ASSIGNMENT-I

Task-1: Implement WordPiece Tokenizer

In that task 1st we define what is **WordPiece Tokenizer** is and why we use:
The WordPiece Tokenizer works by iteratively merging the most frequent subword pairs in a corpus, maximizing the likelihood of the training data. It starts with a vocabulary of characters and subword tokens. The algorithm computes the likelihood of merging two subwords using the formula to compute likelihood:

$$\text{score} = (\text{freq\_of\_pair}) / (\text{freq\_of\_first\_element} \times \text{freq\_of\_second\_element})$$

It then merges the pair with the highest score, updating the vocabulary. This process continues until a predefined vocabulary size is reached.
For example, in the corpus ("hug", "pug", "pun", "bun", "hugs"), the most frequent pair "##u" and "##g" is merged into "##gs". This merging process continues to build subwords like "hug" from smaller components, allowing efficient handling of rare words.
This approach ensures the tokenizer captures frequent subword patterns, making it effective for handling out-of-vocabulary words.

Now the above task we Implement this Tokenizer from scratch using only standard Python libraries as well as Numpy and pandas.

1st we create the **WordPieceTokenizer Class** ,in that we define the function

- Preprocess_data: Although in corpus.txt there is no need of preprocessing because corpus is already preprocessd. But we still parform it. So convert to lowercase, normalize whitespace, and tokenize words + punctuation.
- Compute_word_frequencies: Compute the frequency of each unique word in the corpus.
- Bulid_alphabet: This function are use to create the set of unique characters or vocabulary.
- Construct_splits: Create initial subword splits, adding '##' prefixes to non-initial characters.
- Compute_pair_scores: Compute pair scores for WordPiece tokenization.
- Mereg_pair: Merge the most frequent token pair into a single token.
- Construct_vocabulary: Builds WordPiece vocabulary from the given corpus file.
- Encode_word: Encode a single word into subwords using the WordPiece vocabulary.
- Tokenize: Tokenizes an input sentence into subwords using the WordPiece

tokenizer.

- Tokenize_from_file: Reads sentences from JSON, tokenizes them, and saves output.

  These all function we create and with the help of above , we perform the task 1.

Now we provide the some output snipet of task 1.
Vocabulary:

```
[UNK]
[PAD]
##a
##b
##c
##d
##e
##f
##g
##h
##i
##j
##k
##l
##m
##n
##o
##p
##q
##r
##s
##t
##u
##v
##w
##x
##y
##z
```

Tokenize:

Task-2: Implement Word2vec

In that task we Implement **Word2vec model using CBOW(**Continuous Bag of Words) approach from scratch in pyTorch.

