School of Data and Computer Science, Sun Yat-sen University

Language Model

18308045 Zhengyang Gu

December 23, 2020

Contents

1 Abstract

The language model is implemented through LSTM and the effects of hyperparameters is discussed in the report.

2 Background

2.1 LSTM

The LSTM was introduced in the last report. Briefly, the most advantage of it over the simplest RNN is that it can forget like humans do. However, I do not use the BiLSTM here but LSTM to implement the language model, because when generating a new word, the words after the new word is unkown and should not be considered.

2.2 Cross Entropy

The typical form of cross entropy is given by:

$$H(p, q) = -\sum_{x} p(x) \log(q(x))$$

where the x an assignment of a feature, p is the actual probability distribution of the feature and the q is the predicted probability distribution of the feature. When the two distribution are similar, the cross entropy between them is low.

3 Implementation

3.1 Training

The idea behind the language model is simple. The feature here is the current word and the domain of it is all of the words that appear in the training set. The model is fed with the last hidden state and the last word of the sentence and generate a predicted probability distribution of the word which has the form of vector filled with probabilities and a new hidden state. And we choose the word with the highest probability to be the current predicted word. The loss here is the cross entropy between the actual probability distribution which has the form of one-hot vector and the predicted probability distribution. We want to minimize the loss to make the predicted probability distribution be as similar to the actual probability distribution as possible in order to get a humman-like prediction of the current word based on words in the sentence before the current word.

```
hidden_batch = None
output_batch = model(inputs_batch.to(device), hidden_batch)[0]
loss = cross_entropy_loss(output_batch.reshape(-1, len(ix2word)),
                                                    outputs_batch.to(device).reshape
                                                        (-1))
loss.backward()
```

3.2 Testing

I use the average loss of the every sentence in the training set but not the accuracy to observe its convergence. The reason is that our goal is to get a prediction that is humman-like which means we just need to get a relatively similar probability distribution of the current word but not a predicted word that is completely same as the word in the training set, and the accuracy can be very slow. And I test the loss sperately. It is because here I exploit the batch to speed up the training process, and it's hard to average the loss calculated during this process because of padding. Though it costs a lot of time to calculate the loss with batch size 1, there's no need to calculate the gradient of the loss and to update weights so it's not as time-consuming as the training with batch size 1.

Additionaly, I wrote a script which get a word as the input and output a sentence. We can evaluate the model by just evaluate how fluent the sentence is.

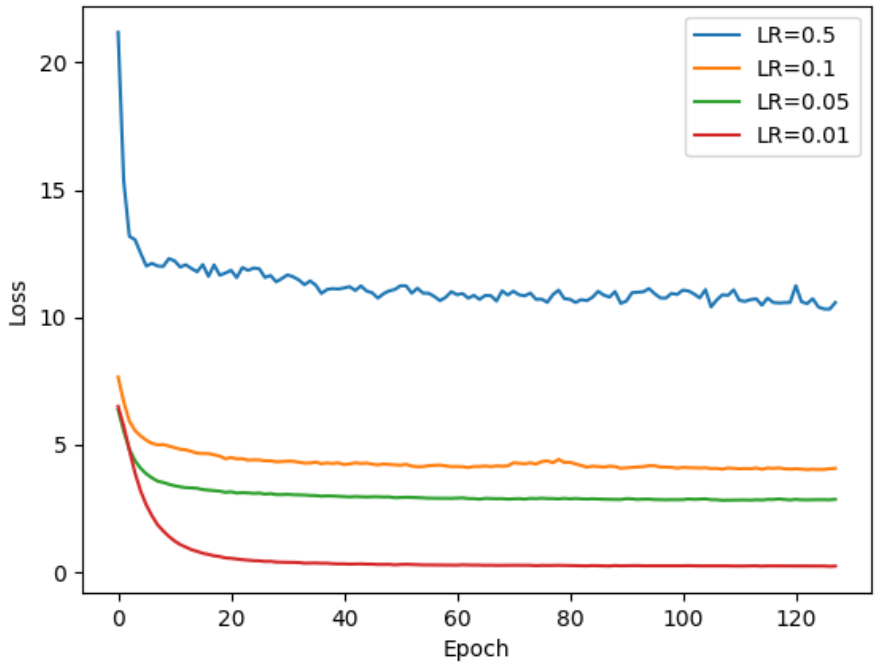
4 Result

4.1 Small Training Set

I first trained the model on a training set with only 3000 sentences. The hyperparameters I set are shown as follows.

```
BATCH_SIZE = 32
EMBEDDING_DIM = 128
HIDDEN_DIM = 128
```

And I tested different learning rate. The result is as follows.



The result shows that the model trained with the learning rate of 0.01 has the best performance.

