Lab2 HPML

Yonathan Daniel

February 2024

C1

In this section I dropped a section of my lab2.py file from the homework

```
def Main():
    '''
    Random cropping with size 32x32 and padding 4
    Random horizontal flipping with prob 0.5
    Normalize each image's RGB with mean (0.4914,0.4822,0.4465)
    '''
    ### might be able to use reference code
    print("welcome to the main function")
    parser = argparse.ArgumentParser(description='PyTorch CIFAR10 Training')
    parser.add_argument('--lr', default=0.1, type=float, help='learning rate')
    parser.add_argument('--device', default='cpu',type = str, help =  "device")
    parser.add_argument('--num_workers',default= 2, type= int, help = "dataloader workers")
    parser.add_argument('--data_path',default="./data", type= str, help = "data path")
    parser.add_argument('--opt', default ='sgd',type = str ,help = "optimzer")
    parser.add_argument('--c7', default=False,type= bool,help ="Question c7")
    args = parser.parse_args()
    device = args.device

    #resnet.to(device)
    if args.c7:
        print("question c7")
        model = ResNet(c7= args.c7)
    else:
        model = ResNet()
    model.to(device)
    #print(f'device:{device} from main ')
    best_acc = 0  # best test accuracy
    start_epoch = 0  # start from epoch 0 or last checkpoint epoch
    ##################################
    print('==> Preparing data..')
    transform_train = transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
    ])
    ##################################
    trainset = torchvision.datasets.CIFAR10(
    root=args.data_path, train=True, download=True, transform=transform_train)
    trainloader = torch.utils.data.DataLoader(
    trainset, batch_size=128, shuffle=True, num_workers=args.num_workers)
    classes = ('plane', 'car', 'bird', 'cat', 'deer',
            'dog', 'frog', 'horse', 'ship', 'truck')
    cross_entropy = nn.CrossEntropyLoss()
    optimizer = optimizer_selection(model= model, opt = args.opt, lr = args.lr)
    ### loss same regardless
    global epoch_time
    epoch_time= 0
    global mini_batch_time
    mini_batch_time = 0
    global io_time
    io_time = 0
    ###########################################
    for epoch in range(start_epoch, start_epoch+6):

        train(model,epoch,cross_entropy,optimizer,device,trainloader)
        if epoch == 0:
            print("Warm-up epoch.....")
            epoch_time= 0
            mini_batch_time = 0
            io_time = 0
            ## ignore epoch 0
            #epoch_time+= dummy1
            #mini_batch_time+= dummy2
            #io_time+= dummy3
    print(f"Total times for epoch: {epoch_time} sec, mini batch computations: {mini_batch_time} sec, IO: {io_time} sec")
    print(f"Average Epoch time:{epoch_time/5}")
    print(f"Number of workers: {args.num_workers} sec")
    parameters_vs_gradients(model)
def train(model,epoch,criterion,optimizer,device,dataloader):
    print('\nEpoch: %d' % epoch)
    model.train()#resnet.train()
    train_loss = 0
    correct = 0
    total = 0
    progress_bar = tqdm(dataloader, desc=f'Epoch {epoch}', leave=False)
    mini_batch_times = []
    io_times = []
    if device == 'cpu':
        epoch_start = time.perf_counter()
        for batch_idx, (inputs, targets) in (enumerate(progress_bar)):#enumerate(trainloader):

            io_start = time.perf_counter()
            inputs, targets = inputs.to(device), targets.to(device)
            io_end = time.perf_counter()

            optimizer.zero_grad()
            outputs = model(inputs)#resnet(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()

            minibatch_end = time.perf_counter()

            train_loss += loss.item()

            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()
            progress_bar.set_postfix(loss=train_loss / (batch_idx + 1), accuracy=100. * correct / total)
            #print(f"\n minibatch :{minibatch_end-io_end}, io: {io_end-io_start}")
            mini_batch_times.append(minibatch_end-io_end)
            io_times.append(io_end-io_start)
        epoch_end = time.perf_counter()
        total_epoch = epoch_end-epoch_start
        print(f"epoch: {epoch} time:{total_epoch} sec")
        avg_mini_batch_time = torch.tensor(mini_batch_times).mean().item()
        avg_io_time = torch.tensor(io_times).mean().item()
        total_io = torch.tensor(io_times).sum().item()
        total_mini_batch = torch.tensor(mini_batch_times).sum().item()
    elif device == 'cuda':
        torch.cuda.synchronize()## wait for kernels to finish....
        epoch_start = time.perf_counter()
        for batch_idx, (inputs, targets) in (enumerate(progress_bar)):#enumerate(trainloader):

            torch.cuda.synchronize()## wait for kernels to finish....
            io_start = time.perf_counter()
            inputs, targets = inputs.to(device), targets.to(device)
            torch.cuda.synchronize()## wait for kernels to finish....
            io_end = time.perf_counter()

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()
            torch.cuda.synchronize()## wait for kernels to finish....torch.cuda.synchronize()## wait for kernels to finish....
            minibatch_end = time.perf_counter()

            train_loss += loss.item()

            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()
            progress_bar.set_postfix(loss=train_loss / (batch_idx + 1), accuracy=100. * correct / total)
            mini_batch_times.append(minibatch_end-io_end)
            io_times.append(io_end-io_start)
            #print(f"\n minibatch :{minibatch_end-io_end}, io: {io_end-io_start}")
        torch.cuda.synchronize()## wait for kernels to finish....
        epoch_end = time.perf_counter()
        total_epoch = epoch_end-epoch_start
        print(f"epoch: {epoch} time:{total_epoch} sec")
        avg_mini_batch_time = torch.tensor(mini_batch_times).mean().item()
        avg_io_time = torch.tensor(io_times).mean().item()
        total_io = torch.tensor(io_times).sum().item()
        total_mini_batch = torch.tensor(mini_batch_times).sum().item()
    else:
        print("Probably entered an invalid device i.e. not (cuda/cpu)")
        train_loss = 0
        correct = 0
        total = 1
        avg_mini_batch_time = 0
        avg_io_time = 0
        total_epoch,total_mini_batch,total_io  =0,0,0
    ######################################################
    average_loss = train_loss / len(dataloader)
    accuracy = correct / total
    print(f'Training Loss: {average_loss:.4f}, Accuracy: {100 * accuracy:.2f}%')
    print(f"average mini batch time:{avg_mini_batch_time} sec, average I/O time: {avg_io_time} sec")
    print(f"mini batch time:{total_mini_batch} sec, I/O time: {total_io} sec\n")
    global epoch_time
    epoch_time+= total_epoch
    global mini_batch_time
    mini_batch_time +=total_mini_batch
    global io_time
    io_time += total_io
    #return total_epoch,total_mini_batch,total_io

def optimizer_selection(model, opt,lr ):
    opt = opt.lower()
    print(f"opt: {opt} in the selection function")
    if opt == "sgd":
        ret = optim.SGD(model.parameters(), lr=lr,
                  momentum=0.9, weight_decay=5e-4, nesterov=False)
    elif opt == "nesterov":
        ret = optim.SGD(model.parameters(), lr=lr,
                  momentum=0.9, weight_decay=5e-4,nesterov=True)
    elif opt == "adadelta":
        ret = optim.Adadelta(model.parameters(), lr=lr,
                  weight_decay=5e-4)
    elif opt == 'adagrad':
        ret = optim.Adagrad(model.parameters(), lr=lr,
                 weight_decay=5e-4)
    elif opt == 'adam':
        ret = optim.Adam(model.parameters(), lr=lr,
                  weight_decay=5e-4)
    else:
        ### default sgd case:
        ret = optim.SGD(model.parameters(), lr=lr,
                  momentum=0.9, weight_decay=5e-4)
    return ret


def parameters_vs_gradients(model):
    print("Finding Gradients vs parameters")
    param_count =[p for p in model.parameters()] #len(resnet.parameters())
    grad_count = [p for p in model.parameters() if p.requires_grad]
    print(f"params: {len(param_count)}, grads: {len(grad_count)}")

epoch_time = 0
mini_batch_time = 0
io_time = 0
if __name__ == "__main__":
    Main()
```

