Project: Using CNN to train the LEDGAR set and do the profiling

Last updated: March 14, 2023

Goal

- Investigate basic CNN operations with PyTorch profiler
- Get familiar with LibMultiLabel Command Line Interface

Outline

1. Introduction
2. Project Contents

Outline

1 Introduction

2 Project Contents

Introduction I

- From the lecture, we've learned how CNN works on image classification.
- You might be interested in the actual running time of different operations.
- To understand this better, we're going to do PyTorch profiling on text data.

PyTorch Profiler I

- Pytorch Profiler is a tool for identifying the performance of PyTorch operators.
- It helps users understand a model's bottlenecks with metrics like CPU time.
- Let's take a look at a profiling result of nn.Conv1d:

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	# of Calls
aten::conv1d	0.00%	7.731ms	14.99%	90.734s	50.129ms	1810
aten::convolution	0.01%	43.094ms	14.99%	90.727s	50.125ms	1810
aten::_convolution	0.01%	83.385ms	14.99%	90.684s	50.101ms	1810
aten::mkldnn_convolution	12.90%	78.101s	12.92%	78.237s	43.225ms	1810
aten::contiguous	0.16%	998.145ms	6.60%	39.943s	7.356ms	5430

PyTorch Profiler II

- The call graph of `nn.Conv1d.forward` is:

```
nn.Conv1d.forward
-> nn.Conv1d._conv_forward
-> F.conv1d
-> aten::conv1d
-> aten::convolution
-> aten::_convolution
-> aten::mkldnn_convolution, and
   aten::contiguous
```

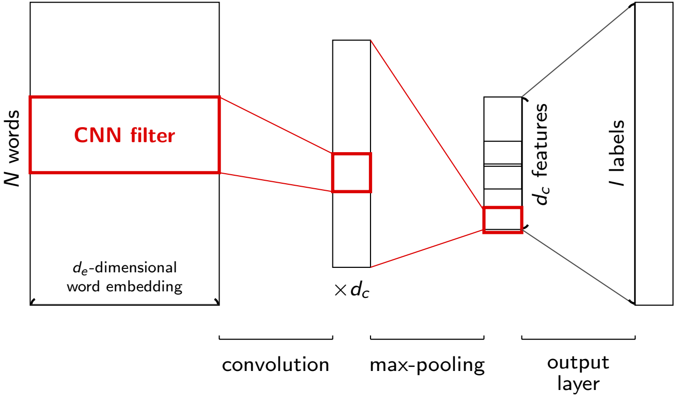
PyTorch Profiler III

- Therefore, the CPU total time of `aten::conv1d` roughly equals the sum of:
  - `aten::mkldnn_convolution`: 78.237s
  - `aten::contiguous`: 7.356ms (CPU time avg) * 1810 (`aten::_convolution`'s # of Calls) = 13.314s
- The prefix `aten::` refers to the tensor library ATen, the building blocks of PyTorch operators.
- Check out the source code of Convolution.cpp and mkldnn/Conv.cpp if you are interested in the details!

CNN for Text Data I

- The CNN architecture we used in LibMultiLabel is called KimCNN (Kim, 2014), which consists of a convolutional layer, a max pooling operation and a linear layer.

CNN for Text Data II



- The figure of KimCNN architecture is modified from Chen et al. (2022)

CNN for Text Data III

- Assume each document has the following word embeddings

$$X = \begin{bmatrix} \boldsymbol{x}_1 & \dots & \boldsymbol{x}_N \end{bmatrix} \in R^{d_e \times N},$$

  where d_e is the word-embedding dimension and N is the document length.
- That is, by some ways we have already obtained some information for each word

CNN for Text Data IV

- For any filter
$$\boldsymbol{v} \in R^{d_e \times k},$$
a convolutional operation is applied to a text region
$$[\boldsymbol{x}_n, \ldots, \boldsymbol{x}_{n+k-1}] \in R^{d_e \times k}$$
of k words
- It is like that we treat X as an image and horizontally extract sub-images

CNN for Text Data V

- Thus the following operation is conducted:

$$(h_n)_j = \sigma(\langle W_{1:d_e,1:k,j}, [\boldsymbol{x}_n, \ldots, \boldsymbol{x}_{n+k-1}]\rangle + b_j),$$

where $\boldsymbol{h}_n$ is the nth output vector, $\langle \cdot, \cdot \rangle$ is the component-wise sum of two matrices,

$$W_{1:d_e,1:k,j} \in R^{d_e \times k}$$

is the jth filter, and σ is an activation function.

