [-0.1, 0.6, 0.2, -0.4, 0.3]`
  - `"mat" → [0.4, -0.5, 0.3, 0.2, -0.7]`
- **Sentence Representation**:
  - `["The", "cat", "sat", "on", "the", "mat"]`
  -
```
[[0.2, -0.4, 0.7, 0.1, 0.8],
 [0.5, 0.3, -0.6, 0.1, -0.2],
 [0.1, -0.3, 0.6, 0.5, 0.4],
 [-0.1, 0.6, 0.2, -0.4, 0.3],
 [0.2, -0.4, 0.7, 0.1, 0.8],
 [0.4, -0.5, 0.3, 0.2, -0.7]]
```

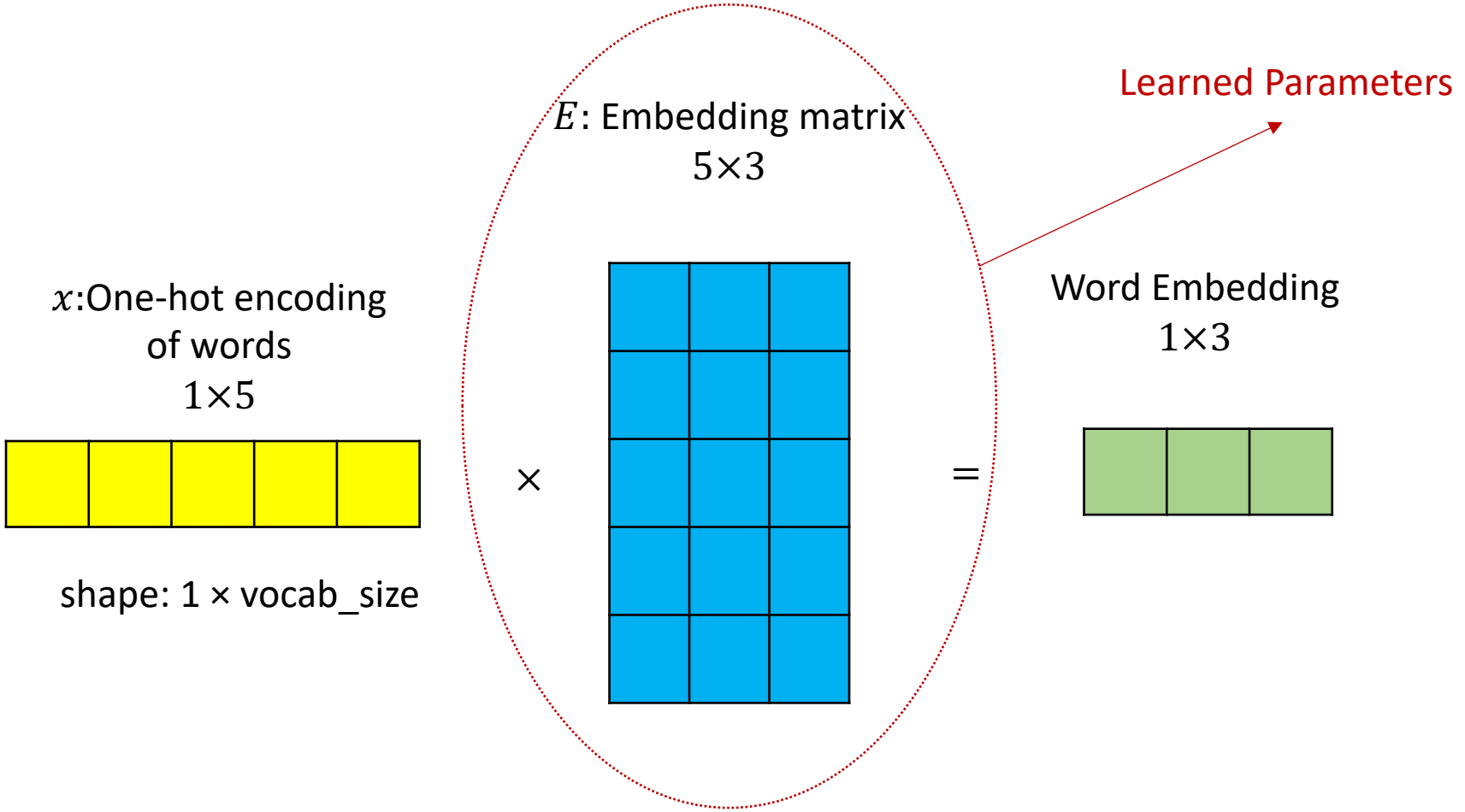
In practice, each word would be represented by a much higher-dimensional vector (typically hundreds of dimensions)

Word Embeddings as Learned Representations



Example: a corpus of 5 words with embedding vector of size 3

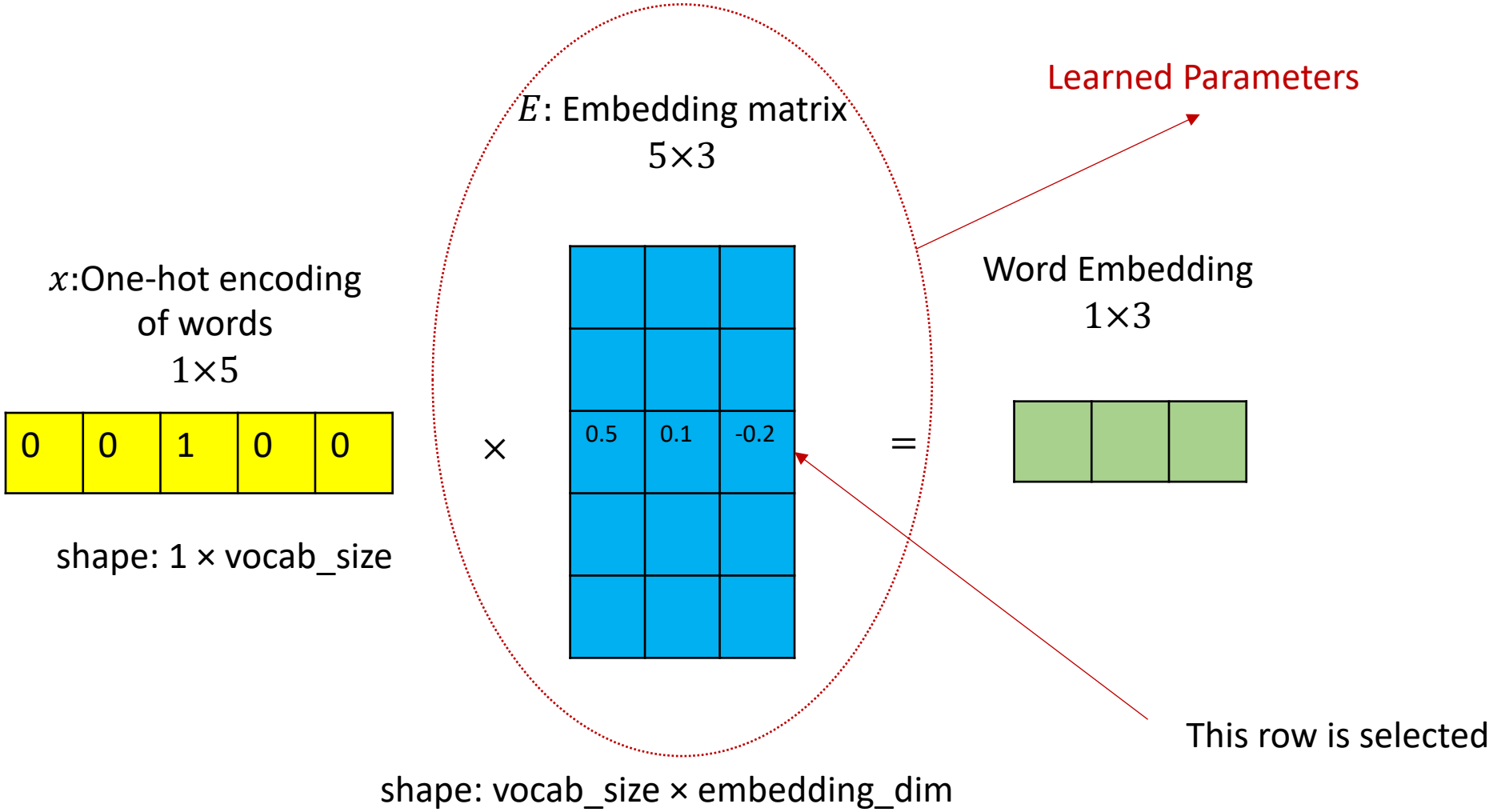
Word Embeddings as Learned Representations



Example: a corpus of 5 words with embedding vector of size 3

The multiplication selects the row from E corresponding to the active index in the one-hot vector.

