NATIONAL UNIVERSITY
of Computer & Emerging Sciences

REPORT

Skip-gram model
(preprocessing, implementation, training, and visualization)

Saad Ahmed
22K 4345
BCS 6A
DLP ASSIGNMENT 3

Skip-gram model - Assignment Report:

Course: Deep Learning for Perception
Instructor: Dr. Jawwad A Shamsi
Semester: Spring 2025 – FAST NUCES, Karachi Campus
Submitted by: Saad Ahmed

Introduction

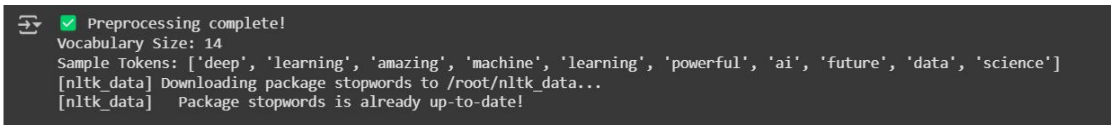
The purpose of this assignment was to implement a simplified version of the Word2Vec algorithm using both the Skip-gram and CBOW architectures. The goal was to process a small text corpus, learn meaningful word embeddings, and visualize their relationships using dimensionality reduction techniques like t-SNE. This report outlines the implementation of Skip-gram and CBOW in five tasks:

Task 1: Preprocessing the Text
Task 2: Implementing the Skip-gram Model
Task 3: Training the Model
Task 4:Visualizing Word Embeddings
Task 5: Implemeting CBOW.

Approach:

TASK 1:

