Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»
Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа
по курсу «Параллельные и распределенные вычисления»

Группа:	М8О-114М-22
Студент:	Полей-Добронравова Амелия Вадимовна

Москва, 2022

Задание

1. Обучить моделифреймворка pytorch на параллельных вычислителях с помощью технологии mpi.
2. Переслать разные куски датасета для этих нейронных сетей для обучения.
3. Переслать данные для предсказания разным процессам с разными нейронными сетями.
4. В управляющем процессе сделать предсказание ансамблевым методом.

Решение

Решение представлено тремя файлами: mpi_pytorch.ipynb с демонстрацией решения, train.py - файл для обучения ансамбля моделей, test.py - файл для тестирования ансамбля моделей.

В **mpi_pytorch.ipynb** продемонстрирована закачка датасета с фотографиями CIFAR10, выведены несколько примеров из датасета, представлены примеры запуска обучения и тестирования ансамбля моделей. Запуск обучения и тестирования производится на 4 процессах с помощью команды mpirun.

```
!mpirun -n 4 python3 train.py
!mpirun -n 4 python3 test.py
```

Ансамбль моделей представляет из себя несколько сверточных сетей VGG16.

В **train.py** создана отдельная функция для обучения одной нейросетевой модели, остальные действия происходят в блоке main. Обучение длится всего одну эпоху, модели не успевают достаточно обучиться, но для демонстрации интеграции pytorch и mpi не требуется обучать модели на высокое качество.
В управляющем процессе с рангом 0 обучающий датасет делится на число частей, равное числу процессов - 1. Далее каждому процессу отправляется своя часть датасета с помощью команды comm.send().
В обычном процессе создается загрузчик данных для своей части датасета, полученной с помощью команды comm.recv(), производится обучение и

сохраняются веса модели в отдельный файл, содержащий в названии ранг своего процесса.

В **test.py** управляющий процесс с рангом 0 отсылает один образец из тестового набора данных всем другим процессам. Потом получает Предсказание от каждого, считает среднее арифметическое по процентам каждого класса и выдает класс с максимальным процентом вероятности.

Код

train.py

```
import torch

import torchvision

import torchvision.transforms as transforms

import torch.nn as nn

import torch.nn.functional as F

import torch.optim as optim

from torchvision.models import vgg16


from mpi4py import MPI


from tqdm import tqdm


def train(my_rank, net, criterion, optimizer, trainloader):

	running_loss = 0.0

	it = 0

	for i, data in tqdm(enumerate(trainloader, 0)):

	if it == 5:

		break

	inputs, labels = data

	optimizer.zero_grad()


	outputs = net(inputs)

	loss = criterion(outputs, labels)
```

```
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        it += 1
        print(f'model: {my_rank} \n loss: {running_loss / it}\n')
        running_loss = 0.0
        return net

if __name__ == "__main__":
        comm = MPI.COMM_WORLD
        my_rank = comm.Get_rank()
        p = comm.Get_size()

        img_size = (300, 450)

        transform = transforms.Compose(
        [transforms.Resize(img_size),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

        if my_rank == 0:
        trainset = torchvision.datasets.CIFAR10(root='./data', train = True,
download=False, transform=transform)
        #разделяем датасет на p-1 частей
        length1 = len(trainset) // (p - 1)
        l = [length1 for i in range(p - 2)]
        l.append(len(trainset) - (p - 2) * length1)
        parts = torch.utils.data.random_split(trainset, l)
        for i in range(len(parts)):
                comm.send(parts[i], dest = i + 1)
```

```
        else:
        #создаем свой экземпляр нейросети
        net = vgg16(pretrained=False)
        num_ftrs = net.classifier[6].in_features
        net.classifier[6] = nn.Linear(num_ftrs, 10)

        criterion = nn.CrossEntropyLoss()
        optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

        #получаем свой кусок датасета
        dataset_part = comm.recv(source = 0)

        #создаем загрузчик датасета
        batch_size = 4
        trainloader = torch.utils.data.DataLoader(dataset_part,
batch_size=batch_size, shuffle=True, num_workers=2)

        #обучаем свой экземпляр нейросети
        net = train(my_rank, net, criterion, optimizer, trainloader)

        #сохраняем обученную модель
        PATH = f'./net_{my_rank}.pth'
        torch.save(net.state_dict(), PATH)
        MPI.Finalize
```

test.py

```
import torch
import torchvision
import torch.nn as nn
import torchvision.transforms as transforms
from torchvision.models import vgg16
from mpi4py import MPI
```

```
classes = ('plane', 'car', 'bird', 'cat',

              'deer', 'dog', 'frog', 'horse', 'ship', 'truck')


if __name__ == "__main__":

       comm = MPI.COMM_WORLD

       my_rank = comm.Get_rank()

       p = comm.Get_size()


       img_size = (300, 450)


       transform = transforms.Compose(

       [transforms.Resize(img_size),

       transforms.ToTensor(),

       transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])


       if my_rank == 0:

       #итерация по тестовому датасету

       testset = torchvision.datasets.CIFAR10(root='./data', train=False,
download=False, transform=transform)

       batch_size = 1

       testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
shuffle=True, num_workers=2)

       it = 0

       for i, data in enumerate(testloader, 0):

              if it == 5:

              break

              inputs, labels = data

              #отправить моделям на предсказание

              for i in range(1, p):

              comm.send(inputs, i)


              preds = []

              #получить предсказания и сделать ансамбль голосов
```

```
        for procid in range(1, p):
        preds.append(comm.recv(source = procid))
        #среднее арифметическое
        buf = torch.zeros_like(preds[0])
        for p1 in preds:
        for idx in range(len(p1)):
               buf[idx] += p1[idx]

        for idx in range(len(buf)):
        buf[idx] /= len(preds)
        it += 1
        print('Predicted: ', ' '.join(f'{classes[torch.argmax(buf)]}'))
        break

else:
#загрузить свою модель
PATH = f'./net_{my_rank}.pth'
net = vgg16(pretrained=False)
num_ftrs = net.classifier[6].in_features
net.classifier[6] = nn.Linear(num_ftrs, 10)
net.load_state_dict(torch.load(PATH))

#получить предсказание по одной картинке
image = comm.recv(source = 0)
outputs = net(image)
comm.send(outputs, dest = 0)
MPI.Finalize
```

Вывод

С помощью mpi можно просто запараллелить обучение нейросетевых моделей, что особенно актуально: нейросетевые вычисления очень тяжеловесные и долгие.