AMS 691.02: Natural Language Processing
Fall 2024 Assignment 2

Rakesh Jeyachandran Vinotha
115734472

1. Neural Networks for Part-of-Speech Tagging:

1.1 Baseline Model:

Developed a baseline feed-forward neural network (FFNN) for part-of-speech (POS) tagging, using both the center word and surrounding context to classify POS tags. Experimented with context window sizes of *w* = 0 (center word only) and *w* = 1 center word with one word on either side). The model was trained on annotated tweet datasets. For *w* = 0, the center word alone was used, while *w* = 1 included the center word plus neighboring words, creating a three-word context. These variations allowed us to explore the effect of context on tagging accuracy.

The architecture consisted of an input layer, i.e., concatenated word embeddings based on context window. A hidden layer, a single layer with 128 units, and a tanh activation function. And output Layer, a softmax-transformed score for each POS tag. All parameters, including word embeddings, were randomly initialized and trained with cross-entropy loss and SGD optimization.

The model was evaluated on the DEVTEST set, and tagging accuracies were recorded for both context configurations:

1. Accuracy for w = 0: 77.65%
2. Accuracy for w = 1: 81.45%

The results indicate that including a context window of size 1 (with one word on either side of the center word) led to a significant improvement in accuracy, from 77.60% to 80.77%. This improvement suggests that additional contextual information allows the model to better capture syntactic patterns that influence POS tagging. For instance, words often have different POS tags depending on their surrounding words, and the inclusion of context provides the model with this relational information, resulting in more accurate predictions.

1.2 Feature Engineering

The selected features include capitalization: a binary feature indicating if the center word starts with an uppercase letter. This feature can help identify proper nouns, which often start with capital letters, and suffixes: binary indicators for common suffixes like “-ing,” “-ed,” and “-ly.” These suffixes frequently signal specific POS tags, such as verbs (e.g., "-ing" for present continuous tense) or adverbs (e.g., "-ly"). And special characters: A binary feature indicating the presence of special characters (e.g., "!", "@"). Special characters can indicate emphasis or informal POS usages common in tweets, like interjections or abbreviations.

With the inclusion of these features, the model achieved an accuracy of **81.05% for** w **= 0** and **82.97% for** w **= 1** on the DEVTEST set. These results show a noticeable improvement over the baseline, underscoring the importance of these additional features in enhancing the model's contextual understanding. The use of engineered features, combined with word embeddings, enabled the model to capture syntactic and stylistic cues that are beneficial for accurate POS tagging. This demonstrates that carefully selected linguistic features can significantly boost the performance of neural network-based POS tagging models.

1.3: Pretrained Embeddings:

1.3.1 Updating (fine-tuning) the pre-trained embeddings:

Loaded pre-trained embeddings and incorporated them into the model's input layer. The embeddings were then fine-tuned alongside other model parameters, allowing the model to retain valuable pre-learned language patterns while adapting the embeddings to the POS tagging task's requirements. This approach aimed to benefit from the semantic nuances present in the pre-trained embeddings, which were derived from a large dataset.

Results: Using pre-trained embeddings and fine-tuning, the model achieved the following accuracies on the DEVTEST set:

1. w = 0: 77.38%
2. w = 1: 82.59%

These results demonstrate a noticeable improvement over the baseline accuracies from Task 1.1, where we achieved 77.65% and 81.44% for w = 0 and w = 1, respectively. The increase in accuracy is attributed to the model's ability to start with contextualized word representations, which provides a more robust starting point for training. The improved accuracies highlight the benefits of using pre-trained embeddings, especially in tasks where labeled data is limited. The trained embeddings capture semantic relationships and word usage patterns that are difficult to learn from scratch in small datasets. The subsequent fine-tuning process allows the model to adapt these embeddings specifically for POS tagging, refining the representations to better distinguish between different parts of speech

1.3.2 Compare updating the pre-trained word embeddings and keeping them fixed:

For w=1, Fine-tuned embeddings achieved an accuracy of 82.59%. While Fixed embeddings (no updating during training) achieved an accuracy of 81.03%. The results indicate that fine-tuning the pre-trained embeddings led to a modest improvement in accuracy. Fine-tuning allowed the model to adapt these embeddings to the nuances of POS tagging, refining the representations to capture specific syntactic cues. In contrast, keeping embeddings fixed limited the model's ability to optimize representations for the tagging context, resulting in slightly lower accuracy.