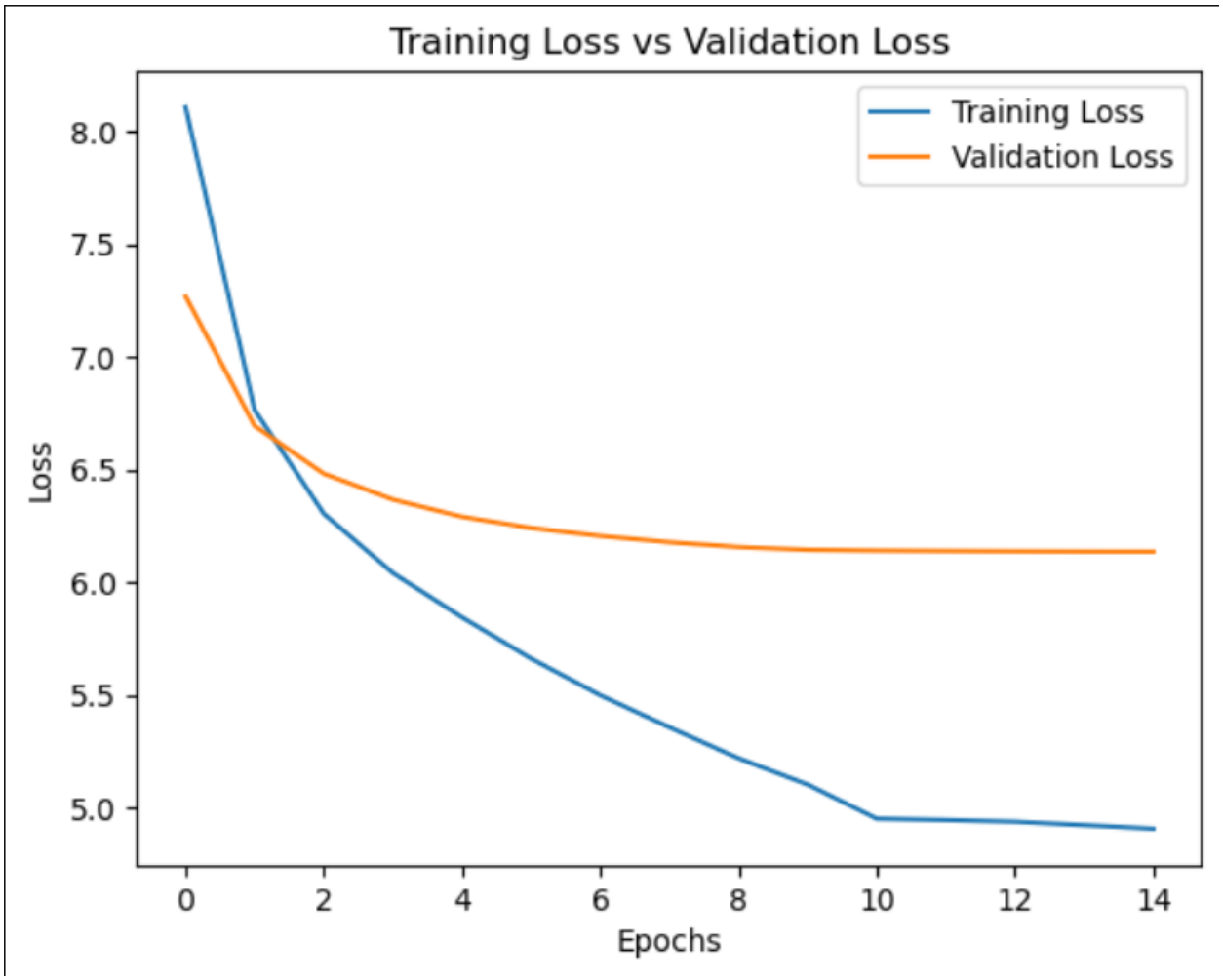
In that the important things is how we use the task 1 (WordPiece Tokenizer) Which perform the tokenization of corpus.txt which we provide as input to the tokenizer.
After creating the tokenization file of the corpus . after that we create **Word2VecDataset Class** which generate the traning_data, which store the pair of all possible pair's of (context,target) in dataset. Total number of training pairs are 144000 which store like:

```
Context: i stand i feel  Target: here
Context: stand here feel empty   Target: i
Context: here i empty a  Target: feel
Context: i feel a class  Target: empty
Context: feel empty class post   Target: a
```

After that the most important thing in that task to make a model Word2VecModel(CBOW). Now we define something about the model means what they and how we train it.
In the model Vocabulary size means the unique word in the corpus is 5430 and the embedding dimension is 300 (means how many feature we use for pr word).

In that model have two type of embedding .
In_embedding: this is nn.embedding layer which learn the input words embedding.
Out_embedding: This is also a nn.embedding layer which store the word embedding of output word.this layer are use for negative sampling…

After that we call the train function which spilit the data training data size 100800 and validation is 43200. And genrate the corss_entorpy_loss graph of traning and validation vs epochs:



Identify two triplets:

```
Word Similarity Triplets:
triplets:
◆  a is similar to ad but different from activities
◆  aa is similar to acronym but different from aesthetics
 a is similar to ad but different from activities
 aa is similar to acronym but different from aesthetics
 aahhh is similar to abyss but different from ached
 abandoning is similar to addiction but different from ached
 abilities is similar to academics but different from adventure
 ability is similar to advance but different from aa
 abit is similar to afraid but different from admire
 able is similar to accepted but different from ad
 abound is similar to abundance but different from abyss
 about is similar to achieve but different from adore
 above is similar to aching but different from added
 absolute is similar to ached but different from a
 absolutely is similar to actual but different from actually
```

```
 Cosine Similarity for 10 pairs:
 Cosine Similarity of (kitchen, ra) = 0.0393
 Cosine Similarity of (kitchen, stuff) = 0.0361
 Cosine Similarity of (kitchen, ini) = 0.0421
 Cosine Similarity of (kitchen, superman) = -0.0739
 Cosine Similarity of (kitchen, antics) = -0.0570
 Cosine Similarity of (kitchen, broke) = 0.0633
 Cosine Similarity of (kitchen, sandwiches) = 0.0389
 Cosine Similarity of (kitchen, exam) = 0.0225
 Cosine Similarity of (kitchen, losing) = 0.0175
 Cosine Similarity of (kitchen, paint) = -0.0820
```

