Mini-Project 1: Implement Skip-Gram Model

Overview

Implement the Skip-Gram model to learn word embeddings from a subset of the Text8 corpus. Detailed instructions are provided from the second page.
You should do this mini-project in groups of 2-3 (same as assignments). When submitting, you should include the names of all the group members.

Dataset

You will use the Text8 corpus.

Tasks Overview

1. Download and preprocess the training dataset.
2. Implement the Skip-Gram model with negative sampling in PyTorch.
3. Evaluate your embeddings on the WordSim-353 test data using Spearman correlation .
4. Visualize the embeddings using t-SNE or PCA (optional).
5. Write a report.

Deliverables

1. Code and instructions on how to run it. Include any dependencies or libraries you used.
2. A PDF report of your implementation and evaluation.
3. Submit a single zip file.

Detailed Instructions

1. Data Preprocessing

Download the Dataset

- Use the provided `download_corpus.py` script to download the Text8 corpus.
- Text8 corpus is a large text compression benchmark. It consists of the first 100 million characters of the English Wikipedia dump from March 3, 2006.
- You will only work with the first 20 million characters.
- Additionally, you can find the corpus here: `http://mattmahoney.net/dc/text8.zip`

Text Preprocessing

- **Sentence Splitting and Tokenization**:
  - Split the text into words using whitespace as the delimiter.
  - **Important**: No need to perform any additional preprocessing steps.

Building the Vocabulary

1. **Word Frequency Count**: Count the frequency of each word in the dataset.
2. **Vocabulary Creation**:
   - Keep the top 60,000 most frequent words.
   - Replace less frequent words with a special `<UNK>` (unknown) token.
3. **Indexing Words**:
   - Assign a unique index to each word in your vocabulary, including the `<UNK>` token.
   - Convert the text data into a sequence of word indices.

Generating Training Data for the Skip-Gram Model

1. **Context Window Size (C)**:
   - Choose a context window size, e.g., $C = 2$.
2. **Creating Skip-Gram Pairs**:

- For each word (the "center word") in your indexed dataset, create training pairs with each of its context words within the window size C on both sides.
- For example, with a window size of 2, the context words for the center word at position i are words at positions $i-2$, $i-1$, $i+1$, $i+2$ (if they exist).

Preparing for Negative Sampling

1. **Compute the Unigram Distribution**:
    - The unigram distribution is a probability distribution over words based on their relative frequencies.
    - For each word w in your vocabulary, calculate its probability $U(w)$:

    $$U(w) = \frac{freq(w)}{N}$$

    where $freq(w)$ is the frequency of word w, and N is the total count of all word frequencies.

2. **Smoothed Unigram Distribution**:
    - Adjust the unigram distribution by raising each probability to the $\frac{3}{4}$ power to smooth it:

    $$U_{\text{smoothed}}(w) = \frac{U(w)^{\alpha}}{\sum_{w' \in V} U(w)^{\alpha}}$$

    where $\alpha = \frac{3}{4}$ and V is your vocabulary.
    - This smoothed distribution will be used to sample negative words during training, giving less frequent words a higher chance compared to the original unigram distribution.

2. Implement the Skip-Gram Model

Model Architecture

- Implement the Skip-Gram model using PyTorch.
- The model should include:
    - **Embedding Layer**: Maps input word indices to dense vectors (embeddings).
    - **Output Mechanism**: Computes similarity scores between the center word embedding and context/negative word embeddings.

Loss Function

- Use the **Negative Sampling Loss**:
  - For each center word and a context word pair, maximize the probability of the context word given the center word.
  - Simultaneously, for negative samples, minimize the probability that they are context words of the center word.

Batch Processing

- Use a `DataLoader` to handle batching of your data.

Training Loop

For each batch during training:

1. **Extract Center and Context Words**:
   - Prepare tensors for center words and their corresponding context words.
2. **Sample Negative Words**:
   - For each center word, sample negative words from the smoothed unigram distribution $U_{\text{smoothed}}(w)$.
3. **Compute the Loss**:
   - Calculate the loss using your negative sampling loss function.
4. **Update Model Parameters**:
   - Use an optimizer like `torch.optim.Adam` to update the model parameters based on the computed loss.

3. Training the Model

- **Training Setup**:
  - Choose appropriate hyperparameters such as embedding size, learning rate, batch size, number of epochs, and number of negative samples.
- **Tip**:
  - Start with a small subset of data to ensure your implementation works before scaling up.

4. Evaluation

a. Word Similarity Task Using WordSim-353

Dataset Description

- **WordSim-353** is a dataset consisting of 353 pairs of English words.
- Each pair has a human-assigned similarity score ranging from 0 to 10, indicating how similar the words are in meaning.
- Download wordsim-353 from `https://gabrilovich.com/resources/data/wordsim353/wordsim353.zip`.

Evaluation Steps

1. **Compute Word Embedding Similarities**:
   - For each word pair (w_1, w_2) in WordSim-353:
     - Retrieve the embeddings $\mathbf{v}_{w_1}$ and $\mathbf{v}_{w_2}$.
     - Compute the **cosine similarity** between the two embeddings:

$$\text{Cosine Similarity} = \frac{\mathbf{v}_{w_1} \cdot \mathbf{v}_{w_2}}{\|\mathbf{v}_{w_1}\| \|\mathbf{v}_{w_2}\|}$$

     - **Note**: If a word is not in your vocabulary, skip that pair.
2. **Calculate Spearman's Rank Correlation**:
   - Compare your computed similarities with the human-assigned scores using Spearman's rank correlation coefficient.
   - This measures how well your model's similarity scores correlate with human judgments.
3. **Report the Correlation**:
   - Report the Spearman correlation coefficient in your report.

b. Visualization (optional)

1. **Select Words to Visualize**:
   - Use the following list of words:

```
king, queen, prince, princess, aunt, uncle, daughter, son,
paris, france, london, england,
apple, potato, mango, fruit,
lion, wolf, tiger, elephant,
car, truck, vehicle, bus,
neptune, saturn, pluto, earth
```

2. **Dimensionality Reduction**:
   - Use **t-SNE** or **PCA** to reduce the dimensionality of your embeddings to 2D.
3. **Plotting**:
   - Create a scatter plot of the reduced embeddings.
   - Label each point with the corresponding word.
4. **Report the Plot**:
   - In the report, include the plot and discuss any patterns you find.

Resources

- *Distributed Representations of Words and Phrases and their Compositionality* by Mikolov et al. (`https://papers.nips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf`)
- *Efficient Estimation of Word Representations in Vector Space* by Mikolov et al. (`https://arxiv.org/pdf/1301.3781`)
- *Placing Search in Context: The Concept Revisited* by Finkelstein et al. (`https://aclweb.org/aclwiki/WordSimilarity-353_Test_Collection_(State_of_the_art)`)