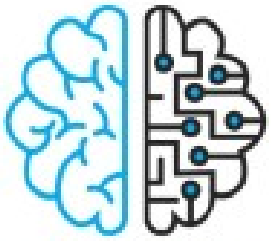

TREXQUANT

Solution Approach to the Hangman Problem
IIT Madras

Submitted by

K.S MOHAN KUMAR (ME19B118)
Inter Disciplinary Dual Degree
Mechanical Engineering and Data Science
IIT Madras
Chennai

Contents

1 Introduction **1**

2 Exploratory Data Analysis for the dictionary **2**
2.1 About the Data 2

3 Masked Language Modelling **4**
3.1 Masked Language Modelling 4
3.2 RoBERTa 4
3.3 Training Setting 5
3.4 Strategy for selecting optimal action 5
3.5 Results 5

4 Reinforcement Learning Approach **6**
4.1 Environment 6
4.2 D3QN 7
4.3 Results 7

5 N-Grams **9**
5.1 N-Grams Description 9
5.2 Strategy used for Guessing Letters 9
5.3 Result 9

Chapter 1

Introduction

The hangman problem is a word-guessing game, where players have a limited number of guesses to predict (here 6 guesses). The assignment was to implement an ML solution to get an accuracy of over 50% in the training dataset. The three approaches used in solving the problem was:

- MaskedLanguage Modelling using the RoBERTa model.
- RL approach with Duelling Double Deep Q-Networks.
- N-grams approach.

The best model that worked out was the one using the concept of n-grams where i got an accuracy of 58.7 percent. Since, I am familiar with Deep Reinforcement Learning, even though it did not give quite the results, it has very high potential. I also tried implementing PPO for the problem.

Chapter 2

Exploratory Data Analysis for the dictionary

2.1 About the Data

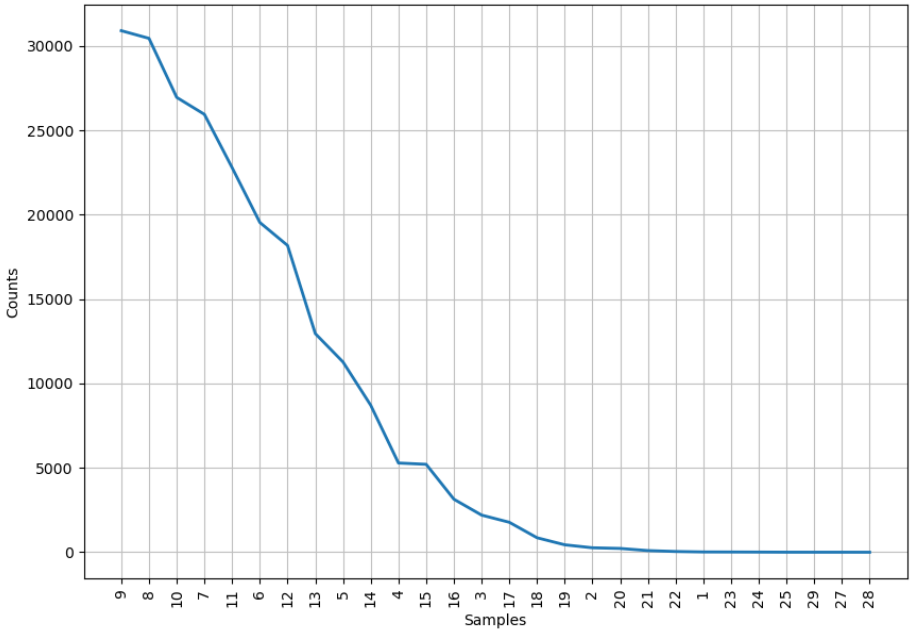

Figure 2.1: Distribution of Length of Words

The dataset comprises 227,300 English words. The average word length is 9.347, and the maximum length is 29, predominantly falling within the 5 to 11 range. The top five occurring alphabets are e, i, a, r, and n. Remarkably, approximately 90% of the words include at least two letters from this top-five list. Therefore, a strategic approach could involve initiating guesses with these high-frequency alphabets, leaving three attempts to partially unveil the word. The alphabets in Frequency order of their appearance in a word:

