Instituto Politécnico Nacional
"La Técnica al Servicio de la Patria"

DIPLOMADO EN INTELIGENCIA ARTIFICIAL

TRANSFORMERS

PARTE 1 - CORPUS Y TOKENIZACIÓN

ALAN BADILLO SALAS

MÓDULO 11

CONTENIDO

CORPUS Y TOKENIZACIÓN

RNN, LSTM Y GRU

SELF-ATTENTION Y TRANSFORMERS

ARQUITECTURA TRANSFORMER

PREENTRENAMIENTO Y FINE-TUNING

LLMS - ENTRENAMIENTO, ÉTICA Y FUTURO

Problema

Los modelos de **IA generativa basados en textos** utilizan un **corpus** en el proceso de entrenamiento, el cual contiene palabras y estructuras gramaticales que **se traducen en probabilidades** y secuencias que llevan a predicciones, generación de nuevos textos y **respuestas** para las que fueron entrenadas.

Dividir correctamente el corpus en **TOKENS** es la parte fina que determinará el grado de aprendizaje y la **capacidad de predicción** de los modelos.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

- Dividir el lenguaje humano en piezas llamadas **tokens**
- **Unidades estructuradas** que el modelo pueda interpretar
- Habilitar el “entendimiento” y **responder a entradas lingüísticas**
- Capturar los **matices del lenguaje**

Why Tokenization Matters

Tokenization is the foundational process that allows Large Language Models (LLMs) to break down human language into digestible pieces called tokens. This transformation is essential as it translates complex language into structured units that the model can interpret, enabling it to “understand” and respond to various linguistic inputs. Tokenization is not merely about splitting text; it sets the stage for how well an LLM can capture nuances in language, context, and even rare vocabulary.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

- La estrategia de tokenización impacta en el rendimiento del modelo
- El detalle del entendimiento y en general la calidad de la salida
- Reduce la carga computacional
- Captura más el contexto y significado
- Hace respuestas más relevantes, precisas y confiables

The Impact on Model Performance

The choice of tokenization strategy directly impacts how efficiently a model performs, the level of detail it understands, and the overall quality of its output. An effective tokenization strategy reduces computational load by streamlining the model's input, optimizing processing speed, and lowering memory usage. It also improves accuracy by capturing more context and meaning, making responses more relevant, accurate, and reliable.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist · Follow · 21 min read · Oct 30, 2024

- Romper una sentencia en pequeñas piezas llamadas tokens
- Palabras individuales
- Subpalabras
- Caracteres

Tokenization Unpacked: The Basics

Defining Tokenization

Imagine you have the sentence:

> "Tokenization helps models understand language."

When tokenizing this sentence, we break it into smaller pieces called tokens. These tokens could be individual words, subwords, or characters, depending on the tokenization strategy.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist · Follow · 21 min read · Oct 30, 2024

- **Romper una sentencia en pequeñas piezas llamadas tokens**
- Palabras individuales
- Subpalabras
- Caracteres

For instance:

- **Word-level tokens** might split this sentence into: `["Tokenization", "helps", "models", "understand", "language"]`.
- **Subword-level tokens** might break it down as: `["Token", "ization", "helps", "model", "s", "understand", "language"]`.
- **Character-level tokens** would split it further: `["T", "o", "k", "e", "n", "i", "z", "a", "t", "i", "o", "n", " ", "h", "e", "l", "p", "s", " ", "m", "o", "d", "e", "l", "s", " ", "u", "n", "d", "e", "r", "s", "t", "a", "n", "d", " ", "l", "a", "n", "g", "u", "a", "g", "e"]`.

By breaking the sentence into tokens, we help the model process it more easily and consistently. This allows it to learn language patterns without being overwhelmed by full sentences.

- El contexto podría no ser capturado en un caracter, pero si en un conjunto de caracteres, por ejemplo:
  - arbol
  - arboles
  - arboleda
  - romper
  - corromper
  - incorporar
  - corporal
  - corporación
  - oración

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Tokens vs. Characters

Tokens and characters may look similar but serve different purposes. For example, let's take the word "Tokenization":

- **Characters**: Breaking down "Tokenization" by character would yield `["T", "o", "k", "e", "n", "i", "z", "a", "t", "i", "o", "n"]`. While the model sees individual letters, it may not understand the full meaning.
- **Tokenization (Subwords)**: A subword approach, such as Byte Pair Encoding, might split "Tokenization" into `["Token", "ization"]`. Now, the model can recognize "Token" as a meaningful component and "ization" as a suffix, aiding in understanding.

Because tokens capture semantic meaning better than single characters, they help the model understand words, phrases, and context more naturally, even with new or uncommon terms.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

- Ladrillos del Modelo de Grandes Lenguajes
- No hay sentencias, sino secuencias de tokens
- Vector de embebidos (palabras incrustadas en una sentencia)
- Contexto y predicción
  - Secuencia de salida (respuesta)

The Role of Tokens in LLMs

Tokens serve as building blocks in Large Language Models. Using our sentence example, “Tokenization helps models understand language,” the LLM doesn’t read the sentence as we do. Instead, it reads a sequence of tokens, like `["Token", "ization", "helps", "model", "s", "understand", "language"]`.

