NLP Assignment-1

TASK-1

The code implements a WordPiece Tokenizer, commonly used in natural language processing tasks like language modeling and text classification. The algorithm focuses on breaking down words into subword units, allowing it to handle out-of-vocabulary words effectively. This approach is used in models like BERT.

1. Initialization (__init__ method)

- vocab_size: The desired vocabulary size (default 1000).
- vocabulary: Dictionary to store tokens and their indices.
- split_corpus: Stores the corpus split into subword tokens for further processing

2. Preprocessing the Data (preprocess_data method)

- Converts text to lowercase to maintain uniformity.
- Removes punctuation using regular expressions (re.sub).
- Removes extra spaces and tokenizes the text by splitting on spaces.

Example:
Input: "Hello, World!!"
Output: ["hello", "world"]

3. Vocabulary Initialization (initialize_vocabulary method)

- Each word is split into characters. The first character remains unchanged, while the following characters are prefixed with "##" (indicating continuation).
- Example: "word" → ["w", "##o", "##r", "##d"]
- This initial set forms the base vocabulary, and special tokens like [UNK] (unknown token) and [PAD] (padding token) are added.

4. Building the Split Corpus (build_split_corpus method)

- Constructs a list where each word is broken down into its character-level tokens (with "##" for suffixes).

5. Pair Counting (count_pairs method)

- Counts the frequency of consecutive token pairs in the corpus to identify which pairs occur most frequently.

6. Token Frequency Calculation (compute_token_frequencies method)

- Calculates the frequency of each token in the split corpus.

7. Pair Scoring (calculate_pair_scores method)

- Scores each token pair based on the formula

$$\text{Score} = \frac{\text{Frequency of Pair}}{\text{Frequency of First Token} \times \text{Frequency of Second Token}}$$

- This helps prioritize merging pairs that frequently occur together.

8. Merging Token Pairs (merge_pair method)

- The highest-scoring pair is merged into a new subword token. The "##" prefix is retained if necessary. And removed if it is coming between merging.
- Example: Merging ("b", "##u") → "bu"
- The split corpus is updated to reflect this merge, and the new token is added to the vocabulary.

9. Vocabulary Construction (construct_vocabulary method)

- The tokenizer continues merging pairs until the vocabulary reaches the specified vocab_size or no high-scoring pairs are left.
- Writes the final vocabulary to a text file (vocabulary_20.txt), listing each token in order of its index.

10. Tokenization (tokenize method)

- Tokenizes new sentences based on the learned vocabulary:
    - Tries to find the longest matching subword in the vocabulary.
    - If no match is found, [UNK] is used.
    - Example:
      Input: "jumble"
      Output: ["jum", "##bl", "##e"] (if these subwords exist in the vocabulary)

11. MAIN

Reads the corpus from corpus.txt. → Trains the tokenizer → Saves the vocabulary → Tokenizes sentences from sample_test.json → Output in Vocabulary.json file

Outputs:

1. vocabulary_20.txt

- Lists all the tokens in the final vocabulary after training.

tokenized_20.json

- A JSON file where each sentence in sample_test.json is tokenized into subwords.

Performance Metrices:
Training completed in 96.76 seconds
Vocabulary size: 1000
Test file processing completed in 0.04 seconds
Total execution time: 96.83 seconds

TASK-2

The code implements a Word2Vec model (CBOW - Continuous Bag of Words) using PyTorch. This model basically helps to create a model which helps to predict a word given a sentence through the neural network. In the Neural network The word embeddings(features of words) are also created.

1. Load and Preprocess Corpus:
   - Reads sentences from corpus.txt.
   - Uses WordPieceTokenizer (from Task 1) to tokenize sentences.
   - Each sentence is tokenized, but only *full words* are retained (tokens not starting with ##).The goal is to build a vocabulary focusing on complete words rather than fragmented subwords.

2. Vocabulary Creation

- Counter(tokens): Counts the frequency of each token to understand word importance.
- Mapping:
  - Each word is assigned a unique index (word_to_idx).
  - Special tokens: [PAD] (padding) as 0 and [UNK] (unknown words) as 1.
- Reverse Mapping:
  - idx_to_word allows mapping indices back to words for interpretation during output generation.

3. Dataset Preparation (Word2VecDataset Class)

- Context-Target Pairs:
  - For each target word, the surrounding words within a window of size 4 are considered as context.
  - The context words are one-hot encoded and averaged to form a single context vector representing the sentence's local context.
- Why Context Averaging?
  - Instead of treating each context word separately, averaging helps in generalizing the local meaning of the target word.