I inputed some common words to the model trained with the learning rate of 0.01 and with 128 epochs. The result it generated is as follows.

```
输入一个词：我们
我们期待有更多这样的作品出版。
输入一个词：人们
人们步入其间，可尽情领略返璞归真，回归自然的奇妙之感。
输入一个词：它
它不涉及高深科研，不要精确计算。
输入一个词：现在
现在，我们要战胜面临的困难，抓住发展的机遇，完成党的十五大所
确定的各项战略任务，就必须下大气力把作风建设好。
输入一个词：终于
终于担忧的是，赤水河流域的一些地方领导正积极要求在赤水河上修
水坝，建电站。
```

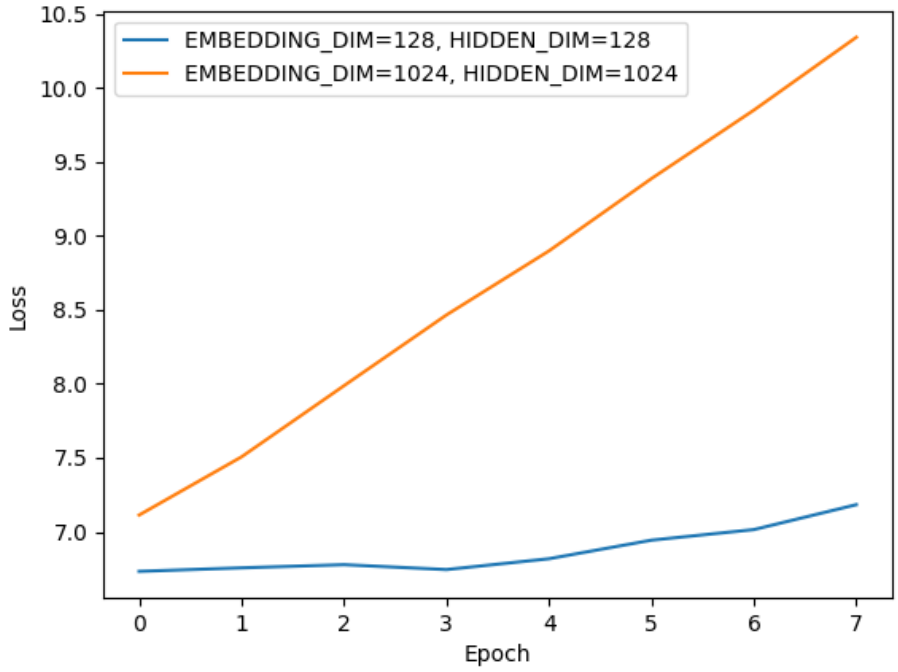
These sentences are quite fluent, but some of them are exactly same as sentences in training set. The reason of it may be that the size of training set is too small.

4.2 Large Training Set

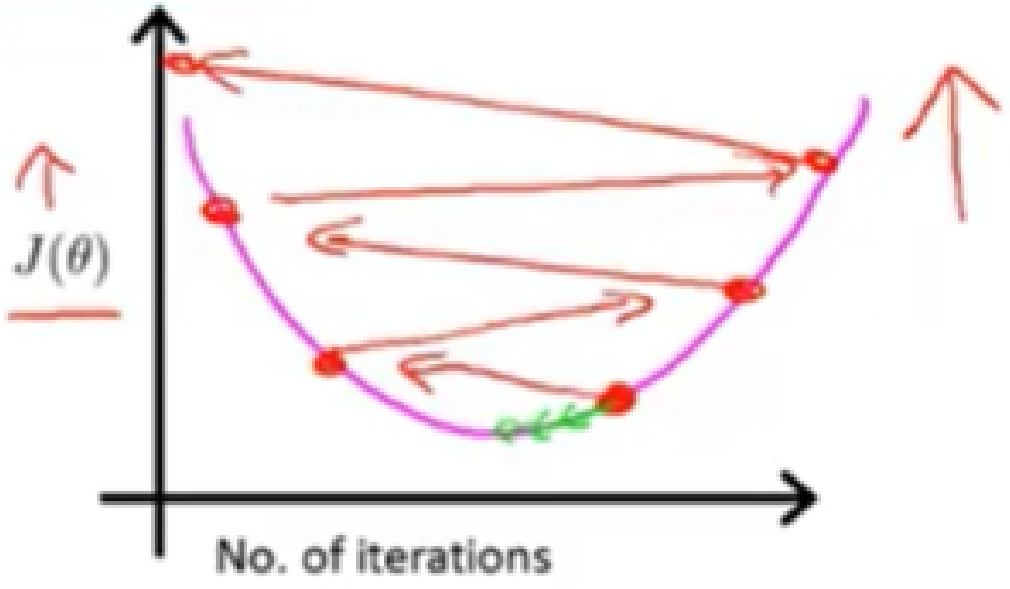
Then I trained the model on the whole training set which has almost 100000 sentences. The hyperparameters I set are shown as follows.

```
BATCH_SIZE = 16
EMBEDDING_DIM = 128
HIDDEN_DIM = 128
```

