CSE556: Natural LanguageProcessing
Assignment-01

—---

Task 1 - Implement WordPiece Tokenizer

Steps involved in WordPiece Tokenizer implementation are as follows:

- **Preprocessing:**
  The *preprocess_data* method in the WordPieceTokenizer class performs the following steps :
  1. Lowercase
  2. Removing non-alphanumeric characters using regex lib
  3. Normalizing spaces.

  Example: "Hello, world!" → "hello world".

- **Vocabulary Construction:**
  The *construct_vocabulary* method performs the following steps:
  1. Read and Preprocess Corpus: The input corpus is read and preprocessed using the preprocess_data method.
  2. Initialize Vocabulary: The vocabulary is initialized with character-level tokens and full words.
  3. Count the frequency of each subword in the training data.
  4. Keep track of which subwords appear at the start of words vs. in the middle/end.
  5. Merge Frequent Pairs:
     i) Score Calculation:
     *Score(A, B) = (freq(AB) * len(vocab)) / (freq(A) * freq(B))*
     - *freq(AB) is the frequency of the combined token*
     - *len(vocab) is the current vocabulary size*
     - *freq(A) and freq(B) are the individual frequencies of A and B*

     ii) Pair selection is based on the highest score
     iii) Add the merged token to the vocabulary.
     iv) If it's not a word-initial subword, add it with the ## prefix.
     v) Update the tokenization of the training data to use this new token where applicable.
  6. Save Vocabulary: The final vocabulary is saved to a text file (vocabulary_54.txt), with each token on a new line.

- **Tokenization:** The tokenize method splits a given sentence into tokens using the *construct_vocabulary*. It uses a *greedy longest-match-first* (*ensures consistent tokenization*) approach to tokenize words
  Tokenize Words: Split the input text into words.
    For each word:
    i). Try to match the longest subword from the vocabulary at the start of the word.
    ii). If a match is found, add it to the output and repeat from the next character.
    iii). If no match is found, add [UNK] token and move to the next character.
    iv). For matches after the first character, use the ## version of the subword.

Example taken from lecture slide-02

a. Corpus: (“hug”, 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
b. Character boundary: (“h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##g" "##s", 5)
c. Initial Vocabulary: (“b”, 4), (“h”, 5), (“p”, 17), (“##g”, 20), (“##n”, 16), (“##s”, 5), (“##u”, 36)
d. Pair Frequencies: (“##u", "##g") = 20, ("##g", "##s") = 5, ("##u", "##n") = 16, ("##u", "##s") = 0, ...
e. Compute likelihood: score = (freq_of_pair) / (freq_of_first_element × freq_of_second_element)
f. Vocabulary Update: ["b", "h", "p", "##g", "##n", "##s", "##u", "##gs"]
g. Token boundary: ("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##gs", 5)
h. 2nd Merge ("h", "##u") → “hu”: Vocabulary Update: ["b", "h", "p", "##g", "##n", "##s", "##u", "##gs", "hu"]
i. 3rd Merge ("hu", "##g") → “hug”: Vocabulary Update: ["b", "h", "p", "##g", "##n", "##s", "##u", "##gs", "hu", "hug"]

References:

1. https://medium.com/@atharv6f_47401/wordpiece-tokenization-a-bpe-variant-73cc48865cbf
2. https://www.youtube.com/watch?v=qpv6ms_t_1A&t=1s
3. https://huggingface.co/learn/nlp-course/en/chapter6/6
4. Lecture Slide -02 Text Processing

Task 2 - Implement Word2vec

The **word2vec** tool takes a text corpus as input and produces the word vectors as output. It first constructs a vocabulary from the training text data and then learns vector representation of words. **CBOW (continuous bag of words)** algorithms learn the representation of a word that is useful for prediction of other words in the sentence

Dataset Preparation

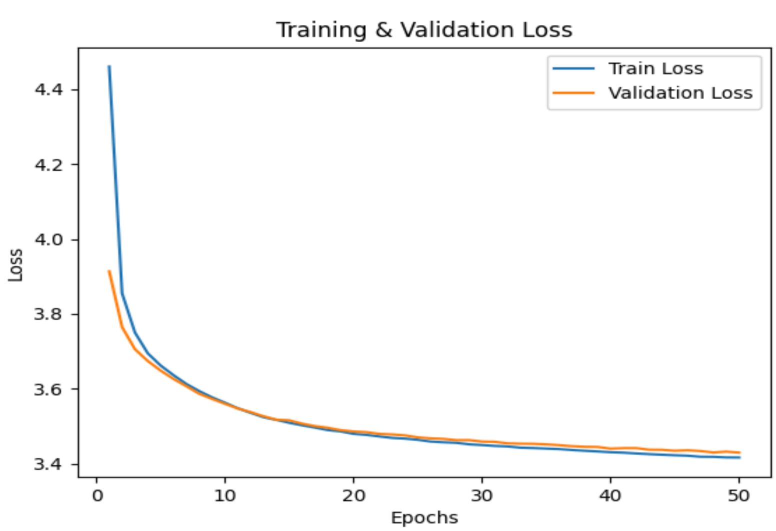
- Used **WordPiece Tokenizer** to segment text into subwords.
- Constructed vocabulary and handled unknown words (<UNK>).
- Created **context-target pairs** with a window size of **2**.

Model Architecture

- Embedding layer (200-dimensional vectors).
- Mean pooling over context word embeddings.
- Fully connected layers with **ReLU activation**.
- **Dropout layer (with probability 0.3)** to prevent overfitting.
- Output layer predicts the target word.

Training Process

- **Optimizer:** Adam with learning rate scheduling (0.0001).
- **Loss function:** Cross-Entropy Loss.
- **Techniques:** Gradient clipping, early stopping, and weight decay (0.0001).
- **Epochs:** Up to 50, but early stopping applied.



The training and validation loss curves indicate a well-behaved learning process:

1. **Effective Learning**: The loss decreases smoothly, with a rapid decline in the early epochs, showing that the model is learning effectively.
2. **Minimal Overfitting**: The validation loss closely follows the training loss, suggesting good generalization.
3. **Stable Convergence**: After around 45 epochs, the loss stabilizes, indicating that additional training may not provide significant gains.

Performance Analysis

- Training and validation loss graph shows **effective learning**.
- **Early stopping** prevents overfitting.
- The trained model is saved as word2vec_cbow_model.pth.

Cosine Similarity & Triplet Analysis

- Cosine similarity metric used to measure word relationships.
- Identified **two triplets**:

```
Selected Triplets:
Similar: outbursts ##motional Cosine Similarity: 0.9983
Dissimilar: willing Cosine Similarity: -0.9826

Similar: ##motional outbursts Cosine Similarity: 0.9983
Dissimilar: willing Cosine Similarity: -0.9826
```

References:

1. https://code.google.com/archive/p/word2vec/
2. https://towardsdatascience.com/a-word2vec-implementation-using-numpy-and-python-d256cf0e5f28
3. https://pytorch.org/tutorials/beginner/basics/data_tutorial.html
4. https://www.youtube.com/watch?v=viZrOnJclY0
5. https://www.youtube.com/watch?v=Qf06XDYXCXI
6. https://www.youtube.com/watch?v=ZcHfDEYOPIQ
7. https://pytorch.org/text/stable/datasets.html

Task 3 - Train a Neural LM

Development and evaluation of three neural language models (NeuralLM1, NeuralLM2, and NeuralLM3) for next-word prediction tasks. The script includes the following key components:

1. Dataset Preparation:

- **NeuralLMDataset Class:** This class processes a text corpus to generate context-target pairs for training. It utilizes a tokenizer (WordPieceTokenizer) and a pre-trained Word2Vec model to convert words into indices and embeddings.

2. Model Architectures:

NeuralLM1:

- **Architecture**: One embedding layer followed by two fully connected layers.
- **Activation Function**: ReLU is used for non-linearity.
- **Key Features**: Simple architecture with one hidden layer, suitable for basic language modeling tasks.

Performance Metrics:

- Training Accuracy: ~51%
- Validation Accuracy: ~48%
- Higher perplexity scores indicate limited language modeling capability

NeuralLM2:

- **Architecture**: Embedding layer followed by three fully connected layers.
- **Activation Functions**: Tanh for the first hidden layer, ReLU for the second.
- **Key Features**: More complex than NeuralLM1, with an additional hidden layer and mixed activations for better learning of complex patterns.

Performance Metrics:

- Training Accuracy: ~55%
- Validation Accuracy: ~53%
- Lower perplexity compared to NeuralLM1 suggests better language modeling

NeuralLM3:

- **Architecture**: Embedding layer followed by four fully connected layers.
- **Activation Functions**: ReLU for the first and third layers, Sigmoid for the second.

- **Key Features**: The most complex model with three hidden layers, suitable for learning deeper, more refined patterns in the data.

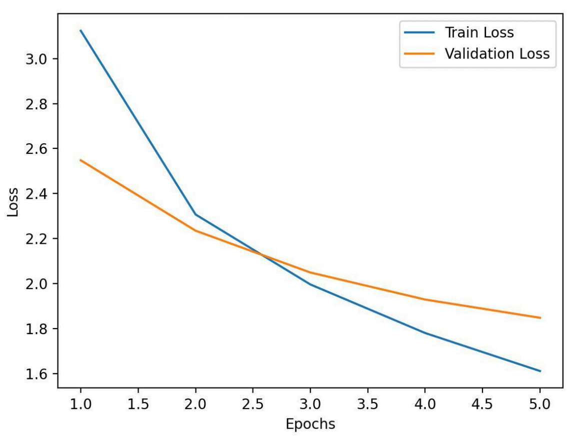
Performance Metrics:

- Training Accuracy: ~63%
- Validation Accuracy: ~60%
- Lowest perplexity scores indicating superior language modeling capability

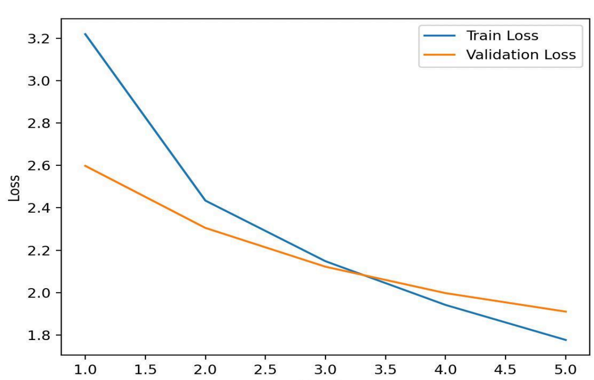
3. Training Function:

- The train function trains each model using the provided dataset. It includes early stopping based on validation loss to prevent overfitting and plots training and validation losses over epochs.

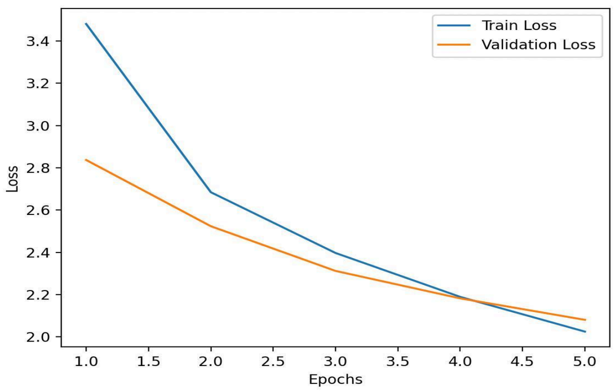
Model-01 Training Vs Validation Graph



Model-02 Training Vs Validation Graph



Model-03 Training Vs Validation Graph



- Model-03 performs the best with stable learning and good generalization, showing consistent convergence between training and validation loss.
- Model-02 shows decent performance but with slight divergence.
- Model-01 exhibits potential overfitting issues after epoch 2.5 where validation loss starts to plateau while training loss keeps decreasing.

Key Differences:

1. Model Complexity vs. Performance:
   - Each additional layer and activation function modification led to improved accuracy
   - NeuralLM3's deeper architecture showed best generalization capability
2. Learning Stability:
   - NeuralLM1: Showed signs of underfitting
   - NeuralLM2: More stable learning but still room for improvement
   - NeuralLM3: Most stable learning curve with best convergence
3. Overall Performance:
   - Clear progression in performance metrics from NeuralLM1 to NeuralLM3
   - The deeper architectures with mixed activation functions performed better at capturing language patterns

