NLP Assignment 1

Manan Aggarwal 2022273
Shobhit Raj 2022482
Souparno Ghose 2022506

February 2, 2025

Task 1

Pre-processing Applied to the Corpus

Before constructing the vocabulary, the input corpus undergoes several pre-processing steps to ensure that the text is in a normalized format. The following transformations are applied:

1. **Lowercasing**: The entire text is normalized by converting into lowercase. Example: "Apple" and "apple" as the same token.
2. **Removing Non-Alphanumeric Characters**: Any character that is not a alpha numeric is replaced by a space. For example:
   - Input: `"Hello, World!"`
   - After processing: `"hello world "`
3. **Tokenization Based on Spaces**: The text is then split into words using spaces and all the empty strings are removed. This forms an initial set of word-level tokens.

The output of this step is a list of words extracted from the corpus.

Vocabulary Construction

The vocabulary is constructed using a statistical approach based on the WordPiece algorithm. The process follows these steps:

1. **Initialize Vocabulary:**
   - The vocabulary starts with two special tokens: `[UNK]` (unknown token) and `[PAD]` (padding token).
2. **Count Word Frequencies:**
   - A frequency dictionary is created where each unique word in the corpus is counted so that we only tokenize unique words in order to reduce the computational complexity. For example, given the text:

     `"hello world hello"`

     The frequency dictionary is:

     `{"hello": 2, "world": 1}`

3. **Split Words into Character Tokens:**

   - Each word is initially represented as a sequence of characters. The first character remains as is, while subsequent characters are prefixed with ##, indicating they are in the middle of the word and not the starting character.
   - Example:

     `"hello"` $\rightarrow$ `["h", "##e", "##l", "##l", "##o"]`

4. **Count individual and bigram tokens:**

   - We then count all the individual tokens and multiply by the frequency of that word to get the total count of that token. A similar approach is also applied to get the bigram counts.
   - Example:

     `{"hello": 2, "world": 1}` $\rightarrow$

     `{"h":2, "##e":2, "##l":5, "##o":3, "w":1, "##r":1, "##d":1}`

5. **Compute Bigram Scores:**

   - Bigrams (pairs of adjacent tokens) are identified, and their frequency is counted. The importance of a bigram is determined by the following formula:

$$\text{score(bigram)} = \frac{\text{bigram frequency}}{\text{frequency of first token} \times \text{frequency of second token}} \tag{1}$$

   - Example:

     `"hello"` $\rightarrow$ `("h", "##e"), ("##e", "##l"), ("##l", "##l"), ("##l", "##o")`

6. **Merge the Best Bigram:**

   - The highest-scoring bigram is merged into a new token.
   - Example:

     If ("h", "##e") is the best bigram, then we replace occurrences of "h" and "##e" with "he".

     `"hello"` $\rightarrow$ `["he", "##l", "##l", "##o"]`

7. **Repeat Until Vocabulary Size is Reached:**

   - The merging process continues until the vocabulary reaches the predefined size (e.g., 5000 tokens).
   - Each time a new token is added to the vocabulary, all words are updated to reflect the new merged tokens.

Task 2

Training and Validation Loss

The training and validation loss curves over epochs are plotted to analyze the model's convergence. The loss values help in understanding whether the model is overfitting, underfitting, or achieving optimal generalization.

