Hangman Strategy using BERT – Trexquant

Abhro Dhar

This report analyses a sophisticated Hangman strategy that employs a multi-model approach to maximize the chances of guessing the correct word. The strategy combines classical techniques with modern natural language processing methods, adapting its approach based on word length and game progress.

1. Core Components and Strategies

1.1 Dictionary-based Approach

The strategy starts with a comprehensive dictionary (training set) of 250,000 words, forming the basis for initial guesses and narrowing down possibilities as the game progresses. This approach quickly eliminates unlikely words based on the revealed pattern.

1.2 Letter Frequency Analysis

Using collections.Counter, the code analyzes letter frequencies in remaining possible words. This statistical approach prioritizes letters more likely to appear in the target word, especially in early game stages.

1.3 Vowel and Consonant Strategies

The code implements separate strategies for guessing vowels and consonants:

- Vowel Strategy: Prioritizes guessing vowels initially, as most English words contain vowels, and revealing them can significantly narrow down word possibilities.

- Consonant Strategy: After vowels, it moves to consonants, using frequency analysis to choose the most likely ones.

This two-phase approach balances the need to reveal crucial vowels with the frequency-based approach for consonants.

1.4 BERT Model Integration

The strategy incorporates a fine-tuned BERT (Bidirectional Encoder Representations from Transformers) model. BERT is particularly effective because:

- It understands context and word structure, going beyond simple letter frequency.

- It can predict masked tokens, aligning well with the Hangman game structure.

- Fine-tuning on a large word dataset allows it to specialize in word prediction tasks.

The BERT model is used as follows:

a) Unknown letters are replaced with [MASK] tokens.

b) BERT predicts the most likely letters for each mask.

c) Positional weighting is applied to favour earlier predictions.

d) The top 5 unique character predictions are returned.

1.5 Pattern-Based Guessing

The `_pattern_based_guess` method leverages common English word patterns:

a) It checks for common suffixes, prefixes, and roots.

b) It considers common letter combinations.

c) It returns up to 5 potential letters based on these patterns.

This linguistic approach leverages common word structures in English, potentially revealing multiple letters with a single guess.

1.6 Combining Multiple Strategies

The `combine_guesses` method integrates predictions from the dictionary-based approach, BERT model, and pattern-based guessing. This ensemble approach leverages the strengths of each method while mitigating their individual weaknesses. It works by:

a) Assigning scores to each guess based on its source and position in the guess list.

b) Weighting these scores according to provided weights.

c) Returning the guess with the highest combined score.

1.7 Adaptive Guessing

The strategy adapts its approach based on word length and the ratio of revealed letters:

For shorter words (<=7 letters):

- It prioritizes the original (dictionary-based) guess and sometimes includes pattern-based guesses.

- As more letters are revealed, it relies more on the original guess.

For longer words (>7 letters):

- It incorporates BERT guesses, especially as more letters are revealed.

- At high revelation ratios (>89%), it gives significant weight to BERT guesses.

This adaptive approach allows the strategy to leverage different strengths for different word types and game stages.

2. Implementation Details

2.1 Main Guessing Function: `guess(self, word, n_word_dictionary)`

This central function orchestrates the guessing strategy by combining the original guess, BERT model guess, and pattern-based guess. It then decides how to combine these guesses based on word length and the ratio of revealed letters. For words less than 8 letters long, I started off my initial guesses

with the 2 most common vowels and consonants for words of that length in the dictionary, as it would maximise my chances of getting an initial hit right.

2.2 Original Guess Function: `_original_guess(self, word, n_word_dictionary)`

This was referred from a pre-existing model built by Rakshith Kumar. (https://github.com/rakshit176/Trexquant-Hangman-Challenge-)

I fine-tuned and made relevant changes to improve performance but the core logic remains virtually unchanged.

This function implements the dictionary-based approach by:

a) Filtering the current dictionary to include only words matching the current pattern.

b) Counting letter frequencies in the remaining words.

c) Selecting letters based on frequency, with special handling for vowels.

d) Expanding the search to n-grams and eventually to the full dictionary if no suitable guess is found.

2.3 BERT Guess Function: `_bert_guess(self, clean_word)`

This function utilizes the fine-tuned BERT model to make context-aware predictions, which are especially valuable for longer words or when several letters are already revealed.

The fine-tuning of the BERT model on the training set of 250,000 words is a crucial step in optimizing the model for the specific task of predicting letters in the Hangman game. Let's break down the process and explain why it's necessary:

Why Fine-tuning is Necessary:

BERT (Bidirectional Encoder Representations from Transformers) is pre-trained on a large corpus of text, giving it a broad understanding of language. However, it's not specifically optimized for the task of predicting missing letters in partially revealed words, which is essential for Hangman. Fine-tuning allows us to:

a) Adapt BERT to the specific vocabulary and patterns present in the Hangman word list.

b) Optimize the model for the task of predicting missing letters rather than its original tasks.

c) Potentially improve performance on shorter words or unusual letter combinations that might be common in Hangman but less so in general text.

Specifics of the Fine-tuning:

a) Task Adaptation: The fine-tuning process adapts BERT's masked language modelling capability to the specific task of predicting letters in Hangman words. This is achieved by training on partially masked words from the Hangman dictionary.

b) Vocabulary Focus: By training on the Hangman word list, the model becomes more attuned to the specific vocabulary and word patterns that appear in the game.

c) Efficient Learning: The use of a pre-trained model allows for efficient learning. Instead of training from scratch, we're adjusting the existing knowledge of BERT to our specific task.

d) Hyperparameters: The choice of 3 epochs and a learning rate of 5e-5 are common starting points for BERT fine-tuning. These hyperparameters balance the need to adapt the model without overfitting to the training data.

Benefits for Hangman:

a) Contextual Understanding: The fine-tuned BERT can leverage its understanding of word structure and context to make more informed predictions about missing letters.

b) Adaptation to Game Patterns: By training on Hangman words, the model becomes better at predicting letters in the types of words commonly used in the game, which might differ from general text.

c) Improved Performance on Partial Words: The model becomes specifically adept at working with partially revealed words, which is the core challenge in Hangman.

In conclusion, fine-tuning BERT on the Hangman word list allows us to create a model that combines the broad language understanding of BERT with specific optimization for the Hangman task. This results in a powerful tool for predicting missing letters, which is a key component of the overall Hangman strategy.

2.4 Pattern-Based Guess Function: `_pattern_based_guess(self, clean_word)`

This function leverages common English word patterns to capture linguistic patterns that might not be evident from simple frequency analysis.

2.5 Fallback Guess Function: `fallback_guess(self)`

This function provides a last-resort guess based on overall letter frequency in the language, ensuring that the system always has a guess, even if other methods fail.

3. Challenges and Failure Points

3.1 Overfitting to Training Data

The main challenge is that the test set is independent of the training set. The model might perform well on words similar to those in the training set but struggle with completely new or unusual words.

3.2 Rare Words and Proper Nouns

The strategy might struggle with very rare words or proper nouns that don't follow common English word patterns.

3.3 Short Words

Very short words can be challenging because they provide less context for the BERT model and pattern-based guessing to work with. Any word with less than 6 or 7 letters is more likely than not to end up not being guessed due to the low probability of each guess (letter) being present in the word.

3.4 Balancing Strategies

Finding the right balance between different strategies (e.g., when to trust BERT vs. letter frequency) can be challenging and might require fine-tuning based on performance data.

3.5 Computational Resources

The BERT model, while powerful, requires significant computational resources. This could be a limitation in resource-constrained environments.

3.6 Adapting to Game Progress

The strategy needs to effectively transition between different approaches as the game progresses. Timing these transitions incorrectly could lead to suboptimal guesses.

3.7 Handling Misleading Partial Words

Partially revealed words that could match multiple possibilities might lead the model astray, especially if the correct word is less common.

