IMPERIAL

Word Representation

01/12/2025

Shamsuddeen Muhammad
Google DeepMind Academic Fellow,
Imperial College London
https://shmuhammadd.github.io/

Idris Abdulmumin
Postdoctoral Research Fellow,
DSFSI, University of Pretoria
https://abumafrim.github.io/

Reference

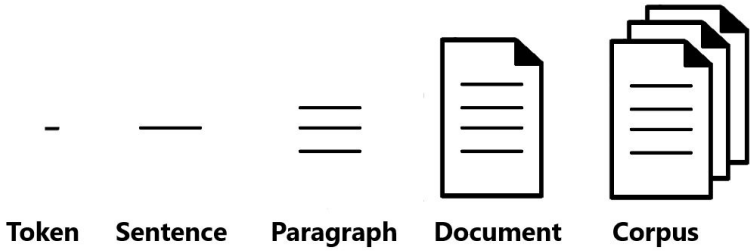
Chapter 6 : Speech and Language Processing (3rd ed. draft)

https://web.stanford.edu/~jurafsky/slp3/

What is a Document?

In **NLP** a document is a fundamental unit of text used for analysis, modeling, or processing. Examples: sentence, paragraph, article, or entire file.

- **Corpus**: A collection of documents
- **Document**: A unit of analysis (e.g., article)
- **Sentence**: A grammatically complete statement
- **Token**: A word or punctuation mark



Corpus vs Dataset?

A **corpus** is a large collection of natural language text, typically gathered for linguistic analysis rather than for training machine learning models. Content may b raw text (no labels),

A **dataset** consists of structured text paired with task-specific labels designed for machine learning, such as sentiment labels, entity tags, or question-answer pairs.

Corpus is raw/general text , while dataset is task-specific with annotation

Word Representation

Effective language modeling requires a meaningful way to represent words.

- Localist Representation
- Using Existing Thesauri or Ontologies
- Distributional Semantics

Localist Representation

In traditional (pre-neural) NLP, each word is treated as a **discrete, atomic symbol** with no built-in notion of similarity.

Words such as *hotel*, *conference*, and *motel* are encoded as **independent identifiers**, even though some are semantically related.

Each word occupies a **unique and isolated slot** in the vocabulary (e.g., one-hot vector). No two words share features.

Localist Representation (One-Hot Encoding)

Given a vocabulary:

$$V = \{\text{hotel}, \text{motel}, \text{conference}, \text{meeting}, \text{airport}\}$$

We can assign a unique one-hot vector representation to each word:

Word	One-Hot Vector
hotel	[1, 0, 0, 0, 0]
motel	[0, 1, 0, 0, 0]
conference	[0, 0, 1, 0, 0]
meeting	[0, 0, 0, 1, 0]
airport	[0, 0, 0, 0, 1]

Localist Representation: No Semantic Similarity

- Orthogonal one-hot vectors encode **zero similarity**, even for closely related words. Example:

$$\text{cosine\_similarity}(\text{“hotel”}, \text{“motel”}) = 0$$

- The model cannot infer semantic relatedness from the representation.

Localist Representation: Poor Generalization

- Knowledge learned about one word **cannot transfer** to similar words.
- If a model learns from *“I stayed at a hotel”*, it cannot generalize to *“I stayed at a motel”* because the vectors share no features.
- This prevents models from capturing broader linguistic patterns

Localist Representation: Sparse and High-Dimensional

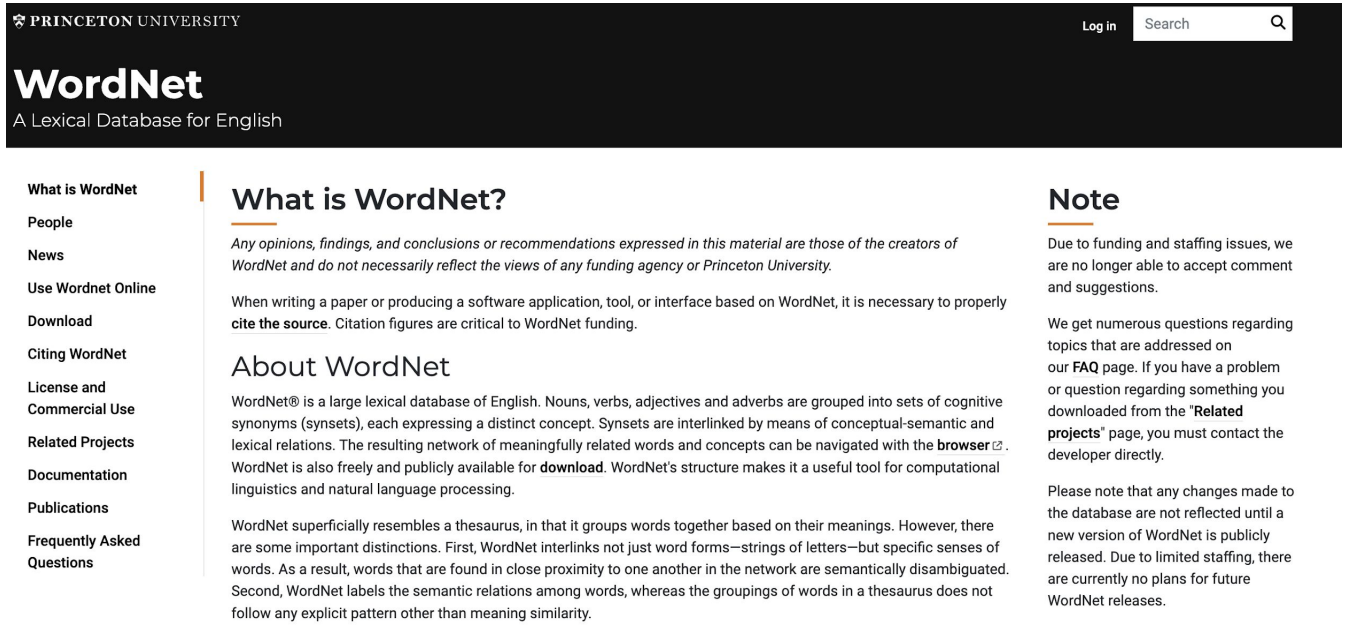
- A one-hot vector of length |V| contains a single 1 and the rest zeros → extremely sparse.
- Larger vocabularies require **very high-dimensional vectors**, increasing storage and computation cost.
- Inefficient for modern NLP tasks.