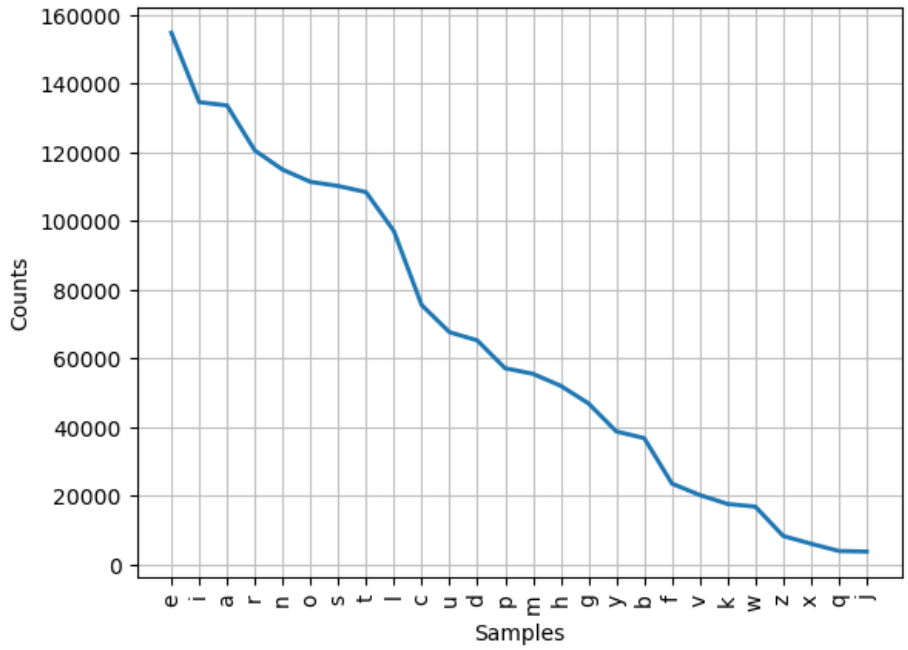

Figure 2.2: Distribution of each alphabet

Chapter 3

Masked Language Modelling

3.1 Masked Language Modelling

What is Masked Language Modeling Masked Language Modeling (MLM) is a popular deep learning technique used in Natural Language Processing (NLP) tasks, particularly in the training of Transformer models such as BERT, GPT-2, and RoBERTa.

In MLM, a portion of the input text is "masked" or randomly replaced with a special token (usually [MASK]) and the model is trained to predict the original token based on the context surrounding it. The idea behind this is to train the model to understand the context of words and their relationships with other words in a sentence.

MLM is a self-supervised learning technique, meaning that the model learns to generate text without the need for explicit annotations or labels, but instead using the input text itself as supervision. This makes it a versatile and powerful tool for a wide range of NLP tasks, including text classification, question answering, and text generation.

3.2 RoBERTa

RoBERTa is a transformers model pretrained on a large corpus of English data in a self-supervised fashion. This means it was pretrained on the raw texts only, with no humans labelling them in any way (which is why it can use lots of publicly available data) with an automatic process to generate inputs and labels from those texts.

More precisely, it was pretrained with the Masked language modeling (MLM) objective. Taking a sentence, the model randomly masks 15% of the words in the input then run the entire masked sentence through the model and has to predict the masked words. This is different from traditional recurrent neural networks (RNNs) that usually see the words one after the other, or from autoregressive models like GPT which internally mask the future tokens. It allows the model to learn a bidirectional representation of the sentence.

This way, the model learns an inner representation of the English language that can then be used to extract features useful for downstream tasks: if you have a dataset of labeled sentences for instance, you can train a standard classifier using the features produced by the BERT model as inputs.

3.3 Training Setting

To encode the data we use the alphabets along with special tokens(start,end,pad,mask) as vocabulary and pad each vector to a size of 29(Maximum size of word in the dictionary). We split the words in the given dictionary into train(90%) and validation set(10%). The model was run for 7 epochs on the training dataset. The architecture of the model is described in the code.

3.4 Strategy for selecting optimal action

I started with the most_freq characters frequency. The naive solution is to use the guessing leetters in the decreasing order o f their frequency in every game.
So what we do is when we have more than 3 guesses, we use the letter from the most frequent characters list which has not been already selected. And then use the tokenizer to encode the word with masks and use the RoBERTa model to predict the probabilities of each character at the masked position and choose the word which has the highest probability.
Since the output of the transformer model is not normalized, i used a softmax function to normalize the values, that are to be compared.
Example. let's take the word to be engineering.
First 3 guesses from maximum frequency word."eiarnostlcudpmhgybfvkwzxqj" The rounds will progress as:

- e _ _ _ _ e e _ _ _ _ (guess=e)
- e _ _ i _ e e _ i _ _ (guess=i)
- e _ _ i _ e e _ i _ _ (guess=a)
- e _ _ i _ e e r i _ _ (guess=r)
- e n _ i n e e r i n _ (guess=r)
- e n _ i n e e r i n _ (guess=o)
- e n _ i n e e r i n _ (guess=s)
- Next we use prediction from RoBERTa model.