Word Embeddings as Learned Representations



Example: a corpus of 5 words with embedding vector of size 3

The multiplication selects the row from E corresponding to the active index in the one-hot vector.

Positional Encoding

- Adding Order to Word Embeddings
  - Word embeddings (e.g., Word2Vec, BERT embeddings) provide **semantic meaning** but **lack order awareness**.
  - Transformers process inputs **in parallel**, so we must **inject positional information**.
- Types:
  - Sinusoidal (fixed). Will be discussed in more details.
  - learned (trainable)
    - Each position has a trainable vector.
    - More flexible but needs to be learned from data.

Example Comparison: Sinusoidal vs. No Position

- **Sentence:**
  "The dog chased the cat"
- **Without Positional Encoding**
  - Each word is embedded independently.
  - “The” → same vector every time
  - “dog”, “chased”, “cat” → just semantic vectors
- The model **can’t tell if ‘the cat chased the dog’** or **‘the dog chased the cat’**
- They look like:

[the, dog, chased, the, cat]
[the, cat, chased, the, dog] ← Same vectors in different order = same input to model

Example Comparison: Sinusoidal vs. No Position

- **Sentence:**
  "The dog chased the cat"
- **Without Positional Encoding**
  - Each word is embedded independently.
  - “The” → same vector every time
  - “dog”, “chased”, “cat” → just semantic vectors
- The model **can’t tell if ‘the cat chased the dog’** or **‘the dog chased the cat’**
- They look like:

[the, dog, chased, the, cat]
[the, cat, chased, the, dog] ← Same vectors in different order = same input to model

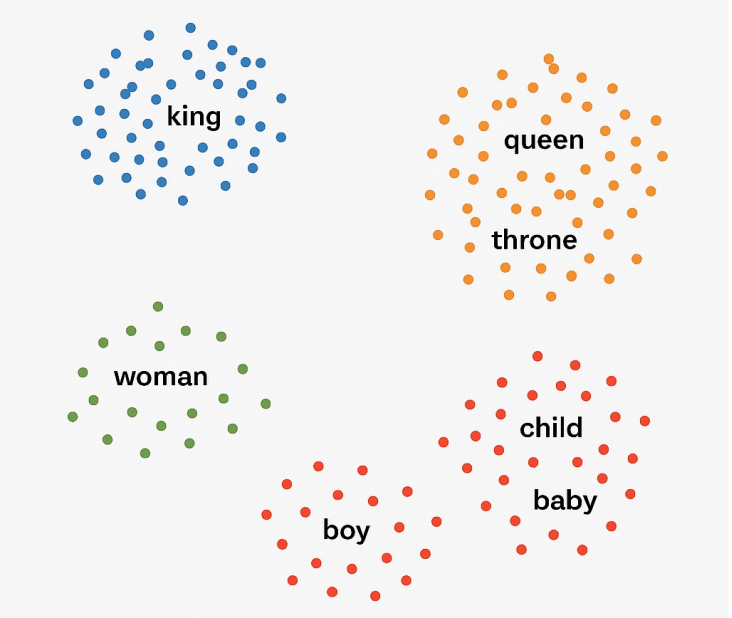
With Positional Encoding

Embedding = Word Vector + Position Vector

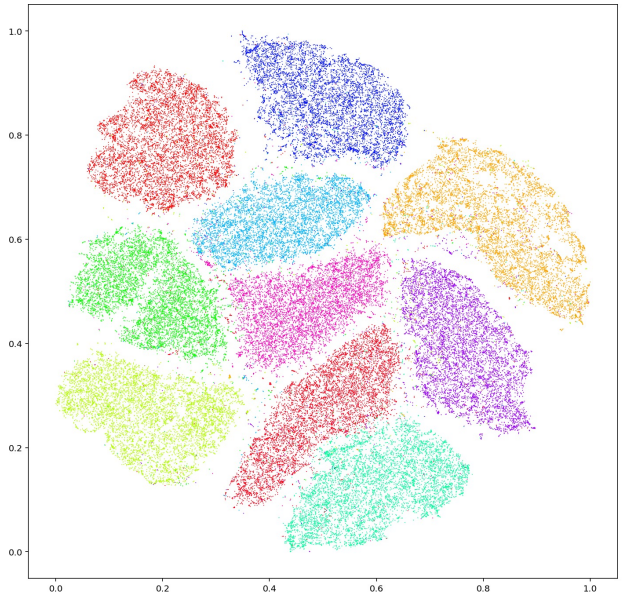
Order now matters; "dog chased cat" ≠ "cat chased dog"

Visualizing Word Embeddings

- Embeddings have 100s of dimensions in practice.
- Use t-SNE or PCA to reduce high-dimensional embeddings to 2D/3D
- Nearby points = semantically similar words



After reducing the dimension, words with close meanings are clustered together



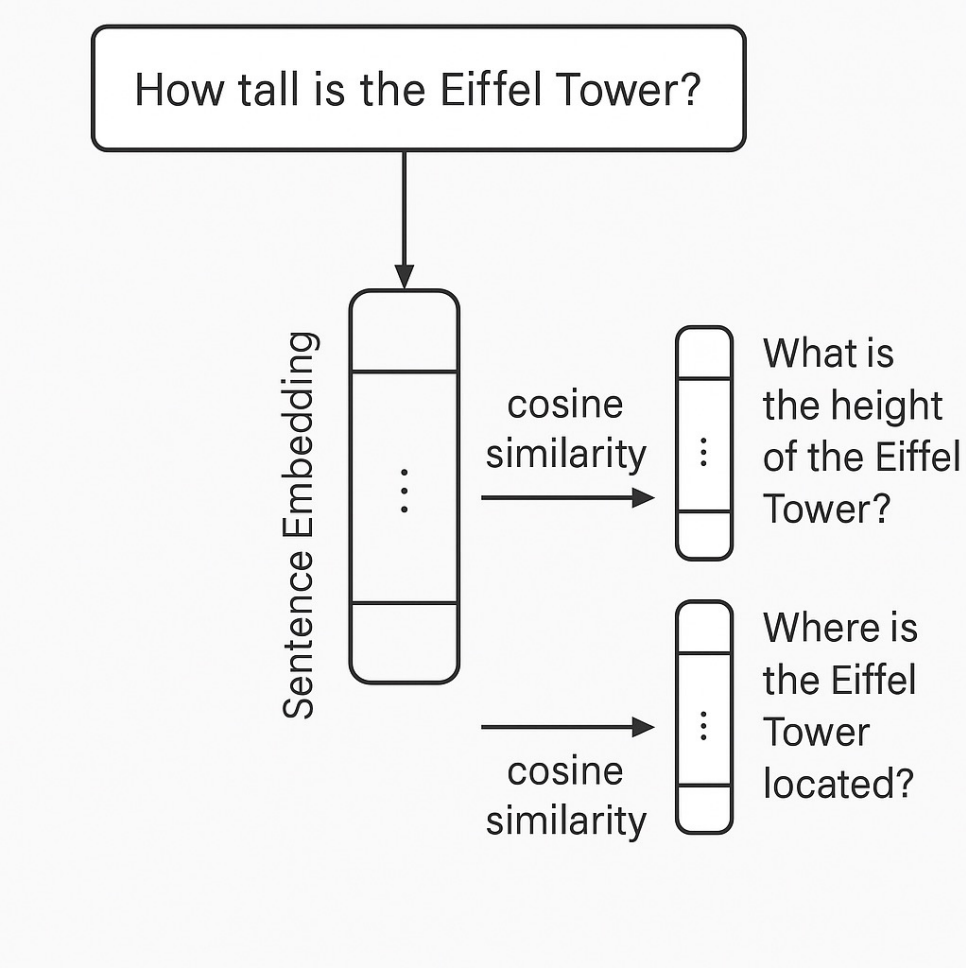
t-SNE visualization of word embeddings from the MNIST dataset, where each point represents a digit (0–9) projected from a high-dimensional space (e.g., 784-dimensional raw pixels or 128+ dimensional embeddings) into 2D

