Lab 2 - 定制一个新的张量运算

实验目的

1. 理解DNN框架中的张量算子的原理
2. 基于不同方法实现新的张量运算，并比较性能差异

实验环境

- PyTorch==1.5.0

实验原理

1. 深度神经网络中的张量运算原理
2. PyTorch中基于Function和Module构造张量的方法
3. 通过C++扩展编写Python函数模块

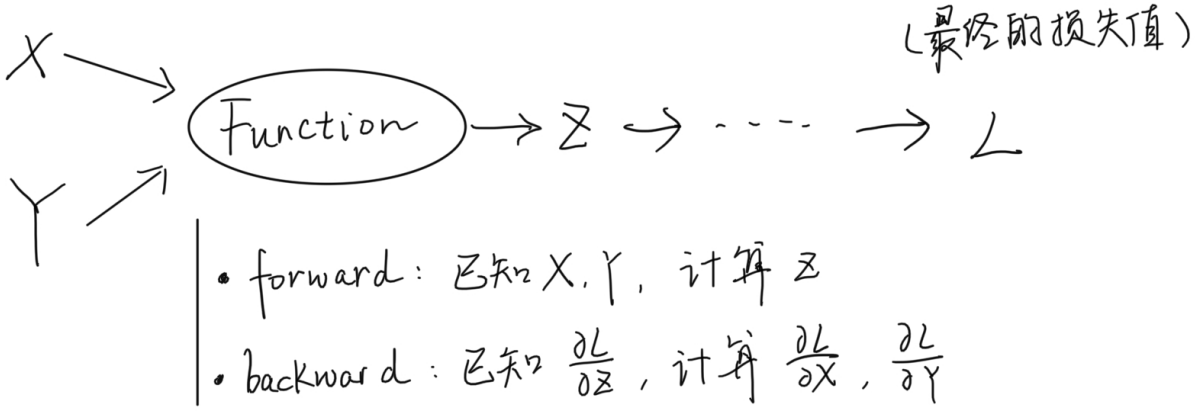
实验内容

具体步骤

1. 在MNIST的模型样例中，选择线性层（Linear）张量运算进行定制化实现
2. 理解PyTorch构造张量运算的基本单位：Function和Module

   Pytorch的核心思想是计算流图，而一个 Function 就表示计算流图中的一个计算节点类：给定输入计算输出，即对于每个 Function 子类都要实现一个 forward() 方法

   而与一般的“函数”不同的是，由于训练时要频繁的通过反向传播计算梯度，Pytorch要求每个计算节点通过 backward() 给出，在给定对输出的梯度的情况下，如何计算对输入的梯度，即

   这也是为什么 backward() 的输入数目等于 forward() 的输出数目，backward() 的输出数目等于 forward() 的输入数目

   Function 只关注如何进行“计算”（包括正向和反向），而 Module 主要用于实例化 Function，指定 Function 的输入，可以理解为计算流图中某个位置的节点
3. 基于Function和Module的Python API重新实现Linear张量运算
   1. 修改MNIST样例代码
   2. 基于PyTorch Module编写自定义的Linear 类模块

3. 基于PyTorch Function实现前向计算和反向传播函数

```
class myLinearFunction(torch.autograd.Function):
    @staticmethod
    def forward(ctx, input, weight):
        ctx.save_for_backward(input, weight)
        output = input.mm(weight.t())
        return output

    @staticmethod
    def backward(ctx, grad_output):
        input, weight = ctx.saved_tensors
        grad_input = grad_weight = None
        grad_input = grad_output.mm(weight)
        grad_weight = grad_output.t().mm(input)
        return grad_input, grad_weight
```

这里没有考虑偏移，对于 forward() 只需将输入张量与参数相乘，同时根据矩阵求导的方法可知 backward()，即

$$Y = AX => \frac{\partial L}{\partial X} = A^T * \frac{\partial L}{\partial Y}$$

