Purwadhika
Digital Technology School

Text Representation

Job Connector Program

Outline

- What is Text Representation
- Text Representation Techniques: Bag of Words
- Text Representation Techniques: TFIDF
- Text Representation Techniques: Word2Vec
- Text Representation Techniques: Glove
- Text Representation Techniques: FastText
- Hands On: Text Representation with Python

Purwadhika
Digital Technology School

Text Representation

From Text to Numbers

The Fundamental Question

Purwadhika Digital Technology School

How do we **represent** the meaning of **words** and **documents** as **numbers**?

Human Language: "The cat sat on the mat"

Machine Needs: [0.2, 0.1, 0.8, 0.3, 0.7, 0.4, 0.9, ...]

Challenge: Preserve meaning in numerical form

What is Text Representation?

Purwadhika Digital Technology School

Text representation (also called text embedding) is the process of converting textual data into numerical vectors where:

- Similar texts have similar vector representations
- Mathematical operations on vectors correspond to meaningful relationships
- The dimensionality is manageable for machine learning algorithms

Document D1	*The child makes the dog happy* the: 2, dog: 1, makes: 1, child: 1, happy: 1
Document D2	*The dog makes the child happy* the: 2, child: 1, makes: 1, dog: 1, happy: 1

↓

	child	dog	happy	makes	the	BoW Vector representations
D1	1	1	1	1	2	[1,1,1,1,2]
D2	1	1	1	1	2	[1,1,1,1,2]

How Machine can Understand Text

Purwadhika Digital Technology School

Original	Text Preprocessing	**Text Representation**	Machine Learning
This movie was absolutely AMAZING! I loved every minute of it!!!	movie absolutely amazing loved every minute	Vector of Text: [0.123, 0.213, ..., 0.945]	Text: "movie absolutely amazing loved every minute" Score: 0.92 (Very Positive) Key words: "amazing," "loved"

Purwadhika
Digital Technology School

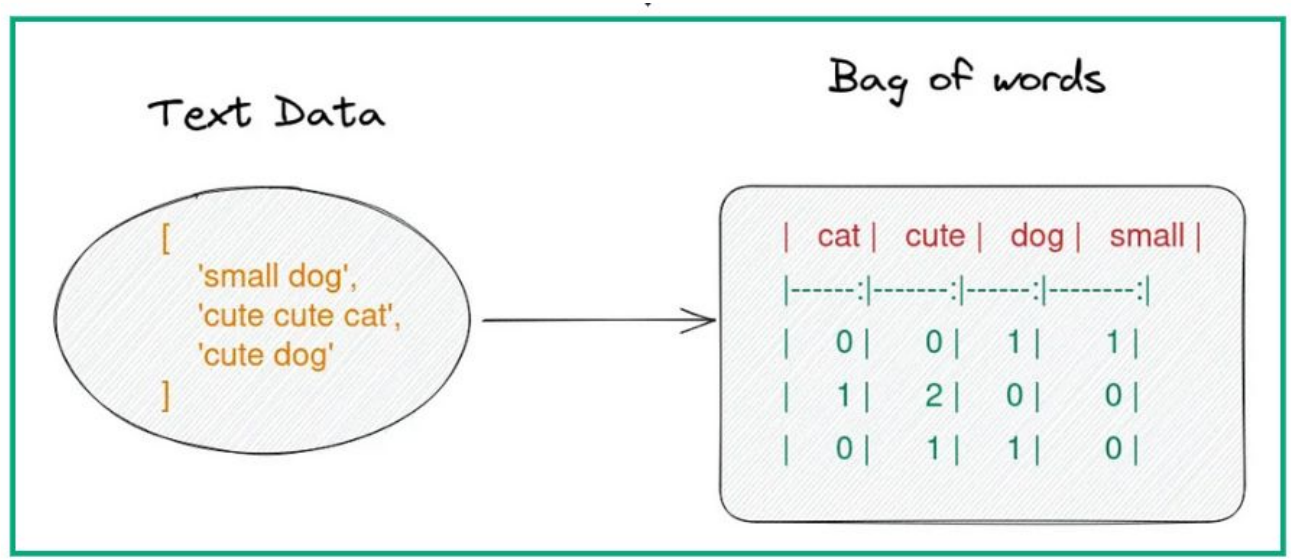
Text Representation

Bag of Words (BoW) Model

Concept and Intuition

The Bag of Words model represents text as a collection of words, disregarding grammar and word order but keeping track of word frequency. Think of it as literally putting all words from a document into a bag and counting them.

Key idea: A document is characterized by the words it contains, regardless of their order.



Step-by-Step Example

Purwadhika Digital Technology School

```python
# Sample documents
doc1 = "I love machine learning"
doc2 = "Machine learning is powerful"
doc3 = "I love powerful algorithms"

# Step 1: Build vocabulary
vocabulary = ["I", "love", "machine", "learning", "is", "powerful", "algorithms"]

# Step 2: Create BoW vectors
doc1_bow = [1, 1, 1, 1, 0, 0, 0]  # [I, love, machine, learning, is, powerful, alg
doc2_bow = [0, 0, 1, 1, 1, 1, 0]
doc3_bow = [1, 1, 0, 0, 0, 1, 1]
```

Implementation

Purwadhika Digital Technology School

```python
from sklearn.feature_extraction.text import CountVectorizer

# Sample documents
documents = [
    "I love machine learning",
    "Machine learning is powerful",
    "I love powerful algorithms"
]

# Create BoW representation
vectorizer = CountVectorizer()
bow_matrix = vectorizer.fit_transform(documents)

# View vocabulary
print("Vocabulary:", vectorizer.get_feature_names_out())
print("BoW Matrix Shape:", bow_matrix.shape)
print("BoW Matrix:\n", bow_matrix.toarray())
```

Additional: N-Grams

Definition: N-grams are sequences of N consecutive words from a text.

Purpose: Capture word relationships and context that individual words miss.

