Transformer-Based Embeddings for Topic Coherence

(joint work with Jason Yang)

Shengtao (Alex) Ding Tarun Rapaka

Research Mentor: Dr. Willy Rodriguez

MIT PRIMES Research Track

SWIM Workshop
August 14, 2025

Outline

Motivation

- **Too much text to handle** — emails, news, research papers… overwhelming for humans.
- **Spam vs. real mail & translation** — everyday filtering and understanding across languages.
- **Extracting topics** — grouping related content to make large collections navigable.

¿Research question?

Can *smaller* models give topics as clear as *big* ones?

Introduction

- **Big idea:** text is messy and hard to work with directly. We will show simple tools that turn a large collection of documents into math objects.
- **Step 1 — Pre-processing:** split text into tokens (words/sub-words) and remove stop words (very common words like “the”, “and”).
- **Step 2 — Vectorize:** map words/sentences to vectors (numbers) so a computer can compare, measure similarity, and sort them.
- **Step 3 — Find themes:** once everything is numbers, we can group similar texts and summarize them as topics.
- **What we will ask later:** do we need very large models for this, or can smaller ones already give clear, useful topics?

Main Goals of NLP

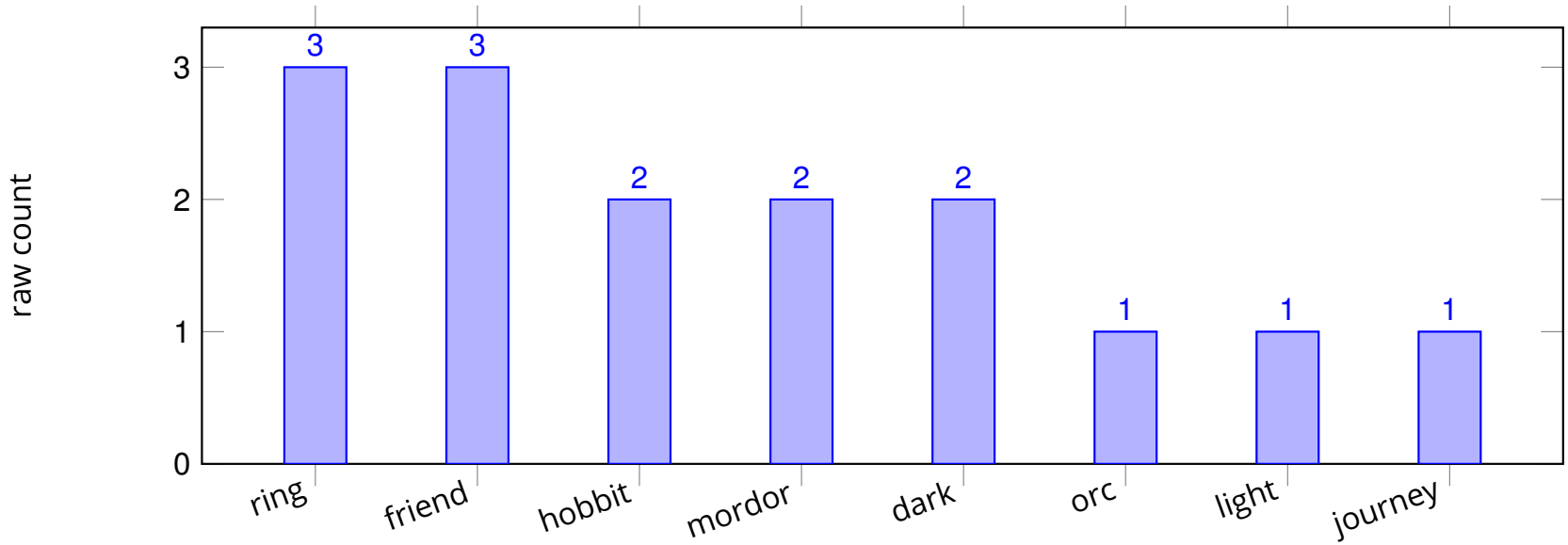
Natural Language Processing (NLP) is the field of teaching computers to work with human language—reading, understanding, and producing text.

1. **Turn text into structure** — break sentences into smaller units (tokens), turn them into vectors, and label key information.
2. **Find & organize information** — search, classify, cluster, and group text into topics.
3. **Understand & generate language** — summarize, translate, answer questions, and have conversations.
4. **Support decisions** — extract clear insights from large collections quickly and reliably.

Mini Example: Count a Few Words

Pick a tiny sample and **count** a few important words (raw frequency).

Mini-corpus (paraphrase): "The hobbit held the ring as the road bent toward the dark mountains of Mordor. His friend urged courage, for the journey ahead was long and shadowed. Orcs stirred in the valleys, and whispers of the ring carried on the wind. Yet a small light glimmered whenever hope returned to the hobbit's heart. They walked in silence, friend beside friend, counting each careful step. Beyond the pass, Mordor loomed, but the ring felt heavier and the night grew dark."



Word Cloud

Bigger words appear more often in the text corpus.
Here: *Lord of the Rings* movie dialogues (Kaggle dataset).