3.8 Multiple valid words

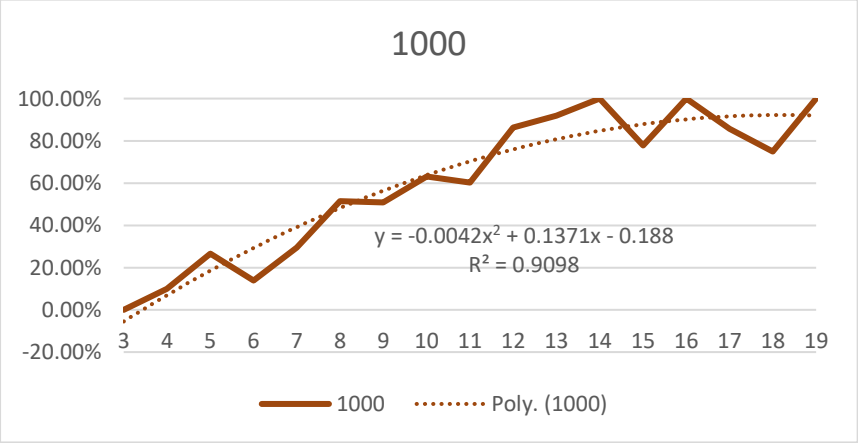
Often, there are many possible valid words to choose or guess, even when there are only 1 or 2 letters remaining – but there is only 1 correct answer according to the game. Thus, multiple guesses are being wasted in this endeavour, eventually leading to a lost game. This issue cannot be resolved by any algorithm whatsoever as it lies in the inherent nature of the Hangman game.

Result and Analysis

After 996 games, my final win rate stands at **54.72%**.
Unfortunately, there was a bug in my code for the first 231 games played. After fixing the issue, my win-rate was **55.82%** over the **next 765 games played**.

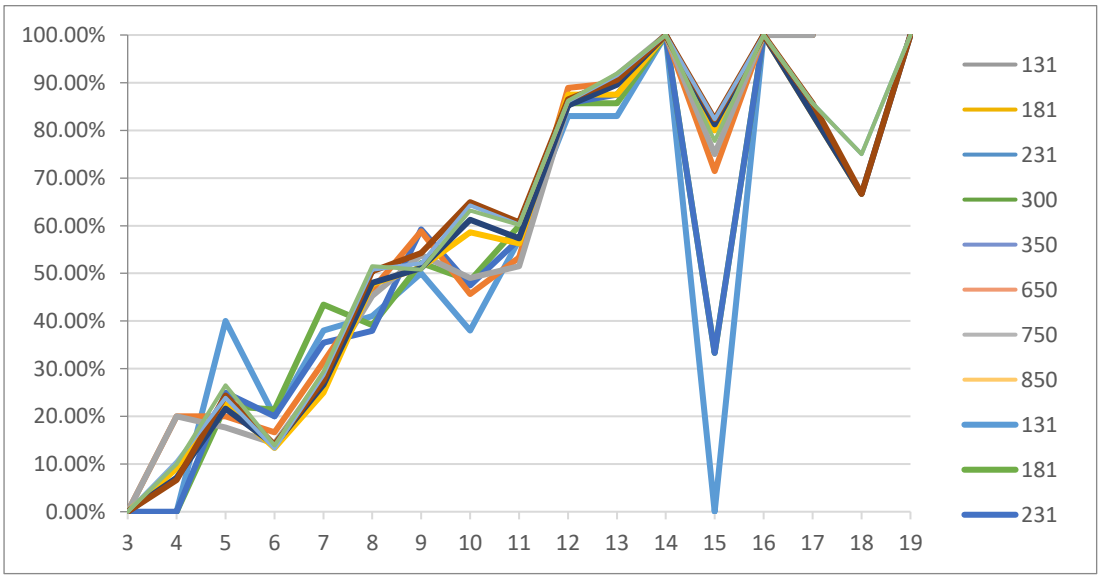
(I have lost data of the first 4 games played with practice=0 – must have played accidentally)

Count of Game	Result		
Length of Word	**Lost**	**Won**	**Grand Total**
3	12		12
4	18	2	20
5	36	13	49
6	62	10	72
7	84	35	119
8	64	68	132
9	63	65	128
10	49	84	133
11	41	62	103
12	11	69	80
13	5	57	62
14		41	41
15	4	14	18
16		15	15
17	1	6	7
18	1	3	4
19		1	1
Grand Total	**451**	**545**	**996**



I have found win-rate as an approximate quadratic function of length of words. This graph represents the win rate v/s the length of words. It is quite low for words with 7 or less letters and steadily increases for larger words, eventually plateauing close to 80-100% for words with more than 15 letters.