Sample Text Original sentence: "I love machine learning"

Different N-gram Types

1. **Unigrams (1-gram)** Individual words: "I", "love", "machine", "learning"
2. **Bigrams (2-grams)** Consecutive word pairs: "I love", "love machine", "machine learning"
3. **Trigrams (3-grams)** Consecutive word triplets: "I love machine", "love machine learning"
4. **4-grams** Consecutive word quadruplets: "I love machine learning"

Real-World Applications

Search Engines:

- Query: "machine learning" (bigram)
- Better than searching "machine" + "learning" separately

Text Classification:

- Email spam detection using phrase patterns
- "Click here now" (trigram) → spam indicator

Key Benefits

- **Context preservation:** Captures word relationships
- **Better features:** More informative than individual words
- **Improved accuracy:** Better performance in NLP tasks

Trade-off: Higher n-grams = more context but larger vocabulary size

Advantages and Disadvantages

Purwadhika Digital Technology School

Advantages:

- Simple to understand and implement
- Computationally efficient
- Works well for document classification
- Interpretable results

Disadvantages:

- Ignores word order and context
- High dimensionality with large vocabularies
- Sparse representations
- No semantic understanding
- Treats all words equally (no importance weighting)

When to use BoW:

- Document classification tasks
- Simple text analysis
- When interpretability is crucial
- Small to medium vocabulary sizes

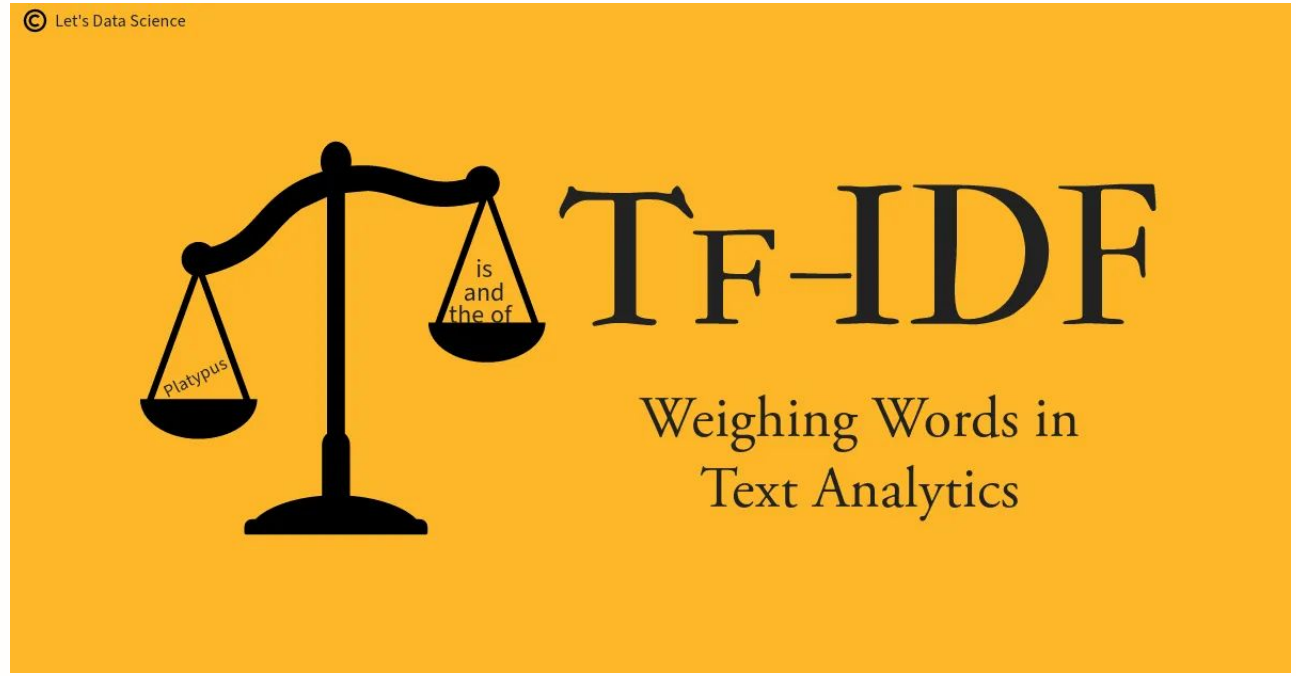
Purwadhika
Digital Technology School

Text Representation

TF-IDF (Term Frequency-Inverse Document Frequency)

Concept and Intuition

TF-IDF addresses a **key limitation of BoW**: not **all words are equally important**. Words that appear frequently in a specific document but rarely across the corpus are **more characteristic** and should be **weighted higher**.



Intuition:

- **Term Frequency (TF)**: How often does a word appear in a document?
- **Inverse Document Frequency (IDF)**: How rare is this word across all documents?
- **TF-IDF**: Combines both to identify characteristically important words

Step-by-Step Example

Purwadhika Digital Technology School

python

```python
import math
from collections import Counter

# Sample corpus
corpus = [
    "the cat sat on the mat",
    "the dog ran in the park",
    "cats and dogs are pets"
]

# Calculate TF-IDF for word "cat" in first document
def calculate_tf_idf(term, document, corpus):
    # Term Frequency
    doc_words = document.split()
    tf = doc_words.count(term) / len(doc_words)

    # Inverse Document Frequency
    docs_with_term = sum(1 for doc in corpus if term in doc.split())
    idf = math.log(len(corpus) / docs_with_term)

    # TF-IDF
    return tf * idf

# Example calculation
tf_idf_cat = calculate_tf_idf("cat", corpus[0], corpus)
print(f"TF-IDF for 'cat' in first document: {tf_idf_cat:.4f}")
```

python

```python
from sklearn.feature_extraction.text import TfidfVectorizer

# Sample documents
documents = [
    "the cat sat on the mat",
    "the dog ran in the park",
    "cats and dogs are pets"
]

# Create TF-IDF representation
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(documents)

# View results
print("Vocabulary:", tfidf_vectorizer.get_feature_names_out())
print("TF-IDF Matrix Shape:", tfidf_matrix.shape)
print("TF-IDF Matrix:\n", tfidf_matrix.toarray())
```

Example Calculation

Sample Documents

1. Document 1: "the cat sat on the mat"
2. Document 2: "the dog ran in the park"
3. Document 3: "cats and dogs are pets"

Step 1: Term Frequency (TF):

Count word occurrences in each document

1. Document 1: [the: 2, cat: 1, sat: 1, on: 1, mat: 1] → Total: 6 words
2. Document 2: [the: 2, dog: 1, ran: 1, in: 1, park: 1] → Total: 6 words
3. Document 3: [cats: 1, and: 1, dogs: 1, are: 1, pets: 1] → Total: 5 words

