Embeddings

Positional Embedding

Function Signature

`def embed_position(sequence: np.ndarray) -> np.ndarray:`

Parameters

- **sequence (np.ndarray)**: A 2D numpy array where the first dimension represents the sequence length and the second represents the embedding dimension.

Return Value

Returns a 2D numpy array with positional embeddings applied to the input sequence.

Description

The `embed_position` function applies positional embedding to a given sequence. The positional embedding is computed using a Cython implementation, which is expected to be faster than a pure Python implementation. The purpose of this embedding is to provide the model with information about the position of elements in the sequence, which can be crucial for certain tasks such as sequence-to-sequence modeling.

Examples

Here's a basic example to demonstrate how to use the `embed_position` function:

```
import numpy as np
from deeprai.embedding.positional_embedding import embed_position

# Create a mock sequence
sequence = np.array([[0.1, 0.2], [0.3, 0.4], [0.5, 0.6]])

# Apply positional embedding
embedded_sequence = embed_position(sequence)
```

```
print(embedded_sequence)
```

Word Vecrotization

Module Import:

`from deeprai.embedding import word_vectorize`

Class Definition:

`class WordVectorizer:`

Initialization:

The WordVectorizer is initialized with an optional corpus, which is used for TF-IDF computations.

`def __init__(self, corpus=None):`

Parameters:

- **corpus (list of str, optional)**: List of words that forms the basis for the term frequency-inverse document frequency (TF-IDF) calculations.

Methods:

1. One-Hot Vectorization:

Converts a given word into a one-hot encoded matrix.

`def one_hot_vectorize(self, word) -> np.ndarray:`

Parameters:

- **word (str)**: The word to vectorize.

Returns:

- **numpy.ndarray**: One-hot encoded matrix representation of the word.

2. Continuous Vectorization:

Encodes a given word into continuous values for each character.

`def continuous_vectorize(self, word) -> np.ndarray:`

Parameters:

- **word (str)**: The word to vectorize.

Returns:

- **numpy.ndarray**: Continuous valued representation of the word.

3. Binary Vectorization:

Converts each character of a word into its binary ASCII representation.

`def binary_vectorize(self, word) -> np.ndarray:`

Parameters:

- **word (str)**: The word to vectorize.

Returns:

- **numpy.ndarray**: Binary ASCII representation of the word.

4. Frequency Vectorization:

Encodes the word based on the frequency of each letter normalized by word length.

`def frequency_vectorize(self, word) -> np.ndarray:`

Parameters:

- **word (str)**: The word to vectorize.

Returns:

- **numpy.ndarray**: Frequency-based representation of the word.

5. N-gram Vectorization:

Vectorizes the word by creating n-grams.

`def ngram_vectorize(self, word, n=2) -> np.ndarray:`

Parameters:

- **word (str)**: The word to vectorize.
- **n (int, default=2)**: The size of the n-grams.

Returns:

- **numpy.ndarray**: N-gram based vector representation of the word.

6. TF-IDF Vectorization:

Vectorizes a word based on term frequency-inverse document frequency.

`def tfidf_vectorize(self, word) -> np.ndarray:`

Parameters:

- **word (str)**: The word to vectorize.

Returns:

- **numpy.ndarray**: TF-IDF representation of the word.

Raises:

- **ValueError**: If the WordVectorizer is not initialized with a corpus.

Description:

The `WordVectorizer` class from the `deeprai.embedding.word_vectorize` module provides multiple ways to represent words as vectors. These include methods like one-hot encoding, continuous encoding, binary encoding, frequency-based encoding, n-gram-based encoding, and TF-IDF encoding. The TF-IDF method requires a corpus to be passed during the initialization of the class.

Examples:

Module Import and Initialization:

First, let's import the necessary module and initialize our WordVectorizer. For methods that require a corpus (like TF-IDF), we'll provide a sample corpus.

```
from deeprai.embedding import word_vectorize


corpus = ["apple", "banana", "cherry", "date", "fig", "grape"]
vectorizer = word_vectorize.WordVectorizer(corpus=corpus)
```

1. One-Hot Vectorization:

This method will transform a word into a matrix where each row is a one-hot encoded representation of a character in the word.

```
word = "apple"
one_hot_encoded = vectorizer.one_hot_vectorize(word)
print(one_hot_encoded)
```

2. Continuous Vectorization:

This method will transform a word into a vector of continuous values.

```
word = "apple"
continuous_vector = vectorizer.continuous_vectorize(word)
print(continuous_vector)
```

3. Binary Vectorization:

This will convert each character of the word into its 8-bit ASCII representation.

```
word = "apple"
binary_vector = vectorizer.binary_vectorize(word)
print(binary_vector)
```

4. Frequency Vectorization:

This method vectorizes a word based on the normalized frequency of each letter in it.

```
word = "apple"
frequency_vector = vectorizer.frequency_vectorize(word)
print(frequency_vector)
```

5. N-gram Vectorization:

This method will break the word into n-grams and vectorize them. For this example, we'll use n=2 (bigrams).

```
word = "apple"
bigram_vector = vectorizer.ngram_vectorize(word, n=2)
print(bigram_vector)
```

6. TF-IDF Vectorization:

This method requires a corpus to compute the inverse document frequency. It will then vectorize a word based on its term frequency and the inverse document frequency from the corpus.

```
word = "apple"
tfidf_vector = vectorizer.tfidf_vectorize(word)
print(tfidf_vector)
```