Default settings	**Per batch loss**	**Accuracy**
epoch 1	**1.3750**	**49.33%**
epoch 2	**1.0976**	**60.71%**
epoch 3	**0.9215**	**67.50%**
epoch 4	**0.8017**	**72.24%**
epoch 5	**0.7153**	**75.36%**

Results for C1 in table form for easiy analysis

C2

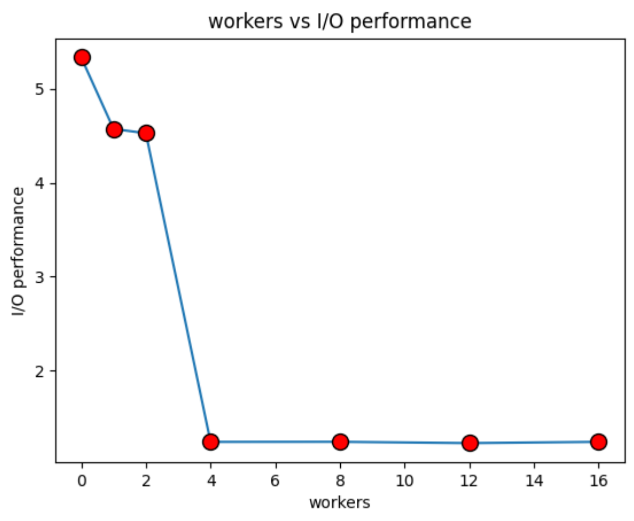
N/A	C2.1	C2.2	C2.3
Epoch	**I/O performance**	**Training**	**Total time**
1	**0.25000402331352234 sec**	**62.26639175415029 sec**	**63.545130733000406 sec**
2	**0.2508372366428375 sec**	**62.251243591308594 sec**	**63.57696755699726 sec**
3	**0.24950550496578217 sec**	**62.26773452758789 sec**	**63.563000470996485 sec**
4	**0.25290557742118835**	**62.23643493652344 sec**	**63.605433936005284 sec**
5	**0.2585148215293884 sec**	**62.246551513671875 sec**	**63.620164542000566 sec**