Step 2: Calculate TF: (Word count in document) / (Total words in document)

For word "the":

Doc 1: 2/6 = 0.33
Doc 2: 2/6 = 0.33
Doc 3: 0/5 = 0.00

For word "cat":

Doc 1: 1/6 = 0.17
Doc 2: 0/6 = 0.00
Doc 3: 0/5 = 0.00

Step 3: Document Frequency (DF) & IDF

DF: Number of documents containing the word
IDF: log(Total documents / DF)
Word "the": DF = 2, IDF = log(3/2) = 0.18
Word "cat": DF = 1, IDF = log(3/1) = 0.48

Step 4: Calculate TF-IDF

TF-IDF = TF × IDF
For "the" in Doc 1: 0.33 × 0.18 = 0.06
For "cat" in Doc 1: 0.17 × 0.48 = 0.08

Result Interpretation

- **"the"** has low TF-IDF (common word, appears in many documents)
- **"cat"** has higher TF-IDF (less common, more distinctive)

Key Insight: TF-IDF gives higher scores to words that are frequent in a document but rare across the entire collection.

Advantages and Disadvantages

Purwadhika Digital Technology School

Advantages:

- Weights words by importance
- Reduces impact of common words
- Better than BoW for most tasks
- Still interpretable and sparse

Disadvantages:

- Still ignores word order and context
- High dimensionality problems persist
- Assumes term independence
- Sensitive to document length

When to use TF-IDF:

- Document similarity and retrieval
- Text classification with importance weighting
- Feature selection for traditional ML models
- Information retrieval systems

Purwadhika
Digital Technology School

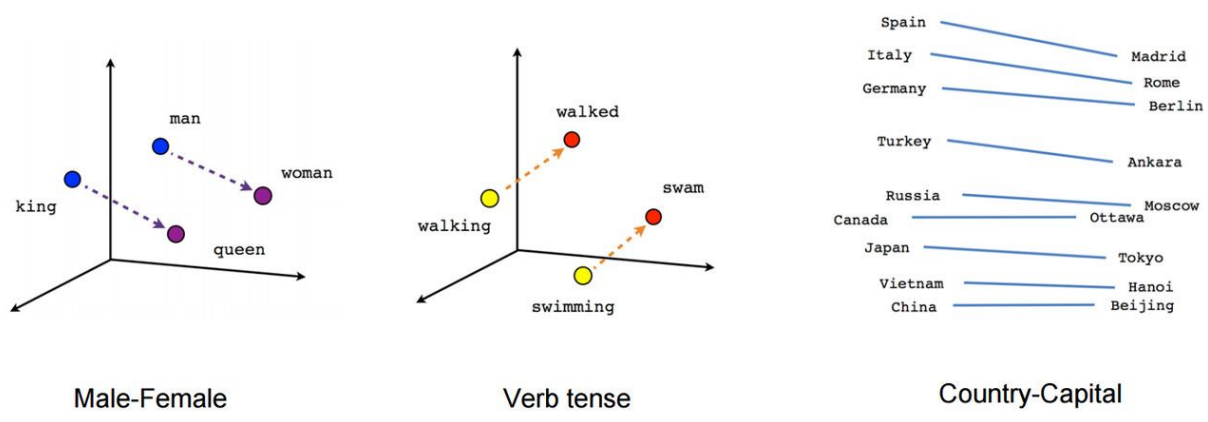
Text Representation

Word2Vec: Learning Word Meanings

Concept and Intuition

Word2Vec revolutionized text representation by learning dense vector representations that capture semantic relationships between words. Unlike BoW and TF-IDF, Word2Vec creates representations where similar words have similar vectors.

Key insight: "You shall know a word by the company it keeps" - words appearing in similar contexts have similar meanings.



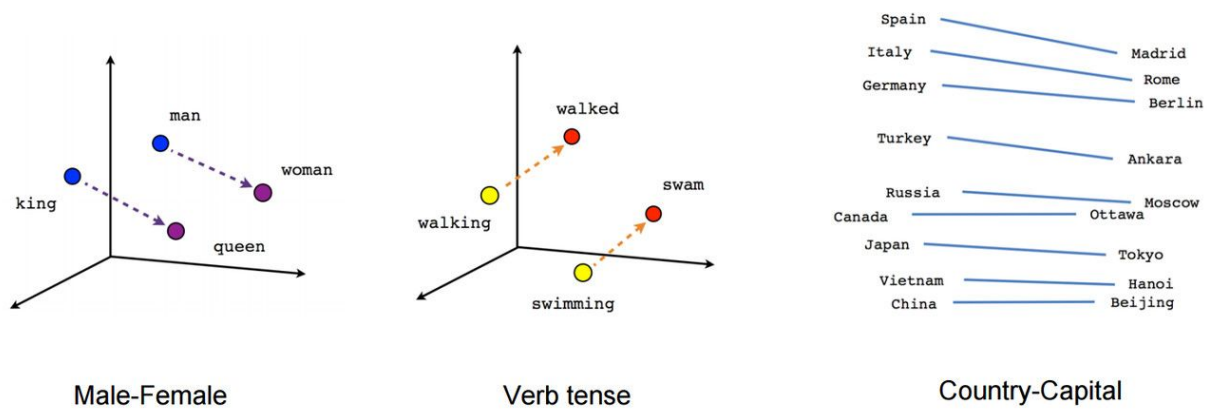
Male-Female Verb tense Country-Capital

Concept and Intuition

The Distributional Hypothesis

Words that occur in similar contexts tend to have similar meanings:

- "dog" and "cat" both appear with words like "pet", "animal", "cute"
- "king" and "queen" both appear with words like "royal", "crown", "palace"