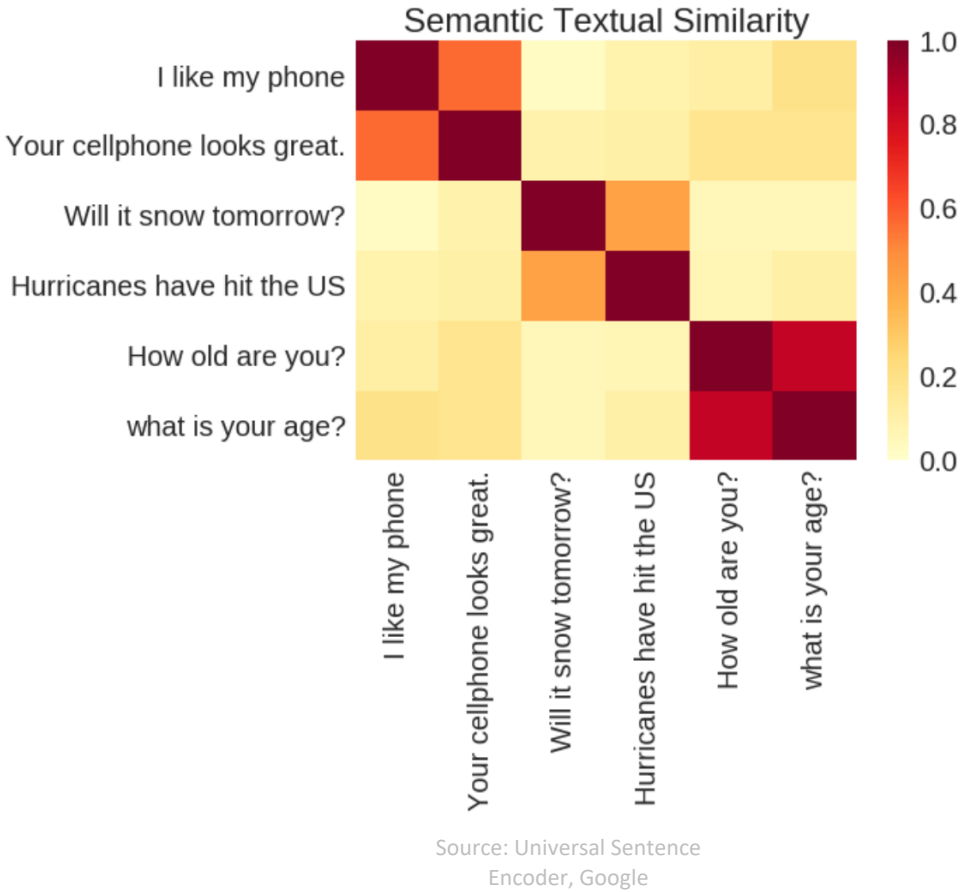
- Each color cluster corresponds to a different digit (0–9).

Embedding Full Sentences

- Word embeddings represent words, but many NLP tasks (like search, paraphrase detection, sentiment analysis) require **sentence-level meaning**.
- **What Are Sentence Embeddings?**
  - A **single vector** that represents the meaning of a **whole sentence**, not just individual words.
  - Unlike averaging word vectors, sentence encoders capture **structure**, **semantics**, and **context**.
- **Example Use Cases**
  - Semantic Search: “What is the capital of France?” ≈ “France’s capital city?”
  - Duplicate Question Detection (e.g., Quora)
  - Intent Matching in chatbots
  - Document clustering and classification

Embedding Full Sentences



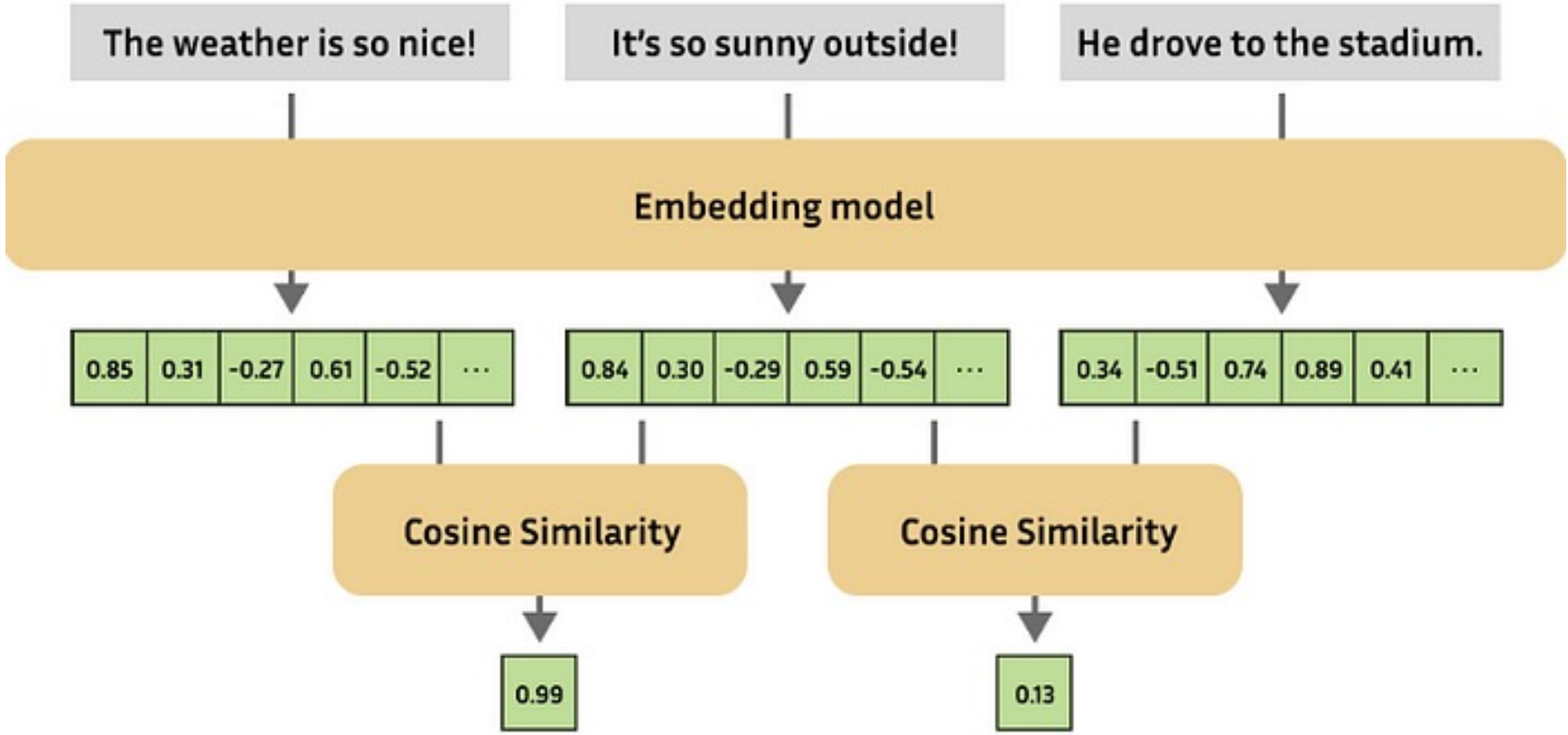


Source: Universal Sentence Encoder, Google

Sentence similarity scores using embeddings from the universal sentence encoder.