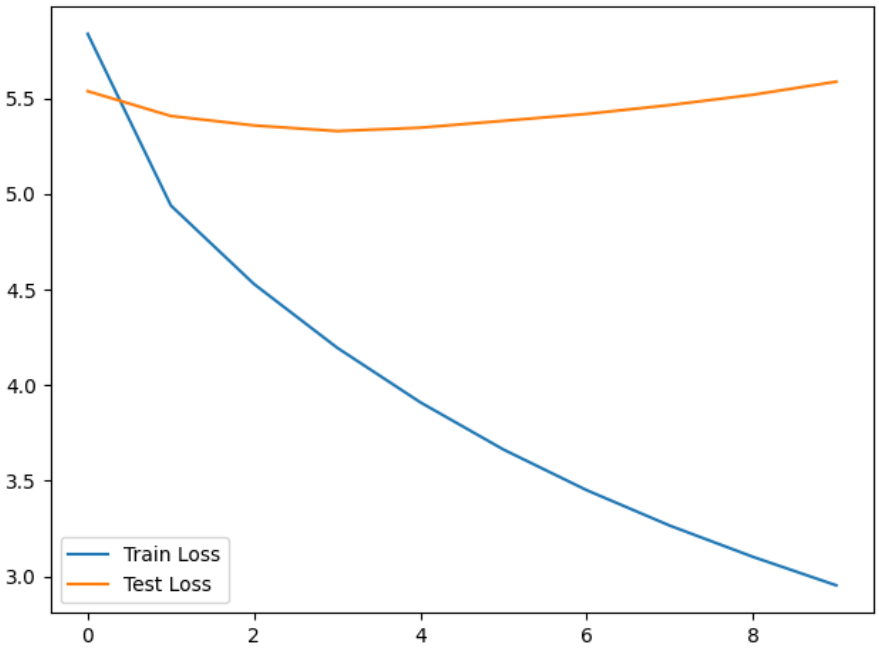

Figure 1: Training and Validation Loss vs. Epochs

The graph in Figure 1 shows the variation of loss with epochs, where the training loss should ideally decrease steadily, and the validation loss should follow a similar trend without significant divergence.

Triplets Identified and Cosine Similarities

The following word triplets were identified along with their respective cosine similarity scores:

Triplet 1: ('see', 'sense', 'illusion')

- **Cosine similarity between see and sense:** 0.846
- **Cosine similarity between see and illusion:** -0.198
- **Cosine similarity between sense and illusion:** -0.170

Explanation: The high similarity score (0.846) between `see` and `sense` suggests a strong semantic relationship, as both words relate to perception. However, the negative similarity between `see` and `illusion` (-0.198) and between `sense` and `illusion` (-0.170) indicates that `illusion` is conceptually opposed to both, as an illusion often distorts or misrepresents what is seen or sensed.

Triplet 2: ('bike', 'wheels', 'fly')

- **Cosine similarity between bike and wheels:** 0.352
- **Cosine similarity between bike and fly:** -0.045
- **Cosine similarity between wheels and fly:** -0.177

Explanation: The moderate similarity (0.352) between `bike` and `wheels` is expected, as wheels are a crucial component of a bike. The negative similarity between `bike` and `fly` (-0.045) and between `wheels` and `fly` (-0.177) suggests that the concept of flying is not strongly related to either bikes or wheels, reinforcing their distinct roles in semantics.

Task 3

Design Choices and Performance Analysis

Training Results

Model 1:

- **Train Accuracy:** 52.98%
- **Train Perplexity:** 24.53
- **Test Accuracy:** 41.92%
- **Test Perplexity:** 109.80

Model 2:

- **Train Accuracy:** 84.58%
- **Train Perplexity:** 5.38
- **Test Accuracy:** 40.43%
- **Test Perplexity:** 509.41

Model 3:

- **Train Accuracy:** 64.06%
- **Train Perplexity:** 14.58
- **Test Accuracy:** 46.13%
- **Test Perplexity:** 91.36

Generated Outputs

Model 1:

i felt like earlier this year i was starting to feel emotional that it charit racist death
i do need constant reminders when i go through lulls in feeling submiss adol inclusiv charit
i was really feeling crappy even after my awesome accidentgaz jour
i finally realise the feeling of being hated and its after effects aregaz jour yar
i am feeling unhappy and weird racist subconscious gho

Model 2:

i felt like earlier this year i was starting to feel emotional that it training racist racist
i do need constant reminders when i go through lulls in feeling submiss earthqua assured od
i was really feeling crappy even after my awesome ghogaz training
i finally realise the feeling of being hated and its after effects are inclusivrytim racist
i am feeling unhappy and weird racist subar inclusiv

Model 3:

i felt like earlier this year i was starting to feel emotional that it inclusivrytim racist
i do need constant reminders when i go through lulls in feeling submiss gho schindlersid
i was really feeling crappy even after my awesomeympathizing racist inclusiv
i finally realise the feeling of being hated and its after effects arecoming racist racist
i am feeling unhappy and weird gho simcoming

Design Choices and Performance Analysis

Model Architectures

We implemented three different neural language models (NLMs) to analyze the impact of architectural choices on performance. These models, **NeuralLM1, NeuralLM2, and NeuralLM3**, were evaluated based on accuracy and perplexity on both training and validation datasets.

NeuralLM1: Basic Feedforward Network

Architecture:

- A single hidden layer with a `Tanh` activation function.
- The input consists of word embeddings concatenated from the context window.
- The output is a softmax layer over the vocabulary size.

Rationale: This simple architecture allows for fast training and serves as a baseline model. The use of `Tanh` helps with smooth gradient flow while maintaining non-linearity.

Impact: The model captures basic semantic relationships but lacks depth, leading to limited generalization.

NeuralLM2: Feedforward with Residual Connection

Architecture:

- A deeper feedforward network with an additional hidden layer.
- Uses both `ReLU` and `Tanh` activation functions.
- Implements a residual connection to improve gradient flow.

Rationale: The additional hidden layer captures more complex dependencies. The residual connection helps mitigate vanishing gradients, leading to better representation learning.

