My strategy

Yanfei Cui, 10/10/2023
Apply for full-time quantitative researcher in China

1. Result

My strategy achieved a winning rate of 54.1% on hangman game.

2. Strategy

2.1 Overall Method

I completely use deep learning methods without setting any artificial rules.

2.2 Details

2.2.1 Model

Encoder: Encoder of a 6-layer attention transformer.
Decoder: One fully-connected layer.

2.2.2 Dataset

For all the words in words_250000_train.txt, I randomly select 90% as train_dataset and the remaining 10% as valid_dataset.

For each word in words_250000_train.txt, I randomly mask i types of letters (i ranging from 1 to len(word_letters)), generating several words with different degrees of masking for model training. The labels are probability distributions of each masked letter. The goal is to make the model predict the probability distribution of the next letter.

2.2.3 Criterion

I use KL divergence as the loss function.

2.2.4 Training Procedure

The entire training process is divided into three parts.

2.2.4.1 Part1 basic training

According to words_250000_train.txt, I randomly generated a training set and a validation set (named pairs_train.txt and pairs_valid.txt). After training for 75 epochs, I obtained a baseline model with approximately 35% winning rate.

2.2.4.2 Part2 Data augmentation

For a word with n distinct letters, if I randomly mask k of them. There are a total of C_n^k different combinations. However, in my

training set for part 1, only one combination was taken. This means that the majority of letter combinations have not been seen by my model. Therefore, I need to perform data augmentation. My method of data augmentation is to generate a new training set and validation set randomly for each epoch during training, keeping them in memory without saving them to the hard disk. This way, it increases the number of letter combinations seen by the model.
Using this approach, after training for 12 epochs (reaching epoch 87), I obtained a relatively good model with a winning rate of 45%.

2.2.4.3 Part3 Improvements for short words

However, after my careful observation, I found that my model has significantly different accuracy rates for long words and short words. Here, I define long words as those with a length greater than or equal to 6. In the case of long words, the success rate of my model can reach up to 70%, but for short words, it is less than 20%. Therefore, I need to fine-tune a specialized model specifically for predicting short words.

Through observation, I discovered that the difficulty with short words lies in the limited number of known words provided in the question. For long words, it is relatively easy to guess several highly probable letters and then rely on the relative structure of these known letters to make quick and accurate inferences. However, this approach does not work well for short words. To address this issue, I introduced incorrectly guessed letters into the training set so that the model could learn to exclude these incorrect guesses and make more accurate predictions.

Firstly, I selected all single-word samples with lengths less than 6 from "words_250000_train.txt" and created a dedicated dataset (named pairs_train_small.txt and pairs_valid_small.txt). Then using the model trained in part2 as a backbone and employing part2's data augmentation methods for training again; however, there was one difference: after each sample word was randomly obscured (masked), I would generate 0-6 letters that do not exist in this word randomly behind it to simulate situations where incorrect guesses were made. These generated letters were concatenated at the end of each masked word. By training until epoch 112 using this method successfully produced an effective model specifically for short words with a success rate reaching 34%.

3. Discussion

In this way, I successfully created a model with a win rate of 54.1% on the test set using the above methods. However, due to limited time, if there was more time available, I believe my model could ultimately achieve a win rate of over 60%.

In fact, because of the tight schedule, my model for part 3 was not trained to its optimal state. Additionally, my original plan was to incorporate information about incorrectly guessed words into the long word model. However, training time would increase by 1.5 times and it was found through experimentation that an additional 100 epochs were needed for convergence. It would take 24 hours to train using an A100 GPU card which I didn't have enough time for; hence I can only submit these results.