[W2S2_example2][HW3_prob1]_CNN_Training_for_CIFAR10

October 26, 2023

[1]:

```
# Imports and Functions
import argparse
import os
import time
import shutil

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torch.backends.cudnn as cudnn



import torchvision
import torchvision.transforms as transforms

from models import *    # bring everything in the folder models



def train(trainloader, model, criterion, optimizer, epoch):
    batch_time = AverageMeter()    ## at the begining of each epoch, this should␣
 ↪be reset
    data_time = AverageMeter()
    losses = AverageMeter()
    top1 = AverageMeter()

    model.train()

    end = time.time()  # measure current time

    for i, (input, target) in enumerate(trainloader):
        # measure data loading time
        data_time.update(time.time() - end)  # data loading time

        input, target = input.cuda(), target.cuda()
        # compute output
        output = model(input)
        loss = criterion(output, target)

        # measure accuracy and record loss
        prec = accuracy(output, target)[0]
        losses.update(loss.item(), input.size(0))
        top1.update(prec.item(), input.size(0))

        # compute gradient and do SGD step
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # measure elapsed time
        batch_time.update(time.time() - end) # time spent to process one batch
        end = time.time()


        if i % print_freq == 0:
            print('Epoch: [{0}][{1}/{2}]\t'
                  'Time {batch_time.val:.3f} ({batch_time.avg:.3f})\t'
                  'Data {data_time.val:.3f} ({data_time.avg:.3f})\t'
                  'Loss {loss.val:.4f} ({loss.avg:.4f})\t'
                  'Prec {top1.val:.3f}% ({top1.avg:.3f}%)'.format(
                   epoch, i, len(trainloader), batch_time=batch_time,
                   data_time=data_time, loss=losses, top1=top1))



def validate(val_loader, model, criterion ):
    batch_time = AverageMeter()
    losses = AverageMeter()
    top1 = AverageMeter()

    # switch to evaluate mode
    model.eval()

    end = time.time()
    with torch.no_grad():
        for i, (input, target) in enumerate(val_loader):

            input, target = input.cuda(), target.cuda()

            # compute output
            output = model(input)
            loss = criterion(output, target)

            # measure accuracy and record loss
            prec = accuracy(output, target)[0]
            losses.update(loss.item(), input.size(0))
            top1.update(prec.item(), input.size(0))

            # measure elapsed time
            batch_time.update(time.time() - end)
            end = time.time()

            if i % print_freq == 0:  # This line shows how frequently print out␣
 ↪the status. e.g., i%5 => every 5 batch, prints out
                print('Test: [{0}/{1}]\t'
                  'Time {batch_time.val:.3f} ({batch_time.avg:.3f})\t'
                  'Loss {loss.val:.4f} ({loss.avg:.4f})\t'
                  'Prec {top1.val:.3f}% ({top1.avg:.3f}%)'.format(
                   i, len(val_loader), batch_time=batch_time, loss=losses,
                   top1=top1))

    print(' * Prec {top1.avg:.3f}% '.format(top1=top1))
    return top1.avg


def accuracy(output, target, topk=(1,)):
    """Computes the precision@k for the specified values of k"""
    maxk = max(topk)
    batch_size = target.size(0)

    _, pred = output.topk(maxk, 1, True, True)
    pred = pred.t()
    correct = pred.eq(target.view(1, -1).expand_as(pred))

    res = []
    for k in topk:
        correct_k = correct[:k].view(-1).float().sum(0)
        res.append(correct_k.mul_(100.0 / batch_size))
    return res


class AverageMeter(object):
    """Computes and stores the average and current value"""
    def __init__(self):
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n     ## n is impact factor
        self.count += n
        self.avg = self.sum / self.count


def save_checkpoint(state, is_best, fdir):
    filepath = os.path.join(fdir, 'checkpoint.pth')
    torch.save(state, filepath)
    if is_best:
        shutil.copyfile(filepath, os.path.join(fdir, 'model_best.pth.tar'))


def adjust_learning_rate(optimizer, epoch):
    """For resnet, the lr starts from 0.1, and is divided by 10 at 80 and 120␣
  ↪epochs"""
    adjust_list = [150, 225]
    if epoch in adjust_list:
        for param_group in optimizer.param_groups:
            param_group['lr'] = param_group['lr'] * 0.1

#model = nn.DataParallel(model).cuda()
#all_params = checkpoint['state_dict']
#model.load_state_dict(all_params, strict=False)
#criterion = nn.CrossEntropyLoss().cuda()
#validate(testloader, model, criterion)
```