Impact: Despite a deeper architecture, the model underperforms compared to NeuralLM1. The high perplexity suggests convergence issues.

NeuralLM3:LSTM with Residual Connections

Architecture:

- A LSTM layer to capture sequential dependencies.
- Three fully connected layers with non-linear activations.
- Residual connections for better information flow.

Rationale: Unlike the previous architectures, LSTMs are better suited for capturing sequential dependencies.

Impact: This model outperforms both previous architectures, achieving the best accuracy and lowest perplexity.

Loss Trends

To analyze the models further, we plot the training and validation loss trends across epochs.

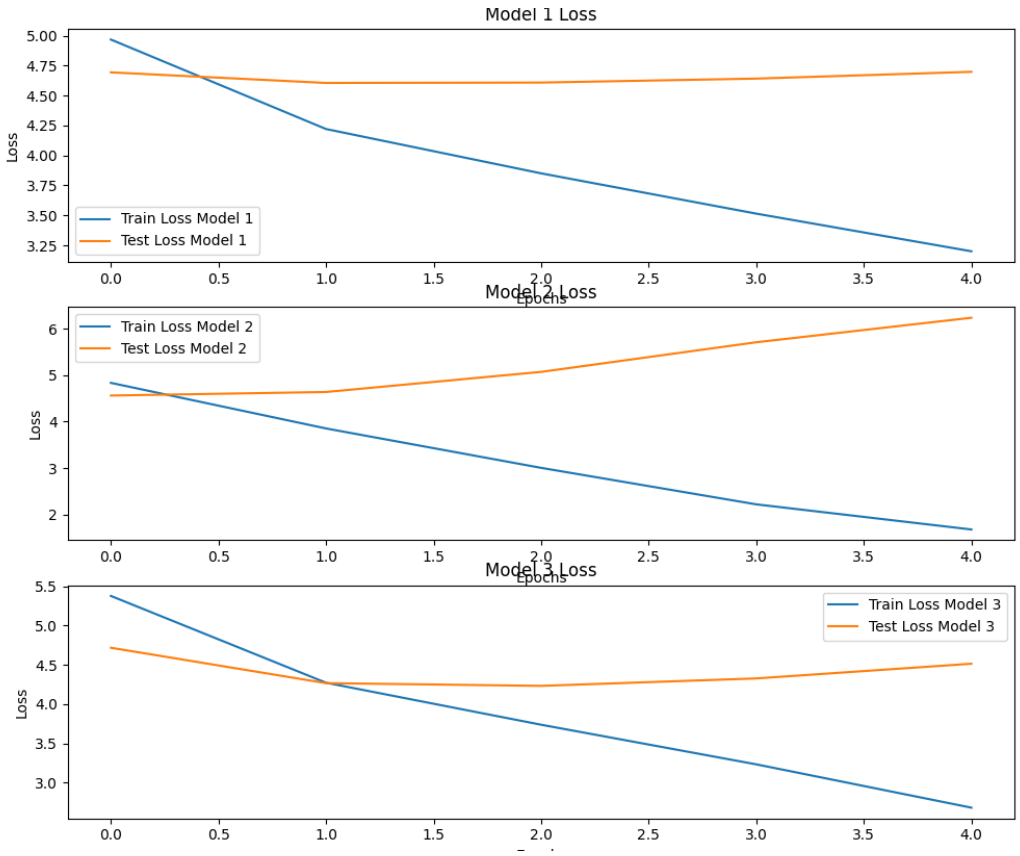

Figure 2: Training and Validation Loss vs Epochs for NeuralLM1, NeuralLM2, and NeuralLM3.

Comparison and Discussion

- **NeuralLM1** performs reasonably well for a simple model but lacks depth.
- **NeuralLM2** introduces depth but suffers from high perplexity.
- **NeuralLM3** achieves the best results by leveraging bidirectional LSTMs.

Conclusion

Among the three models, **NeuralLM3** is the most effective due to its ability to handle sequential dependencies efficiently. Future improvements could include pretraining embeddings or using transformer-based architectures.

Individual Contribution

- **Task 1:** Shobhit Raj
- **Task 2:** Manan Aggarwal
- **Task 3:** Souparno Ghose
- **Report Writing:** Everyone contributed equally

References

1. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781.* Retrieved from `https://arxiv.org/abs/1301.3781`

2. Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research, 3*, 1137-1155. Retrieved from `https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf`

3. Hugging Face. (n.d.). Chapter 6: Attention mechanisms. Retrieved from `http://huggingface.co/learn/nlp-course/en/chapter6/6`

4. PyTorch Documentation. (n.d.). `torch.nn.LSTM`. Retrieved from `https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html`

5. PyTorch Documentation. (n.d.). `torch.nn.Embedding`. Retrieved from `https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html`