Building Large Language Model Applications

Advanced NLP Techniques: N-grams and Word Embeddings

Hamza Farooq
Dr. Saima Hassan

Recap of Lecture 3

Language representation

- **One-Hot Encoding** – Represents each word as a unique binary vector, ignoring word relationships.
- **Bag-of-Words** – Represents text as word frequency counts, disregarding word order.
- **TF-IDF** – Measures word importance by balancing frequency with document uniqueness.

Limitations:

- **TF-IDF** treats words independently, ignoring the sequence or relationships between words.
- Cannot capture context or meaning in phrases.

Learning outcomes

- N-gram
- Word Embedding
    - Word2Vec
        - CBOW
        - Skip-gram
    - Glove
- Conclusion

Next Word Prediction

What is Next Word Prediction?

Next word prediction is a fundamental concept in NLP where **a model predicts the most probable word following a given sequence of words**

How It Works:

- The model analyzes **previous words** in a sentence.
- It predicts the **next word** based on learned patterns from large text corpora.
- Techniques include **N-grams, Word2Vec, Neural Networks**, and **Transformer-based models (GPT, BERT,** etc.).

Example:"The weather is very..."
Prediction:

1. "nice" ☀️
2. "cold" ❄️
3. "hot" 🔥

Real-world applications: *Search engines, text messaging, AI-powered writing assistants.*

Human Word Prediction

How Do Humans Predict Words?

Humans have the ability to imagine the next word in a sentence based on various types of knowledge:

Domain Knowledge – Understanding common phrases in specific fields.

- ***Example:*** "Red blood ___" → *cells, count, pressure*

Syntactic Knowledge – Recognizing grammatical patterns.

- ***Example:*** "The ___" → *adj (beautiful), noun (dress/location)*

Lexical Knowledge – Knowing which words frequently appear together.

- ***Example:*** "Baked ___" → *potato, bread, goods*

Applications

Predicting words based on context is essential for various applications:

Autocorrect & Text Suggestions

*"He will **recieve**/**receive** the email soon."*

Speech Recognition

"I ate a cherry" is a more likely sentence than "Eye eight uh Jerry"

Handwriting Recognition

*"Order 3 more **bottles**/**battles** of water."*

Machine Translation

- ***English:*** "She has a big heart."
- ***French:*** "Elle a un grand cœur." (instead of "cardiaque"

Probabilistic Language Models

Assign a probability to a sentence

- **Machine Translation:**

  P(**high** winds tonight) > P(**large** winds tonight)

- **Spell Correction**

  The office is about fifteen minuets from my house

  P(about fifteen **minutes** from) > P(about fifteen **minuets** from)

- **Speech Recognition**

  P(I saw a van) > P(eyes awe of an)

Summarization, question-answering, etc., etc.!!

Probabilistic Language Models

Goal: Compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1, w_2, w_3, w_4, w_5 \dots w_n)$$

Probability of an upcoming word:

$$P(w_n \mid w_1, w_2, \dots w_{n-1})$$

What is a Language Model?

A model that estimates:

$P(W)$ → Probability of a full sentence

$P(w_n \mid w_1, w_2, \dots w_{n-1})$ → Probability of the next word

How to Compute P(W)?

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

To compute the probability of a sequence of words **P(W)**, we use the **Chain Rule of Probability**.

Chain Rule of Probability:

Recall the definition of **conditional probability**:

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

More variables:

$$P(A,B,C,D) \quad = \quad P(A)\, P(B|A)\, P(C|A,B)\, P(D|A,B,C)$$

The Chain Rule in General Form:

For a sequence of words (w_1, w_2, w_3, ..., w□), we expand using the Chain Rule:

$$P(w_1, w_2, w_3, w_4, w_5 \ldots w_n) = P(w_1)\, P(w_2|w_1)\, P(w_3|w_1,w_2) \ldots P(w_n|w_1,w_2, \ldots,w_{n-1})$$

Chain Rule Application

Applying the Chain Rule to compute joint probability of words in sentence:

$$P(w_1, w_2, w_3, w_4, w_5 \ldots w_n) = P(w_n|w_1,w_2, \ldots,w_{n-1}) =$$

$$= \prod_{k=1}^{n} P(w_k|w_1,w_2, \ldots,w_{k-1})$$

The joint probability of an entire sequence of words can be estimated by multiplying together a number of conditional probabilities.

Markov Assumption

The intuition of the n-gram model is that instead of computing the probability of a word given its entire history, we can approximate the history by just the last few words.

P(delicious | The cake with chocolate frosting looks absolutely)

Bigram model

P(delicious | absolutely)

When using a bigram model to predict the conditional probability of the next word, make the following approximation:

$$\mathbf{P(w_n \mid w_{n-1})}$$

Generalize the bigram to the trigram and then to the n-gram

Language Models

A language model is a machine learning model **LM** that predicts upcoming words.

A **LM** assigns a probability to each possible next word.

What is an N-gram?