3.5 Results

The results of this model is not accurate because of the overestimation. Here, the current state of the problem depends on the previous state. So, it skews our data.
So i experimented with 3 variants each ran on 100 runs in the practice mode, and found the results.

No of guesses after which we should start using the model	Accuracy
3	8
4	22
5	43

Chapter 4

Reinforcement Learning Approach

4.1 Environment

In the Hangman game, we formulate the environment as a Markov Decision Process (MDP), where the state space, action space, and the dynamics of the game are defined.

States

The state space is represented as a vector containing information about the current game configuration:

$$\text{State} = [\text{Binary vector of guessed letters}, \text{Attempts left}, \text{Set of used letters}]$$

The binary vector is of length 26, where each element corresponds to a letter in the alphabet. If a letter has not been guessed, the element is 0; otherwise, it is set to the position of the letter in the alphabet (1 to 26).

Actions

The action space consists of selecting a letter to guess from the alphabet:

$$\text{Action} = \{a, b, c, \ldots, z\}$$

It is a discrete space with 26 possible actions, each corresponding to one of the 26 letters in the English alphabet.

Rewards

The reward function is designed to encourage correct guesses and penalize incorrect ones:

$$\text{Reward} = \begin{cases} 3 & \text{if the guessed letter is correct} \\ -3 & \text{for the first consecutive incorrect guess} \\ -6 & \text{for each subsequent consecutive incorrect guess} \\ 30 & \text{upon successfully completing the word} \\ -30 & \text{if maximum attempts are reached without completing the word} \end{cases}$$

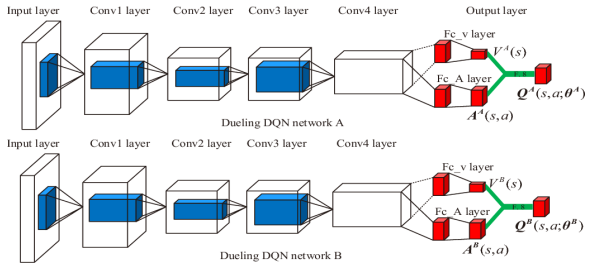

Figure 4.1: Duelling Double DQN

Observations

The observation space is a vector of length 29, that encodes what is the particular letter present in the guesses state of the word. The alphabets are numbered with the index value in english dictionary, the padded numbers are represented by -5 and masks are represented by 0.

4.2 D3QN

The agent used is trained with the D3QN algoithm. The code for the agent is adapted from it's official paper and i have modified the choose action function.

D3QN Training Method

- Training begins when the experience buffer has accumulated experiences equal to or exceeding the batch size.
- Update the target network after training the Q-network for a specified number of times (self.replace times).
- Sample a batch of experiences, resulting in NumPy arrays of states, actions, rewards, next states, and the 'done' variable.
- Predict Q-values for current states and next states using the Q-network and target network, respectively.
- Calculate the action with the maximum Q-value for the next state using the Q-network
- Compute batch indices and perform the update operation using the Double DQN update approach.
- The update equation is multiplied by the 'done' variable, considering that for terminal states, the Q-value is always zero.
- Train the model, update the epsilon parameter, and increment the training step.

4.3 Results

After training for 90,000 epochs on the whole dictionary, the agent was able to produce an accuracy less than 20%.This is because the agent has not seen the whole dictionary.

However, when taking a small subset of 20 words and training our agent for 10000 episodes, the accuracy was more than 60%. Hence, there is high potential for the RL model.

Chapter 5

N-Grams

5.1 N-Grams Description

N-grams are contiguous sequences of n items from a given sample of text or speech. The items can be phonemes, syllables, letters, words, or base pairs according to the application. N-grams are used in various areas of computational linguistics and text analysis.

5.2 Strategy used for Guessing Letters

- We first add a character representing the start and end of a word to each word in the dictionary.
- We then form unigrams, bigrams, trigrams,tetra-grams, and pentagrams based on the count of each of the n-grams and calculate the count.
- Next we iterate through the whole missing word, check 1,2,3,4,5 letter words which have only one letter missing, and calculate the count for each alphabet with which we get the conditional probability.
- I then used an exponential weight array to get the final scores for each of the n-grams and chose the alphabet that is already not chosen and returned the letter. The weights used are exponential powers of 2.The letter with the highest probability is taken.
- The sequence is contiued until the player wins or he makes 6 wrong guesses.

5.3 Result

The n-gram model using upto 5 grams provided a final accuracy of 58.7 % in the final recorded response.