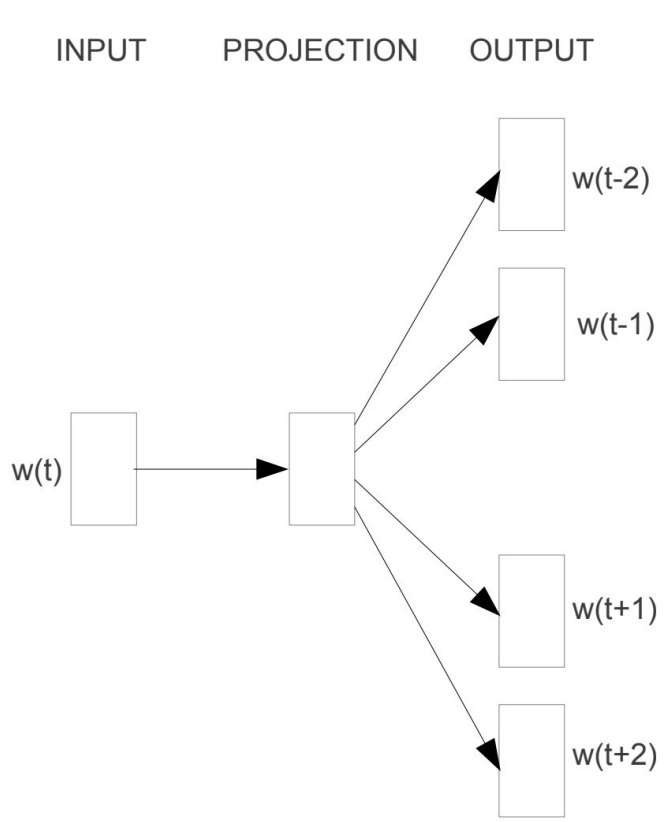
Male-Female | Verb tense | Country-Capital

Concept and Intuition

Purwadhika Digital Technology School

Skip-gram Model

- **Goal:** Given a center word, predict surrounding context words
  - Input: "king"
  - Output: Probabilities for ["royal", "crown", "palace", "queen", ...]
- **Architecture:**
  - Input Word → Embedding Layer → Output Layer (Softmax)



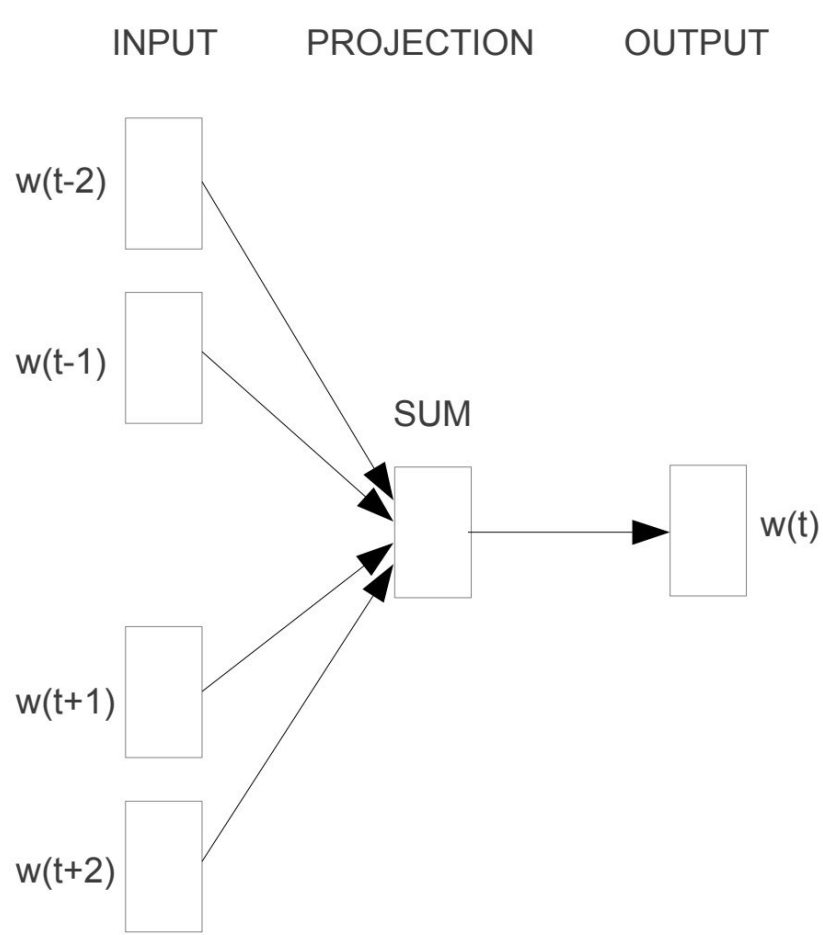
Skip-gram

Concept and Intuition

Purwadhika Digital Technology School

Continuous Bag of Words (CBOW)

- **Goal:** Given context words, predict the center word
  - Input: ["royal", "crown", "palace"]
  - Output: Probability for "king"



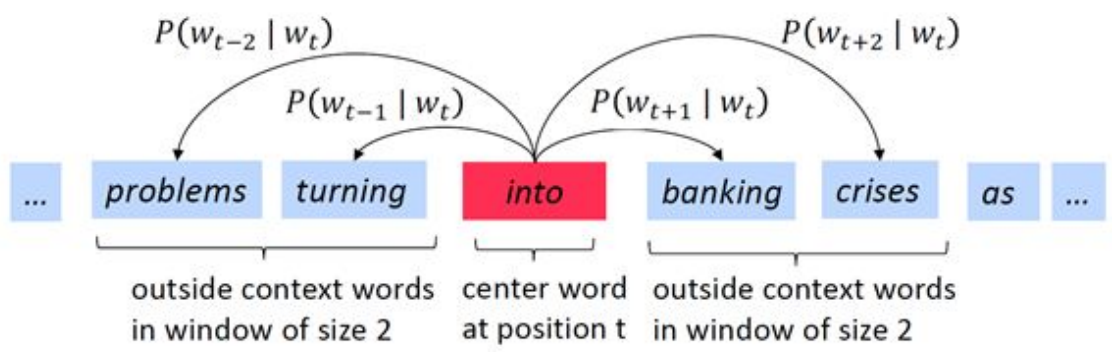
CBOW

Concept and Intuition

Window size determines how many words on each side of a target word are considered as context. For example, with window size 2:

"The quick **brown** fox jumps"

- Context for "brown": ["The", "quick", "fox", "jumps"]



General Guidelines

Small windows (1-3):

- Capture **syntactic** relationships (grammar, parts of speech)
- Better for tasks like POS tagging, parsing
- Example: "quickly" and "fast" (both adverbs) become similar

Large windows (5-10+):

- Capture **semantic** relationships (meaning, topics)
- Better for tasks like document classification, sentiment analysis
- Example: "dog" and "animal" become similar even if rarely adjacent