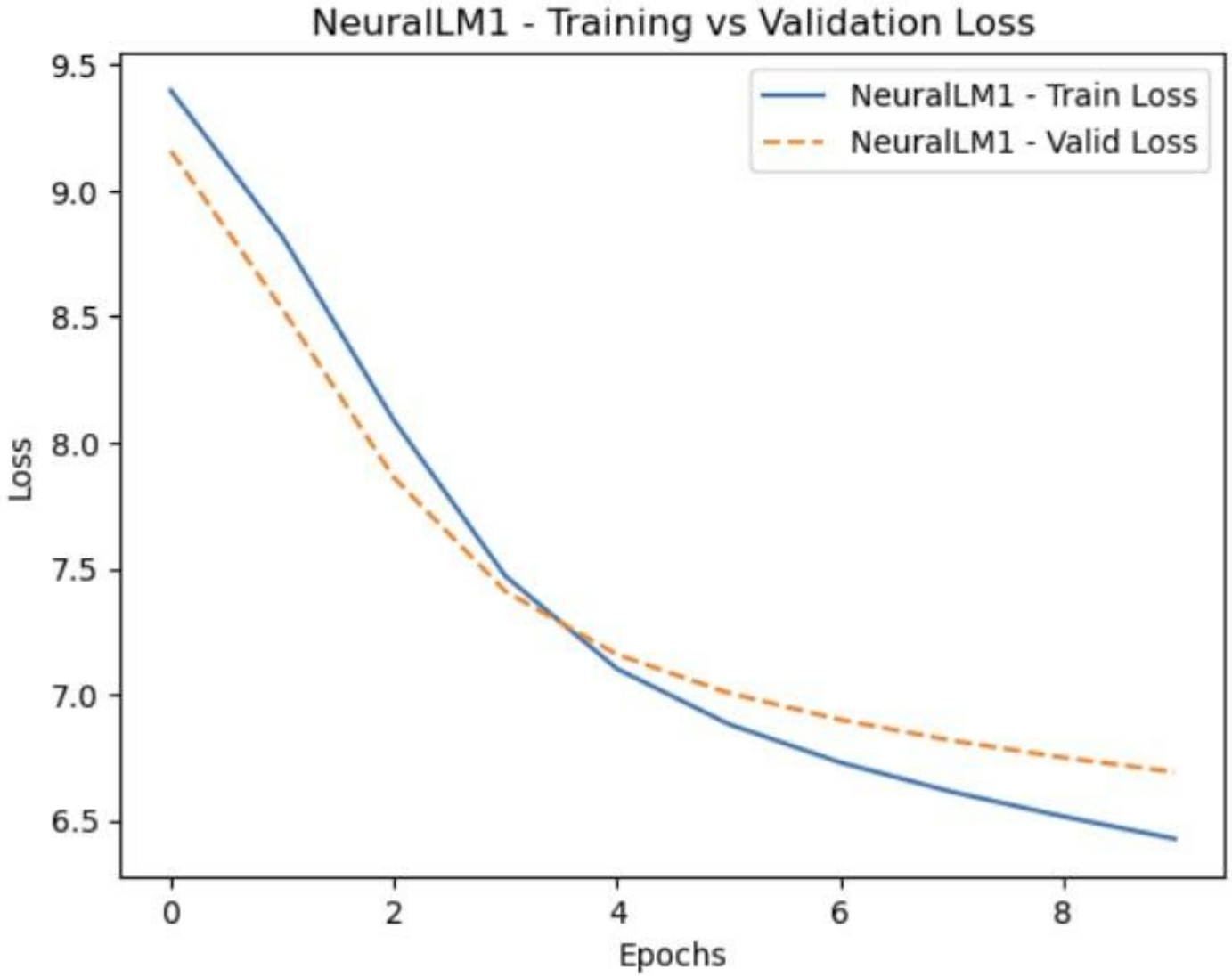
Task-3: Train a Neural LM

In this task we perform train their **Neural Language Model** which will be MLp based using pytorch. In that task the major thing is that to store to merge the task 1 and task 2 and this 3rd task we directly use the saved model and wordpiecetokanizer. This task are mearg in **NeuralLMDataset** which is the class to train the language model.

