Trexquant hangman challenge

The code combines a Trie based data structure to predict words based on the given words dataset along with a LSTM trained to predict next letter by a probability distribution as output. The Trie based data structure is used when more than 40% of the word is not filled yet and after that control is given to the LSTM. This method was able to achieve a accuracy of 55.9% which is already way higher than the traditional benchmark of 18% accuracy. Below is an explanation of the two models.

Trie based model:

The implemented Trie-based system efficiently supports Hangman-style word prediction by leveraging a depth-first search (DFS) approach to generate possible word matches based on a partial word pattern. The system constructs a Trie where each node represents a letter, and words are added incrementally by linking letters in sequence. By utilizing a wildcard pattern (_) for unknown letters, the algorithm dynamically explores all valid word completions stored in the Trie. Additionally, the system predicts the next best letter to guess by analyzing the frequency of unguessed letters in the set of possible words. This method optimizes the guessing process in word games, offering a data-driven approach to letter prediction.

LSTM based model:

First the words in the dictionary are encoded into numerical format. Then each sequence of words is masked randomly to generate a training dataset with input as the masked word and output as the index of the letter being masked. After generating all training examples, the function ensures that all sequences (input words) are of the same length It pads the sequences to a uniform length (`self.max_length`) using `pad_sequences` from Keras. Padding is done at the end of the sequence to ensure all inputs are of the same size. The data is then split into training and testing sets. The model is built using Keras' **Sequential API.**

Here's a breakdown of the architecture:

- **model.add(Embedding(input_dim=28, output_dim=128, input_length=self.max_length, trainable=True))**
- **5 stacked Bidirectional(LSTM(64, return_sequences=True)) layers.**
- **model.add(Dropout(0.5) for regularization**
- **model.add(Dense(26, activation='softmax')) output layer.**

This model is then trained for 10 epochs with train-test split of 0.9-0.1 and batch size of 3k.

This combination of Trie and LSTM based model is good in predicting the letters in the hangman game.