projectPart1VGG

December 4, 2024

[1]:
```
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"
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)
```

```
=> Building model…
VGG_quant(
  (features): Sequential(
    (0): QuantConv2d(
      3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): QuantConv2d(
```

```
      64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (7): QuantConv2d(
      64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (9): ReLU(inplace=True)
    (10): QuantConv2d(
      128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (12): ReLU(inplace=True)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (14): QuantConv2d(
      128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (16): ReLU(inplace=True)
    (17): QuantConv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (19): ReLU(inplace=True)
    (20): QuantConv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (21): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (24): QuantConv2d(
```

```
      256, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (25): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (26): ReLU(inplace=True)
    (27): QuantConv2d(
      8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (28): Sequential()
    (29): ReLU(inplace=True)
    (30): QuantConv2d(
      8, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (31): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (32): ReLU(inplace=True)
    (33): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (34): QuantConv2d(
      512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (35): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (36): ReLU(inplace=True)
    (37): QuantConv2d(
      512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (38): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (39): ReLU(inplace=True)
    (40): QuantConv2d(
      512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
      (weight_quant): weight_quantize_fn()
    )
    (41): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (42): ReLU(inplace=True)
    (43): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (44): AvgPool2d(kernel_size=1, stride=1, padding=0)
  )
  (classifier): Linear(in_features=512, out_features=10, bias=True)
)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

[ ]:
```
# 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
```

[2]:
```
PATH = "result/VGG16_quant4bit/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: 9079/10000 (91%)
```

[3]:
```
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)
####################################################
```

```
datatiter = iter(testloader)
images, labels = next(datatiter)
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
```

[4]:
```
w_bit = 4
weight_q = model.features[27].weight_q # quantized value is stored during the␣
 ↪training
w_alpha = model.features[27].weight_quant.wgt_alpha
w_delta = w_alpha/(2**(w_bit-1)-1)
weight_int = weight_q/w_delta
#print(weight_int) # you should see clean integer numbers
```

[5]:
```
x_bit = 4
x = save_output.outputs[8][0]  # input of the 8th conv layer
x_alpha  = model.features[27].act_alpha
x_delta = x_alpha/(2**x_bit-1)

act_quant_fn = act_quantization(x_bit) # define the quantization function
x_q = act_quant_fn(x, x_alpha)         # create the quantized value for x

x_int = x_q/x_delta
#print(x_int) # you should see clean integer numbers
```

[6]:
```
conv_int = torch.nn.Conv2d(in_channels = 8, out_channels=8, kernel_size = 3,␣
 ↪padding=1, bias = False)
conv_int.weight = torch.nn.parameter.Parameter(weight_int)
relu = nn.ReLU()
output_int = conv_int(x_int)
output_recovered = output_int*w_delta*x_delta
output_recovered = relu(output_recovered)
```

[7]:
```
difference = abs(save_output.outputs[9][0] - output_recovered )
print(difference.mean())  ## It should be small, e.g.,2.3 in my trainned model
```

```
tensor(3.9279e-07, device='cuda:0', grad_fn=<MeanBackward0>)
```

[ ]:
```
x_pad = torch.zeros(128, 8, 6, 6).cuda()
```

[ ]:
```
x_pad[ : ,  :, 1:5, 1:5] = x_int.cuda()
```

[ ]:
```
X = x_pad[0]
X = torch.reshape(X, (X.size(0), -1))
```

[ ]:
```
X.size()
```

[ ]:
```
tile_id = 0
nij = 200 # just a random number
#X = a_tile[tile_id,:,nij:nij+64]  # [tile_num, array row num, time_steps]

bit_precision = 4
file = open('activation.txt', 'w') #write to file
file.write('#time0row7[msb-lsb],time0row6[msb-lst],....,time0row0[msb-lst]#\n')
file.write('#time1row7[msb-lsb],time1row6[msb-lst],....,time1row0[msb-lst]#\n')
file.write('#................#\n')