Here’s how it works:

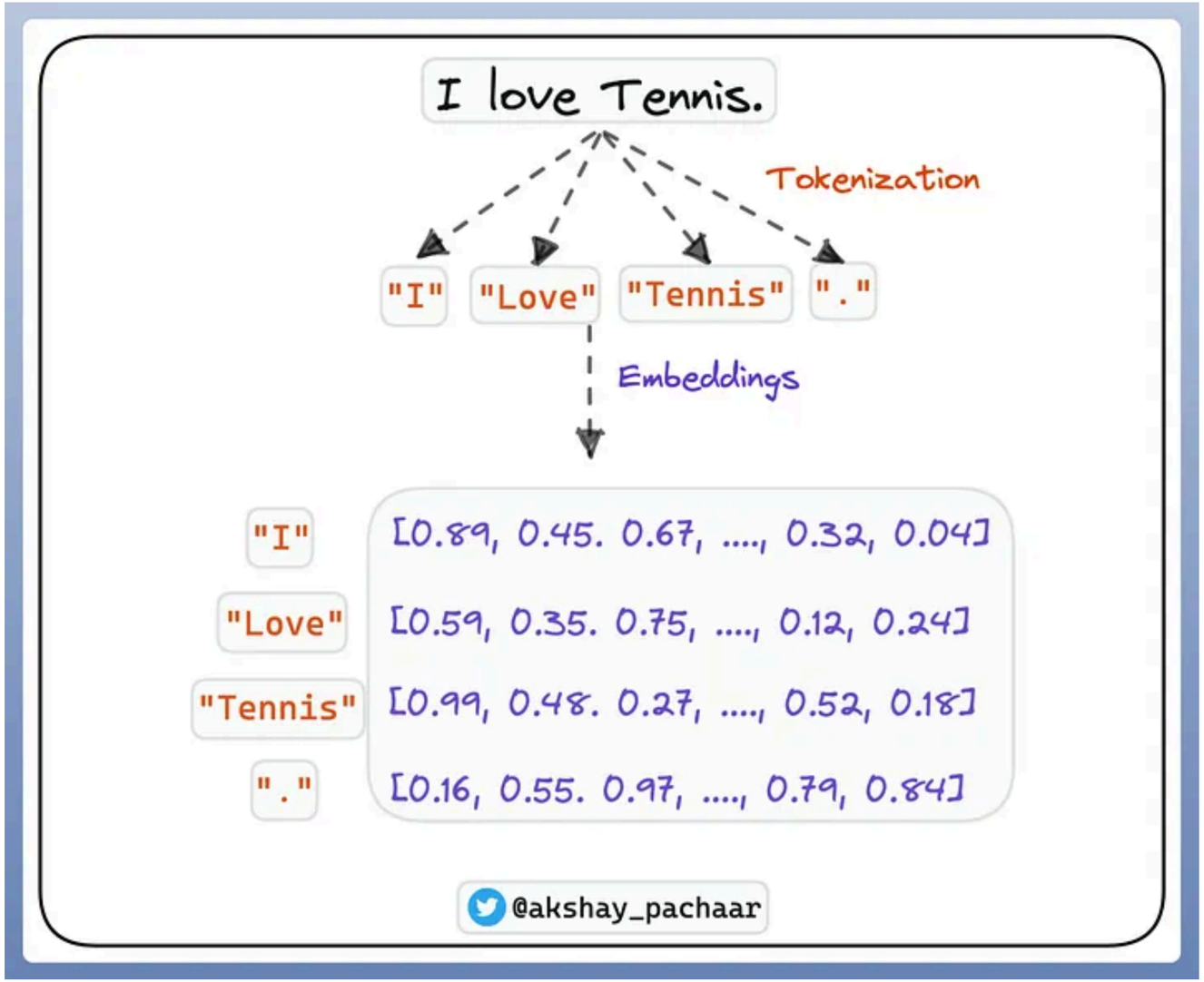
1. **Parsing and Embedding**: Each token is converted into a numerical vector through a process called embedding. These vectors help the model understand relationships between words.
2. **Context and Prediction**: When generating language, LLMs use these token embeddings to predict the most likely next token in a sequence, helping them form coherent and meaningful sentences.

This way, the model processes language as a sequence of tokens, making it more efficient and capable of understanding and generating language that reflects human communication.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

- **Un embedding representa la información semántica y sintáctica** en un token o secuencia en un espacio vectorial
- **Es una proyección es de dimensión fija**
  - 768 posiciones en BERT-base
  - 1024 posiciones en BERT-large
  - 512 posiciones en GPT-2 (small)
- **Captura patrones y relaciones internas** entre tokens y secuencias
- **Permiten construir espacios de comparación**
  - Los tokens o frases con significados similares aparecen cercanos en distancia vectorial (euclídea o coseno)



Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist · Follow · 21 min read · Oct 30, 2024

Tokenización a nivel palabra

- Hay pocas palabras compuestas
- No hay muchas variantes de una palabra
- Consumen menos memoria

Exploring Key Tokenization Techniques

Word-Level Tokenization

Word-level tokenization splits text into individual words, treating each word as a single token. This approach is simple and intuitive, but it has limitations with rare words, compound words, and morphologically rich languages where a word can take many forms.

When to Use It

Word-level tokenization works well when:

- The text is relatively simple, with fewer rare or compound words.
- The language has relatively stable word forms (e.g., English) without complex inflections or derivations.
- You need faster preprocessing and lower memory usage for simpler models or applications.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Tokenización a nivel palabra

- Puede ocurrir el OOV (palabras fuera del vocabulario)
- No captura la complejidad o riqueza morfológica del lenguaje (tiempo, plural, connotaciones)
- Cada palabra es única (alumno, alumna, alumnos, alumnas, alumnado)

Pros and Cons

Pros:

- Simple and intuitive for easy-to-understand tokenization.
- Efficient when dealing with common words.
- Fast to implement and understand.