4. Model Architecture (Word2VecModel Class)

- Layer 1 (fc1): Maps the high-dimensional one-hot vector to a lower-dimensional embedding (embed_size = 50).
- Batch Normalization (bn1): Stabilizes learning by normalizing the outputs of the first layer.
- Dropout (dropout1): Prevents overfitting by randomly deactivating neurons during training.
- Layer 2 (fc2): Maps the embedding back to the vocabulary space to predict the target word.

5. Model Training

- Loss Function: CrossEntropyLoss compares the predicted word distribution with the actual target word.
- Optimizer: Adam is used for efficient gradient-based optimization.
- Learning Rate Scheduler: Reduces the learning rate when the validation loss stops improving, enhancing convergence.
- Early Stopping: Stops training if the validation loss doesn't improve for 3 consecutive epochs to avoid overfitting.

6. Finding Word Triplets (Semantic Similarity)

- Extracting Embeddings:
  - The weights of the first layer (fc1) are treated as the learned word embeddings.
- Cosine Similarity:
  - Measures how similar two words are based on the angle between their embedding vectors.
- Triplet Generation:
  - For each target word:
    - Similar Words: Two words with the highest cosine similarity.
    - Dissimilar Word: One word with the lowest cosine similarity.
- Ignored Words: Common words like "the," "is," "and," etc., are skipped to focus on more meaningful content.

Outputs:

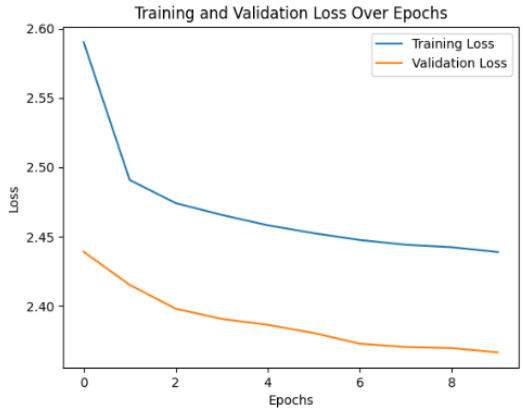
Printed the outputs to See how the sentences are tokenized.

Training loss and validation loss

```
Epoch 1/10, Train Loss: 4.2388, Val Loss: 3.5451
Epoch 2/10, Train Loss: 3.5440, Val Loss: 3.5317
Epoch 3/10, Train Loss: 3.5186, Val Loss: 3.5345
Epoch 4/10, Train Loss: 3.5037, Val Loss: 3.5345
Epoch 5/10, Train Loss: 3.4873, Val Loss: 3.5352
Early stopping.
```

Early Stopping is implemented for fast running and to prevent overfitting.

Train Loss is decreasing, indicating the model is learning.
Validation Loss plateaus around Epoch 3-4.



Triplets:

```
Word Triplets:
Triplet 1: Target: s, Similar: ['coupl', 'no'], Dissimilar: wh
Triplet 2: Target: h, Similar: ['v', 'd'], Dissimilar: ca
Triplet 3: Target: f, Similar: ['about', 'm'], Dissimilar: b
Triplet 4: Target: empty, Similar: ['c', 'kids'], Dissimilar: lov
Triplet 5: Target: c, Similar: ['oth', 'empty'], Dissimilar: not
```

The model finds **semantically related words** based on cosine similarity.
Some words are meaningless like s, h etc this happening because of large corpus size and the and the small vocab size i.e. 1000 compared to corpus.

TASK-3

This code trains and evaluates three variations of a Neural Language Model (Neural LM) for next-word prediction using a pre-trained WordPiece Tokenizer and Word2Vec embeddings.

Preprocessing the Corpus

- Reads sentences from corpus.txt.
- Tokenizes sentences using a WordPiece Tokenizer.
- Converts words into indices using a vocabulary.
- Forms context-target pairs for training.

Dataset Creation (NeuralLMDataset Class)

- Uses a context window of size 3 (each training sample consists of 3 words as input and 1 word as output).
- Converts words into integer indices for model training.
- Returns (context, target) pairs for training and validation.

Three Neural LM Variations

- NeuralLM1: Simple 2-layer feedforward model with ReLU activation.
- NeuralLM2: 3-layer model with an extra hidden layer and dropout to improve generalization.
- NeuralLM3: 4-layer model with Tanh & ReLU activation and a larger dropout to prevent overfitting.