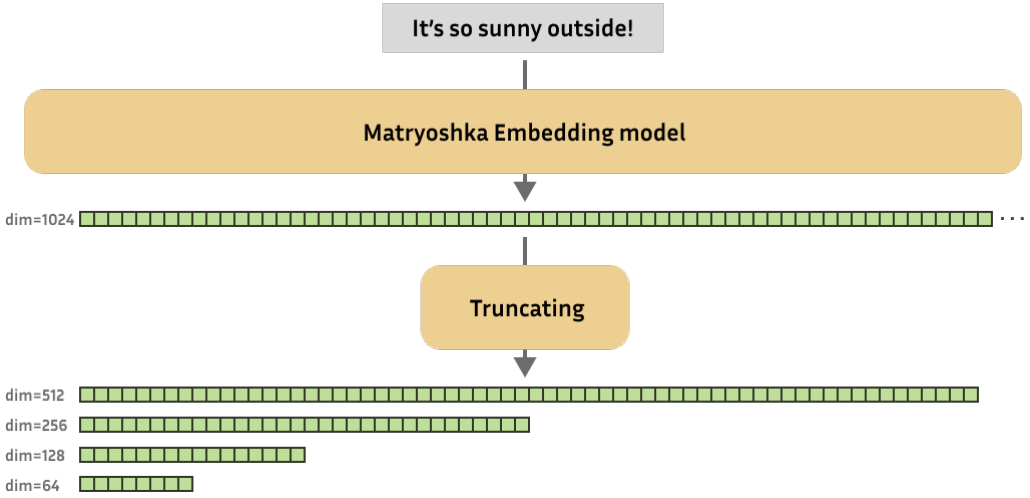
A sentence like "How tall is the Eiffel Tower?" is encoded into a dense vector, which is then compared with other sentence vectors using cosine similarity to determine semantic closeness.

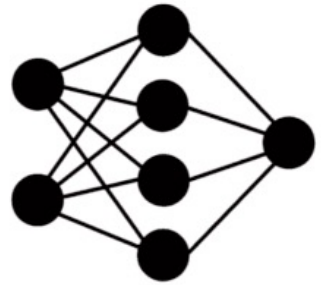
Embedding Full Sentences



Source: https://huggingface.co/blog/matryoshka

Matryoshka Embeddings





1. Compute Matryoshka Embedding

Source: https://huggingface.co/blog/matryoshka

Applications:

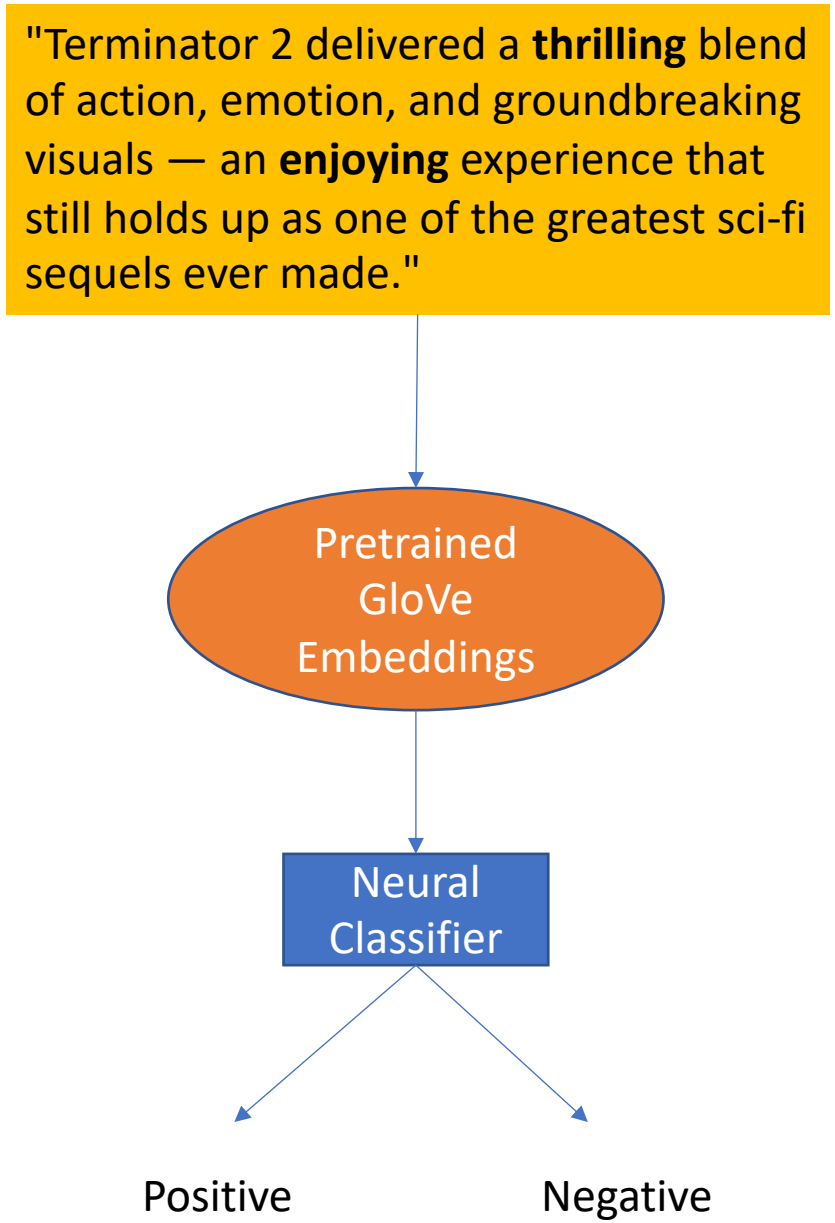
- Low-resource deployment (e.g., mobile, edge devices)
- Fast retrieval with low-dim, reranking with high-dim
- Flexible transfer learning across tasks
- Storage-efficient, scalable embeddings for variable use cases

Word-Level Transfer Learning

- Reusing pretrained word embeddings (e.g., Word2Vec, GloVe) for a new NLP task
  - Word vectors are trained on large corpora and transferred to downstream models
- **Two Modes of Use**
  - **Frozen Embeddings**
    - The embedding layer is **not updated** during training.
    - Faster training, less risk of overfitting.
    - Useful when you have little training data.
  - **Fine-Tuned Embeddings**
    - The embedding layer **is updated** during task-specific training.
    - Learns **task-specific nuances**, leading to better performance.

Fine-Tuning Example

- Task: Classify movie reviews as positive or negative.
- Steps:
  - Start with pretrained GloVe vectors.
  - Feed them into a neural classifier (e.g., LSTM or Transformer).
  - **Fine-tune**: Let the model adjust the embedding weights slightly while training on labeled movie review data.
  - Result: The word "**good**" in this context might shift closer to "**enjoyable**" or "**amazing**," helping with domain-specific accuracy.



Two Pillars of NLP

Representation

How to represent the language to machines?

Modeling

How to model languages statistically?

Modeling

- Language models:
  - Understand, interpret, generate, and respond to human language.
  - Algorithms that **assign probabilities** to **sequences of words**.
  - Predict the likelihood of a **word or sequence** following a sequence of words.

Modeling

- Language models:
  - Understand, interpret, generate, and respond to human language.
  - Algorithms that **assign probabilities** to **sequences of words**.
  - Predict the likelihood of a **word or sequence** following a sequence of words.

Translation Example:

$$P(I\ \boldsymbol{swim}\ to\ the\ school\ everyday) <\ P(I\ \boldsymbol{walk}\ to\ the\ school\ everyday)$$

Modeling

- Language models:
  - Understand, interpret, generate, and respond to human language.
  - Algorithms that **assign probabilities** to **sequences of words**.
  - Predict the likelihood of a **word or sequence** following a sequence of words.