Counts are computed after basic cleaning (lowercasing, removing stop words).

Preprocessing: Tokenization & Normalization

- **Tokenize**: Split raw text into units (tokens) such as words or sub-words.
- **Normalize**: Lowercase, strip punctuation/diacritics, unify Unicode forms—removes superficial variation.

  ¿Why important? Reduces noise, unifies similar forms, and makes models more robust.

  Pipeline Overview: `tokenize` → `normalize` → `remove stop-words` → `lemmatize`

Example of the Pipeline

Raw sentence:

`"The quick brown fox jumps over the dog"`

1. **Tokenize** — split into words
   `[The, quick, brown, fox, jumps, over, the, dog]`
2. **Normalize** — lowercase, remove punctuation
   `[the, quick, brown, fox, jumps, over, the, dog]`
3. **Remove stop-words** — common words (e.g., “the”, “over”) that add little meaning `[quick, brown, fox, jumps, dog]`
4. **Lemmatize** — reduce words to base form
   `[quick, brown, fox, jump, dog]`

From text to mathematical objects

How to represent a text?

Bag-of-Words (BoW)?

What is Bag-of-Words (BoW)?

- Bag-of-Words is a simple and widely used method to represent text data as vectors.
- It **ignores grammar and word order**.
- It only considers the **presence or frequency of words**.
- Used to prepare text data for machine learning algorithms.

How Does It Work?

- Build a **vocabulary** of all unique words in your dataset.
- For each document, count how many times each word in the vocabulary appears.
- Represent the document as a **vector of word counts**.

Example: Two Sentences

Sentence 1: ``I love NLP"
Sentence 2: ``I love AI and I also love NLP''

Vocabulary:

$$\{\text{I}, \text{love}, \text{NLP}, \text{AI}, \text{and}, \text{also}\}$$

Each sentence is represented as a vector of counts:

	I	love	NLP	AI	and	also
Sentence 1	1	1	1	0	0	0
Sentence 2	2	2	1	1	1	1

Key Points

- **Pros:**
  - Simple to understand and implement.
  - Fast computation.
- **Cons:**
  - Ignores word order and context.
  - Example:
    - *Sentence 1:* "I like NLP but I don't like AI"
    - *Sentence 2:* "I like AI but I don't like NLP"

	I	like	NLP	AI	don't
S1	2	2	1	1	1
S2	2	2	1	1	1

  Same vector representation $\Rightarrow$ same BoW, but meaning is different.
  - Vectors can be large and sparse with big vocabularies.

TF–IDF (Simple Idea)

What it does: weights words by importance in a document and rarity in the corpus.

Use: highlights distinctive terms; down-weights very common words.

$$\boxed{\mathrm{tfidf}(w, d) = \mathrm{tf}(w, d) \cdot \log\left(\frac{N}{\mathrm{df}(w)}\right)}$$

Variable glossary:

w	word/term
d	a single document
N	total number of documents in corpus
$\mathrm{tf}(w, d)$	frequency (raw or normalized) of w in d
$\mathrm{df}(w)$	number of documents containing w

TF–IDF: Worked Example

Mini-corpus ($N = 3$):

1. d_1: `large language models`
2. d_2: `language models are powerful`
3. d_3: `models revolutionize nlp`

For the term "language" in d_2:

$$\text{tf}(\text{language}, d_2) = \frac{1}{4} = 0.25, \text{df}(\text{language}) = 2, \text{idf} = \log\left(\frac{3}{2}\right) \approx 0.405$$

$$\boxed{\text{tfidf} = 0.25 \times 0.405 \approx 0.101}$$

Partial TF–IDF matrix:

	language	models	revolutionize
d_1	0.405	0.270	0
d_2	0.101	0.067	0
d_3	0	0.067	0.405

Values rounded; TF normalized by document length.

Distances & Similarity: Formulas

Euclidean distance:

$$d_{\text{euclid}}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

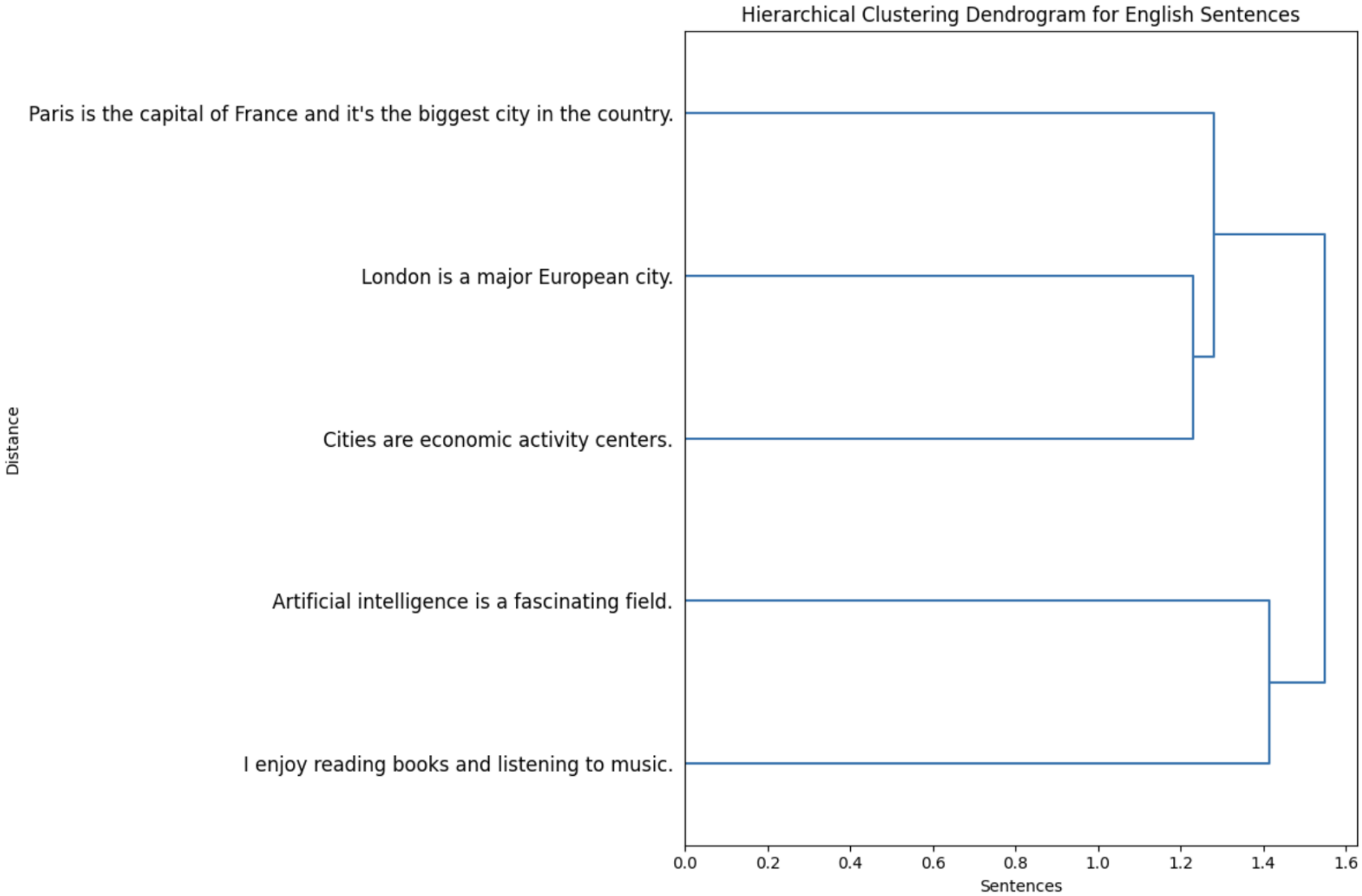
Cosine similarity:

$$\cos(\theta) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \, \|\mathbf{y}\|}$$

$$\text{Cosine distance} = 1 - \cos(\theta)$$

Euclidean: straight-line distance in vector space.
Cosine: angle between vectors, good for TF–IDF because it ignores length.

