

Project: Compare the running time of CNN operations using MATLAB and PyTorch

Last updated: May 2, 2023

Goal

- Do the running time analysis of our own MATLAB implementation with Pytorch.

Outline

- 1 Introduction
 - Background
 - From simpleNN to simpleNN-text
- 2 Project Contents

Outline

- 1 Introduction
 - Background
 - From simpleNN to simpleNN-text

- 2 Project Contents

Background I

- In the previous project, you were asked to analyze the CPU time of KimCNN's forward operations.
- We want to move on to the backward pass. However, it's hard to trace how the gradient of convolution operation is computed on PyTorch.
- In this project, we'll use [simpleNN-text](#), which explicitly runs operations derived in our lectures for forward and backward calculation.

Background II

- simpleNN-text is modified from [simpleNN](#), which implements CNN models for image classification. To compare with LibMultiLabel used for text classification, we modify simpleNN to run the KimCNN model.
- simpleNN-text is a Matlab based implementation, so we can use the Matlab profiler to check the cost of each operation.

Background III

- simpleNN was used previously in this course, but simpleNN-text was only recently generated for this project (after many days of hard work by our TAs). Thus you can help to see if there's any issue in the code. Also the final project of Newton's method for text classification may consider using this code.
- Let's recap the operations for forward/backward passes.

Major operations I

- Forward:

$$\begin{aligned} S^{m,i} &= W^m \text{mat}(P_{\phi}^m P_{\text{pad}}^m \text{vec}(Z^{m,i})) \\ &= W^m \phi(\text{pad}(Z^{m,i})) \end{aligned} \quad (1)$$

$$Z^{m+1,i} = \text{mat}(P_{\text{pool}}^{m,i} \text{vec}(\sigma(S^{m,i}))) \quad (2)$$

- The difference between Conv2d and Conv1d is that $a_{\text{conv}}^m = 1$ in the latter case. Thus,

$$W^m \in R^{d^{\text{out}} \times h a_{\text{pad}}^m} \quad \text{and} \quad S^{m,i} \in R^{d^{\text{out}} \times b_{\text{conv}}^m}.$$

Major operations II

- Backward:

$$\begin{aligned} & \frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T} \\ &= \left(\frac{\partial \xi_i}{\partial \text{vec}(Z^{m+1,i})^T} \odot \text{vec}(I[Z^{m+1,i}])^T \right) P_{\text{pool}}^{m,i} \end{aligned} \quad (3)$$

$$\frac{\partial \xi_i}{\partial W^m} = \frac{\partial \xi_i}{\partial S_{m,i}} \phi(\text{pad}(Z^{m,i}))^T \quad (4)$$

$$\frac{\partial \xi_i}{\partial \text{vec}(Z^{m,i})^T} = \text{vec} \left((W^m)^T \frac{\partial \xi_i}{\partial S_{m,i}} \right)^T P_{\phi}^m P_{\text{pad}}^m, \quad (5)$$

Major operations III

- In sum, we want to implement the following efficiently:
 - generating the linear indices of $\phi(Z^{\text{in},i})$ (used in (1) and (2))
 - matrix-matrix products (used in (1), (3) and (4))
 - $v^T P$ for (used in (3) and (5))

Data storage I

- Similar as lecture slides, we store $Z^{m,i}$, $\forall i = 1, \dots, l$ as the following matrix.

$$\begin{bmatrix} Z^{m,1} & Z^{m,2} & \dots & Z^{m,l} \end{bmatrix} \in R^{d^m \times a^m b^m l}. \quad (6)$$

- We're not sure whether there's a better way for data storage. Discussion about this can be part of your report.
- With this method to store the whole Z , we then discuss how to generate $\phi(\text{pad}(Z^{\text{in},i}))$ used in Conv1d.

The padding issue I

- In LibMultiLabel, we pad 0's at the end so that all instances have the same length as the longest instance in a batch. However, the length among batches may be different.
- An example:
 - Batch 1: three instances with length 5, 10 and 100. After padding, all of three have length 100.
 - Batch 2: three instances with length 10, 20 and 50. After padding, all of three have length 50.

The padding issue II

- To simplify the problem, for the project we provide a dataset with length 100 for all instances, so you can remove the matlab code for padding.

Generate $\phi(Z^{\text{in},i})$ for Conv1d I

- Suppose the matrix $Z^{\text{in},i}$ has height $a^{\text{in}} = 1$, width b^{in} and the number of channels d^{in} :

$$Z^{\text{in},i} = \begin{bmatrix} z_{1,1,1}^i & z_{1,2,1}^i & \cdots & z_{1,b^{\text{in}},1}^i \\ \vdots & \vdots & \ddots & \vdots \\ z_{1,1,j}^i & z_{1,2,j}^i & \cdots & z_{1,b^{\text{in}},j}^i \\ \vdots & \vdots & \ddots & \vdots \\ z_{1,1,d^{\text{in}}}^i & z_{1,2,d^{\text{in}}}^i & \cdots & z_{1,b^{\text{in}},d^{\text{in}}}^i \end{bmatrix}.$$

- In KimCNN for text classification, the b^{in} and d^{in} correspond to the document length and the dimension of word embeddings respectively.