Now after that class we Implement the classes named NeuralLM1,NeuralLM2 and NeuralLM3 . in that in NeuralLM1 we use the activation function :ReLu(**Rectified Linear Unit**) and on other 2 we use different one now me give some ouput which we find.

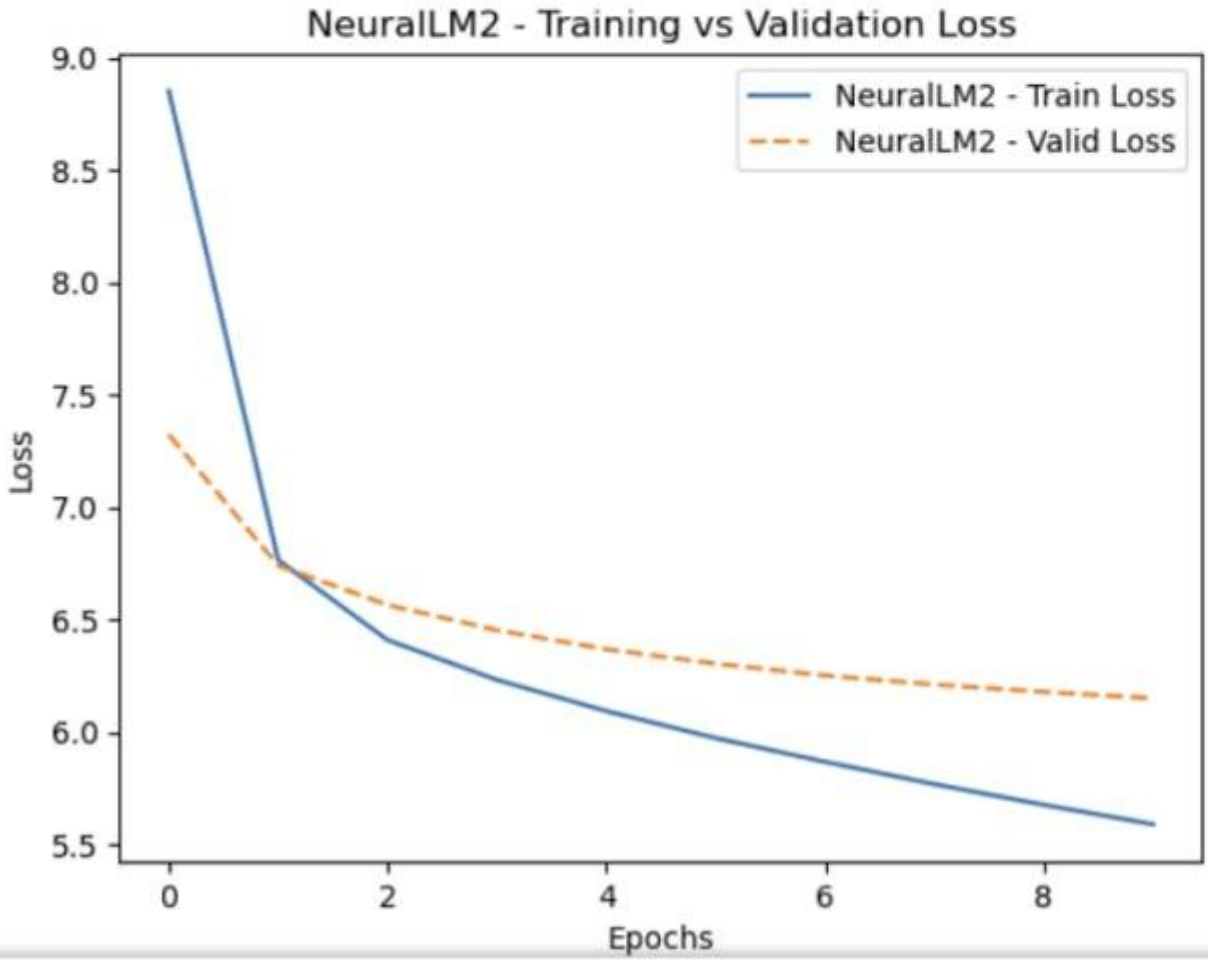
```
Training NeuralLM1...
Epoch 1/10 | Train Loss: 9.3979 | Valid Loss: 9.1579
Epoch 2/10 | Train Loss: 8.8176 | Valid Loss: 8.5305
Epoch 3/10 | Train Loss: 8.0877 | Valid Loss: 7.8609
Epoch 4/10 | Train Loss: 7.4699 | Valid Loss: 7.4094
Epoch 5/10 | Train Loss: 7.1020 | Valid Loss: 7.1587
Epoch 6/10 | Train Loss: 6.8826 | Valid Loss: 7.0059
Epoch 7/10 | Train Loss: 6.7300 | Valid Loss: 6.8997
Epoch 8/10 | Train Loss: 6.6137 | Valid Loss: 6.8171
Epoch 9/10 | Train Loss: 6.5155 | Valid Loss: 6.7493
Epoch 10/10 | Train Loss: 6.4270 | Valid Loss: 6.6917

NeuralLM1 - Train Accuracy: 0.1117, Train Perplexity: 552.9150
NeuralLM1 - Valid Accuracy: 0.1078, Valid Perplexity: 805.6644
```



