NVAE_on_MSCOCO

October 12, 2021

[ ]:
```
!wget http://images.cocodataset.org/zips/train2014.zip
!unzip /content/train2014.zip -d /content
!rm /content/train2014.zip
!wget http://images.cocodataset.org/zips/test2014.zip
!unzip /content/test2014.zip -d /content
!rm /content/test2014.zip
!wget http://images.cocodataset.org/zips/val2014.zip
!unzip /content/val2014.zip -d /content
!rm /content/val2014.zip
```

[7]:
```
import os
import torch
from PIL import Image
from os import listdir
from sklearn import preprocessing
```

[9]:
```
class CocoDataloader(object):

    def __init__(self, data_dir, transform=None):
        self.transform = transform
        self.image_names = [os.path.join(data_dir, img) for img in␣
 ↪listdir(data_dir) if os.path.join(data_dir, img)]

    def __len__(self):
        return len(self.image_names)

    def __getitem__(self, idx):

        image = Image.open(self.image_names[idx])

        if self.transform:
            image = self.transform(image)

        return image
```

[12]:
```
coco_train_data = '/content/train2014'
coco_valid_data = '/content/val2014'
```

```
coco_test_data = '/content/test2014'
```

[10]:
```
import os
import time
import numpy
import json
import torch
import logging
import pandas as pd
import matplotlib.pyplot as plt
from torch import nn
from torchvision.utils import make_grid
from torchvision import transforms
from torch.autograd import Variable
from torch.utils.data import DataLoader, ConcatDataset
from torch.utils.tensorboard import SummaryWriter
from torch.optim.lr_scheduler import ExponentialLR
```

[14]:
```
selectedDevice = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Used device: {selectedDevice}")
```

```
Used device: cuda
```

[85]:
```
image_crop = 375
image_size = 64
mean = [0.5, 0.5, 0.5]
std = [0.5, 0.5, 0.5]
batch_size = 256
learning_rate = 0.001
```

[86]:
```
training_data = CocoDataloader(coco_train_data,
                               transform=transforms.Compose([transforms.
↪CenterCrop((image_crop, image_crop)),
                                                             transforms.
↪Resize((image_size, image_size)),
                                                             transforms.
↪RandomHorizontalFlip(),
                                                             transforms.
↪ToTensor(),
                                                            ␣
↪GreyToColor(image_size),
                                                             transforms.
↪Normalize(mean, std)
                                                             ]))
validation_data = CocoDataloader(coco_valid_data,
                                 transform=transforms.Compose([transforms.
↪CenterCrop((image_crop, image_crop)),
```

```
                                                                    transforms.
↪Resize((image_size, image_size)),
                                                                    transforms.
↪ToTensor(),
                                                                   ␣
↪GreyToColor(image_size),
                                                                    transforms.
↪Normalize(mean, std)
                                                                    ]))
test_data = CocoDataloader(coco_test_data,
                           transform=transforms.Compose([transforms.
↪CenterCrop((image_crop, image_crop)),
                                                          transforms.
↪Resize((image_size, image_size)),
                                                          transforms.
↪RandomHorizontalFlip(),
                                                          transforms.ToTensor(),
                                                         ␣
↪GreyToColor(image_size),
                                                          transforms.
↪Normalize(mean, std)
                                                          ]))

train_test_data = ConcatDataset([training_data, test_data])

dataloader_train = DataLoader(train_test_data, batch_size=batch_size,␣
↪shuffle=True, num_workers=2)
dataloader_valid = DataLoader(validation_data, batch_size=batch_size,␣
↪shuffle=False, num_workers=2)
```

[23]:
```
class depthwise_separable_conv(nn.Module):
    def __init__(self, nin, kernels_per_layer, nout):
        super(depthwise_separable_conv, self).__init__()
        self.depthwise = nn.Conv2d(nin, nin * kernels_per_layer, kernel_size=5,␣
↪padding=2, groups=nin)
        self.pointwise = nn.Conv2d(nin * kernels_per_layer, nout, kernel_size=1)

    def forward(self, x):
        out = self.depthwise(x)
        out = self.pointwise(out)
        return out
```