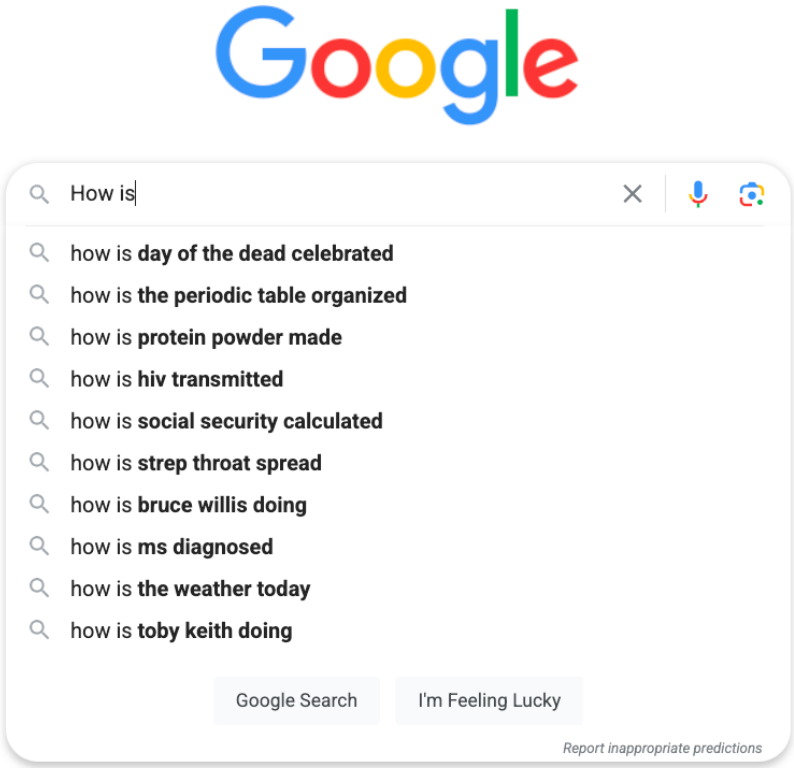
Translation Example:

$$P(I\ \boldsymbol{swim}\ to\ the\ school\ everyday) < P(I\ \boldsymbol{walk}\ to\ the\ school\ everyday)$$

Speech Recognition Example:

$$P(I\ walk\ to\ the\ school\ everyday) > P(I\ talk\ to\ cool\ everyday)$$

Other Applications



Next word prediction

Evolution of Language models

History of Language Models

Model/Technique	Time Period	Description
N-Grams	1990s - Early 2000s	Basic method for text data processing, breaking down text into chunks of 'n' items.
Bag of Words (BoW)	1990s - Early 2000s	Represents text data by counting word frequency, disregarding grammar and word order.
Recurrent Neural Networks (RNNs)	Early 2010s	Neural networks capable of handling sequences of data, with internal memory. Captures short-term dependencies in sequences
Long Short-Term Memory Networks (LSTMs)	Mid 2010s	Advanced RNNs designed to remember long-term dependencies and avoid vanishing gradient.
Word Embeddings (Word2Vec, GloVe)	Early-Mid 2010s	Methods to create dense vector representations of words capturing semantic meaning.
Attention Mechanism	Mid 2017	Allows models to focus on relevant parts of the input sequence for long-range dependencies.
Transformers	2017	Revolutionized NLP with more parallelizable and efficient handling of longer sequences. Enabled parallel processing and long-range dependency modeling.
BERT (Bidirectional Encoder Representations from Transformers)	2018	Pre-trained model understanding context in a bidirectional way, setting new NLP standards.
GPT Series (Generative Pre-trained Transformer)	2018 - Present	Demonstrated the power of large-scale transformer models in text generation.

Bag of Words

Vocabulary: `[the, cat, sat, on, mat, dog]`

"The cat sat on the mat." → [2, 1, 1, 1, 1, 0]

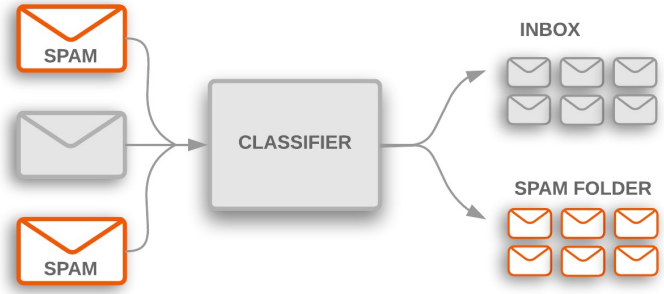
"The dog sat on the mat." → [2, 0, 1, 1, 1, 1]

BoW ignores word order, so both sentences have similar vectors despite different meaning.

- **Applications and Limitations**
  - **Applications**: Text classification, spam detection, information retrieval.
  - **Limitations**:
    - Ignores word order
    - Loss of syntactic and semantic context.
    - Challenges with synonyms and polysemy.
- More common in pre-deep learning era.

Spam Detection

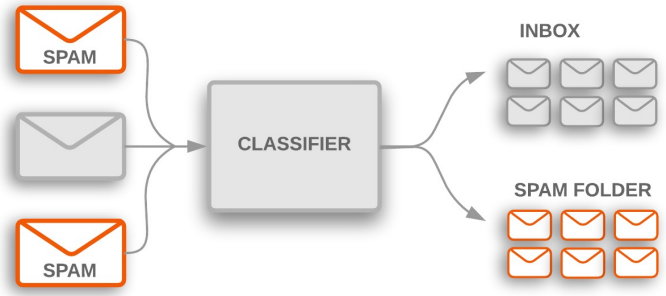
- **Original Emails:**
  - **Email 1** (Spam): "Win money win prizes now"
  - **Email 2** (Ham): "Project meeting at noon"
  - **Email 3** (Spam): "Call now for a free prize"
  - **Email 4** (Ham): "Are we meeting today?"
- Each email is preprocessed to remove common stop words (e.g., "for", "a", "the")
  - **Email 1**: "win money prizes now"
  - **Email 2**: "project meeting noon"
  - **Email 3**: "call now free prize"
  - **Email 4**: "meeting today"
- Vocabulary:
  - `{win, money, prizes, now, project, meeting, noon, call, free, prize, today}`



Source: developers.google.com/machine-learning/guides/text-classification

Spam Detection

- Vocabulary:
  - `{win, money, prizes, now, project, meeting, noon, call, free, prize, today}`
- Emails:
  - **Email 1**: "win money prizes now" ****(Spam)****
  - **Email 2**: "project meeting noon" ****(Not Spam)****
  - **Email 3**: "call now free prize" ****(Spam)****
  - **Email 4**: "meeting today" ****(Not Spam)****
- Vectorization**:**
  - **Vector for Email 1**: [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]
  - **Vector for Email 2**: [0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0]
  - **Vector for Email 3**: [0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0]
  - **Vector for Email 4**: [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1]



Source: developers.google.com/machine-learning/guides/text-classification

BoW vectors are input to a classifier trained to label emails as spam or not.

Enhancing BoW with TF-IDF

- **TF-IDF** balances word importance:
  - **Term Frequency (TF):** How often a word appears in a document
  - **Inverse Document Frequency (IDF):** How rare the word is across all documents
- **Purpose:**
  - Down-weights common words like “**the**”, “**and**”, “is**”**
  - Highlights task-specific, informative words

Enhancing BoW with TF-IDF

$$\text{TF-IDF}(w, d) = \text{TF}(w, d) \times \log\left(\frac{N}{\text{DF}(w)}\right)$$

Components:

- **TF(w, d):** Term Frequency
  - How many times word w appears in document d
- **DF(w):** Document Frequency
  - Number of documents that contain word w
- **N:** Total number of documents in the corpus
- **IDF(w):** Inverse Document Frequency
  - Penalizes very common words:

$$\text{IDF}(w) = \log\left(\frac{N}{\text{DF}(w)}\right)$$

So TF-IDF **highlights words that are frequent in a document but rare across documents**.

- **High TF + High IDF** = Important word (e.g., “neural” in a research paper)
- **High TF + Low IDF** = Common word (e.g., “the” → down-weighted)
- **Low TF** = Likely not very relevant, even if rare

Why Use log in IDF?

- **To Damp the Effect of Rare Words**
  - Without log, rare words would get extremely high IDF scores.
    - Example:

$$\text{IDF}(\text{word}) = \frac{10,000}{1} = 10,000 \text{ (too extreme)}$$

    - With log:

$$\log\left(\frac{10,000}{1}\right) \approx 4$$

  - So **log smooths the scale** — it keeps rare words important, but **not overwhelmingly so**.
- **To Ensure Non-Negative and Scaled Values**
  - If a word appears in all documents:

$$\text{IDF} = \log\left(\frac{N}{N}\right) = \log(1) = 0$$

  Meaning the word is **not informative** (e.g., “the”, “and”).