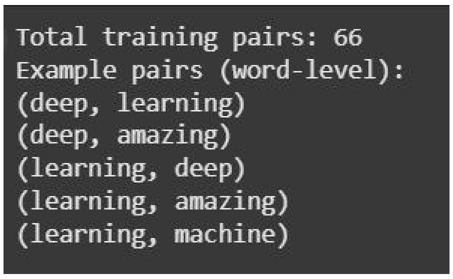
I first created the **text_corpus.txt** file using the example corpus given in the assignment prompt. I read the file content and converted it into lowercase to standardize the text. I then removed punctuation using Python's string module and tokenized the text by simply splitting it into words using the split() function. To clean the data further, I removed common English stopwords using NLTK's stopword list. This helped reduce noise and focus on words that carry actual meaning. Once the text was cleaned, I built a vocabulary of unique words and created two mappings: one from word to index (word2idx) and one from index back to word (idx2word). This helped prepare the text for feeding into the model, as neural networks work with numbers, not raw strings.



```
✅ Preprocessing complete!
Vocabulary Size: 14
Sample Tokens: ['deep', 'learning', 'amazing', 'machine', 'learning', 'powerful', 'ai', 'future', 'data', 'science']
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

TASK 2:

I generated the training pairs for the Skip-gram model. The idea was to take each word as a target and pair it with context words within a fixed window size (2 in this case). I looped through each token and created tuples where the target word was paired with its neighboring words. These pairs were saved in a list and used later for training the model. This step was important because these pairs represent the data the model learns from. Each pair tells the model that the target word is related to its surrounding context words.



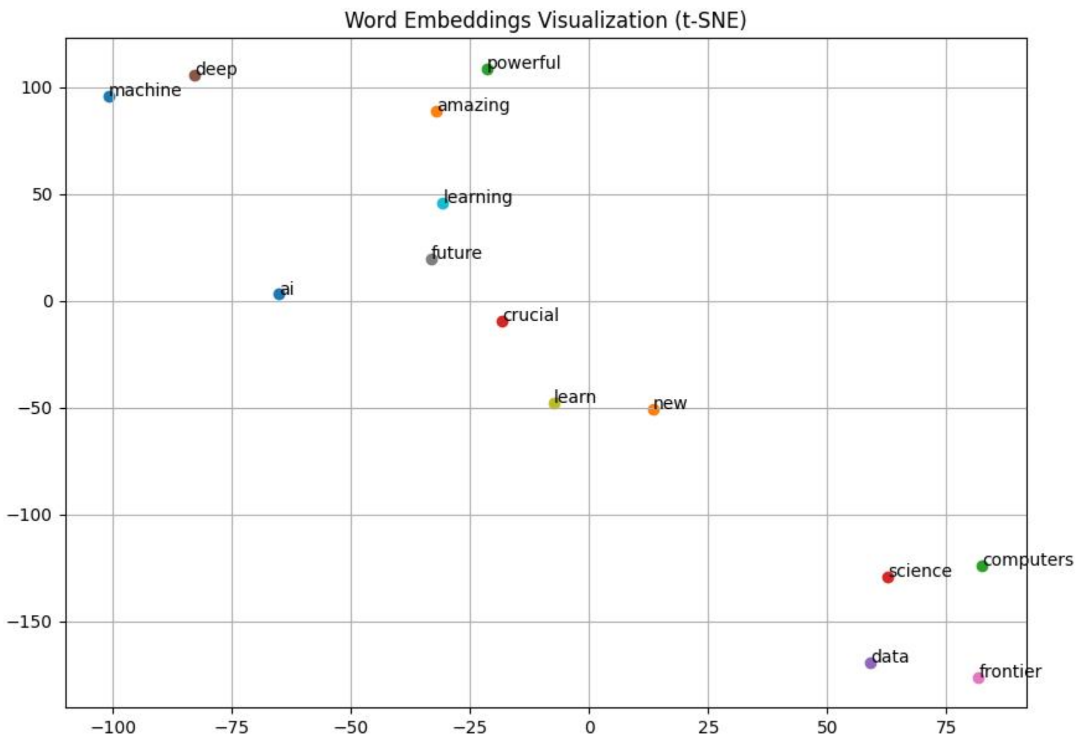
```
Total training pairs: 66
Example pairs (word-level):
(deep, learning)
(deep, amazing)
(learning, deep)
(learning, amazing)
(learning, machine)
```

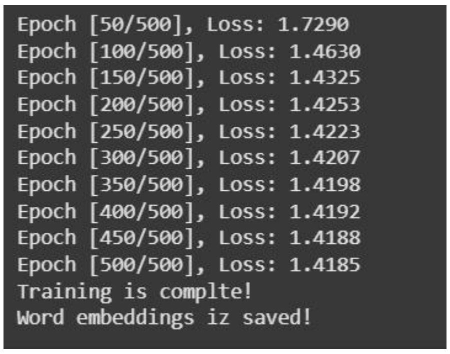
TASK 3:

Task 3 was about building the Skip-gram model using PyTorch. I defined a class with an embedding layer and a linear output layer. The embedding layer maps each word index to a dense vector (which is what we're trying to learn), and the output layer projects that vector to the size of the vocabulary so we can apply softmax and calculate probabilities for each possible context word. The loss function used was CrossEntropyLoss and I used the Adam optimizer for training. I set the embedding dimension to 10, and trained the model using the Skip-gram pairs for 500 epochs. During training, the model gradually learned to predict the correct context words given the input word, which led to the embeddings being shaped meaningfully.

TASK 4:

I saved the learned embeddings and used t-SNE to reduce the dimensions of each embedding vector from 10 to 2. This allowed me to plot the word embeddings in 2D space using Matplotlib. The plot clearly showed that similar or related words appeared closer to each other. For example, "data", "ai", and "learning" were clustered together. This validated that the embeddings were capturing semantic relationships between words based on their usage in context.

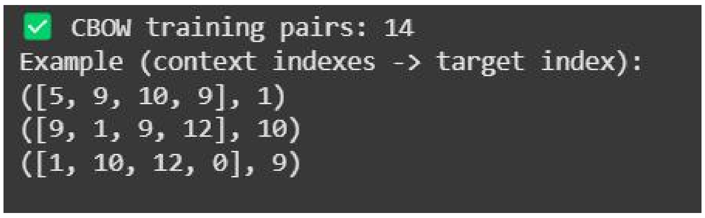




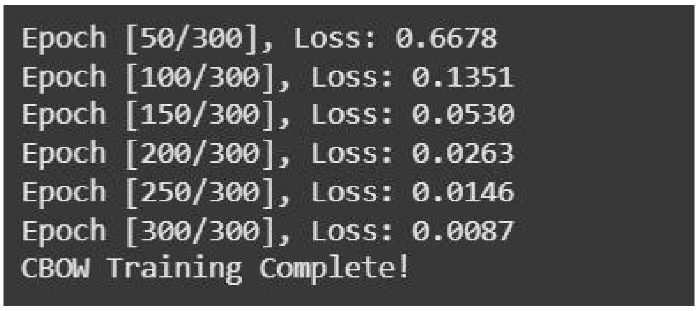
```
Epoch [50/500], Loss: 1.7290
Epoch [100/500], Loss: 1.4630
Epoch [150/500], Loss: 1.4325
Epoch [200/500], Loss: 1.4253
Epoch [250/500], Loss: 1.4223
Epoch [300/500], Loss: 1.4207
Epoch [350/500], Loss: 1.4198
Epoch [400/500], Loss: 1.4192
Epoch [450/500], Loss: 1.4188
Epoch [500/500], Loss: 1.4185
Training is complte!
Word embeddings iz saved!
```

TASK 5:

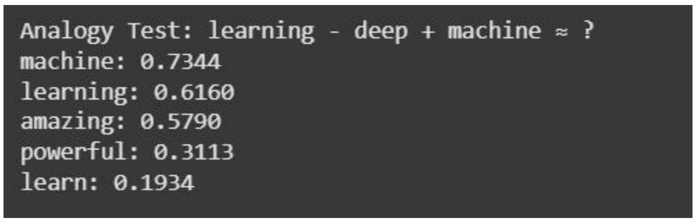
I modified the model to use the CBOW architecture instead of Skip-gram. In CBOW, the input is a set of context words and the model learns to predict the center word. I generated CBOW training pairs by taking a context window around each word (excluding the center word), and averaged their embeddings before passing them through the output layer. I trained the CBOW model on the same corpus and observed similar behavior in loss reduction. I also implemented a basic analogy task to test semantic reasoning within the embeddings. For example, I used the analogy "learning - deep + machine ≈ ?" and retrieved the most similar words based on cosine similarity. The top results showed meaningful associations considering the small vocabulary of the dataset.