4. 使用自定义Linear替换网络中nn.Linear() 类
5. 运行程序，验证网络正确性

```
(base) E:\USTC\2021Spring\ai-system\AI-System\Labs\BasicLabs\Lab2>python mnist_custom_linear.py --no-cuda
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.325924
Train Epoch: 1 [640/60000 (1%)] Loss: 1.318988
Train Epoch: 1 [1280/60000 (2%)]        Loss: 1.056302
Train Epoch: 1 [1920/60000 (3%)]        Loss: 0.633771
Train Epoch: 1 [2560/60000 (4%)]        Loss: 0.246230
Train Epoch: 1 [3200/60000 (5%)]        Loss: 0.463076
Train Epoch: 1 [3840/60000 (6%)]        Loss: 0.459432
Train Epoch: 1 [4480/60000 (7%)]        Loss: 0.304726
Train Epoch: 1 [5120/60000 (9%)]        Loss: 0.299073
Train Epoch: 1 [5760/60000 (10%)]       Loss: 0.305256
```

4. 理解PyTorch张量运算在后端执行原理

5. 实现C++版本的定制化张量运算

1. 基于C++，实现自定义Linear层前向计算和反向传播函数，并绑定为Python模型

   这里采用事先编译的C++拓展方式，在子目录中添加`setup.py`，声明文件名和模块名

```
from setuptools import setup, Extension
from torch.utils import cpp_extension

setup(name='mylinear_cpp',
      ext_modules=[cpp_extension.CppExtension('mylinear_cpp', ['mylinear.cpp'])],
      cmdclass={'build_ext': cpp_extension.BuildExtension})
```

   添加`mylinear.cpp`，编写要用到的函数（这里是 forward 和 backward 函数），并使用pybind11将C++中的函数与Pytorch中要调用的 forward() 和 backward() 方法绑定

```
#include <torch/extension.h>

#include <iostream>
#include <vector>
```

```
std::vector<torch::Tensor> mylinear_forward(
    torch::Tensor input,
    torch::Tensor weights)
{
    auto output = torch::mm(input, weights.transpose(0, 1));

    return {output};
}

std::vector<torch::Tensor> mylinear_backward(
    torch::Tensor grad_output,
    torch::Tensor input,
    torch::Tensor weights
    )
{
    auto grad_input = torch::mm(grad_output, weights);
    auto grad_weights = torch::mm(grad_output.transpose(0, 1), input);

    return {grad_input, grad_weights};
}

PYBIND11_MODULE(TORCH_EXTENSION_NAME, m) {
  m.def("forward", &mylinear_forward, "myLinear forward");
  m.def("backward", &mylinear_backward, "myLinear backward");
}
```

2. 将代码生成python的C++扩展

   使用 `python setup.py install` 生成相应模块

```
(base) E:\USTC\2021Spring\ai-system\AI-System\Labs\BasicLabs\Lab2\mylinear_cpp_extension>python setup.py install
running install
running bdist_egg
running egg_info
writing mylinear_cpp.egg-info\PKG-INFO
writing dependency_links to mylinear_cpp.egg-info\dependency_links.txt
writing top-level names to mylinear_cpp.egg-info\top_level.txt
reading manifest file 'mylinear_cpp.egg-info\SOURCES.txt'
writing manifest file 'mylinear_cpp.egg-info\SOURCES.txt'
installing library code to build\bdist.win-amd64\egg
running install_lib
running build_ext
E:\Anaconda\lib\site-packages\torch\utils\cpp_extension.py:237: UserWarning: Error checking compiler version for cl: 'utf-8' codec can't decode byte 0xd3 in
 position 0: invalid continuation byte
  warnings.warn('Error checking compiler version for {}: {}'.format(compiler, error))
creating build\bdist.win-amd64\egg
copying build\lib.win-amd64-3.7\mylinear_cpp.cp37-win_amd64.pyd -> build\bdist.win-amd64\egg
creating stub loader for mylinear_cpp.cp37-win_amd64.pyd
byte-compiling build\bdist.win-amd64\egg\mylinear_cpp.py to mylinear_cpp.cpython-37.pyc
creating build\bdist.win-amd64\egg\EGG-INFO
copying mylinear_cpp.egg-info\PKG-INFO -> build\bdist.win-amd64\egg\EGG-INFO
copying mylinear_cpp.egg-info\SOURCES.txt -> build\bdist.win-amd64\egg\EGG-INFO
copying mylinear_cpp.egg-info\dependency_links.txt -> build\bdist.win-amd64\egg\EGG-INFO
copying mylinear_cpp.egg-info\top_level.txt -> build\bdist.win-amd64\egg\EGG-INFO
writing build\bdist.win-amd64\egg\EGG-INFO\native_libs.txt
zip_safe flag not set; analyzing archive contents...
__pycache__.mylinear_cpp.cpython-37: module references __file__
creating 'dist\mylinear_cpp-0.0.0-py3.7-win-amd64.egg' and adding 'build\bdist.win-amd64\egg' to it
removing 'build\bdist.win-amd64\egg' (and everything under it)
Processing mylinear_cpp-0.0.0-py3.7-win-amd64.egg
creating e:\anaconda\lib\site-packages\mylinear_cpp-0.0.0-py3.7-win-amd64.egg
Extracting mylinear_cpp-0.0.0-py3.7-win-amd64.egg to e:\anaconda\lib\site-packages
Adding mylinear-cpp 0.0.0 to easy-install.pth file
```

