Text representation Methods

Dr. Nourelhouda ZERARKA

Outline

What is Text representation?

Feature representation is a common step in any ML project, whether the data is text, images, videos, or speech. However, feature representation for text is often much more involved as compared to other formats of data.

Often in NLP, feeding a good text representation to an ordinary algorithm will get you much farther compared to applying a topnotch algorithm to an ordinary text representation.

What is Text representation?

- Process of converting text into numerical vectors
- Essential step for machine learning algorithms
- Enables mathematical operations on text data

Basic vectorization approaches

Basic idea of text representation:

- map each word in the vocabulary (V) of the text corpus to a unique ID (integer value),
- Then, represent each sentence or document in the corpus as a V-dimensional vector.

One-Hot Encoding

- Each word gets a unique binary vector
- Vector length equals vocabulary size
- Only one '1', rest are '0's

Concept

So, each word is represented by a V-dimensional binary vector of 0s and 1

Example

Vocabulary: ["hello", "world", "python"]

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

One-Hot Encoding Limitations

Key Limitations

- Sparse representation
- High dimensionality
- No semantic meaning
- No word relationships

Bag of Words

Concept

1. Tokenize the text (split it into words).
2. Create a Vocabulary of unique words from all the documents.
3. Count how many times each word appears in each document.

Why to use BoW

- Counts word occurrences
- Ignores word order
- Simple but effective approach

Bag of Words

Example

Text: "The cat sat on the mat"
Vocabulary:{"the", "cat", "sat", "on","mat"}
BoW: [2,1,1,1,1]

N-grams

What are N-grams?

- Continuous sequence of n items
- Captures local context
- Common types:
  - Unigrams (n=1)
  - Bigrams (n=2)
  - Trigrams (n=3)

N-grams Example

Text: "The quick brown fox"

- Unigrams:
  - "The", "quick", "brown", "fox"
- Bigrams:
  - "The quick", "quick brown", "brown fox"
- Trigrams:
  - "The quick brown", "quick brown fox"

Bag of N-gram

Concept

The Bag of N-grams method converts n-grams into a numerical vector by:

- Counting the frequency of each n-gram in the text.
- Representing the text as a vector of these frequencies.

Bag of n-grams

Example

corpus = [
"The quick brown fox",
"The quick fox jumps",
"The brown fox is quick"
]
Vocabulary of tri-grams: ['brown fox is', 'fox is quick', 'quick brown fox', 'quick fox jumps', 'the brown fox', 'the quick brown', 'the quick fox']
Bag of Trigrams Representation:

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

TF-IDF

What is TF-IDF?

- TF (Term Frequency):
  - How often word appears in document
  - $TF(t, d) = \frac{\text{Number of occurrences oft in d}}{\text{total number of words in d}}$
- IDF (Inverse Document Frequency):
  - How unique the word is across documents. ie it discounts terms that appear frequently across the entire corpus, reducing their importance.
  - $IDF(t) = \log \frac{N}{1+df(t)}$
  - N: total documents
  - df(t): documents containing t

Example of TF-IDF

- Doc1: "The cat sat on the mat"
- Doc2: "The dog barked at the cat"
- Doc3: "The dog and the cat played together"

calculating TF of t="the"

For Doc1: there are 6 words in the document: $TF(the) = \frac{2}{6} = 0.33$

calculating IDF of t="the"

"the" appears in all 3 documents: $IDF(the) = \log(\frac{3}{1+3}) = -0.12$

TF-IFD(the)=0.0396

Applications in Practice

Sentiment Analysis

- Product reviews
- Social media analysis
- Customer feedback

Document Classification

- Email spam detection
- News categorization
- Topic modeling

Search Engines

- Query processing
- Document ranking
- Relevance scoring

Summary

- **One-Hot Encoding**
  - Simple but sparse
  - Good for small vocabularies
- **Bag of Words**
  - Captures frequency
  - Loses word order
- **Bag of N-grams**
  - Preserves local context
  - Increases vocabulary size
- **TF-IDF**
  - Balances frequency and uniqueness
  - Good for document comparison

To conclude

These simple methods are often more suitable for small dataset because they are computationally efficient and work well with limited data.

Questions?

Questions!

What are Distributed Representations?