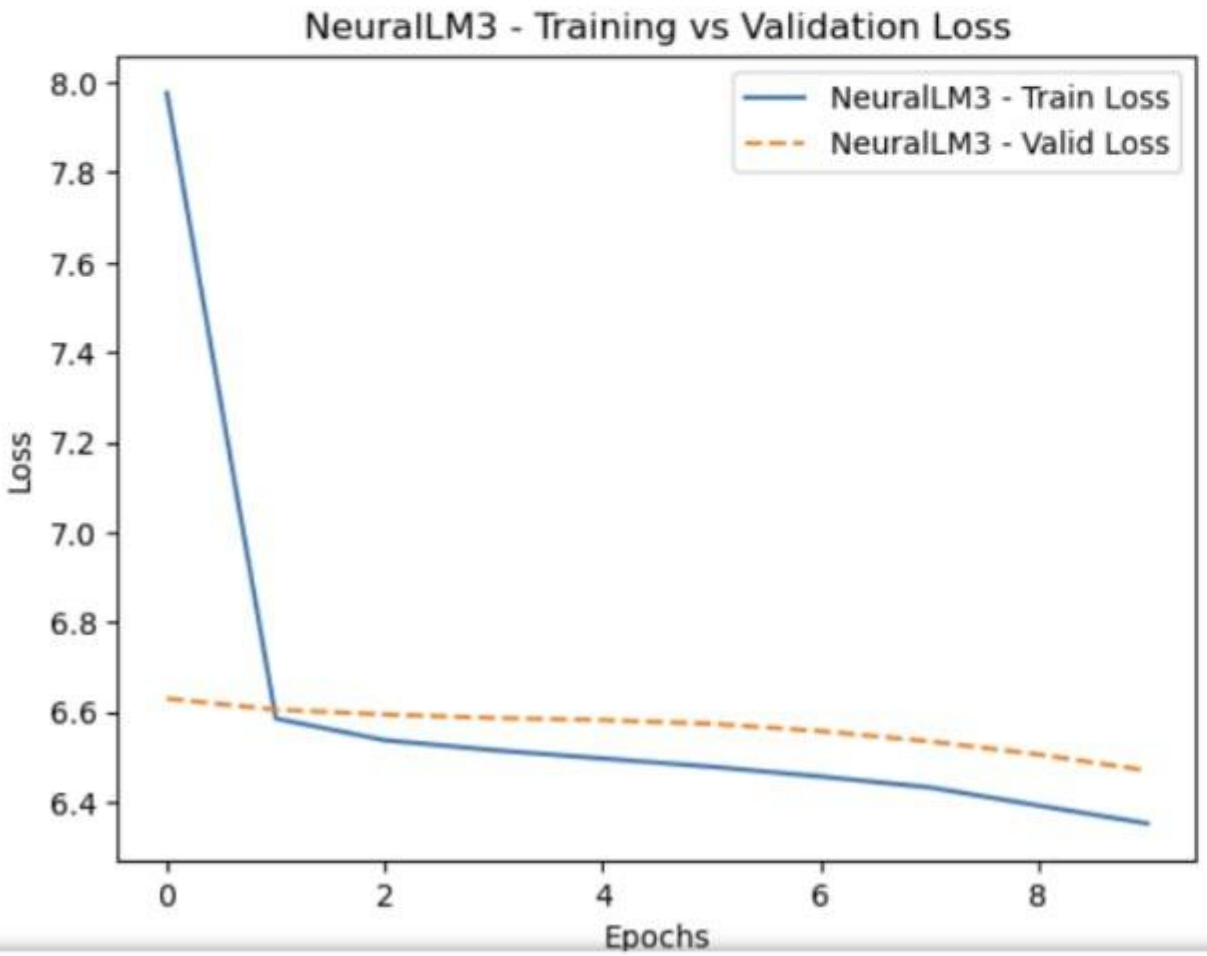
```
Training NeuralLM2...
Epoch 1/10 | Train Loss: 8.8507 | Valid Loss: 7.3219
Epoch 2/10 | Train Loss: 6.7648 | Valid Loss: 6.7398
Epoch 3/10 | Train Loss: 6.4088 | Valid Loss: 6.5646
Epoch 4/10 | Train Loss: 6.2315 | Valid Loss: 6.4525
Epoch 5/10 | Train Loss: 6.0928 | Valid Loss: 6.3680
Epoch 6/10 | Train Loss: 5.9727 | Valid Loss: 6.3035
Epoch 7/10 | Train Loss: 5.8670 | Valid Loss: 6.2513
Epoch 8/10 | Train Loss: 5.7676 | Valid Loss: 6.2109
Epoch 9/10 | Train Loss: 5.6759 | Valid Loss: 6.1781
Epoch 10/10 | Train Loss: 5.5893 | Valid Loss: 6.1498

NeuralLM2 - Train Accuracy: 0.1263, Train Perplexity: 244.7528
NeuralLM2 - Valid Accuracy: 0.1187, Valid Perplexity: 468.6016
```



```
Training NeuralLM3...
Epoch 1/10 | Train Loss: 7.9765 | Valid Loss: 6.6299
Epoch 2/10 | Train Loss: 6.5854 | Valid Loss: 6.6054
Epoch 3/10 | Train Loss: 6.5374 | Valid Loss: 6.5945
Epoch 4/10 | Train Loss: 6.5153 | Valid Loss: 6.5868
Epoch 5/10 | Train Loss: 6.4968 | Valid Loss: 6.5822