Distances & Similarity: Example Dendrogram

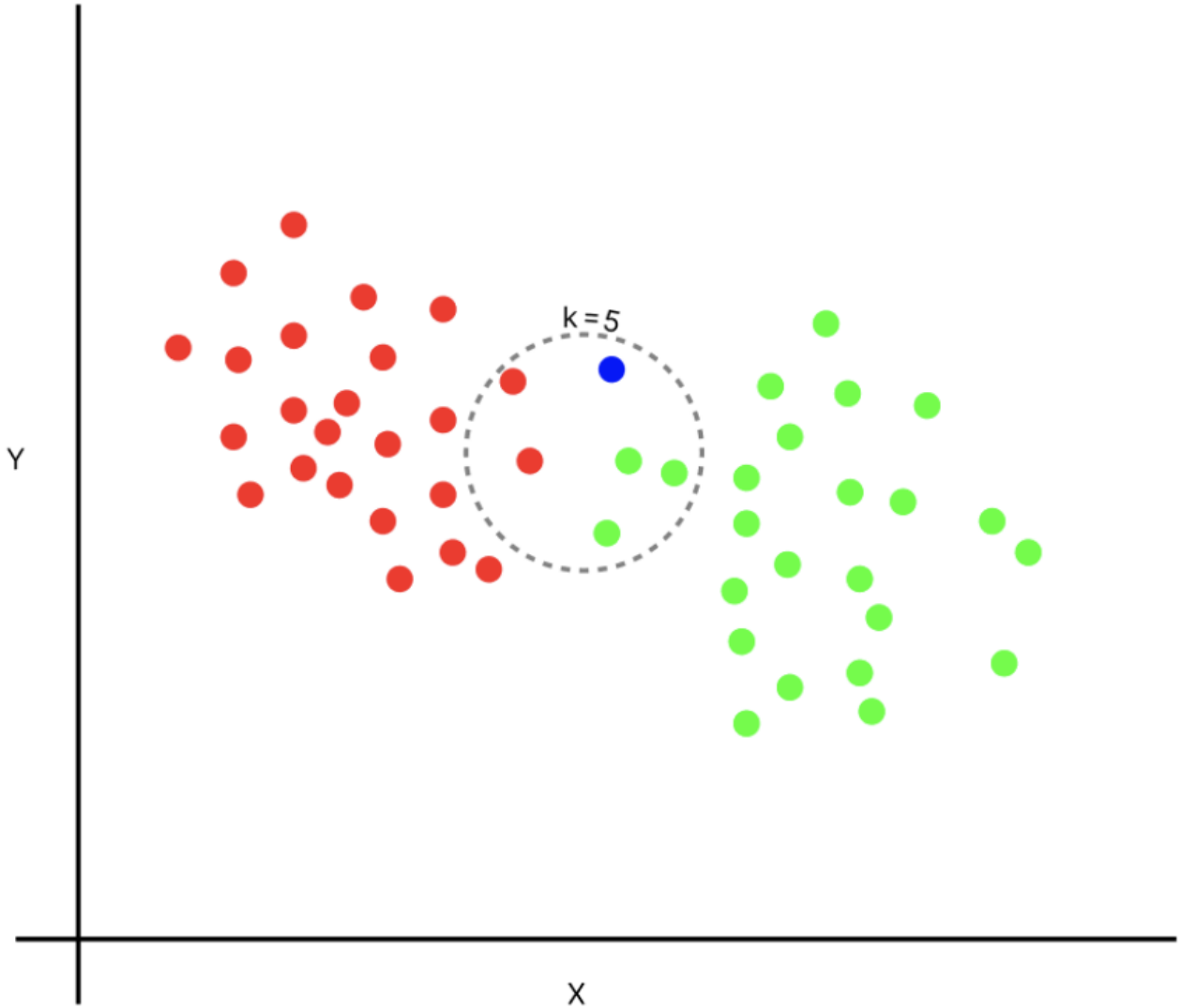


Tree shows cosine distances ($1 - \cos$) between **sentences** from a small made-up dataset. Lower height = more similar; higher height = less similar.

k-Nearest Neighbors (k-NN): Overview

- **Supervised** learning — needs human-labeled training data.
- Predicts new items by looking at the k **closest** labeled points.
- Works for classification (majority vote) and regression (average/weighted).
- Requires choosing k and a **distance metric** (cosine, Euclidean, etc.).

k-NN: Visual Example



Blue = query point; red/green = labeled training points. The dashed circle shows the $k = 5$ nearest neighbors used for prediction.

From TF–IDF to Topics: NMF & LSA

Example Corpus:

- d_1: `machine learning improves models`
- d_2: `deep learning models require data`
- d_3: `biology data from experiments`

First step: build the TF–IDF matrix (non-negative).

NMF — Non-negative Matrix Factorization:

$$\boxed{\mathbf{V} \approx \mathbf{W}\mathbf{H}} \tag{1}$$

- **V**: TF–IDF matrix
- **W**: term $\times$ topic matrix (topic word lists)
- **H**: topic $\times$ document matrix (topic weights)

LSA — Latent Semantic Analysis:

$$\mathbf{V} \approx \mathbf{U}_r \mathbf{\Sigma}_r \mathbf{V}_r^\top \tag{2}$$

Uses SVD to extract "concepts" from co-occurrence patterns.

NMF Example: From Matrix to Topics

TF–IDF Matrix (simplified):

$$\mathbf{V} =$$

	d_1	d_2	d_3
machine	0.5	0.0	0.0
learning	0.5	0.4	0.0
models	0.4	0.4	0.0
data	0.0	0.5	0.4
biology	0.0	0.0	0.5

Factorization:

$$\mathbf{V} \approx \underbrace{\begin{array}{c|cc} & T_1 & T_2 \\ \hline \textit{machine} & 0.6 & 0.0 \\ \textit{learning} & 0.5 & 0.0 \\ \textit{models} & 0.5 & 0.1 \\ \textit{data} & 0.1 & 0.6 \\ \textit{biology} & 0.0 & 0.7 \end{array}}_{\mathbf{W}} \underbrace{\begin{array}{c|ccc} & d_1 & d_2 & d_3 \\ \hline T_1 & 0.8 & 0.7 & 0.1 \\ T_2 & 0.0 & 0.4 & 0.9 \end{array}}_{\mathbf{H}}$$

Latent Dirichlet Allocation (LDA)

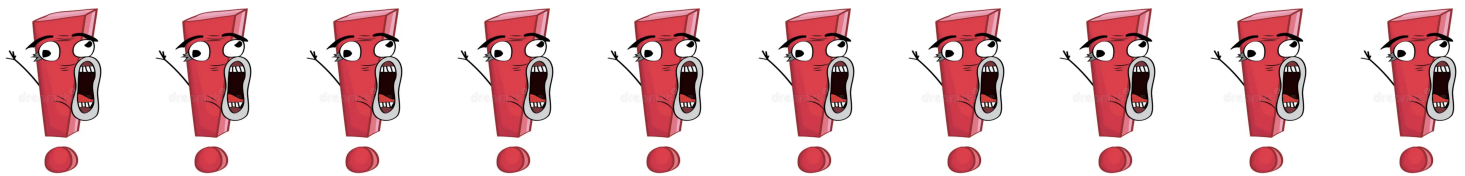
- **Purpose**: classic probabilistic topic model.
- **Input** (BoW / TF–IDF) $\rightarrow$ **Output**: topic mixture per doc + word mixture per topic.
- **View**: docs are blends of topics; topics are related-word groups.
- **Why here**: gives a clear “topic” definition for coherence & divergence.