[24]:
```
def swish(x):
    return x * torch.sigmoid(x)
```

[25]:
```
class ChannelSELayer(nn.Module):
    def __init__(self, num_channels, reduction_ratio=2):
        super(ChannelSELayer, self).__init__()
        num_channels_reduced = num_channels // reduction_ratio
        self.reduction_ratio = reduction_ratio
        self.fc1 = nn.Linear(num_channels, num_channels_reduced, bias=True)
        self.fc2 = nn.Linear(num_channels_reduced, num_channels, bias=True)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, input_tensor):
        batch_size, num_channels, H, W = input_tensor.size()
        squeeze_tensor = input_tensor.view(batch_size, num_channels, -1).
 ↪mean(dim=2)

        fc_out_1 = self.relu(self.fc1(squeeze_tensor))
        fc_out_2 = self.sigmoid(self.fc2(fc_out_1))

        a, b = squeeze_tensor.size()
        output_tensor = torch.mul(input_tensor, fc_out_2.view(a, b, 1, 1))
        return output_tensor
```

[26]:
```
class dec_res(nn.Module):
  def __init__(self,in_channel):
    super(dec_res,self).__init__()
    self.bn1 = nn.BatchNorm2d(in_channel)
    self.c1 = nn.
 ↪Conv2d(in_channels=in_channel,out_channels=2*in_channel,kernel_size=1,stride=1,padding=0)
    self.bn2 = nn.BatchNorm2d(2*in_channel)
    self.dc1 =␣
 ↪depthwise_separable_conv(nin=2*in_channel,kernels_per_layer=3,nout=2*in_channel)
    self.bn3 = nn.BatchNorm2d(2*in_channel)
    self.c2 = nn.
 ↪Conv2d(in_channels=2*in_channel,out_channels=in_channel,kernel_size=1,stride=1,padding=0)
    self.bn4 = nn.BatchNorm2d(in_channel)
    self.SE = ChannelSELayer(in_channel)
  def forward(self,x1):
    x = self.c1(self.bn1(x1))
    x = swish(self.bn2(x))
    x = self.dc1(x)
    x = swish(self.bn3(x))
    x = self.bn4(self.c2(x))
    x = self.SE(x)
    return x+x1
```

[27]:
```
class enc_res(nn.Module):
  def __init__(self,in_channel):
    super(enc_res,self).__init__()
    self.bn1 = nn.BatchNorm2d(in_channel)
    self.c1 = nn.
↪Conv2d(in_channels=in_channel,out_channels=2*in_channel,kernel_size=3,stride=1,padding=1)
    self.bn2 = nn.BatchNorm2d(2*in_channel)
    self.c2 = nn.
↪Conv2d(in_channels=2*in_channel,out_channels=in_channel,kernel_size=3,stride=1,padding=1)
    self.bn3 = nn.BatchNorm2d(in_channel)
    self.SE = ChannelSELayer(in_channel)
  def forward(self,x1):
    x = self.c1(swish(self.bn1(x1)))
    x = self.c2(swish(self.bn2(x)))
    x = self.SE(x)
    return x+x1
```