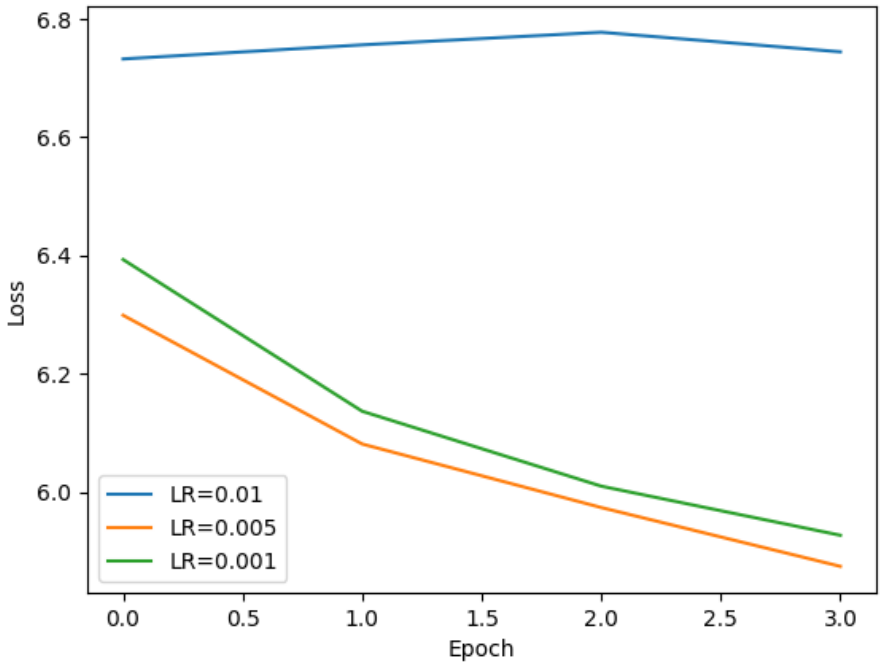
But the loss of it ascend which is quite abnormal. At first I thought the reason might be that the number of words got larger while the size of the word embedding and the hidden state do not got larger. Therefore I tried enlarging the size of the word embedding the hidden state. However, the loss ascend even faster.



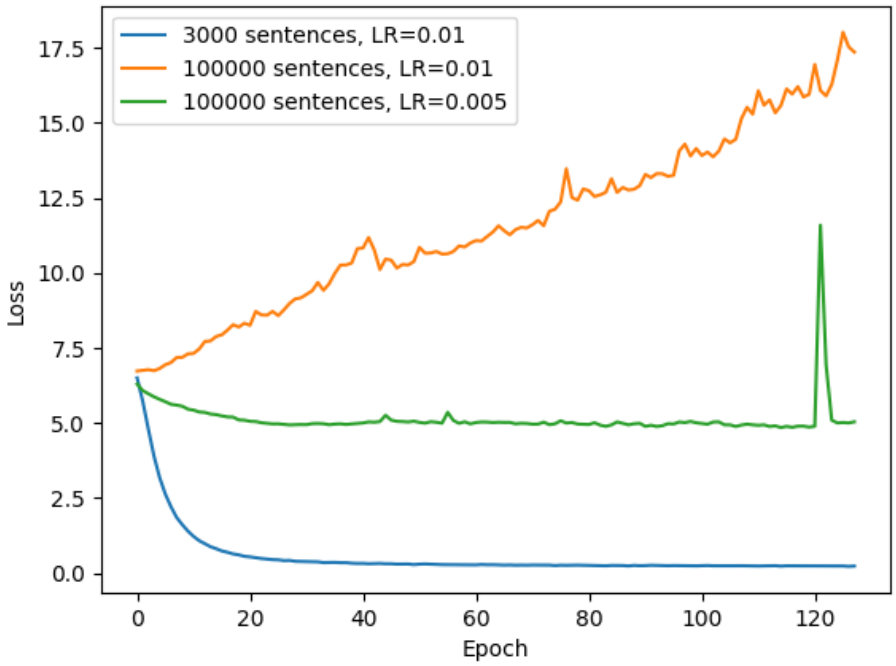
Then I thought that maybe it was the learning rate that is so small that cause the overshooting problem.



Therefore I tried decrease the learning rate. The loss finally went down.



Then I used the learning rate of 0.005 to proceed the training.



The loss's sudden ascent maybe mean that it just miss a locally optimal solution. Therefore I inputed some words in the model trained with 120 epochs which is a relatively good solution which is also relatively close to the locally optimal solution.

```
输入一个词：我们
我们相信，这一问题上孕育着当今世界上显露。
输入一个词：人们
人们不禁感慨地说：“我就是自愿的时候，你及早征服茫然。
输入一个词：它
它是担心，这一问题上找出答案。
输入一个词：现在
现在，分工与合作绝非仅仅是老子，但倘每日上了大人们的心灵素养
。
输入一个词：终于
终于有天然屏障，防火层层层层层层层层层层层层层层层层层层层层层
层层层层层层层层层层层层层层层
```

These sentences are absolutely not from training set but they are not fluent as the sentences generated by the model trained with 3000 sentences.

5 Conclusions And Bonuses I May Get

1. I implemented a language model using LSTM.

2. I use batch and GPU to speed up the training process.
3. I discussed the effects of hyperparameters.
4. I compose my report in English, although it may be not fluent.

Future work may include the experiment with more running epochs.