Topic Coherence

- **IMPORTANT:** What it checks — do a topic's top words appear together in documents?
- Before numbers: UMass coherence

$$C_{\text{UMass}}(T) = \frac{2}{M(M-1)} \sum_{j=2}^{M} \sum_{i=1}^{j-1} \log\left(\frac{D(w_j, w_i) + 1}{D(w_i)}\right) \tag{3}$$

- $T = \{w_1, \ldots, w_M\}$: top-M words of the topic
- $D(w)$: # documents containing w
- $D(w_j, w_i)$: # documents containing both w_j and w_i
- $+1$: smoothing to avoid $\log 0$
- Higher $C_{\text{UMass}} \Rightarrow$ more coherent topic



Topic Divergence

- **Purpose**: measure how different two topic-word distributions P and Q are.
- **Reading**: higher = more distinct; very low = consider merge.

$$\mathrm{KL}(P\|Q) = \sum_i P(i)\,\log\frac{P(i)}{Q(i)}$$

$$\mathrm{JS}(P,Q) = \frac{1}{2}\,\mathrm{KL}(P\|M) + \frac{1}{2}\,\mathrm{KL}(Q\|M)\,, \quad M = \frac{P+Q}{2}$$

$$H(P,Q) = \frac{1}{\sqrt{2}}\,\left\|\sqrt{P} - \sqrt{Q}\right\|_2$$

Pair	JS	Hell.	Top words (each)
code | food	**0.82**	0.74	python, compile | recipe, banana
tech | startup	0.27	0.23	ai, gpu | funding, founder
tech | code	0.09	0.11	ai, gpu | python, algorithm

Advanced Tools (Neural Network Based)
(Neural Network with weight)

One-Hot Encoding

- Simplest bag-of-words representation: every token is mapped to a binary vector of length $|V|$.
- Exactly one entry is 1 (the token's index); all others are 0.

Example vocabulary:
$V = \{\texttt{large}, \texttt{language}, \texttt{model}, \texttt{revolutionize}\}$

$$\text{one\_hot}(V) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Neural Networks

- A **neural network** is a structure which takes a vector **x** as an input and passes it through several layers, producing an output **y** at the end.
- In each layer, we multiply the result from the previous layer by a weight matrix $\mathbf{W}_i$. Then we add a bias vector $\mathbf{b}_i$ and apply an **activation function**. Such functions include the **softmax function**, which sends $(x_1, \ldots, x_n)$ to $\left(\frac{e^{x_1}}{e^{x_1}+\cdots+e^{x_n}}, \ldots, \frac{e^{x_n}}{e^{x_1}+\cdots+e^{x_n}}\right)$.
- To train neural networks, the output **y** is compared with the truth by some norm function (such as the Euclidean norm), and the weight matrices/bias vectors are optimized via backpropagation/gradient descent.

Neural Networks

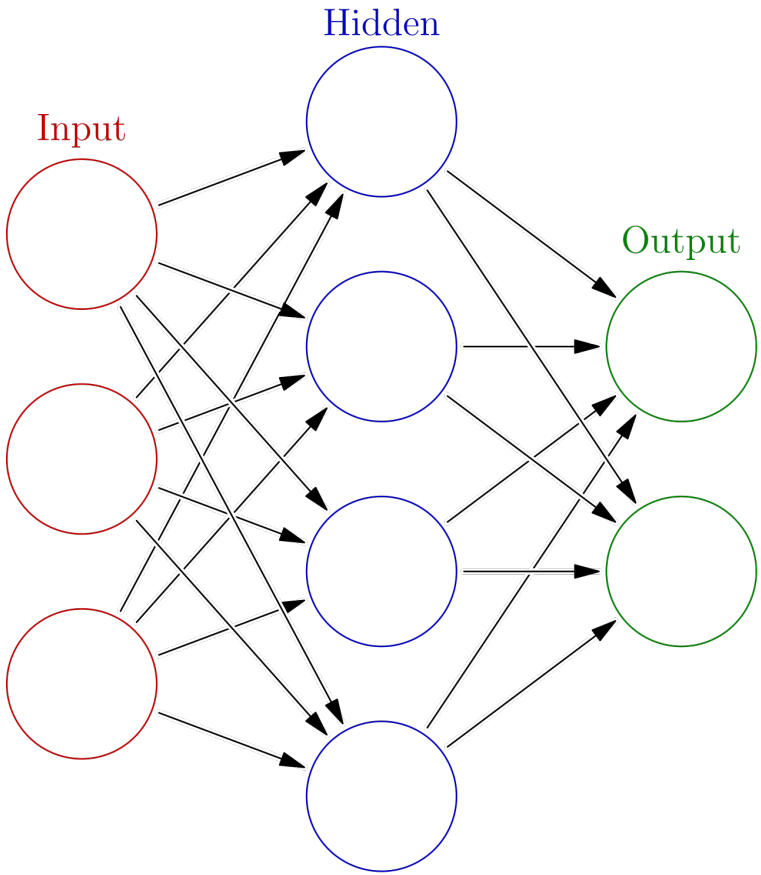


Figure: A pictorial representation of a neural network. (Source: Wikipedia)