Solutions?

Using Existing Thesauri or Ontologies (e.g., WordNet)

WordNet

- *A lexical database of English organized by meaning.*
- Groups words into synsets (sets of cognitive synonyms) linked by semantic relations.



https://wordnet.princeton.edu/

What WordNet Does

Groups words into *synsets*

Synsets = sets of words that share the same meaning (synonyms). *(e.g., {car, automobile}; {big, large})*

Example:

Senses of 'bass':

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations
Display options for sense: (gloss) "an example sentence"

Noun

- S: (n) **bass** (the lowest part of the musical range)
- S: (n) **bass**, bass part (the lowest part in polyphonic music)
- S: (n) **bass**, basso (an adult male singer with the lowest voice)
- S: (n) sea bass, **bass** (the lean flesh of a saltwater fish of the family Serranidae)
- S: (n) freshwater bass, **bass** (any of various North American freshwater fish with lean flesh (especially of the genus Micropterus))
- S: (n) **bass**, bass voice, basso (the lowest adult male singing voice)
- S: (n) **bass** (the member with the lowest range of a family of musical instruments)
- S: (n) **bass** (nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes)

Adjective

- S: (adj) **bass**, deep (having or denoting a low vocal or instrumental range) *"a deep voice"; "a bass voice is lower than a baritone voice"; "a bass clarinet"*

Organizes words in a semantic hierarchy:

wordNet encodes "is-a" relationships (hypernyms/hyponyms), so it knows:

- *dog → animal*
- *rose → flower*
- *car → vehicle*

WordNet 3.0: A Better Way to Represent Word Meanings

WordNet lets you **connect** these words by their meanings. For instance:

- *hotel* and *motel* belong to similar synsets **(synonym set)**.
- The system now understands they are both types **of lodging places**.
- This adds **semantic structure** to your model's understanding.

Property	**Localist (One-Hot)**	**WordNet**
Representation of words	Isolated, no relations	Grouped by meaning (synsets)
Semantic similarity	Not captured	Captured via lexical relations
Structure	Sparse, flat vectors	Hierarchical and structured
Generalization	Poor	Good (through shared meanings)
Vocabulary scaling	Inefficient	More compact and meaningful

Drawbacks of Thesaurus-based Approaches

A useful resource but missing nuance

- e.g., "*proficient*" is listed as a synonym for "*good*", but this is only true in specific contexts.

- WordNet may list offensive synonyms without accounting for connotation or appropriateness.

Drawbacks of Thesaurus-based Approaches

Missing new meanings of words:

- e.g., *wicked, badass, nifty, wizard, genius, ninja, bombest*
- Rapid language change makes it hard to stay up-to-date.

Requires human labor to create and adapt

- Manually curated thesauri like WordNet are expensive and slow to update.

Distributional Semantics

Representing Words by Their Context

Distributional Semantics is an approach that defines a word's meaning based on the contexts in which the word appears.

The concept is from a classic linguistic idea:

> "You shall know a word by the company it keeps."
>
> — J. R. Firth (1957)

Distributional Semantics: How It Works

Distributional semantics builds word meaning by analyzing:

- the surrounding words,
- co-occurrence statistics,
- window-based context,
- frequency patterns in a corpus.

These patterns are turned into **vector representations** of words (embeddings), where:

- Similar words → vectors close together
- Different words → vectors far apart

Distributional semantics forms the basis for: word2vec, GloVe, fastText and modern embedding-based NLP models

How do we represent words by their context?

1. **Count-based Methods**
   - Build a word-context co-occurrence matrix from a corpus.
   - Apply dimensionality reduction (e.g., SVD, PCA) to get dense vectors.
   - Examples:
     - **Term Frequency (TF)**: Number of times a word appears in a document.
     - **Document Frequency (DF)**: Number of documents in which a word appears.
     - **TF-IDF**: Combines TF and DF to downweight common terms and highlight informative words.
     - **LSA (Latent Semantic Analysis)**: Uses SVD on TF-IDF matrix.

2. **Predictive Methods**
   - Use a neural model to predict a word given its context (or vice versa).
   - Train embeddings as part of this predictive task.
   - Examples: Word2Vec (CBOW, Skip-gram), GloVe, FastText

Count-based method: Co-occurrences for Word Similarity

1. Take a large corpus.

2. Choose a vocabulary of words (e.g., the 10k most frequent words).

3. For each pair *(target word, context word)*, count how often they appear together within some context window (e.g., ±5 words).

	aardvark	computer	data	pinch	result	sugar
apricot	0	0	0	1	0	1
pineapple	0	0	0	1	0	1
digital	0	2	1	0	1	0
information	0	1	6	0	4	0

Count Vectors in a Co-occurrence Matrix

- Each **row** represents a **target word**.
- Each **column** represents a **context word**.
- Each cell shows how often the target appears near the context word in the corpus.

Example: Count vector for "digital"

[aardvark: 0, computer: 2, data: 1, pinch: 0, result: 1, sugar: 0]

Interpretation:

- "digital" appears 2 times near "computer", 1 time near "data" and "result".
- Words like "apricot" and "pineapple" have similar context vectors: [0, 0, 0, 1, 0, 1], indicating semantic similarity.

	aardvark	computer	data	pinch	result	sugar
apricot	0	0	0	1	0	1
pineapple	0	0	0	1	0	1
digital	0	2	1	0	1	0
information	0	1	6	0	4	0

Co-occurrences for Word Similarity

The Term-Context matrix (or, word-word matrix)

- Two **words** are similar in meaning if their context vectors are similar

	aardvark	computer	data	pinch	result	sugar
apricot	0	0	0	1	0	1
pineapple	0	0	0	1	0	1
digital	0	2	1	0	1	0
information	0	1	6	0	4	0

Should We Use Raw Counts?

