projectPruning

December 6, 2024

[ ]:
```
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 *


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



batch_size = 128
# model_name = "VGG16_quant4bit_UnstructuredPrune"
model_name = "VGG16_quant4bit_StructuredPrune"
model = VGG16_quant()
model.features[24] = QuantConv2d(256, 8, kernel_size=3, stride=1, padding=1,␣
 ↪bias=False)
model.features[25] = nn.BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True,␣
 ↪track_running_stats=True)
model.features[27] = QuantConv2d(8, 8, kernel_size=3, stride=1, padding=1,␣
 ↪bias=False)
model.features[28] = nn.Sequential()
model.features[30] = QuantConv2d(8, 512, kernel_size=3, stride=1, padding=1,␣
 ↪bias=False)
```

```
print(model)

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 # every 100 batches, accuracy printed. Here, each batch␣
 ↪includes "batch_size" data points
# CIFAR10 has 50,000 training data, and 10,000 validation data.

def train(trainloader, model, criterion, optimizer, epoch):
    batch_time = AverageMeter()
    data_time = AverageMeter()
    losses = AverageMeter()
    top1 = AverageMeter()

    model.train()

    end = time.time()
```

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

        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)
        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
        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 = [60, 80]
    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)
```

[ ]:
```
# This cell won't be given, but students will complete the training

lr = 4e-2
weight_decay = 1e-4
epochs = 100
best_prec = 0

#model = nn.DataParallel(model).cuda()
model.cuda()
criterion = nn.CrossEntropyLoss().cuda()
```

```
optimizer = torch.optim.SGD(model.parameters(), lr=lr, momentum=0.9,␣
 ↪weight_decay=weight_decay)
#cudnn.benchmark = True

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)

    # evaluate on test set
    print("Validation starts")
    prec = validate(testloader, model, criterion)

    # remember best precision and save checkpoint
    is_best = prec > best_prec
    best_prec = max(prec,best_prec)
    print('best acc: {:1f}'.format(best_prec))
    save_checkpoint({
        'epoch': epoch + 1,
        'state_dict': model.state_dict(),
        'best_prec': best_prec,
        'optimizer': optimizer.state_dict(),
    }, is_best, fdir)
        # Stop training if best_prec exceeds 90
    if best_prec > 90:
        print("Training stopped as accuracy surpassed 90%")
        break
```

[3]:
```
# PATH = "result/VGG16_quant4bit_UnstructuredPrune/model_best.pth.tar"
PATH = "result/VGG16_quant4bit_StructuredPrune/model_best.pth.tar"

checkpoint = torch.load(PATH)
model.load_state_dict(checkpoint['state_dict'])
device = torch.device("cuda")

model.cuda()
model.eval()

test_loss = 0
correct = 0
```

```
with torch.no_grad():
    for data, target in testloader:
        data, target = data.to(device), target.to(device) # loading to GPU
        output = model(data)
        pred = output.argmax(dim=1, keepdim=True)
        correct += pred.eq(target.view_as(pred)).sum().item()

test_loss /= len(testloader.dataset)

print('\nTest set: Accuracy: {}/{} ({:.0f}%)\n'.format(
        correct, len(testloader.dataset),
        100. * correct / len(testloader.dataset)))
```

```
Test set: Accuracy: 9054/10000 (91%)
```

[4]:
```
import torch.nn.utils.prune as prune
#unstructured

# print("\nApplying unstructured pruning...")
# for layer in model.modules():
#     if isinstance(layer, QuantConv2d):
#         prune.l1_unstructured(layer, name='weight', amount=0.5)

# structured
print("\nApplying structured pruning...")
for layer in model.modules():
    if isinstance(layer, QuantConv2d):
            prune.ln_structured(layer, name='weight', amount=0.5, n=1, dim=0)
```

```
Applying structured pruning…
```

[5]:
```
### Check sparsity ###
for layer in model.modules():
    if isinstance(layer, QuantConv2d):
            mask1 = layer.weight_mask
            sparsity_mask1 = (mask1 == 0).sum() / mask1.nelement()
            print("Sparsity level: ", sparsity_mask1)
```

```
Sparsity level:  tensor(0.5000, device='cuda:0')
Sparsity level:  tensor(0.5000, device='cuda:0')
Sparsity level:  tensor(0.5000, device='cuda:0')
Sparsity level:  tensor(0.5000, device='cuda:0')
Sparsity level:  tensor(0.5000, device='cuda:0')
Sparsity level:  tensor(0.5000, device='cuda:0')
```

