

Hangman with LLMs

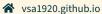
Training Google's CANINE transformer to play Hangman

March 2024

Vaibhav Sahu

University of Pennsylvania







HANGMAN: A Word Guessing Game

Hangman is a popular word-guessing game:

- 1. Player 1 typically chooses a word that Player 2 has to guess. All the letters are concealed at the start of the game.
- 2. Player 2 then has to guess the letters of the hidden word, one letter at a time. All its positions are revealed if the guessed letter is in the word.
- 3. If a guessed letter is not in the word, Player 2 exhausts one failed try. If the player guesses the word before guessing the wrong N times, they win; otherwise, they lose. (typically N = 6 or 7)

How hard is Hangman?

Number of letters	Optimal calling order
1	Al
2	AOEIUMBH
3	AEOIUYHBCK
4	AEOIUYSBF
5	SEAOIUYH
6	EAIOUSY
7	EIAOUS
8	EIAOU
9	EIAOU
10	EIOAU
11	EIOAD
12	EIOAF
13	IEOA
14	IEO
15	IEA
16	IEH
17	IER
18	IEA
19	IEA
20	IE

Figure: Calling Order for the first letter according to word length Image courtesy - Datagenetics

Formulating the problem

Hangman: Actions and States

Applying the rules of Hangman (H), it can be modeled as taking an action (a_t : guessing 1 of the 26 letters in the English Alphabet), given a Hangman game state (s_t : Guessed letters and the state of the hidden word)

$$H(a_t, s_t) = s_{t+1}$$

Encoding the Hangman Game States

We want to model the game states to feed it to our Transformer model:

```
s_t = [CLS] < GUESSED\_LETTERS > [SEP] < HANGMAN\_STATE > [SEP]
```

Example:

Actual word: b i r t h d a y; Guessed letters: e, a, i, s; State: $_i ___ a _$ $s_t = [CLS]$ eias [SEP] [MASK] i [MASK] [MASK] [MASK] [MASK] [SEP]

Machine Learning Task

How do we model the decision that our agent takes at each step? It turns out there is more than 1 way to do so.

- 1. Multi-label Classification Task:
 - Label consists of every hidden letter that hasn't been guessed.
 - Model can simultaneously guess letters!
 - Not too restrictive. Well defined? Hard to train?
- 2. Multi-class Classification Task:
 - Guess the most likely or the most frequent letter.
 - Make a distribution of the likelihood by normalizing the counts.
 - A 1-1 mapping between labels and states. Easier to train?

$$\left[l(s_t)\right]_i = \frac{\sum_j \delta_j \cdot I_{x_j = x_i}}{\sum_i \sum_j \delta_j \cdot I_{x_j = x_i}}$$

where x_i is the i-th letter of the alphabet, and x_j is the letter at the j-th position of the word

Data Generation

We have chosen the right way to label the data. The number of Hangman states in a Hangman game is huge!

$$||\Omega|| \sim 2^k \cdot {26-k \choose N} \cdot N!$$

k: Number of unique letters in the word)

N: Number of allowed wrong tries

How do we sample efficiently?

Biased Random Sampling

Game Simulations

Learn as much as possible from each simulated game!

Biased Random Sampling (p)

Proposed approach:

- With a probability p, sample from one of the missing letters in the word
- The rest of the time, sample randomly from any of the letters that haven't been guessed
- p is a hyperparameter to be tuned. We find p = 0.4 works well!

Model Selection: Google's CANINE

- Our states have character-level information stored in them
- Intuitively, we need a character-level language model that can model the conditional distribution of letters in English words!
- CANINE: [Clark et al., 2022]

Training our Transformer from scratch!

The model does not benefit from transfer learning of sentence-level language modeling. This is why it's actually simpler to work with it from random initialization:

Training Algorithm (Pre-training)

- Preprocess a corpus of words: Random-split to train, validate, and test
- At each epoch, for each word, the training corpus is used to perform gameplay simulation through biased sampling
- The model then learns to match the output distribution as defined before for each state, using Cross-entropy loss

Pre-training Results

Our model has 50+% game-winning accuracy with 6 tries!

Self-play fine-tuning

Our model does well. But can we improve its accuracy further? Biased sampling provides a lot of simulation states, but as the model becomes better, it needs more precise data. We need more cases where we know it does poorly.

Self-play fine-tuning Algorithm - motivated by [Mary Wahl, 2017]

- Let our model navigate through the state space on its own based on its guesses
- If it guesses correctly, we proceed; otherwise, we calculate the loss
- Iteratively update the model to correct these mistakes by optimizing our loss

Self-play fine-tuning Results

Our model has 56% game-winning accuracy with 6 tries and 83% with 10 tries!

Future Scope and Improvements

Future Scope

We have successfully trained our Hangman agent. What other cool downstream tasks can our trained model be used for?

- 1. Our model can be fine-tuned to be a great spell-checker and auto-complete! It can identify if a spelling is correct or not and even provide suggestions to auto-complete.
- 2. Can be an interesting decrypting tool for simple decrypting tasks!
- 3. Can help in speech recognition in filling missing partial phonemes.

Future Scope and Improvements

Improvements

How can we improve our model performance, reduce its latency, and speed it up?

- Our model is very slow at backdrops when working with single samples at a time. If we could batch them together, the fine-tuning process and iteration would achieve significant speedup.
- Parameter Efficient Fine-tuning (PEFT) is another idea worth exploring to speed up fine-tuning our model. Backpropagation would be faster even when working one sample at a time.
- More hyperparameter tuning: We have been very conservative in using Hyperparameter values, and some good enough values have worked. However, we have parameters that still require more analysis. We have a Bias rate of p and also the training and fine-tuning hyperparameters such as η and λ .

References

- Clark, J. H., Garrette, D., Turc, I., and Wieting, J. (2022). Canine: Pre-training an efficient tokenization-free encoder for language representation. Transactions of the Association for Computational Linguistics,
 - Transactions of the Association for Computational Linguistics, 10:73–91.
- Mary Wahl, Shaheen Gauher, F. B. U. K. Z. (2017). Microsoft azure: Play hangman with cntk.