N-gram is the simplest kind of **LM**

Or A sequence of n words used in language modeling

Types of N-Grams:

- 📌 **Unigram (1-gram):** "Learning"
- 📌 **Bigram (2-gram):** "Machine learning"
- 📌 **Trigram (3-gram):** "Deep learning models"
- 📌 **4-gram:** "Artificial intelligence is evolving"

N-gram Example

"She enjoys drinking hot coffee."

Bigram Representation:

("She enjoys"), ("enjoys drinking"), ("drinking hot"), ("hot coffee")

Trigram Representation:

("She enjoys drinking"), ("enjoys drinking hot"), ("drinking hot coffee")

N-grams: Examples

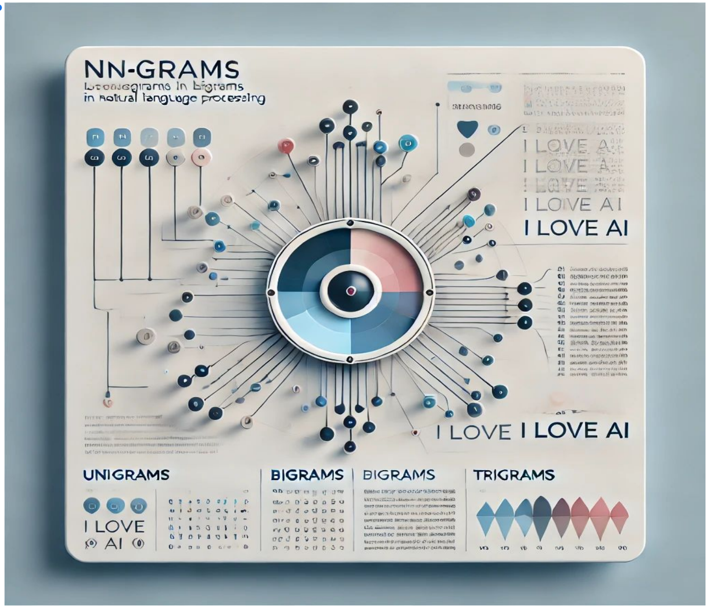
"In language modeling, language modeling is essential."

N-Gram Breakdown:

Unigram (N=1): [‘In’, ‘language’, ‘modeling’, ‘language’, ‘modeling’, ‘is’, ‘essential’]

Bigram (N=2): [‘In language’, ‘language modeling’, ‘modeling language’, ‘language modeling’, ‘modeling is’, ‘is essential’]

Trigram (N=3): [‘In language modeling’, ‘language modeling language’, ‘modeling language modeling’, ‘language modeling is’, ‘modeling is essential’]



N-grams: Model Formulas

Word sequences

$$w_1^n = w_1 \ldots w_n$$

Chain rule of probability

$$P(w_1^n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)\ldots P(w_n \mid w_1^{n-1}) = \prod_{k=1}^{n} P(w_k \mid w_1^{k-1})$$

Bigram approximation

$$P(w_1^n) = \prod_{k=1}^{n} P(w_k \mid w_{k-1})$$

N-gram approximation

$$P(w_1^n) = \prod_{k=1}^{n} P(w_k \mid w_{k-N+1}^{k-1})$$

Estimating Probabilities

N-gram conditional probabilities can be estimated from raw text based on the ***relative frequency*** of word sequences.

Bigram:

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1} w_n)}{C(w_{n-1})}$$

N-gram:

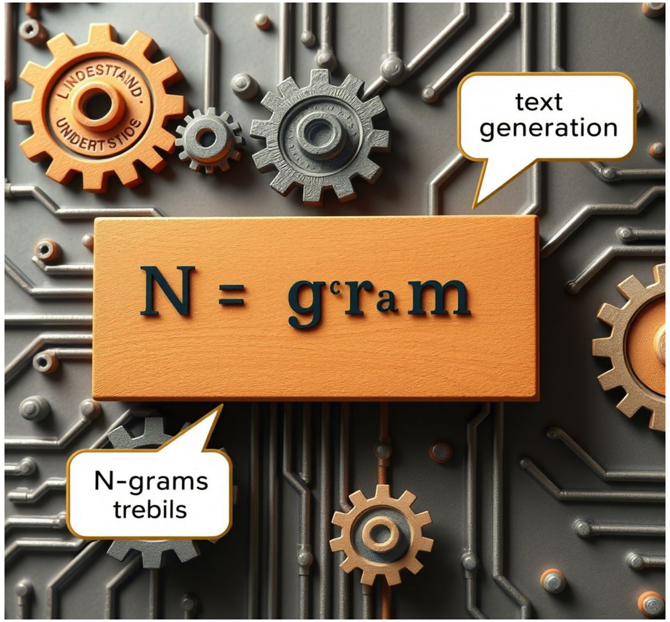
$$P(w_n \mid w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1} w_n)}{C(w_{n-N+1}^{n-1})}$$

To have a consistent probabilistic model, append a unique start (<s>) and end (</s>) symbol to every sentence and treat these as additional words.