While fixed embeddings still leveraged valuable pre-trained information, fine-tuning provided an edge by allowing for task-specific adaptation. This comparison highlights the advantage of embedding updates in tasks where subtle language distinctions, like POS tagging, benefit from additional contextual learning.

1.3.3 Features with the use of pre-trained embeddings:

The model achieved the following accuracies on the DEVTEST set, w = 0: 78.70%, w = 1: 83.23%. These results indicate a slight improvement over task 1.2 (which achieved 81.05% and 82.97% for $w = 0$ and $w = 1$, respectively) for w=1 but not for w=0. The increase, while modest, suggests that the additional features contributed marginally to the model's ability to distinguish POS tags more effectively.

The combined approach of leveraging pre-trained embeddings with feature engineering showed marginal gains, underscoring that pre-trained embeddings already capture much contextual information. Nonetheless, the engineered features provided subtle, additional cues that enhanced the model's tagging performance.

1.4 Architectural Engineering:

1.4.1 Compare the use of 0, 1, and 2 hidden layers:

Experimented with 0, 1, and 2 hidden layers and tested two different widths for each configuration, specifically using sizes of 256 and 512 units. This exploration aimed to identify how adding depth and varying the width of hidden layers affects the model's performance.

Results and Observations:

1. 0 Hidden Layers: This configuration yielded a DEVTEST accuracy of 84.22%, the highest among all setups. The absence of hidden layers suggests a simpler model that directly maps inputs to outputs, which seems beneficial when overfitting is a risk, as in cases with limited data or simpler patterns.
2. 1 Hidden Layer (256 and 512 units): With one hidden layer, the DEVTEST accuracy reached 83.21% for 256 units and 82.63% for 512 units. While this configuration still performed well, increasing the hidden layer size showed a slight decrease in accuracy, possibly due to overfitting or the model capturing redundant patterns that didn't generalize well to the DEVTEST data.
3. 2 Hidden Layers (various combinations of 256 and 512 units): This setup produced the lowest accuracies, with DEVTEST results ranging from 81.74% to 82.88%. The

additional complexity introduced by the second hidden layer seems to have hindered performance, likely because the model overfits the training data. Additionally, the model may have struggled to generalize with a smaller dataset, making additional layers counterproductive.

Overall, the results indicate that a simpler architecture with 0 hidden layers was most effective, achieving the highest accuracy. Increasing the number of hidden layers and their width led to diminishing returns or reduced accuracy, likely due to overfitting. These findings highlight the importance of choosing an architecture that balances model complexity with the dataset's size and task requirements.

1.4.2: Experiment with different nonlinearities:

The model architecture used was 1 hidden layer with 128 units. By experimenting with different nonlinearities—identity (g(a) = a), ReLU, logistic sigmoid, and tanh—we aimed to assess how each function affects the model's capacity to capture patterns in the data and influence final tagging performance.

The model achieved the following DEVTEST accuracies with each activation function:

1. Identity Activation: 82.54%
2. ReLU Activation: 82.73%
3. Sigmoid Activation: 82.99%
4. Tanh Activation (baseline): 83.08%

Identity Activation (no nonlinearity): The model performed moderately well with an accuracy of 82.54%, indicating that even without nonlinearity, the model could capture some patterns. However, the lack of nonlinearity limits the model's ability to handle complex patterns, slightly impacting performance.

ReLU Activation: With ReLU, the model achieved a DEVTEST accuracy of 82.73%. ReLU's tendency to handle positive values while nullifying negatives led to decent performance, but its behavior may have limited the model's robustness for this tagging task, where more balanced activations like tanh can yield better results.