CNN for Text Data VI

- Here
$$j = 1, \ldots, d_c$$
so d_c is the number of filters.
- The output after the convolutional operation is a matrix
$$H = \begin{bmatrix} \boldsymbol{h}_1 & \ldots & \boldsymbol{h}_{N-k+1} \end{bmatrix} \in R^{d_c \times (N-k+1)}$$
- Assuming the input is not zero-padded

Pooling I

- The maximum from each row of H is collected

$$g_i = \max_j H_{ij}$$

$$\boldsymbol{g} = \begin{bmatrix} g_1 & \dots & g_{d_c} \end{bmatrix}^\intercal \in R^{d_c}$$

- This naturally allows for variable document length N

Linear I

- The final layer is a linear layer

$$\boldsymbol{z} = A\boldsymbol{g} + \boldsymbol{c} \in R^{l}$$

where $A \in R^{l \times d_c}$ is the weights, $\boldsymbol{c}$ is the bias and l is the number of classes

Outline

1 Introduction

2 Project Contents

Project Description I

- In this project, you're going to investigate the CNN operations with PyTorch profiler.
- First, clone LibMultiLabel and checkout to branch `profiler` to see the code template.

```
git clone https://github.com/ASUS-AICS/\
LibMultiLabel.git
git checkout profiler
```

Project Description II

- Then, train a CNN model for 5 epochs without validation and test on the LEDGAR data set.

  ```
  python3 main.py --cpu --config \
  example_config/LEDGAR/kim_cnn.yml
  ```

- `example_config/LEDGAR/kim_cnn.yml` is the configuration file that we left blank in the code template. You can modify it from `example_config/rcv1/kim_cnn.yml`.

Project Description III

- For the network config, the hyperparameters are specified by the following arguments in the configuration file:
  - k: `filter_sizes`
  - d_c: `num_filter_per_size`
  - d_e: the word-embedding dimension depends on `embed_file`
    For the HW we use `glove.6B.300d`, where the dimension is 300

  while the rest you can refer to Command Line Options.

Project Description IV

- After the training, you will see a profiler log in `./runs/LEDGAR_kim_cnn*/profile.log`
- You can check
  - a demo video (00:42), and
  - the documentation (Command Line Interface)

  to understand how to set up the configuration file and force the process to run on a single CPU core.
- If it still takes time to set up, come to the TA hours!

Submission I

- Write a 2-page report about your observation on the profiler results.
    - Please analyze the CPU time of each operation of the forward pass. Based on the concept you learned in class, what is the most time-consuming operation of KimCNN? Are the results consistent with your understanding? Why?

Submission II

- How would changing `filter_sizes` and `num_filter_per_size` parameters affect the running time for forward passes? Is the root cause data-specific or affected mainly by the hyperparameters?

- Upload your report in PDF format to NTU Cool before 2023/04/04 23:59.

Misc I

- **Report**: For the above questions, there are no exact answers. We gave only a direction, and you can decide what you want to do.
- **Code**: Besides the template we provided, you are free to modify the code based on your design on the experiments. One thing to remind is the profiling overhead. Stepping into it may take extra hours to get the results.

References I

S.-A. Chen, J.-J. Liu, T.-H. Yang, H.-T. Lin, and C.-J. Lin. Even the simplest baseline needs careful re-investigation: A case study on XML-CNN. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2022. URL https://www.csie.ntu.edu.tw/~cjlin/papers/xmlcnn/xml_cnn_study.pdf.

Y. Kim. Convolutional neural networks for sentence classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, 2014. doi: 10.3115/v1/D14-1181.