4. Evaluation Metrics:

- **Accuracy:** Measures the proportion of correct predictions. Training accuracy peaked at 63%, validation accuracy reached 60.86%
- Best training loss was 1.61, validation loss reached 1.84
- **Perplexity:** Assesses the model's uncertainty in predicting the next word; lower perplexity indicates better performance.
- Training perplexity varied between 5.6-12.4, validation perplexity between 6.3-13.4

```
Epoch 1/5, Train Loss: 3.1235, Val Loss: 2.5477, Train Accuracy: 0.5100, Val Accuracy: 0.4876, Train Perplexity: 11.1221, Val Perplexity: 12.7730
Epoch 2/5, Train Loss: 2.3069, Val Loss: 2.2349, Train Accuracy: 0.5839, Val Accuracy: 0.5472, Train Perplexity: 7.3485, Val Perplexity: 9.2876
Epoch 3/5, Train Loss: 1.9961, Val Loss: 2.0491, Train Accuracy: 0.6325, Val Accuracy: 0.5848, Train Perplexity: 5.6377, Val Perplexity: 7.7553
Epoch 4/5, Train Loss: 1.7802, Val Loss: 1.9289, Train Accuracy: 0.6705, Val Accuracy: 0.6086, Train Perplexity: 4.6152, Val Perplexity: 6.8792
Epoch 5/5, Train Loss: 1.6113, Val Loss: 1.8479, Train Accuracy: 0.7048, Val Accuracy: 0.6254, Train Perplexity: 3.9187, Val Perplexity: 6.3456
Epoch 1/5, Train Loss: 3.2188, Val Loss: 2.5977, Train Accuracy: 0.4837, Val Accuracy: 0.4735, Train Perplexity: 12.3883, Val Perplexity: 13.4370
Epoch 2/5, Train Loss: 2.4339, Val Loss: 2.3051, Train Accuracy: 0.5532, Val Accuracy: 0.5322, Train Perplexity: 8.6115, Val Perplexity: 10.0317
Epoch 3/5, Train Loss: 2.1480, Val Loss: 2.1221, Train Accuracy: 0.6026, Val Accuracy: 0.5678, Train Perplexity: 6.7235, Val Perplexity: 8.3594
Epoch 4/5, Train Loss: 1.9418, Val Loss: 1.9976, Train Accuracy: 0.6430, Val Accuracy: 0.5987, Train Perplexity: 5.5691, Val Perplexity: 7.3675
Epoch 5/5, Train Loss: 1.7768, Val Loss: 1.9104, Train Accuracy: 0.6731, Val Accuracy: 0.6148, Train Perplexity: 4.7629, Val Perplexity: 6.7571
Epoch 1/5, Train Loss: 3.4582, Val Loss: 2.8415, Train Accuracy: 0.4231, Val Accuracy: 0.4213, Train Perplexity: 16.0519, Val Perplexity: 17.1345
Epoch 2/5, Train Loss: 2.6791, Val Loss: 2.5202, Train Accuracy: 0.5062, Val Accuracy: 0.4861, Train Perplexity: 10.8209, Val Perplexity: 12.4368
Epoch 3/5, Train Loss: 2.3838, Val Loss: 2.3281, Train Accuracy: 0.5590, Val Accuracy: 0.5252, Train Perplexity: 8.3340, Val Perplexity: 10.2427
Epoch 4/5, Train Loss: 2.1717, Val Loss: 2.1980, Train Accuracy: 0.5980, Val Accuracy: 0.5527, Train Perplexity: 6.8438, Val Perplexity: 9.0122
Epoch 5/5, Train Loss: 2.0088, Val Loss: 2.0977, Train Accuracy: 0.6322, Val Accuracy: 0.5739, Train Perplexity: 5.8122, Val Perplexity: 8.1529
```

5. Next-Word Prediction:

- The predict_next_tokens function predicts the next three tokens for each sentence in a test file, demonstrating the model's practical application

References:

1. Lecture side-04 Word representation
2. https://scikit-learn.org/stable/modules/neural_networks_supervised.html
3. https://mize.tech/blog/how-does-a-neural-network-work-implementation-and-5-examples/#:~:text=Now%20let's%20move%20on%20to,is%20added%20to%20each%20input.
4. https://www.youtube.com/watch?v=y6ibRbAdV3c

Individual Contributions

All team members contributed equally to the project, with each member actively participating in different aspects of the work. The workload was distributed evenly