3. 使用基于C++的函数扩展，实现自定义Linear类模块的前向计算和反向传播函数

   直接import定义的模块 `mylinear_cpp`，并且在定义Function时，使用 `mylinear_cpp` 中的方法

```
import mylinear_cpp

class myLinearFunction(torch.autograd.Function):
    # Note that both forward and backward are @staticmethods
    @staticmethod
    def forward(ctx, input, weight):
```

```
        ctx.save_for_backward(input, weight)
        output = mylinear_cpp.forward(input, weight)
        return output[0]

    @staticmethod
    def backward(ctx, grad_output):
        input, weight = ctx.saved_tensors
        grad_input, grad_weight = mylinear_cpp.backward(grad_output, input,
weight)
        return grad_input, grad_weight
```

4. 运行程序，验证网络正确性

```
(base) E:\USTC\2021Spring\ai-system\AI-System\Labs\BasicLabs\Lab2>python mnist_custom_linear_cpp.py --no-cuda
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.325924
Train Epoch: 1 [640/60000 (1%)] Loss: 1.318988
Train Epoch: 1 [1280/60000 (2%)]        Loss: 1.056302
Train Epoch: 1 [1920/60000 (3%)]        Loss: 0.633771
Train Epoch: 1 [2560/60000 (4%)]        Loss: 0.246230
Train Epoch: 1 [3200/60000 (5%)]        Loss: 0.463076
Train Epoch: 1 [3840/60000 (6%)]        Loss: 0.459432
Train Epoch: 1 [4480/60000 (7%)]        Loss: 0.304726
Train Epoch: 1 [5120/60000 (9%)]        Loss: 0.299073
Train Epoch: 1 [5760/60000 (10%)]       Loss: 0.305256
```

6. 使用profiler比较网络性能：比较原有张量运算和两种自定义张量运算的性能

   见实验结果中的图，可以发现原有张量运算使用的是`t`和`addmm`，两种自定义使用的是`myLinearFunction, t, mm`

   各种运算的性能差别不大，而且我反复测试后发现C++拓展的性能基本上不如Pytorch拓展的性能，也可见Pytorch本身的优化已经非常出色了

实验报告

实验环境

硬件环境	CPU（vCPU数目）	Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz
	GPU(型号，数目)	NVIDIA GeForce GTX 1050 Ti
软件环境	OS版本	Windows 10
	深度学习框架 python包名称及版本	Pytorch 1.5.0 with Python 3.7.1
	CUDA版本	10.1