Implementation

Purwadhika Digital Technology School

Vector Size (Embedding Dimension):

- Range: 50-300 typically
- Larger: More information, but requires more data
- Smaller: Faster training, less overfitting

Window Size:

- Range: 1-10 typically
- Larger: Captures broader context
- Smaller: Focuses on immediate neighbors

Minimum Count:

- Filters out rare words
- Higher values: Smaller vocabulary, faster training
- Lower values: Preserves more words, larger vocabulary

```python
from gensim.models import Word2Vec
import numpy as np

# Sample sentences (tokenized)
sentences = [
    ["king", "is", "a", "strong", "man"],
    ["queen", "is", "a", "wise", "woman"],
    ["the", "man", "is", "strong"],
    ["the", "woman", "is", "wise"],
    ["prince", "is", "a", "young", "man"],
    ["princess", "is", "a", "young", "woman"]
]

# Train Word2Vec model
model = Word2Vec(sentences,
                 vector_size=100,    # Embedding dimension
                 window=5,           # Context window size
                 min_count=1,        # Minimum word frequency
                 workers=4,          # Number of worker threads
                 sg=1)              # Use skip-gram (1) or CBOW (0)

# Get word vectors
king_vector = model.wv['king']
queen_vector = model.wv['queen']

# Find similar words
similar_to_king = model.wv.most_similar('king', topn=3)
print("Words similar to 'king':", similar_to_king)

# Famous analogy: king - man + woman ≈ queen
result = model.wv.most_similar(positive=['king', 'woman'], negative=['man'], topn=
print("king - man + woman =", result)
```

Advantages and Disadvantages

Purwadhika Digital Technology School

Advantages:

- Captures semantic relationships
- Dense, low-dimensional representations
- Enables arithmetic operations on words
- Generalizes well to unseen words in context

Disadvantages:

- Fixed representations (no context sensitivity)
- Requires large training corpus
- Cannot handle out-of-vocabulary words
- Computationally expensive to train

When to use Word2Vec:

- When semantic similarity is important
- Sufficient training data available
- Need dense, low-dimensional representations
- Traditional NLP tasks like sentiment analysis

Purwadhika
Digital Technology School

Text Representation

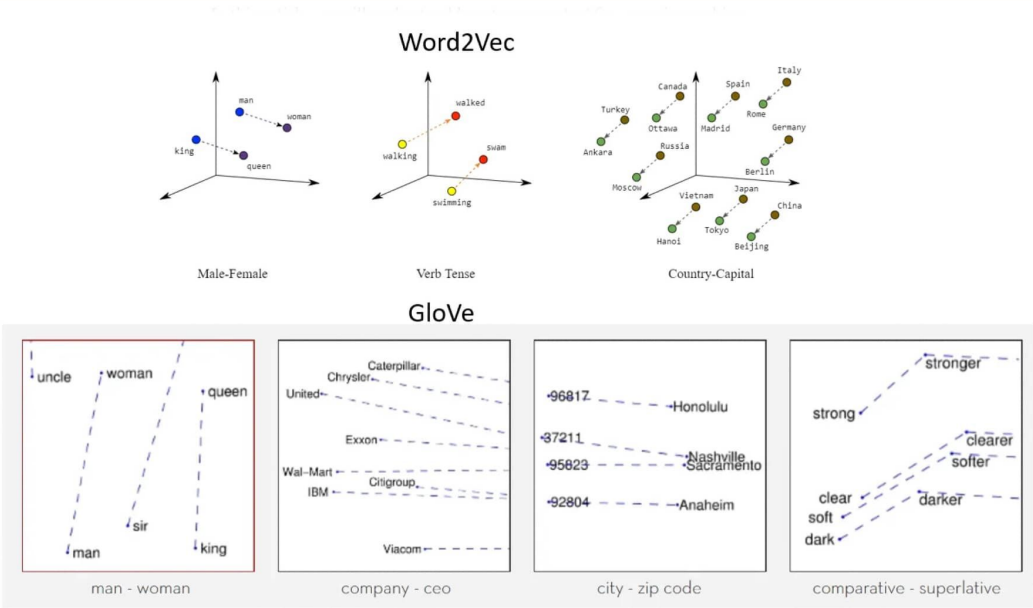
GloVe: Global Vectors for Word Representation

Concept and Intuition

GloVe combines the advantages of two main approaches:

1. **Global matrix factorization** (like LSA) - uses global corpus statistics
2. **Local context window methods** (like Word2Vec) - captures local context

Key insight: Word meaning is determined by global co-occurrence statistics, not just local context windows.




```python
# Example co-occurrence matrix
#          the  cat  sat  on   mat  dog  ran
# the    [[ 0,   2,   1,   2,   1,   1,   1]]
# cat    [[ 2,   0,   1,   0,   1,   0,   0]]
# sat    [[ 1,   1,   0,   1,   0,   0,   0]]
# on     [[ 2,   0,   1,   0,   1,   0,   0]]
# mat    [[ 1,   1,   0,   1,   0,   0,   0]]
# dog    [[ 1,   0,   0,   0,   0,   0,   1]]
# ran    [[ 1,   0,   0,   0,   0,   1,   0]]
```

Glove Example

Sample Corpus:

- "king rules the kingdom"
- "queen rules the queendom"
- "man works hard"
- "woman works hard"

Step 1: Build Co-occurrence Matrix

Count how often words appear together within a window size (for instance in the image, the window size = 1. It means 1 word before and after)

Step 2: Learn Embeddings

GloVe trains vectors so that dot product approximates log probability of co-occurrence.