Dense Layer Operations Illustrated in the Graph

Architecture: $\mathbb{R}^3 \xrightarrow{W_1, b_1,\ \mathrm{ReLU}} \mathbb{R}^4 \xrightarrow{W_2, b_2} \mathbb{R}^2$

Matrix forms

$$\underbrace{\mathbf{x}}_{\in \mathbb{R}^3} \xrightarrow{\mathbf{W}_1 \in \mathbb{R}^{4\times 3},\ \mathbf{b}_1 \in \mathbb{R}^4} \mathbf{a} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \quad , \quad \underbrace{\mathbf{h}}_{\in \mathbb{R}^4} = \mathrm{ReLU}(\mathbf{a})$$

$$\underbrace{\mathbf{y}}_{\in \mathbb{R}^2} = \underbrace{\mathbf{W}_2}_{\in \mathbb{R}^{2\times 4}}\ \mathbf{h} + \underbrace{\mathbf{b}_2}_{\in \mathbb{R}^2}$$

Component-wise (with ReLU on the hidden layer)

$$h_j = \max\Big(0,\ \sum_{i=1}^{3} (W_1)_{ji}\, x_i + (b_1)_j\Big) \quad \text{for } j = 1, \ldots, 4$$

$$y_k = \sum_{j=1}^{4} (W_2)_{kj}\, h_j + (b_2)_k \quad \text{for } k = 1, 2$$

Notes.

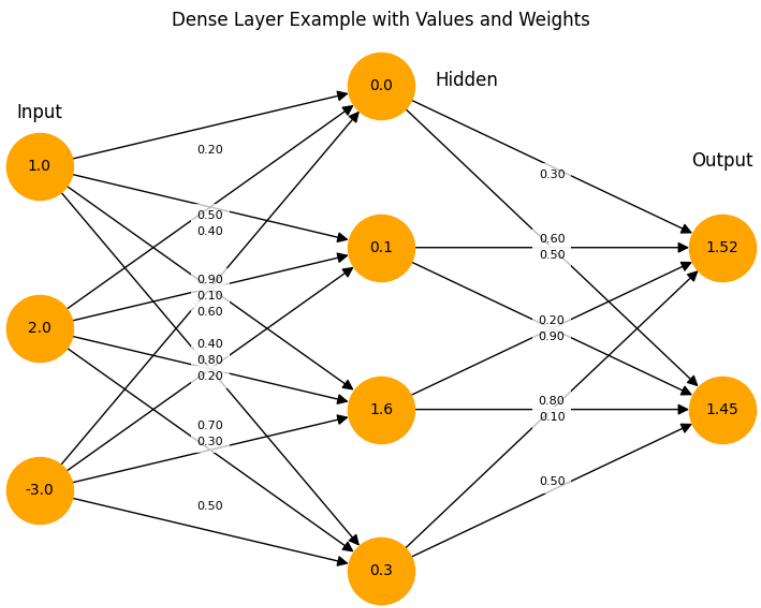
- $\mathrm{ReLU}(t) = \max(0, t)$ applied elementwise.
- In the provided script, the hidden layer uses ReLU and the output layer is linear (no activation).
- Setting $\mathbf{b}_1 = \mathbf{0}$ and $\mathbf{b}_2 = \mathbf{0}$ matches the code if biases are omitted.

Dense Layer Example with Numerical Values

Given:

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ -3 \end{bmatrix}, \quad \mathbf{W}_1 = \begin{bmatrix} 0.2 & 0.4 & 0.6 \\ 0.5 & 0.1 & 0.2 \\ 0.9 & 0.8 & 0.3 \\ 0.4 & 0.7 & 0.5 \end{bmatrix}$$

$$\mathbf{W}_2 = \begin{bmatrix} 0.3 & 0.5 & 0.9 & 0.1 \\ 0.6 & 0.2 & 0.8 & 0.5 \end{bmatrix}$$



Hidden layer:

$$\mathbf{h} = \text{ReLU}(\mathbf{W}_1\mathbf{x}) = \begin{bmatrix} 0 \\ 0.1 \\ 1.6 \\ 0.3 \end{bmatrix}$$

Output layer:

$$\mathbf{y} = \mathbf{W}_2\mathbf{h} = \begin{bmatrix} 1.52 \\ 1.45 \end{bmatrix}$$

Word Embeddings

- A **word embedding** is a function $\mathcal{C}$ from the vocabulary V to $\mathbb{R}^d$ for some d.
- Think of it as a way to assign each word in our vocabulary a real-valued vector which accurately captures the meaning of the word.
- We would expect, for instance, that

$$\mathcal{C}(\text{"king"}) - \mathcal{C}(\text{"man"}) \approx \mathcal{C}(\text{"queen"}) - \mathcal{C}(\text{"woman"}).$$

In other words, the difference between two related vectors should portray the difference between their two related words.

word2vec

- **word2vec** is one of the most popular embedding algorithms.
- It is commonly implemented with **Skip-gram**, which works by using a word to predict the context around it.
  - It is also implemented with **Continuous Bag Of Words (CBOW)**, which works the other way around (using the context around a word to predict the word).