```
✅ CBOW training pairs: 14
Example (context indexes -> target index):
([5, 9, 10, 9], 1)
([9, 1, 9, 12], 10)
([1, 10, 12, 0], 9)
```



```
Epoch [50/300], Loss: 0.6678
Epoch [100/300], Loss: 0.1351
Epoch [150/300], Loss: 0.0530
Epoch [200/300], Loss: 0.0263
Epoch [250/300], Loss: 0.0146
Epoch [300/300], Loss: 0.0087
CBOW Training Complete!
```



```
Analogy Test: learning - deep + machine ≈ ?
machine: 0.7344
learning: 0.6160
amazing: 0.5790
powerful: 0.3113
learn: 0.1934
```

Challenges & Observations:

One challenge I faced during preprocessing was with NLTK's tokenizer, which kept giving errors in Colab. I resolved this by using a basic Python split instead of NLTK's word_tokenize. Another key learning was understanding how Skip-gram and CBOW differ not just in direction of prediction but also in how the input is represented (one word vs multiple context words). Visualizing the embeddings really helped me understand what the model was learning and how meaningful word relationships can be captured through numerical vectors. I also learned to work with PyTorch layers like nn.Embedding, how loss and optimizers are used, and how to structure and train a custom model from scratch.

Conclusion:

Overall, this assignment gave me real insight into how word embeddings work and how they can be trained using context-based methods. I feel much more confident in working with text data and building simple NLP models now. Given a larger corpus, these embeddings could be even more powerful, but even with this small dataset, the results were very clear and satisfying.