Generate $\phi(Z^{\text{in},i})$ for Conv1d II

- As what we have done in the lecture, we count elements in a column-oriented way.
- This leads to the linear indices for $Z^{\text{in},i}$:

$$\begin{bmatrix} 1 & d^{\text{in}} + 1 & \dots & (b^{\text{in}} - 1)d^{\text{in}} + 1 \\ 2 & d^{\text{in}} + 2 & \dots & (b^{\text{in}} - 1)d^{\text{in}} + 2 \\ \vdots & \vdots & \ddots & \vdots \\ d^{\text{in}} & 2d^{\text{in}} & \dots & (b^{\text{in}})d^{\text{in}} \end{bmatrix} \in R^{d^{\text{in}} \times b^{\text{in}}}. \quad (7)$$

Generate $\phi(Z^{\text{in},i})$ for Conv1d III

- Our goal here is to generate the linear indices for the following matrix

$$\phi(Z^{\text{in},i}) = \begin{bmatrix} z_{1,1,1}^i & z_{1,1+s,1}^i & & z_{1,1+(b^{\text{out}}-1)s,1}^i \\ z_{1,2,1}^i & z_{1,2+s,1}^i & & z_{1,2+(b^{\text{out}}-1)s,1}^i \\ \vdots & \vdots & \dots & \vdots \\ z_{1,h,1}^i & z_{1,h+s,1}^i & & z_{1,h+(b^{\text{out}}-1)s,1}^i \\ \vdots & \vdots & & \vdots \\ z_{1,h,d^{\text{in}}}^i & z_{1,h+s,d^{\text{in}}}^i & & z_{1,h+(b^{\text{out}}-1)s,d^{\text{in}}}^i \end{bmatrix} \in \mathbf{R}^{hd^{\text{in}} \times b^{\text{out}}},$$

where h is the filter size.

Generate $\phi(Z^{\text{in},i})$ for Conv1d IV

- To generate the first column for channel j , we first list the selected values and their linear indices:

| values | linear indices in $Z^{\text{in},i}$ |
|-------------|-------------------------------------|
| $z_{1,1,j}$ | j |
| $z_{1,2,j}$ | $d^{\text{in}} + j$ |
| \vdots | \vdots |
| $z_{1,h,j}$ | $(h-1)d^{\text{in}} + j$ |

Generate $\phi(Z^{\text{in},i})$ for Conv1d V

- If $s = 1$, indices can be written as

$$\begin{bmatrix} 0 \\ \vdots \\ h-1 \end{bmatrix} d^{\text{in}} + 1 \quad (8)$$

- To get the indices of the first column for all channels, we compute the outer sum

$$(8) + [0 \ 1 \ \dots \ d^{\text{in}} - 1] \quad (9)$$

Generate $\phi(Z^{\text{in},i})$ for Conv1d VI

- Now we're going to generate the column offset for each sub-image.
- The starting indices for all sub-images are given by

$$1, 1 + sd^{\text{in}}, 1 + 2sd^{\text{in}}, \dots, 1 + (b^{\text{out}} - 1)sd^{\text{in}}.$$

- Thus, the column offset is

$$\begin{bmatrix} 0 & \dots & b^{\text{out}} - 1 \end{bmatrix} sd^{\text{in}} \quad (10)$$

Generate $\phi(Z^{\text{in},i})$ for Conv1d VII

- Finally, the indices to extract elements in $Z^{\text{in},i}$ for $\phi(Z^{\text{in},i})$ can be obtained via computing the outer sum of the column offsets and the first column indices.

$$(10) + \text{vec}((9)). \quad (11)$$

- According to the derivation, you can implement your own `find_index_phiZ(a, b, d, h, s)` for the purpose of 1D convolution and maxpooling.

Loss function I

- Let $x \in R$ be one of the output in the last layer and $y \in \{0, 1\}$ be the corresponding label. In LibMultiLabel, the default loss function is the binary cross entropy loss with logits, given by

$$\xi(x, y) = -[y \log \sigma(x) + (1 - y) \log(1 - \sigma(x))], \quad (12)$$

where

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

is the sigmoid function.

Loss function II

- However, in the [source code of pytorch](#), it is computed by

$$\xi(x, y) = (1 - y)x + \text{ReLU}(-x) + \log \left(e^{-\text{ReLU}(-x)} + e^{-x - \text{ReLU}(-x)} \right), \quad (13)$$

for numerical stability.

- We will first show the equivalence. Then we discuss why this is numerically helpful.