[ ]:

```
# HW

#  1. train resnet20 and vgg16 to achieve >90% accuracy
#  2. save your trained model in the result folder
#  3. Restart your jupyter notebook by "Kernel - Restart & Clear Output"
#  4. Load your saved model for vgg16 and validate to see the accuracy
#  5. such as the last part of "[W2S2_example2]_CNN_for_MNIST.ipynb", prehook␣
  ↪the input layers of all the conv layers.
#  6. from the first prehooked input, compute to get the second prehooked input.
  ↪
#  7. Compare your computed second input vs. the prehooked second input.
```

0.0.1 VGG16 Model

[2]:

```
# VGG16

global best_prec
use_gpu = torch.cuda.is_available()
print('=> Building model...')

batch_size = 168

model_name = "VGG16"
model = VGG16()

normalize = transforms.Normalize(mean=[0.491, 0.482, 0.447], std=[0.247, 0.243,␣
  ↪0.262])

train_dataset = torchvision.datasets.CIFAR10(
    root='./data',
    train=True,
    download=True,
    transform=transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        normalize,
    ]))
trainloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size,␣
  ↪shuffle=True, num_workers=2)

test_dataset = torchvision.datasets.CIFAR10(
    root='./data',
    train=False,
    download=True,
    transform=transforms.Compose([
        transforms.ToTensor(),
        normalize,
    ]))
testloader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size,␣
  ↪shuffle=False, num_workers=2)

print_freq = 100
```

```
=> Building model…
Files already downloaded and verified
Files already downloaded and verified
```

[ ]:

```
# Training loop
lr = 2e-2
weight_decay = 1e-4
epochs = 60
best_prec = 0

model = model.cuda()
criterion = nn.CrossEntropyLoss().cuda()
optimizer = torch.optim.SGD(model.parameters(), lr=lr, momentum=0.9,␣
 ↪weight_decay=weight_decay)

if not os.path.exists('result'):
    os.makedirs('result')

fdir = 'result/' + str(model_name)

if not os.path.exists(fdir):
    os.makedirs(fdir)

for epoch in range(0, epochs):
    adjust_learning_rate(optimizer, epoch)
    train(trainloader, model, criterion, optimizer, epoch)

    print("Validation starts")
    prec = validate(testloader, model, criterion)

    is_best = prec > best_prec
    best_prec = max(prec, best_prec)
    print('best acc: {:.2f}%'.format(best_prec))
    save_checkpoint({
        'epoch': epoch + 1,
        'state_dict': model.state_dict(),
        'best_prec': best_prec,
        'optimizer': optimizer.state_dict(),
    }, is_best, fdir)
```

[3]:

```
# Load the best model and evaluate
model_name = "VGG16"
model = VGG16()

fdir = 'result/' + str(model_name) + '/model_best.pth.tar'
checkpoint = torch.load(fdir)
model.load_state_dict(checkpoint['state_dict'])

criterion = nn.CrossEntropyLoss().cuda()

model.eval()
model.cuda()

prec = validate(testloader, model, criterion)
```

```
Test: [0/60]    Time 1.860 (1.860)      Loss 0.2605 (0.2605)    Prec 94.048%
(94.048%)
 * Prec 90.630%
```

[4]:

```
model_name = "VGG16"
model = VGG16().to("cuda")


class SaveOutput:
    def __init__(self):
        self.outputs = []
    def __call__(self, module, module_in):
        self.outputs.append(module_in)
    def clear(self):
        self.outputs = []

######### Save inputs from selected layer ##########
save_output = SaveOutput()

# Register hooks for the first and second convolutional layers
for layer in model.modules():
    if isinstance(layer, torch.nn.Conv2d):
        print("prehooked")
        layer.register_forward_pre_hook(save_output)
####################################################
use_gpu = torch.cuda.is_available()
print(use_gpu)
device = torch.device("cuda" if use_gpu else "cpu")


dataiter = iter(trainloader)
images, labels = next(dataiter)
images = images.to(device)
out = model(images)

conv1 = model.features[0]
batch = model.features[1]
relu = model.features[2]
manual = relu(batch(conv1(save_output.outputs[0][0])))
(manual - save_output.outputs[1][0]).sum()
```

```
prehooked
prehooked
prehooked
```

```
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
True
```

```
[4]: tensor(0., device='cuda:0', grad_fn=<SumBackward0>)
```

