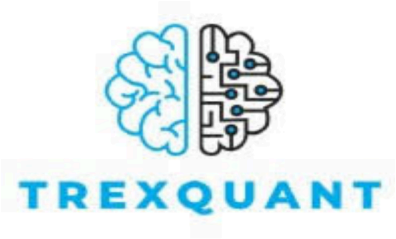


Project Documentation

Sree Snehan
PSG Institute of Technology and Applied Research
B.Tech AI and Data Science

Accuracy Achieved: **66.4%**

Table of Content	
1.	Introduction and Intuition
2.	Approach 1: Basic Transformer
3.	Approach 2: Hybrid (ByT5 + Bi-LSTM)
4.	Approach 3: Hybrid (Canine + 26*XGBoost)
5.	Approach 4: mBERT
6.	Approach 5: Hybrid (Canine + Regex Matching)

Introduction

The approaches to solve the Hangman involves training the model to choose an action in a broad action space based on the positional and characteristics of letter arrangements in English language. This is not possible by basic ML algorithm and requires transformer based architecture to understand complex dependencies.

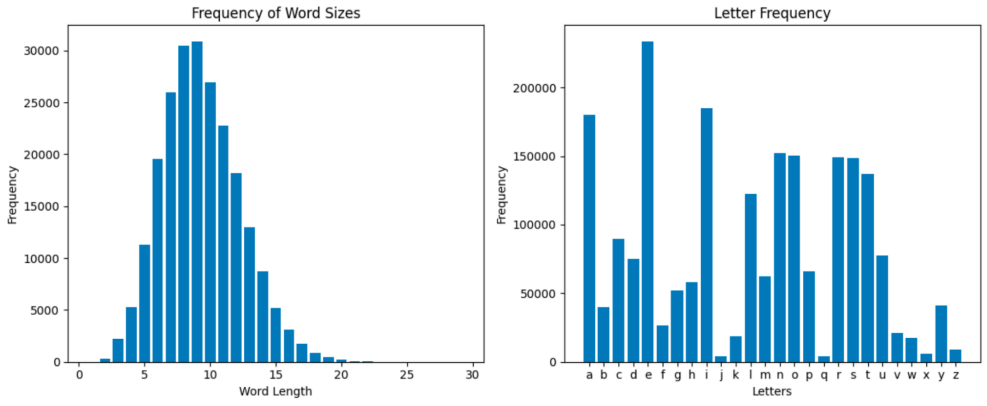
RL based training shines in these types of scenarios with broad search space like in the training of AlphaGo by DeepMind , where there are big action space which cant be properly modelled into the supervised techniques. The RL approach also open roads for the model to be more creative with small tricks and approaches which cant be otherwise trained from Supervised training techniques.

Hardware**: NVIDIA RTX A6000 48GB -** on Paperspace.

Data PreProcessing:

The dataset was relatively clean and required some cleaning to enhance the quality of predictions.Two letter words and Three letter words in the dataset were completely random and void any meaning patterns , so they were removed to ensure that model is trained properly.

```
'ghq',              ['j',
'faq',               'sqq',
'chq',               'jj',
'ahq',               'qv',
'boq',               'qf',
'coq',               'qu',
'loq',               'qp',
'liq',               'qc',
'ceq',               'qh',
'arq',               'qs',
'req',               'qt',
'aoq',               'qa',
'aeq',               'qe',
'piqu',              'vj',
'wsj',               'kj',
'mdqs',              'hj',
'ddj',               'rj',
'mmj',               'sj',
```

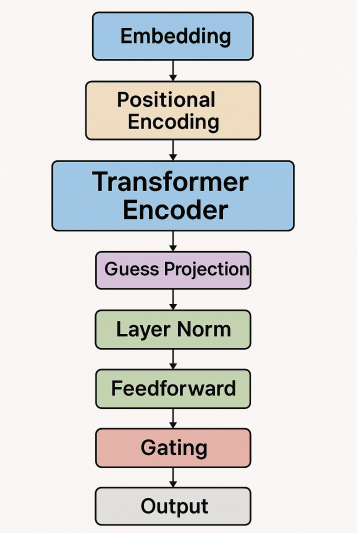


Except for this , the data was mostly clean for training with no duplicates.

APPROACH 1 (Basic Transformer)

Intuition:

The solution space for the model to train is very broad hence requires RL based approaches similar to Alpha-Go, Direct training of an architecture of basic transformer was used , this transformer was directly trained on the dataset.



Key Features:

1. Top-K weighted loss was used instead of individual letter loss since it was more appropriate for this case, the model is not expected to be accurate all the time, it should ensure that it can predict a set of letters ideal in that scenario.
2. The reward function was based on the % of letters that the model can predict in the whole iteration. It was more ideal because of the variation in sizes of words.
3. A stratified variation of the dataset was used to train the transformer architecture.

Performance:

1. The model **failed to converge** and failed to capture relations properly.
2. Constantly predicted the same letters irrespective of the size and found letters in many cases.
3. **Accuracy: 31% max**

Drawbacks:

1. Direct RL was overwhelming for the model to properly understand the foundations of letter modelling. Pre Training was required.
2. The model failed to accurately finish the words in many cases.

3. The model over represented certain letters over others, needed strong regularization and improved architecture.

APPROACH 2 (Hybrid (ByT5 + Bi-LSTM)

Intuition:

ByT5 model architecture supported character level modelling , this was chosen to address the inconsistencies in the original architecture that was used in Approach 1. **Direct - Training** was done instead of direct RL to address the issues of Approach 1. While testing the initial ByT5 , it was identified that the model was struggling to finish the sentences at the end , since these letters were specific raw knowledge rather than being pattern oriented. So an additional model with Bi-LSTM architecture was enhanced and moderated the output of the model.

Methodology:

1. It was identified that there was over representation of certain letters over others and hence model was more confident in certain characters over others , the dataset for the pre training was based on log inverse frequency of individual characters.

```
def simulate_game(word, correct_rate=0.4, max_wrong_guesses=6):
    word_idxs = {}
    all_letters = [chr(i) for i in range(ord('a'), ord('z') + 1)]

    # Inverse frequency weights
    hi = {'p': 0.378417,
         'e': 0.37,
         'r': 0.371867,
         'i': 0.370874,
         'w': 0.410156,
         'g': 0.381564,
         'c': 0.375326,
         'o': 0.371839,
         'n': 0.371764,
         'a': 0.370988,
         'l': 0.372997,
         'u': 0.376671,
         'd': 0.376997,
         'm': 0.379093,
         't': 0.372317,
         'z': 0.454771,
         'x': 0.494058,
         's': 0.371893,
         'k': 0.40794,
         'b': 0.386043,
         'f': 0.395855,
         'y': 0.385503,
         'q': 0.560004,
         'v': 0.403295,
         'h': 0.379977,
         'j': 0.57}
```

2. The dataset for the supervised learning was manually generated for 1M states replicating hangman games with missing letters from words with wrong/correct guesses.
3. The dataset for the LSTM model was prepared by masking two or more letters based on inverse frequency weights shown above.

4. The model was struggling to finish the words , hence the LSTM model was added to support the ByT5 model when it reaches the last two characters.
5. The algorithm predicts the guess based on the ByT5 till n-2 characters , then the prediction from the LSTM model was used.

Performance:

1. The ByT5 model was able to reach an accuracy of 42.2% individually.
2. The LSTM model was able to reach 60% accuracy on the disjoint validation state to predict the last two letters in 3 guesses.
3. When integrated together , the algorithm was able to perform with an accuracy of **46.8% max** and **42%** on the validation set.
4. The model was struggling with words with rare arrangements , even when there exists a pattern in them.