Raw word frequency is not always a good indicator of word importance.

It's often **skewed** by common words (e.g., "the", "of").

We'd prefer a measure that tells us how **informative** a word is in context.

For the term–document matrix:

- We typically use **tf–idf** instead of raw term counts.

Motivation for Better Approaches

High-dimensional vectors: Size of vocabulary V can be very large, leading to high memory usage.

Sparsity: Most entries in the term-context matrix are zero, wasting space and computation.

Stopwords dominate: Common function words (e.g., "the", "is", "of") appear frequently but carry little meaning.

No semantic similarity: "hotel" and "motel" get orthogonal vectors despite being semantically similar.

Lack of generalization: The model doesn't learn similarity beyond observed co-occurrences.

Term Frequency (TF) and Document Frequency

Term Frequency (TF): Measures how often a word appears in a specific document.

Formula: $tf_{t,d} = \text{count}(t, d)$

Document Frequency (DF): Measures how many documents contain the word at least once.

Formula: $df_t = \text{number of documents containing } t$

These metrics are useful for understanding word importance in different documents and are foundational for computing tf-idf scores.

Term Frequency (tf)

Term Frequency (TF) measures how often a word **t** appears in a document **d**.

It reflects the importance of a word in that specific document.

Raw TF:

$$tf_{t,d} = \text{count}(t, d)$$

However, some words (e.g., "the", "is") appear very frequently and dominate counts.

To handle this, we use a **log-scaled version**:

$$tf_{t,d} = \log_{10}(\text{count}(t, d) + 1)$$

Why add 1? To avoid log of zero when the word does not appear in the document.

Example: Term Frequency

Consider the document:

"The cat sat on the mat. The mat was soft."

Raw counts:

- the: 3
- cat: 1
- mat: 2

Log-scaled TF:

- $\text{tf}(\text{the}) = \log_{10}(3+1) = \log_{10}(4) \approx 0.60$
- $\text{tf}(\text{cat}) = \log_{10}(1+1) = \log_{10}(2) \approx 0.30$
- $\text{tf}(\text{mat}) = \log_{10}(2+1) = \log_{10}(3) \approx 0.48$

TF Table: How many times a word appears in each document.
Example Sentence Collection:

- Doc1: Romeo loves Juliet.
- Doc2: Juliet loves Paris.
- Doc3: Romeo loves books and music.

Word	TF in Doc1	TF in Doc2	TF in Doc3
Romeo	1	0	2
Juliet	1	1	0
loves	1	1	2
Paris	0	1	0
books	0	0	1
music	0	0	1

Interpretation: More frequent words have higher TF, but the log scale reduces the dominance of high-frequency words.

Why not just use word frequency?

- Common words (e.g., `the`, `is`, `and`) appear in almost every document.
- They don't help us distinguish between documents.
- We want to give more weight to **distinctive** words that help identify topics.

Document Frequency (DF)

Document Frequency (DF) counts how many different documents a term appears in.
It does not count the number of times a word appears, but *how many documents* contain the word.

$$df_t = \#\{d \in D : t \in d\}$$

DF Table: In how many documents does each word appear?

Word	Document Frequency (DF)
Romeo	2
Juliet	2
loves	3
Paris	1
books	1
music	1

Interpretation:

- "loves" appears in all documents $\Rightarrow$ DF $= 3$
- "Paris" appears in only Doc2 $\Rightarrow$ DF $= 1$

Document Frequency (df)

df_t *is* the number of documents *t* occurs in.

(note this is NOT *collection frequency*: *total count across all documents*)

Example: "*Romeo*" is very distinctive for one Shakespeare play:

	Collection Frequency	Document Frequency
Romeo	113	1
action	113	31

Inverse Document Frequency (IDF)

Definition: IDF down-weights common terms that appear in many documents.

Formula:

$$idf_t = \log_{10}\left(\frac{N}{df_t}\right)$$

where N is the total number of documents.

Intuition:

- Words like *"loves"* appear in almost every document (high DF).
- IDF reduces their weight since they are less informative.
- Words like *"music"* appear in only one document (low DF), so they are more informative.

Inverse Document Frequency (IDF)

IDF measures how unique a word is across all documents.

Intuition: Rare words are more useful for distinguishing documents.

Formula

$$idf_t = \log_{10}\left(\frac{N}{df_t}\right)$$

Where N is the total number of documents.

Word	DF	IDF
Romeo	2	$\log_{10}(3/2) = 0.176$
Juliet	2	$\log_{10}(3/2) = 0.176$
loves	3	$\log_{10}(3/3) = 0.000$
Paris	1	$\log_{10}(3/1) = 0.477$
books	1	$\log_{10}(3/1) = 0.477$
music	1	$\log_{10}(3/1) = 0.477$

TF-IDF

Definition: TF-IDF combines both TF and IDF.

Formula:

$$tfidf_{t,d} = tf_{t,d} \times idf_t$$

Purpose:

- Highlights words that are frequent in a document but rare across documents.
- Helps in selecting important and unique terms.

Good TF-IDF terms are those that are frequent in a specific document but rare overall.