Sigmoid Activation: Sigmoid achieved an accuracy of 82.99%. While it outperformed ReLU and identity, the slower convergence associated with the sigmoid activation's saturation effect at extremes likely hindered its effectiveness slightly compared to tanh.

Tanh Activation (baseline): The tanh activation, used in our initial experiments, resulted in the highest accuracy at 83.08%. Tanh's symmetric range (-1 to 1) and balanced output across positive and negative values made it particularly suitable for capturing the POS tagging task's nuances.

The tanh activation provided the best performance among the tested functions, confirming its suitability for POS tagging where balanced activation across inputs is beneficial. Sigmoid, ReLU, and identity activations also performed reasonably well but showed slight limitations in capturing the tagging-specific patterns, highlighting the importance of selecting a suitable activation function for optimal model performance.

1.4.3 Experiment with w = 2 and compare the results to w = 0 and 1:

Explored the effect of increasing the context window size on POS tagging accuracy. Specifically, we expanded the context window to w = 2, which considers the two preceding and two following words around the center word. This configuration was then compared with the results from w = 0 (center word only) and w = 1 (one word on either side of the center word) to evaluate how much additional context benefits tagging accuracy.

1. w = 0 (center word only): Final DEVTEST Accuracy: 78.70%
2. w = 1 (one word on each side): Final DEVTEST Accuracy: 83.23%
3. w = 2 (two words on each side): Final DEVTEST Accuracy: 83.47%

The results show a clear improvement in accuracy when moving from w = 0 to w = 1, as the model benefits from additional context that helps capture surrounding syntactic cues. When the context window increased from w = 1 to w = 2, the model accuracy showed only a marginal improvement, suggesting that the added context provides diminishing returns. The slight gain in accuracy from w = 1 to w = 2 may be due to some additional useful context words, but it appears that one neighboring word on each side is generally sufficient for capturing the patterns relevant to POS tagging.

Using a larger context window of w = 1 provided significant accuracy gains over w = 0, highlighting the importance of immediate contextual information for POS tagging. However, increasing to w = 2 resulted in only a minor improvement, indicating that too much context may not substantially enhance model performance for this task and could lead to unnecessary complexity. Thus, w = 1 strikes an effective balance between context richness and model efficiency.

1.5 RNN Taggers:

Implemented and compared various Recurrent Neural Network (RNN) architectures for POS tagging. The goal was to test the effectiveness of different RNN models (standard RNN, LSTM, and GRU) and compare them with their bidirectional versions to identify which architecture best captures the temporal dependencies required for POS tagging. The evaluation was based on final DEVTEST accuracies.

Results Summary:

1. Standard RNN: 82.97%
2. Bidirectional RNN: 83.66%
3. LSTM: 79.54%
4. Bidirectional LSTM: 80.13%
5. GRU: 82.09%
6. Bidirectional GRU: 83.16%

- Standard RNN vs. Bidirectional RNN:
    - The standard RNN achieved a final DEVTEST accuracy of 82.97%, while the bidirectional RNN improved to 83.66%. The bidirectional RNN captures both past and future context, which is advantageous for POS tagging tasks where the surrounding words influence the tag of a word. This bidirectional approach allows the model to consider more nuanced patterns, contributing to more accurate tagging.
- LSTM and Bidirectional LSTM:
    - The LSTM model achieved a final DEVTEST accuracy of 79.54%, and the bidirectional LSTM improved this to 80.13%. LSTMs are specifically designed to address the vanishing gradient problem in standard RNNs, making them suitable for longer sequences. However, the moderate performance of both LSTM configurations suggests that, for POS tagging, the relatively short sentence structure may not require the LSTM's long-term memory capabilities as extensively as in tasks like machine translation or text generation.
- GRU and Bidirectional GRU:
    - The GRU achieved a DEVTEST accuracy of 82.09%, and the bidirectional GRU reached 83.16%. GRUs, being simpler than LSTMs, often perform comparably while training faster. The bidirectional GRU's performance close to that of the bidirectional RNN indicates that GRUs effectively capture context-dependent information while requiring fewer resources and less training time than LSTMs.

