Hangman Solution Description

To solve the Hangman problem, I approached the problem with a few approaches, the classical n-gram matching approach, neural networks, LSTM, Bi-LSTM, and transformers. In order to test the efficacy of all the approaches I divided my dataset (words_250000.txt) into 2 parts (90% training, 10% validation). These approaches could not achieve the desired performance level, however, while using transformers I realised that the Hangman game seems like the perfect use-case for BERT, since BERT was also trained to predict masked words in a sentence.

I tried several approaches with the BERT and Canine-S architecture. Canine-S model is similar to BERT but instead of word embeddings it was trained on character level embeddings. I did 16-20 experiments with different losses, architectures and training setups:

- Cross Entropy loss over all masked characters
- Cross Entropy loss over the mean of the output logits of all masked characters (mean pooling)
- Cross Entropy loss over the maximum of the output logits of all masked characters (max pooling)
- KL Divergence loss
- Focal Loss - this basically assigns a higher loss to infrequent characters such as ‘q’, ‘x’ and lower loss to frequent characters like ‘a’ and ‘e’. The frequency was calculated using the training set.
- **Cross Entropy loss combined with n-gram prediction when the word is almost predicted (3 characters or less remaining to be predicted)**
- Reinforcement Learning with BERT model as the policy
- Reinforcement Learning with BERT model as the policy and custom rewards depending on the character frequency (higher reward for infrequent characters)

In my report, I will talk about the highlighted approach which gave me the best results.

I created my training dataset by encoding each letter in the word and then replacing 30-80% of the letters with mask token id (103 in BERT). The characters that were not masked were added to the guessed vector and 0-4 other random characters not present in the word were chosen to be added to the guessed vector to increase variability in the training setup.

For training purposes, I used cross entropy loss. Input provided to the model is the encoded masked word and the vector of previously guessed characters. Since BERT requires labels for each letter in the input, the output label is the completely unmasked word with a slight change. I did not want the loss to be affected by the characters that had already been guessed so for the characters that were already correctly guessed (not masked in the training setup) the corresponding label was set to -100. I passed token id -100 to the cross entropy loss function to ensure that those output labels are not considered for loss computation.

My training setup included finetuning the last 2 encoder layers of the BERT model with a learning rate of 3e-5, a cosine learning rate scheduler with a warmup of 10%, AdamW optimizer with a weight decay of 0.01. The guessed vector was provided to the BERT model in the beginning to ensure that the model learns to negate the effects of those characters

while predicting the next character. In the last step of prediction, the logits associated with the guessed characters are set to an extremely large negative number (-1e9) so that the probability assigned to that character is essentially 0.

When testing this approach I achieved an accuracy of 61% accuracy in winning the hangman game with words from my validation set. Upon initial analysis the model did not perform well with words involving the character ‘q’ and failed even with simple words like ‘queue’ and ‘squid’, highlighting the importance of training dataset quality in training BERT. I tried different approaches to counter that problem such as focal loss and Custom Reward Reinforcement Learning setup, however they did not result in significantly better results.

For further performance improvement I decided to combine the n-gram approach with the BERT approach during evaluation.I created three different dictionaries, one trigram dictionary and two bigrams dictionary. During the last stages of evaluation, when there are only three or less characters remaining to be predicted, we consider the letter to the left and right of each masked word and get the probability of each character from our dictionary. The reason to have two bigram dictionaries is to ensure in cases when the masked character is the first letter or the last letter of the word, or two consecutive letters are masked, I can still utilize this approach by considering the bigram dictionary. I took the weighted sum of these probabilities along with the probabilities from the BERT model to get the final probabilities.

While BERT simultaneously provides a prediction for all the masked characters, we only need to predict one character for each step in the game of Hangman. I took a greedy approach and considering only the masked characters took the character with the maximum probability.

After validating my approach and achieving an above-par performance (66%), I retrained the model on the entire dataset with the same setup and ran on the API provided to get an accuracy of 62% on the practice setup using this specific setup and 60% on the recorded games.