Draw Backs:

1. The model took 2.5 hrs for each epoch on NVIDIA RTX A6000 48 GB despite maximising the VRAM with 128 batch size. The model was compute and time intensive and wasn't improving much after consequent epochs.
2. The model was not as specialised in character level modelling as other ones like CANINE.
3. Longer inference times for each prediction.

APPROACH 3 (Hybrid (CANINE + 26*XGBoost)

Intuition:

From research on benchmarks , CANINE performed better than m-bert and ByT5 in character level modelling , hence this was chosen. The model was Pre-Trained on the dataset directly and RL - based fine tuned on words with unique arrangements. To enhance the predictions of the model in cases with low confidence prediction and large deviation from the normal , XGboost models were trained, instead of single model , to capture exclusive patterns relative to each character , 26 XGBoost models were trained.

Methodology:

1. Supervised Pre Training of the model on the whole dataset to ensure that the basic relation dependencies between characters are captured.
2. It was observed that CANINE model and various previous versions were struggling on the unique arrangements , so a score based train sample selection was done to RL finetune the model.

Score(w) = (1 / |w|) × Σ$_{i=1}$^{|w|} [½ × (1 / f_front(w_i, i) + 1 / f_back(w_i, |w| − i + 1))]

- **w_i** = the i-th letter in the word
- **|w|** = length of the word
- **f_front(c, i)** = frequency of character **c** at front position **i** (from start of word) across all words
- **f_back(c, j)** = frequency of character **c** at back position **j** (from end of word) across all words

Many characters like j occur rarely in middle, but common occurrence at the start of some words like pneumonia etc , so if there is a presence of j in a word in the middle of a words , it needs to have higher preference than a word having j at the start , so freq with position is used.

Since the words have different sizes , a forward based count wouldn't be enough to capture the pattern. A word like Sec**u**re , has u in the middle , but word like bea**u** has word at the end , this is not a common occurrence compared to middle occurrence , but pos=4 in both cases , so backward freq is also added.

The dataset was sorted based on this frequency and uniquely occurring words were identified which was used to finetune the model with RL training.

```
africanizing
jarabe
swordbill
squirrelling
cambeva
blownup
wellforewarning
stbark
zoolaters
effeminating
perquisite
imprejudicate
erythrocytorrhexis
```

3. The XGboost models was trained on the dataset , combined normalised probability is used to properly identify the best prediction out of all the models.
4. The CANINE model was integrated with XGBoost based on normalised confidence scores of CANINE and rank of the predicted letter in the predictions of the XGBoost models. If the model predictions are low confidence and deviated far from the ranks of XgBoost models by 5, i.e if the predictions of the CANINE

are not in the Top - 5 predictions of XGBoost , then use the prediction of the XGBoost model.

XGboost Probability	Canine Probability	Rank of Canine in XGBoost predictions	Prediction of Canine	Prediction of XGboost

```
[WORD]: otorrhagia
  Guess: E  → **********  (Wrong: 1) 0.741 0.021 0, 'Org', e e
  Guess: I  → *********i*  (Wrong: 1) 0.640 0.020 0, 'Org', i i
  Guess: A  → ******a*ia  (Wrong: 1) 0.627 0.016 1, 'Org', a o
  Guess: O  → o*o***a*ia  (Wrong: 1) 0.596 0.013 0, 'Org', o o
  Guess: H  → o*o**ha*ia  (Wrong: 1) 0.342 0.016 2, 'Org', h r
  Guess: R  → o*orrha*ia  (Wrong: 1) 0.434 0.009 1, 'Org', r t
  Guess: N  → o*orrha*ia  (Wrong: 2) 0.081 0.005 2, 'Org', n t
Trigger
  Guess: T  → otorrha*ia  (Wrong: 2) 0.047 0.008 9, 'Org', m t
  Guess: M  → otorrha*ia  (Wrong: 3) 0.048 0.004 3, 'Org', m g
Trigger
  Guess: G  → otorrhagia  (Wrong: 3) 0.019 0.004 7, 'Org', b g
Result: ✅ CORRECT | Final: otorrhagia otorrhagia
```