N-grams: Probability Calculation Example

<S> I am Sam </S>
<S> Sam I am </S>
<S> I do not like green eggs and jam </S>

Some of the bigram probabilities from this corpus can be as:



$P(\mathrm{I}|\langle S\rangle) = 2/3 = 0.67$
$P(\mathrm{Sam}|\langle S\rangle) = 1/3 = 0.33$
$P(\mathrm{am}|\mathrm{I})$ 2/3 = 0.67

$P(\langle /S\rangle|\mathrm{Sam}) = 1/2 = 0.5$
$P(\mathrm{Sam}|\mathrm{am}) = 1/2 = 0.5$
$P(\mathrm{do}|\mathrm{I}) = 1/3 = 0.33$

N-grams: Limitations

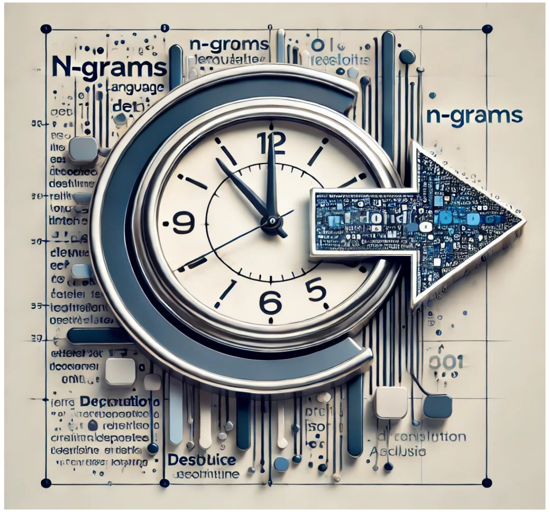
Lacks Long-Range Context: N-grams only consider neighboring words, ignoring distant dependencies.

Data Sparsity: Higher-order N-grams often have insufficient data for accurate probabilities.

Exponential Growth: Vocabulary size increases rapidly with larger N, requiring more storage and computation.

No Semantic Understanding: N-grams rely on word patterns without understanding meaning.

Fixed Window Size: Important context outside the N-gram window is ignored.



Word Embeddings

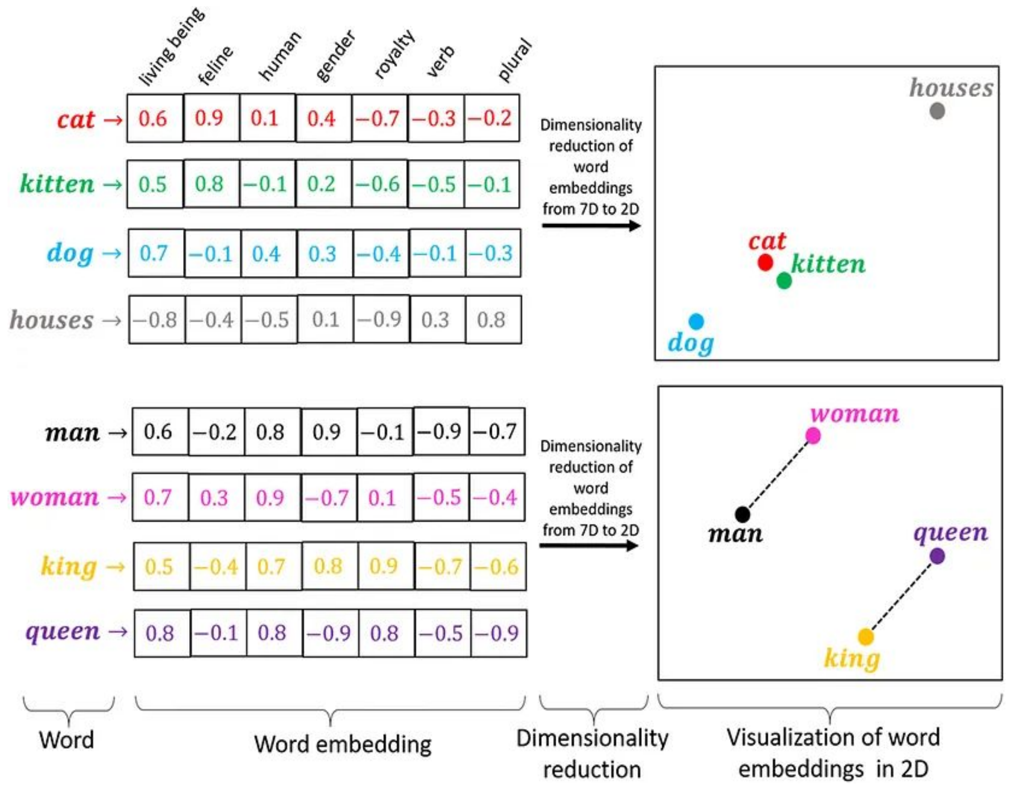
From One-Hot Encoding to Continuous Representation