Epoch 6/10 | Train Loss: 6.4784 | Valid Loss: 6.5739
Epoch 7/10 | Train Loss: 6.4566 | Valid Loss: 6.5577
Epoch 8/10 | Train Loss: 6.4325 | Valid Loss: 6.5348
Epoch 9/10 | Train Loss: 6.3921 | Valid Loss: 6.5059
Epoch 10/10 | Train Loss: 6.3521 | Valid Loss: 6.4700

NeuralLM3 - Train Accuracy: 0.0717, Train Perplexity: 520.2812
NeuralLM3 - Valid Accuracy: 0.0707, Valid Perplexity: 645.4841
```



Reference:

Task 1 - [WordPiece tokenization - Hugging Face NLP Course](), [WordPiece Tokenization - YouTube]().

Task-2: [Word2vec Complete Tutorial | CBOW and Skip-gram | Game of Thrones Word2vec - YouTube](), [word2vec_assignment/w2v_hw/w2v_hw.ipynb at master ·]()

sidenver/word2vec_assignment, Word Embedding in Pytorch - GeeksforGeeks. Word2Vec-CBOW-/w2v_cbow.ipynb at master · vlopatenko/Word2Vec-CBOW-, Word2Vec-CBOW-/w2v_cbow.ipynb at master · vlopatenko/Word2Vec-CBOW-.

Task-3 : bentrevett/pytorch-sentiment-analysis: Tutorials on getting started with PyTorch and TorchText for sentiment analysis., Unit 4.3 | Training a Multilayer Perceptron in PyTorch | Part 1.