Result: Similar words get similar vectors:

- king ≈ queen (both rule)
- man ≈ woman (both work)

Word	hard	king	kingdom	man	queen	queendom	rules	the	woman	works
hard	0	0	0	0	0	0	0	0	0	2
king	0	0	0	0	0	0	1	0	0	0
kingdom	0	0	0	0	0	0	0	1	0	0
man	0	0	0	0	0	0	0	0	0	1
queen	0	0	0	0	0	0	1	0	0	0
queendom	0	0	0	0	0	0	0	1	0	0
rules	0	1	0	0	1	0	0	2	0	0
the	0	0	1	0	0	1	2	0	0	0
woman	0	0	0	0	0	0	0	0	0	1
works	2	0	0	1	0	0	0	0	1	0

Glove vs Word2Vec

Aspect	GloVe	Word2Vec
Method	Global co-occurrence matrix	Local context windows
Training	Matrix factorization	Neural networks (CBOW/Skip-gram)
Data Usage	Uses all corpus statistics	Uses local context only
Speed	Faster training	Slower training
Memory	Requires storing co-occurrence matrix	More memory efficient
Performance	Better on word analogy tasks	Better on word similarity tasks

Glove vs Word2Vec

Example Training Process

Let's say we have the sentence: "The cat sat on the mat"

Word2Vec (Skip-gram):

- Input: "cat" → Try to predict: "the", "sat"
- Input: "sat" → Try to predict: "cat", "on"
- Learns through neural network backpropagation

GloVe:

- First builds a co-occurrence matrix counting how often words appear together
- Cat-sat: 1, sat-on: 1, on-the: 1, etc.
- Then factorizes this matrix to learn embeddings

Resulting Vector Properties

Both produce similar quality embeddings, but with subtle differences:

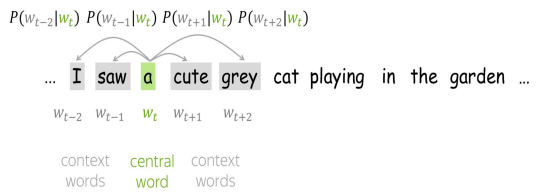
Word2Vec vectors tend to be better at:

- Analogical reasoning (king - man + woman = queen)
- Capturing syntactic relationships

GloVe vectors tend to be better at:

- Word similarity tasks
- Incorporating global corpus statistics
- More stable training on smaller datasets

Local Context Learning



Global co-occurrence matrix

	the	cat	sat	on	mat
the	0	1	0	1	1
cat	1	0	1	0	0
sat	0	1	0	1	0
on	1	0	1	0	0
mat	1	0	0	0	0

Advantages and Disadvantages

Purwadhika Digital Technology School

Advantages:

- Combines global and local information
- More efficient training than Word2Vec on large corpora
- Better performance on word analogy tasks
- Parallelizable training process
- Reproducible results (deterministic)

Disadvantages:

- Requires building large co-occurrence matrix
- Memory intensive for very large vocabularies
- Still produces static word representations
- Limited handling of polysemy (multiple word meanings)

When to use GloVe:

- Large-scale text processing
- Word analogy and similarity tasks
- When training efficiency is important
- Sufficient computational resources for matrix operations

Purwadhika
Digital Technology School

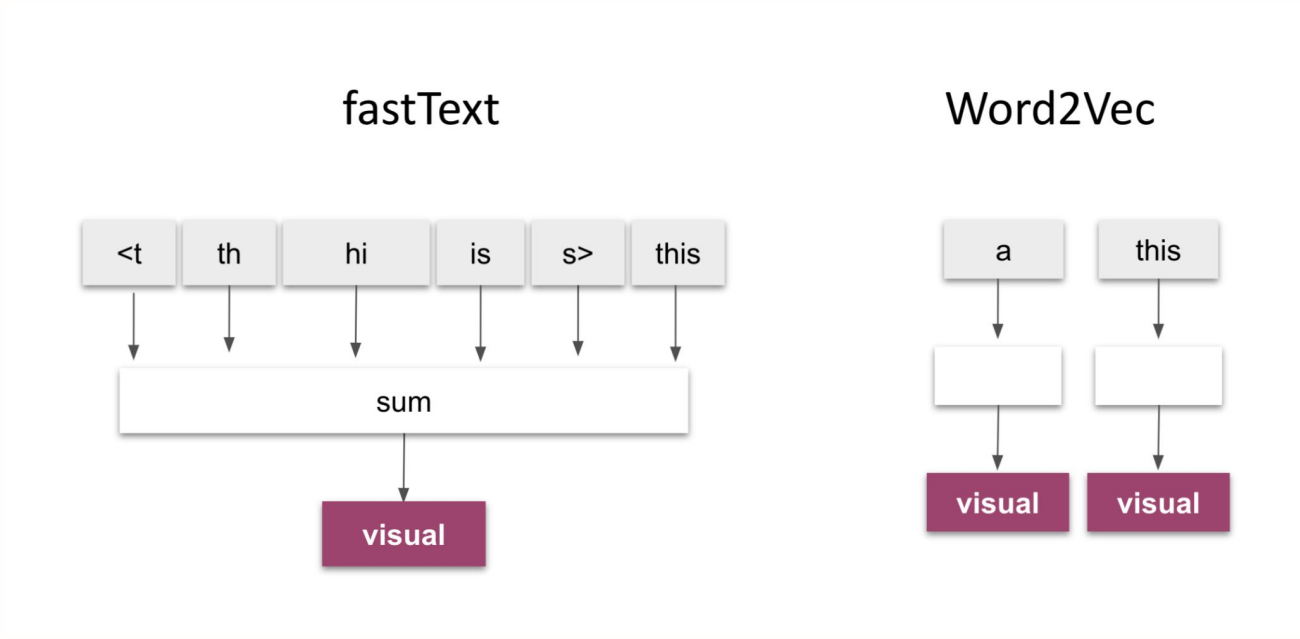
Text Representation

FastText: Subword Information

Concept and Intuition

FastText extends Word2Vec by representing words as bags of character n-grams (subwords), enabling it to handle out-of-vocabulary (OOV) words and capture morphological information.

Key innovation: Instead of treating words as atomic units, FastText breaks them into subword components.



