Trexquant Hangman Challenge

Submitted By: Mathew Manoj
Role: Global Alpha Researcher Candidate
Subject: Code Explanation and Methods Utilization

Overview:

In this challenge, the goal is to develop an algorithm that plays the game of Hangman via an API server. After exploring various approaches, I chose to use a Bidirectional LSTM Recurrent Neural Network (RNN) model because it can predict letters by analyzing both the forward and backward context of the word. This document explains my reasoning for selecting this model, the inspiration from other solutions, and the preprocessing techniques used.

Model Selection:

Why Bidirectional LSTM?

For predicting letters based on incomplete words, I considered different models like Transformers, basic LSTMs, and GRUs. Here's why I decided on the Bidirectional LSTM RNN:

- **Avoiding Overkill with Transformers**: While Transformers are great for long-term dependencies, they are unnecessarily complex for Hangman, where the task is simply to predict letters in relatively short words (up to 29 characters). I preferred a model that balances simplicity with performance.
- **Limitations of Basic LSTMs**: LSTMs are good for sequence prediction, but they can suffer from the vanishing gradient problem. Over time, they tend to forget patterns in large datasets. For Hangman, where understanding both the start and end of a word is important, basic LSTMs may not perform well.
- **Advantages of Bidirectional LSTMs**: This model analyzes the word from both directions—forward and backward—helping to make more accurate guesses. In Hangman, this dual perspective is useful, especially when letters are known in different parts of the word (e.g., "_ p _ l e"). It allows for better pattern recognition and prediction compared to other models.
- **Choosing Bidirectional LSTM Over GRU**: While GRUs converge faster due to fewer parameters, they may not capture the intricate patterns in letter sequences that LSTMs can. For this challenge, I valued the richer context analysis provided by Bidirectional LSTMs over the speed offered by GRUs.

Preprocessing Strategy:

Inspiration and Modifications

I took inspiration from a preprocessing technique I found in an existing solution (from this link), which helped me handle variable-length sequences efficiently. Here's what I applied:

- **Max Word Length Padding**: The longest word in the dictionary is 29 characters, so I padded all input words to this length to maintain consistent input size for the model. This ensures that variable-length words are processed uniformly, which is essential for feeding data into a neural network.
- **Tokenization and Mask Handling**: Each masked word (e.g., "_ p _ l e") was tokenized in a way that the model could process. I used padding to ensure all words were of the same length, and treated underscores as special tokens, helping the model focus on predicting missing letters accurately.
- **Pattern Matching and Frequency Analysis**: I employed a pattern-matching strategy based on partial knowledge of the word. By narrowing the dictionary to words that fit the current pattern and analyzing letter frequencies, I helped the model make more informed guesses. This step reduces the search space and increases the efficiency of predictions.
- **Preprocessing Insights**: Drawing from the statistical methods found in the solution mentioned above, I combined neural network-based guessing with frequency analysis. This hybrid approach allowed my model to make smarter guesses early on, reducing incorrect attempts.

Approach Summary:

The core of my solution is the Bidirectional LSTM RNN, which predicts letters by considering both the past and future context of words. The preprocessing strategy ensures the model has structured, consistent input to work with, improving the accuracy of letter predictions. By integrating frequency analysis and pattern matching, my approach outperforms the benchmark strategy, achieving higher accuracy in predicting the correct letters within the six allowed incorrect guesses.