Skip-gram model

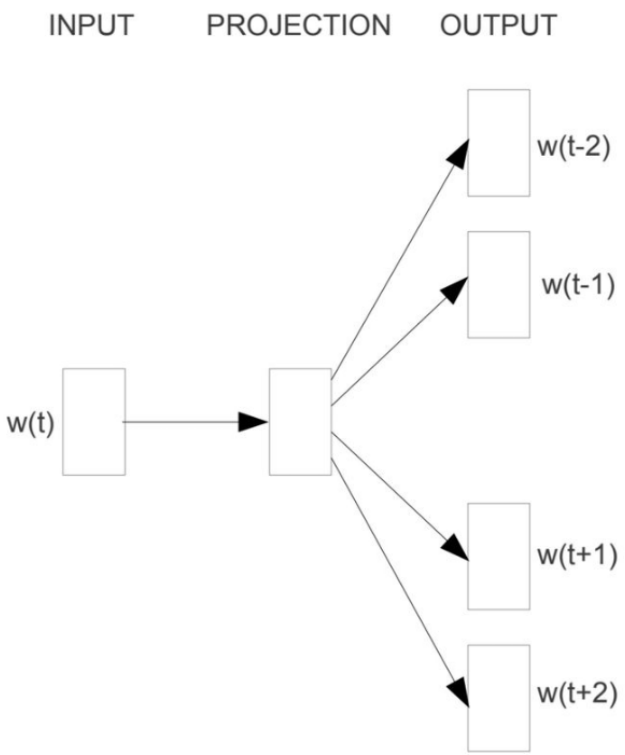


Figure: The architecture of the skip-gram mode. Source: Elias Halych

The problem with word2vec

- Consider the sentence “The bank vault lies on the bank of the river”.
- With word2vec, both “bank” words get the same embedding, even though they have different definitions.
- We need a way to determine the context for each occurrence of ‘bank” (such as ‘vault” or “river”).
- This is where **transformers** and **self-attention** come in.

Transformer Structure - Encoding

- Each encoder has an $n \times d$ input matrix X. It also has n heads, each with $d \times d_K$ weight matrices W_i^Q, W_i^K, W_i^V. The encoder has another $nd_K \times d_O$ weight matrix W_O.
- **Self attention**
  - For each i, we multiply X by each of W_i^Q, W_i^K, W_i^V to obtain Q_i, K_i, V_i, respectively.
  - The attention score for that head is given by

$$\text{Attention}_i(W_i^Q, W_i^K, W_i^V) = \text{softmax}\left(\frac{Q_i K_t^T}{\sqrt{d_k}}\right) \cdot V_i.$$

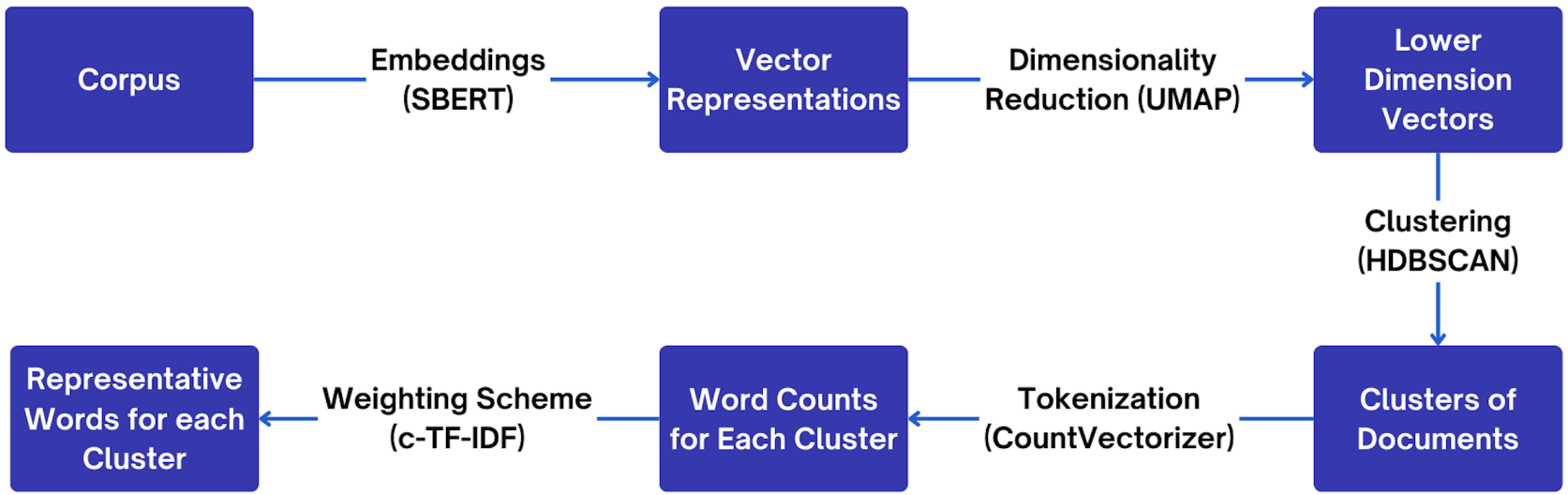
  - To calculate the attention score for the entire encoder, the attention scores for the head are concatenated and then multiplied by W_O. The result, Y, is fed into the next stage.
- **Feed-forward Neural Network** This neural network takes each row of Y as an input, and concatenates all the outputs to produce the final result Z.