Subword Representation

Purwadhika Digital Technology School

FastText represents each word as a collection of character n-grams

```python
# Example: word "running" with n-grams of size 3-6
word = "running"
subwords = [
    # 3-grams
    "<ru", "run", "unn", "nni", "nin", "ing", "ng>",
    # 4-grams
    "<run", "runn", "unni", "nnin", "ning", "ing>",
    # 5-grams
    "<runn", "runni", "unnin", "nning", "ning>",
    # 6-grams
    "<runni", "runnin", "unning", "nning>",
    # Full word
    "<running>"
]
```

Advantages of FastText

1. Handles Out-of-Vocabulary (OOV) Words

Problem with Word2Vec:

- Training corpus: "learn", "learned", "learning"
- New word: "learnable" → **No embedding available!**

FastText Solution:

- "learnable" shares n-grams with known words:
  - lea, ear, arn (common with "learning")
  - Can generate meaningful embedding even if "learnable" wasn't in training

2. Captures Morphological Relationships

Example Word Family: "teach", "teacher", "teaching", "teaches"

Word2Vec: Treats each as completely separate words **FastText:** Recognizes shared substrings:

- All share tea, eac, ach
- "teacher" and "teaching" share more n-grams → more similar embeddings

3. Better for Morphologically Rich Languages

German Example:

- "Unabhängigkeitserklärung" (Declaration of Independence)
- Even if this exact word wasn't in training, FastText can understand it through:
  - una, nab, abh (from "unabhängig" = independent)
  - erk, rkl, klä (from "erklärung" = declaration)

4. Handles Spelling Variations and Typos

Examples:

- Correct: "definitely"
- Typo: "definately"
- Both share many n-grams: def, efi, fin, ite, ely
- FastText can still provide reasonable embedding for the typo

Use Cases Where FastText Excels

1. Social Media & Informal Text

- Handles slang, abbreviations, elongated words
- Example: "goooood" still relates to "good"

2. Multilingual Applications

- Languages with complex word formation (German, Turkish, Finnish)
- Can understand compound words and inflections

3. Domain-Specific Terminology

- Medical terms, technical jargon with common roots
- Example: "cardiology" and "cardiologist" share meaningful substrings

4. Small Datasets

- When training data is limited
- Subword information helps generalize better

Advantages and Disadvantages

Purwadhika Digital Technology School

Advantages:

- Handles out-of-vocabulary words naturally
- Captures morphological information
- Works well with rare words
- Effective for morphologically rich languages
- Smaller vocabulary requirements

Disadvantages:

- Larger model size (stores subword vectors)
- Slower training and inference
- May generate noise from irrelevant subwords
- Less interpretable than word-level models

When to use FastText:

- Languages with rich morphology (German, Turkish, Finnish)
- Domains with many rare or technical terms
- Limited training data
- Need to handle unseen words at inference time

Purwadhika
Digital Technology School

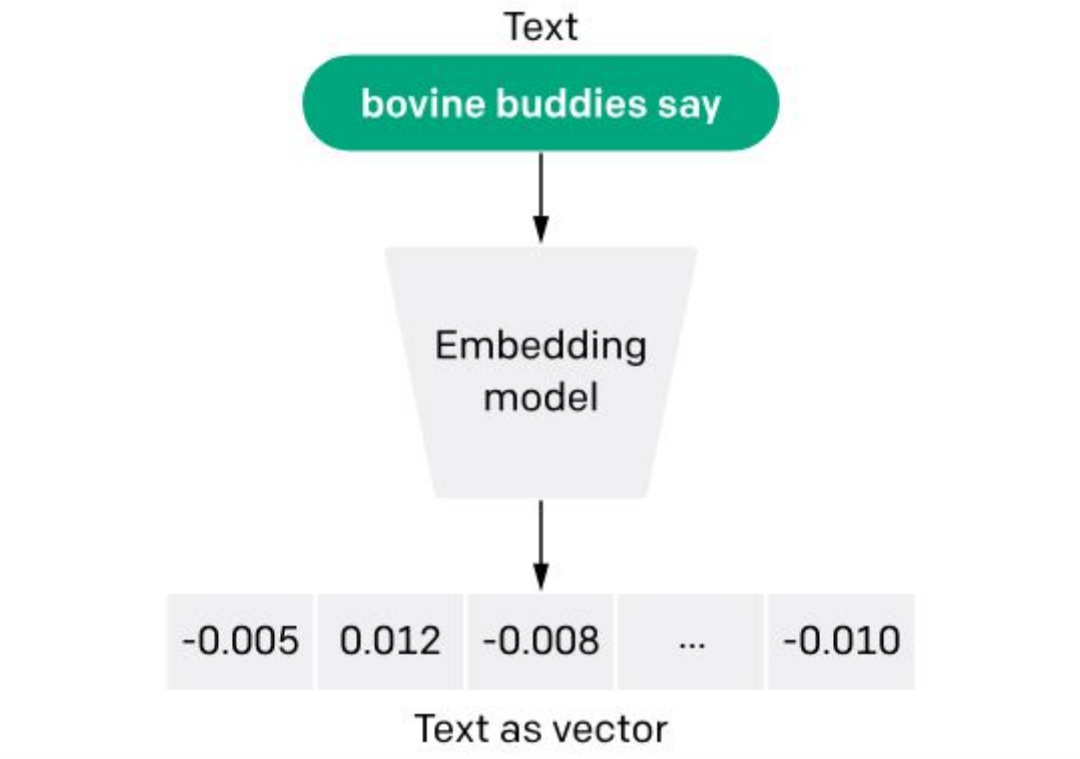
Text Representation

Evaluation Metrics for Text Embeddings

Categories of Evaluation

Text embedding evaluation falls into two main categories:

1. **Intrinsic Evaluation**: Measures embedding quality directly
2. **Extrinsic Evaluation**: Measures performance on downstream tasks



Intrinsic Evaluation Metrics

Purwadhika Digital Technology School

Word Similarity Tasks

Concept: Human-annotated word pairs with similarity scores are compared to cosine similarities between embedding vectors.

Popular datasets:

- WordSim-353: 353 word pairs with human similarity ratings
- SimLex-999: 999 word pairs focusing on true similarity (not relatedness)
- MEN: 3000 word pairs with relatedness scores

```python
from scipy.stats import spearmanr
import numpy as np

def evaluate_word_similarity(embeddings, word_pairs, human_scores):
    """
    Evaluate embeddings on word similarity task

    Args:
        embeddings: Dictionary mapping words to vectors
        word_pairs: List of (word1, word2) tuples
        human_scores: List of human similarity ratings

    Returns:
        Spearman correlation coefficient
    """
    embedding_similarities = []

    for word1, word2 in word_pairs:
        if word1 in embeddings and word2 in embeddings:
            # Calculate cosine similarity
            vec1 = embeddings[word1]
            vec2 = embeddings[word2]

            cosine_sim = np.dot(vec1, vec2) / (np.linalg.norm(vec1) * np.linalg.n
            embedding_similarities.append(cosine_sim)
        else:
            # Handle OOV words
            embedding_similarities.append(0.0)

    # Calculate Spearman correlation
    correlation, p_value = spearmanr(embedding_similarities, human_scores)
    return correlation, p_value

# Example usage
word_pairs = [("king", "queen"), ("dog", "cat"), ("car", "automobile")]
human_scores = [0.8, 0.7, 0.9]
# correlation = evaluate_word_similarity(word_vectors, word_pairs, human_scores)
```

Intrinsic Evaluation Metrics

Purwadhika Digital Technology School

Semantic Coherence

Concept: Measure how well embeddings group semantically related words.