Length of Word	Frequency %	131	181	231	300	350	650	750	850	950	1000
3	1.20%			0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
4	2.01%	0.00%	0.00%	0.00%	20.00%	20.00%	9.09%	7.14%	6.67%	10.53%	10.00%
5	4.92%	40.00%	22.22%	25.00%	20.00%	17.65%	22.58%	21.62%	24.39%	23.91%	26.53%
6	7.23%	20.00%	21.43%	20.00%	16.67%	14.29%	13.33%	14.04%	14.06%	13.24%	13.89%
7	11.95%	38.00%	43.48%	35.48%	31.43%	28.57%	25.00%	26.67%	27.55%	28.70%	29.41%
8	13.25%	41.00%	39.13%	37.93%	46.51%	45.28%	47.73%	48.00%	50.44%	50.79%	51.52%
9	12.85%	50.00%	52.17%	59.26%	58.82%	53.85%	51.22%	51.11%	54.29%	52.10%	50.78%
10	13.35%	38.00%	48.48%	47.50%	45.65%	49.02%	58.62%	61.22%	64.91%	64.29%	63.16%
11	10.34%	57.00%	60.00%	57.14%	53.33%	51.52%	56.25%	57.33%	60.71%	60.20%	60.19%
12	8.03%	83.00%	85.71%	85.71%	88.89%	85.29%	87.50%	85.25%	86.49%	86.08%	86.25%
13	6.22%	83.00%	85.71%	87.50%	90.00%	91.67%	87.50%	89.58%	90.57%	91.53%	91.94%
14	4.12%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
15	1.81%	0.00%	33.33%	33.33%	71.43%	75.00%	80.00%	81.25%	82.35%	82.35%	77.78%
16	1.51%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
17	0.70%	100.00%	100.00%	100.00%	100.00%	100.00%	83.33%	83.33%	85.71%	85.71%	85.71%
18	0.40%						66.67%	66.67%	66.67%	75.00%	75.00%
19	0.10%							100.00%	100.00%	100.00%	100.00%
	Overall Win Probability	48.94%	51.56%	51.07%	52.04%	50.75%	52.19%	52.91%	54.92%	54.74%	54.72%



Win-rate vs length of word for different iterations (upto 1000 games)

The table and the graph show that for the first 230 iterations, the win rate was a bit low (~ 51%). It then finally rose to 54.72%. The peak occurred between the 750th and 850th game, where the algorithm maintained an impressive win rate of **74%.** This was of course a one-off, but demonstrates the potential of this strategy when its strengths are amplified.

Conclusion

This Hangman strategy combines classical approaches like dictionary lookup and letter frequency analysis with modern NLP techniques (BERT) and linguistic knowledge (pattern-based guessing). Its strength lies in this diverse, multi-faceted approach and its ability to adapt to different word types and game stages.

The strategy's effectiveness comes from its ability to leverage different methods based on the current game state. For shorter words, it relies more on frequency-based and pattern-based guessing, while for longer words, it incorporates the contextual understanding provided by the BERT model.

However, its reliance on training data and common word patterns means it might struggle with highly unusual or completely novel words. Continuous refinement based on performance data, especially on words it fails to guess, could further improve its effectiveness.

The implementation of this strategy demonstrates a sophisticated understanding of both the Hangman game mechanics and modern NLP techniques. By combining multiple approaches and adapting them based on word length and game progress, this strategy creates a robust and effective Hangman solver that can handle a wide variety of words and game situations.

I believe it is extremely difficult to consistently cross a win rate of 60-65% in the given scenario - with 6 incorrect guesses permitted, the test set being independent of the training set and containing uncommon words and the high proportion of small words, due to the underlying probabilistic nature of the game. Yet, through the power of deep learning and other tools, we have managed to optimise our strategy to quite a good extent. I am fairly confident that my model can been improved with better computational resources and more time.