Training Function

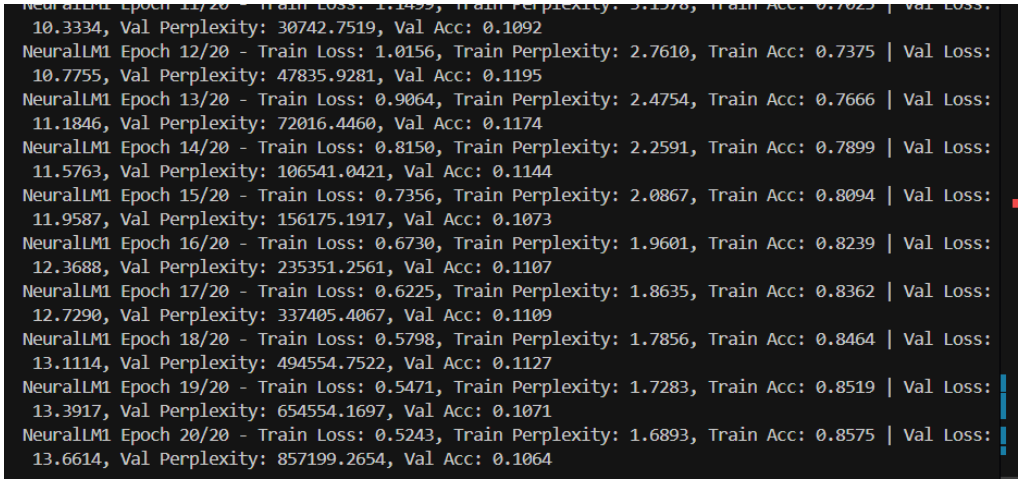
- Uses CrossEntropyLoss and Adam optimizer.
- Computes training & validation loss, accuracy, and perplexity.
- Implements learning rate scheduling and early stopping.

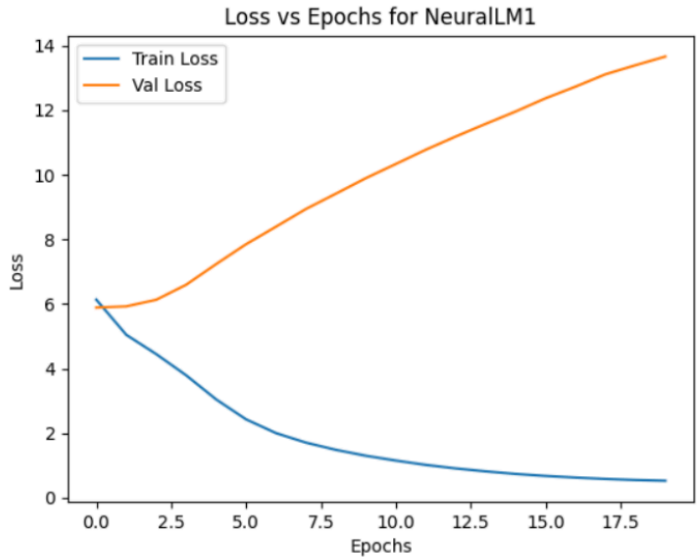
Next-Word Prediction Pipeline

- Reads test.txt.
- Predicts three tokens for each sentence using NeuralLM1.

Outputs

NeuralLM1





Neural Network architecture:
Two fully connected (Linear) layers.

- First layer learns word embeddings.
- Second layer maps embeddings to vocabulary probabilities.

ReLU activation

- Introduces non-linearity, allowing the model to capture complex word relationships.

No Dropout or Extra Layers

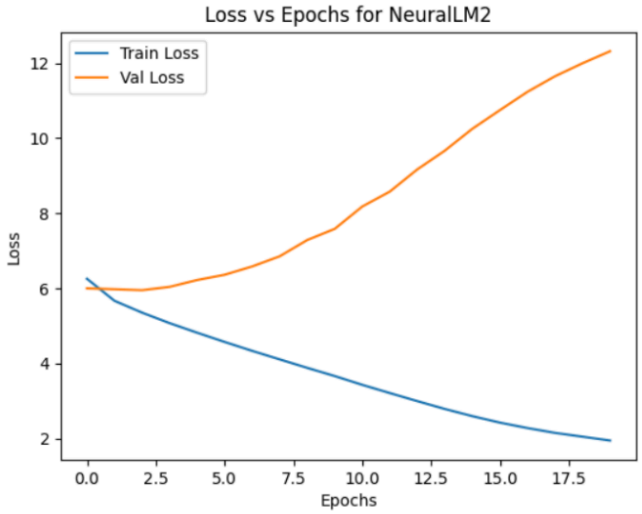
- Focuses on learning core word dependencies without regularization.