Cons:

- Struggles with rare words or out-of-vocabulary terms (e.g., “Tokenization” might not be recognized).
- Doesn’t handle complex or morphologically rich languages well (e.g., in some languages, each word can have multiple forms depending on tense, number, etc.).
- Memory inefficiency due to large vocabulary size, as each unique word is a separate token.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist · Follow · 21 min read · Oct 30, 2024

Tokenización a nivel palabra

PRÁCTICA 1

- Implementar un normalizador de español
  - Remueva los acentos y caracteres especiales
  - Convierta las palabras en minúsculas
  - Obtenga la secuencia de palabras normalizadas
  - Extraiga el vocabulario de palabras únicas
  - Obtenga la secuencia de posiciones del vocabulario

Example in Python

Here's a quick demonstration of word-level tokenization using Python with the `nltk` library.

```
import nltk
from nltk.tokenize import word_tokenize
# Download the punkt tokenizer model (if you haven't already)
nltk.download('punkt')
# Sample text
text = "Tokenization helps models understand language."
# Tokenizing text at the word level
word_tokens = word_tokenize(text)
print("Word-Level Tokens:", word_tokens)
```

Output:

```
Word-Level Tokens: ['Tokenization', 'helps', 'models', 'understand', 'language',
```

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist · Follow · 21 min read · Oct 30, 2024

Tokenización a nivel palabra

- Cada palabra es tratada como un token individual
- Los puntos (periodos) son tratados como tokens
- Preserva las letras capitales

Output:

```
ord-Level Tokens: ['Tokenization', 'helps', 'models', 'understand', 'language', '.']
```

Here's a breakdown of the result:

- Each word is treated as an individual token.
- Punctuation marks like the period are also considered separate tokens.
- Although simple, this method doesn't work well if a term is rare or inflected (e.g., "tokenization" is not recognized as "token").

By using word-level tokenization, the model can process the text but might struggle to generalize on unfamiliar terms or words that don't match the training vocabulary exactly.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist · Follow · 21 min read · Oct 30, 2024

Tokenización a nivel subpalabra (BPE)

1. Cada carácter se trata como un token
2. Se identifican los pares de mayor frecuencia
3. Se fusionan los tokens del par de mayor frecuencia
4. Se repite hasta que ya no haya pares con suficiente frecuencia

Byte Pair Encoding (BPE):

Byte Pair Encoding (BPE) is a subword tokenization technique that iteratively merges frequent pairs of characters or subwords in a text corpus. It strikes a balance between word-level and character-level tokenization, allowing for efficient handling of rare or unknown words by breaking them into known subword components.

Here's how it works, step by step:

1. **Start with characters as the initial vocabulary**: Treat each character in the text as a separate token.
2. **Identify the most frequent pairs of tokens**: Scan the corpus to find the pair of tokens (characters or subwords) that appears together most frequently.
3. **Merge the most frequent pair**: Replace all occurrences of this pair with a single new token, effectively creating a subword unit.
4. **Repeat**: Continue merging the most frequent pairs until reaching a specified vocabulary size or until the pairs no longer occur frequently enough to justify merging.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Tokenización a nivel subpalabra (BPE)

- Se realiza la normalización y preprocesamiento del texto
- Se establece el vocabulario inicial con los caracteres únicos

Byte Pair Encoding (BPE) Training Algorithm: A Step-by-Step Explanation

Byte Pair Encoding (BPE) is a popular subword tokenization method that builds a vocabulary iteratively by merging frequently occurring pairs of characters or subwords. Here's how BPE trains a tokenizer, using a simple example.

Step 1: Build the Initial Vocabulary

After preprocessing and normalizing the text, BPE starts with a basic vocabulary that includes all unique characters in the corpus. For instance, let's say our corpus consists of these five words:

```
"hug", "pug", "pun", "bun", "hugs"
```

The initial vocabulary would be all unique characters:

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u"]
```

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Tokenización a nivel subpalabra (BPE)

- Se toma cada palabra del corpus con su frecuencia

Step 2: Initialize Tokenization

The next step involves splitting each word in the corpus into individual characters, resulting in a tokenized form:

```
Corpus:
("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "
```

Each tuple represents a word, broken into its initial tokens, with the associated frequency in parentheses.

```
g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)
```

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist · Follow · 21 min read · Oct 30, 2024

Tokenización a nivel subpalabra (BPE)

- Se toman las parejas consecutivas de cada palabra y se suman las frecuencias (ocurrencias totales)
- Se selecciona la pareja de mayor frecuencia
  - Se actualiza el vocabulario incluyendo a la pareja como un nuevo token
  - Se actualiza el corpus reemplazando los tokens consecutivos por el nuevo token

Merge the Pair and Update the Corpus:

- We merge (“u”, “g”) to form “ug” and add it to the vocabulary:
- `Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug"] Corpus: ("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)`

Step 3: Identify and Merge Frequent Pairs

BPE iteratively finds and merges the most frequent consecutive character pairs to create new tokens. The goal is to gradually build longer subwords, allowing the model to capture common patterns and subwords in the language.

Find the Most Frequent Pair: In our corpus, we count the occurrences of consecutive character pairs.

- Pair frequencies:
- (“h”, “u”) appears 15 times.
- (“u”, “g”) appears 20 times.
- (“p”, “u”) appears 17 times.
- (“u”, “n”) appears 16 times.

The most frequent pair is **(“u”, “g”)**, so this is our first merge.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Tokenización a nivel subpalabra (BPE)

- Se repite el proceso de buscar la pareja de mayor frecuencia y actualizar el vocabulario y el corpus

Step 4: Repeat Merging Until Vocabulary Reaches Desired Size

BPE continues this process, repeating Steps 3 and 4 until the vocabulary reaches a specified size.

Next Iterations:

Iteration 2:

- Find the next most frequent pair: (“u”, “n”) with 16 occurrences.
- Merge (“u”, “n”) to form “un” and update the vocabulary and corpus:

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug", "un"] Corpus: ("h"
"ug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("h" "ug" "s", 5)
```

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist · Follow · 21 min read · Oct 30, 2024

Tokenización a nivel subpalabra (BPE)