XGBoost prediction used when the following conditions satisfied:

1. Normalised confidence of guess <=0.005
2. Rank of guess w.r.t XGBoost >=5
3. Missing Letters count <=4

The positions of Trigger represent the intervention of XGBoost model to change the guess of CANINE model to less confidence 0.008 and low rank 9 , so changed guess **M -> T** . The second intervention highlights the performance boost offered by XGBoost to finish the last character; it changed the prediction from **B -> G.**

Performance:

1. The CANINE model alone was able to achieve accuracy upto 58% , considering the algorithm to include the intervention of XGBoost model , the accuracy jumped up by 3% , the total max accuracy upto 61% was achieved by the combined implementation on the Original Dataset.

```
Map:   0%|          | 0/2291036 [00:00<?, ? examples/s]
```


[17899/17899 44:49, Epoch 1/1]

Epoch	Training Loss	Validation Loss	F1	Accuracy
1	0.126600	0.125596	0.251273	0.351336

```
Validation accuracy: 0.584
```

2. There are still some improvements to be made on the CANINE as confidence of prediction is skewed towards some , need to balance out the confidence between the characters.
3. The model still struggles to predict on properly on the words of small size like 4/5/6 due to less inherent pattern in occurrence in some of these small words.

APPROACH 4 (mBERT):

Intuition:

https://arxiv.org/pdf/2210.07111

This paper suggested mBERT is better than CANINE and ByT5.

Performance:

The model was compute intensive and was slow to converge , taking 3.5 hrs for each epoch , hence the approach had to be dropped after 6 epochs , reaching **max accuracy of 35%.**

APPROACH 5 (CANINE + Regex Matching):

Intuition:

Majority of the unguessed words were completed till the last two letters, these last two letters depended more on knowledge base rather than pattern or positional relevance. Hence a regex based probability estimator reinforced the outputs in cases of low confidence instead of a model based one.

Methodology:

1. Canine predictions were used alongside Regex Matching to guess the letter with higher confidence. The confidence of the higher one was used to guess the letter.
2. Running a sliding window of size largest len to 3 on the masked word to

construct patterns of different sizes which are compared against the dictionary corpus of 250K words to identify the most relevant letters and their respective probability for a given position.

3. A Confidence Calibration Factor was used to determine the balance proportion and threshold of guesses between canine and regex with least incorrect positives. The factor allowed us to filter the correct threshold of confidence of Canine beyond which it is mostly incorrect.

Performance:

Factor	Correct guesses on 100 train samples.	Canine correct	Canine wrong	Regex correct	Regex wrong
1	68%	176	90	412	282
1.2	74%	217	107	428	245
1.4	73%	243	121	391	227
1.6	**77%**	**277**	**150**	**372**	**184**
1.8	76%	306	172	341	153
2.0	70%	304	214	295	170
2.2	68%	334	245	265	145

Factor of 1.5 was used based on observations.

Accuracy achieved on Unseen data: 66.4%

```
[total_practice_runs,total_recorded_runs,total_recorded_successes,total_practice_successes] = api.my_status() # Get my game stats: (# of trie
success_rate = total_recorded_successes/total_recorded_runs
print('overall success rate = %.4f' % success_rate)
```

```
overall success rate = 0.6640
```

Final Results:

Max probability without Knowledge base: 60.9% (on train set) - 52% (on unseen set)
Max probability with Knowledge base: 78% (on train set) - 66.4% (on unseen set)

References:

1. https://github.com/vsa1920/Hangman-with-Transformers

The functions to set up the **Hangman environment** and **Fine-tuning script** are adopted based on this repository.

2. https://github.com/Azure/Hangman/blob/master/Train%20a%20Neural%20Network%20to%20Play%20Hangman.ipynb

3. https://arxiv.org/pdf/2210.07111

Observation to use mBERT was based on this.

Source Files:

https://github.com/snehandot/Hangman.git