Natural Language Processing, Assignment 2

liwentao, 2020013210

October 18, 2022

Disclosure

Please disclose the following cases if any:

1. If you received any help whatsoever from anyone in solving this assignment, give full details.
2. If you gave any help whatsoever to anyone in solving this assignment, give full details.
3. If you found or came across code that implements any part of this assignment, give full details.

Warning: You should do your own work. Copying solutions (text or code) from any external sources is strictly prohibited.

Problem 1 Train and Test an LSTM Language Model

In this problem, I train a LSTM model in ptb-train dataset and evaluate it on the ptb-valid dataset. The process can be divided into three parts.

1.Data processing

In the data processing part, at first I use the torchtext to build the look-up table(**word2idx and idx2word**), in order to form the relationship between word and index. Then I merge all the sentence in the train dataset and valid dataset separately to bulid the two long sentence. For the long sentence, each time I select sequence-length words and repeat batch-size times to build the **batch**. For each mini sentence, for example, ABCD, I use BCDE as the **ground truth(next-sentences)**.

2.LSTM model

In the model part, I use a LSTM model with a embedding layer(using **glove-300d** to initialize), a LSTM layer(**embed-size=300** , **hidden-size=832**), two dropout layers with p=0.71 and a Dense layer(**input-size=832** , **output-size=10002**). The output of the model will pass by a log-softmax function in order to calculate the perplexity and cross entropy.

3.Training and evaluation detail

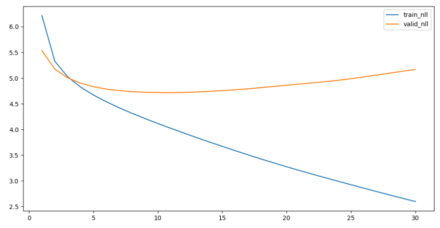
In the training part, I use three methods as required(shuffle-zero,continuous-zero,continuous-former). In each batch, I calculate the average nll loss and perplexity, and take average of them as average loss and perplexity in an epoch.

4.Hyperparameters

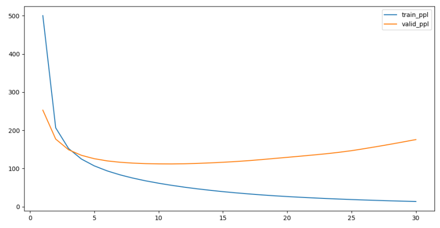
embed-size	300
hidden-size	832
num-layers	1
num-epochs	30
batch-size	256
learning-rate	0.006
seq-length	10
lstm-dropout	0.5
$P_{dropout-layer}$	0.71

5.Training and evaluation result

I present the cross-entropy loss (NLL+logSoftmax) and the perplexity graph for each method.

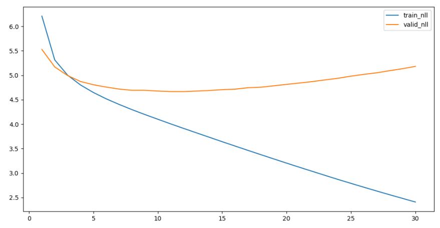


(a) cross-entropy loss (NLL+logSoftmax)

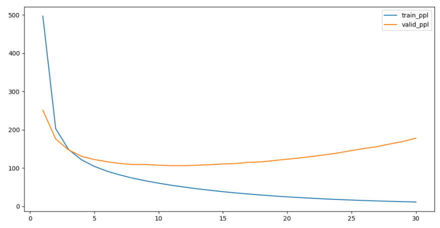


(b) perplexity

Figure 1: continuous batching and zero vector initialize

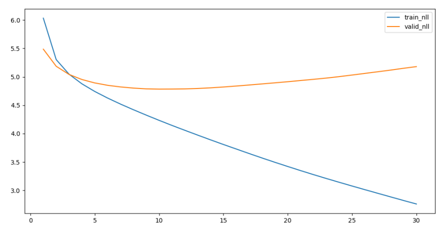


(a) cross-entropy loss (NLL+logSoftmax)

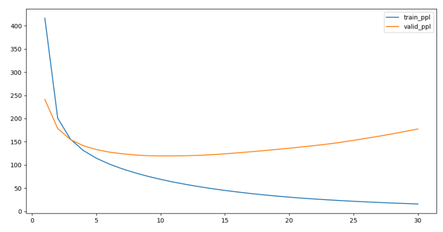


(b) perplexity

Figure 2: shuffle batching and zero vector initialize

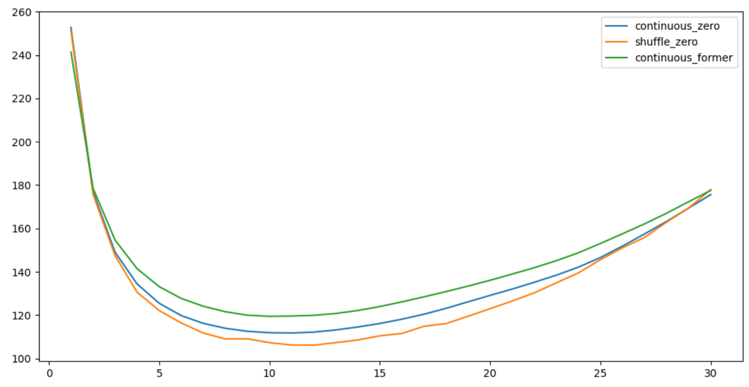


(a) cross-entropy loss (NLL+logSoftmax)