- Se puede usar como límite que no se alcancen al menos dos palabras en el corpus

Iteration 3:

- The most frequent pair now is (“h”, “ug”), appearing 15 times.
- Merge (“h”, “ug”) to form “hug”:

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug", "un", "hug"] Corpus:
("hug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("hug" "s", 5)
```

This iterative process continues until the vocabulary reaches the desired size, capturing commonly co-occurring subwords that help the model efficiently represent words in the language.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Tokenización a nivel subpalabra (BPE)

- Se aplican las reglas de mezcla aprendidas en la secuencia del nuevo texto
- La palabra se parte en caracteres y se aplican las reglas de mezcla para formar subpalabras

Tokenization with BPE

During tokenization, BPE applies the learned merge rules in sequence to new text. The word is split into characters, and the merge rules are applied to form subwords.

For example, with the following learned merges:

```
("u", "g") -> "ug"
("u", "n") -> "un"
("h", "ug") -> "hug"
```

Tokenizing “bug”:

- Start with `["b", "u", "g"]`.
- Apply (“u”, “g”) → “ug”:

```
Tokenized Result: ["b", "ug"]
```

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Tokenización a nivel subpalabra (BPE)

- Algunas palabras podrían no estar dentro del vocabulario

Tokenizing “mug”:

- Start with `["m", "u", "g"]`.
- “m” is not in the learned vocabulary, so “mug” would become `[UNK]`.

Tokenizing “thug”:

- Start with `["t", "h", "u", "g"]`.
- “t” is not in the vocabulary, so “thug” would be `[UNK]` unless additional merges are learned.

BPE’s training process balances efficiency and coverage, providing a flexible vocabulary with common subwords that represent both frequent and rare words.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist · Follow · 21 min read · Oct 30, 2024

Tokenización a nivel subpalabra (BPE)

PRÁCTICA 2

- Aplicar el tokenizador BPE a un corpus
  - Aumentar el tamaño del vocabulario a 100

Why BPE Works

BPE is powerful because it captures frequently occurring word components and patterns, which helps in handling rare words by breaking them down into known subwords. For example, even if “lowest” is a rare word, BPE can split it into known tokens “low” and “est,” allowing the model to process it effectively. This makes BPE particularly useful for languages with rich morphology and for applications where a smaller, adaptable vocabulary is beneficial.

Example of BPE in python (Using tokenizers library):

```python
from tokenizers import CharBPETokenizer

# Sample corpus as a list of sentences
corpus = [
    "This is the Hugging Face Course.",
    "This chapter is about tokenization.",
    "This section shows several tokenizer algorithms.",
    "Hopefully, you will be able to understand how they are trained and generate
]

# Write corpus to a temporary text file for training
with open("corpus.txt", "w") as f:
    for line in corpus:
        f.write(f"{line}\n")

# Initialize the BPE tokenizer
tokenizer = CharBPETokenizer()

# Train the tokenizer on the corpus
tokenizer.train(files=["corpus.txt"], vocab_size=10, min_frequency=1)

# Save the tokenizer
tokenizer.save_model(".", "bpe_tokenizer")

# Test the tokenizer
encoded = tokenizer.encode("understand")
print("Encoded:", encoded.tokens)

# Decode tokens back to text
decoded = tokenizer.decode(encoded.ids)
print("Decoded:", decoded)
```

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Tokenización a nivel subpalabra (WordPiece)

- Cada palabra es recortada y se marcan los caracteres que no son el inicio de una plabra

WordPiece Algorithm

While Google has not open-sourced the exact algorithm for training WordPiece tokenizers, we can deduce a likely approach based on research. WordPiece shares similarities with BPE but introduces additional refinements to improve the selection of merge rules.

Key Steps in WordPiece Tokenization Training

1. **Start with a Minimal Vocabulary:**

- Begin with a small vocabulary containing the model's special tokens (like `[UNK]` and `[CLS]`), the initial alphabet (e.g., "a," "b," "c," etc.), and a prefix indicator (e.g., `##` for subwords that are not the start of a word).
- Each word is initially split into individual characters, with a prefix added to all characters except the first.
  For example, the word "word" would initially be tokenized as:

```
w ##o ##r ##d
```

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist · Follow · 21 min read · Oct 30, 2024

Tokenización a nivel subpalabra (WordPiece)

- Se calcula un puntaje para la pareja basada en la frecuencia de la pareja entre el producto de sus ocurrencias
- Se seleccionan y mezclan las parejas iterativamente

2. Calculate Pair Scores:

Like BPE, WordPiece learns which character pairs to merge iteratively, creating subword tokens based on frequently occurring character or subword pairs.

However, WordPiece does not simply merge the most frequent pairs. Instead, it calculates a score for each pair using a formula that accounts for both frequency and individual frequencies of each part:

$$\text{score} = \frac{\text{freq\_of\_pair}}{\text{freq\_of\_first\_element} \times \text{freq\_of\_second\_element}}$$

By dividing the frequency of a pair by the product of the frequencies of each part, the algorithm favors merging pairs where each part appears less frequently on its own. This discourages merging common subwords that might be useful independently.

Select and Merge Pairs Iteratively:

- The algorithm selects the pair with the highest score, merges it, and updates the vocabulary by adding the new merged subword.
- This process repeats until the vocabulary reaches the desired size.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Tokenización a nivel subpalabra (WordPiece)

- Se establece el vocabulario inicial

```
" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##g" "##s", 5)
```

Example Walkthrough

To illustrate, let's take a corpus with the following tokens and frequencies

```
("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
```

Step 1: Initialize the Vocabulary

Each word is initially split by characters with prefixes, creating a vocabulary such as:

```
Initial Vocabulary: ["b", "h", "p", "##g", "##n", "##s", "##u"]
```

The initial tokenized form of each word would look like this:

```
("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "
```

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Tokenización a nivel subpalabra (WordPiece)

- Se calcula el par con mayor puntuación

```
'##n", "##s", "##u", "##gs"]
 "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##gs", 5)
```

Step 2: Calculate Pair Scores

Now, WordPiece calculates the score for each pair based on frequency. Here's a sample calculation:

- The pair `("##u", "##g")` is the most frequent pair, but since "##u" appears frequently, its score isn't necessarily the highest.
- The pair with the highest score might instead be `("##g", "##s")`, which appears in "hugs" and has a lower individual frequency than `("##u", "##g")`.

So, `("##g", "##s")` is merged first, adding "##gs" to the vocabulary and updating the tokenized form of "hugs":

```
Updated Vocabulary: ["b", "h", "p", "##g", "##n", "##s", "##u", "##gs"]
Updated Corpus: ("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 1
```

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Tokenización a nivel subpalabra (WordPiece)

- A diferencia de BPE, el WordPiece no especifica reglas de mezclado, sino que crea un vocabulario final y palabras tokenizadas encontrando la subpalabra más larga posible que coincida dentro del vocabulario

Step 3: Continue Merging

The algorithm continues by selecting the next highest-scoring pair, perhaps `("h", "##u")`, and merges it to form "hu." After a few more steps, the word "hug" might emerge as a standalone token.

Tokenization Process with WordPiece

Unlike BPE, WordPiece doesn't save the specific merge rules. Instead, it creates a final vocabulary and tokenizes words by finding the longest possible subword match within the vocabulary.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Tokenización a nivel subpalabra (WordPiece)

- Las palabras que no se pueden calcular como la unión de subpalabras provocan un OOV

For example, let's tokenize "hugs" and "bugs" using the final vocabulary:

Tokenizing "hugs":

- The longest matching subword from the beginning is "hug," so the tokenizer splits here and adds `"##s"` for the remaining character.
- Tokenized result: `["hug", "##s"]`

Tokenizing "bugs":

- The longest match is `"b"`, so we split there, then `"##u"` and `"##gs"`.
- Tokenized result: `["b", "##u", "##gs"]`

If a word cannot be fully tokenized by subwords in the vocabulary, it will be replaced by `[UNK]`. For example, a word like "mug" might become `[UNK]` if "m," "##u," and "##g" do not form a complete word in the vocabulary.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist · Follow · 21 min read · Oct 30, 2024

Tokenización a nivel subpalabra (WordPiece)

PRÁCTICA 3

- Aplicar el tokenizador WordPiece a un corpus
  - Aumentar el tamaño del vocabulario a 100

Example of Wordpiece tokenizer in Python

```python
from tokenizers import BertWordPieceTokenizer

# Sample corpus as a list of sentences
corpus = [
    "This is the Hugging Face Course.",
    "This chapter is about tokenization.",
    "This section shows several tokenizer algorithms.",
    "Hopefully, you will be able to understand how they are trained and generate
]

# Write corpus to a temporary text file for training
with open("corpus.txt", "w") as f:
    for line in corpus:
        f.write(f"{line}\n")

# Initialize the BPE tokenizer
tokenizer = BertWordPieceTokenizer()

# Train the tokenizer on the corpus
tokenizer.train(files=["corpus.txt"], vocab_size=10, min_frequency=1)

# Save the tokenizer
tokenizer.save_model(".", "wordpiece_tokenizer")

# Test the tokenizer
encoded = tokenizer.encode("understand")
print("Encoded:", encoded.tokens)

# Decode tokens back to text
decoded = tokenizer.decode(encoded.ids)
print("Decoded:", decoded)
```

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

WordPiece vs. BPE

- **Frequency Calculation**: WordPiece calculates scores based on frequency ratios, emphasizing rare subword pairs, whereas BPE merges based purely on frequency.
- **Tokenization Process**: WordPiece uses the final vocabulary and performs longest subword matching, while BPE applies learned merge rules in sequence.
- **Unknown Words**: WordPiece tokenizes as `[UNK]` if a complete word cannot be tokenized, whereas BPE may partially tokenize unknown words.

WordPiece's probabilistic scoring and careful merging make it particularly effective in preserving linguistic meaning while keeping vocabulary size manageable.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Unigram Tokenization:

Unigram tokenization is a probabilistic approach that starts with a large vocabulary of subword tokens and iteratively removes the least useful tokens until reaching the desired vocabulary size. Unlike BPE and WordPiece, which build vocabularies by merging smaller units, Unigram goes in reverse, pruning tokens based on how much they impact a language model's likelihood over the corpus.

Here's how the Unigram algorithm trains a tokenizer, using a step-by-step example.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Step 1: Initialize the Vocabulary with Substrings

To start, Unigram builds an initial large vocabulary. The base vocabulary could be:

1. All substrings (or “subwords”) that appear in the corpus.
2. Results from running BPE with a large vocabulary size.

For example, if the corpus contains the following words with these frequencies:

```
("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
```

The initial vocabulary could be all strict substrings that appear in these words:

```
["h", "u", "g", "hu", "ug", "p", "pu", "n", "un", "b", "bu", "s", "hug", "gs", "
```

Step 2: Compute Probabilities of Each Token

The probability of each token is calculated based on its frequency in the corpus:

1. **Frequency of each token**: Count how often each token (subword) appears across all words.
2. **Probability**: Divide each token’s frequency by the sum of all frequencies to obtain its probability.

For instance:

```
("h", 15), ("u", 36), ("g", 20), ("hu", 15), ("ug", 20), ("p", 17), ("pu", 17),
("un", 16), ("b", 4), ("bu", 4), ("s", 5), ("hug", 15), ("gs", 5), ("ugs", 5)
```

The total frequency sum is 210, so the probability of each token, such as “ug”, is calculated as:

$$P(\text{“ug”}) = \frac{20}{210} \approx 0.0952$$

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Step 3: Calculate Tokenization Probabilities for Each Word

Unigram tokenization attempts to find the segmentation with the highest probability for each word by computing all possible segmentations and their probabilities. The probability of a segmentation is the product of the probabilities of each token in that segmentation.

For example, let's look at the word "pug":

1. Possible tokenizations of "pug":
   - `["p", "u", "g"]` with probability $P(\text{"p"}) \times P(\text{"u"}) \times P(\text{"g"})$.
   - `["p", "ug"]` with probability $P(\text{"p"}) \times P(\text{"ug"})$.
   - `["pu", "g"]` with probability $P(\text{"pu"}) \times P(\text{"g"})$.

The segmentation with the highest probability is chosen as the tokenization for the word.

Step 4: Calculate Loss

Unigram uses the negative log-likelihood of each word's probability to compute loss:

$$\text{Loss} = -\sum_{\text{word}} \text{freq}(\text{word}) \times \log(\text{P}(\text{word}))$$

For example, suppose the tokenization and score for each word is:

```
"hug": ["hug"] (score 0.071428)
"pug": ["pu", "g"] (score 0.007710)
"pun": ["pu", "n"] (score 0.006168)
"bun": ["bu", "n"] (score 0.001451)
"hugs": ["hug", "s"] (score 0.001701)
```

Then, the total loss would be calculated as follows:

Then, the total loss would be calculated as follows:

$$\text{Loss} = 10 \times (-\log(0.071428)) + 5 \times (-\log(0.007710)) + 12 \times (-\log(0.006168)) + 4 \times (-\log(0.001451)) + 5 \times (-\log(0.001701))$$

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Step 5: Remove Low-Impact Tokens

The algorithm evaluates how the loss would change if each token were removed:

1. Calculate the change in loss for each token.
2. Remove a percentage of tokens (e.g., 10% or 20%) with the lowest impact on the loss.

For instance:

- Removing the token “pu” might have a negligible impact since `["p", "u", "g"]` could replace `["pu", "g"]` without a significant loss increase.
- Removing “hug” would increase the loss substantially since `["hug"]` provides an efficient way to represent both "hug" and "hugs."

This pruning process continues until the vocabulary reaches the desired size.

Step 6: Tokenization with Unigram

Once the vocabulary is trained, tokenizing a word involves choosing the segmentation with the highest probability. This can be efficiently done using the **Viterbi algorithm**, which finds the best tokenization path by iterating through the word and selecting subwords with the best scores.

For example, the word “unhug” could be tokenized as:

- Character 0 (u): “u” (score 0.171429)
- Character 1 (n): “un” (score 0.076191)
- Character 2 (h): “un” “h” (score 0.005442)
- Character 3 (u): “un” “hu” (score 0.005442)
- Character 4 (g): “un” “hug” (score 0.005442)

Thus, “unhug” would be tokenized as `["un", "hug"]`.

Unigram tokenization is unique because:

- It starts with a large vocabulary and removes tokens iteratively.
- Tokens are chosen for removal based on their impact on loss.
- The Viterbi algorithm is used to find the best tokenization, ensuring a high-probability, minimal-length segmentation for each word.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist · Follow · 21 min read · Oct 30, 2024

Tokenización a nivel subpalabra (Unigram)

PRÁCTICA 4

- Aplicar el tokenizador Unigram a un corpus
  - Aumentar el tamaño del vocabulario a 13

```
Word: hug -> Tokens: ['_', 'h', 'u', 'g']
Word: pug -> Tokens: ['_', 'p', 'u', 'g']
Word: pun -> Tokens: ['_', 'p', 'u', 'n']
Word: hugs -> Tokens: ['_', 'h', 'u', 'g', 's']
Word: bun -> Tokens: ['_', 'b', 'u', 'n']
Tokens: [5, 9, 7, 8] -> Decoded word: hug
Tokens: [5, 4, 7, 8] -> Decoded word: pug
Tokens: [5, 4, 7, 3] -> Decoded word: pun
Tokens: [5, 9, 7, 8, 10] -> Decoded word: hugs
Tokens: [5, 6, 7, 3] -> Decoded word: bun
```

Example in Python

```python
import sentencepiece as spm

# Create a sample corpus file
corpus = ["hug", "pug", "pun", "bun", "hugs"]
with open("corpus.txt", "w") as f:
    for word in corpus:
        f.write(f"{word}\n")

# Step 1: Train a Unigram tokenizer
spm.SentencePieceTrainer.Train(
    input="corpus.txt",
    model_prefix="unigram_tokenizer",
    vocab_size=11,  # Adjust based on desired vocabulary size
    model_type="unigram"
)

# Step 2: Load the trained Unigram model
sp = spm.SentencePieceProcessor(model_file="unigram_tokenizer.model")

# Step 3: Tokenize text with the Unigram model
words = ["hug", "pug", "pun", "hugs", "bun"]
for word in words:
    tokens = sp.encode(word, out_type=str)
    print(f"Word: {word} -> Tokens: {tokens}")

