Курсовая работа по дисциплине ПГПиПОД

Выполнил: студент группы М8О-114М-22 Дмитроченко Б.А.

Задание

С помощью MPI отправить на разные ядра части датасета, обучить его на каждом ядре, а затем собрать результаты предиктов на ядре 0 и провести ансамблевое голосование

Используемый датасет

Для работы использовался датасет MNIST

In [2]:

```
! pip install mpi4py
```

```
Requirement already satisfied: mpi4py in c:\users\bonjo\miniconda3\lib\site-packages (3.1.4)
```

In [2]:

```
%%writefile MNISTSet.py
import torch, torchvision
from torchvision import datasets
from torchvision.transforms import ToTensor
import numpy as np

T = torchvision.transforms.Compose([
        torchvision.transforms.ToTensor()
        ])


train_data = torchvision.datasets.MNIST('mnist_data', train=True, download=True, transform=T)

test_data = torchvision.datasets.MNIST('mnist_data', train=False, download=True, transform=T)
```

```
Writing MNISTSet.py
```

In [3]:

```
%%writefile DataLoader.py
from torch.utils.data import DataLoader
import torch, torchvision
from MNISTSet import train_data,test_data
def DataLoader(batchSize, numWorkers,shuffle = False,):
  loaders = {
  'train_dl' : torch.utils.data.DataLoader(train_data, batch_size=batchSize, shuffle=shuffle, num_workers=numWo

  'test_dl'  : torch.utils.data.DataLoader(test_data, batch_size=batchSize, shuffle=True, num_workers=numWorkers
  }
  return loaders
```

```
Writing DataLoader.py
```

In [11]:

```
%%writefile Model.py
import torch.nn as nn

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 6, 5, padding=2),
            nn.ReLU(),
            nn.AvgPool2d(2, stride=2),
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(6, 16, 5, padding=0),
            nn.ReLU(),
            nn.AvgPool2d(2, stride=2),
        )
        self.out = nn.Linear(400, 120, 84, 10)
    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)
        output = self.out(x)
        return output
```

```
Overwriting Model.py
```

In [12]:

```
%%writefile Main.py
from mpi4py import MPI
import torch.optim as optim
import torch
```

```python
from DataLoader import DataLoader
from torch.autograd import Variable
from Model import CNN
import torch.nn as nn
from tqdm import tqdm
def train_model(num_epochs, criterion, test_dataloader, rank, batch_size, optimizer, model, train_dataloader, va
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    model.to(device)
    model_best = 0
    for epoch in range(num_epochs):
        running_loss = 0
        accuracy = 0
        dataset_sizes_train = len(train_dataloader)
        model.train()
        if rank == 0:
            i = 0
            for images, labels in tqdm(train_dataloader):
                comm.send((images, labels),dest = i%(p-1)+1)
                i+=1
                if rank != 0:
                    for i in range(len(train_dataloader)):
                        if i % (p - 1) + 1 == rank:
                            (images,lables) = comm.recv(source=0)
                            images = images.to(device)
                            lables = lables.to(device)

            output = model(images)
            loss = criterion(output,lables)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * images.size(0)
        running_loss = running_loss / dataset_sizes_train
        print("Epoch of train:", epoch + 1,"Loss: [", running_loss, "]", "rank: ", rank)

    MPI.Comm.Barrier(MPI.COMM_WORLD)

    accuracy = 0
    validate_loss = 0.0
    dataset_sizes_val = len(val_dataloader)
    if rank != 0:
        model.eval()
    if rank == 0:
        i = 0
        for images, labels in tqdm(val_dataloader):
            comm.send((images,labels), dest=i % (p - 1) + 1)
            i+=1
    if rank != 0:
        for i in range(len(val_dataloader)):
            if i % (p - 1) + 1 == rank:
                (images,lables) = comm.recv(source=0)
                images = images.to(device)
                lables = lables.to(device)
            with torch.no_grad():
                output = model(images)
                loss = criterion(output,lables)
                validate_loss += loss.item() * images.size(0)
                pred_y = torch.max(output, 1)[1].data.squeeze()
        validate_loss = validate_loss / dataset_sizes_val
        print("Epoch of validation:", epoch + 1,"Loss: ",validate_loss, rank)
        MPI.Comm.Barrier(MPI.COMM_WORLD)
        if rank != 0:
            if epoch == 0:
                model_best = validate_loss
            if validate_loss <= model_best:
                model_best = validate_loss
                torch.save(model.state_dict(), f"./weights/model_{rank}.pth")
    return model

if __name__ == "__main__":
    comm = MPI.COMM_WORLD
    my_rank = comm.Get_rank()
    p = comm.Get_size()
    num_epochs = 3
    batch_size = 30
    num_workers = 4
    train_dataloader = DataLoader(batch_size,num_workers)['train_dl']
    validate_dataloader = DataLoader(batch_size,num_workers)['test_dl']
    model = CNN()
    optimizer = optim.Adam(model.parameters(), lr=1e-5)
    criterion = nn.CrossEntropyLoss()
    model = train_model(num_epochs,criterion,validate_dataloader,my_rank,
                           batch_size,optimizer,model,train_dataloader,validate_dataloader)
    MPI.Finalize
```

```
Overwriting Main.py
```

In [ ]:
```
! mpiexec -np 6 python Main.py
```

In [ ]:
```
%%writefile Test.py
from mpi4py import MPI
import torch.optim as optim
import torch
from DataLoader import DataLoader
from torch.autograd import Variable
from Model import CNN
import torch.nn as nn
from tqdm import tqdm
def test(model, criterion, dataloader_test, dataset_sizes_test):
    score = 0
    runing_loss = 0.0
    model.eval()


    with torch.no_grad():
        if rank != 0:
            print("Start proccess number ",rank)
            for image, label in tqdm(dataloader_test):
                output = model(image)
                comm.send(output, dest=0, tag=0)
                if rank == 1:
                    comm.send(label, dest=0, tag=1)
                _, preds = torch.max(output, 1)
                loss = criterion(output, label)
                runing_loss += loss.item() * image.size(0)
                score += torch.sum(preds == label.data)
            epoch_acc = score.double() / dataset_sizes_test
            runing_loss = runing_loss / dataset_sizes_test
            print("Test process ", rank, ": score: [", epoch_acc.item(), "], loss: [", runing_loss, "]")

        MPI.Comm.Barrier(MPI.COMM_WORLD)
        result = 0
        if rank == 0:
            print("Start proccess number ",rank)
            for i in tqdm(range(len(dataloader_test))):
                label = comm.recv(source=1, tag=1)
                for procid in range(1, p):
                    output = comm.recv(source=procid, tag=0)
                    if procid == 1:
                        result_all_models = output
                    else:
                        result_all_models += output
                _, preds = torch.max(result_all_models, 1)
                result += torch.sum(preds == label.data)
            result = result.double() / dataset_sizes_test
            print("Test process result", rank, result.item())

if __name__ == "__main__":
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()
    p = comm.Get_size()
    num_epochs = 3
    batch_size = 30
    num_workers = 4
    train_dataloader = DataLoader(batch_size,num_workers)['train_dl']
    validate_dataloader = DataLoader(batch_size,num_workers)['test_dl']
    model = CNN()
    if(rank != 0):
        model.load_state_dict(torch.load(f'/content/weights/model_{rank}.pth'))
        optimizer = optim.Adam(model.parameters(), lr=0.001)
        criterion = nn.CrossEntropyLoss()
        model = test(model,criterion,validate_dataloader,len(validate_dataloader) * batch_size)
    MPI.Finalize
```

In [ ]:
```
! mpirun -np 6 python Test.py
```