$$\begin{array}{cccccc}
\text{Man} & \text{Woman} & \text{King} & \text{Queen} & \text{Apple} & \text{Orange} \\
(5391) & (9853) & (4914) & (7157) & (456) & (6257) \\
\begin{bmatrix}0\\0\\0\\0\\\vdots\\1\\\vdots\\0\\0\end{bmatrix} &
\begin{bmatrix}0\\0\\0\\0\\0\\\vdots\\1\\\vdots\\0\end{bmatrix} &
\begin{bmatrix}0\\0\\0\\\vdots\\1\\\vdots\\0\\0\\0\end{bmatrix} &
\begin{bmatrix}0\\0\\0\\0\\0\\\vdots\\1\\\vdots\\0\end{bmatrix} &
\begin{bmatrix}0\\\vdots\\1\\\vdots\\0\\0\\0\\0\\0\end{bmatrix} &
\begin{bmatrix}0\\0\\0\\0\\0\\\vdots\\1\\\vdots\\0\end{bmatrix}
\end{array}$$

Word Embedding

What is Word Embedding?

- Word Embedding is a technique for **representing words and documents in a numerical format**.
- It transforms **words into low-dimensional vectors** that capture their **meaning and relationships.**
- **Words with similar meanings have similar vector representations**, making them useful for AI models.



	living being	feline	human	gender	royalty	verb	plural
cat →	0.6	0.9	0.1	0.4	−0.7	−0.3	−0.2
kitten →	0.5	0.8	−0.1	0.2	−0.6	−0.5	−0.1
dog →	0.7	−0.1	0.4	0.3	−0.4	−0.1	−0.3
houses →	−0.8	−0.4	−0.5	0.1	−0.9	0.3	0.8

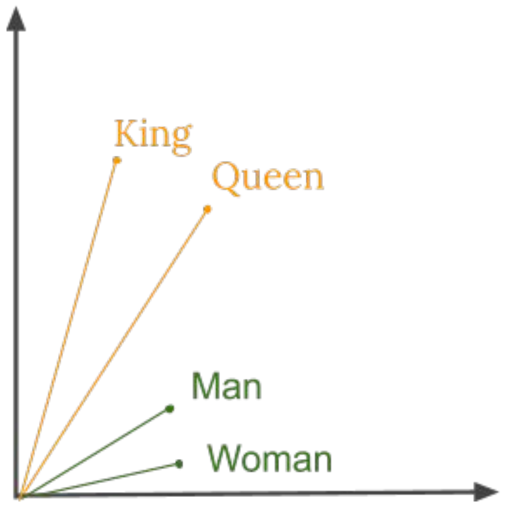
	living being	feline	human	gender	royalty	verb	plural
man →	0.6	−0.2	0.8	0.9	−0.1	−0.9	−0.7
woman →	0.7	0.3	0.9	−0.7	0.1	−0.5	−0.4
king →	0.5	−0.4	0.7	0.8	0.9	−0.7	−0.6
queen →	0.8	−0.1	0.8	−0.9	0.8	−0.5	−0.9

Dimensionality reduction of word embeddings from 7D to 2D
houses
cat
kitten
dog
Dimensionality reduction of word embeddings from 7D to 2D
woman
man
queen
king
Word
Word embedding
Dimensionality reduction
Visualization of word embeddings in 2D

https://web.engr.oregonstate.edu/~huanlian/teaching/ML/2024fall/unit4/word_embeddings.html

Word Embedding

- **Why Use Word Embeddings?**
  - Reduce Dimensionality
  - Overcomes limitations of one-hot encoding and TF-IDF.
  - Enables understanding of relationships like:
    - "king - man + woman = queen"
    - Synonyms and similar words are closer in vector space.
  - Efficient for NLP tasks like classification, translation, and more.
- **Applications:**
  - Sentiment Analysis, Question Answering, Machine Translation



Word2Vec

Word2vec is a technique in NLP for obtaining vector representations of words. These vectors capture information about the meaning of the word based on the surrounding words. The word2vec algorithm estimates these representations by modeling text in a large corpus. Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence. Word2vec was developed by Tomáš Mikolov and colleagues at Google and published in 2013

Word2vec is a two-layer network where there is input one hidden layer and output

Word2Vec

Word2Vec -learn through training on a large text corpus. The features are learned automatically based on the word’s context in the corpus.

1. Each word starts as a one-hot vector

For a vocabulary of 10k words, Each word is represented as a 10k-dimensional one-hot vector

2. Hidden Layer (Embedding Layer)

The one-hot vector is multiplied by a weight matrix (size: V x N where V = vocabulary size, N = embedding size).

The resulting output is a low-dimensional dense vector (word embedding).

The values in this vector act as the features of the word.

3. Features are learned through training

Word2Vec uses either CBOW or Skip-gram.

The model adjusts the weight matrix based on how words co-occur in a given context.