Final tf-idf Weighted Value for a Word

$$w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$$

Raw counts

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

tf-idf

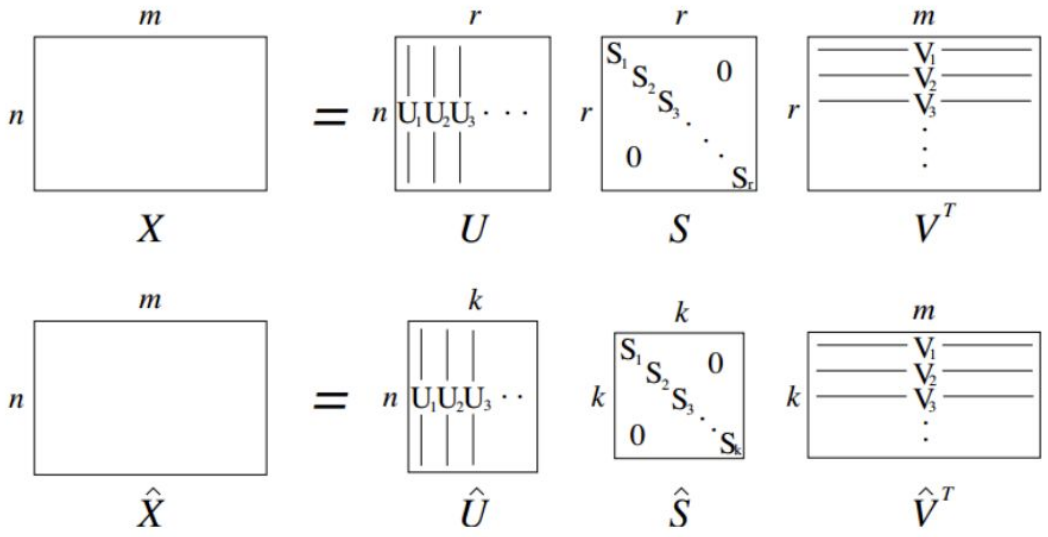
	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Drawbacks of Co-occurrence Matrix Approach

- Quadratic space needed
- Relative position and order of words not considered

Low Dimensional Vectors

- Store only “important” information in fixed, low dimensional vector.
- Singular Value Decomposition (SVD) on co-occurrence matrix
  - $\hat{X}$ is the best rank k approximation to X, in terms of least squares
  - Motel = [0.286, 0.792, -0.177, -0.107, 0.109, -0.542, 0.349, 0.271]



Drawbacks of SVD-based Approach

- Computational cost scales quadratically for n x m matrix: $O(mn^2)$ flops (when n<m)
- Hard to incorporate new words or documents
- Does not consider order of words

Prediction-based Methods

Word Embedding

Dense vector: Unlike one-hot encoding, embeddings represent words in compact, low-dimensional space.

Helps in learning fewer parameters: Smaller vector sizes mean more efficient computation and faster convergence in models.

May generalize better: Embeddings capture distributional properties that allow unseen but similar contexts to influence predictions.

Can capture synonyms better:

- *car* and *automobile* are synonyms; but in one-hot encoding, they have completely different dimensions.
- With embeddings, they can have similar vector representations.
- For example, a model with *car* as a neighbor and another with *automobile* should give similar outputs if embeddings are used.

What is Dense Vector?

A **dense vector** is a numerical representation of a linguistic unit (e.g., word, sentence, or document) where most or all components are non-zero real numbers:

$$\mathbf{v} \in \mathbb{R}^d \quad \text{with} \quad v_i \neq 0 \text{ for most } i$$

Contrast with Sparse Vectors

Property	Sparse Vector (e.g., One-Hot)	Dense Vector
Dimensionality	Very high (e.g., $\|V\|$)	Low (e.g., 300, 768)
Values	Mostly zeros, one 1	Mostly non-zero, real numbers
Captures Semantic Similarity	No	Yes
Manually or Learned	Manual	Learned from data

- **Word embeddings:** Models such as Word2Vec, GloVe, and fastText generate dense vectors for individual words.
  Example: `"king"` $\rightarrow [0.21, -0.14, 0.83, \ldots] \in \mathbb{R}^{300}$
- **Contextual embeddings:** Transformer models like BERT and RoBERTa produce dense representations for words in context.
- **Sentence/document embeddings:** Models like Sentence-BERT produce dense, contextualized embeddings for longer text units (e.g., full sentences or paragraphs). These embeddings reflect the semantics of the entire input, making them useful for tasks like semantic similarity, clustering, and information retrieval.

Why Sentence/Document Embeddings are Contextual

Sentence and document embeddings from models like BERT, RoBERTa, and Sentence-BERT are considered **contextual embeddings** because they compute representations that depend on the surrounding context.

What Makes Them Contextual?

Unlike static embeddings (e.g., Word2Vec, GloVe) that assign one fixed vector per word, contextual models generate different embeddings for the same word depending on its usage.

Examples

Word-level (BERT):

- *"He went to the bank to deposit money."*
- *"She sat by the river bank."*

The word `bank` receives different embeddings in each case because BERT uses context to disambiguate meaning.

Sentence-level (Sentence-BERT):

