Objective

The objective of this task is to develop a model capable of playing the Hangman game with an accuracy exceeding 60%, while limiting the number of incorrect guesses to a maximum of six.

Approach Overview

The solution focused on leveraging an encoder-based architecture in conjunction with a two-phase training process. A masked language modeling technique was used to enable the model to predict letters intelligently by considering both previously guessed letters and the incomplete word.

1. Encoder Model Training

The core of the solution was an encoder model, built using an architecture from scratch using PyTorch. This model was specifically optimised to process partially revealed words, along with a set of already guessed characters, in order to predict the next most probable letter.

- Input Structure:The input consisted of two components:
  - The masked (partially revealed) word.
  - The guessed characters so far.

2. Two-Stage Training Process

Stage 1: Training on Randomly Generated Data

In the initial phase, the model was trained on synthetic data that mimicked real-world gameplay. Two strategies were employed to simulate the guessing process:

- 40% Probability: The model guessed a character from the correct set of letters (i.e., letters that were part of the word).

- 60% Probability: The model guessed from the complete set of possible characters.

In both cases, guessed characters were excluded from further guesses, ensuring that each new guess was unique. This balanced strategy allowed the model to learn both correct and incorrect guesses, which reflects real-world gameplay dynamics.

Stage 2: Self-Play Fine-Tuning

After the initial training, the model was fine-tuned through self-play, where it played the Hangman game against itself. This allowed the model to refine its strategy based on the outcomes of its own guesses.

The self-play process generated additional training data, which was then used to retrain the model. This cycle continued until no further improvements were observed in the training accuracy.

During this phase, the model progressively optimised its ability to predict characters, reducing errors and improving accuracy over time.

3. Initial Guess Strategy

Since all characters in the word are masked at the beginning of the game, the first few guesses rely on a frequency-based heuristic instead of the model's predictions. This is especially useful until the model receives enough context (through a correct guess) to predict intelligently.

1. Frequency Mapping for Word Lengths:
    - During the training phase, a frequency mapping was created that associated the most common letters with specific word lengths. For example, for words with 5 letters, certain characters appear more frequently than others.
2. First Guess Based on Frequency:

- At the start of the game, when no correct guesses have been made, the model selects characters based on the frequency mapping corresponding to the word's length.
- If the most frequent character guess is incorrect, the next most frequent character for that word length is selected, and so on.
- This process continues until the first correct character is guessed or the wrong guess limit is exceeded.

3. Switch to Model-Based Guessing:
    - Once the first correct character is identified, the model begins to make predictions based on the partially revealed word and previously guessed characters, using its learned masked language model approach.

This initial frequency-based guessing strategy helps to mitigate the cold-start problem that could arise when the model has no prior information to base its predictions on.

4. Masked Language Modelling

A masked language modelling approach was adopted to facilitate intelligent letter prediction. The input format for the model was:
[CLS] masked_word [SEP] guessed_characters

Here’s an example of how the input format `[CLS] masked_word [SEP] guessed_characters` would look during a game of Hangman:

- **Word to guess**: *apple*

- **Current state of the word**: `_ _ _ l e`

- **Guessed characters so far**: `a, b, c, e, l`

The input to the model would be formatted as:

[CLS] _ _ _ l e [SEP] a b c e l

The model would then use this input to predict the next best character to guess.

- The model predicted the missing letters in the masked word, while considering the constraints imposed by already guessed characters. This enabled the model to prioritise valid, unguessed letters in subsequent predictions.

5. Prediction Process

During the prediction phase, the model outputs a probability distribution over all possible characters for each position in the masked word. The process is as follows:

1. Character Probability Distribution:
    - For each position in the word, the model predicts a probability distribution over the possible characters.
2. Selecting the Next Guess:
    - The maximum logits for each character across all positions in the word is calculated via max pooling.
    - Before selecting the next best guess, the logits values corresponding to already guessed characters are replaced with -infinity. This ensures that the model does not select a previously guessed character.
    - The logits are then passed through softmax.
    - The character with the highest overall probability is chosen as the next best guess.

This method ensures that the model intelligently chooses the next character based on both the masked word and the already guessed characters, maximising the chances of selecting a correct letter.

6. Loss Functions

Two loss functions were used to optimise the model's performance:

1. Cross-Entropy Loss: This loss function was used to enhance the model’s ability to predict the next best letter with high accuracy.
2. Similarity Score Loss: This additional loss was introduced to ensure that the model avoided assigning higher logits to letters that had already been guessed. It helped increase the model's efficiency by penalising repeated guesses.

7. Experiments

Several experiments were conducted to optimise the model's performance. Initially, only the masked word was passed as input, but this approach yielded an accuracy of nearly 50%. To improve results, the guessed characters were included along with the masked word, providing the model with additional context about previous guesses. This adjustment increased accuracy to approximately 53%.

Next, an additional loss function based on cosine similarity was introduced, further boosting accuracy to 55%. During these experiments, different embedding sizes, specifically 64 and 128 dimensions, were tested while keeping the overall architecture unchanged.

In the final phase, both models were fine-tuned using the self-play approach. The model with a 64-dimensional embedding reached 58% accuracy, while the model with a 128-dimensional embedding achieved 63% accuracy on a test set of 10,000 unseen words.