[79]:
```
class NVAE(nn.Module):
  def __init__(self,start_channel,original_dim):
    super(NVAE,self).__init__()
    self.original_dim = original_dim
    self.conv1 = nn.
↪Conv2d(in_channels=start_channel,out_channels=8,kernel_size=3,stride=1,padding=1)
    self.encblock1 = enc_res(8)
    self.dsconv1 = nn.
↪Conv2d(in_channels=8,out_channels=8,kernel_size=2,stride=2,padding=0)
    self.encblock2 = enc_res(8)
    self.dsconv2 = nn.
↪Conv2d(in_channels=8,out_channels=8,kernel_size=2,stride=2,padding=0)

    self.qmu1 = nn.
↪Linear(original_dim*original_dim*2,original_dim*original_dim*2)
    self.qvar1 = nn.
↪Linear(original_dim*original_dim*2,original_dim*original_dim*2)

    self.qmu0 = nn.Linear(original_dim*original_dim//
↪2,original_dim*original_dim//2)
    self.qvar0 = nn.Linear(original_dim*original_dim//
↪2,original_dim*original_dim//2)

    self.pmu1 = nn.
↪Linear(original_dim*original_dim*2,original_dim*original_dim*2)
    self.pvar1 = nn.
↪Linear(original_dim*original_dim*2,original_dim*original_dim*2)

    self.decblock1 = dec_res(8)
```

```
    self.usconv1 = nn.
↪ConvTranspose2d(in_channels=8,out_channels=8,kernel_size=2,stride=2,padding=0)
    self.decblock2 = dec_res(16)
    self.usconv2 = nn.
↪ConvTranspose2d(in_channels=16,out_channels=16,kernel_size=2,stride=2,padding=0)
    self.decblock3 = dec_res(16)
    self.finconv = nn.
↪Conv2d(in_channels=16,out_channels=start_channel,kernel_size=3,stride=1,padding=1)

  def forward(self,x):
    z1 = self.dsconv1(self.encblock1(self.conv1(x)))
    z0 = self.dsconv2(self.encblock2(z1))

    qmu0 = self.qmu0(z0.reshape(z0.shape[0],self.original_dim*self.original_dim/
↪/2))
    qvar0 = self.qvar0(z0.reshape(z0.shape[0],self.original_dim*self.
↪original_dim//2))

    qmu1 = self.qmu1(z1.reshape(z1.shape[0],self.original_dim*self.
↪original_dim*2))
    qvar1 = self.qvar1(z1.reshape(z1.shape[0],self.original_dim*self.
↪original_dim*2))

    stdvar0 = qvar0.mul(0.5).exp_()
    stdvar1 = qvar1.mul(0.5).exp_()

    e0 = torch.randn(qmu0.shape).to(device)
    ez0 = qmu0+e0*stdvar0
    ez0 = ez0.reshape(ez0.shape[0],8,self.original_dim//4,self.original_dim//4)
    ez1 = self.usconv1(self.decblock1(ez0))

    pmu1 = self.pmu1(ez1.reshape(ez1.shape[0],self.original_dim*self.
↪original_dim*2))
    pvar1 = self.pvar1(ez1.reshape(ez1.shape[0],self.original_dim*self.
↪original_dim*2))

    pstdvar1 = pvar1.mul(0.5).exp_()

    e2 = torch.randn(qmu1.shape).to(device)
    ez2 = pmu1+qmu1 + e2*pstdvar1*stdvar1
    ez2 = ez2.reshape(ez2.shape[0],8,self.original_dim//2,self.original_dim//2)

    final = torch.cat((ez1,ez2),1)

    recons = nn.Sigmoid()(self.finconv(self.decblock3(self.usconv2(self.
↪decblock2(final)))))
```

```
    return qmu0,qvar0,qmu1,qvar1,pmu1,pvar1,recons

  def sample(self,bs):
    e = torch.randn([bs,8,self.original_dim//4,self.original_dim//4]).to(device)
    ez1 = self.usconv1(self.decblock1(e))

    pmu1 = self.pmu1(ez1.reshape(ez1.shape[0],self.original_dim*self.
↪original_dim*2))
    pvar1 = self.pvar1(ez1.reshape(ez1.shape[0],self.original_dim*self.
↪original_dim*2))

    stdvar1 = pvar1.mul(0.5).exp_()

    e1 = torch.randn([ez1.shape[0],self.original_dim*self.original_dim*2]).
↪to(device)
    e1 = pmu1 + e1*stdvar1
    e1 = e1.reshape(e1.shape[0],8,self.original_dim//2,self.original_dim//2)
    recons = nn.Sigmoid()(self.finconv(self.decblock3(self.usconv2(self.
↪decblock2(torch.cat((ez1,e1),1))))))

    return recons

  def loss(self,x):
    qmu0,qvar0,qmu1,qvar1,pmu1,pvar1,recons = self.forward(x)
    klz0 = 0.5*torch.sum(torch.square(qmu0)+qvar0.exp()-qvar0-1)/x.shape[0]
    klz1 = 0.5*torch.sum(torch.square(qmu1)/pvar1.exp()+qvar1.exp()-qvar1-1)
    reconsloss = nn.BCELoss()(recons,x)
    return klz0,klz1,reconsloss
```