实验结果

	（均未开启CUDA）
实现方式（Linear层为例）	性能评测
PyTorch原有张量运算	flatten 0.42% 47.900us 0.42% 47.900us 47.900us 1 [] reshape 0.31% 35.200us 0.31% 35.200us 35.200us 1 [] view 0.21% 23.600us 0.21% 23.600us 23.600us 1 [] t 0.25% 29.100us 0.25% 29.100us 29.100us 1 [] addmm 2.92% 334.500us 2.92% 334.500us 334.500us 1 [] relu 0.13% 15.000us 0.13% 15.000us 15.000us 1 [] feature_dropout 0.32% 37.000us 0.32% 37.000us 37.000us 1 [] empty 0.02% 1.900us 0.02% 1.900us 1.900us 1 [] bernoulli_ 0.08% 9.400us 0.08% 9.400us 9.400us 1 [] div_ 0.12% 13.300us 0.12% 13.300us 13.300us 1 [] mul 0.07% 8.300us 0.07% 8.300us 8.300us 1 [] t 0.06% 7.000us 0.06% 7.000us 7.000us 1 [] addmm 0.18% 20.900us 0.18% 20.900us 20.900us 1 [] log_softmax 0.47% 53.500us 0.47% 53.500us 53.500us 1 [] _log_softmax 0.39% 45.000us 0.39% 45.000us 45.000us 1 [] Self CPU time total: 11.438ms
基于Python API的定制化张量运算	flatten 0.36% 39.900us 0.36% 39.900us 39.900us 1 [] reshape 0.25% 27.400us 0.25% 27.400us 27.400us 1 [] view 0.14% 15.400us 0.14% 15.400us 15.400us 1 [] t 0.28% 31.100us 0.28% 31.100us 31.100us 1 [] addmm 2.86% 312.800us 2.86% 312.800us 312.800us 1 [] relu 0.14% 14.900us 0.14% 14.900us 14.900us 1 [] feature_dropout 0.67% 73.800us 0.67% 73.800us 73.800us 1 [] empty 0.02% 2.000us 0.02% 2.000us 2.000us 1 [] bernoulli_ 0.25% 27.100us 0.25% 27.100us 27.100us 1 [] div_ 0.18% 19.800us 0.18% 19.800us 19.800us 1 [] mul 0.17% 18.600us 0.17% 18.600us 18.600us 1 [] myLinearFunction 0.92% 100.600us 0.92% 100.600us 100.600us 1 [] t 0.05% 5.900us 0.05% 5.900us 5.900us 1 [] mm 0.16% 18.000us 0.16% 18.000us 18.000us 1 [] log_softmax 0.48% 52.400us 0.48% 52.400us 52.400us 1 [] _log_softmax 0.40% 44.100us 0.40% 44.100us 44.100us 1 [] Self CPU time total: 10.943ms
基于C++的定制化张量运算	flatten 0.40% 46.500us 0.40% 46.500us 46.500us 1 [] reshape 0.29% 33.700us 0.29% 33.700us 33.700us 1 [] view 0.18% 21.600us 0.18% 21.600us 21.600us 1 [] t 0.25% 29.300us 0.25% 29.300us 29.300us 1 [] addmm 3.26% 383.000us 3.26% 383.000us 383.000us 1 [] relu 0.13% 15.100us 0.13% 15.100us 15.100us 1 [] feature_dropout 0.46% 53.700us 0.46% 53.700us 53.700us 1 [] empty 0.02% 2.000us 0.02% 2.000us 2.000us 1 [] bernoulli_ 0.09% 10.500us 0.09% 10.500us 10.500us 1 [] div_ 0.12% 14.500us 0.12% 14.500us 14.500us 1 [] mul 0.18% 20.800us 0.18% 20.800us 20.800us 1 [] myLinearFunction 0.91% 107.500us 0.91% 107.500us 107.500us 1 [] transpose 0.08% 9.700us 0.08% 9.700us 9.700us 1 [] mm 0.15% 17.500us 0.15% 17.500us 17.500us 1 [] log_softmax 0.48% 56.000us 0.48% 56.000us 56.000us 1 [] _log_softmax 0.41% 47.600us 0.41% 47.600us 47.600us 1 [] Self CPU time total: 11.749ms

参考资料

- EXTENDING PYTORCH: https://pytorch.org/docs/master/notes/extending.html