Language Model

Yuan Feng, 521030910369
Shanghai JiaoTong University

1. Introduction

Language models is a mathematical model used to compute the probability of language. Language models attempt to understand the structure and patterns of language, and predict the probability distribution of the next word or character in a text. By learning the statistical patterns of language, language models can automatically identify grammatical, syntactic, and semantic features of language, enabling text understanding and generation.

The earliest language models were based on N-gram statistical methods. With the development of deep learning technology, **neural network language models (NNLM)** have gradually become mainstream, including recurrent neural networks (RNN), Gate Recurrent Unit (GRU), long short-term memory networks (LSTM), Transformer, and other models. These models learn the relationships between words by training on large amounts of text data, enabling text modeling and prediction.

Language models have a wide range of applications. In NLP-related tasks, language models can be used for machine translation, text generation, information retrieval, speech recognition, and more. In the area of text generation, language models can be used for automatic writing and intelligent dialogue systems. Additionally, in information retrieval, language models can be applied to document classification, topic modeling, sentiment analysis, and other tasks.

In summary, language models are the foundation for understanding and generating natural language. They play a crucial role in various natural language processing tasks, and also lay the groundwork for the development of large language models, making them of great importance to the field of artificial intelligence.

2. Experiment Tasks Overview

In this project, I primarily conducted the following experiments:

1. I used different model structures to train neural network-based language models (the author tried various models including LSTM, GRU, RNN-TANH, RNN-RELU, and Transformer), and compared their performance differences.

2. I conducted ablation studies, tried different hyperparameters, and discussed their impact on the performance of the language models.

3. Use TensorBoard to plot the loss (or perplexity) trends on the training and validation datasets for each epoch during the training phase.

4. Optimize the perplexity on the test set, with the constraint that the total number of model parameters does not exceed 60M.

3. Perplexity

The commonly used evaluation metric for the quality of language models is **perplexity (PPL)**. Perplexity is a measure of the language model's ability to predict a language sample. The lower the perplexity value obtained on a dataset, the better the language model performs.

The calculation formula is as follows:

$$\begin{aligned} PPL(S) &= P(w_1, w_2, ..., w_N)^{-\frac{1}{N}} \\ &= log(-\frac{1}{N}\sum_{i=1}^{N}\log P(w_i|h)) \end{aligned}$$

among which N is the length of the sentence.

In the training of language models, we typically use the logarithmic form of perplexity: taking the logarithm of the probability at each position and then averaging.

The calculation formula is as follows:

$$\log(PPL(S)) = -\frac{1}{N}\sum_{i=1}^{N}\log P(w_i|h)$$

Perplexity is the geometric average of the reciprocal of the probabilities computed for each word. Intuitively, perplexity can be understood as the average branching factor, that is, the average number of choices the model has when predicting the next word.

4. Experiments

4.1. Different Network Architectures

First, try the impact of different network structures on the performance of the language model. Keep other options unchanged and only change the *model* options. It can be found that the performance of the LSTM network is the best.

Other options:

```
python main.py --cuda --emsize 800
--nhid 800 --dropout 0.5 --nlayers 3
--lr 10 --epochs 20 --model <model>
```

Among them, using too large a learning rate in the RNN model can lead to gradient explosion, so for the RNN model we use the following options:

```
python main.py --cuda --emsize 800
--nhid 800 --dropout 0.5 --nlayers 3
--lr 5 --batch_size 10 --epochs 20
--model <model>
```

The results are shown in Figures 1 and 2.

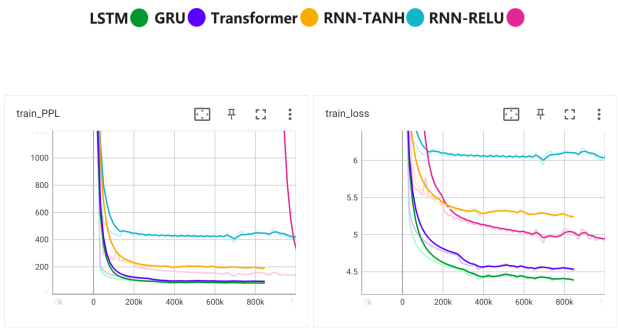

Figure 1. training PPL and Loss

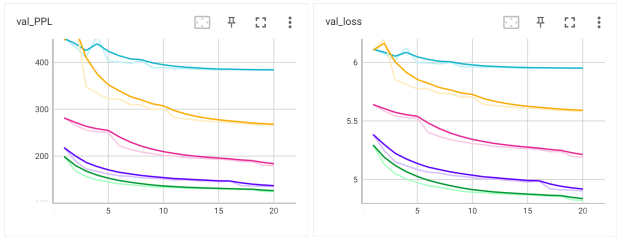

Figure 2. Validating PPL and Loss

4.2. Different *emsize* and *nhid*

emsize represents size of word embeddings, *nhid* represents number of hidden units per layer. Keeping other options unchanged, we let *emsize* and *nhid* vary synchronously, and tried the values 800 (baseline), 640, 1000, and 1080. We can see that the performance is best when $emsize = nhid = 1080$.

Other options:

```
python main.py --cuda --emsize <emsize>
--nhid <nhid> --dropout 0.5 --nlayers 3
--lr 10 --epochs 20 --tied
```

The results are shown in Figures 3 and 4.