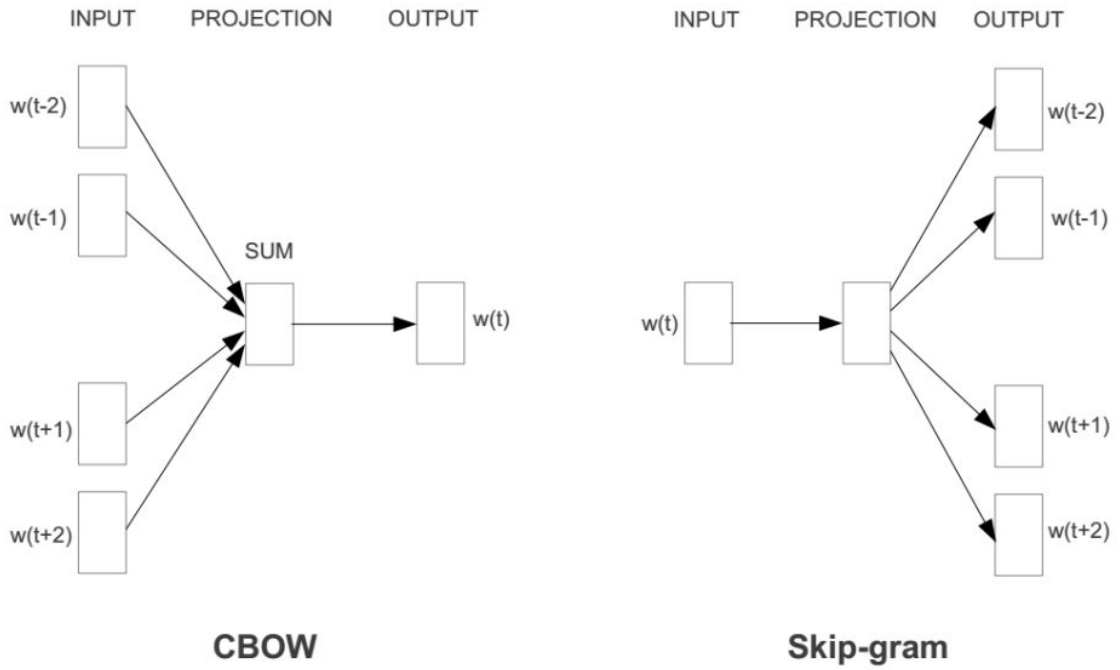
- *"A man is playing a guitar."*
- *"Someone is performing music."*

These sentences have different surface forms but similar meanings. Sentence-BERT maps them to dense vectors that are close in embedding space, reflecting their semantic similarity.

Represent The Meaning of Word: Word2vec

Two basic neural network models:

- **Continuous Bag of Word (CBOW):** use a window of word to **predict the middle word**
- **Skip-gram (SG):** use a word to **predict the surrounding words in window**



Word2Vec

Word2Vec (Mikolov et al., 2013) is a neural model that learns dense word embeddings from large-scale text corpora.

Key Shift: Instead of using count-based statistics (e.g., co-occurrence matrices), Word2Vec formulates a **prediction task**:

- CBOW: Predict a target word from its surrounding context.
- Skip-gram: Predict surrounding context words from a target word.

These tasks are trained using **self-supervision**, where the corpus provides its own training signals.

Learning Through Self-Supervision

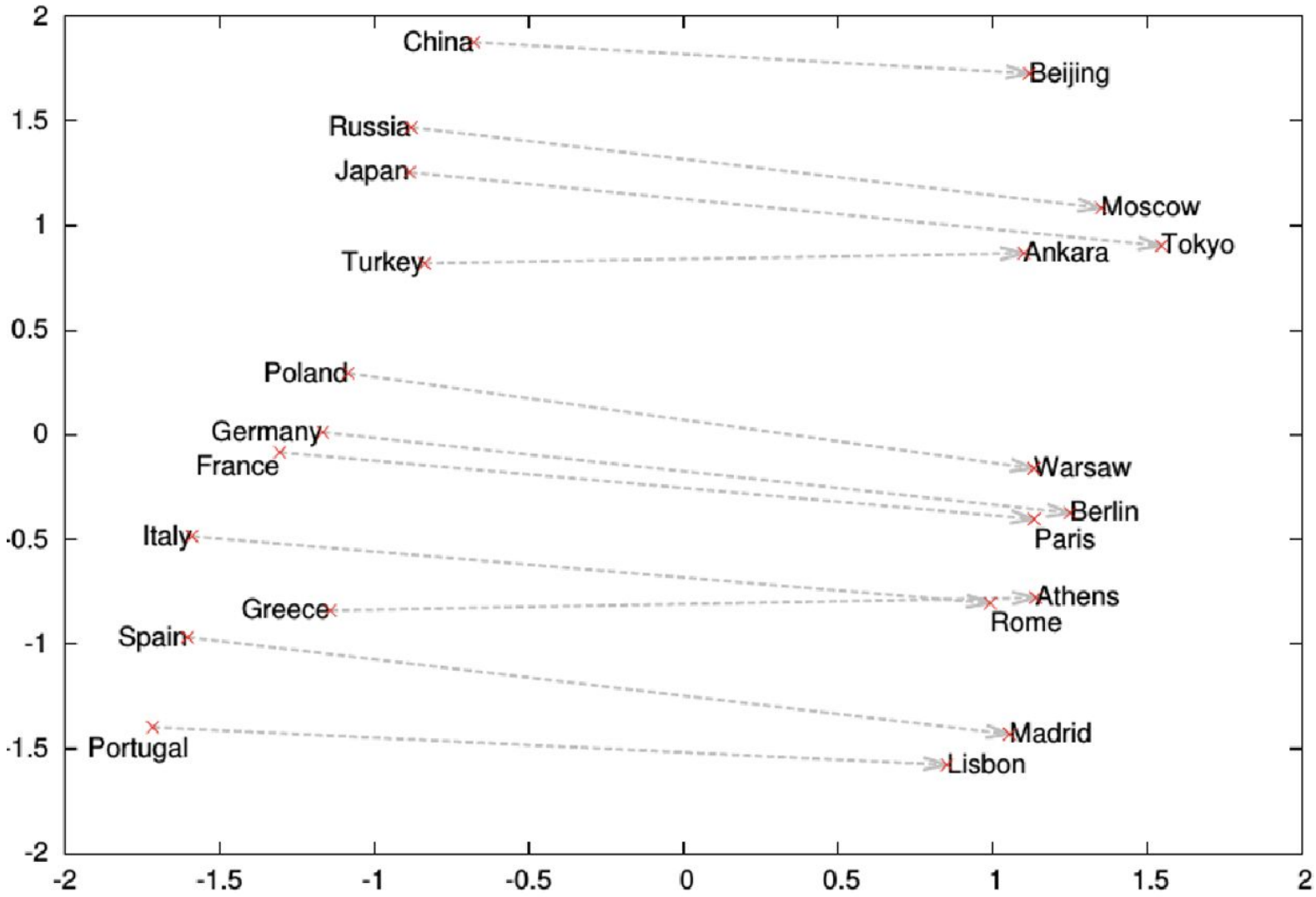
Self-supervised Learning:

- No manual labeling needed.
- The model uses naturally occurring word sequences to create training pairs.
- For example, if "apricot" appears near "fruit" in the corpus, "fruit" becomes a positive context word.

Result:

- Trained classifier weights (in the hidden layer) are used as **word embeddings**.
- Embeddings encode meaningful relations between words.

Embedding encode meaningful relations between words



CBOW: Continuous Bag of Words

Goal: Predict the target word from surrounding context words.

Example Sentence: *"The cat sat on the mat."*
Context window size = 2 $\Rightarrow$ $Context = \{The, cat, on, the\}$
Target: *sat*

Input: {The, cat, on, the}
Output: *sat*
CBOW tries to predict the middle word given its context.

Skip-gram

Goal: Predict context words given the target word.

Example Sentence: *"The cat sat on the mat."*
Target word: *sat*
Context window size = 2
$\Rightarrow$ *Predictwordsaround"sat" :* $\{cat, on, The, the\}$

Input: *sat*
Output: {cat, on, The, the}
Skip-gram does the reverse of CBOW.

Word2vec

- Instead of **counting** how often each word *w* occurs near "*apricot*"
  - Train a classifier on a binary **prediction** task:
    - Is *w* likely to show up near "*apricot*"?
- We don't actually care about this task
  - But we'll take the learned classifier weights as the word embeddings
- Big idea: **Self-supervision**
  - A word c that occurs near apricot in the corpus cats as the gold "correct answer" for supervised learning
  - No need for human labels
  - Bengio et al. (2003); Collobert et al. (2011)

CBOW vs Skip-gram

Feature	CBOW	Skip-gram
Prediction Task	Context $\rightarrow$ Target	Target $\rightarrow$ Context
Training Speed	Faster	Slower
Performance on Rare Words	Weaker	Better
Best For	Frequent words, large corpus	Rare words, small corpus

Why CBOW is Faster and Suited for Frequent Words

Why CBOW is Faster:

- Predicts a single target word from multiple context words (only one update).
- Uses averaged context embeddings: lower computation per example.
- Efficiently parallelizable and well-suited for large datasets.

Why CBOW Works Well with Large Corpora and Frequent Words:

- Frequent words appear in many diverse contexts, leading to stable embeddings.
- Averaging context is effective when training examples are abundant.
- Large corpora provide enough data for CBOW to learn robust patterns for common words.

Why CBOW is Weaker for Rare Words

Limitations with Rare Words:

- Averaging context dilutes the signal from informative rare words.
- Rare words occur less frequently as targets and thus receive fewer updates.
- CBOW doesn't directly optimize rare word vectors—relies on surrounding context.

Skip-gram Advantage:

- Directly updates the vector of the rare word (as input) during prediction.
- Each occurrence of a rare word allows multiple updates via its context.

What Embeddings Capture?

- Similar words have vectors that are close in the embedding space.
- Embeddings capture both syntactic and semantic relationships.

Syntax and Semantics in Embeddings

Sample 5D Embedding Vectors (truncated for simplicity)

Word	v_1	v_2	v_3	v_4	v_5
king	0.52	0.21	0.10	0.89	0.65
queen	0.50	0.22	0.12	0.85	0.66
man	0.48	0.25	0.09	0.81	0.63
woman	0.51	0.26	0.11	0.82	0.64
car	0.10	0.88	0.33	0.21	0.04
truck	0.11	0.85	0.35	0.22	0.06

Word embeddings capture both:

- **Semantics**: Conceptual or topical similarity (*meaning*)
- **Syntax**: Grammatical similarity (*function*, *usage*)

They do this by learning from distributional contexts across large corpora.

Examples of Semantic Similarity

- `vector("king")` $\approx$ `vector("queen")`
- `vector("Paris") - vector("France") + vector("Italy")` $\approx$ `vector("Rome")`

These embeddings reflect meaningful real-world relationships (e.g., gender, geography).

Example of Syntactic Similarity

- `vector("walking") - vector("walk")` $\approx$ `vector("swimming") - vector("swim")`
- `vector("ran") - vector("run")` $\approx$ `vector("slept") - vector("sleep")`

These analogies show how embeddings capture morphological transformations (e.g., tense, verb forms).

How embedding works?

Predict if a Candidate Word is a "Neighbor”

Word2Vec formulates word embedding learning as a prediction problem:

Given a target word t, can we predict whether a candidate word c is a valid neighbor (context word)?

This is cast as a binary classification task.

Prediction Approach

Positive Examples: Pair the target word t with true context words c (within a context window).

Negative Examples: Randomly sample other words from the vocabulary to create false (non-context) pairs.

Train a Classifier: Use logistic regression to distinguish between positive and negative pairs.

Learn Embeddings: Use the learned weights from the classifier as the word embeddings.

Why This Works?

Words that occur in similar contexts will be trained to have similar vector representations.

Negative sampling helps focus learning on distinguishing relevant neighbors.

No manual labels needed – this is a **self-supervised** learning framework.

Example: *Target Word "apple*"

Positive context words: `"fruit"`, `"red"`, `"juice"`
Negative samples: `"engine"`, `"table"`, `"sky"`

The model learns to assign high similarity to true neighbors and low similarity to random (negative) words.