Loss function III

- A simple derivation for the equivalence:

$$\begin{aligned}\xi(x, y) &= - \left[y \log \left(\frac{1}{1 + e^{-x}} \right) + (1 - y) \log \left(\frac{e^{-x}}{1 + e^{-x}} \right) \right] \\ &= - y \log \left(\frac{1}{1 + e^{-x}} \cdot \frac{1 + e^{-x}}{e^{-x}} \right) \\ &\quad - \log(e^{-x}) + \log(1 + e^{-x}) \\ &= - yx + x + \log(1 + e^{-x})\end{aligned}\tag{14}$$

Loss function IV

We further have

$$\begin{aligned} & (14) \\ & = (1 - y)x + \log(e^{\text{ReLU}(-x)}(e^{-\text{ReLU}(-x)} + e^{-x - \text{ReLU}(-x)})) \\ & = (1 - y)x + \text{ReLU}(-x) \\ & \quad + \log(e^{-\text{ReLU}(-x)} + e^{-x - \text{ReLU}(-x)}) \end{aligned}$$

- In (12), when $x \ll 0$, e^{-x} can be large and goes to inf .

Loss function V

- In (13), we have

$$\xi(x, y) = \begin{cases} (1 - y)x + \log(1 + e^{-x}), & x \geq 0 \\ -yx + \log(1 + e^x), & x < 0 \end{cases}$$

so that the exponential term never overflows.

- The same technique is also used in LIBLINEAR. See line 290-293 of [linear.cpp](#).
- However, they use (13) instead of an if statement for the sake of automatic differentiation.

Loss function VI

- For the issue that $x \approx 0$, all we need is the $\log 1p$ function, which gives more accurate result. See the footnote on page 5 of Lin et al. (2007).

Summary I

- We have prepared [simpleNN-text](#) for profiling.
- Here are major parts we modified from simpleNN:
 - the shape of $\phi(Z^{in,i})$ and the “sub-image” size from $h \times h$ to h .
 - modifying the loss to be BCEWithLogitsLoss, the default in LibMultiLabel.

Outline

- 1 Introduction
 - Background
 - From simpleNN to simpleNN-text

- 2 Project Contents

Project Description I

- To ensure that you have correct settings, please first compare simpleNN-text with LibMultiLabel by using the toy example.

| data/repo | simpleNN-text | LibMultiLabel |
|-----------------|--|--|
| config | config/ledgar_toy.config | example_config/LEDGAR/kim_cnn_simplenn_toy.yml |
| train | data/ledgar_toy.mat size(Z) = (100, 10,000) | data/LEDGAR/train_toy.txt |
| test | data/ledgar_toy.t.mat size(Z) = (100, 10,000) | data/LEDGAR/test_toy.txt |
| initial weights | data/ledgar_init_toy.mat | N/A |

Project Description II

- Here the size of input Z is $(100 \ 10000)$, where 100 is $\#$ instances and 10,000 is embedding dimension (100) times max_length (100).
- Then in simpleNN-text we transform it to the form same as (6), with size

(embedding dimension, max_length \times $\#$ instances).

Project Description III

- The following commands should have the same loss in each epoch.
 - In simpleNN-text, please run the following command on Matlab:
`example("-bsize 100 -s 2 -lr 0.1 -C inf -epoch_max 10", 0, "config/ledgerar_toy.config"),`
where `-bsize` specifies the batch size,
`-s` specifies the optimization method,
`-lr` specifies the learning rate,

Project Description IV

-C specifies the C in the objective function

$$f(\theta) = \frac{\theta^T \theta}{2C} + \frac{1}{N} \sum_{i=1}^N \xi(\theta, x_i, y_i),$$

(Here we put $C = \text{inf}$ to turn off the regularization.)

-epoch_max specifies the # epochs to run.

- In LibMultiLabel, please run
`python3 main.py --config
your-path-to-kim_cnn_simplenn_toy.yml
--cpu`

Project Description V

- After that, you can do profiling on the complete LEDGAR dataset.

| | | |
|-----------|---|--|
| data/repo | simpleNN-text | LibMultiLabel |
| config | config/ledgar.config | example_config/LEDGAR/kim_cnn_simplenn.yml |
| train | data/ledgar.mat size(Z) = (60,000, 10,000) | data/LEDGAR/train.txt |
| test | data/ledgar.t.mat size(Z) = (10,000, 10,000) | data/LEDGAR/test.txt |

- Since the full mat file is large, we put it on
linux4: /tmp2/d11922012/optd12023/data.
- Here we may not use the same initial weights since it doesn't affect the profiling results.
- However, you have to ensure that both use the same parameters (e.g. batch size and learning rate).

Project Description VI

- We're interested in checking which implementation is more efficient.
- Let's run SGD without momentum for 10 epochs on both LibMultiLabel and simpleNN-text.
- Please check and analyze
 - the whole running time for forward and backward passes
 - the percentage of each main operation of our simpleNN-text implementation (mentioned in previous slides).

Submission I

- Write a 2-page report on the profiling results, including
 - For forward and backward pass, which leads better performance, simpleNN-text or LibMultiLabel? Why? You can do some experiments to support your idea.
 - Which is the bottleneck operation in simpleNN-text? Can we further optimize it?
- Upload your report in PDF format to NTU Cool before **2023/05/30 23:59**.

References I

H.-T. Lin, C.-J. Lin, and R. C. Weng. A note on Platt's probabilistic outputs for support vector machines. *Machine Learning*, 68:267–276, 2007. URL <http://www.csie.ntu.edu.tw/~cjlin/papers/plattprob.pdf>.