0.0.2 ResNet20 Model

```
[5]: # ResNet20
import argparse
import os
import time
import shutil

import torch
import torch.nn as nn
import torch.optim as optim
import torch.backends.cudnn as cudnn
import torchvision
import torchvision.transforms as transforms

from models import *  # Import the ResNet20 and VGGNet16 models

global best_prec
use_gpu = torch.cuda.is_available()
print('=> Building model...')

batch_size = 160

model_name = "RESNET"
model = resnet20_cifar()

normalize = transforms.Normalize(mean=[0.491, 0.482, 0.447], std=[0.247, 0.243,␣
  ↪0.262])

train_dataset = torchvision.datasets.CIFAR10(
    root='./data',
    train=True,
    download=True,
    transform=transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        normalize,
    ]))
trainloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=2)

test_dataset = torchvision.datasets.CIFAR10(
    root='./data',
    train=False,
    download=True,
    transform=transforms.Compose([
        transforms.ToTensor(),
        normalize,
    ]))
testloader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=False, num_workers=2)

print_freq = 100
```

```
=> Building model…
Files already downloaded and verified
Files already downloaded and verified
```

[ ]:

```
# Training loop
lr = 2.2e-2
weight_decay = 9e-5
epochs = 180
best_prec = 0

model = model.cuda()
criterion = nn.CrossEntropyLoss().cuda()
optimizer = torch.optim.SGD(model.parameters(), lr=lr, momentum=0.9, weight_decay=weight_decay)

if not os.path.exists('result'):
    os.makedirs('result')

fdir = 'result/' + str(model_name)

if not os.path.exists(fdir):
    os.makedirs(fdir)

for epoch in range(0, epochs):
    adjust_learning_rate(optimizer, epoch)
    train(trainloader, model, criterion, optimizer, epoch)

    print("Validation starts")
    prec = validate(testloader, model, criterion)

    is_best = prec > best_prec
    best_prec = max(prec, best_prec)
    print('best acc: {:.2f}%'.format(best_prec))
    save_checkpoint({
        'epoch': epoch + 1,
        'state_dict': model.state_dict(),
        'best_prec': best_prec,
        'optimizer': optimizer.state_dict(),
    }, is_best, fdir)
```

[6]:

```
# Load the best model and evaluate
model_name = "RESNET"
model = resnet20_cifar()

fdir = 'result/' + str(model_name) + '/model_best.pth.tar'
checkpoint = torch.load(fdir)
model.load_state_dict(checkpoint['state_dict'])

criterion = nn.CrossEntropyLoss().cuda()

model.eval()
model.cuda()

prec = validate(testloader, model, criterion)
```

```
Test: [0/63]    Time 0.270 (0.270)      Loss 0.3668 (0.3668)    Prec 93.750%
(93.750%)
 * Prec 90.370%
```

[7]:

```
model_name = "RESNET"
model = resnet20_cifar().to("cuda")



class SaveOutput:
    def __init__(self):
        self.outputs = []
    def __call__(self, module, module_in):
        self.outputs.append(module_in)
    def clear(self):
        self.outputs = []

######### Save inputs from selected layer ##########
```

```
save_output = SaveOutput()

# Register hooks for the first and second convolutional layers
for layer in model.modules():
    if isinstance(layer, torch.nn.Conv2d):
        print("prehooked")
        layer.register_forward_pre_hook(save_output)
#####################################################
use_gpu = torch.cuda.is_available()
print(use_gpu)
device = torch.device("cuda" if use_gpu else "cpu")


dataiter = iter(trainloader)
images, labels = next(dataiter)
images = images.to(device)
out = model(images)


conv1 = model.layer1[0].conv1
conv2 = model.layer1[0].conv2
bn1 = model.layer1[0].bn1
relu = model.layer1[0].relu
bn2 = model.layer1[0].bn2

out_block = relu(bn2(conv2(relu(bn1(conv1(save_output.outputs[1][0]))))) +␣
 ↪save_output.outputs[1][0])
(out_block - save_output.outputs[3][0]).sum()
```

```
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
prehooked
```

```
prehooked
prehooked
prehooked
True
```

```
[7]: tensor(0., device='cuda:0', grad_fn=<SumBackward0>)
```