[89]:
```
model = NVAE(3,64).to(selectedDevice)
```

[90]:
```
optim = torch.optim.Adamax(model.parameters())
device = selectedDevice
```

[91]:
```
epochs=20
```

[83]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
```

[92]:
```
for epoch in range(epochs):
    minloss = 1
    running_kl0_loss=0
    running_recons_loss=0
    running_kl1_loss=0
    num_images=0
    for i, img in enumerate(dataloader_train):
```

```
      img = img.to(selectedDevice)
      optim.zero_grad()
      klz0,klz1,recons = model.loss(img)
      loss=recons+epoch*0.0001*klz0+epoch*0.0001*klz1
      loss.backward()
      optim.step()
      running_kl0_loss = running_kl0_loss + klz0.item()*len(img)
      running_kl1_loss = running_kl1_loss + klz1.item()*len(img)
      running_recons_loss = running_recons_loss + recons.item()*len(img)

      num_images= num_images+len(img)
    print('epoch: '+str(epoch)+' kl0_loss: '+str(running_kl0_loss/num_images)+'␣
↪recons_loss: '+str(running_recons_loss/num_images)+' kl1_loss:␣
↪'+str(running_kl1_loss/num_images))
    imgs = model.sample(64).cpu().detach().reshape(64,28,28)
    plt.gray()
    fig = plt.figure(figsize=(8., 8.))
    grid = ImageGrid(fig, 111, nrows_ncols=(8, 8), axes_pad=0.05)

    for ax, im in zip(grid, imgs):
        ax.imshow(im)
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-92-a501f81f0bd5> in <module>()
     11         loss=recons+epoch*0.0001*klz0+epoch*0.0001*klz1
     12         loss.backward()
---> 13         optim.step()
     14         running_kl0_loss = running_kl0_loss + klz0.item()*len(img)
     15         running_kl1_loss = running_kl1_loss + klz1.item()*len(img)

/usr/local/lib/python3.7/dist-packages/torch/optim/optimizer.py in␣
↪wrapper(*args, **kwargs)
     86                 profile_name = "Optimizer.step#{}.step".format(obj.
↪__class__.__name__)
     87                 with torch.autograd.profiler.
↪record_function(profile_name):
---> 88                     return func(*args, **kwargs)
     89             return wrapper
     90

/usr/local/lib/python3.7/dist-packages/torch/autograd/grad_mode.py in␣
↪decorate_context(*args, **kwargs)
     26         def decorate_context(*args, **kwargs):
     27             with self.__class__():
---> 28                 return func(*args, **kwargs)
     29         return cast(F, decorate_context)
```

```
     30

/usr/local/lib/python3.7/dist-packages/torch/optim/adamax.py in step(self,␣
 ↪closure)
     94                        beta2=beta2,
     95                        lr=lr,
---> 96                        weight_decay=weight_decay)
     97
     98         return loss

/usr/local/lib/python3.7/dist-packages/torch/optim/_functional.py in␣
 ↪adamax(params, grads, exp_avgs, exp_infs, state_steps, eps, beta1, beta2, lr,␣
 ↪weight_decay)
    315         norm_buf = torch.cat([
    316             exp_inf.mul_(beta2).unsqueeze(0),
--> 317             grad.abs().add_(eps).unsqueeze_(0)
    318         ], 0)
    319         torch.amax(norm_buf, 0, keepdim=False, out=exp_inf)

RuntimeError: CUDA error: device-side assert triggered
CUDA kernel errors might be asynchronously reported at some other API call,so␣
 ↪the stacktrace below might be incorrect.
For debugging consider passing CUDA_LAUNCH_BLOCKING=1.
```

[68]:
```
image.size
```

[68]: (640, 425)

[ ]:
```

```