(b) perplexity

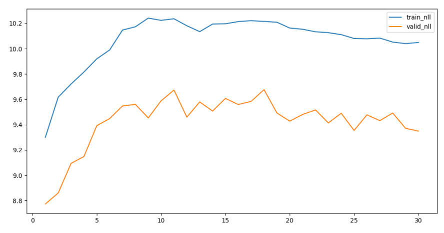
Figure 3: continuous batching and former vector initialize



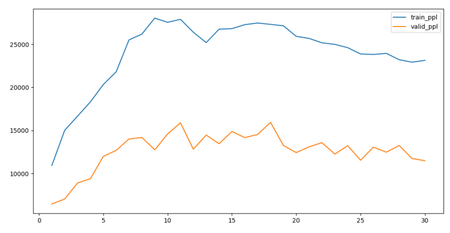
(a) perplexity

Figure 4: compare perplexity of three method

Comparing these diagrams, we find that the shuffle batching method is superior to other method. And we draw a conclusion that the model is overfitting (valid curve rise up). So we use the shuffle-zero method in RNN-mode, and we get the result here.



(a) cross-entropy loss



(b) perplexity

Figure 5: rnn model performance using shuffle batching

In the diagram, the perplexity of RNN language model is much larger than LSTM, this is simply because the RNN can not handle long dependency, and suffer the gradient varnishing problem.

Problem 2. Generation

I calculate three model (1.shuffle no dropout 2.shuffle dropout 3.shuffle dropout 2 layers) loss and perplexity. In the concrete implementation, for each sentence I use top 40 words as alternative thesaurus. And randomly generate the token.

1. shuffle no dropout 1 layer: 10.26 , test-ave-ppl: 28669
2. shuffle dropout 1 layer: test-ave-nll: 9.039 , test-ave-ppl: 8431
3. shuffle dropout 2 layers: test-ave-nll: 9.397 , test-ave-ppl: 12058

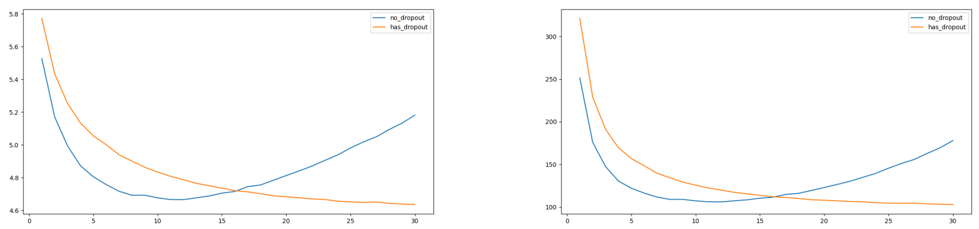
In the specific scenario, we use the model(shuffle dropout 1 layer) to generate the sentence. The examples are as follow, the first one is prefix sequences, the second one is generate sentence(generate 20 words).

1. **the agreements bring to a total of**: the agreements bring to a total of about five million shares and other issues of the company 's common stock for the quarter $< eos >$ the new.
2. **i think this kind of market**: i think this kind of market $< eos >$ but in the past two months the two sides have been able to buy the company 's shares.
3. **it is widely accepted that**: it is widely accepted that the company has to be on the new york stock exchange $< eos >$ the stock market is the first of.

Problem 3. Make it better

1.Use Dropout

I add dropout layers in model, specifically, I use the LSTM layer which dropout=0.5, and both the embedding and the output of feed forward layer will pass a dropout layer. The model performance is persented below.



(a) cross-entropy loss (b) perplexity

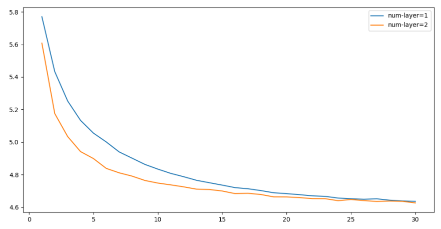
Figure 6: model with and without dropout layer

We can see that dropout layer significantly solve the overfitting problem, the valid perplexity did not rise up as the epoch increase.

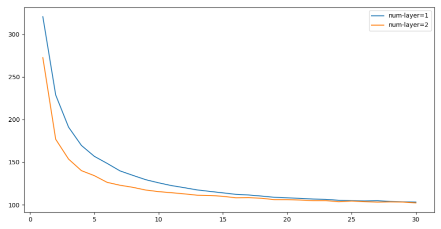
In fact, using dropout layer is equal to average a lot of models. Because in each step, the activated neurons are different, and the model is different. So this method can improve the generalization ability of the model.

2.Add multiple LSTM layers

I try to change the layer number in model, specifically, I increase the num-layer from 1 to 2. The model performance is persented below.



(a) cross-entropy loss



(b) perplexity

Figure 7: model with and without dropout layer

Intuitively, increasing the layer number can improve the fitting ability of the model, leading to a better performance.

3.Other try

I also try the learning rate decay using **pytorch scheduler** and the **early stop method** (if the valid perplexity constantly rise up in three epoch, the training will stop), it turns out that if we do not use the dropout method, these tricks have good effect, but do not increase the model performance significantly(only prevent it to decrease); and if we use dropout, these tricks are not necessary.

4.Best performance

shuffle dropout 2 layers with perplexity = 102.10.