TF-IDF Example

- **Documents:**
  - **Doc 1**: "the cat sat on the mat"
  - **Doc 2**: "the dog sat on the log"
- **Vocabulary:**
  - [the, cat, sat, on, mat, dog, log]

$$\text{TF-IDF}(w, d) = \text{TF}(w, d) \times \log\left(\frac{N}{\text{DF}(w)}\right)$$

Term Frequency (TF)

Word	TF in Doc 1	TF in Doc 2
the	2	2
cat	1	0
sat	1	1
on	1	1
mat	1	0
dog	0	1
log	0	1

Document Frequency (DF)
(How many documents contain the word)

Word	DF
the	2
cat	1
sat	2
on	2
mat	1
dog	1
log	1

TF-IDF Example

- **Documents:**
  - **Doc 1**: "the cat sat on the mat"
  - **Doc 2**: "the dog sat on the log"
- **Vocabulary:**
  - [the, cat, sat, on, mat, dog, log]

$$\text{TF-IDF}(w, d) = \text{TF}(w, d) \times \log\left(\frac{N}{\text{DF}(w)}\right)$$

Term Frequency (TF)

Word	TF in Doc 1	TF in Doc 2
the	2	2
cat	1	0
sat	1	1
on	1	1
mat	1	0
dog	0	1
log	0	1

Document Frequency (DF)
(How many documents contain the word)

Word	DF	IDF
the	2	log(2/2) = 0
cat	1	log(2/1) ≈ 0.693
sat	2	log(2/2) = 0
on	2	log(2/2) = 0
mat	1	log(2/1) ≈ 0.693
dog	1	log(2/1) ≈ 0.693
log	1	log(2/1) ≈ 0.693

TF-IDF Example

- **Documents:**
  - **Doc 1**: "the cat sat on the mat"
  - **Doc 2**: "the dog sat on the log"
- **Vocabulary:**
  - [the, cat, sat, on, mat, dog, log]

$$\text{TF-IDF}(w, d) = \text{TF}(w, d) \times \log\left(\frac{N}{\text{DF}(w)}\right)$$

For Doc 1, TF-IDF is:

$$[0, 0.693, 0, 0, 0.693, 0, 0]$$

But Bow is:

$$[2, 1, 1, 1, 1, 0, 0]$$

TF-IDF Example

- **Documents:**
  - **Doc 1**: "the cat sat on the mat"
  - **Doc 2**: "the dog sat on the log"
- **Vocabulary:**
  - [the, cat, sat, on, mat, dog, log]

$$\text{TF-IDF}(w, d) = \text{TF}(w, d) \times \log\left(\frac{N}{\text{DF}(w)}\right)$$

For Doc 1, TF-IDF is:

$$[0, 0.693, 0, 0, 0.693, 0, 0]$$

But Bow is:

$$[2, 1, 1, 1, 1, 0, 0]$$

TF-IDF Fixes This:

- **Down-weights frequent, uninformative words** like “the”, “on”
- **Up-weights rare, meaningful words** like “cat”, “mat”
- The result: feature vectors are **more focused on informative content**

Probabilistic Language Model

- Purpose:
  - To calculate the probability of a sequence of words:

$$P(\mathrm{W}) = P(w_1, w_2, w_3, w_4, w_5, \ldots, w_n)$$

"I love machine learning" → P(I, love, machine, learning)

- Related Problem (Common use case):
  - Predicting the likelihood of a subsequent word in a sequence:

$$P(w_5 | \, w_1, w_2, w_3, w_4)$$

- *Given "I love machine", what's the probability of "learning"?*

- Language Model (LM):
  - A model computing $P(\mathrm{W})$or $P(w_5 | \, w_1, w_2, w_3, w_4)$.

Probabilistic Language Model

- Purpose:
  - To calculate the probability of a sequence of words:

$$P(\mathrm{W}) = P(w_1, w_2, w_3, w_4, w_5, \ldots, w_n)$$

"I love machine learning" → P(I, love, machine, learning)

- Related Problem (Common use case):
  - Predicting the likelihood of a subsequent word in a sequence:

$$P(w_5 | \, w_1, w_2, w_3, w_4)$$

- *Given "I love machine", what's the probability of "learning"?*

- Language Model (LM):
  - A model computing $P(\mathrm{W})$ or $P(w_5 | \, w_1, w_2, w_3, w_4)$.
    - In other words, a model that assigns probabilities to word sequences, or predicts the next word based on previous words.

Input: w_1, w_2, w_3, w_4 → Model → Predict w_5

Computing Computing the Probability of a Sentence P(W)

Sentence: *"The cat sat on the mat"*

Goal: Compute the joint probability of the full sequence:

$$P(\text{The, cat, sat, on, the, mat}) = ?$$

Computing Computing the Probability of a Sentence P(W)

Sentence: *"The cat sat on the mat"*

Goal: Compute the joint probability of the full sequence:

P(The, cat, sat, on, the, mat) = ?

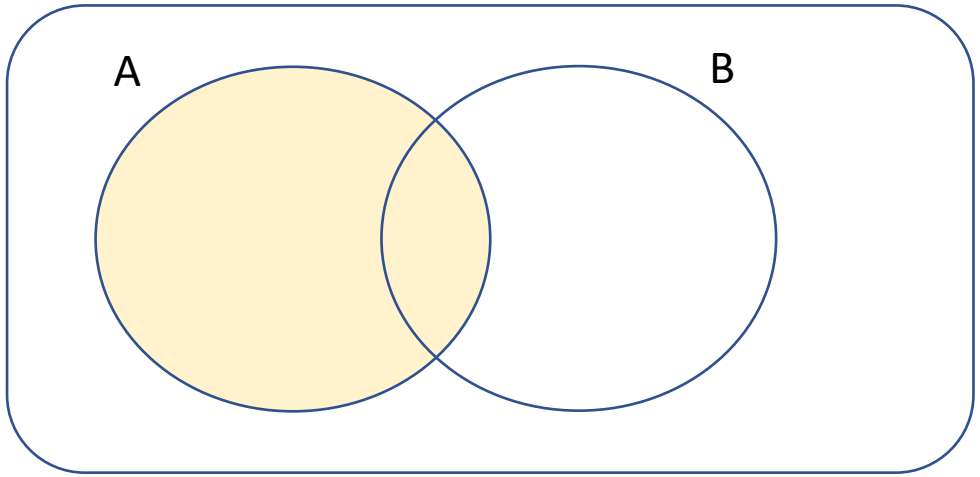
Using the Chain Rule of Probability:

$$P(W) = P(w_1) \cdot P(w_2 \mid w_1) \cdot \ldots \cdot P(w_6 \mid w_1, ..., w_5)$$

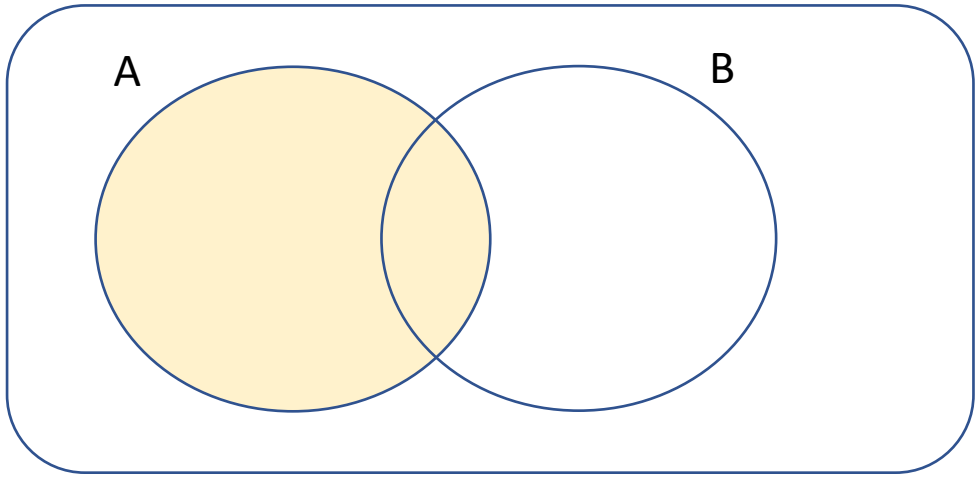
$$= P(\text{The}) \cdot P(\text{cat} \mid \text{The}) \cdot \ldots \cdot P(\text{mat} \mid \text{The}, ..., \text{the})$$

Understanding the Chain Rule

- **Conditional Probabilities Defined**:
  - Given: $\mathrm{P}(\mathrm{B}|\mathrm{A}) = \frac{\mathrm{P}(\mathrm{A},\mathrm{B})}{\mathrm{P}(\mathrm{A})}$