Coherence metrics:

- **Silhouette Score**: Measures cluster quality
- **Intra-cluster vs Inter-cluster distanc** Ratio of within-cluster to between-cluster distances

```python
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans

def evaluate_semantic_coherence(embeddings, word_categories):
    """
    Evaluate semantic coherence using clustering

    Args:
        embeddings: Dictionary mapping words to vectors
        word_categories: Dictionary mapping words to semantic categories

    Returns:
        Silhouette score
    """
    words = list(word_categories.keys())
    vectors = [embeddings[word] for word in words if word in embeddings]
    labels = [word_categories[word] for word in words if word in embeddings]

    # Convert categorical labels to numeric
    unique_categories = list(set(labels))
    label_map = {cat: i for i, cat in enumerate(unique_categories)}
    numeric_labels = [label_map[label] for label in labels]

    # Calculate silhouette score
    score = silhouette_score(vectors, numeric_labels)
    return score
```

Intrinsic Evaluation Example

Purwadhika Digital Technology School

Step 1: Collect Human Ratings for Word Pairs

Human judges rate word pairs on similarity scale (typically 0-10):

- **Example:** "car - automobile" → Human rating: 8.94/10
- Multiple judges rate each pair, then average the scores

Word Pair	Human Rating	Model Similarity
car - automobile	8.94	0.87
gem - jewel	8.96	0.83
journey - car	1.55	0.21
monk - slave	0.92	0.15

Step 2: Calculate Model Similarity between Word Pairs

Compute cosine similarity between embedding vectors:

- Get embedding vectors for both words
- Calculate cosine similarity (result: 0-1 scale)
- **Example:** "car - automobile" → Model similarity: 0.87

Step 3: Use Spearman Correlation to Measure Agreement

Calculate Spearman correlation between human ratings and model similarities:

- Compares ranking order rather than absolute values
- **Result:** Correlation coefficient (0-1, higher = better agreement)
- **Example:** Spearman correlation = 0.65 indicates moderate agreement

Extrinsic Evaluation Metrics

Purwadhika Digital Technology School

Text Classification

Common datasets:

- **Sentiment Analysis**: IMDb movie reviews, Stanford Sentiment Treebank
- **Topic Classification**: 20 Newsgroups, Reuters-21578
- **Spam Detection**: Enron email dataset

Evaluation process:

1. Use embeddings as features for classifier
2. Train classifier on labeled data
3. Measure performance using standard metrics

```python
from sklearn.metrics import accuracy_score, f1_score, classification_report
import numpy as np

def evaluate_text_classification(embeddings, texts, labels, test_size=0.2):
    """
    Evaluate embeddings on text classification task

    Args:
        embeddings: Word embedding model
        texts: List of documents
        labels: List of document labels
        test_size: Fraction of data for testing

    Returns:
        Dictionary with evaluation metrics
    """
    # Convert texts to vectors (simple averaging)
    def text_to_vector(text, embeddings):
        words = text.split()
        vectors = [embeddings[word] for word in words if word in embeddings]
        if vectors:
            return np.mean(vectors, axis=0)
        else:
            return np.zeros(embeddings.vector_size)

    # Convert all texts to vectors
    X = np.array([text_to_vector(text, embeddings) for text in texts])
    y = np.array(labels)

    # Split data
    split_idx = int(len(X) * (1 - test_size))
    X_train, X_test = X[:split_idx], X[split_idx:]
    y_train, y_test = y[:split_idx], y[split_idx:]

    # Train classifier
    classifier = LogisticRegression(random_state=42)
    classifier.fit(X_train, y_train)

    # Make predictions
    y_pred = classifier.predict(X_test)

    # Calculate metrics
    metrics = {
        'accuracy': accuracy_score(y_test, y_pred),
        'f1_score': f1_score(y_test, y_pred, average='weighted'),
        'classification_report': classification_report(y_test, y_pred)
```

Extrinsic Evaluation Example

Task Setup (Sentiment Analysis)

Goal: Classify movie reviews as Positive, Negative, or Neutral

Dataset: IMDB Movie Reviews (50,000 reviews)

- **Training:** 25,000 labeled reviews
- **Testing:** 25,000 labeled reviews

Step 1: Baseline (Without Embeddings)

Use simple bag-of-words approach with keyword counting

- **Method:** Count word frequencies, ignore word relationships
- **Result:** 82% accuracy, 0.81 F1-score

Step 2: Test Different Embeddings

Replace bag-of-words with embedding vectors:

Word2Vec: Average word vectors for each review **Result:** 87% accuracy, 0.86 F1-score

GloVe: Use global co-occurrence statistics **Result:** 85% accuracy, 0.84 F1-score

FastText: Include subword information **Result:** 89% accuracy, 0.88 F1-score

Step 3: Performance Comparison

Approach	Accuracy	F1-Score	Improvement
Baseline (BoW)	82%	0.81	-
Word2Vec	87%	0.86	+5%
GloVe	85%	0.84	+3%
FastText	**89%**	**0.88**	**+7%**

Conclusion

Text representation is the foundation of all NLP tasks - converting human language into numerical vectors that machines can understand and process effectively.

Evolution of Approaches

Traditional Methods (TF-IDF, Bag-of-Words):

- ✅ Simple and interpretable
- ❌ Ignore word order and semantic relationships

Modern Embeddings (Word2Vec, GloVe, FastText):

- ✅ Capture semantic similarities and relationships
- ✅ Dense, efficient representations
- ❌ Static - same word always has same vector

Next Generation (BERT, GPT):

- ✅ Context-aware representations
- ✅ Handle polysemy (multiple word meanings)
- ❌ Computationally expensive

Purwadhika
Digital Technology School

Text Representation

Q&A Session

Purwadhika
Digital Technology School

Thank You!