UNIT: ENGLISH 416

TASK : TRIBAL LINGUISTIC GROUP CAT

LEC: DR. EVANS MECHA

YEAR: 4.1

MEMBERS

1.Simion Mosoti -As15/00455/21

2. Diana Chacha -AS15/00363/21

3.Alex Manyara -AS15/00331/21

Tokenization is the process of splitting text into smaller units (tokens), which are used as input for natural language processing (NLP) models. Different tokenization techniques, including word-level, subword-level, and character-level, have their unique advantages and drawbacks. Here's a comparison of these techniques:

1. Word-Level Tokenization

Definition: Splits text into individual words based on spaces and punctuation.

Examples:

Input: "Tokenization is essential."

Tokens: ["Tokenization", "is", "essential", "."]

Advantages

Easy to implement and understand.

Aligns well with human understanding of language.

Reduces the size of token vocabulary for simpler models.

Disadvantages:

Struggles with unknown or rare words (Out-of-Vocabulary, OOV problem).

Doesn't handle morphological variations effectively (e.g., "run" vs. "running").

Vocabulary size can be very large, especially for languages with rich morphology.

2. Subword-Level Tokenization

Definition: Breaks text into smaller units such as morphemes or common word segments.

Examples:

Input: "Tokenization is essential."

Tokens (using Byte-Pair Encoding): ["Token", "ization", "is", "essential", "."]

Tokens (using WordPiece): ["Token", "##ization", "is", "essential", "."]

Popular Methods:

Byte Pair Encoding (BPE): Repeatedly merges the most frequent pairs of characters or subwords.

WordPiece: Similar to BPE but often used in transformer models like BERT.

SentencePiece: A data-driven approach that uses a language model for segmentation.

Advantages:

Effectively handles rare words and unknown terms by decomposing them into smaller units.

Balances between vocabulary size and generalization.

Enables pre-trained models to adapt to new domains or languages without retraining.

Disadvantages:

Subword tokens can be less interpretable than words.

Requires preprocessing with specialized algorithms.

May result in longer sequences compared to word-level tokenization, increasing computational

cost.

3. Character-Level Tokenization

Definition: Breaks text into individual characters.

Examples:

Input: "Tokenization is essential."

Tokens: ["T", "o", "k", "e", "n", "i", "z", "a", "t", "i", "o", "n", " ", "i", "s", " ", "e", "s", "s", "e", "n", "t", "i", "a", "l", "."]

Advantages:

No Out-of-Vocabulary (OOV) issues.

Handles morphologically rich and low-resource languages well.

Useful for tasks where the structure of words is crucial (e.g., spelling correction, OCR).

Disadvantages:

Results in significantly longer sequences, increasing computational complexity.

Loses semantic information, making it harder to learn meaningful patterns.

Requires more sophisticated models to capture relationships between characters.

Word-Level: Suitable for simpler tasks or where semantic clarity is essential, but struggles with rare words.

Subword-Level: Best for modern NLP models like BERT and GPT, offering a balance of efficiency and generalization.

Character-Level: Ideal for tasks requiring granular text processing or when tackling highly diverse datasets but at the cost of increased complexity.

The choice of tokenization technique depends on the task, language, and computational resources.

The terms "static" and "word embedding" are often used together in natural language processing (NLP) contexts. Let me clarify the distinctions and relationships between them:

1. Word Embedding

A word embedding is a representation of words in a continuous vector space, where similar words are closer in the space. These embeddings are learned from large text corpora and capture semantic relationships between words. Popular word embeddings include Word2Vec, GloVe, and FastText.

2. Static Word Embeddings

Static word embeddings are context-independent representations of words. This means that each word has a single fixed vector representation, regardless of the context in which it appears.

Characteristics:

Fixed Representation: Once trained, a word always has the same embedding.

Context Independence: Cannot differentiate between polysemous words (words with multiple meanings). For example, the word "bank" would have the same embedding whether it refers to a riverbank or a financial institution.

Examples: Word2Vec, GloVe, FastText (though FastText can account for subword information, the word representation is still static).

Advantages:

Computationally efficient and straightforward to use.

Effective for tasks where context sensitivity is less critical.

Limitations:

Fails to capture nuances of context in which words are used.

Limited ability to handle polysemy and word disambiguation.

3. Dynamic (Contextual) Word Embeddings

Dynamic word embeddings (often just called contextual embeddings) are context-dependent representations of words. This means the embedding of a word changes depending on the surrounding words (context). These embeddings are generated by deep learning models, particularly transformers.

Characteristics:

Variable Representation: The same word can have different embeddings in different sentences.

Context Sensitivity: Accounts for polysemy and subtle differences in meaning based on usage.

Examples: BERT, GPT, ELMo, RoBERTa.

Advantages:

Captures the meaning of words more accurately in context.

Superior performance in complex NLP tasks such as sentiment analysis, question answering, and machine translation.

Limitations:

Computationally expensive and requires more resources.

Can be harder to interpret compared to static embeddings.

Key Differences

When to Use Each?

Static embeddings: Suitable for simpler tasks or when computational resources are limited.

Dynamic embeddings: Ideal for tasks requiring nuanced understanding of language, like sentiment analysis or question answering.

Would you like a deeper dive into either type or examples of their applications?