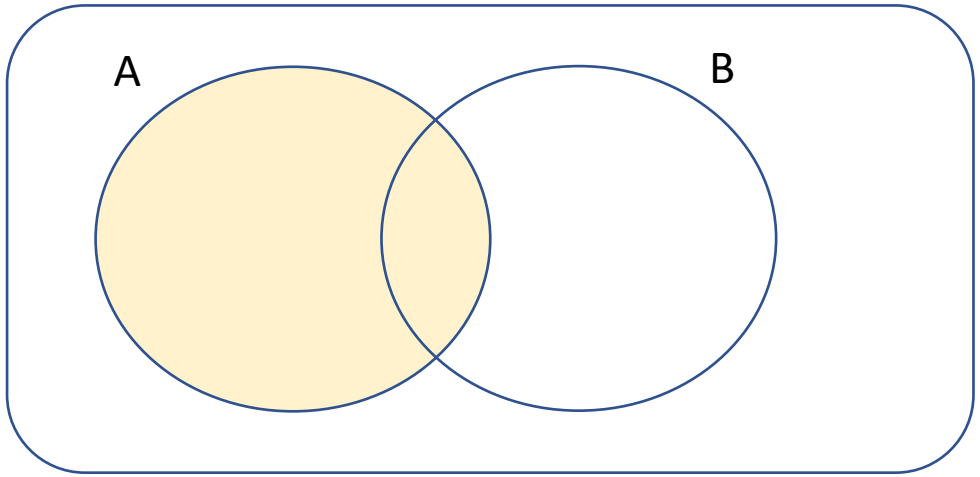


Understanding the Chain Rule



- **Conditional Probabilities Defined**:
  - Given: $\mathrm{P}(\mathrm{B}|\mathrm{A}) = \frac{\mathrm{P}(\mathrm{A},\mathrm{B})}{\mathrm{P}(\mathrm{A})}$
- **With More Variables**:
  - $\mathrm{P}(\mathrm{A}, B, C, D) = \mathrm{P}(\mathrm{A})\mathrm{P}(\mathrm{B}|\mathrm{A})\mathrm{P}(C|A, B)\mathrm{P}(D|\mathrm{A},\mathrm{B},\mathrm{C})$

Understanding the Chain Rule



- **Conditional Probabilities Defined**:
  - Given: $\mathrm{P}(\mathrm{B}|\mathrm{A}) = \frac{\mathrm{P}(\mathrm{A},\mathrm{B})}{\mathrm{P}(\mathrm{A})}$

- **With More Variables**:
  - $\mathrm{P}(\mathrm{A}, B, C, D) = \mathrm{P}(\mathrm{A})\mathrm{P}(\mathrm{B}|\mathrm{A})\mathrm{P}(C|A, B)\mathrm{P}(D|\mathrm{A},\mathrm{B},\mathrm{C})$

- **General Chain Rule Formula**:
  - $\mathrm{P}(\mathrm{X}_1,\mathrm{X}_2,\mathrm{X}_3,\ldots,\mathrm{X}_n) = \mathrm{P}(\mathrm{X}_1)\mathrm{P}(\mathrm{X}_2|\mathrm{X}_1)\mathrm{P}(\mathrm{X}_3|\mathrm{X}_1,\mathrm{X}_2)\ldots\mathrm{P}(\mathrm{X}_n|\mathrm{X}_1,\ldots,\mathrm{X}_{n-1})$

Product of conditional probabilities

Understanding the Chain Rule

- **General Chain Rule Formula**:
  - $P(X_1,X_2,X_3,\ldots,X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1,X_2)\ldots P(X_n|X_1,\ldots,X_{n-1})$

$$P(X_1,X_2,X_3,\ldots,X_n) = \prod_{i=1}^{n} P(X_i \mid X_1\ X_2\ \ldots\ X_{i-1})$$

Events before X_i

Understanding the Chain Rule

- **General Chain Rule Formula**:
  - $P(X_1,X_2,X_3,\ldots,X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1,X_2)\ldots P(X_n|X_1,\ldots,X_{n-1})$

$$P(X_1,X_2,X_3,\ldots,X_n) = \prod_{i=1}^{n} P(X_i \mid X_1\ X_2\ \ldots\ X_{i-1})$$

Events before X_i

To compute the probability of a sentence:

"I am happy"

We apply the chain rule:

$$P(I, am, happy) = P(I) \cdot P(am \mid I) \cdot P(happy \mid I, am)$$

Understanding the Chain Rule

- Use the Chain Rule to determine the joint probability of words in a sentence.
- **General Formula**:

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

  - Given a sentence: $P(w_1\ w_2\ \ldots\ w_n)$
  - Calculation: $P(w_1\ w_2\ \ldots\ w_n) = \prod_{i=1}^{n} P(w_i \mid w_1\ w_2\ \ldots\ w_{i-1})$

Understanding the Chain Rule

- Use the Chain Rule to determine the joint probability of words in a sentence.
- **General Formula**:

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

  - Given a sentence: $P(w_1\ w_2\ \ldots\ w_n)$
  - Calculation: $P(w_1\ w_2\ \ldots\ w_n) = \prod_{i=1}^{n} P(w_i \mid w_1\ w_2\ \ldots\ w_{i-1})$ → Prefix

Understanding the Chain Rule

- Use the Chain Rule to determine the joint probability of words in a sentence.
- **General Formula**:

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

  - Given a sentence: $P(w_1\ w_2\ \ldots\ w_n)$
  - Calculation: $P(w_1\ w_2\ \ldots\ w_n) = \prod_{i=1}^{n} P(w_i \mid w_1\ w_2\ \ldots\ w_{i-1})$ → Prefix

Each word depends on the **prefix (previous words)**.

Understanding the Chain Rule

- Use the Chain Rule to determine the joint probability of words in a sentence.
- **General Formula**:

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

  - Given a sentence: $P(w_1\ w_2\ \ldots\ w_n)$
  - Calculation: $P(w_1\ w_2\ \ldots\ w_n) = \prod_{i=1}^{n} P(w_i \mid w_1\ w_2\ \ldots\ w_{i-1})$ → Prefix

P(sat | cat, The)

Each word depends on the **prefix (previous words)**.

- **Example Calculation**:
  - Sentence: "The cat sat on the mat"
  - Probability Breakdown:
    P(The) *
    P(cat | The) *
    P(sat | The cat) *
    P(on | The cat sat) *
    P(the | The cat sat on) *
    P(mat | The cat sat on the)

How to Calculate These Probabilities?

"A sentence's probability is built one word at a time, conditioned on the words before it."

$$P(w_1\ w_2 \ldots w_n) = \prod_{i=1}^{n} P(w_i \mid w_1\ w_2 \ldots w_{i-1})$$

- **Example Calculation**:
  - Sentence: "The cat sat on the mat"
  - Probability Breakdown:
    P(The) *
    P(cat | The) *
    P(sat | The cat) *
    P(on | The cat sat) *
    P(the | The cat sat on) *
    P(mat | The cat sat on the)

How to Calculate These Probabilities?

$$P(w_1\, w_2 \ldots w_n) = \prod_{i=1}^{n} P(w_i \mid w_1\, w_2 \ldots w_{i-1})$$

How to Calculate These Probabilities?

$$P(B|A) = P(A,B)/P(A)$$

$$P(\text{mat} \mid \text{The cat sat on the}) = \frac{\text{Count(The cat sat on the mat)}}{\text{Count(The cat sat on the)}}$$

How to Calculate These Probabilities?

P(mat | The cat sat on the) =

How to Calculate These Probabilities?

$$P(\text{mat} \mid \text{The cat sat on the}) = \frac{\textbf{Count(The cat sat on the mat)}}{}$$

How to Calculate These Probabilities?