Word2Vec

Example:

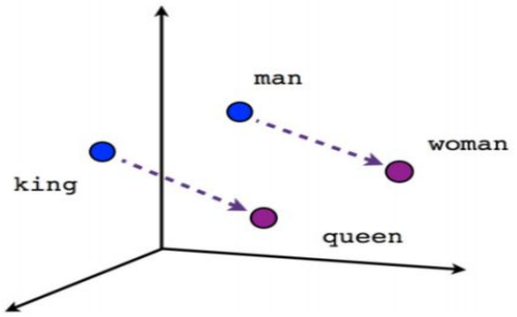
Let's say the word "**king**" gets converted to a 100-dimensional vector:

king=[0.21,-0.12,0.75,...,0.34]

These values represent different hidden features such as:

Gender, Royalty,Age ...

If we compare this with "**queen**", their vectors would be similar but differ in specific features (e.g., gender).



Word2Vec

- Instead of entire documents, Word2Vec uses words ***k*** positions away from each **center word.**
    - These words are called **context words**

Example for **k=3** Sentence:

*"The patient **complained of severe chest pain and shortness** of breath."*

- Word2Vec considers all words as center words, and all their context words

Types of Word Embeddings

1. **Word2Vec** – **A neural network-based model that learns word representations by analyzing context.**
   a. **Skip-gram:** Predicts surrounding words given a target word.
   b. **CBOW (Continuous Bag of Words):** Predicts a target word based on surrounding words.

2. **GloVe (Global Vectors for Word Representation)**

   Based on word co-occurrence statistics.
   Captures both global (document-wide) and local (sentence-level) relationships.
   Generates word vectors by analyzing word frequency patterns in large corpora.

Word2Vec

Generates word embeddings by predicting word relationships.

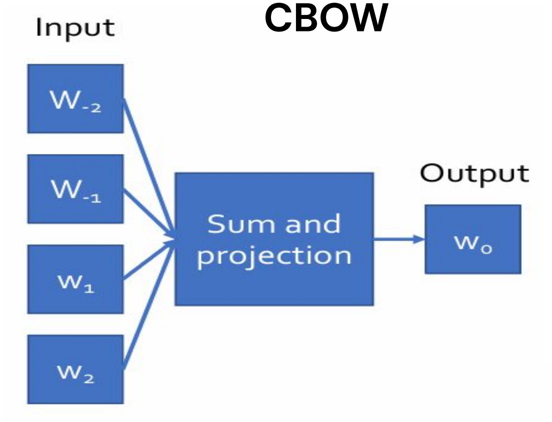
Two architectures:

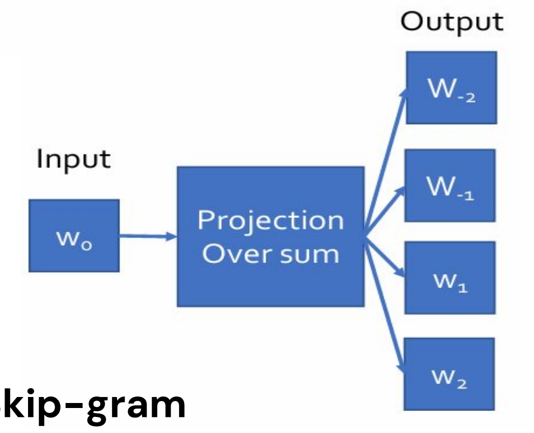
- **Continuous Bag of Words (CBOW):** Predicts a word from its context.
- **Skip-gram:** Predicts the context from a given word.

Applications:

- Text classification.
- Document similarity.

CBOW





Skip-gram

Word2Vec: Continuous Bag of Words (CBOW)

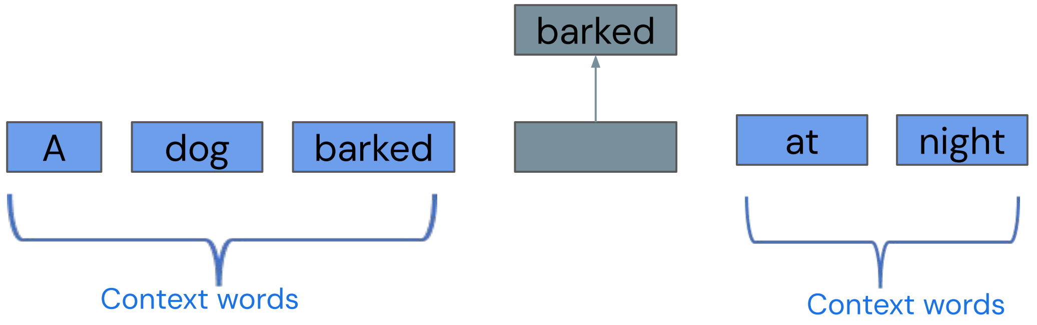
Predict a target word w_t from its surrounding context words.
Efficient for frequent words.

