Hangman Letter Prediction using BERT + Transformer Encoder

Aditya Singh

1 Problem Formulation

The Hangman task is formulated as a **multi-class classification problem**. Given a partially revealed word (e.g., `"_e_l_ Guessed: ae"`), the model must predict the next best character to guess from the English alphabet (26 classes: `a` to `z`).

2 Methodology

2.1 Data Sampling Strategy

The dataset is generated by applying augmentations to a list of English words. For each word, multiple masked variants are created:

- **Endgame Augmentations:** I simulate late-stage Hangman scenarios by masking one or two randomly chosen characters from the set of unique letters in the word. These settings reflect situations where the player is close to completing the word.
- **Random Augmentations:** To diversify the training data, I randomly mask 1 to 4 unique characters in the word. For each masked variant, I generate samples for each of the masked characters.

Each sample includes:

- `masked`: the masked version of the word
- `guessed`: letters already revealed
- `label`: the correct character to guess next

2.2 Input Representation and Tokenization

Each input sample to the model represents the current state of a Hangman game and consists of two parts:

- A masked version of the target word, with unguessed characters replaced by underscores.
- A list of previously guessed characters.

These are combined into a single input string of the form:

```
"_a_a_ Guessed: st"
```

This string is then tokenized using the BERT tokenizer from the `transformers` library. The tokenization process includes:

- Converting the string to a sequence of subword tokens.
- Truncating the sequence to a maximum length (e.g., 32 tokens).
- Padding the sequence to ensure consistent input size.

This tokenized input is fed into the BERT model, allowing it to extract semantic and positional features that capture both the structure of the partially revealed word and the context of previously guessed letters.

3 Model Architecture

3.1 Overview

The model consists of two key components:

1. A pre-trained BERT encoder (`bert-base-uncased`) to process the masked input sentence.
2. A Transformer encoder stack to model deeper token interactions.
3. A deep feedforward classifier that maps the [**CLS**] token embedding to a 26-way character prediction (a–z).

3.2 Why BERT?

BERT is a bi-directional language model that captures rich contextual representations of text. It is well-suited for this task because:

- It handles incomplete/masked sequences well due to its training objective.
- It models contextual dependencies, which is critical in predicting likely characters based on partial words.
- Pre-training on large corpora provides strong generalization even with limited task-specific data.

3.3 Why Transformer on top of BERT?

Although BERT already applies self-attention, an additional Transformer encoder helps in several ways:

- Refines token representations further, specifically in the Hangman context.
- Enables deeper interaction across tokens beyond what BERT alone might achieve for short sequences.
- Serves as a learnable adapter layer tailored to this specific task.

3.4 Classifier Head

The classifier consists of a sequence of fully connected layers:

- **Layer 1:** Linear(768, 512) + GELU + LayerNorm + Dropout(0.3)
- **Layer 2:** Linear(512, 256) + GELU + LayerNorm + Dropout(0.2)
- **Layer 3:** Linear(256, 128) + GELU + LayerNorm + Dropout(0.1)
- **Output:** Linear(128, 26)

The progression of layers helps compress and refine the representation, introducing non-linearity and regularization to prevent overfitting.

4 Training and Evaluation

4.1 Training

The model is trained using CrossEntropyLoss and the AdamW optimizer. Each sample corresponds to a single character prediction. I use gradient checkpointing in BERT to reduce memory overhead. During training, the model learns to map the masked and guessed state to the correct next character.

4.2 Evaluation

Evaluation is conducted by simulating a real Hangman game:

- The model plays each word from scratch.
- At each step, it selects the most probable next character (ignoring already guessed ones).
- The game proceeds until the word is guessed or maximum attempts (6) are exhausted.

Accuracy is reported as the win rate across validation words.

5 Future Work

Several improvements can be explored:

- **Character-aware embeddings:** Use CharBERT or Charformer to improve modeling of subword/character structure.
- **Reinforcement Learning:** Fine-tune the agent using win rate as a reward signal.
- **Curriculum Learning:** Train first on short words and gradually increase difficulty.