Transformer structure - Decoding

- **Decoders**
  - Have a near-identical structure to encoders.
  - However, they have one more attention block, called the encoder-decoder attention.
- **Final Softmax Application**
  - After passing through the decoder stack, we get a final vector v. This is processed through one last linear transformation, we take the softmax of the result. This gives a probability distribution over the vocabulary.
  - During training, our result $\hat{v}$ can be compared to the truth using measures such as KL divergence, and such measures allow us to backpropagate and optimize the weights.

BERTopic Pipeline

- **BERTopic** is a topic modeling technique that finds topics or themes across documents in a corpus
- The pipeline follows these steps where each step is independent from the others, meaning any building block could be replaced:



Pipeline Descriptions

- **Embeddings**: Each document is converted into a vector where similar documents are closer in vector space
- **Dimensionality Reduction**: Vectors with large dimensions are reduced while preserving vector space relations to facilitate computation
- **Clustering**: Similar documents are clustered which are determined using the vectors
- **Tokenization**: A bag-of-words representation is created for each cluster by compiling every document in the cluster
- **Weighting Scheme**: Representative words for each cluster are chosen from the bag-of-words representations

Results & Discussion

- **Bigger $\neq$ always better**: small/medium models gave similar topic coherence/divergence.
- **Cross-domain consistency**: patterns hold on *20NG* and *PubMed*.
- **Reproducible pipeline**: models on Hugging Face · code/data on GitHub.

Encoder	Size	20Newsgroups		PubMed	
		Coherence	Diversity	Coherence	Diversity
all-MiniLM-L6-v2	22M	0.7422	0.9947	0.7004	0.9954
microsoft/MiniLM-L12-H384-uncased	33M	0.7374	0.9947	0.7069	0.9951
distilbert-base-uncased	66M	0.7450	0.9955	0.7137	0.9954
bert-base-uncased	110M	0.7253	0.9946	0.7012	0.9955
roberta-base	125M	0.7420	0.9950	0.7121	0.9949
meta-llama/Llama-2-7b-hf	7B	0.7310	0.9946	0.7047	0.9952
meta-llama/Llama-2-13b-hf	13B	0.7447	0.9948	0.6977	0.9954

Table: Coherence and diversity on 20 Newsgroups and PubMed dataset

Takeaways

- **Vectorization** is the first step toward machine-readable text
- **LDA vs. BERTopic**: LDA excels in interpretability; BERTopic leverages contextual embeddings and scalability
- **Objective metrics** (coherence & divergence) guide model and parameter selection

Acknowledgements

- **Research mentor:** Sincere thanks to Dr. Willy Rodriguez for continuous guidance and support.
- **Scientific contributors:** We thank Dr. Felix Gotti and Dr. Tanya Khovanova for helpful discussions and feedback related to this project.
- **Programs & organizers:** Grateful to the SWIM Workshop, and to the MIT PRIMES programs for providing this research opportunity and community.

References

D. D. Lee and H. S. Seung.
Learning the parts of objects by non-negative matrix factorization.
Nature, 401(6755):788–791, 1999.

K. Stevens, P. Kegelmeyer, D. Andrzejewski, and D. Buttler.
Exploring topic coherence over many models and many topics.
In *Proceedings of EMNLP-CoNLL*, pages 952–961, 2012.

R. Deveaud, E. SanJuan, and P. Bellot.
Accurate and effective latent concept modeling for ad hoc information retrieval.
Document Numérique, 17(1):61–84, 2014.

P. J. O'Hara and M. W. Davies.
Two-stage topic modelling of scientific publications: A case study.
PLOS ONE, 2021.

L. van der Maaten and G. Hinton.
Visualizing data using t-SNE.
Journal of Machine Learning Research, 9:2579–2605, 2008.

L. McInnes, J. Healy, and J. Melville.
UMAP: Uniform manifold approximation and projection for dimension reduction.
arXiv preprint arXiv:1802.03426, 2018.

M. Grootendorst.
BERTopic: Neural topic modeling with a class-based TF-IDF procedure.
arXiv preprint arXiv:2203.05794, 2022.

D. M. Blei, A. Y. Ng, and M. I. Jordan.
Latent Dirichlet Allocation.
Journal of Machine Learning Research, 3:993–1022, 2003.

Q&A!