Summary How to Learn Word2Vec (Skip-gram)

- Start with V random d-dimensional vectors as initial embeddings
- Train a classifier based on embedding similarity
  - Take a corpus and take pairs of words that co-occur as positive examples
  - Take pairs of words that don't co-occur as negative examples
  - Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
  - Throw away the classifier code and keep the embeddings

Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Abstract

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.

hauWE: Hausa Words Embedding for Natural Language Processing

Idris Abdulmumin
Department of Computer Science
Ahmadu Bello University, Zaria

Bashir Shehu Galadanci
Department of Software Engineering
Bayero University, Kano

***Abstract* – Words embedding (distributed word vector representations) have become an essential component of many natural language processing (NLP) tasks such as machine translation, sentiment analysis, word analogy, named entity recognition and word similarity. Despite this, the only work that provides word vectors for Hausa language is that of [1] trained using fastText, consisting of only a few words vectors. This work presents words embedding models using Word2Vec's Continuous Bag of Words (CBoW) and Skip Gram (SG) models. The models, hauWE (Hausa Words Embedding), are bigger and better than the only previous model, making them more useful in NLP tasks. To compare the models, they were used to predict the 10 most similar words to 30 randomly selected Hausa words. hauWE CBoW's 88.7% and hauWE SG's 79.3% prediction accuracy greatly outperformed [1]'s 22.3%.**

Keywords—Hausa, words embedding, natural language processing.

The only work that provides word vectors for Hausa language is that of [1]. The work provides trained models for 294 languages including Hausa. The vectors were trained using fastText [8] and the Hausa vectors model consists of a vocabulary of 4347 words only with more than 1500 English words, representing about 40%.

Learning word representations require a lot of data. The performance of the CBoW and SG varies with the amount of data available. Mikolov et al. [9] have found that with more dataset, CBoW model outperforms SG model. While both models perform well with abundant quantities of training data, the SG model provides good representation for rare words in low resource settings.

In this paper, the work presented describes the processes involved in generating hauWE, an open-source distrib-

Massive vs. Curated Embeddings for Low-Resourced Languages: the Case of Yorùbá and Twi

Jesujoba O. Alabi[⋆†‡] **Kwabena Amponsah-Kaakyire**[⋆†‡] **David I. Adelani**[‡‖] **Cristina España-Bonet**[†‡]
[†]DFKI GmbH, Saarbrücken, Germany
[‖]Spoken Language Systems (LSV), Saarland Informatics Campus, [‡]Saarland University, Saarbrücken, Germany
{jesujoba_oluwadara.alabi, kwabena.amponsah-kaakyire, cristinae}@dfki.de, didelani@lsv.uni-saarland.de

Abstract

The success of several architectures to learn semantic representations from unannotated text and the availability of these kind of texts in online multilingual resources such as Wikipedia has facilitated the massive and automatic creation of resources for multiple languages. The evaluation of such resources is usually done for the high-resourced languages, where one has a smorgasbord of tasks and test sets to evaluate on. For low-resourced languages, the evaluation is more difficult and normally ignored, with the hope that the impressive capability of deep learning architectures to learn (multilingual) representations in the high-resourced setting holds in the low-resourced setting too. In this paper we focus on two African languages, Yorùbá and Twi, and compare the word embeddings obtained in this way, with word embeddings obtained from curated corpora and a language-dependent processing. We analyse the noise in the publicly available corpora, collect high quality and noisy data for the two languages and quantify the improvements that depend not only on the amount of data but on the quality too. We also use different architectures that learn word representations both from surface forms and characters to further exploit all the available information which showed to be important for these languages. For the evaluation, we manually translate the wordsim-353 word pairs dataset from English into Yorùbá and Twi. We extend the analysis to contextual word embeddings and evaluate multilingual BERT on a named entity recognition task. For this, we annotate with named entities the Global Voices corpus for Yorùbá. As output of the work, we provide corpora, embeddings and the test suits for both languages.

AfriVEC: Word Embedding Models for African Languages. Case Study of Fon and Nobiin

Bonaventure F. P. Dossou
Jacobs University Bremen
f.dossou@jacobs-university.de

Mohammed Sabry
University of Khartoum
mhmd.sabry.ab@gmail.com

Abstract

From Word2Vec to GloVe, word embedding models have played key roles in the current state-of-the-art results achieved in Natural Language Processing. Designed to give significant and unique vectorized representations of words and entities, those models have proven to efficiently extract similarities and establish relationships reflecting semantic and contextual meaning among words and entities. African Languages, representing more than 31% of the worldwide spoken languages, have recently been subject to lots of research. How-

padla et al., 2016). This makes it for the sake of scientific research on African Languages very important to consider, because of their scarse data resources. Throughout this paper, our main contribution is to provide standardization and evaluation guidelines to any research on the space, through our methods and experiments.

Fon, and Nobiin are the two African Indigenous Languages (ALs) chosen for this study. Fon is a native language of Benin Republic, spoken in average by more than 2.2 million people in Benin, in Nigeria, and Togo. Nobiin is native to Northern

Problems of Word2vec

Context-Independence

- Each word has a single, fixed vector regardless of the sentence or context it appears in.
- Cannot handle polysemy (*same form, different but related meanings*) or homonymy (*same form, different unrelated meanings*.).

Out-of-Vocabulary (OOV) Words

- Words not seen during training cannot be represented.
- No mechanism for dynamically generating embeddings for new words.

Ignores Word Order

- Treats words as a "bag of words" within a context window.
- Cannot capture syntactic structure or word order information.

Limited Handling of Morphology

- Morphologically similar words (e.g., run, running, ran) are treated as distinct.
- No awareness of subword structures like prefixes, suffixes, or stems.

Static Embeddings

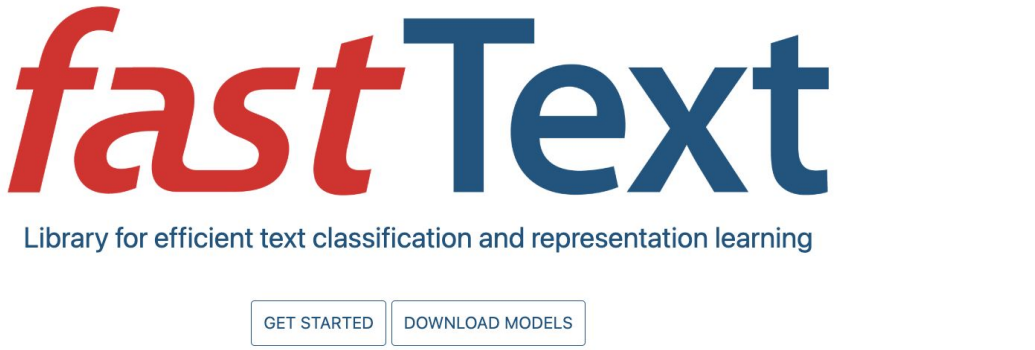
- Does not adapt to different tasks or domains after training.
- Performance degrades in domain-specific or dynamic language contexts.