- Distributed representations are dense vector representations of text that capture semantic and syntactic information.
- Key characteristics:
  - Low-dimensional dense vectors
  - Capture contextual and semantic relationships
  - Enable computational processing of text
- Fundamental shift from sparse, one-hot representations

Main Types of Distributed Representations

1. Word Embeddings
2. Sentence Embeddings
3. Contextual Embeddings
4. Document Embeddings

Word Embeddings: Conceptual Understanding

- Dense vector representations of words
- Capture semantic relationships between words
- similar words (similar context) have similar word embedding

Example of Semantic Relationships:

$$\text{king} - \text{man} + \text{woman} \approx \text{queen}$$

Word Embeddings: Manual Example

Hypothetical Word Vector Space

Consider a 3-dimensional word embedding space:

- *Semantic Axis 1 (Emotion)*: Negative $\rightarrow$ Positive
- *Semantic Axis 2 (Intensity)*: Weak $\rightarrow$ Strong
- *Semantic Axis 3 (Concreteness)*: Abstract $\rightarrow$ Concrete

Example word placements:

- "happiness": [0.8, 0.7, -0.3]
- "sadness": [-0.8, 0.6, -0.4]
- "mountain": [0.2, 0.9, 0.9]

Word Embedding Models: Overview

- Three primary word embedding approaches:
  1. Word2Vec
  2. GloVe (Global Vectors)
  3. FastText
- Each model offers unique approach to capturing word semantics

Word2Vec: Conceptual Framework

Word2Vec Approaches:

- Word2Vec transforms words into numerical vectors (embeddings) such that words with similar meanings or contexts have vectors that are close together in the embedding space. (Predict surrounding words based on context)
- Two primary architectures:
  - Continuous Bag of Words (CBOW)
  - Skip-gram Model

Word2Vec can be used to represent:

- Individual words.
- Sentences (by aggregating word vectors).
- Entire documents (by further aggregation)

Word2Vec: how it works

The goal of Word2Vec is to predict either

- a context words form a word if it uses skip-gram
- or a target word from context words it it uses CBOW

It uses a neural network with a 1 hidden layer
it uses a large corpus of text to train the Skip-Gram or CBOW model.
During training, word embeddings ((weights in the hidden layer) are adjusted to optimize the prediction task.
So, to Obtain Word Embeddings:

- After training, it extracts the dense vector for each word from the hidden layer.
- These embeddings serve as features for each word. It encodes semantic and syntactic properties of words based on their contexts in the training data.

Word2Vec: Manual Example

Hypothetical Word2Vec Semantic Space

Consider a small corpus about cooking:

- Context words for "knife":
  - "chef" [proximity: 0.8]
  - "cut" [proximity: 0.9]
  - "kitchen" [proximity: 0.7]
- Surrounding context creates vector representation:

$$\text{knife} \approx [0.5, 0.7, -0.2, 0.6, \ldots]$$

Observation: Vector captures semantic and syntactic relationships

GloVe: Global Vectors Approach

GloVe (Global Vectors for Word Representation) is a word embedding technique designed to create dense, distributed vector representations for words. It is widely used in text representation because it captures global statistical information about words in a corpus. Once trained, GloVe provides fixed word embeddings that can be used as features for various NLP tasks.

For a vocabulary of size V, construct a V$\times$V matrix X. The co-occurrence matrix captures how often words occur together within a specified context window in a corpus.

GloVe: Global Vectors Approach

GloVe (Global Vectors) Characteristics:

- Global statistical information about word co-occurrences
- Combines global matrix factorization with local context window
- Explicitly uses word-word co-occurrence statistics

Key Differences from Word2Vec:

- Uses global corpus statistics
- More direct optimization of co-occurrence matrix
- Handles rare words more effectively

GloVe: Manual Example

Co-occurrence Probability Analysis

Example

Co-occurrence Matrix Example:

	cat	dog	runs	jumps
cat	2	0	1	0
dog	0	2	0	1
runs	1	0	3	2
jumps	0	1	2	3

FastText: Subword Information Embedding

FastText extends Word2Vec by incorporating subword information, making it more robust and effective for rare or unseen words. FastText is widely used for text representation in NLP tasks due to its ability to handle out-of-vocabulary words and morphologically rich languages.

FastText Unique Characteristics:

- Represents words as bag of character n-grams
- Handles out-of-vocabulary and morphologically rich words
- Creates embeddings from subword information

Key Advantages:

- Better handling of rare words
- Captures morphological information
- More robust for morphologically complex languages

FastText: Manual Example

Subword Representation Analysis

Consider the word “dogs”:

- Example for "dogs": Subwords: <o, dog, ogs, gs >
  Random embeddings:
  <do: [0.3, -0.2, 0.4]
  dog: [0.5, 0.1, -0.3]
  ogs: [0.2, 0.4, 0.1]
  gs>: [0.1, -0.1, 0.3]
  Sum: [1.1, 0.2, 0.5]
  Average: [0.275, 0.05, 0.125]
  Result:
  Embedding for "dogs": [0.275, 0.05, 0.125]

Observation: Captures both semantic and morphological information

Word Embedding Models: Comparative Analysis

Characteristic	**Word2Vec**	**GloVe**	**FastText**
Context Understanding	Local	Global	Subword
Rare Word Handling	Limited	Moderate	Excellent
Morphological Info	No	No	Yes
Computational Speed	Fast	Moderate	Slower
Global or local context	local	global	local

Conclusion

- Distributed representations transform text processing
- Multiple levels: Word, Sentence, Document
- Capture rich semantic information
- Crucial for modern NLP applications

Questions?