Example:

- Sentence: A dog barked loudly at night.
  - **Context words:** “dog," “barked," “at," “night."
  - **Predict:** “loudly.“

Mathematical Objective:

$$\text{Maximize: } P(w_t | w_{t-n}, ..., w_{t+n})$$

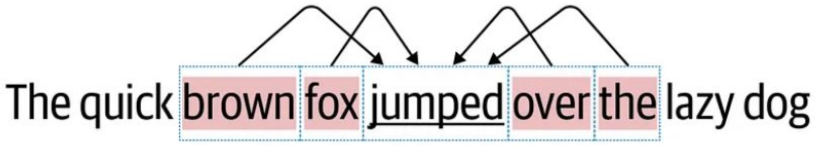


How Does CBOW Work?

Sentence: “The quick brown fox jumped over the lazy dog.”

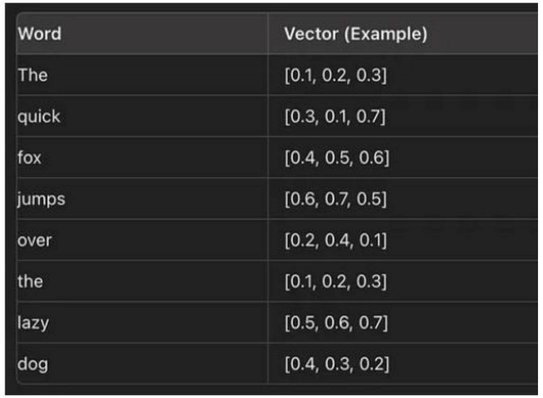
Predicted word = jumped
context size *k* = 2

Context words = “brown,” “fox,” “over,” and “the.”



After setting k=2, our window size (2k+1) is 5 (2 words before + target word + 2 words after).

Convert Words to Vectors
CBOW works with numbers, not words. So, each word in the sentence is represented by a vector of numbers. This is where the **embedding matrix** comes in.



Word	Vector (Example)
The	[0.1, 0.2, 0.3]
quick	[0.3, 0.1, 0.7]
fox	[0.4, 0.5, 0.6]
jumps	[0.6, 0.7, 0.5]
over	[0.2, 0.4, 0.1]
the	[0.1, 0.2, 0.3]
lazy	[0.5, 0.6, 0.7]
dog	[0.4, 0.3, 0.2]

Fetch Vectors for Context Words

Fetch the vectors for the context words“quick,” “fox,” “over,” and “the” from the embedding matrix:

- quick: [0.3, 0.1, 0.7]
- fox: [0.4, 0.5, 0.6]
- over: [0.2, 0.4, 0.1]
- the: [0.1, 0.2, 0.3]

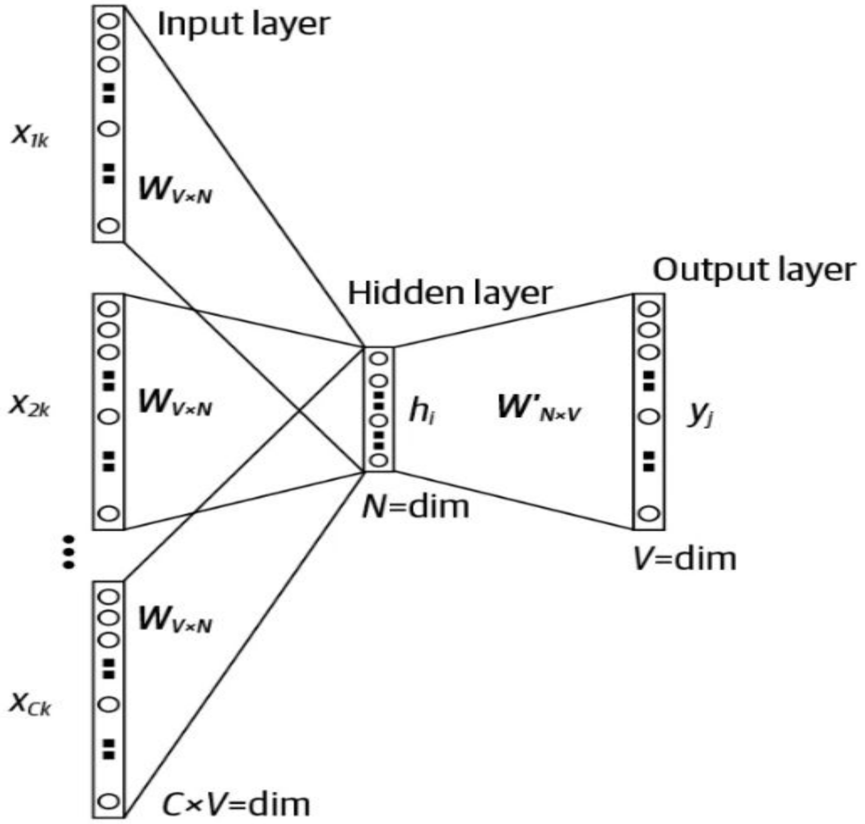
Combine the Context Vectors

Next, combine these vectors by adding them together:
[quick + fox + over + the]=[0.3,0.1,0.7]+[0.4,0.5,0.6]+[0.2,0.4,0.1]+[0.1,0.2,0.3]
Resulting combined vector (w(t)) => [1.0,1.2,1.7]

Pass Through the Network

1. **Input Layer:** The combined vector [1.0, 1.2, 1.7] is input into the network.
2. **Hidden Layer:** The input vector is multiplied by another matrix (E′), transforming it into a new vector.
3. **Output Layer:** The new vector is passed through a softmax function, which assigns probabilities to all the words in the vocabulary, indicating how likely each word is to be the target word "jumps."