Above are the results from C2. They are computed in alongside the results from c1. We can see especially in the CUDA if statement that I had to set it up to wait for threads to synchronize.

C3

Below I have a graph demonstrating number of I/O performance with respect to number of workers in the Resnet-18 implementation. After four workers results hardly change, but I decided that 12 was optimal as it has the best performance. In all honesty it may not have been the best decision in hindsight as at the time I was not thinking of the potential overhead of having 8 additional processes compared to 4 especially with the marginal difference in performance. I provided the table in 3.2 as it has the exact values, and gives the reasoning as to why I choose 12 workers.

C3.1



C3.2

number of workers	I/O performance
0	5.332523912191391 sec
1	4.571821928024292 sec
2	4.525736033916473 sec
4	1.242966279387474 sec
8	1.243903398513794 sec
12	1.229981154203415 sec
16	1.2434450834989548 sec

C4

The table from table 3.2 will come in handy as it provides a clear distinction in the overall dataloading time across all the epochs. I choose 12 workers as optimal based off the pure I/O performance. Hear having 1 worker means there is one main process alongside 1 worker process, and it is a bit slower across the 5 epochs. Over a lot of epochs this could become worse, and this was done on the GPU, so for a local machine there is even more potential for bottlenecks.

C5

Results base off the the choose optimal set of workers for runtime performance on GPU vs CPU. All my executions were on the GCP, and this made it clear how valuable a GPU is as he pure CPU executions were very painful to sit through.

number of workers	average run time performance	CPU/GPU
12	**473.673188990123778 sec**	**CPU**
12	**63.880899448800484 sec**	**GPU**

C6

For varying optimizers I executed that training loop across 5 epochs and averaged the results like so below with 12 workers once again.

Optimizer	average training time	Average loss	Top 1 accuracy
SGD	**63.5821394478 sec**	**0.98222**	**75.36%**
SGD with Nesterov	**63.84255967919889 sec**	**0.87074**	**77.28%**
Adagrad	**62.42804886139929 sec**	**1.53492**	**57.50%**
Adam	**61.36156230460183 sec**	**1.9984**	**19.18%**
Adadelta	**65.1252761199067 sec**	**0.77716**	**79.68%**

C7

This section provided an interesting way to see how important batch normalization is for improving neural network's accuracy.

Optimizer	average training time	Average loss	Top 1 accuracy
SGD	**60.23021916759899**	**1.55002**	**52.64%**

Q1

First input layer is a convolution, every block has at least two convolution layers, this is because based off the paper to allow the input from the from the input of the block to match the shape of the output of the second convolution when there is stride not equal to one a convolution with a 1x1 filter was specified.

Given that we have 6 blocks with stride 2 and 2 blocks with stride 1 we can find the the total convolutions as like so.

$$2 * (2) + 6 * (3) + 1 = 4 + 18 + 1 = 23$$

Q2

In the final layer of the neural network the last layer has an input shape of [128,512], from torch.size(). I had to reshape the tensor before sending it the output layer by flattening it. I think the shape being 128,512 occurs from the stride being set to 2 since form my understanding it halves the dimensions of out of a layer. Given the residual block prior to the output layer has an output channel of 256, given the stride was 2 the actual shape would be halved.

Q3

```
def parameters_vs_gradients(model):
    print("Finding Gradients vs parameters")
    param_count =[p for p in model.parameters()] #len(resnet.parameters())
    grad_count = [p for p in model.parameters() if p.requires_grad]
    print(f"params: {param_count}, grads: {grad_count}")
```

Above is the function used to process the number of parameters and gradients. I call it after the training loop is done each time, and below is the results from the terminal.

```
Finding Gradients vs parameters
#params: 42, grads: 42
```

We can see here that the learnable parameters and the number of gradients are the same in this instance with both being 42.

Q4

The result from the function I mentioned prior was exactly the same.

```
Finding Gradients vs parameters
#params: 42, grads: 42
```