Requires Large Corpora

- Needs extensive training data to learn good representations.
- Performance is poor on smaller datasets.

Fasttext embedding - Subword embedding

- **FastText embeddings** are an extension of Word2Vec developed by Facebook AI Research (FAIR) that improve word representation by incorporating *subword information.* https://fasttext.cc/



Fasttext embedding - Subword embedding

Subword Units:

- Every word is split into character *n*-grams (subword chunks), and each *n*-gram is given its own embedding.

- Special boundary symbols < and > are added to the word before splitting, to capture prefixes and suffixes. For example, with ***n = 3*:**

  Word: where

  Represented as: where, <wh, whe, her, ere, re>

In Fasttext, vocabulary is subwords, while in word2vec vocabulary is set of words

Fasttext embedding - Subword embedding

The skip-gram model (like in Word2Vec) is used for training, but now applied to each *n*-gram instead of just full words.

Word Representation:

1. The final embedding for a word is the sum of the embeddings of its subword n-grams.
   So where = sum (embedding of each n-gram)

Handling Unknown Words:

1. If a word wasn't seen during training (OOV), it can still be represented based on its subword components.
2. This allows FastText to handle unseen or rare words effectively.

Key Features of FastText:

- **Better Morphological Understanding**
  - Especially useful for *morphologically rich languages*.
  - Captures similarities between inflected or derived forms (e.g., “run”, “runner”, “running”).
- **Pretrained Models**
  - Available in multiple languages and trained on large corpora (e.g., Wikipedia, Common Crawl).

Disadvantages of FastText

- Subword embeddings increase the number of parameters significantly. Storing vectors for every n-gram (e.g., 3 to 6 characters) results in larger models than Word2Vec.
- Due to computing and summing multiple n-gram vectors for every word: Training is slower compared to Word2Vec.
- Limited Contextual Representation: Like Word2Vec, FastText produces static word embeddings.
- No Built-in Sentence Representation: FastText only produces word embeddings. For sentence/document embeddings, additional steps (averaging, pooling, etc.) are required

GloVe: Global Vectors for Word Representation

GloVe: Global Vectors for Word Representation

Jeffrey Pennington, Richard Socher, Christopher D. Manning

Introduction

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

Getting started (Code download)

- Download the latest latest code (licensed under the Apache License, Version 2.0).
  Look for "Clone or download"
- Unpack the files: unzip master.zip
- Compile the source: cd GloVe-master && make
- Run the demo script: ./demo.sh
- Consult the included README for further usage details, or ask a question

Download pre-trained word vectors

- Pre-trained word vectors. This data is made available under the Public Domain Dedication and License v1.0 whose full text can be found at: http://www.opendatacommons.org/licenses/pddl/1.0/.
  - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip
  - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
  - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
  - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): glove.twitter.27B.zip
- Ruby script for preprocessing Twitter data

Citing GloVe

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. [pdf] [bib]

Global Vectors for Word Representation

Word2Vec and FastText learn word embeddings by predicting words from local contexts

Question: Can we instead directly learn word vectors from **global co-occurrence statistics**?

GloVe was proposed to leverage this idea — capturing word meaning by analyzing ***how frequently words appear*** together across an entire corpus.

Properties of GloVe Embeddings

High quality semantic and syntactic similarity:

- "king" - "man" + "woman" ≈ "queen"

Superior performance on:

- Analogy tasks
- Semantic similarity

ACL Test-of-Time Award

- The **ACL Test-of-Time Award** is a prestigious recognition presented annually by the Association for Computational Linguistics (ACL) to honor papers that have made a lasting impact on the field of Natural Language Processing (NLP).

- The award typically honors papers published 10 to 12 years prior to the award year.
  - GloVe: Global Vectors for Word Representation (**2014**)
  - Devlin et al. (2018), *BERT (**2023**)*
  - Mikolov et al. (2013), *Word2Vec (**2021**)*


Shamsuddeen Hassan Muhammad, PhD
@Shmuhammadd
The ACL 2024 Test of Time Award was given to the paper titled "GloVe: Global Vectors for Word Representation." #ACL2024NLP #ACL2024
GloVe: Global Vectors for Word Representation (2014)
Jeffrey Pennington, Richard Socher, and Christopher D. Manning
Stanford University
Justifications
GloVe: Global Vectors for Word Representation
Conference on Empirical Methods in Natural Language Processing
Jeffrey Pennington, R. Socher, Christopher D. Manning
Word embeddings are the cornerstone of deep learning methods for NLP between 2013 and 2018 and have continued to exert significant influence. Not only do they enhance the performance of NLP tasks, but they also have notable computational semantics impacts, such as on word similarity and analogy. The two most influential methods for word embeddings are likely skip-gram/CBOW and GloVe. Compared to skip-gram, GloVe was proposed later. Its relative strength lies in its conceptual simplicity, directly optimizing vector space similarity based on the distributional characteristics between words, rather than indirectly as a set of parameters from a simplified language modeling perspective.

Further Reading

Paper: https://nlp.stanford.edu/pubs/glove.pdf

Blog: https://jonathan-hui.medium.com/nlp-word-embedding-glove-5e7f523999f6

We will see how we can use these separately trained word embeddings (or train/update embeddings on-the-fly) as we perform language modeling using Neural Nets!

IMPERIAL

Q and A