After training on many such sentences, the model would correctly predict **"jumps"** as the **target word**.



Word2Vec: Skip-gram

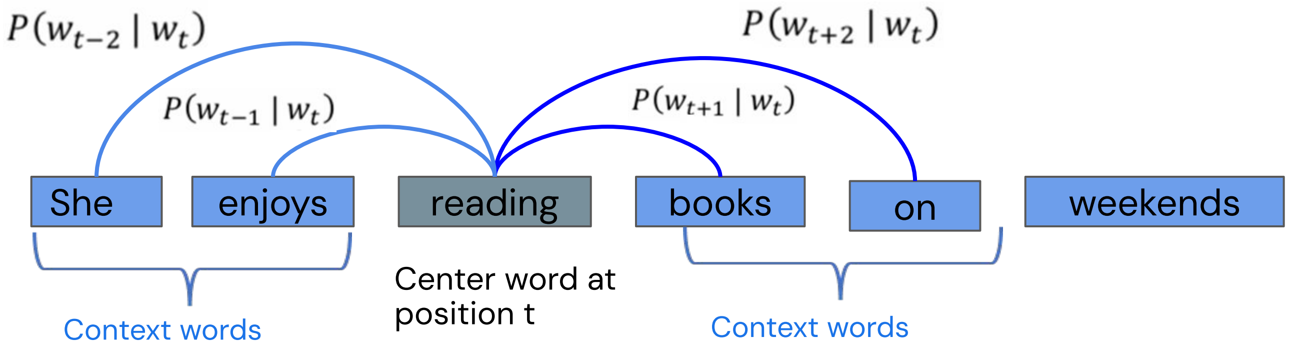
Predicts the surrounding **context** given a **target** word.

Example:

- Sentence: She enjoys reading books on weekends.
  - **Center word**: "reading."
  - **Context words for a window size 2**: “She”, “enjoy", “books“, “on”

Mathematical Objective:

$$\text{Maximize: } \prod_{t=1}^{T} \prod_{-n \leq j \leq n, j \neq 0} P(w_{t+j} | w_t)$$



CBOW vs Skip-gram

Feature	**Skip-gram**	**CBOW**
Input	Target word	Context words
Output	Context words	Target word
Speed	Slower	Faster
Strength	Rare words	Frequent words

Pros of Word2Vec

Captures Word Relationships – Words with similar meanings have similar vector representations.

Low-Dimensional Representation – Uses dense vectors instead of large, sparse one-hot vectors, making it memory-efficient.

Self-Supervised Learning – Does not require labeled data; learns patterns from raw text, making data collection easy.

Cons of Word2Vec

Does Not Preserve Global Information – Word2Vec focuses on local word relationships but does not capture overall document structure.

Solved by GloVe

Limited for Morphologically Rich Languages – Struggles with languages where words have multiple inflected forms (e.g., Arabic, Turkish, Finnish).

Solved by FastText

Lacks Broad Context Awareness – Embeddings are static, meaning the same word has the same representation regardless of context (e.g., "bank" in "river bank" vs. "money bank").

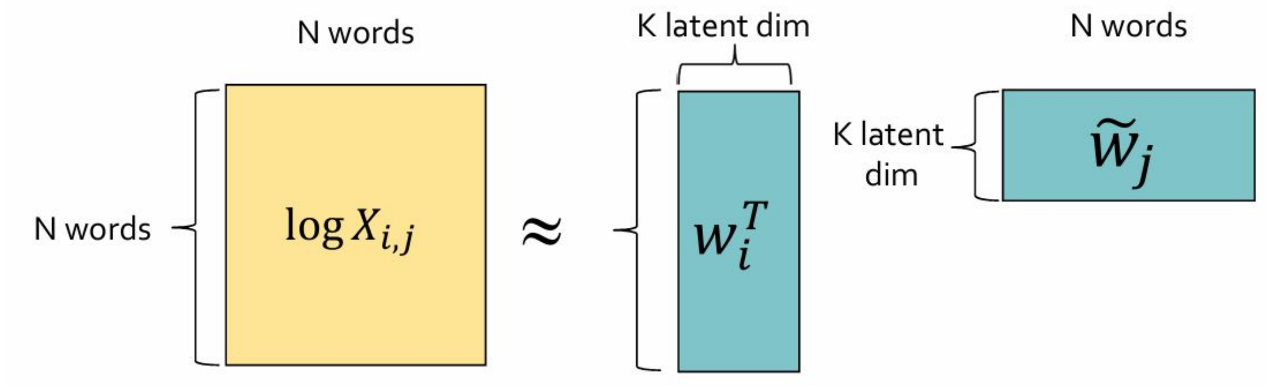
Solved by GPT, BERT, LSTM

GloVe: Global Vectors for Word Representation

While **word2Vec** is a predictive model — learning vectors to improve the predictive ability, **GloVe** is a **count-based model, Pennington et al., 2014**

Count-based models learn vectors by doing dimensionality reduction on a co-occurrence counts matrix

- Factorize this matrix to yield a lower-dimensional matrix of words and features, where each row yields a vector representation for each word



Taken from https://rbouadjenek.github.io/papers/wordembed_v2.0.pdf

GloVe: Global Vectors for Word Representation

It is an unsupervised learning algorithm developed by Pennington et al., 2014 (Stanford) for generating word embeddings by aggregating global word-word co-occurrence matrix from a corpus.