for i in range(X.size(1)):  # time step
    for j in range(X.size(0)): # row #
        X_bin = '{0:04b}'.format(int(X[7-j,i].item()+0.001))
        for k in range(bit_precision):
            file.write(X_bin[k])
        #file.write(' ')  # for visibility with blank between words, you can use
    file.write('\n')
file.close() #close file
```

[ ]:
```
weight_int.size() # 8, 8 , 3, 3
W = torch.reshape(weight_int, (weight_int.size(0), weight_int.size(1), -1))
W.size() # 8, 8, 9
```

[ ]:
```
bit_precision = 4
file = open('weight.txt', 'w') #write to file
file.write('#col0row7[msb-lsb],col0row6[msb-lst],....,col0row0[msb-lst]#\n')
file.write('#col1row7[msb-lsb],col1row6[msb-lst],....,col1row0[msb-lst]#\n')
file.write('#................#\n')
for kij in range(9):
    for i in range(W.size(0)):
        for j in range(W.size(1)):
            if (W[i, 7-j, kij].item()<0):
```

```
                W_bin = '{0:04b}'.format(int(W[i,7-j,kij].
 ↪item()+2**bit_precision+0.001))
            else:
                W_bin = '{0:04b}'.format(int(W[i,7-j,kij].item()+0.001))
            for k in range(bit_precision):
                file.write(W_bin[k])
            #file.write(' ')  # for visibility with blank between words, you␣
 ↪can use
        file.write('\n')
file.close() #close file
```

[ ]:
```
p_nijg = range(X.size(1)) ## psum nij group

psum = torch.zeros(8, len(p_nijg), 9).cuda()
```

[ ]:
```
for kij in range(9):
    for nij in p_nijg:        # time domain, sequentially given input
        m = nn.Linear(8, 8, bias=False)
        m.weight = torch.nn.Parameter(W[:,:,kij])
        psum[:, nij, kij] = m(X[:,nij]).cuda()
```

[ ]:
```
bit_precision = 16
file = open('psum.txt', 'w') #write to file
file.write('#time0col7[msb-lsb],time0col6[msb-lst],....,time0col0[msb-lst]#\n')
file.write('#time1col7[msb-lsb],time1col6[msb-lst],....,time1col0[msb-lst]#\n')
file.write('#................#\n')
for kij in range(9):
    for i in range(psum.size(1)):
        for j in range(psum.size(0)):
            if (psum[7-j,i,kij].item()<0):
                P_bin = '{0:016b}'.format(int(psum[7-j,i,kij].
 ↪item()+2**bit_precision+0.001))
            else:
                P_bin = '{0:016b}'.format(int(psum[7-j,i,kij].item()+0.001))
            for k in range(bit_precision):
                file.write(P_bin[k])
            #file.write(' ')  # for visibility with blank between words, you␣
 ↪can use
        file.write('\n')
file.close()
```

[ ]:
```
address = torch.zeros(16, 9).cuda()

for o_nij in range(16):
    for kij in range(9):
        address[o_nij, kij] = int(o_nij/4)*6 + o_nij%4 + int(kij/3)*6 + kij%3
        #print(address[o_nij, kij])
```

[ ]:
```
file = open('address.txt', 'w') #write to file
file.write('#1st address#\n')
file.write('#2st address#\n')
file.write('#...............#\n')
bit_precision = 11

for i in range(address.size(0)):
    for j in range(address.size(1)):
        a_bin = '{0:011b}'.format(int(address[i, j]))
        for k in range(bit_precision):
            file.write(a_bin[k])
        file.write('\n')
file.close()
```

[ ]:
```
out = output_int[0]
out.size()
```

[ ]:
```
out = torch.reshape(out, (out.size(0), -1))
```

[ ]:
```
bit_precision = 16
file = open('output.txt', 'w') #write to file
file.write('#time0col7[msb-lsb],time0col6[msb-lst],....,time0col0[msb-lst]#\n')
file.write('#time1col7[msb-lsb],time1col6[msb-lst],....,time1col0[msb-lst]#\n')
file.write('#...............#\n')

for i in range(out.size(1)):
    for j in range(out.size(0)):
        if (out[7-j,i].item()<0):
            O_bin = '{0:016b}'.format(int(out[7-j,i].item()+2**bit_precision+0.
 ↪001))
        else:
            O_bin = '{0:016b}'.format(int(out[7-j,i].item()+0.001))
        for k in range(bit_precision):
            file.write(O_bin[k])
        #file.write(' ')  # for visibility with blank between words, you can use
    file.write('\n')
file.close()
```