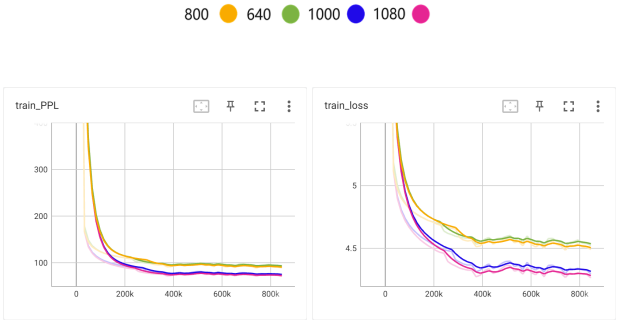

Figure 3. training PPL and Loss

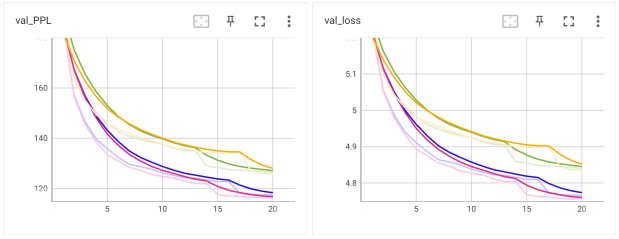

Figure 4. Validating PPL and Loss

4.3. Different Dropout

dropout means dropout applied to layers (0 = no dropout).

We selected the values 0.5, 0.35, and 0.65 to compare. We can see that the model performance is optimal when $dropout = 0.35$.

Other options:

```
python main.py --cuda --emsize 800
--nhid 800 --dropout <dropout> --nlayers 3
--lr 10 --epochs 20 --tied
```

The results are shown in Figures 5 and 6.

4.4. Option *tied*

We turned on the *tied* option, which means tying the word embedding and softmax weights.

I tried the model with and without the *tied* option, and found that the model performance was better when using the *tied* option.

Other options:

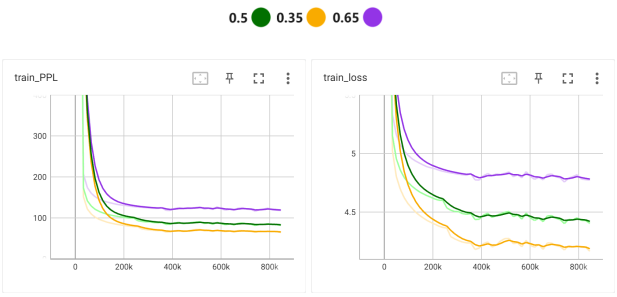

Figure 5. training PPL and Loss

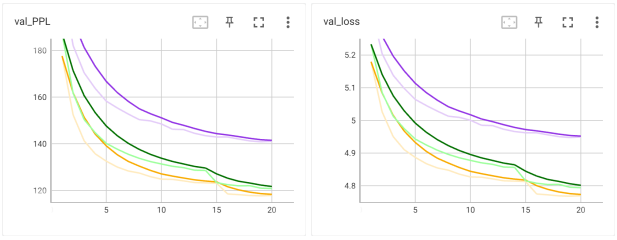

Figure 6. Validating PPL and Loss

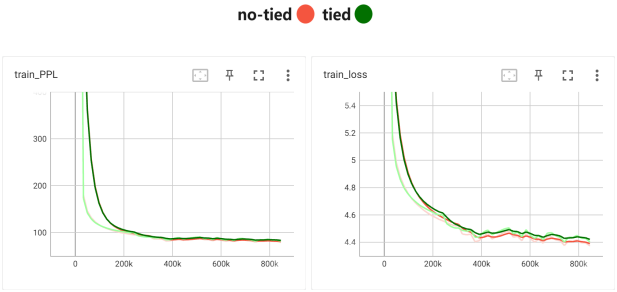

Figure 7. training PPL and Loss

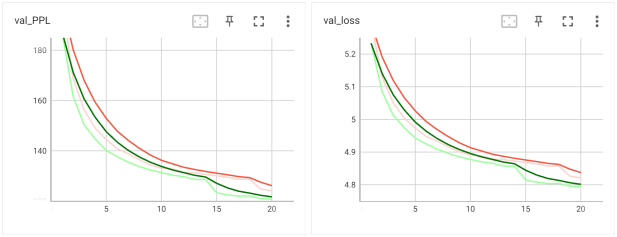

Figure 8. Validating PPL and Loss

```
python main.py --cuda --emsize 800
--nhid 800 --dropout 0.5 --nlayers 3
--lr 10 --epochs 20 <--tied>
```

The results are shown in Figures 7 and 8.

5. Best Result

Use the command:

```
python main.py --cuda --emsize 1080
--nhid 1080 --dropout 0.35 --nlayers 3
--lr 10 --epochs 20 --tied
```

That is, use the LSTM network model, with $emsize$ and $hnid$ both set to 1080, $dropout$ set to 0.35, initial learning rate set to 10, and the $tied$ switch turned on, we trained the model for 20 epochs. The model size is approximately 57.97M.

We were able to obtain the **optimal language model**, and its performance on the test dataset is as follows:

Table 1. Best Result on Testing Dataset

Loss	4.75	**PPL**	115.31