Natural Language Processing, Assignment 2

October 5th, 2022

1 Introduction

You will be training a Recurrent Neural Network (RNN) on the Penn Treebank language modeling dataset with PyTorch. You will learn how to use an RNN to model and generate text. You will also learn about the various techniques we use to regularize recurrent networks and improve their performance.

You are required to finish this homework in the Python (version 3) programming language. We recommend that you look through all of the problems before attempting the first one. However we do recommend you complete the problems in order, as questions often rely on the completion of previous questions.

You should submit the report, your code, a readme file for the code, and the scripts that can reproduce all your experiments at the same time in a .zip file!

2 Train and Improve an Language Model with PyTorch

The following subsections will describe the dataset and what your model is expected to do. You are free to structure the training and engineering of your model as you see fit. In section 2.1, we will first train a language model with a vanilla RNN and an LSTM. In section 2.2, we will try to improve the performance with additional techniques.

2.1 Training a Language Model with RNNs and LSTMs

Dataset and Preprocessing The Penn Treebank language modeling dataset is included in the folder `penn-treebank` and includes three files—"ptb.train.txt", "ptb.valid.txt", and "ptb.test.txt", which should be used for training, validation (e.g., checkpoint selection and hyperparameter tuing), and test respectively.

Word tokens can be obtained by splitting the original data using white spaces. The vocabulary is defined as all the word types that occur in the three files, including a word type "<UNK>" that represents rare words. In addition, we will include another word type "<eos>". You should read the training, validation, and testing files, and replace newlines with the "<eos>" token.

DataLoader To make the most out of our data, we need to make sure the sequences we feed into the model are different each step. You can use PyTorch's DataLoader class but overwrite the "iter" method. Implementing a data loader without PyTorch is also acceptable. The input to the model in a language model is an input text sequence, and the output of the model is the sequence shifted by one token. Please prepare the batched inputs and targets as follows:

1. Concatenate all articles in the training set into one long sequence.
2. Generate batches with shape ($batch\_size$, $sequence\_length + 1$). There are two ways of batching and we will compare them in subsequent problems.
   Option 1: Continuous Batching. Split the long sequence into $batch\_size$ equal-length sequences

(these split sequences are also fairly long sequences). For each split sequence, cut them into continuous parts. Batch k is composed of the k^{th} part of all the sequences. Note that the samples in adjacent batches should be continuous.
For example, assume the i^{th} sequence is "I eat a banana and an apple everyday including today.", and $sequence\_length = 3$. The i^{th} sample in first two batches should be "I eat a banana" and "banana and an apple" (note the shifting here). With this method, it is suggested to shuffle the articles before concatenation in step 1 between epochs.
Option 2: Shuffled Batching. Split the long sequence into sequences with length $sequence\_length+1$. Each batch consists of $batch\_size$ different sequences. Note that adjacent batches might not be continuous, and you should shuffle them.

3. Within each batch, construct the inputs and outputs (targets). To be specific, the inputs should be the batched sequences without the last token, and the outputs (targets) should be the batched sequences without the first token. For example, assume the i^{th} sample in the batch is "I eat a banana", then the input should be "I eat a", and the output should be "eat a banana". This right-shift operation corresponds to predicting the next word given the history.

Word Embedding Initialization You should initialize the word embeddings with Glove[1]. For words not found in Glove, use random initializations.

Training and Generating Now, we have stated the important issues in the implementation, let's train an RNN and generate text with it!

Problem 1 : (40 points) **Train and Test an LSTM Language Model** Recall that the mathematical formulation of the LSTM is

$$i_t = \sigma(W^i x_t + U^i h_{t-1} + b^i) \tag{1}$$
$$f_t = \sigma(W^f x_t + U^f h_{t-1} + b^f) \tag{2}$$
$$o_t = \sigma(W^o x_t + U^o h_{t-1} + b^o) \tag{3}$$
$$\tilde{c}_t = tanh(W^c x_t + U^c h_{t-1} + b^c) \tag{4}$$
$$c_t = i_t \odot \tilde{c}_t + f_t \odot c_{t-1} \tag{5}$$
$$h_t = o_t \odot tanh(c_t) \tag{6}$$

where $\left[W^i, W^f, W^o, U^i, U^f, U^o\right]$ are weight matrices, $\left[b^i, b^f, b^o, b^c\right]$ are bias terms, x_t is the vector input at timestep t, h_t is the current hidden state, c_t is the memory cell state, and $\odot$ is element-wise multiplication. Firstly, you should build the vocabulary according to the dataset and construct the dataloader. Secondly, construct the model and initialize the word embeddings and other parameters. You can use the PyTorch LSTM APIs. Both the embedding size and the hidden size are recommended to be at least 128. Thirdly, you should train an LSTM model for several epochs, and plot the training and validation loss curves during the training process. Finally, you should test its performance on the test set with the best hyper-parameters selected by the validation set. Your model will likely achieve a reasonable NLL loss and perplexity (lower than 100) after training for 30 epochs.

