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
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`:

<table>
<thead>
<tr>
<th>Name</th>
<th>Self CPU %</th>
<th>Self CPU</th>
<th>CPU total %</th>
<th>CPU total</th>
<th>CPU time avg</th>
<th># of Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>aten::conv1d</td>
<td>0.00%</td>
<td>7.731ms</td>
<td>14.99%</td>
<td>90.734s</td>
<td>50.129ms</td>
<td>1810</td>
</tr>
<tr>
<td>aten::convolution</td>
<td>0.01%</td>
<td>43.094ms</td>
<td>14.99%</td>
<td>90.727s</td>
<td>50.125ms</td>
<td>1810</td>
</tr>
<tr>
<td>aten::_convolution</td>
<td>0.01%</td>
<td>83.385ms</td>
<td>14.99%</td>
<td>90.684s</td>
<td>50.101ms</td>
<td>1810</td>
</tr>
<tr>
<td>aten::mkldnn_convolution</td>
<td>12.90%</td>
<td>78.101s</td>
<td>12.92%</td>
<td>78.237s</td>
<td>43.225ms</td>
<td>1810</td>
</tr>
<tr>
<td>aten::contiguous</td>
<td>0.16%</td>
<td>998.145ms</td>
<td>6.60%</td>
<td>39.943s</td>
<td>7.356ms</td>
<td>5430</td>
</tr>
</tbody>
</table>
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`
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!
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.
The figure of KimCNN architecture is modified from Chen et al. (2022)
Assume each document has the following word embeddings

\[ X = [x_1 \ldots x_N] \in \mathbb{R}^{d_e \times N}, \]

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
For any filter $\mathbf{v} \in \mathbb{R}^{d_e \times k}$, a convolutional operation is applied to a text region $[x_n, \ldots, x_{n+k-1}] \in \mathbb{R}^{d_e \times k}$ of $k$ words.

It is like that we treat $X$ as an image and horizontally extract sub-images.
Thus the following operation is conducted:

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

where $h_n$ is the $n$th output vector, $\langle \cdot, \cdot \rangle$ is the component-wise sum of two matrices,

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

is the $j$th filter, and $\sigma$ is an activation function.
Here

\[ j = 1, \ldots, d_c \]

so \( d_c \) is the number of filters.

The output after the convolutional operation is a matrix

\[
H = \begin{bmatrix}
    h_1 & \ldots & h_{N-k+1}
\end{bmatrix} \in \mathbb{R}^{d_c \times (N-k+1)}
\]

Assuming the input is not zero-padded
Pooling I

The maximum from each row of $H$ is collected

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

$$g = [g_1 \ldots g_{dc}]^T \in \mathbb{R}^{dc}$$

This naturally allows for variable document length $N$. 
The final layer is a linear layer

\[ z = Ag + c \in \mathbb{R}^l \]

where \( A \in \mathbb{R}^{l \times d_c} \) is the weights, \( c \) is the bias and \( l \) is the number of classes.
Outline

1. Introduction

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

```python
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`. 
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.
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 1

- 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 1

- **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.