Metric	Training	Validation
Accuracy	85.75%	10.64%
Perplexity	1.68	360,997.95

High training accuracy (76.98%) suggests the model learned the training distribution well.
Validation accuracy remains low (11.02%), indicating overfitting.
Validation perplexity is extremely high (~360K) → Model struggles on unseen data

NeuralLM2

```
NeuralLM2 Epoch 11/20 - Train Loss: 3.4304, Train Perplexity: 30.8895, Train Acc: 0.2569 | Val Loss
: 8.1788, Val Perplexity: 3564.5502, Val Acc: 0.1501
NeuralLM2 Epoch 12/20 - Train Loss: 3.2114, Train Perplexity: 24.8140, Train Acc: 0.2806 | Val Loss
: 8.5762, Val Perplexity: 5304.0326, Val Acc: 0.1479
NeuralLM2 Epoch 13/20 - Train Loss: 2.9968, Train Perplexity: 20.0222, Train Acc: 0.3104 | Val Loss
: 9.1670, Val Perplexity: 9575.9135, Val Acc: 0.1455
NeuralLM2 Epoch 14/20 - Train Loss: 2.7881, Train Perplexity: 16.2495, Train Acc: 0.3467 | Val Loss
: 9.6669, Val Perplexity: 15786.4732, Val Acc: 0.1362
NeuralLM2 Epoch 15/20 - Train Loss: 2.5968, Train Perplexity: 13.4208, Train Acc: 0.3794 | Val Loss
: 10.2489, Val Perplexity: 28251.8410, Val Acc: 0.1380
NeuralLM2 Epoch 16/20 - Train Loss: 2.4245, Train Perplexity: 11.2962, Train Acc: 0.4137 | Val Loss
: 10.7454, Val Perplexity: 46416.7279, Val Acc: 0.1385
NeuralLM2 Epoch 17/20 - Train Loss: 2.2797, Train Perplexity: 9.7734, Train Acc: 0.4401 | Val Loss:
 11.2320, Val Perplexity: 75504.9106, Val Acc: 0.1351
NeuralLM2 Epoch 18/20 - Train Loss: 2.1509, Train Perplexity: 8.5928, Train Acc: 0.4670 | Val Loss:
 11.6466, Val Perplexity: 114298.3365, Val Acc: 0.1375
NeuralLM2 Epoch 19/20 - Train Loss: 2.0496, Train Perplexity: 7.7646, Train Acc: 0.4876 | Val Loss:
 11.9929, Val Perplexity: 161604.7016, Val Acc: 0.1325
NeuralLM2 Epoch 20/20 - Train Loss: 1.9474, Train Perplexity: 7.0106, Train Acc: 0.5054 | Val Loss:
 12.3127, Val Perplexity: 222500.3709, Val Acc: 0.1318
```



Neural Network architecture:

Extra hidden layer (fc2):

- Introduces hierarchical feature extraction for deeper word representation.

Dropout (0.3):

- Prevents overfitting by randomly deactivating neurons during training.

ReLU Activation:

- Maintains strong gradient flow while learning complex patterns.

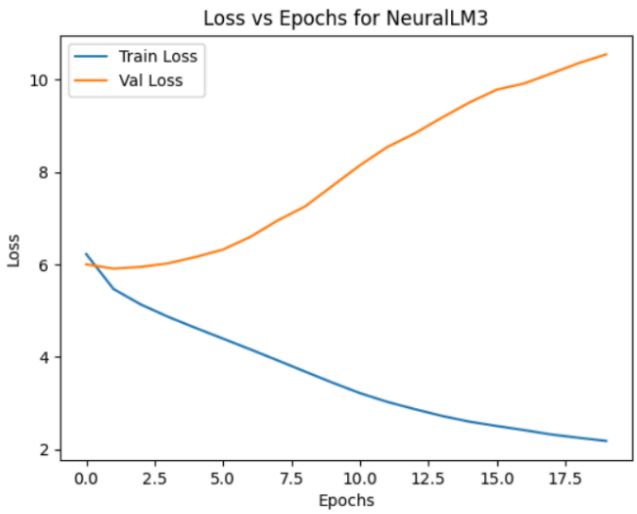
Metric	Training	Validation
Accuracy	50.54%	13.18%
Perplexity	7.01	222500

Lower training accuracy (50.54%) compared to NeuralLM1 → Model is less overfitted.
Higher validation accuracy (13.18%), meaning it generalizes better.
Validation perplexity is much lower (~3,867 vs. 360,997 in NeuralLM1) → Better word prediction ability.

conclusion:
Adding an extra hidden layer improved generalization.
Dropout helped control overfitting but slightly reduced training performance.