An important implementation detail is whether the init hidden state of a batch should be the same as the last hidden state of the previous batch. Read the description of DataLoader in Section 2.1 carefully, and compare the following three cases:

1. Use shuffled batching, and set the first hidden state as a zero vector.
2. Use continuous batching, and set the first hidden state as a zero vector.

[1] https://nlp.stanford.edu/projects/glove/

3. Use continuous batching, and use the last hidden state of the previous batch as the first hidden state of the current batch. (Note that in continuous batching, the last batch and the current batch contain continuous text samples. This mimics the Truncated BPTT algorithm.)

The following contents should be reported:

1. Please describe the details which you think are important in your experiment, for example, the hyper-parameters you choose. Describe briefly the structure of your code, and the effect of classes or functions in your code.
2. For each of the three cases above, plot the training and validation loss curves during the training process.
3. For each of the three cases above, report the test NLL, the validation NLL, the test perplexity, and the validation perplexity. Note that the values reported on the test dataset should be similar to that you calculate on the validation dataset. If these values differ greatly, you are likely to have a bug in your code.
4. Compare the results of the three cases with analysis. Discuss their pros and cons.
5. Using the batching method with the best performance, replace the LSTM architecture with a vanilla RNN. Plot the training and validation loss curves. Report the NLL and perplexity on both the test and validation sets. Compare the results of vanilla RNNs and LSTMs with analysis.

Problem 2 : (20 points) **Generation**

To make use of the language model in a real scenario, you will implement a function that can generate a sequence. Use the best model from Problem 1. Please use top-k sampling for text generation with $k = 40$. Specifically, each time you sample from the top-40 words with the highest probabilities, and the sampling probabilities should be proportional to their corresponding predicted probabilities.

The following contents should be reported:

1. For each sentence in the test set, select the first 5 tokens as the context (discard sentences shorter than 5 tokens), and then continue generating 15 tokens. Compute the NLL loss and perplexity on the generated tokens.
2. Please report some cases that you think are interesting. The cases are not necessarily to be in the data set. For example, you could provide some prefix sequences to the model and make the model generate the following several words. Both good cases and bad cases are okay. Consider generating relatively long sequences.

2.2 Make it better!

Problem 3 : (40 points) Here are some optional techniques that may help boost your performance. We have concluded them into three types. Select some of them to make a better language model (You don't have to utilize all the following techniques). Certainly, we encourage you to stack more methods to get a better performance, which will help you learn better about the language model. Each trick will have a score, and the cumulative score will not exceed the total score of this problem.

1. Change the architecture of your network.
    (a) (20 points) Apply locked dropout between LSTM layers. To be specific, perform variational dropout on the hidden-to-hidden weight matrices $\left[U^i, U^f, U^o, U^c\right]$ within the LSTM (refer to the previous section for these notations). As the same weights are reused over multiple timesteps, the same weights remain dropped for the entirety of the forward and backward pass. Each example within the minibatch uses a unique dropout mask, rather than a single dropout mask being used over all examples, ensuring diversity in the elements

dropped out.

(b) (20 points) You can add multiple LSTM layers. Experiment with different sizes of hidden layers and different non linearities.

(c) (20 points) Weight Tying. Share the weights of the embedding and the output layer, which can reduce the total parameter count in the model and be viewed as a regularization method.

(d) (20 points) Variable length backpropagation sequences. You can do the following: first, select the base sequence length $base\_seq$ to be seq with probability p and $\frac{seq}{2}$ with probability $1-p$, where p is a high value approaching 1 (For example, 0.95). Then select the sequence length according to $N(base\_seq, s)$, where $base\_seq$ is the base sequence length and s is the standard deviation.

2. Try using different optimization schemes.

   (a) (20 points) Analyze the effect of early stopping with patience.

   (b) (20 points) Analyze the effect of learning rate decay.

   (c) (20 points) Analyze the effect of learning rate warmup.

3. Add regularization

   (a) (20 points) Apply activity regularization, which penalizes activations that are significantly larger than 0 as a means of regularizing the network. To be specific, add L2 normalization terms on the hidden states. It is defined as $\alpha L_2(h_t)$, where α is a scaling coefficient. If you apply dropout on the hidden states, it is written as $\alpha L_2(m_t \odot h_t)$, where m_t is the dropout mask.

   (b) (20 points) Apply temporal activity regularization, which penalizes the model from producing large changes in the hidden states. To be specific, add L2 normalization terms on the change of hidden states. It is defined as $\beta L_2(h_t - h_{t-1})$, where β is a scaling coefficient.

Submit a **detailed report** containing the description of approaches you tried to improve the performance of your language model. Also, discuss how those approaches affected the learning curves, performance on the validation set and test set, etc. Describe approaches that worked and that didn't work.