P(mat | The cat sat on the) =

$$\frac{\textbf{Count(The cat sat on the mat)}}{\textbf{Count(The cat sat on the)}}$$

How to Calculate These Probabilities?

$$P(\text{mat} \mid \text{The cat sat on the}) = \frac{\textbf{Count(The cat sat on the mat)}}{\textbf{Count(The cat sat on the)}}$$

Using the Chain Rule + Maximum Likelihood

How to Calculate These Probabilities?

$$P(\text{mat} \mid \text{The cat sat on the}) = \frac{\textbf{Count(The cat sat on the mat)}}{\textbf{Count(The cat sat on the)}}$$

Using the Chain Rule + Maximum Likelihood

- Too hard!
- Need huge amount of data.
- Longer prefixes → Lower counts

Using the Chain Rule + Maximum Likelihood

- To estimate **P(mat | The cat sat on the)**, MLE says:

$$P(w_i|\text{prefix}) = \frac{\text{Count}(\text{prefix} + \text{w}_\text{i})}{\text{Count}(\text{prefix})}$$

$$P(mat|Thecatsatonthe) = \frac{\text{Count}(\text{The cat sat on the mat})}{\text{Count}(\text{The cat sat on the})}$$

So, we're **just counting** how often that full phrase occurs, compared to the shorter prefix.

Why It Struggles?

- Rare or unseen sequences → zero counts → zero probability
- Needs **lots of data** for longer phrases (prefixes)

Markov Assumption in Language Models

$$P(w_1\, w_2 \ldots w_n) \approx \prod_i P(w_i \mid \boldsymbol{w_1 \ldots w_{i-1}})$$

Andrei Markov (1856~1922)

Markov Approximation (limit context to last *k* words)

$$P(w_1\, w_2 \ldots w_n) \approx \prod_i P(w_i \mid \boldsymbol{w_{i-k} \ldots w_{i-1}})$$

- Instead of using all previous words, we approximate using a **fixed number 'k'** of previous words.
- This approximation greatly reduces the computational complexity.

Markov Assumption in Language Models

$$P(w_1\, w_2 \ldots w_n) \approx \prod_i P(w_i \mid w_1 \ldots w_{i-1})$$

Special Case: Unigram Model (k = 1)

$$P(w_1\, w_2 \ldots w_n) \approx \prod_i P(w_i)$$

$$P(w_1\, w_2 \ldots w_n) \approx P(w_1)P(w_2)P(w_3) \ldots P(w_n), for\ k = 1$$

Andrei Markov (1856~1922)

Markov Assumption in Language Models

$$P(w_1\, w_2 \ldots w_n) \approx \prod_i P(w_i \mid w_1 \ldots w_{i-1})$$

Special Case: Unigram Model (k = 2)

$$P(w_1\, w_2 \ldots w_n) \approx \prod_i P(w_i \mid w_{i-1})$$

Andrei Markov (1856~1922)

$$P(w_1\, w_2 \ldots w_n) \approx P(w_1)P(w_2|w_1)P(w_3|w_2) \ldots P(w_n|w_{n-1}), for\ k = 2$$

Example:
With **bigrams**: P(The cat sat on the mat) is approximated as:

$$P(The)P(cat|The)P(sat|cat)P(on|sat)P(the|on)P(mat|the)$$

How to Calculate These Probabilities?

Andrei Markov (1856~1922)

Unigram

$$P(w_1\, w_2 \ldots w_n) \approx P(w_1)P(w_2)P(w_3) \ldots P(w_n), for\ k = 1$$

Bigram

$$P(w_1\, w_2 \ldots w_n) \approx P(w_1)P(w_2|w_1)P(w_3|w_2) \ldots P(w_n|w_{n-1}), for\ k = 2$$

Example:
With **bigrams**: P(The cat sat on the mat) is approximated as:

$$P(The)P(cat|The)P(sat|cat)P(on|sat)P(the|on)P(mat|the)$$

Why We Omit P(w1) in Bigram Models?

- **Start-of-Sentence Token (⟨s⟩)**
  - Sentences begin with a **special token**: ⟨s⟩
  - We don’t model P(⟨s⟩) → it's always present

$$P(\langle s \rangle, w_1, w_2, ..., w_n) = P(\langle s \rangle) \cdot P(w_1 \mid \langle s \rangle) \cdot P(w_2 \mid w_1) \cdot \ldots \cdot P(w_n \mid w_{n-1})$$

Since $P(< s >) = 1$:

$$P(w_1, w_2, ..., w_n) \approx P(w_1 \mid \langle s \rangle) \cdot P(w_2 \mid w_1) \cdot \ldots \cdot P(w_n \mid w_{n-1})$$

Step-by-Step Bigram Probability Estimation

- **Example**:
  - **Sentence**: *"The cat sat on the mat"*
  - *Extracted Bigrams:*
    - *(The, cat)*
    - *(cat, sat)*
    - *(sat, on)*
    - *(on, the)*
    - *(the, mat)*

$c(The) = 1000,\ c(The, cat) = 50$

$c(cat) = 300,\ c(cat, sat) = 60$

$c(sat) = 250,\ c(sat, on) = 40$

$c(on) = 400,\ c(on, the) = 80$

$c(the) = 1500,\ c(the, mat) = 35$

1. $P(cat|The) = \frac{c(The,cat)}{c(The)} = \frac{50}{1000} = 0.05$
2. $P(sat|cat) = \frac{c(cat,sat)}{c(cat)} = \frac{60}{300} = 0.20$
3. $P(on|sat) = \frac{c(sat,on)}{c(sat)} = \frac{40}{250} = 0.16$
4. $P(the|on) = \frac{c(on,the)}{c(on)} = \frac{80}{400} = 0.20$
5. $P(mat|the) = \frac{c(the,mat)}{c(the)} = \frac{35}{1500} = 0.0233$

$P(\text{The cat sat on the mat}) = P(\text{"The"}) \times 0.05 \times 0.20 \times 0.16 \times 0.20 \times 0.0233$

$$P(The\ cat\ sat\ on\ the\ mat) \approx P(The)P(cat|The)P(sat|cat)P(on|sat)P(the|on)P(mat|the)$$

$$P(The, cat) = P(cat|The) = \frac{Count(The, cat)}{Count(The)}$$

Approximating Shakespeare

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

Source: Speech and Language Processing (3rd edition), Chapter 3 Language Modeling with N-grams, Figure 3.3, available at http://web.stanford.edu/~jurafsky/slp3/3.pdf

Approximating Wall Street Journal

1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Source: Speech and Language Processing (3rd edition), Chapter 3 Language Modeling with N-grams, Figure 3.3, available at http://web.stanford.edu/~jurafsky/slp3/3.pdf

Problems with n-gram Language Models

1.Data Sparsity

$$P(\text{mat} \mid \text{The cat sat on the}) = \frac{\text{Count(The cat sat on the mat)}}{\text{Count(The cat sat on the)}}$$

If this is 0, then P is 0.

Problems with n-gram Language Models

- **Data Sparsity**

$$P(\text{mat} \mid \text{The cat sat on the}) = \frac{\text{Count(The cat sat on the mat)}}{\text{Count(The cat sat on the)}}$$

If this is 0, then P is 0.

There are some imperfect smoothing solutions for this (beyond the scope of our course)

Problems with n-gram Language Models

- **Storage Size**

$$P(\text{mat} \mid \text{The cat sat on the}) = \frac{\text{Count(The cat sat on the mat)}}{\text{Count(The cat sat on the)}}$$

How many of these counts should we store?

Problems with n-gram Language Models

- **Storage Size**

$$P(\text{mat} \mid \text{The cat sat on the}) = \frac{\text{Count(The cat sat on the mat)}}{\text{Count(The cat sat on the)}}$$

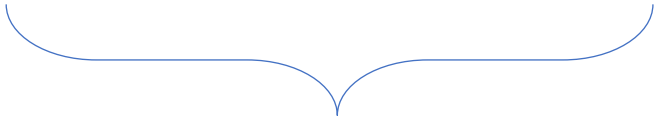
How many of these counts should we store?

For n-grams with k words, we should store $O(k^n)$

Problems with n-gram Language Models

- **Context is limited to n words**

"Remember the white cute cat, which was saw yesterday? He suddenly decided to sit on the ..."



n=7

$$P(\text{mat} \mid \text{He suddenly decided to sit on the }) = ?$$

Problems with n-gram Language Models

- **Disconnection Between N-grams**

These would all be separate and unrelated n-grams, while they have very close meanings.

- "The kitten settled on the rug."
- "The cat rested on the carpet."
- "The pet lounged on the doormat."

Summary of n-grams

N-grams extend the Bag-of-Words (BoW) model by capturing local word order in fixed-size windows.

However, like BoW, they **do not capture grammar or long-range dependencies**, limiting their understanding of context.

(Modern language models address these limitations using deeper contextual representations.)