Evolution of Word Representations

Traditional Representations

- One-Hot Encoding
- Sparse Vectors
- No Semantic Meaning

Advanced Embeddings

- Dense Vector Representations
- Semantic Relationships
- Context-Independent

Embedding Evolution

From static representations to dynamic, context-aware embeddings

Embedding Types: A Comparative Analysis

Key Differences

- Representation Complexity
- Semantic Capture Ability
- Computational Requirements

What are Contextual Embeddings?

Definition

The fundamental of Contextual embedding is its ability to generate word embeddings that change based on the surrounding words. This means the representation of a word is not static but dynamically computed based on its context in a specific sentence.

Static Embedding Example:

"bank" $\rightarrow [0.3, 0.7, 0.2]$

Contextual Embedding:

"bank" in river context $\rightarrow [0.1, 0.2, 0.8]$

"bank" in financial context $\rightarrow [0.7, 0.3, 0.1]$

ELMo: Bidirectional Language Modeling- Technical Architecture

Input Processing

1. The input sentence is tokenized
2. Each word is initially represented by its character-level representation

This allows the model to handle out-of-vocabulary words and capture morphological information

Bidirectional Language Model Training

The model is trained on a large corpus to predict:

1. Next word in a sequence (forward direction)
2. Previous word in a sequence (backward direction)

This training helps the model understand contextual relationships

Embedding Computation

- For each word, ELMo combines representations from all layers
- It uses a task-specific weighted combination of these layer representations
- The weights are learned during fine-tuning for specific tasks

BERT: Bidirectional Encoder Representations

Pre-training Process BERT is pre-trained using two innovative techniques:

- Masked Language Modeling (MLM) Some percentage of input tokens are randomly masked. The model tries to predict these masked words. This forces BERT to develop a deep contextual understanding
- Next Sentence Prediction (NSP): The model learns to predict if two sentences are consecutive. Helps understand sentence-level relationships. Useful for tasks like question answering

BERT: Bidirectional Enocder Representation

Multiple layers of self-attention mechanisms

- Each layer captures increasingly complex linguistic features
- Lower layers capture syntactic information
- Higher layers capture semantic understanding

Models

- BERT-base: 12 layers, 110 million parameters
- BERT-large: 24 layers, 340 million parameters

GPT: Unidirectional Contextual Modeling

Unlike traditional contextual embedding models, GPT doesn't just represent words—it generates contextual representations through a predictive process. Each token's embedding is influenced by:

- Previous tokens in the sequence
- Learned probabilistic relationships
- Contextual nuances captured during pre-training

contextual embedding occurs across multiple transformer layers:

- Lower layers capture syntactic information
- Middle layers begin integrating contextual relationships
- Higher layers develop sophisticated semantic representations

RoBERTa: Enhanced BERT Approach

Improvements over Original BERT

- Removes Next Sentence Prediction entirely
- Dynamically changes masking patterns during training
- Increases masking rate and complexity
- Trains on larger batches and for more epochs

Optimization Strategies

1. Modify pretraining approach
2. Increase model robustness
3. Improve semantic understanding

Implementation and Challenges

Computational Requirements

- High GPU demands
- Significant memory usage
- Pre-training complexity

Key Challenges

Balance between representation power and computational efficiency

Emerging Trends in Contextual Embeddings

- More efficient transformer architectures
- Improved contextual understanding
- Reduced computational requirements
- Multilingual and cross-domain representations

Research Frontiers

Bridging semantic understanding across languages and domains

Key Takeaways

Contextual Embedding Highlights

- Dynamic semantic representation
- Context-aware understanding
- Advanced NLP capabilities

(0,0) circle (1) node Contextual Embeddings;

Final Reflection

Contextual embeddings represent a paradigm shift in language understanding

which technique to use

To select the most appropriate text representation technique, you'll need to evaluate several crucial factors:

- Task Complexity
- Available Computational Resources
- Data Characteristics
- Performance Requirements
- Interpretability Needs

Which technique to use

Table: Text Representation: Performance Comparison

Technique	**Semantic Understanding**	**Computational Cost**	**Interpretability**
One-Hot Encoding	Very Low	Low	High
Bag of Words	Low	Moderate	Moderate
TF-IDF	Moderate	Moderate	Moderate
Word2Vec	Good	High	Low
GloVe	Good	High	Low
ELMo	Very Good	Very High	Low
BERT	Excellent	Very High	Low
RoBERTa	Excellent	Very High	Low
GPT	Excellent	Very High	Low

Questions?