Among all tested models, the bidirectional RNN yielded the best performance, with an accuracy of 83.66%. The bidirectional RNN's simplicity and ability to capture both past and future context proved to be highly effective for POS tagging.

The bidirectional GRU also performed well, indicating that GRUs provide a balance of efficiency and accuracy for tasks like POS tagging, where the context window is relatively limited. The lower performance of LSTM models in this context reflects that their complexity may not translate into significant accuracy gains for shorter, structured tasks like POS tagging.

Bidirectional models outperformed their unidirectional counterparts, underscoring the value of future context in POS tagging. The bidirectional RNN yielded the highest accuracy, showing that simpler architectures can be highly effective when both past and future contexts are captured. The GRU and bidirectional GRU also demonstrated strong performance, suggesting they are efficient alternatives for capturing sequence-based dependencies in POS tagging tasks.

2. Language Modeling:

Language modeling is a fundamental task in natural language processing that involves estimating the probability of a sequence of words in a language. The goal of a language model is to assign a probability to a sequence or sentence, allowing us to predict the likelihood of specific word sequences appearing together. Language models are widely used in applications such as machine translation, speech recognition, and predictive text input.

One common approach to language modeling is to break down a sentence into bigrams, which are pairs of consecutive words. By modeling sequences as bigrams, we simplify the prediction of a word based only on the previous word, which is more computationally manageable and effective for many applications. For example, in the sentence "the cat runs," the bigrams are <s> the, the cat, cat runs, and runs </s>, where <s> denotes the start of a sentence and </s> marks its end.

Two common approaches are:

Unsmoothed Model (Model U): Uses raw counts from the training data

- P(w2|w1) = count(w1,w2) / count(w1)
- Assigns zero probability to unseen bigrams
- Generally assigns higher probabilities to observed sequences

Add-One Smoothed Model (Model S): Adds 1 to all counts before normalizing

- P(w2|w1) = (count(w1,w2) + 1) / (count(w1) + |V|)
- Ensures non-zero probabilities for all possible bigrams
- Usually assigns lower probabilities to observed sequences due to the probability mass redistribution

While Model U typically assigns higher probabilities to observed sequences than Model S, we can construct a dataset where this is not always true. Consider the following dataset:

Dataset D:

<s> a b </s> (occurs 1 time)

<s> b b </s> (occurs 1 time)

<s> b c </s> (occurs 1 time)

Vocabulary V = {a, b, c}

Bigram Probabilities:

Model U (Unsmoothed):

$P(a|<s>) = 1/3 \approx 0.333$
$P(b|<s>) = 2/3 \approx 0.667$
$P(b|a) = 1/1 = 1.000$
$P(b|b) = 1/2 = 0.500$
$P(c|b) = 1/2 = 0.500$
$P(</s>|b) = 1/3 \approx 0.333$
$P(</s>|c) = 1/1 = 1.000$

Model S (Smoothed):

$P(a|<s>) = 2/6 \approx 0.333$
$P(b|<s>) = 3/6 = 0.500$
$P(b|a) = 2/4 = 0.500$
$P(b|b) = 2/5 = 0.400$
$P(c|b) = 2/5 = 0.400$
$P(</s>|b) = 2/6 \approx 0.333$
$P(</s>|c) = 2/4 = 0.500$

Results for the sentence "<s> a b </s>":

- Model U probability: $0.333 \times 1.000 \times 0.333 \approx 0.111$
- Model S probability: $0.333 \times 0.500 \times 0.333 \approx 0.555$

This dataset demonstrates that Model S can assign a higher probability than Model U to an observed sequence. This occurs because the small dataset creates extreme probabilities in Model U (e.g., P(b|a) = 1) and Add-one smoothing moderates these extreme values. The combination of smoothed probabilities results in a higher overall sentence probability for "<s> a b </s>" in Model S. This example challenges the intuition that smoothing always reduces probabilities of observed events and demonstrates how probability redistribution can sometimes lead to higher probabilities for certain sequences.