PRACTICAL ASSESMENT

HAREESHMA.S

ACTIVITY 1

Python Implementation of Word Embeddings using word2vec

Requirements

- Personal computer/laptop
- Google Collab

Procedure

1. Import Necessary Libraries

```
[1] # Import necessary Libraries
    from gensim.models import Word2Vec
    from nltk.tokenize import word_tokenize
    from nltk import download
```

2. Download Required NLTK Data

```
download("punkt")
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

- # Define Example Sentences

```
[2] # Example sentences
sentences = [
    "Natural Language Processing is fun.",
    "Language models are improving every day."
]
```

- # Tokenize Sentences

```
# Tokenize sentences
tokenized_sentences = [word_tokenize(sentence.lower()) for sentence in sentences]
tokenized_sentences
```

```
[['natural', 'language', 'processing', 'is', 'fun', '.'],
 ['language', 'models', 'are', 'improving', 'every', 'day', '.']]
```

- # Train the Word2Vec Model

```
# Train the Word2Vec model
model = Word2Vec(sentences=tokenized_sentences, vector_size=5, window=5, min_count=1, workers=4, sg=0)
# Here sg=0 means the model will use Continuous bag of words architecture and if sg=1 then it will use Skip-gram Model

# Get word vectors
word_vectors = model.wv
print("Word Vector for 'language':", word_vectors['language'])
```

```
Word Vector for 'language': [-0.14233617  0.12917745  0.17945977 -0.10030856 -0.07526743]
```

ACTIVITY 2

Python Implementation of Word Embeddings using GloVe

Requirements

- Personal computer/laptop
- Google Collab

Procedure

- Import the Gensim downloader

```
[2] import gensim.downloader as api
```

- Load the pre-trained GloVe model with 50 dimensions and Check the dimensions of a sample word vector (e.g., 'language')

```
[3] # Load the pre-trained GloVe model with 50 dimensions
glove_vectors_50d = api.load("glove-wiki-gigaword-50")
print("Dimensions of 50d GloVe vector:", len(glove_vectors_50d['language']))

[==================================================] 100.0% 66.0/66.0MB downloaded
Dimensions of 50d GloVe vector: 50
```

- Load the pre-trained GloVe model with 100 dimensions
- Check the dimensions of a sample word vector (e.g., 'language')

```
[ ] # Load the pre-trained GloVe model with 100 dimensions
glove_vectors_100d = api.load("glove-wiki-gigaword-100")
print("Dimensions of 100d GloVe vector:", len(glove_vectors_100d['language']))
```

- Load the pre-trained GloVe model with 200 dimensions
- Check the dimensions of a sample word vector

```
[ ] # Load the pre-trained GloVe model with 200 dimensions
glove_vectors_200d = api.load("glove-wiki-gigaword-200")
print("Dimensions of 200d GloVe vector:", len(glove_vectors_200d['language']))
```

- Load the pre-trained GloVe model with 300 dimensions
- Check the dimensions of a sample word vector

```
[ ] # Load the pre-trained GloVe model with 300 dimensions
glove_vectors_300d = api.load("glove-wiki-gigaword-300")
print("Dimensions of 300d GloVe vector:", len(glove_vectors_300d['language']))
```

ACTIVITY 3

Python Implementation of Word Embeddings using Fasttext

Requirements

- Personal computer/laptop
- Google Collab

Procedure

- Importing Necessary Libraries

```
[1] # Import necessary libarries
    from gensim.models import FastText
    from nltk.tokenize import word_tokenize
    from nltk import download
```

Downloading NLTK Data

```
# Download required NLTK data
download('punkt')
```

Example Sentences

```
[ ]
    # Example sentences
    sentences = [
        "Natural Language Processing is fun.",
        "Language models are improving every day."
    ]
```

Tokenizing Sentences

```
# Tokenize sentences
tokenized_sentences = [word_tokenize(sentence.lower()) for sentence in sentences]
```

Training the FastText Model

```
# Train the FastText model
model = FastText(sentences=tokenized_sentences, vector_size=5, window=5, min_count=1, workers=4, sg=1)

# Get word vectors
word_vectors = model.wv
print("Word Vector for 'language':", word_vectors['language'])

# Get vector for an OOV word
print("Word Vector for 'NLPfun':", word_vectors['nlpfun'])
```

```
Word Vector for 'language': [-0.00461428  0.01921903 -0.00035116 -0.00750383 -0.02619313]
Word Vector for 'NLPfun': [ 0.01152632  0.00589536 -0.01608402 -0.00613909  0.00409522]
```

output the word vectors for 'language' and 'NLPfun'. The FastText model's ability to handle OOV words through subword information is one of its key strengths, making it robust for a variety of natural language processing tasks.