```
Sparsity level:  tensor(0.5000, device='cuda:0')
Sparsity level:  tensor(0.5000, device='cuda:0')
Sparsity level:  tensor(0.5000, device='cuda:0')
Sparsity level:  tensor(0.5000, device='cuda:0')
Sparsity level:  tensor(0.5000, device='cuda:0')
Sparsity level:  tensor(0.5000, device='cuda:0')
Sparsity level:  tensor(0.5000, device='cuda:0')
```

[6]:
```python
## check accuracy after pruning
criterion = nn.CrossEntropyLoss().cuda()
acc = validate(testloader, model, criterion)
print(f"Accuracy after pruning: {acc:.2f}%")
```

```
Test: [0/79]    Time 1.401 (1.401)      Loss 5.6432 (5.6432)    Prec 10.156%
(10.156%)
 * Prec 10.000%
Accuracy after pruning: 10.00%
```

[ ]:
```python
best_prec = 0

lr = 6e-2
weight_decay = 1e-4
epochs = 50

# Modify adjust_learning_rate function
def adjust_learning_rate(optimizer, epoch):
    """For pruned models, adjust LR at epochs 20 and 35"""
    adjust_list = [20, 35]
    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()
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('UnstructuredPrune50')
fdir = 'result/'+str('StructuredPrune50')
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)
    # evaluate on test set
    print("Validation starts")
    prec = validate(testloader, model, criterion)
    # remember best precision and save checkpoint
    is_best = prec > best_prec
    best_prec = max(prec,best_prec)
    print('best acc: {:1f}'.format(best_prec))
    save_checkpoint({
        'epoch': epoch + 1,
        'state_dict': model.state_dict(),
        'best_prec': best_prec,
        'optimizer': optimizer.state_dict(),
    }, is_best, fdir)
```

[8]:
```
## check your accuracy again after finetuning
# PATH = "result/UnstructuredPrune50/model_best.pth.tar"
PATH = "result/StructuredPrune50/model_best.pth.tar"
checkpoint = torch.load(PATH)
model.load_state_dict(checkpoint['state_dict'])
device = torch.device("cuda")

model.cuda()
model.eval()

test_loss = 0
correct = 0

with torch.no_grad():
    for data, target in testloader:
        data, target = data.to(device), target.to(device) # loading to GPU
        output = model(data)
        pred = output.argmax(dim=1, keepdim=True)
        correct += pred.eq(target.view_as(pred)).sum().item()

test_loss /= len(testloader.dataset)

print('\nAccuracy after fine-tuning: {}/{} ({:.0f}%)\n'.format(
        correct, len(testloader.dataset),
        100. * correct / len(testloader.dataset)))
```

```
Accuracy after fine-tuning: 9055/10000 (91%)
```

[9]:
```
## Send an image and use prehook to grab the inputs of all the QuantConv2d␣
 ↪layers
```

```
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()
i = 0

for layer in model.modules():
    i = i+1
    if isinstance(layer, QuantConv2d):
        print(i,"-th layer prehooked")
        layer.register_forward_pre_hook(save_output)
####################################################

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

```
3 -th layer prehooked
7 -th layer prehooked
12 -th layer prehooked
16 -th layer prehooked
21 -th layer prehooked
25 -th layer prehooked
29 -th layer prehooked
34 -th layer prehooked
38 -th layer prehooked
42 -th layer prehooked
47 -th layer prehooked
51 -th layer prehooked
55 -th layer prehooked
```

[10]:
```
##### Find "weight_int" for features[3] ####
w_bit = 4
weight_q = model.features[3].weight_q
w_alpha = model.features[3].weight_quant.wgt_alpha
w_delta = w_alpha /(2**(w_bit-1)-1)

weight_int = weight_q / w_delta
```

[11]:
```
#### check your sparsity for weight_int is near 90% #####
#### Your sparsity could be >90% after quantization #####
sparsity_weight_int = (weight_int == 0).sum() / weight_int.nelement()
print("Sparsity level: ", sparsity_weight_int)
```

```
Sparsity level:  tensor(0.7183, device='cuda:0')
```

[ ]:
```

```