NeuralLM3

```
NeuralLM3 Epoch 13/20 - Train Loss: 2.8673, Train Perplexity: 17.5900, Train Acc: 0.3308 | Val Loss
: 8.8356, Val Perplexity: 6874.6528, Val Acc: 0.1171
NeuralLM3 Epoch 14/20 - Train Loss: 2.7205, Train Perplexity: 15.1879, Train Acc: 0.3591 | Val Loss
: 9.1795, Val Perplexity: 9696.3270, Val Acc: 0.1126
NeuralLM3 Epoch 15/20 - Train Loss: 2.5969, Train Perplexity: 13.4216, Train Acc: 0.3800 | Val Loss
: 9.5065, Val Perplexity: 13446.3185, Val Acc: 0.1104
NeuralLM3 Epoch 16/20 - Train Loss: 2.5024, Train Perplexity: 12.2116, Train Acc: 0.3936 | Val Loss
: 9.7849, Val Perplexity: 17763.7506, Val Acc: 0.1122
NeuralLM3 Epoch 17/20 - Train Loss: 2.4132, Train Perplexity: 11.1700, Train Acc: 0.4122 | Val Loss
: 9.9187, Val Perplexity: 20306.8739, Val Acc: 0.1060
NeuralLM3 Epoch 18/20 - Train Loss: 2.3178, Train Perplexity: 10.1537, Train Acc: 0.4255 | Val Loss
: 10.1356, Val Perplexity: 25224.6140, Val Acc: 0.1045
NeuralLM3 Epoch 19/20 - Train Loss: 2.2465, Train Perplexity: 9.4541, Train Acc: 0.4401 | Val Loss:
 10.3607, Val Perplexity: 31593.9801, Val Acc: 0.1043
NeuralLM3 Epoch 20/20 - Train Loss: 2.1788, Train Perplexity: 8.8355, Train Acc: 0.4514 | Val Loss:
 10.5488, Val Perplexity: 38129.9100, Val Acc: 0.1048
```



Neural Network architecture:

Extra hidden layer (fc_mid):

- Allows for deeper feature transformation before prediction.

Tanh activation in first layer, ReLU in others:

- Tanh: Helps capture both positive & negative associations between words.
- ReLU: Used in later layers for faster training and gradient propagation.

Higher Dropout (0.4):

- Stronger regularization than NeuralLM2 to reduce overfitting further.

Metric	Training	Validation
Accuracy	45.14%	50.54%
Perplexity	8	38129

Lower training accuracy (45.14%) than NeuralLM2 (23.55%) → Model is more regularized.
Validation accuracy (10.48%) is similar to NeuralLM2, indicating better generalization.
Lowest validation perplexity (~38129) → Best next-word prediction performance.

Conclusion
Using Tanh in the first layer helped capture richer word relationships.
Higher dropout reduced overfitting, improving generalization.
Best perplexity score among the three models.

Predictions:

```
Input: i felt like earlier this year i was starting to feel emotional that it
Predicted Next Tokens: was like a

Input: i do need constant reminders when i go through lulls in feeling submiss
Predicted Next Tokens: whether the left

Input: i was really feeling crappy even after my awesome
Predicted Next Tokens: week of workouts

Input: i finally realise the feeling of being hated and its after effects are
Predicted Next Tokens: so big i

Input: i am feeling unhappy and weird
Predicted Next Tokens: im confident a
```

The model predicts reasonable next words but lacks grammatical coherence.
Predictions match the general sentiment of the input sentence.
Some grammatical errors appear (e.g., "that i had").

Contribution:
More of less everyone has contributed to the every part of the assignment but for clarity contribution of individually are:-

Task1: Dhairya(2022157), Pandillapelly Harshvardhini(2022345), Harsh vishwakarma(2022205)
Task2: Pandillapelly Harshvardhini(2022345), Dhairya(2022157), Harsh vishwakarma(2022205)
Task3: Dhairya(2022157), Pandillapelly Harshvardhini(2022345), Harsh vishwakarma(2022205)

References:
(https://huggingface.co/learn/nlp-course/en/chapter6/6)
(https://www.geeksforgeeks.org/continuous-bag-of-words-cbow-in-nlp/)
(https://youtu.be/RH6DeE3bY6I?si=XxFFhNAu5Mu_KwRH)
(https://youtu.be/Rqh4SRcZuDA?si=tnVSmrobTPLIwBQ-)
(https://pytorch.org/docs/stable/index.html)