# Step 4: Decode tokens back to text
for word in words:
    token_ids = sp.encode(word)
    decoded_word = sp.decode(token_ids)
    print(f"Tokens: {token_ids} -> Decoded word: {decoded_word}")
```

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

How to Choose the right tokenization strategy

Choosing the right tokenization strategy depends on several factors, such as the language characteristics, the complexity of the application, and the model requirements. Here's a guide to help you decide which tokenization method might best suit your needs:

1. Word-Level Tokenization

Best For:

- Simpler applications or languages with limited morphological complexity, such as English.
- When computational efficiency is a priority, as word-level tokenization produces fewer tokens.

Pros:

- Straightforward and fast to implement.
- Preserves word-level semantics effectively for common words.

Cons:

- Struggles with out-of-vocabulary (OOV) or rare words.
- Inefficient for languages with complex morphology, as it doesn't handle inflections, prefixes, and suffixes well.

Use Cases:

- Basic NLP applications like keyword extraction.
- Models with a relatively fixed and simple vocabulary.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

How to Choose the right tokenization strategy

Choosing the right tokenization strategy depends on several factors, such as the language characteristics, the complexity of the application, and the model requirements. Here's a guide to help you decide which tokenization method might best suit your needs:

2. Character-Level Tokenization

Best For:

- Languages with rich morphology (e.g., Finnish, Turkish) or with many compound words.
- Applications where handling out-of-vocabulary words is essential.

Pros:

- Eliminates out-of-vocabulary problems since all words are decomposed into characters.
- Ideal for fine-grained control over morphology, making it adaptable to complex linguistic structures.

Cons:

- Generates very long sequences, which can be computationally expensive.
- Sacrifices word-level semantics, making it harder for models to capture the context effectively.

Use Cases:

- Text-to-speech models.
- Applications involving non-standard text, such as social media posts with many abbreviations or creative spellings.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist · Follow · 21 min read · Oct 30, 2024

How to Choose the right tokenization strategy

Choosing the right tokenization strategy depends on several factors, such as the language characteristics, the complexity of the application, and the model requirements. Here's a guide to help you decide which tokenization method might best suit your needs:

3. Subword-Level Tokenization (e.g., Byte Pair Encoding, WordPiece)

Best For:

- Balancing vocabulary size with flexibility in handling rare words.
- Applications requiring efficient memory usage and contextual language understanding.

Pros:

- Reduces the vocabulary size while maintaining coverage for rare or compound words.
- Handles out-of-vocabulary words by breaking them into meaningful subwords.

Cons:

- Slightly more complex than word-level tokenization.
- Requires training a subword model on a representative corpus for optimal results.

Use Cases:

- Transformer-based models like BERT and GPT.
- Multilingual applications, as subwords work well across different languages and reduce vocabulary overlap.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

How to Choose the right tokenization strategy

Choosing the right tokenization strategy depends on several factors, such as the language characteristics, the complexity of the application, and the model requirements. Here's a guide to help you decide which tokenization method might best suit your needs:

4. Unigram Tokenization

Best For:

- Applications where the highest probability segmentation is critical, like in probabilistic language modeling.
- Situations where you want the vocabulary to be learned based on minimizing the number of tokens while maximizing coverage.

Pros:

- Builds a vocabulary by retaining the most impactful subwords, resulting in an efficient tokenization scheme.
- Allows flexibility in tokenizing new or rare words with fewer subwords.

Cons:

- Requires extensive computation for selecting and pruning tokens during training.
- May need sophisticated implementation (like the Viterbi algorithm) for efficient decoding.

Use Cases:

- NLP applications with probabilistic language models (e.g., Google's T5).
- When working with diverse data and looking to maintain efficient token usage.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

How to Choose the right tokenization strategy

Choosing the right tokenization strategy depends on several factors, such as the language characteristics, the complexity of the application, and the model requirements. Here's a guide to help you decide which tokenization method might best suit your needs:

5. Byte-Level BPE

Best For:

- Applications that require complete coverage of input characters, such as handling emojis, special characters, and multilingual text.
- Models that need to handle Unicode characters and byte sequences robustly.

Pros:

- Ensures compatibility with any character, including emojis and special symbols, without creating out-of-vocabulary issues.
- Reduces complexity by encoding everything as bytes, which is ideal for certain models like GPT-2.

Cons:

- Less human-readable than word or subword tokenization.
- May not capture linguistic subwords as effectively, potentially reducing interpretability.

Use Cases:

- Multilingual NLP models where character encoding consistency is essential.
- Large language models that need to process varied and non-standard inputs, like social media or informal text.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

How to Choose the right tokenization strategy

Choosing the right tokenization strategy depends on several factors, such as the language characteristics, the complexity of the application, and the model requirements. Here's a guide to help you decide which tokenization method might best suit your needs:

Summary of Considerations

- **Language Characteristics**: For morphologically rich languages, consider subword or character-level tokenization. For simpler languages, word-level tokenization might suffice.
- **Application Complexity**: For applications that need strong semantic understanding, subword tokenization provides a balance between vocabulary size and language complexity.
- **Model Type**: Transformer models like BERT and GPT often use subword tokenization (e.g., BPE or WordPiece), while character-level tokenization suits RNNs and models for text-to-speech.
- **Memory and Computational Constraints**: Character-level tokenization can be memory-intensive, while subword methods reduce vocabulary size and improve efficiency without losing flexibility.

Choosing the right tokenization strategy involves balancing efficiency with flexibility based on your specific needs, language requirements, and model goals.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Common Challenges and Solutions in Tokenization

Tokenization can introduce several challenges that impact the efficiency and accuracy of NLP models. Here's a breakdown of these challenges and some solutions to address them.

1. Out-of-Vocabulary (OOV) Words

Challenge: Out-of-vocabulary words are terms that are not present in the model's vocabulary. They can lead to poor model performance, especially in domain-specific texts with unique terminologies.

Solutions:

- **Subword Tokenization:** Techniques like Byte Pair Encoding (BPE) or WordPiece break down unknown words into smaller, meaningful subwords or characters, enabling the model to handle rare or new words by using familiar sub-parts.
- **Unigram Tokenization:** Unigram models handle OOV words by assigning probabilities to various segmentations, thus dynamically selecting subword segments from the vocabulary for unknown terms.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Common Challenges and Solutions in Tokenization

Tokenization can introduce several challenges that impact the efficiency and accuracy of NLP models. Here's a breakdown of these challenges and some solutions to address them.

2. Handling Special Characters and Emojis

Challenge: Special characters, symbols, and emojis are common in social media and informal text. Many tokenizers may ignore these characters, leading to information loss.

Solutions:

- **Byte-Level BPE**: Models like GPT-2 use byte-level BPE to encode every character as a byte, allowing them to handle all symbols, emojis, and characters without OOV issues.
- **Custom Tokenizer Training**: If working with a highly specialized domain (e.g., medical text, chemistry formulas), training a custom tokenizer with a vocabulary that includes relevant symbols ensures that essential information is retained.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Common Challenges and Solutions in Tokenization

Tokenization can introduce several challenges that impact the efficiency and accuracy of NLP models. Here's a breakdown of these challenges and some solutions to address them.

3. Tokenization Granularity

Challenge: Selecting the appropriate granularity (word, subword, or character) can be challenging. Too fine-grained (character-level) tokenization increases sequence length, which leads to computational inefficiency. Too coarse (word-level) may fail to capture parts of infrequent or compound words.

Solutions:

- **Hybrid Approaches**: Combining subword and character-level tokenization can provide flexibility in handling rare or complex words while keeping sequence lengths manageable.
- **Dynamic Tokenization**: In some cases, dynamically switching between subword and character tokenization (based on word frequency) may improve both model performance and efficiency.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Common Challenges and Solutions in Tokenization

Tokenization can introduce several challenges that impact the efficiency and accuracy of NLP models. Here's a breakdown of these challenges and some solutions to address them.

4. Ambiguity in Tokenization

Challenge: Words with similar spellings but different meanings (homographs) or words used in multiple contexts can be difficult to tokenize accurately, impacting the model's understanding of the context.

Solutions:

- **Contextual Embeddings**: Using contextual embeddings from models like BERT can help the model disambiguate meanings based on surrounding words.
- **Disambiguation with Additional Features**: When building a tokenizer for a domain-specific model, adding domain-specific subwords or collocations can help disambiguate words that have multiple meanings.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Common Challenges and Solutions in Tokenization

Tokenization can introduce several challenges that impact the efficiency and accuracy of NLP models. Here's a breakdown of these challenges and some solutions to address them.

5. Loss of Linguistic Information

Challenge: Tokenization can cause a loss of linguistic structure, such as prefixes, suffixes, and morphological changes, which are crucial in languages with complex morphology (e.g., Turkish, Finnish).

Solutions:

- **Morphology-Aware Tokenization**: For morphologically rich languages, tokenizers that consider linguistic units (e.g., morphemes or root words) can improve performance.
- **Data Augmentation**: Training the tokenizer on a corpus with morphological variations can help it capture the diverse forms a word may take, ensuring more robust handling of morphology.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Common Challenges and Solutions in Tokenization

Tokenization can introduce several challenges that impact the efficiency and accuracy of NLP models. Here's a breakdown of these challenges and some solutions to address them.

6. Length of Tokenized Sequences

Challenge: Long token sequences are a common issue in character-level or fine-grained subword tokenization, which can cause computational inefficiency and memory issues.

Solutions:

- **Subword Tokenization**: Using BPE or WordPiece reduces the number of tokens by breaking down only rare or compound words, balancing sequence length and vocabulary coverage.
- **Max Sequence Length Adjustment**: For transformer models, adjusting the maximum sequence length parameter to fit your hardware resources can help manage memory without significantly impacting accuracy.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Common Challenges and Solutions in Tokenization

Tokenization can introduce several challenges that impact the efficiency and accuracy of NLP models. Here's a breakdown of these challenges and some solutions to address them.

7. Inconsistent Tokenization Across Languages

Challenge: Multilingual models require tokenizers that work well across various languages. However, tokenizing one language effectively may not translate to another due to differences in morphology, syntax, and character sets.

Solutions:

- **Multilingual Tokenization Models**: Use tokenizers that support multiple languages, like the multilingual BERT or XLM-R, which rely on subword tokenization designed to capture a wide range of linguistic structures.
- **Language-Specific Vocabularies**: For highly multilingual applications, training separate tokenizers with language-specific vocabularies or using a hybrid tokenizer can improve tokenization consistency and accuracy.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Common Challenges and Solutions in Tokenization

Tokenization can introduce several challenges that impact the efficiency and accuracy of NLP models. Here's a breakdown of these challenges and some solutions to address them.

8. Difficulty in Tokenizing Compound Words and Names

Challenge: Languages like German or compound words in technical domains may present difficulties as they combine words into long tokens. Names, abbreviations, or non-standard terms are also challenging to tokenize correctly.

Solutions:

- **Subword Tokenization with Large Vocabulary**: Using a larger vocabulary in BPE or WordPiece can help capture compound words and rare terms as single tokens, reducing the impact on accuracy.
- **Tokenization with Domain-Specific Data**: Training the tokenizer on a corpus rich with compound words or domain-specific terms improves its ability to handle such vocabulary.

Decoding Tokenization Strategies for Large Language Models (LLMs)

Sahin Ahmed, Data Scientist Follow 21 min read · Oct 30, 2024

Conclusion

Tokenization is a critical step in NLP that fundamentally shapes how models understand and generate text. By converting raw text into structured, manageable units, tokenization directly impacts model efficiency, memory use, and performance across various tasks. As we've explored, each tokenization strategy — whether word-level, character-level, subword-based (like BPE and WordPiece), or probabilistic (like Unigram) — offers unique strengths and trade-offs, making it essential to select the most suitable approach based on the language, application, and computational resources.

Alan Badillo Salas

badillosalas@outlook.com

https://github.com/dragonnomada/ipn-cic-diplomado-ia-2025