Factorizes a word-context co-occurrence matrix.

Example:

- Words: "Ice," "Water," "Steam."
- Co-occurrence with "cold" and "hot.“

Mathematical Objective:

$$J = \sum_{i,j} f(X_{ij}) \cdot \left(w_i^T w_j + b_i + b_j - \log(X_{ij})\right)^2$$

GloVe: Example

GloVe training starts by forming a co-occurrence matrix *X* where the *ij*-th entry is the number of times words on *i*-th row and *j*-th column appeared together,

Co-occurrence matrix for a window size of 1 when corpus consists of only the following three sentences

- I like deep learning.
- I like NLP.
- I enjoy flying.

	I	like	enjoy	deep	learning	NLP	flying
I	0	2	1	0	0	0	0
like	2	0	0	1	0	1	0
enjoy	1	0	0	0	0	0	1
deep	0	1	0	0	1	0	0
learning	0	0	0	1	0	0	0
NLP	0	1	0	0	0	0	0
flying	0	0	1	0	0	0	0

GloVe: Example

Co-Occurrence Ratios

After counting the co-occurrences, GloVe argues that for any group of three corpus words, *w1, w2, w3*; one of the following three scenarios applies:

1. ***w3*** is relevant to both ***w1*** and ***w2***:

$$\frac{P(w_3|w_1)}{P(w_3|w_2)} \approx 1$$

For example, let *w1*= *"coffee"*, *w2*= *"tea"* and *w3*= *"beverages"*:

$$\boxed{P(beverages|coffee) = \frac{\#(beverages, coffee)}{\sum_{w \in corpus} \#(w, coffee)}}$$

$$\boxed{P(beverages|tea) = \frac{\#(beverages, tea)}{\sum_{w \in corpus} \#(w, tea)}}$$

$$\boxed{\frac{P(beverages|coffee)}{P(beverages|tea)} \approx 1}$$

GloVe: Example

2. ***w3*** is only relevant to one of ***w1*** or ***w2***, not both:

$$\frac{P(w_3|w_1)}{P(w_3|w_2)} \leq 1$$

$$\frac{P(w_3|w_1)}{P(w_3|w_2)} \geq 1$$

3. ***w3*** is irrelevant to both ***w1*** and ***w2***:

$$\frac{P(w_3|w_1)}{P(w_3|w_2)} \approx 1$$

GloVe

The general form of the **GloVe Cost Function**

$$F(w_i, w_j, w_k \frac{P(w_k|w_i)}{P(w_k|w_j)}$$

GloVe

Nearest words to frog

Frogs
Toad
Litorai
Leptodactylidea
Eleutherodactylus
Rana
Lizard

Litoria

leptodactylidae

Rana

eleutherodactylus

Limitations of Word2vec

- **Lack of Context Awareness**
  - Word2Vec assigns the same vector to a word regardless of its context.
    - Example: "bank" in "river bank" vs. "financial bank" has the same embedding.
- **Difficulty Handling Polysemy**
  - Struggles with words having multiple meanings.
    - Example: "bat" (animal) vs. "bat" (sports equipment).
- **No Sentence-Level Understanding**
  - Word2Vec works at the word level, ignoring sentence structure and relationships.
    - Example: "The cat chased the mouse" vs. "The mouse chased the cat" treated the same.
- **Static Word Representations**
  - Word vectors are pre-trained and remain unchanged, limiting adaptability to domain-specific contexts.
- **Inability to Capture Long-Range Dependencies**
  - Does not consider relationships between words far apart in a sentence.
    - Example: Context in "The patient was prescribed medication because their condition worsened."

The need for Context-Based Models

- To capture **contextual meanings** of words dynamically.
- To understand **sentence structure** and l**ong-term dependencies**.
- To handle **domain-specific aspects** and evolving meanings effectively.

The Solution

The limitations of Word2Vec led to the rise of advanced models like **BERT** and **GPT**, which offer **contextual embeddings** and excel at understanding language in depth.

Conclusion

Advances in Word Embeddings

- **Word2Vec:** Pioneered contextual word representation with its efficient models:
  - **CBOW:** Predicts a word from its context.
  - **Skip-gram:** Predicts context from a given word.
- **GloVe:** Enhanced embeddings by incorporating global statistical information.

Thank You