NATIONAL UNIVERSITY OF MODERN LANGUAGES ISLAMABAD

Natural Language Processing (Assignments)

Assignment: 03

Submitted to
Miss Qurat-ul-ain safder

Submitted By
Adeel Naeem
(BSAI-146)

Submission Date: May 27th ,2025

Word Embeddings :
Word Embeddings are numeric representations of words in a lower-dimensional space, capturing semantic and syntactic information. They play a vital role in **Natural Language Processing (NLP) tasks**. This article explores traditional and neural approaches, such as TF-IDF, Word2Vec, and GloVe, offering insights into their advantages and disadvantages. Understanding the importance of pre-trained word embeddings, providing a comprehensive understanding of their applications in various NLP scenarios.

Need for Word Embedding?
- To reduce dimensionality
- To use a word to predict the words around it.
- Inter-word semantics must be captured.

Pre-trained Word Embedding

Pretrained word embeddings are vector representations of words that are learned from large text corpora and capture the semantic and syntactic relationships between words. They transform words into dense numerical vectors, where similar words have similar representations. Three popular types of pretrained word embeddings are Word2Vec, GloVe, and FastText. Word2Vec, developed by Google, uses neural networks with architectures like CBOW and Skip-gram to learn word associations based on context. GloVe, from Stanford, constructs a word co-occurrence matrix and uses matrix factorization to capture global statistical information about word usage. FastText, developed by Facebook, improves upon Word2Vec by incorporating subword (character n-gram) information, enabling it to handle out-of-vocabulary words and perform better on morphologically rich languages. These embeddings are widely used in natural language processing tasks to enhance the understanding of text data.

1.GloVe (Global Vectors for Word Representation) is an embedding method developed by Stanford. It constructs a word co-occurrence matrix from the corpus and uses matrix factorization techniques to learn word vectors. GloVe focuses on capturing the global statistical information of words by analyzing how frequently word pairs appear together in a text. This allows it to effectively represent both semantic and syntactic relationships between words.

Code:

```
from gensim.models import KeyedVectors

from gensim.downloader import load

glove_model = load('glove-wiki-gigaword-50')

word_pairs = [('learn', 'learning'), ('pakistan', 'pakistani'), ('fame', 'famous')]
```

# Compute similarity for each pair of words

for pair in word_pairs:

    similarity = glove_model.similarity(pair[0], pair[1])

    print(f"Similarity between '{pair[0]}' and '{pair[1]}' using GloVe: {similarity:.3f}")

Output:

```
Similarity between 'learn' and 'learning' using GloVe: 0.802
Similarity between 'pakistan' and 'pakistani' using GloVe: 0.865
Similarity between 'fame' and 'famous' using GloVe: 0.589
```

2.FastText is a word embedding model developed by Facebook. Unlike Word2Vec and GloVe, FastText considers subword information by breaking words into character n-grams. This approach helps the model understand the internal structure of words, making it particularly useful for handling rare or out-of-vocabulary (OOV) words. FastText is especially effective for languages with rich morphology, as it can generate embeddings for words not seen during training.

Code:

import gensim.downloader as api

fasttext_model = api.load("fasttext-wiki-news-subwords-300") ## Load the pre-trained fastText model

# Define word pairs to compute similarity for

word_pairs = [('learn', 'learning'), ('pakistan', 'pakistani'), ('fame', 'famous')]

# Compute similarity for each pair of words

for pair in word_pairs:

    similarity = fasttext_model.similarity(pair[0], pair[1])

    print(f"Similarity between '{pair[0]}' and '{pair[1]}' using FastText: {similarity:.3f}")

Output:

```
Similarity between 'learn' and 'learning' using FastText: 0.642
Similarity between 'pakistan' and 'pakistani' using FastText: 0.708
Similarity between 'fame' and 'famous' using FastText: 0.519
```

3.BERT (Bidirectional Encoder Representations from Transformers):
BERT is a transformer-based model that learns contextualized embeddings for words. It considers the entire context of a word by considering both left and right contexts, resulting in embeddings that capture rich contextual information.

Code:

```
from transformers import BertTokenizer, BertModel

import torch

# Load pre-trained BERT model and tokenizer

model_name = 'bert-base-uncased'

tokenizer = BertTokenizer.from_pretrained(model_name)

model = BertModel.from_pretrained(model_name)



word_pairs = [('learn', 'learning'), ('pakistan', 'pakistani'), ('fame', 'famous')]

# Compute similarity for each pair of words

for pair in word_pairs:

    tokens = tokenizer(pair, return_tensors='pt')

    with torch.no_grad():

        outputs = model(**tokens)

    # Extract embeddings for the [CLS] token

    cls_embedding = outputs.last_hidden_state[:, 0, :]



    similarity = torch.nn.functional.cosine_similarity(cls_embedding[0], cls_embedding[1],
dim=0)
```

```
    print(f"Similarity between '{pair[0]}' and '{pair[1]}' using BERT: {similarity:.3f}")
```

Output:

```
Similarity between 'learn' and 'learning' using BERT: 0.930
Similarity between 'pakistan' and 'pakistani' using BERT: 0.957
Similarity between 'fame' and 'famous' using BERT: 0.956
```