VGG16_Quant_Project-MultiChannel

December 16, 2023

[ ]:
```
# # ECE 284 Project - Team RizzNet (CE) : Krish Mehta, Aryan Devrani, Anoushka Saraswat, Kumar Divij
# This file contains the VGG16 Model with 16x8 squeezed Conv layer, with 4-bit quantization aware training
# This is done to implement 2-way multichannel PEs, which would require 16 inputs instead.
# The achieved accuracy was 92%.
# The text files are written in a slightly different manner for input.txt and weights.txt,
# with one line containing weights for row15-row0
```

[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_quant_project_multichannel"
model = VGG16_quant_project_multi()
#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 = [100,200]
    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…
Files already downloaded and verified
Files already downloaded and verified
```

[ ]:
```
# # # This cell won't be given, but students will complete the training
# PATH = "result/VGG16_quant_project_multichannel/model_best.pth.tar"
# checkpoint = torch.load(PATH)
# model.load_state_dict(checkpoint['state_dict'])
# device = torch.device("cuda")

# lr = 4e-4
# weight_decay = 1e-4
# epochs = 300
```

```
# 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)
# #optimizer = torch.optim.Adam(model.parameters(), lr=lr,␣
 ↪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)
```

[2]:
```
PATH = "result/VGG16_quant_project_multichannel/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: 9200/10000 (92%)
```

[3]:
```
#Prehook and check
```

[4]:
```
## 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()

# print(model.features[27])

# model.features[27].register_forward_pre_hook(save_output);

i=0
for layer in model.modules():

    if isinstance(layer, nn.Conv2d):
        if(i==8 or i==9):
            print("prehooked", layer)
            layer.register_forward_pre_hook(save_output)
        i=i+1;
```

```
####################################################

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

```
prehooked QuantConv2d(
  16, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
prehooked QuantConv2d(
  8, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
  (weight_quant): weight_quantize_fn()
)
```

[5]:
```
w_bit = 4
weight_q = model.features[27].weight_q   #quantized weights for 8x8 layer
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)
```

[6]:
```
# To get input activations for 27th layer, it is the 8th prehooked layer in␣
 ↪save_output
x_bit = 4
x = save_output.outputs[0][0]
x_alpha = model.features[27].act_alpha
x_delta = x_alpha/(2**x_bit - 1)
act_quant_fn = act_quantization(x_bit)
x_q = act_quant_fn(x, x_alpha)

x_int = x_q/x_delta
#print(x_int)
```

[7]:
```
# Defining forward pass convolution through the 27th layer
conv_int = torch.nn.Conv2d(in_channels = 16, out_channels = 8, kernel_size = 3,␣
 ↪padding = 1, bias = False)
conv_int.weight = torch.nn.parameter.Parameter(weight_int)
output_int = conv_int(x_int)
output_recovered = output_int * x_delta * w_delta
#print(output_recovered)
```

[ ]:
```
# The reference output is the prehooked input to the 28th layer, or the 9th␣
 ↪prehooked layer
psum_recovered = model.features[28](output_recovered)            # 28th layer␣
 ↪is ReLu operation
```

```
difference = abs(save_output.outputs[1][0] - psum_recovered)
print(difference.mean())
```

[9]:
```
######## For multi-channel PE, x_pad is sized for 16 input channels

######## Padding before Convolution #######
x_pad = torch.zeros(128, 16, 6, 6).cuda()
# a_pad.size() = [64, 32+2pad, 32+2pad]
x_pad[ : ,  :, 1:5, 1:5] = x_int.cuda()
```

[10]:
```
X = x_pad[0]
X = torch.reshape(X, (X.size(0), -1))
print(X.size())

####### Instead of indexing from row7.....row0 we now have row16.......row0

bit_precision = 4
file = open('input.txt', 'w') #write to file
file.write('#time0row15[msb-lsb],time0row6[msb-lst],....,time0row0[msb-lst]#\n')
file.write('#time1row15[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[15-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
```

```
torch.Size([16, 36])
```

[11]:
```
weight_int.size() # 8, 16 , 3, 3
W = torch.reshape(weight_int, (weight_int.size(0), weight_int.size(1), -1))
print(W.size()) # 8, 16, 9

bit_precision = 4
file = open('weight.txt', 'w') #write to file
file.write('#col0row15[msb-lsb],col0row6[msb-lst],....,col0row0[msb-lst]#\n')
file.write('#col1row15[msb-lsb],col1row6[msb-lst],....,col1row0[msb-lst]#\n')
file.write('#................#\n')


W_temp=0
for kij in range(9):
```

```
    for i in range(W.size(0)):  # Column #
        for j in range(W.size(1)): # row #
            #W_bin = '{0:04b}'.format(int(W[i,7-j].item()+0.001))
            if (W[i,15-j,kij].item()<0):
                W_temp=W[i,15-j,kij].item()+(2**bit_precision)
            else:
                W_temp=W[i,15-j,kij].item()
            W_bin = '{0:04b}'.format(int(W_temp+0.001))
            for k in range(bit_precision):
                file.write(str(W_bin[k]))
#             file.write(' ')  # for visibility with blank between words, you␣
 ↪can use
        file.write('\n')
file.close() #close file
```

```
torch.Size([8, 16, 9])
```

[12]:
```
weight_int.size()
W = torch.reshape(weight_int, (weight_int.size(0), weight_int.size(1), -1))
W.size()
W[0,:,0]
```

[12]:
```
tensor([-1.,  0., -2., -2.,  1., -4.,  3.,  3., -3.,  3., -1., -4., -1.,  2.,
         2., -4.], device='cuda:0', grad_fn=<SelectBackward0>)
```

[13]:
```
output_int.size()
O = output_int[0]
O = torch.reshape(O, (O.size(0), -1))
O.size()
```

[13]:
```
torch.Size([8, 16])
```

[14]:
```
### Complete this cell ###

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 i in range(O.size(1)):  # time step
    for j in range(O.size(0)): # Column #
        if (O[7-j,i].item()<0):
            O_temp=O[7-j,i].item()+(2**bit_precision)
        else:
            O_temp=O[7-j,i].item()
        O_bin = '{0:016b}'.format(int(O_temp+0.001))
```

```
#         O_bin = str(int(O_temp+0.001))
        #psum_tile_bin = '{0:016b}'.format(int(psum_tile[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() #close file
```

[21]:
```
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(O.size(1)):  # time step
    for j in range(O.size(0)): # Column #
        if (O[7-j,i].item()<0):
            #O_temp=O[7-j,i].item()+(2**bit_precision)
            O_temp=0
        else:
            O_temp=O[7-j,i].item()
        O_bin = '{0:016b}'.format(int(O_temp+0.001))
#         O_bin = str(int(O_temp+0.001))
        #psum_tile_bin = '{0:016b}'.format(int(psum_tile[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() #close file
```

[ ]:

[ ]:

[ ]:

[ ]: